プログラミング初心者の次のステップへの支援システム

田中 雅也[1,a)]　寺田 実[1,b)]

概要：本研究ではコードの良し悪しにかかわる Cognitive Complexity という指標を用いて、コードの品質についてコードを書きながら学ぶことができるシステムを提案する．ここではプログラミング入門を終えた学習者を中級者と定義し、プログラミング中級者が良いコードとは何かを理解することによって、よりよいコードを書くことができるようにすることを目的とする． 問題を提示してコードを書かせ、用意した正解コードと比較して、回答のコードに対してフィードバックを行う。 フィードバックの材料として、Cognitive Complexity を用いた複雑度の計測、凝集度や結合度を用いた計測などを用いる。この情報をもとにユーザへどこのコードがよくなかったのかを視覚的に表示するようにする。 また、上記に加えてよりわかりやすいコードを AST パターンマッチによりルールベースで提案することを検討する。

キーワード：プログラミング, 初心者, コード支援

1. はじめに

近年、プログラミング初心者に向けたプログラミング支援システムが増加している。書籍だけでなく、Web サイトやプログラミング教室など講師を設けて行うなどケースがあり、充実している。しかし、コードの質や良いコードとはどういったコードかという学習手段は書籍や実務経験でしか得ることができないが、勤務を行う上で重要な観点として、コードの質は重要である。

そこで、コード学習を終えた初心者に向けたコード学習支援システムを提供する。ユーザにコードを書いてもらい、そのコードのどこが改善できるかをフィードバックとして返すシステムである。

[1] 電気通信大学 情報理工学研究科
[a)] t2131099@edu.cc.uec.ac.jp
[b)] terada.minoru@uec.ac.jp

2. 関連研究

2.1 よいコードの指標について

良いコードとはどういったコードなのかという指標は数多く存在する。その一つとして凝集度と結合度 [3] がある。関数内の複雑さやクラス間の役割がどのくらい密接になっているかを記す。現在において、この指標はよく用いられることがあり、コードの設計であったり、改善を行う上で重要な指標となっている。また、Code Smells[4] という指標がある。これはコードのあるべきでない状態を示すものである。こうしたコードが存在する場合にはコード改善の余地があるため、リファクタリングを行う際の目安や、リファクタリングを行う際にまず最初に確認する項目として使用される。以上のような指標が存在する中で、本研究では Cognitive Complextiy[1] を使用する。

2.1.1 凝集度・結合度

凝集度と結合度は関数内の複雑さやクラス間の役割がどのくらい密接になっているかを記す。凝集度が低いと関数内の役割が明確でなく、責任が曖昧となるため、コードの読みづらさにつながる。また、結合度が高いと２つの関数間、もしくはクラス間の依存性が高く、意図しないバグが起きてしまう可能性がある。

この指標を用いたオブジェクト指向学習支援システム [6] がある。このシステムを使うことで、オブジェクト指向の設計について学ぶことができるようになっている。

2.1.2 Code Smells

Code Smells はコードのあるべきでない状態とはどういう状態のことを指すのかを示すものであり、コードを管理する上でなんとなく良くないと感じるコードを言語化してまとめている。

この指標の具体的な例は数多くの種類がある。例えば、単純にコードの凝集が増えてしまったメソッドやクラス。こうしたコードは保守が大変になるため、よくないとされる。また、コードメソッド間やクラス間で依存性があり、片方を変更するともう片方に影響が出てしまうコードもよくない。意図しないバグが発生する温床となってしまうためである。こうした、メンテナンスが後にしづらくなってしまうコードは Code Smells として言語化され、できるだけ避けるべきコードとして認知できるようにしている。本研究ではこの Code Smells の中からプログラミング初心者でもわかりやすいものを利用して改善を促せるようにする計画である。

2.2 オンライン学習支援システム

web 上でのプログラミング学習支援システムとして Online Python Tutor[5] がある。このシステムを使うことで、python でのデバッグやデータの遷移が理解しやすくなる。また、web 上で利用ができるため、教師から学生へ向けた情報の共有が簡単であることや、ソフトをダウンロードせずに利用できるため、気軽に使用できるという利点がある。本システムでは、Online Python Tutor のようにユーザが入力したコードを元にフィードバックを返すという対話性を取り入れ、視覚的に情報を伝えられるような工夫を行った。

2.3 手を動かす利点について

プログラミング学習においては、いくつかの方法がある。書籍を読んだり、コードを読むといった方法があるが、その中でも手を実際に動かして学ぶという方法が効果が出やすい。こうしたコード学習方法の一つとして、いわゆる写経による学習がある。この方法を用いることで、視覚だけでなく、手を動かすという動作が必要になり、より学習に与える影響が大きくなる。写経は過去の研究でも効果が出ていることがわかっている [2]。本研究では、この情報をもとに実際にユーザが手を動かしながらコードを書いてもらい、より学習の効果が出るようにする。

3. 提案システム

3.1 対象ユーザについて

本システムは、コードの品質に注目して学習してもらうことを目的としている。条件分岐や繰り返しなど、制御構造に着目して支援ができるようにする。オブジェクトなどの型定義によるコード品質などは本システムでは扱わない。また、対象となるユーザとして、コード学習を終えたばかりのプログラミング初心者をターゲットとする。コードの基本がわかってきたタイミングで本システムを触れてもらうことで、よいコードとはどういうコードなのかを学習し、スムーズにプログラマーとして成長ができるのではないかと考えている。

3.2 コードの品質について

今回定義する良いコードとは、「人間が読みやすいコード」として定義する。人間がコードを直感的に読みやすいのか、また理解がしやすいのかで良し悪しを判断する。この基準を使うために Cognitive Complexity を使用する。この指標はコードの複雑さを計測することができる。前述した凝集度や結合度といった従来の指標と比較して人間の感覚に近い複雑さを計測できるのが特徴である。従来の

コード 1– 既存手法による複雑度評価

```
1    def myMethod:
2        try:
3            if condition1: # +1
4                for i in range(10): # +1
5                    while condition2: # +1
6                        ...
7        except ErrorType1: # +1
8            if condition2: # +1
9                .... # 既存手法5
```

コード 2– Cognitive Complexity による複雑度評価

```
1    def myMethod:
2        try:
3            if condition1: # +1
4                for i in range(10): # +2 (
                     nesting=1)
5                    while condition2: # +3 (
                         nexting=2)
6                        ...
7        except ErrorType1: # +1
8            if condition2: # +2 (nesting=1)
9                .... # Cognitive Complexity 9
```

手法では、主に大規模なソフトウェアに対応したものが多く、プログラミングの基礎を終えた学習者には理解が難しいところもあるため、できるだけ直感的で小規模なプログラムに対してもうまく動作するような指標を選択した。

3.3 Cognitive Complexity

Cognitive Complexity は関数内の制御構造のネストの深さに重みをつけた値の総和である。これまで存在していた指標である凝集度や結合度という指標では、コードを学習し終えたばかりのプログラミング初心者には理解が難しい。関数がどう動くべきかの役割付けに関しての知識がまだないためである。しかし、Cognitive Complexity は制御構造に着目し、ネストの深さを中心に判定することでより直感的にコードの複雑さを判定をできるようしている。そのため、関数自身がこうあるべきであるという抽象的な理解はまだ必要なく、純粋にコードはこう書くべきであるという視点からアドバイスができるようになるため、本システムに最も有用である。

3.3.1 評価方法について

Cognitive Complexity は主に制御構造に着目して評価を行う。

同一コードに対する二種類の複雑度の評価をコード 1, 2 に示す。例として使用しているコードはネストが深く、読みづらいコードである。そのため、算出される指標は値が高いものであるとよい。既存の方法では、単純に条件分岐や繰り返しが入ったタイミングで値を +1 する手法がある。この方法で評価を行うとコード 1 のように 5 という値となる。これに対し、Cognitive Complexity では、制御構造に着目し、if, while, try などのネストが発生する場合に値が +1 される。さらに、制御構造が入れ子になる場合には、入れ子の深さ分 Cognitive Complexity の値を増加する。そのため、4 行目では +2, 5 行目では +3 となる。このように、Cognitive Complexity はネストの深さに着目して値を増加させるため、単純に if, while などの制御構造を計測するだけでは得られなかった複雑度をより人間の感覚に近い形で得ることができるようになっている。

3.4 システムについて

3.4.1 概要

本システムは web 上で動作する。ユーザは本システムにアクセスし、ブラウザ上でコードを書いて評価してもらう。システムからアルゴリズム問題などが出題され、ユーザが問題を解決するようなコードを作成する。作成したコードに対して、Cognitive Complexity を算出し、フィードバックを行う。web システムは Python と JavaScript を使って実装されている。Python でバックエンドを作成し、JavaScript でフロントエンドを作成する。

3.4.2 バックエンド

バックエンドは python で記述されており、Cognitive Complexity はバックエンドで測定する。そのため、ユーザが作成したコードは JavaScript によって、バックエンドへと送る必要がある。Cognitive Complexity の計測も、python を使用し、python で書かれているコードの測定を行う。計測

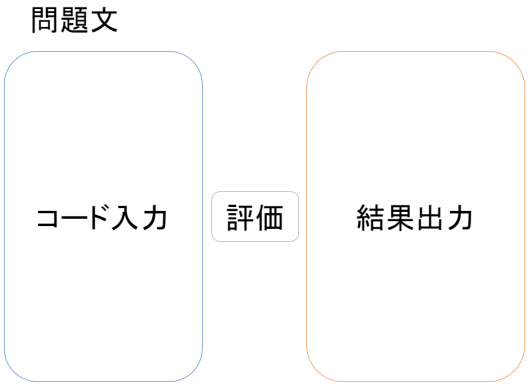

図 1　画面構成

後、正解となるコードと比較を行い、理想のコードとどこが乖離してしまっているかを確認する。ここで得られたデータをフロントエンドへと渡す。

3.4.3　フロントエンド

フロントエンドでは、得られた Cognitive Complexity を出力したり、得られた値を利用してどこが改善できるかのフィードバックを行う。バックエンドで調べたことで得られた、煩雑になってしまったコードを色付けで表示することで、視覚的に理解しやすいように表示する。

図 1 のように、画面左上部に問題を表示させ、ユーザは画面右の入力欄にコードを入力してもらう。真ん中の「評価」ボタンを押すことにより、コードの評価が行われ、画面右の出力欄にフィードバックを表示させる。ここにはコードの複雑さがどのくらいであったかや、コードをどう改善できるのか、どこがより複雑になってしまっているかなどの情報を表示させる。

3.5　ユーザが解く課題について

課題は 2 種類の問題を想定している。

1 つ目に簡単なアルゴリズム問題である。アルゴリズム問題を解いてもらい、書いたコードに対してフィードバックを行う。アルゴリズム問題のため、そのアルゴリズムが正しく動作するのかを確認できる出力画面も用意し、デバッグができるようにする。具体例として、「フィボナッチ数列を 10 個までもとめなさい」等の問題である。

2 つ目にリファクタリング問題である。出題されたコードをユーザが改善し、どのくらい改善されたか、もしくはもっと改善できる個所はあるかなどのフィードバックを行う。アルゴリズム問題を解いてもらうよりリファクタリングの問題を解いてもらうほうがより実践的な課題になる。

3.6　フィードバックについて

3.6.1　Cognitive Complexity の表示

ユーザに書いてもらった python コードの Cognitive Complexity を用いた出力を行う。この指標単体の表示ではユーザが Cognitive Complexity がどういった値なのかを理解していないため、有効的ではないと考えた。そのためどのくらい複雑になっているかを日本語で補助することで、現在のコードがどのくらい良いコードなのかを理解しやすくする。表示を行う際に、理想の回答のコードでの Cognitive Complexity の値と比較することで、ユーザが現時点で書いているコードがどのくらい複雑になってしまっているかを指摘するなどを想定している。

3.6.2　Cognitive Complexity を利用した複雑なコードのハイライト

ユーザにコードのどの個所が複雑になってしまっているかを示す。どこが複雑になっているかの判断として Cognitive Complexity を使用する。

コードのハイライト方法は図 2 のように行う。出力画面に色付けを行い、もっと改善できるコード個所を視覚的に見えるようにする。ユーザはこのハイライトを見てどこがよくないのかを理解し、修正することができるようになる。

3.6.3　より簡潔に改善できるコードの提案

ユーザが回答したコード内で python の文法を使ってより簡潔に書けるコードを提案する。例えばリスト内包表記や、余計な制御構文の削除などである。これの AST のパターンマッチで置き換えることで、具体的なコード改善の提案を行う。このフィードバックを与えることで、ユーザがより簡潔に python のコードを書くことができるようになり、可読性の学習につながる。

3.6.4　コード全体の評価

Cognitive Complexity を使って、全体の複雑度を

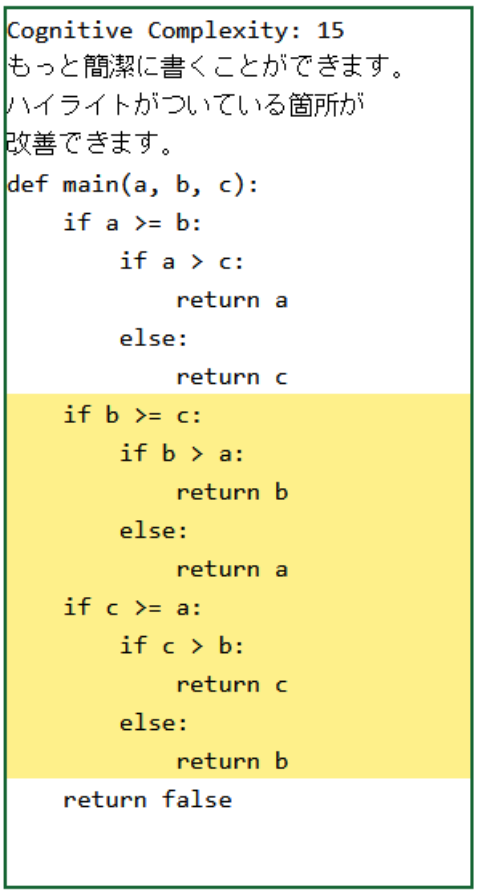

図 2 コードハイライト

確認する。問題を作成するときに、理想の回答を用意する。その回答コードの Cognitive Complexity を使い、どのくらい差があるかを計測する。

4. 評価方法

本システムが有益かどうかの評価方法は以下である。これらは本稿執筆時点での計画である。

- コードの質の理解度
- 最終的な正答率
- システムの使いやすさ

4.1 コードの質の理解度

このシステムを通して、コードの質が良い状態とはどういった状態なのかを学習できたかどうか理解する。評価として、「よくわからなかった」から「よく理解できた」までの５段階評価を行う。

4.2 最終的な正答率

本システムはユーザに問題を解いてもらう形式である。そのため、ユーザにとって適切な問題であったか調査を行う必要がある。また、正答率とコードの質の理解度には相関があるのかを調査し、コードをしっかり書くことができる人は、コード理解が進みやすいのか等の検証も口頭などの確認で行う予定である。

4.3 システムの使いやすさ

本システムはユーザに使ってもらうことを想定している。そのため、システムが使いやすいかどうかはこのシステムが使ってもらえるかどうかにかかわる重要な評価軸である。

5. 今後について

現状の課題として、問題を解く際に複数の解き方があるものに関しての対処が難しいことがある。現在は問題に対して理想となる正解コードを用意する必要がある。その正解コードを利用して、適切な Cognitive Complexity の値を導出するようにしている。

また、Cognitive Complexity だけでなくほかの指標を使ったフィードバックを返せるようにしていく。例えば Code Smells で初心者のユーザにもわかりやすい箇所を指摘できるようになれば、Cognitive Complexity だけでなくもっと多くのフィードバックを与えることができるようになる。

6. まとめ

本研究では、プログラミング初心者に対してよいコードとはどういったコードなのかを学習できるシステムを作成した。このシステムを用いることで、よりプログラミング初心者のコードの理解であったり、エンジニアとして活動するときに必要な知識習得の手助けになればと思う。今後実際にユーザにシステムを使ってもらい、実際にシステムが有用に働くのかどうかの調査を行う。

参考文献

[1] SonarSource. COGNITIVE COMPLEXITY. https://www.sonarsource.com/docs/CognitiveComplexity.pdf, 2016.

[2] 岡本雅子, 喜多一. プログラミングの「写経型学習」における初学者のつまずきの類型化とその考察. 滋賀大学教育学部附属教育実践総合センター紀要，2014.

[3] G．J．Myers（國友ほか訳）. ソフトウェアの複合/構造化設計. 近代科学社，1979

[4] M. Fowler. Refactoring: Improving the Design of Existing Code. Addison Wesley, 1999.

[5] Guo, Philip J. Online python tutor: embeddable web-based program visualization for cs education. Proceeding of the 44th ACM technical symposium on Computer science education, ACM, 2013.

[6] 杉浦啓孝, 松澤芳昭, 酒井三四郎. 凝集度と結合度に注目する OOD 学習支援システム Fourcs の提案. 情報処理学会 第 73 回全国大会講演論文集, 2011.