

産学連携に基づいたコードクロウン 可視化手法の改良と実装

肥後 芳樹

吉田 則裕

楠本 真二

井上 克郎

近年のソフトウェアについて

- ソフトウェアの大規模化に伴い、保守作業のコストが増大
- コードクローンが保守作業を困難にしている原因の1つとされている
 - あるコード片への修正により、そのコード片のコードクローンにも修正がいるかを検討
 - システムが大きいほど面倒
 - 修正漏れによるバグの混入

コードクローン検出手法

- CCFinder[7]
 - コード中のコードクローンを検出し、その位置をクローンペアのリストとして出力
- Gemini[15]
 - CCFinderの検出結果を可視化
 - クローン散布図
 - 算出したメトリクスによるグラフ

[7] Kamiya, T., Kusumoto, S. and Inoue, K.: 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).

[15] 植田泰士, 神谷年洋, 楠本真二, 井上克郎: 開発 保守支援を目指したコードクローン分析環境, 電子情報通信学会論文誌, Vol.86-D-I, No.12, pp. 863–871 (2003).

産学連携

- 産業界のコードクローン分析に対する要望を把握でき、コードクローン可視化手法を実用的な技術に高めた
 - コードクローンセミナー
 - 大学と産業界の意見交換の場
 - メーリングリスト
 - 現場の声を聞く手段

指摘された問題点

- Geminiは、開発現場の使用に向いていなかった
 1. コードクローン情報の中に、調査の必要があるコードクローンとそうでないものが混ざっている
 2. ファイルに着目した分析を行うための機能がない
 3. スケーラビリティが低い

問題点1の原因

- 問題点1
 - コードクローン情報の中に、調査の必要があるコードクローンとそうでないものが混ざっている

問題点1の原因

- 問題点1
 - コードクローン情報の中に、調査の必要があるコードクローンとそうでないものが混ざっている

コードクローン情報を扱う際に特に対象とする必要がないもの

- プログラミング言語やドメイン、フレームワークに依存した定型処理部分

問題点1の原因

- 問題点1
 - コードクローン情報の中に、調査の必要があるコードクローンとそうでないものが混ざっている
- 原因
 - CCFinderの検出結果をフィルタリングせずに可視化していた
 - 調査の必要がある情報を隠蔽し、分析作業の非効率化を招く

問題点2の原因

- 問題点2
 - ファイルに着目した分析を行うための機能がない
- 原因
 - 旧ツールは検出対象ファイルについての選択機構がなかった
 - 利用者はコードクローンを多く含むファイルや重複度の高いファイルに興味を持っている

問題点3の原因

- 問題点3
 - スケーラビリティが低い
- 原因
 - 旧ツールが円滑に動作する対象は10万行程度の規模までだった
 - 商用ソフトウェアを対象とした際、それでは不十分

提案手法1 クローンのフィルタリング

- 調査の必要がないコードクローン
 - プログラミング言語に依存した、言語依存のコードクローン
 - 同じ言語を使っていれば、どんなアプリケーションでも検出可能
 - ドメインなどに依存した、アプリケーション依存のコードクローン
 - 対象ソフトウェアによって大きく異なる

提案手法1 クローンのフィルタリング

- 調査の必要がないコードクローン
 - プログラミング言語に依存した、言語依存のコードクローン
 - 同じ言語を使っていれば、どんなアプリケーションでも検出可能
 - ドメインなどに依存した、アプリケーション依存のコードクローン
 - 対象ソフトウェアによって大きく異なる

提案手法1 クローンのフィルタリング

- メトリクス $RNR(S)$
 - クローンセット S に含まれるコード片がどの程度繰り返し要素を含まないかを表す

$$RNR(S) = 1 - \frac{\sum_{f \in S} TOC_{repeated}(f)}{\sum_{f \in S} TOC(f)}$$

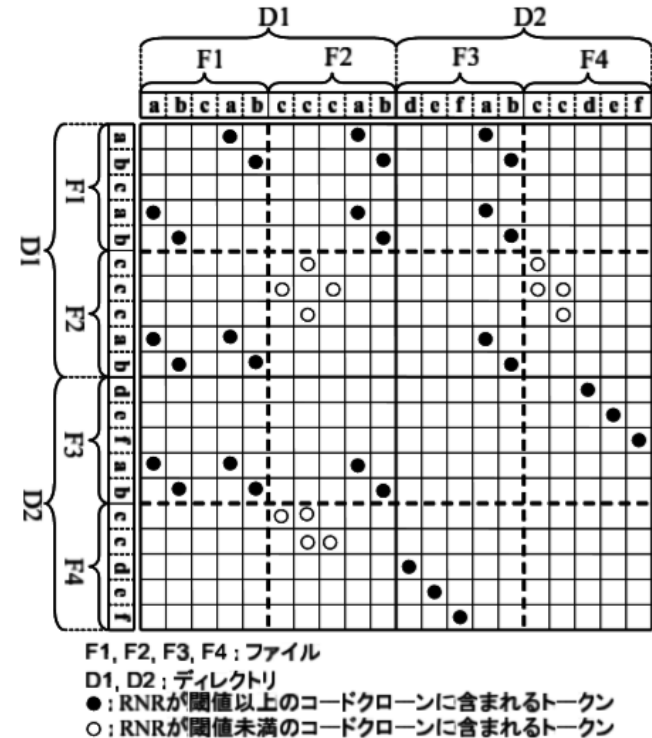
f : クローンセット S に含まれるコード片

$TOC(f)$: f を構成している字句の数

$TOC_{repeated}(f)$: f を構成している字句のうち、繰り返し要素の字句の数

提案手法2 クローン散布図

- a~f
: ファイルを構成する字句
- ●、○に分けることで、分析作業の効率化
- ファイル間、ディレクトリ間で境界線を区別
 - コードクローンを多く含むディレクトリ



提案手法3 クローンセットメトリクス

- $LEN(S)$ [15]
 - クローンセット S に含まれるコード片の大きさ(字句数)の平均
- $POP(S)$ [15]
 - クローンセット S に含まれるコード片の数
- $NIF(S)$
 - クローンセット S に含まれるコード片を所有しているファイルの数
- $RNR(S)$

[15]植田泰士, 神谷年洋, 楠本真二, 井上克郎:開発 保守支援を目指したコードクローン分析環境, 電子情報通信学会論文誌, Vol.86-D-I, No.12, pp. 863–871 (2003).

提案手法4 ファイルメトリクス

- $NOC_{th}(F)$
 - ファイル F に存在するコード片の数
- $ROC_{th}(F)$
 - ファイル F がどの程度重複しているか
- $NOF_{th}(F)$
 - ファイル F がコードクローンを共有しているファイルの数

th : RNRの閾値

新Geminiの実装

- 提案手法1～4に基づき、Geminiを再実装
- 新Geminiが持つビュー
 - クローン散布図
 - メトリクスグラフ
 - ファイルリスト

新Geminiのビュー クローン散布図

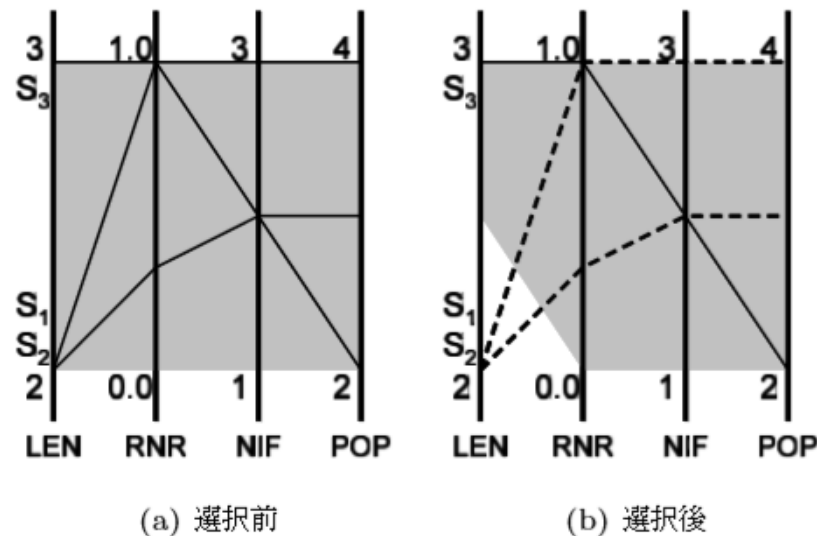
- コードクローンの分布状態を俯瞰的に把握できる
- クローン散布図はスケーラビリティが低く、実用的でない
[8][14]
 - JDK1.5のソースコード全体(約1,880,000行)から検出したコードクローン(クローンセット数:約12,000個、クローンペア数:2,500,000個)に対し、円滑に動作した

[8] Kapser, C. and Godfrey, M.: Improved Tool Support for the Investigation of Duplication in Software, Proc. of the 21st International Conference on Software Maintenance, pp.305–314 (2005).

[14] Rieger, M., Ducasse, S. and Lanza, M.: Insights into System-Wide Code Duplication, Proc. the 11th Working Conference on Reverse Engineering, pp.100–109 (2004).

新Geminiのビュー メトリクスグラフ

- 定量的な特徴に基づいてクローンセットを選択できる
- 多次元並行座標表現を使う[9]



[9]加藤博己: データベースのビジュアルな検索と分析 (OLAP), 情報処理学会会誌, Vol.41, No.4, pp.363-368 (2000).

新Geminiのビュー ファイルリスト

- ソースファイルを選択する
- $LOC(F), TOC(F)$: ファイル F の行数と字句数

File List						
FILE ID	LOC(f)	TOC(f)	NOC(f)	ROC(f)	NOF(f)	File Name
FILE 0.0	137	121	2(2)	82(82)	1(1)	ZipShort.java
FILE 0.1	203	408	2(2)	39(39)	1(1)	ExtraFieldUtils.java
FILE 0.2	113	5	0(0)	0(0)	0(0)	UnixStat.java
FILE 0.3	135	124	0(0)	0(0)	0(0)	UnrecognizedExtraField.java
FILE 0.4	496	861	10(12)	23(23)	14(17)	ZipEntry.java
FILE 0.5	119	6	0(0)	0(0)	0(0)	ZipExtraField.java
FILE 0.6	557	1018	3(7)	6(12)	1(1)	ZipFile.java
FILE 0.7	141	160	3(3)	83(83)	2(2)	ZipLong.java
FILE 0.8	871	1686	8(17)	24(29)	2(2)	ZipOutputStream.java
FILE 0.9	800	1600	8(17)	24(29)	2(2)	ZipInputStream.java
FILE 0.10	442	668	4(4)	22(22)	1(1)	TarBuffer.java
FILE 0.11	191	5	0(0)	0(0)	0(0)	TarConstants.java
FILE 0.12	682	993	2(5)	3(13)	4(6)	TarEntry.java
FILE 0.13	417	628	1(1)	6(6)	1(1)	TarInputStream.java
FILE 0.14	348	462	1(1)	8(8)	1(1)	TarOutputStream.java
FILE 0.15	233	317	0(0)	0(0)	0(0)	TarUtils.java
FILE 0.16	81	22	0(0)	0(0)	0(0)	ErrorInQuitException.java
FILE 0.17	562	969	7(9)	15(19)	15(15)	MailMessage.java
FILE 0.18	131	157	0(0)	0(0)	0(0)	SmtResponseReader.java
FILE 0.19	136	5	0(0)	0(0)	0(0)	BZip2Constants.java
FILE 0.20	873	2069	24(25)	38(39)	2(2)	CBZip2InputStream.java
FILE 0.21	1670	3940	29(47)	21(26)	2(2)	CBZip2OutputStream.java
FILE 0.22	167	81	0(0)	0(0)	0(0)	CRC.java
FILE 0.23	277	295	2(6)	15(39)	1(1)	AnsiColorLogger.java
FILE 0.24	339	716	3(4)	23(26)	1(1)	CommonsLoggingListener.java

新Geminiの適用実験

- IPA ソフトウェア・エンジニアリング・センターの先進ソフトウェア開発プロジェクトにおいて、ベンダー5社が開発しているプローブ情報システムに対して適用
- 対象規模
 - 5社合計で数十万ステップ
 - ある社の担当したサブシステムの規模は、他4社のサブシステムの合計規模よりも大きかった

旧Geminiとの比較

- クローン散布図を使った分析
 - 旧Geminiはスケーラビリティの問題から、分析が困難
- メトリクスグラフを使った分析
 - 旧Geminiでも同様の分析ができたが、フィルタリングメトリクスを加えたことで、より効率的になった
- ファイルリストを使った分析
 - ファイルメトリクスはどれも新Geminiから導入されたもので、旧Geminiで同様の分析はできない

まとめ

1. コードクローン情報の中に、調査の必要があるコードクローンとそうでないものが混ざっている
 - 言語依存のコードクローンのフィルタリングは行えたが、アプリケーション依存のコードクローンへの対策も必要
2. ファイルに着目した分析を行うための機能がない
 - 特定のコードクローン・ファイルの情報を得ることができるようになった
3. スケーラビリティが低い
 - 再実装により高くなったが、更なる改良が必要

主観

- 問題点、それに対する改良点、分析結果、今後の課題が分かりやすくまとめられていた
- 新旧の比較が書かれているから、新Geminiの方でできるようになったことが分かりやすい