ウインターワークショップ 2008 ・イン・道後　論文集　（平成 20 年 1 月）

欠陥検出を目的とした類似コード検索法

吉田則裕[†1]　石尾隆[†1]
松下誠[†1]　井上克郎[†1]

ソースコード中に欠陥が見つかると，その欠陥を含むコードの類似コードを探し，検査する必要がある．本稿では，このような場合に用いることのできる方法について考察し，それを踏まえて識別子の類似性に基づく類似コード検索法の構想を述べる．

Towards Retrieving Similar Code for Defect Detection

Norihiro Yoshida,[†1] Takashi Ishio,[†1] Makoto Matsushita[†1]
and Katsuro Inoue[†1]

In this paper, we discuss available methods for detecting defects caused by the same mistake. First we explain choices of methods that can be used in such situations and then propose a code retrieval framework based on similarity of identifiers in source code.

1. はじめに

ソフトウェア保守を困難にする要因の 1 つとして**類似コード**（コードクローン）が指摘されている[1)–4)]．類似コードは，既に開発された**コード片**（ソースコードの一部）のコピーとペーストによる再利用や定型処理の実装などが理由で作成される[5)]．ソフトウェアの保守を行っている際にソースコード中に欠陥が見つかると，その欠陥を含むコード片の類似コードを探し，検査する必要がある[6)7)]．しかし，ソフトウェア中の類似コードを人手で探すためには大きなコストが必要となる．また，同一の処理を実装したコード片であっても表現上の差異があることが多いため，全ての類似コードを人手で探すことは困難である．特に，大規模ソフトウェアが対象の場合，全ての類似コードを人手で探すことはより困難となる．

本稿では，まず欠陥を含むコード片の類似コードの自動検索に用いることができる方法として，grep を用いた検索やコードクローン検出法[1)–4)] を挙げ，それら方法について考察する．次に，それを踏まえて識別子に類似性に基づく類似コード検索法を提案する．手法の提案では，まず基盤部分を説明し，続いて拡張方法を述べる．現在，提案する検索法の実装を行っている．

†1 大阪大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

2. 既存の類似コード検索法

2.1 grep を用いた方法

grep を用いて欠陥を含むコード片の類似コードを検索する手順を以下に示す．

(1) 開発者は，欠陥と関連すると思われるキーワード（識別子，式など）を抽出する．
(2) そのキーワードを引数として grep を実行する．
(3) 開発者は，grep の出力結果を基に，キーワードを含むコード片を特定する．

この手順にしたがう開発者は，欠陥を含むコード片と検査すべきコード片の間で，識別子や式などのキーワードが共通しているという前提を置いている．しかし，識別子の同義語をはじめ，キーワードには様々な変化形が存在するため，検査すべきコード片を漏れなく検索結果に含めることは難しい．

2.2 コードクローン検出法を用いた方法

既に提案されている種々のコードクローン検出法[1)–4)] を用いて，欠陥を含むコード片のコードクローンを検出することができる．この方法は，コード片をクエリとして与えられるため，開発者がキーワードやパターンを考えなくて良いという利点がある．しかし，既存のコードクローン検出法の多くは，類似したトークン列や構文木を検出する手法[1)–3)] であり，例えばログ出力文や例外処理が追加されてしまったコード片は検索結果に含まれない．その他のコードクローン検出

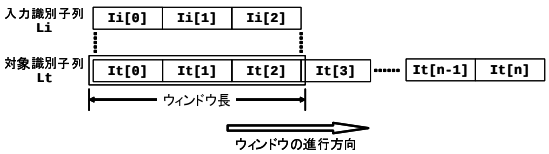

図 1 入力識別子列と対象識別子列の照合

法は計算時間が膨大であるため，大規模ソフトウェアに適用することは現実的ではない[4]．

3. 識別子の類似性に基づく類似コード検索法

本研究では，まず提案手法の基盤部分を実現した後，様々な拡張を行う予定である．そして，適用実験を通して各拡張の有効性を比較したいと考えている．

3.1 提案手法の基盤部分

入力コード片（クエリ）の類似コードを対象ソースコードから検索し，提示する手順を以下に示す．

手順 1（識別子の抽出） 入力コード片および対象ソースコードに含まれる全ての識別子を列挙し，それぞれ**入力識別子列**および**対象識別子列**とする．

手順 2（識別子列の照合） 入力識別子列と対象識別子列を照合し，入力識別子列と同じ識別子数でかつ類似する部分列を対象識別子列から抽出する．抽出された部分列を**類似部分列**と呼ぶ．

手順 3（類似部分列の順位付け） 類似部分列を入力識別子列との類似度によって順位付けする．

手順 4（コード片の提示） 順位付けにしたがって，類似部分列に対応するコード片を提示する．

手順 2 の詳細を述べる．図 1 は，入力識別子列 $Li = \left[Ii[0], Ii[1], Ii[2]\right]$ と対象識別子列 $Lt = \left[It[0], It[1], It[2]\right]$ の照合処理を表している．対象識別子列 Lt 中で照合の対象となる範囲を**ウィンドウ**と呼び，このウィンドウを 1 識別子単位で対象識別子の終端方向（図 1 の右方向）にスライドさせることで照合処理を行う．照合するか否かの判定基準は，入力識別子列とウィンドウ内の部分列の類似度を算出し，類似度が 0 より大きいか否かである．もし 0 より大きいならば，ウィンドウ内の部分列を類似部分列として抽出する．類似度は，入力識別子列 Li からなる集合を Si，ウィンドウ内の部分列 Lw からなる集合を Sw としたときに，式 1 で表される．

$$Similarity(Si, Sw) = \frac{2 * |Si \cap Sw|}{|Si| + |Sw|} \qquad (1)$$

手順 3 では，式 1 に基づいて，抽出された類似部分列に対して順位付けを行う．これにより，入力コード片と多くの識別子を共有するコード片が上位に順位付けされる．また，類似度が 0 より大きいコード片を抽出しているため，識別子を 1 つ以上共有するコード片であれば，検索結果に含まれる．

3.2 拡張方法

(1) 識別子照合法の拡張 例えば，同義語をはじめとする識別子の変化形に対応した照合法に拡張することが考えられる．変化形を考慮した類似度の定義に式 1 を変更することで，変化形が存在する場合でも，検索精度の低下を防ぐことができると考えられる．変化形を特定するためには，単語の共起性に基づいて同義語の判定を行う自然言語処理の手法[8] を応用できると考えられる．

(2) 順位付け法の拡張 例えば，検索結果に含まれたコード片にデータフロー解析など他のソースコード解析法を適用し，解析結果（データフロー等）の類似性を考慮した順位付けを行うことで，順位付けの有効性を高めることが考えられる．検索結果に含まれたコード片とその周辺のみを解析するため，計算にかかるコストは小さいと考えられる．

参 考 文 献

1) Baker, B.S.: Finding Clones with Dup: Analysis of an Experiment, *IEEE Trans. Softw. Eng.*, Vol.33, No.9, pp.608–621 (2007).
2) Baxter, I. D., Yahin, A., Moura, L., Anna, M.S. and Bier, L.: Clone Detection Using Abstract Syntax Trees, *Proc. of ICSM '98*, pp. 368–377 (1998).
3) Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A multi-linguistic token-based code clone detection system for large scale source code, *IEEE Trans. Softw. Eng.*, Vol.28, No.7, pp.654–670 (2002).
4) Komondoor, R. and Horwitz, S.: Using Slicing to Identify Duplication in Source Code, *Proc. of SAS 2001*, pp.40–56 (2001).
5) Kim, M., Bergman, L., Lau, T. and Notkin, D.: An Ethnographic Study of Copy and Paste Programming Practices in OOPL, *Proc. of ISESE 2004*, pp.83–92 (2004).
6) Zeller, A.: *Why Programs Fail*, chapter15 Fixing the Defect, Morgan Kaufmann Pub. (2005).
7) Li, Z., Lu, S., Myagmar, S. and Zhou, Y.: CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code, *IEEE Trans. Softw. Eng.*, Vol.32, No.3, pp.176–192 (2006).
8) Dagan, I., Lee, L. and Pereira, F. C. N.: Similarity-Based Models of Word Cooccurrence Probabilities, *Machine Learning*, Vol.34, No.1-3, pp.43–69 (1999).