R-5

マジックナンバー修正によるソースコード品質改善の提案

橋浦研究室 120I052 岡田吏都

1．はじめに

ソフトウェア開発者にとってソースコード読むことは，ソフトウェアの保守や開発に必要不可欠な作業である．このような，保守や開発を困難にする原因として，技術的負債がある．技術的負債の一つとして，マジックナンバーの存在が挙げられる．本研究におけるマジックナンバーとはソースコードに直に記述された数値のことであり，その意味や意図が記述した本人以外には自明ではないものを指す．マジックナンバーは同じ数字を数カ所で論理値として参照するとき，理解と検索の変更作業の増加に繋がる．一方で，マジックナンバーは SonarQube[1]では検出されないため，これを効果的な方法で支援する必要があると考えた．

2．研究目的

本研究は「マジックナンバーの修正を一部自動化することで，修正における問題の減少化を目指している」を目指している．

3．提案手法

本研究では，定数を使用者が定義を基に，`if` 文の条件式中に存在するマジックナンバー(表 1)を自動的に定数や列挙型に置き換えることで，ソースコード品質改善の支援を目指す．

表 1. 対象となるマジックナンバー

#	マジックナンバーの例
1	`if(hoge == 10)`
2	`if(hoge <= 10)`
3	`if(hoge >= 10)`
4	`if(hoge < 10)`
5	`if(hoge > 10)`
6	`if(hoge != 10)`

本手法の有効性を確認するために，2 つのリサーチクエスチョンを設けた．

RQ1 本ツールでマジックナンバーを修正するとき間違いに差は生じるか

RQ2 本ツールでマジックナンバーを修正するとき間違い発生に影響するか

本研究では，`if` 文の中のマジックナンバーを自動で修正するために，独自のアルゴリズムを用いることで，列挙型と定数型に振り分ける．振り分けた定数が記入漏れや命名の再現性による支援を行う．図 1 は列挙型と定数型の振り分ける例を示す．変数のグループが 2 種類以上であれば定数型から列挙型に振り分ける．例えば，変数 `i` に属するマジックナンバーが 2 種類，変数 `j` に属するマジックナンバーは 1 種類なので，それぞれ列挙型，定数型に振り分ける．

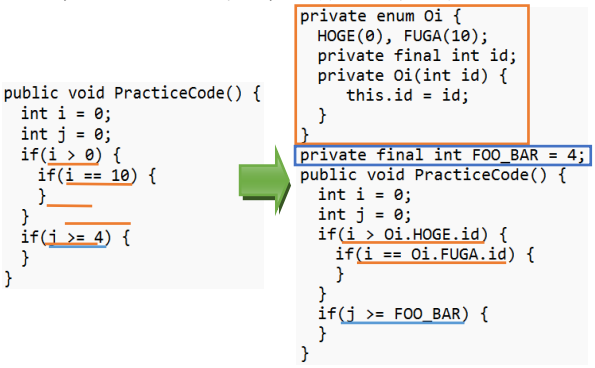

図 1. 列挙型と定数型の振り分けの例

4．ツールの実装

本手法を Eclipse のプラグインとして実装した．以下に実装した機能を示す．

I. `if` 文の中のマジックナンバーの検知
II. 対応するマジックナンバーの任意の定数を定義
III. 対応する定数によるマジックナンバーの置き換え
IV. 列挙型および定数の振り分け

本ツールのマジックナンバー修正画面の例を図 2 に示す．

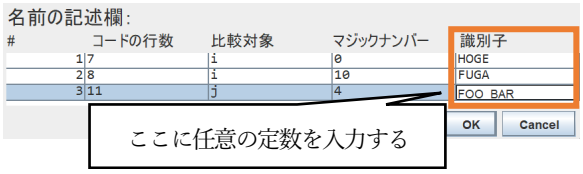

図 2. マジックナンバーの修正画面

ユーザーはマジックナンバーに対応する全ての識別子の項目に名前を記入し，OK ボタンを押すことで，自動的に列挙型と定数型に振り分けられる．

本ツールは全ての `if` 文の中のマジックナンバーを検知することで，マジックナンバーの置き換え漏れを防ぐ．

5．評価

本実験では RQ を確認するため，日本工業大学に所属する Java 経験者 1 名，並びに先進工学部生者 21 名の計 22 名を被験者とし，以下の

手順で実験を行った．被験者はマジックナンバーの修正時に本ツールを使用して修正するグループ（以下，A グループ）とシンボリック定数によるマジックナンバーの置き換え[2]の手順に従って修正するグループ（以下，B グループ）に無作為に分けて実験を行う．

実験の手順は以下のとおりである

1) 実験の概要とリファクタリングの説明
2) マジックナンバーの修正手順の説明
3) 用意したソースコードとソースコードのテストケースと定数を提示
4) 全てのマジックナンバーの修正
5) 実験終了

実験手順 3 においては，あらかじめ用意した定数をランダムに並べ替えて記載したメモ用紙を配布する．被験者はこの用紙を実験終了時まで見ることができる．本研究のソースコード上の誤りは命名の誤り，修正漏れに分類する．

7．実験の結果と考察

図 2 に実験群と統制群の正答率を示す．これらの差を比較した結果，本ツール使用と不使用では有意な差は認められなかった
(対応のない t 検定， $\alpha = 0.05, p = 0.7222$).

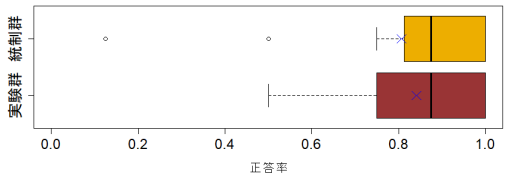

図 3．ツール使用と不使用の正答率の比較

このことから，本ツールはマジックナンバーを修正する場合に，影響が生じないことが確認された．

表 2 にマジックナンバーが含まれている `if` 文の中の条件式を 1 つの単位とし，本ツール使用と不使用による命名間違いと修正漏れの個数を示す．本ツール使用よりも不使用に間違いの発生に影響を与えるかを確認するためにこれらの比較を行った．その結果，有意な差が認められなかった(フィッシャーの正確確率検定，$\alpha = 0.05, p = 0.5294$).

表 2 マジックナンバー修正における間違い個数

#	正答	命名の誤り	修正漏れ	合計
1	74	14	0	88
2	71	15	2	88
3	145	29	2	176

よって，本ツール使用や不使用においてマジックナンバーを修正するとき，間違いの発生に影響は生じないことが確認された．

8．関連研究

Ge[3]らは動的なリファクタリングの検知，リファクタリングの自動化，リファクタリング支援のためのリファクタリング外の支援をするために BeneFactor を作成した．Benefactor は開発者がリファクタリングを行うことを認識する負担を軽減し，開発者が行おうとしているリファクタリングの構成をリファクタリングやリファクタリング以外のことにおいて自動で行うことが出来る．一方で，リファクタリングではない問題に干渉してしまうとリファクタリングの問題を解決できない可能性がある．

これに対し，本研究は半数以上のエンジニアが手動でリファクタリングを行われている現状に対して，自動でリファクタリングを行うことで，リファクタリングは手動よりも自動化が有効であることの確認を行った．

9．まとめと今後の課題

実験結果から本研究では手動化と自動化に差は生じないことが確認できた．有効と結論づけられなかった理由として，提供するソースコードの中のマジックナンバーの数と提供する定数の数が等しかったため，有意差が観測されなかった．

今後の課題として，本研究では自動化によるマジックナンバーの修正アルゴリズムについての評価について焦点を当てていたため，別の観点に対する支援に注目していないことが挙げられる．これに対して，文脈上意味のある名前の自動定義に関するアルゴリズムや本研究の本ツールの有効性について別の視点から評価を行う必要がある．

謝辞

研究を進めるにあたり，貴重な助言をいただいた粂野 文洋教授，荒川 俊也教授，橋浦 弘明准教授に感謝いたします．また，実験に協力してくださった日本工業大学の学生の皆さんに感謝いたします．

参考文献

[1] Sonar Source ,“SonarQube ,”<https://www.sonarsource.com/products/sonarqube/> (Accessed 2023/12/18).

[2] M. Fowler ,“リファクタリング　プログラム体質改善テクニック ,”ピアソン・エデュケーション, May 2000.

[3] X. Ge, Q. L. DuBose and E. Murphy-Hill, "Reconciling manual and automatic refactoring," *2012 34th International Conference on Software Engineering (ICSE)*, Zurich, Switzerland, pp. 211-221, Jun. 2012.