コードクローン間依存関係に基づいたCプログラムのリファクタリング支援

荻野真也† 太田剛‡
静岡大学大学院 情報学研究科† 静岡大学 情報学部‡

1 研究の背景

ソフトウェアライフサイクルにおいて「保守」は非常に重要である．ひとつのソフトウェアを長期間使用するためには，理解しやすいプログラムを設計・作成し，後々の保守を容易にすることが大切になる．

しかし，そのようなプログラムを実際に作成することは困難なことであり，そのため，保守作業も非常に難しい工程となる．保守作業では完成したソフトウェアのソースコードを読み，理解した上で変更を加えなければならない．この保守作業を少しでも容易にするために，しばしば，完成したプログラムの振る舞いを一切変えず，ソースコードのみを理解しやすいように変更する作業が行われる．これを「リファクタリング」という．

オブジェクト指向言語でのリファクタリングでは，オブジェクトの概念を利用した方法論がいくつか提案されている．また，それらの方法とメソッド間の依存関係を組み合わせたリファクタリングも存在する．ところが，手続き型言語ではオブジェクトの概念がないため，リファクタリング手法があまり存在していない．そこで本研究では，手続き型言語のC言語で，関数とコードクローンの関係に注目し，それらの依存関係に基づくリファクタリングを行う方法についての提案と，その実験結果について述べる．

2 コードクローン

コードクローンとは，ソースコードの中で全く同じかまたは類似している部分である．

クローンが発生する典型的な理由は，開発者がソースコードをコーディングするときに，ソースコードをテキストエディタでコピー&ペーストすることである．

クローンがソフトウェアの保守作業において問題になるのは，クローンを持つソースコードは修正が困難なためである．例えば，ソースコード中のバグを発見して修正する場合に，もし修正すべき部分がクローンのコード断片になっていれば，そのすべてのコード断片について，修正を行う必要がある．これは，バグの修正ばかりではなく，ソフトウェアの機能拡張や移植など，ソースコードの修正を伴うすべての作業について当てはまる．作り込まれたクローンは，数ヶ月もするとその存在が忘れられてしまい，特に大規模なソースコードであれば，容易には発見できなくなる．そのため，ソフトウェアを修正するときになって，クローンのコード断片のいくつかを修正し損ない，「修正したはずのバグが特定の条件下で再現する」といった事態を引き起こす．

このような理由により，一般には，ソースコード中のクローンは少ない方がよいとされている[1].

3 研究の概要

本研究の目標はC言語のソースコードから検出されたコードクローンのリファクタリングを機械的に可能とするツールの作成である．しかし，コードクローンを発見したとしても，それらすべてのコードクローンをリファクタリングするべきとは限らない．どのコードクローンをリファクタリングするべきなのかという判断をする必要がある．

コードクローンの対をひとつずつ集約するよりも，ある程度の意味を持つコードクローンの集まりを集約する方がリファクタリングとして有用であり，さらにそれらを機械的に行った時に，人が見て理解しやすいソースコード作ることができると思われる．

本研究ではソースコードとコードクローンの依存関係を元にしたリファクタリング支援を行う．ここでいう依存関係は，C言語プログラムの関数呼び出し関係と，コードクローンの関係である．

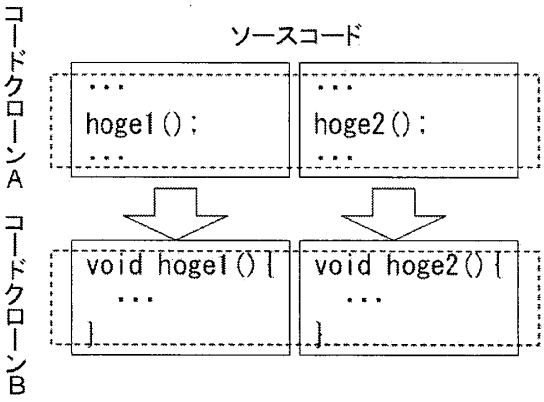

図1．依存関係図

図1のように，ソースコード中の関数呼び出し部分がコードクローンであり，さらに，その関数を定義している部分も別のコードクローンだった場合を依存関係として定義する．Java言語においてはこの依存関係のあるコードクローンが存在している[2]．本研究ではC言語ソースコードでこの依存関係を発見した後，これらに関してファイルの再編成などのリファクタリングを行い，保守性の向上を目指す．

4 システム概要

本システムは，リファクタリング対象のC言語ソースコードにおいて，その対象ソースコードのコードクローン情報と関数の呼び出し関係から，依存関係を持つコードクローンを探し出し，それらのリファクタリング案をユーザーに提示する．

A refactoring supporting tool for C programs, based on relations between code clones.
†SHINYA OGINO, Graduate School of Informatics, Shizuoka University
‡TSUYOSHI OHTA, Faculty of Informatics, Shizuoka University

（1） コードクローン情報の検出

本研究ではコードクローンの検出に CCFinderX[3][4] を使用した．さらに CCFinderX から出力されたコードクローン情報を Athanor で CSV 形式に変換してシステムへのインプット情報とした．

（2） 関数呼び出し関係の検出

対象ソースコードから関数呼び出し部分の情報と，関数を宣言している本体部分の情報の２つを検出する．ここでは，ファイルパス・行番号・関数名・関数呼び出しなのか，関数本体なのかを調べ CVS 形式で出力する．

（3） 依存関係の検出

得られたコードクローン情報と，関数呼び出し情報を照合し，関数呼び出し関係を含んでいるコードクローンを検出する．

その後，検出された依存関係を持つコードクローンを整理して表示する．関数呼び出しを含むコードクローンと，関数本体を持つコードクローンとの参照関係を依存関係表として表形式で表す．

（4） リファクタリング候補の提示

関数呼び出し関係による依存関係を持つコードクローンを依存関係表にまとめた後，それらの表にまとめられたコードクローンをそれぞれリファクタリングし，その結果をユーザーに提示する．

関数の本体部分と，呼び出し部分の機能を，新たに作成したファイルにまとめることで依存関係のリファクタリングを行う．

新たに作成したファイルにコードクローンの集合を集約する手法として，C 言語のマクロを使用する．コードクローンは基本的には同じソースコードであり，少しの差異があるだけなので，それらの差異をマクロの引数として扱うことにより，複数のコードクローンを１つにリファクタリングする．

また，オリジナルのソースコードでは新たなファイルを参照する関数を作成し，その作成した関数の中でリファクタリングしたマクロを適用する．また関数の依存関係を新たなファイルでも維持するために，作成したファイル内で，対象コードクローンの関数の呼び出し関係を持つようにリファクタリングを行う．

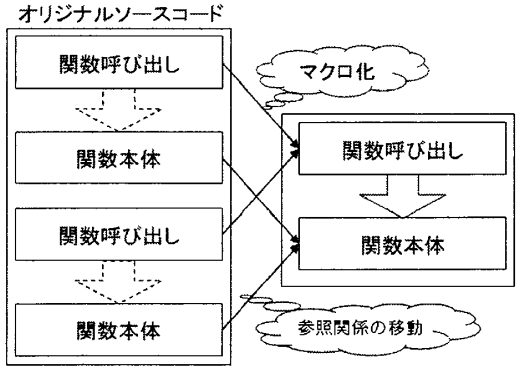

図２．参照関係の移動とマクロ化

5 実験結果

本方式の確認のため linux-2.6.18 の kernel ディレクトリを使用した．ここでは 151 個のクローンの集合，1480 個の関数本体，3834 個の関数呼び出しが存在し，これらから７個の依存関係を持つコードクローンの集合を検出した．この内６個の依存関係がマクロを用いた手法でリファクタリング可能であった．

リファクタリング不可能だったコードクローンは，長い switch 文の一部分をコードクローンとして捉えたものだった．これらは，関数のようなブロックとしてのまとまりを持っておらず，リファクタリング不可能だった．

リファクタリングを行った結果，新たに作成したファイルの中に，依存関係を持つコードクローンを集約することができた．作成したファイル内でコードクローンをマクロ化した関数を作成し，オリジナルのソースコードからはそれらの関数を参照することでリファクタリングができた．

6 今後の課題

本システムのアウトプットは完全なソースコードではなく，ひとつのリファクタリング候補としてユーザーに提供する．ユーザーはこれらを参考にリファクタリングを行うか否かの判断も含め，ソースコードの改善すべき可能性を持つ部分を把握することができる．また，コードクローンの依存関係から，元のソースコードがどのようにして作成されたものか意図を探るためのヒントとして使用することも可能である．

今回の実験では対象プログラムをひとつでしか行っていないが，さらに多くの C プログラムで実験を行うことで，コードクローンと関数呼び出しの依存関係を持つソースコード辺に何らかの共通点を発見できるかもしれない．それらが発見されたとしたら，それらは多くの人が C 言語プログラムによってコードクローンを作成してしまうパターンとなる．そして，マクロ化によるリファクタリングを行うことによって，それらのパターンを解消するための，ある種のデザインパターンのようなものを提供できるかもしれない．

参考文献

[1]神谷年洋：“コードクローンとは，コードクローンが引き起こす問題，その対策の現状”，電子情報通信学会誌，vol. 87, No. 9, pp. 791-797, (2004).

[2]吉田則裕，肥後芳樹，神谷年洋，楠本真二，井上克朗：“コードクローン間の依存関係に基づくリファクタリング支援”，電子情報通信学会誌，vol. 48, No. 3, pp. 1431-1442, (2007).

[3]Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue:“ CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code”, IEEE Trans. Software Engineering, vol. 28, no. 7, pp. 654-670, (2002).

[4]CCFinder Official Site. http://www.ccfinder.net/