修士学位論文

題目

オープンソースソフトウェアの進化におけるコードクローンと障害数の変化の調査

指導教員
井上 克郎 教授

報告者
政井 智雄

平成 24 年 2 月 7 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 ソフトウェア工学講座

平成 23 年度 修士学位論文

オープンソースソフトウェアの進化におけるコードクローンと障害数の変化の調査

政井 智雄

内容梗概

ソフトウェアの保守を困難にしている要因の 1 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同一，または類似したコード片のことであり，生成される原因の 1 つとしてコピーアンドペーストを用いた開発作業が挙げられる．例えばあるコード片に欠陥が含まれている場合，そのコードクローン全てに対し修正を検討する必要が生じる．近年，ソフトウェアが大規模になっており，このような作業には大きなコストがかかる．また一般的にコードクローンは，ソフトウェアにおいて欠陥の原因となりうると言われ，積極的に除去すべきという意見が強い．

本研究では，コードクローンと欠陥の関係について具体的な結果を取得するために，オープンソースソフトウェアである FreeBSD に対し，バージョン毎におけるコードクローン，障害報告についての調査を行った．その結果，大規模改修を行った結果，クローン率や障害報告が急上昇したと考えられるリリースを特定した．また，類似したアーキテクチャのためのコンポーネントはクローンの割合が高いことや，メジャーリリースにおいて，大量の障害が発生していることを確認した．

主な用語

コードクローン
FreeBSD
ソフトウェア保守
障害報告

目 次

1　まえがき

ソフトウェアの保守作業を困難にしている要因の１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，ソースコード中の存在する同一もしくは類似したコード片のことを指す．コードクローンは，コピーアンドペーストなどが原因で生じる．例えばあるコード片に欠陥が含まれている場合，そのコードクローン全てに対し修正を検討する必要が生じる．近年，ソフトウェアが大規模になっており，このような作業には大きなコストがかかる．

これまでに，コードクローン検出手法が数多く提案されており，また，それら手法を用いた研究も行われている．代表的なものとして，オープンソースソフトウェアに含まれるコードクローンの分析を行った研究がある．例えば，Linux に含まれるコードクローンの量的変化や，Linux カーネル中のドライバに含まれるコードクローンの分析を行なっている．

本研究では，オープンソースソフトウェアのオペレーティングシステムとして，Linux とともに代表的である FreeBSD のソースコードに含まれるコードクローンの変化を分析する．あわせて，FreeBSD の障害報告の量的変化を分析する．著者の知る限り，コードクローンの変化と障害報告の変化を分析した研究はなく，また FreeBSD は広く使われているオープンソースソフトウェアであるにもかかわらず，そのコードクローンの変化に関する分析が行われていなかった．本研究では，FreeBSD のソースコードをコンポーネント単位に分割し，クローン率（コンポーネントに含まれるソースファイル中に含まれるコードクローンの割合）や，コンポーネントに対して行われた障害報告の数をリリースバージョンごとに計測する．そして，各コンポーネントに対して，クローン率や障害報告の数の変化を分析する．

分析の結果，以下の考察が得られた．

- 大規模改修を行った結果，クローン率や障害報告が急上昇したと考えられるリリースの存在
- 類似したアーキテクチャのためのコンポーネントは，クローン率が高いこと
- 他のコンポーネントにコードクローンが吸収されたため，クローン率が低下したコンポーネントの存在
- メジャーリリースにおいて，大量の障害が発生
- バージョン 5 系列は，当初障害報告が収束せず，バージョン 5.3.0 まで安定版がリリースされなかったこと

以降，2 節では，本研究の背景として，コードクローンや版管理システム，対象とするオペレーティングシステムである FreeBSD について述べる．次に 3 節で，本研究で用いた調

査手法について説明し，4 節では調査手法を FreeBSD に適用することで得られた結果について考察する．5 節では，本研究の関連研究をコードクローンの変化を分析した研究を中心に紹介する．最後に，6 節ではむすびとして，本研究のまとめと今後課題について述べる．

2　背景

本研究の背景として，問題となるコードクローンとその検出ツールである CCFinder，およびバージョン管理システム，オープンソースソフトウェアである FreeBSD について説明する．

2.1　コードクローン

コードクローンとは，ソースコードの中に存在する同一，または類似したコード片のことである．コードクローンが存在するプログラムでは，欠陥が見つかったコード片を修正する場合，同様の欠陥はこのコード片と同一，類似したコード片にも含まれるため，そのコード片のコードクローン全てに対して修正を検討しなければならない．

コードクローンが発生する原因として以下の項目が挙げられる [3, 14, 25]．

既存コードのコピーアンドペーストによる再利用　オブジェクト指向設計などのソフトウェア設計手法を利用することで，クラスやメソッド，関数などといったソフトウェア部品の再利用が可能となる．しかし，正常な動作が確認されている既存のソースコードを流用し，部分的に変更を加える方が，一からコーディングを行うよりも信頼性が高いため，コピーアンドペーストを用いた場当たり的な既存コードの再利用が多く存在する．

定型処理　キューの挿入処理やデータ構造へのアクセス処理など，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．

プログラミング言語の仕様　プログラミング言語の仕様によりある処理を 1 箇所にまとめて記述することが出来ない場合，その処理を行うコードがプログラム中の複数個所に記載されることがある．例として，プログラミング言語 Java の場合，例外処理は 1 箇所にまとめて記述することはできないため，複数個所に同様の処理を記述する必要がある．

パフォーマンスの改善　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．

コード生成ツールにより自動生成されたコード　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，識別子等を除いて類似したコードが生成される．

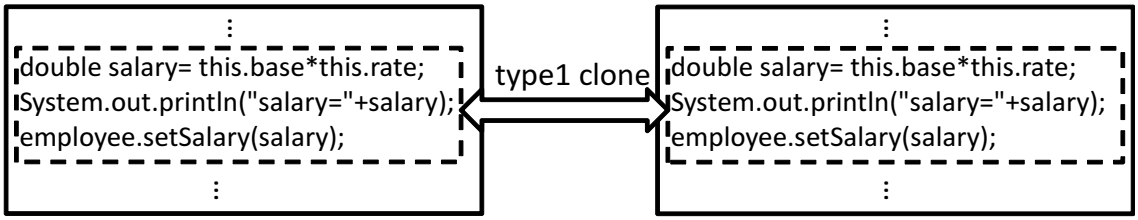

図 1: タイプ 1 のコードクローンの例

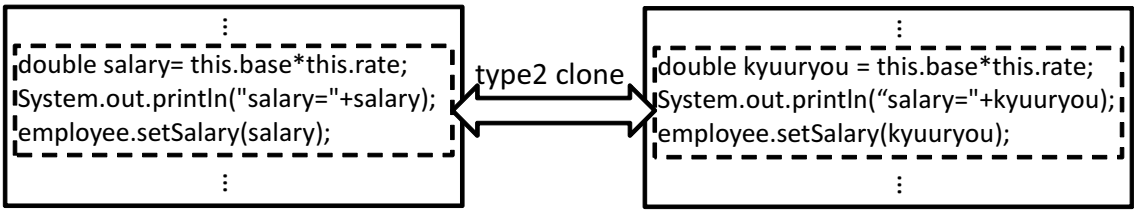

図 2: タイプ 2 のコードクローンの例

複数のプラットフォームに対応したコード 複数の OS（Windows，Macintosh，Linux，FreeBSD など）や CPU（i386 系，amd64 系など）に対応したソフトウェアは，各プラットホ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．

また，Bellon は，コードクローン間の相違の度合いに基づいた以下の 3 つの分類を定義している [5].

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いて，完全に一致するコードクローン（図 1）．

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（図 2）．

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，変更が行なわれたコードクローン（図 3）．

コードクローンはソースコードに対して一貫した変更を加えることを難しくするため，ソフトウェアの保守効率を低下させると言われている [9]．そのため，次に紹介するコードクローン検出手法を用いてコードクローンを検出し，コードクローンを統合する手法が現在までに提案されている [30]．その一方，コードクローンが欠陥修正に及ぼす影響を調べ，コードクローンはソースコードの欠陥修正とは影響しないという結果を報告した事例も存在する [17].

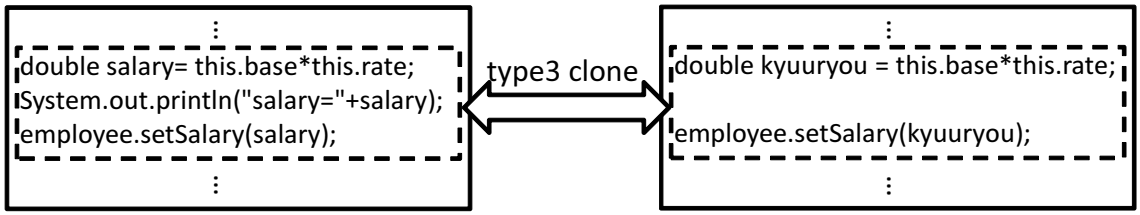

図 3: タイプ 3 のコードクローンの例

コードクローンを検出するため，これまでに数多くのコードクローン検出ツールが提案されている．既存の検出手法はコードクローンを検出する際に用いるデータによって，大まかに以下の通りに分類することが出来る．

- 文字列を用いた検出 [7]
- 字句（トークン）列を用いた検出 [15]
- 抽象構文木を用いた検出 [3, 8, 13, 20, 28]
- プログラム依存グラフを用いた検出 [12, 18, 19]
- メトリクスを用いた検出 [23]

これらの検出ツールのうち，トークン単位でコードクローンを検出するツールとして CCFinder[15] がある．CCFinder は，与えられたソースファイル集合からコードクローンを検出し，コードクローンの位置情報，対応関係を出力する．CCFinder の持つ主な特徴は次のとおりである．

細粒度のコードクローンを検出　字句解析を行うことにより，トークン単位でのコードクローンを検出する．

大規模ソフトウェアを実用的な時間とメモリで解析可能　例えば 10MLOC のソースコードを 68 分 (実行環境 Pentium3 650MHz RAM 1GB) で解析可能である [29].

様々にプログラミング言語に対応可能　言語依存部分を取り替えることで，様々なプログラミング言語に対応できる．現在は，C/C++，Java，COBOL/COBOLS，Fortran，Emacs Lisp に対応している．またプレーンテキストに対しても，分かち書きされた文章として解析可能となっており，未対応の言語に対しても完全一致判定によるコードクローンはとることができる．

実用的に意味を持たないコードクローンを取り除く

- コードクローンは小さくなればなるほど偶然の一致の可能性が高くなるが，最小一致トークン数を指定することができるため，そのようなコードクローンを検出しないようにできる．
- モジュールの区切りを認識することで，複数のモジュールをまたがっているコードクローンを取り除いている．

ある程度の違いは吸収可能

- ソースコード中に含まれるユーザ定義名，定数をパラメータ化することで，その違いを吸収できる．
- クラススコープや名前空間による複雑な名前の正規化を行うことで，その違いを吸収できる．
- その他，テーブル初期化コード，可視性キーワード (protected, public，private 等)，コンパウンド・ブロックの中括弧表記等の違いも吸収することができる．

CCFinder のコードクローン検出手順（ソースコードを読み込んで，クローンペア情報を出力する）は大きく四つの過程から成り立っている．

ステップ 1（字句解析） ソースファイルを字句解析することによりトークン列に変換する．入力ファイルが複数の場合には，個々のファイルから得られたトークン列を連結し，単一のトークン列を生成する．

ステップ 2（変換処理） 実用上意味を持たないコードクローンを取り除くこと，及び，些細な表現上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，この変換により変数名は同一のトークンに置換されるので，変数名が付け替えられたコード片もコードクローンであると判定することができる．

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全て検出する．

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．

2.2 版管理システム

ソフトウェアの開発・保守を効率的に行うために，大規模な開発や複数人での開発においては，多くの場合に版管理システムが用いられている．版管理システムは，ソースコードおよび関係する全てのファイルの作成日時，変更日時，変更箇所，作業者などの履歴を保管することができ，その他多様な機能が版管理ステムそれぞれに実装されている．
版管理システムにはCVS，Subversion（以降SVNと呼ぶ），Gitなどが挙げられる．本論文ではSVN，CVSに関係するもののみ説明する．以下に，版管理システムにおける主に用いられる単語について説明する．

コミット　開発者が，ソフトウェアの更新を行うために，ファイルの変更をリポジトリに反映させ，また必要に応じてコメントを残す操作を指す．この操作を行う際には，後に説明するリビジョン番号がリポジトリに保存される．また開発者は，リポジトリにこれまでに行われたコミットすべての記録を取得することができ，以降この記録をコミットログと呼ぶ．

チェックアウト　開発者が自身の開発環境に，リポジトリに保管されているソフトウェアを取り出す操作を指す．この際，日時やリビジョン番号を指定することで，最新のものだけでなく，開発途中のファイルを取り出すことも可能である．

リビジョン番号　コミットが行われるたびに，そのコミットが行われた直後の状態を一意に特定できるように設定される番号のことを指す．このリビジョン番号の扱いは版管理システム毎に異なる場合があり，CVSにおいては各ファイル毎，SVNにおいては各操作毎に保管される．

ブランチ　開発において，リポジトリを分岐（コピーを作成し，それぞれを別の開発のためのリポジトリとして扱う）する機能を用いて，行われるそれぞれの開発の流れを開発ブランチ，または単にブランチと呼ぶ．同じ状態のソフトウェアから異なる開発を並行して行うことができ，大規模なソフトウェアの開発において開発版と安定版の方向性を分岐する際などに用いられる．

2.3 FreeBSD

FreeBSD は，1993 年に最初のリリース (1.0) が行われた，いわゆる BSD UNIX 系オペレーティングシステムの１つである．Intel x86 を始めとする様々なアーキテクチャの計算機で動作する．FreeBSD のソースコードには，システムの基本的な部分であるカーネル，コ

図 4: FreeBSD のブランチおよびリリースの関係

ンパイラ，各種ツール群などがすべて含まれており，信頼性が高く保たれている大規模システムの１つとして知られている．

FreeBSD には２つの開発ブランチ，CURRENT と STABLE が存在する．CURRENT は，FreeBSD の最新のソースコードに対応しており，開発途上のソフトウェアや，実験的に実装された機能などが含まれている．STABLE は，一般ユーザに公開されるリリースを開発するためのブランチで，CURRENT で試験された機能が取り込まれる．STABLE での開発は CURRENT と並行して行われており，その実装が進行すると，さらに RELENG（RELease ENGineering）というブランチに分岐する．RELENG でリリース用の作業が行われることにより，最終的な RELEASE が作成される．CURRENT を除くブランチはバージョン番号を持っており，STABLE にはメジャーバージョンが，RELENG にはメジャーバージョンとマイナーバージョンが付与され，RELEASE に最終的なリビジョン番号が与えられる．ブランチの関係の例を図 4 に示す．この図では，CURRENT ブランチのある時点のソースコードから 8-STABLE ブランチが分岐しており，8-STABLE のある時点でのバージョンから 8.0-RELENG ブランチが，またさらに開発が進行した時点で 8.1-RELENG，8.2-RELENG ブランチが分岐していることが示されている．そして，8.0-RELENG のある時点でのソースコードから 8.0.0-RELEASE が作成され，8.1-RELENG のある時点でのソースコードから 8.1.0-RELEASE が作成されている．

FreeBSD のカーネルのあるバージョンのソースコードを取り出すと，その構成要素であるソースファイルがディレクトリごとに整理された状態となっている．ディレクトリ名には様々なものがあるが，格納されるソースコードの特性から，3 つのコンポーネントに分類できる．本研究では，個別のディレクトリ単位に加えて，これらコンポーネント単位での分析を行う．

- CPU のアーキテクチャに固有のソースコードを格納するディレクトリ．ディレクトリ名はそれぞれアーキテクチャに対応している．該当するディレクトリは，`alpha`，`amd64`，

arm，i386，ia64，mips，pc98，powerpc，sparc64，sun4v，x86，xen

である．

- 計算機に接続される各種デバイスを駆動するためのドライバを格納するディレクトリ．デバイスの接続方式に対応する名前が付けられている．該当するディレクトリは，cam，isa，ofed，pccard，pci，i4b，dev である．
- その他のカーネルの構成要素．たとえばファイルシステムや，ネットワークプロトコルの処理など，様々な重要な機能を実装したソースコードが存在する．

FreeBSD では，開発者，テスタやユーザからの障害報告（Problem Report）をオンラインで管理するシステムが採用されている．障害報告は，問題の発生状況や観測された振舞いを説明した文章に，障害報告を識別するための番号（PR Number），問題の深刻度，障害に関係する CPU アーキテクチャやコンポーネントなど，障害の分類のための情報を付与したものとなっている．障害報告データベースは，開発者にとって重要な情報源であると同時に，過去のバグ修正などの活動を記録した情報でもある．

3　調査手法

調査にあたって，以下の手順で必要なデータを取得した．

1. ソースコード，コミットログの取得
2. 開発期間の決定
3. コミットの分類
4. 各メトリクス値の取得

それぞれについて説明する．

3.1　ソースコード，コミットログの取得

本研究の調査において，データを取得する元となる FreeBSD の各バージョン，ブランチにおけるソースコード，およびコミットログを取得する．現在，FreeBSD の開発プロジェクトでは，各リリースバージョンおよびブランチは版管理システム SVN を用いて管理されている．FreeBSD プロジェクトにおいて，ソースコードは SVN 上で以下のように分類され管理されている．

head　最新バージョンの FreeBSD の開発ブランチである CURRENT ブランチが管理されている．

stable　STABLE ブランチが管理されている．

releng　RELENG ブランチが管理されている．

release　各バージョンのリリース時点での状態が管理されている．どのブランチにおいてリリースされたのか等は，ここでは区別されていない．
一例としては図 7 を参照のこと．このように管理されているそれぞれのソースコードを“ svn checkout ”コマンド，コミットログを“ svn log -v ”コマンドを用いることで取得した．

3.2　開発期間の決定

開発期間の決定は，スクリプトファイル `newvers.sh`（`src/sys/conf` に存在している）に対する修正を行った FreeBSD プロジェクトの CVS 上でのコミットを追跡し，決定する．`newvers.sh` は，カーネルの現バージョンおよびブランチを表す文字列を出力するシェルス

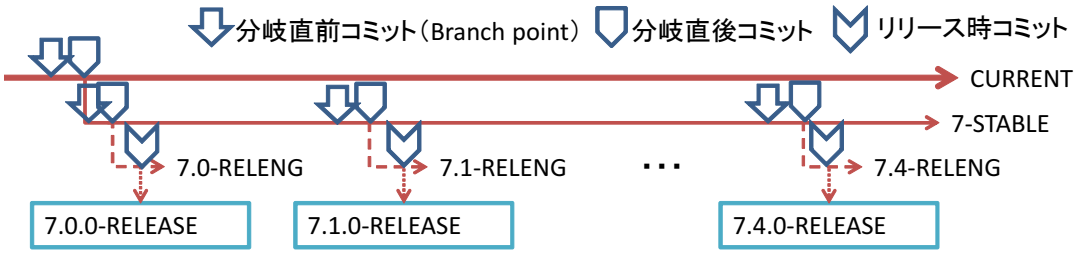

図 5: newvers.sh に対するコミット

クリプトであり，バージョンやブランチの情報において信頼できるファイルの一つである．このファイルに対する修正コミットに注目することで，ブランチ分岐直前の時点，ブランチ分岐直後の時点，リリース時点を特定でき，それらの日時を用いることにより本研究における開発期間を決定した．SVN のコミットログのみを用いて開発期間を決定しない理由としては，当初 CVS を用いて管理が行われていた FreeBSD プロジェクトにおいて，メジャーバージョン 7 の開発を境に SVN を用いた開発管理に移行したことがあげられる．CVS におけるブランチ分岐の情報を，SVN において複数のコミットを用いて登録しているが，CVS におけるブランチ分岐の正確な時間を特定することは非常に困難なため，行われたコミットに記載される日時は予測されるブランチ分岐の日時と極端に異なる場合があり，これらのコミットを用いた開発期間の決定は適切ではないと判断した．図 5 のように開発期間に関係した `newvers.sh` の修正を行うコミットは行われている．本研究においては，分岐タイミング `Branch point` としてに設定されたコミットの日時を基準に，その直後に行われたコミットの日時をブランチ分岐が行われた日時とし，以下のようにそれぞれの期間を決定する．

バージョン毎の全体の開発期間

開発開始時点からリリース時点までをバージョン毎の全体の開発期間とする．開発開始時点は，図 6 のように各バージョンのリリース時点からブランチ分岐を含めて遡った，直前のリリース時点または直前のバージョンのリリース時点が属するブランチのブランチ分岐直後のコミット時点とする．

各ブランチ内での開発期間

図 7 のように，各バージョンのリリース時点から遡った結果，通過したブランチに開発期間を有しているとする．各ブランチにおける開発期間は，開発開始時点またはそのブランチの分岐直後の時点から始まり，リリース時点またはリリース時点の属するブランチへの分岐

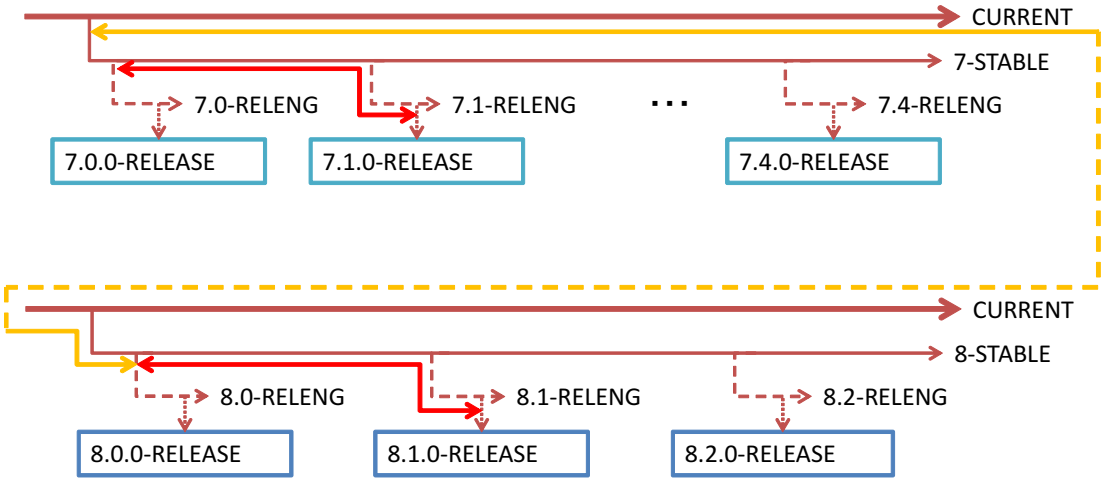

図 6: 開発期間の特定

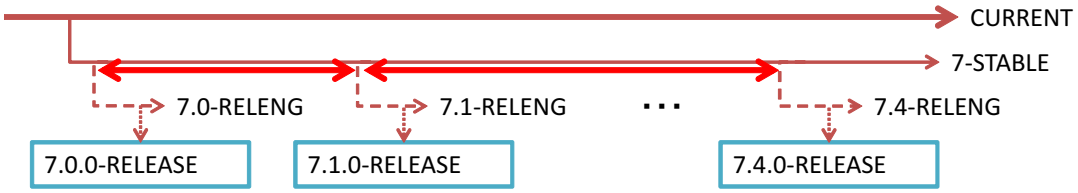

図 7: ブランチ内での開発期間の特定

直後の時点までとする．

3.3 コミットの分類

各バージョンおよび，CURRENT ブランチ，STABLE ブランチのコミットログを分析し，全てのコミットの情報をそれぞれ抽出し，コミットをバージョン毎に分類する．この際，図 8 のように，障害報告に関するコミット以外は分類せず無視する．

CURRENT ブランチおよび STABLE ブランチは，節 3.2 で示した各ブランチ内での開発期間を元に，各バージョンの開発期間に登録日が該当しているコミットを分類する．この時，STABLE ブランチには，分岐前の CURRENT ブランチにおけるコミットの情報が含まれているため，事前に CURRENT ブランチにて分類が完了したコミットに関してはカウントせず，重複を避ける．

```
------------------------------------------------------------------------
r198984 | trasz | 2009-11-06 20:29:10 +0900 (Fri, 06 Nov 2009) | 11 lines
Changed paths:
  M /releng/8.0/sys
  M /releng/8.0/sys/amd64/include/xen
  M /releng/8.0/sys/cddl/contrib/opensolaris
  M /releng/8.0/sys/contrib/dev/acpica
  M /releng/8.0/sys/contrib/pf
  M /releng/8.0/sys/dev/xen/xenpci
  M /releng/8.0/sys/kern/vfs_acl.c

MFC r197789:

Fix ACL support on sparc64.  Turns out that fuword(9) fetches 64 bits
instead of sizeof(int), and on sparc64 that resulted in fetching wrong
value for acl_maxcnt, which in turn caused __acl_get_link(2) to fail
with EINVAL.

PR:             sparc64/139304
Submitted by:   Dmitry Afanasiev <KOT at MATPOCKuH.Ru>
Approved by:    re (kib)

------------------------------------------------------------------------
```

図 8: コミットログにおけるコミットの記述の一例

3.4 計測データの取得

取得する計測データとしては以下のものが挙げられる．

1. コードクローン率
2. 障害報告数
3. 開発日数
4. 行数

それぞれの取得方法を以下に述べる．

コードクローン率

コードクローン検出ツール CCFinder を用いて，各リリースバージョンにおけるソースコードから，トークンベースによるコードクローンを検出する．CCFinder の出力結果から，全ソースコードのトークン数の算出，及びクローンに含まれるトークンの特定を行い，各コンポーネント毎にクローン率を，コンポーネント内の総トークン数に対するクローンに含まれるトークン数の割合として算出する．
また，クローン率の算出方法として，全ソースコードからクローンを検出し，その後コンポーネント毎に分割し算出するという方法と，各コンポーネントのソースコード毎にクローンを検出し，それぞれにおいてクローン率を算出するという 2 つの方法が考えられる．これらの違いは全体的か局所的かという違いだが，これらを比較することで，前者のクローン率にのみ増減が起きた場合はコンポーネント外部において類似したコード片が増減したと判断でき，後者のクローン率にのみ増減が起きた場合は，コンポーネント内部のみにおいて類似したコード片が増減したと判断できる．故に，両値を確認することで大規模なコードのコピーなどが見つけやすいなどのメリットがある．

開発日数

各バージョンの特定した開発期間を元に，開発開始時点の日付から，リリース時点までの日付の日数（両日を含む）を開発日数として算出する．

障害報告数

各バージョンごとに分類されたコミットの記述内容を分析し，障害報告 ID を特定する．各コミットの情報には，リビジョン番号，担当者，日付，編集ファイル，及びコメント等が記載されている．この内，コメントの記述中，またはその後の記述において障害報告を指す障害報告番号を検出した場合（図 8 において赤字で表示，記述の形式は複数存在している），障害報告に関するコミットとして，編集ファイル中にてソースファイルを編集しているコンポーネントと障害報告番号を記憶し，各コンポーネントの障害報告数にそれぞれを加算することで対応付けを行う．この時，障害報告管理システムから取得した障害報告と照合し，重要とされる（重要度が `critical`，`serious` に設定されている）障害報告の番号に該当することを確認し，該当しない場合は，重要でない障害報告として除外する．

行数

C 言語（C 及び C++）で記述されたソースコードを対象に，その総行数を各バージョンごとに取得する．ここで対象としているファイルは，コードクローンの取得時に対象として

いるファイルと同一である.

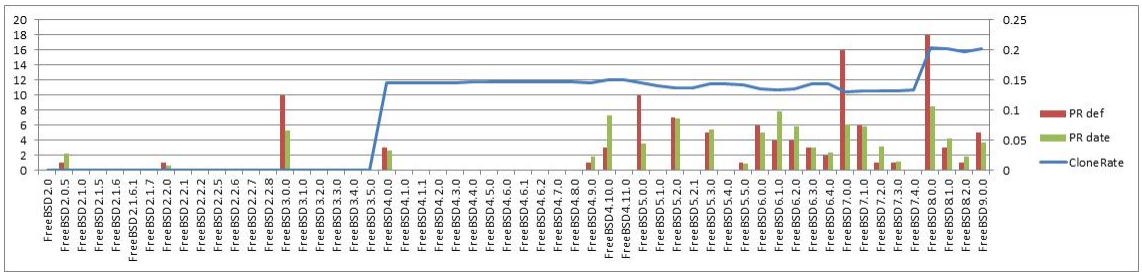

図 9: fs コンポーネント

4 調査結果と考察

図 9，図 12 に，それぞれ fs コンポーネントおよび isa コンポーネントのクローン率の変化と障害報告数の変化を示す．また，図 10 と図 11 に，それぞれ i386 コンポーネントと amd64 コンポーネントのクローン率の変化と総行数の変化を示す．図 13 は，FreeBSD 全体の障害報告数の変化を示している．

クローン率が急上昇したリリースについて バージョン 8.0.0 および 9.0.0 において，クローン率が上昇したコンポーネントが数多く存在した（図 8）．バージョン 8.0.0 および 9.0.0 は，大規模改修が行われたバージョンであり，大規模改修が原因でコードクローンが大量に発生したと考えられる．また，クローン率が急上昇すると同時に障害報告が増加したコンポーネントが存在した（図 9）．これは，品質の低いコードが増加したことで，障害が増加したためであると考えられる．

クローン率の高いコンポーネントについて 特定 CPU のためのコードを含むコンポーネントについては，互いのクローン率が高い傾向が確認できた（図 10, 図 11）．これは，類似したコンポーネントが，コードクローンを共有してるためであると考えられる．また，isa コンポーネントについては，バージョン 5.0.0 以降，クローン率が急激に減少した（図 12）．isa コンポーネントは，元々p98 コンポーネントとコードクローンを共有していたが，コードクローンの部分が pc98 コンポーネントに集約されたことが原因であると考えられ，実際にクローン率の高かったファイルが全て，pc98 コンポーネントにのみ残るよう集約されていた．

障害の多いリリースについて 7.0.0 や 8.0.0 等のメジャーリリースでは，大量の障害が報告されている（図 13）．これは，他のリリースと比べて改修の規模が大きいからであると考えられる．バージョン 5 系列については，メジャーリリース以外でも障害が大量に報告されていた．これは，5 系列は当初障害報告が収束せず，5.3.0 まで安定版がリ

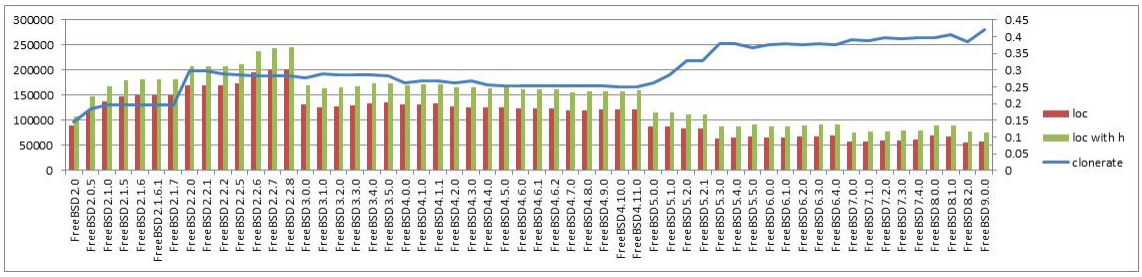

図 10: i386 コンポーネント

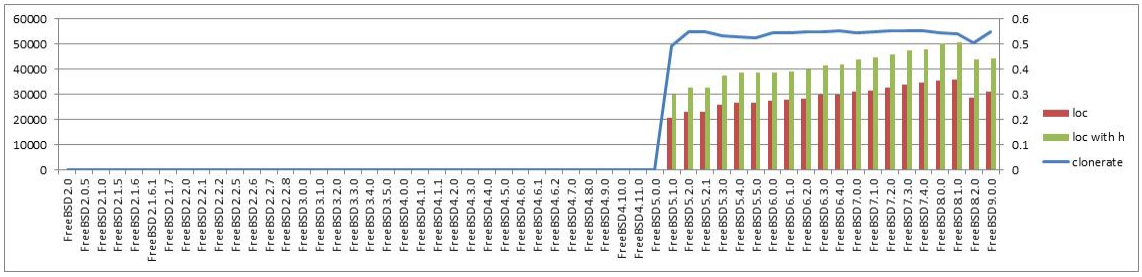

図 11: amd64 コンポーネント

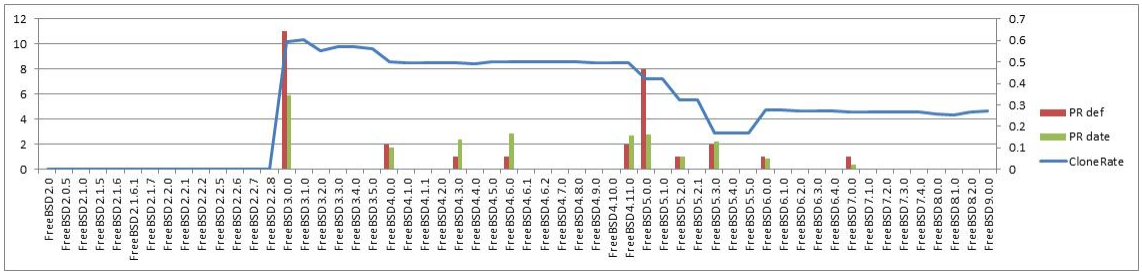

図 12: isa コンポーネント

リースされなかったことが原因であると考えられる.

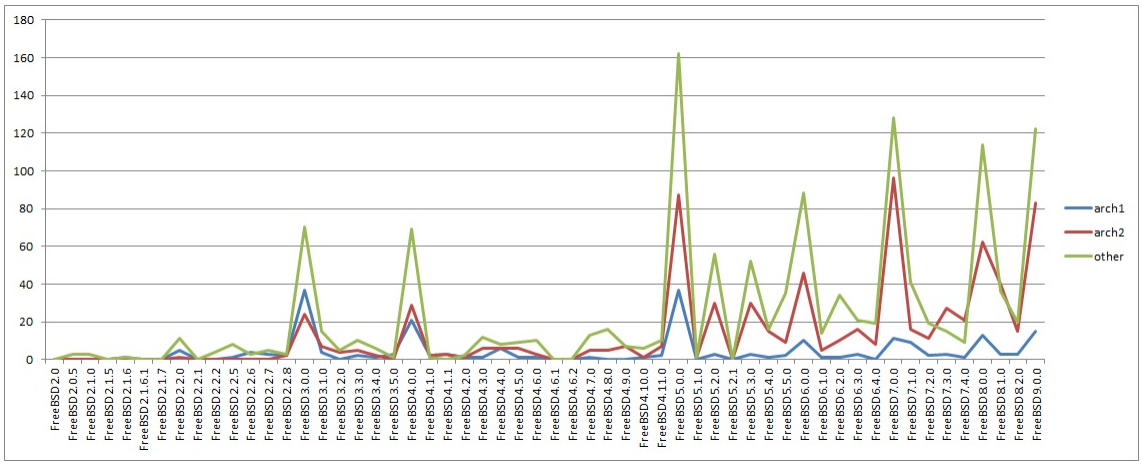

図 13: 3 コンポーネント間の障害報告数

5 関連研究

コードクローンに関して数多くの研究がされている [6].

Roy らは，いろいろなコードクローン検出技術を分類し，それぞれの特徴をまとめている. [24]．その中で，字面上全く等しい Type1，ユーザー定義の識別子を除いて等しい Type2，行の挿入，削除を行った上で等しい Type3，そして意味的に等しい Type4 を定義している．本研究では，主に Type1 および Type2 を対象としている．

Type2 のコードクローンの検出は，簡単なプロセスで高速かつ大規模な対象を分析できる一方，精度も比較的高く [4]，十分実用的であり，多くの企業や組織でそのツールが用いられている．

また，コードクローンの検出手法をテキスト比較，トークン列比較，木の比較，計測値比較，そしてグラフの比較に分類している．本研究で用いたのは，トークン列の比較を行う CCFinder である [14].

CCFinder は，C，C++，Java などのプログラムを入力し，それをトークン列に変換した後，ユーザー定義の識別子を特定の一つの特殊なトークンに正規化する．得られたトークン列に対して，サフィックス木アルゴリズム [26] を適用し，同型部分トークン列を検出する．得られたトークン列を元のテキストに変換すると共に，単純な代入の系列など，無意味なコードクローンを除去して出力する．

CCFinder は数百万行のプログラムを数分で分析する能力を持つが，より大きな対象を分析するためには，対象を分割し，多数のプロセッサで分散実行する D-CCFinder を用いる [21]．これを用いることで数億行の対象を数十時間で分析することが可能になる．

コードクローンの変遷に関しては，Godfrey や Antoniol らが詳しく研究している [11, 10,

27, 1, 2].

Wang と Godfrey は，Linux の SCSI ドライバにおける，クローンの変化を調査している [27]．そこでは，対象のアーキテクチャによる違い，時間と共に変化する割合，ハードウェアの近さとコードクローンの量との関係などに関して，詳細な調査を行なっている．しかし，これらの研究は，対象として非常によく似通ったソースコードの集合を対象としており，今回のような大規模な多様なソフトウェアシステムの集合とは特性が大きく異なっている．

Antoniol らは，Linux の 19 版のカーネルの変遷を分析し，その時間的変遷を調べている．その結果，コードクローンは比較的版を越えて安定して存在することを示した．[2] 一方，コンポーネント毎の分析や障害報告との関連などは行なっていない．

コードクローンは，ソフトウェアの障害との関連が示唆されているが，実際に定量的にそれを分析し示す研究はあまりない [16, 22]．Kapser らは，一概にコードクローンが障害に結びつく，という考えは危険であると主張している [16]．本研究では，これらの関係を調べ，より定量的な主張を行う．

Kim らは，個々のコードクローン毎の変遷を調べ，その変化の理由を開発者へのインタビューを通じて調査している [17]．今回の研究は，よりコードクローンをマクロに見ることにより，版のリリースによる大きな変化を調べる．

6　むすび

本研究では，FreeBSD を対象として，クローン率と障害報告の変化を分析した．分析の結果，以下を確認した

- 大規模改修を行った結果，クローン率や障害報告が急上昇したと考えられるリリース
- 類似したアーキテクチャのためのコンポーネントは，クローン率が高いこと
- 他のコンポーネントにコードクローンが吸収されたため，クローン率が低下したコンポーネント
- メジャーリリースにおいて，大量の障害が発生

今後の課題として，本研究で提案した調査手法を他のソフトウェアへ適用することが考えられる．例えば，Android OS など，近年になって開発を開始した携帯端末用 OS を対象とした分析を行うことで，ドメインや開発体制が与える影響の有無を確認したい．また，CCFinder 以外のコードクローン検出手法を用いた場合の，分析結果の検証を行う必要がある．CCFinder 以外のコードクローン検出手法として，プログラム依存グラフの等価性に基づく手法の適用が考えられる．

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 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.

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] G. Antoniol, G. Casazza, M. Di Penta, E. Merlo, “Modeling Clones Evolution through Time Series”, Proceedings of the 17th IEEE International Conference on Software Maintenance, ICSM 2001, pp. 273-280, 2001.

[2] G. Antoniol, U. Villano, E. Merlo, M. Di Penta, Analyzing cloning evolution in the Linux kernel, Information and Software Technology, Volume 44, Issue 13, 1 October 2002, Pages 755-765, ISSN 0950-5849, 10.1016/S0950-5849(02)00123-4.

[3] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant ’Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In Proc. International Conf. on Software Maintenance, pp. 368-377, Washington, DC, USA, 1998.

[4] S. Bellon, R. Koschke, G. Antoniol, J. Krinke and E. Merlo, “Comparison and Evaluation of Clone Detection Tools”, Transactions on Software Engineering, Vol.33, No.9, pp.577-591, 2007.

[5] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. Comparison and evaluation of clone detection tools. IEEE Trans. Softw. Eng., Vol. 33, pp. 577-591, September 2007.

[6] J. Cordy, K. Inoue, R. Koschke, and S. Jarzabek (ed.), “4th International Workshop on Software Clones (IWSC 2010)”, Cape Town, South Africa, May 2010.

[7] Stephane Ducasse, Matthias Rieger, and Serge Demeyer. A language independent approach for detecting duplicated code. In Proceedings of the IEEE International Conference on Software Maintenance, ICSM ’99, pp. 109-118, Washington, DC, USA, 1999. IEEE Computer Society.

[8] Raimar Falke, Pierre Frenzel, and Rainer Koschke. Empirical evaluation of clone detection using syntax suffix trees. Empirical Softw. Engg., Vol. 13, pp. 601-643, December 2008.

[9] M. Fowler. Refactoring: improving the design of existing code. Addison Wesley, 1999.

[10] M. Godfrey, and L. Zou, “Using Origin Analysis to Detect Merging and Splitting of Source Code Entities”, IEEE Tran. on Software Engineering, Vol. 31, No. 2, Feb. 2005.

[11] M. Godfrey and Q. Tu. "Tracking structural evolution using origin analysis" Proc. of the Int'l Workshop on Principles of Software Evolution, pp.117-119, Orland, Florida, 2002.

[12] Yoshiki Higo and Shinji Kusumoto. Code clone detection on specialized pdgs with heuristics. In CSMR, pp. 75-84, 2011.

[13] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In Proceedings of the 29th international conference on Software Engineering, ICSE '07, pp. 96-105, 2007.

[14] T. Kamiya, S. Kusumoto, K. Inoue: "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code", IEEE Trans. on Software Engineering, Vol. 28, No. 7, pp. 654-670, July 2002.

[15] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. Software Engineering, Vol. 28, pp. 654-670, July 2002.

[16] C. Kapser, and M. W. Godfrey, "'Cloning considered harmful' considered harmful: Patterns of cloning in software", Empirical Software Engineering, Vol. 13, No. 6, pp. 645-692, 2008.

[17] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, "An empirical study of code clone genealogies," Proc. of Foundations of Software Engineering (ESEC/FSE 2005), Vol. 30, No. 5, pp. 187-196, Lisbon, Portugal, Sep. 2005.

[18] Raghavan Komondoor and Susan Horwitz. Using slicing to identify duplication in source code. In Proceedings of the 8th International Symposium on Static Analysis, SAS '01, pp. 40-56, 2001.

[19] Jens Krinke. Identifying similar code with program dependence graphs. In Proc. 8th Working Conf. on Reverse Engineering, pp. 301-309, Washington, DC, USA, 2001.

[20] Mu-Woong Lee, Jong-Won Roh, Seung-won Hwang, and Sunghun Kim. Instant code clone search. In Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering, FSE '10, pp. 167-176, 2010.

[21] S. Livieri, Y. Higo, M. Matsushita, K. Inoue, “Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder”, Proc. of 29th International Conference on Software Engineering (ICSE 2007), pp.106-115, Minneapolis, MN, May 2007.

[22] A. Lozano, M. Wermelinger, B. Nuseibeh, “Evaluating the Harmfulness of Cloning: A Change Based Experiment”, Proc. of Mining Software Repositories (MSR 2007), p. 18-21, Minneapolis, MN, May 2007.

[23] Jean Mayrand, Claude Leblanc, and Ettore Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In Proceedings of the 1996 International Conference on Software Maintenance, ICSM ’96, pp. 244-, 1996.

[24] C. K. Roy, James R. Cordy, R. Koschke, “Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach”, Science of Computer Programming, Vol. 74, No. 7, pp. 470-495, 2009.

[25] Shinji Uchida, Akito Monden, Naoki Ohsugi, Toshihiro Kamiya, Ken ichi Matsumoto, and Hideo Kudo. Software analysis by code clones in open source software. Journal of Computer Information Systems, Vol. XLV, No. 3, pp. 1-11, April 2005.

[26] B. Smyth, “Computing Patterns in Strings”, Chapter 5, pp.109-149, Reading, Pearson, 2003.

[27] W. Wang, and M. Godfrey, “A Study of Cloning in the Linux SCSI Drivers”, Proc. of Source Code Analysis and Manipulation (SCAM), Williamsburg, VA, 2011.

[28] Wuu Yang. Identifying syntactic differences between two programs. Softw. Pract. Exper., Vol. 21, pp. 739-755, June 1991.

[29] 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. 529-536, 2001-09-17.

[30] 肥後芳樹, 神谷年洋, 楠本真二, 井上克郎. コードクローンを対象としたリファクタリング支援環境 (ソフトウェア開発環境・開発支援システム, ¡特集¿システム開発論文). 電子情報通信学会論文誌. D-I, 情報・システム, I-情報処理, Vol. 88, No. 2, pp. 186-195, 2005-02-01.

[31] Free BSD, http://www.freebsd.org.