変数名に着目したリファクタリング支援ツール MCC の現状と課題

田上　諭 [a)]・片山　徹郎 [b)]

Current Status and Issues of Refactoring Support Tool MCC Foucsing on the Naming of Variables

Satoshi TANOUE, Tetsuro KATAYAMA

Abstract

This research has implemented a prototype of refactoring support tool MCC(Make Clean Coder) which focuses on the naming of variables. This prototype helps to describe a clean code by static analysis for the source code written in C language. And, it can help to reduce factors that prevent programmers understanding the source code when they modify it by pointing out improper variable names. We applied some source codes written in C language to the prototype, and confirmed that it works properly. By using this prototype, because it can reduce reduction of time to understand the source code, programmers can shorten the coding time, lower the possibility of embedded bugs, and decrease in the time required to add functions. In this paper, we describe the current status and issues of MCC.

Keywords: Static Analysis, Refactoring, Clean Code,Variable Name

1. はじめに

近年，ICT の進化とともに社会における情報システムの担う役割が増加している．そのため，システム障害やソフトウェアに不具合がもたらす経済的，社会的影響は計り知れないものとなっている．このような背景か，高品質なソフトウェアが求められるようになった．

ソフトウェアの品質を改善･維持するための方法としてリファクタリングがある．リファクタリングは，ソフトウェアの外部的振る舞いを保ったまま，ソースコードがより分かりやすくなるように変更することである [1)]．ソースコードに分かりにくい変数名があると，どのような処理をしているのか分かりにくくなる [2)]．その影響により，ソースコードを修正する際に，必要な処理を消してしまったり，意図された使用用途とは違うように変数や関数を使ってしまったりして，バグを混入させることが起こりうる．

そこで本研究では，コードの品質改善の支援を目的として，リファクタリング支援ツール MCC(Make Clean Coder) を試作した．本稿では，この MCC の現状と課題について述べる．MCC は，C 言語で書かれたソースコードに対して静的解析を行い，変数名の命名に着目したクリーンコードの作成支援を行うツールである．MCC は，命名が適切でない変数名を指摘できる．

a)　工学専攻 機械・情報コース大学院生

b)　情報システム工学科准教授

なお本研究において，命名が適切でない変数名とは，変数名が 1 文字である場合や，変数名が辞書に存在しないものである．辞書データには，実験用に公開されている辞書であるデジ蔵 Web サービスを利用する [3)]．MCC によって指摘された変数名を修正していくことにより，コードを修正する際の理解の妨げになる要因を減少させられる．これにより，ソースコードを理解するための時間が減少し，コーディング時間の短縮，バグ混入の可能性の減少，機能追加に必要な時間が図れる．

2. MCC の外観

MCC の外観を，図 1 に示す．MCC はメニューバー，編集エリア，表示エリアから構成されている．以降，それぞれ説明する．

2.1 メニューバー

MCC のユーザーは，MCC の機能をメニューバーからメニューアイテムを選択することによって，利用できる．ユーザーがメニューバーの「ファイル(F)」をクリックすると，ファイルメニューを表示する．ファイルメニューから，下記に示す 2 つの機能を利用できる．

- ファイルを開く
- ファイルを保存する

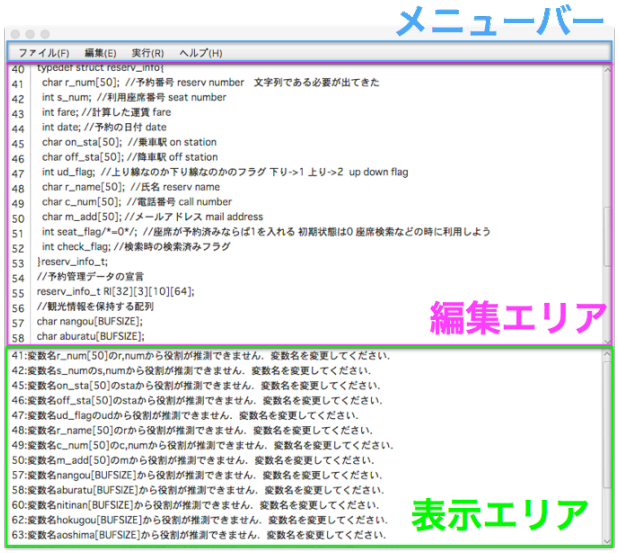

図 1. MCC の外観

ユーザーがメニューバーの「編集(E)」をクリックすると，編集メニューを表示する．ユーザーは，編集メニューから，下記に示す 2 つの機能を利用できる．

- 編集エリアをクリアする．
- 表示エリアをクリアする．

ユーザーがメニューバーの「実行(R)」をクリックすると，実行メニューを表示する．ユーザーは，実行メニューから，下記に示す 1 つの機能を利用できる．

- 変数名を解析

2.2 編集エリア

編集エリアは，ソースコードを編集できるエリアである．ユーザーが，ファイルメニューから「ファイルを開く」をクリックすると，MCC は編集エリアにファイルの内容を表示する．編集エリアでは，簡易的なエディタ機能がある．

2.3 表示エリア

表示エリアは，静的解析した結果を表示するエリアである．ユーザーが，実行メニューから「変数名を解析」をクリックすると，MCC は変数名を解析した結果を表示エリアに表示する．

3. MCC の機能

MCC は以下に示す 5 の機能をもつ．

- ファイルを開く
- ファイルを保存

```
valiableDeclaration::= <type> <name> <expr> (‘,’ <name> <expr>)* ‘;’
name::= <camellCaseName> | <snakecaseName> | <commonName>
camellCaseName ::= <commonName>(<largeLiteral><commonName>)+
snakecaseName::= <commonName>(‘_’<commonName>)+
commonName::= (<smallLiteral>(<digits>)?)+
expr ::=(<array>)*(‘=’<digits>)?
array::=‘[’(<smallLieral>|<digits>|(<largeLiteral>)+)?’]’
type::=“double”|”float”|(“unsigned”)? (“int”|”long”|”short”|”char”)
smallLieral::= [‘a’-’z’]+
largeLiteral::= [‘A’-’Z’]
digits ::=[‘0’-’9’]+
```

図 2. MCC が変数名の抽出に利用している構文解析器の EBNF

- 変数名を解析
- 編集エリアをクリア
- 表示エリアをクリア

MCC の機能の 1 つである「変数名を解析」について説明する．変数名を解析機能では，ソースコードの変数宣言部に着目して，変数名を静的解析する．変数名が辞書にが存在しないものや 1 文字であれば不適切な変数名だと判断し，表示エリアに「行番号:変数名~から変数名の役割が推測できません．変数名を変更してください」と警告を表示エリアに表示する．変数名がスネークケースやキャメルケースを用いた複合語の場合は，複合語の元になっている単語それぞれを辞書で検索する．それぞれの単語が辞書に存在しない場合や 1 文字である場合は，不適切な変数名と判断し「行番号:変数名~の~から変数名が表示できません．変数名を変更してください」と警告を表示エリアに表示する．

4. 実装

MCC の機能の 1 つである「変数名を解析」は，ソースコードに存在する変数名を抽出し，その後，変数名が適切かどうか辞書を用いて判断する．変数名を抽出するための方法及び変数名が適切かどうかの判断の方法について説明する．

4.1 変数名の抽出

ソースコードからの変数名の抽出には，構文解析器を用いる．構文解析器の作成には，JavaCC を利用する [4]．JavaCC は EBNF(Extended Backus–Naur Form)と似た文法で文法ファイルを記述すると，その文法ファイルから構文解析を行う Java コードを生成できる．MCC が変数名を抽出するために利用している構文解析器の EBNF の記述を，図 2 に示す．この EBNF を JavaCC で記述し，アクションを追加することによって変数名が抽出できる．

```
<SearchDicItemResult xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://btonic.est.co.jp/NetDic/NetDicV09">
  <ErrorMessage/>
  <TotalHitCount>1</TotalHitCount>
  <ItemCount>1</ItemCount>
  <TitleList>
    <DicItemTitle>
      <ItemID>027022</ItemID>
      <LocID/>
      <Title>
        <span xmlns="" class="NetDicTitle">name</span>
      </Title>
    </DicItemTitle>
  </TitleList>
</SearchDicItemResult>
```

図 3. リクエスト URL にアクセスした結果得られた XML ファイル

4.2 変数名の命名が適切かどうかの判断

変数名の命名が適切であるかは, 辞書を用いて判断する. 変数名が辞書に存在しない場合は, 命名が適切でないとする. また変数名が 1 文字の場合は, 辞書検索をせずに不適切な変数名とする. 辞書検索には, デ辞蔵 Web サービスを利用する. デジ蔵 Web サービスを用いた辞書検索の例として, name の検索を示す. MCC ではリクエスト URL として, http://public.dejizo.jp/NetDicV09.asmx/SearchDicItemLite?Dic=EJdict&Word=name&Scope=HEADWORD&Match=EXACT&Merge=AND&Prof=XHTML&PageSize=20&PageIndex=0 を使用している. リクエストパラメータとして, Dic に EJdict, Word に name, Scope に HEADWORD, Match に EXACT, Merge に AND, Prof に XHTML, PageSize に 20, PageIndex に 0 を割り当てる. “name” を検索し, 得られた結果の XML ファイル を, 図 3 に示す. Java の XMLParser を用いてこの XML ファイルを解析し, TotalHitCount の値を取得する. TotalHitCount は, 検索した単語が辞書にいくつ存在しているかを示している. 2 以上である場合, 辞書に複数存在している. 単語が 0 である場合, 辞書に存在していない. TotalHitCount の値によって, 辞書に登録されているかどうかが分かる.

5. 適用例

MCC の機能の一つである, 「変数名を解析」が正しく動作することを検証するために, C 言語で書かれたソースコードの例に適用する. MCC で解析した実行結果の一部を, 図 4 に示す. 編集エリアの 41 行目は, 変数名が r_num[50]となっている. これは, r は辞書に存在しているが, 1 文字であり, num は辞書に存在しないので命名が適切でないと判断し「41:変数名 r_num[50]の r, num から役割が推測できません, 変数名を変更してください」と警告を表示エリアに表示していることができる.

同様に編集エリアの 42, 45, 46, 47, 48, 49, 50 行目も辞書に登録されていない文字があるので, 「変数名～から役割が推測できません, 変数名を変更してください」と警告を表示エリアに表示している. また, 57, 58 行目は, ローマ字で命名されており, 変数名を構成する単語が辞書に存在しないので警告を表示エリアに表示している.

図 4. MCC の実行結果の一部

6. 考察

MCC の有用性と, 関連研究について考察する.

6.1 有用性

本論文で試作した MCC は, C 言語で書かれたソースコードを読み込み, 静的解析を行うことができる.静的解析では, 変数名に対して, 適切に命名されているか辞書を用いて判断できる.これにより, 辞書に存在しない 単語がどこにあるのかを簡単に見つけることができる. さらに, スネークケースやキャメルケースの場合, 複合語のどの部分が辞書にない単語であるのかを判断できる.よって, これらの機能により, 1 文字で命名された 変数名や, 辞書

に存在しない単語の発見ができる．また，ソースコードを編集し保存できる．

これらの機能を持つ MCC を利用することによって，コードを修正する際の理解の妨げになる要因を減少させられる．これにより理解にかかる時間を少なくでき，コーディング時間の短縮，バグ混入の可能性の減少，機能追加に必要な時間の減少が図れる．

6.2 関連研究

C 言語の静的解析ツールは，多くの研究及び開発が行われている．

Splint は，セキュリティの脆弱さとプログラミングのミスを修正するために，C 言語のプログラムを静的にチェックするためのツールである [5]. Splint は未使用の宣言，型の不整合，定義する前での使用，到達不能コード，戻り値の無視，リターンが無い実行パス，無限 ループの可能性といった lint チェックを行う．

AdLint は，ソースコード中の信頼性や移植性に欠ける部分について警告メッセージを出力し，同時に，さまざまな品質メトリクスを測定することができる [6]. 品質メトリクスとして，ファイル内の文の数や，関数内の文の数，goto 文の数といったものを測定できる．

これに対して，今回試作した MCC は，辞書を使用して，変数名が適切に命名されているかどうかについて判断でき，変数名が適切でないものを指摘できる．

7. MCC の課題

MCC の実装を及び考察を行う上で，明らかになった課題について述べる．

- 対応していない変数名がある．
  ポインタ変数，構造体で定義された型の変数名を解析できない．これらの変数名は MCC が利用している構文解析器では，変数名の抽出できていため，変数名の解析ができない．これらの変数名の抽出ができるように，構文解析器に構文を追加する必要がある．現在は，JavaCC を用いて構文解析器を作成しているが，JavaCC と同じパーサジェネレータである ANTLR[7] には，C 言語用の文法ファイルが用意されているので，それを利用し構文解析器を作成することにより，変数名の抽出ができるようになる．
- 関数名に対応していない．
  関数名の命名が適切かどうかについては，判断する事ができない．これは，MCC が利用している構文解析器では，関数名の抽出ができないためである．変数名の問題と同様に，ANTLR に存在する C 言語用の文法ファイルを利用し，構文解析器を作成することにより，関数名の抽出が行えるようにできる．関数名は，命名が適切か判断する際に，関数名が動詞で始まっていないか判断する必要がある．
- 辞書に存在する場合でも，変数名が適切でない場合が存在する．
  辞書に存在する場合でも，変数名が適切でない場合がある．例えば，word_word のような場合である．このような変数名は，どのような役割をもつ変数名か分からないので，理解容易性の観点からは不適切である．辞書で検索するだけでなく，変数名の並びや品詞を考慮して命名が適切か判断するように変更する必要がある．形容詞だけの変数名かどうかチェックするなどの条件を追加する．

8. おわりに

本研究では，コードの品質改善の支援を目的として，リファクタリング支援ツール MCC を試作した．MCC は C 言語で書かれたソースコードを読み込み，静的解析を行う．静的解析では，変数名が適切に命名されているか辞書を用いて判断する．変数名が適切でない場合は，その変数名のある行番号と変数名を表示し警告する．加えて MCC はソースコードの編集機能を備えている．

MCC に C 言語で書かれたソースコードの例を適用する事によって，MCC が正しく動作することを確認した．MCC は，命名が適切でない変数名を指摘できる．命名が適切でない変数名とは，辞書に存在しない単語で構成されている変数名や，1 文字で存在する変数名である．MCC によって指摘された変数名を修正していくことにより，コードを修正する際の理解の妨げになる要因を減少させられる．これにより，ソースコードを理解するための時間が減少し，コーディング時間の短縮，バグ混入の可能性の減少，機能追加に必要な時間が図れる．今後は 7 章で述べた課題に対応していく．

参考文献

1) Martin Flower et al.: Refactoring Improving the Design of Existing Code, 1nd edn. Pearson Education, 1999.
2) Dustin Boswell and Trevor Foucher: The Art of Readble Code, O'Reilly Media, 2012.
3) デ辞蔵 Web サービス: デ辞蔵オンラインでもオフラインでも使える電子辞書，http://dejizo.jp/dev/index.html
4) JavaCC: JavaCC Home, https://javacc.java.net/
5) Split: Splint Home Page, http://www.splint.org/
6) AdLint: AdLint, http://adlint.sourceforge.net/
7) ANTLR: ANTLR, http://www.antlr.org/