情報処理学会論文誌　教育とコンピュータ　Vol.8　No.1　24–34 (Feb. 2022)

研究論文

Ruby on Railsの初学者のつまずき要因の分析支援ツール

高橋 圭一[1,a)]

受付日 2021年2月2日，再受付日 2021年7月22日 / 2021年9月29日，
採録日 2021年11月9日

概要：Ruby on Rails（以降，Rails）は Ruby で書かれたオープンソースの Web アプリケーションフレームワークである．Rails を用いたシステム開発や Rails の機能拡張の提案などは公開直後から研究が進められているが，Rails などの Web アプリケーションフレームワークの学習過程に関して詳細に調査した研究は少ない．我々はこれまで，筆者が所属する学科の Web アプリケーション開発科目の演習課題として提出されたログファイルを分析し，受講者がつまずいたことを示す例外は 9 つあり，そのうち 2 つの例外の発生原因はログファイルだけでは特定が困難な場合があるという結果を得た．この 2 つの例外のうちログファイルだけでは特定が困難な誤りを HIEs（Hard to Identify Errors）と呼ぶ．本稿では，バージョン管理ソフトウェアの 1 つである Git を用いて例外発生時のソースコードを自動的に保存し，ログファイルと組み合わせてエラー情報を可視化するツールを用いることにより HIEs の発生原因の特定を試みる．2020 年度の授業に本ツールを適用したところ，33 名から提出されたログファイルから，HIEs が 219 件発生し，Git リポジトリの提出があった 148 件については，その情報を活用することですべての発生原因を特定できた．

キーワード：Ruby，プログラミング演習，開発フレームワーク

A Tool to Support Analysis of Error Factors Made by Novices in Ruby on Rails

Keiichi Takahashi[1,a)]

Received: February 2, 2021, Revised: July 22, 2021/September 29, 2021,
Accepted: November 9, 2021

Abstract: Ruby on Rails (Rails) is an open-source web application framework developed in the Ruby programming language. Soon after the release of Rails, various studies, including those on development of application systems using Rails and proposals for extending its functionality, were conducted. However, few studies have investigated the learning process of web application framework such as Rails in detail. We analyzed the log files of web application development assignments submitted in our department. Our analysis revealed that nine exceptions caused students to make mistakes. In addition, we observed that the causes of two exceptions may be difficult to identify from the log files alone. These two exceptions difficult to identify from the log files alone are referred to as hard to identify errors (HIEs). In this study, we attempt to identify HIEs by using a tool that automatically saves the source code when an exception occurs using Git, a version control software; the tool visualizes error information by combining the source codes and log files. When using the tool for a class (in 2020), 219 HIEs were generated by 33 students, and for 148 of those HIEs; when the Git repository was included in the files submitted by the students, the causes of all HIEs that occurred could be identified.

Keywords: Ruby, programming exercise, development frameworks

1. はじめに

Rails は Ruby で書かれたオープンソースの Web アプリケーションフレームワークである．公開されてから 10 年

[1] 近畿大学
Kindai University, Iizuka, Fukuoka 820–8555, Japan
[a)] ktakahas@fuk.kindai.ac.jp

以上経過しており，GitHub やクックパッドなど国内外に多くの利用者を抱えるサービスでも採用されている．Ruby の開発者が日本人であるため，日本語で読めるリソースがインターネット上に豊富にあり，国内の初学者が学びやすいことも特徴の 1 つと考えられる．Rails のリリース直後から，様々な研究が進められており，2020 年 9 月に Google Scholar で “Ruby on Rails” を指定してフレーズ検索したところヒット件数は約 11,000 であった．検索結果を分類すると，(a) Rails を用いた応用システム開発 [1]，(b) Rails の機能拡張の提案 [2]，(c) Rails と他フレームワークの比較 [3]，(d) 大学における PBL 実施報告 [4], [5]，の 4 種であった．このうち (d) はチーム開発やアジャイル開発など総合的なソフトウェア開発の教育メソッドとして計画し実施された報告である．特に大学院ではこうした Web アプリケーションフレームワークを活用した実践的な演習が実施されているが，初学者の学習過程について詳細に調査した研究は少ない．

初学者が Rails を学習する手段としては，Rails チュートリアルや ProGate などのオンラインドキュメント [6], [7]，ドットインストールや Udemy などの動画教材 [8], [9]，書籍，プログラミングスクールがある．いずれも単独もしくは閉じた教育組織での学習であるため，学習者のつまずきは共有されていない．Rails は 2019 年 8 月にバージョン 6 がリリースされており，今後も Web アプリケーションを開発する有効な手段として活用されることが期待できる．そこで，公開後 10 年以上経った状況ではあるが，初学者にとって Rails を学びやすくするための知見の蓄積は今後も必要であると考えた．

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．この科目の受講者らがつまずいた要因を分析するために，演習課題として提出された Rails のログファイルを分析したところ，受講者がつまずいたことを示す例外は 9 つあり，そのうち，約 8 割の受講者が発生した ActionView::Template::Error と ActionController::RoutingError の例外については，ログファイルのみでは例外の発生原因の特定が困難な場合があることが判明した [10], [11], [12]．これら ActionView::Template::Error と ActionController::RoutingError の例外をすべて FOEs（Frequently Occurred Errors）と呼ぶ．FOEs の中にはログファイルだけで発生原因の特定が可能な場合もあるが，ログファイルのみでは特定できない例外もある．これを HIEs（Hard to Identify Errors）と呼ぶ．FOEs と HIEs の関係を**図 1** に示す．

本稿では，この HIEs の発生原因を特定するため，ログファイルを監視し，つまずき要因を表すキーワードがログファイルに追加されたときに，変更されたソースコードを自動的に保存し，ログファイルと組み合わせてエラー情報を可視化するツール FOEs Viewer を提案する．このツールを前述の Web アプリケーション開発の科目に適用し，HIEs の発生原因の特定の可能性について評価する．

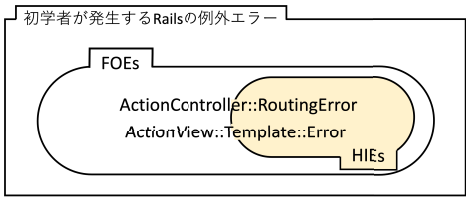

図 1　FOEs と HIEs の関係
Fig. 1　Relationship between FOEs and HIEs.

2. Rails 開発と誤りパターン

2.1 Rails の構成と初学者のデバッグの難しさ

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 2** 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．また，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 デバッグ難易度の分類

本稿では Ruby on Rails を対象としているが，2.1 節で述べたような初学者にとってのデバッグの困難さは，MVC アーキテクチャを採用している他のフレームワークでも共通する問題である．本節では Web アプリケーションフレームワークにおけるデバッグの困難さを**表 1** のように定義する．

レベル 1 に該当する主な誤りは文法誤りである．具体的には，メソッド内で定義あるいは初期化した変数名と使用する箇所とで変数名が不一致である誤り，あるいは，変数名は一致していても，変数の値が空であるためメソッド呼び出しで例外が発生する誤りである．レベル 1 の場合には，エラーメッセージで指摘された箇所を調べればよく，

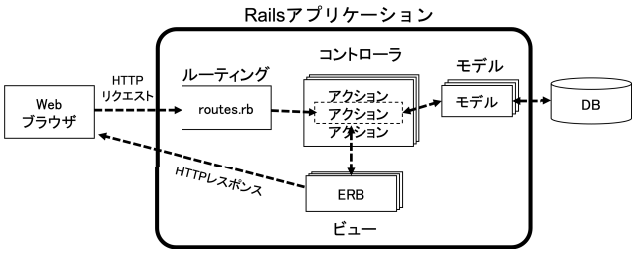

図 2　Rails の MVC の構成図
Fig. 2　A diagram of MVC in Rails.

表 1　デバッグ難易度の分類
Table 1　Classifying the difficulty of debugging.

デバッグ難易度	説明
レベル 1（容易）	エラーメッセージで指摘されたファイルを修正すれば直る
レベル 2（困難）	エラーメッセージで指摘されたファイルは正しく，それ以外のファイルを修正すれば直る

表 2　誤りパターンのデバッグ難易度
Table 2　Difficulty of debugging for each error pattern.

デバッグ難易度	誤りパターン
レベル 1	EP1，EP2，EP8
レベル 2	EP3，EP4，EP5，EP6，EP7，EP9，EP10，EP11

比較的容易に誤りを発見できる．

一方，レベル 2 の主な誤りは，レベル 1 の誤りが複数のファイルにわたって生じる場合であり，2.1 節で述べたように，フレームワークが提供する複数のファイルの関係について理解の浅い初学者にとっては，エラーメッセージで指摘された箇所に関係する誤り箇所を特定することは困難である．

具体的には，次節で Rails の開発手順を示しながら，誤りパターンとそのデバッグ難易度について述べる．なお，1 章で定義した FOEs のデバッグ難易度はレベル 1 もしくはレベル 2 であり，HIEs はレベル 2 に該当する．

2.3　Rails 開発手順と初学者の誤りパターンとそのデバッグ難易度

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）を示し，それぞれのデバッグ難易度を**表 2** に示す．

(i) まず，Rails 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．ここで操作を誤ると実行時にエラーが発生する（EP1）．

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

図 3　コントローラ記述例
Fig. 3　An example of controller program.

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

図 4　ビュー記述例（index.html.erb）
Fig. 4　An example of view program.

(ii) 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持つ 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）．

(iii) 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）．

(iv) 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 3**）．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 4**）．コントローラとビューの連携は 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），ビューで値を受け取れない（EP6）
- コントローラとビューに記述したインタンス変数名が一致しない（EP7）
- ビューファイルでの Ruby の文法誤り（EP8）

(v) 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 5）．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）

筆者の指導経験をもとに誤りパターンごとのデバッグ難易度を表 2 に示す．HIEs のデバッグ難易度はレベル 2 であり，レベル 2 には多くの誤りパターンがあることから，HIEs につながる誤りはほぼすべての開発手順において発生する可能性があることが分かる．

2.4　ログファイルとログファイルから得られるエラー情報

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 6 にログファイルの例を示す．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．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が出力されている．この例外では，エラー発生箇所のソースコード片が示されているため，ビューファイルで img とすべきところを img0 とタイプミスしたことが例外の発生原因であることが分かる．

ここで注意が必要なのは，ログファイルに記載された例外クラス名が同じでも，発生原因となる誤りパターンは様々な場合があるということである．誤りパターンによってはソースコード片が示されない場合もあり，ソースコード片が示されている場合でも，他のファイルで定義した情報に影響を受けている場合，つまり，2.2 節で定義したデバッグ難易度がレベル 2 の場合は，ログファイルの情報だけでは発生原因を特定できない．

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

図 5　ルーティング記述例（routes.rb）
Fig. 5　An example of routing description.

```
Started GET "/images/index" for ::1 at 2020-06-29 16:09:00 +0900
（省略）
ActionView::Template::Error (undefined local variable or method `img0'
Did you mean?  img):
    1: <% @images.each do |img| %>
    2:   <p><%= img0 %></p>
    3: <% end %>
```

図 6　ログファイル（development.log）の例
Fig. 6　An example of development.log.

3.　提案ツール

3.1　ログファイルを用いたつまずき検出方法

本稿では，個々の受講者のつまずき状況には立ち入らず，多くの受講者がつまずいた要因を特定することを優先する．多くの受講者がつまずいたということは，講義や課題や開発環境など受講者に共通する部分に問題がある可能性があり，早急に原因を特定して改善する必要があると考えたからである．

具体的なつまずき検出の方法としては，まず，ログファイル中の例外の個数をエラー発生回数とし，その回数分，受講者がつまずいたと考える．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受講者率を求める．本稿ではこれを**つまずいた受講者率**と呼ぶ．

3.2　Git リポジトリに自動コミットするツール

つまずき要因の発生原因を特定するため，つまずきが発生した時点のソースコードの変更情報を Git リポジトリに自動的にコミットするツールを示す（図 7）．

ログファイルである development.log を tail コマンドで監視し，ログに追加されたメッセージに Error という文字列が含まれているか grep コマンドで検査し，含まれていれば，つまずき要因である例外クラス名をコミットメッセージとして自動的にコミットを実行する．

本ツールを受講者に配布し，各自のパソコンで起動してから課題に着手するように指示する．本ツールを演習開始時に起動することで，受講者はソースコードの保存操作を

```
 1 #!/bin/sh
 2 auto_commit() {
 3   while read i
 4   do
 5     echo $i | grep --line-buffered Error
 6     if [ $? = "0" ];then
 7       git add -A
 8       git commit -m "$i"
 9     fi
10   done
11 }
12
13 touch development.log
14
15 tail -n 0 -F development.log | auto_commit
```

図 7　つまずきを検知して自動コミットするツール
Fig. 7　A shell script tool that detects student mistakes and automatically commits the source code at that time.

意識することなく演習を進めることができる．

3.3　ログファイルと Git リポジトリを組み合わせてエラー情報を可視化するツール（FOEs Viewer）

Rails のディレクトリにはログファイルに加えて，3.2 節のツールで自動保存した Git リポジトリの情報も含まれている．これら 2 つの情報を組み合わせてエラー情報を可視化するツール（FOEs Viewer）を提案する．

FOEs Viewer はソースコード間をハイパーリングで関連付けた HTML ファイル群からなる．以下，その HTML ファイルを生成する前処理のアルゴリズムについて述べる．なお，受講者の提出ファイルの整理には人手が必要であるが，以下に示す前処理はスクリプトで自動実行でき，提出ファイルに対して 1 度実行すればよい．実行時間はマシン性能と受講者数によって変化するが，50 名分の提出ファイルであれば 5 分程度で処理が完了する．

手順 1　受講者の提出物である Rails プロジェクトの圧縮ファイルを展開し，その中で以下を実行する．

手順 2　ログファイルである development.log の中を検索して Error が含まれる行を取得し，それぞれの行の直前に出現する Started で始まる行を取得する．この行の末尾には日時があるため，これをエラー発生日時としキー情報としてエラーメッセージを対応づける．たとえば，{”2020/7/17 09:31:09” => ”ActionView::Template::Error(省略)”} のようにハッシュデータの形式で保存する．

手順 3　次に，git log コマンドを実行して取得したコミット情報からコミット ID とコミット日時を抽出する．前述のエラー発生日時と一致するコミット日時を持つコミット ID を前述のエラー情報に追加する．具体的には，{”2020/7/17 09:31:09” => { エラーメッセージ => ”ActionView::Template::Error(省略)”, コミット ID => ”74cff55”} のようなデータ構造となる．

手順 4　git log コマンドの実行結果から得られたコミット ID 群に対して以下を実行する．

手順 4a　コミット ID でチェックアウトする．

手順 4b　コミット ID 名のディレクトリを作成し（以降，コミットディレクトリ），チェックアウトしたファイルのうち，受講者が作成・修正したファイルをコミットディレクトリにコピーする．受講者が作成・修正したファイルリストの例を**図 8** に示す．これらファイル名は講義資料に示されており，Rails ではファイルの命名規則があるため，分析対象の講義ごとに事前にファイルリストを作成できる．

手順 4c　これまでに得られたエラー情報とコミットディレクトリをもとに，ログファイルと Git リポジトリ内のソースコードを紐付けるための HTML ファイルを生成する．コミット ID がコミットディレクトリ名であることから，コミット ID をファイルパス情報として，エラーメッセージに関連するファイルにアクセスするためのハイパーリンクを設定する．

```
1  Gemfile
2  config/routes.rb
3  config/application.rb
4  app/controllers/images_controller.rb
5  app/views/images/index.html.erb
6  app/views/images/edit.html.erb
7  app/views/images/new.html.erb
8  app/models/image.rb
9  log/development.log
```

図 8　受講者が作成・修正するファイルリスト
Fig. 8　A list of files to be created or modified by the student.

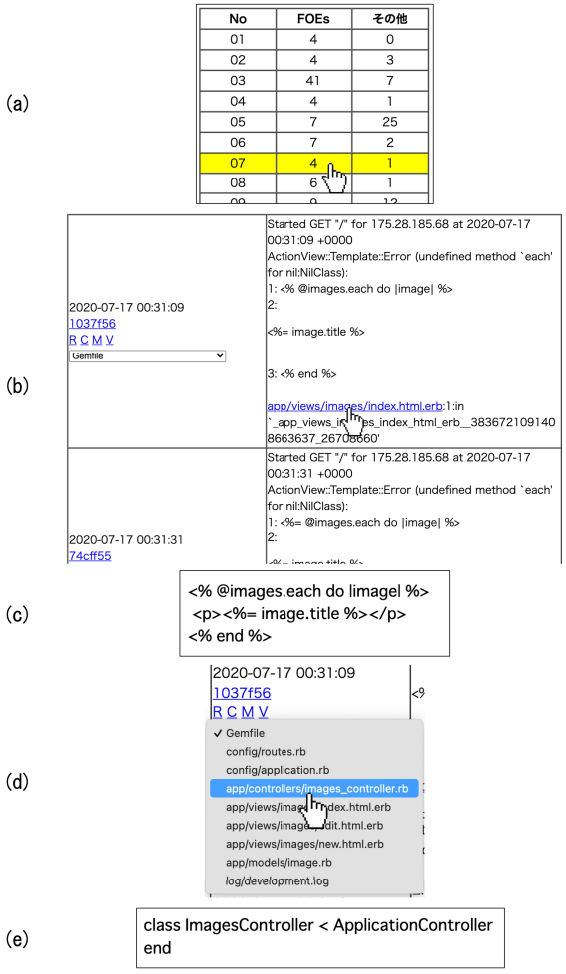

図 9　FOEs Viewer のユーザインタフェース
Fig. 9　UI of FOEs Viewer.

3.4　FOEs Viewer を用いた分析例

図 9 を用いて，FOEs Viewer による分析例について述べる．

(a)　FOEs Viewer のトップページである．No は受講者の整理番号で，FOEs は当該受講者が発生した FOEs の回数で，その他は FOEs 以外の例外の発生回数である．マウスで行を選択すると (b) に遷移する．ここで

は 7 番の受講者を選択する．

(b) 7 番の受講者が提出したログファイル中のエラー情報の一覧である．表の左列はエラー発生日時で，右列はそのエラーメッセージである．一番上のエラーメッセージを読むと，ActionView::Template::Error という例外が発生し，each メソッドを実行するインスタンス変数である@images が nil と分かる．@images が nil になった原因は表示されている情報だけでは分からないので，エラーが発生した index.html.erb のリンクをクリックする．

(c) index.html.erb を見ると，@images は参照のみであるためデータフローの上流であるコントローラを確認する必要があることが分かる．ブラウザの戻るボタンを押して (a) に戻り，(d) の選択メニューからコントローラである images_controller.rb を選択する．この選択肢は図 8 の情報をもとに提示する．

(e) images_controller.rb を見ると，ビューにわたすデータを@images にセットする index アクションが未作成であることが分かった．

このようにログファイルのエラー情報を起点として，エラー発生時のソースコードを巡回して分析を進めることができる．既知の誤りであれば，1 つあたり 30 秒くらいで発生原因を特定できる．FOEs Viewer を用いなかった場合は，ログファイルと Git リポジトリの情報を関連づけてファイルをたどる必要があるためより多くの時間が必要となる．

4. 実験

4.1 実験の目的

HIEs の発生原因を特定するために必要な情報を収集し可視化するツールを 3 章に示した．本ツールで収集する情報が HIEs の発生原因の特定に十分であるか，得られた情報を用いて HIEs の発生原因を特定できるか，また，講師などによる分析作業をどの程度支援できるかを実際の授業に適用して検証する．

4.2 実験方法

筆者の所属学科に 3 年生対象の Web アプリケーション開発の科目がある．本科目の授業内容を**表 3** に示す．本科目は講義と演習が対になった 2 コマの科目である．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 Rails 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持つ画像共有アプリケーションの作成方法を学習する．

2020 年度の本科目の第 11 回を実験対象とする．演習手順は 2.3 節に示したとおりである．3.2 節に示したツールを受講者に配布し，課題に取り掛かる前に実行するように指示する．完成した 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提出させる．課題の提出期間は 1 週間である．2020 年度の受講者 57 名のうち 33 名から提出があった．提出された課題はすべてアプリを完成しており，つまずいても何らかの方法で自己解決できたと見なせる．

表 3　Web アプリケーション開発科目のシラバス
Table 3　Syllabus for web application development.

講義回	授業内容
1	導入講義，Ruby の基礎（復習）
2	クラスモジュール
3	ファイルによる永続化
4	データベース基礎（復習）
5	データベースによる永続化
6	Web アプリケーション基礎
7	蔵書管理アプリの Web 化
8	Web アプリケーション開発
9	Ruby on Rails：ルーティングと MVC
10	Ruby on Rails：scaffold を使わないアプリ開発
11	Ruby on Rails：画像共有アプリ
12	Ruby on Rails：バリデーションとテスト
13	Ruby on Rails：総合演習
14	Ruby on Rails：総合演習
15	定期試験
16	解説

なお，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．プログラミング環境は，Amazon 社が提供する AWS Cloud9 という Linux ベースのクラウド開発環境を使用する．開発環境のバージョンは Ruby が 2.6.3 で，Rails は 5.2.4 で，データベースは Sqlite3 で 1.4.2 である．

4.3 講義および課題の内容

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手順を記入させた（**図 10**）．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．

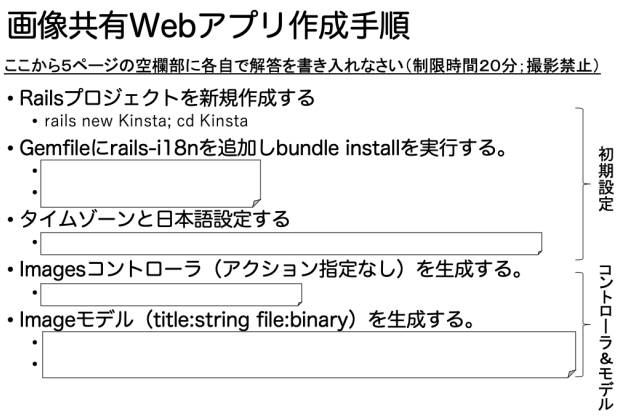

図 10　講義スライド例
Fig. 10　An example of lecture slide.

5. 実験結果

5.1 ログファイルから得られたつまずき状況

提出されたログファイルから抽出した例外は 7 種類であった．3.1 節に示した方法でつまずいた受講者率を求めた結果と例外の発生回数を**表 4** に示す．比較のため 2019 年度に調査した結果を併記する [12]．

全体的につまずいた受講者率は減少傾向にあるが，FOEs の 1 つである ActionController::RoutingError は 0.75 から 0.39 と半減している．講義スライドと演習課題は両年度でほぼ同じであるが，2020 年度はオンライン授業であったため，受講者は講義の解説を視聴しながら目前のパソコンを操作できた．そのため，手順誤りや記述箇所の間違いが減少したと推察される．

一方，もう 1 つの FOEs である ActionView::Template::Error は昨年と同様に約 8 割の受講者がつまずいたことが分かる．FOEs の発生回数は 129+196 で 325 であった．なお，2019 年度の発生回数は 200+171 で 371 であった．

5.2 Git リポジトリを用いた HIEs の発生回数

3.4 節の方法で FOEs の発生原因を分析した結果を**表 5** に示す．FOEs の列は，ActionView::Template::Error と ActionController::RoutingError の例外発生回数の合計である．

FOEs の発生回数は 325 であり，そのうちログファイルのみで発生原因が特定できたのは 106 であり，発生原因の特定に Git リポジトリの情報が必要な例外，つまり HIEs は 148+71 であった．そのうち，Git リポジトリの提出があった 148 の例外については，FOEs Viewer を用いて Git リポジトリのソースコードを参照することですべての発生原因を特定できた．Git リポジトリの提出がない受講者が 6 名いたため，その受講者が発生した 71 の HIEs の発生原因は分からなかった．

情報源ごとに FOEs の内訳をみると，**ログファイル**では ActionView::Template::Error の発生回数が多く，**ログファイル＋Git（HIEs）**では ActionController::RoutingError の方が多い．ActionView::Template::Error は主にビューに関する例外であるため，ログファイル中のエラーメッセージで原因を特定できることが多いようである．一方，ActionController::RoutingError の発生原因は様々であるため，ログファイルのみでは特定が困難であることが分かる．したがって，**ログファイル＋Git（HIEs）**には，2.2 節で述べたレベル 2 のデバッグ難易度の誤りが多く含まれており，エラー発生箇所だけではなく関連する情報を参照する必要がある．

表 4　ログファイルから得られた例外ごとのつまずき状況
Table 4　Student errors for each exception obtained from log files.

例外クラス名	つまずいた受講者率		発生回数
	2020 年度	2019 年度	
ActionView::Template::Error	0.85	0.79	129
ActionController::RoutingError	0.39	0.75	196
SyntaxError	0.39	0.67	52
NoMethodError	0.45	0.53	54
NameError	0.39	0.40	26
ActiveRecord::PendingMigrationError	0.00	0.13	0
ActiveModel::UnknownAttributeError	0.03	0.03	4
Encoding::UndefinedConversionError	0.00	0.03	0
Gem::LoadError	0.03	0.03	2

表 5　原因特定に必要な情報源ごとの FOEs の例外発生回数
Table 5　Number of exceptions to the occurrence of FOEs for each source of information needed to determine the cause.

発生原因の特定に使用した情報源	FOEs（回）	FOEs（%）	Template Error（回）	Routing Error（回）
ログファイル	106	33%	76	30
ログファイル＋Git（HIEs）	148	46%	40	108
特定不能（HIEs）	71	22%	13	58
合計	325	100%	129	196

表 6　ActionView::Template::Error の誤りパターンの発生割合
Table 6　Percentage of error patterns related to ActionView::Template::Error.

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%

5.3 FOEs の発生原因となった誤りパターンの発生割合とデバッグ難易度

ActionView::Template::Error と ActionController::RoutingError の発生原因を 2.3 節の誤りパターンで類別し，それぞれの発生割合を**表 6** と**表 7** に示す．

ActionView::Template::Error の誤りパターンは，表 6 によるとビューの誤りである EP8 の発生割合が 66%と最

表 7　ActionController::RoutingError の誤りパターンの発生割合
Table 7　Percentage of error patterns related to ActionController::RoutingError.

誤りパターン	発生割合
ルーティング定義忘れ（EP9）	13%
ルーティング定義の記述誤り（EP10）	49%
A タグなどのパス名の記述誤り（EP11）	38%

も多く，続いてコントローラの誤りである EP5〜7 の発生割合が 34%（24%+4%+6%）であった．EP8 のデバッグ難易度は表 1 よりレベル 1 であるから，ActionView::Template::Error の 66%はエラー発生箇所を調べればよい．一方，EP5〜7 のデバッグ難易度はレベル 2 であるから，エラー発生箇所とは異なる箇所を調べる必要がある．

ActionController::RoutingError の誤りパターンは，表 7 によるとルーティングの誤りである EP9 と EP10 の発生割合が 62%（13%+49%）と多い．これらの誤りは例外クラス名と発生原因が一致しているためデバッグは容易のように見えるが，今回の実験の例外はビューで発生したため，エラーメッセージの指摘箇所を確認しても誤りは発見できない．EP11 は例外クラス名からルーティングに誤りがあると想像するが，実際にはビューに誤りがあるため誤りの発見は難しい．これら EP9〜11 のデバッグ難易度はいずれもレベル 2 であり，エラー発生箇所とは異なる箇所を調べる必要がある．

6. 考察

6.1 エラーの原因を特定するための十分な情報を収集できたか

本稿では，例外発生時のソースコードを分析することで例外の発生原因を特定できると考え，2 つのツールを提案した．実験前には以下の 2 つの懸念があった．

- ログファイルの監視による自動コミットが受講者の実行を漏れなく記録できるか
- 自動コミット前のソースコードの編集履歴は保存されないが，発生原因の特定に影響はないか

2019 年度のログファイルを調査した結果から，tail コマンドで監視するキーワードを選定したが，筆者が Rails のエラーメッセージをすべて把握しているわけではないため，想定外のエラーメッセージが発生した場合には，ログファイルにエラーメッセージが記録されていてもソースコードを自動保存することができない可能性があった．また，ツールによる自動コミットは受講者が Rails アプリケーションを実行したときのみ実施されるため，コミット実行前の同一ソースコードに対する編集履歴は Git リポジトリには保存されず，エラーの原因を特定できない可能性があった．

5.2 節で述べたように，ログファイルのみでは特定が困難であった 148 の HIEs の発生原因をすべて特定できたことから，本稿の提案ツールで収集した情報はエラーの原因特定に十分であったといえる．

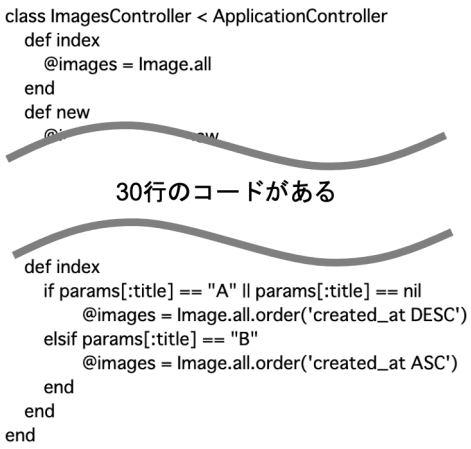

図 11　FOEs Viewer のみでは発生原因の特定が困難な例
Fig. 11　A case in which it was difficult to identify the cause of the error using FOEs Viewer alone.

なお，FOEs Viewer のみでは誤り原因が分からなかった例外が 2 つあった．以下に一例を示す．図 9 (b) と同じ例外を発生し，図 9 (c) と同様にビューに問題はなかった．コントローラを確認すると**図 11** 上部に示すように index アクションが定義されていたため目視では問題がないように見えた．Git リポジトリから例外発生時のソースコードをチェックアウトし，Rails アプリケーションを実行したところ，コントローラの最下部に index アクションを再定義されて例外が発生したことが分かった．今回の実験では，このような例は FOEs 全体の 0.6%（2/325）とわずかであるため，FOEs Viewer の有効性には影響はないものと考える．

6.2 HIEs の発生原因はどの程度特定できたか

実験結果から，ログファイルのみを用いた場合，発生した FOEs 全体の 33%しか発生原因が特定できなかったが，提案ツールを用いることで FOEs 全体の 79%（33%+46%）の発生原因を特定できた（表 5）．

これまでの研究では，ログファイルに記載された情報をもとに発生原因を分析し，情報が不足する場合には指導経験をもとに推察するにとどまった．提案ツールを使用することで，例外発生の直前のソースコードが Git リポジトリに保存されるため，発生した例外に関係するファイルを必要に応じて参照することで例外の原因が特定可能となった．

6.3 提案ツールの問題点

6.3.1 Git リポジトリに自動コミットするツールの問題点

Git リポジトリの提出がない受講者が 6 名おり，その受講者が発生した 71（全体の 22%）の FOEs の発生原因は特定できなかった．演習では Amazon 社の AWS Cloud9 とい

うクラウドベースの Linux 環境を使用していため，3.2 節に示したツールは Linux 版のみ用意した．実際には確認していないが，受講者によっては MS-Windows 上に Rails の開発環境を構築して課題に取り組んだ場合も考えられる．この場合はツールが使用できないため Git リポジトリの提出は不可能である．また，Git リポジトリの提出はあったもののログファイルに記載された例外と対応しない場合も「特定不能」とした．おそらく，ツールを少なくとも一度は実行したものの，何らかの理由で停止してコミットが実行されない状況があったと考える．

Rails アプリケーションの動作を確認するときには，`rails server` というコマンドを入力して開発用の Web サーバーを起動する．この操作時にツールが停止していれば起動する機能を追加することで，ツールの起動忘れの防止が期待できる．

6.3.2　FOEs Viewer の問題点

FOEs Viewer では，ログファイルから抽出したエラー情報のみが一覧表示され，それぞれのエラーに関連するソースコードを 1〜2 クリックで参照できる．分析者は受講者全体のエラーの発生状況を把握しつつ，詳しく調査したい受講者がいればソースコードをたどることで例外の発生原因を確認できる．一方，FOEs Viewer は，エラー情報を受講者ごとに表示するため，FOEs 全体の発生回数や誤りパターンを調べるのは困難である．受講者全体の傾向を分析するには，受講者ごととは別に，例外ごとにエラー情報を一覧表示するビューが必要であることが分かった．

また，FOEs Viewer では，分析者がエラーメッセージと関連するソースコードから誤りの発生原因を読み取る必要がある．3.4 節で述べたように，FOEs の分析には 1 つあたり 30 秒程度の時間がかかるため，今回の実験で発生した 325 件の例外を調査するには 1 時間半くらいの時間が必要である．また，プログラミング環境に対する分析者の熟練度が低い場合は，6.1 節に示したように，アプリケーションを実行して発生原因を確認する必要がある．これらの問題を軽減するため，本稿の実験で得られた知見をもとに分析作業の自動化を目指したい．

6.4　提案ツールの動作環境

ツールを動作させるためには Git リポジトリの初期化が必要であるが，Rails は標準で Git に対応しているため，Rails プロジェクトを新規作成するときに Git リポジトリも新規作成される．そのため，操作を忘れて記録漏れが発生することはない．動作環境としては，ローカル環境はもちろん，我々が演習で利用している AWS Cloud9 のようなクラウド環境や Docker を利用した仮想環境でも利用できる．

また，実用的な開発用フレームワークであれば，Rails の development.log のようなログ機能を具備しているため，本稿で提案したツールを用いて同様の分析が可能である．本稿で提案したスクリプトはきわめて単純な方法ではあるが，ソースコードなどテキストファイルの変更履歴を自動的に蓄積し，分析する手段の 1 つとして，様々な場面で利用できると考えている．

6.5　受講者ごとの誤りパターンに関する分析

3.1 節に，「本稿では，個々の受講者のつまずき状況には立ち入らず，多くの受講者がつまずいた要因を特定することを優先する」とした．しかし，個々の受講者のプログラミングスキルや Rails 開発に対する習熟度はばらつきがあるため，同じ誤りパターンであっても受講者によってデバッグ手順は異なるはずである．フレームワークに関する理解が浅いため同じ誤りパターンを繰り返す受講者もいれば，エラー情報を手がかりとして仮説を立てて着実に検証を進める受講者もいる．つまり，本稿では無視した受講者ごとの誤りパターンの発生回数や誤りパターンの並び順には，受講者のつまずきを支援するための情報が豊富に含まれていることが予見される．今後，誤りパターンの全体の発生割合と合わせて，受講者ごとのつまずき要因を明らかにしていきたい．

7.　関連研究

本稿で提案するツールは Rails に標準で具備されているログファイルと Git リポジトリの情報を組み合わせて実現するきわめてシンプルな構成である．このような構成を選択した理由としては，受講者のツール導入の手間をできるだけ少なくすることと，受講者が学習するときに本来は不必要な手順をできるだけ課さないこと，であった．

一方，これまでプログラミング学習状況を分析したり支援するシステムが多数提案されている．分析の目的や分析に必要な情報によって実現形態に違いが現れる．本章では，既存研究を受講者のプログラミング環境とソースコードなどの学習情報の保存場所で分類し，提案ツールとの関係について述べる（**表 8**）．なお，本稿の提案ツールは Type1 である．

7.1　Type1（汎用環境&クライアント保存）

本方式は，汎用的な開発環境をそのまま利用するため，学習に使用するプログラミング言語やフレームワークやラ

表 8　関連研究の類別表
Table 8　Types for classifying related research.

	プログラミング環境	学習情報の保存場所
Type1	汎用	クライアント
Type2	汎用	サーバ
Type3	専用	クライアント
Type4	専用	サーバ

イブラリの選択に制約が少ない．一方，受講者の学習情報を分析するためには，クライアントから情報を収集する必要があり，演習中などにリアルタイムで情報収集することは困難である．

本方式の事例として，藤原らは，Meadow と Cygwin といった汎用的なツールを受講者に使用させ，コンパイル回数や実行回数などの学習情報を 1 分ごとに受講者の環境に保存し，その情報を収集することで学生らの取り組み状況を推定するシステムを提案している [13]．槇原らは，クラウド上に受講者の人数分の仮想マシンを準備し，受講者は汎用的なエディタである Eclipse で開発を進め，ビルドツール（Apache Ant）の実行時刻，実行結果，エラーメッセージなど実行ログを仮想マシン内に蓄積するシステムを提案している．蓄積したログはローカルに保存されるが，Dropbox で同期した情報にクラウド経由で収集できるため Type2 でもある [14]．

7.2　Type2（汎用環境&サーバ保存）

本方式は，Type1 と同様に汎用的なソフトウェアを使用しているため Type1 と同様のメリットがある．また，受講者の学習情報をサーバに集約して学習状況の分析や支援に活用できる利点がある．

本方式の事例として，玉田らは，エラーメッセージとソースコードをサーバに送信する処理をコンパイラに含めたラッパーコマンドを開発し，そのコマンドを導入した OS イメージをネットブートサーバで公開するシステムを提案している [15]．受講者はそのサーバにアクセスするだけで特別なインストール作業は必要ない．また，受講者の操作は汎用的なコンパイラ（gcc や javac）と同じであるため，学習環境に特化した操作に習熟する必要はない．

7.3　Type3（専用環境&クライアント保存）

本方式は，専用の開発環境を使用する．専用であるため，開発環境の構築や特別な操作が必要となるが，汎用環境と比べて，より詳細な分析や効果的な支援を受けられるメリットがある．

榊原らは，コンパイルエラーの修正方法を短期間で身につけることを目指し，受講者自身による内省を促す機能を付加した GeneRef という専用のエディタを提案している [16]．コンパイルエラーの修正時間が閾値より長い場合に内省の入力を求める画面が表示される．論文では，コンパイルエラーの種別と修正時間を取得し分析しているが，それら学習情報をサーバに集約する記述が読み取れなかったため，クライアントに保存されるものと判断し Type3 とした．

7.4　Type4（専用環境&サーバ保存）

本方式は，Web ブラウザがクライアントであるため開発環境の導入や構築が不要であり，Web ブラウザ上で動作する専用アプリケーションにより，プログラム作成・コンパイル・テスト・提出の一連の流れをサポートする．一方，Rails 開発のように様々なフレームワークやライブラリを利用した演習を行う場合には，利用するフレームワークのバージョンの変更や使用するライブラリの追加に合わせてシステム開発や検証が必要となり，様々なバリエーションの学習環境を提供する負担が生じる．

本方式の事例は多数ある [17], [18], [19], [20], [21], [22]．Web ブラウザ上で動作する専用のソフトウェアで，キーストローク，ソースコード，エラーメッセージ，実行結果といった受講者の詳細な学習情報をリアルタイムに収集することで，様々な分析や支援が可能である．

8.　まとめ

本稿では，バージョン管理ソフトウェアの 1 つである Git を用いて例外発生時のソースコードを自動的に保存するツール，および，Git リポジトリの情報とログファイルを組み合わせてエラー情報を可視化するツール（FOEs Viewer）を提案した．本ツールを 2020 年度の授業に適用したところ，Git リポジトリの提出があった HIEs の発生原因をすべて特定できたことからツールで収集する情報の十分性が確認でき，FOEs Viewer を用いることで効率的に分析作業を進めることができた．

これまでの研究で，講義で扱っている範囲ではあるが，初学者が Rails 開発を学習するときのつまずき要因とその発生原因を明らかにできた．今後，提案ツールを [6], [7], [8], [9] などの一般的な初学者向けの学習手順に適用して有効性を評価する．我々は初学者が上達するためにはつまずきは必要な経験と考えている．現状の演習ではつまずいたときに講師や TA が支援するにとどまり，デバッグ技術を積極的に指導するまでには至っていない．今後，これまで得られたつまずき要因をもとにデバッグ手順を開発し，講義や演習でその効果を評価していきたい．

参考文献

[1] Chunling, C.: Construction of the Individualized College English Learning Management System Using Ruby on Rails, *International Conference on Service Science*, pp.160–63 (2015).
[2] An, J.-H., Chaudhuri, A. and Foster, J.S.: Static typing for Ruby on Rails, *IEEE/ACM International Conference on Automated Software Engineering*, pp.590–594 (2009).
[3] Crawford, T. and Hussain, T.: A comparison of server side scripting technologies, *Proc. International Conference on Software Engineering Research and Practice*, pp.69–76 (2017).
[4] 井上　明，金田重郎：実システム開発を通じた社会連携型 PBL の提案と評価，情報処理学会論文誌，Vol.49, No.2, pp.930–943 (2008).
[5] Hiroyuki, C., Chubachi, Y. and Nakamura, M.: Co-

education of master course students and business people, *Proc. 6th International Conference on Information and Education Technology*, pp.89–96 (2018).
[6] Hartl, M.: Rails チュートリアル，YassLab 株式会社（オンライン），入手先 ⟨https://railstutorial.jp⟩（参照 2021-01-25）.
[7] Progate: Ruby on Rails5, 株式会社 Progate (online), available from ⟨https://prog-8.com/languages/rails5⟩ (accessed 2021-01-25).
[8] 田口　元：Ruby on Rails5 入門，ドットインストール（オンライン），入手先 ⟨https://dotinstall.com/lessons/basic_rails_v3⟩（参照 2021-01-25）.
[9] Udemy: Udemy, Udemy (online), available from ⟨https://www.udemy.com⟩ (accessed 2021-01-25).
[10] 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).
[11] 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).
[12] 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（FOSE2020），pp.103–108 (2020).
[13] 藤原理也，田口　浩，島田幸廣，高田秀志，島川博光：ストリームデータによる学習者のプログラミング状況把握，電子情報通信学会第 18 回データ工学ワークショップ，pp.1–6 (2007).
[14] 槇原絵里奈，井垣　宏，吉田則裕，藤原賢二，川島尚己，飯田　元：ソフトウェア開発 PBL におけるビルドエラーの調査，情報処理学会論文誌，Vol.58, No.4, pp.871–884 (2017).
[15] Tamada, H., Ogino, A. and Ueda, H.: A Framework for Programming Process Measurement and Compiling Error Interpretation for Novice Programmers, *2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pp.233–238 (2011).
[16] 榊原康友，松澤芳昭，酒井三四郎：コンパイルエラー修正時間に着目した学習分析指標の提案と内省学習効果分析への適用，研究報告コンピュータと教育（CE），Vol.2013-CE-118, No.8, pp.1–6 (2013).
[17] 市村　哲，梶並知記，平野洋行：プログラミング演習授業における学習状況把握支援の試み，情報処理学会論文誌，Vol.54, No.12, pp.2518–2527 (2013).
[18] 井垣　宏，齊藤　俊，井上亮文，中村亮太，楠本真二：プログラミング演習における進捗状況把握のためのコーディング過程可視化システム C3PV の提案，情報処理学会論文誌，Vol.54, No.1, pp.330–339 (2013).
[19] 長島和平，長　慎也，間辺広樹，兼宗　進，並木美太郎：Web ブラウザを用いたプログラミング学習支援環境 Bit Arrow の設計と評価，研究報告コンピュータと教育（CE），Vol.2017-CE-138, pp.1–8 (2017).
[20] Warner, J. and Guo, P.J.: CodePilot: Scaffolding End-to-End Collaborative Software Development for Novice Programmers, *Proc. 2017 CHI Conference on Human Factors in Computing Systems*, pp.1136–1141 (2017).
[21] 布施　泉，中原敬広，岡部成玄：プログラムの相互利用と相互評価が可能な初学者用プログラミング授業支援環境の構築，教育システム情報学会誌，Vol.35, No.2, pp.221–226 (2018).
[22] Kohn, T.: The Error Behind The Message: Finding the Cause of Error Messages in Python, *Proc. 50th ACM Technical Symposium on Computer Science Education*, pp.524–530 (2019).

高橋 圭一

1992 年室蘭工業大学第二部電気工学科卒業．1994 年同大学大学院工学研究科情報工学専攻修了．同年ニッテツ北海道制御システム（株）入社．2002 年同大学大学院工学研究科博士後期課程生産情報システム工学専攻修了．2004 年近畿大学産業理工学情報学科講師．2014 年同大准教授．博士（工学）．製造ラインシミュレータおよびソフトウェア開発支援の研究・開発に従事．プロジェクトマネジメント学会，ACM 各会員．