まえがき

プログラム共同委員長　　角田 雅照 [*]　松本 真佑 [†]

本書は, 日本ソフトウェア科学会「ソフトウェア工学の基礎」研究会 (FOSE: FOundation of Software Engineering) が主催する第 29 回ワークショップ (FOSE2022) の論文集です．ソフトウェア工学の基礎ワークショップは，ソフトウェア工学の基礎技術を確立することを目指し，研究者・技術者の議論の場を提供します．大きな特色は異なる組織に属する研究者・技術者が，3 日間にわたって寝食を共にしながら自由闊達な意見交換と討論を行う点にあります．第 1 回の FOSE は，1994 年に信州穂高で開催し，それ以降，日本の各地を巡りながら，毎年秋から初冬にかけて実施しており，今回で 29 回となります．本年は，島根県松江しんじ湖温泉での開催になります．COVID-19 の影響により，2021 年に引き続き，論文集編集時点ではオンラインと現地とのハイブリッドで開催することを予定しています．松江しんじ湖温泉は 1971 年に開湯された比較的新しい温泉で，宍道湖畔に位置します．温泉でリラックスをし，加えて秋の味覚に舌鼓を打ちつつ，活発な議論が行われることを期待します．

本年もこれまでと同様に，以下の 3 つのカテゴリで論文および発表を募集しました．

1. 通常論文ではフルペーパー (10 ページ以内) とショートペーパー (6 ページ以内) の 2 種類を募集しました．投稿は，フルペーパーに 19 編，ショートペーパーに 7 編あり，それぞれ 3 名のプログラム委員による並列査読，および，プログラム委員会での厳正な審議を行いました．その結果，フルペーパーとして 7 編，ショートペーパーとして 18 編の論文を本論文集に掲載しました．
2. ライブ論文 (2 ページ以内の速報的な内容) には，26 編の応募があり，全てを採録として，本論文集に掲載しました．ワークショップでは，論文内容についてポスター発表が行われます．
3. ポスター・デモ発表として，本論文集に掲載されない形でのポスター発表やデモンストレーションを受け付けました．26 件の応募があり，全て発表いただくことを決定しました．

なお，日本ソフトウェア科学会の学会誌「コンピュータソフトウェア」において，本ワークショップと連携した特集号が企画されています．ワークショップでの議論を経てより洗練された論文が数多く投稿されることを期待します．

基調講演では，広島大学の劉 少英 教授により「Software Fault Prevention and Verification for Human-Machine Pair Programming」のタイトルでご講演を予定しています．Human-Machine Pair Programming(HMPP）と呼ばれるソフトウェア開発方法についてご紹介いただきます．この開発方法では，開発者がシステムを構築しつつ，コンピュータがシステムの欠陥予防や妥当性の検証を行います．

最後に，本ワークショップのシニアプログラム委員の皆様，プログラム委員の皆様，出版委員長の近藤 将成氏，ソフトウェア工学の基礎研究会主査の沢田篤史氏，レクチャーノート編集委員の武市正人氏，米澤明憲氏，近代科学社編集部および関係諸氏に感謝いたします．

[*]Masateru Tsunoda, 近畿大学

[†]Shinsuke Matsumoto, 大阪大学

プログラム委員会

共同委員長

角田 雅照　（近畿大学）
柗本 真佑　（大阪大学）

出版委員長

近藤 将成　（九州大学）

プログラム委員

天嵜 聡介　(岡山県立大学)
阿萬 裕久　(愛媛大学)
石尾 隆　(奈良先端科学技術大学院大学)
石川 冬樹　(国立情報学研究所)
市井 誠　(日立製作所)
伊藤 恵　(はこだて未来大学)
伊原 彰紀　(和歌山大学)
上田 賀一　(茨城大学)
鵜林 尚靖　(九州大学)
上野 秀剛　(奈良高専)
大平 雅雄　(和歌山大学)
大森 隆行　(立命館大学)
小笠原 秀人　(千葉工業大学)
小形 真平　(信州大学)
柏 祐太郎　(奈良先端科学技術大学院大学)
鹿糠 秀行　(日立製作所)
神谷 年洋　(島根大学)
岸 知二　(早稲田大学)
切貫 弘之　(NTT)
桑原 寛明　(南山大学)
近藤 将成　(九州大学)
関澤 俊弦　(日本大学)
田原 康之　(電気通信大学)
丹野 治門　(NTT)
崔 恩瀞　(京都工芸繊維大学)
戸田 航史　(福岡工業大学)
中川 博之　(大阪大学)
名倉 正剛　(南山大学)
野田 訓広　(富士通)
萩原 茂樹　(千歳科学技術大学)
蜂巣 吉成　(南山大学)
花川 典子　(阪南大学)
林 晋平　(東京工業大学)
福田 浩章　(芝浦工業大学)
福安 直樹　(大阪工業大学)
本田 澄　(大阪工業大学)
槇原 絵里奈　(同志社大学)
森崎 修司　(名古屋大学)
安田 和矢　(日立製作所)
吉田 則裕　(立命館大学)

シニアプログラム委員

鯵坂 恒夫　(武庫川女子大学)
大西 淳　(立命館大学)
権藤 克彦　(東京工業大学)
杉山 安洋　(日本大学)
本位田 真一　(早稲田大学)
門田 暁人　(岡山大学)
沢田 篤史　(南山大学)
岡野 浩三　(信州大学)
小林 隆志　(東京工業大学)
佐伯 元司　(南山大学)
高田 眞吾　(慶應義塾大学)
立石 孝彰　(日本 IBM)
中島 震　(国立情報学研究所)
野呂 昌満　(南山大学)
山本 晋一郎　(愛知県立大学)
吉岡 信和　(早稲田大学)
吉田 敦　(南山大学)

目次

基調講演

Software Fault Prevention and Verification for Human-Machine Pair Programming
劉 少英（広島大学）.....1

ユーザインタフェース

Placeona に基づいたモバイルデバイス向け適応型ユーザインタフェースの構築支援
池上 潤（早稲田大学），白銀 純子（東京女子大学），
岩田 一（神奈川工科大学），深澤 良彰（早稲田大学）.....3

静的解析と動的解析の組み合わせによる UML ステートマシン図答案の誤り特定自動化手法の提案
五島 光祥，小形 真平（信州大学），槇原 絵里奈（同志社大学），
岡野 浩三（信州大学）.....13

プログラム理解パターン抽出のための構文木と視線移動の自動マッピング手法
吉岡 春彦，上野 秀剛（奈良工業高等専門学校）.....23

教育

オンラインジャッジシステムの問題選択支援に向けた問題文間の意味的類似度の算出
新濱 遼大，槇原 絵里奈，小野 景子，大正 歩夢（同志社大学）.32

Scratch を用いたプログラミング演習における教員支援を目的とした採点支援システムの提案
若松 玲依，槇原 絵里奈，小野 景子，新濱 遼大（同志社大学）.38

プログラミング演習におけるセグメントを用いたソースコードの誤り箇所特定方法の提案
澤田 侑希，梅田 祐一郎，蜂巣 吉成，吉田 敦，
桑原 寛明（南山大学）.....44

VR 環境における文字入力支援システム
清原 隆一，沢田 篤史，野呂 昌満（南山大学）.....50

形式手法

Rust MIR に対する情報流解析の型システムの検討
桑原 寛明（南山大学）.....56

形式手法を用いたPID制御装置の検証
浦岡 竜太郎，伊藤 宗平（長崎大学）.....62
検証パターンに注目した機械学習に基づくモデル検査手法の評価
張 超群，岸 知二（早稲田大学）.....68

ライブラリ

プログラムのベクトル化と記号実行を活用した正誤判定の効率化
大嶋 琉太，阿萬 裕久，川原 稔（愛媛大学）.....74
Pythonにおける機械学習関連ライブラリの自動推薦手法の評価
小柳 慶，秋山 楽登，沖野 健太郎，近藤 将成，亀井 靖高，
鵜林 尚靖（九州大学）.....80
ソフトウェア部品の利用関係に基づくクラスタリングの進化分析
横森 励士，安藤 勇人（南山大学），吉田 則裕（立命館大学），
野呂 昌満，井上 克郎（南山大学）.....86

機械学習

DVCリポジトリにおける機械学習パイプラインの進化に関する調査
中村 悠人，松田 雄河，松尾 春紀，近藤 将成，亀井 靖高，
鵜林 尚靖（九州大学）.....92
定形的ログメッセージの除去とクラスタリングによる異常動作ログの
検出方法の提案
上田 晃義，尾花 将輝（大阪工業大学），花川 典子（阪南大学）102

要求・設計

要求獲得における質疑応答履歴のグラフデータベースシステムの実現
今堀 由唯（南山大学），加藤 潤三（独立コンサルタント），
林 晋平（東京工業大学），大西 淳（立命館大学），
佐伯 元司（南山大学）.....112
チェックリストを用いた設計書レビュー支援のための判定ルール自動生成
大林 浩気，河合 克己，前岡 淳（日立製作所）.....118
エポックワードと名詞の重要度を用いたソフトウェア仕様書からの
ゴール文の抽出
渡辺 啓太郎，中川 博之，土屋 達弘（大阪大学）.....124
アヤトゥス・カルタの拡張によるユーザ視点に基づくMVP抽出手法の提案
田中 貴子（NTTテクノクロス），斎藤 忍 （NTT）.....130

見積・予測

欠損確率に基づいた欠損データ作成手法の提案とソフトウェア開発データにおける評価
上佐 公太郎，柿元 健（香川高等専門学校）. 136

Fault-prone モジュール予測における第三者データに基づいた外れ値除去
西浦 生成，門田 暁人（岡山大学）. 142

1 事例を通じてのストーリーポイントの有用性と見積もり誤差に対する考察
今井 健男（Idein）. 148

キーストロークとマウス操作に基づくプログラミング能力の分析
松本 和樹，西浦 生成，笹倉 万里子，門田 暁人（岡山大学）. . 154

ソースコード

コードクローン検出に基づく IoT を対象とした自動パッチ生成
大野 堅太郎（名古屋大学），吉田 則裕（立命館大学），朱 文青，
高田 広章（名古屋大学）. 160

コード難読化ツールの信頼性を評価するフレームワークの検討
北岡 哲哉（奈良先端科学技術大学院大学），
神崎 雄一郎（熊本高等専門学校），石尾 隆，嶋利 一真，
松本 健一（奈良先端科学技術大学院大学）. 170

ライブ論文

群ロボット制御用ソフトウェアアーキテクチャの提案
箕輪 知也，中谷 多哉子（放送大学），滝本 宗宏（東京理科大学），神
林 靖（日本工業大学）. 180

Web アプリケーション開発における欠陥再現の自動化ツールの提案
高橋 黎（日本工業大学），櫨山 淳雄（東京学芸大学），
橋浦 弘明（日本工業大学）. 182

メソッド名の整合性評価のためのデータセット
峯久 朋也，阿萬 裕久，川原 稔（愛媛大学）. 184

非エンジニア向けの Programming by Examples によるデータ加工支援の試み
倉林 利行，丹野 治門（NTT）. 186

README における項目と説明文の一貫性の分析
石岡 直樹，伊原 彰紀（和歌山大学）. 188

ゲーミフィケーションを用いた C 言語の文法やアルゴリズムの
学習支援アプリケーション Code Quiz の提案
谷本 嵩晃，崔 恩瀞，水野 修（京都工芸繊維大）. 190

GitHub における模範プロジェクトの検出とその成長パターンの分類に向けて

開出 凱斗，玉田 春昭（京都産業大学），戸田 航史（福岡工業大学），
中村 匡秀（神戸大学）. 192

育成の観点を取り入れたプロジェクト管理ゲーム
山形 宥太，西浦 生成，笹倉 万里子，門田 暁人（岡山大学）. . 194

ロジックモデルからステークホルダーバリューネットワークへの変換による価値循環の抽出
丹羽 南，山田 勉，青木 善貴（BIPROGY）. 196

2 つの Web アプリケーション間の類似する操作対象の対応関係抽出
内田 啓太，石尾 隆，嶋利 一真，
松本 健一（奈良先端科学技術大学院大学）. 198

Grad-CAM を用いた画像認識 AI の特徴分析の試み
西村 滋幸，本田 澄（大阪工業大学），山下 育男（関西電力）. . 200

システムテストにおけるキーワード駆動テストの適用とキーワードの階層化設計
櫻井 壮希（日立製作所），塚本 夏基，内木 大地（日立ハイテク）202

ソフトウェア開発者の信頼度の評価に向けて
池田 海斗，西浦 生成，笹倉 万里子，門田 暁人（岡山大学）. . 204

Web GUI に対するオンラインジャッジシステムプロトタイプの実装
岡嶋 隆人（日本工業大学），田中 昂文（玉川大学），
櫨山 淳雄（東京学芸大学），橋浦 弘明（日本工業大学）. 206

技術的負債に関する課題票の分類モデルに単語分散表現が与える影響の分析
田口 舞奈，木村 祐太，大平 雅雄（和歌山大学）. 208

再利用性向上を目的とした数値解析ライブラリ構築パターンの提案
市村 純一，中谷 多哉子（放送大学）. 210

SCDV モデルを利用する技術用語に対応した自然言語文書検索の提案
辻 優太郎，神谷 年洋（島根大学）. 212

Maintainability Index を用いた保守性改善プロセス適用
加賀 洋渡（日立製作所）. 214

効率的なソフトウェアアップサイクルのための事例知識ベースの予備的評価
中田 匠哉，陳 思楠（神戸大学），佐伯 幸郎（高知工科大学），
中村 匡秀（神戸大学）. 216

提案依頼書におけるセキュリティ要件の実態調査
村越 竜介，西浦 生成，笹倉 万里子，門田 暁人（岡山大学）. . 218

自然言語解析を用いた開発プロジェクトリスク状況の定量化・可視化
巴 統哉，北川 健二，川上 真澄（日立製作所）. 220

プログラム理解難易度の経時的な変化の可視化

曾我 遼，鹿糠 秀行（日立製作所），久保 孝富，石尾 隆，
松本 健一（奈良先端科学技術大学院大学）. 222
ソースコード編集履歴再生器の履歴アノテーションによる拡張
大森 隆行（立命館大学）. 224
Dockerfile の依存関係とビルドエラーの関係分析
坂本 廉也（和歌山大学），東 裕之輔（日本総合研究所/和歌山大学），
大平 雅雄（和歌山大学）. 226
ゲーム要素を取り入れた情報リテラシー教育の提案
廣瀬 司，岡本 克也，中谷 多哉子（放送大学）. 228
スマホゲームにおける課金誘導方法の調査研究
麻生直希，西浦 生成，笹倉 万里子，門田 暁人（岡山大学）. . . 230

Software Fault Prevention and Verification for Human-Machine Pair Programming

劉 少英 *

Human-Machine Pair Programming (HMPP) is a new software development paradigm proposed by the speaker in 2018. It is characterized by the combination of human constructing algorithms while machine preventing faults during the construction and verifying the correctness after the construction. The most challenging problems in HMPP are how faults can be effectively and efficiently prevented during the programming process and how the constructed program be automatically verified. In this talk, after a brief discussion of the challenges for the well-studied and applied software development methods and how they can be tackled by the HMPP technology, I will concentrate on the presentation of several fault prevention approaches and the Testing-Based Formal Verification (TBFV) method. Finally, I will point out potential research topics for future research.

*Shaoying Liu, 広島大学

Placeonaに基づいたモバイルデバイス向け適応型ユーザインタフェースの構築支援

Support for building adaptive user interfaces for mobile devices based on Placeonas

池上 潤* 白銀 純子† 岩田 一‡ 深澤 良彰§

あらまし モバイルデバイスは普及とともに, 様々なコンテキストで利用されるが, 現在広く使用されている GUI(Graphical User Interface) が利用できないコンテキストも存在する. そのため, ユーザのコンテキストに対して, 複数の UI(User Interface) を使い分けることができれば, より様々な状況でデバイスを使用することが可能となる. また, 近年, 注目を集めている VUI(Voice User Interface) であるが, 使用時のユーザのコンテキストを考慮した設計に関しては, 未だ確立していない. そこで本研究では, VUI, GUI, シェイク動作をモダリティとし, ユーザのコンテキストに応じて切り替え, 使い分ける適応型ユーザインタフェースの提案と実装の支援をし, その有効性について検証する.

1 はじめに

今日, モバイルデバイスの普及とともに, インタフェースのマルチモーダル化が進んでいる [1][2]. 人間は, 目, 耳, 口, 鼻, 手などの複数のモーダルを持っており, 現在主流となっている視覚というひとつのモーダルから聴覚や嗅覚, 発話などの複数のモーダルを取り入れたインタフェースへの自然な展開が見られる [3]. モバイルデバイスが, 様々なコンテキストで利用されるようになってきている現在, アプリケーションの UI(User Interface) として広く用いられている GUI (Graphical User Interface) が利用できないコンテキストも存在している. そこで, ユーザのコンテキストに対して, 複数の UI を使い分けることができることができれば, モバイルデバイスをより様々なコンテキストにおいて使用することが可能となる. しかし, ユーザのコンテキストに応じて, 複数の UI を使い分けるようなマルチモーダルな適応型 UI の有効性に関する研究は未だ数少ない.

また, コンピュータとの対話手段の中で, 近年, 特に注目を集めているのは, ユーザが音声によってコンピュータを操作する UI である, VUI(Voice User Interface) である [4][5][6]. VUI には, 使用するユーザのコンテキストに依存するという特性があり, VUI が使用できるかは, ユーザを取り巻く環境とユーザ自身の行動や背景に制限される. この際, VUI の性質は GUI や CUI(Character User Interface) といった他の UI の性質とは異なるために, 既存の UI の設計の手法や基準 [7][8] をそのまま適用することができない. しかし, デザイナーが如何にしてユーザのコンテキストを考慮した VUI 設計を行うのかに関しての基準は設けられていない.

ユーザのコンテキストに関して, Placeona という概念が提唱されている [9]. Placeona とは, システムと対話するユーザの目・手・耳・口の使用できる程度を明らかにしたものである. そこで本研究では, Placeona に基づいたユーザのコンテキストに応じて VUI, GUI, デバイスをシェイクすることで操作の一部を実現するシェイク動作の UI を適切に切り替える適応型 UI を提案する.

Placeona の考え方に基づき, 提案 UI(PlaceonaUI と呼ぶ) を適用するコンテキス

*Jun Ikegami, 早稲田大学

†Junko Shirogane, 東京女子大学

‡Hajime Iwata, 神奈川工科大学

§Yoshiaki Fukazawa, 早稲田大学

トに関して定義した後, ユーザのコンテキストに対して, どの UI を適用するかのルールを定める. 次に PlaceonaUI をモバイルデバイスのアプリケーションの UI として使用する際のコンテキストの検知・更新, UI の切り替えの方針を定義し, アプリケーションの UI として組み込む流れを示す. 本研究では, PlaceonaUI の有効性を示すことで, 未だ確立していないユーザのコンテキストを考慮した VUI の設計基準の発展に貢献するとともに, ユーザがモバイルデバイスを用いて様々なタスクを行うことの利便性を高めるマルチモーダルなインタフェースの発展にも貢献する.

2 関連研究

未だ確立していない VUI の設計手法やガイドライン, 評価に関する研究がされている. Murad らは, 音声インタラクションに固有の新しい設計ガイドラインを開発する必要性に対し, GUI ガイドラインは有用な基盤となることから, 既存の文献にある VUI 固有のヒューリスティックと既存の GUI ガイドラインの整合性について分析する研究をしている [10]. 提案されたガイドラインやヒューリスティックにおいて, 音声による相互作用におけるコンテキストの影響に関して挙げられており, これらは, VUI の評価やユーザビリティの問題を特定するために役立つが, それを満たすための設計に関する具体的な解決には至っていない. 本研究では, ユーザのコンテキストを尊重するという項目に対し, 具体的にそれを満たすための VUI の設計におけるアプローチに関して提案する.

複数の入出力モダリティを利用してユーザとコンピュータとのコミュニケーションを強化するマルチモーダルインタフェースの設計や, その有効性を示す研究がされている. Neto らは, GUI との音声インタラクションの長所を検討し, Web での GUI と音声対話を組み合わせることを目的とした, マルチモーダルなインタフェースの設計アプローチに関する研究をしている [11]. この研究では, それぞれのインタラクションの形態の長所を生かすという点に着目したマルチモーダルインタフェースの設計に関して扱っているのに対して, 本研究では, ユーザのコンテキストに対して有効的なインタラクションを提案する. また, インタラクションのスタイルとして, GUI と VUI に加えシェイク動作による操作を使用することができる点で異なる.

システムがシステムまたはその周囲に関する情報を収集し, それに応じて動作を適応するコンテキストアウェアなシステムの提案や設計, その有効性を示すような研究がされている. Natarajasivan らは, ホーム画面のアプリケーションリンクのためのコンテキストアウェアなユーザインタフェースレコメンダーシステムに関する研究をしている [12]. ユーザから収集された, 曜日, ユーザのいる場所, 時間というコンテキストに関するデータから, リンクの配置を推奨するルールを生成するアルゴリズムを提案している. この研究では考慮されるコンテキストが限定的であり, 主にユーザの日常的な行動の傾向をコンテキストとして扱っているのに対し, 本研究では, ユーザがデバイスを操作する手・目・耳・口の使用できる程度に関するコンテキストを考慮し, よりリアルタイムなユーザやその周囲のコンテキストを考慮したシステムであるという点で異なる.

3 Placeona

Placeona[9] はユーザがシステムと対話する際に, ユーザの場所が対話のタイプをどのように制限するのかを示す. 人間は, デバイスを操作するために, 目, 手, 耳, 口を利用することができるが, Placeona はユーザのシナリオに対して, それらの状態を定義することで, ユーザがシステムとの対話において使用できるものを定義する. 例えば, 「ユーザが 1 人で運転中に目的地を変更し, そこまでの道のりの案内を要求する. 」というシナリオにおける Placeona は, 目は忙しい, 手は忙しい, 耳は自由, 口は自由 と定義することができる. この場合, ユーザは耳と口を使用してシステムと対話することが可能であり, 目と手は使用することができないため, 画面により操

作することは難しく, 音声による操作が適切である. このように Placeona を定義することは, ユーザがシステムと対話する際にどのようなインタラクションが適切であり, また対話において解決する必要のある問題や問題を予測することにつながる.

4 UI 利用時のコンテキスト

本研究では, Placeona に基づいた, ユーザのコンテキストに対して, GUI, VUI, シェイク UI の中で適した UI を適用する. まずはじめに, ユーザの目・手・耳・口の利用できる程度に影響を及ぼす具体的なコンテキストを表 1 に示す [13][14]. この際, ユーザのコンテキストの変化に応じて UI を適用するため, 基本的に変化が起こることのないユーザの背景に関するコンテキストについては考えない.

表 1 ユーザの状況に応じたコンテキスト

対話手段	具体的なコンテキスト
目	・目の状態 (開いているか閉じているか) ・目の忙しさ
手	・手の状態 (例: デバイスとユーザの間に距離がある, 手が忙しい, ユーザの手が汚れておりデバイスに触れることができない)
耳	・周囲の環境音 ・耳の状態 (耳を他の音に集中しているかどうか)
口	・口の状態 (例: 何かを食べている, 話している) ・ユーザがデバイスと対話する場所 (例: ユーザが静かでいることが求められる場所にいる)

5 UI の適用

5.1 UI 適用の条件

PlaceonaUI では, 入力部として, 画面入力, 音声入力, シェイク入力, 出力部として, 画面出力と音声出力を使い分けることとする. PlaceonaUI におけるシェイク入力とは, デバイスを振る動作と音声入力を組み合わせることにより, 音声入力のみと比較してより入力を柔軟に行うことができる入力 UI とする. あるユーザの状況において, ユーザに適切な UI を提供するためには, ユーザの状況に応じた UI の適用条件を特定する必要がある. 適用条件は, 目・手・耳・口がどの程度使用できるかに依存する. 本研究において, ある UI を使用することが可能である条件をその UI の前提条件と呼ぶこととする. また, 前提条件を満たしたうえで, 使用することができる UI がただ 1 つに決まる条件をその UI の決定条件と呼ぶこととする. 入力部の適用条件を表 2, 出力部の適用条件を表 3 にそれぞれ示す. ただし, 表では条件を以下のように表し, 条件の否定は, (条件)’と表す.

- 条件 A：目がデバイスを集中して見ることができる
- 条件 B：手で操作することができる
- 条件 C：手でデバイスを持つことができる
- 条件 D：口で話すことができる
- 条件 E：耳で聞くことができる

5.2 Placeona と UI の対応

本手法では, Placeona に基づき, ユーザがデバイスとの対話手段をどの程度使用することができるか分析し, それに応じて UI を適用する. そのため, ユーザの Placeona とそのとき適用する UI を対応づける必要がある. ここで, ユーザのデバイスとの対

表 2　入力部の適用条件

入力部	前提条件	決定条件	任意なコンテキスト
画面入力	条件 A かつ 条件 B	(条件 D) ’	耳
音声入力	条件 D	(条件 A) ’ かつ (条件 C) ’	耳
シェイク入力	条件 C	{(条件 A) ’ または (条件 B) ’ } かつ 条件 D	耳

表 3　出力部の適用条件

出力部	前提条件	決定条件	任意なコンテキスト
画面出力	(条件 A) ’	(条件 E) ’	手, 口に関するコンテキスト
音声出力	(条件 E) ’	(条件 A) ’	手, 口に関するコンテキスト

話手段の使用できる程度をその対話手段における自由度と定義する. 5.1 節で示した入出力部の適用条件に基づいて UI を対応づけるため, 各対話手段の自由度を表 4 のように定める.

表 4　対話手段の自由度

対話手段	自由度	使用できる程度
目	○	デバイスを集中して見ることができる
	×	デバイスを集中して見ることができない
手	○	手で操作することができる
	△	手で持つことができる
	×	手を使用することができない
耳	○	聞くことができる
	×	聞くことができない
口	○	話すことができる
	×	話すことができない

5.1 節で示した入出力部の適用条件において, 決定条件を満たしたとき, 使用可能な UI は 1 つに限定されることから, その UI を適用することとする. また, 複数の UI が前提条件を満たしている場合, 複数種類の UI を必要に応じてユーザ自身が使い分けることができるように設計する. PlaceonaUI では, 入出力部に関して, ユーザが自由に選択できる場合をハイブリッドとし, 入力方法は, ハイブリッド, 画面, 音声, シェイク (シェイク動作と組み合わせた音声) の 4 通り, 出力方法は, ハイブリッド, 画面, 音声の 3 通りを用意する. 5.1 節で示した入出力部の適用条件に基づき, 入出力部を対応づけた. 結果の一部を表 5 に示す. ただし, 全ての入力方法, 出力方法が使用できない場合, PlaceonaUI では, ハイブリッドを適用する.

表 5　PlaceonaUI における UI 適用のルール (一部)

目	手	耳	口	入力部	出力部
○	○	○	○	ハイブリッド	ハイブリッド
○	○	○	×	画面	ハイブリッド
○	○	×	○	ハイブリッド	画面
○	○	×	×	画面	画面
○	△	○	○	シェイク	ハイブリッド

5.3　コンテキストの更新

提案 UI は, ユーザがコンテキストを手動で更新するか, デバイスに搭載されているセンサ [15] を利用することにより, コンテキストの変化を検知し, 更新する. ユーザが手動で更新するための画面を用意することにより, ユーザは手動で自身のコンテキストを更新し, 対応した UI へ切り替えることが可能となる.

デバイスに搭載されているセンサを利用することで, リアルタイムで検出できるコンテキストの変化については, 自動で検出することで, コンテキストを更新し, 対応した UI へ自動的に切り替えることができる. そのため, 第 4 章で定義した具体的なコンテキストのうち, センサを利用することにより検出することが可能なものとその際に使用するセンサについて表 6 に示す.

表 6　センサで取得するコンテキスト

コンテキスト	使用するセンサ類
目の忙しさ (ユーザが移動している)	・位置情報
ユーザがデバイスと対話する場所	・位置情報
手の状態 (ユーザとデバイスの間に距離がある)	・加速度センサ

5.4　UI 適用の流れ

Placeona に基づいて UI を適用する流れを図 1 に示す. まずシステムの起動時にユーザから Placeona の入力を受け取り, Placeona に対応した入出力 UI を適用する. システムを使用している間は, ユーザがデバイスの画面上から, 手動で Placeona を更新するか, デバイスに搭載されたセンサによりコンテキストの変化が検出された際に, Placeona が更新され, 対応した UI へ切り替わる. システムの使用中, ユーザの Placeona はその変化とともに更新され, 対応した UI が適用される. そして, システムを終了する際に, 保持されていた Placeona を初期化し, センサによるコンテキストの収集を終了する.

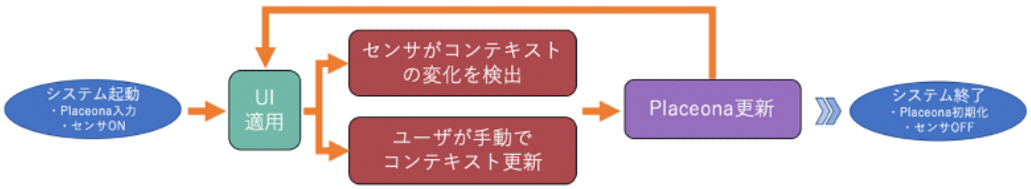


図 1　UI 適用の流れ

PlaceonaUI の実装例において, 自動で UI を切り替えるためのセンサを用いたコンテキストの取得の方法に関して述べる. 表 6 に示した, ユーザが移動しているというコンテキストに関して, デバイスの位置情報の差分を毎秒計算することで, デバイスの速度を算出し, デバイスの速度があらかじめ設定した閾値をあらかじめ設定した秒数を上回ったときに, ユーザが移動しているというみなし, コンテキストを検出する. また, ユーザとデバイスの間に距離があるというコンテキストに関して, ユーザがデバイスを手に持っていないことから, デバイスが静止していることと仮定し, それを加速度センサを用いて検出することで取得する. 加速度センサを用いることにより, デバイスの 3 次元方向にかかる加速度の差分を毎秒算出し, デバイスの速度があらかじめ設定した閾値をあらかじめ設定した秒数を下回ったときに, デバイスが静止していると見なすことで検出している.

6　アプリへの UI 組み込み

6.1　組み込みの流れ

PlaceonaUI のプロトタイプをモバイルデバイスアプリケーションで使用することができる UI として実装した. PlaceonaUI はシステムの機能や音声操作を管理する PlaceonaUISystem, システムとの音声対話の内容を視覚的に表示するための画面要素である VUIView, ユーザの Placeona を画面上で管理するための画面である PlaceonaView から構成されている. これらは, プログラムのコードとして用意されており, コピーしてベースアプリ内に組み込むことでそのまま使用することが可能である. 現状では, これらを Swift で書かれた iOS アプリケーション向けに実装をしている. ただし, 加速度センサや位置情報を扱うことが可能であれば, 他のプログラム言語やデバイスにおいても適用することが可能である.

PlaceonaUI をモバイルデバイスアプリケーションに組み込む流れを図 2 に示す. PlaceonaUI を組み込むベースとなる GUI ベースのモバイルアプリケーション (ベースアプリ) が用意されていることを前提とする. はじめに, ベースアプリの開発者は, ベースアプリの中で画面操作を用いて行うタスクを, 音声操作でも同様に達成することを可能にするため, PlaceonaUI で使用できる記述形式で音声操作のプログラムのコードを用意する. PlaceonaUI 使用するための音声操作は, その操作を実行するための音声入力の命令, 対応するアプリケーションの動作, 音声出力をセットで用意する必要がある. 次に, 開発者が, 用意した音声操作を PlaceonaUISystem の中に挿入する. 開発者は, ベースアプリで使用するための音声操作が用意された PlaceonaUI をベースアプリのプログラムのファイルへ追加し, ベースアプリの View ファイルに, VUIView と PlaceonaView を画面要素として配置するためのコードを記述することで, ベースアプリへの組み込みが実現する. このとき, 各画面要素を配置する場所は, 開発者自身がベースアプリに合わせて決める.

6.2　音声操作を用意すべきタスク

開発者がベースアプリで行うタスクの音声操作を用意する際, 画面操作と音声操作の性質の違いから, 音声操作を用意するべきタスクと不要なタスクが存在する. PlaceonaUI の使用において, 画面を見ながら音声操作を行う場合と, 音声のみで操作を行う場合があるため, それぞれで開発者が用意する必要のあるタスクは以下の通りである. まず, 画面を見ながら音声操作で実現可能なタスクである. このタスクには, 手による複雑な操作が要求されないタスクが該当する. 次に, 音声のみで実現可能なタスクである. このタスクには, 手による複雑な操作が要求されない, 画面を見ながら行う必要がない, 音声で代替することができない画面上の情報を得ることが目的ではないという 3 つの条件の当てはまるタスクが該当する.

7　評価

7.1　評価実験

7.1.1　概要

評価実験の目的を 2 点述べる.

1 つは, PlaceonaUI がユーザのコンテキストを適切に検知できており, かつ, それに合わせて切り替わる UI が適切であるかを検証することである. 2 点目は, PlaceonaUI の有効性を検証することである. 本評価における有効性とは, PlaceonaUI を使用することにおける, ユーザの利便性を表す.

PlaceonaUI のプロトタイプを組み込んだサンプルモバイルデバイスアプリケーションを使用し, ユーザテストにより評価実験を行なった. 被験者のコンテキストが変化するシチュエーションを含む 3 つのシナリオとジャンルの異なる複数のサンプルモバイルデバイスアプリケーションを用意した. 被験者は, アプリケーションの操作に関する説明を受けた後, シナリオを理解し, それぞれのシナリオを再現したタ

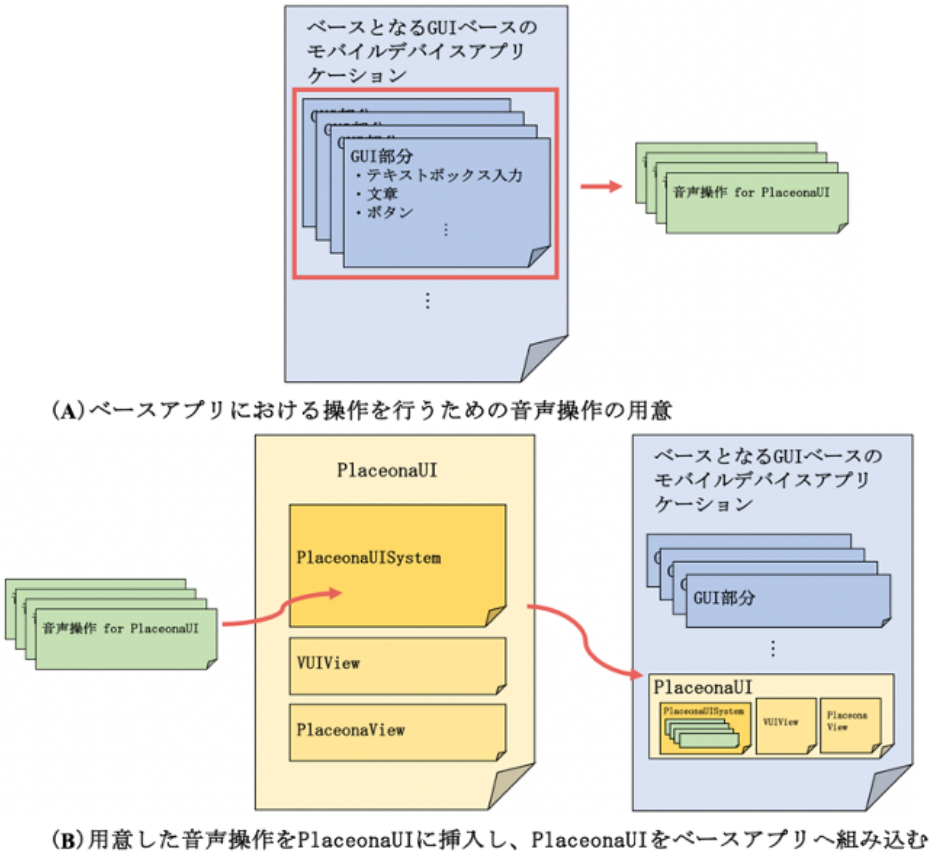


図 2　PlaceonaUI 組み込みの流れ

スクをこなした後, アンケートに回答した. ユーザテストには 20 代の男性と女性の計 14 人が参加した. 被験者のうち 12 人は普段から音声操作を使用していなかった. アンケートでは, リッカート尺度として, そう思わない, ややそう思わない, ややそう思う, そう思うの 4 つの選択肢から回答する質問と自由記述による質問を設けた. アンケート項目の一部を図 3 に示す.

Q. 最も当てはまるものを選択してください

状況の変化に対して、切り替わった操作方法は適切であった
1. そう思わない　2. ややそう思わない　3. ややそ思う　4. そう思う

本シチュエーションにおいて、音声操作と画面操作、シェイク動作による操作を使い分けることができるのは便利である
1. そう思わない　2. ややそう思わない　3. ややそ思う　4. そう思う

本シチュエーションで使用したアプリケーションの操作方法に関して、どのような点が便利であると感じましたか?（自由記述）

図 3　アンケート項目 (一部)

7.1.2　シナリオ

ユーザテストで使用する 3 つのシナリオに関して説明する. 各シナリオをシナリオ 1, シナリオ 2, シナリオ 3 とする.

シナリオ 1 は, ニュースアプリ利用中に作業を行うシナリオである. PlaceonaUI

が, ユーザとデバイスの間に距離があるというコンテキストを検知し, 動作が切り替わることを検証し, 手が使用できなくなるというコンテキストの変化に対して切り替わる UI がシチュエーションの中で有効であるかどうかを検証することができるシナリオとして用意した. また, 6.2 節の条件から音声操作が用意されるタスクで使用できるニュースアプリケーションに PlaceonaUI を組み込み使用した.

シナリオ 2 は, 歩行中にモバイルオーダーのアプリを使用してテイクアウトドリンクを注文するシナリオである. PlaceonaUI が, ユーザが移動しているというコンテキストを検知し, 動作が切り替わることを検証し, 目が使用できなくなるというコンテキストの変化に対して切り替わる UI がシチュエーションの中で有効であるかどうかを検証することができるシナリオとして用意した. また, ユーザ手動によるコンテキスト更新に関しても, 有効性を検証する. また, 6.2 節の条件から音声操作が用意されるタスクで使用できるモバイルオーダーのアプリケーションと ToDo リストのアプリケーションに PlaceonaUI を組み込み使用した.

シナリオ 3 は, 車の運転中に ToDo リストのアプリケーションを使用して, 予定管理を行うシナリオである. PlaceonaUI が, ユーザが移動しているというコンテキストとユーザとデバイスの間に距離があるというコンテキストの両方を検知することで, 動作が切り替わることを検証し, 切り替わる UI がシチュエーションの中で有効であるかどうかを検証することができるシナリオとして用意した. また, シナリオ 2 と同様の ToDo リストのアプリケーションを使用した.

7.2 結果

アンケート結果について, リッカート尺度を用いた質問については, 「そう思わない」を 1 点, 「そう思う」を 4 点としてスコア付けした. また, スコアのバラつきを求めるため, 標準偏差を算出した. 各評価項目とそのスコア, 標準偏差を表 7 に示す.

表 7　評価項目とスコア

評価項目	平均スコア	標準偏差
1. 適用ルール通りに UI が切り替わったか	3.74	0.440
2. 手の状態を検知し, 切り替わった UI が適していたか	3.71	0.452
3. 手の状態を検知し, UI が切り替わることは便利か	3.79	0.558
4. 音声操作と画面操作を使い分けることができることは便利か (シナリオ 1)	3.79	0.558
5. 目の忙しさを検知し, 切り替わった UI が適していたか	3.36	0.610
6. 目の状態を検知し, UI が切り替わることは便利か	3.50	0.500
7. 音声 UI と画面 UI, シェイクを使い分けることができることは便利か (シナリオ 2)	3.43	0.495
8. 手の状態と目の忙しさを同時に検知し, 切り替わった UI が適していたか	3.86	0.350
9. 手の状態と目の忙しさを同時に検知し, UI が切り替わることは便利か	3.79	0.410
10. 音声 UI と画面 UI を使い分けることができることは便利か (シナリオ 3)	3.86	0.350

また, 自由記述による質問について, PlaceonaUI の便利な点と不便な点に関して, 回答が多かった回答に関して, 以下に, 回答の内容と回答数について示す.

- 便利な点
  - 自動で操作が切り替わる (シナリオ 1)：3 件
  - 自動で操作が切り替わる (シナリオ 2)：4 件
  - 自動で操作が切り替わる (シナリオ 3)：6 件
  - 音声操作が使用できる：4 件
  - シェイクが使用できる：2 件
  - マルチタスクが可能になる：3 件
- 不便な点
  - UI が自動で切り替わらなくても良い場合もある：3 件
  - 操作に時間がかかる (シナリオ 1)：3 件
  - 操作に時間がかかる (シナリオ 2)：4 件
  - 操作に慣れていない：6 件

7.3 考察

実験の目的に基づいて PlaceonaUI について評価を行うために, ユーザテストにおいて, PlaceonaUI がユーザのコンテキストを検知し, 対応した UI を適用できている必要がある. これに関して, 表 7 の項目 1 の結果より, PlaceonaUI はユーザのコンテキストを検知して動作が変化したことが示された.

ユーザのコンテキストに対して適用された UI が適切であったかについて評価する. まず, 表 7 の項目 2, 5, 8 の結果より, 適用された UI は概ね適切である. 項目 2, 8 の平均スコアはどちらも 3.7 以上であるのに対して, 項目 5 の平均スコアは 3.36 と劣っている. シナリオ 2 における不便な点として, 周囲が気になるという回答が得られた. これに関して, 音声操作やシェイクに対してユーザが抵抗感を持つことがあることから, 音声操作やシェイクは全てのユーザにとって適切ではないため, ユーザによっては適用する必要がない場合もある. この場合, ユーザは手動でコンテキストを更新することも可能である.

ユーザのコンテキストに対して, UI が自動で切り替わることの有効性について評価する. 表 7 の項目 3, 6, 9 の結果より, UI が自動で切り替わることは有効である. また, それぞれのシナリオにおいて, UI が自動で切り替わることに対して便利であるという回答が複数得られたことからも, UI が自動で切り替わることは有効である. ただし, UI が自動で切り替わらなくて良い場合もあるという回答も得られた. このことから, UI が自動で切り替わるかどうかをユーザ自身が管理, 設定できるとより有効である.

VUI, GUI, シェイクによる UI を使い分けることの有効性について評価する. 表 7 の項目 4, 7, 10 の結果より, これらの UI を使い分けることは有効である. また, 音声操作とシェイクが使用できることが便利であるという回答や, マルチタスクが可能になることが便利であるという回答が得られたことからも, これらの UI を使い分けることは有効である. ただし, 音声操作とシェイクによる操作に慣れていないことが不便であるという回答が複数得られたことから, それぞれの操作方法に慣れているかどうかは, それらの UI を使い分けることの有効性に影響する. また, 音声操作に対して, 操作に時間がかかることが不便であるという回答が得られた. 音声操作で達成することができても, 画面操作と比較して時間を要するタスクに関しては, 音声操作が有効ではない場合もある.

評価の結果, UI が自動で切り替わることをユーザ自身が管理や設定できることや音声操作やシェイクによる操作に慣れることによる有効性への影響が示されていない点に問題はあったが, PlaceonaUI はユーザのコンテキストを検知できており, かつ, 切り替わる UI が概ね適切であることが示された. また, PlaceonaUI において, ユーザのコンテキストを収集し自動で UI が切り替わることと, VUI, GUI, シェイクによる UI をコンテキストに応じて使い分けることが有効であることが示された. このことから, PlaceonaUI は, ユーザのコンテキストを考慮した VUI の使用を実現し

ており, PlaceonaUI の適用ルールは, ユーザのコンテキストを考慮した VUI 設計の基準として有効であることが示された. また, VUI, GUI, シェイクによる UI をユーザのコンテキストに応じて使い分けることの有効性も示されたことで, 本研究の有効性は示された.

8 おわりに

本研究では, Placeona に基づいたユーザのコンテキストに応じて VUI, GUI, シェイクの UI を適切に切り替える PlaceonaUI を提案した. PlaceonaUI をモバイルデバイスのアプリケーションの UI として使用する際のコンテキストの検知・更新, UI の切り替えの方針を定義し, アプリケーションの UI として組み込む流れを示した. そして, PlaceonaUI のプロトタイプを使用したユーザテストによる評価実験を行い, アンケート調査を通じて得られた結果から, その有効性を示した. これにより, 未だ確立していないユーザのコンテキストを考慮した VUI 設計の発展とマルチモーダルインタフェースのさらなる発展に期待できる.

また, 本研究の今後の課題は以下の通りである.

- より多様な被験者による評価実験
- PlaceonaUI のアプリケーションへの組み込み支援
- 各センサを用いた適切なコンテキスト取得や取得精度の向上

参考文献

[1] Muhammad Zeeshan Baig, Manolya Kavakli, Qualitative analysis of a multimodal interface system using speech/gesture, 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), pp. 2811-2816, 2018.

[2] 岡田健一, 葛岡英明, 塩澤秀和, 西田正吾, 仲谷美江: ヒューマンコンピュータインタラクション, オーム社, 2002, pp.128-134.

[3] 井上勝雄 編: インタフェースデザインの教科書 第 2 版, 丸善出版, 2019, pp.113.

[4] Chelsea Myers, Anushay Furqan, Jessica Nebolsky, Karina Caro, Jichen Zhu, Patterns for How Users Overcome Obstacles in Voice User Interfaces, Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems(CHI '18), Paper No.6 pp.1-7, 2018.

[5] Aaron Springer, Henriette Cramer, "Play PRBLMS": Identifying and Correcting Less Accessible Content in Voice Interfaces, Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems(CHI '18), Paper No.296 pp.1-13, 2018.

[6] Chelsea M. Myers, Anushay Furqan, Jichen Zhu, The Impact of User Characteristics and Preferences on Performance with an Unfamiliar Voice User Interface, Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems(CHI '19), Paper No.47 pp.1-9,2019.

[7] Human Interface Guidelines - Design - Apple Developer, https://developer.apple.com/design/human-interface-guidelines/guidelines/overview/, 2022/07/22 閲覧.

[8] マテリアル デザインの基本 - Android Developers, https://developer.android.com/design?hl=ja, 2022/07/22 閲覧.

[9] What are voice personas and placeonas? A guide for designers, https://careerfoundry.com/en/blog/ux-design/voice-design-personas-placeonas/, 2022/07/22 閲覧.

[10] Christine Murad, Leigh Clark, Cosmin Munteanu, Benjamin R. Cowan, Revolution or Evolution? Speech Interaction and HCI Design Guidelines, IEEE Pervasive Computing, Volume:18, Issue:2, pp. 33-45, 2019.

[11] Americo Talarico Neto, Renata Pontin M. Fortes, Adalberto G. da Silva Filho, Multimodal interfaces design issues: the fusion of well-designed voice and graphical user interfaces, Proceedings of the 26th annual ACM international conference on Design of communication(SIGDOC '08), pp.277-278, 2008.

[12] D. Natarajasivan, M. Govindarajan, Location Based Context Aware user Interface Recommendation System, Proceeding of the International Conference on Informatics and Analytics(ICIA-16), Article No.78 pp.1-6, 2016.

[13] Ling Feng, Context-Aware Computing, Walter de Gruyter, pp.6-7, 2017.

[14] Punnarumol Temdee, Ramjee Prasad, Context-Aware Communication and Computing: Applications for Smart Environment, Springer Cham, pp.15-28, 2018.

[15] iOS で使えるセンサー・ハードウェア機能, https://qiita.com/takitakit/items/c84fdac046ef8b0c0b3a, 2022/07/22 閲覧

静的解析と動的解析の組み合わせによるUML ステートマシン図答案の誤り特定自動化手法の提案

An Automated Method of Identifying Errors in Learner's UML State Machine Diagrams by Combination of Static and Dynamic Analysis

五島 光祥[*] 小形 真平[†] 槇原 絵里奈[‡] 岡野 浩三[§]

あらまし UML（Unified Modeling Language）の振る舞いモデルでは同じ振る舞いを複数の異なる方法で記述できる．そのため，ステートマシン図教育では，教育者が学習者にフィードバックを提供する上で，多数の学習者が個々に作成した答案モデルの誤りを特定する作業が大きな負担となっている．この問題を改善するために，我々はこれまで，モデル記述課題において正答例と答案間の記述の違いから答案の誤りを自動特定する静的解析手法を提案し，評価してきた．しかし，変数や，正答例と答案間の状態が1対多等の対応関係の場合に，特定できない誤りがあった．そこで本稿では，正答例と答案を，記述に基づき対応付ける静的解析と，振舞いシミュレーション時の実行情報に基づき対応付ける動的解析を組み合わせて，誤りの箇所や種類を特定する手法を新たに提案する．評価実験では，変数を含む95個の答案に対して提案手法を適用し，先行手法では誤判定され，特定できなかった12個の答案の誤りを正しく特定できることがわかった．

1 はじめに

OMG（Object Management Group）[1]が標準化しているUMLのステートマシン図[2]は，離散的なイベント駆動の振る舞いを表す図であり，システム全体の振る舞いを表現できる．近年では，組込みシステムやCPS（Cyber-Physical System），IoT（Internet of Things）等の分析・設計に用いられており，これらの普及に伴ってステートマシン図を使用する機会が増えている[3] [4]．また，中学校学習指導要領[5]では，情報分野に関して“全体構成やアルゴリズムを図に表す力”の育成についての記載があり，振る舞いを図で表現するための学習として，ステートマシン図を用いた教育が行われている[6]．そして，学習者の“課題文等の要求から必要な情報を読み取る要求分析能力”[7]を育む機会にもなるため，ステートマシン図教育の重要性が高まっている．

ステートマシン図教育では，課題文からステートマシン図を作成する記述課題を学習者に課すことがある．そして，教育者は学習者の答案に対して，どの箇所にどのような種類の誤りがあるのか解説するために，誤りを特定する必要がある．ここで述べる誤りの種類とは，先行研究のステートマシン図教育に関する調査[8] [9]において，要素の過不足や遷移の繋ぎ間違い等の誤りやすい箇所を，学習者にフィードバックすべき誤りとして整理したものである．しかし，ステートマシン図は答案ごとにレイアウトが様々で，一見してどの箇所に誤りを含むのか判別できない場合があるため，教育者がステートマシン図答案から誤りの箇所や種類を特定することは時間がかかる作業である．そのため，ステートマシン図教育では，限られた時間の中で学習者に個別のフィードバックを早期に与えることが難しい．これらの背景から，ステートマシン図の教育現場では，誤りの箇所や種類の特定を支援する手法が必要とされている．

[*]Mitsutada Goshima, 信州大学
[†]Shinpei Ogata, 信州大学
[‡]Erina Makihara, 同志社大学
[§]Kozo Okano, 信州大学

ステートマシン図教育における誤り特定を支援するために，我々はこれまで正答例と答案間の要素を対応付けることで，対応箇所の差分からステートマシン図の誤りを自動で特定する静的解析手法を提案してきた [10]．本手法の評価実験では,「状態，開始疑似状態，終了状態，遷移，実行活動，トリガー」の要素で構成されたステートマシン図に対して，正答例と答案間で要素が 1 対 1 で対応付く場合に，状態と遷移の過不足や遷移の繋ぎ間違い等の誤り箇所を特定できている．しかし，静的解析では対応付けた要素間に対して記述の違いから誤り特定を行うため，記述は異なるが本質的に同じ振る舞いを表すステートマシン図に対応していない．例えば，変数を含むステートマシン図では，変数の初期値等に応じてガード条件の式が変化するが，記述が異なっても同じ振る舞いを表す場合がある．また，正答例と答案が同じ振る舞いを表していても，要素数が互いに異なる場合や，１対多あるいは多対１で対応関係が説明できる要素がある場合は，記述の差分を比較する静的解析で対処することは難しい．

以上より，本稿ではステートマシン図教育の支援を目的とし，変数や正答例と答案間の状態が１対多等の対応関係を含む記述にも適応できるよう先行手法を改良した手法として，静的解析手法と動的解析手法を組み合わせた誤り特定自動化手法を提案する．本稿で新たに提案する動的解析では，ステートマシン図の開始疑似状態を現在状態として，イベントが複数回生起したときに得られる一連の状態遷移を記録した実行トレースを解析することで，変数や状態が１対多等の対応関係を含む記述等の誤りに対処する．動的解析は，正答例と答案のステートマシン図から異なる実行トレースが生成された場合に，正答例と答案間で同じ実行活動を持つ状態の対応関係をもとに同じ振る舞いか比較し，実行トレース上で差異が生じた箇所から誤り特定を行う．提案手法では，振る舞いを自動で正誤判定する先行研究の SML4C（SMart-Learning for Classification） [11] を応用することで，実行トレースの生成を行う．

提案手法の有効性を評価するために，先行研究 [10] の評価実験で用いた 5 種類の記述課題に対して，学習者 19 名が作成した計 95 個の答案を提案手法の入力として与え，先行手法と提案手法の性能を比較した．先行手法では適用事例における 12 個の答案が誤判定されていたが，提案手法ではこれらの答案の誤りを全て適切に特定できていたため，先行手法よりも高い精度で誤りを特定できる見込みを得た．

本稿は以下のように構成されている．2 章では本研究の動機例題について説明する．3 章では提案手法の詳細について説明する．4 章では提案手法の評価実験の詳細及び結果と考察を示す．最後に，5 章ではまとめと今後の課題について述べる．

2　動機例題

本章では，先行研究の静的解析手法 [10] の詳細，及び入力するステートマシン図について説明し，手法の問題点と問題解決のアイディアについて述べる．

2.1　入力するステートマシン図の詳細

先行研究では，振る舞いの開始を表す「開始疑似状態」，振る舞いの終了を表す「終了状態」，ある不変的な条件が成立する状況を表す「状態」，状態から次の状態への変化を表す「遷移」という 4 つの要素で構成されるステートマシン図を入力として与えた．また，状態と遷移を構成する要素として，ある状態中に継続的に実行される動作を表す「実行活動」，遷移の契機となるイベントを表す「トリガー」，遷移が可能となる条件を表す「ガード」，遷移中に行われる処理を表す「エフェクト」を含んでいる．なお,「ステートマシン図の振る舞い」とは，イベントの生起により得られる状態遷移の連なりで表現される動作である．

ステートマシン図の正答例を図 1，誤りを含む答案を図 2 に示し，各要素について説明する．図 1，図 2 はタイマーのステートマシン図を抽象化した図で，図中で

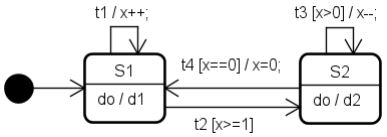


図 1　ステートマシン図の正答例

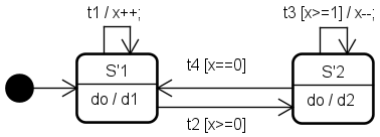


図 2　誤りを含むステートマシン図の答案

は四角形で状態，矢印で遷移を表している．また，実行活動は d1 と d2，トリガーは t1，t2，t3，t4，エフェクトは x++等，ガードは x ＞=1 等が該当する．なお，開始疑似状態から S1，S'1 への遷移はトリガーを記述できない null 遷移であるが，説明のために本稿ではトリガーは t0 として扱う．また，開始疑似状態には実行活動が存在しないが，説明のために本稿では d0 として扱う．

2.2　静的解析の問題点

我々の先行研究では，正答例と答案間の状態を対応付けることで，記述上の誤りの箇所を特定する静的解析手法を提案している [10]．静的解析では，図 1 と図 2 を入力した場合，S1 と S'1，S2 と S'2 が同じ実行活動であるため，1 対 1 で対応付けられる．なお，ステートマシン図には同じ実行活動を重複して含む場合や，異なる対応関係を含む場合があるため，それらのパターンごとに対応付け方法が異なる．「異なる対応関係」とは，正答例のある要素に対して，状態や遷移の要素を複数個にわけて記述している場合や，複数ある要素を一つの要素としてまとめて記述している場合等，要素が 1 対 1 でない対応関係である．静的解析の手順について以下に示す．

1. ステートマシン図間で同じ実行活動の状態間を比較項目 [12] をもとに比較する．比較項目は，隣接する状態，入場遷移のトリガー名，退場遷移のトリガー名，状態名，開始疑似状態との接続の有無，という 5 項目である．
2. 比較項目が最も一致する状態間を 3 つのパターンごとに対応付ける．
   - 同じ実行活動の状態が 1 つしか存在しない場合は，1 対 1 で対応付ける．
   - 同じ実行活動の状態が複数で重複し，正答例と答案間で数が同じ場合は，比較項目をもとに，正答例と答案で各状態ごとに 1 対 1 で対応付ける．
   - 同じ実行活動の状態が複数で重複し，正答例と答案間で数が異なる場合は，状態個別に 1 対 1 では対応付けずに，同じ実行活動のグループとして 1 対 1 に対応付ける．
3. 対応付けた状態間で繋がっている遷移や前後の状態の差分から誤り特定を行う．

本手法の問題点として，対応付けた要素間の記述の違いから誤り特定を行うため，記述は異なるが本質的に同じ振る舞いを表すステートマシン図に対応していない．本研究における「同じ振る舞い」とは，たとえ図間で記述が異なっても，同じイベント列を与えたときに，得られる実行活動とエフェクトの連なりが同じになることを指す．なお，正答例と記述が異なるが正しい振る舞いを表す答案は「別解」として扱う．

図 1，図 2 の例をもとに，変数を用いて異なる記述で同じ振る舞いを表す場合について説明を行う．なお，図 1，図 2 の変数の初期値は「x=0」とする．図 1，図 2 を本手法に入力した場合，図 2 上では「t2 のガード [x ＞=0] の範囲」,「t3 のガード [x ＞=1] の範囲」,「t4 のエフェクト x=0 が不足」，という 3 個の記述上の違いが表れる．手動で図 1，図 2 間の t2 のガードを比較すると，図 1 は t1 のイベントが必ず 1 回以上生起しないと t2 を持つ遷移のガード条件が満たされないことに対し，図 2 は 1 回も t1 を経由しなくても t2 を持つ遷移のガード条件は満たされるため，図 2 の「t2 のガード [x ＞=0] の範囲」は誤りである．一方で，図 1 と図 2 は記述は異なっているが,「t3 のガード [x ＞=1] の範囲」の違いや,「t4 のエフェクト x=0 が不足」によらず，同じ振る舞いを表す．そのため，手動で比較した場合に，実際に誤

りと判定されるのは,「t2 のガード [x ＞=0] の範囲」のみとなる．しかし，静的解析では上記の 3 つの例のように，記述上の違いは全て誤りと判定されてしまう．

上記の例以外にも，変数の初期値の違いや，エフェクトで変数の値の制御等でも異なる記述で同じ振る舞いを表せる．また，適切に一つの要素を記述した答案と，その要素を複数個にわけて記述した答案でも，同じ振る舞いを表せる．これらを踏まえて，ステートマシン図教育では，記述の違いだけでは誤りかどうか判定できない図が存在し，変数や異なる対応関係を含む場合に対応する必要がある．

2.3 解決アイディア

2.2 節で示した先行手法の問題点を解決するために，静的解析手法と動的解析手法を組み合わせることで，実行トレースに基づいた誤り特定を行う手法を新たに提案する．実行トレースとは，開始疑似状態を現在状態として，イベントが複数回生起したときに得られる一連の状態遷移を「実行活動 1，トリガー 1，実行活動 2，トリガー 2,...」のように記録した情報である（以降，トレースと呼ぶ）．また，本稿では，トレースにおいて状態間を遷移する際にイベントが生起した回数を以降，単に生起数と呼ぶ.

正答例と答案間でトレースを生成した際に，正答例と共通でないトレースは「答案に過剰な振る舞いが存在する」，または「答案に必要な振る舞いが不足している」ことを意味し，この誤った振る舞いのトレース上に含まれる状態と遷移の要素に誤りが含まれる．動的解析では，正答例と答案から生成したトレース間の比較により，誤りを含む「正答例と答案間で共通でないトレース」に着目して誤り特定を行う．

従来技術として，ステートマシン図の振る舞いを自動で正誤判定する SML4C（SMart-Learning for Classification）[11] が提案されている．SML4C は，複数のステートマシン図と，生起数を入力にとり，その生起数で生起しうる全てのトレースを正誤判定の過程で生成する．SML4C は，誤りの箇所や種類は特定できないが，変数や異なる対応関係を含むステートマシン図に対処できる．また，生成されるトレースには，生成元となる 1 つ以上のステートマシン図の ID（ファイル名）が明記される．そこで本研究では，正答例と答案のステートマシン図，及び生起数を SML4C の入力として与え，提案手法で用いるトレースを生成する．提案手法で与える生起数は，全てのトレースを総合して，全ての状態と遷移がトレース中に含まれるために必要な最小の数以上に手動で設定する.

SML4C に図 1，図 2 を入力として与え，x の初期値を 0，生起数を 2 回で実行した場合を想定し説明する．この場合のトレースは,「d0，t0，d1，t1，d1」,「d0，t0，d1，t2，d2」,「d0，t0，d1，t2，-」の 3 種類が生成される．なお，トレースではガードとエフェクトは省略してトリガーのみが記載される．また,「-」はイベントが生起したが，変数の値等がガードの条件を満たさずに状態間の遷移が起きなかった場合を表す.「d0，t0，d1，t1，d1」は，図 1，図 2 から共通で生成されるトレースである.「d0，t0，d1，t2，d2」は，図 2 から生成されるが，図 1 からは x の値が 0 であるため t1[x ＞=1] の条件を満たさず生成されない.「d0，t0，d1，t2，-」は，図 1 から生成されるが，図 2 からは生成されない.

提案手法では，SML4C を用いて上記に示すような 3 種類のトレースを生成し，変数や異なる対応関係に対処した誤り特定を行う.

3 提案手法

本章では，提案手法の全体像を示し，提案手法の内容について紹介する.

3.1 ステートマシン図の誤りの種類

Pourali らの研究 [8] と香山らの研究 [9] では，ステートマシン図教育における誤りやすい箇所等の調査が行われている．これらの調査結果を元に，我々は先行研究 [12]

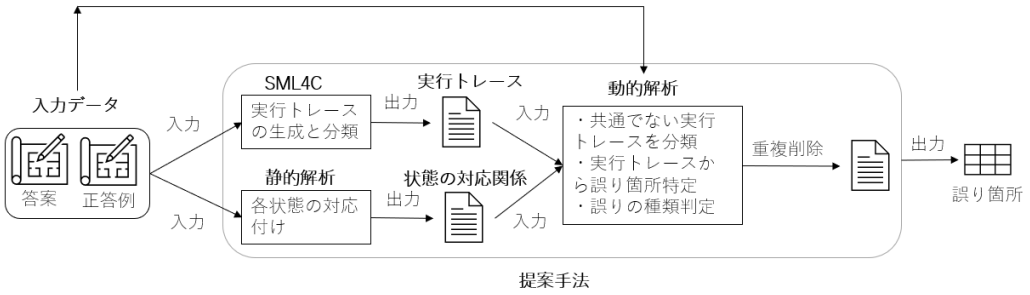


図 3　提案手法の全体像

において，ステートマシン図教育における学習者にフィードバックすべき誤りの種類を「状態の不足」,「状態の過剰」,「遷移の不足」,「遷移の過剰」,「接続遷移の不足」,「接続遷移の過剰」,「遷移先誤り」,「遷移元誤り」の 8 種類と定義した．

提案手法では，先行研究では非対応であった変数を含むステートマシン図課題に関して，適用範囲を拡大している．そのため本稿では，正答例と答案間でガードの条件式の範囲が異なる場合や，エフェクトで変数の値を変化させる処理の記述漏れ等，変数制御による誤りを新たに「変数誤り」として定義する．

3.2　手法の概要

本節では，提案手法の概要として，SML4C，静的解析，動的解析を組み合わせた自動誤り特定の全体像，及び手法を適用する前提について説明を行う．なお，本手法は先行研究 [10] からの継続的な研究であるため，2.1 節の用語集の前提条件等，静的解析の提案で定義した内容を引き続き利用する．

先行研究 [11] では，ステートマシン図の記述課題において，状態の動作を表す「実行活動」と，遷移の契機を表す「トリガー」を指定する用語集が学習者に提供されていた．用語集はステートマシン図の作成に必ずしも必要であるとは限らないが，先行研究では要求仕様通りの記述を実践的に演習で行うために，用語集を用いた課題が学生へ提供されていた．提案手法では，上記の「実行活動」と「トリガー」が用語集で定められた課題を想定しており，用語集を用いることで用語が統一されるため，状態や遷移の対応付けをより正確に行うことができる．

提案手法の全体像を図 3 に示す．提案手法では，2.3 節で示した先行研究の SML4C でトレースを生成し，静的解析と動的解析を組み合わせることで，従来の記述の違いに振る舞いの違いの観点を加え，変数や異なる対応関係を含むステートマシン図に対処し，誤りの箇所，及び 3.1 節の誤りの種類を自動特定する．提案手法の実現アイディアとして，正答例と答案の間で共通でないトレースに対して，互いに差異が初めて生じる箇所を誤り箇所とする．以下に手法の手順を説明する．

手順 1　SML4C に正答例と答案を入力し，トレースを生成する．なお各トレースは，生成元となったステートマシン図の ID が割り付けられたものとなる．

手順 2　静的解析では，動的解析で正答例と答案間の状態の対応関係を用いるために，用語集をもとに入力した全ての答案を正答例の状態と対応付ける．

手順 3　動的解析では，誤りが含まれるトレースを特定するために，正答例と共通でないトレースを生成した答案をラベル付けする．入力は正答例と答案，実行トレース，状態の対応関係を与える．

手順 4　最後に，各トレースに対して，互いに差異が初めて生じる箇所を誤りとして特定する．また，特定した誤りがどの誤りの種類に該当するのか判定する．最終的な出力として，誤りの箇所と種類を出力する．

表 1　各実行トレースの分類

トレースの分類	生成元	トレース
誤った答案のみで生成	図 2	d0，t0，d1，t2，d2，...
誤った答案から生成されない	図 1	d0，t0，d1，t2，-，...
⋮	⋮	⋮

3.3　動的解析手法

本節では，2.2 節での状態間の対応付けと，2.3 節の SML4C のトレースを用いた動的解析手法による誤り特定の詳細について図 1，図 2 を用いて説明する．

3.3.1　実行トレースの分類

まず，SML4C から生成したトレースに対して，割り付けた生成元のステートマシン図の ID をもとに，正答例と答案間で共通のトレースと共通でないトレースをそれぞれ紐づける．ここで，2 種類の共通でないトレースは，実現不能な答案のトレースを「誤った答案のみで生成されるトレース」，答案が再現不能な正答例のトレースを「誤った答案から生成されないトレース」とそれぞれラベル付けする．これら 2 種類の正答例と答案間で共通でないトレースは異なる振る舞いを表し，このトレース上の状態または遷移に誤りを含む可能性があるため，動的解析に用いる．なお，共通のトレースは「正答例と答案に共通で生成されるトレース」としてラベル付けされるが，正答例と答案間で同じ振る舞いを表す差分のないトレースのため，提案手法では用いない．

図 1，図 2 を生起数を 5 回，変数 x の初期値を 0 でトレースを生成した際の結果の一部を表 1 に示す．なお，提案手法では「正答例と答案に共通で生成されるトレース」は使用しないため，「d0，t0，d1，t1，d1，t2，d2...」等の図 1，図 2 に共通のトレースは表 1 で省略している．また，「d0，t0，d1，t1，d1，t2，d2，t3，d2，t4，d1」が，生起数が最小で 5 の時，最短で t4 を生起可能なトレースであり，全てのトレース中に全ての状態と遷移が含まれるため，生起数は 5 回に設定している．

表 1 に示した「d0，t0，d1，t2，d2，...」，「d0，t0，d1，t2，-，...」は，それぞれ 2.3 節で示した図 1，図 2 に共通ではないトレースである．「d0，t0，d1，t2，d2，...」は，図 2 の答案のみで生起可能であるため，「誤った答案のみで生成されるトレース」をラベル付けする．「d0，t0，d1，t2，-，...」は，図 1 の正答例のみで生起可能であるため，「誤った答案から生成されないトレース」をラベル付けする．

3.3.2　実行トレースの比較による誤り特定

ステートマシン図答案から誤り箇所を特定するために，3.3.1 項で分類したトレースと，静的解析の対応付けによる状態の対応関係を入力として用いる．静的解析の対応付けは，2.2 節に示した通り，同じ実行活動が重複して存在する場合は，その状態と繋がっている遷移等の，周辺の要素を比較項目として最も一致するものと対応付けを行う．動的解析ではトレースのラベル付けの結果に応じて，トレースに含まれる状態と遷移を，ステートマシン図の開始疑似状態から順に照合し，差異が初めて生じた箇所を誤りとして特定する．

まず，3.3.1 項で示したトレースのうち，「誤った答案のみで生成されるトレース」にラベル付けされた場合の誤り特定について説明する．「誤った答案のみで生成されるトレース」は答案において，要素が過剰である，ガードの範囲の誤り，遷移を繋げる箇所の誤り等が誤りと考えられる．そのため，正答例から生成されない「誤った答案のみで生成されるトレース」を用いて，正答例でこのトレースと同じトリガーを同じ順番で生起させた際に，答案の状態や遷移のどの箇所で生起が不可能になり，誤りが生じているのかについて検証する．検出された誤りに対して，その生起が不可能になった箇所やその内容に応じて，誤りの種類を特定する．

次に，3.3.1 項で示したトレースのうち，「誤った答案から生成されないトレース」に分類された場合の誤り特定について説明する．「誤った答案から生成されないト

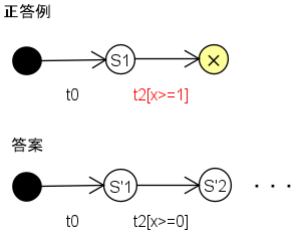


図 4　誤った答案のみで生成されるトレースの判定

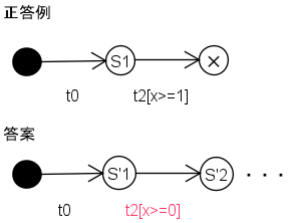


図 5　誤った答案から生成されないトレースの判定

レース」は，正答例に含まれる正しいトレースを答案が含まない場合であるため，その答案でこのトレースと同じトリガーを同じ順番で生起させた際に，答案の状態や遷移のどの箇所で生起が不可能になり，誤りが生じているのかについて検証する．以降は,「誤った答案のみで生成されるトレース」と同じ処理手順である．

3.3.3　誤りの種類の判定

3.1 節で示した誤りの種類の定義に基づき，誤りの種類の判定方法を説明する．

遷移先誤り　正答例との対応関係に基づき，遷移先の状態が正しくない遷移が答案にある．

遷移元誤り　正答例との対応関係に基づき，遷移元の状態が正しくない遷移が答案にある．

変数誤り　正答例との対応関係に基づき，変数の操作・条件に関する意味的な相違を理由として正しくない状態・遷移が答案にある．

「遷移先誤り」,「遷移元誤り」,「変数誤り」は，記述上の違いだけではなく，正答例と同じイベントを生起させて答案が同じ状態に遷移できるかといった実行を伴う判定が必要であるため，動的解析を行う．なお,「状態の不足」,「状態の過剰」,「遷移の不足」,「遷移の過剰」,「接続遷移の不足」,「接続遷移の過剰」は記述上の違いに関する誤りの種類であるため，静的解析を行う．これらの静的解析と動的解析の判定結果を組み合わせることで，9 種類全ての誤り特定の結果を出力する，

表 1 に示した各トレースに対して,「d0，t0，d1，t2，d2，...」の判定を図 4,「d0，t0，d1，t2，-，...」の判定を図 5 に図示して説明を行う．なお，図 1，図 2 の例では，2.2 節の静的解析で説明した通り，実行活動が重複して存在していないため，S1 と S'1，S2 と S'2 が 1 対 1 で対応付けられる．

図 4 は,「誤った答案のみで生成されるトレース」の「d0，t0，d1，t2，d2，...」に対して，正答例を開始疑似状態から順に照合し，どの要素で生起不能な差異が初めて生じるか検証した結果である．まず，正答例に関して，開始疑似状態 d0 とトリガー t0 が実行されて S1 に遷移する際には，全て対応関係にある要素を含むため生起が不可能になる箇所は存在しない．しかし，S1 から t2 を生起させて S2 に遷移する際に，S1 の時の変数 x の値は 0 であるため，正答例はガードの条件 [x ＞=1] を満たさず遷移することができない．一方で，答案は同じ t2 のトリガーによって S'1 から S'2 に遷移することができているため，ガードの範囲が誤っており,「変数誤り」と判定される．

図 5 は,「誤った答案から生成されないトレース」の「d0，t0，d1，t2，-，...」に対して，このトレースが生成されなかった答案を，開始疑似状態から順に照合し，どの要素で生起不能な差異が初めて生じるか検証した結果である．まず，答案に関して，開始疑似状態 d0 とトリガー t0 が実行されて S'1 に遷移する際には，全て対応関係にある要素を含むため生起が不可能になる箇所は存在しない．しかし，S'1 の時の変数 x の値は 0 であるため，正答例はガードの条件 [x ＞=1] を満たさず遷移することができないが，答案はガード [x ＞=0] で条件を満たすため，S'1 から t2 を生

起させて S'2 に遷移でき，ガードの範囲が誤っており,「変数誤り」と判定される．

上記のように，各トレースに対して同じ要素を同じ順番で生起可能か検証して誤り箇所を特定する．なお，今回のように同じ誤り箇所から同じ種類の誤りが出力された場合は重複を排除して特定結果を出力する．そのため，図 4，図 5 の全トレースの出力結果は,「図 2 の t2[x ＞=0] のガードの範囲が誤り」となる．

4 評価実験

本章では提案手法の評価実験の内容について説明を行う．

4.1 実験目的

本実験では，提案手法の有効性を評価するために，静的解析手法の評価実験 [10] で誤判定されてしまった答案に対して，正しい判定結果が得られるかについて検証した．誤判定に対応することで，動的解析が変数や異なる対応関係を含むステートマシン図への適用範囲の拡大の見込みを示す．さらに，実験結果をもとに，提案手法で特定できない誤りや実行時間について考察し，今後の改善点と展望について示す．

4.2 実験概要

本実験では，我々の先行研究 [10] で示した 5 種類の要求文を大学の学部生 19 名に課して得られた 95 個の答案を評価の対象とした．5 種類の要求文は状態数が 4〜8 個で構成されるステートマシン図を記述する課題であった．先行研究の静的解析のみの評価実験では，95 個中 12 個の答案が誤判定されていた．この誤判定された答案のうち，12 個中 3 個の答案は変数と多対 1 の異なる対応関係を含む振る舞いが正しい別解であり，12 個中 9 個の答案は状態が全て 1 対 1 で対応した変数誤りが含まれている答案であることが，手動評価ではわかっている．なお，静的解析では変数と異なる対応関係に対応していなかったため，変数が完全に一致しない箇所や，多対 1 の対応関係の答案で誤判定されていた．

本稿で実験の対象とするチャイルドロック付きのファンヒーターの要求文の一例を以下に示す．

要求 1　電源を入れると温風が送風され，電源を切ると送風を停止する．
要求 2　チャイルドロック中は電源を切ることしかできない．
要求 3　電源の状態によらず，チャイルドロックをかけたり，解除できたりする．
要求 4　ファンヒーターは最初，電源が切れ，チャイルドロックもかかっていない．

課題では上記の要求文の他に，用語集において，実行活動は「送風を停止する」,「温風を送風する」，トリガーは「電源を切る」,「電源を入れる」,「チャイルドロックを解除する」,「チャイルドロックをかける」,「」（空値）が定義されている．なお，トリガーの空値は，開始疑似状態からの遷移等，トリガーを明に与えられない遷移に用いる．また，生起数は 5 で設定し，トレースを生成した．

4.3 実験結果

図 6 の正答例，図 7 の答案をもとに提案手法の適用結果について説明する．図 7 は，正答例と答案間で状態が多対 1 の異なる対応関係で変数を含み，正しい振る舞いの別解のステートマシン図である．先行研究の静的解析に，図 6，図 7 を入力として与えると，図 7 は，正答例に対して状態が 1 対 1 で対応しておらず異なる対応関係を含むため，状態の過剰や接続遷移の過剰等の誤りが含まれると誤判定されていた．しかし，提案手法に図 6，図 7 を入力として与えると,「誤った答案のみで生成されるトレース」,「誤った答案から生成されないトレース」は生成されておらず，誤り箇所は出力されなかった．そのため，動的解析を用いることで，記述上の違いに依存することなく，変数や異なる対応関係を含むステートマシン図の別解に対して正しい判定結果が得られた．

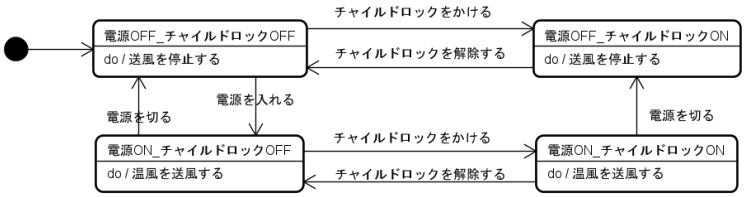


図 6　ファンヒーターの正答例

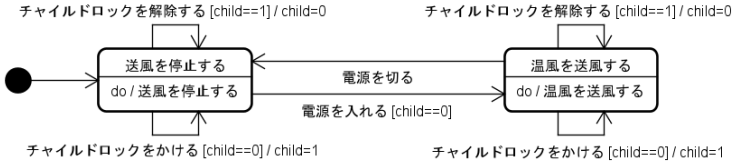


図 7　変数と異なる対応関係を含む答案

さらに，静的解析は変数に対応していなかったため，12 個中 9 個の変数誤りを含む答案に対して誤判定していたが，動的解析では，3.3 節で示した図 1 と図 2 の例と同様の手順で誤り特定が行われていた．例えば，ガードの範囲に関する変数誤りでは，図 4，図 5 に示すように，開始状態から順に各状態における変数の値を変化させていき，生起が不可能になった状態と，その際の変数の値が特定されていた．このように，動的解析では，ガードの範囲がどの状態のどの変数の値で誤っているのか等の，変数誤りを含む場合においても誤り箇所が正しく特定されていた．

また，図 6 の正答例と，図 7 を含む 19 個の答案を入力として生起数を変化させた際の動的解析と静的解析の誤り特定にかかる時間を計測した．結果として，生起数が 5 の時に 1 分 30 秒，生起数が 6 の時に 1 分 45 秒，生起数が 7 の時に 2 分 5 秒，生起数が 8 の時に 2 分 10 秒となった．なお，上記の時間に SML4C を用いた実行トレースの生成にかかる時間は含まれていない．

4.4　考察

4.3 節の実験結果から，提案手法が変数や異なる対応関係を含むステートマシン図においても有効であり，誤り特定を自動化できる見込みを得た．先行研究の静的解析手法では，正答例と答案間の各状態が 1 対 1 で対応していることが適用の前提であったが，動的解析と組み合わせることで，変数や異なる対応関係を含む答案にも適用可能であることが確認できた．また，新たに定義した変数誤りに関して，実行を伴うトレースを用いることで，ガードの条件の誤り等でどの変数の値で誤りが起きたか特定できていた．提案手法の発展により，変数誤りに新たに対応することで，より幅広いステートマシン図課題への適用が可能となった．さらに，教育上の観点として，変数を用いたステートマシン図には別解や異なる対応関係を含む答案が多く存在しうるため，このような答案に対して教育者の手作業による評価ミスを防止することが期待される．

一方で，今回の評価実験を通して新たに判明した問題点について述べる．まず，本稿で使用した答案以外のパターンとして，変数と異なる対応関係を含む誤ったステートマシン図が考えられる．本実験で扱った学習者の答案のうち，変数と異なる対応関係を含む 12 個中 3 個はいずれも正しい振る舞いの別解であった．しかし，実際に提出される答案では，正答例の要素と異なる対応関係で変数を含み，振る舞いが誤っている場合が想定されるため，今後の評価実験では，異なる対応関係を含む

誤った答案を用いた評価を行う必要がある．

次に，トレースを生成する際の生起数の定義に関する問題がある．今回の実験に入力したステートマシン図課題では終了状態がなく，状態間の遷移で生起数に制限がないため，生起数をどの値に設定しても全ての振る舞いを網羅しているとは言えない．そのため，適切な生起数の見極めについては，今後の課題とする．

5 まとめ

本稿では，静的解析手法と動的解析手法を組み合わせることによる，ステートマシン図答案の誤り特定自動化手法を提案した．評価実験の結果として，提案手法を用いることで，先行研究の静的解析手法だけでは特定できなかった変数や異なる対応関係を含むステートマシン図に対応できた．そのため，静的解析手法と動的解析手法を組み合わせることによって，静的解析だけでは特定できなかった誤りに対応し，誤り箇所の自動特定の適用範囲を拡大できる見込みを得た．

今後の課題として，4.4 章で述べた通り，課題によってトレースを生成する際に必要な生起数は異なるため，課題によって必要な生起数の値について定義する必要がある．また，ステートマシン図課題の誤りは多種多様であり，今回入力した 5 種類の課題には含まれていない誤りの種類が存在する可能性がある．そのため，今後は様々な課題に対するステートマシン図を入力として与えることによって，提案手法をより幅広い課題に適応した手法に改善することを目指す．

参考文献

[1] Object Management Group. Omg standards development organization, (閲覧日: 2022.06.24). https://www.omg.org/.

[2] Object Management Group. Omg unified modeling language version 2.5.1, (閲覧日: 2022.06.24). https://www.omg.org/spec/UML/2.5.1/PDF.

[3] I. Graja, S. Kallel, N. Guermouche, S. Cheikhrouhou, and A. Hadj Kacem. A comprehensive survey on modeling of cyber-physical systems. *Concurrency and Computation: Practice and Experience*, Vol. 32, No. 15, p. e4850, 2020.

[4] L.T.W. Agner, I.W. Soares, P.C. Stadzisz, and J.M. Simao. A brazilian survey on uml and model-driven practices for embedded software development. *Journal of systems and software*, Vol. 86, No. 4, pp. 997–1005, 2013.

[5] 文部科学省. 【技術・家庭編】中学校学習指導要領（平成 29 年告示）解説, (閲覧日: 2022.06.24). https://www.mext.go.jp/a_menu/shotou/new-cs/1387016.htm.

[6] S. Hara, M. Kayama, N. Nakano, T. Nagai, and N. Taguchi. A uml programming environment for ict related subject at junior high school. In *Proceedings of the 2019 The 3rd International Conference on Digital Technology in Education*, pp. 141–146, 2019.

[7] S. Sendall and W. Kozaczynski. Model transformation: the heart and soul of model-driven software development. *IEEE Software*, Vol. 20, No. 5, pp. 42–45, 2003.

[8] P. Pourali and J.M. Atlee. An empirical investigation to understand the difficulties and challenges of software modellers when using modelling tools. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pp. 224–234, 2018.

[9] 香山瑞恵, 小形真平, 増元健人, 伊東一典, 橋本昌己, 大谷真. 状態遷移図作成に際する初学者の誤り分析とそれに基づく教育方法の検討. 情報処理学会研究報告コンピュータと教育（CE）, 第 2012-CE-117 巻, pp. 1–9, 2012.

[10] 五島光祥, 小形真平, 槇原絵里奈, 岡野浩三. 静的解析による UML ステートマシン図答案の誤り特定自動化手法の評価. 第 8 回実践的 IT 教育シンポジウム (rePiT2022), pp. 42–49, 2022.

[11] S. Ogata and M. Kayama. Sml4c: Fully automatic classification of state machine models for model inspection in education. In *Proceedings of the ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion*, pp. 720–729, 2019.

[12] 五島光祥, 小形真平, 槇原絵里奈, 岡野浩三. 静的解析による UML ステートマシン図答案の誤り特定自動化手法の提案. ソフトウェアエンジニアリングシンポジウム 2021, pp. 67–75, 2021.

プログラム理解パターン抽出のための構文木と視線移動の自動マッピング手法

Automatic Mapping of Syntax Trees and Eye Movement for Program Comprehension Pattern Extraction

吉岡 春彦 * 上野 秀剛 †

あらまし ソフトウェア開発者がソースコードを理解する過程を明らかにすることは，開発作業や学習の効果・効率を改善するために重要な研究である．これまでに多数の研究が効率的な読み方を分析するために，ソースコードに対する開発者の視線移動を計測している．効率的に読む方法を理解するために，研究者はディスプレイの座標として記録された視線移動の情報からソースコードのどこを読んでいるか対応づける必要がある．しかし，エディタ上でのスクロールやウィンドウの移動などのためディスプレイ上の座標をソースコード上の単語と対応づけることは難しい．また，異なるソースコードに対する同じ意味の理解行動を抽出するためには，制御フローやフォーマット，識別子の違いを考慮する必要があるため分析には時間がかかる．本論文では，視線移動をディスプレイ上の座標から，注視した単語と対応する構文情報に変換する手法を提案する．

1 はじめに

ソフトウェア工学の分野の 1 つに，プログラムを読んでいる間の理解過程を対象としたプログラム理解の研究がある．よりよい理解手法を明らかにすることは開発者に効率的な読み方を教え，実装やデバッグの効率を改善する事で開発コストの削減に寄与する．これまでに多数の研究がプログラム理解の分析を目的にソースコードを読む過程を計測している [1] [2] [3] [4]．そのような研究の一部は，ディスプレイに対する視線移動を視線計測装置によって計測し，注視箇所の傾向や視線移動のパターンを分析している．視線計測装置が計測する視線移動は，ディスプレイ上の座標の時系列情報として記録されるため，複数の先行研究がソースコードを読んでいるときの座標単位の視線移動に基づいて開発者の理解過程を分析している [1] [2] [3] [4]．例えば，Hauser ら [1] と Busjahn ら [5] は熟練者が初心者よりも視線を上下に移動する傾向にあることを発見している．開発者がソースコードを読む際の視線移動を異なる開発者が見ることでバグ発見効率が上がることを確認した研究もあり [6]，視線移動から有用な情報を抽出し，優れた開発者が持つ理解パターンや理解戦略を開発者に提示することは，開発効率の向上や開発者の教育に有用である．

しかし，ディスプレイ上の座標に基づいた視線移動の分析は，座標とソースコードの対応をとる必要がある．表 1 に視線計測装置によって記録されたデータの例を示す．表の各行はある時刻に作業者が注視していたディスプレイ上の座標を表す．視線移動は視線計測装置の計測周期によって 1 秒間に 30〜300 点が計測され，分析の際には一定の範囲に一定時間以上留まった視線を停留としてまとめることが多い．ソースコードを対象とした視線移動の場合，ソースコードはエディタや IDE（統合開発環境）に表示される．そのため，ウィンドウ位置の移動やソースコードのスクロール，タブの切り替えなどによって同じ座標に表示されるソースコードは変化する．

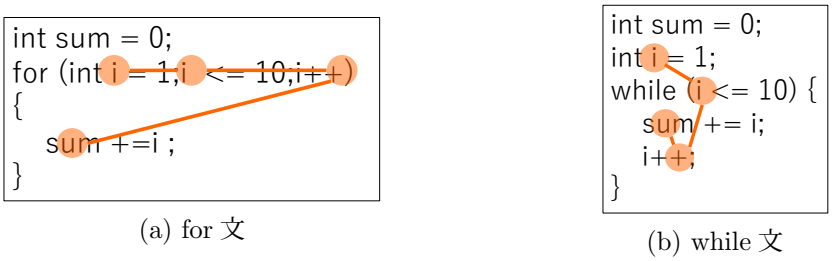
一部の研究は実験用プログラムや IDE のプラグインによって視線移動と操作履歴を組み合わせ，視線移動をソースコード上の行と列に変換している [7] [8]．これらの手法を用いることで同一のソースコードに対する，異なる被験者の視線を比較し，特徴を抽出することができる．一方で，異なるソースコードの場合，制御フローや

*Haruhiko Yoshioka, 奈良工業高等専門学校 システム創成工学専攻 情報システムコース

†Hidetake Uwano, 奈良工業高等専門学校 情報工学科

表 1: ディスプレイにおける座標単位の視線移動

経過時間	X 座標	Y 座標
24:54.1	322	457
24:54.1	322	458
24:54.2	328	463
24:54.2	326	469
24:54.2	320	479



(a) for 文　　(b) while 文

図 1: 2 つのソースコードに対する視線移動

表 2: 構文情報を用いた視線移動の出力例

ID	経過時間	視線移動
1	24:54.1	main メソッド / for 文 / 初期化式 / i
2	24:54.1	main メソッド / for 文 / 条件式 / i
3	24:54.2	main メソッド / for 文 / 変化式 / ++
4	24:54.2	main メソッド / for 文 / ブロック / sum

フォーマットが異なるため，2 つの視線が同一の処理内容に対する遷移であっても異なる座標や行・列番号として記録される．図 1 に同じ制御フローを持つが構文が異なる，2 つのソースコードと，それぞれに対する視線移動の例を示す．図の円と線は視線移動の連続を意味し，それぞれの円は停留を示す．2 つのソースコードは，いずれも 1 から 10 までの和を計算するが，(a) は for 文を，(b) は while 文を用いている．また，2 つの視線移動はいずれも同じ処理（インデックス変数の初期化，条件式，インデックスの増加，結果の計算）を同じ順序に見ている．しかし，2 つの文は異なる構造を持つため，座標単位の視線移動は for 文に対して「左から右へ」見ているものとして，while 文に対して「上から下へ」見ているものとして記録される．複数のソースコードを対象とした視線移動から，より一般的なパターンを抽出することはプログラム理解効率の向上に有用な知見を得るためにも重要であるが，異なる視線移動を手作業で解釈するためには多くの時間を必要とする．

本論文は視線計測装置が出力する座標単位の視線移動を，ソースコードから作成される構文木のノードに対する遷移に変換する手法を提案する．提案手法はレビュー対象のソースコードを構文解析し，行列番号と対応する構文木のノードを割り出す．その後，視線計測装置が出力した座標単位の視線移動を，ソースコード中の行・列番号に変換し，構文木のノードと対応づける．提案手法が出力する視線移動は，視線が停留した単語に対応するノードとその親ノード全ての情報を含む．

表 2 に図 1(a) の視線移動をソースコードの構文情報に基づいて変換した場合の出力例を示す．表の各行は視線の停留した単語に対応するノードの情報を表す．例え

ば ID1 の視線は for 文の初期化式（int i = 1）に対する視線を表す．視線移動の列は視線が停留した単語を表すノードとその親ノードを表し，ID1 の視線が main メソッド内にある for 文の初期化式の i を見ていることを示している．提案手法は構文木のノードに対する遷移として視線移動を表現することで，ソースコードの表示位置やフォーマットによる違いを取り除く．また，視線が停留した単語が属するブロックやメソッド，クラスの情報を含む情報として出力するため，分析目的に応じて異なる粒度で視線移動のパターンを抽出することを可能にする．

2 関連研究

視線移動の計測は初心者と熟練プログラマの違いを分析するためによく用いられる．一定の時間，一定の範囲内に視線が留まる状態を停留 (fixation) と呼び，その中心座標を停留点 (fixation point) と呼ぶ．連続した停留点は読み手が興味を持つ場所の時間変化を表すとされる．Hauser らは初心者と熟練者の間における停留点に基づいた視線移動の違いを比較した [1]．結果，熟練者グループは非線形にソースコードを読む傾向にあるが，初心者は線形に読むことが分かった．

視線移動の分析に用いられる別の定義として Area of Interest (AOI) がある．研究者は着目の対象である画像やソースコードに矩形，または円形の領域を AOI として定義する．AOI はその範囲内を注視した場合，同一の物を見ていると見なされる領域である．同じ AOI に連続した視線が見られた場合，その AOI に対する長時間の注視として視線移動を要約する．例えばソースコードを対象とする場合，各メソッドに AOI を定義することでメソッド間の視線移動の移り変わりを分析する．複数のプログラム理解に関する研究がソースコードの字句に AOI を定義することで開発者の理解パターンを分析している．Rodeghero らはメソッド宣言，メソッドの呼び出し，制御フロー，その他の 4 種類の AOI をソースコード上に定義し，それぞれに対する注視時間の長さを分析した [2]．分析から熟練の Java プログラマはメソッド呼び出しと制御フローをメソッド宣言よりも長い時間見ていることが分かった．Crosby らは初心者と熟練者の間における，各 AOI に対するレビュー時間の配分を分析した [3]．結果，熟練者は初心者よりもコメントを見る時間が短いことが分かった．ソースコードを対象とした研究においては，ソースコードの各行を AOI とすることで，行単位の視線移動を分析している．Peitek らは初心者と中級者の読み方を比較するために，座標単位の視線データを行単位に変換した [7]．結果，中級者は初心者より，より頻繁にソースコードの上の行に視線移動をすることが分かった．

プログラム理解を対象とした視線計測を行っている先行研究の多くは，ソースコードを画像としてディスプレイ上に表示し，停留点がソースコード内のどの単語に相当するか，手作業で抽出するか，各単語・各行に対して AOI を定義することで分析している．そのため，1 画面内に収まらないソースコードや複数のソースコードを対象とした分析はわずかしか行われていない．一部の研究では，実験用のプログラムを作成し，スクロールや複数のファイルを対象とした分析が行われている．オープンソースソフトウェアの 1 つである iTrace は，座標単位の視線情報を，自動的にソースコードの行と列番号に変換する [1]．iTrace は Eclipse のプラグインとして実装されており，iTrace を用いた視線移動の分析も行われている [8] [9] [10]．

3 提案手法

提案手法は視線計測装置が出力する座標単位の視線移動をソースコードから生成される構文木のノードに対する遷移に変換する．図 2 に提案手法の処理概要を示す．四角はシステムを構成するモジュールを表し，細い矢印は情報の流れを表す．ソースコードを読んでいる被験者の視線移動は視線計測装置によって計測される．視線計測装置は各時点における注視点をディスプレイ上の座標（例えば，X:121, Y:313）と

[1] https://www.i-trace.org

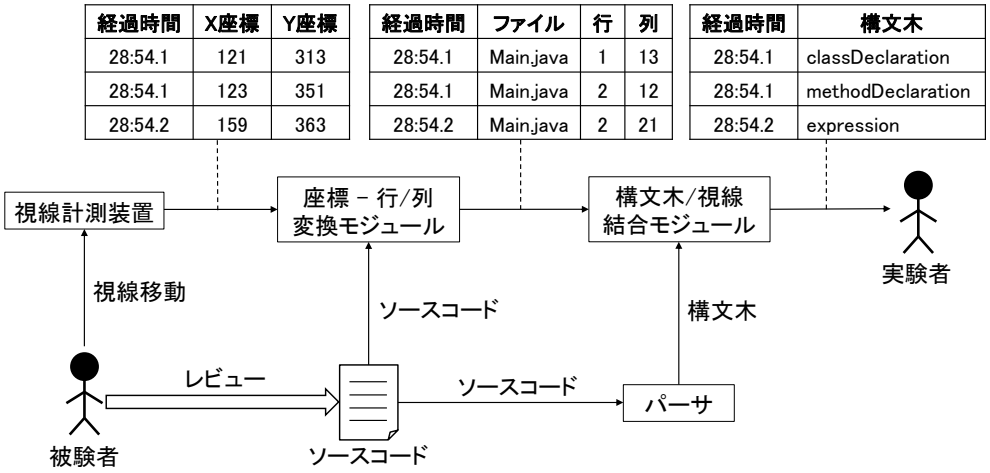


図 2: 提案手法

```
1 public class Main {
2     public static void main(String[] args) {
3         int sum = 0;
4         for (int i = 1; i <= 10; i++) {
5             sum+=i;
6         }
7         System.out.println(sum);
8     }
9 }
```

図 3: Main.java

して時系列に出力する．座標 - 行/列変換モジュールは座標単位の視線移動とソースコードを入力として受け取り，ソースコード名と行・列番号 (Main.java, 行:1, 列:13) として視線を出力する．このとき，行・列番号からソースコードの単語または文字を抽出し，構文解析で得られた構文木上のノードと対応をとる．また，同じ単語，文字に対する連続した注視は 1 つにまとめられる．

ソースコードはパーサにも送られ，構文木が生成される．構文木/視線結合モジュールは構文木と，行・列単位の視線移動を受け取り，構文ノード単位の視線移動を出力する．パーサが出力する構文解析の結果には，ソースコード中の各単語の位置を表す行・列番号と，文字数，およびその単語の構文上の型が含まれる．構文木/視線結合モジュールは行・列番号に変換された視線移動を構文解析結果の各ノードが持つ行・列番号と対応付けすることで，視線移動を構文木上のノード単位に変換する．

図 3 のソースコードから生成された構文木の例を図 4 に示す．紙面の都合，main メソッドに対応するノード (main method @9) 以下のみを示し，一部の中間ノードを削除している．図 4 において，葉ノードはソースコード上の単語を表し，内側のノードは子ノードの属する構文上の要素を表す．例えば，右側にある内側のノード (block @19) は，図 3 の 5 行目にある 3 つの単語 (sum, +=, i) から構成されるブロックを表し，4 行目の for 文 (statement @13) の一部である．提案手法は視線移動を停留点上にある単語と，行・列番号，および単語が属する構文上の要素全てを組み合わせた情報に変換する．より上位の構文要素の情報を含むことで，例えば，図 3 の 4 行目 30 文字目に対する視線を，以下の異なる粒度で捉えることができる．

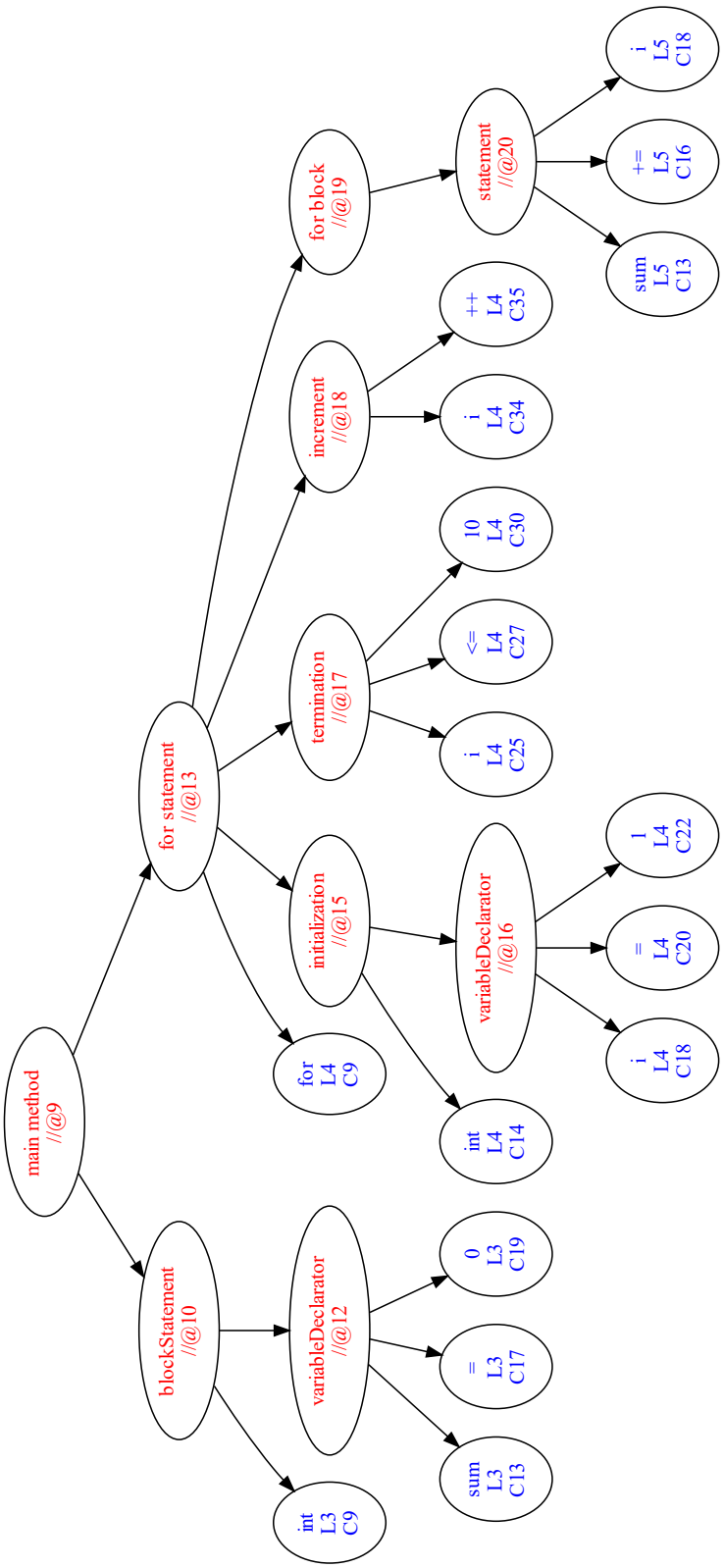

main method //@9
blockStatement //@10
int L3 C9
variableDeclarator //@12
sum L3 C13
= L3 C17
0 L3 C19
for statement //@13
for L4 C9
initialization //@15
int L4 C14
variableDeclarator //@16
i L4 C18
= L4 C20
1 L4 C22
termination //@17
i L4 C25
<= L4 C27
10 L4 C30
increment //@18
i L4 C34
++ L4 C35
for block //@19
statement //@20
sum L5 C13
+= L5 C16
i L5 C18

図 4: 構文木の例

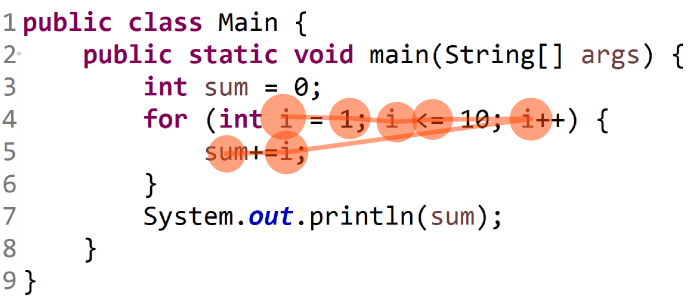


図 5: Main.java に対する視線移動

- 単語 “10” に対する視線
- for 文の条件式に対する視線
- for ブロックに対する視線
- main メソッドに対する視線
- Main クラスに対する視線

提案手法の出力は，分析の目的に応じた粒度（例えばメソッド単位）による視線の要約を最小限の変換処理で可能にする．また，1 つの単語に対する異なる粒度の情報をベクトルとして表現することで，単語が持つ構文上の意味に対する視線の遷移として特徴分析やパターンマイニングが行えると考えられる．

提案手法を Python により実装した．座標単位の視線移動から行・列番号を抽出する “座標 - 行/列変換モジュール” の実装として iTrace を用いた．“構文木/視線結合モジュール” の実装にはオープンソースの parser generator である，ANTLR を使った [2][3]．本論文では対象言語として Java を選択したが，Java 以外の言語を対象とするパーサを用いることで他のプログラミング言語に対する視線移動の分析が可能である．また，分析を補助するためソースコード上に視線移動のデータを可視化する機能を実装した．次章に実装した提案手法を用いた出力例と分析例を示す．

4 分析例

提案手法が開発者の視線移動から理解パターンや理解戦略を抽出するために有用であるか分析例を用いて定性的に評価する．

図 5 に Main.java（図 3）に対する視線移動の一部を実装したシステムによって可視化した結果を示す．図の丸は視線の停留点を示し，丸を結ぶ直線は連続する停留点を示す．視線は 4 行目にある for 文内の i の宣言から条件式，i の増加，sum への追加を順に読んでいることを示す．9 行のソースコードに対する 15.2 秒の視線データを可視化するのに必要な処理時間は CPU が i5-4210U@1.70GHz の PC で 9.9 秒，構文情報に基づいた視線データの出力は 2.4 秒だった．

表 3 に提案手法の出力例を示す．各行は図 5 で可視化された停留点に対応する視線を表す．トークンは停留点上に表示された単語を表し，構文木はトークンに対応する構文木ノードと根ノードの間にある全てのノードの ID を連結した文字列を表す．各ノードの ID は図 4 に対応している．可視化された視線移動を見ると（図 5），4 行目にある for 文の初期化式，条件式，変化式，ブロック内での演算を順に見ていることが容易に理解できる．しかし，表 3 の行・列番号やトークンからは被験者が何を読んでいるかソースコードなしで理解するのは難しい．一方，提案手法により出力された構文木を見ると，一連の視線移動がすべて for 文 (for statement @13) に集中しており，また，初期化式 (initialization @15)，条件式 (termination @17)，変

[2]https://www.antlr.org

[3]https://github.com/antlr/grammars-v4/tree/master/java/java

表 3: 提案手法の出力

ID	時間	停留点	行	列	トークン	構文木
1	02:15	705, 226	4	18	i	block@9/ for statement@13/ initialization@15/ var.Dec.@16/ i
2	02:16	800, 235	4	22	1	block@9/ for statement@13/ initialization@15/ var.Dec.@16/ 1
3	02:18	877, 234	4	25	i	block@9/ for statement@13/ termination@17/ i
4	02:19	928, 228	4	27	<=	block@9/ for statement@13/ termination@17/ <=
5	02:21	1076, 237	4	34	i	block@9/ for statement@13/ increment@18/ i
6	02:23	708, 283	5	18	i	block@9/ for statement@13/ for block@19/ statement@20/ i
7	02:24	618, 282	5	13	sum	block@9/ for statement@13/ for block@19/ statement@20/ sum

化式 (increment @18)，ブロック内 (for block @19) を順に見ていることが示されている．従来の視線移動の表現方法と比較して，提案手法は以下の利点がある．

- **構文上の文を単位とした分析が容易**
  従来の行・列番号で表される視線移動は単語を単位とした表現である．テキスト上の行を単位とした分析 [7] も行われているが，複数の文が含まれてる場合や，複数行で 1 文を構成することもある．そのため，視線が停留した行がどのような内容か研究者が読み取る必要がある．
  一方で提案手法は構文上の “文” に対する一連の視線として解釈可能な文字列を出力する．図 5 で可視化した視線移動を構文木上に表現した結果を図 6 に示す．細い点線は視線移動が停留したトークンに対応するノードの順序を示す．太い点線はトークン単位の視線移動を親ノードを単位とした形で要約した視線移動を示す．提案手法は構文上の親ノードと視線を対応づけることで，構文上の “文” に対する視線として扱うことができる．太い点線が示す視線移動は初期化式，条件式，変化式，ブロック内を順に見ていることが容易に理解できる．プログラムの基本的な単位である文を単位とした視線移動はその解釈が容易であると考えられる．また，提案手法は表 3 に示したように，視線が停留したトークンが属する文が文字列で出力されるため，パターンマイニングの手法を用いることで理解パターンや理解戦略の抽出が可能である．

- **異なる抽象度による視線移動の分析が可能**
  視線を文単位で要約する事と同様に，より上位の構文要素で視線を要約することで，ブロックやメソッド，クラスを単位とした視線移動の生成が可能である．従来研究においては，実験者が分析対象とする粒度で AOI を設定することで同様の分析が可能であったが，座標情報や行・列番号で個別に定義する必要があることから規模の大きいソーコードを対象とした視線分析においては手間が大きい．また，実験で計測した結果に基づいた探索的な分析が困難である．
  提案手法は構文情報に基づいて，AOI の事前定義無しにプログラム理解時における理解単位を構成する文やブロック，メソッド，クラスと視線移動を関連付ける事ができる．そのため，従来分析が困難であった規模の大きなソースコードに対する，長時間にわたる視線移動に対しても分析が可能である．

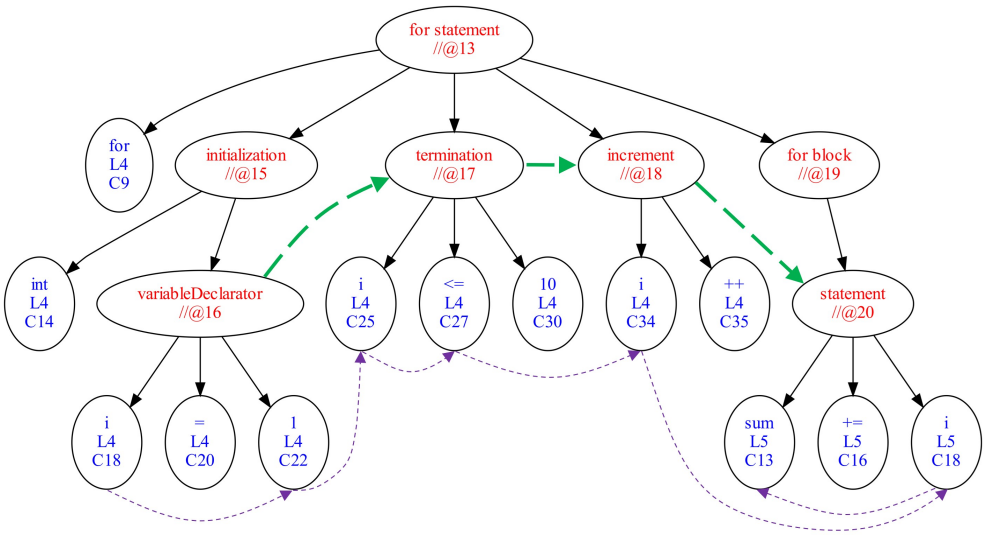


図 6: 構文木を使った視線移動の可視化

- **構文情報のベクトル化によるパターンマイニングとラベル予測**
  提案手法が出力する視線情報は，視線が停留した単語と，単語が属する構文要素（文やブロック，メソッド，クラス）の双方を含む．ある単語に対する視線は，単語が持つ異なる粒度における意味に対する注視として解釈が可能である．例えば，あるメソッド内の for 文で定義されている index 変数への注視は “変数名，型の理解”，“代入値の理解”，“index を用いているループの処理回数の理解”，“メソッド引数に対応する出力値の理解” など異なる粒度の理解過程の一部を構成している可能性がある．ある注視や連続した視線移動がどのような理解過程の一部か理解することは，従来，研究者による手動で行われてきた．提案手法が出力する構文情報から，特定の注視や視線移動を特徴付けるベクトルを生成することで，共通した理解を表すパターンのマイニングや機械学習によるラベル予測が可能になる．

- **異なるソースコードを対象とした分析**
  異なるソースコードの一部に同じ処理内容（例えば配列の合計を計算）が含まれる場合，同一の被験者が，または異なる被験者が各ソースコードに対して同じ読み方をする可能性がある．しかし，同じ処理であっても異なるフォーマットや変数名などを用いている場合，視線の停留位置を表す行・列番号や単語からパターンを抽出することは難しい．提案手法は構文情報を含む視線移動を出力するため，同じ構文要素を持つ単語に対する視線移動を識別子や行・文字数の影響を受けずに抽出することができる．これによって，異なるソースコードに対する視線移動から共通する理解パターンの機械的な抽出が可能になる．

上記の利点はいずれも従来手法では時間を要した視線移動の分析をより短い時間で実行する点で有用であり，開発者の視線移動から理解パターンや理解戦略を抽出することを容易にする．

5 おわりに

本論文で著者らはディスプレイの座標単位で記録される視線移動を，ソースコードから作成される構文木のノードに対する遷移に変換する手法を提案した．提案手法は視線が停留する行・列番号を求めた上で，ソースコードの構文解析結果と対応を取ることで視線が停留した単語の構文要素を視線情報に付与する．分析例を通じて，従来手法と比較した時の提案手法の利点を示した．

本研究の今後の課題として，提案手法を用いた視線移動の分析が挙げられる．4章で述べたように，異なるフォーマット，識別子で書かれているソースコードに対する視線移動から，同じ処理内容を読む様子を抽出することは従来手法では難しい．提案手法を用いて複数の視線移動から同じ処理内容を読むパターンが抽出できれば，視線移動の分析効率を高めることができる．視線パターンの分析によってプログラム理解の能力が高い開発者と低い開発者の特性を明らかにすることは初学者の教育に有用である．また，多くの開発者に広く見られるパターンが見つかれば，より開発効率を高めるための IDE やプログラミング言語の開発に有用な知見となる．

また，提案手法は AOI を定義せずに異なるソースコードに対する視線から構文情報に基づいたパターン抽出ができるため，従来難しかった多数の視線データを対象とした分析を可能にする．ソースコードを対象とした視線移動のオープンなデータセットが少数ながら公開されており [4]，今後，複数の研究によって計測された視線データに共通する理解パターンの分析を行うことは本研究の発展の 1 つである．

謝辞 本研究は JSPS 科研費 JP21K11842 の助成を受けたものです．

参考文献

[1] F. Hauser, S. Schreistter, R. Reuter, J. H. Mottok, H. Gruber, K. Holmqvist, and N. Schorr, “Code Reviews in C++: Preliminary Results from an Eye Tracking Study”, In Proc. Symposium on Eye Tracking Research and Applications (ETRA), pp. 1–5, 2020.

[2] P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch, and S. D’Mello, “Improving Automated Source Code Summarization via an Eye-Tracking Study of Programmers”, In Proc. 36th International Conference on Software Engineering (ICSE), pp. 390–401, 2014.

[3] M. E. Crosby and J. Stelovsky, “How Do We Read Algorithms? A Case Study”, Computer, Vol. 23, No. 1, pp. 24–35, 1990.

[4] I. Bertram, J. Hong, Y. Huang, W. Weimer, and Z. Sharafi, “Trustworthiness Perceptions in Code Review: An Eye-Tracking Study”, In Proc. 14th International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–6, 2020.

[5] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm, “Eye Movements in Code Reading: Relaxing the Linear Order”, In Proc. 23rd International Conference on Program Comprehension (ICPC), pp. 255–265, 2015.

[6] R. Stein, and S. E. Brennan, “Another Person’s Gaze as a Cue in Solving Programming Problems”, In Proc. 6th International Conference on Multimodal Interface (ICMI), pp. 9-15, (2004).

[7] N. Peitek, J. Siegmund, and S. Apel, “What Drives the Reading Order of Programmers? An Eye Tracking Study”, In Proc. 28th International Conference on Program Comprehension (ICPC), pp. 342–353, 2020.

[8] D. T. Guarnera, C. A. Bryant, A. Mishra, J. I. Maletic, and B. Sharif, “itrace: Eye Tracking Infrastructure for Development Environments”, In Proc. Symposium on Eye Tracking Research and Applications (ETRA), pp. 1–3, 2018.

[9] A. Abbad-Andaloussi, T. Sorg and B. Weber, “Estimating Developers’ Cognitive Load at a Fine-Grained Level Using Eye-Tracking Measures”, In Proc. 30th International Conference on Program Comprehension (ICPC), pp. 111-121, 2022.

[10] B. Sharif and N. Mansoor, “Humans in Empirical Software Engineering Studies: An Experience Report”, In Proc. International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 1286-1292, 2022.

[4] http://www.emipws.org/corrected-dataset/

オンラインジャッジシステムの問題選択支援に向けた問題文間の意味的類似度の算出

Calculation of Semantic Similarity between Problem Sentences for Supporting Problem Selection in Online Judging System

新濱 遼大 * 槇原 絵里奈 † 小野 景子 ‡ 大正 歩夢 §

あらまし オンラインジャッジシステムは多種多様な問題が収録されているため，プログラミング教育に導入することでプログラミング言語やアルゴリズム理解を深めることが可能になる．しかし，学習者にとって多種多様な問題から最適な問題を選択することは容易でない．そこで，学習者の問題選択支援として問題文間の意味的類似度算出手法を提案する．本研究では，オンラインジャッジシステムの問題文から構成されたデータセットを作成し，word2vec，glove，doc2vec，sent2vec でモデルを構築する．そして，コサイン類似度を用いて意味的類似度を算出した．提案手法を評価するために，問題文のペアに対して手動で類似度のスコアを付与した評価データを作成した．評価データのスコアと提案手法より算出したスコアのピアソンの相関係数を求めた結果，2 つのスコアに強い相関があることを確認した．

1 はじめに

IT 産業の拡大によりプログラミング学習者が増加傾向にあり，初学者を対象とした講義や学習サイトが増加している．本研究では多様化する学習環境のうち，オンラインジャッジシステム（Online Judge System，以下 OJS）に注目する．OJS とはオンライン上でソースコードを送信すると自動でコンパイル・テスト・採点が行われ，結果を閲覧できるシステムである．OJS は他の学習者が提出したソースコードを閲覧可能である．すなわち，解法がわからない場合に他者のソースコードを参照することで解法の理解に繋がると考える．また，OJS に掲載される問題は難易度や解法に用いるアルゴリズムが多種多様である．以上の理由から，学習者は OJS を利用することでプログラミング言語やアルゴリズムの理解を深める自学自習が可能になると考える．

しかしながら，OJS の問題には明確な回答が存在しないため，問題同士が似ているかどうかを判断し問題を選択することは容易でない．一部の OJS にはタグや難易度によって絞り込みが可能であるが，例えば Codeforces において難易度が 1000 でタグが “implementation” は 100 問以上存在する．つまり，多種多様な問題が存在する OJS から学習者が自身の理解度に応じた問題を選択することは容易でない．学習者が適切な問題を選択できない場合，OJS の利用やプログラミング学習そのものに対するモチベーションが著しく低下する可能性がある．上記の問題に取り組むため，我々は先行研究において問題の解答履歴と正誤判定に基づいた遷移モデルを作成することで問題間の関係性を求めた [1]．しかし，先行研究では解答履歴がない問題や解答者が少ない高難易度の問題を扱うことが難しい．以上より，本研究では既存ユーザの解答数に依存しない問題文間の意味的類似度を算出することで，学習者の問題選択支援を目指す．意味的類似度とは，問題文の意味に基づく類似度であり，ユーザに依存しないデータである．そのため，解答したユーザが少ない高難易度の

*Ryota Shinhama, 同志社大学大学院理工学研究科

†Erina Makihara, 同志社大学理工学部

‡Keiko Ono, 同志社大学理工学部

§Ayumu Taisho, 同志社大学大学院理工学研究科

問題や新規に追加された問題において利用することが可能になると考える．さらに，ユーザの書き方に依存するソースコードでは，例えば同じアルゴリズムを使っていても for 文を用いた場合と関数やメソッドを用いた場合で違う問題として認識されるが，問題文はユーザに関係なく扱うことができる．また，2 つの問題文から “条件分岐” や “並び替え” を読み取れ，類似していると判断できる場合は，解答に必要な知識が共通するため学習者が次に取り組みやすいと考える．つまり，意味的類似度を利用することでユーザに対して解きやすい問題を推薦できるため，学習に対するモチベーション低下防止に繋がる可能性がある．以上より，問題文の意味的類似度は問題選択における新たな指標になりうると考える．そして，難易度やタグなど他の指標と組み合わせることで，問題選択が容易になると考える．

本研究では学習者の問題選択支援に向けた第一歩として，問題文の意味的類似度を求めるために適した手法やプログラミング問題の特徴について調査する．本研究の新規性はプログラミングの問題文からなる独自のデータセットを用いてモデルを構築し意味適類似度を算出したことと，被験者実験より評価データを作成しモデルを評価したことである．本研究では小規模な学習データからも文章ベクトルを獲得可能な word2vec [2]，Global Vectors for Word Representation（以下，glove) [3]，doc2vec [4]，sent2vec [5] を用いて問題文をベクトル化し，問題文間のコサイン類似度を意味的類似度とした．我々の先行研究では word2vec と doc2vec を用いて文ベクトルを獲得したが，評価指標を用いた評価を行っていない [6]．そこで，文章の意味的類似度を評価するタスクである Semantic Textual Similarity（以下，STS）を用いる．STS では人間がスコアリングした意味的類似度を付与したデータセットを利用し，文章間の類似度や翻訳などに応用されている [7] [8]．STS は文章の意味を考慮したタスクであるため，本研究に適していると考える．しかし，プログラミング問題を対象としたデータセットは存在しない．そこでまず OJS を対象に独自のデータセットを作成する．作成したデータセットのうち一部を評価データとし，評価データに対して被験者実験を実施することで意味的類似度を付与する．評価データに付与したスコアと提案手法より算出したスコアのピアソンの相関係数を求めることで提案手法を評価し，提案手法の有効性を示す．

2 Semantic Textual Similarity

STS とは 2 つの文章の意味的な同等性の程度を測定するタスクである [9]．2 つの文章を段階的に評価するため，機械翻訳や自動要約，質問応答など各自然言語処理のタスクに適用可能である．STS ではモデルの評価に，手動で 5 段階にスコアリングしたデータセットを用いることが一般的である．スコアリングは複数人により実施される．そして，モデル評価において全員のスコア平均とモデルの推定スコアのピアソンの相関係数を求める．相関の強さは Evans が提案したガイドラインにより，下記のように評価可能である [10].

- 非常に強い相関：0.80-1.00
- 強い相関：0.60-0.79
- 適度な相関：0.40-0.59
- 弱い相関：0.20-0.39
- 非常に弱い相関：0.00-0.19

STS において多くのモデルが提案されており，BERT の改良形である RoBERTa は事前学習と fine-tuning を用いることで高い精度を示している [11]．しかし，BERT 系のモデルは事前学習に膨大な量のデータや計算コストが必要であり，fine-tuning には文章間のスコアが十分な量必要のため，既存のデータセットを用いない場合は BERT 系のモデルを構築することが容易でない．そのため，既存のデータセットを用いない研究では小規模なデータセットから文章をベクトル化可能な word2vec や sent2vec を用いた手法を提案している [12] [13].

表 1: データセットに含まれる単語，数字，記号の割合

(a) 本研究で作成したデータセット

	個数	割合
単語	1562775	88.62%
数字	60120	3.41%
記号	140531	7.97%

(b) STS benchmark

	個数	割合
単語	174914	93.83%
数字	4104	2.20%
記号	7390	3.96%

3 データセット

プログラミング問題のみで構成されたデータセットが存在しないため，独自のデータセットを作成した．本研究では日本語ではなく英語で書かれた問題文を対象にすることでデータセットを十分に確保した．データセットの問題文は単語の小文字化と句読点を除去する前処理を行なった．データセットにしようした問題文は AtCoder[1]，AIZU ONLINE JUDGE[2]，Codeforces[3]から取得した．上記の OJS から 2022 年 5 月時点で取得可能な 11058 問からデータセットは構成されている．本研究で作成したデータセットと既存の STS Benchmark データセット [4]に含まれる単語，数字，記号の割合を表 1 に示す．表 1 より，プログラミング問題には通常のデータセットと比較して数字と単語が多く含まれることがわかる．

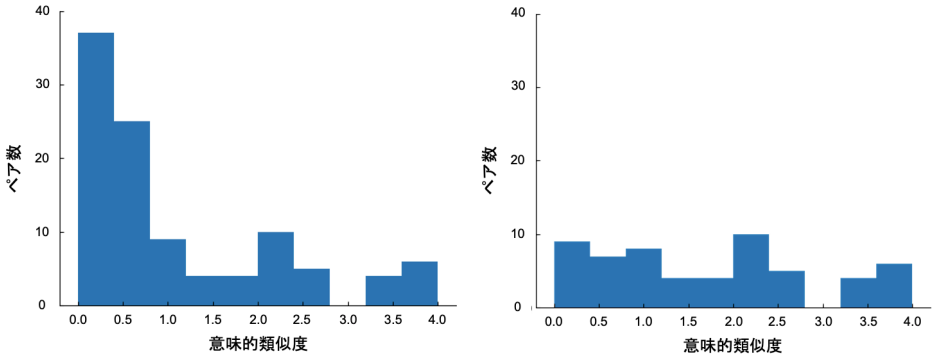
本研究では作成したデータセットを文ベクトルを獲得するためのデータと提案手法を評価するための評価データに分割し利用した．評価データは AtCoder の AtCoder Beginner Contest の A 問題，および B 問題を対象とした．また，長文に対しては問題文間の意味的類似度を正確に推定できない可能性があるため，文中の単語が 60 以内の問題のみを評価データに用いた．なお，評価データにおいては日本語と英語の両方の言語を AtCoder が提供している問題のみを対象とした．事前実験において英語で書かれた問題を使用した場合，被験者は翻訳機をそれぞれ使用して意味的類似度を回答していた．使用した翻訳機によって問題文の捉え方が異なる場合，正確に問題文の意味的類似度を測定できない可能性がある．そのため，本研究では AtCoder が英語と日本語で提供している問題のみを評価データに用いた．以上の条件を満たす 208 問から無作為に 104 ペアを作成することで，多種多様な問題を評価することが可能になる．そして，被験者実験より 104 ペアの意味的類似度を測定し，全被験者の平均を評価データに付与した．被験者実験は情報系学部に所属しており，さらに OJS の利用経験がある大学生と大学院生合わせて 5 名を対象に実施した．被験者には事前に意味的類似度の例を示し，その後に問題文間の意味的類似度を自作の web アプリを用いて測定した．測定した意味的類似度は 0（関係なし）から 4（類似性が高い）までのスコアを付与し，5 人のスコア平均を評価データに付与した．スコア付与後の意味的類似度の分布を図 1(a) に示す．図 1(a) より，評価データは 0 付近に偏っていることがわかる．データが 0 付近に偏っている場合，0 付近を多く出力する提案手法の精度が高くなり，提案手法を正確に評価することが困難である．そこで，本研究では提案手法を評価するために評価データからサンプリングを行い，57 ペアを評価データに用いる．サンプリング後のデータ分布を図 1(b) に示す．図 1(b) よりデータに偏りはなく，提案手法を正確に評価できると考える．

[1]https://atcoder.jp/

[2]https://judge.u-aizu.ac.jp/onlinejudge/

[3]https://codeforces.com/

[4]http://ixa2.si.ehu.eus/stswiki/index.php/STSbenchmark



(a) サンプリング前　　(b) サンプリング後

図 1: 評価データにおける意味的類似度の分布

4　提案手法

本研究では word2vec，glove，doc2vec，sent2vec を用いてモデルを構築することで文ベクトルを獲得する．複数の手法で文ベクトルを獲得することで，プログラミング問題文に適した独自のモデル構築に向けた調査を行う．単語ベクトルを獲得する手法である word2vec，glove は文中の単語ベクトルを平均化することで文ベクトルを獲得する．word2vec と glove は文中に含まれる全ての単語ベクトルを平均化するため，一般的な単語であるストップワードを除去した学習データを用いる．doc2vec と sent2vec は，機械学習より文ベクトルを直接獲得する．全ての手法に共通して，文ベクトルの獲得後にコサイン類似度を用いて意味的類似度を算出する．

既存研究では文中の数字や記号は全て除去することが多いが，本研究ではプログラミング問題を対象にするため文中に数字や数学記号，特別記号が多く含まれる．そのため，数字や記号は空白ごとに分割することで数字や記号を考慮した文ベクトルを獲得した．

5　評価実験

5.1　実験概要

提案手法を評価するために評価データに付与したスコアと提案手法より算出した意味的類似度のピアソンの相関係数を求める．評価データに付与したスコアの範囲は 0 から 4 のため，算出したコサイン類似度を 0 から 4 に正規化し実験を行なった．実験に使用する word2vec と doc2vec は gensim[5]を用いてモデルを構築し，glove と sent2vec は先行研究に示されているプログラムを基にモデルを構築した [3] [5]．表 2 に実験に使用したパラメータを示す．

評価データに異なる前処理を行うことで精度が変わる可能性があるため，以下の条件で評価データに前処理を行う．

- 前処理なし
- 数字除去
- 記号除去
- 数字と記号除去

前処理ごとに実験を行い，それぞれの結果を比較する．

[5]https://radimrehurek.com/gensim/

表 2: パラメータ設定

モデル	パラメータ	値
word2vec	epoch	50
	window	5
	vector_size	100
	sample	1e-3
	sg	1
	negative	20
	alpha	0.025
	min_count	5
glove	max_iter	20
	window_size	15
	vector_size	300
	min_count	1

モデル	パラメータ	値
doc2vec	epoch	500
	window	10
	vector_size	300
	sample	1e-4
	dm	1
	alpha	0.1
	min_count	5
sent2vec	epoch	20
	window	15
	dimention	300
	lr	0.75
	min_count	5
	wordNgram	2

表 3: 提案手法における前処理ごとのピアソンの相関係数

	前処理なし	数字除去	記号除去	数字と記号除去
word2vec	0.662	0.680	0.673	**0.685**
glove	0.670	**0.690**	0.661	0.670
doc2vec	0.607	0.606	**0.608**	0.605
sent2vec	0.714	0.706	**0.726**	0.707

5.2 結果と考察

提案手法における前処理ごとのピアソンの相関係数を表 3 に示す．表 3 より全ての手法で相関があることがわかる．なかでも sent2vec を用いた手法が精度が高いことがわかる．sent2vec は文章を分割し，n-gram の特徴ベクトルを求める．n-gram の特徴ベクトルは単語の文脈を考慮するため，正確に文ベクトルを獲得したと考える．また，特定の前処理によって word2vec と glove，sent2vec は精度が向上する一方で，doc2vec は前処理によって精度は変化しなかった．つまり，扱うアルゴリズムによって前処理を変化させることが重要とわかる．単語ベクトルを獲得する word2vec と glove は数字や記号を除去することで精度が向上した．word2vec と glove は文中に含まれる全ての単語ベクトルを平均化することで文ベクトルを獲得する．そのため，文中の単語数が減少したことで各文章の特徴が正確に表れたと考える．今後は特徴語の抽出や単語ベクトルへの重み付けを行うことで精度が向上すると考える．

被験者実験でスコアリングを行った評価データより，OJS には似ていない問題が多いことがわかる．提案手法を用いることで，学習者が次に選択する問題のうち似ていない多くの問題を排除することが可能となり，学習者の問題選択を支援する指標になりうると考える．

6 おわりに

本研究は多種多様な問題が掲載されている OJS において，問題選択における新たな指標として問題文間の意味的類似度を提案した．問題文間の意味的類似度を算出するために，OJS の問題文で構成された独自のデータセットを作成し word2vec，glove，doc2vec，sent2vec のモデルをそれぞれ構築した．そして，文章間の意味的類似度を測定するタスクである STS に着目し，STS と同様の評価手法を用いることで提案手法を評価した．提案手法を評価するためのデータを被験者実験を実施す

ることで作成した．しかし，評価データにおける意味的類似度の分布を確認したところ，OJS には似ている問題が少ないことがわかった．そこで，評価データからサンプリングを行なったデータを用いて提案手法を評価した．実験の結果，全ての手法で相関があり，特に sent2vec を用いた手法が高い精度を示した．本研究では，問題文間の意味的類似度をプログラミング問題文のみから構築したモデルから算出し評価することで，既存の指標とは異なる観点から問題選択支援を実現できる可能性を示した．問題文間の意味的類似度はユーザに依存しないため，解答したユーザが少ない問題や新規に追加された問題，他の OJS に掲載された問題でも利用可能である．また，OJS には似ている問題が少ないことから，類似していない問題を排除し，学習者の選択範囲を絞ることで問題選択における手間を削減することが可能になると考える．今後は問題文中に含まれるアルゴリズムなどの重要な単語に重み付けを行うことでより精度が向上すると考える．また，意味的類似度に難易度やユーザのレートなどを加えることで，似た問題文かつ難しい問題を選択するための支援を行うことが可能になると考える．

謝辞 本研究は JSPS 科研費 20K14101 の助成を受けたものです．

参考文献

[1] 槇原絵里奈, 池田太郎, 小野景子, 新濱遼大. オンラインジャッジシステムにおける解答履歴を利用した問題の関係性調査. 情報処理学会論文誌, Vol. 63, No. 3, pp. 742–751, 2022.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, pp. 3111–3119, 2013.

[3] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[4] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pp. 1188–1196, 2014.

[5] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. Unsupervised learning of sentence embeddings using compositional n-gram features. *arXiv preprint arXiv:1703.02507*, 2017.

[6] 新濱遼大, 槇原絵里奈, 小野景子, 幾島直哉, 山川蒼平. オンラインジャッジシステムにおける問題文の類似度調査. 研究報告ソフトウェア工学 (SE), Vol. 2021-SE-209, No. 9, pp. 1–7, 2021.

[7] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. * sem 2013 shared task: Semantic textual similarity. In *Proceedings of the Second joint conference on lexical and computational semantics (* SEM 2013), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity*, pp. 32–43, 2013.

[8] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14, 2017.

[9] Eneko Agirre, Daniel Cer, Mona Diab, and Aitor Gonzalez-Agirre. Semeval-2012 task 6: A pilot on semantic textual similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics (* SEM 2012)–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pp. 385–393, 2012.

[10] James D Evans. *Straightforward statistics for the behavioral sciences.* Thomson Brooks/Cole Publishing Co, 1996.

[11] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[12] Yijia Zhang, Qingyu Chen, Zhihao Yang, Hongfei Lin, and Zhiyong Lu. Biowordvec, improving biomedical word embeddings with subword information and mesh. *Scientific data*, Vol. 6, No. 1, pp. 1–9, 2019.

[13] Qingyu Chen, Yifan Peng, and Zhiyong Lu. Biosentvec: creating sentence embeddings for biomedical texts. In *2019 IEEE International Conference on Healthcare Informatics (ICHI)*, pp. 1–5, 2019.

Scratchを用いたプログラミング演習における教員支援を目的とした採点支援システムの提案

Grading Support System for Supporting Teachers in Scratch Class

若松 玲依[*] **槇原 絵里奈**[†] **小野 景子**[‡] **新濱 遼大**[§]

あらまし プログラムの採点業務は，C や Java などのテキストプログラミング言語を用いた教育において重要であるため，ビジュアルプログラミング言語の Scratch を用いた教育においても重要となる可能性は高い．しかし，Scratch はソースコードを任意の箇所に配置でき，さらに実行結果が GUI であるため，ソースコードや実行結果の確認および比較に時間を要する．そのため，Scratch プログラムの採点業務は教員に大きな負担を強いると考えられる．そこで我々は，Scratch プログラムの採点支援システムを提案する．本システムは模範解答との差分表示や実行結果の並列表示，類似度が高いプログラムの検出を行う．模範解答との差分表示や実行結果の並列表示により，プログラムの全体像を効率的に把握できる．また，類似度が高いプログラムを検出することで，採点評価に誤りがないか再確認できる．よって，これらの機能を組み合わせて利用することで，多様なプログラムに対して，効率的かつ正確な採点支援が行えると考える．また，本システムではテストケースを作成する必要がないため，教員能力の依存度が低いと考えられる．

1 はじめに

プログラミングの授業形態として，生徒が教員より与えられた課題の達成に向けて取り組み，教員が生徒の解答を採点する形態が存在する．このような授業形態は大学の教育機関で，Java や C などのテキストプログラミング言語を用いて実施されている．そのため，初等教育機関でもプログラミング教育の充実化に向けて，ビジュアルプログラミング言語の Scratch[1]を用いて，実施する可能性が高い．また，世界的には一部の学校で，Scratch を用いて実施している [1]．よって，初等教育機関において，Scratch プログラムの採点業務は教員の重要なタスクになると考えられる．しかし，Scratch はソースコードを任意の箇所に配置できるため，ソースコードの確認や比較に時間を要する．さらに，実行結果は GUI であるため，実行結果の確認や比較も時間を要する．そのため，Scratch プログラムの採点業務は教員に大きな負担を強いると考えられる．

そこで我々は先行研究として，ソースコードに着目し，Scratch プログラムにおけるソースコードの差分表示を提案した [2]．そして本研究では，提案したソースコードの差分表示を実装し，実行結果や類似度にも着目することで，複合的な観点からプログラムの採点支援を行うシステムを提案する．また，提案システムでは教員に対して，テストケースを作成するほどの専門的知識を要求しないことを目指す．

2 準備

2.1 Scratch

Scratch は MIT メディアラボが開発したビジュアルプログラミング言語であり，学習効果として，テキストプログラミング言語の移行も容易となる [3]．そのため，

[*]Rei Wakamatsu, 同志社大学大学院理工学研究科

[†]Erina Makihara, 同志社大学理工学部

[‡]Keiko Ono, 同志社大学理工学部

[§]Ryota Shinhama, 同志社大学大学院理工学研究科

[1]https://scratch.mit.edu/

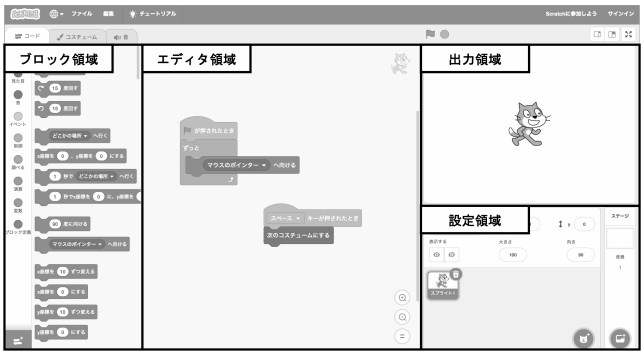


図 1: Scratch の画面

初学者を対象とした教育現場で多く採用されており，文部科学省も Scratch を用いた教育カリキュラムを作成している [4]．図 1 に Scratch の画面を示す．Scratch は主にブロック領域，エディタ領域，出力領域，設定領域の 4 つの領域によって構成される．Scratch では，ブロック領域でプログラムの命令処理にあたるブロックを選択し，エディタ領域で任意の箇所にブロックを配置し繋ぎ合わせることで，プログラムを実装する．また，設定領域ではスプライトと呼ばれるオブジェクトの座標や向きなどのプロパティを設定する．そして，出力領域では実行結果が GUI で表示され，マウス操作やキーボード入力を要求する．

2.2 Scratch の教育支援に関する研究

Scratch を用いたプログラミング教育において，教員の重要なタスクとして，プログラムの指導や採点が考えられる．そこで槇原らは，プログラムの指導に焦点を当て，生徒全員の Scratch プログラムを 1 分ごとに収集し，教員に対して，使用されているブロック数や条件分岐の数などの定量的なメトリクスを提示した [5]．これにより，指導タイミングの推定を行うことができた．また Nina らは，プログラムの採点に焦点を当て，生徒のプログラムを評価するため，ソースコードの制御フローを解析し，模範解答との差異を検出した [6]．そして Stahlbauer らは，オブジェクトのプロパティに基づいて，テストケースによる評価を行う環境を実装した [1]．これらにより，自動的にプログラムを評価することができた．

2.3 Scratch の採点業務に対する課題および要件

教員がプログラムの採点業務を行うにあたり，以下のプロセスが想定される．

手順 1 プログラムの読み込み
手順 2 実行結果の確認や比較
手順 3 ソースコードの確認や比較
手順 4 採点評価
手順 5 採点評価の再確認

このようなプロセスを実施するにあたり，Scratch は C や Java などのテキストプログラミング言語と比較して，課題が多い．具体的には，Scratch ではソースコードを任意の箇所に配置できるため，ソースコードの確認や比較にあたり，任意のソースコードを探す作業が生じる．また，実行結果は GUI であるため，実行結果の確認や比較にあたり，キーボードを入力するだけでなく，マウスを操作する作業が生じる．そのため，Scratch ではソースコードや実行結果の確認や比較にあたり，新たな作業を行う必要がある．よって，採点業務に費やす時間は増加すると考えられる．また，Scratch は実装方法が多く，実行結果の自由度も高いため，各プログラムを

教員の採点基準に対応付けることは容易でない．よって，採点基準に沿ったプログラムに対して，正誤判定が異なるような誤採点が生じる可能性が考えられる．

そこで既存研究 [1] [6] では，プログラムの自動評価により，ソースコードや実行結果の確認および比較を効率化した．しかし，効率化にあたりテストケースを作成する必要があり，すべての条件を適切に満たすテストケースの作成は，プログラムに対する深い知識を要す．また，プログラムの自動評価では，実装方法が異なるソースコードや自由度の高い実行結果など，多様なプログラムを加味することは容易でない．さらに，誤採点を防ぐアプローチは実施されていない．したがって，本研究では以下の要件を満たすことで，専門的知識を要求しない上で，多様なプログラムに対して，複合的な観点から効率的かつ正確に採点支援を行うシステムを提案する．

R1 ソースコードの確認や比較を容易にする
R2 実行結果の確認や比較を容易にする
R3 採点評価が類似するプログラムを検出する

3 提案システム

3.1 概要

我々は，以下の機能を有した Scratch プログラムの採点支援システムを提案する．F1 から F3 の各機能は 2.3 節の要件 R1 から R3 へそれぞれ対応している．

F1 ソースコード間の差分表示
F2 実行結果の並列表示
F3 類似度が高いプログラムの検出

ソースコード間の差分表示では，任意の箇所に存在したソースコードを一定の規則で並び替え，異なるブロックや値を可視化する．これにより，教員は模範解答と学生のソースコードを効率的に比較できるため，採点業務の効率化につながると考える．そして，実行結果の並列表示では，GUI 形式の実行結果に対して入力動作の自動化を行い，実行から終了までの内容全てを記録し，比較対象の実行結果を並列表示する．これにより，教員は模範解答と学生の実行結果を効率的に比較できるため，採点業務の効率化につながると考える．また，ソースコード間の差分表示と実行結果の並列表示を組み合わせて利用することで，複合的にプログラムの全体像を把握できると考える．そして，類似度が高いプログラムの検出では，ソースコード間の編集距離や実行結果間の距離，実装難易度に基づいて類似するプログラム群を可視化する．これにより，教員はプログラムの採点評価に誤りがないか再確認できるため，誤採点の防止につながると考える．また，これらの機能ではテストケースを作成する必要がないため，専門的知識に依存しにくいと考える．

3.2 システム構成

本システムは Web アプリケーションであり，生徒や教員はブラウザを通じて利用する．想定として，生徒は Scratch のオンラインエディタでプログラミングを行い，実装したプログラムを本システムへ提出する．そして，本システムでは提出されたプログラムの分析を行い，3.1 節で述べた各機能を教員へ提示する．また，本システムの Web サーバは Python の Web フレームワークである Flask[2]で構築し，データベースは MySQL[3]で構築した．

3.3 システムの提供する機能

3.3.1 ソースコード間の差分表示

図 2 にソースコード間の差分表示に関する UI を示す．本機能では，JSON ファイルとして格納された Scratch プログラムのブロック情報を解析することで，模範解

[2]https://flask.palletsprojects.com/en/2.1.x/
[3]https://www.mysql.com/

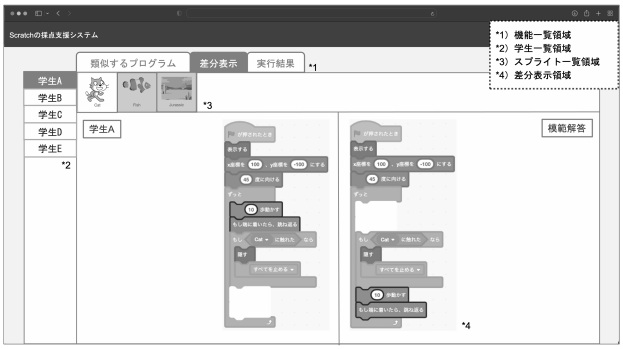


図 2: ソースコード間の差分表示に関する UI

図 3: 実行結果の並列表示に関する UI

答と学生間のソースコードにおける差異を検出する．そして，JSON ファイルを編集することで，異なるブロックやブロック内の値に対して枠線の色を強調的に表示し，改行が必要な箇所は白色のブロックを割り当てる．さらに，Scratch のオンラインエディタを拡張することで，編集した JSON ファイルを読み込み，差分を可視化した．よって，本機能を利用することで，ソースコード間の差異を細かな粒度で認識することが容易となる．また，ソースコードが複数存在する場合，既存研究 [6] では類似したソースコード同士に対応関係をつけていたが，本研究では複数のソースコードを一つに連結し差分表示を実施した．さらに，各スプライト毎にソースコードが複数存在するため，利用したスプライトを表示し，切り替える仕様とした．これらはバックグラウンドで行うため，教員が特別な操作をする必要はない．

3.3.2 実行結果の並列表示

図 3 に実行結果の並列表示に関する UI を示す．本機能では，教員に対して模範解答のプログラムにおける実行動作を要求する．その際，マウス操作やキーボード入力のログを逐次収集し，実行動作を記録する．そして，模範解答と学生のプログラムに対して実行動作の再現および実行結果の録画を実施し，模範解答と学生の実行結果を並列に表示する．よって，本機能を利用することで，実行結果の差異が認識しやすくなるだけでなく，プログラムの実行に必要な入力動作も軽減する．実装は，入力動作の記録および再現が可能な Python の Pynput モジュールを用いた．また，本機能ではマウス操作の速度が一定以上超える場合，実行動作の記録および再現が困難であるため，有効速度の検証や精度の向上は今後の課題とする．

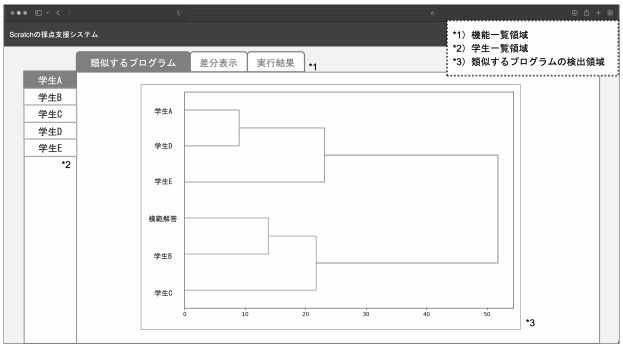


図 4: 類似度が高いプログラムの検出に関する UI

3.3.3 類似度が高いプログラムの検出

図 4 に類似度が高いプログラムの検出に関する UI を示す．本機能では，各生徒のプログラムに対して，(1) ソースコードにおける模範解答との編集距離，(2) 実行結果における模範解答との距離，(3) 実装難易度を取得し，階層的クラスタリングを実施する．そして，階層的クラスタリングで得られたデンドログラムを用いることで，類似度が高いプログラム群を可視化した．よって，本機能を利用することで，採点評価が一致しやすいプログラム群が分かるため，採点評価に誤りがないか確認する際，一つの参考指標となる．また，採点基準の形成にも貢献できると考えられる．

ソースコードにおける模範解答との編集距離は，模範解答と各生徒のプログラムを Scratch から Java へと変換し，抽象構文木を生成した後，編集距離を算出した．木構造による編集距離を採用することにより，差分では加味することが容易でない，一部順序が異なるプログラムや変数名が異なるプログラムなどを近似した距離として取得できる．また，著者らが目視で確認したところ，Java に変換後の編集距離と Scratch の編集距離は，変換不可能なブロックを除いた場合，等価であった．実装は，Scratch を Java へ変換可能な ScratchFoot[4]および木構造の生成や編集距離が算出可能な GumTreeDiff [7] を用いた．次に，実行結果における模範解答との距離は，模範解答と各生徒の実行結果を動画からフレームへと分割し，各フレームに対して輝度情報に基づくハッシュ値を算出した後，模範解答と各生徒の時系列データを生成した．さらに，時系列データの距離を測る手法である動的時間伸縮法 [8] を用いて，模範解答と各生徒の時系列データに対して，距離を算出した．最後に，実装難易度については，Dr. Scratch [9] を用いて算出した．Dr. Scratch は，実装難易度を一定の評価基準で点数化するため採用した．

また，本機能の検出結果が適当であるか事前調査を実施した．具体的には，20 代大学生 5 名に課題を与えることで模範解答を含め 6 件のデータを作成し，本機能による検出結果の適当性について，20 代大学生 3 名に 5 段階評価でヒアリングを行った．被験者に与えた課題は，2 つのスプライトが自動で移動し，重なった際にプログラムが終了するものとした．そのため，ソースコードに必要な要素として，座標の初期化や繰り返し処理，条件分岐処理，終了処理が挙げられる．また，ソースコードの規模は各スプライト毎に 5 行から 10 行である．ヒアリングの結果，平均評価が 4.7 であった．よって，検出結果が適切である可能性が高いため，誤採点を防ぐことが可能だと予想される．しかし，本実験は事前調査であり，被験者数やデータ数が十分でないため，今後は十分な被験者やデータ数を収集し，誤採点を防ぐことが可能なケースや不可能なケースを検証する．

[4]https://github.com/VictorNorman/ScratchFoot

4　提案システムの考察

既存研究 [1] [6] では，ソースコードの差異やテストケースに焦点が当てられている．よって，ソースコードの差異検出やテストケースの利用は，実装方法が一意に決定する場合や教員の能力が高い場合など，特定の状況に対して強力な採点支援であると考えられる．一方，本システムではソースコードや実行結果，類似度などから複合的にプログラムを評価する．そのため，実装方法が異なるソースコードや自由度の高い実行結果など，多様なプログラムに対して採点支援ができると考える．また，テストケースを作成する必要がなく，教員能力の依存度も低いと考えられる．

本システムの有効性を証明するにあたり，多様なプログラムを対象とした評価実験が必要である．よって，同一の課題に対する様々な書き方のソースコードを準備し，提案システムの有無によって採点時間や正確性にどのような差異が生じるか調べる必要がある．また，教員の能力に依存しないと示す必要があるため，教員役の被験者はプログラミング能力が初級者から上級者まで設定する必要がある．

5　終わりに

本稿では，教員の重要なタスクとして Scratch プログラムの採点が顕在化すると見据え，Scratch プログラムの採点支援システムを提案した．本システムでは，Scratch プログラムに対して，ソースコード間の差分表示，実行結果の並列表示，類似度が高いプログラムの検出を行う．そのため，複合的な観点からプログラムの採点支援を行うとともに，教員に対して一定水準のプログラミングスキルを要求しない．よって，課題内容や教員能力の依存度が低いと考えられる．今後は教育現場に準ずる環境で評価実験を行ない，提案システムの有効性を検証する．

謝辞　本研究は JSPS 科研費 20K14101 の助成を受けたものです．

参考文献

[1] Andreas Stahlbauer, Marvin Kreis, and Gordon Fraser. Testing scratch programs automatically. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE2022)*, pp. 165–175, 2019.

[2] 若松玲依, 槇原絵里奈, 新濱遼大, 小野景子, 藤原賢二．ポスター発表: Scratch プログラムの差分情報に着目した採点補助システム構築に向けて．第 28 回ソフトウェア工学の基礎ワークショップ (FOSE2021), 2021.

[3] David Weintrop and Uri Wilensky. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, Vol. 18, No. 1, pp. 1–25, 2017.

[4] 文部科学省. 小学校プログラミング教育に関する研修教材. `https://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1416408.htm`. (Accessed on 07/18/2020).

[5] 槇原絵里奈, 米田浩崇, 小野景子ほか. オンライン scratch プログラミング演習支援にむけたコードメトリクス可視化ツールの提案および評価. 情報処理学会論文誌教育とコンピュータ (TCE), Vol. 8, No. 2, pp. 37–50, 2022.

[6] Nina Körber, Katharina Geldreich, Andreas Stahlbauer, and Gordon Fraser. Finding anomalies in scratch assignments. *arXiv preprint arXiv:2102.07446*, 2021.

[7] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. Fine-grained and accurate source code differencing. In *Proceedings of ACM/IEEE International Conference on Automated Software Engineering (ASE2014)*, pp. 313–324, 2014.

[8] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, Vol. 26, No. 1, pp. 43–49, 1978.

[9] Jesús Moreno-León and Gregorio Robles. Dr. scratch: A web tool to automatically evaluate scratch projects. In *Proceedings of the workshop in primary and secondary computing education (WiPSCE2015)*, pp. 132–133, 2015.

プログラミング演習におけるセグメントを用いたソースコードの誤り箇所特定方法の提案

A Method for Localizing Defects in Source Codes Using Segments in Programming Exercises

澤田 侑希 * 梅田 祐一郎 † 蜂巣 吉成 ‡ 吉田 敦 § 桑原 寛明 ¶

Summary. In programming exercises, we propose a method to localize defects based on automatic program repair. Localizing defects is accomplished by comparing a learner's programs and model answers using program segments, which are sequences of statements with no branches, and replacing a learner's segments with model answers' segments until pass testcases. We implemented a localizing defects tool and confirmed that our tool can find defects for practical use.

1 はじめに

プログラミング演習で，学習者は与えられた演習問題に取り組む．コンパイラが検出不可能な誤りは，知識が少ない学習者にとって自力での特定が難しい．教員やTA が対応することで支援できるが，対応できる人数に限りがある．誤り箇所を自動で特定できれば，学習者の支援に役立つ．Fault Localization や自動プログラム修正に関する研究は多くされている [1] [2] が，単に正しい実行結果が得られれば良いのではなく，学習者に誤り箇所を認識させて自力で修正させる必要がある．

模範解答には別解が存在するが，誤りを認識させるには学習者の意図に近い別解を選択する必要がある．1 つのプログラムの複数の部分に別解がある場合，その組み合わせを網羅すると，模範解答の総数が膨大になることがある．

本研究ではこれらの問題を解決するために，コンパイル可能であるが期待した実行結果が得られない学習者の C プログラムを対象として，セグメントを用いた誤り箇所の特定方法を提案する．学習者が解答したプログラム (以下，修正対象プログラム) と模範解答のプログラム (以下，模範解答プログラム) をセグメント単位で比較して差分を生成することで，誤り箇所を特定する．本研究におけるセグメントとは，制御文の複合文を基にプログラムを分割したものである．セグメントごとに別解を作成することで組み合わせをなくし，模範解答の総数を削減できる．

2 関連研究

Spectrum-Based Fault Localization(SBFL) はテストケースによる実行経路の情報を用いて誤り箇所を推定する手法である [1]．本研究では，正解が既知で小規模なプログラムに対して，教育を目的として学習者の意図に近い誤り箇所の特定方法を提案する．Shalini らの研究では，半教師あり検証済みフィードバック生成方法を提案している [3]．学習者のプログラムをクラスタ化し，誤りがあるプログラムに対して類似度が高く正しい実行結果を得るプログラムを対応付ける．しかし，別解が多い問題ではクラスタが増加し，そこに正しい実行結果が得られるプログラムが存在しない場合，手動で作成して再度クラスタリングする必要がある．

*Yuki Sawada, 南山大学大学院理工学研究科

†Yuichiro Umeda, 南山大学理工学部卒，現株式会社 NTT データ MSE

‡Yoshinari Hachisu, 南山大学理工学部

§Atsushi Yoshida, 南山大学理工学部

¶Hiroaki Kuwabara, 南山大学理工学部

3　誤り箇所特定方法の提案

3.1　問題分析

学習者のコードと模範解答のコードの差分を取り，差異がある部分を置き換えることで誤り箇所を特定できる可能性がある．ソースコード 1 は累乗を計算する演習問題の模範解答プログラムである．修正対象プログラムの例の一部をソースコード 2 に示す．else の入れ子の for 文と，for 文の変数 v2 の計算式に誤りがある．

ソースコード 1　模範解答プログラム

```
1   double func1(double p1, int p2)
2   {
3     int v1;
4     double v2;
5     v2 = 1;
6     if (0 <= p2) {
7       for (v1 = 0; v1 < p2; v1 = v1 + 1) {
8         v2 = v2 * p1;
9       }
10    } else {
11      for (v1 = 0; p2 < v1; v1 = v1 − 1) {
12        v2 = v2 / p1;
13      }
14    }
15    return v2;
16  }
```

ソースコード 2　修正対象プログラム例の一部

```
1   } else {
2     for (v1 = 0; p2 < v1; v1 = v1 + 1) {
3       v2 = v2 * p1;
4     }
5   }
```

セグメントの定義

1. 制御文の本体 (文) はセグメントである．ただし，複合文の波括弧は除く．
2. 制御文の予約語と制御式を囲む丸括弧の部分はセグメントである．
3. これら以外の連続した宣言と文はセグメントである．
4. 制御文の入れ子の場合は 1〜3 のセグメントも入れ子になるが，このうち最内のものをセグメントとする．

ソースコード 2 の誤りは，else 内の文，2，3 行目とソースコード 1 の 11，12 行目を置き換えることで特定できる．しかし，演習では学習者に誤りを自力で修正させる必要があるので，学習者の意図に近い別解を用いて誤り箇所を特定する必要がある．ソースコード 2 では，変数 v2 について「`v2 = v2 / p1;`」で計算する場合と,「`v2 = v2 * p1;`」で計算した後に「`v2 = 1 / v2;`」とする場合が想定できる．ソースコードのみで学習者の意図を汲み取ることは困難であり，適当に置き換えても学習者の意図に近い別解を選択することができないという課題がある．差分が小さい模範解答から順に置き換えることで，学習者の意図を汲み取ることができる可能性が高いので，本研究では差分の大きさを基準に置き換える順番を決定する．

3.2　誤り箇所特定方法の概略

プログラムをコード片の集合と捉え，コード片単位で誤り箇所を特定する．このコード片をセグメントと呼ぶ (3.3 節)．模範解答の別解はセグメント単位で作成する．修正対象プログラムも同様の規則でセグメントに分割し，模範解答のセグメントと差分を取る．差分が小さい模範解答のセグメントを修正対象プログラムのセグメントと置き換えて実行し，用意したテストケースをプログラムがパスしたならばそのセグメントを誤り箇所として特定する．

3.3　セグメント

セグメントの定義を上記に示す．ソースコード 1 は「`int v1; double v2; v2 = 1;`」，「`if (0 <= p2)`」，「`for (v1 = 0; v1 < p2; v1 = v1 + 1)`」，「`v2 = v2 * p1;`」，「`else`」，「`for (v1 = 0; p2 < v1; v1 = v1 - 1)`」，「`v2 = v2 / p1;`」,「`return v2;`」のように 8 個のセグメントに分割する．

誤り箇所を特定するとき，模範解答プログラムとの比較で特定できるが，想定されるすべての別解を模範解答プログラムとして用意する必要がある．プログラムは

セグメントの集合であるので，別解同士で組み合わせが発生し，セグメント単位の別解が重複したプログラムを作成することになる．そのため，別解を網羅すると模範解答の総数が膨大になる．

ソースコード 1 のプログラムでは，指数が正の場合と負の場合などで分岐する．繰り返しは for 文と while 文で記述でき，範囲は 0 以上 p2 未満や 1 以上 p2 以下などで記述できる．これらのセグメントに別解があり，組み合わせが発生する．模範解答をセグメント単位で作成することで別解の組み合わせを考える必要がなくなる．例えばソースコード 1 では，if 文と for 文で別解の組み合わせが発生するが，場合分けの分岐と累乗計算の繰返しのコードを独立して作成すれば良い．

3.4 提案方法

本研究では，修正対象プログラムをセグメント (以下，修正対象セグメント) に分割し，模範解答のセグメント (以下，模範解答セグメント) との差分を用いて誤り箇所を特定する方法を提案する．各修正対象セグメントとすべての模範解答セグメント (以下，模範解答セグメント群) を総当たりで比較することで，誤りが存在するセグメント (以下，誤りセグメント) を特定する．

修正対象セグメントを模範解答セグメントで置き換えるとき，学習者の意図に近いものから順に置き換えたい．行単位で差分を生成すると，比較の対象が大きすぎ 1 行において異なる文字数が考慮されない．セグメント単位で差分を生成すると，変数名の長さなどの影響を受ける．本研究では，トークン単位で差分を生成し (以下，差分情報)，誤り箇所の特定に用いる．差分が小さいものから順に置き換えてプログラムを作成し，テスト実行する．テストケースをパスすれば，そのセグメントを誤りセグメントとして特定できる．処理の流れを図 1 に示す．

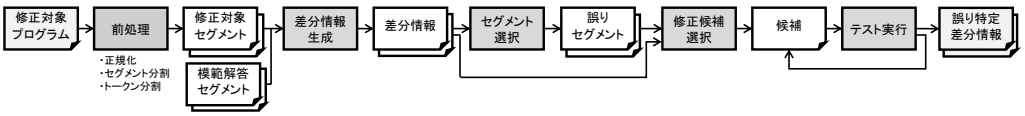


図 1　提案方法の処理の流れ

前提条件として，修正対象プログラムは「コンパイルは可能だが，正しい実行結果を得られない.」,「関数を作成する問題で，関数名と仮引数は与えられる.」を満たすものとする．変数名の対応付けは我々の既存研究 [4] を利用する．

3.4.1 正規化

修正対象プログラムを正規化し，表記の揺れを削減する．正規化の処理は「余分な空白文字，改行文字，コメントの削除」,「変数宣言と初期化式がまとめて記述されている場合，別々に書き換え」,「不等号の向きを>から<，>=から<=に書き換え」,「増分演算子，減分演算子，複合代入演算子を用いた計算式を，代入を用いた二項演算の形式に書き換え」,「制御文の波括弧がない場合，波括弧の付加」を行う．本研究では式の中の余分な括弧の削除や,「a+b」と「b+a」のようなオペランドの順序変更には対処していない．

3.4.2 セグメント分割・トークン分割

定義に従いプログラムをセグメントに分割する．セグメントのコードをトークンごとに改行する．

3.4.3 差分情報の生成

修正対象セグメントと模範解答セグメントを総当たりで対応付け，差分情報を生成する．差分情報は，トークン単位の差分を検出箇所とともにまとめたものである．修正対象セグメントと模範解答セグメントが一致する場合，差分ゼロが差分情報として出力される．

3.4.4　修正するセグメントの選択

差分情報を用いて，誤りセグメントを特定する．差分ゼロの差分情報が存在しない修正対象セグメントを誤りセグメントとして選択する．図 2 にソースコード 1，2 を用いた誤りセグメントの選択の例を示す．例では分かりやすいようにトークン分割の前の状態で示す．差分情報の数字は異なるトークン数を表す．修正対象セグメントはソースコード 2 をセグメント分割したものである．模範解答セグメントはソースコード 1 の模範解答セグメント群から，最小の差分情報の生成元の模範解答セグメントを抜粋したものである．図 2 の場合，6 番目の修正対象セグメントが選択されるが，7 番目の修正対象セグメントも誤りセグメントであるので，6 番目の修正対象セグメントのみ置き換えても正しい実行結果を得られない．

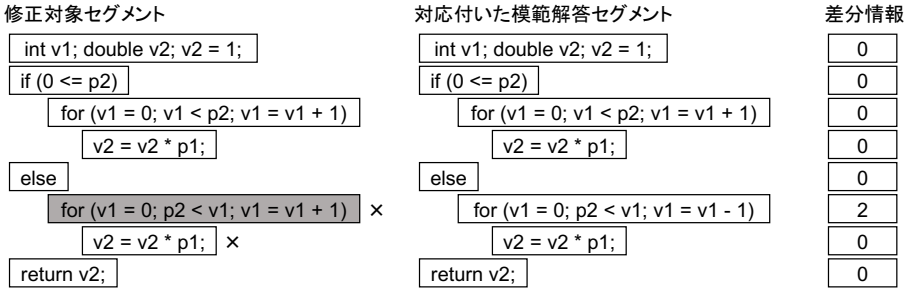


図 2　誤りセグメントの選択の例

セグメントの修正 (3.4.5 節，3.4.6 節) で誤りが解消できない場合，誤りセグメントを追加で探す必要がある．誤りセグメントと同一の模範解答セグメントが存在する場合，差分ゼロとなるので，差分ゼロの修正対象セグメントを順に誤りセグメントの可能性があるものとして修正を試みる．図 2 では 7 番目の修正対象セグメントが誤りセグメントであるが (正しい模範解答セグメントは「`v2 = v2 / p1;`」)，同一の模範解答セグメントが存在して，差分ゼロとなっている．この場合，6 番目の修正対象セグメントに加えて，差分ゼロである 1〜5，7，8 番目の修正対象セグメントも誤りセグメントの可能性があるので，順に選択して修正を試みる．

3.4.5　セグメントの修正候補の選択

3.4.4 節の方法で選択された修正対象セグメントについて，模範解答セグメント群から正しいと思われるセグメントを選び，それで置き換えたプログラムを作成してテストしていく．学習者の意図に近い正しい模範解答セグメントは修正対象セグメントとの差分が小さいと考え，差分がしきい値以下の模範解答セグメントを置換の候補とし，差分の小さい順に選択していく．

図 3 に 6 番目と 7 番目の修正対象セグメントを修正する場合の候補の選択の例を示す．差分のしきい値は 6 とし，候補となる模範解答セグメントを昇順に示している．置換するセグメントが複数あるので，模範解答セグメントの直積 ((6-1，7-1)，(6-1，7-2)，. . .) を作成し，候補とする．

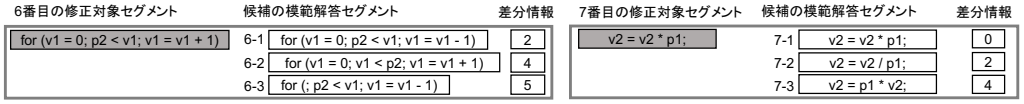


図 3　候補の選択の例

3.4.6　修正したプログラムのテスト実行

候補を用いて誤りセグメントを修正する．セグメントを結合し，セグメント分割の前処理で削除した関数や波括弧などを補完してプログラムを作成する．作成したプログラムをコンパイルおよび実行する．コンパイルエラーが発生した場合，次の候補の処理を行う．続いて，生成したプログラムを実行する．指定した時間以上実

行した場合は無限ループとみなし，タイムアウトさせて次の候補の処理を行う．テストケースをパスした場合，各セグメントの誤りの有無を出力する．また，誤りセグメントでは差分情報を出力する．パスしなかった場合，次の候補の処理を行う．

すべての候補のプログラムを実行してテストケースをパスしなかった場合，どのセグメントが誤りセグメントかわからない場合の誤りセグメントが他にも存在する可能性がある．この場合，特定の対象とするセグメントを増やして再度候補を選択し，プログラムを作成して実行する．図 3 の例では，6 番目の修正対象セグメントのみ置き換えてもテストケースをパスしない．6 番目の修正対象セグメントに加えて 1 番目の修正対象セグメントの候補から順に置き換えていき，最終的に「6-1」と「7-2」の候補でテストケースをパスする．

3.5　誤り箇所特定に要する時間

3.5.1　計算量の分析

模範解答セグメントの総数を m とすると，修正対象セグメントごとに m 個の差分情報を生成する．誤りセグメントの数を k とすると，m^k 個のプログラムを作成する必要がある．修正対象セグメントの総数を n とすると，どのセグメントが誤りセグメントかわからない場合，誤りセグメントの選び方は ${}_n\mathrm{C}_k$ 通りである．この場合，${}_n\mathrm{C}_k \times m^k$ 個のプログラムを作成する必要がある．

3.5.2　誤りセグメントの最大数

修正対象セグメントが 10 個，模範解答セグメントが 100 個で，プログラムの作成と実行に 0.1 秒要する場合の実行時間を，誤りセグメントの数ごとに計算する．

誤りセグメントが 1 個の場合，計算量は ${}_{10}\mathrm{C}_1 \times 100^1$ となり，実行時間は 1 分 40 秒である．誤りセグメントが 2 個の場合，計算量は ${}_{10}\mathrm{C}_2 \times 100^2$ となり，実行時間は 12 時間 30 分である．誤りセグメントが 3 個の場合，計算量は ${}_{10}\mathrm{C}_3 \times 100^3$ となり，実行時間は 416 日と 16 時間である．差分のしきい値を用いて候補を削減するので，実行時間は見積りより短くなる．誤りセグメントが 2 個までの場合，実用的な時間内に誤り箇所が特定できる可能性がある．

4　ツールの試作と評価

提案方法に基づいて誤り箇所特定ツールを試作した．Python で実現し，約 750 行である．C 言語の 3 つの演習問題を対象とした．実行時間という定量的な観点と，誤り箇所が特定できたかという定性的な観点で評価する．演習問題 1 から 3 は「実数 p1 の p2 乗を求める関数 func1 を作成せよ．(p2 は整数)」,「文字列 p1 の中の文字 p2 へのポインタを返す関数を作成せよ．(複数含まれる場合は最も先頭側の文字，含まれていなければ NULL)」,「2 つの正の整数 p1，p2 の最大公約数を再帰で求める関数を作成せよ.」である．演習問題 1 では，模範解答セグメントを 81 個，演習問題 2 では 18 個，演習問題 3 では 17 個作成した．3 つの演習問題で最も規模が大きいものは演習問題 1 であり，修正対象セグメントは最大で 16 個を想定した．ツールは MacBook Pro(CPU は M1 pro で 8 コア，メモリは 16GB，OS は macOS Monterey 12.4) で実行した．差分のしきい値は 10 で，タイムアウトは 1 秒とした．差分は diff コマンドで求め，差分情報は差分の検出箇所と変更のあったトークンの数とする．

4.1　実行時間の評価

誤りセグメントが 1 個の場合，特定にかかる計算量は差分ゼロの模範解答セグメントが存在するとき最悪である．誤りセグメントが 2 個の場合，特定にかかる計算量は 2 個とも差分ゼロの模範解答セグメントが存在するとき最悪である．誤りセグメントが 1 個の場合と 2 個の場合で，最悪のプログラム (誤りセグメントがプログラムの末尾にある) を作成した．time コマンドで計測し，3 回実行して平均した．

誤りセグメントが 1 個で実行に最も時間を要したものは実行時間は 1 分 9 秒であっ

た．誤りセグメントが 2 個では実行時間は 1 時間 57 分 29 秒であった．

4.2　学習者が記述した誤りを含むプログラムの評価

C 言語を一通り学習した学部 3 年生十数名を対象に行った演習問題から実際の誤りプログラムを数個を選び，コードの一部に変更を加えてツールに入力し，誤り箇所が特定できるか評価した．特定対象の誤りは変更していない．

演習問題 1 の修正対象プログラムを図 4 の左に示す．7 番目のセグメントの正しい記述は「`v2 = v2 / p1;`」である．ツールで誤り箇所を特定できた．演習問題 2 の修正対象プログラムを図 4 の中央に示す．1 番目のセグメントの正しい記述は「`*p1 != '\0'`」である．ツールで誤り箇所を特定できた．演習問題 3 の修正対象プログラムを図 4 の右に示す．1 番目のセグメントの正しい記述は「`p2 == 0`」であり，2 番目のセグメントは「`return p1;`」である．ツールで誤り箇所を特定できた．

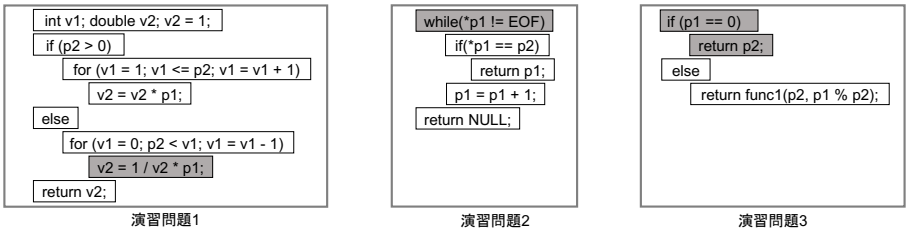


図 4　演習問題の修正対象プログラム

5　考察

記述不足でセグメントが比較されない場合，誤り箇所を特定できない可能性がある．例えば，ソースコード 1 で 15 行目の return 文が不足している場合，記述すべき箇所がセグメントにならないので，比較されず特定できない可能性がある．また，設定する差分のしきい値が誤り箇所を特定した差分情報より小さい場合，誤り箇所の特定に失敗することがある．

セグメントを用いず 1 行を 1 文として行単位で比較する場合，各セグメントに平均 a 行あるとすると ${}_{an}\mathrm{C}_k \times (am)^k$ 個のプログラムを作成する必要があるので，セグメントを用いた方が模範解答の総数と比較回数を削減できる．

6　おわりに

本研究では，セグメントを用いてプログラミング学習におけるソースコードの誤り箇所の特定方法を提案した．今後の課題として，依存関係を考慮したセグメントの対応付けの実現が挙げられる．また，差分のしきい値とツールの性能の関係性を調査し，実際の演習で対象を増やして評価をする必要がある．

謝辞　本研究の一部は 2022 年度南山大学パッヘ奨励金 I-A-2 の助成を受けた．

参考文献

[1] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, Franz Wotawa：A Survey on Software Fault Localization, IEEE Transactions on Software Engineering (TSE), Vol.42, No.8, pp.707-740 (2016).
[2] Claire Le Goues, Michael Pradel, Abhik Roychoudhury, Satish Chandra：Automatic Program Repair, IEEE Software, Vol.38, Issue 4, pp.22-27 (2021).
[3] Shalini Kaleeswaran, Anirudh Santhisr, Aditya Kanade, Sumit Gulwani：Semi-supervised Verified Feedback Generation, FSE 2016：Proceedings of the 2016 24th ACMSIGSOFT International Symposium on Foundationsof Software Engineering (2016).
[4] 蜂巣吉成，石元慎太郎，吉田敦，桑原寛明：プログラミング学習者の編集途中のソースコードと模範解答における変数の対応づけ方法の提案，ソフトウェア工学の基礎 XXVII (FOSE 2020), pp.109-114 (2020).

VR 環境における文字入力支援システム

A System for Supporting Text Entry in Virtual Reality

清原 隆一* 沢田 篤史† 野呂 昌満‡

あらまし 本研究では VR（Virtual Reality）環境での文字入力支援システムの設計と評価を行う．VR 環境で現実と同じように文字入力が行えるようになることで，声が出せない状況でのコミュニケーションが実現するほか，仕事や教育などの文字を扱う場面が多い VR コンテンツを拡大できる．VR 環境上での文字入力は空間把握が難しく，それが誤入力の原因となる．これらを解決するために，本研究では広く普及しているフリック入力に触感を伴わせて空間把握を行いやすくしたものと，空間把握をする必要がない指文字入力の 2 つを提供し利用者の環境に合わせて切り替えられるようにする．これらの方法に予測変換機能を組み合わせることで誤入力を補完できる文字入力システムを設計する．評価実験では提案する入力方法の入力時間と誤入力回数の計測を行った．

1 はじめに

近年 HMD（Head Mounted Display）の普及が進んだことで，VR（Virtual Reality）はより身近なものとなった．VR 環境上ではボイスチャットがよく用いられるが，文字入力を行うコンテンツを拡大するために音声以外の VR 環境上での様々な文字入力機能の充実が必要である．

VR 環境上では空間把握が難しく，キーボードを用いた文字入力は難しい．また，必要とする文字入力方法は利用者のおかれている状況によって異なり，多発する誤入力を補って効率よく入力を行うための支援が求められる．

本研究の目的は，VR 環境上で操作しやすく，誤入力が少ない文字入力支援システムを実現することである．実現には，困難な空間把握を解決すること，複数の方法を使い分けられること，誤入力を容易に修正できることが必要である．

目的を達成するために，本研究では多くの人に慣れ親しまれているフリック入力方法と空間把握の必要がない指文字を用いた入力方法を提供することを提案する．指文字とは手や指で五十音を表現する視覚言語のことである．これら 2 つの入力方法に予測変換機能を組み合わせることで，誤入力の抑制と入力操作の負担軽減をはかる．

提案の妥当性を確かめるために，提案方法を実装し，実験を行った．その結果，提案方法では指文字の誤認識やキーボードの表示位置のずれにより誤入力が多発した．また，利用しやすい文字入力方法は人によって異なることが分かった．フリック入力については，触感があることでより普段の文字入力に近い操作感を得ることができた．

2 VR 環境上での文字入力における課題

2.1 課題

VR 環境上での文字入力の課題には次の 3 点がある．

- 空間把握が難しいこと
- 利用者や利用者の状況によって必要とする入力方法が異なること
- 誤入力に対応しなければならないこと

*Ryuichi Kiyohara, 南山大学

†Atushi Sawada, 南山大学

‡Masami Noro, 南山大学

空間上に配置されているキーボードは触感がない．現実世界での入力と同じように，触感的フィードバックが必要である．

利用者によって必要とする入力方法は異なり，得意・不得意が必ずある．現在多くの HMD に標準で採用されている入力方法には QWERTY 配列のキーボードしかない．複数の方法を提供することで，利用者の習熟状況や得意・不得意に応じた入力が可能となる．

誤入力をすると，削除と再入力を必要とするので入力に時間がかかる．誤入力に対して適切に補完や修正をすることができれば，入力速度の向上につながる．

2.2 先行研究

Jiban ら [1] は音声認識を行う方法を提案している．音声入力と予測変換を用いることで複雑な操作のない文字入力が可能である．

福仲らの研究 [2] では，Leap Motion[1]という手指認識用デバイスを用いたフリック入力方法を提案した（以下，仮想フリック入力方法と呼ぶ）．フリック入力キーボードを仮想空間上に配置し，それを用いてフリック入力するという方法である．

Leap Motion は手指の形状を認識できることから指文字にも用いられている．屋外での使用を目的として，Leap Motion を紐で固定して首から下げ，指文字を認識する研究がある [3]．この研究では，デバイス本体が安定せず認識率が低下したことが報告されている．VR 環境における指文字認識では，屋外等で話者が自由に移動して発話することを必ずしも想定しなくて良いので，安定的に認識させるために，手指認識デバイスを話者の前面の机などに固定させて利用する必要がある．

3 文字入力支援システムの概要

3.1 課題解決のアプローチ

2.1 節で挙げた 3 点の課題を，本研究では以下のアプローチで解決する．

空間把握が難しいこと，操作に習熟が必要であることについては，フリック入力を用いる．多くの若者は日本語の文字入力にフリック入力を使用している [4]．操作に触感を伴わせることにより普段の文字入力に近い操作感にすることで，習熟の必要なく，普段と同じように文字入力ができる．触感はスマートフォンを直接指で触って操作することで得ることができる．

利用者の嗜好や状況によって必要とする入力方法が異なることについては，複数の入力方法を提供して対応する．本研究では，フリック入力に加え，指文字入力を提供する．指文字は音声でのコミュニケーションが困難な利用者やその支援者の中に習熟している者も多い．指文字はキーを押す操作がないので空間把握の必要がなく，これらの利用者にとって利便性が高くなる可能性があると判断した．これら 2 通りの方法を利用者の嗜好や状況に応じて切り替えられるようにする．

誤入力への対応は，予測変換を用いる．欠損した文字の補完を行うことで，入力する文字数が減ることにより利用者の操作量を減らすことができる．

3.2 システムの設計

図 1 に提案するシステムの概要を示す．スマートフォンのディスプレイ操作を使ったフリック入力（以下，物理フリック入力方法と呼ぶ）と指文字を使った文字入力（以下，指文字入力方法と呼ぶ）の 2 つの文字入力方法を提供する．

利用者は VR を利用する環境や嗜好に合わせ，両者を切り替え文字入力を行う．入力された文字は予測変換システムに入力され，変換する単語の候補が提示される．

物理フリック入力方法では手首に装着したスマートフォンでフリック入力を行う．図 1 に示す物理フリック入力方法について，左が現実で使用する様子であり，右が VR 環境上の様子である．手首に取り付けたスマートフォンの画面でフリック入力

[1] https://www.ultraleap.com/

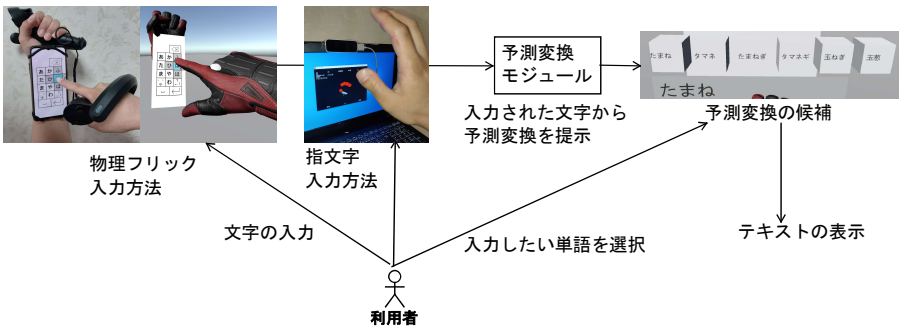


図 1　文字入力支援システムの概要

を行い，スマートフォンの画面を VR 環境上にも出現させる．物理的触感を得る目的としてスマートフォンを採用した理由としては，他のタッチパネルデバイスに比べてスマートフォンを所持している利用者が多いことが挙げられる．

もう 1 つの方法として，指文字入力方法を提供する．指文字を認識するには，それぞれの指が伸びているか曲がっているかを判断する必要がある．VR のハンドコントローラではこれに対する正確な結果が得られない．提案システムでは，机に置いた Leap Motion のカメラで手や指を認識させることで文字入力を行う．キーボードを使った文字入力と違い，指文字での入力は空間把握をする必要がない．

入力を補完するために，入力方法からの入力に対する予測変換機能を搭載し，入力される可能性の高い単語などを予測して提供する．日本語入力での使用を考慮し，単語の予測だけでなく，かな漢字変換機能も搭載するよう設計した．

4　利用実験による評価

4.1　概要

提案手法と予測変換機能の組み合わせの有効性を評価するために行った利用実験について説明する．物理フリック入力方法と指文字入力方法に，既存の入力方法である仮想フリック入力方法 [2] を再現した環境を実装した．

物理フリック入力方法と仮想フリック入力方法は，VR 環境を容易に構築できる Unity で実装する．指文字入力方法は Leap Motion と親和性の高い Processing で実装する．

予測変換モジュールは全ての方法に共通のものを採用する．本研究では Yahoo! JAPAN が提供するかな漢字変換 API(V2)[2]を用いる．この API には予測変換とかな漢字変換を同時に行うことができる利点がある．各方法において入力中の文字列が更新される度に予測変換を検索し利用者に提示する．

4.2　物理フリック入力方法の実装

物理フリック入力方法は，スマートフォンのディスプレイ操作を PC と同期するためのアプリケーション（以下，入力同期アプリケーションと呼ぶ）と PC 上のアプリケーション（以下，VR 環境アプリケーションと呼ぶ）により構成する．図 2 にその概要を示す．ディスプレイ操作のタッチ状態・座標情報の同期については，PUN 2 無料版 [3]が提供する Photon Server を用いたインターネット通信を使用することで，異なるハードウェア間での通信を実装する．入力同期アプリケーションは操作の状

[2] https://developer.yahoo.co.jp/webapi/jlp/jim/v2/conversion.html

[3] https://assetstore.unity.com/packages/tools/network/pun-2-free-119922

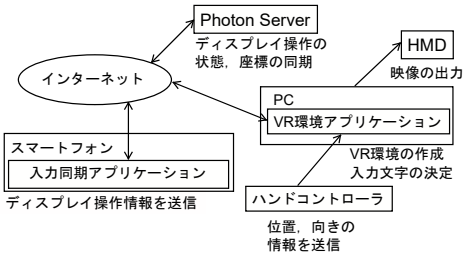


図 2　物理フリック入力方法

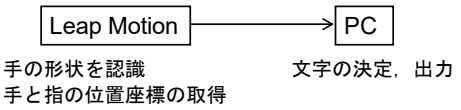


図 3　指文字入力方法

態と二次元座標を常に Photon Server に送り，VR 環境アプリケーションは Photon Server から操作情報を受け取ってキーボードを操作する．PC はディスプレイの操作情報から入力文字を決定し，テキストボックスに反映する．

キーボードは五十音の入力以外に，文字の削除，スペース，改行を行える．文字を入力すると前方に変換入力中の文字と予測変換候のボックスが表示される．利用者は候補の中から文字を選び，ボックスに触れるとその文字が入力される．仮想フリック入力方法に切り替える場合はこのキーボードを空中に設置し，Leap Motion の指先をタッチ座標にして操作を行う．

4.3　指文字入力方法の実装

指文字入力方法の概要を図 3 に示す．Leap Motion は手指の形状を認識して指文字を判定し手と指の位置座標を取得する．

指文字の特徴を元に，指が伸びている本数，指の種類，掌の向き，手の向いている方向，親指と人差し指がくっついているかどうか，という 5 つのカテゴリに分け，これらを組み合わせることで指文字を認識する．

指文字は 3 秒毎に 1 文字の認識を行い，テキストを更新する．画面には認識中の文字，予測変換候補，入力した文字が出現する．また手を画面下の特定の領域に移動させると「濁点・半濁点」,「ん」,「削除」を入力することができる．本実装では，連続する指文字を認識する時間間隔として 3 秒間という値を固定的に設定している．これは，後述の実験において，被験者が 1 文字の指文字を入力するために要する平均的な時間から決定した．より実用的な環境において利便性を向上させるためには，習熟に応じてこの間隔を調整できることが必要となる．

4.4　利用実験の内容

提案システムにより 2.1 節で挙げた課題が解決できているか，その妥当性を利用者実験により確認する．

実験ではフリック入力に成熟している 22 歳から 23 歳の被験者 5 名に物理フリック入力方法の概要を説明後，5 分間の練習時間を与える．その後，無作為に選んだ単語 5 つを連続して入力させる．予測変換なしで入力した場合と予測変換ありで入力した場合の入力時間と誤入力回数を計測し，比較を行う．また，指文字入力方法と従来方法 2 つについても被験者に使用させ，それぞれの方法に対する感想を聴取する．従来方法は実装した仮想フリック入力方法と，SteamVR に搭載されている QWERTY 配列キーボードを用いた入力方法（以下，QWERTY 配列入力方法と呼ぶ）である．被験者からそれぞれの方法に対する感想を聴取する．

4.5　実験結果

物理フリック入力方法の入力時間，誤入力回数，1 分間あたりの入力文字数（Characters per minutes，以下 CPM と呼ぶ），誤入力率の予測変換なしの場合を表 1 に，予測変換ありの場合を表 2 に示す．

表 1　物理フリック入力方法の予測変換なしの場合の結果

被験者	入力時間 (s)	誤入力回数	CPM	誤入力率 (%)
被験者 1	110	12	14.29	46.15
被験者 2	140	14	11.11	53.85
被験者 3	31	1	50	3.85
被験者 4	123	5	12.7	19.23
被験者 5	68	4	16.67	26.92
平均	94	7	16.67	26.92

表 2　物理フリック入力方法の予測変換ありの場合の結果

被験者	入力時間 (s)	誤入力回数	CPM	誤入力率 (%)
被験者 1	130	8	12	30.77
被験者 2	108	8	14.29	30.77
被験者 3	45	13	35.29	50.00
被験者 4	96	3	16.22	11.53
被験者 5	75	7	20.69	26.92
平均	91	7	17.14	30.00

物理フリック入力方法は福仲らの提案手法 [2] で行われた検証と比べて CPM が少なく，誤入力回数が多かった．被験者からは「VR で見えている画面と実際にタッチするものの位置がずれており，正確に入力ができなかった」という意見が挙がった．

指文字入力方法については指の誤認識による誤入力回数が多く，入力時間を計測できなかった．仮想フリック入力方法は物理フリック入力方法と比べて誤入力回数が少なかったが，被験者からは「指の移動が多くて疲れた」という意見が挙がった．QWERTY 配列入力方法は最も最も入力速度が高く，誤入力回数が少なかった．

被験者に最も使いやすい入力方法を聴取した結果は，QWERTY 配列入力方法が 1 名，仮想フリック入力方法が 2 名，物理フリック入力方法が 2 名となり，ばらつきが出た．

5　考察

4.5 節の実験結果でも言及した通り，物理フリック入力方法ではキーボードの位置が VR 上で見えているものと現実にあるものがずれてしまい，既存の方法よりも誤入力が多くなった．これは，ハンドコントローラの位置を参考にキーボードを投影していたことが原因であると考えている．Haiyan ら [5] は物理的な QWERTY 配列キーボードの位置をハンドコントローラから取得し，HMD に取り付けたカメラで手の映像を撮影して VR 環境上に表示する方法を提案している．この方法では，キーボードとハンドコントローラが密接していたのでずれが生じず，現実と同じように文字入力できたことが報告されている．このことから，物理フリック入力方法のずれを解決するには，トラッキングできる VR デバイスをスマートフォンに密着させるか，スマートフォンのセンサを用いたデバイスに頼らない位置特定方法が必要がある．

物理フリック入力方法は誤入力が最も多かったが，2 名の被験者はこの方法を最も使いやすいと評価した．触感があることで普段の入力に操作感が近かったことや，文字の削除が素早く行えたので誤入力を修正する操作が速かったことが要因であると考えている．

指文字入力方法では誤認識が多かった．誤入力の補完ができなかった原因は，予測変換が入力した文字と一致した検索結果しか出力できなかったからと推測している．この問題に対して，似ている指文字同士をあらかじめピックアップし，認識した指文字と誤認識しやすい指文字を同時に候補として挙げて，選択できるようにす

るなどの方法を検討する必要がある．

物理フリック入力方法の効果を阻害する要因として，予測変換の選択にかかる独自な操作があると考えている．独自操作に習熟するための練習によって入力速度が改善する可能性もある．より現実に近い文字入力に近づけるためには，各入力方法において利用者に適したインタフェースを詳細に調節できるようにする必要がある．

より良い文字入力支援を実現するために今後取り組むべき課題としては，誤入力の原因解決と，誤入力を予測して修正案を提示できる予測変換モジュールの実装をすることが挙げられる．

6　おわりに

VR 環境上で文字入力を行う際，空間把握が難しいこと，人によって求める入力方法は異なること，誤入力が起きてしまうこと，という 3 つの課題から，本研究では物理フリック入力方法と指文字入力方法に予測変換を組み合わせた支援システムを設計した．

物理フリック入力方法にはフリック入力を採用しキーボードに触感を伴わせることで，空間把握が困難であるという課題の克服を試みた．フリック入力は現在広く普及していることから，VR 環境上でも普段と同じような入力を実現できる．指文字入力方法は手や指で作った形をカメラで画像認識することによって文字入力ができ，キーボード入力と違い空間把握をする必要がない．これら 2 つの方法に予測変換機能を組み合わせることで誤入力の補完を行った．

実験では提案した方法を用い，被験者 5 名に単語を入力させ，入力時間と誤入力回数の計測と使用感の評価を行った． 物理フリック入力方法はキーボードのずれによる誤入力が発生したが，物理的触感は操作感の向上につながった．一方で，入力方法の好みにはばらつきがあった．

利用者によって最も適した文字入力方法は異なる．本研究で提案したもの以外にも，異なる文字入力方法を加えて切り替え可能にすれば，より多くの利用者にとって使いやすい文字入力支援システムを提供できる．今後は，物理フリック入力方法のキーボードのずれの改善や，誤入力を予測して修正案を提示できるシステムの設計を行う．

謝辞　本研究の成果の一部は，JSPS 科研費（基盤（C） 20K11759），および 2022 年度南山大学パッヘ研究奨励金 I-A-1，I-A-2 の助成による．

参考文献

[1] Jiban Adhikary, Keith Vertanen, “Text Entry in Virtual Environments using Speech and a Midair Keyboard”, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 27, No. 5, pp. 2648-2658, 2021.

[2] 福仲 伊織，謝 浩然，宮田 一乗，“VR 環境におけるフリック入力形式インターフェースの開発”，情報処理学会研究報告，Vol. 2019-HCI-182，No. 3，pp. 1-8，2019.

[3] 河原 圭佑，鈴木 健嗣，“装着型機器を用いた指文字の音声翻訳による対話コミュニケーション支援”，第 77 回全国大会講演論文集，Vol. 2015，No. 1，pp. 619-620，2015.

[4] 長澤 直子，“大学生のスマートフォンと PC での文字入力方法 — 若者が PC よりもスマートフォンを好んで使用する理由の一考察 —”，コンピュータ＆エデュケーション，Vol. 43，pp. 67-72，2017.

[5] Haiyan Jiang, Dongdong Weng, Zhenliang Zhang, Yihua Bao, Yufei Jia, Mengman Nie, “HiKeyb: High-Efficiency Mixed Reality System for Text Entry”, *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 132-137, 2018.

Rust MIR に対する情報流解析の型システムの検討

Towards Type-based Information Flow Analysis for Rust MIR

桑原 寛明 *

Summary. This paper proposes a type system for information flow analysis of Rust MIR programs. Since Rust MIR syntax is much simpler than Rust surface syntax, we expect that focusing on MIR makes it possible to clearly formalize ownership and references of Rust in the context of information flow analysis. In this paper, we formalize type-based information flow analysis for Rust MIR including references, but leave an extension for ownership as future work.

1 はじめに

Rust 言語は，Mozilla で Web ブラウザエンジンを実装するために開発が始まったプログラミング言語であり，C/C++言語が主に利用されているシステムプログラミングに適している．メモリ安全性を静的に保証するための言語機能として所有権システムを備えており，二重解放やダングリングポインタの検出，マルチスレッドプログラムにおけるデータ競合の検出，リソースの自動解放などが実現されている．

メモリ安全性の保証はもちろん重要であるが，その他にもプログラムが満たすべき（満たしてほしい）性質が存在する．その一つは，プログラムが処理しているデータを不必要に漏洩させないことであり，その検査手法として型検査に基づく情報流解析が提案されている [1] [2] [3]．これはプログラムが機密データを外部に漏洩させる可能性がないことを静的に検査する手法であり，データの機密度を型として利用し，型付け可能なプログラムが非干渉性を満たすように型システムを構築する．非干渉性は，機密度の低いデータが機密度の高いデータに直接的および間接的に依存しないことを表し，機密データ自体に加え機密データを推測できる情報も漏洩させないという意味でよい性質である．

Rust プログラムに対する情報流解析が [4] で提案されている．Rust には様々な構文要素があることから Rust のサブセット言語を対象としているが，情報流解析と所有権システムを同時に形式化するため複雑な型システムとなっている．Rust プログラムでは，所有権システムに伴う所有権の借用のために参照を多用する傾向があるが， [4] では参照の意味の定義にバグが残っている．

本稿では，Rust プログラムのコンパイル過程で利用される中間表現である Rust MIR を対象とする型検査に基づく情報流解析を提案する．Rust MIR では，プログラムを制御フローグラフとして表現する．制御フローグラフのノードは基本ブロックであり，基本ブロックは代入文，関数呼び出し，基本ブロック間のジャンプから構成される．構文に着目すると通常の Rust プログラムと比較して非常に簡潔であるため，情報流解析と所有権システムを簡潔に形式化できることが期待される．本稿では，Rust MIR を情報流解析の形式化に必要な要素のみに絞り込んで構文と意味，および情報流解析のための型システムを定義する．なお，所有権システムへの対応は今後の課題として残されている．

2 Rust MIR

2.1 Rust プログラムの中間表現

Rust プログラムのコンパイル過程では，ソースコードから HIR(High-level IR)，MIR(Mid-level IR)，LLVM IR の順に複数の中間表現を経由し，最終的に LLVM を

*Hiroaki Kuwabara, 南山大学理工学部

$$
\begin{array}{rcl}
\eta & ::= & L \mid H \\
\tau & ::= & \eta \mid \&^{\eta}\tau \\
\mathit{MIR} & ::= & F+ \\
F & ::= & \mathsf{fn}\ f(\overline{\_n : \tau}) : \tau\ \{D\ \ \overline{\beta : B}\} \\
D & ::= & \mathsf{vardecl} : \{\overline{V};\ \mathsf{goto}\ \texttt{->}\ \mathsf{bb0};\} \\
V & ::= & \mathsf{let}\ [\mathsf{mut}]\ \_n : \tau \\
\beta & ::= & \mathsf{bb}n \\
B & ::= & \{\overline{S};\ T;\}
\end{array}
\qquad
\begin{array}{rcl}
S & ::= & P = R \\
T & ::= & \mathsf{goto}\ \texttt{->}\ \beta \\
 & \mid & \mathsf{switchInt}(O)\ \texttt{->}\ [\overline{z_i : \beta_i}, \mathsf{otherwise} : \beta] \\
 & \mid & P = f(\overline{O})\ \texttt{->}\ \beta \\
 & \mid & \mathsf{return} \\
P & ::= & \_n \mid *\_n \\
O & ::= & \mathsf{move}\ P \mid \mathsf{copy}\ P \mid \mathsf{const}\ z \\
R & ::= & O \mid \mathit{bop}(O_1, O_2) \mid \mathit{uop}(O) \mid \&^{\eta}[\mathsf{mut}]P
\end{array}
$$

図 1　MIR プログラムの構文

用いてコード生成が行われる．HIR は，構文糖衣の展開などが行われた抽象構文木相当の中間表現である．MIR は，基本ブロックをノードとする制御フローグラフ相当の中間表現である [5]．基本ブロックの中に分岐はなく，入れ子の式は一時変数を用いて展開される．MIR を用いて借用の検査や最適化が行われる．

2.2　構文

MIR の構文を図 1 に示す．MIR プログラムを対象とする情報流解析の形式化のため，Rust MIR のサブセットに型としての機密度の記述を加えている．

機密度定数 η は L と H の 2 段階とし，$L \sqsubseteq L, L \sqsubseteq H, H \sqsubseteq H$ を満たす機密度束 $(\{L, H\}, \sqsubseteq)$ を仮定する．τ は型としての機密度である．$\&^{\eta}\tau$ は参照の型であり，η は参照自体の機密度，τ は参照先の機密度を表す．$\sqsubseteq$ を次のように τ に拡張する．すなわち，$\&^{\eta_1}\tau_1 \sqsubseteq \&^{\eta_2}\tau_2$ **iff** $\eta_1 \sqsubseteq \eta_2 \wedge (\tau_1 = \tau_2 \vee (\tau_1 = \&^{\eta_1'}\tau_1' \wedge \tau_2 = \&^{\eta_2'}\tau_2' \wedge \tau_1 \sqsubseteq \tau_2))$ とする．MIR プログラムは関数宣言 F の並びである．関数宣言中の $\_n$ は仮引数名であり，ここで n は 0 以上の自然数である．MIR プログラム中に出現する変数は $\_0, \_1, \ldots$ であり，$\_0$ は返り値のために予約されている．関数の本体は，関数内に出現するローカル変数の宣言 D と，ラベル付き基本ブロック B のリスト $\overline{\beta : B}$ からなる．D は vardecl でラベル付けされたブロックで，0 個以上の変数を宣言してラベルが bb0 の基本ブロックにジャンプする．変数宣言では，変数 $\_n$ の機密度が τ であることを宣言する．基本ブロックは bb$n$ でラベル付けされており，0 個以上の代入文 S と次に実行する基本ブロックへのジャンプ文 T を含む．代入の左辺 P は変数 $\_n$ か参照解決 $*\_n$ であり，右辺 R は変数，参照解決，定数 z，2 項演算 bop，単項演算 uop，参照生成である．z は整数であり，プログラムが扱うデータを整数型に制限する．move P は値とともに所有権が移動すること，copy P は値を複製して所有権は移動しないことを表す．ジャンプ文 T について，goto は無条件ジャンプ，switchInt は O の値に基づく条件分岐，$P = f(\overline{O})$ -> β は関数呼び出しとジャンプ，return は関数からのリターンを表す．関数呼び出しの正常終了と異常終了とで分岐先が異なる可能性があるが，ここでは簡単のために異常終了の場合を考慮しない．

実用に際して，プログラム中の機密度は開発者が与えるが，開発者は MIR プログラムを直接編集しない．Rust プログラムに注釈やコメントとして機密度を記述し，それを MIR プログラムに移植するなどの方法を検討する必要がある．

2.3　意味

図 1 の構文に従う MIR プログラムの意味を定義する．変数環境 $\mathcal{E}$ を変数名からロケーションへの関数，メモリ $\mathcal{M}$ をロケーションから値への関数とする．変数環境は関数呼び出しごとに独立である．MIR プログラムには定数と演算に加えて参照生成があるため，値は整数またはロケーションである．

図 2 に式 P, O, R を評価する関数 eval の定義を示す．本稿では所有権システムに対応しないため，move P と copy P の意味は同一である．bop，uop はそれぞれ 2 項演算 bop と単項演算 uop を計算する関数である．図 3 に変数宣言ブロック，基本ブロック，代入文，ジャンプ文の意味の定義を示す．fresh は新しいロケーション

$$eval(P,\mathcal{E},\mathcal{M}) = \begin{cases} \mathcal{M}(\mathcal{E}(\_n)) & \text{if } P = \_n \\ \mathcal{M}(\mathcal{M}(\mathcal{E}(\_n))) & \text{if } P = *\_n \end{cases}$$

$$eval(O,\mathcal{E},\mathcal{M}) = \begin{cases} z & \text{if } O = \mathsf{const}\ z \\ eval(P,\mathcal{E},\mathcal{M}) & \text{otherwise} \end{cases}$$

$$eval(R,\mathcal{E},\mathcal{M}) = \begin{cases} eval(O,\mathcal{E},\mathcal{M}) & \text{if } R = O \\ \mathsf{bop}(eval(O_1,\mathcal{E},\mathcal{M}), eval(O_2,\mathcal{E},\mathcal{M})) & \text{if } R = bop(O_1,O_2) \\ \mathsf{uop}(eval(O,\mathcal{E},\mathcal{M})) & \text{if } R = uop(O) \\ \mathcal{E}(\_n) & \text{if } R = \&^{\eta}[\mathsf{mut}]\_n \\ \mathcal{M}(\mathcal{E}(\_n)) & \text{if } R = \&^{\eta}[\mathsf{mut}]*\_n \end{cases}$$

図 2　関数 $eval$ の定義

$$\frac{\mathcal{M}' = \mathcal{M}[\mathcal{E}(\_n) \mapsto eval(R,\mathcal{E},\mathcal{M})]}{\langle \_n = R,\mathcal{E},\mathcal{M}\rangle \to_{\mathsf{S}} \mathcal{M}'} \qquad \frac{\mathcal{M}' = \mathcal{M}[\mathcal{M}(\mathcal{E}(\_n)) \mapsto eval(R,\mathcal{E},\mathcal{M})]}{\langle *\_n = R,\mathcal{E},\mathcal{M}\rangle \to_{\mathsf{S}} \mathcal{M}'}$$

$$\frac{\alpha_i \text{ fresh} \quad i \in \{1,\ldots,l\} \quad \mathcal{E}' = \mathcal{E}[\ldots,\_i \mapsto \alpha_i,\ldots] \quad \mathcal{M}' = \mathcal{M}[\ldots,\alpha_i \mapsto \bot,\ldots]}{\begin{array}{c}\langle\{\overline{\mathsf{let}\ [\mathsf{mut}]\ \_n};\ \mathsf{goto}\ \texttt{->}\ \mathsf{bb0};\},(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \\ \to_{\mathsf{B}} \langle\{\mathsf{goto}\ \texttt{->}\ \mathsf{bb0};\},(\beta,P,\mathcal{E}',f)::cs,\mathcal{M}'\rangle\end{array}}$$

$$\frac{\langle P = R,\mathcal{E},\mathcal{M}\rangle \to_{\mathsf{S}} \mathcal{M}'}{\langle\{P = R;\overline{S};\ T;\},(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \to_{\mathsf{B}} \langle\{\overline{S};\ T;\},(\beta,P,\mathcal{E},f)::cs,\mathcal{M}'\rangle}$$

$$\frac{}{\langle\{\mathsf{goto}\ \texttt{->}\ \beta';\},(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \to_{\mathsf{B}} \langle block_f(\beta'),(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle}$$

$$\frac{\exists i.z_i == eval(O,\mathcal{E},\mathcal{M})}{\begin{array}{c}\langle\mathsf{switchInt}(O)\ \texttt{->}\ [\overline{z_i : \beta_i},\mathsf{otherwise} : \beta'];,(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \\ \to_{\mathsf{B}} \langle block_f(\beta_i),(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle\end{array}}$$

$$\frac{\forall i.z_i \neq eval(O,\mathcal{E},\mathcal{M})}{\begin{array}{c}\langle\mathsf{switchInt}(O)\ \texttt{->}\ [\overline{z_i : \beta_i},\mathsf{otherwise} : \beta'];,(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \\ \to_{\mathsf{B}} \langle block_f(\beta'),(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle\end{array}}$$

$$\frac{\alpha_i \text{ fresh} \quad \mathcal{E}' = [\ldots,\_i \mapsto \alpha_i,\ldots] \quad \mathcal{M}' = \mathcal{M}[\ldots,\alpha_i \mapsto eval(O_i,\mathcal{E},\mathcal{M}),\ldots] \quad i \in \{1,\ldots,l\}}{\begin{array}{c}\langle\{P' = f(O_1,\ldots,O_l)\ \texttt{->}\ \beta';\},(\beta,P,\mathcal{E},e)::cs,\mathcal{M}\rangle \\ \to_{\mathsf{B}} \langle block_f(\mathsf{vardecl}),(\beta',P',\mathcal{E}',f)::(\beta,P,\mathcal{E},e)::cs,\mathcal{M}'\rangle\end{array}}$$

$$\frac{\mathcal{M}' = \mathcal{M}[\alpha \mapsto eval(\_0,\mathcal{E}',\mathcal{M})] \quad \alpha = \begin{cases} \mathcal{E}(P') & \text{if } P = \_n \\ \mathcal{M}(\mathcal{E}(P')) & \text{if } P = *\_n \end{cases}}{\langle\{\mathsf{return};\},(\beta',P',\mathcal{E}',g)::(\beta,P,\mathcal{E},f)::cs,\mathcal{M}\rangle \to_{\mathsf{B}} \langle block_f(\beta'),(\beta,P,\mathcal{E},f)::cs,\mathcal{M}'\rangle}$$

図 3　MIR プログラムの意味

を表す．$\mathcal{E}[\ldots,\_i \mapsto \alpha_i,\ldots]$ は $\mathcal{E}$ について $\mathcal{E}(\_i)$ の値を α_i に更新することを表し，$\mathcal{M}[\ldots]$ も同様である．$\bot$ は値が未割り当てであることを表す．$block_f(\beta)$ は関数 f 内でラベルが β のブロックを表す．$\langle S,\mathcal{E},\mathcal{M}\rangle \to_{\mathsf{S}} \mathcal{M}'$ は代入文 S の意味であり，変数環境 $\mathcal{E}$ およびメモリ $\mathcal{M}$ の下で S を実行するとメモリが $\mathcal{M}'$ に変化することを表す．$\langle B,cs,\mathcal{M}\rangle \to_{\mathsf{B}} \langle B',cs',\mathcal{M}'\rangle$ は基本ブロック B の意味であり，コールスタック cs とメモリ $\mathcal{M}$ の下で B を 1 ステップ実行すると，基本ブロック，コールスタック，メモリがそれぞれ B'，cs'，$\mathcal{M}'$ に変化することを表す．コールスタックは関数呼び

出し順を追跡するためのデータ構造であり，関数から返る先の基本ブロックのラベル β，関数の返り値の代入先 P，仮引数とローカル変数の値を記録する変数環境 $\mathcal{E}$，実行中の関数の名前 f の 4 項組 $(\beta, P, \mathcal{E}, f)$ がスタックフレームである．

各ジャンプ文の意味は以下の通りである．goto -> β ではラベル β の基本ブロックに制御を移す．switchInt(O) -> $[\overline{z_i : \beta_i}, \mathsf{otherwise} : \beta']$ では O の値がいずれかの z_i と等しければラベル β_i，すべての z_i と等しくなければラベル β' の基本ブロックに制御を移す．$P' = f(\overline{O})$ -> β' では，関数 f の各仮引数に対応する新しいロケーション α_i を用いて，仮引数（とローカル変数）を格納する新しい変数環境 $\mathcal{E}'$ と実引数の値を格納したメモリ $\mathcal{M}'$ を準備し，呼び出す関数のためのスタックフレームをコールスタックに積んでから f のローカル変数を宣言するブロックに制御を移す．return では関数の返り値を指定された P' に代入し，コールスタックの先頭のスタックフレームを破棄して，返り先であるラベル β' の基本ブロックに制御を移す．

3　情報流解析のための型システム

情報流解析では，代入に伴う直接的な情報流 (explicit flow) と，条件分岐に伴う間接的な情報流 (implicit flow) を検査する．代入では，代入先の値から代入元の値がわかるため，代入元から代入先への情報流が存在し，代入元の機密度が代入先の機密度以下であればよい．関数呼び出しにおける引数の受け渡しも実引数から仮引数への代入とみなす．条件分岐では，分岐先において代入される変数の値を追跡することで分岐先，すなわち分岐の条件の値が判明する可能性がある．つまり，分岐の条件から分岐先で代入される変数への情報流が存在し，分岐の条件の機密度が分岐先で代入される変数の機密度以下であればよい．

Rust などの高水準言語では条件分岐は構造化されており，条件分岐に伴う情報流の到達範囲は構文木から特定できるため，条件分岐に対する型付け規則の制約は素直に記述できる．一方，MIR プログラムの場合，構文木としては基本ブロックがフラットに並んでいるだけで構造化されていない．条件分岐に伴う情報流の到達範囲の特定には制御フローグラフの解析が必要である．以上から，MIR プログラムに対する情報流解析のための型システムにおける型付け規則を図 4 のように定義する．

FDEC 規則が関数宣言，FBODY 規則が関数本体，BLOCK 規則が基本ブロック，ASSIGN 規則が代入文，GOTO 規則，RET 規則，SWITCH 規則，CALL 規則がジャンプ文，その他が式に対する規則である．$\eta^\dagger = \eta$，$(\&^\eta \tau)^\dagger = \eta$ であり，*ftype* は関数のシグネチャを取得する関数である．Γ は型環境であり，変数名からその機密度への関数である．関数に含まれるすべての基本ブロックが型付け可能であれば関数本体と関数宣言が型付け可能である．基本ブロック，代入文，ジャンプ文の型判定式 $\Gamma \vdash A : \tau$ は，型環境 Γ の下で A の実行に伴って代入される変数の機密度が τ 以上であることを表す．式の型判定式 $\Gamma \vdash e : \tau$ は，型環境 Γ の下で e の機密度が τ 以下であることを表す．

ASSIGN 規則では，代入元の機密度が代入先の機密度以下であることが制約である．関数呼び出しに対する CALL 規則では，実引数の機密度が仮引数の機密度以下であることと，返り値の機密度が返り値の代入先の機密度以下であることが制約である．これらの規則では，代入先の機密度を文の機密度とする．GOTO 規則，RET 規則，SWITCH 規則では，対象の文で代入が発生しないため文の機密度を H とする．

SWITCH 規則では，分岐の条件 O の機密度と分岐後に発生する代入における代入先の機密度を比較するために，O に関する情報流が到達する範囲を特定する必要がある．到達範囲は分岐先の基本ブロックに留まらず，それ以降の基本ブロックまで及ぶ可能性がある．例えば，ラベル bb1 の基本ブロックで bb2 と bb3 に分岐し，一方は bb2，bb4，bb6，bb7 の順に，他方は bb3，bb5，bb6，bb7 の順に基本ブロックを実行するとする．この場合 bb6 以降は双方で実行されるため区別できず，その手前の bb2，bb4，bb3，bb5 の基本ブロックが O に関する情報流の到達範囲となる．

$$\frac{\_0 : \tau, \overline{\_n : \tau} \vdash \{D \ \ \overline{\beta : B}\}}{\vdash \mathsf{fn}\ f(\overline{\_n : \tau}) : \tau\ \{D \ \ \overline{\beta : B}\}}\ [\mathrm{FDEC}]$$

$$\frac{\Gamma, \overline{\_n : \tau} \vdash B_i : \eta_i \quad i \in \{1, \ldots, l\}}{\Gamma \vdash \{\mathsf{vardecl} : \{\overline{\mathsf{let}\ [\mathsf{mut}]\ \_n : \tau;}\ \mathsf{goto}\ \texttt{->}\ \mathsf{bb0};\}\ \beta_1 : B_1\ \ldots\ \beta_l : B_l\}}\ [\mathrm{FBODY}]$$

$$\frac{\Gamma \vdash S_i : \eta_i \quad \Gamma \vdash T : \eta_t \quad i \in \{1, \ldots, l\}}{\Gamma \vdash \{S_1;\ \ldots;\ S_l;\ T;\} : \bigsqcap_i \eta_i \sqcap \eta_t}\ [\mathrm{BLOCK]} \qquad \frac{}{\Gamma \vdash \mathsf{goto}\ \texttt{->}\ \beta : H}\ [\mathrm{GOTO}]$$

$$\frac{\Gamma \vdash P : \tau_p \quad \Gamma \vdash R : \tau_r \quad \tau_r \sqsubseteq \tau_p}{\Gamma \vdash P = R : {\tau_p}^{\dagger}}\ [\mathrm{ASSIGN}] \qquad \frac{}{\Gamma \vdash \mathsf{return} : H}\ [\mathrm{RET}]$$

$$\frac{\Gamma \vdash O : \eta_o \quad \Gamma \vdash \mathit{block}_f(\beta_i) : \tau_i \quad \eta_o \sqsubseteq \tau_i \quad \beta_i \in \mathit{depend}(\overline{\beta}, \beta)}{\Gamma \vdash \mathsf{switchInt}(O)\ \texttt{->}\ [\overline{z : \beta}, \mathsf{otherwise} : \beta] : H}\ [\mathrm{SWITCH}]$$

$$\frac{\begin{array}{l}\tau_1, \ldots, \tau_l \to \tau_r = \mathit{ftype}(f) \quad \Gamma \vdash P : \tau_p \quad \tau_r \sqsubseteq \tau_p \\ \Gamma \vdash O_i : \tau_{o_i} \quad \tau_{o_i} \sqsubseteq \tau_i \quad i \in \{1, \ldots, l\}\end{array}}{\Gamma \vdash P = f(O_1, \ldots, O_l)\ \texttt{->}\ \beta : {\tau_p}^{\dagger}}\ [\mathrm{CALL}]$$

$$\frac{}{\Gamma \vdash \_n : \Gamma(\_n)}\ [\mathrm{VAR}] \qquad \frac{\Gamma \vdash \_n : \&^{\eta}\tau}{\Gamma \vdash *\_n : \tau}\ [\mathrm{DEREF}] \qquad \frac{\Gamma \vdash P : \tau}{\Gamma \vdash \&^{\eta}[\mathsf{mut}]P : \&^{\eta}\tau}\ [\mathrm{REF}]$$

$$\frac{\Gamma \vdash P : \tau}{\Gamma \vdash \mathsf{move}\ P : \tau}\ [\mathrm{MOVE}] \qquad \frac{\Gamma \vdash P : \tau}{\Gamma \vdash \mathsf{copy}\ P : \tau}\ [\mathrm{COPY}] \qquad \frac{}{\Gamma \vdash \mathsf{const}\ P : L}\ [\mathrm{CONST}]$$

$$\frac{\Gamma \vdash O_1 : \eta_1 \quad \Gamma \vdash O_2 : \eta_2}{\Gamma \vdash \mathit{bop}(O_1, O_2) : \eta_1 \sqcup \eta_2}\ [\mathrm{BOP}] \qquad \frac{\Gamma \vdash O : \eta}{\Gamma \vdash \mathit{uop}(O) : \eta}\ [\mathrm{UOP}]$$

図 4　型付け規則

SWITCH 規則における *depend* は，分岐の条件の情報流が到達する範囲の基本ブロック集合を求める関数である．

depend の定義を以下に示す．着目している関数 f の基本ブロックをノードとする制御フローグラフを G とし，G のノードを基本ブロックのラベル β により表す．ラベルが β の基本ブロック $\{\overline{S};\ T;\}$ に対し，

$$\mathit{next}(\beta) = \begin{cases} \{\beta'\} & \text{if } T = \mathsf{goto}\ \texttt{->}\ \beta' \text{ or } T = P = f(\overline{O})\ \texttt{->}\ \beta' \\ \{\overline{\beta}, \beta'\} & \text{if } T = \mathsf{switchInt}(O)\ \texttt{->}\ [\overline{z : \beta}, \mathsf{otherwise} : \beta'] \\ \emptyset & \text{if } T = \mathsf{return} \end{cases}$$

とする．β を始点とする有限の実行経路 $p = \beta_0\beta_1 \ldots \beta_l$ は，$\beta_0 = \beta$，$\mathit{block}_f(\beta_l) = \{\overline{S};\ \mathsf{return};\}$，$\forall i \in \{0, \ldots, l-1\}.\beta_{i+1} \in \mathit{next}(\beta_i)$ を満たす．実行経路 p の i 番目のラベルを $p(i)$，$\overline{\beta}$ に含まれる各ラベルを始点とする有限の実行経路の集合を $\mathit{paths}(\overline{\beta})$ と書く．$\overline{p}$ に含まれるすべての実行経路に出現するラベルを $\overline{p}$ の合流点と呼び，$\mathit{paths}(\overline{\beta})$ の合流点の集合を $\mathit{merge}(\overline{\beta}) = \{\beta' \mid \forall p \in \mathit{paths}(\overline{\beta}).\exists i.p(i) = \beta'\}$ と定義し，$\mathit{paths}(\overline{\beta})$ の最初の合流点の集合を $\mathit{fmerge}(\overline{\beta}) = \{\beta' \mid \forall p \in \mathit{paths}(\overline{\beta}).\exists i.p(i) = \beta' \wedge \forall j < i.p(j) \notin \mathit{merge}(\overline{\beta})\}$ と定義する．以上より，

$$\mathit{depend}(\overline{\beta}) = \{\beta' \mid \exists p \in \mathit{paths}(\overline{\beta}).\exists i.p(i) \in \mathit{fmerge}(\overline{\beta}) \Rightarrow \forall j < i.p(j) = \beta'\}$$

と定義する．直観的には，分岐したすべての実行経路が初めて合流する基本ブロッ

```
fn f(b : &bool) -> i32 {
  let i;
  if *b {
    i = 1;
  } else {
    i = 0;
  }
  i
}
```

図 5　Rust プログラムの例

```
fn f(_1 : &H) : L {
  vardecl : {
    let mut _0 : L; let mut _2 : H;
    goto -> bb0; }
  bb0 : { _2 = *_1;
    switchInt(move _2) ->
      [0 : bb2, otherwise : bb1]; }
  bb1 : { _0 = const 1; goto -> bb3; }
  bb2 : { _0 = const 0; goto -> bb3; }
  bb3 : { return; }
}
```

図 6　図 5 に対応する MIR プログラム

クが最初の合流点であり，各実行経路について分岐先から最初の合流点の直前までのすべての基本ブロックの集合が到達範囲である．

4　例

例として図 5 の Rust プログラムに対応する図 6 の MIR プログラムを考える．紙幅の都合で詳細は省略するが，f の呼び出しを図 3 に従って実行すると適切な返り値が得られる．図 6 では，機密度が H の変数 _2 に基づいて分岐した先で機密度が L の変数 _0 に代入しており，不正な情報流が存在するため型付け不能である．switchInt 文に対して図 4 の SWITCH 規則を適用すると，$\mathit{depend}(\mathsf{bb1}, \mathsf{bb2}) = \{\mathsf{bb1}, \mathsf{bb2}\}$ であり，いずれのラベルの基本ブロックも BLOCK 規則から型が L である．一方，分岐の条件 move _2 の機密度は H であるため条件を満たさず，型付けできない．

5　おわりに

本稿では，Rust プログラムの中間表現である MIR を対象とする情報流解析のための型システムを提案した．MIR プログラムの意味定義は [6] を，SWITCH 規則の *depend* は [7] を参考にした． [6] は Rust MIR に対するテイント解析を， [7] は JVM を簡略化したアセンブリ言語向けの情報流解析のための型システムを提案している．

今後の課題として，非干渉性に対する健全性の証明，可変性 (mutability) への対応，ムーブセマンティクス（所有権の移動）への対応と情報流解析との関係の明確化，機密度の指定方法の検討，検査器の実装などが挙げられる．

謝辞　本研究の一部は 2022 年度南山大学パッヘ研究奨励金 I-A-2 の助成による．

参考文献

[1] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security*, Vol. 4, No. 2, pp. 167–187, 1996.

[2] Andrei Sabelfeld and Andrew C. Myers. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communications*, Vol. 21, No. 1, pp. 5–19, 2003.

[3] 黒川翔, 桑原寛明, 山本晋一郎, 坂部俊樹, 酒井正彦, 草刈圭一朗, 西田直樹. 例外処理付きオブジェクト指向プログラムにおける情報流の安全性解析のための型システム. 電子情報通信学会論文誌 D, Vol. J91-D, No. 3, pp. 757–770, 2008.

[4] 長谷川健太, 桑原寛明, 國枝義敏. Rust プログラムの情報流解析のための型システム. 信学技報, Vol.119, No.452, SS2019-53, pp. 73–78, 2020.

[5] Guide to Rustc Development. `https://rustc-dev-guide.rust-lang.org/mir/index.html`. 2022/09/13 参照.

[6] Emil Jørgensen Njor and Hilmar Gústafsson. Static Taint Analysis in Rust: Using Rusts Ownership System for Precise Static Analysis. Master's thesis, Department of Computer Science, Aalborg University, 2021.

[7] 小林直樹, 白根慶太. 低レベル言語のための情報流解析の型システム. コンピュータソフトウェア, Vol. 20, No. 2, pp. 118–137, 2003.

形式手法を用いたPID制御装置の検証

Verification of PID controller using formal methods

浦岡　竜太郎 * 伊藤　宗平 †

あらまし 一つのエラーが致命的になるセーフティクリティカルシステムでは，安全性や信頼性を保証するために形式手法による検証が国際規格で推奨されている．近年では自動運転システムの制御システムのような物理空間と相互作用するサイバーフィジカルシステムの安全性・信頼性の検証技術が注目を集めている．セーフティクリティカルかつサイバーフィジカルな自動運転システムの中でも特に重要な駆動系の制御を行うものの一つに PID 制御方式がある．PID 制御はゲインと呼ばれるパラメータの設定が安全性に影響を及ぼすため，検証を行うことが望ましい．本研究では，PID 制御装置及びモータをハイブリッドオートマトンでモデル化し，SpaceEx を用いて電圧及びオーバーシュートについての安全性検証を行う手法を提案する．

1　背景と目的

セーフティクリティカルシステムとは，一つのエラーや誤作動で人命や人々への深刻な被害，または機器へのダメージや環境被害を与える可能性のあるシステムのことである．セーフティクリティカルシステムの安全性や信頼性を保証し，リスクを最低限に軽減するために形式手法による検証が国際規格で推奨されている．形式手法とは，検証の対象となるシステムと検証したい性質の両方を数学的に厳密に意味づけられた記述で表現することで，システムがその性質を満たしているかを検証する手法である．近年では，セーフティクリティカルシステムの一つである自動運転システムの開発が自動車業界でも進んでおり，その制御に関わるソフトウェアの規模も増大の一途をたどっている．こういった物理空間と相互作用する計算システムはサイバーフィジカルシステムと呼ばれており，サイバーフィジカルシステムの安全性・信頼性の検証技術が注目を集めている．

自動運転システムの中でも駆動系の制御を行うものに PID 制御方式がある．PID 制御は自動車の速度という連続的な状態変化を扱う一方で，目標とする車速の設定や自動車の転回などは離散的な状態変化である．このように離散ダイナミクスと連続ダイナミクスを併せ持つシステムをハイブリッドシステムと呼び，ハイブリッドシステムを形式的にモデル化したものとして，ハイブリッドオートマトンがある．ハイブリッドオートマトンは危険な状態へと到達しないという性質である安全性の検証を近似的に行うことができる [1]．ハイブリッドオートマトンの検証ツールとしては，SpaceEx が知られている [2].

SpaceEx を使用した類似の研究には一輪車の自動倒立制御 [3] や走行制御システム [4] を題材にした研究がある．前者はコントローラとして PID 方式のモデルも提案しているが，線形システムとして表現できていないため SpaceEx では検証できなかった．後者については，PI 制御モデルを使用して検証を行なっていた．いずれも完全な PID 制御器を SpaceEx では実装できていなかった．

本研究では，FPT ‘ 21 FPGA デザインコンテスト [5] において実際に使用された自動運転ロボットに搭載されている PID 制御装置を対象として要求される安全性の検証を行う．まず PID 制御装置及びその制御対象である DC モータを SpaceEx model editor [2] [6] を用いて作成する．作成したモデルについて，危険状態への到達可能性を検証し安全性を評価する．本論文の構成は次のようになっている．第 2 節

*Ryutaro Uraoka, 長崎大学

†Sohei Ito, 長崎大学

では，本研究における検証の対象となる PID 制御及びハイブリッドオートマトン，SpaceEx について述べる．第 3 節では実際に検証する自動運転ロボットの概要及び自動運転ロボットのモデルについて述べる．第 4 節では検証の結果及び考察について述べる．そして，第 5 節でまとめと今後の課題を述べる．

2 準備

2.1 PID 制御について

PID 制御とは，その出力が次の入力値に影響を及ぼすフィードバック制御の一種であリ，目標値と現在の値の偏差をもとに制御量を計算するものである．最大の特徴としてはこの偏差について比例・積分・微分の計算を行い，それらの結果の和を制御量として出力するところにある．時刻 t での制御量を $u(t)$，偏差を $e(t)$ とすると，次の式で表せる．

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D e'(t)$$

式中の，それぞれ比例・積分・微分ゲインと呼ばれる定数 K_P,K_I,K_D の値の取り方によっては車輪の回転量が目標値を大きく上回ってしまうオーバーシュートと呼ばれる現象や目標値の設定から実際にその値に到達するまでの反応時間が大きくなるといった安全性に影響を及ぼすような問題が発生してしまう事がある．そのため，これらのゲインは適切に設定される必要がある．すなわち，設定されたゲインによる制御が安全であるかを検証することが望ましい．

2.2 ハイブリッドオートマトン

ハイブリッドオートマトンはハイブリッドシステムを表現するモデルの一種である．ハイブリッドオートマトンは有限状態オートマトンに微分方程式を組み込んだもので，連続状態遷移と離散状態遷移を併せ持つことが特徴である．ハイブリッドオートマトンの定義については，例えば [7] を参照していただきたい．

2.3 SpaceEx とは

SpaceEx はハイブリッドシステムを対象とした，安全性・到達可能性検証プラットフォームである．SpaceEx には到達可能性検証アルゴリズムがいくつか用意されているが本研究では STC Support Function と呼ばれるものを用いる [8]．本研究が検証の対象としているような安全性（危険な状態に到達しない）を検証するために，到達してはいけない状態 (Forbidden states) を定義することができる．

STC Support Function(以下 STC scenario) は，ハイブリッドオートマトンの到達可能性検証アルゴリズムの一種である [8]．STC scenario は SpaceEx でも使用できる LGG Support function [9] の改良版である．

ある状態集合から，一定時間内に到達可能な状態の集合をフローパイプと呼ぶが，フローパイプを厳密に求めることは難しい．そこで，与えられたサンプリングタイムを用いた数値シミュレーションにより到達可能な状態の集合を包含する凸包を計算して，フローパイプを過大近似するアルゴリズムが STC scenario である．定義した危険状態が近似したフローパイプ内に存在しなければ，システムが危険な状態に到達しないという事が証明できる．フローパイプの計算は無限に行うことができないため，フローパイプには時間の上限を定める必要がある．SpaceEx ではこの時間の上限のことを local time horizon と呼ぶ．

3 自動運転ロボットとそのモデル化

3.1 自動運転ロボットの概要

本研究における検証対象となる自動運転ロボットについての概要を説明する．自動運転ロボットは左右の二輪からなり，それぞれ別々の PID 制御装置がある．その

ため本研究では，一つの車輪についてモデル化し，検証する．このロボットの PID 制御において制御する対象は車輪の回転量であり，これは単位時間 (0.5s) あたりの車輪の回転角度を度数法で表したものである．まず，制御ソフトウェアより目標となる回転量 (ここでは $accel$ とする) が PID 制御装置に入力される．PID 制御装置は現在の車輪の回転量 (ここでは fb とする) との偏差 $e = accel - fb$ と目標値を用いて制御量 u を計算し，PWM モジュールへと渡す．PWM モジュールでは，PID 制御装置から届いた制御量 u をもとにして，実際にモータに与える電圧を計算する．モータの回転量はロータリーエンコーダからの信号より計算され PID 制御装置の入力となる．このようにして，PID 制御装置によって計算された制御量によって実際のモータの回転量が変化し，そのフィードバックをもとにして再び制御量を計算するという仕組みとなっている．

3.2 検証する性質

制御量 u は PWM モジュールに渡され，PWM モジュールはパルスのデューティ比 ($\frac{\text{パルス幅}}{\text{パルスの周期}}$) に比例した電圧をモータにかけるモジュールである．ここではパルスの周期が $2^{14} - 1$，パルス幅が制御量 u となっている．すなわち制御量 u によって電圧を調節できるようになっている．この時，電圧が最大値の 95 %を超えると挙動が不安定になる恐れがある．また．フィードバックの値のオーバーシュート，アンダーシュート（目標値を超えて下回ること）についても，目標値を大きく超えてしまうと安全とは言えない．よって，以下の 2 点について検証を行う．

1.制御量 u がパルス周期の 95 %を超えないか．
2.フィードバックのオーバーシュート，アンダーシュートについては目標値の 5 %を超えないか．

3.3 モータの数理モデル化

モータの SpaceEx モデルを作成するために，自動運転ロボットに搭載された車輪の回転量を ω，モータの電流を I，モータにかかる電圧を V として，モータの振る舞いを以下のように数理的にモデル化した [10]. なお k_τ,b,J,R,K_b,L は定数である．

$$\omega' = \frac{k_\tau I - b\omega}{J}$$
$$I' = \frac{V - RI - K_b\omega}{L}$$

3.4 PID 制御装置及びモータの SpaceEx モデル作成

次に，モータ及び PID 制御装置を SpaceEx Model Editor を用いてハイブリッドオートマトンとしてモデル化を行う．なお，本研究におけるモデル化は SpaceEx に依存したモデルによるものではないことに注意されたい．以下では，SpaceEx Model Editor を用いて作成したモデルの概要を述べる．

3.4.1 モータの制御フィードバック系全体のモデル化

3.1 節で述べたフィードバック系全体のモデルを図 1 に示す． SpaceEx モデルのブロックには，ハイブリッドオートマトンを表現するブロックと，それらからなるシステムを表すブロックの二種類がある．特に，図 1 中において後者に当たるものは，PID 制御装置を表す PIDcont_1，モータを表す plant_1 が該当する．clock_1 はグローバルクロックであり，経過時間によって目標とする回転量を変更するために用いる．accel_1 ブロックは目標値をタイミングによって変更するブロックであり，具体的な自動運転ロボットの走行シナリオを定める．検証する性質によって走行シナリオは変更することができる．

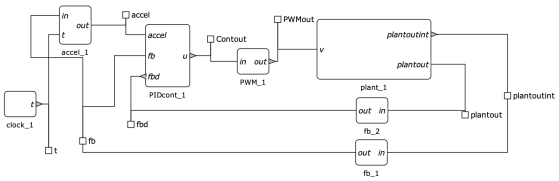


図 1　フィードバック系全体のモデル

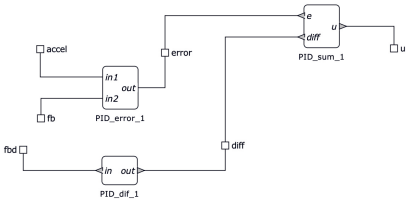


図 2　自動運転ロボットに搭載された PID 制御装置の SpaceEx モデル

3.4.2　PID 制御装置のモデル化

FPT ‘ 21 FPGA デザインコンテストにおいて実際に使用された自動運転ロボットに搭載されている PID 制御装置で用いるゲインは $K_p = 13, K_I = 11, K_d = 2$ である．これらの値を元にその PID 制御装置をモデル化したもの (図 1 中の PIDcont_1 に対応) を図 2 に示す．PID_error_1 は目標値と現在の回転量との偏差 $error(= accel - fb)$ を計算するブロックであり，これが e として PID_sum_1 に入力されている．PID_sum_1 は PID の三項の和を計算して制御量 u を出力するブロックであり，本 PID 制御の核となるブロック (ハイブリッドオートマトン) である．これを図 3 に示す．

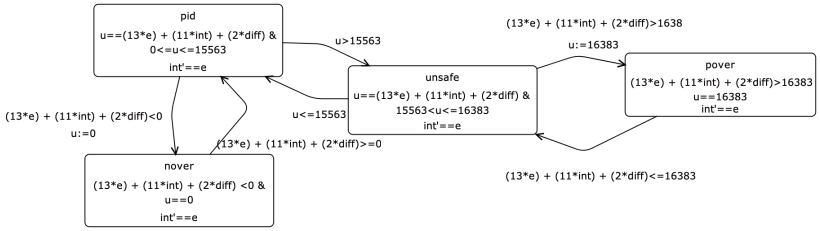


図 3　三項の和を取るブロック PID_sum_1

PID_sum_1 の初期ロケーションは pid で，インバリアントとして制御量 u が PID の制御式で計算されている．u が PWM 制御のパルス周期の 95 %を超えると unsafe ロケーションに遷移し，さらにパルス周期を上回ること (正のオーバーフロー) があれば，pover ロケーションへの遷移が発生し，u は 14bit の最大値に固定される．逆に，u が 0 を下回ること (負のオーバーフロー) があれば nover ロケーションへの遷移が発生し，u が 0 に固定される．pover, nover は制御量 u が最大値，最小値を超えた場合の飽和処理を表したロケーションである．unsafe ロケーションへ到達しないことを確かめることで，検証項目の 1 について検証できる．上図の e は *error* を，*int* は *error* の時間積分を，*diff* が *error* の時間微分を表している．ここで積分項 *int* は，*flow* として定義することができる．一方，PID 制御の微分項 *diff* は $diff == e'$ として定まるがこれは *flow* として表現できないため，PID の三項の計算を単一のハイブリッドオートマトンでは記述できなかった． [3] で実装されていたモデルでは，微分項が $e' == diff$ の *flow* で表現されており，本来定義されるべき *diff* が右辺に現れているため，SpaceEx での解析が不可能であった．本モデルでは，$e' = (accel - fb)' = -fb' = -\omega'$ を利用して，PID_diff_1 ブロックでは $\omega'(\omega$ はモータの回転量，図中の *fbd* に相当) を受け取り $-fbd$ を出力し，PID_sum_1 に *diff* として入力したモデルにすることで，この問題を解決している．

4　検証結果と考察

検証する性質に応じた 2 つのシナリオを考える．一つ目は，図 4 に示した方法で目標値を変更するシナリオで，性質 1 を検証する．このシナリオは停止状態から回転量が accel1 を超えるまで加速し，その後カーブを曲がるために accel2 まで減速し，さらに再び accel1 まで加速し，最終的には停止するようなシナリオである．これは典型的な車両走行シナリオであるため，この場合で性質 1 を満たすことを確かめることは有益である． 二つ目は，図 8 のようなシナリオで，各目標値を設定して

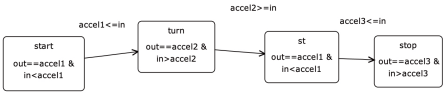


図 4　制御量が安全な範囲に含まれているかを検証するシナリオ

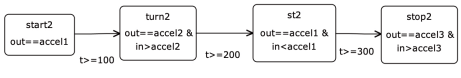


図 5　オーバーシュートの範囲を検証するシナリオ

から十分長い時間を取ることで，性質 2 を検証する．一つ目のシナリオでは車輪の回転量が目標値を超えるタイミングで目標値の変更を行なっているため，フィードバックのオーバーシュート範囲を検証するには不適であったためこのようなシナリオを考える．

どちらのシナリオでも初期状態における全ての連続変数の値は 0 とし，離散遷移が 10 回起こった時点で検証を終了し，t-omegaout(モータの回転量の時間変化) グラフを出力する．離散遷移の回数については，実験的に 10 回で速度 0 の状態に到達することが確かめられたため，PID 制御の検証としては十分と判断した．一つ目のシナリオでは，local time horizon を 60 として検証を行い，Forbidden states として loc(PID_sum_1) == unsafe を定義した．つまり，ブロック PID_sum_1 は unsafe ロケーションに到達しないことを確かめる．local time horizon が 60 である理由はどのフローパイプも 60s 以内に図 5 の目標値設定オートマトンにおいて離散遷移が起こることが実験的に確かめられたからである．すなわち目標速度の変更が起きた時の振る舞いを検証するために十分な長さの時間となっている．

図 6 で示したものが出力されたグラフであり，フィードバックが取りうる値 (到達可能状態) を表したものである．到達可能性解析の結果としては，“Forbidden states are not reachable.” と出力され，PID_sum_1 は unsafe ロケーションには到達しない，すなわち検証項目 1 が満たされることが確認できた．二つ目のシナリオでは，十分長い時間をとってオーバーシュートについて検証するため local time horizon を 110 として検証を行い，Forbidden states として $omegaout > 3.66519$ と loc(accel_1)==turn2 & $omegaout < 2.65290$ を定義した．ここで 3.66519 は，度数法の 105 を有効数字 5 桁で弧度法として表した時の値であり，2.65290 は度数法の 76 を有効数字 5 桁で弧度法として表した時の値である．105，76 はそれぞれ目標値 100，80 に対して許容されるオーバーシュートの最大値，アンダーシュートの最小値であり，これらの値を超えなければ性質 2 について検証ができる．到達可能性解析の結果としては，“Forbidden states

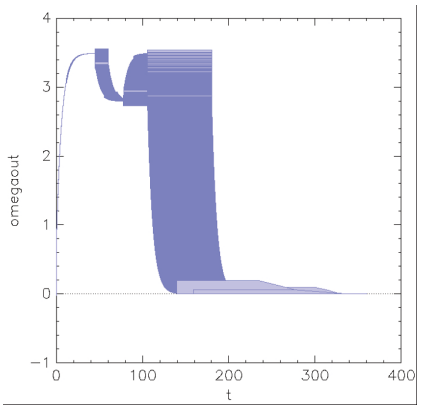


図 6　STC support function で求めたシナリオ 1 でのシステムのフィードバック

are not reachable.” と出力され，検証項目 2 が満たされることが確認された．

以下に考察を述べる．図 6 では，目標値が 0 のとき，色の薄い凸包ができているが，これは nover への遷移が起こったことが原因であると考えられる．また，本研究では検証対象ではない性質であるが，目標値近辺に到達するまでに約 30 秒かかっていることから，PID 制御の反応性という点では実用的ではないと考えられる．

5　まとめと今後の課題

本研究では，ハイブリッドオートマトンを用いて PID 制御装置及びモータの形式的モデル化と安全性の形式的検証を行う手法を提案した．本手法で作成したモータのモデルに対して，PID 制御装置の安全性に関する二つの仕様を検証する事ができた．ハイブリッドオートマトン及び SpaceEx を用いることの利点としては，連続状態として実数値を扱うことができることや動作シナリオが離散遷移を用いて定められること及び危険状態を定義して到達可能性解析が可能であることが挙げられる．

今後の課題としては，様々な動作シナリオや外乱を考えたモデルで検証を行うことで，様々な条件下でも安全性の仕様を満たすかどうかを確かめることが考えられる．また，4.2 節で述べた反応性について，モータのモデル化に使用したパラメータは実機 (自動運転ロボット) から実験的に求めたものではない．そのため，モデルの方に問題がある可能性があり，モータのモデル化についても更なる改良の余地があると考えられる．

謝辞　本研究は JSPS 科研費 JP 21K11756 の助成を受けたものです．また，本研究を行うにあたり，検証の対象となる自動運転ロボットのデータを提供してくださった柴田裕一郎教授ならびに柴田研究室の皆様に心から感謝を申し上げます．

参考文献

[1] Platzer, A., Clarke, E.M.: The Image Computation Problem in Hybrid Systems Model Checking. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) *HSCC2007*. LNCS, vol. 4416, pp. 473–486. Springer, Heidelberg, 2007.

[2] Goran Frehse, Colas Le Guernic, Alexandre Donzè, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler: SpaceEx: Scalable Verification of Hybrid Systems. International Conference on Computer Aided Verification(*CAV2011*), pp. 379-395, 2011.

[3] Felix Freiberger and Holger Hermanns: On the Control of Self-Balancing Unicycles, Proceedings Workshop on Models for Formal Analysis of Real Systems (*MARS2015*), pp.25-36 ,2015.

[4] Nikolaos Kekatos: Verifying a Cruise Control System using Simulink and SpaceEx, *arxiv:2101.00102[cs.LO]*,31 Dec 2020.

[5] Keigo Motoyoshi, Yuta Imamura, et al.: SoC FPGA implementation of an unmanned mobile vehicle with an image transmission system over VNC, *FPT'21*, pp.1-4, 2021.

[6] Scott Cotton, Goran Frehse and Olivier Lebeltel: The SpaceEx Modeling Language, December 8, 2010, http://spaceex.imag.fr/documentation/user-documentation.

[7] Goran Frehse: An Introduction to Hybrid Automata, Numerical Simulation and Reachability Analysis, Formal Modeling and Verification of Cyber-Physical Systems, pp.50-81,2015.

[8] Goran Frehse, Verimag: Brief Experimental Comparison of the STC and LGG Analysis Algorithms in SpaceEx, November 26, 2012, http://spaceex.imag.fr/documentation/user-documentation.

[9] Colas Le Guernic and Antoine Girard: Reachability Analysis of Hybrid Systems Using Support Functions, Computer Aided Verification: 21st International Conference (*CAV2009*), pp.540-554, 2009.

[10] Rajeev Alur : Principles of Cyber-Physical Systems,The MIT Press, 2015.

検証パターンに注目した機械学習に基づくモデル検査手法の評価

Evaluation of model checking method based on machine learning focusing on verification pattern

張 超群* 岸 知二†

あらまし モデル検査技術の検証結果を，機械学習を用いて予測する手法が提案されている．本稿では検証プロパティの類型が予測精度に及ぼす影響について評価する．

Summary. A method for predicting the model checking results using machine learning has been proposed. This paper evaluates the effect of the patterns of validation property on prediction accuracy.

1 研究の背景と目的

モデル検査（Model Checking）は形式手法のひとつであり，プログラムや設計の正しさを数理論理学に基づき全数検査をすることによって検証するために用いられる技術である.1980 年代に Edmund M Clarke，Mien Emerson[1]や Joseph Sifakisc[2]らによって提案され，現在では様々な分野で利用されている．

モデル検査の利用における課題のひとつに状態爆発問題(state explosion problem)がある．対象システムの複雑化にともない状態空間が大きくなり，現実的な時間での探索ができなくなる問題である．現実の適用においては深刻な問題であり，様々な取り組みがなされている．

機械学習は因果関係を捉えることが困難な問題に対しても適切な判断を行うことが可能なため様々な分野への適用が研究されているが，論理の世界への適用はそれほど進められていなかった．しかしながら近年，例えば Hahn[3]らの研究では，ディープラーニングのニューラルネットワークを用いてLTL 式を解く実験を行い良い結果が得られたことが報告されている．

Zhu ら[6]は機械学習を用い，検証対象のシステムモデルとプロパティからモデル検査の結果を推定する手法を提案し，状態爆発の問題を回避しつつ一定精度の推定結果が得られることを示している．推定結果には疑陽性・偽陰性が含まれるため，モデル検査技術を代替するものではないが，短時間で品質を推定するなどの応用が期待される．この手法の有効性，妥当性については評価が必要だが，我々はその一環として，精度の改善方法について検討している．

本稿では，モデル検査で用いられるプロパティには一定の類型があることに注目し，検証に用いるプロパティを定型的なプロパティパターンに限定にすることによる精度の向上可能性についての初期の評価結果を示す．

2 従来研究

2.1 モデル検査

モデル検査は，システムの状態空間を全数探索することによりシステムが重要なプロ

* Chaoqun Zhang,早稲田大学
† Tomoji Kishi，早稲田大学

パティを満たすかどうかを検証する技術である．プロパティが満たされない場合には反例が出力される．モデル検査には様々な手法があり，例えば Lee[4]らによる BDD 手法などがある．検証するシステムのプロパティを記述するためには時相論理が用いられ，具体的には CTL(computation tree logic)， LTL (linear-time temporal logic)[5].などが用いられる．

2.2 モデル検査と機械学習の組み合わせ

Zhu ら[6]は，機械学習とモデル検査を組み合わせた手法を提案している．この手法では複数の Kripke 構造と LTL 式で記述されたプロパティに対してモデル検査を行い，その結果でラベル付けして学習モデルを作成し，これを用いて未知の Kripke 構造とプロパティのモデル検査の結果を推定する．手法の全体像を図 1 に示す.

Boosted Tree(BT)，Random Forest(RF)，Decision Tree(DT)，Logistic Regression(LR)など機械学習のアルゴリズムをそれぞれ用いることで LTL モデル検証の結果を予測し，比較的良い結果を得た．この手法は状態爆発の問題はなく一定の精度の推定ができるという利点があるが，その結果には擬陽性・偽陰性が含まれるためモデル検査技術を代替することはできない．また反例も得られない．しかしながら短時間で大量のプロパティを推定することができるので，品質の推定などへの応用が期待される．

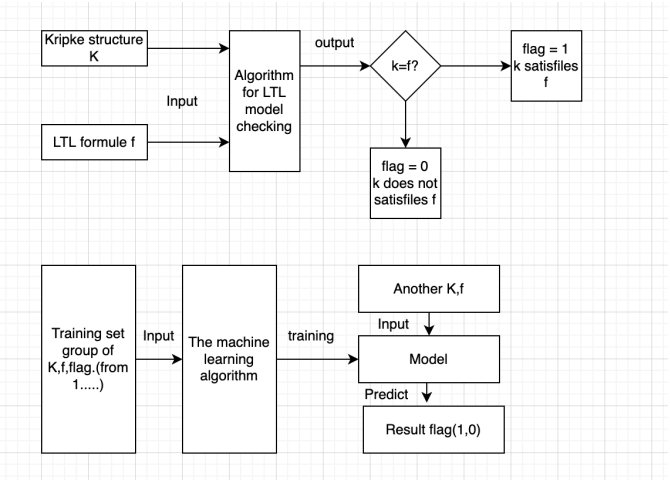


図 1　zhu らの手法のイメージ[6]

3　問題設定

3.1　プロパティパターンの利用

先行研究では機械学習による手法がモデル検査に有効であることが示されているが，この手法ではランダムに生成した Kripke 構造とプロパティを用いている．しかしながら現実のモデル検査で用いられるプロパティには一定の類型があると考えられる．そこで学習や推定に用いるプロパティをそうした類型にあてはまるものに限定することで，予測精度がどのように変わるかを調べる．

3.2　課題

3.1 に基づき，以下の Research Question を設定する．

RQ1: 学習データ中のプロパティパターンと予測データ中のプロパティパターンが同じ場合の精度はどうなるか.

RQ2: 学習データ中のプロパティパターンと予測データ中のプロパティパターンが違

う場合の精度はどうなるか.

4　評価実験

4.1　実験設定

3 章での RQ に答えるために，評価実験を行う．ここでは，Dweyer の時相仕様のパターン[7]から 3 つを選び，以下の設定で予測精度の比較を行う．

実験 A：3 つのパターンを混在して学習と予測をする．比較のベースラインとする．

実験 B ：学習データのパターンと異なるパターンで予測をする．

実験 C：学習データのパターンと同じパターンで予測をする．

4.2　実験の概要

本実験では，モデル検査の対象となるシステムモデルと，LTL で記述されたプロパティを用いてモデル検査を行い，プロパティが成り立つかどうかを確認する．次にそれらのシステムモデル，プロパティ，モデル検査の結果から実験 A, B, C のためのデータセットを構築して学習モデルを作り，それらの予測精度を比較する．次節以降でそれぞれについて説明する．

4.3　システムモデルの準備

本実験では異なる 6 個のシステムモデルを対象とする．システムモデルは Kripke 構造で，それぞれ 3 つの状態を持ち，3 つの原子命題を含む.

4.4　プロパティの準備

本実験では，Dwyer[7]によって提案されたプロパティパターンから標準的な LTL 式のパターンである Absence, Universality, Existence の 3 つのパターンを選択した．Absence, Universality，Existence はそれぞれある区間の中，「ずっとない」，「ずっとある」，「ある」を表す.各パターンには 5 つの LTL 式が含まれているので，15 の LTL 式を実験に用いた．なお，各 LTL 式は 3 つの命題変数を含んでいる．

4.5　モデル検査の実施

システムモデルごとに，15 の LTL 式を使ってモデル検査を行い，結果を得た．なお各 LTL 式は 3 つの命題変数を持ち，システムモデルは 3 つの原始命題を持つため，当てはめ方は 6 通りあるが，今回の実験ではその中の 3 通りを用いた.

4.6　データ構築

Kripke 構造を Zhu ら[6]と同様の方法で数字列としてデータ化する。図 2 は本実験で用いた一つ Kripke 構造のひとつである。ここで状態中の数字は 3 つの原始命題の真偽を表している．データ化においては，各状態での原始命題の真偽を列挙し(この例では s0:011,s1:110,s2:110)次に遷移関係（この例では s0s1:01,s1s2:12,s2s0:20）を列挙する．従ってこの Kripke 構造は 011110110011220 という数字列になる．

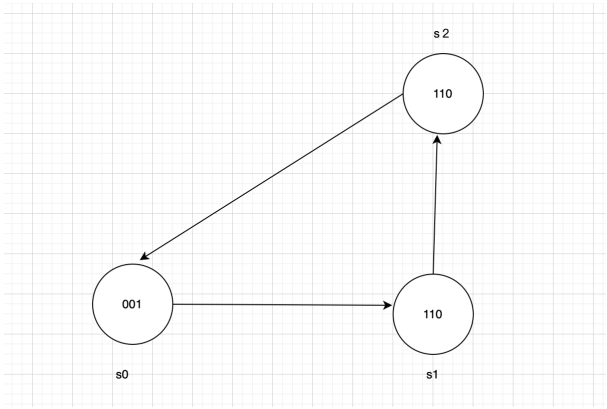


図 2　Kripke 構造の例

プロパティのデータ化の方法も先行研究と同様であり，LTL 式に Kriple 構造中の原始命題をあてはめた文字列をデータとした．モデル検査の結果のデータ化も先行研究と同様であり，性質の成立・不成立を 1 と 0 で表した．以上より，270 個のデータを作成した．

4.7　データセット構築

4.6 で作られたデータは 3 つのプロパティパターン Absence, University, Existence それぞれに 90 個ずつとなる．これらから，以下の 4 種類のデータを作る。

- ALL：すべてのデータ（3 つのパターンに対応したデータをすべて含む）
- AbandUn：Absence と Universality のパターンに対応したものだけからなるデータ
- UnandEx：Universality と Existence パターンに対応したものだけからなるデータ
- AbandEx：Absence と Existence パターンに対応したものだけからなるデータ

上記を用いて，各実験のためのデータセットを以下のように作る．

- 実験 A（3 つのパターンの混在）

ALL をシャフルした後，80%を訓練セット，残り 20%をテストセットとする．

- 実験 B（学習と予測が異なるパターン）

2 つのプロパティパターンに対応した 180 データのデータを訓練セットとし，別のプロパティパターンからランダムに 45 データを抽出してテストセットとした．たとえば，AbandUn を訓練セットとして選択した場合, 訓練セットには Absence と Universality のすべてのデータ 180 個が含まれており，Existence からランダムに 45 個のデータを抽出してテストセットとする．3 種類の訓練・テストセットのペアが作られる．

- 実験 C（学習と予測が同じパターン）

2 つのプロパティパターンに対応したデータセットから訓練セットとテストセットを作る，例えば Absence と Existence の二つのプロパティパターンの場合には，AbandEx の 80%を訓練セットとし，残りをテストセットとする．3 種類の訓練・テストセットのペアが作られる．

上記のデータセットより，実験 1, 実験 2, 実験 3 のデータを表 1 のように構築した．

4.8　モデル作成と予測

訓練セットを用いてモデルを作成し，テストセットで予測を行う．Zhu らが用いたアルゴリズムから分類問題によく用いられる DT と LR を用いた．システムモデルと LTL

式を feature として用い，検証結果が正しいかどうかをラベルとする．その上で，モデル検査結果を予測する機械学習モデルを構築する.なお，各実験は 5 回ずつ行い，最もよい精度を選択した.

表 1　実験に用いたデータセット

	訓練セット	テストセット
実験 A	ALL からランダムに 80%	ALL の残り 20%
実験 B	AbandEx	Universality からランダムに 45 個
	AbandUn	Existence からランダムに 45 個
	ExandUn	Absence からランダムに 45 個
実験 C	AbandEx からランダムに 80%	AbandEx の残り 20%
	AbandUn からランダムに 80%	AbandUn の残り 20%
	ExandUn からランダムに 80%	ExandUn の残り 20%

5　実験結果と考察

5.1　実験結果

DT による実験結果を表 2 に，LR による実験結果を表 3 に示す.

表 2　DT での結果

	データセット	精度（四捨五入，２位）
実験 A	ALL	75.93%
実験 B	AbandEx　＋　Universality	65.22%
	AbandUn　＋　Existence	73.91%
	ExandUn　＋　Absence	63.04%
実験 C	AbandEx	69.44%
	AbandUn	75%
	ExandUn	72.22%

表 3　LR での結果

	データセット	精度（四捨五入，２位）
実験 A	ALL	74.08%
実験 B	AbandEx　＋　Universality	71.74%
	AbandUn　＋　Existence	71.74%
	ExandUn　＋　Absence	73.91%
実験 C	AbandEx	75%
	AbandUn	72.22%
	ExandUn	75%

RQ1 に対しては，プロパティが同じ実験 C は，プロパティが異なる実験 B より高い精度が示されている．一方実験 A との比較では LR で同程度，DT ではやや低い結果となった.

RQ2 に対しては，プロパティが異なる実験 B は，実験 A，実験 C いずれと比較しても

低い精度となった.

5.2 考察

今回の実験では訓練セットとテストセットでのプロパティが同じ方が高い精度が得られた．訓練セットとテストセットでのプロパティのパターンが予測に影響する可能性が示唆されるものと考えられる.

一方，プロパティの組み合わせによって精度の違いがある．またベースランとした実験 A との違いは明確ではない. 実験 A のデータ数が多いことが影響している可能性もあり，実験設定のさらなる工夫が必要である.

アルゴリズムの違いも実験によって異なっており，本実験からは明確な違いは判断できなかった. 両アルゴリズムは近い精度が得られているが，DT アルゴリズムは全体として精度が LR ほど安定しておらず，全体の精度がやや低い．これは DT アルゴリズムが複数のフィーチャーを持つデータを分類した結果に偏っているためとも考えられる.

なお，すべてのデータセットにおいて，Absence ではモデル検査結果ラベルの T と F の数がほぼ同数だが，Existence と Universality では F よりも T の数が多かった．こうしたことが，実験 C で，いずれのアルゴリズムにおいても ExandUn が比較的よい精度を示していることに影響しているとも考えられる.

今回は初期的な評価として小さなデータセットしか用いておらず，結果の正確性は不十分であるが，訓練セットとテストセットでプロパティが同じ方が，精度が高くなる傾向は確認された．今後データ量やパターンを増やすことで，より信頼性，正確性の高い評価を進めたい.

6 まとめ

本稿では機械学習によってモデル検査の結果を予測する手法に関し，プロパティパターンの影響について初期の評価結果を報告した．本手法の有用性については現時点では未知数であるが，高速で予測できるため，一定の精度での予測ができるなら活用の可能性があると考えられる．今後さらなる評価を進めていきたい.

参考文献

[1] Clarke E M, Emerson E A. Design and synthesis of synchronization skeletons using branching time temporal logic[C]//Workshop on Logic of Programs. Springer, Berlin, Heidelberg, 1981: 52-71.
[2] Queille J P, Sifakis J. Specification and verification of concurrent systems in CESAR[C]//International Symposium on programming. Springer, Berlin, Heidelberg, 1982: 337-351.
[3] Hahn C, Schmitt F, Kreber J U, et al. Teaching Temporal Logics to Neural Networks[J]. arXiv preprint arXiv:2003.04218, 2020.
[4] Lee C Y. Representation of switching circuits by binary-decision programs[J]. The Bell System Technical Journal, 1959, 38(4): 985-999.
[5] Pnueli A. The temporal logic of programs[C]//18th Annual Symposium on Foundations of Computer Science (sfcs 1977). IEEE, 1977: 46-57.
[6] Zhu, Weijun, Huanmei Wu, and Miaolei Deng. "LTL model checking based on binary classification of machine learning." IEEE Access 7 (2019): 135703-135719.
[7] https://matthewbdwyer.github.io/psp/patterns/ltl.html

プログラムのベクトル化と記号実行を活用した正誤判定の効率化

An Efficient Judgement of Programs Using Vectorization and Symbolic Execution

大嶋　琉太[*]　阿萬　裕久[†]　川原　稔[‡]

あらまし　近年，プログラミング学習に対するニーズの高まりに伴って，オンラインジャッジシステムのように自主学習に利用できるシステムが注目されている．オンラインジャッジシステムでは，出題されたプログラミング問題に対して，プログラムの正誤を自動判定するために多数のテストケースが使用される．しかしながら，そのようなテストケースを人手で準備することは容易な作業ではなく，記号実行を用いた支援が研究されてきている．本論文は，記号実行を用いたプログラムの自動正誤判定技術に着目し，その効率化に向けた提案と評価を行っている．具体的には先行研究で行われていたプログラムの特徴ベクトル化について，従来の N-gram を用いた手法を Doc2Vec に置き換えることを提案し，19156 個のプログラムを対象とした評価実験を行っている．

1　はじめに

近年，プログラミングが小学校で必修化される等，プログラミングに対する注目度や IT 人材に対する需要が高まってきている．それに伴い，プログラミング学習を支援する Web 上のサービス等も増加してきている．そのような学習支援の 1 つとしてオンラインジャッジシステムが挙げられる．オンラインジャッジシステムとは，プログラミングコンテストにおいて解答プログラムの正誤を自動判定するためのシステムである．コンテスト終了後であっても，そこでの問題とオンラインジャッジシステムを一般に公開しているサイト（Codeforces [1] 等）もあり，プログラミング学習で活用される場面も多くなっている．

オンラインジャッジシステムでプログラムの正誤を自動判定するには，あらかじめテストケースを人手で用意しておかなければならない．通常，プログラムの書き方は人によって千差万別であり，それゆえ用意すべきテストケースも多種多様なものが求められる．したがって，そのような作業は決して容易なものではない．この問題を解決する 1 つの手段として，プログラムの記号実行技術 [2] の活用が挙げられる．記号実行とは，プログラムにおける変数を具体値を持たない “記号” として扱い，すべての実行経路をシミュレーションすることでプログラム解析を行う技術である．これによって，与えられたプログラムのあらゆる実行パターンを効率的に検出でき，テストケースの自動生成を支援できる．そして，記号実行を活用することで人手でテストケースを用意することなくプログラムの正誤判定を行う手法が知られている．しかしながら，記号実行は決して軽量な処理ではないため，そこでの処理コストが別の問題として挙げられる．この問題に対し Rastogi ら [3] は，機械学習を記号実行と併用することで処理コストを削減する手法を提案しているが，同時に “本来は不正解のプログラムを正解と見なしてしまう” という事例が少なくないことが課題となっていた．そこで本論文では，Rastogi らの手法における機械学習の活用部分を再検討し，可能な限り記号実行の回数を削減しつつ，上述の誤判定数も減らすための方法を提案する．

[*]Ryuta Oshima, 愛媛大学大学院理工学研究科電子情報工学専攻

[†]Hirohisa Aman, 愛媛大学総合情報メディアセンター

[‡]Minoru Kawahara, 愛媛大学総合情報メディアセンター

2　正誤判定の効率化: プログラムの類似度評価の活用

2.1　記号実行を活用したプログラム正誤判定

いま，あるプログラミング問題に対し，その解答として提出されたプログラム P_s の正誤を判定したいとする．一般的なオンラインジャッジシステムでは，テストケース集合を用意しておき，それらすべてのテストケースについて期待通りの動作をするならば P_s を正解，さもなくば不正解と判定する．その際，当該プログラミング問題で問われている内容について，出題者等がさまざまな動作を想定し，それらを網羅的に確認できるテストケースを用意しなければならない．これに対し，出題者等が模範解答プログラム P_t を用意しておき，P_s が P_t と等価な動作を行うかどうかを調べることで P_s の正誤を判定するという考え方もある．この場合，図 1 に示すような等価性判定プログラムを作成しておき，さまざまな入力データに対して P_s と P_t が常に同じ結果を出力するかどうかを確認する．記号実行エンジンは関数 `f1` と `f2` の内容，即ち P_s と P_t の内容をそれぞれ網羅的に実行できる入力値を自動的に見つけ出し，実行する．そして，もし P_s と P_t で出力が異なるような入力データが見つかればその時点で記号実行は終了となり，同時に P_s が不正解であることも分かる．一方，P_s が正解プログラムならば，P_s と P_t の両プログラムの全実行可能経路が試された後で記号実行が正常に終了することになる．あわせて，その際に使用された入力データは他のプログラムのためのテストデータとしても使用できる．つまり，記号実行を活用することで人手によるテストケース生成が不要となる．

2.2　先行研究

Rastogi ら [3] は，上述した記号実行による正誤判定法に対し，機械学習を活用した改善案を提案している．まず，文献 [3] では比較のベースラインとして，次の手法が示されている（便宜上，**従来手法 1** と呼ぶ）：

(1) 模範解答プログラム P_t を用意する．テスト入力データ集合 T を空とする．
(2) n 個の評価対象プログラム $P_{s,i}$ について以下の（3），(4）を繰り返す（$i = 1, 2, \ldots, n$）．
(3) T に含まれるすべての入力データで $P_{s,i}$ をテストする（P_t と同じ出力かどうかを確認する）．P_t と異なる出力が得られた場合は $P_{s,i}$ を不合格とする．なお，T が空の場合は何もせず手順（4）へ進む．
(4) 手順（3）において $P_{s,i}$ が不合格にならなかった場合，記号実行を行って $P_{s,i}$ と P_t の等価性を調べる．あわせて，得られたテスト入力データを T へ追加する．

```
関数 f1 ( x1 ){
   y1 = [ Ps(x1) の処理 ];
   return y1;
}
関数 f2 ( x2 ){
   y2 = [ Pt(x2) の処理 ];
   return y2;
}
main(){
   x1, x2, ..., xn の読み込み;
   for ( i = 1 ... n ) {
     if ( f1(xi) ≠ f2(xi) ) 不正解と判断してプログラムを停止;
   }
   上で停止されなければ正解と判断;
}
```

図 1　P_s と P_t の等価な動作を確認することで P_s の正誤を判定

上述の従来手法 1 は，記号実行解析をすべてのプログラム（テストに失敗したものを除く）に対して実施する手法であり，模範解答プログラムとの等価性判定でもって正誤判定を行うものであるが処理コストの点で懸念がある．そこで Rastogi らは，n 個の評価対象プログラムの一部である $n_0(< n)$ 個 $P_{s,1}$, $P_{s,2}$, …, P_{s,n_0} については従来手法 1 を行い，いったん n_0 個のプログラムを “正解プログラムの集合（A）” と “不正解プログラムの集合（W）” に分割する．その後，残りの $n - n_0$ 個のプログラムに関しては，手持ちのテストケース（テスト入力データ）集合 T ではいずれも正しく動作したプログラムについて，その時点での正解プログラム（$\in A$）との特徴の類似性を評価し，一定以上類似していれば記号実行解析を省略して “正解とみなす” という方法を採っている．なお，文献 [3] では n_0 として $n \times 0.1$ に最も近い整数を採用している．本論文では便宜上，これを**従来手法 2** と呼ぶ．

従来手法 2 では，プログラムの特徴を Bag-of-N-gram [4] を用いてベクトル化し，正誤をラベルとして XGBoost [5] で学習している．そして，そのモデルが出力する “正解プログラムの見込み値” （特徴の類似性の高さ）が一定以上であれば “正解とみなす” という方針を採っている．なお，判定の閾値は交差検証によって決定される．あわせて，判別モデルの精度を高めるため，一定回数判定を行った後にはモデルの構築と閾値の設定をやり直すという処理も含まれる．

2.3 先行研究における課題点と提案手法

従来手法 2 では，機械学習による正解可能性の予測が適切に機能すれば記号実行の回数は大きく削減できると期待できる．しかし，算出される正解可能性の精度が低いと “本来は不正解であるはずのプログラムを誤って正解と見なしてしまう” ことが懸念される．さらに，従来手法 2 における特徴ベクトル生成が “Bag-of-N-gram” という単純なものになっている点も気がかりである．多数のプログラムを処理しなければならないことを考慮すると軽量な特徴ベクトル化が望まれるところではあるが，Bag-of-N-gram が必ずしも優れた手法であるとは言い切れない．あわせて，判定の過程で未学習の N-gram に遭遇した場合，それらを特徴ベクトルへ反映できないため機械学習で適切に正解可能性を予測できるかは疑問である．

上述した課題を解決するため，従来手法 2 で採られていた機械学習モデルの活用箇所を再検討し，次のように置き換えた手法を新たに提案する：

(1) 機械学習モデルの入力として使用する特徴ベクトルの生成には Bag-of-N-gram を使うのではなく，Doc2Vec [6] を使用する． Doc2Vec はニューラルネットワークを使って文書の分散表現を生成するモデルであり，意味的に近い文書がベクトル空間上でも近いベクトルとなることが知られている．本論文では文脈を考慮できるという理由から PV-DM という学習モデルを採用する．

なお，Bag-of-N-gram については未知語が無視されてしまう問題を指摘したが，PV-DM で学習した Doc2Vec モデルは未知語に対して乱数を使って対応する．そのことが良い効果を持つかどうかは定かではないが，少なくとも無視はされないこと，並びに Doc2Vec が文書の類似性判定で有用なモデルとして広く使われていることを考慮してこの手法を採用した．なお，文書のベクトル化手法には他にも様々な手法が提案されているが，他の手法の活用については今後の課題としたい．

(2) 機械学習モデルの再学習は行わず，正解可能性に関する閾値はあらかじめパラメータとして与える． 従来手法 2 では一定回数の判定を行うたびに機械学習モデルの再学習を行っていたが，これは Bag-of-N-gram のシンプルさに起因する特徴量の単純さを考慮したものではないかと筆者は考える．一方，Doc2Vec では分散表現を使ってより細かく文書（この場合はプログラム）の内容を表現できるため，事前にある程度多くのデータで学習させておけば，その後は再学習させなくともうまく機能するのではないかと期待できる．そして，再学習を行わないため閾値も再決定する必要はないと考え，あらかじめパラメータとして与えることにする．なお，適切なパラメータについては実験を通じて検討していくことにする．

3 評価実験

本実験では，Codeforces 上のプログラミング問題に対して投稿された多数の解答プログラムについて，従来手法並びに提案手法を用いた正誤判定実験を行う．そして，実行効率並びに判定精度の観点から比較を行い，結果について検討する．なお，本実験では先行研究に倣い，記号実行エンジンとして KLEE [7] [8] を使用する．

3.1 予備実験

KLEE による記号実行では，入力値に対応するすべての変数を記号として扱う必要がある．それゆえ，多数の入力を必要とするプログラムの場合，組合せ数が膨大になり計算機資源の観点から見て解析が困難となる．そこで本実験では，先行研究 [3] と同様に簡単な入出力を扱った問題に限定して議論を進めることにする．具体的には，筆者の環境（表 1）において現状で表 2 に示す問題については KLEE による解析が可能であることを確認できている．しかし，一部のプログラムをサンプルとして抽出して KLEE による記号実行を行ったところ，解析が長時間に及んでしまう事例が多く見られた．そこで予備実験として，サンプルプログラムに対する記号実行を行い，それに要する時間（実時間）を調べた．ただし，60 分経過しても結果が得られない場合はタイムアウトとして処理した．そして，その結果をもとに “すべてのプログラムに対して記号実行解析を行った場合に要すると推定される時間” を見積ると，問題 4A については約 3 時間ですべてのプログラムを解析できるが，それ以外は現実的な時間では完了が見込めないことが分かった．本実験では，従来手法と提案手法を比較する上で，条件を変えながら複数回の正誤判定実験を行う必要があるため，問題 4A を対象として比較実験を行っていくことにする．

3.2 手順

従来手法 1，従来手法 2，及び提案手法それぞれによってプログラムの正誤判定実験を行い，次の 2 つを記録する：

- そこで要した KLEE 解析の回数（便宜上，*N_KLEE* と呼ぶ）
- 誤って不正解プログラムを正解と見なしてしまった件数（便宜上，*FP* と呼ぶ）

ただし，判定対象プログラムの処理順序が結果に影響する可能性があるため，乱数を使ってプログラムの番号（処理順序）はシャッフルする．なお，3 つの手法の間では条件を揃えるために同じ乱数の種を与えることにし，乱数の種を変えて同じ実験を 10 回繰り返してそれらの平均でもって結果を比較する．加えて，提案手法に関しては閾値 τ を $0.90 \sim 0.99$ の範囲で 0.01 刻みで変化させながらそれぞれ（10 種類の乱数で）実験する．便宜上，これらを “提案手法（$\tau = 0.95$）” 等と呼ぶ．

表 1 実験環境

項目	内容
仮想化環境	Docker 20.10.6
CPU	Intel Core i5-9400 2.4GHz
メモリ	8GB(∗1)
記号実行エンジン	KLEE 2.3
ビットコードコンパイラ	Clang 9.0.1

(∗1) Docker に割当てたメモリ容量

表 2 筆者の環境でプログラムの変換と記号実行が可能であることを確認できている問題

問題 ID	投稿数	概要
4A	19156	1 つの整数を入力とし，それに応じた文字列を出力する
189A	523	4 つの整数を入力とし，それらに応じた整数を 1 つ出力する
863A	128	1 つの文字列を入力とし，それに応じた文字列を 1 つ出力する
894A	448	1 つの文字列を入力とし，それに応じた整数を 1 つ出力する

3.3 結果

従来手法 1 及び 2 による正誤判定実験の結果として得られた *N_KLEE* 及び *FP* を表 3 に示す．ただし，表中の値は 10 種類の乱数を用いた結果の平均である．提案手法の目的は，*N_KLEE* を可能な限り減らしつつ，*FP* も少なく抑えることである．つまり，図 2 のように従来手法 1 の結果（点 C_1）と従来手法 2 の結果（点 C_2）を結んだ線分 C_1C_2 を考え，提案手法の結果が線分 C_1C_2 よりも原点側の領域に位置することが望まれる．そこで本実験では，提案手法（閾値として τ を使用）の結果が平面上の点 Q に位置していた場合に Q から C_1C_2 へ垂線を下ろし，その足を R とする．そして，線分 QR の長さでもって提案手法の良さを評価する．便宜上，以下では QR の長さを δ_τ と呼ぶ．評価値 δ_τ を算出すると表 4 に示す通りとなった．

結果として，実験で用いたいずれの閾値 τ についても提案手法は図 2 でいうところの線分 C_1C_2 よりも原点側の領域に位置する点 (*N_KLEE*, *FP*) に対応しており，*N_KLEE* 及び *FP* の 2 つの値をバランスよく削減できた．そして，表 4 より，2 つの値を最もバランスよく削減できるのは $\tau = 0.99$ とした場合であった．今回の評価値はあくまでも *N_KLEE* の削減率と *FP* の削減率の両方を同等に重要なものとして扱っているため，図 2 に示したように線分 C_1C_2 からの距離を採用したが，いずれかの重みを変えて議論することも可能である点に注意されたい．

3.4 考察

今回の実験を通じて明らかになった課題点について述べる．N-gram を使った特徴ベクトルの生成準備はトークン列を整理するだけで済むためほとんど時間はかからないが，Doc2Vec を使った場合は最初にモデル構築が必要となる．さらには各プログラムを特徴ベクトルに変換して XGBoost による正解可能性の算出を行う場合にも，ベクトル化にかかる計算コストに違いが生じる．従来手法 2 と提案手法それぞれにおいて，これらに要した時間を計測したところ平均的には表 5 に示す通りであった．ベクトル化の準備として，Doc2Vec では多数のプログラムのトークン列を入力として学習を行う必要があるため，N-gram の場合に比べて大きく時間がかかっている．XGBoost の学習・再学習にかかる時間は逆に減少しているが，これは提案手法が再学習を行わないためである．最後に XGBoost による予測に要した時間はわずかに提案手法の方が長くなっているが，特徴ベクトルの長さの違いがわず

表 3　*N_KLEE* 及び *FP*

	従来手法 1	従来手法 2	提案手法（$\tau = 0.99$）
N_KLEE	8373.5	3035.3	5143.9
FP	0	717.0	299.9

表 4　提案手法の評価値

τ	0.90	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	**0.99**
δ_τ	65.72	70.33	59.78	65.81	80.66	99.73	102.88	114.03	125.23	**132.69**

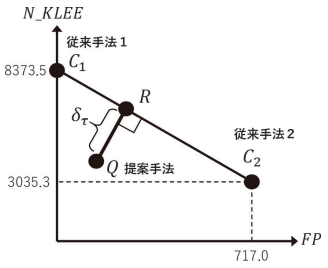


図 2　提案手法の評価値 δ_τ

表 5　ベクトル化や機械学習に要する時間の比較（単位：秒）

項目	従来手法 2	提案手法
ベクトル化の準備	0.1	93
XGBoost の学習・再学習	56	32
XGBoost による予測時間	54	57
合計	110.1	182

かに影響したのではないかと推察される．ベクトルの長さは N-gram の場合は実験における乱数に依存するところがあるが，概ね 100 次元程度となっている．一方，Doc2Vec では 150 次元[1]となっており，この違いが影響したのではないかと思われる．結果として，問題 4A を対象とした実験では提案手法の方がベクトル化及び機械学習に要する時間が 72 秒程度長くなっている．しかしながら，記号実行のコストが大きい問題となれば，この差は誤差の範囲として処理できる可能性もある．

4　まとめと今後の課題

一般に，プログラムの正誤判定には多数のテストケースが必要とされるが，模範解答となるプログラムを出題者が用意しておけば，その後は解答プログラムとの等価性を記号実行によって判定するだけで正誤判定並びにテストケース生成を自動化できるという利点がある．しかしながら，記号実行という処理は決して軽量な処理ではないため，より少ない処理回数ですべてのプログラムの正誤判定を行う手法が従来から研究されてきている．本論文では，プログラムのベクトル表現と機械学習を用いた判定手法に着目し，従来は N-gram を使っていた部分を Doc2Vec に置き換えることを提案した．プログラミングコンテストサイト Codeforces から入手した 19156 個のプログラムを対象とした比較実験の結果，提案手法は記号実行の回数削減と誤判定事例の削減という 2 つの目標を効果的に達成できることを確認できた．

今後の課題として，多様なプログラミング問題に対して提案手法を適用し，その効果の一般性を確認することが挙げられる．また，今回はベクトル化手法として Doc2Vec を用いたが，Code2Vec や CodeBERT を用いた実験も行って判定精度の向上について検討していきたい．

謝辞　本研究の一部は JSPS 科研費 20H04184，21K11831，21K11833 の助成を受けたものです．

参考文献

[1] Codeforces. https://codeforces.com.
[2] 酒井政裕, 岩政幹人. 記号実行によるプログラム改造支援技術. 東芝レビュー, Vol. 67, No. 12, pp. 35–38, Dec. 2012.
[3] Ishan Rastogi, Aditya Kanade, and Shirish Shevade. Active learning for efficient testing of student programs. In Penstein Rosé C. et al., editor, *Artificial Intelligence in Education*, Vol. 10948 of *Lecture Notes in Computer Science*, pp. 296–300. Springer, 2018.
[4] 金明哲. テキストアナリティクスの基礎と実践. 岩波書店, 東京, 2021.
[5] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proc. 22nd ACM SIGKDD Int'l Conf. Knowledge Discovery & Data Mining*, pp. 785–794, Aug. 2016.
[6] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proc. 31st Int'l Conf. Machine Learning*, Vol. 32, pp. 1188–1196, June 2014.
[7] Cristian Cadar, Daniel Dunbar, and Dawson Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proc. 8th USENIX Conf. Operating Systems Design & Implementation*, pp. 209–224, Dec. 2008.
[8] KLEE. https://klee.github.io, 2009.

[1]本実験では，50 – 200 次元の範囲で 50 刻みにしてモデル構築を試し，同じトークン列を 2 つ入力した場合に両者のコサイン類似度が常に 0.95 以上となるような最小のパラメータに選定した．

Pythonにおける機械学習関連ライブラリの自動推薦手法の評価

An Evaluation of automatic recommendation methods for machine learning libraries in Python

小柳 慶 * 秋山 楽登 † 沖野 健太郎 ‡ 近藤 将成 § 鵜林 尚靖 ¶

あらまし 近年，機械学習や深層学習の研究が盛んに行われ，Python においては機械学習ライブラリが多数活用されている．しかし，利用可能なライブラリの数は年々増加し，必要なライブラリを発見することに労力を費やしてしまう場合が多い．そこで，Java のライブラリ推薦手法を参考に，協調フィルタリングを用いて Python における機械学習関連ライブラリを推薦し，従来の推薦手法が Python においてどの程度の精度で推薦可能かを調べるための追実験を行った．追実験の結果，Java における推薦結果と比較して，評価指標値の推移は類似した傾向を示し，従来の推薦手法を Python に適用した場合も同様の推薦結果が得られることがわかった．

1 はじめに

近年，機械学習を活用したソフトウェアシステム開発が盛んである．そのシステム開発では機械学習のモデルを新規にゼロから構築するのではなく，関連ライブラリを活用して実装することが多い．ライブラリの活用は実装の短縮に寄与するため，ソフトウェア開発の効率化に繋がる．しかし，活用可能なライブラリ数は増加傾向にあり，また，それぞれ異なった用途を持つ．そのため，開発者が自身の目的と合致するライブラリをどのように発見するかが問題となっている．この問題解決のために，開発者に対するライブラリ推薦に関して様々な手法が提案されているが，評価対象言語のほとんどは Java である [1] [2] [3]．一方で，機械学習分野においては Python が利用されることが多い．そこで筆者らは，Java に対する提案手法を Python に応用することを考え，Nguyen らの提案手法 [1] を簡易化した推薦手法を構築し，初期評価を行った [4]．しかし，我々の先行研究 [4] では，(1) データセットが小規模であったこと，(2) 構築した推薦手法が簡易的であったことが課題であった．

本稿では実験内容の信頼性の向上を図るため，データセットの拡張および Nguyen らの提案手法を適用し，追加で調査を実施した．

2 背景と目的

2.1 本研究の目的

Java を用いたシステム開発におけるライブラリ推薦は盛んに研究されているが，Python ではあまり研究されていない．Java の手法を Python に応用できれば，Python を用いたシステム開発の効率化が見込める．本稿では，Nguyen らの手法をもとに推薦システムを構築し，推薦精度およびライブラリによって推薦の優位性が存在するのか調査した先行研究 [4] の追加実験を行った．各調査課題を以下に示す．

RQ1 Python において機械学習関連ライブラリをどの程度の精度で推薦できるのか

目的 近年，ソフトウェアシステム開発におけるデータ分析の手段として，機械学

*Kei Koyanagi, 九州大学
†Gakuto Akiyama, 九州大学
‡Kentaro Okino, 九州大学
§Masanari Kondo, 九州大学
¶Naoyasu Ubayashi, 九州大学

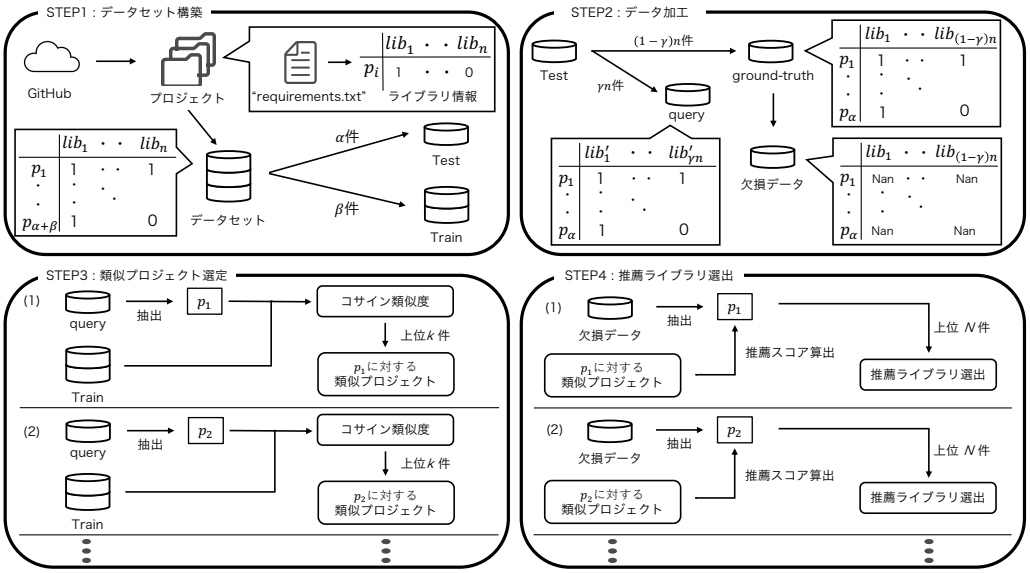


図 1　推薦システムの概要

習や深層学習が積極的に使用されている．そこで，システム開発の効率化のために，Python の豊富な機械学習ライブラリを開発者に自動で推薦することを考えた．本稿では，先行研究 [1] の提案手法の対象を Python の機械学習関連ライブラリに限定して調査を行う．対象を変更した場合の評価指標値の変化を明らかにすることで，今後の Python を用いた推薦システム構築のための手がかりを得る．

RQ2　各ライブラリを使用しているか否かの情報量の違いにより，推薦精度に変化が生じるのか

目的　先行研究では，開発段階のライブラリの使用状況を入力とする．そこで，各ライブラリの使用情報量の違いが推薦システムに与える影響を調査する．

2.2　関連研究

Nguyen らは，プロジェクトの使用するライブラリを入力とし，協調フィルタリングを用いてライブラリを推薦する手法 CrossRec を提案し [1]，Ouni らの LibFinder [2] に加え，LibRec [3]，および LibCUP [5] と性能比較を行った．その結果，全評価指標において従来の手法と同等か優れた結果が得られたことを示した．そのため本稿では，Nguyen らの推薦手法を用いてライブラリ推薦を行う．

3　実験設計

3.1　データセット構築

取得プロジェクトの選定. Python における機械学習プロジェクトを対象とした実験を行うために，よく利用されている機械学習ライブラリを調査する．使用割合が高い機械学習ライブラリを示した Kaggle の調査結果 [6] から，使用割合が 5%よりも高い 12 件の Python のライブラリを利用しているプロジェクトを推薦対象として選定した．本稿では，12 件の各ライブラリにおいて，それぞれのライブラリを使用している GitHub 上のプロジェクトのうち，スター数上位 100 件までを取得した．

使用ライブラリの情報取得. 構築した推薦システムは，プロジェクトが使用しているライブラリの情報を入力として，使用される可能性の高いライブラリを推薦する

手法である．そのため，各プロジェクトが使用しているライブラリの情報を取得する必要がある．Python において ‘requirements.txt’ ファイルには，開発段階で使用されたライブラリとそのバージョンの情報が記されている．本稿では，取得したプロジェクトに含まれる ‘requirements.txt ’ ファイルのライブラリ使用情報を参照した．その結果，プロジェクト数 417 件を取得した．そして，1 つ以上のプロジェクトが使用していた上記の 12 件を含む 2,509 件のライブラリを取得し，これをデータセットとして使用する．なお本稿ではライブラリ名が同じ場合，異なるバージョンでも同一のライブラリとみなす．

評価行列の作成およびデータセットの分割．プロジェクトとライブラリの関係を表した評価行列を作成する．各行には取得した全プロジェクトを，各列には取得した全ライブラリを割り当てる．また各セルには，あるプロジェクトが指定したライブラリを使用していた場合に 1 を，それ以外の場合に 0 を格納する．

本稿では取得したプロジェクトおよびライブラリを用いて，417×2,509 の評価行列を作成し，データセットとする．そして，評価行列の行のうち α 件をテストデータ，β 件を訓練データとして無作為に分割する．

3.2 データ加工

セル値の重みづけを行う．これは，使用頻度の高いライブラリのみから類似プロジェクトが選ばれることを防ぐためである．重みづけの手法は先行研究 [1] と同じものを使用した．

テストデータの評価行列の列のうち，γn 件を無作為に選びクエリデータに，残りの $(1-\gamma)n$ 件を ground-truth データとする．ここで n は評価行列の列数，γ は評価行列の 1 行あたりに存在する値の割合を示し，γ の値が小さいほどプロジェクトが各ライブラリを使用しているか否か示す情報の量が少ないことを示す．そして ground-truth データに含まれるライブラリの使用情報を欠損させ値を Nan とし，欠損データとする．ground-truth データは推薦対象プロジェクトに推薦したライブラリが使用されているかの比較，クエリデータはテスト，訓練データのプロジェクト間の類似度の算出，欠損データは推薦手法を使用して算出した推薦スコア（3.3 節参照）の格納に用いる．

3.3 類似プロジェクト選定と推薦ライブラリ選出

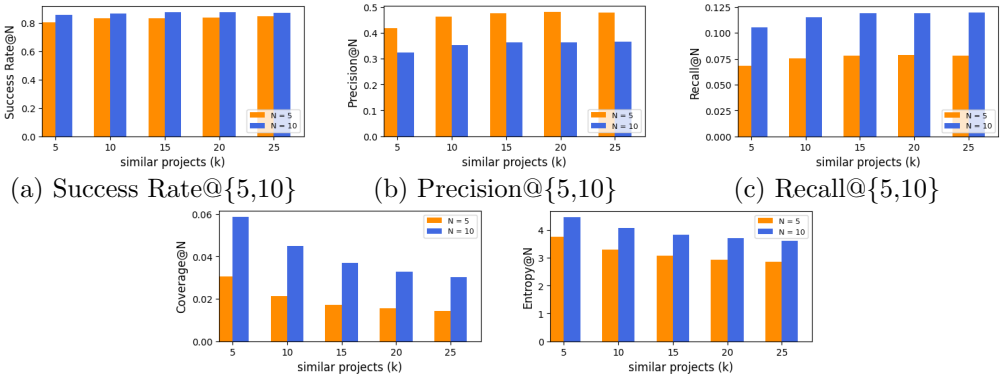
3.2 節で作成した各テストデータに対して，訓練データとのプロジェクト間コサイン類似度をクエリデータに基づき算出し，各プロジェクトに対して類似度上位 k 件のプロジェクトを抽出する．また，欠損データの欠損値 Nan に対して，類似プロジェクトによって作成された評価行列から各ライブラリが使用される可能性を表す推薦スコアを予測する．算出方法については先行研究 [1] に従った．

算出した推薦スコアのうち上位 N 件のライブラリを各プロジェクト p に対して推薦する．そして ground-truth と比較し，評価指標値を算出することで推薦システムの評価を行う．ただし開発段階でグラフの可視化や高度な算術演算など，機械学習モデルの構築に直接関係しないライブラリも使用されることを考慮し，それらのライブラリを機械学習関連ライブラリとして推薦対象とした．

4 調査内容と結果

本章では，2 つの各 RQ のアプローチと結果を示す．なお，推薦システムの精度評価のために，Nguyen ら [1] が用いた評価指標である，$SuccessRate@N$, $Precision@N$, $Recall@N$, $Coverage@N$, $Entropy@N$ を使用した．これらの指標を用いたのは，推薦対象が Java である場合と比較するためである．算出方法は先行研究 [1] に従った．

4.1 RQ1：Python において機械学習関連ライブラリをどの程度の精度で推薦で



(a) Success Rate@{5,10} (b) Precision@{5,10} (c) Recall@{5,10}

(d) Coverage@{5,10} (e) Entropy@{5,10}

図 2　$k \in \{5, 10, 15, 20, 25\}$ で得られた各評価指標値

きるのか

4.1.1　アプローチ

取得したプロジェクト 417 件を図 1 の $\alpha = 41, \beta = 376$ として無作為に分割する．次に $\gamma = 0.5$ とし，クエリデータと訓練データの評価行列の各セルの値から各テストデータのプロジェクトに対して類似プロジェクト k 件を選出し，それぞれに対して N 件のライブラリを推薦する．まず $N \in \{5, 10\}$ に固定し，$k \in \{5, 10, 15, 20, 25\}$ と変化させ，評価指標値の推移を調査する．また $k \in \{10, 20\}$ に固定し，$N \in \{1, 3, 5, 7, 10\}$ と変化させ，同じく評価指標値の推移を調査する．α, β, γ, k, N の値は先行研究と同様に設定した．データによる偏りを防ぐため、各 k, N に対して推薦対象プロジェクトを変更して 10 回実験を行い，その平均値を評価指標値として用いる．

4.1.2　結果

$N \in \{5, 10\}$ と $k \in \{10, 20\}$ にそれぞれ固定した場合の各評価指標の推移は類似していたため，$N \in \{5, 10\}$ に固定したときの結果のみを示す．

全評価指標において，類似プロジェクト数および推薦ライブラリ数の変化による値の推移は，Nguyen ら [1] の先行研究と類似した傾向を示した．類似プロジェクト数の変化の影響が小さかった原因として，選出されたプロジェクトが同じライブラリを使用していたことが考えられる．

4.2　RQ2：各ライブラリを使用しているか否かの情報量の違いにより，推薦精度に変化が生じるのか

4.2.1　アプローチ

RQ1 同様に図 1 の $\alpha = 41, \beta = 376$ として無作為に分割する．次に図 1 の $\gamma \in \{0.2, 0.4, 0.6, 0.8\}$ として無作為に分割する．続いてテストデータの各プロジェクトに対して，訓練データの評価行列 $\beta \times \gamma n$ から $k = 20$ 件の類似プロジェクト行を取得する．取得した k 件の類似プロジェクトから，欠損データの $(1 - \gamma)n$ 件の欠損値 Nan に対して推薦スコアを算出し，推薦スコア上位 $N \in \{1, 3, 5, 7, 10\}$ 件のライブラリを各テストデータのプロジェクトに対して推薦する．$k = 20$ に設定した理由は，RQ1 の結果において $k = 10$ の場合よりも $SuccessRate@N$ の平均値が高かったためである．データによる偏りを防ぐため，各 γ に対して推薦対象プロジェクトを変更して 10 回実験を行い，その平均値を評価指標値として用いる．また，各過程において推薦されたライブラリの推薦回数をカウントし，総推薦回数上位 10 件のライブラリ集合の要素の変化を調査する．

4.2.2　結果

図 3 より γ の値を増加させる，すなわちプロジェクトに対して各ライブラリを使用しているか否かを明らかにするほど $Recall@N$ は向上したが，$Precision@N$ は

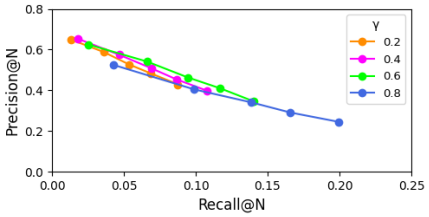


図 3　RQ2 で得た PRC 曲線

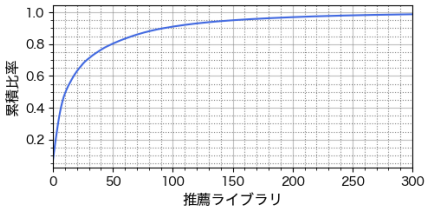


図 4　推薦回数の累積度数分布

表 1　推薦回数上位 20 件のライブラリ

scikit-learn	torch	scipy	matplotlib	numpy
pandas	pytest	pillow	sphinx	requests
sphinx-rtd-theme	tensorflow	tqdm	pytorch-lightning	flake8
pytest-cov	pyyaml	torchvision	transformers	joblib

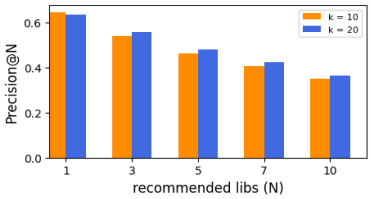
γ の設定を 0.2 から 0.6 ではほとんど変化がなく，0.8 では低下した．入力情報量の増加により $Precision@N$ が低下したのは，より高精度な類似プロジェクト探索が可能になる一方で，プロジェクトが使用していたライブラリのほとんどが入力に含まれ，不用な推薦を行った可能性が考えられる．また $Recall@N$ が向上したのは，ground-truth に含まれるライブラリのうち，使用割合の低いライブラリを推薦可能になったことでプロジェクト特有のライブラリを推薦可能になったためと考えられる．

5　考察

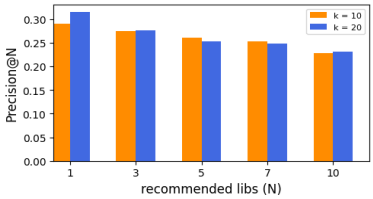
図 4 は，RQ2 で行った調査における総推薦回数のうち，推薦された各ライブラリの推薦回数を累積度数分布として表したものである．図 4 より上位 20 件のライブラリが推薦された全ライブラリの 63%を占めていたことがみてとれる．このことから，開発者は同じライブラリを使用して機械学習システムを構築していると考えられる．実際の推薦回数上位 20 件のライブラリを表 1 に示す．

表 1 より，推薦された上位 20 件の中で機械学習ライブラリは scikit-learn, torch, tensorflow, torchvision, transformers，pytorch-lightning の 6 つであり，Kaggle の調査で上位だった xgboost, keras, pytorch はあまり推薦されなかった．そのため，分析対象のデータセットでこれらのライブラリを使用したプロジェクトが少なかったと考えられる．また scipy，numpy，matplotlib，pandas などのデータ処理に使用されるライブラリも多く推薦された．分析対象のデータセットでこれらのライブラリが使用されていたことから，機械学習ライブラリをより効果的に活用するための，入力データ構造の定義などを支援するデータ処理ライブラリは，頻繁に同時に使用されると考えられる．さらに，pytest や pytest-cov，sphinx，sphinx-rtd-theme, flake8 が推薦されていた．これらのライブラリは Python プログラムのテストなど品質保証に関するものである．そのため，コード上の問題をできる限り取り除く努力が機械学習ライブラリを活用したプロジェクトでも意識されていると考えられる．

また，表 1 の 20 件のライブラリのうち 19 件が実際のプロジェクトにおける使用回数上位 20 件に含まれており，使用回数が多くなるほど推薦頻度が高いことが確認できた．そのため，使用頻度によって推薦結果が大きく左右される可能性が示唆された．そこで，これら 20 件のライブラリを入力情報としてその他のライブラリを各プロジェクトの推薦対象とした場合に，$SuccessRate@N$, $Precision@N$, $Recall@N$ の値に対してどのように影響を与えるのか追加で調査した．類似した結果を示したため，$Precision@N$ の結果を図 5 に示す．図 5(a), (b) よりほとんどの場合で評価指標値は半分近く減少した．このことからも多くのプロジェクトが上位 20 件のライブラリに依存し，その他のライブラリをほとんど使用していないと考えられる．



(a) RQ1 で得た入力がランダムの場合



(b) 入力を固定した場合

図 5　$k \in \{10, 20\}$ における Precision@{1,3,5,7,10}

以上より今回の推薦システムでは，推薦が使用頻度の高いライブラリに偏ってしまったため，使用頻度に関係なくプロジェクトに合わせたライブラリを推薦できるよう，データセットの拡張およびシステムの改善が必要である．

6　妥当性への脅威

内的妥当性．本研究では，各プロジェクトの ‘requirements.txt’ からライブラリ使用情報を取得して調査を行った．ただしファイル名は任意に設定可能なため，別のファイル名を使用している，あるいは，‘requiments.txt’ ファイルを設定していない場合がある．そのため，対象にすべきファイルを取得できていない可能性を考慮し，実際のプログラムからも情報を抽出して実験を行う必要がある．

外的妥当性．本研究では，Python の機械学習ライブラリの使用率上位 12 件のライブラリを最低 1 つ使用しているプロジェクトのうち，GitHub でのスター数各上位 100 件までを対象に調査を行ったが，調査結果をより一般化するためには，依存ライブラリ数，および対象のプロジェクト数を増加させて調査を行う必要がある．

7　おわりに

本稿では協調フィルタリングを用いて Python における機械学習関連ライブラリを推薦し，5 つの評価指標を用いてシステムの評価を行った．調査の結果，各評価指標値の推移は対象が Java の場合と同様の傾向を示したが，先行研究に比べて精度は同等またはそれ以下であった．今後の課題として，データセットの変更や拡張を行った調査，および推薦システムの構造変更が必要である．

謝辞　本研究の一部は JSPS 科研費 JP18H04097・JP21H04877，JP22K17874 および，JSPS・国際共同研究事業（JPJSJRP20191502）の助成を受けた．

参考文献

[1] P. T. Nguyen, J. D. Rocco, D. D. Ruscio, and M. D. Penta. CrossRec: Supporting software developers by recommending third-party libraries. *JSS*, Vol. 161, pp. 1–17, 2020.
[2] A. Ouni, R. G. Kula, M. Kessentini, T. Ishio, D. M. German, and K. Inoue. Search-based software library recommendation using multi-objective optimization. *IST*, Vol. 83, pp. 55–75, 2017.
[3] F. Thung, D. Lo, and J. Lawall. Automated library recommendation. In *Proc. of the WCRE'2013*, pp. 182–191, 2013.
[4] 小柳慶, 秋山楽登, 山手響介, 近藤将成, 亀井靖高, 鵜林尚靖. Python における機械学習関連ライブラリの自動推薦手法の初期評価. 研究報告ソフトウェア工学 (SE), Vol. 2022-SE-210, No. 17, pp. 1–8, 2022. .
[5] M. A. Saied, A. Ouni, H. Sahraoui, R. G. Kula, K. Inoue, and D. Lo. Improving reusability of software libraries through usage pattern mining. *JSS*, Vol. 145, pp. 164–179, 2018.
[6] Kaggle. State of Data Science and Machine Learning 2021. https://www.kaggle.com/kaggle-survey-2021(Last accessed 22 Sep 2022).

ソフトウェア部品の利用関係に基づくクラスタリングの進化分析

Analysis of Evolutions about Clusters based on Use-Relationships

横森 励士* 安藤 勇人† 吉田 則裕‡ 野呂 昌満§ 井上 克郎¶

あらまし 本研究では，利用部品の一致度に基づく階層的クラスター分析によって得た部品群がバージョンの前後でどう変化したかを示す．部品間の利用関係が保守工程でどう維持され，変化していくかを評価する．

Summary. We investigate how the set of components obtained by hierarchical cluster analysis based on the degree of coincidence of using components changes with the progress of versioning. These analyses represent how software grows from the viewpoint of the use relationship.

1 はじめに

ソフトウェアの活用期間が長くなるにつれて，様々な機能追加や新環境への対応などがソフトウェアに適用される．ソフトウェアは徐々に規模が増大し，構成する要素数も増加し，構成要素間の関係は複雑になる．ソフトウェア内の構成要素の理解しやすさを維持するために，リファクタリングなどにより内部構造が整頓されることが求められる．ソフトウェアの内部構造がどのように進化するかという傾向を把握することで，理解支援のための維持活動を支援できる知見が得られると考える．

一方で，我々の研究グループは，分析対象ソフトウェア内の部品を利用関係の一致度に基づいて階層的クラスター分析し，類似部品の集合を得る手法を提案した [1] [2]．オープンソースのプロジェクトに適用したところ，樹形図上でまとまりになっている部品群の多くは，主となる機能に関連した点で機能的な共通点を有していた．

本研究では，ソフトウェアの内部構造がどのように進化するかを調査する一方法として，開発バージョンの進行によって前述の『類似部品の集合』が樹形図上でどうあらわされるかを調査する．実験においては，2 通りの類似度計算方法でクラスタリングを行い，それぞれの計算手法ごとに，初期バージョンで確認した部品群が最新バージョンの樹形図上でどう現れるか，最新バージョンで確認した部品群が初期バージョンで樹形図上でどう存在していたか，を調査する．

バージョン間で『類似部品の集合』が樹形図上でどう変化したかを示したことが本研究の貢献であると考え，今後，ソフトウェアの成長過程を分析する際に，利用関係面からの分析として，本分類手法が活用できることを示すことを目的とする．

2 背景技術

2.1 ソフトウェア部品と利用関係，利用関係に基づく部品分類手法

一般にソフトウェア部品とは，その内容をカプセル化したうえで，それらを交換可能な形で実現したシステムモジュールの一部であると考えることができる．本研究では，開発者が再利用を行う単位として，.java のソースファイルをソフトウェア部品とみなして分析を行っている．ソフトウェアは複数のソフトウェア部品で構成されると考えることができ，継承，変数の宣言，インスタンスの作成，メソッドの

*Reishi Yokomori, 南山大学 理工学部

†Hayato Andou, 南山大学 理工学研究科

‡Norihiro Yoshida, 立命館大学 情報理工学部

§Masami Noro, 南山大学 理工学部

¶Katsuro Inoue, 南山大学 理工学部

呼び出し，フィールド参照など「ある部品が他の部品を利用する」,「他の部品からその部品が呼び出される」といった関係を利用関係として定義できる．

ある部品の上で機能を実現する場合，その部品にすべての機能を実現するわけではなく，実際には他の部品で提供されている機能やライブラリを利用しながら目的となる機能を実現している．我々は，利用先に関して，2 つの部品が利用している部品が一致している割合が高いほど，それらのは機能的な観点で目的や役割が似ている部品となるのではないかと考えた．さらに，文献 [1] では，ソフトウェアの利用先の一致度から各部品の類似度を計算し，部品間の距離行列を生成し，階層的クラスター分析を行い樹形図を得ることで，あるソフトウェア内の部品を分類する手法を提案している．文献 [1] では図 1 の実線矢印で示す,「ソフトウェア内で定義されたクラス」である利用部品の集合から類似度を計算する．実験では，樹形図上で部品群として得られた部品群の大半は，機能的な観点で関連性を強く認識できる部品の集合であったことが確認できた．一方で利用クラスが無いという理由で分類されない部品も多くみられた．以降,『uses-internal』による分類と呼ぶ．

遠藤らは文献 [2] で，ライブラリなどのソフトウェア外で定義された部品への利用関係を抽出し，その利用関係の一致度に基づいて階層的クラスター分析を行う手法を提案し，文献 [1] の手法と比較を行った．文献 [2] では図 1 の破線矢印で示す,「ソフトウェア外で定義された部品」である利用クラスの集合から類似度を計算する．この類似度を用いると文献 [1] と比較して，利用クラスのない部品は減るので，分類された部品の総数は増加した．一方で，java.lang.Object などのクラスの利用で結びつく部分が一定数存在し，その部分は機能的な関連を見出せなかった．以降,『uses-external』による分類と呼ぶ．

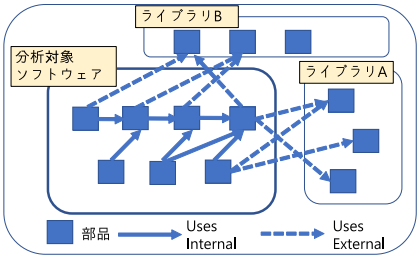


図 1: uses-internal と uses-external

2.2 利用関係に基づくソフトウェア部品の分類の手順

文献 [1] や 文献 [2] の手法における分析の手順を以下に示す．

1. Classycle Analyzer で分析対象の各部品（ファイル）の利用関係を抽出する．
2. 各部品対の 2 つの利用部品の集合の類似度を jaccard 係数を用いて求める．
3. 各部品対の距離を類似度から求め，部品間の距離行列を作成する．
4. R を用いて，距離行列から階層的クラスター分析を行い，樹形図を得る．本研究では，文献 [1] と同様に群平均法を用いる．
5. 樹形図の葉から根のほうにたどりながら，以下の三基準のどれかを満たすことで機能的類似性を持つと判定できる部品の範囲を調査し，最大の部分を類似部品群とする．

基準 1: 部品の扱う対象が同じである．
基準 2: 部品の役割が同じである．
基準 3: 部品の役割や対象やパッケージなどから 1 つの機能群としてまとめられる．

3　調査内容

3.1　調査の動機と目的

本調査では，複数のバージョンを有する Java ソフトウェアシステムとして，住宅を設計し，家具等を配置しながら，3D ビューで間取りを確認できるアプリケーションである SweetHome3D を分析対象とした．SweetHome3D の 2007 年 9 月リリースの Ver1.0（82 クラス）と，2008 年 10 月リリースの Ver1.4（110 クラス）を比較し，最初のバージョンで得られた部品群が後のバージョンでどう現れているか，後のバージョンで得られた部品群が前のバージョンでどうたったかを調査する．以下，

RQ1. 利用関係に基づくソフトウェア部品の分類手法をバージョン間分析に用いることで，どのようなことがわかるか?

RQ2. 類似度の計算手法ごとに得られる結果の特徴があるか?

というリサーチクエスチョンを設定し，『uses-internal』,『uses-external』による分類それぞれに対して調査を行い，手法間の違いを確認し知見を得る．

3.2　初期の部品群は後のバージョンでどう現れたか?

図 2 と 図 3 の上の樹形図では，Ver1.0 で『uses-internal』による分類 (図 2) 時には 6 つの部品群が『uses-external』による分類 (図 3) 時には 4 つの部品群が得られたことがわかる．『uses-external』による分類の場合，樹形図右側は汎用的な部品の利用で結びついており，類似部品群と判定できる部分が存在しなかった．図 2 と図 3 の下の樹形図は，その部品群内の部品が Ver1.4 でどう存在しているかを示す．

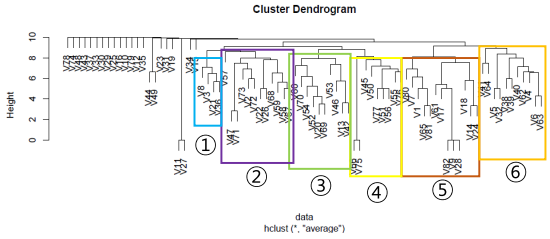
- 図 2 の『uses-internal』による分類での部品群①，②，③，④は，全体を統括する部品（群①），印刷関連の部品（群②），家具などを管理する部品（群③，④）であった．Ver1.4 でも原型が保たれており，関連する新規追加部品が多くなかったと考えられる．
- 図 2 の『uses-internal』による分類での部品群⑤，⑥は，コンテンツマネージャを利用する部品（群⑤），カタログを利用する部品（群⑥）などの比較的細かい内容で一致した部品群であった．近い機能をもつ新規部品の追加により，新規部品を含みながらも，部品群は維持されている．
- 図 3 の『uses-external』による分類での部品群①，②，③，④は，オブザーバを実現する部品（群①），家具を扱う部品（群②），コントローラー（群③），イベントやリスナーを実現する部品（群④）のように，具体的に実現する機能に関連した分類結果が得られた．Ver1.4 でも原型が確認でき，大きな変化は見られないが，群④は，イベント関係と，リスナー関連の部品で細分化された．

3.3　最新バージョンの部品群は以前はどうだったか?

図 4 と 図 5 の上の樹形図では，Ver1.4 では 『uses-internal』による分類 (図 4) 時には 5 つの部品群が 『uses-external』による分類 (図 5) では，6 つの部品群が得られたことがわかる．

- 図 4 の『uses-internal』による分類での部品群①，②，④，⑤は，Ver1.0 でベースとなる部品群が見られた．同様に，図 5 の『uses-external』による分類での部品群①，④，⑤，⑥は，Ver1.0 でベースとなる部品群が見られた．
- 図 4 の『uses-internal』による分類での部品群③は，Ver1.0 では存在せず，バージョン進行中に新たに部品群化した部品群だった．途中で追加された OperatingSystem クラスの共通利用で結びついた．OperatingSystem は OS に関連する情報を整理し，Pane などでの画面出力に活用していた．
- 図 5 の『uses-external』による分類での部品群 ②, ③は，最初のバージョンでは存在せず，バージョン進行の途中に新たに部品群化した部品群であった．部品群②は panel を利用しているクラスが，部品群③は URL を利用しているクラスが，それぞれ新規に追加された関連部品と部品群を形成した．

Ver 1.0



Ver 1.4

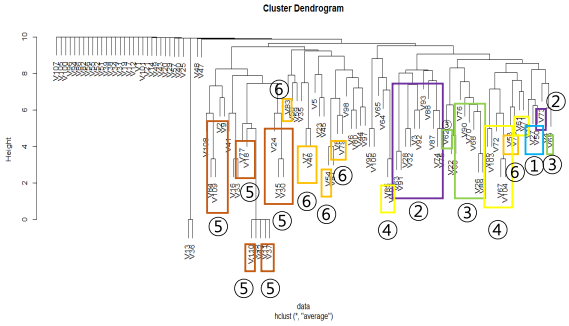
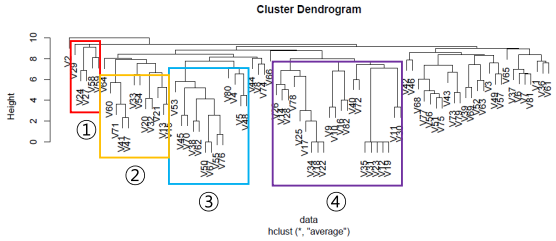


図 2: 初期の部品群は後のバージョンでどう現れたか （『uses-internal』）

Ver 1.0



Ver 1.4

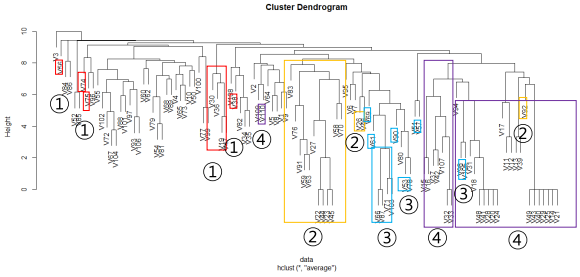


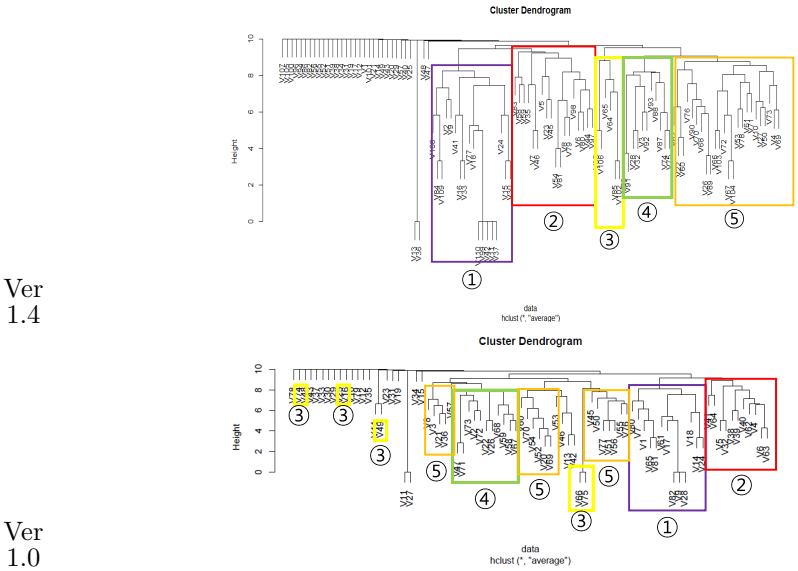
図 3: 初期の部品群は後のバージョンでどう現れたか （『uses-external』）

4 考察

4.1 RQ1: 利用関係に基づくソフトウェア部品の分類手法をバージョン間分析に用いることで，どのようなことがわかると考えられるか?

基本的に，樹形図はクラス数の増加により巨大化しているが，最初から機能的な類似性を示していた部品どうしについては，その多くが関係を維持していた．これは，新たなクラスを定義することで機能追加が実現されることが多い Java のソフトウェアの性質を示していると考えられる．

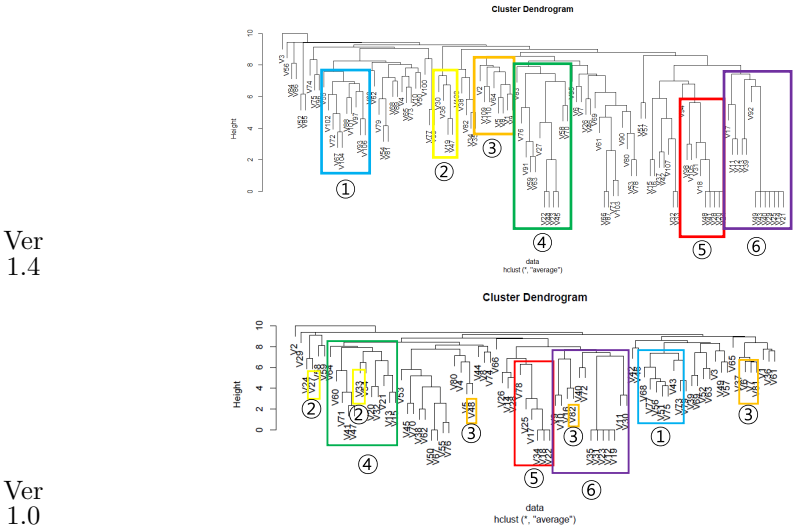
一方で，新規に追加される部品どうしや，新規に追加された部品と関連のある部品



Ver 1.4

Ver 1.0

図 4: 最新バージョンの部品群は以前はどうだったか? （『uses-internal』）



Ver 1.4

Ver 1.0

図 5: 最新バージョンの部品群は以前はどうだったか? （『uses-external』）

どうしが部品群化するという事例が『uses-internal』による分類手法,『uses-external』による分類手法それぞれで確認できた．樹形図上で部品追加が多い部分は，機能追加の対象になっていると仮説が立てられるかもしれない．

4.2 RQ2: 類似度の計算手法ごとに得られる結果の特徴があるか?

『uses-external』による分類手法は，ソフトウェアの初期の段階から類似性を示す部分はほぼ一定で，バージョン間でもその分類結果はほぼ同じだった．ライブラリ部品の利用は，そのクラスが担う機能に基づいておおむね決まり，一度できてしまうと大きく利用部品が変わることがないと考えられる．

『uses-internal』における分類の特徴として，バージョン進行中に既存の部品だけで新たに部品群化した部品群が存在していた．今まで実現していた機能を再編成するために，組織的にサポートすることを目的としたり，既存の部品を整理してより使いやすくすることを目的としたクラスが追加されうる．そのような内部の状況を整備するための部品をどこから使わせるかという開発者の判断が部品群の形成にも影響すると考えられる.『uses-internal』による分類手法は，このような機能整備のための部品の追加で組み変わりが起こる可能性があり，ある程度機能追加が行われて, 利用関係が成熟し変化が落ち着いた段階で, より効果的な分類結果が得られると考える．今後,『uses-internal』による分類手法において，バージョンの進行によって既存の部品が新たに部品群化した部品群に着目し，そのような部品群はどのような部品から構成され，どのような利用関係の追加によって部品群化したか，どのような機能的な類似性をもつかを調査し，知見として役立てたい.

4.3 関連研究

ソフトウェアリポジトリに蓄えられたソフトウェアの再利用を目的として，インターフェイスの観点 [3] や，機能外のプロパティからの観点 [4] ，コンポーネントモデルの観点 [5] などソフトウェアコンポーネントを様々な側面から分類し整理する研究が行われている．我々が提案する手法は，ソフトウェア内の部品の理解を目的として，単一システム内の部品を利用部品の共通性からクラスタリングする．筆者らは，過去にフレームワークとアプリケーション間の利用関係の変化を，バージョンごとに，部品の利用数，被利用数などのメトリクスの変化として調査した [6] ．今回の調査は，利用関係の変化をメトリクス以外の視点から分析したもので，調査結果を総合し，利用関係の変化の点からの分析手法を確立したいと考える.

5 まとめ

本研究では,『利用部品の一致度に基づく分類手法で得た部品群』がバージョンの進行によってどう変化するかを調査した．今後，他のシステムでの分析における既存の部品同士が新たに部品群化した事例に着目し，考察における仮説を検証したい.

謝辞 本研究は，2022 年度南山大学パッヘ研究奨励金（I-A-2）の助成を受けている．本論文は文献 [7] に基づいており，安藤は分析の面で本論文でも関わっている．筆頭著者は卒業論文の指導教員で，本論文にて評価や考察を再構成している.

参考文献

[1] R. Yokomori, N. Yoshida, M. Noro, and K. Inoue, “Use-relationship based classification for software components, ” Proceedings of 6th International Workshop on the Quantitative Approaches to Software Quality, pp.59–66, 2018.
[2] 遠藤智規，平松芳貴，川村駿弥 :“ ソフトウェア外で定義された部品の利用の共通性に基づくソフトウェア部品分類手法 ”，南山大学 情報理工学部 2018 年度卒業論文，2019.
[3] S. Yacoub, H. Ammar, and A. Mili, “A Model for Classifying Component Interfaces, ” in Proceedings of Second International Workshop on ComponentBased Software Engineering, in conjunction with ICSE99, pp. 17–18, 1999.
[4] I. Crnkovic, M. Larsson, and O. Preiss, “Concerning Predictability in Dependable Component-Based Systems: Classification of Quality Attributes, ” Architecting Dependable Systems III, pp. 257–278, 2005.
[5] K. Lau and Z. Wang, “Software Component Models, ” IEEE Transactions on, Software Engineering, vol. 33, No. 10, pp.709–724, 2007.
[6] R. Yokomori, H. Siy, N. Yoshida, M. Noro, and K. Inoue, “International Journal of Computer Science and Application”, International Journal of Computer Science and Application, Vol. 4, Issue.1, pp.18-31, 2015.
[7] 安藤勇人，白山遼祐，竹市豊 :“ 利用関係の一致度に基づくソフトウエア部品分類手法―ソフトウエア更新によって分類結果がどのように変化するかについての考察― ”，南山大学 2020 年度卒業論文，2021.

DVC リポジトリにおける 機械学習パイプラインの進化に関する調査

A Study on ML Pipeline Evolutions on DVC Repositories

中村 悠人[*] **松田 雄河**[†] **松尾 春紀**[‡] **近藤 将成**[§]
亀井 靖高[¶] **鵜林 尚靖**[‖]

あらまし 機械学習パイプラインの作成を支援するために Data Version Control（DVC）と呼ばれるオープンソースソフトウェアが近年利用されている．DVC は yaml ファイルにより機械学習パイプラインを管理することができ，機械学習パイプラインの作成，管理を効率化できる．DVC を導入することは機械学習プロジェクトにおいてメリットがあるが，まだ新しい技術であるため，最適な導入方法や導入時期などベストプラクティスは十分に共有されていない．そのため，本研究では，DVC 上での機械学習パイプラインの変更過程を明らかにし，DVC を使用しているプロジェクトがどのように DVC を導入し活用しているのかについて調査した．DVC のパイプライン機能を使用し，継続的な開発が行われていると考えられる機械学習関連プロジェクト 25 件の機械学習パイプライン 42 件を調査対象とした．調査の結果，現在使用されている機械学習パイプラインは変更が複数回行われたものが 3/4 以上を占めた．また，学習や評価に属するプロセスの修正が多く，最初に登録した機械学習パイプラインのプロセスが現在まで無変更であるものが，変更が行われたプロセスの 1/10 以下であった．さらに，現在使用されている機械学習パイプラインはデータ取得，学習，評価のプロセスを含むものが多いことがわかった．

1 はじめに

近年，機械学習システムはより一層関心を集めており，さまざまな場面で応用されている．その中核となる機械学習モデルの作成にはデータ取得，モデル学習などさまざまなプロセスが存在する [1]．これらをつなげた一連の流れである機械学習ワークフローは，プロセスの分岐や多重に連なることにより複雑で管理が難しくなる．ワークフローが複雑化することにより，モデルの精度向上のための試行錯誤，チーム内での共有，運用のコストが増加する．また，プロセス同士には依存関係にあるものと独立したものがあるため，独立したプロセスを更新した場合にはそのプロセスのみを再実行すれば良い．しかし，プロセスの依存関係を考慮しなければ，そうした再実行対象となるプロセスの選定ができず，すべて再実行してしまい非効率である．以上より，ワークフローの管理は機械学習モデルの作成で重要な位置を占めるといえる．

この課題を支援する手法として，機械学習パイプラインがある [2]．機械学習パイプラインは，機械学習におけるプロセスの単位をステージとして，それらをつなげた機械学習ワークフローを自動で実行できる．この手法を使用することで，複雑なワークフローの管理と，その再現性を確保できる．

機械学習パイプラインの機能を持つアプリケーションとして Data Version Control（DVC）がある [3]．DVC は，機械学習パイプラインを作成できるほか，従来のバー

[*]Yuto Nakamura, 九州大学
[†]Yuga Matsuda, 九州大学
[‡]Haruki Matsuo, 九州大学
[§]Masanari Kondo, 九州大学
[¶]Yasutaka Kamei, 九州大学
[‖]Naoyasu Ubayashi, 九州大学

ジョン管理である Git やそのホスティングシステムである GitHub では難しかった，再現性を確保する上で重要な大量のデータや，モデルなどのファイルのバージョン管理を行うことができる．これらの理由により，今後の機械学習を使用するソフトウェアにおいてもさらなる導入が期待される．

しかし，DVC における機械学習パイプラインの作成は機械学習システムの品質向上において重要であるが，実際にどのように機械学習パイプラインが変更され，作成されているのか知られていない．機械学習パイプラインの変更過程，最終構成をあらかじめ知っておくことにより，開発するシステムの目的にあった機械学習パイプラインの構成がわかり，開発者が試行錯誤するコストを減らすことができる．また，試行錯誤が減ることで，パイプラインの管理がしやすくなり，バグを混入しづらくなることも期待される．

実際に使用されている機械学習パイプラインを調査することは，最適な機械学習パイプライン構成の選択や，そうでないものの除外，及びアンチパターンの知見の蓄積として有用であると考える．また，機械学習パイプラインには技術的負債と呼ばれるコードに存在するバグや解消すべき課題が存在している場合があるため [4]，この調査の知見は技術的負債が少ないパイプライン作成の手助けにもなると考える．このように，DVC の機械学習パイプラインの研究は，機械学習パイプライン作成において重要であると考えられる．

本稿では，DVC における機械学習パイプラインの変更過程の実態を把握することによって，現在開発者の間でよく使われているパイプライン構成，運用方法，パイプラインの使用実態について考察する．目的を達成するため，以下 3 つの研究課題（RQ）を設定し，GitHub 上のプロジェクト 25 件に含まれる 42 件の機械学習パイプラインを対象に調査を行った．それぞれの調査課題の結果を以下にまとめる．

RQ1 機械学習パイプラインの変更はどの程度行われているか
パイプラインファイルの変更数は，1 回であるものが最も多かったが，3/4 以上の機械学習パイプラインは複数回変更されていた．コミットにおけるステージ変更数は 1 つであるものが最も多かったが，3/5 以上の機械学習パイプラインは一度に複数回変更されていた．

RQ2 機械学習パイプラインではどのようなステージが変更されているか
Data Collection のステージでは追加と修正が 1 対 1 で行われていた．一方，Model Training と Model Evaluation では，修正が追加よりもそれぞれ多く（3.7 倍，2.2 倍）行われていた．

RQ3 現在使用されている機械学習パイプラインにはどのような特徴が存在するか
現時点で使用されている機械学習パイプラインは，Data Collection，Model Training，Model Evaluation のステージが多く含まれていた．

以降，第 2 章では関連研究について説明する．第 3 章では本調査における実験設計について説明する．第 4 章では調査課題の内容とその結果について説明する．第 5 章では結果についての考察を行う．第 6 章では妥当性への脅威について述べ，第 7 章では結論と今後の課題について述べる．

2 関連研究

2.1 データバージョン管理

データバージョン管理とは，モデルやデータの変更履歴を管理することである．特に大量のデータを管理したい場合に有用である．従来であれば，ソースコードなど軽量なソフトウェアアーティファクトのみであれば Git を使用することで変更履歴を管理し，GitHub によりこれらのファイルを共有してきた．しかし，GitHub には，機械学習に使用する学習用データのようなサイズの大きいファイルを置くことが難しい．データバージョン管理ツールの登場により，機械学習で使用するような大量のデータの管理ができるようになった．データバージョン管理ツールによって，

学習に使用したファイルをまとめて管理できる．

データバージョン管理ツールで重要視されるのは，再現性である．再現性とは，同じ学習データを入力すれば，同じモデルができることである．再現性の確保のためには，使用されているコードやデータ，モデルを一緒に管理する必要がある．再現性により，チーム間でのデータ，モデルの共有，バージョン変更が容易に行える．

そのため，データバージョン管理ツールには，機械学習プロセスの再現性を確保できる機械学習パイプラインの機能が含まれていることが多い．

Baker の調査 [5] では，70%以上の研究者が他の研究者の再現実験に失敗しており，さらに 50%以上が自身の研究の再現に失敗したことがあるということが報告されている．機械学習研究の再現性の向上に向けた調査も行われており [6]，今後の課題であると考えられる．

データバージョン管理ツールによって，学習に使用したファイルをまとめて管理できるため，再現性の問題も解決できる．よって，今後の研究者や開発者の使用が見込まれる．データバージョン管理ツールには DVC のほか，MLflow [7] や Pachyderm [8] などがある．

Barrak ら [9] は，DVC ファイルとその他のファイルとの間で，変更が与える影響に関する実態調査を行っている．まず，GitHub 上にある DVC を使用しているプロジェクトのうちコミット数が多く，プルリクエストが 10 回を超えた 10 件のプロジェクトを対象にしている．その結果，プルリクエストではソースコードが変更されたうちの 1/4，及びテストコードが変更されたうちの 1/2 で，DVC ファイルが変更される要因となっている．次に，DVC を使用しているプロジェクトのうちコミット数が多い 25 件に，機械学習パイプラインの複雑度が初期コミットから現在までどのような変化をしているかを調査している．その結果，78%のプロジェクトで機械学習パイプラインの複雑度が一定ではないことを明らかにした．

Zaharia ら [10] は，使用されるツールの多さ，実験の追跡や再現性，及び製品のデプロイの難しさといった機械学習を使用するソフトウェア開発における課題を提示している．また，これらの課題解決のためにデータバージョン管理ツールである MLflow を開発した．Python の関数のように使用できるため，ライブラリの使用制限がなく柔軟な機械学習ソフトウェアの開発プロセスを作成できることを示している．

2.2　機械学習パイプライン

機械学習パイプラインは，機械学習のプロセスをあらかじめ登録しておいた順番に自動で実行できる．機械学習では，ハイパーパラメータは変更を繰り返す．これは，最適なモデルを見つけるために．学習を繰り返すからである．従来の方法では，プロセスを最初から 1 つずつ手動で実行するためミスを起こす可能性があった．変更を繰り返してもワークフローを変えずに自動で実行できる機械学習パイプラインは機械学習において重要である．また，機械学習パイプラインを共有することで同じステップで学習を行えるため，チームで研究や開発をする場合に有用である．機械学習パイプラインは機械学習自動化の観点や，ビルドやテスト，リリースを自動化する CI/CD の観点からも研究が進められている [11].

Data Version Control（DVC） [3] とは，2017 年 5 月 4 日に公開されたオープンソースソフトウェアである．DVC は，データバージョン管理，機械学習パイプラインの機能を備えている．さらに，従来のバージョン管理ツールである Git と共存することで機能するため，導入がしやすい．機械学習パイプラインの構成は “dvc.yaml” ファイルで管理される．例を Listing 1 [12] に示す．

ファイルにはステージの情報が含まれている．例では，“stages” 内に含まれている “prepare” が一つのステージであり，ステージ名は任意に設定できる．ステージ中の主な要素を説明する．“cmd” では Python ファイルの実行など，コマンドの実行ができる．“deps” には，ステージに依存するファイルが記述されており，ステー

Listing 1　パイプラインファイルである dvc.yaml の例

```
stages:
  prepare:
    cmd: python src/prepare.py data/data.xml
    deps:
    - data/data.xml
    - src/prepare.py
    params:
    - prepare.seed
    - prepare.split
    outs:
    - data/prepared
```

ジが一度実行されていた場合，依存しているファイルで変更が行われていなければ，そのステージは再び実行せずに済む．“params” 欄には，ハイパーパラメータを指定できる．“outs” には出力するファイルを記述する．

Olson ら [13] は，機械学習自動化ツールである TPOT を開発した．TPOT は，機械学習自動化に必要な機械学習パイプラインを作成する際に，前処理，特徴量抽出や選択，モデル構築のプロセスを枝分かれに組み合わせていく．実験では，さまざまなデータセットを TPOT，RandomForest，及び異なる最適化手法を用いた TPOT に適用し，学習の精度を比較している．その結果，TPOT は RandomForest と同程度の性能を示すことを明らかにしている．また，TPOT により作成されたいくつかの機械学習パイプラインは，RandomForest よりも精度が飛躍的に向上する前処理とモデル構築の組み合わせを発見したことを示している．さらに，多目的最適化を組み込むことによって TPOT は分類精度を低下させずにコンパクトで解釈が簡単であるパイプラインを作成できることを示している．

また，Drori ら [14] は，機械学習自動化ツールである AlphaD3M を提案している．AlphaD3M は，AlphaZero から発想を得た，機械学習パイプライン合成を一人用ゲームとして当てはめ，ニューラルネットワークとモンテカルロ木探索で意思決定を行い，挿入，削除，置換の動作を繰り返し機械学習パイプラインを組み立てていく手法である．この手法の利点は，組み立てる際になぜその行動をするように判断したのかわかるため，完成した機械学習パイプラインの構成が説明可能であるということである．この AlphaD3M を，既存の機械学習自動化ツールと比較する実験を行っている．その結果，既存のツールと同等の精度を得ることができ，学習時間を短縮できることを明らかにした．

3　実験設計

本章では，実際に使用されている機械学習パイプラインの構造の変化を調査するための手法と使用するプロジェクトについて説明する．

3.1　調査課題

本研究では GitHub 上における DVC の機械学習パイプラインの変更についての調査を行う．本研究は DVC における機械学習パイプラインの一般的な変更や，構成を明らかにする基礎となる研究であり，DVC をこれから導入する機械学習開発者にとって重要な知見となる．まず DVC パイプラインが過去から現在までどの程度変更されているかを調査する（RQ1）．さらに，パイプラインが変更されたコミットごとのステージの追加，削除，変更の有無を調査し（RQ2），現在使用されている機械学習パイプライン構成の知見を得る（RQ3）．

表 1　単語パターンによるステージの分類

分類	単語パターン	分類に含まれるステージの種類
Data Collection	load,unzip,get,etl,read,pull,make,fetch,create,prepare*data （load*model,create*model は除く）	37
Data Cleaning	clean	1
Data Labeling	label（ただし，Data Collection に属している場合，and が含まれていること ＜ download labeled data 等を除くため＞，及び train が含まれないこと）	12
Feature Engineering	feat	1
Model Training	train,log,split,prepare*model,load*model,create*model （ただし train が含まれており Data Collection に属している場合，and が含まれていること， 及び train が含まれており validate が含まれている場合，and が含まれていること）	51
Model Evaluation	eval,valid,analyze*model	35
Other		106

3.2　データセットの構築

3.2.1　対象プロジェクトの選定

本研究では，パイプラインファイルである dvc.yaml という名前のファイルの過去から現在まで，全てのブランチにおけるマージコミットではないコミットを調査の対象とする．GitHub 上のプロジェクトのうち，DVC の GitHub リポジトリに依存しているプロジェクトから，dvc.yaml ファイルが含まれるクローンできたプロジェクトを取得した（取得日：2022 年 4 月 7 日）．

3.2.2　セマンティックバージョニングによるフィルタリング

DVC のデモのみを行ったプロジェクトを除き，継続的に更新が行われているプロジェクトに絞るため，セマンティックバージョニングのルールに従って，マイナーバージョンまで記述されたタグが一つでもあるプロジェクトを調査対象とした．

また，本研究では機械学習パイプラインの変更過程の実態を把握することによって，現在開発者の間でよく使われているパイプライン構成，運用方法，パイプラインの使用実態について明らかにするため，

- 作成された時点から現在に至るまで機械学習パイプラインファイル内が空だったもの
- 作成された時点から現在に至るまでステージが登録されなかったもの

を除いた．GitHub 上のプロジェクトから条件を満たす，25 件のプロジェクトに含まれる 42 件の機械学習パイプラインを取得した．

3.2.3　ステージ名による分類

本研究では 3.2.2 節で取得した機械学習パイプラインを，一般的な機械学習開発で使用されている 9 個のステージ分類 [1] のうち，機械学習パイプラインに関係の深い以下の 6 個に分類する．

- Data Collection： 学習に必要なデータを収集したり，データを統合し新しいデータセットを作成する．
- Data Cleaning：ノイズを含むなど正確でないデータを取り除く．
- Data Labeling： 教師あり学習で使われる正解ラベルを割り当てる．また，強化学習などで使用される，デモデータや報酬を調整する．
- Feature Engineering：機械学習モデルに有益である特徴を抽出，選択する．
- Model Training：データセットと正解ラベルをもとにモデルを学習させる．
- Model Evaluation： 事前に定義された評価指標を使用し，モデルを評価する．

調査に使用される各ステージの分類は，パイプラインファイルの作成から現在までに現れるすべてのステージを目視で確認し，その中から関係が深いと思われる単語パターン（表 1）を抽出して，いずれかの単語パターンへの合致の有無により分類した．表 1 に示す分類のうち，2 つ以上のカテゴリに分類されているステージも存在する．全てのコミットからステージ名の異なる 243 種類のステージが確認された．

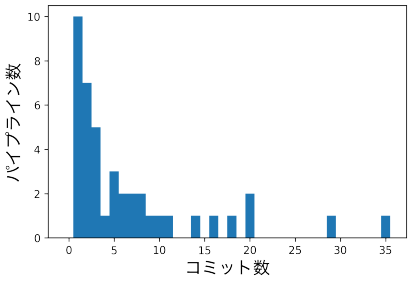


図 1　コミット数ごとのパイプライン ファイル変更件数

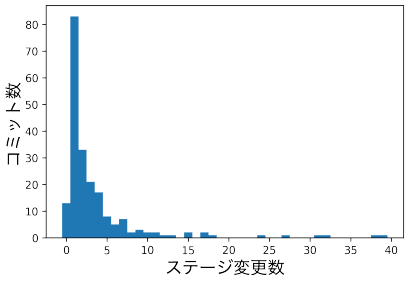


図 2　コミットにおけるステージ変更数の分布

4　調査内容と結果

本章では，3 章で取得した 42 件のパイプラインについて，3 つの RQ の調査結果を報告する．

4.1　RQ1: 機械学習パイプラインの変更はどの程度行われているか

目的：基本的な機械学習のワークフロー [1] はすでに提示されている．そのため，パイプライン構成がワークフローの指針に従って作成されているならば，開発者は完成した機械学習パイプラインを 1 回のみコミットすると予想される．しかし，著者らは，変更が複数回行われている機械学習パイプラインを確認している．つまり，実際に使用されている機械学習パイプラインはプロジェクトごとに異なり，それを開発・保守するために変更が複数回行われていると考えられる．そこで，どの程度の変更が行われているかの実態を把握するために，本 RQ では変更頻度の調査を行う．具体的には，機械学習パイプライン 1 つあたりの変更された回数及びコミット 1 つあたりで変更されたステージの件数を調査する．

アプローチ：3 章で取得した，パイプラインファイル 42 件を対象に，その変更ログを追跡し，機械学習パイプラインが変更された全ての時点でのパイプラインファイルを取得し，ファイル 1 つあたりの変更の回数を調査した．このとき，最初にコミットされたパイプラインファイルの中が空だった場合は考慮に入れず，ステージが追加された時点からの回数を調査した．また，yaml 形式に合っていないパイプラインファイルがコミットされた場合は除外した．直前にコミットされたパイプラインファイルと比較し，1 回のコミットでステージがいくつ変更されるか調べた．

結果：機械学習パイプラインの変更頻度の結果を図 1 に示す．機械学習パイプラインごとの現在までの変更回数は，平均 6.7 回，中央値 3.0 回であった．最も多かったものが 1 回のみの変更で，10 件（23.8%）だった．コミット数は，1,2 回と少ないパイプラインファイルが多いが，10 回以上コミットされたものも存在した．また，1 コミットにおけるステージの変更回数の結果を図 2 に示す．1 コミットごとのステージの変更回数は 1 回が最も多く，83 件（39.7%）だった．

パイプラインファイル 42 件における現在までの変更回数は，1 回が最も多かったが，3/4 以上のパイプラインファイルは 2 回以上変更されていた．また，1 コミットごとのステージの変更回数は 1 回が最も多かったが，3/5 以上のパイプラインファイルは一度に 2 つ以上ステージが変更されていた．

表 2　ステージの変更

分類	1 つ前のコミットとの比較				初期と最終のコミットとの比較			
	追加	削除	修正	変更なし	追加	削除	修正	変更なし
Data Collection	24	17	24	103	7	0	8	3
Data Cleaning	1	0	0	2	1	0	0	0
Data Labeling	10	12	20	69	1	3	0	0
Feature Engineering	1	0	0	0	1	0	0	0
Model Training	39	45	143	227	8	14	14	5
Model Evaluation	27	32	60	251	12	17	5	1
Other	69	60	61	81	28	18	7	5
合計	171	166	308	733	58	52	34	14

4.2　RQ2: 機械学習パイプラインではどのようなステージが変更されているか

目的：本 RQ では，DVC パイプラインの進化についての具体的な特徴を明らかにする．RQ1 では，パイプラインとそのステージの変更頻度のみの調査であった．本 RQ では，ステージに対する変更を追加，修正，削除の 3 つに分類し，それぞれの変更が 3.2.3 節で説明したステージ分類ごとにどの程度の割合で行われているかを分析する．変更が集中しているステージがわかることにより，機械学習パイプライン運用の実態を把握する．

アプローチ：42 件のパイプラインファイルについて，RQ1 で取得したファイルが変更された時点でのパイプラインファイルの内容をもとに調査を行った．まず，初期から現在まで，パイプラインファイルのコミットが行われた時点と 1 つ前のコミットが行われた時点での内容を比較した．また，パイプラインファイルが追加され初期状態になった時点と現在の最新のコミットでの内容を比較した．初期状態とは，ステージが一つ以上登録されており，yaml ファイルとして読み込み可能な状態を指す．

比較した情報より，ステージの追加や削除，修正の有無を調査した．ここでの修正とは，ステージの追加と削除を除く，ステージが直前のコミットと現在のコミットで存在し，内容のみが変化したものを指す．また，追加，及び削除は，比較するファイルによるステージの名前の存在の有無のみにより分析した．最後に，対象のステージを 3 章で行った単語パターンに従い分類した．

結果：結果を表 2 に示す．コミットが行われた時点のファイルと 1 つ前のコミットとの比較では，追加，削除，修正のうち，ステージの修正が最も多かった．ステージが修正されたもののうち，最も多かったステージ分類は Model Training の 143 個である．Other を除いた，追加，及び削除されたステージにおいても Model Training がそれぞれ 39 個，45 個と最も多く，次いで Model Evaluation がそれぞれ 27 個，32 個であった．また，変更されなかったステージの中では Model Evaluation が最も多く，251 個であった．さらに，Data Collection のステージでは修正が，追加と同数行われていたが，Model Training では 3.7 倍，Model Evaluation では 2.2 倍ステージの修正が追加よりも多く発生していた．

初期と最終のコミットとの比較では，追加，削除，修正を含めた変更されたステージ数 144 個に対し，初期から変更がなかったステージは 14 個と 1/10 以下であった．特に，削除されたステージは 52 個であり，追加されたステージ 58 個とほぼ同数存在する．

> Data Collection と比較して，Model Training や Model Evaluation ではステージの修正が多く発生していた．また，多くのパイプラインファイルは初期の機械学習パイプラインから変更が行われていた．

4.3　RQ3: 現在使用されている機械学習パイプラインにはどのような特徴が存在

表 3　機械学習パイプラインのステージ分布

分類	件数
Data Collection	39
Model Training	36
Model Evaluation	21
Data Cleaning	1
Data Labeling	1
Feature Engineering	1
Other	71
合計	139

表 4　ステージ構成

機械学習パイプライン構成	件数
Data Collection, Data Labeling, Model Training, Other	1
Data Collection, Feature Engineering, Model Training, Model Evaluation	1
Data Collection, Model Training	1
Data Collection, Model Training, Model Evaluation	4
Data Collection, Model Training, Model Evaluation, Other	2
Data Collection, Model Training, Other	3
Data Collection, Other	7
Data Cleaning, Model Training, Model Evaluation, Other	1
Model Training, Model Evaluation	1
Model Training, Model Evaluation, Other	5
Model Training, Other	8
Other	8
合計	42

するか

目的：実務者の間ではベストプラクティスとはいえずとも，DVC を使ったパイプラインの構築についての知見が溜まりつつあり，それがパイプラインに反映されている可能性がある．RQ1 及び 2 では時期については見ておらず，DVC がリリースされた当初のものも，現在のものも同時に分析しているため，現在どのようなパイプラインが DVC を使って作成されているかはわからない．そこで，この RQ では，機械学習パイプラインが最後に変更された時点でのパイプラインの構成を調査し，共通するステージの構成があるのかを特徴として明らかにする．

アプローチ：RQ1，RQ2 で使用したパイプラインファイル 42 件から，最終的にコミットされている機械学習パイプラインを取得した．これらの機械学習パイプラインを対象に，使用されているステージを表 1 の 6 つのステージの分類に分類し，それぞれの分類ごとの個数と，その組み合わせを調査した．

結果：構成されているステージの分類結果を表 3 に示す．機械学習パイプラインは，Data Collection，Model Training，Model Evaluation のステージを含むものが多い．また，各パターンで確認できた機械学習パイプラインのステージ構成の分布を表 4 に示す．Model Training が含まれるステージ構成が 27 件と最も多く，次いで Data Collection が含まれるステージ構成が 19 件であった．Other を含むパイプラインを除いた場合，全てのパイプラインに Model Training が存在していた．

パイプラインファイルのコミット数に関係なく最も多い機械学習パイプラインは，Data Collection と Model Training と Model Evaluation で構成されているものであった．

5　考察

本章では，DVC における機械学習パイプラインの構成，運用方法，その使用実態について，調査結果をもとに考察する．

RQ1 の結果より，3/4 以上のパイプラインが 2 度以上パイプラインが変更されていることがわかった．パイプラインの基本的な構成は Google が示しているもの [2] などがあるが，実務で使う際にはそれだけでは不十分な可能性が示唆されている．2 度以上変更されたコミットの中で，依存するファイルの追加や変更 [15]，ステージ中で実行するファイルの変更やコマンドライン引数の変更 [16] が確認できた．

RQ2 では，Model Training と Model Evaluation の修正回数が多いことがわかった．この結果より，パイプラインの中でもモデルの学習と評価に関しては，モデル作成時に発生する試行錯誤の影響を受け，頻繁に変更されている可能性が示唆されている．そのため，パイプライン構成時もモデルの学習と評価に関しては，その後の拡張を考える必要があると思われる．また，先行研究ではソースコードの変更とテストの変更のそれぞれ 1/4, 1/2 が DVC の変更に影響を与えるという結果が得られていた．そのため，多くのファイルと依存性があり，ソースコードの変更が多いと考えられるこれらのステージの変更が多いと考えられる．Data Labeling などのステージは依存するソースコードは少ないため，ステージへの影響が小さく，変更が少なかったと考えられる．

RQ3 では，Data Collection，Model Training，Model Evaluation の 3 つが最も頻繁に使われたステージの分類先であることがわかった．RQ2 の結果からわかるように Model Training と Model Evaluation は変更回数も多く，パイプラインの中でも重要なステージであることが強調された．Data Collection は RQ2 では変更回数が際立って多いということはなかった．一方で，入力データは機械学習モデルにとって重要な要素であり，パイプラインでもその構成要素として重要であることが示唆された結果である．また，表 4 で件数が多かった組み合わせである Data Collection, Other と Model Training, Other には評価のステージが含まれていない．目視で確認すると，これらは機械学習パイプラインが開発段階 [17] や，Model Training のステージで評価を行っている [18] と考えられるものであった．

6　妥当性への脅威

内的妥当性．本研究では，パイプラインファイルを用いて，そのパイプラインファイルが構成するパイプラインのステージを分析した．この時，ステージ名をルールベースな手法で分類したが，このルールは著者が手作業で作成したものであり，主観的である可能性がある．そのため，パイプラインファイル作成者の意図と異なる分類をしている可能性がある．今後，より精度の高い分類が行えるような方法を提案していく必要がある．

外的妥当性．GitHub 上から DVC の機械学習パイプラインを使用しているプロジェクト 25 件を調査対象とした．一般性を向上させるためには，より多くの機械学習パイプラインを含むプロジェクトを対象に，同様の調査を行いたい．

7　おわりに

本研究では，まず RQ1 で，機械学習パイプラインのファイル変更とステージ変更の頻度を調査した．RQ2 では，具体的なファイル変更における内容の調査を行い，そのステージごとの変更頻度について分析した．RQ3 では，現在の機械学習パイプラインがどのような構成であるか，使用されているステージをもとに調査を行った．

RQ1 では，パイプラインファイルの変更数は，1 回であるものが最も多かったが，3/4 以上の機械学習パイプラインは複数回変更されていた．また，コミットにおけるステージ変更数は 1 つであるものが最も多かったが，3/5 以上の機械学習パイプライ

ンは一度に複数回変更されていた．RQ2 では，Model Training，Model Evaluation での変更が多く見られた．RQ3 では，現時点でのプロジェクトは Data Collection，Model Training，Model Evaluation が多く存在し，多くのパイプラインで Model Training のステージが存在することが確認された．

今後の課題として，研究の一般性向上のため，より多くのプロジェクトに対し研究を行う必要がある．また，機械学習パイプライン内のステージをより厳密な機械学習ステップに分類することが考えられる．さらに，今回はパイプラインファイルのみに注目して分析を行ったため，今後はステージが依存するファイルの変更にも注目し分析する．

謝辞 本研究の一部は JSPS 科研費 JP18H04097・JP21H04877，JP22K17874 および，JSPS・国際共同研究事業（JPJSJRP20191502）の助成を受けた．

参考文献

[1] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In *Proceeding of the IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 291–300, 2019.
[2] K. Salama, J. Kazmierczak, and D. Schut. ”Practitioners guide to MLOps: A framework for continuous delivery and automation of machine learning.”. https://services.google.com/fh/files/misc/practitioners_guide_to_mlops_whitepaper.pdf（閲覧日：2022 年 7 月 20 日）.
[3] iterative.ai. ”Open-source Version Control System for Machine Learning Projects”. http://dvc.org/（閲覧日：2022 年 2 月 8 日）.
[4] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, Vol. 28, , 2015.
[5] M. Baker. 1,500 scientists lift the lid on reproducibility. *Nature News*, p. 452, 2016.
[6] J. Pineau, P. V. K. Sinha, V. Larivière, A. Beygelzimer, F.d ’ Alché-Buc, E. Fox, H. Larochelle. Improving reproducibility in machine learning research: a report from the neurips 2019 reproducibility program. *Journal of Machine Learning Research*, 2021.
[7] LF Projects, LLC. ”An open source platform for the machine learning lifecycle”. https://mlflow.org/（閲覧日：2022 年 2 月 8 日）.
[8] Pachyderm Inc. ”The Leader in Data Versioning and Pipelines for MLOps”. https://www.pachyderm.com/（閲覧日：2022 年 2 月 8 日）.
[9] A. Barrak, E. E. Eghan, and B. Adams. On the co-evolution of ml pipelines and source code-empirical study of dvc projects. In *Proceeding of the 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 422–433, 2021.
[10] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, pp. 39–45, 2018.
[11] I. Karamitsos, S. Albarhami, and C. Apostolopoulos. Applying devops practices of continuous automation for machine learning. *Information*, p. 363, 2020.
[12] iterative.ai. ”Get Started: Data Pipelines”. https://dvc.org/doc/start/data-management/pipelines（閲覧日：2022 年 7 月 20 日）.
[13] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the genetic and evolutionary computation conference 2016*, pp. 485–492, 2016.
[14] I. Drori, Y. Krishnamurthy, R. Rampin, R. de Paula Lourenco, J. P. Ono, K. Cho, C. Silva, and J. Freire. Alphad3m: Machine learning pipeline synthesis. *arXiv preprint arXiv:2111.02508*, 2021.
[15] ”update dataset, aerubanov/Proj_Air_Quality@74a14c1”. https://github.com/aerubanov/Proj_Air_Quality/commit/74a14c1（閲覧日：2022 年 9 月 22 日）.
[16] ”Fix typo in DVC config, BlueBrain/Search@f08e086”. https://github.com/BlueBrain/Search/f08e086（閲覧日：2022 年 9 月 22 日）.
[17] ”BlueBraing/atlas-annotation@10b426d”. https://github.com/BlueBrain/atlas-annotation/blob/10b426d/data/dvc.yaml（閲覧日：2022 年 9 月 22 日）.
[18] ”mtszkw/mnist-workflow”. https://github.com/lp-code/TwitterSARai/tree/f1cc929/dvc.yaml（閲覧日：2022 年 9 月 22 日）.

定形的ログメッセージの除去とクラスタリングによる異常動作ログの検出方法の提案

A method for detecting log anomalies based on log messages and clustering techniques

上田晃義 * 尾花将輝 † 花川典子 ‡

あらまし 近年のシステムはパッケージソフトやクラウドサービス等のブラックボックス化されたソフトウェアを利用したものが多い．このようなシステムにおける障害対応はシステムから出力されるログを調査することが一般的である．しかし，大規模なシステムになるほどログの量が膨大となり，障害原因の調査にはコストが掛かる．そこで本論文では出現頻度の低いログメッセージには異常動作が含まれる可能性があるという考えをもとに，ドメインに依存しない少数のログ抽出による異常動作を検出する手法の提案を行う．膨大に出力されるログから少数の疑わしいログメッセージを抽出することにより，障害調査のコスト削減や，早期に障害を特定する手法への発展が期待できる．また，提案手法を実際に運用されている複数のアプリケーションに適用した．結果，抽出されたログメッセージには障害が含まれていることを確認できた．

1 はじめに

近年のシステムはスクラッチ開発を行わずパッケージソフトウェアや SaaS，オープンソースソフトウェア等を組み合わせたシステムが多い．このようなシステムはソフトウェア単体での品質は高い傾向にあるが，ソフトウェアがブラックボックスであるため障害の検知や調査にはソフトウェアが出力するログを用いて調査を行うことが一般的である．しかし，このようなログを用いた障害の調査は莫大なログメッセージから異常動作の可能性のあるログメッセージを絞り込む必要があるため障害の調査にはコストが掛かる．または障害の発生時刻から障害原因を調査することも可能であるが同時刻に類似するログメッセージが出力された場合にも，どのログメッセージが障害原因かの分析にはコストが掛かる．このように，近年のシステムは規模や複雑さが増しており，障害の調査には膨大なコストが掛かる傾向にある [1]．

またログは INFO，WARN，ERROR 等のログレベルにより分類し出力することが多い．これらは一般に RFC5424 [2] が定める重大度コードを元に PSR-3 [3] や，PEP282 [4] で定められているが，実際には開発者やプロジェクトによってログの出力形式や方針は異なる．そのため，開発者が想定した Error と想定されない Error を同じ粒度で扱う問題が発生し，本当に障害なのかわからないため，障害原因を調査することの妨げとなる可能性も挙げられる．

そこで本稿では，処理ごとに出力されるメッセージ（これを本稿ではログメッセージと呼ぶ）のうち，出現頻度の低いログメッセージには障害が含まれる可能性が高いというコンセプトをもと，ドメインに依存しない少数のログを検出する手法を提案する．具体的にはログの整形後，定形的に出力されるログメッセージを削除し，TF-IDF によるベクトル化とクラスタリングを行った結果，クラスタサイズの小さいものを抽出する．本概念を用いることで運用中にソフトウェアから出力される膨大なログから，異常動作の可能性のあるログメッセージを抜粋することで分析コストを削減する他にも，バグの早期発見や，脆弱性をついた攻撃の検知等に発展できる．

2 章では本研究の関連となる研究の調査と比較を行い，3 章では具体的な提案内容

*Akiyoshi Ueda, 大阪工業大学

†Masaki Obana, 大阪工業大学

‡Noriko Hanakawa, 阪南大学

を延べ，4 章では実際に運用しているアプリケーションに対して本手法を適用し，5 章で得られた結果についての考察を行い，6 章でまとめと今後の課題を述べる．

2 関連研究

運用中のシステムの障害の調査においてログの分析は重要な手段となる．古くから運用者の経験に基づく「error」や「failed」のような単語から障害の分析が行われてきた．しかし，システムの大規模化や多種化に伴い，ログの量や複雑さが増している．そのため，ログ分析の自動化や支援を目的とした様々な分析アプローチが提案されている [5]．その 1 つにルールベースに基づいたログ分析アプローチがある [6]．これは，システムアーキテクチャや UML 図等を活用したログ出力をシステムに組み込むことで，分析しやすいログを生成する．しかし適用するシステムに依存することや，既知の障害のシナリオに基づくことによるシナリオ作成自体のコストが掛かることや，想定されない障害においては分析が難しいという課題があり，本提案ではシステムの種類に依存しない点が異なる．

ログパターン（ログキー）から障害や異常動作の検出を行う手法がある．Xu らは，ソースコードのログ出力部分や直接バイナリからログパターンを抽出する手法を提案している [7]．実際にログを出力するコード片を利用しているため，出現頻度の少ないログに関しても解析を行える利点がある．同様にソースコードから取得したログパターンとニューラルネットワークを用いて障害検出を行う手法も提案されている [8]．しかし，パッケージソフトウェアやクラウドサービスと言ったソースコードが入手できないようなアプリケーションは対象としていない．本提案では実際に出力されたログのみを用いて障害の検出を行う．

ログからパターンを抽出するログパースと呼ばれる研究も数多く行われている．解析手法は最長共通部分によるものや，階層的クラスタリング，反復分割，解析木を用いたものなど様々存在する [9] [10] [11] [12]．これらログパーサの手法は他のログ分析手法の前処理としても用いられている．本提案手法においては，オンライン解析で，解析木を用いた手法である Drain [12] と呼ばれるものを一部に取り入れている．また，ログと自然言語処理技術の BERT による過去の正常ログのログパターンとの比較によって障害を検出する手法も存在する [13]．更にログパターンの状態遷移を学習することでログから障害を検出する手法も存在する [14]．しかし，これらは予め正常ログのデータセットを用いて学習を行う必要があり，運用中のログから異常動作のログを検出する本提案とは考え方が異なる．

また，システムログのメッセージをグラフ構築，ランダムウォークと Word2Vec によるグラフ埋め込み・独自のクラスタリング手法を用いた異常検知を行うことで企業内のサイバー脅威の検出を試みる研究も存在する．同一ユーザによる操作や同一オブジェクトの操作のログメッセージを関連付けて分析を行っている．また，Unix システム等のシステムログのルールを評価する研究もある [15]．Simple Event Correlator というログ分析ツールに適用するルールをイベントごとに分類している．これらは本研究の異常動作のログを検出するという目的は同じであるが，サイバー脅威や，Unix システムのログを対象としており，アプリケーションを対象に含めた本研究とは異なる研究である．

3 提案手法

3.1 ログの曖昧性と問題点

一般的にログとは日付や IP，ユーザ ID 等の情報とユーザの操作状況や OS，ミドルウェア，アプリケーションの一連の処理状況を自然言語や JSON といった半構造形式で記録された物を指す．本稿ではこの一連の処理ごとに出力されたメッセージをログメッセージと呼ぶ．しかし，ERROR レベルのログや「エラー」と記載されているログメッセージが必ず障害（システムフォルト）に繋がるとは限らない．

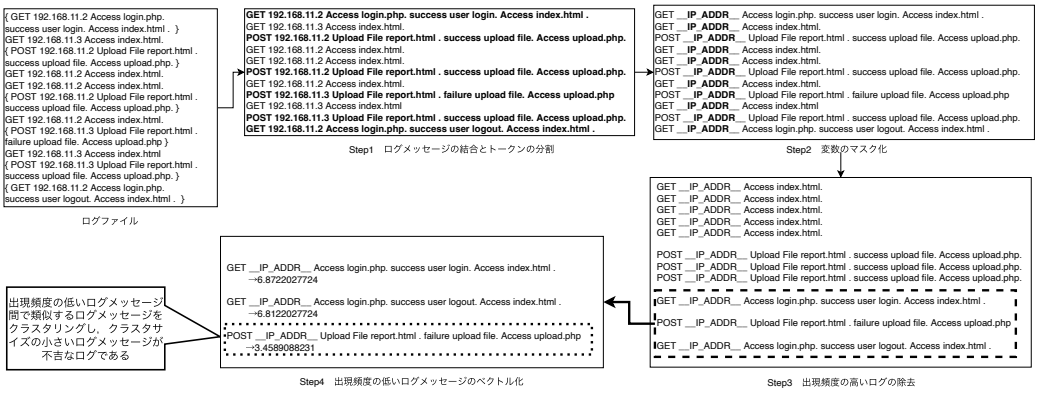


図 1　提案手法の概念図

例えばユーザのログイン時に「Login error」というログメッセージが ERROR レベルで出力されたとする．この時，単純に「error」と記載されているため「障害」に繋がるとも考えられるが，セキュリティの観点から長時間放置されたブラウザからセッション情報が削除され，再度ユーザが操作したため「Login error」と表示する事は「正常」な動作である．このように，ERROR レベル，または例外処理の Exception といったログメッセージは状況によっては正常，異常と捉え方が変わる問題がある．

また，逆の事例も考えられる．INFO レベルのログはユーザのログインや特定の機能を実行した時に記録される「正常」な動作ログである．しかし，アクセス不可であるファイルを参照したり，不正なデータをアップロードができる言った予期しない動作でも正常な動作として INFO レベルのログに記録される可能性がある．これらは，アクセス権限の設定ミスや開発者が想定しない挙動といった障害のため，本来ならば ERROR や CRITICAL と言ったログレベルで記録する必要がある．

このような問題はログレベルの指標が厳密に守られていない，または開発者の想定を超えるような例外によって発生する．特に，INFO レベルのログメッセージしか出力されない障害は危険度が高く，システム運用者や開発者が，障害が発生している事に気が付きにくく，報告があったとしても ERROR として記録されていないため膨大なログから同様の動作を分析する必要がある．

そこで，本提案では，運用中に出力されるログの内，少数のログメッセージを抽出することで異常動作の可能性のあるログの抽出手法を提案する．システムから出力されるログの殆どはアクセスやタスク処理の結果等の動作記録である．一般的に既に運用されているシステムはテスト工程を通過しているため，運用段階のログの殆どが障害と関係のないログメッセージである．しかし，障害発生やシステムの脆弱性を突いた操作時には普段正常に動作している時とは異なるログメッセージが障害の結果として出力されることがある．つまり，運用中のシステムにおける，障害を示すようなログメッセージは極めて少数であると考えられる．そのため少数のログを抽出し，システム運用者や開発者に示すことで，莫大なログメッセージを分析せずとも，抽出されたログメッセージだけを分析することで，障害や脆弱性を突いた攻撃等に，より迅速に対応することができる．

3.2　異常動作の可能性のあるログの抽出手法

異常動作の可能性のあるログの検出方法は定型的なログメッセージを削除後に類似するログメッセージをクラスタリングし，その結果の内クラスタサイズが小さいログメッセージ，つまり少数のログメッセージ群を抽出することによって行う．少数のログメッセージには障害が含有する可能性が高いというコンセプトは従来から存在するが，従来の研究では事前にシステムから出力されるログパターンであった

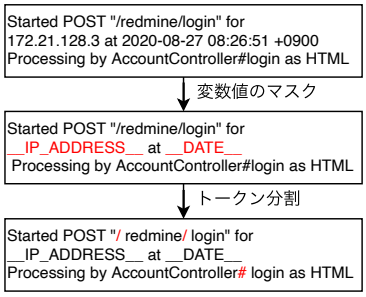


図 2　ログの前処理

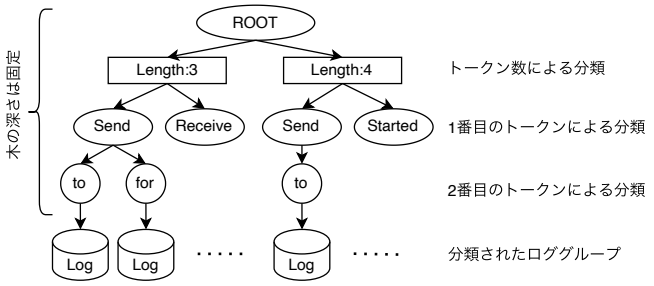


図 3　Drain の概要

り，過去に障害原因となったログの情報を入力する必要があった [8] [16] [17].

本提案の概念図を図 1 に示す．本手法は大きく 4 つの工程に分かれており，図 1 の Step 1 では出力されるログの整形，Step 2 ではログに含まれる変数のマスク，Step 3 では定形的に出力されるログメッセージの除外，Step 4 でクラスタリングの結果，規模の小さなクラスタのログを検出する．次項から各 Step の詳細について述べる．

3.2.1　Step 1: ログの整形

Step 1 ではログの出力形式をある程度統一する．システムによって出力されるログの形状が XML や JSON の様な半構造形式であったり，1 回の処理が複数行で出力されるケースがある．Step 2 以降の工程では，ログメッセージを行ごとに処理するため，他のシステムのログメッセージと同じ粒度に統一する必要がある．

そこで，複数行にまたがって出力されるログメッセージを可能な限り 1 行に変換する．ただし，本工程は対象とするドメインによって区切る手法が変わる．例えば本適用事例では「started」から「completed」が 1 回の出力であった．このようにドメインによっては改行，記号，文字列といった出力を区切るための手法が変わることがあるが，本工程は検出精度向上のための前処理であるため，可能な限り整形を行うものとする．

3.2.2　Step 2: 変数値のマスクとトークンの分割

本手法ではトークンの並びやその出現頻度によって最終的にはクラスタリングを行う．そのため，IP アドレスやユーザ名といった情報が特徴のある単語として処理されてしまう可能性がある．そこで，本工程ではノイズ除去のために自明な変数をマスクすることでノイズの影響を減らす．

本稿では単純に IP アドレス，日時の値を正規表現とドメイン知識を用いて変数種別に応じた固定文字列に置換する．例えば図 2 の IP アドレスであれば「__IP_ADDRESS__」のように置換している．また，ユーザ情報等はデータベース等から取得することでマスクすることが可能であると考える．しかし，本工程も Step 1 同様にログの検出精度を向上させるものであり，例えばユーザ ID やタスク ID も本工程に含むことができるがドメインによって形式が異なるため，どこまで対応する必要があるかは今後の課題とする．

続いてトークンの分割について述べる．製品によってはログメッセージの単語の区切り文字にタブや半角スペースを使わないことがある．例えば，図 2 にある「AccountController#login」の「#」が該当する．「AccountController」はクラス名を示しており，「#」以降の「login」はメンバ関数を示している．事例以外にもログのフォーマットが CSV 形式である等からトークンの区切りによる分割を行う．

3.2.3　Step 3: 出現頻度の高い同種のログメッセージの分類

Step 3 では頻繁に出力される定形的なログメッセージの除去を行う．ログ全体に自然言語処理による類似度測定を行うと，同種のログメッセージの影響を大きく受ける．そこで，本工程では定形的なログメッセージをある程度パターンで分類し，膨大に出力される同種のログメッセージの除去を行う．本稿ではこの工程を既存の

ログパース手法である Drain [12] を用いることで実装している．

図 3 に Drain の概念図を示す．Drain は最初にログメッセージのトークン数でおおまかにログメッセージを分類し，その後各トークンが一致しているかの分類木を作成することで同種のログメッセージを分類する手法である．グループ化するトークンの一致率の閾値はハイパーパラメータである．また，類似の手法である LogMine [18] や LenMa [19] よりも精度が高い [20]．

Drain は頻繁に出力されるログメッセージの分類手法としては優秀であるが少数のログメッセージが多数分類される特徴がある．そのため，本工程で行う同種かつ大量に出力されるログメッセージを分類するには向いている手法である．また，Step 2 の工程を Drain がある程度行うが，より精度を上げるためにも Step 2 の変数のマスクは可能な限り行うことが望ましい．

3.2.4 Step 4: 出現頻度の低い同種のログメッセージ群のクラスタリング

Step 3 で定形的なログメッセージの除去を行い，残った出現頻度の低いログメッセージをさらに自然言語処理にてベクトルを作成しクラスタリングを行う．

障害が含まれるようなログメッセージには特有の単語が多く含まれることが多い．例えば，Exception 等の単語がある場合にはエラーが発生したと考え調査を行う．しかし，ファイルをアップロードしてから利用するシステムにおいてはアップロードせずにシステムを利用した場合は File Exception がログに記録される．このようなシステムの仕様の場合と障害発生時における Exception とは意味が大きく異なる．システムのドメインによってログ内の単語の意味が変わるため，Step 4 では Step 3 で作成された小さいグループに対してクラスタリングを行い，規模の小さなクラスタの抽出を行う．

本稿の適用ではドメインの中から特徴のある単語を算出するために単語の出現頻度に基づいて計算される TF-IDF を用いた．本工程はベクトルが作成できれば良いので Doc2Vec や Word2Vec 等でも可能であるが辞書の作成はドメイン依存のベクトル生成となるため本手法の概念に反すると考える [21]．また，クラスタリング手法としては K 平均法（k-means）を用いている．

4 提案手法の適用

4.1 適用アプリケーション

本手法を Redmine(プロジェクト管理ツール)，Jitsi Videobridge(ビデオ会議ツール)，Opennebula(クラウド基盤) と大学教育システム（ポータルサイト，以下教育システムと呼ぶ）の 4 つに適用した．それぞれログのフォーマットは異なっており，またログの期間や行数等の概要は表 1 に示す通りである．どのアプリケーションもリリース版を使用しており，安定動作が期待されるバージョンである．本章では，これらのアプリケーションに本手法を適用し，少数のログの検出を行う．少数なログには異常動作のログメッセージ等も考えられるが，本章ではシステム障害が含まれるか調査を行う．また，本提案は異常動作を検知することを目的としており，サイバー攻撃等も検出可能であるが，本適用事例では障害のみを対象とした分析を行った．

表 1　適用したアプリケーションログの概要と前処理の結果

アプリケーション (バージョン)	適用した ログの期間	行数	単語種	単語種 (マスク後)	単語種 (トークン分離後)
Redmine(4.0.0)	16 ヶ月	38,430	32,054	20,627	16,420
Jitsi Videobridge(2.0.6293)	3 ヶ月	3,664,385	306,557	293,159	292,090
Opennebula(6.0.0.1)	9 ヶ月	24,031,912	89,438	3,118	3,118
教育システム	13 日	1,900,094	327,346	20,552	20,300

4.2 パラメータの決定

ログ検出を行うためのパラメータの決定方法について述べる．Step 1 ではログの整形で行うが，今回の適用事例では Redmine のみが対象となり，started から completed が 1 つの処理であったためそれらを結合した．また，ログは全て自然言語で書かれており，半構造形式のログは無かった．続いて Step 2 の変数のマスクでは，すべての適用事例において IP アドレスと日時のみを固定文字に置き換えた．また，トークンの分割に関しては「.」,「/」,「,」,「=」といった記号で分割可能であったため，これらの記号の後ろに半角スペースを挿入し，1 つのトークンを複数のトークンに分割した．これらの各処理を行った結果の単語の種類数を表 1 の単語種に示す．

Step 3 では定形的なログメッセージを除外するために本稿では Drain を用いた．Drain は木構造の深さを決定する必要があり，本稿では引用元の論文と同じ 4 とし，トークン一致率の閾値は 0.55 とした．引用元の論文では 0.5 というパラメータを利用して分析を行っていたが，0.5 ではトークン数の少ないログメッセージの場合，全く異なるログパターンのログメッセージが過剰にグループ化されたため今回は 0.55 とした．Drain は本来ログメッセージ中に出現する変数を抽出すること目的としており，本稿の定形的なログメッセージを除外という目的での利用とは異なるため値を変更したが，厳密な値の決定方法は今後の課題とする．なお，Drain の実装には，ログパーサベンチマークを参考にした [20] [22]．また，定形的なログメッセージとして除去する基準として，Drain から出力されるログメッセージ群のうち，類似のメッセージだと判定されたログメッセージの上位 10%を定形的なログメッセージとして扱い，残りのログメッセージで Step 4 のクラスタリングを行った．

最後に Step 4 では TF-IDF を用いてベクトルを作成し，k-means にてクラスタリングを行った．クラスタ数の決定には二乗和誤差 (SSE) の最大曲率点を Kneedle アルゴリズムから決定し，クラスタリングを行った [23]．また，最大曲率点の確認にはエルボー法を用いて目視での確認も行い，クラスタサイズが小さいものを少数のログとして算出するが，今回の適用事例ではクラスタサイズが下位 25%に属するログメッセージ群を抽出した．

4.3 結果

本提案を適用した結果を表 2 に示す．また，図 4 は Step 4 で算出されたクラスタサイズの推移を示した結果である．横軸は個々のクラスタ，縦軸はクラスタサイズとなっている．教育システムのみ，クラスタサイズの分布が大きいため，クラスタサイズが小さい部分だけ拡大し表示している．また，Jitsi Videobridge は，クラスタ数が最も少ない結果となった．

適用結果より Redmine で 35 個，Jitsi Videobridge で 14 個，Opennebula で 45 個，教育システムでは 9 個のログが検出された．また，検出されたログを全て調査したところ，教育システム以外の検出されたログには障害と関係のあるログメッセージが含まれていることも確認できた．各適用結果について詳細に述べる．

Redmine は 1 つの処理を複数行に出力するログフォーマットになっており，Step 1 を唯一行った適用事例である．また，表 2 から Drain で検出されたグループ数が 4 つのアプリケーションの中で最も多い結果となった．このことから一連の処理ご

表 2　Drain で検出されたグループ数と k-means による結果

アプリケーション	Drain で検出されたグループ数	k-means のクラスタ数	検出されたログ	検出されたログに含まれていた障害件数
Redmine	2,032	140	35	1
Jitsi Videobridge	154	55	14	1
Opennebula	1,266	177	45	3
教育システム	21	310	78	0

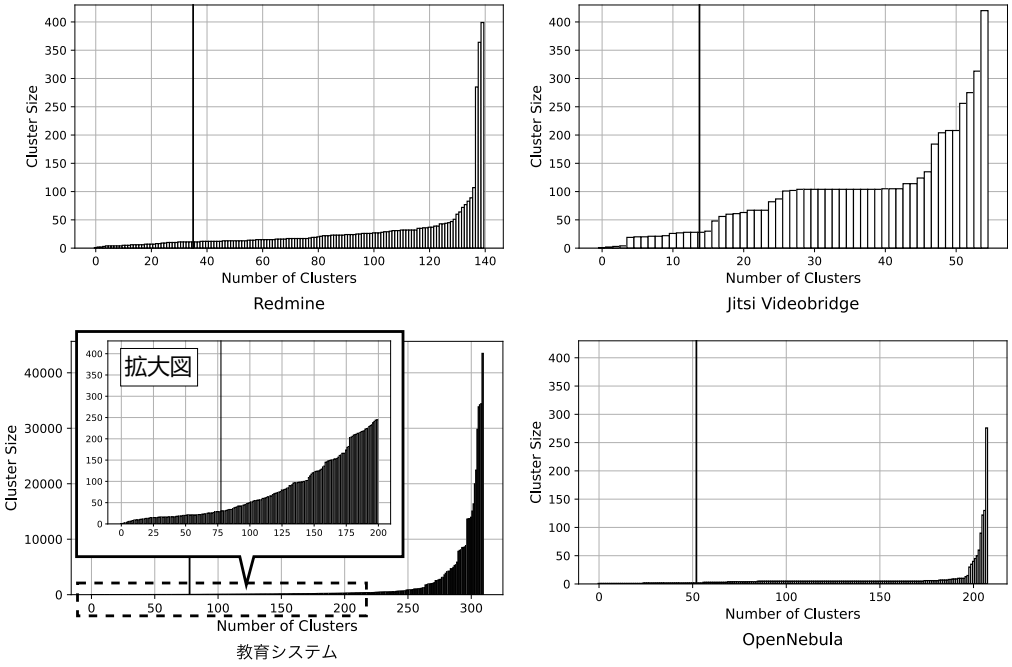


図 4　クラスタに含まれるログメッセージの数

とにログを出力する，または他の製品と連携されるようなログの場合には実行パラメータを多く含んでしまい，結果 Step 3 の定形的なログメッセージの除去が難しい傾向になることがわかる．また，抽出されたログを分析したところ，ログイン時に外部の認証サーバとの接続エラーにより，正常にログインできないという障害が発生したログメッセージが書かれていた．また，障害は後のアップデートにより修正がされていた．この障害ログは複数の類似したログメッセージが含まれており，各ログメッセージが Step 3 では別の種類のログとして分類されていたが，Step 4 のクラスタリングではログメッセージが類似するため同種のログとして扱われていた．

続いて，Jitsi Videobridge について述べる．Jitsi Videobridge は単語種が多いアプリケーションであり，Step 2 における変数のマスク後も単語種にあまり変化がなかった．この理由として，ログメッセージに何らかの ID やパケットサイズ等の変数が多く出力されており Step 2 のマスクがうまくできないログであった．また，検出されたログを調査したところ，「SocketException」を含むログメッセージが出力されていた．調査した結果，該当時刻にユーザがビデオ会議中にネットワーク障害等に関係なく会議が切断される事象が確認されたが，直接的な原因を発見することはできなかった．本事例は，Exception や Error といった単語は障害を示す単語として挙げられるが，そのような辞書に基づいていないにも関わらず Exception という単語が含まれるログメッセージを検出できた事例である．

Opennebula は約 2400 万行と最もログメッセージ数が多いログであり，また，図 4 より Step 4 のクラスタリングサイズに差がないことがわかる．また，表 1 の単語の種類は最小である事から定形的なログメッセージを出力し続けるログであった．そのため Step 3 の手法で定形的なログメッセージを大幅に削除することでログの絞り込みがうまくいった事例である．また，検出されたログを調査したところ複数の障害を示すログメッセージを発見した．構成設定ミスによる PCI デバイスとディスクファイルのパーミションエラー，直接的な原因が不明の DB のデータ更新漏れに

よるファイルアクセスエラーの障害であった．原因が不明の障害に関しては実際の開発者に問い合わせなければ詳細は不明であるが，構成設定ミスに関しては確かに設定ミスをしていた事があり，本事例はコーディングミスによる障害だけでなく，インフラの設定ミスに関する障害も検出できることがわかった事例であった．

最後に教育システムのログについて述べる．本ログのみソリューションとして開発されたプロジェクトであり事前に障害情報を把握していたログである．当ログは図 4 から Step 4 の対象となるログメッセージ数が最も多く，Step 3 で定形的なログの抽出がうまくいかなかった事例である．このログは出力される情報が少ないという特徴があった．加えて前後の処理との関係性が強い傾向にあるログであったため検出されたログの中に障害が含まれなかった．しかし，検出されたログを調査した結果，「/..」にアクセスしようとする，パストラバーサルを試みたと思われるログがあったり，Internal Server Error を示すログメッセージが出力されていた．これらは疑わしい挙動をするログメッセージであり，本来ならば調査される対象であるが，障害情報の中にはこれらの内容が含まれていなかった．つまり，本来障害として対応すべき内容にも関わらず報告が無く，結果改修されなかった可能性が考えられる．

以上のように，全てドメインやログの構造が異なる，または出力されるキーワードが異なる製品に適用でき，またソフトウェアのバグだけでなく設定ミスといった人的ミスによる障害等も検知できる事を示唆できたと考えられる．

4.4 検証

適用事例ではリリース版のアプリケーションを運用し，その時に出力されたログデータから障害として少数のログ検出を行った．そのため，通常利用では障害がほぼ発生せず，検出されたほとんどのログが正常に動作したログという結果であった．そこで OSS プロジェクトでバグとして報告された障害を意図して発生させ，その動作のログメッセージを適用事例のログデータに埋め込み，埋め込んだログメッセージが本手法によって検出できるかの検証を行った．

検証に用いたバグは，Redmine の「BOM 付き UTF-8 のファイルのインポートに失敗する」である．Redmine からエクスポートされるファイルが BOM 付きのファイルにも関わらずインポートすると「このファイルは CSV ファイルではない」というエラーメッセージが出る．また，このバグはリビジョン 17786 で修正されており，運用中の Redmine のバージョンでは修正されていないバグである．

当バグを意図的に発生させ，ログメッセージを適用事例のログデータに埋め込こんだ．対象のログメッセージは「エンコード指定の内容送信からエンコード指定画面から遷移しない」である．また，正常に動作したログメッセージと比較を行った結果，障害が発生したログメッセージにはリダイレクトメッセージが欠落しており，Error や Critical といったネガティブワードは一切含まれていなかった．

新たに，障害が含まれたログメッセージを埋め込んだデータセットを再度 4 章の適用事例と同様の手順で少数のログ検出を行った．結果，埋め込みを行ったログメッセージを本手法によって検出することができた．

5 考察

5.1 適用結果による考察

本手法の Step 1 では 1 つの処理を 1 つのログメッセージとして細分化を行うが，情報量の少ないログを出力する製品は処理の前後関係や他のアプリケーションのログも連結する必要がある．そのため，適用事例の教育システムでは障害が含まれたログメッセージがあるにも関わらず，該当のログメッセージを検出することができなかった．このように，障害のあるログメッセージを検出するためには前後の動作や他のアプリケーションログの連結をすることが望ましい事がわかる．

また，本稿では k-means によるクラスタリングの後に閾値で少数のログの検出を

行うが，閾値の調節もシステムのドメインや，分析する時期によって異なる．本稿の適用事例では 25%で行っているが，分析する製品のログ構造や，情報量やシステム規模，デバッグモードでの実行なのか，または他の製品との連携の有無等のドメインの性質や環境によって大きく異なる．他にも，システムテスト時，リリース直後，安定期等の時期によってもログの調査対象の範囲が大きく異なる．このように，閾値の調整は環境や状況，時期によって大きく変わるため，ドメインごとに調整を行うことでより正確に少数のログを検出することができる．

本提案は過去の類似研究のように学習用の過去のログやソースコード，ログパターン等を必要としない．特に深層学習を用いた異常検知の手法における障害や異常動作をしたデータセットの収集は困難であり，単純には類似研究と比較できなかったため，類似研究との比較は今後の課題とする．

5.2 本提案の限界

本稿では稀に出力されるログメッセージの抽出を行うことによって，障害の検出を試みた．そのため抽出結果として稀に実行される機能のログメッセージや，起動時のみに出力される様なログメッセージも検出されてしまう．本稿で述べる少数のログはそれらのログメッセージと障害の可能性が含まれたログメッセージを区別することができない．これに対しては運用者が検出されたログに含まれる正常なログメッセージにマークをし，そのログを除外することにより，誤検出を減らせる．

また反対に障害時に発生するログメッセージが抽出結果に含まれないパターンも考えられる．障害のログメッセージが膨大な件数出力される場合や，障害が発生しているが正常なログメッセージと差分がない場合が考えられる．前者については出力が多いため運用者による障害メッセージの発見が比較的容易であると考えられるが，後者に関しては手動でログ分析を行った場合でも発見することが困難であると考えられる．そのため，出力するログレベルをより詳細な設定に変更することや，ソフトウェア自体のログメッセージ出力処理の見直しが必要である．

6 まとめ

本研究では出現頻度が低いログメッセージには障害が含まれる可能性が高いという考えにもとづいたドメインに依存しない異常動作のログを検出する手法を提案した．定形的なログメッセージの除去を行った後にクラスタリングされたうち，少数のログメッセージ群を異常動作の可能性のあるログとして検出する手法である．本提案を実際している 4 つのアプリケーションに適用し，3 つのアプリケーションから抽出した少数のログには障害が含まれていることを確認できた．また，3 つのアプリケーションそれぞれの性質が異なる障害原因であるにも関わらず障害を検出する事が確認できた．今後の課題は本手法によって抽出されず正常ログと判断されたログに障害が含まれないかの精査や，関連する他の製品のログと結合したログに対して本手法が有用であるかの検証，既存の類似研究との定量的比較を行う．

謝辞 本研究は JSPS 科研費 JP19K20244 の助成を受けた．

参考文献

[1] 松田晃一, 村岡恭昭, 斎藤毅. 情報システムの障害状況 2019 年後半データ. Technical report, 独立行政法人情報処理推進機構社会基盤センター, 2020.
[2] R. Gerhards. The syslog protocol. RFC 5424, RFC Editor, March 2009.
[3] PHP-FIG. Psr-3: Logger interface. PSR 3, PHP-FIG, 2013.
[4] Trent Mick Vinay Sajip. Pep 282 – a logging system. PEP 282, 2002.
[5] 尾花将輝, 花川典子. システムログとアプリケーションログを用いた障害起因特定のためのフレームワーク提案. コンピュータ ソフトウェア, Vol. 38, No. 3, pp. 58–74, 2021.
[6] Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia. Event logs for the analysis

of software failures: A rule-based approach. *IEEE Transactions on Software Engineering*, Vol. 39, No. 6, pp. 806–821, 2013.
[7] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, pp. 117–132, New York, NY, USA, 2009. Association for Computing Machinery.
[8] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pp. 1285–1298, New York, NY, USA, 2017. Association for Computing Machinery.
[9] Min Du and Feifei Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 859–864, 2016.
[10] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 Ninth IEEE International Conference on Data Mining*, pp. 149–158, 2009.
[11] Adetokunbo A.O. Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pp. 1255–1264, New York, NY, USA, 2009. Association for Computing Machinery.
[12] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pp. 33–40, 2017.
[13] Haixuan Guo, Shuhan Yuan, and Xintao Wu. Logbert: Log anomaly detection via bert. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021.
[14] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 Ninth IEEE International Conference on Data Mining*, pp. 149–158, 2009.
[15] John P. Rouillard. Refereed papers: Real-time log file analysis using the simple event correlator (sec). In *Proceedings of the 18th USENIX Conference on System Administration*, LISA '04, pp. 133–150, USA, 2004. USENIX Association.
[16] 高田哲司, 小池英樹. 見えログ：情報視覚化とテキストマイニングを用いたログ情報ブラウザ. 情報処理学会論文誌, Vol. 41, No. 12, pp. 3265–3275, 12 2000.
[17] 渡辺幸洋, 松本安英. メッセージパターン学習による障害発生検知. インターネットと運用技術シンポジウム 2009 論文集, 第 2009 巻, pp. 23–30, dec 2009.
[18] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pp. 1573–1582, New York, NY, USA, 2016. Association for Computing Machinery.
[19] Keiichi Shima. Length matters: Clustering system log messages using length of words, 2016.
[20] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 121–130. IEEE, 2019.
[21] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, Vol. 32 of *Proceedings of Machine Learning Research*, pp. 1188–1196, Bejing, China, 22–24 Jun 2014. PMLR.
[22] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. An evaluation study on log parsing and its use in log mining. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 654–661, 2016.
[23] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pp. 166–171, 2011.

要求獲得における質疑応答履歴のグラフデータベースシステムの実現

Implementing a Graph Database System for Q & A records in Requirements Elicitation

今堀 由唯[*] 加藤 潤三[†] 林 晋平[‡] 大西 淳[§] 佐伯 元司[¶]

あらまし ソフトウェア開発の要求仕様書の作成過程では顧客の要求を集めて分析者が要求リストを作成していく. しかし質疑応答を繰り返すことで変更が繰り返され議論に矛盾が生じたり，議論をせずに未承諾事項が残ったりすることが度々発生する. これらが発生してしまうと，顧客の意図とは合わない要求，要求の欠落が起こったりする. 要求分析者が質疑応答結果に基づく変更内容をすべて把握しておくことはシステム等の規模が大きくなればなるほど困難であるという問題がある.

本研究では要求リストを作成するための顧客と分析者の質疑応答の履歴をグラフデータベースに構造化して蓄積し，データベースシステムの検索機構や可視化機構を使って，質疑応答履歴における要求リストの作成を支援するシステムの実現を目的とする．本論文ではまずそのための質疑応答を格納するためのグラフデータベースの設計と実装について述べる.

質疑応答履歴をオブジェクト指向でモデル化し，Neo4j でデータベースを構築する．Neo4j のデータベース操作言語 Cypher を使い，未承諾事項の検索などの機能が実現できることを確認した.

1 はじめに

ソフトウェア開発では要求仕様書の作成過程で顧客の要求を集めて分析者が要求リストを作成していく．IPA は要求定義に求められる 3 つのことの 1 つに”要求仕様を「抜け」「漏れ」「あいまい」なくシステム開発につなげる”ことを挙げている [1].

また JUAS の調査にて「仕様が非常に明確な場合は「(以降の開発過程での) 変更なし」＋「軽微な変更が発生」は 97.6 %，仕様が非常にあいまいな場合は「大きな変更が発生」＋「重大な変更が発生」が 69.7 %になっており特徴が表れている.」ということがわかっている [2].

これらのことから要求リストを正確かつ明確に作成していく必要があることがわかる．通常，要求分析者はステークホルダからの初期要求リストをもとに，彼らと議論を重ね完全なものへと仕上げていく．議論中に，未議論事項や未承諾事項などの「抜け」や「漏れ」が残っていると，議論の最終成果物である要求リストにも「抜け」や「漏れ」が生じたり，ステークホルダの意図とは異なる要求が入ったりする. 要求分析者が未承諾事項などの質疑応答履歴をすべて把握しておくことは，システム等の規模が大きくなればなるほど困難であるという問題がある. そのため，未承諾事項などの議論の内容を保持するシステムが必要である．これまで，発話行為論などを用いて質疑応答履歴を構造化し蓄積し活用しようとする研究 [3] [4] があるが，質疑応答から生じる要求リストの低品質化などの対処支援は十分に行われておらず，蓄積した履歴の有用な活用法が問題である.

本研究では質疑応答履歴を蓄積し，その履歴を活用することにより要求リストの作成を支援するシステムの実現を目的とする．そのため，質疑応答履歴をデータベー

[*]Yui Imahori, 南山大学

[†]Junzo Kato, 独立コンサルタント

[‡]Shinpei Hayashi, 東京工業大学

[§]Atsushi Ohnishi, 立命館大学

[¶]Motoshi Saeki, 南山大学

ス化することを考え，データベースの操作言語を用いることにより，質疑応答での低品質化を防ぐ支援を考える．本論文ではまずそのための質疑応答を構造化し格納するためのグラフデータベースの設計と実装について述べる．

グラフデータベースの設計と実装には Neo4j [5] を使用し，要求リストを作成するための顧客と分析者の質疑応答履歴を構造化して蓄積する．データベースシステムの検索機構や可視化機構を使うことで未承諾事項の確認，議論の内容等を把握することが可能になる．

2 アプローチ

2.1 グラフデータベースの使用

質疑応答が進むにつれて，参加者の議論や要求リスト，およびそれらの間の関係が生成されたり削除されたり動的に変化していく．そのため，ノード間の関係をエッジでつなぐグラフデータベース [6] は Excel などの表形式での管理と比較し，これらの動的な更新に柔軟に対応できると考えた．また，グラフ表示による履歴の可視化などが可能となる．そのため，本研究では顧客と分析者の質疑応答履歴をグラフデータベースである Neo4j を用いて構築していく．

Neo4j は，Neo Technology 社によって開発されたグラフデータベースシステムである．Cypher という宣言型のグラフクエリ言語を使用しデータ操作を行う．プロパティグラフモデルと呼ばれるモデル化手法に基づいており，自然に近い形でデータをモデル化することができるため Neo4j を選択した．

2.2 グラフモデル定義

質疑応答履歴を格納するモデルを設計するにあたり，オブジェクト指向でモデル化し，それを使ってグラフデータベースを設計した．それを図 1 に示す．

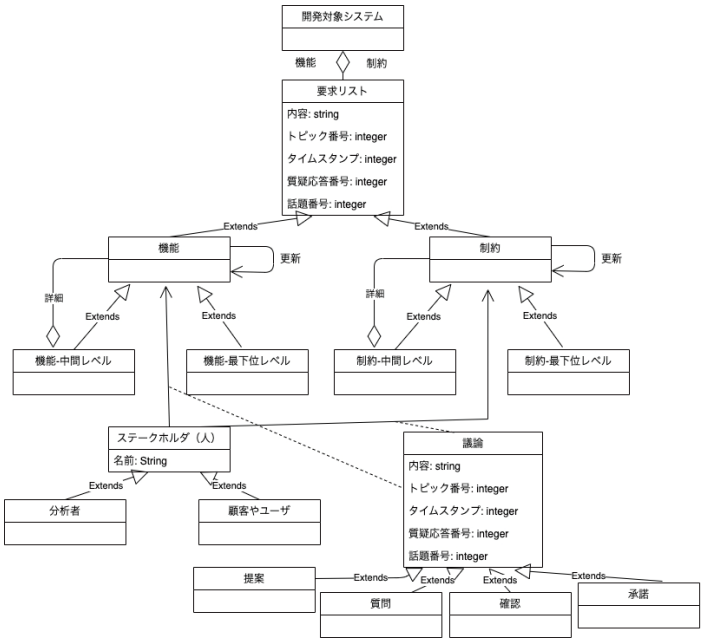


図 1: 質疑応答モデル

これは加藤らの研究 [7] での質疑応答モデルを分析し作成した．この例は図書館

の業務を支援する情報システムで，要求分析者が図書館の利用者（以下，利用者），図書館員と質疑応答を行い，初期要求リストを詳細化し，更新していくプロセスである．質疑応答モデルのクラスをノード，関連をエッジ，属性をプロパティと対応づけてグラフデータベースを設計する．開発対象システムは機能と制約に分けられる．図書館員や利用者，分析者などのステークホルダと質疑応答の対象となる要求リストは，図 1 ではクラスで表され，両者の関連には議論の内容を表す属性（リンク属性）が付与されている．議論はさらに確認，承諾，提案，質問の 4 種類に分けられており，これらがステークホルダの活動を表している．グラフモデルではこれらはエッジで表現される．要求リスト（機能，制約）は階層構造をしている．機能，制約が質疑応答によって更新されていくため，それらの間には更新という関連がある．

トピック番号とはシステムの機能や制約の識別番号である．議論中の質疑応答については，質問や応答ごとに，議論開始から時間経過にしたがって通し番号を振りこれを質疑応答番号とする．例えば，質疑応答番号 n は，最初から n 番目の質問もしくは応答である．話題番号は質問や応答を機能，制約ごとにまとめたときの識別番号である．例えば，ある機能 A に話題番号 m が振られている場合，A についての質問もしくは応答で m 番目のものを表す．質疑応答番号と話題番号を分けて保持することで質疑応答の後戻りを把握しやすくする役割がある．

3　グラフデータベースの実現

3.1　適用事例

2 章で述べたグラフモデル定義に図書館のシステムを適用させる．本論文では図書館のシステムの質疑応答履歴の中の「本の貸し出しと返却」の議論をグラフデータベースで構造化した．実際に使用した質疑応答の例を図 2 に示す．図 2 のメールは要求リストの内容の確認メールである．これをグラフモデルに適用させると図 3 となる．

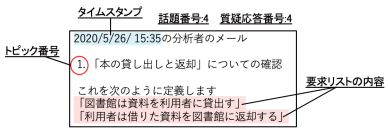


図 2: 質疑応答の例

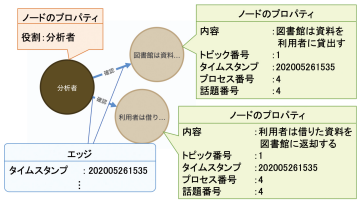


図 3: グラフモデルの例

分析者から「これを次のように定義します」とあり，このメールの内容に対して図書館員や利用者は承諾や提案などのリアクションを返す．要求リストの内容である「利用者は借りた資料を図書館に返却する」と「図書館は資料を利用者に貸出す」がそれぞれノードとなる．そのため，分析者から確認のエッジで要求リストの内容のノードと結びつく．このときエッジのプロパティとしてタイムスタンプを保持している．ノードのプロパティとして，それぞれのノードのプロパティに内容とトピック番号の「1」，年・月・日・時刻の 12 桁で表現されたタイムスタンプの「202005261535」，プロセス番号と話題番号である「4」を持つ．

3.2　本の貸し出しと返却の例

3.1 章と同様にトピック番号 1 の本の貸し出しと返却をグラフデータベースに格納すると図 4 となる．ステークホルダには分析者，図書館員，利用者の 3 人が存在する．

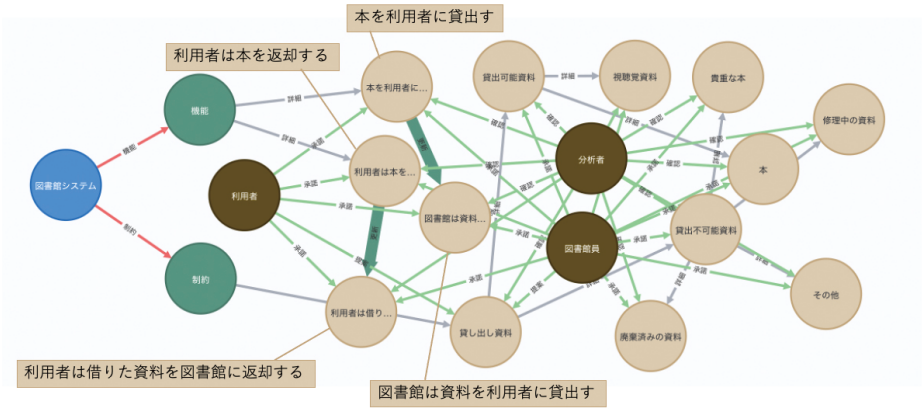


図 4: 本の貸し出しと返却のグラフモデル

ステークホルダである分析者，図書館員，利用者を濃い色で表している．ノード数は 19，エッジ数は 46 となった．Neo4j のグラフ表示の特徴として他のノードに結びつきの強いノードが中心にくる．図 4 の特徴からステークホルダのうち分析者と図書館員の 2 つのノードはグラフの中心にあることが確認できる．利用者は 2 つのノードと比較した際に中心にはないことがわかる．ここから分析者と図書館員の 2 人は同等の発言量であるのに対し，利用者は発言量が少ないことがわかる．

発言量が極端に少ないステークホルダの存在は要求の聞き取り不足を示唆し改善することで要求リストの出戻りの減少に期待できる．

3.3 グラフ検索事例

Cypher で検索文を記述することでさまざまなサブグラフを抽出できる．本論文では未承諾事項・承諾事項のサブグラフと議論の進捗状況を追跡するための時系列でのグラフの変化を紹介する．

3.3.1 未承諾事項の発見

分析者からの確認事項に対して図書館員，利用者の両者の承諾があるか否かを確認する．検索クエリのパターンは以下に記す．

```
match (e{名前:"分析者"})-[:確認]->(f),(g {名前:"ステークホルダ"})
where NOT (g)-[:承諾]-(f)
return f ;
```

この検索クエリでは 1 行目の match 文で分析者のノードから確認のエッジが結ばれている要求リストとステークホルダを取り出す．そこから，ステークホルダの承諾が得られていないグラフを出力すると図 4 に対し，図 5 が得られる．

左側の利用者の図は検索クエリの 1 行目を (g 名前:”利用者”)，右側の図書館員の図は (g 名前:”図書館員”) として出力したもので，利用者からの承諾，図書館員からの承諾が各々得られていないものである．図書館員は 1 つの機能の承諾をまだ行っていないのに対し，利用者は承諾していない機能が 9 つある．特に両者に共通の「図書資料の貸し出し」については議論を行っていかなければならないことがわかる．

この検索クエリによってそれぞれのステークホルダの未承諾事項の検出が可能になる．また，議論途中の事項の確認や途中で議論が終了している事項の発見につながり，議論が承諾されるまで行うことで議論の活性化に期待ができる．

3.3.2 承諾事項の確認

分析者からの確認に対して図書館員と利用者の両者から承諾がされたものを要求

(a) (g 名前:”利用者”) (b) (g 名前:”図書館員”)

図 5: 未承諾事項の発見

リストの承諾事項として検索クエリを記述し，承諾の有無を見つける．

```
match (n)-[i:承諾]->(k)
match (m)-[j:承諾]->(k)
where n.名前="図書館員" and m.名前="利用者"
RETURN n,m,i,j,k
```

この検索クエリでは match 文で承諾のエッジをはっているノードを取り出し，そこから図書館員と利用者の両者からの承諾があるノードに絞り込み出力すると図 4 に対し，絞り込みをかけると図 6 が得られる．

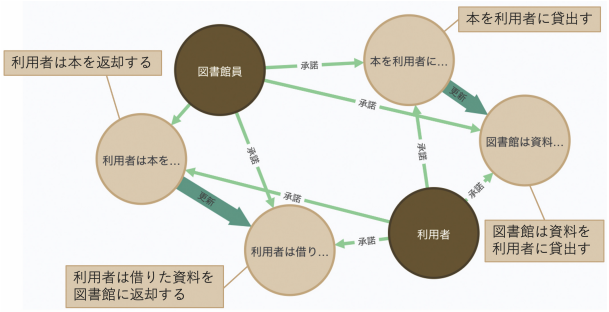


図 6: 承諾事項のサブグラフ

細い矢印が承諾の矢印，太い矢印が要求リストの更新の矢印である．本の貸し出しと返却モデルのうち承諾事項は 4 つあることが確認できる．このサブグラフから両者から承諾を得た後に要求リストの更新が行われていることが確認できる．

このサブグラフでは「利用者は本を返却する」に「利用者は借りた資料を図書館に返却する」が追加され，「本を利用者に貸出す」に「図書館は資料を利用者に貸出す」が追加され，要求リストが更新されている．更新前だけでなく更新後もしっかりと承諾が行われていることが確認できる．

3.3.3 議論の進捗状況の追跡

議論の進捗状況を時刻の変化から調べる．これはステークホルダの活動からタイムスタンプで絞り込みをかけたものである．検索クエリ

```
match (n)-[k]-(j)
where k.タイムスタンプ <= ’12 桁の数字’
return n,k,j
```

で絞り込みをかけると図 7 が得られる．

タイムスタンプが’202005251345’ と’202005261535’ の変化を表している．タイムスタンプが’202005251345’ と’202005261535’ のグラフを比較すると分析者の発言に

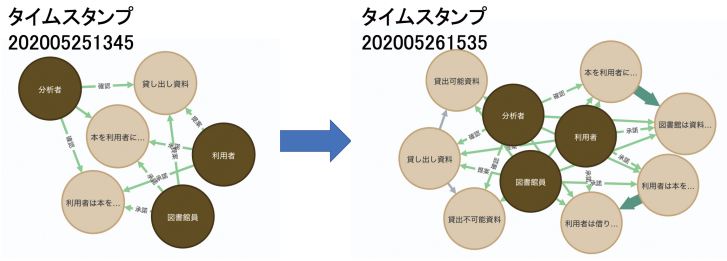


図 7: 時系列でのグラフの変化

より分析者から確認のエッジが増えていることが確認できる．このように一定期間でのグラフを絞ることで議論の進捗状況を追うことが可能になる．今回は指定した議論とその 1 つ前で進捗がどれくらいあるかを確認するためにタイムスタンプの数字を 12 桁細かく指定したが，ある日付までの進捗を確認したい際は時刻である後ろ 4 桁を 0 にすることで進捗状況の追跡が可能になる．

ノードやエッジが増えているときには議論が活発に行われている．承諾のエッジが増えてくると議論が収束に向かっているなどノードやエッジの数，内容を確認することで議論の進捗状況の追跡を行うことが可能になると考える．

4　まとめと今後の課題

本論文ではシステム全体を作成する際の質疑応答履歴をグラフデータベースで構築することで要求獲得支援のアプローチを考えた．Neo4j を用いてグラフデータベースを構築し実際の質疑応答履歴に適用させた．グラフモデルの作成により質疑応答での発言量の差，未承諾事項の発見，議論の進捗状況の追跡などが可能となった．

今後はシステム全体でのグラフデータベースの構築を行い，検索クエリで表現できることをさらに考え要求獲得支援に向けて取り組んでいく．また，評価方法についても考えていく．

Kato らの研究 [8] の質問の出し方を誘導する手法と連携すると，一層の効果が期待できる．今回は，メールでの質疑応答であったが，グラフデータベースへの入力は人手であり，大きな労力となった．自然言語処理技術などを用いて省力化することも考えられる．角らの研究 [9] は要求抽出会議における会話プロセスの特徴をとらえようとしており，その成果をもとに，図 1 の質疑応答履歴のグラフモデルを洗練することも今後の課題である．

参考文献

[1] ユーザのための要件定義ガイド 第 2 版 要件定義を成功に導く 128 の勘どころ，IPA，2019.
[2] ユーザー企業 ソフトウェアメトリックス調査 【システム開発・保守調査報告書】 2020 年版，一般社団法人　日本情報システム・ユーザー協会，2020.4.
[3] Colin Potts, et al., Inquiry-Based Requirements Analysis, IEEE Software, 11(2), pp.21-31, 1994.
[4] Motoshi Saeki, et al., Structuring utterance records of requirements elicitation meetings based on speech act theory, ICRE 1996, pp.21-30, 1996.
[5] Neo4j, https://neo4j.com/.
[6] I. Robinson, et al., Graph Databases: New Opportunities for Connected Data, 2nd ed.,O'Reilly, 2015 [木下 哲也 (監訳)，グラフデータベース，オライ リー・ジャパン, 2015].
[7] 加藤 潤三 他，要求獲得のためのシソーラスの統合，情報処理学会研究報告ソフトウェア工学 (SE)，2021-SE-208-4 8 ページ，2021.
[8] Junzo Kato, et al., A Model for Navigating Interview Processes in Requirements Elicitation., APSEC 2001, pp.141-148, 2001.
[9] 角 秀樹 他，ソフトウェア要求抽出における実験的会話分析，FOSE2005，pp.145-154，2005.

チェックリストを用いた設計書レビュー支援のための判定ルール自動生成

Automatic Generation of Rules to Support Design Document Review using Checklist

大林 浩気 * 河合 克己 † 前岡 淳 ‡

あらまし チェックリストを用いた設計書レビューの効率向上を目的とし，チェック項目毎に事前定義した判定ルールによりチェック項目に関する設計書中の該当記載箇所を特定する設計書レビュー支援ツールを提案する．さらに，過去プロジェクトの既存ドキュメントを学習することにより記載箇所特定精度の良い判定ルールを自動生成する手法を提案する．実プロジェクトを対象とした評価実験の結果，自動生成した判定ルールによって実用的な精度でチェック項目に対する設計書内の該当記載箇所を特定できること，自動生成した判定ルールを学習したプロジェクト以外のプロジェクトにも適用できることを確認した．

1 はじめに

システム開発において作成する設計書の品質担保のために，設計書が満たすべき条件をチェック項目にした**チェックリスト**を用いてレビューすることが効果的である [1] [2] [3]．図 1 にチェックリストの一例を示す．しかし，設計書が各チェック項目の条件を満たしているか否かを判定するためには，各チェック項目に関する設計書中の該当記載箇所を探して内容を確認する必要があるため，対象の設計書やチェック項目が多い場合にレビュアの作業負荷が大きいという課題がある．そこで筆者らは先行研究 [4] において，チェックリストを用いた設計書レビューの効率向上を目的とし，チェック項目毎に事前定義した特徴語に基づく判定ルールによりチェック項目に関する設計書中の該当記載箇所を特定し提示する，**設計書レビュー支援ツール**を提案した．本稿では，設計書レビュー支援ツールの現場導入時のコストを低減するため，過去プロジェクトの既存ドキュメントを学習することにより判定ルールの作成を自動化する手法を提案する．

本稿の構成は以下のとおりである．第 2 章では本稿の前提となる設計書レビュー支援ツールについて述べ，第 3 章で判定ルール作成の自動化に関する提案手法について述べる．第 4 章で評価実験について述べ，第 5 章で実験結果と効果について考察する．第 6 章で全体をまとめる．

2 設計書レビュー支援ツール

2.1 基本方式

本稿の前提となる設計書レビュー支援ツールについて説明する．設計書レビュー支援ツールの基本的な方式は，大きく (1) 判定ルールの作成，(2) 設計書の分割，(3) チェック項目に関する該当記載箇所の特定，(4) マトリクス形式による結果の提示，の 4 つのステップからなる (図 2)．以下，それぞれの内容について説明する．

(1) **判定ルールの作成**
チェックリストの各チェック項目について，該当記載箇所を特定するための判定ルールを定義する．判定ルールは，該当記載箇所が含んでいる可能性が高い単語である**特徴語**とその**重要度**，該当記載箇所が必ず含んでいるべき単語であ

*Hiroki Ohbayashi, (株) 日立製作所
†Katsumi Kawai, (株) 日立製作所
‡Jun Maeoka, (株) 日立製作所

る**必須語**からなる．特徴語は，該当記載箇所の章節タイトルが含んでいる可能性が高い**タイトル特徴語**と該当記載箇所の本文が含んでいる可能性が高い**本文特徴語**の 2 種類に細分される．判定ルールの一例を図 3 に示す．

(2) **設計書の分割**
レビュー対象設計書の文書構造を解析し，章節単位に分割する．

(3) **チェック項目に関する該当記載箇所の特定**
(2) で作成した設計書のそれぞれの分割単位に対し，(1) で作成した判定ルールを適用することで，各チェック項目に関する設計書内の該当記載箇所を特定する．具体的には，以下の手順で行う．

(i) 分割単位内のテキストが判定ルールに定義された必須語を含むかどうかを判定する．必須語を含む場合，手順 (ii) に進む．必須語を含まない場合，分割単位はチェック項目に関する該当記載箇所ではないと判定する．

(ii) 分割単位内のテキストと判定ルールに定義された特徴語を突合し，チェック項目 c に関する分割単位 d の該当スコアを算出する．該当スコアは，タイトル特徴語と分割単位の章節タイトルのマッチ度合いを示す**タイトル該当スコア** $\mathrm{Score}_{\mathrm{title}}(c,d)$ と，本文特徴語と分割単位の本文のマッチ度合いを示す**本文該当スコア** $\mathrm{Score}_{\mathrm{content}}(c,d)$ の 2 つを算出する．これら該当スコアの算出方法については 2.2 節で詳説する．

(iii) タイトル該当スコアがあらかじめ設定した閾値 α(例えば $\alpha = 1$) よりも高い場合，分割単位はチェック項目に関して「タイトルに記載あり」と判定する．同様に，本文該当スコアが閾値 α よりも高い場合,「本文に記載あり」と判定する．最後に,「タイトルに記載あり」または「本文に記載あり」と判定された場合，分割単位はチェック項目に関する該当記載箇所であると判定する．

(4) **マトリクス形式による結果の提示**
チェック項目に関する該当記載箇所の特定結果を，行方向を設計書の分割単位，列方向をチェックリストのチェック項目とするマトリクス形式の表で提示する (図 4)．分割単位 d がチェック項目 c に関する該当記載箇所である場合，マトリクスの (d,c) に対応する位置に○，△，▲のいずれかの記号を出力する．記号の凡例は以下のとおりである．

○ タイトルと本文両方に記載あり
△ タイトルのみに記載あり
▲ 本文のみに記載あり

NO	チェック項目	合否判定結果	該当記載箇所
1	システムの目的が明記されているか	OK	1.1 システム概要
2	前提条件としてハードウェア構成が明記されているか	NG	
3	機能一覧が明記されているか	OK	2.1.1 機能要件
4	入出力項目仕様が明記されているか	OK	3.1.2 入出力
…	…		

図 1　チェックリストの例

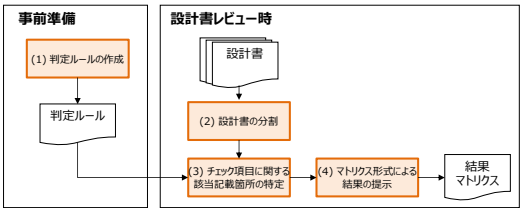


図 2　設計書レビュー支援ツールの処理フロー

NO	チェック項目	必須語	タイトル		本文	
			特徴語	重要度	特徴語	重要度
1	システムの目的が明記されているか	システム	目的	0.9	目的	0.7
2	前提条件としてハードウェア構成が明記されているか	構成	前提	0.6	ネットワーク	0.3
			ハードウェア	0.8	ストレージ	0.2
3	機能一覧が明記されているか	機能	一覧	0.9	一覧	0.8
			表	0.6	示す	
…	…					

図 3　判定ルールの例

		アラート	設計書の分割				
			1.1 システム概要	1.2 基本方針	2.1.1 機能要件	2.1.2 処理方式	…
チェック項目	システムの目的が明記されているか		○	▲			
	前提条件としてハードウェア構成が明記されているか	記載なし					
	機能一覧が明記されているか				○		
	入出力項目仕様が明記されているか					△	
	…						

図 4　マトリクス形式による結果出力例

また，各チェック項目について，該当記載箇所がひとつもない場合，チェック項目に対して「記載なし」のアラートを出力する．

2.2 特徴語の重要度を考慮した該当スコアの算出

前節のステップ (3)(ii) における該当スコアの算出方法について説明する．

例えば,「スループットの目標値を明確にすること」というチェック項目に対して,「性能」と「スループット」のどちらも特徴語として考え得るが,「スループット」の方が当該チェック項目をより強く特徴付ける単語であり，重要であると考えられる．そこで，このようなチェック項目に対するそれぞれの特徴語の重要度を踏まえ，重要度の高い特徴語を含むほど該当スコアが高くなるようにするため，タイトル該当スコア $\mathrm{Score}_{\mathrm{title}}(c,d)$ を以下の式で定義する．

$$\mathrm{Score}_{\mathrm{title}}(c,d) = \sum_{w \in W_{\mathrm{title}}(c) \cap d_{\mathrm{title}}} \mathrm{weight}_{\mathrm{title}}(c,w)$$

ただし，c はチェック項目，d は設計書の分割単位，d_{title} は d のタイトルテキストに含まれる単語の集合，$W_{\mathrm{title}}(c)$ はチェック項目 c に関するタイトル特徴語の集合である．$\mathrm{weight}_{\mathrm{title}}(c,w)$ はチェック項目 c に対するタイトル特徴語 w の重要度であり，重要度が高いほど高い値になるように設定する必要がある．そこで，過去プロジェクトの既存ドキュメントをもとに作成した設計書分割単位とチェック項目の対応関係を教師データとして SVM(Support Vector Machine [5]) で学習することにより学習該当記載箇所の判定モデルをチェック項目ごとに作成し，作成したモデルのパラメータを利用して特徴語の重要度を推定する．具体的な手順は以下の通りである．

(1) チェック項目 c を選ぶ．

(2) 全ての設計書分割単位 d について，以下の方法でタイトル特徴語の数 $|W_{\mathrm{title}}(c)|$ と同じ次元のベクトル v_d にエンコードする．具体的には，v_d の i 番目の要素 $v_d(i)$ を以下で定める．

$$v_d(i) = \begin{cases} 1 & d \text{ のタイトルテキストが i 番目の特徴語 } w_i \text{ を含む} \\ 0 & d \text{ のタイトルテキストが i 番目の特徴語 } w_i \text{ を含まない} \end{cases}$$

(3) 全ての設計書分割単位 d について，d がチェック項目 c の該当記載箇所である場合 1，そうでない場合 0 としてラベル付けする．

(4) エンコードしたベクトルとラベルを教師データとして SVM で学習すると以下のような識別関数が得られる．

$$f(x) = \sum_{i=0}^{|W_{\mathrm{title}}(c)|} a_i x_i + b$$

これは，設計書分割単位 d をエンコードしたベクトルを入力した場合に値が正であればチェック項目 c の該当記載箇所と判定する関数になっている．

(5) バイアス項 b は判定の閾値，係数 a_i は i 番目の特徴語 w_i の重要度と考えることができる．そこで，閾値が 1 となるように正規化し，$\mathrm{weight}_{\mathrm{title}}(c,w_i)$ を以下で定める．

$$\mathrm{weight}_{\mathrm{title}}(c,w_i) = a_i/|b|$$

本文該当スコア $\mathrm{Score}_{\mathrm{content}}(c,d)$ についても，本文特徴語の集合 $W_{\mathrm{content}}(c)$，本文特徴語の重要度 $\mathrm{weight}_{\mathrm{content}}(c,w)$ を用いて同様に算出式を定義する．

3 判定ルール作成の自動化

判定ルールを手作業で作成することも可能であるが，該当記載箇所特定精度の良い判定ルールを作成するには大きな工数がかかる．また，チェックリストはチェック項目の追加や内容の変更が頻繁に行われるため，手作業では判定ルールのメンテナンスが困難である．したがって，該当記載箇所特定精度の良い判定ルールを少ない工数で作成する方法の確立が課題となる．この課題の解決のため，過去プロジェクトの既存ドキュメントを基に必須語と特徴語を自動で選定することで精度の良い判定ルールの作成を自動化する手法を提案する．提案手法は大きく (1) 教師データの自動生成，(2) 特徴語の自動選定，(3) 必須語の自動選定の 3 つのステップからなる

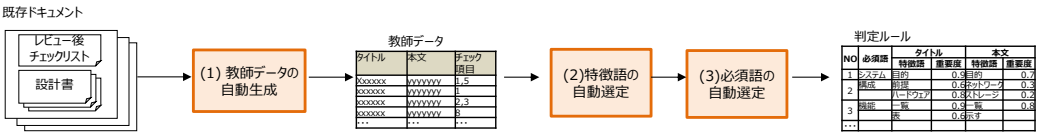


図 5　判定ルール作成自動化手法の流れ

(図 5)．以下，それぞれのステップの詳細について述べる．

3.1　教師データの自動生成

過去プロジェクトの設計書レビュー後のチェックリストには，各チェック項目についてレビュアが OK と判断した根拠となる該当記載箇所の情報が記載されている．これを解析することにより，特徴語を自動選定する際の学習に必要な教師データを自動生成する．以下に具体的な処理の流れを示す．

(1) 設計書レビュー後のチェックリストを解析し，各チェック項目 c に対する該当記載箇所の情報 (設計書名，章節番号，章節タイトル) を抽出する．
(2) レビュー対象となっていたすべての設計書を解析し，章節単位 d に分割する．
(3) (1) で抽出したチェック項目 c に対する該当記載箇所の情報と (2) で分割した設計書の分割単位 d の設計書名，章節番号，章節タイトルを照合し，チェック項目と設計書内の該当記載箇所の対応関係を特定する．
(4) 特定したチェック項目と設計書内の該当記載箇所の対応関係をもとに，学習用の教師データ (設計書の全分割単位についての，章節タイトルテキスト，本文テキスト，対応するチェック項目の情報) を作成する．

3.2　特徴語の自動選定

3.1 節で自動生成した教師データを SVM で学習することにより設計書に含まれる全ての語彙の各チェック項目に対する重要度を推定し，重要度の高い単語を選択することにより，チェック項目ごとの特徴語を自動選定する．単語の重要度の推定の基本的な手順は 2.2 節で説明した方法と同様であるが，以下の 2 点が大きく異なる．

- 設計書の分割単位をベクトルにエンコードする際に，設計書のテキストに含まれる語彙全てを対象にする．これにより，設計書のテキストに含まれる語彙全てについて重要度を推定することができる．
- SVM の学習時に L1 正則化を適用する．L1 正則化は学習後のモデルのパラメータを疎にする効果がある．これにより該当記載箇所の判定にあまり寄与しない単語について重要度が 0 と推定されるようになり，より本質的な特徴語のみを絞り込むことができる．

以下にタイトル特徴語を自動選定する具体的な手順を示す．

(1) チェック項目 c を選ぶ．
(2) 設計書のテキストに含まれる語彙全てを抽出し，その集合を W とする．
(3) 全ての設計書分割単位 d について，以下の方法で設計書のテキストに含まれる語彙の数 $|W|$ と同じ次元のベクトル v_d にエンコードする．具体的には，v_d の i 番目の要素 $v_d(i)$ を以下で定める．

$$v_d(i) = \begin{cases} 1 & d \text{ のタイトルテキストが i 番目の単語 } w_i \in W \text{ を含む} \\ 0 & d \text{ のタイトルテキストが i 番目の単語 } w_i \in W \text{ を含まない} \end{cases}$$

(4) 全ての設計書分割単位 d について，d がチェック項目 c の該当記載箇所である場合 1，そうでない場合 0 としてラベル付けする．
(5) エンコードしたベクトルとラベルを教師データとして L1 正則化を適用した SVM で学習すると以下のような識別関数が得られる．

$$f(x) = \sum_{i=0}^{|W|} a_i x_i + b$$

(6) a_i が 0 より大きい単語 w_i を選出し，それらをチェック項目 c に対するタイトル特徴語とする．すなわち，$W_{\text{title}}(c) = \{w_i \in W | a_i > 0\}$ とする．

本文特徴語 $W_{\text{content}}(c)$ についても，(3) のベクトルにエンコードする際の判定対象のテキストを d の本文テキストとして，同様に選定する．

3.3 必須語の自動選定

3.1 節で自動生成した教師データから得られる，各チェック項目 c に該当する設計書分割単位の集合を $D(c) = \{d_1, d_2, \cdots, d_n\}$ とする．c の特徴語のうち $D(c)$ の設計書分割すべてのテキストに含まれる単語を c の必須語として選定する．すなわち，c の必須語の集合 $W_{\text{require}}(c)$ を以下により決定する．

$W_{\text{require}}(c) = \{w \in W_{\text{title}}(c) \cup W_{\text{content}}(c) | \forall d \in D(c), d \text{ のテキストが } w \text{ を含む}\}$

4 評価実験

提案手法の有効性を評価するため，過去プロジェクト 5 案件のデータを対象として実験を行った．これら 5 案件はすべて金融分野のシステム開発プロジェクトであるが，プロジェクトのメンバやシステムの業務内容は 5 案件でそれぞれ異なる．評価観点は以下の通りである．

(a) 自動生成した判定ルールによる記載箇所特定が実用的な精度か
(b) 自動生成した判定ルールに汎用性があるか（あるプロジェクトのデータをもとに自動生成した判定ルールを他プロジェクトに適用した場合に精度が出るか）
(c) 自動生成のもとになる教師データのプロジェクト数を増やした場合に，精度が向上するか

実験の流れを以下に示す．

(1) 過去プロジェクト 5 案件のデータを，学習データ (1〜4 案件)，テストデータ (1 案件) に分割する．
(2) 学習データから 3 章に説明した方式で判定ルールを自動生成する．

表 1 学習データを 1 案件，テストデータを 1 案件とした場合の F 値の計測結果

学習データ	テストデータ	F 値
PJ1	PJ2	78%
PJ1	PJ3	75%
PJ1	PJ4	85%
PJ1	PJ5	64%
PJ2	PJ1	70%
PJ2	PJ3	75%
PJ2	PJ4	90%
PJ2	PJ5	76%
PJ3	PJ1	58%
PJ3	PJ2	63%
PJ3	PJ4	92%
PJ3	PJ5	71%
PJ4	PJ1	58%
PJ4	PJ2	45%
PJ4	PJ3	76%
PJ4	PJ5	71%
PJ5	PJ1	65%
PJ5	PJ2	73%
PJ5	PJ3	73%
PJ5	PJ4	88%
平均		72%

表 2 学習データを 4 案件，テストデータを 1 案件とした場合の F 値の計測結果

学習用データ	テストデータ	F 値
PJ2,PJ3,PJ4,PJ5	PJ1	69%
PJ1,PJ3,PJ4,PJ5	PJ2	76%
PJ1,PJ2,PJ4,PJ5	PJ3	80%
PJ1,PJ2,PJ3,PJ5	PJ4	86%
PJ1,PJ2,PJ3,PJ4	PJ5	71%
平均		76%

表 3 プロジェクトの設計書数と分割単位数

プロジェクト	設計書数	分割単位数
PJ1	12	80
PJ2	44	348
PJ3	30	57
PJ4	9	73
PJ5	19	108

(3) 自動生成した判定ルールをテストデータに適用して 2 章に説明した方法で記載箇所特定を行い，精度を計測する．
(4) (1)〜(3) を学習データとテストデータの分割を変えて複数パターンで実施する．

実験で対象としたチェックリストのチェック項目の数は 44 項目である．過去プロジェクト 5 案件それぞれの設計書の数は表 3 の通りである．

実験結果として，学習データを 1 案件，テストデータを 1 案件とした場合の F 値の計測結果を表 1，学習用データを 4 案件，テスト用データを 1 案件とした場合の F 値の計測結果を表 2 に示す．

5 考察

実験結果に基づき，それぞれの評価観点について考察する．

(a) **自動生成した判定ルールによる記載箇所特定が実用的な精度か**
開発現場との議論により，ツールが提示する記載箇所の適合率，再現率が共に 70 %程度あれば，ツール適用によりチェック項目の該当記載箇所特定作業を効率化できるという結論になった．そこで，本ツールの精度の目標値を平均 F 値 70 %と定めた．表 2 より，4 案件のデータを学習データとした場合に，F 値が平均 76 %，最低でも 69 %であった．上記は目標値を達成しており，実用に十分な精度であると考える．

(b) **自動生成した判定ルールに汎用性があるか**
学習データ以外の案件をテストデータとして，上で述べた通りに実用的な精度が出ていることから，自動生成した判定ルールに汎用性があると結論付けられる．

(c) **自動生成のもとになる教師データのプロジェクト数を増やした場合に，精度が向上するか**
表 1 と表 2 より，学習用データを 1 案件とした場合の F 値が平均 72 %，最低 45 %であり，学習用データを 4 案件とした場合の F 値が平均 76 %，最低 69 %であった．このことから，教師データのプロジェクト数を増やすことにより，精度が向上すると考える．

6 まとめ

本稿では，チェックリストを用いた設計書レビューの効率向上を目的とし，チェック項目毎に事前定義した判定ルールによりチェック項目に関する設計書中の該当記載箇所を特定する設計書レビュー支援ツールを提案した．さらに，過去プロジェクトの既存ドキュメントを学習して精度の良い判定ルールの作成を自動生成する手法を提案した．実プロジェクトを対象とした評価実験の結果，自動生成した判定ルールによって実用的な精度でチェック項目に対する設計書内の該当記載箇所を特定できること，自動生成した判定ルールに汎用性があることを確認した．今後の課題として，該当記載箇所の特定のみでなく，記載内容がチェック項目の内容を満たしているかどうか等の妥当性の確認を支援する方法を検討する．

参考文献

[1] G. Sabaliauskaite, F. Matsukawa, S. Kusumoto and K. Inoue: An experimental comparison of checklist-based reading and perspective-based reading for UML design document inspection, *Proceedings International Symposium on Empirical Software Engineering*, 2002, pp. 148-157.
[2] T. Thelin, P. Runeson and C. Wohlin: An experimental comparison of usage-based and checklist-based reading, *IEEE Transactions on Software Engineering*, vol. 29, no. 8, pp. 687-704, Aug. 2003.
[3] 情報処理推進機構（IPA）: 続 定量的品質予測のススメ〜IT システム開発における定量的品質管理の導入ノウハウと上流工程へのアプローチ〜, 2011.
[4] 大林 浩気，前岡 淳，河合 克己，緒方 孝一郎，鈴木 一平，三島 典子，手塚 聡子，嶋村 伸子: 設計書レビュー支援のためのチェック項目該当記載箇所の特定方法, 情報処理学会全国大会, 2022.
[5] 竹内 一郎，烏山 昌幸: サポートベクトルマシン, 講談社, 2015.

エポックワードと名詞の重要度を用いたソフトウェア仕様書からのゴール文の抽出

Extracting goal sentences from speculations using epoch words and importance of nouns

渡辺 啓太郎 * 中川 博之 † 土屋 達弘 ‡

あらまし ソフトウェアの多様化，大規模化に伴い，ソフトウェアに対する要求を人力で抽出・分析することはますます難しくなっている．そこで，本研究では要求分析モデルの 1 つであるゴール指向要求モデルに着目し，全自動でゴールの抽出を行う手法の構築を目的とする．その初期研究として，本論文では仕様書からゴールを含む文を抽出する手法を提案する．この手法は仕様書からエポックワードと呼ばれる特定語，加えて仕様書に出現する名詞の重要度を基準にソフトウェアへの要求を含む文の特定・抽出を行うものである．本論文ではこの手法を実装し，その正確性を評価する．

1 はじめに

ソフトウェアの開発規模は拡大の一途を辿っており，ソフトウェアの満たすべき要求も年々複雑化している．これに伴い，ソフトウェアの仕様の過不足ない把握，管理は徐々に難しくなっている．これを克服する一つの要求モデルがゴール指向要求モデルである．これまでに提唱されたゴールモデルとしては，KAOS [5]，i* [6]，NFR [7] などが存在する．本モデルは，仕様書に記された要求を対象のソフトウェアが満たすべき目標，即ちゴールとして定義し，各ゴールを形式化 [3]，可視化 [1] することで開発ソフトの要求の管理を容易にする．先行研究としては，Arora らによる仕様書からのドメインモデル抽出 [4] や，Pimetel らによるゴールモデル構築を補助する Web ツール [11]，Woldeamlak らによる KAOS の利用 [12]，Nguyen らによるゴールとユースケースの自動抽出 [13]，Horkoff ら [14] や Zee ら [15] によるゴールモデルによる分析が存在する．また，本研究は Shimada [2] によるゴールモデルの構築の自動化から着想を得ている．

本論文では仕様書の文章から要求，即ちゴールが含まれる文（センテンス）を自動的に抽出する手法を提案する．特に，仕様書内に出現頻度の高い単語や，要求に含まれやすいキーワードを目印とするヒューリスティックな手法の実装を行う．また，エポックワードと呼ばれる語句を手掛かりとした抽出を実装する．エポックワードは，要求に関する記述が続く可能性が高い語句である．このエポックワードを目印にゴール文の判定を行うことで抽出の精度を上げる方法を提案し，効果を検証する．

2 ゴール文の抽出

ソフトウェアに対する要求を可視化し，把握を容易にすることを目的とした**ゴール指向要求分析モデル**において，要求はゴール（目標）に対応する．本論文で提案する手法は，仕様書の中からゴールを含む文を抽出することを目的とする．以降，ゴールを含む文をゴール文と表現する．本論文のゴール文抽出手法では，キーワードを用いた簡易な判定（判定 1），名詞の重要度（判定 2）とエポックワードによる判定（判定 3）を扱う．判定 1，2，3 の順に詳細を述べる．

判定 1: ゴール文は要求を定義する以上,「〜べきである」「〜でなければならな

*Keitaro Watanabe, 大阪大学 大学院情報科学研究科, k-watanabe@ist.osaka-u.ac.jp

†Hiroyuki Nakagawa, 大阪大学 大学院情報科学研究科, nakagawa@ist.osaka-u.ac.jp

‡Tatsuhiro Tsuchiya, 大阪大学 大学院情報科学研究科, t-tutiya@ist.osaka-u.ac.jp

Some of the complaints about the current system as reported by university authorities, library staff, department members or students include the following: ... The new UWON* library system should address such problems through software-based solution integrating all department libraries.

大学当局、図書館職員、学科員、あるいは学生から、現行システムに対する次のような苦情が寄せられている。（中略）
UWONの新しい図書管理システムは、すべての学部図書館を統合的に管理するソフトウェアベースのソリューションを通じて、このような問題に対処する必要がある。

（下線部はエポックワード）

図 1　エポックワードの具体例

い」等の表現が含まれ得る．このため，判定 1 では「should や must などの “キーワード”を含む文はゴール文である」という判定方法を採用する．

判定 2: この判定は，ゴール文には仕様書において出現頻度の高い単語が用いられる，というヒューリスティックに基づく．仕様書の対象であるソフトウェアが事物 A に関するものであった場合，仕様書の中で頻出する単語は A，または A に関連する単語と見られる．よって，判定 2 では仕様書内で出現回数の高い単語を含む文をゴール文として抽出する．

判定 3: この判定では，ある段落の中で，条件を満たす語句，エポックワードが出現した位置以下の文をゴール文と判定する．エポックワードの条件を次に示す．

- 名詞句/節である．
- new，envisioned，to be designed などの，仕様書の対象となるソフトウェアの修飾語を含んでいる．
- 仕様書全体で出現頻度が高い語，または system，software が用いられている．

これらを満たす語は，例を挙げると,「新しいソフトウェア」「作られる予定のシステム」などの語である．図 1 は用いる仕様書 [10] の文章を元にエポックワードの例を示したものである．仕様書中，単に「ソフトウェア」「システム」などと記載される場合は，仕様書の対象でない，別のシステムや旧システムに関する言及である可能性がある．一方,「新しいシステム」「予定される（出現頻度の高い語彙）」と明確に述べた語句は仕様書で扱われるソフトウェアを指す可能性が高く，同時にこのような語句を含む文章は要求について触れたゴール文である可能性が高い．よって，このような語句をエポックワードと呼び，ゴール文抽出の手掛かりに用いる．

3　実装

本研究では，2 節に述べた手法を Python で実装する．実装した手法の，図 2 に示す各処理を次に述べる．

仕様書の解析：以下に述べる処理全てを統括する処理である．まず指定された仕様書のファイルを開き，句点などにより文章を文に分割する．その後，図 2 の通りに処理を行う．処理後，結果を表示し，ログを出力する．

(1) 出現した名詞の抽出：仕様書内の名詞の出現頻度を知るため，仕様書内である名詞が出現した回数を記録する．これを行うためには，文中のある単語が名詞か否かを判定する処理が必要となる．このため，この処理では Python の自然言語処

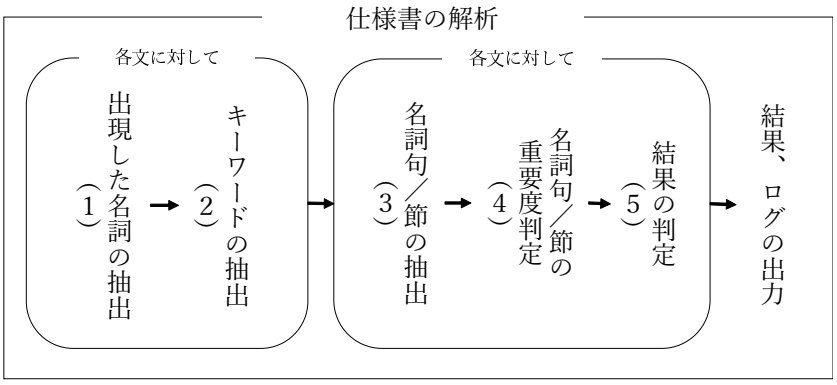


図 2　ゴール文抽出手法の処理フロー

理用パッケージ “Stanza” [8] を使用する．Stanza は，英文中の一つ一つの単語の品詞（名詞，形容詞など）を特定できる．例えば，仕様書中に “system” という単語が存在する場合，Stanza を使用すれば system を名詞として検出し，何回使用されたかを数えられる．留意すべきこととして，形が異なる名詞は同じ名詞と捉え，まとめて数える．例えば，systems は単数形である system と同じ単語と見なす．これを実現するため，本処理では取得した名詞のレマタイズを行う．レマタイズとは，名詞や動詞を原型に戻す処理である．Stanza は単語のレマタイズに対応しており，この処理では専ら名詞を単数形に戻す機能として使用している．

(2) キーワードの抽出：仕様書内の各文が重要であるか否かを判定するため，文中から should や must などの語を抽出する．この結果は判定 1 において使用する．

(3) 名詞句/節の抽出：文章から名詞句/節と呼ばれる語群を抽出する．ここで抽出した名詞句/節に対し，それぞれ **(4) 名詞句/節の重要度判定**を行う．名詞句/節の抽出は判定 2，判定 3 において必要な処理である．名詞を中心とする 2 語以上の単語のまとまりを名詞句/節と呼ぶ．名詞句/節の抽出にはスタンフォード大学が開発した自然言語処理ソフトウェアである Stanford Parser [9] を用いた．Stanford Parser は英文を構文木化する機能を有し，これを用いて名詞句を抽出した．

(4) 名詞句/節の重要度判定：(3) 名詞句/節の抽出の処理で仕様書内の一文から抽出した名詞句/節を読み込み，重要度を出力する．実際に仕様書の文から名詞の重要度を判定する処理を図 3 に示した．図中の文は図 1 から抜粋したものであり，網掛け部は名詞句/節を表す．名詞の下にあるのは **(1) 出現した名詞の抽出**で判定した，名詞が仕様書中に現れた回数＝名詞の重要度である．ただし，“system” や “software” の単語の重要度は十分に大きい値に設定されている．名詞句/節の重要度は，その中に出現する名詞の重要度の最大値となる．例えば，“The new UWON library system” という名詞句/節の重要度は出現する名詞の中で最も重要度が高い “system” の重要度となる．この処理はこのように読み込んだ名詞句/節の重要度を定義し，出力する．

重要度の高い名詞句/節は，判定 2，判定 3 の双方において用いる．重要度が規定値以上の名詞句/節を含む文は，判定 2 において重要な文，即ちゴール文であると判定される．本論文では，この規定値は経験的に 10 としている．また，判定 3 では，「new や envisioned などの単語を含む，重要な名詞句/節」をエポックワードと判定する．このため，

- new, upcoming などのキーワードを含む
- 重要度が規定値（ここでは 10）より大きい名詞句/節

これらの条件を満たす名詞句/節はエポックワードとして出力される．図 3 の 1 行目にある下線の引かれた new はこのキーワードに相当する．同時に網掛け部全体の重

The new UWON* library system should address such problems through
8 13 **15** **1**

software-based solution integrating all department libraries.
15 1 14 13

（網掛け部は名詞句／節　下線部はエポックワードに含まれる語）

図 3　名詞句/節の重要度判定

要度は 15 であるため，この名詞節はエポックワードである．

(5) 結果の判定：キーワードの抽出，名詞句/節の重要度判定から得た情報を元に，各文がゴール文であるか否かの判定を下す．

4　評価

本研究の手法の性能を評価するため，分野・内容の異なる 3 つの仕様書を用意し，実験では，3 節の内容を実装した Python コードを実行，結果を記録した．実験条件について次に記す．

4.1　実験

実験に際し，仕様書と評価指標を用意した．仕様書は図書管理システム（477 単語，3397 字），会議日程調整システム（570 単語，3651 字），電車の制御システム（1080 単語，7151 字）に関する，英語の仕様記述 [10] の 3 つである．抽出すべきゴール文は実験者が自ら読んで設定した．指標については，次の 4 つを用いる．

- **Accuracy（正解率）**：判定全体の正確性を示す指標である．ファイルに含まれる文の内，正しい判定（ゴール文であればゴール文として抽出，さもなければ抽出しない）を行えたものの割合を意味する．
- **Precision（適合率）**：ゴール文判定の正確性を示す指標である．ゴール文として抽出された文の内，実際にゴール文であったものの割合を意味する．
- **Recall（再現率）**ゴール文の抽出率を示す指標である．ファイルに含まれるゴール文の内，ゴール文と判定され，抽出された文の割合を示す．
- **F 値**　Precision と Recall の調和平均である．Precision と Recall，いずれかが低ければ低い値となる．

3 種の仕様書からゴール文を抽出した結果について，各々の指標の値が高ければ高いほど判定が正確であると見なす．各手法の比較検討を行うため，実験では以下の 4 つの判定法でゴール文抽出を行い，結果を比較する．

- **判定 1 のみを用いたゴール文抽出（判定 1 のみ）**
- **判定 2 のみを用いたゴール文抽出（判定 2 のみ）**
- **判定 3 のみを用いたゴール文抽出（判定 3 のみ）**
- **判定 1, 2, 3 を併用したゴール文抽出（判定 123）**

最後の判定（判定 123）では，判定 1 を満たすか，あるいは判定 2 と判定 3 を満たす文がゴール文と見なされる．これは比較的安定して良い結果を出す判定 1 をベースに，判定 1 で検出しづらいゴール文を判定 2 で検出し，同時に判定 2 の誤検出を抑えるために誤検出の少ない判定 3 を用いる事を意図している．

表 1　仕様書からのゴール文抽出結果

仕様書	判定方法	Accuracy	Precision	Recall	F 値
図書管理システム	判定 1 のみ	1.000	1.000	1.000	1.000
	判定 2 のみ	0.357	0.227	0.833	0.357
	判定 3 のみ	0.750	0.462	1.000	0.632
	判定 123	0.750	0.462	1.000	0.632
電車制御システム	判定 1 のみ	0.784	0.818	0.600	0.692
	判定 2 のみ	0.649	0.550	0.733	0.629
	判定 3 のみ	0.865	0.750	1.000	0.857
	判定 123	0.838	0.765	0.867	0.813
会議日程システム	判定 1 のみ	0.648	0.625	0.600	0.612
	判定 2 のみ	0.574	0.525	0.840	0.646
	判定 3 のみ	0.722	0.916	0.440	0.595
	判定 123	0.667	0.640	0.640	0.640

4.2　結果

実験の結果について述べる．表 1 は，今回用いる 3 つの仕様書を分析した際の Accuracy，Precision，Recall，F 値を示している．判定 1 は，図書管理システムの仕様書に関して高い精度を持ち，他 2 つの仕様書でも 6 割以上の Accuracy となっている．判定 2 は全体的に精度が低いが，抽出結果を個別に確認すると，判定 1 では抽出できなかったゴール文のいくつかを抽出していたことが分かった．判定 3 は全ての仕様書において 70%以上の Accuracy となっており，他 2 つの判定と比較しても結果は悪くない．全判定を用いた判定 123 では，Accuracy，Precision，Recall の値が，3 つの仕様書において他の判定と比較した際に安定して高い値を記録している．このことから，全判定を用いた抽出方法は各判定の欠点を均一化した上でよい結果となっていることが分かる．

5　議論

判定 123 における各判定の組み合わせは判定 1 or （判定 2 and 判定 3）としているが，これは比較的安定して良い結果を出す判定 1 をベースに，判定 1 で検出しづらいゴール文を判定 2 で検出し，同時に判定 2 の誤検出を抑えるために Recall の高い判定 3 を用いる事を意図している．結果的に各判定の欠点を相互に補うことに成功したと言えるが，より良い組み合わせが存在する可能性も考えられる．今後は各判定の性能を向上させつつ，より良い組み合わせの存在を調査すべきである．

今回の研究では，仕様書から抽出する対象をゴール自体ではなく「ゴールを含む文」とした．これは初期研究の段階でゴールの定義を明確に出来ていなかったことが原因である．ゴール自体を扱わない以上，目標としての抽象性でゴールを分類すること，ゴールを大目標と部分目標の形で階層化するなどの議論が本論文では行われていない．本研究とは別に，使用者への質問を行いながらゴールを抽出する，セミインタラクティブなゴール構築手法 [16] の構築が進められている．これは Shimada による使用者への質問を交えたゴールモデル構築手法 [2] の発展形であり，本手法に比べ精度を高めた手法である．ゴールの階層性，抽象度を論じるためには文章の意味や文と文の意味的な繋がりなど，自動的な手法では限界のある要素を考慮に入れる必要がある．セミインタラクティブな手法は使用者への質問からこの点に関する情報が得られるため，上記の問題はこちらの手法の課題としたい．

6 まとめ

本研究では，エポックワードと呼ばれる特定の語や名詞の重要度を用い，ソフトウェア仕様書からゴール文を抽出する手法を実装し，精度について検証した．ゴール文の抽出には，仕様書内で出現回数の多い単語やキーワードの有無を参照する他，“new”や “envisioned”などの修飾語と重要な名詞の組み合わせであるエポックワードの存在を手掛かりとした．

今後の課題としては，実験に用いる仕様書のサンプルの増加，より精度の高い抽出方法の考察などが挙げられる．また，現在の判定法におけるキーワードやエポックワードの種類についても，より多くのサンプルを参照することで幅広く確保すべきである．加えて，ゴールの抽象性や階層性を考慮したゴールの整理や機能の拡充，及びこれらの問題への対処法を継承した新手法への移行が考えられる．

参考文献

[1] M. Rahimi, M. Mirakhorli, J. Cleland-Huang, “Automated Extraction and Visualization of Quality Concerns from Requirements Specifications”, In RE, 2014.

[2] H. Shimada, H. Nakagawa, and T. Tsuchiya, “Constructing a goal model from requirements descriptions based on extraction rules”, Proc. Asia Pacific Requirements Engineering Conference (APRES 2017): Requirements Engineering for Internet of Things, vol.809, pp.175-188, 2018. 10.1007/978-981-10-7796-8_14.

[3] Darimont, R., van Lamsweerde, A.: Formal refinement patterns for goal-driven requirements elaboration. In: Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering. pp. 179–190. SIGSOFT ’96 (1996)

[4] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, “Extracting domain models from natural-language requirements: Approach and industrial evaluation,” Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp.250–260, MODELS ’16, ACM, New York, NY, USA, 2016.

[5] A. Dardenne, A. vanLamsweerde, and S. Fickas, “Goal-directed requirements acquisition,” SCIENCE OF COMPUTER PROGRAMMING, pp.3–50, 1993.

[6] E.S.K. Yu, “Towards modelling and reasoning support for early-phase requirements engineering,” Proceedings of the Third IEEE International Symposium on Requirements Engineering, 1997., pp.226–235, January 1997.

[7] J. Mylopoulos, L. Chung, and B. Nixon, “Representing and using nonfunctional requirements: a process-oriented approach,” IEEE Transactions on Software Engineering, vol.18, no.6, pp.483–497, June 1992.

[8] Stanford NLP Group. “Stanza – A Python NLP Package for Many Human Languages”. github.io. 2022/7/14. https://stanfordnlp.github.io/stanza/, (2022/7/20).

[9] The Stanford NLP Group. “Stanford Parser”. The Stanford NLP Group. 2022/7/20. https://nlp.stanford.edu/software/lex-parser.shtml, (2022/7/20).

[10] A. V. Lamsweeide, “Requirements Engineering”, Wiley, 2009.

[11] Woldeamlak, S., Diabat, A. and Svetinovic, D.: Goal-Oriented Requirements Engineering for Research-Intensive Complex Systems: A Case Study, Systems Engineering, Vol. 19, No. 4, pp. 322–333 (2016).

[12] Pimentel, J., Vilela, J. and Castro, J.: Web tool for Goal modelling and statechart derivation, in Proc. of the 23rd IEEE International Requirements Engineering Conference (RE 2015), pp. 292–293 (2015).

[13] Nguyen, T. H., Grundy, J. and Almorsy, M.: Rule-based Extraction of Goal-use Case Models from Text, in Proc. of the 10th Joint Meeting on Foundations of Software 67 Engineering (ESEC/FSE 2015), ESEC/FSE 2015, pp. 591–601, New York, NY, USA (2015), ACM.

[14] Horkoff, J., Salay, R., Chechik, M. and Sandro, A. D.: Supporting early decisionmaking in the presence of uncertainty, in Proc. of the 22nd IEEE International Requirements Engineering Conference (RE 2014), pp. 33–42 (2014).

[15] Zee, van M., Bex, F. and Ghanavati, S.: Rationalization of goal models in GRL using formal argumentation, in IEICE TRANS. INF. & SYST., VOL.E103–D, NO.6 JUNE 2020.

[16] H. Nakagawa, H. Shimada, T. Tsuchiya.: Interactive Goal Model Construction Based on a Flow of Questions, in IEICE Transactions on Information and Systems, Volume E103.D Issue 6, pp. 1309-1318 (2020).

アヤトゥス・カルタの拡張によるユーザ視点に基づくMVP抽出手法の提案

Extraction of user-centered MVP using goal-oriented requirements analysis

田中 貴子 * 斎藤 忍 †

あらまし フィンランドの初等教育で活用されているツールであるアヤトゥス・カルタを拡張して，ユーザ視点に基づいたMVPを抽出する手法を提案する．提案手法は，ゴール指向分析とアヤトゥス・カルタを応用しており，要求工学の初学者でも容易にユーザ視点でMVPの抽出ができるのが特徴である．本稿では本手法の手順，および適用例を示す．

Summary. This paper proposes a methodology for extracting MVP from User-Centered Requirements Engineering using goal-oriented requirements analysis by extending the Finnish method Ajatus Kartta. This methodology applies the kartta to goal-oriented requirements analysis that can clarify functional and non-functional requirements that contribute to goals. Even beginners of User-Centered Requirements Engineering can easily extract MVPs from the user's perspective by this. This paper discusses the procedure of this methodology and demonstrates its effectiveness by applying the concrete case.

1 はじめに

サービスやプロダクトは機能の数の多さや性能を磨くだけではユーザから選ばれにくくなっている．VUCAの時代は不確実性の高い要素が多く，何が選ばれるかの答えが誰も分からないケースも多い．さらに，デジタル技術による価値提供が可能になることにより参入障壁が下がり，業界内の競合他社だけでなく，業界外の企業が競合となることになる．そのため，素早くユーザにサービスやプロダクトの価値を提供してフィードバックを得ながら，答えを見つけていくことが求められ [4]，アジャイルやリーンのアプローチが注目されている．

素早くユーザに価値提供するには，サービスやプロダクトを体現し，顧客が抱える課題を解決する最小限のプロダクト (MVP:Minimum Viable Product，以降MVP) を見極め，素早く形にすることが求められる．サービスやプロダクトの開発者が，MVPとして機能を絞り込むポイントは，ターゲットユーザにとって価値があると感じるもの（ユーザの価値基準）を考慮することである．しかし，開発者がユーザ視点のアプローチに慣れていないと，MVPを絞り込む際にターゲットユーザの価値基準が曖昧になる．ユーザ視点のアプローチでは，ターゲットユーザの価値基準を明確にする手法として，ユーザ体験のモデル化や体験価値の探索の手法が用いられる．例えば，ペルソナ法やアイデア発想のコンセプト作成にバリュープロポジションキャンバス，ビジネスモデルキャンバスが挙げられる [1]．

一方，各手法が提案するテンプレートを埋めることのみに注力してしまうと，各手法の有効性を必ずしも発揮できない．筆者らは，ユーザ視点のアプローチに関する初学者が多いチームに遂行支援をおこなっているが，各手法を十分に活用できないケースに幾つも遭遇している．例えば，ペルソナシート（テンプレート）を埋めているとはいえ，ペルソナの属性の内容が曖昧である，開発者側の価値観で最終的にペルソナの内容を決めている，といったケースがある．また，B to Bサービスを開発する場合にユーザ視点が欠落してしまうケースにも遭遇してきた．一般にB to

*Takako Tanaka, NTT テクノクロス株式会社

†Shinobu Saito, 日本電信電話株式会社

B サービスで考えるユーザは,「顧客」と「利用者」の 2 つの属性がある．顧客は，サービスの対価を支払うかどうか（導入判断）をする人達（例: 企業の IT 部門）である．利用者は，当該サービスを利用するが，自らは対価を支払わない人達（企業の業務部門）である．B to B サービスを開発者が検討する際に，顧客ばかりに目が行ってしまい，利用者の考慮が不足しているケースも散見される．

そこで本稿では，ユーザ視点のアプローチに不慣れな初心者でも，ユーザを考慮した MVP 抽出を支援する方法を提案する．本稿の構成は以下のとおりである．2 章では，提案手法の手順，および具体的な適用事例で詳細を説明する．3 章では，適用事例の内容を説明する．第 4 章では，適用事例で得られた知見と今後の課題を示す．

2　アヤトゥス・カルタの拡張によるユーザ視点に基づく MVP 抽出手法

ゴール指向要求分析は，ユーザの課題やニーズを起点（最終ゴール）として設定し，それを達成するための手段をサブゴールとして展開していく手法である．展開されたサブゴールは，最終的にサービスに対する機能要求や非機能要求まで落とし込んでいく [2]．それぞれの機能要求や非機能要求が最終ゴールに寄与するかを明確に分析可能であり，要求の網羅性の担保にも有用である．しかしながら，ゴール指向要求分析によって得られた機能要求や非機能要求が，ターゲットユーザにとって本当に欲しい内容であるかの判定は難しい．そこで本稿では，ゴール指向要求分析にフィンランドの初等教育で活用されているツールの 1 つである「アヤトゥス・カルタ（以降，カルタ）」のコンセプトを適用し，初学者でも容易に MVP を抽出可能とする手法を提案する．以降，2.1 ではカルタの基本コンセプトについて記述し，2.2 では提案手法を具体的な適用例とともに説明する．

2.1　カルタの基本コンセプト

カルタは，フィンランドの教育の中で活用されており，小学生の国語や読解を中心に利用されている [3]．思いついたものを書き連ねるマインドマップに似ているものである．カルタがマインドマップと異なる点は，マインドマップは無秩序に思いついたものを書き連ねていくが，カルタは中心に連想する言葉を書き，それは何か，それはどんなものなのか，それは何をするものかを，それぞれの枝に広げていく形で書いていく（図 1）．

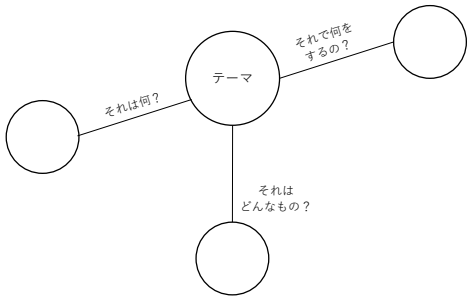


図 1　アヤトゥス・カルタの基本構成

例えば，ロシアに伝わる昔話である「おおきなかぶ」をカルタで分析すると，図 2 のようになる．基本的な質問事項の枝を伸ばし（図 2 左側），その質問事項に関して，物語に書かれていることを細かい情報を含めて書き出し，枝を伸ばしていく

（図 2 右側） [3]．物語に書かれていること以外に，自分の解釈を追加することもできる．例えば，「いつ？」については物語では書かれていないが，「300 年くらい前」と自分で考えたものがあれば，枝を伸ばして文字の色を変えて物語に書かれているものと区別して書き込む．

このようにカルタはテーマを中心に置き，問いかけの枝を伸ばし，その問いかけに関する内容や回答から発想を広げていく．提案手法では，テーマを中心に質問事項の枝を伸ばすカルタのアプローチを適用する．即ち，中心のテーマとして最終ゴールを置き，そこから伸ばす枝としての問いかけをユーザ視点で発想できる形で用意する．これにより，ユーザ視点に基づく MVP を抽出できるようになる．この問いかけや具体的なカルタを用いた MVP の抽出手法を次節で示す．

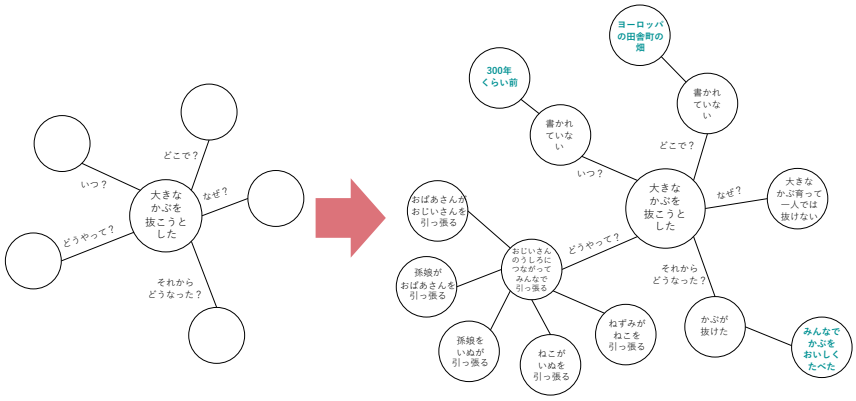


図 2　カルタによる物語の分析の例（[3] に基づき筆者らが独自に作成）

2.2　カルタの拡張

提案手法では，カルタを拡張し具現化する機能を考える前に，MVP を抽出する目的，ターゲット，提供価値を明確化しておく必要があるが，それらの詳細は本稿では割愛する．

- **MVP 候補となる提供価値を具現化する機能を考える**

明確化した提供価値をカルタの中心に書き，そこから以下の問いかけをセットにして MVP 候補となるサービスやプロダクトに必要な機能を考えていく．

a. その提供価値を満たすために顧客や利用者は何を得たいか
b. それを実現する機能はどのようなものか
c. その機能はなぜ必要か

はじめに，提供価値を書いた中心の円に a のコネクタをつなぐ．そのコネクタの先に，中心の円の提供価値から想像されるユーザ（顧客や利用者）の「〜したい」を書き出す．このとき，コネクタ 1 個につき想定されるユーザの「〜したい」を対応づける．次に、その書き出した項目にコネクタ b をつなぐ．つないだユーザの「〜したい」を実現する機能を，そのコネクタ b の先に書き出していく．最後に，書き出した機能にコネクタ c をつなぎ，そのコネクタの先に書き出した機能がなぜ必要なのか，その理由を書き出す（図 3）．コネクタは図 4 のようにつないでいく．

MVP は，ユーザにとって本当に必要な機能に絞り込む必要がある．書き出した機能に対してなぜ必要なのかの理由を書き出すコネクタを用意している．これによ

り，機能を抽出した後，改めてなぜその機能が必要なのか考える仕組みを加えている．もし，理由が書けない機能がある場合，ニーズから想定される機能であっても，ユーザの価値にはならないことを，機能を考える時点で検知することができる．

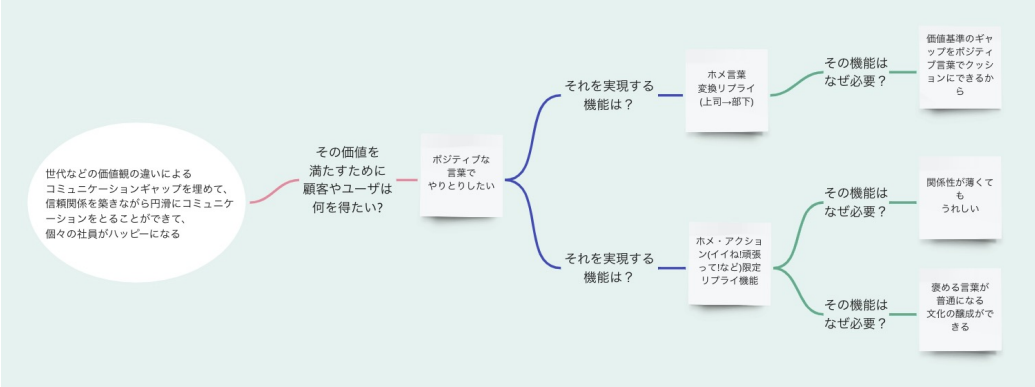


図 3　MVP 抽出用カルタ（実際の適用事例）

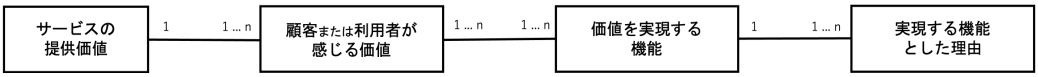


図 4　コネクタの多重度

- **提供価値からその価値を具現化する機能を考える**

提供価値からつないだコネクタに書き出した内容を俯瞰して，書き出した機能がユーザ（顧客と利用者）の提供価値なのかを判断する．判断をビジュアル化するために価値の提供先用アイコン（図 5 左）を用いて機能に置いていく．価値の提供先用アイコンの置かれた機能と，テーマとして中心に書いた提供価値の双方を見比べて，サービスやプロダクトにとって MVP になり得る機能を絞り込む（本例では 3 個に絞り込みを実施）．その上で，選ばれた機能に MVP 選択用アイコン（図 5 右）を置いていく．生成された成果物としてのカルタは図 6 である．

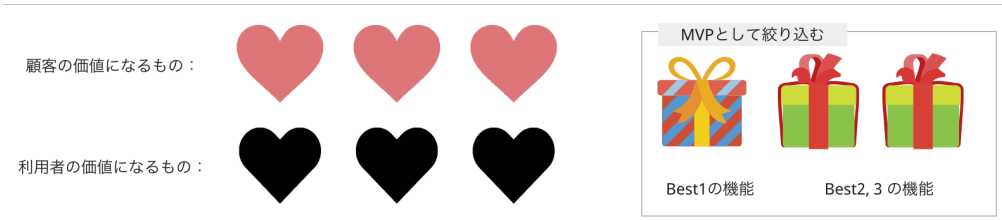


図 5　価値提供先アイコンと MVP 選択用アイコン

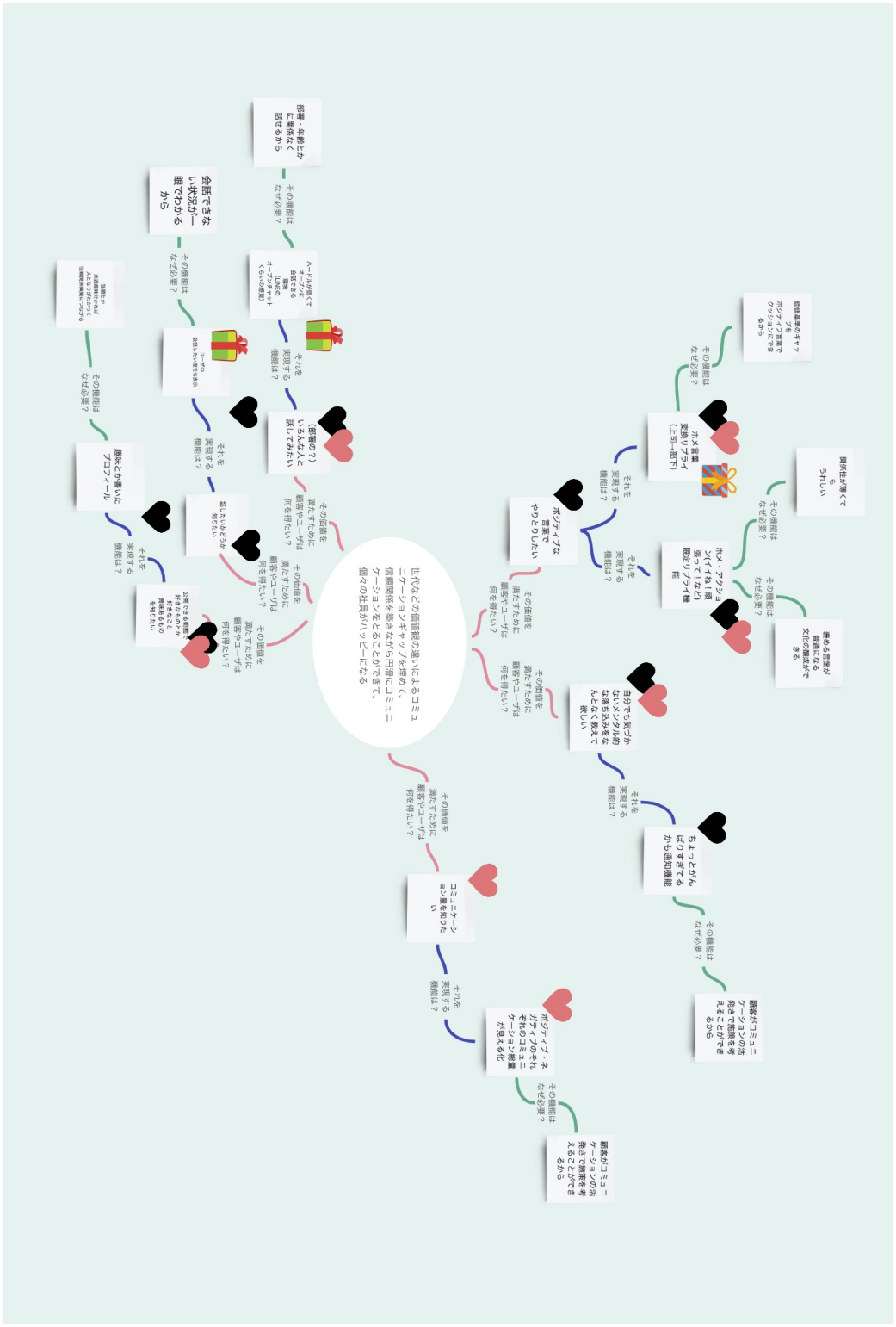


図 6　MVP を抽出したカルタ（実際の適用事例）

3　提案手法の適用事例

3.1　実施者の属性と時間配分

企業内において要求工学の初学者（1 名）に提案手法の実践をしてもらった．初学者はソフトウェア開発経験はあるが，サービスやプロダクトの企画や創出経験がなく，MVP という考え方もあまりよく知らない．実際の実践では，提案手法の有識者も参画し，初学者と有識者の 2 名で議論をしながら，提案手法を表 1 に示す手順と時間配分で実施した．なお，本事例では，あらかじめ手順 1 は事前に実施し、初学者には提示をしている．

3.2　実行可能性の確認

表 1 に示すように，約 2 時間（作業の説明と作業時間）で MVP の抽出と検証可能性の検討までを実施できた．受講者からは,「MVP というものが理解でき,（提案手法は自分の）仕事への貢献度が高い」,「プロトタイプ（実装）という形に縛られずユーザのニーズを早期にキャッチする考え方を実務に活かしたい」といった感想が得られた．最短で本事例のように約 2 時間で MVP が抽出可能と考えられる．

表 1　研修内での作業時間

手順 1	開発対象のサービスやプロダクトの目的を明確にする	– [*1]
手順 2	（アーリーアダプターを含む）ターゲットとする ユーザ（顧客と利用者）の課題を明確にする	18 分
手順 3	明確にしたユーザへの（サービスやプロダクトの） 提供価値を定義する	18 分
手順 4	提供価値からその価値を具現化する機能を考える	20 分
手順 5	ユーザ（顧客と利用者）の価値となる機能を整理し， MVP を抽出する	30 分
手順 6	抽出した MVP がプログラム実装以外に検証可能か検討する	15 分

[*1] 参加者に事前に提示

4　まとめ

本稿では，フィンランドの初等教育で活用されているツールの 1 つであるアヤトゥス・カルタを拡張し，ゴール指向要求分析を用いたユーザ視点の MVP の抽出手法を提案した．実際の企業の研修において実施された適用事例を用いて，カルタによる拡張 MVP の抽出手法とその成果物の内容を示した．

適用事例では要求工学の初学者の参加者が中心となり，MVP を作成した．本提案手法をテンプレート化し，そのテンプレートを埋めていく形で，有識者が参加者としてディスカッションする形で伴走した．これにより，初学者であっても，約 2 時間で MVP の抽出までおこなうことができた．参加者は MVP を理解し、実務に活用可能性を感じていた．ユーザ視点の検討が不慣れな初学者であっても，ユーザを意識した MVP 抽出を本提案手法で支援できることにより，提案手法の実現可能性を確認できた．今後は，提案手法の実践を通じて，抽出されたユーザ視点の MVP の有効性や他手法との比較評価を検討していく．

参考文献

[1] 安藤 昌也: UX デザインの教科書, 丸善出版, 2016
[2] 鵜林 尚靖: レクチャー　ソフトウェア工学, 数理工学社, 2021.
[3] 北川 達夫，フィンランド・メソッド普及会: 図解 フィンランド・メソッド入門, 経済界, 2005.
[4] デビッド ロジャース: DX 戦略立案書, 白桃書房, 2021

欠損確率に基づいた欠損データ作成手法の提案とソフトウェア開発データにおける評価

Proposal of a Method for Creating Missing Data Based on Missing Probability and Evaluation Using Software Development Data

上佐 公太郎* 柿元 健†

あらまし 定量的管理手法の欠損値に対する評価において，様々な欠損値を含むデータを準備する必要があるが，評価に必要なだけ準備することは難しいため，人為的に欠損値を与えることとなる．しかし，従来の欠損データ作成手法は，欠損メカニズムの定義に厳密に則っているとは言えず，与えられる欠損率の割合にも制限があった．そこで，本稿では，欠損メカニズムの定義により則った欠損確率に基づいて欠損値を与える欠損データ作成手法を提案する．実証的評価の結果，提案手法においてそれぞれの欠損メカニズムらしい結果が得られ，より妥当性のある評価となった．

1 はじめに

ソフトウェア開発プロジェクトでは開発初期段階において，プロジェクトを成功に導くために様々な定量的管理が行われる．定量的管理の一例であるコスト見積では，正確にコストを見積もることで，コストが超過しプロジェクトが赤字になることや，コスト余剰となり開発効率が悪くなることを防ぐ効果が期待される．このように，正確な定量的管理がプロジェクトの成功を導く[1].

定量的管理において，企業等で過去に実施されたソフトウェア開発プロジェクトの実績データ等のソフトウェア開発データが用いられる．しかし，ソフトウェア開発データには，記録漏れ等の様々な要因によって値が欠落している部分である欠損値が含まれていることが多い. 欠損値は, データが欠損した要因によって Missing Completely At Random (MCAR)，Missing At Random (MAR)，Non-ignorable Missingness (NM)の 3 種類の欠損メカニズムに分類される[4]．定量的管理において用いられる見積手法には，モデル作成に利用するデータに欠損値が存在すると見積を行うことができない手法[3][5]や，見積は行えても見積精度が低下する手法がある[2]．そこで，データから欠損値を取り除くために欠損値処理が適用される．しかし，適用する欠損値処理が適当でないと見積精度の低下を招くおそれがある．そのために，データに含まれる欠損値の状況と欠損値処理の有効性の関係を明らかにする必要がある．しかし，欠損メカニズムが明らかとなっている，様々な欠損値の割合のソフトウェア開発データを準備することは難しい．そのため，データに人為的に欠損値を与えることで，実証的評価を行う必要がある．

欠損メカニズムに応じて欠損値を与える方法として，Strike ら[6]の方法がある．これまで，我々の研究グループでは Strike らの方法を用いてきた[2][9]．しかし，Strike らの方法は，ランダム要素の影響が多いため，欠損メカニズムに厳密に則っているとは言い難く，実証実験の結果に少なからず悪影響を与えていると考えられる．

そこで本稿では，各欠損メカニズムに応じた欠損確率に基づいた欠損データ作成手法を提案し，欠損値を含むデータによる実証実験の結果をより妥当性のあるものにするこ

* Kotaro Uwasa, 香川高等専門学校専攻科創造工学専攻
† Takeshi Kakimoto, 香川高等専門学校電気情報工学科

とを目的とする．そのため，提案手法と Strike らの方法によって作成された欠損データを用いて比較評価を行う．なお，MCAR は完全ランダムで欠損するメカニズムであり，欠損確率を用いての改善の余地がないため提案手法では対象としない．

評価実験では，従来手法によって作成された欠損データと提案手法によって作成した欠損データに，それぞれ平均値代入法やコサイン類似度に基づく補完法，ユークリッド距離の逆数を類似度とした類似度に基づく補完法，そして多重代入法の 4 種類の補完手法を適用する．評価尺度には相対誤差である MBRE を用いる．

2 欠損メカニズム

2.1 Missing Completely At Random (MCAR)

MCAR は，ある値が欠損する確率は，データ中のいかなる値にも依存しない欠損メカニズムである．すなわち，ランダムに欠損するケースを示している．例として，不注意による記録漏れや記録紛失などが挙げられる．

2.2 Missing At Random (MAR)

MAR は，ある値が欠損する確率は，ある変数(メトリクス)の値の大きさに依存する欠損メカニズムである．例として，規模が小さなプロジェクトほどレビューが省略されやすく発見不具合数が欠損値となりやすい場合が挙げられる．この場合，レビューにおける発見不具合数が欠損する確率は規模に依存する．なお，どのように依存するかは定義されていない．

2.3 Non-ignorable Missingness (NM)

NM は，ある値が欠損する確率は，その変数（メトリクス）の値自体の大きさに依存する欠損メカニズムである．例として規模が小さなプロジェクトほど人員的余裕がないため規模が記録されず欠損値となりやすい場合が挙げられる．この場合，規模が欠損する確率は規模自体の大きさに依存する．

3 欠損値処理

3.1 概要

欠損値処理は，欠損値を含むデータから欠損値をなくすための手法である．欠損値処理には，条件を満たす欠損値を削除することでデータから欠損値をなくす手法と，欠損値に既存のデータから何らかの補完値を算出し欠損値に挿入することで欠損値をなくす手法がある．削除による手法は，削除後のデータは元データの一部のため誤差は含まないが，データが持つ情報量が低下してしまう．補完による手法は，データの損失はないが，補完値によって誤差の含有を招く可能性がある．本稿では，補完による欠損値処理の補完精度により欠損データ作成手法を評価する．本稿で用いる欠損値処理手法を以降に示す．

3.2 平均値挿入法

平均値挿入法は，欠損値を含む各メトリクスの平均値を補完値として欠損値に挿入する手法である．同じメトリクスに複数の欠損値が存在している場合，全て同一の値が補完値として挿入されることになる．

3.3 類似度に基づく補完法

類似度に基づく補完法は，コサイン類似度やユークリッド距離の逆数を用いて類似度

を算出し，類似度の高いケースの類似度を重みとした重みつき平均値を補完値として扱う．

3.4 多重代入法

多重代入法は欠損値そのものを推定するのではなく，母集団の分布を推定することによって欠損値を補完する手法である．本稿で採用した EMB アルゴリズムは以下の通りである．

EMB アルゴリズムは期待値最大加法(EM)とノンパラメトリック・ブートストラップから構成されるアルゴリズムである．以下にその概要を述べる．

① ノンパラメトリック・ブートストラップ法により復元抽出により複数の再標本を得る．
② 各再標本に対し，以下の EM アルゴリズムを適用する．
  ②-(1) 欠損値を含む不完全データとパラメータ推定値を条件とした，完全データの対数尤度の条件付き期待値を計算する．
  ②-(2) 尤度を最大化するパラメータ値を推定する．
  ②-(3) 上記をパラメータ値が収束するまで繰り返す．
③ EM アルゴリズムの結果から欠損値を補完する．

4 従来手法による欠損データ作成手法

Strike ら[7]の MAR，NM 欠損メカニズムによる欠損値の与え方を示す．MAR は依存するメトリクスの値によりソートし 5 分割する．NM は欠損値を与えるメトリクスの値によりソートし 5 分割する．5 分割した各グループに対して，最も影響を受けるグループから順に，与える欠損値数の 40%，30%，20%，10%だけ与える．ソートの方向は想定する依存内容で決定され，各グループにおける欠損値の与え方はランダムである．そのため，規模の大きなプロジェクトほど，ランダム要素の影響を強く受け，欠損メカニズムの定義に厳密に則っているとは言い難い．

5 欠損確率による欠損データ作成手法

5.1 概要

本稿では，MAR は依存するメトリクスの値の大きさ，NM は各メトリクスの値の大きさに基づいた欠損確率を各値に付与することで，従来手法のランダム要素による，欠損メカニズムに厳密に則っていなかった原因の解消を狙った手法を提案する．

5.2 MAR 作成手法

以下に提案手法による MAR の欠損データ作成手法の手順を示す．

① 依存するメトリクスの値の大きさにより各メトリクスをソートする．
② 依存するメトリクスの各値の大きさを，依存するメトリクスの最大値で割った値を欠損確率とし，各メトリクスの各値に付加する．これにより，従来手法で各グループ内では値の大きさに関係なくランダムで欠損していた問題点を解消する．
③ 欠損確率で降順に，各値ごとにランダムで閾値を割り当て，閾値が欠損確率未満であれば，その値を欠損させる．
④ 与えたい欠損値数となるまで上記③を繰り返す．

5.3 NM 作成手法

以下に NM の欠損データ作成手法の手順を示す．

① 各メトリクスを値の大きさによりソートする．
② 各値の大きさをそのメトリクスの最大値で割った値を欠損確率として，各値に付加する．これにより，従来手法で各グループ内では値の大きさに関係なくランダムで欠損していた問題点を解消する．
③ 欠損確率で降順に，各値ごとにランダムで閾値を割り当て，閾値が欠損確率未満であれば，その値を欠損させる．
④ 与えたい欠損値数となるまで上記③を繰り返す．

6 評価実験

6.1 目的

本稿では，従来手法と提案手法を用いて MAR，NM のメカニズムで過去データに欠損を与えた欠損データを作成し，既存の欠損値処理を適用することによって，欠損データ作成法の比較評価を行った．

6.2 使用データセット

過去のソフトウェア開発プロジェクトの実績データで，欠損値を含まないデータセットである China Software ProcessBenchmrk Standard Group(CSBSG)データセット[8]を用いた．CSBSG データセットは要素数 7984，プロジェクト数 499，メトリクス数 16 で，欠損値は存在しない．メトリクスのうち Deleted は 8 割以上に 0 が記録されており，本実験では使用しないこととした．また，MAR の欠損データを作成する際，依存変数は全て Effort とした．

6.3 実験方法

実験手順を以下に示す．
① CSBSG データセットに MAR，NM，それぞれの欠損メカニズムに応じた欠損を従来手法と提案手法で 10 回ずつ与えた．データが欠損している割合による変化を調査するために，与える欠損の割合は 10%，20%，30%，40%の 4 パターンとした．また，MAR と NM は値が大きいほど欠損する確率が高くなるものとした．
② 欠損を与えたデータに平均値挿入法，類似度に基づく補完法，EMB アルゴリズムによる多重代入法を適用し，欠損値を補完した．
③ 補完値と実測値から予測精度を求めた．

6.4 評価尺度

本実験では，予測精度の評価尺度として，過大予測，過小予測を等しく評価できる相対誤差である BRE の平均値である MBRE(Mean Balanced Relative Error)を用いた．MBRE は以下の式で求めることができる．

$$\mathrm{MBRE} = \frac{1}{m}\sum_{i=1}^{m}\frac{|X_i - E_i|}{\min(X_i, E_i)} \quad (1)$$

ここで，$\boldsymbol{X_i}$は実測値，$\boldsymbol{E_i}$は補完値，$\boldsymbol{m}$は補完値数を表している．また，MBRE は値が小さいほど予測精度が高いことを表す．

7 実験結果

従来手法と提案手法により MAR で欠損させたデータに各補完手法を適用した場合の補完精度である MBRE を表 1，2 に示し，従来手法と提案手法により NM で欠損させた

表 1　従来手法で与えた MAR における補完精度（MBRE）

補完手法	10%	20%	30%	40%
平均値挿入法	2.36	2.27	2.25	2.26
コサイン類似度	1.12	1.17	1.48	1.90
ユークリッド距離の逆数	7.10	6.70	6.66	6.27
多重代入法	1.87	2.16	2.74	2.43

表 2　提案手法で与えた MAR における補完精度（MBRE）

補完手法	10%	20%	30%	40%
平均値挿入法	2.50	2.30	2.52	3.12
コサイン類似度	1.24	1.29	1.37	1.68
ユークリッド距離の逆数	7.29	7.14	9.27	11.80
多重代入法	1.32	1.93	2.48	3.31

表 3　従来手法で与えた NM における補完精度（MBRE）

補完手法	10%	20%	30%	40%
平均値挿入法	1.91	1.85	2.05	2.73
コサイン類似度	1.07	1.18	1.47	2.91
ユークリッド距離の逆数	4.66	4.29	4.60	5.49
多重代入法	2.64	1.25	2.10	3.36

表 4　提案手法で与えた NM における補完精度（MBRE）

補完手法	10%	20%	30%	40%
平均値挿入法	4.10	4.22	4.86	6.55
コサイン類似度	1.12	1.17	1.48	1.90
ユークリッド距離の逆数	7.00	7.49	8.46	10.09
多重代入法	3.02	4.39	5.91	12.80

データに各補完手法を適用した場合の補完精度である MBRE を表 3，4 に示す.

表 1，2 より，MAR において，各補完手法を比較すると，従来手法，提案手法とも，ユークリッド距離の逆数を類似度として用いた類似性に基づく補完法の補完精度が最も低く，コサイン類似度を用いた類似性に基づく補完法が最も補完精度が高くなった．また，欠損値の割合で比較すると，従来手法は欠損値の割合と補完精度にあまり関連が見られないが，提案手法は，多くの場合欠損値の割合が高くなると補完精度が低下した.

表 3，4 より，NM においても，各補完手法を比較すると，従来手法，提案手法とも，ユークリッド距離の逆数を類似度として用いた類似性に基づく補完法の補完精度が最も低く，コサイン類似度を用いた類似性に基づく補完法が最も補完精度が高くなった．また，欠損値の割合で比較すると，従来手法は欠損値の割合と補完精度にあまり関連が見られないが，提案手法は，欠損値の割合が高くなると補完精度が低下した.

8　考察

MAR については，どの補完手法も従来手法と提案手法の補完精度に大きな差は出なかったが，NM については，コサイン類似度を除いて提案手法のほうが補完精度が大きく低下した．これは，今回は値が大きいほど欠損させており，ソフトウェア開発データは指数的に値が大きくなるメトリクスが多いことから，各メトリクスとも大きな値がほぼ欠損したことにより，補完手法において推測が難しく，補完値が小さくなり補完精度

が著しく低下したと考えられる．また，NM の方がそれぞれの傾向が顕著に表れているが, MAR は欠損する確率が依存するメトリクス変数が残っているため推測しやすいためと考えられる．

また，7.で述べたように MAR，NM ムとも，提案手法では欠損値の割合が高くなるほど補完精度が低下する傾向が顕著に表れている．これまで，CSBSG データセットに従来手法で欠損値を与えた研究では，他のデータセットと異なり，このような傾向が出にくく，CSBSG データセットの特性ではないかと考えられてきたが，欠損データ作成手法の影響であったのではないかと考えられる．

以上のことから，提案手法の方が各欠損メカニズムの影響について妥当な評価ができていると考えられる．

ソフトウェア開発データでは，各メカニズム以外に，バースト的な欠損が含まれることが多い．しかし，一般的な欠損メカニズムとはされていないため，バースト的な欠損のメカニズムの解析と欠損データ作成手法の構築が今後必要である．

9　まとめ

本稿では，各欠損メカニズムに応じた欠損確率に基づいた欠損データ作成手法を提案し，従来手法と提案手法によって作成された欠損データを用いて 4 種類の補完手法を用いて評価実験を行った．実験結果より，MAR，NM ともに従来手法に比べて，欠損値の割合が高くなるほど補完精度は低下する傾向が顕著に表れた．また，NM では提案手法で欠損値を与えた方が補完精度は著しく低下した. 従来手法で欠損値を与えた研究では，このような傾向が出にくかったため，提案手法で欠損値を与えることでより妥当な評価ができたと考える．

今後は，バースト的な欠損のメカニズムの解析と欠損データ作成手法の構築を行うとともに，その他の補完手法による精度比較や予測手法での精度比較，および別のデータセットによる同様の評価実験を行う予定である．

謝辞　本研究の一部は JSPS 科研費 JP19K11915 の助成を受けた．

参考文献

[1] Boehm, B.W. : *Software engineering economics*, Prentice Hall, 1981.
[2] 柿元健, 角田雅照, 大杉直樹, 門田暁人, 松本健一：協調フィルタリングに基づく工数見積もり手法のデータの欠損に対するロバスト性の評価, 電子情報通信学会論文誌 D, Vol.J89-D, No.12(2006), pp.2602-2611.
[3] Kromrey, J. and Hines, C. : Nonrandomly Missing Data in Multiple Regression: An Empirical Comparison of Common Missing-data Treatments, *Educational and Psychological Measurement*, Vol.54, No.3(1994), pp.573-593.
[4] Little, R.J.A., and Rubin, D.B. : *Statistical Analysis with Missing data*, Third edition, John Wiley and Sons, 2019.
[5] Mendes, E., Watson, I., Triggs, C., Mosley, N., and Counsell, S. : A Comparative Study of Cost Estimation Models for Web Hypermedia Applications, *Empirical Software Engineering*, Vol.8, No.2(2003), pp.163-196.
[6] 内閣府経済社会総合研究所 景気統計部：欠損値補完に関する調査研究報告書【詳細版】, 2017. http://www.esri.cao.go.jp/jp/stat/report/report_all_detail.pdf
[7] Strike, K., El Eman, K., and Madhavji, N. : Software Cost Estimation with Incomplete Data. *IEEE Transaction on Software Engineering*, Vol.27, No.10(2001), pp.890-908.
[8] Wang, H., Wang, H., and Zhang, H. : Software Productivity with CSBSG Data Set. *2008 International Conference on Computer Science and Software Engineering*, 2008, pp.587-593.
[9] 渡辺竜, 柿元健：欠損値の補完と削除を用いたハイブリッド欠損値処理の提案, コンピュータソフトウェア, Vol.34, No.4(2017), pp.144-149.

Fault-prone モジュール予測における第三者データに基づいた外れ値除去

Outlier Removal Based on Third-Party Data in Fault-prone Module Prediction

西浦 生成 * **門田 暁人** †

あらまし ソフトウェア開発において，バグを含む可能性の高いモジュール（fault-prone モジュール）を高い精度で予測することができれば，テストやデバッグの効率化に繋がる．予測精度の向上を目的として，予測モデルの訓練データに含まれる，予測に悪影響を与えるような外れ値データの除去が取り組まれている．本稿では，バージョン間予測を対象とし，よりロバストな外れ値除去手法として，予測対象とは異なるプロジェクトから得られた第三者データセットを用いて，訓練データ中の外れ値を特定・除去する手法を提案する．評価実験の結果から，提案手法は大多数のプロジェクトで予測精度を向上でき，また代表的な既存の外れ値除去手法である MOA や CC-MOA よりも高い効果を持つことが示された．

1 はじめに

ソフトウェア開発においてバグの混入は不可避であり，リリース前にできる限りバグを発見し除去することが望ましい．一方で，通常，バグ発見のためのレビューやテストに割かれるリソースには制限があり，全てのモジュール（ソフトウェアを構成する部品）を網羅的に検証することは難しい．こうした問題に対し，各モジュールに含まれるバグの有無をあらかじめ予測することで，バグを含む可能性が高いモジュール（fault-prone モジュール）に検証リソースを集中させることができる [1]．また fault-prone モジュール予測の精度が高いほど，バグ発見作業のより高い効率化が期待できる．

予測精度向上を目的とした研究分野の一つとして，予測モデルの構築のための訓練データに着目し，外れ値を特定し除去することで性能の良いモデルの構築を目指す手法（外れ値除去法）が提案されている [2] [3] [4]．外れ値除去の代表的な手法として，マハラノビス距離を用いて単一のデータセット内で外れ値を識別する Mahalanobis Outlier Analysis（MOA） [2] および Cross-Class Mahalanobis Outlier Analysis（CC-MOA） [3] が挙げられる．これらの手法は，2 章で示すように，直観的に分かりやすいアイデアを基にしており，評価実験によってその有効性が示されている [3]．

ただし，単一のデータセット内での分析により外れ値を検出する従来の外れ値除去手法では，当該データセットへのオーバーフィッティングが発生しやすいと考えられる．同一プロジェクトの過去のバージョンのバグデータを基に新しいバージョンのバグ予測を行うバージョン間予測では，コンセプトドリフト（ソフトウェアの時間変化に伴ってバグに影響する要因が変化すること） [5] [6] が生じる場合があることが知られており，こうしたオーバーフィッティングによって予測性能向上の頭打ちに繋がる可能性がある．

本稿では，この問題に対処するための新たな外れ値除去方法を提案する．提案方法では，まず，予測対象外のプロジェクトのデータセット（第三者データセット）を多数用意し，それらから特定のプロジェクトに依存しない「一般的なバグ」の特徴を学習させる．そして，バージョン間予測のための訓練データに対し，一般的なバグの特徴に当てはまらないバグありモジュールを外れ値として除外する．また逆に，

*Kinari Nishiura, 岡山大学大学院自然科学研究科

†Akito Monden, 岡山大学大学院自然科学研究科

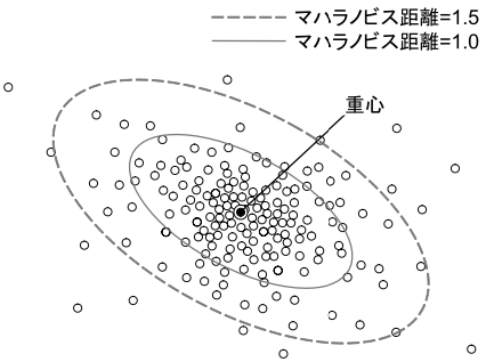


図 1　マハラノビス距離（ [3] より引用）

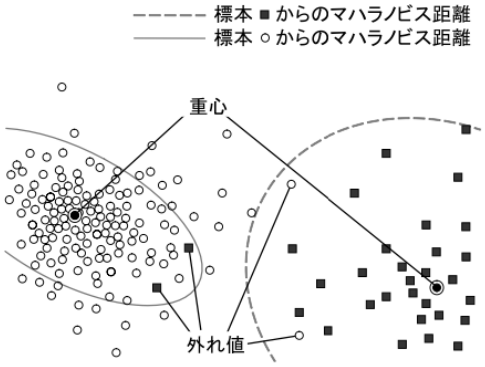


図 2　CC-MOA（ [3] より引用）

バグが無いにも関わらず一般的なバグに類似した特徴を持つモジュールについても外れ値として除外する．これらにより，コンセプトドリフトに対してよりロバストな外れ値除去の実現が期待できる．

また，既存研究 [3] では，従来の外れ値除去法である MOA や CC-MOA の有効性が示されているものの，実験上の課題として，(1) 高い精度が確認されているランダムフォレスト等のアンサンブル学習モデルを採用していない．(2) 評価用データセットとして使用されたのはモジュール数の多い 3 個のプロジェクトのみである，(3)F1 値よりもロバストな評価尺度である AUC (Area Under the Curve) of ROC (Receiver Operating Characteristic) が採用されていない，が挙げられる．

そこで本稿では，実験に使用する fault-prone 予測モデルにはランダムフォレストを採用し，10 個のプロジェクトからそれぞれ 2 個のバージョンを抽出したデータセットを用いて評価実験を行い，AUC of ROC による予測精度評価を通して MOA および CC-MOA と提案手法を比較する．

2　関連研究

2.1　Fault-prone モジュール予測

Fault-prone モジュール予測モデルは，モジュールがバグ（fault）を含んでいるか否かを予測することを目的とし，過去に開発されたモジュールの特性値とバグの有無を記録したデータセットを用いて構築される [1]．これまでに多数の予測モデルが提案されているが，本研究ではその高い効果によって広く用いられているランダムフォレストによるモデリング手法を採用する．

ランダムフォレストは，複数の機械学習モデルを組み合わせることでより強力なモデルを構築するアンサンブル学習の一種であり，決定木による複数の弱学習器を統合させることで汎化能力の向上を目指している．また，Fault-prone モジュール予測において高い精度を示すことが知られている [7].

2.2　MOA

Mahalanobis Outlier Analysis（MOA）とは，多変量データを対象とし，マハラノビス距離が閾値 θ_{MOA} を超える個体を外れ値と見なし除去する手法である．

マハラノビス距離とは，標本の重心から計算対象とする個体までの距離を，標本のばらつき度合いに基づいて算出される距離尺度である．マハラノビス距離の概要図を図 1 に示す．$\bar{x}$ を各変数の平均値のベクトル，x_i を個体 i の持つ変数のベクトル，n を個体の総数，T を転置ベクトルとしたとき，共分散行列 S は

$$S = (n-1)^{-1} \sum_{i=1}^{n} (x_i - \bar{x})(x_i - \bar{x})^T$$

と表され，個体 i のマハラノビス距離 MD_i は

$$MD_i = \sqrt{(x_i - \bar{x})^T S^{-1} (x_i - \bar{x})}$$

と表される．MD_i は平均が 1 になるように正規化され，個体 i が重心と同じ場所に位置する場合 MD_i は最小（$MD_i = 0$）となり，重心から離れた個体ほど大きな値をとる．本稿の実験における MOA の運用として，バグを含むモジュールの標本と含まない標本のそれぞれに対し MOA を独立に適用することで各標本の外れ値の除去を行う．

2.3 CC-MOA

Cross-Class Mahalanobis Outlier Analysis（CC-MOA）は，標本に 2 つのクラス（本稿ではバグあり/なし）が存在することを前提とした外れ値除去手法である．CC-MOA の概要図を図 2 に示す．CC-MOA では，あるクラスの個体（例: バグを含むモジュール）がもう一方のクラスの標本（例: バグを含まないモジュール）の重心に近い場合，つまり，他のクラスからのマハラノビス距離が閾値 θ_{CC-MOA} を下回る場合を外れ値と見なす．これにより，たとえばバグを含むにもかかわらず，バグを含まないモジュール群と似た特徴を持つような個体を外れ値として検出することが可能となる．

3 提案手法

本稿では新たな外れ値除去手法として，訓練データおよび予測対象のデータとは無関係の第三者データセットから fault-prone 予測モデルを構築し，訓練データ中のバグモジュールの予測を行うことで，予測が難しいモジュールを外れ値と見なし，訓練データから除去する方法を提案する．

提案手法はモジュール単位でのバージョン間予測への適用を前提とする．予測対象となるバージョンのデータセットと予測モデルを構築するバージョンのデータセット（D_{fit}）に加えて，外れ値除去用のモデルを構築するデータセット（D_{others}）を用意する．特定のプロジェクトで発生するバグに依存せず，なるべく一般的なバグの情報が多く含まれるよう，複数のプロジェクトから構成されるデータセットが望ましい．また，各モジュールの特徴値であるメトリクスの種類は全てのデータセットで共通している必要がある．

提案手法の手順を以下に示す．

Step.1 D_{others} を基に fault-prone モジュール予測モデルを構築する．

Step.2 構築したモデルを用いて D_{fit} の各モジュールのバグを予測したスコアをそれぞれ得る（スコアの値域は $[0, 1]$ であり，高いほどバグの可能性が高い）．

Step.3 独自に設定する判定基準により，D_{fit} のうち特にバグが疑われるモジュール群（M_D），および，特に安全と思われるモジュール群（M_S）を決定する．

Step.4 M_D のうち実際にはバグのないモジュール，および，M_S のうち実際にはバグのあるモジュールを D_{fit} から除去する．

Step.3 で使用する判定基準には，スコアが閾値以上/以下の場合や，スコアが全体の上位/下位 n% 以内の場合などが考えられる．本稿内では予備実験の結果効果の高かった判定基準として，スコアが上位 30%以内のものを M_D，下位 30%以内のものを M_S とする．

表 1　データセット

	訓練データ				テストデータ			
プロジェクト	ver.	総数	バグ有	混入率	ver.	総数	バグ有	混入率
ant	1.5	293	32	10.9%	1.6	350	92	26.3%
camel	1.4	856	144	16.8%	1.6	945	188	19.9%
forrest	0.7	29	5	17.2%	0.8	32	2	6.3%
ivy	1.4	241	16	6.6%	2.0	352	40	11.4%
jedit	4.2	367	48	13.1%	4.3	492	11	2.2%
log4j	1.1	109	37	33.9%	1.2	205	189	92.2%
lucene	2.2	247	144	58.3%	2.4	340	203	59.7%
poi	2.5	384	248	64.6%	3.0	441	281	63.7%
prop	4	8,702	840	9.7%	5	8,506	1,298	15.3%
synapse	1.0	157	16	10.2%	1.1	222	60	27.0%

4　評価実験

4.1　実験概要

提案手法が fault-prone モジュール予測の精度向上に及ぼす効果を評価するため，評価実験を行った．実験では，提案手法，MOA，CC-MOA を使用して訓練データから外れ値除去を行ったもの，および，無除去のものからそれぞれ fault-prone モジュール予測モデルを構築し，各モデルの予測精度を比較する．

外れ値除去の各手法および fault-prone 予測モデルの実装は著者により行われた．ランダムフォレストの実装には Python の機械学習パッケージである scikit-learn を使用した．決定木の本数や最大深さなどのハイパーパラメータには，全てデフォルトのものを使用した．

4.2　データセット

本実験で使用するデータセットの内容を表 1 に示す．データセットは 10 個のプロジェクトからなり，それぞれ 2 個のバージョンが抽出されている．本実験におけるバージョン間 Fault-prone モジュール予測において，古いバージョンを訓練データ，新しいバージョンをテストデータとして使用する．また予測対象とするプロジェクト以外の全てのプロジェクトの両バージョンを第三者データセットとして外れ値除去用のモデル構築に使用する．表には，各バージョンにおけるモジュールの総数，バグ有りモジュール数，およびバグ混入率を記載している．

本データセットは Zenodo データリポジトリ [8] から取得した．これらのデータセットは Jureczko らによって寄贈されたものであり [9] [10]，データ測定の詳細については [10] に記述されている．各モジュールはバグの個数を含む 21 種類のメトリクスを有している．

4.3　評価尺度

予測精度の評価尺度には AUC of ROC（以下 AUC）を用いる．AUC は，予測スコアからバグ有/バグ無を決定するための特定の閾値に依存しない評価尺度であり，真陽性エラー率と偽陽性エラー率の関係を評価するために閾値を動かして計算される．AUC の値域は [0, 1] を取り，値が大きいほど精度が高いことを表す．

本実験では機械学習モデルによる予測のぶれを考慮し，各プロジェクトに対して各手法（無除去を含む）で外れ値除去を行って構築したモデルによる fault-prone 予測を 100 回繰り返し，AUC の平均値を算出する．

表 2　AUC による精度比較

プロジェクト	無除去	MOA	CC-MOA	提案手法
ant	0.733	0.739	0.684	0.791
camel	0.669	0.667	0.618	0.638
forrest	0.910	0.912	0.898	0.886
ivy	0.741	0.772	0.693	0.798
jedit	0.664	0.656	0.688	0.665
log4j	0.534	0.533	0.612	0.588
lucene	0.631	0.634	0.632	0.687
poi	0.657	0.646	0.755	0.789
prop	0.601	0.605	0.616	0.550
synapse	0.647	0.689	0.631	0.701
平均	0.679	0.685	0.683	0.709
トップ数	1	1	3	5

表 3　外れ値除去後の訓練データ内モジュール数

	MOA		CC-MOA		提案手法	
プロジェクト	全て	バグ有	全て	バグ有	全て	バグ有
ant	271	32	191	10	211.4	25.6
camel	824	139	321	77	650.1	121.2
forrest	29	5	10	5	23.3	4.6
ivy	183	16	135	11	175.0	14.2
jedit	356	48	234	25	281.9	45.5
log4j	102	37	45	17	90.0	30.5
lucene	234	135	231	139	187.5	108.6
poi	363	234	38	24	295.9	189.5
prop	8,278	792	1,969	303	6,290.6	665.9
synapse	110	16	146	9	115.5	14.1

4.4　事前実験

既存手法である MOA および CC-MOA を使用するために，閾値 θ_{MOA} および θ_{CC-MOA} を設定する必要がある．CC-MOA の提案論文 [3] で閾値決定に用いられている方法に従い，各プロジェクトで適切な閾値を求めるため二分割交差検証による事前実験を行った．事前実験では各プロジェクトの訓練データをランダムに二等分割し，一方に外れ値除去を行い，もう一方に含まれるバグを予測する操作を，閾値を 0.1 から 10.0 まで 0.1 刻みに変化させ繰り返す．一つの閾値について操作を 10 回繰り返し，AUC の平均値が最も高かった閾値を本実験で用いる．

5　実験結果

各手法適用後の予測モデルの精度を AUC で比較した結果を表 2 に示す．AUC の値は 100 回分の平均である．全プロジェクトの AUC の平均値は，外れ値除去を行っていないものが 0.679 となり，MOA，CC-MOA，提案手法による外れ値除去を行ったものはそれぞれ 0.685，0.683，0.709 となった．この結果から，提案手法が平均的に最も高い精度向上を実現していることがわかる．また MOA と CC-MOA は．AUC で評価した場合でも無除去に比べて若干の改善が見られたものの，提案手法に

比べて顕著ではないといえる．また，プロジェクトごとに見た場合，提案手法は 10 件中 5 件で最も高い精度を実現しており，これは全ての手法の中で最も多い．ただし，全てのプロジェクトで提案手法が最も高い精度を実現できるわけではなく，プロジェクトによる差異が見られた．特に，camel，forrest，prop では，外れ値除去を行わなかった場合よりも精度が低くなっている．これらのプロジェクトに共通する明確な特徴は現状では見つかっていない．

加えて表 3 に，各手法による外れ値除去後の訓練データ中のモジュール数を示す．提案手法の除去数は試行ごとに異なる場合があり，表の数値は 100 回分の平均値である．本実験における全体的な傾向として，提案手法によって除去される外れ値の数は MOA より多く，CC-MOA より少ないことがわかる．MOA と CC-MOA の平均 AUC が同程度かつ提案手法より低いことから，提案手法は fault-prone モジュール予測に悪影響を与えている適切な量のモジュールを除去できていると考えられる．

6 おわりに

本研究では，fault-prone モジュール予測の精度向上を目的として，第三者データセットによる予測モデルを用いた外れ値除去手法の提案を行った．実験の結果，本提案手法は大多数のプロジェクトで予測精度を向上でき，また既存の外れ値除去手法よりも高い効果を持つことが確認できた．

今後の課題として，ランダムフォレストのハイパーパラメータの最適化を行った場合や，ランダムフォレスト以外のモデル構築手法を使用した場合の評価への影響の調査が挙げられる．また，マハラノビス距離を使用しない外れ値除去手法との比較や，別のデータセットを用いた評価も行いたい．さらに提案手法自体についても，判定基準の最適化や，効果的な第三者データセット数の分析等を行いたい．

謝辞 本研究の一部は JSPS 科研費 JP20K11749, JP20H05706 の助成を受けた．

参考文献

[1] Kamei, Yasutaka and Shihab, Emad.: Defect Prediction: Accomplishments and Future Challenges. Proc. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 33-45, 2016

[2] Jimenz-Marquez, S.A., Lacroix, C. and Thibault, J.: Statistical data validation methods for large cheese plant database. Journal of Dairy Science, Vol.85, No.9, pp. 2081-2097, 2022

[3] まつ本 真佑, 亀井 靖高, 門田 暁人, 松本, 健一: Fault-prone モジュール判別における外れ値除去法の比較, 情報処理学会論文誌, vol.49, No.3, pp. 1341-1351, 2008

[4] A. Monden, J. Keung, S. Morisaki, Y. Kamei, K. Matsumoto: A heuristic rule reduction approach to software fault-proneness prediction. Proc. Asia-Pacific Software Engineering Conference (APSEC2012), pp. 838-847, 2012.

[5] João Gama and Indrė Žliobaitė and Albert Bifet and Mykola Pechenizkiy and A. Bouchachia: A survey on concept drift adaptation. ACM Computing Surveys (CSUR), vol.46, No.4, pp. 1-37, 2014

[6] Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., and Zhang, G.: Learning under Concept Drift: A Review, IEEE Transactions on Knowledge and Data Engineering, Vol. 31, No. 12, pp. 2346–2363, 2019

[7] S. Lessmann, B. Baesens, C. Mues and S. Pietsch: Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transactions on Software Engineering, vol.34, no.4, pp. 485-496, 2008

[8] Zenodo, Large Defect Prediction Benchmark, `https://zenodo.org/record/6342328#.YtjDaHbP2Uk`

[9] M. Jureczko and L. Madeyski: Towards identifying software project clus-ters with regard to defect prediction. Proc. 6th International Conference on Predictive Models in Software Engineering (PROMISE ’10), pp.9:1–9:10, 2010

[10] M. Jureczko and D. Spinellis: Using object-oriented design metrics to predict software defects. In Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki WrocÅCawskiej, pp. 69–81, 2010.

1事例を通じてのストーリーポイントの有用性と見積もり誤差に対する考察

Discussion on the Usefulness and Estimation errors of Story Points through a Case Study

今井 健男 *

あらまし　ストーリーポイント（Story Points, SP）はスクラム開発によく用いられる作業規模の指標である．多くのスクラムプロジェクトでは，SP を使って開発の見積もりやベロシティ（開発の速さ）の計測が行われている．本稿では，あるスクラム開発プロジェクトで取り組んだ SP による 7ヶ月間の見積もりとベロシティ計測の取り組みについて述べ，その上で SP の有効性と，SP を使った見積もり誤差の分類について考察する．

Summary. A story point (SP) is a metric of the work size often used in Scrum development. Many Scrum projects use SPs to estimate workloads and measure velocity (speed of development). This paper describes a seven-month effort to estimate and measure velocity using SP in a Scrum development project. Then we discuss the effectiveness of SP and the types of errors in estimation using the metric.

1　はじめに

ソフトウェアの工数見積もりはソフトウェア工学において長年の課題となってきた．古くは SLOC（Software Lines of Code）やファンクションポイント法 [1] などが指標として提案され，様々な研究がなされたが，実際の開発現場には殆ど定着せず，アドホックな見積もりがなされることが大半であった．

他方，アジャイル開発において見積もりは独自の進化を遂げてきた．その 1 つがストーリーポイント（SP） [2] であり，現在アジャイル開発で主流となっているスクラム [3] とともに多くの開発現場で用いられている．

しかし，スクラムは経験主義を標榜していることなどから，それらを使用しての知見は特に国内のソフトウェア工学研究にあまり還元されてこなかった．

我々は，2021 年 12 月から 2022 年 6 月にかけて，あるソフトウェア開発プロジェクトに SP を導入し，見積もりやプロジェクトの進捗確認に用いる試みを行った．その結果を端的に述べれば，SP は非常に有用で，プロジェクトの進捗の可視化や開発計画立案が容易になった．一方，運用に際して特異な現象をいくつか確認でき，見積もりと実績の差分（見積もり誤差）について様々な知見を得た．

本稿では，我々のプロジェクトにおける 7ヶ月間の SP の利用を 1 つの事例として紹介する．その上で，SP を使った見積もりの誤差に関して考察を加える．

2　前提

本節では，以降の節で必要となる幾つかの概念を整理する．

2.1　スクラム開発

スクラムはアジャイル開発手法の 1 つで，国内では主に SaaS（Software as a Service）の開発現場で多く採用されている．

その詳細な説明は割愛するが，以降の議論に必要なものとして，スプリントと 2 つのバックログについての説明のみここで行う．

*Takeo Imai, Idein 株式会社

まずスプリントとは，スクラムでの開発期間の単位のことである．スクラムで通常は 1 〜 4 週間の決まった長さの期間を 1 スプリントとし，そのスプリントごとに，開発すべきゴールを定め，そのゴールに向かって開発を進め，終了のタイミングでレビューや振り返りをする．そのスプリントを繰り返しながら開発を進める．

その中で，開発要件はプロダクトバックログ（PBL）とスプリントバックログ（SBL）で管理される．

PBL は開発項目が優先順位順に並んだ順序つきリストである．PBL に並ぶ開発項目のことをプロダクトバックログアイテム（PBI）と呼ぶ．スクラムでは原則として，PBL が定める順に沿って PBI に着手し，開発していく．ただし，その順序の変更は適宜可能であり，それによって，顧客のニーズの変化など，プロジェクト内外の変化に常に適応した開発を進めることが可能となる．

スプリントの開始時に着手する PBI 群を決め，それらをより具体的な作業項目（タスク）に分割する．それら 1 スプリント分のタスクのリストが SBL である．

2.2　アジャイルでの見積もりの対象

ここでは，アジャイル開発とウォーターフォール型開発における見積もりの考え方の違いを，プロジェクト管理の「鉄の三角形」を用いて説明する．これには複数のバリエーションがあるが [4]，ここではコスト・期間・スコープ（対象となる開発項目）の 3 つから成る三角形（図 1）を使用する．

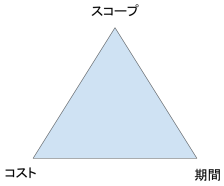


図 1　鉄の三角形

これら 3 つはそれぞれが互いにトレードオフ関係にあり，3 つ全てを事前に確定させることはできない．よって開発計画を立てる際，2 つを決めたらそれに合わせて残り 1 つを調整する必要がある．

ウォーターフォール型開発で計画を立てる際は，大抵は期間とスコープを決め，コスト（主に開発リソース）を調整する．従って主に見積もる対象は工数である．

一方，アジャイルではコストを固定する．その上で (a) 期間を決めてスコープを調整するか (b) スコープを決めて期間を調整するか，で計画する．

よって，アジャイルにおける見積もりの対象は，(a) の場合は開発規模，(b) の場合は開発期間となる．我々のプロジェクトは，本論文の対象期間は (a) で，四半期内にどれだけの規模を開発できるか見積もりながら開発を進めた．

ここで，その指標として我々が使用したのが後述のストーリーポイントである．

2.3　ストーリーポイント

ストーリーポイント (SP) [2] は元は eXtreme Programming [5] で用いられ始めた作業規模の指標で，スクラム開発でもよく流用され，主に PBI の開発規模の見積もりに使用される．この指標は以下のような特徴を持つ．

開発者の主観に基づく評価　対象の PBI が何ポイントかは，実際に開発に携わっている開発チームが主観に基づいて決める．

相対的な評価　値は絶対的な意味を持たず，他と比較して作業規模が何倍か，といった相対評価でポイントをつける．

チーム単位　スクラムは 1 つの PBI にチーム全員で取り組むため，SP もチーム単位で算定される．

SP の見積もり方法として，我々はプランニングポーカー [2] を用いた．これは各開発者が数字の書かれたカードのデッキを保持しておき，(1)PBI の開発規模に最も近いと直感的に思うカードを一斉に出し (2) 数値に差があれば議論し，再び一斉にカードを出して，以降 (1)〜(2) を全員が合意するまで繰り返す，というものである．

2.4 ベロシティと実績値

スクラム開発におけるベロシティとは，1 スプリントで完了した作業量，すなわち開発作業の速さを実測値で測定したものである．特に，1 スプリントで完了した PBI の全 SP 数の総和をそのスプリントのベロシティと計算する方法が多くのスクラム開発プロジェクトで用いられている．

開発予定の全 PBI の総見積もり値をベロシティで除算すると開発期間が見積もれる．あるいはベロシティを想定開発期間で乗算すると開発規模が見積もれる．

一般的には，スプリント中に完了しなかった PBI の実績 SP 数はゼロとカウントされることが多い（未完となった PBI はその時点で残り作業分のポイントを見積もりし直す）[6]．しかし，これでは未完の PBI のために費やされた作業がベロシティに反映されない．そこで我々は，ベロシティと，各 PBI の完了に要した作業の実績値を以下のように計測することにした．

あるスプリント s において，見積もり SP 数が SP_p である PBI p に対して発生した作業の実績値 $W_p|_s$ を以下のように定義する．

$$W_p|_s = SP_p \times \frac{s \text{ 期間中に完了した } p \text{ のタスク数}}{s \text{ 期間中に存在した } p \text{ の全タスク数}}$$

スプリント s のベロシティV_s は，全 PBI P での作業の実績値の総和で定義する．

$$V_s = \sum_{p \in P} W_p|_s$$

また，ある PBI p の完了にかかった実績値 W_p は全スプリント S での作業の実績値の総和で定義する．

$$W_p = \sum_{s \in S} W_p|_s$$

表 1 は以上の計算方法でベロシティと実績値を求めた例である．2 つの PBI I_1 と I_2 を連続する 3 スプリント S_1〜S_3 の間に開発している．

たとえば I_2 は S_2 で SP 数 3 と見積もられ，5 つのタスクに分解されてそのうち 2 つが完了した．そして S_3 では再び SP 数 3 と見積もられ，残り 2 つのタスクが全て完了した．スプリントを跨ぐ際，以前に見積もられた SP 数は後続のスプリントで見積もり直す際に一切考慮されないことに注意されたい．

ここで I_2 の実績値は $3 \times 2/5 = 1.2$ と $3 \times 2/2 = 3.0$ の和 4.2 となり，当初の見積もりより 1.2 増えたことになる．また S_2 のベロシティは 2.0 と 1.2 の和 3.2 となる．

表 1 ベロシティと実績値の例．**est.** はそのスプリントでの各 **PBI** の **SP** の見積もり，**ratio** の分母はそのスプリントで着手すべきタスク数，分子は完了したタスク数．**act.** は **ext.** と **ratio** の積から得られる，各スプリントでの各 **PBI** の実績値である．これらの **PBI** ごとの総和がその **PBI** の実績値に，スプリントごとの総和がベロシティとなる．

	S_1			S_2			S_3			実績値
	est.	ratio	act.	est.	ratio	act.	est.	ratio	act.	
I_1	5	3/5	3.0	2	2/2	2.0				5.0
I_2				3	2/5	1.2	3	3/3	3.0	4.2
ベロシティ			3.0			3.2			3.0	

3 プロジェクトにおける SP 導入とその利用

本節では，我々のスクラム開発プロジェクトの概要と SP を導入・運用した期間，及びその期間中のプロジェクトの推移について述べる．

3.1 プロジェクトの概要

本プロジェクトは，ある IoT サービスの開発プロジェクトである．このサービスは運用現場に設置されたエッジデバイスから周期的にデータを取得し，クラウドに

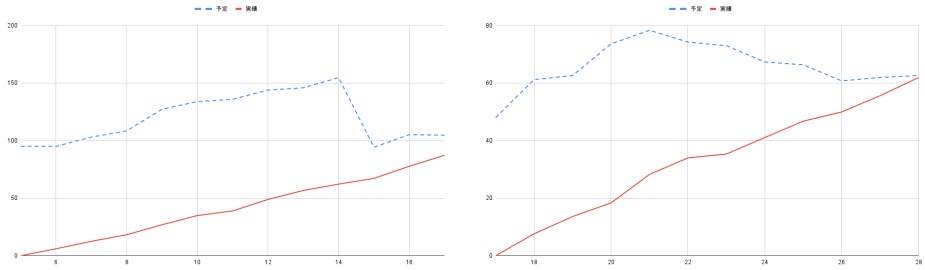


図 2　プロジェクトの進捗状況を表すバーンアップチャート．左は **1-3** 月期，右は **4-6** 月期のもの．いずれも横軸はスプリント番号，縦軸は **SP** 数を表す．青い破線が予定された **PBI** の総見積もり **SP** 数で，赤い実線が実績値の累積である．

送信する．クラウドでは，所定量のデータが収集されるとデータを自動処理し，結果をエッジデバイスにフィードバックする．

よって，このプロジェクトはエッジとクラウドの双方に開発要素がある．

本プロジェクトは 2021 年 4 月に開発が始まり，1 スプリントを 1 週間としてスクラムを適用した．だが，チームのスクラムへの習熟度等を勘案して当初は SP による見積もりを使用せず，必要な際には理想時間 [2] を用いた見積もりを行っていた．

そして，チームが十分に成熟したのを見て同年 12 月から SP を導入した．本稿の報告対象はそれ以降 2022 年 6 月までの 7ヶ月間，28 スプリントである．開発チームメンバーは当初 3 名で，以降同一のメンバーが継続して開発に携わっており，最終 2 スプリントのみ開発者が 1 名増えて 4 名になっている．

以降，プロジェクトの推移を時間の経過と共に見ていく．なお以降，開始から n 番めのスプリントを Sprint n と表記する．

3.2　12 月（Sprint 1-4）

2021 年 12 月から SP による見積もりを開始した．プロジェクト運営上機能するか見定めるため，最初の 4 スプリントは試験運用としてベロシティの計測のみ行った．

結果，この間のベロシティの平均 7.19, 分散 0.057 を計測した．例えばこのベロシティの平均と分散を使って四半期（12 スプリント分）の開発規模を見積もると $7.19 \times 12 = 86.28$ で，1 σ の誤差が約 2.86，約 3.3%に収まることになり，極めて正確な予測ができることになる．

よって，我々は以降 SP での見積もりをプロジェクト運営に使用することにした．

3.3　1-3 月（Sprint 5-16）

次の四半期では，3 月に概念実証（Proof of Concept, PoC）を控え，特に 2 月後半はその準備に追われた．そして 3 月は PoC と新規開発を並行して進めた．

結果として，この期間のベロシティは平均で 7.64，分散 5.11 と 12 月よりはばらつきが出た．PoC の準備や PoC 環境での運用，そしてその評価と報告書の執筆など，開発とは一概に言えない作業が発生し，それらを実施しているスプリントのベロシティは上昇した．例えば Sprint 11 のベロシティは 12.5 を記録している．

とはいえ図 2 (左) にあるように，全体では安定したベロシティを計測している．

一方で，期中の見積もり増大も大きく，Sprint 14 までに 62 ポイント消化したのに対し，見積もり数も約 60 ポイント増加した．そのため Sprint 14 で計画の大幅な見直しを行った．そのため図 2 (左) で見積もり SP 数が大きく下がっている．

3.4 4-6 月（Sprint 17-28）

4 月からの 4 スプリント（Sprint 17-20）は，3 月の PoC で見つかった不具合や追加の性能調査などの追加対応に追われた．これらは元々開発計画になかった措置で，そのため，この 4 スプリントの間は消化した SP（図 2 (右) の赤い実線）とほぼ同数の SP（青い破線）が追加の見積もり分として同時に積み上がっている．

この間はベロシティに特に変化はない（平均 7.06）が，Sprint 19 のみベロシティが 4.73 に落ち込んでいる．これまでクラウドの開発を重点的に行ってきたが，このスプリントのみエッジの開発をする必要があり，しかもこれまで社内でも経験のない開発に取り組む必要があったからと思われる．

そして 4 月末頃（Sprint 21）より，クラウド側のアーキテクチャの大幅な改修を行った．これは当初からの開発項目にあるものだったが，見積もった SP も当初から大幅に増え，同時にベロシティも下がった．具体的には，7ヶ月間全体でのベロシティの平均は 6.81，4 月の PoC 追加対応を行っていた間のベロシティの平均は 7.06 だったが，Sprint 21 から 6 スプリントのベロシティの平均は 4.56 であった．

その後，最後の 2 スプリントで開発者が 1 名加入し，ベロシティはやや回復している（平均 7.36）．これもあって 6 月中に改修作業を終了できた．

4 考察

4.1 SP の有用性

表 2 計測されたベロシティ

期間	平均	分散	1 σ
1-3 月期	7.64	5.11	29.6%
4-6 月期	5.86	5.02	38.2%
全 7ヶ月	6.71	4.96	33.2%

SP を導入して見積もりとベロシティ計測を行ったところ，想像以上にプロジェクトの進捗管理に役立った．計測できたベロシティは表 2 となった．多少の変動はあるが原因はかなり推測が容易なものが多かった．

そしてベロシティをベースラインとして開発規模が想定を超過する可能性もスプリント毎，つまり週毎に予測でき，対策を即座に打つことができるようになった．

4.2 見積もり誤差の分析

一方で，例えば 1〜2 月に，ベロシティは安定しているのに予定作業量が期間を通じて増大していくなど，SP 導入前には想定していなかった現象にも遭遇した．

この増大には予定外の PBI が追加された分もあるが，期初から予定していた PBI に大きな見積もり誤差，すなわち着手前に見積もった SP 数より実績値が増えてしまった分も多かった．1 月からの 6 スプリントの総実績値 37.9 のうち，見積もり誤差による増大分が 6.19 と全体の 16.3%を占めた．また 5-6 月には，実績値が見積もりより大きく増大し，かつベロシティも下がったため，各数値の見た目以上にプロジェクトへの影響が大きかった．

以上から，SP を用いた見積もり誤差には**作業量誤差**と**ベロシティ誤差**の 2 種類があり，それらの混合が最終的な誤差として現れるのでは，と我々は考えた．

4.2.1 作業量誤差

作業量誤差は，開発者が見積もり時に想定しない，あるいは軽視しがちな作業があり，その作業量を十分見積もれないことによって起こる誤差である．我々の 1-2 月に発生した見積もり誤差は主にこれに起因すると思われる．

この期間で特に誤差が大きかった PBI に対し，開発者に誤差の要因をインタビューしたところ，次のような原因だったとの回答があった．

- 修正が必要なコンポーネント数が多く，同様の（しかしそれぞれ微妙に異なる）修正を繰り返し適用する分の作業量を事前に見積もりきれなかった
- プルリクエスト上で修正内容に関して議論が発生し，それが長期化した
- 着手前に想定していない，タスク外の事象（システムテストの実施，チーム外の人間の作業，など）に対する待ちが発生した

SP の見積もりにおいては開発者は技術的な困難さを重視しがちである一方，単純作業や，コーディングやテストといった典型的開発作業に当てはまらない付加的な作業を見落としがちであることが回答から伺える．作業量誤差はこういった見積もり時の開発者の思考の傾向から生まれるのでは，と推測される．

4.2.2 ベロシティ誤差

ベロシティ誤差は，端的にいえば作業の困難さを見誤ることで発生する．

2-3 月に PoC 関連の作業を行ったが，運用上のパラメタ設定や報告書の作成など，想像より簡単な作業だったと思われ，それでベロシティが上昇したと思われる．

5-6 月においてはアーキテクチャの改修に取り組んだが，全体の設計を終えて開発に着手した後で，更に良いと思われる別のアーキテクチャがチーム内で提案され，議論の結果そちらに乗り換えた．

結果として，タスクを全面的に見直し，新アーキテクチャについて開発者全員が理解し直し，多くの議論をし，詳細部分の設計も見直す必要があった．

こうした経緯で，開発者が当初考えたよりも困難な作業をする必要があったのではないかと思われ，それによってベロシティが低下したものと考えられる．

4.3 見積もり誤差に対する対策

2 種類それぞれの誤差が発生する状況について，ある程度の傾向は事前に掴める可能性がある．今回の事例からは，次のような傾向が推定できた．

- 対象となる箇所が多い PBI ほど単純な作業量も増え，作業量誤差が起きやすい
- 作業のジャンルが変わると難易度が変わり，ベロシティ誤差が起きやすい
- 実際に着手し，ある程度設計してからでないと全体の作業内容が見えないような PBI では作業量誤差もベロシティ誤差も起きやすい

これらの傾向が事前に読めるのであれば，開発者が見積もった値をパターンに応じて重み付けし，補正することでより正確な見積もりが行える可能性がある．

5 おわりに

本稿では，あるスクラム開発プロジェクトにおけるストーリーポイント（SP）の導入と，それによる 7ヶ月間の見積もり及びベロシティ計測の事例を報告した．またその事例において，ベロシティと見積もり誤差の変化の具合が乖離している現象が散見されたことから，見積もり誤差を作業量誤差とベロシティ誤差の 2 種類に分け，それぞれの原因と対策について考察した．

SP は非常に有用であり，それゆえ誤差に対する精緻な対応も可能になると考えられる．2 種類の誤差を区別しながら各々の傾向を分析していくことで，より正確な見積もりができるのではないかと考えている．

謝辞

Idein 株式会社の関係者，特に日々の開発に勤しむ LP チームの諸氏に感謝する．

参考文献

[1] Allan J. Albrecht, Measuring Application Development Productivity, Proc. IBM Appplication Develop. Symp., p83, 1979.
[2] Mike Cohn, 安井 力（訳），角谷 信太郎（訳）, アジャイルな見積りと計画づくり ～価値あるソフトウェアを育てる概念と技法～（原書 : Agile Estimating and Planning）, マイナビ出版, 2009.
[3] ScrumGuides.org, The Scrum Guide, https://scrumguides.org/, 2022 年 7 月 21 日閲覧.
[4] Pollack, J., Helm, J. & Adler, D., What is the Iron Triangle, and How Has It Changed? International Journal of Managing Projects in Business, 11(2), 527-547, 2018.
[5] Kent Beck: Extreme Programming Explained: Embrace Change, Addison–Wesley. 2000.
[6] 吉羽 龍太郎, 着手したもののスプリント中に完成しなかったプロダクトバックログアイテムはどう扱えばいいですか？ - Ryuzee.com, https://www.ryuzee.com/faq/0028/, 2022 年 7 月 21 日閲覧.

キーストロークとマウス操作に基づくプログラミング能力の分析

Analysis of Programming Ability Based on Keystrokes and Mouse Operations

松本 和樹 * **西浦 生成** † **笹倉 万里子** ‡ **門田 暁人** §

あらまし プログラミング能力は個人差が大きく，その定量的な評価が求められている．本稿では，被験者実験によりプログラミング時のキーストロークとマウス操作を計測し，プログラミング能力との関係を定量的に分析する．実験では，14 名の被験者にプログラムを作成してもらい，キー入力，マウスの使用率，打鍵速度，Web における検索行動等について分析した．その結果，プログラムの動作確認のためのキー入力を多く行っていた場合やアルファベットの打鍵速度が速い場合にプログラミング能力が高いこと，またW eb 上で C 言語の標準関数などを検索する時間が多い人はプログラミング能力が低いこと等が示された．

1 はじめに

プログラミング能力は，個人差が極めて大きいとされている [1]．一方で，多くの企業では能力の高い人を採用したり，能力の程度によって開発者に仕事を割り振りたいという考えがある．そのため，プログラミング能力を定量的に評価することが求められている．従来，プログラミング能力の一端を定量化する手段として，キーストロークに着目した方法が研究されている [2] [4]．従来研究では，特定のキー入力における打鍵速度が速い人ほどプログラミング能力が高いことが示されている．一方で，キーストロークにおける打鍵速度のみがプログラミング能力と結びつくとは考えにくく，マウス操作やショートカットキーの使用，Web からのプログラミング関連技術の検索方法などがプログラミング能力に影響している可能性がある．

本研究では，14 名の被験者を対象とし，キーストロークに加えてマウス操作を計測するとともに，Web ブラウザ閲覧行動についても計測し，プログラミング能力との関係を実験的に分析する．本稿におけるプログラミング能力は，プログラミングに要した時間とプログラムの完成度から定量化する．また，実験では，ショートカットキーの使用率が高い人はプログラミング能力が高い，Web で標準ライブラリ関数の仕様方法などを検索する時間が多い人はプログラミング能力が低い，といった仮説を立て，その検証を行う．

2 従来研究

Richard らによる被験者実験 [2] は，プログラミングタスク実施中の打鍵速度とプログラミング能力（パフォーマンス）の間に関係があるかどうかを検証することを目的として行われた．この実験におけるプログラミングタスクは，新規のプログラム作成ではなく，与えられたプログラムに対して 3 つの変更を加えるものであり，各変更について，動作の正常性の観点から 0〜4 点の評価点が与えられ，その合計点をプログラミング能力としている．打鍵速度の分析にあたっては，キーストロークを「アルファベット」「数値」「コントロールキー」「ブラウジング（カーソル移動）キー」「その他」に分類している．

*Kazuki Matsumoto, 岡山大学大学院自然科学研究科

†Kinari Nishiura, 岡山大学大学院自然科学研究科

‡Mariko Sasakura, 岡山大学大学院自然科学研究科

§Akito Monden, 岡山大学大学院自然科学研究科

分析の結果，異なるタイプ間（ブラウジングキーを除く）を遷移する際の打鍵間隔が短い（打鍵速度が速い）人ほどプログラミング能力が高いことが示され，この打鍵速度とプログラミング能力（評価点）との相関係数は-0.516 であった．さらに，アルファベットまたは数値の文字列入力は，特別な構文 (() [ ] = ; などの使用) の入力時よりも高速であることが示された．このことは，心理学の観点から，意味のない文章は通常の文章よりもゆっくりと入力されるという単語の開始効果 [3] に反しない．

Richard らの追実験として，Leinonen ら [4] によってより長期にわたる実験が行われ，やはり異なるタイプ間の遷移時の打鍵速度とプログラミング能力には関連があり，その相関係数は-0.276 であったことが報告されている．

本稿では，これらの実験を踏まえ，キーストロークに加えてマウス操作を計測するとともに，Web ブラウザ閲覧行動についても計測し，プログラミング能力との関係を実験的に分析する．また，本稿では，プログラミングタスクをプログラム変更作業に限定するのではなく，一からプログラムを作成するタスクを対象とした実験を行う．

3　プログラミング行動に関する仮説と入力操作の分類

3.1　仮説

プログラミング能力の高い人は，作業を効率よく行うため，打鍵速度が速くショートカットキーを多用することや，一度ホームポジションから手を離すとそれだけ多くの時間が必要となるためマウスの使用率が少ない可能性がある．また，プログラミング能力が低い人ほど，プログラミング言語の基本的な構文などを記憶しておらず Web での検索が多くなる可能性がある．これらを考慮し，以下の仮説を設ける．

仮説 1　ショートカットキーの使用率が高い人は，プログラミング能力が高い．
仮説 2　マウスの使用率が低い人は，プログラミング能力が高い．
仮説 3　打鍵速度が速い人は，プログラミング能力が高い．
仮説 4　Web で検索する時間が長い人は，プログラミング能力が低い．

3.2　入力操作の分類

従来研究 [2] [4] を参考にし，各キーへのストロークとマウスクリックを次のいずれかの操作タイプに割り当てて分析を行う．

- アルファベット文字キー（例: a，L）
- 数字キー（例: 0，9）
- ショートカットキー（例: control，alt）
- ブラウジング（カーソル移動）キー（例: 矢印キー，HOME）
- その他のキー (例: enter, backspace, space, =, ;)
- マウスクリック

4　実験

4.1　実験の概要

本実験では，被験者に仕様書を与えて C 言語でプログラムを作成してもらい，その間のキーとマウスの入力を記録する．キーとマウス入力の記録のためのツールとして，TaskPit [6] とマウスレコーダー [7] を使用する．

プログラミング課題は,「ヒットアンドブロー」と呼ばれる数字を当てるゲームの実装を採用した．従来研究 [2] では特定のプログラムの修正作業が課題として与えられていたが，本実験ではプログラムをゼロベースで作成してもらうことで，簡単な設計やデバッグ作業も含めたプログラミング能力の分析を行う．

プログラム作成の制限時間は 2 時間とした．コーディングスタイルは問わず，関数名，変数名やコメント文にも制限は設けない．

プログラミング能力の評価にプログラム作成時間を含めているため，プログラムの可読性や変更容易性を向上させるよりも，できる限り早くプログラムを完成させることを優先するようにあらかじめ伝えた．また，C 言語の理解度による差を計測できるようにするため，C 言語の標準関数などを Web で検索することを許可し，その際のキーストロークも記録対象としている．プログラミング課題の仕様書は pdf ファイルとして被験者に配り，紙での配布は行っていない．また，プログラミングのためのエディタは Visual Studio Code に，ディスプレイは 1 台に統一した．

被験者は，岡山大学の工学部情報系学科の 4 年生が 5 名，大学院電子情報システム工学専攻の学生が 7 名，自然科学研究科の教員が 2 名の合計 14 名である．

4.2 プログラミング能力の評価

プログラミング能力は，プログラミングに要した時間とプログラミングスコアの２つを加味したトータルスコアで評価する．プログラミングスコアは，プログラムの完成度の評価点であり，プログラムの完成に必要な 15 個の項目を定め，実際にプログラムを動作させてバグの有無も含めて各項目を評価し，正常に動いた場合は 1 点，動かなかった場合は 0 点として合計 15 点満点とした．プログラミングスコアに関する 15 個の項目は以下の通りである．

1. 乱数シード入力を受け付ける．
2. マイナスの乱数シード入力に対するエラー処理が正しく行われる．
3. 1 万以上の乱数シード入力に対するエラー処理が正しく行われる．
4. 乱数の生成が行われる．
5. 生成された乱数が正しい条件となっている．
6. 乱数シード入力時の再入力受付が正しく行われる．
7. 質問入力を受け付ける．
8. 質問回数が 5 回で正しく終了している．
9. 正しい正誤判定が行われる．
10. 4 桁以外入力に対するエラー処理が正しく行われる．
11. 不正な 4 桁入力に対するエラー処理が正しく行われる．
12. 質問入力時の再入力受付が正しく行われる．
13. 勝敗，解の表示が正しく行われる．
14. 正答時の処理が正しく行われる．
15. オーバーフロー処理が正しく行われる．

トータルスコアは以下の計算式で求めた．

$$\text{トータルスコア} = PS \times (1 + (1 - \frac{T_p}{T_l})) \tag{1}$$

- PS: プログラミングスコア $(0 \leq x \leq 15)$
- T_p: プログラミングに要した時間
- T_l: 制限時間

5 結果と考察

5.1 各操作タイプの使用率に関する分析

トータルスコアと各操作タイプの使用率との関係を表 1 に示す．ここで，使用率とは，本実験中の全入力回数の中で各操作タイプの入力回数が占める割合のことである．まず，数字キーの使用率とトータルスコアとの間に有意な正の相関がみられ，相関係数は 0.649 と中程度の強度であった．数字キー使用率とトータルスコアの散布図を図 1 に示す．図からも正の相関があることが見て取れる．実験課題は乱数によって決められた数字を当てるものであるため，プログラムの動作確認のためには多数の数値入力が必要となる．そのため，テスト作業（動作確認）を多く行っていた被験者のトータルスコアが高くなったと考えられる．なお，テスト作業は，プログラムの完成後に行うとは限らず，作成中に行われる場合も少なくなかった．プログ

表 1　各操作タイプの使用率とトータルスコアの関係

操作タイプ	平均使用率	トータルスコアとの相関係数	p 値
アルファベットキー	0.225	-0.168	0.566
数字キー	0.101	0.649	0.012
ショートカットキー	0.022	0.225	0.439
ブラウジングキー	0.217	-0.020	0.946
その他のキー	0.328	-0.193	0.509
マウスクリック	0.112	-0.415	0.140

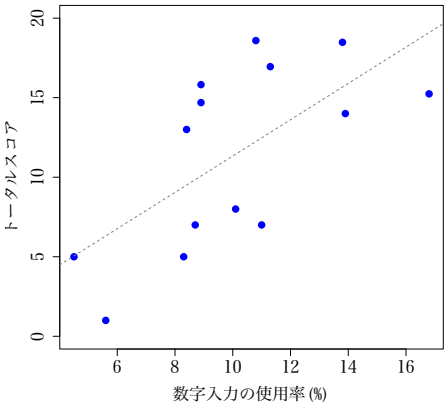


図 1　数字キーの使用率とトータルスコアの散布図

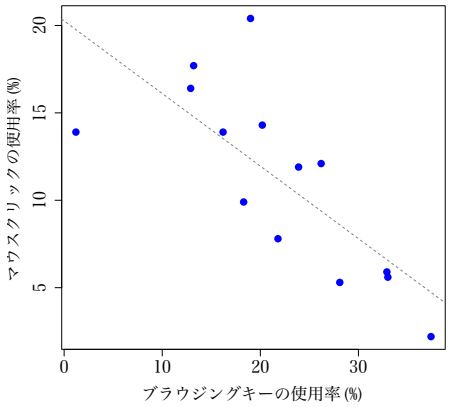


図 2　ブライジングキーの使用率とマウスクリックの使用率の散布図

ラムの作成中，完成後を問わず，頻繁にテストを行うことが高いスコアにつながったと考えられる．

ショートカットキーに関しては，使用率とトータルスコアの間に有意な相関がみられず，仮説 1 は支持されなかった．ただし，本実験ではキーボードとエディタは指定のものを使う必要があり，各被験者の使い慣れた環境ではないためショートカットを使いづらかった可能性がある．

マウスクリックの使用率に関しても有意な相関はみられず，仮説 2 は支持されなかった．ただし，相関係数は -0.415 とその絶対値は小さくなく，今後，より詳細な分析を行うことで，プログラミング能力との何らかの関係を見いだせる可能性がある．補足の分析として，ブラウジングキーの使用率とマウスクリックの使用率の関係を図 2 に示す．相関係数は -0.74，p 値は 0.002 であり，マウスクリックとブラウジングキーの使用率の間にやや強い負の相関がみられる．このことから，カーソル移動にマウスを主に使う人はブラウジングキーをあまり使わず，逆にブラウジングキーを主に使う人はマウスをあまり使わないと考えられる．

5.2　その他の分析

打鍵の間隔が 1.0 秒以内のアルファベットの打鍵を対象とし，打鍵速度とトータルスコアの関係を分析した．アルファベットのみを対象とした理由は，全操作タイプを対象とするとキーを押しっぱなしでカーソル移動するなど，打鍵速度と関係のな

表 2　打鍵速度とトータルスコアの関係

	打鍵速度の平均	トータルスコアとの相関係数	p 値
打鍵速度 (外れ値有り)	4.70	0.191	0.514
打鍵速度 (外れ値無し)	4.61	0.527	0.064

い入力まで対象となるためである．相関係数と p 値を表 2 に示し，打鍵速度とトータルスコアの関係を図 3 と図 4 に示す．ここで，測定値には 1 つの外れ値が存在していたため，外れ値を含めた場合と除去した場合のそれぞれについて相関を分析した．その結果，外れ値を除外しない場合は有意な相関が見られなかったが，外れ値を除去した場合に打鍵速度とトータルスコアとの間で有意な負の相関がみられた．以上のことから，仮説 3 の打鍵速度とプログラミング能力の関係はおおむね支持されるものの，打鍵が非常に速いにも関わらずプログラミングスコアが低いという人も中には存在するといえる．

また，プログラム作成中にブラウザを開いていた時間の割合とトータルスコアとの関係を分析した結果，有意な負の相関がみられ，相関係数は −0.48，p 値は 0.08 であった．ブラウザ使用率とトータルスコアの関係を図 5 に示す．値のばらつきが大きく相関の強さも中程度であるが，仮説 4 について，Web で C 言語の標準関数などを検索する時間が多い人はプログラミング能力が低い傾向にあることが示唆される．これは，本実験のタスクが，scanf や printf などの基本的な標準関数のみで遂行可能であり，プログラミング能力の高い者はそれら関数に熟知しており検索の必要性が少なかったことが考えられる．

従来研究 [2] において関係が確認された異なるタイプ間（ブラウジングキーを除く）を遷移する際の打鍵速度とトータルスコアとの関係についても分析を行った．その結果，本実験においても有意な負の相関がみられ，相関係数は −0.46，p 値は 0.09 であった．従来研究ではプログラムの修正タスクを対象としていたが，本研究におけるプログラム作成タスクにおいても同様の結果が得られたことから，この結果は一般性が高いと考えられる．

6　おわりに

本研究では，プログラミング能力を定量的に計測する一助とするために，キーストロークとマウス操作とプログラミング能力との関係を分析するための被験者実験を行った．実験の結果，プログラムの動作確認を多く行っていた人やアルファベットの打鍵速度が速い人はプログラミング能力が高いこと，また web で C 言語の標準関数などを検索する時間が多い人はプログラミング能力が低いことなどが示された．その一方で，マウスやショートカットキーの使用率とプログラミング能力との間には関係が見られなかった．

今後は，また，将来的には，プログラミング課題を与えるのではなく，プログラミングを含む日常業務におけるキー入力と能力との関係を明らかにしていきたい．

謝辞　本研究の一部は JSPS 科研費 JP20H05706 の助成を受けた．

参考文献

[1] Tom Demarco and Timothy Lister, "Peopleware: Productive Projects and Teams (3rd ed.)," Addison-Wesley Professional, 2013.
[2] Richard C. Thomas, Amela Karahasanovic and Gregor E. Kennedy,"An Investigation into Keystroke Latency Metrics as an Indicator of Programming Performance," Proceedings of the 7th Australasian conference on Computing education ,2005.
[3] Timothy A. Salthouse, "Perceptual, Cognitive, and Motoric Aspects of Transcription Typ-

ing," Psychological Bulletin 9(3), Pages 303-319, 1986
[4] Juho Leinonen, Krista Longi , Arto Klami and Arto Vihavainen, "Automatic Inference of Programming Performance and Experience from Typing Patterns," Proceedings of the 47th ACM Technical Symposium on Computing Science , Pages 132-137, 2016.
[5] Allen Newell, "Unified Theories of Cognition," Harvard University Press Cambridge, 1994
[6] 門田暁人, 上野秀剛, "開発行動記録システム TaskPit," http://www.taskpit.jpn.org/, 最終参照日 2022-6-26
[7] エムティ・ソフト, "マウスレコーダー　マウスとキーボード操作を記録／再生," http://mt-soft.sakura.ne.jp/web_dl/petit_tool/mrecorder/, 最終参照日 2022-6-26

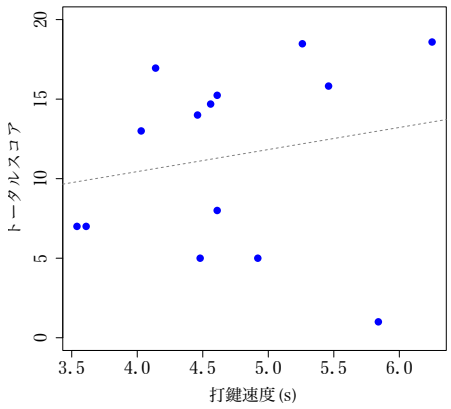


図 3　打鍵速度とトータルスコアの散布図 (外れ値有り)

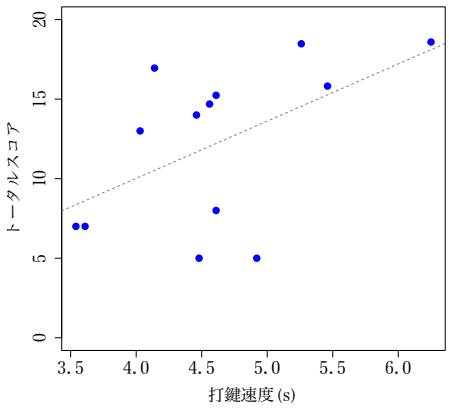


図 4　打鍵速度とトータルスコアの散布図 (外れ値を除外)

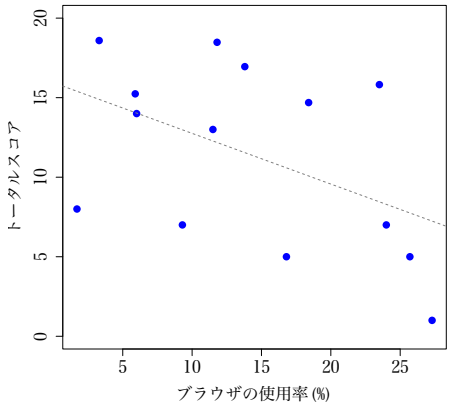


図 5　ブラウザの使用率とトータルスコアの散布図

コードクローン検出に基づく IoT を対象とした自動パッチ生成

Automatic Patch Generation for IoT Based on Code Clone Detection

大野 堅太郎 * 吉田 則裕 † 朱 文青 ‡ 高田 広章 §

あらまし IoT は軽量のプラットフォームやネットワークプロトコルがよく使用され，インターネットを介してサービスを提供するため，セキュアな設計が求められる．ソフトウェアシステムを対象とした研究において，68%の欠陥は，複数箇所を同一または類似の変更により修正をしたという報告がある．このような修正は，1 箇所の変更を行い，その変更を複製し，編集することで実現される．そのため，修正を自動化することができる可能性があると考えられる．本研究では，セキュアな設計が必要な IoT における，同一または類似の変更により修正された欠陥を対象として，自動パッチ生成ができるか調査を行った．その結果，コードクローン検出された IoT の欠陥事例に対して 96%の自動パッチ生成に成功した．

1 はじめに

Internet of Things (IoT) は，標準の通信プロトコルを使用して，デバイスをインターネットに相互接続することで，複雑なネットワーク構築を行う [1] [2]．相互接続されたデバイスは，デジタル世界での物理的または仮想的な表現，検知/作動機能，プログラム可能な特徴を備えており，一意に識別が可能である．IoT 接続デバイスの数は 5 年ごとに倍増しており [3]，2025 年までに世界中で 750 億を超える IoT 接続デバイスが存在すると推定されている [2] [4]．

IoT に接続されたエッジデバイスの計算リソースは限られており，軽量のプラットフォームやネットワークプロトコルがよく使用される．軽量のプラットフォームやネットワークプロトコルは攻撃に対する耐性が低く，開発者はセキュアなソフトウェア開発が求められる．また，IoT はインターネットを介してサービスを提供するため，従来のシステムと比べて複雑性が増しており [5]，システムに発生する脆弱性などの欠陥を含む可能性が高いといわれている [6]．そのため，IoT におけるソフトウェア上に含まれる欠陥を容易かつ早急に検出・修正することが求められる．

現在までに，コードクローン検出を用いて欠陥検出を行う開発が進められている [7] [8] [9] [10]．たとえば，Kim らはスケーラビリティを持ち，正確なコードクローン検出へのアプローチである VUDDY [8] を提案した．これは，関数に堅牢な解析と新しいフィンガープリントメカニズムを採用している．Gao らは，IoT 関連の欠陥検出として IoTSeeker [10] を提案した．これは，バイナリコードからラベル付きのセマンティックフローグラフを作成し，セマンティック特徴をベクトルとしたニューラルネットワークモデルを構築することで実現する．加えて，ソフトウェアプログラム解析におけるコミュニティでは，自動プログラム修正として，対象となる欠陥が含まれるプログラムに対して，制約条件を満たし，テストが成功するようにプログラムを変異させるアプローチで研究が進められてきた．これらのアプローチは一定の有効性を示す一方で，修正可能な欠陥の数，パラメータの調整，複数行の修正，テストケースの数における課題を持つ．

本論文では，自動プログラム修正の従来とは異なるアプローチとして，コードク

*Kentaro Ohno, 名古屋大学

†Norihriro Yoshida, 立命館大学

‡Wenqing Zhu, 名古屋大学

§Hiroaki Takada, 名古屋大学

ローン検出に基づく自動パッチ生成を提案する．この提案手法は，コードクローン検出による欠陥箇所の特定とプログラム修正となるパッチの生成を自動的に行う．本研究では，容易かつ早急な対応が求められる IoT の欠陥に適用することで提案手法の有効性の評価を行った．

本研究の貢献は，以下の 2 点である．

- 様々なプログラム構造に一般化した自動パッチ生成手法を提案した．
- IoT の欠陥事例を対象とした適用調査を行い，96%の事例で自動パッチ生成を実現し，高い有効性を示した．

2 章では本研究で使用するツールと関連研究の紹介を行う．3 章では提案手法，4 章では IoT の欠陥を対象とした提案手法の評価を説明する．5 章では評価の妥当性の検討を，6 章では本研究のまとめを記す．

2 関連研究

2 章では，本研究で使用するツールとしてコードクローン検出ツール CCFinderSW [11] と編集スクリプト生成ツール GumTree [12] の紹介と，本手法の可能性を示した研究として Madeiral らの研究 [13] の紹介を行う．また，コードクローンの同時修正支援手法と，コード修正事例を用いた自動プログラム修正手法との比較を行う．

本研究では，Zeller [14] が定義しているように，欠陥を不正なプログラムコードとして定義する．

2.1 コードクローン検出ツール CCFinderSW

CCFinderSW [11] は，コードクローン検出ツールの 1 つで，構文解析器生成系の 1 つである ANTLR の構文定義記述を入力として与えることで，さまざまなプログラミング言語で記述されたプログラムからコードクローンを検出することができる．CCFinderSW は，字句解析，変換処理，およびコードクローン検出/出力整形の 3 つのステップでコードクローン検出を実行する．

2.2 編集スクリプト生成ツール GumTree

GumTree [12] は，開発者の意図に近い編集スクリプトを計算することを目的とした，移動，追加，削除を含む抽象構文木に基づく編集スクリプトを計算するアルゴリズムを導入することにより，抽象構文間の差分を計算するツールである．行の単位の編集に基づく編集スクリプトを計算するのではなく，移動，追加，削除を含む抽象構文ツリーの粒度で計算することで，編集スクリプトから構文の変更を推測することが容易となる．本研究では，コードクローンに対して，GumTree を適用し，テキストベースの出力ファイルから，追加，削除となっているトークンと関数呼び出しの順序が異なるコード片を抽出し，パッチ生成に適用する．

2.3 Madeiral らの研究

Madeiral らの研究 [13] は，欠陥の修正において，コードクローンがどの程度編集されているかを明らかにした．具体的には，96 プロジェクトに適用された 3,049 個のパッチを手作業で分析した結果，コードクローンに対する修正はパッチにおいて頻出することがわかった．コードクローンに対する修正はパッチの 68%に存在し，それらの 70%はコードクローンに対する変更のみで構成されている．さらに，修正のコードクローンのみの変更で構成されたパッチの 89%には，同一の変更を加えた修正のコードクローンが含まれていることを明らかにした．一方で，Madeiral らの研究は，複数箇所に対して類似した修正を行うケースは自動化できる可能性が高いことを示唆しているが，自動化のための手法の提案には至っていない．この理由として，Madeiral らは，様々なプログラム構造に一般化できる自動プログラム修正において考えられる課題と解決策が未検討である点を挙げている．様々なプログラム

構造に一般化した自動プログラム修正の 1 つの解決策として，本手法を提案する．

2.4 コードクローン修正支援手法

コード片に欠陥がある場合，そのコードクローンは同時に修正する必要があることが多く，コードクローンの同時修正支援に関する研究が報告されている [15] [16] [17]．[15] は，コードクローン検出ツールで出力されたコードクローンを自動的に追跡し，コードクローンが変更されたことを開発者に通知することで欠陥修正を助ける．しかし，[15] はコードクローンの変更を通知するのみで自動的に欠陥を修正しない．我々の提案手法は，1 つのコード片を修正したパッチからコードクローンのパッチを自動的に生成することで欠陥を修正することができる．[16] は，統合開発環境の拡張モジュールとして提案された手法で，コードクローンを可視化することで，コードクローンを 1 つのコード片として視覚的に認識したまま修正が可能である．コードクローンを同時に修正することも可能と言及されており，自動的に欠陥を修正することが可能である．一方，[16] はコピーアンドペーストの際にそのコード片をコードクローンとして認識するため，既に開発が進んでいるプロジェクトに対して，過去にコピーアンドペーストされたコードクローンの追跡が難しい．我々の提案手法は修正の際に，コードクローンを精査するので，プロジェクトの途中であっても欠陥修正が可能である．[17] は Emacs 上の機能として提案された手法で，[16] と同様に視覚的にコードクローンを追跡可能である．しかし，[17] はコード片をリンクさせることでコードクローンを追跡できるが，リンクする際のコードクローンの判定を開発者の判断に依存しているため，開発者のコードクローンの判定の方法に対して検討が必要である．我々の提案手法はコードクローンの判定として CCFinderSW の検出結果を利用しているため，一意的にコードクローンを認識することが可能である．

また，統合開発環境 Eclipse のプラグインである LAC [18] は，コードクローンに対する編集操作を Eclipse 上で特定し，その編集操作を他のコードクローンへ適用する．本研究は，コード片に対して適用されたパッチから変更内容を特定し，特定した変更内容をそのコード片のコードクローンに対して適用しており，適用場面が異なると考える．LAC は単一の開発者が行うコードクローンに対する編集を効率化することができるが，本研究は複数の開発者が行うコードクローンの編集を効率化できる可能性がある．

2.5 コード修正事例からの学習や学習したパターンの適用

複数のコード修正事例から，修正パターンの学習 [19] を行う，もしくは学習した修正パターンに基づき他のコード片の修正を行う手法 [20] [21] [22] [23] が提案されている．本研究と同様に，単一の修正事例を基に他のコード片の修正を行うツールとして GenPat [21] が挙げれれる．GenPat は修正事例を収集したコーパスを用いることで，単一のコード修正事例を修正パターンに一般化し，その修正パターンを他のコード片に適用する．GenPat は，本研究が生成するパッチと同様のコード変換を提示できる可能性があるため，今後比較実験を行う必要がある．GenPat は本研究と異なりコードクローン検出に基づきコード変換を行わないため，非コードクローン（例：for 文の条件式のみ一致するコード片）に対するコード修正事例を基に対象のコード片を修正（例：条件式を修正）できる可能性があるが，変換対象のコード片全体（例：for 文全体）が一致しているかどうかは考慮しない．そのため，不適切なコード変換を行う可能性がある．一方，本研究はコードクローン（例：for 文全体が一致するコード片）に対してパッチを生成するため，非コードクローンへのコード変換を行う GenPat よりは，適用対象となるコード片は少ないものの信頼性の高いコード変換を行うことができている可能性が考えられる．また，過去の修正履歴や動的解析に基づき複数箇所を修正する自動バグ修正ツール Hercules [24] も提案されている．提案手法と Hercules についても性能の比較を行う必要がある．

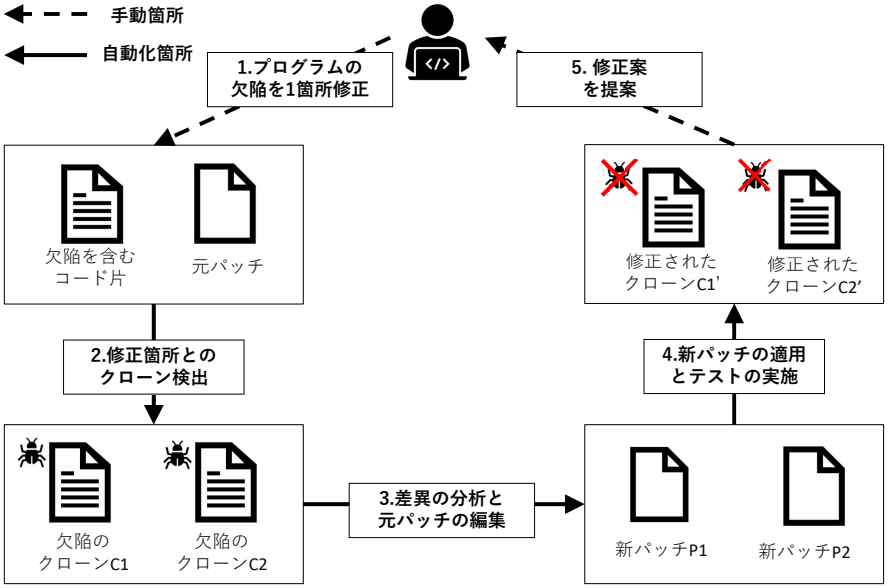


図 1　提案手法の概要図

3　提案手法

3 章では，本手法のアプローチを記す．多くの欠陥は複数箇所にコードクローンとして存在し，開発者はそのすべての欠陥を修正する必要がある．そのため，開発者はソースコード上に含まれる 1 つの欠陥を修正した後，対象としているプログラムに含まれる修正した欠陥に類似したコードクローンを検出し，検出された欠陥のコードクローンに対して修正を行うことが求められる．提案手法は以上の修正の流れを自動化した手法である．そのため，この手法は開発者が 1 つの欠陥を修正したことを前提としている．本手法の概要図を図 1 に示す．図 1 において，実線部が自動化箇所である．

本手法は，3 ステップの処理に分かれている．第 1 ステップでは，コードクローン検出ツールを用いて，開発者が修正した欠陥とのコードクローンを検出する．

Step 1-1　パッチファイルから,「ソースファイル名」,「修正するソースコード行数」を抽出する．

Step 1-2　対象ソースコードをコードクローン検出ツールにかけることで対象ソースコード内のコードクローンを検出する．

Step 1-3　コードクローンの検出結果から，パッチファイルで抽出した「ソースファイル名」における「修正するソースコード行数」の箇所とコードクローンになっている,「欠陥のコードクローン」を特定する．

Step 1-4　特定された「欠陥のコードクローン」に該当する「ソースファイル名」,「ソースコードの行数」を得る．

Step 1-5　「ソースファイル名」と「ソースコードの行数」から，検出された「欠陥コードクローンのコード片」を抽出する．

欠陥のコードクローンが検出されなかった場合，第 2 ステップに進まず，第 1 ステップで終了する．

第 2 ステップでは，元の欠陥と欠陥のコードクローンを比較し，異なる箇所を入れ替えることで，開発者が発見した元の欠陥の修正内容が記述されたパッチ (以下,元パッチと呼ぶ) を編集し，欠陥のコードクローンを修正した内容が記述されたパッ

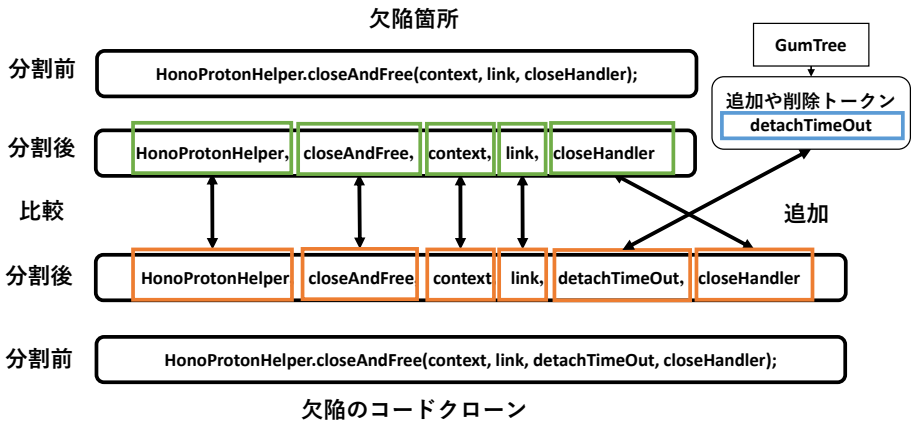


図 2　追加・削除を含む場合における比較例

チ (以下，新パッチと呼ぶ) を生成する．以下のような 3 つの条件分岐とそのステップを踏むことで，元の欠陥と欠陥のコードクローンを比較し，元パッチを編集する．

条件分岐 1　関数呼び出しの順序は異なるか
条件分岐 2　挿入トークンや削除トークンはあるか
条件分岐 3　対応するトークンは異なるか

Step 2-1　パッチファイルから,「修正前のコード片」,「修正後のコード片」を抽出する．追加のみの修正の場合は,「修正後のコード片」のみ抽出する．

Step 2-2　第 1 ステップで抽出した「欠陥コードクローンのコード片」と，パッチファイルから抽出した「修正前のコード片」を編集スクリプト生成ツールにかけることで,「追加・削除されているトークン」と「関数呼び出しの順序が異なるプログラム」を抽出する．追加のみの修正の場合は，第 1 ステップで抽出した欠陥コードクローンを含むスコープ単位で行う．

Step 2-3　「関数呼び出しの順序が異なるプログラム」がある場合,「欠陥コードクローンのコード片」の順序を「修正前のコード片」の順序に入れ替えておく．

Step 2-4　第 1 ステップで抽出した「欠陥コードクローンのコード片」とパッチファイルから抽出した「修正前のコード片」をトークン単位に分割する．

Step 2-5　「修正後のコード片」を複製する．

Step 2-6　分割したトークン同士で順番に比較していく．異なるトークンの場合,「複製した修正後のコード片」のトークンを入れ替える．このとき,「追加・削除のトークン」がある場合は比較せず，比較が終わった後「複製した修正後のコード片」に「追加・削除のトークン」を反映させる．

Step 2-7　「関数呼び出しの順序が異なるプログラム」が「複製した修正後のコード片」に含まれる場合，元の「欠陥コードクローンのコード片」での順序に入れ替えておく．

Step 2-8　ファイル名や変更前後の行数を記述して,「欠陥のコードクローン」の特徴を反映させたパッチファイルを生成する．

図 2 に追加・削除を含む場合における修正前の欠陥と欠陥コードクローンの比較例を，図 3 にトークンの入れ替えが発生する場合における修正前の欠陥と欠陥コードクローンの比較例を示す．

第 3 ステップでは，第 2 ステップで生成したパッチをプログラムに適用させる．さらに，パッチを適用させたプログラムに対して，ソフトウェアテストを実施することで修正後における不具合の有無を確認する．すべてのソフトウェアテストに合格

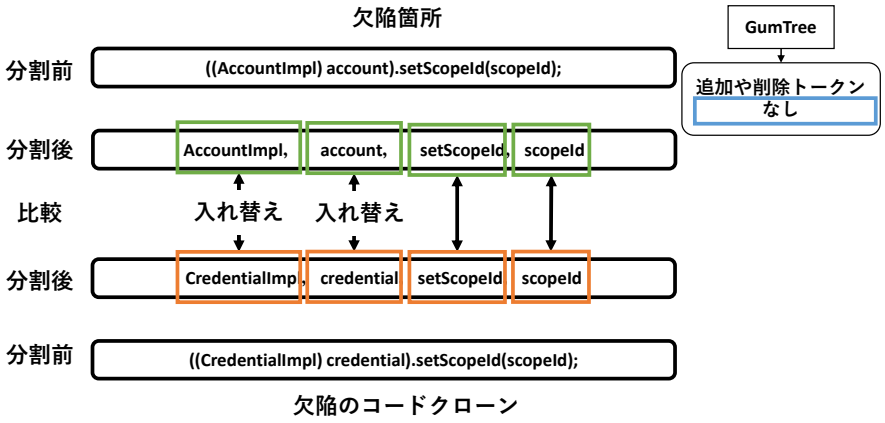


図 3　トークンの入れ替えが発生する場合における比較例

した場合，開発者に修正後のプログラムと修正箇所を提示する．

Step 3-1　パッチ適用コマンドによってパッチを対象ファイルに適用させる．パッチが適用できない場合，エラーメッセージと共にパッチファイルを開発者に提示する．

Step 3-2　パッチを適用したファイルを含むプログラム全体に対してソフトウェアテストを実施する．ソフトウェアテストに通らなかった場合，パッチ適用前のプログラムに戻し，エラーメッセージと共にパッチファイルを開発者に提示する．

以上の 3 ステップの処理を自動化した．本手法は，1 つの欠陥を修正した前後の差分を抜き出したパッチファイルと，修正前のコード片を含む対象とするプログラムが格納されているディレクトリ名を入力として与えることで，欠陥のコードクローンがある場合，欠陥のコードクローン箇所を修正したプログラムを出力する．ただし，コードクローンが検出されない場合，パッチが適用できない場合，ソフトウェアテストに通らない場合はそれぞれの詳細を出力する．

4　ケーススタディ

提案手法を実装し，IoT の欠陥事例を対象として自動パッチ生成における評価実験を行った．

4.1　IoT の欠陥事例の収集

IoT の欠陥事例の収集にあたり，Makhshari らのデータセット [1]を利用した．このデータセットは，IoT に関わるプロジェクトの欠陥レポート（イシュー）を収集したデータセットである．以下のような流れで，関連するプルリクエストにおける変更した差異を記述したパッチファイルと修正前のソースコードを収集し [2]，その中から，修正箇所がコードクローンとなっている IoT の欠陥事例を目視で収集した．さらに，修正箇所がコードクローンとなっている IoT の欠陥事例の中でも，strictly-cloned patches [13]（すべての修正がコードクローンとなっているパッチ）を抽出した．

i　Makhshari らのデータセットからリポジトリのオーナー名，リポジトリ名，イシュー番号を取得する．

ii　次に，REST API を使用して，オーナー名，リポジトリ名，イシュー番号から

[1] https://github.com/IoTSEstudy/IoTbugschallenges

[2] https://zenodo.org/record/5090430\#.YgZcit_P02w

プルリクエスト番号を検索する．これは，イシュー番号から直接パッチファイルとパッチファイルにより変更されたファイル群を取得することはできないためである．また，イシューとプルリクエストの関係は一様ではないので，すべてのプルリクエスト番号を取得することはできない．

iii 取得したオーナー名，リポジトリ名，プルリクエスト番号から GitHub の REST API を使用して，パッチファイルとパッチにより変更されたファイル群を取得する．

iv パッチファイルとパッチにより変更されたファイル群を入力として，修正される前のファイル群をコマンドにより取得する．

v パッチファイルを目視で確認し，修正箇所がコードクローンとなっている IoT の欠陥事例を手動で収集する．

vi 修正箇所がコードクローンとなっている IoT の欠陥事例から strictly-cloned patch を抽出する．

本研究の評価では，修正箇所がコードクローンとなっている IoT の欠陥事例の中で，すべての修正がコードクローンとなっている 41 事例（strictly-cloned patches を持つ欠陥事例）を対象とした．

4.2 使用ツール

本調査において，実際に使用したコードクローン検出ツール，編集スクリプト生成ツール，自動ビルドツールについて紹介する．コードクローン検出ツールは CCFinderSW [11] を採用した．CCFinderSW は検査したいディレクトリと言語を指定することでコードクローンを検出することができる．CCFinderSW を採用した理由は，事前調査で欠陥検出において有効性を示した点と，テキストベースのファイルを出力とするため 2 次利用が容易である点からである．欠陥コードを含むファイル名と欠陥の行数をキーワードに CCFinderSW が出力するテキストベースのファイルを精査する．編集スクリプト生成ツールは GumTree [12] を採用した．GumTree は，元の欠陥と欠陥のコードクローンを比較して差分を抽出する．関数呼び出しの順序が異なる場合，‘move-node ’ または ‘move-tree ’ として出力され，挿入・削除トークンがある場合，‘insert-node ’ または ‘insert-tree ’ または ‘delete-node ’ または ‘delete-tree ’ と出力される．そのため，‘move ’ と ‘insert ’ と ‘delete ’ をキーワードとして出力ファイルを精査する．自動ビルドツールは Maven を採用した．Maven は Java プログラムをビルドするためのツールで，今回は自動テストに使用する．開発者はテストコードを作成し，XML ファイルに記述することで自動ソフトウェアテストが可能となる．

4.3 評価指標

本研究では，コードクローンとなっている修正箇所の 1 つを元パッチとして用いることで，それ以外のコードクローンとなっている修正箇所を新パッチとして生成することができれば，自動パッチ生成成功と定義する．また，以下のように適合率，再現率を定義する．

- 適合率＝パッチ生成成功数/パッチ生成数
- 再現率＝パッチ生成成功数/欠陥検出数

適合率は，パッチが生成された中でどれだけ正しかったかを示し，コードクローン検出による欠陥の誤検出とパッチ生成失敗に影響している．再現率は，パッチが生成された中でどれだけ取りこぼしがないかを示し，パッチ生成成功率を表す．欠陥検出率は，コードクローン検出によって，どれだけ取りこぼしなく欠陥検出を行えたかを示し，コードクローン検出における欠陥検出能力を表す．また，パッチ生成成功数，パッチ生成数，欠陥検出数の関係を図 4 に示す．

また，欠陥コードと欠陥のコードクローンが以下の関係の場合分けによって，評価を行った．

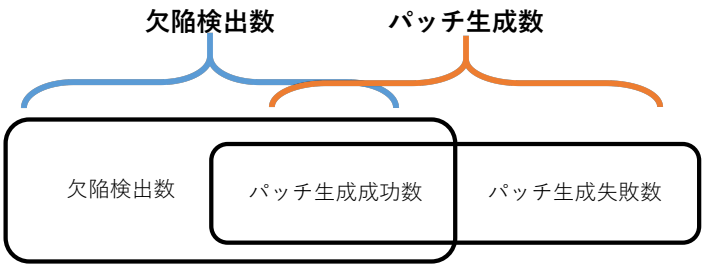


図 4　パッチ生成成功数，パッチ生成数，欠陥検出数の関係

表 1　自動パッチ生成の評価結果

	欠陥 事例数	欠陥 検出数	パッチ生成 成功数	欠陥 検出率	適合率	再現率
場合 1	16	9	9	56%	64%	100%
場合 2	15	9	8	60%	43%	89%
場合 3	9	7	7	78%	83%	100%
場合 4	1	1	1	100%	100%	100%
合計	41	26	25	63%	60%	96%

場合 1　欠陥コードと欠陥のコードクローンが全く同じクローンの場合
場合 2　欠陥コードと欠陥のコードクローンの対応するトークン数は一致するが，異なるトークンがある場合
場合 3　欠陥コードと欠陥のコードクローンの間に挿入・削除トークンがある場合
場合 4　欠陥コードと欠陥のコードクローンの関数呼び出しの順序が異なる場合

4.4　結果

ケーススタディの結果を表 1 に示す．本調査では IoT の 41 の欠陥事例に対して，63%の欠陥検出に成功した．また，欠陥検出された IoT の欠陥事例における，96%の事例に対して，パッチ生成に成功した．これらの結果は，本手法である自動パッチ生成において高い有効性を示した．

4.5　考察

本調査では，コードクローン検出された IoT の欠陥事例に対して，自動パッチ生成の再現率は 96%となり，本手法の有効性を示した．また，本調査では，41 の IoT の欠陥事例に対して，自動パッチ生成を試みた．事例を増やすことにより，多数の事例での本手法の有効性を示すことが求められる．

本調査では，コードクローンとして検出された IoT の欠陥事例に対して，自動パッチ生成において 60%の適合率を示した．これは，生成されたパッチにおいて 40%が実際にパッチとして適用されないことを表している．この実際に適用されないパッチには，開発者が修正が必要でないと判断して適用しなかった場合と，欠陥と全く関係ない場合が考えられる．開発者が修正が必要でないと判断して適用しなかった場合を本手法で判断することは難しい．そのため，最終的にパッチを適用する判断は開発者にゆだねられるオプションを作成した．このオプションにより，生成されたパッチの中で実際に適用されないパッチに対応が可能である．欠陥と全く関係ない場合は往々にしてパッチがプログラムに対して適用できない，またはソフトウェアテストに通らないことが多い．そのため，パッチ適用時，ソフトウェアテスト時

にフィルタリングされる．

5　妥当性への脅威

本研究は，Java における IoT の欠陥を対象とした．IoT プログラミングでは，Python(21%)，Java(18%)，JavaScript(17%)，C(13%)，C++(13%) [2] が使用されることが多い．また，Go，Ruby，C#なども使用される．そのため，IoT の欠陥の検出のためには複数言語に対応可能であることが求められる．また，コードクローンが存在しやすい言語の特徴としてオブジェクト指向であることが挙げられる．オブジェクト指向のように文法が複雑である方がコードクローンが存在しやすく，単純であるほど存在しにくい傾向にある．C，Java，Python の順で複雑性が高く，コードクローンが多くなると予測されるため，その順に則した形で再現率も変化すると考えられる．Python などのインタプリタ型の言語はソースコードが短くなりやすく，文法が多様である．つまり，同じ内容のプログラムでも違うコードで記述が可能であり，同じ内容の欠陥が存在しても検出・修正が難しい場合がある．

本研究では，IoT の欠陥を広く収集した．その中で，コードクローンとなっている欠陥を手動で収集することで，本研究の調査対象とした．調査対象は，コードクローンのみで構成される欠陥としたが，コードクローンを含む箇所があり，かつ含まない箇所がある欠陥も存在する．コードクローンを含む箇所があり，かつ含まない箇所がある欠陥の修正対象としてはコードクローンを含む箇所に限られるが，このような欠陥は複雑な場合が多い．本手法は，コードクローンとなっている欠陥を対象としているが，複雑な場合は対応できない場合がある．これは本手法がソースコードの行数をもとに欠陥箇所を比較する構造から，改行や，空白行の追加がある場合，修正においてずれが発生する可能性がある．この対応として，ソースコードの正規化を自動的に行う工程の追加が求められる．また，IoT の欠陥は，デバイス，サーバ，デバイスとサーバの通信のいずれかに関連する．その中でも，デバイスとサーバの通信に関連する欠陥はデバイス側とサーバ側の両サイドを考える必要があるため，欠陥の修正箇所は複雑になる可能性が高い．さらに，片側のプログラムコードが本手法で修正された場合，反対側のプログラムコードの変更も考えられる．そのため，テストに通らないケースが考えられる．

6　まとめ

本研究は，開発者が 1 つの欠陥を修正した後，その欠陥と類似している欠陥を検出し，自動的にパッチを生成する手法を提案した．また，IoT の欠陥事例を対象とした自動パッチ生成における評価実験の結果は，適合率は 60%，再現率は 96%であり，本手法の有効性を確認した．

自動プログラム修正手法の一種に，修正パターンを収集し，その修正パターンを該当箇所に当てはめることでプログラムを修正する方法がある．修正パターンを収集した既存研究として，Java を対象とした CPatMiner [25] や Python で記述された機械学習システムを対象とした R-CPatMiner [26] があるが，本研究は IoT システムを対象とした自動プログラム修正における修正パターン収集の一助になると考えられる．今後は，自動プログラム修正におけるパターン収集のアルゴリズムを分析し，本手法を利用できるか検討することで，自動プログラム修正の効率化に取り組む．

謝辞　本研究は，JST さきがけ JPMJPR21PA および JSPS 科研費 JP20K11745 の助成を受けた．

参考文献

[1] Roberto Minerva, Abyi Biru, and Domenico Rotondi. Towards a definition of the Internet of Things (IoT). *IEEE Internet Initiative*, 2015.

[2] Amir Makhshari and Ali Mesbah. IoT bugs and development challenges. In *Proc. of ICSE 2021*, pp. 460–472, 2021.
[3] Katie Costello. Predicts 2021: Cloud and edge infrastructure cloud infrastructure edge. https://www.gartner.com/smarterwithgartner/gartner-predicts-the-future-of-cloud-and-edge-infrastructure, 2021. (Accessed on 07/19/2022).
[4] Statista. Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions).
[5] Mark Hung. Leading the IoT, gartner insights on how to lead in a connected world. *Gartner Research*, Vol. 1, pp. 1–5, 2017.
[6] Xingbin Jiang, Michele Lora, and Sudipta Chattopadhyay. An experimental analysis of security vulnerabilities in industrial IoT devices. *ACM TOIT*, Vol. 20, No. 2, pp. 1–24, 2020.
[7] Nam H Pham, Tung Thanh Nguyen, Hoan Anh Nguyen, Xinying Wang, Anh Tuan Nguyen, and Tien N Nguyen. Detecting recurring and similar software vulnerabilities. In *Proc. of ICSE 2010*, Vol. 2, pp. 227–230, 2010.
[8] Seulbae Kim and Heejo Lee. Software systems at risk: An empirical study of cloned vulnerabilities in practice. *Comput. Secur.*, Vol. 77, pp. 720–736, 2018.
[9] Norihiro Yoshida, Takeshi Hattori, and Katsuro Inoue. Finding Similar Defects Using Synonymous Identifier Retrieval. In *Proc. of IWSC 2010*, p. 49–56, 2010.
[10] Jian Gao, Xin Yang, Yu Jiang, Houbing Song, Kim-Kwang Raymond Choo, and Jiaguang Sun. Semantic learning based cross-platform binary vulnerability search for IoT devices. *IEEE Trans. Industr. Inform*, Vol. 17, No. 2, pp. 971–979, 2019.
[11] 瀬村雄一, 吉田則裕, 崔恩瀞, 井上克郎. 多様なプログラミング言語に対応可能なコードクローン検出ツール CCFinderSW. 電子情報通信学会論文誌 D, Vol. J103-D, No. 04, pp. 215–227, 2020.
[12] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. Fine-grained and accurate source code differencing. In *Proc. of ASE 2014*, pp. 313–324, 2014.
[13] Fernanda Madeiral and Thomas Durieux. A large-scale study on human-cloned changes for automated program repair. In *Proc. of MSR 2021*, pp. 510–514, 2021.
[14] Andreas Zeller. *Why Programs Fail, Second Edition: A Guide to Systematic Debugging*. Morgan Kaufmann, 2009.
[15] Ekwa Duala-Ekoko and Martin P. Robillard. Clone region descriptors: Representing and tracking duplication in source code. *ACM Trans. Softw. Eng. Methodol.*, Vol. 20, No. 1, 2010.
[16] Daqing Hou, Patricia Jablonski, and Ferosh Jacob. CnP: Towards an environment for the proactive management of copy-and-paste programming. *Proc. of ICPC 2009*, pp. 238–242, 2009.
[17] Michael Toomim, Andrew Begel, and Susan L. Graham. Managing duplicated code with linked editing. In *Proc. of VL/HCC 2004*, pp. 173–180, 2004.
[18] Sushma Sakala, Vamshi Krishna Epuri, Samuel Sungmin Cho, and Myoungkyu Song. LAC: Locating and applying consistent and repetitive changes. In *Proc. of COMPSAC 2019*, Vol. 1, pp. 179–184, 2019.
[19] Yoshiki Higo, Shinpei Hayashi, Hideaki Hata, and Meiyappan Nagappan. Ammonia: an approach for deriving project-specific bug patterns. *Empir. Softw. Eng.*, Vol. 25, No. 3, pp. 1951–1979, 2020.
[20] Georg Dotzler, Marius Kamp, Patrick Kreutzer, and Michael Philippsen. More accurate recommendations for method-level changes. In *Proc. of ESEC/FSE 2017*, p. 798–808, 2017.
[21] Jiajun Jiang, Luyao Ren, Yingfei Xiong, and Lingming Zhang. Inferring program transformations from singular examples via big code. In *Proc. of ASE 2019*, pp. 255–266, 2019.
[22] Kunihiro Noda, Haruki Yokoyama, and Shinji Kikuchi. Sirius: Static program repair with dependence graph-based systematic edit patterns. In *Proc. of ICSME 2021*, pp. 437–447, 2021.
[23] John Jacobellis, Na Meng, and Miryung Kim. LASE: An example-based program transformation tool for locating and applying systematic edits. In *Proc. of ICSE 2013*, pp. 1319–1322, 2013.
[24] Seemanta Saha, Ripon K. Saha, and Mukul R. Prasad. Harnessing evolution for multi-hunk program repair. In *Proc. of ICSE 2019*, p. 13–24, 2019.
[25] Hoan Anh Nguyen, Tien N. Nguyen, Danny Dig, Son Nguyen, Hieu Tran, and Michael Hilton. Graph-based mining of in-the-wild, fine-grained, semantic code change patterns. In *Proc. of ICSE 2019*, pp. 819–830, 2019.
[26] Malinda Dilhara, Ameya Ketkar, Nikhith Sannidhi, and Danny Dig. Discovering repetitive code changes in python ml systems. In *Proc. of ICSE 2022*, p. 736–748, 2022.

コード難読化ツールの信頼性を評価するフレームワークの検討

Towards a Framework to Evaluate the Reliability of Code Obfuscation Tools

北岡 哲哉[*] 神崎 雄一郎[†] 石尾 隆[‡] 嶋利 一真[§] 松本 健一[¶]

あらまし ソフトウェアを解析や改ざんから保護するためのツールとして，コード難読化を実装したツール（難読化ツール）が多数提案されている．しかし，難読化ツールが対象コードに不具合を生じさせずに解析を困難にできるか，すなわち「信頼できる」ものかを調べる方法は明らかになっていない．本研究では，難読化ツールの信頼性を評価するための方法を検討する．具体的には，信頼性の高い難読化ツールかを判断するためのメトリクスとして，与えられたコードの機能を維持したまま難読化を適用できる度合いを表す機能維持性と，与えられたコードの内容を変化させる度合いを表すコード変形性という 2 つの指標を提案し，多数のプログラムを用いてこれらを実証的に評価するためのフレームワークを提案する．提案したフレームワークを用いた実験では，実際に公開されている 2 種類の難読化ツールを対象に機能維持性とコード変形性の評価を行い，得られた結果から，実験対象の難読化ツールの信頼性について議論する．

1 はじめに

ソフトウェアに対する不正な解析および改ざん攻撃 [1] の成功を遅らせる方法として，コード難読化（code obfuscation）が広く用いられている．コード難読化は，与えられたプログラムコードを，コードの機能を維持したまま人間や機械による解析が困難な形に変換する技術である．コード難読化を実装したツール（以下，難読化ツールと呼ぶ）として，多数の商用のツールに加え，Tigress [2,3] や Obfuscator-LLVM [4] といった無償のツールも活発に開発され，公開されている．

コード難読化は，変換前のコードが持つ入力と出力の対応関係（入出力関係）が，変換後のコードでも維持されることが前提となる．一方で，コード難読化では，制御構造の平滑化，コードの仮想化，実行時コード生成など，プログラムの構造を大きく変形し得る様々な変換を行う [5]．そのため，難読化ツールを適用することで，ソフトウェア開発者が意図していない不具合がコードに生じる可能性は否定できない．実際の Android アプリケーションの開発者を対象にしたコード難読化に関する調査においても，コード難読化によって自身が作成したソフトウェアに不具合が生じる（ソフトウェアが「破壊」されてしまう）ことを懸念する声が報告されている [6]．難読化ツールによってソフトウェアが適切に保護されるためには，難読化ツールが不具合を生じさせずに解析を困難にできるか，すなわち「信頼できる」ものかを把握することが重要である．しかし，個々の難読化ツールの開発者が，難読化対象となる様々なソフトウェアを収集し，信頼性の評価を行うことは難しい．

そこで本研究では，難読化ツールの信頼性を評価するための方法を提案する．本稿では，その第 1 の基準として，機能維持性，すなわち，様々なプログラムコード

[*]Tetsuya Kitaoka, 奈良先端科学技術大学院大学

[†]Yuichiro Kanzaki, 熊本高等専門学校

[‡]Takashi Ishio, 奈良先端科学技術大学院大学

[§]Kazumasa Shimari, 奈良先端科学技術大学院大学

[¶]Kenichi Matsumoto, 奈良先端科学技術大学院大学

に対して，不具合を生じさせることなく元来の機能を維持してコードの変換を行える度合いを表す指標を設ける．機能維持性は，多数のコードをツールに与えたときに，難読化前後で同一の入出力関係を保っているコードの割合によって評価する．

難読化されたコードが元来の機能を維持できているとしても，元来のコードと内容が同一であったり，ごくわずかな変化しかない場合は，コードの解析が困難になったことを期待できるとはいい難い．そのため，第 2 の基準として，コード変形性，すなわち，様々なプログラムコードに対して，コードの内容を変化させた度合いを表す指標を設ける．コード変形性は，多数のコードをツールに与えたときに，各コードの難読化前後のコード間の距離（類似度の低さ）の平均によって評価する．

本研究では，このような機能維持性やコード変形性といった信頼性に関するメトリクスを定義し，多数のプログラムを用いてそれらを実証的に評価するためのフレームワークを提案する．また，実際に公開されている 2 種類の難読化ツールを対象に機能維持性やコード変形性を評価する実験を行い，得られた結果から，実験対象の難読化ツールの信頼性について議論する．

本稿では，2 章で関連する研究を紹介する．3 章で提案手法について述べ，4 章で実験について述べた後，5 章で実験結果について議論を行う．最後に 6 章でまとめと今後の課題について述べる．

2 関連研究

コード難読化の有効性，すなわち解析攻撃に対する強さを評価する研究は従来さかんに行われており，近年も増加傾向にある [7]．例えば，Banescu らは，シンボリック実行を用いた自動解析攻撃に対する防御方法として，既存の難読化方法の有効性を，難読化されたコードの実際の解析時間によって議論している [8]．また，難読化された Java プログラムを理解・改ざんする困難さを，実際に人間が解析する時間に基づいて評価する手法 [9] や，難読化機構が攻撃者に発見されにくい度合い（ステルス性）を確率的言語モデルを用いて評価する手法 [10]，コードに難読化を適用した難読化ツールの特定がどれだけ困難かを Java バイトコードのオペコード列の出現頻度の類似性をもとに評価する手法 [11] も提案されている．加えて，文献 [12] では，名前難読化の逆変換の困難さを評価する手法が提案されている．この研究では，Java バイトコードのオペコード列が難読化ツールの適用によってどの程度変化するのかを，難読化前後のオペコード列の編集距離によって算出する実験を行っており，この点においてコード変形性の評価にも関連がある．

本研究は，特定の攻撃モデルに対する防御の強さではなく，「元来の機能を維持したままコードの内容を十分に変化させることができるか」という難読化方法（難読化ツール）の基本的な品質を評価することを試みる．提案手法は，一定の攻撃モデルに特化した従来の有効性評価の方法を組み合わせることが可能であり，難読化方法やツールの性能を正確に評価するための一助になると考える．

3 提案手法

本研究では，難読化ツールの信頼性を評価することを目的として，難読化ツールの信頼性に関するメトリクスを定義し，これらを用いた評価フレームワークを提案する．難読化対象となるコード群は事前に与えられているものとして，難読化ツールを実行し，機能維持性とコード変形性という 2 つのメトリクスを計測する．難読化ツールは複数の難読化方法を実装している場合が一般的であるため，本フレームワークでは，難読化ツールが実装している難読化方法ごとに機能維持性とコード変形性を評価する．つまり，両メトリクスとも，難読化ツールが実装している 1 つの難読化方法に対して，1 つの値が出力される．

難読化方法 o を難読化対象コード群 P に対して適用した場合の機能維持性とコード変形性を以下のように定義する．

機能維持性 S： 難読化ツールが，与えられたコードの機能を維持したまま難読化を適用できる度合いを示す．このメトリクスは，難読化されたコードの入出力等価性，すなわち，入力値とコードに入力値を与えたときの出力が難読化前後ですべて一致しているようなコードの割合として，以下のように算出する．

$$S(o) = \frac{|\{p \in P | Equivalent(p, o(p))\}|}{|P|}$$

ここで，$o(p)$ は難読化方法 o を適用した結果である．また，$Equivalent(p, o(p))$ は，難読化前後のコードの組の入出力が同一であるときに真となる述語である．機能維持性は 1 となることが望ましい．難読化ツールが入出力等価性を満たすコードを生成できなかった場合，難読化後のコードは難読化前と同じ挙動をしないことを示すため，機能維持性は低下する．

コード変形性 D： 難読化ツールが，与えられたコードの内容を変化させる度合いを示す．このメトリクスは，難読化されたコードごとにコード間距離（コード類似度の低さ）を計測し，その平均から算出する．

$$D(o) = \frac{\sum_{p \in P} Distance(p,\ o(p))}{|P|}$$

難読化ツールが元来のコード内容や表現を十分に変化させていない場合，コード変形性が低くなる．

これらのメトリクスを算出するフレームワークの全体像を 図 1 に示す．本フレームワークへの入力は，難読化ツール，難読化対象のコード群とコード群に対応したシンボリック実行を行うスクリプトの 3 つで，出力は，機能維持性とコード変形性の 2 つである．提案するフレームワークは大きく 3 つのステップで構成される．Step Ⅰでは，まず，入力として与えられたコード群 P に対して，評価対象の難読化ツールの適用を行う．また，$Equivalent(p, o(p))$ の判定を行うための準備として，難読化前のコード群に対してシンボリック実行を適用し，コードの各出力に到達する入力値を探索し，得られた入出力関係をテストケースとして保存する．シンボリック実行による入力値の探索では，コードから発見した各実行パスごとに，実行パスが通過する具体的な入力値を 1 つずつ生成する．Step Ⅱでは，得られたテストケースを難読化後のコード群に対して実行し，それぞれのコードが入出力等価性を満たすかを検証し，機能維持性を算出する．

Step Ⅲでは，入力として与えられたコード群と，Step Ⅰで難読化されたコード群を用いて，コード変形性を算出する．コード間距離の計算には，様々なアプローチが考えられるが，本稿では，難読化前後での N-gram アセンブリ命令列からなる集合同士の Simpson 係数（Overlap coefficient） [13] をもとに測定する．Simpson 係数は，要素数が少ない方の集合の要素数に対する集合同士で共通する要素数の割合から求められるため，この値を 1 から減算することでコード間距離が測定可能である．また，Simpson 係数は一方の集合が別の集合の真部分集合であったときに算出値が 1 となる性質を持っているため，コード間距離が 0 となったときには難読化されたコードから元来のコードが予測されやすいと判断することができる．なお，Simpson 係数の測定において N-gram を採用した理由は，アセンブリ命令列の文脈を考慮して評価を行うためである．難読化前後の N-gram アセンブリ命令列の集合をそれぞれ $A(p)$，$A(o(p))$ としたとき，コード間距離は次の式で求められる．

$$Distance(p, o(p)) = 1 - \frac{|A(p) \cap A(o(p))|}{\min\left(|A(p)|, |A(o(p))|\right)}$$

コード間距離の測定のために，それぞれのコード群から難読化前後のコードを抽出し，実行可能コードをそれぞれ生成する．そして，生成された実行可能コードを逆アセンブルして得られた，N-gram アセンブリ命令列同士の Simpson 係数からコー

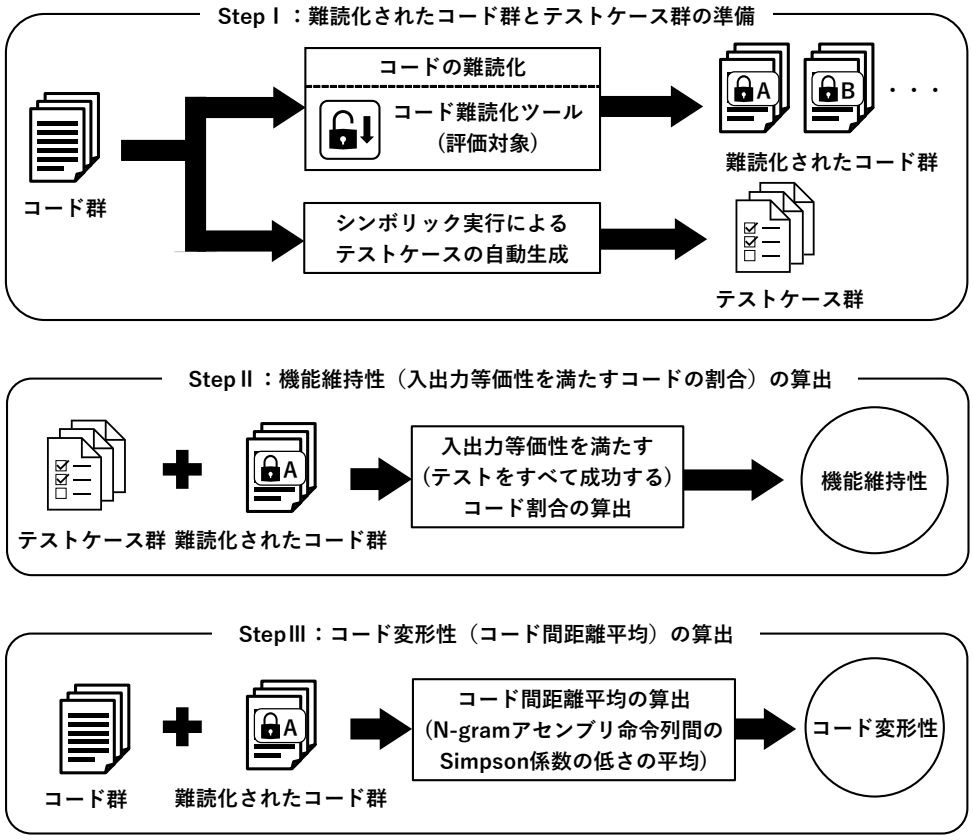


図 1: 提案するフレームワークの全体像

ド間距離を求める．その後，それぞれのコード群から測定したコード間距離の平均が，コード変形性として算出される．

以上の手順から，評価対象とする難読化ツールが実装している各難読化方法の信頼性に関する傾向を分析する．

4 実験

3 章で述べた提案手法のフレームワークを用いて，よく知られた難読化ツールである Tigress [2,3] や Obfuscator-LLVM [4] で実装されている難読化方法の機能維持性とコード変形性を評価する実験を行う．本実験で評価対象とする難読化方法や，難読化の適用対象となるコードの説明を次に示す．

4.1 実験方法

4.1.1 難読化ツールと難読化方法

評価対象の難読化ツールは，Tigress [2, 3] と Obfuscator-LLVM [4] の 2 つである．Tigress は C 言語のソースコードのレベルで難読化を行い，Obfuscator-LLVM は LLVM IR のレベルで難読化を行う．Tigress と Obfuscator-LLVM はどちらも複数の難読化方法を実装していることから，本実験では，対象とする難読化方法ごとに機能維持性とコード変形性を評価する．

今回の実験で対象とする Tigress の難読化方法を表 1(a) に，Obfuscator-LLVM

表 1: 実験対象の難読化ツールが実装している難読化方法

(a) Tigress が実装している難読化方法 (一部)

Abbr.	**Description**
EncA	Encode Arithmetic: 整数演算を複雑な表現に置換する
EncL	Encode Literals: 整数や文字列リテラルを不明瞭な表現に変換する
Flat	Flatten: 構造化された制御構造を平滑な形に変換する
AddO4	Add 4 Opaque predicates: Opaque predicate [5] を 4 つ挿入する
AddO16	Add 16 Opaque predicates: Opaque predicate [5] を 16 個挿入する
Virt	Virtualize: コードを仮想化する，すなわち独自のバイトコードとそのインタプリタの組合せに変換する

(b) Obfuscator-LLVM が実装している難読化方法

Abbr.	**Description**
BCF	Bogus Control Flow: Opaque predicate [5] を用いたダミーの条件分岐を追加する
CFF	Control Flow Flattening: 構造化された制御構造を平滑な形に変換する
ISub	Instructions Substitution: 加算・減算・論理演算などの演算を複雑な表現に変換する

の難読化方法を表 1(b) にそれぞれ示す．Tigress は，動的にコードを変形するものを含め多数の難読化方法の実装が公開されているが，今回は表 1(a) に挙げたものを用いる．Tigress の場合は，対象となる関数を指定して難読化を適用する一方で，Obfuscator-LLVM では，対象となる関数は指定せず，コード単位で難読化が適用される．

本実験では，表 1 に示した難読化方法を，1 つ単独で適用した場合と，2 つ組み合わせて適用した場合を評価する．本実験では，Banescu らによる，自動解析攻撃に対する難読化方法の有効性を評価する研究 [8] の実験で評価対象とされた組合せと同一のものを用いることとし，Tigress では計 28 種類，Obfuscator-LLVM では計 9 種類の難読化方法について，それぞれ機能維持性とコード変形性を評価する．難読化方法を組み合わせた場合の名称は，それらの方法を適用した順番に，難読化方法の名称を 2 つつなげて表記する．たとえば AddO16_EncA は，AddO16 を適用したあとに EncA を適用したことを意味する．また，同じ難読化方法を 2 回適用した場合は，x2 として表現している．

4.1.2 難読化対象のコードとテストケース

難読化の適用対象となるコード群は，Tigress に実装されている RandomFunc 機能によって自動生成された，公開されている 5,760 個の C 言語ソースコード [8] [1]から構築する．このコード群は，main 関数とランダムに生成された関数（RandomFunc 関数）を含んでおり，プログラムとしてはコマンドライン引数の値をもとに RandomFunc 関数が処理した値を出力するものとなっている．特定の値がプログラ

[1] `https://github.com/tum-i4/obfuscation-benchmarks/`

表 2: 難読化の適用対象となるコード群 (1,318 個) の情報

	Min	Max	Mean
論理 LOC	80	120	90.20
1 コードに対するテストケース数	3	5	3.107
1 コードに対するテストのカバレッジ (gcov の Line executed で計測)	90.00%	96.00%	92.49%

ムに与えられた場合，プログラムは “You win!” を出力する．

本実験では，シンボリック実行エンジンとして angr [14] を採用し，テストケースの自動生成を行う．具体的には，特定のコードの出力（“You win!” の文字列など）に到達する入力値をシンボリック実行によって探索させ，得られた入力値（コマンドライン引数）とそれに対する出力値の組で構成されるテストケースを生成する．ソースコードからコンパイルした実行ファイルにテストケースを与え，C 言語用のコードカバレッジの計測ツール gcov で求められた Lines executed の合計が 90%以上となるコードを，難読化の適用対象となるコード群として抽出した．結果として本実験では，1,318 個の C 言語のソースコードを難読化の適用対象となるコード群を得た．

難読化の適用対象となるコード群に関する情報を表 2 に示す．コード群全体で論理 LOC は 80 から 120 となっており，図 1 の Step. Ⅰによって，1 つのコードに対して 3〜5 つのテストケースが自動生成されている．図 1 の Step. Ⅱに該当する機能維持性の算出では，この生成したテストケースを実行し，元来のコードの機能に対応する実行が得られたかを計算する．

Tigress は特定の関数を選んで難読化を適用するツールであることから，main と RandomFunc のそれぞれに個別に難読化を適用し，部分的に難読化されたコードを 2 ファイル作成し，それらを個別に難読化したプログラムとみなして，機能維持性，コード変形性の計算に使用する（2,636 個のコード群から評価値を求める）．これらの実行可能コードは，GCC （バージョンは Red Hat 版の 8.5.0-12）を用いて生成する．Obfuscator-LLVM はファイル単位で難読化を適用するため，1,318 個のプログラムそれぞれに対して難読化を行い，その機能維持性，コード変形性を求める．LLVM IR レベルで難読化変換が行われる都合上，Clang 4.0.1 を用いて実行可能コードを生成する．

4.1.3 コード変形性の計測に用いるツール

図 1 の Step. Ⅲに該当するコード変形性の算出では，難読化前後のコードから N-gram アセンブリ命令列を求めて，難読化前後のコードのコード間距離を算出する．難読化の適用対象となるコード群の実行可能コードは，評価対象となる難読化ツールごとに対応するコンパイラを用いて生成する．本実験では，アセンブリ命令としてオペコードのみを扱う．オペコードの取得には，GNU Binutils の objdump[2] による線形逆アセンブルを利用し，対象コードの `.text` セクションに含まれるものを取得する．今回の実験では，N-gram の長さ（N）を 1〜3 まで変化させてそれぞれで評価を行う．

4.2 実験結果

4.2.1 機能維持性

実験によって得られた，各難読化方法の機能維持性の一覧を表 3 に示す．表 3 より，Tigress に実装されている難読化方法は，機能維持性が 0.9890〜0.9996 となっ

[2] `https://www.gnu.org/software/binutils/`

表 3: 難読化ツールに実装されている難読化方法ごとの機能維持性

Tigress に実装されている難読化方法					
AddO16	0.9970	EncA_Virt	0.9905	Flat_Virt	0.9890
AddO16_EncA	0.9970	EncL	0.9992	Flat x2	0.9985
AddO16_EncL	0.9970	EncL_AddO16	0.9973	Virt	0.9898
AddO16_Flat	0.9977	EncL_EncA	0.9996	Virt_AddO16	0.9898
AddO16_Virt	0.9909	EncL_Flat	0.9989	Virt_EncA	0.9898
AddO4	0.9970	EncL_Virt	0.9894	Virt_EncL	0.9898
EncA	0.9992	Flat	0.9985	Virt_Flat	0.9901
EncA_AddO16	0.9977	Flat_AddO16	0.9962	Virt x2	0.9898
EncA_EncL	0.9996	Flat_EncA	0.9981		
EncA_Flat	0.9985	Flat_EncL	0.9989		
Obfuscator-LLVM に実装されている難読化方法					
BCF	0.9734	CFF	0.9734	ISub	0.9742
BCF_CFF	0.9727	CFF_BCF	0.9727	ISub_BCF	0.9742
BCF_ISub	0.9734	CFF_ISub	0.9734	ISub_CFF	0.9734

ていることから，元来のコードの機能をほぼ維持して変換できていることがわかる．Virt（コードの仮想化）は，他の難読化方法を組み合わせた場合を含めても 0.9890〜0.9909 となっており，その他の難読化方法と比べると低い値となっている．同様に，Obfuscator-LLVM に実装されている難読化方法は，機能維持性が 0.9727〜0.9742 となっていることから，元来のコードの機能をほぼ維持して変換できることがわかる．

なお，入出力等価性を満たさなかったコード群を調べると，与えられたテストケースによるテストをすべて失敗するようなコードは存在せず，いずれも少なくとも 1 つのテストには成功していたことがわかった．さらに，テストが失敗したコードについて難読化前の状態を調査したところ，確保されていないメモリへの参照があるために，動作（出力）が不安定なコードが含まれていた．難読化対象となるコード群に不具合のあるコードが含まれている場合は，機能維持性の評価に影響を与える可能性があるため，このようなコードを難読化対象から除外するための対策が求められる．この点は今後の課題とする．

4.2.2 コード変形性

実験によって得られた，各難読化方法のコード変形性を図 2 に示す．まず，Tigress の難読化方法のコード変形性に注目すると，Virt や，Virt を組み合わせた難読化のコード変形性は比較的高いことがわかる．Virt を用いた難読化は，機能維持性は比較的低かったものの，コードの内容は大きく変形させる傾向があるといえる．一方，EncL は，1-gram，2-gram，3-gram いずれの場合でも，コード変形性が最も低かった．EncL は，整数や文字列のリテラルを不明瞭にしてそのリテラルを用いているコードを変形するという手法であるために適用対象が大きく限定され，コード内容の変化が小さかったものと考えられる．

Obfuscator-LLVM の難読化方法については，いずれの場合も同程度のコード変形性があることがわかる．Obfuscator-LLVM のほうが Tigress よりもコード変形性が高いのは， Tigress が関数単位での部分的な難読化を適用しているのに対し，ファイルを対象とした難読化を適用しているという違いが影響していると思われる．1-gram でのコード変形性は Tigress の Virt などと大きな違いがないので，使用して

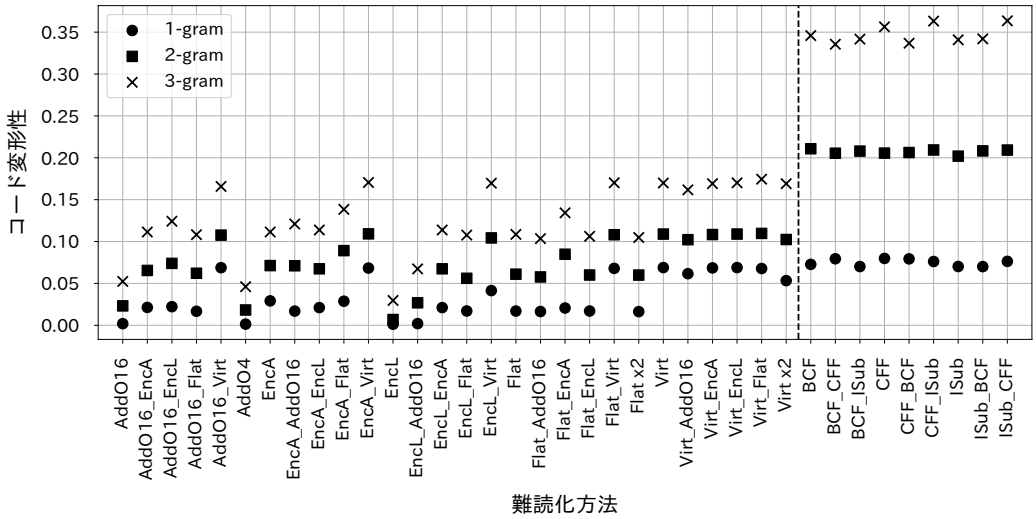


図 2: 難読化ツールに実装されている難読化方法ごとのコード変形性．破線の左側は Tigress，右側は Obfuscator-LLVM.

いる命令の集合の変更度合いという点では大きな違いはないが，2-gram や 3-gram の違いから，難読化後は多様な命令列が使用されていることが分かる．

難読化方法を組み合わせた場合のコード変形性については，その構成要素となる難読化方法のうち高いほうの値に近い結果となっている．つまり，単体で適用した場合と，複数組み合わせて適用した場合どちらについても，コード変形性には大きな差は生じなかった．これは，難読化方法ごとに使用する命令の集合が限られているためであると考えられる．

5 考察

本研究で難読化ツールの信頼性の第 1 の基準と定めた機能維持性については，Tigress の難読化方法はいずれも 0.9890 以上，Obfuscator-LLVM の難読化方法はいずれも 0.9727 以上と高かった．また，第 2 の基準と定めたコード変形性については，Tigress の EncL のようにコード内容の変化が小さいものが存在したものの，ほとんどの難読化方法がコード内容を変化させていることを確認した．どちらの難読化ツールも，大多数のプログラムに対して機能を維持したまま，一定量コードの内容を変化させているという点で，一定の信頼性があるツールであると考えられる．ただし，2 章で述べたように，難読化ツールによって攻撃者の解析が困難になるか，という点については評価を行っているわけではないので，特定の攻撃に対してどれだけ解析が困難になるかの評価が求められる場合は，各攻撃モデルに特化した評価技術を，本フレームワークと組み合わせて利用することが必要である．

本実験では，難読化をするほどコード変形性は上がるが，機能維持性が下がるというトレードオフの関係が示唆されている．機能維持性が 1 でなかったことから，難読化ツールによって不具合が生じる可能性があるという利用者の懸念は実際に正しいことが明らかとなった．難読化ツールの利用者にとっては，難読化したコードに対するテストの実施が必須である．今後，利用者が難読化を適用したいコードと，機能維持に失敗したコードの特徴を比較することにより，難読化の失敗リスクを見積もり，失敗の可能性が高いコードに対するテストを支援するなど，本フレームワークから得られる情報の活用につなげていきたい．

難読化ツールの開発者にとっては，本フレームワークによって機能維持ができな

かったと確認されたコードは，難読化方法およびツールの品質を高めるための重要な手がかりとなると期待している．ただし，今回の実験では，自動生成された多数の C 言語のソースコードを用いたが，その一部には 4.2.1 でも言及したように，動作が不安定なものが含まれていた．現時点では，難読化によって機能が維持されない場合に，それが難読化ツールのバグに起因する（将来修正されうる）のか，難読化方法自体の限界か，あるいは難読化対象のコード自体の問題であるのかを切り分けることが難しい．今後，難読化前の状態においてすでに動作が不安定なコードを除去するなど，評価に悪影響を及ぼす可能性のあるコードを除去するための基準や方法の開発が必要である．

なお，本実験では，それぞれのツールが開発された環境に合わせて，Tigress と Obfuscator-LLVM で異なるコンパイラを用いて実験を実施した．コード変形性はそれぞれのコードごとに測定しているため同一コンパイラ上での効果ではあるが，コンパイラの違いによって難読化方法の効果に差が生じる可能性はありうるため，ツールごとの動作環境の違いを吸収しながら測定条件を均一化することが，技術的な課題の 1 つである．

本研究のフレームワークにおいては，評価プロセスを自動化するために，シンボリック実行を用いている．今回は出力が明確でサイズの小さいコードを対象にしたためシンボリック実行が可能であったが，シンボリック実行は制御構造が複雑なコードに対しては失敗することも多い．そのため，難読化対象の各コードの入力を自動生成できない場合について，ソフトウェアの機能テストからテストケースを収集するなどの対策が求められる．

6 おわりに

本研究では，難読化ツールの信頼性を評価するための方法を検討した．具体的には，難読化ツールの信頼性を判断するためのメトリクスとして，機能維持性およびコード変形性の 2 つの指標を提案し，これらの指標を実証的に評価するためのフレームワークを提案した．本フレームワークは，難読化ツール，難読化対象のコード群とコード群に対応したシンボリック実行を行うスクリプトの 3 つを用意するだけで，難読化ツールに実装されている難読化方法の機能維持性とコード変形性を算出できる．提案したフレームワークを用いた実験では，よく知られた難読化ツールである Tigress と Obfuscator-LLVM の 2 つを対象に機能維持性とコード変形性の評価を行った．どちらの難読化ツールも，実験対象としたプログラムのうちの大多数に対して機能を維持したまま，コードの内容を変化させているという点で，一定の信頼性があることが確認できた．一方で，難読化ツールによって機能が維持されない場合があるという懸念が正しいことも明らかとなった．

今後の課題として，まず，難読化対象とするコードのセットの拡充が挙げられる．オープンソースソフトウェア等で様々なソースコードを収集することは容易であるが，そこから不備のある難読化対象のコードを除外し，除去するための方法の開発や，収集したコードに対する効果的なテストケースの生成あるいは収集方法の検討が必要である．また，難読化ツールにはそれぞれ使用するコンパイラの違いや CPU アーキテクチャの違い，対象プログラミング言語の違いなど，難読化に影響を及ぼす可能性のある要素が多数あることから，ツールの動作環境の違いを吸収しながら測定条件を均一化していく方法を検討していきたい．他の課題として，難読化ツールがコードに与える変化についての評価を，別の角度から行うことが挙げられる．本稿で述べた，Simpson 係数に基づくコード変形性が 0 に近い難読化ツールは，コードに与える変化が小さく，その難読化ツールによって難読化されたコードはオリジナルのコードを予測しやすい傾向にあると判断できる．一方，現状の方法では難読化ツールがコードに与える構造的な変化は十分に把握できないため，特徴的なアセンブリ命令の増加率やソースコードの認知的複雑度 [15] などの指標を用いて，難読

化ツールがコードに与える変化の度合いを異なる角度から評価する方法を検討している．将来的には，難読化ツールの開発者に本フレームワークを利用してもらい，機能維持性およびコード変形性の情報を蓄積，公開することで，難読化ツールの利用者が容易に複数のツールの性能を比較することのできる環境を実現したい．

謝辞 本研究の一部は，JSPS 科研費 JP22K11986，JP22K21279，JP20H05706 および JP19K11916 の助成を受けた．

参考文献

[1] Paolo Falcarin, Christian Collberg, Mikhail Atallah, and Mariusz Jakubowski. Software protection (guest editors' introduction). *IEEE Software, Special Issue on Software Protection*, Vol. 28, No. 2, pp. 24–27, Mar. 2011.
[2] Christian Collberg, Sam Martin, Jonathan Myers, and Jasvir Nagra. Distributed application tamper detection via continuous software updates. In *Proc. of the 28th Annual Computer Security Applications Conference*, pp. 319–328, Dec. 2012.
[3] Christian Collberg. The Tigress C obfuscator. `https://tigress.wtf`.
[4] Pascal Junod, Julien Rinaldini, Johan Wehrli, and Julie Michielin. Obfuscator-LLVM – software protection for the masses. In *Proc. of the IEEE/ACM 1st International Workshop on Software Protection, SPRO'15*, pp. 3–9, May 2015.
[5] Christian Collberg and Jasvir Nagra. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Program Protection*. Addison-Wesley Professional, 2009.
[6] Dominik Wermke, Nicolas Huaman, Yasemin Acar, Bradley Reaves, Patrick Traynor, and Sascha Fahl. A large scale investigation of obfuscation use in Google Play. In *Proc. of the 34th Annual Computer Security Applications Conference*, ACSAC '18, pp. 222–235, Dec. 2018.
[7] Shouki A. Ebad, Abdulbasit A. Darem, and Jemal H. Abawajy. Measuring Software Obfuscation Quality–A Systematic Literature Review. *IEEE Access*, Vol. 9, pp. 99024–99038, Jul. 2021.
[8] Sebastian Banescu, Christian Collberg, Vijay Ganesh, Zack Newsham, and Alexander Pretschner. Code obfuscation against symbolic execution attacks. In *Proc. of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16, pp. 189–200, Dec. 2016.
[9] Mariano Ceccato, Andrea Capiluppi, Paolo Falcarin, and Cornelia Boldyreff. A large study on the effect of code obfuscation on the quality of Java code. *Empirical Software Engineering*, Vol. 20, No. 6, pp. 1486–1524, Oct. 2015.
[10] Yuichiro Kanzaki, Akito Monden, and Christian Collberg. Code artificiality: A metric for the code stealth based on an n-gram model. In *Proc. of 2015 IEEE/ACM 1st International Workshop on Software Protection*, pp. 31–37, May 2015.
[11] 玉田春昭, 神崎雄一郎. Java バイトコードを対象とした命令の頻度解析による適用難読化ツールの特定. コンピュータセキュリティシンポジウム 2019 論文集, pp. 119–124, Oct. 2019.
[12] 磯部陽介, 玉田春昭. ランダムフォレストを用いた名前難読化の耐タンパ化性能の評価. 情報処理学会論文誌, Vol. 60, No. 4, pp. 1063–1074, Apr. 2019.
[13] Vijaymeena M.K and Kavitha K. A Survey on Similarity Measures in Text Mining. *Machine Learning and Applications: An International Journal*, Vol. 3, No. 1, pp. 19–28, Mar. 2016.
[14] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *Proc. of IEEE Symposium on Security and Privacy*, pp. 138–157, May 2016.
[15] Marvin Muñoz Barón, Marvin Wyrich, and Stefan Wagner. An Empirical Validation of Cognitive Complexity as a Measure of Source Code Understandability. In *Proc. of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '20, pp. 1–12, Oct. 2020.

群ロボット制御用ソフトウェアアーキテクチャの提案

An proposal of control software for multi-robot systems

箕輪 知也 * 中谷 多哉子 † 滝本 宗宏 ‡ 神林 靖 §

あらまし 群ロボットを協調動作させるためのソフトウェアアーキテクチャを提案する．様々なサービスからなる複雑なロボットシステムは，個々のロボットに比較的単純なサービスを実装し，それらを組み合わせた群ロボットとして実現することができる．我々は，このような群ロボットによって提供されるサービスを実装するためのアーキテクチャを開発した．本稿では，提案するアーキテクチャを警備群ロボットシステムに適用することで，個々のロボットの開発が単純になり，効率的に複雑なロボットシステムを開発できることを示す．

1 はじめに

我々は，2004 年より静的エージェント及び動的（移動）エージェントを使用して複数のロボットを協調動作するソフトウェアの開発に携わってきた [1]．多くの群ロボットシステムを開発する中で，共通するパターンを発見することができた。すなわち 1）全体最適化を図るエージェント，2）個々のロボットを操縦するエージェント，3）群ロボット間のコミュニケーションを図るエージェント，そして 4）個別のロボット上で動作する各種センサを管理するエージェントである．基本的なアーキテクチャを示した後に，アプリケーションに適応した例を示すことで，その有用性を提示する．このアプリケーションは，建物内で使用する警備群ロボットシステムである [2]

2 基本設計

1. 全体最適化を図る静的エージェントは，群ロボットの行動範囲を規定し，個別のロボットの標準的な移動経路を計算する．探索アプリケーションに PSO を使用する場合のグローバルベストの計算等も行う．
2. 操縦エージェントは，ロボットの制御ソフトウェア毎に製作する必要があるが，現在では ROS として標準化されている．状態遷移オートマトンとして設計できる．
3. 群ロボット間のコミュニケーションを図るエージェントは，群ロボットの場合一定の隊形を整えることが主目的となる．われわれの設計ではベクトル値をもつ移動エージェントを個々のロボットに送ることにより中央制御によらない隊形編成を可能にした．
4. 個別のロボット上で動作する各種センサ毎にサービスを提供するエージェントを提供する．センサとエージェントを対とすることにより簡潔で柔軟なロボット制御システムが構成できる．

3 屋内警備群ロボットシステム

各エージェントの構成を図 1 に示す．

*Tomoya Minowa, 放送大学

†Takako Nakatani, 放送大学

‡Munehiro Takimoto, 東京理科大学

§Yasushi Kambayashi, 日本工業大学

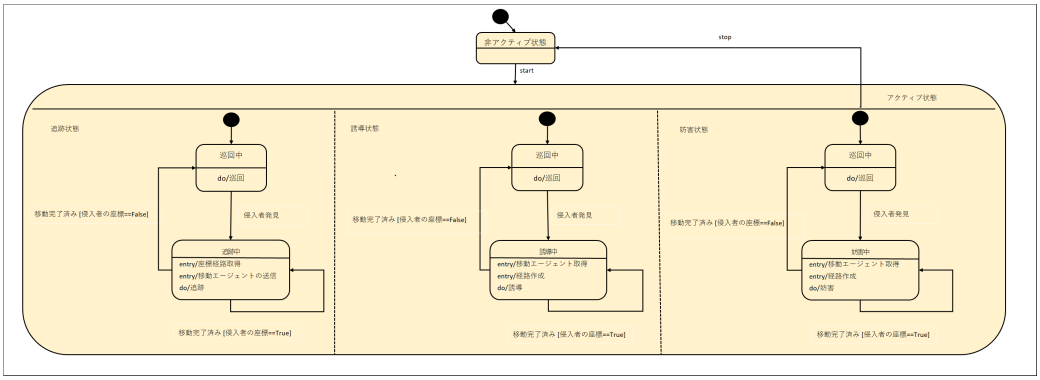


図 1　図 1 マルチエージェントシステム概要図

3.1　メインシステムエージェント

メインシステムエージェントはデータベースとして各フロアの地図をもち，ロボット上のドライバエージェントに巡回経路を与える．

3.2　ドライバエージェント

巡回状態，追跡状態，誘導状態，妨害状態とその時々の環境変化に応じて状態を遷移させつつ与えられた座標あるいは計算した座標に向けてロボットを操縦する．

3.3　コミュニケーションエージェント

われわれの群ロボットシステムでは，編成を組むために移動させたいロボットに対し相対位置のベクトル値を与えることで隊形を整える [3]．群ロボットでのコミュニケーションのほとんどは隊形を整えることに消費される．これは移動エージェントとして実装され，侵入者を発見して追跡状態にあるロボットが，巡回状態にある他のロボットを誘導状態または妨害状態へと遷移させるとともに，各状態で必要となる座標情報をベクトル値として保持する．

3.4　各種サービスエージェント

ロボットに搭載されている LIDAR センサ，周囲を観察するカメラ，衝突防止用超音波センサ等を制御するエージェントはデバイスと対になっており，ロボットの構成によって適宜入れ替えることができる．獲得した情報は，ドライバエージェントが利用できるように標準書式で共有メモリに書き込む．

4　まとめ

エージェントに基づいた群ロボット制御ソフトウェアの基本アーキテクチャを示した．実装の途中であるが，シミュレータでは有効性が認められた．

参考文献

[1] Yasushi Kambayashi and Munehiro Takimoto, Higher-Order Mobile Agent for Controlling Intelligent Robots, International Journal of Intelligent Information Technologies, vol. 1, no. 2, pp.28-42, 2005.

[2] Tadashi Shoji, Munehiro Takimoto and Yasushi Kambayashi; Capture of Multi Intruders by Cooperative Multiple Robots Using Mobile Agents, Proceeding of the Twelveth International Conference on Agents and Artificial Intelligence, vol. 1, pp.370-377, Feb, 2020..

[3] Yasushi Kambayashi, Hideaki Yajima, Tadashi Shyoji, Ryotaro Oikawa, and Munehiro Takimoto; Formation Control of Swarm Robots Using Mobile Agents, Vietnam Journal of Computer Science, vol.6,no.2, pp.193–222, 2019.

Webアプリケーション開発における欠陥再現の自動化ツールの提案

高橋 黎 * 櫨山 淳雄 † 橋浦 弘明 ‡

あらまし 本稿ではバグレポートに基づいた欠陥再現の自動化手法を提案し，ツールとして実装した結果について述べる．

1 研究の背景と目的

今日のOSSによるWebアプリケーション開発では，GitHub[1]などのソースコード管理サービスを用いて欠陥の報告や修正の確認が行われている．開発者は報告者から寄せられたバグレポートを元に欠陥の再現を行い，原因を特定した上でコードの修正を行って欠陥を除去する．しかしながら，バグレポートには特定のフォーマット等がない場合が多く，報告者によって記載内容が異なるという問題がある．加えて，開発者は欠陥の修正や確認を行う際，バグレポートに記載された再現手順を繰り返し実行しなければならない．

このような問題に対し，Chaparroら [1] [2] は，バグレポートの再現手順と期待される動作の明示的な記述が51.4%以下に留まっていることを指摘し，再現手順等の欠落を自動検出する手法を提案すると共に，得られた情報を報告者や開発者にフィードバックすることでバグレポートの品質を向上させる試みを行っている．本研究はこのようなWebアプリケーションのバグレポートの再現性の問題に対し，欠陥再現を自動化するツールを提案する．

2 提案手法とツールの実装

本研究では報告者や開発者がWebアプリケーションの欠陥の再現を自動化し，欠陥の修正前後の比較を容易にするという2つの機能によりソフトウェアの欠陥除去作業を支援する．実現にあたってはSelenium IDE[2]とGitHub API[3]を用いる．

Selenium IDEはブラウザ操作を記録・再生するためのソフトウェアである．報告者はSelenium IDEでブラウザの操作の記録を行いながら欠陥を再現することで，文章による詳細な記述を行うことなく再現手順を記録することができ，開発者は報告者から共有された記録を再生することで，再現性を担保することが可能となる．しかしながら，これらの作業は報告者・開発者共に煩雑な作業が必要となるため，本研究では全ての機能をGoogle Chromeの拡張機能として実装し，GitHub APIと連携することでこれらの作業を自動的に行うことが可能となった．本ツールを用いることで報告者がバグレポートを作成したり，開発者がバグレポートを確認したり，再現手順をダウンロードする作業を，GitHubのバグレポートページを開くことなく行うことが可能である．

実装したツールの動作例を図1，図2に示す．開発者はWebアプリケーションの画面上でGitHubに報告された欠陥を確認したり，修正前後の動作の違いを並列に並べた1つの画面上で同時に確認することができる．

*Rei Takahashi, 日本工業大学

†Atsuo Hazeyama, 東京学芸大学

‡Hiroaki Hashiura, 日本工業大学

[1] https://github.com/

[2] https://github.com/SeleniumHQ/selenium-ide

[3] https://docs.github.com/ja/rest

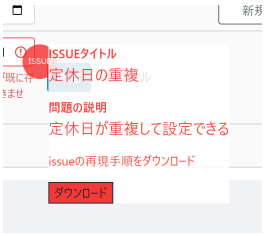


図 1　バグレポートの表示例

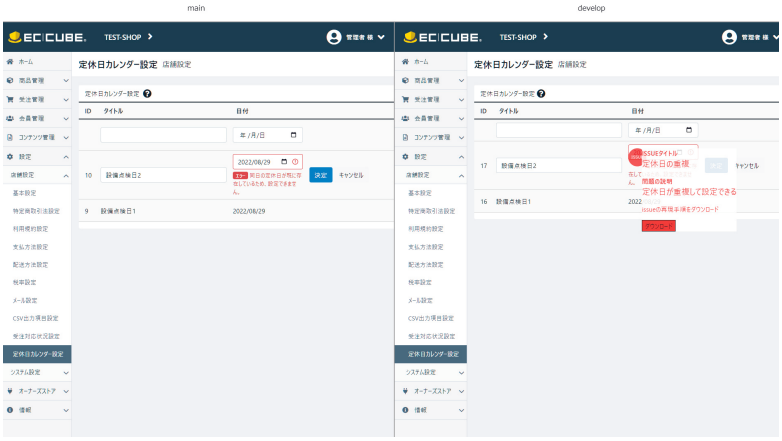


図 2　修正差分確認ページの例

3　評価

ツールの有効性を確認するために，実際の OSS アプリケーション開発におけるバグレポートを用いてケーススタディを行った．ケーススタディの対象とした欠陥は，対象のソフトウェアが OSS で開発されているソフトウェアであること，欠陥が既に修正済みであること，バグレポート内に欠陥を再現するための詳細な手順が記載されていないもの 4 件を対象として実施した．

具体的には，EC-CUBE のあるバグレポート [4]などを用いた．

評価は本ツールを用いてバグレポートに記載されたバグを再現するための手順が記録・再生することが可能か，さらに自動再現により修正前後の差分を確認することが可能かの 2 点について確認を行うこととした．確認の結果，ケーススタディで用いた全てのバグレポートにおいて欠陥の記録・再現，および修正前後の差分の確認を行うことが可能であった．

4　まとめと今後の課題

本稿では著者らが考案した欠陥自動再現・修正差分確認ツールを提案し，実装した結果について述べた．また，有用性を確認するためのケーススタディを行った結果について述べた．

今後はツールの改良とともに，実際の開発においてツールの有用性を確認していきたい．

謝辞　本研究の一部は JSPS 科研費 20K19941，21K12179 の助成を受けた．

参考文献

[1] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng, “Detecting missing information in bug descriptions,” ESEC/FSE 2017, Pages 396-407, August 2017

[2] Oscar Chaparro, Carlos Bernal-Cárdenas, Jing Lu, Kevin Moran, Andrian Marcus, Massimiliano Di Penta, Denys Poshyvanyk, and Vincent Ng, “Assessing the quality of the steps to reproduce in bug reports,” ESEC/FSE 2019, Pages 86-96, August 2019

[4] [4.2] ご注文手続き画面で h2 タグのマージンがない #5534: `https://github.com/EC-CUBE/ec-cube/issues/5534`

メソッド名の整合性評価のためのデータセット

A Data Set for Evaluating Method Name Consistency

峯久 朋也 * 阿萬 裕久 † 川原 稔 ‡

あらまし メソッドの名前は，単なる識別子というだけでなく，その処理内容を適切に表現していることが望ましい．一般にメソッド名と内容の整合性評価には人手による内容理解やレビューが必要とされるが，近年では機械学習を活用した評価手法が注目されるようになってきている．本論文ではメソッドレベルのリポジトリ FinerGit を活用し，処理内容と整合した名前を持つメソッドとそうでないメソッドのデータを多数収集して整理した結果について報告する．データセットは https://bit.ly/3QrZxUD から入手可能となっている．

1 はじめに

一般にメソッド（関数）は 1 つの処理単位として設計・実装される．その際に付けられるメソッド名は，文法上は単なる識別子であるためどのような名前であってもそれが処理内容に影響を及ぼすことはないが，現実には第三者がそのメソッドの内容を理解する上で重要な情報源となっていることが多い [1]．これが不適切な名前，即ち処理内容を適切に反映していない名前になっているとソースコードの可読性を低下させてしまい，円滑な品質管理の妨げとなってしまう [2]．

メソッド名の適切さを評価するには，そこに適切な単語や用語が使われていて，なおかつその単語・用語で意図している内容が処理内容と合致していることを判定しなければならない．それゆえ人手によるコードレビューが必要とされるが，レビュー作業そのものは決して容易で低コストなものではない [3]．この問題を解決ないし緩和するため，近年では機械学習を活用したメソッド名の自動的評価が研究されてきている [4]，[5]．先行研究では，プログラムの AST 解析やトークンの置き換えといった前処理を行った後のデータが機械学習に使用されているが，それゆえに論文ごとに公開されているデータセットも提案手法に依存したものになっている．換言すると，ある論文で公開されているメソッド名データは，あくまでも当該論文での実験の再現用であって，他の論文での手法と比較したい場合には改めてリポジトリから同じソースコードを取得することから始めなければならないという手間がある．

そこで本論文では，特別な前処理を施す前のソースコードであって，なおかつメソッド名の内容との整合性（適切・不適切）のラベルの付いたデータセットを用意することにした．多数のデータを用意するにあたり，人手で整合性をチェックするのは現実的ではないため，メソッドの変更履歴情報（特に名前の変更）に基づいてメソッド名の整合性ラベルを付与することにした．メソッド単位での変更履歴情報の収集にはメソッドレベルのリポジトリである FinerGit [6] を活用した．

2 データセットの構築

（2-1） リポジトリの収集：
まず，先行研究 [4] で使われていた 430 個の Git リポジトリを収集する．これらは Apache，Spring，Hibernate 並びに Google といった開発コミュニティで開発・保守が行なわれている Java ソフトウェアのリポジトリである．

*Tomoya Minehisa, 愛媛大学大学院理工学研究科電子情報工学専攻

†Hirohisa Aman, 愛媛大学総合情報メディアセンター

‡Minoru Kawahara, 愛媛大学総合情報メディアセンター

（2-2）メソッドレベルのリポジトリへ変換：
各リポジトリに対して FinerGit を使用し，メソッドレベルの Git リポジトリを取得する．変換後のリポジトリでは，1 つのメソッドが 1 つのファイルとして抽出され，その変更履歴を Git コマンドで追跡できる．FinerGit によって抽出されたファイルの名前はメソッドのパスや名前，引数などから自動的に付けられるため，メソッド名が変更された場合もファイルの名前変更として検出することが可能となる．
（2-3）不整合データの収集：
コミット履歴を解析し，過去にメソッド名のみが変更され，その後は現在まで名前が変更されていないメソッドを抽出する．そして，以下の条件に当てはまるものを名前が不整合なデータと見なす：

- 修正前後の名前について，短い方が長い方の部分文字列にはなっていない．
- 修正前後の名前では，それぞれの先頭単語が異なる．
- 編集距離が 3 以上である（タイプミスによる名前変更を除外するため）．
- 抽象メソッドやコンストラクタではない．

（2-4）整合データの収集：
コミット履歴を解析し，比較的多くの回数（ここでは 5 回以上とした）の内容変更はあるが，メソッド名の変更は 1 回も行われていないメソッドに注目する．そのようなメソッドの場合，名前が不適切であったとは考えにくいため，それらが抽象メソッドやコンストラクタでなければメソッド名が内容と整合したデータと見なす．

3　データセットと今後の課題

上述の手順を経て，メソッド名が処理内容と整合していると思われるメソッド 40,626 個，及び整合していないと思われるメソッド 2,662 個を取得できた．例えば，不整合データに該当するメソッド名として，次のようなものが見つかった：

- （変更前）getTwoWaySsl →（変更後）isTwoWaySsl
- （変更前）setErrors →（変更後）addError
- （変更前）checkQueryAccess →（変更後）testQueryAccess

データセットは https://bit.ly/3QrZxUD から入手可能となっている．今後は構築したデータセットを使って，新たなメソッド名評価手法の提案，並びに既存手法との比較実験を進めていく予定である．

謝辞　本研究の一部は JSPS 科研費 20H04184，21K11831，21K11833 の助成を受けたものです．

参考文献

[1] Ben Liblit, Andrew Begel, and Eve Sweetser. Cognitive perspectives on the role of naming in computer programs. In *Proc.18th Annual Psych. Prog. Workshop*, pp. 53–67, Sept. 2006.
[2] Florian Deissenboeck and Markus Pizka. Concise and consistent naming. *Softw. Quality J.*, Vol. 14, No. 3, pp. 261–282, Sept. 2006.
[3] Peter Rigby, Brendan Cleary, Frederic Painchaud, Margaret-Anne Storey, and Daniel German. Contemporary peer review in action: Lessons from open source development. *IEEE Softw.*, Vol. 29, No. 6, pp. 56–61, November 2012.
[4] Kui Liu, Dongsun Kim, Tegawendé F. Bissyandé, Tae-young Kim, Kisub Kim, Anil Koyuncu, Suntae Kim, and Yves Le Traon. Learning to spot and refactor inconsistent method names. In *Proc. 41st Int'l Conf. Softw. Eng.,*, pp. 1–12, May 2019.
[5] Son Nguyen, Hung Phan, Trinh Le, and Tien N. Nguyen. Suggesting natural method names to check name consistencies. In *Proc. 42nd Int'l Conf. Softw. Eng.*, pp. 1372–1384, May 2020.
[6] Yoshiki Higo, Shinpei Hayashi, and Shinji Kusumoto. On tracking java methods with git mechanisms. *J. Syst. Softw.*, Vol. 165, pp. 110571:1–110571:13, 2020.

非エンジニア向けのProgramming by Examplesによるデータ加工支援の試み

An attempt of data processing support by Programming by Examples for non-engineers

倉林 利行 * 丹野 治門 †

あらまし 本論文では，データ活用を高度な知識やスキルがない人でも実施できることを目的にした，データ加工支援技術を提案する．

1 はじめに

データ活用は企業の課題解決や付加価値創造に不可欠な要素となっている一方で，データ分析やプログラミング等の高度な知識やスキルが要求される．本論文では，データ活用を高度な知識やスキルがない人（以下非エンジニア）でも実施できるようにすることで，あらゆる業務におけるデータの活用を目指す．

データ活用においては，データを活用できる状態に整形するデータ加工と呼ばれるプロセスが含まれる．データ加工では，データフォーマットの変形，複数のデータの結合，欠損値の補完等が行われる．データ加工を行うためには，加工方法や加工を行うためのプログラム作成の知識が求められるため，本論文でターゲットにしている非エンジニアには困難な作業となる．

2 Programming by Examplesによるデータ加工と課題

Programming by Examples（以下 PbE）は非エンジニアでもプログラムを作成できる可能性を秘めた技術である．PbE では，実現したいプログラムの入出力例からその入出力例を満たすプログラムを自動生成する．PbE をデータ加工に適用することで，非エンジニアでもデータ加工前後の入出力例を作成するだけで，加工プログラムを自動生成できる [1]．PbE によるデータ加工を図 1 に示す．図 1 は Purchase Table 内の購入情報に対して，Purchase Table 内の購入者の情報を紐づけることを目的にした加工の例である．そのためには Customer Table と Purchase Table（右）を，cus_id をキーにして右結合する必要があるが，加工の知識がない非エンジニアには困難である．PbE によるデータ加工では，ユーザははじめに加工したいデータの一部に対する加工後のデータ例を作成し，加工したいデータの一部とその加工後のデータ例を入出力例として PbE によるデータ加工技術に与える．PbE によるデータ加工技術はその入出力例を満たす加工プログラムを生成し，そのプログラムを加工したいデータに対して実行して加工後のデータを得る．最後にユーザはその加工後のデータを確認し，問題がなければ受理し，不適切な箇所があれば入出力例を作り直して上記のプロセスを再度実行する．

PbE によるデータ加工には，適切な入出力例の作成が非エンジニアにとって負担になるといった課題が存在する．正しいプログラムを生成するためには，実現したい加工をすべて反映させた入出力例を作成する必要がある．例えば図 1 の Customer Table において，一行目だけのデータ（0001, Taro）のみを与えた場合，`pandas.merge(“Customer Table”, “Purchase Table”, on=”cus_id”, how=”outer”)` といった外部結合を実行するプログラムも入出力例を満たしてしまう．このようにデータ加工の知識がない非エンジニアにおいて，適切な入出力例の作成は

*Toshiyuki Kurabayashi, NTT ソフトウェアイノベーションセンタ

†Haruto Tanno, NTT ソフトウェアイノベーションセンタ

図 1　PbE によるデータ加工

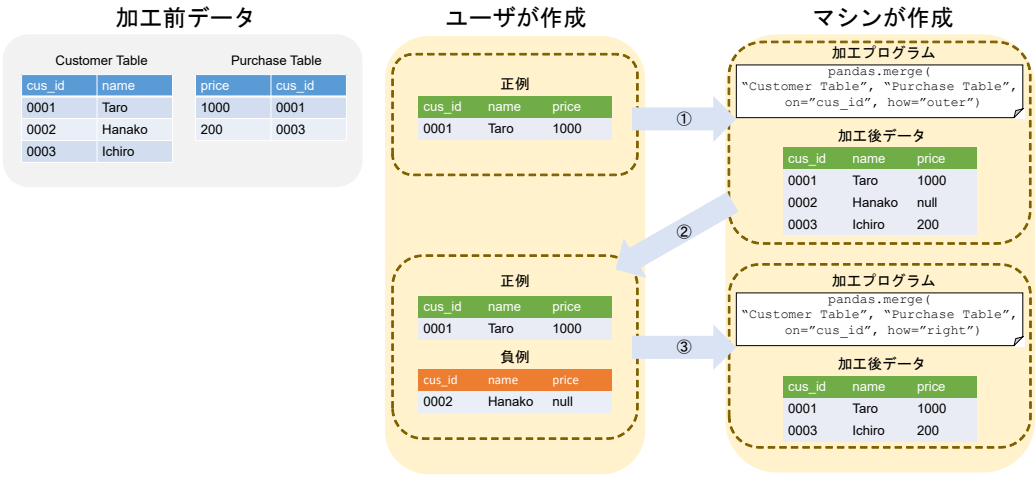


図 2　提案手法によるデータ加工

困難である．

3　提案手法

提案手法ではユーザとマシンがインタラクティブにデータ加工を行うことで，非エンジニアでも容易にデータ加工を実現できるようにする．提案手法によるデータ加工を図 2 に示す．提案手法では以下の手順でデータ加工を行う．ただし正例は加工後のデータに含まれてほしいデータ，負例は含まれてほしくないデータを意味する．

1. ［ユーザ］加工後データの一部を，正例として示す
2. ［マシン］正例を含むようなデータ加工を行うプログラムを自動生成し，そのプログラムを実行することで得られる加工後のデータをユーザに示す
3. ［ユーザ］加工後のデータをレビューし，不適切な箇所があれば修正し，その修正箇所を正例に加える．また含まれてほしくないデータについては負例に加える．ユーザが所望する加工後データが入手できるまで，2 と 3 を繰り返す．

提案手法では，ユーザは加工後のデータの一部を示すだけで良いため，入出力例を作成する必要がない．またマシンが生成した加工後のデータに対して，不適切な箇所を修正するだけで良いため，加工の知識についても不要である．よって提案手法は非エンジニアでも容易にデータ加工を可能にする技術だと言える．

参考文献

[1] Zhongjun Jin, Michael R. Anderson, Michael Cafarella, and H. V. Jagadish. 2017. Foofah: Transforming Data By Example. In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17). Association for Computing Machinery, New York, NY, USA, 683–698. https://doi.org/10.1145/3035918.3064034

README における項目と説明文の一貫性の分析

An analysis of consistency between heading and content in README file

石岡 直樹 * 伊原 彰紀 †

あらまし README に記述すべき項目について調査した研究が多数報告されているが，具体的な説明内容の調査は数少ない．本研究では，開発者に対してプロジェクトやソフトウェアの特徴を踏まえた README の作成方法を推奨するための第一歩として，README の項目に対する説明文の一貫性を分析した．その結果，説明文と一貫性がある項目を明らかにした．

1 はじめに

多くのオープンソースソフトウェア（OSS）プロジェクトは，利用者およびプロジェクトに興味を持つ開発者に向けて，ソースコードと併せて README と呼ばれるドキュメントを公開する．README は，ソフトウェアを正しく利用するための情報や開発を促進するための情報を，他の開発者や利用者に共有する重要なドキュメントであり，2017 年に行われた調査 [1]でも，開発者は README を重要視していることが報告されている．しかし，README が不完全であることが指摘されている．Prana ら [1] は，README に記述される項目を調査しており，説明文に基づき見出しを自動的にラベル付けする手法を評価したが，項目によって F1 値が異なることがわかった．これは言い換えると，README の項目のみを分析しても説明文が異なるため，README に記述すべき項目は明らかにできていないと示唆する．本研究では，同一項目でも説明文の内容が異なるのか否かを明らかにするために，README の項目に対する説明文の一貫性を調査する．

2 分析

2.1 データセットと手法

本研究では，Libraries.io [2]で公開される JavaScript ライブラリを対象に，Ikeda らと同様の手順で 250,075 件の README を対象とする．各 README 中の#や##からはじまる見出しを従来研究 [2] で示された 20 項目に分類し，それぞれの項目を説明文に基づき自動的にラベル付けするモデルを，ランダムフォレストを用いて構築し，評価した．本研究は Prana らの対象プロジェクト数に比べて極めて多くのプロジェクトを対象にし，また項目数もさらに細分化している．評価指標には一般的に機械学習モデルの評価に用いられる適合率・再現率・F1 値を用いる．この時，適合率・再現率ともに高い精度で予測できる項目と説明文は一貫性があり，共通の内容が説明文に記述されていると判断する．

2.2 結果と考察

予測結果から，LICENSE や CONTRIBUTE などの説明文と一貫性がある項目，また USAGE や OPTION などの説明文と一貫性がない項目が存在することがわかった．図 1 はモデルの予測結果から得られた混同行列を示す．横のカラーバーは色の濃淡と値の対応を示しており，色が濃いほどサンプル数が多いことを示す．各マス

*Naoki Ishioka, 和歌山大学

†Akinori Ihara, 和歌山大学

[1]https://opensourcesurvey.org/2017/

[2]https://libraries.io/

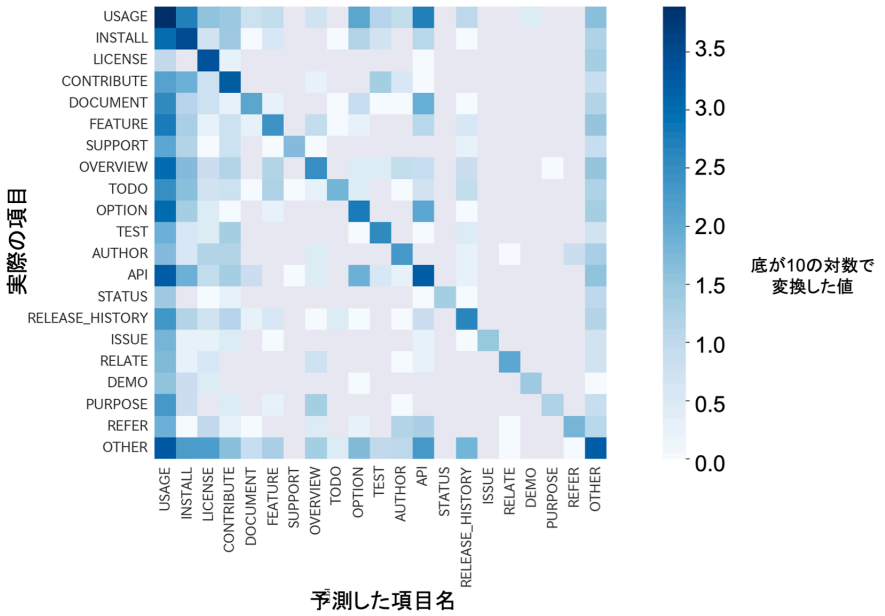


図 1　モデルの予測結果から得られた混同行列．各ラベルは項目名を表す．

の値は，サンプル数の違いによって値が潰れることを防ぐために，底が 10 の対数で変換している．使用方法を説明する項目である USAGE の列に着目すると，USAGE と予測した 16,590 件中，API，OVERVIEW，OPTION の項目をそれぞれ 1,671, 1,007 件, 976 件 USAGE であると誤予測している．また，USAGE の行に着目すると，USAGE の項目 9,003 件中，INSTALL や API，OPTION であると，それぞれ 499, 494, 123 件で誤予測しており，USAGE の説明文に他の項目の説明文が混在していることが示唆される．USAGE の説明文を目視で確認したところ，INSTALL や OPTION などの内容が USAGE の項目に含まれていることを発見した．

3　おわりに

本研究では，README における項目と説明文の一貫性を明らかにした．LICENSE や CONTRIBUTE などの説明文と一貫性がある項目は，多くの開発者が共通した内容を説明文に記述している．一方，USAGE などの説明文と一貫性がない項目には他の項目の内容が混在している．従来研究では見出しのみを調査していたが，README に記述すべき内容を明らかにするためには，見出しだけでなく説明文も併せて分析する必要がある．今後は，README の更新過程において説明文にどのような内容が追加・変更されたかを分析することで，README に記述すべき内容を明らかにする．

参考文献

[1] Gede Artha Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the Content of GitHub README Files. Empirical Software Engineering. vol.24, no.3, 1296-1327, 2019.

[2] Shohei IKEDA, Akinori IHARA, Raula Gaikovina KULA, and Kenichi MATSUMOTO. An empirical study of readme contents for javascript packages. IEICE Transactions on Information and Systems, Vol. E102.D, No. 2, pp. 280-288, 2019.

ゲーミフィケーションを用いたC言語の文法やアルゴリズムの学習支援アプリケーションCode Quizの提案

Proposal of Code Quiz, an application to support learning C syntax and algorithms using gamification

谷本 嵩晃 * 崔 恩瀞 † 水野 修 ‡

あらまし 本稿では，ゲーミフィケーションを用いて C 言語の学習を支援するアプリケーション Code Quiz を提案する．また，予備実験を行い，Code Quiz が学習のモチベーションを継続させることを確認した．

1 はじめに

情報工学において，プログラミング学習は非常に重要である．しかし多くの初学者にとって，プログラミング学習は困難である反面，アルゴリズムのみの学習は難しくない [1]．つまり，初学者にとって最も困難なことは，各プログラミング言語の文法を勉強し，さらにアルゴリズムを実装することである．現在までにプログラミング学習を支援するための様々なツールが存在する．例えば,「pgtracer」は，初学者を対象にした穴埋めを用いた，C++および Java プログラミング教育支援ツールである [2]．しかし，pgtracer を用いて学習する場合，一問あたりにかかる時間が 4~5 分程度ということが報告されている．また，学習頻度やモチベーションに関する調査は行われていない．本研究は，初学者が授業などで学習する文法を演習問題で予習，復習することで，プログラミングに対するモチベーション維持を目標としている．本研究の目的を達成するために，本研究では，プログラミング学習を支援するアプリケーション Code Quiz をネイティブアプリケーションとして開発した．

2 Code Quiz の概要

Code Quiz では，ユーザのモチベーションを向上させるために，ゲーミフィケーションという手法を用いている [3]．ゲーミフィケーション要素として，レベル，バッジ，ポイントを採用した．各問題には，正解時に獲得できるポイントを設定しており，一定のポイントを獲得すると，レベルが上がるシステムになっている．現在のレベルと，所有しているポイントは，図 1 の Menu Page で確認できる．特定の条件を満たすと，バッジを獲得できる．獲得したバッジは，図 1 の Badge Page で確認できる．メニュー画面では，日々の成果を確認できるように，カレンダーを設置した．Code Quiz では，初学者だけでなく，中級者，上級者のプログラミング学習も支援している．具体的には，初級の問題では，適切な型を選択する問題等を，上級の問題では，二分木探索のアルゴリズムを作成するための，穴埋め問題等を提供している．

Code Quiz の支援対象は主に，本学の学生であるため，学習対象言語は C 言語とした． 図 1 に Code Quiz の概要を示す．初回起動時ユーザは，Register Page でユーザ登録を行い，登録後 Menu Page に移動する．Menu Page から Select Page に移動し，Select Page で問題難易度が選択できる．その後，Question Page で 7 問解き，結果を Result Page で確認する．

*Takaaki Tanimoto, 京都工芸繊維大学

†Eunjong Choi, 京都工芸繊維大学

‡Osamu Mizuno, 京都工芸繊維大学

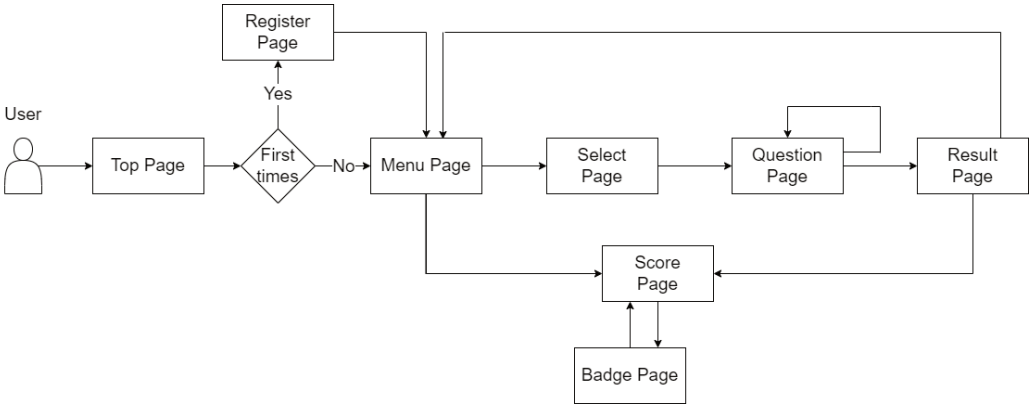


図 1　Code Quiz の概要

3　予備実験

Code Quiz の有効性を確認するために，5 人の被験者に約 1 週間程度 Code Quiz を使ってもらった後に，アンケートを行った．アンケートの結果から，Code Quiz を使うことで，プログラミング学習へのモチベーションを継続させることができたが，向上させることができなかったことがわかった．その原因として，被験者が全員 C 言語の経験年数が 3 年以上であったこと，ユーザが回答する際のユーザビリティが良くなったことやバッジシステムを有効活用できていなかったことなどが考えられる．一方で，手軽にプログラミング学習を行えるという印象を与えることができた．

4　まとめと今後の課題

本研究では，ゲーミフィケーションの手法を用いたプログラミング学習支援アプリケーション Code Quiz を開発した．予備実験の結果，Code Quiz を用いることでプログラミング学習に対するモチベーションを向上させることはできなかったが，継続できることが明らかになった．Code Quiz にはユーザビリティの面で多くの改善点が見られたが，手軽にプログラミング学習を行えるという印象を与えることができた．この点から，Code Quiz は，プログラミングに対するモチベーションを向上させる可能性を含んでいると言える．

今後は，まずユーザビリティの改善の予定である．具体的には，C 言語において頻繁に使われる記号 (*,=,+等) のショートカットの作成，ソースコードに対して，シンタックスハイライトを適用する等を行う予定である．また，今回はモチベーションに関して，アンケートを行ったが,「ログイン回数」,「解答問題数」,「ログイン時間」などより定量的な尺度を用いて評価を行う予定である．

謝辞　本研究は JSPS 科研費 18H04094, JP19K20240, JP21H03416 の助成を受けた．また貴重な助言を頂きました同志社大学 槇原 絵里奈 助教に深く感謝致します．

参考文献

[1] Mark Guzdial. Why is it so hard to learn to program? In Greg Wilson. Mowery Andy Oram and Richard R. Nelson, editors, *Making Software*, chapter 7, pp. 111–124. O'Reilly Media, Inc., 2010.

[2] 掛下哲郎, 柳田崚, 太田康介. 穴埋め問題を用いたプログラミング教育支援ツール pgtracer の開発と評価. 情報処理学会論文誌教育とコンピュータ, Vol. 2, No. 2, pp. 20–36, 2016.

[3] 井上明人. ゲーミフィケーション-< ゲーム > がビジネスを変える. NHK 出版, 2012.

GitHubにおける模範プロジェクトの検出とその成長パターンの分類に向けて

Toward detecting the model projects and categorizing their growing patterns from GitHub

開出 凱斗 * 玉田 春昭 † 戸田 航史 ‡ 中村 匡秀 §

あらまし ソフトウェア開発において，GitHubを用いた開発が主流となって久しい．そのため，開発中の様々な履歴データはGitHubに日々蓄積されており，それらをもとに様々な角度からの分析が行われている．本稿では，GitHubからのデータを用いて，プロジェクト運営の良し悪しの測定を試みる．そのためにまず，プロジェクトの典型的なパターンを検出し，それらのパターンから良し悪しを推定する．

1 はじめに

より良いプロジェクト運営のために，プロジェクト運営を都市に見立てる方法，Project as a City（PaaC）が提案されている [1]．PaaCでは，プロジェクトの現在の状況を把握し，理想との差を検出することで進むべき方向を導出することを目指している．しかし現状では，プロジェクトの現在の状態を把握する方法は提案されているものの，理想とは何かや，理想との差については議論されていない．そこで本稿では，理想となるプロジェクトの検出を目指す．

しかし，理想となるプロジェクトと言っても，プロジェクトの内容や，規模，評価の観点から，各プロジェクトの理想像は異なるものになると考えられる．そのため，プロジェクトの理想像を唯一に定義できるものではない．一方で，GitHubで公開されている大量のプロジェクトの中には，様々な観点で理想像となり得るプロジェクトが存在すると期待できる．そこで本稿ではまず，いくつかのプロジェクトから典型的なパターンが確認できるかを確かめる．典型的なパターンが確認できれば，より対象を広げ，そのパターンの意味の推定や別のパターンの検出を行っていく．

2 キーアイデア

本稿では，いくつかのプロジェクトで典型的なパターンが見つかるかを確認する．そのために，対象プロジェクトからある観点における時系列データを取り出し，折れ線グラフを描画する．そのグラフから典型的なパターンを目視で確認する．今回は時系列データとして過去5年間の月毎の残りIssue数の推移を用いる．

表1 対象としたプロジェクトとそのデータ

Repository	Commits	# of Star	Issues	PRs
appscale/gts	10,307	2,412	999	2,231
azure/azure-sdk-for-java	24,729	1,575	8,231	22,428
broadleafcommerce/broadleafcommerce	17,761	1,544	1,437	1,274
movingblocks/terasology	11,900	3,395	2,018	3,030
opentripplanner/opentripplanner	16,269	1,752	2,623	1,810
osmandapp/osmand	79,735	3,309	9,288	5,269
robolectric/robolectric	12,539	5,518	2,578	4,964
signalapp/signal-android	10,565	22,739	9,750	2,603
spring-projects/spring-framework	24,984	49,018	22,403	3,881
spring-projects/spring-integration	11,051	1,320	591	3,261

3 ケーススタディ

3.1 対象データ

まず，GHS (GitHub Search) [2] を用いて，GitHubでホストされているプロジェクトの中から一定の条件のものを取得した．設定した条件は，コミット数10,000以

*Kaito Kaide, 京都産業大学大学院

†Haruaki Tamada, 京都産業大学

‡Koji Toda, 福岡工業大学

§Masahide Nakamura, 神戸大学

上，スター数 1,000 以上，イシュー数 500 以上，プルリクエスト数 100 以上である．結果として得られたプロジェクト一覧から表 1 に示す 10 プロジェクトを選択した．これらのプロジェクトから過去 5 年間の月毎の残り Issue 数の推移を確認する．なお，全てのプロジェクトはこれ以前から活発に開発されている．

3.2 結果

表 1 の各プロジェクトから 2017 年 1 月から 2022 年 8 月まで毎月の残り Issue 数の推移を描画したグラフを図 1 に示す．残り Issue 数はプロジェクトごとに正規化している．図 1 から急激に Issue 数が下がっているパターンが確認できる．一般的に著名なプロジェクトには多くの利用者が Issue を登録するため，右肩上がりで残り Issue 数が増加していくと考えられる．そのため，Issue 数が徐々に増加していくことは著名なプロジェクトの日常的なことであると考えられる．一方，Issue 数の急激な減少は短期間に大量に Issue をクローズしていることになる．これは，集中的に Issue に対応しているか，重複 Issue や解決済みだが未クローズの Issue の片付けのいずれかであると考えられる．実際にこのパターンのプロジェクトのリリース日時を確認すると，ほとんどがリリースの前に Issue をクローズしていた．唯一 `robolectric/robolectric` のみが 4.0 のリリース（2018-10-26）後に大量に Issue をクローズしていたことがわかった．

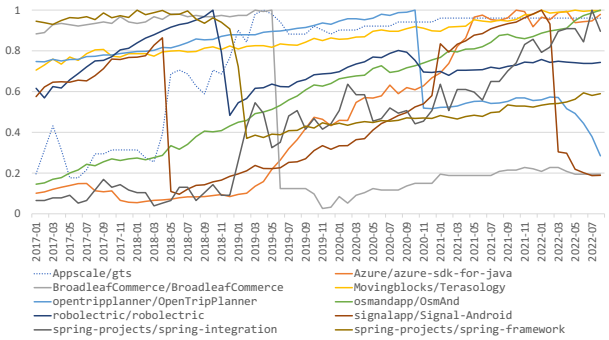


図 1　各プロジェクトの残り Issue 数の推移

一方，点線で表されている `AppScale/gts` に着目すると，2017-01 からと，2018-03 からなど何度か大きく Issue 数が増加している．このプロジェクトのリリース日時を確認すると，大きく Issue 数が増加した後にはバージョンの更新があり，多くの機能が追加されていたことがわかった．このことから，次のリリースに向けての Issue を登録したのだと考えられる．また，2021 年 1 月から Issue 数が停滞していることがわかる．最終のリリースは 2019-10-21 であり，最後に Issue がクローズされた日が 2019-11-05 であることを鑑みると，この頃から開発があまり行われなくなったと考えられる．

4　まとめ

本稿では，PaaC の考えをもとに，プロジェクトの理想像を設定するため，現状の著名なプロジェクトがどのようなパターンを持つのかを確認した．そのために GitHub から一定の開発履歴を持つ 10 個のプロジェクトを選択し，月毎の残り Issue 数の推移を折れ線グラフにプロットした．結果として，リリース前後での不要な Issue の棚卸しパターン，平常パターンの 2 つが見られた．それぞれのパターンが現れた時期のコミット数やリリースを確認すると，概ね想定通りの理由であったことが確認できた．今後は，対象プロジェクトを拡充し，他のメトリクスの推移においても，同様にパターンの検出を行っていく．

謝辞 この研究は JSPS 科研費 20K11761 の助成を受けた．

参考文献

[1] K. Toda, H. Tamada, M. Nakamura, and K. Matsumoto. Characterizing project evolution on a social coding platform. In *Proc. IEEE/ACIS SNPD 2019*, pp. 525–532, July 2019.
[2] O. Dabic, E. Aghajani, and G. Bavota. Sampling projects in github for msr studies. In *Proc. IEEE/ACM MSR 2021*, pp. 560–564, June 2021.

育成の観点を取り入れたプロジェクト管理ゲーム

A Project Management Game with Engineer Development Perspectives

山形 宥太[*] 西浦 生成[†] 笹倉 万里子[‡] 門田 暁人[§]

あらまし 急速な技術革新を伴う今日のソフトウェア開発では，開発者に新技術を学んでもらいながらプロジェクトを成功へと導いていくことが必須である．本研究では，実務者や学生に，育成を伴うソフトウェア開発プロジェクト管理を体験し，その重要性について学んでもらうことを目的とする．昨今では育成要素のあるゲームが流行していることから，開発者育成を伴うプロジェクト管理ゲームとして実現することで，楽しみながら体験学習できると期待される．本稿では，ゲーム要件の概略を述べる．

1 はじめに

近年，デジタルトランスフォーメーション（DX）によるビジネス変革のために，AI や IoT 等の先進技術を用いた情報システムの開発が求められている．また，ソフトウェア開発技術の変化も急速であり，様々な言語，フレームワーク，ライブラリが日々新たに出現し，技術者に求められるスキルは日々変化している．例えば，IT エンジニア向け転職・就職・学習プラットフォーム paiza における 2014 年の調査では，67%のユーザが 1 週間に 5 時間以上を新たな技術の勉強に費やしている[1]．また，IT 人材白書 2020 の調査では，先進 IT 従事者の 1 週間あたりの勉強時間は約 2.7 時間であった[2]．今日のソフトウェア開発では，開発者に新技術を学んでもらいながらプロジェクトを成功へと導いていくことが必須である．

本研究では，実務者や学生に，育成を伴うソフトウェア開発プロジェクト管理を体験し，その重要性について学んでもらうことを目的とする．新しい技術を身につけなければ多様な案件を受注し成功に導くことができず，ソフトウェア企業としても成長していくことは難しいと考えられる．昨今では多種多様なキャラクタを成長させながらゲームを有利に導いていく「育成」要素のあるゲームが流行していることから，開発者育成を伴うプロジェクト管理ゲームとして実現することで，多くの若者が楽しみながら体験学習できることを目指す．また，近年，ゲーミフィケーション，エデュテイメントというキーワードに代表されるように，娯楽を通した学びの有効性が示されており，ゲームによる学びは効果が大きいと期待される．

2 ゲームのデザイン

従来，ソフトウェア開発管理ゲーム教材が一部の大学で活用されており，米国カリフォルニア大学で開発された SIMSE[**]は開発者割り当てと進捗管理について学習できる．

[*] Yuta Yamagata, 岡山大学工学部情報工学科
[†] Kinari Nishiura, 岡山大学学術研究院自然科学学域
[‡] Mariko Sasakura, 岡山大学学術研究院自然科学学域
[§] Akito Monden, 岡山大学学術研究院自然科学学域
[**] https://www.ics.uci.edu/~emilyo/SimSE/

本ゲームにおいても，開発者割り当てと進捗管理を体験できるようにする．

また従来，学習の要素を取り入れたソフトウェア開発プロセスモデルが提案されている．[3][4]，本ゲームにおいても，個々の技術についての学習の進展に伴って，その技術を必要とする作業の生産性が向上する仕組みを取り入れる．さらには，開発者の増加に伴う生産性の低下，開発の途中で人員を投入することによる生産性の低下も考慮する．

以下，本ゲームの要件の概要を述べる．

- ゲームの目的は，多数の開発者を育成しながら多数のプロジェクトを成功に導くことでソフトウェア開発企業を成長させていくこととする．
- ターン制とし，1 ターン（=1 週間）ごとに各開発者が取り組む行動を選択する．
- 納期，各技術の必要スキルレベル，報酬がそれぞれ設定された（1 件以上の）開発案件を企業が受注することで，プロジェクトとして開発を進めることができる．
- プロジェクトを成功させることで企業の信用度が上がり，より大きな案件を受注できるようになる．
- ゲーム開始時から雇用している開発者に加えて，新たに開発者を雇うことができる．
- 各開発者には月給，技術ごとのスキルレベル等が設定されており，スキルレベルの高さに応じて月給も上がる．
- 雇用している開発者を受注したプロジェクトの開発に割り当てた状態でターンを進めることで，開発者のスキルに応じてプロジェクトの進捗が進む．
- １プロジェクトあたりの開発者の人数が多いほど，開発生産性は低下する．
- 開発の途中で開発者をプロジェクトに投入した場合，開発生産性は低下する．
- 開発者に開発作業ではなくトレーニングを割り当てることで，1 ターンを消費してスキルレベルを上げることができる．
- プロジェクトが成功したとき，そのプロジェクトの開発に携わった開発者は OJT によりスキルレベルが少し上昇する．
- 開発案件の進行中に選択肢式のトラブルイベントが発生し，正しい選択肢を選ばないと進捗が遅延する．

3　おわりに

現在, 前章で述べたゲーム要件を元に, JavaScript を用いてゲームの開発を進めている．また，受注済み案件の進捗度を視覚的にわかりやすくするための折れ線グラフの追加などを行っている．今後はゲームの完成に向けて，プロジェクト管理を学ぶために必要な追加の要件の検討を行いながら開発を進めていく予定である．

4　参考文献

[1] paiza 開発日誌, “IT エンジニアの勉強時間は月 20 時間以上！？好きなことして生きる方法,” 2014 年 9 月 16 日, https://paiza.hatenablog.com/entry/2014/09/16/IT エンジニアの勉強時間は月 20 時間以上！？好きなこ

[2] 独立行政法人情報処理推進機構, “IT 人材白書 2020, 2020. https://www.ipa.go.jp/jinzai/jigyou/about.html

[3] Noriko Hanakawa, Hajimu Iida, Ken-ichi Matsumoto, Koji Torii, “Genereation of Object-Oriented Software Process Using Milestones,” Int. J. Softw. Eng. Knowl. Eng., Vol.9, No.4, pp.445-466, 1999.

[4] Noriko Hanakawa, Ken-ichi Matsumoto, Koji Torii, “A Knowledge-Based Software Process Simulation Model,” Ann. Softw. Eng., Vol.14, No.1-4, pp. 383-406, 2002.

ロジックモデルからステークホルダーバリューネットワークへの変換による価値循環の抽出

Extracting Value Cycles by Transforming Logic Models into Stakeholder Value Network Analysis

丹羽 南* 山田 勉† 青木 善貴‡

あらまし 事業活動を通して社会課題解決に取り組む場合，協働するステークホルダーに対し社会的・経済的価値の還元を受けられることを示し理解を得ることが重要である．本稿ではロジックモデルで定義した事業構想をステークホルダーバリューネットワークに変換し分析を行うことで事業における価値循環を抽出する手法を提案する．

1 はじめに

企業が事業活動を通して社会的課題の解決に取り組む場合，多種多様なステークホルダーとの協働が必要になる．また，社会的課題の解決には長期間かかることが多い．事業を長期間維持するためには，事業構想の段階でステークホルダーに対し事業活動を通して社会的・経済的価値の還元を受けられることを示し理解を得ることが重要である．

事業活動が社会的インパクトを達成するまでの道筋は，McLaughlin ら[1]と Knowlton ら[2]のロジックモデルによってモデル化されることが多い．しかし，ロジックモデルは投入する資源，それを元にした活動とその結果得られる製品やサービス，そこから波及する個人や環境の変化・効果，それらがもたらす社会的な変化や効果を一方向に表現するため，ステークホルダーへの還元を示すことができないという課題がある．

一方，Feng ら[3]のステークホルダーバリューネットワーク分析（以下，SVN 分析）は，ステークホルダー間の関係を社会的交換理論に基づいた価値交換とみなすことで多重有向グラフとしてモデル化し，重要なステークホルダーや価値循環を定性的・定量的に認識しようとするものである．SVN 分析を行うにはステークホルダーへのヒアリング等のデータ収集が必要であり，一般には大規模・複雑なプロジェクトにおける状況分析やイシュー・マネジメントに用いられる．

本稿では，ロジックモデルを変換し SVN を作成することで，事業構想におけるステークホルダーへの社会的・経済的価値の還元を抽出する手法を提案する．提案手法により，事業構想段階においてもステークホルダーに対し事業における価値循環を視覚的に示すことができるようになる．

2 提案手法

本稿では，地方都市におけるモビリティデータ活用構想の事例をもとに，ロジックモデルで示した事業構想における価値循環を SVN 分析により抽出する手法を提案する．この事業構想は，住民の高齢化や中心市街地の空洞化などの課題を抱えている地方都市に

* Minami Niwa, BIPROGY 株式会社
† Tsutomu Yamada, BIPROGY 株式会社
‡ Yoshitaka Aoki, BIPROGY 株式会社

おいて，官民連携事業体を設立しステークホルダーが持つデータを集約・分析することで，地域公共交通機関の利用や中心市街地への集客を促進する住民向けサービスを提供するものである．事業構想をロジックモデルとして記載したものが図１である．

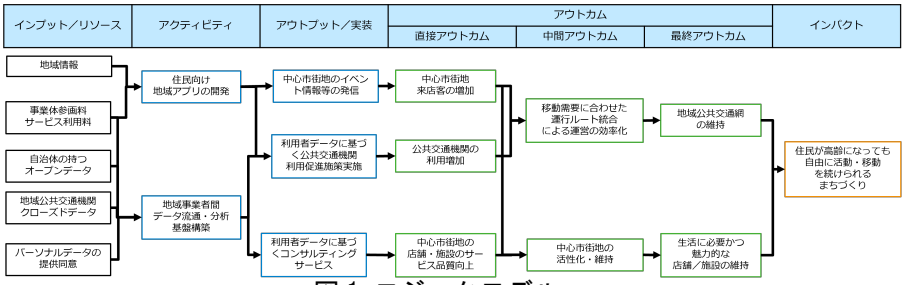


図１ ロジックモデル

次にロジックモデル上のインプット，アウトプット，アウトカムをステークホルダー間の価値交換と捉え，供給元・供給先となるステークホルダーを特定する．これによりロジックモデルの要素を SVN 上の辺に変換でき，図２のように SVN を作成することができる．

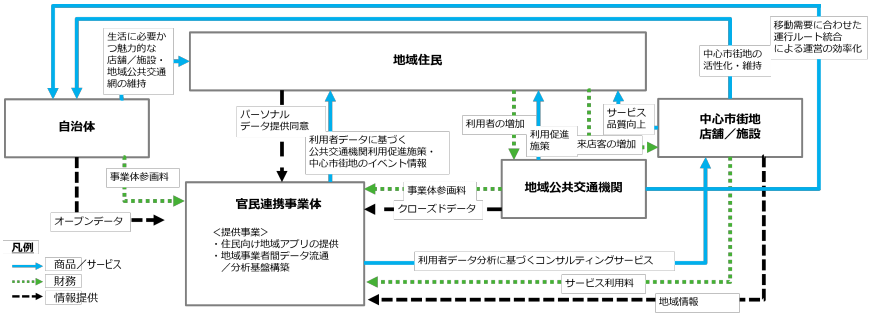


図２ ステークホルダーバリューネットワーク

作成した SVN により全てのステークホルダーが事業を通して価値の還元を受けることを確認でき，事業に参加する意義を示すことができた．また，価値循環がない場合は，事業構想を見直す必要があると判断できる．

3　まとめ

本稿では，ロジックモデルにより定義した事業構想を SVN に変換し価値循環を抽出する手法を提案した．今後は提案手法の有効性を示すため，複数の事例への適用および，評価を行う．また，SVN 分析では価値交換を定量的に分析することもできるため，事業構想の定量評価などへの応用を目指す．

参考文献

[1] McLaughlin, J. A. and G. B. Jordan: Logic models: a tool for telling your programs performance story, Evaluation and Program Planning, 22, 65–72 (1999)
[2] Knowlton, L. W. and C. P. Cynthia: The Logic Model Guidebook: Better Strategies for Great Results, Sage Publications, (2008)
[3] Wen Feng let.al,: Understanding the impacts of indirect stakeholder relationships – Stakeholder value network analysis and its application to large engineering projects, MIT Sloan Research Paper No. 4978-12, 2013.

2つのWebアプリケーション間の類似する操作対象の対応関係抽出

Extracting mapping of similar operation targets between two web applications

内田 啓太* 石尾 隆† 嶋利 一真‡ 松本 健一§

あらまし UIテストのテストスクリプトの設計を効率化することを目的に，他のシステムの類似機能から同じ操作対象を推定する手法を提案する．4種類の似たUIを用意して操作を記録し，操作前後で変化したDOMの差分の類似度を用いて手法の有効性を検証した．類似度を元に操作対象の対応関係を評価した結果，操作対象は一致したが操作は一致しなかった．

1 はじめに

ソフトウェアのテストの1つに，Webアプリケーションを実際に操作し，期待通りの画面になっているかを検証するUIテストがある．UIテストはSeleniumなどのツールを用いてクリックや文字入力等の操作を自動化することで効率化できるが，そのためには操作の種類と順序，操作対象となる要素を記したテストスクリプトを作成する必要がある．テストスクリプトの設計や再構築は開発者の経験とノウハウに任せられており，テスト項目の洗い出しやテスト方法の検討に時間を要する [1]．

過去に作成されたシステムが保有するUIのテストスクリプトから，機能が類似した箇所を再利用できれば，さらなるテストの効率化が見込める．そこで本研究では，UIテストを実施したいシステムとは別のシステムにある，類似した機能を持つ操作対象を見つける方法を提案する．具体的には，操作を行ったときのDOMの変化差分同士の類似度を基準に対応関係を設定する．試作した手法を評価するために，ログインフォームのUI例を対象として対応関係の一致度を確認する．

2 提案手法

本研究では，類似機能の対応関係を判定するために，DOMの変化差分を比較する．DOMとはWebページのUIの情報が記されたツリー状の構造体で，ブラウザはHTMLをDOMとして解釈して表示している．クリックなどのUI操作によってDOMが変更された場合，その差分はDOMツリーの枝の追加や削除によって表現される．またクリック等の入力を操作，クリックした要素を操作対象と呼ぶ．ここで，ある操作対象の操作によって生じたDOMの変化差分を2つのシステム間で比較し，差分が類似したものを，操作が類似するものとして操作対象同士を対応付ける．

本研究の1つ目のコアなアイデアは，2つのシステムをそれぞれ操作することにより取得したDOMの差分を比較し，差分情報がよく類似しているものから順に同じ操作対象だと推定することである．2つ目のアイデアは操作対象を基準にして差分を比較することである．DOMツリーをそのまま比較するのではなく，UIの操作対象をツリーの根とみなした構造に変換して比較することで，差分箇所の操作対象からの相対的位置を考慮する．差分そのものは，比較が容易となるようにツリー構造で表現する．

*Keita Uchida, 奈良先端科学技術大学院大学
†Takashi Ishio, 奈良先端科学技術大学院大学
‡Kazumasa Shimari, 奈良先端科学技術大学院大学
§Kenichi Matsumoto, 奈良先端科学技術大学院大学

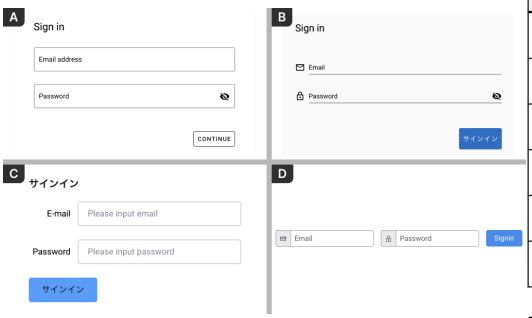


図 1: 4 種類のログインフォーム UI

UIペア	一致した操作対象	最小TED	操作番号 Xn	操作番号 Yn	一致
A, B	メールアドレス欄	28	1	1	○
	パスワード欄	49	3	3	○
A, C	メールアドレス欄	12	1	1	○
	パスワード欄	49	4	3	×
A, D	メールアドレス欄	12	1	1	○
	パスワード欄	15	4	3	×
B, C	メールアドレス欄	10	1	1	○
	パスワード欄	48	4	3	×
B, D	メールアドレス欄	10	1	1	○
	パスワード欄	14	4	3	×
C, D	メールアドレス欄	0	1	1	○
	パスワード欄	5	2	2	○

図 2: UI ペア毎の，TED が最小で一致した操作対象ペアとその操作番号

3 ケーススタディ

GitHub の検索機能で「vue ui」を検索して見つかった UI ライブラリのうち，よく利用されているであろうスター数上位 3 つを使用して，同一機能で異なるデザインを持つ 4 つのログインフォームを用意した (図 1)．そのうち A と B は同じライブラリを使用した．メールアドレスとパスワードの入力欄が 1 つずつあり，メールアドレスでない文字やパスワードが 8 文字以下の場合は各入力欄下部にエラーが表示される．各 UI で (1) メールアドレス欄の選択，(2) 一文字入力，(3) パスワード欄の選択，(4) 一文字入力という 4 つの操作を行い，各操作前後の DOM を記録した．

記録した DOM ツリーをそれぞれ操作対象を根としたツリーに変換し，1 操作ごとに操作前後の差分を取得した．差分は各 UI，各操作ごとに存在するため 16 個生成される．次に 2 つの UI 間で操作対象が一致するか確認するため，4 つの UI のうち 2 つを選ぶ 6 通りの UI ペア (X,Y) を作成した．操作は各 UI に 4 つずつあるため，1 つの UI ペアに 16 通りの操作ペア (Xn,Yn) が作成できる．ペアである，Xn の変化差分のツリーと Yn の変化差分のツリーの木編集距離 (TED) [2] を算出し，TED の低いペアから順に，類似度の高いペアとして操作対象の対応関係を採用した．

図 2 に，UI ペアごとの，一致した操作対象と最小 TED，操作番号を示す．操作対象の対応関係は全ペアで一致した．しかし，操作は 4 組で一致せず (33%)，文字入力とパスワード欄の選択が同じ操作として認識されている．パスワード入力欄に関しては，メールアドレス欄のペアが決定したため，消去法で一致したと考えられる．

4 おわりに

本研究では，類似機能の対応関係を判定するために，DOM の変化差分同士を TED で比較する方法を提案した．4 種類の UI で総当たり的に評価した結果，操作対象は全ペアで一致し，対応関係の判定は達成された．しかし操作が約 33%の割合で一致せず，精度には難が残る．今後の課題として，操作の一致精度を高めるための変化差分のフォーマット最適化や，スタイルデータの参照，ログインフォーム以外の種類の UI での検証などが挙げられる．

謝辞 本研究は，JSPS 科研費 JP20H05706，JP22K21279 の助成を受けたものです．

参考文献

[1] H. Kirinuki et al. Recommending correct locator for broken test scripts using various clues in web application. コンピュータ ソフトウェア, Vol. 36, No. 4, pp. 3–17, Nov 2019.
[2] P. Bille. A survey on tree edit distance and related problems. In *Theoretical Computer Science*, Vol. 337 of *1-3*, pp. 217–239, June 2005.

Grad-CAMを用いた画像認識AIの特徴分析の試み

An Attempt to Analyze Features of Image Recognition AI Using Grad-CAM

西村 滋幸 * 本田 澄 † 山下 育男 ‡

あらまし 機械学習モデルの選択では，一般に使いやすさや計算速度などがポイントとなるが，モデルの注目箇所の特徴を考慮して選択することも重要となる．本論文では，画像認識 AI を対象とし，Grad-CAM を利用して学習モデルである VGG16 と ResNet18 について注目箇所の特徴を分析し，その差異を明らかにした．

1 はじめに

機械学習モデルの選択において，使いやすさや計算速度などを考慮して選択することが多いが，学習モデルの注目箇所を考慮した選択方法はあまり見られない．現在のソフトウェア開発では画像認識 AI を組み込む事例が増えつつあり，ソフトウェアの品質向上のため，その特徴把握が重要となっている．本論文では傷がある金属組織画像に対し，画像認識 AI の学習モデルである VGG16 と ResNet18 について，注目箇所を可視化することでモデルの特徴を調べる．

2 評価方法

本論文では Simonyan らが提案した 16 層ある CNN モデルである VGG16 と，He らが提案した 18 層ある CNN モデルである ResNet18 [1] を対象として，傷のある金属組織画像 150 枚と傷の無い金属組織画像 1000 枚のデータセットを持つ DAGM2007 [2] を訓練データとして利用する．DAGM2007 には傷の箇所の中心とその傷を囲む楕円の情報が含まれている．本論文では Grad-CAM を用いて注目個所の中心を特定し，傷の中心を比較し評価する．Grad-CAM は CNN モデルの予測結果に寄与する重要な箇所をヒートマップ化する手法である [3]．

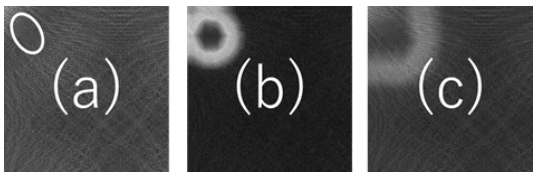


図 1 傷のある金属組織画像と機械学習モデルの注目箇所

図 1 に傷のある金属組織画像と Grad-CAM を用いた学習モデルが注目した箇所のヒートマップを示す．図 1 の (a) は傷のある金属組織画像を示し，白い楕円に傷があることを示す．(b) は VGG16 の注目した箇所をヒートマップ化した画像であり，(c) は ResNet18 の注目箇所をヒートマップ化した画像である．両モデルの注目箇所が傷を中心としていることがわかる．楕円の中心座標とヒートマップの最大となる座標（最大注目座標）までの距離が，楕円の短軸より小さい場合に傷を正しく注目していると評価した．さらに，ヒートマップの重みが 0.5 以上となる領域と傷

*Shigeyuki Nishimura, 大阪工業大学
†Kiyoshi Honda, 大阪工業大学
‡Ikuo Yamashita, 関西電力株式会社

との重なりの割合を用いて，どちらのモデルが正しく傷を注目しているのか分析を行った．

3　評価結果

2 つのモデルの精度を 0.95 付近に調整し，傷の中心座標と最大注目座標との距離と，注目領域のピクセル数を収集し平均値を評価した．平均距離と注目領域から，正しく傷を注目しているのか比較ができる．表 1 に VGG16 と ResNet18 の精度を同程度にするために調整したパラメータを示す．また傷の中心座標と最大注目座標との距離と，注目領域のピクセル数の平均値を表 2 に示す．

表 1　VGG16 と ResNet18 の訓練に利用したパラメータ

	傷あり	傷なし	エポック数
VGG16	150	1000	75
ResNet18	105	650	10

表 2　傷の座標から中心箇所の距離と注目個所のピクセル数

学習モデル	VGG16	ResNet18
楕円の中心と最大注目座標の平均距離	14.05	30.80
注目領域の平均ピクセル数	0.93	14.50

4　考察

表 2 から ResNet18 と比べて VGG16 の方が傷の中心と最大注目座標の平均距離が小さく，傷の中心を注目していることがわかった．また，注目領域についても ResNet18 と比べて VGG16 の方が狭いことがわかった．このことから、それぞれの学習モデルによって注目している箇所が異なることがわかった．

5　関連研究

機械学習モデルの解釈や判断根拠の説明を出力するモデル，またそれに関する技術や研究分野については XAI(Explainable AI) と呼ばれ，盛んに研究されている．本研究は XAI 技術を利用し，画像認識 AI のモデルの特徴を分析した．

Canziani らは，ディープニューラルネットワークモデルについて精度，演算回数，推論時間，消費電力などの観点で比較し分析している [4]．本論文では，学習モデルの注目箇所を可視化し，その特徴を比較した．

6　まとめ

傷のある金属組織画像に対して傷のありなしを評価するモデルを VGG16 と ResNet18 で構築し，注目箇所からモデルの特徴を分析した．VGG16 は傷の中心に近く狭い範囲で傷のある画像を識別していることがわかった．ResNet18 は傷の中心とは外れた場所に注目箇所の中心があり，注目領域は広い範囲で傷のある画像を識別していることがわかった．

謝辞　本研究は JSPS 科研費 JP19K20242 の助成を受けたものです．

参考文献

[1] He, Kaiming, et al. “Deep residual learning for image recognition.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
[2] DAGM2007, https://conferences.mpi-inf.mpg.de/dagm/2007/prizes.html
[3] Selvaraju, Ramprasaath R., et al. “Grad-cam: Visual explanations from deep networks via gradient-based localization.” Proceedings of the IEEE international Conference on Computer Vision. 2017.
[4] Canziani, Alfredo, et al. “An analysis of deep neural network models for practical applications.” arXiv preprint arXiv:1605.07678, 2016.

システムテストにおけるキーワード駆動テストの適用とキーワードの階層化設計

Application of Keyword-Driven Testing to System Tests with Layered Keywords

櫻井 壮希 * 塚本 夏基 † 内木 大地 ‡

あらまし キーワード駆動テストにおけるテストケース作成効率化の為，テスト関連の各ロールの関心事に合わせたテストケース記述を可能とする，キーワードの階層化設計方式を考案した．実製品にて同方式を適用した結果，抽象度の高いキーワードを使用した記述により，テストケース作成効率の向上が見られた．

1 はじめに

派生開発の効率化の為には，上流工程だけでなく下流工程 (テスト) の資産も再利用性を高めることが重要である．本研究では半導体製造装置を対象とし，システムテスト工程の効率化とテスト数増大を通じた品質向上をめざし，キーワード駆動テスト [1] の導入とキーワードの階層化設計によるテスト資産の再利用性向上を図った．

2 システムテストへのキーワード駆動テスト導入

2.1 シミュレーションテスト環境

本研究の対象としたシステムテスト環境の全体概要を図 1 に示す．キーワード駆動テスト (KDT) の基盤としては Robot Framework [2] を使用し，テストケース (TC) に記述されたキーワードを元に装置シミュレータに対するコマンド送受信を行い，テスト手順の実行と結果の判定を行う．

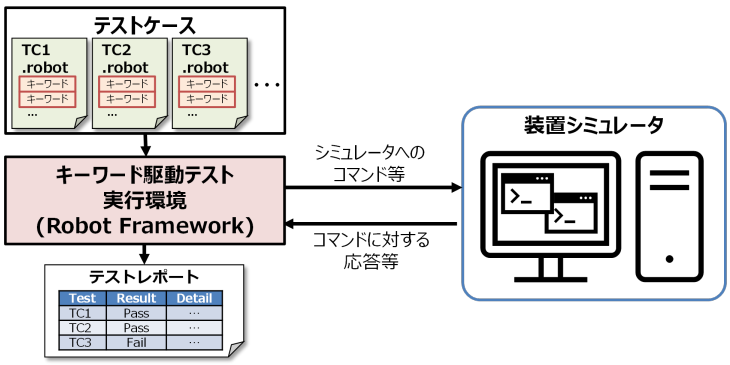


図 1 キーワード駆動によるシステムテスト環境

2.2 考案したキーワード階層化設計

前節で述べたテスト環境では，KDT の導入によりテスト手順をキーワードとして資産化することで再利用性を向上し，TC 作成効率の向上を図っている．本研究においては，TC 作成の更なる効率向上を目的とし，テスト関連の各ロールの関心事に合わせたキーワードの階層化設計を行い，再利用するテスト手順の粒度を上げられる方式を考案した．

*Soki Sakurai, 株式会社 日立製作所 サービスシステムイノベーションセンタ
†Natsuki Tsukamoto, 株式会社 日立ハイテク ナノテクノロジーソリューション事業統括本部
‡Daichi Naiki, 株式会社 日立ハイテク ナノテクノロジーソリューション事業統括本部

図 2 左に示すとおり，テスト関連のロールとしては「テストシステム開発者」「装置ソフト設計者」「QA」の 3 種類を定義した．それぞれのロールではテストにおける関心事が異なっており，テストケース作成時にはそれら関心事に応じて記述可能となっていることが望ましい．

そこで，三つのロールにおける関心事に対応するよう，図 2 右に示すようにキーワードの階層化を行い，Level1 Level3 までの 3 段階のキーワードを定義した．Level1 キーワードは，最もプリミティブなテスト手順である，シミュレータへのコマンドを表すものである．Level2 キーワードはより抽象度の高い，装置への一連の操作を表現するもので，Level1 キーワードの組合せで実現される．Level3 キーワードは最も抽象度が高く，QA のテストに合わせて顧客の運用条件を指定するキーワードとした．Level3 キーワードは Level1 2 キーワードの組合せで実現する．

テスト関連のロール	担当するアクション
テストシステム開発者	テスト環境やキーワードの作成、動作確認、保守。
装置ソフト設計者	製品の仕様に沿ったテストを作成・実施
QA	顧客のユースケースに沿ったテストを作成・実施

キーワードレベル	記述の抽象度	対応する表現の粒度	対応するロール
Level1	低	シミュレータや、シミュレータ上の各装置へのコマンド送信、状態取得等の記述	テストシステム開発者
Level2	中	装置への一連の操作の記述 (Level1キーワードの組合せで実現)	装置ソフト設計者
Level3	高	顧客の運用条件の記述 (Level1~2キーワードの組合せで実現)	QA

図 2　テスト関連ロールに応じたキーワードの階層化設計

3　評価・考察

前節で述べたキーワード階層化設計を段階的に導入した際の，累計 TC 作成数と TC 作成効率の推移を示したのが図 3 である．棒グラフ (左軸) は累計 TC 数を示し，開始時を 0，10 か月目時点を 100 として規格化している．折れ線は各月における人数あたりの TC 作成効率 (TC/月/人) を示す．導入開始後から 3 か月目までは Level2 キーワードを使用して TC 作成を行い，4 か月目以降は Level3 キーワードも含めた TC 作成を行った．これら二つの期間における平均の TC 作成効率を比較すると，高レベルキーワード (Level3) を使用した後者の期間では，前者の期間に比べて約 2.6 倍の作成効率となっていることが判明した．各期間における開発案件の状況等について精査は必要であるが，キーワード階層化によるテスト記述の「高級言語化」が効率向上に寄与した可能性が有ると考えられる．

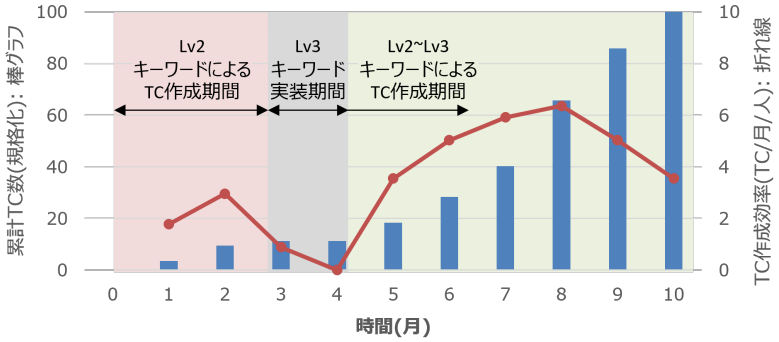


図 3　KDT により作成した累計 TC 数 (規格化) と作成効率の推移

参考文献

[1] 29119-5-2016 - ISO/IEC/IEEE International Standard - Software and systems engineering - Software testing - Part 5: Keyword-Driven Testing.
[2] Robot Framework: https://robotframework.org/.

ソフトウェア開発者の信頼度の評価に向けて

Toward Assessing the Trustworthiness of Software Developers

池田 海斗[*]　西浦 生成[†]　笹倉 万里子[‡]　門田 暁人[§]

あらまし　多様な開発者が集まる GitHub 上のソフトウェア開発では，プロジェクトの成否・盛衰は，開発者の特性（プログラミング能力，貢献タイプなど）に依存する．本研究では，開発者の特性のうち「信頼度」の定量化を試みる．信頼できる開発者とは，例えば，割り当てられた作業を迅速に遂行している，メンションに対し迅速に返信している，自身のコメントに対し他者からのリアクションが付きやすい，といった特徴を持つ．本研究では，Meta Platforms 社（旧 Facebook）が開発に関わる React プロジェクトを対象とした検討を行う．

1　はじめに

今日のソフトウェア開発では，GitHub を主たる活動拠点とする Open Source Software (OSS)開発が広まっており，GitHub 上に 3000 万人を超える開発者が参加している．このような開発形態において，プロジェクトの成否・盛衰は，プロジェクトに参加する開発者の人数，および，開発者の特性（プログラミング能力，貢献タイプなど）に依存する．

従来，開発者の特性を定量化する様々な取り組みが行われてきた．例えば，Cheng ら[1]は，GitHub 上の 29 件のプロジェクトの開発者を分析し，因子分析によって 6 つの貢献タイプを同定している．池本ら[2]は，GitHub 上の約 10 万人の開発者を原型分析により分析し，7 つの貢献タイプを同定している．また，Emmanuel ら[3]は，開発者の利用するライブラリの種類に基づいて，開発者のスキルを同定している．

本研究では，貢献タイプやスキルといった観点ではなく，開発者の「信頼度」に着目した定量化を試みる．ここでいう信頼度とは，仕事の遂行に対する他の開発者からの信頼の度合いのことであり，例えば，割り当てられた作業を迅速に遂行している，メンションに対し迅速に返信している，自身のコメントに対し他者からのリアクションが付きやすい，といった開発者は信頼度が高いと考える．

定量化の方法の検討や試行のための開発プロジェクトとして，本研究では JavaScript の UI 構築のためのライブラリ React を用いる．React を用いる理由は，JavaScript の UI 構築のためのライブラリ／フレームワークの中で最も人気が高く，多くのユーザに用いられていること，および，Meta Platforms 社（旧 Facebook）が開発に携わっており，会社の業務として開発作業を行う信頼度の高い開発者が含まれていると考えられることである．

2　現時点の調査結果

本稿では，「投稿した issue に対しコメントを得られる率（コメント獲得率）が高い者

[*] Kaito Ikeda, 岡山大学工学部情報工学科
[†] Kinari Nishiura, 岡山大学学術研究院自然科学学域
[‡] Mariko Sasakura, 岡山大学学術研究院自然科学学域
[§] Akito Monden, 岡山大学学術研究院自然科学学域

表 1　コメント獲得率に基づく信頼性ランク

ランク	コメント獲得率	Issue 投稿数	平均獲得コメント数
1	100%	31	9.94
2	99.14%	116	3.64
3	99.10%	223	2.25
4	97.44%	39	5.08
5	97.06%	34	5.50
40	67.18%	131	3.03
41	62.07%	58	1.64
42	55%	40	0.95
43	54%	50	1.48
44	2.44%	41	0.02

ほど信頼度が高い」とみなして分析する．Issue を報告したにも関わらず誰からもコメントを得られない者は，その人の報告は信頼するに値しないとみなされていると考えられるためである．React プロジェクトに issue を投稿したことのある者は 9922 名であり，そのうち 30 件以上の issue を投稿した者は 44 名であった．このうち，コメント獲得率の平均値は 85%であった．表 1 に，コメント獲得率の上位 5 名と下位 5 名を示す．上位 5 名はコメント獲得率が 97%を超えており，ほぼすべての issue 投稿に対し，コメントを獲得している．一方，最下位のランク 44 の者はコメント獲得率が 2.44%であり，41 件の issue 投稿に対し，コメントが得られたのはわずか 1 件であった．また，コメント獲得率と issue 投稿数の間に関連はみられなかった（相関係数は-0.038 であった）．一方，コメント獲得率と平均獲得コメント数の相関係数は 0.52 であり，ある程度の相関がみられた．

コメントをほとんど獲得できていないランク 44 の者の issue を確認したところ，具体的な説明のない同文の Issue を 41 件も連投していた．また，ランク 42 の者の Issue は軽微なタイポ修正等のコミットが主であり，説明文等も空欄であった．また，ランク 43 の者は，説明文中にテストや再現性などを明記しており，ディスカッションが行われることなくマージされる割合が多かった．しかし，プルリクエストの Approve 時にコメントが行われている場合が多くあり，今回それらのデータを収集できていなかったため，Approve 時のコメントを加算した上で再調査を行なってみたい．

3　おわりに

本稿では，「コメント獲得率が高い者ほど信頼度が高い」とみなし，信頼度の定量化を試みた．React プロジェクトを分析した結果，コメント獲得率には大きなばらつきがあることが分かった．今後，信頼度の他の要因について分析を進めていく予定である．

4　参考文献

[1] J. Cheng, J. L. C. Guo, "Activity-based analysis of open source software contributors: roles and dynamics," Proc. 2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), Aug. 2019.

[2] 池本, 門田, "原型分析による OSS 開発者の貢献タイプの分析," コンピュータソフトウェア, Vol. 37, No. 4, pp.17-23, Nov. 2020.

[3] W. C. Emmanuel, A. Monden, "Human Resource Analysis Based on Used Libraries in Eclipse Projects on GitHub," Proc. 22nd IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, No. WP1-07, pp.1-4, Nov. 2021.

Web GUIに対するオンラインジャッジシステムプロトタイプの実装

岡嶋 隆人 * 田中 昂文 † 櫨山 淳雄 ‡ 橋浦 弘明 §

あらまし 現代の実用的なソフトウェア開発においてGUIの実装に関する知識は必須であるが，既存のプログラム評価サービスはGUIをもつアプリケーションに対応していないことが多い．本稿ではVisual Testing技術を用いることによって，GUIを持つアプリケーションに対応したオンラインジャッジシステムのプロトタイプの実装を行った結果について述べる．

1 はじめに

今日のスマートフォンやWebで利用されているアプリケーションにおいて，GUI（Graphical User Interface）は必要不可欠な要素である．Pengnateら [1] は，ソフトウェアの商業的な成功において，GUIのデザインが重要な要素であるとを指摘している．そのため，実用的なソフトウェアを開発するためは，GUIを実装する知識が必要不可欠である．こうした知識を習得するためには，学習者が講義や授業だけではなく，課外時間に自主的な学習を行う必要がある．

2 研究目的

プログラミングの知識習得のための自主学習を行うことができるサービスとしてpaiza[1]などが挙げられる．このようなサービスは，学習者が提出した課題 (プログラム) を自動的に採点し，その結果をフィードバックする機能を持っており，学習者は時間や場所を選ばずに自分のペースでプログラミングの自主学習ができるようになっている．しかしながら，こうしたサービスではGUIを持つプログラムに対応していないことが多い．このため，橋浦ら [2] は，Visual Testing技術を用いた，GUIに対応したオンラインジャッジシステム (OJS) を提案している．本稿では，GUIに対応したOJSについて，その実現可能性を確認するためのプロトタイプの実装を行う．

3 GUIに対応したOJSの実現

OJSのプロトタイプを実装するにあたり，課題出題画面，課題提出画面，採点結果詳細確認画面の3つについて実装を行った．また，採点対象のGUIはWebページであるものとし，外部CSS(.css) やJavaScript(.js) などを含まないHTMLファイル単体で構成されるものとした．

本プロトタイプはOJSの性質上,Webアプリケーションとして実装する必要がある．このため，Webアプリケーションフレームワークとして Django[2]を，Visual Testingエンジンとして SpiderTailed [3] を実装に用いた．実装したプロトタイプの画面を図1に示す．

課題出題画面 (図 1-①) には課題の一覧が表示されており，学習者が解答したい

*Takato Okajima, 日本工業大学大学院

†Takafumi Tanaka, 玉川大学

‡Atsuo Hazeyama, 東京学芸大学

§Hiroaki Hashiura, 日本工業大学

[1] https://paiza.jp/

[2] https://djangoproject.com

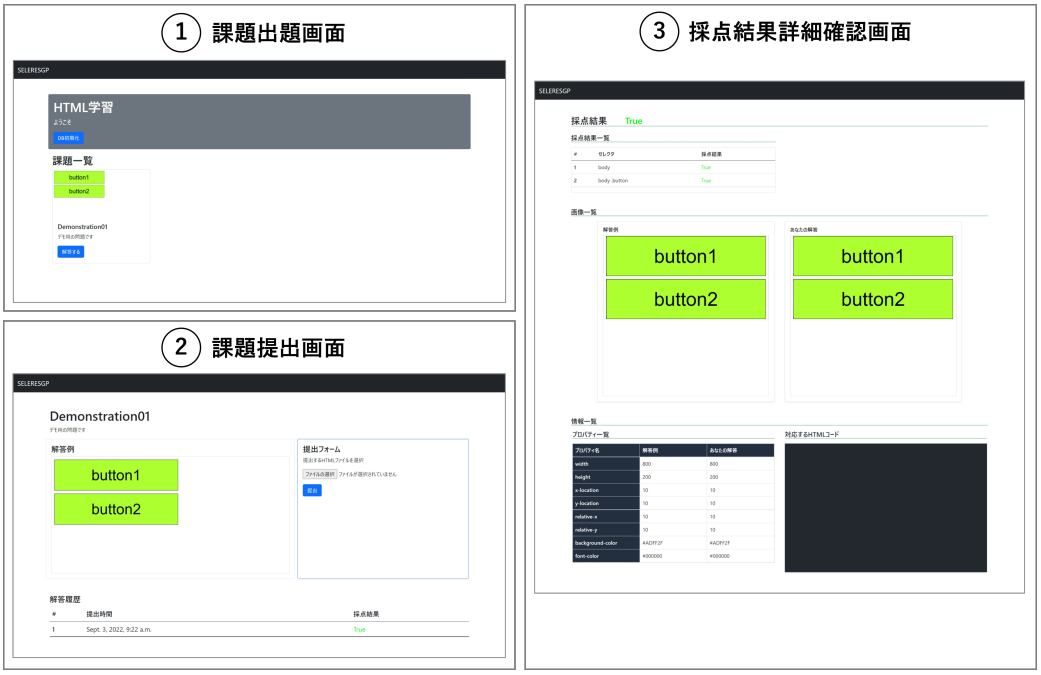


図 1　実装した OJS プロトタイプの例

課題を選択すると，課題提出画面 (図 1-②) に遷移する．課題提出画面では，課題が満たすべき HTML の出力結果が表示されており，学習者は出力結果に基づいて HTML を作成し，画面内のフォームから提出を行うことで課題の採点を受けることができる．

課題の採点結果は課題提出画面の下部に表示される．学習者は採点結果を選択することで Web ページ内の各要素の画像や状態といった，採点結果の詳細な情報を確認することができる (図 1-③)．

4　おわりに

本稿では，Visual Testing 技術を用いた Web ページに対する OJS のプロトタイプを実装し，その実現可能性を確認した．

今後は本システムの完成を目指すために，ユーザ認証機能の追加や詳細な採点機能の実装を行うとともに，実際の学生の使用による評価を通して，さらなる実証可能性を確認していきたい．

謝辞　本研究の一部は JSPS 科研費 21K12179 の助成を受けた．

参考文献

[1] Supavich (Fone) Pengnate and Rathindra Sarathy. An experimental investigation of the influence of website emotional design features on trust in unfamiliar online vendors. *Computers in Human Behavior*, Vol. 67, pp. 49–60, 2017.

[2] 橋浦弘明, 岡嶋隆人, 田中昂文, 櫨山淳雄. Web アプリケーションに対するオンラインジャッジシステムの提案. 情報処理学会第 84 回全国大会講演論文集, Vol. 4, pp. 397–398, Mar. 2022.

[3] Takato Okajima, Takafumi Tanaka, Atsuo Hazeyama, and Hiroaki Hashiura. SpiderTailed: A tool for detecting presentation failures using screenshots and DOM extraction. In *Proceedings of the 14th International Joint Conference on Knowledge-Based Software Engineering (JCKBSE 2022)*, 2023. (In printing).

技術的負債に関する課題票の分類モデルに単語分散表現が与える影響の分析

Analyzing the Impact of Word Embeddings on Classification Models for Technical Debt-Related Issues

田口 舞奈 * 木村 祐太 † 大平 雅雄 ‡

あらまし 本研究の最終的な目標は，技術的負債に関する課題票を自動分類するための深層学習モデルを構築することである．本研究では，分類モデルに用いる単語分散表現が分類性能にどのような影響を与えるのかを調査する．

1 はじめに

技術的負債（Technical Debt: TD）は，時間的制約などの理由により最適でない手法を選択することで導入される不完全な設計や実装を指す [1]．TD が蓄積すると保守コストの増加につながるため，可能な限り早く TD を発見し返済することが重要である．そのため，TD に関する研究がこれまで盛んに行われている．

特に近年では，開発者自らが TD の存在を認識し明文化した SATD（Self-Admitted Technical Debt）[2] [3] に注目が集まっている．SATD は大きく分けて，(1) 個々の開発者がソースコードコメント中に TD の存在を指摘するケース（SATD-Comment）[2] と，(2) プロジェクトの課題管理システムに TD の存在を報告するケース（SATD-Issue）[3] の 2 種類がある．これらは開発者視点からの TD の表明であるため，SATD の多面的な分析を通じて，TD の実用的な優先順位付け手法や自動修正手法の構築のための有益な知見が得られると期待されている．

前者の SATD-Comment に関しては，特定のキーワードを用いて比較的容易に抽出可能であるためこれまで数多くの研究事例が存在する．一方，後者の SATD-Issue 関しては，大規模な OSS プロジェクトのごく少数において運用が開始された段階にあり，十分な研究事例が存在していない．しかしながら，課題票として報告される SATD-Issue は，(1) TD の種類が多岐に渡ること（設計や実装のみならず，要件定義や各種ドキュメント中の TD も指摘される），(2) TD の存在のみならず，混入原因や返済方法についても開発者間で議論されることから，SATD-Comment に比べても分析対象としての価値が高いと考えられる．そこで，SATD-Issue を収集するための分類手法が提案されている [4]．

2 SATD-Issue の自動分類手法

先行研究 [4] の手法では，4 つの観点（報告者，テキスト，プロセス，ソースコード）から SATD-Issue の特徴分析をおこない，分析結果から得られた特徴量を用いて分類モデルを構築している．

しかし， [4] の手法は SATD-Issue を収集することを目的として構築されているため，モデル構築に用いられる特徴量は解決済みの課題票からでしか抽出できないものが多い．そのため，報告された直後の課題票には適用できないという問題がある．また， [4] の手法におけるテキスト特徴量は，BoW や TF-IDF といった極めてシンプルな単語分散表現を利用しているため，分類性能の改善余地が大きいと考えられる．

*Maina Taguchi, 和歌山大学
†Yuta Kimura, 和歌山大学
‡Masao Ohira, 和歌山大学

そこで本研究では，課題票のテキストデータのみを利用し，BoW や TF-IDF 以外の単語分散表現を用いた深層学習モデルを構築する．加えて，SATD-Issue に含まれる TD の種類を考慮した分類モデルの構築も試みる．

3　単語分散表現に基づく課題票の分類

本研究では，先行研究 [4] と同様に，GitHub で管理される SATD-Issue を扱う 4 つの OSS プロジェクト (microsoft/vscode, influxdata/influxdb, saleor/saleor, nextcloud/server) を対象として，

(1) ソースコード，設計，ドキュメント，テスト，要件定義など，[5] で定義されている種類の TD と課題票で報告される TD との対応関係の調査
(2) (1) で特定した TD の種類ごとでのテキスト特徴量の違いの分析
(3) 後述の手法で構築する分類モデルの性能評価

を実施する．

分類モデルの構築に関しては，まず，課題票が報告された時点で抽出可能な「課題票のタイトル」と「本文のテキスト」を 5 種類の単語分散表現（BoW，TF-IDF，FastText，Word2Vec，Glove）を利用して特徴量とする．次に，ELMo または BERT などの深層学習モデルを用いて得られた特徴量を学習させることで技術的負債に関する課題票を分類するモデルを構築する．分類モデルの評価では，Precision，Recall，F1 値を評価指標として用いる．

4　まとめと今後の課題

本研究では，課題票が投稿された時点で取得可能なテキストデータのみを用いて，5 種類の単語分散表現と 2 種類の深層学習モデルを組み合わによって SATD-Issue 分類モデルを複数構築し分類性能を比較する．また，TD の種類を定義し，それぞれのテキスト特徴量の違いを分析することで，SATD-Issue を TD の種類ごとに自動分類するモデルを構築する．

今後の課題として，まず，データセットの作成が挙げられる．本研究が対象とする前述の 4 つの OSS プロジェクトでは，課題票に対して “TD” や “Debt” などのラベルを付与することで技術的負債に関連している課題票を管理している．しかしながら，TD の種類をラベルやその他の方法で明示していないため，SATD-Issue を TD の種類ごとに自動分類するモデルを構築するためには，課題票を目視で確認し TD の種類を特定しておく必要がある．また，更なる展望として，課題票に紐づけられたソースコードや課題票の報告者のデータなど，報告時点の課題票から入手可能なテキスト以外のデータについて，特徴量として追加できるか検討する必要がある．

謝辞　本研究の一部は，文部科学省科学研究補助金（基盤（C）：22K11974）による助成を受けた．

参考文献

[1] Ward Cunningham, The WyCash Portfolio Management System, ACM SIGPLAN OOPS Messenger, Vol.4, No.2, pp.29–30, 1992.
[2] Aniket Potdar and Emad Shihab, An Exploratory Study on Self-Admitted Technical Debt. In Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME ’14) , pp. 91-100, 2014.
[3] Laerte Xavier, Fabio Ferreira, Rodrigo Brito, and Marco Tulio Valente, Beyond the Code: Mining Self-Admitted Technical Debt in Issue Tracker Systems. In Proceedings of the 17th International Conference on Mining Software Repositories (MSR ’20), pp.137–146, 2020.
[4] 木村祐太, 大平雅雄, 技術的負債に関連する課題票分類手法の構築, 情報処理学会論文誌, Vol. 64, No. 1, 2023.（投稿中）
[5] Nicolli S. R. Alves, Leilane F. Ribeiro, Vivyane Caires, Thiago S. Mendes, and Rodrigo O Spínola. Towards an Ontology of Terms on Technical Debt. In Proceedings of 2014 Sixth International Workshop on Managing Technical Debt (MTD ’14), pp.1–7, 2014.

再利用性向上を目的とした数値解析ライブラリ構築パターンの提案

The patterns for numerical simulation libraries to improve reusability

市村 純一[*] **中谷 多哉子**[†]

あらまし 複雑な問題に対して様々な数値解析による検証を行う場合，解析モデルの再利用性を向上させることは重要である．筆者らは，様々な現象に対して数値解析モデルの再利用性を向上をさせることを目的として「対象 (Object)」と「作用 (Action)」へ現象を分解するパターンを提案している [1]．本発表では提案のパターン中の Through 変数の定義方法の任意性について，計算の安定性から最も階数の高い微分値を選択すること，また結合度の観点から Object へ定義すべき属性について明らかにした．

1 背景・目的

近年，社会が直面する問題はより複雑化し，様々な側面から統合的に問題を検討することの重要性が増してきている．また，コンピュータ技術が発達し，様々な現象を対象に数値解析を用いた検討が進んできている．このため，効率的に数値解析用のモデルを作成するためには，現象を再利用可能な形に分解し・それらを再構成する形のモデリングアプローチが重要となる．他方，物理分野においては再利用性を高めるために，保存則を活用した物理モデリングというコンポーネントベースのモデリング手法が利用されている [2]．筆者らは保存則が明確ではない系へ拡張し，現象を分解し，コンポーネントベースのライブラリ構築するパターンを提案している．本研究では提案パターンに対して，Through 変数の選択方法について解析の安定性・再利用性の観点から明らかにする．

2 Object-Action パターン

筆者らは，現象を「対象」と「作用」に分離する Object-Action パターンを提案している．本パターンは，現象の変化を微分方程式で表現する連立微分代数方程式 (DAE) を対象とし，以下の 4 つの項目で構成されている (図 1).

STEP.1 Object-Action への分解

「対象 (Object)」には現在の状態を識別するために必要な変数及び，その変数間に存在する普遍な関係を定義する.「作用 (Action)」には Object が持つ状態がどのように変化するのか，すなわち Object が持つ属性による時間変化の関係を定義する.

STEP.2 Through/Across 変数の定義

Object から Action へは現在の Object がもつ状態量を入力する (Across 変数). Action から Object へは状態の変化量を入力する (Through 変数)．これらの関係は Modelica 言語ではオブジェクト同士の接続に対して定義されている.

STEP.3 Action の分解/統合

Action の各項を一つづつの Action として分解後共通の項を統合する．これにより，各 Action の組み合わせによりさまざまな現象のモデルが再現できるようになる.

STEP.4 Action の共通部分の抽出

分解した Action に共通の関係を抽出する (例：作用・反作用の法則).

[*]Junichi Ichimura, 放送大学/ニュートンワークス株式会社

[†]Taeko Nakatani, 放送大学

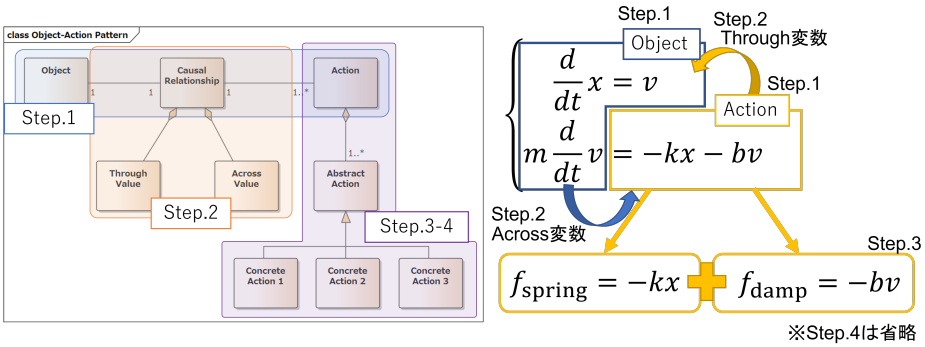


図 1　Object-Action パターン

3　Through 変数に関する定義の任意性についての検討

STEP.2 で示した Through 変数は数式変化により 2 つの任意性が含まれる．これらの選択は解析の安定性，結合度の観点から決定可能である．

微分演算に対する任意性　数式の両辺を微分/積分することにより，同一の現象を別の形式により表現が可能となる．一方，Object と Action を完全に分離して計算する場合，Through 変数は不連続値としてやり取りするため，Through 変数の微分値は計算できない．
このため，Through 変数は Object で微分演算が必要をなくすため最も階数の高い Across 変数の微分値を選択すべきである．なお、Object と Action が分離し計算するケースである複数のツールで計算を分ける場合のツール間での入出力変数の選び方として類似の議論行われている [3].

係数の任意性　数式の両辺を定数倍することにより，同一の現象を別の形式により表現が可能となる．この係数の考え方によりパターンの再利用性が変わる．例えばモノの動きのライブラリを作成する場合，速度の時間変化である加速度を Through 変数に定義した場合は Through 変数の計算に必要な質量は Action に定義する必要がある．この場合前述の STEP.3 の操作で Action を分解した場合，各 Action に対して定義する必要が生じ結合度が高くなる．一方，速度の時間変化に比例した値である力 (=加速度×質量) を Through 変数に定義した場合は，力の計算に質量は必要なくなるため，質量は Object にのみ定義され結合度が低くなる．
このように，結合度を下げ再利用性を高めるためには Step.2 で Action 全体にかかる係数は Object に定義する必要がある．

4　まとめ

本研究では，数値解析の再利用性を向上させるための提案パターンに対して，Through 変数の定義方法を計算の安定性・再利用性の観点から明らかにした．

参考文献

[1] J. Ichimura and T. Nakatani,“Pattern to improve the reusable numerical simulation”14th International Joint Conference on Knowledge-Based Software Engineering, Cyprus, Aug. 2022
[2] Modelica Homepage, https://modelica.org/index.html. Last accessed 11 Apr. 2022
[3] 経済産業省，　”自動車開発におけるプラントモデル I/F ガイドライン ver1.0”, https://warp.da.ndl.go.jp/info:ndljp/pid/10341576/www.meti.go.jp/press/2016/03/20170331010/20170331010.html，31 Mar. 2021

SCDV モデルを利用する技術用語に対応した自然言語文書検索の提案

Retrieval of Natural Language Documents Containing Technical Terms Using SCDV Model

辻　優太郎 * 神谷 年洋 †

あらまし SCDV(Sparse Composite Document Vectors) は文書 (自然言語のテキスト) から文書ベクトルを算出する軽量な手法であり，文書のコーパスを学習データとして必要とする．コーパスのサイズが小さいと高品質な学習モデルを作成できない．本研究では一般的な文書の大規模コーパスから作成した汎用的学習モデルと併用することで，特定の分野に特化した小規模なコーパスから作成した学習モデルの精度を向上させる手法を提案する．

Summary. SCDV (Sparse Composite Document Vectors) is a lightweight method to calculate document vectors from documents (natural language text) and requires a corpus of documents as training data. If the size of the corpus is small, it is not possible to create a high-quality model. In this study, we propose a method to improve the accuracy of models created from a small corpus of a particular domain, by using them conjunction with a general-purpose model created from a large corpus.

1　はじめに

自然言語処理の分野では，単語や文書からベクトル表現を得る手法について多くの研究が行われており，Word2Vec [1], Doc2Vec [2], SCDV [3], Bart [4] などの手法が提案されている．本研究では，SCDV(Sparse Composite Document Vectors) を利用した自然言語文書検索において，分野を限定すると学習データのとなるコーパスの量が減少して学習モデルの品質が下がる問題を解決する手法を提案する．具体的には，コンピュータ技術用語の分野の文書から精度よく文書検索を行う手法を提案する．

2　SCDV

SCDV は文書データから文書ベクトルを算出する手法の 1 つである．文書に含まれる単語毎の Word2Vec のベクトル，単語の IDF 値による重み付けや単語のクラスタリングを援用することが特徴である． Doc2Vec をはじめとした他の文書ベクトルのマルチラベル分類と比較して次の利点をもつ [3].

- 既存手法よりも文書の分類精度が優れているケースが多い.
- 計算にかかる時間やメモリといったリソースが既存手法よりも少なくて済む.

SCDV はこのような特徴により，比較的リソースが乏しい個人の PC などでも実用的な検索を実現できる手法であるといえる．

3　目的

特定の分野に特化した検索であれば，その分野の文書をコーパスとして利用したいが，特定の分野の文書は一般の文書と比べてコーパスの量が少なくなるという問

*Yutaro Tsuji, 島根大学総合理工学部

†Toshihiro Kamiya, 島根大学総合理工学部

題がある．

本研究では，小規模なコーパスを用いて学習モデルを作成し，その学習モデルによる検索を補強するために，一般的な文書の大規模コーパスから作成した汎用的学習モデルも併用する手法を提案する．これにより，特定分野に特化した小規模なコーパスから作成した学習モデルの精度を向上させることを目指す．

4　アプローチ

提案する手法の概略を図 1 に示す．提案手法では，技術用語を含む文書を文書ベクトルに変換するために 2 つの学習モデルを利用する．学習モデルの 1 つは，一般的な文書のコーパスから生成したもの（図中「一般文書モデル」）であり，もう 1 つは，技術用語に関する文書のコーパスから生成したもの（図中「技術用語学習モデル」）である．与えられた文書から，2 つの学習モデルによる文書ベクトルを算出し (一般文書学習モデルによるベクトルを Vall，技術用語学習モデルによる文書ベクトルを Vcom とする)，それらを結合したベクトルを最終的な文書ベクトル（V2m）を用いて文書検索を行う．

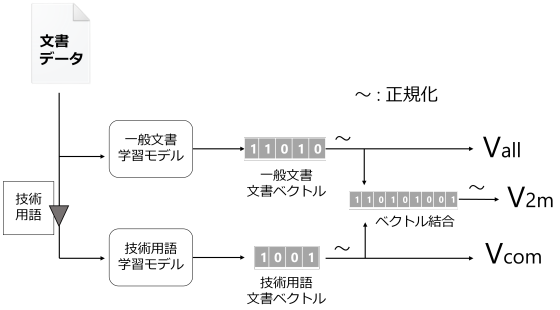


図 1　提案する文書検索手法における文書ベクトルの生成

5　おわりに

本研究では SCDV を用いて，技術用語に対応した文書検索技術するための手法を提案した．今後は評価実験により提案手法のモデルの精度を評価する．オープンソースの一般文書から作成した学習モデルと併用により学習データの量の問題を軽減することで，特定の分野の文書を対象とした，その分野の用語の意味に即した文書検索を実現することを目指す．

謝辞　本研究は JSPS 科研費 22K11975 の助成を受けたものである．

参考文献

[1] T. Mikolov, W. Yih, G. Zweig. “Linguistic Regularities in Continuous Space Word Representations,” Proc. 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 746–751, 2013.
[2] Q. Le, T. Mikolov. “Distributed representations of sentences and documents,” Proc. 31st International Conference on Machine Learning (ICML 2014), pp. 1188–1196, 2014.
[3] D. Mekala, V. Gupta, B. Paranjape, H. Karnick. “SCDV : Sparse Composite Document Vectors using soft clustering over distributional representations,” Proc. 2017 Conference on Empirical Methods in Natural Language Processing, pp. 659–669, 2017.
[4] J. Devlin, M. Chang, K. Lee, K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” arXiv:1810.04805 [cs.CL], 2018.

Maintainability Index を用いた保守性改善プロセス適用

Application of Improvement Process for Software Maintainability by Maintainability Index

加賀 洋渡*

あらまし 派生開発では，大規模化・複雑化によりソースコードの保守性が低下する．一方で，保守性の低下は，不具合発見の遅れの原因の一つと考えられている．本研究では，ソースコードの保守性を定量化し，不具合との相関を明らかにした上で，保守性を改善するプロセスを適用した．

Summary. In derivative development, defecting failure is rate due to large-scale. It required quantification maintainability and improvement. In this research, we analyzed correlation between maintainability and failure and studied application of implement process.

1 はじめに

長年にわたるソフトウェアの派生開発では，ソースコード規模が膨大かつ複雑になり保守性が低下する．一方で，保守性の低下は，不具合発見の遅れの原因の一つと考えられている．製品リリース後の不具合発生は，その対応に大きな工数を要するため，保守性向上が求められる．そこで本研究では，保守性向上を目的として，ソースコードの保守性をメトリクスで定量化すると共に，不具合発生率と MI の相関関係を調査する．また，MI 改善プロセスを策定し，これらの手順を迅速に進めるための MI 改善ツールを開発する．

2 保守性と不具合の相関関係の分析

ソースコードの保守性を表すメトリクスとして Maintainability Index(MI)がある．本研究では Microsoft 社が提唱する算出式計算式[1]を採用する．MI は 0 から 100 までの値を取り低いほど保守性の低いことを示す．ある製品の MI を測定し，関数ごとの MI と，不具合発生率の関係を図 1 のように図示した．MI が低い関数群ほど不具合が発生した関数を含む割合が高いことがわかる．MI 改善の閾値として 20 以上を設定した．

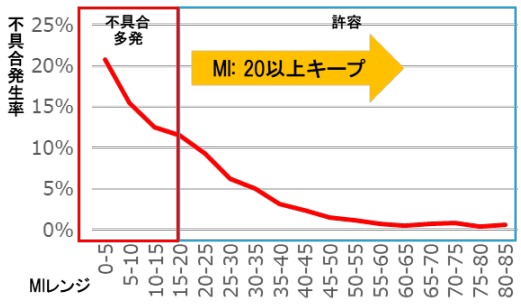


図 1 不具合発生と MI の相関

* Hiroto Kaga, 株式会社 日立製作所 デジタルサービス研究統括本部

3　MI 改善方針

MI は，ステップ数や条件分岐の数，論理演算子の数，単語数や種類によって MI 値を計算する．これらの値が大きいほど，その関数の MI 値は低下する．この内，ステップ数と論理演算子の削減に注目した．リファクタリングにおけるアンチパターンとして広く知られるソースコードの状態を対象とし，これらを改善する方針とした．

- メソッド抽出: MI を劣化させる長大な関数におけるいくつかの処理を関数として抽出することで，元の関数の役割を縮小し，ステップ数を削減する
- 条件文の分解: 条件文において&&(and), ||(or)の論理演算子により複数の条件が記述されている箇所がある．これらの複数の条件式をまとめて関数として抽出することで，元の関数の論理演算子の数を削減する

4　MI 改善プロセスの策定とツール開発

MI 改善プロセスの概要を**図 2** に示す．テスト工程での MI 確認ではコード修正の手戻りが大きいため，実装工程で実装の度に細かい間隔で確認する必要がある．そこで，MI 改善プロセスの適用工程を実装工程と定めた．MI 改善プロセスの流れを以下に示す．

1. ソースコードを実装する
2. MI を測定する
3. MI が 20 以下の場合，ソースコードを見直して MI の改善を図る
4. ソースコードをコミットする

また，これらの手順を迅速に行うために，MI 改善ツールを開発した．MI 改善ツールは MI を測定し，改善箇所をレコメンドし，改善効果を表示する．MI 改善プロセスをある製品のプロジェクトの実装工程に適用した結果，MI が 20 未満である関数が 0 になった．また，既存開発関数の平均 MI が約 37 であるのに対し，新規開発関数の平均 MI は約 53 であり結果平均 MI が 43%改善した．

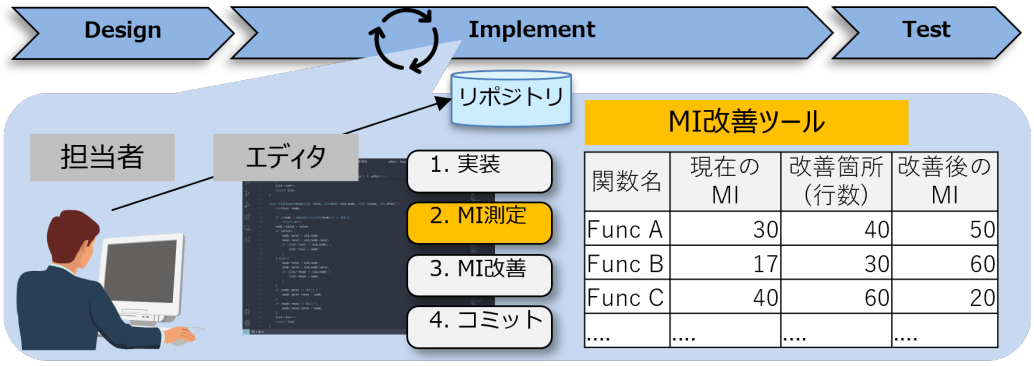


図 2　MI 改善プロセスとツール

参考文献

[1]Microsoft 社，コードメトリックス - 保守容易性インデックスの範囲と意味，https://docs.microsoft.com/ja-jp/visualstudio/code-quality/code-metrics-maintainability-index-range-and-meaning?view=vs-2019

効率的なソフトウェアアップサイクルのための事例知識ベースの予備的評価

Preliminary Evaluation of Case Knowledge Base for Efficient Software Upcycling

中田 匠哉[*] 陳 思楠[†] 佐伯 幸郎[‡] 中村 匡秀[§]

あらまし ソフトウェアアップサイクルは，過去プロジェクトを資産化し活用することでソフトウェア開発を効率化する手法である．本研究では，アップサイクルを効率化する事例知識ベースの予備的評価として，アップサイクル事例をインタビュー形式で収集し，分析・考察を行う．

1 はじめに

1.1 背景

今日，DX やスマートフォンの普及により産業・生活を問わず広い分野でソフトウェアの需要が高まっている [1] [2]．開発者は，限られた資産で高い価値を持つ創造的なソフトウェアを開発することを求められる．この課題を解決する手法に，蓄積された過去の開発プロジェクトから一部分をプロジェクト素材として抽出し，新規の価値あるソフトウェア資産に転換する**ソフトウェアアップサイクル**がある [3]．

1.2 先行研究: アップサイクル事例共有システムの提案

筆者は，先行研究として図 1 に示す**アップサイクル事例共有システム**を提案・検討している [4]．提案システムの目的は，アップサイクルの実績・アイデアをアップサイクル事例として記録・共有することで知識ベースを構築し，未来のアップサイクルを効率的に実行できるようにすることである．

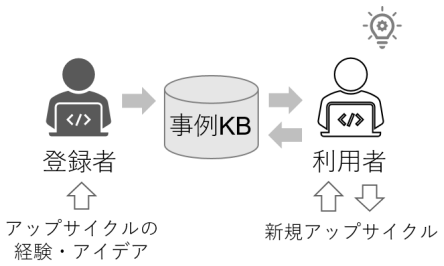


図 1 事例知識ベースの利用方法

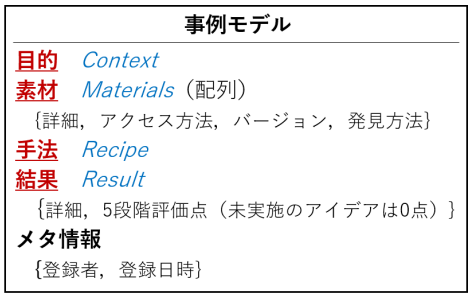


図 2 定義した事例モデル

2 事例モデルの予備的検討

2.1 アップサイクル事例モデル

アップサイクル事例モデルは，図 2 に示す通り，*Context*・*Materials*・*Recipe*・*Result* を主な要素として定義される．それぞれ，アップサイクルの目的・資産化す

[*]Takuya Nakata, 神戸大学
[†]Sinan Chen, 神戸大学
[‡]Sachio Saiki, 高知工科大学
[§]Masahide Nakamura, 神戸大学

る素材・レシピ・結果を表現する文字列である．プロジェクト素材を文字列として表現することで，ソースコード再利用に留まらないデータモデル，アーキテクチャ構造などの知的資産を有効に蓄積・活用することが可能になる．

2.2 実験計画

本実験の目的は，事例共有システムの評価実験を行うための予備的検討として，システムを用いずにアンケート形式で事例を収集し，分析・考察することである．収集される事例は，前節で定義したアップサイクル事例モデルの形式に従う．アンケートの対象者は，同じ研究室に所属する学生 7 人である．アンケートの方法は，対象者がアップサイクルに準じると考える開発事例を，事例モデルに沿った形式で記入してもらった．事例の記入にあたっては，不明な項目は未記入可とした．

3 実験結果

8 件の事例が集まり，その中に合計で 10 個の素材が含まれた．図 3 に事例の概要を示す．素材の種別を重複ありで分類し，図 4 にグラフとしてまとめた．素材のバージョン管理の未実施によってバージョン欄が空欄の素材が 3 件存在した．

WebSocketを用いたタイマーアプリを開発するために、Pub/Subサービスのソースコード・ライブラリ依存を活用した。（結果：4点）
IoT基盤サービスの開発のために、Spring Bootを用いた知識共有サービスの設計アーキテクチャ・ソースコードを活用した。（結果：4点）
スケジューリングサービスをGASで開発するために、ほぼ同じ機能のJavaサービスからアルゴリズムを再利用した。（結果：5点）
Pythonでクラスタリング分析を行うために、他のクラスタリング分析のコードを再利用し、パラメータを微修正した。（結果：5点）

図 3 インタビューで得られた事例のサンプル

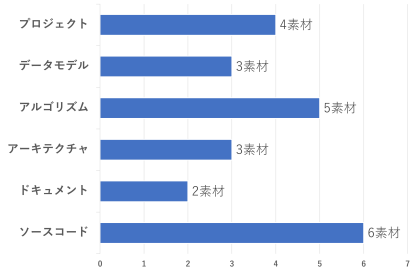


図 4 各素材の種別分類（重複を許す）

4 考察

事例の内容把握のためにモデルにタイトル属性の追加が必要である．また，良い結果の事例が多いため，悪い結果を集めるための工夫が必要である．素材情報が欠損した事例は，情報の質は落ちるが蓄積すべき知的資産であるため，より有効に扱う手法の検討が必要である．ソースコード片だけでなくより広い概念の素材を用いた事例が多くあり，幅広い素材を対象とした事例を収集することができた．

謝辞 本研究の一部は JSPS 科研費 JP19H01138, JP20H05706, JP20H04014, JP20K11059, JP22H03699, JP19K02973, 特別研究員奨励費 22J13217, および, 立石科学技術振興財団の研究助成を受けて行われている．

参考文献

[1] Christof Ebert and Carlos Henrique C Duarte. Digital transformation. *IEEE Softw.*, Vol. 35, No. 4, pp. 16–21, 2018.
[2] Zhiling Guo and MA Dan. A model of competition between perpetual software and software as a service. *Mis Quart.*, Vol. 42, No. 1, pp. 101–120, 2018.
[3] 寺川航平, 陳思楠, 中村匡秀. プロジェクトコーパスを用いたソフトウェア概要 推測手法の実験的評価. 電子情報通信学会技術研究報告, 第 121 巻, pp. 90–96, March 2022.
[4] 中田匠哉, 陳思楠, 中村匡秀. 効率的なソフトウェアアップサイクルのための事 例共有システムの検討. 電子情報通信学会技術研究報告, 第 121 巻, pp. 71–74, March 2022.

提案依頼書におけるセキュリティ要件の実態調査

An Empirical Analysis of Security Requirements in Request for Proposals

村越竜介[*]　西浦 生成[†]　笹倉 万里子[‡]　門田 暁人[§]

あらまし　近年，サイバー攻撃の脅威が増大しており，情報システムに対して多様なセキュリティ要件を盛り込むことが必須となっている．本研究では地方自治体や図書館などが Web 上で公開している提案依頼書を対象とし，セキュリティ要件がどの程度網羅的に記述されているかについて実態調査を行う．調査においては，独立行政法人情報処理推進機構にて策定された非機能要求グレードに基づいて，網羅性を評価する．

1　はじめに

近年，情報システムに対するサイバー攻撃の脅威が増大している．アクセンチュア社の調査[1]では，2019 年におけるサイバー攻撃によるセキュリティ侵害は 1 社あたり平均 145 回であり，これは前年の 130 回から 11%増加している．また，警察庁の報告[2]によると，2021 年におけるサイバー犯罪の検挙件数は 12,209 件であり，これは前年の 9,875 件から 23%以上の増加である．また，独立行政法人情報処理推進機構による 2020 年の調査報告[3]では，中小企業 1,117 社に設置した機器において外部からの不審なアクセスを 181,536 件も検知したことが報告されている．その一方で，全国の中小企業 4,074 社を対象とした 2021 年度の調査報告[4]では，情報セキュリティ対策投資を行っていないと回答した企業は 33.1%にも上る．

サイバー攻撃を防ぐためには，アクセス制御やマルウェア対策を含めた，多種多様なセキュリティ要件を情報システムの設計・実装・運用に取り入れることが重要となる．ただし，今日運用されている情報システムにどの程度のセキュリティ要件が盛り込まれており，どういった要件が手薄になっているかといった調査は，従来行われていない．

そこで本研究では，地方自治体や図書館などの Web 上で提案依頼書が公開されている情報システムの構築案件を対象とし，セキュリティ要件がどの程度網羅的に記述されているかについて実態調査を行う．調査においては，独立行政法人情報処理推進機構にて策定された非機能要件グレードに含まれるセキュリティ要件について網羅性を評価する．

2　現時点の調査

これまでに 5 つの地方自治体と 5 つの病院の計 10 個の提案依頼書について評価を行

[*] Ryusuke Murakoshi, 岡山大学工学部情報工学科
[†] Kinari Nishiura, 岡山大学学術研究院自然科学学域
[‡] Mariko Sasakura, 岡山大学学術研究院自然科学学域
[§] Akito Monden, 岡山大学学術研究院自然科学学域

った．データの秘匿化やログの保管，アクセス・利用制限といったセキュリティ要件を達成している団体は多く，内容の質に差はあるものの，開発ベンダとの打ち合わせによって解消する程度だと判断できる．一方，すべての依頼書でセキュリティリスク分析及びセキュリティ診断に関する記述が存在せず，セキュリティリスクの見直しは深刻である．またセキュリティリスク管理に関する記述は 1 つの自治体と 2 つの病院のみであり，加えて開発ベンダが提案することが条件になっている．この結果から，自治体や病院は開発・運用するシステムのセキュリティの実態調査に消極的であると考えられる．このことは，多様化と高度化を繰り返すサイバー攻撃への対抗を難しくしている．

今回調査した病院の提案依頼書には厚生労働省の「医療情報システムの安全管理に関するガイドライン」を遵守するよう記載があり，自治体や病院のセキュリティリスク管理への意識の薄さが上記ガイドラインの内容に由来している可能性がある．そこで上記ガイドラインについても同様に，非機能要件グレードに含まれるセキュリティ要件についての評価を実施した．

評価の結果，セキュリティリスク分析，セキュリティ診断の 2 つに関してはガイドラインに記述があった．そのため提案依頼書にガイドラインの遵守を条件づけることで，依頼者は上記 2 点を満たしたシステム開発・運用を開発ベンダに依頼できる．しかしセキュリティリスク管理については記述があまりなく，セキュリティ要件をガイドラインのみに依拠した提案依頼書では，依頼者が運用開始後のセキュリティリスクの見直しを依頼することは難しくなる．またこのように，提案依頼書の記述内容を外部のガイドラインに依拠することによりセキュリティ要件を満たさなくなる状況が他の団体でも起きているのではないかと予想できる．

3 おわりに

現時点までの調査により上記のような予想を立てたが，提案依頼書の数や団体の種類が十分ではなく他のセキュリティ要件の傾向が分かりづらい．今後は調査対象の拡大や団体ごとの遵守するガイドラインも含めた評価を行いたい．

4 参考文献

[1] Accenture, “Ninth Annual Cost of Cybercrime Study,” March 2019, https://www.accenture.com/us-en/insights/security/cost-cybercrime-study
[2] 警察庁, “令和 3 年におけるサイバー空間をめぐる脅威の情勢等について,” April 2022, https://www.npa.go.jp/publications/statistics/cybersecurity/data/R03_cyber_jousei.pdf
[3] 独立行政法人情報処理推進機構, “令和 2 年度中小企業サイバーセキュリティ対策支援体制構築事業（サイバーセキュリティお助け隊事業）成果報告書,” https://www.ipa.go.jp/security/fy2020/reports/sme/otasuketai_houkoku.html
[4] 独立行政法人情報処理推進機構, “2021 年度 中小企業における情報セキュリティ対策に関する実態調査,” https://www.ipa.go.jp/security/fy2021/reports/sme/index.html
[5] 独立行政法人情報処理推進機構, “非機能要件グレード 2018,” https://www.ipa.go.jp/sec/softwareengineering/std/ent03-b.html

自然言語解析を用いた開発プロジェクトリスク状況の定量化・可視化

Quantification and Visualization of the risk in software development projects with NLP

巴 統哉* 北川 健二† 川上 真澄‡

あらまし ソフトウェア開発プロジェクトの失敗を防ぐには，リスク状況を把握し，早期に検知・対処することが重要である.PMO/QA は多数の開発プロジェクトを監視するが，個々のプロジェクトの状況を継続的に監視することは容易ではない.本研究では，開発プロジェクトの管理文書内の記載から，リスク度合いを定量化する方法を提案する.この定量化手法を用いて時系列でのリスク度合い及びプロジェクト状況サマリの可視化を行い，PMO/QA の監視作業を支援する.

1 はじめに

多数のソフトウェア開発プロジェクトを管理する企業では，プロジェクト失敗によるロスコストが経営課題となっている.プロジェクト失敗を防ぐには，工程遅延や，手戻りなどが発生する前に，原因となるリスクを早期に検知・対処することが重要である.プロジェクトマネージャー(PM)は，担当プロジェクトの管理文書(報告書や各種帳票)や，管理指標などから，リスクを常時分析し，PMO/QA へ情報共有を行う.PMO/QA は，複数プロジェクトを管理し，PM からの情報や管理指標を基にプロジェクトへの支援を行う.管理文書は，管理指標に比べて豊富な情報量を持つため，管理文書を有効活用したリスク分析を行いたい.しかしながら，PMO/QA が管理文書を分析するにあたり，担当者ごとのリスク度合いの認識差異や，継続的に多量の管理文書を読み続けることが困難という課題がある.そのため，管理文書を自動的に処理した定量的な情報に基づき，PMO/QA の監視作業を支援する仕組みが求められている.

2 リスク度合いの定量化

そこで，本研究では，プロジェクト管理文書から，リスク度合いを定量化する方法を提案する.プロジェクトの進行過程において，リスクが顕在化した場合，リスクに関連したキーワードが管理文書中に観測される.ここでは，そのような単語をリスクワードと呼ぶ.管理文書中のリスクワードの出現頻度に注目し，リスクの度合いの定量化を行う.リスク定量化手法として，TF-IDF 法を採用した.TF-IDF 法は，複数の文書の中で，特定の文書にのみ多く存在する単語に高い値を付与する方法である.あるプロジェクトの時間軸で蓄積された管理文書を単位周期ごと(例. 週ごと)の文書に結合し，それらに対して TF-IDF 法を適用する.その後，事前に手作業にて作成したリスクワード辞書を用いて，リスクワードとその他の単語を区別する.TF-IDF 法により，突発的に出現したリスクワードに高い値が付与され，単位時間内の出現単語の値を集計することで該当単位周期におけるリスク定量値を得ることができる.

* Motoya Tomoe, 日立製作所
† Kenji Kitagawa, 日立製作所
‡ Masumi Kawakami, 日立製作所

3　リスク状況の可視化

PMO/QA は個々のプロジェクトの定量値を可視化したものを定期的に見ることで，複数のプロジェクトを監視することができる.そのような機能を提供する可視化手法としてヒートマップとワードクラウドを採用した.図 1(左側)に，リスク定量値をヒートマップにより可視化した例を示す.ヒートマップの横軸を時間軸とし，縦軸をリスクワードの分類 (コスト(C)，納期(D)，品質(Q))とした.データソースの都合で CD の情報量が少なかったため CD 行を統合し，Q の情報量が多かったため、原因と現象で行を分離した.

PMO/QA は，リスク度合いの高い期間(ヒートマップの赤い部分)に起きた状況を把握することで，より具体的な支援を行うことができる.これを実現する可視化手法として，ワードクラウドによるリスクキーワードのサマリ可視化を行った.図 1 (右側)のように，各語の定量値に従って文字の大きさを変化させ，リスクワード分類に従い文字の色を割り当てた.これにより，PMO/QA は，発生したリスク事象や周辺の状況を一瞥に把握することができる.

PMO/QA は全プロジェクトのヒートマップを一覧として確認することで，リスク状況が悪化したプロジェクトを発見し，ワードクラウドを用いて状況の概要を把握することができる.

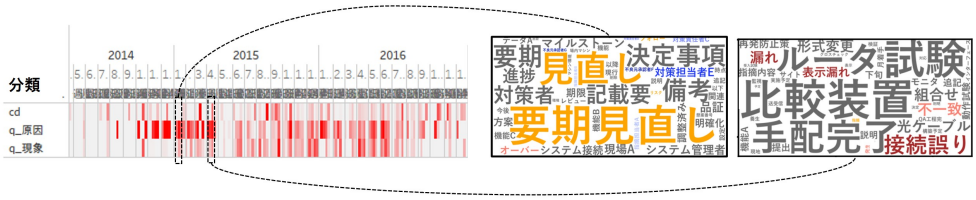


図 1　ヒートマップ，ワードクラウドによる提案手法の適用結果の可視化

4　評価

まず，本手法の評価として，開発の混乱が起きたプロジェクトと起きていなかったプロジェクトをヒートマップで比較した.前者は赤い部分が多く，後者は白い部分が多く，一瞥で見分けられることが分かった.次に，混乱が起きたプロジェクトにおいて問題となった事象が，該当時期のヒートマップ・ワードクラウドから読み取れるかどうか確認した.大きな問題が発生した時期にはヒートマップに赤い部分が連続し，該当時期のワードクラウドに問題に関連する事象が読み取れることを確認できた.これらの確認は，該当プロジェクトの状況を把握していた QA 担当に確認を行っており，実際に役立ちそうな感触が得られた.

5　まとめ

本研究では，TF-IDF 法を用いて，管理文書からプロジェクトのリスク定量値を算出する手法を提案した.また，リスク定量値をヒートマップ，ワードクラウドにより可視化することにより，PMO/QA の監視業務を支援する仕組みを構築した.本研究により，失敗プロジェクトの減少によるロスコストの削減が期待される.

参考文献

[1] PMI: プロジェクトマネジメント知識体系ガイド第 5 版, Project Management Institute.

プログラム理解難易度の経時的な変化の可視化

Visualizing the perceived difficulty in lines and seconds

曾我 遼 * 鹿糠 秀行 † 久保 孝富 ‡ 石尾 隆 § 松本 健一 ¶

あらまし プログラム理解への適切なタイミングでの介入に向け，プログラム理解難易度の経時的な変化の可視化を試みた．

1 はじめに

開発者の生体信号はプログラム理解難易度を反映し，プログラム理解の過程で経時的に変化する [1]．我々は，生体信号からプログラム理解難易度を推定したうえで経時的変化を可視化し，リアルタイムなプログラム理解支援を実現したい．

プログラム理解難易度の経時的な変化を，開発者に細やかに回答させることは現実的ではない．開発者に回答させずに理解難易度を定量化する手法として，開発者の視線や心拍から特徴量を入力とする機械学習アプローチが提案されている [2,3]．これらの研究では，プログラム理解完了時のプログラム理解難易度を可視化しているが，本研究では，プログラム理解過程中の難易度の経時的変化の可視化を試みる．

2 プログラム理解難易度の経時的な変化の可視化

プログラム理解開始から 1 秒毎に，各行のプログラム理解難易度を推定する (図 1)．

まず，プログラム理解タスク中の視線と心拍を収集する (図 1(i))．視線は Eye Tracker 4C (Tobii AB; image sampling rate: 90Hz) を用いて計測し，iTrace [4] を用いて閲覧行番号を算出した．心拍は胸囲心拍計 (Polar H10; Polar Electro) を用いて計測し，PersonalAnalytics [5] を用いて収集した．

次に，視線と心拍を組み合わせて，我々の過去の研究 [3] に従って特徴量を算出する (図 1(ii))．経過時間にわたって最大時間幅 5 分の時間窓を 1 秒ごとに移動させ，時間窓毎に開発者が指定行番号に着目した度合いと心拍指標を算出する．着目度で心拍指標を重みづけ加算した値を加重平均心拍指標として算出し，特徴量とする．

最後に，我々の過去の研究 [3] で訓練したプログラム理解難易度推定モデルへ特徴量を入力し，行単位，時間ごとのプログラム理解難易度を推定する (図 1(iii))．推定モデルの訓練データを収集するため，プログラム理解タスクを実施し，タスク中に視線と心拍を計測し，タスク完了後に行単位のプログラム理解難易度を回答させた．プログラム理解タスクは，被験者がタスク完了したと判断した時点か，プログラム理解開始から 10 分経過した時点で完了とした．回答の収集では，回答行為がプログラム理解難易度に影響を与えないよう配慮した．まず，プログラム理解を妨げないため，秒毎でなく試行終了時に回答させた．続いて，プログラムの再解釈を防ぐため，行毎でなく被験者がプログラムを理解した単位で回答させた．

3 現状と今後の課題

被験者 1 名のプログラム理解タスク 1 試行に対し，プログラム理解難易度の経時的変化を可視化した結果を図 2 に示す．提示プログラムと試行終了時の理解難易度

*Ryo Soga, 株式会社日立製作所

†Hideyuki Kanuka, 株式会社日立製作所

‡Takatomi Kubo, 奈良先端科学技術大学院大学

§Takashi Ishio, 奈良先端科学技術大学院大学

¶Kenichi Matsumoto, 奈良先端科学技術大学院大学

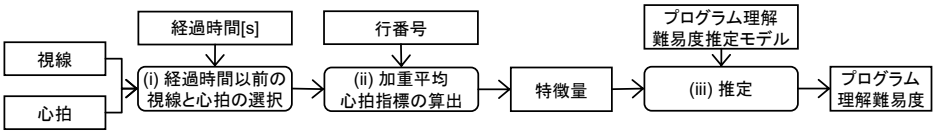


図 1: プログラム理解難易度の経時的変化の推定

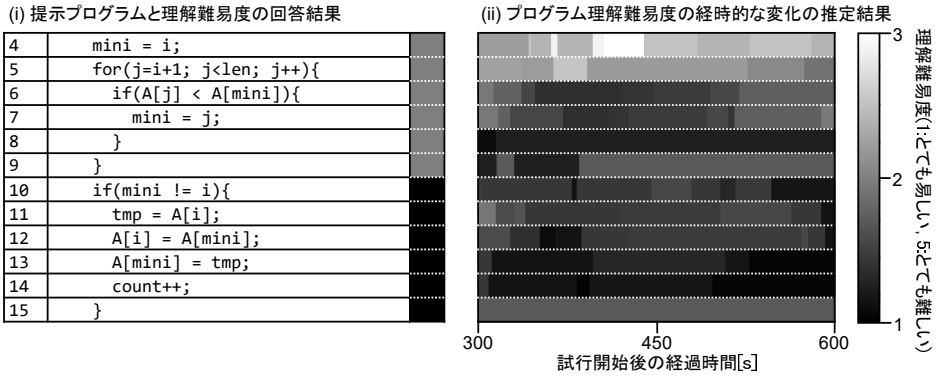


図 2: 提示プログラム及び理解難易度の推定結果

の回答結果を図 2(i) に示し，可視化した結果を図 2(ii) に示す．

被験者がプログラム理解難易度を回答した 4–9 行目と 10–15 行目に対して，600 秒での推定結果を平均したところ，4–9 行目では 1.8，10–15 行目では 1.3 となり，被験者の回答の傾向と一致した．この結果は，推定モデルの妥当性を示す．今後は被験者数と試行数を増やし妥当性の検証を進める．

試行中のプログラム理解難易度が試行終了時より高い行は，7,8,9,15 行目を除く 8 行あった．この結果がもしも本当にこの時点の理解難易度が高かったことを示すなら，経時的な変化の可視化により適時的なプログラム理解支援を実現できる可能性を示唆するものと考えられる．提案手法の時間分解能は心拍計測の時間分解能に依存しており，1 秒より短くすることは難しい．時間分解能 1 秒を前提とした場合，熟練者による PullRequest のマージ可否判断など，プログラム理解時間が短い作業の支援は難しい．一方，初心者が既存プログラムを理解する場合など，プログラム理解時間が長い場合には，特定行の理解難易度が高くなった場合に周囲へアラートを出すといった形で支援できる．今後は，こうした支援策の有用性を検証する．

謝辞 本研究の一部は，JSPS 科研費 JP20H05706 の助成を受けた．

参考文献

[1] T. Ishida et al. Synchronized analysis of eye movement and eeg during program comprehension. In *Proceedings of the EMIP*, p. 26–32, 2019.

[2] H. Hijazi et al. iReview: an Intelligent Code Review Evaluation Tool using Biofeedback. In *Proceedings of the ISSRE*, pp. 476–485, 2021.

[3] 曾我遼, 横山由貴, 鹿糠秀行, 久保孝富, 石尾隆. 視線と心拍を用いた主観的なプログラム理解難易度の推定. ソフトウェア工学の基礎ワークショップ論文集, Vol. 28, pp. 71–80, 2021.

[4] TR. Shaffer et al. iTrace: enabling eye tracking on software artifacts within the ide to support software engineering tasks. In *Proceedings of the ESEC/FSE*, p. 954–957, 2015.

[5] AN. Meyer et al. Design recommendations for self-monitoring in the workplace: Studies in software development. *Proc. ACM Hum.-Comput. Interact.*, Vol. 1, No. CSCW, pp. 1–24, 2017.

ソースコード編集履歴再生器の履歴アノテーションによる拡張

Extension of Source Code Edit History Replayer by History Annotation

大森 隆行 *

あらまし 本稿では，Java プログラム開発における操作履歴に対する履歴アノテーションを用いた情報付加手法，操作履歴グラフ FOHG に基づく履歴アノテーションの生成，履歴アノテーションを扱うための編集履歴再生器の機能拡張によるコード変更理解支援について述べる．

1 はじめに

ソフトウェア保守において，過去にどのようにプログラムが変更されてきたのかを正確に知ることは重要である．改版履歴からは知ることのできない IDE のエディタ上でのソースコード編集過程を記録することで変更理解を支援する手法が提案されている [1, 2]．著者らが以前提案した OperationReplayer（以下 ORep）[3] は，OperationRecorder [4] により記録された Java ソースコード編集時の操作履歴を再生することで，コード変更の詳細な調査を可能とした．ここで再生とは，操作履歴に基づいて復元された過去のある時点のソースコードに対して，実際にその時に行われた編集操作（コード断片の挿入や削除）を１つ１つ適用していくことを意味する．このような履歴の再生は，操作数が多い場合，時間がかかることが難点である．また，履歴自体には操作対象の構文情報，意味情報が含まれないため，ユーザが再生時に復元コードを読み，理解する必要があった．一方，近年，著者らは，編集過程のコードの各状態を解析した結果（抽象構文木 (AST)）を FOHG と呼ぶグラフ構造で保持する手法について研究を進めてきた [5]．FOHG では，変更（移動・改名・内部のコードの一部変更等）が加えられた AST のノードについて，変更前後のノードをエッジで結ぶことで変更を直接的に表現している．本稿では，操作履歴に FOHG のエッジ情報に基づく付加情報（履歴アノテーション）を加えることで ORep での再生時にプログラムの変更の理解を支援する手法を提案する．

2 履歴アノテーションと再生器機能拡張による変更理解支援

以下に提案手法の概要を示す．

- **操作履歴の履歴アノテーションによる拡張**　XML 形式 [4] で保存されている操作履歴に対して付加的な情報を XML 形式の履歴アノテーションとして加える．アノテーションは，アノテーション種別の他，インデックス（何番目の操作に対して情報を付加するか），付加する情報（任意の文字列）により構成される．
- **FOHG が持つ変更情報による履歴アノテーション生成**　FOHG が持つ情報のうち，特に理解に役立つと考えられるエッジの情報を抽出する．本研究では，構文要素の変更，移動，カット＆ペーストを表現するエッジに着目した．エッジの始点・終点にあたるノードが持つ時刻情報に最も近い時刻の（履歴中の）編集操作に対して，抽出した情報をアノテーションとして付加した．付加する情報は，エッジが連結されているノードの種類により異なる．ユニーク名（完全修飾されたクラス名やメソッド名等）を持つ場合はユニーク名，キーワードやリテラルの場合はそれに相当する文字列表現が付与される．
- **ORep における履歴アノテーションの活用**　ORep を拡張し，現在の着目操作

*Takayuki Omori, 立命館大学

に付与されているアノテーションの情報を表示可能とした．この情報は Operation History ビュー（図 1 参照）で確認できる．情報が増えた場合の視認性を改善するため，同ビュー上のツールチップとしても表示する．また，この情報は，ORep のフィルタリング機能により再生対象を絞り込む際にも使用できる．

3　履歴アノテーションによる変更理解支援の事例

図 1 にアノテーションが再生時の変更理解を支援する事例を示す．図 1 では，`"Stone"`という文字列を挿入する編集操作を Operation History ビューにおいて選択しており，Source Viewer では，復元コード中の当該挿入文字列がピンクの背景色で強調表示されている．この編集操作には FOHG に基づくアノテーション（Edge 欄）が付与されており，メソッド`"canPut()"`を`"canPutStone()"`に変更 (`CHG`) していることが読み取れる．操作履歴を見るだけでは文字列の挿入がどのような構文要素に対して行われたのか，それが改名を意図したものなのかどうかを判別することはできない．アノテーションにより，変更意図の推測を速やかに行えるようになり，コード変更の理解支援ができていると考える．

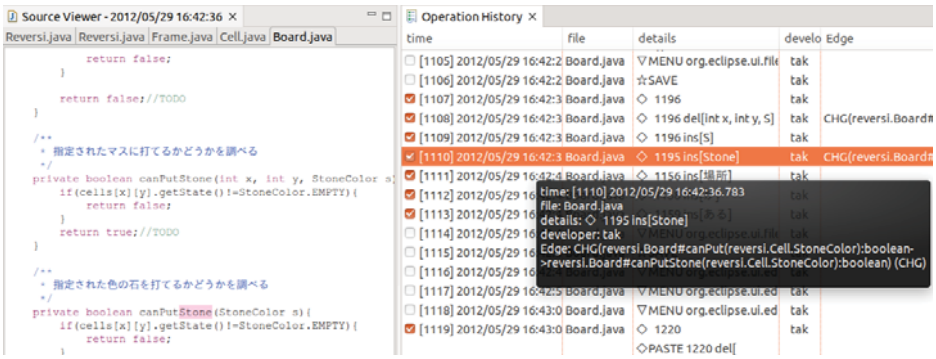


図 1　操作履歴再生における履歴アノテーションの例

4　おわりに

本稿では，履歴アノテーションがプログラム変更理解を支援する事例を示した．これまでの実験では，時刻情報のズレによってアノテーション付加対象がずれてしまうなど，直感的にわかりにくい事例も散見された．今後，より再生時の理解性を高めるよう，手法を改善していく必要がある．

謝辞　本研究の一部は JSPS 科研費 20K11762 による．

参考文献

[1] 大森隆行，林晋平，丸山勝久：統合開発環境における細粒度な操作履歴の収集および応用に関する調査，コンピュータソフトウェア，vol.32, no.1, pp.60–80, 2015.
[2] Soetens, Q.D., Robbes, R., and Demeyer, S., “Changes as First-Class Citizens: A Research Perspective on Modern Software Tooling,”, *CSUR*, vol.50, no.2, pp.18:1–18:38, 2017.
[3] Omori, T. and Maruyama, K., “An Editing-operation Replayer with Highlights Supporting Investigation of Program Modifications,” *Proc. IWPSE/EVOL’11*, pp.101–105, 2011.
[4] `http://www.ritsumei.ac.jp/~tomori/operec.html`
[5] 大森隆行，大西淳：リファクタリング検出のための拡張操作履歴グラフ，ソフトウェア工学の基礎 XXVIII 日本ソフトウェア科学会 FOSE2021，pp.121–126, 2021.

Dockerfileの依存関係とビルドエラーの関係分析

An Analysis of the Relationship between Dependencies and Build Errors of Dockerfiles

坂本 廉也[*] 東 裕之輔[†] 大平 雅雄[‡]

あらまし 本研究の目的は，Dockerfile の自動修正に向けて，Dockerfile の依存関係とビルドエラーの関係を明らかにすることである．本稿では，5,405 件の Dockerfile を対象に行った分析の結果について報告する．

1 はじめに

近年のソフトウェア開発では，プロダクトの迅速なデリバリーを実現する手段の 1 つとして，Docker をはじめとするコンテナ型仮想化技術が普及しつつある．Docker では Docker イメージを基にコンテナを生成する．Docker イメージとはコンテナを動作させるために必要な特定の環境のスナップショットである．Docker イメージは Dockerfile と呼ばれるテキストファイルをビルドすることで作成される．Dockerfile には基本的には 1 行に 1 命令が記述されており，Dockerfile 内の命令を上から逐次実行することでビルドが行われ Docker イメージを作成する．

しかしながら，Dockerfile は，他の言語がベースのプロダクトに比べ，ビルドに失敗しやすいこと [1]，ビルドエラーの修正により多くの時間を要すること [2] がわかっている．そのため，Dockerfile のビルドエラーの修正は迅速な開発の妨げになる可能性がある．

Dockerfile のビルドエラーの多くは，命令間の依存関係により生じる互換性の問題が原因であると指摘されており [1]，エラーメッセージをルールベースで分類した上で修正パッチを生成する手法 [3] が提案されている． [3] の手法では，ビルドエラーが発生した命令行を対象にルールベースを構築しているが，ビルドエラーはそれ以前に実行された命令との依存関係が原因で発生することがある．そのため， [3] の手法では，ビルドエラーが発生する Dockerfile の内，約 18%しか自動でパッチを作成することが出来ないという問題がある．本研究ではまず，Dockerfile 内および Dockerfile 間の依存関係とビルドエラーの関係を明らかにし，Dockerfile のビルドエラー自動修正手法の性能向上のための手がかりを得ることを目的とする．

2 Dockerfileの依存関係

以下に，Dockerfile をビルドする際に発生する 2 種類の依存関係について説明する．

Dockerfile 内の依存関係 Dockerfile は 1 つの命令ごとに 1 つのレイヤーが生成され，Dockerfile 内のある命令はそれ以前に生成されたすべてのレイヤーとの依存関係によりビルドされる．ここで以前の命令とある命令に互換性がなければその Dockerfile はビルドに失敗する．

Dockerfile 間の依存関係 Dockerfile を記述する際には使用したい OS やアプリケーションを既にビルドされたイメージから継承することもできる．継承元のイメージ（ベースイメージ）も Dockerfile からビルドされており，OS やアプリケーションのバージョン情報などのメタデータも含まれている．そのため，Dockerfile の作成者はベースイメージの命令の互換性を意識して命令を記述する必要がある．

[*]Rennya Sakamoto, 和歌山大学

[†]Yunosuke Higashi, 株式会社日本総合研究所，和歌山大学

[‡]Masao Ohira, 和歌山大学

3 分析

以下では，先行研究 [3] で使用されたものと同じ Dockerfile を対象に分析を行う．

3.1 Dockerfile 内の依存関係

Dockerfile 内の依存関係とビルドエラーの関係を分析するために，[3] が公開しているエラー分類に基づいて Dockerfile 内の命令間の依存関係が原因で発生しているエラーを特定した．その結果，ビルドエラーが発生する 5,405 件の Dockerfile のうち，764 件（14.1%）が Dockerfile 内の命令間の依存関係に問題があることが判明した．

3.2 Dockerfile 間の依存関係

Dockerfile 間の依存関係とビルドエラーの関係を分析するために，ビルドエラーが発生する Dockerfile が指定するベースイメージが継承しているイメージの数（継承の深さと呼ぶ）を計測した．ただし本分析では，ビルドエラーが発生する 5,405 件の Dockerfile のうち，ベースイメージを指定していないものやマルチステージビルドを用いている場合などを除いた 1,696 件の Dockerfile を対象とする．

表 1，表 2 にそれぞれ，オフィシャルイメージとユーザイメージが指定しているベースイメージの継承の深さとビルドエラーの関係を示す．表 1, 2 より，オフィシャルイメージ，ユーザイメージともに，ビルドに失敗している件数が大多数を占めているは継承の深さが 0（ベースイメージのみ使用）の Dockerfile であることがわかった．この点については [1] において指摘されている問題を裏付ける結果となった．ただし，継承の深さが 0 以外のものも無視できない程度にはビルドエラーが発生しており，今後はエラー原因の詳細な分析が必要である．

表 1　オフィシャルイメージの継承の深さとビルドエラーの関係

深さ	0	1	2	3	4	5	合計
件数（件）	1,178	146	13	7	65	1	1,410
割合（%）	83.5	10.3	0.9	0.4	4.6	0	100

表 2　ユーザイメージの継承の深さとビルドエラーの関係

深さ	0	1	2	3	4	5	合計
件数（件）	223	44	19	8	1	1	286
割合（%）	77.9	15.3	6.6	2.7	0.3	0.3	100

4 おわりに

本稿では Dockerfile の依存関係とビルドエラーの関係について分析を行った．今後はビルドエラーの原因を理解するためにビルドエラーが発生しやすいイメージの種類や特徴を詳細に分析する予定である．

謝辞　本研究の一部は，文部科学省科学研究補助金（基盤（C）：22K11974）による助成を受けた．

参考文献

[1] Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S. and Gall, H. C.: An Empirical Analysis of the Docker Container Ecosystem on GitHub, *Proceedings of MSR '17*, pp. 323–333 (2017).
[2] Wu, Y., Zhang, Y., Wang, T. and Wang, H.: An Empirical Study of Build Failures in the Docker Context, *Proceedings of MSR '20*, pp. 76–80 (2020).
[3] Henkel, J., Silva, D., Teixeira, L., d'Amorim, M. and Reps, T.: Shipwright: A Human-in-the-Loop System for Dockerfile Repair, *Proceedings of ICSE '21*, pp. 1148–1160 (2021).

ゲーム要素を取り入れた情報リテラシー教育の提案

Proposal for Game Based Education in Media Literacy

廣瀬 司 * 岡本 克也 † 中谷 多哉子 ‡

あらまし 本稿では，インシデントに対する情報リテラシー教育を二段階で行う必要があることを示し，実際にこの手法を適用した結果によって，その有効性を明らかにする．インターネットの利用が日常的な行為となった．それに伴いインターネットを利用した時の事件および事故が増加した．利用者が情報セキュリティのインシデントにどう対処すべきかを考えられるようになる教育が必要である．我々が提案する教育手法では，インシデントの情報を収集して説明する場面と，インシデントの解決策を考える場面の 2 つの場面から構成されるゲームを適用する．教育にゲーム要素を取り込むことによって，参加者の学習意欲を継続させる効果を期待できる．提案する手法の有効性を評価するために，社会人に教育を実施した．その結果，有効性が明らかになった．

1 はじめに

本稿では，インシデントに対する情報リテラシー教育を二段階で行う必要があることを示し，実際にこの手法を適用した結果によって，その有効性を明らかにする．インターネットの利用が日常的な行為となった．それに伴いインターネットを利用した時の事件および事故が増加した [1]．事件および事故の被害防止のために，利用者が情報セキュリティのインシデントにどう対処すべきかを考えられるようになる教育が必要である．ICT 技術の高度化に伴い，さまざまな新しい事件および事故が発生する．したがってインシデントに関するリテラシー教育は，継続して行うことが望ましい．本研究では，インシデントに対する情報リテラシー教育にゲーム要素を取り込むことで，学習意欲を継続させるための手法を開発した．本稿では，インシデントに対する情報リテラシー教育を二段階で行う必要があることを示し，実際にこの手法を適用した結果によって，その有効性を明らかにする．

2 提案手法

- Game Based Education の提案：教育にゲーム要素を用いた研究は様々な分野で行われ，成果を上げている [3]. 学ぼうと思う参加者の学習意欲を継続させるための手法を開発した．本提案手法でもゲームの要素を取り入れる．
- ゲームの構造 : 参加者がインシデントについて深く理解するために，2 つのプロセスが必要である．第一のプロセスでは，参加者が多くのインシデントの情報を得て，それらのインシデントの原因を分析して考える．第二のプロセスでは，インシデントが発生した際の対策や予防策を考える．また，それらの対策や予防策が有効でない場合の対処について考える．これらの 2 つのプロセスを提供するために，本研究では，2 つの場面によって構成されるゲームを開発した．
- ゲームの内容：ゲームとして対戦できるように，参加者を複数のグループに分ける．第一の場面では，インシデント発生について参加者が考えて説明して議論できるようにする．各グループ内で参加者はインシデントの紹介を順に行い，それぞれのインシデントの重大さを説明して比較し，順位付けを行う．各グループ内で議論の上もっとも重大だと思われるものを選出し，そのインシデントを

*Tsukasa Hirose, 放送大学

†Katsuya Okamoto, 放送大学

‡Takako Nakatani, 放送大学

説明した人を勝者にする．
次に，ゲームの第二の場面では, 攻守に分かれて複数のグループが対戦する．守る側はインシデントを説明し，攻める側は解決策を提案し，守る側は解決策の不備を指摘する．議論が終わったところで攻守を入れ替える．最終的に解決策の有効性によってファシリテータがグループ対抗の勝敗を決める．

- 導入教育:ゲームに先立ってあらかじめ，ファシリテータが情報リテラシーにおけるインシデントとは何か，解決策にはどのようなものがあるか，ゲームを行うための基礎知識として，基本的な知識を参加者に提示する．導入教育はゲーム中に参加者が雑談に逸脱することを予防する効果がある [2] ので，参加者の注意がインシデントに十分当てられるように行う．

3 実施結果

我々は，提案する手法の有効性を評価するために，ゲームを実施した．2022 年 6 月に社会人 8 人がゲームに参加した．このゲームによって提案する手法の有効性を，参加者にインシデントに対する学習を継続する意欲ができたかどうかで．評価した．

このゲームでは ZOOM と Jamboard を使った．評価は，参加者にゲーム終了後，アンケートを配布し，その回答をもとに行った．以下の回答があった

- 他人のインシデントを聞いて自分自身の振返りができた．
- 危険だと思う基準がみんな違うところに面白みがあった．
- インシデントに直面したときに誰かに相談しようという気になった．
- 情報リテラシーを継続して学びたい．
- 新たなメンバーでこのゲームを行い，新たな話を聞きたい．

第一の場面では，参加者はインシデントについて理解し，それぞれ参加者が適切だと思うインシデントを選出することができた．また，グループ内の他人と参加者自身のインシデントについて検討できた．第二の場面では，さまざまな解決策の提案があったが，守る側からの多くの事情や言い訳により，解決策は却下され，さらなる解決策の議論を重ねることができた．

また，提案したゲームの継続的な参加意欲を確認したところ，継続して学びたいという参加者や，このゲームを使って新たなメンバーの話が聞きたいという参加者がいた．さらに，ゲーム実施から 2 週間後に参加者から，身の回りのインシデントに気づくことが多くなったという意見があった．

4 まとめ

本稿では，インシデントに対する情報リテラシー教育を二段階で行う必要があることを示し，実際にこの手法を適用した結果によって，その有効性を明らかにした．ゲーム要素を取り入れたことで，ゲームの参加者に学習の継続意欲がみとめられた．ゲームを二段階にしたことで，第一の場面ではインシデントの収集と第二の場面では解決策の検討，および解決策の妥当性についてそれぞれ集中して議論する環境を提供できた．

今後は，インシデントの理解度および解決策の議論の質について定量的に評価する手法を取り入れて，ゲームの完成度を高める．

参考文献

[1] JPCERT コーディネーションセンター,JPCERT/CC インシデント報告対応レポート：2022 年 1 月 1 日 2022 年 3 月 31 日, 計装 65 (6), 50-56, 2022-06
[2] 廣瀬 司, 中谷 多哉子, インシデントリテラシ向上のための情報セキュリティゲームに導入教育を採用することの検証, ソフトウェアシンポジウム論文集,6,86–93,2020 論文集,
[3] 藤本徹, ゲーム学習の新たな展開 , 放送メディア研究,12,233–252,2015, 丸善プラネット,

スマホゲームにおける課金誘導方法の調査研究

An Analytical Study on How to Induce Billing in Smartphone Games

麻生直希*　西浦 生成†　笹倉 万里子‡　門田 暁人§

あらまし　スマートフォンのゲームアプリ（スマホゲーム）の普及に伴い，ゲームに過度に課金するユーザ（いわゆる重課金，廃課金プレイヤー）の存在が問題となっている．スマホゲームには，ユーザが課金したくなるような仕組みや，課金を繰り返させるような仕組みが設けられており，そのような課金誘導方法を知ることは，やみくもな課金を避ける上で重要であると考える．本研究では，複数の代表的なスマホゲームを実際にプレイし，課金誘導方法を調査する．

1　はじめに

スマートフォンの世界的な普及に伴い，スマートフォン上のゲームアプリ（スマホゲーム）の市場は成長を続けている．Sensor Tower 社の調査[1]によると，スマホゲーム市場における 2021 年の総売上高は 896 億ドルであり，2020 年に比べて 12.6%増加している．また，2021 年のスマホアプリ全体に占めるスマホゲームの売上高の割合は 67.4%であり，スマートフォン利用におけるゲームの位置付けは極めて大きいといえる．特に，日本人の課金額は世界の中でも極めて多く，Sensor Tower 社の別の調査[2]によると，2021 年の第 1～第 3 四半期のスマホアプリに対する一人当たりの課金額は，日本は 149 ドルで第 1 位であり，2 位の韓国 95 ドル，3 位の米国 90 ドルと比べても突出している．

スマホゲーム市場が成長する一方で，未成年がゲーム中の課金アイテムを歯止めなく購入し，多額の請求が行われるという事例も多数報告されるようになった．また，常識的な金額を大きく超えるような課金を行う重課金，廃課金プレイヤーと呼ばれるユーザが存在し，日常生活に支障をきたす事例も報告されている．国民生活センターに寄せられるスマホを含むオンラインゲームに関する相談件数は，年々増大する一方である[3]．

本研究では，複数の代表的なスマホゲームを実際にプレイし，課金誘導方法を調査する．本研究の調査結果を広く公表し，ゲームのユーザに知ってもらうことができれば，やみくもな課金への依存を低減できるのではないかと期待する．

2　現時点の調査結果

表 1 は，Android Google Play のセールスランキング上位の 5 つのゲーム，(1)モンストライク（モンスト），(2)ドラゴンボール Z ドッカンバトル（ドカバト），(3)パズル&ドラゴンズ（パズドラ），(4)実況パワフルプロ野球（パワプロ），(5)Pokémon GO（ポケ GO）について調査した結果をまとめたものである．表 1 より，多くのゲームでは「ガチャ」と呼ばれる，確率により報酬（キャラクタやアイテム）が得られる方式を採用しており，これはパチンコのようなギャンブル依存者を生み

* Naoki Asou, 岡山大学工学部情報工学科
† Kinari Nishiura, 岡山大学学術研究院自然科学学域
‡ Mariko Sasakura, 岡山大学学術研究院自然科学学域
§ Akito Monden, 岡山大学学術研究院自然科学学域

表１　5 つのゲームにおける課金誘導方法

分類	課金誘導方法	モンスト	ドカバト	パズドラ	パワプロ	ポケGO
お得なガチャ	ステップアップガチャ		○		○	
	ガチャ保証	○	○	○		
	期間限定ガチャ	○	○	○	○	
	コラボガチャ	○	○	○	○	
お得な課金	サブスク課金	○	○	○		
	課金セール	○	○		○	
	アイテムパック	○	○	○	○	○
	初心者パック	○			○	
	回数限定パック	○	○	○	○	○
	コラボ課金	○		○		
課金者特権	上限撤廃	○	○	○	○	○
	キャラ強化		○	○	○	○
時間短縮	時間短縮	○	○	○	○	
キャラ成長	同キャラ複数保持で成長	○	○	○	○	
アツい演出	高期待値のガチャ演出	○	○		○	

出す仕組みに類似している．また，様々な種類の「お得なガチャ」「お得な課金」が設けられており，通常よりも安い課金で報酬が得られるため，安さやお得感に釣られて課金してしまうユーザが少なくないと考えられる．また，課金者が有利となるような仕組み（課金者特権）や，課金による時間短縮も多くのゲームに設けられており，他のプレイヤーに勝ちたい，あるいは，短時間でゲームを進めたいユーザが安易に課金してしまう恐れがある．さらに，多くのゲームにはキャラクタを成長させる要素を持っており，思い入れのあるキャラクタを成長させたいユーザに訴求している．また，ガチャを回す上で，いわゆる「アツい演出」が見られる場合があり，この演出が出ると目玉のアイテムやキャラクタを獲得できる，もしくは獲得の確率が高くなる．このような高期待値の演出は，パチンコやパチスロにおいても「激アツ」と呼ばれており，ユーザの感情をゆさぶることで脳内の興奮物質ドーパミンを放出させ，ガチャを回すこと自体が快感に繋がるようになる可能性がある．

3　おわりに

自身の経済状況に合わせてゲームを楽しむ上で，ゲームの課金誘導方法を知り，むやみに課金してしまうことを避けることが重要であると考える．

4　参考文献

[1] Stephanie Chan, “Global Consumer Spending in Mobile Apps Reached $133 Billion in 2021, Up Nearly 20% from 2020,” Sensor Tower Blog, Dec. 2021, https://sensortower.com/blog/app-revenue-and-downloads-2021.
[2] Stephanie Chan, “Japan Leads Mobile App Spending Per Capita in 2021, Climbing 18% Over 2020 to Nearly $150,” Sensor Tower Blog, Oct. 2021, https://sensortower.com/blog/per-capita-app-spending-q1-q3-2021.
[3] 独立行政法人国民生活センター, “各種相談の件数や傾向＞オンラインゲーム,” https://www.kokusen.go.jp/soudan_topics/data/game.html