# Ruby on Rails の初学者の躓き要因とデバッグ難易度に関する分析

髙橋圭一 [1]

**概要**：我々はこれまで Ruby on Rails を用いた Web アプリケーション開発科目の提出物であるログファイルと Git リポジトリを用いて受講者の躓き要因を分析してきた．その結果，受講者が課題着手中に発生した例外は 9 つであること，また，それぞれの例外が発生した原因として 11 個の誤りパターンがあることが確かめられた．本稿では，それぞれの例外を修正するための試行回数と修正時間を調査し，例外ごとのデバッグの難易度を明らかにする．

**キーワード**：Ruby，プログラミング演習，開発フレームワーク，デバッグ

# Novice programming mistakes and difficulty of debugging in learning Ruby on Rails

KEIICHI TAKAHASHI[†1]

***Abstract***: We have analyzed the log files and Git repositories submitted as assignments in a web application development course using Ruby on Rails to determine the causes of students' mistakes. The analysis showed that there were nine exception errors that occurred while students were working on their assignments, and that there were eleven error patterns that caused the exception errors to occur. In this paper, we investigate the number of attempts to fix each exception and the time to fix it, and we clarify the difficulty level of fixing each exception for the students.

***Keywords***: Ruby, Programming Exercise, Development Frameworks, Debugging

## 1. はじめに

筆者が所属する学科に 3 年生対象の Web アプリケーション開発の科目がある[1][2]．その科目では 15 回の授業のうち後半の 6 回分で Ruby on Rails（以降，Rails）を用いた Web アプリケーション開発（以降, Rails 開発）を学習する．この科目の第 11 回では画像アップロード機能を含んだ Rails 開発について学習する．2019 年度の授業で 30 名の受講者から提出された Rails のログファイルを分析したところ，受講者が躓いたことを示す例外のエラーメッセージは 9 つであり，そのうち ActionView::Template::Error と ActionController::RoutingError という例外を約 8 割の受講者が発生したことがわかった[3]．この 2 つの例外の発生原因はログファイルの情報だけでは特定できなかったため，例外発生時に Git リポジトリに自動的にコミットするスクリプトを開発し，2020 年度の同授業に適用したところ，例外は 11 個の誤りパターンによって発生することが特定できた[4].

これまでの研究成果を表 1 に示す．表中の躓いた受講者率は，受講者から提出されたログファイルをもとに例外別にエラー発生回数を求め，エラー発生回数が 1 以上の受講者数を集計し，これを全受講者数で割った値である．表中の誤りパターンは, それぞれの例外が発生した原因であり，筆者が指導中に観察した誤り事例に番号を付したものである．詳細は 2.2 節にて述べる．

表 1 例外クラス名ごとの躓いた受講者率と誤りパターン

| 例外クラス名 | 躓いた受講者率 | | 誤り |
|---|---|---|---|
| | 2020年度 | 2019年度 | パターン |
| ActionView::Template::Error | 0.85 | 0.79 | 4, 5, 6, 7, 8 |
| ActionController::RoutingError | 0.39 | 0.75 | 9, 10, 11 |
| SyntaxError | 0.39 | 0.67 | 8 |
| NoMethodError | 0.45 | 0.53 | 8 |
| NameError | 0.39 | 0.40 | 8 |
| ActiveRecord::PendingMigrationError | 0.00 | 0.13 | 2 |
| ActiveModel::UnknownAttributeError | 0.03 | 0.03 | 3 |
| Encoding::UndefinedConversionError | 0.00 | 0.03 | ※1 |
| LoadError | 0.03 | 0.03 | 1 |

※1 2019年度はソースコードを取得していないため誤りパターンを特定できていない

これまでの研究で，本科目の手順にしたがって演習を進めたときに受講者が陥る誤りパターンと誤りパターンによって 受講者が発生した例外の割合が明らかになった. 躓いた受講者率が高い例外を抽出し，その誤りパターンを特定することで指導方法や支援方法の検討に進めると考えていた．しかし，躓いた状態から正常状態に復帰するまでの過程，つまり，デバッグ過程は現在のところ未知であり，もし，デバッグの難易度が高い例外や誤りパターンがあるとすれば，躓いた受講者率の大小に関わらず，難易度を考慮

1 近畿大学産業理工学部情報学科
Kindai University

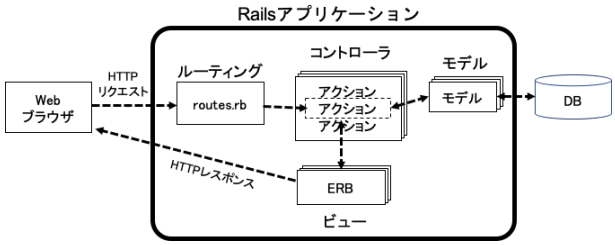


図 1 Rails の基本的な構成

した支援する方法を検討する必要があると考えた．

そこで本稿では，[4]と同様のデータをもとに，受講者がそれぞれの例外に遭遇したときに，どのようにデバッグし修正したのか分析を試みる．研究を進めるために，以下の2つのリサーチクエスチョンを立てる．

**RQ1**：例外ごとのデバッグの難易度に違いはあるのか？違いがあるとすれば，その原因は何か？

**RQ2**：受講者ごとのデバッグの難易度に違いはあるのか？違いがあるとすれば，その原因は何か？

以降，2 節で Rails 開発とその手順を示しながら，表 1 の誤りパターンについて述べる．3 節では，ログファイルから受講者のデバッグ難易度を取得する方法について述べる．4 節に実験方法や前提条件について述べ，5 節にデバッグの難易度に関する調査結果を示し，6 節で考察する．8 節で関連研究を紹介し，最後にまとめと今後の展開を述べる．

## 2. Rails 開発と誤りパターン

### 2.1 Rails の構成

開発者は，Rails が採用している MVC（Model View Controller）アーキテクチャを理解し，rails コマンドなどの各種ツールを使いこなすことで品質の高いソフトウェアを効率的に開発できる．図 1 に Rails の基本的な構成を示す．

Rails アプリケーションが Web ブラウザから HTTP リクエストを受信すると，ルーティング（routes.rb）の記述内容に従って，コントローラ（Ruby のクラス）内のアクション（Ruby のメソッド）が呼び出される．呼び出されたコントローラはモデル（Ruby のクラス）およびビュー（ERB ファイル*）を呼び出し，Web ブラウザに HTTP レスポンスを返す．MVC のそれぞれのファイルは rails コマンドを実行することで雛形を自動生成できるため，開発者はクラス定義など定型的なコードを記述する必要はない．

こうした仕組みは開発者にとっては便利であるが，初学者は Rails の仕組みや規約の理解が不十分であるため，エラーを起こさずにルーティングや MVC のモジュールを組み合わせることは困難である．一方，Rails のような開発用フレームワークは構成要素が多いため，同じエラーメッセージでもその発生原因は様々な状況が考えられる．こうしたデバッグの難しさが初学者の学習を困難にすると考えている．

### 2.2 Rails 開発手順と初学者の誤りパターン

授業で取り上げた，画像を共有する Rails アプリケーション（myapp）の開発手順を示しながら，筆者がこれまで学生指導で観察した初学者の誤りパターン（EP1～11）について述べる．

① まず，プロジェクトを新規作成する．操作は *rails new myapp* である．この操作により，カレントディレクトリに myapp フォルダが新規作成され，その中に必要なファイル群が生成される．最初に，タイムゾーンや使用言語や使用するライブラリを設定ファイルに追記する．ここで記述を誤ると実行時にエラーが発生する（EP1）．

② 次にモデルを作成する．操作は *rails generate model Image title:string file:binary* である．この操作により，String 型の title と Binary 型の file のインスタンス変数をもつ Image クラスと，このモデルがアクセスするテーブル作成用のスクリプトファイルが生成される．*rails db:migrate* という操作で，前述のスクリプトファイルが実行されテーブルが作成される．この操作を忘れて Rails アプリケーションを実行するとエラーが発生する（EP2）．また，モデル名やカラム名や型の記述を誤る場合もある（EP3）．

③ 次にコントローラを作成する．操作は *rails generate controller images index* である．この操作によって，一覧表示のための index アクションを含んだ ImagesController クラスと index.html.erb というビューファイルが生成される．なお，基本的にモデル名は単数形で，コントローラ名は複数形である．このルールを忘れてコーディングするとエラーが発生するほか（EP4），手動でアクションを追加する場合は，ルーティングに対応するアクション名の誤りや，アクションを追加し忘れてエラーが発生する場合もある（EP5）．

④ 画像情報を一覧表示するため，開発者は，テーブルから読み込んだデータをインスタンス変数@images にセットするコードを ImagesController の index アクションに追加する（図 2）．この@images を用いて一覧表示する HTML を，ビューである index.html.erb に追加する（図 3）．コントローラとビューの別々のファイルの連携は Rails によって暗黙的に行われるため，以下のような誤りパターンによって様々なエラーに遭遇する場合がある．

- コントローラでテーブルデータをセットする変数に@をつけ忘れてしまい（ローカル変数になる），ビューで変数を受け取れない（EP6）
- コントローラとビューに記述したインタンス

* Ruby コードを埋め込める HTML ファイル

```
class ImagesController < Application
  def index
    @images = Image.all
  end
end
```

図 2 コントローラ記述例

```
<% @images.each do |img| %>
  <p>タイトル: <%= img.title %></p>
<% end %>
```

図 3 ビュー記述例（index.html.erb）

```
Rails.application.routes.draw do
  get 'images/index', to: 'images#index'
end
```

図 4 ルーティング記述例（routes.rb）

- 変数名が一致しない（EP7）
- ビューファイルでの Ruby の記述誤り（EP8）

⑤ 最後に, Web ブラウザからアクセスされたパスとコントローラを紐つける定義を routes.rb に追加する（図 4）．ここでは以下のような誤りが考えられる．

- コントローラとビューを追加したが，ルーティングの定義を追加し忘れる（EP9）
- ルーティングの定義の記述を間違える（EP10）
- コントローラとビューを追加し，ルーティングの定義も正しく追加したが, A タグや FORM タグなどでパス名の記述を誤る（EP11）

## 3. 分析手法

### 3.1 ログファイルの構成

開発中の Rails アプリケーションの実行ログは development.log というファイルに書き出される．このログファイルは開発者が意図的に削除しない限り, Rails プロジェクト作成直後から現在まですべての情報が記録される．

図 5 にログファイルの例を示す．以下，(A)〜(C)の内容について説明する．

(A) Web ブラウザからアクセスされると Started で始まる情報が日時とともに記録される．この例では”/images/new”のパスに対して HTTP リクエストの GET メソッドで要求があり，Completed で始まる情報が出力され，HTTP レスポンスとしてサーバエラーが発生したことを示すステータスコードである 500 番が記録されている．

(B) Rails アプリケーションのコントローラやビューを実行したときに例外が発生すると, 例外のクラス名とエラー情報がログファイルに出力される．この例では，ActionView::Template::Error という例外名とともに undefined local variable or method というエラーメッセージとソースコード片とエラー箇所（new.html.erb の 1 行目）が出力されている．form_for と書くべきところを from_for とタイプミスしたためこの例外が発生した．

```
(A) Started GET "/images/new" for xxx.xxx.xxx.xxx at 2020-07-16 02:39:48 +0000
    （省略）
    Completed 500 Internal Server Error in 98ms (ActiveRecord: 0.0ms)

(B) ActionView::Template::Error (undefined method `from_for' for #<#<Class:0x0000
    Did you mean?  form_for):
      1: <%=from_for @image,url:{action: :create} do|f|%>
      2: <p><%=f.label :title,'タイトル'%>
      3: <%=f.text_field :title%></p>
      4: <%=f.submit'登録'%>

    app/views/images/new.html.erb:1:in `_app_views_images_new_html_erb__2388
    （省略）

(C) Started GET "/images/new" for xxx.xxx.xxx.xxx at 2020-07-16 02:39:48 +0000
    （省略）
    Completed 200 OK in 46ms (Views: 44.9ms | ActiveRecord: 0.0ms)
```

図 5 ログファイル（development.log）の例

(C) 上記の（A）と同じ URL にアクセスして，正常終了した場合の例である．サーバ処理が正常に終了したため, Completed ではじまる行に成功レスポンスのステータスコードとして 200 番が記録されている．

### 3.2 ログファイルを用いたデバッグ難易度の取得方法

3.1 節に示したログファイルの情報をもとに受講者のデバッグの難易度を取得する手順を以下に示す．受講者にとってのデバッグの難易度には様々な要因が関係するが，本稿では，誤りを修正するまでの試行回数と修正時間が多いほど難易度が高いと考える. なお, ログファイル中の Started から Completed までのログ情報をセッションと呼ぶ.

(1) Stared で始まる行があれば新規セッションの開始とする．

(2) Completed で始まる行があれば，当該セッションの終了とし，HTTP レスポンスのステータスコードが 500 番の場合，次行以降に現れる例外情報（図 5 の場合，ActionView で始まる行全体）を記録する．なお，当該セッションの開始行に示された日時を例外発生時刻，かつ，**デバッグ開始時刻**とする．

(3) 上記(2)で例外情報が発生した場合，例外情報と HTTP リクエストの情報が同一である次行以降のセッションを探索する．

(3-1) HTTP レスポンスのステータスコードが 500 番のとき，同一エラーを繰り返しているとみなし，当該の例外の**繰り返し回数**をカウントする．

(3-2) HTTP レスポンスのステータスコードが 200 番†のとき，受講者によるデバッグが終了したとみなし，当該セッションの開始行に示された日時を**デバッグ終了時刻**とする．

以上の手順により，ログファイルに記録された例外ごと

† サーバ処理で正常終了したあと，他のページに転送する場合があるため，200 番の他，300 番台のステータスコードも同様に処理する

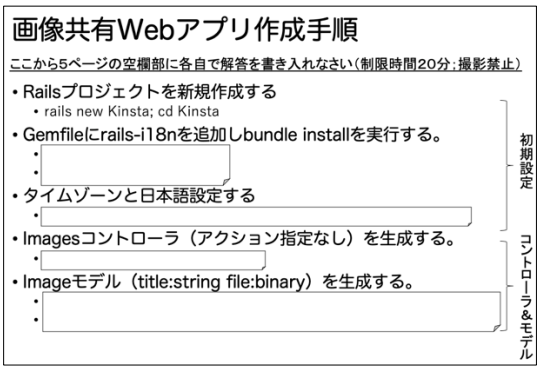


図 6 講義スライド例

に繰り返し回数，デバッグ開始時刻，デバッグ終了時刻が得られる．これらの値を利用して，デバッグ作業の難易度を示す 2 つの指標を計算する．

- 試行回数：繰り返し回数
- 修正時間：デバッグ終了時刻 − デバッグ開始時刻

## 4. 実験

### 4.1 実験方法

本稿の冒頭でも述べたように，筆者が所属する学科の 3 年生対象の Web アプリケーション開発の科目を対象とする[1][2]．本科目は，講義 1 コマと演習 1 コマが時間割で連続しており，2019 年度には講義で基礎を学んだあと，演習室に移動して課題に取り組んだが，2020 年度は，講義も演習もすべて Zoom を利用したオンライン授業形式で実施した．15 回の演習のうち，第 8 回までは Ruby の応用プログラミングを学び，第 9 回から 6 回に渡って Rails 開発について学習する．第 9 回と第 10 回に CRUD アプリケーションの作成方法を学習し，第 11 回にファイルアップロード機能をもつ画像共有アプリケーションの作成方法を学習する．開発手順は 2.2 節に示した通りである．例外発生時に Git リポジトリに自動的にコミットするスクリプトを受講者全員に配布し，課題に取り掛かる前に実行するように依頼した[4]．この第 11 回に受講者が取り組んだ Rails アプリケーションのログファイルと Git リポジトリの情報を収集する．なお，ログファイルも Git リポジトリも Rails プロジェクトを新規作成したときに自動生成される標準的なファイルであり，課題提出のための特別な操作は不要である．課題の提出期間は 1 週間であったが，2020 年度の受講者 57 名のうち 33 名から期限内に課題提出があった．提出された課題はすべてアプリを完成しており，躓いても何らかの方法で自己解決できたとみなせる．

### 4.2 講義及び課題の内容

第 9 回と第 10 回で基本的な手順は学習済みのため，第 11 回の講義では，開発手順の定着を促すため，入力すべきコマンドやプログラムを空欄にした資料を配布し，講義中に解説しながら受講者に記入させた（図 6）．演習課題は「講義資料をもとに画像共有 Web アプリを完成させなさい」であり，講義中の解説をもとに受講者が資料を正確に完成していれば，課題完成までに各自で試行錯誤を必要としない．2020 年度では，前述のとおり講義も演習もすべて Zoom を利用したオンライン授業形式で実施し，講義終了後には，録画した講義動画を受講者に公開したため，受講者は必要であれば繰り返し視聴して手順を確認できる環境であった．

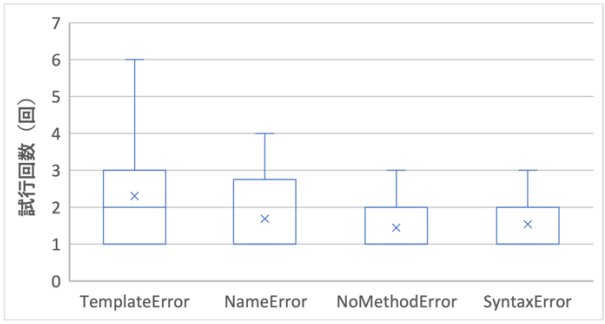


図 7 発生した例外を修正するまでの試行回数

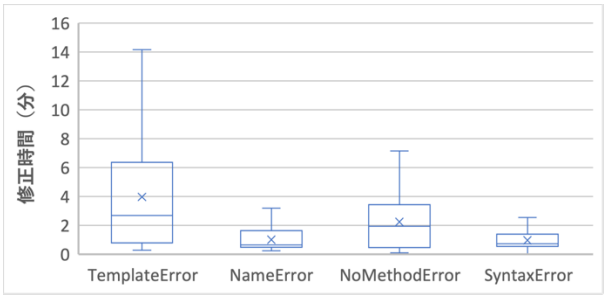


図 8 発生した例外を修正するまでの修正時間

## 5. 実験結果

2020 年度の提出物では，表 1 に示すように，9 つの例外のうち 2 つは発生しなかった．また，LoadError と ActiveModel::UnknownAttributeError はともに発生件数が 1 だったため分析対象から除外した．さらに，ActionController::RoutingError については，発生件数としては十分であったが，3.2 節に示した方法では修正時間の取得が困難であったため除外した．したがって，実験結果として示すのは ActionView::Template::Error を含む 4 つの例外である．

### 5.1 例外ごとのデバッグ難易度

4 つの例外のデバッグの難易度を示す試行回数と修正時間の箱ひげ図を図 7 と図 8 に示す．それぞれの例外のデータ数は 62(3)，16(0)，19(1)，24(4)である．データ数の括弧内の数値は外れ値として除外した数を表している．図中の×印は平均値である．平均値はいずれの例外でも中央値より上回っており，半数以上の受講者がデバッグに時間を要したことがわかる．

グラフからは読み取りにくいが，試行回数の中央値は TemplateError から 2，1，1，1 であり，修正時間の中央値は TemplateError から 2.7，0.7，1.9，0.7 である．これらの値を見ると，TemplateError とその他の例外で 2〜4 倍程度の差

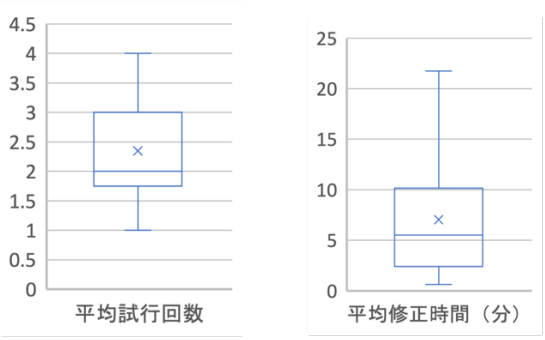


図 9　発生した例外を修正するまでの平均試行回数と平均修正時間

がある．そこで TemplateError とその他の例外の 2 群についてマンホイットニーの U 検定を行ったところ試行回数と修正時間はともに統計的に有意差が認められた（$p<0.01$）．

### 5.2 受講者ごとのデバッグ難易度

受講者が発生した例外を修正するまでの試行回数と修正時間を受講者ごとに集計し，受講者が発生した例外の個数で割った値を求める．それぞれ平均修正回数，平均修正時間とし，その箱ひげ図を図 9 に示す．平均試行回数の中央値は 2 回で，平均修正時間の中央値は 5.5 分であった．それぞれの最大値は 4.0 回と 21.7 分であることから，最大値の受講者は，中央値付近の受講者と比べて，発生した例外を解決するまでに約 2 倍の試行を繰り返し，約 4 倍の修正時間を要することがわかった．

## 6. 考察

### 6.1 RQ1：例外ごとのデバッグの難易度に違いはあるのか？違いがあるとすれば，その原因は何か？

5.1 節で述べたように，TemplateError とそれ以外の例外では，デバッグの難易度を示す試行回数と修正時間はともに統計的に有意差があることが確かめられたことから，少なくとも TemplateError とそれ以外の例外についてはデバッグの難易度に差があると言える．

図 8 より，NameError，SyntaxError の修正時間の中央値はいずれも 1 分未満である．テキストベースのプログラミング言語でプログラミングするため，テキストエディタの自動補完機能を使用したとしてもタイプミスを防ぐことは困難である．例外の発生回数にもよるが，タイプミスをしても 1 分未満で誤りを発見し修正できるのであれば演習遂行への影響は小さく，むしろプログラミング上達のためのデバッグ経験としては有益であり，特別な支援を必要ないと考える．一方，NoMethodError と TemplateError の修正時間の中央値は 2 分以上であり，特に TemplateError の最大値は中央値の約 5 倍の 14.2 分であることから対策が必要と考える．

次に，TemplateError とそれ以外の例外の難易度の違いの原因を調べる．表 1 に示したように，TemplateError の誤りパターンは EP4～8 の 5 種類である．それぞれの誤りパターンの発生割合は[4]で調査済みであるため表 2 に結果を再掲する．発生割合をみると EP8 が最も多く，続いて EP5 が多く発生していることがわかる．これら誤りパターンのデバッグ難易度を分析するため，それぞれの修正時間の調査結果を表 3 に示す．

表 2　ActionView::Template::Error の誤りパターンの発生割合

| 誤りパターン | 発生割合 |
|---|---|
| コントローラ名を複数形にしてない（EP4） | 0% |
| コントローラのアクション定義忘れ（EP5） | 24% |
| ビューに渡す変数に@のつけ忘れ（EP6） | 4% |
| コントローラ・ビューの変数名の不一致（EP7） | 6% |
| ビューで文法誤り（EP8） | 66% |

表 3　誤りパターンごとの修正時間（分）

| | EP5 | EP6 | EP7 | EP8 |
|---|---|---|---|---|
| 平均 | 8.7 | 17.5 | 0.8 | 10.2 |
| $\sigma$ | 15.3 | － | － | 15.5 |
| 中央値 | 5.9 | － | － | 2.9 |
| N | 16 | 1 | 1 | 27 |

EP5・8 は EP6・7 と比べて十分にデータ数があるため統計的に傾向を分析できる．EP5 の中央値は EP8 の中央値の 2 倍であり，EP8 と比べると EP5 のデバッグ難易度が高いことが推察できる．例外発生時のソースコードを調査したところ，誤りの原因はタイプミスもしくは入力忘れであった．タイプミスはソースコードの至るところで発生する可能性があるが EP5 と EP8 の修正時間にこうした差が生じたのはなぜだろうか．

EP5 と EP8 のエラーメッセージの例を図 10 と図 11 に示す．EP5 と EP8 のエラーメッセージを読むと，ビューである ERB ファイルに誤りがあると指摘されている．EP8 では，エラーメッセージの指摘の通り，ERB ファイルの 2 行目を image とすべきところ@image とタイプミスした．受講者がエラーメッセージを適切に読んでいれば誤りを発見するのは容易である．一方，EP5 のエラーメッセージでは，ERB ファイルの 1 行目に誤りがあると指摘されているが，この場合は，ERB ファイルには問題はなく，データフローの上流側であるコントローラファイルに@images が未定義であるという誤りが原因であった．つまり，EP8 と比べて EP5 の方が探索範囲が広いため修正時間が多くなることが推察できる．Rails の仕組みとしては EP6・7 も EP5 と同様であり，EP8 と比べて修正時間は増加する可能性が高い．なお，NameError，NoMethodError，SyntaxError は EP8 と同様にエラーメッセージで指摘されたファイル内に誤りが存在することがほとんどであり，それぞれの修正時間の中央値が 3 未満と EP8 の中央値とほぼ同じであることからデバッグの難易度は低いと言える．

```
ActionView::Template::Error (undefined method `each' for nil:NilClass):
  1: <% @images.each do |image| %>
  2: <p><%= image.title %></p>
  3: <% end %>
  4:
app/views/images/index.html.erb:1:
```

図 10　EP5 のエラーメッセージ例

```
ActionView::Template::Error (undefined method `title' for nil:NilClass):
  1: <% @images.each do |image| %>
  2:   <p><%= @image.title %></p>
  3: <% end %>
app/views/images/index.html.erb:2:
```

図 11　EP8 のエラーメッセージ例

### 6.2 RQ2：受講者ごとのデバッグの難易度に違いはあるのか？違いがあるとすれば，その原因は何か？

5.2 節で述べたように, 平均試行回数と平均修正時間の分布の調査結果より，中央値と最大値では 2〜4 倍の違いがあることがわかった．デバッグの難易度の違いが生じた原因について考えると，受講者の論理的思考力，プログラミングの習熟度，科目修得や課題提出に対するモチベーション，躓いた箇所の誤り発見の容易さなどの複数の因子が考えられ，お互いに複雑に影響することが予想されるが，今回の研究でログファイルから抽出した情報は試行回数と修正時間のみであり，この情報のみで受講者間のデバッグの難易度が生じた原因をボトムアップ的に説明しするのは困難である．一方，6.1 節で述べたように，TemplateError と TemplateError 以外の例外にはデバッグ難易度に差異があることがわかった．このことから，TemplateError を多く発生した受講者はデバッグ難易度が高くなり，試行回数と修正時間が多くなるのではないだろうか．そこで，各受講者の TemplateError の発生個数と試行回数および修正時間についてスピアマンの順位相関係数を求めたところ，試行回数と TemplateError 個数の相関係数は 0.69 と強い相関があり，修正時間と TemplateError 個数の相関係数は 0.50 と相関があることがわかった．無相関検定を行ったところ，いずれも有意水準 5%で相関関係があることがわかった．

## 7. 関連研究

**フレームワーク学習の困難さ**[5]： Coker らは Android と ROS（Robotic Operating System）のフレームワークを利用するソースコードに欠陥を埋め込んだタスクを用意し，被験者がデバッグする様子を記録し分析した．結果として，被験者はソフトウェア開発時に共通する課題に対する解決策をフレームワークが提供することに利点を感じる一方，フレームワークの抽象概念を理解する困難さ感じており，特に制御の反転（Inversion of Control）の理解に苦慮したと報告されている．制御の反転とは，開発者が書いたコードの実行タイミングや実行順がフレームワークによって決定されることを表している．我々の研究でも ActionView::Template::Error と ActionController::RoutingError という例外を約 8 割の受講者が発生したが[3]，この原因はまさにこの制御の反転の理解不足によると考えている．

**複数ファイルを扱う学習者の認知的負荷**[6]： Fronza らは，プログラム作成時に複数のファイルを往来するときの開発者の認知的負荷を cognitive shift（CS）として捉え，CS と成績との関係について調査した．具体的には，166 名の学生に単一ファイルで解答する課題と複数ファイルで解答する 2 つのプログラム課題を与え，学生らのキー入力情報から単一ファイル内の CS と複数ファイルを切り替える CS を求め，成績との相関を調査した．結果としては，単一ファイル内の CS より複数ファイルの CS の方が成績と強い負の相関があることを示した．フレームワークの初学者にとっては，前述の制御の反転の理解が問題として認識されているが，Rails 開発ではルーティングや MVC に関わる複数のファイルを往来する必要があり，そのときの CS もフレームワークの学習を困難にする要因の 1 つと考えられる．

**ログ情報を用いた学習者の行動分析**[7][8]：学生の躓きをログ情報を用いて自動検出する提案として浦上らの研究がある．プログラム開発環境 Bit Arrow には，実行のたびに実行時のソースコードなどの編集履歴情報をログ情報として書き出すログ収集機能がある．このログ情報に含まれる正解のソースコードと各時点のソースコードの一致行数を比較することで学生の躓き状態を把握できることを示した．Meier らは，オンライン上で Python IDE である Thonny を利用したプログラミング入門コースを公開し 1,518 名の受講者からログ情報を得た．そのログ情報を Thonny に読み込み，受講者のプログラミング過程をリプレイして観察することによって分析した．いずれの研究もプログラム開発中の学習者の行動や状態の分析に重きがある．我々は初学者の躓き情報からフレームワーク自体の問題点や学習時に支援が必要となる学習箇所を明らかすることを目指している点が異なる．

**バグ修正時間によるエラー別の修正難易度分析[9]**：Alqadi らは，初学者にとって修正困難な論理エラーを調査するため，C/C++言語を使用した場合の 10 種類の論理エラーを選定し，その論理エラーを 1 種類ずつ埋め込んだ短いコード片を用意し，83 名の学生にエラーを修正させる実験を実施し，修正時間と正解率を得た．結果として，エラー種別によって修正時間と正解率にばらつきがある，つまり，学生にとって修正が難しいエラーとそうではないエラーが存在することがわかった．この背景として著者らは，学生らが過去に課題などで経験したことがあるエラーは修正の難易度が低下する傾向にあると述べ，また，被験者である学生のプログラミング経験年数の長短が，正解率および修正時間に相関があることも示された．本研究では，被験者全体では例外によってデバッグ難易度に有意差があることが認められる結果となったが，受講生の中にはプログラミング

経験が豊富な学生も存在するだろう．本稿では，例外ごとのデバッグ難易度を整理したが，プログラムの修正時間に対する受講生のプログラミング経験年数の影響を調べていく必要があるだろう．

## 8. まとめ

本稿では，それぞれの例外を修正するための試行回数と修正時間を調査し，例外ごとのデバッグの難易度を調査した．結果としては，TemplateError とそれ以外の例外については差異があることがわかった．差異が生じた原因として考えられるのは，TemplateError では，エラーメッセージで指摘されたビューファイルではなく，データフローの上流であるコントローラに誤りがある場合が多く，誤り箇所の特定に時間を要したことが原因と考えられる．一方，受講者間のデバッグ難易度の違いについては，平均試行回数と平均修正時間において，それぞれ中央値と最大値で 2〜4 倍の違いがあることがわかったが，これら個人差の原因を今回の研究で得られた情報で説明するのは困難であった．しかし，受講者が発生した TemplateError の個数とデバッグ難易度の間には相関があることが示されたことから，TemplateError が発生したときのデバッグ方法や事例紹介などに時間を割くことでプログラムや課題の完成率を高める可能性が示唆された．本稿の調査の結果，デバッグ時間に影響する因子として，エラーメッセージの指摘箇所と実際の箇所との距離が関係している可能性が示唆された．今後の研究で関連性について分析を進めていきたい．

## 参考文献


[1] 高橋圭一, Ruby on Rails による Web アプリ開発の授業実践, 情報処理学会九州支部火の国シンポジウム 2018, 2019.

[2] 高橋圭一，Ruby on Rails によるチーム開発の授業実践, 情報処理学会 情報教育シンポジウム 2019, pp.10-16, 2019.

[3] 高橋圭一，Ruby on Rails の初学者の躓き要因分析，ソフトウェア工学の基礎 XXVII，日本ソフトウェア科学会, pp.103-108, 2020.

[4] 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，情報処理学会 情報教育シンポジウム 2020, pp.1-7, 2020.

[5] Zack Coker, David Gray Widder, Claire Le Goues, Christopher Bogart, and Joshua Sun-shine, A qualitative study on framework debugging, International Conference on Software Maintenance and Evolution, pp.568-579, 2019.

[6] Ilenia Fronza, Arto Hellas, Petri Ihantola, Tommi Mikkonen, An Exploration of Cognitive Shifting in Writing Code, Proceedings of the ACM Conference on Global Computing Education, pp.65-71, 2019.

[7] 浦上理，長島和平，並木美太郎，兼宗進，長慎也，プログラミング学習者のつまずきの自動検出，情報処理学会 研究報告コンピュータと教育（CE），2020-CE-154，No.4，pp.1-8, 2020.

[8] Heidi Meier, Eno Tõnisson, Marina Lepp, Piret Luik, Behaviour Patterns of Learners while Solving a Programming Task: an Analysis of Log Files, IEEE Global Engineering Education Conference (EDUCON), pp.27-30,2020.

[9] Basma S. Alqadi, Jonathan I. Maletic, An Empirical Study of Debugging Patterns Among Novices Programmers, Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, pp.15-20, 2017.