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

佐々木 亨[1] 肥後 芳樹[2] 神谷 年洋[3] 楠本 真二[2] 井上 克郎[2]

[1]大阪大学基礎工学部情報科学科 [2]大阪大学大学院情報科学研究科 [3]科学技術振興機構 さきがけ

1. まえがき

ソフトウェア保守を困難にする要因としてコードクローンがある．コードクローンとは，ソースコード中の同一もしくは類似したコード片のことで，既存のコードの「コピーとペースト」による再利用や頻繁に用いられる同一処理などによってプログラム中に作りこまれる．もし，複数のコードクローンを持つコード片を修正するならば，そのコードクローン全てに同じ修正を行うかどうかを確認する必要がある．しかし，全てのコードクローンに対して一貫した確認・修正を行うことはきわめて困難である．

これまでに様々なコードクローン検出法が提案されている[1][2]．我々の研究グループにおいてもツール CCFinder[5]を開発したが,その出力中，コードクローン情報はソースコードの行番号で提示されており，当然ながら，機能追加等のために修正されたソースコードとコードクローン情報との間に行番号のずれが生じる．従って，複数箇所の修正（機能追加）を一度に行う場合にはそ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る．

本稿では，コードクローン情報をソースコード中にコメントとして付加するツールに関して述べる．

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，単一又は複数のソースコード中から全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通りである．

(1)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コードクローンを検出する．

(2)コードクローン検出アルゴリズムにサフィックス木[3]を用いて高速化を図り，数百万行規模のシステムを実用時間で解析可能である．

(3)実用上，意味のないコードクローン(モジュールの先頭にあるテーブルの初期化文の繰り返し等)は検出しない．

(4)複数のプログラミング言語へ対応している．

3. コードクローン情報付加ツール

3.1 想定する利用状況

本ツールは，ソースコードの変更(バグ修正・機能追加)時の利用を想定している．具体的には，次の(1)～(4)中の作業を支援する．

(1)CCFinder から得られた解析結果を基にあらかじめソースコード中にクローン情報をコメントとして付加しておく．コメントにはクローンクラス（クローンの同値類[4]）毎に一意に付けた ID を利用する．

(2)変更箇所を特定し，変更を行う．

(3)変更箇所のコメントを調べ，コメント中のクローンクラス ID をたどって，そのクローン全てに対して，変更が必要であるか検討し，必要であれば変更する．

(4)全ての必要箇所に対して検討・変更を行った後で，不要となったコメントを除去する．

3.2 ツールの概要

3.1 で述べた利用状況に対する支援を実現するために以下の 4 つの機能が実装されている．

(1)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，ソースコードに付加する(ただし，編集するための一時的なファイルを作り，元のファイルには追加しない)．

(2)ファイルオープン機能
コメントを追加したファイルを編集するためにエディタを開く．

(3)コメント除去機能
不要となったコメントを，クローンクラスの ID を指定して除去する．複数を同時に除去することも，クローン情報に関する全コメントを除去することもできる．

(4)ファイルコピー機能
編集用の一時的ファイルの変更を元のファイルに反映するために，全コメントを除去したのちにコピーする．

Code clone information addition tool for software maintenance
(1) Toru SASAKI
Department of Informatics and Mathematical Science, School of Engineering Science, Osaka University
(2) Yoshiki HIGO, Shinji KUSUMOTO, Katsuro INOUE
Graduate School of Information Science and Technology, Osaka University
(3) Toshihiro KAMIYA
PRESTO, Japan Science and Technology Agency

```
2399 static //#CL1311,1318
2400 ProcWideReq7(buf) //#CL1311,1318
2401 BYTE *buf ; //#CL1311,1318
2402 { //#CL1311,1318
2403     ir_debug( Dmsg(10, "ProcWideReq7 start!!¥n") ); //#CL1311,1318
2404 //#CL1311,1318
2405     buf += HEADER_SIZE; Request.type7.context = S2TOS(buf); //#CL1311,1318
2406     buf += SIZEOFSHORT; Request.type7.number = S2TOS(buf); //#CL1311,1318
2407     buf += SIZEOFSHORT; Request.type7.yomilen = (short)S2TOS(buf); //#CL1311,1318
2408     ir_debug( Dmsg(10, "req->context =%d¥n", Request.type7.context) ); //#CL1317
2409     ir_debug( Dmsg(10, "req->number =%d¥n", Request.type7.number) ); //#CL1317
2410     ir_debug( Dmsg(10, "req->yomilen =%d¥n", Request.type7.yomilen) ); //#CL1317
2411  //#CL1317
2412     return( 0 ) ; //#CL1317
2413 } //#CL1317
```

A

例えば，この部分がクローンクラス1317のコード片の一つになっている

図1 実行例 コメント追加後のソースコード

4. 適用例

疑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」[6]のバージョン 3.6 と 3.6p1 の間でのセキュリティ問題の修正を例題とした．この修正は，バッファのオーバーフローを調べる処理の追加で，1 ファイル全 20 箇所に同じ修正を行っている．

まず，「かんな」のバージョン 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た．この処理には約 3 秒の実行時間を要した．コメント追加後のソースコードの一部を図 1 に示す．図 1 の太字で示される箇所が本ツールの機能によって追加されたコメントである．例えば A の箇所の場合，この行がクローンクラス 1311 と 1318 に含まれることを意味している．

次に，図 1 のソースコード中の A の次行に上述のバッファオーバーフロー検査処理を追加する．具体的には以下の 2 行が挿入される．

```
if(Request.type7.datalen!=SIZEOFSHORT* 3)
  return( -1 );
```

この修正によりソースコードの行番号は変化してしまうが，コードクローン情報をコメントとして追加しているため，他のコードクローンに対して修正を行いたい場合にも行番号のずれを意識せずに済む．他の修正箇所を探すときには付近にあるコメントのクローンクラス ID を手がかりとすればよい．この例の場合，この関数中には 1311,1317,1318 という 3 つのクローンクラスがあることがわかるので，そのクローンクラスを含む他の箇所に対しても修正処理の追加の是非を検討する必要がある．

最後に，あるクローンクラスに対する修正を終えた時点で，あるいはすべての修正を終えた時点で，コメント除去機能により不要になったクローンクラスのコメントを除去すればよい．

5. 考察

本ツールを利用することで，複数箇所の修正を一度に行いたい場合に，修正によるコードクローン情報とソースコードとの間の行番号のずれを意識せずに作業できることがわかる．

また，grep 等の検索ツールで修正箇所を探す場合には，ユーザが明示的に検索対象のコード片を指定する必要があり，結果がユーザの能力に依存すると考えられるが，本ツールの場合は，指定する必要がないので，そのような心配はないと期待される．

6. まとめと今後の課題

本稿では，ソフトウェア保守支援を目指したコードクローン情報付加ツールについて述べた．今後の課題としては，実際の保守現場での適用と評価が考えられる．

文献

[1] M.Balazinska, E.Merlo, M.Dagenais, B.Lague, and K.Kontogiannis, “Measuring Clone Based Reengineering Opportunities”, *Proc. of METRICS99*, 1999, 292-303.

[2] I.D.Baxter, A.Yahin, L.Moura, M.Sant’Anna, and L.Bier, “Clone Detection Using Abstract Syntax Trees”, *Proc. of ICSM98*, 1998, 368-377.

[3] D.Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, 1997.

[4] 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.

[5] T.Kamiya, S.Kusumoto, and K.Inoue, “CCFinder:A multilinguistic token-based code clone detection system for large scale source code”, *IEEE Trans. on Softw. Eng.*, 2002, 28(7):654-670

[6] 日本語入力システム「かんな」 http://canna.sourceforge.jp/