

ソフトウェア・シンポジウム 2023 in 仙台

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 43 回目を迎える 2023 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、SS2022 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：2023 年 6 月 12 日(月曜日)～ 12 日(水曜日)

■ 場所：仙台国際センター(オンラインとのハイブリッド開催)

■ 主催：ソフトウェア技術者協会

■ 後援：情報処理推進機構

■ 協賛：宮城県情報サービス産業協会、マシンインテリジェンス研究会、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

澤田 恵介（仙台高等専門学校）
福川 優（パイオニアシステムテクノロジー）

実行委員

荒木 啓二郎（九州大学名誉教授）
池田 暁（クオリティアーツ）
伊藤 昌夫（ニルソフトウェア）
小笠原 秀人（千葉工業大学）
岡本 圭史（仙台高等専門学校）
小楠 聡美（HBA）
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.）
滝田 諭（パイオニアシステムテクノロジー）
富松 篤典（八代工業高等学校）
中野 一英（NDKCOM）
中野 秀男（中野秀男研究所）
奈良 隆正（NARA コンサルティング）
野村 行憲（ワイノムラドットコム）
橋本 明弘（パイオニアシステムテクノロジー）
原口 寛史（パイオニアシステムテクノロジー）
本多 慶匡（東京エレクトロン）
宮田 一平（SHIFT）
三輪 東（SCSK）

プログラム委員会

プログラム委員長

岡本 圭史（仙台高等専門学校）
根本 紀之（東京エレクトロン）

プログラム副委員長

小楠 聡美（HBA）

プログラム委員

秋山 浩一（ASTER）
安達 賢二（HBA）
天峯 聡介（岡山県立大学）
荒木 啓二郎（九州大学名誉教授）
伊藤 昌夫（ニルソフトウェア）
梅田 政信（九州工業大学）
大平 雅雄（和歌山大学）
小笠原 秀人（千葉工業大学）
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）
日下部 茂（長崎県立大学）
楠本 真二（大阪大学）
熊澤 努（SRA）
栗田 太郎（ソニー）
河野 哲也（メルカリ）
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）

プログラム委員会

阪井 誠 (SRA)
酒匂 寛 (デザイナーズデン)
佐原 伸 (ソフトウェア・エンジニア)
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 (八代工業高等学校)
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 コネクト)
西 康晴 (電気通信大学)
野村 行憲 (ワイノムラドットコム)
端山 毅 (NTT データ)
久住 憲嗣 (芝浦工業大学)
本多 慶匡 (東京エレクトロン)
増田 聡 (東京都市大学)
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)
宗平 順己 (武庫川女子大学)
森崎 修司 (名古屋大学)
森本 千佳子 (東京理科大学)
八木 将計 (日立製作所)
米島 博司 (パフォーマンス・インプルースメント・
アソシエイツ)
劉 少英 (広島大学) 阪井 誠 (SRA)

事務局

栗田 太郎 (ソニー)

ソフトウェア・シンポジウム 2023 in 仙台 目次

■ [DX] 論文・Future Presentation

[経験論文] DX 推進に必要なデジタルプラットフォーム整備の課題 宗平 順己（武庫川女子大学）		1
---	-------	---

[Future Presentation]

DX 成功への方策について 山本 修一郎（名古屋国際工科専門職大学）		11
---------------------------------------	-------	----

■ [自然言語処理応用] 論文・報告・Future Presentation

[Future Presentation]

テストプロセスにおける「ChatGPT」の活用の提案 堀川 尚人（株式会社 HBA）		13
---	-------	----

[Future Presentation]

文章埋め込み表現を用いた文章検索におけるコサイン類似度の補正 濱崎 利之（株式会社 NDKCOM）		17
--	-------	----

[事例報告] Generative AI とソフトウェア開発への応用の試行

酒匂 寛（Designers' Den Corp.）		20
----------------------------	-------	----

[経験論文] GPT-3 と LlamaIndex を用いた生成知識の比較

塚田 良央（株式会社ジェーエフピー）、佐々木 亮（株式会社ジェーエフピー）、 佐々木 千春（株式会社ジェーエフピー）、漆原 憲博（株式会社ジェーエフピー）		21
--	-------	----

[研究論文] 技術的負債に関する課題票の単語分散表現を用いたテキスト分類

田口 舞奈（和歌山大学）、木村 祐太（和歌山大学）、大平 雅雄（和歌山大学）		28
--	-------	----

■ [形式手法] 論文

[経験論文] Testing-Based Formal Verification on Software Programs : A Review Ai Liu (広島大学), Shaoyin Liu (広島大学)	38
[研究論文] Testing-based formal verification for ReLU-based neural networks Haiyi Liu (広島大学), Shaoying Liu (広島大学), Ai Liu (広島大学), Yujun Dai (広島大学)	44
[経験論文] 自然言語処理を用いた仕様書から VDM++仕様への変換支援について 酒井 玲弥 (仙台高等専門学校), 岡本 圭史 (仙台高等専門学校)	51

■ [システム開発・AI] 論文

[研究論文] ビルドエラーを引き起こす Dockerfile の依存関係の分析 坂本 廉也 (和歌山大学), 東 裕之輔 (日本総合研究所・和歌山大学), 大平 雅雄 (和歌山大学)	61
[研究論文] IoT システムにおけるワークフローモデルを用いた AI 推論処理の割当て最適化 伊藤 弘毅 (三菱電機株式会社), 徳本 晋 (富士通株式会社), 栗田 太郎 (ソニー株式会社), 石川 冬樹 (国立情報学研究所)	71
[研究論文] 除雪出動決定支援のための除雪関連情報共有サイトの構築 大倉 博貴 (北海道大学), 横山 想一郎 (北海道大学), 山下 倫央 (北海道大学), 川村 秀憲 (北海道大学)	78
[研究論文] 人物追跡手法とマッチングアルゴリズムを用いた路線バスにおける乗客 OD 推定 西浦 翼 (北海道大学), 横山 想一郎 (北海道大学), 山下 倫央 (北海道大学), 川村 秀憲 (北海道大学), 佐藤 好美 (株式会社シーズラボ), 長谷川 怜 (株式会社シーズラボ), 平澤 幸 (株式会社シーズラボ)	86

■ [品質・教育] 論文・報告・Future Presentation

[経験論文] 「テストからはじめよ」～忍者式テスト 20 年の実践から～

深谷 美和（キヤノンメディカルシステムズ株式会社），
関 将俊（キヤノンメディカルシステムズ株式会社） 96

[事例報告] 「ソフトウェアレビュー研究結果の認知拡大と適用促進」活動状況報告

安達 賢二（株式会社 HBA），中谷 一樹（TIS 株式会社），
上田 裕之（株式会社 DTS インサイト） 105

[経験論文] STAMP/STPA による農業用ハウスの運用シナリオの分析

日下部 茂（長崎県立大学），鍵 文音（長崎県立大学），有田 大作（長崎県立大学）
..... 112

[Future Presentation]

それって技術の仕事！？：“仕様の輻輳”問題～心理的輻輳から脱出して健康な開発に～

池田 暁（クオリティアーツ），松尾谷 徹（デバッグ工学研究所） 120

DX 推進に必要なデジタルプラットフォーム整備の課題

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

要旨

DX 白書 2023 はサブタイトルを「進み始めた「デジタル」、進まない「トランスフォーメーション」としており、日本の DX 推進の現状を示している。「トランスフォーメーション」推進に必要なのがデジタルプラットフォームであるが、DX への取り組みが「デジタルオペティマイゼーション」にとどまり「デジタルトランスフォーメーション」にまで至らない原因の一つとして、デジタルプラットフォームがうまく整備されていないのではないかと考え、この仮説を検証することとした。

本レポートでは、日本の DX 取組の現状を DX 白書 2023 で確認後、まず DX の教科書とでもいえる存在である Designed for Digital をベースに、デジタルプラットフォームの要件を整理し、その後、日本で紹介されている事例(DX基盤と呼ばれることもある)をチェックし、デジタルプラットフォーム整備における課題を明らかにする。

1. DX への取り組みにおける課題

DX 白書 2023 において、DX への取組の成果が出ている企業に対して、取組領域ごとの成果状況を尋ねた結果を示したのが図-1 である[1]。IT の延長線上である「業務の効率化による生産性の向上」において、成果が出ている割合が約 80%であり米国と差がなくなっているのに対し、デジタルトランスフォーメーションに相当する「新規製品・サービスの創出」「顧客起点の価値創出によるビジネスモデルの根本的な変革」については20%台で、米国の約 70%とは大きな差がある[1]。



図-1 DX の取組内容と成果

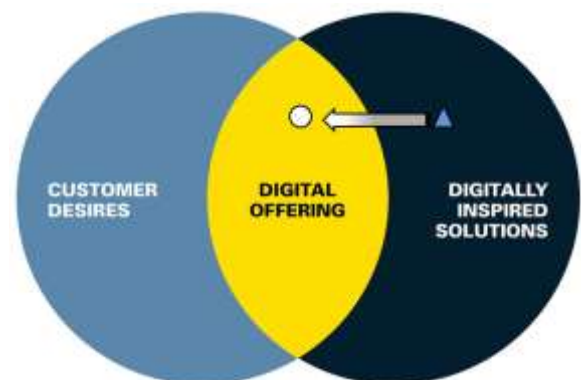
既存事業の強化／延命のための IT・デジタル活用はこれまでの取り組みの延長なので、容易に取り組めているが、デジタルを活用した新たなビジネスの創造には苦勞していることが DX 白書には示されている。

2. デジタルプラットフォームの要件

2.1. 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体験を提供するのであるが、MIT Sloan の Rossらは Designed for Digitalにおいて、「Digital Offering によって顧客に新たな体験を提供する」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-2 である。



※▲と○、矢印は今回追記

図-2 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(IT企業のみでなく DX に取り組む企業全般を指す)が新たな顧客体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 Ross らは Test-and-Learn のプロセスと名付けている[2]。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。

2.2. 5 Building Block

Ross ら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-2 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。以降各ブロックについて説明する。[2]

2.2.1. 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。

2.2.2. 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。(日本とは言葉の定義が異なっていることに留意頂きたい)

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なであるが、将来の競争優位を作ることとはできない。

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、Digitization によって Operational Backbone の効率化(Operational Excellence)を追求する。

2.2.3. 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。



図-3 5 Building Blocks

当初, Digital Offering のためにモノリシ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。

2.2.4. 2.2.4 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いうるものではなく, 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。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。

2.2.5. 2.2.5 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かaos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ながら, 図-6 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-4 デジタルコンポーネントのマネジメント原則

2.3. デジタルプラットフォームの設計要件

表-1 に, デジタルプラットフォームと Operational Backbone の要件を対比したものである(出典 2 より著者が翻訳)。デジタルプラットフォームの設計要件を理解す

るためのものとして紹介する。

表-1 デジタルプラットフォームの要件[2]

	Operational Backbone	Digital Platform
目標とする成果	プロセスの効率化, 予測可能性, 信頼性の向上による収益性の向上	収益拡大につながる新しいデジタルサービスの迅速な革新
技術要件	安定した, 拡張性のある, 安全なオペレーションシステム, 反復的なプロセスの自動化	API に対応したビジネス, データ, インフラストラクチャのコンポーネントのリポジトリ
必要なデータ	正確でアクセス可能な取引データおよびマスタデータ	ビッグデータ(センサ, ソーシャルメディアなど)の分析に適した柔軟なリポジトリ
プロセス要件	トランザクション処理アプリケーションの計画的・計画的な設計と実装	デジタルオフリングの設計, 開発, 構成, 商品化の反復
人的要件	プロセスオーナー, データアーキテクト, 大規模プロジェクトにおけるプロジェクトリーダー	プラットフォームアーキテクト:コンポーネントの機能性を仮説, テスト, 管理できるコンポーネントオーナー

2.4. デジタルプラットフォームの詳細仕様

Digital Offering を提供するために必要となるのが, デジタルプラットフォームであるが, その仕様について文献 2 に記載されている内容を抜粋・要約し, 見出しを付けたものを以下に記載する。

① 再利用可能なコンポーネントから構成

- ・デジタルプラットフォームは, ビジネス, データ, インフラストラクチャのコンポーネントのリポジトリであり, デジタルサービスを迅速に構成するために使用する。
- ・再利用可能なデジタルコンポーネントデジタルプラットフォームを構築すると, 企業は将来のデジタルサービスに役立つ可能性のあるコンポーネントのポートフォリオを蓄積していく。
- ・コンポーネントは, 特定のタスクを実行するコンポーネントである。例えば, 顧客の口座残高を取得する, ある場所への道順を指定する, センサーの測定値から機器の故障確率を計算する, ショッピングカートに顧客の注文を蓄積する, ユーザの身元を確認する, ダッシュボードでパフォーマンス結果を表示する, などのタスクを実行するコンポーネントがある。
- ・これらのコンポーネントをデジタルプラットフォーム上に配置することで, 新しいソフトウェアプロダクトを開発する個人が利用できるようになる。開発者は, ソフトウェアプロダクトに必要なすべての機能に対応するコードを

記述する代わりに、既存のコンポーネントを「呼び出す」ことでソフトウェアプロダクトを構成することができる。

② 各コンポーネントは API 経由で接続

- ・コンポーネントを再利用可能にする(つまり、開発者が新しいコードを書く代わりにコンポーネントを呼び出せるようにする)ために、開発者はコンポーネントを API 対応にする。
- ・よく設計されたデジタルプラットフォームでは、各コンポーネントが API を提供する(つまり、API が有効である)。そのため、API を利用することで、独立したコンポーネント間の事前定義された「プラグアンドプレイ」接続が可能になる。

③ 3つのタイプのコンポーネント

デジタルプラットフォームは、クラウド技術をベースに構築された以下の3つのリポジトリで構成される。

データコンポーネント

インフラストラクチャコンポーネント

ビジネスコンポーネント

・データコンポーネント

このリポジトリはデジタルプラットフォームの中心となる。企業は、デジタルサービスを構築する際、業務用バックボーンの業務データを活用するだけでなく、センサーやスマートデバイス、その他のウェブサービスからデータを購入・収集することになる。そして、再利用可能な API 対応のソフトウェアコンポーネントを構築または購入する。それぞれのコンポーネントには、データのいくつかの要素を保存、操作、分析、表示するコードが含まれている。

・インフラストラクチャコンポーネント

このリポジトリは、クラウドプラットフォームに組み込まれたサービスを、企業の提供するサービスや顧客の特定のニーズに適合させるための技術的なサービスコンポーネントを提供する。このコンポーネントには、ユーザ認証とアクセス制御、スマートデバイスの接続性、デバイス間の通信のオーケストレーション、使用状況の追跡とデータプライバシーを保証するサービスなどが含まれる。これらのコンポーネントは、ビジネスコンポーネントとクラウドサービスの橋渡し役として機能し、ビジネスコンポーネントが特定のクラウドプロバイダーのプラットフォームサービスを直接利用することを避け、特定のプロバイダーにロックインされないようにすることを目的としている。

・ビジネスコンポーネント

このリポジトリは、複数のデジタルサービスに必要な機能を提供する。これらのサービスには、ダッシュボード、

特定のイベントを顧客や従業員に通知するルール、新規顧客との関係を構築するオンボーディングプロセス、顧客に標準的なサポートサービスを提供するボットなどが含まれる可能性がある。企業によっては、関連性のない複数のデジタルビジネスを展開しているため、共通のビジネスコンポーネントのリポジトリが複数存在する場合もある。

④ コンポーネントの独立性

- ・デジタルプラットフォームは、1 コンポーネントずつ段階的に開発することができ、その構成要素は使い物にならなくなったら捨てることができる。

⑤ スケーラビリティ

- ・デジタルプラットフォームは継続的に進化することが可能であり、またそうでなければならない。

⑥ コンポーネントガバナンス

- ・デジタルプラットフォームは、ひとつのデジタル製品/サービスに必要なコンポーネントからスタートすることが考えられる。PoC が成功すれば、ビジネスチャンスが見つかり、企業は必要なコンポーネントの構築を開始することができる。しかし、このようなアプローチでは、初期のサービス提供の開始が遅れる。そのため、特定のサービスの機能を一回限りのモノリシックな方法でコーディングしかねない。しかし、このようなやり方は、初期のソフトウェアプロダクトでは有効かもしれないが、顧客の要求に対応するための機会が増えてくると、手戻りが発生する。

- ・デジタルプラットフォーム構築の段階的アプローチにおける課題は、正式な管理が必要となる十分な数のコンポーネントが揃うまで、プラットフォームが重要でない(あるいは、問題にする価値がない)ように思われることである。企業全体がデジタルプラットフォームの必要性を認識する頃には、誰も責任を負いたくないモノリシックなデジタルソフトウェアプロダクトという、かなり厄介なレガシーを構築してしまっている可能性がある。

- ・このため提案されたソフトウェアプロダクトをレビューして再利用可能なコンポーネントを特定するためのアーキテクトの役割と関連プロセスを確立する。例えば、BNY メロンでは、CIO のレポート担当者が、提案された新規または刷新された金融サービスのレビューを行い、どの機能が他のサービスで再利用できる可能性が最も高いかを判断する役割を担っている。

3. 日本のデジタルプラットフォームの検証

以上、Designed for Digital に記載されたデジタルプラットフォームの定義と対比する形で、日本で紹介されてい

るデジタルプラットフォームについて検証を進める。

3.1. 検証対象としたデジタルプラットフォーム

日経クロステック: 失敗しない DX 基盤の作り方より

- ・東京日動火災海上
- ・住友生命
- ・オイシックス
- ・三菱伊勢丹
- ・ZOZO

NDL-OPAC の検索結果より

- ・建物デジタルプラットフォーム

2021年度発表より

- ・みずほ FG
- ・Panasonic

DX 実践手引書より

- ・スサノオ・フレームワーク

3.2. 各デジタルプラットフォームの概要

3.2.1. 東京日動火災海上

東京日動火災海上では、迅速なサービス開発と継続的な改善を続けるために、マイクロサービスとサーバレスを組み合わせた CI/CD に対応した DevOps 基盤を AWS 上に構築している。基幹システムとは API を介して保険契約に関わる情報を連携している。[3]

DevOps 基盤と対をなすようにデータ分析基盤とデータ活用基盤が整備されている。顧客のふるまいを解析するなど、新たなサービスのヒントをつかんだり、リリースした新たなサービスの利用状況を分析したりするために利用している。

基幹システムからのデータ取り込みにあたってはオペレーショナルデータストアというシステムによって、プロジェクト毎に異なるデータ項目などを統一できるようにしてからデータストアに蓄積するようになっている。[4]

3.2.2. 住友生命

東京海上日動と同様に顧客分析のためのデータ分析基盤を住友生命も構築している。

主にレポートिंगのための新情報分析システムと AI 開発のためのスミセイデータプラットフォームから構成さ

れている。

スミセイデータプラットフォームはデジタルプラットフォームに該当するもので、顧客な大規模なライフログ健康データを蓄積し、Amazon SageMaker などによる分析を通じて、Well-being as a Service を提供している。[5]

3.2.3. オイシックス

オイシックスは、モノリシックになってしまったアプリを柔軟に変更できるようにするために、コンテナ環境を有するマイクロサービスベースの開発基盤を Azure 上に構築した。

当初は、既存システムから機能単位でデータをとアプリを切り出して個別サービスとしてサービス化しようとしたが、アプリとデータベースの関係が複雑なものがあり、既存の OracleDB を利用する必要のある機能は一つのサービスに集約し、独立できるものは個別サービス、切り出せる DB は個別 DB を構築している。[6]

3.2.4. 三越伊勢丹

三越伊勢丹じゃ短期間のサービス開発・リリースを実現するために DevOps 環境を Azure, AWS が提供する機能を組み合わせて構築している。

ここで構築されたデジタル系のアプリはビジネスプラットフォームという API 群を経由して基幹システムと連携している。基幹システムとの連携が必要になる都度 API を作成してこのプラットフォームに登録している。[7]

3.2.5. ZOZO

ZOZO がアプリケーションの開発スピードを向上させるため、モノリシックな既存システムをマイクロサービスに移行するためにマイクロサービス環境を構築した。スマホアプリは API ゲートウェイを経由して、機能単位に構築されたマイクロサービスを呼び出すようになっている。

従来システムから機能単位にマイクロサービスに段階的に移行するというアプローチを採用している。[8]

3.2.6. 建物デジタルプラットフォーム

清水建設が建物運用にあたって、多種多様な設備機器類の制御アプリケーションの相互連携を可能にするために構築したデジタルプラットフォームが図-5 に示す建物 OS である。[9]

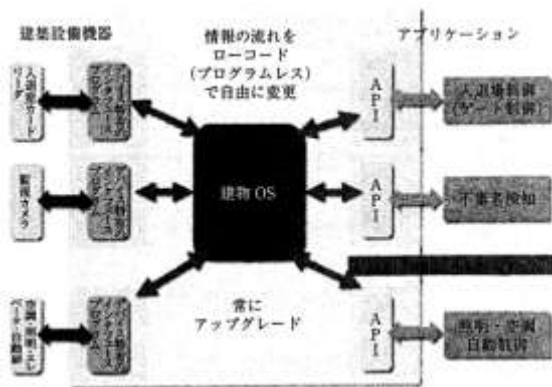


図-5 建物 OS

この利用イメージを図-6 に示す。3 密場所での換気強化のユースケースで、監視カメラ設備②の画像で、人の混雑を検知③したらその場所の換気量①を増やし、その制御結果をメッセージソフト⑤でスマホ④に知らせる。

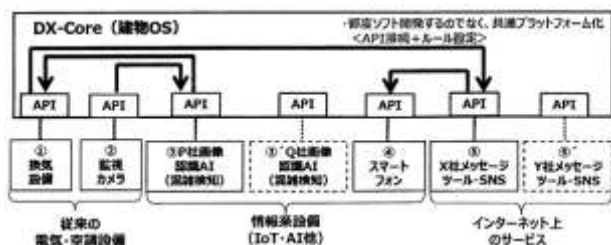


図-6 建物 OS の利用イメージ

また、API 連携としているため、③の入替、⑤の追加が容易にできることを示している。[10]

3.2.7. みずほ FG

図-7 はみずほ FG が構築した銀行 API である. [11]

みずほ銀行APIバンキング構成イメージ

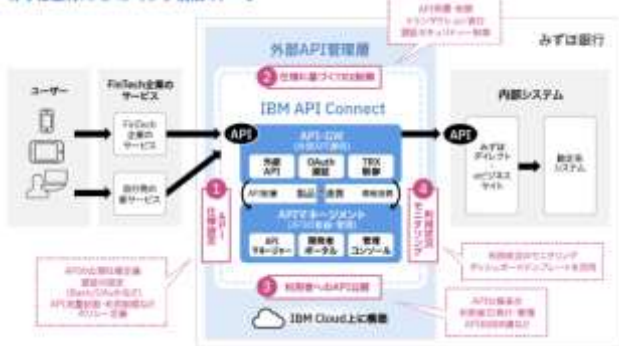


図-7 みずほ FG API

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



図-8 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。

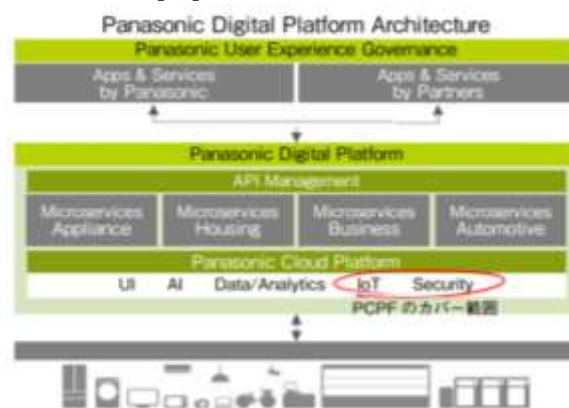


图-9 Panasonic Digital Platform

3.2.9. スサノオ・フレームワーク

IPA「DX 実践手引書 IT システム構築編」には、DX を実現するための IT システムの共通的な要素には、IT システムとその開発運用の体制が変化に対して俊敏かつ柔軟に対応できる「スピード・アジリティ」、社内外の円滑かつ効率的なシステム間連携を目指す「社会最適」、データ活用を中心に据えて社内外へ新たな価値を生み出してゆく「データ活用」の三つが挙げられるとして、この3つの要件を備えるようなシステムのあり方として図-10 が示されている。[14]

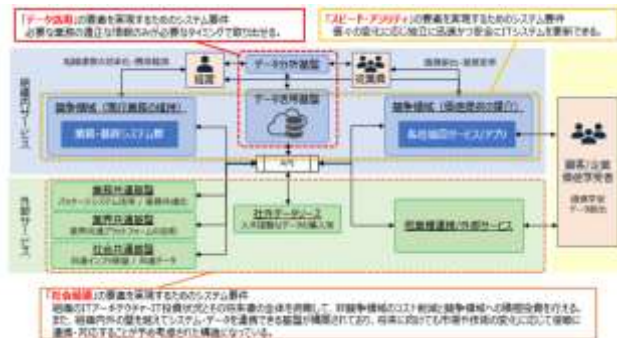


図-10 あるべきITシステムの要件

そしてこのシステム要件を実現する技術要素群が図-11 に示すもので、これを「スサノオ・フレームワーク」と名付けている。以下、その説明を引用する。

「あるべき IT システムは、各企業の競争領域として、変化に強い独自アプリケーションやそこに生かすデータの活用を行う基盤など自社の独自性を強化するための「組織内サービス」と、各企業の社内リソースを競争領域に注力するために、外のサービスや共通の基盤など、社外のリソースを活用する「外部サービス」を組み合わせた構成となっている。



図-11 スサノオ・フレームワーク

あるべき IT システムの個々の要素は、社内外の様々なサブシステムと連携するため、API(F)を基本のインター

フェイスとしている。顧客や市場の変化に耐えうるため、競争領域の独自アプリケーション(A)を、顧客や取引企業と接点を持つアプリケーションとしている。社内の現行業務を維持するとともに、競争領域の独自アプリケーション(A)と連携する形で、現行システム(B)をおいている。現行システム(B)を再構築する際には、移行を助けるため、媒介層(C)が他の IT システムとの連携を仲介する。また、媒介層(C)は、現行システム(B)が他の IT システムと連携する役割も担う。そうした、社内の IT システムや社外から取得するデータを、集約し、蓄積して、分析するため、データ活用基盤(D)・データ分析基盤(E)をデータ活用のための基盤としている。以上の組織内のサービスについて、システムリソースの柔軟な変更などを可能とするため、システム基盤(G)を稼働環境としている。

顧客や取引企業との設定を持つ競争領域の独自アプリケーション(A)が提供する機能を強化するため、特化した個別の機能は、外部サービス(H)と連携する。また、非競争領域となる業務は、外部が提供する環境を利用するため、外部共通基盤(I)と連携する。企業内にあるデータと、組み合わせることで、整理や分析に活用するため、社外データソース(J)を取り込むことも可能としている。」

(以上引用おわり)

また、競争領域の独自アプリケーション(A)の構築に関する技術として、マイクロサービスアーキテクチャの採用を提案しており、開発体制としては、1つのチームが、1つのサービスを開発し、その運用を行うこととしている。

3.3. 日本のデジタルプラットフォーム例の検証

Ross らのデジタルプラットフォームへの要求事項に対し、これまで紹介してきた日本のデジタルプラットフォームの例がどのように対応しているかをまとめたものを表-2 に示す。以下、個々の評価内容について示す。

・東京日動火災海上

継続的な発展を考慮したマイクロサービスベースの開発基盤であり、ほぼデジタルプラットフォームの要件を満たしているのではあるが、マイクロサービスの種類を意識しているかどうかは不確かであり、また重複開発やモノリシックを防ぐようなガバナンスを有しているかどうかは定かではない。

・住友生命

アプリケーションに関し、レポーティングは Tableau、AI 開発は SageMaker を使っており、マイクロサービスを使った開発環境については、対外発表されていない。

・オイシックス

全体を吟味してマイクロサービスを構築していることか

表-2 日本のデジタルプラットフォーム例の評価結果

MIT Digital Platformの要求事項	再利用可能なコンポーネントから構成	各コンポーネントはAPI経由で接続	ビジネスコンポーネント	インフラコンポーネント	データコンポーネント	コンポーネントの独立性	スケーラビリティ	コンポーネントガバナンス
東京火災海上	○	○	△	△	△	○	○	?
住友生命	?	?	?	?	○	?	?	?
オイシックス	○	○	○	△	○	○	?	△
三菱伊勢丹	○	?	?	?	?	?	?	?
ZOZO	○	○	○	○	△	○	?	?
建物OS	?	○	?	?	△	○	?	?
みずほFG	?	○	△	○	△	○	?	○
Panasonic	○	○	○	○	○	○	○	?
スサノオ・フレームワーク	○	○	?	?	○	○	?	○

ら、コンポーネントガバナンスがある程度効いていることは推測されるが、インフラコンポーネントに相当するものは公表資料にはなく、またスケーラビリティについての仕組みも定かではない。

・三菱伊勢丹

コンテナ基盤をベースとした DevOps 基盤ではあるが、マイクロサービスの開発環境については公表されておらず、ほとんどの項目について判断ができない。

・ZOZO

マイクロサービスをその機能を意識して開発しており、デジタルプラットフォームの要件を良く満たしているが、徐々に移行するにあたってのガバナンスや既存マイクロサービスのレベルアップなどの方策をとっているかどうかは定かではない。

・建物デジタルプラットフォーム

デジタルプラットフォームと命名しているが、実態は各機器の持つソフトウェア間のデータ連携基盤である。各機器のソフトウェアを開発する機能は有していない。

・みずほ FG

API ガバナンスが効いた構造になっているが、API の再利用性やスケーラビリティについての機能は公開資料には見当たらない。またインフラ以外のコンポーネントの種類としてどのようなものがあるのかも不明である。

・Panasonic

コンポーネントガバナンスについては不明ではあるが、デジタルプラットフォームの要件を満たしている。

・スサノオ・フレームワーク

マイクロサービスの利用とその開発運用体制については示しているが、マイクロサービスの再利用やスケーラビリティについては言及されていない。

Web メディアの記事などからの情報であるため、入手委できている情報は限定的なものであることは考慮にされる必要があるが、全体的な特徴として以下の点が指摘

できる。

- マイクロサービスを使うことは前提としているが、どのようなタイプのものがあるのかがまだわかっていない。
- データ連携を ESB の様なイメージで捉えている。
- データコンポーネントについてはクラウドサービスを使うなどしてデータ分析基盤として構築している。
- API はマイクロサービス間の連携というよりも既存システムやデータ分析基盤との連携のツールとして認識されており、このためデジタルプラットフォーム＝データ連携基盤という理解をしているものも見受けられる。

スケーラビリティやコンポーネントガバナンスについてはほとんど考慮されている様子が見られない。

4. DX 推進の技術面での課題

4.1. 日本の DX 推進の現状との関連

「DX への取り組みにおける課題」において新たな顧客価値創出への取り組みに大きな遅れがみられることをしめしたが、図-18 に示すように、同様の結果が AI への取り組みにおいてもみられている。[1]



図-18 AIの導入目的

AI の活用においても日本ではこれまでの開発手法を適用しやすい、効率化や品質向上への適用が中心にな

っているのに対し、米国では顧客価値向上のために AI を活用する方へとシフトしている。

なぜ、このような違いが生まれるのか、その理由の一つとして、新たな開発手法の採用が遅れていることにあることが、図-19 から分かる。[1]

効率性向上のためのデジタル技術の活用は従来型の IT 導入と同様に、ウォーターフォール型でも開発を進めやすい。

これに対し、顧客価値向上のための開発は、Test-and-Learn で進めなければならないが、図-19 に示すようにそのために必要な開発手法の導入が米国と比して大きく遅れている。



図-19 新たなシステム開発手法・技術の活用状況

4.2. 日本の DX 推進の技術的課題

本論の初期仮説は、「DX への取り組みが「デジタルオペティマイゼーション」にとどまり「デジタルトランスフォーメーション」にまで至らない原因の一つとして、デジタルプラットフォームがうまく整備されていないのではないかと」いうことであった。

この観点から公開情報から様々なデジタルプラットフォームの事例を検証したが、以下の点が明らかになった。

- マイクロサービスを使うことは前提としているが、どのようなタイプのものがあるのかがまだわかっていない。
- スケーラビリティやコンポーネントガバナンスについてはほとんど考慮されている様子が見られない。

なぜマイクロサービスの利用にあたって、重要な観点である上記の内容が欠けているのか疑問であったが、図-19 に示すように、日本ではそもそもマイクロサービスの導入が進んでおらず、加えて、デザイン思考、アジャイル開

発といった Test-and-Learn において必要な開発手法の導入も進んでいないことから、国内において十分な知見が溜まっていないのではないかと推測できる。

スサノオ・フレームワークにも上記の点が欠けていることも、このことから納得できる。

しかしながら、今後利活用が進むと知見が溜まるから大丈夫かというということではない。

Ross らは「⑥コンポーネントガバナンス」に関して以下の様な記述をしている。

「企業全体がデジタルプラットフォームの必要性を認識する頃には、誰も責任を負いたくないモノリシックなデジタル製品という、かなり厄介なレガシーを構築してしまっている可能性がある。

実際、デジタルプラットフォームが十分に発達している企業は、平均して、デジタル製品の 61%を再利用可能なコンポーネントから構築しているのに対し、デジタルプラットフォームがあまり発達していない企業では 32%にとどまっている。」[2]

日本がこのまま手探りでデジタルプラットフォーム上での開発を続けると、同じことを繰り返す危険が大いにあると考えている。

米国では、SOA の導入が進んでいたことから、「再利用可能なコンポーネント」の重要性やそのためのガバナンス(SOA ガバナンス)の理解があったと思われるが、それでも上述の様にまずい対応をする企業も多いということである。

SOA の導入があまり進まなかった日本ではさらに大きな混乱が生じるものと危惧される。

米国など先行して取り組んでいる企業の知見を調査し、日本企業にガイドとして紹介することが早急に必要になると考えられる。

参考文献

- [1] 独立行政法人情報処理推進機構、「DX 白書 2023」, <https://www.ipa.go.jp/files/000108041.pdf>, 2023/2/9
- [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] 森山徹, 「DX の「エンジン」になる IT 基盤, 先行事例に見る 2 つの不可欠な要素とは」, <https://xtech.nikkei.com/atcl/nxt/column/18/01688/070200008/>, 2021/7/12
- [4] 森山徹, 「東京海上日動が新たな「データ活用基盤」

で DX 加速, AI アプリを次々生み出す」,
<https://xtech.nikkei.com/atcl/nxt/column/18/01688/061400001/>, 2021/6/16

- [5] 森山徹,「DWH とデータレイクを運用する住友生命, なぜ 2 つのデータ分析基盤が必要なのか」,
<https://xtech.nikkei.com/atcl/nxt/column/18/01688/070200006/>, 2021/7/6
- [6] 森山徹,「マイクロサービスに挑むオイシックス, 既存システムの機能とデータをどう切り出すか」,
<https://xtech.nikkei.com/atcl/nxt/column/18/01688/070200004/>, 2021/7/8
- [7] 森山徹,「三越伊勢丹は DX で「4 倍速開発」, 下支えする 2 つの IT 基盤とは」,
<https://xtech.nikkei.com/atcl/nxt/column/18/01688/061600003/>, 2021/6/18
- [8] 森山徹,「ZOZO が API ゲートウェイを「自作」, クラウドのサービスは使わないワケ」,
<https://xtech.nikkei.com/atcl/nxt/column/18/01688/061500002/>, 2021/6/17
- [9] 菅裕之,「デジタルプラットフォームによる高度運用の促進とデジタルツイン」, 空気調和・衛生工学, 大 95 巻第 11 号, 2021/11, pp31-36
- [10] 山崎元明, 廣瀬啓一, 白石 理人,「建物デジタルプラットフォーム「DX-Core(建物 OS)」の開発」, 清水建設研究報告 第 99 号, 2021/12, pp23-27
- [11] <https://www.ibm.com/blogs/solutions/jp-ja/ca-mizuho/>
- [12] 大徳雅博, 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
- [13] 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, pp9-14
- [14] 独立行政法人情報処理推進機構 社会基盤センター,「DX 実践手引書 IT システム構築編」,
<https://www.ipa.go.jp/files/000094497.pdf>, 2023/3/2

DX 成功への方策について

山本 修一郎
名古屋国際工科専門職大学
syamamoto@acm.org

要旨

デジタル変革(DX)を推進しようとする多くの企業にとって、自社の未来を描くことが難しいようである。

以下では、まず、我が国の DX が進まない根本原因が問題解決にあることを指摘する。次いで、DX の 4 本柱、ビジネスモデルの描き方、データ駆動工程設計法を説明する。最後に、まとめと今後の課題について述べる。

1. DX が進まない原因

DX 戦略を描こうとすると、「現在のことはよく分かるのだが、未来が分からない」という企業人が多い。未来が描けないと、変革計画も描けないから、DX 戦略が描けなくなる。企業の実務家には、現在の業務をどうするかで頭の中が一杯で、先のことを考える余裕がない。

従来から SWOT 分析、PPM 分析、AHP などの手法が知られている。しかし、これらの手法は、既存の事業や製品に基づく分析手法であって、正常化バイアスに陥った日本企業が、あるべき姿を描く上では限界がある。未来から現在を見るバックキャストが大切だと分かっているにもかかわらず、過去からしか現在を見ることができなければ未来を描くことはできない。

2. 問題解決の 2 重ループ

問題解決には、外部ループと内部ループがある。外部ループでは、あるべき姿に対する問題を定義して、その解決策を反復的に探索する。内部ループでは、解決策を実現する手段を反復的に改善する。

「なぜ、日本企業は、あるべき姿を描けないのか」という問いに対する筆者の答えは、以下の通りである。すなわち、日本企業は、すでにある解決策を改善することが問題解決だと考えているから、解決策が複雑化してガラパゴス化が加速する。一方、問題解決の外部ループに取り組まないから、企業の将来の姿を明確にできない[1]。

3. DX の 4 本柱

企業の業務を変革するために実行主体の変革が必要である。実行主体が変わらなければ業務工程を変革できない。業務の実行主体には、顧客、社員、経営層、パートナー企業がある。これらの実行主体に応じて、CX(Customer eXperience), EX(Employee eXperience), MX(Management Transformation), PX(Partner eXperience)を定義できる[2]。この 4 本柱で DX を俯瞰できるだけでなく、相互関係や取組み順序を識別できる。

CX と PX は企業の外部境界の体験、EX と MX は企業内部の体験である。この 2 つの体験の変革が循環する。たとえば、まず、経営価値「データ駆動経営」を定義することにより MX を推進する。次いで、顧客分析システムを実現し、高頻度購入者にポイントを提供する顧客接点を提供することにより CX を推進できる。このように 4 本柱に基づく DX プロジェクトが複数あることから、SWOT 分析、PPM 分析、AHP などの既存手法を用いることにより効果的な DX 推進戦略を策定できると思われる。

クリステンセンのジョブ理論[3]では、成功するイノベーションを予測するには、顧客の関心事 Jobs to be done を解決するサービスや製品を創造することだと指摘している。4 本柱の顧客、社員、経営者、パートナー企業の困りごとを発見することで、あるべき姿としての新規事業を描くことができると思われる。筆者は Enterprise Architecture に基づいて 具体的な製品やサービスを設計できる MBJT を提案している[4]。

4. ビジネスモデルの描き方

新規のデジタルビジネスを創出しようということで、ビジネスモデルキャンパス(BMC)を描こうとすることがある。しかし、BMC では簡潔にビジネスモデルを表現できる反面、抽象化が必要になるため、経験に乏しい初心者がビジネスモデルのアイデアを考えるのは容易ではない。たとえば、本学で実施した「MaaS のためのビジネスモデルを創

造する地域共創デザイン実習」では、担当した学生たちから「自分たちには BMC でビジネスモデルを描くのは難しい」と言われた。そこで、まずカスタマージャーニーマップ[5]を用いて顧客の活動を抽出して、サービスブループリント[6]で顧客の活動を支援するサービスを定義してから、BMC を考えることを提案した。この結果、学生たちも BMC で MaaS のビジネスモデルを描くことができた[7]。

このように、まず、具体的なサービスを提供する断片的なユーザーストーリーを集めて、ボトムアップでまとめていくことにより、ビジネスモデルを考えるほうが自然である。前述した 4 本柱を具体化するために、対象とする顧客や社員に対するユーザーストーリーを描くことでビジネスモデルを創出できる可能性がある。

5. 段階的 DX 推進方法

DX を推進する最初の一步をどうするか？この問いに対して、経営者が DX に関与することとして、まず「儲かる DX」を示す必要がある。DX を成功に導くためのデジタル学習曲線[8]は、モダナイゼーション、全社変革、新規事業創出の 3 段階からなる。まず、既存業務プロセスを簡素化することで効率化を目指す DX を実現する。次いで、部門横断によるオペレーション改善と顧客体験・社員体験の向上により、全社変革を実現する。さらに、外部連携によるエコシステムと組織境界の見直しによる新規事業創出を実現する。この学習曲線によれば、日本企業が DX に踏み出す最初の取り組みは既存業務プロセスの効率化である。日本企業の業務形態が人に紐づく石垣型であり、ジョブ型業務ではないことが業務プロセス変革の問題になる。複雑な現行業務を分析し、無駄を発見して簡潔化するために多くの工数が必要である。

データ駆動工程設計法[9]では、現行業務の内部を洗い出すのではなく、現行業務への外部入力と、最終的な外部出力を明確にして、外部入出力の依存関係に基づいて業務工程を設計することができる。複雑な業務の詳細ではなく、入力を出力に変換する必要最小限の業務だけを設計するので、業務工程を最小化できる。

6. まとめ

自社の未来を描くことは、企業の持続的成長のために、「なぜ、誰に、何を、どうするか」という問いに答えることである。ダイムラーによるシステム思考に基づく企業組織の行動変革[10]では、目的「なぜ」、戦略「何を」、行動指針「どのように」、社員スキル「誰に」を定義している。したが

って、DX 推進企業では、データ戦略を「なぜ」「何を」「誰に」「どのように」を明確に定義する必要がある。これにより、全社戦略に基づくデータ利活用を推進できる組織に変革できる。このような組織文化の再構築を一気に実現するのは難しく、対話と試行運用を反復するプロセスが必要になる。

本稿では、「なぜ」について、問題解決の 2 重ループ、「誰に」について DX の 4 本柱、「何を」についてビジネスモデルの描き方、「どうするか」についてデータ駆動工程設計法を説明した。

今後、システム思考に基づいて、既存手法を含めて、これらの手法を統合するとともに、有効性を実証していく必要がある。

参考文献

- [1] 山本修一郎, DX の本質は課題解決, DX 白書 2023, p.176
- [2] 山本修一郎, DX 成功への道筋, リスクマネジメント TODAY, 3 月号, pp.6-9, 2023
- [3] Christensen, C., Hall, R., Dillson, K., and Duncan, D., *Competing Against Luck*, HarperCollins Publishers LLC, USA, 2016
- [4] Yamamoto, S., et.al., *e-Healthcare Service Design Using Model Based Jobs Theory, Intelligent Interactive Multimedia Systems and Services*, Vol. 98, 2019, ISBN: 978-3-319-92230-0
- [5] Shostack, L. (1984) "Design Services that Deliver." *Harvard Business Review*: 133-139.
- [6] Richardson, A., *Using Customer Journey Maps to Improve Customer Experience*, HBR 15(11), pp.1-5, 2010
- [7] Yamamoto, S. et.al., *A Technique to Integrate Service Business Models with ArchiMate*, CENTERIS2022, <https://www.sciencedirect.com/science/article/pii/S1877050923003241>
- [8] ディディエネ・ボネ, DX を成功に導くカギは「デジタル学習曲線」にある-- 3 つの段階を踏んで成熟度を高めることが必要な理由, 2022.11.16, <https://dhbr.diamond.jp/articles/-/8979>
- [9] 山本修一郎, 細見純子, データ駆動型工程設計法の提案, KBSE 研究会信学技報 KBSE2022-24, vol. 122, no. 139, pp.79-84, 5/21, 2022
- [10] Daimler, Melissa, *ReCulturing: Design Your Company Culture to Connect with Strategy and Purpose for Lasting Success*, McGraw Hill, 2022

テストプロセスにおける「ChatGPT」の活用の提案

堀川 尚人
株式会社HBA
horikawaxn@hba.co.jp

要旨

本稿では、膨大かつ複雑化するシステムテストにおける課題を解決するために人工知能技術の ChatGPT を活用する方法を提案する。

ChatGPT を活用して、テスト担当者の負荷を軽減し、高品質なテストを実現することが可能か、テストプロセスの一部をサポートさせることが可能かを検証し、その有効性や可能性について議論したい。

1. はじめに

システムテストは、ソフトウェア開発においてシステム全体の品質を確保するための重要なフェーズである。システムテストを効果的に実施することは、高いソフトウェア品質を達成するために必要不可欠である。しかし、テストプロセスはしばしば時間がかかる。

筆者は車載開発に関するテスト業務に関わっているため自動車を例に挙げると、自動車に搭載する機能が年々増加かつ複雑化しており、それによってテストケースも増加かつ複雑化している。

今後の車載開発では「機能や複雑度が増加したにもかかわらず、エンドユーザーは自動車の機能に対して従来と同等またはそれ以上の品質」[1]を求めるため、今後さらに増加かつ複雑化するテストケースを、従来と同様の期間で作成できるようにすることが今後の課題となる。これは車載開発に関わらず他の多くのシステム開発でも同様であると考えられる。

また、テスト品質はテスト担当者の経験にも依存するため、高品質なテストを実施するには経験豊富な人材が必要となる。しかしソフトウェアテストの経験や知識に富んだ人材はなかなかおらず、人材育成にも時間と予算が必要となる。

そこで、これらの問題を解決するために人工知能に注目した。

近年の人工知能の進歩により、自然言語処理技術を

活用した ChatGPT が注目されている。ChatGPT は、人工知能によって学習された言語モデルであり、自然言語処理技術を用いて文章の生成や応答を行うことができる。

テストプロセスにおいて求められる能力の一つに、日本語で書かれた要求を分析することが挙げられる。そこで、ChatGPT を用いてテストプロセスの一部をサポートできれば、テスト担当者の負荷軽減や高品質なテストの実現が期待できると考えられる。ただし、大規模開発に ChatGPT を活用した具体的な事例はなく、テストプロセスに ChatGPT を活用した事例も見つけることができなかった。

そのため、本稿では小規模な事例を用いてテストプロセスにおける ChatGPT の活用可能性を検証し、テストプロセスをサポートする方法とその有効性を提案する。

2. 検証方法

本稿で検証するテストタイプとテストレベルは、テストレベルがシステムテスト、テストタイプは機能テストを想定した。

検証方法として、数種類の要求サンプルを用意し、それらに対して ChatGPT に質問を投げ、その出力結果から有用性を検証した。ChatGPT のモデルは GPT-4 を使用した。なお、本検証では ChatGPT の回答の正確性に焦点を当てるのではなく、ChatGPT が実現できる機能を検証することに重点を置いている。

検証方法の例は図 1 である。要求サンプルの一つに、マイヤーズの三角形問題[2]を使用したため、それを用いたケースである。マイヤーズの三角形問題とは、ユーザーが入力した 3 つの整数を三角形の辺の長さとして、不等辺三角形、二等辺三角形、正三角形のうちどれに該当するかを判定し、結果をメッセージで出力するプログラムに対して、テストケースを導出する問題である。

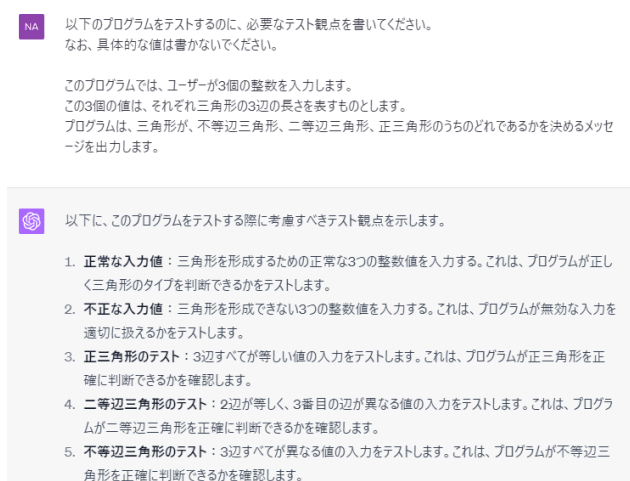


図 1

このように要求サンプルに対して、ChatGPT にテストプロセスの様々な質問を繰り返し、ChatGPT の回答からその活用の可能性を検討した。

3. 提案

本章では、各テストプロセスにおいて、ソフトウェアテストにおける ChatGPT の活用可能性を示していく。

3.1. テスト分析

ChatGPT は、言語処理技術に優れているため、テスト分析において要件仕様書やドキュメントを解析し、解析した結果をアウトプットすることができると考えられる。例えば、要求仕様書を解析することで、テスト観点の抽出が可能だ。

実例としては、図 1 のように要求を記載してテスト観点の抽出する旨を求めると、ChatGPT はテスト観点の代表的な例を出力した。あくまで代表的な例であるため、出力した内容をそのまま使用することはできないが、ヒントとしては利用できるであろう。

また、要求仕様書の曖昧な点、矛盾点、考慮点の抽出も可能だ。要求仕様書には多数の要求が含まれており、その要求の不備や内容の矛盾が含まれる場合がある。そのケースを抽出できるかどうかとも検証した。

実例としては図 2 の通り、マイヤーズの三角形問題に対して、曖昧な点、矛盾点、仕様として漏れている点の質問をしたところ、入力値の範囲、ユーザーが整数をどのような形式で入力するか、ユーザーが無効な入力をした場合のエラー処理が仕様に記載されていないことを挙げた。

このように ChatGPT を使うことで、仕様の漏れやあいまいな点といった要求仕様書の不備を自動で抽出することできるため、要求仕様書の品質向上にもつなげられるであろう。

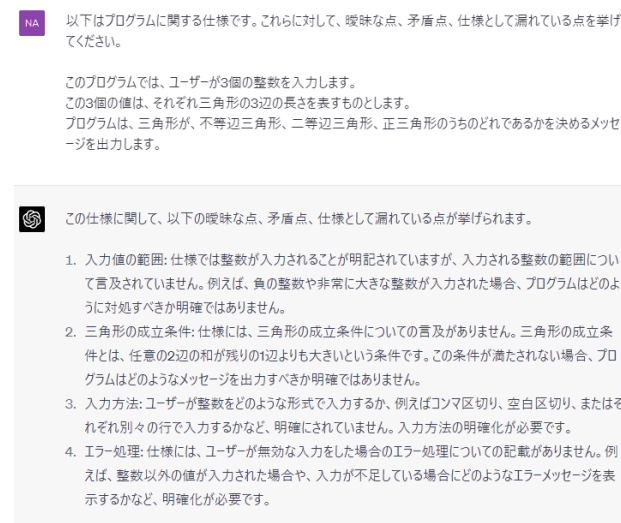


図 2

3.2. テスト設計

テスト設計においても、テスト分析と同様に、要求仕様書やテスト分析結果を ChatGPT にインプットとして使用することで、テストプロセスのサポートが可能と考えている。

例えば、テスト分析で挙げたテスト観点に対して、ChatGPT に網羅の基準とその網羅理由を挙げさせることができる。実例の一部が図 3 となる。

図 3 では、マイヤーズの三角形問題に対して事前に出力していた、「すべての辺の長さが正の整数である場合、正しい種類の三角形が出力されることを確認する」というテスト観点に対して、網羅の基準とその網羅理由を挙げる指示を出したところ、網羅の基準として「不等辺三角形、二等辺三角形、正三角形の 3 種類の三角形が出力されること」を挙げ、網羅理由は「三角形の種類を正確に判別できることを確認するために、各種類について最低 1 つの例をテストする。」という内容を挙げた。

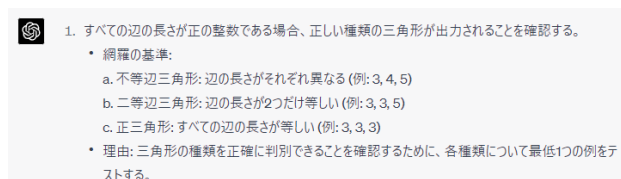


図 3

さらに、要求から適切なテスト設計手法を出力できるかどうかを検証した。

この検証に使用した要求サンプルは、エアコンの仕様を簡素化したもので、「温度を 1℃ずつ変えることができ、冷房運転時の設定範囲は 20 度～31 度、暖房運転時の設定範囲は 16 度～25 度」という内容だ。

この仕様に対してテスト設計手法を考えるように指示したところ、「境界値分析、同値分割、状態遷移テスト」の 3 つのテスト設計手法を当てはめられることを挙げた。

このように、ChatGPT が出力した網羅の基準やテスト設計手法などのテスト設計に必要な情報をヒントとして活用することで、自分で一から作成する場合よりテスト担当者の負担や時間をかけずにテスト設計を行うことができるであろう。

3.3. テスト実装

テスト実装フェーズでは、ChatGPT をテストケースの作成に活用できると考えている。ChatGPT は、テスト設計のアウトプットである「何をどのようにテストするか」という内容に基づいて、テストケースを生成することができる。

実例として図 3 の内容をもとに、ChatGPT でテストケースを出力した内容が図 4 である。図 3 で出力した三角形の網羅基準をもとに、入力値と出力値をテストケースとして ChatGPT が出力している。

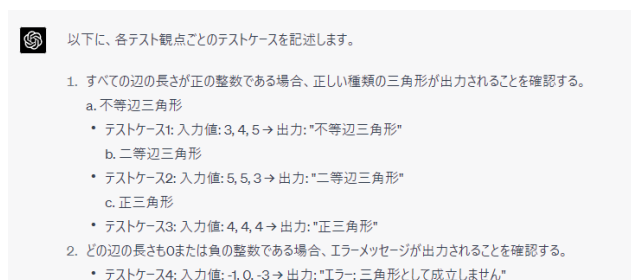


図 4

さらに図 4 で出力したテストケースに対してバリエーションを増やすよう指示をすると、入力値が異なるテストケースを ChatGPT は出力した。

以上から ChatGPT はテスト観点をインプットすることで、図 4 のようにその観点に基づいたテストケースを出力することができる。テストケースが足りない場合は、類似のテストケースを出力する指示を ChatGPT にすることで、その指示に従ったテストケースの出力もできる。

このことから既存のテスト仕様書をインプットにテストケースの拡張も対応でき、テスト設計工程で作成したテスト観点があればテストケースの生成も可能であろう。

また、テストケースの入力内容(前提条件、テスト手順、期待値等)を指定することで、テスト担当者が期待したテストケースを生成することも可能だ。図 4 のテストケースの入力内容を指定したケースが図 5 である。

テスト担当者が求めているテストケースのフォーマットがある場合は、それらを指定することで ChatGPT の出力したデータを形成する手間も省くことができるであろう。

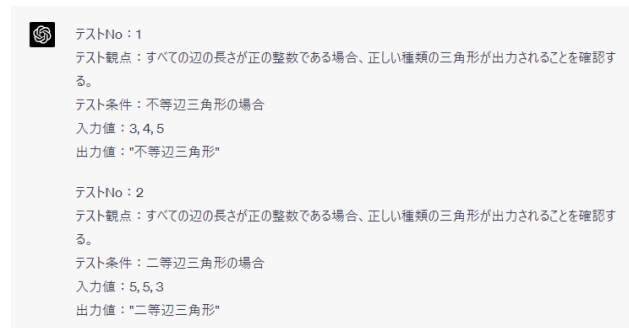


図 5

以上のように、ChatGPT を用いることで、テストプロセスの多くの部分をサポートし、テスト担当者の負荷を軽減することができる可能性が見えてきた。

ただし、ChatGPT の精度には限界があるため、テスト担当者は自らの知識や経験を持って、ChatGPT の回答を適切に評価する必要がある。もしテストプロセスに取り込むのであれば、全プロセスを ChatGPT に置き換えるのではなく、各プロセスの一部を ChatGPT に置き換えるところから始めるべきであろう。

4. 課題

本稿の検証内容に限った話ではないが、ChatGPT を活用する際にはいくつかの課題が存在する。

前述の内容と重複するが ChatGPT の精度には限界があるため、テスト担当者が ChatGPT の回答を適切に評価する必要がある。ChatGPT は質問に対して人間らしい回答をするが、情報の正確性や信頼性は保証されていない。人間が作成した成果物と同様に、ChatGPT の成果物に対してもレビューが必要となるであろう。

さらに、情報漏洩等のセキュリティ上の問題も懸念される。ChatGPT の FAQ に「API 経由で送られたお客様のデータを弊社のモデル改良のために使用することはありません。」[3]とあるため、API 経由であればこの問題は解決できそうだが、現状は機密情報や個人情報ではないデータを用いるべきであろう。

5. おわりに

本稿では, ChatGPT を使用してソフトウェアテストのプロセスをサポートする方法を提案し, その有効性を検証した.

結果として, ChatGPT を使用することで, テストケースの作成やテスト観点の抽出, 顧客要求の曖昧な点や矛盾点の特定などの作業のサポートできる可能性が見えてきた.

ただし, ChatGPT はリリースされて間もないサービスである. 活用やセキュリティなど解決すべき課題があるが, ChatGPT の可能性はまだまだ秘めていると考えているため, 今後の進歩に期待したい.

参考文献

- [1] ISTQB テスト技術者資格制度
Foundation Level Specialist シラバス 自動車ソフトウェアテスト担当者 日本語版 Version 2018.J03
https://jstqb.jp/dl/ISTQB-SyllabusFoundation_Version2018V31.J03.pdf
- [2] Glenford J. Myers, ソフトウェア・テストの技法 第 2 版, 近代科学社, 2006
- [3] ChatGPT APIFAQ
<https://help.openai.com/en/articles/7039783-chatgpt-api-faq>

文章埋め込み表現を用いた文章検索におけるコサイン類似度の補正

濱崎 利之

株式会社 NDKCOM

hamasaki.toshiyuki@ndkcom.co.jp

要旨

文章埋め込み表現を用いた文章検索におけるコサイン類似度の補正について提案する。提案手法では、検索語句と検索語句を多く含む検索対象とがなす角を、適切に補正することで、そのコサイン類似度を大きくするように構成している。提案手法は、検索語句としてスペースで区切られた断片的なキーワードを入力した場合に有用である。

1. はじめに

文章を高次元空間に埋め込み、ベクトルとして扱う文章埋め込み表現は、2つの文章の意味的な類似性を定量的に扱う手法であり、検索語句として入力した文章に対して意味的な類似文章の検索が可能となる。したがって、文章埋め込み表現はオペレータ業務のように検索を行って情報を提供する業務の作業効率化に貢献すると期待される。

しかしながら、文章埋め込み表現を用いた検索では、入力した検索語句が文章として十分な意味を成さない場合には文章の類似を適切に判断できない。オペレーター業務では、聞き取った内容を断片的なキーワードで入力することもあり、この場合は検索者の期待とかけ離れた検索結果が得られることがある。文章の類似性を定量的に扱う利点を生かしながら、キーワード検索の利点を持つ検索ができれば、より良い検索結果を提供できると考える。

本稿では、文章埋め込み表現を使用した検索にて、検索語句が文章として十分な意味を成さない断片的なキーワードを入力した場合であっても、検索者が期待する結果を得るための、類似度計算の補正方法について提案する。

2. 文章埋め込み表現

文章埋め込み表現とは、文章を高次元空間に埋め込んだ意味的な情報を持つベクトル (特徴ベクトル) のことである。文章埋め込み表現を生成する手法には、Universal Sentence Encoder (USE) [1] および BERT [2] があるが、本稿では USE を使用した。USE は Google の研究者によって 2018 年に Tensorflow Hub で公開された文書埋め込み表現生成手法である。USE は多言語に対応しており、異なる言語間でも定量的に類似した文章の判断が可能である。USE により生成された文書埋め込み表現は、512 次元空間上の点として表現され、また規格化されているため、512 次元球面上の点と考えることができる。

3. コサイン類似度

2つの文章埋め込み表現がどれだけ似ているかを定量的に判断する尺度としてはコサイン類似度が一般的である。2つの文章埋め込み表現を v, w とすると、コサイン類似度 S は

$$S = \frac{v \cdot w}{\|v\| \|w\|} = \cos \theta \quad (1)$$

で与えられる。ここで、最初の等式の $\cdot$ はベクトルの内積を表し、 $\|\cdot\|$ はベクトルの大きさ (L2 ノルム) を表す。また θ は 2つの文章埋め込み表現のなす角である。なお、USE の文章埋め込み表現は規格化されているため ($\|v\|=\|w\|=1$)、USE の文章埋め込み表現を用いる場合、式 (1) の分母を計算する必要は無い。コサイン類似度の値の範囲は $-1 \leq S \leq 1$ であり、2つの文章埋め込み表現のなす角 θ が小さいほど類似度が高い (ベクトルが同じ方向であるほど類似度が高い)。

4. 提案手法

2つの文章埋め込み表現の類似度を定量的に評価する方法として式(1)のコサイン類似度を使用した場合、検索語句としてスペースで区切られた断片的なキーワードを入力すると、入力した検索語句を含まない文章の類似度を高く評価してしまう場合がある。例えば、「ソフトウェアシンポジウム 仙台」を検索語句とした場合、「ソフトウェアシンポジウム」や「仙台」を含まないものが高い類似度となることがある。これは、検索語句として入力したものが文章として十分な意味を成さないことにより、適切な埋め込みができなためと考えられる。そこで、式(1)の類似度に対して下記手順で補正を行う：

1. 検索語句をスペースで分解する。分解して得られる個々の検索語句をキーワードと呼ぶ。
2. 検索語句に含まれるキーワードの数を N とする。
3. 検索対象に含まれるキーワードの数をカウントする。その数を x ($\leq N$ 、つまり重複は排除) とする。
4. N, x を使って次の式でコサイン類似度を補正する。

$$\alpha = 1 - 0.5 \times \frac{x}{N} \tag{2}$$

$$S' = \cos(\alpha \times \theta) \tag{3}$$

ここで、 S' は補正後のコサイン類似度、 θ は補正前のコサイン類似度のなす角である。

コサイン類似度では、2つのベクトルのなす角が小さいほど類似度が大きくなり、なす角が大きいほど類似度は小さくなる。そこで、キーワードを多く含むものは、なす角を最大で2分の1にすることで、類似度を大きくするように補正している。

5. 結果

独自に収集したヘルプデスクへの問い合わせ約4000件に対して、検索語句に「変更契約 金額」を入力した場合の補正の有無における検索結果上位5件を、表1および表2に示す。表1が補正無しの場合の検索結果であり、表2が補正有りの場合の検索結果である。

両者を比較すると、順位の1および2番目については差異は無いが、順位の3番目以降について異なる検索結

表 1. 補正無における「変更契約 金額」での検索結果

順位	類似度	内容
1	0.7219	変更契約を伴わない金額変更についてどのように入れたらいいか。
2	0.7052	変更契約画面で金額の変更はないはずなのに、今回請負金額の箇所に違う金額が入っている。
3	0.6220	変更契約を入れたいが、入れれない。
4	0.6082	支払条件の変更は、変更契約が終わっていても行うことができるか。
5	0.6055	業者を介さない形で当初の契約額を変更したいどうしたらいいか。

表 2. 補正有における「変更契約 金額」での検索結果

順位	類似度	内容
1	0.9278	変更契約を伴わない金額変更についてどのように入れたらいいか。
2	0.9233	変更契約画面で金額の変更はないはずなのに、今回請負金額の箇所に違う金額が入っている。
3	0.8920	変更契約画面で、今回請負金額の金額が間違っている。どうしたらいいか。
4	0.8878	変更契約画面で、登録をしようとした際に、「契約額の合計と請負金額が等しくありません。」というメッセージが表示される。
5	0.8872	変更契約画面で、再見積もりにより金額が減額になったため、その金額を登録をしようとした際に、「契約額の合計と請負金額が等しくありません。」というメッセージが表示される。

果が得られている。補正無しの場合の順位3番目の内容は、「変更契約を入れたいが、入れれない。」であり、「金額」というキーワードを含んでない。対して、補正有りの場合の順位3番目の内容は、「変更契約画面で、今回請負金額の金額が間違っている。どうしたらいいか。」であり、検索語句のキーワードを全て含むものとなっている。このように、補正後の検索結果では、検索語句に入力したキーワードを含むものが高い類似度を示すように補正されており、検査語句を含むものを検索結果に期待する検索者にとっては、望んだ結果に近づくと考えられる。

6. 課題

(検索結果の妥当性検証)

本稿で示した結果では、検索者がどのような検索結果

を期待して、検索語句を選んだのかを調査していない。つまり、「変更契約 金額」を入力した検索者にとって、補正有りの検索結果がどの程度妥当であったかは示していない。今後の課題として、実際の業務に即した形で本提案の補正を施した検索を行い、検索結果がどの程度妥当であるかを検証する必要がある。

(補正の計算式)

本稿で示した式 (2) の補正では、キーワードの数に対して直線的になす角を小さくする補正を行っている。なす角を小さくする方法については、直線的な補正ではなく、例えば、 $\tanh$ ようなシグモイド型の非線形関数を使用することも考えられる。また、なす角の補正は、式 (2) の通り、最大で 2 分の 1 まで小さくするよう構成しているが、この 2 分の 1 という値についても、検討の余地がある。さらに、本稿では類似度の計算にコサイン類似度を使用した。コサイン類似度以外の類似度計算方法 [3] について同様の補正が可能であるか、およびコサイン類似度と比較して結果にどのような変化があるかは一考する価値がある。より効果的な補正の計算式については今後の課題である。

7. おわりに

本稿では、文章埋め込み表現を用いた文章検索におけるコサイン類似度の補正について提案した。提案した補正方法では、検索語句が文章として十分な意味を成さないようなスペースで区切られた断片的なキーワードを入力した場合に、入力したキーワードを含む文章のコサイン類似度を大きくするように構成した。

提案した補正方法は、類似度を用いた検索の本来の仕組みを変更するものではない。つまり、検索語句に文章を入力した場合は、当該文章と類似したものを検索でき、類似度を用いた検索の機能を損なうことは無い。提案した補正方法は、類似度を用いた検索の仕組みの中でキーワード検索のような検索結果を得ることを可能にしており、これは、キーワード検索の検索結果を期待する場合に、より良い検索結果を提供できると考えられる。

類似度を用いた検索の利点は、細かい表現の差異や同

義語に依らず、意味が同じと考えらるものが検索できることである。類似度を用いた検索では、検索対象に対する理解が浅い人 (例えば業務の初心者) の知識の不足や記憶の曖昧さを補うことが可能となる。一方、キーワード検索の利点は、検索語句が明確に定まっている場合に、当該検索語句を含むものを検索できることである。キーワード検索では、検索対象に対する理解が深い人 (例えば業務の熟練者) が使用すると、求める検索結果を少ない労力で取得することが可能である。本稿の提案は、このような業務の初心者にも、業務の熟練者にも、心地よい検索体験を継ぎ目なく提供する試みでもあり、課題に記載した検索結果の妥当性について、業務の初心者および熟練者両者による検証は特に重要である。

参考文献

- [1] Daniel Cer, et al.: Universal Sentence Encoder *arXiv preprint*, arXiv:1803.11175.
- [2] Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding *NAACL-2019*, pp. 4171–4186 (2019).
- [3] 吉成末菜里, 岡本圭史: 要求文のベクトル化による要求文間の包含関係検出法の提案, 情報処理学会第 78 回全国大会講演論文集 2016 (1), pp. 427–428 (2016).

Generative AI とソフトウェア開発への応用の試行

酒匂 寛

Designers' Den Corp.
sako.hiroshi@gmail.com

要旨

2022 年中盤からいわゆる Generative AI (生成型 AI) が話題に上るようになった。様々な文章を生成し、長いドキュメントを要約し、ときにはプログラムまで生成してしまうという Generative AI は、ソフトウェアエンジニアに大きな影響を与えることが予想される。

本経験論文は、その Generative AI の応用の可能性を、様々な角度から試行した結果の報告である。なお主に 2022 年末に発表された ChatGPT を Generative AI として利用している。なお原稿提出の直前に、最新言語モデルの GPT-4 が発表されたので本報告もそれに沿って行われている。

1. はじめに

Generative AI が話題である。報じられるところでは、様々な機能を持っていてソフトウェア開発にも大きなインパクトを持っていそうである。しかし現時点 (2023 年 3 月) ではその可能性は十分に試行されているとは言い難い。

本事例報告は、日々新しい動きが見られる Generative AI がソフトウェア開発にどのような形で使えるかを、いくつかの視点から試行した結果を述べるものである。

なお筆者は Generative AI の利用に大きな可能性を感じているのだが、今回報告できる内容はその可能性のわずかな一部に過ぎないことをお断りしておく。

2. 報告内容について

本事例報告スライドは以下のような構成となっている。

- (1) Generative AI の現状
 - ・ 現状と様々なツールの説明
- (2) 直接的な応用
 - ・ コーディングアシスト
 - ・ レビューアシスト

- ・ 解析アシスト
- (3) より上位の応用
 - ・ 業務手順開発
 - ・ 業務手順改善
 - ・ プロンプトエンジニアリング
 - (4) 考察
 - ・ ソフトウェア開発のための Generative AI の応用
 - ・ 大きな開発フローと Generative AI
 - (5) まとめ

- (1) は概論である。
- (2) はソフトウェア開発のソースコードに関わる直接的な側面を取り上げて ChatGPT (GPT-4) を使ってインタラクションを行った試行結果を示している。
- (3) の内容は、もっと上位の業務分析に関わる部分の試行を行ったものである。こちらも ChatGPT (GPT-4) を用いている。もとよりごくごく狭いテーマしか扱えていないが、可能性の片鱗が示せていると思う。
- (4) では (2), (3) の内容を受けて考察を行っているが、特に AI への指示 (プロンプトと呼ばれる) もの使い方が重要であることを述べている。
- (5) 2023 年 3 月時点での試行を行った上での所感を述べている。

参考文献

- [1] OpenAI, GPT-4, <https://openai.com/research/gpt-4>, 2023
- [2] dair-ai, **Prompt-Engineering-Guide**, <https://github.com/dair-ai/Prompt-Engineering-Guide>
- [3] Gartner, **ChatGPT を越えてその先へ: 企業におけるジェネレーティブ AI の未来**, <https://www.gartner.co.jp/ja/articles/beyond-chatgpt-the-future-of-generative-ai-for-enterprises>

GPT-3 と LlamaIndex を用いた生成知識の比較

塚田 良央	佐々木 亮	佐々木 千春	漆原 憲博
株式会社ジェーエフピー	株式会社ジェーエフピー	株式会社ジェーエフピー	株式会社ジェーエフピー
tsukada_yoshio@jfp.co.jp	ssk@jfp.co.jp	sasaki1000@jfp.co.jp	japanfp@jfp.co.jp

要旨

製造業やソフトウェア産業では、国際規格を遵守した製品開発が要求されることが多い。一例として、ISO 21448 は自動車の意図した機能の安全性 (SOTIF) に関する国際規格であり、自動運転の分野で特に重要な規格である。しかし、ISO 21448 の文章は抽象的であり、自動車開発に精通していない者にとっては、自動車開発の具体的な知識との関係を明確に把握できない。

そこで、ISO 21448 の知識を読み込んだ質疑応答システムを構築することにより、ISO 21448 の理解が容易になり、自動車開発の現場で活用されることが期待される。

本研究では、ISO 21448 の知識を読み込んだ質疑応答システムを GPT-3 と LlamaIndex を用いて構築した (提案手法)。

このシステムと ChatGPT (従来手法) に ISO 21448 に関する質問を入力し、回答を比較した。そして、正確さと分かりやすさという 2 種類の観点から、主観評価を行った。主観評価で、正確さと分かりやすさがともに提案手法が従来手法に劣るという結果となった。

1. はじめに

1.1. 研究の背景

経済のグローバル化により、共通のルールのもとでビジネスをすることが求められるようになった。製造業でもソフトウェア産業でも、国際規格を遵守した製品開発が要求されている。開発の現場では、これらの国際規格に習熟し、適切に対処することが必要である。そのためには、知識獲得と運用の支援があることが望ましい。

一例として、ISO 21448 [1] は自動車の意図した機能の安全性 (SOTIF) に関する国際規格であり、自動運転の分野で特に重要な規格である。しかし、ISO 21448 の文章は抽象的であり、自動車開発に精通していない者にとっては、自動車開発の具体的な知識との関係を明確に把握できない。

本論文の第 1 オーサーは SOTIF の知識獲得 (学習) を 4 カ月行った。結果として規格書の理解はできたと思ったが、ただし自動車の安全制御に関する具体的な要求仕様や機能仕様のことは学ばなかった。このような知的な状態を、知識を構造として見た場合には、知識の抽象的な構造 (部分) は分かったが、具体的な構造 (部分) が分からなかったという状態と言える。

折しも、ChatGPT [2]、Bing AI [3]、Bard [4] の発表により、著者たちは、抽象と具体の知識の構造にこれらのツールが良き貢献をしてくれるのではないかと期待した。ChatGPT は大規模言語モデル GPT-3 [5] をもとに、質問と回答のデータを強化学習したものである。このようなツールが知識の構造に関してまでサポートできるか、まずはトライすることとした。

ChatGPT はインターネットから収集した大量のデータをもとに回答する。ただし、インターネットで公開されているものを除き、ISO 21448 への専門的な知識は入力されていないと考えられる。

ChatGPT などの大規模言語モデルがインターネットの情報をもとにしているのに対して、インターネットに接続されていない、もしくは制限されている企業や自治体など組織内部の非公開データを用いることが提案されている [6]。

ところで、GPT-3 に専門的な知識を追加して質疑応答システムを作成するソフトウェアとして LlamaIndex [7, 8, 9, 10] が知られている。LlamaIndex は、まず、文字列として入力された知識から、インデックスを作成する。続いて、質問が与えられると、そのインデックスと GPT-3 を用いて推論し、回答を出力する。

1.2. 提案手法

そこで、本研究では、GPT-3 と LlamaIndex を用いて、ISO 21448 の原文と日本語訳を知識として読み込ませ (提案手法)、質疑応答システムを構築した。

これにより、ISO 21448 についての専門的な知識を、質疑応答システムを通じて獲得できることが期待できる。すなわち、ISO 21448 の抽象的な概念に、具体的な知識

を関連付けることが可能になることが期待される。

2. 実験

2.1. 実験手法

本研究では、ISO 21448 に関連する同一の質問に対して、提案手法の回答と、ChatGPT（従来手法）の回答を、正確さと分かりやすさの観点から比較した。

本研究では、提案手法で使用する知識データとして、ISO 21448 の英文と、その日本語訳を用いた。

ISO 21448 の原文は、PDF ファイルとして販売されている。まずは、これをテキストファイルに変換した。続いて、本文の条（clause）と、附属書（annex）の節ごとに、空行を挿入した。LlamaIndex では空行をトークンの区切りとするためである。

一方、ISO 21448 の原文を DeepL [11] で日本語訳し、さらに、その日本語訳を手作業で訂正した。これをテキストファイルに変換し、本文の条と、附属書の節ごとに、空行を挿入した。また、文ごとに半角スペースを挿入した [6]。これは、GPT-3 が、英文の文の区切りをピリオドの後のスペースを認識して行うことに合わせるためである。

これら 2 個のテキストファイル（英文と日本語訳）を、知識として LlamaIndex に読み込ませた。

開発環境には Google Colaboratory [8, 9, 12] を用いた。

実験は 2023 年 3 月 13 日時点のソフトウェアで行った。

ChatGPT と、ISO 21448 を読み込ませた LlamaIndex に質問するため、5 個の質問文を用意した。質問文は 2.2 節に示す。

従来手法と提案手法のそれぞれの回答について、正確さと分かりやすさという 2 種類の観点から、1 点から 5 点までの得点で評価した。ただし、評価は主観的なものとした。本研究では現場で使えるかを目的にしたので、おおよその感覚を重視した。次のような印象による点数である。

正確さ

- 1: 全く誤り。
- 2: 知識が不足している。
- 3: 知識に少々不足がある。
- 4: 知識はおおむね正確である。
- 5: 知識は正確である。

分かりやすさ

- 1: とても難解である。
- 2: 難解である
- 3: 理解できる。
- 4: 容易に理解できる。
- 5: とても容易に理解できる。

実際の評価は、ISO 21448 の知識をよく知る、本論文の著者のうちの 1 名が行った。

2.2. 実験結果

各質問文の回答と主観評価を以下に示す。なお、回答の中には、文の途中で途切れているものがある。これは、回答時の字数制限によるものである。ChatGPT では「続けてください。」などのプロンプトによって回答の続きを出力させることができるが、LlamaIndex にはこの続きの機能がない。回答が長くなりすぎることと、続きの機能が両者揃わないことから、1 回のやりとりで得られた回答のみを採用した。従って、フルの状態と比較することはできないという限界がある。

質問 1

ISO 21448 において「意図した機能」とは何ですか？

従来手法の回答

ISO 21448 は、自動車の安全性に関する国際標準規格であり、SOTIF (Safety Of The Intended Functionality、意図した機能の安全性) の概念を中心に構成されています。

ISO 21448 における「意図した機能」とは、自動車のシステムまたは部品の設計者が意図した機能を指します。つまり、ある機能が設計された目的や仕様に従って正常に動作することが想定されている機能です。

例えば、自動車の自動ブレーキシステムは、車両が前方の障害物に接近した際に自動的にブレーキをかけることが意図されています。この場合、自動ブレーキシステムの「意図した機能」とは、車両が前方の障害物に接近した場合にブレーキがかかることです。

SOTIF では、自動車のシステムや部品の設計者は、意図した機能が正常に動作することを保証するだけでなく、意図しない動作や異常な動作が発生しないように設計することが求められます。

提案手法の回答

ISO 21448 では、「意図した機能」とは、車両レベルで定義される機能の詳細な記述であり、ドライバーまたはユーザーと自動化システムの互いの役割を定義する、レベル 0(自動運転なし)からレベル 5(完全自動運転)までの相互に排他的な運転自動化レベルのセットを指します。

主観評価

従来手法の正確さ: 1

従来手法の分かりやすさ: 2

提案手法の正確さ: 2

提案手法の分かりやすさ: 1

本論文著者の見解 (期待と相違点)

ISO 21448 では、意図した機能 (intended functionality) とは specified functionality であると定義されている。上記のいずれの回答も、specified functionality については語っていない。我々 (人間) が意味の議論をする場合には、原文を忠実にひもとこうとする (一つのアプローチではあるが)。あるいは、また、原文をよりどころにして議論する。他方、ChatGPT と LlamaIndex の場合は、そのようなアプローチはとっていない。全体をまとめたような回答に感じられる。このような回答を得る方法について評価する技術は著者らの目的を超えている。

以下の質問については、本論文著者の見解は割愛している。できるだけ正しい知識の結果を期待する。

質問 2

ISO 21448 の目的は何ですか？

従来手法の回答

ISO 21448 は、自動車安全性に関する国際標準化機構(ISO)が策定した「SOTIF (Safety of the Intended Functionality)」のためのガイドラインです。

ISO 21448 の主な目的は、自動車の新しい安全性機能や技術が使用される場合に、設計者や製造業者が「意図された機能の安全性 (Safety of the Intended Functionality, SOTIF)」を適切に評価し、確保することです。

SOTIF は、車両に搭載されたシステムの設計上の欠陥が原因で発生する可能性のある事故を防ぐために、設

計者や製造業者が考慮すべき概念です。SOTIF は、自動車の安全性に対する新しいアプローチを提供し、従来の安全性機能とは異なり、車両の機能が意図したとおりに機能し続けることを確保します。

ISO 21448 は、自動車の設計・開発におけるリスクマネジメントプロセスの標準化を提供することで、より高い安全性の確保に役立っています。

提案手法の回答

ISO 21448 の目的は、自動運転システムのセーフティを保証するためのプロセスを提供することです。このプロセスは、開発プロジェクトの初期段階から、SOTIF 活動のすべての責任を確認し、開発関係者間で適切な技術情報が交換されるようにすることを目的としています。

主観評価

従来手法の正確さ: 3

従来手法の分かりやすさ: 3

提案手法の正確さ: 2

提案手法の分かりやすさ: 4

質問 3

ISO 21448 と ISO 26262 の違いは何ですか？

従来手法の回答

ISO 21448 は、自動車の安全性に関する国際標準である SOTIF (Safety of the Intended Functionality) の規格です。これは、自動車のシステムが意図した機能を安全かつ正確に実行することを確保するために、設計、開発、評価のためのプロセスを定義します。SOTIF は、自動車のシステムが誤動作を引き起こす可能性がある場合に焦点を当てています。

一方、ISO 26262 は、自動車の機能安全に関する国際標準で、自動車に搭載されたエレクトロニックシステムの安全性を評価し、設計、開発、評価、検証、および保守のためのプロセスを定義します。これは、自動車のシステムが誤動作を引き起こすことがないようにするために、安全性要件を定義しています。

ISO 21448 と ISO 26262 は、ともに自動車の安全性に関する国際標準であり、それぞれ異なる側面を取り上げています。ISO 21448 は、システムが意図した機能を正確

に実行することを確認することに重点を置いています。一方、ISO 26262 は、システムが誤動作を引き起こさないようにすることに重点を置いています。

提案手法の回答

ISO 26262 は、自動車用電子/電気システムの安全性を確保するための規格です。ISO 21448 は、自動車用安全システムの設計と開発に関する規格です。ISO 26262 は、安全性を確保するためのプロセスを定義していますが、ISO 21448 は、安全性を確保するためのプロセスを定義および改善するためのガイダンスを提供しています。

主観評価

従来手法の正確さ: 2

従来手法の分かりやすさ: 3

提案手法の正確さ: 3

提案手法の分かりやすさ: 2

質問 4

ISO 21448 ではソフトウェアの安全をどう扱っていますか？

従来手法の回答

ISO 21448 は、自動車の機能安全に関する国際規格であり、ソフトウェアの安全性に関する指針も含まれています。ISO 21448 では、ソフトウェアの安全に対する要件と評価方法が規定されています。

具体的には、ISO 21448 では、以下のようなソフトウェアの安全性に関する要件が規定されています。

- ・ソフトウェアの設計や実装において、エラーを防止するための方法を適用すること
- ・ソフトウェアの信頼性を評価するための分析手法を適用すること
- ・ソフトウェアの安全性を確保するためのテストを実施すること
- ・ソフトウェアの開発プロセスを文書化し、監視すること
- ・ソフトウェアの品質管理を行うこと

ISO 21448 では、これらの要件を満たすことが、ソフトウェアの安全性を確保するために必要なこととされています。

す。また、これらの要件を満たすための方法や評価方法についても、規定されています。

提案手法の回答

ISO 21448 では、ソフトウェアの安全を、機能安全と SOTIF (安全性、オペレーション、及び確実性) の 2 つの観点から扱っています。機能安全は、指定された動作の実装に関する系統的な問題や、E/E 要素のランダムなハードウェア故障を扱います。SOTIF の分析では、機能的な不具合、その潜在的な誘発条件、車両挙動への影響に着目しています。さらに、合理的に予見可能な間接的(続かず)

主観評価

従来手法の正確さ: 1

従来手法の分かりやすさ: 3

提案手法の正確さ: 1

提案手法の分かりやすさ: 3

質問 5

ISO 21448 において、機械学習を用いるうえでの留意事項について教えてください。

従来手法の回答

ISO 21448 は、自動運転車両のための安全性の評価ガイドラインであり、機械学習を含む人工知能技術の安全性に関する要件も含まれています。以下に、ISO 21448 において機械学習を用いる際に留意すべき事項を示します。

1. データの品質の評価: 機械学習によるシステムは、学習に用いるデータの品質に依存します。ISO 21448 では、データ品質の評価手順に関する要件が記載されています。学習に使用するデータは、信頼性、一貫性、正確性、および安全性に関して評価される必要があります。

2. 機械学習の透明性: 機械学習によるシステムは、通常、ブラックボックスと呼ばれ、内部の処理が不透明であるため、システムの内部の動作が説明できない場合があります。ISO 21448 では、システムの透明性を確保する

方法についての要件が定められており、システムの設計と開発の段階で透明性を確保するための方法が提供されます。

3. 機械学習の評価: 機械学習によるシステムは、設計時に十分に評価される必要があります。ISO 21448 では、機械学習によるシステムの設計、実装、評価に関する要件が定められており、評価方法についての基準が示されています。

4. 機械学習による不確実性の取り扱い: 機械学習によるシステムは、不確実性を伴う可能性があります。ISO 21448 では、機械学習によるシステムの不確実性を評価するための要件が定められており、不確実性を管理する方法に関する指針が示されています。

5. 人間との協調: 機械学習によるシステムは、人間との協調が必要な場合があります。ISO 21448 では、機械学習によるシステムの協調性に関する要件が定められており (続かず)

提案手法の回答

ISO 21448 では、機械学習を用いる場合、以下の留意事項を検討する必要があります。

1. 機能性及びシステム設計: SOTIF プロセスにおいて重要な役割を持つ関連する ODD を含むユースケースの仕様は、ML ベースの機能のトレーニング、検証、テストのためのデータセットを収集し作成するためにも重要です。

2. トレーニングデータセットの質: 既知のシナリオや非危険なシナリオで適切に機 (続かず)

主観評価

従来手法の正確さ: 3

従来手法の分かりやすさ: 3

提案手法の正確さ: 1

提案手法の分かりやすさ: 1

平均点

主観評価の平均点は、

正確さの観点では、

従来手法が 2.0 点、

提案手法が 1.6 点 であった。

分かりやすさの観点では、

従来手法が 2.8 点、

提案手法が 2.4 点 となった。

3. 議論

提案手法の正確さの主観評価が伸びなかった理由として、情報源である ISO 21448 の次の特徴が考えられる。ISO 21448 は箇条書きと表が多く、LlamaIndex による情報の抽出に適していなかった可能性がある。

また、本研究の期待として、ISO 21448 の抽象的な記述から、自動車開発に必要な具体的な知識が得られることを期待していた。しかし、大規模言語モデルの性質として、そのような変換は行われないものであり、主観評価が低い点数にとどまったものと思われる。

4. 将来の研究

GPT-3 の後継の大規模言語モデルとして GPT-4 [13] が発表された。すでに GPT-4 を言語モデルとする ChatGPT が利用できるようになっている。(ただし、本研究で使用した ChatGPT の言語モデルは GPT-3 である。)

GPT-4 を言語モデルとする ChatGPT は、GPT-3 を言語モデルとした場合に比較して、正確さと分かりやすさが向上する可能性がある。LlamaIndex が GPT-4 に対応した場合も同様である。

実際に、LlamaIndex で GPT-4 を利用する [14] ことが可能であるが、2023 年 5 月 9 日時点で、API がタイムアウトすることがあるという問題があり、導入の障害になっている。

GPT-4 を言語モデルとする ChatGPT と、GPT-4 を言語モデルとする LlamaIndex の回答の一例を、以下に示す。

質問 1

ISO 21448 において「意図した機能」とは何ですか？

GPT-4 を言語モデルとする ChatGPT の回答

ISO 21448 は、道路車両の「安全関連部品とシステム」の機能安全性に関する規格です。この規格は、特に自動運転システムの安全性を評価するために使用されます。

「意図した機能」は、そのシステムまたは部品がどのように機能することが期待されているか、設計者や製造者

がどのようにその機能を期待しているかを指します。たとえば、自動ブレーキシステムの「意図した機能」は、ある特定の状況（車が障害物に近づいている場合など）で車を停止させることかもしれません。

ISO 21448 は、この「意図した機能」が適切に機能し、予想外のシナリオや故障時にも安全を確保することを目指しています。この規格は、設計、製造、テストの各段階で、システムがその「意図した機能」を達成し、同時に安全性を維持するためのプロセスと手順を提供します。

GPT-4 を言語モデルとする LlamaIndex の回答

「ISO 21448 において「意図した機能」とは、システムが設計された目的に従って正常に動作することを指します。これは、システムが適切に設計され、運用され、保守されることによって達成される安全性を意味します。意図した機能は、システムが予期される使用状況下で安全に動作することを保証するために、機能安全と SOTIF (Safety of the Intended Functionality) の概念と密接に関連しています。」

また、LlamaIndex のインデックスの構造として、以下のものが用意されている [15]。

- List Index
- Table Index
- Tree Index
- Vector Store Index
- Structured Store Index
- Knowledge Graph Index

本研究では、API 使用量が少ないことから、Vector Storage Index を用いた。他のインデックス構造を用いた場合には、回答の正確さと分かりやすさが向上する可能性がある。

提案手法の正確さと分かりやすさが向上した場合に、どのような応用が考えられるだろうか？ まず、質疑応答システムによる回答の真偽を自分で判断できる有識者が使用することが考えられる。万が一、知識が偽であっても、有識者は修正できるからである。

さらに、提案手法の正確さが十分に信頼に足ると判断された場合には、専門的な知識がなくても、提案手法を使用できる可能性がある。ただし、十分に信頼に足るという判断は困難である。

5. 結論

GPT-3 と LlamaIndex を用いて、ISO 21448 の英文と日本語訳を知識として読み込ませ、質疑応答システム (チャットボット) を作成した。ChatGPT と比較したところ、主観評価で、正確さと分かりやすさがともに提案手法が劣るという結果となった。

なお、英文と日本語を比較すると、LlamaIndex の場合も ChatGPT の場合も、英文の方が正確であったという印象を持った。日本語での質問は英語に訳されてさらに回答されているためと思われる [16]。

ChatGPT は知識に関する推論機構がないと言われていた [16]。抽象的な概念に具体的な知識を関係付けること、すなわち概念の整理あるいは知識の推論が、これからの大規模言語モデルには望まれる。

参考文献

- [1] ISO/FDIS 21448:2022(E) Road vehicles — Safety of the intended functionality, 2022
- [2] OpenAI, Introducing ChatGPT, <https://openai.com/blog/chatgpt>, 2022
- [3] Bing AI, <https://www.bing.com/?ai>, 2023
- [4] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D.. (2020). Language Models are Few-Shot Learners.
- [5] James Manyika, An overview of Bard: an early experiment with generative AI, <https://ai.google/static/documents/google-about-bard.pdf>
- [6] 中川裕志, 人工知能と社会 (上)「仕事」の再定義 必須に, 日本経済新聞, 2023 年 5 月 8 日
- [7] Jerry Liu, LlamaIndex, https://github.com/jerryliu/gpt_index, 2022
- [8] npaka, GPT Index で専門知識を必要とする質問 応 答 チャット ボ ッ ト を 簡 単 作 成 , <https://note.com/npaka/n/nd23bdf33d929>, 2023
- [9] npaka, LlamaIndex クイックスタートガイド, <https://note.com/npaka/n/n8c3867a55837>, 2023

[10] shimajiroxyz, gpt-index で日本語 PDF を読み込む【Python】
<https://qiita.com/shimajiroxyz/items/aa1e22e19d690a3944ba>, 2023

[11] DeepL Translate,
<https://www.deepl.com/translator>

[12] Google Colaboratory,
<https://colab.research.google.com/>

[13] OpenAI, GPT-4 Technical Report, 2023

[14] Defining LLMs,
https://gpt-index.readthedocs.io/en/latest/how_to/customization/custom_llms.html

[15] Indices,
<https://gpt-index.readthedocs.io/en/latest/reference/indices.html>

[16]「生成AIとどうつきあうか」、松原仁、視点・論点 (NHK E テレ)、放送日 2023 年 5 月 17 日

技術的負債に関する課題票の単語分散表現を用いたテキスト分類

田口 舞奈

和歌山大学システム工学部

taguchi.maina@g.wakayama-u.jp

木村 祐太

和歌山大学システム工学研究科

kimura.yuta@g.wakayama-u.jp

大平 雅雄

和歌山大学システム工学部

masao@wakayama-u.ac.jp

要旨

技術的負債は、納期や予算などの制約により実装や設計の品質を妥協した結果、後に発生する追加のコストを指す。近年のオープンソースソフトウェア（OSS）開発では、課題管理システムを用いて技術的負債を管理するプロジェクトが増えてきているものの、不具合報告などと区別されることなく管理されることがいまだ一般的である。技術的負債と不具合とは、原因および影響範囲の調査や修正に要するコスト、優先順位付けや担当者の割り当て方法など、様々な点で異なる場合があるため、それぞれを適切に管理するためには課題票が登録された時点で区別されていることが望ましい。

本研究の目的は、課題管理システムに登録された時点で課題票の内容が技術的負債に関するものかどうかを判別するための分類モデルを構築することである。課題票が登録された時点で入手可能な情報として、課題票のタイトルと本文に含まれるテキストデータがある。本論文では、これらテキストデータを5種類（*BoW*, *TF*IDF*, *Word2Vec-Wiki*, *Word2Vec-SO*, *GloVe-Wiki*）の単語分散表現に変換し、3種類の分類器（ランダムフォレスト、ロジスティック回帰、サポートベクターマシン）を用いて分類モデルを構築することで、それぞれの分類性能を定量的に示すとともに、単語分散表現が技術的負債に関連する課題票の分類に与える影響を明らかにする。実験の結果から、作成した分類器を目的に応じて使い分けることが有効であるとわかった。

1 はじめに

ソフトウェア開発では、納期や予算などの制約などにより、設計や実装の品質を犠牲にする場合がある。最適ではない設計や実装をおこなった結果、後に保守作業の効率を低下させ余分のコストを発生させる。Cunningham [1] は、このような事象のことを、会計用語を交えて技術的負債（Technical Debt: TD）と表現した。

技術的負債は、放置する時間が長引くほど保守コストが増大してしまうため早期の発見と除去が重要であり、開発者・研究者双方の立場から高い関心を集めている [2]。これまで、技術的負債の存在を開発者がソースコードコメントとして表明する Self-Admitted Technical Debt (SATD) に関する研究 [3–13] が盛んに行われてきたが、特に近年では、課題管理システムを用いて課題票として管理される技術的負債の研究 [14, 15] が注目されつつある。

課題票として管理される技術的負債には、課題管理システムに登録される時点で技術的負債であることが判明している場合と、課題票として登録された後の開発者同士の議論を通じて技術的負債であることが判明する場合がある。前者は、開発者がすでに技術的負債であることを認識しているため、SATDと同様のものとして管理することが可能である。一方後者は、課題票の登録時点では、不具合報告や改善要求などと区別しにくいいため、開発者が技術的負債であると認識するまでに時間を要することがある。さらに、技術的負債と不具合とは、原因および影響範囲の調査や修正に要するコスト、優先順位付けや担当者の割り当て方法など、様々な点で異なる

場合があるため、それぞれを適切に管理するためには課題票が登録された時点で区別されていることが望ましい。

そこで本論文では、課題管理システムに登録された時点で課題票の内容が技術的負債に関するものかどうかを判別するための分類モデルを構築する。課題票が登録された時点で入手可能なテキストデータを 5 種類 (BoW, TF-IDF, Word2Vec-SO-corpus, Word2Vec-Wiki-corpus, Glove-Wiki-corpus) の単語分散表現に変換し、3 種類の分類器 (ランダムフォレスト, ロジスティック回帰, サポートベクターマシン) を用いて分類モデルを構築することで、それぞれの分類性能を定量的に示すとともに、単語分散表現が技術的負債に関連する課題票の分類に与える影響を明らかにする。

2 動機

本章では、本研究の動機を説明するために、課題票として管理される技術的負債の研究意義について述べた後、課題票として管理される技術的負債に関する研究を紹介し、最後に先行研究 [15] と本研究の位置付けの違いを明確にする。

2.1 課題票として管理される技術的負債の研究意義

技術的負債は開発者・研究者双方の立場から注目が集まっており、技術的負債の検出ツールなどの開発が盛んに行われている [2]。これまで開発されたツールは技術的負債の可能性のあるものに関して大量の検出結果が得られる。しかし、偽陽性や開発者が重視しない結果も多数含まれるため、積極的なツールの利用には至っていない [16]。そのため、ソースコードコメント中に表明した開発者が返済すべきであると認識している技術的負債 (SATD) に関する研究 [3–13] が盛んに行われてきた。

特に近年では、課題管理システムを用いて課題票として管理される技術的負債の研究 [14, 15] が注目されつつある。SATD は開発者個々人がソースコードコメント中で表明しているものであり必ずしもプロジェクトとして管理対象となる保証はないが、課題票として管理される技術的負債はプロジェクト全体としての関心事であると捉えられるため、相対的に SATD よりも重要性が高いものである可能性が高い。実際、課題票として管理される技術的負債のうち SATD を対象としたものは 28.7% [14] しかなく、SATD の大多数はプロジェクトとしての管理対象とはなっていない。また、製品のリリース後も 26.3%

から 63.5% しか SATD は返済 (除去) されない [3] ことから、SATD の多くはプロジェクトとして重大な関心事ではない可能性がある。そのため今後は、SATD ではなく課題票として管理される技術的負債について研究する価値が高いと言える。特に、課題票として管理される技術的負債を大量に収集しその特徴や返済方法を体系化することは、プロジェクトとして重要度の高い技術的負債の検出するツールの開発や、技術的負債の効率的・効果的な返済 (自動化など) に貢献できる可能性がある。

2.2 技術的負債に関する課題票の分類

Xavier らの調査 [14] によると、GitHub にホスティングされたスター数上位 5000 件のプロジェクトのうち、課題票に技術的負債を示すラベルを付与して明示的に管理しているプロジェクトはわずか 23 件のみである。さらに、技術的負債を示すラベルを課題票に 10 件以上付与しているプロジェクトは、23 件のうち 4 件とさらに少なくなる。課題票に付与されたラベルに基づいて技術的負債を抽出することは技術的に容易であるが、特徴分析や体系化をおこなうには絶対数が不足している。ただし、技術的負債を課題票として管理しているプロジェクトがごく少数しか存在しないのではなく、技術的負債に高い関心が集まったのは比較的最近の現象であり、技術的負債の明確かつ具体的な定義は存在しないため、ほとんどのプロジェクトは技術的負債と不具合とを明確に区別していないのが実態である。

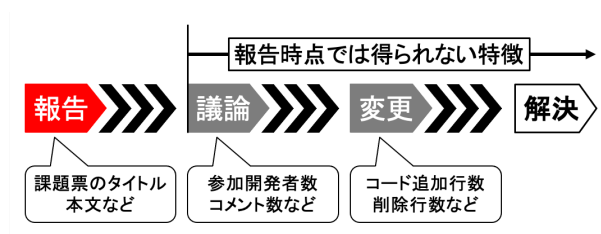
木村らは、ラベルを用いて明示的に管理していないが実質的には技術的負債を取り扱っている課題票をより広範囲に収集することを目的として、技術的負債に関する課題票とその他の課題票を分類する手法 [15] を提案している。[15] では、技術的負債を示すラベルが付与された課題票を収集し構築した分類モデルが、Precision 0.995, Recall 0.903 という高い分類性能を示しており、ラベルは付与されていないが実質的には技術的負債を扱っている課題票を今後大量収集できる可能性がある。

2.3 本研究の位置付け

木村らの分類モデル [15] は、課題票として管理される技術的負債の特徴や返済方法を研究するためのデータ収集を支援するために、ラベルは付与されていないが実質的に技術的負債を扱っている課題票とその他の課題票を分類することを目的としている。そのため、分類モデルの構築においては、すでに返済が完了した技術的負債

クリーニングの対象	除去のための正規表現	置換後のテキスト
Issue の ID	#[0-9]+	issueid
コードスニペット	“ “? ”	code snippet
インラインコード	`.*?`	inline code
画像	 !\[.*?\]\(.*?\)	image url
HTML タグ	<img(/)?.*?>	置換語なし
URL	http(s)?://.*?(\\s\$) \[.*?\](.???)	link url
ファイル名	[^\\s]+\.[a-z]+(\\s\$)	filename
ディレクトリパス	/?[\\.a-zA-Z0-9]+/.*(\\s\$)	pathname
日付	0-0-0(T\\s)	date number
時間	0:0:0(Z\\s)	time number
ユーザ名	@[a-zA-Z0-9]+	username
関数名	[^\\s]+\\(.*?)	functionname
記号	[=\\+\\-&%#\$,,:;~*\\'\\.\\/— □ ▢ ▣ ▤ ▥ ▦ ▧ ▨ ▩]+	置換語なし
数字	[0-9]+	0

一方，本研究で構築する課題票の分類モデルは，課題票が登録された時点でその内容が技術的負債に関するものかそうでないのかを分類する．技術的負債の早期発見を支援することを意図したものであり，[15] の分類モデルとは目的が異なる．また，分類モデルの構築においては，木村らの分類モデルとは異なり，課題票が登録された時点で取得可能な特徴量（図 1）のみを用いて分類モデルを構築する必要がある．課題票が登録された時点で取得可能な特徴量としては，課題票のタイトルや本文に含まれるテキストの特徴と課題票の登録者（報告者）の特徴などが挙げられるが，本論文では分類モデルの基本性能を確認するために課題票に含まれるテキストデータのみを用いて分類モデルを構築する．その他の特徴量を用いた分類モデルの構築と分類性能の検証については今後の課題とする．



3 分類モデルの構築

3.1 前処理

3.1.1 クリーニング

課題票には、テキストデータを分散表現で扱う際にノイズとなるデータが含まれているため、正規表現によりノイズを除去し特定の単語の置換を行う。表1に、除去したノイズと置換した単語を示す。また、全てのテキストを小文字に変換し、大文字と小文字で単語を区別しないようにする。

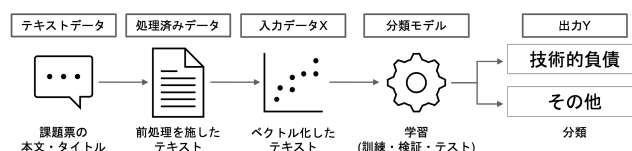


図 2. 分類モデル作成の手順

3.1.2 トークン化

クリーニングを行なったテキストデータを、単語ごとに分割することでトークン化する。トークン化には、spaCy 3.4.4¹を利用する。トークン化に使用する辞書は、spaCy が公開している `en_core_web_lg` とする。トークン化した際に、それぞれのトークンは品詞ごとに分類され、品詞を記した POS (Part-of-speech) タグが付与される。POS タグが句読点 (PUNCT)、記号 (SYM)、空白 (SPACE)、その他 (X) のトークンはノイズとなるため除去する。さらに、辞書に登録されていない単語 (Out-of-Vocabulary: OOV) であるトークンは「oov」に置換する。また、英文では「it」や「the」のような単語が頻出する。ストップワードと呼ばれるこれらの頻出単語はノイズとなるため除去する。

3.1.3 レンマ化

英文では、動詞の活用や名詞の単数形と複数形などにおいて単語が変化するため、テキストデータのトークン化のあとは spaCy を用いて辞書の見出し語に統一するレンマ化を行う。レンマ化は、トークン化と同様に spaCy を用いて `en_core_web_lg` モデルを使用して変換する。

3.2 ベクトル化

上述の手順でノイズを除去したテキストデータを、計算機で扱える形式にするためにベクトル化を行う。具体的には、課題票に出現した単語のベクトルの平均値を求める。ベクトル化にあたっては、以下の5つの手法を採用した。

bag of words: BoW

BoW は、全テキストデータに含まれる単語の種数を次元数とし、文書 (本研究では課題票) ごとに含まれる単語の数を要素とした単純な分散表現である。

TF*IDF

TF はある文書におけるある単語の出現頻度、IDF は全文書中のある単語を含む文書の珍しさ、つまり逆文書頻度を表す。BoW に TF と IDF を掛け合わせた重み付けをおこなうことで、文書ごとに異なる単語の重要度をより表現できる。

Word2Vec

Word2Vec [17] は、単語分散表現の代表的な手法と

されており、2層のニューラルネットワークからなるシンプルな構造を持つ。中心語から周囲語を確率的に予測する skip-gram モデルと、周囲語から中心語を確率的に予測する continuous BoW (CBOW) モデルの2種類の構造が実装されている。本論文では、頻度の低い単語に対して有用でありより精度が高いとされる skip-gram モデルを利用する。BoW や TF*IDF と大きく異なる点として、コーパスと呼ばれる自然言語の使用方法を構造化して大規模に集積したものを利用して、単語の数よりもはるかに低い次元のベクトル空間で単語の意味を表現できることが挙げられる。

Global Vector: Glove

GloVe [18] も Word2Vec と同様な単語分散表現の手法の一つである。Word2Vec では単語ごとに周囲語や中心語を予測していたが、GloVe ではコーパスから集積された単語のペアの共起確率を線形回帰により予測する。BoW や TF*IDF のような単語の出現頻度に基づいた手法と、Word2Vec のような確率的に単語を予測する手法を折衷した手法であるため、学習が早く大きなコーパスに対応でき、小さなコーパスや小さなベクトルでも優れた性能を示すとされている。

BoW と TF*IDF は scikit-learn 1.2.0²を用いて実装する。また、Word2Vec, GloVe については、Wikipedia から収集しレンマ化した文書データをコーパスとした学習済みモデル³を利用する。それぞれの学習済みモデルを本論文では、Word2Vec-Wiki, GloVe-Wiki と呼ぶこととする。また、Word2Vec に関しては、プログラミング技術に関する Q&A サイトである Stack Overflow から収集しレンマ化した文書データをコーパスとした学習済みモデル⁴ (Word2Vec-SO と呼称する) も利用する。

3.3 分類器

分類器の作成にあたっては、代表的な教師あり分類アルゴリズムであるサポートベクターマシン、ロジスティック回帰、ランダムフォレストの3つの手法を利用する。いずれの手法も scikit-learn 1.2.0 を用いて実装する。

ランダムフォレスト

ランダムフォレストは、単体では分類性能の低い弱

¹<https://spacy.io/>

²<https://scikit-learn.org/stable/>

³<http://vectors.nlpl.eu/repository/>

⁴<https://zenodo.org/record/1199620#.Y7wiF-zP23L>

表 2. データセットの概要

オーナー / リポジトリ名	TD ラベル	一般 (件数)	TD (件数)
influxdata / influxdb	kind/tech-debt	8,514	120
saleor / saleor	technical debt	2,230	108
nextcloud / server	technical debts	7,255	83
microsoft / vscode	debt	79,829	2,044
合計	-	97,828	2,355

分類器を複数組み合わせることで高性能の分類器を作る, アンサンブル学習の一つである [19]. 本論文では, 学習時のハイパーパラメータは, 決定木の木数 k を 200, 決定木の分岐に使用する特徴の個数を全特徴の個数の平方根, ランダムシードを 42 に設定する. その他のハイパーパラメータは scikit-learn のデフォルト値を利用する.

ロジスティック回帰

ロジスティック回帰は, 説明変数を集めた入力データ X から確率的にクラスに分類する統計的パターン認識手法の一つである [19]. 本論文では, 学習時のハイパーパラメータは, 正則化項を L2 ノルム, 反復回数を 2000, ランダムシードを 42 に設定する. その他のハイパーパラメータは scikit-learn のデフォルト値を利用する.

サポートベクターマシン

サポートベクターマシンは, 入力データ X の各要素との距離, つまりマージンが最大となるようにデータ X を分割する m 次元の超平面を求めることで分類するパターン認識のための機械学習アルゴリズムである [19]. 本論文では, 学習時のハイパーパラメータは, 正則化項を L2 ノルム, 反復回数を 2000, ランダムシードを 42 に設定する. その他のハイパーパラメータは scikit-learn のデフォルト値を利用する.

4 分類性能評価実験

本章では, 5 種類の単語分散表現と 3 種類の分類器を用いて構築した合計 15 種類の分類モデルの性能を比較するためにおこなった評価実験について述べる.

4.1 実験の目的

本実験の目的は, 5 種類 (BoW, TF-IDF, Word2Vec-Wiki, Word2Vec-SO, GloVe-Wiki) の単語分散表現と 3 種類の分類器 (ランダムフォレスト, ロジスティック回帰, サポートベクターマシン) を用いて構築した合計 15 種類の分類モデルそれぞれの分類性能を定量的に示すとともに, 単語分散表現が技術的負債に関連する課題票の分類に与える影響を明らかにすることである. 本実験の対象とする課題票に含まれる文書は, 多くが自然言語で記述される. そのため, 先行研究でも用いられた BoW や TF-IDF のような単語の出現頻度に依存した単純な単語分散表現よりも, Word2Vec や GloVe のような単語同士の関係などを表現できる複雑な単語分散表現手法を用いた分類器がより高い性能を示すと予想する.

4.2 データセット

Xavier らの研究 [14] と同様に, GitHub 上のスター数が多い上位 5000 件のプロジェクトのうち, 技術的負債を課題票として習慣的に管理している influxdb, saleor, server, vscode の 4 つのプロジェクトからデータを収集する. (1) GraphQL⁵を用いて 2022 年 4 月 26 日までに close された課題票をすべて収集した後, (2) 報告回数が 1 回のみの開発者による課題票を削除⁶し, (3) 残りの課題票をデータセットとした. 表 2 に本実験で用いるデータセットの概要を示す. 課題票が技術的負債 (TD) に関するのなのか, その他一般に不具合等に関するものなのかについての判断は, プロジェクト毎に用意されているラベルに基づいて分類した.

4.3 実験手順

以下では, 本評価実験の手順について説明する.

⁵<https://graphql.org>

⁶報告自体に不慣れなため報告方法や内容に不備があることが多くノイズとなるため

4.3.1 データの正規化

3.1 節で説明した前処理を行なったデータセットについて、ロジスティック回帰やサポートベクターマシンのような線形分類モデルでは、外れ値の影響を受けることで訓練データに過剰に適合してしまう過学習が起こる可能性がある。そのため、本実験では分類モデルへの入力となるデータを、最大値が 1、最小値が 0 になるように正規化することで、過学習を防ぐ。正規化には次の式 (1) を用いる。

$$X' = \frac{X - x_{\min}}{x_{\max} - x_{\min}} \quad (1)$$

4.3.2 データセットの分割

正規化されたデータセットを入力とし、3 章の手順で作成する分類モデルの訓練と評価を行う。

4.3.3 層化 k 分割交差検証

訓練と評価を行う交差検証について、まず学習データを正例と負例が同じ比率になるよう k 等分する。本実験では、データセットの分割に scikit learn 1.2.0 を利用し、交差検証のための分割数はデフォルト値である $k = 5$ とした。次に、学習データを 5 等分したうちの 1 つを評価データ、残りを訓練データとするモデルの訓練と評価を行う。等分されたデータが、1 度ずつ評価データとなるように、訓練と評価を $k = 5$ 回繰り返す。これらの操作を層化 5 分割交差検証と呼びこれを 20 回おこない、訓練と評価をそれぞれ合計 100 回実行する。このような手順をとるのは、訓練データと評価データに分割した際のデータの偏りにより、不正確な結果となること、つまり過学習を防ぐためである。

4.4 評価指標

分類の予測結果と実際の正誤の関係について、技術的負債に関する課題票を正しく予測した場合を真陽性 (True Positive: TP)、技術的負債に関する課題票をその他の課題票であると誤って予測した場合を偽陰性 (False Negative: FN)、その他の課題票を技術的負債に関する課題票であると誤って予測した場合を偽陽性 (False Positive: FP)、その他の課題票を正しく予測した場合を真陰性 (True Negative: TN) とし、分類性能を Accuracy (2)、Precision (3)、Recall (4)、F1 値 (5) を用いて評価する。それぞれの評価指標は、以下の式で求められる。

$$Accuracy = \frac{TP + TN}{TP + FP + FN + FP} \quad (2)$$

$$Precision = \frac{TP}{TP + FP} \quad (3)$$

$$Recall = \frac{TP}{TP + FN} \quad (4)$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (5)$$

これらの指標が取りうる値の範囲は 0 から 1 である。4.3.3 節で示した手順により 100 回実行して得られた指標を平均し、有効数字 3 桁で示した値を最終的な評価値とする。

4.5 実験結果

表 3 に実験結果を示す。4 つの評価指標において最も高い値はそれぞれ太字で強調している。

表 3 より、作成した分類モデルのうち Accuracy が最も高かったのは、TF*IDF でベクトル化したロジスティック回帰で、0.978 を示した。ただし、Accuracy に関してはいずれの分類モデルもほぼ同等の高い値を示しておりモデル間で大差はない。

Precision が最も高い値を示したのは TF*IDF でベクトル化したランダムフォレストモデルで、0.896 であった。次いで高い値を示したのは BoW でベクトル化したランダムフォレストモデルで、0.880 であった。これら 2 種類の組み合わせについては他の組み合わせによる分類モデルよりも良好な結果を示していると言える。

Recall および F1 値に関しては、TF*IDF を用いたサポートベクターマシンモデルで、Recall で 0.240、F1 値で 0.341 とそれぞれ最も高い値を示した。ただし、0.240 という Recall の値は決して高くなく TP の見逃しが多数発生していることを示唆している。また、0.341 という F1 値も実用性の観点からは十分ではない。

本実験では、単語分散表現が分類モデルの性能に与える影響を調べるために 5 種類のベクトル化手法を用いたが、我々の当初の予想に反して BoW や TF*IDF を用いたシンプルな分類モデルの方が、より複雑なベクトル化手法である Word2Vec や GloVe を用いた分類モデルよりも性能が高かった。5 章では、追加調査を実施し得られた結果をより良く理解するための考察をおこなう。

表 3. 性能評価実験の結果

機械学習アルゴリズム	ベクトル化手法	Accuracy	Precision	Recall	F1
ランダムフォレスト	BoW	0.978	0.880	0.059	0.111
	TF*IDF	0.978	0.896	0.059	0.110
	Word2Vec-SO	0.977	0.768	0.018	0.035
	Word2Vec-Wiki	0.977	0.663	0.020	0.038
	GloVe-Wiki	0.977	0.639	0.002	0.038
ロジスティック回帰	BoW	0.977	0.662	0.014	0.028
	TF*IDF	0.979	0.736	0.156	0.257
	Word2Vec-SO	0.977	0.518	0.113	0.185
	Word2Vec-Wiki	0.977	0.360	0.005	0.010
	GloVe-Wiki	0.977	0.495	0.071	0.124
サポートベクターマシン	BoW	0.977	0.520	0.092	0.156
	TF*IDF	0.978	0.589	0.240	0.341
	Word2Vec-SO	0.977	0.652	0.019	0.037
	Word2Vec-Wiki	0.977	0.563	0.006	0.013
	GloVe-Wiki	0.977	0.624	0.020	0.039

表 4. 重要度の高い上位 5 件の単語

ベクトル化	重要度の高い単語				
BoW	oov	unique	inline	link	debt
TF*IDF	oov	link	unique	code	debt

oov : spaCy の辞書外の単語であることを示す

unique : 固有名詞であることを示す

inline : プログラムコードであることを示す

link : リンクであることを示す

5 考察

本章では、分類モデルの性能評価実験で得られた結果について考察する。

5.1 分類に寄与する単語の調査

表 3 より、Precision が最も高い値を示したのは TF*IDF でベクトル化したランダムフォレストモデルで、0.896 であった。次いで高い値を示したのは BoW でベクトル化したランダムフォレストモデルで、0.880 であった。このような結果が得られた理由を知るために、ランダムフォレストモデルで得られる単語の重要度 $M(T)$ について、上位の単語 5 つを調査した。調査結果を表 4 に示す。重要度の高さは、ランダムフォレストに基づく分類モデルを評価する際に合計 100 回学習した過程で算出された結果から集計した表 4 より、BoW と TF*IDF で共に、前処理のレンマ化ができない辞書外の単語 (oov) や固有名詞 (unique)、プログラムコード (inline)、リンク (link)、技術的負債を示す単語「debt」

などの重要度が高いことがわかった。これらの単語のうち、特に辞書外の単語や固有名詞については、複雑なベクトル化手法を使用した場合、置き換えた後の単語の特徴をうまく捉えたベクトル化ができなかった可能性が考えられる。

5.2 不均衡データへの考慮

表 3 から結果を俯瞰すると、Accuracy と Precision が高く、Recall が低い分類モデルを作成できた。これは、分類モデルが技術的負債であると予測した課題票については正しく分類できたが、実際には予測の取りこぼしが多かったことを示している。Precision が高く Recall が低い理由としては、正例と負例のデータの偏りが考えられる。表 2 のとおり、技術的負債以外を扱う課題票が技術的負債を扱う課題票の約 40 倍以上あり、正例が非常に少ない。これにより、技術的負債を扱っている課題票の特徴を掴めず、技術的負債を扱っている課題票の多くをその他の課題票であると予測している可能性がある。

そこで、それぞれの分類モデルを学習させる際に、ハイパーパラメータの `class_weight` を `balanced` に設定することで分類するクラスの偏りに応じて重み付けを行い、再度実験を実施した。表 5 に実験結果を示す。

表 5 より、作成した分類モデルのうち、Accuracy が最も高かったのは、BoW と TF*IDF でベクトル化したランダムフォレストモデルで、同率の最高値の 0.978 を示した。Precision で最も高い値を示したのは、TF*IDF でベクトル化したランダムフォレストモデルで、値は 0.857 であった。さらに、Word2Vec-SO でベクトル化し

表 5. 正例データに重み付けした場合の結果

機械学習アルゴリズム	ベクトル化手法	Accuracy	Precision	Recall	F1
ランダムフォレスト	BoW	0.978	0.855	0.064	0.118
	TF*IDF	0.978	0.857	0.050	0.095
	Word2Vec-SO	0.977	0.749	0.029	0.056
	Word2Vec-Wiki	0.977	0.798	0.022	0.042
	GloVe-Wiki	0.977	0.786	0.020	0.039
ロジスティック回帰	BoW	0.827	0.094	0.738	0.166
	TF*IDF	0.916	0.017	0.672	0.272
	Word2Vec-SO	0.834	0.101	0.776	0.179
	Word2Vec-Wiki	0.821	0.094	0.768	0.167
	GloVe-Wiki	0.831	0.099	0.766	0.175
サポートベクターマシン	BoW	0.872	0.112	0.647	0.190
	TF*IDF	0.928	0.172	0.546	0.261
	Word2Vec-SO	0.906	0.155	0.669	0.252
	Word2Vec-Wiki	0.834	0.100	0.770	0.178
	GloVe-Wiki	0.837	0.101	0.759	0.179

表 6. 重要度の高い上位 5 件の単語 (重み付け後)

ベクトル化	重要度の高い単語				
BoW	image	version	unique	vscode	os
TF*IDF	image	version	link	unique	os

image : 画像であることを示す

たロジスティック回帰モデルは, Recall で最高値 0.776 を示した. 最後に, TF*IDF を用いたロジスティック回帰モデルは, 最も高い F1 値として 0.272 を示した. クラスに重み付けする前と結果を比較すると, 分類モデルにより増減幅の差はあるものの, ほとんどの分類モデルで Precision が低くなり, また多くの分類モデルで Recall が高くなった. また, F1 値の最高値は, クラスに重み付けする前の方が高かった.

クラスに重み付けすることで Recall が Precision をよりも大きく上回るようになったロジスティック回帰とサポートベクターマシンに基づく分類モデルは, 正例を比較的取りこぼさずに予測できるようになったが, 偽陽性を多く含むようになったことを示している. 一方, ランダムフォレストを用いた分類モデルでは, クラスに重み付けしても結果がほとんど変わらなかった.

クラスに重み付けすることにより重要な単語が変わる可能性があるため, 単語の重要度を調査した結果を表 6 に示す. 表 6 に示す通り, クラスに重み付けしたことにより重要な単語が変化した. 表 4 と表 6 を比較すると, クラスに重み付けすることで辞書外の単語やプログラムコードなどの重要度よりも, 画像や「version」などの既

存の単語の重要度が高くなるという変化があった.

以上の結果から, BoW や TF*IDF 以外の分散表現を用いた分類モデルの性能を向上させるためには, 辞書外の単語について前処理の改善を行う必要があることがわかった. また, 学習データにおける正例・負例を均衡させるためにクラスに重みを付けた分類モデルは, Recall を改善できることがわかった. そのため, はじめに作成した Precision が高い値を示す分類モデルは, 量は期待できないが, 技術的負債を取り扱う課題票を正しく収集する際に利用できる. また, 追加調査で作成した Recall が高い値を示す分類モデルは, 正しさは保証できないが, 技術的負債を取り扱う課題票の取りこぼしがないように収集するときに利用できると言える.

5.3 妥当性

5.3.1 内的妥当性

本論文では, 「debt」等ラベルが付与された課題票を技術的負債を管理するための課題票として定義した. 技術的負債を管理するためのその他のラベルが利用されている可能性があり, 実験結果に影響している可能性がある. また, 本論文ではレンマ化において spaCy を利用しているが, 辞書外に特徴量となる単語 (OOV など) が多数あることが分かったため, 辞書外の単語の置き換えについて再考する必要がある. また, より新しい単語分散表現手法の RNN や CNN, そして最新の単語分散表現手法である Transformer やそれをベースとした BERT や GPT などについても比較することで, より広く分類性能への影響を分析できるため, 今後の課題としたい.

表 7. 本実験で利用した Issue とそのプロジェクトの情報

オーナー / リポジトリ名	主な開発言語	アプリケーションの概要
influxdata / influxdb	Go	時系列データベース
saleor / saleor	Python	EC サイト構築のプラットフォーム
nextcloud / server	PHP	オンラインストレージ
microsoft / vscode	JavaScript	ソースコードエディタ

5.3.2 外的妥当性

技術的負債を課題票として扱うプロジェクトが未だ少ないことから、本実験が一般化できるかについては不明である。しかし、表 7 に示す通り、本論文のデータセットとした 4 つのプロジェクトはそれぞれ異なる言語と目的で開発されている。そのため、さまざまな分野のプロジェクトにおいて本論文で作成した分類モデルをある程度適用させることができると考える。今後の課題として、本実験のデータセットには含まれなかった分野のプロジェクトについてデータを収集し検証する必要があると考える。

6 関連研究

6.1 ソースコードコメントに表明される技術的負債

Potdar ら [3] は、開発者が技術的負債を意識的に管理する方法として、ソースコード中にコメントで表明する技術的負債 (SATD: Self-Admitted Technical Debt) の存在を明らかにした。ソースコード中に記述されるコメントから技術的負債を特定できることから、近年 SATD の検出に関する研究が盛んに行われている。Potdar ら [3] は、SATD-Comment に含まれる、技術的負債を示す 62 種類のコメントパターンを特定した。また、2.4% から 31.0% のソースコードファイルに SATD が含まれること、主に経験豊富な開発者が活動全体で技術的負債を導入すること、製品の公開までの期間やコードの複雑さと技術的負債の間に強い相関は存在しないこと、そして製品の公開後も 26.3% から 63.5% しか技術的負債が除去されないことについても明らかにした。その後、ソースコードコメントに基づいて技術的負債を検出する手法 [4–7] が研究された。さらに、単語分散表現などの自然言語処理を用いてさらに検出精度を高める研究 [8–13] なども進められている。

6.2 課題票として管理される技術的負債

技術的負債は、課題票として管理される事例があることが Xarvier らの研究 [14] によって明らかになった。木村らの先行研究 [15] では、課題票として管理される技術的負債の特徴分析に基づいて技術的負債に関する課題票を分類する手法を構築している。本研究は、課題票として管理される技術的負債の早期検出を支援するために、課題票に含まれるテキストのみを入力とした分類モデルを作成し、その性能について比較評価した。

7 おわりに

本論文では、課題管理システムに登録された課題票からタイトルと本文のテキストデータを収集し、5 種類の単語分散表現と 3 種類の機械学習アルゴリズムを用いて、技術的負債として管理すべき課題票を分類するモデルを作成した。性能比較実験の結果、BoW や TF*IDF といったシンプルな単語分散表現を用いた分類モデルが高い Precision を示すことがわかった。また、学習データに含まれる正例・負例を均衡させるように学習させた場合、ロジスティック回帰モデルとサポートベクタマシンモデルを用いた分類モデルで Recall が大きく改善することを確認した。

BoW や TF*IDF 以外の単語分散表現については、本実験の設定では特徴を掴めなかった単語への前処理の工夫を行うほか、文脈を把握し未知語へのより柔軟な対応が可能である最新の単語分散表現を加えることで、更なる分類モデルの性能向上を目指す。さらに今後は、設計に関する技術的負債やテストに関する技術的負債など、技術的負債の種類ごとに分類するモデルを構築する予定である。

謝辞

本研究の一部は、文部科学省科学研究補助金（基盤（C）：22K11974）による助成を受けた。

参考文献

- [1] Ward Cunningham. The wycash portfolio management system. *SIGPLAN OOPS Mess.*, Vol. 4, No. 2, pp. 29–30, 1992.
- [2] Zengyang Li, Paris Avgeriou, and Peng Liang. A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, Vol. 101, No. C, pp. 193–220, 2015.
- [3] Aniket Potdar and Emad Shihab. An exploratory study on self-admitted technical debt. In *Proceedings of the International Conference on Software Maintenance and Evolution*, ICSME, pp. 91–100, October 2014.
- [4] Xiaoxue Ren, Zhenchang Xing, Xin Xia, David Lo, Xinyu Wang, and John Grundy. Neural network-based detection of self-admitted technical debt: From performance to explainability. *ACM Transactions on Software Engineering and Methodology*, Vol. 28, No. 3, 2019.
- [5] Zhe Yu, Fahmid Morshed Fahid, Huy Tu, and Tim Menzies. Identifying self-admitted technical debts with jitterbug: A two-step approach. *IEEE Transactions on Software Engineering*, Vol. 48, No. 5, pp. 1676–1691, 2022.
- [6] Irene Sala, Antonela Tommasel, and Francesca Arcelli Fontana. Debthunter: A machine learning-based approach for detecting self-admitted technical debt. In *Proceedings of the Evaluation and Assessment in Software Engineering*, EASE, pp. 278–283, June 2021.
- [7] Murali Sridharan, Mika Mäntylä, Maëlick Claes, and Leevi Rantala. Soccminer: A source code-comments and comment-context miner. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR, pp. 242–246, May 2022.
- [8] Jernej Flisar and Vili Podgorelec. Enhanced feature selection using word embeddings for self-admitted technical debt identification. In *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA, pp. 230–233, August 2018.
- [9] Supatsara Wattanakriengkrai, Rungroj Maipradit, Hideki Hata, Morakot Choetkiertikul, Thanwadee Sunetnanta, and Kenichi Matsumoto. Identifying design and requirement self-admitted technical debt using n-gram idf. In *Proceedings of the 9th International Workshop on Empirical Software Engineering in Practice*, IWESEP, pp. 7–12, December 2018.
- [10] Jernej Flisar and Vili Podgorelec. Identification of self-admitted technical debt using enhanced feature selection based on word embedding. *IEEE Access*, Vol. 7, pp. 106475–106494, 2019.
- [11] Fiorella Zampetti, Alexander Serebrenik, and Massimiliano Di Penta. Automatically learning patterns for self-admitted technical debt removal. In *Proceedings of the 27th International Conference on Software Analysis, Evolution and Reengineering*, SANER, pp. 355–366, February 2020.
- [12] Leevi Rantala. Towards better technical debt detection with nlp and machine learning methods. In *Proceedings of the 2nd International Conference on Software Engineering: Companion Proceedings*, ICSE, pp. 242–245, October 2020.
- [13] Amleto Di Salle, Alessandra Rota, Phuong T. Nguyen, Davide Di Ruscio, Francesca Arcelli Fontana, and Irene Sala. Pilot: Synergy between text processing and neural networks to detect self-admitted technical debt. In *Proceedings of the International Conference on Technical Debt*, TechDebt, pp. 41–45, May 2022.
- [14] Laerte Xavier, Fabio Ferreira, Rodrigo Brito, and Marco Tulio Valente. Beyond the Code: Mining Self-Admitted Technical Debt in Issue Tracker Systems. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR, pp. 137–146, June 2020.
- [15] 木村祐太, 大平雅雄. 技術的負債に関連する課題票分類手法の構築. 情報処理学会論文誌, Vol. 64, No. 1, pp. 2–12, 2023.
- [16] Emerson Murphy-Hill, Chris Parnin, and Andrew P. Black. How we refactor, and how we know it. *IEEE Transactions on Software Engineering*, Vol. 38, No. 1, pp. 5–18, 2012.
- [17] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.
- [18] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.
- [19] Christopher M. Bishop, 元田浩, 栗田多喜夫, 樋口知之, 松本裕治, 村田昇. パターン認識と機械学習: ベイズ理論による統計的予測. 丸善出版, 2012.
- [20] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information, 2016.
- [21] Pei Wang, Jinqiu Yang, Lin Tan, Robert Kroeger, and J. David Morgenthaler. Generating precise dependencies for large software. In *Proceedings of the 4th International Workshop on Managing Technical Debt*, MTD, pp. 47–50, May 2013.

Testing-Based Formal Verification on Software Programs: A Review

Ai Liu
Hiroshima University
liuai@hiroshima-u.ac.jp

Shaoyin Liu
Hiroshima University
sliu@hiroshima-u.ac.jp

Abstract

Testing-based formal verification has been proposed to automatically verify whether a software program satisfies the requirements written as a formal specification by exploring program paths. There are two tools that can be used to verify path correctness: Hoare logic or symbolic execution. In this document, we will discuss their merits and demerits, as well as the future research directions.

1. Introduction

Software testing, one of the most frequently used quality assurance methods, is focused on sampling the executions, according to some coverage criteria, and comparing the actual behavior with the expected behavior. Nevertheless, a typical weakness of testing is that it cannot generally tell the absence of bugs in programs. To find more bugs, more test data need to be generated and more program executions, which will inevitably increase the cost. An idea for improvement is to ensure the correctness of the executed path for a given test case with formal methods and then all of the test cases, resulting in the same executed path, will satisfy the expected behavior.

Formal methods, based on some mathematical theories, are a collection of notations and techniques, including specification and verification [1]. Formal specification techniques introduce a precise and unambiguous description of the software behavior. Given such a specification, it is possible to use formal verification techniques to demonstrate that a software design is correct with respect to the specification. However, most of software verification methods do not guarantee the correctness of actual code, but rather allow one to verify some abstract model of it. Hence, due to possible differences between the actual software system and its abstract model, the correctness of the proof may not carry over. To ensure the correctness of the code, deductive verification, including symbolic execution and

Hoare logic, has been demonstrated mainly for simple programs which contain no loop sentences and no side effects.

Hoare logic is established based on predicate logic and provides a set of axioms to define the semantics of programming language [2]. For each program construct, such as sequence, selection, or iteration, an axiom for defining its semantics is defined. These axioms can be used to reason about the correctness of programs written in a programming language. However, the axiom for iteration is hard to be directly used since the derivation of loop invariants may not be easy.

Symbolic execution systematically explores the possible paths of a program's execution by executing the program with symbolic inputs instead of concrete values and then allows the tester to explore different execution paths and detect potential bugs and vulnerabilities [3]. Symbolic execution is often used in combination with other techniques such as model checking and theorem proving to verify the correctness of software systems. There are severe challenges of symbolic execution such as path explosion and constraint solving. Nevertheless, it is an active area of research, and new techniques and tools are constantly being developed to improve its efficiency and effectiveness.

A natural idea is to combine testing and formal methods (specification and verification) to make the best use of their merits while reducing the cost. As Whittaker noted "Without a specification testers are likely to find only the most obvious bugs" in 2000, specification-based testing (SBT) has been advocated for a long time [4, 5]. However, due to the weakness of testing to tell the absence of bugs in programs, specification-based testing also need high test cost in order to improve software reliability. For the purpose to improve specification-based testing with formal verification techniques, a method known as testing-based formal verification (TBFV), characterized by the integration of specification-based testing with Hoare logic to ensure the correctness of program paths which contain no loop sentences and no side effects, has been ex-

plored and its initial idea is proposed in [6]. The idea is further developed to combine with program inspection later [7]. A collection of prototype tools to illustrate how to support the TBFV method is presented in [8]. The original framework of TBFV is extended in [9] to deal with operations, which may involve with complex data structure and side effects, in software packages, such as Vector, ArrayList and LinkedList packages in Java language. Before that extension, testing-based formal verification with symbolic execution is alternatively proposed as an improved method by replacing the application of Hoare logic with symbolic execution in [10]. Branch sequence coverage (BSC) for TBFV-SE is considered in [11] and a fault localization approach is proposed to further pinpoint the problematic positions in the incorrect program paths given by TBFV-SE in [12]. Moreover, the idea is also applied to the verification of neural networks, resulting in TBFV-INN, a new framework for verifying and improving neural networks [13].

As previously mentioned, since TBFV is firstly proposed, it has been constantly improved on two roads: one is based on Hoare logic and the other is based on symbolic execution. In this document, we will enjoy different scenery and discuss which treasure we intend to dig on the two roads in the future.

The rest is organized as follows. Section 2 introduces the framework of testing-based formal verification. Section 3 describes the framework of test-based formal verification with symbolic execution. Section 4 discusses the merits and demerits of TBFV and TBFV-SE.

2. The Framwork of TBFV

In formal methods, the first step to implement an operation is usually to write a formal specification to describe its requirements. The assumption for TBFV is that the operation specification S can be represented as a functional scenario form $\bigvee_{i=1}^n (T_i \wedge D_i)$ where T_i is called a test condition and D_i is called a defining condition. A test condition is a constraint only on the input variables, while a defining condition contains at least one output variable. The functional scenario $T_i \wedge D_i$ describes a single specific functional behavior: when T_i is true, the output of the operation is defined using D_i . Given an operation specification $S = \bigvee_{i=1}^n (T_i \wedge D_i)$ with a program P , the TBFV method is proposed to tackle the problem whether both input and output variables satisfy D_i after P is executed if input variables satisfy the corresponding T_i .

The framework of TBFV is illustrated in Figure 1 and explained as follows.

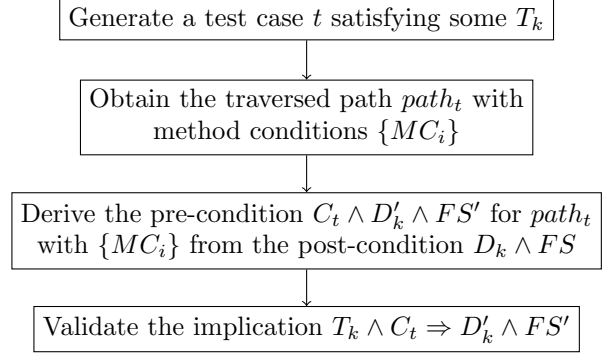


Figure 1. The framework of TBFV

- The first step is to randomly generate a test case t satisfying some test condition T_k . Only one test condition will be satisfied due to the well-formedness of the specification. Note that when the program involves with a method invocation of a complex component (such as Vector, ArrayList and LinkedList in JAVA), the test case t needs to contain the initial component state.
- The second step is to execute program P with test case t and obtain a traversed program path $path_t$, which is a sequence of guard conditions, assignment statements and method invocations. Each method has already been equipped with a Hoare tripe which may contain multiple activated guard conditions. The activated guard condition MC_i of the i -th invoked method should also be recorded.
- As the test case t must contain the initial component state, correspondingly, the post-condition should be not only the defining condition D_k , but the conjunction of the defining condition and the final component state $D_k \wedge FS$ (each component has a final state representation FS). Assuming such post-condition, we can backwards derive the pre-condition $C_t \wedge D'_k \wedge FS'$ of $path_t$ with method conditions $\{MC_i\}$ where C_t is the conjunction of guard conditions in $path_t$ after replacing all intermediate variables with expressions of input variables and $D'_k \wedge FS'$ is obtained by replacing all output variables with expressions of input variables in the derivation process.
- The final step is to find whether $T_k \wedge C_t$ implies $D'_k \wedge FS'$ with the aid of theorem provers. If it is a tautology, $path_t$ with method conditions $\{MC_i\}$ is correct with respect to the specification S ; otherwise, there must be errors in $path_t$.

If the path correctness is ensured, all test cases validating $T_k \wedge C_t$ will satisfy the specification. Then, we

can choose a test case t validating $T_k \wedge \neg C_t$ to continue the above steps.

Example 1. Consider a jump function f whose domain is the set of integers $\mathbb{Z}$ and output is

$$y = f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Then, the corresponding FSF S is

$$(x \in \mathbb{Z} \wedge x \geq 0 \wedge y = 1) \vee (x \in \mathbb{Z} \wedge x < 0 \wedge y = 0)$$

Consider the following program:

```
int jump(int x){
    int y = 0;
    if (x >= 0)
        y = (x + 2)/(x + 1);
    return y;
}
```

where variable x is the input and variable y is the output. If the given input is $x = 1$, the corresponding testing condition is $x \in \mathbb{Z} \wedge x \geq 0$ and defining condition is $y = 1$. Then we can derive the pre-condition for the corresponding path as follows

$$\begin{aligned} &\{x \geq 0 \wedge 1 = (x + 2)/(x + 1)\} \\ &\quad y = 0 \\ &\{x \geq 0 \wedge 1 = (x + 2)/(x + 1)\} \\ &\quad x \geq 0 \\ &\{1 = (x + 2)/(x + 1)\} \\ &\quad y = (x + 2)/(x + 1) \\ &\{y = 1\} \end{aligned}$$

The next step is to consider the implication

$$x \in \mathbb{Z} \wedge x \geq 0 \wedge x \geq 0 \Rightarrow 1 = (x + 2)/(x + 1)$$

which is true for $x > 0$ due to the division operation of integers but false when $x = 0$. If we only use testing, we may select many test cases in which $x \neq 0$ and produce the test results as expected. In this case, the contradiction can not be revealed. However, using TBFV, we can get the same path and implication once a test case $x \geq 0$ is chosen, and find the contradiction.

3. The Framework of TBFV-SE

Testing-based formal verification with symbolic execution (TBFV-SE) has already been a method to automatically verify the correctness of all the representative program paths given the formal specifications and

derived a fault localization approach. In TBFV-SE, the correctness of each path is also converted into a theorem obtained during a dynamic symbolic execution, rather than the derivation by Hoare logic. The framework of TBFV-SE is illustrated in Figure 2 and explained as follows.

- Naturally, the first step is also to generate a test case t , but there is no need to fix a test condition. The initial component state should be included in the test case if some component method is invoked.
- After executing program P with test case t , a traversed program path can be obtained. Replacing test case t with symbol values and applying symbolic execution for that path, all the output variables, path conditions and component states are finally related to the symbol values. We use C_t and $state_t$ to denote path-condition, the integration of all path conditions and a symbolic state including all output variables and all component states, respectively.
- Although the test case t only validates a testing condition, there may exist other test cases, validating different testing conditions, which result in the same traversed path. If the implication $C_t \wedge T_i \Rightarrow state_t \wedge D_i$ is tautologous, it means that the test cases validating testing condition T_i and resulting in this traversed path satisfy the specification. There, the theorem we intend to prove is $\forall i C_t \wedge T_i \Rightarrow state_t \wedge D_i$, which ensures the path correctness.
- Branch sequence coverage (BSC) is proposed to automatically partition the domain by dynamically gathering all the necessary symbolic paths since each symbolic path is selected to represent a sub-domain. This partition is depended on a dynamic light analysis of code structure without judging if the branch conditions are serialized or nested. Unlike conventional partition testing selecting classical values from classes to test, BSC focuses on automatically selecting a real symbolic path for representing each sub-domain according to the real structure of codes. There is an algorithm to automatically find test cases satisfying BSC in [11].
- A fault localization algorithm is proposed in [12] to provide a rigorous way to automatically analyze the correctness of statements in a program based on the theorems of the traversed paths and help locate the faults in a small set of statements by examining very small percentage of the code.

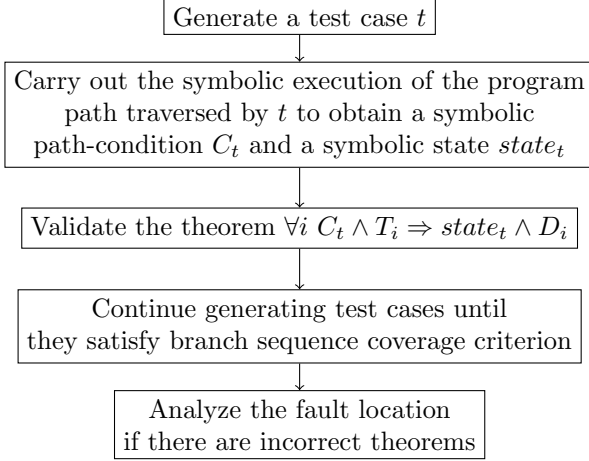


Figure 2. The framework of TBFV-SE

Example 2. Reconsidering Example 1 with TBFV-SE, the derivation of path-condition and the record of states for the traversed path by the input x with numeric value 1 and symbolic value X is displayed in Table 1. The last line of second column of Table 1 shows both the symbolic state and symbolic path-condition. Then, the induced theorem is

$$\begin{aligned}
 & (x \in \mathbb{Z} \wedge x \geq 0 \wedge x \geq 0 \Rightarrow y = 1 \wedge y = \frac{x+2}{x+1}) \wedge \\
 & (x \in \mathbb{Z} \wedge x < 0 \wedge x \geq 0 \Rightarrow y = 0 \wedge y = \frac{x+2}{x+1})
 \end{aligned}$$

where the first sub-formula is false while the second sub-formula is true. We just use this simple example to show how symbolic execution help derive theorems in TBFV-SE.

Table 1. Single path of symbolic execution

program code	symbolic execution
int jump(int x)	input: $x = 1$ symbolic value of x : X
int y = 0	state: $\{y = 0\}$
if (x >= 0)	path-condition: $X \geq 0$
y = (x + 2)/(x + 1)	state: $\{y = \frac{x+2}{x+1}\}$
return y	$\{y = \frac{x+2}{x+1}, X \geq 0\}$

4. Merits and Demerits

As for our goal to verify whether a program satisfies its corresponding specification, TBFV and TBFV-SE can both automatically generate test cases, obtain traversed paths and induce theorems. Next, external theorem provers will be used to validate the induced

theorems. Further, TBFV-SE has been enriched with two algorithms to find representative program paths and analyze the correctness of statements. We mainly compare the common procedures of TBFV and TBFV-SE.

Given a test case t , TBFV and TBFV-SE will obtain the same program path $path_t$. The difference is the way to derive the theorems, as depicted in Example 1 and 2. In TBFV, we need to preselect a functional scenario $T_i \wedge D_i$ and the derived theorem can only represent whether the traversed path satisfies that functional scenario. In TBFV-SE, the derived theorem can be used to judge whether the traversed path satisfies all functional scenarios. Given the test cases inducing a traversed path, if some of them validate a testing condition T_i and the rest validate another testing condition T_j , we just need one derivation process of the theorem in TBFV-SE while we should need two derivation processes of the theorem in TBFV. In that case, TBFV-SE should be more efficient than TBFV. Nevertheless, TBFV will have an advantage than TBFV-SE in cases including many intermediate variables and irrelevant variables, as depicted in Example 3.

Example 3. Consider the following program which implements a piecewise function,

```

int piece(float x){
    float y1 = 1 - x;
    float y2 = 2x;
    float y3 = 1/x;
    if (x < -4)
        y = y1;
    else if (-4 ≤ x ≤ 0)
        y = y2;
    else
        y = y3;
    return y;
}
  
```

For any input value, the traversed path must include the first three assignment statements. Assume the input is 0 and the corresponding functional scenario is $T \wedge D$ where D only includes the variable y , the derivation by Hoare logic is below.

$$\begin{aligned}
 & \{-4 \leq x \leq 0 \wedge D(2x/y)\} \\
 & \quad y1 = 1 - x \\
 & \{-4 \leq x \leq 0 \wedge D(2x/y)\} \\
 & \quad y2 = 2x \\
 & \{-4 \leq x \leq 0 \wedge D(y2/y)\}
 \end{aligned}$$

$$\begin{aligned}
& y3 = 1/x \\
& \{-4 \leq x \leq 0 \wedge D(y2/y)\} \\
& -4 \leq x \leq 0 \\
& \{D(y2/y)\} \\
& y = y2 \\
& \{D\}
\end{aligned}$$

Then, the theorem to be proved in TBFV is

$$T \wedge -4 \leq x \leq 0 \Rightarrow D(2x/y)$$

The corresponding symbolic execution is shown in Table 2 where we omit the skipped statements. Then, the theorem to be proved in TBFV-SE is

$$\begin{aligned}
& T \wedge x \geq 4 \wedge -4 \leq x \leq 0 \\
& \Rightarrow D \wedge y1 = 1 - x \wedge y2 = 2x \wedge y3 = \frac{1}{x} \wedge y = 2x
\end{aligned}$$

Table 2. Symbolic execution for $x = 0$

program code	symbolic execution
int piece(float x)	input: $x = 0$ symbolic value of $x : x$
float y1 = 1 - x	state: $\{y1 = 1 - x\}$
float y2 = 2x	state: $\{y1 = 1 - x, y2 = 2x\}$
float y3 = 1/x	state: $\{y1 = 1 - x, y2 = 2x, y3 = \frac{1}{x}\}$
if x < -4	path-condition $x \geq 4$
else if -4 ≤ x ≤ 0	path-condition $x \geq 4 \wedge -4 \leq x \leq 0$
y = y2	state: $\{y1 = 1 - x, y2 = 2x, y3 = \frac{1}{x}, y = 2x\}$
return y	$\{y1 = 1 - x, y2 = 2x, y3 = \frac{1}{x}, y = 2x, x \geq 4 \wedge -4 \leq x \leq 0\}$

Obviously, the theorem obtained by TBFV-SE is more complex than the theorem obtained by TBFV. Actually, we must simplify the theorem obtained by TBFV-SE as

$$T \wedge x \geq 4 \wedge -4 \leq x \leq 0 \Rightarrow D \wedge y = 2x$$

which can be proved by theorem provers, since the formulas $y1 = 1 - x$, $y2 = 2x$ and $y3 = \frac{1}{x}$ are undecidable equations. Moreover, $D \wedge y = 2x$ can be further simplified as $D(2x/y)$. Therefore, the theorem obtained by TBFV is more concise and there is no need to record intermediate variables (e.g. $y2$) and irrelevant variables (e.g. $y1$ and $y3$) in TBFV.

5. Conclusion and Future Work

TBFV and TBFV-SE are both currently able to reduce test cost by ensuring path correctness, while each of them has its advantages and disadvantages. TBFV-SE includes a fault localization algorithm based on the traversed paths obtained by test cases satisfying branch sequence coverage criterion. In the future, we intend to locate faults in a single path which is proved to have errors by TBFV.

References

- [1] Peled, D. A. “Software Reliability Methods”, Springer, 2001.
- [2] Hoare, C. A. R. and Wirth, N. “An Axiomatic Definition of the Programming Language PASCAL”, *Acta Informatica* 2, pp. 335-355, 1973.
- [3] Cadar, C. and Sen, K. “Symbolic Execution for Software Testing: Three Decades Later”, ACM, 2013.
- [4] Whittaker, J. A. “What is software testing? Why is it so hard? Practice Tutorial”, *IEEE Softw.* 17(1), pp. 70-79, 2000.
- [5] Chen, Y. and Liu, S. “An Approach to Detecting Domain Errors Using Formal Specification-Based Testing”, *Proceedings of APSEC 2004*: 276-283.
- [6] Liu, S. “Utilizing Hoare Logic to Strengthen Testing for Error Detection in Program”, *Proceedings of Turing-100 2012*, pp. 229-238.
- [7] Liu, S. and Nakajima, S. “Combining Specification-Based Testing, Correctness Proof, and Inspection for Program Verification in Practice”, *Proceedings of SOFL+MSVL 2013*, pp. 3-16.
- [8] Liu, S. “A Tool Supported Testing Method for Reducing Cost and Improving Quality”, *Proceedings of QRS 2016*, pp. 448-455.
- [9] Liu, A. and Liu, S. “Enhancing the Capability of Testing-Based Formal Verification by Handling Operations in Software Packages”, *IEEE Trans. Software Eng.* 49(1), pp. 304-324, 2023.
- [10] Wang, R. and Liu, S. “TBFV-SE: Testing-Based Formal Verification with Symbolic Execution”, *Proceedings of QRS 2018*, pp. 59-66.

- [11] Wang, R. and Liu, S. “Branch Sequence Coverage Criterion for Testing-Based Formal Verification with Symbolic Execution”, *Proceedings of QRS-C 2019*, pp. 205-212.
- [12] Wang, R. et al. “A Fault Localization Approach Derived from Testing-Based Formal Verification”, *Proceedings of ICECCS 2022*, pp. 165-170.
- [13] Liu, H. et al. “Verifying and Improving Neural Networks Using Testing-Based Formal Verification”, *Proceedings of SOFL+MSVL 2022*, pp. 126-141.

Testing-based formal verification for ReLU-based neural networks

Haiyi Liu

Hiroshima University

d200101@hiroshima-u.ac.jp

Shaoying Liu

Hiroshima University

sliu@hiroshima-u.ac.jp

Ai Liu

Hiroshima University

liuai@hiroshima-u.ac.jp

Yujun Dai

Hiroshima University

d201609@hiroshima-u.ac.jp

Abstract

Since neural networks are widely used, how to ensure the reliability of neural networks has become a hot research topic. The first difficulty is that it is difficult to give the pre-condition and post-condition for neural networks, and the second difficulty also exists in traditional software, i.e., the problem of exploding execution paths. Fortunately, the output range of a neural network is easier to give and when an input is given, we can obtain the activation order of neurons in the neural network. Based on the above facts, we propose DeepTBFV, a method for pre-trained neural networks, which uses a testing-based formal verification algorithm to derive the pre-condition of the neural network on a specified path by post-condition. The purpose is to verify and explain the behavior of the neural network.

1. Introduction

Although neural network has been widely used in many fields such as NLP (natural language processing), image processing and even MMML (Multi-modal Machine Learning), its poor reliability has also been criticized by users for a long time. Specifically, there are two aspects. One is that the neural network is difficult to be verified. The reason is that the architecture of the neural network is based on experience, and the parameter is constructed by back-propagation of the training data. It is difficult to give a formal specification like traditional software, and the verification of the neural network is an NP-hard problem[1], which makes it difficult to achieve complete verification of the large models. The second is that the neural network

is difficult to be explained, that is, it is difficult for us to figure out what features the result of neural network reasoning is based on. For instance, although a neural network correctly recognizes the cat in the picture, we cannot determine whether the neural network correctly recognizes the cat through its features or the watermark in the picture.

Generally, the post-condition of a neural network is easy to give. If the pre-condition can be derived from the neural network by using the post-condition, then users can use the pre-condition to explain and verify the reliability of the neural network. However, since the verification of neural networks is an NP-hard problem, it is very difficult to derive the preconditions from the post-conditions simply by using the verification algorithm.

Existing methods mainly focus on how to accelerate the verification speed of neural network under the premise of given formal specification [2, 3, 4, 5]. These studies provide a basis for the verification of neural networks, but do not discuss how to give a pre-condition to explain neural networks. Test-based neural network interpretation algorithm can explain the behavior of the local neural network by testing and constructing the local interpretation model of the neural network [6, 7, 8]. But they cannot be used to formally verify the reliability of neural networks. In other words, it can not be guaranteed that there is no adversarial examples in the neural network within a certain range.

In this paper, We propose *DeepTBFV*, a method to give the pre-condition of neural network by backward derivation of the post-condition of neural network. *DeepTBFV* is inspired by Testing-based formal verification (TBFV) [9, 10, 11] in traditional software. First, we select a test case that can be correctly recognized by the neural network and execute it. During the execution of the test case, the activation of each neuron is recorded. Here, we compare the activation sequence of neurons to the execution path of the test case in traditional software. Therefore, recording the activation sequence of neurons is the execution path of the test case generated by the neural network. Then, according to the execution results of the neural network, the post-condition of the test case is given, which is used to derive the pre-condition.

Finally, we combine the post-condition and the execution path on the neural network to form a system of linear inequalities with multiple variables. The pre-condition of the neural network is obtained by solving the system of linear inequalities with multiple variables.

In brief, the main contributions of this paper are as follows:

- To my best knowledge, we develop the first testing-based formal verification method for DNNs.
- We design a verification based neural network re-training algorithm.

2. Preliminary

2.1. Testing-based formal verification

The testing-based formal verification (TBFV) is proposed to ensure the correctness of all traversed program paths in traditional software. The first step of TBFV is to generate a test case T based on the test condition in the formal specification. The second step is to obtain a traversed program *path* by executing the test case T execution program P , where the *path* contains a series of conditions. The third step is to verify the reliability of the *path* under the formal specification by using symbolic execution or Hoare logic [12].

2.2. Floyd-Hoare Logic

Floyd Hoare logic, also known as Hoare logic, represents predicate logic and a set of axioms in the form of Hoare triples, and then defines the semantics of the programming language. The specific form of Hoare triples is as follows:

$$\{pre\} c \{post\}$$

where c is a specific program code, pre indicates the preconditions in the program, which describes the program state before executing c , and $post$ indicates the post condition in the program, which describes the programs the program state after executing c . Such a Hoare logic triple indicates that if an input of the program c meets its pre , the output of c should meet $post$ after executing program c . Otherwise, program c must have errors.

In order to formalize the program, Hoare logic defines inference rules for each of grammar clauses. The detailed rules are as follows.

$$Assign \quad \frac{}{\{Q(E/x)\} x := E \{Q\}}$$

The rules is to perform the assignment statement $x := E$ on all x in the precondition $\{Q(E/x)\}$, that is, to replace all x in the precondition with E to obtain the post condition $\{Q\}$. Similarly, the following two rules can exist.

Preconditions reinforcement rule:

$$Pre \quad \frac{P \Rightarrow P' \quad \{P'\}c\{Q\}}{\{P\}c\{Q\}}$$

Post conditional weakening rule:

$$Post \quad \frac{\{P\}c\{Q'\} \quad Q' \Rightarrow Q}{\{P\}c\{Q\}}$$

By formalizing the program with these rules, we can get a series of constraints of the program, and then use SMT solver. Solve the corresponding rules.

2.3. Verification of neural networks

Deep neural network is a mapping in high-dimensional space, which can be formally expressed as $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$.

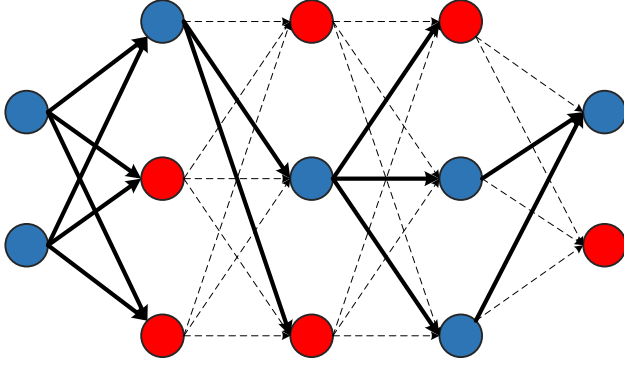


Figure 1. Execution path generation

if there is a set of constraints ϕ which is the precondition of f on $\mathbb{R}^n$, and existence a set of constraints φ which is the post condition of f on $\mathbb{R}^m$. Then, the problem of neural network verification is transformed into proving that $\forall x \in \mathbb{R}^n : \phi(x) \rightarrow \varphi(f(x))$ is satisfied or not.

3. Methodology

In this section, we propose an algorithm to derive the pre-condition of a pre-trained neural network from its post-condition, named DeepTBFV. The pre-condition can be used to verify and explain the behavior of the neural network and assist developers to increase the reliability of the neural network. First, we forward propagate a test case on a neural network, and generate the execution path of the test case. Second, we propose a technique to encode the execution path as a multivariate linear inequality system, and since the activation state of neurons is fixed in the execution path, the linear constraints generated by the execution path do not need to introduce the relaxation variables. Finally, we use the classification results obtained from this test case to define the post-condition of the test case and derive the pre-condition in reverse. The process of defining the post-condition is based on an assumption that will be described in the third summary.

3.1. Testing-based neural network execution path generation

The execution of a test case by the neural network means that the test case is used as the input for forward propagation. In the process of forward propagation, the state of each neuron is fixed. That is, we consider a neuron n_i in a neural network NN to be activated for the test case if its output is greater than a threshold value. Conversely, if n_i is less than the threshold, then the neuron n_i is not activated. We record the activation state of each neuron when the NN forward propagate for the test case.

Formally, we define all neurons in NN as the set *neurons*. Then, the neurons activated in NN can be defined as the set $ActiveN = \{n \mid n \in neurons \wedge Out(n) > \theta\}$, where θ is the threshold in neurons and $out(n)$ record as the output value of neuron. Similarly, the inactive neurons in NN can be defined as the set $InactiveN = \{n \mid n \in neurons \wedge Out(n) \leq \theta\}$. as shown in figure 1, in the forward propagation of neural network, we use blue for the neurons activated in the test case and red for the inactive neurons. Then, the process of assigning activation states to the neuron in *neurons* through the test cases is called execution path generation.

3.2. Formal modeling of neural network pathes

Since the state of activation of each neuron is fixed, we can convert the state of each neuron into a multiple linear inequality. Let $Neuron_j^i = \{x_1^{i-1}, x_2^{i-1}, \dots, x_n^{i-1}\}$ is the set of inputs of the j -th neuron in the i -th layer of the neural network NN . If the activation function of Neuron is ReLU and is activated by the test case, the activation state of the neuron can be modeled as follows,

$$\begin{aligned} &If : Neuron_j^i \in ActiveN \\ &Then : Out(Neuron_j^i) = w_1^{i-1} \cdot x_1^{i-1} \dots + w_n^{i-1} \cdot x_n^{i-1} > 0 \end{aligned}$$

Similarly, if the neuron is not activated by the test case, then the activation state of the neuron can be modeled as follows,

$$If : Neuron_j^i \in InactiveN$$

Then : $Out(Neuron_j^i) = w_1^{i-1} \cdot x_1^{i-1} \cdots + w_n^{i-1} \cdot x_n^{i-1} \leq 0$

According to the above rules, we can model the process of forward propagation of test cases in the neural network into a system of multivariate linear inequalities. Algorithm 1 specifically describes how to model the execution path of nn .

Algorithm 1: Formal modeling of NN pathes

Input: $Neurons$; $ActiveN$; $InactiveN$
 // List of linear constraints
 corresponding to neural network
Output: $Constraint$

```

1  $Constraint = list()$ 
2 for  $neuron$  in  $Neurons$  do
3   if  $neuron$  in  $ActiveN$  then
4      $Constraint.append(Out(neuron) > 0)$ 
5     else
6        $Constraint.append(Out(neuron) \leq 0)$ 
7     end
8 end
9 return  $Constraint$ 

```

3.3. Deriving Preconditions from Postconditions

When the state of each neuron in the neural network is fixed, we only need to carry out the Floyd-Hoare Logic assignment statement for the constraint of each neuron. This means that the constraints of each neuron are replaced by variables through the assignment statement. The constraints of each neuron we can get only include the input of the neural network.

Specifically, let $Inp = \{x_1, x_2, \dots, x_n\}$, $Inp \in R^n$ as the input of the NN and $Out = \{y_1, y_2, \dots, y_m\}$, $out \in R^m$ as the output of the NN . Moreover, We record the replacement expression of the j -th neuron in the i -th layer as f_j^i . Then the constraint corresponding to neurons can be denoted as

$$f_j^i(x_1, x_2, \dots, x_n) \leq 0$$

or

$$f_j^i(x_1, x_2, \dots, x_n) > 0$$

The constraints of the output layer of the neural network are different from those of the middle layer. It should conform to the user-defined post-condition. Here, the constraints of neurons in the output layer can be denoted as,

$$\alpha_j \leq y_j(x_1, x_2, \dots, x_n) \leq \beta_j$$

Where $y_j(x_1, x_2, \dots, x_n)$ is a multivariate linear polynomial. j represents the j -th output of the output layer. The interval $O_j = [\alpha_j, \beta_j]$ means the post-condition of y_j .

In addition, we also propose a method to construct a post-condition for the path generated by test cases. If the test case activates a neuron in the output layer, it means that the output value of the activated neuron is greater than that of other inactive neurons. Therefore, we can formally record this statement as $y_j^a > y_i^{in}$, where $i \in \{1, 2, \dots, m\} \setminus \{j\}$, a represents activated neuron, in represents inactive neurons. The activated neurons are marked as j . Since the output value of a neuron has an upper and lower limit, such as $INT8$ quantization of a neural network, the output values of neurons are quantized to the $[-128, 127]$. Therefore, we can also give the upper and lower limits of the output value of neurons in the output layer. Let the upper and lower bounds of the output layer neurons be $[l, u]$, where the l denote upper bound, and u denote lower bound. Then,

$$\forall y_i \in Out, \exists \alpha_i \leq y_i \leq \beta_i$$

where $i \in \{1, 2, \dots, m\}$, and m is the number of output layer.

4. Case Study

In this section, we use a three-layer neural network to show how DeepTBFV works and to demonstrate its effectiveness. Consider a ReLU-based neural network G . The structure and weights of this network are shown in Fig 2. Without loss of generality, we can assume that $bias = 0$ for each neuron in G . We denote the

neurons of the input layer, the neurons of the hidden layer and the neurons of the output layer as $\{x_1, x_2\}$, $\{h_1, h_2, h_3\}$, $\{y_1, y_2\}$ respectively.

To execute DeepTBFV for G , all the steps are given in sequence as follows:

Step 1: Select the test case of interest as input to the neural network, suppose we use $x_1 = 1, x_2 = 2$ as the test case.

Step 2: Record the activation of each neuron in the hidden layer of the neural network,

$$h_1 = \text{ReLU}(-x_1 + 2x_2 = 3) \in \text{ActiveN}$$

$$h_2 = \text{ReLU}(2x_1 - x_2 = 0) \in \text{InactiveN}$$

$$h_3 = \text{ReLU}(x_1 + x_2 = 3) \in \text{ActiveN}$$

Step 3: Generate neural network middle layer constraints,

$$h_1 = -x_1 + 2x_2 > 0$$

$$h_2 = 2x_1 - x_2 \leq 0$$

$$h_3 = x_1 + x_2 > 0$$

Step 4: Generate output layer constraints, here we assume that the output layer has constraints,

$$5 < y_1 = h_1 - h_2 + h_3 < 7$$

$$3 < y_2 = h_1 + 2h_2 + 0.5h_3 < 5$$

Step 5: Through Floyd-Hoare Logic, the system of inequality equations for each neuron in the input layer is derived backwards,

$$2x_1 - x_2 \leq 0$$

$$-x_1 + 2x_2 > 0$$

$$-x_1 - x_2 < 0$$

$$3 < -0.5x_1 + 2.5x_2 < 5$$

$$5 < 3x_2 < 7$$

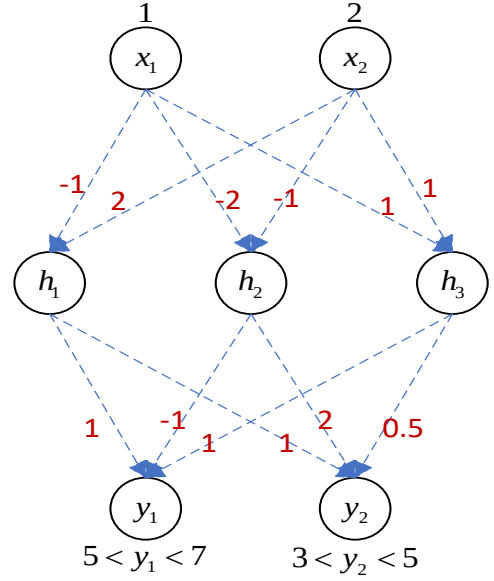


Figure 2. Example showing How to execute DeepTBFV

5. Related Work

In recent years, with the extensive application of deep learning, its reliability and interpretability have been increasingly concerned by the security community. Heuristic algorithms provide a fast and efficient way to interpret neural networks.

The heuristic algorithm provides a fast and effective way to explain the neural network. LIME[13] uses a testing-based method combine regression to explain the local characteristics of neural networks. Similar algorithms are anchors, etc[14, 15, 16]. Although heuristic algorithms can provide quick explanations, the resulting explanations are not guaranteed by formal verification.

Meanwhile, rigorous formal verification provides another perspective to ensure reliability and provide explanations. The work of reluplex et al.[17, 18, 19] performs formal verification of neural networks based on specification. It can guarantee that the verified neural network is safe and secure under formal specification.

However, reluplex[?] also proved that the verification problem of neural networks is NP hard. And the interpretation of verification-based neural networks is likewise based on verification solvers. Therefore, Trustable XAI et al.[20, 21] faces the same problem of difficulty in interpreting large neural networks.

6. Conclusion

We propose DeepTBFV, a test-based approach to generate execution paths for a test case in a neural network and derive the pre-condition(linear constraints) of the input to the neural network backwards by means of a custom post-condition. To my best knowledge, we are the first to apply the idea of TBFV to the field of verification and interpretation of neural networks.

7. Future Work

In this paper, we focus on how to use the principle of TBFV to generate the pre-condition of neural network. However, it did not analyze in detail how to generate test cases and how to use the generated constraints to verify or explain the behavior of neural networks. In future work, we will focus on the following two parts:

- How to choose the appropriate test case so that the constraints generated by DeepTBFV for that test case are more locally interpretable.
- How to analyze the constraints generated by DeepTBFV so that the constraint results can help people intuitively understand the neural network.

8. The Acknowledges

This authors acknowledge funding from the JST SPRING, Grant Number JPMJSP2132.

References

- [1] Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J. ‘Reluplex: An efficient smt solver for verifying deep neural networks’. In: Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30. (Springer, 2017. pp. 97–117
- [2] Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S. ‘Formal security analysis of neural networks using symbolic intervals’. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). (, 2018. pp. 1599–1614
- [3] Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R. ‘Efficient verification of relu-based neural networks via dependency analysis’. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34. (, 2020. pp. 3291–3299
- [4] Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., et al. ‘The marabou framework for verification and analysis of deep neural networks’. In: Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31. (Springer, 2019. pp. 443–452
- [5] Shriver, D., Elbaum, S., Dwyer, M.B. ‘Dnnv: A framework for deep neural network verification’. In: Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33. (Springer, 2021. pp. 137–150
- [6] Sun, Y., Huang, X., Kroening, D., Sharp, J., Hill, M., Ashmore, R. ‘Deepconcolic: Testing and debugging deep neural networks’. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). (IEEE, 2019. pp. 111–114
- [7] Pei, K., Cao, Y., Yang, J., Jana, S. ‘Deepxplore: Automated whitebox testing of deep learning systems’. In: proceedings of the 26th Symposium on Operating Systems Principles. (, 2017. pp. 1–18
- [8] Goodfellow, I.J., Shlens, J., Szegedy, C.: ‘Explaining and harnessing adversarial examples’, *stat*, 2015, **1050**, pp. 20

- [9] Liu, S. ‘Testing-based formal verification for theorems and its application in software specification verification’. In: Tests and Proofs: 10th International Conference, TAP 2016, Held as Part of STAF 2016, Vienna, Austria, July 5-7, 2016, Proceedings 10. (Springer, 2016. pp. 112–129
- [10] Wang, R., Liu, S., Sato, Y.: ‘Sit-se: a specification-based incremental testing method with symbolic execution’, *IEEE Transactions on Reliability*, 2021, **70**, (3), pp. 1053–1070
- [11] Liu, A., Liu, S.: ‘Enhancing the capability of testing-based formal verification by handling operations in software packages’, *IEEE Transactions on Software Engineering*, 2022, **49**, (1), pp. 304–324
- [12] Pratt, V.R. ‘Semantical considerations on floyd-hoare logic’. In: 17th Annual Symposium on Foundations of Computer Science (sfcs 1976). (IEEE, 1976. pp. 109–121
- [13] Ribeiro, M.T., Singh, S., Guestrin, C. ‘” why should i trust you?” explaining the predictions of any classifier’. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. (, 2016. pp. 1135–1144
- [14] Ribeiro, M.T., Singh, S., Guestrin, C. ‘Anchors: High-precision model-agnostic explanations’. In: Proceedings of the AAAI conference on artificial intelligence. vol. 32. (, 2018.
- [15] Lundberg, S.M., Lee, S.I.: ‘A unified approach to interpreting model predictions’, *Advances in neural information processing systems*, 2017, **30**
- [16] Smilkov, D., Thorat, N., Kim, B., Viégas, F., Wattenberg, M.: ‘Smoothgrad: removing noise by adding noise’, *arXiv preprint arXiv:1706.03825*, 2017,
- [17] Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: ‘Efficient formal safety analysis of neural networks’, *Advances in neural information processing systems*, 2018, **31**
- [18] Gehr, T., Mirman, M., Drachler, D., Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M. ‘Ai2: Safety and robustness certification of neural networks with abstract interpretation’. In: 2018 IEEE symposium on security and privacy (SP). (IEEE, 2018. pp. 3–18
- [19] Ehlers, R. ‘Formal verification of piece-wise linear feed-forward neural networks’. In: Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15. (Springer, 2017. pp. 269–286
- [20] Ignatiev, A., Narodytska, N., Marques, J. ‘Abduction-based explanations for machine learning models’. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33. (, 2019. pp. 1511–1519
- [21] Calegari, R., Ciatto, G., Omicini, A.: ‘On the integration of symbolic and sub-symbolic techniques for xai: A survey’, *Intelligenza Artificiale*, 2020, **14**, (1), pp. 7–32

自然言語処理を用いた仕様書から VDM++仕様への変換支援について

酒井 玲弥
仙台高等専門学校
sakainit@gmail.com

岡本 圭史
仙台高等専門学校
okamoto@sendai-nct.ac.jp

要旨

ソフトウェアの不具合がもたらす悪影響は大きく、信頼性を高めるための設計手法の一つとして形式手法が用いられている。VDMは形式手法の一種であり、形式仕様記述言語 VDM++を持つ。VDMは実際の開発現場においても活用され、成果を挙げている。しかし、自然言語の要求をモデル化するプロセスにおいては、計算機による支援が進んでいない。そのため、人手による作業がほとんどであり、形式仕様の作成者への負担が大きいという問題がある。本論文では、形式手法導入コスト低減を目指した、自然言語で記述された要求仕様書から VDM++仕様への変換手法の提案を試みる。また、型定義、関数・操作定義に関する支援手法を実装し、評価を行う。評価の結果、人間の感覚に近い用語抽出を目指した、複合名詞の結合手法の有効性を確認することができた。しかし、定義する用語を効率的に探索することを目的とした抽出した語の並べ替え手法は、並べ替えを行わない場合と比較して精度は向上したが、改善の余地が残る結果となった。以上の結果をもとに VDM++で定義される用語の特徴について考察を行い、得られた知見を共有する。

1. はじめに

日常生活を支えるシステムがソフトウェア集約的になるにつれ、ソフトウェア技術への依存度が高まっている。また、近年のソフトウェアは規模の拡大、複雑化を続けている。そのため、ソフトウェアの不具合がもたらす悪影響は大きく、信頼性を高めるための設計手法が求められる。形式手法は、数理論理に基づいて設計された形式仕様言語による、曖昧さが排除された仕様記述と設計方法論、及び専門知識を総合し、システムに求められる信頼性の達成を目指すアプローチである。形式手法は、実際の開発現場においても活用されており、成果を挙げている[1]。VDM(Vienna Development Method)は、形式手法の一種であり、世界初の ISO 標準仕様記述言語である VDM-

SL と VDM++を持っている。これら VDM の形式仕様記述言語は VDM Tools[2]や Overture Tools[3]といったツールにより仕様の検証が可能であり、VDM Tools には C++と Java のコード生成機能がある。加えて、拡張機能として C#コードへの変換も開発されるなど、VDM++を開発現場に取り入れるための研究開発が活発に行われている[4][5]。また、IC カードの Felica ファームウェア開発プロジェクトでは、設計プロセスにおいて VDM++が適用されており、日本語での開発現場においても形式手法が適用されている[6]。以上のようにソフトウェアの開発工程に形式手法を適用することには、あいまいさの除去やツールによるテストの自動化など多くの利点があるが、形式手法を適用した開発の初期段階である、自然言語の要求をモデル化するプロセスにおいては、計算機による支援が進んでいない。そのため、人手による作業がほとんどであり、形式仕様の作成者への負担が大きいという問題がある。

以上の問題を解決するアプローチとして、VDM++仕様作成プロセスの一部自動化による変換支援手法を試みた。VDM++仕様作成を人手で行う際に発生する作業である、モデル化する用語の抽出作業を支援する。また、変換支援プロセス内で発生する抽出された単語のリストをもとに、見落としや自然言語の曖昧さに起因する曖昧な定義や冗長な定義を防止する。以上により、形式手法導入コストが低減され、プロジェクトへの形式手法導入が容易になるため、ひいては高品質なソフトウェア開発が可能になると考えられる。本論文の目的は、以上のアプローチに関する実験で得られた知見の共有である。モデル化する用語の抽出作業への支援として実装を行った、候補となる単語の抽出と並び替えの実験に関して、結果と考察を論じる。

2. 関連研究

2.1. VDM++仕様と自然言語仕様を併せた品質向上

大森らの研究では、仕様策定過程の全体を通じて、

自然言語で記述された仕様と形式手法による形式モデルの相互変換により、自然言語記述の品質向上を行う手法が提案された[8]。これにより、関係者の共通理解の得やすい自然言語記述と、厳密な形式モデルの長所を両立させることが可能となる。この手法では、自然言語と形式モデルの相互変換を頻繁に行うため、支援ツールの開発が行われた。支援ツールにおいては、キーワードのマーキングを行い、マークしたキーワードの定義を記述する用語辞書とキーワードの関連を構造化するデータ辞書の生成を補助する。そして、生成された辞書をもとに形式モデルのテンプレートが出力可能である。しかし、支援ツールにおけるキーワード抽出は手動で行う必要がある。手動によるキーワード抽出は、一般に使われる意味とは違う、対象ドメイン特有の意味が存在する単語に対応できるなどの利点があるが、大規模な要求仕様書に適用する場合、キーワードの抽出に大きな労力がかかると考えられる。そのため、本論文で提案する手法では、キーワードの候補となる単語、複合語を自動で抽出し、作業量を低減する。また、モデル化する単語を自動抽出された候補の中から人手により選択することで、手動による作業の利点を損なわない。

2.2. 機械学習を用いたアプローチ

執行らは、機械学習を用いた VDM++仕様生成へのアプローチとして、自然言語仕様から単語を抽出し、多項ロジスティック回帰分析を用いて型定義と定数定義への分類を行った[9]。また、菅らの研究では、執行らの手法に加え、日本語 WordNetを用いて、ある単語の下位概念との関係を表す概念レベルを算出し、クラスとインスタンス変数となる単語の分類を行った[10]。これらの手法は、関数・操作の定義に対応していないが、本論文では動詞の抽出と係り受け関係による支援手法を提案している。機械学習による単語の分類を行うにあたって、関連研究では、自然言語の仕様書を教師データとして、事前に学習済みモデルを用意していたが、仕様書の分野が変わると予測精度が低下するという問題がある。要求仕様書は公開されているものが少ないため、最適な教師データを手に入れられるとは限らない。本論文のアプローチでは、入力文書に対する自然言語処理で支援が完結するため、要求仕様書の分野や所持しているデータ量によらない支援が可能である。また、機械学習を用いた VDM++仕様の自動生成は、モデル作成者の作業量がかからず、知識が不要であるという利点があるが、モデル化に必要な単語を網羅するには至っておらず、誤った定義や定義漏れを避けられない。要求仕様の完全性という

観点において、自然言語仕様に記述されている仕様を網羅することは重要である。そのため、本論文では一部自動化による変換支援という異なるアプローチにより、人間の知識を一部介入させる。これにより、モデル化する単語の正確性が向上し、ひいては、モデル化する仕様の網羅性の向上が期待できる。

3. 変換支援手法のアプローチ

3.1. 変換支援手法の使用シナリオ

大森らの手法[8]では、VDM++を用いた開発フェーズとして、図 1 のような手順を想定している。

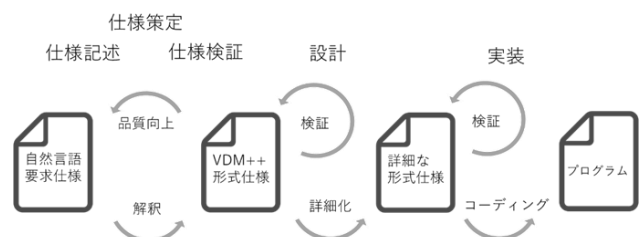


図 1 要求の洗練プロセス (大森 2010, p.20)[8]を参考に作成

本手法は、図 1 のプロセス中の自然言語仕様を VDM++へ変換する、仕様策定段階での使用を想定している。3.2 節以降で示す手法は、変換作業の一部自動化と必要な知識を作成者が補うことで、VDM++への変換を半自動的にを行い、最終的に陰仕様の完成を目指すものである。陰仕様では、要求仕様書中に記述される「要求」と「仕様」をモデル化し、数理的な性質を満たしているか静的検証による仕様検証が可能である。

本論文においては、以上の一部実装として、型とシグニチャの抽出作業の支援に注力した。作成者は自然言語で記述された仕様書から、仕様をモデル化するために必要な単語を抽出する。抽出作業において、必要な単語の候補をシステムが自動で優先順位をつけて抽出し、作成者は型や関数になる単語を選択することで、モデリングが可能である。以上の手法を取るため、VDM++では、様々な抽象度での仕様記述が可能であるが、本手法によって作成される VDM++仕様は、自然言語仕様に含まれる単語を各要素に対応付けた、逐語訳的な変換をされたものである。よって本手法は、形式手法および VDM++の記述に関して最低限の知識があり、初歩的な VDM++仕様が記述できる技術者を対象としている。

3.2. 変換支援手法全体の流れ

変換支援手法全体の流れを図2に示す。入力された仕様書データを分析するため、形態素解析と構文解析を行う。そして、複合名詞の結合を自動で行い、類義語の統合を行うことで、後のモデル化に適した分かち書きデータを得られる。次に、分かち書きデータをもとに、型定義の支援を行う。型として定義される候補となる単語の抽出を自動で行い、型候補のリストをもとに人手により型の定義を行う。ここで、単語を無作為に提示すると人間が判断する上で効率が悪いので、型候補のリストは、型になる可能性が高い単語を上位に並べ替えたリストを提示する。ここまでで得られる、分かち書きデータと定義された型をもとに関数・操作定義の支援を行う。関数・操作の候補となる単語を自動で抽出し、得られた関数・操作候補のリストをもとに人手により関数・操作の定義を行う。関数・操作の定義も同様に最終判断を人手により行うため、候補のリストは、関数・操作になる可能性が高い単語を上位に並べ替えたリストを提示する。選択した型および関数・操作は辞書データとしてまとめられ、辞書データをもとにVDM++陰仕様が出来上がる。

3.3. 形態素解析・構文解析

はじめに、型定義や関数定義に必要な情報である、各単語の品詞情報を特定するため、形態素解析をツールの使用により行う。本論文では、形態素解析ツールとして、Sudachi を使用している。なお、Sudachi の辞書はデフォルトの Core、分割モードは最小単位で分割されるモードの A を選択している。構文解析は機械学習ベースの自然言語処理ツールである GiNZA[12]を使用した。GiNZA は spaCy と Sudachipy を用いて実装されており、spaCy の CNN マルチタスク学習器を用いて日本語

Universal Dependencies(以下 UD) の品詞ラベルと係り受けのラベルを付与する。また、GiNZA v5 では Transformers モデルにより、さらに高精度な構文解析を実現している。

3.4. 複合名詞の結合

要求仕様書はシステムの機能および、実現方法が記述された文書であり、一般的な文書と比較して専門性の高い用語や、システム独自の用語が用いられる可能性が高い。また、「話題沸騰ポット要求仕様書」[11]のVDM++仕様を人手により作成した際にモデル化した用語にも、複合名詞が多いという特徴が見られた。中川らは「専門用語の多くは複合語、とりわけ複合名詞であることが多い。」(中川 2003, p.27)[13]と述べており、専門性が高い文書である要求仕様書には複合名詞が多く含まれていると言える。以上より、要求仕様書をモデル化する際、複合名詞の用語を抽出することで人間の定義する型に近い用語抽出が行えるという仮説を立てた。また、専門用語の抽出に関して中川らの手法では、最小単位の名詞および複合名詞を候補として抽出し、優先順位付けを行うことで抽出を行っている[13]。VDM++において必要な単語は前述の通り、専門用語が多いと考えられるため、複合名詞の結合および、3.5 節で後述する並べ替えに関して、「話題沸騰ポット要求仕様書」に対し複数の実験を行い、型として定義される単語について考察する。

複合名詞の結合に関して、形態素解析の段階で、複合名詞を一つの単語として分割するアプローチと、最小単位に分割された単語を後から結合するアプローチが考えられる。形態素解析の段階で分割する場合は、各ドメイン知識に対応できる辞書をそれぞれ構成する必要があるが、現状では困難である。そのため本論文では、形態素解析により最小単位に分割された単語をもとに、後から

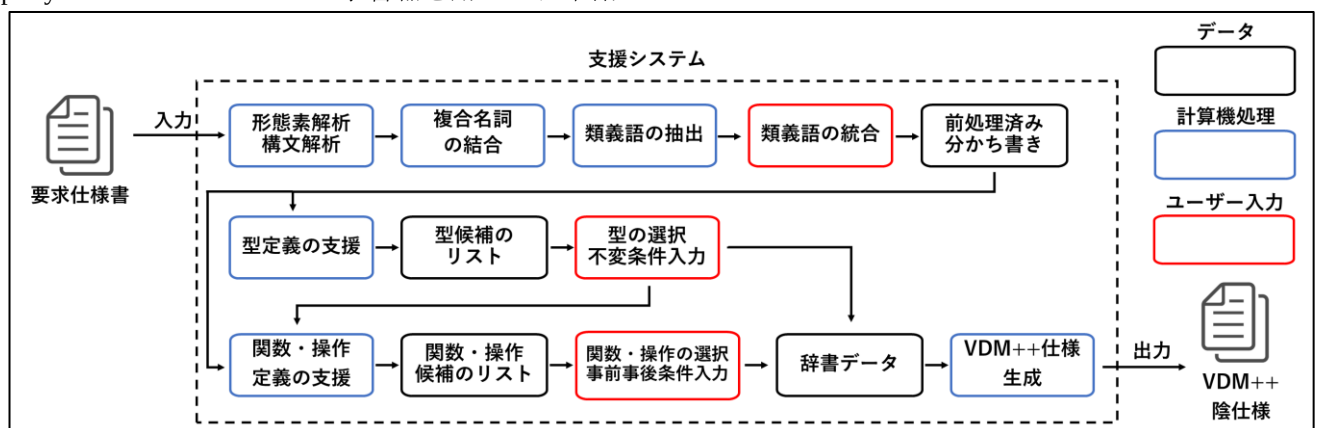


図2 変換支援手法全体の流れ

結合を行うことで、特定のドメイン知識によらない複合名詞抽出を行う。

3.5. 型定義の支援

本節で提案する手法では、名詞を列挙し、型になる可能性が高いと予想される名詞を優先的に表示することで、型定義を支援する。VDM++の一般的な作成手順で、「主に名詞から型を定義する」[7]とされているように、VDM++で型としてモデリングされるのは、要求仕様書中に含まれる名詞が主となる。そこで、本論文の変換支援手法では要求仕様書中に含まれる名詞および複合名詞を自動で抽出し、型の候補とする。また、型の候補は抽出された名詞および複合名詞全てとしている。全てを候補とすることで、作成者が確認すべき候補の数が多くなるというデメリットはあるが、本論文では、モデル化すべき単語の取りこぼしを減らすことを重視している。また、要求仕様書は専門性の高い文書であり、要求仕様書に含まれる用語の意味を正確に解釈することは困難である。そのため、最終的な判断は要求仕様作成者や対象ドメインの専門家のような人間によって行うほうが、正確な仕様記述が可能であると考えられる。しかし、型の候補を無作為に提示すると、作成者が必要な単語を探索する際の効率が悪いと、型になる可能性が高いと考えられる単語が上位になるように並べ替えることで、作成者は目的の単語を早期発見できるようになり、用語探索の負担が軽減される。

型になる可能性が高い単語について、専門用語のような文書の特徴が現れる単語は型になりやすく、多くの文書に含まれる一般的な名詞は型になりにくい。という仮説のもと、型候補の並べ替えに関して2つの実験を行う。

3.6. 関数・操作定義の支援

本節で提案する手法では、動詞を列挙し、関数・操作になる可能性が高いと予想される動詞を優先的に表示することで、関数・操作の定義を支援する。VDM++の一般的な作成手順で、「主に述語(動詞)から、関数・操作のインターフェース(シグネチャ)を記述する」[7]とされているように、VDM++で関数・操作としてモデリングされるのは、要求仕様書中に含まれる動詞が主となる。そこで、本論文の変換支援手法では要求仕様書中に含まれる動詞を自動で抽出し、関数・操作の候補とする。前節の型候補の並び替えと同様に優先的に提示する候補が持つ要素を考え、動詞が影響を及ぼす対象である目的語の関係を注目した。目的語は文章中の「何を、誰を」に当たる要

素である。これを VDM++に置き換えて考えると、操作はインスタンス変数を変える可能性がある操作であるため、影響を及ぼす対象が存在していると考えられる。また、検証用 VDM++仕様では、型名とインスタンス変数名が共通している定義が多かった。そこで、すでに定義されている型を目的語とする動詞を抽出することで、操作にあたる動詞を抽出できると仮定し、実験を行う。動詞の抽出に際し、抽出した動詞を形態素解析辞書の見出し語に変換する(レンマ化)。これにより、活用形を終止形に統一する。型を目的語とする動詞を抽出できたが、前述の仮定は操作の定義に対するものであり、インスタンス変数へアクセスしない関数の定義には対応できていないと考えられる。しかし、型候補の抽出と同様に、モデル化すべき単語の取りこぼしを減らすことを重視するため、型を目的語とするものの以外の動詞も関数の候補として提示する。

4. 変換支援手法の実現

4.1. 複合名詞の結合

複合名詞の抽出手法は複数あるが、本論文では2つの手法について実験を行った結果を示す。はじめに、仮説検定を用いたコロケーション抽出について実験を行った。コロケーションとは、物事について言及するため、慣用的に使われる2つ以上の単語からなる表現である。コロケーション抽出を行う統計的な手法として仮説検定を用いた方法が複数あり、その中で最も利用されているのが、尤度比検定に基づく手法である[14]。仮説検定は、データの統計量に基づいて、ある仮説の正否を検証する手法であり、尤度比検定は統計量として2つのモデルの尤度の比を用いている。算出した対数尤度比が高いほど、共起する可能性が高いペアだと言える[15]。対数尤度比が高い単語のペアを結合した複合名詞を表1に示す。

表1 対数尤度比が上位20個の複合名詞

	複合名詞	対数尤度比			
1	保温行為	171.616	11	水位センサ	91.857
2	温度制御	168.492	12	タイムアップ	86.041
3	蓋センサ	146.218	13	表示窓	83.608
4	操作量	124.574	14	min0	72.379
5	100msec	121.324	15	ロック解除	70.159
6	制御方式	115.356	16	0sec	68.151
7	要求仕様	111.954	17	水位メータ	63.178
8	沸騰行為	109.519	18	カルキ抜き	59.430
9	操作パネル	106.863	19	タイマボタン	58.043
10	残り時間	105.416	20	msec以上	56.267

次に、機械学習を用いた手法として、機械学習ベースの構文解析ツールである GiNZA を用いた実験を行う。3.3 節の通り、GiNZA は Transformers モデルを用いて日本語 UD の品詞ラベルと係り受けのラベルを付与する。GiNZA によって付与される、日本語 UD 係り受けタグの中に、複合名詞を表す”compound”関係があり、最小単位に分割された単語のなかで、複合名詞を構成しているものに付与される。そこで、”compound”関係を付与された単語同士を結合させることで、複合名詞とする。入力データの記述には、改行を句点相当の文の区切りとして扱う箇条書きのような記述が存在するが、GiNZA の出力で改行は空白として処理されており、文の区切りが不明瞭になっている場合があった。そのため、GiNZA に入力するデータは改行(¥n)を句点に置き換えている。また、”compound”と分析された単語の中で、名詞化の「こと」を含む組み合わせおよび、「()」の中にあることで、複合名詞とされているサ変可能な名詞を例外として結合しないこととした。 ”compound”関係の単語を結合した結果は表 2 のとおりである。

表 2 GiNZA により抽出された複合名詞(一部)

	複合名詞				
1	ハードウェア構成	11	水位メータ	21	要求仕様
2	ハードウェア要求仕様	12	温度制御行為	22	操作パネル
3	内部構成	13	沸騰行為	23	給湯口
4	操作パネル部	14	保温行為	24	ポット上部
5	操作要求仕様	15	温度制御方式	25	正面側
6	沸騰ボタン	16	エラー検知	26	貯水部
7	保温設定ボタン	17	状態遷移	27	満水センサ
8	解除ボタン	18	話題沸騰ポット状態遷移図	28	買い物袋
9	給湯ボタン	19	話題沸騰ポット状態遷移表	29	未使用時
10	タイマボタン	20	話題沸騰ポット	30	本体接合部

4.2. 型定義の支援手法

・TF-IDF を用いた並べ替え

TF-IDF とは文章に含まれる単語の重要性を表す指標である。TF(Term Frequency) は単語の出現頻度、IDF(Inverse Document Frequency)は逆文書頻度を意味している。つまり、同じ文章内で多く出現し、他の文章で出現していない単語の重要度が高くなる。「話題沸騰ポット要求仕様書」から名詞を抽出し、GiNZA による複合名詞の結合を行った型候補のリストの各単語に対し、TF-IDF を算出し、TF-IDF の高い順に並べ替えた結果を表 3 に示す。

表 3 TF-IDF による並べ替え結果

	型候補					
1	水温	11	時	...	385	通常
2	on	12	off	...	386	際
3	保温行為	13	コンセント	...	387	上
4	°C	14	タイマ	...	388	他
5	ヒータ	15	こと	...	389	度
6	沸騰行為	16	セット	...	390	数
7	温度	17	ブザー	...	391	方
8	ポット	18	水量	...	392	状況
9	蓋	19	タイマボタン	...	393	設計
10	表示	20	操作量	...	394	もの

・FLR を用いた並べ替え

FLR とは、中川らが提案した専門用語抽出手法[13]で用いられた重要度を表す指標である。中川らは、ある単語や複合語が持つ分野固有の概念への関連性の強さである、ターム性と、ある単語や複合語がコーパス内で安定して使用されている度合いを表すユニット性に注目し、これらの性質反映した重要度の指標として、FLR を提案した。「話題沸騰ポット要求仕様書」から名詞を抽出し、GiNZA による複合名詞の結合を行った型候補のリストの各単語に対し、FLR を算出し、FLR の高い順に並べ替えた結果を表 4 に示す。

表 4 FLR による並べ替え結果

	型候補					
1	沸騰行為	11	水位	...	385	塩素
2	温度	12	モード	...	386	殺菌
3	温度制御行為	13	on	...	387	上
4	保温行為	14	時	...	388	タイミング
5	水位センサ	15	タイマボタン	...	389	無意識
6	温度制御方式	16	表示	...	390	指示
7	ポット	17	蓋	...	391	目次
8	蓋センサ	18	タイマ	...	392	状態 (アイドル)
9	沸騰ボタン	19	ヒータ	...	393	コストダウン
10	状態	20	時間	...	394	dE

4.3. 関数・操作定義の支援手法

検証用 VDM++仕様で定義された型に対して変換支援手法を適用し、型を目的語とする動詞を抽出した結果を表 5 に示す。

表 5 型を目的語とする動詞(一部)

	型	操作候補			
1	水位	検出	11	保温行為	中止
2	水温	検出	12	給湯ボタン	押す
3	コンセント	差し込む	13	給湯	停止
4	コンセント	抜く	14	タイマボタン	押す
5	蓋	閉じる	15	タイマ	起動
6	水位	確認	16	タイマ	操作
7	沸騰行為	する	17	タイマ	リセット
8	蓋	閉める	18	タイマ	停止
9	蓋	開ける	19	ブザー	鳴らす
10	沸騰ボタン	押す	20	操作パネル	見る

動詞の抽出に際して、関数・操作として定義されないと考えられる動詞の除外を行う。係り受け関係 fixed で助動詞に接続されている動詞は、複合機能表現と呼ばれる、単語の組み合わせによって慣用的に用いられる表現である。例としては、「～である」の「ある」や「～における」の「おく」などが該当する。これらの動詞は、組み合わせの一部であり、動詞の意味を直接表現するものではないため、除外する。また、サ変可能な名詞と「する」が接続され、サ行変格活用の動詞になっている場合、動作は語幹となる名詞に表れるため、名詞部分を抽出する。「する」が動詞の場合単体では動作の内容を特定できないため、候補から除外するが、助詞の「を」と接続し「～をする」の形(話題沸騰ポット中に含まれる例としては「沸騰行為をする」など)になっているとき、「を」の直前の語が目的語となり、「する」が動詞になる。このときの「する」は何かしらの動作を起こすという意味で、目的語で動作の内容を特定できると考えられるため、「～をする」は一つの動詞として扱う。以上により、型を目的語とする動詞のリストとそれ以外の動詞から一部を除外したリストが得られる。ここで、さらに効率的に単語を探索できるよう、得られたリストに対し個別に並べ替えを行う。並べ替えは、要求仕様書中に頻繁に表れる動詞ほど関数・操作となる可能性が高いという仮定のもと、出現頻度の高い順に並べ替える。並べ替えを行った結果を表 6 に示す。

表 6 並べ替えを行った関数・操作候補のリスト

	関数・操作候補				
1	押す	11	リセット	...	125 求
2	表示	12	保つ	...	126 取り除く
3	鳴らす	13	OFF	...	127 合う
4	検出	14	点灯	...	128 止まる
5	停止	15	起動	...	129 計算
6	消灯	16	知る	...	130 代用
7	閉じる	17	沸騰行為をする	...	131 対応
8	検知	18	確認	...	132 断線
9	制御	19	中止	...	133 カルキ抜きをする
10	止める	20	差し込む	...	134 吊り下げる

5. 変換支援手法の評価・考察

5.1. 検証用 VDM++仕様データの作成

支援手法の検討、検証を目的とし、「話題沸騰ポット要求仕様書」の VDM++仕様を人手により作成した。また、VDM++仕様として抽出される単語は一意に定まらないため、研究室所属の学生 3 名に「話題沸騰ポット要求仕様書」の VDM++仕様作成を依頼することで、複数パターンの VDM++仕様を作成した。なお、作成を依頼した学生は、プログラミングの基礎知識や、ソフトウェア工学の基礎知識を有するが、VDM++によるモデリングの経験はなかったため、VDM++モデリングについて 90 分程度の簡単な講習を行った後に、話題沸騰ポット要求仕様書についてモデル作成を行った。作成者のスキルとモデリングにかかった時間を表 7 に示す。

表 7 VDM++作成者のスキルと作成時間

作成者	学年	形式手法に関する経験	作成時間
著者	専攻科 2 年	- UML によるモデリングの講義 90 分×16 週を履修済み - Alloy を用いた形式手法の講義 90 分×16 週を履修済み - VDM++の 3 ヶ月程度の学習およびモデリング	4 時間
学生 A	専攻科 1 年	- UML によるモデリングの講義 90 分×16 週を履修済み - モデル検査手法 NuSMV および Spin の 20 時間程度の学習 - VDM++に関する 90 分程度の講習 - VDM++モデリングの 20 時間程度の学習	6.2 時間
学生 B	本科 4 年	- UML によるモデリングの講義を履修中 - VDM++に関する 90 分程度の講習	2 時間
学生 C	本科 4 年	- UML によるモデリングの講義を履修中 - VDM++に関する 90 分程度の講習	2.7 時間

なお、VDM++仕様はサブクラスへの分割を行わず、「話題沸騰ポット要求仕様書クラス」としてモデリングを行った。また、作成する仕様は陰仕様までとした。作成した VDM++仕様の一例を表 8、表 9 に示す。

表 8 著者作成 VDM++(一部)

```

types
public コンセント = <差し込まれている>|<抜けている>;
public 満水センサ = <on>|<off>;
public 第 n 水位センサ = <on>|<off>;
public 蓋センサ = <on>|<off>;
public ロック = <ロック>|<ロック解除>;
public サーミスタ = real
inv temp == -10 <= temp and temp <= 150;
public ヒータ用電源 = <on>|<off>;
...
public 保温ランプを点灯 : () ==> ()
    保温ランプを点灯() == is not yet specified
post 保温ランプ = <点灯> and 沸騰ランプ = <消灯>;

```

表9 学生A作成 VDM++(一部)

```

types
public t稼働状態 = <on>|<off>;
public t温度 = real
  inv w温度 == -10.0 <= w温度 and w温度 <= 150.0;
public t水位 = <state0>|<state1>|<state2>|<stat...
public tモード = <高温>|<節約>|<ミルク>;
public t操作パネル ::
  沸騰ボタン : t稼働状態
  給湯ボタン : t稼働状態
  保温設定ボタン : t稼働状態
  タイマボタン : t稼働状態
  ...
public トリガ_保温設定ボタン : () ==> ()
  トリガ_保温設定ボタン() == is not yet specified
pre i操作パネル.モード = <高温> or i操作パネル.モ
ード = <節約> or i操作パネル.モード = <ミルク>
post i操作パネル.モード = <高温> or i操作パネル.モ
ード = <節約> or i操作パネル.モード = <ミルク>;

```

5.2. 評価指標

本章で使用する評価指標について述べる. 複合名詞の結合手法に関して, 再現率(Recall)を用いて評価を行う. 再現率は検索や推薦, 2 値分類の機械学習などのタスクを評価する際に用いられる指標である. 再現率の最小値は 0.0, 最大値は 1.0 となり, 値が大きいほど, 正例と予測すべきもののうち, 正例と予測できたものの割合が大きいことになる. また, 再現率は一般に, 正例と予測したもののうち, 実際に正例であったものの割合を表す, 適合率(precision)とともに用いられる.

型候補の並べ替えについて, PR 曲線(Precision-Recall Curve)と平均適合率(AP: Average Precision)を用いて評価を行う. PR 曲線は, 再現率を縦軸に, 適合率を横軸に取ったグラフであり, 予測データ数をしきい値によって変動させ, 各データ数における再現率と適合率をプロットする. 再現率と適合率はトレードオフの関係になっており, しきい値を下げ予測データ数を増やすほど, 再現率は上昇し, 適合率は低下する. 再現率が上昇しても適合率の低下が少ないものが理想的であるため, 予測の精度が良いほど右上に曲線が膨らむ. また, しきい値ごとの適合率を加重平均した値である AP や PR 曲線の下側の面積を積分計算した AUC によって PR 曲線は評価できる.

5.3. 複合名詞の結合手法についての評価・考察

複合名詞の結合手法について妥当性の評価を行う. 予測データは 4.1 節の実験により結合させた複合名詞と結合しなかった名詞から重複を取り除いた単語のリストで

ある. また, 複合名詞を結合しなかった場合の, 最小単位に分割された名詞のみのリストについても検証する. 正解データは, 検証用 VDM++仕様で, 型として定義された単語を合わせて重複を取り除いたものである(表 10).

表 10 型の正解データ(一部)

	型			
1	満水センサ	11	タイマ	...
2	第 n 水位センサ	12	タイマ状態	38
3	蓋センサ	13	水位メータ	39
4	ロック	14	保温モード	40
5	サーミスタ	15	保温温度	41
6	ヒータ用電源	16	水量	42
7	ランプ	17	動作状態	43
8	インジケータセル	18	沸騰行為	44
9	ボタン	19	保温行為	45
10	ボタンタイマ	20	稼働状態	46

3.1 節の通り, 本手法はモデル化すべき単語の選択は作成者の判断に依存している. そのため本手法では, モデル化すべき単語の取りこぼしを減らすことを重視しており, 候補の中から実際にモデル化されなかった単語の数は重視しないため, 再現率のみを用いて評価を行う. 各手法によって抽出された単語のリストの再現率は表 11 のとおりである.

表 11 各手法の再現率

複合名詞の結合手法	再現率	TP	TP+FN
複合名詞の結合をしない	0.43	20	46
GiNZA	0.76	35	46
LLR上位10ペアを結合	0.52	24	46
LLR上位20ペアを結合	0.57	26	46
LLR上位30ペアを結合	0.59	27	46
...	...	...	...
LLR上位80ペアを結合	0.65	30	46
LLR上位90ペアを結合	0.65	30	46
LLR上位100ペアを結合	0.67	31	46

表 11 より, 複合名詞の結合をしない場合に対して, GiNZA と LLR 双方ともに再現率が高くなった. この結果から, 人間が型として抽出する単語は複合名詞が多いと言える. また, GiNZA によって結合した結果の再現率が最も高く, 人間が抽出する複合名詞に最も近い手法であると言える.

以上の結果をもとに, 複合名詞の結合に関して考察を述べる. 抽出できなかった単語は, [ボタンタイマ, タイマ状態, 保温モード, 保温温度, 動作状態, 稼働状態, ロック解除中ランプ, 制御ランプ, 制御状態, インジケータランプ, タイマ時間]である. これらの単語のうち, 保温温度以外の単語は要求仕様書中に含まれていない単語で, VDM++仕様作成中に人間の知識によって個人の理解しやすい用語が作られたためである. 例えば, 「話題沸

「沸騰ポット要求仕様書」には、沸騰させた湯を保温する際の仕様として、高温モードは98℃、節約モードは90℃、ミルクモードは60℃で保温するという仕様がある。これらをモデリングする過程で、現在のモードや、各温度を一つの型で管理するほうが適切だと考え、「保温モード」や「保温温度」といった単語が定義されたと考えられる。以上のように、本論文で用意したVDM++仕様は、関数・操作のシグネチャまで記述しているため、仕様記述中に洗練が進んだのである。現行の技術では、要求仕様書中に含まれない語彙を抽出することは困難であるが、上位、下位概念の学習やなど組み合わせることで、予測できる可能性があると考えられる。

5.4. 型候補並べ替えについての評価・考察

型候補並べ替え手法について妥当性を評価する。3.4節の実験で並べ替えを行った型候補のデータに関して、PR曲線をプロットし、APを算出する。TF-IDFによる並べ替えを行ったデータのPR曲線を図3、FLRによる並べ替えを行ったデータのPR曲線を図4に示す。また、図3のAPは0.36、図4のAPは0.36である。

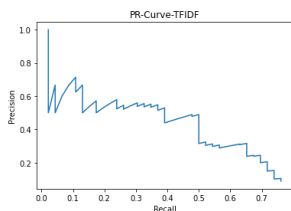


図3 TF-IDFのPR曲線

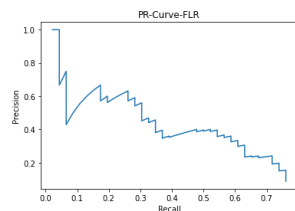


図4 FLRのPR曲線

以上の結果から、並べ替え手法の精度はあまり高くないと言える。並べ替えを行わない(要求仕様書中出现する順)場合のAPは0.21であるため、並べ替えを行わない場合と比較して精度は向上していると言える。しかし、図3、4より型として定義されない単語が上位に複数存在していることがわかる。

以上の結果をもとに、型候補並べ替えに関して考察を述べる。TF-IDFによる並べ替えでは、「ランプ」、「ボタン」、「センサ」といった単語が下位になった。これらの単語は「蓋センサ」や「沸騰ボタン」というように複合名詞となることで、頻度が下がったためだと考えられる。これらの具体的な要素を汎化したような単語の重要度を上げるには、WordNetのような単語の上位、下位概念を指標に取り入れる必要があると考える。また、上位になった定義されない単語の例として、「on」、「off」、「ポット」が挙げられる。「on」や「off」はセンサやボタンの状態を表す単語であり、VDM++では、型の値として定義される。「ポット」は「話題

沸騰ポット要求仕様書」のシステム全体を指す単語であり、クラスとして定義される。こうした、システムと等価なクラスである単語や状態を表す単語が高頻度で現れることが、TF-IDFの精度を下げる要因になったと考えられる。

FLRによる並べ替えでは、「ロック」や「サーミスタ」といった単語が下位になった。FLRは複合名詞の要素となっている単語の重要度が高くなるため、単独での出現が多い単語の順位が下がったと考えられる。また、TF-IDFと同様に、システムと等価なクラスである単語や、状態を表す単語が出現頻度によって重要度が高くなっていると言える。

次に、型になる可能性が高い名詞について考察を行う。専門用語のような文書の特徴が現れる単語は型になりやすいという予想は、抽出結果より概ね正しかったと考えられるが、仕様の中で最も特徴的な単語だと言える、モデル化する対象全体に相当する単語は型にならないという事実も見えて取れる。また、本論文で使用した指標は頻度を元に重要度を算出するものであり、仕様書中に一度だけ出てくる「第n水位センサ」のような単語の重要度が上がりづらいという問題もある。「温度」のような多くの文書で現れると考えられる単語が型として定義され、「ランプ」や「ボタン」のように具体的な要素を元に汎用的な単語が定義されたことから、一般的な名詞は型になりにくいという予想は、一概に正しいと言えないことが分かった。

本論文では、単語の専門性を型になる単語の特徴として扱ったが、実験により異なる特徴が見えてきたため、新たな指標を取り入れ、再定義する必要がある。解決案として、型候補の並べ替えに関する変換支援手法には、本論文の並び替えに用いた頻度に加え、上位、下位概念のような単語同士の関係性を表すオントロジーを指標に取り入れることが考えられる。単語同士の関係性を反映したデータとして、日本語WordNet[16]が公開されており、オントロジーを学習するための様々な研究が行われている[17]。単語同士の関係性を指標に取り入れることで、モデル化するシステムの構造が反映され、システム全体のクラスに相当する単語の提案や、具体的な要素をまとめ、汎化した単語を抽出することができれば、より高精度な型の抽出が可能になると考える。

また、「ポット」が抽出されたことから、抽出された名詞の中にクラスになる可能性がある単語が含まれていると言える。オブジェクト指向においてクラスと型の間に明確な違いはないため、システム全体を指す単語以外は、クラスと型どちらでも定義することが可能であり、基本的に作成者が判断する。VDM ToolsのVDM++開発手法ガイドラインでは、『用語集にある名詞は、そのモデルの目

的に“実質的”機能(読み／書きに加えるもの)をまったく持たないのならば、型としてモデル化されるべきである。』(九州大学 2016, p8)[18]とされている。この基準をもとにクラスと型を区別する手法を考察すると、関数・操作となる動詞の主語が機能を持っている名詞だと考えられる。そのため、関数定義の後ならばクラスとしての機能を持つ単語を区別する事ができ、将来的には洗練作業としてのクラス定義の支援が可能になると考える。

5.5. 関数・操作候補並べ替えについての評価・考察

関数・操作候補並べ替え手法について妥当性を評価する。予測データは 3.5 節の実験で目的語の関係をもつ動詞を抽出し、並べ替えを行った関数・操作候補のリストである。正解データは、検証用 VDM++仕様で、関数・操作として定義された単語を、4.3 節で行った動詞抽出のルールに合わせて整形を行い、重複を取り除いたものである。また、人手により作成したデータ中に「トリガ_保温設定ボタン」や「トリガ_タイマボタン」のように操作をモデリングした例がある。「ボタンが押された際に実行される操作」であると予測できるが、動詞を一意に定められないため、正解データから除外した。(表 12)。

表 12 関数・操作の正解データ(一部)

	関数・操作				
1	差し込む	11	点灯	...	...
2	沸騰行為をする	12	点滅	30	解除
3	ロック解除	13	停止	31	検出
4	遷移	14	カルキ抜きをする	32	沸騰
5	切り替える	15	消灯	33	保温
6	ロック	16	中止	34	設定
7	給湯	17	検知	35	閉じる
8	止める	18	確認	36	加算
9	セット	19	判断	37	起動
10	リセット	20	鳴らす	38	終える

4.3 節で抽出され、並べ替えられたデータに関して、再現率は 0.82 となった。抽出できなかった単語は、[切り替える, 始動, タイムアップ, 接続, 切断, 要求, 到達]である。この内、「切り替える」と「切断」は「話題沸騰ポット要求仕様書」中に含まれない語彙であるため、抽出できなかった。しかし、「切断」はコンセントを切断する仕様だが、「抜く」といった候補は抽出できているため、定義することは可能である。「要求」や「到達」といったその他の単語は、「沸騰要求」や「沸点到達」といったイベントの名称となる複合名詞の一部として要求仕様書中に存在し、「要求する」や「到達する」という形で使われていなかったため、抽出できなかったと考えられる。これらの動詞は、形態素解析で最小単位に分割した際に、別個記録することで抽出

は可能であると考ええる。

関数・操作候補の並べ替え手法について妥当性を評価するため、並べ替えを行った関数・操作の候補のデータに関して、PR 曲線をプロットし、AP を算出する。関数・操作の候補の並べ替えを行ったデータの PR 曲線を図 5 に示す。

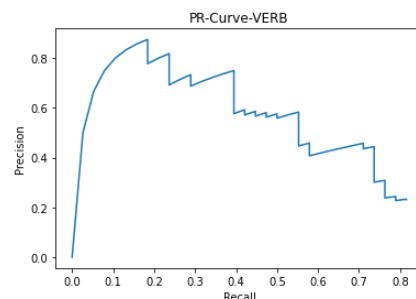


図 5 並べ替えデータの PR 曲線

図 5 の AP は 0.49 である。並べ替えを行わない場合(要求仕様書に出現する順)の AP が 0.25 であるため、並べ替えを行わない場合と比較して精度は向上したと言えるが、一位に不正解データが含まれており、改善の余地を残す結果となった。4.3 節で、すでに定義されている型を目的語とする動詞を抽出することで、操作にあたる動詞を抽出できると予想したが、型を目的語とする動詞の中で、実際に定義されていたのは 38 個中 23 個であったため、この予想は概ね正しかったと言える。

以上の結果をもとに、関数・操作候補並べ替えについて考察を行う。図 6 の通り、最も可能性が高いと判断された動詞の「押す」が正解データでは定義されなかった。これは、「話題沸騰ポット要求仕様書」内でボタンを押すという動作は頻出したが、ボタンを押すのは人間であり、ソフトウェアの動作ではないため、「押す」という語で定義されないと考えられる。そのため、主語が人間を表す単語(ユーザなど)となっている動詞の優先度を下げること、改善が可能であると考ええる。しかし、日本語は主語や目的語が省略されるという特徴があり、実際には省略されている主語を予測するなどの課題があると考ええる。

6. まとめ

本論文では、形式手法導入コスト低減を目指すため、自然言語で記述された要求仕様書から VDM++仕様を作成する作業を、一部自動化により支援するアプローチの提案および、型定義、関数・操作定義に関する支援手法の実装を試みた。さらに、支援手法について評価・考察を行うことで知見の共有を行った。型定義の支援手法

に関して、人間の感覚に近い用語抽出を目指した、複合名詞結合手法の有効性を確認できた。一部抽出できなかった用語は洗練作業の中で作られたものだと考えられる。型として定義する用語を効率的に探索することを目的とした並べ替え手法は、型として定義される単語の特徴を重要度によって精度良く表すことはできず、改善の余地が残る結果となった。これらは、具体的な要素を汎化した単語やシステム全体を表す単語が実際には定義されないため精度が低下したと考えられる。関数・操作定義の支援手法に関しては、型を目的語とする動詞の優先度を上げることで実際に定義される単語を優先的に提示することができた。また、主語が人間である動詞は、実際に定義されづらいという知見を得ることができた。

参考文献

- [1] 独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター, 「形式手法適用調査 調査報告書」, 2010-07-29.
- [2] 佐原 伸, 荒木 啓二郎「オブジェクト指向形式仕様記述言語 VDM++ 支援ツール VDMTools」, コンピュータソフトウェア, 24 巻 2 号, pp.14-20, 2007.
- [3] 「Overture Tool Formal Modelling in VDM」, <https://www.overturetool.org/>.
- [4] 千坂 優佑「VDM++仕様から C#コードを生成するツールの開発」, 情報処理学会全国大会講演論文集 78 巻 1 号, pp.363-364, 2016-03-10.
- [5] Yuma Yamano, Toshihiko Ando, Keishi Okamoto 「A Tool Generating a C# Code with Contracts of Code Contracts from a VDM++ Model with Conditions」, International Journal of Software Engineering (IJSE) Vol.8 No.3, pp.27-39, 2020.
- [6] 独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター, 「厳密な仕様記述における形式手法成功事例」, 2013-01-25.
- [7] 独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター, 「対称を如何にモデル化するか?」, 実務家のための形式手法シリーズ, 2013-03-28.
- [8] 大森洋一, 荒木啓二郎「自然言語による仕様記述の形式モデルへの変換を利用した品質向上に向けて」, 情報処理学会論文誌プログラミング (PRO) 3 巻 5 号, pp.18-28, 2010-12-10.
- [9] Yasuhiro Shigyo, Tetsuro Katayama「Proposal of an Approach to Generate VDM++ Specifications from Natural Language Specification by Machine Learning」, 2020 IEEE 9th Global Conference on Consumer Electronics (GCCE), 2020.
- [10] 菅健将, 片山徹郎「VDM++仕様書におけるクラスとインスタンス変数定義の機械学習を用いた生成手法の提案」, ソフトウェアエンジニアリングシンポジウム 2022 論文集, pp.79-88, 2022-08-29.
- [11] SESSAME「話題沸騰ポット要求仕様書 (GOMA-1015 型) 第7版」, https://www.sesame.jp/workinggroup/WorkingGroup2/POT_Specification_v7.PDF, (最終アクセス 2022-01-11).
- [12] 松田 寛「GiNZA - Universal Dependencies による実用的日本語解析」, 自然言語処理 27 巻 3 号, pp.695-701, 2020-9-15.
- [13] 中川 裕志, 湯本 紘彰, 森 辰則「出現頻度と接続頻度に基づく専門用語抽出」, 自然言語処理 10 巻 1 号, pp.27-45, 2003-01-10.
- [14] Alice Zheng, Amanda Casari「機械学習のための特徴量エンジニアリング —その原理と Python による実践」, 株式会社ホクソエム訳, 2019-2-25.
- [15] Evert Stefan, 「The statistics of word cooccurrences : word pairs and collocations」OPUS - Online Publikationen der Universität Stuttgart, 2005.
- [16] 国立研究開発法人情報通信研究機構 (NICT)「日本語 WordNet」, <https://bond-lab.github.io/wnja/>, (最終アクセス 2023-01-25).
- [17] 森田 武史, 山口 高平「オントロジー学習の現状と動向(<特集>オントロジーの進化と普及(前編))」, 人工知能学会誌 25 巻, pp. 354-365, 2010.
- [18] 九州大学「VDM++開発手法ガイドライン 1.0」, 2016, <http://fmvdm.org/>, (最終アクセス 2023-02-16)

ビルドエラーを引き起こす Dockerfile の依存関係の分析

坂本 廉也

和歌山大学システム工学部

sakamoto.renya@g.wakayama-u.jp

東 裕之輔

日本総合研究所/和歌山大学システム工学研究科

higashi.yunosuke@g.wakayama-u.jp

大平 雅雄

和歌山大学システム工学部

masao@wakayama-u.ac.jp

要旨

近年、企業で急速に普及しつつあるコンテナ型仮想化技術を用いた仮想化プラットフォームとして *Docker* が挙げられる。*Docker* では、コンテナと呼ばれるアプリケーション実行環境を生成する *Docker* イメージを *Dockerfile* からビルドする。しかし、*Dockerfile* はビルドに失敗することが多々あり、その修正には多くの時間が多く費やされる [1]。そこで、*Dockerfile* のビルドエラーに対する自動修正手法である *Shipwright* [2] が提案されている。しかし、*Shipwright* では *Dockerfile* が持つ依存関係までは考慮しておらず、自動修正がおこなえた *Dockerfile* は 18.9% にとどまっている。本研究では、自動修正手法の改善に向けて *Dockerfile* の依存関係を考慮したビルドエラーの分析をおこなう。*RQ1* では、依存関係が原因のビルドエラーがどの程度存在するかを調査した。その結果、ベースイメージレイヤとカスタムイメージレイヤの間に発生する依存関係が半数以上を占めていることが明らかになった。*RQ2* では、依存関係が原因のビルドエラーを修正するのに要する時間を調査した。その結果、レイヤレベルの依存関係によるビルドエラーの修正時間は約 5.8 時間、イメージレベルの依存関係によるビルドエラーの修正時間は約 6.3 時間であることが分かったが、それぞれのビルドエラーの修正時間に統計的な有意差は見られないことが明らかになった。これらの結果から、今後、依存関係が原因のビルドエラーを体系化することで、*Dockerfile* の自動修正手法を大きく改善できる可能性があることがわかった。

1. はじめに

1.1 Dockerfile と Docker イメージ

高度情報化社会において、めまぐるしく移り変わる社会のニーズに対応するために、近年のソフトウェア開発ではプロダクトの迅速なデリバリーが求められている。しかし、さまざまな障壁によって迅速な開発が妨げられることがあり、例えば、障壁となる要素の 1 つとして開発環境の違いが挙げられる。特にオープンソースソフトウェア (OSS) 開発では、チームメンバーが開発環境を各々で用意し、ソフトウェアの実装やテストをおこなう。しかし、チームメンバーごとに使用する OS やライブラリが異なるために、実装やテストが実施できないことがある。前述の問題を解決するべく、コンテナ型仮想化技術を利用したソフトウェア環境として *Docker* が多くの企業で利用されている [3]。*Docker* を用いて開発環境を軽量なイメージファイル (*Docker* イメージ) を作成し、各チームメンバーに配布することで、同様の環境上での開発を可能にする。図 1 にコンテナが作成されるまでの流れを示す。

Docker では、**Docker イメージ (以下、イメージ)** を基にコンテナと呼ばれる仮想的なアプリケーション環境を生成する。イメージとはコンテナを動作させるために必要な特定のアプリケーション環境のスナップショットである。また、イメージは、*Docker Hub*¹ と呼ばれるコンテナレジストリサービスなどで公開することもできる。*Docker Hub* にはあらゆるユーザが作成したイメージが

¹<https://hub.docker.com/>

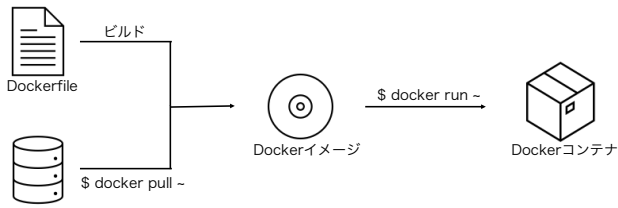


図 1. コンテナ作成までの流れ

```

1 FROM ubuntu:20.04
2 RUN apt-get update && apt-get install python3
3 COPY . /app
4 WORKDIR /app

```

図 2. Dockerfile の例

公開されており、それらのイメージを基に新しいイメージを開発することも可能である。

Docker Hub やローカル環境のイメージを基にして、新しいイメージを開発するには、**Dockerfile** を記述する必要がある。Dockerfile は、イメージに含める内容を指示するための命令が記述されたテキストファイルである。Dockerfile の命令を上から順に実行すること（以下、ビルド）でイメージを作成することができる。イメージは、レイヤ構造となっており、Dockerfile の 1 つの命令に対し 1 つのレイヤが追加される。ビルドにより新しく追加されるレイヤは、以前に積み重ねられたレイヤとの親子関係を持つため、イメージは依存関係によってビルドされるといえる。Dockerfile には、イメージを作成するための様々な命令が用意されている。まず最初は FROM 命令を記述する必要がある²。FROM 命令で既存のイメージ名を指定することで、その既存イメージの内容に変更を加える形で新たなイメージを容易に作成することができる。一方、既存のイメージを流用せずに新たなイメージを作成したい場合は FROM 命令に “scratch” を指定する。本研究では、既存のイメージを流用することをイメージの継承と呼び、継承元のイメージをベースイメージと呼ぶ。図 2 に Dockerfile の例を示す。図 2 では、1 行目で FROM 命令を用いてベースイメージに Ubuntu のバージョン 20.04 のイメージを継承している。イメージは “<イメージ名>:<タグ名>” の形で記載する。イメージ名には利用したい既存のイメージ名を記載し、タグ名はオプションであるが、利用したいバージョンを記

載する [4]。タグ名を記載せずに Dockerfile をビルドした場合、“latest” タグが割り当てられ、最新バージョンのイメージが自動で選択される。

しかしながら、Dockerfile はビルドに失敗することがある（以下、**ビルドエラー**）。Dockerfile は他のプログラミング言語がベースのソフトウェアプロジェクトのビルドよりも失敗しやすいこと [5] や、その修正には多くの時間が費やされること [1] が明らかになっている。そこで、ビルドエラーが発生している Dockerfile の修正を支援するためのツールとして、Shipwright [2] が提案されている。Shipwright では、ビルドエラーが発生している Dockerfile のビルドログを分類し、その分類を基にビルドエラーに対して修正パッチを作成している。ただし、Shipwright は完全にビルドエラーを解決できるわけではない。ビルドエラーが発生した Dockerfile のうち、18.9%しか修正できていないことが報告されてる [2]。

1.2 動機と目的

既存の Dockerfile の自動修正手法が完全にビルドエラーを解決できない理由の一つとして、ビルドエラーの原因を考慮した修正をおこなえていないと考えられる。Dockerfile の依存関係は非常に複雑であり、ビルドエラーの根本原因の特定が困難であると想定される。そのため、Dockerfile の自動修正手法を大きく改善するには、まず、ビルドエラーの原因を考慮した自動修正手法の性能向上に向けた分析が必要となる。しかし、Dockerfile の依存関係がビルドエラーに与える影響は未だ明らかになっていない。そこで、本研究では、依存関係が原因で発生しているビルドエラーがどの程度存在するのかを明らかにし、そういったビルドエラーがどの程度の修正時間で修正されているのかを明らかにする。

2. イメージのレイヤ構造とビルドエラー

2.1 イメージのレイヤ構造

イメージを構成するレイヤはベースイメージレイヤとカスタムイメージレイヤの 2 つに大別できる。図 3 にそれぞれの関係を示す。

ベースイメージレイヤ：ベースイメージで実行された命令のみで構成される読み込み専用のレイヤである。ベースイメージは、ある Dockerfile から既にビルドされたイメージであり、それを開発者が自由に選択し Dockerfile

²<https://docs.docker.jp/engine/reference/builder.html#from>

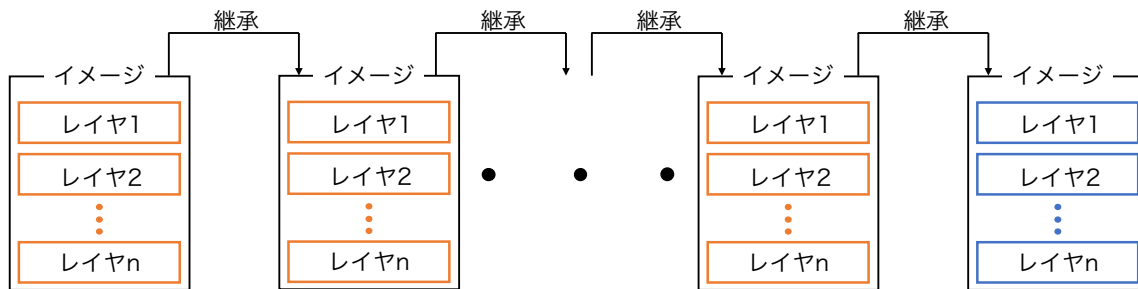


図 3. ベースイメージレイヤとカスタムイメージレイヤ

内で継承をおこなう。図 3 のように、ある開発者が記述した Dockerfile で指定されたベースイメージがさらに別のイメージを継承している、というようなネスト構造になることがある。この場合、継承元のイメージは全てベースイメージレイヤになる。

カスタムイメージレイヤ：新たに Dockerfile 内で記述された命令に対応するレイヤである。ベースイメージレイヤの上にカスタムイメージレイヤが構築され、ベースイメージレイヤとカスタムイメージレイヤには親子関係が存在することになり、ベースイメージの機能を継承する。開発者が変更できる範囲は自身で記述した Dockerfile 内の命令のみであり、開発者はベースイメージレイヤを構成する命令との互換性を意識して新たな命令を記述する必要がある。

2.2 Dockerfile のビルドエラーと依存関係

Dockerfile のビルドは他の言語がベースのソフトウェアよりも失敗しやすいことが明らかになっている [5]。そこで、ビルドエラーが発生している Dockerfile の修正を支援するためのツールとして Shipwright が提案されている。Shipwright は、ビルドエラーが発生している Dockerfile のビルドログを分類し、その分類を基にビルドエラーに対して修正をおこなう。しかしながら、Shipwright ではビルドエラーが発生した Dockerfile のうち、18.9%の Dockerfile しか修正できていない。この原因として、Shipwright はビルドエラーの原因を特定した修正をおこなっていないことが考えられる。ベースイメージレイヤの書き換えは不可能であるため、ビルドエラーの根本原因がベースイメージ内に存在する場合、ビルドエラーを直接的に解消するのではなく、何らかの回避策によってビルドエラーを解消する必要がある。つまり、Shipwright が正しくビルドエラーを修正可能になる

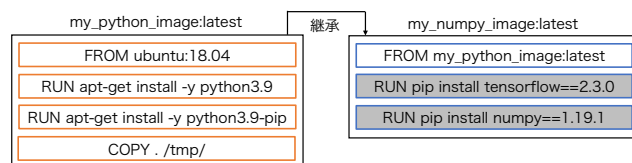


図 4. レイヤレベルの依存関係でのビルドエラー

ためには、まず、そのエラーが Dockerfile の直接的な修正で対応が可能なのか、FROM 命令で指定している既存イメージを考慮した変更が必要なのかに分類することが必要であると言える。

Dockerfile が持つ依存関係はカスタムイメージレイヤ間で発生する依存関係と、ベースイメージレイヤとカスタムイメージレイヤ間で発生する依存関係に大別できる。

レイヤレベルの依存関係：カスタムイメージレイヤ内に存在するレイヤ同士の依存関係である。つまり、開発者が記述した命令間で発生する依存関係である。レイヤレベルの依存関係が原因で発生するビルドエラーの例を図 4 に示す。図 4 では、“my_python_image” という名前の “latest” タグを持ったイメージをベースイメージとすることで “my_numpy_image” イメージを作成している。しかし、灰色で示す tensorflow のバージョン 2.3.0 と numpy のバージョン 1.9.1 との間に互換性がないため、ビルドエラーが発生する。

イメージレベルの依存関係：ベースイメージレイヤ内のあるレイヤとカスタムイメージレイヤのあるレイヤの間で発生する依存関係である。イメージレベルの依存関係が原因で発生するビルドエラーの例を図 5 に示す。図 5 の場合、ベースイメージレイヤでは python3.9 をインストールする命令が記述されており、カスタムイメージレイヤでは numpy1.14 をインストールする命令が記述されている。しかし、python3.9 と numpy1.14 には互

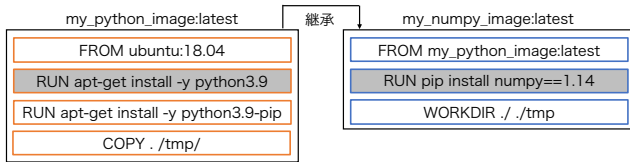


図 5. イメージレベルの依存関係でのビルドエラー

換性がないため、ビルドエラーが発生する。また、図 3 のように継承がネスト構造になっている場合があり、開発者が指定したベースイメージがさらに継承しているイメージの各レイヤとの依存関係も存在する。

Docker Hub 上のイメージは、Dockerfile やドキュメントと共に管理することができるが、Dockerfile と共に管理されているイメージは多くない [6]。Docker Hub からベースイメージのみを取得した場合、開発者はベースイメージの具体的なファイル構成、バージョン情報を意識できない可能性がある。結果的に、カスタムイメージレイヤ内の命令とベースイメージレイヤ内の命令との互換性を意識した修正が困難になると考えられる。そこで本研究では、ビルドエラーの自動修正手法の性能向上に向けて、2.2 節で定義した 2 つの依存関係に着目し、それぞれの依存関係がビルドエラーに与える影響を調査する。

3. 分析

本研究では、Dockerfile の依存関係がビルドエラーに与える影響を明らかにするため、以下の 2 つのリサーチクエスチョンに取り組む。

RQ1: レイヤレベルの依存関係とイメージレベルの依存関係が原因のビルドエラーはどの程度存在するか

RQ2: レイヤレベルの依存関係とイメージレベルの依存関係が原因のビルドエラーはどの程度の時間で修正されているか

3.1 データセット

本研究では Shipwright で使用されたデータセットを使用する。Shipwright では、binnacle データセット [7] を拡張したものを使用しており、ルートディレクトリに単一の Dockerfile が存在する 20,526 件の GitHub リポジトリを取得している。また Shipwright では、すべての Dockerfile に対してビルドをおこなっており、データ

表 1. 本論文のデータセット

Dockerfile 数	ビルドエラー件数	ビルドエラー率
16,099 件	4,440 件	27.5%

セット内の Dockerfile に対応するビルドログも提供されている。本研究で扱うデータセットは、20,526 件のリポジトリから、以下の条件に合うデータを除外する。

1. 破損しているデータを持つリポジトリ
2. 現在は存在していないリポジトリ
3. マルチステージビルドを利用している Dockerfile を持つリポジトリ

Docker では 2017 年 5 月以降、マルチステージビルドと呼ばれる機能がサポートされている。マルチステージビルドとは、イメージサイズを小さくするために、ビルド途中で生成される不要な生成物を取り除くことを効率的に実現する機能である。マルチステージビルドでは複数の FROM 命令が記述され、各 FROM 命令でビルドするステージが異なる。以前に実行されたステージの実行結果から、イメージに必要な生成物のみを次のステージに引き継ぐことが可能である。そのため依存関係がベースイメージレイヤ間で発生する可能性があり、1 章で定義した Dockerfile が持つ依存関係とは別の依存関係が発生している可能性があるため、今後更なる調査が必要であるが、本研究では分析対象外とする。

これらの Dockerfile をフィルタリングした結果、16,099 件のリポジトリを本研究のデータセット（表 1）として扱う。16,099 件のうち、4,440 件の Dockerfile でビルドエラーが発生しており、本データセットにおける Dockerfile のビルドエラー発生率は 27.5% である。

3.2 RQ1: レイヤレベルの依存関係とイメージレベルの依存関係が原因のビルドエラーはどの程度存在するか

動機: Shipwright ではビルドエラーが発生している Dockerfile のうち、18.9% しか自動修正できていない。ビルドログにはカスタムイメージレイヤに対応する命令の実行結果しか記録されていないため、ベースイメージレイヤとカスタムイメージレイヤの間の依存関係により発生したビルドエラーは見つけることができない。Shipwright

```

Sending build context to Docker daemon 274.9MB

Step 1/57 : ARG UBUNTU_VERSION
Step 2/57 : ARG CUDA_VERSION
Step 3/57 : ARG CUDNN_VERSION
Step 4/57 : FROM nvidia/cuda:${CUDA_VERSION}-cudnn${CUDNN_VERSION}-devel-ubuntu${UBUNTU_VERSION}

```

図 6. レイヤレベルの依存関係が原因で発生しているビルドエラーのビルドログ

```

Sending build context to Docker daemon 34.3kB

Step 1/9 : FROM python:3-alpine
3-alpine: Pulling from library/python
df20fa9351a1: Already exists
36b3adc4ff6f: Already exists
7031d6d6c7f1: Already exists
81b7f5a7444b: Already exists
0f8a54c5d7c7: Already exists
Digest: sha256:c5623df482648cacece4f9652a0ae84b51576c93773ccd43ad459e2a195906dd
Status: Downloaded newer image for python:3-alpine
----> 8ecf5a48c789
Step 2/9 : RUN apt-get update && apt-get upgrade -yqf && apt-get install -yqq build-essential
----> Running in 5f58dd870d65
/bin/sh: apt-get: not found

```

図 7. イメージレベルの依存関係が原因で発生しているビルドエラーのビルドログ

の手法でビルドエラーの修正率が低いのは、イメージレベルの依存関係がビルドエラーの原因の多くを占めている可能性がある。そのため、まずビルドエラーの原因うち、本研究で定義した 2 種類の依存関係がどの程度占めているのかを明らかにし、依存関係を修正することが自動修正手法の性能向上につながるのかを分析する必要がある。

分析方法：ビルドエラーが発生している 4,440 件のビルドログを目視することにより、ビルドエラーの原因をレイヤレベルの依存関係、イメージレベルの依存関係、その他の 3 つに分類する。例として、レイヤレベルの依存関係が原因で発生しているビルドエラーのビルドログを図 6 に示す。図 6 では、ARG 命令を用いて環境変数を FROM 命令に渡そうとしているが、環境変数の受け渡しがうまくおこなわれていないため、発生するエラーである。このように、ビルドエラーが発生した命令行自体にエラーの根本原因がなく、ビルドログ内にその原因を特定できる命令が存在する場合、レイヤレベルの依存関係として扱う。

イメージレベルの依存関係が原因で発生しているビルドエラーのビルドログの例を図 7 に示す。図 7 では、ベースイメージの OS に Alpine Linux が使用されておりパッケージマネージャは APK が採用されているにもかかわらず、Dockerfile 内で debian 系の Linux ディストリビューションで採用されているパッケージマネージャの APT の apt-get コマンドを用いてパッケージをインス

表 2. 依存関係が原因でビルドエラーが発生した Dockerfile の数

ビルドエラーの原因	Dockerfile 数	割合
レイヤレベルの依存関係	621	14.0%
イメージレベルの依存関係	2,546	57.3%
その他	1,273	28.7%
合計	4,440	100.0%

トールしようとしている。この場合、ベース OS に採用されている OS の種類を確認せずに、パッケージインストールコマンドを記述したことにより、ビルドエラーが発生したと考えられる。このように、Dockerfile 内の命令でベースイメージの機能に関する操作をおこなっており、その行でビルドエラーが発生している場合、イメージレベルの依存関係として扱う。

結果：4,440 件のビルドログを目視した結果を表 2 に示す。表 2 より、レイヤレベルの依存関係が原因で発生しているビルドエラーが 621 件 (14.0%)、イメージレベルの依存関係が原因で発生しているビルドエラーが 2,546 件 (57.3%)、その他が原因で発生しているビルドエラーが 1,273 件 (28.7%) 存在することがわかった。その他が原因で発生しているビルドエラーとは、命令の記述の誤りによるものや、変更が必要な外部ファイルによるものなどである。命令の記述の誤りは、引数やオプションの指定が記述規則に則っていないことがビルドエラーの原因になる。変更が必要な外部ファイルとは、シェルスクリプトや Makefile などの外部ファイルを指し、外部ファイルの誤りがビルドエラーの原因となる。本研究では Dockerfile 以外のファイルを修正することを想定していないため、外部ファイルの誤りが原因となるビルドエラーは本研究の対象外とし、“その他”へ分類している。

RQ1 の結果から、Dockerfile のビルドエラーは、イメージレベルの依存関係が原因で発生するビルドエラーが原因の半数以上を占めていることがわかった。つまり、ベースイメージレイヤとカスタムイメージレイヤの命令間に互換性がないことがビルドエラーの主な原因となっている。一方で、レイヤレベルの依存関係が原因、つまり、カスタムイメージレイヤ内のある 2 つの依存関係によって引き起こされるビルドエラーは少数である。これらの結果より、ベースイメージレイヤがビルドエラーにもたらす影響が大きいことが明らかになった。

図 8. Dockerfile の修正時間の取得方法の例

3.3 RQ2：2つの依存関係はどの程度の時間で修正されているか

動機：RQ2 では修正が必要な Dockerfile を特定するために分析をおこなう。Dockerfile の修正には多くの時間が費やされることが明らかになっている [1] が、その中でも特に原因特定や修正に時間のかかる種類のビルドエラーに対して自動修正をおこなうことで、ソフトウェアの迅速なデリバリーを実現するための支援がおこなえると考える。特に、Dockerfile が持つ 2 つの依存関係のうちイメージレベルの依存関係は、ビルドエラーの根本原因が Dockerfile 内に記述されていないため、原因の特定により多くの時間がかかる可能性がある。

分析方法：RQ2 ではデータセットに含まれる各リポジトリのメタ情報を用いて、GitHub API³で各リポジトリの Dockerfile に対するコミットを全て取得する。Shipwright のデータ収集の直前におこなわれた Dockerfile へのリビジョンを特定し、その 1 つ後の Dockerfile へのリビジョンとの時間差を Dockerfile の修正時間として計算する。図 8 に RQ2 での Dockerfile の修正時間の取得方法を示す。図 8 では、Dockerfile が ver.2 のときにデータ取得をおこなっている。このとき ver.2 の Dockerfile の修正時間は、rev.2 のコミットがおこなわれた時間と rev.3 のコミットがおこなわれた時間の時間の差とする。

一般的にビルドエラーが発生している Dockerfile がリポジトリにある場合、その Dockerfile への変更内容はビルドエラーの修正である可能性が非常に高い。そのため、データセットの Dockerfile に対するコミットと、その直後のコミットを Dockerfile の修正時間として計算する。また、Google の研究 [8] で提案されているように、結果の公平性を保つためにビルドエラーの修正に 12 時間以上の時間を要しているビルドエラーを除外する。その後、修正時間の四分位範囲の 1.5 倍を上下限を外れ値として除去する。事前調査の分析結果を基にレイヤレベルの依存関係とイメージレベルの依存関係によるビルドエラー

図 9. 依存関係別のビルドエラーの修正時間の分布

の修正時間の差を比較する。なお、RQ2 では、データセット内の Dockerfile が収集された時点以降変更されていない Dockerfile については、修正時間を計算することができないため、RQ2 では分析対象外としている。各ビルドエラーの修正時間を算出した後、レイヤレベルの依存関係が原因のビルドエラーとイメージレベルの依存関係が原因のビルドエラーの間に統計的な有意差があるのかについて統計的検定をおこなう。

結果：依存関係別のビルドエラーの修正時間の分布を表す箱ひげ図を図 9 に示す。また、箱ひげ図に示した各依存関係が原因で発生しているビルドエラーでの標本数、および修正時間の平均値と中央値を表 3 に示す。

表 3 より、レイヤレベルの依存関係によるビルドエラーの修正時間の平均値は 6.9 時間、中央値は 5.8 時間であった。一方、イメージレベルの依存関係によるビルドエラーの修正時間の平均値は 7.5 時間、中央値は 6.3 時間であった。つまり、レイヤレベルの依存関係が原因のビルドエラーの修正時間に比べ、イメージレベルの依存関係が原因のビルドエラーの修正時間は約 1 時間長いことがわかる。また、平均値についても、イメージレベルの依存関係が原因で発生しているビルドエラーの修正時間の方が約 0.6 時間長いことがわかる。

マンホイットニーの U 検定を用いて 2 つの依存関係が原因のビルドエラーの修正時間に統計的有意差（有意水準 1%）があるかを調査した結果、p 値が 0.430 となり、2 つの依存関係およびその他が原因のビルドエラーの修正時間に有意差はないことが明らかになった。これらの結果より、依存関係の種類により修正時間に有意な違いはないが、RQ1 の結果を踏まえるとイメージレベ

³<https://docs.apitester.org/guides/github-graphql-api-guide>

表 3. 依存関係が原因で発生するビルドエラーの修正時間

ビルドエラーの原因	Dockerfile 数	修正時間の平均値 (時)	修正時間の中央値 (時)
レイヤレベルの依存関係	88	6.9	5.8
イメージレベルの依存関係	297	7.5	6.3
その他	129	6.3	5.7

ルの依存関係によるビルドエラーに重点を置く方が自動修正率の向上に寄与することがわかった。

4. 考察

4.1 継承の深さとビルドエラーの関係

RQ1 では、イメージレベルの依存関係が、ビルドエラーの原因の半数以上を占めていることが明らかになった。ただし、Dockerfile 内で指定したベースイメージに対応する Dockerfile がさらに別のベースイメージを指定している場合がある。その場合、Dockerfile の依存関係が複雑になり、ビルドエラーが発生しやすくなる可能性がある。本節では、Dockerfile 内で指定されたベースイメージの継承の深さがビルドエラーに与える影響について追加調査をおこなう。Dockerfile の継承の深さとは、Dockerfile が継承しているベースイメージの数を意味する。例えば、ベースイメージの FROM 命令にコンテナ内の OS 部分にあたる “ubuntu:latest” が指定されている場合を考える。このとき、“ubuntu:latest” イメージを作成するための Dockerfile の FROM 命令には “scratch” を指定しているため、“ubuntu:latest” のさらなる継承は存在しない。そのため、“ubuntu:latest” を FROM 命令で指定しているイメージの継承の深さは 1 になる。

継承の深さ別のビルドエラー件数とその割合を表 4 と表 5 に示す。表 4 および表 5 より、継承が深くなるほどビルドエラーの発生率が高くなるという関係は存在しないことがわかる。特にイメージレベルの依存関係が原因でビルドエラーが発生する件数は継承の深さが 1 の場合が最も多い (表 5) ことから、継承の深さよりも継承をおこなうこと自体がビルドエラーを誘発する原因となっていることがわかる。これは継承を活用したイメージ作成を推奨するベストプラクティス⁴とは相反する事象であると思われるが、少なくともベースイメージの選択には細心の注意を払う必要があることを示唆している。

⁴<https://docs.docker.com/develop/develop-images/dockerfile-best-practices/>

4.2 “latest” タグがビルドエラーに与える影響

前節で、継承の深さはビルドエラーに影響はなく、継承をおこなうこと自体がビルドエラーを引き起こしやすくなる可能性があることがわかった。ただし、継承をおこなう際には、イメージの選択の他に、タグも一緒に選択する必要がある。タグを選択する場合に、“latest” タグもしくはタグ名記載なしで継承をおこなう場合、ビルドするたびに継承するイメージのバージョンが最新化されることで、ビルドエラーが発生する可能性がある。そのため、本節では、イメージレベルの依存関係において “latest” タグがビルドエラーに与える影響が大きいのかについて追加調査をおこなった。“latest” タグを使用している Dockerfile のビルドエラー率は、2 つの依存関係によって異なるのかを調査した。

表 6 に “latest” タグを使用しておりビルドエラーが発生している Dockerfile の件数を、依存関係別に示す。また、表 6 に示した結果に統計的有意差が見られるかを確認する。そのために、本研究で定義した 2 つの依存関係において、“latest” タグでベースイメージを継承している Dockerfile のビルドエラーの発生割合が有意に違いがあるのか、有意水準を 0.01 としてカイ二乗検定をおこなった。その結果、p 値が 0.003 となり、つまり、イメージを “latest” タグで継承することはイメージレベルの依存関係を引き起こしやすいと言える。

4.3 自動修正手法の構築における修正対象

RQ2 では 2 種類の依存関係及びその他が原因で発生するビルドエラーの修正時間において、それぞれの修正時間の間に統計的有意差は見られないことがわかった。しかしながらデータセットに含まれる Dockerfile に、Shipwright がデータ収集をおこなった 2020 年 6 月以降、変更 (修正) されていない Dockerfile が多く存在していた。表 7 にデータセット収集以降 Dockerfile を修正していないリポジトリの件数を示す。表 7 より、イメージレベルの依存関係が原因のビルドエラーは、レイヤレベル

表 4. レイヤレベルの依存関係が原因でビルドエラーが発生した Dockerfile の割合（継承の深さ別）

継承の深さ	Dockerfile 数	ビルドエラーが発生した Dockerfile の数	割合
1	3,349	231	6.8%
2	1,517	34	2.2%
3	158	5	3.1%
4	57	4	7.0%
5	309	3	0.9%
6	8	0	0.0%

表 5. イメージレベルの依存関係が原因でビルドエラーが発生した Dockerfile の割合（継承の深さ別）

継承の深さ	Dockerfile 数	ビルドエラーが発生した Dockerfile の数	割合
1	3,349	631	18.8%
2	1,517	236	15.5%
3	158	33	20.8%
4	57	11	19.2%
5	309	45	14.5%
6	8	0	0.0%

の依存関係が原因で発生しているビルドエラーと比べて半数以上が修正されていない。これは、ベースイメージレイヤにまたがって存在する、破損した依存関係の修正は困難であることが原因である可能性が考えられる。そのため、イメージレベルの依存関係を修正する自動修正手法を構築することが期待される。

4.4 妥当性への脅威

4.4.1 内的妥当性

本研究では先行研究で作成されたデータセットを用いている。データセットに含まれる Dockerfile やイメージは先行研究が実施された当時のものであり、現在、Dockerfile をビルドした場合の結果とは異なる可能性がある。

また、本研究では第 1 著者が目視によってビルドエラーの分類をおこなったため、分類結果は著者の判断に依存する。ただし、3.2 節に記載した基準にしたがって分類したため、第三者が分類をおこなったとしても、結論に影響を与えるほどの個人差は発生しないと考えている。

4.5 外的妥当性

本研究では、Shipwright で使用された 20,526 件の GitHub リポジトリからフィルタリングをおこない、

16,099 件の GitHub リポジトリとそれに対応する Dockerfile を対象に分析をおこなっている。本論文でおこなったフィルタリングによって、データの性質に偏りが発生している可能性がある。また、コンテナレジストリは DockerHub 以外にも、Amazon Elastic Container Registry(Amazon ECR)⁵や Azure Container Registry⁶, Google Container Registry⁷などが存在する。Docker で使用されるパブリックコンテナレジストリのデファクトスタンダードは Docker Hub であり、最も多く利用されているコンテナレジストリサービスの 1 つである [9]。

5. 関連研究

5.1 Dockerfile の品質問題

近年、Dockerfile の品質に関する研究が多くされているが、特にソフトウェア保守の観点から、品質に影響を与えるようなソースコードや実装に関する研究が盛んにおこなわれている。Dockerfile の品質に影響を与えるようなソースコードや実装は将来的にビルドエラーを引き起こす可能性があり、これらを事前に検出し修正するこ

⁵<https://aws.amazon.com/jp/ecr/>

⁶<https://azure.microsoft.com/ja-jp/products/container-registry/>

⁷<https://cloud.google.com/container-registry>

表 6. ビルドエラーが発生した “latest” タグを使用している Dockerfile 数

依存関係の種類	Dockerfile 数	ビルドエラーが発生した Dockerfile	割合
レイヤレベル	621	43	6.90%
イメージレベル	2,546	275	10.80%

表 7. Dockerfile が修正されていないリポジトリの件数

依存関係の種類	Dockerfile 数	修正されていない Dockerfile	データセット全体に対する割合
レイヤレベル	621 件	461 件	22.4%
イメージレベル	2,546 件	2,053 件	46.2%

とでビルドエラーを未然に防ぐことができる。

Wu ら [10] は, GitHub 上に存在する Dockerfile のうち約 84% に非推奨とされる命令記述 (コードスメル) が存在することを示した。また, プロジェクトが作成された年数やプロジェクトで使用するプログラミング言語によっても, コードスメルの数や種類に違いがあることを発見した。一方, Eng ら [11] は, 7 年間にわたる Dockerfile の変更履歴から Dockerfile の記述方法の変化についての知見を提供している。近年では, コードスメルを持つ Dockerfile の数が減少しており, 開発者は Docker 社が提供する Dockerfile 記述のためのベストプラクティスに従う傾向が強い可能性があることを示した。Azuma ら [12] は, 開発者がコメント文でアドホックに記載したコードの改善策 (Self-Admitted Technical Debt: SATD) が Dockerfile 内にどの程度存在しているかを調査した。その結果, Dockerfile に含まれるコメントのうち, 約 3.0% が SATD の存在を示唆していることが明らかになった。また, Dockerfile 中の SATD は 5 つのクラスと 11 のサブクラスに分類可能であり, Dockerfile 特有の SATD が存在することも明らかになった。

Dockerfile においてビルドエラーが発生した際には, コードスメルや SATD の種類からビルドエラーの原因を理解することで, Dockerfile の自動修正手法の性能向上への手掛かりにできる可能性がある。

5.2 ビルドエラーの自動修正

Hassan ら [13] は, ビルドスクリプトを自動修正するための手法として HireBuild を提案している。HireBuild では, 既存のビルドスクリプトの修正から自動生成した修正パターンとビルドログの類似性に基づいて, ビルドエラーが発生しているビルドスクリプトに修正パッチを

自動生成している。その結果, 175 件のビルドエラーから, 135 件の修正パターンを抽出した。この修正パターンは, 再現性のあるビルドエラーのうち, 45% を手動修正と同程度の時間で自動修正することができた。

Dockerfile の自動修正に関する研究は, 2.2 節で紹介した Shipwright のみである。Shipwright では, Dockerfile のビルドエラーが発生した際に得られるエラーメッセージをルールベースで分類し, その分類を基に修正パッチを手動で生成している。それにより, 13 個の修正パターンと 50 個の修正の提案を作成している。その結果, 自動で修正をおこなうことができた Dockerfile は 18.9% である。また, ビルドエラーが発生している Dockerfile を持つプロジェクトに対して, 修正パッチの提案をおこなった結果, 45 件中 19 件の修正パッチが受け入れられた。

本論文では, Dockerfile が持つ依存関係を考慮した Dockerfile の自動修正手法の構築へ向けた事前分析をおこなった。依存関係を考慮した自動修正手法と, [13] や [2] を組み合わせることで, 自動修正手法の性能向上が期待できる。

6. おわりに

本論文では, Dockerfile の自動修正手法の性能向上に向けたビルドエラーを引き起こす Dockerfile の依存関係について分析をおこなった。RQ1 では, イメージレベルの依存関係が Dockerfile のビルドエラーの大半を占めていることが明らかになった。RQ2 では, 依存関係が原因のビルドエラーを修正するのに要する時間を調査した。その結果, レイヤレベルの依存関係によるビルドエラーの修正時間は約 5.8 時間, イメージレベルの依存関係によるビルドエラーの修正時間は約 6.3 時間であることが分かったが, それぞれのビルドエラーの修正時間に統計

的な有意差は見られなかった。一方、ビルドエラーが発生している Dockerfile のうち修正がおこなわれていない Dockerfile が多く存在していた。

RQ1 と RQ2 の結果を合わせると、Dockerfile のビルドエラーは、ベースイメージレイヤと Dockerfile 内の命令の互換性によるものが半数以上を占め、そのようなビルドエラーは修正が困難である可能性があることが明らかになった。今後は、修正が困難な Dockerfile のビルドエラーが多数存在する可能性があり、今後依存関係ごとのビルドエラーの内容分析を通じてビルドエラーを体系化し、自動修正手法を改善する予定である。

謝辞

本研究の一部は、文部科学省科学研究補助金（基盤（C）：22K11974）による助成を受けた。

参考文献

- [1] Yiwen Wu, Yang Zhang, Tao Wang, and Huaimin Wang. An Empirical Study of Build Failures in the Docker Context. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR '20)*, pp. 76–80, 2020.
- [2] Jordan Henkel, Denini Silva, Leopoldo Teixeira, Marcelo d'Amorim, and Thomas Reps. Shipwright: A human-in-the-loop system for dockerfile repair. In *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE '21)*, pp. 198–199, 2021.
- [3] Michael Ferranti. 2017 annual container adoption survey: Huge growth in containers. <https://portworx.com/blog/2017-container-adoption-survey/>. Accessed on February 22, 2023.
- [4] Microsoft. コンテナ イメージのタグ付けとバージョン管理に関する推奨事項. <https://learn.microsoft.com/ja-jp/azure/container-registry/container-registry-image-tag-version>. Accessed on March 6, 2023.
- [5] Jurgen Cito, Gerald Schermann, John Erik Wittern, Philipp Leitner, Sali Zumberi, and Harald C. Gall. An Empirical Analysis of the Docker Container Ecosystem on GitHub. In *Proceedings of the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR '17)*, pp. 323–333, 2017.
- [6] Hasan Ibrahim, Mohammed Sayagh, and Ahmed E.Hassan. Too many images on dockerhub! how different are images for the same system? *Empirical Software Engineering*, Vol. 25, No. 5, pp. 4250–4281, 2020.
- [7] Jordan Henkel, Christian Bird, Shuvendu K. Lahiri, and Thomas Reps. A Dataset of Dockerfiles. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR '20)*, pp. 528–532, 2020.
- [8] Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, and Robert Bowdidge. Programmers' build errors: A case study (at google). In *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*, pp. 724–734, 2014.
- [9] Ahmed Zerouali, Tom Mens, Gregorio Robles, and Jesus M Gonzalez-Barahona. On the relation between outdated docker containers, severity vulnerabilities, and bugs. In *Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER' 19)*, pp. 491–501, 2019.
- [10] Yiwen Wu, Yang Zhang, Tao Wang, and Huaimin Wang. Characterizing the Occurrence of Dockerfile Smells in Open-Source Software: An Empirical Study. *IEEE Access*, Vol. 8, pp. 34127–34139, 2020.
- [11] Kalvin Eng and Abram Hindle. Revisiting dockerfiles in open source software over time. In *Proceedings of the 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR '21)*, pp. 449–459, 2021.
- [12] Hideaki Azuma, Shinsuke Matsumoto, Yasutaka Kamei, and Shinji Kusumoto. An empirical study on self-admitted technical debt in Dockerfiles. *Empirical Software Engineering*, Vol. 27, No. 2, 2022.
- [13] Foyzul Hassan and Xiaoyin Wang. HireBuild: An automatic approach to history-driven repair of build scripts. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*, pp. 1078–1089, 2018.
- [14] Inc. Docker. Docker official images. https://docs.docker.com/docker-hub/official_images/. Accessed on February 22, 2023.
- [15] Yinyuan Zhang, Yang Zhang, Xinjun Mao, Yiwen Wu, Bo Lin, and Shangwen Wang. Recommending Base Image for Docker Containers based on Deep Configuration Comprehension. In *Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER '22)*, pp. 449–453, 2022.
- [16] Chris Tomy, Tingmao Wang, Earl T. Barr, and Sergey Mechtaev. Modus: A Datalog dialect for building container images. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESE/FSE '22)*, pp. 595–606, 2022.

IoT システムにおけるワークフローモデルを用いた AI 推論処理の割当て最適化

伊藤 弘毅
三菱電機株式会社
Ito.Hiroki@dr.MitsubishiElectric.co.jp

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

栗田 太郎
ソニー株式会社
taro.kurita@sony.com

石川 冬樹
国立情報学研究所
f-ishikawa@nii.ac.jp

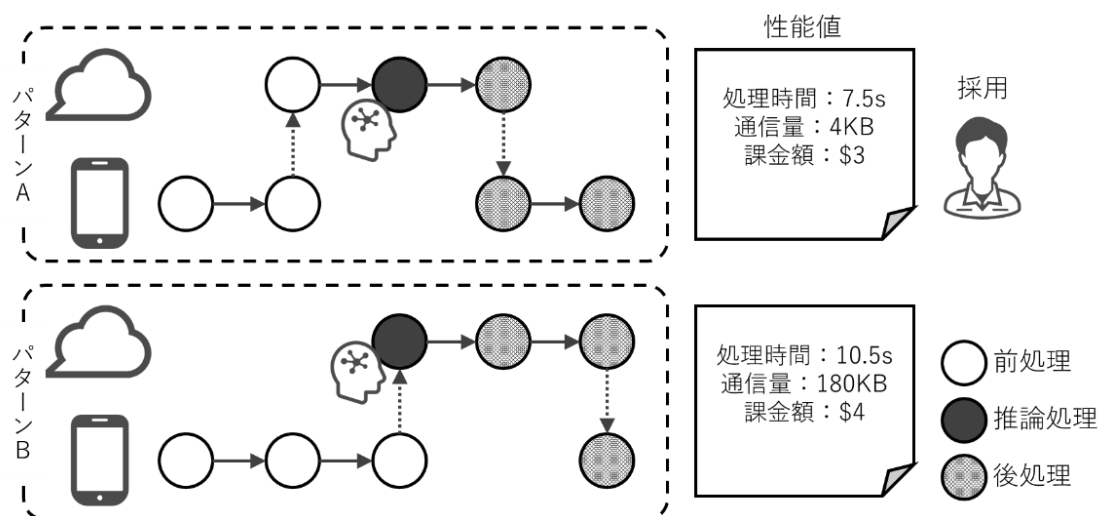
要旨

エッジデバイスの計算リソースの増大に伴い、AI の推論処理をクラウド上ではなくエッジ側で実行するエッジ AI を実現できるようになり、AI を IoT システムに搭載する際の設計の選択肢が広がっている。AI の推論処理を適切にクラウド側とエッジ側に割当てない場合、処理性能が想定を下回ったり、高額なクラウド課金が発生するなど、要求品質を満たさない可能性がある。決定した処理割当て方法が要求品質を満たすか検証する必要があるが、実機を用いて検証する場合、検証用ソフトウェアの開発やシステム的环境構築に工数を要する。本論文では、AI の推論処理および付随する前処理と後処理の実行時間

と通信量を表現するワークフローモデルを構築することにより、処理割当てのパターンごとに全体の処理時間や課金額を推測し、最適なパターンを開発者に提示する手法を提案する。また、推測結果と実機の実測結果を比較することにより、提案手法の精度と制約について論ずる。

1. はじめに

クラウドとエッジデバイスを連携させた IoT システムを構築し、システムから収集したデータを活用したソリューションを創出することにより、顧客に新たな価値を提供する試みが継続して行われている。IoT の市場規模は 2021 年に前年比 22.4%増加しており、2027 年まで継続して拡大する予測[1]もあり、将来的にも企業のビジネスにとって



重要な要素となると思われる。IoT システムと AI は親和性が高く、エッジデバイスから収集したデータを AI に学習させることで、ユーザに価値を提供するソリューションを効率的に提供することが可能になる。

AI を搭載した機能を IoT システムにデプロイする場合、比較的計算リソースの潤沢なクラウド上で推論処理を実行し、実行結果をエッジデバイスに返す構成としていたことが多い。しかし、AI の推論処理をクラウドで実行する場合、エッジで収集したデータをクラウドに転送する必要があるため、通信帯域を多く消費するほか、クラウドのサーバに対する負荷も増大する[2]。そのため、近年ではリアルタイム性能の確保やセキュリティの確保、クラウド利用による課金額の削減を目的として、エッジデバイス上で推論処理を実行するエッジ AI の構成を採用する場合も出てきている。

上記背景から、AI をシステムに搭載する時に、開発者は推論処理と付随する処理をクラウドとエッジでどの処理を実行するか決定する必要がある。その際、搭載機能に対する要件を考慮して処理の割当てを決定する必要がある。開発者が複数ある候補の中から最適なものを選択する難易度は高い。

そこで、本論文ではワークフローモデルと呼ばれる AI 推論フローを抽象化したモデルを構築することにより、最適な処理割当て方法を導出する方法を提案する。

2. 課題

IoT システムに AI の推論フローを実装する場合、フローを構成する処理をクラウドまたはエッジのどちらに割当てて。図 1 に、IoT システムに AI の推論フローをデプロイする時の、処理の割当て例を示す。

AI の推論フローは、学習モデルへ入力するデータを加工するための前処理と、推論結果をアプリケーションが利用する形式に加工する後処理を含んでおり、これらの処理が逐次的に実行されることにより全体の機能が実現される。推論フローをクラウドとエッジで連携して実現する場合、処理の間でデータを送受信しながら処理を実行する。

AI の推論処理を含めた機能を搭載する場合、要求品質を考慮して当該機能が実行する処理をエッジ側で実行するかクラウド側で実行するかを判断し実装する必要がある。処理割当ての候補は、図 1 に示すとおり複数考えられるが、各処理の計算量や処理間の通信量の違い

から、候補によって処理時間やリソース使用量が変化する。そのため、誤った候補を選択すると、エッジークラウド間の通信量が多くなったり、クラウドの処理により過大な課金額が発生したりする可能性がある。

実際の性能値や課金額は、実機を用いて機能を搭載したシステムを稼働して検証する必要があるが、機能実装完了後のシステム検証時において問題が明らかになると、設計をやり直し再実装する必要が出てくるため、大きな手戻りとなる。そのため、機能実装前に処理時間などの性能を見積もる必要があるが、数多くの処理割当ての組み合わせの中から、要件を満たす最適な方法を見つけ出すのは困難である。

上記の課題に対して、数理的なアルゴリズムを適用することにより、クラウドとエッジが連携して実現される機能について、処理割当ての最適解の導出を試みた先行研究が複数存在する。Haghighi らは、処理の流れをグラフで表現したモデルを構築し、消費電力と処理時間を制約条件として費用を最小化するアルゴリズムを提案している[3]。Zhang らも、処理フローをグラフで表現し、消費電力を最小化するアルゴリズムを提案している[4]。また、問題領域を数式化しマルコフ決定過程やディープラーニングを利用して、同課題の解決を図る手法も提案されている[5][6]。

しかし、実開発への適用を考えた場合、先行研究の手法で導出された最適解のみから処理割当てを決定することは難しい。実開発ではステークホルダーへの説明のため、複数の処理割当て候補を比較し、選択した候補がどの程度優位か説明を求められる場合もある。また、推測された性能値が、実システムで動作させた時の実測値と比較して許容できない誤差がないかを評価する必要がある。すなわち、処理割当て方法を効率的に決定することに加えて、結論の説明性と精度を考慮した手法を提案する必要がある。

本論文の研究課題を以下に示す。

RQ1(説明性): 処理時間やリソース利用量の推測結果から割当て方法を比較検討できるか

RQ2(精度): 推測された処理時間やリソース利用量は実測値を精度良く再現できるか

RQ3(効率性): 提案手法の推測は、実機を用いた実測よりも効率良く候補を導出できるか

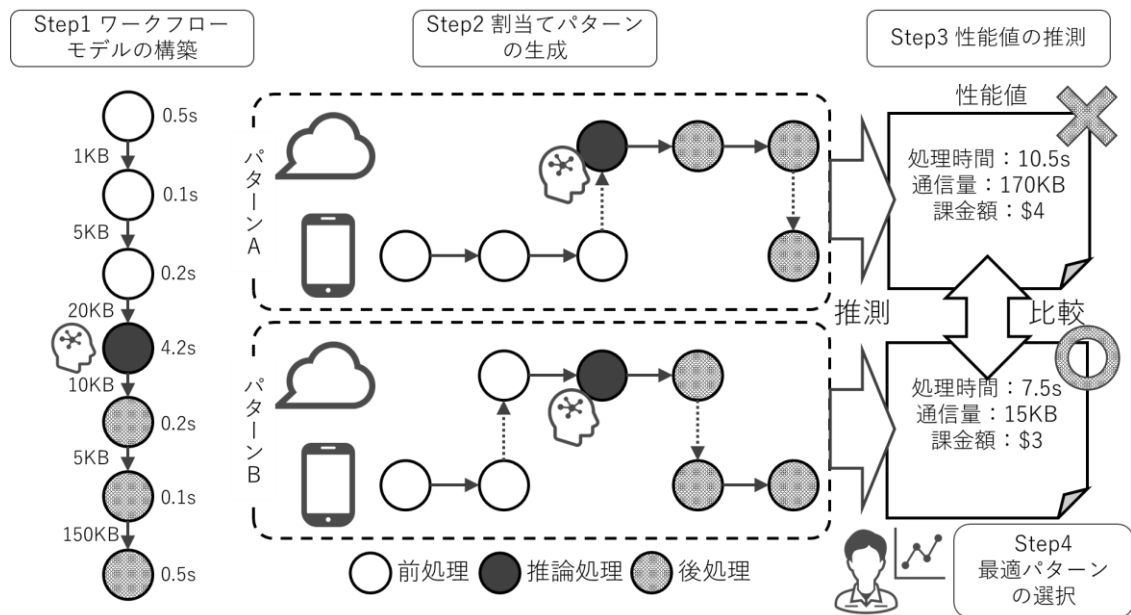


図 2 提案手法の全体像

3. 提案手法

AI の推論フローにおいて、処理の最適な割当てを導出する手法の全体像を図 2 に示す。提案手法は以下の 4 ステップから構成される。以下、詳細を説明する。

- Step1 ワークフローモデルの構築
- Step2 割当てパターンの生成
- Step3 性能値の推測
- Step4 最適パターンの選択

3.1. ワークフローモデルの構築

実装する推論フローを表現するワークフローモデルを構築する。ワークフローモデルは推論フローにおける処理をノード、使用データの流をエッジとして表現される。ワークフローモデルのノードに対象処理の実行時間、エッジに処理間で渡されるデータのサイズを設定する。設定する値は、実機や汎用 PC で推論フローを実行した時の実測値とする。

3.2. 割当てパターンの生成

ワークフローモデルで表現した各処理について、実行する装置をクラウドまたはエッジに配分した割当てパターンを生成する。処理のうち実行装置を固定するものを決定し、それ以外の処理をクラウドまたはエッジに割当てる。処理割当てについては、想定される全ての組合せをパタ

ーンとして生成して性能値を推測する。

3.3. 性能値の推測

各割当てパターンについて、推論フローを実行した時の性能値を推測する。提案手法では、実機での処理時間とクラウド〜エッジ間の通信量、クラウド課金額の 3 つを推測する。

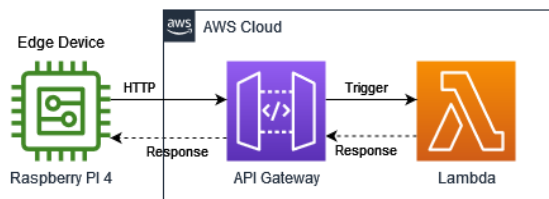
処理時間は、推論フローの処理時間を測定した実測マシンと実機マシンの性能比から推測する。推測処理時間は、以下の式から算出される。

$$\text{推測処理時間} = \text{実測処理時間} \times \frac{\text{実測マシン性能値}}{\text{実機マシン性能値}}$$

マシンの性能値には、CPU のスペックを設定する。例えば、CPU のクロック周波数や PassMark[7]等のベンチマークの値が該当する。また、処理の特性に応じて性能値に追加の補正を加える。例えば、マルチスレッドにて動作する処理の場合、使用可能な CPU のコア数を性能値に乗算して補正する。また、ダウンロード処理やファイルの保存処理など CPU スペックに処理時間が依存しない処理に関しては、性能比による補正を実施しない。

通信量は、処理の実行装置が切替わるフローのデータサイズを加算して推測する。その際、課金額の算出式を考慮し、エッジからクラウドへの通信量とクラウドからエッジへの通信量を分けて算出する。

また、データの通信時間を以下の式で算出し、処理時



クラウド	サービス	AWS Lambda
	CPU	Intel Xeon Processor E5-2680 v3
	クロック周波数	2.5GHz
	割当てコア数	3
	メモリ	4096MB
エッジ	デバイス	Raspberry PI 4 Model B
	CPU	ARM Cortex-A72
	コア数	4
	クロック周波数	1.5GHz
	メモリ	8GB

図 3 本実験のシステム構成

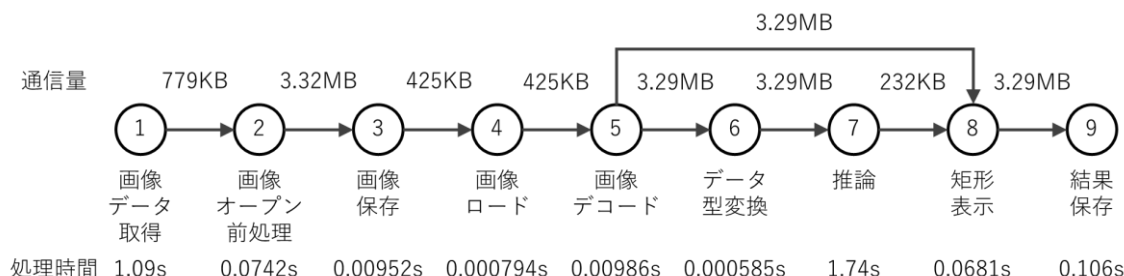


図 4 物体検知のワークフローモデル

間に加算する.

$$\text{通信時間} = \frac{\text{通信データサイズ}}{\text{通信速度}}$$

課金額は、使用するクラウドサービスを想定し、当該サービスの料金体系をもとに導出する。例えば、データ転送の料金の場合、想定するユースケースをもとに単位時間当たりのリクエスト数を定数パラメータとして設定し、推測した通信量を入力して利用額を算出する。

3.4. 最適パターンを選択

各割当てパターンについて推測した性能値をもとに、開発者は最適な割当てパターンを選択する。一般的に、IoT システムにおける処理時間と課金額はトレードオフの関係にある。すなわち、CPU 性能の高いクラウドで処理を実行すると処理時間は短くなるが、クラウドとの通信量やサービス利用による課金額は増大する傾向にある。

よって、提案手法が対象とするのは最適解が一意に定まらない多目的最適化問題であり、複数の最適解の中から対象機能の要求品質を満たす最適パターンを選択する必要がある。選択に際し、複数の性能指標を評価軸として設定し、推測した性能値をプロットすることでパレートフロントを導出する。ここでパレートフロントは、プロットした割当てパターンのうち、他のどの解にも優越されない

解(パレート解)を繋げたトレードオフ曲線である。導出したパレートフロントを確認し、対象機能の要求品質を満たす最適な割当てパターンを選択する。

4. 実験

4.1. 実験概要

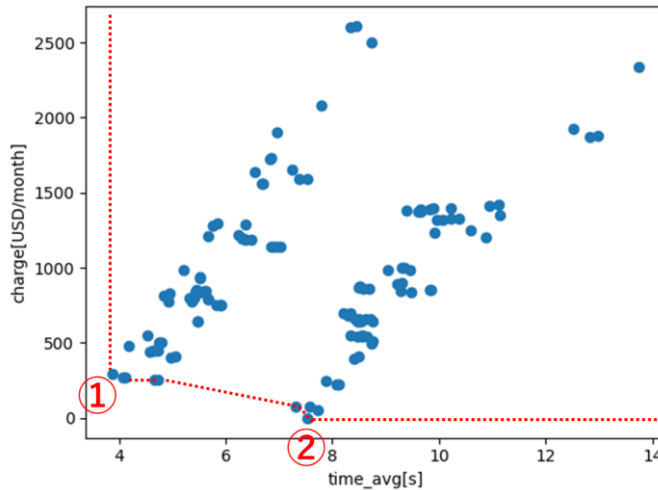
筆者は、物体検知の推論フローを表現したワークフローモデルを構築し、クラウドとエッジで構成されるシステムに実装した時の処理時間と通信量を推測する実験をした。

本実験のシステム構成を図 3 に示す。クラウドのプロバイダーは AWS とし、AWS Lambda を利用したサーバーレスな構成にて推論フローの処理を実行する想定とした。AWS Lambda の実行トリガーは API Gateway とし、外部から HTTP 通信で処理に必要なデータを受け取り処理を実施する。エッジ側は、デバイスは Raspberry PI 4 を使用することを想定し、一部処理をクラウド側に HTTP 通信でリクエストして実行することにより、全体の推論フローを実現する。

物体検知の推論は、フレームワークは TensorFlow を、学習済みモデルは COCO2017 のデータセットで学習済みの Faster R-CNN Resnet-50 V1[8]を利用した。

表 1 実験で使した CPU の性能

	実測PC	クラウド	エッジ
CPU	Intel Core i7-1185G7	Intel Xeon E5-2680 v3	ARM Cortex-A72
クロック周波数	3.0GHz	2.5GHz	1.5GHz
総コア数	4	12	4
割当てコア数	2	3	4
Integer Math	35743	62261	5676
Integer Math(1コア相当)	8936	5188	1419



	パターン①	パターン②
処理割当て	クラウド：2～8 エッジ：1、9	クラウド：なし エッジ：1～9
処理時間 (平均)	3.88s	7.54s
課金額	297USD/month	0USD/month

図 5 割当てパターンの処理時間と課金額

4.2. 提案手法の適用

本実験で構築した物体検知の推論フローのワークフローモデルを図 4 に示す。当該フローは 9 つの処理から構成され、逐次的に処理が実行される。処理 1 から 6 までは推論の前処理、処理 7 が推論の本処理、処理 8 と 9 が後処理に相当する。モデルの各処理に処理時間を、処理間のフローに通信量を設定している。モデルに設定した処理時間と通信量は、物体検知の各処理を汎用 PC 上で動作させた時の実測値を反映している。

上記ワークフローモデルの処理について、各処理をクラウドとエッジに配分した割当てパターンを生成した。本実験では、処理 1 と 9 の割当てをエッジに固定し、それ以外の 7 つの処理をクラウドかエッジか選択して割当てるようにパターンを生成した。割当てパターンの総数は、処理 2 から 8 をクラウドまたはエッジに割当てた時の全組合せとなる $128(=2^7)$ である。

各割当てパターンの処理時間や通信量、課金額の推定方法を説明する。推論フローで実行される各処理は、

実行する CPU の性能により処理時間が変化する。ワークフローモデルに設定した処理時間は実測 PC 上で実行したときの値のため、クラウドまたはエッジ上での処理時間を CPU 性能に応じて補正する必要がある。本実験では、処理割合の大きい推論処理の整数演算のベンチマーク (Integer Math) 値を利用し、クラウドとエッジで想定する CPU と実測 CPU のベンチマーク値の比を乗算し、処理時間を補正した。具体的な推論処理時間の算出式は以下の通り。ただし、推論処理(処理 7)はマルチスレッドで動作するため、Integer Math(1 コア相当)の値にコア数を乗算した値で処理時間を計算する。一方、ダウンロード処理やファイル保存処理(処理 1, 3, 9)には CPU 性能に関連する補正計算を実施しない。

推論処理時間 = 実測PC 処理時間

$$\times \frac{\text{実測CPU Integer Math(1 コア相当)}}{\text{想定CPU Integer Math(1 コア相当)}}$$

測定に使用した実測 PC の CPU 性能と、本実験で想定するクラウドとエッジの CPU 性能を表 1 に示す。通信量は、処理の間で実行装置が変わる場合に通信が発生

表 2 推測値と実測値(平均)の比較

		総時間	処理1	処理2	処理3	処理4	処理5	処理6	処理7	処理8	処理9	通信時間
パターン①	予測値	3.88	1.12	0.126	0.0097	0.00138	0.01731	0.000997	1.99	0.116	0.107	0.396
	実測値	6.38	0.991	0.0595	0.0070	0.00064	0.00724	0.000510	1.93	0.045	0.118	3.51
パターン②	予測値	7.54	1.08	0.466	0.0097	0.00509	0.0609	0.00350	5.39	0.420	0.101	—
	実測値	13.96	0.906	0.122	0.0224	0.00167	0.0293	0.00173	12.5	0.224	0.117	—

するものとし、推論フロー全体の通信量に加算する。クラウドエッジ間のネットワーク速度は、100Mbps とした。課金額は、使用するクラウドサービスの課金の計算式に、推測した処理時間や通信量を代入して導出する。本実験では、5 秒に 1 回の割合で推論フローがリクエストされるものとし、一か月あたりの金額を算出した。

5. 実験結果

5.1. 最適パターンの導出

割当てパターンごとに処理時間と課金額の推測結果をプロットしたグラフを図 5 に示す。また、処理時間と課金額を評価軸としたときのパレートフロントを点線で示した。図 5 のパレートフロントを確認すると、処理時間が最小のパターン(図中①)と課金額が最小のパターン(図中②)が、開発者が選択する最適解として有力な候補となると考えられる。

上記 2 つの候補の概要を図 5 右側の表に整理した。パターン①は割当て可能な処理を全てクラウド側で実行し、パターン②はすべての処理をエッジ側で実行する。

5.2. 実測値との比較

筆者は、前節で導出した推定値の精度を確認するため、図 3 のシステムを実際に構築し、パターン①と②の推論フローを実際に動作させた。そして、各処理の実測処理時間と通信時間を測定して平均値を算出し、推定値と比較した。結果を表 2 に示す。パターン①と②の双方について、推測値と実測値の間に誤差が発生していることが分かる。以下に、発生誤差について考察する。

パターン①の通信時間については、実測の通信時間の方が長い結果となっている。この誤差は、提案手法の通信時間の予測値がデータの転送時間となっており、ネットワークの輻輳等を考慮していないため発生したと考えられる。ネットワーク輻輳の表現などモデルを改良することで、通信時間の予測精度が改善する可能性がある。

パターン②の処理 7 は、予測値よりも実測値の方が約

2.3 倍長い結果となった。処理 7 は推論処理となっておりマルチスレッドで動作する。マルチスレッドの動作によりエッジの CPU リソースを全て消費してしまい性能劣化した可能性があるが、他のモデルで動作させた時の動作と比較し、原因を検証する必要がある。

5.3. 考察

本論文の研究課題について、以下にて考察する。

RQ1(説明性): 処理時間やリソース利用量の推測結果から割当て方法を比較検討できるか

物体検知の推論フローを対象として、ワークフローモデルを構築することにより、割当てパターンごとの処理時間や通信量、課金額を机上で推測できることを確認した。また、割当てパターンごとに推測した処理時間や通信量、課金額を評価軸として設定し、パレートフロントを導出することにより、最適な処理割当てパターンの候補を複数導出し、比較検討できることを確認した。

本論文の検証に用いた推論フローは、構造が簡潔なものとなっている。処理が複雑なフローに対してワークフローモデルを構築する時に発生する課題があるか検証する必要がある。また、通信のスループットなどの制約条件をモデルに組み込み、想定する IoT システムを構築する上での前提を満たしているかについて、確認できる仕組みにした方が良く考える。

RQ2(精度): 推測された処理時間やリソース利用量は実測値を精度良く再現できるか

導出された割当てパターン候補は、パターン①が処理時間最適、パターン②が課金額最適となっていた。実測結果においても、パターン①はクラウドの計算リソースを利用することでパターン②よりも処理時間が短い結果となり、推測結果と同等の傾向があることを確認できた。一方で、個々の処理時間の推測精度は改善の余地があり、特に性能データの変換方法について以下に示す改良点があると考ええる。

- 高負荷時における CPU 性能劣化の考慮

- 通信時間推測モデルの改良

RQ3(効率性): 提案手法の推測は、実機を用いた実測よりも効率良く候補を導出できるか

提案手法による推測に、2 日程度の時間を要した。おおよその内訳は、ワークフローモデルのプログラム実装に 1 日、割当てパターンの生成と性能値の推測に半日、パレートフロントの導出に半日である。

一方、実機を用いた実測は、エッジの環境構築とクラウドサービスの構築、エッジとクラウド間の通信プログラムの作成、処理時間のログ出力など多くの作業が必要となり、約 7 日程度の時間を要した。

実機を用いて検証する場合は、提案手法による推測とは異なり、実装上の課題を解決しなければならない。例えば、AWS Lambda はデプロイパッケージのサイズに上限があり、サイズの大きい機械学習ライブラリを使用する場合は、当該制限を回避するためコンテナを作成する必要がある。API Gateway と Lambda の接続等、クラウドサービス間の連携方式についても実装する必要があり、接続のための設定や動作確認にも時間を要する。さらに、実システムを用いた検証には、エッジデバイスの調達や、クラウドアカウントやロールの設定など、環境構築に着手する前段階での準備が必要となる場合がある。本実験においては不要であったが、状況によっては上記準備の対応のため、検証期間がさらに長期化する可能性がある。

また、実機を用いた場合、一つの環境で検証できる割当てパターンは基本的に一つのみである。複数の割当てパターンを検証するには、複数のプログラムを作成するか、割当てに応じて処理を柔軟に変更できるプログラム構造にする必要があるため、さらに対応工数が増加する。この観点においても、提案手法を用いた推測の方が、効率的にかつ網羅的に最適な割当てパターンを検証できると考える。

6. おわりに

本論文では、物体検知の推論機能を対象としたワークフローモデルを構築することにより、クラウドとエッジで構成されるシステムにおける処理割当て方法を決定する手法を示した。提案手法により、割当てパターンごとに処理時間やリソース利用量を推測した結果をプロットしてパレートフロントを導出することで、最適な割当てパターンの候補を効率良く提示できることを確認した。

今後の展望を以下に示す。

- 他の AI 推論フローに対する適用を通じた汎用性の評価
- クラウドとエッジでの処理時間と通信時間の推測精度向上
- デバイス変更時の影響分析などユースケースに応じた提案手法の応用可能性の検討

謝辞

本研究を進めるにあたり、有意義な議論の場を提供頂いた一般財団法人日本科学技術連盟に深く御礼申し上げます。また、議論の場において多くの有益な意見を頂戴した研究会の皆様には感謝いたします。

参考文献

- [1] IoT Analytics, State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally, <https://iot-analytics.com/number-connected-iot-devices/> (2022/12/27 参照)
- [2] Z. Zhou et al., Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing, Proceedings of the IEEE, Vol.107, No.8, P.1738-1762, 2019.
- [3] V.Haghighi and N. S. Moayedian, An Offloading Strategy in Mobile Cloud Computing Considering Energy and Delay Constraints, IEEE Access, Vol.6, P.11849-11861, 2018.
- [4] W. Zhang and Y. Wen, Energy-Efficient Task Execution for Application as a General Topology in Mobile Cloud Computing, IEEE Transactions on Cloud Computing, Vol.6, No.3, P.708-719, 2018.
- [5] K. R. Alasmari et al., Mobile Edge Offloading Using Markov Decision Processes, EDGE 2018, P.80-90, 2018.
- [6] H. Wu et al., Collaborate Edge and Cloud Computing With Distributed Deep Learning for Smart City Internet of Thing, IEEE Internet of Things Journal, Vol.7, No.9, P.8099-8110, 2020.
- [7] PassMark Software - CPU Benchmark Charts, <https://www.cpubenchmark.net/>, (2022/12/27 参照)
- [8] TensorFlow Hub: Faster R-CNN Resnet-50 V1 640x640, https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_640x640/1 (2022/12/27 参照)

除雪出動決定支援のための除雪関連情報共有サイトの構築

大倉 博貴

北海道大学 大学院情報科学院
hiroki0113@eis.hokudai.ac.jp

山下 倫央

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

横山 想一郎

北海道大学 大学院情報科学研究院
yokoyama@ist.hokudai.ac.jp

川村 秀憲

北海道大学 大学院情報科学研究院
kawamura@ist.hokudai.ac.jp

要旨

北海道を始めとする積雪地帯において、道路除雪業務は道路交通を維持する重要な役割を果たしている一方で、除雪作業を行う従業員の業務負担は大きい。除雪業務の出動判断は局所的な天候の変動によって出動判断が覆ることが多いため、除雪作業員は休日でも出動に備えておく必要があり、除雪従業員の大きな負担となっている。

このような背景に基づき、本研究では担当者の行う除雪出動の判断が覆らず、除雪対象範囲を安全に遠隔監視するために、除雪出動の決定に必要な情報を自動で収集し、可視化する除雪出動の意思決定支援システムの開発に取り組んでいる。本稿では、除雪出動決定支援システムの一部として既に実用化されている収集した道路の積雪状況や気象情報を可視化するための除雪関連情報共有サイトを構成について述べる。また、除雪関連情報共有サイトに追加して掲載する情報として、定点カメラ画像における積雪割合推定機能を開発している。深層学習を用いて定点カメラ画像の積雪範囲を数値化する積雪割合推定モジュールを提案し、実際の定点カメラ画像に対する推定結果を確認する。

1. はじめに

道路除雪業務は積雪地帯における冬季の道路交通や生活インフラを維持するために重要な役割を果たしている。道路除雪業務の問題点のひとつに、道路除雪業務の出動判断の難しさが挙げられる。道路除雪業務の出動判

断は、担当者が16時に翌日2時に除雪業務を行うかどうかを判断する。翌日2時からおこなう除雪業務はそれまでの道路の除雪状況や16時以降の局所的な天候変化に大きく影響を受けるため、ベテラン担当者でも16時に行った判断を覆すことが多い。

このように出動判断が覆ることが多くあるため、実際に除雪をおこなう除雪作業員は16時に出動がないと判断された場合でも、出動に備える必要があり、除雪作業員の大きな負担となっている。また、道路除雪の直前に実際に除雪対象エリアに自動車で行って積雪状況を目視で確認する雪見巡回が除雪出動の判断を下すために行われている。しかし、積雪している深夜の雪道の走行は車両のスタックやスリップ事故の危険を伴うため、代替手法が望まれている。

本研究では、このような除雪出動の意思決定に伴う負担を軽減するため、大規模かつ複雑なデータに対して視覚的なインターフェースを利用して解析的な推論をおこなう Visual analytics に基づくアプローチを採用する。[1] 道路の積雪状況や気象情報を収集・可視化して、担当者の除雪出動の決定支援をおこなうシステムの開発をおこなっている。本研究で開発を目指している除雪出動決定支援システムの機能のひとつとして、除雪対象エリアに設置した定点カメラからリアルタイムに道路画像を取得し、気象情報を収集する機能を実装する。

収集した情報をシステム上で可視化して、担当者の意思決定に利用し、天候状況の変動があっても覆らない出動判断をおこない、雪見巡回の手間を省くことを目的とする。さらに、除雪作業員も収集した情報を閲覧可能と

することで、天候の変動等の出動判断が変更される予兆を共有可能とすることを目指す。

本稿では、除雪出動決定支援システムの一部として先行して利用を開始している収集した道路の積雪状況や気象情報を可視化するための除雪関連情報共有サイトを構成について述べる。また、除雪関連情報共有サイトに掲載する情報として、定点カメラ画像の積雪割合が挙げられる。定点カメラの目視確認を自動化するために、定点カメラ画像の積雪範囲を数値化する深層学習を用いた積雪割合推定モジュールの提案を行い、その推定機能を確認する。

2. 関連研究

本章では積雪割合推定に利用する画像処理に関する関連技術について説明する。

2.1 セマンティックセグメンテーション

セマンティックセグメンテーションとは画像をピクセル単位に分割し、すべてのピクセルに意味的ラベルの付与を行う深層学習を用いた画像認識タスクのことを指す。このタスクの分野では、Fully Convolution Network[2](以下、FCN) が発表されて以来、この FCN の構造を利用した手法が基本となっている。高解像度の特徴量マップを得るために、Encoder として FCN を利用し、Decoder として Encoder の特徴量マップを取り入れて空間情報を回復する Encoder-Decoder 構造が提案され、その中で代表的なものとして、Seg-Net[3]、U-Net[4] などがある。Seg-Net は前半の Encoder でダウンサンプリングを行った回数分、後半の Decoder でアップサンプリングを行う。各プーリング層での最大値の空間インデックスを、解像度が同じ Decoder 側のアップサンプリングの際に利用することで、より明確に推論できる。U-Net は Seg-Net と同じ Encoder-Decoder 構造であるが、低次元の特徴を高次元の特徴へ結合するスキップ接続が採用されており、これはダウンサンプリングの際に失われてしまう情報を回復するための手段である。この他にも PSP-Net[5] や DeepLab[6] を利用した手法があり、さまざまな研究が行われている。

また、ロードヒーティングの対象路面を俯瞰する画像を用いて、セグメンテーションを行うことで路面の積雪を判定する研究がある。今原の研究 [7] では、札幌市内

の駐車場を 30 メートル程の高さから俯瞰する画像の積雪判定を行った結果、ロードヒーティングの制御において問題のない判定精度が得られることが示されている。

本研究の積雪割合推定でセマンティックセグメンテーションを用いる理由を述べる。現在、除雪対象地点に設置された定点カメラの撮影画像を目視で確認することで除雪出動決定の判断材料として利用がされている。そこで、目視による積雪状況の判定を自動で数値化することに取り組む。画像の積雪判定を回帰問題として捉え、画像認識を用いて行う上で邪魔になる車や建築物、人などの障害物に対処するためには事前処理が必要である。この事前処理は障害物自体の移動や、カメラ角度の変化が起こる関係で無視できない作業量となってしまう。そのため障害物領域の推定もセマンティックセグメンテーションで試みることで、作業量を減らす。他にも 1 ピクセルごとに学習データがあると考えられるので、1 枚の積雪割合を回帰問題として捉えるよりもデータ数を多く扱うことができる点、積雪が多くなりやすい領域や積雪が少なくなりやすい領域の情報を利用できる点が挙げられる。

3. 除雪出動意思決定支援システム

本研究で開発する除雪出動の意思決定を支援するシステムに関して、可視化するデータとシステム構成について述べる。

3.1. 可視化対象データ

本研究で開発するシステムは、北海道留萌市内の除雪対象の 5 地点に対して、定点カメラ画像と気象情報を収集し可視化する。各地点には異なる角度から撮影する定点カメラ 2 台を設置し、詳細な積雪状況を把握する。システムで可視化するデータは以下の 4 つである。

定点カメラの撮影画像 各地点に設置してあるネットワークカメラでリアルタイムで撮影した道路画像。

定点カメラの撮影画像の解析による積雪率 道路画像に写る除雪対象路面領域の中の積雪領域の面積の割合を、セマンティックセグメンテーションによる解析で求めた値。

気象情報 WeatherNews が提供する天気予報に関する情報 [8]。天気予報は 72 時間先までの 1 時間おきの天

気予報である短期予報と、10 日間先までの 1 日おきの天気予報である中期予報がある。短期予報は、天気コード、気温、降水量、気圧、風速、風向、相対湿度からなる。中期予報は、天気コード、最高気温、最低気温、降水確率からなる。

積雪深情報 WeatherNews が提供する積雪深に関する情報 [8]。積雪深とは芝上の積雪を想定した cm 単位での数値である。10 分毎の積雪深実況と、60 時間先までの 1 時間おきの積雪深予報からなる。

気象情報と積雪深情報は、緯度経度を指定することで除雪対象地点の詳細な情報を得ることができる。

3.2. システムの構成

除雪出動の意思決定支援システムの構成を図 1 に示す。本提案システムは、クライアントサイド、サーバサイド、データベース、外部データの 4 つから構成される。外部データは定点カメラの撮影画像と気象情報、積雪深情報の 3 つのデータを示している。外部データをサーバサイドで呼び出し、データベースへと格納することで外部データを収集する。また、サーバサイドがデータベースに格納されたデータを API 化することで、クライアントサイドが常にデータベースからデータを取り出すことが可能である。

本提案システムの開発に利用した技術スタックを以下に示す。

- クライアントサイド：JavaScript, React
- サーバサイド：Python, FastAPI
- データベース：MySQL

3.3. システムの機能

本提案システムには、除雪対象地点の定点カメラ画像をリアルタイムで表示する機能、除雪出動の意思決定に利用する気象情報と積雪深情報を表示する機能を実装する。クライアントサイドからは以下の画面を通じて、情報の入力や閲覧をおこなう。

地点選択画面 除雪対象の全地点が表示され、情報を閲覧したい地点を選択することができる画面。表示される地点ごとにリアルタイムの定点カメラの撮影画

像とその更新時刻を表示する。具体的な画面を図 2 に示す。

地点ごとのリアルタイム情報画面 選択された地点のリアルタイムの定点カメラの撮影画像とその地点の気象情報、積雪深情報を表示する画面。

地点ごとの過去情報画面 選択された地点の過去の定点カメラの撮影画像とその地点の気象情報、積雪深情報を表示する画面。気象情報と積雪深情報についてはグラフで可視化することで時系列的な情報を確認できる。具体的な画面を図 3 に示す。

地点ごとの予報情報画面 選択された地点の気象予報情報、積雪深予報情報を表示する画面。気象予報情報と積雪深予報情報を可視化することで時系列的な情報を確認できる。

積雪深値の入力フォーム画面 除雪作業員が計測した積雪深を入力する画面。

4. システムの導入効果の検証

本提案システムは、i) 出動判断をおこなう担当者が事後に覆らない出動判断をおこなうこと、ii) 雪見巡回の手間を省くこと、iii) 除雪従業員が出動判断の担当者と同じ情報を閲覧し出動に備えること、を目指している。

道路画像に写る積雪領域の割合は、出動判断に大きな影響を与えるため、セマンティックセグメンテーションによる自動解析は本提案システムの有用性に大きく貢献する。そこで、2.1 節で述べた、定点カメラの撮影画像から目視による積雪状況の確認を自動化・数値化する技術を適用し、積雪割合推定の精度検証を行う。また、画像時系列に対する積雪割合推定の結果と除雪出動の有無を比較し、技術が除雪出動意思決定支援における有用性を考察する。

4.1. 深層学習を用いた積雪割合推定の精度検証

深層学習による積雪割合推定の精度に関して、領域推定や積雪割合の推定の精度を評価指標として検証を行う。

4.1.1 データセット

検証で利用するデータセットについて述べる。3.1 節で述べた定点カメラの撮影画像を利用する。本検証では

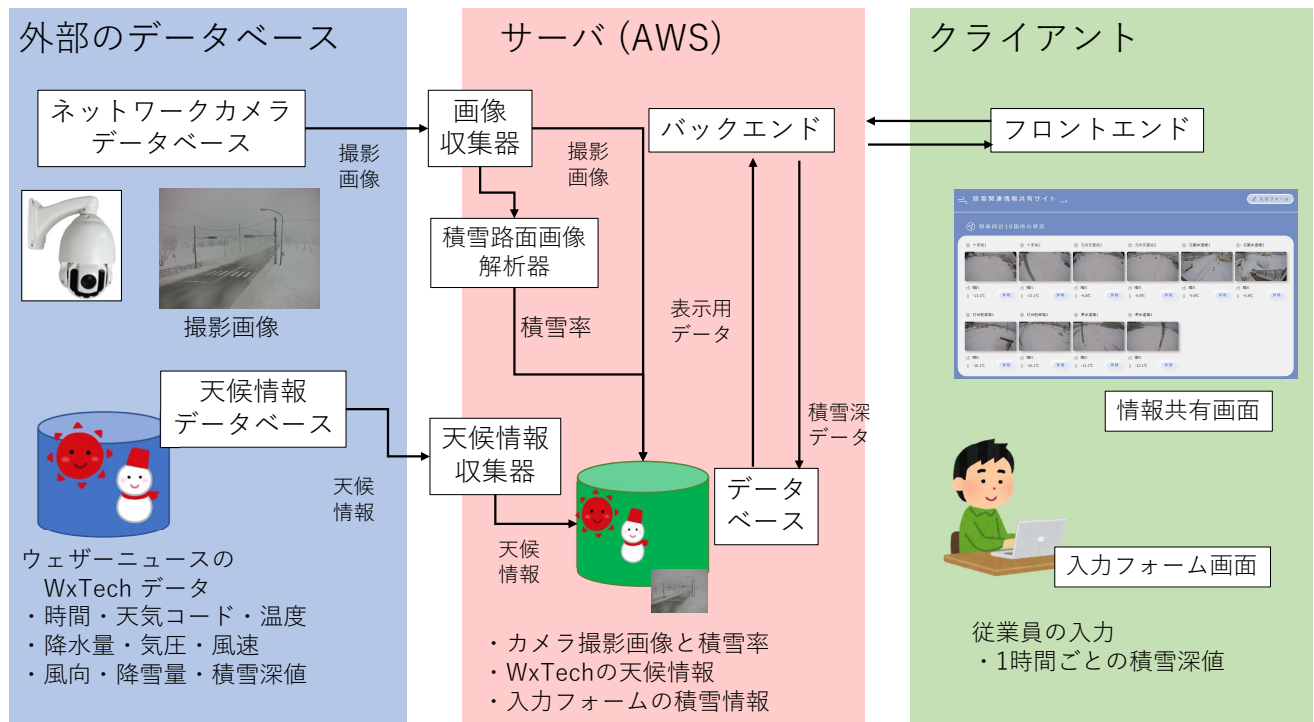


図 1. システム構成の概要

ラベル付けやシステムの都合上 10 地点中 8 地点の撮影画像のみを利用する。撮影期間は 2022 年 12 月 19 日から 2023 年 1 月 27 日であり、その中から同日に 1 から 3 枚程度を選択した 752 枚をデータセットとして利用する。画像の選択について、それぞれ 6 時間以上時間間隔が離れており時系列的な偏りはない。また天候的な偏りについて、積雪が全くない状況と路面が見えなくなるほどの積雪がある状況、それらのどちらにも当たらない状況をそれぞれ選択し、偏りはあるものの様々な状況を選択する。

学習用データのラベル付け定義について述べる。対象となる撮影画像に対し正解ラベルを付与し、学習用の正解であるマスクデータとする。

画像内の全ピクセルにラベルを付与するものとし、画像内のピクセルはすべていずれかのクラスに属するものとする。画像に付与する正解ラベルは下記の 3 種類である。

積雪 積雪が確認された除雪対象路面の領域に付与するラベル。

非積雪 積雪が確認されていない除雪対象路面の領域に

付与するラベル。

障害物 上記の 2 クラスの判定の障害となるもので除雪対象路面に関連するものに付与するラベル。建物、車、木、人、電柱など該当する。

無関係な領域 上記の 3 クラスに分類されないすべての領域に付与するラベル。

このようなマスクデータを撮影画像の選択枚数だけ用意し、作成したマスクデータと対応する撮影画像のデータセットを用いて学習を行う。

4.1.2 学習モデルと設定

セマンティックセグメンテーションのタスクで高いスコアを残していた学習モデルとして、Unet++を利用し、Encoder の特徴抽出の部分では Xception[9] のネットワーク構造を用いる。また、Imagenet のデータセットによる事前学習済みモデルを利用する。

学習パラメータは表 1 に示す。



図 2. 地点選択画面

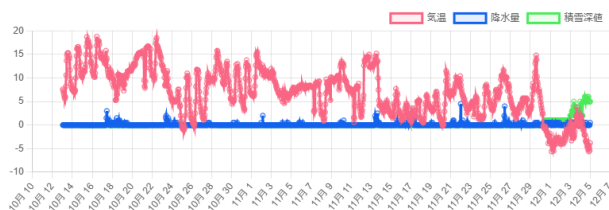


図 3. 地点ごとの過去情報画面

データセットについて、4.1.1 節で述べたものを学習画像、検証画像の割合が 8:2 の比になるように撮影地点、撮影日ごとに分配した。撮影日ごとに学習画像と検証画像を分けることによって、学習画像と異なる日付に撮影された画像に対する推定精度を評価し、システムの実運用の際の性能をより正確に見積もることができる。

表 1. 学習パラメータ

エポック数	40
ミニバッチサイズ	2
最適化	Adam
学習率	初期値:0.0001 25 エポック以降:0.00001

4.1.3 評価指標

評価指標には、Intersection over Union を用いた。

Intersection over Union(以下 IoU) とは、物体検出の評価指標である。正解ラベルを付与した画像と推定結果の画像を比べた際に、正解領域と推定領域が重複する割合を表す。画像 1 枚 1 枚に対し、積雪、非積雪、障害物クラスの IoU の平均値を算出する。各クラスをそれぞれ正例とし、画像内の各画素に対して表 2 のような混同行

表 2. 混同行列

		推定結果	
		正例	負例
正解データ	正例	TP(True Positive)	FN(False Negative)
	負例	FP(False Positive)	TN(True Negative)

列を作成する。

このとき IoU は下記式で与えられる。

$$IoU = \frac{TP}{TP + FP + FN} \quad (1)$$

4.1.4 積雪割合推定方法

積雪割合推定方法について述べる。今原ら [7] が提案した積雪割合を利用した積雪判定を行う。2つの利用したデータセットを対象に、積雪割合について算出を行う。各画像の積雪割合を式 2 で定義する。

$$\text{積雪割合} = \frac{\text{積雪ラベルのピクセル数}}{(\text{積雪ラベル} + \text{非積雪ラベル}) \text{のピクセル数}} \quad (2)$$

本研究ではこの値を利用した積雪判定が、目視による確認の自動化・数値化として用いることができるかを考察する。

4.1.5 検証結果

評価指標に対する結果として、IoU は 0.951 となり学習がうまく実行されていることがわかる。

検証画像として利用した 172 枚に対する積雪割合推定結果を述べる。式 2 で示される積雪割合を正解データと推定結果のマスクデータについて算出し、その違いを比較する。

正解データと推定結果の積雪割合の混同行列を表 3 に示す。多くの画像では正解データと推定結果の値の範囲が一致し、大きな外れ値がないことがわかる。

4.2. 画像時系列に対する積雪割合推定の検証

4.1 節で検証した積雪割合推定を定点カメラ画像に対して適用し、除雪出動意思決定の支援における有用性を調査する。

4.2.1 実験設定

検証で利用するデータセットについて述べる。3.1 節で述べた全 10 地点の定点カメラの撮影画像を利用する。撮影期間は 2022 年 2 月 21 日 12 時から 2023 年 2 月 25 日 12 時、撮影間隔は 10 分ごと、撮影画像枚数は 1 地点ごと 576 枚で全地点合計 5,760 枚である。

学習モデル、パラメータについて、4.1.2 節で述べた学習済みモデルを利用する。本検証では、未学習のデータセット、さらに学習画像のない 2 地点を含むものを利用する。

4.2.2 検証方法

検証方法について述べる。データセットに対して学習済みモデルを適用、積雪割合を推定し 10 地点の平均値を扱い、除雪出動が必要・不要な時間帯を比較する。ただし、定点カメラの撮影画像を目視で確認し、道路面が見えなくなる程度の積雪が確認できる時間帯とし、それ以外の時間帯を除雪出動が不要な時間帯とする。

4.2.3 検証結果

検証結果について述べる。データセットに対する推定積雪割合の平均値の時間推移を図 4 に示す。除雪出動が必要な時間帯は 2 月 21 日 19 時から 2 月 22 日 5 時までの時間帯と、2 月 25 日 6 時から 2 月 25 日 7 時までの時間帯である。図 4 から、除雪出動が必要な時間帯は積雪割合が 100 % に近い数値を 30 分間以上維持することがわかる。また、除雪出動が不要な時間帯では、積雪割合が増加するがすぐに減少する、または 100 % よりも低い数値を維持することがわかる。これは、画像時系列に対する積雪割合推定が、除雪出動が必要な時間帯を捉えていることを示し、これまで目視により行われていた積雪状況の確認の自動化・数値化を通して除雪出動意思決定の支援における有用性を明らかにしている。

5. まとめ

本稿では、除雪出動決定支援システムの一部として、既に実際に利用が開始されている。除雪関連情報共有サイトを概説した。除雪関連情報共有サイトは、除雪対象地点のリアルタイムで道路画像とその地点の気象情報を

表 3. 積雪割合の混同行列

		正解データの積雪割合										
		0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
推定結果の積雪割合	0.0	2	0	0	0	0	0	0	0	0	0	0
	0.1	9	22	0	0	0	0	0	0	0	0	0
	0.2	0	4	2	0	0	0	0	0	0	0	0
	0.3	0	1	1	0	0	0	0	0	0	0	0
	0.4	0	1	0	0	0	0	0	0	0	0	0
	0.5	0	0	0	0	1	0	0	0	0	0	0
	0.6	0	0	0	0	0	0	0	0	1	1	0
	0.7	0	0	0	0	0	0	0	0	0	0	0
	0.8	0	0	0	0	0	0	0	0	0	0	0
	0.9	0	0	0	0	0	0	0	0	0	0	1
	1.0	0	0	0	0	0	0	0	0	0	0	126

自動収集・サイト上での可視化をおこない、除雪出動を決定する担当者だけでなく除雪作業員も閲覧して、関連情報の共有を可能にした。さらに、今後の除雪出動決定支援システムの導入効果の検証に関して概説した。また、収集した定点カメラの撮影画像を用いた深層学習による積雪割合推定に取り組み、その結果と利用可能性を説明した。今後の展開として、出勤判断の変更回数、担当者の負担軽減への寄与度、雪見巡回の負荷軽減、情報閲覧による除雪従業員の精神的な負担軽減を調査し、収集した気象情報と除雪出動履歴、推定積雪割合を用いて、深層学習による除雪出動必要性の推定に取り組む。

謝辞

本研究は、PRISM (Public/Private R&D Investment Strategic Expansion PrograM) 官民研究開発投資拡大プログラムの国土交通省「建設現場の生産性を飛躍的に向上するための革新的技術の導入・活用に関するプロジェクト」の支援を受けたものである。

参考文献

- [1] James J. Thomas and Kristin A. Cook. Illuminating the path: The research and development agenda for visual analytics. 2005.
- [2] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.
- [3] Roberto Cipolla Vijay Badrinarayanan, Alex Kendall. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 2481–2495, 2017.
- [4] Thomas Brox Olaf Ronneberger, Philipp Fischer. U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pp. 234–241, 2015.
- [5] Shi J. Qi X. Wang X. Zhao, H. and J. Jia. Pyramid scene parsing network. *IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, pp. 6230–6239, 2017.
- [6] George Papandreou Florian Schroff Hartwig Adam Liang-Chieh Chen, Yukun Zhu. Encoder-decoder with atrous separable convolution for semantic image segmentation. *European Conference on Computer Vision (ECCV)*, pp. 801–818, 2018.
- [7] 今原智広, 横山想一郎, 山下倫央, 川村秀憲. ロードヒーティング制御に向けたセマンティックセグメン

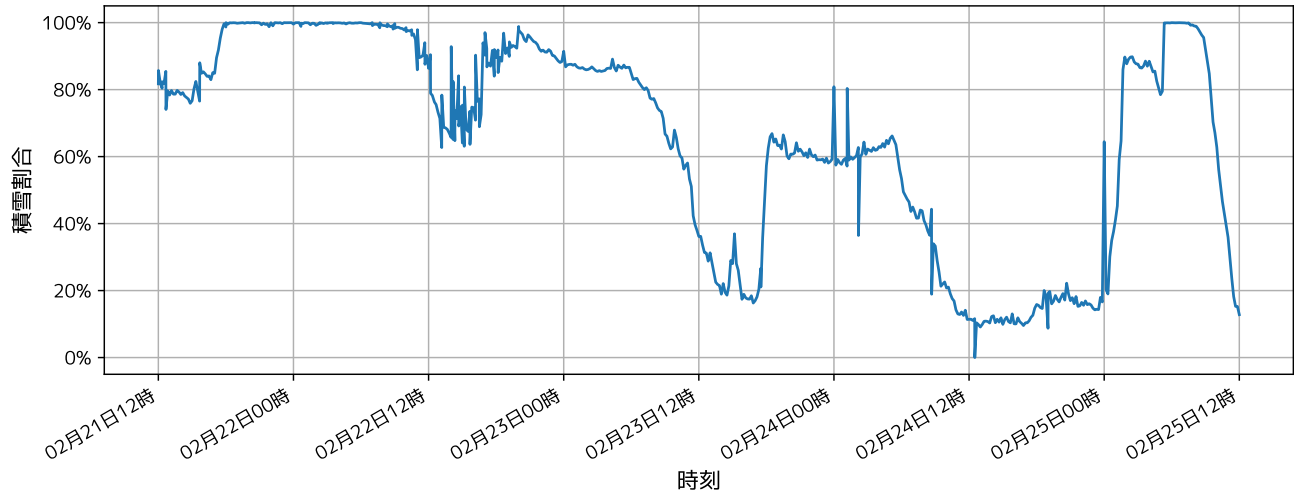


図 4. 画像時系列に対する推定積雪割合平均値の時間推移

テーションによる路面画像の積雪状況認識. 第 201
回知能システム研究発表会, 2021.

[8] ウェザーテック (wxtech[®]): 気象データのビジネス
活用・分析. <https://wxtech.weathernews.com/>.
(Accessed on 01/12/2023).

[9] Francois Chollet. Xception: Deep learning with
depthwise separable convolutions. *IEEE Confer-
ence on Computer Vision and Pattern Recognition
(CVPR)*, July 2017.

人物追跡手法とマッチングアルゴリズムを用いた路線バスにおける乗客 OD 推定

西浦 翼

北海道大学大学院情報科学院

tsubasanishiura@ist.hokudai.ac.jp

横山 想一郎 山下 倫央 川村 秀憲

北海道大学大学院情報科学研究院

佐藤 好美 長谷川 怜 平澤 幸

株式会社シーズラボ

要旨

日本国内では深刻な少子化による人口減少からコンパクトシティ化が進んでおり、特にバス路線に関して公共交通ネットワークを再編成する必要がある。交通計画を策定するためには住民の移動を集計した OD (Origin-Destination) データが必要である。OD データの収集はその需要と裏腹に、高いコストなどから収集できていない現状にある。そこで本研究では路線バスを対象に、カメラで撮影した動画データからバス乗客 OD を推定するシステムを開発する。様々な要素技術を検討し、物体検出、物体追跡、メトリックラーニング、割当アルゴリズムを組み合わせたバス乗客 OD 推定システムを提案する。データセットとして、実際に運行している路線バスにカメラを取り付けてバスに乗降する人物の動画を撮影し、人物矩形のアノテーションを行った。モデルの選定やモデルの学習を行って推定精度を検証し、精度低下の原因を分析するとともに、精度向上の施策を行った。本研究の結果は、コストや労力をかけずに OD データを収集する手段として貢献することが期待される。

1. はじめに

日本は少子化による人口減少という深刻な問題に直面しており、コンパクトシティ開発に向けた議論が進んでいる。その結果、特にバス路線に関して、公共交通ネットワークを再編成する必要性が高まっている。都市中心部では、主要な幹線道路を走るバス路線が多く重複し、競合している。これを解決するために、幹線と支線を区

別し、各々の役割を明確化することが求められている。また、路線の長さを短くすることで定時性を確保することも重要である。新しいバス交通計画を策定するためには、バス利用者の移動を集計したバス乗客 OD データが必要である。

バス乗客 OD データには、「バス停別乗降者数」と「乗車区間別乗降者数」という2つの種類がある。バス停別乗降者数を分析すれば、バス停や時間帯ごとの利用者の多寡を把握でき、バス停周辺の路線の追加・削除や、その時間帯の増便・減便が可能になる。また、乗降区間別乗降者数から人流を分析できれば、利用区間ごとの運賃の見直しなどにも活用できる [1]。

実際に、交通計画策定に向けたバス OD データ収集の需要が高まっており、平成 28 年度に国土交通政策研究所が自治体に対して行ったアンケートで、バス乗客 OD データの必要性が裏付けられている [2]。民間バス事業者の持つデータについて、地方公共団体が必要だと考えるものの回答 684 件のうち、56.4 %にあたる 386 件が「各バス停間の OD」であった。しかし、回答 581 件のうちわずか 11.2 %にあたる 65 件しか、「対象路線の各バス停間の OD」を民間バス事業者から収集できていなかった。このようにバス乗客 OD データは、バス事業者が必要だと考えているにもかかわらず、収集できていない状況にある。

現在の OD データ収集手法としては、パーソントリップ調査などのアンケート調査や調査員を派遣した人手による調査、IC カードの使用履歴データなどがある。アンケート調査や調査員による手法は、費用や工数の面で非常にコストが高い。また、IC カードデータは、IC カードを発行している事業者が管理しており、外部から利用

するのにハードルが高いという問題がある。

このような背景に基づいて、本研究では、バス車内にカメラを設置し、乗降する人物を動画で撮影することで、深層学習を用いた人物追跡技術を適用し、OD データを推定する乗客 OD データ推定システムを開発した。このシステムを利用することで、バス乗客 OD データの収集が効率的に行われることが期待される。これにより、バス路線の最適化や運賃制度の見直しなど、さまざまな交通計画の策定が可能になると考えられる。

本稿の構成を以下に示す。2 章では、バス乗客 OD 推定に必要な技術に関する研究について概説する。3 章では、提案システムの概要を説明する。4 章では、本研究で利用する乗降客の動画データセットを説明する。5 章では、その有効性を確認するための実験について説明し、実験結果を考察する。最後に 6 章で本論文をまとめる。

2. 関連研究

本章では、バス乗客 OD 推定に用いる 4 つの技術、物体検出、物体追跡、メトリックラーニング、割当アルゴリズムに関して概要を説明する。

2.1. 物体検出

Computer Vision のタスクの 1 つに物体検出 (Object Detection) がある。これは入力となる画像の中から定められた物体の位置を推定するタスク (Localization) とクラスを推定するタスク (Classification) が複合したタスクである。特に、人物クラスを検出することを人物検出 (Human Detection) と呼ぶ。

有名な手法として、2005 年に提案された HOG (Histogram of Oriented Gradients) 特徴 [3] がある。HOG 特徴は計算コストが低く、実際の問題に対して非常に効果的である。しかし、その数年後、深層学習を用いた手法が登場した。2012 年の ILSVRC で AlexNet[4] が優れた結果を残して以降、ResNet[5] をはじめとする畳み込みニューラルネットワーク (Convolutional Neural Network; CNN) が画像認識に広く用いられるようになった。近年では、高い精度を維持しながら効率的な計算を実現する EfficientNet[6] が開発されている。

物体検出に CNN アルゴリズムを応用した先駆けとなったモデルが、2014 年に提案された R-CNN (Region-based CNN) [7] である。CNN は計算量が大きく推論速

度が遅いという課題があったが、Selective Search と呼ばれる物体候補アルゴリズムを導入することで計算を高速化した。その後、Fast R-CNN[8]、Faster R-CNN[9]、SSD (Single Shot Detector) [10] などが提案されてきた。

特に注目されるのが、YOLO (You Only Look Once) [11] シリーズである。YOLO は画像全体をあらかじめグリッド分割し、各グリッドごとに物体の矩形と確信度を出力する方法を採用している。このため、推論速度が高いことが特徴である。YOLO シリーズは現在も様々な研究者や研究チームによって派生モデルが開発されている。

各手法の性能については、YOLO シリーズが高速な推論速度を誇るが、精度面では Faster R-CNN や SSD といった他の手法と比較して劣ることがある。一方で、Fast R-CNN や Faster R-CNN は精度が高いものの、推論速度が遅いことが課題である。

2.2. 物体追跡

物体追跡 (Object Tracking) は動画内に映っている物体や人物を追跡するタスクで、画像内から物体の位置とクラスを推定する物体検出 (Object Detection) と、複数の画像間で検出された同一の物体をマッチングさせる再同定 (Re-Identification) から構成される。物体追跡技術は、カルマンフィルター、パーティクルフィルター、光流法、Mean Shift などの従来手法から、近年の深層学習を用いた手法 (Deep SORT[12] や MDNet[13] など) まで、さまざまなアルゴリズムが提案されている。

一般的な Re-Identification と比べて、画像の時系列情報や画像のどこに矩形が位置していたのかという座標情報が使える点が異なる。Object Detection と Re-Identification のそれぞれに多くの課題が存在し、評価指標も物体追跡を行う目的に応じて変化する。物体追跡の性能評価には、MOT Challenge や VOT Challenge などのデータセットや、MOTA, MOTP, ID スイッチなどの評価指標が利用されている。

また、人物追跡では、追跡の対象となるデータによって難易度が大きく変わる。まず、人物追跡のタスクの難易度は追跡する対象の多寡によって異なる。リアルタイムで対象を追跡するオンラインのタスクでは、現在までのフレームしか使えない。それに対して、全区間のフレームが利用可能なオフラインのタスクでは、ある時点からの未来にあたるフレームも利用することができる。対象が別の物体と重なるオクルージョンが発生すると対

象に同一の ID を振り続けられないため、対策が必要となる。さらに、対象の細かい状態や動作も追跡することが求められるのであれば、対象を矩形で検出するのではなく、セグメンテーションすることも必要となる。

物体追跡技術は、自動運転、スポーツ分析、監視カメラなど、さまざまな分野で応用されている。本研究では、バス停別乗降者数の推定と乗降区間別乗降者数の推定に向けて、乗車口と降車口のそれぞれで同一人物群の生成を目的としている。また、リアルタイムにバス停別乗降人数や乗降区間別乗降者数が必要とされてはいないため、オフラインで使用でき、追跡対象が数人規模のモデルを用いる。このような状況に適した物体追跡アルゴリズムを選択・適用することが、本研究の重要なポイントとなる。

2.3. メトリックラーニング

乗車人物と降車人物の同一判定においては、日々同様の乗客が乗り降りするとは限らない。よって機械学習アルゴリズムによって人物を認識するために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判定する必要がある。しかし CNN を使用したクラス分類は事前に学習したクラス以外の分類に対しての適用はできず、2つの入力データに対して類似度を計算する構造を持っていない。

このような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ある。メトリックラーニングで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行われ、これまでの研究において車両の認証 [14] や顔の認証 [15] 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研究されており [16]、本研究では、その中で優れた結果を残している BoT [17] という手法を使用する。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行う triplet loss を用いた手法 [18] があるが、BoT は triplet loss を用いたベースラインに、提案されている BNNeck に加え、学習率の Warmup, Random Erasing Augmentation [19], LabelSmoothing [20], そして Last Stride [21] の複数の学習トリックを加えること

で、精度の向上を図っている。学習トリックに含まれる Center Loss [22] 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いる。

2.4. 割当アルゴリズム

代表的な組み合わせ最適化問題の一つに割当問題がある。割当問題とは集合 A の要素を集合 B の要素のどれに割り当てるかを決定する問題である。割当問題はタスクによって、ボトルネック割当問題、一般化割当問題、2次割当問題など多くの種類に分類され、定式化の仕方や解法が異なる。

本研究は乗車人数と降車人数をシステムで推定するため、同一人物であってもオクルージョン等により途中で ID が切り替わる (ID スイッチ) ことで重複して数えることや、人物を追跡できず欠損するなど、乗車人数と降車人数が必ずしも一致するとは限らない。よって、M 人の乗車人物と N 人の降車人物の二部グラフと考えられる。M > N として、重複を許して M 人を N 人に割り当てて M ペアを作成する場合、ID スイッチが発生しても正しいペアを作成できる可能性があるが、特定の降車人物 1 人に多くの乗車人物が割り当てられてしまう可能性もある。重複を許さず N 人を M 人に割り当てて N ペアを作成する場合、乗降人物が欠損しても残りの人数分はペアを作成することができる。本研究では人数が少ない方のペアを作成することで、マッチング問題として定式化することができる。

重み付き二部グラフのマッチング問題の解法として、ハンガリアン法が存在する。計算量は $O(n^3)$ であるため、中規模の割当問題に対して効率的であるが、サイズが大きい問題には不適切である。本研究の対象はバス車内という限定的な空間であるため、十分に利用可能である。

3. バス乗客 OD 推定システム

本システムでは、バス停別乗降者数と乗車区間別乗降者数の推定を行う。システムの概要を図 1 示す。処理は大きく 2 段階に分かれる。1 段階目では乗降口における個人識別を行うことで各バス停で何人乗降したかを推定する。乗降口で撮影された動画を入力として人物検出と人物追跡を行うことで乗降する個人を軌跡として識別するとともに、バス停別乗降者数を出力する。2 段階目ではその乗車人物の軌跡と降車人物の軌跡の間で同一人物

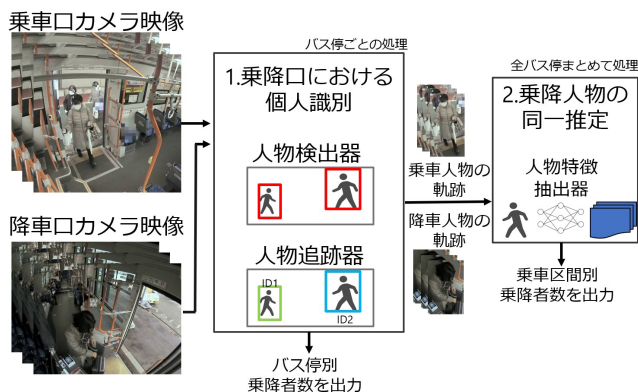


図 1. バス乗客 OD 推定システムの処理の流れ

の推定を行うことでどこで乗った人物がどこで降りたのかを推定する。メトリックラーニングを用いて各人物の軌跡を特徴空間に埋め込んで類似度を求め、割当アルゴリズムを用いて乗車・降車ペアの作成を行って乗車区間別乗降者数を出力する。これら 2 つの処理について以下で説明する。

3.1. 乗降口における個人識別

乗客が乗降する動画を入力としてバス停別乗降者数を出力することを目指す。候補となる要素技術には、顔認識、骨格推定、頭部検出、人物検出、人物追跡などが挙げられる。4 章でデータセットの詳細を説明するが、個人情報保護の観点から動画内の乗客の顔には全てモザイク処理が施されているため、顔認識により個人を識別することはできない。また、バス車内、特に乗降口付近は人物同士のオクルージョンが多く、カメラで体全体をとらえることができないため、骨格推定による個人識別も難しい。車内全体をカメラで撮影できれば頭部検出による車内の乗客数の変化から乗降客数を推定することも可能であるが、バス車内はカメラの設置位置に制限があり全体が撮影できなかったり手すりとのオクルージョンがあったりと、こちらも現実的でない。そこで本研究では人物検出と人物追跡を用いて乗降口における個人識別を行う。

乗降口における個人識別のアルゴリズムを図 2 に示す。入力となる動画に対してまず人物検出器を適用することで各フレームに写る乗客を矩形として推定する。次に推定された矩形情報を入力として、人物追跡器を適用することでフレームをまたいだ同一人物の矩形に対して同一の ID を付与することで人物の軌跡を出力する。最後に

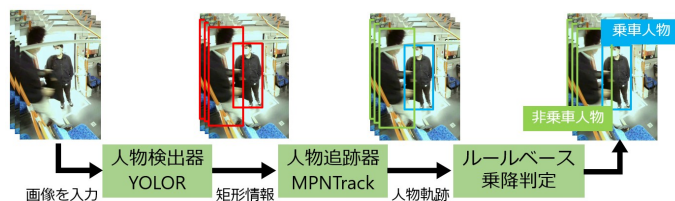


図 2. 乗降口における個人識別のアルゴリズム

出力された人物の軌跡が、乗車 (降車) してきた人物なのか元々バス車内にいた人物なのかをルールベースで判定する。

人物検出器は YOLOR[23] と YOLOv8[24] の精度を比較する。バス停別乗降者数の推定はオフラインで行われることを想定しているため、既存研究におけるオフラインの人物追跡器として MPNTrack[25] を用いる。MPNTrack は追跡途中で対象人物の軌跡が入れ替わってしまう ID スイッチの少ないモデルで、2020 年に行われたマルチオブジェクトトラッキングのベンチマークとなっている MOTChallenge¹ の MOT20 において最も ID スイッチの発生回数が少なかったモデルである。現在の研究において、用いられているモデルとしては Siamese CNN[26] をベースとしたモデルが挙げられるが、MPNTrack はマルチオブジェクトトラッキングの定式化に Message Passing Network [27] を応用したモデルである。

乗車口、降車口の画像例を図 3 に示す。乗降の判定に関しては、それぞれ以下の判定方法を適用する。乗車口の画像では、ドアが開く前から映っている人物は既にバスに乗っている乗客である。また、降車しようとするバス後方にある乗客はバス後方 (画像の左) から現れて、バス前方 (画像の右) に移動する。そのため、乗車の判定に関しては、ある乗客の軌跡が、i) ドアが開いた時間以降に現れている、ii) 最初のフレームの矩形の中心座標が乗車口の範囲に入っている、という 2 つの条件を満たしている場合に乗車と判定する。乗降口の範囲は図 4 のように画像の赤い矩形部分とする。

降車の判定に関しては、ある乗客の軌跡が、i) ドアが閉まる時間以前に軌跡が消失している、ii) 最後のフレームの矩形の中心座標が降車口の範囲に入っている、という 2 つの条件を満たしている場合に降車と判定する。

また、乗降者数推定の精度を下げる要因の 1 つとして

¹<https://motchallenge.net/>



図 3. 乗車口と降車口の画像例

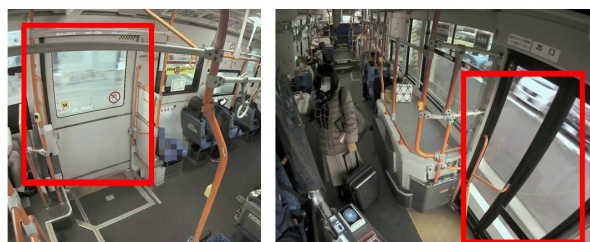


図 4. 乗降判定の範囲

ID スイッチがある。これを抑制するために人物検出によって出力される矩形に対していくつかの条件を設け、人物追跡のノイズとなる矩形を取り除く。

3.2. 乗降人物の同一推定

乗車区間別乗降者数の推定は、乗降口における個人識別によって作成された乗車人物の軌跡群と降車人物の軌跡群で同一人物のペアを推定することで各人物の乗車地点と降車地点を特定することで行う。

乗降人物の同一推定のアルゴリズムを図5に示す。乗車口における個人識別によって推定される乗車人数と降車人数には差異があるため、推定した乗車人物を M 人、推定した降車人物を N 人とする。ある乗車人物 A の軌跡が i フレーム、ある降車人物 B の軌跡が j フレームあるとき、 A と B に対してメトリックラーニングモデル BoT を適用することで $i \times j$ の特徴量距離行列を出力することができる。各軌跡は乗客の移動時間や人物検出・人物追跡の精度によって撮影されているフレーム数が数枚から数十枚と幅広いこと、撮影された人物の角度や明るさによって同一人物間でも特徴量距離に大きな差があることから、 $i \times j$ の特徴量距離行列から上位 5 つを抽出して平均したものを乗車人物 A と降車人物 B の特徴量距離とする。

この処理を乗車人物 M 人と降車人物 N 人に対して総当たりで実行して、 $M \times N$ の特徴量距離行列を求める。

最後に図6に示すように割当アルゴリズムを適用して乗車人物と降車人物のペアを推定する。乗車人物と降車人物のペアを出力する問題は二部グラフのマッチング問題と捉えられるため、乗車人物の軌跡と降車人物の軌跡の特徴量距離を割り当てコストとみなし、ハンガリアン法を適用する。ハンガリアン法の適用において、軌跡間の特徴量距離をユークリッド距離行列の行ごとに順位で置き換えた後にハンガリアン法を適用する手法が考えられる。本稿では、こうした手法をランキング法と呼び乗車区間別乗降者数の推定精度を比較する。

ここで、乗車したバス停より前に降車した人物とのペアが作成される可能性があり、実際にはあり得ないペアが出力されるという問題が発生しないように制約として、ハンガリアン法を適用する前に距離行列に対して簡単なバス停ルールを適用する。バス停ルールでは、乗車人物の軌跡群と降車人物の軌跡群のバス停の関係性を考慮し、乗車したバス停が降車したバス停と同じか降車したバス停より後の場合、該当する距離行列の要素を無限大に置換する。これにより組み合わせが最小となるように計算するハンガリアン法において実際に起こりえないペアの作成を防ぐ。出力されるペアを乗車・降車ペアとし、乗車・降車ペアの数は乗車人物の軌跡群と降車人物の軌跡群を比較したときに要素数が少ない方と等しくなる。

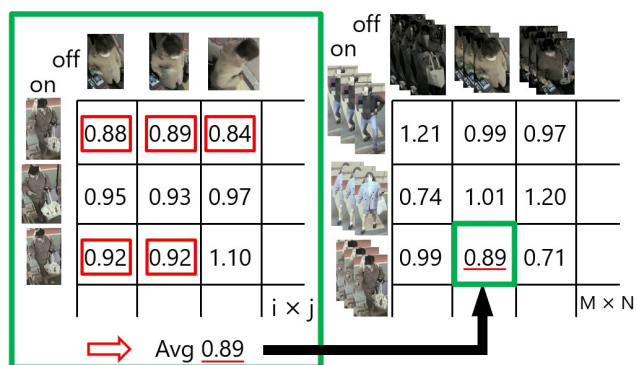


図 5. 特徴量距離行列の作成

4. データセット

本システムの検証のため、実際に札幌市内を走行する路線バスにカメラを取り付け、2021 年 12 月 5 日と 2021

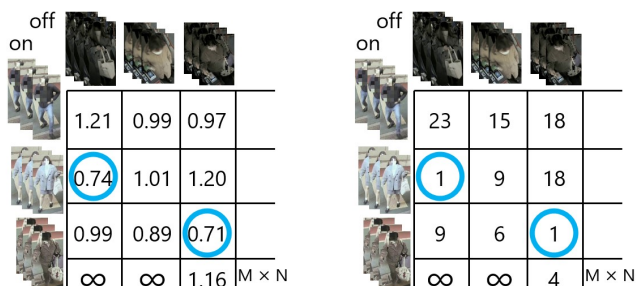


図 6. ハンガリアン法による割当 (左) とランキン
グ法による割当 (右)

年 12 月 6 日に乗客が乗降するデータを収集した。撮影したバスの路線は全部で 5 本あり、動画データをバス 1～5 とする。各データには乗車カメラの動画と降車カメラの動画が含まれ、停車したバス停数分の動画がある。データセットの詳細を表 1 に示す。バス 1 とバス 2 の動画は全て画像に切り出し、乗降人物に対して人物矩形のアノテーションを行った。バス 3～5 の動画は目視で乗降人数のみ計測した。

図 7 に示されるようにバス車内の乗車口・降車口にカメラを設置して、4fps で撮影した。乗客が簡単に触れることができない天井近くにカメラを設置した。撮影はドアが開く 10 秒前から撮影が開始され、ドアが閉まってから 10 秒後に撮影を終了している。個人情報の取り扱いの観点から、バスに動画を撮影していることを示す張り紙をし、動画に関しては撮影と同時に顔にモザイク処理を施してから保存している。

図 8, 9 に示される動画は実際に撮影した乗車口、降車口の映像である。解像度は乗車口で縦 882 ピクセル・横 886 ピクセル、降車口で縦 1,024 ピクセル・横 768 ピクセルとなっている。動画のフレーム数は 100～200 フレーム程度だが、最初のバス停の動画は発車時刻までドアを開けたままにして乗客が徐々に乗ってくるため長くなっている。

5. 実験と考察

5.1. 人物検出器の精度比較

人物検出器 YOLOv8 と YOLOR の精度比較実験を行う。どちらのモデルもオープンデータセット MSCOCO

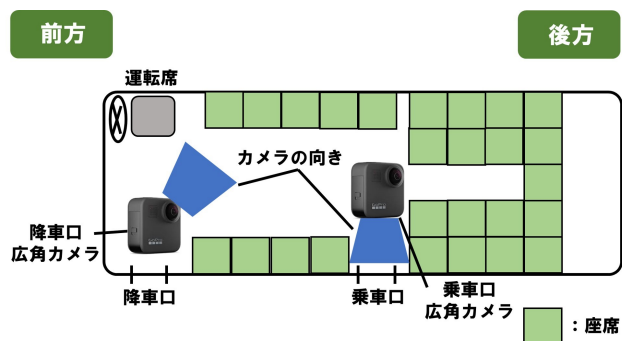


図 7. カメラの設置箇所

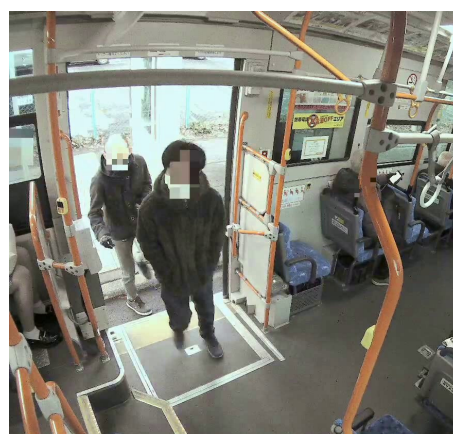


図 8. 乗車口カメラの画像

で学習させ、バス 1、バス 2 の動画に対して適用して人物検出の精度を比較する。評価指標は物体検出において最も一般的な指標である AP(Average Precision) を用いる。推論時の確信度の閾値を 0.2～0.6 の間で 0.1 ずつ変化させて AP を計測する。その結果を表 2 に示す。今回の実験では YOLOR を確信度 0.4 に設定した場合が AP 0.61 と最も高くなった。推定した矩形を確認すると、図 10 に示すように YOLOv8 はオクルージョンによって後ろに小さく写っている人物も検出できているのに対して、YOLOR はオクルージョンに弱い結果となっている。しかしアノテーションの段階で、体の一部が見切れている人物は軌跡を出力する際にノイズとなる恐れがあり、アノテーションを行わなかったため YOLOv8 の AP の方が低くなる結果となった。

表 1. データセットの詳細

	乗客の数	バス停の数	乗車動画の数	降車動画の数	アノテーション
バス 1	25	19	6	12	✓
バス 2	32	31	13	15	✓
バス 3	25	17	9	8	
バス 4	26	24	7	14	
バス 5	22	16	11	4	



図 9. 降車口カメラの画像

表 2. 人物検出器の平均適合率

Confidence	0.2	0.3	0.4	0.5	0.6
YOLOv8	0.60	0.60	0.52	0.58	0.55
YOLOR	0.48	0.55	0.61	0.58	0.54

5.2. バス停別乗降者数推定

乗降口における個人識別によるバス停別乗降者数の推定を行う。人物検出器については 5.1 の実験に基づいて YOLOR を確信度 0.4 で適用する。人物追跡器に関しては MPNTrack をバス 2 の動画で学習させて用いる。評価に関してはバス 1, バス 3, バス 4, バス 5 の動画を用いる。精度評価は以下の計算で求められる誤差率を用いる。誤差率は各バス停ごとに計算し、それを全バス停で平均した値を最終的な誤差率として計算する。

$$\text{誤差率} = \frac{|\text{実際の人数} - \text{推定人数}|}{\text{実際の人数}}$$

推定結果を表 3 に示す。乗車口では全体平均 13.3 %, 降車口では全体平均 5.24 % となった。乗車口では降りるために車内を移動する人物と乗車してくる人物との間にオクルージョンが発生し ID スイッチが起きやすくなっている。特に乗車人物が少ない時に ID スイッチが発生



図 10. YOLOv8(左) と YOLOR(右) の推定結果

して推定人数が増えると誤差率が極めて大きくなる傾向にある。

表 3. バス停別乗降者数の誤差率

	乗車口	降車口
バス 1	8.33%	8.33%
バス 3	12.5%	0.00%
バス 4	7.38%	10.0%
バス 5	25.0%	2.63%
平均	13.3%	5.24 %

5.3. 乗降区間別乗降者数推定

乗降口における個人識別において推定された乗車人物の軌跡群と降車人物の軌跡群を用いて乗降人物の同一推定を行うことで、乗車区間別乗降者数を推定する。メトリックラーニングモデル BoT は DukeMTMC-ReID で学習済みのモデルを使用する。ハンガリアン法で乗車・降車ペアを作成する場合、人数が少ない方と同数のペアが出力される。ID スイッチ等で乗車人数も降車人数も実際の人数と異なっている場合、出力されるペア数が実際の人数と異なる可能性がある。しかし、5.2 の実験から乗車人物の誤差率も降車人数の誤差率も十分小さいと

判断した。マッチングアルゴリズムには通常のハンガリアン法と、人物特徴量距離の値を行ごとの順位で置き換えた後にハンガリアン法を適用するランキング法とを比較する。テストデータにはアノテーションが付与されたバス1の動画を用いる。精度評価には以下の式で表される再現率、適合率、F 値を用いる。

$$\text{再現率} = \frac{\text{正解した乗車・降車ペア数}}{\text{実際の全乗車・降車ペア数}} \quad (1)$$

$$\text{適合率} = \frac{\text{正解した乗車・降車ペア数}}{\text{推定した全乗車・降車ペア数}} \quad (2)$$

$$\text{F 値} = \frac{2 * \text{再現率} * \text{適合率}}{\text{再現率} + \text{適合率}} \quad (3)$$

実験結果を表4に示す。ハンガリアン法では全乗降者25人中18人分の、ランキング法では14人分のODペア作成に成功した。推定した乗車人物が26人、降車人物が24人であったため、少ない方の24人分のODペアが出力され、適合率の分母にあたる推定した全乗車・降車ペア数は24となっている。今回の実験では通常のハンガリアン法がどの指標でも0.7を超える値を示すことができた。

また、精度を下げている原因として、乗車人物の候補がIDスイッチにより増加していること、降車人物の候補が人物追跡の段階で1人追跡できなかったことが第一に挙げられる。これは、バス停別乗降者数の推定精度を向上させる必要がある。次に、乗車人物と降車人物の映り方の差異が挙げられる。図11に示すように乗車口では自然光が多く入り比較的明るかったり白飛びした画像になったりする傾向にあるが、降車口ではバス中央の通路から料金箱付近では暗くなる傾向にある。こういった明るさの違いが乗車人物と降車人物との特徴量距離に影響を与えていることが考えられる。さらに図12に示すように、乗車口では乗車人物を正面で撮影した画像が多いのに対して、降車口では通路部分では正面、料金箱付近では頭上、ドア付近では後方から撮影され、体の角度も斜めになることがある。今回学習に用いたDukeMTMC-
ReIDは人物を正面から撮影した画像が多いため、タスクに適したデータセットを作成・学習させる必要がある。

表 4. 乗車区間別乗降者数の推定結果

	再現率	適合率	F 値
ハンガリアン法	0.72	0.75	0.74
ランキング法	0.56	0.58	0.57



図 11. 乗車口(左)と降車口(右)での明るさの違い

6. おわりに

人物検出器 YOLOR、人物追跡器 MPNTrack、メトリックラーニングモデル BoT、マッチングアルゴリズムハンガリアン法を組み合わせた路線バスにおける乗客OD推定手法を開発し、バス停別乗降者数と乗車区間別乗降者数の推定を行った。実データを用いた検証においては、バス停別乗降者数の推定では乗車口でも降車口でも誤差は1人ずつと十分な精度で推定できることを確認して、誤差の原因の分析とその対策を考察した。また、乗車区間別乗降者数の推定では再現率、適合率、F 値が全て0.7を超えることを確認して、精度を低下させている要因が乗車口と降車口で明るさや撮影角度の違いであることを確認した。今後は更なる精度向上に向けてバス車内という環境でのデータセット作成及びモデルの学習を行っていく。

参考文献

- [1] 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）



図 12. 乗車口 (左) と降車口 (右) での角度の違い

～, <https://wwwtb.mlit.go.jp/chubu/aichi/kikaku/seminafukumotosennsei.pdf> (2019).

- [2] 国土交通省国土交通政策研究所：地方公共交通事業における収益性と効率性に関する調査研究, <https://www.mlit.go.jp/pri/houkoku/gaiyou/kkk138.html> (2016).
- [3] Dalal, N. and Triggs, B.: Histograms of Oriented Gradients for Human Detection, *2005 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2005).
- [4] Krizhevsky, A., Sutskever, I. and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12*, p. 1097–1105 (2012).
- [5] He, K., Zhang, X., Ren, S. and Sun, J.: Deep Residual Learning for Image Recognition, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778 (2016).
- [6] Tan, M. and Le, Q.: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97, pp. 6105–6114 (2019).
- [7] Girshick, R., Donahue, J., Darrell, T. and Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation (2014).
- [8] Girshick, R.: Fast R-CNN, *CoRR*, Vol. abs/1504.08083 (2015).
- [9] Ren, S., He, K., Girshick, R. B. and Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks., *NIPS* (Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M. and Garnett, R., eds.), pp. 91–99 (2015).
- [10] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C.-Y. and Berg, A. C.: SSD: Single Shot MultiBox Detector., *ECCV (1)* (Leibe, B., Matas, J., Sebe, N. and Welling, M., eds.), Lecture Notes in Computer Science, Vol. 9905, Springer, pp. 21–37 (2016).
- [11] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection (2015). cite arxiv:1506.02640.
- [12] Wojke, N., Bewley, A. and Paulus, D.: Simple Online and Realtime Tracking with a Deep Association Metric (2017).
- [13] Santellani, E., Sormann, C., Rossi, M., Kuhn, A. and Fraundorfer, F.: MD-Net: Multi-Detector for Local Feature Extraction (2022).
- [14] He, S., Luo, H., Chen, W., Zhang, M., Zhang, Y., Wang, F., Li, H. and Jiang, W.: Multi-Domain Learning and Identity Mining for Vehicle Re-Identification, *Proc. CVPR Workshops* (2020).
- [15] Wang, H., Wang, Y., Zhou, Z., Ji, X., Gong, D., Zhou, J., Li, Z. and Liu, W.: CosFace: Large Margin Cosine Loss for Deep Face Recognition, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).
- [16] Wojke, N. and Bewley, A.: Deep Cosine Metric Learning for Person Re-identification, *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, pp. 748–756 (2018).

- [17] Luo, H., Gu, Y., Liao, X., Lai, S. and Jiang, W.: Bag of Tricks and a Strong Baseline for Deep Person Re-Identification, *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1487–1495 (2019).
- [18] Hermans, A., Beyer, L. and Leibe, B.: In Defense of the Triplet Loss for Person Re-Identification, *CoRR*, Vol. abs/1703.07737 (2017).
- [19] Zhong, Z., Zheng, L., Kang, G., Li, S. and Yang, Y.: Random Erasing Data Augmentation, *Proceedings of the AAAI Conference on Artificial Intelligence* (2020).
- [20] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z.: Rethinking the Inception Architecture for Computer Vision, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826 (2016).
- [21] Sun, Y., Zheng, L., Yang, Y., Tian, Q. and Wang, S.: Beyond Part Models: Person Retrieval with Refined Part Pooling (and A Strong Convolutional Baseline), *Proceedings of the European Conference on Computer Vision (ECCV)* (2018).
- [22] Wen, Y., Zhang, K., Li, Z. and Qiao, Y.: A Discriminative Feature Learning Approach for Deep Face Recognition, *Computer Vision – ECCV 2016* (Leibe, B., Matas, J., Sebe, N. and Welling, M., eds.), pp. 499–515 (2016).
- [23] Wang, C.-Y., Yeh, I.-H. and Liao, H.-Y. M.: You Only Learn One Representation: Unified Network for Multiple Tasks, *arXiv preprint arXiv:2105.04206* (2021).
- [24] Jocher, G., Chaurasia, A. and Qiu, J.: YOLO by Ultralytics (2023).
- [25] Brasó, G. and Leal-Taixé, L.: Learning a Neural Solver for Multiple Object Tracking, *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6247–6257 (2020).
- [26] Koch, G., Zemel, R. and Salakhutdinov, R.: Siamese Neural Networks for One-shot Image Recognition, Vol. 37 (2015).
- [27] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. and Dahl, G. E.: Neural Message Passing for Quantum Chemistry, *Proceedings of the 34th International Conference on Machine Learning* (Precup, D. and Teh, Y. W., eds.), Proceedings of Machine Learning Research, Vol. 70, PMLR, pp. 1263–1272 (2017).

「テストからはじめよ」 ～忍者式テスト 20 年の実践から～

深谷 美和
キヤノンメディカルシステムズ株式会社
miwa.fukaya@medical.canon

関 将俊
キヤノンメディカルシステムズ株式会社
masatoshi.seki@medical.canon

要旨

1999 年に出版された eXtreme Programming[1](以下 XP)に端を発したアジャイル開発は世界中で広く普及し、アジャイル開発の基礎となる反復開発、テスト駆動開発[2]も広く知られることとなった。

私たちのチームは X 線 CT 装置を XP で開発している。本稿では、私たちの 20 年の実践から、反復開発の利点と反復開発に有効なプラクティス「忍者式テスト」を説明する。

1. はじめに

Fig. 1 は Royce による大規模ソフトウェア開発のための実行ステップである。システム要求からコーディングまでの作り出すステップと、テストの確認するステップで構成される。

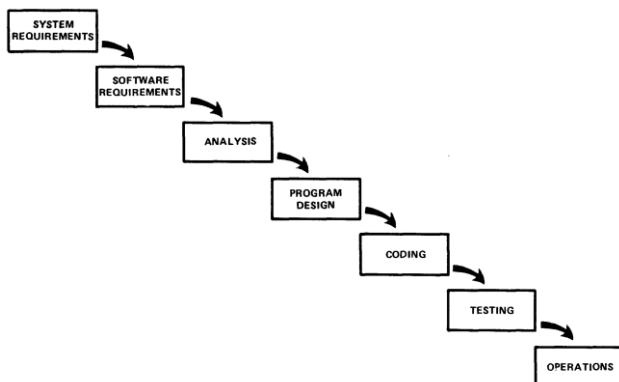


Fig. 1 Implementation steps to develop a large computer program for delivery to a customer. (参考文献[3] Figure 2.を引用)

Fig. 1 では確認するステップはテストのみである。テストを展開し、各ステップに対応するテストを配置すると V 字モデル (Fig. 2) となる。

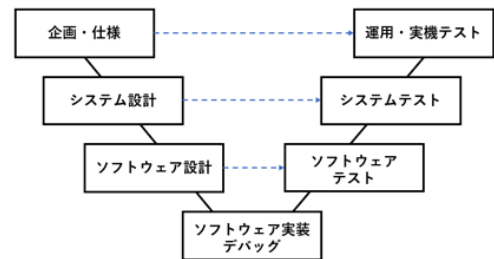


Fig. 2 V 字モデル (工程名は IPA[4]に合わせた)

実際のソフトウェア開発は各ステップを1回やって終わりではない。ステップは1つずつ進むが、問題が見つかり、n ステップ前に戻る (Fig. 3)。たとえば、ソフトウェアテストで問題が見つかり、ソフトウェア設計に戻った結果、さらにシステム設計まで戻るケースもある。

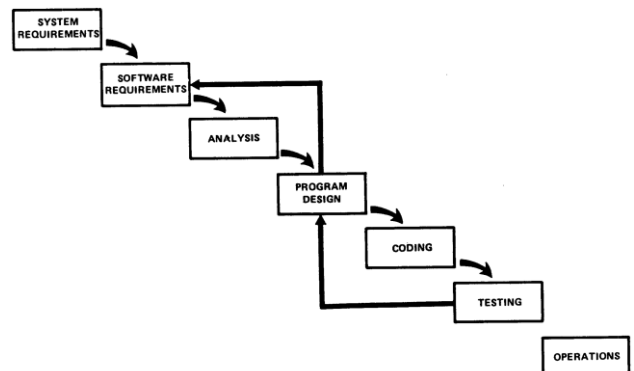


Fig. 3 Unfortunately, for the process illustrated, the design iterations are never confined to the successive steps. (参考文献[3] Figure 4.を引用)

このように作り出すステップで仮説を立て、確認するステップで検証する。これを繰り返すことで、開発は進んでいく。開発は単にモノを組み立てることではなく、製品を理解していく行為と言える。

これらのモデルは、各ステップの依存関係を示したものであるが、時系列のモデルと勘違いされることがある。たとえば、前のステップへ戻らないことを前提にしたスケジュールを立ててしまうケースである。「確認するステップ」で見つかる問題を修正し、それを確認する期間を確保していない。確保していたとしても「作り出すステップ」が遅延した場合のバッファと考えている。

1.1. ステップごとの問題発見能力の違い

ここで、IPA[4]を参照して、数字から分かる事実を二つ述べる。一つ目は不具合の原因の 73%が「作り出すステップ」にあるが、「確認するステップ」で 65%の不具合を発見している (Fig. 4)。不具合の原因工程と不具合の発見工程 (Fig. 5) から分かるように、企画・仕様、設計の問題をその工程で見つけるのは難しいのである。「作り出すステップ」がサボっているのではなく難しいという事実を関係者全員が真摯に受け止めなければならない。二つ目はテストは問題を見つける能力が高いということである。「作り出すステップ」で試さずに会議やレビューだけですべての問題を発見するのは高コストであり、試してみたほうが安いことを示している。これはソフトウェアの特徴のひとつである。

これを利用し「確認するステップ」をプロジェクトの初期の段階から実施すれば、早い時期に問題を発見する可能性が高まる。早い時期に問題を発見できれば修正コストも安くなる。

問題の発見が早ければ早いほどそれを修正できる可能性が高くなる。修正コストは開発の進捗に従って劇的に増加していく。統合とテストを最終段階で行うアプローチは、統合段階に到達しなければ実際の作業状況を認識できないという問題がある。
[7]

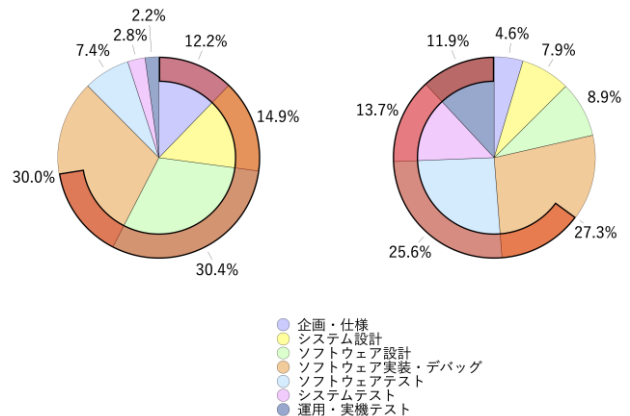


Fig. 4 不具合の原因工程の比率、発見工程の比率 (参考文献[4]の図 Q5-4I, Q5-4II を引用, 情報を追記)

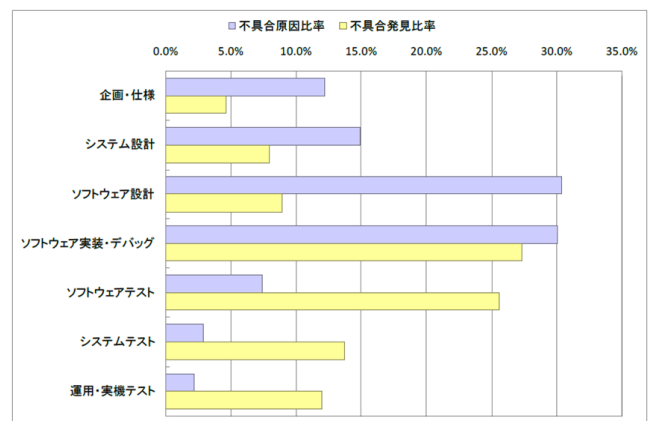


Fig. 5 不具合の原因工程と不具合の発見工程 (参考文献[4]の図 Q5-4 を引用)

2. 実施内容・方法

反復開発の考え方と、忍者式テストを中心とした反復開発の実践の様子を述べる。

2.1. 反復開発

V 字モデルの後半の「確認するステップ」をプロジェクトの早い時期から実施するため、プロジェクト全体を n 回の V 字で構成する。これを反復開発と呼ぶ。各 V 字ではすべてのステップを行う。

初回の V 字に注目するとプロジェクトの初期に「確認

するステップ」が実行されるのが分かる(Fig. 6). 次の V 字には前回の V 字で発見した問題や違和感, 知見をフィードバックできる. 初期の V 字で開発した機能は確実に搭載されるうえ, より長い期間, なんどもテストされることになる.

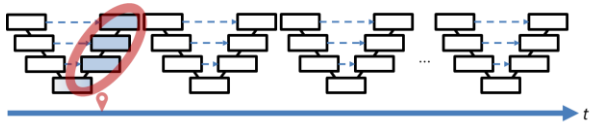


Fig. 6 反復開発

これを成果物の視点で言い換えると次のようになる. まず小さな動くシステムから始める(V.0.0). そのシステムになんらかの機能を追加し, 次の完結したシステム(V.0.1)を作る. これが 1 反復である. これを繰り返していく. レイヤーごとに部品をそろえて最後に結合するスタイルではなく, Fig. 7 のようにすべてのレイヤーを対象にその反復に必要な部分を変更して都度結合するスタイルである.

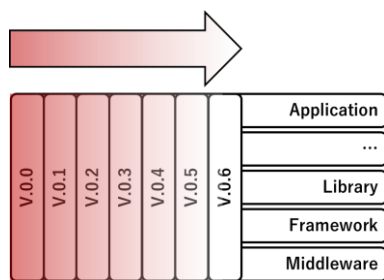


Fig. 7 作る順序(反復開発)

反復開発で反復ごとに結合してシステム全体を確認するのは, 製品のバージョンアップ開発のシステムテストと考え方は同じである. 新しい反復により改定された仕様の確認と, それまでに搭載したものが問題なく動作することを確認するテストが必須になる. Fig. 7 でいうと V.0.6 時点での確認は V.0.6 で変更した内容だけでなく, V.0.0 から V.0.5 で搭載したものが問題なく動くことの確認も必要である. 反復開発でも一般的なバージョンアップ開発と同様に, 反復ごとにシステム全体をカバーするテストが非常に重要であり, これはシステムを安全に保つための生命線である. 短い周期で反復すると, 頻繁にシステムテスト, 運用・実機テストを実施することになり負担が増えることが想像できる. 本稿で述べる「忍者式テスト」は, このテストを行う活動である.

次に忍者式テストと, これを行うために必要なプラクティスについて説明する.

2.2. 忍者式テスト

設計する段階ですべての問題を発見することは難しく, またテストは問題を発見する能力が高いことを説明した. XP においても「テスト」は主役であり, あらゆる場合に登場する.

エクストリームプログラミング[1] 第 5 章 原則に自己相似形についての説明がある.

先に失敗するテストを書いてから, それを動かすという開発の基本的なリズムがある. これはあらゆる規模で作用する.

あらゆる規模というのは, プログラミングの場面だけではなく, ひとかたまりの機能であったり, 大きな要求であったり, プロジェクトの進め方であったり, という意味である.

私たちのチームでは「うまくいったらどうなるの?」という問いかけからはじめる. うまくいったとしたらどうなるのか, それをどう試すのかを考えて, どうやら確かめることができそうだと, とわかってから実行する. 確かめかたを考えるだけでも多くの問題を見出すことができる. テスト駆動開発において, よいテストコードを考えることによる副次的な効果として, よいインターフェースが発見されたりすることよく似ている.

これから述べる忍者式テストは「テストファーストプログラミング」「テスト駆動開発」の相似形である. プログラミングの場面でうまくいくアイデアを, 要求・機能の場面へ拡張する.

エクストリームプログラミング[1]では多くの場面でテストが「たいくつなもの」「サボってしまうもの」「ストレスfulなもの」として登場し, プログラミングによって「自動化できるもの」と説明されている. なんとかしてプログラマーにテストを定着させようという努力が感じられる.

しかし私たちのチームのプログラマーにとって, テストはたいくつなものでもないし, サボるようなものでもないし, ストレスfulでもない. 自動化できない領域も積極的にテストしていきたいと考えている.

2.2.1. システムを変更する最小単位＝ストーリー

ここでは私たちのチームのストーリーについて説明する。

反復開発では、開発全体をひとつの大きなかたまりで考えるのではなく、既存システムへの小さな変更の連続として考える。小さな変更ごとに V 字の全ステップを行う。

この小さな変更をストーリーと呼び、それらをチケットで管理する。ストーリーは、システムがどのように変わったのかをユーザーが目で見えて分かるものでなければならない。つまり、システムテストで確認できることが求められる。これは「作り出すステップ」のフェーズから「確認するステップ」を考えるということである。このストーリーができるとユーザーは何が嬉しいのか、ユーザーが困っていることは解決できるのか、意図したとおりに作れたことはどう試すと分かるのか、といった議論からはじめる。ストーリーを考えると、私たちはテストから考えている。

XP と異なる点がある。XP ではストーリーという単位で計画し、ストーリーには複数のタスクが紐づいている。このタスクを変更の最小単位としている。タスクが完了してもストーリーが完了していない状態がある。一方、私たちはタスクを使用せず、ストーリーを変更の最小単位としている。システムテストで確認できるものだけを変更の単位と見なす。

2.2.2. ストーリーの大きさ

ストーリーの大きさ、粒度は、実際に開発しながら扱いやすい大きさに調整し続けていくものである。

私たちが扱うストーリーの多くは数日で完成する大きさである。小さな変更がシステムにどのような影響を及ぼすのかを議論しながら、V 字の各ステップを行き来して完成させる。この数日間は結合して試せない期間であり、開発のリスクとして全員が認識しておく。よって、ストーリーは小さいほうがよい。ただし、目で見えて変化の様子が分かる程度の大きさは必要である。

大きな要求をストーリーに分割し開発の順序を計画する。この分割や順序が適切でない場合は、ストーリーがなかなか完成しない、試し方が思いつかない、制約が多すぎて本来のストーリーのテーマが試せないなど、さまざまなシグナルがあがる。シグナルに気づいたら、そのつど計画の見直しや、ストーリーの再構築を行う。

2.2.3. チケットでストーリーとテストケースを紐づける

ストーリーはチケットで管理されている。チケットには一般的な項目（タイトル、概要、担当者、ステータスなど）の他に、テストケースを書く。これは、新しいストーリーを搭載したことによるシステムの具体的な変化と、その確認方法である。テストケースはストーリーと同一のチケットに書くため自動的に紐づく。

また、開発中に見つけたバグの修正もシステムへの変更と考え、ストーリーと同様に扱い、チケットで管理する。バグ管理システムで別管理ということはない。よって、ストーリーと同じように、バグを修正したことが分かる具体的な変化と確認方法をテストケースとしてチケットに書き、バグとテストケースを紐づける。

テストケースはストーリーを設計、実装したプログラマーが記載する。1 つのストーリーは複数のプログラマーで担当することが多く、テストケースの作成もその過程で行う。V 字の期間中にもたらされたさまざまな議論やテスターから示された具体的なシナリオなどもテストケース作成の入力になる。

テストケースを記載するタイミングはプログラマーに任せているが、いくらソースコードをコミットしていてもテストケースがないと完成とは見なさない。チケットには必ずテストケースをつける。

初回のシステムテストはテストケースを記載したプログラマー自身によって実施される。その後は、チームのプログラマーやテスターによる容赦ないテストが行われる。その際、システムの動作だけでなく、同時にテストケースの内容が適切かどうかテストする。テストケース自体も疑っているのである。

これらのチケットにはテストケースの他に、要求の背景や、日々のシステムの変化の様子（以降、開発日記）が書かれている。ToDo リストではなく、具体的な作業内容の記録であり、なぜそのような設計や実装にしたかの根拠、実験した内容や結果、懸念点や解決法やそれを選択した理由、プログラムコード片など多岐にわたる。開発日記は朝会で V 字の期間中、前回との差分（たいていは前日分）を全員で読み合わせている。その様子は設計レビューに近い。

2.2.4. システムがうまく動いている根拠

反復開発では、反復ごとにシステム全体をカバーするテストが非常に重要である。

今日のシステムはこれまで作ってきたすべてのストーリー

一とすべてのバグの修正からなっている。ということは、システム全体をカバーするテストとは、今日までに累積されたすべてのチケットに書かれたテストケースであると言える。

そこで私たちは毎日すべてのチケットをテストし直すことにした。すべてのチケットのテストケースをパスすることが、今日のシステムがうまく動いている根拠になるからである。毎日ストーリーは増え続け、試すべきテストケースも増えていく。しかしそれでも私たちはテストし続けるのである。

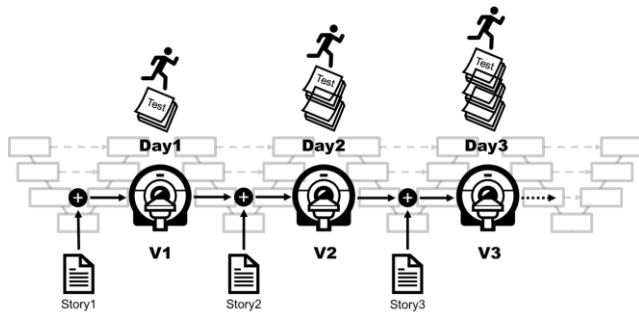


Fig. 8 忍者式テストの日々

忍者式という名前は、忍者が毎日成長する麻や竹の上を飛び越える修行に由来している。毎日増えるテストケースの束を毎日全部やり直す様子が似ているのである。

2.3. 手動テストを採用している理由

私たちのテストは、ストーリーに書かれているテーマを起点として「本当によいシステムになっているのか」について、本物のシステムを使って確かめる過程である。操作手順のような基本シーケンスだけでなく、チケットに書かれた開発日記（要求の背景、開発の経緯など）を読み直し、実際に今日のシステムで操作して、問題がないか、さらには、本当によいものになっているのかを実験する。単に不具合を見つけただけでなく、使いづらさや誤解をまねく仕様や手触りの良し悪しなども含めた理想との違いを知りたいのである。その理想さえも市場や環境の変化に応じて変わっていく可能性がある。

ストーリーに書いてあるテストを実施して分かるのは“テストケースに書かれている前提や操作において”うまく動くか、動かないかである。変更による意図しない副作用を検出するためのリグレッションテストとしての効果はある。しかし、私たちが手に入れた問題を発見するには、このテストケースだけでは足りない。そのために、書かれて

いるテストケースを出発点として、さらに新しいテストケースを考えながらテストしなくてはならない。これらを実現するには手動でのテストが適している。

2.4. 問題を見つけたらどうするか

理想の製品との違いを見つけたときにどうするかについて述べる。

仕様やふるまいについて、ちょっとした違和感や気になる挙動を見つけた場合は、良い仕様とはどのようなものかを議論する。必要と判断したらストーリーとしてチケットを発行する。

不具合の場合は見かけの現象によらず、原因の特定までを最優先で行う。システムにどのような影響があるかは発見された現象だけではわからない。いくら軽微な現象に見えても、重大な障害を引き起こす可能性のある実装になっている場合もあるからだ。原因を特定する過程で、どのチケットで問題が入ったのかを調べ、さらに既存のテストケースで検出できなかった理由も明らかにする。

テストケース自体の問題を発見する場合もある。テストケースの過不足の他に、ストーリーが増えることによって既存のテストケースが影響を受けるときがある。新しいストーリーが仕様を上書きし、操作手順などが変わった場合は、現在のシステムにあわせてテストケースを修正する。また上書きによってテストケースが不要になったら、その旨をチケットに記載し、テスト対象から外す。

毎日すべてのチケットをテストしながらテスト自体も見直し、テストケースを最新の状態にしている。

2.5. 規模に対する工夫

忍者式テストをはじめた 20 年前はチケットの数も少なく 1 日ですべてのテストケースを確かめることができていた。しかし、開発が進みチケットが増えるにつれテストケースも増加し、1 日で回せなくなってきた。そこで、一定期間でテストケースをすべて回す作戦に切り替えた。効果の高いテストケースを優先的に、かつ、一定の期間ですべてのテストケースを試せるアルゴリズムを開発した。まだテストされていない新しいストーリー、修正したばかりのチケットのテストケースを最優先とし、前回のテスト結果がパスしたテストケースの頻度を徐々に落とす、といった具合である。また、開発の状況に応じて機能ごとに出現頻度を調整している。

このアルゴリズムを用いると、新しい機能、問題のあつ

た機能を優先にしつつ、システム全体をある期間でテストすることができる。アルゴリズムにより抽出された今日のテストスイートを「本日のおすすめテスト」と呼んでいる。

Fig. 9 に、私たちの 20 年分の主たる開発におけるテスト実施の記録を元に作成した散布図を示す。

散布図の横軸は営業日、縦軸はチケットの番号である。ひとつひとつの点がテストの実施を示している。水色はテスト結果が OK、赤色は NG になったものである。なお、1 つのチケットには複数のテストケースが書かれているため、テストケースの数ではない。ここから読み取れることを、いくつか述べる。

- (1) 開発初期はすべてのチケットのテストを実施しているため色付けが濃い。開発が進みチケットが増えていくと色付けがだんだん薄くなる。全体を見ると、まんべんなくすべてのテストケースを実施している。(エラー! 参照元が見つかりません。)
- (2) 営業日と完成したチケットの数がリニアに増えている。システムが複雑になり、要求も高度になっているが、チケットの増え方は一定である。これは同じペースで開発していることを示している。XP の仮説のとおり、規模が大きくなっても変更コストは一定である。(エラー! 参照元が見つかりません。)
- (3) グラフの稜線のあたりに赤色が散見される。これは新しいチケットがしばしば NG になっていることを示している。作る前のレビュー、V 字の期間中に議論を重ねてもなお、実際に触って分かることがある事実と、さらに、テストも手を緩めず実施していることを示している。(エラー! 参照元が見つかりません。)
- (4) 3800 日目くらいから下半分に薄い部分が増えるが、次世代の製品にシフトした時期と一致している。事業計画に従って、テストの注力点を調整しているのが履歴からも見て取れる。その後、徐々に全体を試すように回復している。(エラー! 参照元が見つかりません。)

20 年前に作成したテストケースを読み、開発日記で当時どのようなことが起きたのかを知り、今日のシステムで試すことは私たちの日常である。これはまた自分が書いた開発日記やテストケースが後世に渡って何度も読まれ、試され続けるものであると実感する瞬間でもある。

2.6. プログラマーによるシステムテスト

私たちのチームではプログラマーに毎日1時間、システムテストの時間を割り当てている。前述した「本日のおすすめテスト」にしたがって、最新のシステムでテストする。

問題を発見するという本来の効果の他に、副次的な利点がある。それは、システムを学習または再学習し、製品を知り、理解を深められる点である。

私たちの製品は開発規模が大きい。製品について知っていることもあれば知らないこともある。プログラマーによるシステムテストは、知らないことを学習する機会になるし、知っていることを再学習する機会になる。過去の開発は記憶が薄れがちである。それをはっきりと思い出し、自身の記憶を再構築するのである。

実際にテストしている様子を観察してみると、単にテストケースを実行するだけでなく、開発日記を読んでいる。これは、対象のストーリーがどのような意図、経緯で作成したのか、当時の開発で起きていたことを知ろうとしている。バグ由来のテストケースの場合は、過去にどのような経緯でバグが混入し、その原因と対処を読み取るであろう。

このような学習を毎日繰り返すことで、製品やシステムに対する理解や知識を積み重ねている。テストケースに書かれていないことを試せる、そして、未知の問題を見つけられるのは、これらの学習の成果だと思う。この成果は製品開発のすべてのフェーズに活かされているだろうと想像する。

2.7. 反復開発と規模と組織

反復開発を実施してきた経験からさまざまなことに気づいた。ここでは、規模と組織に関するテーマを挙げ、それらについての考え方を述べる。

2.7.1. スケールアップと実際のヒント

反復開発の V 字が十分小さくなると、二つの点から、スケールしやすくなる。なお、ここでのスケールとは増員分だけ規模が増やせることである。ひとつは、時期による負荷の偏りが少ない点、もうひとつは1チームで複数バージョンの並行開発が可能な点である。

反復開発ではプロジェクトの期間中ずっと各ステップの作業が発生する。そのため、プロジェクトの前半だけ忙しい、あるいは終盤だけ仕事が発生するという人員はい

なくなり、負荷の偏りがなくなる。ステップごとの職種、たとえば、要求だけを考える人であっても、毎日出来上がってくる製品が期待どおりか確かめなくてはならず、毎日新しい要求について考える必要がある。また開発領域についても同様で、事前に大きな部品だけを作る、ということではなく、その V 字に必要な部分だけ作り、結合して確認するようになるため、プロジェクト期間中でロードの偏りが発生しない。つまり増員分だけこなせる規模が増やせるのである。ただし「要求する人＝要求を確認する人」の増員を怠ると、「作る係」をいくら増やしても開発規模を増やせる効果はない。要求の確認がボトルネックとなるからである。

システムはストーリーの集まりに分解され、ひとつのキューで管理される。チームは同時に数ストーリーを開発することができ、その様子はまるでマルチスレッドでひとつのキューのジョブを処理していくかのようなのである。これを拡張して、ひとつのキューに複数バージョンのストーリーを積むことで、自然とチームは複数バージョンに対応できるようになる。それぞれの状況に応じて投入するメンバーを調整できるうえ、バージョン間での問題の共有が行える。メンバーをバージョンに固定する方式では、バージョンのリーダーがメンバーを独占して使いたがるため、相互の協力が難しくなる。

私たちのチームはこの複数バージョンを開発できる方式で、複数の製品の並行開発を行っている。まったく別の製品であっても1つのチームで扱うことが可能なのである。

2.7.2. サブシステム別チーム

大きなシステムでは複数のサブシステムから構成され、それぞれにチームを割り当てて開発を行うことが多い。その場合、チーム間の開発ペースの違いにより衝突が起こりがちである。隣のチームのリリースが遅い、質問の返事がこないなどといった不満がたまる。これは、次のように考えるとよい。

1チームの開発でも同時に複数のストーリーを開発している。完成するタイミングはそれぞれ異なるが、そのタイミングをイテレーションの区切りなどに合わせていない。忍者式テストによるシステムテストによって、毎日完成を確認するからだ。なるべく長い期間テストできるように早めにコミットしたほうが喜ばれる。このようにチームの中でもペースの違いは発生し、それは許容されている。

この構造はチームとチームの間でも同様である。異なるペースを混ぜても、結合された状態で確認するという原則

則を守っていれば、1チームの開発とそれほど変わらない。他のチームからの変更が届くのが、自分たちの予想よりも早かったり遅かったりするかもしれないが、自チームのメンバーの作業が遅延するのと同様である。

イテレーションの切れ目にみんなの完了がきれいに揃うことは重要でないで、隣のチームのペースを無理に変える必要はない。タイミングを合わせると効率がよい場面では自分たちが合わせればよい。

となりのチームは遅いのだろうか。「遅い」のではなく、実はタイミングが合わないことへの不満なのではないか。となりのチームもサボっていないのでイライラしない。となりのチームの人たちも同じチームの人たちのように考えよう。相手がなぜその時に出せないのか、相手の都合を聞いてみよう。

さて、あなたのチームはどこまでか。同じチームの人のようにとなりのチームに気をかけていると、果たしてどこから違うチームなのかわからなくなる。チームの境界が曖昧になることは悪いことではない。自チームだけの成功は製品としては失敗なのである。

2.7.3. 工程別／ロール別のチーム

工程別／ロール別にチームを作るケースもあるだろう。たとえば、設計するチームと実装するチーム、開発するチームとテストチームといった具合である。その場合、お互いの領域に踏み込めない雰囲気になりがちである。前後の工程のチームに対して遠慮したり、不信心を持ったりしてしまう。自分たちはうまくできているが前後の工程に問題がある、といった対立構造が生まれやすい。

自分の工程の良し悪しは製品で確認するとよい。どの工程を担当していても、良い製品ができたかが根拠となる。前後の工程のチームがうまくいくように気をつかうべきである。あなたの工程だけが成功しているという状況はありえないのである。

3. 結果・考察

忍者式テストを始めて最初に起きたことは、難しそうな問題も躊躇せずに修正できるようになったことである。不具合や性能などの実装レベルの問題に積極的に対応できるようになった。次に起きたのは、直せる幅が広がったことだ。理想の製品をイメージして、それとの差分も積極的に探しだす者が現れた。数年後には全員がずっと「良い製品とは何か」を問いながら開発できるようになった。こ

れが忍者式テストの一番大きな効果である。

忍者式テストの副次的な効果として学習機会の創出がある。毎日のテスト時間、朝会のチケットの読み合わせなどさまざまな場面で、製品の仕様、設計、実装など、製品に関する質疑応答が絶えず行われる。これは圧倒的な量のドリル学習に相当する。これにより、ひとりひとりの脳内にシステムのあらゆることが配線され、製品を動かしたときに瞬時におかしさに気づくようなことが起きる。誰かのちょっとした疑問が別の誰かの思考を刺激する。チームはひとつの巨大な脳を持った生き物のようになる。

4. おわりに

事前に分かることと試してみても分かることがある。私たちはこの事実に真摯に向き合っている。あらゆる場面で「うまくいったらどうなるの?」という問いかけからはじめる。うまくいった状態がイメージできると、期待通りでないことを認識できる。認識できれば修正できる。これが私たちのあらゆる行動の元となるカタチだ。問題が見つかること皆で喜び、見つけた人は感謝される。早く失敗して良い製品に仕上げていくことに全員の目が向いている。

本稿では、反復開発の利点とプラクティス「忍者式テスト」を説明した。忍者式テストは XP を拡張するプラクティスである。XP はその名前から受ける印象とは裏腹にビジネスとテクノロジーの調和を述べているが、忍者式テストも同様にテストを起点とした開発全体の有り様を語っているのである。

テストからはじめよ。

参考文献

- [1] Kent Beck, Cynthia Andres, 角 征典[訳]. エクストリームプログラミング: オーム社, 2015
- [2] Kent Beck, 和田 卓人[訳]. テスト駆動開発: オーム社, 2017
- [3] Dr. Winston W. Royce: Managing the Development of Large Software Systems: 1970
- [4] 独立行政法人 情報処理推進機構. 2012 年度「ソフトウェア産業の実態把握に関する調査」: 2013
- [5] Glenford J. Myers, Tom Badgett, Todd M. Thomas, Corey Sandler, 長尾 真[監訳], 松尾 正信[訳]. ソフトウェア・テストの技法 第2版: 近代科学社, 2006
- [6] Tom DeMarco, Timothy Lister, 伊豆原 弓[訳]. 熊とワルツを - リスクを愉しむプロジェクト管理: 日経 BP 社, 2003
- [7] 落水浩一郎, "WATERFALL Model 再考 ソフトウェア開発管理とソフトウェア開発方法論の融合について", SEAMAIL. 2006, Vol.14, No.9-10. 岸田孝一編. ソフトウェア技術者協会.
- [8] 中島滋, "Ninja Testing at Toteka03", slideshare, 2014-10-04, <https://www.slideshare.net/ledsun1/03-39903732>.

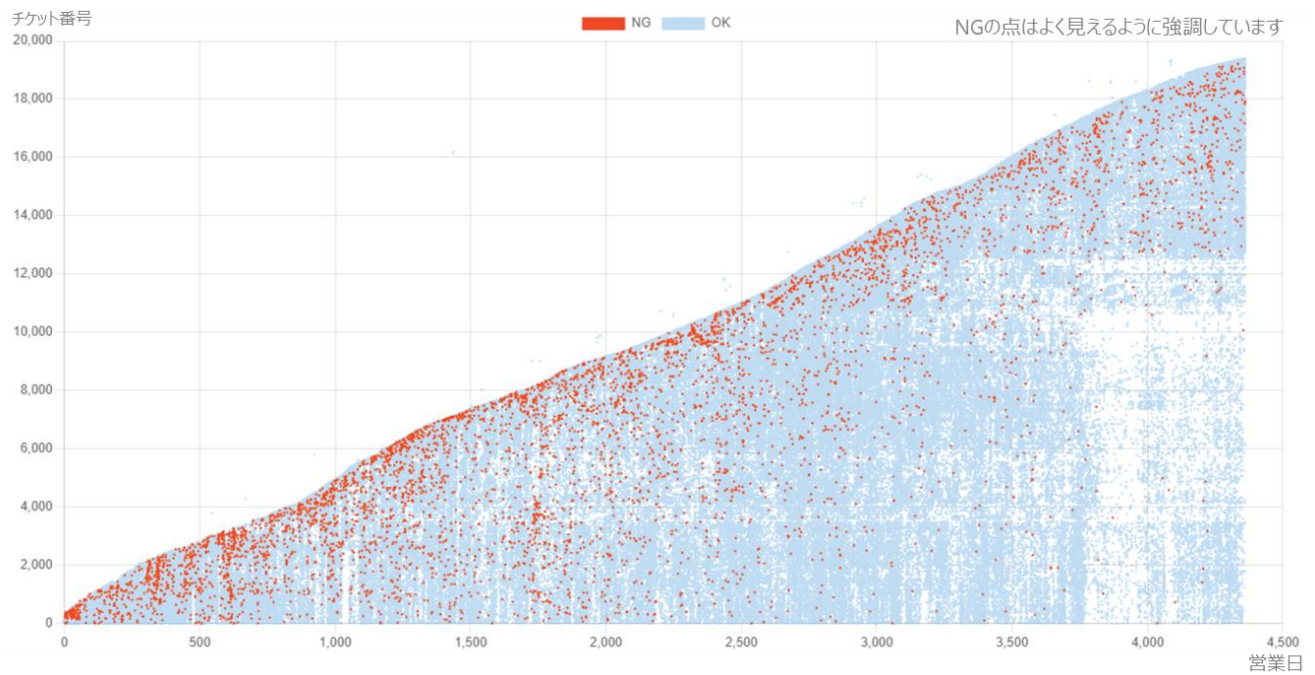


Fig. 9 テスト実施の記録(20 年分)

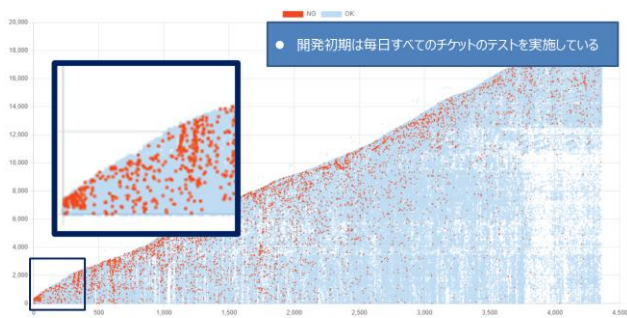


Fig. 10

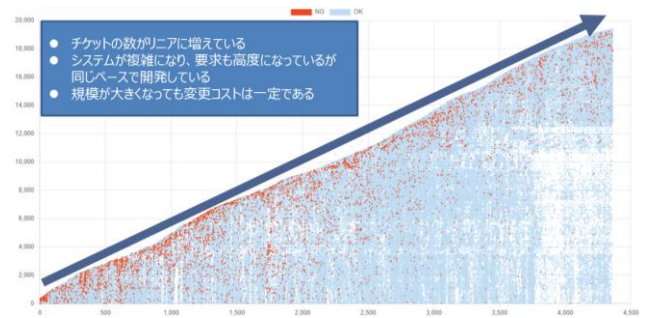


Fig. 11

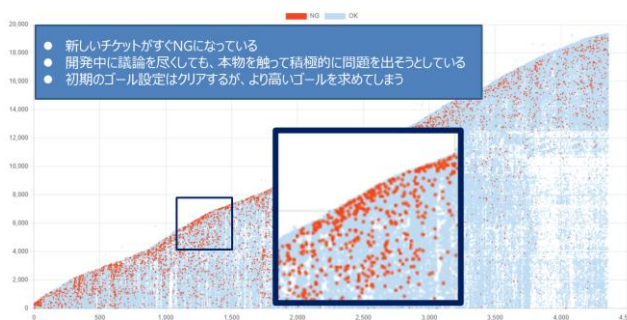


Fig. 12

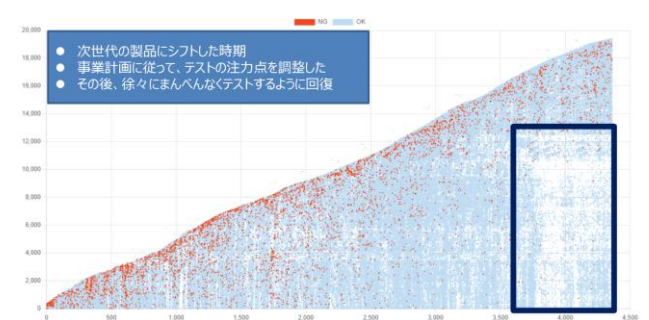


Fig. 13

「ソフトウェアレビュー研究結果の認知拡大と適用促進」活動状況報告

安達 賢二
株式会社 HBA
adachi@hba.co.jp

中谷 一樹
TIS 株式会社
nakatani.kazuki@tis.co.jp

上田 裕之
株式会社 DTS インサイト
hiroyuki.ueda@dts-insight.co.jp

要旨

ソフトウェアシンポジウム 2022 にて経験論文として発表した「ソフトウェアレビュー研究結果の認知拡大と適用促進」で計画していた各種活動に対する状況を報告し、今後の対応を述べる。

1. はじめに

日本科学技術連盟ソフトウェア品質管理研究会レビュー分科会(以降,SQiP 研究会とする)では 10 年ほどソフトウェアレビュー研究が実施され,その成果が 30 件の論文として発表されている。(2022 年 3 月現在)
しかし,それらの研究成果は業界内に広く認知されている状態ではなく,適用も進んでいないと感じていた。
その状況打開のための方策を検討した結果を「ソフトウェアレビュー研究結果の認知拡大と適用促進」としてソフトウェアシンポジウム 2022 にて経験論文として発表した。
当報告では,その後実施したソフトウェアレビュー研究結果の認知拡大と適用促進活動状況と現状の課題,今後の対応について述べる。

2. 当初計画(元論文の要旨)

当報告における活動実績と評価の対象となる活動計画～ソフトウェアシンポジウム 2022 にて経験論文として発表した「ソフトウェアレビュー研究結果の認知拡大と適用促進」の内容を要約して述べる。(個別詳細は当該論文を参照のこと)

2.1. 解決したい課題

日本科学技術連盟ソフトウェア品質管理研究会レビュー分科会では 10 年ほどソフトウェアレビュー研究が実施され,その成果が 30 件の論文として発表されている。(2022 年 3 月現在)
しかし,「それら研究成果は業界内に広く認知されている状態とは言えず,適用もあまり進んでいない」と感じていた。これを解決したい課題とした。

2.2. 課題を引き起こしている要因

解決したい課題を引き起こしている主な要因は以下と想定した。

・要因 1: 研究成果物が認知されにくい

SQiP 研究会と研究成果物の掲載場所を認識している人,その内容を意図的に確認する人はそう多くないと思われる。

・要因 2: レビューの課題が特定できていない

要因 1 を解決できたとしても,現状のレビューに様々な問題が存在している中で,どれが取り組むべき課題なのかまで認識している人は多くない。

・要因 3: レビュー課題の解決手段の引当てが難しい

要因 2 を解決できたとしても,多くの研究成果からその課題を解決する手段を引当てるのは簡単ではない。

・要因 4: 研究結果はそのままの適用が難しい

要因 3 を解決できたとしても,実務に研究内容をそのまま適用できることは稀である。自らの状況への読み替えやカスタマイズを行う必要があり,それは簡単ではない。

以上の要因は,認知の問題(要因 1)と適用の問題(要因 2,3,4)に分類することができ,要因 1 から順に解決する必要がある。

2.3. ターゲットユーザーの特定

方策を展開するために活動できるメンバーは 3 名で,組織もロケーションも異なり,実務の空き時間で当活動を行う必要があるため使えるリソースには限りがある。レビューに興味のない人,問題意識のない人たちを相手にしても貴重なリソースを無駄に浪費し,効果も期待できない。

この状態において方策による効果を可能な限り高めるため,どのようなユーザーをターゲットにすると効果的なのかをあらかじめ特定した。

ターゲットユーザーは,

「現状のレビューに問題意識を持ち,改善したいと思っている,しかし,改善に効果が期待できるレビュー手法を把握できずにいる人」

とした。

2.4. 認知拡大と適用促進に対する方策

課題を引き起こしている要因とターゲットユーザーを考慮したうえで、認知拡大と適用促進のための必要な方策を明確にした。その結果を表 1 に示す。

表 1. 認知拡大と適用促進に対する方策

ID	方策の内容
認知拡大のために	
H1	ターゲットユーザーの目に触れやすい関連イベントや一般公開型勉強会など、SQiP 研究会の枠を超えたパブリックな環境で自ら働きかけて情報発信する能動的なアプローチを採る。 →ソフトウェア品質やレビュー、テスト関連のイベントや勉強会、コミュニティ活動等が有力な候補 当初は投稿募集中であったソフトウェアテストシンポジウム 2022 東京をその最初の舞台としてワーク実施を投稿し、採録された。
H2	単発の情報発信では認知を広げる効果が期待できないため継続的な情報発信を行う。 →一か月～数か月に 1 度程度の継続した実践に加え、無償で参加できる等の気軽さも必要である。今回は、Web 上の IT 勉強会支援プラットフォームサービスに「レビュー勉強会」グループを立ち上げ、勉強会を継続運営することとした。
適用促進のために	
H3	ターゲットユーザーが持つレビューの問題・課題にヒットしやすくするため、「レビューの典型的な負の事象群」を特定し、整理する。 →図 1：レビューで発生しがちな負の事象の連鎖と循環
H4	「レビューの典型的な負の事象群」を解決するための研究結果を選定し、その構成要素から部品化された解決手段を作成する。 →図 2：方策による負の事象への働きかけ +表 2：レビュー観点導出技法の実装（部品化）
H5	必要な技法・アプローチを確実に体得するための継続的な取り組みの場を設ける。 →自らのレビューの問題・課題の解決手段を効果的に体得するアプローチが不可欠。 受講者が自らレビューの問題解決を実践し、結果をアウトプットする「ワーク形式」で提供する。

図 2 の下線部に示された方策記述を具体化するため、表 2 にレビュー観点導出技法として部品化した。

ここでレビュー観点とは、レビューアによるレビュー対象の見方を表している。つまりレビューで検出したい欠陥を見つけるためにレビューアが集中して着目する、対象成果物の側面を意味している。レビューアがどのように確認するのかを表したものである、と考えてもよい。レビュー観点は段階的に詳細化したレビュー目的にも相当し、「利用者課題の解決可能性」のように抽象度が大きい高位レベル観点から、「高齢者が一読で認識できる文字の大きさであること」のように具体的な低位レベル観点まで、さまざまな粒度の観点がある。

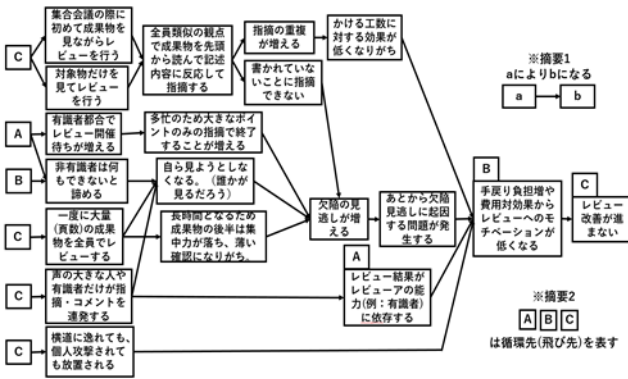


図 1. レビューで発生しがちな負の事象の連鎖と循環

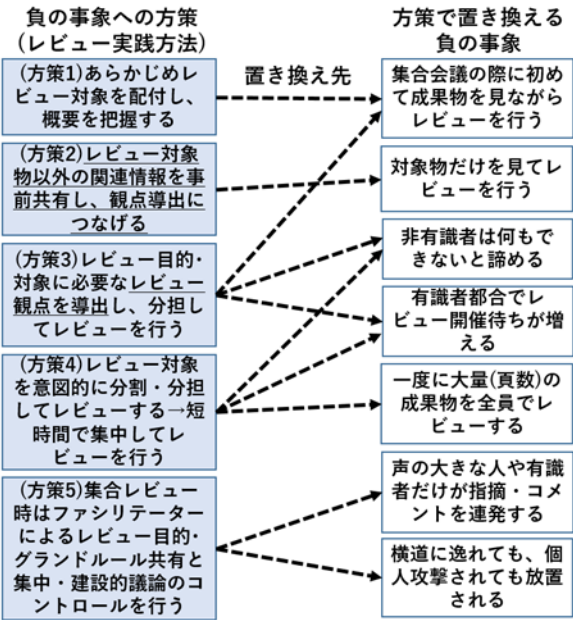


図 2. 方策による負の事象への働きかけ

表 2. 構築したレビュー観点導出技法 (部品化)

ID	技法の概要
M1	作成者コンテキストからレビュー観点設定
M2	利害関係者の関心事からレビュー観点設定
M3	プロダクトリスクからレビュー観点設定
M4	対象成果物に求められる事項から確認方法設定
M5	対象成果物の典型的な欠陥から確認方法設定
M6	利用シナリオ+状態遷移図に沿ってレビュー実施

2.5. 方策実施に向けた実践体系とワーク実施環境構築

表 1 の H1,H2,H5 を実施するために,さらに H4 の表 2 を包含した“レビュー実践体系”を構築した.

その全体像を図 3 に,そしてレビューワーク,勉強会を実施するために構築した Miro (オンラインホワイトボード) 環境を図 4 に示す.

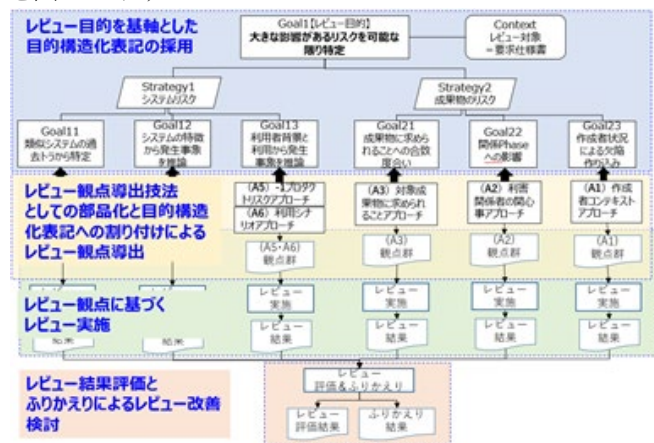


図 3.当初構築したレビュー実践体系

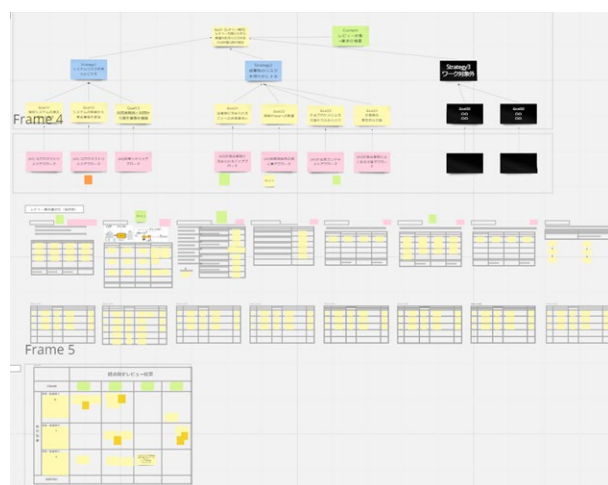


図 4.当初構築したレビューワーク環境

3. この活動の評価指標

この活動の目的は,レビュー研究結果の認知拡大,そして実務適用の促進である.よって活動の評価指標は表 3 とした.

表 3.活動の評価方法

目的	達成要件 or 達成内容	指標	判断方法
認知拡大	活動を実施・継続している	活動継続度 →3.1 参照	期間中の平均 1 回/月以上
	レビュー実践体系を認知済の人が多くなる	活動への参加者数 →3.2 参照	認知した人数が当初参加者数より増える
適用促進	より内容が理解しやすい	レビュー実践体系への理解度 →3.3 参照	受講後評価: 理解度が 8 割以上
	現状のレビュー課題の解決に役立つ	レビュー実践体系が実務へ有効であると感じた度合い →3.3 参照	受講後評価: 実務への有効性が 8 割以上
		レビュー指摘の効果向上 →3.4 参照	全指摘数に対して効果大・中の指摘数が占める割合が高まる
	より適切な解を取得するために選択肢が増える	レビュー実践体系への研究成果の組込み度 →3.5 参照	研究結果の組込み件数が当初の 6 件より増える
	活動が継続している	活動継続度 →3.1 参照	期間中の平均 1 回/月以上

3.1. 活動継続度

レビュー実践体系を知ってもらうには,一度だけではなく何度も知ってもらう機会を提供する必要がある.

また,レビュー実践体系には,レビュー目的を段階的に詳細化するレビュー観点導出など,汎化・具体化を都度適切に行き来することが必要な側面が多いため,一度の経

験では習得できない内容が含まれている。そのため適用を促進するためには継続して体験できる機会を提供することが重要である。

限られたリソースの範疇で、可能な限りトレーニングと次トレーニングの間隔を小さくし、継続実践できる運営ができているかを評価する。

対象期間中に平均で1回/月以上活動を実施していれば適切に活動が継続していると判断する。

3.2. 当活動への参加者数

当活動は認知拡大を目指すものであり、活動への参加人数が多いほど認知が拡大したとみなすことができる。

当初参加者数より新たに認知した人数が増えているかで評価するが、同一人物が何度も参加した場合をカウントしないようにする。

3.3. レビュー実践体系への理解度と実務有効度

当活動では、レビュー研究結果をレビュー実践体系上にマッピングして整備し、それを実践できるきっかけや継続的に学び体得する機会を提供する。これらの実務適用を促進する上で重要になるのは活動参加時にどれだけレビュー実践体系に対する理解度を上げ、実務に有効であると実感してもらえるかである。

活動への参加者がどの程度理解できたと感じたか、実務に有効であると実感したかを評価してもらい、その結果が8割以上であるか判断する。

なお、判断基準については、プロセス評価の国際標準において“完全に”が86~100%、“概ね”が51~85%を参考に、全体の8割は高水準であると考えられることから受講者評価結果が“8割以上”とした。

ただし、一開催、一側面のみでの評価結果ではその場限りの偶然性を排除できない可能性もある。よって、複数回開催において複数の側面（理解度と実務への有効性、および満足度）での評価結果を確認し、3つの側面でその平均が8割以上であれば、理解度・実務への有効性の評価結果が有効であるとみなす。

3.4. レビュー指摘の効果

レビューの主な目的が欠陥検出であることから、レビュー実践体系に基づきレビューを実践した結果、検出した指摘事項の効果が適用前より向上していることが期待される。ここでの“効果”とは、その欠陥が見逃された場合に以降のフェーズで発生する被害（手戻り工数や利用者への悪影響）の大きさを指す。

2010年~2015年にのべ4社20チーム(1チーム3~5名)に対して対象成果物を一斉に手渡し、その場で15分間レビューを実施するアドホックレビューを実施した結果、全検出欠陥数に対して効果が小さい(軽微な)欠陥数の占める割合が大きくなることを確認している。

レビュー指摘例と効果大、中、小の判断事例を示し、各チームで検出した指摘事項がどの効果レベルかを1件ずつ割り付けた結果、効果大/中/小それぞれの全欠陥数に占める割合は19%/44%/37%(20チーム平均)となった。

今回実施したワークにおいてもレビュー実施時間は15分としている。よって今回はこの効果大中小の割合数値を比較対象のベースラインとし、レビュー実践体系に基づくレビューで検出した指摘事項の効果レベル割合が大・中側にシフトしていればレビュー指摘の効果が向上しているとみなす。

なお、活動Bについては、各開催回にてレビュー実践体系内の個別手法(例:レビュー結果評価のみ)をトライアルする内容となっているため当指標による評価の対象外とする。

3.5. レビュー実践体系への研究成果の組込み度

2022年2月の活動開始時点のレビュー実践体系は、実践に必要な枠組みと必要最小限の実践選択肢を取り揃えたものである。当初の活動検討時に存在したレビュー研究成果物30項目のうち、レビュー実践体系として組み込んだのは6項目に過ぎない状態であった。

よって、継続的な活動を通じて残り24項目を解析してレビュー実践体系に徐々に組み込み、さまざまなレビューの状況に対応可能な選択肢を充実させていく必要がある。評価指標は「レビュー実践体系に新たに組み込んだ／内容を見直したレビュー研究成果物数」とする。この数値が0より高いほど活動が活性化していると判断する。

4. 活動実績

当初論文作成時以降の活動内容は、以下の3つのカテゴリに分かれている。これらの実施実績を表4に示す。

A. 関連イベントでのワークショップの実施

2.5時間でレビュー実践体系を通して体験するワークショップ

B. 一般公開型レビュー勉強会の継続開催

レビュー実践体系の構成要素の一部や、新しくレビュー実践体系に組み込む候補である研究成果を試行体験するワーク

C. 特定企業 Z 社向けレビューワーク(トレーニング)の

継続実施

3時間で活動Aと同様レビュー実践体系を通して
体験するワークショップ

なお,SS2022にて発表した経験論文「ソフトウェアレビュー研究結果の認知拡大と適用促進」は,2022.3に実施したJaSST'22東京におけるレビューワークショップの内容を含んでいるため,これを原点となる活動として0回目としている.

当報告は,その後の活動(2022年4月以降/1回目〜)についての事例報告となっている.

また,当報告作成時(2023.3.17)は,18回目のレビュー勉強会への参加者募集中のため参加者数を“―”とした.

表 4.レビュー研究結果の認知拡大と適用促進活動実績

※注意:活動A,Bの参加者数(参考値)には同一人物の重複が含まれる

回	開催日	テーマ	活動	参加者数
0	2022/3/11	JaSST'22 東京レビューワーク	A	23
1	2022/6/15	レビューワークショップ解説&ふりかえり	B	16
2	2022/7/11	レビュー目的の構造化	B	12
3	2022/7/12	Z社向けレビュー観点導出ワーク(1)	C	15
4	2022/8/25	HDR法を使った仮説検証アプローチ	B	15
5	2022/9/16	Z社向けレビュー観点導出ワーク(2)	C	21
6	2022/10/11	レビューファシリテーション	B	14
7	2022/10/25	Z社向けレビュー観点導出ワーク(3)	C	7
8	2022/10/28	JaSST'22Reviewレビュー観点設計ワーク	A	20
9	2022/11/14	レビュー結果評価とふりかえり	B	10
10	2022/11/15	Z社向けレビュー観点導出ワーク(4)	C	11
11	2022/11/29	Z社向けレビュー観点導出ワーク(5)	C	7
12	2022/12/20	Z社向けレビュー観点導出ワーク(6)	C	9
13	2022/12/26	欠陥知識の活用	B	9
14	2023/1/17	Z社向けレビュー観点導出ワーク(7)	C	11
15	2023/1/24	認知バイアスに着目した欠陥予測	B	9
16	2023/2/13	製品・サービスリスクに着目したレビュー観点導出	B	13
17	2023/2/14	Z社向けレビュー観点導出ワーク(8)	C	7
18	2023/3/27	レビュー時の「発言のしやすさ」を考えてみよう!	B	―
―	―	―	計	229

5. 活動評価結果

活動A,B,Cそれぞれに対して評価指標による評価を行った結果を,認知拡大目的分を表5に,適用促進目的分を表6に示す.(2023年3月17日現在)

表 5 当活動の評価結果(認知拡大目的分)

目的	認知拡大	
指標	活動継続度	活動への参加者数
活動A	期間中1回実施 2022/10/28ソフトウェアレビューシンポジウム2022	ワーク受講者数 20名(増)
活動B	期間中9回実施	勉強会登録者:90名増 ※参考:勉強会参加者数のべ98名
活動C	期間中7回実施	受講者数計 88名(増)
計	のべ17回 Avg 1.4回/月	(活動当初の23名より) 198名増

表 6 当活動の評価結果(適用促進目的分)

目的	適用促進			
指標	実践体系への理解度	実践体系の実務への有効度	指摘事項効果向上 効果大:中:小	研究成果の組込度
活動A	Avg 77.1 ※ < 80のため未達	Avg 88.6 ※満足度Avg 85.7	5チームのAvg 31:45:24	拡張2事項 見直し1事項
活動B	未計測	未計測	体系内個別手法実践のため対象外	研究成果5事項のトライアル実践済
活動C	Avg 91.3	Avg 91.8 ※満足度AVG 91.1	16チームのAvg 41:39:20	(解説スライド見直し)
計	理解度Avg 88.7	有効度Avg 91.2	全21チームのAvg 38:41:21	拡張7見直し1

なお,“活動継続度”については認知拡大,適用促進の両

面共通指標となっているため、その結果を表 5 にのみ示し、表 6 からは省略した。

表 6-活動 A-実践体系への理解度 1 事項の未達と一部未計測部分はあるものの、概ね期待の効果を獲得していることが確認できたため、引き続きこの活動を継続していく価値はあると判断した。

また、以上の評価と併せて活動 A,C(ワークでレビュー実践体系全体を実施する活動)の受講者がワークショップふりかえり時に提示したコメントの内容から、当活動の効果と課題を考察した。

5.1. ポジティブコメント

全ふりかえりコメント数 66 件のうち、ポジティブコメントは 44 件であり、その内容は想定通りのものとなっていた。その内訳は以下の通りである。

- ・アドホックレビューでは検出できなかった影響度大の指摘ができる／有識者的なレビューができる:15 件
- ・レビュー観点設計技法実践による効果を実感:8 件
- ・レビューがしやすくなる:5 件
- ・普段のレビュー時の観点が偏っていることに気づいた:5 件
- ・レビューアの立ち位置や観点の違いで指摘事項が変化することを実感した:4 件
- ・いつもは色々見るつもりが結果的に見ることができていない／観点を絞ると指摘漏れ防止＋実施が効率的になると感じた:4 件
- ・観点を個別に割り付けて見る(役割分担)有効性を実感した:3 件

5.2. ネガティブコメント

全ふりかえりコメント数 66 件のうち、ネガティブコメントは 22 件であった。

コメントの内訳は以下の通りである。

- ・観点の粒度が不適切だと指摘に繋がらない:5 件
- ・短時間でレビューしなければならないのに、この手法だといろいろ手間と時間がかかる:5 件
- ・観点の抽出が難しい:3 件
- ・観点割り付けすると逆に漏れが発生するのではないかと不安になる:2 件
- ・観点を網羅的に検討できているのかがわからない:1 件
- ・時間制約がある中で多岐に渡る観点をどこで割り切ればよいのか:1 件
- ・観点設計の効果を実感できない:1 件

- ・いつも少人数でレビューしているので役割分担できないがどうしたらよいのか:1 件
- ・レビュー対象の事前読み込みをするのが前提だが全員実践できるのか?:1 件
- ・しっかり考える必要があるこのアプローチをどうやって広げていけばよいのか:1 件
- ・どこまで細かく指摘すると良いのかがわからない:1 件

観点抽出と粒度調整に関する問題とそれに起因する時間不足などが計 18 件、現状のコンテキストに Fit させる方法について 4 件であり、これら対しては何かしらの対策が必要になる。

6. 当活動の課題と今後

これまでの活動を通じて表面化した課題と今後の対応を表 7 に示す。これらの対応を含めて今後も継続して当活動を実施していく所存である。

表 7.当活動の課題と今後の対応

現状の課題	今後の対応
5.2 ネガティブコメント「観点抽出に関する問題」	継続勉強会を通じて徐々に体得してもらう
5.2 ネガティブコメント「粒度調整に関する問題」～観点導出手法として高位レベルレビュー観点導出手法と低位レベルレビュー観点導出手法を特に識別していない。ワーク実践者の所有スキルとアンマッチになる手法を選択してしまうと観点がうまく導出できない場合がある。	高位レベルレビュー観点導出手法と低位レベルレビュー観点導出手法を識別してスキルレベルによる選択方法をガイドする。
5.2 ネガティブコメント「現状のコンテキストに Fit させる方法」	ワーク解説資料に典型的な状況に対する Fit 例を追加
レビューを体系的に実践するための枠組みであるため、すべての要素を連携して実践した結果どのような効果を獲得できるかを確認するための十分な時間を取ったワークが実現できない。	長時間テーマに取り組む合宿等で実施することを視野に検討を進める。
勉強会では適用促進度を測る受講者評価を取得していない。	今後の勉強会で取得し、定期的に効果を評価する。

参考文献

- [1] 「ソフトウェアレビュー研究結果の認知拡大と適用促進」 安達 賢二, 中谷 一樹, 上田 裕之
(ソフトウェア・シンポジウム 2022 発表 経験論文)

STAMP/STPA による農業用ハウスの運用シナリオの分析

日下部 茂
長崎県立大学

鍵 文音
長崎県立大学

有田 大作
長崎県立大学

要旨

農業用ハウスを用いる施設栽培では外界とハウス内耕地を分離し、外部環境の影響を抑えて内部環境を制御することにより作物に適した環境を管理・調整し、安定した収穫を可能とする。そのような施設栽培でハウス内環境を制御している代表的な機器として、温度を制御する暖房機、日長を制御する照明機と遮光装置、水やりを自動的に行う灌水装置、光合成に必要な二酸化炭素を放出する二酸化炭素発生装置などがある。農業用ハウスのライフサイクルではそのような機器構成に変更が生じる可能性があり、その際は運用シナリオも再考する必要がある。我々はそのような運用者のシナリオの分析に、制御の観点で対象システムをモデル化して分析する STAMP/ STPA を適用する研究を行っている。本稿では事例として、二酸化炭素発生装置と天窓、暖房機の組み合わせからなるシステムを対象に、運用者による実践を想定したモデル化と分析を行い、その有効性を確認した。

1. はじめに

1.1. 背景

農業用ハウスを用いる施設栽培では外界とハウス内耕地を分離し、外部環境の影響を抑えて内部環境を制御することにより作物に適した環境を管理・調整し、安定した収穫を可能とする。より具体的には、

- 日射量や温度などの環境条件を調整することで、本来とは異なる収穫時期にずらして作物の希少性を高め、収入の増加につなげることができる、
- 台風や日照りなどの天候災害、虫や鳥獣といった被害から作物を守ることができる、
- 悪天候による作業の遅れがなく計画通りに作物栽培を行うことが出来る、

といった利点がある。

そのような施設栽培でハウス内環境を制御している代表的な機器として、温度を制御する暖房機、日長を制御する照明機と遮光装置、水やりを自動的に行う灌水装置、光合成に必要な二酸化炭素を放出する二酸化炭

素発生装置などがある。今挙げたものは日本でのハウス農業に導入されている機器の一部である。より精緻に、安定して環境を制御して栽培を行いたいと考える農家はこれ以上の機器を取り入れている所もある。このような農業用ハウスの典型的な運用シナリオは、IT 化された自動制御を農家が利用するもので、図 1 のように農業用ハウス内のシステムと人による制御の組み合わせで実現される。農業用ハウスの施設栽培の場合、外界とハウス内耕地を分離し、内部環境を作物に適した環境に制御することにより安定した収穫を目指す。下向きの矢印は何らかの制御、上向きの矢印は何らかのフィードバックを概念的に示している。制御階層の最上位には農家、施設内環境に直接的に働きかけるのは農業用ハウスシステムである。

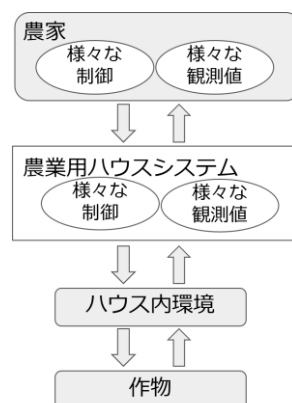


図 1 農業用ハウス管理の概念的なモデル

このような農業用ハウスの制御機器は複数種類ありハウスを建てた当初から全て導入されているとは限らない。何らかの契機での追加導入の可能性があり、そのような追加導入は、以下のような理由でハウス建築当初からは計画されていない場合も多い。

1. 政府の補助金を利用した機器導入が検討され、補助金の条件への適合度を優先して導入機器の選定が行われる場合がある。
2. ハウス建設時に普及していた機器を導入済でも、後に登場した新しい機器等を導入する場合がある。

このような制御機器の追加的拡張はシステムレベルでの運用シナリオの問題を生じることがある。また、追加で

なくとも、既存ハウスを他者に譲渡したような場合、新しい所有者は適切な運用シナリオを把握していない場合がある。以下、そのような運用シナリオに関し、農業用ハウスのライフサイクルの特性に起因する問題について論じる。

1.2. アプローチの概要

農業用ハウスシステムで理想的な運用を実現するには、以下の点が重要と考える。

- 目的が適切に定義されているか
- 対象を観測できるか
- 適切な制御を判断できるか
- 対象に制御を伝達できるか
- 実際に制御が実行されるか

これらの点をふまえ農業用ハウスシステムの運用シナリオの問題に対し、制御の観点でモデルを構築し、栽培管理で守るべき制約などを分析する研究を行っている[1]。広義の安全工学の知見を農業用ハウスの制御に応用するため、モデリングに STAMP (System-Theoretic Accident Model & Processes) [2]を用いている。

図 2 のような社会技術システムの開発と運用での体制と相互作用の視点で見ると、前述の農業用ハウスの運用シナリオの問題は、図 2 の下部中ほどに点線の楕円で囲った部分の改定運用手順の獲得に相当すると考える。シナリオベース設計[4]やユースケースシナリオ作成、シナリオテストなどもシナリオ観点の活動だが開発プロジェクト内部で実施され開発ドキュメントにもアクセス可能で

ある。一方ここで対象とする農業用ハウス運用手順の獲得は開発プロジェクト外で開発ドキュメントへのアクセスや開発者の関与が必ずしも期待できない場合がある。

本稿では、事例として、二酸化炭素発生装置と天窓、暖房機の組み合わせを用いる農業用ハウスシステムを対象に STAMP によるモデル化と STPA(System-Theoretic Process Analysis)[3]を用いた分析を行う。STAMP/STPA にはハンドブックが公開され、その中に適用時のヒントなども書かれているが、それに加え運用者レベルのガイドラインの抽出も目指す、本稿の構成は以下の通りである。第 2 節で、農業用ハウスについて説明する。第 3 節では、STAMP/STPA について述べる。第 4 節では、事例のモデリングについて論じる。第 5 節ではまとめを述べる。

2. 農業用ハウス

2.1. ハウス内環境とハウス内機器

農業用ハウス内で、植物に影響を与える条件の代表的なものとして、温度、湿度、二酸化炭素濃度、土壤水分、風量、日射量といったものがある。これらのバランスによって作物の生長に差が出る。そのような複数のハウス内環境条件を制御して所定の作物を収穫するため複数の機器が導入されることがある。以下に、日本のハウスで使われる代表的な機器を挙げる。

- 暖房機:ハウス内温度を上昇させるために使用。
- 循環扇:ハウス内の空気を循環させ、空気を全体的

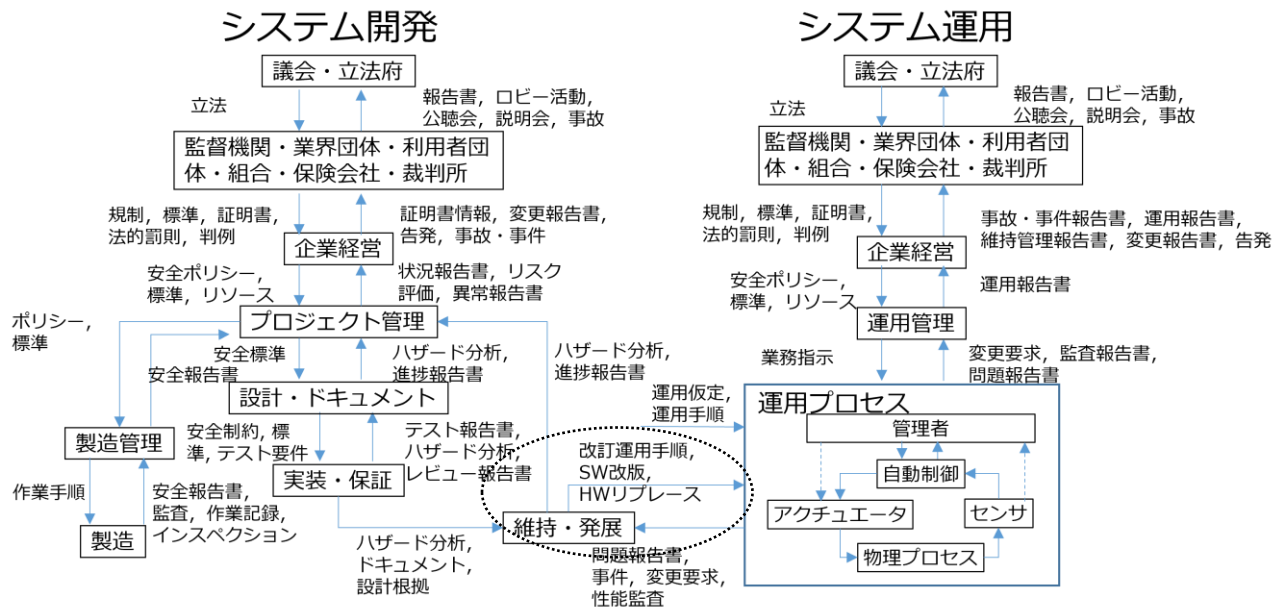


図 2 社会技術システムの開発と運用での制御構造

に均一にするために使用.

- 灌水装置: 水をやる装置.
- 天窓と側窓: 窓を開閉し, 外気を取り入れ, ハウス内環境をハウス外環境と同様にするために使用.
- 照明機: 植物の光合成には光が必要なので, 光量を増やすために使用される.
- 遮光装置: 植物に光が当たりすぎても良くないので, カーテンのようなもので日光を遮るために使用.
- ミスト発生装置: ミストを発生させ, ハウス内の気温を下げるために使用.
- 二酸化炭素発生装置: 燃料を燃やして光合成に必要な二酸化炭素を発生させるために使用.

例えば, 二酸化炭素について説明する. 大気の二酸化炭素濃度は約 400ppm であるが, 光合成後のハウス内濃度は 100ppm 以下になることも多々ある. そのような低い濃度は植物には不適切であり, その防止のため二酸化炭素発生装置が用いられる. また, 二酸化炭素発生装置は光合成の活性化のため, 二酸化炭素濃度を 800~1000ppm にするためにも使用される. このような制御機器と制御される環境条件の組み合わせを表 1 に示す.

表 1 制御機器と主に制御される環境条件

	暖房機	循環扇	灌水装置	天窓・側窓	照明機	遮光装置	CO ₂ 発生装置	ミスト発生装置
温度	✓			✓				✓
湿度								✓
CO ₂ 濃度							✓	
土壌水分			✓					
風量		✓						
日射量					✓	✓		

2.2. 農業用ハウスの運用シナリオの問題

ハウス内の環境条件を制御する際, まずそれらの環境条件の値を観測し, 農家や導入機器のコントローラが, それらの観測値に基づいて対応する機器を制御してハウス内環境を変化させ, それにより作物の成長の制御を試みる. このような制御に用いる機器はハウスを建てた当初から全て導入されているとは限らず, 追加導入されることもある. その際, 機器同士の相互作用を分析の上で運用シ

ナリオの改定が必要であるが, 適切な分析の不足により想定していなかった動作が引き起こされる場合がある.

説明のための仮の例として, 二酸化炭素発生装置と天窓を備えたハウスで, 二酸化炭素濃度を上げる際の天窓のいわゆるフィードバック[5]の可能性について述べる. 二酸化炭素濃度を大気中より高くして光合成を活性化するために二酸化炭素発生装置が使用されることがある. その際に, 機器間の協調がないと, ハウス内温度が天窓の設定温度よりも高くなれば, 温度を下げるために天窓が自動で開く. そうすると, 二酸化炭素発生装置により増加させた二酸化炭素は開いた天窓からハウス外へと放出されてしまう. このように, 各機器とそれらを統合した動作シナリオの分析が不足していると, 環境が十分に制御されなかったり, 不要なコストが発生したりといったしてしまう可能性がある. 想定していなかった相互作用によりシステムの目標達成が失敗したり悪影響を受けたりするような不具合を分析するため, システム理論に基づき, 相互作用によって起こり得る不具合も分析可能な STAMP/STPA を用いたモデリングと分析を行う.

3. STAMP/STPA

本稿では, 農業用ハウスシステムの栽培管理のモデル化とシナリオ分析に用いる STAMP/STPA を概説する.

3.1. モデル表現

STAMP によるモデル表現では図 3 の基本構造が示すようなモデル要素を持っており, 図 1 に示されたような構造を持つ農業ハウスシステムの分析に有用と考える.

- コントローラとコントロール対象のプロセス
- それらのプロセス間での制御とフィードバック
- コントローラが持つ対象プロセスのモデル

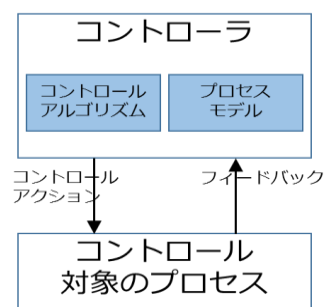


図 3 STAMP の基本的制御構造のモデル

図 3 の STAMP モデルの基本構造は, コントローラとそれによってコントロールされる対象プロセスからなる. コ

ントローラはプロセスモデルとアルゴリズムを持つ。プロセスモデルはコントロール対象のプロセスの状態をモデル化したものである。プロセスの状態は、プロセスをセンサ等で観測した観測値として表される。アルゴリズムはプロセスの状態をもとにコントロール対象プロセスへのアクションを決める。コントロール対象のプロセスからのフィードバックによって、プロセスモデルの観測値は更新され、再度新しい観測値を参照してアルゴリズムがアクションを決定するというループ構造を形成する。

複数のコントローラや制御機器、センサを持つようなシステムも、上記基本構造の組合わせで表現できる。このような STAMP/STPA の制御構造のモデリングで農業ハウスシステムもモデル化出来ると考え本研究を実施する。

3.2. 手順

STAMP/STPA の基本的な分析手順は図 4 に示すように大きく分けて 4 つのステップになっている。

STEP1

STEP 1 は解析の目的を決める。「どのような損失をこの解析で防ぐことを目的としているか」、「解析対象のシステムは何か」というような基本的な設定を明らかにする。今回は農業ハウスシステムを対象とし、作物が望むように収穫できないことを防止することを目的とする。STEP 1 もさらに以下の4つの手順に分けられる。

1. 損失の識別
2. ハザードの識別
3. 安全制約の識別
4. ハザードの精密化

1つ目の損失識別では、利害関係者に受け入れられないような人間の傷害、物的損害、ミッションの喪失などの損失を識別する。今回は2つ目のハザードの識別では、

損失に繋がるようなシステムの状態や条件を列挙する。3つ目の安全制約の識別では、ハザードを防ぐために必要なシステムの条件や動作を確定させる。4つ目のハザードの精密化では、2つ目で挙げたハザードにつながるアクションやプロセスを考える。

STEP2

STEP 2 では、システムの制御構造図を構築する。典型的な制御構造図の構築は抽象的なレベルから始め、必要に応じて対象範囲を限定し詳細化することを繰り返す。抽象レベルの制御構造図中のコントローラやプロセスをより具体的な制御構造で表現し分析の詳細化を進める。システムのモデル化と分析の対象範囲は目的に合わせ任意に決定出来る。今回は、サブシステムを組み合わせでシステムを実現するような制御構造図の構築を行う。

STEP3

STEP 3 では、STEP 2 で作成した制御構造図を用いて分析を行う。コントロールループ内の各コントロールアクションが STEP 1 で定義したハザードに繋がる可能性があるか否かを調べる。ハザードに繋がると判断されたコントロールアクションは非安全なコントロールアクション(以下「UCA」とする)と呼ばれる。

STEP4

STEP 4 では、システム内で UCA が発生する可能性のある要因を識別する。さらに、どの要因が、どのようなUCA を引き起こし、損失に繋がる可能性があるかどうかを説明するためにシナリオを識別する。

このようなSTAMP/STPAは、ボトムアップでなく、トップダウンで分析を行うため複雑なシステムでも分析しやすい

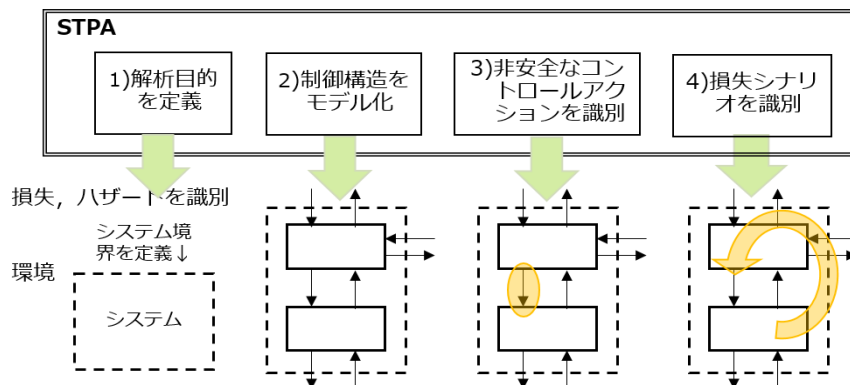


図 4 STAMP/STPA の基本的な分析ステップ

い。また、ソフトウェア、人間、組織というような事故や因果関係要因を別々に扱うことなく1つのモデルの中で扱う。そのため特別なITの知識を持たない農業用ハウスの運用者でもUCAにつながる要因の分析や、UCAが引き起こされ損失が生じるようなシナリオの分析が可能と考える。

4. 分析事例

本研究では説明のために、解析対象とする農業用ハウスの機器と環境条件を次のように限定して行った;暖房機、二酸化炭素発生装置、天窓の3つの機器と温度、二酸化炭素濃度の2つの環境条件である。このような限定された機器と環境条件であっても、追加的な組み合わせによって問題が生じ得る。このように限定された条件からはじめて、より複雑な組み合わせについても解析可能とすることを目指す。以下、人身的な傷害や農業用ハウスの物理的損壊といったものは省き、作物の望ましい収穫に焦点を当て、運用者の立場での分析について述べる。

4.1. STEP 1: 解析目的の決定

ハウス内機器システム

まず農業用ハウス内システム全体について検討した。そのSTEP 1の解析結果について述べる。まず、対象システムとシステムの目的は、望ましい収穫のためハウス内環境を効果的に制御することとした。

- 損失識別
 - L1. 望ましいように作物が収穫できない(3つの定(定量・定品質・定時)が守れない)
 - L2. 無駄なコストがかかる
 - ハザード識別
 - H1. 3つの定に適したハウス内環境を制御できない[L1]
 - H2. ハウス内環境の制御の効率が悪い[L2]
 - 安全制約
 - SC1. 3つの定に適したハウス内環境を維持する[H1]
 - SC2. 効率よくハウス内環境を制御する[H2]
 - ハザードの精密化
 - H1.1 ハウス内温度が適正值より高い
 - H1.2 ハウス内温度が適正值より低い
 - H1.3 ハウス内二酸化炭素濃度が適正值より低い
 - H2.1 ハウス内環境の制御が阻害されている
 - H2.2 ハウス内環境の制御が打ち消されている
- L1の3つの定というのは、農業をするにあたって農家が意識している3項目のことである。一定量を定品質で、定

まった時期、期間に出荷出来ることを目指している。

天窓システム

天窓についてのSTEP 1の解析結果について述べる。まず天窓システムの目的は、ハウス内温度を下げることで、以下のような分析を行った。

- 損失識別
 - L1. 望ましいように作物が収穫できない
 - L2. 無駄なコストがかかる
- ハザード識別
 - H1. 作物に不適切な温度になる[L1]
 - H2. 温度管理の効率が悪い[L2]
- 安全制約
 - SC1. 作物に適切な温度を保つ[H1]
 - SC2. 温度管理を効率よく行う[H2]

暖房機システム

暖房機についてのSTEP 1の解析結果について述べる。まず暖房機システムの目的は、ハウス内温度を上げることとし、以下のような分析を行った。

- 損失識別
 - L1. 望ましいように作物が収穫できない
 - L2. 無駄なコストがかかる
- ハザード識別
 - H1. 作物に不適切な温度になる[L1]
 - H2. 温度管理の効率が悪い[L2]
- 安全制約
 - SC1. 作物に適切な温度を保つ[H1]
 - SC2. 温度管理を効率よく行う[H2]

二酸化炭素発生装置

二酸化炭素発生装置についてのSTEP 1の解析結果について述べる。二酸化炭素発生装置の目的は、二酸化炭素濃度を上げることとし、以下のような分析を行った。

- 損失識別
 - L1. 望ましいように作物が収穫できない
 - L2. 無駄なコストがかかる
- ハザード識別
 - H1. 作物に不適切な二酸化炭素濃度になる[L1]
 - H2. 二酸化炭素濃度管理の効率が悪い[L2]
- 安全制約
 - SC3. 作物に適切な二酸化炭素濃度を保つ[H1]
 - SC4. 二酸化炭素濃度管理を効率よく行う[H2]

4.2. STEP 2: 制御構造図

農業用ハウスシステムは後付的に拡張される傾向があるため、個別機能の制御構造図をまず考え、それらを合成して農業用ハウスシステム全体の制御構造図を考える。また、例えば[6]のような製品資料といった、運用者の立場で入手可能な資料や、システムを目視して得られる情報から判断可能なレベルの詳細度を想定する。

天窓システム

天窓の制御構造図を図 5 に示す。天窓には、ハウス内環境の温度データを得る温度センサとハウス外環境である雨量、風速、風向のデータを得るセンサがついている。設定温度よりもハウス内温度が高くなれば天窓を開け、温度が下がると天窓を閉めるよう制御する。ただし、風速と風向の観測値をもとに、風が強い場合は風上の天窓は開けずに風下の天窓だけ開け、風がより強い場合は風下の天窓も開けないという制御も行っている。このようにして天窓はハウス内温度について環境制御している。

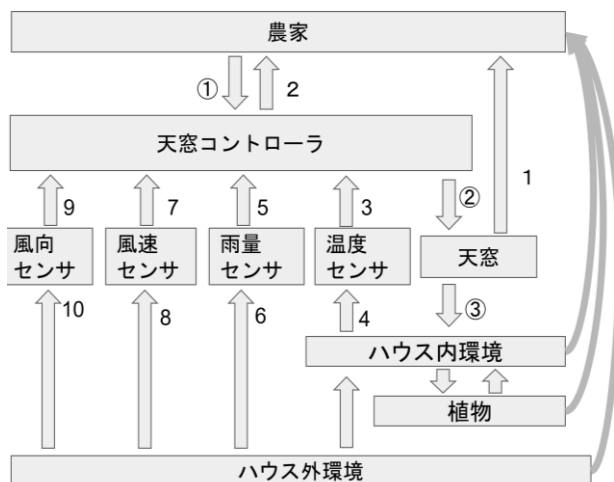


図 5 天窓の制御構造

暖房機システム

暖房機システムの機制御構造を図 6 に示す。ハウス内環境である温度のデータを得る温度センサの観測値を元に、設定温度よりもハウス内温度が低くなっていたら暖房機の電源をいれてハウス内温度を上昇させるなどして自動的に暖房機の電源がオン・オフされハウス内温度について環境制御している。暖房機にはタイマーもついており、農家が設定した時刻になると電源が入り、その後電源が落とされるというような制御もついている。

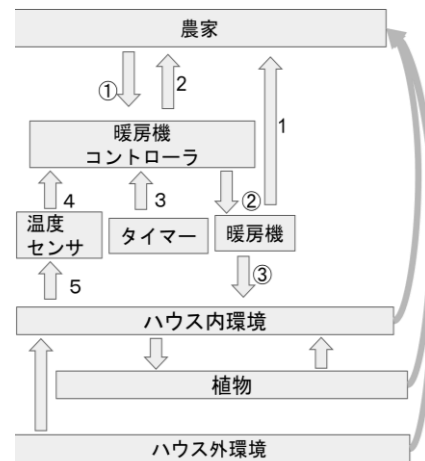


図 6 暖房機の制御構造

C02 発生装置

二酸化炭素発生装置の制御構造を図 7 に示す. 二酸化炭素発生装置には, ハウス内環境である二酸化炭素濃度のデータを得るセンサがついている. この観測値を元に, 設定濃度よりも二酸化炭素濃度が低くなったら装置を稼働させ, 二酸化炭素濃度を上げるということを行い, ハウス内二酸化炭素濃度について環境制御している.

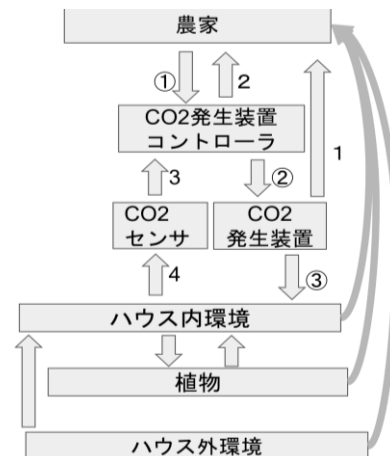


図 7 CO₂ 発生装置の制御構造

農業用ハウスシステム

上述の天窓、暖房機、二酸化炭素発生装置の制御構造図を 1 つにまとめ、簡略化すると図 8 になる. コントローラが持つプロセスモデルはカックで囲んで記述している. 天窓についている温度センサと暖房機についている温度センサは別物で得られる観測値間に差が発生する可能性もある. 2つの機器が独立して設置されているので天窓コントローラのプロセスモデル「温度」と暖房機コン

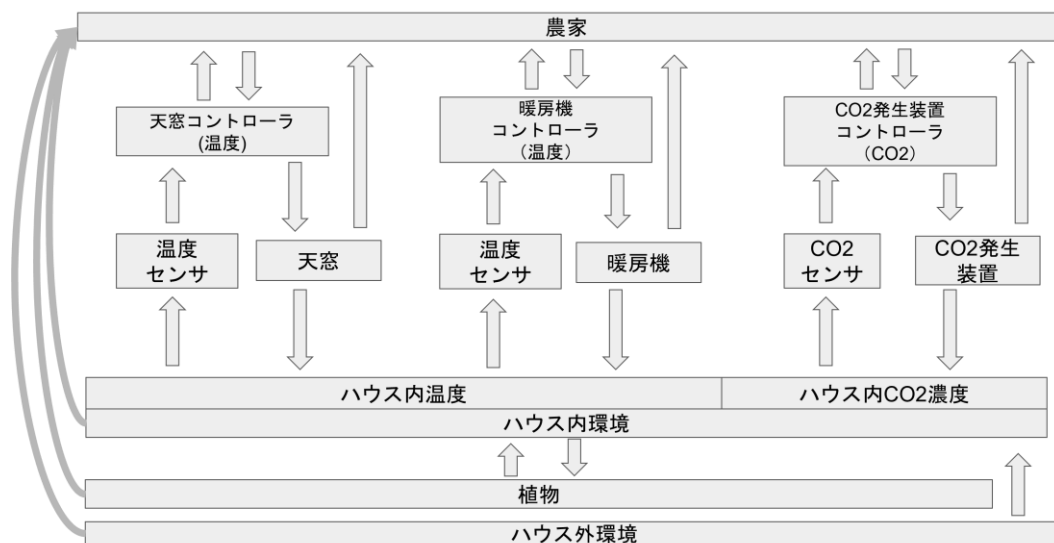


図 8 農業用ハウスシステムの制御構造図

トローラの「温度」は別物である

4.3. STEP3:UCA の識別

UCA の識別では各制御アクションに関するハザードの可能性として、与えたから、与えなかったから、不適切な順序で与えたから、連続的アクションの長さが不適切だったから、の4つの観点から分析する。UCA は形式的には、上記4観点をタイプとし{ソース、タイプ、コントロールアクション、コンテキスト、ハザードへのリンク}というタプルで表現される。典型的には、表形式で各制御アクションに対する前述の4つの観点での分析を記述しながら分析を行うがそのような表は一般にかなり大きなサイズになる。

ここではその一部として天窓コントローラが天窓開放指示を与えることによるハザードについて述べる。形式的には、ソースが「天窓コントローラ」、タイプが「与える」ことによりハザード、コントロールアクションが「天窓開放」になる。以下はそのような組み合わせのUCAで、コンテキストと、ハザードへのリンクを含んだ例である。

- UCA1. {天窓コントローラ, 与える, 天窓開放, 作物育成に適したハウス内環境の時に天窓開放によってハウス内環境が外部環境と同じ状態になる, H1}
- UCA2. {天窓コントローラ, 与える, 天窓開放, 環境管理機能によって作物育成に適したハウス内環境にした後に天窓開放によってハウス内環境を外部環境と同じ状態にしてしまい環境管理機能を再実行すべき状態になる, H2}

4.4. Step4: 損失シナリオの識別

図 8 のようなシステム内でUCA が発生する可能性のある要因を識別する。さらに、どの要因が、どのようなUCA を引き起こし、損失に繋がる可能性があるかどうかを説明するためにシナリオを識別する。ここではガイドラインにつながるパターンの抽出を念頭にUCA2 とUCA1 に関する分析を行う。

UCA2 に関する損失シナリオ例

天窓コントローラも暖房機のコントローラも温度をモニタリングしているが、個別のサブシステムを組み合わせる農業用ハウスシステムを構築する場合、温度センサも個別に存在し、それぞれのコントローラで個別にフィードバックを受けている可能性がある。

例えば暖房機システムの温度センサが低い位置にあり、天窓システムの温度センサ高い位置にある場合、前者が低めの温度を、後者が高めの温度をフィードバックする可能性がある。そうすると暖房機システムのコントローラは低めの温度をもとに温度を上げようとする一方、天窓システム側は高めの温度をもとに天窓を開けて温度を下げようとする状況になり得る。農業用ハウスの中で農作物が存在する位置の気温は適温であったとしても、本来不要な暖房機の稼働により燃料コストが高くなってしまふ。

これは複数のコントローラが同一の被制御対象のプロセスモデルを個別に持ち、それらの間に協調がない場合は問題が発生する可能性がある、といった一般化が可能

である。このような状況は、図 9 に温度の例を模式的に示すように、御構造図上でプロセス変数の重複やコントロールプロセス間の相互作用を確認すれば良い。

このような場合は以下のような確認ガイドラインができる。

- 同一制御対象に対し、複数コントローラが持つプロセスモデル変数間の同一性や変換関係の確立
- 複数コントローラが矛盾する指示を出す被制御対象値の分析や必要な協調の分析。

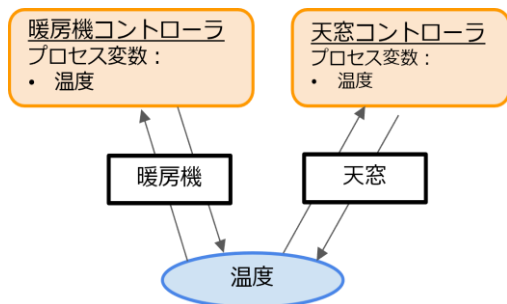


図 9 温度とフィーチャインタラクションの例

UCA1 に関する損失シナリオ例

天窓システム導入の目的は暖気を排出することとされ、実製品の例でもセンサとコントローラは温度制御に対応している[6]。しかしながら、実際に天窓を開放すると温度以外の条件も外気同様になり、いわゆる副作用が生じる。例えば天窓を開け外気を取り込むことにより、二酸化炭素濃度も大気中の二酸化炭素濃度と同等になる。一部農家は、意図的にこの副作用を活用する一方、前述のように意図しない二酸化炭素濃度の低下となる場合もある。

このような副作用をふまえると図 8 は、天窓が二酸化炭素濃度に影響を与えることが反映されていない。この点に関連のある部分に焦点を当てた図が図 10 である。天窓側では二酸化炭素濃度のフィードバックがない。また、そのフィードバックを加えると UCA2 の説明同様、複数コントローラ間での競合がパターンに当てはまる。これらは以下のようなガイドライン化が可能である。

- コントロールアクションの副作用を確認
- 副作用あれば制御構造図に反映し再分析

5. おわりに

農業用ハウスを用いる施設栽培では外界とハウス内耕地を分離し、外部環境の影響を抑えて内部環境を制御することにより作物に適した環境を管理・調整し、安定した収穫を可能とする。我々は、そのような農業用ハウスのライフサイクルでの運用手順の獲得や改定の問題を分

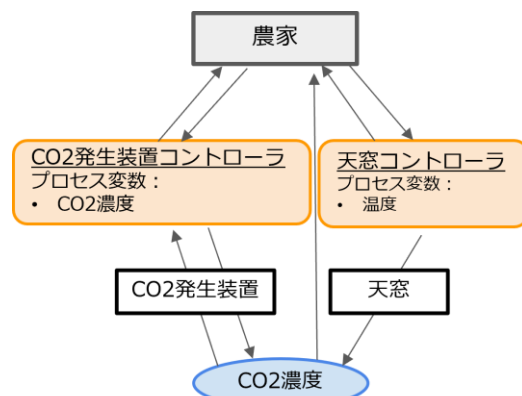


図 10 フィードバックの欠如例

析するために、制御の観点で対象システムをモデル化して分析する STAMP/ STPA を適用している。本稿では事例として、二酸化炭素発生装置と天窓、暖房機の組み合わせを用いるシステムを対象にモデル化と分析を行った。

ライフサイクルでの開発フェーズ外の段階での、運用手順の獲得や改訂では開発者の関与や開発時の資料へのアクセスは限定されたり困難だったりする可能もある。そのような状況でも STAMP/ STPA により有用な個別の分析や、ガイドライン化可能な知見の獲得が可能であった。事例では、制御対象の競合、安全制約維持に必要なフィードバックの欠如、といった問題の分析や視覚化に制御構造図が有効であった。事例は限定された組合せの農業用ハウスシステムであったが、引き続き研究を継続しより多くの知見の一般化やガイドライン抽出を目指す

参考文献

- [1] 鍵文音, STAMP/STPA による農業用ハウスにおける栽培管理制約の形式知化, 長崎県立大学卒業論文, 2023 年 3 月
- [2] N. Leveson, Engineering a Safer World, MIT press, 2012.
- [3] N. Leveson & J. Thomas, STPA Handbook 日本語版 (JAXA による翻訳), http://psas.scripts.mit.edu/home/get_file2.php?name=STPA_handbook_japanese.pdf (参照 2023-05-19)
- [4] John M. Carroll, Making Use: Scenario Based Design of Human-Computer Interactions, MIT press, 2003
- [5] Feature Interaction in Telecommunications, Vol. 1-VI, IOS Press (1992-2000)
- [6] 東都興業株式会社, 換気シリーズ, <https://www.toto-vp.com/products/?ssid=1000005> (参照 2023-05-19)

それって技術の仕事！？：“仕様の輻輳”問題 ～心理的輻輳から脱出して健康な開発に～

池田 暁
クオリティアーツ
akira.ikeda@quality-arts.com

松尾谷 徹
デバッグ工学研究所
matsuodani@gmail.com

要旨

議論する.

本 FP では、多くのエンジニアが経験する「仕様の輻輳問題」を取り上げて議論する。仕様の輻輳を「エンジニア個人に対して仕様にまつわる調整事項がエンジニア個人に集中し、開発進捗に大きな影響を生じる状態」と定義した。仕様の輻輳の状態が悪化すると、エンジニアの「心理的輻輳」を引き起こし、それが更に仕様の輻輳を悪化させる。仕様の輻輳は多くの現場でエンジニア個人が解消しようとしているが、本来は PM (Project Manager) の仕事である。べき論はあるが、日々の実際としてエンジニア個人はどう対応すべきだろうか。エンジニア側かつ文化や人間的側面から議論する。

1. はじめに

近年、IT プロジェクトの対象が複雑化し、工数ベースの請負作業から、DX など対象分野のスキルや、機械学習/仮想環境のスキルなど、スキル重視ベースへと変化している。高度なスキルを持つエンジニア (コアエンジニアと呼ぶ) パフォーマンスを引き出すことがプロジェクト管理の中で重要な課題となっている。

コアエンジニアがジレンマに陥り、そのパフォーマンス低下を引き起こす仕様の輻輳問題は、実装段階以前にステークホルダー間で調整し、契約や仕様のコミットが必要な様々な要件を決められず、先送りして実装段階へ投げるのが主な原因と推測される。我が国独特のあいまいな契約や工程完了のコミット文化が影響している。

文化や組織、人間的な原因を含んでおり、技法や技術だけで解決できない課題であり、かつ状況が深刻化することでプロジェクトや企業の開発競争力にも影響することから、FP のテーマとして取り上げる。今回は、開発現場で実際に生じている問題状況を主にとりあげ、当事者であるエンジニアの側から問題を明らかにし、その後で原因メカニズムのモデル化を試み、解決への糸口について

2. 仕様にまつわる悩み

ソフトウェア開発において、仕様にまつわる悩みは多い。仕様定義はもちろんとして、仕様の管理や仕様のプログラムへの変換、テストにおける仕様の取り扱いなど挙げるときりがない。この仕様に関する課題については、仕様記述言語やモデリング手法、仕様のバージョンやトレーサビリティの管理、等に関して様々な手法や提案があり、実用化したり広く普及したりしているものもある。

仕様の不確定や変更から生じる問題という、必ずしも技術だけでは解決できない問題もある。例えば、仕様の実装フェーズにて、顧客をはじめとしたステークホルダーから急な仕様変更依頼が生じたり、それら依頼が複数のステークホルダーから非同期的に生じたり、依頼内容が衝突したりすることが頻発することで、それらを受けたエンジニアには「調整」というアウトプットを生み出さないタスクが増え、本来予定していた進捗に影響を生じるということである。この例のような「仕様の輻輳」が生じると、エンジニア個人はもちろん企業の開発進捗等にも悪影響が生じる。

3. 仕様の輻輳とは

本 FP での議論を進めるにあたって、仮ではあるが、「仕様の輻輳」を定義する。

「仕様の輻輳とは、エンジニアのロール外で、納期や仕事量や顧客間で調整が必要な要件変更などを立場上、の差(顧客と請負など)を利用して求め、それら解決が困難なジレンマ状態」とする。輻輳とは物が一か所に集中し混雑することであるが、エンジニアに仕様についての物事が集中することを想起させる言葉として採用している。

仕様の変更は言い換えると仕様の対立である。例とし

て、「顧客対エンジニア」や「ビジネス対開発」がある。この「仕様の対立」は、その「主張者の立場(ロール)」、「仕様に対する観点」、「視座や視点の異なり」等が考えられる。対立があるので解消のための調整が必要となる。

仕様の対立によって生じるエンジニアへの仕様変更依頼が増大していくことで輻輳はより深刻になっていく。

皆さんも、様々な顧客や担当者から、ある仕様項目について定義内容や修正依頼がばらばらに”異なる内容で”なされ、その調整に時間を取られ、開発の進捗も芳しく無くなり、にっちもさっちもいかなかった経験があると思う。そして、おそらく、現在進行系で悩んでいる方も多いと想像する。

4. 仕様の輻輳を引き起こす事象の例

仕様の輻輳問題を考えるにあたって、それを引き起こす例をいくつか挙げてみる。

- 仕様に関する、発注者と受注者での認識が異なっている
- 開発とは同期していない、ビジネス面からのリリース変更や機能追加のさしこみが生じる(確定していた要求が変わる)
- 前工程で要求が固まっていない、(顧客が)誤って承認してしまった等の理由で、あとで要求の変更や追加、優先度付の変更が発生する
- 顧客側の手続きや納期的な面からの調整
- 仕様の実装のために複数の顧客担当者との調整が必要な場合(例えば、サブシステム同士と繋げるためのインタフェース実装)
- ステークホルダーの連携不十分による、異なった、同期しない仕様変更の発生

これらのような事象の発生により、仕様に関する調整依頼連絡がエンジニアにもたらされ、集中することで、仕様の輻輳が発生してしまう。

5. 仕様の輻輳が引き起こす心理的輻輳

仕様の輻輳が発生したとき、その解決や沈静化にエンジニアは、要求や仕様に関する手法を用いたり導入に挑戦したりすることで、事態の改善を図ろうとしてきた。

ただ、実際には、手法のありなしにかかわらず、仕様

変更依頼の量に加え、複数の担当者との調整や仕様の輻輳は、エンジニアに心理的な負担を生じさせ、それが大きくなることでいわば「心理的な輻輳」を引き起こさせる。いわゆる「いっぱいいっぱい」な状態になってしまい、仕様の輻輳を解消するための調整すらも進まなくなってしまう。結果として開発も進まなくなることで、エンジニアはより精神的に疲弊し、ますます事態は悪化していく。担当エンジニアが仕様の調整に忙殺され、疲弊していく姿は皆さんも経験したり目の当たりにしたりしてきたことだろう。

6. 心理的輻輳を引き起こしている状況

エンジニアが心理的輻輳状態になっているとき、どういった状況であるか、いくつか例を挙げる。

- 担当者と、または本来の担当者とは別の担当者との整に忙殺されている
- 直接アクセスできないステークホルダーへの確認の試みが生じている
- ロール外責務の、仕様の取り扱いレベルや実装スケジュール、開発リソースの調整
- 仕様変更解消に向けたプレゼンテーションの準備と実施、フォローアップ

果たして、これらは本来エンジニアの仕事だろうか。

7. 仕様の輻輳を抑えるのは本来 PM の仕事

顧客との、仕様の内容変更や優先度・重要度、スケジュールやリソース配分等の調整は本来 PM が実施すべきことであり、プロセスで解決すべき問題である。だがエンジニア個人は、その生真面目さにより個人技で解決を図ろうとして沼にはまってしまう。

エンジニアは、仕様の対立が生じたときそれを解決するのは PM であるということをあらためて理解する必要がある。そして、仕様の輻輳が起きた場合はその解決を PM やステークホルダーに調整や判断を委ねる(提案する)ことが大切である。ただし、委ねるためには、仕様の対立における全体像を理解し、然るべき対策に分け、そのうえで委ねる必要がある。もちろん仕様の輻輳そのものも理解する必要があるし、顧客やPMを含むステークホルダーが仕様の輻輳にどのようなかわり方をしているかも理解する必要がある。さらにはその大きな背景である、国内特有の受発注文化等も理解する必要がある。理解するこ

とが多いため、モデル化等の方法を使い全体理解をしやすくすることが必要であると考え。またモデルがあることでステークホルダー間での使用の輻輳の抑制について議論がしやすくなる。

8. まとめ

本 FP では仕様の輻輳および心理的輻輳について議論提起したが、現時点ではあくまで思索にとどまる。本 FP ではエンジニアより「自身や身の回りに起きている仕様の輻輳」について例示いただき、それらが持つ特質や特徴、特に解決すべき課題はなんなのか、複数の事例に共通している課題は何かといったようなことの議論を期待する。

その議論を受けて、今後は、以下のようなことについて議論を深めていきたい。

- 「仕様の輻輳」「心理的輻輳」の議論と定義
- 「仕様の輻輳」「心理的輻輳」のモデル表現
- モデルの活用検討や活用のためのスキル定義
- スキル獲得のための教育
- その他関連する事柄

これらを SS の WG で継続議論をしていく。関心がある方はぜひ議論に加わっていただきたい。

ソフトウェア・シンポジウム 2023 in 仙台
論文集

© ソフトウェア技術者協会

2023 年 6 月 7 日 初版発行

編者 三輪東
小笠原秀人

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/>

ISSN 2758-8572



ソフトウェア技術者協会