リファクタリングのための類似コード片抽出ツール Gemini

肥後 芳樹[†] 神谷 年洋[‡] 楠本 真二[†] 井上 克郎[†]

Gemini: A Duplicated Code Detection Tool for Refactoring

Yoshiki Higo[†], Toshihiro Kamiya[‡], Shinji Kusumoto[†], and Katsuro Inoue[†]

1. はじめに

近年，保守工程はソフトウェア開発で最もコストを要する過程であると指摘されている．コードクローンはソフトウェア保守を困難にしている一つの要因である[1]．コードクローンとはソースコード中に存在する同一，または類似したコード片のことであり，コピーアンドペーストによる修正，拡張などにより発生する．ソースコードにバグが含まれている場合，該当部分のコードクローンとなっている全ての部分に対して修正の是非を考慮する必要がある．特に大規模開発において，これは非常に手間のかかる作業となる．

組込み開発のソフトウェアのソースコードは近年大規模になっており，また，実行時のパフォーマンスを稼ぐためにループを意図的に展開するためにコードクローンが作り込まれる場合があるなど，コードクローンの発生は避けられない問題となっている．本稿では，我々が開発してきたコードクローン検出ツール CCFinder[2]およびグラフィカルユーザーインターフェイスを備えたコードクローン分析環境 Gemini[3]を紹介する．

2. CCFinder

CCFinder は与えられたソフトウェアのソースコード中に存在するコードクローンを検出し，その位置を出力する．コードクローン検出手順は大きく 4 つの過程からなる．

ステップ 1(字句解析)：ソースファイルを字句解析してトークン列に変換する．入力ファイルが複数の場合には，個々のファイルから得られたトークン列を連結し，単一のトークン列を生成する．

ステップ 2(変換処理)：実用上意味を持たないコードクローンを取り除くこと，及び，些細な表現上の違いを吸収することを目的とした変換ルールによりトークン列を変換する．例えば，この変換により変数名は同一のトークンに置換されるので，変数名が付け替えられたコード片もコードクローンであると判定することができる．

ステップ 3(検出処理)：トークン列の中から指定された長さ以上一致している部分をコードクローンとして全て検出する．

ステップ 4(出力整形処理)：検出されたコードクローンのソースコード上での位置を出力する．

†大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
‡科学技術振興機構さきがけ
Presto, Japan Science and Technology Agency

3. Gemini

Gemini は GUI ベースのコードクローン分析環境であり，ユーザに以下のビューを提供し，対話的な解析を可能としている．

3.1. クローン散布図

クローン散布図はソースコードのどの部分にコードクローンが存在するのかを示す図である(図 1)．一目でソースコード中のコードクローンの分布状況がわかるので，

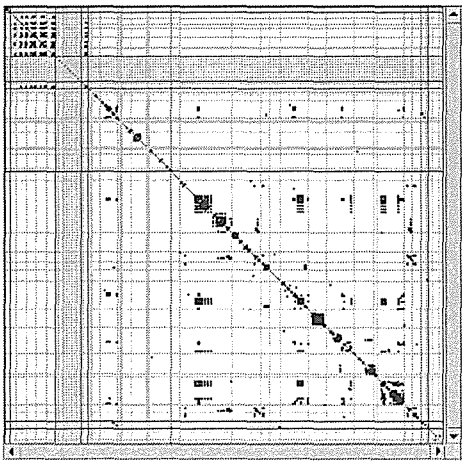

図 1 クローン散布図の表示例

Fig. 1 A sample display of clone scatter plot

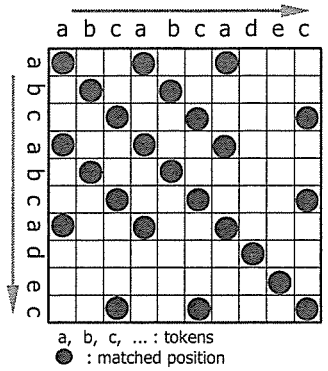

図 2 クローン散布図の模式図

Fig. 2 Model of clone scatter plot

解析の初期段階では非常に有効な解析手段となりうる. 図 2 はクローン散布図を説明するための模式図である. 図では, a, b, c などの文字がトークンを表している. 縦軸と横軸にはソースコード中のトークンが出現順に配置される. 格子内の黒い点はその縦軸のトークンと横軸のトークンが等しいことを意味している. コードクローンは, 散布図においてある一定以上の長さを持った線分となる.

3.2. メトリクスグラフ

メトリクスグラフを用いることにより, ユーザはコードクローンを定量的な特性に基づいて選択することができる(図 3). 表示されるメトリクスは, 以下のように定義されている.

RAD(C): コードクローンのディレクトリ階層における分散度を示す. すなわち, あるコード片とコードクローンになっている全てのコード片の集合 C(以降「クローンクラス」と呼ぶ)について, それらコード片を含むソースファイルがディレクトリ階層内の広い範囲で分散していると値が大きくなる. RAD の値が大きいクローンクラスは, ユーティリティ関数などのような様々な場所で使われるルーチンがクローンとなっていることを示す.

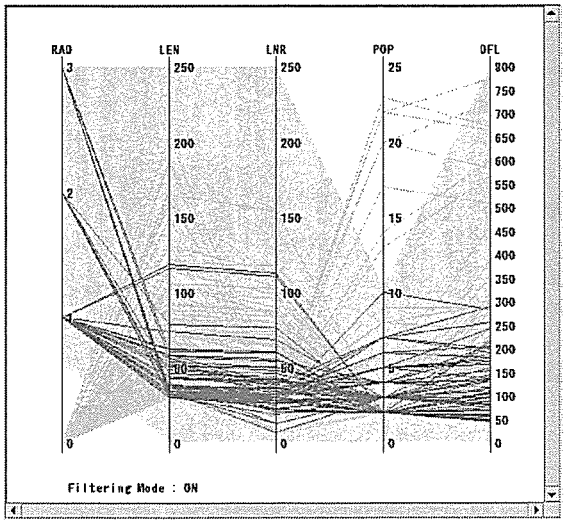

図 3 メトリクスグラフの表示例

Fig. 3 A sample display of metric graph

LEN(C): クローンクラス C に含まれるコード片の最大トークン長を表す.

LNR(C): クローンクラス C に含まれるコード片から, 繰り返し部分(代入文の並びなど)に含まれるトークンを取り去ったときの最大トークン長を表す. 定義により, LNR(C) ≤ LEN(C). LNR が LEN に比べて著しく小さいコードクローンは, ループの意図的な展開などにより, 同じコード断片が繰り返されていることを意味する.

POP(C): クローンクラス C に含まれるコード片の数.

DFL(C): クローンクラス C に含まれるコード片に共通するロジックを実装するサブルーチンを作り, 各コード片をそのサブルーチンの呼び出しに置き換えた場合に減少が予測されるトークン数である. 概ね, LEN × POP に比例する.

4. まとめ

本稿ではコードクローン検出ツール CCFinder およびコードクローン分析環境 Gemini について説明した. これらのツールを用いて, ソースコードに含まれるコードクローンを検出し対話的に分析することで, 効果的にコードクローンを取り除くことが可能になる.

参考文献

[1] M. Fowler, *Refactoring: improving the design of existing code*, Addison Wesley, 1999.

[2] T. Kamiya, S. Kusumoto, and K. 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, (2002-7).

[3] Y. Ueda, T. Kamiya, S. Kusumoto, K. Inoue, "Gemini: Maintenance Support Environment Based on Code Clone Analysis", *Proc. 8th International Symposium on Software Metrics*, June 4-7, 2002.