1 English abstract

In this letter, we develop a code clone information retrieval tool for software maintenance activity. Then, we have applied the tool to the debug data collected from a certain software project and evaluated the usefulness of it.

2 English keyword

Code Clone, Software maintenace

ソフトウェア保守のための類似コード片検索ツール

泉田　聡介[†]　植田　泰士[†]
神谷　年洋[††]　楠本　真二[†] (正員)
井上　克郎[†] (正員)

Code clone information retrieval tool for software maintenance

Sousuke IZUMIDA[†], Yasushi UEDA[†],
Toshihiro KAMIYA[††], *Nonmembers*, 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
Graduate School of Information Science and Technology, Osaka University
[††] 科学技術振興事業団 さきがけ研究 21
PRESTO, Japan Science and Technology Corp.

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に基づいて, ソフトウェア保守を効率よく実施するためのコードクローン検索ツールについて述べる.

キーワード　コードクローン ソフトウェア保守

1. まえがき

近年, ソフトウェアシステムの大規模化, 複雑化に伴い, プログラムの保守・デバッグ作業に要するコストが増加してきている. ソフトウェア保守を困難にしている一つの要因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, ソースコード中に含まれる同一または類似したコード片のことである [2]. それらは多くの場合, 既存システムに対する変更や拡張時におけるコピーとペーストによる安易な機能的再利用の際に発生する. もしあるコード片にバグが含まれていた場合, そのコード片に対する全てのコードクローンについて修正を行わなければならない. 大規模ソフトウェアから効率よくコードクローンを検出する手法が求められている.

これまで様々なコードクローン検出法が提案されている [1] [2]. 我々もトークン単位でのコードクローンを検出するツール (CCFinder [4]) を開発してきており, 様々なソフトウェアに対する適用を行ってきている. これらの適用の中で, CCFinder は大規模なソースコードも, 細粒度でのコードクローン解析を非常に高速に行うことが可能であると評価されてきた. しかし, CCFinder の出力結果は, テキスト情報であり, 実際に保守を実施する場合には, コードクローンの位置を直感的に把握することが困難であった.

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に基づいて, ソフトウェア保守を効率よく実施するためのコードクローン検索ツールについて述べる.

2. コードクローン検索ツール CCFinder

コードクローン検出ツール CCFinder は, 単一または複数のファイルのソースコード中から全てのコードクローンを検出し, クローンの位置情報として出力する. CCFinder の持つ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クローンを検出する.

(2) クローン検出アルゴリズムにサフィックス木を用いることで高速化を図っており, 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.

(3) 実用的に意味のないクローン (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. コードクローン検索ツール

3.1 基本方針

今回作成したコードクローン検出ツールは, デバッグ時の利用と機能追加時の利用を想定している. 具体的には, 修正（あるいは, 機能追加）対象のコード片が特定された時に, そのコード片に対応するコードクローンを前もって検出しておき, それらについても修正（あるいは, 機能追加）が必要であるかを検討する時の支援を行う.

検索ツールとしては, ユーザが入力したコード片に対して, そのコードクローンを含むファイルを検索し, ユーザに対してわかりやすく提示する必要がある. また, それらの処理を簡便に行えることも重要である.

3.2 概　　要

開発した検索ツールでは, まず, 入力として, ユーザが指定するコード片とコードクローン検出対象となるファイル名のリストを与える. その結果, 出力としてユーザが指定したコード片のクローンを含むファイルのリストを表示する.

検索ツールの画面例を図 1 に示す. ユーザが指定するコード片をBに入力する. エディタ等からコピーとペーストによって入力することもできる. また, コードクローン検出対象となるファイル名のリストをAに入力する. このリストの作成は, Cに提示されているディレクトリ構造から検索対象ファイルを選ぶ操作によっても行える.

図 1 の手前のウィンドウに実行結果を示す. ユーザが指定したコード片のクローンを含むファイルのリストが, 新しいウィンドウに表示される. リスト中の下

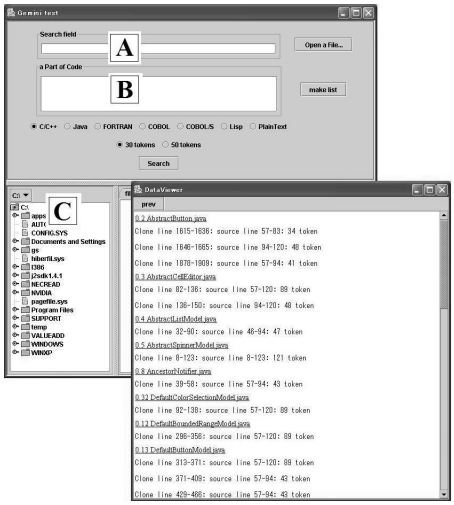

図 1 画 面 例
Fig. 1 snapshot

線のついたファイル名を選ぶことで, そのファイルの内容が表示され, コードクローンがハイライト表示される.

3.3 適 用 例

開発したツールを用いた疑似デバッグによって, ツールの適用例を示すとともに, その有効性を評価する.

実験では, SourceForge で開発されている日本語入力システム「かんな」[3] の修正例を用いた. 具体的には, 「かんな」バージョン 3.6 とバージョン 3.6p1 間でのセキュリティ問題の修正で, バッファ処理の前にオーバーフローを調べる処理を追加してあり, その中でほぼ同じ修正を行っている 21 箇所を対象とした.

修正されたコード片の一つから残り 20 個のコード片を検索する作業において, 標準的な検索ツールである grep と本ツールを比較する. まず, grep を用いた検索では, 修正箇所で使用されている "Request.type" という変数名の一部で検索を行った. 本ツールではバッファ処理を行っている部分の 2 行を入力コード片として与えて検索を行った. 検索対象ファイルは「かんな」ver3.6 の全ソースコード (約 21, 000 行) である.

検索結果と検索に要した時間を表 1 に示す. grep での検索結果では 234 行が検出された. このうち, 2 行以上連続したコード片を 1 つの修正対象コード片とみなすと, 全部で 58 箇所となった. このうち, 134 行 (20 箇所) が修正箇所に関連した場所であった. 一方, 本ツールの検索結果では, 17 箇所が検出され, 4 箇所が発見されなかった. しかし, 発見された 17 箇所は全て正しく修正箇所を示していた. 本ツールで発見されなかった部分は, 入力コード片として与えた 2 行が連続していなかった部分であった.

表 1 検 索 結 果
Table 1 retrieval result

	検索結果	検索時間
grep	243 行 (58ヵ所)	1 秒以内
本ツール	17ヵ所	8 秒

表 2 結果の比較
Table 2 Comparison

	完全性	効率性	f 値
grep	95%	34%	0. 50
本ツール	81%	100%	0. 90

この 2 つの結果を f 値 [5] を用いて比較する. f 値とは完全性と効率性から情報検索の精度を評価するものであり,

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

で求められる.

ここで, 完全性は必要な情報のうち実際に検索された情報の割合を, 効率性は実際に検索された情報のうち必要な情報の割合と定義され, 両方ともに高いほど優れた検索システムであると判断される. 従って, f 値の値が大きいほど, 情報検索の精度が高いといえる. 本実験結果に対して, f 値を計算した結果を表 2 に示す. 本ツールの結果が, grep を用いた検索結果よりもよい結果を示している.

4. ま と め

本論文では, ソフトウェア保守を支援するためのコードクローン検索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・評価や検索されたコードクローンの絞り込み等の改良が考えられる.

文 献

[1] M. Balazinska , E. Merlo, M. Dagenais, B. Lagüe, 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] 日本語入力システム「かんな」http://canna.sourceforge.jp/
[4] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code", *IEEE Trans. on Soft. Eng.*, 2002, 28(7):654-670.
[5] 徳永 武信, "情報検索と言語処理", 東京大学出版会, 2002.

(平成 x 年 xx 月 xx 日受付)