特別研究報告

題目

メトリクス値の変化に基づくコードクローンの編集傾向分析

指導教員
井上 克郎 教授

報告者
東 誠

平成 20 年 2 月 19 日

大阪大学 基礎工学部 情報科学科

平成 19 年度 特別研究報告

メトリクス値の変化に基づくコードクローンの編集傾向分析

東 誠

内容梗概

ソフトウェアシステム内に存在する重複したコード片をコードクローンという．従来，コードクローンはソフトウェアシステムの保守性を悪化させる問題の 1 つとして扱われてきたが，近年の研究によって必ずしもすべてのコードクローンを除去することが効果的なわけではないことが明らかとなった．このことから，ソフトウェアシステム内に存在するすべてのコードクローンは，取り除くのではなく適切に管理する必要がある対象と考えられるようになった．

本研究では，コードクローンの管理を目的として，コードクローンを作成あるいは編集した開発者に着目したコードクローンの編集傾向分析を行う．具体的には，開発者のコードクローンに対する編集傾向の有無を統計的手法を用いて検定し，開発者ごとの編集傾向を調査する．調査対象を取得するために，版管理システムの 1 つである CVS からソフトウェアの編集履歴を取り出し，開発者がどのような変更をコードクローンに対して行ったかを，コードクローンに対するメトリクス値の変化として抽出する．最後に，開発者ごとにメトリクス値の変化を集め，開発者によって変化量に差があるかを分析する．提案する調査方法を用いて，実際のソフトウェア開発履歴を分析した結果，メトリクスの変化は開発者ごとに差があることが分かった．

主な用語

コードクローン
ソフトウェア保守
版管理システム

目 次

1 はじめに

ソフトウェアシステムの保守性を悪化させる問題の 1 つとして，ソフトウェアシステム内に存在する重複したコード片 (コードクローン) がある [34]．例えば，あるコード片に欠陥が存在する場合，そのコード片とコードクローンになっているコード片にも同様の欠陥が存在する可能性が高い．このため，欠陥を修正する際にはすべてのコードクローンに対して修正の必要性を検討する必要があり，保守作業の労力が大きくなることが知られている [15]．この問題に対処するために，コードクローンを効率的に検出し，取り除くための様々な手法やツールが提案されてきた．私が所属する研究室でもコードクローン検出ツール CCFinder[18] を開発してきた．

しかしすべてのコードクローンを除去することが効果的なわけではない [19]．例えば，プラットフォームを変更する際に，プラットフォーム特有のソースコードを書く代わりに，コードクローンを作成して変更を加えることがある．この場合，コードークローンを除去するのではなく，コードクローンの情報を文書化する必要がある．また，保守性に悪影響を与えるが，取り除くことが困難なコードクローンが存在することも判明している [22]．これらのことから，すべてのコードクローンを取り除くのではなく，コードクローンを適切に管理することが必要であると言われている [19, 22] .

本研究では，コードクローンを管理するための基準として，コードクローンを編集した開発者に着目する．開発者はそれぞれ異なった技術を持ち，様々な役割や目的に従ってソフトウェアを変更するため，コードクローンに対する編集にも一定の傾向があるのではないかと考えられる．

そこで本研究では，開発者のコードクローンに対する編集傾向の有無を統計的手法を用いて検定し，開発者ごとの編集傾向を調査する．具体的には，まず，版管理システムからソフトウェアの編集履歴を取り出し，開発者がコードクローンに対して行なった変更を，コードクローンのメトリクス値の変化として抽出する．そして開発者によってメトリクス値の変化に差があるかどうかを調査する．コードクローンを編集した開発者の情報は版管理システムから容易に取得できるため，この調査手法は実際のソフトウェア開発の現場でも簡単に利用できる．また，この調査方法の応用としては，保守性を悪化させるようなメトリクス値の変化を事前に予測することなどが考えられる．

以降，2 章では本研究の背景となるコードクローンと版管理システムについて述べる．3 章では手法の説明として，版管理システムからソフトウェアの編集履歴を取り出し，それを利用して開発者のコードクローンに対する編集傾向の有無を検定する方法について述べる．4 章ではその手法の実装について述べる．5 章では本手法を実際のソフトウェアに対して適用した結果と，それに対する考察を述べる．最後に，6 章で本研究のまとめと今後の課題に

ついて述べる.

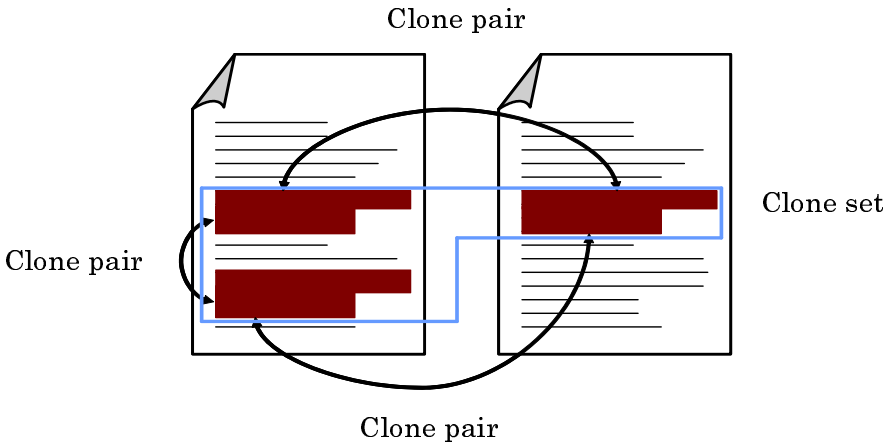

図 1: クローンペアとクローンセット

2　研究の背景

本節では，研究の背景となっている問題について説明する．その問題を説明するために，コードクローンとコードクローン検出ツール CCFinderX，版管理システムについて述べた後，それらにおける既存の関連研究について述べる．

2.1　コードクローン

本節では，コードクローンとコードクローン検出ツール CCFinder，版管理システムについて述べた後，それらにおける既存の関連研究について述べる．

2.1.1　コードクローンと既存のコードクローン検出法

コードクローンとは，ソースコード中に含まれる同一もしくは類似したコードのことであり，いわゆる“ 重複したコード ”のことである．

2 つのコード片 α，β が重複しているとき，α は β のクローンであると定義する (その逆もクローンであるという)．また (α，β) を**クローンペア** (図 1 参照) と呼ぶ．さらに，クローンの同値類を**クローンセット** (図 1 参照) と呼び，ソースコード中でのクローンを特にコードクローンと呼ぶ．

コードクローンがソフトウェアの中に作りこまれる，もしくは発生する原因として次のようなものがある [9, 18].

既存コードのコピーとペーストによる再利用

近年のソフトウェア設計手法を利用すれば，構造化や再利用可能な設計が可能である．

しかし，ゼロからコードを書くよりも既存コードをコピーして部分的な変更を加える方が信頼性が高いということもあり，実際には，コピーとペーストによる場当たり的な既存コードの再利用が多く存在する．

コーディングスタイル

規則的に必要なコードはスタイルとして同じように記述される場合がある．例えば，ユーザインターフェース処理を記述するコードなどである．

定型処理

定義上簡単で頻繁に用いられる処理，例えば，給与税の計算や，キューの挿入処理，データ構造アクセス処理などである．

プログラミング言語に適切な機能の欠如

抽象データ型や，ローカル変数を用いられない場合には，同じようなアルゴリズムを持った処理を繰り返し書かなくてはならないことがある．

パフォーマンス改善

リアルタイムシステムなど時間制約のあるシステムにおいて，インライン展開などの機能が提供されていない場合に，特定のコード片を意図的に繰り返し書くことによってパフォーマンスの改善を図ることがある．

コード生成ツールの生成コード

コード生成ツールにおいて，類似した処理を目的としたコードの生成には，識別子名等の違いはあろうとも，あらかじめ決められたコードをベースにして自動的に生成されるため，類似したコードが生成される．

偶然

単純に偶然一致してしまう場合もあるが，大きなコードクローンになる可能性は低い．

もしコードクローンが存在した場合には，一般的にコードの変更等が困難であるといわれ，保守容易性の低下の一因となっている．このようなコードクローンによる問題に対処する方法としては，

- コードクローン情報の文書化を行うことで変更の一貫性を保つ
- コードクローンを自動で検出する

の二つがある [37]．しかし，コードクローン情報の文書化にはすべてのコードクローンに対する情報を常に最新に保つことに非常に手間がかかるため，現実的に困難である．そこで，

これまでに Covet や CloneDR 等，様々なコードクローン検出手法やツールが提案されている [9, 28].

いずれの手法，ツールにおいても提案者によってコードクローンの定義が微妙に異なっており，検出されるコードクローンが異なっている．つまり，コードクローンの定義とは検出アルゴリズムそのものによって定義される．Burd ら [11] も，私が所属する研究室で開発された CCFinder を含めた五つのツールを用いて，それぞれ検出されるコードクローンの比較を行っているが，すべての面において他のツールより優れているツールはなく，使う場面に応じて，適切なツールを選ぶことが必要となってくると述べている．

また，近年の研究により，必ずしもすべてのコードクローンが生産性を悪化させるわけではないことが明らかになっている [19]．そのため，コードクローンを検出するだけではなく，どのコードクローンを優先的に対処すべきかを提示したり，コードクローンを適切に管理することが必要になる．

CCFinder においては，あるトークン列中に存在する 2 つの部分トークン列 α，β が等価であるとき，α は β のクローンであると定義する (その逆もクローンであるという)．また，α，β それぞれを真に包含するどのようなトークン列も等価でないとき α，β を極大クローンという．

CCFinder は，単一または複数のファイルのソースコード中からすべての極大クローン検出し，それをクローンペアの位置情報として出力する．

2.1.2 コードクローン検出ツール CCFinder

CCFinder のコードクローン検出処理は，以下の 4 ステップで構成されている．

ステップ 1: 字句解析

ソースコードをプログラミング言語の文法に沿ってトークン列に変換する．その際，空白とコメントは機能に影響しないので無視される．ファイルが複数の場合には，単一ファイルの解析と同じように処理できるよう，単一のトークン列に連結する．

ステップ 2: 変換処理

実用的に意味を持たないコードクローンを取り除くこと，および，ある程度の違いを吸収することを目的とした変換ルールによりトークン列を変換する．例えば，変数名，関数名などはすべて同一のユニークなトークンに置換される．

ステップ 3: 検出処理

トークン列の中から指定された長さ以上一致している部分をクローンペアとしてすべて検出する．

ステップ 4: 出力整形処理

検出されたクローンペアについて，元のソースコード上での位置情報を出力する．

2.1.3 コードクローン検出ツール CCFinderX

CCFinderX は神谷氏が CCFinder を再設計したものである [12] ．CCFinder に比べ，以下の点において性能か向上している．

- 計算の高速化およびスケーラビリティの向上
- 新しいプログラミング言語への対応
  - 現在は C，C++，Java，COBOL，VB，C# に対応
- 高度な分析機能の追加
- より正確な検出

特に，新しく追加された分析機能として，コードクローンを検出する際にクローンセットのメトリクスを計算して表示することができる．

また，CCFinderX では 2.1.2 節における字句解析結果をファイルに格納している．このファイルのことををを**プリプロセスファイル**と呼ぶ．

プリプロセスファイルの形式は以下のようになっている．なお，トークンである文字列のソースファイルにおける通し番号のことを**トークン番号**と呼ぶことにする．

- 1 行につき 1 トークンについての情報が書かれている
- トークンが 1 行に収まる場合は，そのトークンについての情報として以下のものが書かれる
  - トークンを表す文字列
  - トークンが存在する行の番号
  - トークンが存在する列の番号
  - トークンである文字列のトークン番号
  - トークンの長さ
- トークンが複数行に跨っている場合は，そのトークンについての情報は以下の形式で書かれる

- トークンを表す文字列
- トークンの開始位置が存在する行の番号
- トークンの開始位置が存在する列の番号
- トークンの開始位置である文字列のトークン番号
- トークンの終了位置が存在する行の番号
- トークンの終了位置が存在する列の番号
- トークンの終了位置である文字列のトークン番号

また，CCFinderX はコードクローンを検出せずにプリプロセスファイルを作成することも可能である．

2.1.4 CCFinderX による検出例

実際に，CCFinderX がどのようなコードクローンを検出するのか例を示す．図 2 は説明のための Java ソースコードである．このソースコードには，互いに似通った 2 つのメソッドが含まれ，左端には行番号が付されている．ここで，最小一致トークン数を 5 トークンに定め，図 2 のソースコードに対しコードクローン検出を行うと，図 2 中の A1(4 行目-6 行目) と A2(16 行目-17 行目)，B1(8 行目-10 行目) と B2(20 行目-22 行目)，そして C1(12 行目) と C2(25 行目) がそれぞれクローンペアとして検出される．それぞれのクローンペアの長さは順に 7,18,6 トークンとなっている．見ての通り，A1 と A2 の間，B1 と B2 の間には次のような幾らかの違いが含まれているがコードクローンとして検出可能となっている．

- 名前空間の違い (e.g.“ org.apache.regexp.RE ”と“ RE ”)
- 変数名の違い (e.g.“ pat ”と“ exp ”)
- 改行とインデントの違い

2.2 版管理システム

本節では，まず，版管理システムとその具体例について説明した後，版管理システムを用いて抽出することのできるコミットトランザクションとスナップショットについて説明する．

2.2.1 概要

版管理とは，主として以下の 3 つの役割を提供する機構である．

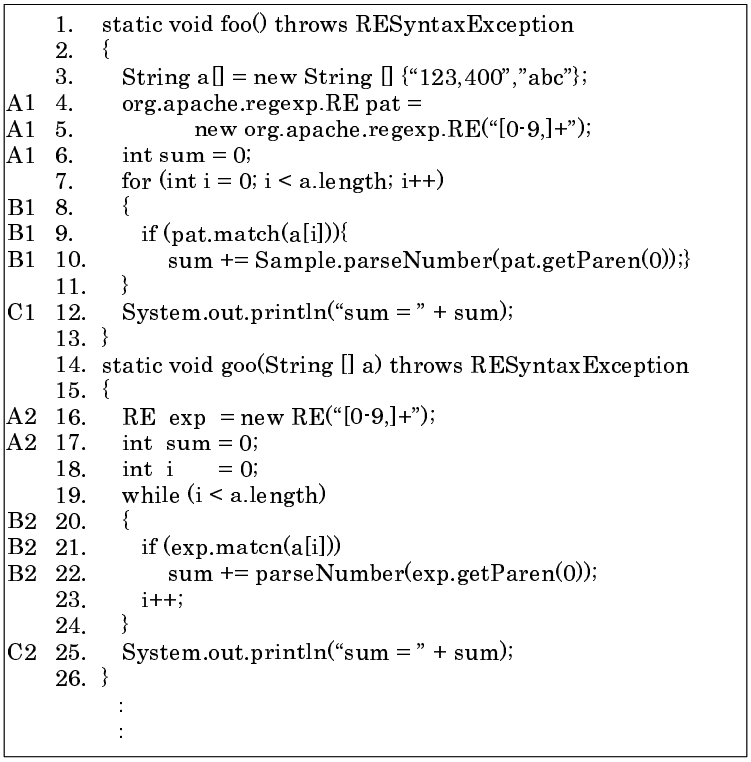

図 2: コードクローン検出例

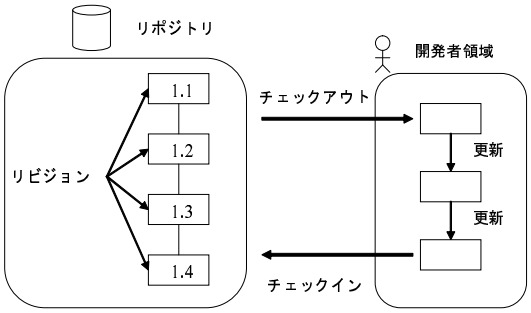

図 3: Checkout/Checkin Model

- プロダクトに対して施された追加・削除・変更などの作業を履歴として蓄積する.
- 蓄積した履歴を開発者に提供する.
- 蓄積したデータを編集する.

各プロダクト (ソースコード，リソースなど) の履歴データは，**リポジトリ (Repository)** と呼ばれるデータ格納庫に蓄積される．その内部では，プロダクトのある時点における状態である**リビジョン (Revision)** を単位として管理する．１つのリビジョンには，ソースコードやリソースなどの実データと，作成日時やメッセージログなどの属性データが格納されている.

また，リポジトリとのデータ授受をする為に，開発者はシステムに依存したオペレーション (operation) を利用する必要がある.

版管理手法を述べるにあたり，その基礎となるモデルが数多く存在する [33, 38] ．本節では，多くの版管理システムが採用している Checkout/Checkin モデルについて概要を述べる．なお，以降本文において，プロダクトのある時点における状態のことをリビジョンと呼ぶことにする.

Checkout/Checkin モデルは，ファイルを単位としたリビジョン制御に関して定義されている (図 3 参照).

リビジョン管理下にあるコンポーネントはシステムに依存したフォーマット形式のファイルとしてリポジトリに格納されている．開発者のそれらのファイルを直接操作するのでなく，各システムに実装されているオペレーションを介して，リポジトリとのデータ授受を行う．リポジトリより特定のリビジョンのコンポーネントを取得する操作を**チェックアウト (Checkout)** という．逆に，データをリビジョンに格納し，新たなリビジョンを作成する操作を**チェックイン (Checkin)** という.

2.2.2 既存の版管理システムの紹介

版管理システムと呼ばれるものは，多数存在する．UNIX 系 OS では，多くの場合，RCS[40] や CVS[10] といったシステムが標準で利用可能となっている．ClearCase[17] のように商用ものも存在する．また，UNIX 系 OS だけではなく，Windows 系 OS においても，SourceSafe[39] や PVCS[32] をはじめ，数多く存在する．さらに，ローカルネットワーク内のみではなく，よりグローバルなネットワークを介したシステム [14] も存在する．

ここでは，版管理システムのうちのいくつかを紹介する．

- RCS

RCS[40] は UNIX 上で動作するツールとして作成された版管理システムであり，現在でもよく使用されているシステムである．単体で使用される他，システム内部に組み込み，版管理機構を持たせる場合などの用途もある．RCS ではプロダクトをそれぞれ UNIX 上のファイルとして扱い，１ファイルに対する記録は１つのファイルに行われる．

RCS におけるリビジョンは，管理対象となるファイルの中身がそれ自身によって定義され，リビジョン間の差分は diff コマンドの出力として定義される．各リビジョンに対する識別子は数字の組で表記され，数え上げ可能な識別子である．新規リビジョンの登録や，任意のリビジョンの取り出しは，RCS の持つツールを利用する．

- CVS

CVS[10] は RCS 同様，UNIX 上で動作するシステムとして構築された版管理システムであり，近年最も良く使われるシステムの１つである．RCS と大きく異なるのは，複数のファイルを処理する点である．変更した複数のファイルを一度のコミット作業でリポジトリに登録できる点である．

，この時の一群のファイルに対する変更をまとめて**コミットトランザクション**と呼ぶ．なお，CVS のリポジトリに格納されている情報を利用して，コミットトランザクションの直前または直後の時点におけるソースコード (以下，**スナップショット**とする) を取得することも可能である．

また，リポジトリを複数の開発者で利用することも考慮し，開発者間の競合にも対処可能となっている．さらに，ネットワーク環境 (ssh， rsh 等) を利用することも可能である為，オープンソースによるソフトウェア開発やグループウェアの場面で活躍の場が多い．その最たる例が，FreeBSD[36] や OpenBSD[29] 等のオペレーティングシステムの開発である．

2.2.3 コミットトランザクションとスナップショット

CVS のように，版管理システムの中には変更した複数のファイルを一度のコミット作業でリポジトリに登録できるものが存在する．この時の一群のファイルに対する変更をまとめて**コミットトランザクション**と呼ぶ．

また，版管理システムのリポジトリに格納されている情報を利用して，コミットトランザクションの直前または直後の時点におけるソースコードを取得することも可能である．このソースコードを**スナップショット**と呼ぶ．

2.3 関連研究

本節では本研究の背景になった過去の研究について述べる．

2.3.1 除去が効果的でないコードクローン

除去が効果的でないコードクローンの研究として，Kapser らの研究 [19] が挙げられる．この研究において Kapser らは，開発者がコードクローンを作成するパターンを 8 つ列挙し，各パターンに該当するコードクローンの管理方法を提案している．挙げられているパターンは以下の通りである．

Hardware variations.

ハードウェアのために新しいドライバを作る時，それと類似したハードウェアが既にドライバを持っていることがある．その場合，これら 2 つのドライバのソースコードはコードクローンとなるが，このコードクローンを修正するのは困難かつ危険である．

このパターンに該当するコードクローンの管理方法は，コードクローンとなっているドライバを含むクローンセットを特定し，そのクローンセット内の潜在的なバグを調査することである．

Platform variation.

ソフトウェアを新しいプラットフォームに導入する時，プラットフォームとやり取りをする関数は修正する必要がある．この時，どんなプラットフォームにも対応する関数のソースコードを書くより，関数のコードクローンを作成して修正する方が修正時間が短く，実行速度も速い上に安全である．

ただし，この方法によって作成される関数は，同じソフトウェアの要求とプラットフォームの変更という 2 つの要因によって修正されていく．そのため，同じクローンセットに含まれるコードクローンになっている関数のうち，どの関数にバグがあるのかわからなくなり，すべてのコードクローンのバグを修正するのが困難になる．

このパターンに該当するコードクローンの管理方法は，そのコードクローンについての情報を文書化し，すべてのコードクローンに対して同じ修正ができるようにしておくことである．

Experimental variation.

ソースコードを修正したり最適化したりする際に，テストベッドとしてコードクローンを作成し，そのコードクローンに対して修正や最適化を行う．そして修正されたコードクローンの動作が安定したら，基のソースコードとコードクローンを統合する．

このパターンにおいては，基のソースコードとコードクローンを統合する際に問題が発生する．基のソースコードがコードクローンとは独立に修正されすぎた場合，基のソースコードとコードクローンを統合するのが難しくなる．

このパターンに該当するコードクローンの管理方法は，基のソースコードとコードクローンをできるだけ近い状態に維持することである．

Boiler-plating due to language in-expressiveness.

プログラミング言語の制約上，必然的にコードクローンが発生することがある。特に，COBOL を用いて開発されたソフトウェアシステム内では共通してこのコードクローンが発生する．

このパターンに該当するコードクローンの管理方法は，すべてのクローンセットについての情報を文書化することである．

API/Library protocols.

特定の API やライブラリを利用することにより，それを用いて実装したソースコードがコードクローンとなる．

このパターンに該当するコードクローンの管理方法は，適切な抽象化を用いてそのコードクローンに用いられている API やライブラリを拡張することである．また，これらのコードクローンを厳格に評価し，安全であることを保証する必要がある．

General language or algorithmic idioms.

プログラミングの世界では一種のイディオムとなっているソースコードの書き方が存在する．これらのイディオムに従って書かれたソースコードはコードクローンになりやすい．

このパターンのコードクローンが問題になるのは，イディオムの使い方に一貫性がなかったり，イディオムを誤って使っていたりする場合である．また，これらのイディオムには効果的でないものもある．

このパターンに該当するコードクローンの管理方法は，効果的でないイディオムに従って書かれているコードクローンを除去し，それ以外のイディオムに従って書かれているコードクローンを修正して一貫したイディオムの使い方になるようにすることである．

Bug workarounds.

あるコード片にバグがある場合に，そこにバグのないコード片をコピー&ペーストすることによってバグを修正する．ただしこの方法による解決は一時的なものであり，本質的なバグの解決ではない．

このパターンに該当するコードクローンの管理方法は，バグが解決できたら，このパターンに該当するコードクローンすべてを除去することである．

Replicate and specialize.

問題を解決するソースコードを書く際に，似たような問題を解決したソースコードを探し，そのコードクローンを作成して修正することによって解決する．この方法はソースコードのテストや修正の労力を一時的に軽減させるが，長期的には労力を増加させる．

このパターンに該当するコードクローンの管理方法は，コードクローンに適切な抽象化を行うことである．もし適切な抽象化ができないのなら，文書やツールを用いてそのコードクローンについての情報を記録する．

2.3.2 コードクローンの作成または編集，削除に関わった開発者の特定手法

コードクローンの作成または編集，削除に関わった開発者を特定する研究として，川口らの研究 [21] が挙げられる．この研究において川口らは，版管理システムから取り出したソースコードの編集履歴を用いて，各コードクローンを作成または編集，削除した開発者を特定する手法を提案している．なお，版管理システムとして CVS，コードクローン検出ツールとして CCFinder を用いている．

コードクローンを作成または編集，削除した開発者を特定するには，まず，版管理システムからソースコードの編集履歴を取り出す．ソースコードの編集履歴からは以下の情報が得られる．

- リビジョンの一覧
- 各リビジョンの情報
  - リビジョン番号
  - 編集日時

– 編集を行った開発者
– コミットログ

次に，編集履歴から得られたリビジョンの集合に対し，各リビジョン間における編集差分を逐次的に取得する．リビジョン間の編集差分は「編集が行われた領域」と「編集内容」の繰り返しとなっている．このうち，編集されたコードクローンの特定に用いる「編集が行われた領域」の情報は，以下の 3 つの情報から構成される．

- 編集元の領域．編集元の開始位置と終了位置
- 編集操作．追加，削除，編集の 3 種類
- 編集先の領域．編集先の開始位置と終了位置

この編集差分の中に，編集日時に存在するコードクローンを編集しているものがあれば，「編集を行った開発者」をそのコードクローンの編集者とする．開発者がコードクローンを編集しているかどうかを調べるには，開発者が編集した領域の中にコードクローンが存在するかどうかを調べる．もしその領域の中にコードクローンが存在した場合，そのコードクローンはこの編集によって何らかの変更をされていると考えられる．

2.4 問題

本節ではコードクローンを管理する必要性と，開発者の編集傾向を用いたコードクローン管理を本研究の問題とする理由について説明する．

2.4.1 コードクローン管理の必要性

2.3.1 項のように，除去以外の方法で対処すべきコードクローンが存在する．特に，Platform variation. は JIS 規格 [35] で定められているソフトウェア保守のうち，利用環境の変化にソフトウェアを適応させる適応保守を行う際に有益なコードクローンである．また，ソフトウェアの保守性に悪影響を与えるにも関わらず，取り除くことが困難なコードクローンも存在する [22].

これらのコードクローンに対処する方法の 1 つとして，コードクローンを管理することが考えられる．コードクローン管理の例としては，保守性を悪化させるコードクローンは除去し，開発者が設計や保守を行い易くするために作成したコードクローンは除去しない，といったものが考えられる．

2.4.2 開発者の編集傾向を用いたコードクローン管理

本研究では，コードクローン管理に用いる情報として，開発者の情報に着目する．

開発者の情報を用いる理由は，各開発者の役割や目的，経験には違いがあるからである．例えば，開発者はそれぞれ異なった技術を持ち，様々な役割や目的に従ってソフトウェアを変更する．また，熟練した開発者とそうでない開発者とでは，その編集内容が異なると考えられる．

これらの違いによって，開発者がコードクローンを編集する際に，コードクローンに特徴が表れると考えられる．例えば，経験の浅い開発者によって編集されたコードクローンは，保守性を悪化させる可能性が高いと考えられる．この特徴のことを**編集傾向**と呼ぶことにする．

そして，各開発者の編集傾向によってコードクローンを区別し，管理できるのではないかと考えられる．このような管理の例としては，コードクローンの保守性を悪化させるという編集傾向を持つ開発者の情報を用いたコードクローン管理が挙げられる．まず，上記の開発者によって編集された回数が多いコードクローンと，そうでないコードクローンを区別する．その開発者に編集された回数が多いコードクローンは将来ソフトウェアの保守性を悪化させる可能性が高いので除去する．その開発者に編集されていないコードクローンは，それらの情報を文書化しておく．後にそれらのコードクローンの 1 つにバグが発見された時，すべてのコードクローンに対して同じ修正を行う．

本研究では編集傾向を知るために，コードクローンの性質を表すメトリクス値の変化を調査する．メトリクス値の変化によって開発者がコードクローンにどのような編集を行なったかを知ることができるため，これを調査すれば開発者の編集傾向を知ることができる．

3　開発者ごとの編集傾向の調査手法

本節では，2.4.2 項で述べた問題を解決するための，開発者ごとの編集傾向の調査手法について述べる．

コードクローンの性質を表すメトリクス値の変化を調査するには，まず，過去のソースコードに存在するコードクローンを検出する必要がある．過去のソースコードの復元には CVS のリポジトリ，コードクローンの検出には CCFinderX をそれぞれ用いる．

検出したコードクローンの性質を表すメトリクス値の変化を取得するために，本手法ではソースコード中の同じ位置にあるコードクローン 2 つを特定し，それらのメトリクス値の差を計算する．この 2 つのコードクローンは，それらの間に成り立つ関係を抽出することによって特定する．

各開発者によるメトリクス値の変化の調査には，統計的な手法とメトリクス値の変化の分布を用いる．まず，開発者ごとにメトリクス値の変化を集め，開発者間に有意差があるかを統計的な手法で検定する．次に，その検定結果とメトリクス値の変化の分布を用いて，各開発者によるメトリクス値の変化を特定する．

上記の調査手法の概要を図 4 に示す．図 4 のように，調査手法は大きく分けて以下の 5 つのステップからなる．

1. リポジトリからコミットトランザクション情報を抽出し，その直前および直後の時点におけるスナップショットを取得する
2. 各スナップショットにおいてコードクローンを検出する
3. スナップショット間のコードクローンの対応関係を抽出する
4. コードクローンの対応関係から，検定に用いるためのコードクローンセットメトリクスの値の変化の差を取得する
5. メトリクス値の変化の差を検定することにより，開発者のコードクローンに対する編集傾向の有無を判断する

以降の節で，各ステップの詳細を説明する．

3.1　コミットトランザクション情報の抽出

まず，図 4 の (1) のように CVS のリポジトリの履歴からコミットトランザクションの情報を抽出する．各コミットトランザクションの情報は以下の情報から成り立つ．

- 編集されたファイルのパス名

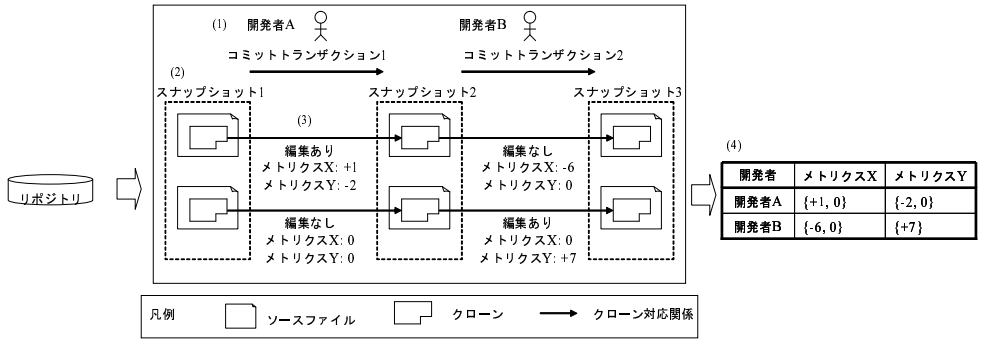

図 4: 検定手法の概要

- 編集されたファイルの変更差分
- 編集後のリビジョン番号
- 編集後のファイルの状態
- コミットした開発者
- コミットされた日時
- コミットログ

コミットトランザクションはコミットされた日時の早い順に並べ，1 から順に通し番号を振る．これにより，各開発者がソースコードを編集した順番がわかるようになる．

そして，図 4 の (2) のように以下の時点におけるスナップショットを取得し，日時の早い順に通し番号を振る．

- コミットトランザクション 1 の直前
- 各コミットトランザクションの直後

3.2 スナップショットにおけるコードクローンの検出

次に，3.1 節で取得したスナップショットにおけるコードクローンを検出する．コードクローンの検出には CCFinderX を利用する．

また，コードクローン検出の際に，CCFinderX によって各コードクローンのコードクローンセットメトリクス (LEN, RNR, TKS, LOOP, COND) を 計算する．以下に各メトリクスの説明をする．

LEN(S)

クローンセット S に含まれるコードクローンの平均トークン数を表す．この値が大きいほどコードクローンのサイズが大きいことになる．

RNR(S)

クローンセット S に含まれるコードクローン内の重複した処理の少なさの度合いの平均値を表す．コードクローン中の総トークン数を $Tokens_{all}$，コードクローン中の繰り返し部分のトークン数を $Tokens_{repeated}$ とすると，RNR は式 (1) で表される．

$$RNR = 1 - \frac{\sum_{C \in S} Tokens_{repeated}(C)}{\sum_{C \in S} Tokens_{all}(C)} \tag{1}$$

TKS(S)

クローンセット S に含まれるコードクローンにおけるトークンの種類数の平均値を表す．TKS が極端に小さなクローンセットのコードクローンは，変数の宣言の連続などである可能性が高い．

LOOP(S),COND(S),McCabe(S)

LOOP(S) はクローンセット S に含まれるコードクローンにおけるループの数の平均値を表す．COND(S) はクローンセット S に含まれるコードクローンにおける条件分岐の平均値を表す．McCabe(S) は LOOP(S) と COND(S) の合計値を表す．これらの値が大きいほど，複雑なコード片がコードクローンになっていることを表している．

3.3 コードクローンの対応関係の抽出

図 4 の (3) のように，スナップショット間のコードクローンの対応関係を抽出する．

本研究においては，t 番目のスナップショット S_t に存在するコードクローン C_c と，t+1 番目のスナップショット S_{t+1} に存在するコードクローン C_d の関係 $CR(C_c, C_d)$ は，以下の条件をすべて満たす時のみ真とする．また，$CR(C_c, C_d)$ が真となるとき，C_c は C_d 間において**クローン対応関係**があるとする．

- $C(S)$:スナップショット S に存在するすべてのコードクローンの集合
- $Sim(a, b)$:コード片 a,b の類似度．詳細は 3.3.1 項にて説明する
- $N(c, d)$: コード片 d に対するコード片 c の対応の真偽．計算方法は以下の通りである

- $N(c,d) = Sim(c,d) > 0 \wedge (\forall a \in C(S) Sim(c,a) \leq Sim(c,d))$

- $M(a,S)$:コードクローン a に対応する，スナップショット S に存在するコードクローンの集合．計算方法は以下の通りである
  - $M(a,S) = \{b \mid N(a,b) \vee N(b,a), b \in C(S)\}$

- $CR(C_c, C_d) = C_d \in M(C_c, S_{t+1})$

3.3.1 コード片の類似度 Sim の計算方法

ソースファイル F_t に存在するコード片 a と，ソースファイル F_{t+1} に存在するコード片 b の類似度 $Sim(a,b)$ の計算方法は，大きく分けて以下の 5 つのステップからなる．

1. コード片 a,b が存在する各ソースファイルのパス名が同じかどうかを確認する
2. コード片 a,b が存在する各ソースファイルをトークン列に変換する
3. トークン列に変換されたソースファイルの変更差分を取得する
4. 変更差分の情報を用いて，ソースファイルの各トークンの対応関係を求める
5. トークンの対応関係を用いて，$Sim(a,b)$ の値を計算する

以下，各ステップの詳細について説明する．

ステップ 1: ソースファイルのパス名の確認

F_t, F_{t+1} のパス名が同じかどうかを確認する．パス名が異なっていれば $Sim(a,b) = 0$ とし，以降のステップには進まない．パス名が同じであればステップ 2 へ進む．

ステップ 2: ソースファイルのトークン列への変換

2.1.2 節で述べた CCFinderX の字句解析機能を用いて，F_t, F_{t+1} をトークン列に変換する．具体的には，F_t, F_{t+1} に対するプリプロセスファイルを作成し，プリプロセスファイル中のトークンに関する情報から，トークンを表す文字列以外の情報すべてを削除する．こうすることにより，1 行につき 1 トークンだけが存在するファイルを作成することができる．

以下，F_t, F_{t+1} をトークン列に変換したファイルをそれぞれ P_t, P_{t+1} と呼ぶことにする．また，各トークンは，ファイルのパス名と行番号によって一意に識別される．

ステップ 3: ソースファイルの変更差分の取得

P_t, P_{t+1} の差分情報を取得する．これらのファイルにおいては 1 行につき 1 トークンが存在するので，P_t, P_{t+1} の差分情報における行番号が F_t, F_{t+1} におけるトークン番号になる．これにより，何番目のトークンが変更されたのかを調べることができる．

ステップ 4: トークンの対応関係の抽出

P_t, P_{t+1} 間のトークンの対応関係を抽出する．

P_t に存在するトークン T_c と，P_{t+1} に存在するトークン T_d の対応関係 $TR(T_c, T_d)$ は，以下の条件をすべて満たす時に真とする．

- T_c が差分情報における P_t の変更箇所に含まれていない
- T_d が差分情報における P_{t+1} の変更箇所に含まれていない
- T_c の P_t における行番号を，差分情報によって P_{t+1} 上の行番号に調整 [20, 21] した値が，T_d の P_{t+1} における行番号に一致する．

ステップ 5: $Sim(a, b)$ の計算

$TR(T_c, T_d)$ の情報を用いて $Sim(a, b)$ を計算する．$Sim(a, b)$ は以下の式によって求められる．

- $T(C)$:コード片 C に存在するすべてのトークンの集合
- $TokPair(a, b)$:$\{(c, d) \mid TR(c, d), c \in T(a), d \in T(b)\}$
- $TokSum(C, D)$:コード片 C のトークンのうちコード片 D に対応するトークンのあるトークンの数．計算方法は以下の通りである
  - $TokSum(C, D) = TokPair(C, D)$ の要素数
- $Sim(a, b) = TokSum(a, b) \times 2 \div (a$ のトークン数 $+ b$ のトークン数$)$

3.4 検定用データの取得

図 4 の (4) のように，開発者ごとのメトリクス値の変化を調査するための検定用データを取得する．検定用データは開発者を行，メトリクスを列とする表である．行が開発者 A であり，列がメトリクス X である升目の要素は，開発者 A のすべてのコミットトランザクションにおいてクローン対応関係にあるコードクローン C_c,C_d のメトリクス X の値の差のうち，以下のいずれかの条件を満たすもののリストとする．

なお，開発者 A によって編集されたコードクローンとは，開発者 A のコミットトランザクションの直前の時点のスナップショットに存在し，コミットトランザクションにおいて編

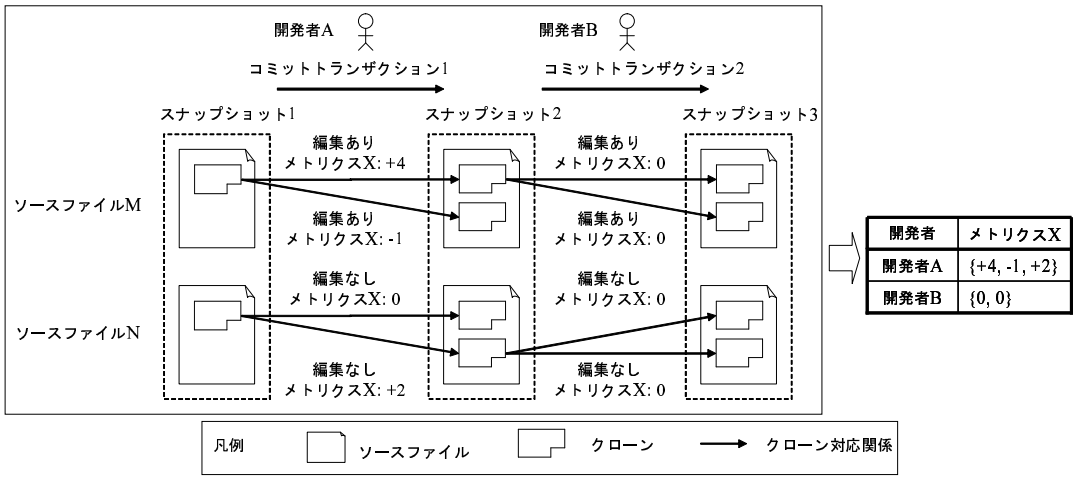

図 5: 検定用データの例

集されたファイルの変更差分と重複する箇所に存在するコードクローンを指す．これは 2.3.2 節のように，CVS の差分情報とコードクローンの開始位置，終了位置を用いて特定する．

- C_c が開発者 A によって編集されたコードクローンである
- C_c と C_d のメトリクス X の値の差が 0 でない

検定用データの例を図 5 に示す．

図 5 において，開発者 A のコミットトランザクションではソースファイル M のコードクローンが編集されているため，メトリクス X の値の差はすべて検定用データの要素に含める．また，ソースファイル N のコードクローンは編集されてないので，メトリクス X の値の差のうち，+2 だけを検定用データの要素に含める．

一方，開発者 B のコミットトランザクションではすべてのメトリクス X の値の差が 0 だが，ソースファイル M のコードクローンは編集されているので，そのメトリクス X の値の差は検定用データの要素に含める．

3.5 メトリクス値の変化の調査

3.4 節で取得した検定用データのそれぞれのメトリクスについて，Kruskal-Wallis 検定 [25] を行う．この検定は，メトリクス値の差の分散における開発者間の有意差の有無を検定する．そのため，この検定において有意差があると判断されれば，少なくとも 1 人の開発者は他の開発者と有意差があるといえる．

また，有意差があると判断されたメトリクスの列の検定用データに対し，各開発者と他の開発者間の有意差の有無を検定する．開発者 A とメトリクス X に対するこの検定は以下の手順で行う．

1. 検定用データの要素のうち，メトリクス X の列にあるものを，開発者 A の群とそれ以外の開発者の群に分けて集める
2. 1 で作成した 2 群に対し，Mann-Whitney の U 検定 [27] (以下，単に U 検定とする) を行う

また，列がメトリクス X の検定用データの要素に対し，開発者 A と他の開発者間に有意差があった場合，以下のようにして開発者 A がメトリクス X の値をどのように変化させるかを調べる．

1. 検定用データの要素のうち，メトリクス X の列にあるものすべてを 1 つの群にまとめる
2. 1 で作成した群の要素を，値の小さい順から順位付けする．同順位があった場合には，平均順位を求める
3. 各値につけた順位を，開発者 A の群とそれ以外の開発者の群に分けて集める
4. 開発者 A の群の平均値 AVE_A, それ以外の開発者の群の平均値 AVE_{other} をそれぞれ求める
5. AVE_A,AVE_{other} の大小関係により，開発者 A によるメトリクス X の値の変化を以下の 3 種類に分類する
    - $AVE_A > AVE_{other}$ となる場合
        - 開発者 A は他の開発者に比べて，メトリクス X の値を有意に増加させている
    - $AVE_A < AVE_{other}$ となる場合
        - 開発者 A は他の開発者に比べ，メトリクス X の値を有意に減少させている
    - $AVE_A == AVE_{other}$ となる場合
        - 開発者 A は他の開発者と有意差があるとはいえない

4　調査手法の実装

本節では，3 節で説明した開発者ごとの編集傾向の調査手法を実装したシステムについて述べる．

実装したシステム全体の概要を図 6 に示す．図 6 のように，システムは大きく分けて 4 つに分けられる．各サブシステムの機能の概略は以下のようになっている．

- コードクローン検出部
  - 3.1 節で述べたコミットトランザクションの抽出
  - 3.2 節で述べたスナップショットにおけるコードクローン検出
- クローン対応関係抽出部
  - 3.3 節で述べたコードクローンの対応関係の抽出
- 検定用データ作成部
  - 3.4 節で述べた検定用データの取得
- 編集傾向調査部
  - 3.5 節で述べたメトリクス値の差の検定

以下の節で，システムの各部分の詳細を説明する．

4.1　コードクローン検出部

コードクローン検出部の入出力は以下のようになっている．

入力　CVS リポジトリ，取得開始日，取得終了日

CVS リポジトリはコミットトランザクションやスナップショットの情報を取得するために用いる．

取得開始日，取得終了日はコミットトランザクションを抽出する期間を指定する．取得開始日，取得終了日の形式を BNF 記法で表すと以下のようになる．

- 取得開始日 ::= 日付
- 取得終了日 ::= 日付
- 日付 ::= 年 ’/’ 月 ’/’ 日

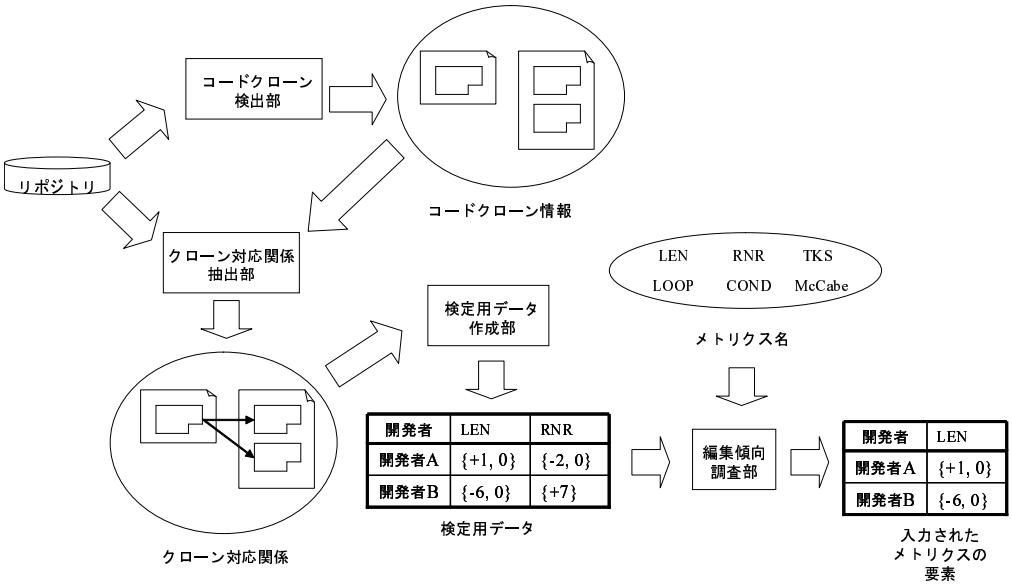

図 6: 実装したシステムの概要

- 年 ::= 数字 数字 数字 数字
- 月 ::= 数字 数字
- 日 ::= 数字 数字
- 数字 ::= ’0’ | ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’ | ’8’ | ’9’

取得開始日，取得終了日は必ずしも入力する必要はない．取得開始日，取得終了日の入力と，コミットトランザクションの抽出期間の関係は表 1 のようになっている．

出力 各スナップショットにおけるコードクローンの情報 (テキストファイル)
出力ファイルの形式を ER 図で表すと図 7 のようになる．また，出力ファイルの具体例を図 8 に示す.

表 1: 取得開始日，取得終了日の入力と利用する変更情報の関係

取得開始日	取得終了日	コミットトランザクションの抽出期間
有	有	取得開始日～取得終了日の 1 日前
有	無	取得開始日以降 (取得開始日を含む)
無	有	取得終了日以前 (取得終了日を含まない)
無	無	すべての期間

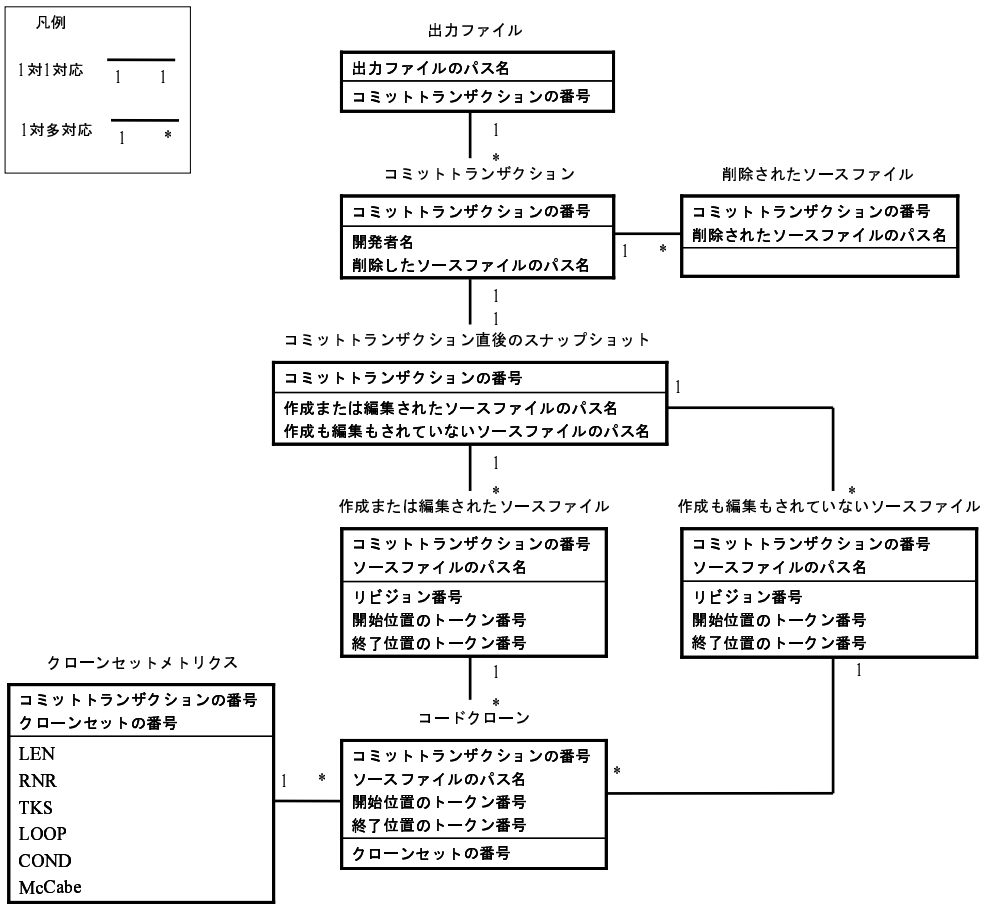

図 7: コードクローン検出部の出力ファイルの形式

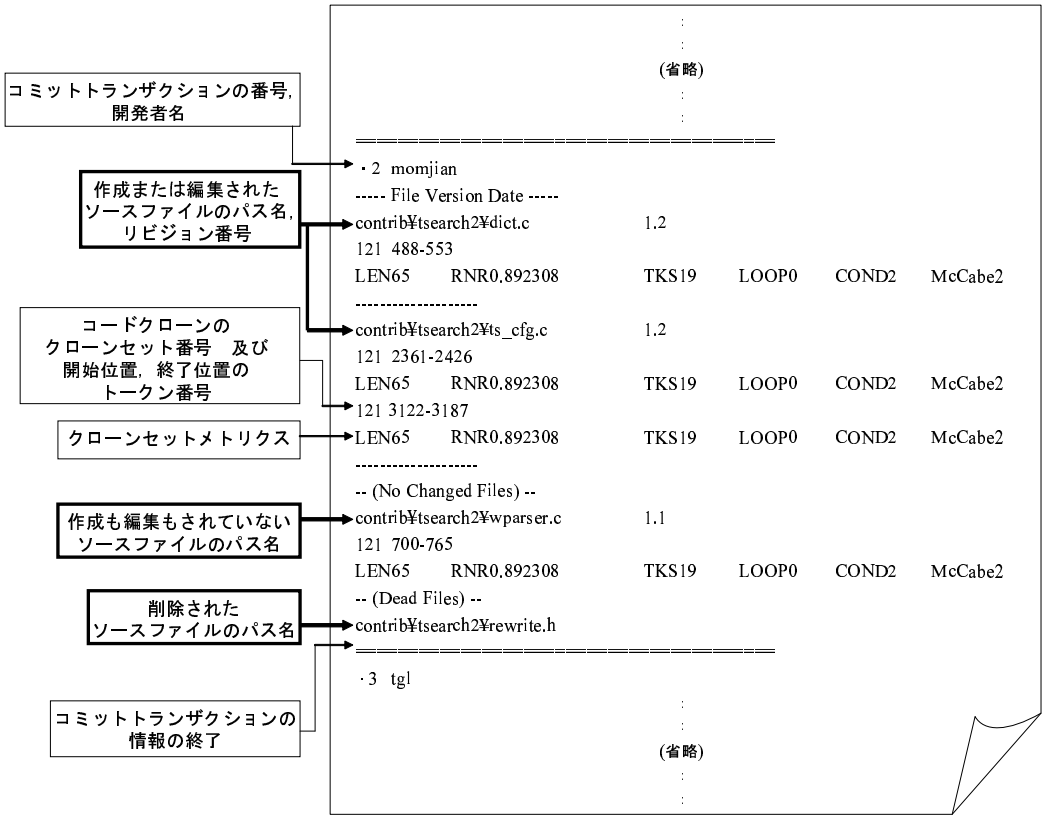

図 8: コードクローン検出部の出力ファイルの例

図 7 のように，出力ファイルには抽出したすべてのコミットトランザクションおよびその直後のスナップショットの情報が格納されている．特定のコミットトランザクション，スナップショットの情報を取得するには，各コミットトランザクションにつけた通し番号を用いる．この番号はそのスナップショット内のソースファイルの情報を取得する際にも用いる．これは，同じパス名，同じリビジョンのソースファイルでも，スナップショットによってそこに存在するコードクローンが異なる場合があるからである．なお，リビジョン番号は 4.2 節において，コード片の類似度 Sim を求める 際に用いる．

また，3.1 節において，コミットトランザクションの情報の 1 つとしてコミットされた日時を挙げているが，図 7 のコミットトランザクションにはその情報は含まれていない．その代わり，以下の情報をコミットトランザクションがコミットされた日時とする．

- コミットトランザクションにおいて作成または編集されたソースファイルがある場合
  - 作成または編集されたソースファイルの中で，ソースファイルのパス名が辞書式順序で最も前に来るものの最終変更日時
- コミットトランザクションにおいて削除されたソースファイルしか存在しない場合
  - 削除されたソースファイルの中で，ソースファイルのパス名が辞書式順序で最も前に来るものの削除日時

上記のようなソースファイルの最終変更日時をコミットトランザクションがコミットされた日時として用いるのは，CVS リポジトリでは，コミットトランザクションを構成するファイルやリビジョン情報が記録されていないからである．

各スナップショットにおけるコードクローンの抽出方法は大きく分けて以下の 2 ステップからなる．

1. 入力したリポジトリからコミットトランザクションを抽出する
2. 各コミットトランザクションの直前直後の時点におけるスナップショットを取得し，各スナップショットにおけるコードクローンを検出する

以下，各ステップの詳細について説明する．

4.1.1 コミットトランザクション情報の抽出

入力したリポジトリからコミットトランザクションを抽出する．CVS リポジトリでは，コミットトランザクションを構成するファイルやリビジョン情報が記録されていない．そのため本システムでは，変更の集合が以下の 3 つの条件をすべて満たす時，その変更の集合をコミットトランザクションとした [43].

- コミットした開発者が同じである
- コミットログが同じである
- コミットされた日時の差が 200 秒以内である

4.1.2 スナップショットの取得およびコードクローンの検出

4.1.1 節で抽出した各コミットトランザクションの直前直後の時点におけるスナップショットを取得し，各スナップショットにおけるコードクローンを検出する．

なお，3.1 節において，コミットトランザクション 1 の直前の時点におけるスナップショット (スナップショット 1) を取得することを述べているが，これは取得開始日の入力の有無によって取得方法が異なる．取得開始日の入力がない場合は，コミットトランザクション 1 の直前の時点においてはソースコードが存在しないため，スナップショット 1 は取得しない．取得開始日の入力がある場合は，取得開始日より早くにコミットされたコミットトランザクションのうち，最新の日時にコミットされたコミットトランザクションを抽出する．その後，そのコミットトランザクションの直後のスナップショットをスナップショット 1 として取得する．

また，各スナップショットを作成する時には，そのスナップショットの直前のコミットトランザクションにおいて各ソースファイルが作成または編集されているかどうかを調べておき，出力ファイルに記録する．この情報は検定用データ作成部 (4.3 節) において用いる．

各スナップショットを取得した後に，CCFinderX を用いてそのスナップショットにおけるコードクローンを検出する．その際，コードクローンセットメトリクスも計算する．

コードクローンを検出したら，各スナップショットにおけるコードクローンの情報を格納したテキストファイルを作成する．

4.2 クローン対応関係抽出部

入力 CVS リポジトリ，各スナップショットにおけるコードクローンの情報 (テキストファイル)

CVS リポジトリは 4.1 節の入力と同じものを用いる．また，4.1 節で出力されたファイルをコードクローンの情報として用いる．

出力 各コミットトランザクションにおけるクローン対応関係 (テキストファイル)
出力ファイルの形式を ER 図で表すと図 9 のようになる．また，出力ファイルの具体例を図 10 に示す.

図 9 のように，出力ファイルにはすべてのコミットトランザクションにおけるクローン対応関係が格納されている．クローン対応関係は，コミットトランザクション t 直前のスナップショット t に存在するコードクローン C_c を基準として格納する．そして，図 10 のように，コミットトランザクション t 直後のスナップショット t+1 に存在するコードクローンの中から，C_c とクローン対応関係にあるものすべての情報を，C_c の後に格納しておく．また，C_c に対する開発者の編集の有無を格納しておき，4.3 節で検定用データを求める際に用いる

スナップショット t に存在するコードクローン C_c におけるクローン対応関係の抽出方法は，大きく分けて以下の 3 ステップからなる．これをスナップショット t に存在するすべてのコードクローンに対して行うことにより，すべてのクローン対応関係を抽出する．

1. C_c の情報を抽出する
2. スナップショット t+1 に存在するコードクローンの中から C_c との類似度が最も大きくなるコードクローン C_{max} を探索する
3. C_c, C_{max} とそれらのメトリクス値を出力ファイルに書き込む

以下，各ステップの詳細について説明する．

4.2.1 クローン対応関係を取るコードクローン C_c の情報の抽出

C_c に関する以下の情報を抽出する．

- C_c の開始位置のトークン番号
- C_c の終了位置のトークン番号
- C_c を含むクローンセットのクローンセットメトリクスの値
- C_c が存在するソースファイル SF のパス名
- スナップショット t における SF のリビジョン番号 R_t
- スナップショット t+1 における SF のリビジョン番号 R_{t+1}

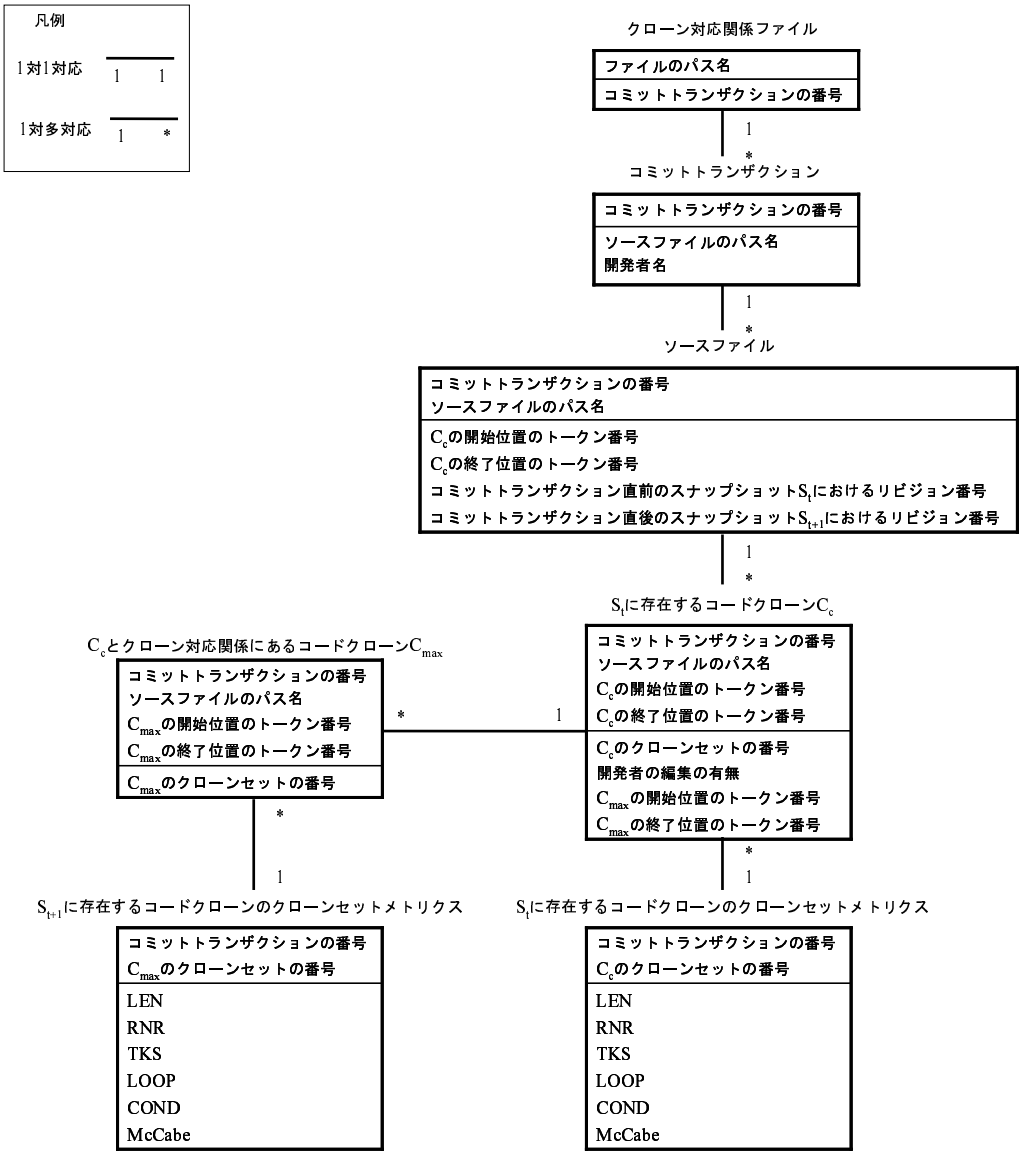

図 9: クローン対応関係抽出部の出力ファイルの形式

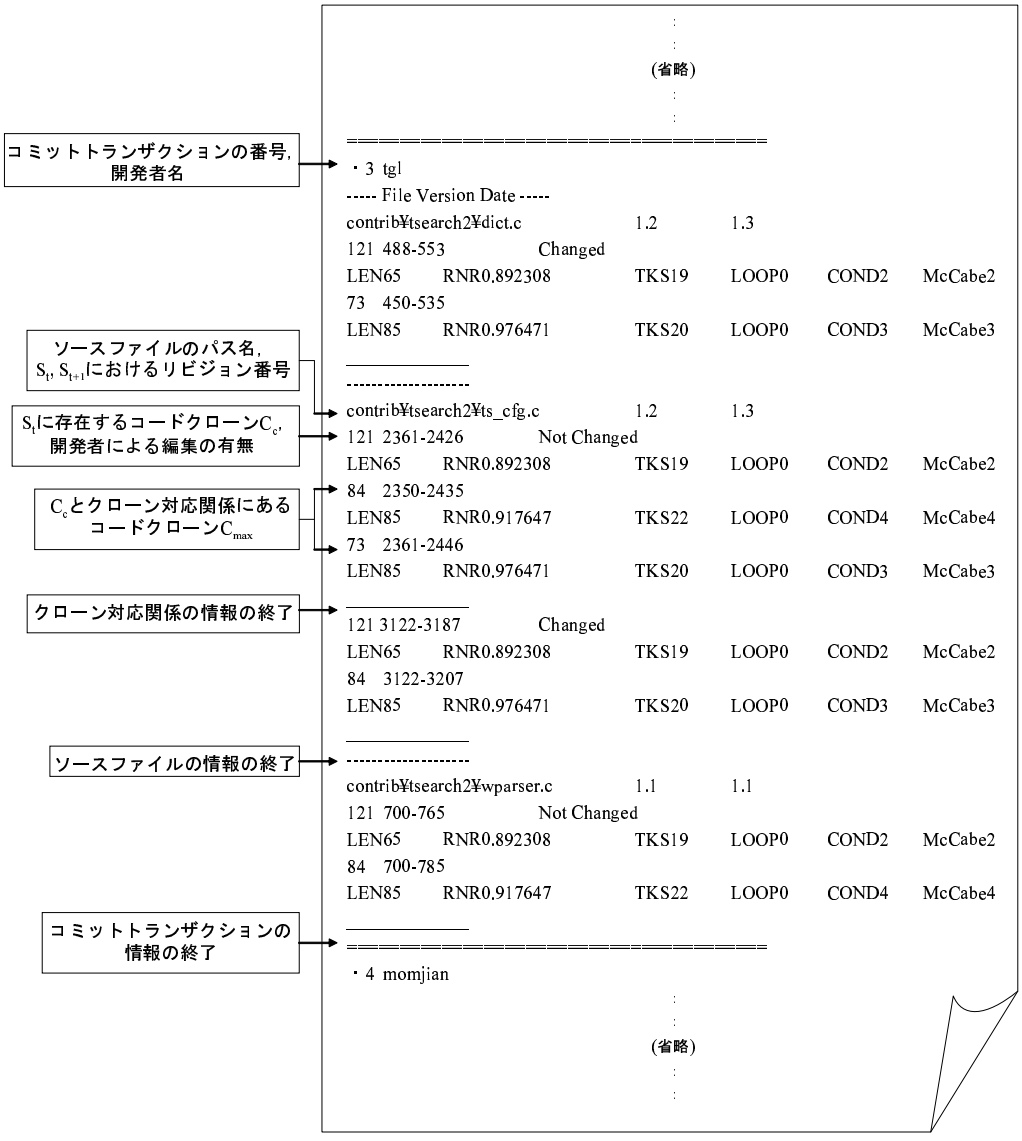

図 10: クローン対応関係抽出部の出力ファイルの例

4.2.2 C_c との類似度が最も大きくなるコードクローン C_{max} の探索

スナップショット t+1 内の SF に存在するコードクローンの中から，C_c との類似度が最も大きくなるコードクローン C_{max} を探索する．C_{max} となるコードクローンが複数ある場合，4.2.3 節においてそれらすべての情報をクローン対応ファイルに書き込む．

スナップショット t+1 内の SF に存在するコードクローン C_d と C_c の類似度は，以下の 4 ステップの手順で探索する．なお，SF に存在するコード片のうち，

- C_c の開始位置のトークン番号 $\leq$ コード片の開始位置のトークン番号
- コード片の終了位置のトークン番号 $\leq C_c$ の終了位置のトークン番号

という条件をすべて満たすコード片を C_c の**クローン片**と呼ぶことにする．

1. SF のリビジョン番号 R_t，R_{t+1} 間における差分情報を取得する
2. 1 で求めた差分情報を用いて，C_c のクローン片を抽出する
3. C_c の各クローン片のトークンのうち，C_d に対応するトークンのあるトークンの数を計算する
4. 3 で求めた対応しているトークンの数から，C_c と C_d の類似度 $Sim(C_c, C_d)$ を計算する

以下に，各ステップの詳細について説明する．

ステップ 1: SF の差分情報の取得

SF のリビジョン番号 R_t，R_{t+1} 間における差分情報を取得する．ただし，SF がコミットトランザクション t において編集されていない場合は，SF の差分情報が存在しないのでステップ 2 へ移る．また，SF がコミットトランザクション t において削除されている場合は，C_c とクローン対応関係にあるコードクローンが存在しないので下記の処理はすべて省略する．なお，SF がコミットトランザクション t において作成されている場合は，スナップショット t 内に SF が存在しないため，そもそも C_c が SF 内に存在することはありえない．

SF の差分情報を取得するには，まず，リビジョン番号 R_t，R_{t+1} の SF をそれぞれ復元し，それらをトークン列に変換したファイルの変更差分を取る．SF の復元は入力した CVS リポジトリの情報を用いて行う．

ソースファイルを復元したら，3.3.1 項のステップ 2,3 の方法を用いて差分情報を取得する．差分情報におけるファイルの変更箇所は開始位置のトークン番号が小さい順に並べ，1 から通し番号を振る．

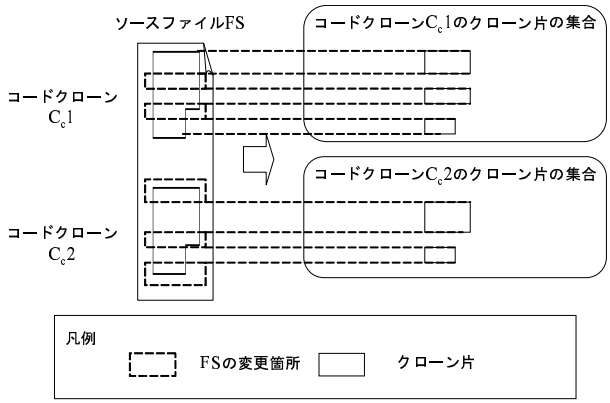

図 11: クローン片の求め方

ステップ 2: C_c のクローン片の抽出

差分情報を用いて C_c のクローン片を抽出する．差分情報においてファイルの変更箇所が存在しない場合や，C_c が差分情報におけるファイルの変更箇所と重複しない場合は，C_c 自体を C_c のクローン片とする．

C_c が差分情報におけるファイルの変更箇所と重複する場合，図 11 のように，C_c から変更箇所と重複している箇所を取り除いたものを C_c のクローン片とする．例えば，C_c の開始位置が $m(m \in N)$ 番目の変更箇所 D_m と重複し，C_c の終了位置が $n(n \in N, m < n)$ 番目の変更箇所 D_n と重複する場合，C_c のクローン片は以下の条件すべてを満たすコード片になる．なお，$i(i \in N, m \leq i < n)$ 番目の変更箇所を D_i とする．

- 開始位置が D_i の終了位置の 1 トークン後の位置
- 終了位置が D_{i+1} の開始位置の 1 トークン前の位置

なお，C_c のクローン片を求める際に，2.3.2 項の方法を用いて C_c が開発者によって編集されているかどうかを調べる．これは 4.4 節において，検定用データの要素を抽出すべきクローン対応関係を特定するために用いる．

ステップ 3: C_c の各クローン片のトークンと C_d のトークンの対応数の計算

C_c の各クローン片のトークンのうち，C_d に対応するトークンのあるトークンの数を計算する．C_c のクローン片の 1 つ C_cP_i のトークンのうち，C_d に対応するトークンのあるトークン数 $TokSum(C_cP_i, C_d)$ は以下の方法で計算する．

まず，差分情報によって，C_cP_i の開始位置，終了位置をリビジョン番号 R_{t+1} の SF 上の位置に調整する [21]．これにより，C_cP_i の各トークンと対応しているトークンの位置がわかるようになる．

そして，C_d と位置調整後の C_cP_i が重複している箇所のトークン数を求める．このトークン数が $TokSum(C_cP_i, C_d)$ の値になる．

ステップ 4: 類似度 $Sim(C_c, C_d)$ の計算

ステップ 3 で求めた C_c の各クローン片と C_d において対応するトークンの数を用いて，C_c と C_d の類似度 $Sim(C_c, C_d)$ を計算する．$Sim(C_c, C_d)$ は以下の式によって求められる．

- $CP(C)$:コードクローン C のすべてのクローン片の集合
- $TokSum(a, b)$:コード片 a のトークンのうちコード片 b に対応するトークンのあるトークンの数
- $TokSumClone(C, D)$: コードクローン C のトークンのうちコードクローン D に対応するトークンのあるトークンの数．計算方法は以下の通りである
  - $TokSumClone(C, D) = \{TokSum(a, D) \mid a \in CP(C)\}$ の要素の総和
- $Sim(C_c, C_d) = TokSumClone(C_c, C_d) \times 2 \div (C_c$のメトリクス LEN の値$+C_d$ のメトリクス LEN の値)

4.2.3 C_c, C_{max} とそれらのメトリクス値の出力ファイルへの書き込み

4.2.1 節，4.2.2 節で抽出した情報のうち，以下の情報を出力ファイルへ書き込む．

- C_c に対する開発者の編集の有無
- C_c の開始位置のトークン番号
- C_c の終了位置のトークン番号
- C_c を含むクローンセットのクローンセットメトリクスの値
- C_{max} の開始位置のトークン番号
- C_{max} の終了位置のトークン番号
- C_{max} を含むクローンセットのクローンセットメトリクスの値
- C_c, C_{max} が存在するソースファイル SF のパス名
- スナップショット t における SF のリビジョン番号 R_t
- スナップショット t+1 における SF のリビジョン番号 R_{t+1}

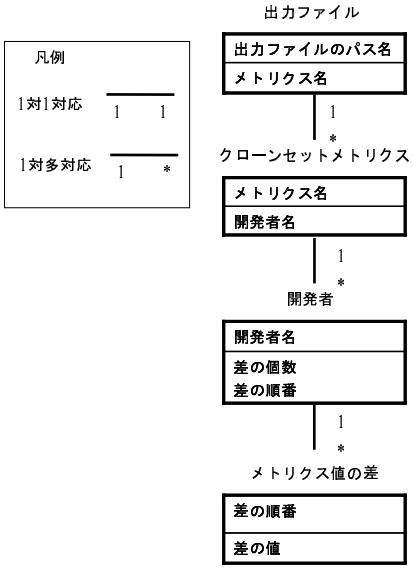

図 12: 検定用データ作成部の出力ファイルの形式

4.3 検定用データ作成部

入力 各コミットトランザクションにおけるクローン対応関係 (テキストファイル)
4.2 節で出力されたファイルをコードクローンの情報として用いる.

出力 検定用データ (テキストファイル)
出力ファイルの形式を ER 図で表すと図 12 のようになる．また，出力ファイルの具体例を図 13 に示す.

図 12 のように，出力ファイルには各メトリクス値の各開発者における差の集合が格納されている．メトリクス値の差は，その値を取得したクローン対応関係のあるコミットトランザクションおよびソースファイルの順に並んでいる．ただし，Kruskal-Wallis 検定においてはデータの順番は検定の結果に影響を及ぼさないので，差には通し番号は振っていない.

検定用データの作成方法は大きく分けて以下の 2 ステップからなる.

1. 入力ファイルからメトリクス値の差を抽出する
2. 抽出した情報を開発者ごとに集め，検定用データを作成する

以下，各ステップの詳細について説明する.

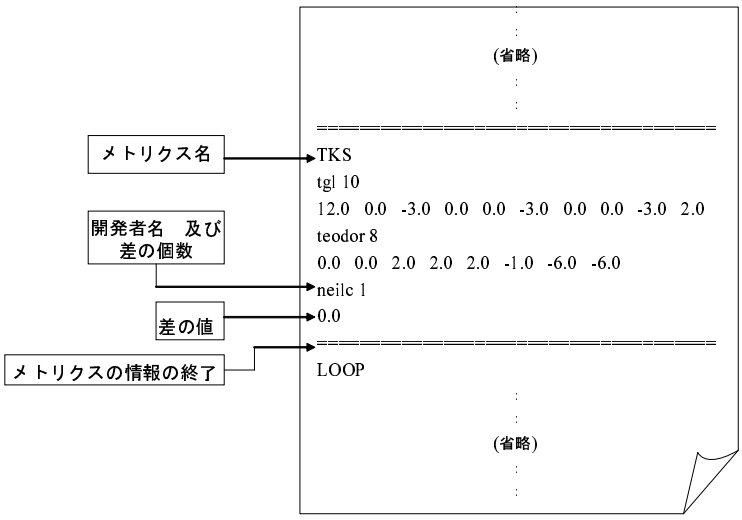

図 13: 検定用データ作成部の出力ファイルの例

4.3.1　メトリクス値の差の抽出

入力ファイルから各コミットトランザクション内のクローン対応関係を読み出し，そこにおけるメトリクス値の差を抽出する．

図 9 のように，入力されたファイルにおいてクローン対応関係は，C_c および C_c とクローン対応関係にあるコードクローンの集合として格納されている．これに対し，3.4 節の基準に従ってメトリクス値の差を求める．なお，C_c に対する開発者の編集の有無は入力されたファイルに格納されているので，それを用いてメトリクス値の差を求めるクローン対応関係を特定する．

求めたメトリクス値の差は，それを求めたクローン対応関係のあるコミットトランザクションごとに集めておく．

4.3.2　開発者による抽出した情報の分類

4.3.1 節で求めたメトリクス値の差を，それが存在するコミットトランザクションの開発者ごとに集める．これを検定用データとしてファイルに出力する．

4.4　編集傾向調査部

入力　メトリクス名，検定用データ (テキストファイル)

差の検定を行うメトリクスの名前を入力する．また，4.3 節で出力された 2 つのファイルのいずれかを検定用データとして用いる．

出力 入力されたメトリクスの要素

検定用データから，入力されたメトリクス名の列の要素をすべて抜き出して出力する．これに対して Kruskal-Wallis 検定と U 検定を 3.5 節で述べた方法で用いて，各開発者によるメトリクス値の変化を調査する．

5　コードクローンに対する編集傾向の調査

本節では 4 章で実装したシステムを用いて，実際のソフトウェアにおける開発者の編集傾向を調査を行う．また，検定結果および開発者の編集傾向に対する考察についても述べる．

5.1　調査の概要

システムに対する入力としては，以下のものを用いる．

- CVS リポジトリ：PostgreSQL[30] の CVS
- (取得開始日, 取得終了日)：以下の各組み合わせごとに入力して検定用データを取得
  - (1997/07/01,1998/01/01)
  - (1998/01/01,1998/07/01)
  - (1998/07/01,1999/01/01)

この入力から得られた検定用データにおいて，各クローンセットメトリクスの値における有意差の有無を検定を行う．

5.2　検定結果

各入力から得られた検定用データに対し有意水準 5%で Kruskal-Wallis 検定を行ったところ，統計量 H の値および開発者間の有意差の有無は表 2 のようになった．なお，Kruskal-Wallis 検定の帰無仮説は「各群で差がない」というものなので，帰無仮説が棄却されているものには有意差があるといえる．

また，Kruskal-Wallis 検定において有意差があると判断されたメトリクスの検定用データに対し，3.5 節で述べた U 検定による方法を有意水準 5%で用いて，各開発者によるメトリクス値の変化を調べたところ，表 3 のようになった．なお，表における表記の意味は以下のようになっている．

- + : その開発者は他の開発者に比べ，そのメトリクス値を有意に増加させていた
- - : その開発者は他の開発者に比べ，そのメトリクス値を有意に減少させていた
- 空白 : その開発者は他の開発者との有意差があるとはいえなかった
- X : そのメトリクスは Kruskal-Wallis 検定によって有意差があるとはいえないと判断された

5.3 考察

本節では，5.2 節の検定結果から編集傾向について考察した後，編集傾向を用いたコードクローン管理手法の例を提案する．

5.3.1 編集傾向の考察

表 3 において，開発者 F は開発に参加した全期間で多くのメトリクス値を増加させていた．本研究に用いた RNR 以外のメトリクスはコードクローンの複雑度を表すので，開発者 F の編集するコードクローンは複雑になる傾向があるといえる．

一方，開発者 B は最初の期間に COND を増加させた以外は多くのメトリクス値を減少させている．また，開発者 C は最初の期間に LEN を増加させた以外は多くのメトリクス値を減少させている．これらのことから，開発者 B と開発者 C の編集するコードクローンはシンプルになる傾向があるといえる．

また，開発者 A や開発者 D は，増加するメトリクスと減少するメトリクスが混在している．さらに，メトリクスの増減が，開発に参加した期間によって異なっている．

5.3.2 編集傾向を用いたコードクローン管理の例

5.3.1 節の考察から，各開発者の編集傾向を用いた以下のようなコードクローン管理手法が提案できる．なお，各開発者は自分の目的を達成するために，前に編集した箇所やその近傍を連続して編集する傾向があるものとする．

メトリクス値を増加させる開発者が編集したコードクローン

開発者 F が最近編集した所の近傍にあるコードクローンは，今後複雑になる可能性が高い．そのため，そのコードクローンの複雑度が現在は低くても，注意して早い時期に除去すべき可能性が高い．

メトリクス値を減少させる開発者が編集したコードクローン

開発者 B や開発者 C が最近編集した所の近傍にあるコードクローンは，今後もシンプルになる可能性が高い．そのため，そのコードクローンの複雑度が現在は高くても，除去する必要がない可能性が高い．

また，開発者 A や開発者 D は時期によって編集傾向が異なっているため，今後も継続して編集傾向を調査すべきだと考えられる．

表 2: Kruskal-Wallis 検定の結果

入力	1997/07/01,1998/01/01		1998/01/01,1998/07/01		1998/07/01,1999/01/01	
自由度	3		5		6	
検定結果	統計量 H	帰無仮説	統計量 H	帰無仮説	統計量 H	帰無仮説
LEN	11.9516	棄却	117.2066	棄却	67.5199	棄却
RNR	6.9323	採択	92.5814	棄却	34.9754	棄却
TKS	17.1416	棄却	38.3730	棄却	21.2050	棄却
LOOP	3.9001	採択	6.2575	採択	4.6103	採択
COND	8.7310	棄却	32.3617	棄却	92.4722	棄却
McCabe	6.2770	採択	33.3957	棄却	90.6987	棄却

表 3: U 検定の結果

開発者	入力	LEN	RNR	TKS	LOOP	COND	McCabe
A	1997/07/01,1998/01/01			+			
	1998/01/01,1998/07/01			-			
	1998/07/01,1999/01/01	+		-		+	+
B	1997/07/01,1998/01/01	-				+	
	1998/01/01,1998/07/01	-				-	-
	1998/07/01,1999/01/01	-				-	-
C	1997/07/01,1998/01/01	+				-	
	1998/01/01,1998/07/01		-				
	1998/07/01,1999/01/01		-			-	-
D	1997/07/01,1998/01/01						
	1998/01/01,1998/07/01	+	-				
	1998/07/01,1999/01/01		+			-	
E	1997/07/01,1998/01/01	X	X	X	X	X	X
	1998/01/01,1998/07/01						
	1998/07/01,1999/01/01	X	X	X	X	X	X
F	1997/07/01,1998/01/01	X	X	X	X	X	X
	1998/01/01,1998/07/01	+	+			+	+
	1998/07/01,1999/01/01			+		+	+
G	1997/07/01,1998/01/01	X	X	X	X	X	X
	1998/01/01,1998/07/01	X	X	X	X	X	X
	1998/07/01,1999/01/01	+	+				
H	1997/07/01,1998/01/01	X	X	X	X	X	X
	1998/01/01,1998/07/01	X	X	X	X	X	X
	1998/07/01,1999/01/01						

6　まとめと今後の課題

本研究では，メトリクス値の変化を用いて，コードクローンの編集傾向を調査した．その結果，メトリクスの変化は開発者ごとに差があり，各開発者に編集傾向があることが分かった．

今後の課題としては，他のメトリクス値を用いた編集傾向の調査や，モジュールや開発時期といった開発者以外の要因による影響の除外が挙げられる．

謝辞

本研究において，常に適切な御指導および御助言を賜わ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致します．

本研究において，適切な御指導および御助言を賜わ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 准教授に深く感謝致します．

本研究において，適切な御指導およ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 助教に深く感謝致します．

本研究において，適切な御指導およ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 早瀬 康裕 特任助教に深く感謝致します．

本研究において，逐次適切な御指導およ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 肥後 芳樹 助教に深く感謝致します．

本研究において，様々な御指導およ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 市井 誠 氏に深く感謝します．

本研究において，御助言を頂きました奈良先端科学技術大学院大学情報科学研究科情報システム学専攻 川口 真司 助教に深く感謝致します．

最後に，その他様々な御指導，御助言等を頂い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致します．

参考文献

[1] A. Aiken, “A System for Detecting Software Plagiarism (Moss Hompage)”, **http://theory.stanford.edu/˜aiken/moss/**, [Last visited 30 Jan 2008]

[2] B. S. Baker: “A Program for Identifying Duplicated Code”, *Proceedings of the 24th Symposium of Computing Science and Statistics*, pp.49-57, 1992.

[3] B. S. Baker: “On Finding Duplication and Near-Duplication in Large Software Systems”, *Proceedings of the 2nd Working Conference on Reverse Engineering*, pp.86-95, 1995.

[4] B. S. Baker: “Parameterized Duplication in Strings: Algorithms and an Applicationto Software Maintenance”, *SIAM Journal on Computing*, Vol.26, No.5, pp.1343-1362, 1997.

[5] T. Bakota, R. Ferenc and T. Gyimothy: “Clone Smells in Software Evolution.”, *Proceedings of the 23rd International Conference on Software Maintenance*, pp.24-33, 2007.

[6] M. Balazinska, E. Merlo, M. Dagenais, B. Lague and K. Kontoginannis: “Advanced Clone-Analysis to Support Object-Oriented System Refactoring”, *Proceedings of the 7th Working Conference on Reverse Engineering*, pp.98-107, 2000.

[7] M. Balazinska, E. Merlo, M. Dagenais, B. Lague and K. Kontoginannis: “Measuring Clone Based Reengineering Opportunities”, *Proceedings of the 6th International Symposium on Software Metrics*, pp.292-303, 1999.

[8] M. Balazinska, E. Merlo, M. Dagenais, B. Lague and K. Kontoginannis: “Partial Redesign of Java Software Systems Based on Clone Analysis”, *Proceedings of the 6th International Working Conference on Reverse Engineering*, pp.326-336, 1999.

[9] I. D. Baxter, A. Yahin, L. Moura, M. Sant’Anna and L. Bier: “Clone Detection Using Abstract Syntax Trees”, *Proceedings of the 14th International Conference on Software Maintenace*, pp.368-377, 1998.

[10] B. Berliner: “CVS II: Parallelizing Software Development”, *Proceedings of the USENIX Winter 1990 Technical Conference*, pp.341-352, 1990.

[11] E. Burd and J. Bailey: “Evaluating Clone Detection Tools for Use during Preventative Maintenance”, *Proceedings of the 2nd International Workshop on Source Code Analysis and Manipulation*, pp.36-43, 2002.

[12] CCFinder Official Site, **http://www.ccfinder.net/**, [Last visited 30 Jan 2008]

[13] S. Ducasse, M. Rieger and S. Demeyer: “A Language Independent Approach for Detecting Duplicated Code”, *Proceedings of the 15th International Conference on Software Maintenance*, pp.109-118, 1999.

[14] P. Frohlich and W. Nejdl: “WebRC: Configuration Management for a Cooperation Tool”, *Proceedings of the 7th Workshop on Software Configuration Management*, pp.175-185, 1997.

[15] R. Geiger, B. Fluri, H. C. Gall and M. Pinzger: “Relation of code clones and change couplings”, *In Proceedings of the 9th International Conference of Funtamental Approaches to Software Engineering*, pp.411-425, 2006.

[16] D. Gusfield: “Algorithms on Strings, Trees, and Sequences” , *Cambridge University Press*, 1997.

[17] IBM Rational ClearCase, **http://www-306.ibm.com/software/awdtools/clearcase/**, [Last visited 31 Jan 2008]

[18] 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, 2002.

[19] C. Kapser and M. W. Godfrey: ““Cloning Considered Harmful” Considered Harmful”, *Proceedings of the 13th Working Conference on Reverse Engineering*, pp.19-28, 2006.

[20] 川口真司, 松下誠, 井上克郎: “版管理システムを用いたクローン履歴分析手法の提案”, 電子情報通信学会論文誌D, Vol.J89-D, No.10, pp.2279-2287, 2006.

[21] 川口真司, 松下誠, 井上克郎, 飯田元: “コードクローン履歴閲覧環境を用いたクローン評価の試み”, 情報処理学会研究報告, Vol.2006, No.125, pp.49-56, 2006.

[22] M. Kim and D. Notkin: “Using a clone genealogy extractor for understanding and supporting evolution of code clones”, *Proceedings of the 2nd International Workshop on Mining Software Repositories*, pp.17-21, 2005.

[23] R. Komondoor and S. Hirwitz: “Using Slicing to Identify Duplication in Source Code”, *Proceedings of the 8th International Symposium on Static Analysis*, pp.40-56, 2001.

[24] J. Krinke: “Identifying Similar Code with Program Dependece Graphs”, *Proceedings of the 8th Working Conference on Reverse Enngineering*, pp.562-584, 2001.

[25] W. H. Kruskal and W. A. Wallis: “Use of ranks in one-criterion variance analysis”, *Journal of the American Statistical Association*, Vol.47, No.260, pp.583-621, 1952.

[26] Z. Li, S. Lu, S. Myagmar and Y. Zhou: “CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code”, *IEEE Transactions on Software Engineering*, Vol.32, No.3, pp.176-192, 2006.

[27] H. B. Mann and D. R. Whitney: “On a test of whether one of two random variables is stochastically larger than the other”, *Annals of Mathematical Statistics*, Volume 18, pp.50-60, 1947.

[28] J. Mayland, C. Leblanc and E. Merlo: “Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics”, *Proceedings of the 12th International Conference on Software Maintenace*, pp.244-253, 1996.

[29] OpenBSD, **http://www.openbsd.org/**, [Last visited 31 Jan 2008]

[30] PostgreSQL, **http://www.postgresql.org/**, [Last visited 30 Jan 2008]

[31] L. Prechelt, G. Malpohl and M. Philippsen: “Finding Plagiarisms among a Set of Programs with JPlag”, *Journal of Universal Computer Science*, Vol.8, No.11, pp.1016-1038, 2002.

[32] Pvcs version control software, **http://www.serena.com/pvcs-version-manager.html**, [Last visited 31 Jan 2008]

[33] C. Reidar and W. Bernhard: “Version models for software configuration management”, *ACM Computing Surveys*, Vol.30, No.2, pp.232-282, 1998.

[34] C. K. Roy and J. Cordy: “A Survey on Software Clone Detection Research”, *Technical Report No. 2007-541 School of Computing Queen’s University at Kingston Ontario*, 2007.

[35] K. Shioya: “Overview of JIS X0161:2002 Software Maintenance”, *Technical Report SRA-KTL-2003J-001*.

[36] The FreeBSD Project, **http://www.freebsd.org/** [Last visited 31 Jan 2008]

[37] Y. Ueda, T. Kamiya, S. Kusumoto and K. Inoue: “On Detection of Gapped Code Clones using Gap Locations”, *Proceedings of the 9th Asia-Pacific Software Engineering Conference*, pp.327-336, 2002.

[38] A. Ulf, B. Lars, B. C. Henrik and M. Boris: “The Unified Extensional Versioning Model”, *Proceedings of the 9th International Symposium on System Configuration Management*, pp.100-122, 1999.

[39] Visual sourcesafe 2005, **http://msdn2.microsoft.com/en-us/vstudio/aa718670.aspx**, [Last visited 31 Jan 2008]

[40] F. T. Walter: “RCS - A System for Version Control”, *Software - Practice and Experience*, Vol.15, No.7, pp.637-654, 1985.

[41] X. Yan, J. Han and R. Afshar: “CloSpan: Mining Closed Sequential Patterns in Large Datasets”, *Proceedings of the 3rd International Conference on Data Mining*, pp.166-177, 2003.

[42] 吉田則裕, 肥後芳樹, 神谷年洋, 楠本真二, 井上克郎: “コードクローン間の依存関係に基づくリファクタリング支援”, 情報処理学会論文誌, Vol.48, No.3, pp.1431-1442, 2007.

[43] T. Zimmermann and P. Weissgerber: “Preprocessing CVS Data for Fine-Grained Analysis”, *Proceedings of the 1st International Workshop on Mining Software Repositories*, pp.2-6, 2004.