プログラミング演習のデバッグに対する過去のエラー情報に基づいた助言の可能性に関する分析

高橋圭一 [1]

概要：プログラミング演習中に講師が巡回していると，過去に似たようなエラーに対応したことを思い出すことがある．学生が起こすエラーの類似性について 2019 年と 2020 年に開講した同一のプログラミング科目で調べたところエラーの分布はほぼ等しいことが確かめられた．そこで，我々は学生が起こしたエラーに講師がデバッグ方法を付加したデータベースを用意することで，類似するエラーを発生した学生にデバッグ方法を助言するシステムが構築できると考えた．本稿では，実際にシステム開発を進める前に予備調査として，近畿大学産業理工学部情報学科の 3 年生を対象とした Web プログラミング科目において，2021 年に収集したエラー情報をもとに 2022 年の同科目のエラー情報に対して助言が可能かどうかを調べた．結果として，2022 年に発生した 248 件の誤り情報のうち 76 件（31%）に対して助言が可能であることがわかった．

キーワード：プログラミング教育，デバッグ支援，エラーメッセージ

Analyze the possibility of advice based on historical error information for debugging during programming exercises

KEIICHI TAKAHASHI[†1]

Abstract: When making the rounds during programming exercises, the instructor may recall handling similar errors. We compared errors made by students in the same programming courses in 2019 and 2020 and found that errors were almost identical. Therefore, if we prepared an error database that added debugging methods to the errors made by students, we could advise students who encounter similar errors in the future on how to remove bugs. In this paper, as a preliminary experiment, we examined the possibility of advising students in a Web programming course for third-year students at Kindai University in 2022 based on the error database collected in 2021. As a result, we found that advice was possible for 76 (31%) of the 248 errors that occurred in the 2022 class.

Keywords: Programming Education, Debugging Support, Error Messages

1. はじめに

プログラミング演習中に講師が巡回していると，過去に似たようなエラーに対応したことを思い出すことがある．教授内容が同じであればプログラミング演習の基本的な手順は同じであるため，翌年の同演習で類似したエラーが発生する可能性が考えられる．2019 年と 2020 年に開講した同一のプログラミング科目でエラーの発生状況を比較したところ，エラーメッセージの発生割合は両年でほぼ同じ傾向であった[1]．このことから発生割合が多いエラーメッセージ，つまり多くの学生が発生したエラーは未来においても発生する可能性が高いことを示唆している．

我々はこの考えをもとに，過去に学生が発生したエラー情報を収集し，そのエラー情報にデバッグ方法を関連づけたデータベースを構築し，未来において学生が発生したエラーに最も近いエラー情報のデバッグ方法を提示することにより，学生自身によるエラーの自己解決を促す支援システムの開発を目指している．図 1 にシステムの概要を示す．本システムは，(a)エラー情報収集ツールと(b)デバッグ方法提示システムから構成されている．以下，それぞれについて説明する．

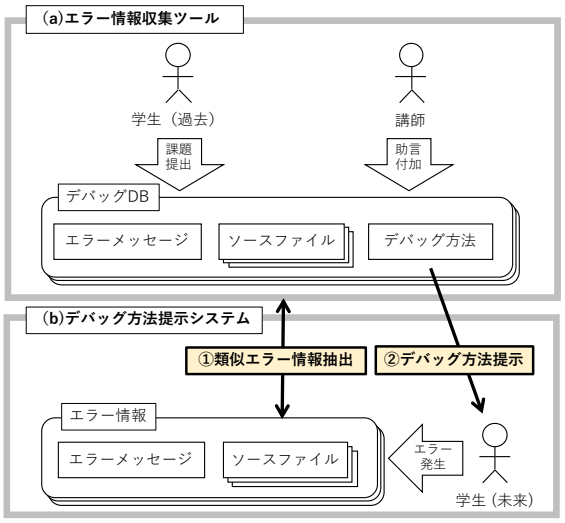

図 1　エラー自己解決支援システム

エラー情報収集ツール（図 1(a)）：プログラミング演習中に学生がエラーを発生するとその時点のソースファイルを自

1 近畿大学産業理工学部情報学科
Kindai University

動的に保存するツールである[1]．ソースファイルの保存にはバージョン管理ソフトウェアの 1 つである Git を利用する．エラーメッセージは Web アプリケーションのログファイルに記録されるため，そこから抽出する．このログファイル中のエラーメッセージと Git リポジトリに保存されたソースファイルをエラー発生時刻で関連づけ，さらに学生がつまずいたときに与える講師の助言をデバッグ方法として付加する．

このようにエラー情報とデバッグ方法を関連付けて収集したデータベースを本稿ではデバッグ DB と呼ぶ．本ツールは既に完成しており，2019 年からプログラミング演習科目で使用しエラー情報を収集している．

デバッグ方法提示システム（図 1(b)）：エラー発生時のソースファイルとデバッグ DB に登録された過去の学生のソースファイルを比較し，類似度が最も高いエラー情報を抽出する．これを類似エラー情報抽出機能（図 1①）と呼ぶ．この機能の実行結果で得られたエラー情報に関連づけられたデバッグ方法を学生に提示するのがデバッグ方法提示機能（図 1②）である．本システムはこれから開発する予定であるが，類似エラー情報を抽出するアルゴリズムについては既に提案している[2]．

本稿では，エラー自己解決支援システム（図 1）の開発に着手する前の予備調査として，既に収集しているエラー情報をもとに，過去のエラー情報から未来に発生したエラーに対してデバッグの助言が可能か調査する．以下，2 章に調査方法を述べ，3 章に調査結果を示す．4 章にて調査結果について考察し，最後にまとめる．

2. 調査方法

2.1 調査対象

近畿大学産業理工学部情報学科 3 年生対象の Ruby による応用プログラミングを学習する科目の第 12 回を調査対象とする．本科目は講義と演習が連続した 2 コマ（180 分）の授業である．学生は 1 年次より Ruby の基礎を学習しており，さらに本科目の第 1〜8 回までに様々な応用プログラミングを学び，第 9〜11 回に Web アプリケーションフレームワークの 1 つである Ruby on Rails（以降，Rails）プログラミングの基礎を学ぶ．第 12 回では Rails アプリの応用例として，画像ファイルをアップロードする Web アプリケーションを開発する．

2021 年および 2022 年の本科目の提出物に含まれたエラー情報およびソースファイルを調査対象とする．課題を提出した人数はそれぞれ，2021 年は 67 名，2022 年は 50 名であった．

2.2 プログラミング演習中の操作手順

調査対象である第 12 回の講義資料の例を図 2 に示す．本講義で新たに学ぶ画像アップロード機能を実現するプログラムについては講義資料に詳細が記載されているが，その他は第 9〜11 回の資料に書かれているため，講義資料としては目隠しをした状態で配布し，講義を聞きながら各自で空欄を埋めさせる．Web アプリケーションを完成させるまでの操作手順は講義資料の目隠しの白枠を 1 ステップとすると 24 ステップである．

Rails は MVC（Model-View-Controller）アーキテクチャを採用しているため，Web アプリケーションを構築するためには，MVC の各ファイルにプログラムを記述する必要がある．そのため，学生は URL と呼び出すプログラムを紐つけるルーティングファイル，Ruby で書かれたコントローラファイル，HTML と Ruby を混在して記述するビューファイルなど合計で 6 つのファイルにプログラムを記述する必要がある．そのため，単一ファイルのプログラムと比べてプログラムの入力誤りが発生しやすい傾向がある．

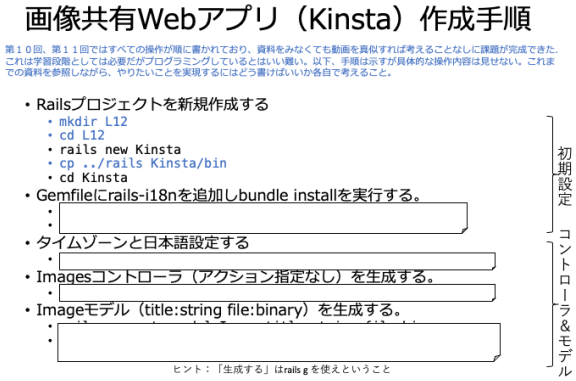

図 2 第 12 回の講義資料の例

2.3 類似エラー情報抽出アルゴリズム

図 1 に示した類似エラー情報抽出機能のアルゴリズムについて述べる[2]．本アルゴリズムは提案済みであるが，本稿の議論に関わるため本節にて説明する．

学生の提出物に含まれるソースファイル群をエラー発生ごとに作成したディレクトリに格納する．2021 年と 2022 年のディレクトリ同士を diff コマンドで比較し類似度を算出する．diff コマンドは片方のディレクトリのみに存在するファイル数，および，同じファイル名の内容の差分行数を出力する．同じファイル名の内容の差分情報として，diff コマンドは追加(added)・削除(deleted)・変化(changed)を区別して出力するが，本アルゴリズムではすべて差分行数としてカウントする．なお，プログラム以外を比較対象から除外するため，空白，タブ，コメント，空行を除去した後，diff コマンドを実行する．

ファイル比較する 2 つのディレクトリのうち片方にのみ存在するファイル数を D_{diff} とし，同一ファイル名の差分行数を F_{diff} とする．これら 2 つの指標を用いて以下のルールを順次適用し，類似度が最も高いエラー情報を決定する．

(1) D_{diff} が最小であるエラー情報が 1 つである
(2) D_{diff} が最小，かつ，F_{diff} が最小であるエラー情報が 1 つである
(3) 以上で 1 つに絞れない場合は，それまでの候補の最

古のエラー情報を最も類似するエラー情報とする

本アルゴリズムの実行例を図 3 を用いて説明する．2022 年に C1～C3，2021 年に P1～P3 とそれぞれ 3 つずつエラー情報があるものとする．ここでいうエラー情報とはディレクトリであり，ディレクトリにはエラー発生時のソースファイルが 1 つ以上格納されている．図 3 は 2022 年のエラー情報 C1 に最も類似する過去のエラー情報を決定する例である．まず，C1 と P1，P2，P3 について D_{diff} と F_{diff} を求める．その後，ルール(1)より D_{diff} を調べると最小は P1 と P2 でエラー情報は 1 つに絞れない．そこでルール(2) より P1 と P2 の F_{diff} を比較すると，P2 が最小であるため C1 に最も類似するエラー情報は P2 と決定できる．もし，両者の F_{diff} が同値であれば，ルール(3)により，最古のエラー情報である P1 が最も類似するエラー情報となる．

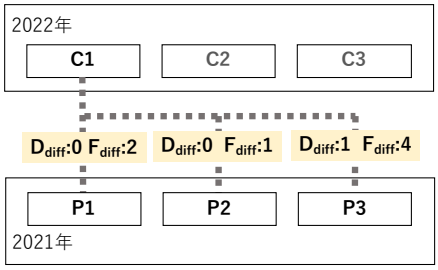

図 3　類似エラー情報抽出アルゴリズムの適用例

2.4 助言の可能性の判定方法

2021 年のエラー情報をもとにデバッグ DB を準備する．前節のアルゴリズムで 2022 年に発生したエラー情報に最も類似したエラー情報を決定し，そのエラー情報に関連づけられたデバッグ情報を得る．このデバッグ方法が 2022 年に発生したエラーのデバッグ方法の助言として適切か筆者が判断する．

表 2 をもとに助言の可能性の検証例を示す．表 2 では index.html.erb というファイルで学生が「助言対象コード」を入力し，Web アプリケーションを実行したところ ActionView::Template::Error というエラーに遭遇した状況を表している．このエラー情報に対して，前節の類似エラー情報抽出アルゴリズムを適用した結果，「助言内容」に示されたコードおよび助言が得られたとする．「助言対象コード」は「助言内容」のコードと同様に，Ruby のブロック引数とブロック内で使用している変数名が異なることで発生したエラーであり，「助言内容」に示された助言を読むことで学生自身で誤りに気づくことが期待できるため助言可能と判定する．

表 2　助言の可能性の検証例（助言可能と判定）

ファイル名	index.html.erb
エラーメッセージ	ActionView::Template::Error
助言対象コード	1:<% @Images.each do |image| %> 2:　　<%= image.title %> 3:　　<%= image_tag "/get_image/#{image.id}" %> 4:　　<%= link_to '削除', "/bookmarks/#{book.id}", method: :delete %> 5:　　<%= link_to '編集', "/bookmarks/#{book.id}/edit" %> 6:　　 7:<% end %> 8: <%= link_to '新規追加', '/images/new' %>
助言内容	**ブロック引数bookはブロック内の変数(image)と同じである必要があります．以下の例では|book|を|image|に変更するとよいです．** 1:<% @image.each do |book| %> 2:<%= link_to image.title %> 3:<%= image_tag "/get_image/#{image.id}" %> 4:<% end %> 5: <%= link_to '新規追加', '/images/new' %>

3. 調査結果

3.1 ファイル別のエラーの発生割合

エラー情報をエラーが発生したファイル名別に集計する．「エラーファイル名とエラーメッセージ」の組み合わせを 1 種とすると，2022 年には 22 種（248 件）のエラーが発生し，2021 年には 18 種（236 件）のエラーが発生した．

このうちエラー発生件数の上位 7 種を表 1 に示す．エラー件数はそれぞれ，2022 年は 222 件，2021 年は 192 件であった．全エラー件数に対する各エラーの発生割合を知るため表 1 には発生件数ではなく相対度数を示している．2022 年および 2021 年ではともに，ビューファイルの 1 つである index.html.erb で発生した ActionView::TemplateError が最多であったことがわかる．さらに累積相対度数より上位 7 種のエラーが全エラーのほぼ 9 割を占めることがわかる．2021 年も同様の傾向である．以下，この上位 7 種のエラーについて分析を進める．

3.2 エラー別の助言の可能性

表 1 の上位 7 種のエラーをもとに，エラーファイル名・エラーメッセージが同じエラー情報のソースファイル群を 2.3 節のアルゴリズムで比較し，類似度が最も高いエラー情報を決定する．そのエラー情報に対して 2.4 節の方法で助言の可能性を判定した結果を表 3 に示す．

表 1　ファイル別のエラーの発生割合（上位 7 種）

			2022年		2021年	
No.	エラーファイル名	エラーメッセージ	相対度数	累積相対度数	相対度数	累積相対度数
1	index.html.erb	ActionView::Template::Error	33.6%	33.6%	42.7%	42.7%
2	index.html.erb	SyntaxError	17.0%	50.6%	6.8%	49.5%
3	images_controller.rb	NameError	10.5%	61.1%	9.1%	58.6%
4	images_controller.rb	ActionController::RoutingError	9.9%	71.0%	2.9%	61.5%
5	images_controller.rb	NoMethodError	9.0%	79.9%	9.7%	71.2%
6	debug_exceptions.rb	ActionController::RoutingError	5.9%	85.8%	7.4%	78.6%
7	images_controller.rb	SyntaxError	3.7%	89.5%	2.6%	81.2%

表 3　エラーファイル名・エラーメッセージ別の助言可能件数（2022 年）

No.	エラーファイル名	エラーメッセージ	助言可能件数
1	index.html.erb	ActionView::Template::Error	49　(88)
2	index.html.erb	SyntaxError	13　(33)
3	images_controller.rb	NameError	9　(24)
4	images_controller.rb	ActionController::RoutingError	1　(32)
5	images_controller.rb	NoMethodError	2　(21)
6	debug_exceptions.rb	ActionController::RoutingError	0　(19)
7	images_controller.rb	SyntaxError	2　(5)
		合計	76　(222)

表 3 の括弧内の数値は当該エラーの発生件数である．上位 7 種のエラーは全部で 222 件発生し，そのうち 76 件(34%)が助言可能であった．7 種のエラーのうち index.html.erb で発生した ActionView::TemplateError が最多で 88 件発生し，そのうち 49 件（56%）が助言可能であった．続いて多いのは index.html.erb で発生した SyntaxError で 33 件発生し，そのうち 13 件（39%）が助言可能であった．

この表より，ビューファイルである index.html.erb が他のファイルと比べてエラーが発生しやすく助言が成功する傾向が見られる．ビューファイルでは HTML を生成するためプログラム行数が増加傾向にあり，そのため入力誤りが発生しやすくなる．一方，プログラム行数が増えることで特徴的な情報量が増え，過去の類似するエラー情報と適合しやすくなることが考えられる．

3.3　エラー別の誤り要因と助言の可能性

前節でエラーファイル名とエラーメッセージ別に助言の可能性について検証したところ，偏りがあることがわかった．本節ではこの助言可能性に影響する因子を調べる．

各エラーが発生した誤り要因を類別したところ，学生による誤り要因は 5 つに分類できることがわかった（表 4）．ここで挙げた誤り要因によって助言の可能性は変化するだろうか．エラー別の誤り要因を示す（図 4）．

表 4　エラー別の誤り要因

No.	誤り要因	誤り例
1	入力忘れ	ブロックのendや閉じタグの記入忘れ
2	入力多すぎ	不必要な文字や記号を入力している
3	入力間違い	ミススペル（入力場所は正しい）
4	場所間違い	入力場所が違う（コードは正しい）
5	実行方法間違い	呼び出しURLが間違い（コードは正しい）

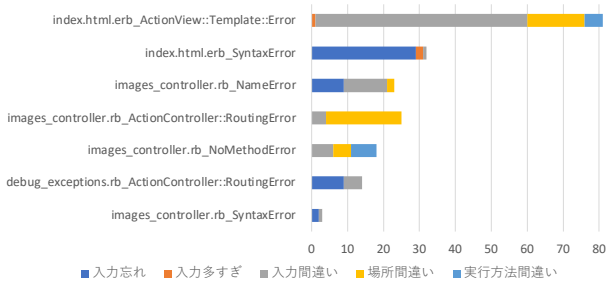

図 4　エラー別の誤り要因（2022 年）

index.html.erb で発生した ActionView::TemplateError の誤り要因の多くは「入力間違い」であった．「入力間違い」が多ければ助言の可能性が高まると考えるが，2 番目に多い index.html.erb で発生した SyntaxError では「入力忘れ」が多くを占めており，一貫性が見られない．

これはエラー発生の仕組みを考えると当然の結果とも考えられる．つまり，Ruby などのスクリプト言語ではプログラム実行時に処理系がコンパイルし実行する．エラーが最多であった ActionView::Template::Error では，テンプレートエンジンが HTML を生成する過程で誤りを発見するとエラーが発生する．誤り要因が同じであっても誤りを含むファイルが異なれば処理する主体が変わるため，異なるエラーが発生する．

そこで，誤り要因別に助言の可否を調査した（表 5）．「入力忘れ」と「入力間違い」は発生件数が多く，助言可能件数が多いことがわかった．「入力多すぎ」「場所間違い」はそもそも発生件数が少ないことから，講義資料の見間違いなど偶発的に発生するものと考えられる．そのため，再現性が低く過去のエラー情報による助言が不可だったことが考えられる．一方，「入力忘れ」と「入力間違い」については，まさに本研究の動機となった「過去に似たようなエラーに対応したことを思い出す」の事例であると考えられる．頻度よく発生する誤りであるため，その誤り要因で発生したエラー情報がデバッグ DB に豊富にあるため，助言の可能性が高まったと考える．

表 5　誤り要因ごとの助言の可否（2022 年）

	助言	
誤り要因	可能	不可
入力忘れ	23	10
入力多すぎ	0	3
入力間違い	50	23
場所間違い	1	6
実行方法間違い	0	2

4.　考察

4.1　過去のエラー情報に基づいた助言の可能性

本稿の調査では 3.2 節で述べたように，上位 7 種のエラーにおいて 222 件中 76 件（34%）が助言可能であった．講師がデバッグ方法を付加する手間はあるものの，過去のエラー情報をそのまま利用して類似度を計算する単純な方式にも関わらず，2022 年のプログラミング演習で発生したエラーの 3 割に助言が可能であったことは良好な結果と考える．一方，2022 年のエラー情報は全部で 248 件発生したため，実質的に助言可能であったのは 31%になる．講師がデバッグ方法を付加してもその 7 割が学生によるエラーの自己解決に利用されないのは効率が悪い．何らかの工夫によって助言の可能性は高められるのだろうか．そこで，今回の調査結果をもとにエラー別に助言可能／失敗であった代表例を振り返り，助言の可能性を高める対処方法について

検討する．

4.1.1 index.html.erb の ActionView::TemplateError に対する助言

表 3 より，index.html.erb で発生した ActionView::TemplateError では 49 件が助言可能であった．このうち 44 件が表 6 に示すエラーであった．着色した部分が両者の差異であるが，異なる間違い方をしていることがわかる．それでも，同じ部分と異なる部分の位置がほぼ同じであるため，差分情報が小さくなり適合しやすかったと考える．

表 6 index.html.erb・ActionView::TemplateError の助言成功例

ファイル名	index.html.erb
エラーメッセージ	ActionView::Template::Error
助言対象コード（2022年）	<% @images.each do |image| %> <%= image.title %> <%= image_tag "/get_image/#{e.id}" %> <%= link_to '削除', "/images/#{image.id}",method: :delete %> <%= link_to '編集', "/images/#{image.id}/edit" %> <% end %> <%= link_to '新規追加', '/images/new' %>
最類似エラー情報（2021年）	<% @image.each do |book| %> <%= link_to image.title %> <%= image_tag "/get_image/#{image.id}" %> <%= link_to '削除', "/images/#{book.id}", method: :delete %> <%= link_to '表示', "/images/#{book.id}" %> <%= link_to '編集', "/images/#{book.id}/edit" %> <% end %> <%= link_to '新規追加', '/images/new' %>

同じ対象で助言に失敗した例を表 7 に示す．表 6 のプログラムと比べると，こちらの方が両者のプログラムが似ているように見える．差分行数はわずか 1〜2 行である．しかし，実は助言として示された 2021 年の index.html.erb にはプログラム誤りはない．データフローで上流にあたるコントローラファイルのプログラムに誤りがあるため，index.html.erb のローカル変数@images が nil となりエラーが発生した事例である．そのため，このプログラムに対する助言は「助言対象コード」とは異なるため助言は失敗と判定する．複数ファイルの差分行数の総計が類似度として算出されるため，エラーが発生したソースファイルの類似度が高くても助言に失敗する可能性がある．

表 7 index.html.erb・ActionView::TemplateError の助言失敗例

ファイル名	index.html.erb
エラーメッセージ	ActionView::Template::Error
助言対象コード（2022年）	<% @images.each do |image| %> <%= image.title %> <%= link_to '削除', "/bookmarks/#{book.id}", method: :delete %> <% end %> <%= link_to '新規追加', '/images/new' %>
最類似エラー情報（2021年）	<% @images.each do |image| %> <%= image.title %> <% end %> <%= link_to '新規追加', '/images/new' %>

4.1.2 index.html.erb の SyntaxError に対する助言

index.html.erb で発生した SyntaxError であり，助言可能だった 13 件の代表例を表 8 に示す．両者に共通する誤りは「<%= link_to」に対応する閉じタグ「%>」の入力忘れである．入力を忘れた閉じタグはわずか 2 文字である．しかし，その他のプログラムが類似していて，かつ，閉じタグが欠落している特徴があるため，最も類似しているエラー情報として抽出に成功したと考えられる．

表 8 index.html.erb・SyntaxError の助言成功例

ファイル名	index.html.erb
エラーメッセージ	SyntaxError
助言対象コード（2022年）	<% @images.each do |image| %> <%= image.title %> <%= link_to '削除', "/images/#{image.id}", method:
最類似エラー情報（2021年）	<% @images.each do |image| %> <%= image.title %> <%= link_to '削除', "/images/#{image.id}", <% end %> <%= link_to '新規追加', '/images/new' %>

一方，同様に閉じタグ「%>」の入力忘れ（赤矢印）によって発生したエラーであっても助言に失敗することがある（表 9 **エラー! ブックマークが自己参照を行っています。**）．この事例の助言のプログラムは閉じタグのミスではなく，引用符の記号誤りによるエラーであるため助言は失敗と判定する．同じ誤り要因であったとしても，その周辺のプログラムの類似具合によって必ずしも適切な過去のエラー情報が抽出できない場合があることを示している．

表 9 index.html.erb・SyntaxError の助言失敗例

ファイル名	index.html.erb
エラーメッセージ	SyntaxError
助言対象コード（2022年）	<% @images.each do |image| %> <%= link_to image.title %> <%= link_to '削除', "/images/#{image.id}", method: :delete %> <%= link_to '編集', "/images/#{image.id}/edit" <% end %> <%= link_to '新規追加', '/images/new' %>
最類似エラー情報（2021年）	<% @images.each do |image| %> <%= link_to image.title %> <%= image_tag "/get_image/#{image.id}" %> <%= link_to '削除', "/images/#{image.id}", method: :delete %> <%= link_to '表示', "/images/#{image.id}" %> <%= link_to '編集', "/images/#{image.id}/edit" %> <% end %> <%= link_to '新規追加', '/images/new' %>

4.1.3 images_controller.rb の NameError に対する助言

この事例は前の 2 例とは異なり Ruby で書かれたコントローラファイルで発生した NameError の例である．こちらは助言可能件数が 9 件と 3 番目に多い事例である．助言に成功した表 10 の助言対象コードでは，ローカル変数 file に初期値を代入する前に @image.update メソッドで file を使用したため発生したエラーである．類似エラー情報でも，Image.new で変数 title を使用しており，メソッドの引数を使用する前に値をセットしたかを確かめる助言を学生に提示することでデバッグできる可能性があるため，助言に成功と判定した．

表 10　images_controller.rb・NameError の助言成功例

ファイル名	images_controller.rb
エラーメッセージ	NameError

助言対象コード
（2022年）
	（省略）
	def update
	  @image = Image.find(params[:id])
	  title = params[:image][:title]
	  @image.update(title: title, file: file)
	  file = params[:image][:file].read
	  redirect_to '/'
	end
	（省略）

最類似エラー情報
（2021年）
	（省略）
	def create
	  file = params[:image][:file].read
	  @image = Image.new(title: title , file: file)
	  image.save
	  redirect_to '/'
	end
	（省略）

一方，表 11 の例では，助言対象コードは変数@file にデータをセットしているものの，@image.update メソッドの引数に@を除いた file という変数を指定している誤りである．このエラーに対する助言は，変数 title に値をセットせずに使用している誤りである．変数名を@file と間違っていたとしても，学生としては「@file = params[:image][:file].read」という代入文を記述しているため，与えられる助言から誤りに気づくのは困難と考えられる．

表 11　images_controller.rb・NameError の助言失敗例

ファイル名	images_controller.rb
エラーメッセージ	NameError

助言対象コード
（2022年）
	（省略）
	def update
	  @image = Image.find(params[:id])
	  title = params[:image][:title]
	  @file = params[:image][:file].read
	  @image.update(title: title , file: file)
	  redirect_to '/'
	end
	（省略）

最類似エラー情報
（2021年）
	（省略）
	def create
	  file = params[:image][:file].read
	  @image = Image.new(title: title , file: file)
	  image.save
	  redirect_to '/'
	end
	（省略）

4.2 助言の可能性を高める対処案

前節では，助言が可能であった代表的な事例を示しながら，助言が可能であった要因を検証した．このことを踏まえて，本手法において，助言が可能となる要件を整理すると，以下の 2 つが考えられる．

- デバッグ DB に類似するエラー情報が存在する
- デバッグ DB から誤り要因が同じエラー情報を発見抽出できる

本手法では，あらかじめ助言を用意しておく必要があるため，プログラミング演習で初めて発生した種類のエラーは助言できない．一方，前節に示したように，デバッグ DB に類似するエラー情報があったとしても，複数ファイルの差分行数を積算して類似度を決定するため，必ずしも同じ症状のエラー情報を抽出できるとは限らない．誤り要因は様々であり，プログラム入力を間違える場所も様々であるため，類似するエラー情報を発見できる可能性を定量的に見積もることは困難である．

コードクローン研究では，様々な類似プログラムの探索方法が提案されている[3]．しかし，研究対象の多くがが文法的に正しいプログラムを対象としている．本研究で対象としているプログラムには誤りが含まれているため，正しいプログラムと比べると記述の自由度は高くなり，類似したプログラムを発見することは困難になる．

本稿にて実際に集めたデータを利用して調査を進めた結果，本手法の問題点が 1 つ見えてきた．2.3 節に示した現状のアルゴリズムでは，類似するエラー情報を抽出するためには，学生がプログラミング演習で入力した全ファイルの差分情報を求める．4.1 節で示したように，エラーが発生したファイル以外の類似度が高い影響で誤ったエラー情報が抽出されてしまう可能性があった．類似度を算出するための差分情報を求めるファイルを，エラーが発生したファイルおよびデータフローで上流にあるファイルに限定することで，適切なエラー情報に適合できる可能性が高まり，より適切な助言が可能となることが考えられる．

5. まとめ

本稿では，近畿大学産業理工学部情報学科の 3 年生を対象とした Web プログラミング科目において，2021 年に収集したエラー情報をもとに 2022 年の同科目のエラー情報に対して助言が可能か調査した．結果として，2022 年に発生した 248 件の誤り情報のうち 76 件（31%）に対して助言が可能であることがわかった．

本手法は，講師がデバッグ方法を付加する手間はあるものの，過去のエラー情報をそのまま利用して類似度を計算する単純な方式にも関わらず，2022 年のプログラミング演習で発生したエラーの 3 割に助言が可能であったことは良好な結果と考える．一方，講師がデバッグ方法を付加してもその 7 割が学生によるエラーの自己解決に利用されないのは効率が悪いと考える．

今後，エラー情報の類似度の算出方法を改善するなど，助言の可能性を高める方法を検討したい．また，最終的には，プログラミング演習中に学生が発生したエラーに対して自己解決を促す支援システムを開発していきたい．

参考文献

[1] 高橋圭一：ログファイルと Git リポジトリを用いた Ruby on Rails の初学者の躓き要因の分析,情報教育シンポジウム 2020 論文集, pp. 69-74 (2020).
[2] 高橋 圭一：Web アプリケーション開発フレームワークの学習進度推定ツール，ソフトウェア工学の基礎ワークショップ 2021 論文集, pp. 97-102 (2021).
[3] 崔恩瀞，藤原裕士，吉田則裕，水野修：コードクローン検索手法の調査，コンピュータソフトウェア, 39(3), pp. 3_47-3_59 (2022).