研究速報

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†, *Regular 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，吹田市
Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871 Japan
†† 独立行政法人科学技術振興機構 さきがけ
PRESTO, Japan Science and Technology Agency, Japan

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良く実施するためのコードクローン検索ツールについて提案する．また適用例を通じて本ツールの有効性を示す．

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すべ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良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，実際の保守作業において利用する場合には，コードクローンの位置情報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良く実施するためのコードクローン検索ツールについて提案する．また，本ツールの有効性を grep との比較を用いて示した．

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，単一または複数のファイル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として出力する．CCFinder のもつ主な特徴は以下のとおりである．

（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返し等）は検出しない．

（4） 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索ツールは，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ときに，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ときの支援を行う．

コードクローンの検出部分として，CCFinder を用いているが，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コードクローンを含むファイルのリストを表示する．

検索ツールの構成図を図 1 に示す．図のように本ツールはユーザの入力情報から CCFinder へのオプション，入力ファイルを作成し，検索結果をファイルの位置がわかるように表示を行う．

検索ツールの画面例を図 2 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．

コードクローンの検索が終了すると，結果を表示するために新たなウィンドウが開かれ（図 2 の手前のウィンドウ），ユーザが指定したコード片のコードクローンを含むファイルのリストが表示される．ユーザはこのリストからファイルを選択して（図中，下線が引かれているファイル名をクリックする），そのファイルのテキストを表示させ，コードクローンを確認することができる（検索されたコードクローンはハイライトで示される）．

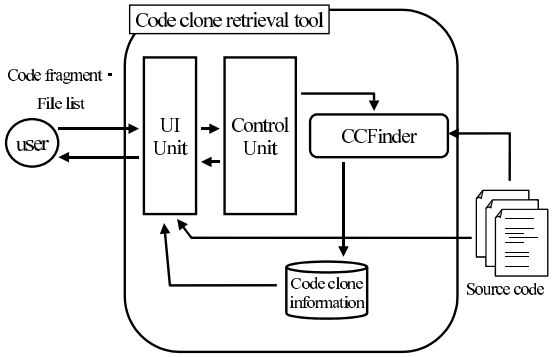

図 1 構 成 図
Fig. 1 System configuration.

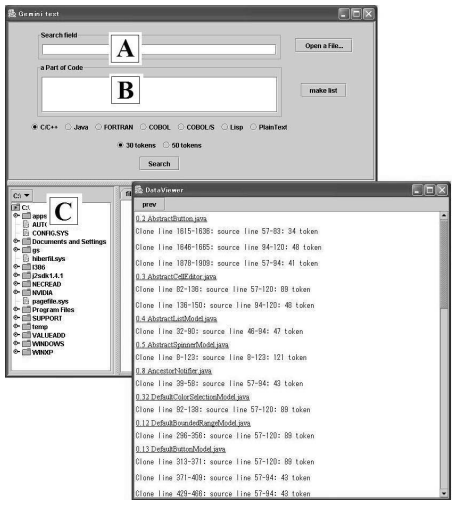

図 2 画 面 例
Fig. 2 Snapshot.

3.3 適 用 例

開発したツールを用いた擬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（C 言語，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（Result）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一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 箇所はすべ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．

この二つの結果を f 値（F-value）[5] を用いて比較する．f 値とは完全性（Recall）と効率性（Precision）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．

表 1 検索結果
Table 1 Retrieval result.

	Result	Time
grep	243lines(58places)	less than 1sec
Tool	17places	8sec

表 2 結果比較
Table 2 Comparison.

	Recall	Precision	f-value
grep	95%	34%	0.50
Tool	81%	100%	0.90

したがって，f 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良い結果を示している．

4. むすび

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，本ツールの有効性を示した．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. Sixth International Symposium on Software Metrics (METRICS99), pp.292–303, 1999.

[2] I.D. Baxter, A. Yahin, L. Moura, M. Sant’Anna, and L. Bier, “Clone detection using abstract syntax trees,” Proc. 14th International Conference on Software Maintenance (ICSM 98), pp.368–377, 1998.

[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. Softw. Eng., vol.28, no.7, pp.654–670, 2002.

[5] 徳永武信，情報検索と言語処理，東京大学出版会，2002.

（平成 15 年 4 月 14 日受付，7 月 7 日再受付）