コードクローン検出技術を用いた Linux カーネル進化の調査

リビエリ　シモネ[†]　　肥後　芳樹[†]（正員）
松下　　誠[†]　　井上　克郎[†]（正員）

An Investigation on Linux Kernel Evolution Using Code Clone Detection Technique

Livieri SIMONE[†], *Nonmember*, Yoshiki HIGO[†], *Member*, Makoto MATSUSHITA[†], *Nonmember*, and Katsuro INOUE[†], *Member*

[†] 大阪大学大学院情報科学研究科，豊中市
Graduate School of Information Science and Technology, Osaka University, 1–3 Machikaneyama-cho, Toyonaka-shi, 560–8531 Japan

あらまし　本論文では，コードクローン検出技術を用いた Linux カーネル進化の調査結果について述べる．本論文の実験は，筆者らが過去に開発した分散処理型コードクローン検出システム D-CCFinder を用いて行われた．検出結果は，ヒートマップとして可視化された．

キーワード　コードクローン，ソフトウェア進化，可視化

1. まえがき

コードクローンとは，ソースコード中の同一または類似したコード片を表す．近年，コードクローン検出技術が注目を集めており，筆者らはこの技術がソフトウェア進化の理解に役立つと考えている．つまり，バージョン間でコードクローン検出を行うことにより，ソフトウェアがどのように進化したのかを調査することができる [1].

コードクローン検出は，ソースコード以外のプロジェクトの成果物を必要としない．長期間保守されておりドキュメントが存在しないレガシーシステムや，存在していても更新されておらずソースコードとの対応がとれていないような場合でも適用できる．

また，適切に検出結果の抽象化を行うことにより，素早く大まかに進化の様子を知ることもできるであろう．現行プロジェクトに新規に投入された人員がソースコードの理解をしなければならない場合や，運用ソフトウェアの保守管理者が交代した場合など，分析者が対象ソースコードに対してあまり知識をもたない場合に特に有効ではないかと思われる．本論文では，コードクローン検出をソフトウェア進化の調査に用いた一例を示す．

しかし，バージョン間のコードクローン検出は，検出対象のソースコード量が大規模であるために，安直に検出処理を行うのは現実的ではない．効率良くコードクローンを検出することが重要である．そこで，筆者らが過去に開発した分散処理型コードクローン検出システム D-CCFinder [2] を用いる．D-CCFinder は，ネットワークで接続された複数台のマシンで分散してコードクローン検出を行い，検出処理の終了後にその結果を一つにまとめて可視化を行う [3].

表 1　Linux カーネルの規模
Table 1　The scale of Linux kernel.

バージョン	行数 (LOC)	サイズ (KByte)	バージョン数
1.0	141,361	3,926	1
1.2.0〜	234,704	6,534	14
1.2.13	237,888	6,596	
2.0.0〜	564,360	16,076	41
2.0.40	768,061	21,952	
2.2.0〜	1,310,698	37,056	27
2.2.26	1,970,123	58,812	
2.4.0〜	2,366,218	69,200	34
2.4.33.4	3,864,539	112,148	
2.6.0〜	4,120,925	120,030	19
2.6.18.3	5,475,540	157,290	
総バージョン数			136
総ファイル数			376,596
総行数 (LOC)			266,943,565
総サイズ (KByte)			7,760,910

本論文では，以下の理由により，ソフトウェア進化の調査対象として Linux カーネルを選択した．

- 各バージョンが十分に大規模であり，進化の調査を行う価値があると判断したため．
- 十分に多くのバージョンが公開されており，進化の調査を行うことができると判断したため．

本論文で調査に用いたのは，Linux カーネル 1.0〜2.6.18.3 の計 136 バージョン，総行数は約 2 億 6 千万行（総サイズは約 7.8 GByte）である．

2. 調査方法

Linux カーネルの各バージョンのペアにつき，類似度を算出した．バージョン V_i とバージョン V_j の類似度 $Coverage(i,j)$ は以下の式で表される．

$$Coverage(i,j) = \frac{LOC(C_{V_i}(V_j)) + LOC(C_{V_j}(V_i))}{LOC(V_i) + LOC(V_j)}$$

ただし，

$C_{x_0}(x_1)$: x_0 の中で，x_1 と重複している部分，

$LOC(x)$: x の行数．

バージョン V_i と V_j が完全に重複している場合，$Coverage(i,j)$ は最大値 1 となり，全く重複していない場合は，最小値 0 となる．

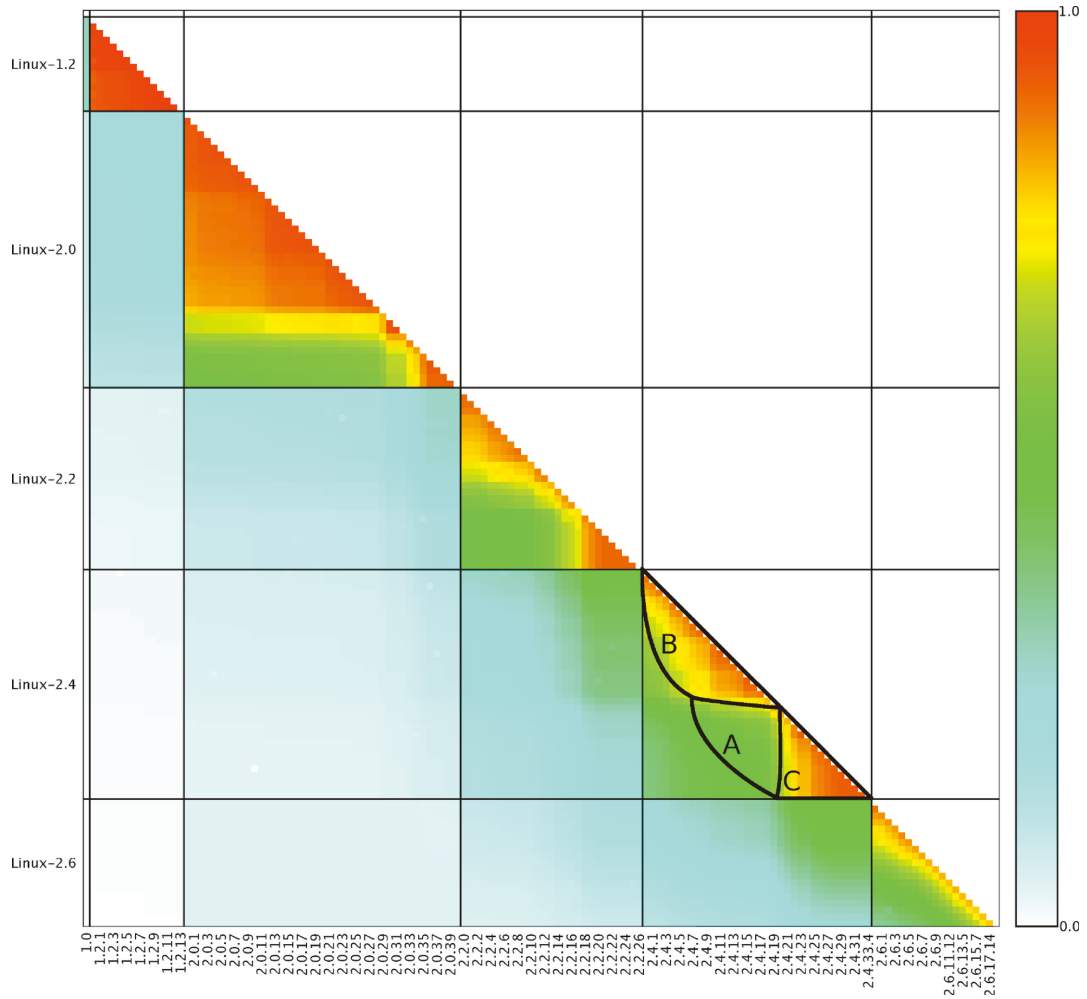

図 1　Linux カーネルのヒートマップ
Fig. 1　Heatmap of Linux kernel.

本実験では，連続して 50 字句以上一致している部分をコードクローンとして検出した．D-CCFinder を 80 台のワークステーション上で実行したところ，コードクローン検出に約 6 時間，類似度の算出に約 50 分を要した．進化の調査が目的であるため，バージョン間のみのコードクローンを検出しており，バージョン内については検出を行っていない．

類似度の算出後，進化の様子を俯瞰的に表すために，ヒートマップを用いて可視化を行った．

3. 結　　果

3.1 主対角線に近いほど類似度が高い

図 1 は，全バージョン間の類似度を表したヒートマップである．このヒートマップでは，類似度が高いほど赤に近い色になり，低いほど白に近い色になる．この図より，ひと目で主対角線に近い部分，つまり近いバージョン間では類似度が高いことが分かる．また，ヒートマップ上では近い位置にあるバージョン間であっても，メジャーバージョンをまたいでいる場合は，そうでない場合に比べて類似度が低くなっている．このことから，メジャーアップグレードはマイナーアップグレードに比べて，多くの変更が施されていることが分かる．

3.2 2.0 系，2.2 系，2.4 系に共通の進化パターン

2.0 系，2.2 系，2.4 系の各内部には，類似度の変化に共通の進化パターンが現れている．図 1 の A，B，C は 2.4 系内でのパターン部分を表している．B と C

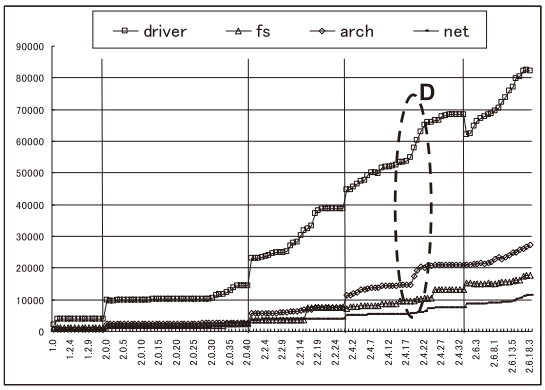

図 2　drivers，fs, arch, net の行数の変遷
Fig. 2　LOC transition of drivers, fs, arch, and net.

の部分は類似度が高いが，A の部分はそれほど類似度が高くなっていない．

2.0 系，2.2 系，2.4 系の類似度が低くなっている各部分について詳しく調査を行ったところ，新しいハードウェアドライバ，新しいアーキテクチャ，新しい機能（ネットワークプロトコルやファイルシステム）のためのコードが追加されていることが分かった．それらの追加によるソースコード行数の変化を図 2 に示す．図 2 から，2.0 系，2.2 系，2.4 系において，大きく行数が増加しているバージョンアップがあることが分かる．この図の D の部分は，新しいドライバとアーキテクチャの追加により行数が大きく増加している．このため，図 1 の A の部分は類似度がそれほど高くなっていない．

ソースコード中のコメントより，大きく行数が増加している部分のコードはバックポート(注1)によるものであることが分かった．このことから，2.0 系，2.2 系，2.4 系の共通進化パターンは，バックポート型開発の特徴を表しているといえるであろう．一方，2.6 系はバックポート型開発ではないために，同様のパターンが現れなかったと推測される．

(注1)：Linux 開発におけるバックポートとは，unstable ブランチにおいてある程度テストされ安定したと思われるコードを stable ブランチに移植する作業を指す．

4. むすび

本論文では，コードクローン検出を用いた Linux カーネルの進化の様子について調査を行った．調査の結果，マイナーアップグレードとメジャーアップグレードの違いやバックポートの特徴を効率良く確認することができた．これらは，ドキュメント等を精査することにより把握することも可能であろうが，本システムを利用することにより，効率良く視覚化でき，容易に把握することができた．

コードクローン分析はドキュメント等が存在しなくても適用することができるため，その利用可能範囲は広いといえるであろう．

今後は，他のソフトウェアの進化についても調査を行い，コードクローン検出を用いた進化分析の有益性を明らかにしたいと考えている．

文　献

[1] 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，” 信学論（D），vol.J89-D, no.10, pp.2279–2287, Oct. 2006.

[2] S. Livieri, Y. Higo, M. Matsushita, and K. Inoue, “Very-large scale code clone analysis and visualization of open source program using distributed ccfinder: D-ccfinder,” Proc. 29th International Conference on Software Engineering, pp.106–115, May 2007.

[3] S. Livieri, Y. Higo, M. Matsushita, and K. Inoue, “Analysis of the linux kernel evolution using code clone coverage,” Proc. 4th Workshop on Mining Software Repositories, pp.22.1–22.4, May 2007.

（平成 19 年 7 月 20 日受付，9 月 5 日再受付）