5J-08

メソッドに対するコメント自動生成によるバグ発見の補助

十亀 眞怜 *　　斎藤博昭

慶應義塾大学理工学部情報工学科

1 背景と目的

Sridhara らによって Java メソッドのソースコードからメソッドの動作を要約したコメントを自動生成する手法が提案されている [3]．本研究は，Sridhara らの手法を拡張し，初心者に対し Scala 言語のプログラムに含まれるバグ発見の補助となるコメントの自動生成を行うことを目的とする．

2 提案手法

提案手法は，プログラム前処理部，メソッド名処理部，コメント生成部の三部から構成される．

2.1 プログラム前処理部

入力されたコンパイル可能な Scala プログラムのソースコードから抽象構文木を得るために，Scala コンパイラのプラグインを実装し，Scala コンパイラ内部で型付けが完了した直後の段階の抽象構文木を取得し，構文木中の各メソッド定義に対して Sridhara らの手法に基づいてメソッドの主要な動作に関わる構文木を抽出する．

2.2 メソッド名処理部

抽出された各構文木に含まれるメソッド呼出し構文に対して，Hill らの手法 [1] を用いてメソッド名を単語に分割した後に自然言語処理し，メソッド名及び引数から動詞 · 目的語 · 補助的な対象の三つ組を抽出する．Hill らの手法では，目的語及び補助的な対象はメソッド呼出しの仮引数名を用いていたが，本手法では Sridhara らの研究に基づいて，目的語及び補助的な対象はメソッド呼出しの実引数名を再帰的に解析したものを用いた．

2.3 コメント生成部

抽出された各構文木に対して，用意したテンプレートに基づきメソッド名処理部で抽出した三つ組を当てはめることで各構文木に対するコメントを生成し，それらを構文木間の関係に合わせて組み合わせることで最終的なコメントを生成する．本手法では，構文木中に表 1 に示すアルゴリズムの誤り (以下 バグ) の要因となり得る構造が現れた際にそのようなプログラムの構造を強調するテンプレートを用いた．

2.4 生成されるコメント例

ソースコード 1 にコメント生成対象のサンプルコードと図 1 に生成されたコメントを示す．図 1 に示したプログラムは，1 から引数として指定された整数までの各自然数を fizzbuzz 変換した結果の文字列を出力するためのプログラムであるが，5,7 行目において，正しくは + = の演算子を用いるべき箇所で = を用いてしまっている．

ソースコード上では演算子の誤りは一文字分の誤りであるが，生成されたコメントにおいては変数を上書き (Assign) しているか，既存の値を用いて更新しているか (Update) しているかという形式で区別されている．

ソースコード 1　サンプルコード

```
def printFizzBuzz(num: Int): Unit = {
  var buf = ""
  for (n <- 1 to num) {
    if (n % 15 == 0) {
      buf = "FizzBuzz"
    } else if (n % 5 == 0) {
      buf = "Buzz"
    } else if (n % 3 == 0) {
      buf += "Fizz"
    } else {
      buf += "␣" + n
    }
  }
  println buf
}
```

```
Take Int parameter num
Define buf as the String ""
for each item of Range from 1 to num as n
  Function that
    Take Int parameter n
      If n % 15 is 0 holds
          Assign  "FizzBuzz" to buf
        else
          If n % 5 is 0 holds
            Assign  "Buzz" to buf
          else
            If n % 3 is 0 holds
              Update buf by buf + "Fizz"
            else
              Update buf by buf + " " + n
println buf
```

図 1　生成コメント例

3 評価実験及び結果

3.1 評価実験

Scala 初心者及び経験者の大学院生 5 人が評価実験に参加した．

バグを含んでいるものとそうでないもの 7 つずつ合計 14 個のメソッドに対してそれぞれ生成したコメントを，ランダムな順序でメソッドの行うべき動作の説明とともに提示し，コメントの内容からメソッドにバグが含まれているかについて，有り · 無し · 判断ができないの選択をしてもらい，バグが有ると選択した被験者に対してはバグの内容についての説明を求めた．

その後，コメントの元となったソースコードを提示し，コメントのソースコードに対する正確性
(1: 全く正確でない · · ·5:とても正確である),
バグ発見に不必要な情報の有無
(1:全く含まれていない · · ·5:とても含まれている),
バグ発見に必要な情報の欠如
(1:全く欠如していない · · ·5:とても欠如している)

* MASATO SOGAME <poketo7878@gmail.com>

表 1　対象としたバグ及び対応するテンプレート例

対象とするバグ	テンプレート例
代入演算子 (=) と 更新を伴う代入演算子 (+ =, − = 等) の誤り	Assign **変数名** to **値** Update **変数名** by **式**
型を明示ししなかった変数が 型推論によって意図と異なる型が推論されている誤り	Define **変数名** as the **型名 値**
比較演算子の誤り (未満 < と 以上 <=)	**変数名** is less than or equal to **変数名**

についてそれぞれ 5 段階評価をしてもらった．また，コメントに対する意見について任意で回答してもらった．

3.2　実験結果

3.2.1　バグの有無の判定

表 2　バグの有無の判定結果

バグの有無	正しく判定された割合
無し	75%
有り	67%

バグ有無の判定結果において，正しく判定されなかったプログラムについて確認してみると，代入演算子によるバグは正しく判定されているのに対し，型推論及び比較演算子の誤りに起因するバグは正しく判定されていなかった．

これらのバグの判定の結果について考察すると，代入演算子の誤りはコメントにおいて文頭の形態で差異が表現されているのに対し，型推論によるバグ及び比較演算子のバグについては，文中において差異が現れているのみであるためであると考えられる．

正しく判定されなかったコメントに対する意見においても，“読んでいると自然と読み飛ばしてしまう事があった” と指摘されていた．

このように，バグが発見されるためには，コメントの文頭の形態がはっきりと異なるか，重要な箇所として**視覚的に**強調されている必要があると考えられる．

3.2.2　コメントの正確性

表 3　コメントの正確性 (5 段階評価)

	バグ判定		
バグの有無	有り	無し	わからない
有り	4.7	4.6	3.5
無し	2.3	4.5	4.5

コメントに対する意見及び対象のプログラムを確認してみると，“プログラムのソースコード自体には表示されていない else 節がコメントに挿入されている” という意見があった．これは，本システムを Scala コンパイラのプラグインとして作成したために，else 節がソースコード自体に存在しない場合もコンパイラ内部では抽象構文木として存在するためにコメントに出力されている事による．

3.2.3　コメントに不必要な情報が含まれているか

コメントに対する意見では，コメントのインデント及び，前述のソースに存在しない else 節などのコメントの形式に対する指摘がなされていた．これらのコメントの形式の不備に対する指摘が，コメントに不必要な情報が含まれているとして評価に反映されたと考えられる．

表 4　コメントに不必要な情報が含まれているか (5 段階評価)

	バグ判定		
バグの有無	有り	無し	わからない
有り	3.0	2.3	2.8
無し	3.0	3.3	1.0

3.2.4　コメントに必要な情報が欠如しているか

表 5　コメントに必要な情報が欠如しているか (5 段階評価)

	バグ判定		
バグの有無	有り	無し	わからない
有り	2.6	3.0	3.2
無し	3.0	2.8	2.5

コメントに対する意見を確認すると，コメント内で呼び出されているメソッドの動作についての情報をコメントに含んでいてほしいという意見が多く見られた．呼び出しているメソッドからの返値の扱いを誤っているバグの発見を想定すると，コメント内で呼びだされているメソッドの動作の概要についてもコメントに含めておく必要があると考えられる．

4　結論 · 今後の課題

本稿では，ソースコードに対するコメント生成によるバグ発見の補助の手法を提案した．今後の課題としては，バグの要因となり得るプログラムの構造がより認識されやすいコメントのテンプレートの考案が挙げられる．

参考文献

[1] Emily Hill, Lori Pollock, and K. Vijay-Shanker. Automatically capturing source code context of NL-queries for software maintenance and reuse. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 232–242, Washington, DC, USA, 2009. IEEE Computer Society.

[2] Paul W. McBurney and Collin McMillan. Automatic documentation generation via source code summarization of method context. In *Proceedings of the 22nd International Conference on Program Comprehension*, ICPC 2014, pages 279–290, New York, NY, USA, 2014. ACM.

[3] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker. Towards automatically generating summary comments for Java methods. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 43–52, New York, NY, USA, 2010. ACM.