特別研究報告

題目

ソースコードの編集に基づいたコードクローンの分類とその分析システム

指導教員
井上 克郎 教授

報告者
山中裕樹

平成 24 年 2 月 16 日

大阪大学 基礎工学部 情報科学科

平成 23 年度 特別研究報告

ソースコードの編集に基づいたコードクローンの分類とその分析システム

山中裕樹

内容梗概

ソフトウェアの保守工程における大きな問題の一つとしてコードクローンが指摘されている．コードクローンとは，ソースコード中に存在する，互いに類似または一致した部分を持つコード片のことであり，主にソースコードのコピーアンドペーストや同一処理を意図的に繰り返し書くことによって発生する．

コードクローンに対する保守作業として，同時修正と集約が挙げられる．同時修正とは，クローンセット（互いにコードクローンとなっているコード片の集合）に含まれる全てのコードクローンを一貫して編集することである．例えば，クローンセットの中の 1 つのコードクローンに欠陥が含まれている場合，同じクローンセットに属する他のコードクローンについても同様の欠陥が含まれている可能性がある．このようなコードクローンは同時に修正を行う必要性が考えられる．また，集約とはクローンセットに含まれるコードクローンを 1 つのメソッドなどにまとめることである．集約を行うことによって保守の対象となるコードクローンの存在を除去することが可能となる．

上述したようなコードクローンに対する保守作業を効率良く行うために，コードクローンの変更管理が考えられる．例えば，クローンセット中の一部のコードクローンが編集された場合，そのクローンセットに属する他のコードクローンについても一貫した編集が必要となる可能性が考えられる．また，新たに発生したコードクローンは集約の対象となる可能性が考えられる．一般的に，ツールを用いた自動的なコードクローン検出が行われているが，このような保守作業の対象となる変更されたコードクローンの確認作業は人手で行う必要がある．従って，検出されたコードクローンが膨大な量となる場合，その中からコードクローンの変更情報を確認するコストは大きいと考えられる．

そこで本研究では，過去のバージョンと現在のバージョンの 2 バージョン間のソースコードの編集に基いて，コードクローン・クローンセットを分類する手法を提案した．そして，分類結果に基づいて，2 バージョン間のコードクローン・クローンセットの変更情報を開発者に提供するシステムの開発を行った．また，実際に開発したシステムを企業のソフトウェア開発現場に適用し，開発者にアンケートをとることによってその有用性を確かめることができた．

主な用語

コードクローン
ソフトウェア開発履歴
ソフトウェア保守

目 次

1　まえがき

ソフトウェアの保守工程における大きな問題の一つとしてコードクローンが指摘されている．コードクローンとは，ソースコード中に存在する，互いに類似または一致した部分を持つコード片のことである [1]．コードクローンは主にソースコードのコピーアンドペーストや同一処理を意図的に繰り返し書くことによって発生する．

コードクローンに対する保守作業として，同時修正と集約が挙げられる．同時修正とは，互いにコードクローンとなっているコード片の集合（以下，クローンセット）に含まれる全てのコードクローンを一貫して編集することである．例えば，クローンセットの中の 1 つのコードクローンに欠陥が含まれている場合，同じクローンセットに属する他のコードクローンについても同様の欠陥が含まれている可能性がある．このようなコードクローンは同時に修正を行う必要性が考えられる．従って，クローンセット中の一部のコードクローンが編集された場合，そのクローンセットに属する他のコードクローンについても一貫した編集が必要となる可能性が考えられる．また，集約とはクローンセットに含まれるコードクローンを 1 つのメソッドなどにまとめることである．集約を行うことによって保守の対象となるコードクローンの存在を除去することが可能となる．従って，新たに発生したコードクローンは集約の対象となる可能性が考えられる．

上述したような新たに発生したコードクローンや，一部のコードクローンが編集されたクローンセットなどの変更情報を提供することによって，ソフトウェア開発者は保守作業の対象となるコードクローンの存在を把握することが可能となる．一般的にソフトウェア開発ではツールを用いた自動的なコードクローン検出が行われる [2]．しかし，検出されたコードクローンの中には，悪影響を及ぼさないなどの理由から意図的に除去されていないコードクローンが多く存在する [3]．もし検出されたコードクローンが膨大な量となる場合，コードクローンやクローンセットの変更情報を確認するコストは大きいと考えられる．

そこで本研究では，過去のバージョンと現在のバージョンの 2 バージョン間のソースコードの編集に基き，コード片の追加・編集・削除などに応じてコードクローン・クローンセットを分類する手法を提案した．そして，分類結果に基づいて，保守作業が必要となる可能性がある 2 バージョン間のコードクローン・クローンセットの変更情報を開発者に提供するシステムの開発を行った．開発者への提供は，テキストベースの電子メールによる通知とウェブベースのユーザインタフェースを実現している．また，開発したシステムをオープンソースソフトウェアや実際のソフトウェア開発現場に適用することによって，その有用性を評価した．

以降，2 節では本研究の背景を説明し，3 節ではソースコードの編集に基づくコードクローン・クローンセットの分類手法について説明する．また，4 節では本研究で開発したシステ

ムの概要について説明し，5 節で本システムの評価実験について述べる．そして，6 節で本研究の関連研究について説明し，7 節で本研究のまとめと今後の課題を述べる．

2　背景

本節では，本研究の背景として，コードクローン，および，コードクローンの履歴分析について説明する．

2.1　コードクローン

コードクローン (Code clone) とは，ソースコード中に存在する互いに一致または類似したコード片を指す [1]．一般的に，コードクローンの存在はソフトウェアの保守を困難にすると言われている．例えば，コードクローンとなっているコード片中に欠陥が存在する場合，そのコード片と一致または類似した他のコードクローンについても同様の欠陥が存在する可能性がある．ソフトウェアの規模が非常に大きい場合は，開発者が全てのコードクローンを認識することは非現実的である．そのため，ツールを用いた自動的なコードクローン検出が行われる．

一般的に，互いに一致または類似したコードクローンの対をクローンペア (Clone pair) と呼び，クローンペアにおいて推移関係が成り立つコードクローンの集合をクローンセット (Clone set) と呼ぶ．

2.1.1　発生原因

コードクローンがソースコード中に発生する原因として，以下のものが挙げられる [2, 4, 5]．

既存コードのコピーアンドペーストによる再利用

一からソースコードを書くよりも，同様の処理を行う既存コードを流用し，部分的な変更を加える方が信頼性が高い．従って，実際にはコピーアンドペーストによる既存コードの再利用が多く存在する．

定型処理

定義上簡単で，頻繁に用いられる処理はコードクローンになる傾向がある．例として，キューの挿入処理，データ構造アクセス処理などが挙げられる．

プログラミング言語における適切な機能の欠如

抽象データ型やローカル変数を用いることができない場合には，同じようなアルゴリズムを持つ処理を繰り返し書かなくてはならない場合がある．

パフォーマンスの改善

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，特定のコード片を意図的に繰り返し記述することで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

コード生成ツールはあらかじめ定められたコードをベースにして自動的にコードを生成する．そのため，目的の処理が類似している場合，識別子などを除いて類似したコードが生成される．

複数のプラットフォームに対応したコード

複数の OS や CPU に対応したソフトウェアは各プラットフォーム用のコード部分に重複した処理が存在する傾向がある．

偶然

偶然，開発者が同一のコードを書いてしまう場合がある．

2.1.2 コードクローンの定義

コードクローンには様々な検出方法が存在するが，そのどれもが異なったコードクローンの定義を持つ．従って，コードクローンの厳密で普遍的な定義は存在しない．Bellon は，コードクローン間の違いの度合いに基づき，それらを以下のような三つの定義に分類している [6, 7].

タイプ 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を除き，完全に一致するコードクローン．

タイプ 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．

タイプ 3

タイプ 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．

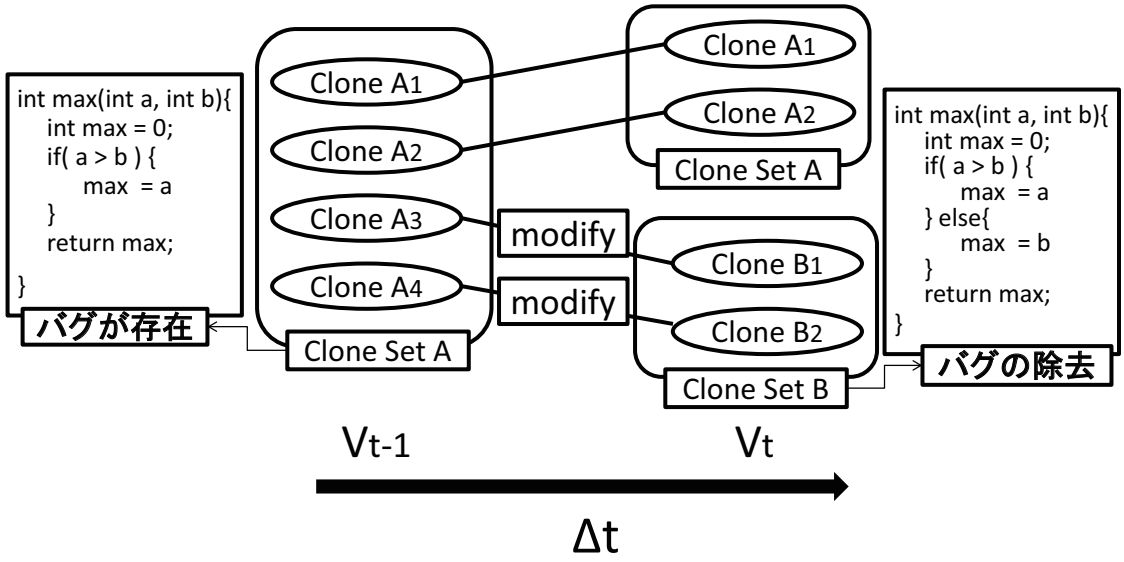


図 1: クローンセットの分岐

2.1.3 コードクローン検出ツール

コードクローン検出手法には，ソースコードの字句解析に基づく手法 [8, 9, 10] や，特徴メトリクスに基づく手法 [11, 12] などが存在する．ソースコードの字句解析に基づく手法では，ソースコード中で同一の文字列を検索することでコードクローンの検出を行う．特徴メトリクスに基づく手法では，クラスや関数，ファイルのようなプログラム中のある種の単位ごとに特徴メトリクスを定義・算出し，それらのメトリクス値が類似したものをコードクローンとして抽出する．

本研究では，字句解析ベースの検出ツール CCFinder [2] を利用してコードクローンの検出を行う．CCFinder は高いスケーラビリティを有しており，大規模なソフトウェアに対しても実用的な解析を行うことができる．また，表現上の差異があるコードクローンを検出することができる．実際に，さまざまな大規模ソフトウェアへ適用され，その有用性が確認されている [13].

CCFinder でコードクローンの検出を行う場合，その処理は，字句解析，変換処理，検出処理，出力整形処理からなる．字句解析でソースコードをトークン列に変換し，変換処理で変数名や関数名等を同一のトークンに変換する．その後，検出処理で閾値以上の長さの共通トークン列を探索し，全てのコードクローンの対のリストを出力する．従って，CCFinder は 2.1.2 節で述べたタイプ 1，タイプ 2 のコードクローンを検出することが可能となる．

2.2 コードクローンの履歴分析

ある時点のソースコードにおいて欠陥が含まれているコードクローンが発見された場合，同じクローンセットに属する他のコードクローンについても同様の欠陥が含まれている可能性がある．それらの一部のコードクローンのみが修正された場合，他のコードクローンについても同様の修正が必要となる可能性がある．このような修正されたコードクローンが含まれるクローンセットの情報を開発者に提供することによって，欠陥を含むコードクローンを確認することが可能となる．

しかし，実際には関連性が高いにも関わらず，ある一時点のソースコードを解析するだけでは検出することができないコードクローンが存在する．そのような例を図 1 に示す．この例では，時刻 t のバージョンにおけるソースコードの集合を V_t，そこから Δt だけ過去の時点におけるバージョンのソースコードの集合を V_{t-1} とし，V_t，V_{t-1} にそれぞれ含まれるコードクローンを表している．そして，V_{t-1} 中のクローンセット A に不具合が発見されたため，コードクローン A_3，A_4 が修正され，新たなクローンセット B に分岐している．このような場合，コードクローン A_1, A_2 についても同様の修正が必要となると考えられる．しかしながら，V_t のソースコードに対してのみ分析を行うだけでは，クローンセット A と B は無関係のものとなってしまい，それらの関連性の高さを知ることはできない．

文献 [14] では，このようなクローンセットの分岐などを分析するため，コードクローンの履歴分析手法を提案している．コードクローンの履歴分析では，まず，コードクローンの開始行と終了行の対応関係に基づいて，過去のバージョンのコードクローンが現時点のどのコードクローンに対応しているかを調べている．そして，コードクローンがどのような遷移をたどってきたのかを特定している．

本研究では，コードクローンの履歴分析の手法に基づいて，2 バージョン間で対応するコードクローンを求めている．そして，コード片の追加・編集・削除やクローンセットの分岐などの変更情報に基づいて，コードクローン・クローンセットの分類を行っている．また，それらの分類結果から保守作業の対象となるコードクローンの変更情報をソフトウェア開発者に提供するシステムの開発を行っている．

3　提案手法

1 節で述べたように，コードクローン・クローンセットの変更情報を開発者に提供することによって，コードクローンに対する保守作業が効率的になると考えられる．本研究では，2 バージョン間のソースコードにおけるコードクローンの親子関係を求め，コード片の編集状況に基き，コードクローン・クローンセットの分類を行っている．コードクローンの親子関係については，3.2 節で詳細に定義する．本節では，これらの分類手法について説明する．ここで，本節で用いる各種記号の一覧を表 1 に示す．

3.1　概要

コードクローン，および，クローンセットの分類手法の概要を以下に示す．入力は，分析の対象となる 2 バージョンのソースコード V_t，V_{t-1} である．V_t は、最新バージョンのソースコード，また，V_{t-1} は過去のバージョンのソースコードを意味している．以降，V_t のソースコードに含まれる全てのコードクローンの集合を C_t，V_{t-1} のソースコードに含まれる全てのコードクローンの集合を C_{t-1} とする．

STEP1: コードクローン集合の取得

V_t と V_{t-1} 全体に CCFinder を適用し，コードクローンの集合 C_t，C_{t-1} の検出を行う．

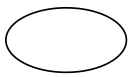
STEP2: コードクローンの親子関係を求める

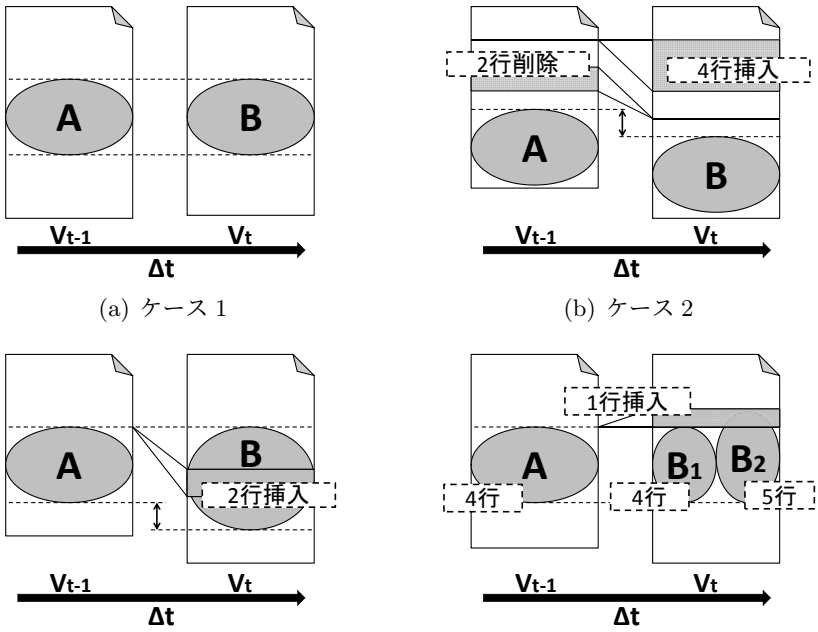
コードクローンの変化の分析を行うためには，C_t に含まれるコードクローンと C_{t-1} に含まれるコードクローンの対応関係を調べる必要がある．そこで，3.2 節で説明する定義に基づいて，コードクローンの親子関係を求める．

STEP3: コードクローンの分類

3.3 節で説明する定義に基づいて，2 バージョンのソースコードに含まれる全てのコードクローンを分類する．

表 1: 各種記号一覧

ソースファイル	コードクローン	コード片	クローンセット	親子関係
				



(a) ケース 1　(b) ケース 2

(c) ケース 3　(d) ケース 4

図 2: コードクローンの親子関係

STEP4: クローンセットの分類

3.4 節で説明する定義に基づいて，2 バージョンのソースコードに含まれる全てのクローンセットを分類する．

3.2 コードクローンの親子関係

コードクローンの変化の分析を行うためには，C_t に含まれるコードクローンと C_{t-1} に含まれるコードクローンの対応関係を調べる必要がある．

本手法では，あるコードクローン $A \in C_{t-1}$ に対応するコード片をその開始行と終了行に基づいて求め，そのコード片がコードクローンとなっている場合，親子関係を定義する．そして，コードクローン $A \in C_{t-1}$ に対応するコードクローンを $B \in C_t$ と表した場合，コードクローン A をコードクローン B の**親クローン**，コードクローン B をコードクローン A の**子クローン**とする．また，コードクローン A と B の間には**親子関係が存在する** とする．

実際にコードクローンの親子関係を求めるためには，2 バージョン間のソースコードの差分が必要となる．そこで，本手法では GNU diff [1]を用いて差分の取得を行なっている．コードクローン $A \in C_{t-1}$ と $B \in C_t$ の間の親子関係を定義するに当たって，以下の 4 つの場合を考える．

[1]http://www.gnu.org/software/diffutils/.

ケース 1

V_{t-1} において，図 2(a) のように，あるコードクローン $A \in C_{t-1}$ が含まれるファイルが編集されていない場合，A に対応する全く同一のコード片 B が V_t の同一ファイル中に存在するはずである．この場合，A と B の開始行と終了行は同一となる．B が C_t に含まれ，A と B がクローンペアである場合，コードクローン A と B の間に親子関係が存在すると定義する．

ケース 2

次に，コードクローン $A \in C_{t-1}$ が含まれるファイルに何らかの編集がされているが，A には編集が行われていない場合を考える．そのような例を図 2(b) に示す．この場合，2 バージョン間でソースコードの編集操作を考慮することによって，V_t 中に A と対応する同一のコード片 B を求める事が可能となる．すなわち，A の前で 4 行の挿入と 2 行の削除が行われているため，B の開始行と終了行はそれぞれ A の開始行と終了行に 2 行追加した値となる．B が C_t に含まれ，A と B がクローンペアである場合，コードクローン A と B の間に親子関係が存在すると定義する．

ケース 3

次に，コードクローン $A \in C_{t-1}$ が編集されている場合を考える．そのような例を図 2(c) に示す．この場合も，A 中の編集操作を考慮することによって，V_t 中に A と対応するコード片 B を求めることが可能となる．すなわち，A の前で編集操作は行われていないために B の開始行は A の開始行と同じになる．また，A に 2 行の挿入が行われているために B の終了行は A の終了行に 2 行追加した値となる．B が C_t に含まれる場合，すなわちコードクローンとなる場合，コードクローン A と B の間に親子関係が存在すると定義する．

ケース 4

最後に，コードクローン $A \in C_{t-1}$ の開始部分，または，終了部分が編集されている場合を考える．この場合，V_t 中に A と対応するコード片を一意に定めることはできない．そのような例を図 2(d) に示す．この例では，“コードクローン A の直前に 1 行挿入された” という解釈と “コードクローン A に 1 行挿入された” という解釈の 2 つの解釈が可能となる．前者の場合は A に対応するコード片は B_1 となり，後者の場合は A に対応するコード片は B_2 となる．従って，2 つのコード片 B_1, B_2 がそれぞれコードクローンとなる場合，A の子クローンの候補は 2 つ存在することになる．この場合，A とのテキスト類似度が大きい方のコードクローンを子クローンと定める．コード片 X と Y のテキスト類似度は以下の式で表

される．

$$TextSim(X,Y) = \frac{2|X \cap Y|}{|X| + |Y|}$$

ただし，

$$\begin{cases} |X| = \text{コード片 } X \text{ の行数} \\ |Y| = \text{コード片 } Y \text{ の行数} \\ |X \cap Y| = GNU\ diff \text{ によって同一と判定された行数} \end{cases}$$

とする．

図 2(d) の場合，A と B_1 は同一のコード片なのでテキスト類似度は 1，A と B_2 のテキスト類似度は約 0.89 となる．従って，A の子クローンは B_1 となる．

3.3 コードクローンの分類

本研究では, 2 バージョン間のソースコード V_t，V_{t-1} に含まれる全てのコードクローンを，そのコード片の編集状況などに応じて，“Stable Clone”，“Modified Clone”，“Moved Clone”，“Added Clone”，“Deleted Clone”の 5 項目に分類している．これらの分類を定義する上で，まず，C_t，C_{t-1} に含まれる任意のコードクローン X について以下の 4 つの命題を定める．

$$\begin{cases} P(X) : X \text{ の親クローンが存在する} \\ Q(X) : X \text{ の子クローンが存在する} \\ R(X) : 2 \text{ バージョン間で } X \text{ が編集されている} \\ S(X) : X \text{ の親クローンと } X \text{ がクローンペアである} \end{cases}$$

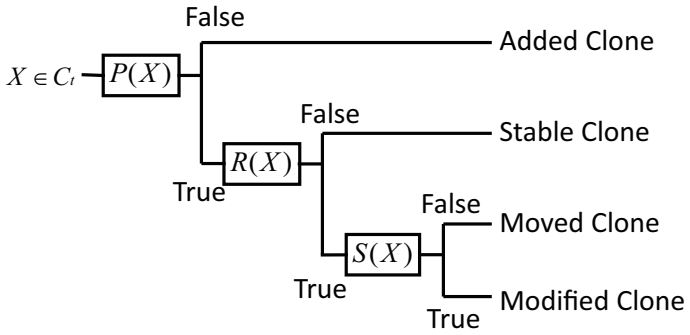
これらの命題を用いて，コードクローンの分類を定義する．C_t に含まれるコードクローンの分類を図 3(a)，C_{t-1} に含まれるコードクローンの分類を図 3(b) に示す．

Stable Clone

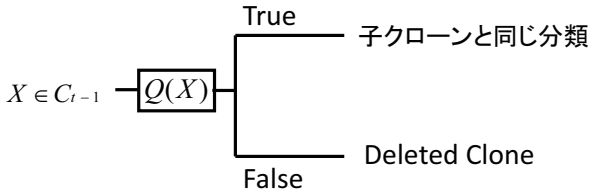
以下の命題論理式を真にするコードクローン $X \in C_t$ を Stable Clone と定義する．

$$P(X) \wedge \neg R(X)$$

すなわち，Stable Clone とは図 4(a) 中のコードクローン A_1, A_2, A_3 のように，コード片が編集されておらず，全く同一のコードクローンを表す．また，その親クローンも Stable Clone に分類する．



(a) コードクローン $X \in C_t$ の分類



(b) コードクローン $X \in C_{t-1}$ の分類

図 3: コードクローンの分類

Modified Clone

以下の命題論理式を真にするコードクローン $X \in C_t$ を Modified Clone と定義する．

$$P(X) \wedge R(X) \wedge S(X)$$

すなわち，Modified Clone とは図 4(b) 中のコードクローン A_3 のように，変数名の変更などの編集がなされ，2 バージョン間で同じクローンセットに属しているコードクローンを表す．また，その親クローンも Modified Clone に分類する．

図 4(b) において，もし V_{t-1} 中でクローンセット A に属する全てのコードクローンに不具合が含まれ，コードクローン A'_3 のみが修正された場合，編集が行われなかったコードクローン A'_1, A'_2 に対しても一貫した修正が必要となる可能性が考えられる．従って，Modified クローンが含まれるクローンセットの情報を開発者に提供することによって，このような保守作業が必要となるコードクローンの存在を確認することが可能となる．

Moved Clone

以下の命題論理式を真にするコードクローン $X \in C_t$ を Moved Clone と定義する．

$$P(X) \wedge R(X) \wedge \neg S(X)$$

すなわち，Moved Clone とは図 4(c) 中のコードクローン B_1 のように，文の挿入などの編集がなされ，2 バージョン間で異なるクローンセットに属するようになったコードクローンを表す．また，その親クローンも Moved Clone に分類する．

図 4(c) において，もし V_{t-1} 中でクローンセット A に属する全てのコードクローンに不具合が含まれ，コードクローン A'_3 のみが修正された場合，編集が行われなかったコードクローン A'_1, A'_2 に対しても一貫した修正が必要となる可能性が考えられる．Moved クローンが，V_{t-1} で属していたクローンセットと，V_t で属していたクローンセットの情報を開発者に提供することによって，このような保守作業が必要となるコードクローンの存在を確認することが可能となる．

また，図 1 のコードクローン A_3, A_4 も，コード片が修正されたことによって V_t では新たにクローンセット B を形成している．従って，コードクローン B_1, B_2 と，その親クローンであるコードクローン A_3, A_4 は Moved Clone に分類されることになる．このように，Moved Clone が含まれるクローンセットはその分岐情報を表している．

Added Clone

以下の命題論理式を真にするコードクローン $X \in C_t$ を Added Clone と定義する．

$$\neg P(X)$$

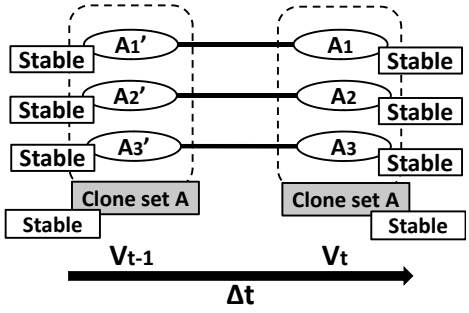
すなわち，Added Clone とは図 4(d) のコードクローン A_1, A_2, A_3 のように，コピーアンドペーストなどによって 2 バージョン間で新たに発生したコードクローンを表す．このようなコードクローンは，集約の対象となる可能性が考えられる．

Deleted Clone

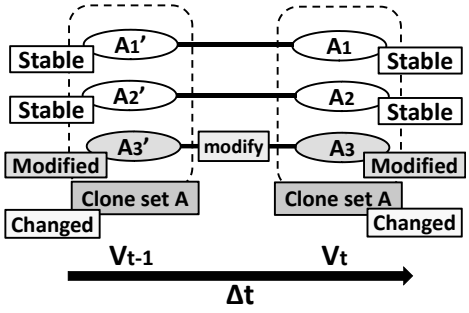
以下の命題論理式を真にするコードクローン $X \in C_{t-1}$ を Deleted Clone と定義する．

$$\neg Q(X)$$

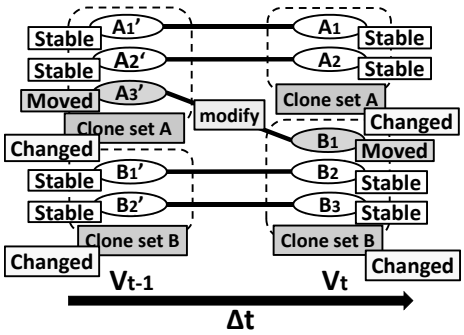
すなわち, Deleted Clone とは図 4(e) のコードクローン A'_1, A'_2, A'_3 のように，コード片の削除などによって V_t ではコードクローンではなくなったものを表す．もし，コード片に文の挿入などの大幅な編集がなされたことによってコードクローンではなくなってしまった場合，V_{t-1} で属していたクローンセットに含まれる他のコードクローンについて，一貫した編集が必要である可能性が考えられる．



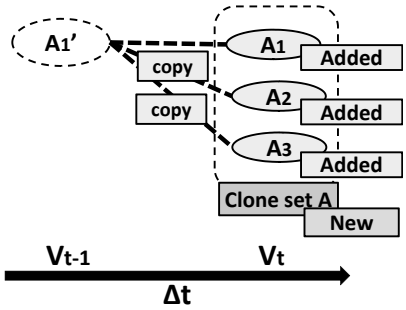
(a) Stable Clone Set



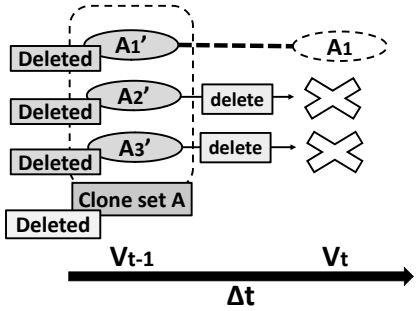
(b) Changed Clone Set ケース 1



(c) Changed Clone Set ケース 2



(d) New Clone Set



(e) Deleted Clone Set

図 4: クローンセットの分類

3.4 クローンセットの分類

本研究では，2 バージョン間に含まれる全てのクローンセットについて，その属するコードクローンの種類などに応じて，“Stable Clone Set”，“Changed Clone Set”，“New Clone Set”，“Deleted Clone Set” の 4 項目に分類している．

Stable Clone Set

V_t，V_{t-1} の 2 バージョンに渡って存在するクローンセットで，属するコードクローンが全て Stable Clone に分類されるものを Stable Clone Set と定義する．

Stable Clone Set の例を図 4(a) に示す．この例のように，2 バージョンに渡って差分が存在しないクローンセットが Stable Clone Set に分類される．

Changed Clone Set

V_t，V_{t-1} の 2 バージョンに渡って存在するクローンセットで，属するコードクローンに１つでも Modified Clone, Moved Clone, Added Clone, Deleted Clone が含まれるものを Changed Clone Set と定義する．

図 4(b) のクローンセット A には Modified Clone が含まれているため，Changed Clone Set に分類される．また，図 4(c) のクローンセット A, B のように，Moved Clone が 2 バージョンに渡って含まれるクローンセットについても Changed Clone Set に分類される．Changed Clone Set に分類されるクローンセットの情報を開発者に提供することによって，2 バージョン間でコードクローンの変更情報を知ることができる．

New Clone Set

V_t のみに存在するクローンセットを New Clone Set と定義する．

New Clone Set の例を図 4(d) に示す．この例では，C_{t-1} に含まれなかったコード片がコピーアンドペーストされたことによって，V_t で新たにクローンセット A が発生している．また，図 1 のクローンセット B のように，分岐によって発生したクローンセットも New Clone Set に分類される．

Deleted Clone Set

V_{t-1} のみに存在するクローンセットを Deleted Clone Set と定義する．

Deleted Clone Set の例を図 4(e) に示す．この例では，クローンセット A に含まれるコードクローン A_1 のクローンペアが全て削除されてしまったため，V_t ではクローンセットを形成していない．また，属するコードクローンが全て Moved Clone となり，異なるクローン

セットに含まれてしまったため，V_t では存在しなくなったクローンセットも Deleted Clone Set に分類される．

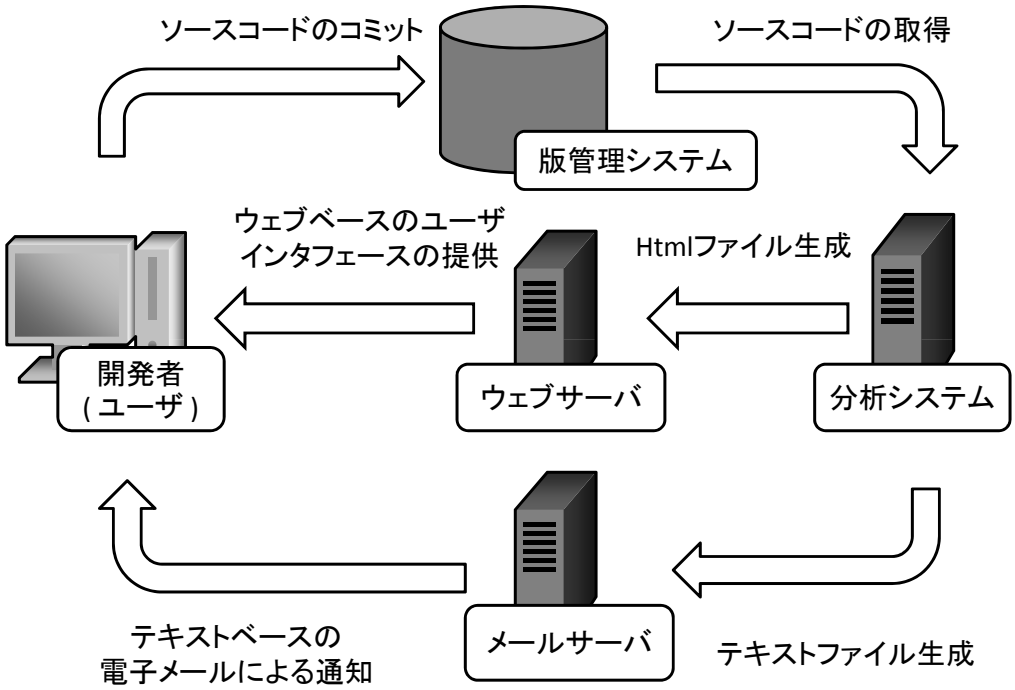


図 5: システム概要図

4 分析システム

本研究では，3 節で説明した手法でコードクローン・クローンセットの分類を行い，これらの分類結果に基づいて，2 バージョン間のコードクローン・クローンセットの変更情報をソフトウェア開発者に提供するシステムの開発を行った．そして，テキストベースの電子メールによる通知と，ウェブベースのユーザインタフェースを実現している．本節では，本システム全体の概要と，分析結果の情報提供方法について説明する．

4.1 システム概要

図 5 に本研究で開発を行ったシステムの概要図を示す．本システムでは，ソフトウェア開発に CVS [2]，Subversion [3]といった版管理システムを用いることを想定している．版管理システムとは，プロダクトの保管や変更履歴などを管理するために用いられるシステムである．これにより，過去のソースコードを参照することなどが可能となるため，複数人でソフ

[2]http://www.cvshome.org/.
[3]http://subversion.tigris.org/.

トウェアを開発するために用いられることが多い．

システムの処理の概要を以下に示す．

1. 開発者（ユーザ）がコミットした現在のバージョンのソースコード V_t を版管理システムから取得する．V_{t-1} は，過去のバージョンのソースコードを意味している．そこで，前回分析時に最新のバージョンであったソースコードを V_{t-1} として利用する．

2. 3 節で説明した手法で，V_t と V_{t-1} のソースコードに含まれるコードクローン・クローンセットの分類を行う．

3. コードクローン・クローンセットの分類結果に基づいて，2 バージョン間の変更情報を提示するために必要な html ファイルとテキストファイルの生成を行う．

本システムでは，生成したテキストファイルを添付した電子メールを送信することによって，開発者への通知を行なっている．また，情報提示に必要な html ファイルをウェブに公開されているディレクトリに出力することによって，ユーザはウェブブラウザを通してこれらの分析情報を見ることができる．テキストベース・ウェブベースの情報提示の概要については，4.2 で説明する．

4.2 分析結果の情報提示

4.2.1 テキストベースの電子メールによる通知

テキストベースの情報提示の例として，Apach Ant プロジェクト[4]の任意の 2 バージョン間に適用した場合の結果を図 6 に示す．このようなテキストファイルを電子メールに添付することによって，開発者に保守作業の対象となる可能性がある，コードクローン・クローンセットの変更情報を提示している．テキストファイルには，主に以下の情報が出力されている．

- **プロジェクト情報**

  プロジェクト情報の出力例を図 6(a) に示す．この例のように，プロジェクト情報には以下の情報を出力している．

  - **ファイル情報**

    総ファイル，追加ファイル，削除ファイル，コードクローンを含むファイルについて，それぞれのファイル数を出力している．

[4] http://ant.apache.org/.

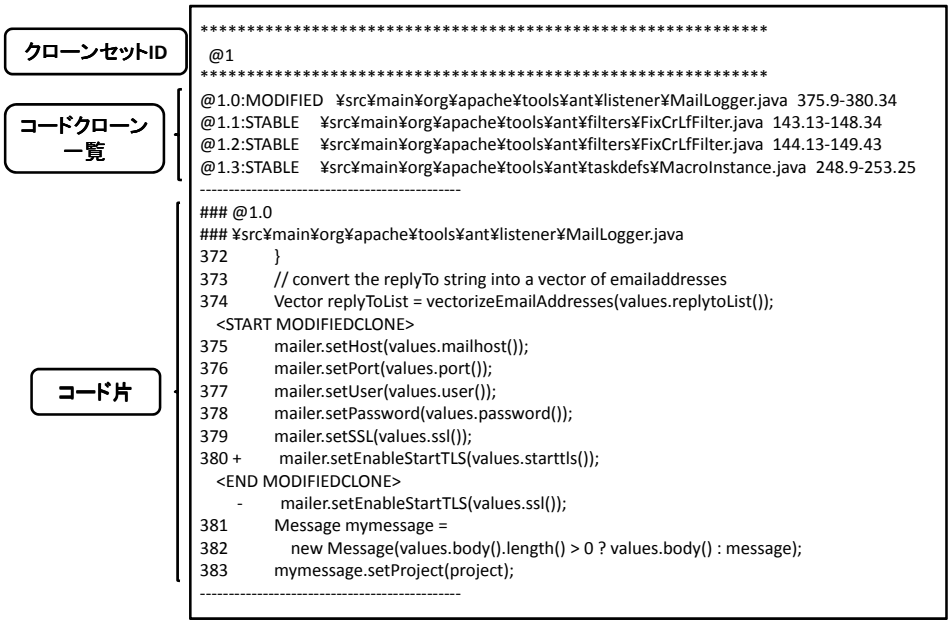
```
############################################################
  プロジェクト名:Ant
  2012/XX/XX
############################################################

【ファイル情報】
        総ファイル数:1193
        追加ファイル数:8
        削除ファイル数:1
        コードクローンを含むファイル数:506

【クローンセット分類情報 】
        Stable Clone Set数:1676
        Changed Clone Set数:29
        New Clone Set数:34
        Deleted Clone Set数:10

【コードクローン分類情報】
        Stable Clone数:5578
        Modified Clone数:21
        Moved Clone数:60
        Added Clone数:40
        Deleted Clone数:25
```

(a) プロジェクト情報



(b) クローンセット一覧

図 6: テキストベースでの情報提示の例

– **クローンセット分類情報**

Stable Clone Set, Changed Clone Set, New Clone Set, Deleted Clone Set のそれぞれに分類されたクローンセット数を出力している．

– **コードクローン分類情報**

Stable Clone, Modified Clone, Moved Clone, Added Clone, Deleted Clone のそれぞれに分類されたコードクローン数を出力している．

- **クローンセット一覧**

Changed Clone Set, New Clone Set, Deleted Clone Set についてそれぞれに含まれるクローンセットを一覧として出力している．Stable Clone Set は，2 バージョン間で変更がなかったクローンセットである．従って，Stable Clone Set の情報は省略している．図 6(b) は Changed Clone Set に含まれるクローンセットの一例を示している．この例のように，各々のクローンセットに関して以下の情報を出力している．

– **クローンセット ID**

V_t, V_{t-1} に存在する全てのクローンセットに割り当てられた ID である．

– **属するコードクローン一覧**

該当するクローンセットに属する C_t 中のコードクローン一覧を示す．ただし，Deleted Clone, Moved Clone については V_{t-1} で属していたクローンセットの一覧にも出力される．各々のコードクローンについて出力される情報を以下に示す．

* コードクローン ID
* コードクローンの分類情報
* コードクローンが存在するファイルへのパス
* コードクローンが存在するファイル中の位置（開始行・終了行）

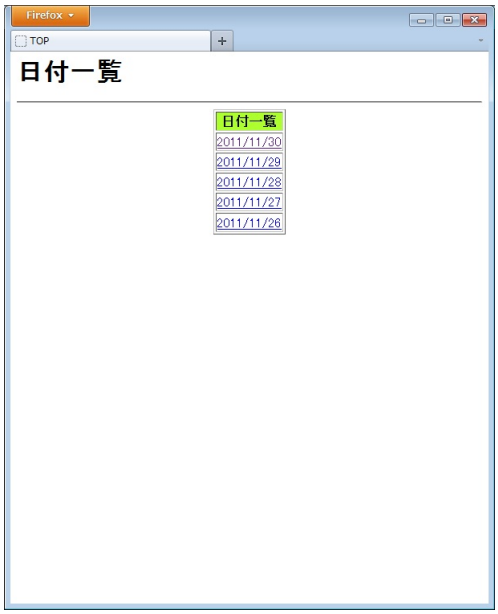
– **コードクローンのコード片**

プロダクト V_t での行番号を表示し，挿入行を “+”，削除行を “-”で表している．

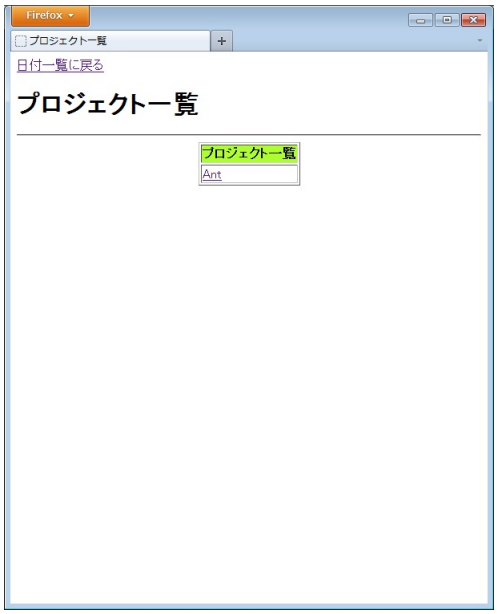
4.2.2 ウェブベースのユーザインタフェースの提供

ウェブベースの情報提示の例として，任意のウェブブラウザから見た例を図 7,8 に示す．この例のように，ウェブベースのユーザインタフェースは以下のページ群から成り立っている．

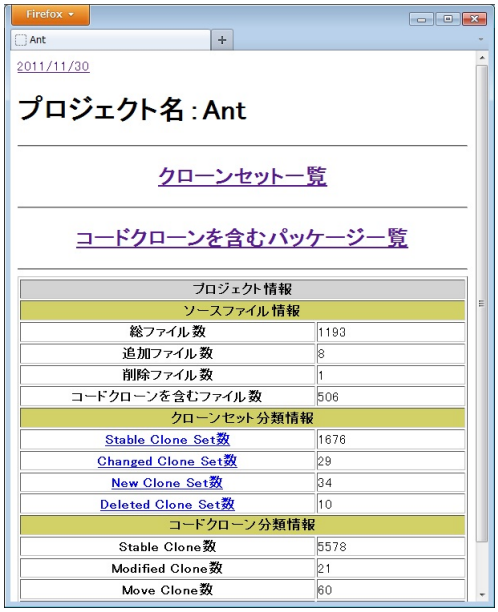
トップページ（図 7(a)）



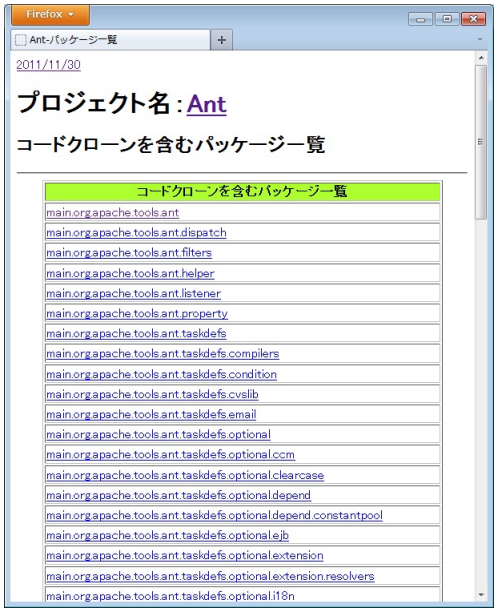
(a) トップページ



(b) プロジェクト一覧ページ

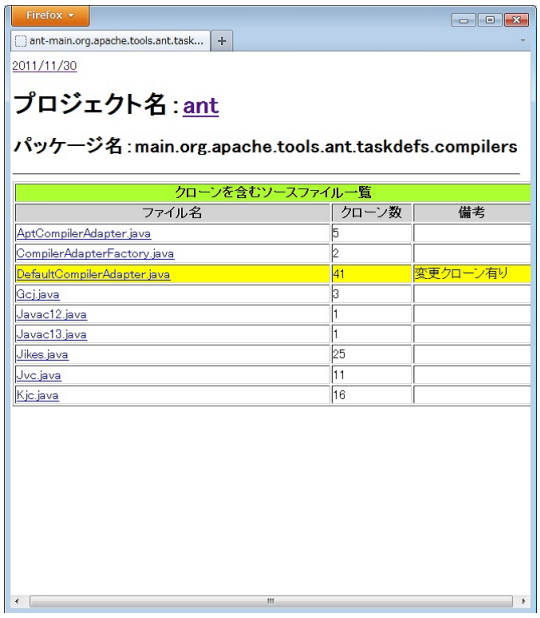


(c) プロジェクト情報ページ

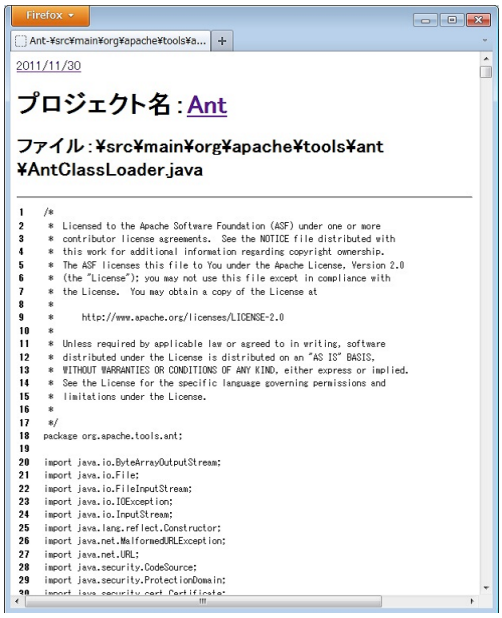


(d) パッケージ一覧ページ

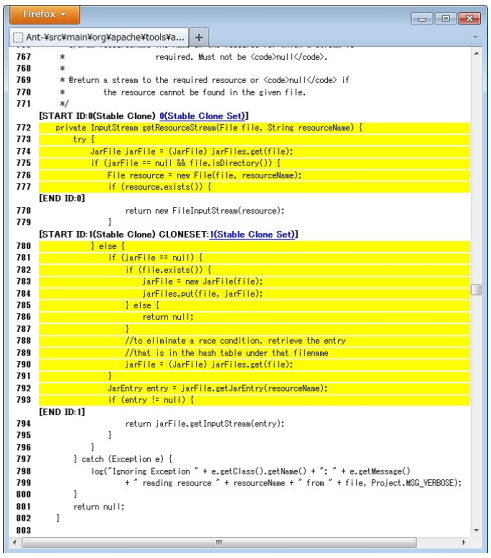
図 7: ウェブベースの情報提示の例 1



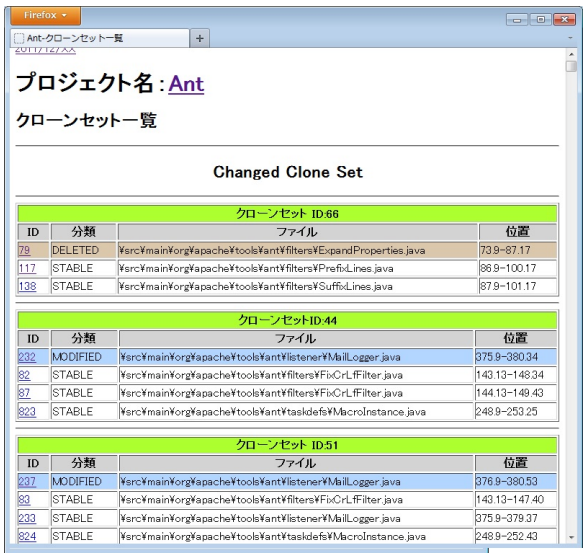
(a) ソースファイル一覧ページ



(b) ソースファイルページ 1



(c) ソースファイルページ 2



(d) クローンセット一覧ページ

図 8: ウェブベースの情報提示の例 2

システムを起動し，分析が行われた日付の一覧が表示される．任意の日付をクリックすることによって，該当日に分析を行ったプロジェクト一覧ページへ移動する．

プロジェクト一覧ページ（図 7(b)）

選択された日付に，分析が行われたプロジェクトの一覧が表示される．任意のプロジェクトをクリックすることによって，該当プロジェクトのプロジェクト情報ページへ移動する．

プロジェクト情報ページ（図 7(c)）

4.2.1 節で説明したプロジェクト情報と同様の内容が表示される．また，クローンセット一覧，コードクローンを含むパッケージ一覧をクリックすることによって，それぞれのページへ移動する．

パッケージ一覧ページ（図 7(d)）

コードクローンを含むパッケージの一覧が表示される．任意のパッケージをクリックすることによって，該当パッケージに含まれるソースファイル一覧ページへ移動する．

ソースファイル一覧ページ（図 8(a)）

選択されたパッケージに含まれるソースファイル一覧が表示される．任意のソースファイルをクリックすることによって，該当ソースファイルページへ移動する．

ソースファイルページ（図 8(b),8(c)）

選択されたソースファイルの内容が表示される．行番号は V_t に含まれるソースファイルのものを表し，V_t と V_{t-1} の間で，挿入行は “+”，削除行は “-”で表示されている．また，コードクローンとなっているコード片については図 8(c) のように，色付けすることによって見やすくしている．

クローンセット一覧ページ（図 8(d)）

4.2.1 節で説明したクローンセット一覧と同様の情報が表示される．ただし，各々のコードクローンのコード片は表示せず，任意のコードクローンの ID をクリックすることによって該当するソースファイルページへ移動する．

5　評価実験

本研究では，開発したシステムをオープンソースソフトウェアや実際の企業でのソフトウェア開発に適用することによってその有用性を評価した．本節では，それらの評価実験の内容とその結果について説明する．

5.1　オープンソースソフトウェアへの適用

5.1.1　実験内容

オープンソースソフトウェアへの適用として PostgreSQL [5] を対象に，本システムの動作実験を行った．PostgreSQL を対象とした理由を以下に示す．

- ソフトウェア開発に版管理システムを用いており，公開されている．
- C 言語で開発されており，CCFinder を利用してコードクローンを検出できる．
- 開発が頻繁に行われている．

本実験では，PostgreSQL のリポジトリの中から，ソースコードが格納されている src ディレクトリ以下のすべてのファイルを対象とした．そして，2011/11/01 を時点 $t = 0$ とし，1 週間間隔で 2011/11/08 から 2011/12/27 までの間，$t = 1, 2, ..., 8$ の計 8 時点に対して本システムを用いて分析を行った．

5.1.2　結果と考察

各分析時点での各々のカテゴリに分類されたクローンセットの数を表 2 に示す．この表から，各分析時点において大部分のクローンセットが Stable に分類されていることがわかる．Stable Clone Set は，いわば 2 バージョン間で変更されていないクローンセットである．従って，本システムを用いることによって，全てのコードクローンの中から保守作業の対象となる可能性があるコードクローン・クローンセットの変更情報を効率良く提供することが可能となる．

図 9 は，2 バージョン間で変更されたコードクローンの例である．このクローンセットは 2011/11/29 時点の分析によって検出されたものであり，V_{t-1} は 2011/11/22 時点のバージョンのソースコードを，V_t は 2011/11/29 時点のバージョンのソースコードを表している．2011/11/22 時点では，コードクローン 1,2 の 2 つからクローンセットが形成されている．しかし，2011/11/29 時点では，それらの一方が編集されたことによってクローンセットを形

[5] http://www.postgresql.jp/.

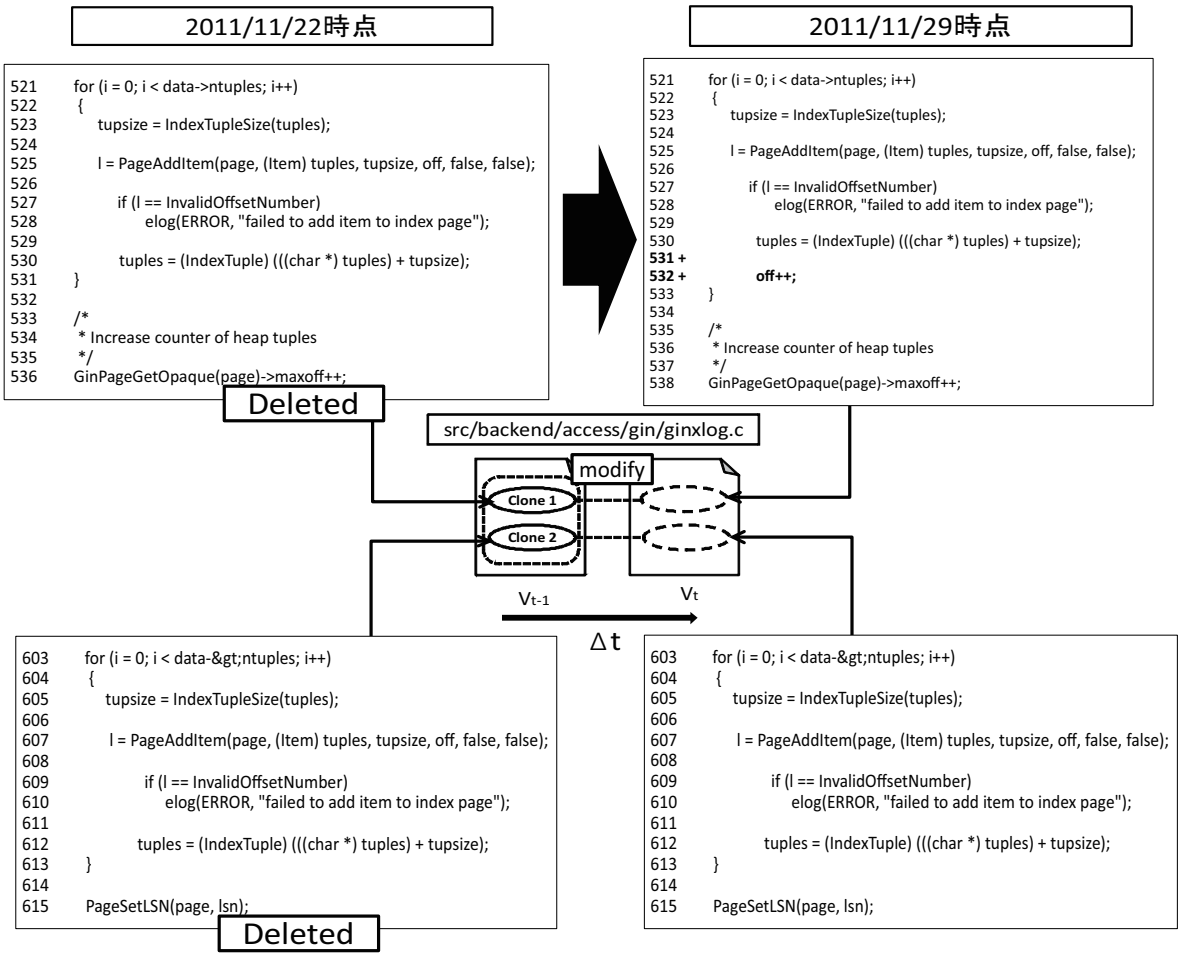


図 9: Deleted Clone Set の例

表 2: postgreSQL 適用時のクローンセットの分析結果

分析日	Stable	Changed	New	Deleted
2011/11/08	10690	434	113	1
2011/11/15	11128	102	15	3
2011/11/22	10416	687	62	62
2011/11/29	10448	687	101	30
2011/12/06	10914	211	47	34
2011/12/13	11118	39	26	29
2011/12/20	10861	149	221	49
2011/12/27	10715	364	66	122

成していないため，Deleted Clone Set に分類されている．版管理システムのコメントログによると，コードクローン 1 にバグが発見されたため，531 行目と 532 行目を挿入し修正が行われている．しかし，コードクローン 1 とクローンペアになっているコードクローン 2 は編集されておらず，2011/11/29 時点ではクローンセットの関係がなくなったため，どちらも Deleted Clone に分類されている．2011/11/29 時点のソースコードのみを用いて分析を行うだけでは，それらはコードクローンとは検出されず，全く無関係のコード片となる．もし，コードクローン 2 にも同様の不具合が含まれている場合は，コードクローン 1 と一貫した修正が必要となる可能性がある．本システムを用いることによって，このような一貫した修正が必要となる可能性があるクローンセットの情報を開発者に提供することができると考えられる．

5.2 企業のソフトウェア開発への適用

5.2.1 実験内容

本研究では，企業におけるソフトウェア開発への適用として，日本電気株式会社のウェブアプリケーションソフトの開発を対象に実験を行い，本システムの有効性の評価を行った．対象としたソフトウェアは，JAVA 言語で実装されており，ファイル数は約 350，行数は約 12 万である．適用期間は 2011/12/18 から 2012/01/06 であり，その期間の 4 時点（2011/12/19，2011/12/28，2011/12/29，2012/01/06）において分析が行われた．そして，実際に開発者にアンケートをとり，その有用性を評価した．アンケート内容を以下に示す．

Q1 システムの有用性の評価に関する質問

分析結果から同時修正，集約などの保守作業が必要となるコードクローンを確認することができたか

Q2 本システムによって発見された保守作業の対象となるコードクローン・クローンセットに関する質問

表 3: 開発現場適用時のクローンセットの分析結果

分析日	Stable	Changed	New	Deleted
2011/12/19	873	9	2	1
2011/12/28	833	34	37	14
2011/12/29	895	1	0	2
2012/01/06	895	0	0	3

(a) 保守作業が必要となるコードクローンの情報

1. 検出日
2. プロジェクト名
3. コードクローン/クローンセット ID

(b) 保守作業が必要となるコードクローン・クローンセットについて，既に存在を知っていたか否か

(c) 保守作業が必要となるコードクローン・クローンセットについて，すぐに対策を行う必要があるか否か

(d) 保守作業の方法

1. 集約：対象クローンセットをひとつのメソッドまたは，プロジェクト外のライブラリにまとめて置く
2. 同時修正：クローンセット中のコードクローンを同時に修正する
3. ドキュメント化：管理台帳にクローンセットの位置を記録する
4. コメント：ソースコードにコメントとして他のクローンセットの位置を書く
5. その他

Q1 は，本システムの有効性に関する質問であり，開発者が “YES” と答えた場合，保守作業が必要となるそれぞれのコードクローンについて Q2 に解答する．Q2(d) では，実際の保守作業の方法を 5 項目から選択する．

5.2.2 結果と考察

各分析時点での各々のカテゴリに分類されたクローンセットの数を表 3 に示す．PostgreSQL への適用時と同様に多くのクローンセットが Stable Clone Set に分類されている．従って，本システムを用いることによって，全てのコードクローンの中から保守作業の対象となる可能性があるコードクローン・クローンセットの変更情報を効率良く提供することが可能となる．

表 4: 保守作業が必要となったクローンセット

	分類	Q2(b)	Q2(c)	Q2(d)
クローンセット 1	New Clone Set	No	Yes	集約
クローンセット 2	New Clone Set	No	Yes	集約

```
for (int i = 0; i < contents.size(); i++) {
    try {
        Content content1 = contents.get(i);
        Content content2 = (Content) content1.clone();
        content2.setTitle(StringUtil.concatPath(toPath, content1.getName()),
    true);
        if (content1 instanceof Page) {
            copyPage((Page) content1,(Page) content2);
        } else if (content1 instanceof File) {
            copyFile((File) content1,(File) content2);
        } else if (content1 instanceof Category) {
            copyCategory((Category) content1,(Category) content2);
        }
    } catch (Exception e) {
        if (!(e instanceof AccessDeniedException))
            throw e;
    }
}
```

(a) コンテントのコピー機能

```
for (int i = 0; i < contents.size(); i++) {
    try {
        Content content1 = contents.get(i);
        Content content2 = (Content) content1.clone();
        content2.setTitle(StringUtil.concatPath(toPath, content1.getName()),
    true);
        if (content1 instanceof Page) {
            move Page((Page) content1,(Page) content2);
        } else if (content1 instanceof File) {
            move File((File) content1,(File) content2);
        } else if (content1 instanceof Category) {
            move Category((Category) content1,(Category) content2);
        }
    } catch (Exception e) {
        if (!(e instanceof AccessDeniedException))
            throw e;
    }
}
```

(b) コンテントの移動機能

図 10: 集約が必要となったクローンセットの例

アンケートの結果としては，Q1 に対する回答が “YES”であり，開発者は本システムを用いて保守作業が必要となる 2 つのクローンセットを確認することができた．表 4 に，開発者が本システムで発見したクローンセットの一覧を Q2 の回答と共に示す．これらのクローンセットは Q2(b) に “No”と答えていることから，本システムによって新たに確認することができたクローンセットである．

開発者が確認することができた，保守作業が必要となったクローンセットの例を図 10 に示す．このクローンセットは 2 つのコードクローンから形成され，それぞれコンテンツのコピー機能と移動機能を有しており，類似した処理を行うコード片である．Q2(c)(d) の結果より，これらは迅速に集約することによって保守作業が行われた．もう一方のクローンセットについても，同じく類似した処理を行うコード片であり，集約することによって保守作業が行われた．

このように，開発者は本システムを用いて集約の対象となる新たに発生したクローンセットを確認することができた．従って，2 バージョン間で変更されたコードクローン，および，クローンセットの情報を開発者に提供する本システムは，実際のソフトウェア開発においても有益であると言える．

6 関連研究

本研究と同様にコードクローンの履歴を調査する研究として，Kim らの研究 [15] がある．Kim らは，クローンセットの履歴に対して，以下のようなモデルを定義した．

- Same: 属する全てのコードクローンに変化がない．
- Add: 1 つ以上のコードクローンが追加されている．
- Substracet: 1 つ以上のコードクローンが削除されている．
- Consistent Change: 全てのコードクローンに一貫した修正が加わっている．
- Inconsistent Change: 一部のコードクローンに一貫した修正が加わっている．
- Shift: 一つ以上のコードクローンの位置が変化している．

そして，これらの作成したモデルを長期間に渡って分析することによって，コードクローンの存在する期間とその特徴の関係について調査を行っている．

Kim らの研究では，分析の対象になっているのはクローンセットのみであるが，本研究ではコードクローンの変化に基づいても詳細に分析を行なっている．また，本研究はソフトウェア開発者への保守作業の対象となるコードクローン・クローンセットの変更情報の提供を目的としている．そのため，隣り合う 2 バージョンのみソースコードに対して分析を行なっており，実際に企業で行われているソフトウェア開発に適用するためのシステムの開発を行い，その有用性を評価している．

7　まとめと今後の課題

本研究では 2 バージョンでのソースコードの編集に基づいてコードクローン・クローンセットを分類する手法を提案した．また，分類結果に基き，コードクローン・クローンセットの変更情報を開発者に提供するためのシステムの開発を行った．そして，オープンソースソフトウェアや実際の企業で行われているソフトウェア開発に適用することによってその有用性を確かめることができた．

しかし，開発現場への適用期間は短く，本システムの有効性をさらに評価するためには長期間に渡る適用と，ユーザからのフィードバックが必要である．また，本システムでは CCFinder を利用したが，他のコードクローン検出ツールを利用することによってまた異なる結果が得られるかもしれない．

また，現状の開発したシステムは改良する点が多い．例えば，本手法では隣り合う 2 バージョン全体に CCFinder を適用しているため，計算量が大きい．また，ウェブベースのユーザへの提示については，図 8(c) のようにタグを用いてコードクローンを表している．しかし，多くのコードクローンがオーバーラッピングして存在する場合，そのようなタグが重複して出力されるため，出力が見難くなることがある．このように，ユーザインタフェースについても，さらに改良を行なっていく必要があると考えられる．今後，このようなシステムの問題点について開発者からのフィードバックを参考に改善を行い，実際のソフトウェア開発現場で貢献できるシステムを構築していく必要がある．

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 助教に深く感謝いたします.

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奈良先端科学技術大学院大学情報科学研究科ソフトウェア設計学講座 吉田則裕 助教に深く感謝いたします.

本研究において，適時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 眞鍋雄貴 特任助教に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 崔恩瀞 氏に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日本電気株式会社 三橋二彩子 氏，佐野建樹 氏に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日本電気株式会社の皆様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いたします.

参考文献

[1] 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.

[2] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transaction Software Engineering*, Vol. 28, No. 1, pp. 654–670, 2002.

[3] C. Kapser and M. W. Godfrey. Cloning considered harmful considered harmful. *Proc. Working Conference on Reverse Engineering (WCRE 2006)*, pp. 19–28, 2006.

[4] I. Baxter, A. Yahin, L. Moura, M. Anna, and L. Bier. Clone using abstract syntax trees. *Proc. International Conference on Software Maintenance '98*, pp. 368–378, 1998.

[5] S. Uchida, A. Monden, N. Ohsugi, T. Kamiya, K. Matsumoto, and H. Kudo. Clone using abstract syntax trees. *Journal of Computer Information Systems*, Vol. XLN, No. 3, pp. 1–11, 2005.

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

[7] J.H. Johnson and E. Merlo. Substring matching for clone detection and change tracking. *Proc. International Conference on Software Maintenance '94*, pp. 120–126, 1994.

[8] 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.

[9] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transaction Software Engineering*, Vol. 32, No. 3, pp. 176–192, 2006.

[10] H.A. Basit and S. Jarzabek. Detecting higher-level similarity patterns in programs. *Proc. ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 156–165, 2005.

[11] J. Mayrand, C. Leblanc, and E. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. *Proc. International Conference on Software Maintenance '96*, pp. 244–253, 1996.

[12] K. Kontogiannis, R. DeMori, E. Merlo, M. Galler, and M.Bernstein. Experiment on the automatic detection of function clones in a software system using metrics. *Automated Software Engineering*, Vol. 3, pp. 77–108, 1996.

[13] 門田暁人, 佐藤慎一, 神谷年洋, 松本健一. コードクローンに基づくレガシーソフトウェアの品質の分析. 情報処理学会論文誌, Vol. 44, No. 8, pp. 2178–2188, 2003.

[14] 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, Vol. J89-D, No. 10, pp. 2279–2287, 2006.

[15] M. Kim, V. Sazawal, D. Notkin, and Gail C. Murphy. An empirical study of code clone genealogies. *Proc. Joint meetings of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE) '05*, pp. 187–196, 2005.