大規模言語モデルを用いてメソッドのコメントとソースコードの整合性を確認する手法の提案

2020SE093 武田思美
指導教員：名倉正剛

1　はじめに

ソースコードに記述したコメントは，他の開発者がプログラムの動作や記述を理解する場合に理解の助けになる．そして一般的にコードの概要や意図，クラスや変数の制約，そのコードが必要な理由などを記述する．コメントによりコードを記述した開発者がそのコードを記述した意図を，開発者間で共有できる．

2　コードとコメントの整合性に関する課題

コメントにソースコードに関する情報が欠落していたり，コメントとソースコードに整合性がなかったりすると，開発者のソースコード理解を妨げる．ソースコード 1 は JSON の文字列として受け取ったドキュメントのデータ行数を数えて返すメソッドである．そしてこのプログラムには “Counts the number of lines.” というコメントが記述されている．

コメントに記述すべきこと，記述すべきでないこと，および誤解されないようなコメントを記述するための方法について経験的に紹介されている文献がいくつも存在する．例えば参考文献 [1][2] には関数の入出力についての説明を記載すべきであることが，参考文献 [3] には関数の名前とコメントに同様の情報しか含まれていない場合に対象や条件などを補足すべきであることが述べられている．これらに従うと，このプログラムに記述されたコメントには，関数の入出力や，何に対する行数を数え上げる処理であるかが明確に記述されていない．この場合，メソッドの処理対象や処理の詳細を把握することは難しい．

3　提案手法

3.1　概要

本研究では，メソッドに記述されたコメントがソースコードと整合しているか確認する手法を提案する．メソッドのコメントにしたがって生成したソースコードが，コメントの記述されたメソッドと同一の機能を実現するのであれば，コメントがソースコードに整合していると判断する．なおここでコメントからのソースコード生成には大規模言語モデルを利用する．また機能の同一性の判断は，自動テストケース生成手法によって生成したテスト結果の比較により行う．

ソースコード 1　コメントが記述されたメソッドの例

```
1 / * Counts the number of lines. * /
2 public static int getRowCount(String json) throws
      JSONException{
3   JSONArray array = new JSONArray(json);
4   return array.length();
5 }
```

3.2　大規模言語モデルの利用

大規模言語モデルは大量のデータで学習した自然言語処理モデルであり，近年では Transformer ベースの言語モデルが提案されている [4]．提案手法ではメソッドのコメントから，大規模言語モデルである GPT [1] を利用しソースコードを生成する．メソッドにどのようなコメントを記述するかは，開発者により異なる．そこで自然言語処理言語モデルによってコメントを解釈させてコードを生成させた結果，開発者が記述したコードと同等の機能のコードを生成できれば，必要十分なコメントが記述されているとみなす．

3.3　手順

提案手法では，図 1 に示す次の手順によってコメントとソースコードの整合性を確認する．

① ソースコードを解析し，整合性を確認する対象のメソッドに記述されたコメントとメソッドのインタフェース情報を抽出する．

② ①で抽出したコメントとインタフェース情報からメソッドを生成する．

③ 生成したメソッドを元のプログラムの対象メソッドと置き換えた新たなプログラムを生成する．

④ 元のプログラムと③で生成したプログラムから，それぞれテストケースを生成する．

⑤ ④で生成したテストケースを利用し，元のプログラムと③で生成したプログラムに対してテストを実行する．

⑤で出力された実行結果から，開発者には実行に失敗したテストケースとその実行結果を，プログラムコードとコメントの整合性の確認結果として提示する．これによりコメントの不整合の内容を把握できるようにする．

②においてソースコード 1 のコメントを利用して GPT によって生成したメソッドを，ソースコード 2 に示す．ソースコード 1 では JSON の文字列が示すデータの行数を数えることを，“Counts the number of lines.” というコメントによって表現していた．このコメントに従うと，行を数えるという処理しか解釈されず，JSON 文字列が対象であるとは理解できない．よってこのコメントから GPT は，単純に受け取った文字列の行数のカウントを，改行文字の個数を数えることにより実現するメソッドを生成した．

④において元のプログラムから生成したテストケースの一部を，ソースコード 3 に示す．このテストケースは name と age を表すキーと値の組を 1 つの行に表した

[1] OpenAI GPT-3.5 Turbo, GPT API を使用

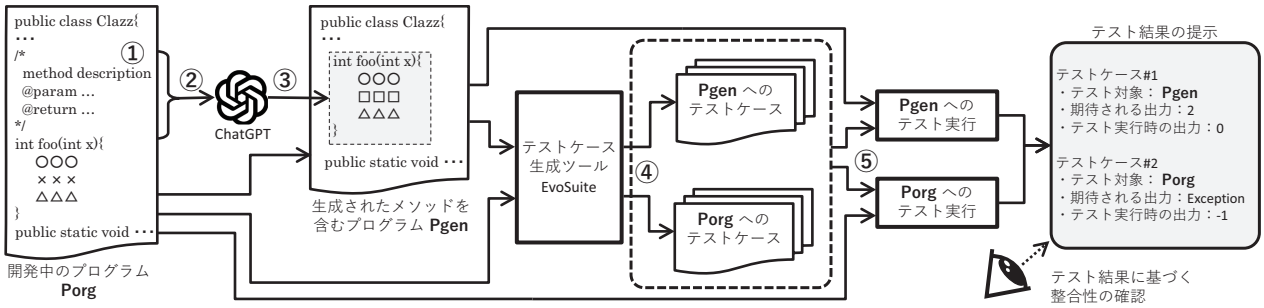

図 1 提案手法の概要（丸囲み数字は 3.3 節に記載した手順に対応する）

JSON の文字列を 2 データ分，引数として与えている．ソースコード 1 のプログラムは文字列を JSON ドキュメントとして評価したときのデータ行数を返却するメソッドであったので，このテストケースでは 2 が返却されることを期待している (ソースコード 3 の 6 行目)．しかし，② で生成されたソースコード 2 のメソッドでは文字列の行数を数えるため，期待通りの動作にならずテスト実行が失敗する．

4 評価

OSS プロジェクトに存在するメソッドを利用して，提案手法によりコメントとメソッドのソースコードの整合性を確認できるかどうか実験した．GitHub に存在するプロジェクトをスター順に取得し，そのうちメソッドコメントの多いプロジェクト上位 2 件の 86 個のメソッドを対象として実験した．提案手法の実行により整合性を確認可能だったメソッドは 86 個中 13 個で，このうちテスト実行に失敗してコメントとコードが整合していないことを確認できたのが 12 個であった．残りの 1 個はインタフェース情報に記述された返却値の型の特定のクラスが何かしらの特定のデータを返却することを暗黙的に示していたため，提案手法によってテスト実行に成功し，コメントとコードが整合していると判定した．

5 考察

提案手法によってコメントとソースコードの整合性について確認できることがわかったが，コメントのどのような記述が原因でテスト実行の成否が異なったのかを理解することは難しいため，コメントのどのような記述からコードを生成したのかを説明することができる手段が必要であると考えられる．

ソースコード 2 コメントから生成したメソッドの例

```
1  / * Counts the number of lines. * /
2  public static int countLines(String inputString) {
3      if (inputString == null || inputString.isEmpty()) {
4          return 0;
5      }
6      int count = 1;
7      int length = inputString.length();
8      for (int i = 0; i < length; i++) {
9          if (inputString.charAt(i) == '\n') {
10             count++;
11         }
12     }
13     return count;
14 }
```

ソースコード 3 生成されたテストケースの例

```
1 @Test(timeout = 4000)
2 public void test1()  throws Throwable  {
3   int int0 = Countlinesgpt.getRowCount
4           ("[{\"name\":\"Alice\",\"age\":25},
5                 {\"name\":\"Bob\",\"age\":30}]");
6   assertEquals(2, int0);
7 }...
```

6 関連研究

コメントとコード間の整合性を評価する指標が提案されている [5]．これはメソッドコメントで使用される用語とメソッド本体で使用される用語の重複を測定するもので，重複度が高いほどそのコードの可読性が高いと想定されているが，参考文献 [1][2][3] に記述されているような内容はメソッド内の識別子だけからでは表現できない．したがって，適用範囲が限定的であるといえる．

7 おわりに

本研究ではコメントから生成したプログラムコードと元のコードとの同一性を判断した結果を提示することによって，開発者によるプログラムコードとコメントの整合性確認を支援する手法を提案した．

参考文献

[1] D. Boswell, T. Foucher, 角征典 (訳). リーダブルコード — より良いコードを書くためのシンプルで実践的なテクニック. オイラリー・ジャパン, 2019.

[2] T. Long, 秋勇紀 (訳), 高田新山 (訳). Good Code, Bad Code ~ 持続可能な開発のためのソフトウェアエンジニア的思考. 秀和システム, 2023.

[3] 石川宗寿. 読みやすいコードのガイドライン 持続可能なソフトウェア開発のために. 技術評論社, 2022.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, Vol. 33, pp. 1877–1901, 2020.

[5] S. Scalabrino, M. Linares-Vasquez, R. Oliveto, and D. Poshyvanyk. A comprehensive model for code readability. *Journal of Software: Evolution and Process*, Vol. 30, No. 6, pp. 1–29, 2018.