ソフトウェア・シンポジウム 2021 (オンライン開催)

論文集

ソフトウェア技術者協会

募集案内

ソフトウェア・シンポジウムは，ソフトウェア技術に関わるさまざまな人びと，技術者，研究者，教育者，学生などが一堂に集い，発表や議論を通じて互いの経験や成果を共有することを目的に，毎年全国各地で開催しています．

第 41 回目を迎える 2021 年のソフトウェア・シンポジウムでは，この数年間で試みてきた新しい取り組み（チュートリアルや Future Presentation など）をさらに発展させたものにしたいと考えています．このほか，SS2020 に引き続き，論文発表や事例報告と，ワーキンググループで議論を行います．

開催概要

■ 日程 ： 2021 年 6 月 2 日（水曜日）～ 4 日（金曜日）

■ 場所 ： J:COM ホルトホール大分 を予定しておりましたが，
COVID-19（coronavirus disease 2019） 感染症対策として，オンライン開催に変更しました

■ 主催 ： ソフトウェア技術者協会

■ 後援 ： 情報処理推進機構，ハイパーネットワーク社会研究所

■ 協賛 ： 大分県情報サービス産業協会，ソフトウェアテスト技術振興協会，アジャイルプロセス協議会、オープンソースソフトウェア協会，情報サービス産業協会，情報処理学会，ソフトウェア・メインテナンス研究会，電子情報通信学会，日本ソフトウェア科学会，組込みシステム技術協会，日本 SPI コンソーシアム，日本ファンクションポイントユーザ会，派生開発推進協議会，日本科学技術連盟，組込みソフトウェア管理者・技術者育成研究会，TOPPERS プロジェクト，PMI 日本支部

スタッフ一覧

実行委員会

実行委員長

森 秀文（大分県情報サービス産業協会）
吉田 和幸（大分大学）

実行委員

青木 栄二（ハイパーネットワーク社会研究所）
荒木 啓二郎（熊本高等専門学校）
伊藤 昌夫（ニルソフトウェア）
市川 尚（岩手県立大学）
漆原 憲博（ジェーエフピー）
小笠原 秀人（千葉工業大学）
小楠 聡美（HBA）
紙名 哲生（大分大学）
岸田 孝一（SRA）
栗田 太郎（ソニー）
小松 久美子（帝塚山学院大学）
杉田 義明（福善上海）
鈴木 裕信（usp lab.）
富松 篤典（電盛社）
中野 秀男（中野秀男研究所）
奈良 隆正（NARA コンサルティング）
布川 博士（岩手県立大学）
野村 行憲（岩手県立産業技術短期大学校）
本多 慶匡（東京エレクトロン）
宮田 一平（SHIFT）
宮本 虎汰郎（ザイナス）
三輪 東（SCSK）
山本 竜伸（ザイナス）

プログラム委員会

プログラム委員長

梅田 政信（九州工業大学）
小田 朋宏（SRA）
日下部 茂（長崎県立大学）

プログラム委員

秋山 浩一（ASTER）
安達 賢二（HBA）
鰺坂 恒夫（武庫川女子大学）
天嵜 聡介（岡山県立大学）
荒木 啓二郎（熊本高等専門学校）
伊藤 昌夫（ニルソフトウェア）
臼杵 誠（富士通）
大平 雅雄（和歌山大学）
小笠原 秀人(千葉工業大学)
小楠 聡美（HBA）
小田 朋宏（SRA）
落水 浩一郎（University of Information Technology, Myanmar）
片山 徹郎（宮崎大学）
紙名 哲生（大分大学）
喜多 義弘（長崎県立大学）
北須賀 輝明（広島大学）
楠本 真二（大阪大学）
熊澤 努（SRA）
栗田 太郎（ソニー）
河野 哲也（ディー・エヌ・エー）
後藤 徳彦（NEC ソリューションイノベータ）
古畑 慶次（デンソー）
小林 展英（デンソークリエイト）
小林 修（SRA）
阪井 誠（SRA）

プログラム委員会

酒匂 寛（デザイナーズデン）
佐原 伸（法政大学）
菅原 広行（ソニー）
鈴木 裕信（usp.lab.）
鈴木 正人（北陸先端科学技術大学院大学）
高木 智彦（香川大学）
田中 康（奈良先端科学技術大学院大学）
張 漢明（南山大学）
角田 雅照（近畿大学）
土肥 正（広島大学）
富松 篤典（電盛社）
中谷 多哉子（放送大学）
中西 恒夫（福岡大学）
中森 博晃（パナソニック）
西 康晴（電気通信大学）
根本 紀之（東京エレクトロン）
野村 行憲（岩手県立産業技術短期大学校）
野呂 昌満（南山大学）
端山 毅（NTT データ）
久住 憲嗣（芝浦工業大学）
藤川 昌雄（富士通九州ネットワークテクノロジーズ）
本多 慶匡（東京エレクトロン）
松尾谷 徹（デバッグ工学研究所）
松本 健一（奈良先端科学技術大学院大学）
水野 修（京都工芸繊維大学）
宮本 祐子（宇宙航空研究開発機構）
宗平 順己（武庫川女子大学）
村上 純（熊本高等専門学校）
森崎 修司（名古屋大学）
八木 将計（日立製作所）
山本 修一郎（名古屋国際工科専門職大学）
米島 博司（パフォーマンス・インプルーブメント・
アソシエイツ）
劉 少英（法政大学）
渡辺 律子（ハイパーネットワーク社会研究所）

事務局

栗田 太郎（ソニー）

ソフトウェア・シンポジウム 2021 （オンライン開催） 目次

■論文・報告 ［アーキテクチャ・設計］

[研究論文] 機能共鳴分析法の図式表現を用いた視覚的シナリオベース設計支援
日下部 茂 (長崎県立大学) …… 1

[研究論文] ゴール指向要求分析とシステム安全分析を利用した
AI システム品質の個別ガイドライン導出方法の提案
相津 一寛 (パナソニック株式会社), 石川 冬樹 (国立情報学研究所),
小宮山 英明 (コニカミノルタ株式会社), 栗田 太郎 (ソニー株式会社),
柳原 靖司 (ブラザー工業株式会社), 徳本 晋 (富士通株式会社) …… 9

[経験論文] DX 推進に向けたエンタープライズアーキテクチャの設計プロセスの検討
宗平 順己 (武庫川女子大学) …… 19

■論文・報告 ［UI/可視化/シミュレーション］

[経験論文] リアルタイムトラフィック可視化システムとその拡張性について
吉田 和幸 (大分大学), 宇野 秀亮 (大分大学), 池部 実 (大分大学), 吉崎 弘一 (大分大学)
…… 29

[研究論文] 個体群動態論に特化したマルチエージェントシミュレーション基盤 Re:Mobidyc
小田 朋宏 (株式会社 SRA), Gael Dur (静岡大学) …… 35

[研究論文] スマートフォンにおけるフリック操作による画面ロック解除手法の提案
喜多 義弘 (長崎県立大学), 山下 湧介 (長崎県立大学) …… 45

■論文・報告 ［地域連携・産学連携］

[事例報告] 大分県における GPU 活用に向けた取組み
石松 博文（公益財団法人ハイパーネットワーク社会研究所），
青木 栄二（公益財団法人ハイパーネットワーク社会研究所），
渡辺 律子（公益財団法人ハイパーネットワーク社会研究所），
原田 美織（公益財団法人ハイパーネットワーク社会研究所） …… 53

[経験論文] 深層学習を用いた路線バスにおける
乗客 OD および混雑状況の推定手法の開発
山下 倫央（北海道大学），西浦 翼（北海道大学），久保 田遼裕（北海道大学），
横山 想一郎（北海道大学），川村 秀憲（北海道大学），弓崎 潔（株式会社 シーズ・ラボ），
佐藤 好美（株式会社 シーズ・ラボ） …… 54

[事例報告] 人工知能技術の産業応用に向けた産学連携に関する事例報告
山下 倫央（北海道大学大学院情報科学研究院） …… 64

■論文・報告 ［組み込み・制御］

[経験論文] 組込みシステムへの派生 MBD 適用に向けたパラメータ設定方式の検討
秋下 耀介（株式会社日立製作所），大島 浩資（株式会社日立製作所），
若林 昇（株式会社日立製作所），川上 真澄（株式会社日立製作所） …… 65

[研究論文] シェルスクリプトを用いた UNIX 哲学に基づくリアルタイム制御
柳戸 新一（有限会社 USP 研究所），松浦 智之（有限会社 USP 研究所），
鈴木 裕信（有限会社 USP 研究所），大野 浩之（金沢大学） …… 73

[経験論文] UNIX 機における IoT 機器制御のためのタイミング管理
松浦 智之（有限会社 USP 研究所），柳戸 新一（有限会社 USP 研究所），
鈴木 裕信（有限会社 USP 研究所），大野 浩之（金沢大学） …… 82

■論文・報告 ［ソースコード解析］

[研究論文] 構成管理ツールにおける命令的操作が及ぼすソースコードレビューへの影響調査
頭川 剛幸（京都工芸繊維大学），近藤 将成（九州大学），
西浦 生成（京都工芸繊維大学），崔 恩瀞（京都工芸繊維大学），
水野 修（京都工芸繊維大学） …… 92

[研究論文] 深層学習を用いたプログラム品質向上のための
ソースコード画像分析手法の提案
小川 一彦（放送大学大学院），中谷 多哉子（放送大学大学） …… 102

[研究論文] Arduino プロジェクトにおける Example Sketch の再利用分析
寺村 英之（京都工芸繊維大学），崔 恩瀞（京都工芸繊維大学），
近藤 将成（九州大学），吉田 則裕（名古屋大学），
水野 修（京都工芸繊維大学） …… 112

■論文・報告 ［教育］

[事例報告] 子供たちの学びたい欲求を引き出すオンラインツール
根本 紀之（アジャイル札幌），野尻 育代（北海道北見市立上仁頃小学校教諭）
…… 122

[事例報告] 高専専攻科におけるテンソルデータ処理プログラミング教育の実践
山本 直樹（熊本高等専門学校），石田 明男（熊本高等専門学校），
村上 純（熊本高等専門学校） …… 124

[事例報告] WHY に重点を置いたプロセストレーニングの改善
片山 泰司（オムロン ソーシアルソリューションズ株式会社） …… 125

■論文・報告 ［品質・信頼性］

[研究論文] アリコロニー最適化法を用いたモデル検査の効率化のための探索戦略
熊澤 努（株式会社 SRA），滝本 宗宏（東京理科大学），神林 靖（日本工業大学）
…… 126

[研究論文] 要求仕様に対するテストカバレッジ分析におけるグラフクエリの適用について
有若 新悟（大阪大学），中川 博之（大阪大学），土屋 達弘（大阪大学）
…… 136

[研究論文] ソフトウェア信頼度成長モデルとベイズ統計機械学習による
オープンソースソフトウェア動的信頼性モデルの提案
杉山 透（放送大学），中谷多哉子（放送大学） …… 142

■論文・報告 ［自然言語分析］

[研究論文] 要求仕様に対する形態素ベースドレビューの提案
柏原 一雄（株式会社デンソークリエイト），鈴木 淳（株式会社デンソークリエイト），
新留 光治（株式会社デンソークリエイト），松原 潤弥（株式会社デンソークリエイト），
李 路雲（株式会社デンソークリエイト），不破 慎之介（株式会社デンソークリエイト）
…… 152

[研究論文] 自然言語処理的アプローチによるクラス図関連線の予測
井原 輝人（奈良工業高等専門学校），内田眞司（奈良工業高等専門学校），
飯田 元（奈良先端科学技術大学院大学） …… 162

[研究論文] クラス名における命名バグの検出手法
有村徳崇（大分大学），紙名哲生（大分大学） …… 172

■論文・報告 [Future Presentation]

[Future Presentation (1)] リアクティブプログラミングに基づく分散計算基盤の可能性
紙名 哲生 (大分大学) …… 178

[Future Presentation (2)] マンモスの牙 ～ Monetize Metrics Vs. Engineering Metrics ～
松尾谷徹 (有限会社デバッグ工学研究所),
増田礼子 (フェリカネットワークス株式会社) …… 182

機能共鳴分析法の図式表現を用いた視覚的シナリオベース設計支援

日下部 茂
長崎県立大学
kusakabe@sun.ac.jp

要旨

本稿ではソフトウェアシステムの開発上流工程におけるシナリオ分析に機能共鳴分析法 FRAM の図式表現を用いることを論じる. シナリオは，人やシステムの機能や活動に関する記述であり， 開発されるソフトウェアシステムにより何が実行されるのか，どのような効果がもたらされるのかなどを記述できる. 自然言語で記述されたシナリオは，ソフトウェアの専門家でなくとも理解や誤りの修正など可能であるが，文字情報の読み込みを前提とする. このようなシナリオの記述や分析を，機能や活動に着目してモデル化と分析を行う FRAM を用いて支援することを提案する. FRAM は図式表現を持っており，視覚的で直観的な理解の促進といったシナリオ分析支援が可能と考えた. 本稿ではこのような FRAM の図式表現を用いたシナリオ分析支援の方法と事例について論じる.

1. はじめに

本稿では，ソフトウェアシステムの開発上流工程の，特にシナリオ分析に，機能共鳴分析法 (Functional Resonance Analysis Methods, 以降 FRAM と呼ぶ) [1]の図式表現を用いることを論じる. シナリオは，人やシステムの機能や活動に関する記述であり， 開発されるソフトウェアシステムにより何が実行されるのか，どのような効果がもたらされるのかなどを記述できる. このようなシナリオの記述や分析を，機能や活動に着目してモデル化と分析を行う FRAM を用いて支援することを提案する.

自然言語で記述されたシナリオは，特別な知識やスキルなどなくともシナリオの理解や誤りの修正など可能で，多様なステークホルダが関わる上流工程での要求の記述や分析にも活用できる. しかしながら，開発側の要求分析者とステークホルダでは，経験も知識も使用する用語なども異なることがあり，自然言語以外の表現により，コミュニケーションを円滑にとりながら要求の獲得やその記述，分析を支援することは重要と考える. FRAM はテキスト表現だけでなく図式表現もあり，文字情報の読み込みを前提としない，図式表現による視覚的で直観的な理解も含め，シナリオ分析を支援できると考えた.

FRAM では，機能や活動を記述の単位としたモデル記述を行う. 後述するように各機能や活動は，入力(I)，前提条件(P)，時間(T)，資源(R)，制御(C)，出力(O)という六つの側面を持ち得る. このような側面を通して機能間のつながりを記述し，機能間の相互作用をモデル化し分析する. このような FRAM は，もともとレジリエンスエンジニアリングのもので，その分析法に厳密で唯一絶対的なものはない[2]. 本稿では，FRAM の図式表現を，二次元の半順序グラフと考え，機能や活動の結合関係も意識し，記述ルールや便宜的補助線などを追加する方法を論じる. このような方法によりレジリエンスエンジニアリングだけでなくソフトウェア工学，特にシナリオベース設計[3]に有効な記述や分析が促進されるとの仮説を立てた.

本稿では，そのような仮説の下，記述ルールや便宜的補助線など追加して行うFRAMの図式表現やシナリオの記述、分析の方法について，特にスマートフォン向けアプリを対象に説明を行う. 要求工程でのシナリオ記述時に，ユーザおよびスマートフォンの抽象的な機能や活動と，それらの間のつながりを記述した FRAM の図式表現も作成する. その際，機能・活動の実行主体を明示化するといったいくつかの記述ルールを追加し，ユーザストーリーマッピング[4]を意識した記述を行う.

ユーザストーリーマッピングでは，ストーリー(ユーザにとっての価値)を付箋紙などに書き出し，ユーザの体験順に時系列的には左から右に，似た機能は上下(基本機能を上，派生的な機能は下)になるように，壁などに配置していく. ここでは付箋紙ではなく，FRAM の記法を用いて，以下のような手順で記述，分析を進める.

1. 自然言語によって記述されたシナリオからFRAMの図式表現を作成する.
2. ユーザ視点のストーリーの中で，開発者が生起を想定しているユーザ行動の明確化と，そのユーザ行動が生起する仕組みの分析のため，FRAMの図式表現と行動分析[5]の ABC 分析と組み合わせて行う.

3. FRAM の図式表現上で，ソフトウェアの MVP (Minimal Viable Product) 構造と，付加価値を高める追加の開発を意識した表現を行い，工程設計も支援する．
4. 画面のシナリオベース設計のための記述と分析を行う．スマートフォンのアプリケーションでは，画面上のユーザインターフェイス（以降 UI）も重要な役割を持ち Android アプリケーションでは画面単位で Activity クラスを開発する必要がある．このような点をふまえ，画面を介した機能の相互作用の記述と分析も FRAM の図式表現上で行う．

本稿では，著者が所属している組織の大学・学部レベルでのスマートフォン(Android)のアプリケーションの設計演習の事例を使い，提案する手順について説明する．その演習講義の受講生は学部 3 年生で，オブジェクト指向プログラミングは入門レベル，UML はその概要を知っているが記述したことはほとんどないというレベルの学生が多数派である．例題としては Android アプリケーション開発の入門書[6]の例題の一つ，じゃんけんゲームを用い，提案の手順に加え，演習講義の受講学生に対して行ったアンケート結果も併記しながら説明を行う．

本稿の構成は次の通りである．2 章で FRAM について説明する．3 章で上記の手順 1 である FRAM とそのシナリオ分析への適用，4 章で手順 2 である行動分析とそのシナリオ分析への適用，5 章で上記手順 3 と 4 について，それぞれの手順とアンケートによる評価結果について述べる．最後にまとめと今後の課題を述べる．

2. FRAM

2.1. FRAM の概要

FRAM は複数の組織や人，機器といった構成要素が相互作用を行うシステムを対象にした，レジリエンスエンジニアリングの手法として提案されている．従来の安全性と異なる新しい安全性や，回復・復元力について広義のエンジニアリングを行う．従来のように，安全性を「物事が悪い方向へ向かわない状態」としてアプローチするものは Safety-I，「物事が正しい方向へと向かうことを保証する」と考えてアプローチするものを Safety-II として区別し，Safety-II の考えに基づくモデリング手法の一つとして提唱されている．以下の四つの原則に基づいて，ものごとが悪い方向に向かうことより，正しい方向に向かうことに焦点を当てて分析を行う．

① 成功と失敗の等価性の原則，
② 近似的な調整の原則，
③ 創発の原則，
④ 機能共鳴の原則

FRAM では，機能について以下に示す六つの側面によってモデル化を行う．

- 入力(I)：機能が処理の対象とし，変化するもの．あるいは機能を開始するもの．
- 前提条件(P)：機能が実行される前に存在・成立すべき条件．
- 時間(T)：機能に影響を与える時間的制約(開始時刻，終了時刻，継続時間など)
- 資源(R)：機能が実行されるときに必要とされるもの，あるいは出力を提供するために消費されるもの．
- 制御(C)：機能がどのようにモニターされ，あるいは制御されるか．
- 出力(O)：機能の結果，何らかの実体か状態変化．

FRAM の図式表現では，図 1のように，各機能を六角形で示す．図式表現でのモデリングでは，典型的には FMV (FRAM Model Visualizer) のようなツールを用いてモデルを記述し，視覚化する[7]．

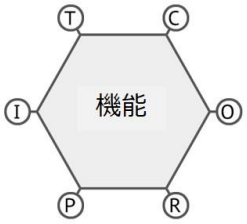


図 1 機能の図式表現

2.2. FRAM の分析

FRAM の分析には，複数機能間の相互作用に着目した，システム特性の分析が含まれる．機能の共鳴の観点から，複数の機能が相互に作用した結果，外乱に柔軟に対応して回復する場合もある一方，逆に変動の影響が増幅し，安全を脅かす場合もあるといった分析も行う．

FRAM では機能に着目したモデリングを行い，機能の目的，行っている処理，存在する入出力といったものを明らかにし，機能に名前を付け定義する．定義を進めるにあたり，各側面の要素と名前，相手の機能の名前も明らかにする．しかしながら，モデルの構築や分析の具体的な方法に唯一絶対のものはない．例えばモデルのテキスト表現を重要視したものもある一方[1]，図式表現でのネットワークトポロジーに着目した分析法もあり[8]，文

献[8]では以下のような分析の観点が紹介されている.

- 機能が受け取るもの
- 条件が変わった場合の適応
- 正常ではない条件への反応
- 正常ではない条件の発生頻度
- リソース供給は安定的か, 不安定要因は何か
- 外部要因はどのくらい安定か, 不安定要因は何か
- 前提条件の取りこぼしはないか
- 時間制約にかかるプレッシャーの場所
- 特別なスキル, 機能, 高信頼性の必要箇所
- 最適な実行方法はあるか

上記のような観点をふまえて機能の特徴を十分に引き出した後, シナリオの成功要因やリスク要因の分析を行う. 今回は, ユーザ行動の可変性や調整を前提に, 想定する状況下での資源や制約を明確にし, 行動分析の手法を併用してユーザの行動の結果をより予測しやすくする. これにより, 期待しているユーザ行動が本当に生じるかどうかの分析を促進しようとするものである.

2.3. 要求工程への FRAM の導入

要求の記述や分析ではステークホルダの共通理解が重要とされており, FRAM の図式表現がそのような共通理解に有効と考える. 例えば, ユーザストーリーを記述では, 特定タイプのユーザがやりたいと思うことを表す短い動詞句を書くカードを用いたりするが, そのような短い動詞句を FRAM での機能と対応づけ可能と考える. ストーリーの起点や終点, その間のナラティブフローのつながりといったものも FRAM の記述で明確化できると考えた.

例として, Android アプリケーション開発の入門書[6]の例題の一つ, じゃんけんゲームを用いて説明を行う. ユーザがじゃんけんの手を決め, システムがじゃんけんの勝敗を表示するという流れを, 二つの活動・機能からなる FRAM としたものを図 2に示す. ユーザが手を決める機能の出力 O とじゃんけんの勝敗を表示する機能の入力 I が, 側面「ユーザの手」でつながっている.

上記のように FRAM を用いることにより, ユーザストーリーにおいて, ユーザの行動とシステムの動作が連携して起点から終点までつながっていること, またそれがどのように成功となるかといった確認が適切な抽象レベルで行いやすくなると考えた. しかしながら, FRAM はシナリオベース設計のためのものでも, ユーザストーリーマッピングのためのものでもないため, オリジナルのままではそのような記述や分析はできない. そのために, 追加的なルールや補助線の導入を検討する.

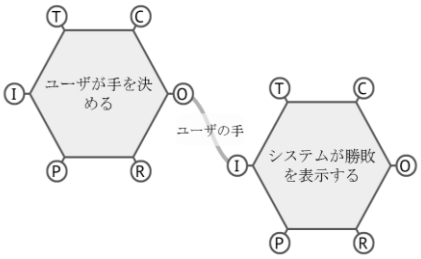


図 2 じゃんけんアプリケーションの初期 FRAM 図

手順 1 ではユーザ側とシステム側の区別を明確化するため, 機能の配色や並べ方のルール, 境界を示す補助線を導入する. ユーザの行動をユーザ側の機能とし, システムの動作をシステム側の機能として, 色で区別したり, 横に揃えたりする. また, ユーザ側とシステム側の境界を示す補助線の追加も行った上で, ユーザストーリーのシナリオが完結するか, 相互作用がインターフェースを介して適切につながっているかといったことを確認する.

手順 2 では, ユーザストーリーに沿ったシナリオの分析に, 行動分析の ABC 分析をとり入れる. 行動(Behavior)そのものだけでなく, 行動のきっかけを与える先行刺激(Antecedent stimulus)と行動した結果与えられる後続刺激(Consequent stimulus)や, 確立操作も分析する. これにより, 対象ユーザ行動が実際に引き起こされるかについて, 先行刺激や後続刺激などの存在やつながりなど, 行動分析の知見に基づく分析が出来ると考えた. そのため, A,B,C に相当する部分を独自の記法でマークする.

手順 3 では, アジャイル型の開発のように, 小さな単位で開発を繰り返すことを念頭に FRAM の図式表現を作成する. MVP から始まり, 開発が終了した機能から随時提供可能となるような開発工程の図式表現を作成する.

手順 4 では, 画面やクラスのシナリオベース設計に向けた記述と分析を行う. スマートフォンのアプリケーションでは画面上のユーザインターフェイスは重要で, Android アプリケーションでは画面ごとに Activity クラスを開発する. これを念頭に画面の切り替えを示す補助線を追加する. その上で, 画面を介した円滑な相互作用のため, ユーザが視認し操作する画面上に何がどうあるべきかといった分析を行う. 画面の仕様を考える際に, 先行刺激, 行動, 後続刺激や確立操作が画面要素とどう対応し実現されるか, といった分析も FRAM 上で行う.

3. シナリオ記述

まず人やシステムの機能・活動に関するシナリオを自然言語で記述する. ソフトウェアシステムにより何が実行されるのか, どのような効果がもたらされるのかなどを記

述する．前述のじゃんけんの例題について，まず，以下図 3のようなユースケースを考えシナリオを記述する．

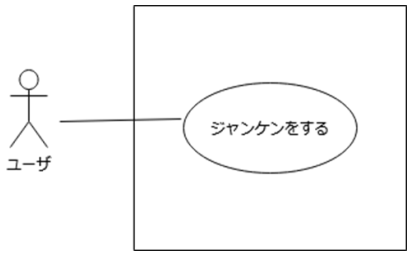


図 3 じゃんけんアプリのユースケース

自然言語のシナリオ記述では以下の下線部のように行動主体を明記する．

レベル：海面レベル
主成功シナリオ(MSS)：
1. ユーザがアプリを起動
2. システムが画面を表示
3. ユーザが手を決める
4. システムが手を決める
5. システムが勝敗を決定

このようなシナリオのユーザストーリーの起点や終点の機能，その間の機能のつながりといったものを FRAM で記述する際に以下のようなルールを用いる．(図 1図 4 参照)

- 動作主体ごとに，機能の色を揃え，水平方向に揃えて並べる．図中，ユーザが動作主体のものが青，システムが動作主体のものが緑である．
- FRAM 用のツール，FMV で作成した FRAM の図に，別途，シナリオ記述での海面，ユーザとシステムの境界に補助線を引く．図では中央の破線の横線が補助線である．

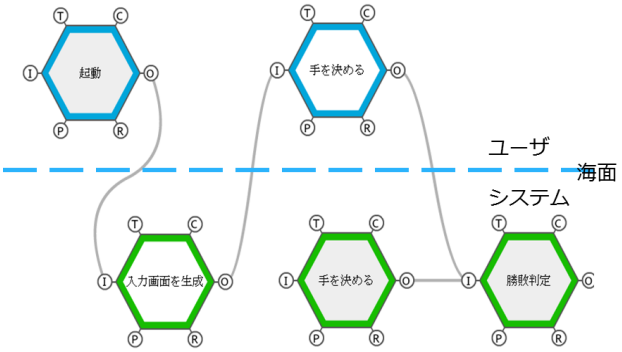


図 4 じゃんけんアプリの FRAM 図の例

そのFRAMの図式表現を図 4に示す．上述の自然言語シナリオの1と3に対応する「起動する」と「手を決める」がユーザ側なので，それらの横並びの配置と色を揃えている．上述の自然言語シナリオの2,4,5に対応するものがシステム側なので，それらの横並びの配置と色を揃えている．さらに，ユーザ側とシステム側との間に補助線を引きその境界の存在を表現している．

4. 行動分析

さらにFRAMのモデリングとABC分析を組み合わせることにより行動分析の知見の活用を試みる．前述のようなFRAM によるモデリングで，ユーザストーリーの起点や終点，その間の機能間のつながりといったものを行動分析学の知見を用いて分析する．行動分析の知見の活用については，一般向けの書籍[9]も出版されており，行動分析の非専門家でもその知見は活用可能と考える．

4.1. ABC 分析

有機体の行動には，刺激によって反射的に誘発されるレスポンデント行動と，そのような誘発刺激によらずに自発されるオペラント行動がある．このような行動のそれぞれのタイプに対し，その変容の二つの原型として，不随意的な行動の変容過程(条件反射) であるレスポンデント条件付けと，随意的な行動の変容過程であるオペラント条件づけがある[5]．

ここではユーザの随意的なオペラント行動を対象とした分析を行う．オペラント条件付けは，有機体が環境などに働きかけるために能動的に自発する行動であるオペラント行動の条件付けであり，あるオペラント行動に刺激を随伴させるとそのオペラント行動の頻度が増加するとき，その行動は強化されたといい，そのような操作をオペラント強化という．そのような操作でオペラント行動の頻度が増加することをオペラント条件付けという．オペラント条件付けがなされたオペラント行動は，自発されたときに強化されないと頻度が減少する．このような操作と事実をオペラント消去と呼ぶ．ユーザストーリーで，ユーザの苦痛や喜び，報酬といったものも考慮する際に，行動が強化されるか消去されるかといった観点を導入する．

提示されることで，あるオペラント行動の自発が強化される強化刺激を正の強化刺激や好子と呼ぶ．除去されることであるオペラント行動の自発が強化される刺激を負の強化刺激や嫌子と呼ぶ．好子の出現や嫌子の消失は直前のオペラント行動の自発頻度を増大させる．

上述のような随意的なオペラント行動の仕組みに着目

して分析を行うにあたって，行動分析学の三項随伴性の分析を用いる．三項随伴性は，きっかけとなる先行条件・弁別刺激(Antecedent stimulus)の下に生じたオペラント行動(Behavior)の後に引き続く強化刺激(Consequent stimulus)が随伴すると，その行動が増えるというという関係を示したものである(図 5参照)．先行条件・弁別刺激，行動，強化刺激の英語表記の頭文字をとってこの分析をABC 分析と呼ぶ．

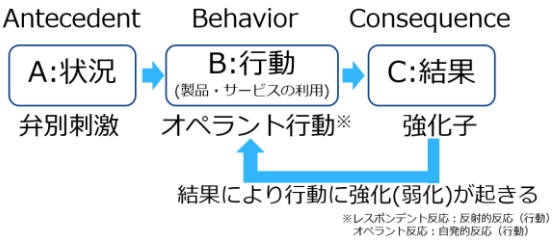


図 5 三項随伴性

行動分析では前述のような行動の随伴性を考えるときは慣習的に随伴性ダイアグラムと呼ばれるものを作成し，行動の直前直後を記述して分析する．ダイアグラム記述により行動の随伴性による影響を整理し，行動の強化・弱化について言葉だけで記述するより分かりやすく捉えることが可能とされている．しかしながら，記法の統一された定義やツールなどはない[10][11]．ここでは，随伴性の分析を，エンジニアリングの観点で効果的に実施することを目指し，FRAM のモデリングと関連付けた随伴性分析を行う．ユーザの行動に加え，弁別刺激や好子や嫌子の機能と関連づけて FRAM のモデリングを行う．機能間のつながりを，三項随伴性の刺激と反応の連鎖の観点で，行動分析学の知見と照らし合わせた分析を行う．

4.2. FRAM 図式での ABC 分析

前述の例題のシナリオに対して，ABC 分析を行った例を以下に示す．全体的にも，部分的にも，の連鎖も行動と結果(強化刺激)の連鎖があることを確認する．また，

その連鎖に入る弁別刺激や確立操作があること，それらの連鎖がこれらの連鎖が強化で終わっていることを確認する．前述の自然言語でのシナリオは以下のように書き換えられる．下線付き太字の部分が追加された部分になる．ユーザ視点では，ユーザが手を決めるという動作の結果がなかったため，ユーザが結果を見るという動作と，そのためのシステムによる結果表示が加えられている．

レベル：海面レベル
主成功シナリオ(MSS)：

1. ユーザがアプリを起動
2. システムが画面を表示
3. ユーザが手を決める
4. システムが手を決める
5. システムが勝敗を決定
6. **<u>システムが画面を表示</u>**
7. **<u>ユーザが結果を見る(</u>**再度じゃんけんなら 2 へ)

図 6にこのような分析の試みを，FRAM の図式表現上で行った例を示す．じゃんけんゲームを起動してプレイするのが標的行動 B だとして，その先行刺激 A や結果の提示 C が何になるかを FRAM 図上で分析するため，図の上側に ABC の範囲に相当する楕円を記入している．この表記は，FRAM のツール FMV には無い機能なので，図式表現をエクスポートして別ツールで書き加えた．そのような全体的な ABC 連鎖を詳細化したレベルの各部分でも，ABC の連鎖がつながっている必要がある．その分析のため B1 がゲームを起動するユーザ行動，B2 がじゃんけんの手を決めるユーザ行動として詳細化している．

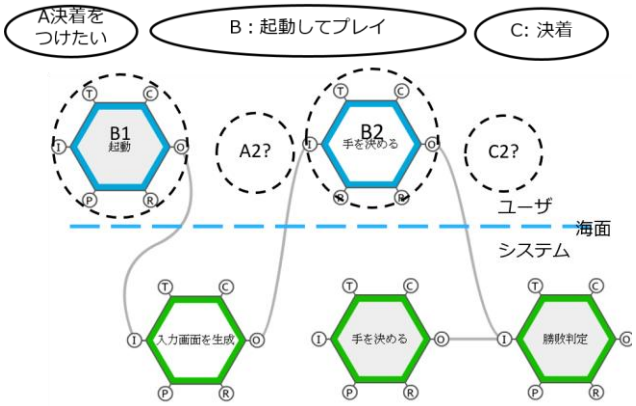


図 6 FRAM 図上での ABC 分析の試み

FRAM 図上で，ABC 分析を進め，行動だけでなく先行刺激や結果の提示を明確化したものが図 7になる．FRAM 図全体で中心となる標的行動 B は「起動してプレイ」であり，その先行刺激 A と結果 C と並べて，実線の楕円で図示している．そのような全体的な ABC 連鎖を分解したレベルでも，ABC の連鎖がつながっている必要があり，図中，点線で囲った A1, B1, C1, A2, B2, C2 として記入している．図中点線の楕円で囲まれた”C1/A2”は，B1 の結果である画面表示 C1 が，B2 に対する先行刺激 A2 となっていることを示している．このように上位レベルの ABC を実現する，下位レベルの ABC の連鎖を分析している．また自然言語のシナリオ 7 に対応する再じゃんけんに対応する要素も記述した．

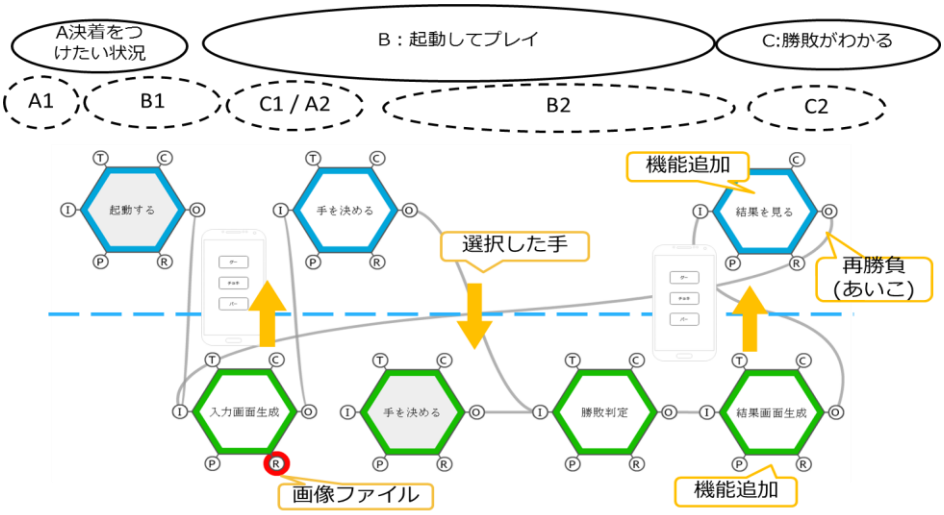


図 7 FRAM 上での ABC 分析の結果

4.3. アンケートによる ABC 分析支援の評価

上記のような ABC 分析をシナリオの自然言語記述に対して行うのと，FRAM の図式表現に対して行うのと，どちらが良いか問うアンケートを，演習の受講学生に対して Google Form を用いて行った．その結果を図 8に示す．FRAM の図式表現の方が良いとの回答が多いが，そうでない回答も約三分の一ある．

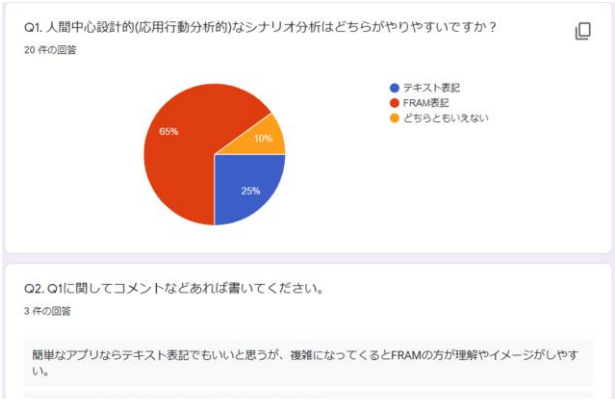


図 8 行動分析を行いやすいシナリオ表現の評価

5. ソフトウェア構造の分析

ここでは，以下の二つの観点でのソフトウェア構造の分析につながるシナリオ分析について論じる．

- アジャイル型の開発のように，機能ごとの小単位で開発を繰り返し，MVP から始まり，開発が終了した機能から随時提供可能となるような開発工程を想定し，FRAM の図式表現を作成する．
- 画面やクラス構造の，上位レベル設計につながる記述と分析を行う．画面を介した機能・活動の円滑な相互作用の実現のためには，ユーザが操作する画面やシステム内のソフトウェアの構造がどうあるべきかについての分析を行う．

5.1. MVP と追加機能の段階的リリースの構造

ここでは，最初のリリースでは4.2節で紹介したようなシナリオを実現するものを，二回目のリリースではよりゲーム性を高めるようなものをリリースするという，段階的リリースのための分析について論じる．ゲーム性を高める二回目のリリースでは，システム側のじゃんけんの手の決定にこれまでの勝負の履歴を反映させるシナリオを考える．

リリース 2
主成功シナリオ(MSS)：

1. ユーザがアプリを起動
2. システムが画面を表示
3. ユーザが手を決める
4. システムが履歴を読み出す
5. システムが履歴を使い手を決める
6. システムが勝敗を決定
7. システムが履歴を保存
8. システムが画面を表示

9. ユーザが結果を見る(再度じゃんけんなら 2 へ)

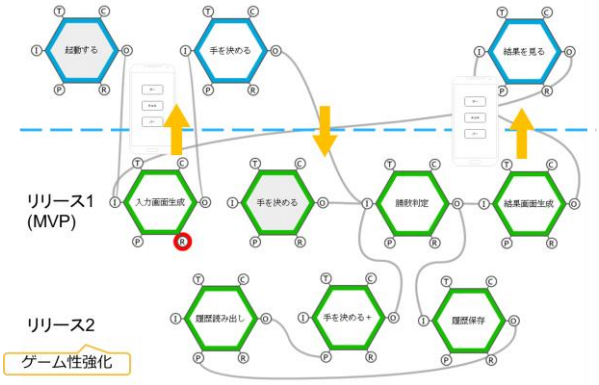


図 9 段階的リリースの FRAM 図式表現

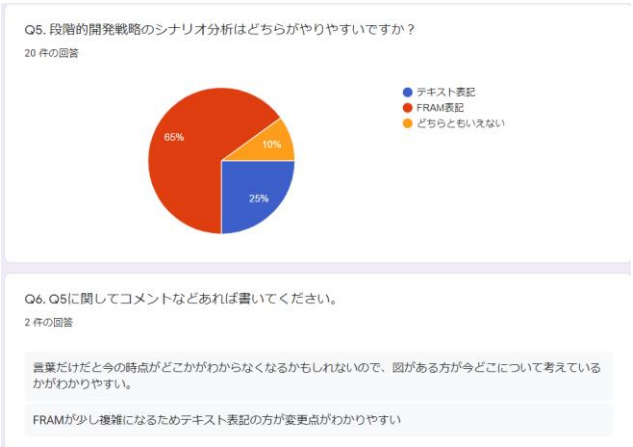


図 10 段階的リリース構造の分析支援の評価

このような段階的リリースに対応する FRAM の図式表現を図 9に示す. 破線より下のシステム側で, 機能の連鎖をリリースの回数に対応させた並びにする. 今回の例では二回リリースがあるので, 一回目のリリースの MVP の機能の並びと, ゲーム性を高めた二回目のリリースで, 並びを変える. 後続のリリースでは, これまでのリリースの機能を含むことを前提とするが, 機能の置き換えなどがある場合はそれがわかるような命名規則を使い, FRAM のテキスト表現に書いておくなどする.

5.2. 段階的リリース構造支援のアンケート評価

上記のような段階的リリースの分析を, シナリオの自然言語記述に対して行うのと, FRAM の図式表現に対して行うのと, どちらが良いか問うアンケートを, 演習受講生に対し Google Form を用いて行った. その結果を図 10 に示す. そうでない回答も約三分の一あり絶対ではないが, FRAM の図式表現の方が良いとの回答が多かった.

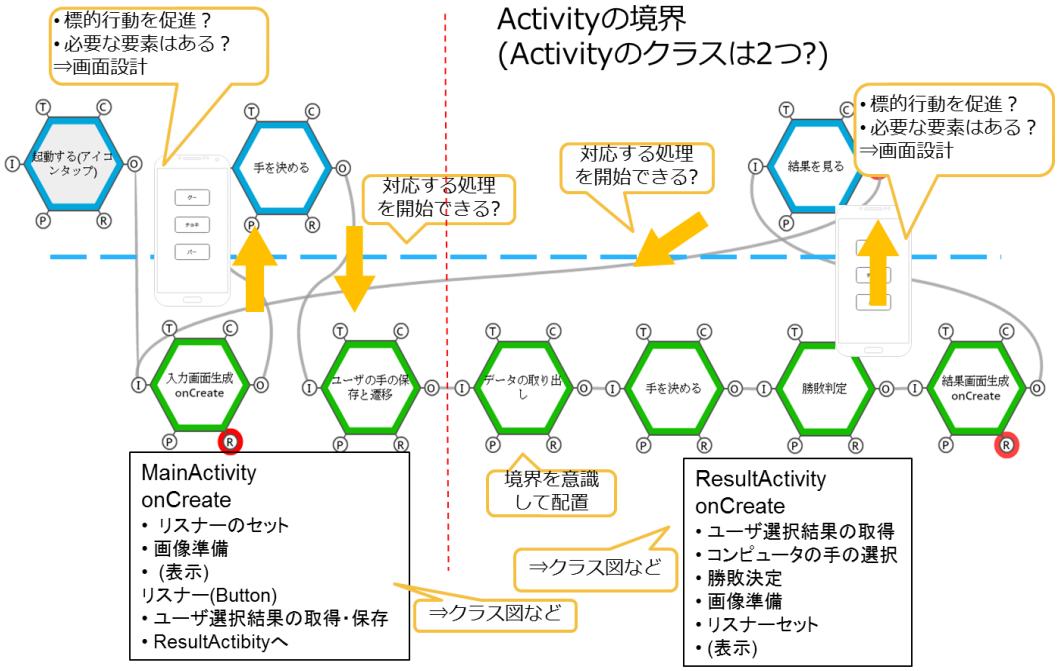


図 11 画面の仕様やクラスの分析着手時の FRAM 図

5.3. シナリオベースの画面仕様やクラスの分析

画面仕様やクラス構造などのシナリオベース設計に向けた分析を行う. スマートフォンのアプリケーションでは画面を介した機能・活動の円滑な相互作用を分析することは重要である. Android アプリケーションでは, 画面単位で Activity のクラスをつくる必要があるといった点も考慮しながら, ユーザが操作する画面やシステム内のソフトウェアの構造がどうあるべきかについての分析を行う.

図 11に, 図 7相当の FRAM の図式表現をベースに, 画面仕様や Activity のクラスの分析を行った図を示す. 縦の点線が Activity の境で, この線をまたぐデータの移動には, 別途明示的な処理が必要になる. また, ユーザ側とシステム側の境界を示す点線をまたぐ相互作用について, 下向きのものはユーザがシステムを起動するもの, 上向きのものはユーザに画面を提示して行動を促すものとして考える. このような分析を行うことで, より実装に近いレベルの機能を抽出することが可能になる.

5.4. シナリオベースの画面仕様やクラスの分析の評価

上述の, 画面仕様やクラスの分析での FRAM の図式表現が支援として役立ったかのアンケート結果を図 12 に示す. そうでない回答も約五分の一あり絶対ではないが FRAM の図式表現の方が良いとの回答が多かった.

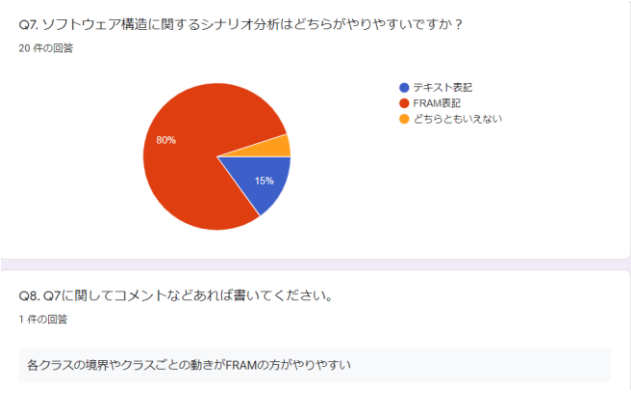


図 12 画面仕様やクラス分析支援の評価

6. おわりに

本稿ではソフトウェア, 特に UI が重要なスマートフォンのアプリケーション開発上流工程で, シナリオ分析に機能共鳴分析法 FRAM の図式表現を用いることを論じた. FRAM にはテキスト表現だけでなく図式表現があり, 文字情報の読み込みを前提としない, 視覚的で直観的な理解の促進も含めたシナリオ分析支援が可能と考えた.

提案手順では, まず, シナリオから FRAM の図式表現を作成する. 次に, 生起を想定するユーザ行動の明確化と, その生起の仕組みの分析のため, 行動分析の ABC 分析を併用する. さらに, FRAM の図式表現上で段階的リリースに対応するソフトウェア構造を検討する. スマートフォンのアプリケーションでは, 画面上のユーザインターフェイスも重要な役割を持ち, Android アプリケーションでは画面ごとに Activity クラスの開発も行う必要がある. このような点を踏まえ, 画面を介した機能の相互作用も FRAM の図式表現で記述, 分析することを論じた.

著者が担当のスマートフォン(Android)アプリケーションの設計演習(学部)で, 自然言語によるものと, 提案した FRAM の図式表現による記述と分析を比べるアンケートを行った. その結果, 約三分の二程度の受講者が FRAM の図式表現のものが良いとする回答結果を得た.

モデル構築や分析, シミュレーションなどを支援する新しいツールも出ており[12][13], ソフトウェア開発も含め FRAM の活用法について引続き研究を行う予定である.

参考文献

[1] E. Hollnagel (小松原明哲監訳). 社会技術システムの安全分析-FRAM ガイドブック, 海文堂出版, 2013
[2] E. Hollnagel, Personal communication, 2018
[3] J. M. Carrol, Making Use --- Scenario-Based Design of Human-Computer Interactions, MIT press, 2000
[4] Jeff Patton (川口恭伸監訳), ユーザーストーリーマッピング, オライリー, 2015
[5] 佐藤方哉, 行動理論への招待, 大修館書, 1976
[6] 金田浩明, はじめての Android プログラミング 第 4 版, SB クリエイティブ, 2019
[7] FMV FRAM Model Visualizer, https://functionalresonance.com/FMV/index.html
[8] 野本秀樹, 道浦康貴, 石濱直樹, 片平真史, FRAM-機能共鳴分析手法による成功学に基づく安全工学, SEC journal, 14(1), pp.42-49, Aug. 2018
[9] 島宗理, 使える行動分析学, ちくま新書, 2014
[10] M. A. Mattaini, Contingency Diagrams as Teaching Tools, The Behavior Analyst, 18(1), pp.93-98, 1995
[11] S. Toogood, Using contingency diagrams in the functional assessment of challenging behavior, Int'l J. of Positive Behavioural Support, 2-1, pp.3(10), 2012
[12] R. Patriarca, G. D. Gravio, F. Costantino, myFRAM : an open tool to support FRAM. 12th FRAMily meeting and workshop, pp.11-13, 2018.
[13] FRAM Model Interpreter,https://functionalresonance.com/the-fram-model-interpreter.html

ゴール指向要求分析とシステム安全分析を利用した AI システム品質の個別ガイドライン導出方法の提案

Individual Guideline Derivation Method in AI System Quality Assessment by use of Goal-Oriented Requirements Analysis and System Safety Analysis

相津 一寛
パナソニック株式会社
aizu.kazuhiro@jp.panasonic.com

石川 冬樹
国立情報学研究所
f-ishikawa@nii.ac.jp

小宮山 英明
コニカミノルタ株式会社
hideaki.komiyama@konicaminolta.com

栗田 太郎
ソニー株式会社
taro.kurita@sony.com

柳原 靖司
ブラザー工業株式会社
yasushi.yanagihara@brother.co.jp

徳本 晋
富士通株式会社
tokumoto.susumu@fujitsu.com

要旨

本論文では，AI システムの品質を保証する手段として，IGDM-AIQA 法(Individual Guideline Derivation Method in AI system Quality Assessment)を提案する．現在，AI システムは多くの分野で開発，運用されているにも関わらず，その特性から品質保証の方法が確立されていない．このような問題に対し，AI 開発の知見を集約してガイドライン化することが議論されているが，これらガイドラインは有識者向けで抽象度の高い内容となっており，AI の知見が必ずしも十分でない品質保証担当者が効果的に活用できるとはいえない．IGDM-AIQA 法を用いることで，対象システムの要件に基づいて品質アセスメントに必要な観点を導出し，品質保証の現場担当者が精度良く品質アセスメントを行える．

1. はじめに

機械学習技術の著しい発展により様々な産業分野で AI システムが開発，利用され始めているが，AI システムが内包する特有の性質から従来型の品質保証アプローチが通用しない課題が議論されている．このような課題に対して，AI システムに特化した品質保証のあり方が研究され，様々な AI システムの品質保証ガイドラインが発行されている．しかし，機械学習技術の応用分野は幅広く，既存の品質保証ガイドラインでは AI システムに共通する上位水準の知見，又は個別産業で共通する知見を提示しているのが実際である．これらガイドラインの記述は抽象度が高く，機械学習技術に精通していない品質保証担当者では記述の解釈が難解である．

本研究では，ゴール指向要求分析手法のひとつである AGORA[1](Attributed Goal-Oriented Requirements Analysis method)と，システム安全分析手法のひとつである FRAM[2] (Functional Resonance Analysis Method)を応用して，AI システムの要求から品質保証の点で確認すべき項目（サブガイドライン）を抽出する IGDM-AIQA 法を考案し，実用性を評価した．仮想のクレジットカード与信審査システム（以降，FinTech 与信判定システムと略す）を例とした第三者による実験では，機械学習技術に詳しくない技術者であってもサブガイドラインを活用して有効な指摘ができることが分かり，要求工学やシステム安全性向上の知見に基づいてサブガイドラインを導出する意義を確認した．

以下，本論文の構成を述べる．まず，2 章では現状分析と課題提起を行う．次に 3 章では関連技術について言及し，4 章，5 章で夫々，解決策の提案と評価結果を示す．6 章で評価結果に対する考察を行い，最後に 7 章で成果と将来への発展で結ぶ．

2. 解決すべき課題

2.1. 現状分析

機械学習技術を応用したシステム開発では，外界から収集された学習データに基づきモデルを生成しながら開発を進めて行く帰納的手法を採用しているため，従来の演繹的手法に基づくシステム開発及び品質保証アプローチを適用できない．このような状況に対応するため，開発・運用フェーズにおいて AI システムの品質作り込みや品質評価の指針を与えるものとして，汎用的な AI システム品質のためのガイドライン（以降，汎用ガイドラインと略す）が発行されている．

2.2. 課題提起

既存の汎用ガイドラインは，機械学習技術に対する知識が一定以上ある開発者を想定しており，多岐にわたる AI システムの応用分野における共通事項をまとめているため内容が抽象的で，機械学習技術の知見が十分でない品質保証担当者が活用することが難しい．例えば，機械学習品質マネジメントガイドライン[3]では，「このガイドラインは，できるだけ広範な応用に適用できるような汎用的な記述内容になっており，利用者が具体的な応用に即して，記述内容を取捨選択・具体化して用いることを想定している．実際の利用場面においては，各利用者の状況に応じて，応用分野毎にこのガイドラインの内容を具体化した個別ガイドラインを作ることを想定している．」という記述が本文に記載されている．また，池村・佐藤・松本(2021)は AI 品質マネジメントガイドラインの具体化に関する研究[4]の中で，AI 技術の実務経験有無の違いによりガイドライン解釈の精度に相違がでることを示唆しており， AI 技術初学者がガイドラインを読み解く上で有識者の補助が必要であると述べている．

実際の企業の開発現場に AI 品質の汎用ガイドラインを広く普及させる上では，ガイドラインの具体化と展開のための仕組み作りが必要であると考える．

3. 関連技術の説明

我々の研究の技術的拠り所として用いている，AI システム品質に関する汎用ガイドライン，AGORA 及び FRAM について説明する．関連技術として AGORA と FRAM に着眼した理由であるが，一般に AI システムでは要件の間でトレードオフが発生することが多いため，システムの要求から要件を獲得する過程を俯瞰的に可視化しながら分析できる点で AGORA，AI システムの運用時にプロジェクト関係者だけでなくエンドユーザや社会（コミュニティ）といった多様なステークホルダが関連するため，複雑な機能連関構造を可視化しながら分析できる点で FRAM が夫々適していると考えた．

3.1. AI システム品質に関する汎用ガイドライン

機械学習技術を利用したシステムやプロダクトの品質保証に対する共通指針を与えるものとして，機械学習品質マネジメントガイドライン，AI プロダクト品質保証ガイドライン[5]が発行されている．企業・大学等の有識者が知見に基づいて取りまとめたものであり，製品やサービスの開発者，利用者らが参照することを想定して記載されている．

3.2. AGORA

ゴール指向要求分析手法のひとつで，AND-OR ツリーグラフに属性値を付与した上で，主要求から下流に向かって副要求を展開しながら要件を導出する．リーフの各要件について，ステークホルダの満足度行列を調べることで主要求に対するゴール適合度や意見の対立度を計算することができる点が特徴である．図 1 に AGORA のモデル例を示す．

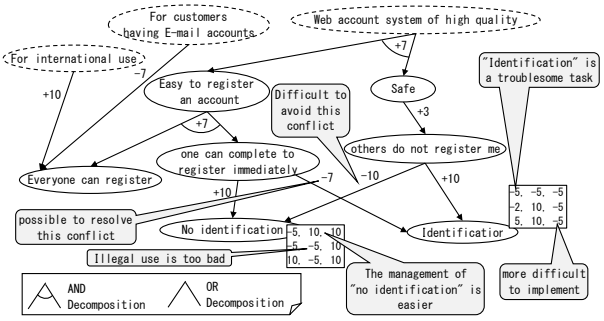


図 1 AGORA による要求分析例

なお，ゴール指向要求工学（GORE: Goal-Oriented RE）の中には先行研究として KAOS[6] (Knowledge Acquisition in Automated Specification), i*[7], NFR Framework[8] (Non-Functional Requirements Framework) 等が存在する．何れもツリー図やネットワーク図を利用しながらゴールを実現するためのサブ要求や要件を分解しながら展開することで，因果関係・AND/OR 関係等を論理的に表現する点が共通している．

3.3. FRAM

システム安全性向上におけるモデリング手法のひとつである. 複数の機能がインタラクションする構造をモデルベースで分析することで, システムの長所や短所を特定できる. システムが正常に働くパターンを増強することを目的として利用する. 図 2 に FRAM のモデル例を示す.

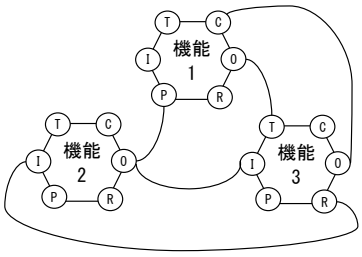


図 2 FRAM による分析例

なお, システム安全分析手法の中には先行研究として STAMP/STPA[9] (Systems-Theoretic Accident Model and Processes/System-Theoretic Process Analysis), MBSE[10] (Model-Based Systems Engineering)等が存在する. 前者はシステム間の連携やシステムと人の関わりにおいて構成要素の相互作用からハザードが発現するプロセスをモデル分析することにより, 創発特性から生じるシステム事故の防止に役立てる手法, 後者は高い抽象度でシステムをモデリングし, 俯瞰的に検証することで全体最適化の実現を支援する手法である. MBSE のモデリング手段として SysML(Systems Modeling Language)を使うことができるが, SysML で記述したモデルを STAMP/STPA で解析した研究報告[11]も存在する.

4. 解決策の提案

4.1. 課題の解決方針

汎用ガイドライン[3]では AI システムの品質観点を, ユーザに対する「利用時の品質」, システムの構成全体に求められる「外部品質」, 構成要素の固有特性として定義される「内部品質」として捉えている. 本研究では, 当該ガイドラインにおける 3 つの品質観点を考慮しながら, 個別 AI システムに対する品質保証上の着眼点をビジネスゴールに適合するように導出する枠組み(IGDM-AIQA 法)を提案する(図 3). また, 現場の品質保証業務に対する IGDM-AIQA 法の有効性を評価するため, FinTech 与信判定システム(図 4)に適用した結果を示す.

[IGDM-AIQA 法の特徴]

・ ゴール指向要求分析手法(AGORA)により利用時の品質と外部品質を考慮しながら AI システムを分析することで, 該当システムに求められる要件群を目的指向で導出する.

・ AI システムの有効性と公平性を定量的に解析する上で, 与えられたデータセットに対する機械学習モデルのシミュレーションを行いながらデータ分析することで, 当該システムの内部品質に求められる特性を把握する.

・ AI システムの主要求に関わるステークホルダの機能連関構造(FinTech 与信判定システムでは公平性に配慮した社会受容性の構造)を明らかにする上で, 各機能のインタラクションを FRAM により分析し(図 5), ステークホルダの重要度を特定する.

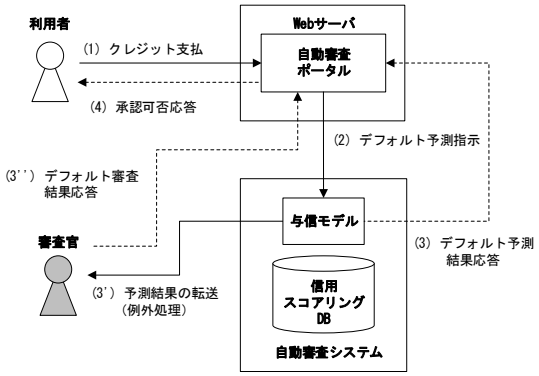


図 4 FinTech 与信判定システム

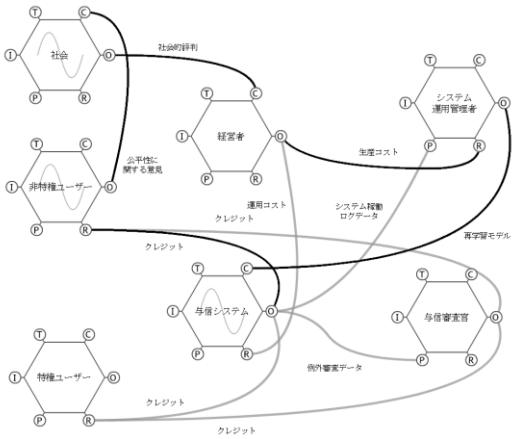


図 5 ステークホルダの機能連関分析 (FRAM 適用例)

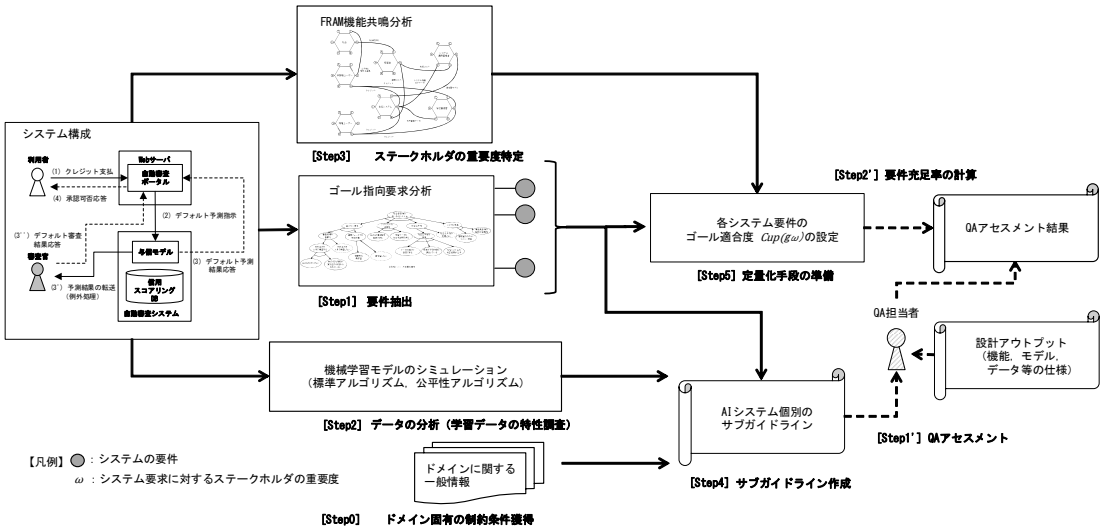


図 3 IGDM-AIQA 法（AI システム品質のサブガイドライン導出の枠組み）

4.2. IGDM-AIQA 法を用いたサブガイドラインの導出

IGDM-AIQA 法は，AGORA，FRAM といったシステム開発で利用される既知の分析手法を活用しながら個別 AI システムに合わせて汎用ガイドラインを解釈するための枠組みである．品質保証業務に携わる QA 担当者は，表 1 に示す手続きに則って進めることで実用的なサブガイドラインを導出することができる．

[ゴール適合度の算出手順]

佐藤・石川・猪原(2011)は，ゴール指向要求分析手法（AGORA）によって導出した各要件に対して，対象システムのゴール適合度（貢献度）を定量化する計算手法を示している[17]．我々の研究では，この定量化方法を参考に，FinTech 与信判定システムに登場するステークホルダ（図 6 凡例参照）のうち主要なステークホルダ（Primary Stakeholder）である，UU: Unprivileged User（非特権ユーザ），OW: Owner（経営者），OM: Operations Manager（システム運用管理者）のゴール適合度を式(1)に基づき計算することにした．

表 1 IGDM-AIQA 法を用いたサブガイドラインの導出手順

手順	目的	□入力情報，■出力情報	手続き
STEP0	AI システムの開発に付帯する制約の明確化	□ システム仕様・設計情報，ドメインの関連資料[1] ■ ドメイン固有の制約情報	ドメイン固有の情報から，目的システムの開発や運用に関わる制約情報（機械学習，運用，規制等の知見）を獲得する．
STEP1	AI システムの品質観点を定義する際に参照する要件群の導出	□ STEP0 の出力，汎用ガイドライン ■ 目的システムの要件群（品質観点の定義に適した粒度）	ゴール指向要求分析手法 AGORA を用いて，目的システムの主要求（ゴール）をツリー状にサブゴールへと展開しながら要件群を導出する．品質観点として，汎用ガイドラインに掲載されている内容（リスク回避性，AI パフォーマンス，公平性[2]等）を考慮する．
STEP2	機械学習コンポーネントの学習データに対する推論特性の調査	□ 目的システムが前提とする機械学習のデータセット ■ 機械学習アルゴリズムの推論特性，データセットの被覆性・均一性に関する情報	目的システムに搭載されている機械学習コンポーネントの学習データに対する推論特性をシミュレータ等で解きながら把握する．（機械学習の汎用アルゴリズムに対して Colaboratory(Google)，公平性アルゴリズムに対して AI Fairness 360(IBM)[15]，Fairlearn(Microsoft)[16]等を利用する）

[1] FinTech 与信判定システムの場合は，改正割賦販売法[12]，日本銀行のワークショップ報告書[13]，研究事例[14]等を参照した．

[2] 対象システムよっては公平性に対する要求が小さいため，適宜判断して除外する．（例：株価の予測，交通標識の識別）

STEP3	AI システムのゴールに強い影響を及ぼすステークホルダの特定	□ STEP0, 1, 2 の出力 ■ Primary Stakeholder のリストとゴール(主要求)の貢献度に応じた重みω	システムの運用時に関わるステークホルダのうち，目的システムのゴール適合度に影響する Primary Stakeholder を FRAM 分析[3]により特定する. ステークホルダの機能連関構造を参考にしながら，システムの主要求に対する貢献度に応じて各 Primary Stakeholder の重みωを決める.
STEP4	QA アセスメントで使用するサブガイドラインの導出	□ STEP1, 2, 3 の出力 ■ サブガイドライン	システムの運用時に目的システムの機能が正しく発現されるための品質観点を，要件毎に汎用ガイドライン及び学習データの推論特性に基づいて検討し，ステークホルダの重要度を加味した上でサブガイドラインを導出する.
STEP5	AI システムのゴール適合度 $Cup(g_\omega)$の計算	□ STEP1, 3 の出力 ■ ゴール適合度	サブガイドラインを用いた QA アセスメントタスクの達成度を計測するため，図 6 に示す手順に基づき，目的システムのゴール適合度$Cup(g_\omega)$(式 1)を要件毎に求める.

$$Cup(g_\omega) \stackrel{\text{def}}{=} \frac{\sum_{s\in Stakeholder, p\in Primary\ stakeholder} \omega \cdot m(g)_{s,p}}{|\ Stakeholder\ | \cdot |\ Primary\ stakeholder\ |}. \quad (1)\ [17]$$

評価の視点(被評価者の立場)

評価者(役割) ＼ 要件名: 与信システムの判定結果が公平	PU	UU	OW	OM	DV	役割毎の評価基準
PU	5	8	8	7	6	社会における機会均等性は理解できる
UU	8	10	9	8	6	より良い社会に向けて弱者に対する配慮が欲しい
OW	5	8	8	7	7	AI システムの性能と公平性はバランスが必要
OM	5	8	8	7	6	運用時に継続的にモデルを改善する必要がある
DV	4	7	7	6	6	技術によって不公平性を緩和することは難しい

STEP5-1
満足度行列(左図)に各評価者の役割で，被評価者の視点に着目して-10～+10 の素点(要件に対する重要度)を入力する.

STEP5-2
$Cup(g_\omega)$を計算する. [左図例 7]

分子:左図のグレー部分の和. [左図例 105]
※役割毎に重み ω を付与
(ω の値は STEP3 参照)
分母:左図の実線部と破線部に含まれる要素の集合濃度の積の平方根. [左図例 15]

[補足] 本研究では，研究員3名が夫々作成した満足度行列から$Cup(g_\omega)$を求め，これらを平均した.

【凡例】PU: Privileged User(特権ユーザ*1)，UU: Unprivileged User(非特権ユーザ*1)，OW: Owner(経営者)，OM: Operations Manager(システム運用管理者)，DV: Developer(開発者)
(*1) FinTech 与信判定システムでは PU が男性，UU が女性である.(特定の年代で，女性が与信額で不利な扱いを受ける場合がある)

図 6 $Cup(g_\omega)$導出のための満足度行列

4.3. 仮説と研究設問

本研究の仮説と，研究設問 RQ を示す.

[仮説]

IGDM-AIQA 法から導出された AI システム品質評価のためのサブガイドラインを用いれば，ガイドラインがない場合，及び汎用ガイドラインを参照した場合に比べ，社会受容性を含むビジネスゴールを持った AI システムの品質保証のアセスメント精度が向上する.

[研究設問]

RQ1: 機械学習技術に詳しくない技術者がサブガイドラインを参照すると，ガイドラインがない場合，及び汎用ガイドラインを参照した場合に比べ，システム要件に関わる欠陥指摘の精度が改善する.

[3] FinTech 与信判定システムの例では，図 6 の凡例に示すステークホルダのうち，図 5 の機能ループ(太線)を増強する重要なステークホルダとして，非特権ユーザ $_{UU}$，経営者 $_{OW}$，システム運用管理者 $_{OM}$を選定した. なお，AGORA の$Cup(g_\omega)$の計算値に影響を与えるのは UU, OW, OM であるが，各ステークホルダの素点を同列で扱うのでなく，FRAM 機能連関構造内の「社会(六角形のオブジェクト)」に対する影響の仕方(強弱)を加味することにした. 社会に直接作用する非特権ユーザ UU，社会からの出力を受けて指令を出す経営者 OW，間接的に機能ループの強化に作用するシステム運用管理者 OM という解釈を行い，3 名の研究員で合意して ω の重みを決定した. 要件のゴール適合度を$Cup(g_\omega)$と再定義することで，ステークホルダの機能連関構造から生じる影響を要件の重みに反映させた(要件に対する重みは$\omega_{UU} = 1.0$, $\omega_{OW} = 0.9$, $\omega_{OM} = 0.8$を付与した).

RQ2[4]: 機械学習技術に詳しい技術者がサブガイドラインを使っても，ガイドラインがない場合，及び汎用ガイドラインを参照した場合に比べ，システム要件に関わる欠陥指摘の精度は改善しない．

5. 解決策の評価

5.1. 評価方法

仮説の検証は，FinTech 与信判定システムの概略設計資料（設計アウトプット）に含まれる欠陥事項をアセスメントするような枠組み（図 7）を用いた．独立した 2 つの被験者グループを用意することで RQ の定量的評価と，属性毎の定性的評価を行えるようにした． なお，5.2 節において前者は$\sum_i A_i \cdot Cup(g_{i\omega})$，後者は$\overline{A_i}$及び$\sum_i A_i$に着目して被験者のアセスメント精度を分析した．（$A_i$は要件 i に対する *Assessment accuracy* を意味する）

■ 実験意図の説明	⇒	■ 検証対象の提示	⇒	■ 実験の試行	⇒	■ 実験結果の評価
被験者に対し，システム要求（主要求と副要求）及びシステム構成図を提示し，（本実験の）品質アセスメントで問われている期待結果を理解させる．		設計チームが作成した不完全なシステム基本設計情報（表 2）を提示する．当該情報には，モデル，データ，その他の各区分で，設計意図が詳細に記述されている．		被験者グループ*2 I群とII群は，ガイドラインなし，汎用ガイドライン参照，サブガイドライン参照の各条件で，基本設計情報に含まれる欠陥を指摘する． ※30 分間で最大 10 個		被験者から提出してもらった指摘事項を，システムの要件（表 3）と照合しながら3段階で採点する*3. [A_i: *Assessment Accuracy*] 1 点: 指摘と理由が適合 0.5 点: 指摘はできているが理由が不明瞭 0 点: 指摘は要件の範疇外

(*2) 被験者グループ I群：機械学習技術に詳しくない技術者，II群：機械学習技術に詳しい技術者
(*3) AGORA を使って導出された要件群に対する適合性で採点する．

図 7 IGDM-AIQA 法の効果測定の枠組み

表 2 システム基本設計情報（一部抜粋）

区分	項目	説明
モデル	アルゴリズム	分類問題を解くためのアルゴリズムとして，表現力が高く，正答率を高めやすいDNN（Deep Neural Network）を選定した．モデルのハイパーパラメータとして，BATCH_SIZE：25, EPOCH：20とした．

表 3 要件と対応するサブガイドライン

主要求：省人化に寄与する性能だけでなく，公平性に配慮した社会受容性の高い与信システム				
#i	要件	副要求	$Cup(g_{i\omega})$	サブガイドライン（要約）
1	ML*4 の計算過程を解釈できる	アウトプットの透明性	5.4	・モデルのアルゴリズムは，説明性の高いアルゴリズムを使用しているか．（※国内の与信判定システムでは，一般的に決定木，XGBoost，LightGBM 等が好まれる）
2	ML の汎化性能が実社会の状況から外れていない	省人化に寄与	4.7	・モデルのアルゴリズムに含まれる汎化のために採用している制約によって，少数の重要なデータが無視されていないか．
3	データの偏りが受容できる	社会公平性	7.7	・学習データの内容の分布が，偏っていないか． －年齢データの比率 －最終学歴データの比率 －学習データ内のデフォルトの比率
4	標本が予測対象と適合している	社会公平性	6.5	・学習データに，クレジットカードのデフォルト予測で取り扱うすべての審査対象者のデータが網羅的に含まれているか．
5	学習データの加工を説明できる	アウトプットの透明性	3.5	・正例（デフォルト），負例（非デフォルト）の不均衡を解消するため，SMOTE 等の標準アルゴリズムにより近接データの内挿を行って，データを増やしているか．

(*4) ML は Machine Learning（機械学習）の略

[4] 本研究の構想段階において，IGDM-AIQA 法から導出したサブガイドラインは，機械学習技術に詳しくないが演繹的手法に基づくシステム開発には精通している技術者に対してのみ有効であると仮定した．一方，機械学習技術に詳しい技術者に対してはサブガイドラインの有効性が低く，既存の汎用ガイドラインを参照することで品質の作り込み活動を行えるものと考えた．

6	ML の誤りが少ない	省人化に寄与	4.0	・機械学習コンポーネントの推論結果における正答率, F1 値, AUC が十分であるか.
7	与信システムの判定結果が公平	社会公平性	7.1	・学習データでデフォルトになっているデータは特定の性別に偏っていないか. ・システムが出力する与信の判定結果が, 特定の属性に対して不公平な結果になっていないか. ・少数の非特権ユーザに対して不利な判定結果になるような, 機械学習アルゴリズムのバイアスに対する補正が実施されているか.
8	機動的なモデルの再学習	省人化に寄与	2.9	・機械学習コンポーネントの再学習に要する時間は, 運用フェーズにおいて許容できる時間内であるか.
9	管轄省庁のガイドラインに準拠	リスクの低減	2.9	・収入が低い世代の人に対してのクレジット額が高くなっていないか. (※年代毎の与信のデータ分布)
10	事故発生時の解析の容易性	リスクの低減	3.0	・機械学習モデルの作成時に用いた学習データ・検証データ・モデル学習履歴等を適宜参照できるように記録されているか.

5.2. 評価結果

異なる業種(製造, 情報・通信, 金融)に属する計13社の被験者(機械学習技術に詳しくない技術者 I群: 25 名, 機械学習技術に詳しい技術者 II群: 13 名)に対して, 仮想 FinTech 与信判定システムのシステム概略設計資料をアセスメントしてもらい, 当該資料に含まれる欠陥事項を日本語で指摘してもらった. (回答集計率 95%)

[RQ に関する評価結果]

システムの要求を期待結果とした前記システム概略設計資料のゴール適合度(欠陥指摘精度)を検証するため, 各条件(条件 a: ガイドラインなし, 条件 b: 汎用ガイドライン参照, 条件 c: サブガイドライン参照)における被験者の回答結果(自然言語)を$\sum_i A_i \cdot Cup(g_{i\omega})$で定量化した(表 4, 5). 4.3 節に示す RQ を検証する上で, 被験者回答データが正規分布に従っていると仮定した上で, 条件 a と条件 c, 及び条件 b と条件 c の各母集団に有意な差があるか否かを t 検定(有意水準 5%)によって検証した. t 検定の準備として, F 検定によりサンプルの等分散性を確認したところ, I群については等分散性がなく, II群については等分散性があったので, 夫々Welch の t 検定, Student の t 検定を用いた. I群に関する t 検定の統計量は条件 a と条件 c が 3.01×10^{-8}(＜0.05), 条件 b と条件 c が 6.76×10^{-6}(＜0.05)であり, 両者とも有意な差があることが分かったので RQ1 の妥当性が確認できた. II群に関する t 検定の統計量は, 条件 a と条件 c が 8.27×10^{-5}(＜0.05), 条件 b と条件 c が 1.33×10^{-3}(＜0.05)であり, 両者とも有意な差があることが分かったので RQ2 の妥当性が確認できなかった. 即ち, 保有する機械学習技術の知見に依らずサブガイドラインがあれば, 個別 AI システムの設計アウトプットの欠陥指摘精度が向上するので, 条件付きで仮説を検証できたといえる.

表 4 被験者回答精度(I群 $\sum_i A_i \cdot Cup(g_{i\omega})$)

I群	条件 a	条件 b	条件 c
平均	9.0	13.2	26.1
分散	17.8	37.9	102.9
要件充足率	18.9%	27.6%	54.7%
サンプル数	25	25	23

(補足)一部の被験者から回答データを回収できなかった

表 5 被験者回答精度(II群 $\sum_i A_i \cdot Cup(g_{i\omega})$)

II群	条件 a	条件 b	条件 c
平均	16.8	19.7	30.6
分散	32.3	26.4	73.9
要件充足率	35.2%	41.3%	64.2%
サンプル数	13	12	12

(補足)一部の被験者から回答データを回収できなかった

[定性的な評価結果]

設計資料に含まれる欠陥事項に対する被験者の回答傾向を把握する上で, 回答結果(自然言語)の要件適合性($\overline{A_i}$ 及び$\sum_i A_i$)で可視化した.

<u>要件毎の被験者回答精度</u>

被験者の回答精度をシステムのゴール適合度(主要求への適合性)という全体視点で見ると, FinTech 与信システムの基本設計情報に含まれる欠陥事項の回答指摘精度はサブガイドラインを参照した場合に有意に改善することが分かったが, 要件毎に分解して見た場合には, 各要件の改善幅に差があった(図 8, 9). 表 6 に定性的に分析した結果を示す.

品質保証におけるサブガイドラインの効果

所属企業での役割が QA 担当者(N=6)である被験者の回答結果を抽出して可視化したものを示す(図 10). QA 担当者の場合, サブガイドラインを参照しながら欠陥指摘を行うと(条件 c), 参照しない場合(条件 a, b)に比べ, 夫々, 4.2 倍(1.8pt→7.5pt), 2.5 倍(3.0pt→7.5pt)の改善効果が認められた. この結果は, 機械学習技術の有識者(II群)の対応する改善効果(1.6～1.7 倍)と比べても顕著である.

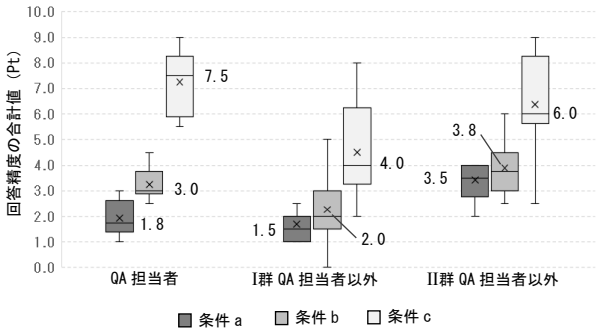


図 10 役割に着目した被験者回答精度$\sum_i A_i$

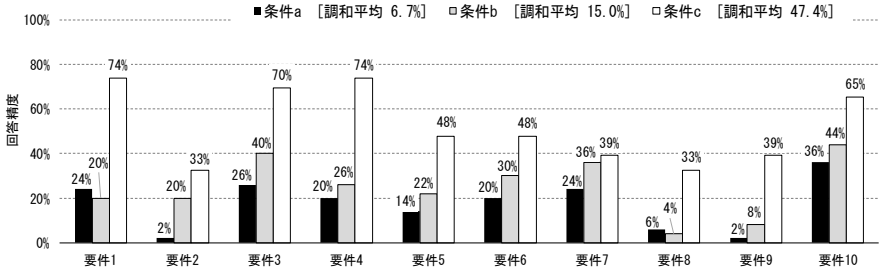


図 8 要件毎の被験者回答精度(I群 $\overline{A_i}$)

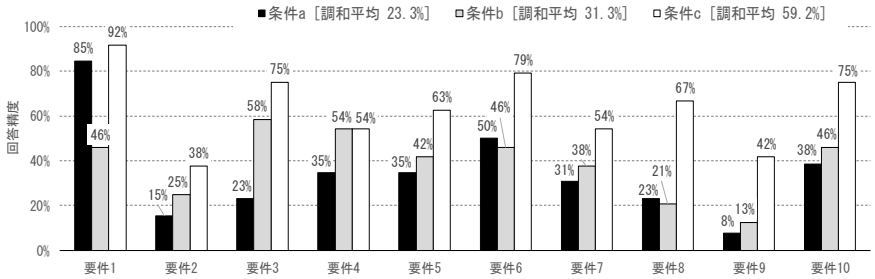


図 9 要件毎の被験者回答精度(II群 $\overline{A_i}$)

表 6 要件別被験者回答精度の定性的評価

I群 $\overline{A_i}$	推定要因
改善幅が大きい要件:要件 1, 4 (条件 a と c の差 50pt, 54pt)	サブガイドラインの対応箇所では機械学習アルゴリズムの種類, 学習データの網羅性について言及しているが, 機械学習技術に対する豊富な知識がなくても概念として理解しやすい.
改善幅が小さい要件:要件 7 (条件 a と c の差 15pt)	サブガイドラインの該当箇所では機械学習の公平性について言及しているが, 推論結果の偏りやそのバイアスを補正する考え方に対する理解が難しい. (後者は, 研究領域として発展中)

II群 $\overline{A_i}$	推定要因
改善幅が大きい要件:要件 3, 8 (条件 a と c の差 52pt, 44pt)	機械学習技術に詳しい技術者は, 機械学習アルゴリズムの種類と性能(要件 1, 6)を深く掘り下げて欠陥事項を指摘する傾向があり, 学習データの偏りと機械学習の再学習時間(要件 3, 8)については関心が低かった. サブガイドラインの中で着目すべき視点を明示することで該当箇所にも意識を向けられる.
改善幅が小さい要件:要件 1 (条件 a と c の差 7pt)	サブガイドラインの対応箇所では機械学習アルゴリズムの説明性について言及しているが, 既に被験者が保有している機械学習技術の知見でも, 基本設計情報に含まれる欠陥を容易に指摘できる.

共通	推定要因
条件 b(汎用ガイドライン参照)の精度が条件 a, c に比べて低い要件:要件 1, 8	要件 1:「説明性が高い機械学習アルゴリズム」という点について, 個別システムに応じた解釈方法が汎用ガイドラインでは解説されていないので精度が下がった. 要件 8:「機械学習の再学習」は当該分野の初歩の知識領域であるため, 敢えてガイドラインには掲載されておらず, それ故, 視点として着眼されにくかった.

6. 考察

6.1. 得られた知見

[仮説に対する整合性]

(関連: 5.2 節 [RQ に関する評価結果])

AI システムの品質保証活動において, 基本設計情報の欠陥を要件群に照らして指摘するタスクについては, 機械学習技術に関する知識や業務経験に依らず, 要件群から導出した QA アセスメントのためのサブガイドラインに基づいて実施した方が精度を改善できる. 記述の抽象度が高い, 既存の汎用ガイドライン[3]を参照するよりも有意に改善する.

[サブガイドラインの記述]

(関連: 5.2 節 要件毎の被験者回答精度)

個別 AI システムの要件群に適合するようなかたちでサブガイドラインを導出する際, 機械学習技術に関する背景知識に依存して理解度にばらつきが生じないようにするため, IGDM-AIQA 法のサブガイドライン導出部において言語化の工夫が必要である.

[品質保証活動の現場での有効性]

(関連: 5.2 節 品質保証におけるサブガイドラインの効果)

現場の実務でシステムやソフトウェアの品質保証活動を行っている担当者が, IGDM-AIQA 法から導出されたサブガイドラインを参照すると欠陥指摘の精度を 4 倍近く改善できるので, QA 担当者の本業の知見を補完しながら実務を遂行する上で合理性が高い.

6.2. IGDM-AIQA 法の実用性

[サブガイドライン導出の再現性]

IGDM-AIQA 法では機械学習, 要求工学(AGORA), 及びシステム安全性向上(FRAM)の学際的知見を活用している. 品質保証部門の現場で必要となる QA アセスメントタスクでは, 各分野の緻密な理論を習得するよりは寧ろ欠陥検出やレビューに必要な視点を保有する方が重要であるため, 初学者レベルの知識を保有すればよい. 本研究のケーススタディに関わった 3 名の研究者は, 計半年間の各分野の自習によりサブガイドラインの導出を行えた. なお, 複数人の視点で抜け漏れを検証すれば, 網羅性や十分性を担保できると考える.

また, 5.2 節「要件毎の被験者回答精度」で示したサブガイドライン毎の QA アセスメント精度のばらつきを考慮すると, サブガイドラインの項目によっては十分な精度を得られていないものがあるため, 振り返りを行って得られた知見を初学者向けにパターン化して反映する等, サブライドラインの記述内容へのフィードバックを繰り返すことで精度向上が見込まれる.

[投資対効果]

品質保証の現場に IGDM-AIQA 法を展開する上で, 従来の開発手法に慣れた QA 担当者に対して, 前記 3 つの分野の初学者レベルの知識獲得を目的とした教育投資を行えばよい. これにより, ガイドラインを使わない場合(図 10 条件 a)の約 4 倍の QA アセスメント精度を見込める. 他方, 汎用ガイドラインを活用するためには, 機械学習有識者の指導の下, プロジェクトでの活用を経験し, ガイドラインの咀嚼方法を学ぶ必要がある. 経営的視点で機械学習有識者の資源を教育活動に振り向けることは, 投資対効果の点で課題がある.

6.3. 妥当性への脅威

・ 被験者の在籍している企業の多くは製造業や情報・通信に属しており, FinTech に特有の機械学習技術の知見は必ずしも持ち合わせていない. 5.2 節の実験(条件 a, b)の試行において, ドメイン固有の知識を十分に持っていなかったことが不利に働き, 条件 c の改善効果を見かけ上増長させた可能性がある.

・ 今回の実験では, 夫々の被験者が条件 a, b, c の下で QA アセスメントを行うような内容とした. 各条件での試行は 1～2 週間間隔で取り組んでもらったが, 過去の実験で得られた知識が次の実験に影響するような経験効果が働いた可能性を否定できない. そこで, 被験者が回答したデータを精査し, 各回での重複回答の比率を分析したところ, 下記に示すとおり調和平均で 14.8～20.9%の範囲に留まっていた. また, 条件 a→b→c という連続的な実験の試行による経験累積効果は確認されなかった. これらのことから, 今回の実験では, 過去の実験で得られた経験知よりも, 各実験回で被験者へ提示したインプット情報の方が実験結果に与える影響が大きかったと考える.

＜各条件間の被験者回答データの重複率＞

I群: [a, b] 19.3%, [a, c] 14.8%, [b, c] 20.9%

II群: [a, b] 17.9%, [a, c] 20.6%, [b, c] 20.5%

・ 本研究では, FinTech(クレジットカードの与信審査)という, 特定領域に対して IGDM-AIQA 法を適用したので, 手法の汎用性を示す上ではさらなる実験が必要である.

7. まとめ

7.1. 成果

本研究では，品質保証の現場で活用しやすい AI システムの品質アセスメントのためのサブガイドラインを導出する枠組みとして IGDM-AIQA 法を提案した．FinTech 与信判定システムを事例に本手法から導出したサブガイドラインを品質保証ケーススタディに適用した結果，ガイドラインがない場合，及び汎用ガイドラインを参照した場合と比較してアセスメントの精度が向上することを確認した．特に，現場の QA 担当者がサブガイドラインを活用した場合に 4 倍程度の精度改善を見込めることが分かった．

7.2. 将来への発展

本研究では，FinTech 与信判定システムを対象に IGDM-AIQA 法の有効性を評価したが，IGDM-AIQA 法によるサブガイドラインの導出方法は，特定のドメインに関係なく汎用的な手法であるため，他ドメインのシステムについても適用が期待される．

8. 謝辞

本研究は，日本科学技術連盟 2020 年度ソフトウェア品質管理研究会（SQiP）研究コース 5「人工知能とソフトウェア品質」の中で取り組んだ内容です．研究活動の場をご提供頂きました SQiP 研究会，並びに本研究の実験にご協力頂きました研究コース 5，一般企業の有志の方々に厚く御礼申し上げます．

付録

本研究のケーススタディとして用いた FinTech 与信判定システムに対する IGDM-AIQA 法の適用結果の詳細は，本論文の著者らが SQiP 研究会で執筆した成果報告書（付録）[18]に記載がある．

参考文献

[1] H. Kaiya, H. Horai, M. Saeki (2002), AGORA: attributed goal-oriented requirements analysis method, 10th Anniversary IEEE Joint International Requirements Engineering Conference (RE'02), pp.13-22.

[2] Erik Hollnagel, Örjan Goteman (2004), The Functional Resonance Accident Model, Cognitive System Engineering in Process Control 2004.

[3] 産業技術総合研究所，機械学習品質マネジメントガイドライン第 1 版, https://www.cpsec.aist.go.jp/achievements/aiqm/ (閲覧 2020-12-27).

[4] 池村，佐藤，松本 (2021), AI 品質マネジメントガイドライン具体化における AI 経験有無の影響明確化, 第 36 年度ソフトウェア品質管理研究会 分科会報告書 pp.107-131.

[5] AI プロダクト品質保証コンソーシアム，AI プロダクト品質保証ガイドライン 2020.08 版，http://www.qa4ai.jp/download/ (閲覧 2020-12-27).

[6] A. van Lamsweerde (2001), Goal-Oriented Requirements Engineering: A Guided Tour, Symposium on Requirements Engineering, 2001, pp. 249-263.

[7] E. Yu and J. Mylopoulos (1994), Understanding “Why” in Software Process Modelling, Analysis, and Design, Proc. 16th Int. Conf. Software Engineering.

[8] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos (1999), Non-Functional Requirements in Software Engineering, Kluwer Academic Publishers.

[9] Nancy G. Leveson (2011), Engineering a Safer World – Systems Thinking Applied to Safety, The MIT Press.

[10] INCOSE (2006), Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, INCOSE-TP-2003-002-03.

[11] 三原，十山，石井，松田，三縄，八嶋 (2016), システムの安全性・信頼性分析手法，SEC journal Vol.12 No.1 Jul. 2016.

[12] 経済産業省 商務情報政策局，割賦販売法，https://www.meti.go.jp/policy/economy/consumer/credit/11kappuhanbaihou.html (閲覧 2020-12-20).

[13] 日本銀行 金融機構局，AI を活用した金融の高度化に関するワークショップ 第 3 回，https://www.boj.or.jp/announcements/release_2019/rel190215d.htm/ (閲覧 2020-12-20).

[14] 小野 (2016)，インテックの与信モデルの特徴と今後の展開，ITJ2016.9 第 17 号.

[15] Rachel K. E. Bellamy *et al*. (2019), AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias, IBM Journal of Research and Development, Vol.63, Numbers 4/5, July-Sept. 2019.

[16] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudik, John Langford, Hanna Wallach (2018), A Reductions Approach to Fair Classification, In Proceedings of the 35th International. Conference on Machine Learning

[17] 佐藤，石川，猪原 (2011)，貢献度と顧客のニーズに関する妥当性の間のコンフリクト検出指標，ソフトウェアエンジニアリングシンポジウム 2011, pp.1-6.

[18] 相津，小宮山，柳原(2021)，ゴール指向要求分析とシステム安全分析を利用した AI システム品質の個別ガイドライン導出方法の提案，第 36 年度ソフトウェア品質管理研究会 分科会報告書 pp.132-155.

DX 推進に向けたエンタープライズアーキテクチャの設計プロセスの検討

宗平 順己
武庫川女子大学経営学部
t_mune@mukogawa-u.ac.jp

要旨

DX レポート 2 では,「デジタルトランスフォーメーションはデジタルエンタープライズになるまでのプロセスのことである」との定義がなされ, デジタルビジネス向けのシステムの開発と既存業務システムのデジタル化の両方の取組を同時に行う必要があるとされている.

デジタルビジネスを支えるデジタルプラットフォームのアーキテクチャについては IT ベンダー等から様々な提案がなされるだけでなく, AWS での構築事例も報告されている.

既存業務システムのアーキテクチャのベストプラクティスについても MIT Sloan からアーキテクチャ成熟度モデルが提案されている.

しかしながら, デジタルビジネスを支えるシステムは単独で構築されるよりも既存業務システムと関連することが多く, 両者を連携させると同時に, 既存業務システムのモダナイズも同時に進める必要がある.

しかしながら, その具体的な方法についてはDXレポート2には明確な提示はない.

本レポートでは, DX の教科書とでもいえる存在である Designed for Digital をベースに, DXに取り組む企業の指針となるアーキテクチャの設計プロセス検討結果を報告する.

1. はじめに

2020 年末に中間発表された DX レポート 2 はデジタルトランスフォーメーション(以降DX)に取り組むためのより具体的な方法が提示され, デジタルビジネス向けのシステムの開発と既存業務システムのデジタル化の両方の取組を同時に行う必要があるとされている.

デジタルビジネス向けのシステムについては既に多くの意欲的な事例が多く出てきているものの, 既存システムのデジタル化(モダナイズ)の方法については具体的な案が示されていない.

そこで, 本論ではまず第 2 章においてDXレポート2について概観し, 第 3 章ではこれから企業が目指すべきデジタルエンタープライズの特性について, MIT Sloan の Designed for Digital に記載内容も参照して整理する. 第 4 章ではデジタルエンタープライズへ変革する(=DX)にあたってのシステム面でのアプローチ内容として 5 Building Block のフレームワークを紹介する. 第 5 章ではデジタルエンタープライズを支えるデジタルプラットフォームについて, 既に有効な取り組み事例が出てきていることを紹介し, 第 6 章で DX 推進のアーキテクチャ面での課題として「既存システムのモダナイズの具体的なガイドの作成」があることを指摘する. 第 7 章以降はこの課題への回答として, MIT Sloan の Architecture Maturity Stages をベースとしたデジタル化への対応方法の検討結果を紹介する.

2. DX レポート2

2020 年の年末, 12 月 28 日に「デジタルトランスフォーメーションの加速に向けた研究会の中間報告書『DXレポート2(中間取りまとめ)』」が経済産業省より公表された[1]. このレポートにおいて,「デジタルトランスフォーメーションとはデジタルエンタープライズになるためのプロセスのことである」と定義されている. このレポートから要点を抜粋して, 以下に記載する. [1]

2.1. DX の誤った理解と取り組みの大幅な遅れ

独立行政法人情報処理推進機構(IPA)が 2020 年 10 月時点での企業約 500 社における DX 推進への取組状況を分析した結果, 実に全体の9割以上の企業が DX にまったく取り組めていない(DX 未着手企業)レベルか, 散発的な実施に留まっている(DX 途上企業)状況であることが明らかになった. (中略)我が国企業全体における DX への取組は全く不十分なレベルにあると認識せざるを得ない.

このことは, 先般の DX レポートによるメッセージは正しく伝わっておらず,「DX=レガシーシステム刷新」, あるいは, 現時点で競争優位性が確保できていればこれ以上の DX は不要である, 等の本質ではない解釈が是となっていたとも言える.

2.2. 経営者の責任

DX の本質とは, 単にレガシーなシステムを刷新する,

高度化するといったことにとどまるのではなく，事業環境の変化に迅速に適応する能力を身につけること，そしてその中で企業文化（固定観念）を変革（レガシー企業文化からの脱却）することにあると考えられる．

当然ながらこうした変革は誰かに任せて達成できるものではなく，経営トップが自ら変革を主導することが必要である．

2.3. もう以前には戻らない

コロナ禍によって，テレワーク等をはじめとしたデジタル技術による社会活動の変化に対応し，新たな価値を次々と産み出している．これは，単なるコロナ環境下での一過性の現象ではなく，人々の固定観念が大きく変化したことを表しているのである．人々は新たな価値の重要性に気付き，コロナ禍において新しいサービスを大いに利用し，順応している．（中略）人々の固定観念が変化した今こそ企業文化を変革する絶好の機会である．ビジネスにおける価値創出の中心は急速にデジタル空間へ移行しており，今すぐ企業文化を刷新しビジネスを変革できない企業は，デジタル競争の敗者としての道を歩むことになるであろう．

2.4. 「両利きの経営」が必要～デジタル一辺倒ではない

「顧客や社会の問題の発見と解決による新たな価値の創出」と，「組織内の業務生産性向上や働き方の変革」という二つのアプローチを同時並行に進めることが重要である．いわゆる「両利きの経営」と言われるように，既存事業の効率化と新事業の創出は両輪で検討すべきである．既存事業の見直しにより産まれた投資余力を新事業の創出にあてることで，企業の競争力と経営体力を高めることが出来る．

2.5. DX への取り組みの3つの段階

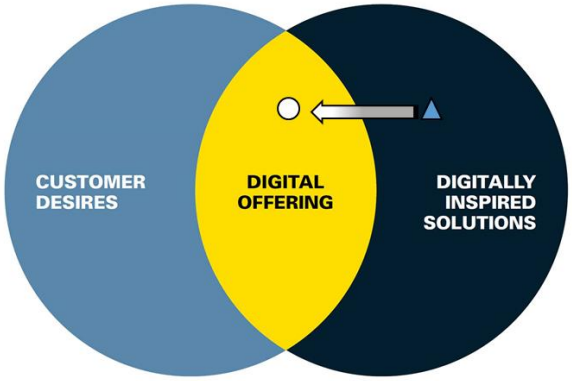
企業が DX の具体的なアクションを組織の成熟度ごとに設計できるように，DXをデジタイゼーション，デジタライゼーション，デジタルトランスフォーメーションという 3 つの異なる段階に分解する．ここでは，デジタイゼーションは，アナログ・物理データの単純なデジタルデータ化のことであり，典型的には，紙文書の電子化である．デジタライゼーションは既存事業の業務・プロセスのデジタル化であり，さらに，デジタルトランスフォーメーションはデジタルエンタープライズとして顧客起点の価値創造ができるようになるために事業やビジネスモデル，システムを変革することである．

3. デジタルエンタープライズの特性

3.1. Digital Offering

デジタルエンタープライズではデジタル技術を用いて新たな顧客経験を提供するのであるが，MIT Sloan は Designed for Digital において，「Digital Offering によって顧客に新たな経験を提供する」というようにその研究成果をとりまとめた．[2]

では，Digital Offering とは何か？ その概念を説明したのが次の図である．



※▲と○，矢印は今回追記

図-1 Digital Offering

左側の円は顧客の期待（潜在ニーズと理解してよい），右側の円は企業側の活動で，デジタル技術を活用した新しいソリューションを示しており，両者が交わったところが Digital Offering であると定義している．

企業が新たな顧客経験を生み出そうとする場合，企業はデジタルで何ができるか経験が十分ではなく，顧客側は実際に手にするまで自分たちは何が欲しいのかわかっていないというのが現実の姿である．

企業は一生懸命マーケットリサーチをして，新たな製品やサービスを提供するのであるが，ほとんどの場合，最初は多くのユーザの評価を得ることができない．ユーザも何が欲しいのか良くわかっていないので，手をかえ品をかえこの潜在ニーズを明らかにしていく必要がある．こうして改良を繰り返すことによって，やがては顧客の持つ潜在ニーズと合致し，ユーザがお金を払っても良いと考えるものに仕上がる．このような状態になったものを「Digital Offering」と呼んでいるわけである．なお，対応する日本語の概念は無いので，英語の表記のままにしておくこととする．

この改良プロセスのことを MIT は Test-and-Learn のプロセスと名付けている．一般には試行錯誤とか Try & Error と繰り返す呼ぶことが多いが，ユーザの評価が得ら

れなかったのは失敗では無く，顧客の期待やニーズについての学びがあったという成果があったととらえ，次の Try に活かそうという意図が込められている．特に日本では，失敗があるとプロジェクトが中止になる企業が多いことから Test-and-Learn と呼ぶのは良いアイデアである．

3.2. DX レポート2における表現

Digital Offering に該当するアプローチについて DX レポート2では，アジャイルマインドでの継続した変革として，図-2 の様に表現している．[1]

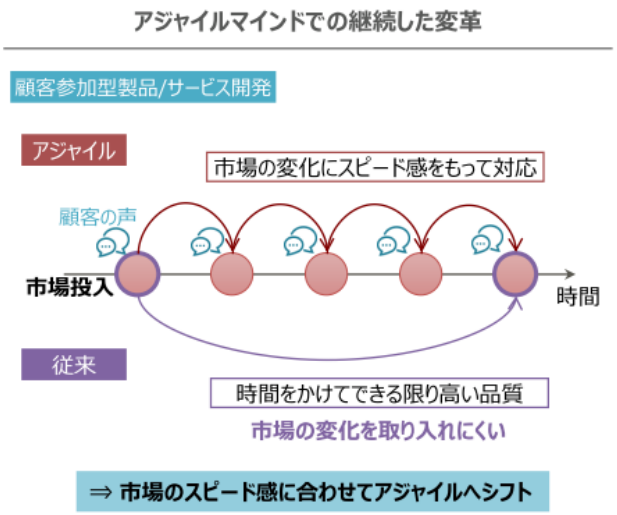
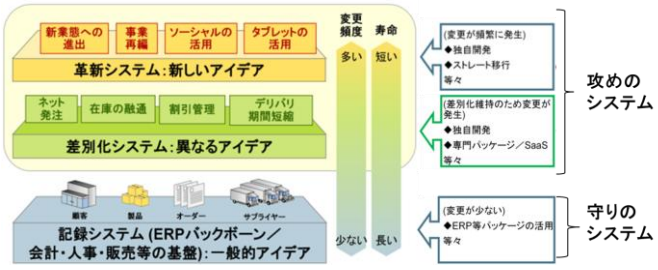


図-2　DX レポート 2 での説明

3.3. デジタルエンタープライズに求められるシステムの特性

図-3 に示すペースレイヤーアプローチはガートナーが 2012 年に提唱したフレームワークで，主にシステムの変更頻度をもとにアプリケーションを分類するもので，ペースレイヤーごとに開発手法が異なる．[3]



出典 5 を元に筆者が加筆

図-3　ペースレイヤーアプローチ

これまでの企業情報システムは，基幹システムと呼ぶ人が多い記録システム（Systems of Record）と戦略システムである差別化システム（Systems of differentiation）とで構成され，これらのシステム開発についてはノウハウの蓄積がなされている．一方，フロントシステムである革新システム（Systems of Innovations）は，これらのシステムとは全く性格が異なり，最初に要件を定義することができない．差別化システムでも，記録システムほど確実性は高くはないが，企業の戦略が先に立案され，その実現に資するものとして要件が定義されている．

これに対し，革新システムは何が正解かわからず，まずはリリースし，顧客や利用者の反応を見ながら，よりロイヤリティを高めるために，素早く改良を加えるということが求められる．リスクを早期に潰す反復型のアプローチでも反復頻度が間に合わず，アジャイル開発が必須であり，リリース期間を短縮するために DevOps を利用し，スケールアウト／インを迅速に行うためにクラウドが必須となっている．

先の DX レポート 2 の定義と照らし合わせると，デジタイゼーションが記録システム，デジタイゼーションが差別化システム，デジタルエンタープライズのシステムは SoI に相当する．

さて，図-4 は航空会社の座席予約システムの構成をペースレイヤーアプローチと関連して説明したものである．[4]

デジタルエンタープライズのシステムは単独で構築されるように誤解されていることが多いが，実際にはこのように，3つのレイヤーのアプリケーションの組み合わせによって構築されることになる．

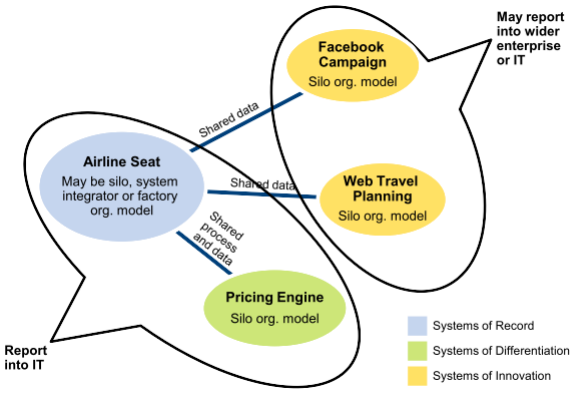


図-4　3つのレイヤーの連係モデル

4. DX へのシステム面でのアプローチ

4.1. 両利き経営のアプローチ

DX レポート2では図-5 に示すように「デジタルトランスフォーメーションにおいて，既存システムは新規構築するシステムと相互に連携しながら見直し，デジタルエンター

プライズの実現を目指す」としており，図-4 と符合する．[1]

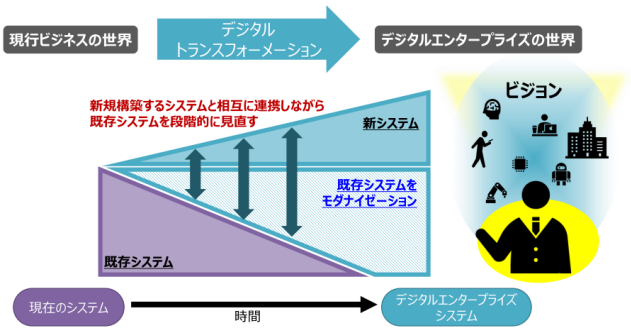


図-5 DX でのシステム面でのアプローチ

4.2. 5 Building Block

MIT Sloan は新たな顧客価値を生み出す DX を実現するためのアプローチとして Designed for Digital において図-6 に示す 5 Building Blocks というフレームワークを提示しており，もし企業がしっかりとした Operational Backbone を構築していれば，他の4つのブロックはその上に築くことができるとしている．図-5 をさらに詳細に説明したものになっている．以降各ブロックについて説明する．[2]

4.2.1. Shared Customer Insights

Digital Offering について企業はデジタルで何ができるか経験が十分ではなく，顧客側は実際に手にするまで自分たちは何が欲しいのか不明である．

そのため test -and -learn のプロセスを繰り返すことによって顧客インサイトを得て企業でシェアできるようになる．いろいろなアイデアの可能性を継続的に試すことが必要で，このことを企業カルチャとして浸透させなければならない．

4.2.2. Operational Backbone

現行ビジネスを支える業務システムを指す．デジタル技術の活用について次の整理をしている．

Digitize：SMACIT（social, mobile, analytics, cloud, IoT）でビジネスプロセスを強化すること．短期的な現行の競争優位の担保に必要であるが，将来の競争優位を作ることはできない．

Digital：これに対し SMACIT を顧客への価値提案のために適用することが Digital であり，デジタルトランスフォーメーションの本質は Digital である．

しかしながら，Digital はすぐに主力になるわけではなく，現行ビジネスを継続せねばならず，Dizitization によって Operational Backbone の効率化(Operational Excellence)を追求する．

4.2.3. Digital Platform

Operational Backbone が信頼性と透明性で顧客満足に貢献するのに対し，Digital Platform は Digital Offering の迅速かつ継続的な改良により顧客を喜ばせる．

このプラットフォームは，ビジネス，データ，インフラの3つのタイプの再利用型コンポーネントから構成され，Digital Offering はこれらの組み合わせによって実現される．このため，各コンポーネントには API が実装される．

当初，Digital Offering のためにモノシリックなシステムを構築してしまうことが多いが，継続的な改善，再利用性，両方の要求を満足するためには，コンポーネントベース（＝マイクロサービス）のアプリケーション開発を行う必要に気づく．

必要なコンポーネントを早期に準備することは難しく，Digital Platform はインクリメンタルに増強することになる．

GE の Predix が Digital Platform の先駆けであり，日本ではパナソニックが既に AWS 上に構築している．

4.2.4. External Developer Platform

Digital Platform の拡張機能であり，デジタルコンポーネントを活用するパートナーとのエコシステムをサポートする．顧客への新たな価値提案は企業が単独でなしうるものではなく，様々なパートナーと共同することで初めて提供できるものとなってきている．先に示した Panasonic や三菱電機はデジタルプラットフォームをパートナーに公開している．

＜DBS の事例＞

- 200 以上の Digital Components を API で公開している
- 認証などのインフラ系のコンポーネントもあるが多くはクレジットカード処理，ローン審査，外貨レート，顧客消費パターンなどビジネス系，データ系のコンポーネントである
- ハッカソンなどを通じてサードパーティとともに新たなサービスを構築している（SoCash が有名）
- パートナーにはポータルを通じてコンポーネントを公開している
  - サイトへのサインアップ
  - 安全なサンドボックスでの自由な実験
  - DBS API の継続的な利用への申請

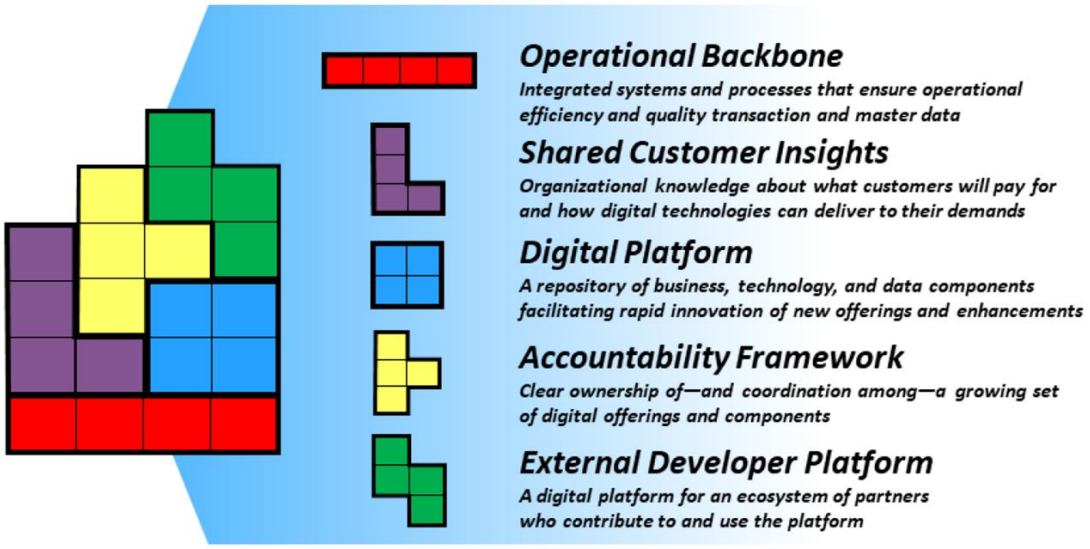


図-6　5 Building Blocks

4.2.5 Accountability Framework

階層型組織は標準化された業務プロセスの効率化には有効であるが，Digital Offering の開発プロセスは Test & Learn であり，独立し empowerment されたイノベーションのチームにより推進される．

ただ各チームが独自の思惑で動くとカオス状態になることから component owner への Accountability Framework が必要となる．

Digital platform の各コンポーネントが Test & Learn によって絶えず更新されていきながら，図-7 に示す，API を通じて他のコンポーネントと協調して動くことを担保する（Autonomy and Alignment）8つの原則からなるマネジメントのしくみが必要となる．

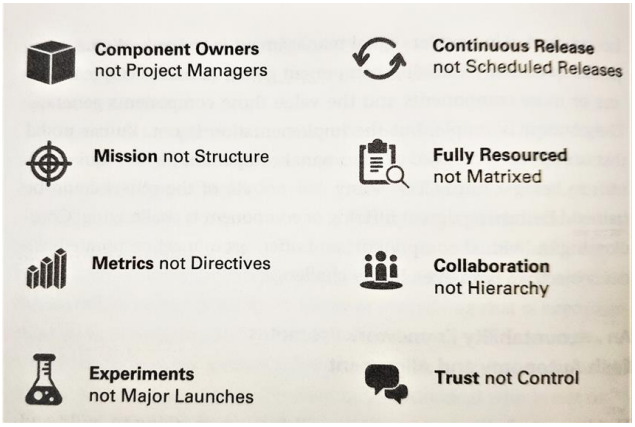


図-7　デジタルコンポーネントのマネジメント原則

5.　Digital Platform の例

デジタルエンタープライズの基盤となる Digital Platform についてはすでに多くの構築事例がある．

5.1. みずほ FG の事例

図-8 はみずほ FG が構築した銀行 API であるが，5 Building Block の External Developer Platform に相当する．[5]

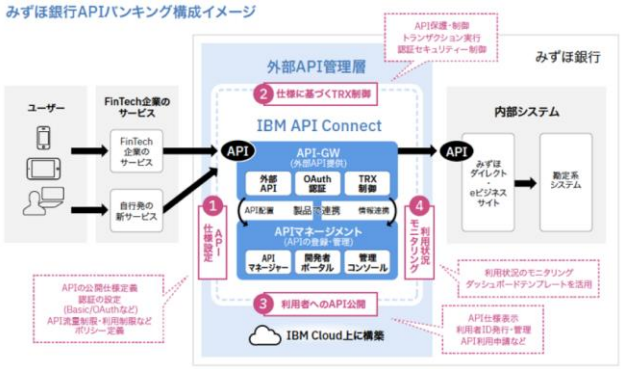


図-8　みずほ FG　API

基幹システムの再構築の際に，業務ロジックを処理する共通業務 AP 層と，顧客との接点となるプレゼンテーション層に分けるというアーキテクチャを採用し，業務ロジックは再利用可能な粒度で Web サービスとして実装していたことから，公開する API をスムースに構築することができた．また，API の仕様も銀行の視点ではなく，Fintech 企業が新しいサービスを構築できるようにするという視点で設計されている．

5.2. Panasonic の事例

図-9 は IoT 家電を支える Panasonic Cloud Service Platform のアーキテクチャである. [6]

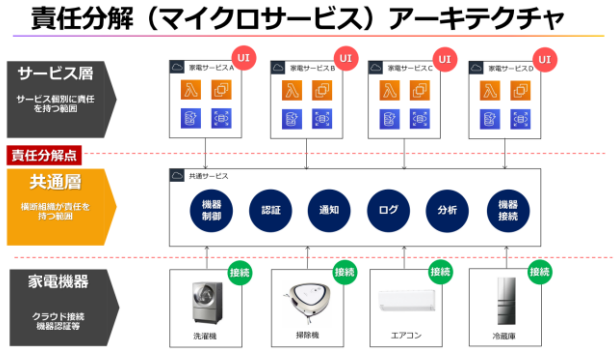


図-9 Panasonic のアーキテクチャ

IoT 家電の前段となるスマート家電サービスには 2012 年から取り組んでおり, 当初はオンプレミス環境で自前主義でしくみを構築していたが, 継続的な進化を続けるために, まずクラウドファーストとし, 家電サービスと共通機能を分離し, 個々に進化できるアーキテクチャに変更した. 従来型のモノシリックなシステムでは, 高速な進化に対応できないことから設計したものであり, 各サービスが独立して運用が行えることを狙いとしている. そして, マイクロサービス化し, 次々とサービスを追加できるようにしている.

そして, 図-10 に示す Panasonic Digital Platform の構築を進めている. [7]

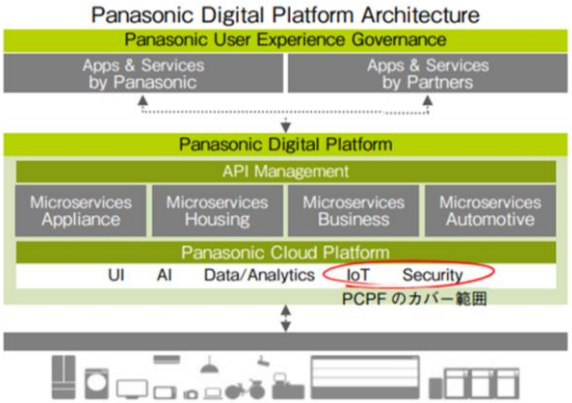


図-10 Panasonic Digital Platform

さらにこの Panasonic Digital Platform を BtoB 向けに公開したのが図-11 に示す μSockets であり, 5 Building Block の External Developer Platform に相当する. [8]

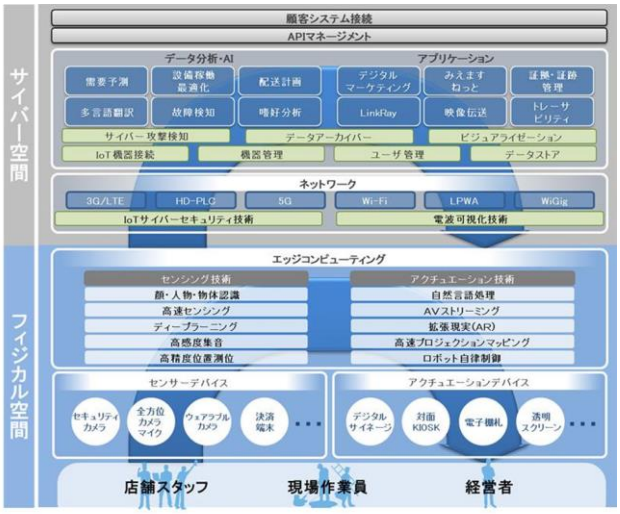


図-11 Panasonic の External Developer Platform

6. DX 推進のアーキテクチャ面での課題

4 章でみたように, Digital Offering を提供する Digital Platform については, 意欲的な取り組みが進められており, AWS をはじめクラウドベンダーも新たなサービスを次々と構築してくれている.

例えば AWS の Re:Invent2020 において, JPMORGAN CHASE が AWS の種々のサービスを使って Digital Platform を構築している旨の講演があった. 既存アプリケーションの再構築を共同で進めていることへの言及もあった.

Digital Platform と既存システムの連携については, 既存システム側に API を設けることが一般的である.

なお, フロントシステムと業務システムとの連携はECサイトの開発で例えば在庫連携など多くの事例がある. ECサイトの開発と業務システムの開発は一般にベンダーが異なるだけでなく, 開発の作法も大きく異なる. そのため両社が共同で開発するということはなく, 両システム間のインターフェイスの取り決めのみを行い, それぞれ独立して開発を進めるのが通例になっている.

そのため, API で既存システムとデジタル側を疎結合にするというアーキテクチャはなじみ深い. (ただし既存システム側のハードルは高い)

ここで課題となるのが, 既存システムのモダナイズである. 既に述べたように, DX レポート2では「デジタルトランスフォーメーションにおいて, 既存システムは新規構築するシステムと相互に連携しながら見直す」とされており, また 5 Building Block においても既存ビジネスの競争力を当面担保するために, 既存システムにデジタル技術の導

入を進めなければならないとされている.

残念ながら DX レポート 2 では, 既存システムを技術的負債として, 計画的な刷新が必要であること, デジタルビジネスと関連する部分から順に再構築するという抽象度の高い記述しかない.

DX の取り組みにあたっては, デジタル側と既存ビジネス側の両方の取り組みが必要なのであるが,「既存システムのモダナイズの具体的なガイドの作成」という重大な課題がある.

7. Operational Backbone のモダナイズ

Operational Backbone の整備に関して, MIT Sloan は 2006 年に Enterprise Architecture As Strategy[9]において Architecture Maturity Stages というフレームワークを提示しており, Designed for Digital ではこのフレームワークを参照するようにとしている.

7.1. Architecture Maturity Stages

Architecture Maturity Stages は, MIT Sloan の Center for Information Systems Research (CISR)が, 1995 年から 2006 年にかけて 456 の企業を研究調査した結果をまとめたもので, 企業が SOA の導入成果を得るために, 図-2 に示すように次の4つのステージを順にステップアップしていく必要があるとしている.

Stage1:Business Silos:
個々の利用部門のビジネスニーズやシステムへの要求を最大限満たすことに注力している

Stage2:Standardized Technology:
技術の標準化や集中化を通じて, ITの効率化を追求する

Stage3:Optimized Core:
企業のビジネスモデルに応じて, 全社視点からのコア業務とコアデータの標準化を進める

Stage4:Business Modularity:
疎結合状態でサービス化されたビジネスプロセスコンポーネントを再利用して, コア部分はグローバル標準を担保しつつ, ローカルの自由度を許している

5 Building Block において,「企業がしっかりとした Operational Backbone を構築していれば, 他の4つのブロックはその上に築くことができる」としているが, これは企業が Architecture Maturity Stages の Stage3 に到達していることを求めている.

ただし, Enterprise Architecture As Strategy はデジタル化以前に記述されたものであり, したがって, DX で使うには Architecture Maturity Stages をデジタル技術を反映したものにブラッシュアップしなければならない.

Stage2 は共通インフラの構築であり, VPC を含めてプライベートクラウドを構築することで多くの企業は対応済である. したがってデジタル技術反映の対象となるのは Stage3 である.

表-2 は Stage3 での取り組み内容を EA の体系に従って整理したものである. コアコンピタンスを支えるビジネスプロセス, データ, アプリケーションの標準化を進める.

表-2 Architecture Maturity Stages の内容

	Stage3 Optimized Core
BA	・ビジネスプロセスの標準化を進めている ・定義されたビジネスプロセスは企業のコアコンピタンスをとらえている ・コアプロセスの変更は難しいが, 新たなプロセスの追加は容易かつ迅速になっている ・再利用可能なデータとビジネスプロセスはコアプロセのセットとして構築されている. ・共通データやコアプロセスの標準化では, ビジネスプロセスの設計の主体を個別ビジネスリーダから移管する
DA	・データを企業全体の視点からみる ・個々のアプリからトランザクションデータを抽出することで冗長性を減少させ, 他のアプリから利用できるようにする ・重要な企業データへのインターフェイスを準備している ・コアデータが企業のコアコンピタンスを反映している ・コアデータの変更は難しいが新たなサービスを追加は容易かつ迅速になっている ・再利用可能なデータとビジネスプロセスはコアプロセのセットとして構築されている.
AA	・アプリを企業全体の視点からみる ・アプリケーションの標準化を進めている
TA	・共通インフラに関わる投資はほぼ終息し, 投資の重点は共通アプリ, 共通データ整備移行する ・インフラ部分については効率化が次の課題となる
マネジメント	・IT 投資が個別アプリやシェアードインフラから全社システムやシェアードデータに移る ・IT の役割は再利用可能なデータとビジネスプラットフォームを構築することにより企業の目標達成を容易にすることである ・CIO は標準化がイノベーションを可能にすると信じている ・コアデータやコアプロセスは既存および新規の業務や顧客対応のための基盤を提供している ・各企業のユニークな戦略的優位性はこの基盤の構築に起因している ・標準化やコアの定義はトップダウンでのみ実現できる

7.2. Operating Model

この Stage3 おいてコアの標準化を図る場合，自社(グループ)の事業モデル(Operating Model)が，表-3 の4つのモデルのいずれに該当するか見極めたうえで，コアプロセスの標準化を進めるべきであるとしている．

表-3　4 つの Operating Model

Business Process Integration			
	High	Coordination	Unification
	Low	Diversification	Replication
		Low	High
		Business Process Standardization	

Stage3 での標準化の範囲が図-12 に示すようにオペレーティングモデルによって異なるからである．

①Unification

全社的に基盤プロセス の共通化を進めるこのオペレーティングモデルでは BA，AA，DA，TA 全てのレイヤーで標準化を行う．

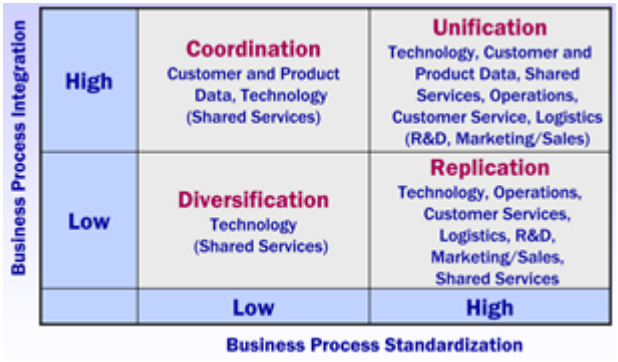


図-12　Operating Model とその標準化ポイント

②Replication

扱う商品や顧客ターゲットが異なるものの業務プロセスが類似するこのオペレーティングモデルでは DA 以外を標準化する．

③Coordination

バリューチェーンプロセス毎に事業部制を敷く場合に該当するこのオペレーティングモデルでは，顧客は共通であるが，各事業部のプロセスは固有であることから，DA と TA が標準化対象となる．

④Diversification

このオペレーティングモデルは顧客も業務プロセスも異なるが持ち株会社によってグループ化されている場合が該当するため，標準化の対象は TA のみとなる．ただし，グループ傘下の各社はそれぞれの事業群または会社内において他の3つのどのオペレーティングモデルに該当するのかの検討を行う必要がある．

以降，Optimized Core の Operating Model 毎のアプローチ方法をレビューしつつ，デジタル対応の方法について考察する．ただし，Diversification についてはインフラの共通化のみとなるので，対象とはしない．

8. Operating Model 毎のデジタル化の検討

8.1. Unification

Unification での EA 設計方法を示したのが図-13 である．図の上段は設計プロセス，下段は成果物の表現方法を示している．この2段構図は4つのオペレーティングモデルに共通である．

Unification ではまず主要顧客を特定し，その顧客にサービス・商品を提供するためのコアプロセスを特定し，その標準プロセスを設計するとともに連携させるアプリケーションを決定する．並行して共有化すべきデータも特定する．どのような技術を用いるかは決定するがこれだけは主に IT 部門の役割となる．

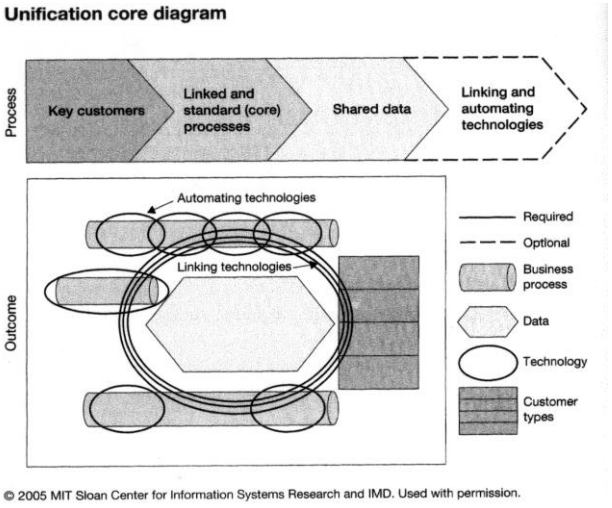


図-13　Unification での EA 設計方法

この設計手法を用いてデルタ航空の EA を表現したのが図-14 である．

デルタ航空の例では Operational Pipeline，Customer Experience，Business Reflexes，Employee Relationship Management の4つのコアプロセスが識別され，それらで共通利用すべき9つのデータベースが定義されている．

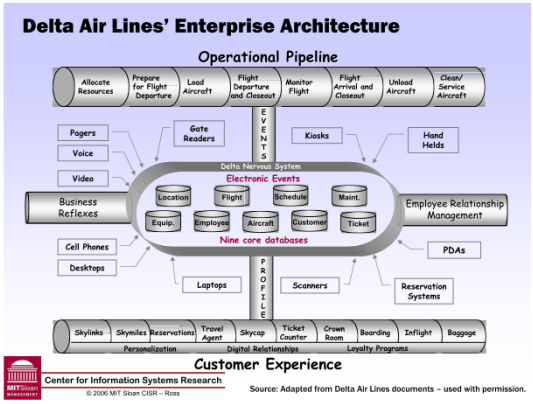


図-14 Unification での EA 設計例

＜デジタル化のポイント＞

標準化の対象となるコアプロセスは主要顧客に商品やサービスを提供するプロセスであることから，プロセス識別後の詳細なデザインはサービスデザイン／デザイン思考を用いるデジタルエンタープライズの設計プロセスと同じものを採用する必要がある．この設計プロセスではサービスブループリントを描いてフロント側と業務システム側のインタラクションを定義し，その後システム要件の詳細定義を行う．

デジタル化の要素としてはフロント側からの Input が IoT になったり，データ処理において AI を用いて Personalization を行うことなどが想定される．

8.2. Replication

Replication での EA 設計方法を示したのが図-15 である．Replication では標準化（複写される）プロセスを特定し，その標準化プロセスを支え共通利用させるアプリケーションとその技術を決定する．その後必要に応じてグループ企業／事業部門ごとのデータを設計する．

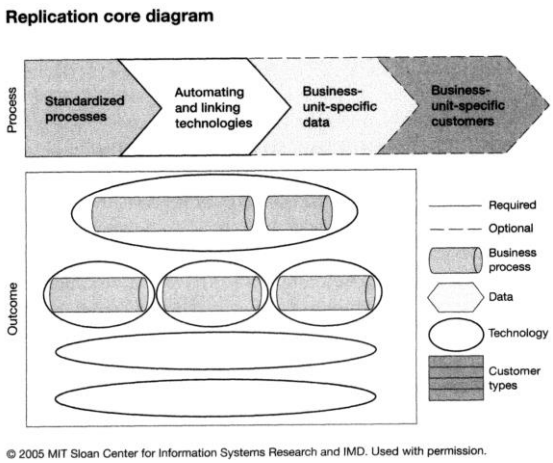


図-15 Replication での EA 設計方法

この設計手法を用いて ING Direct 社の EA を表現したのが図-16 である．

ING Direct 社は Direct Banking サービスを各国で展開するにあたり，ローカルルールは許さない替わりに，各国で利用するビジネスモジュールを選択できることにしており，図には7つのサービスグループとして表示されている．

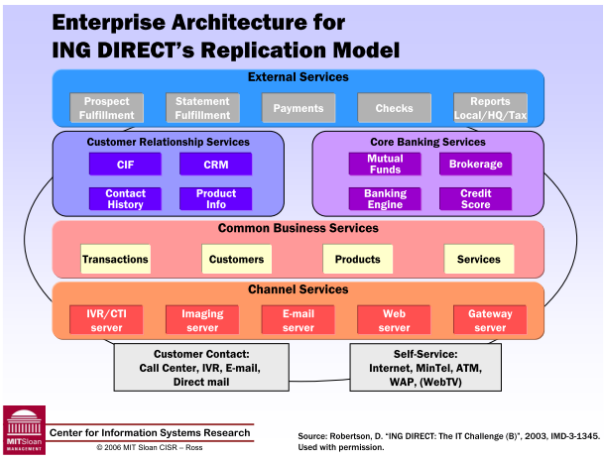


図-16 Replication での EA 設計例

＜デジタル対応のポイント＞

アプリケーションについて標準化と若干のローカライズを許可するようにする必要があることから，SaaS のマルチテナントと同様の設計をする必要がある．また個々のロジック部分についても AI の導入なども検討する必要がある．

8.3. Coordination

Coordination での EA 設計方法を示したのが図-17 である．

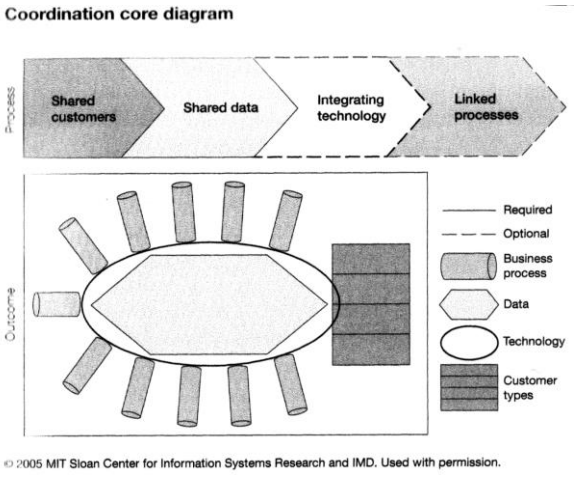


図-17 Coordination での EA 設計方法

このオペレーティングモデルでは複数のビジネスユニ

ットで顧客を共有することから，まずはその対象となる顧客を見極め関連して共有すべきデータを決める．このデータを共有し連携するプロセスを抽出し，そのうえでIT部門はどのように共有データへのアクセスやプロセス連携を実現するのかを決める．

この設計例を示す図－18 では，さまざまな保険・金融商品の横断的な統合顧客サービスを実現するイメージが示されている．左側は顧客側のプロセス，右側は Metlife 社内のアプリケーションを示しており，統合 HUB という概念で統合データにアクセスさせている．

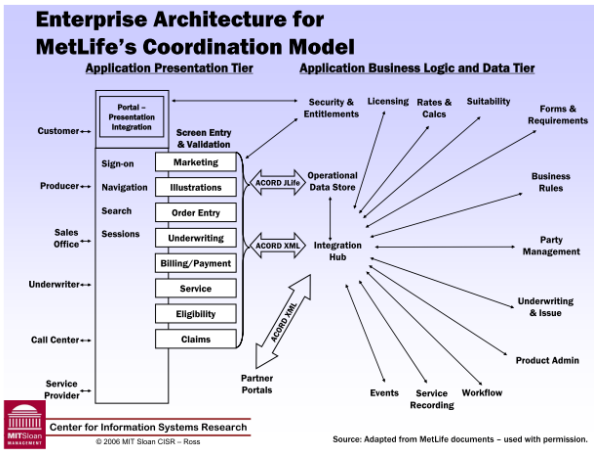


図-18 Coordination での EA 設計例

＜デジタル対応のポイント＞

Coordination ではデータ基盤のアクセスへの設計が鍵となる．

共有するデータにIoTやSNSのデータなどが含まれることを想定したデータ基盤を設計する必要がある．AIなどの分析系の技術の適用は個別アプリで検討することとなる．

9. おわりに

以上，DXの検討において課題となっている，既存業務システムのモダナイズの方法について Architecture Maturity Stages の Stage3 に到達するものとし，Operating Model 具体的なに基づいた設計プロセスについて検討した．

検討した結果，デジタル化以前にまとめられたフレームワークではあるが，実装方法で最近のデジタル技術の適用を考慮する必要はあるものの，設計プロセスそのものは変更なく適用できることは分かった．

今後の課題として，DX の事例研究ではデジタルエンタープライズ側の取り組みに焦点が当たっていることが多いことから，同時並行的に進められている既存システムのモダナイズ事例を調査し，今回考察したフレームワークとの整合性を検証することが必要である．その際，Operating Model の検討にあたって，全社を画一的に議論することは現実的ではない．業務群ごとにどの Operating Model を採用するのかを検討すべきであることも留意しておく必要がある．

参考文献

[1] デジタルトランスフォーメーションの加速に向けた研究会の中間報告書『DX レポート 2（中間取りまとめ）https://www.meti.go.jp/press/2020/12/20201228004/20201228004.html

[2] Jeanne W. Ross, Cynthia M. Beath, Martin Mocker, “Designed for Digital: How to Architect Your Business for Sustained Success”, The MIT Press September 2019.

[3] https://www.slideshare.net/jeffshuey/pace-layered-approach-and-winshuttle-share-point-conference-nov-2012-jeff-shuey

[4] Mary Mesaglio, Matthew Hotle, “Pace-Layered Application Strategy and IT Organizational Design: How to Structure the Application Team for Success”, Gartner, 2016.7 https://www.gartner.com/binaries/content/assets/events/keywords/applications/apn30/pace-layered-applications-research-report.pdf#search=%27PaceLayered%27

[5] https://www.ibm.com/blogs/solutions/jp-ja/ca-mizuho/

[6] 大穂雅博,箱田航太郎,「パナソニックの IoT 家電を支えるクラウド基盤の進化」, AWS SUMMIT Tokyo 2019

[7] 浅井将則他「パナソニックの IoT サービスを支えるクラウドプラットフォームの開発」,Panasonic Technical Journal Vol.64 No.1,May 2018

[8] https://news.panasonic.com/jp/press/data/2017/10/jn171002-4/jn171002-4.html

[9] Jeanne W. Ross, Peter Weill, David Robertson, “Enterprise Architecture As Strategy: Creating a Foundation for Business Execution”, Harvard Business School Press; 2006/8/8

リアルタイムトラフィック可視化システムとその拡張性について

吉田和幸　宇野秀亮　池部実　吉崎弘一
大分大学　大分大学　大分大学　大分大学
yoshida@oita-u.ac.jp

要旨

ソフトウェア開発の初期段階においてプロトタイプを組み立てて実際に動かしてみる作業は有用である．インターネットトラフィックの増大に伴い，サーバやそこで動作しているサービスを調査するスキャンも増加している．スキャン活動のいろいろな側面をリアルタイムに表示するシステムを作成した．宛先 IP アドレスの下位2オクテット，宛先ポート番号など 16 ビットを取り出し，それを基に分類し，それぞれのパケット数をカウントして，256×256 に展開し，全体の分布を表示している．これにより，ネットワークトラフィックの概要の把握が容易になる．パケットヘッダの中の 16 ビット幅のいろいろな場所について分布を表示できるように容易に拡張できるように設計した．

1. はじめに

インターネットの普及に伴い，インターネットのトラフィックも増大している．「我が国のインターネットにおけるトラヒックの集計・試算」[1]によると，2020 年 11 月時点で日本における固定系ブロードバンド契約者の総ダウンロードトラフィックは約 19.8Tbps であり，前年同月比で 56.7%増加している．また，ブロードバンドサービス契約者の総アップロードトラフィックは約 2.4Tbps であり，前年同月比で 51.1%増加している．これに伴いサーバやそこで動作しているサービスを調査するスキャンも増加している．

個々のパケットや，サーバ・クライアント間の一連のパケット(フロー)を詳細に調べることは，不正侵入等の検知には重要であるが，大量のパケットが来るスキャン等の傾向を把握するには，個々のパケットやフローの解析では困難である．宛先 IP アドレスの第 3，第 4 オクテットやポート番号など，パケットヘッダ中の 16 ビットの箇所の分布をリアルタイムに表示するシステムを設計・試作した．

本論文では，2 章で関連研究について述べ，3 章でシステム構成について述べ，ラピッドプロトタイピングの観点からプログラム構成について述べる.4 章で実行例を示して，5 章でまとめと今後の課題について述べる．

2. 関連研究

NICTER (Network Incident analysis Center Tactical Emergency Response)[2]は，サイバー攻撃観測・分析・対策システムである．ダークネットを観測することで収集したパケットデータを解析している．ダークネットとは，インターネットに向けて公開されている IP アドレスのうち，未使用のアドレスによって構成されたネットワークであり，通常の通信では，存在するホストに対してパケットを送信するため未使用のアドレス空間宛に通信が発生することはない．そのため，ダークネットで観測されるすべてのトラフィックは不正な通信である可能性が高い．ブラックホールモニタリングと呼ばれる，受信されるパケットに対して一切応答せず，記録する手法を取っている．NICTER による観測・分析結果の一部は，NICTERWEB 2.0[2]によって一般公開している．NICTERWEB 2.0 ではダークネットトラフィックを，立方体にマッピングした「Cube」や，地図上に表示した「Atlas」によって，リアルタイムに可視化している．

新川ら[3]は IP アドレスの下位 8bit，ポート番号の下位 8bit を用いた 2 次元マトリクスを表示する手法を提案，実装した．2 次元マトリクス上の点をクリックすると，その通信に関する IP アドレス，ポート番号，時間，プロトコル，方向が表示される．しかし，下位 8bit が重複するホストの通信が複数ある場合，重なって表示されてしまうため，ホストごとの正確な通信状況を把握できない．

宇都木ら[4]は TCP コネクションごとのトラフィックを帯状に表示し，リアルタイムなトラフィック量によって色のグラデーションで示すことでトラフィックを可視化した．ポート番号や IP アドレスによってトラフィックをソートしたり，フィルタしたりできる．トラフィックの可視化に特化しており，パケットの宛先の分布などを把握できない．

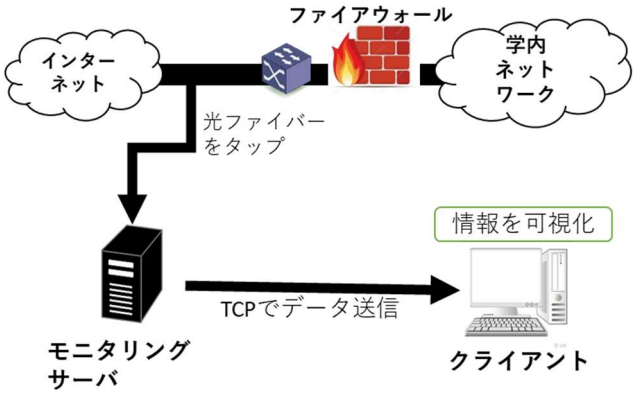


図 1　システム構成

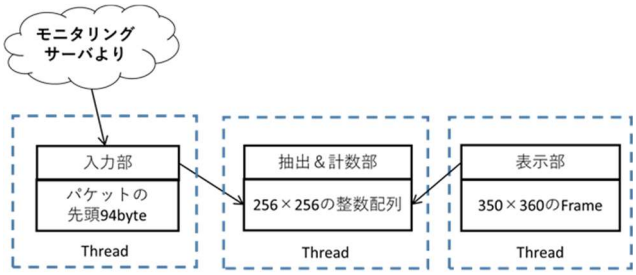


図 2　プログラム構成

3. リアルタイムトラフィック表示システム

3.1. システム構成

図 1 にシステム構成を示す. インターネットへのアクセス線(10Gbase-SR)×2 の光ファイバをタップにより分岐して, モニタリングサーバに送る. モニタリングサーバでは, tcpdump コマンド[5]によりインバウンド(外から内)パケットのヘッダ情報を収集し, ncat コマンド[6]を用いて, クライアント PC へ送る. クライアント PC では, Java プログラムにより, データを読み込み, 必要な部分を抽出・計数し, 結果を表示する. Java プログラムについては次節で詳述する.

3.2. プログラム構成

プログラムの構成を図 2 に示す. モニタリングサーバから毎秒 30000 個ほどのパケットヘッダが送られてくる. 入力部は, 独立した thread とし, 入力したパケットヘッダ情報をパラメータに抽出＆計数部のメソッドを呼び出し, 計数させる. 抽出＆計数部では, 計数結果がオーバフローしないように, 指数平滑移動平均を計算する thread を 10 秒ごとに動作させている.

```
protected int d[][]=new int[256][256];
public boolean cnt(byte b[]) {
        if(b[23]==1)                    // protocol=ICMP
            d[b[32]&0xff][b[33]&0xff]++;
                // 宛先IPアドレスの３，４オクテット
        return r;
}
public int getd(int x,int y) { return d[x][y]; }
```

図 3　抽出＆計数部のメソッド例

図 3 に, 抽出＆計数部の計数するメソッド(cnt)と, 結果を表示部へ渡すメソッド(getd)を示す. cnt メソッドは, パケットの先頭 94byte(イーサヘッダ 14byte, IP ヘッダ 20～60byte, TCP ヘッダ 20byte) を受け取り, 計数したいパケットを抽出し, 計数する. この例では, icmp パケットの, 宛先アドレス(学内ネットワーク側のアドレス)の第 3, 第 4 オクテットで計数している. getd メソッドは, 2つのインデックス値をもらい, 対応する計数結果を返している.

表示部は, 抽出＆計数部から計数結果の 256×256 の整数配列をもらって, 計数結果を色で表して, 256 dot ×256 dot の散布図として表示する. 図 4 **エラー! 参照元が見つかりません。**に計数結果から色への対応を示す. 計数値を白(0)～青(4)～水色(8)～黄色(16)～赤(32)～マゼンタ(64)～紫(128)～黒(255)へマップしている(()内の数は計数値). 値が小さいところでは, 早く変化し, 値が大きいところでは, ゆっくり変化するようにしている. 図 5 に表示例を示す. 抽出している 16 ビットのうち上位 8 ビット(0～255)を縦軸に, 下位 8 ビットを横軸にして 256×256 のマトリックスで表示し, 上下端, 左右端に 8 ごとに目盛りを表示している. この例では, インバウンドパケットの宛先 IP アドレスの第 3 オクテットを縦軸に示し, 第 4 オクテットを横軸に示す. なお, 縦軸では, 上端が 0, 下端が 255, 横軸では左端が 0, 右端が 255 である.

cnt メソッドの抽出・計数条件を変更することにより, ポート番号, 宛先アドレスの第 3, 第 4 オクテット, 送信元アドレスの第 1, 第 2 オクテット, 宛先アドレスの第 4 オクテットとポート番号の下位オクテット[3]など種々のフィールドを, 計数することができる. 入力部から複数の抽出＆計数部を呼び出せるようにして, 新たなフィールドの計数を容易に組み込めるようにしている(図 6).

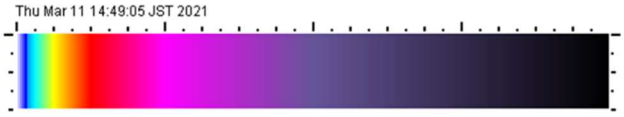


図 4　カラーマップ

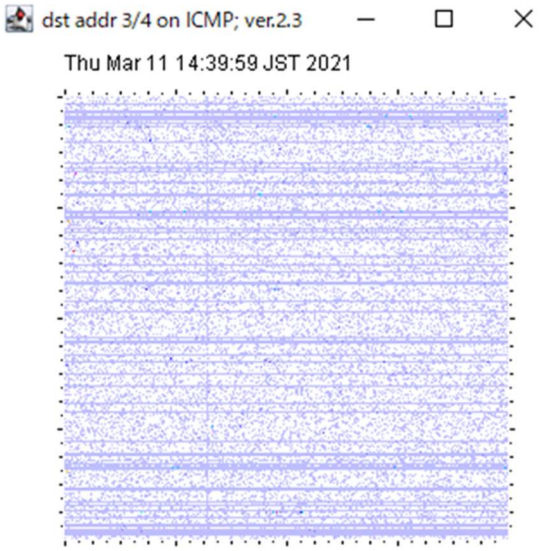


図 5 ICMP パケットの宛先アドレスの分布

3.3. 実行例

実行結果を図 5 のほかに，図 7～図 15 に示す．これらは，本システムを 52 分間実行した結果であり，モニタリングサーバ上で動かしている tcpdump コマンドによると 84,750,410 パケット受信し，233,253 パケット(0.3%)取りこぼしている．10 秒ごとに値を半分にする指数平滑移動平均を計算しているため，おおむね 20 秒間のパケット到着数を示している．

図 7 は，インターネットから学内LANに入ってくるすべてのパケット(インバウンドパケット)の宛先 IP アドレスの第 3，第 4 オクテットの分布である．学内 LAN では，/16 の

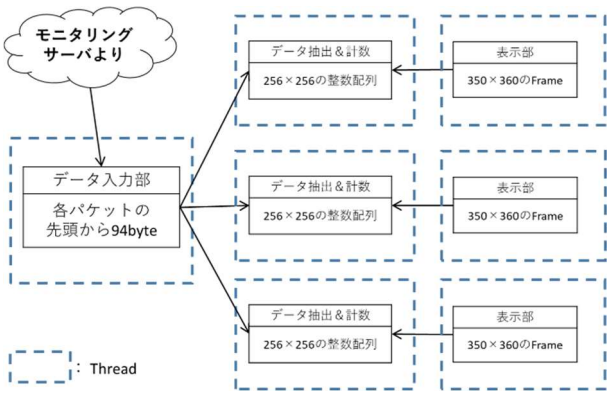


図 6 プログラムの拡張

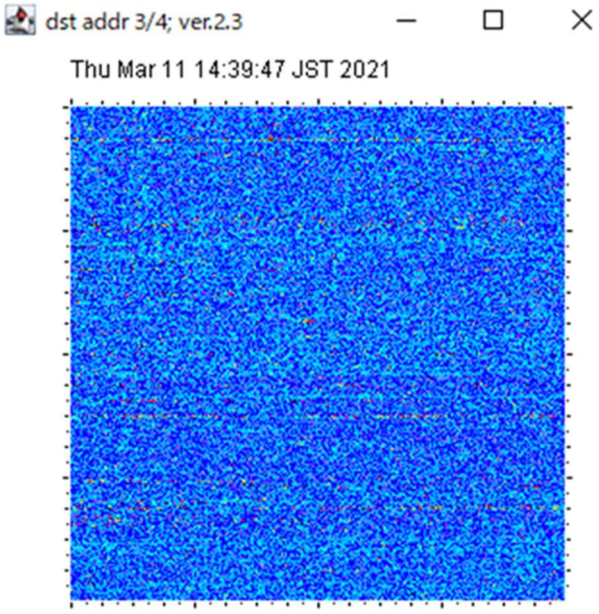


図 7 宛先 IP アドレスの分布

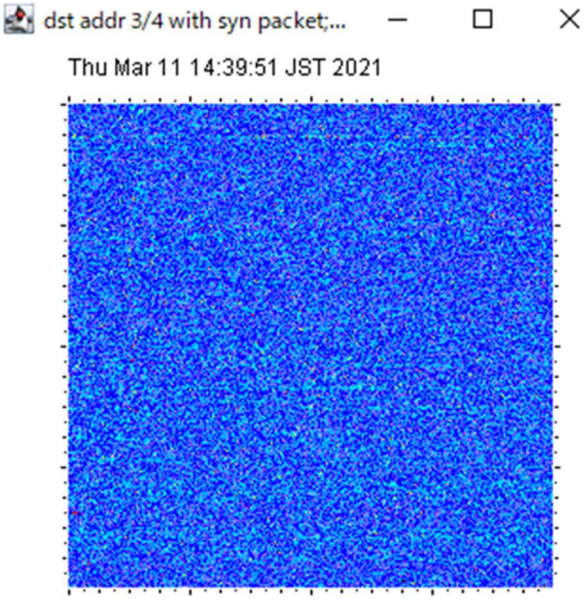


図 8 SYN パケットの宛先アドレスの分布

IP ネットワークを利用しているので，学内 LAN のアドレス空間すべてを表現できる．

図 8 は，インバウンドパケットのうち，TCP のコネクション確立要求である SYN フラグのみ立っているパケットの宛先 IP アドレスの第 3，第 4 オクテットの分布である．図

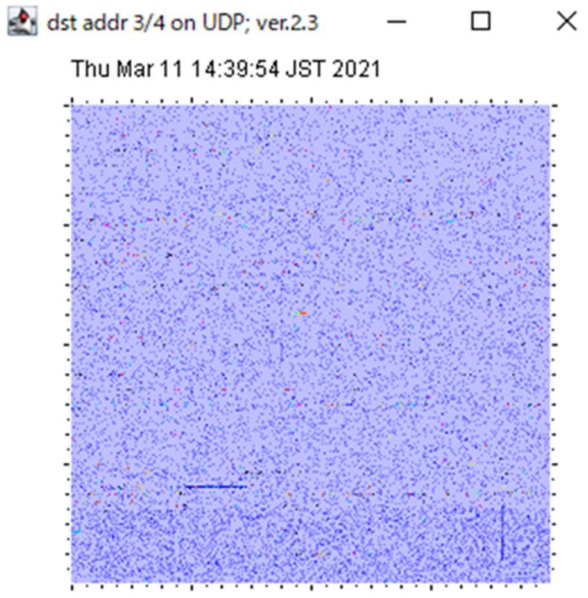


図 9 UDP パケットの宛先 IP アドレスの分布

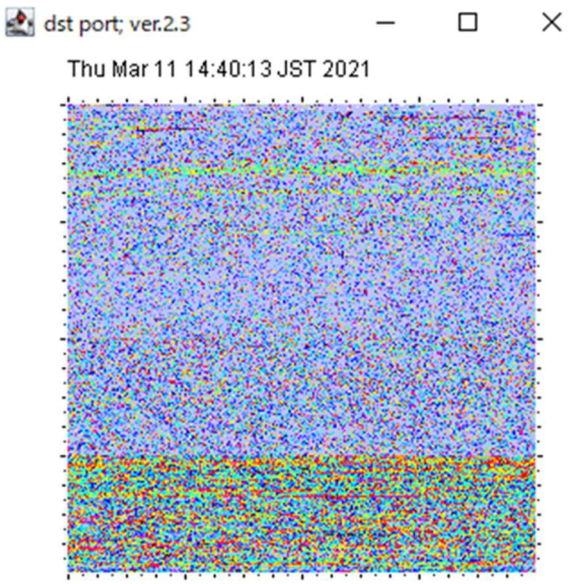


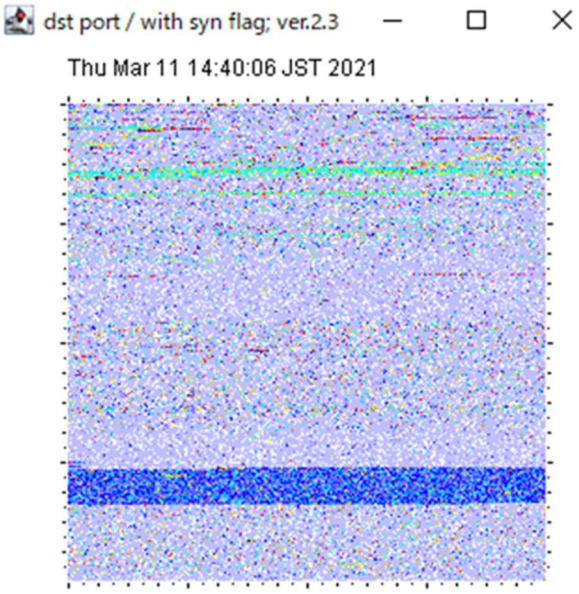
図 10 ポート番号の分布

6 と図 7 とで，図 6 のほうが計数値が大きい(水色)が，大きな違いはない．スキャンの大部分は，TCP スキャンであることがわかる．

図 5，図 9 は，インバウンドパケットのうち，ICMP，UDP パケットの宛先 IP アドレスの第 3，第 4 オクテットの分布である．図 9(UDP)では，アドレス空間全体をスキャンされてはいるが，短い横線と縦線が見える．第 3 オクテットを固定して，第 4 オクテットを変化させるスキャンと，第 4 オクテットを固定して第 3 オクテットを変化させるスキャンが行われたことがわかる.図 9(ICMP)は，図 3 で示したプログラムで集計したものである．横線が目立つが，少数の縦線も確認できる．

図 10 には，すべてのインバウンドパケットのポート番号の分布，図 11 には TCP の SYN フラグのみ立っているパケットのポート番号の分布である．図 10 では，ポート番号の上位バイトが 192 以降の部分が多くなっている．これば，学内の PC が外部のサーバにアクセスするときのエフェメラルポートであろう．この影響を排除するため，外部からの TCP コネクション要求に絞ると図 11 になる．上位バイトが 32～40 付近(ポート番号 8000～10000)と 192～216 付近(ポート番号 49000～55000)で，スキャンが多いことがわかる．

文献[3]にならい，図 12 に，すべてのインバウンドパケットの宛先 IP アドレスの第 4 オクテットとポート番号の下位バイトの分布を，図 13 に，TCP の SYN フラグのみ立っているパケットの宛先 IP アドレスの第 4 オクテットとポート



番号の下位バイトの分布を示す．図 12 で縦線が目立つ

図 11 SYN パケットの宛先ポート番号の分布

のは，図 10 と同様にエフェメラルポートの影響である．それを排除した図 13 では，横線(ポートを固定して，IP アドレスを変化させる)が目立っている．うっすらではあるが縦線(IP アドレスを固定して，ポート番号を変化させる)のスキャンも見える．

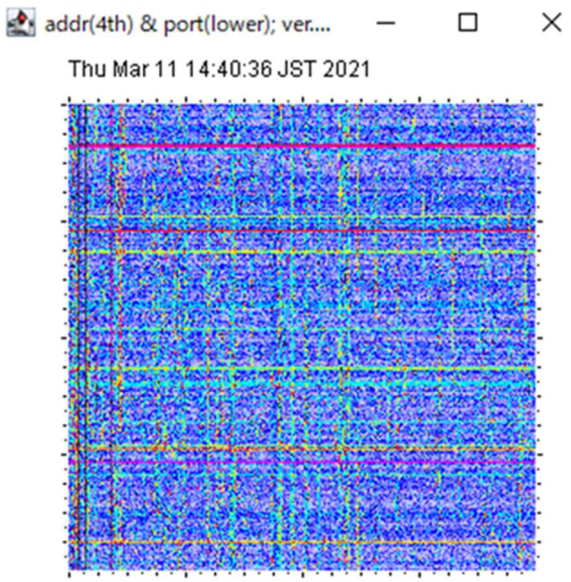
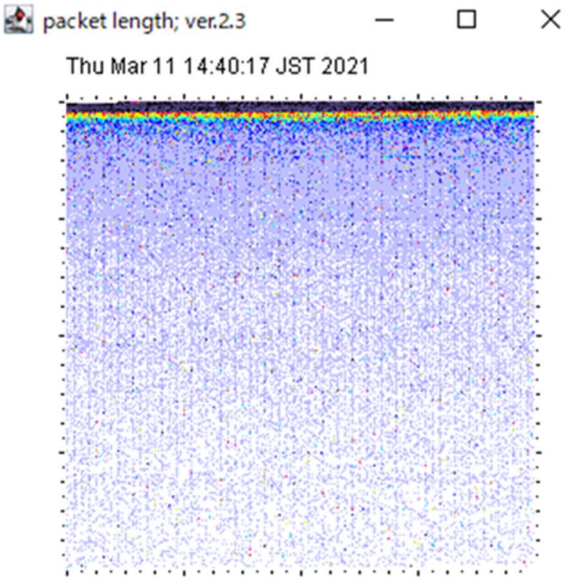


図 12 宛先アドレスとポート番号の分布

図 14 にパケット長の分布を示す. イーサネットの最大パケット長は 1500 であるが, モニタリングサーバのインターフェースカードのハードウェアオフロード機能により, TCP パケットが最大 window size になるまで結合された長大なパケットを観測している. 長大なパケットの長さが, 8 の倍数に集中しているので, この時は, Window size 拡張オプションが 3 (8 倍)の TCP コネクションで, 大量のデータの受信を行っていたことがわかる.



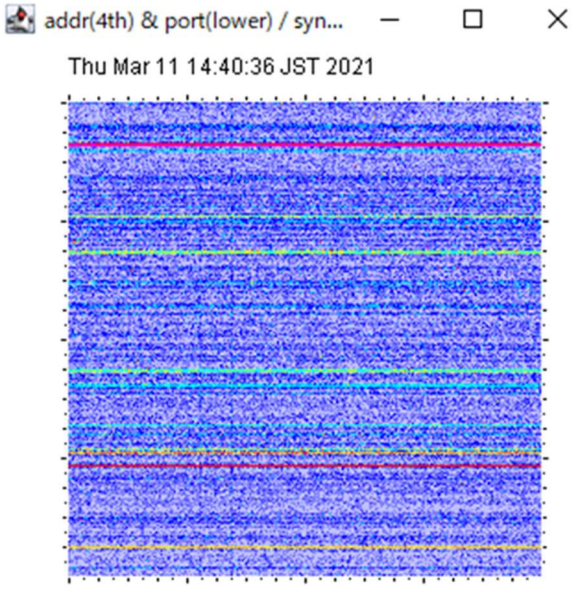


図 14 パケット長の分布

4. おわりに

パケットヘッダ情報を解析し，パケット数を計数することにより，スキャン活動の宛先 IP アドレスの分布，ポート番号の分布等をリアルタイムに可視化するリアルタイムトラフィック可視化システムについて述べた. その実行例から, **ポート番号の分布では，8000～10000，49000～55000 の 2 か所に集中していること，宛先 IP アドレスは第 4 オクテットを変化させるスキャンが大部分であることなどがわかった.**

パケットの新たな分布を調査したいとき容易に拡張できる. 本システムを使い, 新たな分布図の作成を試行することにより, ネットワークの運用管理, セキュリティ監視にさらに有用なものを見つけたい. .

参考文献

[1] 総務省, “我が国のインターネットにおけるトラヒックの集計・試算”, https://www.soumu.go.jp/menu_news/s-news/01kiban04_02000182.html, 2021.2

[2] nicter, https://www.nict.go.jp/, 国立研究開発法人情報通信研究機構,閲覧日: 2021 年 2 月 4 日.

[3] 新川拓也,山之上卓,IP アドレスとポートによる二次元平面を用いた通信トラフィックの可視化について, 情報処理学会研究報告, Vol.2006,No97,pp.31-

36,2006 年 9 月 15 日.
[4] 宇都木進,渡邉晶,TCP コネクション単位でトラフィックの可視化を行うツールの開発,情報処理学会研究報告,Vol.2009-IOT-7,No.4, pp.1-5,2009 年 10 月 9 日.
[5] tcpdump コマンド, https://www.tcpdump.org
[6] ncat コマンド, https://nmap.org/ncat/

個体群動態論に特化したマルチエージェントシミュレーション基盤 Re:Mobidyc

小田 朋宏
株式会社 SRA
tomohiro@sra.co.jp

Gael Dur
静岡大学
dur.gael@shizuoka.ac.jp

要旨

Re:Mobidyc は生物の個体群動態に特化したマルチエージェントシミュレーション *(MAS)* 基盤である．科学研究の道具としての要請に応えるために，*Re:Mobidyc* ではスケーラビリティ，再現性，検証可能性を重視してモデリング言語，モデリング環境および実行系を設計した．本論文では，*Re:Mobidyc* のモデリング言語，モデリング環境および実行系について，個体群動態研究からの要求に基づいた設計指針および実現について紹介し議論する．

1. 背景

Agent-Based Modelling and Simulation (ABMS) は，1990 年代以降の環境学研究に広く応用され，様々な構文および意味論を持つシミュレーションツールが開発されてきた．特に，個体の行動をエージェントとしてモデリングする個体ベースモデル (Individual Based Model, IBM) は，個体差の表現や，各個体の一生の中での行動の変化（生活史段階），環境への適応と進化，個体から環境への影響などを表現するための柔軟性を持つことから，環境学や生物学の研究に用いられてきた [1]．現在でも NetLogo [2] をはじめとする多様なマルチエージェントシミュレーション (Multi-Agent Simulation, MAS) 環境が開発され，環境学や生物学の多様な領域の ABMS に利用されている．

ソフトウェア研究の視点からは，ABMS は LOGO をはじめとするプログラミング言語の応用と見なすことができる．それらのプログラミング言語は汎用のプログラミング言語であり，プログラムの構成要素は生物学や環境学の用語ではなく，プログラミングの用語で定義されている．例えば，LOGO は手続き型のプログラミング言語であり，タートルの移動のための専用の命令文や，変数への代入文や，条件分岐やループなどの制御構造や，手続き呼び出しにより，タートルの動きを定義する．プログラミング言語のこれらの機能は，チューリング完全なプログラムを構成するものであり， 計算の表現の強力さとともに，ループの停止性やプログラムの副作用など，ソフトウェア研究がこれまで取り組んできた多くの困難をもたらす．また，環境学や生物学で扱う個体の機能要素は必ずしも汎用プログラミング言語の機能要素と対応していない．例えば，LOGO のタートルを生物の個体のメタファーとして利用する場合，生物の変態による個体の行動の変容をシミュレーションに反映させるためには，ユーザプログラムによりその行動の変容の仕組みを実装する必要がある．IBM を構成するためには，プログラミング言語と環境学，生物学の間のギャップを埋める必要がある．

ABMS を科学研究に応用する場合，そのモデルおよびシミュレーション結果が，他の科学者に理解され，その特性が検証され，再現可能であることが望ましい．本研究では，生物学研究における個体群動態を ABMS の適用対象として，モデリング言語およびシミュレーション環境を一種のドメイン特化言語 (Domain Specific Language, DSL) として設計し実現した．DSL を設計実現する上で，プログラミング言語の構文定義や意味定義，実行系の設計や実装については多くの技術が確立されている．本研究では，特に汎用のプログラミング言語および処理系にはない，ABMS に特化した要求に注目して，モデリング言語およびシミュレーション環境を設計・実装した．

以後，第 2 節では，個体群動態の研究に利用されてきた他の ABMS ツールを示し，本研究と比較する．第 3 節では，個体群動態のシミュレーションに求められる要求

を議論し，第 4 節では本研究で開発しているシミュレーション環境 Re:Mobidyc で要求をどのように満たしたかを説明する．第 5 節で Re:Mobidyc 上のシミュレーションの例を紹介し，第 6 節でまとめと今後の課題を示す．

2. 関連研究

2.1. NetLogo

NetLogo [2] は Java で実装された LOGO ベースのマルチエージェントモデリング言語であり，自然環境や社会での現象のモデリングに広く利用されている．NetLogo は移動可能なタートルと空間を格子状に分割したパッチの，2 種類のエージェントを扱い，それぞれのエージェントを並行動作させる．各エージェントの動作は命令的な手続きにより定義される．

NetLogo は個体群動態にも適用可能であるが，生活史段階など個体群動態に必要な概念が言語機能に反映されていないため，例えば各エージェントの生活史段階を属性として保持して，各手続きの中でその生活史段階に対する条件分岐によって，各段階での振る舞いを記述するなどの工夫をする必要がある．また，各エージェントの行動の記述は命令型プログラミングそのものであり，モデリングにはプログラミング特有のスキルが要求される．Re:Mobidyc は個体群動態のシミュレーションに特化し，プログラミング特有のスキル要求を排除することを設計指針としている点で異なる．

2.2. Cormas

Cormas [4] は Smalltalk で実装されたマルチエージェントモデリング言語であり，共有資源管理のための住民参加型の意思決定の支援を目的として設計された．シミュレーションによる理解共有を促進するために，Cormas のシミュレーション環境は対話性および可視化を大きな特徴として備えている．

Cormas は，移動可能な社会的エージェント，および，空間を格子状に分割した空間的エージェント，さまざまな仲介を行う受動的エージェントの 3 種類のエージェントによりシミュレーションモデルを構成する．各種エージェントは Smalltalk のサブクラスとして定義され，それぞれのエージェントの行動は Smalltalk ベースの内部 DSL により定義される．内部 DSL により統計関数など多くの機能が提供されるが，モデリングにおけるエージェントの行動は基本的に Smalltalk のメソッドとして記述される．したがって，モデリングには Smalltalk プログラミングのスキルが要求される．Re:Mobidyc は，Cormas と同じく Smalltalk 上に構築されているが，エージェントの行動を記述する DSL を外部 DSL として，構文や言語機能について個体群動体論特有の概念を取り入れている点が異なる．

2.3. Mobidyc

Mobidyc [3] は個体群動態のシミュレーションを目的に Smalltalk 上に開発されたマルチエージェントモデリング環境であり，生活史段階を言語の基本要素として表現可能なモデリング言語を備えている．Mobidyc では，移動可能な個体を表す Animat，空間を格子状に分割した Cell，時系列データの読み込みや記録をおこなう NonLocatedAgent の 3 種類のエージェントでシミュレーションモデルを構成する．Animat の行動は，他の Animat や Cell との相互干渉を行う Task のリストとして定義される．Eat や Move 等の多くの生物種の個体に共通した行動が Task として組み込まれているほか，MathScript と呼ばれる DSL によってユーザ定義することも可能である．ユーザーは，シミュレーションに用いる生物種のそれぞれについて，卵，幼生，成体など複数の生活史段階を定義し，生活史段階ごとに Task のリストを定義する．

Re:Mobidyc は Mobidyc の設計指針を受け継いで，個体群動態に必要な言語機能やモデリング環境を実現した上で，スケーラビリティおよび再現性，検証可能性の向上を図り設計された．

3. 個体群動態シミュレーションに求められる特性

本節では，現在おこなわれている個体群動態でのシミュレーションの利用状況から，個体群動態のためのシミュレーションに求められる特性を挙げる．

R1：生活史段階の表現

個体群動態において重要な概念が，生活史段階である．生活史段階とは，個体の一生のなかで生態の変化を捉えたもので，卵生の動物では大きく分けて卵，幼生，幼体，未成体，成体などの段階を持ち，種によってそれぞれより細分化された段階を持つ．図 1 に動物性プランクトンの一種であるカイアシ類 Eurytemora affinis の生活史段

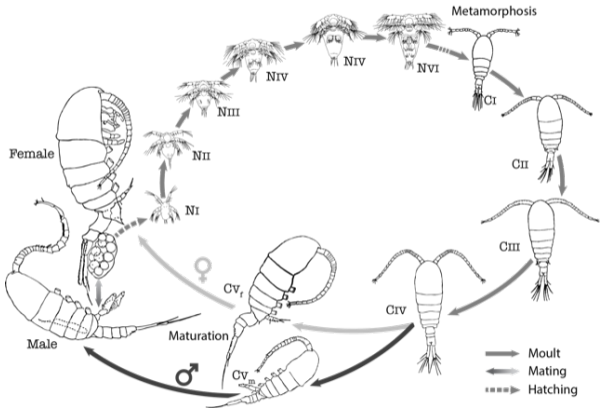


図 1. 生活史段階の例

階を示す．卵から孵化してノープリウス幼生 (N_I〜N_{VI}) の各生活史段階を脱皮により順に進み，N_{VI} からコペポディド幼生 C_I に変態し，ふたたび脱皮により順に C_{II} から C_V に脱皮し，C_V では雌雄に分化する．

個体群動態のシミュレーションでは，大量の個体の行動をモデリングしコンピュータ上でシミュレーションすることで，群れの個体数の分布や経時変化を予測する．個体の行動特性は生活史段階により大きく異なるため，個体の行動モデルは生物種ごとではなく，生活史段階ごとに定義される．生活史段階ごとに，個体の移動，捕食，代謝，交配，生殖などの行動の定義に加え，次の生活史段階への遷移（脱皮または変態）が定義される．

R2：個体および環境の相互作用

個体の行動をモデリングするために，個体と他の個体および環境との相互作用を扱うことが必要である．例えば，動物の行動には，近接する餌を捕食するなど，他の個体との間の相互作用として行動をモデリングする．また，個体の行動は，水温，水の透明度，酸素濃度，二酸化炭素濃度，餌など，位置に依存した局所的な環境が持つ量により影響を受けると同時に，個体が二酸化炭素を排出するなど環境に影響を与えることから，個体と局所的環境の間の相互作用を表現する必要がある．既存のシミュレーション環境の多くでは，パッチまたはセルと呼ばれる小区画をエージェントとして扱い，個体をあらわすエージェントとの間の相互作用が記述可能である．

R3：プログラミングスキル要求の軽減

個体群動態シミュレーションの目的は生物学や環境学の研究であり，求められるプログラミング言語への習熟が可能な限り低いことが望ましい．命令的なプログラミングは，破壊代入や制御構造や再帰呼び出しなど，プログラミング特有の概念を習得することが求められるため，その利用は最低限に留めることが望ましい．

R4：分析可能性，再現性

シミュレーションを仮説検証に用いる場合，シミュレーションするモデルが，検証対象である仮説に忠実であることが求められる．モデリング言語が LOGO 等の命令型言語である場合，個体の行動を変数への代入やループによる繰り返しによる手続きとして記述することになるが，その手続きが仮説を忠実にプログラム化したものであることを検証することは，一般に容易ではない．個体の行動を記述するモデリング言語は，数理的な分析に適していることが求められる．シミュレーションを科学研究で用いるため，シミュレーション上で発生した事象は，同じモデル同じ初期条件から再現できることが望ましい．

R5：スケーラビリティ

シミュレーションは一定の時間幅を決めた離散時間により，状態の更新を行う．筆者の 1 人はカイアシ類の個体群動態の研究で Mobidyc 上のシミュレーションを行なっている．これまでは，最長 5 年分のシミュレーションを 6 時間刻みで，すなわち 7,300 ステップを実行し，最大 300,000 個体のシミュレーションを行なった．1 個体あたり 64 ビットの数値データを格納する領域を 10 スロット保持すると，300,000 個体を 7,300 ステップ実行すると，163G バイト強のデータ量になる．

今後，シミュレーションによる個体群動態論研究を進めるためには，より長いシミュレーション期間，より短い刻み幅，より多い個体数を対象にしたシミュレーションが求められることが予想される．Re:Mobidyc は，より多くのステップ数や，より多くの個体数を求める要請に応えることが期待されている．

4. Re:Mobidyc の概要

筆者らは現在，Re:Mobidyc の実装を進めている．Re:Mobidyc は Pharo Smalltalk [1] 上に構築され，github レポジトリ [2] 上にオープンソースソフトウェアとして公開されている．開発はまだ完了していないが，既にいくつかの簡単なシミュレーションが実行可能になっている．本節では，Re:Mobidyc の設計および実装にあたって基準にした重要な指針を示す．

[1] https://pharo.org/
[2] https://github.com/tomooda/ReMobidyc/

4.1. モデリング言語

Re:Mobidyc はマルチエージェントシミュレーション環境であり，シミュレーションモデルを構成する基本単位は，動物の個体を表す Animat，環境を格子状に分割した Cell，環境全体を表す World の 3 種類のエージェントである．各エージェントはその状態を特定するための属性と，取り得る行動を記述したタスクにより定義される．本論文では，エージェントの種類とインスタンスを区別するために，後者を指す時には明示的にインスタンスと呼ぶ．また，コンピュータ操作環境と，シミュレーションモデル中の空間を指す環境を区別するために，本論文ではコンピュータ操作環境をシミュレーション環境またはモデリング環境と呼び，シミュレーションモデル中の個体の外界としての環境を単に環境と呼ぶ．

Animat は具体的な対象の種類ごとに定義される．〈**R1：生活史段階の表現**〉として挙げた通り，個体群動態では脱皮や変態によって生活史段階を遷移していくことが重要である．Re:Mobidyc ではそれぞれの種類の Animat は，それぞれ 1 つの生活史段階を表現する．すなわち，Re:Mobidyc では，1 つの個体は，その個体の誕生から死までのシミュレーション時刻を連続的にカバーし，かつ，同一シミュレーション時刻に重複しない，いくつかの Animat のインスタンスを継いで接続したものとして表現される．図 1 のカイアシ類 Eurytemora affinis の雌の個体の場合には，`Egg`，`N1`〜`N6`，`C1`〜`C4`，`C5f` のそれぞれのインスタンスを連結したものになる．

Cell は環境をグリッド状に分割した区画をあらわす．現在は矩形分割のみを実装しているが，六角形分割にも対応する予定である．Cell は 1 つのシミュレーションモデルに 1 種類のみ定義され，`Cell` という識別子で参照される．World は環境全体をあらわすエージェントであり，Cell と同様に，1 つのシミュレーションモデルに 1 種類のみ定義され，`World` として参照される．

図 2 に Animat の定義例を示す．`Goat` は，その属性として `x`，`y`，`blood_sugar`，タスクとして `respire`，`move`，`graze` の 3 つを遂行する Animat として定義されている．ただし，`x` および `y` は Animat に暗黙に定義されている属性である．

属性の定義

Re:Mobidyc では，各インスタンスが保持する属性は数値のみを扱い，全ての属性宣言に単位を宣言することを義務付けている．宣言された単位は後述のタスク定義の計算式の中で次元の整合性の検査に使われる．属性の単位の宣言は，自然科学研究で用いられる単位を使って計算式の整合性を検査してモデルの誤りを発見することで，〈**R3**: プログラミングスキル要求の軽減〉の実現に貢献することを目的に導入した．また，計算式中の数値リテラルや外部ファイルから読み込まれる数値データの単位の違いによる誤りを防ぐことを目的に，数値リテラルおよび外部ファイル中の数値データには単位を明示することを要求する．Re:Mobidyc 実行系は数値を全て国際単位系 (SI) に変換して計算を行う．属性値はメモリ中には SI に変換された値で保持され，属性宣言で与えられた単位に変換して表示する．

```
Goat is Animat with
    blood_sugar [kcal].

Goat respire namely lose where
    my amount -> my blood_sugar
    my loss ->
        min(my amount, 2000 [kcal/day] * Δtime).

Goat move where
    my heading -> direction neighbor's grass
    my speed -> 1[km/day].

Goat graze namely collect where
    my thing -> my blood_sugar
    here's thing -> here's grass
    my greed -> min(
        10000[kcal]-my blood_sugar,
        3000[kcal/day] * delta time,
        here's grass).
```

図 2. Animat 定義の例

図 2 の中で，エージェント `Goat` の属性は以下の部分で宣言されている．

```
Goat is Animat with
    blood_sugar [kcal].
```

属性 `blood_sugar` が単位 `kcal` 付きで宣言されているが，Animat は必ず空間中の位置を持つため，x 座標を表す `x [m]` および y 座標を表す `y [m]` が暗黙に宣言されている．したがって，エージェント `Goat` には `x`，`y`，`blood_sugar` の 3 つの属性が宣言されている．`Goat` の各インスタンスは，`x` 属性および `y` 属性として x 座標および y 座標をメートル単位で，`blood_sugar` 属性として血中に蓄えられたエネルギーを kcal と同次元の SI 単位であるジュール単位で格納する．

一般にマルチエージェントシミュレーションでの各エージェントの属性値の更新には，非同期的な更新を行うも

のと同期的な更新を行うものがある．非同期的な更新では，タスクの実行による属性値の更新は直ちに反映され，同一ステップでの他のエージェントあるいは他のタスクの実行に影響を与える．同期的な更新では，各エージェントタスクによる属性値の変更は即座には反映されず，シミュレーション時刻の 1 ステップ終了時に全インスタンスの全属性値が一斉に更新される．非同期的な更新は，手続き型言語をベースにしたシミュレーション環境や分散環境でのシミュレーション環境に多くみられる方式で，属性値更新のオーバーヘッドが少なく効率的に処理できる利点がある一方，エージェントおよびタスクの実行順序によってシミュレーション結果が影響されるという欠点を持つ．Re:Mobidyc では，〈 **R4：分析可能性，再現性** 〉を目的に，同期的更新を採用した．同期的更新と擬似乱数のシードの管理によって，Re:Mobidyc では任意のシミュレーション時刻での状態から次の状態の再現性を確保している．タスクを実行するインスタンスの順序やタスクの順序に関わらず，同じ属性に対する同じ値への更新が再現される．また，各シミュレーション時刻での全てのインスタンスの属性の値は上書きされずに記録される．したがって，Re:Mobidyc 上でのシミュレーションは，初期状態から開始したそれぞれのシミュレーション時刻における全属性値が記録され，かつ，再現されることから，シミュレーションモデル，初期状態，シミュレーション結果の間の整合性を各シミュレーション時刻ごとに検証することができる．

タスク定義

Re:Mobidyc では，〈 **R2：個体および環境の相互作用** 〉はタスクにより表現する．すなわち，各エージェントの移動，捕食，排泄，代謝，生殖などの行動は，タスクとして定義される．タスクは，タスクを実行するインスタンスを含む 1 つ以上のインスタンスの属性を変化させることで，エージェント間の相互作用を定義する．〈**R3：プログラミングスキル要求の軽減** 〉および〈 **R4：分析可能性，再現性** 〉のために，タスクによる属性値の変更は，各シミュレーション時刻での処理の最後に一括して同期的に行われる．各シミュレーション時刻において，それぞれの属性の値は一定である．

タスクの定義は，タスク名，相手エージェント，事前条件，属性更新定義のリスト，局所定義のリストからなる．ただし，相手エージェント，事前条件，局所定義は省略可能である．図 3 に，標準ライブラリに組み込まれているタスク `move` の定義のソースコードを示す．1 行目では，タスク名 `move` が宣言され，2 行目と 3 行目で，それぞれ属性値 `x` および `y` の更新が定義されている．4 行目は以降が局所定義であることを示し，5 行目と 6 行目でそれぞれ局所定数 `theta` および `r` が定義されている．

```
to move is
    my d/dt x' = cos(theta) * r
    my d/dt y' = sin(theta) * r
where
    theta = my heading
    r = my speed.
```

図 3. タスク定義の例

表 1. 属性値更新の左辺値の種類と構文

左辺値種別	構文
更新値	‘my’, 属性名, ‘ ’’ エージェント参照,‘ ’s’, 属性名, ‘ ’’
差分値	‘my delta’, 属性名, ‘ ’’ ‘my Δ’, 属性名, ‘” エージェント参照,‘ ’s delta’, 属性名, ‘ ’’ エージェント参照,‘ ’s Δ’, 属性名, ‘ ’’
微分値	‘my d/dt’, 属性名, ‘ ’’ エージェント参照,‘ ’s d/dt’, 属性名, ‘ ’’

属性値の更新定義には，(1) 値の直接代入，(2) 値の差分値，(3) 値の微分値の 3 種類の定義がある．表 1 に 3 種類の更新定義の左辺値をまとめる．いずれの形式でも，左辺値は属性値への参照と区別するために，末尾に *'* が付けられる．右辺値には，左辺値と互換性のある単位を持つ任意の数式を記述することができる．表 2 に主な数式を示す．

タスクによる属性値の変更は，直ちには反映されず，各シミュレーション時刻での処理の最後に一括して同期的に行われる．値の直接代入は，右辺を評価した結果得られた数値を次のシミュレーション時刻におけるそのインスタンスの属性の値になる．1 つのインスタンスの同一属性には，1 シミュレーション時刻につき 1 回しか直接代入できない．値の差分値による更新定義は，右辺を評価した結果得られた数値を，次のシミュレーション時刻におけるそのインスタンスの属性の値の増分とする．値の微分値による更新定義は右辺を評価した結果得られた数値とシミュレーション時刻の刻み幅の積を，次のシミュレーション時刻におけるそのインスタンスの属性の

表 2. 主な表現式の種類と構文

数式の種類	構文
リテラル	数値, '[', 単位, ']'
属性参照	'my', 属性名 エージェント参照, ' 's', 属性名
差分参照	'my delta', 属性名 'my Δ', 属性名 エージェント参照, ' 's' delta, 属性名 エージェント参照, ' 's Δ', 属性名
微分参照	'my d/dt', 属性名 エージェント参照, ' 's d/dt', 属性名
算術演算	数式 ('+' | '-' | '*' | '/' | '^') 数式
負数	'-', 数式
関数適用	関数名, '(', 数式, {',', 数式 }, ')'
場合分け	数式, 'if', 条件式, ',', { 数式, 'if', 条件式, ','}, 数式
擬似乱数	'uniform', リテラル, 'to', リテラル
方向	'direction', エージェント参照
属性勾配	'direction neighbor 's', Cell 属性名
距離	'distance', エージェント参照
時刻	'time'
刻み幅	(' Δ ' | 'delta'), 'time'

値の増分とする．差分値による更新定義および微分値による更新定義は，1 つのインスタンスの同一属性に 1 シミュレーション時刻につき複数回の更新定義することができる．シミュレーション時刻 t から $t+1$ に移行する時，インスタンス i の属性値 pi は以下の漸化式に従って更新される．

$$pi_{(t+1)} = pi'_{(t)} + \Sigma_{jk}(\Delta pi'_{jk_{(t)}}) + \Sigma_{jk}(d/dt\, pi'_{jk_{(t)}} * \Delta\ time)$$

ただし，$pi_{(t+1)}$ は時刻 $t+1$ におけるインスタンス i の属性 p の値である．$pi'_{(t)}$ は，時刻 t にインスタンス i の属性 p への直接代入が行われた場合にはその値，行われなかった場合には時刻 t における当該属性の値とする．また，$\Delta pi'_{jk_{(t)}}$ は時刻 t に実行されたインスタンス j のタスク k による差分更新の右辺値，$d/dtpi'_{jk_{(t)}}$) は時刻 t に実行されたインスタンス j のタスク k による微分更新の右辺値，$\Delta time$ はシミュレーション時刻の刻み幅とする．

キーワード `where` 以下では，局所定義を列挙する．局所定義の右辺値には，属性更新定義の右辺値と同様に，数値リテラルや属性への参照を使った数式を記述する．図 3 の記述では，`theta` および `r` の 2 つの局所定義がされ，それぞれ，タスクを実行しているインスタンスの `heading` 属性と `speed` 属性の値を取る．

置換によるタスクの適応

Re:Mobidyc では，〈**R3**: プログラミングスキル要求の軽減〉を目的として，ライブラリ化された抽象的なタスクを，構文的置換により特化させることで，具体的なタスクを定義する仕組みを実装した．図 2 の中で，エージェント `Goat` のタスクとして `move` が，以下の記述によって標準ライブラリの組み込みタスク `move` の特化として定義されている．

```
Goat move where
    my heading -> direction neighbor's grass
    my speed -> 1[km/day].
```

エージェント `Goat` のタスク `move` は，汎用の組み込みタスク `move` の `my heading` を `direction neighbor's grass` に，`my speed` を `1[km/day]` に置き換えたものとして定義される．すなわち，下記のタスク定義と等しい．

```
to move is
    my d/dt x' = cos(theta) * r
    my d/dt y' = sin(theta) * r
where
    theta = direction neighbor's grass
    r = 1[km/day].
```

ここで，`direction neighbor's grass` は Re:Mobidyc の組み込み命令で，実行するエージェントが位置している Cell インスタンスの周辺の，属性 `grass` の勾配の方向を返す．したがって，このタスクは，インスタンスを `Cell` の `grass` 属性の値がより大きな方向に向けて移動させる．移動量は，1 日かけて 1km 移動する速度であり，仮にシミュレーション時刻の刻み幅が 6 時間の場合，1 ステップあたり `250[m]` 移動させる．

置換によりタスクを特化させる時，特化によりタスク名をより具体的な名前が適切になることがある．Re:Mobidyc では，置換対象となるタスク名に続いて `namely` 句により新しいタスク名を宣言することができる．図 2 の `Goat` エージェントの定義では，`respire` タスクを以下のように `lose` タスクから特化させている．

```
Goat respire namely lose where
    my amount -> my blood_sugar
    my loss ->
```

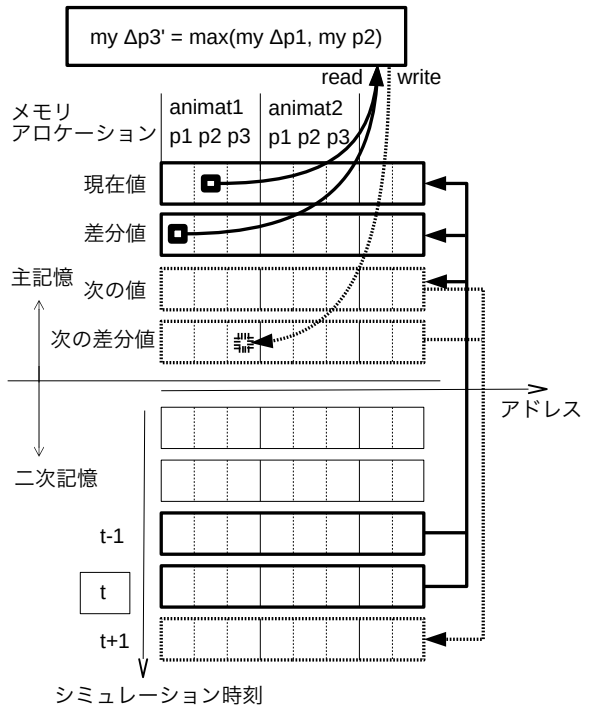


図 4. 時系列メモリモデルの概要

```
min(my amount, 2000 [kcal/day] * Δtime).
```

4.2. 時系列数値データに特化したメモリモデル

Cormas 等の汎用プログラミング言語をベースにした MAS 環境の多くは，ユーザが記録する属性を指定して，ファイル等にデータを出力する機能を持つ．それらの MAS 環境では，各インスタンスはヒープメモリ上にアロケートされ，シミュレーション時刻における属性値をメモリ上に持つ．Re:Mobidyc は，〈**R4：分析可能性，再現性**〉のために，ユーザが指定した属性だけでなく，初期状態からの各シミュレーション時刻における全ての属性値を記録する．全ての属性値の記録は Re:Mobidyc の設計の基盤である，時系列メモリモデルが行う．

図 4 に時系列メモリモデルの概要を示す．いくつかの，水平方向に区分けされた横長の矩形が並んでいる．これらの矩形はメモリイメージを表しており，複数のメモリイメージが 1 つのアドレス空間上にオーバーレイされる形で統合されている．このメモリモデルでのアドレス空間は，この横長の矩形を水平方向に区分けしたスロットの整数インデックスである．

中央の水平線より上の 4 つのメモリイメージは，主記憶上に保持された，現在実行中のシミュレーション時刻における各属性値の読み出しや更新のための領域である．最上段の現在値とラベルが付けられたメモリイメージは，現在シミュレーションしている時刻での各属性の値を格納している．2 段目の差分値とラベルが付けられたメモリイメージは，現在シミュレーションしている時刻での各属性の，前のシミュレーション時刻での値からの差分値を格納している．これら 2 つのメモリイメージは，インタプリタがタスクの属性更新定義の右辺値を計算するために利用され，インタプリタからは read-only である．例えば，このアドレス空間上に，2 つのインスタンス `animat1` と `animat2` について，それぞれ属性数分の連続領域がアロケートされ，それぞれ 3 つの属性 `p1`，`p2`，`p3` を持っていることを仮定する．ここで，インタプリタがインスタンス `animat1` のタスクとして属性地更新定義 `my Δp3′ = max(my Δp1, my p2)` を評価実行する時，まずは右辺値を計算する．`my Δp1` の評価実行では，上から 2 番目の差分値のメモリイメージから，インスタンス `animat1` の属性 `p1` がアロケートされている左から 1 番目のスロットの値を読み込む．続いて，`my p2` の評価実行では，上から 1 番目の現在値のメモリイメージから，インスタンス `animat1` の属性 `p2` がアロケートされている左から 2 番目のスロットの値を読み込む．2 つの値のうち，より大きな値が右辺値の値となる．

属性更新定義の左辺値への更新には，3 段目と 4 段目のメモリイメージが使われる．3 段目と 4 段目のメモリイメージはそれぞれ，直接値による属性更新定義での更新値と差分値および微分値による属性更新定義での更新値を保存するための領域である．例えば，インタプリタがインスタンス `animat1` のタスク中の `my Δp3′ = max(my Δp1, my p2)` を評価実行する場合，4 段目の次の差分値とラベル付けられたメモリイメージの，`animat1` の `p3` がアロケートされた左から 3 番目のスロットに，右辺値の評価結果の数値を加算して上書きする．

中央の水平線より下のメモリイメージは，二次記憶に確保された，各シミュレーション時刻での全属性値を保存するメモリイメージである．シミュレーション開始時には，まず，初期状態 ($t = 0$) でのメモリイメージが初期化される．各シミュレーション時刻 (`t`) の処理の開始時と終了時に，時系列メモリモデルはインタプリタを利

用せずに主記憶上の 4 つのメモリイメージの更新処理を行う．この更新処理は，インタプリタとは独立して，時系列メモリモデルが内部処理として行う．

シミュレーション時刻 (t) の処理の開始時の更新処理として，時系列メモリモデルは主記憶中の現在値，差分値，次の値，次の差分値の 4 つのメモリイメージを初期化する．初期化には，まずその時刻でのメモリイメージを現在値メモリイメージと次の値メモリイメージに転送し，前時刻でのメモリイメージとの差分を差分値メモリイメージの全スロットに書き込む．また，次の差分値メモリイメージの全スロットを 0 に初期化する．以上の初期化により，タスクの評価実行に必要な主記憶上のメモリイメージが準備され，インタプリタは各インスタンスでの各タスクを評価実行する．評価実行により，次の値メモリイメージと次の差分値メモリイメージが更新される．シミュレーション時刻 (t) の処理の終了時の更新処理として，時系列メモリモデルは，次の値メモリイメージの各スロットと，対応する次の差分値メモリイメージのスロットの和を，二次記憶上のシミュレーション時刻 $t+1$ のメモリイメージに格納する．

インタプリタは時系列メモリモデルを通して主記憶上の 4 つのメモリスロットに対してのみ読み出しと書き込みを行い，二次記憶上の各シミュレーション時刻でのメモリイメージを扱う必要はない．この機構により，Re:Mobidyc は〈**R5：スケーラビリティ**〉について，シミュレーション時刻に関するスケーラビリティを確保している．また，時系列メモリモデルはファイルシステムなど二次記憶の種類によって，インタプリタに対して互換性を確保した複数の実装が可能である．現在はメインメモリに保存するものとファイルシステム上に保存するものが実装されている．今後，DBMS などに向けた時系列メモリモデルを実装していく予定である．

5. 例題：牧草地と山羊

Re:Mobidyc は現在も開発が続けられているが，現時点で簡単なモデルを構築しシミュレーションを実行することができる．本節では，動物と環境の相互作用を表現した簡単な例として，牧草地と山羊のモデルを Re:Mobidyc 上で実行する例を紹介する．

牧草地と山羊は，エネルギーの供給と消費のバランスを再現するモデルである．牧草地は太陽からのエネルギーを，光合成により牧草の栄養価に変換して蓄積する．山羊は，牧草地の中をより豊かな牧草地へ移動しながら，牧草を食べて，牧草の栄養価を血中糖という形でエネルギーを獲得する．ただし，山羊はその代謝活動によりエネルギーを失う．すなわち，牧草地と山羊が持続可能なエネルギー系を構成するためには，太陽から牧草地に供給されたエネルギーと代謝がバランスする必要があり，その間のバッファとなるのが牧草の栄養価と血中糖である．

```
Cell with
        grass [kcal].

Cell grow_grass namely gain_upto where
    my thing -> here's grass
    my amount -> 2000 [kcal/day Δ]*time
    my maximum -> 1000000 [kcal].
```

図 5. Cell 定義の例

牧草地と山羊のモデルでは，2 種類のエージェントを定義する．牧草地の小区画を表す `Cell` と山羊の個体を表す `Goat` である．このモデルでの `Cell` の定義を図 5 に示す．`Goat` の定義は図 2 で示された通りである．

`Cell` のタスク `grow_grass` は，太陽から供給されたエネルギーを牧草の栄養価に変換する相互作用を表す．`grow_grass` は，当該区画の牧草の量を毎日 2000kcal のペースで最大 1000000kcal まで成長させる．このタスク定義でベースにしている組み込みタスク `grain_upto` は，インスタンスの 1 つの属性値を，上限値付きで一定量増加させるための抽象タスクであり，`my thing`，`my amount`，`my maximum` の 3 つの参照を具体化することで，牧草の栄養価を獲得する具体的なタスクを定義する．この `grow_grass` タスクを定義している GUI のスクリーンショットを図 6 に示す．タスク定義 GUI は上段と下段に分かれており，上段でタスクの原型となる既存の抽象タスクを選択し，下段でその抽象タスクに対する置換操作によって，特化したタスクを定義する．

`Goat` のタスクも同様にして定義することができる．特に，タスク `graze` は，`Cell` との相互作用によって，牧草の栄養価を山羊の血中糖に変換するタスクであるが，これは組み込みタスクである `collect` をベースにしている．`collect` は Animat が位置する `Cell` から，ある量を獲得し，`Cell` から同量を減少させる抽象タスクである．Re:Mobidyc では，同一インスタンス内での量の移動，インスタンス間での量の移動などを，抽象タスクとして定義して，標準ライブラリの中で提供している．

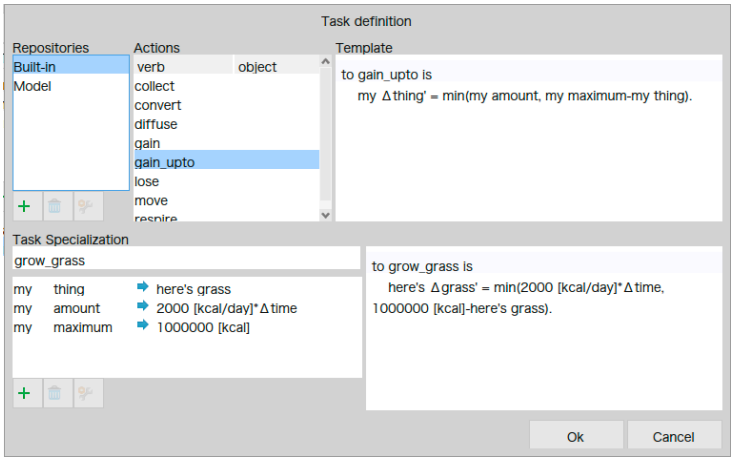


図 6. タスク定義 GUI のスクリーンショット

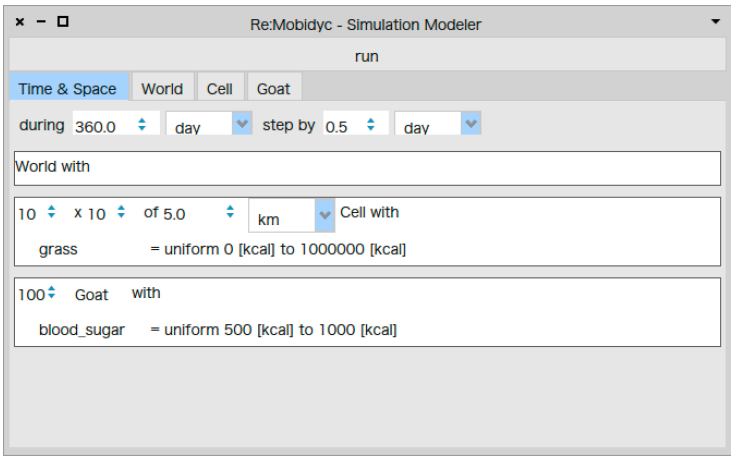


図 7. シミュレーション条件設定 GUI のスクリーンショット

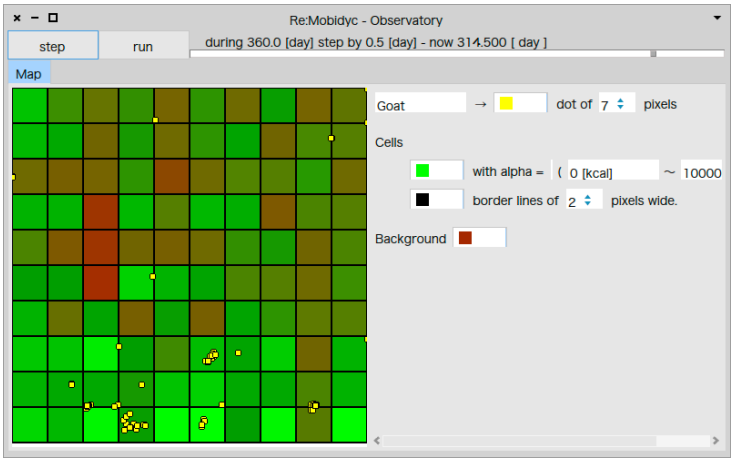


図 8. シミュレーション環境のスクリーンショット

ユーザーはこれらの基礎的な抽象タスクを利用して，さまざまなエージェント間の相互作用を表現することができる．

図 6 の上段左側の Repositories とあるリストには，Built-in および Model の 2 項目があり，組み込みタスクのレポジトリを表す Built-in が選択されている．上段中央の Actions には，左側で選択されたレポジトリに格納されているタスクの一覧が表示され，特化のベースとなる抽象タスクとして `gain_upto` が選択されている．その定義のソースが上段右側に表示される．下段では，左側で 3 つの置換が定義され，その置換を適用した結果が右側に表示される．

シミュレーションの期間やセル分割，Animat の初期化について設定する GUI のスクリーンショットを図 7 に示す．during で始まる行で，シミュレーション期間を単位付きで指定されている．ここでは単位として `day` が使われているが，`hour`，`second` など他の時間単位を選択することもできる．`Cell` は 5km 四方の矩形が 10*10 個の計 100 個設定され，その属性 `grass` の初期値を 0kcal から 1000000kcal までの一様分布から設定する．また，エージェント `Goat` のインスタンスを 100 個，初期状態として生成し，その属性 `blood_sugar` を 500kcal から 1000kcal の一様分布から設定する．最上段の run ボタンを押すと，図 8 にあるように時間経過とともにモデルの状況が可視化される．

6. まとめと今後の展望

筆者らは，個体群動態のシミュレーションを目的としたマルチエージェントシミュレーション環境 Re:Mobidyc について，個体群動態の科学的モデリングに求められる特性を起点として，モデリング言語および実行系を設計し実装してきた．特に，シミュレーション環境の基礎となるモデリング言語では，生活史段階を言語機能に取り入れることで，既存の汎用プログラミング言語をベースにしたシミュレーション環境では直接的な表現が困難なモデルを，ユーザである環境学や生物学の研究者がその研究領域の語彙でモデリングすることを可能にした．また，その実行系の基礎となるメモリモデルでは，時系列の数値データに特化したメモリモデルを設計し実装することで，シミュレーションの計算過程を検証可能な形で記録に残すことを実現した．

Re:Mobidyc の開発はまだ初期段階であり，これから追加実装すべき機能として，研究チームによるモデリングおよびシミュレーションを想定した，ネットワーク機能が挙げられる．1 つは，マルチユーザに対応したバージョン管理機能付きのモデルレポジトリである．シミュレーションモデル全体だけでなく，個別のエージェントの定義，抽象タスクの定義，シミュレーション結果など，研究チーム内での共有を可能にする機能が必要である．もう 1 つは，複数のシミュレーション実行をモニタリングするサーバである．LAN 上に Re:Mobidyc の実行状況を監視・表示するためのモニタリングサーバを設置し，複数のコンピュータ上で Re:Mobidyc が実行されている時，どのコンピュータ上でどのモデルのシミュレーションが実行されているかを把握できるようにすることで，研究チームのメンバーがコンピュータ資源を有効に利用できるようにする予定である．

参考文献

[1] Sameera Abar, et al. ”Agent Based Modelling and Simulation tools: A review of the state-of-art software”, *Computer Science Review*, vol 24, pp. 13-33, 2017

[2] Seth Tisue and Uri Wilensky. ”Netlogo: A simple environment for modeling complexity”, *Proceedings of the 5th International Conference on Complex Systems (ICCS’04)*, pp. 16-21, 2004.

[3] Vincent Ginot and Christophe Le Page. ”Mobidyc, a generic multi-agents simulator for modeling populations dynamic”, *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial In Telligence and Expert Systems: Tasks and Methods in Applied Artificial Intelligence (IEA/AIE 1998)*, pp. 805-814, 1998.

[4] Pierre Bommel, et al. ”Cormas: an agent-based simulation platform for coupling human decisions with computerized dynamics”, *Simulation and gaming in the network society*, Translational Systems Sciences, vol 9, pp. 387-410, 2016.

スマートフォンにおけるフリック操作による画面ロック解除手法の提案

喜多 義弘
長崎県立大学
kita@sun.ac.jp

山下 湧介
長崎県立大学
bs217040@sun.ac.jp

要旨

スマートフォンにおいて，画面ロックを解除するための本人判定手法として，パスワード入力，暗唱番号入力，指紋認証，顔認証が用いられている．これらの手法は，ユーザにとって扱いやすく，かつ一定の安全性を保つことができる．しかし，パスワード入力や暗証番号入力においては覗き見攻撃への耐性が低いという課題が，一方，指紋認証や顔認証においては生体情報の再登録が容易ではないという課題がそれぞれある．そのため，解除に必要なこれらの情報が漏洩した場合，安全性が損なわれてしまうことが懸念される．そこで本研究では，これらの課題へ柔軟に対応するために，フリック操作による画面ロック解除手法を提案する．具体的には，パスワードとしての入力文字と，生体認証としてのフリック操作をそれぞれ判定することで，ユーザ本人かを判定する手法である．これにより，パスワードが漏洩してもフリック操作の生体情報で解除を防ぐことができ，漏洩後にパスワードを変更するだけでフリック操作の生体情報も更新することが容易にできる．

1. はじめに

スマートフォンには，タッチスクリーンの誤操作や他人による操作を防ぐことを目的として，画面ロック機能が搭載されている．画面ロックとは，タッチスクリーンでの操作を受け付けなくする機能であり，その解除のための本人判定手法としてパスワード入力，暗証番号入力，パターン入力，指紋認証，または顔認証が用いられている．

これらの手法は，入力が容易であるため，ユーザにとって扱いやすい．しかし，各手法には安全性に関する課題がある．例えば，パスワード入力，暗証番号入力，およびパターン入力は，解除中に後ろから画面を覗き見られ（この攻撃を以降，覗き見攻撃とする），入力情報が漏洩する．一方，指紋認証および顔認証は，覗き見攻撃への耐性はあるが，指紋や顔の生体情報が何らかの形で漏れた場合，漏れた情報を用いて画面ロックが解除されてしまう．実際に指が写った写真から指紋の情報を抽出し，それを用いて指紋認証が成功している例を報告した研究 [1] がある．そして，指紋認証や顔認証などの身体的な部分を用いた生体認証の場合，その生体情報の再登録には時間が掛かることや，再登録の回数に制限があることなど，生体情報漏洩への柔軟な対応ができていない．

そこで本研究では，覗き見攻撃への対応と，生体認証における情報漏洩への柔軟な対応を備えるため，フリック操作による画面ロック解除手法を提案する．具体的には，パスワードとして任意の文字列をフリック操作にて入力し，さらにフリック操作から動きの特徴を抽出し，それを生体情報としてユーザ本人の識別に用いる．これにより，後ろから画面を覗き見られてパスワードが漏洩し，そのパスワードを入力されても，そのフリック操作入力の動きがユーザ本人であると判定されなければ，ロックが解除されることはない．そして，ユーザ本人が次に使用する際に前回他人が操作したことを提示することで，パスワードが漏洩したと気づき，その文字列をフ

リック操作の生体情報も併せて即座に変更することにより，漏洩への対応を柔軟にかつ迅速に行うことができる．

本論文では，この提案手法をスマートフォン上に実装し，その有用性について評価および考察を行う．

2. 関連研究

スマートフォンを対象とした覗き見攻撃への対策として，フリック操作やキーボード操作の動きを用いた本人識別が複数提案されている．

Shahzad ら [2] は，フリック操作によるジェスチャーを利用した画面ロック解除手法を提案している．この提案では，タッチスクリーン上をフリックしながら特定のジェスチャーを行うと，その動きから，指の速度，デバイスの加速度，ストローク時間などの特徴を抽出し，それらから本人を判定する．これらの特徴は本人特有のものであるため，他人から覗かれ，動きを真似されても再現することができず，ロックは解除されない．

伊藤ら [3] は，スマートフォンにおいて，母音ごとのフリック操作から特徴を抽出し，その母音ごとの特徴から本人を識別する手法を提案している．この手法では，1 クラスサポートベクターマシン [4]（One Class Support Vector Machine, 以降，OCSVM とする）を用いている．

OCSVM は，教師なし学習の機械学習法の 1 つであり，すべての学習データを 1 つのクラスにまとめ学習し，その境界を設けることにより，境界から外れたデータは別のクラスとして扱う．これにより，本人ではないデータが入力されたとしても，本人とは異なる別のクラスとして扱うことができる．本研究では，この OCSVM を本人の判別器として用いる．

泉ら [5] は，キーボードを用いたキーストローク認証をスマートフォンに応用することを提案している．この手法では，本人識別に最近傍決定測を用いており，未登録の文字列から特徴を抽出し，予め登録している本人を含む複数人の特徴と比較し，特徴量の差を絶対的な距離に置き換えた場合に，最も距離が近い人物を入力した者として判定している．この手法は，認証の際に入力する文字列は任意のものでよいため，認証時のユーザ負担が小さいという利点がある．しかし，その文字列での特徴と比較するためには，事前にある程度の文字列長を持つ文章を登録する必要があるため，登録時のユーザ負担が大きいことが課題である．

3. フリック操作による画面ロック解除手法の提案

3.1. 提案手法の概要

本研究では，覗き見攻撃や情報漏洩への対応を備えるため，パスワード方式と生体認証方式を組み合わせた，フリック操作による画面ロック解除手法について提案する．具体的には，フリック操作で入力した文字列をパスワードとし，同じ文字列が入力された際にフリック操作の特徴を抽出し，本人の特徴とを比較し本人識別を行う方法である．パスワード方式と生体認証方式を組み合わせることにより，各方式の利点を活かしてもう片方の欠点をフォローし合うことができる．なお，各方式の利点および欠点は以下のとおりである．

- パスワード方式
  - 利点：パスワードの変更が容易である．
  - 欠点：覗き見攻撃への耐性が低い．
- 生体認証方式
  - 利点：覗き見攻撃への耐性が高い．
  - 欠点：生体情報の再登録が容易ではない，または，再登録回数に制限がある．

図 1 に，本提案手法の処理の流れを示す．この手法は，登録フェーズとロック解除フェーズの 2 部で構成する．

登録フェーズは，パスワードとそれを入力する際のフリック操作の特徴をそれぞれ登録するフェーズである．まず，ユーザは任意の文字列をフリック操作にて入力し，登録する．次に，登録した文字列はパスワードとして，ストレージへ保存する．フリック操作の特徴は，OCSVM を用いて学習を行い，ユーザ本人の特徴としてマップを作成する．このマップはパスワードとは別にストレージへ保存する．

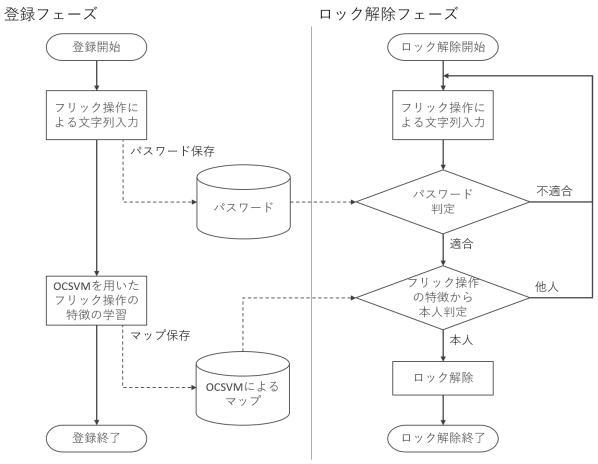


図 1: 本提案手法の各フェーズにおける処理の流れ

一方，ロック解除フェーズは，画面ロックを解除する際に登録時と同様のフリック操作を行い，画面ロックを解除するフェーズである．まず，ユーザは登録した文字列と同様の文字列を，登録時と同様のフリック操作で入力を行う．次にパスワードとして，入力した文字列が登録した文字列と同様かを判定する．この時点で入力した文字列が登録した文字列と異なっていれば，処理を中断し，文字列の入力待機に戻る．

入力した文字列が登録したものと同様であれば，文字列を入力した際のフリック操作の特徴から，ユーザ本人かの判定を行う．このとき，事前に作成していた OCSVM のマップを読み取り，そのマップ上に今回入力したフリック操作の特徴をプロットする．このマップからユーザ本人かの判定を行い，本人と判定されれば画面ロックを解除し，他人と判定されれば処理を中断し，文字列の入力待機に戻る．

3.2. フリック操作における特徴量の定義

フリック操作における特徴量として，以下を抽出する．

- 1 つのフリック操作にかかる時間（timePR）
- フリック操作の間隔時間（timeRP）
- フリック操作の挙動の長さ（lengthF）
- フリック操作の角度（angleF）

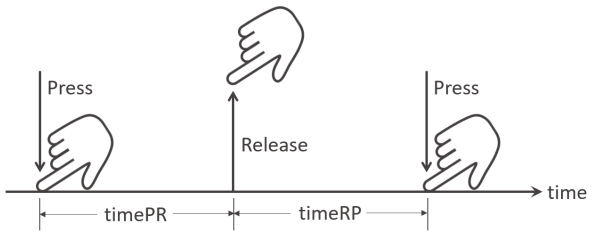


図 2: フリック操作時の指の動作における時間の定義

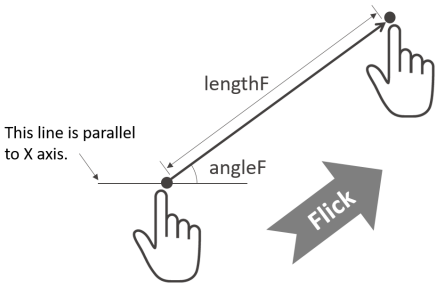


図 3: フリック操作における長さおよび角度の定義

以下では，各特徴量の定義およびその抽出方法について述べる．

図 2 に，フリック操作時の指の動作における時間の定義を，図 3 に，フリック操作における長さおよび角度の定義を，それぞれ示す．

図 2 において，スマートフォンの画面上を指で押してから離すまでの時間を timePR，指を離してから再び画面を押すまでの時間を timeRP と定義する．これらを文字数だけ取得するため，文字数を n としたとき，timePR は n 個，timeRP は $n-1$ 個の特徴をそれぞれ取得することができる．

図 3 において，画面上を指で押した座標（以降，始点とする）から，フリック操作を行った際に指が離れた座標（以降，終点とする）までの直線距離を lengthF，始点を通り X 軸に平行な線を想定したとき，その線と始点と終点を結んだ線から成る角度を angleF と定義する．これらも文字数だけ取得するため，文字数を n としたとき，それぞれ n 個ずつの特徴を取得することができる．

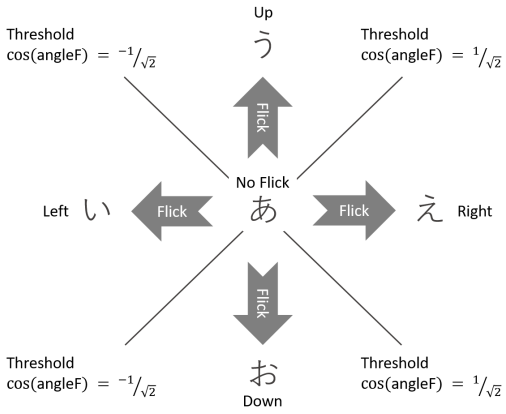


図 4: フリック操作の方向における母音判定の定義

図 4 に，フリック操作の方向における母音判定の定義を示す．フリック操作は,「フリックなし」および上下左右へのフリックの全 5 種類の操作を定めており，5 つの母音に対して図 4 のようにそれぞれ割り当てている．各操作の境界として，angleF の余弦が $1/\sqrt{2}$ または $-1/\sqrt{2}$ を閾値とする．また，子音については，始点の座標を基に判定する．

これらの特徴量をベクトルとし，OCSVM に入力する．このベクトルの次元数は，パスワードの文字数 n としたとき，$4n-1$ 次元となる．

3.3. 提案手法の実装

提案手法の有用性を確認するため，Android スマートフォンに対応するアプリとして実装した．

図 5 に，本提案手法を実装したアプリ画面におけるキーボードの外観を示す．キーボードには各子音を示すボタンを配置し，ボタン上でフリック操作を行うと，その位置に該当する子音とフリック操作に対応した母音を組み合わせて文字を入力する．このキーボードのキー配置は，Android スマートフォンに標準で搭載しているキーボードを参考にしている．

OCSVM を実装するため，機械学習のオープンソースライブラリである scikit-learn[7] を用いる．しかしながら，スマートフォン上で OCSVM のマップを作成するのは，実機のスペックによっては不安定になったり，処理に時間がかかったりと実用性に問題が生じる．

図 5: アプリ画面におけるキーボードの外観

そこで，スマートフォンにて取得したフリック操作の特徴量のデータをサーバに送信し，サーバ上で OCSVM のマップを作成した後，そのマップデータをスマートフォンに送信する手法を採用する．また，複雑な識別にも対応するため，OCSVM に用いるカーネルとして RBF カーネルを採用する．

パスワードの長さは，ひらがな 46 種を用いると想定し，少なくとも 7 文字の長さとする．この長さは，NIST SP800-63[6] において，標的を定めたパスワード推測攻撃に対するレベル 2 の要求事項（攻撃成功確率 2^{-14} 未満）を満たすことを基準にしている．ひらがなに限定した理由として，英数字 36 種（英文字 26 種 + 数字 10 種）の場合では必要になる長さが 8 文字以上になり，また，これらを組み合わせた場合では入力文字種の切り替えによる誤入力が出る可能性があり，入力操作によるユーザの負担が最小になることを考慮したためである．

実装したアプリの使用手順を以下に示す．

1. インストール後の初期実行時，パスワード入力画面を表示し，パスワードの登録をユーザへ促す．
2. ユーザはパスワードを 10 回入力し，登録する．
3. サーバへフリック入力のデータを送信し，OCSVM のマップデータを取得する．
4. 画面点灯時に本アプリが起動するように設定する．

5. 画面点灯し本アプリ起動後、画面ロックとしてパスワード入力画面を表示する．ユーザはパスワードを入力し，画面ロックを外す．以降，画面点灯時，手順 5 のみを行う．

本人判定時に他人であると判断した場合は，入力したのは他人であることを画面に明示し，ロックは解除せずパスワード入力画面に戻る．さらに，その後，本人がロックを解除した際，前回に他人がロック解除を試みた可能性があることを明示し，パスワードの変更を促すようにした．

4. 提案手法における有用性の評価実験

本提案手法の有用性を確認するため，以下の項目を検証する．

- 本人が入力した場合にロックが解除されること
- 他人が本人の真似をした場合はロックが解除されないこと

これらを検証するための実験を行う．以下では実験準備と実施，および，実験結果について述べる．

4.1. 実験準備と実施

実験の被験者は，長崎県立大学の 4 年次学生 10 名（男性 8 名，女性 2 名）とした．この中で，普段からフリック操作で文字を入力している被験者は 8 名であり，残り 2 名（いずれも男性）はボタンを連続でタップして文字を入力する「トグル入力」を普段用いている．今回の実験では，トグル入力を行わず，全被験者はフリック操作による入力で統一する．また，入力に用いる利き手は 10 人全員が右手である．各被験者に実験用のスマートフォンを 1 台ずつ渡し，各自，椅子に座り，スマートフォンを右手に持った状態で入力を行う．

登録するパスワードは，ひらがな 7 文字でかつ 5 種の母音が必ず入る文字列 10 パターンをこちらで事前に用意し，各被験者に示した．これらの文字列はランダムに生成したため，単語としての意味は持たない．文字数は 7 文字であるため，OCSVM に入力するデータのベクトルとしての次元数は，$4 \times 7 - 1 = 27$ 次元となる．

実験手順を以下に示す．今回の実験では，登録または解除のために入力したデータを抽出および分析できるようにパスワードの入力のみを行い，画面ロック解除の機能は外している．そのため，本人判定の成否は，PC 上で判定する．なお，OCSVM のマップはスマートフォンで利用するマップと同じものであるため，PC 上で判定してもその成否は変わらない．

1. 各被験者が 10 回ずつパスワードを入力する．このとき入力したデータを「登録用データ」とする．
2. 30 分間のブランク（休憩）を設け，パスワード登録時における被験者が持つ入力操作の感覚をリセットさせる．
3. 各被験者は登録したパスワードを再び 10 回入力する．このとき入力したデータを「解除用データ」とする．
4. 被験者間でスマートフォンを交換し，各被験者はそのスマートフォンで登録したパスワードを他人として 1 回入力する．これを本人以外のスマートフォンすべてが回るように 9 回繰り返す．このとき入力したデータを「模倣用データ」とする．
5. 登録用データを用いて被験者ごとの OCSVM を作成し，解除用データおよび模倣用データをそれぞれ入力して本人判定の成否を調べる．

手順 4 にて，被験者間でスマートフォンを交換する理由として，端末の違いによるフリック操作感への影響を軽減するためである．また，ユーザ本人が所有するスマートフォンを他人が操作することを考慮すると，パスワードを登録したスマートフォンに他人も同じパスワードを入力することが望ましいと考えられる．

4.2. 実験結果

表 1 に各被験者の本人判定成功数を，表 2 に各スマートフォンでの本人判定数を，それぞれ示す．各表中の値は，10 回または 9 回の入力のうちに判定成功または誤判定した数であり，末尾の値は平均回数を示す．被験者 I および被験者 J は，普段トグル入力を用いている者である．

表 1: 各被験者の本人判定成功数（10 回中）

被験者	A	B	C	D	E	
成功数	8	9	9	8	8	
被験者	F	G	H	I	J	平均
成功数	10	10	8	8	7	8.5

表 2: 各スマートフォンでの本人誤判定数（9 回中）

被験者	A	B	C	D	E	
誤判定数	1	1	3	2	1	
被験者	F	G	H	I	J	平均
誤判定数	2	1	2	0	1	1.4

表 1 より，本人判定成功率は被験者によって 10〜7 回となり，平均は 8.5 回であった．被験者 J は 7 回であったが，序盤の回に失敗が集中していたことから，操作に慣れていないことが考えられる．その他の被験者も前半 5 回に失敗が偏っていたため，実験による緊張や慣れが影響していると考える．

一方，表 2 より，他人が同じパスワードを入力した際にユーザ本人と誤判定する回数は，スマートフォンの端末によって 0〜3 回となり，平均は 1.4 回であった．誤判定が起こる理由として，本人と他人のフリック操作が似ていることが考えられる．そのため，登録用データに対する，解除用データおよび模倣用データの各 cos 類似度を計測し，定量的に比較する．

表 3 に，各被験者における登録用データと各データとの cos 類似度の平均値と分散を示す．これらの値は，登録データにおけるすべての被験者および回ごとのデータ（データ数：100）に対して，解除用データおよび模倣用データのすべての被験者および回ごとのデータ（データ数：100 および 90）をすべての組み合わせで cos 類似度を求め，それぞれの平均を算出したものである．

cos 類似度の各平均値は，模倣用データの方が解除用データに比べて低い傾向にあるため，本人と他人とを区別することは可能であると考えられるが，分散を考慮すると類似度の高低が逆転することもあり，明確に区別することは難しいことが表 3 より読み取ることができる．

表 3: 各被験者における登録用データと各データとの cos 類似度の平均値と分散

被験者	A	B	C	D	E
解除用データ					
平均値	0.962	0.985	0.982	0.977	0.973
分散	0.031	0.012	0.012	0.014	0.022
模倣用データ					
平均値	0.943	0.952	0.971	0.969	0.955
分散	0.025	0.023	0.006	0.020	0.032
被験者	F	G	H	I	J
解除用データ					
平均値	0.998	0.989	0.988	0.958	0.966
分散	0.001	0.003	0.007	0.033	0.031
模倣用データ					
平均値	0.967	0.951	0.972	0.891	0.903
分散	0.021	0.034	0.021	0.032	0.042

表 4: 母音別の登録用データと各データとの cos 類似度の平均値と分散

母音	あ	い	う	え	お
解除用データ					
平均値 (A)	0.998	0.975	0.978	0.982	0.988
分散	0.001	0.011	**0.019**	0.010	0.007
模倣用データ					
平均値 (B)	0.996	0.962	0.953	0.974	0.980
分散	0.001	0.026	**0.032**	0.012	0.010
平均値の差					
(A)−(B)	**0.002**	0.013	**0.025**	0.008	0.008

さらにパスワードに使用する文字により本人判定の精度へ影響を考慮するため，使用したパスワードの文字を各母音ごとに分類し，それぞれの文字について上記と同様に cos 類似度を算出した結果を，表 4 に示す．

表 4 より，解除用データと模倣用データの平均値の差が最も小さいのは母音「あ」であり，最も大きいのは母音「う」であることが確認できる．平均値の差が小さいほど互いのデータは似ているため，本人判定の精度が低くなり，差が大きいほど他人は模倣しづらいということになり，本人判定の精度が高まると予想される．また，両データにおける各母音の分散に注目すると，母音「う」の分散が大きいことも確認できる．

5. 考察

5.1. 実験での検証項目についての考察

今回の実験の検証項目について考察する．まず，項目「本人が入力した場合にロックが解除されること」について，表 1 より，本人判定成功数の平均は 10 回中 8.5 回であった．連続で失敗することを考慮した場合，2 回連続で失敗する確率は約 2.3%，3 回連続で失敗する確率は約 0.3%になり，実用的な回数のうちにロックを解除できることが見込まれる．また，最も成功数が少ない被験者 J は，普段トグル入力を用いてたことから操作に不慣れでいたことが考えられることから，フリック操作に慣れることによってその動きが安定し，失敗しにくくなると考える．

次に，項目「他人が入力した場合にロックが解除されないこと」について，表 2 より，I 以外の被験者は何れかのユーザ本人として 1 回以上は誤判定されていることや，被験者 C においては 10 回中 3 回と最も多い誤判定があることから，本項目については達成できていないと考えられる．この要因としては，OCSVM において学習に用いる登録データの数が少ないことも挙げられるが，同じパスワードを打つことでフリック操作も同じ操作となるため，図 3 で示すとおり，本人と他人との差が出にくいことが最も大きく影響していると考えられる．

他人の場合，自身が普段から扱うスマートフォンと異なる場合もあり，持ち方やパスワードの文字入力間隔に本人との違いが出てくるため，それらが関わる特徴量を分析する必要がある．図 4 において，母音ごとの本人と他人との差を cos 類似度で算出した結果，前述したとおり，母音「あ」では cos 類似度の差はなく，母音「う」が cos 類似度の差が最も出ることが確認できた．母音「あ」の場合は，フリック操作はなくタップのみで入力する形になるため，他の母音よりも差が出る特徴量が少ないことが考えられる．一方，母音「う」の場合は，上方向へのフリック操作であり，操作する指は伸ばす形になる．この指の長さや，スマートフォンを持つ位置によってはフリックの角度 angleF が変わるため，個人差が表れやすいと考えられる．同様に，左方向へ指を伸ばす操作になる母音「い」も，母音「う」に次いで cos 類似度の平均の差が出ていることから，操作する指の動きが本人判定へ影響を与えると考えられる．母音「え」や母音「お」の場合，指を曲げながらのフリック操作になり，フリックの長さは短く，角度も差が出にくい．そのため，個人差が出にくいと考えられる．

5.2. 既存手法との比較

本提案は，パスワード方式とフリック操作による生体認証方式を組み合わせることにより，覗き見攻撃への耐性と，パスワードを容易に変更できる柔軟性を有している．これは既存手法にはない，本提案独自のものであり，新規性であると考える．

本提案に近い既存手法として，Shahzad ら [2] の研究があるが，彼らの手法はスマートフォンの画面上でジェスチャーを表すためのフリック操作をしている．ジェスチャーを表すことにより，その様子を後ろから覗き込む攻撃者にとっては，そのジェスチャーが画面ロック解除に直接結びついた鍵であることが明らかになり，そのジェスチャーを模倣することに注力する．一方，本提案においては，パスワードと同時に，そのパスワードを入力するフリック操作も画面ロック解除の鍵としているため，攻撃者にとっては両方を同時に覚える必要がある．そのため，本提案は既存手法よりも攻撃しにくい手法であると考える．

しかしながら，本提案における本人判定の精度については，既存手法と比較して低いことが懸念される．伊藤ら [3] や泉ら [5] の研究では，認証精度は 9 割台に達しているが，本提案では実験において 10 回中平均 8.5 回の成功数に留まっている．その要因として，OVSVM における学習させるデータやその特徴量が少ないことが挙げられるが，画面ロックを解除する度にその入力データを蓄積しておき，定期的に学習することでその差を補うことができると考える．特にスマートフォンはユーザ本人のみが扱うため，本人に特化した OCSVM のマップを作成することが可能であり，他人からの攻撃を判別しやすくなると考える．

5.3. 入力の慣れへの対応についての考察

今回の実験を通して被験者を観察していると，回を重ねるごとに入力がスムーズになり，文字入力の間隔が短くなっている傾向にあった．本実験では，事前に作成

したランダムなパスワードを用いたため，被験者は当初慣れない文字列で入力が遅かった．このことが実験結果に影響していることは多少懸念されるが，実用性を考慮した場合，画面ロック解除の際に入力したデータを蓄積しておき，定期的に学習を行い，OCSVM のマップを更新する必要があると考える．

OCSVM のマップを作成するには時間を要するため，例えば，サーバと連携し，サーバで OCSVM のマップを作成しておき，その間にスマートフォンでは画面ロック解除時の入力データを蓄積しておく．そして，OCSVM のマップを更新するため，サーバからそのマップを受け取り，それと同時にサーバへ今まで蓄積したデータを渡す．これらの繰り返しにより，定期的に OCSVM のマップを更新することが可能になる．

このような入力の慣れに対応することにより，ユーザ本人に特化した OCSVM のマップへ更新でき，本人判定の精度も向上できることが見込まれる．

6. おわりに

本研究では，覗き見攻撃への対応と，生体認証における情報漏洩への柔軟な対応を備えるため，フリック操作による画面ロック解除手法を提案した．本提案をスマートフォン上に実装し，被験者 10 人に対して試用実験を行った．その結果，本人判定の際に他人からの入力をユーザ本人として誤判定する可能性があることを確認したが，その解決方法として，個人差を判定しやすい特徴量を用いることや，ユーザ本人に特化した OCSVM のマップへと更新し続けることにより，対処していく方法を考察し，今後検討していく方針を定めた．

これらの課題を解決することにより，本人判定の精度を向上することができ，本提案手法の有用性が認められれば，画面ロック解除時における覗き見攻撃の対策や，生体情報漏洩による再登録の柔軟性に寄与できることが見込まれる．

今後の課題として，以下を挙げる．

- フリック操作から抽出できる特徴量の再検討
- ユーザの「慣れ」への対応
- 安全なサーバ運営を含む実用性を考慮したシステムの検討

参考文献

[1] 越前功, 大金建夫, “写真からの指紋復元の脅威とその対策技術”, 情報処理学会論文誌, Vol.58, No.9, pp.824-829, 2017.

[2] Shahzad, M., Liu, X.A., Samuel, A., “Secure unlocking of mobile touch screen devices by simple gestures: you can see it but you can not do it”, Proceedings of the 19th Annual International Conference on Mobile Computing & Networking, pp.39-50, 2013.

[3] 伊藤駿吾, 白石陽, “スマートフォンのフリック入力方式の特徴に着目した継続認証手法の提案”, 第 25 回マルチメディア通信と分散処理ワークショップ論文集, pp.1-8, 2017.

[4] Schölkof, B., Smmola, A.J., Williamson, R.C., and Bartlett, P.L., “New Support Vector Algorithms”, *Neural Computation*, Vol.12, No.5, pp.1207-1245, 2000.

[5] 泉将之, 佐村敏治, 西村治彦, “フリック入力による日本語非定型文のキーストローク認証”, 情報処理学会関西支部 支部大会, E-15, pp.1-8, 2013.

[6] National Institute of Standards and Technology, “Special Publication 800-63: Digital Identity Guidelines, Version 1.0.2”, 2006.

[7] David Cournapeau, “scikit-learn Machine Learning in Python”, https://scikit-learn.org/stable/ (最終閲覧：2021 年 3 月 22 日)

大分県における GPU 活用に向けた取組み

石松 博文 青木 栄二 渡辺 律子 原田 美織
ishimatsu@hyper.or.jp blue@hyper.or.jp watanabe@hyper.or.jp harada@hyper.or.jp
公益財団法人ハイパーネットワーク社会研究所

要旨

国内外では，既に GPU（Graphics Processing Unit）をコア技術とした様々な商品，サービスが提供されている。日本では米中両国が GPU を含む xPU の開発に鎬を削る中，機器の開発はおろか,それを使いこなす人材や問題認識までも不足する現状がある。しかしながら私は,このような ICT 技術の大転換期こそ，その技術を活用した日本の独自性がある商品，サービスを展開できるチャンスではないかと推察する。

大分県は，九州では福岡に次ぐ2番目の工業出荷額を誇る一方で，県内の事業所数減少や人手不足等の根本的課題が存在する。このため，企業が抱える課題，更には地域の課題についてGPUを活用することにより解決できる様々な具体的事例を創出し，国内外に展開することが地域，更には日本にとって有益なことと考えている。当研究所は、2019 年に開催した別府湾会議において、近い将来には、GPU をいつでもどこでもだれでも使いこなせるように、「おおいた AI テクノロジーセンター」の設立を宣言した後、産学官 10 団体と共に設立、運営している。本センターの地域における GPU 活用を進めるための取組みを紹介する。

1. はじめに

当研究所は，九州工業大学，オートバックスセブン株式会社とともに，視覚障害者の為の自律外出支援サービスに関する研究を行い，2019年には「Jetsonを使ったモビリティにおける障害物の検出実証実験」を実施した。本センターを設立し，更にGPU を活用した新規事業企画，技術研修を行い，新たな農業，産業分野での変革を促す以下の活動に取り組んでいる。

2. 活動内容

2.1. 普及啓発

本センターでは,活動に賛同した企業・団体・個人が参加する GPU 活用事例の紹介，GPU に関する最新動向の講演を行っている。中でも，先駆的な取組は，全国の自治体で初となる大分県，株式会社 NTTPC コミュニケーションズとの「大分県におけるGPUの活用に関する三者協定」を締結することにより，大分県内の企業等に無償で提供し，GPU 活用を試みる企業，団体に対し，活用のハードルを下げたことである。更に，津久見市でのグループワークによる産業AIのロングランハッカソンを行い，地域課題解決の為の複数プロジェクト企画を行った。

2.2. 人材育成

学生（高校生・大学生）への Jetson 等実機を用いた授業を開催し，センターの参加メンバーである企業担当者がサポートを行った。2022 年度は企業担当者を対象に GPU サーバを用いた研修会を開催することにより，対象やレベルに合せた人材育成を行う。また，大分県教育庁，株式会社ザイナスと連携し，農業関係高校生を対象にした実技研修カリキュラムを実施し，企業人と学生間の交わりによるスキル向上を図っている。

2.3. ビジネスプロデュース

人材不足等の様々な課題を抱える企業に対し，IT 企業とコンソーシアムを組み，課題解決に向けた実施計画や企画のコンサル，GPU を活用した実証トライアルの実施，検証を行う。具体例は，佐伯市ひじき加工会社「株式会社山忠」の異物混入識別の省力化,津久見市「古手川産業株式会社」の生石灰生成工程での自動化と品質向上を目的とした，原石サイズ管理力の課題について，画像識別により定量化，分類に取り組む。

2.4. 情報発信

2.1～2.3 の活動を広く県内外に発信することにより，本センターへの協力,参加を増やし,更なる活動の推進及び多方面への展開を図る。このため，センターのウェブサイトに事例を紹介すると共にメール，SNS，メディアを活用した活動紹介や参加の案内を展開する。更に，アイデアや利活用事例に定量的な目標を設け，公開することで，更なる取組の活性化に繋げている。

参考文献

[1] Eiji Aoki, Shinji Otsuka, Takeshi Ikenaga, Hideaki Kawano, Masaaki Yatsuzuka, ″Study on Autonomous Outing Support Service for the Visually ImpGPUred,″ Proc. of the 13th Int'l Conf. on Complex, Intelligent, and Software Intensive Systems (CISIS-2019), pp.713-722, Jul.3-5 2019.

[2] Eiji Aoki, Zenjiro Oba, Ritsuko Watanabe, ″Study on Data Utilization of Regional Industry in Cross-cutting and Systematic Regional Community Networks,″ Proc. of the 11th Int'l Conf. on Complex, Intelligent, and Software Intensive Systems (CISIS-2017), pp.795-804, Jul.10-12 2017.

深層学習を用いた路線バスにおける 乗客 OD および混雑状況の推定手法の開発

山下 倫央
北海道大学
tomohisa@ist.hokudai.ac.jp

西浦 翼
北海道大学
tsubasanishiura@ist.hokudai.ac.jp

久保 田遼裕
北海道大学
kubota@ist.hokudai.ac.jp

横山 想一郎
北海道大学
yokoyama@ist.hokudai.ac.jp

川村 秀憲
北海道大学
kawamura@ist.hokudai.ac.jp

弓崎 潔
株式会社 シーズ・ラボ
yumisaki@cslab.co.jp

佐藤 好美
株式会社 シーズ・ラボ
yoshimi@cslab.co.jp

要旨

本稿では，路線バスを対象としたバス車内モニタリングシステムの開発をおこない，路線バスを安全で利便性の高い公共交通機関とすることを目指す．バス車内モニタリングシステムの実用に向けて，深層学習を用いた乗客の *OD* データの推定手法やバス車内の混雑状況の推定手法を提案する．これらの提案手法の有効性を検証するため，実際の路線バスを利用して検証用データを作成した．検証用データを用いた実験をおこない，バス乗客の *OD* データの推定やバス車内の混雑状況の推定の実用可能性を確認した．

1 はじめに

近年，車両を運転するドライバーを計測して， 居眠りやわき見の検知やふらつき運転への警告する運転指導が注目を浴びている．また，車内を撮影するカメラでバス乗客を対象とした車内モニタリングがバスの効率的な運行やバス乗客の満足度の向上の観点から検討されはじめている．これまでのバス事業者に対するヒアリングの結果，乗客の乗降したバス停 (OD：Origin and Destination) の情報やバス車内の混雑状況に対して強いニーズがあることが分かってきた．

バス乗客 OD の推定に関しては，バスの乗客の利用状況を集計したバス乗客 OD データの分析による効率的な運営への改善が期待されており，実際に国土交通省はビッグデータの活用した経営革新支援や，データ分析・収集ツールの発展・高度化を図っている [1]．バス料金支払い用の交通系 IC カードの普及で把握が容易になったと思われがちだが，実際は IC カードの利用情報は IC カードの発行・管理をおこなう事業者に管理されている．IC カードの利用状況の分析結果を利用するためにはバス事業者の追加料金の支払いが必要である．そのため，収益の低い路線バス事業者には利用が困難であるが収益改善には必要であるため，簡易かつ安価に乗客の OD データを収集するシステムが求められている．

また，バス車内の混雑状況の推定に関しては，バス車内動画を解析することで，乗客数や混雑具合を把握して，バスの運行の管理や運行の円滑化に貢献する．さらに混雑状況をリアルタイムに利用者に配信することで，バス利用者がいつ，どのバスに乗るかという判断する手助けになる．店舗・施設や電車では混雑度情報の発信が順調に普及している一方で，路線バスにおいては状況が異なる．路線バスでは鉄道駅のように改札を設置して人数を数えることが難しく，出入口に赤外線センサーを設置し

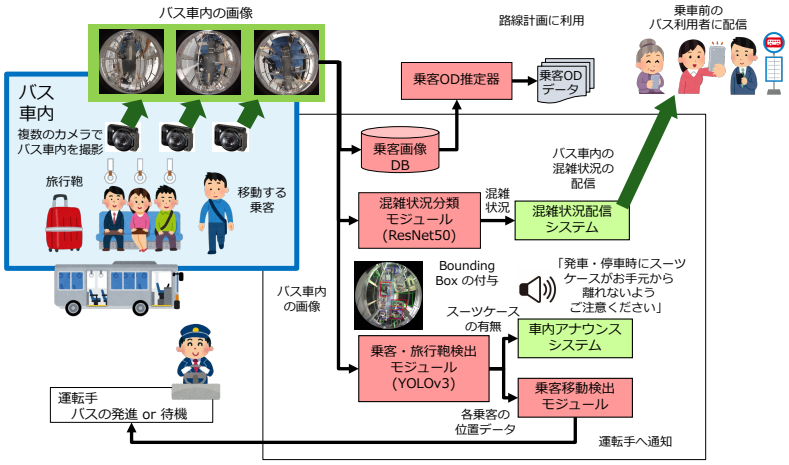


図 1. バス車内モニタリングシステムの構成の概要

ても人や荷物の重なりによって正確に数えることが難しい．西鉄グループのように一部 IC カードの利用実績から混雑度を配信している企業もあるが，前述のバス乗客 OD の推定と同様に，多くの路線バスではバス事業者と IC カード発行・管理する事業者が異なるため，バス事業者が IC カードの利用情報を参照することは容易ではない．そのため、バス事業者が主体となって混雑度を測定することが求められる．

このような状況を踏まえて，路線バスにおいて効率的なバス運行やバス乗客の満足度を向上させるため，バス乗客 OD データを収集し，バス車内の混雑状況を推定するバス車内モニタリングシステムの開発を目指す（図 1）．本稿では，バス乗客の乗降時の画像から OD データを推定する手法とバス車内の混雑状況を推定する手法を提案する．さらに，実際のバスを用いて収集したデータを用いてそれぞれの手法の有効性を検証する．

本稿の位置付けとしては，従来手法を統合して提案するバス乗客 OD 推定手法や混雑状況推定手法に関して，大規模な実データの収集や製品化への本格的な開発をおこなう前に，小規模なバス車内で撮影したデータで提案手法の有効性を確認することである．

本稿の構成を以下に示す．2 章ではバス乗客 OD 推定手法を提案し，3 章で提案手法の有効性を検証する．4 章ではバス車内の混雑状況推定手法を提案し，5 章で提案手法の有効性を検証する．最後に，6 章で本稿の結論と今後の展望を記す．

2　バス乗客 OD 推定手法の提案

本章では，図 2 に示されるように乗降口における個人識別と乗車人物・降車人物の同一推定から構成されるバス乗客 OD の推定手法を提案し，提案手法の実装方法について述べる．

2.1　バス乗客 OD 推定手法の概要

バス乗客 OD データにはバス停毎における乗降人数を記録したバス停別乗降人数と、乗客単位での乗車したバス停と降車したバス停を集計した乗車区間別乗降者数がある．これらのデータを分析することで収益改善のための効率的なダイヤへの見直しや路線の増便・減便が可能となる．さらに，乗車区間別乗降者数を把握することで運賃の見直しにも活用することができる [2].

バス乗客 OD の推定においては，まず，乗降口における個人識別で乗降口と降車口で人物を識別して，識別された人数をバス停別乗降者数として出力する．乗降口における個人識別は，人物特徴量抽出器で画像から人物画像の抽出と，人物画像の特徴量抽出をする前処理をおこなう．次に乗車人物・降車人物の同一推定をおこない乗車・降車ペアの作成をおこない，乗車・降車ペアを合計して乗車区間別乗降者数として出力する．

2.2　メトリックラーニングの適用

バス車内における同一人物判定においては，訓練データには存在しない未知の乗客の画像を同一人物か否かを

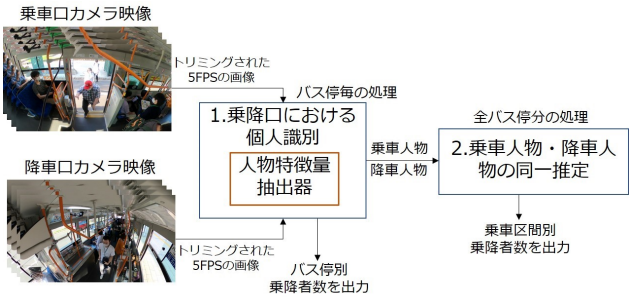


図 2. バス乗客 OD データ推定手法の概要

判定する必要がある．畳み込みニューラルネットワーク (Convolutional Neural Network;CNN) を用いたクラス分類は事前に学習したクラスの分類には有効だが，乗客の OD 推定で求められている未知の 2 つのデータに対して類似度を計算する構造を持っていないため，事前に学習したクラス以外の分類において高い性能を発揮することは期待できない．そのため，ある未知のデータが類似しているか否かを学習する手法としてメトリックラーニングが開発されている．メトリックラーニングでは，似た特徴を持つデータ間の類似度を近く，異なる特徴を持つデータ間の類似度を遠くなるよう学習がおこなわれ，車両の認証 [3] や顔の認証 [4] といった分野において優れた結果を残してきた．メトリックラーニングは Person Re-Identification という人物を認証するタスクに対しても適用されている [5].

メトリックラーニングを用いた Person Re-Identification においては，基準となるアンカーに対して異なるクラスであるネガティブなデータよりも同クラスであるポジティブなデータとの距離をより近くなるような学習をおこなう triplet loss を用いた手法 [6] がある．Person Re-Identification において優れた結果を残している深層学習モデル BoT は，画像分類の深層学習モデル ResNet50 と triplet loss を用いた標準的なベースラインに，メトリック損失と分類損失を 2 つの異なる特徴空間に分離する BNNeck という手法，学習率の Warmup，Random Erasing Augmentation[7]，Label Smoothing[8]，Last Stride[9] といった複数の手法を適用し，Center Loss[10] と，triplet loss と人物 ID の分類損失である ID loss の 3 つを損失を組み合わせて利用することで，他手法に比べて高い精度を得られている．そのため，本提案手法では BoT を用いた人物画像からの特徴量の抽出を適用する．

2.3 バス乗客 OD 推定手法の実装

2.3.1 乗降口における個人識別

乗降口における個人識別の処理の流れを図 3 に示す．ここでは，次章で説明する路線バスで収集したバス停別乗降者データの画像サイズ (ピクセルサイズ 1920×1080) である画像を対象とする．

人物画像からの特徴量の算出 路線バスの乗降口を撮影した画像から人物画像を抽出するために，物体検出モデル YOLOv4 を適用して，確信度 60 %以上で「person」と認識された矩形部分を元画像から抽出する．

まず，乗降中の客ではない人物も多く抽出されてしまう問題がある．例えば，乗車口画像では別の路線のバス待ちで並ぶ人物や歩行者が写りこむことがある．また，降車口画像では乗車後に座席に座ったりつり革に掴まって立っている人物が写りこむことがある．さらに，乗降中の客の中でも身体の一部のみが検出されたり，人物同士が大きく重複する画像は人物画像としてノイズとなる．これらの問題を避けるために，画像サイズが乗車口画像では 650×900，降車口画像では 820×650 にトリミングして，この画像に対して人物画像を抽出する．

次に，人物同士が重複した矩形が画像が抽出されないように，人物が検出されたときに矩形面積を計算し，その面積に 25 %以上重複する他に検出された物体の矩形がある場合には，その画像を利用しない条件を設定した．また，身体の一部のみが写った画像が抽出されないように矩形サイズの条件と乗降中の乗客以外の人物や車外の人物が抽出されないように矩形の中心位置による条件を設定した．矩形サイズの条件は，縦ピクセル数と横ピクセル数の積が 30,000 以上であることとし，矩形の中心座標が設定した範囲の座標内にある場合に抽出した．

このように，乗降車口の画像から抽出した人物画像から BoT を用いて特徴量を算出する．BoT には人物画像を 128×256 にリサイズして入力し，出力として 2048 次元の特徴量が得られる．

人物画像の特徴量を用いた個人識別 人物画像に対して付与されている出現時刻，検出座標，特徴量を用いて乗降口における個人識別をおこなう．

乗降口における個人識別はバス停毎に乗車口カメラと降車口カメラで別々におこなわれる．個人識別として乗車人物群と降車人物群を作成し，それぞれの人物群の

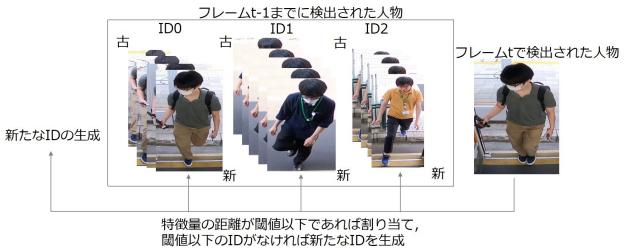


図 3. 乗降口における個人識別の流れ

数を乗車人数，降車人数として出力する．一連の処理は [11] における同一人物のグルーピング手法を参考に，人物画像間の特徴量の距離を算出して比較をおこない，ハンガリアン法により ID を割り当てる．[1] 全ての人物画像間で特徴量を比較するのではなく，図 3 に示されるように，ある人物画像の出現前の時刻までに検出された人物との特徴量とユークリッド距離の比較をおこなう．

ある人物画像と別の人物画像の映っている人物を同一人物と判定するユークリッド距離の閾値を設定し，閾値以下の中でユークリッド距離が最小の人物と同じ ID を割り当てる．閾値以下の人物が存在しない場合は新たな人物として，新たな ID を割り当てる．

各バス停において乗車口と降車口で抽出された人物画像の出現時刻順に ID を割り振っていき，あるバス停 k における乗車人物群 P_{on}^{k} と降車人物群 P_{off}^{k} を作成する．

2.3.2 乗車人物・降車人物の同一推定

乗降口における個人識別によって作成された乗車人物群 P_{on} と降車人物群 P_{off} から同一人物と思われる人物のペアを作成するための乗車人物・降車人物の同一推定手法について説明する．本稿での乗車人物・降車人物の同一推定は，全バス停におけるデータ収集が終わった後に一括しておこなう処理方式を想定している．

まず，乗降口における個人識別で全てのバス停における乗車人物群 P_{on} と降車人物群 P_{off} で特徴量のユークリッド距離行列を作成する．このユークリッド距離行列は降車人物群 P_{off} の要素数を i，バス車内人物群 P_{on} の要素数を j としたとき，$i \times j$ の行列となる．ただし，この距離行列に対して，ペアを作成するためのハンガリアン法をそのまま適用すると，乗車したバス停より前に降車した人物とのあり得ないペアが同一人物とされる可能性がある．そのため，ハンガリアン法を適用する前にユーグリット距離行列に対して簡単なバス停ルールを適用する．バス停ルールは，バス乗客はあるバス停で乗車してから，必ずそれ以降のバス停で降車するという仮定が成り立つとして，(i) ある人物の乗車画像に関して，それ以前のバス停での降車時に撮影された人物画像は候補としない，(ii) ある人物の降車画像に関して，それ以降のバス停での乗車時に撮影された人物画像は候補としない，として，該当するユーグリット距離行列の要素を無限大に設定する．

[1] ハンガリアン法は scipy 0.18.1 の optimize.linear sum assignment を用いて実装する．Scipy.org, https://docs.scipy.org/doc/scipy/reference/release.0.18.1.html.

3 バス乗客 OD 推定手法の検証

本章では，バス乗客 OD データ推定手法の検証用のデータ収集について概説し，提案手法により算出した乗降口における個人識別によりバス停別の乗降者数の精度を確認する．

3.1 バス停別乗降者データの収集

バス停別乗降者数と乗車区間別乗降者数の推定手法の検証用のデータを確保するため，路線バスをレンタルして，研究グループのメンバーを含めた 46 名が乗客役となり，撮影をおこなった．画像のピクセルサイズ 1920×1080 の広角で撮影をおこなうデジタルカメラ（GoProMax）を乗車口と降車口の 2 箇所に設置した．カメラの設置位置に関しては，乗客が簡単に触れることができない地上高の高い位置に設置するという条件のもと，実際に運用することを考慮した位置に設置している．

乗車口カメラと降車口カメラで撮影した例を図 4，5 に示す．乗降者や車内の乗客が増えることによってオクルージョンが頻繁に発生し，画像に乗客の全身が写るタイミングは限られる．

仮のバス停と設定した三地点を周回し，バス停毎に乗降者数を変えながら計 24 回の乗車映像，計 28 回の降車映像を撮影した．

3.2 検証データの作成

特徴抽出に用いる BoT には既存の学習済みモデルがあり，本研究では，さらに Market-1501，DukeMTMC-ReID の学習済みモデルを用いる．しかしそれらのデータセットと比べると，バスに設置したカメラは検出され

図 4. 乗車口を撮影した画像例

図 5. 降車口を撮影した画像例

た人物画像は，上から見下ろした画角が多くなる．予備実験ではバスで収集したデータで訓練データを作成し，新たなモデルを作成する．次に提案したバス乗客 OD 推定手法の検証のため，乗降データを組み合わせることで，計 22 人が 11 箇所のバス停で 7 回ずつ乗降する 22 人のバス停別乗降画像データと，訓練データを含まない計 11 人が 9 箇所のバス停で 5 乗降する 11 人のバス停別乗降画像データを作成した．各乗客は乗車・降車を各 1 度ずつのみおこなった．

3.3 バス停別乗降者数の推定結果

乗降口における個人識別によりバス停別の乗降者数の推定精度の検証をおこなう．乗降口における個人識別された乗降人数と，実際の乗降人数の絶対誤差と絶対誤差率を確認する．11 人のバス停別乗降画像データでは各 5 回の乗降を対象に作成した 2 つの学習済みモデルと作成した 3 つのモデルを合わせた 5 つのモデルで，22 人のバス停別乗降画像データでは各 7 回の乗降を対象に 3 つのモデルで実験をおこなった．

11 人のバス停別乗降画像データの実験結果を表 1，2，22 人のバス停別乗降画像データの実験結果を表 3，4 に示す．いずれのバス停データにおいてもバス停別の推定結果は，推定誤差は 0 または 1 人であり，学習済みモデルに対してファインチューニングモデルの方が同等または少ない誤差で人数の推定を行うことができた．しかし，データによっては複数のモデルで共通して誤差が発生している箇所あり，図 6 のように人物がカメラに写る角度によって，実際は同一人物の画像が別人として識別されたり，逆に別人同士が同一人物として認識された．

図 6. 個人識別の誤判定パターン

また，車内を移動する人物が降車口の人物画像に写りこんでしまい，降車人物として認識されてしまう例もあった．

表 1. 11 人の乗車人数推定誤差

実際の人数	2	2	4	1	2	計
Market-1501 (Pre-trained)	0	1	0	0	1	2
Market-1501 (Fine-tuned)	0	0	0	0	0	0
DukeMTMC-ReID (Pre-trained)	0	0	0	0	1	1
DukeMTMC-ReID (Fine-tuned)	1	0	0	0	0	1
バスデータモデル	1	0	0	0	2	3

3.4 乗車区間別乗降者数の推定結果

乗車人物・降車人物の同一推定により作成された乗車人物と降車人物のペアに対して，乗車区間別乗降者数を

表 2. 11 人の降車人数推定誤差

実際の人数	2	1	5	2	1	計
Market-1501 (Pre-trained)	0	0	2	0	0	2
Market-1501 (Fine-tuned)	0	0	1	0	0	1
DukeMTMC-ReID (Pre-trained)	0	0	1	0	0	1
DukeMTMC-ReID (Fine-tuned)	0	0	0	0	0	0
バスデータモデル	0	0	1	0	0	1

表 3. 22 人の乗車人数の推定誤差

実際の人数	2	2	7	2	1	4	4	計
Market-1501 (Pre-trained)	1	0	0	0	0	0	0	1
DukeMTMC-ReID (Pre-trained)	0	0	3	0	0	1	0	5

表 4. 22 人の降車人数の推定誤差

実際の人数	2	7	1	4	5	2	1	計
Market-1501 (Pre-trained)	0	0	0	1	2	0	0	3
DukeMTMC-ReID (Pre-trained)	0	0	0	0	1	0	0	1

用いた推定精度を検証をおこなう．作成される乗車人物と降車人物のペアの数は実際のペアと同じ数とは限らないので，評価には再現率，適合率，F 値を用いる．

11 人のバス停別乗降画像データの実験結果を表 5，22 人のバス停別乗降画像データの実験結果を表 6 に示す．

11 人のバス停別乗降画像データでは，DukeMTMC-ReID 学習済みモデルにおいて F 値が 1.000 となった．全般的に高い推定精度が得られたのは，推定の対象となるデータの規模が 11 人と小規模であったため，推定が比較的容易でどのモデルでも推定精度が高く，モデル間で差がつかなかったと考えられる．

22 人のバス停別乗降画像データでは，Market-1501 学習済みモデルに比べ，DukeMTMC-ReID 学習済みモデルの方が精度が約 20 %高くなったが，どちらのモデルも 11 人のバス停映像データと比べると精度が下がっている．誤判定の原因としては，学習済みモデルでは姿勢の変化で同一人物であっても距離が大きくなってしまい，別人と判定されててしまうことがあったため，バス停別乗降画像データに大きな姿勢変化を含むデータが少ないことが原因と考えられる．

表 5. 乗車区間別乗降者数の推定精度の比較 (11 人)

モデル	再現率	適合率	F 値
Market-1501 (Pre-trained)	0.909	0.909	0.909
Market1501 (Fine-tuned)	0.818	0.900	0.857
DukeMTMC-ReID (Pre-trained)	1.000	1.000	1.000
DukeMTMC-ReID (Fine-tuned)	0.909	0.909	0.909
バスデータモデル	1.000	0.917	0.957

表 6. 乗車区間別乗降者数の推定精度の比較 (22 人)

モデル	再現率	適合率	F 値
Market-1501 (Pre-trained)	0.682	0.682	0.682
DukeMTMC-ReID (Pre-trained)	0.909	0.870	0.889

3.5 考察

前章で提案したバス乗客 OD 推定手法に対して，バス停別乗降者数と乗車区間別乗降者数の推定をおこない，精度の検証をおこなった．

バス停別乗降者数の推定では，どのバス停でも 0 から 1 人の誤差に抑えて推定することができた．そのため，本稿で提案した乗降口における個人識別手法の有用性を確認することができた．本稿では同一人物かどうかの判定には特徴量距離のみを用いたが，今後は画像内の座標を用いた手法 [11] や，人物画像の撮影時間の時系列情報

を扱うことで，乗降口における個人識別の精度だけでなく，乗車人物・降車人物の同一推定の精度も向上することが期待される．

乗車区間別乗降者数においても小規模であれば，高精度での推定をおこなえることを確認した．また，ファインチューニングにより同一人物・別人での特徴量距離の差は大きくなったが，乗客の姿勢の変化の影響を受けやすくなることを確認した．今回の実験で得られたバス停別乗降者数と乗車区間別乗降者数の推定精度であれば，学習データの数や多様性を増加させることで精度向上が期待できる．そのため，今後は実運用に向けて訓練データに路線バスで実際の乗客が乗降する状況のデータを取得して，バス停別乗降画像データを充実させることを目指す．

4 バス車内混雑度推定手法の提案

本章では，画像分類モデルと物体検出モデルを用いた二つのバス車内の混雑度推定手法を提案する．

4.1 画像分類による混雑状況の分類

画像分類モデルである CNN を用いて，バス車内の画像を入力として，バス車内が空いている，普通，混んでいるの 3 クラス分類をおこなう．乗車人数が 9 名までを空いているとして混雑度 0，乗車人数が 10〜25 名の普通として混雑度 1，乗客数が 26 名以上を混んでいるとして混雑度 2 とする．混雑度 1 の上限はバス車内の座席数とした．CNN のモデルとして，ResNet50 を用いる．

4.2 物体検出によるバス乗客のカウント

物体検出モデルを用いて，バス車内の画像を入力として，画像内の人物検出をおこなう．検出された人数によりバス車内の人数をカウントして，混雑度を推定する．物体検出モデルには，Microsoft COCO のデータセット [12] で学習済みモデルの YOLOv3 を用いる．

5 バス車内混雑度推定手法の検証

本章では画像分類モデル ResNet50 を用いた 3 クラス分類と物体検出モデル YOLOv3 を用いた人数カウントの 2 つの手法の精度の検証をおこなう．画像分類による混雑度の 3 クラス分類ではバス車内全体が写っている前方広角カメラと中央魚眼カメラの画像を用いる．物体検出によるバス乗客のカウントでは，物体検出が適用可能な前方広角カメラと後方広角カメラの画像を用いる．

5.1 バス車内画像の収集

今回実験に用いるデータセットの収集のために，3.1 節で記述されたバス停別乗降者データの収集と並行して，混雑度推定用の検証データも収集した．

混雑度推定用のバス車内画像の収集のために，バス車内の前方に広角カメラ 2 台，バス中央に魚眼カメラ 1 台，バス後方に広角カメラ 1 台を設置した．動画は全て 30fps で撮影されており，サイズは 1,920*1,080 ピクセルである．また，撮影に際して，実際の使用状況に近い撮影をするため，マスク・帽子・上着・鞄の有無，乗降時に座席の変更をおこなった．乗車人数に関しては，3, 4, 6, 7, 8, 24, 25, 38 名の状況を撮影した．バス停からバス停を 1 つのシナリオとして撮影した動画を分割し，30fps の動画を全て画像として切り出した．

予備実験の結果，バス車内全体を撮影できて，手すり等の干渉を受けずに適切な画像分類により混雑度の推定をおこなえる画像を撮影していたのは，前方広角カメラ 1 台と中央魚眼カメラ 1 台であることが確認された．そのため，画像分類による混雑状況の分類には，バス前方広角カメラと中央魚眼カメラで得られた画像データを対象に推定をおこなう．前方広角カメラに関して，73,024 枚の訓練データをバッチサイズ 128，30 エポックで学習させた．中央魚眼カメラに関して，52,891 枚の訓練データを同様にバッチサイズ 128，30 エポックで学習させた．

また，中央魚眼カメラの画像は人物の上下が揃っておらず，また端に行けば行くほどゆがみが大きくなり，人物を検出することができなかった．そのため物体検出によるバス乗客のカウントは，前方広角カメラと後方広角カメラで得られた画像データを対象に推定をおこなう．前方広角カメラの画像では後部座席の面積が小さく，座っている人物の多くが頭部しか写っていないため，後部座席最奥中央しか人物の検出をすることができない．そのため前方広角カメラの画像の後部座席部分をマスクして，後方広角カメラと合わせて人数カウントをおこなう．

5.2 画像分類による混雑状況の 3 クラス分類結果

画像分類による混雑状況の 3 クラス分類の結果として，前方広角カメラのテストデータ 22,967 枚と中央魚眼カ

表 7. 3 クラス分類の正解率

混雑度	前方広角カメラ	中央魚眼カメラ
0	0.976	0.955
1	0.993	0.940
2	1.00	1.00
全体	0.990	0.958

表 8. 混雑度推定結果

混雑度	推定人数 (前+後)	正解人数 (前+後)
0	4.93 (1.63+3.30)	6.43 (2.93+3.50)
1	16.2 (5.02+11.2)	19.3 (7.84+11.5)
2	19.8 (6.16+13.6)	22.9 (7.88+15.0)
全体	13.9 (4.35+9.59)	16.6 (6.51+10.1)

メラのテストデータ 17,344 枚に対する正解率を表 7 に示す．いずれの混雑度においても前方広角カメラの方が中央魚眼カメラより正解率が高かった．この原因としては，中央魚眼カメラでは通路上のオクルージョンが激しいこと，バスの前方・後方の奥側がゆがんで潰れていることなどが考えられる．

5.3 物体検出によるバス乗客のカウントの結果

物体検出によるバス乗客のカウントの結果として，

前方広角カメラと後方広角カメラの両方の画像に対して YOLOv3 で人物検出をおこなう．データセットとして，前方広角カメラと後方広角カメラで撮影した画像 110 枚を用いる．この 110 枚の画像は各シナリオから 5 枚ずつ抽出した，混雑度 0 の画像 30 枚，混雑度 1 の画像 55 枚，混雑度 2 の画像 25 枚から構成される．

各カメラ画像に対する性能評価を表 9 と表 10 に示す．後方広角カメラの画像では，適合率も再現率もある程度の性能を発揮しているのに対して，前方広角カメラの画像の性能は大きく劣っている．特に再現率が 0.509 と極めて低い値となっている．この原因として，座っている乗客で，立っている乗客やバスの手すりとのオクルージョンが大きいときに検出できないケースが多かったためである．

各混雑度ごとの混雑度推定結果を表 8 に示す．いずれのクラスに関しても正解よりも少ない人数を検出する結果となった．正解との差は後方広角カメラよりも前方広角カメラが大きかった．オクルージョン等により十分に人物を検出できなかった結果であると言える．

表 9. 前方広角カメラの人物検出性能

混雑度	適合率	再現率	F 値
0	0.956	0.522	0.653
1	0.780	0.496	0.598
2	0.673	0.523	0.580
全体	0.803	0.509	0.609

表 10. 後方広角カメラの人物検出性能

混雑度	適合率	再現率	F 値
0	0.983	0.861	0.911
1	0.914	0.854	0.882
2	0.892	0.744	0.809
全体	0.928	0.831	0.873

各混雑度の画像に対して，物体検出をおこない，人物を検出した例を図 7 〜 図 10 に示す．バス車内画像に YOLOv3 が人物として出力した矩形を重畳している．

前方広角カメラに関して，図 7 に示されるように，混雑度 0 では画像左側に座っている人物は矩形が抽出できているが，画像右側に座っている人物はバス天井に取り付けられている手すりとのオクルージョンがあり，検出できなかった．後部座席の人物は面積が小さく，また頭部しか写っていないため検出されなかった．図 8 に示されるように，混雑度 1 では画像の左側が依然として検出できているのに加えて，右側で立っている人物も検出できた．しかし，立っている人物と座っている人物とのオクルージョンがあると座っている人物を検出できないケースが多かった．図 9 に示されるように，混雑度 2 では立っていて体が十分写っている人物を検出できたが，座っている人物の頭部や体が一部しか映らないケースが多く，車内の人数を十分に把握することができなかった．

後方広角カメラに関して，図 10 に示されるように，少ない人数の時はもちろん，満席に近い状態でも十分に人物を検出することができた．カメラの角度と撮影する面積が適切であったため，十分な性能を発揮できたと考えられる．画像右奥で一部，乗客の位置によってはオクルージョンが起き，1,2 人検出できないケースがあった．

5.4 考察

前章で提案したバス車内混雑度推定手法に対して，バス車内画像を用いて精度の検証をおこなった．

画像分類による混雑状況の分類に関しては，全体で 0.99 を超える正解率は数字だけを見れば十分実用に足る性能である．しかし一方で，今回用いたデータセットが設定するタスクとして簡単であった．3, 4, 6, 7, 8, 24, 25, 38 名の状況に対して，混雑度 0 は 9 名以下，混雑度 1 が 10 名から 25 名まで，混雑度 2 は 26 名以上となっている 3 クラスを設定しているため，検証用のデータのクラスごとの差が明確であったため，高い正解率が得られた．そのため，乗客数に関してはより網羅的なデータを収集して，各クラスの境界付近の人数である 10 余人や 30 人前後の画像を含めて学習することが必要である．

また，物体検出によるバス乗客のカウントに関しては，YOLOv3 を用いた人物検出では実用に足る精度での推定が困難であり，その原因としてオクルージョンの問題が挙げられた．改善方法として，人物の全身を抽出する検出方法に加えて，頭部検出などを用いて人物の一部でも写っていれば乗客として検出するモデルを適用することが挙げられる．今回のデータ収集では，バス車内の状況をモニタリングするカメラとして想定していたが，乗降口を写すカメラで乗車数，降車数をカウントして，現在の乗客数を算出する手法を適用することも考えられる．

6 おわりに

路線バスにおいて効率的なバス運行やバス乗客の満足度を向上させるため，バス車内モニタリングシステムにはバス乗客 OD データやバス車内の混雑状況を推定する機能が求められている．このような背景を踏まえて，本稿では，バス乗客 OD データの推定手法とバス車内の混雑状況手法の提案と検証をおこなった．

バス乗客 OD データの推定手法に関しては，メトリックラーニングを適用して，乗降口における個人識別をおこない，乗降人物・降車人物の同一推定をおこなう手法を提案した．提案したバス乗客 OD データの推定手法の実用性を検証するために，実際の路線バスを利用したバス停別乗降者データの収集をおこない，異なるデータセットで学習した推定器のバス停別乗降者数と乗車区間別乗降者数の推定精度を比較して，今後のデータ収集と改善による実用性を確認した．

バス車内の混雑状況手法においては，画像分類モデル ResNet50 を用いて混雑度を推定する手法と，物体検出モデル YOLOv3 を用いて車内の人数を推定する手法を提案した．提案したバス車内の混雑状況手法の実用性を検証するために実際の路線バスを利用したバス車内データの収集をおこない，それぞれの混雑状況推定手法の推定精度を確認した．画像分類による混雑度の推定に関しては，実験においては実用性を示したが，検証用のデータが不足しているため，今後のデータの収集の必要性を確認した．また，推定精度が低かった物体検出によるバス乗客のカウントに関しては，改善案を検討し，この手法の有効な状況を検討した．

謝辞

本研究の一部はノーステック財団「研究開発助成事業」の助成を受け実施されたものです．ここに感謝の意を表します．

参考文献

[1] 国土交通省． 「ビッグデータの活用等による地方路線バス事業の経済革新支援」(平成 27 年度) 「ビッグデータの活用によるデータ収集・分析ツールの発展高度化」(平成 29 年) について． https://www.mlit.go.jp/sogoseisaku/jouhouka/sosei_jouhouka_fr1_000014.html, 2017.

[2] 福本雅之. 公共交通利用実態データの調べ方と活用法あなたのまちの公共交通、把握・分析、しっかりできていますか？～地域公共交通セミナー（秋期）～. https://wwwtb.mlit.go.jp/chubu/aichi/kikaku/seminafukumotosennsei.pdf, 2019.

[3] Shuting He, Hao Luo, Weihua Chen, Miao Zhang, Yuqi Zhang, Fan Wang, Hao Li, and Wei Jiang. Multi-domain learning and identity mining for vehicle re-identification. In *Proc. CVPR Workshops*, 2020.

[4] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep

face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[5] Nicolai Wojke and Alex Bewley. Deep cosine metric learning for person re-identification. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 748–756. IEEE, 2018.

[6] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *CoRR*, Vol. abs/1703.07737, , 2017.

[7] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

[9] Yifan Sun, Liang Zheng, Yi Yang, Qi Tian, and Shengjin Wang. Beyond part models: Person retrieval with refined part pooling (and a strong convolutional baseline). In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[10] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pp. 499–515, 2016.

[11] 山田遊馬, 廣森聡仁, 山口弘純, 東野輝夫. 路線バスにおけるカメラ画像を用いた od 計測システムの提案. 研究報告モバイルコンピューティングとパーベイシブシステム（MBL）, 第 93 巻, pp. 1–8, 2019.

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollar. Microsoft coco: Common objects in context, 2015.

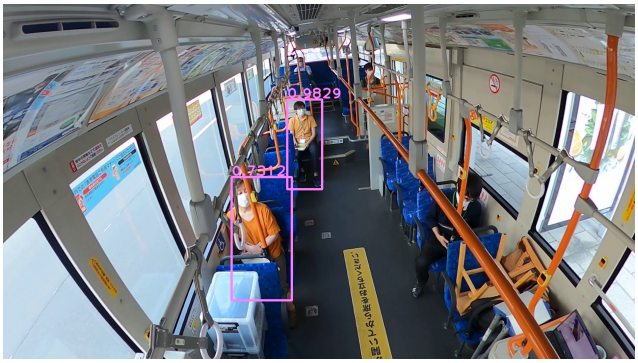


図 7. 人物検した前方広角カメラの画像例 (乗客数 6 人の場合)

図 8. 人物検出した前方広角カメラの画像例 (乗客数 24 人の場合)

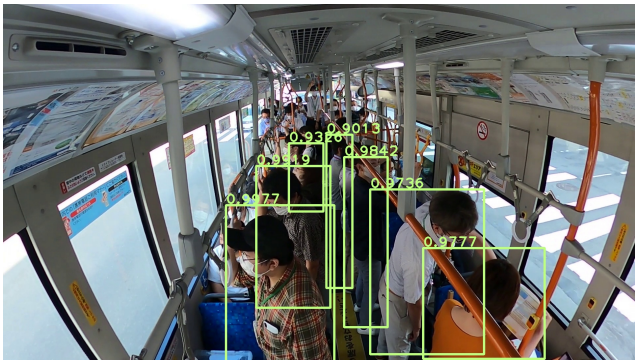


図 9. 人物検出した前方広角カメラの画像例 (乗客数 38 人の場合)

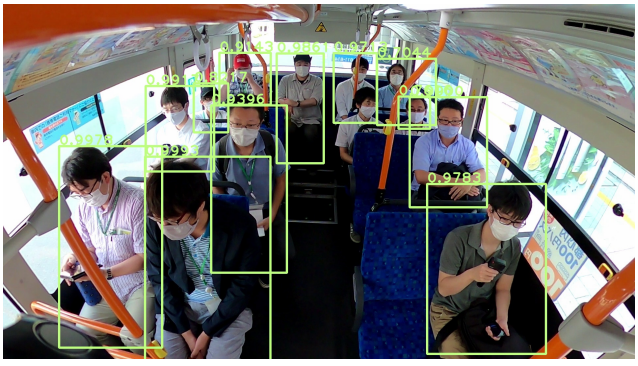


図 10. 人物検出した後方広角カメラの画像例

人工知能技術の産業応用に向けた産学連携に関する事例報告

山下 倫央
北海道大学大学院情報科学研究院
tomohisa@ist.hokudai.ac.jp

要旨

本発表では，大学教員サイドから見た共同研究を主軸とした産学連携に関する事例を報告する．人工知能技術の研究に取り組み，産業応用を進めている大学教員の立場から，民間企業との連携方法の一つである共同研究の準備フェーズから実施フェーズまで流れを概説し，円滑に共同研究を進めるためのノウハウを共有する．本発表が契機となり民間企業と大学の共同研究が促進され，イノベーションの創出につながることを期待する．

近年，大学では民間企業からの研究資金等受入額が増加しており，共同研究費や受託研究費は大学にとって貴重な収入源になってきていることが報告されている[1]．北海道大学では，教育と研究という基本使命に加えて，研究成果の社会還元を「第三の使命」として位置付けて，産学・地域協働推進機構が 2015 年 4 月に設立された[2]．北海道大学においては民間企業からの研究資金等に関して，令和元年度の共同研究の受入額が約 20 億円となっている．産学・地域協働推進機構は全学的な共同研究費に関する目標として，対前年度増加率 12% を掲げている．

深層学習をはじめとする人工知能技術の特徴として，インターネット上での情報の収集が容易であることが挙げられる．例えば，ウェブサイト arXiv [3]にはプレプリント論文（査読前の論文）がにアップされており，自由に閲覧することができる．また，ソースコードをホスティングするソフトウェア開発のプラットフォームである GitHub [4] を通じて，人工知能技術の最新のソースコードを入手することができる．技術情報共有サービス Qiita には人工知能技術を含むプログラミングに関する Tips やノウハウが公開されており，閲覧することができる[5]．しかし，プレプリント論文や公開されたソースコードでスピルオーバーした最新の知見を理解して，産業応用することは容易ではない．

大学の研究室ではプレプリント論文や GitHub に公開されたコードを常に研究に利用しているため,「スピルオーバーを利用している人」と連携すること，つまり，大学教員との連携は最新の知見の迅速な利用には有効な手段である．共同研究は，大学教員と企業等の研究者が対等の立場で共通の課題について共同で研究を行う制度である．前述のように大学教員が企業に知見を提供するというメリットがあるが，情報系の大学教員が企業との共同研究を行うメリットとして，実データの提供と「現場」への介入が挙げられる．

特に人工知能技術の応用研究においては，実際の対象となるデータを扱い，問題解決策の社会実装を目指すことが重要である．インターネット上で公開されているオープンデータを用いてもデータ分析はできるが，介入実験ができないため，問題解決策の検証が困難である．そのため，実際の問題状況を含んだデータと問題解決策を適用可能な「現場」を提供する共同研究には大きなメリットがある．

本発表では，人工知能技術を用いて新たなサービスを開発する際に直面する知見と人材の不足への対応策としての大学との連携方法・契約形態を概観する．さらに，著者の所属する調和系工学研究室の取り組んでいる共同研究において設定された課題，実施体制や進め方について実施事例を紹介する．

[1]平成３０年度 大学等における産学連携等実施状況について (令和２年１月１７日), 文部科学省 科学技術・学術政策局 産業連携・地域支援課 大学技術移転推進室, https://www.mext.go.jp/content/20200109_mxt_sanchi01_00000 3783_02_01.pdf

[2]産学官連携ポリシー，北海道大学，https://www.hokudai.ac.jp/research/innovation/policy.html

[3]arXiv, https://arxiv.org/, arXiv には計算機科学だけでなく，物理学, 数学等のプレプリント論文がアップされている．

[4]GitHub, https://github.com/

[5]Qiita，https://qiita.com/

組込みシステムへの派生 MBD 適用に向けたパラメータ設定方式の検討

秋下 耀介
株式会社日立製作所
yosuke.akishita.nw@hitachi.com

大島 浩資
株式会社日立製作所
kosuke.oshima.kn@hitachi.com

若林 昇
株式会社日立製作所
noboru.wakabayashi.mu@hitachi.com

川上 真澄
株式会社日立製作所
masumi.kawakami.ch@hitachi.com

要旨

組込みソフトウェア開発においては，ソフトウェア開発規模が増大しつつ短納期化する中で効率的に開発を行うため，開発者は派生開発により新規開発量を削減している．また，モデルベース開発(MBD: Model Based Development)を取り入れることでより効率的なアプリケーション開発を実現する． 特に制御ソフトウェアでは，ハードウェアの種類により派生製品が存在しており，類似した機能を持つソフトウェアを開発することが多い．そこで，MBD における派生開発では，ハードウェア仕様の異なる部分のパラメータ化によりモデル内で制御ロジックとハードウェア仕様を分離し，パラメータ変更のみで派生製品を実現する方法が考えられる． 一方でパラメータ設定においては，不具合を起こさないため設定対象となるパラメータの適切な選択や，パラメータ数が多い場合には設定の効率化が必要となる．

本論文では，仕様書とパラメータ間のトレーサビリティに基づき，仕様に対して変更すべき適切なパラメータを選択することで派生開発を実現するパラメータ設定方式を検討する．これを実現するため，過去の実プロジェクトの仕様書やパラメータを分析しパラメータ設定ツールを開発した．パラメータ設定ツールを実プロジェクト適用した結果，仕様に応じて適切なパラメータを選択可能であるほか，パラメータ設定工数の 94%が削減可能であり，工数削減に有効であることがわかった．

1. はじめに

輸送用機器や産業用機器のような組込みシステムに搭載するソフトウェア開発では，周辺装置の制御や情報処理機能の高度化に伴い，ソフトウェア開発規模が増大している[1]．このような状況下で，信頼性や安全性への要求を満たす高品質なソフトウェア開発技術の重要性が増している．一方でソフトウェアの要求は，顧客の要求やシステム特性，周辺装置仕様などの要因で変化するため，これらを満たすソフトウェアを効率良く開発する必要がある．特に制御ソフトウェアでは，ハードウェアの種類に応じて派生製品が存在しており，類似した機能を持つソフトウェアを開発することが多い．一般に，類似した機能を持つソフトウェアを効率良く開発するために，派生開発の考え方を取り入れて開発を行う[2][3]．また，MBD を取り入れることでより効率的なアプリケーション開発を実現する．MBD における派生開発では，ハードウェア仕様の異なる部分のパラメータ化により制御ロジックとハードウェア仕様を分離し，パラメータ変更のみで派生製品を実現することが考えられる[4][5]．

本論文で例として検証した組込みシステムでは，ベースソフトウェアを選定し，要求機能に対して新規開発ではなく，ベースソフトウェアの既存モデルを再利用し，モデル内の制御ロジックの変更かパラメータ値を設定して動作を変更することで，各プロジェクトの開発を行ってきた．プロジェクトにより要求の機能に対しパラメータ設定で対応できる量は異なるが，パラメータを設定することによって効率的に派生製品を実現している．ただし，ベースからパラメータを変更する際には，ある仕様に対し複数のモデルの中から適切なパラメータを選択する必要がある．しかし，仕様書とパラメータ間のトレーサビリティが確保されておらず，開発者は経験に基づいてベースソフトウェアの中から変更すべきパラメータを判断しており，不適切なパラメータを選択する可能性がある．また，ハードウェアとの組み合わせ過程では，重量など計画値と実際の値が変わる場合があり，シミュレーションでは再現できない不確定要素となっているため，テスト段階において設計者とは異なる設定者がパラメータの調整を行う場合がある．派生開発を繰り返す中で，パラメータ間には暗黙知の関係をもつものも存在しており，それらを考慮できずパラメータ

を変更し，動作不良となる場合がある．さらに，パラメータの設定作業は人手による計算・入力を伴うため，計算ミスや変更モレなどの人的ミスが発生する可能性があるほか，パラメータが多数存在する場合は，全パラメータの設定に工数を要する．

本論文では，適切かつ効率的にパラメータを設定するため検討したパラメータ設定方式と，開発したパラメータ設定ツールについて報告する．2 章では，対象とするシステムの概要と前提とする開発プロセスを示し，本論文が扱うパラメータ設定の課題について述べる．3 章では，課題に対して検討したパラメータ設定方式と，これに基づき開発したパラメータ設定ツールについて提案する．4 章では，パラメータ設定ツールを実プロジェクトに適用した結果を示し，課題の解決に対して考察する．最後に 5 章で本論文のまとめと今後の課題について述べる．

2. 背景と課題

2.1. 対象システム

本論文の対象として，以下の特徴を持つ組込みシステムを定義する．

- 開発対象システムは，ソフトウェアが動作する装置として一つ以上のコントローラを有する．
- 開発対象システムは，コントローラ以外の機器として一つ以上のシステム内機器を有する．システム内機器はコントローラと接続され，コントローラに入力を与えるか，コントローラによる制御の対象となる．システム内機器としては，速度のような情報を取得するセンサや，モータのような物理的運動を行うアクチュエータが接続される．
- 開発対象システムは，一つ以上の外部システムと接続する．外部システムはシステムと情報を送受信する．
- 開発対象システムは，システム内機器および外部システムから受信した情報に基づいて，システム内機器を制御する．
- 開発対象システムは，外部システムが要求する情報を，システム内機器および外部システムから受信した情報に基づいて算出し，外部システムに送信する．

以上のシステムの構成例を図 1 に示す．図 1 の開発対象システムは，コントローラ，システム内機器 1, 2 で構成され，外部システム 1, 2 と接続する．開発対象システムはシステム内機器および外部システム 1, 2 から受信した情報に基づいてシステム内機器を制御し，外部システム 1, 2 が要求する情報を送信する．

図 1 の組込みシステムでは，コントローラが制御ソフトウェアを有しており，MBD においては，例として図 2 のようなブロック線図により設計される．図 2 は，アクチュエータを外部システムの入力値に基づき制御する機能を実現するブロック図の例であり，制御の最大値をパラメータとして持つ．また，センサ故障検知機能を有しており，センサ故障を判断するための下限値や，センサ有無による機能切り替えのパラメータを持つ．

本論文で例として検証した組込みシステムは，複数の制御パラメータを有しており，制御パラメータに基づき動作が変化する．パラメータの例としては，電圧などシステムの要求仕様値を元に換算して導出した制御リミット値や，機器制御信号の出力時間のような制御値や，センサの有無による機能切り替えのようなパラメータなどが挙げられる．

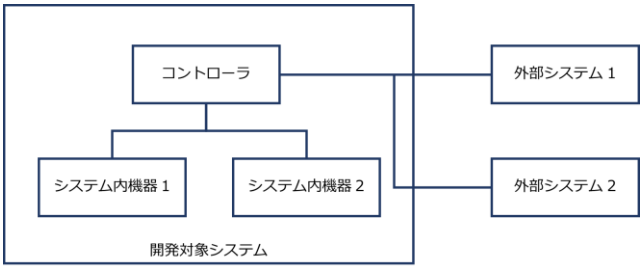


図 1 開発対象システムの構成例

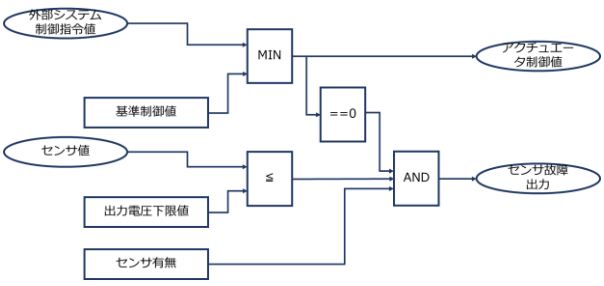


図 2 制御ソフトウェアのブロック線図例

2.2. 派生 MBD

2.1 節のシステムについて，前提となるシステム開発工程を定義する．図 3 のような V 字開発プロセスにおいて MBD を適用する．MBD ツールを使用することで，仕様化，シミュレーション，コード生成，品質や再利用性を改善する[6]．各プロジェクトの開発時には，ベースソフトウェアを選定し派生開発を実施する．なお，MBD で作ったものについては，モデルが仕様書に代わるため，積極的に再利用され得る[7]．

【P1: 要求定義】
顧客から提供される要求仕様書やヒアリングをベースにシステムの要求を定義する．
【P2: 基本設計】
システムへの要求に基づき，外部システムとのインタフェースを設計する．そして，要求および外部システムとのインタフェースを実現するための，ハードウェア，ソフトウェアそれぞれの仕様を定義する．このとき，ベースとするソフトウェアを選定し，ソフトウェアの変更箇所を抽出する．ソフトウェア変更箇所の抽出においては，各機能について比較表を用いつつ P1 の結果に基づき変更の必要有無を検討する．
以降プロセスではソフトウェアについて述べる．
【P3: 機能設計】
P2 の結果に基づき，ソフトウェア全体の機能を厳密に定義する．
【P4: 詳細設計】
P3 の結果に基づき，ソフトウェアの各機能を関数に至るまで詳細化する．なお，開発対象システムのソフトウェアについて，制御設計の効率化のためアプリケーションを MBD で，そのほか性能要求が厳格な個所は C 言語で開発する．MBD ツールとしては社内製ツールを利用する．社内製 MBD ツールは，モデルで使用するパラメータを一覧管理する機能を有しており，パラメータ値を設定する場合には一覧から値を入力する．
各機能についてベースソフトウェアがある場合には，開発者が機能の変更箇所に応じて該当するモデルを探し出し，処理内容やパラメータ名から制御ロジックやパラメータの変更有無を判断し，それらを修正する．
【P5: コーディング】
P4 の結果に基づき，プログラミングを実施する．このとき，MBD ツールで設計したアプリケーションについてはコードを自動生成する．
【P6: 単体テスト，P7: 結合テスト，P8: システムテスト】
P2 から P4 の仕様に基づきソフトウェアをテストする．テスト対象が関数単体からソフトウェア全体に至るよう段階的にテスト対象を拡大する．
【P9: 試運転】
顧客先環境において，システムが要求定義どおりの動作となることをテストする．

なお，MBD ツールの活用においては，ベースソフトウェアのモデルについて，モデルのパラメータを設定して動作を変更する． パラメータ設定手順として，開発者は仕様に基づいて動作を変更したいモデルから変更すべきパラメータを選択し，要求仕様などから当該パラメータの数値計算を行い，MBD ツールへ計算した値を手入力する．

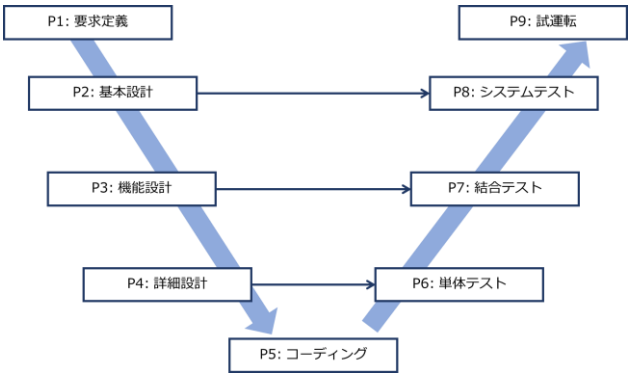


図 3 システムの開発工程

2.3. パラメータ設定における課題

2.2 節に示す派生 MBD により，モジュールの新規開発より少ない工数で開発が可能であるが，本論文の対象システムにおけるパラメータ設定では以下の課題を定義する．これらは，図 3 の P4(詳細設計)においてパラメータを設定するにあたって発生する課題である．

【課題 1】パラメータの不十分な考慮による不適切なパラメータ選択
パラメータを変更する際には，複数のモデルの中から適切なパラメータを選択する必要がある．適切なパラメータを選択するためには，上位仕様からパラメータまでトレ

ーサビリティが確保されている必要がある. しかしながら, 派生開発を繰り返す中で, トレーサビリティが確保されない場合があり, 仕様変更に対して不適切なパラメータを選択する可能性が考えられる.

本論文で例として検証したシステムにおいては, 仕様書とパラメータ間のトレーサビリティが確保できておらず, 開発者は経験に基づいてベースソフトウェアの中から変更すべきパラメータを判断していた. このシステムにおいては, パラメータは 1456 個存在し, 類似した名称や数値を持つパラメータが多数存在するため, 不適切なパラメータを選択する可能性がある. また, パラメータの選択には工数を要する. このシステムでは, 全パラメータのうち 400 個以上をベースソフトウェアから変更する.

【課題 2】暗黙知の関係を持つパラメータの変更漏れ

パラメータの中には, 仕様として明示されていないが同時変更が必要なパラメータなど, 暗黙知の関係を持つパラメータが存在する場合がある.

本論文で例として検証したシステムにおいては, システムのある要求値に基づき計算されたパラメータなどがあるが, 同じ要求値を参照して相関関係にあるパラメータが存在する. さらに, 各モジュールの動作のみならず, システム全体の動作を考慮して設定する必要があるパラメータも存在する. 派生開発を繰り返すほか, テスト結果に基づく変更を繰り返す中で, これらが仕様化されず暗黙知となっている. しかしながら, 同時に変更すべきものやシステム全体の動作を把握してパラメータを設定することは, 熟練した設計者でなければ困難である. さらに, ハードウェアとの組み合わせ過程では, 重量など計画値と実際の値が変わる場合があり, シミュレーションでは再現できない不確定要素となっているため, 設計者のみならずテスト実施者など複数のパラメータ設定者が存在する. したがって, 暗黙知を把握できておらずパラメータの変更漏れが発生する可能性がある.

【課題 3】パラメータ計算・入力間違い

パラメータ設定においては, 変更すべきパラメータについて要求仕様に基づいた数値計算や計算値の MBD ツール入力を行う. しかしながら, これらの作業は人手によって実施されるため, パラメータの計算間違いや, ビット位置の間違いなどパラメータ入力間違いが発生する可能性がある. また, 本論文で検証したシステムにおいては, パラメータ数が 1456 個であることから, 全パラメータを計算・入力するには工数を要する.

3. 派生 MBD 適用に向けたパラメータ設定方式

3.1. パラメータ設定方式の概要

本論文では, 2.3 章で述べた課題に対し, 以下のキーアイディアを提案する.

【キーアイディア 1】上位仕様書-パラメータ間のトレーサビリティに基づく派生開発: パラメータまでトレーサビリティを確保することで仕様に基づいた根拠あるパラメータ設定を実現し, 課題 1 で示すような経験に基づいたパラメータ選択や, 課題 2 に示すようなシステム全体の動作を考慮しないパラメータ設定を防止する. さらに, パラメータ選択に伴う調査工数を削減する.

【キーアイディア 2】入力インターフェースの共通化: 同じ入力値を参照して設定すべきパラメータについては, 要求値の入力インターフェースを共通化することで, 課題 2 に示すような同時変更が必要なパラメータの変更漏れを防止する.

【キーアイディア 3】自動計算, 自動入力: 従来の人手による作業を自動化することで属人性を排除し, 課題 3 に示すような計算や入力間違いを防止する. さらに, 計算や入力に伴う工数を削減する.

上記アイディアを実現する方法として, 図 4 に示すパラメータ設定ツールを開発する. 以降の各節では, 各キーアイディアの実現過程や, パラメータ設定ツールで実装した機能の詳細を述べる. なお, 実際の製品システムを対象にドキュメント調査やパラメータ分析などを行いパラメータ設定ツールを開発した.

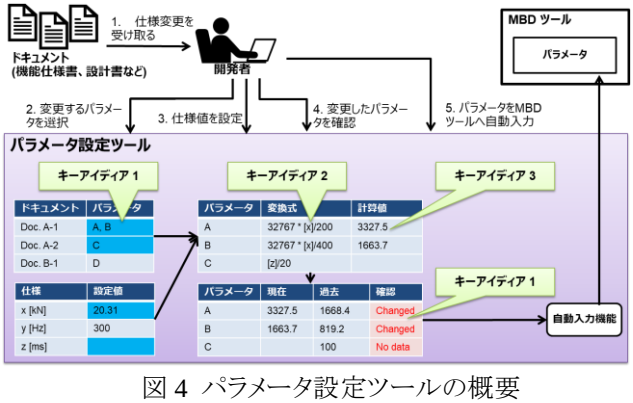


図 4 パラメータ設定ツールの概要

3.2. 上位仕様書-パラメータ間のトレーサビリティに基づく派生開発

パラメータと仕様書間のトレーサビリティを明確にすることで，ある仕様変更に対して変更すべきパラメータを示す．ハードウェア仕様の異なる部分をパラメータ化しているため，類似した機能をもつ製品であれば，パラメータ変更のみで同じ制御ロジックを用いて派生製品を実現することができる．

トレーサビリティを明確にするため，我々は対象システムについて 23 種のドキュメント調査を行うほか，過去の仕様変更やパラメータ変更実績を分析した．さらに，システムとしての動作考慮の必要があり安易に変更できないパラメータなどの暗黙知について，変更実績をベースに開発者へ繰り返しヒアリングを行った．この結果，対象システムのパラメータ 1456 個のうち 558 個はプロジェクトの中で変更する可能性が高いが，残りのパラメータは変更する可能性が低く，変更する場合にシステムとしての動作が不安定になる可能性を考慮して変更可否を議論すべきパラメータであるとわかった．

パラメータ設定ツールにおいては，各パラメータについて仕様とのトレーサビリティを示すタグを埋め込むほか，ベースソフトウェアのデフォルト値から変更する際には変更可否の議論が必要なパラメータを明示する．

3.3. 入力インターフェースの共通化

同じ要求値を参照して相関関係にあるような同時変更が必要なパラメータについて，パラメータ設定における入力インターフェースを共通化することで，パラメータの変更漏れを防止する．入力インターフェースを作成するため，我々はパラメータ値とパラメータの計算式を分析し，同時変更の必要性があるパラメータ候補をリスト化した．その後，開発者にそれらの関係性をヒアリングし，仕様化されていないパラメータの関係を明確にした．

3.4. 自動計算・自動入力

パラメータ設定ツール内に計算式を実装することで自動計算を実現し，MBD ツールへの入力機能を実装することで自動入力を実現する．

表 1 従来とツール適用後のパラメータ設定作業

作業項目	従来	ツール適用後
ベースソフトウェア調査によるパラメータ選択	ベースソフトウェアのパラメータを調査し，経験に基づき変更可否を判断	パラメータ設定ツール内のパラメータ変更可否情報やトレーサビリティを元にパラメータ選択
パラメータ設定根拠の調査	パラメータ設定設定根拠となるドキュメントを探す．または経験を根拠とする．	トレーサビリティに基づきドキュメントを確認
パラメータ計算	ベースソフトウェア調査やパラメータ設定根拠の調査で得られた値から手計算	仕様との差分やトレーサビリティに従い設定値を入力し自動計算
MBD ツールへ入力	MBD ツールへ計算したパラメータを手入力	パラメータ設定ツールから自動入力
パラメータ設定検証	パラメータを一つずつ確認	ツール内で変更内容を一括確認
手順・環境構築等の情報共有	散在したドキュメントやツールを共有	ツール内に必要な情報を集約
パラメータ設定不良対応	パラメータ選択間違いや計算・入力間違いなどの修正	-

3.5. パラメータ設定ツールを用いた開発手順の定義

適切なパラメータを選択し，誤りなく入力を行うため，以下の手順でパラメータ設定ツールを運用することを考えた (各手順は図 4 の手順に対応する).

1. 仕様変更を受け取る．パラメータを調整する場合であっても，仕様書の変更から行うことで，要求仕様を意識して設定する．
2. 仕様変更のあったドキュメントに基づいてパラメータを選択する．
3. 電圧などの仕様値を入力する．

4. パラメータが変更できていることを確認する.
5. MBD ツールへ自動入力する.

上記手順を, 2.2 節の開発プロセスおける P4 に組み込む. パラメータ設定ツール導入前後で, 表 1 のようにパラメータ設定作業が変化した.

4. 評価

4.1. 評価方法

パラメータ設定方式の有効性は, 実プロジェクトへのツール適用によって評価する. 定量的評価として, 従来のパラメータ設定作業に関する工数とパラメータ設定ツールを用いた場合の作業工数を計測・ヒアリングして比較評価する. さらに定性的評価として, 開発者へのヒアリングを実施しパラメータ設定ツールの効果や改善点のフィードバックを得る.

ツール適用前後での作業工数計測・ヒアリングは同じソフトウェア開発者へ実施する. また, ツール適用前後のプロジェクトは同じソフトウェアをベースとしているが, プロジェクトが異なるため, 変更量や変更するパラメータは異なる(同じパラメータも含まれる).

4.2. 評価結果

パラメータ設定ツール適用の工数削減効果を表 2 に示す. 表 2 の「ベースソフトウェア調査によるパラメータ選択」から「パラメータ設定検証」までは, 1 パラメータあたりの各作業時間を計測し, プロジェクトの中で変更する可能性が高いパラメータの数 (558 個)を乗算, 人日換算した値である. 「手順・環境構築」と「パラメータ設定不良対応」は, 各作業時間をヒアリングし抽出した値である. パラメータ設定ツール適用の結果, 1 プロジェクトあたりのパラメータ設定作業について 18.36 人日を要していたものが 1.44 人日となり, 従来に対して 94%の工数を削減した.

また, パラメータ設定ツールを使用した開発者へヒアリングを実施した結果, 表 3, 4 のようなフィードバックを得た. パラメータ設定の作業負荷削減や, 変更するパラメータ・参照ドキュメントの明示, 一度の入力によるパラメータ変更が実現できている旨の回答を得た. 一方で, ツールへの入力に関する改善や, パラメータ変更に伴う確認に関する機能拡張などの要望を得た.

4.3. 考察

従来に対して 94%の工数を削減できることから, ベースソフトウェアについてモデル内のパラメータ変更で派生製品を実現する場合には, パラメータ設定ツール適用が工数削減に有効であると言える. またパラメータ設定間違いは起きていないことがわかる.

課題 1 に対し, パラメータ設定者はトレーサビリティに基づきドキュメントを参照しつつパラメータを設定していることから, 不適切なパラメータ選択を防止しているとわかる. 課題 2 に対し, パラメータ設定ツールは変更するパラメータの明示や設定値入力に伴う自動計算により, 従来の暗黙知の関係を持つパラメータについて変更漏れを防止しているとわかる. 課題 3 に対し, パラメータ設定ツールは要求仕様などの設定値を元にパラメータを自動計算し, MBD ツールへ自動入力しており, パラメータの入力・計算間違いを防止しているとわかる. ただし, パラメータ設定ツールにはユーザビリティの観点で改善点があることが判明しており, 今後アップデートが必要である.

本論文では組込みシステムの1種についてパラメータの分析やツール開発を行った. 一方で本論文のアプローチは他の組込みシステムについても同様に適用可能であると考えており, 今後他の派生 MBD プロジェクトへの展開により効果を検証し, 手法の汎用化を図る.

5. まとめ

組込みソフトウェア開発では, 開発効率化のため MBD ツールを用いるほか, 派生開発によりソフトウェア資産を流用する. 特に制御ソフトウェアでは, ハードウェアの種類により派生製品が存在しており, 類似した機能を持つソフトウェアを開発することが多い. そこで派生開発を実現する方法として, ハードウェア仕様の異なる部分のパラメータ化によりモデル内で制御ロジックとハードウェア仕様を分離し, モデル内のパラメータを設定してモジュールの動作を変更することが考えられる. 一方でパラメータ設定においては, 不具合を起こさないため設定対象となるパラメータの適切な選択や, パラメータ数が多い場合

には設定の効率化が必要となる.

本論文では, パラメータ設定を適切かつ効率的に行うためのパラメータ設定方式を検討した. パラメータ設定方式では, 仕様書とパラメータ間のトレーサビリティに基づき, 仕様に対して変更すべき適切なパラメータを選択することで派生開発を実現する. またこれを実現するためパラメータ設定ツールの開発に取り組んだ. パラメータ設定ツールにおいては, 各パラメータと仕様書のトレーサビリティや暗黙知となっているパラメータ間の関係を明確化することで, パラメータの設定間違いを防止した. さらに, 自動計算や MBD ツールへの自動入力によりパラメータの設定工数を削減した. パラメータ設定方式の有効性を確認するため, ツールを実プロジェクトへ適用した結果, パラメータは適切に設定できており, パラメータ設定工数の 94%が削減可能であることがわかった. したがって, ベースソフトウェアを選定し, モデル内のパラメータ値を設定して動作を変更する場合には, 本論文で提案するパラメータ設定方式が工数削減や不具合防止に有効であると言える.

今後の課題として, 派生 MBD におけるパラメータのトレーサビリティ管理方法を検討することを挙げる. 本論文では, 派生 MBD 適用に向けたパラメータ設定方式を実現するため, 実際の製品システムについてパラメータを分析しトレーサビリティを明確にした. しかし, システムのさらなる機能高度化や試験結果に応じ新たなパラメータが拡充される場合には, 拡充されたパラメータのトレーサビリティの精度や粒度などにばらつきが発生し, その後の派生製品の品質への影響が見込まれる. そのため, パラメータのトレーサビリティについて管理方法を検討し, 後の派生製品の品質を維持していくことを考える.

表 2　1 プロジェクトあたりのパラメータ設定ツールの適用効果(人日)

作業項目	従来	ツール適用後	効果
ベースソフトウェア調査によるパラメータ選択	3.00	0.10	−2.90
パラメータ設定根拠調査	3.00	0.10	−2.90
パラメータ計算	1.20	0.02	−1.18
MBD ツールへ入力	1.55	0.00	−1.55
パラメータ設定検証	6.00	1.20	−4.80
手順・環境構築等情報共有	0.52	0.02	−0.49
パラメータ設定不良対応	3.10	0.00	−3.10
計	18.36	1.44	**−16.92**

表 3 パラメータ設定ツールの良かった点

開発者	良かった点
A	・パラメータ設計作業が楽になった. ・チェック/変更するパラメータがツール内での明示により削減されている.
B	・参照するドキュメントが示されているため探す手間が省ける.
C	・パラメータが一覧化できる. ・試験中にパラメータの一括変更をやりやすくなる.

表 4 パラメータ設定ツールの改善点

開発者	改善点
A	・パラメータ設定部分が見づらい.
B	・ある時点のパラメータと現在のパラメータをツール内で比較したい.

参考文献

[1] 経済産業省, “2012 年版ものづくり白書 第 2 章 我が国ものづくり産業が直面する課題と展望” 2012.

[2] Pohl, K., Böckle, G. and van der Linden, F. J.: Software Product Line Engineering, Springer-Verlag(2005). 林 好一, 吉村健太郎, 今関 剛(訳):ソフトウェアプロダクトラインエンジニアリング ソフトウェア製品系列開発の基礎と概念から技

法まで. 株式会社エスアイビー・アクセス, 2009.
[3] 筒井賢, “プラットフォーム開発, XDDP 開発, そしてSPL 開発へ 〜 移行のきっかけ(取り巻く環境)と成功/失敗のステップ 〜”, 第4回 アフォード・フォーラム, 2013.
[4] Dziobek, Cristian. “Functional variants handling in simulink models.” MathWorks Automotive Conference 2008, 2008.
[5] Polzer, Andreas, Stefan Kowalewski, and Goetz Botterweck. "Applying software product line techniques in model-based embedded systems engineering." 2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software. IEEE, 2009.
[6] Liebel, G., Marko, N., Tichy, M., Leitner, A., & Hansson, J. “Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice.” Software & Systems Modeling, 17(1), 91-113, 2018.
[7] IPA, “組込みシステムの先端的モデルベース開発実態調査報告”, 2012.

シェルスクリプトを用いたUNIX哲学に基づくリアルタイム制御

柳戸 新一
（有）USP 研究所
s.yanagido@gmail.com

松浦 智之
（有）USP 研究所
richmikan@richlab.org

鈴木 裕信
（有）USP 研究所
h-suzuki@usp-lab.com

大野 浩之
金沢大学
hohno@staff.kanazawa-u.ac.jp

要旨

ソフトウェアを中心としたシステムの移植性・持続性は重要な観点である．*Gancarz*の*UNIX*哲学においてはハードウェアに依存する高性能なシステムより，多少性能が劣っても移植性に優れたシステムの方が将来的に淘汰されにくいとされる．筆者らはこの*UNIX*哲学に基づき，移植性・持続性を最大まで高めるために，シェルスクリプトのストリーム指向を活用した開発手法を提案してきた．これらの手法はサーバやものづくりをはじめとする幅広い分野で適用され，その有効性が実証されてきた．

一方，これまでリアルタイム制御を必要とする組込み型システムで*GNU/Linux*を用いる場合，低遅延カーネルを利用したり，スケジューラのスケジューリングポリシーをリアルタイム（*RT*）クラスに変更するなどの工夫が行われてきた．しかし，スケジューラの変更はマルチタスク処理を活かしたストリーム指向プログラミングと相性が悪く，並列処理が実施されなくなるといった危険性がある．

本研究ではリアルタイム制御を行うシステムに対しても*UNIX*哲学を適用可能であることを示すために，標準スケジューラで動作するシェルスクリプトを用いた倒立振子の安定化を試み，評価を行った．検証の結果，本手法に一定の実用性があることが実証された．

1. はじめに

操作対象となるシステムを電子回路を用いて計測ならびに制御を行う場合，コンピュータを使ったデジタル処理が広く行われている．このコンピュータにOSを搭載し，OSにUNIX系OSを用いることにより，次のようなこれまでにない利点が得られる．まず豊富なUNIXコマンドを用いたデータの利活用が挙げられる．また，多くのUNIX系OSは既にTCP/IPスタックを持っているので，インターネットと連携するIoT機器の製作も容易に可能となる．そして筆者らが考える最大の利点は，数十年以上に渡って積み上げられてきたUNIXの高互換性・持続性により，UNIX上で制御機構を実装すれば，そのシステムやソフトウェア資産の寿命が何十年に渡ると期待できる点である．

しかし，UNIX環境ではタスクスケジューラの影響でプロセスの実行タイミングが正確に予測できず，リアルタイム制御に支障をきたしてしまう．これまでリアルタイム制御を必要とする組込み型システムでGNU/Linuxを利用する場合，実行時のプロセスのプライオリティ値を変更したり，あるいはスケジューラのリアルタイム・ポリシーをSCHED_FIFOに設定するなどしてプロセスの優先度を高めるといった手法を用いられてきた[4]．また，それに加えて利用するLinuxカーネルを低遅延カーネルに置き換えるなどの工夫が必要となる[5]．このようにプロセスに優先度を設定する場合，どれだけの優先度を与えれば良いか，また優先度を与えたときに，他のプロセスにどのような影響を与えるかを考慮に入れなくてはならない．

そこで本論文では，フィルタや制御量計算といった自動制御で用いられるデータの連続的な変換を，UNIX が持つパイプラインの仕組みで実現する手法を提案する．このとき，スケジューラはディストリビューションの標準である Completely Fair Scheduler（CFS）の設定のまま使い，優先度をあげるといった処理は行わないことを前提としている．これは，Gancarz の UNIX 哲学 [3]（以下，UNIX 哲学）に基づいたリアルタイム制御のための手法であり，以降で本手法の有効性を検証する．

Gancarz は UNIX の考え方として，9 個の定理と，10 個の小定理を示している．その中で本論文の開発手法が従う定理と，その定理を満たすために採用した手法との対応ついて次に示す．

定理 1 (小さいものは美しい).
対応 : すべての処理を行う一つのプログラムを作成するのではなく，機能ごとにプログラムを作成.

定理 2 (一つのプログラムには一つのことをうまくやらせる).
対応 : 適材適所の考え方を導入した，ものグラミング 2[12] に基づいたハードウェア構成.

定理 4 (効率より移植性を優先する).
対応 : 移植性の高い開発手法である POSIX 中心主義 [18] に基づき，加えてスケジューラは変更を加えない標準のものを使用.

定理 5 (数値データは ASCII フラットファイルに保存する).
対応 : プログラム中のどこに流れているデータであっても即座に保存できるよう，プログラム間のやり取りは数値データであっても ASCII データを利用.

定理 7 (シェルスクリプトによって梃子の効果と移植性を高める).
対応 : 各プログラムはコマンドの形式を取り，シェルスクリプトを用いて協調して動作.

定理 9 (すべてのプログラムをフィルタとして設計する).
対応:シェルスクリプトのストリーム指向を活用した設計.

小定理 6 (同時に考える).
対応 : シェルスクリプトのストリーム指向を活用した並列処理.

2. 関連研究

まず，筆者らが提案してきた UNIX 哲学に基づく手法について述べる．松浦らはプログラムの移植性向上を目的とした標準化規格である POSIX.1(IEEE Std 1003.1) 文書に記載された仕様に極力準拠しつつ，さらなる互換性向上のためのプログラミング手法，POSIX 中心主義を提唱し，その有効性を実証してきた [18]．また，大野らはこの手法をものづくりに応用し，POSIX に準拠した Raspbian（現 Raspberry Pi OS）を搭載可能な Raspberry Pi[6] に加え，必要に応じてセンサ・アクチュエータとのインターフェースとなる Arduino[1] を使用した，ものグラミング 1[13] ／ものグラミング 2[12] を提唱し，普及活動を行っている．

また，本論文での報告内容は筆者らの一連の研究の一部であり，試行錯誤によって得られた他の結果については松浦らの報告 [19] を参照されたい．

Linux をリアルタイム処理で活用する試みは数多く行われており，先に述べたプロセスの優先度変更や低遅延カーネルでの利用の他にも，複数の手法が存在する．リアルタイム処理を行うための Linux カーネルとしては RTLinux[10] や ART-Linux[11] が知られている．また，清水らは 2010 年にカーネルの変更に加えて POSIX の範囲内でリアルタイム処理を行うための枠組みを提案している [16].

熊谷らは 2004 年に通常の Linux（カーネルバージョン 2.4 以前）を用いたリアルタイム制御と RTLinux によるリアルタイム制御の違いついて述べ，標準の Linux を用いる利点としては開発の容易さと互換性の確保を挙げている [14]．また，標準の Linux 環境における周期的なプロセスの実行のために，`usleep` 関数を用いる方法を提案しており，RTLinux を用いなければならないほどの精度でなければ `usleep` 関数でほぼ周期的な処理が可能な場合もあることを示している．

3. 提案手法

3.1. シェルスクリプトのストリーム指向を利用した制御

UNIX 哲学で挙げられている定理を満たすために，シェルスクリプトを用いたシステム制御を提案する．シェルスクリプトで推奨されるストリーム指向のプログラミング手法は多くの利点を持つ．その利点は次の通りである．

- シェルスクリプトおよびそれに使用するためのコマンドは，持続性・移植性が高いといわれる POSIX[7] の範囲で記述可能である．これにより将来的なシステムのアップデート等で動かなくなるというリスクを，最小限に抑えられる．

- 容易に変更・拡張が可能である．シェルスクリプトはコマンドをパイプで連結して記述する．このことから各コマンドの変更を行っても他のコマンドの動作に影響を与えることは少なく，新たな制御アルゴリズムの導入もコマンドの置き換えで済ませられる．

- 容易にデバッグ，ロギングができる．任意のコマンド間に tee コマンドを挟むことによってそこに流れるデータをログとして保存できる．加えて，ストリーム指向で記述することにより，パイプを延長するようにプログラミングができ，流れるデータを確認しながら開発を進められる．

- セキュリティホールとなる危険性が比較的少ない．UNIX 環境という下地の上で動作するので，シェルスクリプトそれ自体がセキュリティホールとなる可能性が少ない．UNIX 環境は一般に広く使用されており，セキュリティ強度はそれに準ずる．

また，シェルスクリプトのストリーム指向は，制御工学におけるブロック線図と相性がよい．これはどちらもデータの流れを記述するという点で共通であるためである．ブロック線図における伝達要素，分岐点，加え合わせ点はシェルスクリプトにおけるコマンドに対応付けることができ，信号線はパイプに相当する．

図 1 は本システムで扱うフィードバック系のブロック線図である．図中の破線はシェルスクリプトでの対応付けを示している．制御対象 G はシリアルインターフェースで接続されることを想定し，入出力には cu コマンドを介する．cu コマンドは 1980 年代初頭の UUCP 時代に開発されたコマンドの一つで，UNIX シェルからシリアルインターフェースを操作する機能を持つ．本研究ではこの cu コマンドをシリアルインターフェースと標準入力の相互接続に利用した．分岐点は tee コマンドを使用し，一方の観測値ベクトル $\boldsymbol{y}$ をログとしてファイルに落とし，他方に名前付きパイプ named_pipe を用いることでフィードバックを実現する．制御器 controller は加え合わせ点と C を包含させている．これは PID 制御のような目標値 r が必要な制御では，r をその制御コマンドのパラメータとして設定するためである．controller では内部で偏差 $\boldsymbol{e}$ を計算し，それを $\mathbf{0}$ に近づけるよう制御を行う．得られるセンサ値を $\mathbf{0}$ に近づけるような制御の場合は，この r は与えない，もしくは 0 を与えるという方法を取る．また controller の入力に制御量 u が必要な場合，簡単化のために u は $\boldsymbol{y}$ に含めることにする．

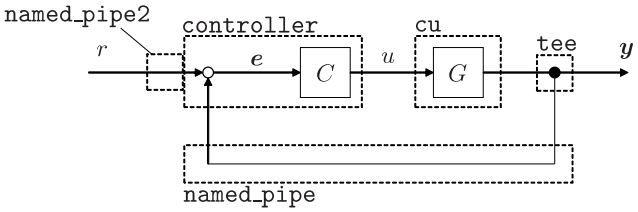


図 1. 本システムのブロック線図とコマンドの対応

図 1 の構造をシェルスクリプトのワンライナーで表現すると次のようになる．

```
$ cat named_pipe | controller -f named_pipe2 |
 cu -l /dev/G | tee named_pipe > y.output
```

名前付きパイプ named_pipe によってデータの循環が表現できていることがわかる．名前付きパイプを使用するためには，mkfifo コマンドを用いてあらかじめ名前付きパイプを作成しなくてはならない．また実際には，controller は制御用のコマンドに置き換える．加えて，データの変形や列の入れ替えのために，コマンド間に別のコマンドを適宜挿入する．

シェルスクリプトでリアルタイム処理を行うときに問題となるのが，各コマンドのバッファリングである．バッファリングはコマンドを高速に動作させるために実装されている機能であり，これによってある程度出力データが溜まるまで，データは出力されずに内部で保持される．しかし，リアルタイム処理ではデータを処理する時間に意味があるため，このバッファリングがリアルタイム制御の妨げとなってしまう．C 言語をはじめとする多くの言語の標準入出力では，コマンドの標準出力が端末ではなくパイプに接続されると，標準でフルバッファリングが有効になる．これはプログラム内部で設定を変更すれば無効化できるが，既存のコマンドを使用する場合，このような方法を取ることができない．そのため，フルバッファリングをコマンドの外部から無効化する方法が必要となる．

フルバッファリングを無効化するために，今回は ptw

コマンド [9] を用いた．バッファリングを行う例として，awk コマンドや Python スクリプトが挙げられる．awk の実装の一つである gawk にはバッファ内のデータを出力するための関数 fflush がある．しかし mawk など，他の awk 実装では使用できないという問題がある．バッファリングを無効化するために stdbuf コマンドを使用することも可能だが，Python スクリプトは内部で独自にバッファを管理しており，stdbuf も汎用的な手段ではない．一方，ptw コマンドは対象となるコマンドに，パイプではなく端末が接続されていると判断させ，フルバッファリングを無効化させようとする．このやり方は汎用的であり，バッファリングを無効化させたいコマンドを ptw でラッピングさせるだけと使用も簡単であるため，この方法を用いた．

この一連のスクリプトは，ディストリビューションの標準スケジューラを用いて動作させるが，一般には組込みのようなリアルタイム性が重視される場合には専用のスケジューラが使用されることが多い．例えば，低遅延カーネルを選択する，プロセススケジューリングをリアルタイムに設定する，あるいはプライオリティを優先するなどの手法がそれにあたる．しかし，プロセスに優先度をつけることは，優先度の低いプロセスが処理されない事態が発生する可能性がある．今までこのリアルタイム性とのトレードオフは，組込み Linux の一つのリスクであった．

本研究では実験時点で最新の Linux カーネル 4.19.66 のデフォルトスケジューラである CFS を用いる．CFS は，全てのプロセスが同じ時間だけリソースを取得するように設計されたスケジューラである．CFS は Linux カーネルバージョン 2.6.23 から使用されるようになったスケジューラであり，設定を変更しない場合には標準でこれが使われる．特別なスケジューリングを行わずに制御が可能であるならば，前述のリスクを負わなくていいという点で優れた手法である．

3.2. 倒立振子の安定化による検証

上記のような手法でリアルタイム制御が可能となるのかという検証のために，今回は本手法を用いた同軸 2 輪車型倒立振子の安定化を試みた．

倒立振子とは重心が支点よりも高いところに位置する振り子のことをいう．同軸 2 輪車型は，2 つの車輪が回転軸を共有し，その軸を中心に動く上部の構造体（以降では本体）を軸上で支えるという構造である．重心が支点よりも上部にあるため，倒立状態を維持するには何らかの制御により安定化させる必要がある．このために必要な制御機構は，駆動のための電源を含めて，全て本体に搭載する．

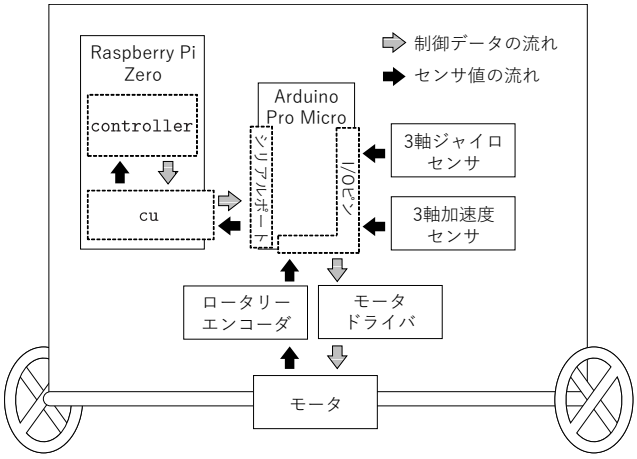


図 2. 倒立振子のシステム構成

3.3. 倒立振子のシステム構成

本実験で用いる倒立振子の構成を図 2 に示す．本システムは，UNIX 環境としての Raspberry Pi Zero（Raspbian）と，センサ・アクチュエータを統括するための Arduino Pro Micro からなる．Arduino は複数の I/O ポートを持ち，センサやアクチュエータとのやり取りを得意とする．一方で制御量の計算のような複雑な処理は Raspberry Pi で行う．Arduino と Raspberry Pi のこのような役割分担はものグラミング 2 の考え方に基づく．

ものグラミング 2 はマイコンを含むシステムに対して，適材適所の考え方を導入したものである．センサやアクチュエータは使用するためにいくつもの通信規格があるため，マイコン（Arduino）はそれら値の取得や制御に専念する．一方でセンサ値の複雑な変換処理は，豊富なリソースや柔軟な拡張が可能な UNIX 環境（Raspberry Pi）で行う．

Arduino に接続するセンサは姿勢を検出するための 3 軸加速度センサ，3 軸ジャイロセンサ，モータの回転量を測るためのロータリーエンコーダである．アクチュエータはモータを制御するためのモータドライバを使用する．Raspberry Pi とは USB で接続し，センサの値は一定間隔でシリアルに出力する．このときの出力は，各

行が特定の時刻における観測値となるような，フィールド形式 [8] をとる．つまり，ある時刻におけるセンサ値は決められた順番で，一行の空白区切りで出力される．これはフィールド形式が出力先のシェルスクリプトで処理しやすいフォーマットであるためである．シリアルから Arduino へと入力された数値データも同様にフィールド形式をとり，これはモータの速度として解釈し反映させる．

Raspberry Pi では，前述の通り，cu コマンドを通して Arduino と通信を行う．cu からの出力はセンサの値であり，図 1 に示した通り，名前付きパイプを通して制御器 C へ入力される．そして制御器で制御量を求めた後，それを cu に入力し，Arduino にモータ回転速度の指示を送る．ここまでで一連のフィードバック制御となる．

制御器には別府（2019）と同様に，線形 2 次ガウシアン（Linear Quadratic Gaussian，LQG）制御を用いる．これは制御アルゴリズムである線形 2 次レギュレータ（Linear Quadratic Regurator，LQR）にノイズ除去のための線形カルマンフィルタ（Linear Kalman Filter，LKF）を加えたフィードバック制御を指す．LKF は，センサから得られた一連のデータから，より正確な機体の状態を推定し，観測値に含まれたノイズの除去を行う．LQR は LKF で得られた機体の状態をもとに，制御量を計算する．

これら LQR，LKF の処理は UNIX 機で行う．しかし，それぞれ行うことができるコマンドが存在しないため，実装を行った．どちらのコマンドも計算に必要なパラメータが存在する．例えば，システムの状態方程式 $\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}$ やセンサのサンプリング間隔 Δt があげられるが，必要なパラメータは全てコマンドのオプションで与える．従って，次のようなワンライナーで制御を行う[*1]．

```
$ cat named_pipe | kalfltr <options> |
 lqreg <options> | cu -l /dev/G |
 tee named_pipe > y.output
```

ここで，kalfltr が LKF コマンド，lqreg が LQR コマンドを表す．また，それらのコマンドで必要となるオプションは上記では<options>と省略しているが，実際には適切に設定する必要があることに注意されたい．これらのコマンドは流れるデータのフォーマットに合わせるため，標準入力から 1 行入力されるごとに出力値を計算し，その出力値をフィールド形式で 1 行に出力するように設計した．また，列の入替えや単位の変換のため，コマンド間に適宜 awk 等のコマンドを用いる．このとき，フルバッファリングに注意し，使用の際は ptw awk <args>のように，ptw コマンドでラップする．

[*1] 倒立に用いた実際のスクリプト：⟨https://github.com/NCNT/HerFirstSteps⟩

表 1. 倒立振子制御のためのパラメータ

パラメータ	値
Q	$\mathrm{diag}(1\ /\mathrm{rad}^2, 1\ /\mathrm{rad}^2 \cdot \mathrm{s}^{-2}$ $, 1000\ /\mathrm{rad}^2, 10\ /\mathrm{rad}^2 \cdot \mathrm{s}^{-2})$
R	$1000000\ /\mathrm{V}^2$
Δt	10 ms, 20 ms, 50 ms, 80 ms, 100 ms

4. 倒立振子制御の実験

本提案手法に基づいたリアルタイム制御が可能であるという実証のために，倒立振子を安定化させることを試みた．機体の設計および制御量計算のアルゴリズムは別府（2019）を基にした [17]．相違点は NUCLEO-F401RE の代わりに，Arduino Pro Micro と Raspberry Pi Zero を用いているという点である．また，それに伴った電源の変更等を行った．

実験は LKF の有無という 2 通りの制御方法によって倒立状態の安定化を試みた．比較はそれらのログ（前述したワンライナーでいう y.output）を解析して行う．

4.1. パラメータの設定

一連のシステムを動作させるために決定するパラメータは別府（2019）とほぼ同一とするが，異なる点に関しては表 1 に示す．ここで，状態ベクトル $\boldsymbol{x}$，入力ベクトル $\boldsymbol{u}$ に対して，LQR の評価関数を

$$J = \int_0^\infty \left(\boldsymbol{x}^\top Q\boldsymbol{x} + \boldsymbol{u}^\top R\boldsymbol{u}\right) dt$$

とする．また表中の $\mathrm{diag}(c_1, c_2, \ldots, c_n)$ は，(i, i) 成分が c_i であるような対角行列を表す．

実験ではサンプリング間隔 Δt を変化させ，倒立振子の安定性を評価する．

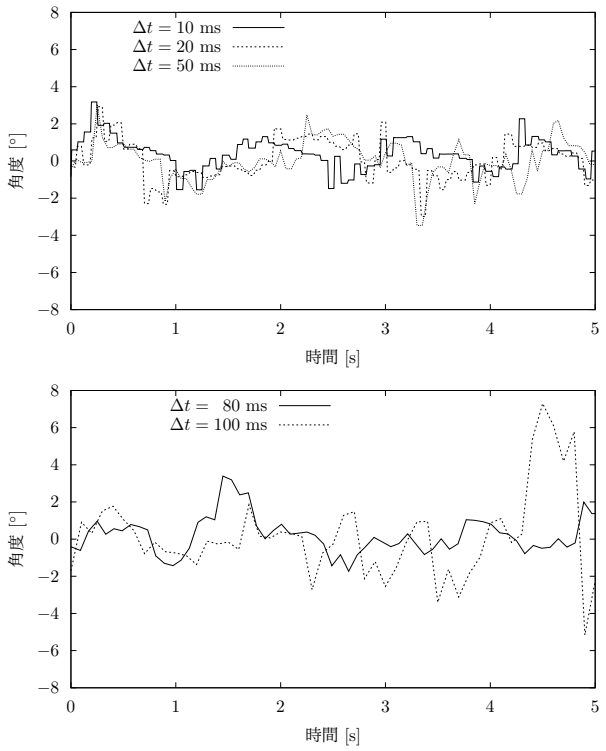


図 3. サンプリング周期を変化させた際の倒立振子の傾き推移

4.2. 倒立振子の安定性

サンプリング周期 Δt を変化させ，倒立振子の傾きの推移を測定した結果，図 3 のようになった[*2]．ここで，0° が直立状態となるように縦軸を調整し，横軸は倒立振子が自立した時刻を 0 としている．図は倒立後 5 秒間のグラフだが，Δt が 80 ms 以内であれば本倒立振子は安定し，少なくとも 1 分間倒立状態を維持することが確認できた．$\Delta t = 100$ ms のときは 4 s 以降に不安定になり倒れてしまった．

図を見ると，$\Delta t \leq 50$ ms のとき，Δt の減少に対して，倒立振子の傾きの周期が短くなっており，サンプリング間隔を短くすることが倒立振子の素早い制御に寄与していることが見て取れる．安定して倒立したとき（$\Delta t \leq 80$ ms のとき）は振幅はどれも 4° 以内であり，サンプリング周期が振幅に与える影響は見られなかった．

$\Delta t = 0.01$ ms とし，LKF を使用しない場合の，倒立振子の挙動を図 4 に示す．LKF がない場合にはセンサのノイズを除去することができず，動作が不安定になり，5 秒持たず倒れてしまったことが見て取れる．

[*2]倒立検証時の動画：〈https://www.youtube.com/watch?v=Ixh-YFtMypY〉

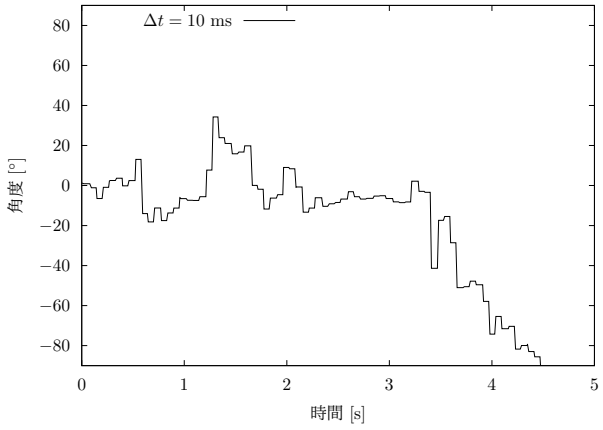


図 4. 線形カルマンフィルタがない場合の倒立振子の傾き推移

5. 考察

5.1. センサの値が不定期に出力される場合のリアルタイム処理

本研究で採用したシステムでは観測値の取得に Arduino を使用しており，内部のクロックを用いた等間隔の観測が可能であった．しかし，ネットワーク越しにデータを取得する場合などはこのように等間隔でデータが得られることは少ない．例えば，近年では IoT におけるリアルタイム通信のために軽量なプロトコルである MQ Telemetry Transport（MQTT）を用いることも多い [15]．しかし MQTT を使用する場合，通信による不規則な遅延や，Quality of Service（QoS）の設定によってはデータが欠損する可能性も考えられる．本システム構成のままではこのような事態に弱い．なぜならばデータが等間隔に到来するという仮定の下で，離散化したフィルタリング・制御量計算を行っているためである．

しかし，このようにデータが得られる間隔が不均一の場合であっても本手法を適用することができ，そのために `linets` コマンド [9] を使用する．これは標準入力から与えられた各行に対し，その行が入力された時間を付加し出力するコマンドである．観測値が得られた直後，

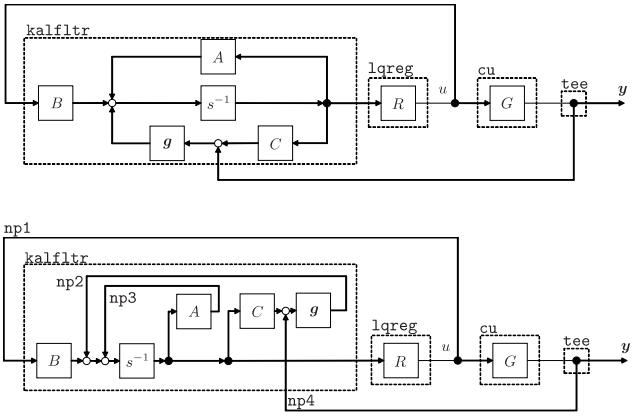


図 5. LKF を詳細に記述したブロック線図（上）とそれをシェルスクリプト適用のために書換えたもの（下）

具体的には cu コマンドの直後に linets を追加することにより，観測値をタイムスタンプ付きで後続するコマンドに渡すことが可能となる．このように linets を使用することで観測値が得られる間隔のばらつきに強いシステムを構成することが可能になると考える．

5.2. 他のブロック線図への適用

今回は図 1 の単純なフィードバック制御を実装した．しかし，このように単純化された状況は少なく，一般にはより複雑な，複数のフィードバックが混在した制御が行われる．このような場合であっても本手法が適用可能であることを示す．

図 5 は図 1 と本質的に等価だが，LKF の内部処理をより詳細に記述したブロック線図である [2]．コマンドはモジュール毎に作成されるため，さらに要素を分解し，それぞれ独立にコマンド化することは試作段階を除いて少ないが，これだけ複雑なブロック線図であっても本提案手法が適用可能であるということを示す．

まずは，次の記述方法を定める．

- 図 5 中の各ブロック内の文字を等幅フォントに直し，それをコマンド名とする．例えば A の処理をコマンド化したものを A，s^{-1} をコマンド化したものを s^-1 で表す．
- np*を名前付きパイプとする．例えば np1，np2 は名前付きパイプである．

```
$ cat np1 | B | add np2 | add np3 | s^-1 |
> A -a | tee -2 np3 | cut -d ' ' -f 1 |
> C -a | add np4 | g |
> tee -2 np2 | cut -d ' ' -f 1 |
> lqreg | tee np1 | cu | tee np3 >y.output
```

図 6. kalfltr をより単純なコマンドで置き換えた場合のワンライナー

- 各コマンドは-a オプションがない場合，入力された最終列に対して処理をし，その列を書き換える．
- 各コマンドに-a オプションを付けた場合，最終列に処理した結果をその後ろに付け加える．
- 分岐点に対応する tee コマンドに新しいオプション-n を定義する．これは n 列目を引数で与えた名前付きパイプに書き出す．
- 加え合わせ点に対応する add コマンドを導入する．これは引数で与えられたデータを標準入力から与えられたデータの最終列に加算して出力する．

このルールの下で図 5 をストリーム指向のシェルスクリプトで記述すると，図 6 のようになる．ここで，lqreg および cu のオプションは省略した．また，cut コマンドはフルバッファリングを行うため，実際には ptw コマンドの使用が必要だが，ここではそれを省略した．

スクリプトの構成法は次のとおりである．例えば，図 5（下）ではブロック s^{-1} からの出力はその直後の分岐点でブロック A とブロック R の方向に分岐している．このように並列なデータの流れはストリーム指向で記述することが難しいので，A の出力と R 方向のデータを一旦合流させる（図 6，2 行目の A -a）．その後，A の出力のみをフィードバックし（tee -2 np3），R（および C）の入力に不要な A の出力を取り除く（cut -d ' ' -f 1）．このような構成法により，並列に流れるデータをストリームという 1 次元上で扱うことができ，図 5 のような複雑なブロック線図であっても，図 6 のようにシェルスクリプトのワンライナーで表すことができる．

この構成法によって，図 1 のような単純なフィードバック制御システムに限らず，任意のブロック線図がシェルスクリプトのワンライナーで表現可能であると考えられる．

6. おわりに

本研究ではソフトウェアの移植性や持続性をこれまで以上に高めるべきという筆者らのこれまでの主張に基づき，シェルスクリプトのストリーム指向を利用したリアルタイム制御の実装手法を提案し，本手法の検証を行った．結果として，提案手法を用いて同軸 2 輪車型倒立振子の安定した倒立を LQG 制御で達成することができた．これにより，制御システムに特化した UNIX 環境でなくても，シェル上でコマンドをパイプでつなぐという UNIX 的な手法がリアルタイム制御にも有効であることが示された．

本研究では倒立振子の安定化のみ行った．しかし，倒立振子自体は外部からの指示による前後の移動に加え，2 輪を非同期に動作させれば平面上の自由な移動や坂の上り下りができることが知られている．このようなより複雑な状況下でも本手法が適用できることが示せるよう研究を継続すると共に，ドローンや 2 足歩行の制御も行えるよう本手法の検証を進める．加えて，本手法が適用可能であるためのリアルタイム性に関する条件も調査していく．

謝辞

本論文執筆にあたり，USP 研究所の矢嶋遼氏，並びに北嶋完基氏には倒立振子の開発補佐や動画の作成をしていただいた．また，矢嶋氏，北嶋氏をはじめとする同社研究部門の方々との議論は有意義なものであり，研究の示唆を得ることができた．本研究は USP 研究所と金沢大学総合メディア基盤センターの共同研究の一環として進められたものであり，USP 研究所の當仲寛哲代表には研究環境および研究資金をいただき，滞りなく研究をすすめることができた．ここに記して感謝する．

参考文献

[1] Arduino Foundation “Arduino”, https://www.arduino.cc/（2021 年 5 月 20 日閲覧）.

[2] Lachhab, A., A. Ed-Dahhak, and M. Guerbaoui “Synthesis of an optimal dynamic regulator based on linear quadratic Gaussian (LQG) for the control of the relative humidity under experimental greenhouse”, *International Journal of Electrical and Computer Engineering*, vol. 6, no. 5, pp. 2262–2273, 2016.

[3] Gancarz, M. (芳尾桂 訳) “UNIX という考え方”, オーム社.

[4] Garg, A. “Real-Time Linux Kernel Scheduler”, Linux Journal, 2009. https://www.linuxjournal.com/article/10165（2021 年 3 月 15 日閲覧）.

[5] Jones, M. T. “Anatomy of real-time Linux architectures”, BM developerWorks, 2008. https://www.ibm.com/developerworks/library/l-real-time-linux/index.html（2021 年 3 月 15 日閲覧）.

[6] Raspberry Pi Foundation “Teach, Learn, and Make with Raspberry Pi”. https://www.raspberrypi.org/（2021 年 5 月 20 日閲覧）.

[7] The Open Group “The Open Group Base Specifications Issue 7, 2018 edition”. https://pubs.opengroup.org/onlinepubs/9699919799/（2021 年 3 月 15 日閲覧）.

[8] USP 研究所 “Tukubai オンラインコマンドマニュアル”, https://uec.usp-lab.com/TUKUBAI_MAN/CGI/TUKUBAI_MAN.CGI?POMPA=MAN5_field-format, 2021 年 3 月 15 日閲覧.

[9] USP 研究所 NCNT プロジェクト “Timing Management Commands in POSIX”, GitHub. https://github.com/NCNT/TimingCmds_in_POSIX（2021 年 3 月 15 日閲覧）.

[10] Yodaiken, V. “The RTLinux Manifesto”, *Proceedings of the 5th Linux Expo*, 1999.

[11] 石綿陽一, 松井俊浩, 国吉康夫 “高度な実時間処理機能を持つ Linux の開発”, 日本ロボット学会学術講演会予稿集, vol. 16, no. 1, pp. 355–356, 1998.

[12] 大野浩之, 松浦智之, 森祥寛 “ものグラミング 2 - 諸機能の選択と集中を徹底した POSIX 中心主義に基づく IoT 開発方式の提案”, 研究報告インターネットと運用技術（*IOT*）, vol. 2019-IOT-46, no. 16, pp. 1–8, 2019.

[13] 大野浩之 他 “ものづくりのための「ものグラミング」と実践的教育環境の構築”, **マルチメディア，分散，協調とモバイルシンポジウム** *2016* **論文集**, vol. 2016, pp. 1335–1340, 2016.

[14] 熊谷正朗，森友一朗，橋本浩一 “Linux 系 OS によるリアルタイム制御”, **計測と制御**, vol. 43, no. 6, pp. 515–520, 2004.

[15] 佐々木大知, 福田浩章, 高橋和也 “製造業における製品品質のオンラインリアルタイム診断ツール”, **研究報告ソフトウェア工学（***SE***）**, vol. 2020-SE-204, no. 16, pp. 1–6, 2020.

[16] 清水正晴, 戸田健吾, 林原靖男, 大和秀彰, 古田貴之 “Linux 標準機能を利用した RT ミドルウェア周期実行機能のリアルタイム化”, **計測自動制御学会論文集**, nol. 46, no. 1, pp. 16–23, 2010.

[17] 別府伸耕 “特集　月着陸船アポロに学ぶ確率統計コンピュータ”, **トランジスタ技術** *2019* **年** *7* **月号**, 2019.

[18] 松浦智之, 大野浩之, 當仲寛哲 “ソフトウェアの高い互換性と長い持続性を目指す POSIX 中心主義プログラミング”, **デジタルプラクティス** vol. 8, no. 4, pp. 352–360, 2017.

[19] 松浦智之, 柳戸新一, 大野浩之 “UNIX 機における IoT 機器制御のためのタイミング管理”, **ソフトウェア・シンポジウム** *2021*（投稿中）, 2021.

UNIX 機における IoT 機器制御のためのタイミング管理

松浦 智之
(有)USP 研究所
richmikan@richlab.org

柳戸 新一
(有)USP 研究所
s.yanagido@gmail.com

鈴木 裕信
(有)USP 研究所
h-suzuki@usp-lab.com

大野 浩之
金沢大学
hohno@staff.kanazawa-u.ac.jp

要旨

さまざまな電子デバイスを *IoT* 機器として活用するためには，*UNIX* 系 *OS* 搭載コンピュータ（以降，*UNIX* 機）と接続・連携できると都合が良い．なぜなら，*UNIX* 機は *TCP/IP* スタックを既に持っていて，かつ広く普及してるなど，開発や保守にかかるコストを抑えられるという期待が持てるからである．しかし，*UNIX* 機からそれら電子デバイスを直接制御する場合には，解決しなければならない課題が多い．例えば，プリエンプティブなタスクスケジューラや，パイプライン上のバッファリング機能などにより，制御にとって重要な「タイミング」を損なう要因が *UNIX* にはいくつかある．また，*POSIX*（*POSIX.1-2017*）の範囲では精密なタイミング管理のためのコマンドも十分に揃っていない．

そこで著者らは，タイミングの変動を低減あるいは管理可能にするためのアイデアと，それに基づくコマンドを考案した．ただし，*POSIX* 仕様を逸脱しない範囲の *UNIX* 機（*POSIX* 機と称す）で実現するという制約を課した．逸脱するほど *UNIX* というプラットフォームから外れていき，*UNIX* が本来持っている高い汎用性や持続性という恩恵が得られなくなっていくからである．そして，新たに作成したタイミング管理コマンドを用い，*POSIX* 機での *PID* 制御による倒立振子を実現するなど，いくつかの研究で実用性が確認できたため，*POSIX* 機でのタイミング管理実現のための考え方，および作成したコマンド実装について本論文で報告する．

1. はじめに

2021 年現在，産業用機器から家電に至るまで，UNIX 系 OS 搭載コンピュータ（以降，UNIX 機[*1]）は組み込み機器の分野においても広く活用されている．これは，UNIX の系譜がさまざまに分派した後に POSIX という共通インターフェース仕様が提唱され，POSIX は無償かつ改訂も少なく準拠しやすかったことから，普及率の高いプラットフォームになれたことが大きな要因である．さらに，今や UNIX 機は TCP/IP スタックを標準的に実装しているため，UNIX 機を活用することで IoT 機器を開発することも容易になった．しかし，UNIX 機を活用したことで UNIX 機が持つ高い互換性や持続性がシステム全体として発揮されるようになったとは言いきれない．

文献 [1] には工場のネットワーク構成図の一例が示されているが，同様にして UNIX を活用した IoT 機器について考えると典型的に図 1 の (a) のような構成になっている．UNIX 機はインターネットクラウド上に存在するサーバから受けた操作指示をローカルデバイス機器に伝達したり，あるいはローカルデバイス機器から得られた動作状況をサーバに報告する役割を担ったりという，サーバとローカル機器の橋渡し役を担うのみでデバイス機器の制御自体は担当しない．UNIX 機と末端のセンサやアクチュエータの間には別途コントローラが介在し，計測と制御を担当する．そのコントローラ自体もまたプロセッサを搭載したコンピュータであり，別の OS ある

[*1] 仮想マシンも本論文における UNIX 機の範疇とする．

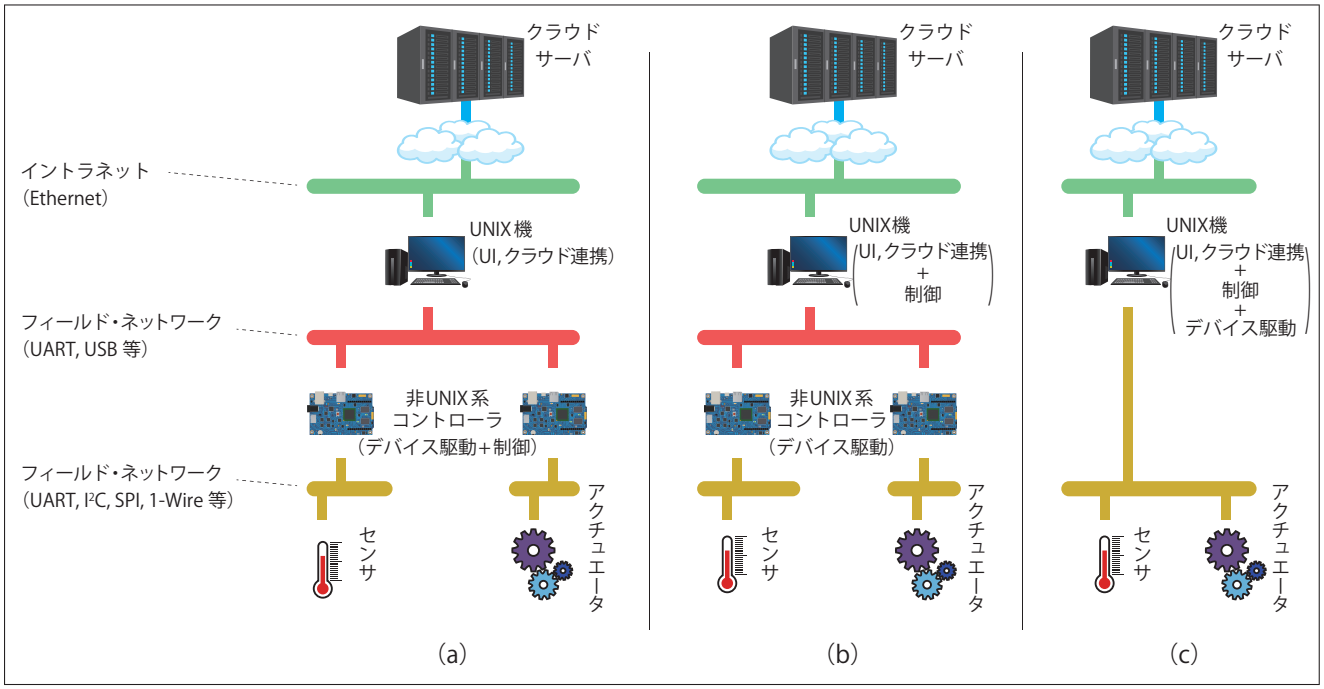


図 1. UNIX 機を利用した IoT 機器の構成例（文献 [1] の図を元に著者らが作成）

いはプログラミング言語で動作している．役割を単純化して分担を進めることにはシステム全体の見通しを良くする効果がある．

しかし，図 1 の (b) や (c) のような構成の方がより適切なケースも考えられる．(b) は制御の役割をコントローラから UNIX 機へ移したもの，(c) はさらにセンサやアクチュエータの駆動についても UNIX 機が担うことでコントローラを排したものである．これにより，以下の利点が得られる．

- UNIX 機に役割を委譲することによる，コントローラ側のソフトウェア・ハードウェアの簡素化や削減
- POSIX 仕様ベースでの実装の割合増加による，システム全体としての互換性・持続性の向上

特に後に示した利点は，著者らが「POSIX 中心主義」[2] と呼ぶものである．目的の機能・性能を得るために最適であるという理由のみでプラットフォーム（OS やライブラリ，プログラミング言語等）を選択すると，保守コストが増大したり持続性を損なうリスクが高い．反対に，持続性で高い実績のある POSIX に最大限準拠した開発を行うことで，長期にわたる持続性を得られる可能性が高いことを前述の論文で示した．

ところが，(b) や (c) のような構成にするには，少なくとも以下の問題がある．

1. プリエンプティブなスケジューラにより，各プロセスの動作タイミングの予測が困難
2. パイプラインのバッファリングにより，データの到来タイミングの予測が困難
3. タイミングを管理（記録や生成）するためのコマンドの不足

それぞれの詳細は次節で述べるが，上記 1 への対策としてはリアルタイム OS（RTOS）化された UNIX の規格や実装がある．例えば商用ライセンス製品では QNX[3] があり，POSIX のリアルタイム拡張である POSIX 1003.1b に準拠した API を持つ．一方オープンソースでは，RTLinux[4] や ART-Linux[5] が Linux カーネルに対するパッチとして提供されている．しかし，これらはそれぞれに動作可能なハードウェアが限定され，具体的なプログラミング方法が違うなどにより，非リアルタイムな通常の UNIX 機のような汎用性及び互換性を有してはいない．実際，RTLinux を導入しようとした場合，まず使用を予定しているハードウェアがそれをサポートしているかを確認し，さらに使用を予定している Linux カーネ

ルのバージョンのソースコード一式と，それに対応したRTLinux パッチを用意し，ビルドを成功させなければならない．また，リアルタイム性を確保するために，リアルタイムプロセスからは一般のシステムコールが利用できないなど，プログラミング上の特別な制約もある．加えて，組み込み機器はしばしば大規模システムの基盤としても活用されるため，製品寿命の長さは重要な性能の一つである．

なお，Linux 2.6.23 以降から導入された Completely Fair Scheduler（CFS) スケジューラ [6] では FIFO（First In, First Out）スケジュールポリシーを用いるなどでリアルタイム処理に対応することが可能である．しかし，これもアプリケーションの実行時にスケジュール・ポリシーを変更するなどの対応が別途必要である．

以上の背景に基づき，著者らは，広く普及している非リアルタイムな UNIX 機全般で通用する範囲，すなわちオプション扱いである POSIX 1003.1b を含まない POSIX（POSIX.1-2017 とする）[7] のみに準拠させながら，前述の問題解決を図った．

2. UNIX 機のタイミング管理の問題点

前節でも指摘したように，UNIX 機で実用的なタイミング管理を実現するには，次の三つの問題への対策が必要である．

2.1. スケジューラによる処理タイミングのぶれ

非リアルタイムかつプリエンプティブな通常の UNIX 機には，動作中の各プロセスにリソースを公平に割り当てるためのスケジューラが実装されており，処理タイミングにぶれが生じる．(POSIX の範囲ではないが）ナノ秒単位のタイムスタンプと stat コマンドを持つ UNIX 機であれば，次のコマンドによりその大きさを確認できる．

```
$ touch a; sleep 1; touch b; sleep 1; tou
ch c; sleep 1; ... ; touch k
$ stat [a-k]
```

このコマンドにより 11 個のタイムスタンプが得られ，順番的に隣り合うタイムスタンプ間の経過時間が 10 か所で求められる．それら経過時間のばらつきから，UNIX 機の処理タイミングのぶれの大きさがわかる．

この試験を，組み込み用途の UNIX 機としてもよく用いられている Raspberry Pi 3B+に Raspberry Pi OS (buster) を標準インストールした直後の環境で実行したところ，上記のようにして求めた経過時間同士の差の大半はナノ秒オーダであったものの，最も大きいものと小さいものでは最大 13 ミリ秒の差（処理タイミングのぶれ）があった．従って，少なくともこの環境では，マイクロ秒単位やそれ以下の精度を要求するときはプロセスのスケジューリングを意識し，優先度を変更する，あるいはスケジューリングポリシーを変更するなどの手続きが別途必要となる．

2.2. バッファリングよるデータ転送の遅延

標準入出力のバッファリングによっても発生する遅延がある．その様子は次のワンライナで観測できる．

○バッファリング遅延の再現

```
$ while sleep 1; do date; done | tr 1 1 |
 cat
```

○ stdbuf コマンドによる解決方法

```
$ while sleep 1; do date; done | stdbuf -
oL tr 1 1 | cat
```

このワンライナは 3 段階のコマンドから構成される．1 段目では，date コマンドによる現在時刻を 1 秒の休みを置きながら出力している．2 段目ではその文字列を tr コマンドが受け取るものの，結果的に何も手を加えず内容をそのまま次に送る．3 段目では，cat コマンドがやはり何の加工もせず次へ送るので，刻々と変わる現在時刻を画面に表示することになる．この時，最初の sleep コマンドの作用によって約 1 秒ごとに現在時刻が一行ずつ画面に表示されるわけではなく，実際には数十秒以上経ってから，それまで出力されなかった分の時刻が一斉に表示される．

この数十秒以上の遅延は，タスクスケジューラの影響ではなく，明らかにバッファリングによるものである．制御にとってこれは過大な遅延であるうえに，可変長データを扱う場合，どのタイミングでバッファリングされたデータがフラッシュされるのかの予測も困難である．

バッファリングモードを変更するコマンドとしては，GNU プロジェクトが stdbuf コマンドを公開している．これにより，Linux であれば 2 番目に例示したワンライナでこの問題が解決する．しかし stdbuf は POSIX に掲

載されているものではないので，すべての UNIX 機で使える保証がないうえに，効果が及ばない場合もある．実際，macOS にあるオリジナルの `tr` コマンドには作用しない．macOS 上で必要な場合は，GNU core utilities 版の `gtr` というコマンドをインストールして使わなければならない．また，コマンド自身がバッファリングモードを管理しているもの（`perl` や `python` コマンド等）にも通用しない．これらは，各コマンドのスクリプトやオプションで切り替えなければならない．このように，UNIX 機というだけではバッファリングに関して処理タイミングの互換性を担保できない．

2.3. タイミング管理・生成用のコマンド不足

データ転送における情報の価値は，流れるデータの値のみならず，その送受信タイミングにもある．キーボードやマウスを始めとしたヒューマン・インタフェース・デバイスとの通信はその代表例である．

しかし，POSIX の範囲で提供されているコマンドで，時間調整に使えるコマンドはほとんど無い．時刻を知るコマンドとしては `date`，待機時間を作るコマンドとしては `sleep` があるが，どちらも POSIX 仕様としては秒単位までででの利用しか保証されていない．そのために，UNIX 機によって対応・非対応が分かれるうえ，例え対応していたとしてもコマンド書式が統一されていないので，例えば Linux なら Linux 用に限定したシェルスクリプトを書かざるを得ない．

また仮に，POSIX 仕様で定められた秒未満対応の `date`，`sleep` コマンドがあったとしても，シェルスクリプトの動作速度を考慮すると十分とは言えない．

3. UNIX 機のタイミング管理の改善

前節で述べた問題に対し以下の対応を行った．

3.1. データの行列指向テキスト形式化とタイムスタンプの付加

2.1 項では，処理タイミングのぶれが 10 ミリ秒の規模で生じる例を示したが，これ自体は非リアルタイムかつプリエンプティブな UNIX 機を使っている限り避けられない問題である．しかし，制御において，出力の計算にはサンプリング間隔 Δt の管理が重要であるため，仮に各周回の実行時間 Δt を精密に測定できれば，制御においては次の周回の出力値計算のために利用でき，UNIX 機上の処理タイミングのぶれを補償できる．そのためには，UNIX 機の中を通過する各周回の計測値・出力値データにタイムスタンプを付けられればよい．

3.1.1 ものグラミング 2

データにタイムスタンプを付加するために，今回は大野らが提唱する「ものグラミング 2」[8] というアプローチを採用した．

ものグラミング 2 は IoT デバイス開発手法の一つであり，その特徴はデバイスに対して次の三つの要求をすることにある．

- やり取りするデータは，バイナリ形式ではなく，原則として行列指向のテキスト形式とすること．
- UNIX 機からキャラクタデバイス（標準入出力に接続可能なデバイス）として扱えるインターフェース（例えばシリアルポート）でのデータ授受をすること．
- 複雑な計算は UNIX 機に任せ，センサ読み取りまたはアクチュエータ操作の役割に徹し，汎用性・持続性の高いシンプルなハードウェアの選択（例えば Arduino）に努めること．

デバイスがこれらのルールを守ることで，UNIX の持つ高い汎用性・持続性をシステム全体としても継続させやすくなる．また，デバイスに対するコストも抑えられ，多くの人が慣れ親しんでいる UNIX によるプログラミングの知識が活かせる領域が広がるために学習コストも抑えられる．

これがものグラミング 2 の概要である．

3.1.2 行頭へのタイムスタンプ列の追加

UNIX におけるテキスト形式は，基本的にデータ長は固定されておらず，次の改行コードまでが一つのレコードとして扱われる．その性質により基本的な UNIX コマンド (`awk` 等) を使って列の挿入・削除が簡単に行える利点を有している．従って，行列指向のテキストデータであれば，タイムスタンプの列を各行頭に挿入できるという柔軟性をもつ．

例えば，ものグラミング 2 の要求に従い，シリアルポートにテキスト形式で分解能 10 ビットの 3 軸加速度センサの読み値を 0.1 秒毎に送るデバイスであれば，図 2 のように行頭に UNIX 機への到達時刻を付加する．

```
20210310090000.000000 92 411 823
20210310090000.100000 98 412 831
20210310090000.200000 101 411 834
                    :
                    :
```

図 2. タイムスタンプ付加データの例

この例では，カレンダ日時形式（YYYYMMDDhhmmss）にマイクロ秒精度の小数を付けたフォーマットとしているが，年月日の並べ方の習慣やタイムゾーンで齟齬が生じる恐れがある場合には，UNIX 時間で表現する方がよい．いずれにせよ時刻が記録されていれば，各値がいつ到来したものかという情報が残り，次の行との時間差（Δt）も求められる．

UNIX 側でタイムスタンプを付加する場合には，既に述べたように，タスクスケジューラによる処理タイミングのぶれによる精度の限界はあるが，デバイス側にリアルタイムクロック（RTC）を搭載しなくてよいという利点がある．RTC の搭載によって利便性が向上するように見えるが，実際には搭載した RTC の精度を把握する必要があるだけでなく，RTC が保持する時刻を基準時に同期させる必要があるなど，タイムスタンプ問題をすべて解決するには至らない．

3.2. stdbuf より効果のあるバッファリング回避方法

2.2 項でバッファリングによる遅延の発生と，`stdbuf` コマンドでもなおこの問題を解消できない例を示した．

`stdbuf` コマンドは，一部の UNIX 実装で提供されている LD_PRELOAD 機構を用い，動的リンクによってバッファリングモード切替ルーチンを対象コマンドに送り込み，対象コマンドの起動時に実行させている [9]．この機構は POSIX で保証されたものではなく，そもそも静的リンクでビルドされたコマンドには作用せず，stdbuf の効かないコマンドが存在する一因になっている．

著者らは，POSIX 文書に記載されているライブラリやシステムコールで，より簡単にこの問題を解決した．

3.2.1 標準出力への疑似端末（PTY）の接続

文献 [10] の 5.4 Buffering によれば，ANSI C の要請により，標準入出力ライブラリ `stdio.h` が提供するバッファリングモードは，標準エラー出力を除く標準入出力が対話的な装置に対する場合に限り，完全バッファリングではなく行バッファリングをデフォルトとしている．

そこで，`stdbuf` コマンド同様に，ラッパーコマンドを作る．ここではそれを `ptw`（Pseudo Terminal Wrapper）コマンドと呼ぶ．`ptw` コマンドは，疑似端末（PTY）を作成し，ラッピング対象コマンドの標準出力と次のコマンドの標準入力の間に介在する．すると対象コマンドの `stdio.h` ライブラリは，対話的な端末に接続されていると思い込んで自発的に行バッファリングモードにすると期待できる（図 3）．3.1 項のようにしてデータは既に行単位にしてあるため，行単位のバッファリングであれば実質的にバッファリングの影響は無い．

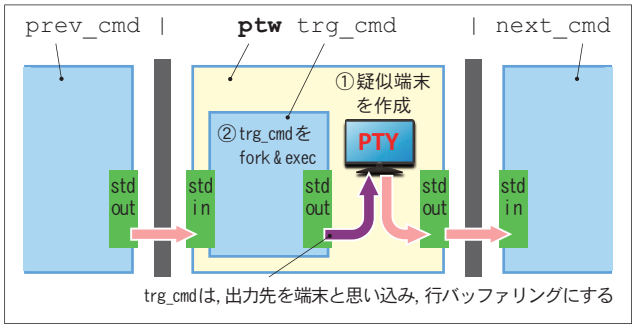


図 3. 疑似端末を利用した完全バッファリング回避

3.3. タイミング生成

2.3 項で述べたコマンドの貧弱さへの対策として以下を実施した．

3.3.1 タイムスタンプ付データの活用

これには 3.1 項で発案した行列指向テキストデータ化とタイムスタンプの付加が役立つ．タイムスタンプ付のデータを一行ずつ読み取り，そこに記されているタイムスタンプから前後の行との時間間隔に基づいた休みを挿入しつつ，タイムスタンプの列を除いた残りの文字列を標準出力に書き出すことでタイミングを再現できる．

タイミングを再現するのではなく生成したい場合には，前述のコマンドの前段階で，タイムスタンプを計算して

決めるプログラムを置けばよい．例えば awk コマンドを使い，出力値の変化が緩やかな区間では長い間隔で，激しい区間では短い間隔で出力時刻を定めることもできる．また，倍速再生などもできる．先程と同様に awk コマンドを使い，タイムスタンプの増分が 1/2 になるようにタイムスタンプを変更すればよい．

3.3.2 処理タイミングのぶれへの対策

UNIX 機の処理タイミングのぶれによってタイムスタンプの指示した時刻から実際の出力時刻がずれることが想定される．これに対しては，次の行の出力時刻はその都度 RTC を参照しながら決めることにより，局所的にはずれても，大域的には相殺できる．

ただし，遅れた場合にその遅れを次の周回で取り戻そうとして短時間しか休まなければ，瞬間的な速度が想定を上回り，その先に繋がれたデバイスがデータを取りこぼす恐れが考えられる．デバイス側が十分なバッファを搭載するなど，フロー制御が適切に動作している場合には問題ないが，そうでない場合には，一つの周回で取り戻してよい遅れの時間を制限したり，あるいは一切許さないなどの対策もしなければならない．

4. タイミング管理コマンドの実装

以上の解決案に基づき，著者らはタイミング管理機能を強化するための UNIX コマンドをいくつか作成し，公開した [11]．なお，本論文で冒頭から述べているように，UNIX の高い互換性・持続性は極めて重要であるため，作成したコマンドはすべて，C99（ISO/IEC 9899:1999）または POSIX 文書に掲載されている関数さえサポートされている環境であればビルドでき，かつ使用できることを保証した．

4.1. linets コマンド

標準入力または引数で指定したファイルからテキスト形式のデータを受け取ると，各行の到来時刻を行頭に付加して標準出力へ書き出す．到来時刻は，カレンダ日時（タイムゾーンは環境変数 TZ で変更可），UNIX 時間，コマンド起動時からの秒数，1 行目到来時刻からの秒数の 4 種類が選択でき，出力精度についても，秒，ミリ秒，マイクロ秒，ナノ秒から選択できる（ただし信頼度は UNIX 機の動作速度や処理タイミングのぶれによる制限を受ける）．

また，制御の分野では時間差（Δt）が利用されることも考慮し，タイムスタンプ列の後（2 列目）に前行からの経過秒数を追加するオプションも用意した．

なお，各行の到来日時のタイムスタンプを付記するコマンドとして moreutils[12] の ts コマンドがあるが，Perl で書かれており，かつ一部オプションを利用するには追加 Perl モジュールを要するなどにより POSIX 中心主義に反するため，独自実装した．

4.2. ptw コマンド

これは，2.2 項で述べた仕組みを実現するためのラッパーコマンドである．

書式は単純で，対象となるコマンドの手前に半角空白を一つ置いて “ptw” の記述を追加するだけである．

ライブラリを静的リンクしたコマンドなど，stdbuf コマンドでは効果の及ばなかったものの多くに効果があるが，次のような一部のコマンドには効果が無い．

- 意図的に完全バッファリングモードを指定している場合
- mawk コマンドなど，ANSI C や stdio.h の要請に従わず，出力先が端末でも行バッファリングにしない場合（mawk では，-W interactive オプションが必要）
- シェルスクリプトで書かれており，内部でパイプによって別のコマンドをカスケーディングしている場合（内部で使用しているコマンドが “*cmd*” である時，シェルスクリプトの冒頭で “alias='ptw *cmd*'” などとエイリアスを定義することで対応可能）

4.3. tscat コマンド

タイムスタンプを意識した cat コマンドという由来で，linets コマンドの反対の動作をするものであり，linets の生成するタイムスタンプ形式を理解する．すなわち，1 列目のタイムスタンプが示す時刻が到来した時に 2 列以降行末までを標準出力に書き出し，次の行を読み込む．ただしこれではタイムスタンプが過去のものであった場合には意味が無いので，コマンド起動時の時刻または 1

行 1 列目に記されている時刻からの相対時間で出力時刻を決めるオプションも用意した．

なお，linets コマンド同様，タイムスタンプはナノ秒単位まで対応しているが，実際の信頼度は UNIX 機の動作速度や処理タイミングのぶれによる制約を受ける．ただし，1 行毎に clock_gettime() 関数が返す現在時刻を参照しながら次の出力時刻を決めているため，大域的にはそれらの影響が打ち消される．

4.3.1 linets, ptw, tscat の応用例

著者らの web サイトに，以上の三つのコマンドの使用例を掲載した [13]．今回作成したコマンドの他，awk や cat，tee など，基本的な UNIX コマンドをパイプ（|）でカスケーディングすることで，ロボットの手動操作を記録・再生，0.5 倍速再生するという応用もできる．

4.4. その他のコマンド

タイミング管理のため，他にコマンドを二つ作成した．

4.4.1 valve コマンド

「弁」が由来のこのコマンドは，標準入力またはファイルから到来したデータを 1 バイト毎または 1 行毎に，それぞれが一定間隔になるように標準出力に送り出すコマンドである．

例えば，ファイル sensor1 の内容を，300 bps で出力したければ “valve 300bps sensor1” と書け，行単位に各行を 100 ms 毎に出力したければ “valve -l 100ms sensor1” と書ける．

このコマンドを作成した理由の一つは，3.3.2 の後半で述べたオーバーフロー対策である．タイムスタンプに従う tscat コマンドに実装すると複雑になるため，別コマンドとして作成し，二つのモードを用意した．瞬間的にも速度超過を避けるため，例え遅れたとしても次の周回で後れを取り戻すことを一切しないモードと，一定の範囲内であれば取り戻しを許容するモードである．

なお，流速を調整できるものとして pv コマンド [14] が知られているが，そちらはデータ送出周期が約 90ms（バージョン 1.6.6 の場合）で，その周回毎に設定流速に沿った量のデータをまとめて送るのに対し，valve コマンドは 1 バイトまたは 1 行送るごとに POSIX の nanosleep() 関数を呼び出すことで，より高品質にタイミング再現ができるように設計した．

4.4.2 秒未満に対応した sleep コマンド

これは，一部の UNIX 実装で提供されている sleep コマンドと同様に，引数に指定できる待機時間を小数を含む値（ナノ秒単位まで）にも対応させたものである．

車輪の再発明をした理由は，一部の UNIX 実装にしか存在しないそのような sleep コマンドを POSIX 環境で保証するためである．

なお，本項に記した二つのコマンドも他のコマンドと同様，実際の信頼度は UNIX 機の動作速度や処理タイミングのぶれによる制約を受ける．

5. 使用事例

5.1. 非 RTOS 化 Linux による倒立振子制御

linets と ptw の二つのコマンドにより，同軸二輪車型倒立振子（名称: クララ[*2]）を立たせることに成功した．

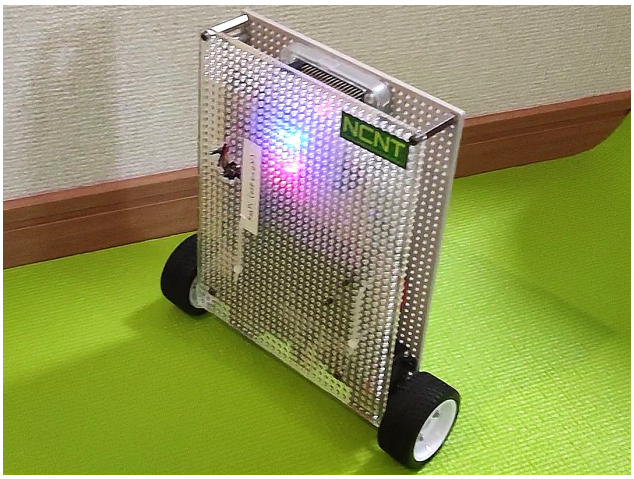


図 4. 倒立振子「クララ」

クララは，3 軸の加速度センサとジャイロセンサ，ロータリーエンコーダを装備し，逐一それらを計測し，カルマンフィルタによる真値推定と，制御による車輪モータへの出力値を計算しながら倒立する．柳戸らの報告 [15] では，制御アルゴリズムとして最終的に線形 2 次レギュレータ（Linear Quadratic Regulator:LQR）を採用した

[*2] Klara．正式名称は “Kinetical Leg Astability Runs Away.”

が，選定段階では古典制御論の重要な成果物である PID 制御でも実装したので，本稿ではこちらを解説する．

この実験の重要な点は，前述の真値推定と制御計算を，カーネルに一切カスタマイズや調整を加えていない通常の Linux（Raspberry Pi Zero + Raspbian[*3]）で実現した点である．Linux カーネル 2.6.23 以降ではプロセスのスケジューラは CFS が標準となっているが，その Linux カーネルの環境において別途リアルタイム処理などの設定などの適切な指定を行わないと，2.1 項で示したような処理タイミングのぶれが生じる可能性がある．そこで `linets` コマンドによって，センサから送られてきた値に Raspberry Pi 側のタイムスタンプと前回測定時刻との差 Δt を付加し，カルマンフィルタと PID 制御のプログラムで利用することで処理タイミングのぶれを考慮した制御を実現した．

図 5 に，倒立動作中にクララが記録した，時刻対角度のグラフを示す．

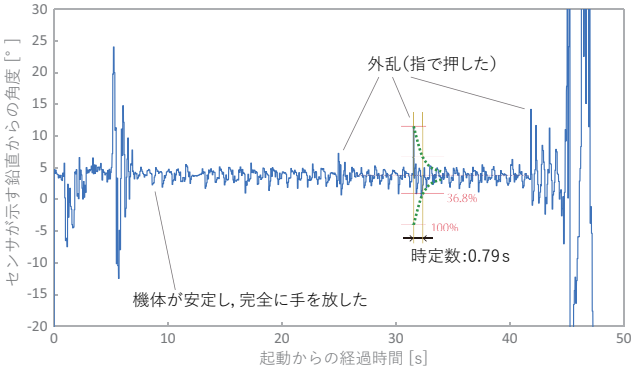


図 5. 倒立振子「クララ」の時定数推定

グラフによれば，起動から 9 s 頃に機体が安定したために手を放され，加速度センサに基づく鉛直からの傾き角が約 3.8 ℃で安定し，その後　25 s，31 s，42 s 頃に機体を指で軽く押されて外乱を加えられたという，一連の動作が観測されている．そして，外乱を加えられるたびに過渡現象が発生しているが，2 番目の 31 s 付近のものから時定数を推定したところ，0.79 s であった．このことから，POSIX のコマンドでは用意されていない 1 秒未満のより精密なタイミング管理を目指して作成したコマンドが，実際に 1 秒未満の時定数を持つ系の制御に応用できることが示された．

なお，当初作成した PID 制御用のコマンドは，時刻参照をコマンド自身が内部で実施していた．しかし，正確な時刻を得るためには値の入力・計算・出力を非同期にするなどでプログラムが複雑になることや，PID 制御のシミュレーションが実時間でしかできないなどといった問題があり，外部の `linets` コマンドに任せる形で作成し直した．これは，単機能の組み合わせで課題を解決するのがよいと説く UNIX 哲学の効果の顕れと言える．

5.2. gnuplot を用いた心電図情報の動的表示

著者の一人（大野）は，医療用「ECG マルチパラメーターモニターモジュール PM6750」[16]（以下，本項では ECG センサ）から受け取った生体情報を複数の関係者とクラウドを介して共有する試みを続けている．

この ECG センサは，PC とシリアルインタフェースを介して接続でき，PC 上の専用ソフトウェアは ECG センサから心臓の電位情報（これを図示したものが心電図，ECG（Electrocardiogram））, 心拍数，血圧などをリアルタイムで取得し直ちにグラフとして描画できる．このソフトウェアは掃引（スイープ）時間が 5 秒以上のオシロスコープを模しており，表示されるグラフの波形はスイープの進捗に合わせて変化する．

この専用ソフトウェアが表示するグラフの見栄えは優れているが表示中のデータをクラウドに転送する機能はない．しかし，ECG センサから PC に送られるデータは無手順のバイナリデータであり，さらにバイナリのデータフォーマットは販売元から入手可能で，既存のコマンドで形成できるほど単純な形式であった．そこで (1)ECG センサから送られてくるバイナリデータを (2)`cu` コマンド[*4]で受け取り，(3)`awk` での処理に配慮して各バイトを 10 進数表記のアスキー文字列に加工した上で，(4) これを集めてアスキー形式のデータフレームにし，(5) さらに 1 フレーム/行の形にした上で，(6) 行頭にタイムスタンプをつけて，(7) クラウド上の MQTT ブローカに publish（以下「出版」）した．このデータを MQTT ブローカから subscribe（以下「購読」）するとデータは以下の形式になる．

1 秒間に購読できるデータ数は生体情報の種類によって異なり，心電図情報の場合は毎秒 25 回前後，心拍数などのデータの場合は毎秒 1 回前後である．著者らは，掃引時間を 5 秒ないし 10 秒に設定したオシロスコープ

[*3] 現在の Raspberry Pi OS.

[*4] もともとはシリアルポートを介して他のシステムと直接あるいはモデム経由で接続するためのコマンドだが，ここではシェルの標準入出力とシリアルポートを相互につなぐ目的で利用している．

```
タイムスタンプ, 生体データの種類, データの数, データ 1, データ 2,... データ N, チェックサム
```

図 6. タイムスタンプを追加したデータフレーム

を gnuplot[17] を用いて再現し，購読した心電図データの動的表示を試みた．グラフを表示する方法はいくつもあるが，今回は POSIX 環境で確実に利用でき，著者らが長年使い慣れているという理由で gnuplot を選択した．

gnuplot は，与えられたデータを少ない手順でグラフにできるが，オシロスコープのように素早く変化するデータを動的に表示する機能はない．しかし，標準入力からのデータ読み込みとグラフ描画を 0.1 秒程度ごとに繰り返すようにすればデータを動的に表示しているように見える．そこで，MQTT からデータを購読してリングバッファに溜め込む部分と，このスクリプトからデータを読み出してはタイムスタンプを参照してプロットを行う部分からなる図 7 のようなスクリプトを用意した．

一つ目のスクリプトでは MQTT から購読したデータを一つのファイルに蓄積し，この例では 5 秒ごとに一つ目のファイルに蓄積されたデータを二つ目のファイルに移している（実際には `mv` している）．二つ目のスクリプトでは最も新しいデータを基準に過去 5 秒間のデータを `dg3` コマンドに渡している．`dg3` コマンドは内部で gnuplot を `popen()` を使って呼び出し，標準入力から読み込んだひとまとまりのデータを使ってグラフを描く．

ここでは ECG センサから送らたデータを直ちにグラフに描いたが，タイムスタンプが付与されているので，データが到着したタイミングを正確に再現した描画がいつでもできる．

6. おわりに

UNIX はパイプラインの仕組みを持っており，それはコマンドやシェルスクリプトから簡単に利用できる．一方，自動制御や信号処理の分野では，しばしば各種フィルタをカスケーディングさせて実装するため，両者の相性は本質的に良いと考える．しかし，POSIX で定められいる必要最低限の UNIX コマンドはもとより，GNU core utilities など広く普及している UNIX コマンドまで見渡しても，シェルスクリプトベースで IoT 機器制御を実現するためのコマンドは充実してるとは言えず，先行事例も少ない．その主な原因の一つはタイミング管理

```
# === (1) データを蓄積する側 ===
mosquitto_sub -h HOSTNAME -t ECG/demo     |
linets -3e                                |
ptw awk '{print int($1/5)%2,$0;}'         |
awk '                                     #
 BEGIN{n=0}                               #
 n!=$1{system("mv -f 1 0");n=$1}          #
 {                                        #
  print substr($0,index($0," ")+1) >> "1" #
  close("1")                              #
 }'

# === (2) データを取り出す側 ===
yes | valve -l 200ms | while read dummy; do
  latest_ts=$(cat 0 1                |
              sed -n '${s/ .*//;p;}')
  cat 0 1                            |
  awk -v t=$latest_ts '$1>=(t-5)' |
  dg3
done
```

図 7. gnuplot で直近 5 秒間のデータを繰り返し描くシェルスクリプト

のためのコマンドが不十分であるからだと考え，いくつかのコマンドを作成し，シェルスクリプトベースのプログラミングにより倒立振子を立たせることに成功するなど，いくつかの事例において作成したコマンドの有用性が示された．

2021 年現在，IoT 機器の開発では Python などの高級言語が多用されているが，それらが製品・開発基盤の寿命の長さなど，業界の要求に応えられるのか疑問である．もし廃れてしまえば資産は失われて開発の大部分がやり直しになり，あるいは廃れなくても後方互換性の無い仕様変更がなされれば更新を迫られ，システム障害のリスクが増してしまう．UNIX 機を用いたシェルスクリプトベースの開発はそういった問題を避けるための良策である．また，UNIX 哲学にも即している．macOS や WSL の登場などが示唆しているように，UNIX 的アプローチや実装は今，多くの開発現場で支持されている．

著者らは，IoT 機器の開発においても UNIX 哲学的アプローチを積極的に採用すべきであると考える．その際，基本的な UNIX コマンドで不十分であると感じるので

あれば，高級言語プラットフォーム導入の前に，POSIX に即したコマンドの開発を検討すべきである．

謝辞

この論文は，研究活動を支えてくださった多くの方々のご厚意・ご協力によって完成に至ったものである．まずは，同じ研究チーム「NCNT プロジェクト」の北嶋完基氏と矢嶋遼氏，そしてチームの良きアドバイザーとしてご助言くださった田口淳一氏に感謝を申し上げる．さらに，研究内容をご理解くださって色々な面でご協力くださった USP 研究所の皆様，並びに金沢大学の皆様にも御礼を申し上げる．最後に，私達が作成したコマンドに GitHub 上で意見をくれた秘密結社シェルショッカー構成員のミツイ氏，そして POSIX 中心主義アプローチを支持してくれるすべての皆様に感謝する．

参考文献

[1] 株式会社マクニカ, “産業用ネットワークの種類･シェアの最新情報【RS485 はもう古い？】”, https://emb.macnica.co.jp/articles/4277/, 2021-03-15 閲覧.

[2] 松浦智之, 大野浩之, 當仲寛哲, “ソフトウェアの高い互換性と長い持続性を目指す POSIX 中心主義プログラミング”, **デジタルプラクティス**, vol. 8, no. 4, pp. 352–360, 2017.

[3] BlackBerry Limited, “BlackBerry QNX”, https://qnx.com/, 2021-03-14 閲覧.

[4] Yodaiken, V. “The RTLinux Manifesto”, *Proceedings of the 5th Linux Expo*, 1999.

[5] 石綿陽一, 松井俊浩, 国吉康夫, “高度な実時間処理機能を持つ Linux の開発”, **日本ロボット学会学術講演会予稿集**, vol. 16, no. 1, pp. 355–356, 1998.

[6] openSUSE ドキュメンテーション, “openSUSE 13.1: 第 14 章 タスクスケジューラのチューニング”, https://manual.geeko.jp/ja/cha.tuning.taskscheduler.html, 2021-03-15 閲覧.

[7] The IEEE and The Open Group, “The Open Group Base Specifications Issue 7, 2018 Edition, IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)”, https://pubs.opengroup.org/onlinepubs/9699919799/, 2021-03-13 閲覧.

[8] 大野浩之, 松浦智之, 森祥寛, “ものグラミング 2 - 諸機能の選択と集中を徹底した POSIX 中心主義に基づく IoT 開発方式の提案”, **研究報告インターネットと運用技術（*IOT*）**, vol. 2019-IOT-46, no. 16, pp. 1–8, 2019.

[9] Free Software Foundation, coreutilitles/stdbuf.c, https://github.com/coreutils/coreutils/blob/master/src/stdbuf.c, 2021-03-13 閲覧.

[10] Richard, W. S., and S. A. Rago “Advanced Programming in the UNIX Environment Third Edition”, *Addison-Wesley Professional*, 2013.

[11] USP NCNT プロジェクト, “Timing Management Commands in POSIX”, https://github.com/NCNT/TimingCmds_in_POSIX, 2021-03-13 閲覧.

[12] Joey Hess, “moreutils”, https://joeyh.name/code/moreutils/, 2021-05-17 閲覧.

[13] USP NCNT プロジェクト, “標準入出力上のデータの流れるタイミングを記録・再生する方法”, https://growi.ncnt.work/NCNT/info/linets_and_tscat.ja, 2021-03-13 閲覧.

[14] Andrew Wood, “Pipe Viewer”, http://www.ivarch.com/programs/pv.shtml, 2021-05-17 閲覧.

[15] 柳戸新一, 松浦智之, 鈴木裕信, 大野浩之, “シェルスクリプトを用いた UNIX 哲学に基づくリアルタイム制御”, **ソフトウェア・シンポジウム *2021*（投稿中）**, 2021.

[16] Shanghai Berry Electronic Tech Co.,Ltd, “ECG マルチパラメーターモニターモジュール PM6750”, https://www.medicalexpo.com/ja/prod/shanghai-berry-electronic-tech-co-ltd/product-122578-866840.html, 2021-03-15 閲覧.

[17] Williams, Thomas, et al. “gnuplot homepage”, http://www.gnuplot.info/, 2021-03-15 閲覧.

構成管理ツールにおける命令的操作が及ぼすソースコードレビューへの影響調査

頭川 剛幸
京都工芸繊維大学 工芸科学部
t-zukawa@se.is.kit.ac.jp

近藤 将成
九州大学 システム情報科学研究院
kondo@ait.kyushu-u.ac.jp

西浦 生成
京都工芸繊維大学 大学院工芸科学研究科
k-nishiura@se.is.kit.ac.jp

崔 恩瀞
京都工芸繊維大学
情報工学・人間科学系
echoi@kit.ac.jp

水野 修
京都工芸繊維大学
情報工学・人間科学系
o-mizuno@kit.ac.jp

要旨

近年複雑化する IT インフラに対応するため，インフラをコードで定義する構成管理ツールが用いられている．表現方法として計算機の状態を定義する宣言型と，スクリプトを記述する命令型がある．命令型は，環境の再現性や堅牢性の脅威になるとされているが，ソフトウェア開発にどのような影響を与えているのか明らかになっていない．本研究では，構成管理ツールの実行の最小単位であるモジュールから，命令的操作を行うモジュール（以下命令的モジュール）によって，コードレビューの時間が増大すると仮説を立てた．命令的モジュールを含むプルリクエスト（PR）について，PR が評価されるまでの時間，レビュースレッドの所要時間，変更行数およびファイル数を調査した．その結果，命令的モジュールの有無によって，マージされた PR およびレビュースレッドの所要時間に差は見られなかった．一方，変更行数およびファイル数に増加が見られた．これらの結果から，命令的モジュールを使用することで変更規模を増大させ，レビュー時の確認項目が増大する可能性があるが，レビュー時間は変化しなかったため，コードレビューにおいて命令型の使用は問題とならない可能性があることがわかった．

1. 緒言

近年のソフトウェアに対する需要の高まりから，企業は迅速かつ高品質に，信頼性のある製品を提供することが求められている．需要に対応するため，開発プロセスと運用を一つのサイクルにした DevOps モデルが普及した．DevOps の要素の 1 つとして Infrastructure as Code（IaC） というプラクティスが存在する [1–3]．IaC は，運用基盤の構成をコードによって記述し．開発プロセスのプラクティスを運用にも利用可能にする．これにより，プロビジョニング，構成，デプロイを自動化することができ，運用，開発規模の増大に対応することができる．

また，IaC が繰り返し実行可能かつ堅牢なシステムであるための基礎として冪等性の概念が存在する [4]．冪等性とは，異なる条件下において複数回の繰り返しを行うことで全て同一な状態に収束する概念のことであり，代数学の基礎として研究されている [5, 6]．例として任意の整数のゼロ乗算や，ある整数の絶対値を求める操作などが該当する．

IaC で冪等性を担保する方法として，ツールが用意する冪等な操作を行うモジュール（以下冪等モジュール）を用いる場合と，ユーザ自らが冪等性の担保を行う場合の 2 種類が存在する．冪等性を持たないモジュール（以下非冪等モジュール）を用いる場合，ユーザがその実行に冪等性をもたせる必要がある．実際，我々研究グループの先行研究 [7] から Ansible [8] における非冪等モジュールを使用したタスクの内約 51.32%がその実行について

条件を持つため，ユーザはコマンドが及ぼすマシン状態への影響を把握し，冪等性を持たせようとしている可能性がある．

IaC にはシステムの表現方法として，命令型と宣言型の 2 種類が存在する．宣言型はマシンのあるべき状態を記述することによってシステムを構築し，命令型では制御対象のマシンの状態毎に必要な操作を明示的に記述する．命令型の使用は，環境再現性や堅牢性の脅威となると考える．環境再現性への影響として，命令型は非冪等な操作となる可能性が高く，開発者は利用するコマンドが与える影響を把握した上で，冪等な操作となるように，実行条件を与える必要がある．また，外部スクリプトの実行が可能であり，その変更による環境への影響を把握する必要がある．堅牢性への影響として，複雑な命令を利用した場合に，可読性への影響や，操作に対する変更，宣言型への変換などが容易で無い可能性がある．これらの問題点から，命令型のレビュー時に，宣言型に比べ実行条件や結果の調査が必要になり，利用するスクリプト等についての保守や，コマンドに対する理解が求められる．

そこで，本研究では命令型の使用によってレビュー時に，変更内容の妥当性の確認に掛かる時間や，変更箇所が増大すると仮説を立てて調査を行った．今回は先行研究 [7] と同様に Ansible に着目し，データセットとしてユーザが自由にアップロード可能なサイトである Ansible Galaxy[1]からリポジトリのリストを抽出し，GitHub で公開されているリポジトリからプルリクエスト（以下 PR）に関するデータを抽出する．それらを用いて，命令的モジュールを使用した場合 PR 中のレビュー時間や変更規模に変化が見られるかを明らかにした．ユーザが自由にスクリプトを実行可能である `command`・`shell`・`raw`・`script` の 4 種類の命令的モジュールを分析対象とし，以下の研究設問（以下 RQ）について調査した．

RQ1： PR および PR 中のレビュースレッドの所要時間に差が見られるか

命令的モジュールの使用によって制御構造や実行結果の予測が複雑化し，レビューに必要な時間が増加する仮説を立てた．PR が開始してからマージもしくはクローズされるまでの時間，PR に含まれるレビュースレッドが開始してから最終のコメントが行われるまでの時間のパーセンタイルを求めた．結果として，マージされた PR において命令的モジュールを使用する場合，最大値を除き各パーセンタイルの値が増加する傾向にあった．またレビュースレッドの場合では，マージ，クローズのどちらも大きな差が見られず，一貫した結果が得られなかった．

RQ2： PR 中の変更規模に差が見られるか

命令的モジュールの使用により変更箇所が複雑化する場合，変更規模が大きくなると仮説を立てた．命令的モジュールを含むマージされた PR では変更行数，変更ファイル数それぞれに増加が見られ，命令的モジュールの使用がこれらの値に影響を与えている可能性がある．

RQ1 および RQ2 の結果から，命令的モジュールの使用はレビューへの負荷になりにくいが，開発者が変更を行う際の負担が増加する可能性がある．

以降の本稿の構成を紹介する．第 2 章では，簡単な例を示し本研究の必要性を述べる．第 3 章では，研究背景である IaC および PR について説明する．第 4 章では，調査の目的と方法について説明する．第 5 章では，使用したデータセットについて説明する．第 6 章では，それぞれの研究設問に対する動機，調査手順および結果をまとめる．第 7 章では，結果に対し議論を行う．第 8 章では，本研究の結論を述べる．

2. 本研究の着想に至った動機の例

命令的モジュールの使用がユーザへ与える負担となる例をソースコード 1 および 2 に示す．例 1 は Ansible の公式ドキュメントに掲載されている例 [9] であり，例 2 は筆者が例 1 を命令的モジュールを使用するように変更したものである．この例はどちらも AWS 上で同様のインスタンスを作成するが，前者は Ansible によって用意された宣言的モジュールを使用し，後者はシェルスクリプトの実行によってインスタンスを作成するものである．前者の場合，システムの堅牢性を保つための要素である冪等性はモジュールが担保しているため，ユーザは必要なパラメータを指定すればよい．具体的には，例 1 の `community.aws.rds` より下（4 行目以降）に示されるパラメーターを定義することで，宣言的にシステムの状態を記述している．後者は，実行する CLI のオプションを調査した上で，それが冪等性を保つように実行制御を行う必要がある．具体的には，`shell` から始まる `aws` コマンドのオプションを指定し，命令的に実行の制御を行っている．また，冪等性を持たせるため，一度実行を

[1] `https://galaxy.ansible.com`

ソースコード 1. 宣言的モジュールを用いたデータベース作成例 [9]

```
1  ---
2  - name: Basic mysql provisioning example
3    community.aws.rds:
4      command: create
5      instance_name: new-database
6      db_engine: MySQL
7      size: 10
8      instance_type: db.m1.small
9      user_name: mysql_admin
10     password: 1nsecure
```

行った場合，変数（例では is_instance_exist）を用意し，実行結果を Ansible に記録させる．2 度目の実行が行われた場合，failed_when によって条件を追加し，結果が記録されている場合には実行させないことで冪等性を担保している．また，この場合実行結果の状態について，宣言的モジュールを用いた場合，最終的な状態が記述されているため視覚的に理解しやすいと言える．本研究では，これらの差異がソースコードレビューにどのような影響を与えているのか明らかにする．

3. 背景

3.1. Infrastructure as Code

Infrastructure as Code（IaC）は，ソフトウェア開発のプラクティスに基づいたインフラストラクチャ自動化のアプローチである．これは，システムのプロビジョニングと変更，およびその構成の変更を行うための操作を，繰り返し可能で一貫性のあるものにすることに重きをおいている．つまり，システムの状態を全てコードで定義し，変更を加える場合は対象のマシンを直接操作するのではなく管理するコードに変更を加え，自動化を使用してテストを行い，その変更をシステムに適用する運用方法である [10].

ソースコード 2. 命令的モジュールを用いたデータベース作成例

```
1  ---
2  - name: Basic mysql provisioning example
       (imperative)
3    shell: "aws rds create-db-instance\
4          --allocated-storage 10\
5          --db-instance-class db.m1.small\
6          --db-instance-identifier new-
               database\
7          --engine mysql\
8          --master-username mysql_admin\
9          --master-user-password 1nsecure"
10   register: is_instance_exist
11   failed_when: is_instance_exist not in
         [0..1]
```

3.2. 構成管理ツール

IaC を達成するために用いられるツールであり，主に対象のコンピューティングリソースに対して環境構築を行うことを目的とする．ミドルウェアの設定やクラウドリソースの構築，ネットワークトポロジの設定等を行う．代表的なツールとして，Ansible，Chef，Puppet がある．

3.3. 構成管理ツールの冪等性

冪等性は，複数の状態から一意の状態へ収束する性質のことであり，IaC において冪等性はシステムの再現性，堅牢性に対し重要である．Chef や Ansible 等の構成管理ツールでは冪等な操作を行うモジュールを用意しており，ユーザはこれらを用いることでシステムの冪等性を担保している．また，構成管理ツールは冪等でないモジュールも用意しており，これらの使用は冪等性に対する脅威となる．Chef におけるシステムの冪等性を判定するモデルを作成した Hummer らの研究 [4] では冪等性に対する脅威として，(1) 命令的な Chef スクリプトを用いる，(2) スクリプトリソースを用いる，(3) 全体のシステム冪等性を持たない，(4) 外部のリソースに依存している点を挙げている．

3.4. Ansible

本研究の調査対象とする構成管理ツールである．Red Hat, Inc. によって開発されているオープンソースソフトウェア（OSS）である．特徴として，管理対象のマシンに特別なソフトを必要としないことが挙げられる．

3.4.1 モジュール

リモートマシンで実行されるコードの最小単位であり，アプリケーションのインストールやクラウド上のインスタンス作成など様々な種類を持つ．多くのモジュールは key=value 引数を指定することで実行内容を制御する．執筆時点での最新安定版である Ansible 2.10.5 において，4,573 個のモジュールが存在する．

3.4.2 宣言的モジュールと命令的モジュール

宣言的モジュールとは，引数によって状態を定義するモジュールであり，実際に実行される命令は各モジュールに定義されている．命令的モジュールとは，ユーザが命令を実際に記述し状態を制御する．宣言的モジュールと命令的モジュールについて Ansible は，各モジュールがどちらに属するものか明記していない．本研究では，先行研究 [7] と同様にシェルスクリプトを直接実行可能である `command`，`shell`，`script`，`raw` モジュールを命令的モジュールとした．

3.4.3 ロール

Ansible が規定するディレクトリ構造の名称であり，それに従って適切にファイルを配置することで一連の機能の再利用を容易にすることができる．また，Ansible Galaxy[2]ではロールの形式に従ってファイルが共有されている．

3.5. GitHub におけるプルリクエスト（PR）

PR とは，第三者がある変更を行った際にそれらの変更が妥当なものであるかリポジトリのコアメンバー等がチェックを行うものであり，GitHub 上の OSS 開発で一般的に用いられている．また，PR の機能として変更箇所に対してレビュースレッドを建てることができ，それらに対して詳細な議論を行うことができる．

3.5.1 PR への時間的調査

本研究と同様に，GitHub の PR に対して時間的な調査を行った研究が存在している．Yue らは，PR の評価が終了するまでに掛かる時間に影響を与える因子を調査している [11]．Yue らが調査した因子の例としては，リポジトリが生成されてからの期間の長さや参加者の数，PR に含まれるコミット数，PR で追加，削除された行数，PR 投稿者の被マージ率等を調査している．PR が終了するまでの時間へ影響を与える因子として，コミット数，及び追加された行数が増えるほど PR の評価に時間を要し，リポジトリのコアメンバー，フォロワーの多い投稿者，過去のマージ率が高い投稿者による PR はより短時間で評価を終えるとされている．

4. 調査概要

先行研究 [7] において命令的モジュールの使用には問題があるとされているが，実際には約半数のロールに命令型が使用されていると報告されている．本研究の目的は，命令型の使用が開発現場において実際にどのような影響を与えているか明らかにすることである．命令型の使用は冪等性のための条件分岐や，実行内容の把握の煩雑化が，レビューに掛かる時間が増大させると仮説を立て調査を行った．

調査対象である Ansible では，OSS コミュニティである Ansible Galaxy と呼ばれる Web サービスが存在し GitHub のリポジトリが公開されている．公開されているリポジトリからレビューに関する情報を得るため，GitHub の機能である PR のデータを収集した．

RQ の調査に用いるデータとして，PR を閉じるまでの所要時間，PR 中のレビュースレッドでの所要時間，変更された総行数，ファイル単体での変更行数および変更ファイル数を用いた．また，分類を行うために PR がマージまたはクローズかどうかの状態，変更が行われたファイルおよび変更箇所を収集した．これらの情報から，命令的モジュールの使用状況と PR の状態によって，レビュースレッドを 1 件以上含む PR に対し，命令的モジュールの有無，PR がマージもしくはクローズであるか分類を行い，それぞれのパーセンタイルを求め，値に

[2] https://galaxy.ansible.com

差が見られるかどうかによって影響の有無を調査した．マージまたはクローズされた PR という場合分けを行った理由として，マージされた PR はリポジトリに変更を与えるため，より活発な議論になる可能性が考えられる．その場合，クローズされた PR との違いを明らかにするため，これらの場合分けを行った．

5. 調査対象のデータセット

5.1. データセットの概要

本研究では先行研究 [7] と同様に Ansible に限定して調査を行った．また，先行研究で行われた，ファイル中の命令的モジュールの有無に対する分類結果を用いたため，使用したデータセットは先行研究と同様のものとした．Ansible Galaxy で公開されているロールの内，2019 年 10 月 22 日時点でダウンロード数が上位 10%であり，その時点から過去 1 年以内に更新されたものとしている．それらのロールから GitHub のリポジトリの URL を取得し，GitHub API [12] からマージまたはクローズされた PR に関するデータを収集した．

5.2. データセットの取得方法

GitHubAPI から得られるデータのうち以下の要素を取得した．また，それらを図 1 に示す．

1. PR の状態，すなわちマージまたはクローズ
2. PR の開始時間及びマージまたはクローズされた時間
3. 各 PR 中に含まれるレビュースレッド開始時間及び最終コメント時刻
4. PR 全体及びファイル単体での追加，削除された行数
5. PR 中に変更が行われたファイル群
6. レビュースレッドの対象となっている変更内容

対象とする Ansible ロールの全てのリポジトリに対してこれらのデータを収集した．対象とした Ansible ロールは先行研究 [7] で使用された 884 個のロールを選択しデータの収集を行った．データ収集が完了した 2021 年 2 月 1 日時点での 26,567 件の PR から，レビュースレッドを含む，マージまたはクローズされた 2,652 件の PR，

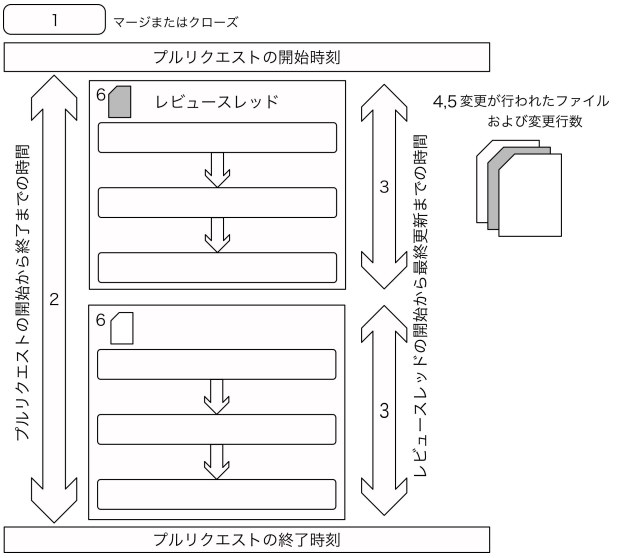


図 1. PR の持つ要素

および PR 中に行われた 7,944 件のレビュースレッドを取得した．

6. 研究設問の調査手順と結果

6.1. RQ1) PR および PR 中のレビュースレッドの所要時間に差が見られるか

6.1.1 動機

命令的モジュールの使用が，制御構造や実行結果の予測を複雑にする場合，レビュー時間が増大すると考えた．命令的モジュールの使用が実際に影響を与えているか確認する．

6.1.2 手順

PR 内に一件以上レビュースレッドを持つものに関して，PR の開始時刻からマージまたはクローズされるまでの時刻を，PR の解決までの所要時間（以下変更解決時間）として調べた．同様に，レビュースレッドに関して，開始時刻と最終更新時刻の差から所要時間（以下レビュー時間）を求めた．また，レビュースレッド中に議論が行われていないものに関しては，集計の対象として含めない．調査の際，以下の場合に分けてそれぞれの時間を調べた．

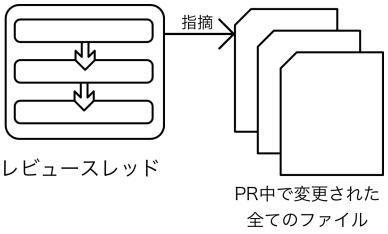


図 2. 命令的モジュールを含まない PR のレビュースレッド

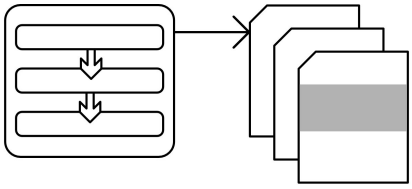


図 3. 命令的モジュールを含む PR のレビュースレッド

- 条件 1: 変更されたファイルのうち命令的モジュールを含んだファイルが 1 つ以上存在する
- 条件 2: レビュースレッドの指摘対象となる変更内容に命令的モジュールが 1 つ以上存在する

これらの条件を視覚的に表したものを図 2，3 および 4 に示す．

条件 1 では，レビュースレッドの指摘箇所に関わらず，変更された全てのファイルのうち命令的モジュールを含むものがあるかどうかによって判断を行う．よって，図 2 は命令的モジュールを含まない PR，図 3 は命令的モジュールを含む PR となる．

条件 2 では，レビュースレッドの指摘箇所が命令的モジュールを含むかによって判断する．図 3 では指摘箇所に命令的モジュールを含まない PR，図 4 では指摘箇所に命令的モジュールを含む PR とした．また，図中のグレーの箇所はファイルに含まれる命令的モジュールを示している．

6.1.3 結果

条件 1 で分類した場合における，PR 全体および各レビュースレッドの所要時間に関する調査結果を図 5 および 6 に示す（以降の図に含まれるラベル IM，DM，IC，DC はそれぞれ，命令型を含むマージ済のもの，命令型を含まないマージ済のもの，命令型を含むクローズされたもの，命令型を含まないクローズされたものを示す）．ここで，命令的モジュール持つ PR は 573 件であり全体の約 21.6%（573/2,652），命令的モジュールを持つレビュースレッドは 1,364 件であり，全体の約 17.2%（1,364/7,945）であった．PR での差が最も大きくなったものはクローズ状態での 75 パーセンタイルの値どうしの比較で約 159 日（42, 902, 418 秒 − 29, 155, 392 秒）の差が生じた．レビュースレッドでの比較でも同様にクローズ状態での 75 パーセンタイルでの差が最も大きくなり，半日（346, 547 秒 − 290, 160 秒）程度の差が生じた．また，全体的な傾向としてマージまでの時間はクローズされるまでの時間よりも短くなることがわかった．

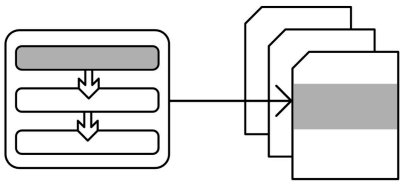


図 4. 命令的モジュールを対象とする PR のレビュースレッド

次に，条件 2 によって分類し，同様に調査を行った結果を図 7 および 8 に示す．また，命令的モジュールを持つ PR は 292 件であり全体の約 11%（292/2,652），命令的モジュールを持つレビュースレッドは 999 件であり全体の約 12.5%（999/7,945）であった．PR の所要時間では，PR がマージされた場合，命令的モジュールを使用する場合の各パーセンタイルでの値が，そうでない場合より長くなる傾向にあった．レビュースレッドでの比較では，マージ状態および，クローズ状態の PR どちらも一貫した結果が得られなかった．

この RQ1 の調査によって，PR およびレビュースレッドの所要時間は，命令的モジュールの使用により長期化する場合があることがわかったが，それらは実用上で問題とするほどの差では無かった．よって命令的モジュールの使用は，PR およびレビュースレッドの所要時間に影響を与えない可能性がある．

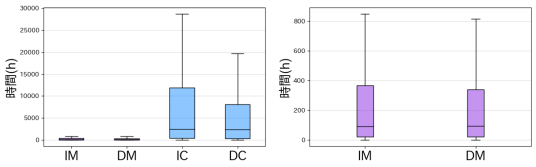


図 5. 条件 1, 変更解決時間

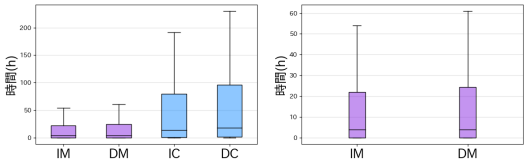


図 6. 条件 1, レビュー時間

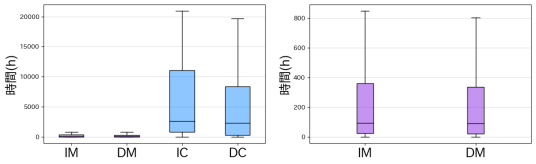


図 7. 条件 2, 変更解決時間

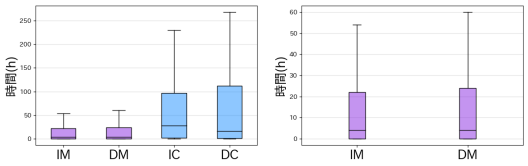


図 8. 条件 2, レビュー時間

6.2. RQ2) PR 中の変更規模に差が見られるか

6.2.1 動機

命令的モジュールはシェルスクリプトを直接実行できるため，自由な記述が可能であり外部スクリプトの実行が可能であるが，その分制御が複雑になり，変更が必要な部分が増加すると考えた．PR 中に変更された行数を調べることで，命令的モジュールを使用する場合に変更の規模が増加しているのか調査した．

6.2.2 手順

PR 中に変更されたそれぞれのファイルでの追加および削除された行数を取得した．新たに 1 行が追加された場合は追加行数 1，ある 1 行が削除された場合は削除行数 1，ある 1 行が一部変更された場合は追加および削除となり 2 行とカウントされる．これらの値をファイルごとに合計したものをファイル変更行数，その PR での各ファイルの変更行数を合計したものを総変更行数とした．変更された全てのファイル数を変更ファイル数とした．RQ1 と同様の条件を用いて命令的モジュールの有無による分類を行い，各パーセンタイルを比較した．

6.2.3 結果

ファイル中に命令的モジュールを含むかどうかによって分類した場合における，PR 中の総変更行数，ファイル変更行数および変更ファイル数に関する調査結果を図 9a，9b および 9c に示す．また，調査対象の PR のうち命令的モジュールを含むファイルは 874 件であり，全体の約 8.0%（874/10,978）であった．クローズされた PR 中の総変更行数の 25 パーセンタイル（図 9a）を除き，全ての四分位数で，命令的モジュールを含む場合に値が増加した．

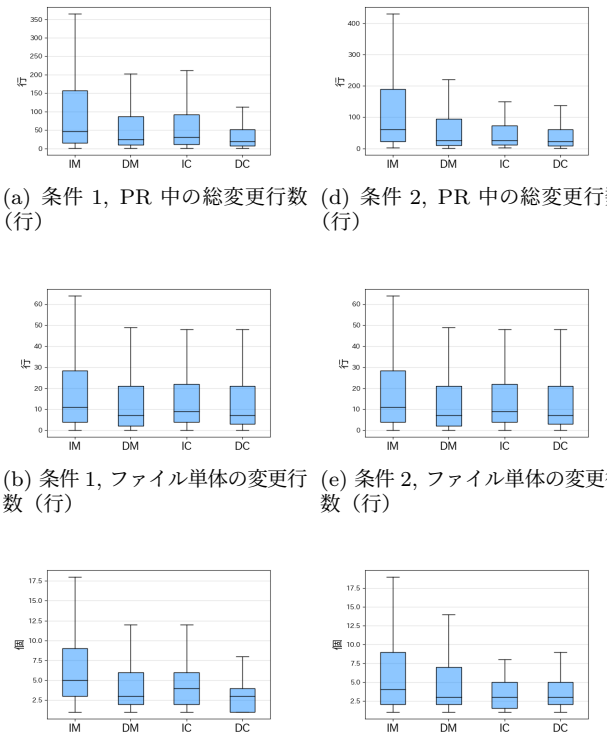
変更箇所での命令的モジュールの有無によって分類した場合における，PR 中の総変更行数，ファイル変更行数および変更ファイル数に関する調査結果を図 9d，9e および 9f に示す．変更箇所に命令的モジュールを含むファイル数は 287 件であり全体の約 2.6%（287/10,978）であった．PR での総変更行数およびファイル単体の変更行数は，最大値を除き，命令的モジュールを使用した場合に変更行数が増加した．変更が行われたファイル数は，クローズされた PR の 25 パーセンタイルを除き，命令的モジュールを含む場合に四分位数の値が増加もしくは同じ値となった．

この RQ2 の調査によって，命令的モジュールの使用による PR の総変更行数，ファイル中の変更行数およびファイル数にそれぞれ増加が見られたため，PR 中の変更規模に差が見られた．よって命令的モジュールの使用は，PR の変更規模を増加させる可能性がある．

7. 議論

7.1. レビュー，変更時間への影響

6.1.3 節で説明した RQ1 の結果から，命令的モジュールを使用した場合および使用しなかった場合の変更解決時間には顕著な差が見られなかった．特にマージされた PR はプロジェクトに変更を与えるが，各パーセンタイル



(a) 条件 1, PR 中の総変更行数（行）
(d) 条件 2, PR 中の総変更行数（行）
(b) 条件 1, ファイル単体の変更行数（行）
(e) 条件 2, ファイル単体の変更行数（行）
(c) 条件 1, 変更ファイル数（個）
(f) 条件 2, 変更ファイル数（個）

図 9. 変更規模の四分位範囲

での時間差があまり見られなかった．また，レビュー時間に関しても同様に，マージされる PR において各パーセンタイルの値に顕著な差が見られなかった．これらの結果から，命令的モジュールの有無によるレビューや変更への時間的影響は少ないと考えられる．

クローズされた PR およびレビュースレッドは，命令的モジュールの使用の有無に関わらずマージされた PR に比べ長期化していたが，その差は顕著であり命令的モジュールの使用の有無とは別の要因がある可能性がある．実際に調査を行った PR のうち，長期間放置された PR に対しクローズを行うボットである stale[3]によってクローズされている PR が複数発見された．このような放置されている PR と，実際に議論が長期化した場合との区別を行っていないため，命令的モジュール有無による影響を調査できていない可能性がある．対策として，ボットにクローズされるような長期間放置されている PR を除外する，議論が活発に行われていない PR を除外する，リポジトリにおけるコアメンバーの PR を重視する等が考えられる．

[3]https://github.com/marketplace/stale

7.2. 変更規模への影響

6.2.3 節で説明した RQ2 の結果から，命令的モジュールをファイルに含んでいる場合はそれらが変更箇所に含まれているかどうかに関わらず，PR 全体の変更行数を増大させていることがわかった．命令的モジュールを変更箇所に含む場合，PR 全体およびファイル単体での変更行数が増加しており，命令的モジュールによる変更規模の増加が考えられる．また，PR 中での変更行数はクローズされた PR よりもマージされた PR の方が多く，より活発に議論および変更が行われている可能性がある．

命令的モジュールを含む場合は同時に変更されたファイル数が増加する場合があり，変更が必要なファイルが多くなる可能性が考えられる．ファイル数増加の理由として，命令的モジュールによる外部スクリプトの呼び出しが考えられる．実際に命令的モジュールを含むいくつかの PR 中の変更ファイルに Python やシェルスクリプトの存在が確認された．

7.3. 研究目的に対する議論

RQ1 の結果より命令的モジュールの使用が，レビューに必要な時間に与える影響は少ないと考えられる．また RQ2 の結果より命令的モジュールの使用が，変更に必要な記述量や関連ファイルに影響を与える可能性が考えられる．

命令的モジュールを使用した場合にファイル数や記述量が増加したにも関わらず，レビューに必要な時間に差が見られなかったため，より効率的にレビューができた可能性がある．理由としては，宣言的モジュールを使用した場合，構成管理ツールによってどのような命令が行われるか，ユーザが直接コードを確認しても不明であることが多く，実行内容の確認が容易ではないため，レビュー時に変更内容の検証が困難である場合があった可能性がある．

本研究では，命令的モジュールの使用は環境との兼ね合いや冪等性の観点から非推奨とされているが，実際にはレビュー時間には変化が見られず，変更規模に関してのみ影響が見られた．先行研究 [11] では追加された行数，コミット数が増えるほど PR の評価までの時間が長期化するとされているが，本研究の結果では，変更行数の増加が見られたが PR の評価までの時間は増大しなかった．そのためソースコードレビューにかかる負担の観点から

は，命令的モジュールの使用は問題ないが，変更行数は増大しているため，変更を行う開発者への負担は増加していると言える．

7.4. 妥当性への脅威

本研究ではデータセットとして Ansible Galaxy にて公開されているロールから GitHub 上のリポジトリを取得したため，GitHub で最も一般的に用いられているレビューツールである PR に着目したが，他にも Phabricator[4]，Gerrit[5]，Atlassian Crucible[6]などのソースコードレビューツールが存在するため，それらが同様の結果を示すかどうか今後の調査が必要である．

PR からそれらの所要時間や変更された行数を抽出したが，それらの値が開発者やレビュワーの技量によって左右される可能性がある [11, 13]．また，レビューの質問に対する回答が放置されている場合や，回答までの期間が異常に長い場合など，プロジェクトの品質にバラつきがあり，これらを同一に扱うべきか検討する必要がある．実際に第三者によって立てられたスレッドの内約半数で議論が行われていなかった．

レビュー箇所に命令的モジュールを含むかどうかによって分類を行っていたが，レビューが実際にそれらのモジュールを対象としていたか疑問の余地がある．前後のコードや指摘内容を吟味する事によってより正確な分類を行うことができると考えられる．しかし，それらの判断には 7,000 件を超えるレビュースレッドを確認する必要があり，場合によっては目視で確認するため，その実行には多大な労力が必要である．

Ansible にのみ絞って調査を行ったため，一般的な結論を得るために他の構成管理ツール（Chef, Pupet, Terraform）でも同様の結果が得られるか確認する必要がある．また，GitHub 上で公開されているリポジトリからのみデータを収集したが，企業での実態が同様の性質を持つかどうか今後の調査が必要である．しかし，これらのデータは機密情報を多く含むため取得は困難である．

8. 結言

本研究では，IaC に関するソースコード開発現場において，命令的モジュールの使用がどのような影響を与えるか調査した．OSS の開発プロセスである PR に着目し，PR が評価されるまでの時間やレビュースレッドの所要時間，変更行数や変更ファイル数の比較を行い，命令的モジュールの有無によって差が見られるか比較を行った．

調査結果から，命令的モジュールの使用によるレビュー時間への影響はそれほど大きなものではなく，変更が行われるファイル数及び行数に変化が見られた．マージされた PR にのみ着目した場合，レビュースレッドおよび PR そのものの所要時間に大きな差は見られず，実際の開発においてそれほど大きな影響ではないと言える．クローズされた PR ではそれまでに放置されていた可能性や，モジュールに関係のないところで時間がかかっていた可能性があるため一概に命令的モジュールが原因であるとは言えない．変更規模に関して，命令的モジュールを使用した場合変更行数，ファイル数ともに増加した．また，変更内容に命令的モジュールを含む場合それらの差がより増大した．これらの結果から，命令的モジュールを使用した場合でも変更規模の増加が生じても，レビューの際に問題となることは少なく，実際のソフトウェア開発現場において命令的モジュールの使用が問題とはならない可能性がある．

本研究では，レビュー時間と変更規模についての生産性に関わる調査を行ったが，命令的モジュールの使用による冪等性の実現性や，ヒューマンエラー率などのソフトウェア品質についての影響は明らかになっていない．そのため，今後の課題として故障間隔や修復時間などの品質面についての調査を行った上で，命令的モジュールの使用がもたらす課題についてさらなる研究を行いたい．

謝辞

本研究の一部は，JSPS 科研費 JP19J23477, JP19K20240, JP18H04094 の助成を受けました．

参考文献

[1] J. Smeds, K. Nybom, and I. Porres, “Devops: a definition and perceived adoption impediments,” In Proceedings of the International conference on agile software development, pp.166–177, Springer, 2015.

[2] R. Jabbari, N. binAli, K. Petersen, and B. Tanveer, “What is devops? a systematic mapping

[4] https://www.phacility.com
[5] https://www.gerritcodereview.com
[6] https://www.atlassian.com/ja/software/crucible

study on definitions and practices," In Proceedings of the Scientific Workshop Proceedings, pp.1–11, 2016.

[3] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, and D.A. Tamburri, "Devops: introducing infrastructure-as-code," In Proceedings of the IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pp.497–498, IEEE, 2017.

[4] W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam, "Testing idempotence for infrastructure as code," In Proceedings of the ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, pp.368–388, Springer, 2013.

[5] J. Gunawardena, "An introduction to idempotency," Idempotency (Bristol, 1994), vol.11, pp.1–49, 1998.

[6] J.K. Baksalary and O.M. Baksalary, "Idempotency of linear combinations of two idempotent matrices," Linear Algebra and its Applications, vol.321, no.1-3, pp.3–7, 2000.

[7] S. Kokuryo, M. Kondo, and O. Mizuno, "An empirical study of utilization of imperative modules in ansible," In Proceedings of the IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), pp.442–449, IEEE, 2020.

[8] RedHat, Inc, "Ansible is simple it automation," https://www.ansible.com, (参照 2021-01-25).

[9] RedHat, Inc, "community.aws.rds – create, delete, or modify amazon rds instances, rds snapshots, and related facts – exapmle," https://docs.ansible.com/ansible/latest/collections/community/aws/rds_module.html, (参照 2021-01-25).

[10] K. Morris, Infrastructure as Code, O'Reilly Media, 2020.

[11] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for it: Determinants of pull request evaluation latency on github," In Proceedings of the IEEE/ACM 12th working conference on mining software repositories, pp.367–371, IEEE, 2015.

[12] GitHub, Inc., "Github graphql api - github docs," https://docs.github.com/en/graphql, (参照 2021-01-25).

[13] Y. Yu, H. Wang, G. Yin, and C.X. Ling, "Reviewer recommender of pull-requests in github," In Proceedings of the IEEE International Conference on Software Maintenance and Evolution, pp.609–612, IEEE, 2014.

深層学習を用いたプログラム品質向上のためのソースコード画像分析手法の提案

小川 一彦
放送大学大学院文化科学研究科
kgg03771@nifty.com

中谷 多哉子
放送大学大学院文化科学研究科
tinakatani@ouj.ac.jp

要旨

我々は，画像認識と物体検出のアルゴリズムを用いて，ソースコードを画像化してプログラムの不具合の可能性のある箇所を指摘可能であることを検証した．不具合を起こすプログラムは見た目に共通点があり，教師あり学習の深層学習を用いて学習を行うことで，同様の不具合のある箇所を指摘すると考えたのである．教師あり学習の深層学習で，画像に変換したソースコードを学習させるため，動物や人工物などを学習する場合と同じ手法，パラメータを用いて学習させても，精度は上がりにくい．プログラムの不具合を推論させるために，画像に変換したソースコードが教師ありの深層学習で正しく学習されなければ，高い精度で推論を行うことはできない．作成した学習データが正しく学習されるため，学習データをどのように作成すればよいか，学習が行なわれるための深層学習のパラメータの選択はどうすればよいかということを明らかにする必要がある．本研究では，画像認識と物体検出のアルゴリズムを用いて深層学習を行う際に，学習のためのデータをどのように作成することで，教師あり学習の深層学習で学習することができるか明らかにする．併せて，各アルゴリズムで作成した学習モデルを使用して被験者のプログラムを推論した結果の比較と検証を行なう．

1. はじめに

近年，システムの規模は増大している．しかしシステムを開発するための費用はシステムの規模に合わせて増大しておらず，寧ろコストダウンを要求されている．コストダウンが行なわれることで，システム開発の各工程では効率化が行なわれ，コストダウンに対応する．システム開発の工程の一つであるプログラムの製造工程もコストダウンの対象となる．システム開発において，品質を確保するため，プログラムの製造工程で行なわれるコードレビューは，重要である．コードレビューではプログラムが規約に沿って正しく記述されていること，仕様書に書かれている機能を満たしているかという事などをチェックし，指摘を行い，プログラムを改善することでソフトウェアの品質を確保している．コードレビューで品質を確保するには工数が掛かるためコストが必要となる．コストダウンは，品質を保ちつつ工数を削減する必要がある．コードレビューで，プログラムにある不具合の可能性のある箇所を予め示すことができれば，示された箇所を重点的にレビューを行うことが可能になり，工数を減らすことが期待できる．

我々は，これまでプログラムのソースコードを色付きの要素を持つ画像に変換して学習と推論を行った [5] [9]．プログラムを作成するとき，プロトタイプからの作成，仕様が近い作成済みのプログラムを再利用し，効率化を図る．共通性が高いことで，不具合の発生個所は，ソースコードの見た目にも共通点があると考えた．

本稿は，畳み込みニューラルネットワーク（CNN）を用いた画像認識と物体検出のアルゴリズムが，画像から動物や人工物を検出するように，ソースコードの不具合の箇所を検出することを検証する．どのような学習データであれば，各アルゴリズムで教師あり学習が行われ，推論が可能な学習モデルが作成できるのか，学習のためのパラメータ（ハイパーパラメータ）とともに示す．

研究の目的は、以下の問いに答えることである．

- 画像分析手法に用いるアルゴリズムは，どのような学習データとパラメータであれば画像化したソースコードの学習が可能になるか？
- 不具合の推論で認識された「怪しい箇所」は，実際

の不具合箇所を正しく認識できているのか？

本稿は，以下の内容で構成されている．第 2 節は，関連研究を示す．第 3 節は，研究のアプローチについて説明する．第 4 節で，結果を示す．第 5 節で，考察を示す．第 6 節で，まとめで，研究の結果と，今後の課題を述べる．

2. 関連研究

これまで，ソフトウェアの品質は，統計学的な手法により推論されてきた．McCabe [10] は、プログラムの複雑さは制御構造の複雑さで定まるとし，循環的複雑度（Cyclomatic complexity）を提唱した．

下村ら [7] は、ソフトウェアの品質評価に実製品データを用いて，単体テスト（Unit Test，以下 UT とする）のメトリクスとソースコードのメトリクスを組み合わせて行った．分析の結果，UT 見逃し欠陥数と他のメトリクスとの相関を見ると，規模，サイクロマチック数，分岐条件数との間に正の相関があるとしている．

また、近藤ら [3] は，ソフトウェアの不具合予測をするため，ソフトウェアの変更をしたソースコード片のみに対して深層学習を適用することで不具合を予測する手法（W-CNN）を提案している．研究で，テキスト分類問題として捉え，ソースコード片による不具合予測を行った．W-CNN を用いて評価した結果，ソースコード片に対して不具合予測の学習は可能であるとしている．

ソースコードを画像化して深層学習で学習させる手法として，森崎ら [4] は，ソースコードの見た目に着目し，深層学習によるレビューを行わせた．研究では，ソースコードの美しさを深層学習で学習させ，美しいソースコードは読みやすいソースコードであり，品質の向上につながるとしている．ソースコードを 100 色程度で色分けして画像のサイズに分割し学習データを作成し畳み込みニューラルネットワーク（CNN）で学習させた．学習結果に基づいてソースコードの美しさを判断した結果，55.5 %が良いと判断したプロジェクトの専門家の所見では可読性と保守性にムラがあるとされ，93.9 %が良いと判断したプロジェクトでは可読性と保守性が均質であり，読みやすいという所見があったとしている．

また，Fowler [8] は，コードの不吉な臭いとは、プログラムのソースコードに深刻な問題が存在することを示す何らかの兆候のことであるとして，いつソースコードのファクタリングを行い、どのようなリファクタリング手法を用いるかを発見するための方法であるとした．

メトリクスを計測することで，不具合の可能性を推測することは可能である．複雑なプログラムに不具合がある可能性は高い．しかし，複雑ではないプログラムにも不具合は混入する．

ベテランのレビューワは，プログラムのレビューをする際に，ソースコードを見て記述の仕方が規約から外れているように見える箇所に不吉な匂いを感じて重点的にレビューをする．同様に，ソースコードの不具合がある箇所を学習し，学習モデルが不具合を認識すればベテランのレビューワのように，ソースコードの見た目から不吉な臭いがする箇所を指摘すると考えた．本稿では，ソースコードを画像に変換し，画像認識と物体検出のアルゴリズムより推論を行なうため，それぞれのアルゴリズムでどのように学習を行なえば推論が可能になるかを明らかにし，実際に推論を行なった結果を検証する．

3. 研究のアプローチ

3.1. 概要

我々は，プログラムのソースコードを画像に変換し，教師あり学習により画像認識と物体検出のアルゴリズムを用いる．本稿の学習及び推論を深層学習で行うため，Windows10 HomeEdition 64bit に，Anaconda 仮想環境を使用して．学習および推論の環境を作成する．

1. 画像認識のアルゴリズムで学習する環境．
   Google の TensorFlow-gpu 1.12.0 に別途 KERAS 2.2.4 をインストールした仮想環境を作成する．

2. 物体検出のアルゴリズムで学習する環境．
   YOLOv3 [6] をインストールした仮想環境を作成する．

ハードウェアの構成は，CPU：Corei7-6700K，Memory：24GB，GPU：GeforceRTX2070Super で実験を行う．また，本稿の画像認識アルゴリズムによる学習と推論は，これまでに作成したプログラムを使用して行なった． [9]

3.2. 研究方法

本研究では，プログラムのソースコードを画像に変換し，プログラム中にある不具合の箇所を，教師あり学習で学習させる．正しく学習が行なわれるため，どのようにして学習データを作成し，学習のパラメータを設定す

ればよいか，教師あり学習の経緯とともに示す．教師あり学習の結果，作成されるモデルを使用して被験者のプログラムを推論させ，不具合の可能性を予測する．
その理由を以下に示す．

1. プログラムの不具合は見た目も共通性がある．
   不具合を起こすソースコードは，コードの記述に特徴があると考えられる．近年は特にプログラム作成でソースコードを再利用することが多く，見た目の共通性が高いと考えた．

2. 複雑ではない箇所のプログラムの不具合を推論する可能性がある．
   メトリクスの計測は，プログラムの複雑な箇所を不具合の可能性が高い箇所として推測する．プログラムの不具合を，見た目で学習することは複雑ではない箇所を，不具合と指摘する可能性があると考えた．

3.3. 研究の手順

本研究の手順は以下の通りである．

1. 画像認識アルゴリズムで学習する．
   プログラムのソースコードを画像化し，教師あり学習で不具合の有無を学習させる．学習には画像認識のアルゴリズムを用いて行なう．

2. 画像認識アルゴリズムで学習した結果を評価する．
   教師あり学習で学習するモデルが，プログラムのソースコードを変換した画像から特徴を捉えるためには，学習データとなる画像をどのように作成すればよいか検証する．また，学習するモデルはどのように学習を行なえば良いか，学習のパラメータと共に検証を行なう．

3. 物体検出アルゴリズムで学習する．
   プログラムのソースコードを画像化し，教師あり学習で不具合の有無を学習させる．学習には物体検出のアルゴリズムを用いて行なう．

4. 物体検出アルゴリズムで学習した結果を評価する．
   画像認識アルゴリズムを用いる場合と同じく，学習データとなる画像をどのように作成すればよいか検証する．

5. それぞれのアルゴリズムで作成した学習モデルで推論した結果を比較および評価する．
   画像認識および物体検出アルゴリズムで作成した学習モデルを用いて，被験者のプログラムで推論した結果を検証する．

3.4. 研究内容

1. 学習モデルの選択

   (1) 画像認識アルゴリズムで使用する学習モデル
   学習モデルは，ImageNet の学習済みモデルである VGG16（VGG ILSVRC 16 layers）を使用する．転移学習を行なうことで，入手本数が限られる，画像に変換されたソースコードによる学習が可能になる．

   (2) 物体検出アルゴリズムで使用する学習モデル
   学習には，YOLO（You look only once）を用いる．学習は，本研究で作成する学習データのみで行なわれる．

2. 学習データの作成．
   画像認識アルゴリズムと物体検出アルゴリズムは，どちらも画像から推論のための特徴量を抽出する．プログラムのソースコードを変換した画像からプログラムの品質を評価するため，ソースコードの構成要素を画像から区別しやすくする．区別しやすくするため，色分けを行なう．色分けは，複数パターンを用意して，どのパターンがより学習されやすいか検証する．

3. 学習パラメータの選択
   学習を行なうため，学習のパラメータ（ハイパーパラメータ）をチューニングし，学習の精度が良いパラメータを検証する．

4. 学習結果のモデルを用いた推論結果の比較と評価
   画像認識アルゴリズムと物体検出アルゴリズムで，同一の被験者のプログラムを使用して推論を行ない，結果を比較および評価する．推論結果の検証内容を以下に示す

   (1) プログラムの不具合を推論した数を計測する
   (2) プログラムで実際に不具合対応された数を計測する
   (3) プログラムの不具合を推論した数のうち，プログラムで実際に不具合対応された数を計測する

(4) 計測結果を，適合率と再現率とF値で評価する

それぞれのアルゴリズムで検証された結果を比較し，推論の手法による違いを評価する．

4. 結果

教師あり学習で，学習結果の評価と推論結果の評価を行うため，学習データと推論データを定性的、定量的に作成する．

4.1. 画像に変換されたソースコードを認識する学習モデル

4.1.1 学習データの作成

画像認識と物体検出の各アルゴリズムで行なう教師あり学習で，学習モデルが認識する学習データを作成する．学習データは，以下の手順で作成する．

1. 学習データの収集.
   同一プロジェクトの575本のプログラムより，5人のプログラマが作成した44本を選択した．この44本全てを教師あり学習に使用する．プログラムは，全てVisualBasic2008で作成されている．

2. プログラムを画像に変換するため，構文の色分けを行う.
   色分けのパターンで，教師あり学習の結果に変化があるか検証するため，2種類のパターンを用意した．
   色分けパターン1

   - 命令文：青（FF0000）
   - 予約語：オレンジ（00A5FF）
   - コメント：緑（008000）
   - モジュール名：明るい緑（太字、90EE90）
   - グローバル変数：濃い青（太字、8B0000）
   - ローカル変数：黒（000000）
   - コントロール名：ピンク（CBC0FF）
   - ユーザ関数，サブルーチン名：赤（0000FF）
   - ユーザ定義名：ブラウン（2A2AA5）
   - 文字列：紫（800080）

   色分けパターン2

   - 命令文：紫（800080）
   - 予約語：黒（000000）
   - コメント：緑（008000）
   - モジュール名：明るい緑（太字）
   - グローバル変数：ピンク
   - ローカル変数：オレンジ（00A5FF）
   - コントロール名：ブラウン（2A2AA5）
   - ユーザ関数，サブルーチン名：黒（000000）
   - ユーザ定義名：濃い青（太字、8B0000）
   - 文字列：濃い青（太字、8B0000）

   色に続く括弧内はカラーコードである．
   画像認識アルゴリズムでは，色分けの種類に加えて，背景色が白と黒の2種類，構文の色分けと併せて4種類の学習データを作成した．学習データの違いを，図1に示す．物体検出アルゴリズムでは，命令文などの色分けで2種類の学習データを作成している．

3. 変換したプログラム画像より学習データを作成.

   (1) 画像認識アルゴリズムの教師あり学習で使用する学習データ
   プログラムのソースコードを，B5横のサイズに合わせて作成する．学習と推論は，255*255ピクセルに画像を縮小するため，これ以上のサイズでは判別が難しい．判別可能にするため，1枚の画像に30行，フォントはMSゴシックで6.8pt，1行は135文字，1行に収まらない場合は行を折り返す様にする．

   (2) 物体検出アルゴリズムの教師あり学習で使用する学習データ
   プログラムのソースコードを，A4横のサイズに合わせて作成する．学習と推論は，416*416ピクセルに画像を縮小するためである．判別可能にするため，1枚の画像に60行，フォントはMS ゴシックで9pt，1行は120文字，1行に収まらない場合は行を折り返す様にする．

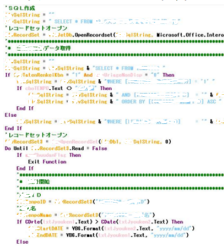
   ソースコードの画像化にあたり，画像化のためのツールを作成した．不具合の修正前と修正後のソースコードを用意して，プログラムの内容を比較する．

画像化されたソースコードは，画像 1 枚単位でフォルダに格納される．

色分けパターン1、背景色白

色分けパターン1、背景色白

色分けパターン2、背景色白

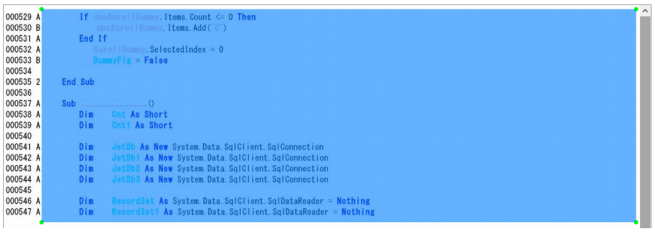


色分けパターン2、背景色黒

図 1. 画像認識アルゴリズムで学習に使用する学習データ

4. 学習データのアノテーション情報を作成．
物体検出アルゴリズムを適用した深層学習で学習を行うため，学習するカテゴリと不具合の箇所であるソースコード片を紐づける必要がある．画像認識アルゴリズムではアノテーション情報は作成しない．アノテーションデータは，学習するカテゴリと画像内の位置情報を記録している．データの作成には，フリーソフトである「labelImg」を使用する．アノテーションデータは以下の 2 通り作成する．

行全体を学習領域とした場合



ソースコードのみ学習領域とした場合

図 2. アノテーション情報の違い

(1) 行全体を学習領域とするアノテーション情報

(2) ソースコードの構文のみを学習領域とするアノテーション情報

アノテーション情報の設定方法の違いを図 2 に示す．各アルゴリズムとも，教師あり学習の検証データはホールドアウト法で選択した．ホールドアウト法では，学習データの一部を検証のためのデータとして予め選択する．検証に使用されるデータが固定されるため，学習データ全体より，異なる見た目や行数の組み合わせが網羅されるよう検証に用いるデータを選択した．選択した検証データで，教師あり学習はクロスバリデーション（交差検証）を行う．

5. 学習データの件数．
教師あり学習では，学習データに正解のラベルを付与して学習を行なう．このラベルを学習のカテゴリとする．

(1) 画像認識アルゴリズムで使用する学習データ

- 学習カテゴリ
  2 カテゴリ（不具合なし，不具合あり）
- 学習に使用する画像の枚数
  不具合あり 945 枚
  不具合なし 545 枚
- 検証に使用する画像の枚数
  不具合あり 213 枚
  不具合なし 109 枚

(2) 物体検出アルゴリズムで使用する学習データ

- 学習カテゴリ
  1 カテゴリ（不具合あり）
- 学習に使用する画像の枚数
  863 枚
- 検証に使用する画像の枚数
  98 枚

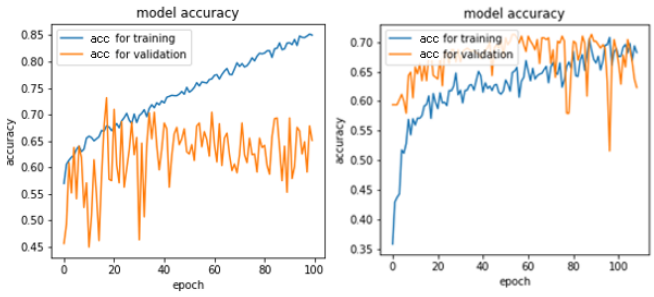
4.1.2 学習結果

1. 各アルゴリズムによる教師あり学習を行う．

(1) 画像認識アルゴリズムによる学習結果学習は，学習済みデータセットである VGG16 [1] の，最後の畳み込み層である BLOCK5 と全結合層を，研究で使用する学習データで学習した．最終結合層を，softmax 関数で 2 カテゴリの出力として学習している．学習の結果，学習モデルが反応しないか過学習となった．学習が正しく行われるよう，学習のパラメータをチューニングした．学習で試行したパラメータの変更を以下に示す．

- Data Augmentation
  `height_shift_range`,
  `channel_shift_range`,
  `brightness_range`, `mix_up`,
  `random_crop`, `random_eraser`
- 学習のコンパイルパラメータ
  `Optimizer`（`SGD`, `Adam`），L2 ノルム正則化

学習のパラメータチューニングを行なった結果，過学習を解決することは出来なかった．学習のパラメータで解決できなかったため，モデルのチューニングを行なった．チューニング



Flattenを使用　　Global Average poolingを使用

図 3. 学習経過の違い

を行なった結果，全結合層で，Flatten [1] ではなく Global Average Pooling(GAP) [2] を使用することで，過学習のない学習結果を示すようになった．学習の経過の違いを，図 3 に示す．

図 3 のグラフは，縦軸が適合率，横軸が学習回数である．青い線は学習データのうち学習に使用したデータの適合率，オレンジの線は検証に使用したデータの適合率である．青い線とオレンジの線が共に右上がりのグラフが良い学習結果であり，過学習ではオレンジの線である検証用データの適合率が向上しない．Global Average Pooling を使用した学習モデルが，Flatten を使用する方法より過学習を抑制していると判断した．プログラムを画像化して学習する場合に，学習データのうち検証に使用したデータ（validation）の適合率が向上していることが図 3 より確認できるためである．

試行した結果，教師ありに使用するモデルとパラメータを以下に示す．

i. 最終結合層のモデル
```
GlobalAveragePooling2D()(x)
Dense(256, activation='relu')(x)
Dropout(0.5)(x)
```
出力は、softmax 関数を使用する．

ii. Data Augmentation
`channel_shift_range`,
`brightness_range`, `mix_up`,
`random_eraser`

iii. 学習パラメータ

```
Optimizer SGD. lr=0.00002,
nesterov=True, decay=1e-6
```

学習のカテゴリは，不具合ありと不具合なしの 2 カテゴリである．

4 種類の学習データで学習した結果，表 3 の通りであった．

表 1. 学習結果

色分けパターン	背景	学習の適合率	検証の適合率
パターン 1	白	0.69	0.65
パターン 1	黒	0.69	0.64
パターン 2	白	0.67	0.65
パターン 2	黒	0.70	0.60

学習を行なった結果，色分けパターンと背景色の組み合わせは，プログラムのソースコードを画像に変換して教師あり学習する場合，学習結果に大きな違いはなかった．推論には，色分けパターン 1 で背景色が白である学習データを用いて教師あり学習を行なった結果の学習モデルを用いた．

(2) 物体検出アルゴリズムによる学習結果学習は，物体検出アルゴリズムである YOLOv3 を使用して行う．学習データは，色分けを行なった 2 種類のデータに，アノテーション情報で行全体と構文のみの 2 種類，合わせて 4 種類の学習を行なった．学習データのカテゴリは，不具合ありの 1 カテゴリである．学習のパラメータのチューニングは 1 カテゴリとしたことによる変更を行なっている．

4 種類の学習データで学習した結果，表 5 の通りであった．適合率（mAP）は全カテゴリの平均適合率であり，平均 IoU は検出したバウンディングボックスが正確に対象を囲んでいる割合で 0.5 以上 [6] が目安である．学習回数は，1 学習カテゴリあたりで，2,000 回以上が推奨されている [6]．学習の経過を確認するため，推奨値を超える 5,000 回で学習を行なった．

適合率，平均 IoU は，平均 IoU が一番高い数値を出した時点の数値である．適合率，再現率は学習回数が 2,000 回の時点の成績が良い結果となった．学習カテゴリは，1 つであるため，学習回数の推奨値は 2,000 回 [6] である．推奨値の 2,000 回であるため問題ないと判断した．

表 2. 学習結果

色分けパターン	アノテーション	適合率（mAP）	平均 IoU
パターン 1	行全体	0.95	0.66
パターン 1	構文	0	0
パターン 2	行全体	0.95	0.65
パターン 2	構文	0	0

学習は，行全体を学習領域とした場合に，学習が行なわれている．構文のみを学習領域とした場合は，学習が進まなかった．推論には，色分けパターン 1 でアノテーションに行全体を学習領域として教師あり学習を行なった結果の学習モデルを用いた．

4.2. 学習モデルを用いた推論処理

各アルゴリズムで教師あり学習した結果のモデルを使用して，被験者として選択した，5 人のプログラマが作成したプログラムを使用して，不具合の可能性を推論した．

被験者のプログラムの推論データの規模を以下に示す．

1. 画像認識アルゴリズムで使用する推論データ被験者のプログラムの規模を表 3 に示す．

表 3. 画像認識アルゴリズムの被験者のプログラムの規模

No	プログラムの行数	推論する画像の枚数
a	897	18
b	1,669	19
c	3,471	61
d	7,143	138
e	12,538	221

2. 物体検出アルゴリズムで使用する推論データ被験者のプログラムの規模を表 4 に示す．

表 4. 物体検出アルゴリズムの被験者のプログラムの規模

No	プログラムの行数	推論する画像の枚数
a	897	14
b	1,669	27
c	3,471	61
d	7,143	116
e	12,538	209

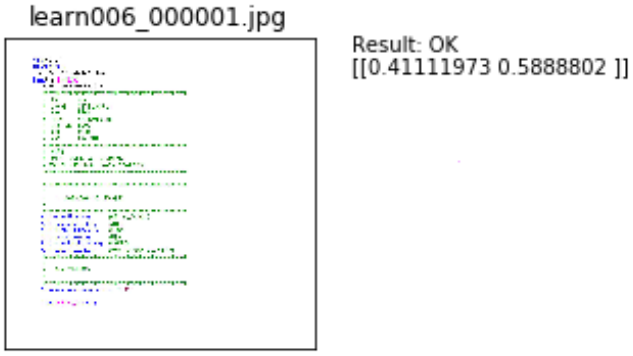


図 4. 画像認識の推論結果

No　…ソースコードに付与した連番
不具合判定数　…推論で不具合と判断した画像の枚数
実際の不具合数　…不具合対応した画像の枚数
正解数　…推論と不具合が一致した画像の枚数

図 5. 物体検出の推論結果（閾値なし）

4.2.1　推論処理の結果

画像認識アルゴリズムによる推論の出力結果を，図 4 に示す．推論の結果は，カテゴリ別に合計が 1 になるよう出力される．数値の大きい方を推論の結果とした．
画像認識アルゴリズムによる推論の検証結果を，以下の表 5 に示す．

表 5. 適合率・再現率・F 値

No	不具合判定数	実際の不具合数	正解数	適合率	再現率	F 値
a	8	9	6	0.75	0.67	0.71
b	14	3	2	0.14	0.67	0.24
c	28	15	7	0.25	0.47	0.33
d	18	13	3	0.17	0.23	0.19
e	91	59	41	0.45	0.69	0.55

物体検出アルゴリズムによる推論の出力結果を，図 5 に示す．推論の結果は，不具合の可能性がある個所をバウンディングボックスで囲むことで指摘する．バウンディングボックスが重複しなくなるまで，閾値の設定を 0.05 単位で変更と確認を行ない，閾値を 0.8 に定め，推論の結果とした．閾値を指定した出力結果を図 6 に示す
物体検出アルゴリズムによる推論の検証結果を，以下の表 6 に示す．

表 6. 適合率・再現率・F 値

No	不具合判定数	実際の不具合数	正解数	適合率	再現率	F 値
a	8	4	2	0.25	0.5	0.33
b	5	1	1	0.2	1	0.33
c	9	3	2	0.22	0.66	0.33
d	38	13	3	0.07	0.23	0.11
e	54	48	19	0.35	0.39	0.37

また，物体検出アルゴリズムによる推論結果を，循環的複雑度（Cyclomatic Complexity Number，以下 CCN）で計測した結果と比較した．循環的複雑度の基準値は C4 Software Technology Reference Guide [11] で High Risk となる 21 とした．比較した結果を，表 7 に示す．物体検

図 6. 物体検出の推論結果（閾値あり）

表 7. 循環的複雑度の計測値と物体検出の推論結果の比較

No	CCN が 21 以上の数	物体検出で一致しない数	一致せず正解した数
a	1	2	1
b	1	2	0
c	5	3	0
d	6	23	1
e	17	19	7

出で一致しない数は，物体検出の推論した箇所の CCN が 21 未満となる件数，一致せず正解した数は物体検出で一致しない数で物体検出が正解した件数である．物体検出で推論した結果，検出した行数の平均は，33 行で分散は 18.23 であり，CCN が 21 以上の時の行数の平均は 405 行で分散は 51499.5 であった．

4.2.2 推論結果の傾向

各アルゴリズムで推論を行なった結果，画像認識アルゴリズムで推論した方が精度（F 値）が高かった．適合率と再現率では，画像認識アルゴリズムでは適合率が高く，物体検出アルゴリズムでは再現率が高い結果となった．

4.3. 各アルゴリズムの推論結果の評価

各アルゴリズムがプログラムの不具合の可能性を推論した結果を比較した．

1. 画像認識アルゴリズムの推論結果の評価
   - F 値は最小 0.19 で最大 0.71
   - 適合率，再現率では適合率が良い結果
   - 画像 1 枚に不具合が含まれていることを推論するため推論対象の範囲が広い
2. 物体検出アルゴリズムの推論結果の評価
   - F 値は最小 0.11 で最大 0.37
   - 適合率，再現率では再現率が良い結果
   - 画像 1 枚のどこに不具合があるかを推論するため推論対象の範囲が狭い

比較した結果，全体の精度は画像認識アルゴリズムの方が，プログラムの不具合の可能性を推論する．しかし，物体検出アルゴリズムの方が，詳細に不具合の可能性を推論することが出来ている．

5. 考察

5.1. 画像分析手法に用いるアルゴリズムでソースコードの不具合を学習することは可能か

プログラムの構文の色分けを行い学習データを作成したが，背景色や構文の色を変更しても学習結果に差が無いため，画像認識と物体検出の各アルゴリズムで色分けの違いは，影響がないと考えられる．画像認識アルゴリズムでは，学習モデルの全結合層に GAP [2] を使用することで過学習を起こさなくなった．ソースコードを変換した画像は，画像分析手法で分析する動物や自然物と比較すると，画像の模様は単純である．GAP を使用することで，教師あり学習の収束が緩やかになったことが，過学習を抑制した原因だと考えられる．物体検出アルゴリズムでは，学習させるプログラムの不具合の箇所を，インデントを含めて教師あり学習させることで過学習を起こさなくなった．ソースコードの構文を変換した画像は，インデントを含めることで，画像ごとの模様の複雑さが増し，インデントを含めない場合と比較して特徴を捉えやすくなると考えられる．画像分析手法に用いるア

ルゴリズムでソースコードの不具合を学習することは，学習の経過から過学習が進みすぎる事なく学習された．

5.2. 不具合の推論で認識された「怪しい箇所」は，実際の不具合の箇所を指摘したか

各アルゴリズムで被験者のプログラムより，プログラムの不具合の可能性がある「怪しい箇所」を推論した結果，プログラムにある不具合の箇所を指摘することは，推論しやすい不具合と推論しにくい不具合があった．画像認識アルゴリズムの精度は最大で 0.71，物体検出アルゴリズムの精度は最大で 0.37 であった．どちらのアルゴリズムでも埋め込み SQL 文の推論がよくできており，構文の不具合はインデントが大きく変わるような場合に推論出来ているものがあるが，変数名など部分的な修正がされている場合は推論できなかった．

プログラムの「怪しい箇所」は，完全ではないが，推論しやすい不具合で最大 7 割指摘している．また，物体検出アルゴリズムでは循環的複雑度と比較して複雑ではない箇所を 9 箇所指摘した．メトリクスと組み合わせて推論の精度を向上させることが考えられるが，これは課題となる．

しかし，プログラムの不具合を推論した結果は，偶然精度が高かった可能性を否定することが出来ていない．偶然精度が高かった可能性を否定するため，全結合層にランダムフォレストを適用し，Boruta パッケージで判別に寄与しない特徴量を意図的に与え特徴を選択させることが考えられるが，学習モデルに適用することと，実際にレビューワがソースコードから感じた不吉な匂いのする箇所と，各アルゴリズムで教師あり学習と推論をした結果が一致するか検証を行なうことは課題である．

6. まとめ

我々は，プログラムの不具合は，ソースコードの構文の形に共通点があるとして，深層学習による画像分析手法である画像認識と物体検出のアルゴリズムで分析することを提案した．画像認識アルゴリズムでは，不具合を含む箇所を，最大で 7 割，物体検出アルゴリズムでは，不具合を含む箇所を，最大で 4 割弱の精度で指摘できた．

本稿の問いに対する答えを以下に示す．

- 教師あり学習は，画像認識アルゴリズムではモデルの結合層に GAP を使用することが効果的であり，物体検出アルゴリズムでは，不具合のあるソースコードの構文を，インデントを含めて教師あり学習させることで学習を進めることができた．
- プログラムの不具合を推論することで認識されたプログラムの「怪しい箇所」は，実際の不具合を認識できた箇所もあったが，出来ない箇所もあった．

今後，課題を解決し，実システム開発で，レビューおよびテストを行う際に，本研究で推論した結果と比較および検証を行ない，工数の削減が可能になることを検証したい．

参考文献

[1] K.Simonyan, A.Zisserman: Very Deep Convolutional Networks for Large-Scale Image Recognition, CoRR, pp. 1409-1556, Apr. 2015.

[2] M.Lin, Q.Chen, S.Yan: Network In Network, International Conference on Learning Representations 2014, Mar. 2014.

[3] 近藤将成, 森啓太, 水野修, 崔銀恵: 深層学習による不具合混入コミットの予測と評価, ソフトウェアエンジニアリングシンポジウム 2017，pp. 35-45, Aug. 2017.

[4] 森崎 雅稔: ディープラーニング活用事例と使いこなしの勘所 :［最適化・推論分野］6．AI によるソースコードのレビュー -ディープラーニングでコードの美しさを診断する-, 情報処理, Vol.59, number.11, pp. 985-988, Oct. 2018.

[5] K.Ogawa, T.Nakatani: Predicting Fault Proneness of Programs with CNN, 11th International Conference on Agents and Artificial Intelligence, Vol.1, pp. 321-328, Feb. 2019.

[6] J.Redmon, S.Divvala, R.Girshick, A.Farhadi: You Only Look Once:Unified, Real-Time Object Detection, University of Washington, Allen Institute for AI, Facebook AI Research, May. 2016.

[7] 下村哲司, 森岳志, 野中誠, 佐藤孝司: ソフトウェアメトリクスを用いた単体テストの品質リスク評価, ソフトウェア品質シンポジウム (Web), A3-2, 2013.

[8] M.Fowler: Refactoring:Improving the Design of Existing Code Second Edition, Pearson Education, Inc, 2019.

[9] 小川一彦，中谷多哉子: CNN-BI システムの複数モデルによる精度向上のための研究, ソフトウェア・シンポジウム 2020 論文集, pp.55-64, 2020.

[10] T.J.McCabe: A Complexity Measure, IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, pp. 308-320, Dec. 1976.

[11] J.T.Foreman, J.Gross, R.Rosenstein, D.Fisher, K.Brune: C4 Software Technology Reference Guide: A Prototype, Software Engineering Institute, pp.145-150, Jan. 1997.

Arduinoプロジェクトにおける Example Sketchの再利用分析

寺村 英之
京都工芸繊維大学
h-teramura@se.is.kit.ac.jp

崔 恩瀞
京都工芸繊維大学
echoi@kit.ac.jp

近藤 将成
九州大学
kondo@ait.kyushu-u.ac.jp

吉田 則裕
名古屋大学
yoshida@ertl.jp

水野 修
京都工芸繊維大学
o-mizuno@kit.ac.jp

要旨

ソフトウェア開発を効率的に行うためのソースコードの再利用はコードクローンを作り込む原因となる．コードクローンの存在はソフトウェアの品質に影響を与えうるため，これらの調査が必要である．例えば組み込みシステムの開発に用いられる *Arduino* にはコード再利用の仕組みがあり，再利用の調査ができると便利であると考えられる．しかし我々の知る限りでは *Arduino* におけるソフトウェアプロジェクトであるスケッチの再利用に関する調査は行われていない．本研究はスケッチでの再利用調査をするために *Arduino* に特化したコードクローン検出手法を提案し，実際にスケッチから再利用と考えられるコードクローンを検出した．

1. 緒言

ソフトウェア開発では既存のソフトウェアのソースコードからその一部を再利用することがある．Sojer と Henkel がオープンソースソフトウェア開発者へのインタビューと既存の研究をもとに調査した結果，開発者は新たな機能の実装時に平均して 30%の割合でソースコードを再利用していることがわかった [1]．既存のソースコードの再利用によってコードクローンが生成される．コードクローンとは，ソースコード中に存在する互いに一致または類似したコード片を指す [2]．ソフトウェア中に存在するコードクローンを調べることでソフトウェア開発において有用な情報を得ることができる．例えばプロジェクト内でのコードクローンの調査結果に基づいて開発者は，よく使われている部分のライブラリ化や機能が似ているモジュールの統合にその情報を使うことができる．また Imoize ら [3] は再利用のメトリクスによって Reuse Library の評価を行えることを指摘している．Reuse Library とは再利用可能なソースコードを管理し，開発者によるソースコードの再利用を補助するシステムである．

Arduino はプロトタイピングや組み込みシステムの開発などに利用されている身近な電子工作プラットフォームの 1 つである．開発者が開発するソフトウェアはスケッチと呼ばれる．スケッチは Arduino ボードを PC に接続して Arduino IDE(Integrated Development Environment) から 1 クリックでアップロードするだけで動作させることができ，LED ランプなどの電子回路をその場で動作させながら開発できる．

Arduino はライブラリというソースコードの再利用を容易にする仕組みを備えている．ライブラリにはサンプルコードである Example Sketch が付属する．開発者は Example Sketch の一部を自身のプロジェクトに再利用することがあるので，コードクローンが発生する．Arduino は組み込みシステムの開発に用いられることがあり，その際は発生するコードクローンを把握することが重要である．例えば開発者がソフトウェア中のコードクローンについての情報を把握できれば，コードクローンに関係する問題が発生した場合に迅速に対応できる．また，Arduino プロジェクトからコードクローンを検出することで，ソースコードの再利用を見つけられれば，開発における Example Sketch などの役割を議論できる．

しかし，我々の調査した限りでは Arduino スケッチにおけるソースコードの再利用の調査についての研究は存在しない．

そこで本研究では Arduino におけるソフトウェア開発でのコードの再利用の調査を目的として既存のコードクローン検出ツールを組み合わせた調査方法の検討と再利用調査をした．特に Arduino における再利用の仕組みの 1 つであるライブラリの Example Sketch からのコードの再利用に注目した．この研究における研究設問を以下に示す．

RQ1 Arduino における Example Sketch からのコードの再利用を既存の方法を応用して検出するにはどうすればよいか．

RQ2 再利用されているコードの特徴はなにか．

この研究の貢献は次の 2 つである．

- Arduino スケッチにおける Example Sketch からのコードの再利用を既存のコードクローン検出ツールを応用して検出した．
- 検出されたコードクローンの内，再利用と考えられるものの特徴について議論した．

最後にこの論文の章構成について簡潔に述べる．第 2 章ではこの研究の関心の対象である Arduino とソースコードの分析に利用するコードクローン検出ツールについて説明する．第 3 章では既存のコードクローン検出ツールを応用して Arduino スケッチにおけるソースコードの再利用を検出する方法を提案する．続いて第 4 章でその手法を用いた再利用の検出結果を示す．その後第 5 章で研究設問に答えるための議論をする．最後に第 6 章で妥当性への脅威について述べ，第 7 章でこの研究の結論と貢献についてまとめる．

2. 背景

2.1. Arduino における Example Sketch

Arduino のスケッチは一般的な IDE でいうプロジェクトと対応する．スケッチには唯一のスケッチファイルと任意の数の C あるいは C++ソースコードやヘッダファイルが含まれる．作成したスケッチは Arduino IDE でコンパイルし Arduino ボードにアップロードするとボード上で動作させられる．

開発者は必要に応じてライブラリを自分のスケッチで使うことができる．ライブラリのヘッダファイルをスケッチのソースコードでインクルードするとその機能を使うことができる．

Arduino ライブラリはその使い方を説明するための Example Sketch を含むことがある．多くのライブラリでそれが提供する機能や対応しているハードウェアの種類ごとに分類されたスケッチが用意されている．これらは IDE から簡単に一覧表示でき，またボードにアップロードできる．このようなスケッチをライブラリの Example Sketch と呼ぶ．

Example Sketch はライブラリの利用時に頻繁に使うコードをしばしば含んでおりそこからの再利用はスケッチの開発においてよく行われる．例えば Ethernet ライブラリの WebClient という Example Sketch には Ethernet シールドの初期化処理と LAN への接続処理が含まれている．Ethernet シールドを使いたい開発者はこの部分をコピーして自分のスケッチで簡単にシールドを使い始めることができる．

2.2. コードクローン検出ツール CCFinderSW

本節では，コードクローンの定義とこの研究で利用したコードクローン検出ツールである CCFinderSW[4] について説明する．

コードクローンとは，ソースコード中に存在する互いに一致または類似したコード片を指す [2]．CCFinderSW は意味的ではなく，構文的に類似しているコード片をコードクローンとして検出する．例えばある配列に連続して値を保存する部分があるとする（リスト 1）．これに対してあるアドレスにあるバッファに対してデータを格納する部分を考える（リスト 2）．ここでリスト 1 の 2〜5 行目とリスト 2 の 2〜5 行目はどちらも配列への値の格納だからコードクローンである．このように完全一致するコード片だけでなく相違があるものもある種のコードクローンとして考える．コードクローンはどれだけの相違を許すかによって 4 つのタイプに分類されている [5]．

次に提案手法で利用したコードクローン検出手法である CCFinderSW について説明する．CCFinderSW はトークンベースのコードクローン検出ツールの一つである．トークンベースとはソースコードの比較にプログ

リスト 1. 配列にデータを保存するコード片

```
1 int array[4];
2 array[0] = 0;
3 array[1] = 1;
4 array[2] = 2;
5 array[3] = 3;
```

リスト 2. バッファにデータを格納するコード片

```
1 int create_packet(char * buffer) {
2   buffer[0] = 0x20;
3   buffer[1] = 0x3F;
4   buffer[2] = 0x00;
5   buffer[3] = 0xFF;
```

ラミング言語のトークンを単位として用いるという意味である．CCFinderSW がコードクローンを検出する仕組みを説明する．まず入力されたソースコードからコメントを除去し，予め与えられた文法定義に基づいて入力ソースコードをトークンに分割する．次にトークン列に含まれる英数字列を対応するプログラミング言語の予約語リストを元に予約語と識別子に分ける．その後，比較したいソースコードのトークン列同士で比較を行い一致している部分をコードクローンとして検出する．つまり識別子や数値はその値にかかわらず同じものとして扱われるのでソースコード内の変数名や定数が異なっていてもソースコードのトークン列が似ているならばコードクローンとして検出する．CCFinderSW は Java 言語で書かれたコマンドラインツールとして実装されている[1]．CCFinderSW でコードクローンを検出するには，対象のソースコードをディレクトリに集めて実行するだけで良い．また結果の出力フォーマットが決まっているので処理結果を機械的に処理することもできる．

2.3. Arduino プロジェクトでの再利用分析の必要性

本節では，Arduino プロジェクトでの Example Sketch の再利用分析の必要性について述べる．

まず，実世界にあるスケッチについてどの Example Sketch が頻繁に再利用されているのかを知ることで Example Sketch がスケッチ開発にどれほど影響を与えているか調べられる．例えば似た機能を実装している複数のスケッチに対して調査することである機能を実現することに関連したライブラリの機能を知ることができる．

[1] https://github.com/YuichiSemura/CCFinderSW

さらに，スケッチの開発者が自分のスケッチを分析することで利用していたライブラリや再利用した Example Sketch を知ることができる．これによりどのライブラリを使っていたか記録されていなかった場合にその情報を復元できる可能性がある．また，スケッチ内の Example Sketch からの再利用部分を管理することにも役立つ．

このように一般的なソフトウェアでの再利用分析の利点だけでなく，Arduino における開発特有の利点もある．

3. 提案手法

3.1. コードクローン検出

Example Sketch のコードからの再利用はコードクローンとして検出できる．例えばある Example Sketch のコードの一部がスケッチにコピーされているとする．このときその Example Sketch とスケッチとの間のコードクローンは該当再利用箇所を含む可能性がある．また，スケッチがある Example Sketch をもとにして作られている場合はその Example Sketch とのコードクローンとなる．したがって両者の間のコードクローンを検出しその結果を調べることで Example Sketch から再利用されたコードを知ることができる．

既存の研究では様々なコードクローン検出ツールが提案されている [6][7]．本研究でコードクローン検出ツールを選択する上で重要な要素は 2 つある．1 つは C++ 言語に対応していること，もう 1 つは関数単位，あるいはそれよりも細かい粒度でコードクローンを検出できることである．スケッチの主要なソースコードであるスケッチファイルとそれに付随するソースコードはすべて Arduino 言語で書かれている．そして Arduino 言語は C++言語をもとにしているので検出手法が C++言語に対応していることが重要となる．また，Example Sketch からの再利用ではファイル単位よりも小さなクローンが多く発生すると考えることができる．なぜならば開発者は Example Sketch を再利用した結果それとほとんど同じスケッチを作成する可能性が非常に低いからである．

従って，本研究では C++言語のサポートがあり，かつ細かい粒度でクローン箇所を検出できる CCFinderSW をコードクローン検出ツールとして選択した．

2.2 節で説明した通り，CCFinderSW でソースコード間のコードクローンを見つけるためには対象となるソースコードをすべて CCFinderSW の入力として与える必

要がある．したがって，スケッチと Example Sketch との間のコードクローンを見つけるにはそれらのソースコードをすべて入力すれば良い．しかし，これだけでは 1 つのスケッチに対してかなり時間がかかってしまう．実際，検出の最初の試みとして後述するスケッチ big_aquatan と様々な Example Sketch とを入力とし，研究室の計算機で CCFinderSW を実行したところ 3 日以上経っても検出が完了しなかった[2]．

この処理に時間がかかるのは入力の大きさに原因があると考えられた．まず，ライブラリの種類は IDE からデフォルト設定で検索できるものだけでも 3,560 ほどある．それぞれのリリースされたバージョンも合わせれば全体で約 16,000 種類になる．あるスケッチで使われるライブラリはこの数よりもかなり少ない上，実際に参照された Example Sketch も一部にとどまると考えられる．したがって，最新版ライブラリのすべての Example Sketch と big_aquatan のソースコードを入力として CCFinderSW に渡す場合，多くの Example Sketch に対する処理が無駄になる．また，CCFinderSW は入力として与えられたソースコードから 2 つのソースコードを選んだペアを作り，コードクローンを検出する．ところが，本研究で注目したいものは Example Sketch とスケッチの間のコードクローンである．このまま入力すれば Example Sketch 同士の分析も行われてしまうのでほとんどの処理が無駄になる．そこで効率よくスケッチと Example Sketch のコードクローンを検出を行えるように CCFinderSW への入力を作成する手法を作成した．

3.2. 作成した手法の説明

3.1 節では単に Example Sketch とスケッチを入力とすると，スケッチで使われていない Example Sketch についても分析が行われる（問題 1），Example Sketch 同士についても分析が行われる（問題 2）という 2 つの問題が生じることを説明した．

ここでは本研究で作成した手法で以上の問題をどのように解決したかを説明する．

問題 1 はスケッチと関係があると考えられる Example Sketch を絞り込むことで解決した．まず，ライブラリごとにライブラリのソースコードであるヘッダファイルの集合を作る．次に，すべての Example Sketch のスケッチファイルについてその中でインクルードされているヘッダファイルの集合を作る．その後 Example Sketch のヘッダ集合と対応するライブラリのヘッダファイル集合の積をとり，その Example Sketch の特徴ヘッダファイル集合とする．スケッチから Example Sketch を絞り込むには次のようにする．まず，スケッチのスケッチファイルでインクルードされているヘッダファイルの集合を作る．その後すべての Example Sketch の特徴ヘッダファイル集合についてそれがスケッチのヘッダファイル集合に含まれているか確かめる．そのとき，もし含まれているならば対応する Example Sketch を分析対象に入れる．特徴ヘッダファイル集合はスケッチから利用できるライブラリのヘッダファイルのうち Example Sketch で使われているヘッダファイルの集合である．そのヘッダファイルの集合がスケッチでインクルードされているということは対象の Example Sketch で使われている機能が使われている可能性が大きい．これによってスケッチで使用されようとしている機能に最も近い Example Sketch を選ぶことができる．

問題 2 はスケッチと Example Sketch のソースコードを予めペアにして入力とすることで解決した．まず，スケッチと Example Sketch のソースコードをそれぞれ 1 つずつペアにしたすべての組み合わせを作成する．次に，それらのペアごとにディレクトリを作りそれぞれのソースコードを格納する．そして作成したすべてのディレクトリを入力として CCFinderSW を実行し結果を収集する．これによって比較する意味のないペアを除いて分析できる．

3.3. 検出の条件

本節では提案手法を用いて実際に再利用検出を行った際の各種条件を述べる．

検出対象のライブラリ群として Arduino IDE のデフォルトライブラリメタデータ[3]にあるものを用いた．メタデータとライブラリのダウンロードはどちらも 2021 年 1 月 4 日から 1 月 5 日のうちに行った．ライブラリは計算リソースの都合上ダウンロードした時点で最新バージョンのものだけを検出対象にした．CCFinderSW のバージョンは 1.0 を用い，CCFinderSW のパラメータは表 1 のように設定した．

[2]CPU は Intel Xeon Silver 4210，RAM は 128GB，作業用 SSD は 512GB の NVMe SSD だった

[3]https://downloads.arduino.cc/libraries/library_index.json

表 1. CCFinderSW のパラメータ

パラメータ	値
対象の言語 (-l)	cpp
検出対象の範囲 (-w)	2（ファイル間のみ）
-antlr	”ino|pde|c|h|cpp|hpp|cxx|hxx|cc”
文字コード (-charset)	auto

この表に記述がないものは入力ディレクトリと出力ファイル名に関わるオプションである”-d”と”-o”を除いてすべてデフォルト値を指定した．また，Arduino 言語のファイルも分析対象に入れるため，grammarsv4 ディレクトリに拡張子”ino”，”pde”のための文法定義を追加した．文法ファイルは C++言語のために予め用意されている CPP14.g4 を流用した．-antlr オプションの正規表現に拡張子 hh にあたるパターンが含まれていないがこれは意図的なものでない．検出試行に用いたソースコード全てに対し拡張子が hh であるようなヘッダファイルは存在しなかったため再度試行しなくて良いと判断した．

検出試行は Example Sketch の絞り込みを行った場合と行わなかった場合の両方行った．

4. 検出の結果

検出の結果 Example Sketch を絞り込んだ場合は 5 ペア，絞り込まなかった場合は 86 ペアのソースコードからコードクローンが検出された．この結果からさらにコードが実装している機能をもとにコードクローンを分類した．それぞれの条件に分けて以下で述べる[4]．

4.1. Example Sketch を絞り込んだ場合

この条件で検出されたクローンはすべて SparkFun APDS9960 RGB and Gesture Sensor ライブラリ[5]バージョン 1.4.2 を使った初期化コードだった．特に 1 つのコードクローンはジェスチャ検出機能を有効化する部分が含まれていた．big_aquatan での該当箇所（リスト 3）を示す．

この部分は GestureTest という Example Sketch のスケッチファイル[6]の 71-88 行目および ColorSensor という Example Sketch のスケッチファイル[7]の 53-66 行目とコードクローンとして検出された．

リスト 3. スケッチの big_aquatan.ino の該当箇所

```
127 Wire1.begin(PIN_SDA1, PIN_SCL1);
128
129 eyes.begin(-5, 0, 0, 0);
130
131 if (apds.init()) {
132   Serial.println(F("LAPDS-9960␣
        initialization␣complete"));
133 } else {
134   Serial.println(F("Something␣went␣wrong␣
        during␣APDS-9960␣init!"));
135 }
136 if (apds.enableGestureSensor(true)) {
137   Serial.println(F("Gesture␣sensor␣is␣now␣
        running"));
138 } else {
139   Serial.println(F("Something␣went␣wrong␣
        during␣gesture␣sensor␣init!"));
140 }
```

SparkFun APDS9960 RGB and Gesture Sensor ライブラリは APDS-9960[8]センサの制御ライブラリである．このライブラリはセンサの初期化とジェスチャ認識などの各種機能の有効化機能を実装している．big_aquatan スケッチは APDS-9960 センサを用いてハンドジェスチャを認識する機能がある． また，該当部分のメッセージ文字列やコードの書式がよく似ており，検出された部分は再利用された部分であると言える．

これらの部分は 5 個のソースコードのペアから見つかり，範囲が重なっているコードクローンを含めて全部で 5 個だった．

4.2. Example Sketch を絞り込まなかった場合

この条件で検出されたクローンは次のように分類できた．

分類 1 SparkFun APDS9960 RGB and Gesture Sensor ライブラリを使った初期化コード

[4] この結果の再現用のデータなどは次の場所にある．: https://github.com/ikubaku/ss2021-scripts

[5] https://github.com/sparkfun/APDS-9960_RGB_and_Gesture_Sensor/

[6] https://github.com/sparkfun/APDS-9960_RGB_and_Gesture_Sensor/blob/V_H1.0_L1.4.2/Libraries/Arduino/examples/GestureTest/GestureTest.ino

[7] https://github.com/sparkfun/APDS-9960_RGB_and_Gesture_Sensor/blob/V_H1.0_L1.4.2/Libraries/Arduino/examples/ColorSensor/ColorSensor.ino

[8] https://www.broadcom.com/products/optical-sensors/integrated-ambient-light-and-proximity-sensors/apds-9960

リスト 4. スケッチの big_aquatan.ino の該当箇所

```
653 DateTime now = rtc.now();
654 char str[20];
655 sprintf(str, "%04d-%02d-%02d␣%02d:%02d:%02d"
        , now.year(), now.month(),
656         now.day(), now.hour(), now.minute(),
                now.second());
657 ts = str;
658 return ts;
```

分類 2 リアルタイムクロックの時刻取得コード

分類 3 SNTP 関連のコード

分類 4 arduino-esp32 の OTA イベント処理コード

分類 5 再利用ではないコード

4.2.1. 分類 1 に属するコードクローン

分類 1 に属するコードクローンの数は 5 つだった．これは Example Sketch を絞り込んだ場合に検出されたものだと考えられる．なぜならこの条件で比較した Example Sketch の中に絞り込んだ場合の Example Sketch が全て含まれている上，検出されたクローンが同じ 5 箇所だったからである．

4.2.2. 分類 2 に属するコードクローン

分類 2 に属するコードクローンは主にリアルタイムクロックから取得した時刻を書式化する処理を実装していた．big_aquatan での該当箇所（リスト 4）は MCP7940 ライブラリの SetAndCalibrate という Example Sketch のスケッチファイル[9]の 173-180 行目とコードクローンとして検出された．

コードクローンを含む Example Sketch に対応するライブラリはすべてリアルタイムクロックの制御ライブラリだった．これらの Example Sketch も多少の改変を除けば現在時刻を取得し文字列に初期化する処理を含んでいた．しかし，big_aquatan では sprintf 関数の行の改行位置が now.day メソッドの呼び出し直前にある．それに対し Example Sketch では now.year や now.hour メソッド呼び出しの直前で改行されていた．また書式文字列は ISO8601 に基づいた日時表現とよく似ているほか，現在時刻のアクセス部分も対応するメソッドを呼び出すだけで実現できるので特殊なものであるとは言えない．したがってこの部分が再利用であるか判断できなかった．

これらの部分は 8 個のソースコードのペアから見つかり，範囲が重なっているコードクローンを含めて全部で 10 個だった．

4.2.3. 分類 3 に属するコードクローン

分類 3 に属するコードクローンはさらに 2 種類に分類できた．1 つは SNTP パケットを作成するコード，もう 1 つは SNTP サーバから受信したデータをもとに 1900 年からの経過秒数を計算するコードだった．big_aquatan での該当箇所（リスト 5，6）はそれぞれ RTCDue ライブラリの RTCDue_NTP_WIFI_with_timezone という Example Sketch のスケッチファイル[10]の 183-203 行目，log4Esp ライブラリの AdvancedDemo という Example Sketch のスケッチファイル[11]の 137-143 行目とコードクローンとして検出された．

SNTP パケットを作成する部分は識別子や周辺のコメントなどが一致しており，再利用されたコードであると言える．

一方で受信したデータを処理するコードはスケッチのものと Example Sketch のものの間で違いが見られた．big_aquatan のコードでは Example Sketch のコードで実装されているタイムゾーンに合わせて秒数を調整するコードが失われている．また packetBuffer[43] の直後の 0 ビットシフトは Example Sketch のコードには存在しなかった．それでもスケッチと Example Sketch のコード片には識別子などの共通点が見られるので，共通の源流となるコードを持つなど何らかの関係があると考えられる．

これらの部分は 55 個のソースコードのペアから見つかり，範囲が重なっているコードクローンを含めて全部で 171 個だった．このことからコードクローンとして検出された部分は様々なライブラリの Example Sketch で再利用されていることがわかる．

[9] https://github.com/Zanduino/MCP7940/blob/v1.2.0/examples/SetAndCalibrate/SetAndCalibrate.ino

[10] https://github.com/MarkusLange/RTCDue/blob/RTCDue_Version_1.1/examples/RTCDue_NTP_WIFI_with_timezone/RTCDue_NTP_WIFI_with_timezone.ino

[11] https://github.com/hunsalz/log4Esp/blob/v1.0.1/examples/AdvancedDemo/AdvancedDemo.ino

リスト 5. スケッチの ntp.cpp の SNTP パケット作成部分

```
10
11 // send an NTP request to the time server at
       the given address
12 void NTP::sendPacket() {
13
14   // set all bytes in the buffer to 0
15   memset(packetBuffer, 0, NTP_PACKET_SIZE);
16   // Initialize values needed to form NTP
       request
17   // (see URL above for details on the
       packets)
18   packetBuffer[0] = 0b11100011; // LI,
       Version, Mode
19   packetBuffer[1] = 0; // Stratum, or type
       of clock
20   packetBuffer[2] = 6; // Polling Interval
21   packetBuffer[3] = 0xEC; // Peer Clock
       Precision
22   // 8 bytes of zero for Root Delay & Root
       Dispersion
23   packetBuffer[12] = 49;
24   packetBuffer[13] = 0x4E;
25   packetBuffer[14] = 49;
26   packetBuffer[15] = 52;
27   // all NTP fields have been given values,
       now
28   // you can send a packet requesting a
       timestamp:
29   udp.beginPacket(timeserver.c_str(), 123);
       //NTP requests are to port 123
30   udp.write(packetBuffer, NTP_PACKET_SIZE);
```

リスト 6. スケッチの ntp.cpp の受信データ処理部分

```
37 unsigned long secsSince1900 = 0;
38 // convert four bytes starting at location
       40 to a long integer
39 secsSince1900 |= (unsigned long)packetBuffer
       [40] << 24;
40 secsSince1900 |= (unsigned long)packetBuffer
       [41] << 16;
41 secsSince1900 |= (unsigned long)packetBuffer
       [42] << 8;
42 secsSince1900 |= (unsigned long)packetBuffer
       [43] << 0;
43 return secsSince1900 - 2208988800UL; //
       seconds since 1970
```

4.2.4. 分類 4 に属するコードクローン

分類 4 に属する検出部分には OTA のエラーイベント処理の実装が含まれていた．big_aquatan での該当箇所（リスト 7）は IRremoteESP8266 ライブラリの IRrecvDumpV3 という Example Sketch の BaseOTA.h[12] の 39-54 行目とコードクローンとして検出された．

リスト 7. スケッチの big_aquatan.ino の該当箇所

```
162 .onProgress([](unsigned int progress,
        unsigned int total) {
163   digitalWrite(2, (progress / (total / 100))
        % 2);
164   Serial.printf("Progress:␣%u%%\r", (
        progress / (total / 100)));
165 })
166 .onError([](ota_error_t error) {
167   Serial.printf("Error[%u]:␣", error);
168   if (error == OTA_AUTH_ERROR)
169     Serial.println("Auth␣Failed");
170   else if (error == OTA_BEGIN_ERROR)
171     Serial.println("Begin␣Failed");
172   else if (error == OTA_CONNECT_ERROR)
173     Serial.println("Connect␣Failed");
174   else if (error == OTA_RECEIVE_ERROR)
175     Serial.println("Receive␣Failed");
176   else if (error == OTA_END_ERROR)
177     Serial.println("End␣Failed");
178 });
179 ArduinoOTA.begin();
180
181 webServer.on("/", handleStatus);
182 webServer.on("/reboot", handleReboot);
```

OTA とはネットワーク上にあるデバイスのソフトウェアのアップデートを可能にする機能である．Arduino の場合は ArduinoOTA ライブラリでその機能を実現している．検出された部分では”ArduinoOTA”という識別子が出現するのでこの機能に関係があると考えられる．しかし，ArduinoOTA ライブラリ[13]の Example Sketch と検出部分を比較したところ，検出された部分は ArduinoOTA には存在しなかった．更に調査したところ検出された部分と似た部分を以下の Example Sketch が見つかった．

- arduino-esp32 の BasicOTA[14]
- ESP8266/Arduino の BasicOTA[15]

これらはそれぞれ ESP32 シリーズと ESP8266 マイクロコントローラ向けの Arduino ソフトウェアの実装に含まれる Example Sketch である．実際，検出されたコー

[12] https://github.com/crankyoldgit/IRremoteESP8266/blob/v2.7.14/examples/IRrecvDumpV3/BaseOTA.h
[13] https://github.com/jandrassy/ArduinoOTA
[14] https://github.com/espressif/arduino-esp32/blob/master/libraries/ArduinoOTA/examples/BasicOTA/BasicOTA.ino
[15] https://github.com/esp8266/Arduino/blob/master/libraries/ArduinoOTA/examples/BasicOTA/BasicOTA.ino

リスト 8. スケッチの actionqueue.h の該当箇所

```
34 public:
35   actionQueue(AquatanEyes *e, AquatanArms *a
         , Camera *h,
36               NeoPixels *c); //, Charger *c);
37   void queueEyes(int l_shape, int r_shape,
         int m);
38   void queueArms(int left, int right, int
         time);
39   void queueArmLeft(int left, int time);
40   void queueArmRight(int right, int time);
41   void queueHead(int pan, int tilt);
42   void queueHeadPan(int pan);
43   void queueHeadTilt(int tilt);
```

ドクローンを含む Example Sketch に対応するライブラリはすべて ESP32 か ESP8266 を用いたシステムを前提としていた．したがってこれらの Example Sketch でもコードが再利用されたと考えられる．

コードクローンを観察するとエラーメッセージやエラーコードを確認する際の比較の順番などが一致しており，検出された部分は再利用されている可能性があると考えられる．特に big_aquatan の該当部分は ESP8266/Arduino の BasicOTA にある部分とよく似ていた．したがって該当部分はその Example Sketch の実装を源流としている可能性がある．

これらの部分は 6 個のソースコードのペアから見つかり，範囲が重なっているコードクローンを含めて全部で 30 個だった．

4.2.5. 分類 5 に属するコードクローン

分類 5 に属するコードクローンは再利用ではないと考えられるもので，さらに 2 種類に分類できた．

- 関数プロトタイプ宣言とクラスメソッド宣言
- 配列への値の保存

検出例をそれぞれリスト 8，9 とリスト 10 に示す．Example Sketch の配列への値の保存部分はすべてスケッチの SNTP パケット作成部分（リスト 5）とペアになっていた．

big_aquatan のソースコードに含まれる関数プロトタイプ宣言やクラスメソッド宣言は Example Sketch の名前

[16] https://github.com/hideakitai/ES920

[17] https://github.com/madhephaestus/Esp32SimplePacketComs

リスト 9. ES920[16]-0.1.9 の openFrameworks/ascii/src/ofApp.h の該当箇所

```
13 void keyReleased(int key);
14 void mouseMoved(int x, int y );
15 void mouseDragged(int x, int y, int button);
16 void mousePressed(int x, int y, int button);
17 void mouseReleased(int x, int y, int button)
       ;
18 void mouseEntered(int x, int y);
19 void mouseExited(int x, int y);
20 void windowResized(int w, int h);
21 void dragEvent(ofDragInfo dragInfo);
22 void gotMessage(ofMessage msg);
```

リスト 10. Esp32SimplePacketComs[17]-0.7.0 の SwarmRobotExample/SwarmRobotExample.ino の該当箇所

```
13 // User data is written into the buffer to
       send it back
14 void event(float * buffer) {
15   int numberOfFloats = 15;
16   buffer[0] = 10; // X Location
17   buffer[1] = 0; // Y Location
18   buffer[2] = 0; // Z Location
19   buffer[3] = 45; // azimuth
20   buffer[4] = 0; // elevation
21   buffer[5] = 0; // tilt
22   buffer[6] = 100; // X size
23   buffer[7] = 100; // Y size
24   buffer[8] = 100; // Z size
25 // Serial.println(
```

にある異なる関数とペアとして検出された．big_aquatan スケッチにあったこれらの関数プロトタイプ宣言やクラスメソッド宣言はすべて独自の機能に関係するものだと考えられるので再利用ではないと言える．

また，ntp.cpp に含まれる SNTP パケット作成処理の部分と Example Sketch で見られる様々な配列への代入部分もコードクローンとして検出されていた．これらは処理の意味が異なっており再利用ではないと考えられる．

これらの部分は 12 個のソースコードのペアから見つかり，範囲が重なっているコードクローンを含めて全部で 39 個だった．

5. 議論

この章では前の章の結果に基づいて議論し，研究設問に回答する．

5.1. 再利用の検出手法

まず研究設問 1 に回答する．

再利用の検出試行では提案手法を用いて再利用と考えられるいくつかのコード片を発見できた．Example Sketch の絞り込みのありなしにかかわらず比較は 3 日以内に完了し，再利用ではない検出結果も少なかった．したがってこの手法は現実的なリソースで Example Sketch からの再利用を探すことに一定の効果があると言える．

一方で提案した Example Sketch の絞り込みが再利用されたコードの発見を妨げていることもわかった．検出できなかったコードは第 4 章の分類 2 にあたるもの，分類 3 にあたるもののうちパケット送信を行うコード，分類 4 にあたるコードの 3 つである．それぞれについて検出できなかった原因として考えられるものを議論する．

まず分類 2 のコードについて考える．Example Sketch を絞り込んだときリアルタイムクロック関係のものとして RTClib の Example Sketch が対象に含まれていた．しかし RTClib の Example Sketch には発見されたような書式化処理はなかった．つまりヘッダファイルの共通性から Example Sketch を絞り込むと似たような機能を持つ異なるライブラリからの再利用を検出できないと考えられる．

次に SNTP パケット送信コードがなぜ検出できなかったについて議論する．このコードは UDP 通信機能を使っているあるいは提供しているライブラリの Example Sketch にあった．例えば log4Esp の AdvancedDemo では WifiUdp.h が include されており，WiFi ライブラリの UDP 通信機能が SNTP 通信のために使われている．一方 big_aquatan ではスケッチファイルでなく ntp.h の中で include されていたため絞り込みの際にそのヘッダを考慮に入れることができていなかった．したがってどのヘッダが include されているのかを絞り込みの情報に使うことはそのままでは難しく，ヘッダファイル内の include プリプロセッサ命令まで考慮する必要があるとわかった．

分類 4 のコードが検出できなかったことには他の要因が関わっている．ライブラリをダウンロードする際は，ライブラリメタデータに登録されているものをダウンロードした．しかし，これらには Arduino コアが提供するライブラリは含まれていない．Arduino コアとは Arduino ボードやマイクロコントローラの種類ごとに実装されている中核となるソフトウェアである．これにはボードに書き込まれるブートローダや入出力など基本的な操作を提供するコード，そしていくつかのライブラリが含まれている．コアに含まれるライブラリは基本的な機能を提供する少数のライブラリとボード特有の機能を提供するライブラリがある．arduino-esp32 の ArduinoOTA ライブラリや WiFi ライブラリもこのライブラリの一種である．これらのライブラリの Example Sketch も考慮に入れたい場合は Arduino コアのライブラリからも Example Sketch を予め集めておく必要があると言える．

5.2. 再利用の特徴

次に研究設問 2 に回答するため，前の章の発見できたコード片の分析をまとめる．

ここでは再利用であるか判断できなかったコードを省いた，分類 1，分類 3 のうちパケット送信を行うもの，分類 4 に当たるコードそれぞれについて特徴をまとめる．

分類 1 や分類 4 のコードはエラーの提示方法を工夫するなどの動機がない限り改変する必要のないコードだった．また，SNTP パケット送信コードについても時刻を取得するだけならば内容を工夫する必要はない．実際 Blynk ライブラリ[18]ではこの部分が BlynkSimpleEthernetSSL.h に取り込まれている．したがって再利用されているコードにはある機能を実装するコードスニペットあるいは一般的なソフトウェア開発でのライブラリとも言える特徴があると考えられる．

しかしこの検出試行で発見できた再利用の種類は 3 種類にとどまった上，対象としたスケッチも 1 つのみであるため一般的な Arduino における開発に一般化できない．再利用の特徴についてはさらなる調査が必要であると言える．

6. 妥当性への脅威

検出試行では計算リソースの都合上ダウンロードした時点で最新のライブラリのみ検出に使用した．したがって，もし古いバージョンにのみ見られるコード片が big_aquatan で再利用されていた場合はその部分を検出できない．このライブラリバージョンの制約は妥当性への脅威となりうる．

また，提案した手法を実現するツールは執筆者が開発し，必要と思われる部分にはテストを行った．しかし，

[18] https://github.com/blynkkk/blynk-library

ツールにまだ未知のバグが含まれている可能性は否定できず，妥当性への脅威となりうる．

7. 結論

本研究では Arduino プロジェクトにおける Example Sketch からのコードの再利用を CCFinderSW への入力を工夫して作成することで分析し，再利用されている部分の特徴を調べた．一方で，再利用部分を見つけただけではそのコードがどこから再利用されたのか明らかにできないという課題が見つかった．

本研究における今後の課題は以下のとおりである．まず，再利用元や再利用の目的を明らかにするため，開発者にインタビュー調査を行う予定がある．また，提案手法において入力データの絞り込み方法に課題があった．今後は提案手法を課題を明確化し手法の改良や結果の応用方法について研究を進めるべきであると考える．

参考文献

[1] M. Sojer and J. Henkel, “Code reuse in open source software development: Quantitative evidence, drivers, and impediments,” Journal of the Association for Information Systems, vol.11, no.12, pp.868–901, 2010.

[2] 肥後芳樹，楠本真二，井上克郎，“コードクローン検出とその関連技術,” 電子情報通信学会論文誌. D，vol.91，no.6，pp.1465–1481，jun 2008.

[3] A.L. Imoize, D. Idowu, and T. Bolaji, “A brief overview of software reuse and metrics in software engineering,” World Scientific News, vol.122, pp.56–70, 2019.

[4] 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.103，no.4，pp.215–227，2020.

[5] C.K. Roy, J.R. Cordy, and R. Koschke, “Comparison and evaluation of code clone detection techniques and tools: A qualitative approach,” Science of computer programming, vol.74, no.7, pp.470–495, 2009.

[6] H. Sajnani, V. Saini, J. Svajlenko, C.K. Roy, and C.V. Lopes, “SourcererCC: Scaling code clone detection to big-code,” In Proc. of ICSE, pp.1157–1168, 2016.

[7] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, “Deckard: Scalable and accurate tree-based detection of code clones,” In Proc. of ICSE, pp.96–105, IEEE, 2007.

子供たちの学びたい欲求を引き出すオンラインツール

根本紀之
アジャイル札幌
nemorine@gmail. com

野尻育代
北海道北見市立上仁頃小学校教諭
iku719iku@gmail. com

1. 背景と課題

Covid-19 の広がりに伴い，ソフトウェア業界では Zoom や Teams や Discord など急速にオンラインツールが普及した．そのうちの一つにオンラインホワイトボードの miro がある．これは複数人が同時に作業可能なアジャイル開発向けのオンラインのツールである．

一方，Covid-19 は北海道の地方都市での教育にも影響を及ぼしており，北見の小学校では下記の問題が発生し，解決策を模索していた．

・突然臨時休校になり，授業が進まない
・登校可能になった後も，子供たちの安全な距離を保ちながら学ぶ環境を整えなくてはならない

2. 解決方法

前述の問題を解決するため，小学校５年生と６年生の 7 名の複式クラスを対象に校内での授業（オフラインでの授業）の中で miro を導入した．miro を使うことでオフラインでのグループワークでも安全な距離を保つことができる．

最初の授業としては，人物や出来事が関連線でつなぐことができる日本の歴史を選んだ．ある要素同士を関連線でつなぐことは，オンラインホワイトボードツールの得意とするところである．

2.1. 社会の授業における歴史絵巻の作成

社会の授業では時代ごとに人物，出来事を整理し，miro 上に分担しながら図 1 のような絵巻を作った．また miro 上の成果物を担当者が発表することで知見を共有した．

この歴史絵巻の作成をきっかけに子供たちは歴史に興味を持ち始め，家庭学習では歴史をテーマに選ぶ子供たちが増えた．

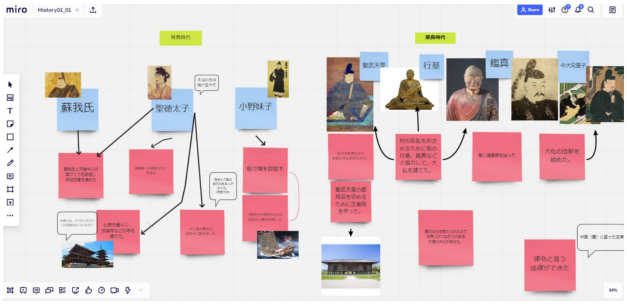


図 1 歴史絵巻

3. 考察

3.1. 考察 1 – オンラインツールと模造紙の違い

miro の場合は模造紙と異なり，タイプすることで綺麗な文字を書くことができ，図形，線，文章など何度でもやり直すことができる．そのため子供たちは新しい知識の取り入れや新しいデザインに挑戦することができた．

従来の模造紙での作業を分析したところ，下記の理由により創造的な協働作業ができていなかった．

・模造紙のサイズが決まっている
・字を書くのが苦手な子供は綺麗な下書きの字をなぞる係になる
・絵や写真を貼るときは印刷などのひと手間がかかる
・人数が多い場合は見る方向がバラバラになる
・作業の待ち時間が多い
・やり直しがきかない

一方オンラインツールである miro の場合は創造的な協働作業が可能となる．

・ホワイトボードのサイズは無限大である
・字はタイプなので人に依存しない
・絵や写真をインターネット経由ですぐに取り込める
・みんな同じ画面（同じ方向）を見ることが可能である
・作業待ちは発生しない

・修正が容易である

3.2. 考察 2 – 学びの力

子供たちはお互いに褒めあうことで承認欲求を満たす. 子供たちは他の子の良いところをすぐに取り入れる柔軟性も持ち, 成果物の完成度を高めていく. 完成度の高い成果物はまた褒められるためポジティブループが回っていく. また失敗してもすぐに修正が可能なため, 挑戦の回数が従来の模造紙を使った協働学習より増えることとなる.

このポジティブループによって子供たちが自ら歴史絵巻を作りたいと考えるようになり, 事実として子供たちは休み時間や放課後に自主的に作成をしている. すなわち子供たちが「勉強しなきゃいけない」という態度ではなく自ら「作りたい」と思っていることが分かる.

4. 結論

オンラインホワイトボードの miro は, 従来の模造紙に比べ, 子供たちが創造性を発揮しやすいことが分かった. 大きなポイントはその修正容易性であり, その性質により挑戦の回数が増え, 創造的な作業に挑戦することができる. また誰かが考えたアイディアは子供たちの間で即座に採用され, より高度な成果物を作ることができる. Miro を使った創造的な作業によって子供たちの学びたいという欲求を引き出すことができた.

子供たちの関係性もあるため, 全ての学級に適用できるわけではないが, 学びを深めるツールとして miro は有用であると言える.

高専専攻科におけるテンソルデータ処理プログラミング教育の実践

山本 直樹　　石田 明男　　村上 純
熊本高等専門学校・電子情報システム工学系　熊本高等専門学校・リベラルアーツ系　熊本高等専門学校・電子情報システム工学系
naoki@kumamoto-nct.ac.jp　ishida@kumamoto-nct.ac.jp　jun@kumamoto-nct.ac.jp

要旨

本報告では，熊本高専専攻科の授業科目の中で実施されているテンソルデータ処理のプログラミング教育の実践事例について述べる．

1. はじめに

テンソルはデータ処理では多次元配列として利用される重要なデータ構造の1つであり，テンソル分解の形で，信号処理，数値線形代数，コンピュータビジョン，数値解析，データマイニング，グラフ解析，神経科学など幅広い分野で応用されている[1]．これまで我々は，テンソルデータ処理の理解支援のために立体パズルを用いたツールを開発してきている[2]．最近では，テンソルデータ処理を学習するための自学自習用教材も開発している[3]．本報告では，これまで開発されたツールおよび教材の利用も含めた，熊本高専専攻科の授業科目で実施されているテンソルデータ処理に関するプログラミング教育の実践事例について述べる．

2. テンソルデータ処理を実施する科目の概要

テンソルデータ処理を実施した科目は，本校専攻科電子情報システム工学専攻1年の後期選択科目の『データサイエンス』である．本科目では，最初にプログラミング言語の使い方に慣れるために行列演算などの基礎を行い，次に，機械学習の諸手法の学習が行われる．テンソルデータ処理は，本科目の最後の方の項目として実施された．この科目で使用するプログラミング言語は R であり，授業の進め方は，授業内容の説明後に R を用いて上記の内容の例題，ケーススタディに取り組む流れとなる．

3. テンソルデータ処理教育の実践事例

テンソルデータ処理の授業は，「高階テンソル」と「テンソル分解」の 2 回実施された．まず，1 回目の「高階テンソル」の授業では，その概要や定義，テンソル用 R パッケージなどの説明が行われた後，学生達は行列展開，モード積などの例題に取り組んだ．さらに，行列展開，モード積などの理解を深めるために，学生達は，我々が独自に開発したマクマホン立方体や立体パズルを利用した自学自習教材に取り組んだ．次に，2 回目の「テンソル分解」の授業では，その概要，その分解の 1 つである高次特異値分解（HOSVD）[4]の定義やアルゴリズム，カラー画像およびカラー動画像の HOSVD などについて説明が行われた後，学生達は基礎的な数値データの分解，応用的な画像・動画像データの分解の例題に取り組んだ．

4. 授業の評価とまとめ

本稿では，テンソルデータ処理教育の実践事例について報告した．この教育では，教材や例題にマクマホン立方体，立体パズル，カラー画像・動画像を用いており，視覚的にも処理結果を確認しやすいという特徴がある．さらに，本授業後に行列展開，モード積，畳み込みに関する理解度の確認を筆記テストにより行い，全問正解の学生が約 7 割以上あることから概ね理解できているものと考えられる．また，HOSVD アルゴリズムの実装状況についてレポートにより確認したところ，約 8 割以上の学生が実装できていることが分かった．

参考文献

[1] T. Kolda, B. Bader, “Tensor Decompositions and Applications,” *SIAM Review*, Vol. 51, No. 3, pp. 455–500, 2009.

[2] 山本直樹，村上純，石田明男，“テンソル分解プログラミングの理解支援のための立体パズルの利用,” *SS2019 論文集*, pp.114-123, 2019.

[3] 山本直樹，石田明男，大石信弘，村上純，“マクマホン立方体を用いたテンソルデータ処理に関するデータサイエンス教育の試み,” *熊本高専研究紀要*, Vol. 12, pp.45-48, 2021.

[4] L. D. Lathauwer, B. D. Moor, J. Vandewalle , “A Multilinear Singular Value Decomposition,” *SIAM J. Mat. Anal. Appl.*, Vol.21, No.4, pp.1253-1278, 2000.

WHY に重点を置いたプロセストレーニングの改善

片山 泰司
オムロン ソーシアルソリューションズ株式会社
yasuji.katayama@omron.com

要旨

品質をより良くするには，品質を良くしたいと考えて，実際に行動までする人財を増やす必要がある．プロセスとは何か，何のために必要なのか，を現場に納得してもらい，プロセス定着による品質確保につなげることを狙って活動に取り組んだ．WHY（なぜそれをやるのか）に重点を置いた，プロセストレーニングの改善を実施し，プロセスの目的や意義を伝えることで、プロセスに対する現場の納得感や関心を高めた．本発表ではプロセストレーニングの改善事例について紹介する．

1. 背景

本発表の対象となる活動に取り組んだ主体は，鉄道事業の組み込みソフト設計部門である．改善したいことは「プロセスの必要性を理解・納得していない」→「プロセス不遵守増加」→「ソフト不具合増加」のサイクル．この負のサイクルの歯止めのために，まずはプロセスに対する現場の納得感を高めることに取り組んだ．

2. 活動内容

取り組みは何事も，現場が理解・納得しないと，長続きしないと考えている．プロジェクトを進めるために必要な「当たり前のこと」＝「プロセス」の定着につなげるため，WHYに重点を置いたプロセストレーニングの改善を実施した．新人からベテランまで，プロセスの目的・必要性に納得してもらえるよう，トレーニングの工夫に取り組んだ。トレーニングの対象プロセスは，エンジニアリング領域とプロジェクト管理領域．まずエンジニアリング領域のトレーニング改善を実施し，現場の納得につながったため，プロジェクト管理領域にも展開した．従来のトレーニングでは手薄だった，プロセスとは何か，何のために必要なのか，の目的や意義の話を充実させたり，広く知られた言葉の意味を問いかけたり，プロセスの何が嬉しいのかの話，やりがいを感じてもらうための話，の説明を強化した．トレーニングアンケート結果はベテランから「再認識」，若手から「有意義」のコメントが多く得られた．新人からベテランまで，従来トレーニングよりも，現場の納得感や関心，プロセス遵守，凡事徹底，の意識につなげることができたと考えている．

参考文献

[1] WHY に重点を置いた人材育成 SPI Japan 2017 http://www.jaspic.org/event/2017/SPIJapan/session2B/2B1_ID001.pdf
[2] OMRON BASICS オムロン株式会社 グローバル人財総務本部 2017
[3] 芝本秀徳の『プロジェクトマネジメントの守破離』プロセスって何それ？食べれるの？ 2012 http://blogs.itmedia.co.jp/hideshibamoto/2012/07/post-8b1d.html
[4] PHP 文庫『子どもが育つ魔法の言葉』ドロシー・ロー・ノルト著 レイチャル・ハリス著 石井千春訳 1999

アリコロニー最適化法を用いたモデル検査の効率化のための探索戦略

熊澤 努
株式会社 SRA
kumazawa@sra.co.jp

滝本 宗宏
東京理科大学
mune@rs.tus.ac.jp

神林 靖
日本工業大学
yasushi@nit.ac.jp

要旨

モデル検査は，ソフトウェアシステムの挙動を記述したモデルが，望ましい仕様を満たすかどうかを検査する形式的検証技術である．モデルや仕様には，有向グラフを用いた状態遷移モデルによる表現を採用することが多い．モデル検査は，モデルと仕様によって構成される状態空間を探索して仕様違反を発見し，反例として返す．モデル検査では，状態の指数関数的な増加のために検査が困難になる状態爆発問題が知られているが，加えて，反例の可読性を高めることも重要である．これらの問題を解決するために，群知能の一種であるアリコロニー最適化法を用いた検査技法が提案されている．これは，状態空間を非網羅的に探索することで，状態爆発問題を軽減し，同時に可読性の高い短い反例を求める方法である．本論文では，アリコロニー最適化法によるモデル検査技術の更なる効率化のための探索戦略を提案する．提案する戦略は，アリコロニー最適化法を用いた従来の技法よりも広範囲に渡る探索を実現し，大規模なモデルに対しても反例の発見を速めて状態爆発を低減する．従来法との性能評価実験を行い，提案する探索戦略が実行時間の高速化と消費メモリの低減を向上させ，効率的な探索を実現することを確認した．

1. はじめに

モデル検査は，ソフトウェアのモデルが所望の仕様を満たすかどうかを自動検証する技術である [1, 2]．モデルはソフトウェアの振る舞いを記述したもので，オートマトンに代表される状態遷移モデルが採用されることが多い．モデル検査は，状態遷移モデルと仕様が表す状態空間上を探索して，システムの挙動に関する仕様違反を発見する．発見した仕様違反は反例として出力される．従来のモデル検査では，反例の検出に，主に深さ優先探索法に基づく網羅探索技術が用いられてきた．

本論文では，モデル検査における次の 2 点の課題に取り組む．

- 状態爆発問題：実用的なソフトウェアシステムは状態数が多く，大規模であることが多い．それゆえ，モデル検査が扱う状態空間もまた状態数が非常に大きくなるので，大規模な空間上を効率的に探索するアルゴリズムが必要である．特に，複数のプロセスから成る並行システムを扱う場合は，取りうる状態数がプロセス数に対して指数関数的に増加し，従来の網羅探索技術で効率的に検査することが難しい．
- 反例短縮問題：反例は，ユーザがシステムの不具合を特定するための手掛かりとなる診断情報で，状態遷移モデル上の路（状態列）で与えられることが多い．長大な反例は可読性が悪く，その解析に時間を要する恐れがある．そのため，短い反例を出力することが望ましい．

これらの 2 つの問題を解決するために，群知能あるいはメタヒューリスティクスを用いた検証技術が研究されている [3]．群知能の一種であるアリコロニー最適化法 [4] を用いた探索アルゴリズム ACOhg [5] は，大規模グラフを確率的に探索して最短路を効率的に発見するアルゴリズムである．ACOhg は，ソフトウェア検証，特にモデル検査への応用が試みられてきた [5, 6, 7, 8, 9, 10]．文献 [8] では，ACOhg を 2 段階に分けて実行することで，グラフ上で閉路を構成する反例を発見するアルゴリズム ACOhg-live が提案された．ACOhg-live は，従来の網羅探索に基づく検証技法よりも，状態爆発問題と反例短縮問題に有効であることが報告されている．

本論文では，ACOhg-live の性能を向上させ，更なる効率化を実現する新たな探索戦略を 2 つ提案する．

- スキップ戦略：ACOhg-live は，探索の進行に応じて探索範囲を拡大していく方法を採る．しかし，モデル検査が扱う状態空間によっては，探索が効率的に進まず，その範囲を十分に拡大できない恐れがある．その場合に，各探索の終点となる状態で探索を停止せずに継続することで，探索範囲を拡大させる．
- 差替え戦略：ACOhg-live は，有向グラフ上の路を繰り返し探索して，反例の見込みのある路を絞り込んでいく．しかしながら，反例を発見するまで十分に探索が進んでいない段階で絞り込みが行われると，以降の探索の進行が遅れる恐れがある．そこで，探索が進まなくなった時点で，絞り込みから除外された路へと探索を切り替えることによって，以降の探索を進めやすくする．

2 種類の探索戦略を組込んだ試作プログラムを実装して，従来の ACOhg-live との性能評価実験を行った．その結果，提案する探索戦略を組合せることで，実行時間とメモリ消費量が低減し，効率的に探索が進むことを確認した．

本論文の構成は次の通りである．2 節で，技術的な背景として，モデル検査と ACOhg，ACOhg-live の概要を説明する．3 節で，提案する 2 つの戦略の詳細を説明する．4 節では，提案手法を実装したプロトタイプの概要と，プロトタイプを用いて行った実験の結果を議論する．5 節では，関連する先行研究について述べ，6 節で結論と今後の課題を述べる．

2. 背景

本節では，モデル検査の概略を述べる．さらに，本論文で着目するモデル検査技法 ACOhg 並びに ACOhg-live を，文献 [8] に基づいて説明する．

2.1 オートマトン理論を用いたモデル検査

モデル検査は，ソフトウェアの振る舞いが与えられた仕様を満たすかどうかを形式的に検証する技術である．モデル検査は，オートマトン理論を用いたモデル検査 [11, 2] と記号的モデル検査 [2] とに大きく分けることができる．オートマトン理論を用いたモデル検査は，状態遷移モデルと仕様を有限オートマトンで表し，その受理可能性を判定する検査法である．有限オートマトンは有向グラフとみなせるので，判定にはグラフ探索法を使用する．一方，記号的モデル検査は，モデルの振る舞いを論理式で表して検証する技法である．本論文は，オートマトン理論を用いたモデル検査を対象とする．

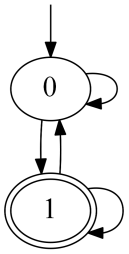


図 1: Büchi オートマトンの例

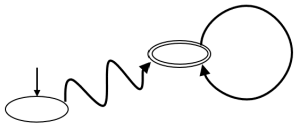


図 2: 閉路型反例の形状

オートマトン理論を用いたモデル検査では，無限語を受理する Büchi オートマトンを用いて検証を行う．有限語を受理する古典的な有限オートマトン（NFA）と同様に，Büchi オートマトンは初期状態と受理状態を持つ有向グラフで表現できる．ただし，NFA が受理状態に到達した語を受理するのに対して，Büchi オートマトンは受理状態を無限回通過する語を受理する．図 1 に Büchi オートマトンの例を示す．接続元となる状態のない矢印を付した状態 0 が初期状態であり，2 重丸の状態 1 が受理状態である．受理状態 1 自身が閉路を構成しているので，状態 0 を出発して状態 1 を無限回通る語を受理するが，状態 0 だけを無限回通る語は受理しない．このように，Büchi オートマトンの受理語は，受理状態を含む閉路とその閉路に達する路から成る（図 2）．

モデル検査の手順は次の通りである [2]．まず，仕様違反となるモデル上の路だけを受理する Büchi オートマトンを構成する．次に，構成したオートマトンが受理する語が存在するかどうかを，グラフ探索を網羅的に実行して調べる．そのために，有向グラフの強連結成分検出アルゴリズム [12] などを使用して，受理状態を含む閉路を検出する．検出した受理語は仕様に違反しているので，受理語が通過した路を反例として出力する．

2.2 ACOhg

ACOhg [5] は，大規模グラフを対象として最短路を確率的に探索するアルゴリズムである．ACOhg は，アリコロニー最適化法 (ACO) [4] に基づいている．ACO は，ユーザが事前に定めた始点（巣）から出発した多数の探索エージェント（アリ）が終点（エサ）を目指して探索空間上を探索する．エージェント同士は，路に配置したフェロモンによって相互に対話する．探索に際して，各エージェントは確率的に遷移を選択するが，先行するエージェントが配置した量の高い路ほど高確率で選択する．すなわち，フェロモンは選択する路を絞り込むために使用される．局所最適に陥ることを防ぐために，フェロモンは自然に低減（蒸発）するものとする．この手続きを繰り返し，最短路を発見する．

従来の ACO は，路の長さに関する制約がないので，エージェントは状態空間上を長時間遷移し続け，状態爆発を引き起こす恐れがある．そこで，ACOhg は，各エージェントが探索する路の長さに上限を設けることで，実行時間の超過とメモリ消費の増大を抑制して，大規模なグラフの探索を実現する．ただし，1 回の探索で終点に到達するとは限らないので，指定された回数探索を行う毎に，先行したエージェントが最後に到達した状態から後続のエージェントが遷移を開始するように始点を変更して，探索範囲を広げていく．

Algorithm 1 ACOhg [5]

1: フェロモンを初期化する;
2: **while** 探索回数が $mstep$s 回以下である **do**
3: **for all** $a \in A_{ant}$ **do**
4: a は路の長さが λ_{ant} になるか，終点に到達するまで探索する;
5: 上位の路を選択する;
6: **end for**
7: フェロモンを更新する;
8: σ_s 回に 1 回探索範囲を拡大する;
9: **end while**

アルゴリズム 1 に ACOhg の概要を示す．アルゴリズム 1 の 1 行目では，エージェント同士の相互対話に利用するフェロモンをランダムな値で初期化する．2 行目以降でエージェントによる探索を $mstep$ 回繰り返す．3 行目から 6 行目が個数 $colsize$ のエージェントの集合 A_{ant} による探索である．ここで，$mstep$ と $colsize$ はユーザが定めるパラメータである．4 行目では，エージェントは状態遷移を繰り返して探索する．フェロモンが多く配置された状態ほど遷移しやすくなるように，現在の状態からの遷移先を確率的に決定する．状態 i から状態 j へのエージェント k の遷移確率 p_{ij}^k は次式で計算する．

$$p_{ij}^k = \frac{[\tau_j]^\alpha [\eta_{ij}]^\beta}{\Sigma_{l \in N_i} [\tau_l]^\alpha [\eta_{il}]^\beta}, \quad \text{ただし，} j \in N_i. \tag{1}$$

ここで，τ_j は状態 j のフェロモン量，η_{ij} は適当なヒューリスティクス，N_i は i の後続状態の集合である．フェロモンとヒューリスティクスの寄与度をパラメータ α, β で定める．遷移の後には，現在の状態のフェロモン量を一定の割合 ξ だけ減少させる．つまり，状態 j のフェロモン量を $\tau_j = (1-\xi)\tau_j$ とする．他のエージェントが同じ路を選択する確率を低減して，局所最適に陥るのを防ぐことが目的である．

5 行目では，直前の探索と今回の探索でエージェントが通過した路の集合から上位 ι 個の路を選出することで，今後より深く探索するための始点となる状態を求める．ι はパラメータである．選出には，エージェントが通過した路の質を評価する目的関数を使用する．反例短縮問題の解決のために，短い路ほど良好な値をとるように，エージェント k の路 π^k に対する目的関数 f を次式で定める．

$$f(\pi^k) = \begin{cases} |\pi_{full}^k|, \quad (\pi_{-1}^k \text{が終点の場合}) \\ |\pi_{full}^k| + h(\pi_{-1}^k) + p_p + p_c \dfrac{\lambda_{ant} - |\pi^k|}{\lambda_{ant} - 1} \\ \qquad (\pi_{-1}^k \text{が終点ではない場合}). \end{cases} \tag{2}$$

ここで，π_{-1}^k は π^k の最後の状態，π_{full}^k は ACOhg の実行時点での始点から π_{-1}^k までの路，h は状態 π_{-1}^k に対するヒューリスティック関数である．パラメータ p_p は，終点に到達しなかった路の質を低く判定する懲罰の尺度，p_c は π^k に閉路が含まれる場合の懲罰の尺度である．

7 行目のフェロモンの更新処理では，各状態 i に配置済みのフェロモンを一定の割合 ρ だけ低減させ，$\tau_i = (1-\rho)\tau_i$ とする．エージェントの通る路の偏りを是正することが目的である．最良の路については，後続のエージェントを集中させて探索を深く進めるために，目的関数 f の値を使って，フェロモン量を高めるように更新する．8 行目では探索範囲を拡大する．終点に未到達のエージェントが最後に到達した状態に始点を変更し，フェロモン量を初期化する．

2.3 ACOhg-live

ACOhg-live [8] は，図 2 に示す閉路を成す反例を探索するモデル検査アルゴリズムである．ACOhg-live は ACOhg を 2 段階に分けて実行する．

アルゴリズム 2 に ACOhg-live の概要を示す．第 1 段の ACOhg はオートマトンの受理状態を探索する．次に，発見した各受理状態を始点として，その受理状態自身を探索する第 2 段の ACOhg を実行する．受理状態に到達した場合には探索を終了し，そうでない場合には，第 1 段に戻って探索を継続する．このとき，ACOhg-live は保持しているタブーリストに受理状態を格納する．1 つの受理状態に対して第 2 段を複数回実行することがないように，格納された受理状態を以降の探索で無視する．

Algorithm 2 ACOhg-live [8]

```
1: repeat
2:    ACOhg によって受理状態を探索する; { 第 1 段 }
3:    for all 受理状態 do
4:       ACOhg によって閉路を探索する;{ 第 2 段 }
5:    end for
6: until 第 1 段で受理状態を発見しない
```

図 3 は ACOhg-live の実行例である．図 3a は，第 1 段の ACOhg を実行した結果, 初期状態 0 を始点として受理状態 1 と 2 を発見した状況を示している．図では，エージェントが探索した受理状態までの路を灰色の状態で表す．図 3b と図 3c が第 2 段の ACOhg の実行の様子である．まず，受理状態 1 から ACOhg を実行するが，状態 1 自身に戻る路が存在せず，探索に失敗する（図 3b）．続けて，受理状態 2 を新たな始点として ACOhg を実行して，状態 2 を含む閉路の探索に成功した後，ACOhg-live の実行を終える（図 3c）．

3. 提案する探索戦略

本節では，ACOhg-live の性能向上のために，エージェントによる新たな探索戦略を提案する．

3.1 スキップ戦略

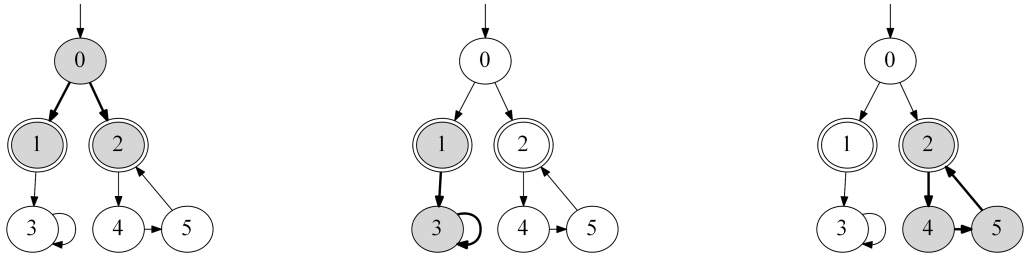
ACOhg-live の第 1 段の ACOhg は，受理状態に到達する最短路を計算する．一般に，Büchi オートマトンは複数の受理状態を含むが，図 4 の Büchi オートマトンのように，1 つの路に複数の受理状態が存在する場合には，第 1 段の ACOhg でのエージェントによる探索が十分に深く進まず，ACOhg-live の性能に大きな影響を与える可能性がある．図 4 の初期状態 0 に対して第 1 段の探索を実行すると，アルゴリズム 1 の 4 行目の探索終了条件に従って，受理状態 1 を発見して終了する．状態 1 を含む閉路は存在しないので，続く第 2 段の探索に失敗する．したがって，閉路を構成する受理状態 2 を発見するためには，状態 0 から第 1 段の探索を再度実行する必要がある．1 回目の第 1 段の探索を深く進めて状態 2 を発見することができれば，第 1 段を繰り返し実行する必要がなくなり，探索効率の向上が期待できる．

そこで, ACOhg における新たな探索戦略であるスキップ戦略を提案する．スキップ戦略は，各エージェントが路の長さが上限値 λ_{ant} に達するまで遷移を続け，探索中には終点に到達したかどうかの判定を行わない戦略である．エージェントが探索を終えた後に，探索した路が終点を通過したかを判定する．図 4 にこの考え方を適用すると，$\lambda_{ant} > 2$ ならば第 1 段の探索を 1 回実行するだけで受理状態 2 を発見できる．

Algorithm 3 スキップ戦略

```
1: フェロモンを初期化する;
2: while 探索回数が msteps 回以下である ∧ 終点に達していない do
3:    for all a ∈ A_ant do
4:       a は路の長さが λ_ant になるまで探索する;
5:       a が通過した路から，終点に到達する部分路をランダムに選択する;
6:       上位の路を選択する;
7:    end for
8:    フェロモンを更新する;
9:    σ_s 回に 1 回探索範囲を拡大する;
10: end while
```

アルゴリズム 3 に, ACOhg にスキップ戦略を組込んだアルゴリズムを示す．4 行目では，エージェントは λ_{ant} 回遷移を繰り返して探索を行う．ACOhg とは異なり, 終点である受理状態に到達してもエージェントは遷移を継続する．5 行目では，エージェントが通った路から，受理状態に到達した部分路を 1 つランダムに選択する．確率的に部分路を選択することで，エージェントの数 $colsize$ が十分大きければ，通過した受理状態を網羅的に抽出す



(a) 第 1 段: 状態 0 から受理状態の探索　**(b)** 第 2 段: 状態 1 を含む閉路の探索　**(c)** 第 2 段: 状態 2 を含む閉路の探索

図 3: ACOhg-live 実行例

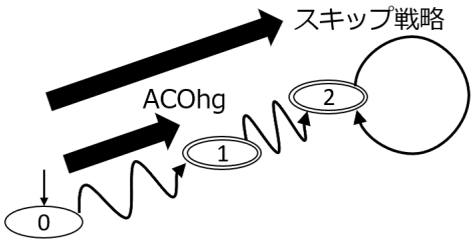


図 4: ACOhg とスキップ戦略の比較

ることが期待される．少なくとも 1 個体のエージェントが受理状態に到達した場合に，アルゴリズムは終了する．この終了条件は文献 [7] で採用している終了条件と同一であり，アルゴリズムの効率化のために必要である．5 行目で部分路を選択できない場合には，エージェントの通った路に終点が存在しないことを意味するので，従来の ACOhg と同様に，6 行目で質の高い路を選んでアルゴリズムを継続する．

3.2 差替え戦略

ACOhg において，先行エージェントが終点に達しなかった場合，後続エージェントは，より探索を深めて路を伸展させていく必要がある．しかしながら，先行エージェントが発見した路の目的関数値が好ましい場合，アルゴリズム 1 の 5 行目の選出手続きで後続エージェントが発見した路が選出されず，繰り返し探索を行ったとしても，既知の路の再選出を繰り返す恐れがある．この場合，路が終点に達しないにも関わらず，エージェントが状態空間の狭い領域に集中する．その結果，局所最適に陥ってしまい，新たな路を発見しながら効率的に探索を深めることが困難になる，

そこで，局所最適を回避して ACOhg の効率化を図る戦略として，差替え戦略を導入する．差替え戦略は，選出した路に変化がない場合に、目的関数値が好ましくない路を確率的に選択する戦略である．アルゴリズム 1 の 5 行目で選出した上位の路の集合が，選出前の集合と同一の場合には，4 行目で求めた路に差替えて路の選出をやり直す．路の差替えは次の手順で行う．4 行目から 6 行目のエージェントによる探索の終了後に，5 行目で選出したそれぞれの路について，確率 0.5 で刈り取るかどうか決定する．次に，4 行目で発見した路から上位の路を，刈り取らずに残した路に追加する．このとき，追加する路の数を刈り取った路の数と同数にし，路の数を最大 ι 個とすることで，ACOhg の 7 行目以降の手続きを変更なく実行することができる．

4. 評価実験

ACOhg-live について，提案する探索戦略を組込んだ方法と従来法とで性能評価実験を行った．本節ではその結果を報告する．

4.1 実験の概要

ACOhg-live と提案する探索戦略を実行する比較実験用プロトタイプを，Python 言語で実装した．使用した Python のバージョンは 3.6.8 である．

プロトタイプへの入力は，システムを記述した状態遷移モデル，検査する仕様を記述した Büchi オートマトン，ACOhg のパラメータの値などを記した設定情報である．提案する探索戦略を使用するかどうかも設定情報に含める．モデルと仕様はラベル付き遷移系 [13] で

与え，その記述フォーマットは Aldebaran フォーマット [14] とする．ラベル付き遷移系は状態遷移モデルの一種であり，イベント列で表されるシステムの挙動を記述するのに適している．入力はユーザが JSON 形式で記述する．プロトタイプは，モデルと仕様を読み込んだ後に，ACOhg-live を実行するために必要なパラメータと使用する探索戦略を設定する．続いて，プロトタイプは ACOhg-live を実行して反例を探索する．効率化のために，状態を探索中に必要に応じて生成する On-the-fly 検査方式 [2] を実装した．プロトタイプは，仕様違反を検出した場合には反例をイベント列で，検出しなかった場合には Python の組込み定数 None を出力する．

評価実験に使用するモデルと仕様には，分散データベースの事例 [13] を用いた．実験のためのモデルと仕様は文献 [13] のモデル記述を参考に作成した．分散データベースは，通信路を通してリング状に接続した複数のノードから成るシステムである．各ノードは，隣接するノードからの通信に応じて，自身の情報を更新する．そして，その情報をもう 1 つの隣接ノードに伝えることで，システムが保持する情報の整合性を保つ．全ノードが内部の情報の更新を終えることを要請する仕様をはじめとする 3 件の仕様 A，B，C を実験で扱うこととした．

ACOhg-live と提案する探索戦略の評価実験では，次の探索条件を用いた．

- OR：提案する探索戦略を使用せずに，ACOhg-live を実行する．
- ORT：スキップ戦略の終了条件だけを第 1 段の ACOhg に適用する．これは，文献 [8] の ACOhg とスキップ戦略とで，アルゴリズムの終了条件が異なることを考慮した探索条件である．
- SK：第 1 段にスキップ戦略を適用する．
- RE: 第 2 段に差替え戦略を適用する．
- RET：スキップ戦略の終了条件だけを第 1 段に適用し，第 2 段には差替え戦略を適用する．
- HY：第 1 段にスキップ戦略を，第 2 段に差替え戦略を適用する．

それぞれについてプロトタイプを 30 回実行し，ACOhg-live が出力した反例の長さ，実行に要した時間，消費したメモリを測定した．評価実験に使用するパラメータ設定は，先行研究である文献 [8] を元に決定した．パラメータの値を表 1 に示す． 2.2 節で述べたように，ACOhg は問題領域の経験的知識をヒューリスティクスとして活用する．本実験では，状態空間上での受理状態の位置を特定できないので，第 1 段ではヒューリスティクスを使用せずに定数値 1 とした．第 2 段では，状態をバイナリ表現に変換したときの現在の状態と探索の終点である受理状態の間のハミング距離を，終点までの距離を見積もったヒューリスティクスとした [8]．実験には HP ProDesk 600 G4 SFF を使用した．これは，OS に Windows 10 Pro を搭載し，Intel Core i7-8700, CPU 3.20GHz, RAM 16.0 GB を備えたマシンである．実行性能に影響を与える可能性を考慮して，Python に実装されているガベージ・コレクション機能を無効にして測定を行った．

4.2 実験の結果と考察

分散データベースの各仕様での測定結果を，仕様 A については表 2 と図 5 に，仕様 B については表 3 と図 6 に，そして，仕様 C については表 4 と図 7 に示す．

まず，スキップ戦略（SK）の結果を論じる．分散データベースのいずれの仕様の場合も，従来の ACOg-live（OR）より実行時間とメモリ消費量が減少し，性能の改善が見られた．終了条件をスキップ戦略と同一にした場合の結果（ORT，RET）と比較して，実行時間とメモリ消費量に関してはスキップ戦略の効率が高く，本戦略が探索性能の向上に有効であることが確認された．一方で，反例長については，スキップ戦略を使用すると長くなる傾向が現れた．スキップ戦略は探索を深める戦略なので，従来の ACOhg-live ほどには反例短縮の効果が得られなかったと考えられる．

次に，差替え戦略（RE）の結果を論じる．差替え戦略を適用すると，いずれの仕様の場合でも，従来の ACOhg-live（OR）と比べて反例長，実行時間，メモリ消費量に大きな違いは見られなかった．差替え戦略を単独で使用した場合には，性能の向上が得られるとは言えないものと考えられる．

最後に，スキップ戦略と差替え戦略を共に用いた結果（HY）を論じる．分散データベースの仕様 A を用いた実験では，実行時間の平均がすべての実験条件の中で最小となり、中央値はスキップ戦略に次いで 2 番目に小さい値となった（表 2）．仕様 B の実験の結果（表 3）によると，実行時間並びにメモリ消費量の平均，標準偏差，

表 1: 実験で使用したパラメータ設定

パラメータ	説明	第 1 段の ACOhg	第 2 段の ACOhg
$msteps$	探索回数	10	10
$colsize$	1 回の探索におけるエージェントの数	10	10
λ_{ant}	1 回の探索での路の長さの上限	20	10
σ_s	探索範囲拡大までの回数	2	2
ι	始点の保持数	10	10
ξ	探索中のフェロモンの蒸発率	0.7	0.5
a	フェロモンの下限に対する上限の比	5	5
ρ	各探索後のフェロモンの蒸発率	0.2	0.2
α	遷移時のフェロモンの寄与度	1.0	1.0
β	遷移時のヒューリスティクスの寄与度	2.0	2.0
p_p	終点に到達しなかった場合の懲罰度	1000	1000
p_c	閉路に対する懲罰度	1000	1000

表 2: 仕様 A を用いた実験結果

		OR	ORT	SK	RE	RET	HY
反例長	平均	15.23	21.67	17.50	**15.17**	17.87	16.90
	標準偏差	3.48	10.44	7.42	**3.33**	7.51	6.33
	中央値	**15**	16	16	**15**	**15**	16
実行時間 [sec]	平均	3.45	1.87	1.41	3.40	1.94	**1.40**
	標準偏差	0.70	0.24	0.39	0.68	**0.20**	0.47
	中央値	3.49	1.81	**1.25**	3.29	2.02	1.26
メモリ消費量 [KB]	平均	267.35	184.18	**154.59**	266.97	176.98	158.70
	標準偏差	35.38	15.72	23.89	41.48	**11.96**	19.72
	中央値	262.51	187.37	**148.71**	267.49	174.33	156.24

中央値が最小となった．仕様 C での実験結果（表 4）においても，実行時間とメモリ消費量のどちらの結果も最小となり，実行性能に改善が見られた．スキップ戦略だけを使用した場合と比較すると，仕様 A の実行時間以外の結果について，標準偏差を低減させる傾向を確認でき，実行性能のばらつきを抑える効果があることが期待できる．一方，仕様 B と C の実験において，反例長は最も大きいという結果だった．このことは，提案する探索戦略によって探索が効率よく進んだ一方，エージェントが状態空間の狭い領域を集中的に探索して得られる反例の短縮効果は弱まったことを示唆する．ただし，実行性能の改善と比べると，反例長の増加はわずかであると考えられる．

以上の実験結果から，提案する戦略を共に使用することで，従来法に比べて効率が向上しており，状態爆発問題を緩和する効果が期待できる．反例短縮問題については，提案する戦略で改善させることはできないものの，反例の可読性を大きく損なうほど悪化させてはいないと考えられる．ACOhg-live は古典的な探索法と比べて大きく反例長を短縮したという実験結果を Chicano と Alba は得ている [8] ことから，提案する探索戦略は，ACOhg-live に次ぐ反例短縮効果が期待できる．

5. 関連研究

群知能やメタヒューリスティクスを用いた形式検証技術は，近年活発に研究されている [3]．文献 [15] や [16] では，遺伝的アルゴリズム（GA）を用いた検査技法が提案されている．文献 [17] では，プロトコルの検証に，代

表 3: 仕様 B を用いた実験結果

		OR	ORT	SK	RE	RET	HY
反例長	平均	**27.47**	27.57	29.40	27.60	28.03	30.17
	標準偏差	**1.93**	1.94	2.74	1.96	1.97	2.31
	中央値	**26**	**26**	28	**26**	28.5	31
実行	平均	11.53	9.66	**1.16**	11.83	9.44	**1.16**
時間	標準偏差	0.68	1.18	0.37	0.73	1.07	**0.32**
[sec]	中央値	11.53	9.40	1.09	11.84	9.15	**1.04**
メモリ	平均	785.81	622.97	175.72	765.14	615.33	**164.95**
消費量	標準偏差	66.24	70.35	35.88	42.26	65.18	**28.88**
[KB]	中央値	773.68	628.91	174.52	774.07	603.93	**160.18**

表 4: 仕様 C を用いた実験結果

		OR	ORT	SK	RE	RET	HY
反例長	平均	28.93	**28.00**	29.70	29.07	29.77	31.30
	標準偏差	1.77	1.83	2.84	**1.69**	2.33	3.62
	中央値	30	**27.5**	29	30	30.5	31.5
実行	平均	12.04	9.56	1.49	11.78	4.83	**1.12**
時間	標準偏差	1.35	4.36	0.74	1.35	2.34	**0.39**
[sec]	中央値	11.71	9.16	1.41	11.42	4.55	**1.03**
メモリ	平均	770.61	483.03	176.92	726.17	349.29	**142.43**
消費量	標準偏差	129.75	193.30	44.22	52.75	124.68	**19.11**
[KB]	中央値	741.25	454.47	175.67	723.51	340.87	**139.15**

表的な群知能の 1 つである粒子群最適化法（PSO）を使った方法が提案されている．Chicano 等は，形式検証にシミュレーテッドアニーリング（SA）法を採用した [18]．加えて，従来の網羅探索によるモデル検査技法と GA，ACOhg，PSO，SA などの確率的探索技法を用いた検査技法の比較評価を行い，確率的探索の方が網羅探索よりも良好な結果が得られたという結果を報告している．

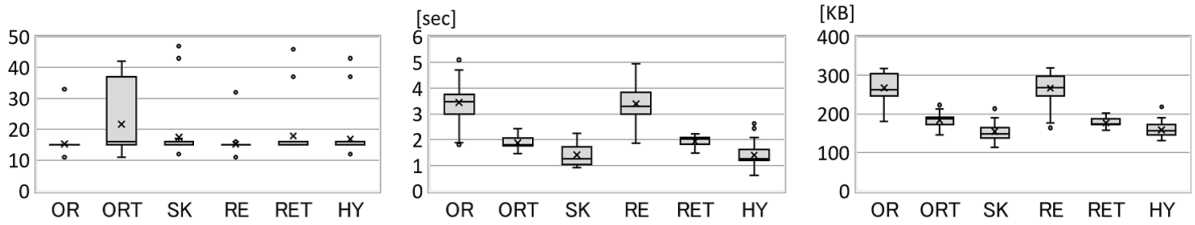
これまで述べた方法以外にも，分布推定アルゴリズム [19, 20]，モンテカルロ木探索法 [21]，PSO と重力探索法とのハイブリッド法 [22]，人口ハチコロニーアルゴリズムと SA とのハイブリッド法 [23] などを用いた検証技法が研究されている．

近年では，検証だけでなくソフトウェア工学の諸問題の解決に群知能を活用する，探索に基づいたソフトウェア工学と呼ばれるアプローチが注目されており，様々な研究成果が報告されている [24, 25]．本論文も，探索に基づいたソフトウェア工学の研究成果に位置づけられる．

6. おわりに

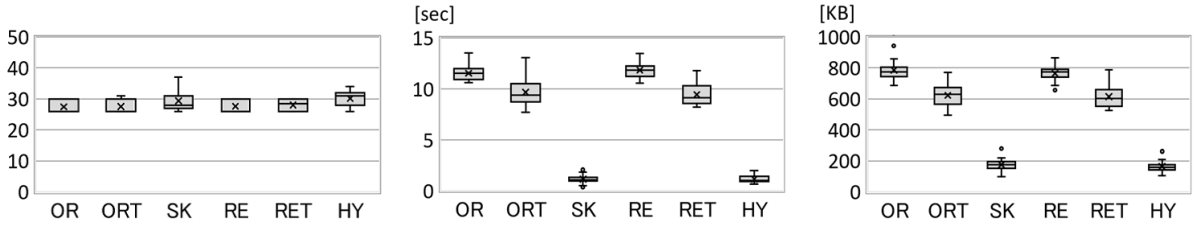
モデル検査における主要な研究課題に，状態爆発問題と反例短縮問題がある．これらの問題の解決のために，モデル検査に群知能を活用する方法が研究されている．本論文では，アリコロニー最適化法を用いたモデル検査アルゴリズム ACOhg-live の性能向上を目指し，新たな探索戦略であるスキップ戦略と差替え戦略を提案した．性能評価実験の結果，2 つの探索戦略を組合せることで，従来の ACOhg-live と比較して，実行速度とメモリ消費量については性能が向上することを確認した．反例の長さについては改善は見られないものの，可読性を大きく損なうほどには悪化しない，という結果であった．

今後の研究すべき課題について，次の点が重要である．第一に，本論文では，提案する探索戦略にヒューリスティクスを組込んだ場合について論じることができなかったので，適切なヒューリスティクスの下での探索戦



(a) 反例長 **(b)** 実行時間 [sec] **(c)** メモリ消費量 [KB]

図 5: 仕様 A を用いた実験結果の箱ひげ図



(a) 反例長 **(b)** 実行時間 [sec] **(c)** メモリ消費量 [KB]

図 6: 仕様 B を用いた実験結果の箱ひげ図

略の性能評価を行う必要がある．モデル検査で使われるヒューリスティクスは，有向モデル検査 [26] で研究されている．加えて，ACOhg-live 以外の群知能を用いる方法など，先行する検証技術との性能の比較評価も必要である．第二に，評価実験のために実装したプログラムはプロトタイプに留まるので，より実践的なプログラムに拡張することが挙げられる．従来のモデル検査で実装されている最適化法 [2] を組込むなど，実用化に向けた取り組みが必要である．

謝辞

有益なご指摘を頂いた査読者の方々に感謝する．

参考文献

[1] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, 1982.

[2] E. M. Clarke, Jr., O. Grumberg, D. Kroening, D. Peled, and H. Veith. *Model Checking*. MIT Press, 2nd edition, 2018.

[3] T. Kumazawa, M. Takimoto, and Y. Kambayashi. A survey on the applications of swarm intelligence to software verification. *Handbook of Research on Fireworks Algorithms and Swarm Intelligence*, pages 376–398, 2020.

[4] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Company, MIT Press, 2004.

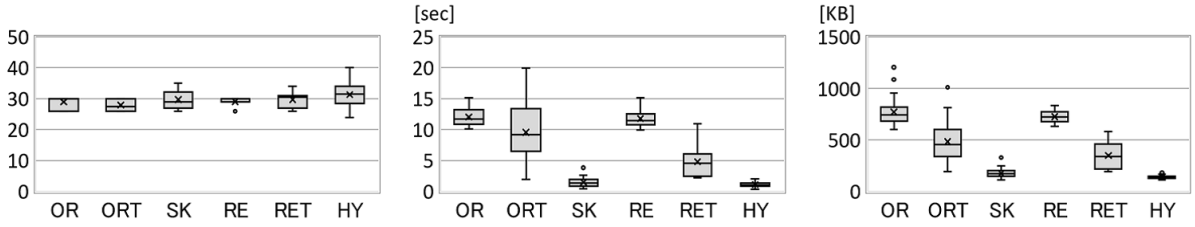
[5] E. Alba and F. Chicano. ACOhg: Dealing with huge graphs. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pages 10–17, 2007.

[6] E. Alba and F. Chicano. Finding safety errors with aco. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pages 1066–1073, 2007.

[7] F. Chicano and E. Alba. Ant colony optimization with partial order reduction for discovering safety property violations in concurrent models. *Information Processing Letters*, 106(6):221–231, 2008.

[8] F. Chicano and E. Alba. Finding liveness errors with ACO. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2997–3004, 2008.

[9] T. Kumazawa, C. Yokoyama, M. Takimoto, and Y. Kambayashi. Ant colony optimization based model checking extended by smell-like pheromone. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, 16(7), 4 2016.



(a) 反例長 **(b)** 実行時間 [sec] **(c)** メモリ消費量 [KB]

図 7: 仕様 C を用いた実験結果の箱ひげ図

[10] T. Kumazawa, K. Takada, M. Takimoto, and Y. Kambayashi. Ant colony optimization based model checking extended by smell-like pheromone with hop counts. *Swarm and Evolutionary Computation*, 44:511–521, 2019.

[11] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, 1986.

[12] R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[13] J. Magee and J. Kramer. *Concurrency: State Models & Java Programming*. John Wiley & Sons, Inc., 2nd edition, 2006.

[14] Aldebaran Format: https://www.mcrl2.org/web/user_manual/language_reference/lts.htm.

[15] E. Alba and J. M. Troya. Genetic algorithms for protocol validation. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 870–879, 1996.

[16] R. Yousefian, V. Rafe, and M. Rahmani. A heuristic solution for model checking graph transformation systems. *Applied Soft Computing*, 24:169–180, 2014.

[17] M. Ferreira, F. Chicano, E. Alba, and J. A. Gómez-Pulido. Detecting protocol errors using particle swarm optimization with java pathfinder. In *Proceedings of the High Performance Computing & Simulation Conference*, pages 319–325, 2008.

[18] F. Chicano, M. Ferreira, and E. Alba. Comparing metaheuristic algorithms for error detection in java programs. In *Proceedings of the Third International Conference on Search Based Software Engineering*, pages 82–96, 2011.

[19] J. Staunton and J. A. Clark. Searching for safety violations using estimation of distribution algorithms. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, pages 212–221, 2010.

[20] J. Staunton and J. A. Clark. Finding short counterexamples in promela models using estimation of distribution algorithms. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 1923–1930, 2011.

[21] S. Poulding and R. Feldt. Heuristic model checking using a monte-carlo tree search algorithm. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1359–1366, 2015.

[22] V. Rafe, M. Moradi, R. Yousefian, and A. Nikanjam. A meta-heuristic solution for automated refutation of complex software systems specified through graph transformations. *Applied Soft Computing*, 33(C):136–149, August 2015.

[23] N. Rezaee and H. Momeni. A hybrid meta-heuristic approach to cope with state space explosion in model checking technique for deadlock freeness. *Journal of AI and Data Mining*, 8(2):189–199, 2020.

[24] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 45(1):11:1–11:61, 2012.

[25] 半田久志. 探索に基づいたソフトウェア工学：SBSE − ソフトウェア工学におけるメタヒューリスティックスの援用 − . **知能と情報**, 24(6):224–229, 2012.

[26] S. Edelkamp, V. Schuppan, D. Bosnacki, A. Wijs, A. Fehnker, and H. Aljazzar. Survey on directed model checking. In *Proceedings of the 5th International Workshop on Model Checking and Artificial Intelligence*, volume 5348 of *Lecture Notes in Computer Science*, pages 65–89. Springer, 2008.

要求仕様に対するテストカバレッジ分析におけるグラフクエリの適用について

有若 新悟
大阪大学
s-ariwaka@ist.osaka-u.ac.jp

中川 博之
大阪大学
nakagawa@ist.osaka-u.ac.jp

土屋 達弘
大阪大学
t-tutiya@ist.osaka-u.ac.jp

要旨

本研究では，グラフデーターベース，および，グラフクエリのソフトウェア工学上の課題への応用について議論する．具体的には，要求仕様の集合とテストケース記述の集合，および，それらの要素間の類似度がデータとして与えられていることを前提に，どの要求仕様がどの程度テストされているかをグラフデータベースを用いて分析する方法を提案する．典型的な分析項目に対するグラフクエリと *SQL* クエリを示し，記述の簡潔さについてグラフクエリが優れていることを示す．また，実用規模のデータを用いて，クエリの処理時間についても比較を行う．

1. はじめに

グラフデータベースは，グラフ構造を有するデータを管理するデータベースであり，関係データベースにおける表形式での扱いが難しいデータの扱いに優れる [1]．グラフクエリは，グラフデータベース上のデータへのアクセスを行うクエリであり，探索したいデータのグラフ上の特徴を直観的に指定することができる．様々なグラフ構造を有するデータを取り扱うために，グラフデータベースが普及しつつある一方で，ソフトウェア工学分野における課題への応用例は多くない．

本研究では，ソフトウェアシステムの要求仕様をテストケースがどの程度網羅しているかというカバレッジの分析において，グラフデータベース，および，データクエリが有効に利用できるかどうかについて議論する．前提として，要求仕様の集合とテストケース記述の集合，および，それらの要素間の類似度が得られていることを仮定する．その上で，このデータをネットワーク構造として表現し，データクエリを用いて，カバレッジ分析に必要な情報を得ることが可能なことを示す．また，関係データベースで用いられる SQL クエリと，クエリの理解しやすさと簡潔さの点で比較を行う．更に，実際のプロジェクトから得られたデータを用いて，クエリの処理時間についても議論する．

2. データと抽出する情報

本論文では，既存研究 [2, 3, 4, 5] で扱っている形式のトレーサビリティデータが得られることを想定する．これらの研究では，自然言語で記述されたテストケースと要求仕様について，それらの類似度を自然言語処理を用いて推定している．このトレーサビリティデータは，テストケースの集合と要求仕様の集合，および，それらの要素間の類似度から構成される．類似度は，1) 異なる二つの要求仕様間，および，2) テストケースと要求仕様間，に定義される．また，類似度は，0 以上 1 以下の実数とする．表 1 にデータの例を示す．ここでは，要求仕様の集合は $\{r0, r1, \ldots, r6\}$, テストケースの集合は $\{t0, t1, \ldots, t9\}$, 類似度は表 1 の通りである．

次に，このデータを用いて，テストケースによってどの要求仕様がどの程度テストされているのかを調べることを考える．このカバレッジ分析において，データから得られる情報で有用と考えられる例を検討した [6]．そ

表 1. トレーサビリティデータの例

(a) 要求仕様間

$r0$	$r1$	0.14
$r0$	$r5$	0.08
$r0$	$r6$	0.59
$r1$	$r4$	0.97
$r1$	$r5$	0.84
$r2$	$r3$	0.2
$r2$	$r4$	0.94
$r2$	$r6$	0.34
$r4$	$r5$	0.93

(b) テストケースと要求仕様間

$t0$	$r0$	0.13
$t1$	$r0$	0.57
$t1$	$r1$	0.82
$t2$	$r2$	0.35
$t3$	$r2$	0.16
$t4$	$r2$	0.59
$t4$	$r4$	0.21
$t5$	$r4$	0.01
$t5$	$r5$	0.03
$t7$	$r1$	0.91
$t7$	$r2$	0.41
$t7$	$r4$	0.09
$t8$	$r0$	0.06
$t8$	$r1$	0.28
$t8$	$r2$	0.48
$t8$	$r5$	0.5
$t9$	$r1$	0.21
$t9$	$r2$	0.66
$t9$	$r5$	0.54

れらの例のいくつかを以下に挙げる．

1. 仕様 R を直接・間接にテストしているテストケースの一覧
2. テストケース T が直接・間接にテストしている仕様の一覧
3. 一つ以上のテストケースで直接・間接にテストされている仕様の一覧

ここでは，ある要求仕様に対して，類似度が一定のしきい値（X）以上のテストケースが存在している場合，この要求仕様はこのテストケースによって直接テストされていると定義する．また，ある要求仕様に対して，関連度が一定のしきい値（Y）以上である要求仕様が存在し，後者の要求仕様を直接テストしているテストケースが α 個以上存在する場合，前者の要求仕様は後者の要求仕様を直接テストしているテストケースによって間接テストされていると定義する．論文記述の簡素化のため，以降，三つのパラメータには X=0.5，Y=0.7，α=2 を割り当てる．

3. グラフデータとグラフクエリ

3.1. グラフデータ

本論文では，グラフ構造の表現形式として，グラフデータベース Neo4j のものを用いる．グラフ構造は，ノード・リレーションシップ・プロパティの 3 要素から成るデータ構造である．通常のグラフ理論の用語では，ノードは頂点，リレーションシップはノード同士を繋ぐ有向辺を表す．また，プロパティはノードとリレーションシップに付与される属性を意味する．

2 節で述べたトレーサビリティデータをグラフデータとして以下のように表現する．テストケースと要求仕様はノードに対応する．以降，テストケースに対応するノードをテストケースノード，要求仕様に対応するノードを要求ノードと呼ぶ．そして，要求仕様間，および，テストケースと要求仕様間に定義される類似度は，リレーションシップと，それに付与されるプロパティに対応する．ただし，類似度が 0 の場合は，リレーションシップ (辺) を設定しない．また，リレーションシップの向きは，ここで扱うデータにとっては意味を持たないので，要求ノード間，および，要求ノードとテストケースノード間

には，いずれかの向きのリレーションシップを一方のみを設定する．

3.2. グラフクエリ

2 節で挙げたカバレッジ分析に関する項目それぞれについて，対応する情報を抽出するためのグラフクエリを作成した．グラフクエリの記述には，Neo4j で用いられているクエリ言語である Cypher を用いた．Cypher はグラフ構造上の検索パターンを短いクエリで表現できることが特徴である．以降ではこれらのグラフクエリと，2 節の例に適用して得られる結果を示す．

(1) 仕様 R を直接・間接にテストしているテストケースの一覧

```
1  CALL {
2      MATCH (r:Requirement)-[s:
           Similarity]-(t:Testcase)
3      WHERE r.name = $s AND s.value >=
            0.5
4      RETURN t.name AS test_name
5
6      UNION
7
8      MATCH (r1:Requirement)-[s1:
           Similarity]-(r2:Requirement)
9      WHERE r1.name = $s AND s1.value
           >= 0.7
10     OPTIONAL MATCH (r2)-[s2:
           Similarity]-(t:Testcase)
           WHERE s2.value >= 0.5
11     WITH r2, count(t) AS count WHERE
            count >= 2
12     OPTIONAL MATCH (r2)-[s3:
           Similarity]-(t2:Testcase)
           WHERE s3.value >= 0.5
13     RETURN t2.name AS test_name
14 }
15 RETURN test_name
16 ORDER BY test_name
```

このクエリは，要求ノードの名前を指定し，その要求仕様を直接・間接にテストしているテストケースの一覧を取得する．3 行目，9 行目の WHERE 句で，要求ノードの名前を指定している．直接テストしているテストケースを求める部分（2 行目〜4 行目）と，間接にテストしているテストケースを求める部分（8 行目〜13 行目）を UNION 句によって合成している．

2 節の例に対して，クエリを実行した場合，要求ノード r1 を直接または間接にテストしているテストケースとして，t1, t7, t8, t9 が得られる．

(2) テストケース T が直接・間接にテストしている仕様の一覧

```
1  CALL {
2      MATCH (t:Testcase)-[s:Similarity
           ]-(r:Requirement)
3      WHERE t.name = $s AND s.value >=
            0.5
4      RETURN r.name AS req_name
5
6      UNION
7
8      MATCH (t1:Testcase)-[s2:
           Similarity]-(r1:Requirement)
9      WHERE t1.name = $s AND s2.value
           >= 0.5
10     OPTIONAL MATCH (r1)-[s3:
           Similarity]-(t2:Testcase)
           WHERE s3.value >= 0.5
11     WITH r1, count(t2) AS count
           WHERE count >= 2
12     OPTIONAL MATCH (r1)-[s1:
           Similarity]-(r2:Requirement)
            WHERE s1.value >= 0.7
13     RETURN r2.name AS req_name
14
15 }
16 RETURN req_name
17 ORDER BY req_name
```

このクエリでは，テストケースノードの名前を指定することで，そのテストケースが直接・間接にテストしている要求仕様の一覧を得る．3 行目，9 行目の WHERE 句で，テストケースノードの名前を指定している．このクエリを実行すると，テストケース t1 が直接または間接にテストしている要求仕様として，r0, r1, r4, r5 が得られる．

(3) 一つ以上のテストケースで直接・間接にテストされている仕様の一覧

```
1  CALL {
2      MATCH (r:Requirement)-[s:
           Similarity]-(t:Testcase)
           WHERE s.value >= 0.5
3      RETURN r.name AS req_name
4
5      UNION
6
7      MATCH (r1:Requirement)-[s1:
           Similarity]-(t:Testcase)
           WHERE s1.value >= 0.5
8      WITH r1, count(t) AS count WHERE
            count >= 2
9      MATCH (r1:Requirement)-[s2:
           Similarity]-(r2:Requirement)
            WHERE s2.value >= 0.7
10     RETURN r2.name AS req_name
11 }
12 RETURN req_name
13 ORDER BY req_name
```

このクエリでは，一つ以上のテストケースで直接・間接にテストされている仕様の一覧を得るクエリである．2

節の例に対してこのクエリを実行した場合，要求仕様 r0, r1, r2, r4, r5 が得られる．

3.3 SQL クエリとの比較

比較のため関係データベースを用いて，同様の処理を実現する方法を検討した．まず，2 節のトレーサビリティデータを管理するためのテーブルを設計した．具体的には，要求仕様の集合，テストケースの集合，類似度の集合を管理するため，それぞれに対して独立したテーブルを設定した．グラフデータベースでは一つのグラフ構造によってデータが表現できたのに対し，このように関連データベースでは複数の表を操作する必要があった．

上記のようにテーブルを設定した上で，クエリ言語である SQL を用いて，同様の目的を果たすクエリを記述した．表 2 に，対応するクエリにおける文字数の比較結果を示す．Cypher を用いた場合，記述量を半分近く削減できていることが分かる．

表 2. クエリの文字数

クエリ	Cypher	SQL
クエリ 1	415	724
クエリ 2	413	795
クエリ 3	325	599

例として，クエリ 2 に対する SQL クエリを以下に示す．SQL クエリは，データ同士の関係性を処理するために，JOIN キーワードを用いて何度もテーブルを結合する必要がある．また，グラフ構造に対して，グラフクエリのような，パターンマッチを用いた検索を行うこともできない．そのため，直接グラフ構造を扱うことができるグラフクエリに対して，クエリが冗長になっている．

(2) テストケース T が直接・間接にテストしている仕様の一覧 (SQL)

```
(SELECT req.name AS req_name
FROM edge_req_to_test edge
JOIN node_req req
ON edge.from_id = req.id
JOIN node_test test
ON edge.to_id = test.id
WHERE test.name = %s AND edge.
    similarity >= 0.5

UNION

(WITH req AS (
    SELECT id1 FROM (
        SELECT req.id AS id1
        FROM edge_req_to_test edge
        JOIN node_req req
        ON edge.from_id = req.id
        JOIN node_test test
        ON edge.to_id = test.id
        WHERE test.name = %s AND
            edge.similarity >= 0.5
    ) AS r
    JOIN edge_req_to_test edge
    ON edge.from_id = r.id1
    JOIN node_test test
    ON edge.to_id = test.id
    WHERE edge.similarity >= 0.5
    GROUP BY id1
    HAVING count(*) >= 2
)

SELECT req2.name AS req_name
FROM edge_req_to_req edge
JOIN node_req req2
ON edge.to_id = req2.id
WHERE edge.from_id IN (SELECT id1
    FROM req) AND edge.similarity >=
     0.7

UNION

SELECT req2.name AS req_name
FROM edge_req_to_req edge
JOIN node_req req2
ON edge.from_id = req2.id
WHERE edge.to_id IN (SELECT id1 FROM
     req) AND edge.similarity >= 0.7
))

ORDER BY req_name;
```

4. ケーススタディ

本節では，実際のシステム開発において得られたデータをグラフとしてグラフデータベースに格納した上で，前節で示したクエリを実行した際の結果について説明する．用いたデータの規模は，テストケース 3,855 個，要求仕様 260 個であった．

グラフデータベースとして Neo4j を，関係データベースとして postgreSQL を用いた．実験を行った計算機は，CPU AMD Ryzen 5 3600, メモリ 16GB，OS は Windows 10 である．

実験の前処理として，自然言語処理を通して得られた要求仕様間，および，テストケースと要求仕様との間の関連度のデータを，Cypher クエリを繰り返し実行することで，グラフ構造としてデータベース上に保存した．この処理は Python プログラムからデータベースの API を呼び出す形で実行した．この処理にかかった時間は，Neo4j では約 47 分，PostgreSQL では約 4 分であった．

各クエリを実行した結果，完了までにかかった時間を表 3 に示す．なお，クエリ 1 と 2 については，検索の起点として，要求仕様，テストケースをすべてを選んで実行し，実行時間の平均を取った．また，Neo4j では初回のアクセスには時間がかかるため，その影響を受けないように，事前に適当なクエリを実行した後に各クエリの実行時間の計測を行うようにした．PostgreSQL ではどのクエリも短時間で実行することができた．Neo4j では，一つ以上のテストケースで直接・間接にテストされている仕様の一覧を得るクエリ 3 で，比較的長い実行時間が必要であった．260 個の要求ノードすべてに対して網羅的に計算を行い，140 個もの仕様が抽出されたが，その場合でも実行時間は 5 秒程度であった．したがって，関係データベースよりは時間が必要ではあるが，十分実用的と考えられる．

表 3. クエリの実行時間 (秒)

クエリ	Neo4j	PostgreSQL
クエリ 1	0.054	0.075
クエリ 2	0.047	0.125
クエリ 3	5.086	0.098

5. 関連研究

本研究は，著者の以前の研究である文献 [6] を発展させたものである．具体的には，グラフクエリの記述を見直し，より簡素な記述を得た．また，その結果，処理時間についても改善が見られ，たとえばクエリ 3 では 50% 以上低減された．さらに，SQL クエリとの比較を行うため，関係データベースでのテーブルの設計と，その設計に基づく SQL クエリを記述し，記述の簡潔さだけでなく，クエリの処理時間についても評価を行った．

ソフトウェアアーティファクトのトレーサビリティに関して，グラフデータベースを応用した研究が幾つか知られている．文献 [7] では，テストケースのトレーサビリティに関するクエリについて，SQL と Cypher を含む 4 種類のクエリ言語を比較し，Cypher が表現力，理解容易性などの点で優れていることを示している．また，文献 [8] では，ネットワークによって要求，コード，テストケース間のトレーサビリティリンクを表現し，2 種類の単純なクエリについて，SQL と Cypher とでの表現の簡潔さについて比較している．これらの研究と本研究とでは，前提としているトレーサビリティデータや，クエリ実行の目的が大幅に異なっている．例えば，本研究ではこれらの先行研究で扱っているようなアーティファクト間の明示的なリンクではなく，類似度という数量を前提としており，また，要求仕様に対するテストカバレッジの分析という目的も，先行研究とは異なっている．文献 [9] では，データが大規模な場合におけるクエリの処理性能について議論されており，クラスタコンピューティングフレームワークである Spark 上の関係データベースによって SQL クエリを処理する方が，Neo4j と Cypher を用いるよりも処理性能が高いことを示している．単一の計算機を用いた本研究でも SQL クエリの方が処理時間の点で比較的有利であったが，前節の通り，ケーススタディで用いたデータセットに対してはグラフクエリでも十分に実用的な時間で処理ができた．トレーサビリティ以外のソフトウェア工学分野におけるグラフクエリの応用としては，文献 [10] において，グラフデータベースを用いて，ソフトウェアプロダクトラインにおける部品関係の可視化を行った事例について報告されている．

トレーサビリティリンクの自動構築については，いくつか研究が知られている．テストケースと要求仕様との類似度を用いて要求仕様のテストカバレッジを分析するアプローチについては，本研究の著者等による先行研究で，類似度の算出とトレーサビリティリンクの自動構築を行っている [2, 3, 4, 5]．本研究のケーススタディでは，これらの研究の方法で類似度を計算した．その他のトレーサビリティリンクの自動構築については，文献 [11, 12, 13] 等の研究がある．

6. おわりに

本研究では，要求仕様とテストケースに対して類似度がデータとして与えられていることを前提に，カバレッジ分析にグラフクエリを用いることが可能なことを示した．具体的には，重要な情報がグラフクエリによって記述できることを示し，SQL クエリより記述量を減らせることを示した．また，実用規模のデータを用いたケーススタディにより，クエリの処理時間について SQL 程ではないが，十分に実用的であることを示した．

今後の課題としては，より定量的な評価がある．たとえば，今回は二つの種類のクエリを文字数で比較したが，トークン数など，クエリの構造をより反映した尺度での比較評価の方が，より適切である可能性がある．また，

他のデータ，特に，規模の大きいデータへの適用や，ソフトウェア工学上の他の課題への応用などを検討したい．

謝辞 本研究は JSPS 科研費 JP20K11747 の助成を受けたものである．

参考文献

[1] I. Robinson, J. Webber, and E. Eifrem, Graph Databases: New Opportunities for Connected Data, 2nd edition, O'Reilly Media, Inc., 2015.

[2] H. Nakagawa, T. Hasegawa, S. Matsui, and T. Tsuchiya, "Visualization of specification coverage: A case study of a web application development in industry," 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp.77–80, 2017.

[3] H. Nakagawa, S. Matsui, and T. Tsuchiya, "A visualization of specification coverage based on document similarity," Proceedings of the 39th International Conference on Software Engineering Companion, p.136–138, ICSE-C '17, IEEE Press, 2017. https://doi.org/10.1109/ICSE-C.2017.117

[4] 東和幸，中川博之，土屋達弘，"文書間の類似度に基づいたトレーサビリティリンクの精度向上手法の検討," 電子情報通信学会 知能ソフトウェア工学研究会 (SIG-KBSE) 信学技報，pp.25–30，Nov. 2019．

[5] 松井勝利，中川博之，土屋達弘，"文書間の類似度に基づいた要求カバレッジ可視化手法," 日本ソフトウェア科学会 学会誌『コンピュータソフトウェア』，vol.35，no.1，pp.67–75，Feb. 2018．

[6] 有若新悟，中川博之，土屋達弘，"要求-テストケース間のカバレッジ分析におけるグラフクエリの応用可能性の検討," 電子情報通信学会 知能ソフトウェア工学研究会 (SIG-KBSE) 信学技報，pp.53–58，Nov. 2020．

[7] M. Rath, D. Akehurst, C. Borowski, and P. Mäder, "Are graph query languages applicable for requirements traceability analysis?," Joint Proceedings of REFSQ-2017 Workshops, Doctoral Symposium, Research Method Track, and Poster Track co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2017), vol.1796, p.(unassigned), CEUR Workshop Proceedings, 2017.

[8] R. Elamin and R. Osman, "Implementing traceability repositories as graph databases for software quality improvement," 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp.269–276, 2018.

[9] J. Lin, Y. Liu, and J. Cleland-Huang, "Supporting program comprehension through fast query response in large-scale systems," Proceedings of the 28th International Conference on Program Comprehension, p.285–295, ICPC '20, Association for Computing Machinery, New York, NY, USA, 2020.

[10] 川井隆之，小川雄太，水藤倫彰，"文書間の類似度に基づいたトレーサビリティリンクの精度向上手法の検討," ソフトウェア・シンポジウム 2020，p.84，2020．

[11] F. Erata, M. Challenger, B. Tekinerdogan, A. Monceaux, E. Tüzün, and G. Kardas, "Tarski: A platform for automated analysis of dynamically configurable traceability semantics," Proceedings of the Symposium on Applied Computing, p.1607–1614, SAC '17, Association for Computing Machinery, New York, NY, USA, 2017. https://doi.org/10.1145/3019612.3019747

[12] A. Goknil, I. Kurtev, and K. Van Den Berg, "Generation and validation of traces between requirements and architecture based on formal trace semantics," Journal of Systems and Software, vol.88, pp.112–137, 2014.

[13] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," Computer, vol.40, no.6, pp.27–35, 2007.

ソフトウェア信頼度成長モデルとベイズ統計機械学習によるオープンソースソフトウェア動的信頼性モデルの提案

杉山 透
放送大学
1520390580@campus.ouj.ac.jp

中谷 多哉子
放送大学
tinakatani@ouj.ac.jp

要旨

近年，産業界でオープンソースソフトウェア（OSS）を製品ソフトウェアに組み込んで使用することが増えてきているが，品質，特に信頼性に課題があると指摘されている．しかし，信頼性を評価するための既存のソフトウェア信頼性モデル，特に動的信頼性モデルは OSS 特有の開発形態と合っていないことが指摘されている．そこで，従来の動的信頼性モデルを取り込んだ状態空間モデルとベイズ統計機械学習による信頼性モデルを，従来の動的信頼性モデルに加えて OSS 特有の開発形態を考慮した動的信頼性モデルとして提案し，)実際の OSS の故障発生件数の予測を行った．提案手法を用いた予測結果を，従来の動的信頼性モデルにより同様の予測を行った結果と比較し，評価を行った．

1. 背景

近年，産業界におけるオープンソースソフトウェア（OSS）の開発・利用が一般的になっている．例えば，Anaconda,Inc. による Anaconda は，Jupyter Notebook のような OSS が多数組み込まれており，TypeFox 社の提供する gitpod には OSS である統合開発環境の theia が用いられている．

OSS の活用には，様々な利点が存在する．例えば，OSS が特定のベンダーに依存しない点，実装が公開されており独自に拡張することができる点等が挙げられる．しかし，OSS の品質が大きな課題となっている [8]．OSS の品質が社内基準を満たさない場合が多いためである．

システムやソフトウェアの品質について定めた ISO25010 において，ソフトウェア製品の品質は以下の 8 つの特性に分類されている [2].

1. Functional Suitability（機能適合性）
2. Performance Efficiency（性能効率性）
3. Compatibility（互換性）
4. Usability（使用性）
5. Reliability（信頼性）
6. Security（セキュリティ）
7. Maintainability（保守性）
8. Portability（移植性）

これらの品質特性の内，主に機能適合性や信頼性がソフトウェア製品に OSS を組み込む際に問題となると指摘されている [8]．このため，ソフトウェア製品に組み込む OSS の信頼性を作り込む必要がある．そして，信頼性を作り込んだ結果の評価に有用な手段がソフトウェア信頼性モデルである．

ソフトウェア信頼性モデルとは，開発プロセスや人的要因等を考慮して，ソフトウェア信頼性の定量的な評価を行う数理モデルである．ソフトウェア信頼性モデルには，大きく分けてテスト時間や運用時間等の時間的挙動を考慮せずに構築する静的ソフトウェア信頼性モデルと，時間的挙動を考慮して構築する動的ソフトウェア信頼性モデルの 2 つが存在する．しかし，OSS の信頼性モデル，中でも動的ソフトウェア信頼性モデルに基づいたものに関する研究はほとんど行われていない [11]．OSS の品質における特徴は，多くの開発者やユーザを巻き込んだ持続的なコミュニティを形成することで品質を向上さ

せていくことである [8]．このような特徴は，GitHub を始めとするソフトウェアの共同開発を支援するシステムが近年現れたことにより盛んになってきている．

対して，これまでの動的ソフトウェア信頼性モデルは，時間軸としてテスト項目の実施数を使うことがあるように，テストチームによるテスト工程を経ることが前提となっている．即ち，これまでのソフトウェア信頼性モデルと，OSS における信頼性の間にある乖離が大きくなってきている．そのため，OSS を自社製品に組み込むために信頼性を評価する場合，これまでのソフトウェア信頼性モデルでは正しく信頼性を評価できなくなる．この問題を解決するため，筆者らは状態空間モデルとベイズ統計機械学習を基本技術として用いた，OSS の信頼性を評価できる動的信頼性モデルを提案した [10]．その結果，従来の動的信頼性モデルが提案手法よりも高い精度で信頼性を評価できるという結果を得た．そこで，提案手法に従来の動的信頼性モデルを組み込むことで，提案手法の高精度化を試みた．本稿において，その結果を報告する．

本稿の構成を以下に述べる．まず 2 節にてソフトウェア信頼性モデルに関する関連研究を紹介し，関連研究における課題抽出を行う．3 節にて，モデル構築の基本技術について述べる．4 節では，公開されている実際の OSS バグレポートの集計結果と GitHub 上の開発履歴等を用いた信頼性モデルを実際に構築・評価し，従来の信頼性モデルによる評価結果との比較を行う．最後，5 節にて結果の考察や将来の展望を示す．

1.1. 用語の説明

本稿にて使用する用語の意味を表 1 に示す．

2. 関連研究

2.1. 本節で取り上げる関連研究について

ソフトウェアは，フォールトが見つかり，除去されることで信頼性が向上すると考えられる．信頼性の向上は信頼性の成長と言われ，定量評価手段として SRGM(software reliability growth model) が用いられている．SRGM は数多く提案されており，本論文では代表的な SRGM として以下 2 種を取り上げる．

1. 遅延 S 字型 SRGM
2. 対数ポアソン実行時間モデル

遅延 S 字型 SRGM は，モデルの意味が明白な代表的モデルであり [9]，筆者らによる研究で，OSS を対象としても高い精度で信頼度の成長を予測できた [10]．対数ポアソン実行時間モデルは，OSS の信頼度成長を表現する先行研究にて使用された実績 [11][14] がある．

加えて，状態空間モデルとベイズ統計機械学習を用いた信頼度推定を行った研究 [4][10] を示す．従来の SRGM は，故障件数をテスト時間や運用時間で説明するモデルであり，観測データのみで構成されている．対する当該研究においては，モデルに「状態」と呼ばれる潜在変数（状態変数）を考慮する．これにより，状態変数の時系列的変化に伴い故障件数も時系列的変化するモデルを表現することが可能になる．この状態変数の時系列的変化は，従来の SRGM よりも自由度の高いモデリングが可能であり，柔軟なモデルを構築できる．そして，状態空間モデルは構成要素として確率分布を用いる．そのため，データからモデルに用いられている確率分布のパラメータを推定する必要があり，推定の有効な手段となるのがベイズ統計機械学習である．

2.2. NHPP モデルによる SRGM

2.2.1 遅延 S 字型 SRGM

遅延 S 字型 SRGM（delayed S-shaped software reliability growth model）は，Yamada ら [13] が提案した SRGM である．本モデルは，故障の認知から，原因解析を経てフォールトの認知に至るまでのタイムラグを盛り込まれていることが特徴である．本モデルにおける平均値関数 $H(t)$ と強度関数 $h(t)$ を式 (1)(2) に示す．式中の a はその時点でソフトウェアに残存するフォールト数，b は 1 個あたりのフォールト発見率である．

$$H(t) = a[1 - (1 + bt)e^{-bt}] \quad (a > 0, b > 0) \tag{1}$$

$$h(t) = ab^2te^{-bt} \tag{2}$$

本モデルは，ソフトウェア内に存在する総フォールト数が有限であることが想定されている．しかし，OSS のような継続的に開発が行われていくようなソフトウェアでは，デバッグ時に新たなフォールトが混入する可能性があり，発見される総フォールト数は理論上無限大となると考える方が自然である．次に示す対数ポアソン実行時間モデルは，発見される総フォールト数が無限であるとしたモデルである．

表 1. 用語の意味一覧

用語	意味
ADVI	Automatic Differentiation Variational Inference，自動微分変分ベイズ
HMC 法	Hamiltonian Monte Carlo methods，ハミルトニアンモンテカルロ法
MCMC 法	Markov chain Monte Carlo methods，マルコフ連鎖モンテカルロ法
NHPP モデル	nonhomogeneous Poisson process モデル，ポアソン過程モデル
故障	アイテムが要求どおりに実行する能力を失うこと [7]
故障強度（瞬間故障強度）	修理アイテムの時間区間 $(t, \Delta t)$ での故障数を，その区間幅 Δt で除した値で，Δt を限りなくゼロに近付けたときの極限値 [7]
信頼性	アイテムが，与えられた条件の下で，与えられた期間，故障せずに，要求どおりに遂行できる能力 [7]
（ソフトウェア）フォールト	要求どおりに実行するのを妨げるソフトウェアアイテムの状態 [7]

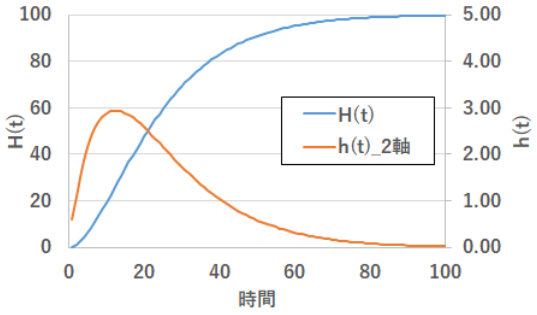


図 1. 遅延 S 字型 SRGM における信頼度成長（例）

2.2.2 対数ポアソン実行時間モデル

対数ポアソン実行時間モデル（logarithmic Poisson execution time model）は，Musa and Okumoto[6] が提案した SRGM である．本モデルは，強度関数 $h(t)$ が，実行時間（CPU 時間）までの平均値関数 $H(t)$ に関して指数関数的に減少するとしている．また，先述の通り，総フォールト数の上限を想定しておらず，テストをし続ける限り，無限にフォールトが発見されることになる．本モデルにおける平均値関数 $H(t)$ と強度関数 $h(t)$ を式 (3)(4) に示す．式中の λ_0 は初期故障強度，θ はソフトウェア故障 1 個あたりの故障強度減少率である．加えて，対数ポアソン実行時間モデルに則った信頼度成長の例を，図 2 に示す．

$$H(t) = \frac{1}{\theta}\ln(\lambda_0\theta t + 1) \quad (\lambda_0 > 0, \theta > 0) \tag{3}$$

$$h(t) = \frac{\lambda_0}{\lambda_0\theta t + 1} \tag{4}$$

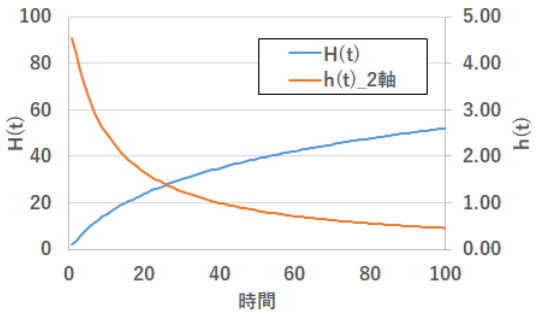


図 2. 対数ポアソン実行時間モデルにおける信頼度成長（例）

2.3. 状態空間モデルとベイズ統計機械学習による故障強度推定

2.3.1 カルマンフィルタによるアプローチ

貝瀬 [4] は，ソフトウェアの故障発生履歴から故障強度を推定する研究を行った．ソフトウェアを含むシステムの故障強度は，運用開始から時間経過と共に傾向が変化する．この傾向によって，システムの状態は初期故障

期間（故障率・故障強度減衰期間），偶発故障期間（故障率・故障強度一定期間），摩耗故障期間（故障率・故障強度増加期間）の 3 つの期間に分類される．分類法は，故障発生履歴から式 (5) に示すワイブル分布を推定し，パラメータ β の値によって判断するものが広く知られている．

$$f(t|\alpha,\beta) = \frac{\beta}{\alpha}(\frac{t}{\alpha})^{\beta-1}\exp[-(\frac{t}{\alpha})^{\beta-1}] \quad (\alpha>0,\beta>0) \tag{5}$$

式中の t は運用開始からの経過時間，α は尺度パラメータ，β は形状パラメータである．

貝瀬も，故障強度推定と同時にワイブル分布のパラメータを推定することで，ソフトウェアが各時刻においてどの期間の故障傾向を示すか推定した．当該研究では，ワイブル分布のパラメータが時間変化するモデルを構築し，ベイズ推定のデータからのパラメータ推定に長けた性質を利用して，カルマンフィルタによるパラメータのベイズ推定を行った．モデルの評価を行った結果，ワイブル分布のパラメータが時間変化するモデルは，パラメータが時間変化しないモデルよりもデータによくフィットする結果を得た．

当該研究は，観測データに含まれない潜在変数を柔軟に設定し，モデルに盛り込むのが特徴である．当該技術は，次節に示す筆者らの研究においても参考にした．

2.3.2 OSS 開発プロジェクトから抽出したデータを活用したアプローチ

筆者らは，OSS 開発プロジェクトの Git から抽出したデータを，様々な潜在変数と組み合わせて盛り込んだモデルを，OSS の信頼性評価に相応しいモデルとして提案した [10]．抽出・利用したデータは，コミットとアップデートの履歴である．筆者らは，これらを，1 章で示した OSS の特徴であるコミュニティにおける活動で信頼性を含む) 品質を高めた履歴であると捉えた．提案したモデルは，以下の 2 つである．前者を線形トレンドモデル，後者を定数係数モデルと呼称している．

以下に示すのが，線形トレンドモデルである．本モデルは，OSS の対数故障強度（$loglambda$）はトレンド要素（$trend$）と季節要素（$season$），コミット履歴（$C_{[t]}$）を定数倍したコミット要素（$commit$），アップデート履歴（$U_{[t]}$）を定数倍したアップデート要素（$update$）から成り立つとしたものである．本モデルのトレンド要素は，2 階線形トレンド要素として広く知られるものであり，トレンドがゆっくり滑らかに変化することを仮定したものである．この性質が SRGM のような滑らかな曲線を描くのに適していると考えた．

$$\begin{aligned}
trend_{[t]} &= \mathrm{Norm}(2*trend_{t-1} - trend_{t-2}, \sigma_{sys}) \\
\sum_{l=0}^{6} season_{[t-l]} &= \mathrm{Norm}(0, \sigma_{season}) \\
commit_{[t]} &= b_1 * C_{[t]} \\
update_{[t]} &= b_2 * U_{[t]} \\
loglambda_{[t]} &= trend_{[t]} + season_{[t]} + commit_{[t]} \\
&\quad + update_{[t]} \\
Y_{[t]} &= \mathrm{Poisson}(\exp(loglambda_{[t]}))
\end{aligned} \tag{6}$$

また，以下に示すのが定数係数モデルである．本モデルは，線形トレンドモデルのトレンド要素を変更したものである．モデル中の C_1 は定数係数であり，本係数が 0 より大きく 1 未満と推定されれば，トレンドが時事刻々と減衰し，SRGM のような曲線となると考えた．

$$trend_{[t]} = \mathrm{Norm}(C_1 * trend_{t-1}, \sigma_{sys}) \tag{7}$$

しかし，線形トレンドモデルは計算が発散し，定数係数モデルは故障発生予測精度が SRGM を下回った．

2.4. 関連研究における課題

OSS の信頼性評価について，動的信頼性モデルを用いた信頼性評価に関する研究がほとんど行われておらず，傾向分析に基づく事例研究としての文献はいくつか提案されているものの，これらは OSS のもつ特有の開発形態を考慮した信頼性評価を提案したものではないと指摘されている [11]．OSS 信頼性評価に関する研究として，田村ら [11]，山田 [14] による研究では，総フォールト数が無限となることを許容する対数ポアソン実行時間モデルが，OSS の開発形態を考慮した SRGM であると提案している．また，田村らは OSS 特有の要素であり信頼性を評価する際に重要な要素として，フォールト発見事象の複雑性やバグフィックス及びコンポーネントの加除が常に行われていることによるソフトウェア故障強度の変化を挙げている [11]．バグフィックスの履歴等は，近年は Git のコミット履歴といった形で詳細なデータに容易にアクセスできることが多く，モデルにも積極的に取

り込める環境が整ってきている．このようなデータを活かすには，先述の関連研究で用いられている，故障履歴と故障が発生した時間といった観測データ同士を直接結びつけるアプローチをさらに発展させ，様々なデータを取り入れることができるモデルを用いる必要がある．

貝瀬の研究 [4] では，状態空間モデルを構築し，モデル中の確率分布のパラメータをデータを用いベイズ統計機械学習するアプローチを取っている．これは，状態として OSS 特有の開発形態を盛り込むことができるアプローチであると考えられる．当該研究においても，故障発生間隔がワイブル分布に従うと考え，ワイブル分布のパラメータをデータからカルマンフィルタで逐次推定している．この手法により，例えばリリース直後の OSS の故障を除去することで，初期故障期間から偶発故障期間へ推移する様子を推定することが可能になる．しかし，カルマンフィルタは正規分布によって構築されたモデルにのみ適用可能な手法である．また，近年は理論とソフトウェアの進歩により，さらに複雑なモデルを構築し，ベイズ統計機械学習を適用することがことが可能になってきており，Git 上のデータもモデルに用いることができる．

コミット履歴のような，OSS 開発における様々なデータを柔軟に取り込み，OSS の信頼性評価に相応しいモデルとして筆者らが提案したのが，先述の線形トレンドモデルと定数係数トレンドモデルである [10]．しかし，故障発生予測精度が SRGM を下回る結果となってしまった．そこで，原因を精査し，さらに SRGM をトレンド要素として取り込むことで，提案手法の精度を向上させる．

筆者らの手法は，SRGM を状態空間モデルの枠組みで捉えてモデルの一部とし，ベイズ統計機械学習でパラメータ推定を行うものである．これら 2 つの技術は，筆者らの提案手法における基本技術である．次章にて，2 つの基礎技術について述べる．

3. 本稿で用いる基礎技術

3.1. 状態空間モデルについて

状態空間モデルは時系列解析において広く用いられるモデルであり，データ点同士が互いに関係を持つデータを確率的にとらえることができる．従来の SRGM では，観測値同士（例：故障発生時間と累積故障発生件数）の関係を直接記述するモデルを作成していた．対して，状態空間モデルでは，状態変数という潜在変数を導入したことにより，解釈に都合が良い状態変数を複数組み合わせて複雑なモデルを構築することが容易になった [3]．OSS の信頼性モデルを状態空間モデルで構築する場合，ソフトウェアのコミットのアップデート履歴データといった，OSS 特有の開発形態によりもたらされる豊富なデータを状態変数として用いることができる．

例として状態空間モデルの枠組みでソフトウェア信頼性モデルをとらえることを考える．この場合，観測量が時点 t における故障発生件数，状態が故障強度と考えることができる．さらに，ある時点 t における故障強度 h_t は，1 ステップ前の故障強度 h_{t-1} にのみ依存し，t における故障発生件数 Y_t は，t における故障強度 h_t によってのみ決まることになる．これを式で表現すると以下のようになる．このような，状態空間モデルの構造を表現する代表的な手法として，グラフィカルモデルを用いた表現と方程式を用いた表現の 2 つが存在する．グラフィカルモデルによる表現例を，図 3 に示す．状態である h_t が，観測量である Y_t の背後で変化している様子が表現されている．

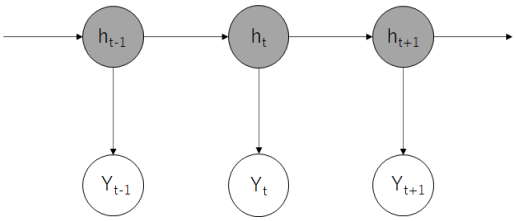


図 3. 状態空間モデルのグラフィカルモデル表現例

また，同じモデルを方程式で表現した例を，式 (8)(9) に示す．

$$h(t+1) = \mathrm{F}(h(t)) \tag{8}$$

$$Y(t) = \mathrm{G}(h(t)) \tag{9}$$

状態である h_t が状態方程式において関数 F に従って推移し，観測量である Y_t は，観測方程式において状態が関数 G に入力されることで生成されている様子が表現されている．

3.2. ベイズ統計機械学習について

ベイズ統計機械学習は，式 (10) のベイズの定理を基礎としている．式 (10) にて，x はデータ，θ は確率分布

のパラメータを指す.

$$p(\theta \mid x) = \frac{p(x \mid \theta)\, p(\theta)}{p(x)} \propto p(x \mid \theta)\, p(\theta) \qquad (10)$$

各項と文字の意味を以下に示す.

- x：観測値
- θ：モデル内における確率分布のパラメータ
- $p(\theta)$：事前分布．観測値 x を得る前のパラメータ θ の分布.
- $p(x \mid \theta)$：尤度．パラメータ θ のときデータ x が生成される確率
- $p(\theta \mid x)$：事後分布．観測値 x を得たときの θ の分布.
- $p(x)$：事後分布の面積を 1 にする正規化定数とみなせる.

上式は，モデルにおけるパラメータをデータにより更新する式と解釈することができる．しかし，パラメータが多い高次元モデルとなると，正規化定数 $p(x)$ の計算が難しくなることが知られている.

そこで，近年計算ベイズ統計でよく利用されている MCMC 法では，正規化定数 $p(x)$ を無視し，事後分布に比例する分布 $p(x \mid \theta)p(\theta)$ から乱数を大量に生成して，得られた分布から事後分布を作る．また，ADVI は，変分法を用いることで，近似解ではあるが MCMC 法よりも高速に事後分布を得ることができる.

4. ケーススタディ

4.1. モデル作成方針

先述の基本技術を用いて SRGM を取り込んだソフトウェア信頼性モデルを構築し，有効性を実際の OSS におけるデータを用いたケーススタディで検証する．モデル構築のインプットとして，OSS の故障履歴データ，コミット履歴を用いる．2.3.2 項における筆者らの研究 [10] では，アップデートの履歴も使用した．しかし，アップデートの履歴はコミットの履歴に内包されるため，結果に悪影響を及ぼす可能性があると考え，今回は使用していない．故障履歴データを分析することにより，故障のトレンド傾向や故障の周期的傾向の有無を抽出することができる．故障にトレンド傾向や周期性が見られた場合は，非定常データで過去に依存する傾向を持つデータであり，対応する要素をモデルに組み込む.

4.2. 推定と予測に用いるデータ

Eclipse のプラグインとして提供されている OSS である，JDT（Java development tools）の故障データを使用した．バグレポート 1 件を故障 1 件とし，起票日を故障発生日とした．また，開発チームの Git より，コミット履歴データを抽出した．抽出対象は，全てのコミットの日付と回数とした.

データを取得した期間を，表 2 に，JDT の故障発生件数の履歴を図 4 に示す.

表 2. JDT の各データ取得期間

データ	期間	日数（日）
故障データ	2001/10/10〜2013/11/19	4424
コミット履歴	2002/05/09〜2013/11/19	4213

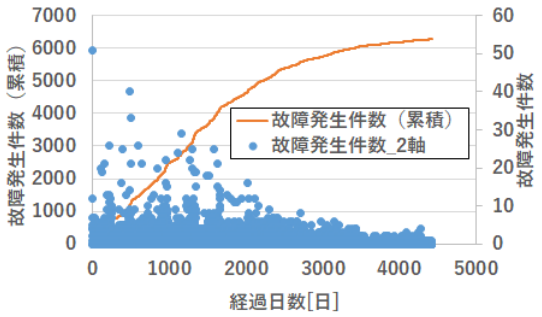


図 4. JDT の故障発生履歴

累積故障発生件数の上昇が，日数の経過ごとに鈍くなっているため，トレンドを持つデータであることがわかる．また，JDT の自己相関係数によるコレログラムを図 5 に示す．図中の青線が，自己相関の有無を判定する閾値である．図 5 より，概ね 7 日間で高い自己相関を示していることがわかり，7 日周期を表現する要素をモデルに組み込む必要がある.

さらに，JDT の偏自己相関係数によるコレログラムを図 6 に示す．自己相関係数によるコレログラムは，t_i 日目のデータと t_{i+a} 日目のデータの間の関係を調べようとする時，t_{i+1} 日目から t_{i+a-1} 日目までのデータの影響を受ける．しかし，偏自己相関係数によるコレログラムは，t_i 日目のデータと t_{i+a} 日目のデータの関係だ

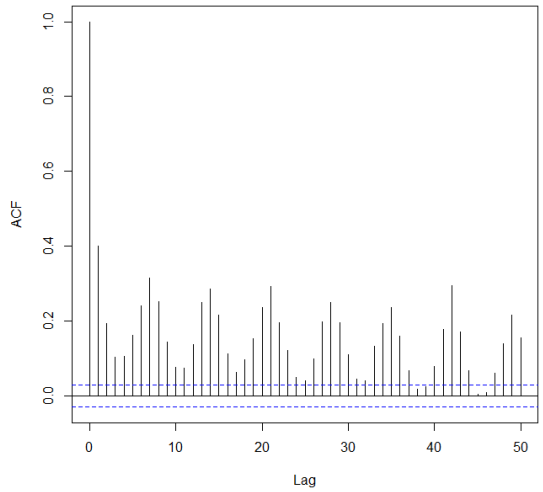


図 5. コレログラム˙50 日分（横軸：[日]）

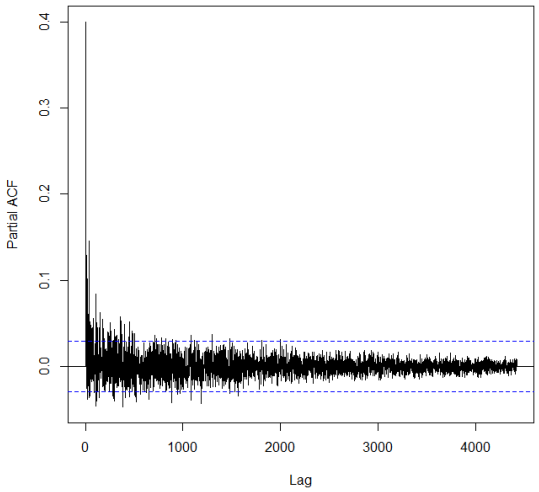


図 6. コレログラム（偏自己相関係数）（横軸：[日]）

けを純粋に調べることができる．よって，トレンドの効果を完全に除去してデータを評価できると考えた．

図 5 と同じく，図中の青線が，自己相関の有無を判定する閾値である．高い自己相関を示すのは，図 5 と同じく，概ね 7 日周期の成分である．違いは，徐々に自己相関係数が減衰し，概ね 2000 日後からは相関が認められなくなる．よって，2000 日以後は，季節要素の影響を極めて小さくする必要がある．以前の筆者らの研究 [10] では，1 日目から最後まで季節要素が同じように効果を及ぼすモデルであったため，学習が終わって予測を行う際に，季節要素が大きくなりすぎて予測に失敗している．

以上より，JDT の故障履歴はトレンド傾向を示し，2000 日目程度まで 7 日の周期を持つデータであることがわかったため，モデルにトレンド要素と，データの最初から 2000 日の期間のみ，7 日周期の季節要素を盛り込む．

これらのデータを用い，状態空間モデルとベイズ統計機械学習による故障発生件数の予測を試みた．藤原 [1] によれば，テストが中盤に差し掛かるか，テスト工程を 60 %程度消化するまでは，モデルのパラメータ推定は行わないほうが良い．そのため，筆者らの研究 [10] において 60%以上のデータを用いて解析したケースである，75%分のデータを用いて学習し，残りの区間における故障件数を予測するケースの解析を実施し，SRGM による予測結果と比較する．予測に用いたモデルは次節に示す．

4.3. 解析モデル

4.3.1 遅延 S 字型トレンドモデル

遅延 S 字型 SRGM を取り込んだ状態空間モデルを以下に示す．本モデルは，トレンド要素（$trend$）を，遅延 S 字型 SRGM の強度関数である式 (2) を変形することで作成し，7 日周期を表現する季節要素（$season$）と，コミット履歴に依存するコミット要素（$logcommit$）から対数故障強度（$loglambda$）が成り立つとした．

$$
\begin{aligned}
trend_{[t]} &= trend_lin_{[t]} + \log(trend_log_{[t]}) \\
trend_lin_{[t]} &= trend_lin_{[t-1]} - b \\
trend_log_{[t]} &= trend_log_{[t-1]} + a * b^2 \\
\sum_{l=0}^{6} season_{[t-l]} &= \mathrm{Norm}(0, \sigma_{season}) \\
logcommit_{[t]} &= logcommit_{[t-1]} + \mathrm{Norm}(0, \sigma_{commit}) \\
loglambda_{[t]} &= trend_{[t]} + season_{[t]} \\
&\quad + (c_1 * logcommit_{[t]} + d_1) \\
C_{[t]} &= \mathrm{Poisson}(\exp(logcommit_{[t]})) \\
Y_{[t]} &= \mathrm{Poisson}(\exp(loglambda_{[t]}))
\end{aligned}
\tag{11}
$$

式中の $C_{[t]}$ は，日付 t におけるコミット回数である．また，$\mathrm{Norm}(\mu,\ \sigma)$ は平均 μ，標準偏差 σ の正規分布，

Poisson(λ) は平均 λ のポアソン分布である．

季節要素は，任意の連続した 7 つの $season[t]$ を足すと，非常に小さな値 $\mathrm{Norm}(0, \sigma_{season})$ になるとした．これにより，7 日周期の変動を表現することができる．

コミット要素は，毎時のコミット履歴（$C_{[t]}$）から毎時のコミット発生回数の平均値を推定したものである．これを c_1 倍し，さらに d_1 を足すことで対数故障強度 $loglambda$ 算出に用いている．

コミット回数と故障発生件数は，ポアソン分布に従うとした．ポアソン分布は計数データを表現する標準的な確率分布であると同時に，NHPP モデルに用いられているためである．

対数故障強度 $loglambda$ は，トレンド要素，季節要素，コミット要素（定数倍・定数加算済）の線形和とし，対数故障強度の指数を取ったものをポアソン分布に適用することで，故障発生件数となるモデルとした．

4.3.2 対数ポアソン実行時間トレンドモデル

遅延 S 字型トレンドモデルの，トレンド要素の式を以下のように変えたものである．これは，対数ポアソン実行時間モデルの強度関数である式 (4) を変形したものである．式中の λ_0 と θ の意味は，同式と同じく，各々初期故障強度と故障 1 個あたりの故障強度減少率である．

$$trend_{[t]} = -\log(trend_log_{[t]} + 1) + \log(\lambda_0) \quad (12)$$

$$trend_log_{[t]} = trend_log_{[t-1]} + \lambda_0\theta \quad (13)$$

4.4. 解析結果

前節にて示したモデルを，Google Colaboratory 上で，Pystan(ver.2.19.1.1) を用いて推定を行った．Pystan は MCMC 法に属する HMC 法と変分法に属する ADVI を使用できるが，計算時間の問題から ADVI を用いた．

推定した対数故障強度 $loglambda$ の中央値に指数関数を適用し，累積和を取ることで平均値関数 $H(t)$ とした．各 SRGM も，75%の故障発生件数データを学習データとして残りの故障発生件数を推定し，ベイズ統計機械学習の結果と比較した．SRGM は，学習データに対して最小二乗法でフィッティングし，予測に用いた．

4.4.1 遅延 S 字型トレンドモデル

結果を図 7 に示す．図中の黒色の実線は，累積故障発生件数実績値である．また，橙色の実線はベイズ統計機械学習による解析結果，2 本の点線は SRGM （遅延 S 字型 SRGM，対数ポアソン実行時間モデル）による解析結果である．なお，学習データと予測対象データの境界は，図中に赤色の点線で示した．また，図 8 は，対数領域における $loglambda$ の各要素への分解結果である．モデル内のパラメータの推定結果を，表 3 に示す．

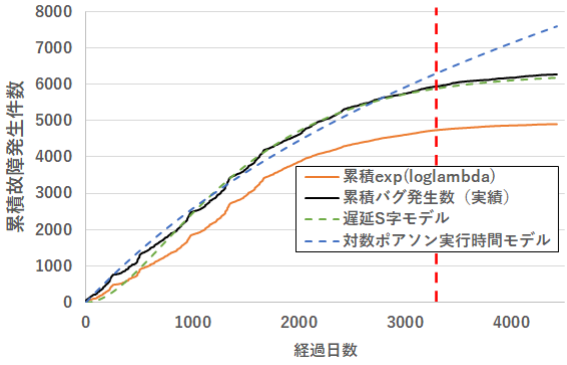


図 7. 故障発生件数の予測結果

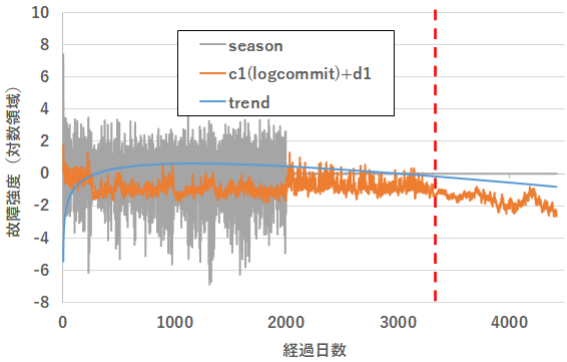


図 8. loglambda の各要素への分解結果

表 3. パラメータの推定結果

a	b	c_1	d_1
5955.49	0.000851	0.853769	-0.11752

本ケースにおける解析結果は，学習時点の間から実績の故障発生数より低めであり，SRGM，特に遅延 S 字型

SRGM が学習期間・予測期間ともに高い精度を示した．また，実績の総故障発生数は 6270 だが，予測総故障数である a の推定結果は 5955.49 と低めである．

4.4.2 対数ポアソン実行時間トレンドモデル

結果を図 9 と図 10 に示す．図中の線の意味は，遅延 S 字トレンドモデルと同じである．また，モデル内のパラメータの推定結果を，表 4 に示す．

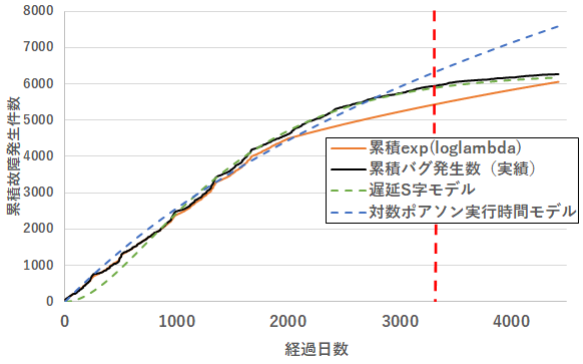


図 9. 故障発生件数の予測結果

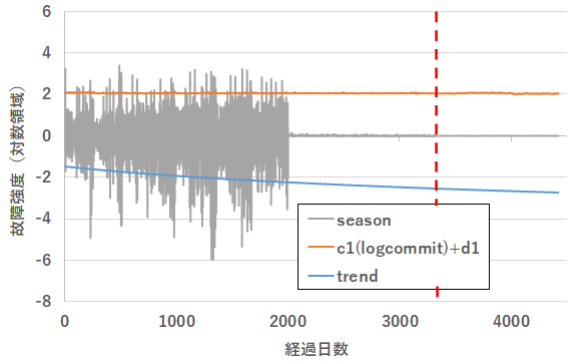


図 10. loglambda の各要素への分解結果

表 4. パラメータの推定結果

λ_0	θ	c_1	d_1
0.227249	0.002493	0.04024	2.09571

予測期間中において，対数ポアソン実行時間モデルよりも良好な予測精度を示しており，信頼性が安定しているという判断は正しく行えると考えられる．遅延 S 字型 SRGM よりも予測精度は下回ったが，500 日目までの複雑なバグの発生履歴には，SRGM2 種よりも高い追従性を示している．また，コミット要素に定数倍と定数加算を施した要素は，対数領域において，毎時ほぼ一定となった．

5. 考察と課題

提案手法である対数ポアソン実行時間トレンドモデルが，元の SRGM よりも学習期間中の一部を除き，高い予測と追従の精度を見せた．学習期間も，図 9 における 500 日目までの複雑な故障の発生履歴によく追従している．これは，7 日周期の季節要素をはじめとする，SRGM に無い要素の効果である．2000 日付近の，季節要素を操作した範囲で実績累積バグ発生件数と推定値に乖離が生じてしまったが，この箇所は季節要素の影響をゼロにする操作を開始した箇所であり，本操作が結果に悪影響を及ぼしている可能性がある．季節要素もトレンドのように減衰させるなどの工夫すれば精度の良い推定，さらには予測が可能になり得る．これは，バグレポートが作成される数の曜日ごとのムラの性質を把握できるようになることを意味し，信頼度成長曲線の横軸にテスト工程消化具合を使うのが難しく，日数を使う場合に OSS の信頼度を表現する大きな助けになると考えらえる．OSS の信頼度を的確に表現し成長を予測できるようになれば，OSS 採用を採用しようとしている組織にとって，採用可否や採用時にさらなる信頼性の作り込みを要するか否かの意思決定にも有用であると考えられる．そのためには，JDT 以外に様々な OSS，特に JDT と異なる故障の発生傾向を持つ OSS に対して本稿における提案手法を使用し，有効性を確かめていく必要がある．

反面，遅延 S 字型トレンドモデルは，元の SRGM よりも低い精度と追従の精度となってしまった．これは，遅延 S 字型 SRGM において，ソフトウェアが内包する予測故障数をパラメータ a として持っているため，SRGM 以外の要素も故障に関係すると考える本稿におけるアプローチは相性が悪いためであると考えた．a の推定結果と，総故障発生数のギャップもこのためであると考えられる．対して，対数ポアソン実行時間モデルは，故障強度が初期故障強度と時間だけに依存して減少するモデルであるため，他の要素を組み込むことができたと考えられる．この仮説を実証するならば，例えば，似た性質のワイブル過程モデル等の SRGM を用いてモデルを構築

して解析する必要があるだろう．

本稿におけるアプローチは，状態変数を用いてデータを要素に分解するものである．本アプローチは，経済・経営学 [16] 等，幅広い分野の時系列や空間データ解析に活用されている．よって，このようなデータと，適切な要素を具備したモデルを構築できれば，信頼性に留まらずソフトウェア工学の諸分野に適用できると考えられる．

図 10 にて，コミット要素を変換した要素が毎時ほぼ一定になり，トレンド要素が減少傾向になった．これは，故障強度が減衰するトレンド成分と，コミット履歴から推定された，毎時一定の成分に分解されたと考えられる．

三中が指摘するように，統計モデルによるデータの説明は，科学の方法論におけるアブダクションに従っている [5]．このアブダクションとは，複数の対立するモデルの中から最も良いモデルを選び出すこと（モデル選択）を指す [5]．本稿におけるモデルでも，SRGM の選択や，要素の取捨選択など，モデル選択の選択肢は多岐に渡る．その中で効率的に解析を行うための手法として，周辺化対数尤度や AIC を取得データと考えての直交表実験といったものが考えられる．直行表実験を使用するには様々な条件があるが，使用可能であればモデル選択の大きな助けになる．そのため，実験計画法の使用可否の検討や，モデル選択にも使える，実験計画法に相当する手法の構築が必要であろう．

参考文献

[1] 藤原隆次（著），木村光宏（編），信頼性技術叢書編集委員会（監）：ソフトウェアの信頼性，日科技連，2011

[2] ISO/IEC 25010 Software Product Quality：http://iso25000.com/index.php/en/iso-25000-standards/iso-25010（2021 年 5 月 21 日確認）

[3] 萩原淳一郎，瓜生真也，牧山幸史（著），石田基広（監）：基礎からわかる時系列分析 R で実践するカルマンフィルタ・MCMC・粒子フィルタ，技術評論社，2018

[4] 貝瀬徹:ワイブル分布に基づく階層的ベイズモデルの状態空間表現とフィルターによる推定，信頼性シンポジウム発表報文集，第 13 巻 (2000)

[5] 三中信宏：統計思考の世界 曼荼羅で読み解くデータ解析の基礎，技術評論社，2018

[6] J.D.Musa and K.Okumoto: A Logarithmic PoissonExecutionTime Model for SoftwareReliability Measurement, ICSE ’84: Proceedings of the 7th international conference on Software engineering, 1984, pp.230-238

[7] 日本規格協会：JIS Z 8115 ディペンダビリティ（総合信頼性）用語, 2019

[8] 野村佳秀，木村巧作，福寄雅洋，谷田英生：『オープンソースソフトウェア工学シリーズ』企業における OSS 活用の実際，コンピュータソフトウェア，Vol.33，No.3(2016)，pp.50-65.

[9] 尾崎俊治：非定常ポアソン過程モデル，情報処理，Vol.31，No.12(1990)，pp.1631-1640

[10] 杉山透，中谷多哉子：ベイズ統計機械学習を用いたオープンソースソフトウェア信頼度評価方の提案，信学技報 vol.120，No. 423（2021），pp.71-76

[11] 田村慶信，田中智朗，山田茂：オープンソースソフトウェアに対する SRGM に基づく予測精度の検証，数理解析研究所講究録，1636 巻（2009），pp.243-250

[12] 山田茂：ソフトウェア信頼性の基礎-モデリングアプローチ，共立出版，2011, pp.19-22

[13] Shigeru Yamada, Mitsuru Ohba, Shunji Osaki：s-Shaped Software Reliability Growth Models and Their Applications, IEEE Trans. Reliability, Vol.R-33, No.4(1984), pp.289-292.

[14] 山田茂：OSS プロジェクトデータに基づく統計的プロセス管理法とその応用に関する研究，オペレーションズ・リサーチ，Vol.61, No.10（2016），pp.666-667

[15] 山田茂，井上真二，田村慶信：ソフトウェア信頼性研究-モデリングアプローチ-，電子情報通信学会 基礎・境界ソサイエティ Fundamental Review，Vol.12，No.1(2018)，pp.38-50

[16] 山口類，土屋映子，樋口知之：状態空間モデルを用いた飲食店売上の要因分解，オペレーションズ・リサーチ，Vol.49，No.5（2004），pp.316-324

要求仕様に対する形態素ベースドレビューの提案

柏原 一雄
株式会社デンソークリエイト
kazuo.kashiwabara.j3a@jpgr.denso.com

鈴木 淳
株式会社デンソークリエイト
atsushi.suzuki.j3h@jpgr.denso.com

新留 光治
株式会社デンソークリエイト
mitsuharu.shintome.j8u@jpgr.denso.com

松原 潤弥
株式会社デンソークリエイト
junya.matsubara.j4v@jpgr.denso.com

李 路雲
株式会社デンソークリエイト
uyun.li.j4z@jpgr.denso.com

不破 慎之介
株式会社デンソークリエイト
shinnosuke.fuwa.j8h@jpgr.denso.com

要旨

要求仕様が自然言語のような曖昧さを含む記法で記述されている場合，曖昧に表現された文が不具合を誘発させることがある．実際に我々の組織では，「修飾語」または「同義語」を使用している文が不具合を誘発していた．要求仕様書から修飾語・同義語が使用されている曖昧な文を検出するために，「形態素ベースドレビュー」を考案した．実開発に形態素ベースドレビューを導入し，効果を確認した．

1. はじめに

ソフトウェア開発において，仕様は関係者の間の情報ハブ[1]となり，仕様定義者は多数の関係者と仕様によりコミュニケーションをする．文献[1]では，形式的な品質の高い仕様は，定義内容の検索や変換，そして抽出や機械的な検証を行うことが容易になるため，プロジェクトのさまざまな局面での活用が可能になると述べられている．仕様の品質を向上するためには，曖昧さのない文を書くことが大切になる．文献[2]では，文の曖昧さは，表 1に示すように多義性・修飾性・類義性の 3 つに分類されると述べられている．

実際に，我々の組織では，要求仕様の修飾性と類義性により，不具合が引き起こされた事例があった．具体的には，「修飾語」または「同義語」を使用していた文が，不具合を誘発していた．要求仕様に対するレビューを実施していても，修飾性と類義性の問題を見逃していた．

表 1 文の曖昧さの分類

文の曖昧さの分類	説明
多義性	一つの単語で異なる意味を表現する．
修飾性	修飾関係が一意に決まらない．
類義性	異なる単語で同じ意味を表現する．

本研究では，欠陥混入のリスクを低減することを目的とし，自然言語で記述された要求仕様書から，修飾語・同義語が使用されている曖昧な文を検出する手法を考案した．曖昧表現[3]すべてを解決の対象とするのではなく，実際に不具合に繋がった曖昧表現のみを解決の対象とする．また，考案手法は，稼働中のプロジェクトにも少ないコストで導入できるようにした．

考案手法は，「形態素ベースドレビュー」と呼ぶ．形態素ベースドレビューは，要求仕様に対して形態素解析を実施した結果を入力とするリーディング技法[4]である．

実験では，実開発で作成された要求仕様書を対象に形態素ベースドレビューを実施した．評価の結果，形態素ベースドレビューが以下の要求を満たすことを確認できた．

- アドホックレビューでは見逃した曖昧な文を検出できる
- 不具合を誘発した曖昧な文を検出できる
- 稼働中のプロジェクトに対して導入しやすい

実開発において，形態素ベースドレビューを活用することで，要求仕様の表現を起因とした不具合を減らす効果が期待できる．

これ以降の本稿の構成は次のとおりである．2 章で現状分析の結果と研究の課題を示す．3 章では，課題解決の参考とした先行研究を示す．4 章では，考案した解決策を提案する．5 章では，提案手法の評価結果と考察を示す．6 章では，まとめと今後の進め方を示す．

2. 課題設定

2.1. 現状分析

(1) 不具合を誘発した曖昧な文

我々の組織では，要求仕様の表現を起因とした不具合が発生していた．不具合の事例を分析すると，不具合を誘発した曖昧な文では，「抽象的な語」「修飾語」「同義語」を使用していた．この3種類の単語が使われている文で発生する問題を表 2に示す．

表 2 不具合を誘発した単語の種類

単語の種類	関連する曖昧性	発生する問題
抽象的な語	多義性	仕様の誤解釈
修飾語	修飾性	仕様の誤解釈
同義語	類義性	検索での関連仕様の抽出漏れ

「抽象的な語」が誘発する問題に対しては，文献[5]で解決策の一つとなる手法「Domain Word Modeling」が提案されている．Domain Word Modeling は，“抽象的に表現された用語”を“不足している前提知識”で誤解釈することにより発生していた手戻りを防止する．

本研究では，「修飾語」または「同義語」を使用している文が誘発する不具合を防止することを目的とする．

(2) 修飾語

修飾語を使用している文は，仕様の誤解釈を誘発しやすい．修飾語を付けて対象とすること・ものを説明している場合に，その絞り込んでいる対象範囲を誤解釈しやすい．例えば，以下のような文が問題を引き起こした．

例：
- “使用可能な”モード以外には遷移しない．
- “不要となる”処理を実行しないように変更する．
- 初期化が“必要な”データを更新する．

上記の例の場合，誤解釈を防止するために，「“使用可能な”モード」，「“不要となる”処理」「初期化が“必要な”データ」が，他の文で具体的に定義されている必要がある．

修飾語となる品詞[6]には，形容詞・副詞・形容動詞・副詞可能等がある．これらの品詞が，誤解釈を引き起こしやすいと考えた．

(3) 同義語

同義語[7][8]を使用している文は，検索で関連する仕様の抽出漏れを誘発する．例えば，以下のような同義語が問題を引き起こした．

例：
- OffCounter / OFF カウンタ / 消去判定カウンタ
- 合計額 / 合計金額 / 投入金額

“ドメイン用語[5]であり”かつ“表記ゆれでない”同義語が，特に問題を誘発しやすいことがわかった．ドメイン用語は，一般的な用語に比べて検索対象となることが多く，問題を引き起こす可能性が高い．また，文章作成ソフト等の表記ゆれチェック機能では検出できない同義語は見逃しやすい．例えば，「売切ランプ」と「売り切れランプ」のような似ている単語は，ツールで表記ゆれとして検出できるが，例のような似ていない単語は表記ゆれとして検出できない．表記ゆれでない単語は，ツールでは同義語として検出しにくく，問題を引き起こす可能性が高い．

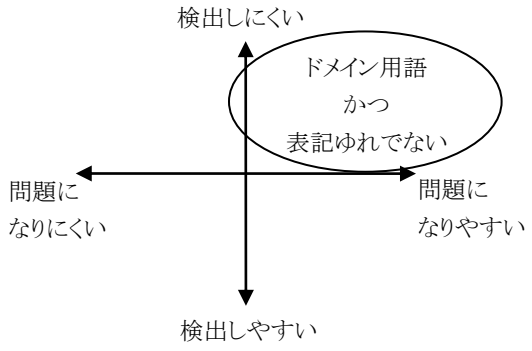


図 1 問題を誘発しやすい同義語

(4) 要求仕様のレビュー

我々の組織で実施されている要求仕様に対するアドホックレビューでは，上から順に文を読む．このとき，文脈と短期記憶に依存して読むため，曖昧な文の検出漏れが起きやすい．

文脈に依存して読むため，レビューで修飾語を使用している文の曖昧さを指摘することが難しい．文献[9]で，「わかった状態」になるメカニズムが説明されている．“人にとって「わかった」状態は安定状態であり，「わかった」状態になりたがる．人は，文章を読むとき，「わかった」状態にするために，文脈を使う”と説明されている．つまり，人は文脈をもとに読むと，厳密な記述でなくても，「わかった」状態になってしまう．「わかった」ことを「わからないかもしれない」と，レビューで指摘することは難しい．更に，書き手と読み手がそれぞれ使用している文脈はレビュー対象の文書には表現されない．表現されておらず見えない書き手と読み手の文脈の違いを，レビューで指摘することは難しい．

短期記憶に依存して読むため，レビューで同義語を使用している文を検出することが難しい．同義語を抽出するためには，他の文で使用されている単語を記憶しておく必要がある．しかし，人は短期的には 7±2 程度のものしか覚えられないと考えられている[10]．そのため，レビュー対象の文の量が多くなればなるほど，単語を網羅的に比較し，同義語を抽出することは難しくなる．

「修飾語」または「同義語」を使用している曖昧な文を検出するためには，“上から順に文を読む”ではないリーディング技法が必要となる．

2.2. 課題提起

本研究では，要求仕様書から，「修飾語」「同義語」が使用されている曖昧な文を検出する手法を考案することを課題とする．

考案手法は，不具合を防ぐため，以下の問題を防止することを目的とする．

- 修飾語の使用箇所で起きる仕様の誤解釈
- 同義語の使用箇所で起きる検索での関連仕様の抽出漏れ

考案する手法は，早く実開発に導入し，確実に効果を得るために，以下の 3 つの要求を満たすものとする．

- “上から順に文を読む”アドホックレビューで見逃した要求仕様の曖昧な文を検出できる
- 実際に不具合を誘発した要求仕様の曖昧な文を検出できる
- 要求仕様が自然言語で定義されている稼働中のプロジェクトに対して導入しやすい

考案手法を導入しやすくするために，「成果物を追加・変更するコスト」「手法を習得するコスト」「ツールを用意するコスト」を下げる．「成果物を追加・変更するコスト」を下げるため，既に存在している自然言語で記述されている要求仕様書のみを入力に実行できる手法とする．「手法を習得するコスト」を下げるため，考案手法を未経験の状態からでも，手順の説明を受けただけで実行できる手法とする．「ツールを用意するコスト」を下げるため，フリーソフトウェアで実行できる手法とする．

本研究では，無駄に曖昧表現を検出・修正する工数を増やさないために，すべての曖昧表現を解決の対象とするのではなく，実際に不具合につながった曖昧表現のみを解決の対象とした．また，通常の“上から順に文を読む”リーディング技法では検出が難しい曖昧表現のみを解決の対象とした．

3. 先行研究

3.1. キーワードベースドレビュー

文献[11]で，ドキュメントのあいまいさや不備を検出する手法として，キーワードベースドレビューが提案されている．キーワードベースドレビューは，ドキュメントのあいまいさや不備につながるキーワードを整理し，キーワードが使われている箇所を確認するリーディング技法である．キーワードは観点毎に整理されている．観点とキーワードの例の一部を表 3に示す．

表 3 観点とキーワードの例（文献[11]から引用）

観点	具体的記述の例
数値的表現	から，～，以上，以下
時間的表現	あとで，先に，事前に，後程，常に，いつも，常時，逐次，一定
条件指定表現	とき，でないとき，以外，以内，の際，ではないとき，している（する）最中
不明瞭な表現	ぐらい，とか，みたいな，的には，おそらく，ほぼ，ような，できるだけ
形容表現	多，少，速，遅
指示表現	これ，あれ，それ，どれ，その，この

キーワードベースドレビューは，曖昧な文となり得る修飾語を使用している文を検出するために，有効な手法と考えられる．ただし，キーワードベースドレビューを実施するためには，検索するキーワードを列挙したキーワードリストを，事前に作成する必要がある．

考案手法では，キーワードではなく品詞をもとに，修飾語を使用している曖昧な文を検出する方針とする．観点が同じキーワードには，同じ品詞を関連づけられる可能性がある．例えば，時間的表現のキーワードの多くは「副詞」，形容表現のキーワードは「形容詞」というように，キーワードに品詞を関連づけることができる．あいまいさにつながるキーワードで検索するのではなく，あいまいさにつながる品詞で検索をすることができれば，キーワードを列挙したキーワードリストを作成する必要はなくなると考えた．

3.2. 省略と修飾パターンを用いた用語不一致検証

文献[12]では，要求仕様の「一貫性」に着目し，要求仕様の一貫性検証支援ツールを実現している．ツールには，検証ルールと辞書が組み込まれている．仕様書内で設計要素の用語名が一貫しない状態を，省略パターン，修飾パターン，表記ゆれの 3 つに分類し，それらを検証するための知識が，用語不一致検証ルールとして定義されている．3 つの用語不一致となる概念の説明を表 4に示す．また，用語不一致検証を行うために，設計要素別の識別辞書・表記ゆれ辞書を用意している．

表 4 用語不一致となる概念（文献[12]から引用）

概念	説明
省略	要求仕様書中に定義された設計要素から，形容詞や名詞を省略して，要求仕様書中で利用している場合に，用語不一致とみなす．
修飾	要求仕様書中に定義された設計要素に，形容詞や名詞を修飾して，要求仕様書中で利用している場合に，用語不一致とみなす．
表記ゆれ辞書一致	あらかじめ定義した表記ゆれ辞書に定義された表記ゆれ関係にある用語が，要求仕様書中に利用されている場合に，用語不一致とみなす．

検証ルールと辞書を利用した用語不一致検証は，同義語を抽出するために，有効な手法と考えられる．ただし，検証を実施するためには，ツールに組み込む辞書を，事前に作成する必要がある．また，ツールや辞書の制約により，ツールによる検証結果には，不要な指摘が含まれることがある．そのため，ツールでは用語不一致の候補を抽出し，用語不一致の最終的な判断は人手で行う必要がある．

考案手法では，用語不一致のパターンを活用し，人手で，同義語を抽出する方針とする．省略パターン・修飾パターンという用語不一致の状態を検出する方法は，ツールを使用せずに，人手で同義語の候補を抽出するときにも活用できると考えた．単語の比較をするときに，ドメイン知識が不足していても，省略パターン・修飾パターンに当てはまるかという判定はしやすい．

3.3. 形態素解析と KH Coder

形態素解析[13]は，自然言語処理の基礎となる要素技術である．形態素解析では，文の分かち書きを行い，単語列を同定し，個々の単語の品詞の決定などを行う．

KH Coder[14][15]は，定量テキスト分析のためのフリーソフトウェアである．内部では形態素解析器である茶筌を利用している．非常に手軽なマウス操作によってテキスト型データの分析が行えるという特徴がある．

KH Coder では，テキストファイルを入力に形態素解析器で，図 3と図 2に示すように抽出語のリストと複合語のリストを作成することができる．

形容動詞		副詞可能		未知語		形容詞		副詞		副詞B	
可能	22	場合	24	アクター	25	無い	2	再び	1	かつ	1
異常	2	時間	5	払	7	若い	1	次に	1	また	1
正当	2	結果	3	センサ	1	新しい	1	逐次	1	まだ	1
不当	2	前	2			大きい	1	同時に	1		
切	1	すべて	1								
適切	1	一番	1								
		今	1								
		全て	1								
		途中	1								

図 3 抽出語のリスト

複合語	出現数
販売ボタン	24
当該商品	14
販売可能	11
自動販売機	9
代金投入	8
合計金額	7
返金処理	7

図 2 複合語のリスト

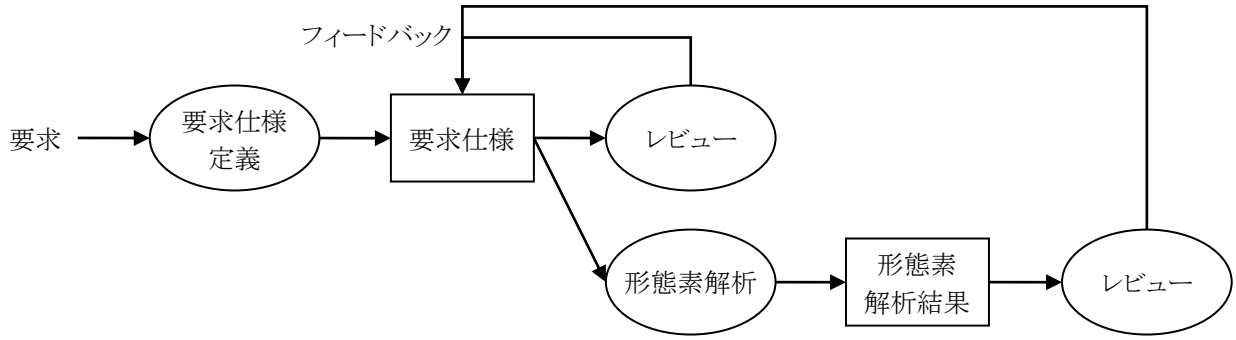


図 4 形態素ベースドレビューの概要

抽出語のリストから，修飾語となり得る品詞の単語を，漏れなく抽出できると考えた．更に，抽出語のリストと複合語のリストから，同義語の判定のために比較をする対象の単語を，漏れなく抽出できると考えた．また，KH Coder は，フリーソフトウェアであり手軽に操作できる特徴により，導入コストが低い．

4. 解決策の提案

4.1. 課題の解決方針

考案手法は，導入コストを下げるために，事前にキーワードリストや辞書を作成しない方針とする．曖昧な文の事例が多数収集できていないと，有効なキーワードリストを作成することは困難である．要求仕様を定義する前の使用する単語が明確になっていない段階で，有効な辞書を作成することは困難である．

考案手法では，修飾語・同義語が使用されている文を抽出するために，形態素解析を活用する．形態素解析器で，品詞毎に単語を抽出し，修飾語となり得る品詞である単語を特定しやすくする．また，形態素解析器で，単語一覧を作成し，同義語を抽出するために実施する単語の比較をしやすくする．

考案手法は，要求仕様の文ではなく，形態素解析器により抽出された単語をもとにレビューをすることで，以下の特徴をもつ手法となる．

- 文脈の力を発動させず，文の曖昧さに気づきやすくする
- 短期記憶に頼らず，単語の比較をすることを可能にする

考案手法は，形態素ベースドレビューと呼ぶ．図 4に形態素ベースドレビューの概要を示す．アドホックレビューに加えて，形態素ベースドレビューを実施することで，アドホックレビューでは見逃しやすい曖昧な文を検出することを可能とする．

本研究では，形態素解析を実施するために，KH Coder を使用した．本稿では，形態素ベースドレビューの手順を KH Coder を使用することを前提として定義する．

4.2. 修飾語の抽出の方法

考案手法では，修飾語は品詞をもとに抽出する．修飾語となり得る品詞は，KH Coder で特定可能な形容詞・副詞・形容動詞・副詞可能の 4 種類とする．この 4 種類の品詞に分類される単語を修飾語と判定する．この品詞に分類された単語をキーワードとして，要求仕様書を検索し，単語の使用箇所を特定する．表 5に，品詞をもとに抽出される修飾語の例を示す．この例に示した単語は，テスト設計コンテスト’14 の ASTER 自動販売機ユースケース仕様書を対象に抽出したものの一部である．

表 5 品詞をもとに抽出される修飾語の例

品詞	単語
形容詞	無い，若い，新しい，大きい
副詞	再び，次に，逐次，同時に
形容動詞	可能，異常，適切
副詞可能	場合，結果，前，すべて，途中

4.3. 同義語の抽出の方法

考案手法では，人手により同義語の抽出を行うため，効率化と抽出漏れ防止の対策が必要となる．

同義語の抽出を効率化するために，比較対象の単語の絞り込みを行う．同義語を抽出するために比較対象と

する単語は，検索で使用する可能性の高い「ドメイン用語」に限定する方針とする．ドメイン用語である可能性の高い単語は，形態素解析器で入力としている標準の辞書に登録されていない単語であると考える．KH Coder では，辞書に登録されていない単語は，「未知語」「複合語」として抽出可能である．同義語を抽出するため比較する単語は，未知語・複合語とする．

同義語の抽出の漏れを防ぐために，同義語を抽出するときの 3 つのポイントを以下に示す．

- 出現頻度の少ない単語を確認する
  同義語は，出現頻度の少ない単語のほうが，修正すべき単語の可能性が高い．誤って他と異なる表現を使ってしまった単語は，出現頻度が低くなる．誤記の場合にも，出現頻度が低くなる．出現頻度が高い単語は，表現が統一されているということであり，修正対象とはならない．
- ソートして比較する
  未知語・複合語をソートすることで，似ている単語が並ぶことになり，比較しやすくなる．例えば，「合計額」と「合計金額」が並ぶことで，同義語を抽出しやすくなる．
- 用語不一致パターンに当てはまるか判定する
  複合語に対して，省略パターンまたは修飾パターンの用語不一致になっていないか確認する．複合語の一部の語を省略した単語が使用されていないかを確認する．複合語の末尾に同じ名詞が使われている単語を比較対象とする．例えば，「懸賞あたりランプ」という複合語があった場合，「あたりランプ」「ランプ」という単語が使用されていないかを確認する．

4.4. 形態素ベースドレビューの手順

形態素ベースドレビューは，「1.形態素解析」「2.修飾語の抽出」「3.同義語の抽出」「4.曖昧な文の検出」の 4 ステップで実施する．各ステップで実施することを次に示す．

1. 形態素解析
   要求仕様を入力に，形態素解析を実施し，抽出語のリストと複合語のリストを作成する．
2. 修飾語の抽出
   抽出語のリストから形容詞・副詞・形容動詞・副詞可能に分類される単語を抽出する．
3. 同義語の抽出
   抽出語のリストから比較対象の未知語を抽出する．複合語のリストから比較対象の複合語を抽出する．未知語・複合語として抽出した単語を比較し，同義語であるか判定する．
4. 曖昧な文の検出
   ステップ 2，ステップ 3 で抽出した修飾語・同義語の使用箇所を要求仕様から特定し，修正要否を判断する．

5. 解決策の評価

5.1. 評価方法

考案手法「形態素ベースドレビュー」を以下の 3 つの観点で評価する．

- アドホックレビューでは見逃した曖昧な文を検出できるか
- 不具合を誘発した曖昧な文を検出できるか
- 稼働中のプロジェクトに対して導入しやすいか

考案手法を評価するために，以下の 3 種類の実験を行う．

A) 複数の異なる要求仕様書に対して実行
B) 不具合を誘発した曖昧な文に対して実行
C) 考案手法の未経験者が実行

3 つの実験の方法を以下に示す．形態素解析を実施するために使用したツールは，KH Coder 3 である．選択した形態素解析器は茶筌である．

A) 複数の異なる要求仕様書に対して実行
   アドホックレビュー済みの要求仕様書を対象として，考案手法を実行し，修飾語・同義語を使用している曖昧な文が検出できるかを確認する．表 6に計測するデータを示す．

表 6 計測データの種類（実験 A）

データ	説明
仕様定義者数	仕様定義者の数． 複数人か一人を示す．
総抽出語数	抽出語のリストに出力される各抽出語の出現頻度の合計．
修飾語数	考案手法で検出した曖昧な文で使用していた修飾語の数．
同義語数	考案手法で検出した曖昧な文で使用していた同義語の数．

確認対象は，4 つの異なる案件の要求仕様書とする．形態素ベースドレビューの実施者は，仕様定義者自身とする．

B) 不具合を誘発した曖昧な文に対して実行

不具合の原因となった曖昧な文を含む要求仕様書を対象として，考案手法を実行し，不具合の原因となった箇所が検出できるかを確認する．

2 つの不具合事例を対象に確認をする．同義語により不具合を誘発した要求仕様書は用意できなかったため，事例はどちらも修飾語により不具合を誘発した要求仕様書である．

C) 考案手法の未経験者が実行

形態素ベースドレビューを未経験の状態から，手順の説明を受けるだけで，実行できるかを確認する．形態素解析は事前に実施しておき，抽出語のリストと複合語のリストを入力に，「2.修飾語の抽出」「3.同義語の抽出」「4.曖昧な文の検出」ができるかを確認する．表 7に計測するデータを示す．

4 人の被験者を対象に確認をする．1 名のみ形態素ベースドレビューの経験者，残りの 3 名は未経験者とし，結果を比較する．全員，同じ要求仕様書を対象にレビューを実施する．「ASTER 自動販売機ユースケース仕様書」をレビュー対象の要求仕様書とする．

表 7 計測データの種類（実験 C）

データ	説明
レビュー時間	考案手法の実施時間．形態素解析の作業時間は除く．
修飾語数	考案手法で検出した曖昧な文で使用していた修飾語の数．
同義語数	考案手法で検出した曖昧な文で使用していた同義語の数．

5.2. 評価結果

各実験の結果を以下に示す．

A) 複数の異なる要求仕様書に対して実行

表 8に，4 つの要求仕様書毎に，形態素ベースドレビューの実行結果を示す．

すべての要求仕様書で，修飾語または同義語を使用している曖昧な文を検出した．検出した問題は，すべてアドホックレビューでは見逃したものであった．仕様定義者や形態素数が増えるほど，検出される問題が増える傾向があった．曖昧な文で使用していた修飾語は，すべて「形容動詞」であった．

表 8 形態素ベースドレビューの結果（仕様毎）

ID	仕様定義者数	総抽出語数	修飾語数	同義語数
01	複数人	5,611	5	10
02	一人	1,447	2	0
03	一人	490	2	0
04	一人	237	1	0

B) 不具合を誘発した曖昧な文に対して実行

2 つの不具合事例で，不具合の原因となった曖昧な文を検出することができた．不具合を誘発した修飾語は，どちらも「形容動詞」であった．

C) 考案手法の未経験者が実行

表 9に，被験者毎に，形態素ベースドレビューの実行結果を示す．入力とした要求仕様書の総抽出語数は，1,036 である．また，表 10に実験に参加した被験者の特徴を示す．

被験者全員が，手順の説明を 1 時間程度受けるだけで，考案手法を実行できた．考案手法の未経験者であっても，全員が 1 時間のレビューで，曖昧な文を検出することができた．ただし，検出した問題については差がある．考案手法の経験者が検出した問題の一部を考案手法の未経験者は見逃していた．要求仕様のレビュー経験がない場合，修飾語を使用している曖昧な文を検出できない傾向があった．

表 9 形態素ベースドレビューの結果（人毎）

ID	レビュー時間（H）	修飾語数	同義語数
ア	0.5	1	4
イ	1.0	1	2
ウ	1.0	0	6
エ	1.0	0	2

表 10 被験者の特徴

ID	考案手法の経験有無	要求仕様レビューの経験有無	ソフト開発の経験年数
ア	あり	あり	10 年以上
イ	なし	あり	2 年
ウ	なし	なし	1 年
エ	なし	なし	1 年

実験結果をもとに，5.1評価方法に示した 3 つの観点で考案手法を評価する．評価結果を以下に示す．

- 実験 A の結果から，アドホックレビューでは見逃した曖昧な文を考案手法で検出できることが確認できた．
- 実験 B の結果から，実際に不具合を誘発した曖昧な文を考案手法で検出できることが確認できた．
- 実験 A の結果から，既存の要求仕様書を入力に，成果物の追加・変更をせずに，考案手法が実行できることが確認できた．また，実験 C の結果から，考案手法を未経験の状態からでも，手順の説明を受けただけで実行できることが確認できた．

この結果から，形態素ベースドレビューは，2.2課題提起で示した3つの要求を満たす手法であると考えられる．ただし，形態素ベースドレビューでも，他のリーディング技法と同様に，検出できる問題はレビューアの能力に依存する．形態素ベースドレビューのレビューアには，要求仕様のレビューの経験が必要となる．要求仕様のレビューの経験が不足している場合，曖昧な文の判断の効率が落ちたり，曖昧な文を見逃したりする可能性が高まる．

5.3. 結果の考察

形態素ベースドレビューは，文ではなく単語を確認する．これにより，アドホックレビューより，修飾語・同義語を使用している曖昧な文を検出しやすくなった．これは，形態素ベースドレビューがもつ以下の 2 つの特徴が要因と考えられる．

- 文脈の力を発動させず，文の曖昧さに気づきやすくする
- 短期記憶に頼らず，単語の比較をすることが可能である

形態素ベースドレビューは，導入コストが低いため，稼働中のプロジェクトでも導入できた．キーワードリストや辞書を事前に用意せず実行できる点が，導入コストを下げられた大きな要因である．

また，実験の結果，考案手法の改善が必要な点を 2 点明らかにした．1 つ目は「修飾語の抽出手順」，2 つ目は「同義語の抽出手順」に関する改善点である．

修飾語を使用している曖昧な文を検出するための手順は改善の余地がある．具体的には，不具合につながる可能性のない文を無駄に検出することがないように，確認対象の品詞の見直しが必要である．形態素ベースドレビューで問題ありと判断された文で使用していた修飾語の品詞は，形容動詞のみであった．また，不具合の原因となっていた文で使用していた修飾語の品詞も，形容動詞のみであった．実験で入力とした要求仕様書において，形容動詞の単語は，名詞＋「な」の形で表現され，登場することが多かった．定性的な語[3]である形容詞や副詞等は，考案手法を使わずに，一つの文を普通に読むだけで，曖昧と判断できている可能性が高い．

同義語の抽出には時間がかかるため，手順の改善を検討する必要がある．同義語を抽出する工数を削減することで，手法の導入のしやすさを更に向上させることができる．要求仕様書で使用している単語が多くなればなるほど，単語を比較する回数が増え，同義語を抽出する工数が増える．同義語の抽出は，すべての要求仕様の定義完了後にまとめて 1 回実施のではなく，要求仕様の定義途中に繰り返し複数回実施するほうが，工数が削減できる可能性がある．また，評価結果から，“仕様定義者が一人”かつ“使用している単語が少ない”という条件では，同義語が使用される可能性は低いことがわかった．このような条件を満たす場合は，同義語のチェックを省略しても不具合にはつながらず，チェックを省略することで工数を削減できる可能性がある．

6. おわりに

6.1. まとめ

実際に不具合を誘発させた要求仕様の表現を分析し，「修飾語」または「同義語」を使用している文が，不具合を誘発する要因となることを明らかにした．

本研究では，要求仕様書から修飾語・同義語が使用されている曖昧な文を検出するために，「形態素ベースドレビュー」を考案した．評価の結果，形態素ベースドレビューは，以下の条件を満たす手法であることが確認できた．

- “上から順に文を読む”アドホックレビューで見逃した要求仕様の曖昧な文を検出できる
- 実際に不具合を誘発した要求仕様の曖昧な文を検出できる
- 要求仕様が自然言語で定義されている稼働中のプロジェクトに対して導入しやすい

要求仕様書に対して，形態素ベースドレビューを実施することで，「仕様の誤解釈」「検索での関連仕様の抽出漏れ」を防止し，不具合を減らす効果が期待できる．更に，形態素ベースドレビューと Domain Word Modeling を組み合わせることで，過去に不具合を誘発したすべての種類の曖昧な文を検出することが期待できる．

文献[16]では，推敲の効果を上げるために，多様な読み方で多様な推敲を行うことが良い方法であるということが，推敲のコツとして示されている．リーディング技法に対しても，推敲のコツを当てはめることができる．文献[17]でも，要求仕様書の特性に着目し，レビュー手法毎に，検出しやすい欠陥の特徴を明らかにする研究がなされている．要求仕様に対するレビューで，欠陥を見逃さないためには，一つのリーディング技法だけを使用するのではなく，複数のリーディング技法を使用することが有効と思われる．形態素ベースドレビューは完全性や妥当性の検証はしにくいため，形態素ベースドレビューも他のリーディング技法に加えて実行することが有効と思われる．

文献[18][19]では，形態素解析を利用したレビュー支援手法が提案されている．同じ形態素を入力とするレビュー手法を組み合わせることでも，効果を高められる可能性がある．

6.2. 今後の課題

今後は，形態素ベースドレビューの実開発での活用を継続させながら，有効性と効率性を更に高めるために，以下の取り組みを行う．

- 手法適用の効果の確認

  手法適用による効果の大きさは確認できていない．形態素ベースドレビューで防止できる不具合の種類については確認できているが，防止できる不具合の量については確認できていない．

  形態素ベースドレビューの適用前後で，不具合の傾向がどう変化するかを分析する．また，効果確認を通して，形態素ベースドレビューにより効果を得るための前提条件も明らかにする．
- 手法の適用対象の拡大

  自分のプロジェクトで定義した要求仕様書だけでなく，要求仕様定義の入力となる文書に対しても，受け入れ確認の位置づけで形態素ベースドレビューを適用し，効果を確認する．

  また，例えばマニュアルなど要求仕様書以外の文書に対しても形態素ベースドレビューを適用し，効果を確認する．
- 修飾語の抽出手順の改善

  曖昧な文の検出の漏れと誤りを減らすために，不具合事例をもとに，曖昧な修飾語を検出するための判定条件を見直す．具体的には，確認対象とする品詞を見直す．評価結果から，確認対象は形容動詞のみでも効果があると考えられる．
- 同義語の抽出手順の改善

  同義語の抽出にかかる工数を削減するために，手順の改善を検討する．評価の結果をもとに検討した同義語の抽出手順の改善案を図 5に示す．改善した手順では，形態素ベースドレビュー後に，形態素解析の結果から，ドメイン用語の辞書を作り，作成した辞書を次の形態素ベースドレビューの入力とする．そして，確認対象の単語の数が多くならないうちに，小さく継続的にこのサイクルを回す．これにより，同義語の抽出作業の効率を向上させられる可能性が高い．また，要求仕様定義時に，辞書を参照すれば，同義語を使用することが予防され，同義語のチェック自体を省略できる可能性もある．

  更に，Word2Vec[20]などの自然言語処理を活用し，辞書を作成せずに同義語の抽出を自動化することを検討する．

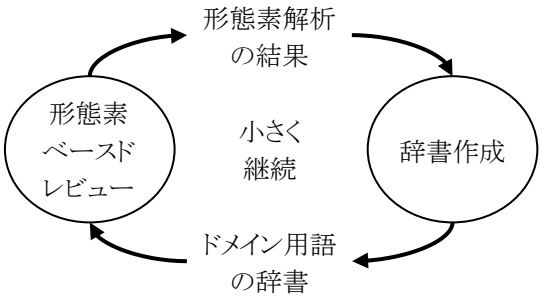


図 5 同義語の抽出手順の改善案

謝辞

本研究にご参加いただいた 株式会社デンソークリエイトとSky株式会社のプロジェクトメンバに感謝の意を表する.

本研究のきっかけをいただいた ソニー株式会社 栗田太郎氏, 国立情報学研究所 石川冬樹氏, 第 35 年度ソフトウェア品質管理研究会研究コース5のメンバに感謝の意を表する.

参考文献

[1] 厳密な仕様記述WG委員, “厳密な仕様記述入門”, 独立行政法人情報処理推進機構, 2013

[2] 山本修一郎, “要求工学 連載第87回 要求文の曖昧さの摘出法”, https://www.bcm.co.jp/site/youkyu/youkyu87.html, 2005

[3] 阿部圭一, “情報伝達型の日本語文章に現れるあいまい表現の類型化とその改善例”, 情報処理学会デジタルプラクティス Vol.5 No.1 p.70-79, 2014

[4] 誉田直美, “レビュー技術動向”, ソフトウェア品質シンポジウム 2017, 2017

[5] 柏原一雄, 長井亘, 不破慎之介, 林香織, 石川冬樹, 栗田太郎, “要求仕様の誤解釈を検出する Domain Word Modeling の提案”, ソフトウェア・シンポジウム 2020, 2020

[6] 原正幸, 松本裕治, “ipadic version 2.7.0 ユーザーズマニュアル”, 奈良先端科学技術大学院大学, 2003

[7] 井上昇, 高橋宏季, 位野木万里, “振る舞い用語の同義語に着目したあいまいな要求仕様の検証手法の提案”, 情報処理学会第81回全国大会講演論文集, 2019

[8] 中島千壽, 高橋宏季, 位野木万里, “要求仕様のあいまいさを解消するためのプロセスフレームワークの提案”, 情報処理学会第 82 回全国大会, 2020

[9] 西林克彦, “わかったつもり 読解力がつかない本当の原因”, 光文社新書, 2005

[10] 村田真樹, 内元清貴, 馬青, 井佐原均, “日本語文と英語文における統語構造認識とマジカルナンバー7±2”, 自然言語処理 6 巻 7 号 p.61-71, 1999

[11] 河野哲也, 猪塚修, 藤森麻紀子, 本間周二, 茂中義典, “キーワードベースドレビュードキュメントのあいまいさや不備に着目したレビュー手法ー”, ソフトウェアテストシンポジウム東京 2010, 2010

[12] 位野木万里, 近藤公久, “省略と修飾パターンを用いた用語不一致検証による要求仕様の一貫性検証支援ツールの実現と適用評価”, コンピュータソフトウェア 35 巻 3 号 p.109-127, 2018

[13] 保田明夫, “形態素解析と分かち書き処理”, テキスト・マイニング研究会 テキスト型データのマイニング, p.145-180, https://www.wordminer.org/wp-content/uploads/2013/04/63_30.pdf, 2006

[14] 樋口耕一, KH Coder 3, http://khc.sourceforge.net/, 2019

[15] 樋口耕一, “フリーソフトウェア「KH Coder」による計量テキスト分析: 手軽なマウス操作による分析からプラグイン作成まで”, 研究報告人文科学とコンピュータ(CH) Vol.2015-CH-107 No.9 p.1-2, 2015

[16] 結城浩, “数学文章作法 推敲編”, ちくま学芸文庫, 2015

[17] 岡本博幸, 内山敬太, 鈴木彩子, 高橋一仁, 西山茂, 野中誠, “要求仕様書の特性に着目した個人レビュー手法の実験的評価”, ソフトウェア品質管理研究会 第 20 年度 (2004 年度) 分科会成果報告, 2005

[18] Takehiro Wakabayashi, Shuji Morisaki, Norimitsu Kasai, Noritoshi Atsumi, Shuichiro Yamamoto, “Tool Supported Detection of Omissions by Comparing Words between Requirements and Design Document”, Journal of Information Processing 28 巻 p.136-149, 2020

[19] Shuji Morisaki, Masato Nishiguchi, Tetsuya Yonemitsu, Atsushi Motoyama, “A Method to Remove Extraneous Words in Defect Log by Using Common Vocabulary”, コンピュータ ソフトウェア 37 巻 2 号 p.2_120-2_126, 2020

[20] 伴凌太, 高橋宏季, 位野木万里, “word2vecを用いた同義語辞書自動作成手法の提案と適用評価”, 情報処理学会第 81 回全国大会講演論文集, 2019

自然言語処理的アプローチによるクラス図関連線の予測

井原　輝人
奈良工業高等専門学校　専攻科 *
akiteruto@gmail.com

内田 眞司
奈良工業高等専門学校
uchida@info.nara-k.ac.jp

飯田　元
奈良先端科学技術大学院大学
iida@itc.naist.jp

要旨

ソフトウェア品質特性 *[1]* の一つに理解性がある．理解性とは，成果物を読み手が容易に理解できるかを指す．*UML* 図における理解性を定量的に表現できれば，評価値に基づいた設計品質の評価や向上に繋がると期待される．しかし，理解性は個人の感覚や経験則に依存する部分が大きく，定量化する実用的な手法は存在していない．中村 *[2]* らは読み手の理解性が図の構成要素個々の妥当性判断に基づいて行なわれていることに注目し，妥当性を定量的に評価するメトリクスの定式化を試みた．しかし，*UML* 図においては個々の図形要素が複雑に関連付けられ得るため，理解性に影響を与える要素とそれらの関係性をすべて定式化することは極めて困難である．

よって我々は同様の目的に対して機械学習を用いた妥当性の定量化を提案する．本研究ではその第１段階として，関連線の種類の予測を行うモデルを作成した．被験者実験を実施し，モデルの予測結果と，被験者の判断結果を比較したところ，高い精度で一致したことから，予測モデルは個々の関連線の人間による関連線に対する妥当性評価を摸倣可能であるといえ提案手法で関連線の種類において理解性の定量化が可能であることが明らかとなった．関連線の種類や命名の妥当性を予測モデルに基づいて *UML* 図の理解性について部分的に定量的な評価可能となることを示した．今後，他の要素も含めた予測モデルへと拡張をすることで *UML* 図 *(*クラス図）全体の理解性評価手法に結びつけられると期待される．

*2021 年 4 月より奈良先端科学技術大学院大学

1. 序論

ソフトウェアの開発工程において，開発者間の相互理解を目的とした統一モデリング言語である UML が幅広く利用されている．特に，開発の上流工程では，UML を用いた設計や分析モデルの記述が行われ，その開発成果物を基に以降の実装や検証が進められる．一般に設計文書の品質が以降の工程に与える影響は大きく，UML 図の品質評価はソフトウェア品質の確保に重要であると言える．

品質特性 [1] とはソフトウェアに求められる要求を分類したものである．品質特性の 1 つである理解性とは成果物を読み手が容易に理解できるかを表す．開発者間の相互理解を目的とする UML 図において理解性は重要な品質特性であり，当然理解性の高い UML 図は良い UML 図とされる．

UML 図における理解性を定量的に表現できれば以下のことが可能となる．

- リファクタリング効果の可視化や，理解性のボトルネック検出によるリファクタリング補助
- 理解性を報酬とした強化学習モデルによるリファクタリングの自動化
- 機械学習を用いた UML 図の自動生成における特徴量としての活用

しかし，理解性は個人の経験則や主観に依存する部分が大きく，定量的に評価する実用的な手法は存在していない．中村ら [2] は UML 図の 1 つであるクラス図に対

して理解性の定量化に取り組んだ．彼らは読み手が図の構成要素に対して妥当性評価を行うことで理解性判断を行っていることに注目し，妥当性をメトリクスとして定式化することで定量化を試みた．彼らはクラス図内に存在するクラス名の類似度が高いほど，読み手はクラス名の間の関係を強く感じ，そのようなクラス名の関係が強いクラスがクラス図上で近くに存在していると理解性が高いという仮説を立て，その仮説に基づいて理解性メトリクス CRSU を定義した．クラス名の類似度計測には類義語辞書である WordNet[5] を用いた．評価実験としてメトリクス値と専門家の理解性評価の相関を調べたところ 0.64 となり，仮説が正しいことを明らかにした．しかし，中村らのメトリクスでは汎化や集約といった関連線の種類を考慮していないという問題点がある．読み手は関連線の種類から何かしらの情報を得ていると考えられ，考慮しないことで情報の欠損が発生していると推測できる．そこで筆者らは先行研究と同様に WordNet の「上位語-下位語」,「全体語-部分語」のそれぞれの概念を活用し，汎化，集約に対応したメトリクスを提唱した [4]．被験者実験を行い，被験者の理解性評価とメトリクスの評価値を比較したところ，メトリクスの精度向上を確認する事ができた．このことから関連線の種類が読み手の理解性判断に影響を与えていることが明らかとなった．

しかし，一方でいくつかの手法の問題点が明らかとなった．

1 つ目は，有意な評価実験が困難であることである．現在，クラス図のデータセットは存在しておらず統計的考察を行うために十分な数を用意することは困難であり，どちらの先行研究の結果も有意性について疑問が残る形となった．

2 つ目は自然言語処理技術として WordNet のみを用いていることである．単語の類似度計測が可能な自然言語処理技術は他にも多く存在しておりそれらを導入，併用することで精度向上が可能であると考えられる．

3 つ目は定式化による高精度な妥当性定量化が困難なことである．どちらの先行研究も関連要因の洗い出し，加重平均による重み付け，尺度の調整等をすべて考慮し式に落とし込むことによる定式化を行うことで妥当性の定量化を試みている．

我々の提案する手法では，教師あり学習を用いてクラス図内の任意の要素の予測モデルを構築する．予測モデルでは対象のクラス図の構成要素の一つを空欄に置き換え，周辺の情報から空欄部分に当てはまる要素を予測する．予測モデルを用いた妥当性評価方法について図 1 に例を示す．

妥当性評価では，はじめに対象のクラス図の構成要素の一つを空欄に置き換え，次に予測モデルを用いて空欄部分に当てはまる要素候補とそれぞれの確率を予測する．最後に予測結果と元の構成要素を比較し，予測結果の確率に応じて妥当性評価値を算出する．用いた予測モデルが人間の妥当性判断を模倣していることを証明することで妥当性を保証できる．

機械学習モデルを用いることで定式化では困難であった各要因の重み設定並びに尺度の調整が可能となり，妥当性評価の精度向上が期待される．

図 1. クラス名予測モデルを使った妥当性評価の例

2. 提案手法

本研究では研究の初段階として，関連線の種類を予測する予測モデルを作成する．予測モデルは教師あり学習を用いて生成する．図 2 に学習の全体図を示す．説明変数は対象の関連線の両側に位置するクラス名 2 つ，目的変数は関連線の種類と設定する．関係線は 2 つのクラス間の関係性を表現するため，関連線が結ぶ 2 つのクラス名の関係性を複数の自然言語処理技術を用いて特徴量として数値化し，予測モデルの入力として用いることで精度向上を試みる．

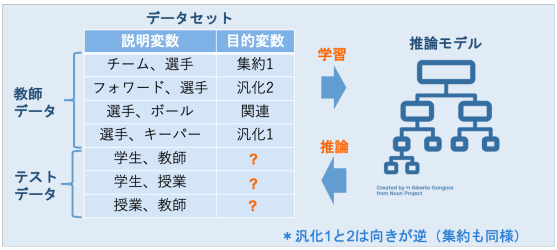


図 2. 教師あり学習の全体図

第 1 章でも述べたとおりこの予測モデルを用いて妥当性評価を行う．関連線予測モデルを用いた妥当性評価の

例を図 3 に示す．妥当性評価ではまず評価対象となる関連線の両側に位置するクラス名から予測モデルを用いて関連線の予測を行い，関連線候補とその確率を算出する．そして元の関連線と予測結果を照らし合わせ，確率を妥当性評価値として受け取る．

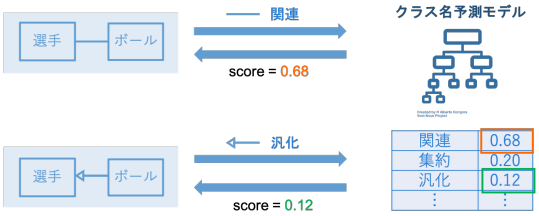


図 3. 関連線予測モデルを用いた妥当性評価の例

また，提案手法のメリットとして 2 つの先行研究が抱えていたその他の問題点も解決可能である．1 つ目は先行研究では限られたデータを用いて評価実験を行っており，データの有意性の部分で問題があった．一方，提案手法では分析対象をクラス図全体から関連線へと置き換えているため，ある程度の数が手作業でも確保可能となる．2 つ目は先行研究では自然言語処理技術として WordNet のみを用いており，他手法を適用，併用することで精度向上の余地があるという問題があった．提案手法では教師あり学習を用いており，特徴量として他の手法を複数組み合わせることができる．よって，特徴量を組み合わせ，複合的に妥当性評価を行うことができるとともに，それらの特徴量に対する予測モデルの学習結果を見ることで，どの特徴量がより予測に役立っているかを判断する事ができる．

3. 予測モデルの作成

この章では予測モデルの作成過程について説明する．

3.1. データセット作成

現在，有力な公開データセットは存在していないため参考書や web ページといった複数のドメイン上に存在するクラス図から，クラス名とその間の関連線を抽出することでデータセットを手作業で作成した．最終的に汎化 1，集約 1，関連をラベルとした 260 データを用意した．ただし，汎化や集約は方向性を持った関連線であり，その向きによって関係性が異なるため，向きに応じて別のラベルを与える必要がある．よって汎化 1 と集約 1 をそのまま反転させ，それぞれ汎化 2 と集約 2 として追加した．つまり，ラベルに含まれる数字がそれぞれの向きを表している．これにより 425 個のデータセットが完成した．データセットの関連線の種類ごとの分布を表 1 に示す．

表 1. 関連線の分布

関連線の種類	ラベル	データ数
関連	k	95
汎化 1	h1	83
汎化 2	h2	83
集約 1	s1	82
集約 2	s2	82

次にデータセットのドメインを分類した際の各データ数を表 2 に示す．

表 2. データセットにおけるドメインの大まかな分類

ドメイン	ラベル	データ数
サンプルサイト	creately	100
web	web	67
過去の実験データ	me	68
参考書	book	156
OSS	swing_game	34

このようにデータセットではドメインを可能な限り分散させることで予測モデルの汎化性能を確保した．また，データセットの言語を英語で統一するために，日本語で記述されたデータは翻訳サービス DeepL[12] を用いて英語に翻訳した．さらに語句は表記ゆれの観点からすべて名詞単数形に変換した．その後，データセット内に存在する単語に対して WordNet の synset 形式に変換するための辞書 CSV ファイルを手作業で作成した．

3.2. 特徴量作成

関係線はクラス間の関係性を表現するものであるため，関連線の両端にある 2 つのクラス名の関係性を複数の自然言語処理技術を用いて特徴量として数値化する必要がある．本研究で作成した特徴量一覧を表 3 に示す．

表 3. 特徴量一覧

ラベル名	説明	使用コーパス	次元数
sim_by_mcg	is-a群の類似度	Microsoft Concept Graph	1
is_a_by_mcg	is-a関係かの評価値	Microsoft Concept Graph	1
lexvec_sim	ベクトルのcos類似度	LexVec(word2vec)	1
lexvec_diff	ベクトルの差分	LexVec(word2vec)	300
wiki2vec_sim	ベクトルのcos類似度	Wiki2Vec(word2vec)	1
wiki2vec_diff	ベクトルの差分	Wiki2Vec(word2vec)	300
wn_hu_ave	先行メトリクス（汎化関係）	WordNet	1
wn_pu_ave	先行メトリクス（集約関係）	WordNet	1
wn_sim_path	pathsimilarity（関連関係）	WordNet	1
wn_sim_wup	WordNetを用いた類似度	WordNet	1
bert_sim	説明文ベクトルのcos類似度	BERT/WordNet	1
bert_diff	説明文ベクトルの差分	BERT/WordNet	768
is_include_word	同じ単語を含むか	なし	1

表 3 においてオレンジ色は先行研究で用いられていた理解性メトリクスを改良したものとなっている．これらの特徴量をデータセットに対して算出した．欠損値が発生する特徴量に対しては最頻値補間を行い対応した．以下の節でそれぞれの特徴量について説明を行う．

3.2.1. Microsoft Concept Graph を用いた特徴量

MCG は Microsoft 社が行っている自然言語処理プロジェクトである [6]．彼らは Probase という自然言語コーパスを所持しており，その活用方法として，自然言語処理を用いて物事の概念化をゴールに研究を行っている．その研究過程での成果物として彼らは単語間の is-a 関係を定量的に評価する事のできるサービスを提供している．具体的には特定の英単語を与えるとその単語と is-a 関係を持つ単語群と各単語の Score が返される．この Score は is-a 関係に強さを表している．MCG は is-a 関係がベースとなっているため，同様に is-a 関係がベースとなっている汎化の分類に有用であることが期待される．まず 1 つ目の特徴量として，sim_by_mcg を定義した．sim_by_mcg では 2 つの単語に対して，それぞれ単語の is-a 単語群における共通単語を調べることで関係性を表現する．アルゴリズムの説明にあたって核となるコードを図 4 に示す．

まず，_get_dict_form_ms_api メソッドを用いて word_a と is_a 関係を持つ上位 30 個とその評価値を辞書形式

```
def word_sim(word_a, word_b):
    di_a = MCG._get_dict_from_ms_api(word_a)
    di_b = MCG._get_dict_from_ms_api(word_b)
    eva = 0
    if di_a == {} or di_b == {}:
        return np.nan
    for key, value in di_a.items():
        if key in di_b.keys():
            eva += 10 * (value + di_b[key])
    return np.average(eva)
```

図 4. sim_by_mcg のコード

(key:単語,value:評価値) で MCG の API にアクセスして取得する．word_b に対しても同様の処理を行う．MCG 内で word_a もしくは word_b が存在していない場合は欠損値として扱うため NaN を返す．どちらも存在した場合 word_a の辞書と word_b の辞書で共通している単語のそれぞれの評価値の合計に 10 をかけたものの合計を返す．したがって共通する単語が多ければ多いほど値は大きくなり，一方で共通単語が存在しなかった場合は 0 を返す．

複合語の場合は複合語 A と複合語 B 間の単語のすべての組み合わせに対してこの値を算出し平均を取る．

同じく MCG をベースとした特徴量として is_a_by_mcg を定義した．この特徴量は単純に MCG 上で 2 つの単語が is-a 関係でつながっているかどうか，つながっていた場合は評価値がどの程度かを確認する．前述の sim_by_mcg との差として，is_a_by_mcg は 0 の値をとる傾向があり，MCG の単純な評価値を直接用いるためノイズが紛れ込みくい利点があると考える．アルゴリズムの説明のため，図 5 に特徴量算出の核となる部分のコードを示す．

まず，_get_dict_form_ms_api メソッドを用いて word_a と is_a 関係を持つ上位 30 個とその評価値を辞書形式 (key:単語,value:評価値) で取り出す．word_a が MCG に存在していない場合は欠損値として扱うため NaN を返す．存在した場合は先程の辞書内に word_b があればその評価値に 100 を掛けたもの，そうでなければ 0 を返す．

複合語の場合は複合語 A と複合語 B 間の単語のすべての組み合わせに対してこの値を算出し平均を取る．

```
def eva_is_a(word_a, word_b):
    di_a = MCG._get_dict_from_ms_api(word_a)
    if di_a == {}:
        return np.nan
    if word_b in di_a.keys():
        return di_a[word_b] * 100
    else:
        return 0
```

図 5. is_a_by_mcg のコード

3.2.2. Word2Vec 系の技術を用いた特徴量

Word2Vec[7] は単語をベクトル化する Word Embeddings 手法の一つであり，その汎用性の高さから多くの派生手法が研究されている．本研究ではその中でも LexVec[8] と Wiki2Vec[9] について取り扱う．LexVec は Word2Vec の後継モデルであり，純粋により高い精度が期待できる．Wiki2Vec は Wikipedia をコーパスとしており，ほかの Word2Vec 系の特徴量とは異なる特徴量として機能することが期待できる．本研究ではタイプの異なる 2 つの Word2Vec 派生の手法を用い，その差を比較することで，Word2Vec の手法が異なることで本タスクに影響があるのかどうか考察を行う．

一般的に Word2Vec を用いて単語間の類似性を見る際は cos 類似度を用いるため，LexVec を用いた cos 類似度による特徴量 lexvec_sim，Wiki2Vec を用いた cos 類似度による特徴量 wiki2vec_sim を作成した．基本的には cos 類似度をそのまま返すが，モデルに登録されていない未知語であった場合は欠損値として NaN を返す．クラス名が複合語の場合は複合語を構成する複数の単語のベクトルの加算平均を取りそのベクトルを用いて最終的にもう一方のクラス名のベクトルとの cos 類似度を求める．

今回のタスクである関連線の分類では向きを考慮する必要があるが前述の cos 類似度では向きは考慮する事ができない．よって cos 類似度に加えてベクトル差分による特徴量 lexvec_dff，wiki2vec_diff を作成した．ベクトル差分では引き算であるため向きを考慮する事ができる．さらに cos 類似度では多次元を 1 次元に圧縮することになるため情報が抜け落ちてしまいやすいが，ベクトル差分では情報量を保ったまま関係性を表現できることに加え，分類に適した形で次元圧縮を併用する事ができる．基本は word_a から word_b を引いたベクトル差分をそのまま返す．モデルの未知語であった場合は欠損値として NaN を返す．複合語の場合は複合語を構成する複数の単語のベクトルの加算平均を取り，そのベクトルを用いて最終的にもう一方のクラス名のベクトルとのベクトル差分を求める．

3.2.3. BERT による特徴量

本来 BERT[10] は文章のベクトル化に特化したアルゴリズムであるため単語のベクトル化に用いることは一般的ではない．しかし，近年の NPL 分野でデファクトスタンダードな BERT と WordNet を組み合わせることで精度向上が期待できる．

WordNet の概念を表す各 Synset には定義文が定義されている．この文章を事前学習済み BERT モデルに投げることで定義文ベクトルを取得できる．定義文は単語をその単語を使わずに説明しているわけであるため単語と定義文ベクトルは意味的に等しいはずである．

この考えのもと word_a，word_b それぞれで定義文ベクトルを生成し，Word2Vec 系と同様に bert_sim は 2 つのベクトル間の cos 類似度，bert_diff は 2 つのベクトル間の差分を特徴量として定義した．複合語の場合は複合語を構成する複数の単語のベクトルの加算平均を取ったものを定義文ベクトルとして用いる．

3.2.4. 理解性メトリクスを用いた特徴量

先行研究で定義された理解性メトリクスを特徴量として用いることで 2 章で述べたとおり，今回新たに作成する特徴量との比較を行う事ができる．理解性メトリクスは木構造上に単語が格納された類義語辞書である WordNet をベースに作られている．WordNet では上位語，下位語という概念が定義されており 2 つの間には is-a 関係が成り立つ．木構造はこの上位語，下位語の関係性で形成されたネットワークである．

理解性メトリクスは関連の評価を行う CWsim メトリクス，汎化の評価を行う HU メトリクス，集約の評価を行う PU メトリクスに分解できる．よってそれぞれを特徴量として定義した．

まず，汎化関係の理解性評価を行う HU メトリクスから特徴量 wn_hu_ave を生成した．アルゴリズムの説明のため，図 6 に特徴量算出の核となる部分のコードを示す．

```
def hu(word_a, word_b):
    wn_a = WordNet.get_wn(word_a)
    wn_b = WordNet.get_wn(word_b)
    if wn_a == wn_b:
        return 1.0
    coms = wn_a.lowest_common_hypernyms(wn_b)
    stack = []
    for com in coms:
        if wn_a == com:
            return 1.0
        stack.append(wn_a.path_similarity(com))
    return max(stack)
```

図 6. wn_hu_ave のコード

まず，get_wn でそれぞれの単語を Synset 形式に変換する．Synset 形式とは WordNet に

おける同音異義語を集めた一つの概念を表す．wn_a.lowest_common_hypernyms(wn_b) はもっとも下位に存在する共通の上位語を返す．複合語の場合は複合語 A と複合語 B 間の単語のすべての組み合わせに対してこの値を算出し平均を取る．

次に，集約関係の理解性評価を行う PU メトリクスから特徴量 wn_pu_ave を生成した．アルゴリズムの説明のため，図 7 に特徴量算出の核となる部分のコードを示す．

```
def pu(word_a, word_b):
    part_wn = WordNet.get_wn(word_a)
    wn_b = WordNet.get_wn(word_b)
    max_pu = 0.0
    if part_wn.part_holonyms() != []:
        part_holo_wns = part_wn.part_holonyms()
        for part_holo_wn in part_holo_wns:
            pu = part_holo_wn.path_similarity(wn_b)
            max_pu = max(max_pu, pu)
    return max_pu
```

図 7. wn_pu_ave のコード

まず，get_wn でそれぞれの単語を Synset 形式に変換する．part_wn.part_holonyms() で word_a の全体語のリストを取得している．この全体語のリストと word_b の path_similarity を算出し，最大値を返している．全体語が１つもない場合は 0 を返す．

最後に関連の理解性評価を行う CWsim メトリクスから wn_sim_path ならびに wn_sim_wup ２つの特徴量を生成した．CWsim は WordNet が提供する単語類似度算出手法の PathSimilarity 法をベースに作成されている．しかし，WordNet では他にもいくつかの単語類似度算出手法が定義されており，当然各手法で表現できる特徴も異なる．よってそれぞれの手法において，各手法間の相関値を算出し，PathSimilarity 法と最も相関が低くなった WuPalmer 法をベースとして wm_sim_wup ２つの特徴量を生成した．

3.2.5 その他の特徴量

is_include_word は単純に２つの語句間で共通している単語があるかどうかを確かめる特徴量である．共通していれば 1 を返し，そうでなければ 0 を返す．他特徴量の多くはクラス名間で共通単語が出てきた場合，値が著しく高くなる傾向があり，それによりモデルが単語間の関係性ではなく共通単語に対して学習を行うリスクがある．よって is_include_word をモデルが重視していれば，上のリスクが高いことがわかり，そうでなければ想定通り単語間の関係性を学習できている事がわかる．

3.3 次元圧縮

現在のデータセットは 1377 次元の特徴量を持つがその中でベクトル系の特徴量が占める割合が表 3 で示したとおり非常に大きい．これによって次の２つの問題が発生するリスクがある．１つは次元の呪いによって学習がうまく行かないリスクがあることである．そしてもう１つは特徴量の評価が困難になることである．XGBoost で Feature Importance を算出した際，１つの特徴量が多くの次元を持つと次元ごとに評価値が算出され，他特徴量との比較が複雑化する．これらの理由から本実験ではデータセット内のベクトル系特徴量に対してのみそれぞれ主成分分析 (PCA)[11] を行うことで次元圧縮を行う．

各ベクトル特徴量ごとに１つ PCA のモデルを用意し次元圧縮を行う（以後これら３つのモデルの集合体をまとめて次元圧縮モデルと呼ぶ）．各 PCA モデルの圧縮次元数についてはデータセットすべてのデータでの累積寄与率を算出しそれぞれ決定した．

3.4 学習

学習推論は層化 90 分割交差検証法をベースとして行った．

予測モデルに用いる数理モデルとして XGBoost を選択した．理由について説明する．一般的に用いられる NN 系の数理モデルではパラメータチューニングに依存する部分が大きく，過学習のリスクが高く，さらに，説明性が低いという問題点があり，本研究には適切でないと判断した．よって次に精度が出やすいとされる Boosting アルゴリズムの中でも，最も古く，多くの知見がある XGBoost を使用する．チューニングの結果，パラメータは木の数を 300，木の最大深さを 17，学習率を 0.25 と設定した．学習，推論の全体図を図 8 に示す．

図について解説する．まず層化 90 交差検証法に従いデータをラベル分布が等しくなるように 90 等分する．その中から１つをテストデータとし，その他をトレーニングデータとする．この際，テストデータの正解ラベルは予め記録しておく．まず次元圧縮モデルに対してトレーニングデータを用いて学習を行う．この際の圧縮次元数は 3.3 節で定めたものを使う．学習した次元圧縮モデルに対してトレーニングデータ並びに，テストデータを

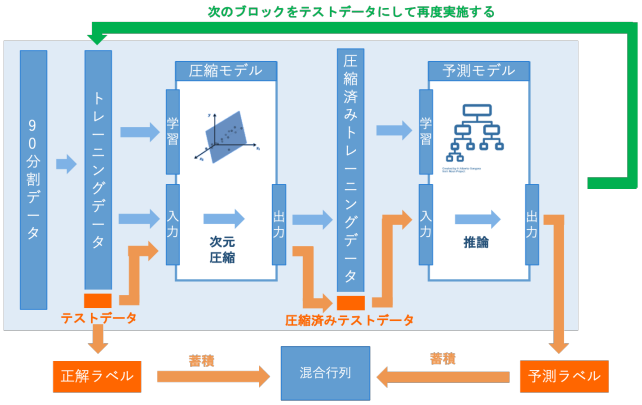


図 8. 学習，推論の全体図

入力し，圧縮済みトレーニングデータ，圧縮済みテストデータを受け取る．次に圧縮済みトレーニングデータを使って予測モデルの学習を行う．その後学習した予測モデルに対して圧縮済みテストデータを入力し，予測ラベルを受け取る．そして予め記録しておいた正解ラベルと予測ラベルを混合行列に追加する．ここまで完了したらテストデータを次のブロックに変更し，同様の工程を行う．これを 90 ブロックすべてで行う．これによって混合行列には正解ラベルと予測ラベルが 425 の全データ分格納される．

結果として得られた混合行列を図 9 に示す．

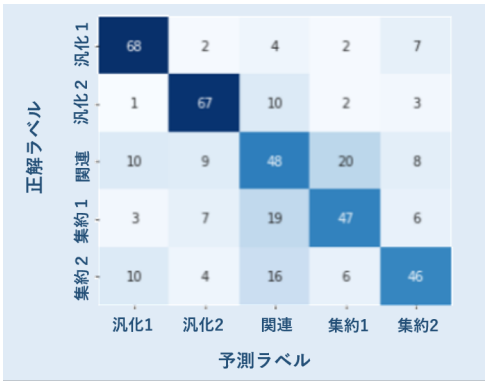


図 9. 予測モデルの混合行列

図のセル内の数字はデータ数を表している．ここから Accuracy を算出すると 0.65 となった．

4. 評価実験

本章では作成した予測モデル評価のために行った評価実験について述べる．

4.1. 実験内容

本校情報工学科 5 年生 1 名，システム創成工学専攻情報システムコース 1 年生 1 名，同コース 2 年生 4 名で被験者実験を行った．彼らはいずれも授業でクラス図について学習を行っている．ただし，前 2 人は半期の授業 1 つ，後ろ 4 人は半期の授業 2 つで学習時間が異なっている．実験は提案手法と同様にデータセットに含まれるクラス名のペアから関連線の種類 (汎化 1，汎化 2，関連，集約 1，集約 2）を予測し回答するという形式で実施した．時間は無制限とし，クラス図についての知識が曖昧になっている可能性や，データセットに英語が含まれるため，個人の英語能力に依存する可能性を考慮しデータセットのドメインを除く辞書や web の利用を許可した．

この評価実験の目的は，予測モデルと可能な限り同等の環境で人間のパフォーマンスを計測し，予測モデルと比較することで，予測モデルの精度を評価することである．人間のスコアが予測モデルと同等程度であれば予測モデルは人間の妥当性判断を模倣できているといえ，提案手法は妥当性評価が可能であると導ける．

ただし以下の点で被験者と予測モデルは異なっている．

- 予測モデルはクラス図についての学習先がデータセットのみであるが，被験者は授業を通して学習を行っている．
- 予測モデルは事前学習から各単語の前提知識を構築するが，人間は，日常生活やこれまでの豊富な知見から前提知識を構築する．
- 予測モデルは推論時，1 つのデータのみから推論を行うが，人間は前後データからの情報を推論に活用できる．(1 つのクラス図から複数のデータを抽出しているため)

4.2. 実験結果

まず実験結果として各被験者の Accuracy を表 4 に示す．

表 4 より被験者全体での平均は 0.58 となった．しかし，Accuracy に注目してみると被験者 1〜3 と 4〜6 で大きく 2 グループに分割することができる．accucary が低かった 1〜3 のグループについては学習時間が少ない本科 5 年生，専攻科 1 年生が含まれることからクラス図の理解度がもう 1 つのグループに比べて低かったことが

表 4. 関連線の分布

被験者番号	学年	Accuracy
1	本科 5 年	0.40
2	専攻科 1 年	0.40
3	専攻科 2 年	0.48
4	専攻科 2 年	0.69
5	専攻科 2 年	0.72
6	専攻科 2 年	0.76

読み取れる．よって 4〜6 のグループをクラス図に対して十分な理解がある習熟グループとして Accuracy の平均を算出すると 0.72 となった．

ここの結果からいえることはある程度クラス図に対して習熟している学生でも 7 割程度の精度が限界であるということである．この原因として，もともとクラス図はプログラマによって一意に定まらないケースが存在する曖昧性の高いタスクであることに加え，本実験では 2 クラスのみという情報量が削減された環境（被験者は前後から推測可能ではある）だったことにより，より一層，答えを定めるのが困難なタスクとなってしまったと考える．

また習熟グループについてそれぞれ混合行列を出力し，それらの平均をとったものを図 10 に示す．

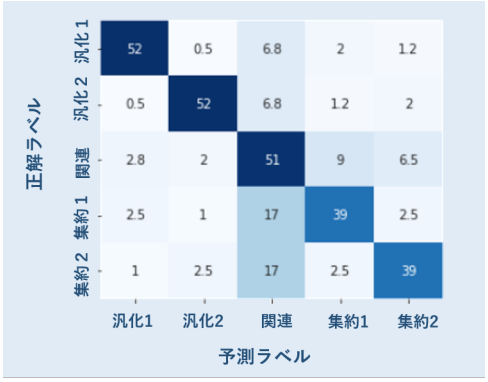


図 10. 習熟グループの結果から算出した混合行列

5. 考察

この章では考察について述べる．

5.1. 予測モデルの有意性についての考察

作成した予測モデルの有意性について考察を行う．我々は以下の 3 つの根拠から予測モデルが人間の妥当性評価を模倣できていると主張する．

- 幅広いドメイン・異なる種類の図からデータセットを作成しており汎用性が高いこと
- 被験者の正答率が 0.7 程度あり，データセットの信頼性が保証されていること
- 被験者の回答結果と予測モデルの予測結果間での相関が高いこと

それぞれについて説明を行う．1 つ目について説明する．作成したデータセットでは予測モデルに十分な汎化性を持たせるため，幅広いドメインから抽出しており，かつ図の種類も分析用の図や教科書の例題，コードからの生成等様々である．例を出すとテトリスや，野球，ショッピングシステムと言ったように全く類似性のないクラス図が対象となっている．さらにこれらは特定の作者によって作成されたものではなく，いずれも異なる作者によって異なる目的を持って作成されたクラス図である．このような分散の大きくかつ少量のデータセットで精度を出すのは非常に困難なことである．その点を考慮すると本研究の予測モデルの精度は十分に高いといえる．さらに，同様の理由から未知のデータに対しても精度を落とさず予測が可能であるといえる．

2 つ目について説明する．上で述べたとおり，データセットの一部は Web サイト等のフリーソースから抽出しており，理解性が低く外れ値となるデータが含まれている可能性がある．外れ値が多く含まれたデータを学習してしまうと予測モデルが外れ値に対して過学習を起こす可能性が高くなる．しかし，評価実験での被験者の正答率が 0.7 程度であることから外れ値データは 0.3 未満であることが推測できる．よって上で述べた可能性は存在せず，正しく学習ができていると推測できる．

3 つ目について説明する．予測モデルと，被験者のそれぞれの混合行列，図 9 と図 10 を比較するとかなり類似している．さらに，習熟グループの 3 人の答え (subject_ans1-3) とモデルの予測ラベル (model_prediction) 間でクラメール連関係数を計算した結果を図 11 に示す．

図 11 を見てみるといずれも 0.5 を超えていることから仮説は棄却され，非独立性が証明されたため本提案手法で作成した各特徴量が人間の判断基準を十分にとらえているといえる．

これらの根拠から，予測モデルが人間の妥当性評価を模倣できていると主張する．

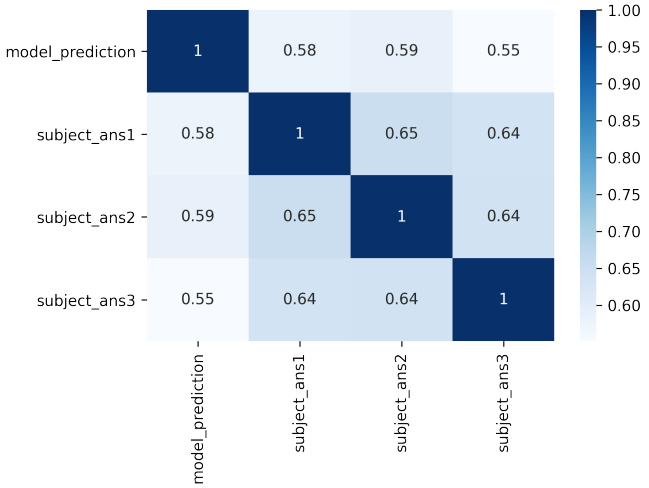


図 11. 各被験者とモデル間のクラメール関連係数

5.2. 各特徴量についての考察

各特徴量について考察を行う．まず特徴量間の相関を示す相関行列を図 12 に例を示す．ただし 1 次元のみの比較であるため diff 系は含まない．

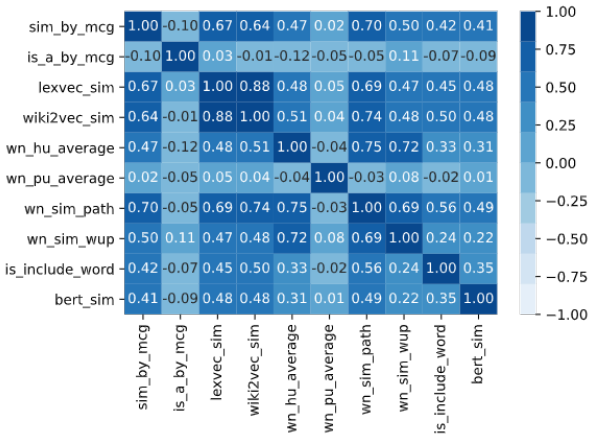


図 12. 相関行列

図 12 を見ると，事前に相関が高くなりすぎる特徴量に関しては除外しているため基本的にはそこまで高いものは存在していないことが確認できる．これは多重共線性の観点から学習にとって好ましいことである．ただし wiki2vec_sim と lexvec_sim は wn_sim_wup と wn_sim_path については相関が比較的高い値となっている．これらのことから，3.2 節で述べたように Word2Vec 系，WordNet の Similarity 系どちらも手法間の差は本タスクにおいては発生しにくいと判断する事ができる．

各特徴量を比較するために全データを用いて XGBoost を学習させ，Feature Importance を算出した．ただし，ベクトル系の特徴量については 1 次元に圧縮を行っている．結果を図 13 に例を示す．

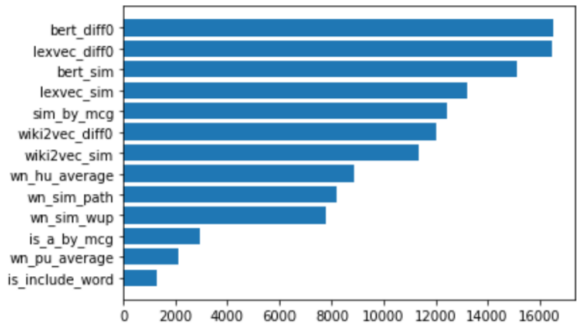


図 13. Feature Importance

全体的に見てみると，BERT や LexVec といった特徴量が高い値を出しており，一方で WordNet 等が低い値となっている．これは概ね一般的な自然言語処理タスクの結果の傾向と類似している．

まず，ベクトル系特徴量のベクトル差分と cos 類似度について比較するとベクトル差分が高いスコアであることから，やはり 3.2 節で述べたとおり，cos 類似度によってデータの特徴が圧縮されてしまうため，純粋にベクトル差分を取る方法が有効であると言うことが明らかとなった．また，今回作成した WordNet と BERT を組み合わせた特徴量である bert_diff 並びに bert_sim の精度が高く，単語という限られた情報量かで有用な手法であることが示された．

また MCG について見てみると sim_by_mcg がベクトル系についで高い値となった．MCG は未だマイナーな手法であるがこのように自然言語処理タスクの特徴量として用いる事ができるという可能性を明らかにする事ができた．ただし is_a_by_mcg については低い値となり，この原因としては殆どの単語の組み合わせで特徴量が 0 となってしまったからであると推測する．

Word2Vec 系の特徴量について見てみると，LexVec がわずかに良い値であるという結果となった Wiki2Vec について，データセット内で 12 データ分未登録単語があったため欠損値が発生していた．よってこの差が精度の差として現れたといえる．

また先行研究で用いられた WordNet 系の特徴量につ

いて見てみると，どれも下位の方に位置しており，従来研究よりも高い精度での予測が可能であると分かる．

6. まとめ・今後の展望

本研究ではクラス図の理解性の定量化を最終的な目的とし，先行研究と同様に妥当性の定量化を試みた．本研究では予測モデルを用いた妥当性の定量化手法を提案した．手法提案の初段階として，本研究では関連線の種類の予測を行う予測モデルを作成した．被験者実験を実施し，予測モデルの予測結果と，被験者の判断結果を比較したところ，予測モデルは人間の妥当性判断を模倣可能であることが明らかとなった．このことから提案手法を用いることで関連線の妥当性評価が可能であることが示された．今後の展望としては以下の通りである．

- クラス名等のクラス図の他要素に対して提案手法を適用する．
- 予測モデルを関連線の自動生成の目的で使用する．
- 本研究では WordNet の synset との紐づけを手作業で実施しているが，仕様書の文章等から紐づけを自動化する．

参考文献

[1] Jorgen Boegh, A New Standard for Quality Requirements, IEEE Software, pp.20-27, January 2008.

[2] 中村悠人, 鷲崎弘宜, 深澤良彰，単語の類似度を用いた UML クラス図の理解性の測定と評価，ソフトウェアエンジニアリング最前線， 情報処理学会 SE シンポジウム 2009，pp.165-172，September 2009.

[3] 高橋直久, 丸山勝久:ソフトウェア工学, 森北出版株式会社, 2010

[4] 井原輝人, 内田眞司 , 単語類似度を用いた UML クラス図理解性メトリクスの改良, 信学技報, vol. 119, no. 246, SS2019-24, pp. 65-70, October 2019

[5] WordNet - Princeton University Cognitive Science Laborator, http://wordnet.princeton.edu/, 2019/9/23 閲覧

[6] Zhongyuan Wang and Haixun Wang, Understanding Short Texts, in the Association for Computational Linguistics (ACL) (Tutorial), August 2016.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Efficient Estimation of Word Representations in Vector Space, arXiv:1301.3781v3 [cs.CL] 7, September 2013

[8] A. Salle, A. Villavicencio and M. Idiart, Matrix factorization using window sampling and negative sampling for improved word representations, In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pp. 419–424, 2016

[9] Ikuya Yamada, Hiroyuki Shindo, Hideaki Takeda, Yoshiyasu Takefuji,Joint Learning of the Embedding of Words and Entities for Named Entity Disambiguation, Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning,pp 250-259, 2016

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, arXiv preprint arXiv:1810.04805v2, May 2019.

[11] Sebastian Raschka, Vahid Mirjalili, ［第 2 版］Python 機械学習プログラミング 達人データサイエンティストによる理論と実践，インプレスブックス，2018

[12] DeepL, https://www.deepl.com/ja/translator, 2020/12/2 閲覧

クラス名における命名バグの検出手法

有村 徳崇
大分大学
v1653005@oita-u.ac.jp

紙名 哲生
大分大学
kamina@oita-u.ac.jp

要旨

オブジェクト指向におけるクラスは多くの場面で再利用される．しかし，クラス名がその機能を正しくユーザに伝えられなければ，ユーザはそのクラスの機能を理解するのに失敗し，再利用性を損なう．本研究では，クラス名に命名バグがあるかどうかをそのクラスの設計者に伝えるための検出手法の開発を試みた．まず，あるクラスにおいてその意味内容とクラス名からユーザが読み取る意味内容が異なるならばクラス名に「命名バグ」があるとして，クラス名における命名バグを「クラスメンバにあるものをクラス名が表現できていない場合」と「クラス名が示すものがクラスメンバに存在しない場合」に大別する．次に，クラス名とクラスメンバの特徴をソースコードから抽出し，それを予め準備された「クラスカテゴリ」（クラス名とクラスメンバの特徴に関するルール）の集まりと順次比較する．抽出されたクラス名とクラスメンバの特徴と、クラスカテゴリで示したルールのうちのどれかの間に矛盾があった場合，命名バグとしてユーザに報告する．オープンソースソフトウェア（*OSS*）をもとにデザインパターンに関するいくつかのクラスカテゴリを作成し，その妥当性を評価した．

1. はじめに

オブジェクト指向プログラミングは現在も多くのソフトウェア開発で用いられている．オブジェクト指向的開発環境は様々なソフトウェアシステムに共通のニーズを満たす再利用可能なクラスを提供し，オブジェクト指向で開発されたクラスは多くの場面で再利用される．しかし，クラス名がその機能を正しくユーザに伝えられなければ，ユーザにそのクラスを理解するための負担がかかり，再利用性を損なう．既存の命名規則は命名に統一性を与えるが，意味のある適切な命名を保証しない．

本研究では，クラス名に命名バグがあるかどうかをそのクラスの設計者に伝えるための検出手法の開発を試みた．まず，あるクラスにおいてその意味内容とクラス名からユーザが読み取る意味内容が異なるならばクラス名に「命名バグ」があるとして，クラス名における命名バグを「クラスメンバにあるものをクラス名が表現できていない場合」と「クラス名が示すものがクラスメンバに存在しない場合」に大別する．次に，クラス名とクラスメンバの特徴をソースコードから抽出し，それを予め準備された「クラスカテゴリ」（クラス名とクラスメンバの特徴に関するルール）の集まりと順次比較する．そして，抽出されたクラス名とクラスメンバの特徴とクラスカテゴリで示したルールのうちのどれかの間に矛盾があった場合，命名バグとしてユーザに報告する．

本研究では，本手法の有効性を検証するため，以下の二つの適用実験を行った．

1. 実際のオープンソースソフトウェア（OSS）のソースコードを用いて，作成したクラスカテゴリの妥当性を検証する実験
2. 作成・検証されたクラスカテゴリを用いて命名バグを検出する実験

1. については，デザインパターンに関するいくつかのクラスカテゴリと実際のクラス定義において，クラス名に含まれる単語とクラス要素に含まれる単語に高い頻度で重複のあることが確認できた．2. については，1. で作成した Visitor パターンのクラスカテゴリを用いて，簡単なサンプルプログラムにおいて命名バグの自動検出に成功した．

本論文の構成は以下のとおりである．2 節で関連研究

について紹介する．3 節で本研究で提案する命名バグの検出手法について詳説する．4 節で実装した命名バグ検出ツールを紹介し，それを用いて行った適用実験の結果について報告する．最後に，5 節で結論を述べる．

2. 関連研究

2.1. 命名バグに関する研究

Høst らは，メソッド名の命名バグを検出する手法を実現した [6]．この手法ではメソッド名を抽象化し，その抽象化したメソッド名が満たすべき機能を求め，それをルールとして運用することで命名バグを検出する．Binkley らは，品詞タグ付けの技術を用いて不適切な命名がなされているクラスメンバの検出を行った（例えば動詞で始まるフィールド名など）[4]．これらの手法はいずれも，メソッド名には通常動作を表す品詞が用いられるなどの経験則を用いて，その動作を抽象化して機能やルールの分類を行う．一方でクラス名の場合，それが表すのは通常名詞であるため，これらの手法をそのまま本研究に適用することはできない．そこで本研究では，クラス名とクラスメンバに基づいたカテゴリを考え，クラスがカテゴリに属する場合にクラス名とクラスの内容が同時に満たすべきことを記述し，それをルールとして運用することで命名バグを検出する．

Arnaoudova らは，命名に関するアンチパターン [5] を定義し（例えば “set” メソッドが値を返すなど），ソースコード中からそのアンチパターンに従うコードを検出するツールを提案した [2]．アンチパターンとは，否定的な結果に導くような一般的にみられる開発方法を記述した形式のことである．ここでは，メソッドやフィールドの命名についてのアンチパターンは議論されているものの，クラス名についての議論はない．また，この検出には予め定義されたアンチパターンが必要になる．本研究でも予め定義したクラスカテゴリが必要になるが，このカテゴリはクラスメンバに出現する用語の頻度などの比較的単純な統計情報に基づいており，こうしたアンチパターンと比べて自動生成が容易になると考えられる．

2.2. 命名を支援する手法に関する研究

Allamanis らはメソッドやクラスの命名を推薦する手法を提案している [1]．命名の推薦は一般的に局所的な情報のみを用いて行うことは難しく，ここではソースコード中における命名を対象とした確率ニューラル言語モデル [3] に基づいて，ソースコード中の文脈における命名法を学習する．このような推薦と命名バグの検出は相互に補間するものであり，本研究では，命名の推薦は行わない代わりに適切でない命名が行われているクラスを検出する．

```
public class TestObj{
    public boolean visit(Element e) {
        // ...
    }
}
```

図 1. クラスメンバにあるものをクラス名が表現できていない命名バグ

3. 本研究のアプローチ

上述したとおり，メソッド名やフィールド名における命名バグの検出についてはこれまでに一定の成果があがっているものの，クラス名については筆者らの知る限りほとんど取り組まれてきていない．一つには，クラスはメソッドやフィールドよりも抽象度が高く，品詞やアンチパターンなどを用いた分類が難しいからであると考えられる．しかしながら，クラス名はその機能を正しくユーザに伝える重要な役割を持っており，不適切な命名はクラスの再利用性を損うことに繋がる．

本研究では，クラス名に命名バグがあるかどうかを，そのクラスの設計者に伝えるための検出手法について提案する．

3.1. クラスにおける命名バグ

クラス名とそのクラスの定義は対応するべきである．このことを「クラス名がある特徴的な要素を持つならばそれに応じた要素を持つクラスメンバを持つ」,「クラスメンバがある特徴的な要素を持つならばそれに応じた要素を持つクラス名を持つ」という仮定に落とし込み，これらの仮定に基づき命名バグを検出する．

ここでクラス名における命名バグを「クラスメンバにあるものをクラス名が表現できていない場合」と「クラス名が示すものがクラスメンバに存在しない場合」に大

```
public class TestVisitor{
    public void print() {
        // ...
   }
}
```

図 2. クラス名が示すものがクラスメンバに存在しない命名バグ

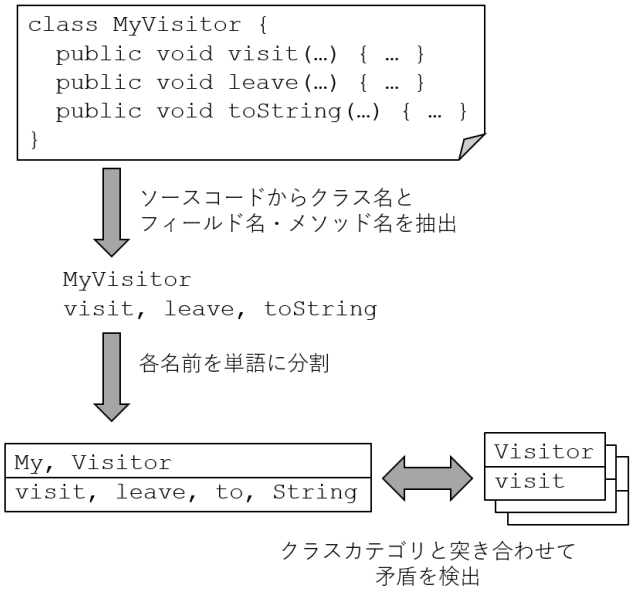


図 3. 提案手法の概要

別する．図 1 は，クラスメンバにあるものをクラス名が表現できていない場合を示した例である．この例では visit することがこのクラス名からはわからない．デザインパターンの一つである Visitor パターンを用いているならば Visitor という単語を含むべきである．visit という操作をされるとしてもこのクラス名では visit という特徴的な操作を想定できず不適当である．図 2 は，クラス名が示すものがクラスメンバに存在しない場合を示した例である．この例では，クラス名に Visitor とあるがこのクラスは visit しない．

3.2. アプローチの概要

図 3 に提案手法の概要を示す．まず，検査対象となるソースコードを構文解析することで各クラスのクラス名と，全てのフィールド名とメソッド名を得る．次に，これらクラス名，フィールド名，メソッド名それぞれを単語に分割する．単語への分割の際には，Java で用いられることの多いキャメルケースで書かれた名前を対象に，大文字ごとに区切る方法を採用した．大文字ごとに区切るだけでは対応できない名前（例えば “XML” のように大文字のみから構成される単語を含む場合）については，今回は除外し考慮しないことにした．なお，ここではクラス名を構成する単語を**命名要素**，フィールド名やメソッド名を構成する単語をそのクラスの**特徴的要素**と呼ぶことにする．

こうして得られた命名要素及び特徴的要素を，別途用意した複数のクラスカテゴリと突き合わせることで命名バグの有無を検査する．別の言い方をすると，クラスカテゴリとは，クラスの命名に関して「ある命名要素を含むクラスのフィールド名やメソッド名には，関連する特徴的要素を持たなければならない」というルールを記述したものであり，そのルールに則ってクラス名が付けられているかどうかを検査する．

命名要素	Visitor
特徴的要素	visit, Visit

図 4. Visitor パターンが期待する命名要素と特徴的要素を表すクラスカテゴリの例

3.3. クラスカテゴリ

本研究では，クラス名とクラスメンバの命名に関する対応関係をまとめるためにクラスをカテゴライズする．各カテゴリには，ある命名要素に対して期待される特徴的要素が情報としてまとめられており，それをファイルに保存する．これは一種の単純な命名パターンであると見なすこともでき，先述のアンチパターンを用いた方法とは逆の方法を用いている．つまり，クラス名においては明確なアンチパターンを探すことが難しいため，逆に「あるべきパターン」に矛盾する命名を検出するという方法をとっている．

例として，クラスカテゴリ「Visitor」を考える（図 4）．デザインパターンの Visitor パターンを採用したクラスにはクラス名の単語として Visitor，クラスメンバにはメソッドとして visit という特徴的な要素がある．クラスカテゴリ「Visitor」はこれらを内容とする．

3.4. ルールの適用

ソースコードを構文解析することによって得られた各クラスの命名要素と特徴的要素の組を，複数のクラスカテゴリと突き合わせ，両者の間の矛盾の有無を調べることによって命名バグの有無を検査する．つまり，クラス名の要素とクラスメンバの要素の両方で，クラスカテゴリに設定された特徴的要素に当たる要素が存在する（あるいは要素の両方でそのクラスカテゴリに設定された特徴的要素に当たる要素が一切存在しない）場合にはそのクラスメンバを持つクラスのクラス名はそのカテゴリにおいて命名バグに当たらないと判断し，それ以外の場合において命名バグを報告する．

なお本研究では，メソッド名は適切に付けられていることを想定している．メソッド名は適切に付けられているにもかかわらずクラス名が不適切な場合としては，例えばプログラムの改修を進めていくうちにクラスの役割りが変わっていき、名前が適切でなくなるケースなどが有り得る．

4. 命名バグ検出ツールの実装

本手法の有効性を検証するため，実際に命名バグを検出するツールを実装した．実装したツールは主に以下の二つの機能に分けられる．

- コードベースにあるソースコードをもとに，クラスカテゴリの生成を助ける機能
- クラスカテゴリを用いて，検査対象のソースコードから命名バグを検出する機能

なお，今回対象としたプログラミング言語は Java である．構文解析には統合開発環境の Eclipse に付随する JDT の ASTParser クラスを用いた．

4.1. クラスカテゴリの作成

クラスカテゴリの生成を補助するため，クラス名の特定の単語とクラスメンバ名の特定の単語がどれほどの頻度で重複して用いられているかを調べるツールを実装した．このツールは，ユーザが設定したクラスカテゴリが妥当なものかどうかを調べるため，コードベースにあるソースコードとクラスカテゴリの比較を行う．例えば，コードベース上で，あるクラスカテゴリが表す命名要素と特徴的要素（メンバ名で用いられる単語群）の間に高い頻度での重複を確認することができれば，そのクラスカテゴリが示す命名規則は広く用いられているということを意味するので，そのクラスカテゴリは妥当であると考えることができる．

命名要素	Iterator, Iterable
特徴的要素	List, list, Next, next, Remove, remove, Has, has

図 5. Iterator パターンのカテゴリ

今回は，図 4 で示した Visitor パターンに関するカテゴリのほか，Iterator パターンに関するカテゴリを準備した（図 5）．コードベースには，GitHub 上にあり，Java で書かれていて，評価数が多くソースコード量の多い物として，elasticsearch[1]，google/guava[2]，TheAlgorithms/-Java[3]，RxJava[4]を用いた．コードベースに含まれる Java ソースコードのファイル数はおよそ 19800 個になる．またコードベース中には，単語 “Visitor” を含むクラスは 41 個，単語 “Iterator” を含むクラスの数は 214 個，単語 “Iterable” を含むクラスは 88 個存在した．

これらのカテゴリをコードベース上で試した結果を表 6 に示す．この表からは，命名要素 Visitor と Iterator の二つにおいて，カテゴリ名とクラスメンバの特徴的要素が高い頻度で重複していることが確認できる．Visitor については，一つのクラス中に単語 “visit” を含むメソッドが複数存在した．よって，Visitor と Iterator についてはこれらのクラスカテゴリは世の中の命名規則をよく反映していると言える．一方 Iterable については，ある程度の重複が確認できるものの，必ずしもこれらの特徴的要素を持たないクラスも多く存在する．

また，コードベース中でこれらのクラスカテゴリに従わなかったクラス名（つまり本手法で命名バグだと検出されるケース）は，“Visitor” を含むクラス名で 7 個，“Iterator” を含むクラス名で 53 個，“Iterable” を含むクラス名で 53 個存在した．これらの中には，例えば抽象的な Visitor を継承しているクラスなどが多く存在しており，必ずしも命名バグであるとは言えない．現時点では，本手法はクラスの構文解析から得られる情報しか用いておらず，スーパークラスの情報などを活用できてい

[1] https://github.com/elastic/elasticsearch
[2] https://github.com/google/guava
[3] https://github.com/TheAlgorithms/Java
[4] https://github.com/ReactiveX/RxJava

命名要素と特徴的要素の対応	マッチング回数
Visitor-visit	657
Iterator-next	187
Iterator-Next	144
Iterator-remove	73
Iterator-List	46
Iterator-has	105
Iterator-Remove	26
Iterator-Has	12
Iterable-Next	13
Iterable-next	4
Iterable-remove	7
Iterable-Remove	1
Iterable-Has	3
Iterable-has	3
Iterable-List	4

図 6. コードベース上におけるクラスカテゴリのマッチング結果

ない．今後はクラス階層解析などの結果を用いて本手法を拡張したうえで，改めて検出精度の評価などを行っていく予定である．

4.2. 命名バグの検出

人工的に命名バグを含むプログラムを作り、その命名バグを検出できるかを確認する実験を行った．デザインパターンの Visitor パターンに基づいたカテゴリを設定し，例 1，例 2 のように想定した二種の命名バグを含むクラスの Java ソースコードをそれぞれ作成した．これらを対象に命名バグ検出を行なった．クラスカテゴリファイル「Visitor.csv」にクラス名の特徴的要素「Visitor」とクラスメンバの特徴的要素「visit」を設定した．命名バグを含むクラスは「testVisitor というクラス名で visit という単語を持つメンバを持たないクラスの Java ソースコード」と「visit というメソッドを持つ Visitor という単語を含まないクラス名のクラスの Java ソースコード」の二つを用意した．

これらクラスカテゴリファイルと Java ソースコードを開発したツールに与え，命名バグ検出処理を実行した．結果は「クラス名 testVisitor が特徴的要素 Visitor を持つが，そのカテゴリ Visitor に応じる要素を持つクラスメンバがない」という命名バグと「クラスメンバ visit が特徴的要素 visit を持つが，そのカテゴリ Visitor に応じる要素をクラス名が持たない」という命名バグを検出することができた．

4.3. 妥当性への脅威

本研究で実施した命名バグの検出は、簡単なプログラムを対象に、意図的に挿入された命名バグが実際に検出できるかどうかを確認したものである．実際にユーザが書いたプログラム上でこうした命名バグの検出が役に立つのかどうかについては、今後実証的な実験を重ねて確認していく必要がある．

今回設定したクラスカテゴリについては，その妥当性についてコードベース上で評価を行なっている．このコードベースは評価数やソースコード量の多い OSS を用いて作られており，実際に広く用いられているソフトウェアである．したがってソースコードの品質の高さにおいては一定以上のレベルにあると考えられるため，評価結果については信頼のおけるものであると筆者らは考えている．しかしながら，対象としたソフトウェアは４種類と未だ少なく，それぞれ異なった分野から選択されてはいるものの，例えば GUI アプリケーションを含んでいないなど，分野にはまだ偏りがあるため，コードベースをさらに充実させた上でさらなる検証が必要であると考えられる．

また，今回設定したクラスカテゴリはいずれもデザインパターンに関するもので，命名方法について一定の慣習があるものから選ばれている．一方で，こうした明らかな慣習は存在しないものの，不適切な命名によってプログラマにクラスの機能について誤解を与えるケースは様々であると考えられる．例えば，ドメイン分析などからわかる命名規則がある場合，それに従わないような命名を検出できるかなど，様々なケースに適用できる一般性を本研究で示したアプローチが持っているかどうかについて，今後検討を重ねていく必要がある．

なお，本研究はメソッドは適切に命名されているという前提に立っている．メソッド名が不適切につけられているようなケースでは，本研究のアプローチは必ずしも成功しない．ただし，メソッド名の命名バグについてはメソッド本体を調べることによって検出できる場合があることが知られており [6]，こうした手法と組み合わせることによって提案手法の有用性を高めることができる

と考えられる．

5. 結論

オブジェクト指向で開発されたクラスは多くの場面で再利用され，クラス名がその機能を正しくユーザーに伝えられなければ、ユーザーにそのクラスを理解するための負担がかかり，再利用性を損なう．そこでクラスの意味内容とそのクラス名から読み取れる意味内容が異なる場合にそのクラス名には「命名バグ」があるとし，クラス名とクラスメンバの関連性から命名ルールを導き出すことを目指した．

本研究では，クラス名とクラスメンバは対応関係にあると仮定し，この対応関係をクラスカテゴリとしてまとめ，このクラスカテゴリをルールとして運用してクラス名の命名バグ検出するツールを開発した．実験の結果，少なくとも一部のデザインパターンについてはこうした対応関係について確認することができ，それを用いて命名バグを実際に検出できることが示された．

5.1. 今後の課題

このツールを活用するには十分な数のクラスカテゴリファイルが必要である．しかし現段階ではごくわずかな数しか用意できていない．そのため，実験で示された結果も限定的なものとなっている．今後はより多くのコードベースを用いて様々なクラスカテゴリを設定し，実際のユーザのコードに適用させるなどの実験を重ねていく必要がある．そのための一つの方策として，クラスのカテゴライズの自動化方法を検討している．

クラスカテゴリを用いた命名バグの検出方法としては，現時点での構文解析から得られる情報だけでなく，クラス階層構造などを含めたより広範な静的プログラム解析から得られる情報をもとにしたより精度のよいものを目指したうえで，改めて評価を行うことを検討している．

参考文献

[1] Miltiadis Allamanis, Earl T. Baar, Christian Bard, and Charles Sutton, Suggesting accurate method and class names. *ESEC/FSE'15*, pp.38–49, 2015.

[2] Venera Arnaoudova, Massimiliano Di Penta, Giuliano Antoniol, and Yann-Gaël Guéhéneuc, A new family of software anti-patterns: linguistic anti-patterns. *2013 17th European Conference on Software Maintenance and Reengineering*, pp.187–196, 2013.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin, A neural probabilistic language model. *Journal of Machine Learning Research*, vol.3, pp.137—1155, 2003.

[4] Dave Binkley, Matthew Hearn, and Dawn Lawrie, Improving identifier informativeness using part of speech information. *MSR'11*, pp.203—206, 2011.

[5] William J. Brown, Raphael C. Malveau, Hays W. McCormick III, and Thomas J. Mowbray, AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. John Wiley and Sons, Inc, Canada, 1998.

[6] Einar W. Høst and Bjarte M. Østvold, Debugging method names. *ECOOP 2009 – Object-Oriented Programming*, LNCS 5653, pp.294–317, 2009.

リアクティブプログラミングに基づく分散計算基盤の可能性

紙名 哲生
大分大学
kamina@oita-u.ac.jp

要旨

リアクティブプログラミング（*RP*）言語が，実世界に分散配置された様々なモノやサービスを繋げて新たな価値を生み出すための適切な抽象化を提供できるかを考える．とくに，*RP* における時変値を分散システムに適応させる際の整合性維持や永続化などの課題について議論する．

1. はじめに

我が国が目指すべき未来社会の姿として，仮想空間と現実世界を高度に融合させて様々な課題を解決する人間中心の社会 (Society 5.0) が提唱されている．ここでの目標は，ソフトウェアの技術を用いてあらゆる人とモノとが IoT(Internet of Things) 機器等を通じて繋がり，様々な知識や情報サービスの共有によって新たな価値を生み出すことである．ここでいうソフトウェア技術のひとつとして代表的なものにマイクロサービスアーキテクチャがある．これは複数の小さなサービスを API によって連携させ，個々の修正や変更を迅速に行うことができる現在主流の分散アーキテクチャである．

一方こうした連携を実現する際には，個々のサービス間のデータ流の分析が欠かせない．しかしながら，データ流の分析と命令的なプログラミング言語を用いた実装との間には大きなギャップがある．連携をより直観的に記述するには，サービス間のデータ流を宣言的に記述できる抽象度の高いプログラミング言語が必要である．

そのような抽象化を行うプログラミング技術の一つに，**リアクティブプログラミング（RP）**がある．これは環境やユーザからの入力など，時間変化する値に応答するプログラムの記述に適している．とくに Angular，React，Vue などに代表される RP フレームワークは Web アプリケーションのフロントエンドとして成功している．

本稿では，RP 言語で用いられる抽象化である**時変値**を用いて，高い抽象度でのサービス間連携を宣言的に記述できるかについて，またそれを実現する上で解決すべき課題について議論する．

2. 時変値

時変値とはプログラムにおける変数の一種であるが，通常の変数とは異なり，明示的な再代入を行わず，時間とともにその値が変化する[1]．例として，Java を拡張した RP 言語 SignalJ[6] による時変値の宣言を以下に示す．

```
signal int a = 6;
signal int b = Integer.bitCount(a);
a++;
System.out.println(b); // 3
```

SignalJ では時変値の宣言時に修飾子 `signal` を用いる．上の例では二つの時変値 `a` と `b` が宣言されており，`b` は `a` を用いて計算される．`a` の値がインクリメント演算によって更新されると，`b` の値（`a` をビット列としたときの “1” の数）も再計算され，自動的に 2 から 3 に更新される．このように時変値同士を連携させる言語機構は，GUI やセンサとモータの連携など，時間変化する入力列を処理するプログラムの記述に便利である．

実世界に存在するデータ（例えば気象情報，IoT センサ，SNS におけるタイムラインなど）も多くは時間とともに値が変わる．これらを時変値として抽象化し，上の `bitCount` のように何らかのオペレータを用いて宣言的に連携を記述できれば，データ流をそのままプログラム

[1] 本稿では論理的な時間を扱う．ただし，後に示すとおり，永続時変値においてはそれが実時間の時刻印と結び付くことになる．

```
var2 = var1 * 2
var3 = var2 + var3
```

図 1. グリッチの例

として記述することができ，分析とのギャップが小さくなる．果たしてそのようなことは可能であろうか？

3. グリッチと分散 RP

RP 言語が抱える問題の一つに，分散環境におけるグリッチの回避がある．グリッチとは，値更新時に起こる一時的な不整合のことである．例えば，図 1 に示す関係式で成り立つ三つの時変値 var1，var2，var3 があるとする．var2 は var1 を用いて，var3 は var1 と var2 を用いて計算される．いま，var1 の値が 1 に設定されたとする．すると var2 と var3 の値が再計算されるが，もし var3 の値が var2 の値よりも先に計算されてしまうと，まだ確定していない var2 の値を用いて var3 の値が計算されてしまうことになる．

一般的には，こうした不整合はクロックを用いたり [5] 各時変値の計算の順序を正しく決めれば回避できる [10]．しかし，それぞれの時変値が異なる計算ノードで計算される分散環境においてはこの回避は容易ではない．まず，ネットワークの遅延や障害の影響を考慮に入れる必要がある．また，時変値の計算の順序を誰がどのように決定するのかという問題がある．閉じた一つのプログラムにおいては，それはコンパイラが決定すればよかったが，分散システム上においては，特別なコーディネータが決定を行うか（コーディネータで障害が起きた際の影響は大きい）[4]，あるいはノード同士が情報を送りあって計算の順序を決定するような分散アルゴリズムを考える必要がある [11]．

一方で，一般的に整合性の問題は分散システムにおいては古くから議論されていることであり，RP 言語もその議論の上に成立させることが望ましいと考えられる．

4. 時変値の永続化

また，上に示した気象情報などの「時間変化する値」は多くの場合永続データで，過去の値に対する問い合わせを行える場合が多い．一方で RP 言語の時変値は通常一時的な値で計算が終わると消えるため，こうした「時間変化する値」の適切な抽象化にはならない．そのため，**永続時変値** [7] が提案された．これは値変化の履歴を含めて抽象化した時変値のことで，上述した時変値の特徴に加えて，値変化に関する問い合わせを行う API を持っている．内部的には，各履歴の時刻印をキーにした時系列データベースを用いて実装される[2]．

例として，車輌追跡システムにおける以下の車輌クラスを考える．

```
signal class Vehicle {
  persistent signal double x, y;
  signal double v = calcVelocity(x,y);
  Vehicle(String id, double x,
  double y) { init(x,y); ... }
}
```

各車輌は状態として現在の座標値を表す x，y，それに各座標の変化量から求められる推定速度 v を持つ．これらはいずれも時変値である．さらにこれらが永続時変値であることを示す修飾子 persistent が付けられている．ここでは persistent と連携する時変値も永続時変値として扱われるため，v も永続時変値である．車輌クラスをインスタンス化する際には明示的に識別子が与えられ（上のコンストラクタ中の仮引数 id は必須である），その識別子と各 x，y，v の値変化が紐づけられる．ある車輌に対して，例えば指定した時刻における推定速度を求めるには，以下のように snapshot メソッドを用いる（これは上の車輌クラスのように signal と修飾されたクラスが暗黙的に持つ API メソッドである）．

```
double sampleV =
  new Vehicle("Oita_a_1234")
  .snapshot("2018-06-01T18:10:00").v;
```

まず，車輌クラスのインスタンス生成に識別子として"Oita_a_1234"を与えることにより，その識別子と紐づいた時変値の集まり（x, y, v）を得る．次に snapshot メソッドを用いて時間を遡り，フィールド v にアクセスすることでその時点での推定速度を得る．

上で生成されたインスタンスが消滅したり，アプリケーションそのものが停止したとしても，時変値が持つ履歴は，明示的に破壊されない限りディスクに残り続ける．永続時変値では，このように通常の時変値とは異な

[2]履歴は，理想的には全てを残しておけるのが望ましいが，それが現実的でない場合は不要になったデータを捨てることになる．そのような管理も，データベースを前提にすれば容易になる．

るライフサイクル管理の機構が用意されている．また，永続時変値は，センサとモータの連携のような閉じたアプリケーションだけではなく，オープンな時系列データとして提供されることも想定している [8].

5. 永続時変値と分散 RP

ところで，分散 RP におけるグリッチやその他の整合性の問題は，永続時変値においては分散時系列データベースにおける整合性の問題であると言い替えることができる．まず，オープンにアクセスされる場面を想定すると，永続時変値にアクセスする複数のプロセス間に矛盾があってはならない．一方，永続時変値においては全ての更新履歴に時刻印が付けられるため，時刻印の存在を前提にできる．時刻印の付けられ方が実世界で発生した事象の順序を忠実に再現することを要求するのか，何らかの因果関係との間に矛盾がなければよいのか [1]，あるいは観測された結果において矛盾がなければよいのか [2] については，アプリケーションに依存するであろう．また時刻印についても，大域的な時刻を想定するのか，Lamport の時刻印 [9] のようなものを用いるかなど，様々であろう．いずれにせよこうした問題については古くから数多くの研究がなされている．

グリッチの回避については，例えば関連する全ての永続時変値の値を先に計算しておき，それぞれに適切な時刻印（例えば共通する時刻印）を付与して分散したデータベースを同時更新する方法がある．図 1 の例の場合，予め `var1`，`var2`，`var3` の値を計算しておく（単一のノードで計算するのでグリッチは容易に回避できる）．次にそれらの値に同じ時刻印を適切に付与し，各ノード上のデータベースに格納する．分散コミット [3] の仕組みで各更新が成功しなければ観測されることはなく，不整合は生じない．永続時変値はこのような古くからの分散システムの分野における議論の上に成立しているため，分散 RP におけるグリッチ問題に対する解は既に与えられている場合が多そうだと予測できる．

6. まとめと今後の課題

RP 言語の抽象化である時変値を用いて宣言的なサービス間の連携が可能かどうかについて，とくに分散 RP におけるグリッチと時変値の永続化の問題を取り上げて議論した．永続データを含む「時間変化する値」は永続時変値を用いて RP 言語上で適切に表現することができ，またそれによりグリッチ問題も分散システムにおける整合性の問題へと帰着させることができる．永続時変値の研究は始まったばかりであり，それを用いた具体的なサービス間連携の実証が行われているわけではないものの，それが Society 5.0 を支えるソフトウェア開発のための便利な言語要素となる可能性は十分にある．

最後に，永続時変値を実用的な広域分散計算基盤として使える言語要素にしていくために今後取り組むべき課題について議論する．

ID 管理： 永続時変値において，インスタンス生成時に与える識別子はライフサイクル管理や個々の時変値の取得に用いられる必須のものである．識別子はプログラマが自由につけられるが，名前の衝突があってはならない．とくに広域的な計算基盤として永続時変値を用いるには，名前の衝突回避のための ID 管理機構を考える必要がある．

ソフトウェア進化： 永続時変値を内部的に管理する時系列データベースは，永続時変値をメンバに持つクラス（例えば上述の車輌クラス）の宣言を反映している．クラス定義に変更があれば，時系列データベースも同時に変更されるべきであり，その際に不整合が生じないようなデータベース変更の機構を考える必要がある．

参考文献

[1] Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, and Phillip W Hutto. Causal memory: Definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, 1995.

[2] Sebastian Burckhardt. Principles of eventual consistency. 2014.

[3] Eric C. Cooper. Analysis of distributed commit protocols. In *Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data*, pages 175—-183, 1982.

[4] Joscha Drechsler, Guido Salvaneschi, Ragnar Mogk, and Mira Mezini. Distributed REScala: an update algorithm for distributed reactive programming. In *OOPSLA'14*, pages 361–376, 2014.

[5] Nicholas Halbwachs, Paul Caspi, Pascal Paymond, and Daniel Pilaud. The synchronous data flow programming language Lustre. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[6] Tetsuo Kamina and Tomoyuki Aotani. Harmonizing signals and events with a lightweight extension to Java. *The Art, Science, and Engineering of Programming*, 2(3), 2018.

[7] Tetsuo Kamina and Tomoyuki Aotani. An approach for persistent time-varying values. In *Onward!'19*, pages 17–31, 2019.

[8] Tetsuo Kamina, Tomoyuki Aotani, and Hidehiko Masuhara. Signal classes: A mechanism for building synchronous and persistent signal networks. In *ECOOP 2021*, volume 194 of *LIPIcs*, pages 19:1–19:30, 2021.

[9] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558—565, 1978.

[10] Leo A. Meyerovich, Arjun Guha, Jacob Baskin, Gregory H. Cooper, Michael Greenberg, Aleks Bromfield, and Shriram Krishnamurthi. Flapjax: A programming language for Ajax applications. In *Proceedings of the 24th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Application (OOPSLA'09)*, pages 1–20, 2009.

[11] Florian Myter, Christophe Scholliers, and Wolfgang De Meuter. Distributed reactive programming for reactive distributed systems. *The Art, Science, and Engineering of Programming*, 3(3):5:1–5:52, 2019.

マンモスの牙
〜 Monetize Metrics Vs. Engineering Metrics 〜

松尾谷 徹
有限会社 デバッグ工学研究所
matsuodani@biglobe.jp

増田 礼子
フェリカネットワークス株式会社
Ayako.Masuda@FeliCaNetworks.co.jp

要旨

この資料はソフトウェア・シンポジウム *2021* の *FP (Future Presentation)* の資料として作成したものです. 提案の主旨は，ソフトウェア工学 *(SW* 工学*)* が産業界の「働き方」に及ぼした意図しない副作用 *(*牙*)* を示し，その対策について検討することです.

意図しない副作用とは，技術者に対し生産性向上の責務を語る労働強化です. 産業界が *SW* 工学に期待したのは，*QCD* 改善や属人性排除など，*Monetize (*金銭化*)* 側面での生産性向上であり，*SW* 工学は，この期待に対して様々な技法や技術を研究し，政策と共にソフトウェア工場やソフトウェア *CAD* を推進してきました. しかし，ソフトウェア関連活動の実態は，雇用されている技術者の属人性に依存しており，本質的な特性は変わりません. 掲げられた生産性向上の目標指標 *(Monetize Metrics)* は，工学的な手段ではなく，雇用された人々への労働強化的な副作用を生み，職種としての評価すら下げてしまいました.

ここでは，ソフトウェア産業における雇用者の「働き方」の何が特殊なのか？，*SW* 工学は「働き方」を守って技法や技術を提案できるのか？など，この問題について考え方や方向性を示し，その評価を行う実証的な *Engineering Metrics* の必要性について提案します.

1. はじめに

ソフトウェアに係る技術や産業は，他の産業と比べても飛躍的に拡大し，GAFA [1] に代表される世界最大級のビジネスに成長しました. この産業を支える一つの分野に，ソフトウェア工学 (SW 工学) があります. SW 工学は，産業の黎明期であった，1980 年代から盛んになり，それまでの研究から実用に向けた多様なアプローチが行われました.

SW 工学を活用する産業界から眺めると，過度な期待を持ち導入を進めた時代が終わると，系統的な技術が未成熟であることに気づき，個々の課題について，技術を適切に選択する活用へと変化しました. 新人研修で学ぶ，常識的な用語や知識としての SW 工学と，個々の課題に対応する個別の技術 (たとえば，セキュリティ課題で Fuzzing をどのように使うか？など) は，活用側で学習コストが必要です. 専門家は，この状況を理解していますが，ビジネス競争下では，根拠の無い生産性 (コスト) 競争が存在し，その結末が，雇用されている技術者への労働強化として作用し「働き方」の状況を悪化させています.

SW 工学は，産業界に弊害を与える意図は全くないのですが，ここで議論する「牙」問題は，工学界の意図と，産業界の意図が本質的に異なることと，ソフトウェア産業における労働の特殊性 (属人性) から生じています. 歴史的な経緯を含め，この問題について 2 章で説明します. 2 章では，Brooks の著書 ”The Mythical ManMonth ” [2] (日本語版「人月の神話」[3–6] 俗称「マンモスの神話」) で指摘された， Monetize Metrics の代表である「人月指標」を例に説明します. その後，産業界は急激に変化しており，ここ 10 年ほどを見ると Brooks の時代とは全く異なる状況になっており，人月問題とは別の構造で「牙」が生じています. 3 章では，この状況における「働き方」の問題について説明します. この状況は，

日本ではまだ一部での変化であり，今後，企業の存亡と共に拡大すると思われます．4 章では，サラリーマン型雇用下にある日本の技術者を対象として，ソフトウェア関連の仕事環境がどのように影響しているのかについて論じます.「働き方」の問題は，性能や不具合の数などで計測できる特性とは全く異なり，研究や計測は心理学の知見を利用することになります．紹介するのは，著者らが行った，仕事満足度に関連する質問紙を用いた調査 (1,700 名) など，いくつかの実証型の研究です.

これらを通して明らかになったことは,「働き方」問題は「人」を対象とする課題であり，課題構造が多様であること，局面で変化することです．構造的には，集団 (チームや組織) と職種 (技術者) の特性が，一般的な労働者の特性と異なっています．現在の「働き方」に対する評価は，仕事満足面では自己成長が若干強いですが，一般と変わりません．一方，仕事不満足は高く，仕事の成果との相関が報告されています.

2. マンモスの歴史

本章では，産業界においてソフトウェアの価値を Monetize の観点から「人月」を用いた歴史的な経緯を説明します．それを支援した SW 工学の考え方は「属人性の排除＝再現性を高める＝工学」です．ソフトウェア関連活動の付加価値生産性は，現実として属人性依存ですが，属人性要因を切り離しています．SW 工学が「牙」を生んだ直接の原因ではありませんが，現実に生じた問題として，属人性に依存する技法や技術を工学的に応用する場合に考慮すべき事例です.

2.1. 黎明期

半世紀前，コンピュータ産業の黎明期におけるソフトウェア開発は限られた職場でのみ行われる特別な仕事でした．主に次の 3 種類の職場です.

1. 汎用機メーカ：OS やコンパイラなどハードウェア販売の付属として提供するソフトウェアの開発
2. 研究・開発：コンピュータを使った研究や商品開発のためのソフトウェア開発
3. コンピュータ導入組織：コンピュータ導入による組織内事務作業の省力化のためのソフトウェア開発

この中で，ソフトウェア開発の仕事が急拡大したのは 3 番目の，新たにコンピュータを導入する組織 (企業や公共団体) でした．業務の内容も，前の 2 者とは異なり，組織固有のさまざまな伝票に基づく事務処理の電算化です．コンピュータ導入時の一時的な作業なので，人的資源を外部から調達し，プロジェクト (一時的な活動) で進める方法が使われました．その活動内容は 主に次の 2 種類が中心で，技術面はコンピュータを受注したメーカが行いました.

- データ入力，コンピュータ操作 など
- 伝票や帳票処理のプログラム作成 など

組織外にデータを持ち出せない事情から，プロジェクトの要員は派遣の形態となり，その取引に「人月」が自然に使われました．日本における初期のソフトウェア企業は，この種のビジネスから始まっています．監査法人や会計事務所など事務処理の専門集団から始まった海外の事務所理系ソフトウェア企業の多くとは違った流れです.

この時代における産業分類では，サービス業に分類されており，その後，電算機センターと呼ばれたコンピュータの共同利用ビジネス業界と統合され，1984 年に JISA (情報サービス産業協会) [7] が生まれました．それ以降の歴史は JISA の資料 [8] に詳しく書かれています.

2.2. ビジネス環境の変化

黎明期における派遣型のビジネスは，絶対的な供給量不足から，当時の一般的な派遣単金と比べとても高額を維持しており，ビジネスは繁栄しました．その後，大型の新規導入が減少し，中小組織への導入になると，コスト競争が始まりました．それでも，コンピュータメーカのパーク維持競争から，数年ごとに機種を変え，マイグレーションを行うことから，市場規模は維持されました.

事務処理系のプロジェクト購入主であるユーザ組織では，プロジェクトの投資効果が低いことから，人月単金の引き下げ，工数削減，納期確保など，生産性向上の要求が強くなり，結果，価格競争が始まります．供給側は，人件費の安い海外での開発など為替差益的な手法で対応したため，さらに人月単金は低下しました.

行政上の変化としては，1986 年に施行された労働者派遣法が，1996 年，1999 年に改定されました．これにより，プロジェクトでの指揮的マネジメント機能が弱体化し，プロジェクトで生じる問題の対応策が遅れ，さら

に生産性が低下しました．政策としては，情報処理技術者試験によるソフトウェア技術者を認定し，人的資源のスキル価値を区別して市場に供給する方策を続けていますが，資格と生産性の実証が無く，市場価格への影響は強くありません．

2.3. 牙の形成

ビジネスの世界においては，ソフトウェアに限らず生産性が評価されます．一人一人の評価ではなく，組織や業界としての生産性について考えます．一般的な生産性は次の 2 要素から構成されます．

1. 付加価値生産性 ＝ 付加価値額 ÷ 労働量
2. 量的生産性 ＝ 生産量 ÷ 労働量

黎明期における生産性は，量理的生産性で表して，顧客からは評価されました．この時の生産量はプログラムの量や作成した帳票の数などで定義でき，労働量は人月です．定型的な活動，入力と出力が明確な活動であれば，この方法が使えます．他の業界でも，たとえば，タイル張りであれば，時間あたり幾らの面積などの定義が業界で決まっています．

付加価値生産性は，付加価値額を測らないと分かりません．パッケージビジネスなら，パッケージの売価と販売数で測ることができます．プロジェクトの場合は，プロジェクトの売価になります．現実の問題として，黎明期のプロジェクト売価構造は,「プロジェクト原価 − コンピュータメーカの補助 + 利益」であり，顧客は，運用期間中のハードウェアレンタル料金をコンピュータメーカに支払い，コンピュータメーカは補助を回収する仕組みでした．

初期の携帯電話ビジネスのように，初期費用は安く見せて，月額で回収するビジネスモデルです．顧客にとってもこのモデルは価値があり，電算機導入後の事務処理コストを合理化した後で，レンタル経費を経理処理できます．

このビジネスモデルが崩壊したのは，汎用機のレンタルビジネスが破綻し，コンピュータメーカの補助が無くなり，プロジェクト売価が上昇したにも関わらず，プロジェクトの中身は変わらない事態になったためです．このギャップも従事する技術者の労働強化となって作用し，「牙」が形成されていきました．

2.4. 人的資源の特性

SW 工学においても，属人的な特性を計測し，データを用いた実証型の研究は存在します．代表的な研究として，Boehm が 1981 年にまとめた著書 ”Software engineering economics” [9] では，属人的な生産性の差をチーム単位で計測し，それをモデル化した ”COCOMO (COnstructive COst MOdel)” を示しました．このモデルは，一般的な線形回帰式 (1) で表すことができます．

$$Y = b_b X + A_E \tag{1}$$

過去の統計データから，見積もりや改善を行う方法は統計的品質管理の手法が使えると考え，産業界は大きな期待を持ちました．しかし，結果は残念な結末となりました．式の目的変数 Y は，log (人月)，説明変数 X は log (規模) です．式は線形ですが「人月」と「規模」の関係は対数正規分布であり，実軸に変換するとべき分布に分類されることから，一般的な統計手法が役立ちません．

対数に変換する前のリアルな値では「人月」と「規模」の関係は弱く，一般的な統計手法でモデルを作ることができません．その後，国内で行われた大規模な実証研究においても，同様の結果が観測されています [10,11]．一般的に調査範囲を広げ，データ量を増やせば，真実に近づきモデルの精度は上がるとされていますが，対象の特性が「対数正規分布」の場合，そのような特性は見られません.「対数正規分布」は，対数の世界では統計モデルが成立しますが，リアル軸に戻した場合には，産業界が求める精度には達しませんでした．

この研究は，結論として「ソフトウェア開発は再現性が低い」こと,「規模が大きくなると生産性が低下する」ことなど，一般的なマスプロダクトの知見とは異なることを明らかにしています．市場取引にとっては不都合な事実であり，ビジネス分野で積極的に使われることはありませんでした．

3. プロダクトビジネスの衰退

GAFA に代表される現代のビジネスは，サービスとそのインフラの提供です．その基盤はインターネット関連のインフラ提供であり，インターネットとスマートフォンやタブレットの普及とともに，世界中で無償で使えるサービスの提供を行い，覇権を制した上でさまざまな有償サービスを展開しています．

20 世紀末に生じた，プロダクトの権利問題や訴訟問題は，インターネット上で稼働するプロダクトの著作権を主張することが困難であることを示しました．Linux [12] や Android [13] などインターネットを利用するプロダクトは，独自の著作権主張するのが難しくなっています．結果的に，2010 年頃から，一斉にオープンソースへの移行が行われました．

プロダクトを納入して利益を得るビジネスと，サービスによって利益を得るビジネスの違いは何でしょうか．その仕事に従事する技術者の働き方と共に考えます．

3.1. 実質的な生産性の向上

ソフトウェアの特性として，流用の容易性，流用コストが小さい特性があります．流用の効果は，ソフトウェアに限らず，広く使われているナレッジであり，確かめられた先行事例の活用や蓄積が製品開発の基本的なプロセスとされています．

プログラミングを例にすると，ライブラリを活用する／しないによる生産性の差は大きいです．たとえば，WEB スクレイピング (WEB サイトからデータを取得し，情報の抽出・加工を行う) を行う場合において，任意のプログラム言語で，すべてを開発することは可能ではあります．別の方法としては，Python の Beautiful Soup [14] など専用のライブラリを使って開発することができます．

例に挙げたライブラリ Beautiful Soup は OSS なので使用権に対するコストはゼロであり，圧倒的に後者の実質生産性が高くなります．実質とは，目的達成に必要なコストで生産性を評価した場合で，人月指標だと，逆に前者が高くなります．

現実的な事例としては，2020 年 2 月末に始まった東京都 新型コロナウイルス感染症対策サイト [15,16] があります．この事例では，約 60 ほどの地域で Fork (ソースコードを複写して派生すること) され，サイトが開設されています．このような規模での再利用は，プロプライエタリ・ソフトウエアの世界 (人月の世界と重複) では考えられません．

SI ビジネスにおける企業規模が大きくなれば，人月指標の世界でも再利用による実質生産性が向上すると考えられます．しかし，個々の開発案件は顧客との契約があり，開発企業がすべての著作権を持っていないため，流用は限られ，流用による生産性向上は進んでいません．

顧客側にとっても，自社のシステムのソースコード以外を流用することはできず，購入したパッケージも使用権に制約があるため，実質生産性の向上にはつながりません．

まとめると，流用はソフトウェア生産性向上の基本的手段ですが，プロプライエタリ・ソフトウエアの場合，または，量的生産性評価の場合においては，阻害され，結果として付加価値生産性が向上しません．

3.2. ワークフロー標準化

オープンソースの慣習による開発は，開発の進め方やテストなどワークフローに関する情報の公開につながりました．その結果，実務に従事する技術者が体験した効率的な開発環境やワークフローが，デファクトスタンダードとして広がり，かつ改善が続いています．例を次に示します．

- **他の OSS の利用と依存関係**
  OSS が使用権，利用権を開放していることから「使えるものは作らない」が徹底しており，外部の機能を多用しています．その状況は GitHub [17] であれば Dependency Graph で表示されます．先に紹介した東京都 新型コロナウイルス感染症対策サイトの例でも，利用している外部機能は 1,200 件を超えています．OSS のプロダクトやサービスは氷山のようなもので，海面下では当該リポジトリには含まれない他の OSS に支えられ，その成果の上でさらに機能を開花させています．

- **マイクロサービスの概念導入**
  マイクロサービスとは，機能ごとに独立したサービスに分割し，相互に影響しない作り方です．この設計概念により，同時に複数の機能について変更を加えても，相互影響は小さくなります．特別な実装方法を使わなくても，多くの機能が既存の OSS を利用する呼び出しや接続であることが貢献しています．

- **自動テストなどの自動化**
  機能の独立性が高いことから，1 行のコード修正であっても当該機能の統合テストでデグレードを確認する習慣が定着しています．それを支える自動テストも機構間で独立である特性から，統合は

システムレベルですがテスト内容は機能テストで済みます.

また，膨大な依存関係から生じる依存先のバージョン問題については，Bot による自動検出を利用するなど，最新の技術が共有されています.

- **内部ドキュメントの廃止**
  従来の品質概念の中には内部ドキュメントの重要性があります．受発注の納入物にも含まれ，膨大なドキュメントにリソースが投入されていました．これらのドキュメントは，OSS 開発では全く見当たりませんが，大勢の技術者が外部から参加し，短期間で成果を上げています．マネジメントと同様，無い方が効率的だという実証結果です.

  これには，さまざまな原因が考えられます．たとえば，コードは公開され，必ず Review を受けるので，非効率でトリッキーな処理は排除されるなど作成者の技術習熟度が高くなっており，ドキュメント自動生成ツールなどから得る他との関係情報に重点が置かれています.

3.3. 技術者のモチベーション

商用に利用されている OSS 開発においても，雇用関係にないメンバが多数参加しています．このような環境下で，技術者はどのようにチームを作り協働を行っているのか，どのようなモチベーションを持っているのかについて，次の 3 つの現象から考えます.

1. **働き方**
   雇用や契約による労働提供下では観測されない現象として，活動の分布が指数分布かべき分布であること [18,19]，また，活動時間帯が広い範囲に及んでいる [18] ということが挙げられます．これは，自由意志での参加であり，その背景に社会貢献としての善行の意識があると考えられます．雇用されている一部のメンバも，この意識の影響を受け，行動変容が生じているように推察されます.

2. **協働と役割**
   3.2 節において組織内対立の課題を述べましたが，OSS 開発の中では「やらされ感」がなく，仕事に対する「オーナー感」が生じていると考えます．図 1 は，東京都 新型コロナウイルス感染症対策サイトにおける Issue / Pull Request を提起 (Create) あるいは Rewiew したメンバ (2 件以下の貢献を除く) の重複をベン図で示したものです．図 1 から，「オーナー感の範囲」も広く，役割の枠 (提起者 / Reviewer および Issue / Pull Request) を超えて協働が生じていることが分かります.

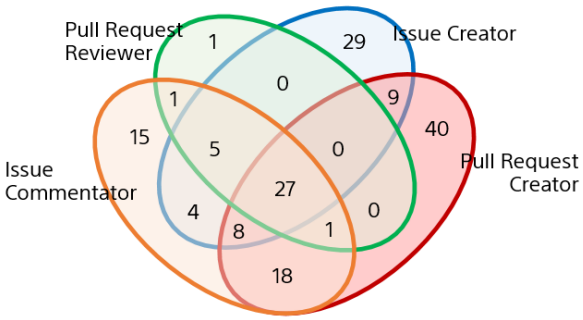


図 1: Create / Review における役割重複を示すベン図

3. **チームや仲間意識**

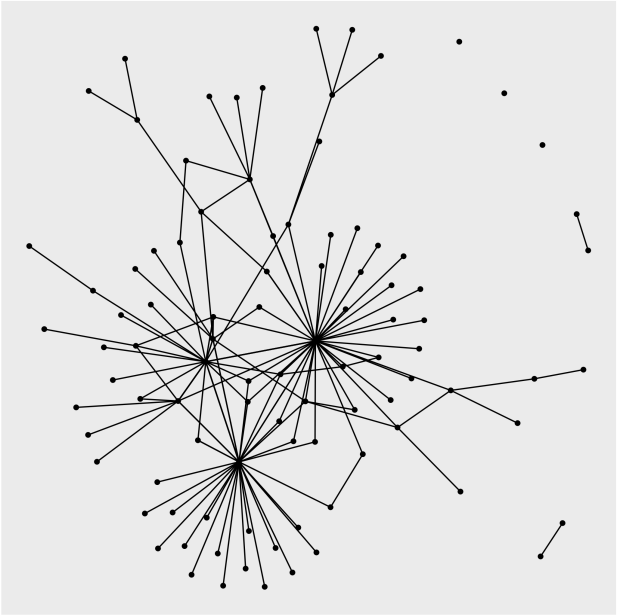


図 2: Pull Request と Review におけるチーム形成

連携活動のほぼすべてがサイバー空間で行われ，個々の活動はローカル環境での個人作業です．この環境下において，仲間意識や小集団としてのチー

ムが生まれるのかについて分析を行いました．図 2 は，東京都 新型コロナウイルス感染症対策サイトにおいて，開始後 1 ヶ月間の Pull Request と Review の協働活動の一部をネットワーク図で表したものです．図 2 のネットワーク分析においてクラスタと呼ばれる集団が確認できることから，チーム形成が行われていることが分かります．

従来型の仕事の進め方，あるいは技術者の働き方は Monetize の影響を強く受けています．この影響が強く同一化した価値観として「牙」が出現し，さまざまな弊害を生んでいます．比較分析として，直球勝負的な Monetize を絶ち，繁栄した OSS を調べることにより，今まで気づかなかった問題が見えてきたと感じています．

3.4. 新たな「牙」

サービス・ビジネスの世界では,「人月」問題とは異なる新たな「牙」が生じています．それは「成果主義」と「育成問題」です．

1. **成果主義**
   チームや組織に対する評価が明らかになりフィードバックされるようになると，その構成メンバである技術者個人の成果が識別されます．成果物に対するメンバの貢献分布は，多くの場合「べき分布」となり，少数の「Head」と多数の「Tail」に分かれています．べき分布が現実の中で，多様性や平等性をチームの規範として取り入れるかという問題があります．

2. **育成問題**
   技術者としてスキルを磨き成長するには，経験を積む必要があります．しかし，経験が無いと経験を積むポジションに就くことができないジレンマがあります．GitGub や一部の企業では，インターンシップを拡大して運用し，成長支援を行っています．

4. 働き方の測り方

属人的な特性を持つ活動の価値，すなわち生産性をどのように捉えるのでしょうか．ビジネスの世界における評価の流れは次の通りです，

- 供給不足の時代：頭数を揃える「人月」
- 量的生産性の時代：伝票数や規模の量と「人月」
- 終着段階：付加価値など，さまざまな要素

付加価値評価の終着段階においても，ビジネス観点からの評価は技術者の「働き方」にとって，多少ルールは異なりますが，労働強化として作用することになります．そのルール下の競争により，よりビジネスに貢献できる技術者を選別する仕組みです．強者にとっては，良い仕組みかもしれません．

「働き方」問題は，この競争に自ら志願して参加する強者の働き方だけではなく，サラリーマン型雇用下の技術者も対象とします．結論的に，ビジネスルールでの計測指標では「働き方」の良否に強いバイアスがかかり，この計測だけでは正しく表現できない，と言えます．

働いている技術者から観た「働き方」の状態を測る必要があります．この種の計測は，物理的な計測方法ではなく，人の行動を引き起こす意識を測り，統計処理を行って特性を明らかにする方法です．SW 工学においても GUI や要求の評価に用いられています [20]．

心理学では心理尺度と呼ばれ，その手法が確立しています．心理尺度の作成には質問紙法と呼ばれる手法が用いられています [21,22]．質問紙法は，計測対象を構造的にとらえ，下位尺度と呼ばれる要素に分解し，下位尺度ごとに質問項目を作成して計測を行います．プロジェクトマネジメントにおける先行研究では，プロジェクト・メンバのモチベーションを「職務満足感」「仕事意欲」「精神健康・ストレス反応」の 3 つの下位尺度に分け質問紙を作成し，そこから主成分を抽出して尺度化を行っています [23–25]．

ここでは，ソフトウェア分野における「働き方」について，実際に計測した 2 つの先行研究 [25,26] について，主に計測方法と分析方法について示します．

4.1. パートナー満足の計測と分析

この研究は，従業員満足のプロジェクト版であるパートナー満足 (PS：Partner Satisfaction) 調査として 2000 年に始まりました．質問紙による調査であり，サーベイの結果から，試作した質問紙を使って内部で評価を行い，その後，実際のプロジェクトで予備調査と分析を行いました．

最初の予備調査は，調査票の評価を兼ね，調査時期は 2001 年 10 月で，回収された件数は約 200 件でした．本調査は，情報サービス産業 3 社を主体とするプロジェクトに従事する技術者を対象とし，2002 年 1 月から 3 ヶ月間で行い，調査件数は回収ベースで約 1,700 件でした．

予備調査と本調査から，データの主成分分析を行った結果，目的変数を総合的な満足度とし，7 つの説明変数を抽出しました．7 つの説明変数は，次の構成です．

- **リーダーシップ**：プロジェクト (チーム) リーダのリーダシップ行動の影響
- **プロジェクトの運営方針**：プロジェクト運営の方針，方針に関する説明，運営状況の影響
- **コミュニケーション**：プロジェクト間，プロジェクト内チーム間，プロジェクト内のコミュニケーションの影響
- **スキルアップ**：プロジェクトの業務によるパートナーの市場価値の向上，仕事の挑戦性などの影響
- **業務上の問題**：業務上，調整や対応が必要となる問題 (不満足) の影響
- **業務外の問題**：業務上の直接の障害ではないが，プロジェクト内の力関係や雰囲気等，業務に影響を与える問題 (不満足) の影響
- **評価**：プロジェクトにおける評価の公正性，給与への反映などの影響
- **総合的な満足感**：プロジェクト及び自身の業務への全般的な満足感

この分析により，因子の尺度化を行い，プロジェクト間の差，職種による差，年齢層による差があることが明らかになりました．ここで得られた尺度は，その後の PS 調査の基礎データとして使われています．

4.2. 現場力とプロジェクト成果：計測と評価の試み

この調査は，2 つの異なったプロジェクト群 (調査事例 1：20 プロジェクト，調査事例 2：55 プロジェクト) に対して，現場力の要素とプロジェクト成否の関係を質問紙法を用いて計測し，その影響について判別分析により評価を行っています．相対的な評価ですが，2 事例におけるプロジェクト成否の判別分析では，95 % (20 プロジェクト: 4 段階評価) と 82 % (55 プロジェクト:5 段階評価) の高い判別率を得ることができています．

計測に用いた質問紙の構成は次の通りです．

表 1: 調査事例 1:下位尺度の質問数と因子数

下位尺度	質問数	因子数
顧客との関係性	7	2
仲間意識	5	1
役割意識	4	2
規範意識	4	2
合計	20	7

表 2: 調査事例 2:下位尺度の質問数と因子数

下位尺度	質問数	因子数
仲間意識	4	1
役割意識	7	2
規範意識	5	2
成果意識	4	2
環境意識	4	1
合計	24	8

質問紙を使った調査と分析について 2 つの先行研究を示しました．この種の調査方法は，主観的な評価に対して一般的に用いられており，対象が存在する「働き方」の計測としては有望と思われます.「働き方」の構成因子は多様なので，質問紙の質問項目で抜けると計測から漏れてしまう点や，同じ質問項目を繰り返して使うとバイアスが生じるなど，課題もある．

5. おわりに

ここでは，ソフトウェアに係る技術者の「働き方」をテーマとしました．産業界の求める生産性向上の期待をそのまま SW 工学の課題として捉えると，想定外の副作用として「働き方」に影響が及ぶことを示しました．想定外を想定するには「働き方」の主人公である技術者に対して，客観的な計測：Engineering Metrics として質問紙など心理調査の方法を取り入れ普及することを提案します．

参考文献

[1] Wikipedia, 「ビッグ・テック」, https://ja.wikipedia.org/wiki/ビッグ・テック (accessed:2021/05/14)

[2] Brooks, Frederick P, "The Mythical Man-Month : essays on software engineering", Addison Wesley, 1975

[3] フレデリック・フィリップス・ブルックス, 山内正弥 (訳), 「ソフトウェア開発の神話」, 企画センター, 1977

[4] フレデリック・P・ブルックス Jr., 滝沢徹 (訳), 牧野祐子 (訳), 宮澤昇 (訳), 「人月の神話 狼人間を撃つ銀の弾はない」 (原著発行 20 周年記念増訂版), 星雲社, アジソン・ウェスレイ・パブリッシャーズ・ジャパン, 1996

[5] フレデリック・P・ブルックス Jr., 滝沢徹 (訳), 牧野祐子 (訳), 宮澤昇 (訳), 「人月の神話 狼人間を撃つ銀の弾はない」 (原著発行 20 周年記念増訂版, 新装版), ピアソン・エデュケーション, 2002

[6] フレデリック・P・ブルックス Jr., 滝沢徹 (訳), 牧野祐子 (訳), 宮澤昇 (訳), 「『人月の神話 狼人間を撃つ銀の弾はない (新装版)」, 丸善出版, 2014

[7] 一般社団法人情報サービス産業協会, 「JISA とは」, https://www.jisa.or.jp/about_jisa/admission/admission/tabid/923/Default.aspx (accessed:2021/03/20)

[8] 一般社団法人情報サービス産業協会, 「情報サービス産業とは」, https://www.jisa.or.jp/explain/tabid/754/Default.aspx (accessed:2021/03/20)

[9] Barry W Boehm, "Software engineering economics", IEEE transactions on Software Engineering, No.1, pp. 4-21, 1984

[10] 寺本雅則, 松尾谷徹, 上村松男,「ソフトウェア開発過程を定量的に解析して生産性と品質を向上させるソフトウェアメトリクス」, 日経エレクトロニクス 6 月 4 日号, 1984

[11] 松尾谷徹,「開発コストのメトリクスの現状と課題」, 昭和 63 年電気・情報関連学会連合大会予稿集, 1988

[12] Linux, https://www.linux.com (accessed:2021/05/14)

[13] Android, https://www.android.com (accessed:2021/05/14)

[14] Leonard Richardson, "Beautiful Soup", https://www.crummy.com/software/BeautifulSoup/ (accessed:2021/03/20)

[15] 東京都 新型コロナウィルス感染症対策サイト, https://stopcovid19.metro.tokyo.lg.jp/ (accessed:2021/03/21)

[16] 東京都 新型コロナウイルス感染症対策サイト (GitHub) (Tokyo COVID-19 Task Force website), https://github.com/tokyo-metropolitan-gov/covid19 (accessed:2021/03/20)

[17] GitHub, https://github.com/ (accessed:2021/03/20)

[18] 松尾谷徹, 増田礼子,「コミュニティ型チーム活動の進化 〜 COVID-19 対策サイト開発から学ぶ〜」, ソフトウェア・シンポジウム 2020 論文集, pp. 117–123, 2020

[19] 増田礼子, 森本千佳子, 松尾谷徹, 津田和彦, 「大規模オープンソース・ソフトウェアプロジェクトにおける開発効率の計測」, 電気学会論文誌 C (電子・情報・システム部門誌), Vol.138, No.8, pp.1011–1019, 一般社団法人 電気学会, 2018

[20] 中尾仁, 津田和彦, 中谷多哉子,「ERP 導入の為のカスタマージャーニーマップを用いた顧客満足分析」, 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, Vol.112, No.314, pp.31–36, 一般社団法人電子情報通信学会, 2012

[21] 小塩真司, 西口利文,「質問紙調査の手順」, ナカニシヤ出版, 2007

[22] 吉田富二雄 (編),「人間と社会のつながりをとらえる「対人関係・価値観」: 心理測定尺度集 2」, サイエンス社, 2001

[23] 松尾谷徹,「パートナ満足によるソフトウェア生産性の向上」, 第 20 回ソフトウェア生産における品質管理シンポジウム, 日本科学技術連盟, 2001

[24] 松尾谷徹,「パートナー満足 (PS) と人的リソースのパフォーマンス」, プロジェクトマネジメント学会誌, Vol.4, No.1, pp.3–8, 一般社団法人プロジェクトマネジメント学会, 2002

[25] 榎田由紀子, 松尾谷徹,「Happiness & Active チームを構築する実践的アプローチ: チームビルディングスキルの開発 (¡ 特集¿ コミュニケーション・マネジメント」, プロジェクトマネジメント学会誌, Vol.7, No.1, pp.15–20, 一般社団法人プロジェクトマネジメント学会, 2005

[26] 松尾谷徹,「IT に現場力は存在するのか：その計測と評価の試み」, ソフトウェア・シンポジウム 2014 in 秋田 論文集, pp.1–8, ソフトウェア技術者協会, 2014

ソフトウェア・シンポジウム 2021
論文集
© ソフトウェア技術者協会

2021 年 5 月 30 日　初版発行

編者　　小笠原秀人
　　三輪東

発行者　ソフトウェア技術者協会

〒157-0073 東京都世田谷区砧二丁目 17 番 7 号
株式会社ニルソフトウェア内
ソフトウェア技術者協会 事務局
TEL: 03-6805-8931
http://sea.jp/
ISBN 978-4-916227-27-0

ソフトウェア技術者協会