研究速報

プログラム変更支援を目的としたコードクローン情報付加ツールの実装と評価

佐々木　亨† 肥後　芳樹†
神谷　年洋††（正員） 楠本　真二†（正員）
井上　克郎†（正員）

A Code Clone Information Supplement Tool to Support Program Change

Toru SASAKI†, Yoshiki HIGO†, *Nonmembers*,
Toshihiro KAMIYA††, Shinji KUSUMOTO†,
and Katsuro INOUE†, *Members*

† 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，豊中市
Graduate School of Information and Science Technology, Osaka University, Toyonaka-shi, 560–8531 Japan
†† 科学技術振興機構さきがけ
PRESTO, Japan Science and Technology Agency

あらまし　本論文では，コードクローン検出ツールCCFinderで検出されたコードクローン情報をコメントとしてソースコード中に付加するツールを提案する．また，適用例を通じて本ツールの有用性を示す．

キーワード　コードクローン，ソフトウェア保守，コメント，ツール

1. まえがき

ソフトウェアの保守を困難にする要因の一つとしてコードクローンがある．コードクローンとは，ソースコード中に存在するコード片で，同形のコード片が他に存在するものであり，既存コードの「コピーとペースト」による再利用等によりプログラム中に作り込まれる．あるコード片に対する変更を行う場合，そのコードクローンすべてに対して同一の変更を検討する必要があるが，大規模ソフトウェアに対してはそのような作業は極めて困難である．

我々は，ソフトウェア保守支援を目的としてソースコード中のコードクローンを検出するツール（CCFinder [3]）を開発してきている．その出力では，コードクローンの位置情報はソースコード中での行番号で提示されており，その情報を利用して機能追加等のためにソースコードを修正することを考えると，コードクローン情報とソースコードとの間に行番号のずれが生じる．したがって，複数個所の修正（あるいは，機能追加）を一度に行う場合には，その行番号のずれを意識しながら作業しなければならず，作業効率が悪くなってしまう．

そこで本論文では，行番号のずれを意識せずにソフトウェアの一貫した変更を支援するために，コードクローン情報をコメントとしてソースコード中に付加するツールを提案する．更に，本ツールを実システムの修正事例に対して適用し，その有用性を確認する．

2. コードクローン検出ツール CCFinder

コードクローン検出ツールCCFinderは，単一または複数のソースコード中からすべてのコードクローンを検出し，コードクローンの位置情報を出力する．その主な特徴は以下のとおりである．

- ソースコードをトークン単位で直接比較することによりクローンを検出する．
- クローン検出アルゴリズムにサフィックス木 [1] を用いて高速化を図り，数百万行規模のシステムにも実用時間で解析可能である．
- 実用上意味のないクローン（モジュール先頭のテーブル初期化文の繰返し等）は検出しない．
- 複数のプログラミング言語（C/C++，JAVA，COBOL，Fortran，EmacsLisp，Plain text）へ対応している．

3. コードクローン情報付加ツール

3.1 基本方針

本ツールは，プログラム変更時の利用を想定している．具体的には，次の（1）〜（5）の作業を支援する．

（1） CCFinderから得られた解析結果をもとにソースコード中にクローン情報をコメントとして付加する．コメントにはクローンクラス（クローンの同値類 [2]）ごとに一意に割り付けたIDを利用する．

（2） 変更個所を特定し，変更を行う．

（3） 変更個所のコメント部分のクローンクラスのIDを調べ，それを手掛りにそのクローンクラスに属している別のコード片の先頭個所を特定する．

（4） 先頭個所から修正すべき個所を特定し，必要ならば修正を行う．

（5） すべての個所に修正を行った後，不要となったコメントを除去する．

3.2 概　　要

3.1 で挙げた利用状況を実現するために以下の五つの機能を実装したツールを開発した．

［コメント追加機能］ クローンの位置情報をコメントとしてソースコード中に付加する機能．

コメントの形式として，クローンクラスの先頭行には“//@$クローンクラスのID@”を，先頭行以外には“//@クローンクラスのID@”の形式をとる．

［クローンクラス検索機能］ 指定したクローンクラスのIDを含むコメントが存在するファイル情報を検索・

表示する機能.
［一時ファイル編集機能］ 付加されたクローン情報を利用して編集するため，利用者の選択したエディタでファイルを開く機能.
［コメント除去機能］ 指定したクローンクラスの ID を含むコメントを除去する機能.
［一時ファイル書き戻し機能］ 編集用の一時的ファイルで行った変更を元のファイルに反映する機能.

4. 適 用 例

擬似デバッグにより本ツールの適用例を示す．ここでは日本語入力システム「かんな」のバージョン 3.6 と 3.6p1 の間での修正を例題とした．この修正は，バッファオーバフローを調べる処理の追加であり，全 20 個所にほぼ同様の修正をしている．ある一つの修正個所から残りの 19 個所を探すデバッグ作業を考える．まず，バージョン 3.6 の全ソースコード（92 ファイル，約 9 万行）に対して本ツールでクローン情報をコメントとして追加し，次に，バッファオーバフローを調べる処理を追加する．コメント，処理を追加した後のソースコードの一部を図 1 に示す．太字で示される個所が本ツールの機能で追加されたコメントであり，A の部分が追加した処理である．次に，利用者は修正個所の前後にあるコメント部分のクローンクラスの ID（図中の B）を調べる．すると，クローンクラス 1311 と 1318 の中にこの修正処理を追加したことが分かるので，この後，1311 と 1318 のクローンクラスに属しているコード片に対して同様の修正を行えばよいと分かる．この作業の間，利用者は行番号のずれを意識せずに作業することができる．

次に，本ツールはある修正個所から他の修正個所を探すという情報検索的な利用をする点から，f 値 [4] による評価を行う．f 値とは完全性と効率性から検索結果の精度を評価するもので，次式で求められ，その値が大きいほど情報検索の精度が高いとされる．

$$f\ 値 = \frac{2 \times 完全性 \times 効率性}{完全性 + 効率性}$$

また，ここでは「完全性」を修正が必要な個所のうち実際に検出された個所の割合，「効率性」を検出された個所のうち実際に修正個所であった割合とした．また，本ツールにおける「検出された」の意味を，ある修正個所を特定したときに，その個所にあるコメント内のクローンクラスの ID をたどってたどり着ける個所とした．適用事例に対する，検出された修正個所の数，f 値を計算し，標準的な検索ツールである grep と比較する．grep 検索の引数には，バッファ処理時に使われる変数名の共通部分 “Request.type” を用いた．

結果を表 1 に示す．この結果から，本ツールは grep よりも修正が不要な部分を検出しないことにより f 値が大きくなっている（精度が高くなっている）といえる．なお，検出されなかった 2 個所も，CCFinder で検出するコードクローンの長さを小さくすることで，検出することができた．

表 1 検出結果と f 値
Table 1 Detection result and f-value.

	総修正個所	全検出個所	検出修正個所	完全性	効率性	f 値
grep	20 個所	61 個所	19 個所	95%	31%	0.50
本ツール	20 個所	18 個所	18 個所	90%	100%	0.95

```
2399: static //@$1311@,@$1318@
2400: ProcWideReq7(buf) //@1311@,@1318@
2401: BYTE *buf ; //@1311@,@1318@
2402: { //@1311@,@1318@
2403:     ir_debug( Dmsg(10, "ProcWideReq7 start!!\n") ); //@1311@,@1318@
2404:  //@1311@,@1318@
2405:     if(Request.type7.datalen != SIZEOFSHORT * 3)
2406:         return( -1 );
2407:     buf += HEADER_SIZE; Request.type7.context = S2TOS(buf); //@1311@,@1318@
2408:     buf += SIZEOFSHORT; Request.type7.number = S2TOS(buf); //@1311@,@1318@
2409:     buf += SIZEOFSHORT; Request.type7.yomilen = (short)S2TOS(buf); //@1311@,@1318@
2410:     ir_debug( Dmsg(10, "req->context =%d\n", Request.type7.context) ); //@$1317@
2411:     ir_debug( Dmsg(10, "req->number =%d\n", Request.type7.number) ); //@1317@
2412:     ir_debug( Dmsg(10, "req->yomilen =%d\n", Request.type7.yomilen) ); //@1317@
2413:  //@1317@
2414:     return( 0 ) ; //@1317@
2415: } //@1317@
```


図 1 コメント追加・修正後のソースコード
Fig. 1 The source code after a comment addition and correction.

5. むすび

本論文では，複数個所のプログラム修正作業を支援するためのコードクローン情報付加ツールについて述べ，有用性を確認した．今後の課題としては，実際の開発・保守現場での適用実験と評価が考えられる．

謝辞 本研究は一部，文部科学省リーディングプロジェクト「e-Society 基盤ソフトウェアの総合開発」の支援を受けている．

文献

[1] D. Gusfield, Algorithms on Strings, Trees, And Sequences, Cambridge University Press, 1997.

[2] 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.47–54, 2001.

[3] T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: A multilinguistic token-based code clone detection system for large scale source code,” IEEE Trans. Softw. Eng., vol.28, no.7, pp.654–670, 2002.

[4] 徳永健伸，情報検索と言語処理，東京大学出版会，2002.

（平成 16 年 3 月 31 日受付）