ソフトウェア・シンポジウム 2019 in 熊本

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 39 回目を迎える 2019 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，SS2018 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 ： 2019 年 6 月 5 日（水曜日）～ 7 日（金曜日）

■ 場所 ： 熊本国際交流会館

■ 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 ： QUEST，九州 IT 融合システム協議会，くまもと技術革新・融合研究会,熊本県情報サービス産業協会，ソフトウェアテスト技術振興協会，アジャイルプロセス協議会、オープンソースソフトウェア協会，情報サービス産業協会，情報処理学会，ソフトウェア・メインテナンス研究会，電子情報通信学会，日本ソフトウェア科学会，組込みシステム技術協会，日本 SPI コンソーシアム，日本ファンクションポイントユーザ会，派生開発推進協議会，日本科学技術連盟，組込みソフトウェア管理者・技術者育成研究会，TOPPERS プロジェクト，PMI 日本支部

スタッフ一覧

実行委員会

実行委員長

荒木 啓二郎（熊本高等専門学校）
富松 篤典（電盛社）

実行委員

芦原 秀一（QUEST）
伊藤 昌夫（ニルソフトウェア）
小笠原 秀人（千葉工業大学）
小楠 聡美（HBA）
河北 隆生（元熊本県産業技術センター）
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.）
萩原 美穂（アートシステム）
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（フリー）
本多 慶匡（東京エレクトロン）
宮田 一平（SHIFT）
三輪 東（SCSK）
村上 純（熊本高等専門学校）

プログラム委員会

プログラム委員長

栗田 太郎（ソニー）
末吉 敏則（熊本大学）

プログラム委員

秋山 浩一（富士ゼロックス）
安達 賢二（HBA）
鰺坂 恒夫（和歌山大学）
天嵜 聡介（岡山県立大学）
石井 宏昌（SCSK 九州）
伊藤 昌夫（ニルソフトウェア）
臼杵 誠（富士通）
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）
神谷 年洋（島根大学）
喜多 敏博（熊本大学）
北須賀 輝明（広島大学）
久我 守弘（熊本大学）
日下部 茂（長崎県立大学）
楠本 真二（大阪大学）
河野 哲也（ディー・エヌ・エー）
後藤 徳彦（NEC ソリューションイノベータ）
古畑 慶次（デンソー）
小林 展英（デンソークリエイト）
阪井 誠（SRA）
酒匂 寛（デザイナーズデン）
佐原 伸（法政大学）
菅原 広行（ソニー）
鈴木 裕信（usp.lab.）

鈴木 正人（北陸先端科学技術大学院大学）
鈴木 克明（熊本大学）
高木 智彦（香川大学）
田中 康（奈良先端科学技術大学院大学）
張 漢明（南山大学）
角田 雅照（近畿大学）
土肥 正（広島大学）
富松 篤典（電盛社）
中谷 多哉子（放送大学）
中西 恒夫（福岡大学）
中野 裕司（熊本大学）
中森 博晃（パナソニック）
西 康晴（電気通信大学）
根本 紀之（東京エレクトロン）
野村 行憲（フリー）
野呂 昌満（南山大学）
端山 毅（NTT データ）
久住 憲嗣（九州大学）
藤川 昌雄（富士通九州ネットワークテクノロジーズ）
松尾谷 徹（デバッグ工学研究所）
松本 健一（奈良先端科学技術大学院大学）
水野 修（京都工芸繊維大学）
宮本 祐子（宇宙航空研究開発機構）
宗平 順己（Kyoto ビジネスデザインラボ）
村上 純（熊本高等専門学校）
森崎 修司（名古屋大学）
諸岡 隆司（中電シーティーアイ）
八木 将計（日立製作所）
山本 修一郎（名古屋大学）
米島 博司（パフォーマンス・インプルーブメント・アソシエイツ）
劉 少英（法政大学）

ソフトウェア・シンポジウム 2019 in 熊本 目次

■ 未来に開く教育改革

[事例報告]
小中学生を対象としたロボット・プログラミング教育とコンテストの実施
前原 栄輔(NPO 法人 HITO プロジェクト),
菅原智裕(熊本県立技術短期大学校),
久我守弘(熊本大学) …… 1

[事例報告]
社会人リカレント教育 enPiT-everi における熊本大学の取り組み
久我 守弘, 末吉 敏則(熊本大学), 中武 繁寿(北九州市立大学) …… 2

[経験論文]
ソフトウェアプロセスの自己改善は自学自習でも可能なのか？
梅田 政信, 片峯 恵一, 荒木 俊輔, 橋本 正明(九州工業大学),
日下部 茂(長崎県立大学) …… 3

[経験論文]
勉強会を活用した組織成長モデル ～参加型勉強会の適用事例～
伊藤 修司(SCSK 株式会社) …… 13

■ 要求抽出・形式仕様記述応用

[研究論文]
要件定義計画を強化するアセスメント項目の提案
越前谷 達朗(株式会社日立ソリューションズ・クリエイト),
久保 光寛(日本システム技術株式会社),
渋谷 公寛(東京海上日動システムズ株式会社),
新田 史弥(株式会社東光高岳), 吉竹 宏幸(TIS 株式会社),
石川 冬樹(国立情報学研究所), 栗田 太郎(ソニー株式会社) …… 19

[研究論文]
新規製品開発時の発想支援ツールの提案
辻脇 優一
中谷 多哉子(放送大学) …… 29

[研究論文]
日本語非機能要件の自動分類における教師あり学習アルゴリズムの評価
大東 誠弥, 福井 克法, 宮崎 智己, 大平 雅雄(和歌山大学) …… 36

[研究論文]
軽量形式手法 VDM によるバーチャルマシンの開発
小田 朋宏(株式会社 SRA), 荒木 啓二郎(熊本高等専門学校) …… 46

■ ソースコード解析の応用

[研究論文]
コードクローンへの欠陥混入防止に向けた欠陥混入クローンの特徴分析
野口 耕二朗, 大平 雅雄(和歌山大学) …… 56

[研究論文]
組み込みソフトウェアにおけるコードクローン出現に関する考察
若林 奎人, 水野 修(京都工芸繊維大学) …… 66

[研究論文]
CNN-BI システムによるプログラムの不具合発見の検証
小川 一彦, 中谷 多哉子(放送大学) …… 75

[研究論文]
LSTM を用いたソースコード内の演算子推定手法
舟山 優, 水野 修(京都工芸繊維大学) …… 84

■ ふりかえり・問題解決

[事例報告]
ソフトウェアエンジニアリングにおけるコミュニケーション技術の活用
艸薙 匠，田村 朱麗，藤原 聡子(株式会社東芝) …… 94

[事例報告]
コンセプト＆ゴール指向のふりかえりの提案
小楠 聡美(株式会社 HBA) …… 96

[事例報告]
＜違い＞を捉えよう ～違いの分析から始める問題解決～
常盤 香央里(WACATE 実行委員会) …… 104

[経験論文]
なぜなぜ分析とシステム理論に基づく STAMP/CAST の事例による比較
～ソフトウェアメンテナンスプロジェクトでの問題の分析事例に基づく比較～
日下部 茂(長崎県立大学)，三輪 東(SCSK 株式会社) …… 106

■ IoT 関連技術と応用

[事例報告]
IoT システム開発におけるエッジデバイスソフトウェア開発の難しさ
阪井 誠(株式会社 SRA) …… 114

[研究論文]
テンソル分解プログラミングの理解支援のための立体パズルの利用
山本 直樹，村上 純，石田 明男(熊本高等専門学校) …… 115

[研究論文]
マイクロサービス開発への SOA 開発プロセスの拡張可能性の検討
宗平 順己(Kyoto ビジネスデザインラボ) …… 125

[研究論文]
データベース・アプリケーションへの実行時モデル検査導入手法の提案
宮永 照二(株式会社テクノネット) …… 135

■ 予測・テスト設計

[研究論文]
開発者に着目した Fault-Prediction 技術
高井 康勢, 加藤 正恭, 三部 良太(株式会社日立製作所),
林 晋平, 小林 隆志(東京工業大学) …… 145

[研究論文]
質問に対する回答者推薦手法に用いられるデータの期間についての一検討
眞鍋 雄貴, 西原 弘樹(熊本大学) …… 153

[経験論文]
Web サービス・モバイルアプリケーション開発における
テスト設計を支援するための標準テスト観点の整備
河野 哲也, 柏倉 直樹, 前川 健二, 菊武 祐輔(株式会社ディー・エヌ・エー)
…… 162

■ Future Presentation

[Future Presentation]
システムオブシステムズの現状と課題
落水 浩一郎(UIT, Myanmar), 小笠原 秀人(千葉工業大学), 日下部 茂(長崎県立大学),
艸薙 匠(株式会社), 熊谷 章(金沢諏訪堂の会), 栗田 太郎(ソニー株式会社),
塩谷 和範(ISO/IEC SC7 エキスパート), 新谷 勝利(新谷 IT コンサルティング),
鈴木 正人(北陸先端科学技術大学院大学), 瀬尾 明志(日本ユニシス株式会社),
富松 篤典(株式会社電盛社), 奈良 隆正(コンサル 21 世紀),
羽田 裕(日本電気通信システム株式会社), 方 学芬(株式会社),
Huyen Phan Thi Thanh(株式会社日立製作所), 堀 雅和(株式会社インテック),
矢嶋 健一(株式会社 JTB) …… 172

[Future Presentation]
プログラミング言語横断類似コード断片検出ツールの試作
神谷 年洋(島根大学) …… 176

[Future Presentation]
日本版エンジニアの心理的安全性 ～トリセツ活動その 2 実践編～
増田 礼子(フェリカネットワークス株式会社),松尾谷 徹(有限会社デバッグ工学研究所)
…… 179

■ ソフトウェア・シンポジウム 2019 パンフレット …… 182

小中学生を対象としたロボット・プログラミング教育とコンテストの実施

前原　栄輔
NPO 法人 HITO プロジェクト
eisuke@npo-hitoproject.or.jp

菅原智裕
熊本県立技術短期大学校
sugahara@kumamoto-pct.ac.jp

久我守弘
熊本大学
kuga@cs.kumamoto-u.ac.jp

要旨

2020 年度より小学校で必修化されることとなる「プログラミング教育」を，それに先立つ 2007 年度より地域貢献の一環として熊本県内で実施してきた．また，生徒の成果発表の場として国際ロボットコンテストの熊本地区大会を開催してきた．本稿では，これまでの取り組みについて報告する．

1. はじめに

「ものづくり体験を通した工学分野への興味喚起による素地づくり」および「論理的思考，課題解決能力等の涵養による人材育成」を目的として，小中学生を対象とした「ロボット・プログラミング教育」を，助成金を活用して 2007 年度より熊本県内で実施してきた．また，生徒の成果発表の場として自動制御技術を競う国際ロボットコンテスト“World Robot Olympiad（WRO）”[1]の熊本地区大会を開催してきた．

2. プログラミング教育

2.1. ロボット・プログラミング

LEGO 社の Mindstorms を用いたロボットの作成およびプログラミング開発教材を活用している．LEGO Mindstorms により容易にロボットを作成できると共に，GUI により制御プログラムを開発できるため，プログラムの構造が視覚的にわかりやすく，小学生の初学者でも扱うことが可能である．プログラミングを行う過程において論理的な思考力や対応能力を養うのに適した教材であるといえる．

2.2. 育まれる能力

文部科学省による平成 29 年告示の学習指導要領[2]では，情報活用能力として「プログラミング的思考の育成」を挙げている．また，総務省の報告書[3]では，プログラミングに関する教育がもたらす効果として，「課題解解決力」，「創造力」，「コンピュータの原理に対する理解」が挙げられている．本活動はこれらの能力を身に付けることができる取り組みであるといえる．

3. 活動内容

3.1. 体験教室の実施

ロボットの組み立ておよびプログラミングに興味を持ってもらう取り組みとして，大学生ボランティアの協力のもと体験教室を行っている．1 回の教室は 1~4 日程度の短期講座である．はじめは順次処理でモータの回転数や方向を制御して，正しい手順を組むところから行う．その後，ライントレースを通してくり返し処理と分岐処理を学習し，簡単なプログラミング構造の理解につなげる．

3.2. WRO ロボットコンテスト熊本地区大会の実施

現在 60 以上の国・地域で開催されている自動制御技術を競う国際コンテスト（WRO)の熊本地区大会を開催している．体験教室で興味を持った生徒に対して参加を勧めている．このコンテストは年々難易度が上がり初学者には困難なため，熊本独自の部門を設けるなど参加しやすい配慮もしている．本年度，中学生部門で 1 位となった「.exe」チームは，タイ国で開催された国際大会に出場することができた．順位は 90 位と振るわなかったものの，これまで続けてきた活動により，ロボットの作成およびプログラミングのノウハウを蓄積できた結果であるといえる．

参考文献

[1] WRO Japan 事務局，WRO Japan2018 公式サイト，http://www.wroj.org/2018/.
[2] 文部科学省，小学校学習指導要領（平成 29 年告示）解説 総則編，pp.83-87，2017/7.
[3] 総務省，「プログラミング人材育成の在り方に関する調査研究」報告書，2017/6/5.

社会人リカレント教育 enPiT-everi における熊本大学の取り組み

久我 守弘
熊本大学
kuga@cs.kumamoto-u.ac.jp

末吉 敏則
熊本大学
sueyoshi@cs.kumamoto-u.ac.jp

中武 繁寿
北九州市立大学
nakatake@kitakyu-u.ac.jp

要旨

本稿では，Society5.0 に対応した高度技術人材育成事業として実施する社会人リカレント教育 emPit-everi の概要，および，連携大学として参加する本学の取り組みについて報告する．

1. はじめに

我が国が目指すべき未来社会の姿として，第 5 期科学技術基本計画において Society 5.0 が提唱された[1]．文部科学省は「Society 5.0 に対応した高度技術人材育成事業」enPiT[2]を公募し，社会人向けリカレント教育 enPiT-Pro のひとつとして北九州市立大学を代表校とする enPiT-everi “地域産業の競争力強化を図る人工知能とロボット技術を駆使した IoT 技術の社会実装を推進する実践的人材育成コースの開発・実施”が採択された[3]．本学は，その連携大学の 1 校として参画している．

2. enPiT-everi の概要

enPiT-everi では，製造業，自動車産業，介護業，農業畜産業および観光業の 5 つの分野に特化した教育テーマを設定し，IoT（Internet of Things），AI（Artificial Intelligence），ロボットなどの企業への導入を推進できる高度技術人材を育成することを目的としている．設定されたコースを受講し履修した者は，「IoT アーキテクト」または「IoT エンジニア」として修了認定を行う．

表 1：熊本大学が担当する科目一覧

基盤科目	画像処理
	論理回路
	ハードウェア記述言語入門
応用科目	データマイニングの基礎
	深層学習
	FPGA による組込みシステム技術
	Raspberry Pi による組込みシステム技術
ラボ	製造業 IoT 実践ラボ演習

3. 熊本大学の取り組み

連携大学の 1 校である熊本大学は，表 1 に示す基盤科目，応用科目およびラボ演習を担当する． カリキュラムの中でもラボ演習は各分野において学んだ基盤科目，応用科目の知識を基にして，課題解決型 Project Based Learning を実施するもので，総仕上げとなる科目として位置付けられている．

担当するスマートファクトリ A コースは，製造業分野を想定したカリキュラムとなっており，IoT を実現する上で不可欠な組込みシステムに関する技術を習得することを目標としている．「製造業 IoT 実践ラボ演習」は，IoT を実現するための組込みシステムやサーバの仕組みについて理解を深めると共に，データ収集・分析システムの開発を例として演習を行う．組込みシステムとしてプロセッサ混載 FPGA や ARM プロセッサを搭載した SoC デバイスを使用し，その設計・操作法が学べるようになっている．

4. まとめ

以上，社会人リカレント教育である enPiT-everi の概要，および，熊本大学の取り組みについて述べた．enPiT-everi は 2018 年度後期のパイロット開講を経て，本年度より開講されており，Society5.0 に向けた高度 IoT 技術者の人材育成に貢献していく．

参考文献

[1] 内閣府：科学技術政策 Society 5.0, https://www8.cao.go.jp/cstp/society5_0/index.html.
[2] 文部科学省：成長分野を支える情報技術人材の育成拠点の形成（enPiT）, http://www.mext.go.jp/a_menu/koutou/kaikaku/enpit/index.htm.
[3] enPiT-everi：地域産業の競争力強化を図る人工知能とロボット技術を駆使した IoT 技術の社会実装を推進する実践的人材育成コースの開発・実施, https://www.enpit-everi.jp/.

ソフトウェアプロセスの自己改善は自学自習でも可能なのか？

梅田政信
九州工業大学
umerin@ci.kyutech.ac.jp

片峯恵一
九州工業大学
katamine@ci.kyutech.ac.jp

荒木俊輔
九州工業大学
araki@ci.kyutech.ac.jp

橋本正明
九州工業大学
masaaki.hashimoto@kyudai.jp

日下部茂
長崎県立大学
kusakabe@sun.ac.jp

要旨

パーソナルソフトウェアプロセス (PSP) は，ソフトウェア技術者のための自己改善プロセスである．*PSP for Engineers* コースは，*SEI* が提供する *PSP* トレーニングコースの一つであり，受講者は，講義と演習とを通じて，高品質ソフトウェアの開発に必要なスキルを習得できる．同コースの講義資料等は，*2018* 年 *10* 月より *Creative Commons* ライセンスに基づき入手可能となり，*PSP* を自学自習する環境は整ってきた．しかし，改善提案の根拠となるプロセスデータの正確性と精密性とを自己で判断することは必ずしも容易ではなく，自学自習によるソフトウェアプロセスの自己改善の有効性は明らかになっていない．本論文では，九州工業大学における *PSP for Engineers* コースの *2013* 年～*2018* 年の実施結果に基づいて，*PSP* インストラクタが果たしている役割を定量的に分析し，自学自習によるソフトウェアプロセスの自己改善の課題を明らかにする．

1. はじめに

情報通信技術の発展に伴い，情報システムの重要性は増すばかりであり，ソフトウェアの品質は，ビジネス上の成功や安心・安全な社会を実現する上で不可欠な要素となっている．そのため，ソフトウェアの品質や開発生産性の向上を目的として，形式手法やモデル駆動開発等の様々な手法やツールが提案されている [1]．しかし，どのような手法やツールを用いたとしても，ソフトウェアの品質を最終的に担保するのはソフトウェア技術者であり，ソフトウェアの品質改善にソフトウェア技術者のスキル向上が不可欠なことに変わりはない．

パーソナルソフトウェアプロセス (PSP) [2, 3] は，米国カーネギーメロン大学ソフトウェアエンジニアリング研究所 (SEI) の Watts S. Humphrey により開発されたソフトウェア技術者のための自己改善プロセスである．PSP for Engineers コースは，企業の実務経験者を主な対象として，SEI が提供する PSP トレーニングコースの一つである．受講者は，PSP インストラクタによる講義とプログラム開発の演習，および自己分析レポートを通じて，高品質ソフトウェア開発に必要なスキルを習得できる．

従来，同コースに関する講義資料等は，SEI とのパートナー契約に基づいて利用可能な他，一定の条件下で Web サイト [4] からも一部入手可能であった．しかし，2018 年 10 月より Creative Commons ライセンス [5] の基に SEI Digital Library [6] から無償で入手可能となっている．そのため，ここより入手した講義資料と市販の書籍等とを組合せて PSP を自学自習する環境が整ったとも言えるが，これらの資料等に基づいて演習を行うだけでは，自己改善のスキルを習得することは必ずしも容易ではない．実際，自学自習により改善効果が得られないことから，PSP の有効性に疑問を呈する意見もある [7]．

PSP for Engineers コースにおいて，PSP インストラクタは，講義により単に知識を伝達するだけでなく，演習結果に対する指導や助言を通じて，効率的で効果的なソフトウェアプロセスの改善を促すことが期待されてい

る．しかし，ソフトウェアプロセスの自己改善において，PSP インストラクタが果たしている役割を定量的に分析，評価した例は知られていない．そこで本論文では，九州工業大学の大学院生を対象とした PSP for Engineers コースの実施結果に基づいて，PSP インストラクタが果たしている役割を定量的に分析し，自学自習によるソフトウェアプロセスの自己改善の課題を明らかにする．

以下では，まず 2 節で PSP for Engineers コースの概要を述べ，3 節で九州工業大学の大学院教育に導入した PSP コースとそれによるプロセス改善結果について述べる．次に，4 節で PSP コースの実施結果に基づいて PSP インストラクタの役割を定量的に分析し，5 節で自学自習によるソフトウェアプロセスの自己改善の課題について考察する．

2. PSP for Engineers コースの概要

2.1. PSP for Engineers コースの構成

ソフトウェアの品質は，それを構成する最低品質の部品によって決まり，各部品の品質は，それを開発した個人とその時に用いたプロセスの品質によって決まる．そのため，ソフトウェアの品質改善には，個人のスキル改善が不可欠である．

PSP は，ソフトウェア技術者のための自己改善プロセスである．図 1 は，PSP におけるプロセスの発展，および PSP とチームソフトウェアプロセス (TSP)[8, 9] との関係を図示したものである．同図中，PSP0，PSP0.1 においては，欠陥記録，時間記録，規模測定，改善提案，および，これらを確実に実施できる規律の重要性を学ぶ．PSP1，PSP1.1 においては，要求の実現に必要な部品の同定と，これに基づく規模と時間の見積り，開発計画と進捗追跡を学ぶ．PSP2，PSP2.1 においては，品質計画，設計とコードのレビュー，および設計テンプレートを用いた設計と検証とを学ぶ．

PSP for Engineers コースは，企業の実務経験者を主な対象として，SEI が提供する PSP トレーニングコースの一つであり，PSP-Planning（PSP0～PSP1.1）とそれに続く PSP-Quality（PSP2～PSP2.1）の 2 つのコース（各 5 日間）からなる．各コースの受講者は，半日の講義後に数 100 行程度の小規模なソフトウェアを開発する演習を行う．また，最終日には講義後に自己分析レポートが課される．受講者は，自身が行った演習に関するプロセスデータを定量的に分析し，それに基づいて自己のソフトウェアプロセスに対する改善提案を行い，次の演習に適用する．この過程を継続的に繰り返すことにより，高品質ソフトウェア開発に必要なスキルの習得を目指す．

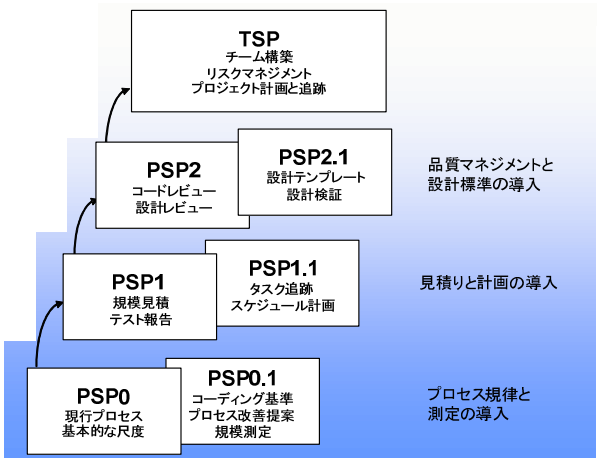


図 1. PSP プロセスの発展

2.2. PSP for Engineers コースの流れ

PSP for Engineers コースを構成する 1 日の講義と演習の大まかな流れを図 2 に UML アクティビティ図として示す．PSP インストラクタは，書籍による事前学習を前提として講義を行い，その中でプログラム開発の演習課題を課す．受講者は，プロセス（PSP0～PSP2.1）に従って，ソフトウェア開発の計画を立案し，計画のレビューを PSP インストラクタに依頼する．PSP インストラクタは，計画の内容を確認し，必要に応じて指導や助言を行う．計画に不備がなければ，受講者は，その計画に基づいてプログラムを開発する．開発が完了すると，受講者は，演習レポートのレビューを PSP インストラクタに依頼する．PSP インストラクタは，演習レポートの内容を確認し，必要に応じて指導や助言を行う．受講者は，必要に応じて演習レポートを修正し，再度のレビューを依頼する．

受講者は，一つの演習課題に着手した時点から演習レポートが受理されるまでの一連の活動の中で，欠陥や時間，規模等に関するプロセスデータを逐次記録する．図 3 と 4 に欠陥記録と時間記録の例を示す．演習中に記録したこれらのプロセスデータは，自己のソフトウェアプロセスに対する改善提案を行うための基礎的なデータであり，もし表 1 に示す欠陥型 [3] の選択誤りや時間記録

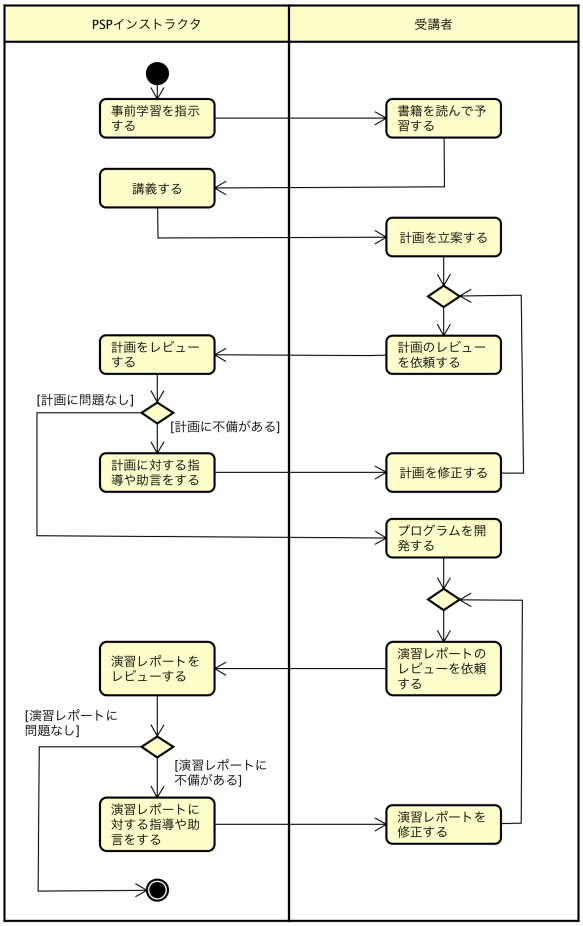
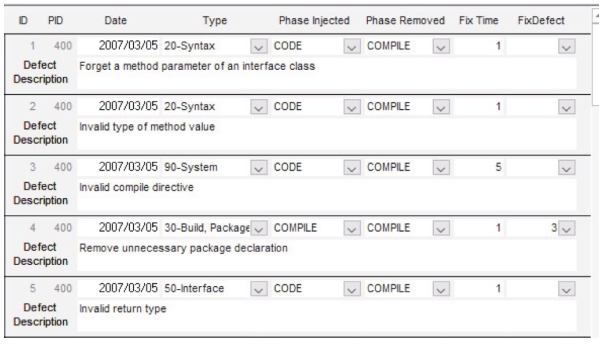
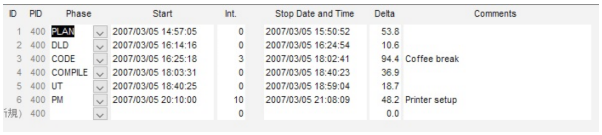


図 2. PSP for Engineers コースにおける講義と演習の流れ



ID	PID	Date	Type	Phase Injected	Phase Removed	Fix Time	FixDefect
1	400	2007/03/05	20-Syntax	CODE	COMPILE	1	
Defect Description	Forget a method parameter of an interface class						
2	400	2007/03/05	20-Syntax	CODE	COMPILE	1	
Defect Description	Invalid type of method value						
3	400	2007/03/05	90-System	CODE	COMPILE	5	
Defect Description	Invalid compile directive						
4	400	2007/03/05	30-Build, Package	COMPILE	COMPILE	1	3
Defect Description	Remove unnecessary package declaration						
5	400	2007/03/05	50-Interface	CODE	COMPILE	1	
Defect Description	Invalid return type						

図 3. 欠陥記録の例



ID	PID	Phase	Start	Int.	Stop Date and Time	Delta	Comments
1	400	PLAN	2007/03/05 14:57:05	0	2007/03/05 15:50:52	53.8	
2	400	DLD	2007/03/05 16:14:16	0	2007/03/05 16:24:54	10.6	
3	400	CODE	2007/03/05 16:25:18	3	2007/03/05 18:02:41	94.4	Coffee break
4	400	COMPILE	2007/03/05 18:03:31	0	2007/03/05 18:40:23	36.9	
5	400	UT	2007/03/05 18:40:25	0	2007/03/05 18:59:04	18.7	
6	400	PM	2007/03/05 20:10:00	10	2007/03/05 21:08:09	48.2	Printer setup
計規)	400			0		0.0	

図 4. 時間記録の例

表 1. PSP の欠陥型標準

番号	名前	説明
10	Documentation	Comments, messages
20	Syntax	Spelling, punctuation, typos, instruction formats
30	Build, Package	Change management, library, version control
40	Assignment	Declaration, duplicate names, scope, limits
50	Interface	Procedure calls and references, I/O, user formats
60	Checking	Error messages, inadequate checks
70	Data	Structure, content
80	Function	Logic, pointers, loops, recursion, computation, function defects
90	System	Configuration, timing, memory
100	Environment	Design, compile, test, or other support system problems

の漏れ等の正確性や精密性を欠くデータが含まれていると，これがソフトウェアプロセスの改善を阻害する恐れがある．そのため，PSP インストラクタは，演習レポートのレビューにおいて，欠陥記録や時間記録，規模記録等がソフトウェアプロセスの改善に支障がないと判断されるまで，繰り返し指導や助言を行う．

このように，PSP インストラクタは，受講者に対して，講義を通じて単に PSP に関する知識を伝達するだけでなく，演習結果に対する指導や助言を通じて，効率的で効果的なソフトウェアプロセスの改善を促すことが期待されている．

3. 九州工業大学におけるソフトウェアプロセスの改善結果

3.1. 九州工業大学の PSP コースの概要

九州工業大学は，文部科学省「先導的 IT スペシャリスト育成推進プログラム」の一環として，2007 年より

SEI と連携し，PSP と TSP とを大学院教育に取り入れ，情報化社会を牽引する高度情報通信技術者の育成に取り組んでいる [10, 11, 12, 13, 14]．この大学院科目群は，PSP for Engineers コースに基づく PSP コースと，教育向けに設計された Introductory TSP (TSPi) に基づく TSP コースとからなる．

PSP コースは，PSP-Planning と PSP-Quality とに対応する二つの演習科目からなり，PSP for Engineers コースと同様の内容を実施している．ただし，受講者が同時期に他の科目を履修しても演習時間を十分確保できるように，1 日分の内容を一週間で実施し，全体を半期で終えるスケジュールとなっている．

PSP コースの運営は，SEI 認定の PSP インストラクタ資格を有する教員が，SEI のライセンスに基づいて行なっている．このため，同コースの修了者には，SEI が実施するコースと同様に，PSP for Engineers コースの修了証が授与される[1]．

3.2. ソフトウェアプロセスの改善結果

PSP コースは，2007 年度から 2018 年度までに，PSP-Planning とそれに続く PSP-Quality とを，それぞれ 111 名とその内の 99 名とが履修し，95 名と 19 名とが修了[2]している．以下では，PSP-Planning を受講した 111 名とそれに続くに PSP-Quality を受講した 99 名とについて，PSP コースを通じてソフトウェアプロセスがどのように改善されたのか，その概略を示す．

3.2.1 規模と時間の見積り誤差

図 5 は，規模の見積り誤差の最小値，平均値，および最大値の推移を表したものである．横軸は演習課題の番号，縦軸は見積り誤差を表す．この図から，当初，過小見積りの傾向が見られるものの，コースの進捗に伴い誤差が小さくなり，過小と過大とにバランス良く見積れるように変化していることが分かる．

同様に，時間の見積り誤差も，図 6 に示すように，過小と過大とにバランス良く見積れるように変化している．

見積り誤差は，小さいに越したことはないが，複数の部品からなるシステム全体を見積る際には，部品の見積り誤差を互いに相殺できるように過小と過大とにバランスすることの方が重要である．

3.2.2 欠陥密度

図 7，8 は，コンパイルとテストにおいて除去した欠陥の密度（1000 行当たりの欠陥数）の推移を四分位で表したものである．横軸は演習課題の番号，縦軸は欠陥密度を表す．これらの図より，コンパイル欠陥密度は，コースの進捗に伴い着実に減少していることが分かる．また，テスト欠陥密度は，一部の例外はあるものの，コースの進捗に伴い減少傾向を示し，課題 8 における第 3 四分位の欠陥密度は，課題 1 における第 1 四分位の欠陥密度と比べて低くなっている．すなわち，コース受講当初に多数の欠陥を作り込んでいた受講者の多くは，コース受講当初の中では優れたとされる品質と比較して，それを超える改善が見られる．このことは，図 9 に示すように，コンパイル前までの欠陥除去率 (プロセスイールド) が最終的に平均で 70 %を超えていることからも確認できる．

図 10，11 は，開発時間に占めるコンパイル時間とテスト時間の割合の推移を四分位で表したものである．コンパイル時間の割合は，コンパイル欠陥密度の減少に伴い低下している．一方，テスト時間の割合は，テスト欠陥密度の減少程に低下しているとは必ずしも言えない．これは，テストにおいて一つの欠陥を除去する時間にばらつきが大きいことが一因である．

これらの結果は，企業の実務経験者を対象に実施された PSP for Engineers コースの結果と大差なく，小規模なプログラミング演習経験しか有しない大学院生であっても，ソフトウェアの品質向上に必要なスキルを習得可能であり，そのツールとして PSP コースが有効なことを示している．

4. PSP インストラクタの役割分析

4.1. 分析方法

2.2 節で述べたように，PSP インストラクタは，計画に対するレビュー時と演習レポートに対するレビュー時との主に二つの場面で，指導や助言の機会がある．ここでは，演習レポートに対するレビューにおいて，その指導や助言の前後で受講者のプロセスデータが変化することに着目し，その変更記録からソフトウェアプロセスの

[1] 2018 年 10 月より Creative Commons ライセンスに移行するため 2018 年度までとなる．

[2] 修了は，四つの演習課題とその後の自己分析レポート課題を全て終えていることを言う．

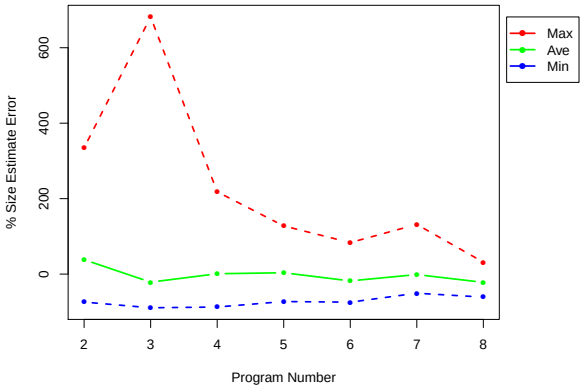


図 5. 規模見積り誤差の推移

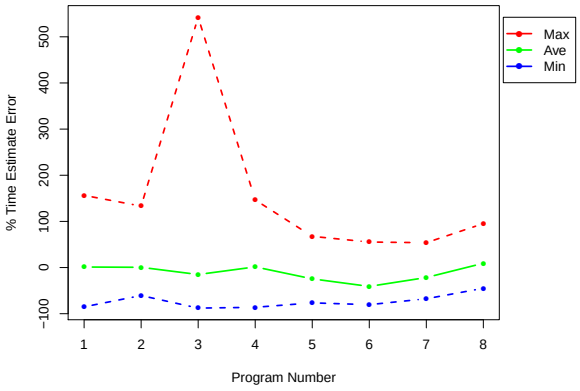


図 6. 時間見積り誤差の推移

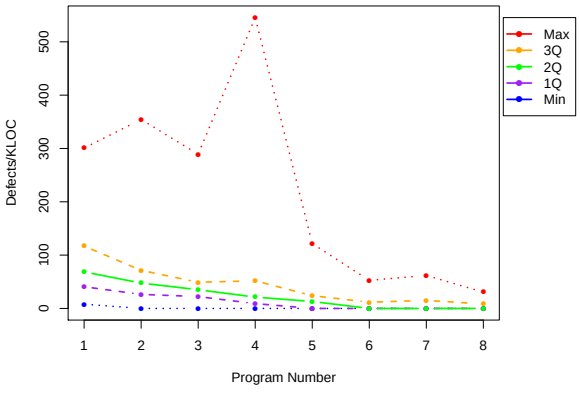


図 7. コンパイル欠陥密度の推移

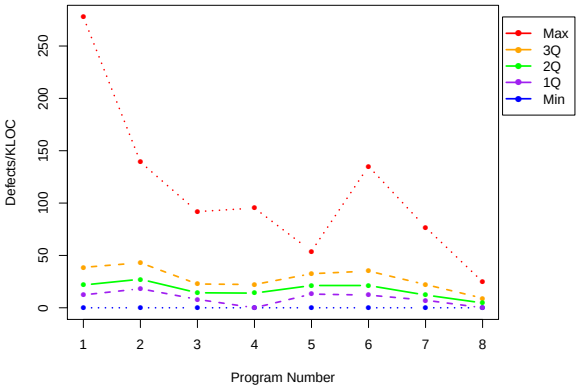


図 8. テスト欠陥密度の推移

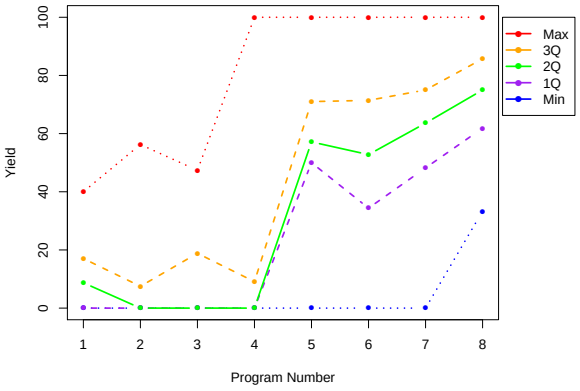


図 9. プロセスイールドの推移

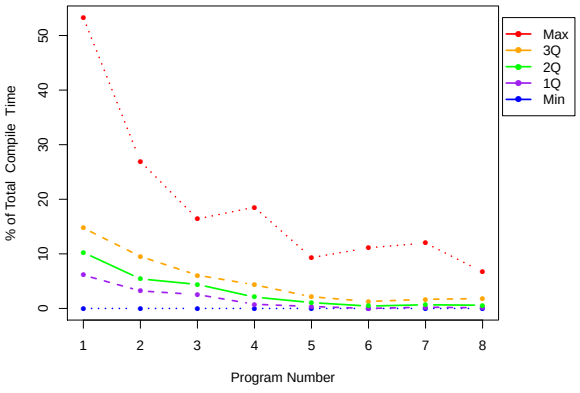


図 10. コンパイル時間割合の推移

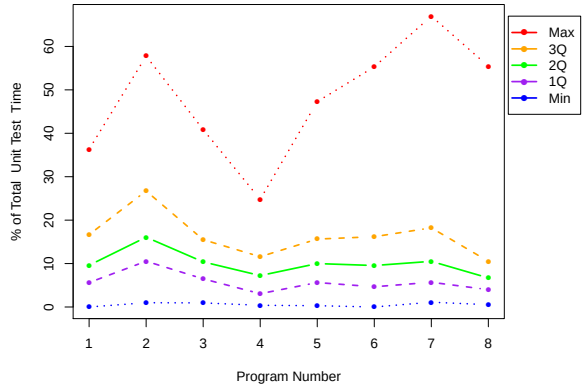


図 11. テスト時間割合の推移

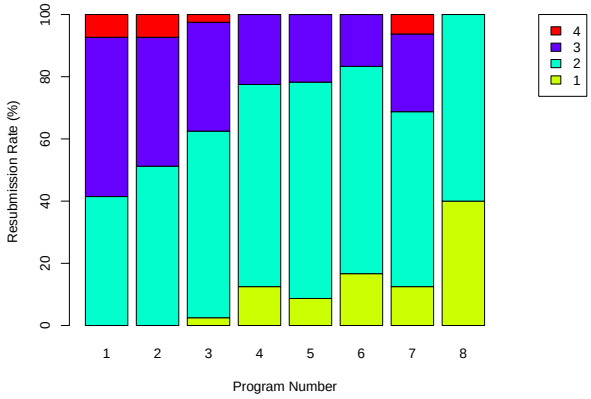


図 12. 演習レポートのレビュー回数の推移

改善に PSP インストラクタが果たしている役割の一部を明らかにする．

分析対象のプロセスデータは，2013 年～2018 年の PSP コース受講者の中で，演習レポートのレビュー依頼時に，その都度プロセスデータの提出を受けた 39 名分である．これらには，レビュー依頼時に間違ったファイルを提出した等の理由により，プロセスデータに不足があるものは含まれていない．

4.2. 分析結果

演習レポートのレビュー回数　図 12 は，一つの演習課題当りの演習レポートのレビュー回数の割合の推移を表したものである．レビュー回数は，最大 4 回[3]であり，コースの進捗に伴い次第に減少している．演習レポートの再提出に至る理由は様々であるが，後述するように，欠陥型の間違いなど，プロセスデータの正確性や精密性に問題がある場合が大半である．このことから，プロセスデータを適切に記録するスキルがコースの進捗に伴い次第に定着していることが確認できる．

欠陥型の修正　図 13 は，欠陥記録において，欠陥型の変更を伴う修正の割合の最小値，平均値，および最大値の推移を表したものである．また，図 14 は修正前の欠陥型別の欠陥数とその累積割合を，図 15 は修正前の欠陥型に対する修正後の欠陥型の型別個数を，それぞれ表したものである．図 13 より，欠陥型の修正は，当初，平均約 35.7%であったものがコースの進捗に伴い減少し，最終

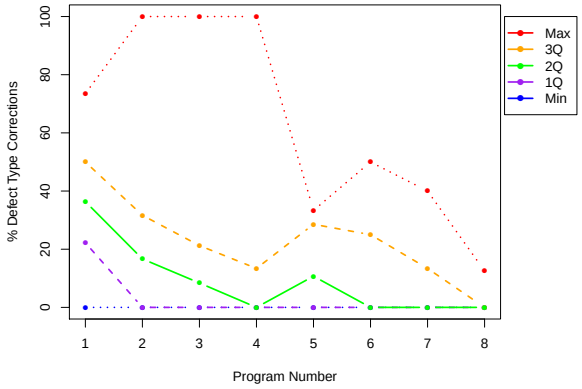


図 13. 欠陥型の修正割合の推移

的に高々12.5%にまで低減している．この結果は，適切な欠陥型を判断するスキルがコースの進捗に伴い定着したことを示している．また，図 14 より，修正前の欠陥型は，20 番 (Syntax) が約 25.7%，70 番 (Data) が 22.8%，40 番号 (Assignment) が約 16.8%を占めており，これらだけで全体の約 65%を占めている．一方，図 15 より，欠陥型 20 番 (Syntax) の誤りの多くは，50 番 (Interface) と 80 番 (Function) であり，たとえば，関数やメソッドの引数の順序やデータ型の誤り等をタイプミス等の構文誤りと取り違えている．

欠陥の埋込フェーズの修正　図 16 は，欠陥記録において，欠陥の埋込フェーズの変更を伴う修正の割合の最小値，平均値，および最大値の推移を表したものである．また，図 17 は，修正前の埋込フェーズに対する修正後の埋込フェーズのフェーズ別個数を表したものである．図

[3]演習レポートを最大 3 回再提出していることになる．

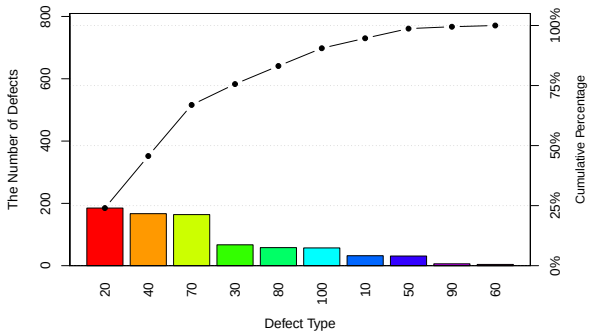


図 14. 修正前の欠陥型の割合

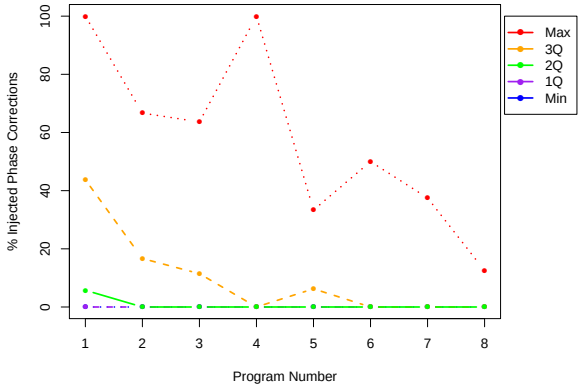


図 16. 埋込フェーズの修正割合の推移

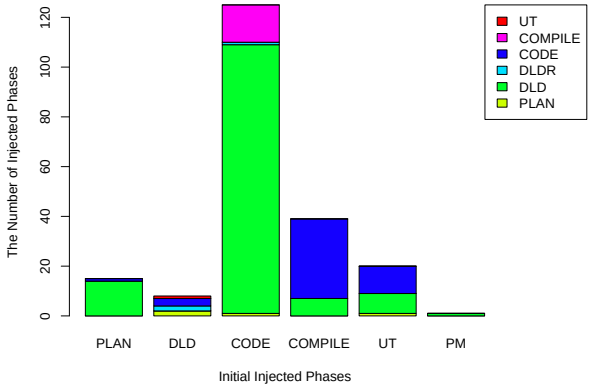


図 17. 修正前の埋込フェーズに対する修正後の埋込フェーズ

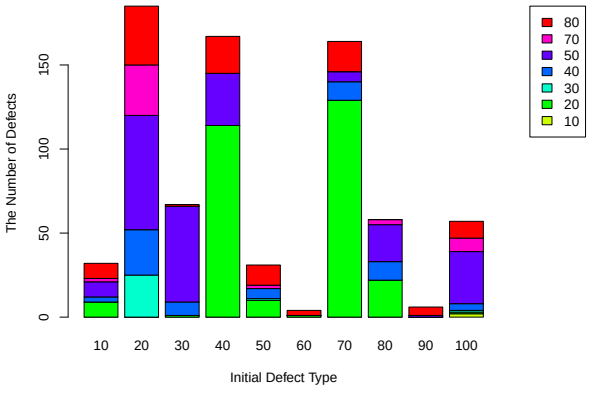


図 15. 修正前の欠陥型に対する修正後の欠陥型

16 より，埋込フェーズの修正は，当初，平均約 20.3%であったものがコースの進捗に伴い減少し，最終的に高々 12.5%にまで低減している．欠陥型の場合と同様に，当初，欠陥の埋込フェーズを誤って判断する場合が少なからずあり，適切な埋込フェーズを判断するスキルがコースの進捗に伴い定着したことを示している．また，図 17 より，欠陥を CODE(コーディング) フェーズで埋め込んだとする誤りが最も多く，その内の 85.5%は DLD(詳細設計) フェーズとすべきものである．

欠陥の除去フェーズの修正　図 18 は，欠陥記録において，欠陥の除去フェーズの変更を伴う修正の割合の最小値，平均値，および最大値の推移を表したものである．また，図 19 は，修正前の除去フェーズに対する修正後

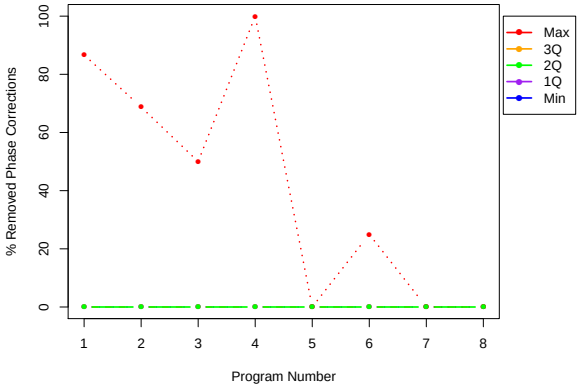


図 18. 除去フェーズの修正割合の推移

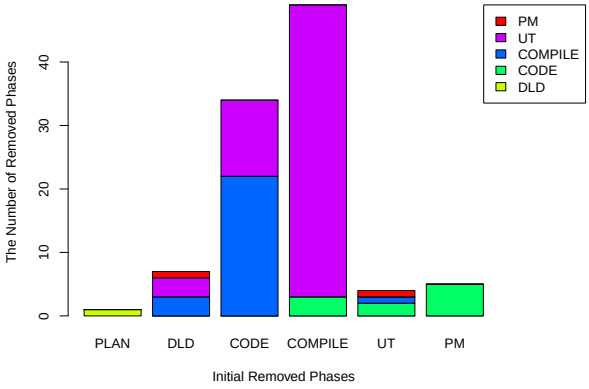


図 19. 修正前の除去フェーズに対する修正後の除去フェーズ

の除去フェーズのフェーズ別個数を表したものである．図 18 より，除去フェーズの修正は，埋込フェーズ程に多くはないものの，一定数あり，適切な除去フェーズを判断するスキルがコースの進捗に伴い定着したことが分かる．また，図 19 より，欠陥を COMPILE(コンパイル) と CODE(コーディング) フェーズで除去したとする誤りが最も多く，それぞれ UT(単体テスト) フェーズと COMPILE(コンパイル) フェーズとすべきものである．これらの誤りの多くは，フェーズと作業内容（行為）との混同[4]である．

[4]たとえば，UT(単体テスト) フェーズにおいて，欠陥を除去するためにプログラムを修正し，コンパイルしても，欠陥を除去したフェーズは CODE や COMPILE ではなく UT である．

5. 自学自習によるソフトウェアプロセス自己改善の課題

5.1. 自学自習による自己改善の限界

欠陥型は，欠陥の埋込に対する対策や欠陥の除去方法を考案する上で重要な手掛りとなる．たとえば，欠陥型の 20 番 (Syntax) に対しては，変数のタイプミスを防止するために，変数一覧を用意し，そこからコピー＆ペーストする等の対策が考えられる．一方，50 番 (Interface) に対しては，関数引数の順序やデータ型の間違いを上流工程で除去するために，設計レビュー用のチェックリストを更新する等の対策が考えられる．このように，欠陥の防止や除去のための対策が欠陥型により異なるため，欠陥型の誤りは，自己改善の阻害要因となる可能性が高い．

一方，欠陥の埋込フェーズは，欠陥の埋込に対する対策をどのフェーズで行うべきか，また，欠陥の除去フェーズは，欠陥の除去をどのように行うべきか，またそもそも欠陥の埋込はどのように防ぐべきだったのかを，それぞれ判断する上で重要な手掛かりとなる．欠陥の埋込に対する対策は，たとえば，設計時とコーディング時とでは一般的に異なる．このため，欠陥型と同様に，埋込フェーズや除去フェーズの誤りは，自己改善の阻害要因となる可能性が高い．

このように，欠陥型や欠陥の埋込フェーズ等のプロセスデータは，自己改善の手がかりとなるものであり，正確性や精密性を欠けば自己改善に支障を来す恐れがある．しかし，4.2 節の分析結果は，プロセスデータの正確性や精密性を受講者が単独で判断することが容易ではないこと，また，プロセスデータの妥当性を判断するスキルが PSP インストラクタからの指導や助言を通じて向上し定着していることを示している．これらの結果は，ソフトウェアプロセスの自己改善を PSP インストラクター等の第三者の介入なく自学自習のみで行うことの限界を示している．

5.2. PSP 教育コースと PSP インストラクタの品質保証

PSP の自学自習は，Creative Commons ライセンスに移行した講義資料等と市販の書籍とを用いて可能になった．そのため，自学自習によりソフトウェアプロセスの自己改善に取り組む事は，これまで以上に容易である．しかし，前述したように，PSP インストラクタによる指

導や助言は，自己改善に重要な役割を果しており，自学自習のみによる自己改善には限界があることも明らかとなった．したがって，効率的で効果的な自己改善には，PSP for Engineers コースと PSP インストラクタによる指導や助言とが不可欠である．

一方，Creative Commons ライセンスへの移行に伴い，SEI による PSP インストラクタの認定制度も終了した．このため，PSP for Engineers コースの内容や実施方法，PSP インストラクタの役割等も自由化された．しかし，効率的で効果的な自己改善には，PSP for Engineers コースと同等な PSP 教育コースとそれを実施する PSP インストラクタの品質保証は欠かせない．この品質保証の枠組み作りは，PSP/TSP コミュニティで取り組むべき今後の大きな課題である．

6. おわりに

本論文では，PSP for Engineers コースに基づく九州工業大学の PSP コースを対象として，ソフトウェアプロセスの改善結果を示し，次に，その改善結果を導いた PSP インストラクタの役割をプロセスデータの変更記録に基づいて分析した．その結果，欠陥型や欠陥の埋込フェーズ等のプロセスデータの正確性と精密性とを自己で適切に判断することは必ずしも容易ではなく，ソフトウェアプロセスの自己改善に，PSP インストラクタによる指導や助言が重要な役割を果していることを定量的に明らかにした．

PSP/TSP に関連する講義資料等が Creative Commons ライセンスに移行したことに伴い，Team Process Community of Practice (TP CoP) [15] において PSP/TSP の今後の展開について議論されている．関係各位の積極的な参加を期待したい．また，俊敏さと高品質とを兼ね備えたソフトウェア開発手法の重要性は益々高まっており，今後は，TP CoP とも連携しながら，PSP/TSP に Agility を取り入れた手法の開発に取り組む必要がある [16, 17]．

また，PSP コースは，演習が途中で停滞してコースを修了できない受講者が一定数あり，修了率は 20%〜30% 程度に止まっている．新しいスキルが定着するまでコースを継続できるかどうかは，受講者の動機づけが深く関わっていると考えられるため，動機づけに着目した教育手法の改善も今後の重要な課題である [18, 19, 20, 21, 22, 23]．

謝辞

九州工業大学への PSP/TSP 関連科目の導入は，文部科学省「先導的 IT スペシャリスト育成推進プログラム」の支援によるものである．PSP/TSP 関連科目の導入を主導された秋山義博客員教授，ならびに日頃よりご支援頂く関係各位に感謝申し上げます．

参考文献

[1] 栗田太郎：モバイル FeLiCa のソフトウェア開発における品質確保のための構造と実践-抽象度の制御やコミュニケーションの活性化に向けて-, 情報処理学会デジタルプラクティス, Vol. 1, No. 3, pp. 148–157 (2010).

[2] Humphrey, W. S.: *A Discipline for Software Engineering*, Addison-Wesley (1995), (邦訳:松本 正雄 監訳, ソフトウェア品質経営研究会訳:『パーソナルソフトウェアプロセス技法』, 共立出版, 1999 年).

[3] Humphrey, W. S.: *A Self-Improvement Process for Software Engineers*, Addison-Wesley (2005), (邦訳:秋山 義博 監訳, JASPIC TSP 研究会訳:『PSP ガイドブック ソフトウェアエンジニア自己改善』, 翔泳社, 2007 年).

[4] Software Engineering Institute, : PSP Academic Material, http://www.sei.cmu.edu/tsp/tools/academic (2013).

[5] Creative Commons, : クリエイティブ・コモンズ 表示 4.0 国際パブリック・ライセンス, https://creativecommons.org/licenses/by/4.0/legalcode.ja (2019).

[6] Software Engineering Institute, : Team Software Process (TSP) and Personal Software Process (PSP) Materials, https://www.sei.cmu.edu/go/tsp (2019).

[7] 海谷治彦, 小島彰, 海尻賢二：大学におけるプロセス改善教育のあり方について –PSP 法実践の経験をもとに–, in *Software Symposium 2001 Proceedings*, pp. 137–142 (2001).

[8] Humphrey, W. S.: *Introduction to the Team Software Process*, Addison-Wesley (1999), (邦訳:秋山義博 監訳, JASPIC TSP 研究会訳:『TSPi ガイドブック』, 翔泳社, 2008 年).

[9] Humphrey, W. S.: *TSP Leading a Development Team*, Addison-Wesley (2005).

[10] 秋山義博, 片峯恵一, 梅田政信, 橋本正明, 乃万司：九州工業大学におけるパーソナルソフトウェアプロセス教育ーソフトウェア品質向上のためのスキル修得ー, *SEC Journal*, Vol. 6, No. 3, pp. 118–125 (2010).

[11] Katamine, K., Umeda, M., Hashimoto, M. and Akiyama, Y.: Changing Software Management Culture from Academic, in *TSP Symposium 2011 Proceedings*, pp. 12–18 (2011).

[12] Katamine, K., Umeda, M., Hashimoto, M. and Akiyama, Y.: A STRATEGY IN EFFECTIVE TEACHING OF SOFTWARE ENGINEERING PROCESS FOR GRADUATE STUDENTS, in *The Proceedings of IADIS International Conference Information Systems 2012*, pp. 259–266 (2012).

[13] 梅田政信, 片峯恵一：PSP/TSP による実践的な ICT 人材育成の取り組み, 情報処理学会誌, Vol. 53, No. 10, pp. 1084–1087 (2012).

[14] Umeda, M., Katamine, K., Hashimoto, M. and Akiyama, Y.: Improving Introductory TSP for Creating High Performance Student Teams, in *TSP Symposium 2013* (2013).

[15] TP CoP Working Groups, : TEAM PROCESS COMMUNITY OF PRACTICE, https://www.tpcop.org (2019).

[16] 片峯恵一, 梅田政信, 橋本正明：PSP/TSP を基礎とした軽量なソフトウェア開発手法に関する取り組み, プロジェクトマネジメント学会 2017 年度秋季研究発表大会予稿集, pp. 131–132 (2017).

[17] 片峯恵一, 梅田政信, 荒木俊輔, 橋本正明：TSP の俊敏性向上に関する取り組みと評価, ソフトウェア品質シンポジウム 2017 (SQiP 2017), pp. B3–2 (p.6) (2017).

[18] Ishibashi, K., Hashimoto, M., Umeda, M., Katamine, K., Yoshida, T. and Akiyama, Y.: A Preliminary Study on Formalization of Motivation Process in Personal Software Process Course, in *The Proceedings of the 10th Joint Conference on Knowledge-Based Software Engineering*, pp. 128–137 (2012).

[19] Umeda, M., Katamine, K., Ishibashi, K., Hashimoto, M. and Yoshida, T.: Motivation Process Formalization and its Application to Education Improvement for the Personal Software Process Course, *IEICE Transactions on Information and Systems*, Vol. E97-D, No. 5, pp. 1127–1138 (2014).

[20] 田頭薫, 片峯恵一, 梅田政信, 石橋慶一, 橋本正明：動機づけプロセスの状態遷移モデルを用いた PSP コース受講生の分析, プロジェクトマネジメント学会 2016 年度秋季研究発表大会予稿集, pp. 247–251 (2016).

[21] 日下部茂, 梅田政信, 片峯恵一, 石橋慶一：ソフトウェアプロセス教育向け動機づけモデルをシステム理論に基づく STAMP/STPA により効果的に活用する手法の提案, プロジェクトマネジメント学会 2017 年度秋季研究発表大会予稿集, pp. 301–306 (2017).

[22] Kusakabe, S., Umeda, M., Katamine, K. and Ishibashi, K.: Managing Personal Software Process Education Course Based on Motivation Process Model by Using System-Theoretic Method STAMP/STPA, in *Proceedings of the 11th International Conference on Project Management (ProMAC2017)*, pp. 1066–1072 (2017).

[23] 日下部茂, 梅田政信, 片峯恵一, 石橋慶一：システム理論に基づくモデリングと質的研究を併用したソフトウェアプロセス教育の動機づけシナリオ改善, ソフトウェア・シンポジウム 2018 論文集, pp. 100–107 (2018).

勉強会を活用した組織成長モデル～参加型勉強会の適用事例～

伊藤 修司
SCSK 株式会社
Shuuji.Itou@scsk.jp

要旨

本論文は，「参加型勉強会」を活用して，新しい組織成長モデル構築を試みた事例を紹介する．「参加型勉強会[1] [2]」とは，組織全体で，3 年間に渡り実践した，グループワーク主体の勉強会である．

この勉強会を立ち上げた目的とねらい，業務と並行で長期にわたり継続するための工夫や，実際に組織にどんな影響を及ぼし，どんな課題が明らかになったのかについて，評価と考察を述べる．

一定の成果を出して，さらなる成長を目指す組織，タックマンモデル[3]に例えると，機能期（遂行期）の段階に入った組織が，さらに成長していくために適用した事例である．類似の課題に直面している組織が，解決に向けたアプローチの一例として，参考にしていただくことを期待する．

1. はじめに

公共性の高い基幹システムを保守・運用する組織やチームは，システムが稼働し続ける限り，もしくは契約が続く限り，経験を有するメンバー中心の体制を維持しなければならない．故に，長年にわたり，同じメンバーで仕事を継続することが多くなり，知の循環や継承の機会に対して，優先度が下がることが多い．このような環境で，組織が継続的に成長していくためには，どのような方法があるのか．その解決策の一つとして，組織的な取り組みとして，グループワーク中心の勉強会を実践した事例を紹介する．

2. 背景と解決したい課題

我々の組織は，金融機関の基幹系システムを，長年にわたり，エンハンス開発しながら保守を継続している組織である[4]．我々の組織は，システム改変と品質改善活動を繰り返しながら，成長の階段を上ってきた．タックマンモデル[3]に例えると，組織は，形成期→混乱期→統一期を経て，機能期（遂行期）の段階を迎えていると言える．

機能期を迎えた組織は，解決しなければならない大きな課題と直面した．年々重厚になる管理プロセスや開発プロセス，膨大なチェックリストやテストケース，プロセス偏重の再発防止策．本当にこのままで良いのだろうか．メンバーはそれぞれ立場で不安を感じていた．今の組織にある資産や過去の成功体験が邪魔をして，今までの考え方や仕事の進め方の延長でしか解決策を見いだせなくなっていたことが，この根底にはある．

複雑に絡み合ったシステムと組織の課題と向き合いながら，中長期的に組織を維持・成長させていくためには，組織に変化を起こす必要があると考えた．

3. 課題解決策

組織の外側から変化を起こすために，「スキルアッププラン」と称して，組織のメンバー一人一人に，予算を設定して，外部の研修やセミナーを受講する活動を推奨した．

また，外部の研修やセミナーで得た知見や気づきを，組織の内側やプロジェクトに還元する取り組みも必要と考えた．これが，勉強会を始めた経緯である．「参加型勉強会[1] [2]」は，組織の中に学びの機会を設けて，成長を促し，学びを組織の文化にすることを目指すという，強い信念と決意をもって始めた活動である． 課題解決策の全体像を図 1 に示す．

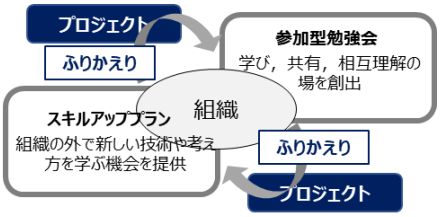


図 1 課題解決策の全体像

3.1「参加型勉強会」 の説明

「参加型勉強会」の特徴，他育成手法との比較，苦労した点と工夫について述べる．

3.1.1「参加型勉強会」 の特徴

チームや個人が，自律的に学び，成長するためのきっかけづくりとして，始めた勉強会で，以下 3 つの特徴がある．

表 1.「参加型勉強会」の特徴

	特徴
1	グループワーク中心で座学よりもアウトプットを重視
2	参加者全員が主役となる場を演出
3	プロジェクトと勉強会を一体運営

(1) グループワーク中心で座学よりもアウトプットを重視

勉強会の構成は，冒頭 30 分をプレゼン+60 分のグループワークを基本とした．グループワークのアウトプットをホワイトボードに残して，勉強会の参加者全員で共有するスタイルとした．短時間でも，成果をアウトプットして，共有することに重きを置いた．毎回，様々なメンバーとグループを組むことで，気づきを得る機会にすることができると考えた．ワークの設計は，フューチャーセンターの方法論[5] を参考にしている．

(2) 参加者全員が主役となる場を演出

プレゼン，グループワークをとおして，参加者は必ず 1 回以上発言する機会を設けた．アイスブレイクで自己紹介，年代の近いメンバーを同じチーム分けとする，グループワーク後に議論の内容を発表して共有するなどである．

(3) プロジェクトと勉強会を一体運営

勉強会を学びのきっかけとするだけではなく，プロジェクトと勉強会が連動しやすいように設計した．具体的には，プロジェクトの課題を，勉強会のテーマに選定したり，勉強会の成果を，プロジェクトに活用したりすることで，学びと成長の PDCA を無意識のうちに実行できるようになった．

3.1.2「参加型勉強会」 と 他の育成手法の比較

3.1.1 で上げた特徴を踏まえて、若手を育成・教育するケースを想定して，a.参加型勉強会と b.座学中心の勉強会，c.OJT の特徴を比較する(表 2)．

表 2. 参加型勉強会と他の勉強会の比較

	評価観点	a.参加型勉強会	b.座学中心の勉強会	c.OJT
1	獲得できる知識の量	少ない	多い	多い
2	発言の機会	**多い**	少ない	多い
3	対話や想いを共有する機会	**多い**	少ない	少ない
4	仕事に対するマインドに触れる機会	**多い**	少ない	少ない
5	担当以外の領域を体験する機会	**多い ※疑似体験**	少ない	少ない
6	若手が教える機会	**多い**	少ない	少ない

a.参加型勉強会の強みは，No.2 発言の機会，No.3 間の対話や想いを共有する機会，No.4 仕事に対するマインドに触れる機会が多いことである．また，若手が自ら教える側に回る機会があるというのも，特徴である．

3.1.3 勉強会を開始する際に苦労した点

この活動を開始するにあたり，苦労した点を 2 つ示す．

(1) 長くは続かないだろうという思い込みを払拭すること

過去にも何度か勉強会を開催していたが，いずれも一過性の活動で終わっていた．続かない原因としては，目的やゴールを明示できなかった点があると推察した．

今回は，活動の理念とゴールやグランドルールを定めて，組織全体で共有した．詳細は，3.1.4 勉強会を継続するための工夫で述べる．

(2) 周囲を巻き込むこと

業務多忙を理由に，企画者がいない，参加者も集まらないという状況を，なんとしても避けなければならないと考えた．できるだけ多くのメンバーを巻き込んで活動を展開するために，勉強会を，組織の戦略的な活動として位置づけ，全体会議で活動の目的を組織のメンバー全員と共有した．

また，勉強会運営委員チームを立ち上げて，参加を募った．育成の趣旨で，若手が企画・運営を担う方式とし，中堅層以上の支援を引き出しやすくした．

3.1.4 勉強会を継続するための工夫

この種の活動は，立ち上げ当初は，一定の成果が見込めるが，時間の経過とともに，目的意識が薄れ，業務多忙を理由に準備する時間がなくなり，形骸化することが多い．しかし，文化の醸成には時間が必要である．そこで，勉強会を継続するために，取り込んだ工夫と，そのねらいを示す(表 3)．

表 3. 勉強会を継続するための工夫

	工夫	ねらい
1	活動の理念とゴールを定めた	活動を継続していく中で，本来の目的を見失わないために示す
2	グランドルールを設けた	グループワークで，参加者が発言しやすい雰囲気を作り出す
3	ふりかえりを活用	勉強会で得た気づきを自らの学びにつなげることを意図
4	ランチ開催	参加者の多様性を確保 雰囲気づくりとして活用
5	若手育成の場として活用	この活動を通して，若手と中堅層が協働する
6	勉強会の成果をプロジェクトで明示的に活用	活動の成果を可視化する
7	報告レポートを作成して組織の資産として共有	参加者の想いや考えを記録し，組織の資産いつでも引き出せるように

表 3 の中で，特に重要と考えた点について補足する．

(1)活動の理念とゴール

我々は，勉強会を開始するにあたり，活動の理念とゴールを設定し，勉強会開催の都度，メンバー全員で共有した(図 2)．なぜなら，この活動に関わる全てのメンバーが，活動の目的を目に見える形で，手に取るように理解することと，活動を継続し

ていく中で，本来の目的を見失わないために繰り返し確認することが重要と考えたからである．

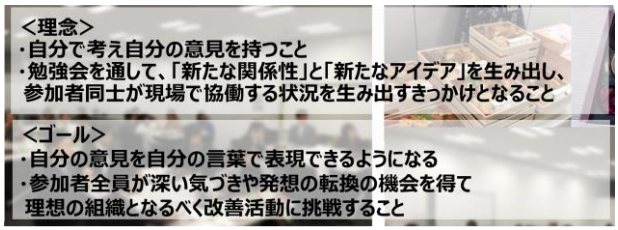


図 2．参加型勉強会の理念とゴール

(2)グランドルール

グループワークで，参加者が発言しやすい雰囲気を作り出すために，グランドルールを設けた(図 3)．

これらは，既存メンバーに対してだけでなく，組織に新たなメンバーを迎え入れる際や，他の組織の方に，この活動の趣旨を説明し，理解していただく際のツールとしても機能した．

1．一人ひとりの**想い**を大切にする
議論が止まる批判的意見はNG
継続的な判定意見はWelcome

2．**発言**すること
お互いに発言機会を提供しあい、よい**関係性**を
つくりあげる 「分からない」というだけでもOK

3．**時間厳守**
定刻に初めて定刻に終える

図 3．参加型勉強会のグランドルール

(3)ふりかえりを活用

毎回，勉強会後に，KPT[6] を使ったふりかえりシートを記入して，結果を一覧化して組織の資産として保管・共有した．この情報をチームやプロジェクトの改善活動のインプットとして役立てることができると考えたからである．

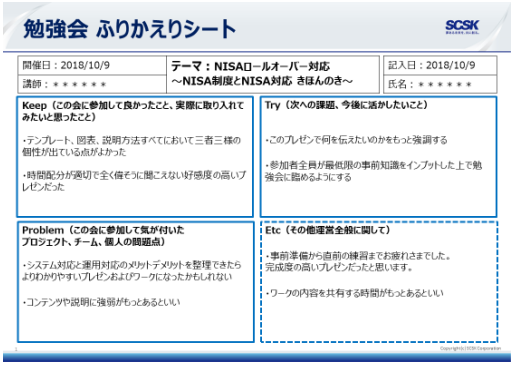


図 4．ふりかえりの例

(4) ランチ開催

業務後の開催では，一部の特定メンバーだけの会になってしまうことを懸念して，多様なメンバーが参加できるように，ランチタイムを活用した．お弁当付きの勉強会とすることで，勉強会参加者を一定数以上確保することを意図した．これは，きっかけが，お弁当であっても，参加して気づきを得ることができればよいと考えたからである．また，一緒にお昼を食べることで，勉強会が堅苦しい時間にならないようにする意図もあった．

3.2 勉強会開催の流れとこれまでの開催実績

3.2.1 勉強会開催の流れ

テーマ選定から，実際に勉強会を開催するまでの流れを，示す(図 5)．テーマの選定を別にすると，企画チーム発足から，勉強会開催までの期間はおよそ 1 か月間が目安となる．

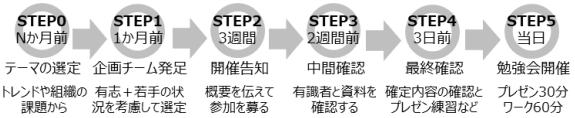


図 5．勉強会開催の流れ

STEP1：テーマの選定は，この流れの中で最も重要なタスクである．顧客動向、業界トレンド，組織やチームやプロジェクトの課題，新たなプロジェク立上げなど，様々な事象がテーマとなり得る．勉強会というキーワードに素早く反応し，具現化するために行動につなげることができるかどうかが，ポイントである．テーマをストックして，時期が来たら開催というケースもある．

STEP2～STEP6：企画チーム発足から，勉強会開催までの期間はおよそ 1 か月間である．準備期間中は，事前教育の位置づけで，テーマに詳しいメンバーから情報を引き出す機会を設けた．短い期間の中で，有識者からの協力を多く引き出すことができるか否かが，勉強会成功のポイントとなる．

3.2.2 これまでの開催実績

過去 3 年間の勉強会開催実績(表 4)と実際の勉強会の様子(図 6)を示す．

表 4．勉強会開催実績

対象組織の規模	延べ参加人数(参加率)	主要テーマ	
1 年目 約 60 名	約 130 人 (55%)	1	自分達の目指すゴールを考えてみる
		2	組織の強み弱み～SWOT 分析～
		3	ブロックチェーンに習え～イノベーションについて考える～
		4	投資信託と投信システムについて学ぶ
2 年目 約 50 名	約 150 人 (71%)	1	つみたて NISA って何だ？
		2	株式取引と株式システムについて学ぶ
		3	インフラチームの仕事を知る
		4	株式等決済期間短縮化対応 対応の背景
3 年目 約 50 名	約 210 人 (47%)	1	NISA ロールオーバーって何？
		2	現状をちょっと良くするためのマインド～プロマネに学ぶ～
3 年目 約 100 名		3	事例に学ぶ～セーフティ・セキュリティ～
		4	株式等決済期間短縮対応 きほんのき
		5	貸株取引 きほんのき

1～2 年目は，同規模の組織に対して，年間 4 回のペースで実施した．3 年目になり，対象組織の規模を拡大し，より大規模な勉強会を開催した．

図 6．参加型勉強会の様子

4. 成果の評価と考察

この活動の成果を，勉強会参加者を対象に実施したアンケート結果と，勉強会後に実施したふりかえり内容の変化から，評価，考察する．

4.1 アンケート結果の評価

3 年間に渡り，勉強会に参加したメンバーに対して，勉強会の満足度，組織にどんな影響があったか等の簡単なアンケートを実施した．アンケートの対象と方式は以下のとおり．

(1)アンケートの対象：3 年間に渡り継続して，本勉強会の対象となった組織（約 50 名）．

(2)アンケートの方式：それぞれの設問に対して，「1.思わない（不満）」から「5.思う（満足）」までの，5 段階評価と，回答理由についてコメントによる評価．

(3)アンケートの回答率：回答率は，約 56%（28 名）．

4.1.1 勉強会の満足度と組織への影響

各設問の 5 段階評価結果（図 4）と評価理由（表 5）を示す．

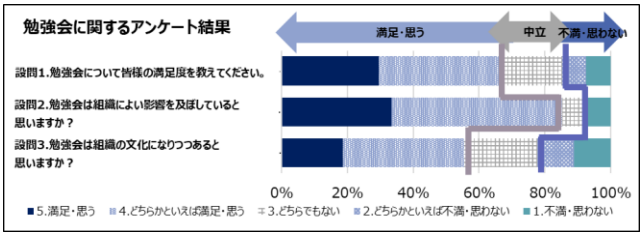


図 7．勉強会に関するアンケート結果

設問 1：「勉強会の満足度」

約 67%のメンバーが，「5.満足」，「4.やや満足」と回答した．一方で，不満と感じているメンバーも，15%程度いることが分かった．

「意見交換やコミュニケーション促進の場として機能している」という，肯定的な意見と，「中堅層以上にも響くテーマが欲しい」，「勉強会後のフォローアップが不足」，「話を聞きたいメンバーが参加していない」など，満足するにはまだ足りないといった意見があった．

表 5．勉強会の満足度

評価	コメント内容
満足 (67%)	他部署，他チームのメンバーと**コミュニケーション**をとることができる有意義な時間である．
	配属直後で不安だった時に，この勉強会をきっかけにして，プロジェクトメンバーと**意見交換**することで，現場に溶け込むことができた．
	プレゼンの機会をいただき，準備を通じて業務を学ぶことができた．
	若手が前向きに取り組む姿勢が刺激になる．
中立 (18%)	**若手育成**の場としては良いが，参加者へのフィードバックが弱い．中堅以上のメンバーは勉強会に付き合ってあげているという印象がある．
	勉強会というよりは，発表会になっている．参加するだけで勉強した気になっていないか心配．
	テーマについて何も知らない時は，勉強になるが，知っていると物足りない．
不満 (15%)	PM やリーダー層のマネジメント論，品質管理などのテーマの方が，中堅以上のメンバーには響くのではないか．**話を聞きたいメンバー程，多忙という問題をクリアする必要はあるが．**
	勉強会のワークやふりかえりでTRY に対しての評価がない．いいアイデアがでたとしてもその後のフォローアップがないので埋もれている．
	勉強会に参加しないメンバーにこそノウハウがある．有識者の知を継承する場としては不十分．

設問 2：「勉強会が組織に良い影響を与えているか」

約 85%のメンバーが，「5.思う」，「4.どちらかといえば思う」と回答した．

組織に良い影響を与えていると思う理由としては，メンバーの育成や成長，コミュニケーションの活性化，経験や知の継承が上位であった．

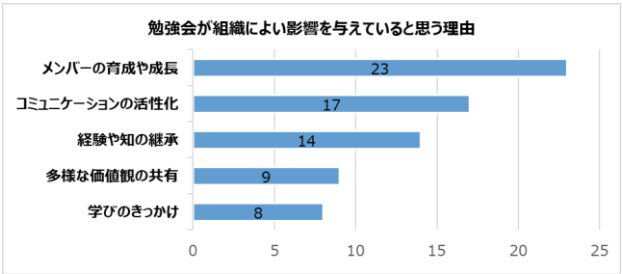


図 8．勉強会が組織に良い影響を与えていると思う理由

設問 3：「勉強会は組織の文化になりつつあるか」

約 55%のメンバーが肯定的な見解を示した．これは，勉強会の参加者が想像より多く，3 年間に渡り一定の規模を維持でき

ていることに対する評価であった．一方，否定的と評価した理由は，「文化とするには時期尚早」．「ボトムアップの活動にならなければ文化とは言えない」などであった．

表 6．勉強会は組織の文化になりつつあるか

評価	コメント内容
思う (55%)	参加者が多いのは，良い方向に向かっている証拠である．
	参加者に気付きを与える場となっている．
	コミュニケーション不足を解消する場として機能している．組織の文化を作るのは人であり，人と人のコミュニケーションが文化をより強固なものにすると思う．
中立 (22%)	現在の運営体制がなくなったら，活動を継続できるかに疑念がある．業務を犠牲にしてまで継続しようとはならないと思う．
思わない (23%)	まだ，そこまでの実感はない．
	現場が自主的に取り扱いたいテーマを選定して企画するようにならないと文化とは言えない．文化はボトムアップで醸成するものと考える．
	参加者へのフィードバックが機能していない．運営メンバーが継続して呼びかけないと形骸化すると思う．

4.1.2 過去の勉強会で印象に残っているテーマ

過去の勉強会で印象に残っているテーマについて、投票したところ，活動 3 年目（2018 年度）が上位を占めた．その理由として想定するのは，以下 2 点である．

①直近開催分の方が，記憶にあたらしく，関係者も多い

②年月を経て，活動内容のレベルが上がってきた

また，上位になった勉強会の，グループワークのテーマを分類すると，マインドセット（思考様式・価値観・信念），プロジェクト（テスト観点）が，上位になっていることがわかった．

表 7．印象に残っている勉強会のテーマ

順位	テーマ名	得票数（票）	グループワークのテーマ
1	2018 年度第 1 回 現状をちょっと良くするためのマインド～プロマネに学ぶ～	10	**マインドセット**
2	2018 年度第 3 回 NISA ロールオーバーって何？	7	**プロジェクト（テスト観点）**
3	2018 年度第 4 回 株式等決済期間短縮対応～きほんのき～	6	**プロジェクト（テスト観点）**
4	2018 年度第 5 回 貸株取引～きほんのき～	5	**マインドセット**
5	2018 年度第 2 回 過去のトラブル事例に学ぶ～セーフティ・セキュリティ～	3	事例共有

4.2 ふりかえりによる評価

毎回，勉強会終了後に実施してきたふりかえりの内容が，3 年間でどう変化したのかを評価，考察する．

4.2.1 ふりかえりの定量評価

ふりかえりの定量的な変化を示す（図 9）．①～③KPT 要素の数，④勉強会参加者のふりかえり記入率，⑤勉強会参加者一人当たり KPT 起票数の観点で計測した．

1 年目は，記入率も，参加者一人当たりの KPT 起票数も高かったが，2 年目以降は低下傾向であった．要因として，4.1 のアンケート結果にもあったとおり，ふりかえり結果のフォローアップ不足，アイデアが次につながらないという問題が，記入率の低下の主要因であると考えることができる．ふりかえりをしてもその先のフォローアップがないと，徐々に意見すらも出なくなるという悪循環に陥っていたことが分かった．

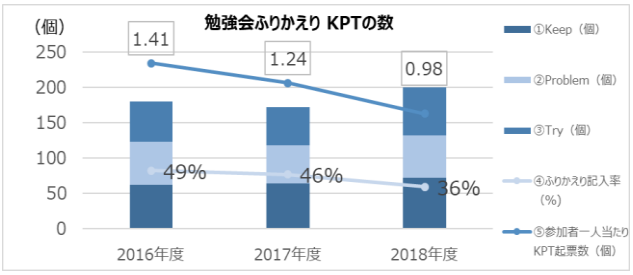


図 9. KPT の定量的な変化

4.2.2 ふりかえりの定性評価

ふりかえりの Try（次に試すこと，試したいこと）について， 1 年目と 3 年目の内容を，単語の出現回数で比較した結果を示す（図 10）．

2016年度のTry	
名詞	出現回数
必要	11
理解	9
イノベーション	8
投信	8
業務	8
プロジェクト	7
チーム	7
案件	6
知識	6
仕事	6
強み	5
工夫	5
提案	5
今後	5
システム	5

未来を意識した言葉が減った

2018年度のTry	
名詞	出現回数
仕事	13
今後	10
発表	10
内容	9
NISA	8
マインド	7
説明	7
対応	7
業務	6
実施	6
テスト	6
理解	6
株	5
勉強会	5
本番	5

図 10．ふりかえり Try の変化

1 年目と 3 年目の Try を比較すると，1 年目は「イノベーション」，「強み」，「工夫」，「提案」といった，未来志向の言葉が多かったが，3 年目は，同種の言葉が減った．これは，勉強会のテーマに大きく左右された結果と推察する．

実際に，勉強会開催実績（表 4）のテーマから判断すると，1 年目は自分たちの目指すゴール，SWOT 分析，イノベーションといった，未来を見据えたテーマであった．一方で，3 年目は，プロジェクトの内容をテーマにした回が中心であった．新しい考え方や発想に触れる機会は少なかったと言える．

日々の業務から離れて，中長期的な課題と向き合い，柔軟

な発想を得るための場として期待したはずが，現実的な課題を取り扱うだけの場になってしまった．この結果は，勉強会のテーマ選定が，参加者に気づきを与える機会を限定してしまう恐れがあるという，重要な課題を示唆していると認識した．

4.3 考察

4.3.1 アンケート結果とふりかえり内容の変化からの考察

4.1 アンケート結果の評価より，参加型勉強会は，若手育成と現場のコミュニケーション活性化に大きく寄与する可能性が高いことを確認した．

若手育成への効果が大きかったのは，勉強会当日のプレゼン実施に向けて，若手が主体的に行動し，周囲を巻き込みながら，勉強会を一つの形にすることが，疑似的なプロジェクトとなっているからであると考える．組織全体を巻き込んで活動することを，若手の内に経験できる意義は大きい．

現場のコミュニケーション活性化への寄与が大きかったのは，勉強会のワークで，メンバーが，どんな考え方で，信念やこだわりを持って仕事に取り組んでいるのかを，お互いに意見交換しながら，共有できたからであると考える．仕事との向き合い方やスタイルは，積極的に発信し，共有するべき情報であるということが分かった．

4.3.2 参加型勉強会の副次的効果

3 年目の途中から，勉強会の対象組織を拡大し，複数組織合同で勉強会を開催した．ここで，勉強会慣れしている組織メンバーの方が，ワークの中で抵抗感なく，積極的に発言していることが分かった．これは，過去の勉強会を通じて，日頃の業務に対する問題意識や改善の意識が高くなっているからではないかと推察した．これは，勉強会の継続が，中長期的には大きな効果を生むのではないかと期待できる点である．

5. 今後の課題

5.1 課題

この活動を 3 年間継続して，明らかになった課題を整理した．

(1) 勉強会後のフォローアップ

勉強会後のフォローアップの不足が，勉強会の満足度低下の一因であった．グループワークやふりかえりで出た，アイデアや要素の行き場を作り，組織の資産として管理し，継続的な改善活動や，新たな挑戦のインプットとすること．

(2) 勉強会に中堅層以上を巻き込む

若手の育成には一定の効果があったとする反面，中堅層やリーダー層の関与が限定的であるとの意見が多かった．現場の中核を担うメンバーを巻き込む場合は，時間の確保が大きな壁になると推察する．現在の勉強会の方式に固執することなく，様々な形を模索していく．

(3)テーマ選定と活動の主体のバランス

文化はボトムアップで醸成するものであるとの意見があった．また，ふりかえり内容の変化から，テーマ選定の重要性を認識した．現場主体，ボトムアップを志向しながらも，目の前の課題だけに偏らずに，中長期的な課題にも向き合っていく点に配慮する必要がある．

5.2 今後の展開

3 年間を経て，この活動は一つの区切りを迎えたが，効果は限定的で，学びを組織の文化にするという段階には，到達していない．また，組織の文化も，絶えず強化し，変化していくものである[7]．メンバー成長や交代を経ながら，組織の形が変わっていく中で，勉強会の在り方も変化していくものと考える．

6. 謝辞

勉強会をこれまで支えてきてくれた，上司，参加者の皆様と企画運営に携わったメンバー全員に，感謝の意を表する．

また，本論文執筆に当たり，業務多忙な中，多くのメンバーにアンケートにご協力頂いた．ここに感謝の意を表する．

参考文献

[1] [事例報告]勉強会を活用した組織成長モデル～機能期のチームが継続的に成長するために～，伊藤修司，山口真，豊田圭一郎(SCSK)，ソフトウェアシンポジウム 2017

[2] [事例報告]勉強会を活用した組織成長モデル ～活動 2 年目の成果と課題～，伊藤修司，山口真，豊田圭一郎(SCSK)，ソフトウェアシンポジウム 2018

[3] Project Management Professional 第 5 版，翔泳社 P530-P534 ISBN:978-4798137292

[4] [事例報告]保守と保守開発における Software Evolution の事例報告，三輪東(SCSK)，ソフトウェアシンポジウム 2017

[5] フューチャーセンターをつくろう 対話をイノベーションにつなげる仕組み，野村恭彦，プレジデント社 ISBN: 978-4833420099

[6] これだけ! KPT，天野勝，すばる舎 ISBN: 978-4799102756

[7] 学習する組織 システム思考で未来を創造する，ピーター・M. センゲ，英治出版 第 14 章 戦略 ISBN: 978-4862761019

要件定義計画を強化するアセスメント項目の提案

越前谷　達朗
株式会社日立ソリューションズ・クリエイト
tatsuro.echizenya.rp@hitachi-solutions.com

久保　光寛
日本システム技術株式会社
m.kubo@jast.co.jp

渋谷　公寛
東京海上日動システムズ株式会社
kimihiro.shibuya@grp.tmnf.jp

新田　史弥
株式会社東光高岳
nitta.fumiya@tktk.co.jp

吉竹　宏幸
TIS 株式会社
yoshitake.hiroyuki@tis.co.jp

石川　冬樹
国立情報学研究所
f-ishikawa@nii.ac.jp

栗田　太郎
ソニー株式会社
taro.kurita@sony.com

要旨

我々は要件定義計画の不備に起因する要件定義の失敗を回避するために，要件定義開始前に具体化するべき計画事項と計画内容に対するアセスメント項目を創出し，過去の開発プロジェクト 53 件に対して調査を行った．結果，要件定義計画の不備を抱えた開発プロジェクトが多く存在すること，それらの要件定義で発生した問題の予防にアセスメント項目が有効であることを確認した．

1. はじめに

実現するビジネスや業務，開発するシステムを決定する要件定義工程は，開発プロジェクト毎に進め方や決めるべき事柄，範囲が多様である．よって要件定義の品質や作業の効率・実現性を担保するためには個々の開発プロジェクトの特性や制約等を考慮した最適な要件定義計画が必要であり，成否の鍵を握る．要件定義実践の観点から要求工学知識を体系化した『要求工学知識体系 REBOK』[1]においては，共通的な知識領域の一部に「要求の計画と管理」を定義し，「プロジェクトの特性や遂行条件を考慮して，要求開発の方法や手順を検討する必要がある」と解説している．しかし我々は自らの経験から，要件定義計画の未作成，計画の具体性不足，等は普遍的な問題であると推察した．

我々は，開発プロジェクトの要件定義計画の改善を促すことを目的とし，現場で実践可能なガイドラインとして利用できる『要件定義計画アセスメント項目』を提案する．今回作成した要件定義計画アセスメント項目を過去の開発プロジェクト 53 件に適用した調査の結果，要件定義計画の課題・リスクを事前に抽出し，要件定義で発生している主要な問題を予防する可能性があることを確認した．

以下本論文の構成を述べる．2 章で要件定義計画の概要や実態，課題を説明する．3 章で課題への対策となる要件定義計画アセスメントの考え方や内容，利用手順を説明する．4 章でアセスメントによる要件定義計画の不備・リスク抽出調査の内容と結果を説明し，5 章で調査結果を考察する．6 章で要件定義計画の課題に対するアセスメント項目の効果を，7 章でアセスメント項目の活用方法と今後の課題を述べる．

2. 要件定義計画の概要と現状

2.1. 要件定義計画の概要

要件定義計画は下記事項について，開発プロジェクト毎の特性や制約に合った最適かつ実行可能な内容を検

討し，ステークホルダと合意，共有するものである．検討事項の多くは他工程の実行計画に同様の項目があり理解しやすいが，「(3) 3)」「(4) 1)」のように，要件定義に依存する特徴的な検討事項が一部に含まれる．

(1) 要求開発プロセスの基本方針と前提条件
  1) 要求開発プロセスの目的
  2) プロジェクトに関係するステークホルダとその役割
(2) 要求開発プロセスのアプローチ
  1) プロジェクトライフサイクルの選択とアクティビティの設定
  2) 要求獲得，要求分析，要求仕様化，要求の検証・妥当性確認・評価の各プロセスでのアプローチ
(3) 要求開発プロセスのアクティビティ
  1) 各アクティビティの実施手順，使用する書式
  2) 要求定義文書の標準的な要求記述ガイド
  3) 要求の優先順位付けプロセス
  4) 要求のレビュー方針と確認方針，受け入れ基準と評価基準
(4) 要求開発プロセスのコミュニケーション計画
  1) ステークホルダ間のコミュニケーションの内容と方法
  2) コミュニケーションスケジュール
(5) プロジェクト管理と要求定義，システム構築との関係
  1) プロジェクト管理：メトリクス，リスク管理，問題管理
  2) システム構築：見積りの範囲と精度
(6) 要求管理計画
  1) 要求管理プロセス
  2) 構成管理

※要求工学知識体系 REBOK 第 7 章 要求の計画と管理 から引用

2.2. 業界内における取り組み

業界内の要件定義計画強化に関連する取り組みを主要な標準，知識体系で確認したが，いずれも具体的な定義事項やガイドライン，ノウハウは示さず，実効性は限定的であった．共通フレーム 2013[2]では「2.2 要件定義プロセス」に「要件定義プロセス実行計画の作成」が定義されているが，要件定義実施に関する標準，方法論，手順，役割分担を決定することの必要性を説明するに留まっている．要求工学知識体系 REBOK では，共通知識領域「要求の計画と管理」に「要求開発プロセスの計画」が定義されているが，2.1 に挙げた要件定義計画の検討事項を定義するに留まっている．システム再構築を成功に導くユーザガイド[3]では，要件定義計画事項が定義されているが，マイグレーションに限定されている．

2.3. 開発プロジェクトでの要件定義計画立案の実態

要件定義計画立案の実態を把握するため，回答者が担当した直近の開発プロジェクトでの要件定義計画の内容を表1のアンケートで調査した．また，回答内容との依存関係を分析するため，回答者とプロジェクトのプロフィールも収集した．回答者へのアセスメント項目は 33 項目を対象に合否評価を実施した．(3 章 表 10 参照)

表 1. アンケート項目と概要

No	アンケート項目	アンケート概要
1	プロジェクト プロフィール	回答対象プロジェクトの開発種別，開発対象分野，開発形態等
2	回答者 プロフィール	回答者の要件定義の経験プロジェクト数，累計経験期間，役割・担当領域
3	要件定義計画の問題の有無	要件定義計画書の有無，計画内容に関する問題の有無
4	アセスメント項目	要件定義計画の内容に対するアセスメント実施結果(33 項目での合否評価)

53 件のアンケート回答を集計し，以下 2 点について調査した結果を示す．

(調査 1) 要件定義計画の実態
(調査 2) 要件定義計画の実態と回答者プロフィールとの相関関係

調査 1 要件定義計画の実態

アンケート結果の分析により，以下 2 点の見解が得られた．

(1) 要件定義計画が未作成または問題があるケースが多い

要件定義計画の有無と計画の品質について，評価を 3 択で回答させた．回答結果を図 1 に示す．さらに，計画がある開発プロジェクトの要件定義計画に 33 項目のアセスメントを適用し，各項目の内容が計画書に反映されている項目を合格とし、合否を判定した．合否判定結果から，各プロジェクトのアセスメント合格率を算出し，分布を作成した．これを図 2 に示す．図 1 および図 2 より以下の結果が得られ，多くの開発プロジェクトで要件定義計画に軽視できないレベルの問題があることが確認できた．

① 回答者 53 名の約 94%が要件定義の「計画無し」または「計画に問題あり」と評価(図 1 参照)
② 回答者 53 名の内、要件定義計画があると回答された 42 件のプロジェクトの 6 割が，アセスメント合否判定の合格率 40%から 70%に集中(図 2 参照)

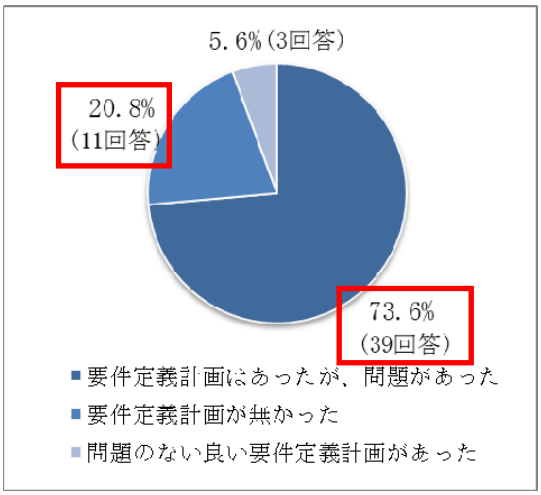


図 1 要件定義計画の有無と計画の品質

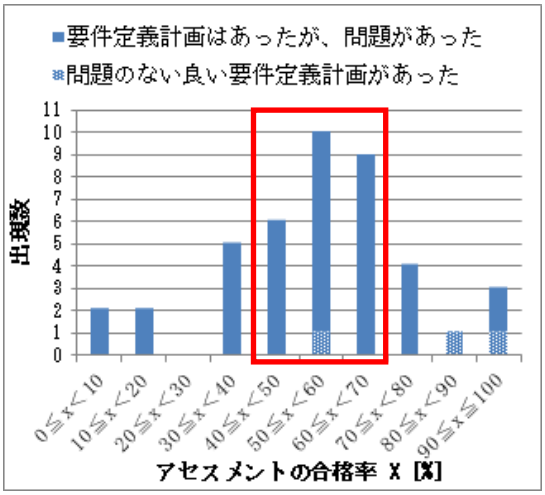


図 2 各プロジェクトのアセスメント合格率の分布

(2) 要件定義特有の観点を考慮した要件定義計画が立案できていない

計画がある開発プロジェクト 42 件を対象にアセスメント項目ごとの合格率を求めた．合格率が上位・下位のアセスメント項目の一部を表 2，表 3 に示す．合格率上位には「進捗や品質・リスクの共有」「プロジェクト目的」「スケジュールやタスク・担当の明確化」といったプロジェクト計画でも扱う認知度の高い計画事項が入った．合格率下位には「現行踏襲要件の認識齟齬の防止」や「要件の曖昧さの排除」や「成果物の記述レベル・方法の明確化」といった，要件定義特有かつ重要な計画事項が入った．一方で「開始条件」「完了条件」といった認知度の高い計画事項の合格率が低いケースが確認できた．これは要件定義で決めるべきこと，やるべきこと，開始時の状態等が開発プロジェクト毎に異なり多様であるため，汎用的な条件を設定し難いためと考える．

表 2. 合格率が高いアセスメント項目 (抜粋)

合格率 [%]	アセスメント内容
88.1	関係者間において，進捗や品質・リスクを定例会などで共有するようになっており、課題の対応方針の確認など，チェックポイントが設定されているか.
85.7	プロジェクトの目的が明確になっているか. あるいは，目的を明確にするためのタスクが計画されているか.
83.3	関係者間において，スケジュール，体制，タスク，分担が明確になっているか.

表 3. 合格率が低いアセスメント項目 (抜粋)

合格率 [%]	アセスメント内容
26.2	定義した要件に対するステークホルダ間の理解齟齬を防止するために，要件文書の曖昧さを排除する検証計画を立てているか.
31.0	お客さま担当成果物も含め，各成果物内容，記述事項，記述レベル・方法が，その必要性を含めて明確になっているか.
40.5	要件定義工程の開始条件と，それに対する現状を整理し，要件定義作業の開始可否を評価可能な準備を行っているか.
40.5	お客さまとの間で「現行踏襲」要件の範囲や内容の認識齟齬を最大限防止するための取り組みを計画しているか.
42.9	要件定義工程の完了条件および判断ルールを成果や品質の観点から明確にし，途上管理計画や審議計画を行っているか.

調査 2 要件定義計画の実態と回答者プロフィールとの相関関係

調査 1 の結果と回答者・対象プロジェクトのプロフィールとの相関関係の分析から，有意な特徴として以下 2 点が確認できた．

(1) 表 4 より新規開発，派生開発に比べて，マイグレーションプロジェクトのアセスメント合格率が低いことが分かる．マイグレーションの場合，基本的に業務要件が現行と同じで，顧客に業務要件を確認するための計画事項が少なく済むためと考える．

(2) 表 5, 6, 7 より以下のいずれかの条件を満たす人が携わるプロジェクトの要件定義計画書はアセスメント合格率が高いことが分かる．

・要件定義で広い役割を経験
(システム要件，アーキテクチャ，インフラ等)
・要件定義の経験プロジェクト数が 7 件以上
・要件定義の累積経験期間が 15 ヶ月以上

このことから要件定義計画の良し悪しは計画担当者の要件定義実践経験と暗黙知に依存していると考えられる．

表 4. 開発形態によるアセスメント合格率

プロフィール情報のアンケート回答		アセスメント合格率
開発形態	回答数	
新規開発	12	57.6%
派生開発	16	58.4%
マイグレーション	14	45.7%

表 5. 要件定義工程における役割によるアセスメント合格率

プロフィール情報のアンケート回答(複数回答)		アセスメント合格率
要件定義工程における役割	回答数	
業務要件定義	16	50.2%
システム要件定義(機能要件)	33	54.8%
システム要件定義(非機能要件)	22	56.5%
ソフトウエアアーキテクチャ	11	62.6%
インフラ・NW	4	61.1%
移行	9	64.0%
プロジェクトマネジメント	20	52.4%
プロジェクトマネジメントオフィス	2	63.3%
その他	1	9.9%

表 6. 要件定義を担当したプロジェクト数によるアセスメント合格率

プロフィール情報のアンケート回答		アセスメント合格率
要件定義を担当したプロジェクト数	回答数	
1～2 件	10	55.3%
3～6 件	16	48.1%
7 件以上	16	58.9%

表 7. 要件定義を担当した累積期間によるアセスメント合格率

プロフィール情報のアンケート回答		アセスメント合格率
要件定義を担当した累積期間	回答数	
1～2 ヶ月	0	0.0%
3～6 ヶ月	11	53.6%
7～14 ヶ月	7	42.3%
15 ヶ月以上	24	57.5%

2.4. 要件定義計画の課題

2.2, 2.3 の調査結果より，要件定義計画に関して表 8 の課題があると考える．

表 8. 要件定義計画に関する課題一覧

No	課題内容
課題 1	標準・知識体系に定義されている要件定義計画の検討事項は粒度が粗く，ガイド・ノウハウとしての実効性に限りがある(2.2, 2.3 調査 2 参照)
課題 2	開発プロジェクトの要件定義計画には問題があり，要件定義特有の観点を考慮した要件定義計画が立案できていない傾向にある(2.3 調査 1 参照)

3. 要件定義計画を確認するアセスメント項目の提案

3.1. アセスメント項目の目的

前章で挙げた課題を解決するため，要件定義開始前に要件定義計画の内容を評価し，その改善を促すことを目的とした要件定義計画アセスメント項目を提案する．

アセスメント項目に準じて各開発プロジェクトの要件定義計画の内容を確認することで，要件定義計画立案者の経験やノウハウの不足を補い，計画の問題やリスクに対する気付きを得ることを期待できる．

3.2. アセスメント項目の概要

アセスメント項目を表 10 に示す．アセスメント項目立案にあたり，アセスメント対象となる計画書の目次構成は表 10 の章・節列の内容のとおり具体化した．その計画書目次を網羅するアセスメント項目を定義した．アセスメント項目は，計画が書かれていることよりも，計画内容が妥当であることの確認に重点を置いた．具体的なアセスメント項目の内容は，著者らの経験や知見を整理し，議論の上決定した．

3.3. 利用手順

開発プロジェクトでの利用手順を表 9 に示す．また，その利用イメージを図 3 に示す．ただし，本研究の調査では過去の開発プロジェクトが対象であるため，この手順に沿った利用はしていない．

重要な点は，顧客を含む全ステークホルダグループが表 9 の No.3 と No.4 の手順に参加することである．これにより，要件定義開始後に起こりがちな認識齟齬の発覚による作業停滞や計画外作業を予防できる．

表 9. アセスメント項目の利用手順

No.	手順	説明
1	項目選別	アセスメント項目を選別する． アセスメントは全項目実施が必須ではない．開発プロジェクトの特性や条件，リスク等を考慮し，実施不要/不可の項目を対象から外す．特定の開発プロジェクト固有の確認項目の必要性も確認する
2	紐づけ	アセスメント項目とアセスメント対象を紐づける． 開発プロジェクト毎に計画を記載する文書やその形式が異なるため，アセスメント対象の計画事項が記載される文書，目次を特定し，アセスメント項目と紐付ける
3	アセスメント実施	複数担当者でアセスメントする． 顧客を含む全てのステークホルダグループから選出した担当者が個別にアセスメントを実施する
4	結果比較	アセスメント結果を評価する． 担当者全員の評価を確認し，評価が低い項目や担当者間の評価差異が大きい項目について，全担当者で理由や認識をすり合わせる
5	対応検討	要件定義計画の改善内容を確定する． 要件定義計画の再検討箇所，検討方針等をステークホルダ間で確定，合意する

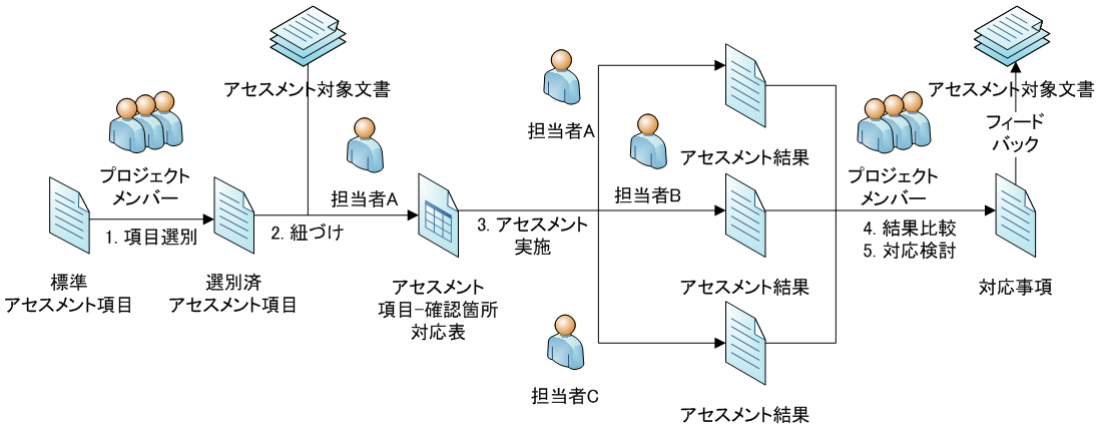


図 3 アセスメント項目の利用イメージ

表 10. アセスメント項目

章	節	アセスメント項目
１．要件定義方針		
	（１）プロジェクトゴール	
		①プロジェクトの目的が明確になっているか。あるいは、目的を明確にするためのタスクが計画されているか。
	（２）要件定義スコープ	
		①要件定義作業や検討領域、業務、ITシステム機能の範囲は、読み手による解釈の違いがないように具体的に定義されているか。
	（３）要件定義遂行上の制約、前提	
		①お客さまが示す要件定義実施に関する制約条件が明確か。
		②要件定義計画の完遂に必要な前提条件を自社から明示しているか。
		③制約内で実行可能な要件定義計画を立てているか。
２．要件定義実施計画		
	（１）実施計画概要	
		①要件定義計画書がお客さまのプロジェクトオーナーをはじめとする関係者、ステークホルダーに説明され、合意・承認を得ているか。
	（２）要件定義の進め方	
		①採用予定の技術や製品、パッケージ等と、業務要件・システム要件との適合評価および対策検討の実施を計画しているか。
		②お客さまとの間で「現行踏襲」要件の範囲や内容の認識齟齬を最大限防止するための取り組みを計画しているか。
		③要件定義の進め方の論理的な流れや方法論を具体化し、プロジェクトが実行可能な要件定義プロセス/成果物を計画しているか。
	（３）ご提示頂く情報	
		①システム企画・システム化計画・提案依頼書・業務要件定義等のインプットについて、内容の範囲や網羅性、具体性等を確認し、想定に対する不備・不足があれば対策を計画に織り込んでいるか。
		②現行業務、現行システムを可視化した文書の整備状況、品質に問題はないか？問題が明確な場合、現行調査実施の判断や実施計画立案を行っているか。
	（４）要件調整の進め方	
		①要件規模が予定リソースを超過した際に、実現要件を調整するルール、手順、基準を定義し、調整実施タイミングを計画しているか。
	（５）品質計画	
		①プロジェクトの目的・目標に対する要件の妥当性をお客様が評価・判断する実施計画を立てているか。
		②定義した要件を業務またはシステムとして実現できる見通し、裏付けを取る検証計画を立てているか。
		③プロジェクトの目的や目標、解決対象の課題等と紐づけて、要件の必要性を確認する検証計画を立てているか。
		④定義した要件に対するステークホルダー間の理解齟齬を防止するために、要件文書の曖昧さを排除する検証計画を立てているか。
	（６）体制	
		①関係者間において、スケジュール、体制、タスク、分担が明確になっているか。
		②お客様業務やシステムに関する知識・経験を保有した、自社体制を構築しているか。
		③自社要員は、要件定義計画にまとめた要件定義の進め方や成果物体系、背景にある考え方を理解し、実践する準備ができているか。
	（７）スケジュール	
		①難易度、複雑度、現行の可視化度合い等による、業務やシステム機能ごとの重みやリスクの違いを踏まえた、実現性あるスケジュールを計画しているか。
	（８）成果物定義	
		①お客さま担当成果物も含め、各成果物内容、記述事項、記述レベル・方法が、その必要性を含めて明確になっているか。
		②後続の、方式要件・基盤要件・外部設計や開発工数見積に対する十分なインプット情報を備えた成果物体系、様式、記述ルールになっているか。
		③前/後工程を含め、成果物間の前方/後方トレーサビリティが明確であり、要件の存在理由、必要性を確認可能な、成果物体系、様式、記述ルールになっているか。
	（９）コミュニケーション計画	
		①お客様との意思決定や、要件合意および承認に関するルールや実施タイミングが明確になっているか。
		②ステークホルダー間で要求が纏まらない場合などに、プロジェクトとして判断を下す最終意思決定者が決められ、意思決定ルールとして明確になっているか。
		③関係者間において、進捗や品質・リスクを定例会などで共有するようになっており、課題の対応方針の確認など、チェックポイントが設定されているか。
		④プロジェクトに必要なお客さまとのコミュニケーション密度を踏まえて、お客さまオフィスでの常駐作業の要否を検討しているか。
	（１０）工程開始/終了基準	
		①要件定義工程の開始条件と、それに対する現状を整理し、要件定義作業の開始可否を評価可能な準備を行っているか。
		②要件定義工程の完了条件および判断ルールを成果や品質の観点から明確にし、途上管理計画や審議計画を行っているか。
	（１１）要件定義の重要成功要因と対策	
		①要件定義の成功のために決定的に重要となるプロジェクト特有の要因を分析し、要件定義計画で「要件定義の重要成功要因と対策」として具体的活動に落とし込んでいるか。
３．お客さまへの依頼事項		
		①要件定義活動におけるお客様の責任、役割に基づく、具体的なタスクや対応依頼事項（依頼先、ボリューム、時期、期待アウトプット等）が明確になっているか。
４．課題、リスク		
		①要件定義計画の各項目に関するリスク・課題は網羅的に整理が行われ、対策および自社/お客さまでの担当範囲は明確か。
		②要件定義開始に間に合わなかった計画事項がある場合、課題として管理し、影響を管理できているか。

4. 調査

アセスメント項目の有効性と網羅性を検証するためアンケートを実施した.

4.1. 調査方法

著者らの所属会社で要件定義経験者に下記 3 点を確認するアンケートを実施した. なお、本アンケートは, 2.3 要件定義計画立案の実態調査のアンケートと合わせて実施した.

- ・過去の開発プロジェクトの要件定義で発生した重要な問題(Q1)
- ・上記問題に最も高い予防効果を期待できるアセスメント項目(Q2)
- ・我々のアセスメント項目に対する追加項目案(Q3)

4.2. 調査結果

要件定義で発生した重要な問題を内容で分類した結果を表 11 に示す. 回答者 53 名から得た重要な問題は 5 つに分類できた. なお、分類にあたっては, アンケート回答にあった問題事象を1件ずつ確認し、表 11 内訳の粒度で分類した後, それらを要約して 5 点の重要問題として整理した.

また, 重要な問題ごとに, 予防効果が高いと回答者が評価したアセスメント項目と回答数を整理した結果を表 12 に示す. 表 11 で分類した主要な問題 5 つを横軸に, アセスメント項目の分類(3 章 表 10 の章・節列)を縦軸に並べ, 問題に対してアセスメント項目が有効と回答された件数を各マスに記載した. 重要な問題 51 件のうち 49 件に対して, 予防効果があるアセスメント項目がアセスメントシート内に存在することが確認できた.

表 11. 要件定義で発生した重要な問題内容と比率

#	重要な問題		件数	比率
1	顧客要件の変更/拡大		12	23%
	内訳	顧客要件の変更	5	
		コスト以上の要求	4	
		追加要件の発生	3	
2	顧客意思決定の遅れ		10	19%
	内訳	顧客間における要件の共有不足	4	
		顧客側の要件確認遅延	3	
		顧客要件の決定遅延	3	
3	要件の抜け漏れ/認識齟齬		16	30%
	内訳	要件定義が不十分	12	
		要件の認識齟齬	4	
4	現行システム理解不足		6	11%
	内訳	現行業務の可視化に対する顧客の協力が無い	3	
		現行システムの調査が不足	3	
5	マネジメント不備		7	13%
	内訳	資料作成工数増加による工程遅延	2	
		体制を確保できずに要件定義を開始して有識者の高負荷発生	1	
		実現方法に議論が集中し工程遅延	1	
		過去プロジェクトをなぞる計画で進めたため品質に問題発生	1	
		成果物定義を決定せず要件調整を進め、顧客指摘が多発し工程遅延	1	
		リソース調整に時間がかかり工程遅延	1	
6	重要な問題なし		2	4%
7	合計		53	100%

表 12. 要件定義で発生した問題に対する有効なアセスメント項目(目次別)

#	目次 ＼ 重要な問題		顧客要件の変更/拡大		顧客意思決定の遅れ		要件の抜け漏れ/認識齟齬		現行システム理解不足		マネジメント不備		重要な問題なし		合計		
			件数	比率	件数	比率	件数	比率	件数	比率	件数	比率	件数	比率	件数	比率	比率(#18 除く)
1	1．要件定義方針	（1）プロジェクトゴール	0	0%	0	0%	0	0%	0	0%	0	0%	0	0%	0	0%	0%
2		（2）要件定義スコープ	0	0%	0	0%	4	25%	0	0%	1	14%	0	0%	5	9%	10%
3		（3）要件定義遂行上の制約、前提	1	8%	2	20%	0	0%	0	0%	1	14%	0	0%	4	8%	8%
4	2．要件定義実施計画	（1）実施計画概要	3	25%	3	30%	1	6%	0	0%	0	0%	0	0%	7	13%	14%
5		（2）要件定義の進め方	0	0%	0	0%	1	6%	2	33%	1	14%	0	0%	4	8%	8%
6		（3）ご提示頂く情報	1	8%	0	0%	0	0%	2	33%	0	0%	0	0%	3	6%	6%
7		（4）要件調整の進め方	1	8%	1	10%	0	0%	0	0%	1	14%	0	0%	3	6%	6%
8		（5）品質計画	1	8%	0	0%	2	13%	0	0%	0	0%	0	0%	3	6%	6%
9		（6）体制	1	8%	0	0%	0	0%	1	17%	1	14%	0	0%	3	6%	6%
10		（7）スケジュール	0	0%	0	0%	0	0%	0	0%	0	0%	0	0%	0	0%	0%
11		（8）成果物定義	0	0%	0	0%	4	25%	0	0%	2	29%	0	0%	6	11%	12%
12		（9）コミュニケーション計画	2	17%	0	0%	0	0%	0	0%	0	0%	0	0%	2	4%	4%
13		（１０）工程開始/終了基準	0	0%	0	0%	2	13%	0	0%	0	0%	0	0%	2	4%	4%
14		（１１）要件定義の重要成功要因と対策	0	0%	1	10%	0	0%	0	0%	0	0%	0	0%	1	2%	2%
15	3．お客さまへの依頼事項		0	0%	2	20%	1	6%	1	17%	0	0%	0	0%	4	8%	8%
16	4．課題、リスク		0	0%	1	10%	1	6%	0	0%	0	0%	0	0%	2	4%	4%
17	該当するアセスメント項目は無い		2	17%	0	0%	0	0%	0	0%	0	0%	0	0%	2	4%	4%
18	QCD に影響する問題は無かった		0	0%	0	0%	0	0%	0	0%	0	0%	2	100 %	2	4%	-
19	合計		12	100 %	10	100 %	16	100 %	6	100 %	7	100 %	2	100 %	53	100 %	100%

追加アセスメント項目案 36 件を我々のアセスメント項目とマッチングした結果を図 4 に示す. 30 件は定義済のアセスメント項目と同義であり, 6 件は定義済項目にない新たな観点であった.

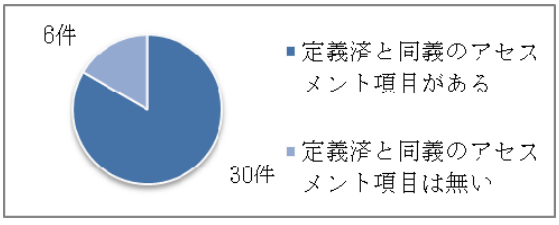


図 4 追加アセスメント項目案の分類

5. 考察

5.1. アセスメント項目の有効性評価

表 11,12 から要件定義の重要問題全件の 72%を占める上位 3 問題に対する予防効果が確認でき，アセスメント項目の有効性が認められる．問題毎に考察を纏める．

(1) 顧客要件の変更/拡大

表 12 の#4(実施計画概要)との関係が強い．#4 では要件定義計画書を顧客に説明し合意・承認を得ることをアセスメントする．要件の変更/拡大防止には顧客からの無理難題な要求を抑制する必要があるため，次点の#12(コミュニケーション計画)において要件の合意・承認ルールを決定し，ルールを顧客と合意すること(#4)の有効性が認められる．

(2) 顧客意思決定の遅れ

表 12 の#4(実施計画概要)との関係が強い．限られた要件定義期間で要件確定するには，全ステークホルダがそれぞれの責務を計画通りに果たす必要がある．次点の#15(お客様への依頼事項)において顧客側キーマンに然るべき時期の意思決定が重要責務であることを理解させ，計画に責務を定義し合意すること(#4)の有効性が認められる．

(3) 要件の抜け漏れ/認識齟齬

表 12 の#2(要件定義スコープ)，#11(成果物定義)との関係が強い．#2 のアセスメント項目には要件定義作業や検討領域，業務，IT システム機能の範囲は読み手による解釈の違いがないように具体的に定義されているか，#11 のアセスメント項目にはお客さま担当成果物も含め各成果物内容，記述事項，記述レベル，方法がその必要性を含めて明確になっているかについてアセスメントすることとしており，要件の抜け漏れ/認識齟齬の抑制に対する有効性が認められる．また検証妥当性確認プロセスや工程開始終了基準を確認する#8(品質計画)，#13(工程開始/終了基準)が次点に入っており，本問題への根本的対策となる「ステークホルダ間の共通認識/理解の形成」に対する有効性が認められる．

5.2. アセスメント項目の網羅性評価

アセスメント項目で確認すべき対象の網羅性を考察する．3 章で述べたとおり，要件定義計画書の目次構成を網羅するアセスメント項目を整備していること，表 12 で大多数のアセスメント項目に主要問題のいずれかに対する予防効果が確認できること，回答者から追加アセスメント項目の要望が少ないこと，から一定の網羅性が認められる．

(1) 大多数のアセスメント項目に有効性が認められる

表 12 で，#1(プロジェクトゴール)，#10(スケジュール)を除く88%のアセスメント項目に主要5問題のいずれかへの予防効果があると評価されたことから、主要5問題を網羅していると認められる．

(2) アセスメント項目追加の提案数は少なく，内容も限定的

アセスメント項目追加提案6件を分類すると「プロジェクト外要因に関する確認事項」「コミュニケーションの改善に関する確認事項」に関するものであった．前者は不足していた視点で，改善点と考える．後者は定義済の同類項目の強化に役立てるべきものであった．

6. 結論

開発プロジェクトへの事後適用調査により，表 13 に挙げた効果を確認した．

今回提案する『要件定義計画アセスメント項目』では，計画書内に計画内容が記載されていることでは不十分であるため，計画書の目次項目(表 10 の章・節列参照)に合わせた内容に踏み込んだアセスメント項目とした．その結果，主要問題の予防効果が認められ，『要件定義計画アセスメント項目』が実践で利用可能なガイドラインであったことが確認できた．要件定義計画の不備を補完するにあたり必要な項目が定義されていたことが確認できたことから，提案するアセスメント項目を要件定義計画の内容のアセスメントに適用することで，開発プロジェクトごとに最適な計画へと強化するための改善点を得られることが確認できた．

表 13. 要件定義計画に関する課題に対する効果

No	課題内容	効果
1	標準・知識体系に定義されている要件定義計画の検討事項は粒度が粗く，ガイド・ノウハウとしての実効性に限りがある	アセスメント項目は要件定義の主要な問題に対して予防効果がある(5.1 参照)
2	開発プロジェクトの要件定義計画には問題があり，要件定義特有の観点を考慮した要件定義計画が立案できていない傾向にある	アセスメント項目は要件定義計画が備えるべき事項を網羅する項目を備えており，要件定義特有の観点を考慮した計画立案ができる(5.2 参照)

7. 今後の活用

7.1. 要件定義計画以外への適用の提案

以下ユースケースにおいても，アセスメント実施による効果を期待できる.

(1) 提案活動段階での，提案する要件定義業務の妥当性，実現性，リスクの分析
(2) 設計工程開始段階での，他社が実施した要件定義業務の妥当性の確認

顧客から提案依頼書を受領し提案を行う(1)の場合，コストやスケジュール等に大きく影響する重要な計画事項を提案書に記載し，顧客の合意を得る. 要件定義で決めることや進め方，顧客を含めた役割分担，実施期間，顧客側の対応事項等である. 提案に含めた事項は受注後の変更が難しいため，提案前にアセスメントする価値がある.

設計工程から開発を請負う(2)の場合，何かしらの方法で設計工程開始の可否や妥当性を確認する. その一部として要件定義成果物の品質確認に加え，要件定義計画の事後アセスメントでプロセス品質を確認する. 要件定義に参加したステークホルダや進め方，使用した文書，承認プロセス等の確認により，設計工程開始判断の妥当性向上を期待できる.

7.2. 今後の課題

アセスメント項目自体も要件定義の多様性に対応する必要があり，特に下記 2 点に対する取り組みが必要と考えている. また，進行中プロジェクトへの適用も含め，研究，実証実験を継続し，実用効果を最大化したい.

・プロジェクト特性やリスク等に合わせたアセスメント項目の選別方法の整備
・利用者のコンテキストに依存したアセスメント項目内容の解釈差の抑制

謝辞

本論文の執筆に当たり，日本科学技術連盟・第 34 年度(2018 年度)ソフトウェア品質管理研究会・研究コース 5 アドバイザーの熊本高等専門学校の荒木啓二郎校長，研究コース 5 の研究員の皆様，日本科学技術連盟・事務局の皆さまにお世話になりました. 厚く御礼申し上げます.

参考文献

[1] 一般社団法人 情報サービス産業協会 REBOK 企画 WG，要求工学知識体系 REBOK，第 1 版，近代科学社,2011
[2] IPA SEC，共通フレーム 2013～経営者、業務部門とともに取組む「使える」システムの実現～，第 1 版，IPA，2013
[3] IPA SEC，システム再構築を成功に導くユーザガイド～ユーザとベンダで共有する再構築のリスクと対策～，第 2 版，IPA，2018

新規製品開発時の発想支援ツールの提案

辻脇　優一
yuichi.tsujiwaki@gmail.com

中谷　多哉子
放送大学
tinakatani@gmail.com

要旨

新規製品開発時には，既存製品に新しい価値をつけることが必要である．ここで製品の価値とは，製品のステークホルダのゴールを満たすものである．そのような新しい価値を新製品に組み込む際に，現状では開発者の閃きや思いつきに大きく依存している．そのようなアイディアを開発者がコンスタントに生み出せるようにするためには，工学的な手法が必要である．その際の発想支援として，著者のチームでは経験的な知見に基づいて作られた独自の発想支援ツールを考案し，用いてきた．この手法では，重要性として二項目，さらに三種類のステークホルダを抽出し，誰にとってどのような機能が目的を達成するために必須なのか，あるいは，付加価値を与えるものなのかを定義する．したがって，手法の入力は，目的，機能，その機能の重要性，その機能のステークホルダである．この手法の課題は第一に，目的とそれを達成するための機能が一段階しか定義できない．第二に，より多様なステークホルダを定義することが困難である．第一の課題を解決するために，我々はこの手法と KAOS 法とを統合した．また第二の課題を解決するために，人が担うロールによってステークホルダを分類した．これによって各ロールのユーザエクスペリエンスを分析し，課題を抽出するとともに，課題を階層的に分析することが可能となる．最終的に本稿で提案する手法では，ロール毎の課題と課題を解決するための手段を明らかにし，その手段から新製品を開発するための機能を要求として定義することが可能となる．また，この要求は別のステークホルダ，マーケティング担当者によって新製品の新しい機能による価値が評価される．本稿では，本手法を検証するために，目的と手段の関係を明確にできるか，顧客のニーズに答えることができる機能を創出することができるかを，実際の新製品開発において，新たに追加された機能を用いて検証した．

1. 背景と目的

新規製品を開発する時には，既存製品に対し，新しい価値を付加する必要がある．製品における価値とは，顧客のゴールを達成することであり，製品とはそのゴールを満たす手段である．

筆者の所属するチームでは，現状，そのような新しい価値のアイディアは，開発者の閃きや発想に頼っており，コンスタントに生み出すことができていない．本研究では，要求工学的なアプローチで，新製品の価値を創出するための手法を，独自ツールを改良することにより開発する．ここで独自ツールとは，筆者のチームが現在使用しているアイディア発想のための手法である．本研究の目的は，独自ツールの課題を明らかにし，それを解決した新しい手法を開発して提案することである．

2. 関連研究

2.1. KAOS 法

KAOS 法は，典型的なゴール指向要求分析法である[1][2]．ゴール指向要求分析法は，顧客のゴールを満たすシステムの構築のために使用される．ただし，ゴール指向要求分析法は顧客のゴールが明確な場合に有効な手法であり，本研究で解決しようとしている課題のように，製品の新しい価値，すなわち顧客の未知のゴールを発見するという場面では，有効な手段にはなりにくい．

2.2. 発想支援に関する研究

要求工学の手法の分類を図 1 に示す[3]．この図は，要求分析のプロセスが静的であるか動的であるか（横軸），分析する空間が閉じているか，開いているか（縦軸）によって各手法をマッピングしたものである．

ここでは右下の象限の 2 つの手法，リッチピクチャと CATWOE 分析を紹介する．

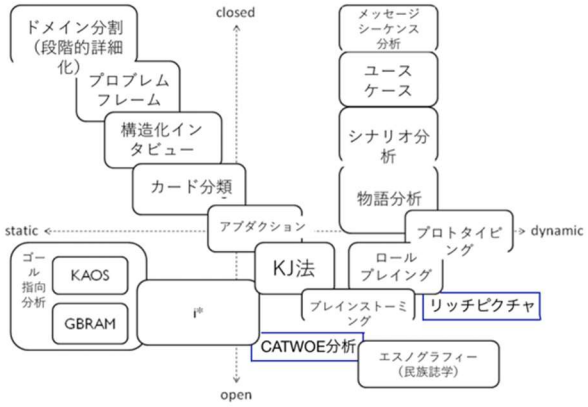


図 1 要求工学の手法の分類
（[1]の図に青枠部分の項目を加筆）

2.2.1. リッチピクチャ

リッチピクチャとは，イラストと吹き出しを用いて，現状を視覚的にわかりやすく表現する手法である[4]. リッチピクチャの目的は，ステークホルダの抽出と，各ステークホルダの課題の発見である. 図 2 にその例を示す.

図 2 リッチピクチャの例

図 2 の絵をリッチピクチャと呼ぶ. このような絵を介することによって，直感的に現状を理解することができ，また種々の関係を表現することが可能となる. 吹き出しで各ステークホルダの意見を表すことにより，誰がどのような事柄を課題と認識しているのかを抽出できる.

2.2.2. CATWOE 分析

CATWOE とは，Customer（T の犠牲者もしくは受益者），Actors（T を行う人々），Transformation process（T_pre: A 状態を T_post: B 状態に変換する. 世界観 W と整合する状態への変換），Weltanshauung または World View（T を意味あるものにしたいと考えている人の世界観），Owners（T を止めることができる人々. 推進すべきと考えている人. T を実現する責任者，支持者），Environment（環境，資源，制約）の頭文字からなっており，各項目を明示する手法である[5]. 特に Owner の世界観を W として明示することによって，Owner が現状を評価するときの基準を明らかにすることができる. ここから，新たな要求を抽出することができる.

以上 2 つの手法は，現状を理解するための手段であるため，新しい価値を発想するためのガイドが必要である.

2.3. 独自ツール

筆者の所属するチームで，新規製品に新しい価値を付加することを目的として，経験的な知見に基づいて作られた独自のツールを使用している. その略式図を図 3 に示す.

図 3 独自ツールの略式図

図 3 に示したように，この独自ツールでは，最初に目的を定義し，目的に対して複数の機能要求を定義する. 次に，各機能の優先順位を「必須」，と「あったら良い」によって示す. さらに，各機能を要求するステークホルダを「使用者」，「開発者」，「その他」によって分類する.

新製品を開発するときに，「使用者」にとって「必須」の機能を実現することに新しい価値はないが，「開発者」や

「その他」のステークホルダーにとって，必須ではないが「あったら良い」ことを多く取り入れることができれば，それが新たな価値を生み出す可能性があると考えられる．

この独自シートの課題は以下の 3 点である．

- 目的と手段の関係が階層構造をもっていない．そのため，複雑な関係を分析することができない．
- ステークホルダ「その他」をさらに分類する必要があるが，「その他」を識別するための明確なルールがない．
- 新しい発想を創造するための系統的プロセスが定義されていない．

3. 課題解決へのアプローチ

3.1. 概要

既存の製品・機能によってすでに満たされているゴールを維持ゴール，未だ満たされていないゴールを達成ゴールと呼ぶ[6]．筆者の所属するチームで求められている，「今までにない新機能を付加する」というのは，達成ゴールを見出し，それを実現する手段を考案・実装することである．達成ゴールが決まれば，既存のゴール分析の手法などで，実現方法を考えることは可能である．よって，本研究の目的を達成するためには，ステークホルダの達成ゴールを見出す必要がある．

先に示した独自ツールの課題は以下の 3 つである．

- 目的と手段の関係が階層構造をもっていない．そのため，複雑な関係を分析することができない．
- ステークホルダ「その他」をさらに分類する必要があるが，「その他」を識別するための明確なルールがない．
- 新しい発想を創造するための系統的プロセスが定義されていない．

1 つ目の目的と手段との関係が一段階しか表現できないという課題を解決するために，我々はこの独自ツールと KAOS 法を統合する．また 2 つ目の課題を解決するために，人が担うロールによってステークホルダを分類する．これによって各ロールのユーザエクスペリエンスを分析し，課題を抽出するとともに，課題を階層的に分析することが可能となる．そして 3 つ目の課題を解決するために，達成ゴールを考えるための考え方を提供する必要がある．

3.2. ツールの開発

3.1 の手法により発想を支援するにあたり，視覚的に実践をサポートするツールを開発，提供する必要もある．KAOS 法と同様に，ゴールや手段を表すノードと，ゴールと手段の関係を表せられることが必要である．また，ステークホルダを分類するためのロールを表せることも必要である．それらに加え，新しい発想を促すための質問などを表示できるように設計する．

4. 提案手法

4.1. 表記法

提案手法は KAOS 法と同じように，ゴールを表すノードを作成することができ，その目的・手段の関係を矢印によって表すことができる．また各ノードは二段構成になっており，上部にゴールを，下部にそのゴールのロールを記入することができる．その外観を図 4 に示す．

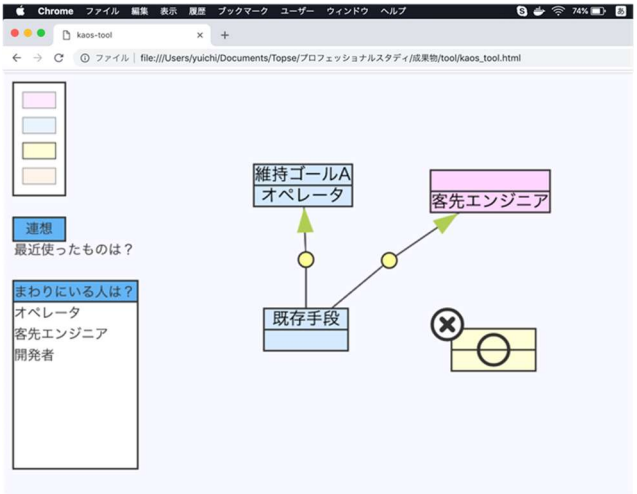


図 4 提案ツール外観

4.2. 思考プロセス

達成ゴールを分析するとき，次の 4 つのアプローチが考えられる．

I. 維持ゴールから考える「トップダウン」
II. 既存手段から考える「ボトムアップ」
III. 別のドメインから考える「横展開(1)」
IV. 別のロールから考える「横展開(2)」

これらの思考プロセスを支援する手法を検討しなければならない．以降の節で，各思考プロセスを支援するための手法について述べる．

4.2.1. 維持ゴールから考える

ここでは，維持ゴールをサブゴールと捉えたとき，それが満たす上位ゴールは何かと思考する．このように思考した結果得られる上位ゴールが達成ゴールとなる．なぜならば，この上位ゴールは未定義だったゴールであり，したがって，まだ達成されていないゴールだと考えられるからである．

このように達成ゴールを定義することができれば，次にこの上位ゴールを共有する，既定義の維持ゴールとは兄弟の関係にあるサブゴールは何か，を思考する．これによって，新手段を導出することができる．この概念図を図 5 に示す．

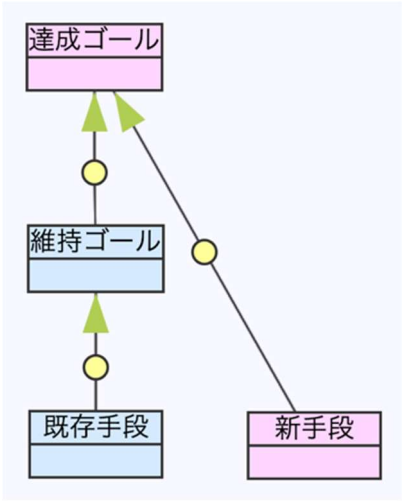


図 5 維持ゴールから考える

独自ツールでは，1 枚の紙に対して，目的が 1 つと，そのサブゴールとなる複数の機能要求を箇条書きする構造になっている．目的をすでに規定した状態から始めるため，さらに上位の目的に遡って考えることが難しかった．

4.2.2. 既存手段から考える

維持ゴールを満たすための既存手段が，満たすことのできる他のゴールはないかと考える．例えば，当初は食材として売られていた重曹を，シンク周りの清掃という別の目的に使用するなどである．その概念図を図 6 に示す．

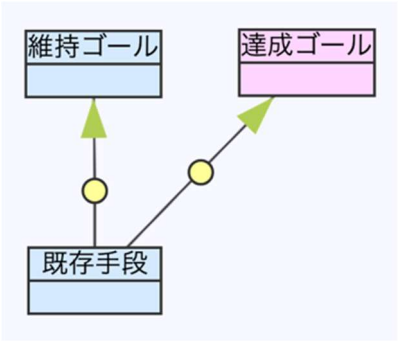


図 6 既存手段から考える

この概念も 4.2.1 節の「維持ゴールから考える」同様，独自ツールでは表現の難しかった点である．

4.2.3. 別のロールを考える

維持ゴールのロールとは別のロールを考えることで，新たな達成ゴールを見出す．既存手段に関わる人は誰か，またその人達は既存手段により，どのような恩恵を受けたいと思っているか，と考える．例えばカーナビゲーションの場合，「目的地までの道筋を知る」という維持ゴールのロールは運転手であるが，助手席の同乗者という別のロールを考えると「目的地までの所要時間を知る」という別の達成ゴールを考えることができる．その概念図を図 7 に示す．

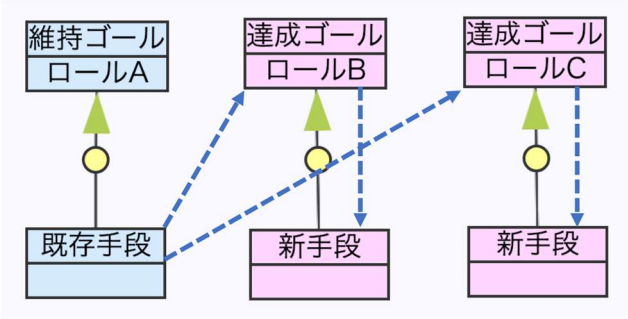


図 7 別のロールを考える

独自ツールでは，ゴールのロールは「使用者」，「開発者」，「その他」というように表されていた．ここでいう「使用者」とは上述の例では運転手のことであり，維持ゴールのロールを指している．新たなアプローチでは，ペルソナ分析のように，具体的なロールを規定することでよりそのゴールを想像しやすくしている．

4.2.4. 別の領域から考える

別の領域の維持ゴールと手段から，自領域の達成ゴールを見出す．別の領域の手段は，他の製品で関心のある機能はないだろうか，と考えることで導出する．別領域特有の維持ゴールを汎化し，自領域への転用を試みる．その概念図を図 8 に示す．

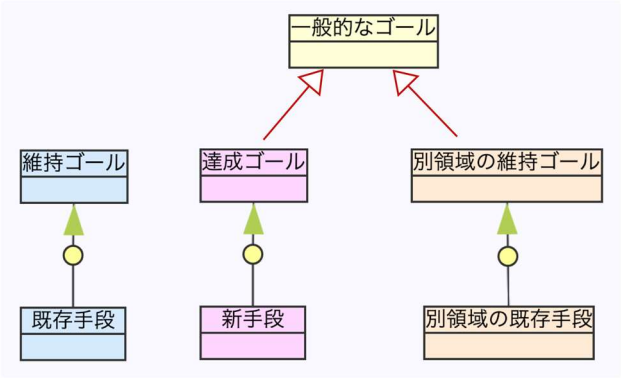


図 8 別の領域から考える

独自ツールでは別領域という概念を表す箇所はなく，サブゴールを列挙するときに自分でひらめくに任せていた．

4.3. ツール詳細

本ツールは Web ベースのアプリケーションとして，4.1 節で述べた表記法を踏襲している．また，4.2 節で述べた「別の領域から考える」，「別のロールから考える」を促進するために，図 4 の左側にあるように，連想のヒントを表示するスペース(図 9)と，既存手段に関連するロールを記述しておけるスペース(図 10)を設けた．

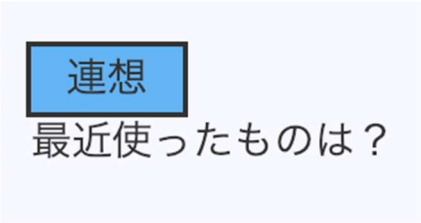


図 9 連想機能（連想ボタンをクリックすることで，別領域の維持ゴール・既存手段を発想するためのヒントが表示される）

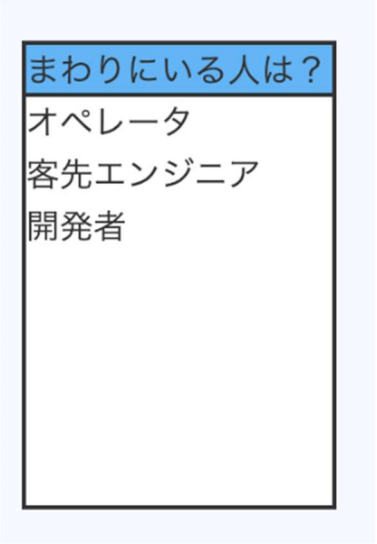


図 10 ロール記述スペース

4.4. 適用プロセス

新規製品開発時には，まず提案ツールを用いて,4 つの思考プロセスである，「維持ゴールから考える」，「既存手段から考える」，「別のロールから考える」，「別の領域から考える」のいずれか，もしくはすべてを実践し，達成ゴールを発想する．その結果，価値のある新機能を導くことができれば，プロセスは終了し，価値のある新機能を導くことができなければ，もう一度本ツールを用いて，達成ゴールの発想を試みる．以上の提案手法の適用プロセスを図 11 に示す．

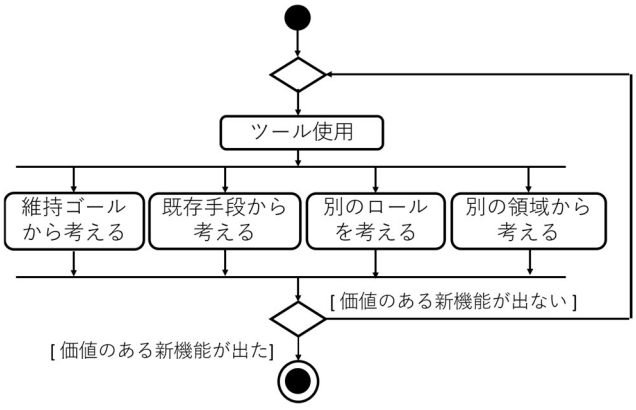


図 11 適用プロセス

5. 例題

上記提案手法が，実際の新既製品開発の過程を表現することができるかを検証するため，実際の新規製品の機能を用いた導入例を図 12 に示す．

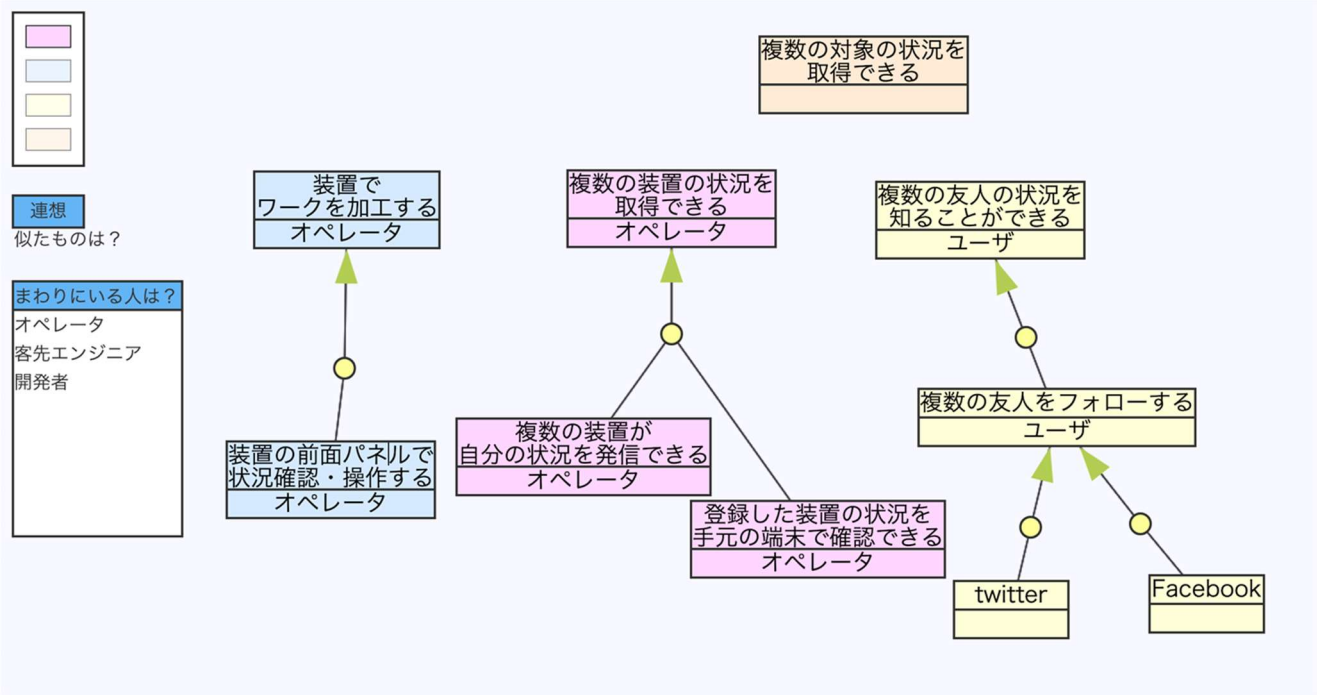


図 12 提案手法の使用例

製造工場においては，一人のオペレータが複数の産業用装置を操作している．ここで新しい価値を発想する際，まずツールの連想機能を使用し，「似た事例はないか」というヒントを得る．一人のオペレータに対し，複数の監視対象が存在するという面から，twitter や Facebook といった SNS を発想する．これら SNS が，どのような維持ゴールを満たしているか分析する．今回のケースでは，「複数の友人の状況を知る」という維持ゴールを導き出すことができる．この維持ゴールを汎化すると，「複数の対象の状況を取得できる」というゴールが得られる．この汎化されたゴールに対して，4.2.4 節の「別の領域から考える」思考プロセスを適用し，製造装置とオペレータの領域ではどのようなゴールになるのかを分析することで，「複数の製造装置の状況を取得できる」という達成ゴールを得られる．ここで得られた達成ゴールを満たす手段を考えることにより，「複数の装置が自分の状況を発信できる機能」や「登録した装置の状況を手元の端末で確認できる機能」を発想することができる．

以上の例より，新規製品開発時において本提案手法が十分な表現力を有しているといえる．

6. 考察

実際に現場に適用する際の課題としては以下があげられる．

- ツールを使うということへの心理的ハードルへの対処
- 実際に効果があるのかという疑念の解消

この課題を解決するための取り組みとして，まずはツールではなく，現状の仕事のやり方に近い，紙やポストイットなどを用いたアナログな手法を用いて，本手法の考え方の枠組みを導入するのが有効だと考えられる．また 4 章で示した 4 つのアイディアで多様なステークホルダの価値を導出するために，簡単な質問を用意し，思考支援を行なっていく．それらの質問が頭の中に入り，自問自答をするようになれば，達成ゴールを導出する確度も上がるだろう．事例数をこなすことによって，より効果的な質問内容や，ツールのブラッシュアップをすることもできる．

今まではひらめきに頼っていた新製品構想を，上記のような支援を継続し，改善していく予定である．

参考文献

[1] 大西淳，妻木俊彦，白銀純子，トップエスイー基礎講座 2 要求工学概論 要求工学の基本概念から応用まで，近代科学社，2009

[2] van Lamsweerde, A. Goal-oriented requirements engineering: A guided tour, in *Proc. Of 9th International Symposium on Requirements Engineering,* 2001, pp.249-263.

[3] Tsumaki, T., Tamai, T. A framework for matching requirements elicitation techniques to project characteristics. Software Process, Improvement and Practice 11(5), 505-519, 2006

[4] Nakatani, T., Tsumaki, T., Tamai, T. Instructional Design of a Requirements Engineering Education Course for Professional Engineers, Multimedia Services in Intelligent Environments, Software Development Challenges and Solutions, Springer pp.119-151, 2010

[5] P. Checkland, J. Scholes, Soft Systems Methodology in Action, Wiley, Chichester, 1990

[6] Anton, A. I. Goal Based Requirements Analysis, Proc. Second Int'l Conf. on Requirements Eng., ICRE '96, pp. 136–144, 1996.

日本語非機能要件の自動分類における教師あり学習アルゴリズムの評価

大東 誠弥
和歌山大学 システム工学部
ohigashi.seiya@g.wakayama-u.jp

福井 克法
和歌山大学大学院 システム工学研究科
fukui.katsunori@g.wakayama-u.jp

宮崎 智己
和歌山大学大学院 システム工学研究科
miyazaki.tomoki@g.wakayama-u.jp

大平 雅雄
和歌山大学 システム工学部
masao@sys.wakayama-u.ac.jp

要旨

要件定義の際に作成される要件定義書に機能要件と非機能要件がある．中でも非機能要件は，明確な定義が存在せず文献によって定義が異なることがある *[1]*，といった理由により見落とされやすいことが知られている．また，要件定義書はシステムの規模が大きくなるにつれ要件も多くなる．大量の要件を目視で確認し，非機能要件の見落としを防ぐことは困難である．

非機能要件の見落としを未然に防ぐために非機能要件分類手法が提案されている．先行研究 *[2]* では，*k*-近傍アルゴリズム，*SMO* アルゴリズム，*Naïve Bayes* アルゴリズムを用いた場合の分類精度を比較し，*SMO* アルゴリズムが最も分類精度が高いことを示している．しかしながら，先行研究が分類の対象としている非機能要件はすべて英語で記述された非機能要件であり，英語と文構造が異なる日本語非機能要件の分類においても先行研究の手法が最も適しているとは限らない．

本研究では，日本語非機能要件の分類に適した教師あり学習アルゴリズムの評価を行う．また，品詞を限定した場合についての分類精度を比較するとともに，類似する単語を統一する単語の正規化処理を行った場合の分類精度についても調査する．評価実験を行った結果，日本語非機能要件の分類においても先行研究と同様に *SMO* アルゴリズムを用いた場合が最も精度が高いことが確認された．また，*SMO* アルゴリズムを用いる場合，分類に用いる単語を名詞のみ，類似単語を統一することで，精度が向上することが確認された．

1. はじめに

ソフトウェア開発は，大きく分けて要件定義，設計，実装，テストの工程に分かれている．その中でも要件定義は，発注者が，開発者にどのようなシステムの開発を依頼するか定義する工程である．要件定義での定義が曖昧であったり間違った定義を行ってしまうと，開発の途中で手戻りが発生したりシステム障害の原因となることがある．また，要件定義では，システムのセキュリティやシステムの仕様についての定義も行う．これらのことから，要件定義は重要視されている工程の 1 つとなっている．

要件定義の際に作成されるのが要件定義書であり，要件定義書内に記述される要件には，機能要件と非機能要件がある．中でも非機能要件は，明確な定義が存在せず文献によって定義が異なることがある [1]．また，非機能要件は，機能要件と混在して記述される [3]，要件定義書の記述者によって記述表現が異なる [4]，定義が曖昧になる [5] などの問題があり，機能要件と比較すると見落とされやすいことが知られている．要件定義書はシステムの規模が大きくなるにつれ要件も多くなる．大量の要件を目視で確認し，非機能要件の見落としを防ぐことは困難である．

非機能要件の見落としを未然に防ぐために，非機能要件分類手法が提案されている [6,7]．先行研究では，機械学習を用いて非機能要件の分類を行っている．Slankas らの研究 [2] では，線形アルゴリズムである SMO アルゴリズムが最も高い分類精度になることを示した．また，分類精度向上のために分類に用いる品詞に注目し，冠詞

を除いて分類を行うことで分類精度が向上することを示した．しかしながら，先行研究で分類の対象としている非機能要件は，すべて英語で記述された非機能要件である．英語と日本語では文の構造が異なる．また，文を解釈する際に重要になる品詞も異なる．そのため，日本語非機能要件を分類対象とした場合，先行研究の手法が最も適しているとは限らない．

本研究では，日本語非機能要件の分類に適した教師あり学習アルゴリズムの評価を行う．また，品詞を限定した場合についての分類精度を比較するとともに，類似する単語を統一する単語の正規化処理を行った場合の分類精度についても調査する．

2. 非機能要件分類

2.1. 非機能要件

要件定義はシステム開発の初めに行う工程であり，発注者が開発者にどのようなシステムの開発を依頼するか定義する工程である．要件定義では，システムの概要，システム要件，システムを作成する目的，開発スケジュールなどが決められる．設計や実装，テストを行う開発者は，要件定義で決めた内容をもとに各工程を行う．要件定義が適切に行われていないと，他の工程に影響を及ぼすため，要件定義は重要な工程の 1 つとなっている．

要件定義で決めた内容を記した書類は要件定義書と呼ばれる．要件定義書に記述される要件のうち，システム要件には大きく分けて機能要件と非機能要件が存在する．

- 機能要件：システムの機能について定義した要件
- 非機能要件：システムの機能以外に関わるセキュリティやシステムの性能について定義した要件

機能要件と非機能要件は，それぞれ章ごとに分けて記述されていることがほとんどである．しかし，実際には機能要件を記述している章の中に非機能要件が混在している場合がある [3]．非機能要件は明確な定義が存在せず，文献によって定義が全く異なる [1]．また，要件定義書の記述者によって記述表現が異なる [4]，定義が曖昧になる [5] ことがあり，他の開発者が誤った解釈をしてしまうこともある．そのため，非機能要件を目視によって，要件定義書内から特定することは困難である．

2.2. 先行研究

近年，機械学習を用いた非機能要件分類手法が提案されている [2,8–11]．教師あり学習，半教師あリ学習，教師なし学習それぞれを用いた手法が提案されているが，本研究では教師あり学習を用いた手法に着目する．

Zhang らは SVM を用いた非機能要件分類手法を提案している [8]．分類に用いる特徴量に，単語のみ（BoW ベクトル），文字 N-gram，形容詞・名詞・前置詞から構成される複合語，の 3 つのパターンを設けて実験を行った結果，単語を特徴量に用いた場合が精度が最も高かった．

Slankas らは k-近傍アルゴリズムと SMO アルゴリズム，Naïve Bayes アルゴリズムを用いた非機能要件分類手法を提案している [2]．ナイーブベイズと SMO を用いる場合，要件文中に出現する単語を特徴量とした BoW ベクトルを用いて分類器を構築している．k 近傍法を用いる場合，要件文間の単語の編集回数（挿入・削除・置換）を用いて分類器を構築している．実験の結果，SMO アルゴリズムが最も精度が高く，次いで k-近傍アルゴリズム，Naïve Bayes アルゴリズムの順に精度が高いことを示した．

また，Riaz らはセキュリティ要件の詳細な分類を対象とした分類手法を提案している [10]．先行研究 [2] と同様の方法で分類器を構築しており，評価実験の結果では SMO と k 近傍法の精度が高かった．

これら先行研究で用いられたデータセットは，すべて英語で記述された非機能要件である．[2,10] で最も分類精度が高かった SMO アルゴリズムが，日本語非機能要件の分類に対しても最も精度が良くなるとは限らないため，本研究では [2] で用いられた 3 つの教師あり学習アルゴリズムを日本語非機能要件の分類に適用して評価を行う．

3. 研究の目的

本研究の目的は，日本語非機能要件の分類に適した教師あり学習アルゴリズムを評価することである．本研究では以下のリサーチクエスチョンを設定する．

RQ1：日本語非機能要件の分類において，最も分類精度が良い教師あり学習アルゴリズムは何か？

先行研究 [2] では，英語で記述された非機能要件について，k-近傍アルゴリズム，SMO アルゴリズム，Naïve

表 1. ISO25010 による非機能要件の品質特性

非機能要件の種類	説明
機能適合性	明示された状況下で使用するとき，明示的ニーズ及び暗黙のニーズを満足させる機能を，製品又はシステムが提供する度合い．
性能効率性	明記された状態 (条件) で使用する資源の量に関係する性能の度合い．
互換性	同じハードウェア環境又はソフトウェア環境を共有する間，製品，システム又は構成要素が他の製品，システム又は構成要素の情報を交換することができる度合い，およびその要求された機能を実行できる度合い．
使用性	明示された利用状況において，有効性，効率性及び満足性を持って明示された目標を達成するために，明示された利用者が製品又はシステムを利用することができる度合い．
信頼性	明示された時間帯で，明示された条件下に，システム，製品又は構成要素が明示された機能を実行する度合い．
セキュリティ	人間又は他の製品もしくは，システムが認められた権限の種類及び水準に応じたデータアクセスの度合いをもてるように，製品又はシステムが情報及びデータを保護する度合い．
保守性	意図した保守者によって，製品又はシステムが修正することができる有効性及び効率性の度合い．
移植性	1 つのハードウェア，ソフトウェア又は他の運用環境若しくは利用環境からその他の環境に，システム製品又は構成要素を移すことができる有効性及び効率性の度合い．

Bayes アルゴリズムの 3 種類の教師あり学習アルゴリズムを用いて分類精度の比較を行っている．しかしながら，日本語非機能要件の分類を行う場合には，英語と日本語とで文の構造が異なっている点，日本語には主語を省略することがある点の二点の違いが挙げられる．以上の二点から，必ずしも先行研究で最も精度が高かったアルゴリズムが日本語非機能要件の分類にも適しているとは限らない．RQ1 では，日本語非機能要件に適した教師あり学習アルゴリズムを比較実験により明らかにする．

RQ2 : 分類に使用する品詞の限定や，単語の正規化を行うことで分類精度に違いは生じるか？

先行研究 [2] では，分類に使用する品詞から冠詞を取り除くことで分類精度が向上することを明らかにしている．しかしながら，日本語非機能要件の分類において，同じように品詞を限定した場合に精度が良くなるとは限らない．また，品詞の問題だけでなく，英語とは違い日本語の場合，書き手によって要件文の語尾が異なることや要件の表現が異なることが考えられる．RQ2 では，分類に使用する品詞と，類似する単語の統一（単語の正規化）に着目して分類精度を比較する．なお，分類に使用する品詞は，独立して意味を持つ名詞と動詞に着目して分類を行う．

4. ケーススタディ

4.1. データセット

本研究では，データセットとして厚生労働省が公開している要件定義書，調達仕様書を使用する．データセットとして使用した，要件定義書及び調達仕様書の内訳及び，各システムの要件数を表 2 に示す．

表 2. 各システムの要件数

	ハローワーク (4 件)	労働保険 (6 件)	年金 (2 件)	合計 (12 件)
機能適合性	12	12	9	33
性能効率性	45	98	72	215
互換性	30	21	22	73
使用性	48	14	1	63
信頼性	21	54	33	108
セキュリティ	97	93	94	284
保守性	89	188	128	405
移植性	25	28	16	69
その他	561	654	521	1,736
合計	901	1,134	881	2,916

厚生労働省の要件定義書と調達仕様書を選んだ理由は以下の通りである．

- インターネット上に公開されている要件定義書及び調達仕様書の数が豊富であること
- 公開されている文書が pdf 形式で用意されており，データを入手しやすいこと

要件文には非機能要件の分類ラベルが付いていないため，要件定義書から 1 文ずつ抜き出し，1 文ごとに ISO25010 の品質特性に基づき手作業でラベル付けを行う．ISO25010 が定めた品質特性の項目は表 1 の通りである．なお，8 つの品質特性のどれにも属さないと判断されたものに対しては,「その他」のラベルを付けて対応した．ただし，表形式で要件が記述される場合があり，この場合は図 1 のように表の 1 列分を 1 文と解釈する．

No	利用者	機能	スループット	備考
1	すべての拠点の利用者	適用徴収機能	382,000件／日 35.370 件 ／ 秒 (※1)	※1 ピーク時アクセス件数（件／日）÷（3時間×3600秒）で算出。
2	インターネット経由でアクセスする利用者	適用事業場公開機能	1,433,000 ページビュー／月	

図 1. 表形式の記述例

職員等の利用するシステムは、使いやすい操作性を持たせること。	使用性
本調達に係る業務で発生し得るセキュリティリスクについて、受注者において対策を実施すること。	セキュリティ
一連の処理又はリクエストを発行してからレスポンス（画面表示）が返るまで、5秒以内とすること	性能効率性

図 2. ラベル付けの一例

また，要件定義書や調達仕様書には，機能要件や非機能要件だけでなく，調達の背景や目的，開発スケジュールなどが記されている．本研究では，これらの記述内容については対象外とする．また，図として記述された要件についても対象外とする．

ラベル付けを行うにあたり，個人でラベル付けを行った場合に主観が混入する恐れがあるため，ソフトウェア工学の知識を有する学生 2 名でラベル付けを行った．この際，他者が付けたラベルを見てラベル付けを行ってしまうとお互いの意見に影響される可能性がある．そのため，それぞれ独立してラベル付けを行い，その後 2 名でラベルの照らし合わせを行った．両者が異なったラベルを付けていた場合は，話し合いにてどちらかのラベルに決定するか，もしくは，1 文に対して 2 つのラベルを付けた．ラベル付けの一例を図 2 に示す．

4.2. 分類手順

本節では，設定した RQ1,2 について分析を行うための手順について説明する．手順は以下の 4 つのステップから構成される．

1. 要件文を単語ごとに分割するために形態素解析を行う
2. 教師あり学習アルゴリズムで扱うために，自然言語で記述された要件文を Bag of Words を用いてベクトル化する

```
脆弱性の対策を行った
脆弱性	名詞,固有名詞,一般,*,*,*,脆弱性,ゼイジャクセイ,ゼイジャクセイ
の	助詞,連体化,*,*,*,*,の,ノ,ノ
対策	名詞,サ変接続,*,*,*,*,対策,タイサク,タイサク
を	助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
行っ	動詞,自立,*,*,五段・ワ行促音便,連用タ接続,行う,オコナッ,オコナッ
た	助動詞,*,*,*,特殊・タ,基本形,た,タ,タ
EOS
```

図 3. MeCab で形態素解析を行った例

3. RQ1 の実験として各要件文の BoW ベクトルを入力として，各教師あり学習アルゴリズムを適用し要件文の分類を行う
4. RQ2 の実験として分散表現を利用して類似する単語の統一（単語の正規化）を行う

4.2.1. 形態素解析

形態素解析とは，1 文を単語ごとに分割する手法である．本研究では，オープンソース形態素解析エンジンの MeCab [12] を使用して形態素解析を行う．なお，MeCab 標準の辞書では,「厚生労働省」のような固有名詞を単語に分割しようとした場合,「厚生」と「労働省」といった分割をしてしまう．この問題を解決するため，より単語の登録数の多い辞書である mecab-ipadic-NEologd [13] を使用した．MeCab を用いて形態素解析を行った例を図 3 に示す．MeCab の出力から後述する単語の分散表現を作成し，類似する単語の統一（単語の正規化）を行う．

4.2.2. Bag of Words

自然言語で記述された要件文のままでは教師あり学習アルゴリズムを用いて分類することができない．本研究では，一般的なベクトル化手法である Bag of Words を用いて各要件文のベクトル化を行う．Bag of Words とは，出現する単語の各回数を特徴量としてベクトル化する手法である．本研究で扱う Bag of Words は，各要件文を行，各要件文に出現する単語の各回数を列とした行列である．

4.2.3. 教師あり学習アルゴリズム

Bag of Words を用いて生成した行列を入力として，教師あり学習アルゴリズムを用いて分類を行う．分類に使

用する教師あり学習アルゴリズムは [2] において採用されていた以下の 3 種類である．

- k-近傍アルゴリズム：学習データとテストデータそれぞれの距離を計算しておき，未知データ（テストデータ）から最も近い距離となる k 個の学習データを取得し多数決でクラスを決定する．なお，本研究では，[2] と同様に $k = 1$ で距離計算を行っている．
- SMO アルゴリズム [14]：学習データとテストデータそれぞれに対して，マージンと呼ばれる 2 クラスの境界面から各クラスに属するデータの最短距離が最大となるよう境界線を引いて分類を行う．
- Naïve Bayes アルゴリズム [15]：ベイズの定理に基づいてテストデータが属するクラスの確率を算出し，確率の最も高くなったクラスがテストデータの属するクラスと推定する．

4.2.4. 類似単語の統一処理（単語の正規化）

RQ2 についての実験では，1 文ごとに分割した単語を用いてベクトル化を行う前に，分類に用いるすべての単語のうち類似度の高い単語同士の置換を行う．単語の正規化を行う理由は以下の 2 点である．

- 要件定義書ごとに要件文に表記揺れがあり，同じ内容の要件文でも語尾が変化すること，表現の仕方が変わることで，別の要件文に分類される可能性があること
- 日付や時間などに関する要件文を Bag of Words を用いてベクトル化した場合，同じ日付および時間に関する要件であっても，数字が異なるだけで全く別の非機能要件と分類される可能性があること

単語の正規化を行うために本研究では単語の分散表現手法を用いる．単語の分散表現手法は，単語を任意の次元ベクトルで表現する手法である．単語をベクトルで表現することにより，ベクトル間の類似度を計算し単語間の類似度を算出することができる．

本研究では，分散表現手法のうち Word2Vec [16] を用いることで単語のベクトル表現を獲得する．Word2Vec は，2 層のニューラルネットワークを用いた分散表現手法である．Word2Vec では，単語の意味は周辺の単語によって構成されるという仮説に基づいて学習が行われる．

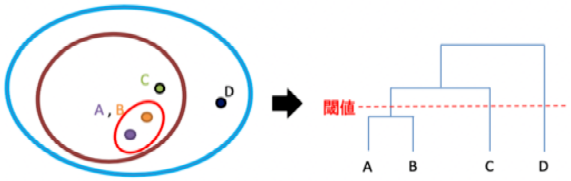


図 4. 樹形図作成の例

つまり，周辺に位置する単語が似ている単語同士は類似した単語であるとみなすアルゴリズムである．

本研究では，鈴木らが作成した日本語 Wikipedia エンティティベクトル [17] を用いて分散表現を取得する．日本語 Wikipedia エンティティベクトルは，Word2Vec [16] を用いて日本語版 Wikipedia を学習させている．

分散表現を取得した後，階層クラスタリングを用いて樹形図を作成する．分散表現の各単語間の距離計算は，コサイン類似度を用いて計算する．また，クラスタ間の距離計算には群平均法を用いる．階層クラスタリングを用いた樹形図作成の例を図 4 に示す．

図 4 のように閾値以下の集合を単語の類似集合とすることで，類似する単語を代表単語で置換することにより統一することができる．閾値は 0 から 1 の値を取り，閾値が大きくなるほどより多くの単語が類似していると判定する．集合に含まれる単語の中から任意の単語を代表単語としてその他の単語を代表単語と置換する．なお，閾値の設定によって抽出されるクラスタが異なるため，置換すべき単語は閾値に依存する．そのため，予備実験を行い最も精度が良かった時の閾値を採用する．

4.3. 分類精度の比較方法

教師あり学習アルゴリズムの分類精度を比較するために，式 1 で表せる適合率 (Precision)，式 2 で表せる再現率 (Recall)，式 3 で表せる適合率と再現率の調和平均である F 値 (F measure) を用いる．教師あり学習アルゴリズムが予測した非機能要件分類を集合 A，実際の要件文に割り当てられている非機能要件分類を集合 B（正解集合）とした場合，適合率は予測した分類の中に正解がどのくらい含まれていたかを表し，再現率は実際の正解集合の中に予測した分類がどのくらい含まれていたかを表す．例えば，人手によって機能適合性のラベルが付けら

表 3. 教師ありアルゴリズムごとの分類精度

	k-近傍			SMO			Naïve Bayes		
	P	R	F 値	P	R	F 値	P	R	F 値
機能適合性	**0.359**	**0.191**	**0.240**	0.247	0.164	0.186	0.102	0.053	0.068
性能効率性	0.711	0.668	0.687	**0.789**	**0.755**	**0.770**	0.594	0.686	0.635
互換性	0.273	0.198	0.224	**0.394**	**0.295**	**0.331**	0.128	0.138	0.131
使用性	0.726	0.402	0.508	**0.777**	**0.528**	**0.620**	0.422	0.251	0.307
信頼性	0.354	0.343	0.344	**0.475**	**0.420**	**0.442**	0.305	0.293	0.295
セキュリティ	0.578	0.648	0.610	**0.759**	**0.724**	**0.740**	0.456	0.686	0.546
保守性	0.474	0.458	0.465	**0.513**	**0.516**	**0.513**	0.263	0.512	0.347
移植性	0.335	0.354	0.339	**0.444**	**0.392**	**0.409**	0.237	0.303	0.263

表 4. 品詞の限定や正規化を行った際の分類精度（平均値）

		k-近傍			SMO			Naïve Bayes		
		P	R	F 値	P	R	F 値	P	R	F 値
正規化なし	品詞すべて	0.476	0.408	0.427	0.550	0.474	0.501	0.313	0.365	0.324
	名詞	0.490	0.432	0.450	**0.572**	**0.494**	**0.523**	0.294	0.395	0.322
	名詞，動詞	0.495	0.412	0.438	0.556	0.486	0.511	0.308	0.382	0.326
正規化あり	品詞すべて	0.471	0.417	0.431	0.555	0.483	0.510	0.317	0.393	0.338
	名詞	0.489	0.432	0.450	**0.571**	**0.501**	**0.526**	0.285	0.417	0.326
	名詞，動詞	0.491	0.412	0.436	0.566	0.493	0.519	0.303	0.400	0.333

れた要件文が 33 件，教師あり学習アルゴリズムによって機能適合性と予測し分類した要件文が 40 件，予測した 40 件のうち機能適合性と正しく分類できた要件文が 20 件であったとする．その時，適合率は $20 \div 40 = 0.500$ と算出でき，再現率は $20 \div 33 = 0.606$ と算出することができる．

$$Precision = \frac{|A \cap B|}{|A|} \tag{1}$$

$$Recall = \frac{|A \cap B|}{|B|} \tag{2}$$

$$F - measure = \frac{2 \times P \times R}{P + R} \tag{3}$$

5. 結果

5.1. RQ1：日本語非機能要件の分類において，最も分類精度が良い教師あり学習アルゴリズムは何か？

RQ1 では，それぞれの教師あり学習アルゴリズムについての分類精度を評価するために，適合率，再現率および適合率と再現率の調和平均である F 値を用いて分類精度の評価を行った．表 3 に 3 種類の教師あり学習アルゴリズム，*k*-近傍アルゴリズム，SMO アルゴリズム，Naïve Bayes アルゴリズムを用いて非機能要件を分類した際の分類精度を示す．

表 3 より，3 種類の教師あり学習アルゴリズムのうち機能適合性以外の非機能要件分類では，SMO アルゴリズムが Precision，Recall，F 値のすべてにおいて最も精度が高くなることが分かる．

5.2. RQ2：分類に使用する品詞の限定や，単語の正規化を行うことで分類精度に違いは生じるか？

RQ2 では，分類に使用する品詞を限定した場合の分類精度，単語を正規化した場合の分類精度の評価を行った．表 4 に分類に用いる品詞を限定した場合，単語の正規化を行った場合についてのそれぞれの分類精度を示す．なお，Precision，Recall，F 値のそれぞれは，8 つの非機能要件分類の結果の平均値を示している．

表 4 より，8 つの非機能要件分類の結果の平均値として SMO アルゴリズムが Precision，Recall，F 値のすべてにおいて最も精度が高くなることが分かる．また，品詞を名詞に限定することが分類精度に最も寄与することが分かる．正規化処理の効果については，SMO アルゴリズムにおいて正規化を行わなかった場合に僅差ながら適合率が最も高く，正規化を行った場合には再現率および F 値が最も高い結果となった．

次章では，RQ1 および RQ2 の結果を踏まえて分析を追加し考察を行う．

6. 考察

6.1. 日本語非機能要件の分類に適したアルゴリズム

RQ1 では，3 つの教師あり学習アルゴリズムを用いて分類を行った．先行研究 [2] と同様に，SMO アルゴリズムの分類精度が最も高い結果となり，Naïve Bayes アルゴリズムが最も低い分類精度となった．ただし,「機能適合性」では SMO アルゴリズムが k-近傍アルゴリズムよりも分類精度が低い結果となった．[2] では，非機能要件の分類ごとに分類精度は算出しておらず（表 4 のように）平均値のみで議論されているため単純な比較はできないが，RQ1 において「機能適合性」のみ SMO アルゴリズムが k-近傍アルゴリズムに分類精度が劣っていた原因について考察するため，要件文に頻出する単語を分析する．

表 5 に,「機能適合性」に分類される要件文に頻出する単語の上位 10 件とその出現割合を示す．また，表 6 に，機能適合性以外の非機能要件分類の 1 例として,「使用性」に分類される要件文に頻出する単語の上位 10 件とその出現割合を示す．

表 5 より,「機能適合性」では，出現割合が最も高い単語が「機能」（7.9%）である.「機能」を含む全要件文 532 件中 42 件が「機能適合性」に該当することを意味する．RQ1 では品詞の限定（および単語の正規化処理）は行っていないため,「機能」に続いて動詞の「する」が出現回数の 2 番目として出現している．さらに,「こと」が続き，一般的に広く使われる単語が上位を占めている．実際,「する」「こと」は，全要件文 2,916 件中，それぞれ 2,831 件，2,082 件に出現しており,「機能適合性」を分類する上で重要な単語（特徴語）として機能していない可能性が高い．4 番目以降の単語についても一般的な単語と言えるものが多く，実際，出現割合も低い．

表 5. 「機能適合性」における出現上位 10 単語

出現単語	出現回数	出現割合
する	24/2035	1.2%
機能	18/346	5.2%
こと	12/1857	0.6%
提供	11/244	4.5%
及び	10/645	1.6%
者	8/749	1.1%
情報	8/307	2.6%
業務	5/408	1.2%
システム	5/636	0.8%
資料	5/138	3.6%

表 6. 「使用性」における出現上位 10 単語

出現単語	出現回数	出現割合
こと	49/1857	2.6%
する	41/2035	2.0%
等	32/830	3.9%
システム	16/636	2.5%
し	15/995	1.5%
利用者	15/72	20.8%
利用	14/133	10.5%
アクセシビリティ	13/16	81.3%
要件	13/178	7.3%
ユーザビリティ	12/13	92.3%

一方，表 6 より,「使用性」では，半数は出現割合が低い単語が含まれるものの,「ユーザビリティ」（94.7%）や「アクセシビリティ」（81.8%）などの単語は出現割合が高く，要件文の中でも「使用性」に関する要件を記述す

表 7. 全文での動詞の出現回数上位 5 件

出現単語	出現回数	出現割合
する	2831	34.0%
し	1302	15.6%
行う	366	4.4%
さ	295	3.5%
示す	280	3.4%

表 8. 最も分類精度が高かった組み合わせ

	アルゴリズム	分類に用いる品詞	単語の正規化	F 値
機能適合性	k 近傍	名詞と動詞	あり	0.251
性能効率性	SMO	品詞すべて	あり	0.779
互換性	SMO	名詞	あり	0.369
使用性	SMO	名詞	なし	0.666
信頼性	SMO	名詞	あり	0.504
セキュリティ	SMO	名詞と動詞	なし	0.755
保守性	SMO	名詞と動詞	あり	0.525
移植性	SMO	名詞と動詞	あり	0.430

る際に特徴的に使用される単語である可能性が高い．また,「機能適合性」は全要件文 2,916 件中 33 件のみが分類されている．したがって，学習データそのものの少なさが分類精度に影響を与えている可能性がある．実際，表 3 の結果から,「機能適合性」の分類精度はいずれのアルゴリズムにおいても実用性の観点で不十分であると言える．

以上より，本研究で比較したアルゴリズムの中では SMO アルゴリズムが日本語非機能要件分類に最も適していると考えられる．しかしながら，非機能要件の分類ごとに分類精度のばらつきが大きく，また，F 値が 0.70 を超えた「性能効率性」および「セキュリティ」についても実用の観点からはさらなる精度向上が必要である．今後は学習データを増やすなどともに，他の教師あり学習アルゴリズムの適用を検討する予定である．

6.2. 品詞を限定した場合の分類精度への影響

RQ2 では，品詞をすべて用いて分類を行った場合，名詞のみを用いた場合，名詞と動詞を用いた場合に着目して分類精度を比較した．表 4 より，すべての教師あり学習アルゴリズムにおいて F 値が高くなったのは，正規化を行った場合であった．また，分類に使用する品詞の違いと合わせた結果では，Naïve Bayes アルゴリズムの場合は正規化を行い，かつ品詞をすべて分類に用いた場合が最も F 値が高くなった．また，k-近傍アルゴリズムと SMO アルゴリズムは正規化を行い，かつ名詞のみで分類を行った場合が最も F 値が高い結果となった．

3 つの教師あり学習アルゴリズムのうち，最も分類精度が高い結果となった SMO アルゴリズムで，正規化を行い名詞のみを分類に使用した場合と，正規化を行い名詞と動詞を分類に使用した場合に着目すると動詞を取り除くことで分類精度が向上していることがわかる．名詞のみに限定した場合に最も分類精度が高くなった理由について考察するため，要件文内に出現する動詞の出現回数を分析する．表 7 にすべての要件文内に頻出する単語の上位 5 件とその出現割合を示す．

表 7 より,「する」および「する」の活用形である「し」と「さ」が多く出現していることが分かる．前節でも議論したように，動詞の「する」や「し」「さ」などは，どのような非機能要件においても出現すると考えられる．例えば,「制御する」を形態素解析し品詞ごとに分割した場合，名詞の「制御」と動詞の「する」の 2 つに分割される．一方で，英語の「制御する (control)」を形態素解析すると,「control」という 1 単語の動詞で表される．このように，動詞だけで特徴となりえる英語とは異なり，日本語ではすべての文で多く出現する「する」などの動詞を含むこととなり，動詞を加えた場合に分類精度が下がったと考えられる．日本語にはサ行変格活用やナ行変格活用があり，形態素解析すると名詞と動詞に分割される場合が多いため，今後は名詞のみに限定して分類精度の向上に取り組む予定である．

6.3. 分類精度が最も高くなった組み合わせ

本研究では，先行研究 [2] を参考に，日本語非機能要件の分類に適したアルゴリズムを調べるための評価実験を行った．[2] のデータセットは英語の要件文であるため，本研究では厚生労働省が公開している要件定義書，調達仕様書計 12 件から抽出した合計 2,916 件の要件文を人手により分類（ラベル付け）した．そのため，非機能要件の分類ごとにデータの数のばらつき大きく,「機能適合性」などデータの少ないものは分類精度が著しく低いものになっていると考えられる．今後は，[2] と同程度の規模のデータセットを用いて精度向上を図る必要があるが，現時点での知見をまとめるために，各非機能要件分類の分類精度（F 値）が最も高くなった場合の組み

合わせを表 8 に掲載する．

まず，アルゴリズムについては,「機能適合性」を除く 7 つの分類で SMO の分類精度が最も高い結果となった．先行研究 [2] および [10] においても同様の傾向が示されていることから，日本語非機能要件の分類においても SMO が有効であることがわかった．ただし，３つの教師あり学習アルゴリズムを試行した段階であり，別のアルゴリズムの適用についても今後検討する必要がある．

分類に用いる品詞については,「性能効率性」が品詞すべてを用いた場合,「機能適合性」「セキュリティ」「保守性」「移植性」は名詞と動詞を用いた場合,「互換性」「使用性」「信頼性」は名詞のみを分類に用いた場合が最も高い精度となった.「機能適合性」「セキュリティ」「保守性」「移植性」では名詞と動詞を用いた場合に最も精度が高くなるが，前節で考察したように今後データを増やした場合にサ行変格活用やナ行変格活用の問題が生じ，分類精度が低下することが予想されるため，実験により検証する必要がある．

単語の正規化の有無に関しては,「使用性」と「セキュリティ」を除いた 6 つの分類で単語の正規化を行うことが好ましい結果となった．ただし，表 4 より，単語の正規化の効果は限定的であることも見て取れる．今後は正規化の効果をより高めるための工夫が必要となる．

7. まとめと今後の課題

本研究では，日本語非機能要件の分類に適した教師あり学習アルゴリズムを明らかにするために，先行研究で用いられた教師あり学習アルゴリズムによる分類精度の比較を行った．厚生労働省の 3 種類のシステムの要件定義書をデータセットに用いて，k-近傍アルゴリズム，SMO アルゴリズム，Naïve Bayes アルゴリズムの分類精度の比較を行った結果，先行研究 [2] と同様に SMO アルゴリズムの分類精度が最も高い結果となった．ただし，データ数が少ない「機能適合性」は k-近傍アルゴリズムを用いて分類する方が精度が高かった．また，最も精度が高かった SMO アルゴリズムで分類する場合，分類に用いる単語を名詞のみにすることで精度が向上することが確認された．英語とは異なり，日本語では名詞と動詞を組み合わせることで動作を表現することがあるため，日本語非機能要件の分類においては名詞のみを用いた方が精度が高くなったと考えられる．単語の正規化に関しては，8 つのうち 6 つの分類で好ましい結果となった．

本研究で用いたデータセットは非機能要件ごとにデータ数にばらつきがあり，少数しかない非機能要件がある．また，学習に用いたシステムは 3 種類と少なく，異なる種類のシステムの要件定義書を分類した場合に，本実験の結果と異なる可能性がある．そのため，先行研究 [2] で用いられたデータセットと同程度の規模のデータセットで実験を行い，分類結果の一般性を確かめる必要がある．

また，先述の通り，決定木などの本実験で用いたアルゴリズム以外の教師ありアルゴリズムも用いて評価実験を行い，日本語非機能要件の分類に適した教師ありアルゴリズムを明らかにする予定である．

謝辞

本研究の一部は，文部科学省科学研究補助金（基盤 (A): 17H00731，基盤 (C): 18K11243）による助成を受けた．

参考文献

[1] M. Glinz, “On non-functional requirements,” Proceedings of 15th IEEE International Requirements Engineering Conference (RE ’07), pp.21–26, 2007.

[2] J. Slankas and L. Williams, “Automated extraction of non-functional requirements in available documentation,” Proceedings of 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE ’13), pp.9–16, 2013.

[3] L.M. Cysneiros, “Evaluating the effectiveness of using catalogues to elicit non-functional requirements,” Proceedings of Workshop em Engenharia de Requisitos (WER ’07), pp.107–115, 2007.

[4] Z.S.H. Abad, A. Shymka, S. Pant, A. Currie, and G. Ruhe, “What are practitioners asking about requirements engineering? an exploratory analysis of social q a sites,” Proceedings of 24th International Requirements Engineering Conference Workshops (REW ’16), pp.334–343, 2016.

[5] A. Borg, A. Yong, P. Carlshamre, and K. Sandahl, “The bad conscience of requirements engineering: An investigation in real-world treatment of

non-functional requirements," Proceeding of 3rd Conference on Software Engineering Research and Practice in Sweden (SERPS '03), pp.1–8, 2003.

[6] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," Requirements Engineering, vol.12, no.2, pp.103–120, 2007.

[7] V.S. Sharma, R.R. Ramnani, and S. Sengupta, "A framework for identifying and analyzing non-functional requirements from text," Proceedings of the 4th International Workshop on Twin Peaks of Requirements and Architecture (TwinPeaks '14), pp.1–8, 2014.

[8] W. Zhang, Y. Yang, Q. Wang, and F. Shu, "An empirical study on classification of non-functional requirements," Proceedings of 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE '11), pp.444–449, 2011.

[9] A. Casamayor, D. Godoy, and M. Campo, "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," Inf. Softw. Technol., vol.52, no.4, pp.436–445, 2010.

[10] M. Riaz, J. King, J. Slankas, and L. Williams, "Hidden in plain sight: Automatically identifying security requirements from natural language artifacts," Proceedings of 22nd International Requirements Engineering Conference (RE '14), pp.183–192, 2014.

[11] A. Mahmoud, "An information theoretic approach for extracting and tracing non-functional requirements," Proceedings of 23rd International Requirements Engineering Conference (RE '15), pp.36–45, 2015.

[12] T. Kudo, K. Yamamoto, and Y. Matsumoto, "Applying conditional random fields to japanese morphological analysis," Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP '04), pp.230–237, 2004.

[13] 佐藤敏紀，橋本泰一，奥村　学，"単語分かち書き辞書 mecab-ipadic-neologd の実装と情報検索における効果的な使用方法の検討," 言語処理学会第 23 回年次大会発表論文集，pp.875–878，2017.

[14] J.C. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Technical report, Microsoft Research, 1998.

[15] G.H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," Proceedings of of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI '95), pp.338–345, 1995.

[16] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," Proceedings of 26th International Conference on Neural Information Processing Systems (NIPS '13), pp.3111–3119, 2013.

[17] 鈴木正敏，松田耕史，関根　聡，岡崎直観，乾健太郎，"Wikipedia 記事に対する拡張固有表現ラベルの多重付与," 言語処理学会第 22 回年次大会発表論文集，pp.797–800，2016.

軽量形式手法 VDM によるバーチャルマシンの開発

小田 朋宏
株式会社 SRA
tomohiro@sra.co.jp

荒木 啓二郎
熊本高等専門学校
araki@kyudai.jp

要旨

バーチャルマシン *(VM)* はソフトウェアで実装された仮想コンピュータシステムであり，プログラミング言語の実行系アーキテクチャとして広く採用されている．*VM* はユーザプログラムを実行するための基盤であり，高い信頼性と処理速度の両立が求められることから，その開発には高度に専門化された高い技能が要求されるとともに，安定した実装が得られるまで数年から *10* 年以上の長い期間を要することが多い．本稿では，*VM* の仕様記述に実行可能な形式仕様記述言語 *VDM-SL* を適用した事例として，現在進行している *ViennaVM* の開発を紹介し，*VM* 開発における軽量形式手法の効果を議論する．

1. はじめに

バーチャルマシン (以下，VM) はプログラミング言語の処理系の実装で採用されている実行系アーキテクチャである [1]．VM は仮想的なコンピュータシステムであり，命令セットやメモリモデル，外部リソース，外部入出力などを持つ．VM の命令セットはバイトコードまたは中間コード (以下，IR コード) セットと呼ばれ，メモリモデル，外部リソース，外部入出力と併せて，しばしば特定のプログラミング言語（以下，ゲスト言語）向けに設計される．例えば，Java VM (以下，JVM) はゲスト言語として Java 言語を想定して設計された IR コードセットを持つ [2]．JVM の IR コードセットは Java 言語でのメソッドの種類および呼び出し方法に応じて，invokedynamic，invokeinterface，invokespecial，invokestatic，invokevirtual など，それぞれ個別に特化されたメソッド呼び出しのための IR 命令が定義されている．プログラミング言語に特化しつつコンパクトな IR コードセットを定義することで，プラットフォーム間の移植性の高さと，階層化されたメモリを持つ現代的な計算機アーキテクチャにおける効率的な実行を両立した処理系を実装することができる．同一ソースプログラムを複数プラットフォーム上で動作させる場合，VM を界面として，プラットフォーム依存な部分を VM に集中させ，VM 上で動作する IR コード列はプラットフォーム非依存にすることができる．同一仕様の VM を複数プラットフォーム上に実装して，プログラムソースから単一のコンパイラを利用して IR コード列を生成し，IR コード列を複数プラットフォーム上で動作させることができる．

VM には利点がある一方で，その開発には高い技量を要するという課題がある．VM の仕様，特に IR コードセットの定義およびレジスタとメモリモデルの設計は，ゲスト言語を効率的に実行する上で重要である．シンプルかつゲスト言語の実行モデルに適合した IR コードセットを定義することで，ゲスト言語の単純なインタプリタよりもコンパクトな処理系を実現することができ，かつ，そのコンパクトさによって階層化されたメモリを持つ現代的な計算機アーキテクチャでの実行効率の改善が期待できる．レジスタおよびメモリモデルについても，高性能な VM を実現するためには，適切な抽象度により IR コードセットを単純化し，かつ，ゲスト言語でのメモリ使用の特性に適合する，オーバーヘッドの少ないシンプルなメモリモデルを設計する必要がある．

VM の仕様には高い厳密性が求められる．VM の利点の 1 つである移植性の高さは，同一仕様の VM はいずれの実装でも同一の仕様に従って動作することに依拠している．VM の仕様に解釈上の曖昧さがあると，ゲスト言語レベルにおいて，実装ごとに異なった振る舞いをする可能性がある．

実行系としての VM には処理速度と信頼性の高いレベ

ルでの両立が求められる．VM の実装には効率的な実行のために技芸的な技術がしばしば適用され，仕様に対して正しい実装となっているかを確認することは容易ではない．個々の IR コードの実装の正しさを確保するためには，適切な粒度と十分な厳密さを持つ IR コードセットの定義が求められる．メモリ管理，外部リソースや外部 IO についても同様である．また，IR コードレベルの最適化や IR コードからネイティブコードへの JIT コンパイルなどの技術が普及しつつあるが，これらの技術も VM 仕様に依拠したものである．

ViennaVM[4] は著者らが開発中の，実行可能形式仕様記述言語 VDM-SL を実行するための VM である．ViennaVM の開発では，VM の仕様は VM の実装の処理速度および信頼性の両方に大きな影響を与えると考え，仕様の品質を向上するためのアプローチとして VDM-SL による仕様記述を行なった．本論文では，2 節で VDM-SL の簡単な概要を説明し，3 節で ViennaVM のレジスタに関する特徴を説明し，4 節でレジスタへのアクセスおよび演算命令の VDM-SL 仕様と C 言語での実装を示す．5 節で，仕様記述工程および実装工程への影響についてテストを中心に議論し，また，現在の実装での性能を評価する．最後に，6 節で全体をまとめるとともに，今後の課題を示す．

2. VDM-SL の概要

本節では，ViennaVM の形式仕様記述言語として採用した VDM-SL について概要を説明する．VDM-SL はモデル規範型の形式仕様記述言語であり，仕様記述者は対象をモジュールに分解して，各モジュールが持つ機能を内部状態および内部状態に対する操作としてモデル化する [5].

VDM-SL では，システムをモジュール化し，各モジュールをデータ型，定数，関数，状態空間，操作の 5 種類の定義によって記述する．各モジュールの外部インターフェイスは `exports` 文によって宣言され，外部モジュールが定義する要素への参照は `imports` 文によって宣言する．各モジュールではデータ型を `types` 部で宣言し，定数および関数をそれぞれ `values` 部および `functions` 部で定義することによって，そのモジュールが扱う概念とその関係を記述する．VDM-SL では定数および関数は参照透明である．`state` 部では，モジュールが持つ内部状態を定義するために，静的に型付けられた変数のリスト，状態が常に守るべき不変条件および初期状態を宣言する．モジュールが提供する操作を `operations` 部で定義する．操作はプログラミング言語の手続きに相当し，静的に型付けされている．

VDM-SL は実行可能なサブセットを持つ．VDM-SL で記述された仕様は，関数や操作について評価する式や文を陽に記述し，かつ，`forall` などの限量子や有限集合の内包表記など，全要素の評価が必要な式について，型ではなく有限集合に束縛することで，インタプリタ等で実行することができる．VDMTools[6]，The Overture tool[7, 8]，ViennaTalk[9] などの VDM-SL 開発環境はインタプリタを持ち，ユーザは記述中の仕様をインタプリタ実行することで記述対象が持つ振る舞いや特性を理解することができる．さらに，vdmUnit や ViennaUnit[10] などの単体テストフレームワークを利用して，VDM-SL 仕様を継続的に単体テストすることが可能である．実際の開発適用事例においても，VDM-SL のオブジェクト指向拡張である VDM++[11] による仕様記述に対するテストが行われ，仕様記述の品質向上および成果物の信頼性向上に高い成果を上げている [12].

3. ViennaVM の特徴

ViennaVM は VDM-SL 仕様を実行することを目的とした VM で，VDM-SL 仕様から自動生成された IR コードを，IoT を含む多様なプラットフォーム上で実行することを想定している．VDM-SL 仕様からの実装の自動化として，VDM-SL から C++，Java，Smalltalk などの多様なホスト言語を対象とするプログラムソース自動生成器が研究され開発されている．プログラムソースの自動生成器は実装工程のコストを劇的に削減する一方，ホスト言語の言語仕様および処理系の更新に追従して保守する必要がある．VDM-SL の実行に必要な機能を持つ ViennaVM の仕様を厳密に記述し，様々なプラットフォーム上でそのプラットフォームに適したプログラミング言語で ViennaVM を実装することで，VDM-SL から単一の IR コード自動生成器によって多様なプラットフォーム上で動作する実装を得ることが ViennaVM 開発の目標である．

図 1 に ViennaVM のモジュール構成を示す．ViennaVM は，ViennaVM が扱うデータフォーマットを定義する Data モジュール，レジスタおよびレジスタに対する読み書きを行う Register モジュール，ヒープメモリの管理および

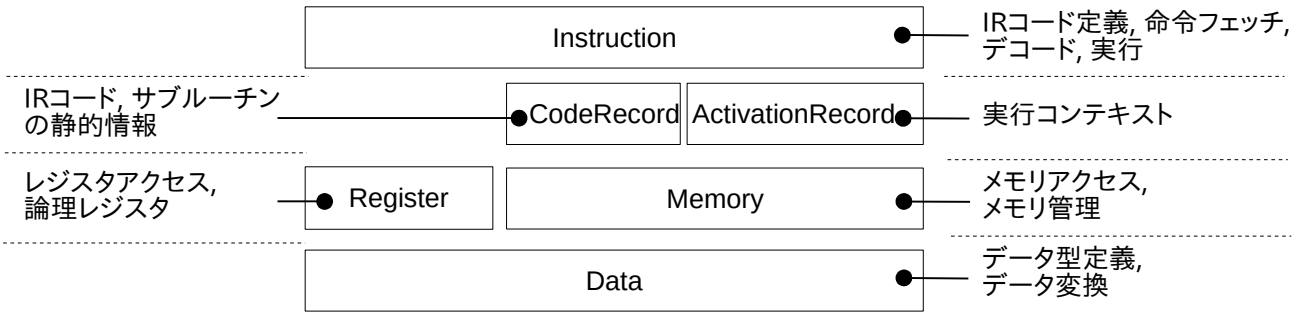


図 1. ViennaVM のモジュール構成

ヒープメモリに対する読み書きを行う Memory モジュール，コードブロックを管理する CodeRecord モジュール，実行コンテキストを管理する ActivationRecord モジュール，および，コードブロックから IR コードを読み出し実行する Instruction モジュールから成る．本論文では，ViennaVM のモジュールの中でも特に ViennaVM に特徴的な設計となっている Register モジュールに着目し，その特徴を説明する．

3.1. 安全で効率的なタグ付けとタグ除去

VDM-SL は静的型付け言語であるが，実行には実行時型情報が必要である．ViennaVM では，データオブジェクトを 64 ビットのタグ付きワードによって表現する．具体的には，63 ビット符号付き整数，32 ビット浮動小数点数，32 ビットユニコード文字，および 56 ビットポインタを 64 ビットのタグ付きポインタに符号化する．

データオブジェクトとしてタグ付きワードを採用する VM の多くは，メモリやレジスタにタグ付きワードを格納し，算術演算などタグを除去しての演算が必要な場合にのみ，演算処理を行う時に一時的にタグを除去したデータを取り出し，演算を行い，結果をタグ付きワードに変換する．この方式では，メモリ上のデータ表現をタグ付きワードに統一することで，タグ付きワードとタグを除去されたデータを混同する危険を減少させ，安全にデータを扱うことができる．一方で，複雑な算術演算を行う時には多くのタグ付けとタグ除去が実行されるという欠点がある．

冗長なタグ付けとタグ除去を回避するために，明示的にタグ付けおよびタグ除去を行う IR コードを定義するアプローチもある．IR コードが明示的にタグ付けおよびタグ除去を行う場合には，タグ除去されたデータオブジェクトの正当性は IR コードに依拠することになる．例えば，整数値を持つタグ付きワードをタグ除去し，演算を行った結果をポインタとしてタグ付けすることが可能になると，不正なアドレスへのメモリ書き込みが可能になる．

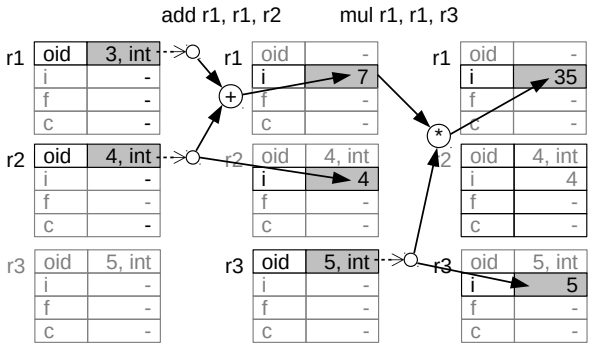


図 2. レコード型のレジスタによる自動タグ除去と演算の例

ViennaVM では，冗長なタグ付けとタグ除去を削減するために，タグ付与とタグ除去をレジスタ内部で処理するよう設計した．レジスタをタグ付きワード，符号付き整数，浮動小数点数，ユニコード文字の 4 つのフィールドを持つレコードとして，算術演算命令やメモリアクセスなどで利用される時にのみタグの除去およびタグの付与が実行され，その結果がフィールドに格納される．その後，同じレジスタに対する算術命令が実行される時には，フィールドに格納されたタグなしデータオブジェクトを直接読み込んで算術演算が実行される．これによっ

て，冗長なタグ付けの回避と安全なタグ付けおよびタグ除去を両立させている．

図 2 に ViennaVM の演算命令における自動的なタグ除去の例を示す．2 つの命令 `add r1, r1, r2` および `mul r1, r1, r3` によって，`r1 ← (r1 + r2) * r3` を実行するためのタグ除去と算術計算の手順を説明する．従来の，算術命令で常にタグ除去を行い，算術演算の結果にタグ付与を行う場合，各算術命令の前に 2 つのオペランドに対するタグ除去と，各算術命令後に 1 回のタグ付与が行われ，合計すると 4 回のタグ除去と 2 回のタグ付与が実行される．ViennaVM での自動的なタグ除去およびタグ付与では，3 回のタグ除去で同様な算術結果が得られる．

まず左端に初期状態として，3 つのレジスタ `r1`, `r2`, `r3` に，それぞれ整数 3, 4, 5 がメモリからロードされた状態を仮定する．ここでまず，命令 `add r1, r1, r2` を実行する．この命令は，2 つのレジスタ `r1` と `r2` を加算した結果を `r1` に格納する．レジスタ `r1` にはタグ付きワードフィールド（oid）のみ格納されているため，まずはタグを除去して整数 3 を得る．レジスタ `r2` も同様で，タグを除去して整数 4 を得る．加算した結果である整数 7 をレジスタ `r1` の整数フィールド（i）に格納する．また，レジスタ `r2` の整数フィールド（i）にもタグを除去した結果である整数 4 を格納する．

次に，命令 `mul r1, r1, r3` を実行する．この命令は，2 つのレジスタ `r1` と `r3` を乗算した結果を `r1` に格納する．レジスタ `r1` には整数フィールド（i）に 7 が格納されているため，これをそのまま利用する．レジスタ `r3` にはタグ付きワードフィールド（oid）のみ格納されているため，まずはタグを除去して整数 5 を得る．乗算した結果である整数 35 をレジスタ `r1` の整数フィールド（i）に格納し，レジスタ `r3` の整数フィールド（i）にもタグを除去した結果である整数 5 を格納する．

以上の手順で 2 つの算術演算命令が実行されるが，この中でタグ除去は 3 回のみ行われる．さらに，この後でレジスタ `r2` やレジスタ `r3` の整数値が必要な算術演算命令を実行する場合には，タグ除去なしで整数値を取り出すことができる．レジスタ `r1` にはタグ付きワードフィールド（oid）に値が格納されていないが，タグ付きワードが必要な命令を実行する時点でタグ付与を行なってタグ付きワードフィールドを得る．ViennaVM は以上の方法で，タグ付けされたデータに対して安全かつ少ないオーバーヘッドによる演算処理を実現している．

3.2. レジスタ空間

JVM を初めとする多くの VM はスタックマシンであり，手続きへの引数および返値の受け渡しはスタックを介して行われる．レジスタの内容を引数または返値としてメモリ上のスタックに書き込むと，第 3.1 節で説明した自動的にタグ除去されたデータオブジェクトが失われてしまう．また，ViennaVM では参照カウントによる自動ガーベージコレクション (自動 GC) を採用している．コピー GC など，参照を追跡していくことで参照のないメモリブロックを発見する他の自動 GC アルゴリズムに比べて，参照カウントによる自動 GC はカウンタの増減という比較的小さなオーバーヘッドで参照のないメモリブロックを自動的に発見し解放することができる．しかし，手続きへの引数や返り値としてポインタ値を保持しているレジスタをメモリ上のスタックに書き込むと，参照カウンタの増減処理が必要になり，メモリ管理のオーバーヘッドを増加させる．ViennaVM では，メモリ上のスタックを介したデータの受け渡しを避けるために，手続きへの引数および返値の受け渡しはレジスタを介して行う．

引数および返値の受け渡し以外に，スタックはレジスタの退避場所としても利用されることがある．一般に多くの VM では，スタックを使ってレジスタの内容の退避を行う．ViennaVM ではレジスタは 4 つのフィールドからなるレコードとして定義されるため，レコード全体をスタックに退避させレジスタに復元すると，データトラフィックが大きくなり，性能面で不利である．一方，ViennaVM ではヒープメモリ上のメモリブロックにはタグ付きワードのみを格納する．スタックにレジスタ内のデータのうちタグ付きワードを退避復元させる場合には，タグ付けおよびタグ除去の回数が増加し，これも性能面で不利になる．

ViennaVM では，IR コードのオペランドとして指定されるレジスタ番号にオフセットを加えたものを物理的なレジスタ番号として，オフセットを変更することでレジスタの退避および復元を行う．図 3 にレジスタ番号のオフセットによるレジスタ退避の例を示す．ViennaVM のサブルーチンは，そのコードが利用するレジスタ数を保持する．初期状態として，4 つのレジスタを使うサブルーチンをレジスタオフセット値が 3 で実行しているとする．この時，サブルーチン内では r1, r2, r3, r4 の 3 つの論理レジスタ名を利用することができ，それらはそれ

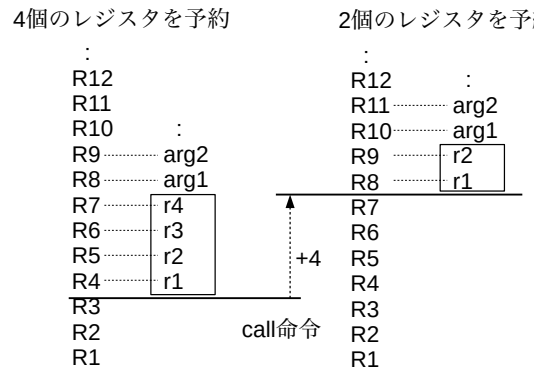


図 3. 論理レジスタ番号によるレジスタ退避の例

ぞれ実レジスタ R4, R5, R6, R7 を指す．このサブルーチンから，レジスタを 2 つ使う別のサブルーチンを `call` 命令で呼び出す時，引数は r5 相当の実レジスタ R8, r6 相当の実レジスタ R9 で渡される．`call` 命令はレジスタオフセットを呼び出し側のレジスタ数分増加させる．この例では，+4 増加させる．するとレジスタオフセットは 7 となり，呼び出された側のサブルーチンでは論理レジスタ名 `r1` および `r2` はそれぞれ実レジスタ `R8` および `R9` を指す．すなわち，第 1 引数として渡されたデータをレジスタ名 `r1` として受け取る．

サブルーチンからの復帰を行う `RET` 命令は，レジスタオフセットを復帰先のレジスタ数分だけ減少させるとともに，復帰元で使用していたレジスタの内容を初期化することで参照カウンタの増減処理を行う．

このレジスタ退避の方式の利点は，メモリトラフィックの節約である．例えば手続き呼び出しによってレジスタを 10 個退避し，後で復元する場合，レジスタレコード単位で退避する方式では 640 バイトのメモリ間データ転送が必要となる．タグ付きワードのみを退避する場合には，160 バイトのメモリ間データ転送と，最大 10 回のタグ付けと最大 10 回のタグ除去が必要となる．ViennaVM が採用するレジスタオフセットの場合，オフセット時のホスト言語での整数型の加減算のみでレジスタの退避および復元が可能である．

ViennaVM は実レジスタ番号および論理レジスタ番号ともに 16 ビットのレジスタ空間を持つ．Dalvik VM[3] をはじめとする広いレジスタ空間を持つレジスタマシン方式の VM では，豊富なレジスタ数を生かしてサブルーチン呼び出しのインライン展開をより深く行うような最適化を可能にしている．ViennaVM では，レジスタ番号をオフセットすることで，インライン展開をしなくても豊富なレジスタ数を有効に利用することができる．

4. ViennaVM の仕様と実装

本節では，ViennaVM の仕様と実装について，3.1 節および 3.2 節で説明したレジスタへのデータの読み書きおよび論理レジスタ番号に関する VDM-SL 仕様と C 言語による実装を説明する．

4.1. レジスタに対するデータの読み書き

3.1 節で説明した通り，ViennaVM では安全かつ効率的なタグ付きワードの扱いのために各レジスタに静的型ごとのフィールドを持たせ，必要に応じてタグ付けおよびタグ除去を行い，その結果を次の使用時に備えて適切なフィールドに格納する．これは ViennaVM 独自の設計であり，その正確な機能仕様を自然言語および表などの前形式的な表現で定義するだけでは解釈が一意に定まらない危険がある．また，解釈が一意に定まる場合でも，仕様に誤りがある場合に，自然言語などによる記述では誤りの発見が困難であり，しばしば実装後に誤りが発見され，手戻りの原因になり得る．

また，ViennaVM でのレジスタの読み書きには，タグ付けおよびタグ除去だけでなく，型変換も伴う場合がある．VDM-SL では整数型や自然数型が実数型の部分型として定義されている．VDM-SL の言語仕様では実数型の内部表現および精度に関する規定はなく，全ての整数値は実数型の値であり，`1.0` と `1` は同じ値を意味する．例えば，式 `1 = 1.0` は `true` である．

そこで，ViennaVM では，整数値が格納されたレジスタから浮動小数点数を読み出す時に，読み出し処理の中で整数値から浮動小数点数に変換した値が得られるものとした．本節ではそれが VDM-SL によってどのように定義されているのかを説明する．

図 4 に ViennaVM 内部でのデータ型定義の一部を抜粋する．`OID` 型がタグ付き 64 ビットワード，`Int` 型は 63 ビット符号付き整数，`Float` 型は 32 ビット浮動小数点数，`Char` 型は 32 ビットユニコード，`Pointer` 型は

```
types
  Qword = nat
    inv qw == qw < 0x10000000000000000;
  OID = Qword;
  Int = int
    inv i == (i > -0x8000000000000000
      and i < 0x8000000000000000)
    or i = invalidIntValue;
  Float ::
    sign : nat
    exponent : nat
    fraction : nat
        inv mk_Float(s, e, f) ==
          (s < 2 and e < 0x100) and
          f < 0x1000000;
  Char = nat
    inv f == f < 0x100000000
      or f = invalidCharValue;
  Pointer = nat
    inv p == p < 0x100000000000000
      or p = invalidPointerValue;
```

図 4. VDM-SL 仕様でのデータ型定義の抜粋

```
types
  Reg ::
    oid : Data`OID
    i : Data`Int
    f : Data`Float
    c : Data`Char;
  Register = nat;
```

図 5. レジスタの定義

56 ビットポインタとして定義されている．具体的な定義としては，タグ付きワード型 OID や整数型 Int がそれぞれ自然数型および整数型に不変条件を与えることによって定義している．また，浮動小数点数型 Float をレコード型で定義している．

図 5 に Register モジュールでのレジスタの定義を示す．レコード型 Reg が，oid，i，f，c の 4 つのフィールドから構成されるレコード型として定義されている．それぞれのフィールドには，図 4 で定義された OID 型，Int 型，Float 型，Char 型が静的につけられている．また，レジスタ番号を表す型 Register が自然数として Register モジュールで定義されている．

レジスタにデータを書き込む時には，まずは論理レジスタ番号から実レジスタ番号を求め，該当するレジスタの書き込みデータの型に対応するフィールドに書き込み，それ以外のフィールドを無効値に設定する．例として，レジスタに整数値を書き込む操作 write_int の仕様を図 6 に示す．5 行目はレジスタオフセットで論理レジスタ番号をオフセットして実レジスタ番号 r を求めている．8-9 行目は参照カウントによるメモリ管理に関する記述であり，書き込み前のレジスタがポインタであった場合に，書き込みによりその参照が失われるためメモリブロックの参照カウントを減らす．10 行目以降で，i フィールドに整数値を，それ以外のフィールドには無効値を書き込む．

```
operations
  write_int : Register * Data`Int ==> ()
  write_int(dst, i) ==
    let
      r : Register = baseRegister + dst,
      p = registers(r).oid
    in
      (if Data`isPointer(p)
      then Memory`decrement_reference_count(p);
      registers(r) := mk_Reg(
        Data`invalidTag,
        i,
        Data`invalidFloatValue,
        Data`invalidCharValue));
```

図 6. レジスタへの整数値の書き込みの仕様

レジスタからデータを読み出す時には，まず対応するフィールドの値が無効値でなければその値を返す．無効値であれば，互換性のある型のフィールドから型変換をしてフィールドに書き込んだ上で，読み出し値とする．

例として，レジスタから整数値を読み出す操作 read_int の仕様を，関連する 2 つの操作とともに図 7 に示す．操作 basic_read_int は，指定されたレジスタの i フィールドに値が格納されていたらそれを返し，さもなくば oid フィールド（タグ付きワード）に格納された値を整数型に型変換した結果を i フィールドに書き込んだ上で，それを返す．操作 basic_read_float は同様のことを f フィールドに対して行う．

前述のように，VDM-SL では，整数型 int および自然数型 nat および nat1 は実数型 real の部分型であり，例えば 1.0 は実数であるだけでなく整数型の値でもある．つまり，式 1 = 1.0 は true である．そこで ViennaVM では，浮動小数点数が格納されたレジスタからも，それが整数でもある場合には整数値として読み出し可能とした．整数 basic_read_int 操作は，指定されたレジスタから basic_read_int 操作で整数値を読み出し，もし結

```
operations
  basic_read_int : Register ==> Data`Int
  basic_read_int(src) ==
    let
      r : Register = baseRegister + src,
      reg : Reg = registers(r),
      i : [Data`Int] = reg.i
    in
      (if i <> Data`invalidIntValue
      then return i;
      let i2 : Data`Int = Data`oid2int(reg.oid)
      in
        if
          i2 <> Data`invalidIntValue
        then
          (registers(r) .i := i2;
          return i2);
      return Data`invalidIntValue);

  basic_read_float : Register ==> Data`Float
  basic_read_float(src) ==
    let
      r : Register = baseRegister + src,
      reg : Reg = registers(r),
      f : [Data`Float] = reg.f
    in
      (if f <> Data`invalidFloatValue
      then return f;
      let
        f2:Data`Float=Data`oid2float(reg.oid)
      in
        if
          f2 <> Data`invalidFloatValue
        then
          (registers(r) .f := f2;
          return f2);
      return Data`invalidFloatValue);

  read_int : Register ==> Data`Int
  read_int(src) ==
    (let i : Data`Int = basic_read_int(src)
    in
      if i <> Data`invalidIntValue
      then return i;
    let f : Data`Float = basic_read_float(src)
    in
      (if
        f <> Data`invalidFloatValue
      then
        (let
          r = Data`float2real(f),
          int_r = floor (r + 0.5)
        in
          (if r = int_r
          then
            (registers(baseRegister+src).i :=
              int_r;
            return int_r))));
    return Data`invalidIntValue);
```

図 7. レジスタからの整数値の読み出しの仕様

```
  idiv : Register`Register * Register`Register
         * Register`Register ==> ()
  idiv(dst, src1, src2) ==
    if
      (dst > 0 and src1 > 0) and src2 > 0
    then
      let
        int1:Data`Int=Register`read_int(src1),
        int2:Data`Int=Register`read_int(src2)
      in
        if
          int1 <> Data`invalidIntValue
            and int2 <> Data`invalidIntValue
        then
          Register`write_int(dst, int1 div int2)
        else
          err("idiv: operands not integer")
    else
      err("idiv: operands not specified");
```

図 8. idiv 命令の仕様

果が未定義値であった場合には，basic_read_float 操作で浮動小数点数値を読み出し，もし結果が未定義値でなく，かつ，整数値であったら，Int 型に型変換した結果を i フィールドに書き込んだ上で，それを返す．

上記の浮動小数点数から整数への型変換は，ViennaVM が VDM-SL に特化した VM であることから有用であって，浮動小数点数 1.0 と整数 1 が異なる数値として扱われる他の多くのプログラミング言語では不要である．VDM-SL で仕様を記述することで，上記のような複雑な内容を曖昧さなく厳密に記述することができる．

write_int および read_int 操作を使った IR コードの仕様の例として，図 8 に idiv 命令の仕様を示す．第 2 オペランドおよび第 3 オペランドとして指定されたレジスタから Register モジュールの read_int 操作で整数値をそれぞれ読み出し，両方のレジスタから正常に整数値が読み出せた場合に整数除算の結果を第 1 オペランドとして指定されたレジスタに書き込む．

idiv 命令の C 言語での実装を図 9 に示す．図 8 の VDM-SL 仕様に忠実な実装であることがわかる．

5. 議論

本節では，VM の開発に VDM-SL を導入したことの影響を議論する．VM に要求される品質項目で特に重視されるものとして，信頼性と性能に着目し，5.1 節でテストやデバッグについて，5.2 で性能面について論じる．

```
void idiv(Register dst, Register src1,
          Register src2) {
  if (dst && src1 && src2) {
    Int int1 = read_int(src1);
    Int int2 = read_int(src2);
    if (int1 != invalidIntValue
        && int2 != invalidIntValue) {
      write_int(dst, int1/int2);
      return;
    }
    err("idiv: operands not integer");
  } else {
    err("idiv: operands not specified");
  }
}
```

図 9. idiv 命令の C 言語での実装

5.1. テストおよびデバッグ

ViennaVM の VDM-SL 仕様で定義された関数および操作は，ユニットテストフレームワーク viennaUnit [4] を使ってテストされた．図 10 に，`Register` モジュールに対応するテストモジュール `RegisterTest` の抜粋を示す．ViennaVM の各モジュールに対応してテストモジュールを記述し，仕様を修正する度に自動的に全テストモジュールを実行した．また，VDM-SL で記述されたテストモジュールは C 言語に移植され，それを C 言語での実装に対する単体テストとして利用した．表 1 に VDM-SL 仕様および C 言語での実装およびテストモジュールの行数およびテストケース数をまとめた．

ViennaVM の開発では，VDM-SL 向けユニットテストフレームワーク ViennaUnit を利用した．ViennaUnit は，仕様を編集し保存する度に，IDE がバックグラウンドタスクとして全てのテストケースを実行し，テストが失敗した場合にのみデスクトップ上に通知を表示する機能を提供している．VDM-SL 仕様を記述したファイルへの編集が保存されると，ViennaUnit はモジュール名が `Test` で終わるモジュールをテストモジュールと見なし，テストモジュール内に定義された操作のうち操作名が `test` で始まる操作をテストケースと見做して実行する．この機能を利用することで，仕様記述者が明示的にユニットテストを実行するよりも，仕様記述者の負担を増やすことなく高頻度でテストを実行し，仕様の誤りを早期のうちに発見することができた．また，デバッグのための仕様修正に対しても高い頻度でテストが自動的に実行されたため，修正によるリグレッションを早期発見し，再修正することができた．

```
values
  int1 : Data`Int = 0x1234;

operations
  testWriteBoxedIntAndUbox : () ==> ()
  testWriteBoxedIntAndUbox() ==
    (Register`write_oid(1, Data`int2oid(int1));
    assertEquals(
      Register`read_int(1),
      int1,
      "int->int");
    assertEquals(
      Register`read_float(1),
      Data`real2float(int1),
      "int->float");
    assertEquals(
      Register`read_char(1),
      Data`invalidCharValue,
      "int-char");
    assertEquals(
      Register`read_pointer(1),
      Data`invalidPointerValue,
      "int->pointer"))
```

図 10. RegisterTest モジュールの抜粋

ViennaUnit は，表明検査のための操作を 2 つ提供している．`assert` 操作は，引数を 2 つ取り，第 1 引数が true であるかどうかを確認し，true でない場合に第 2 引数をエラーメッセージとしてテスト失敗とする．`assertEquals` 操作は，引数を 3 つ取り，第 1 引数と第 2 引数が等しいかどうかを確認し，等しくない場合に第 3 引数をエラーメッセージとしてテスト失敗とする．テストケースでは上記 2 つの表明検査を使って，テスト対象の動作を確認する．

ViennaVM での開発では，仕様の誤りの多くはデータ型に付与された不変条件に対する違反として検出された．テストケース内での `assert` 操作および `assertEquals` 操作はテスト対象の動作結果に対してのみ検査するのに対して，データ型に付与された不変条件はテスト対象となる動作の中間状態も含め継続的に検査が行われる．多くのテストケースでは，テストケースに直接記述された表明検査よりも，多くの種類のデータ型不変条件が，多くの回数で，テストケースの表明検査よりも先に検査されることが，データ型不変条件が仕様の誤りの発見に有効であった理由と考えられる．

C 言語での実装のユニットテストは，VDM-SL によるテストモジュールを C 言語に移植することで実装した．図 11 にレジスタへのデータアクセスのテストの一部を

表 1. 仕様および実装の行数およびテストケース数

モジュール名	VDM-SL 仕様の行数	VDM-SL テストの行数	C 実装の行数	C テストの行数
Data	279	209	138	57
Register	255	147	254	204
Memory	328	253	291	233
CodeRecord	100	105	127	119
ActivationRecord	63	29	50	31
Instruction	657	473	649	455
合計	1760	1131	1589	1136

```
static Int int1 = 0x1234;

static int testWriteBoxedIntAndUbox() {
  write_oid(1, int2oid(int1));
  assertEquals(
    read_int(1),
    int1,
    "int->int");
  assertEquals(
    read_float(1),
    (Float)(real2float((float)int1)),
    "int->float");
  assertEquals(
    read_char(1),
    invalidCharValue,
    "int-char");
  assertEquals(
    read_pointer(1),
    invalidPointerValue,
    "int->pointer");
  return 0;
}
```

図 11. Register モジュールの C 言語でのテストコードの抜粋

抜粋する．C 言語での実装に対するテストで発見された誤りの多くは実装上の誤りであった．例えばレジスタへのアクセスにおいてポインタを使って計算量を節約して発生した誤りが発見された．一方で，一般のプログラミングで発生する境界条件の誤りや場合分けの抜けなどは，現在のところ見つかっていない．仕様に対するテストで計算式やアルゴリズムの誤りが発見され修正されていたためであると考えられる．

形式手法を導入した開発では，仕様記述の工数が増加する一方で実装の工数が大幅に短縮されるフロントローディング効果が報告されている．VM 開発でもテスト工程で要する工数が仕様記述工程に前倒しされたと考えられる．VDM-SL の実行可能な仕様は，プロトタイプやリファレンス実装でのプログラムに近いように見えるが，システムが取りうる状態に対する制約条件を記述するモデル規範型の形式仕様記述言語としての言語機能が，機能定義の誤り発見と実装上の詳細の誤り発見を分離し，フロントローディング効果につながったと考えられる．

5.2. 性能

一般に，言語処理系のランタイムとしての VM に対する要求として，性能への優先順位は高い．VM を開発するにあたって，要求される性能が実現可能であるかどうかは，常に重大な関心事である．本節では，ViennaVM の開発に VDM-SL を導入した結果として，ViennaVM の C 言語での実装の性能をフィボナッチ数の計算で計測し，それを VDM インタプリタ VDMJ と Python3 と比較した．

表 2 にその計測結果をまとめた．実行環境として，サーバ機やデスクトップ PC 向けに広く普及している x64 アーキテクチャの Core i5 プロセッサと，組み込みなどで広

表 2. fib(30) による性能ベンチマーク

プロセッサ	実行時間 (ミリ秒)	
	ViennaVM	Python3
Core i5	348.7	466.7
ARMv7	5,107.1	6,140.6

Core i5: 2.5GHz (macOS Mojave, 64 ビット)
ARMv7: 1.4GHz (Raspbian Stretch, 32 ビット)

く普及している ARMv7 プロセッサを対象とした．各プロセッサ上で，VDMJ，ViennaVM，ベンチマークとして 30 番目のフィボナッチ数を計算するのに要する時間をそれぞれの処理系で 10 回計測し，その平均を求めた．いずれのプロセッサでも，Python 3 を上回る性能を示した．ただし，既に長く実用されている Python 3 に対して，ViennaVM は開発途上の段階であり，これからコード量が増加すると性能の劣化が起こる可能性がある．

6. まとめ

ViennaVM の開発に VDM-SL を導入し，C 言語による実装を行なった．VM の仕様の多くの部分は，VM の動作を記述するための，VM が扱うデータ型の定義や各 IR コードの処理内容の操作的な意味を与えることに占められる．VDM-SL を導入することで，VM の仕様を曖昧さなく記述し，仕様の誤りの発見と実装の誤りの発見を明確に分離することができた．また，性能面でも広く利用されているプログラミング言語インタプリタと同等の速度を実現できた．これにより，VM 開発に対しても VDM-SL の有効性を示すことができた．

ViennaVM の開発は継続中であり，VDM-SL の実行可能なサブセット全体に必要な機能を完成させた上で，JIT コンパイルや IR コード書き換えによる最適化などを仕様記述し実装する予定である．

参考文献

[1] A. Rigo and S. Pedroni, “PyPy’s Approach to Virtual Machine Construction”, In companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA ’06), pp. 944-953, 2006.

[2] T. Lindholm, et al, “ The Java Virtual Machine Specification”, Addison-Wesley Professional, 2014.

[3] D. Ehringer, “The dalvik virtual machine architecture”, available at `http://show.docjava.com/posterous/file/2012/12/10222640-The_Dalvik_Virtual_Machine.pdf`, 2010.

[4] T. Oda, K. Araki and P. G. Larsen, “ViennaVM: a Virtual Machine for VDM-SL development”, In proc. of the 16th Overture Workshop, pp. 39-56, 2018.

[5] J. Fitzgerald and P. G. Larsen “Modelling Systems – Practical Tools and Techniques in Software Development”, Cambridge University Press, 1998.

[6] J. Fitzgerald, P. G. Larsen and S. Sahara “VDMTools: advances in support for formal modeling in VDM”, ACM Sigplan Notices, Vol. 43, No. 2, pp. 3–11, 2008.

[7] P. G. Larsen, N. Battle, M. Ferreira, J. Fitzgerald, K. Lausdahl and M. Verhoef “The Overture Initiative – Integrating Tools for VDM”, *SIGSOFT Softw. Eng. Notes*, Vol. 32, No. 1, pp. 1–6, 2010.

[8] P. G. Larsen, K. Lausdahl, P. W. V. Tran-Jørgensen, A. Ribeiro, S. Wolff and N. Battle “Overture VDM-10 Tool Support: User Guide”, The Overture Initiative, TR-2010-02, 2010.

[9] 小田朋宏, 荒木啓二郎 “形式仕様工程の初期段階に着目した統合仕様記述環境 ViennaTalk”, コンピュータソフトウェア, 34 巻, 4 号, ppp. 4_129-4_143, 日本ソフトウェア科学会, 2017.

[10] 小田朋宏, 荒木啓二郎 “アジリティのある探索的形式仕様記述のためのテストフレームワーク”, ソフトウェアシンポジウム 2018 論文集, 2018.

[11] J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat and M. Verhoef “Validated Designs For Object-oriented Systems”, Springer, 2005.

[12] T. Kurita and Y. Nakatsugawa “The Application of VDM++ to the Development of Firmware for a Smart Card IC Chip”, *Intl. Journal of Software and Informatics*, Vol. 3, No. 2-3, pp. 343–355, 2009.

コードクローンへの欠陥混入防止に向けた欠陥混入クローンの特徴分析

野口 耕二朗
和歌山大学 システム工学部
noguchi.kojiro@g.wakayama-u.jp

大平 雅雄
和歌山大学 システム工学部
masao@sys.wakayama-u.ac.jp

要旨

大規模ソフトウェアの品質および保守効率の改善を目的として，欠陥が混入したコードクローンを検出する手法や欠陥混入クローンの特徴分析などの研究が行われている．しかしながら，コードクローンに欠陥が混入する原因を明らかにした研究は存在しておらず，クローンに対して欠陥が混入するのを未然に防ぐ方法については十分な検討がなされていない．本論文では，*RxJava* プロジェクトを対象としてコードクローンに欠陥が混入する原因を分析した結果について報告する．欠陥混入クローンのソースコードメトリクスの特徴を分析した結果，欠陥が混入していないコードクローンと比較して，*(1)* コードクローンに対する入力の数が大きい，*(2)* 規模に関するメトリクスの値が全体的に小さい，という結果が得られた．また，欠陥混入クローンの欠陥を分析した結果，コードクローンを持たない部分に混入した欠陥と比較して，*(1)* コードの不整備による欠陥が多く，*(2)* 欠陥の修正範囲が大きい，という結果が得られた．

1. はじめに

短期納品が求められるソフトウェア開発では生産性の向上を図る必要がある．生産性向上の手段の１つに，コピー&ペーストによる既存ソースコードの再利用がある．コピー&ペーストが行われる中で作成される互いに「一致」または「類似」したソースコードの断片をコードクローンと呼ぶ．

生産性の向上を目的としてコピー&ペーストを多用しすぎると，ソフトウェア中のコードクローンが増加するため，コードクローンに対して変更を加える必要が生じた場合，対象コードクローンの探索や変更のためのコストが必要となるため，コピー&ペーストの多用は結果的に生産性を低下させる場合がある．また，一部のコードクローンに変更漏れがあった場合，新たな不具合を混入することがあるため，品質を低下させる要因ととしてコードクローンは除去の対象とされることが多い．コードクローンによるソフトウェア品質の低下に対処するため，これまでにコードクローンの検出手法や [1–6] やコードクローンの追跡・除去手法 [7–12] に関する研究が盛んに行われてきた．また，コードクローンの性質に関する研究 [13,14] も盛んに行われている．

一方，Kasper らの研究 [15] は，コードクローンが保守性に悪影響を与えることが少ないことが報告されている．コードクローンすべてを注意して管理する対象とするのではなく，品質に悪影響を与えるものに限定して管理することが重要であることが明らかになったことから，近年ではソフトウェアの品質を低下させる要因となりうる欠陥混入クローンに関する研究 [16–18] がいくつか行われている．特に，Mondal らの研究 [19] では，欠陥混入クローンの 33%から欠陥が混入したままさらにコードクローンが生成されることを明らかにしている．

欠陥混入クローンから欠陥が混入したままコードクローンが生成されると，さらなる品質低下につながるだけではなく欠陥を修正する手間が増えることになる．そのため，コードクローンへの欠陥混入を防止することが重要となるが，欠陥混入の防止手段を考えるためには，コードクローンに欠陥が混入する原因をまず明らかにする必要がある．しかし，コードクローンに欠陥が混入する原因について調査した研究はほとんど存在しない．本研究では，コードクローンに欠陥が混入する原因を明らかにすることを目的として，欠陥混入クローンのソースコードおよび欠陥の特徴を分析する．

2. 欠陥混入クローン

本章では，欠陥混入クローンに関する先行研究を紹介し，本研究の位置付けを明確にする．

2.1. 先行研究

コードクローンは欠陥が混入した際の修正のコストを上げる要因と考えられる場合が多く，コードクローンに関する研究は現在までに多数行われている．特に，コードクローンの検出手法に関する研究 [1–6] やコードクローンの追跡・除去手法に関する研究 [7–12] はこれまで盛んに行われてきた．

コードクローンの検出手法や追跡・除去手法以外には，コードクローンそのものの性質を分析した研究 [13–18] がある．これまでの研究により，ソフトウェアの保守性に悪影響を与えるコードクローンは少ないこと [15] や，コードクローンの欠陥密度がコードクローンでないコード片よりも小さいこと [16] などが明らかになっており，コードクローンがソフトウェアの品質に必ずしも悪影響を与える訳ではないことが示唆されている．したがって，コードクローンすべてを注意して管理する対象とするのではなく，品質に悪影響を与えるものに限定して管理することが重要である．

欠陥が混入したコードクローン，すなわち，欠陥混入クローンに関する研究もいくつか行われている．Mondal らの研究 [17] では，より直近に変更されたコードクローンに欠陥が混入しやすいことを明らかにしている．Li らの研究 [18] では，ソースコードをプログラム依存グラフに変換し重複した欠陥が混入しているコードを探索するツール CBCD を開発している．Mondal らの研究 [19] では，コードクローンに混入した欠陥の拡散の様子を調査し，欠陥混入クローンの 33%から欠陥が混入したままさらにコードクローンが生成されることを明らかにしている．

2.2. 本研究の位置付け

先行研究では主に欠陥混入クローンそのものの性質や保守に与える影響について分析を行っている．一方本研究は，コードクローンに欠陥が混入する原因を明らかにすることを目的としている．コードクローンに欠陥が混入する原因を明らかにすることで，コードクローンに欠陥が混入するのを防止する手段を考えることができるようになる．

Mondal らの研究 [17] は，より直近に変更されたコードクローンに欠陥が混入しやすいという欠陥混入クローンの性質を明らかにしており，直近の変更を監視することで欠陥の混入を早期に除去することができるという知見を提供しているという点においては本研究と類似している．しかしながら，本研究ではすべてのリビジョンを対象に欠陥混入クローンのおよび欠陥の特徴を広範に分析することで，コードクローンに欠陥が混入するのを未然に防ぐための知見を提供しようとする点で Mondal らの研究 [17] とは目的が異なる．

3. アプローチ

本章では，欠陥混入クローンの調査を行うにあたって設定したリサーチクエスチョンについて説明し，それぞれの具体的な分析手順について述べる．

3.1. リサーチクエスチョン

本研究では，以下のリサーチクエスチョンを設定し欠陥混入クローンについての分析を行う．

RQ1 欠陥混入クローンと非欠陥混入クローンとの間でソースコードメトリクスにどのような違いがあるか

- **RQ1-1** 複雑度に関するメトリクスにどのような違いがあるか
- **RQ1-2** 規模に関するメトリクスにどのような違いがあるか

RQ2 コードクローンに混入した欠陥とコードクローンを持たない部分に混入した欠陥の特徴にどのような違いがあるか

- **RQ2-1** 欠陥の種類にどのような違いがあるか
- **RQ2-2** 修正範囲はコードクローンに混入した欠陥の方が大きいか

RQ1 を設定した理由は，欠陥混入クローンのメトリクスの特徴的な値を調査し，**コードクローンに欠陥が混入する原因となったソースコードの特徴**を明らかにするためである．そのため，RQ1-1 では，ソースコードの複雑さと欠陥の混入しやすさとの関係を調査する．調査

表 1. RQ1 で用いる複雑度および規模メトリクス

カテゴリ	メトリクス	説明
複雑度	CountInput	メソッドに対する入力（パラメータの数，使用しているグローバル変数 測定対象のメソッドが呼び出されている回数）の合計
	CountOutput	メソッドに対する出力（メソッド内で呼び出しているメソッドの数など）の合計
	Cyclomatic	メソッドの制御フローグラフにおいて，(制御グラフ内のエッジ数)− (制御グラフ内のノード数) + (連結されたコンポーネントの数) で計算される複雑度
	CyclomaticModified	Cyclomatic 複雑度を測る際に複数の分岐制御構造（java や C 言語における switch 文など） のそれぞれの分岐をカウントせず全体を 1 としてカウントする
	CyclomaticStrict	Cyclomatic 複雑度を測る際に論理演算などをそれぞれ 1 としてカウントする
	Essential	Cyclomatic 複雑度を測る際に単純な条件構造（if-else，while，do-while など）を 単一のステートメントで置き換えて計測した複雑度
	Knots	break, goto などにより制御フローの経路が交差する数
	CountPath	制御文の実行可能な経路の数
	CountPathLog	ユニークな経路を常用対数で示した値
	MaxNesting	制御構造（if，while，for，switch など）の最大ネストレベルの値
規模	CountLine	行数
	CountLineBlank	空白行数
	CountLineCode	空白行やコンパイルされない行を除いたコードの行数
	CountLineCodeDecl	宣言文の行数
	CountLineCodeExe	実行可能なコードの行数
	CountLineComment	コメントの行数
	CountSemicolon	セミコロンの数
	CountStmt	宣言文や実行可能なステートメントの数
	CountStmtDecl	宣言のステートメント数
	CountStmtExe	実行可能なステートメント数
	RatioCommentToCode	コード行に占めるコメント行の割合

するソースコードの複雑さは条件分岐構造の経路数とネストの深さ，サイクロマティック複雑度の大きさである．RQ1-2 では，ソースコードの規模と欠陥の混入しやすさとの関係を調査する．調査するソースコードの規模は行数とステートメント数，コメント数の大きさである．

RQ2 を設定した理由は，コードクローンに混入した欠陥を調査し，**コードクローンに混入しやすい欠陥の特徴**を明らかにするためである．そのため，RQ2-1 では，コードクローンにどのような種類の欠陥が混入しやすいかを調査する．RQ2-2 では，コードクローンに混入した欠陥を修正した範囲の大きさを調査する．

3.2. 分析手順

3.2.1. 分析準備

本研究ではコードクローン追跡ツール CCT (Code Clone Tracer) [12] を用いて欠陥混入クローンの抽出を行う．紙面の都合上，CCT の詳細については割愛するが，CCT は粗粒度なコードクローン検出手法 [6] をベースとして，分散型版管理システムと不具合管理システムとを関連付ける機能を付加したツールである．ブロック単位でのコードクローンの検出・追跡が行うのと同時に，不具合混入クローンの特定・追跡が行える．CCT で用いられているコードクローン検出手法で検出したコードクローンの最小トークン数は 30 である．

3.2.2. RQ1 の分析手順

RQ1 では，欠陥が混入したコードクローン（欠陥混入クローン）と欠陥が混入していないコードクローン（非欠陥混入クローン）の複雑度および規模メトリクスを計測し比較する．メトリクスについては，ソースコード解析ツール Understand[1] を用いて取得する．表 1 に，Understand で得られるメトリクスの内，本分析で用いるメトリクスとその概要を示す．

分析ではまず，対象プロジェクトの対象期間の全リビジョンから CCT を用いて欠陥混入クローンと非欠陥混入クローンを特定し，Understand によりメトリクスを取得する．以降では，メトリクスの集計や比較の手順について説明する．

コードクローンの親子関係の特定：コードクローンの親子関係とは，あるリビジョンの前後で前のリビジョンのコードクローンが後のリビジョンの同じ位置にある場合に結ばれる関係である．図 1 は，分散型版管理システム Git でのコードクローンの親子関係を表したものである．図中の Rev5 に含まれるコードクローン集合 A の親をたどることで Rev1 に含まれるコードクローン集合 A’

[1] Understand, https://www.techmatrix.co.jp/product/understand/

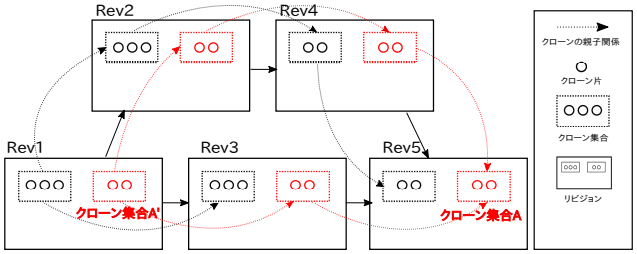


図 1. コードクローンの親子関係の例

が親であることを特定することができる．また，コードクローン集合 A からコードクローン集合 A' までのコードクローン集合の親子関係がわかるようになる．

本分析ではコードクローンに欠陥が混入してから修正されるまで欠陥混入クローンがどのリビジョンに含まれているかを知る必要がある．図中のコードクローン集合 A に欠陥が混入していて Rev5 が欠陥を修正する直前のリビジョンとする．コードクローン集合 A に混入している欠陥を欠陥 D とし，Rev1 でコードクローン集合 A' に混入したことが分かっているとする．コードクローン集合 A の親をコードクローン集合 A' までたどることで，どのリビジョンのどのコードクローン集合に欠陥 D が混入しているかがわかるようになる．

メトリクスの集計：どのリビジョンにどの欠陥混入クローンが含まれているかが分かれば，それぞれのリビジョンで欠陥混入クローンと非欠陥混入クローンとを区別することができる．Understand で計測したメトリクスが記録されている CSV ファイルにはどのコード片がどのようなメトリクスを持っているかがわかるようになっている．欠陥混入クローンであるコード片と非欠陥混入クローンであるコード片とで別々にメトリクスを集計する．

メトリクスの比較：比較対象とする非欠陥混入クローンは，欠陥混入クローンを含むリビジョンの中から欠陥混入クローン以外のコードクローンを用いる．欠陥混入クローンと非欠陥混入クローンのメトリクスの比較は，それぞれのメトリクスで欠陥混入クローンのメトリクスの値と非欠陥混入クローンのメトリクスの値に対してマンホイットニーの U 検定の両側検定を有意水準 $\alpha = 0.05$ で行い，有意差があるかどうかを確かめる．また，それぞれのメトリクスで欠陥混入クローンのメトリクスと非欠陥混入クローンの平均値にどれだけ差があるかを確かめるために Cohen's d による効果量を算出する．

3.2.3. RQ2 の分析手順

RQ2 ではまず，コードクローンに混入した欠陥の不具合票を CCT を用いて取得する．次に，コードクローンを持たない部分に混入した欠陥の不具合票をコードクローンに混入した欠陥の不具合票と同数になるようにを無作為に取得する．コードクローンと非コードクローンそれぞれに混入した欠陥に対する不具合票を調査し，欠陥の種類と修正箇所の範囲を比較する．以降では，欠陥の種類の分析と修正範囲の分析の手順について説明する．

欠陥の分析：コードクローンに混入した欠陥とコードクローンを持たない部分に混入した欠陥の不具合票を目視し，欠陥を分類する．欠陥の種類については分析対象固有の分類は行わず，ドメインに依存しない粒度で欠陥の種類を分類する．例えば，無限ループやメモリリーク，不正な値の出力などは「想定外の振る舞い」，新たに必要となる機能の追加は「不足している機能の追加」といった一般性の高い分類を行う．コードクローンに混入した欠陥とコードクローンを持たない部分に混入したコードクローンの欠陥を，種類ごとに集計した結果をそれぞれ棒グラフで示し，どのような違いがあるかを比較する．

修正範囲の分析：欠陥の修正範囲を行数とファイル数の 2 種類を用いて集計する．修正された行数やファイル数は Git の各リビジョンで記録されている．欠陥修正リビジョンから修正された行数やファイル数を抽出することで，その欠陥を修正するのに必要となった範囲（欠陥の影響範囲）がわかるようになる．

4. 分析結果

4.1. データセット

本研究では，リアクティブプログラミングを Java で使用するためのライブラリである RxJava プロジェクトを対象に分析を行う．表 2 にデータセットの概要を示す．

分析対象として RxJava を選んだ理由の 1 つは，本研究で用いるコードクローン検出ツール CCT は Java にしか対応しておらず，Java を使用しているプロジェクト

表 2. 対象プロジェクトの対象期間とリビジョン数

プロジェクト	対象期間	リビジョン数
RxJava	2014 年 8 月 30 日〜2017 年 3 月 29 日	1,703

表 3. 欠陥混入クローンなどの個数

欠陥混入クローン集合	非欠陥混入クローン集合	リビジョン数
2,478	62,654	691

である必要があるためである．2 つ目の理由は，RxJava のリビジョン数が多いからである．リビジョン数が多いことで検出することが出来るコードクローンの数も多くなるため，リビジョン数が多い RxJava は分析対象として適していると言える．3 つ目の理由は，不具合の報告が他のプロジェクトに比べて活発に行われているからである．多種多様な不具合のデータを多く集めることが出来るため RxJava は分析対象として適していると言える．

4.2. RQ1：欠陥混入クローンと非欠陥混入クローンとの間でソースコードメトリクスにどのような違いがあるか

表 3 に，特定した欠陥混入クローン集合と非欠陥混入クローン集合の個数および欠陥混入クローン集合を含むリビジョンの数を示す．非欠陥混入クローンとは，欠陥混入クローンを含むリビジョンに存在する欠陥が混入していないコードクローンを指す．表 2，3 より，分析対象期間の全リビジョン数は 1,703 件あるのに対して，欠陥混入クローン集合を含む 691 件あり，約 41% (691/1703) のリビジョンのコードクローンに欠陥が混入している状態であることがわかる．なお，RxJava は GitHub[2] 上のプロジェクトであるため分散型版管理システム Git を用いた開発を行っている．3.2.2 項の図 2 に示したように，分析対象はメインブランチだけではなく分岐し将来的にマージされるサブブランチも含まれている点には留意されたい．複雑度と規模メトリクスを比較した結果については以降で詳細に説明する．

RQ1-1：複雑度に関するメトリクスにどのような違いがあるか

複雑度に関するメトリクスが欠陥混入クローンと非欠陥混入クローンとの間でどのように違いがあったかについて報告する．表 4 に，欠陥混入クローンと非欠陥混入クローンとの間の複雑度メトリクスの有意差検定の結果と効果量を示す．表 5 に，欠陥混入クローンと非欠陥混入クローンのそれぞれの複雑度メトリクスの中央値と平均値を示す．

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

表 4. RQ1-1 の検定結果

メトリクス	p 値	Cohen's d
CountInput	0.000	**0.312**
CountOutput	0.000	**-0.399**
Cyclomatic	0.000	-0.070
CyclomaticModified	0.000	-0.068
CyclomaticStrict	0.000	0.090
Essential	0.000	**-0.245**
Knots	0.000	**-0.255**
CountPath	0.000	-0.165
CountPathLog	0.000	**-0.298**
MaxNesting	0.000	**0.269**

表 5. 複雑度に関するメトリクスの中央値と平均値

	中央値		平均値	
メトリクス	欠陥混入	非欠陥混入	欠陥混入	非欠陥混入
CountInput	4	3	4.334	3.733
CountOutput	3	4	3.421	4.084
Cyclomatic	3	2	2.823	2.936
CyclomaticModified	3	2	2.823	2.933
CyclomaticStrict	4	2	3.206	3.047
Essential	1	1	1.409	1.769
Knots	0	1	0.953	1.514
CountPath	3	2	2.973	3.576
CountPathLog	0	0	0.187	0.327
MaxNesting	1	1	1.555	1.322

表 4 より，欠陥混入クローンと非欠陥混入クローンとの間ですべての複雑度メトリクスに有意差があることがわかる．また，CountInput, CountOutput, Essential, Knots, CountPathLog, MaxNesting の効果量が，小から中程度の値（0.2〜0.4 程度）であることがわかる．効果量の値が正である場合は欠陥混入クローンのメトリクスの平均値の方が大きく，効果量の値が負である場合は欠陥混入クローンのメトリクスの平均値の方が小さいことを示している．

CountInput, MaxNesting の 2 種類の効果量は正の値であり，欠陥混入クローンの方が平均値が大きいことがわかる．表 5 からも欠陥混入クローンの方が CountInput が大きいが読み取れる．したがって，値が大きいコードクローンには欠陥が混入しやすいと言える．Cyclomatic の効果量 (-0.070) は小さかったものの MaxNesting は小程度の正の効果量が示されていることから，ネストが深いコードクローンには欠陥が混入しやすいと言える．ただし，表 5 からはネストの数の中央値がどちらも 1 であり平均値も大きな差としては観察されていないという点には注意が必要である．

一方，CountOutput，Essential，Knots，CountPathLog の 4 種類の効果量は負の値であり，欠陥混入クローンの方が平均値が小さいことがわかる．表 5 からも欠陥混入クローンの方が CountOutput が小さいことが読み取れる．したがって，内部で呼び出しているメソッドの数が少ないコードクローンに欠陥が混入しやすいと言える．同様に，Essential から，単純な条件構造を 1 つのステートメントとした場合の複雑度が小さいコードクローンに欠陥が混入しやすいと言える．また，Knots から，break 文や goto 文が少ないコードクローンに欠陥が混入しやすいことが分かった．CountPathLog から，実行可能な経路数が小さいコードクローンに欠陥が混入しやすいことが分かった．なお，表 5 より，これら 5 種類のメトリクスについても中央値と平均値の差は僅差であるため注意が必要である．

RQ1-2：規模に関するメトリクスにどのような違いがあるか

規模に関するメトリクスが欠陥混入クローンと非欠陥混入クローンとの間でどのように違いがあったかについて報告する．表 6 に，欠陥混入クローンと非欠陥混入クローンとの間の規模メトリクスの有意差検定の結果と効果量を示す．表 7 に，欠陥混入クローンと非欠陥混入クローンのそれぞれの規模メトリクスの中央値と平均値を示す．

表 6 より，欠陥混入クローンと非欠陥混入クローンとの間ですべての規模メトリクスに有意差があることがわかる．また，CountLine，CountLineCode，CountLineCodeExe，CountSemicolon，CountStmt，CountStmtExe の効果量が，小程度（0.2 程度）の負の値を示していることがわかる．欠陥混入クローンにおける上記メトリクスの平均値の方が非欠陥混入クローンに比べて小さいことを示している．したがって，行数やステートメント数などが小さいコードクローンに欠陥が混入しやすいと言える．ただし，表 7 からわかるように，RQ1-1 の複雑度メトリクスと同様に，規模メトリクスも中央値と平均値の差は僅差であるため注意が必要である．

4.3. RQ2：コードクローンに混入した欠陥とコードクローンを持たない部分に混入した欠陥の特徴にどのような違いがあるか

CCT により，コードクローンに混入した欠陥に紐づけられた不具合票は合計 86 個検出された．コードクローンに混入した欠陥と同数のコードクローンを持たない部分に混入した欠陥を無作為に抽出し，欠陥の種類と修正範囲の比較を行う．

表 6. RQ1-2 の検定結果

メトリクス	p 値	Cohen's d
CountLine	0.000	**-0.251**
CountLineBlank	0.000	0.033
CountLineCode	0.000	**-0.220**
CountLineCodeDecl	0.000	-0.065
CountLineCodeExe	0.000	**-0.235**
CountLineComment	0.000	-0.164
CountSemicolon	0.000	**-0.263**
CountStmt	0.000	**-0.223**
CountStmtDecl	0.000	-0.127
CountStmtExe	0.000	**-0.226**
RatioCommentToCode	0.000	-0.191

表 7. 規模に関するメトリクスの中央値と平均値

	中央値		平均値	
メトリクス	欠陥混入	非欠陥混入	欠陥混入	非欠陥混入
CountLine	9	11	11.849	14.168
CountLineBlank	0	0	0.688	0.645
CountLineCode	9	9	10.144	11.652
CountLineCodeDecl	2	1	1.836	1.915
CountLineCodeExe	5	6	6.277	7.469
CountLineComment	0	0	1.059	1.909
CountSemicolon	3	5	4.352	5.300
CountStmt	6	7	7.119	8.212
CountStmtDecl	2	2	1.800	1.961
CountStmtExe	4	5	5.319	6.251

RQ2-1：欠陥の種類にどのような違いがあるか

図 2 に，コードクローンに混入した欠陥とコードクローンを持たない部分に混入した欠陥の不具合票から目視し，欠陥を種類ごとに集計した結果について示す．また，表 8 に分類した欠陥の種類と概要を示す．

図 2 からいずれのコードクローンにおいても，「想定外の振る舞い」，「既存の機能の改良」，「不足している機能の追加」が欠陥の種類として多く観察された．一方で，コードクローンに混入した欠陥には，「コードの不整備」に関するものが比較的多いことがわかる．また，コードクローンを持たない部分に混入した欠陥には，「JavaDoc の部分の修正」が比較的多いことが見て取れる．しかし，「JavaDoc の部分の修正」はメトリクスに影響を与えないことに注意する必要がある．また，「不要な機能の削除」については，コードクローンを持たない部分に比べ（7 件），コードクローンに混入することが少ない（3 件）

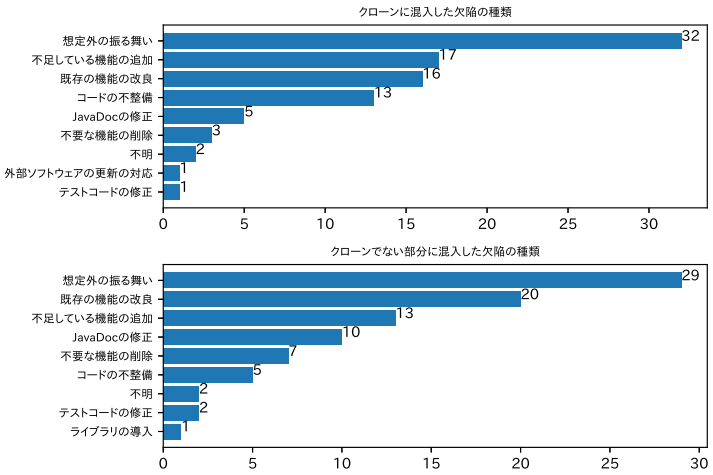


図 2. 欠陥の種類 (上:コードクローン 下:非コードクローン)

表 8. 欠陥の種類の説明

欠陥の種類	説明
想定外の振る舞い	意図していない値の出力や動作を引き起こす部分を是正する際に必要となる修正
既存の機能の改良	パフォーマンスの向上や新たに追加された機能の対応の際に必要となる修正
不足している機能の追加	他の部分の変更に対応して必要となる機能の追加や要件を満たすための機能の追加の際に行われる修正
不要な機能の削除	一致または類似した機能の存在により一方の機能を削除する際に必要となる修正
コードの不整備	修正を行った時点では必ずしも修正が必要ではなかったが，ソースコードの品質を上げるために行われた修正
JavaDoc の修正	java で記述するソフトウェアの API 仕様書を書くために必要となる JavaDoc を記述している部分で行われる修正
テストコードの修正	テストコードの部分だけ行われた修正
外部ソフトウェアの更新の対応	プロジェクトが用いているライブラリなどのバージョンアップに対応するために必要となる修正
ライブラリの導入	ライブラリの導入の対応のために必要となる修正

ことがわかる．

RQ2-2：修正範囲はコードクローンに混入した欠陥の方が大きいか

欠陥の修正範囲に関するメトリクスについて，コードクローンに混入した欠陥とコードクローンを持たない部分に混入した欠陥の間にどのような違いがあったかを報告する．表 9 に，コードクローンに混入した欠陥とコードクローンを持たない部分に混入した欠陥との間の修正範囲に関するメトリクスの有意差検定の結果と効果量を示す．表 10 に，コードクローンに混入した欠陥とコードクローンを持たない部分に混入した欠陥のそれぞれの修正範囲に関するメトリクスの中央値，平均値を示す．

表 9 より，コードクローンに混入した欠陥とコードクローンを持たない部分に混入した欠陥の修正範囲との間で，行数およびファイル数のどちらのメトリクスにも有意差があることがわかる．また，行数およびファイル数のどちらのメトリクスも効果量が正の値かつ 0.2 を超えていることが見て取れる．コードクローンに混入した欠陥の修正範囲がコードクローンを持たない部分に混入し

表 9. RQ2-2 の検定結果

メトリクス	p 値	Cohen's d
行数	0.000	0.321
ファイル数	0.000	0.418

表 10. 欠陥の修正範囲に関するメトリクスの中央値と平均値

	中央値		平均値	
メトリクス	欠陥混入	非欠陥混入	欠陥混入	非欠陥混入
行数	562	150	8038.1	631.2
ファイル数	7	3	54.1	10.5

た欠陥の修正範囲に比べて大きいことを示唆している．表10から，コードクローンを持たない部分に混入した欠陥を修正する場合と比較して，コードクローン混入した欠陥を修正する場合には中央値で約3.8倍，平均値で約12.7倍の範囲の修正が必要であることがわかる．同様に，ファイル数についてもコードクローン混入した欠陥を修正する場合には中央値で約2,3倍，平均値で約5.2倍のファイルを対象に修正を行う必要があり，修正漏れ（≒品質低下）の原因の1つとなり得ることが示唆される．

5. 考察

本章では4章でケーススタディにより得られた結果に対して考察を行う．

5.1. 欠陥混入クローンのソースコードメトリクス

5.1.1. 複雑度に関するメトリクスについて

欠陥混入クローンと非欠陥混入クローンとの間で複雑度に関するメトリクスの値を比較した結果，すべてのメトリクスで有意差があることが確認できた．非欠陥混入クローンと比較して欠陥混入クローンの方が平均値が大きかったメトリクスは，CountInput，MaxNestingであり，平均値が小さかったメトリクスはCountOutput，Knots，Essential，Knots，CountPathLogである．以下ではそれぞれのメトリクスについて考察する．

CountInputとは，メソッドで使用されているパラメータ，グローバル変数，測定対象のメソッドが呼び出されている回数の合計である．CountInputの値が大きいということは，メソッドが外部と結ばれている関係が多いことを意味する．関係が結ばれている部分で修正が行われた場合にメソッドも修正する必要がある可能性が発生するため，コードクローンに欠陥が混入しやすくなると考えられる．

MaxNestingとは，メソッドの最大ネスト数である．ネスト数が大きくなることでコードが複雑になることがあり，可読性が低下する場合がある．非欠陥混入クローンと比較して欠陥混入クローンの方がMaxNestingが大きいため，欠陥混入クローンの可読性がより低下したことを示唆しており，コードクローンに欠陥が混入しやすくなった原因の一つと考えられる．

Essential，Knots，CountPathLogという条件分岐構造の経路の多さや複雑さを意味するメトリクスが小さいクローンに欠陥が混入しやすくなった．一方，RQ2-1では，「不足している機能の追加」と「既存の機能の改良」が36.6%という多くの割合を占めてることがわかった．経路の多さや複雑さに関係なく混入する「不足している機能の追加」と「既存の機能の改良」により，欠陥混入クローンの条件分岐構造の経路数や複雑度自体は小さくなった可能性があり，今後詳細に分析する必要がある．

5.1.2. 規模に関するメトリクスについて

複雑度に関するメトリクスと同様に，欠陥混入クローンと非欠陥混入クローンとの間でメトリクスの値に有意差があることが確認された．効果量に関してはほとんどのメトリクスにおいて負の値で絶対値0.1以上となっており，それらのメトリクスの平均値は欠陥混入クローンより非欠陥混入クローンの方が大きいことが示されている．つまり，非欠陥混入クローンの方が欠陥混入クローンに比べてコードの行数やステートメントの数などが大きいことを意味している．

規模が小さいコード片からクローンを生成する際は，その時点では規模が小さく欠陥が混入していないことがわかっている中でクローンを生成する．しかし，コードが変更されていく中でそれらクローンに対しても変更が加えられ，その中で欠陥が混入するものと考えられる．一方で，規模が大きいコードからクローンを生成する際は，以降に機能の追加や改善が必要ないものである可能性がある．これらの仮説を検証するためには，今後さらに詳細に分析する必要がある．

5.2. 欠陥混入クローンに混入した欠陥

5.2.1. 欠陥の種類

クローンに混入した欠陥とクローンでない部分に混入した欠陥との間で欠陥の種類を比較した結果，クローンに混入した欠陥の種類として多く見られたのはコードの不整備である．実際にコードが不整備であることによって修正された例を図3に示す．行頭に-がついた行は，修正の際に削除された行で，+がついた行は追加された行である．削除された部分は図3のクローン以外にも存在しており，同じような修正が行われている．削除された部分を修正したり機能追加する場合に同じような修正がこのコードが存在する部分で必要となるため，保守性が低下する．保守性の低下を避けるために図3のようにク

```
@Override
public void onNext(U t) {
        frc.dispose();
        if (tus.compareAndSet(false, true)) {
-               serial.onSubscribe(EmptySubscription.INSTANCE);
-               serial.onComplete();
+               EmptySubscription.complete(serial);
        }
}
...
@Override
public void onComplete() {
        frc.dispose();
        if (tus.compareAndSet(false, true)) {
-               serial.onSubscribe(EmptySubscription.INSTANCE);
-               serial.onComplete();
+               EmptySubscription.complete(serial);
        }
}
```

図 3. コードの不整備によって修正された例

ローンを 1 つのメソッドに集約し，修正の手間を減らすような処置を行うことがある．クローンが存在する場合は図 3 のような修正が必要になる場面がクローンでない部分と比較して多いことから，クローンに混入した欠陥としてコードの不整備が多く見られたと考えられる．

5.2.2. 欠陥の修正範囲

コードクローンに混入した欠陥とコードクローンを持たない部分に混入した欠陥との間で欠陥の修正範囲の大きさを比較した結果，全てのメトリクスで有意差が見られた．また，効果量がすべて正で 0.2 を超える値となっている．つまり，コードクローンに混入した欠陥の方が修正範囲が広いことを意味している．

欠陥を修正する際にその欠陥と関係があるコードも修正する場合がある．例えば，欠陥が発生した箇所がメソッドであり，そのメソッドの戻り値の型を変更する必要があるとする．そのメソッドを呼び出している部分でその値を格納している変数があった場合，その変数の型をその戻り値の型と同じものにする必要がある．このようにして欠陥が発生した箇所が他のコードと多くの関係があるほど修正範囲も大きくなる．

クローンに混入した欠陥の修正範囲はメソッドの呼び出しの関係などに加えて，クローン集合の一部または全体を修正する可能性がある．クローンが存在することにより欠陥の修正範囲が大きくなる場合があり，クローンに混入した欠陥の修正範囲が広くなったと考えられる．

6. まとめと今後の課題

本研究ではコードコードクローンに欠陥が混入する原因を明らかにするために欠陥混入クローンの特徴の分析を行った．複雑度に関する分析から，欠陥混入クローンに対する入力数が大きく，出力数が小さいという結果が得られた．また，条件分岐構造の最大ネスト数は大きいが，経路数と経路の複雑度が小さいという結果が得られた．規模に関する分析では，ほとんどのメトリクスにおいて欠陥混入クローンの方が非欠陥混入クローンと比べて小さいという結果が得られた．クローンに混入した欠陥の種類に関する分析では，欠陥混入クローンに特有の欠陥の種類が「コードの不整備」であるという結果が得られた．欠陥の修正範囲に関する分析では，クローンに混入した欠陥の修正範囲はクローンでない部分に混入した欠陥の修正範囲に比べて大きいという結果が得られた．修正範囲が大きいことは欠陥が混入した箇所の影響範囲が大きいことを示している．クローンに欠陥を混入させないことが保守の生産性・品質向上に重要であることが改めて示された．

本研究の今後の課題は，分析対象プロジェクトを増やして知見の一般性を高めること，得られた分析結果からコードクローンに欠陥が混入することを未然に防ぐためのツールを開発することなどがある．

謝辞

本研究を実施するにあたり有益なアドバイスを頂きました久木田雄亮氏に心より感謝します．本研究の一部は，文部科学省科学研究補助金（基盤 (A): 17H00731，基盤 (C): 18K11243）による助成を受けた．

参考文献

[1] T. Kamiya, S. Kusumoto, and K. Inoue, “Ccfinder: A multilinguistic token-based code clone detection system for large scale source code,” IEEE Transactions on Software Engineering (TSE), vol.28, no.7, pp.654–670, 2002.

[2] I.D. Baxter, A. Yahin, L. Moura, M. Sant’Anna, and L. Bier, “Clone detection using abstract syntax trees,” Proceedings of the International Conference on Software Maintenance(ICSM1998), pp.368–377, 1998.

[3] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, “Deckard: Scalable and accurate tree-based detection of code clones,” ICSE ’07, pp.96–105, 2007.

[4] J. Krinke, “Identifying similar code with program dependence graphs,” WCRE ’01, pp.301–309, 2001.

[5] C.K. Roy and J.R. Cordy, “Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization,” ICPC ’08, pp.172–181, 2008.

[6] 堀田圭佑，楊 嘉晨，肥後芳樹，楠本真二，“粗粒度なコードクローン検出手法の精度に関する調査,” 情報処理学会論文誌，vol.56，no.2，pp.580–592，2015．

[7] 肥後芳樹，植田泰士，神谷年洋，楠本真二，井上克郎，“コードクローン解析に基づくリファクタリングの試み,” 情報処理学会論文誌，vol.45，no.5，pp.1357–1366，2004．

[8] N. Yoshida, Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue, “On refactoring support based on code clone dependency relation,” METRICS ’05, pp.10pp.–16, 2005.

[9] 肥後芳樹，吉田則裕，“コードクローンを対象としたリファクタリング,” コンピュータ ソフトウェア，vol.28，no.4，pp.4_43–4_56，2011．

[10] E. Duala-Ekoko and M.P. Robillard, “Clone region descriptors: Representing and tracking duplication in source code,” ACM Transactions on Software Engineering and Methodology (TOSEM), vol.20, no.1, pp.3:1–3:31, 2010.

[11] Y. Higo, K. Hotta, and S. Kusumoto, “Enhancement of crd-based clone tracking,” IWPSE ’13, pp.28–37, 2013.

[12] 久木田雄亮，大平雅雄，“コードクローン変更過程における開発者のインタラクションとソフトウェア品質の関係,” ソフトウェア・シンポジウム 2017 in 宮崎，pp.120–129，2017．

[13] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, “An empirical study of code clone genealogies,” ESEC/FSE-13, pp.187–196, 2005.

[14] V. Saini, H. Sajnani, and C. Lopes, “Comparing quality metrics for cloned and non cloned java methods: A large scale empirical study,” ICSME ’16, pp.256–266, 2016.

[15] C.J. Kapser and M.W. Godfrey, “”cloning considered harmful” considered harmful: Patterns of cloning in software,” Empirical Software Enggineering, vol.13, no.6, pp.645–692, 2008.

[16] H. Sajnani, V. Saini, and C.V. Lopes, “A comparative study of bug patterns in java cloned and non-cloned code,” SCAM ’14, pp.21–30, 2014.

[17] M. Mondal, C.K. Roy, and K.A. Schneider, “Identifying code clones having high possibilities of containing bugs,” ICPC ’17, pp.99–109, 2017.

[18] J. Li and M.D. Ernst, “Cbcd: Cloned buggy code detector,” ICSE ’12, pp.310–320, 2012.

[19] M. Mondal, C.K. Roy, and K.A. Schneider, “Bug propagation through code cloning: An empirical study,” ICSME ’17, pp.227–237, 2017.

組み込みソフトウェアにおけるコードクローン出現に関する考察

若林 奎人
京都工芸繊維大学 大学院工芸科学研究科
博士前期課程 情報工学専攻
k-wakabayashi@se.is.kit.ac.jp

水野 修
京都工芸繊維大学 情報工学・人間科学系
o-mizuno@kit.ac.jp

要旨

近年組み込みソフトウェアにおいて，個人でハードウェア部品を安価に入手できることや部品の多様化に伴い，ソフトウェア中にはコードクローンが発生することがある．

組み込み開発プラットフォームの 1 つである *Arduino* は，特にハードウェア部品が安価であり，また，ハードウェア，ソフトウェア共にオープンソースであるため，派生品や互換機も数多く存在し，環境下には多くのコードクローンが発生していると考えられる．コードクローンはソフトウェアの保守を困難にしている要因の 1 つであると指摘されている．

そこで，*GitHub* から *watch* 数の多い *Arduino* のプロジェクトを *170* 個取得し，コードクローン検出ツールである *CCFinder* に入力することで，コードクローンの位置を把握し，他のプロジェクトにおけるコードクローンの数と比較することで *Arduino* プロジェクトにおけるコードクローンの発生率について考察する．さらに組み込みソフトウェア特有のコードクローンについて考察することで，組み込みソフトウェアにおけるコードクローンに関する知見を得る．

CCFinder を用いた結果，*Arduino* プロジェクトには他のプロジェクトよりもコードクローンが多く発生していることがわかった．また，発生率の高いコードクローンの種類や，組み込みソフトウェア特有と思われるコードクローンを発見した．

1 はじめに

ソフトウェアの果たす社会的な役割は大きく，ソフトウェアの品質を高く保つことはソフトウェア工学における重要な目標である．また，ソフトウェア品質を高く保つためには，ソフトウェアの保守性を高める必要がある．しかし，1950 年代にソフトウェアが生まれて以来，ソフトウェアの保守性を高めるのは困難な課題の 1 つである．

ソフトウェアの保守性を下げる要因の 1 つにコードクローンがある．コードクローンは主にソースコードの再利用によって生まれる，ソースコードの内容が同じ構造になっているものを指す．そのため，不具合を持つソースコードのクローンが他のソフトウェアに含まれる場合，その不具合全てを発見し，修正することは困難になる．特に，組み込みソフトウェアにおいては，個人でのソフトウェア開発者も多いことから，コードクローンが幅広く発生していると考えられる．中でも Arduino [1] プロジェクトはハードウェア，ソフトウェアともにオープンソースであり，一般に広く再利用されやすい構造であるため，組み込みソフトウェアの中でもコードクローンが発生しやすいと考えられる．

そこで，コードクローン検出ツールである CCFinder [2] を Arduino プロジェクトに適用することで，コードクローンの発生状況を調査する．

本研究では，Arduino プロジェクトと組み込みソフトウェア以外のソフトウェアにおけるコードクローンの発生率を比較し，Arduino プロジェクトの方がコードクローンの発生率が高いことを確認した．また， Arduino プロジェクトにおいて，発生率の高いコードクローンについて考察し，その中から組み込みソフトウェア特有と思われるものを発見した．

本報告書の構成を以下に示す．2 章では本研究を行う目的について述べる．3 章では本実験に必要な前提事項や対象となるプロジェクトについて述べる．4 章では実験の手順について説明する．5 章では実験結果を報告する．6 章では実験結果の考察を行う．7 章では本研究のまとめと今後の課題について述べる．

2 研究の目的

本研究の目的は，CCFinder を Arduino プロジェクトに適用し，出力としてコードクローンのメトリクスを得た上で，Arduino 環境に存在するコードクローンについて考察することである．

この目的のため，以下の研究設問を設定し，実験結果について考察する．本研究における研究設問を以下に示す．

RQ1 Arduino 環境においてどの程度の割合でコードクローンが存在するか．

RQ2 ソースコードのどのような部分にコードクローンが存在するか．

RQ3 組み込みソフトウェア特有のコードクローンは存在するか．

3 準備

3.1 Arduino

Arduino とは，安価で入手できるオープンソースのプラットフォームである．Arduino の基板は，光センサーやスイッチ，音センサーから入力を受け付け，モーターや LED などへの出力を可能とする．Arduino 基板を動作させるには，C++言語を拡張した Arduino 言語と Arduino 用統合開発環境を用いる必要がある．

Arduino はイタリアのイベーラインタラクションデザイン大学で誕生した，エレクトロニクスやプログラミングの知識のない学生でも利用できるように開発されたツールである．幅広いコミュニティで利用されるようになると，シンプルな 8 ビット基板から，ウェアラブル機器，3D プリント，組み込み環境など，新しい形に変化していった．全ての Arduino 基板はオープンソースであり，個人のニーズに合わせて開発できる．ソフトウェアも同様にオープンソースであり，世界中のユーザーの知見によって拡大し続けている．

数年に渡り，Arduino は日用品から工業用まで様々なプロジェクトに応用されている．学生からプログラマー，アーティスト，専門家など，世界中の人々が Arduino を利用しており，知見を寄せ合っている．Arduino はそのシンプルさによって数多くのアプリケーションに利用されている．また，その統合開発環境である Arduino IDE は Mac，Windows，Linux 上で動作させることができる．

本研究では Arduino プロジェクトにおけるコードクローンの分析を行う．Arduino には派生プロジェクトが多く存在するため，コードクローンの影響が強いと予想される．

3.2 GitHub

GitHub [3] とは，開発プラットフォームであり，オープンソースプロジェクトやビジネスユースまで，GitHub 上にソースコードをホスティングすることで他の開発者からレビューを得ることや，プロジェクトを管理しながらソフトウェアの開発を行うことができる．

バージョン管理は Git によって行われる．Git は分散型バージョン管理システムの一つであり，開発参加者 1 人 1 人がリポジトリのコピーを保有できる．

開発者は他者にレビューをリクエストでき，他者は開発者にプルリクエストをしてコードの変更を提案できる．GitHub にはフォーク機能があり，他者のリポジトリをコピーし変更を加えることができる．変更が加えられたら，コピー元の開発者はマージすることでその変更を自分のコードに導入できる．また，変更内容の差分を確認できるので，他者はレビューを迅速に行うことができる．

GitHub が誕生する以前は，オープンソースのプロジェクトに貢献するために，そのプロジェクトのソースコードを手作業でコピーし，変更をローカルに加え，パッチを元の開発者にメールで送る，という手順を踏む必要があった．しかし現在では，全て GitHub のプラットフォーム上で行うことができる．

3.3 Google Big Query

Google Big Query[1] [4] とは，Google が提供するビッグデータ解析ツールである．データベースの操作には SQL

[1] https://cloud.google.com/bigquery

文を使用でき，GitHub のリポジトリのデータベースが存在する．

3.4 コードクローン

コードクローンとは，複数のソースファイル中で類似したコード部分のことである [4]．コードクローンは，コピーアンドペーストなど，コードの再使用によって生まれる．また，改変が繰り返された生存時間の長いプロジェクトにもしばしばコードクローンが発生する．さらに，コードをコピーした後，僅かに変化を加えたものもコードクローンとなり得る．

コードクローンはソフトウェアの保守性を下げる．例えば，複製され，変化を加えられたコードクローンを複数持つソフトウェアが存在したとする．複数あるコードクローンの内，どれか 1 つに不具合が見つかったとき，エンジニアは全てのコードクローンの不具合を除去しなければならない．複数のエンジニアが開発に携わる，複雑で大規模なソフトウェアであった場合，不具合を除去するのはさらに困難になる．

もしコードクローンの存在を事前に知っており，メンテナンスをしていれば，不具合の除去は比較的簡単になるが，コードクローンの情報を保持し続けるのは困難でコストがかかる．

コードクローンは，シンボル数ではなく文字数が最大になるように定義される．例えば，

a x y z b x y z c x y d

という文字列があった場合，コードクローンは，xyz，xy，yz の 3 種類があるが，このとき文字数が最大のものを選ぶので，コードクローンは xyz となる．

次に，コードクローンを含むソースコードの例を，ソースコード 1 に示す．クラス名が異なるだけで，同じ処理をしている．

ここで，MultiButtonUI と MultiColorChooserUI の 2 つのクラスは，クラス名が異なるだけで同じ処理をしているため，コードクローンであると言える．

ソースコード 1. コードクローン例

```
public class MultiButtonUI extends ButtonUI {
  public static ComponentUI createUI(JComponent a) {
    ComponentUI mui = new MultiButtonUI();
    return MultiLookAndFeel.createUIs(mui,
                        ((MultiButtonUI) mui).uis,a);
  }
}

public class MultiColorChooserUI extends ColorChooserUI {
  public static ComponentUI createUI(JComponent a) {
    ComponentUI mui = new MultiColorChooserUI();
    return MultiLookAndFeel.createUIs
              (mui,((MultiColorChooserUI) mui).uis,a);
  }
}
```

3.5 CCFinder

CCFinder とは，神谷らによって開発されたコードクローン検出ツールである [2]．工業用の大規模なソフトウェアシステムのような，複数のエンジニアが数十年に渡りメンテナンスを続けているプログラムに含まれるコードクローンを効率的に検出するツールが存在しなかったことが CCFinder 開発の動機となっている．

CCFinder の特徴は下記のようになっている．

- メモリと計算能力があれば，100 万行を超える工業用ソフトウェアシステムのソースコードにも適用できる．
- 大規模なシステムで大量のコードクローンが検出された場合，役に立つ情報を持つコードクローンのみを選ぶ．例えば，変数の初期化などは役に立たないものとして選出されない．
- 完全に同じコードだけでなく，変数名が異なったものなど，僅かに異なったコードからもコードクローンを検出する．
- 数種類のプログラミング言語に対応している．

CCFinder は接尾辞木を用いて文字列マッチングを行っており，行マッチングよりも計算量が大きい．しかし，いくつかの最適化を行い，大規模なソフトウェアにも実践的に使えるように設計されている．また，数種類のメトリクス値が計算され，コードクローンに関する情報を得ることができる．

各メトリクスの説明を表 1 に示す．

表 1. CCFinder が算出するメトリクス

メトリクス名	略称	内容
Number of File	FILE	入力するファイル数を表す.
Length	LEN	コードクローンの文字数を表す.
Line of Code	LOC	入力されたソースファイル群の合計行数を表す.
Source Line of Code	SLOC	LOC から空行とコメント行の数を引いたもの.
Clone Line of Code	CLOC	コードクローンの合計行数を表す.
Population of a Clone	POP	コードクローンの出現回数を表す.
Coverage of Clone % LOC	CVR %LOC	合計行数のうちのコードクローンの行数の割合を表す.

4 実験

本章では実験する対象と内容について説明する．本研究では，多くの Arduino モジュールのライブラリが GitHub のリポジトリとして存在することと，実験対象となるプロジェクト数が多いことから，Google Big Query を用いて GitHub から Arduino 用ライブラリとそのライブラリを使用しているプロジェクトを取得する．その後，取得した Arduino プロジェクトに CCFinder を適用し，出力されるコードクローンに関するメトリクスを観測することで研究設問に答える．

4.1 実験の準備

実験に使用するツールと実験対象データの取得方法について説明する．

まず，実験でコードクローンを検出するためのツールである CCFinder[2] を入手する．公式の最新バージョンである CCFinder は 2010 年で開発が止まっており，当時の実行環境 (Win32，Ubuntu9.1) を用意することが困難であるため，GitHub 上で第 3 者によってフォークされたものを使用した．次に，実験対象データである Arduino プロジェクトを Google Big Query を用いて GitHub から入手する．2016 年 1 月から 5 月の間でウォッチ数が 10 以上のプロジェクトを集めたところ，170 個のプロジェクトを入手した．このとき使用したクエリを 2 に示す．さらに，Java のビルドツールである Apache Ant (v1.10.5) を Apache ソフトウェア財団のホームページ[3] [5] からダウンロードする．

ソースコード 2. Arduino プロジェクトを取得するクエリ

```
SELECT
  repo_name,
  watch_count
FROM
[bigquery-public-data:GitHub_repos.sample_repos]
WHERE
repo_name IN(
  SELECT sample_repo_name
  FROM [bigquery-public-data:GitHub_repos.sample_contents]
  WHERE content LIKE '%#include "Arduino.h"%'
      OR sample_repo_name LIKE '%Arduino%'
                  OR sample_path LIKE '%.ino%'
)
ORDER BY watch_count DESC
LIMIT 200;
```

4.2 実験内容

各研究設問に対する実験内容について説明する．

4.2.1 RQ1

設問内容:Arduino 環境においてどの程度の割合でコードクローンが存在するか.

GitHub から入手した 170 個の Arduino プロジェクトそれぞれに CCFinder を適用し，CVR % LOC を取得し，Ekram ら [10] の実験結果から，テキストエディタである Gedit，Glimmer とウィンドウマネージャである icewm の CVR % LOC の値と比較する．

次に，コンパイル機能を持つ ESP8266[4] [6] の Arduino ライブラリと，Java のコンパイラである Apache Ant に対しそれぞれ CCFinder を適用することで，同じ機能を持つプロジェクト間におけるコードクローンの検出結果を比較する．

[2] https://github.com/radekg1000/ccfinderx

[3] http://www.apache.org

[4] https://github.com/esp8266/Arduino.git

4.2.2 RQ2

設問内容:ソースコードのどのような部分にコードクローンが存在するか.

RQ1 で得た 170 個の Arduino プロジェクト全体に CCFinder を適用し，ソフトウェアへの影響が大きいコードクローンを選ぶため，CLOC が 500 行以上で POP が 10 回以上のコードクローンを集め，ソースコードのどのような部分にコードクローンが発生しているかを調べる.

4.2.3 RQ3

設問内容:組み込みソフトウェア特有のコードクローンは存在するか.

RQ2 で得たコードクローンのソースファイルを観察することで，そのコードクローンを含むプロジェクトの特徴を調べ，組み込みソフトウェア開発経験者の手で分類を行い，組み込みソフトウェア特有のコードクローンの存在について調べる.

5 結果

各研究設問に対する実験結果を示す.

5.1 RQ1

設問内容:Arduino 環境においてどの程度の割合でコードクローンが存在するか.

170 個の Arduino プロジェクトにおけるコードクローンに関する情報の平均値と，Wi-Fi モジュールである ESP8266 の Arduino ライブラリ，Java のビルドツールである Apache Ant に含まれるコードクローンに関する情報と，Ekram ら [10] の実験結果から，Gedit，Glimmer，icewm における CVR % LOC を表 2、表 3 に示す．また，170 個の Arduino プロジェクトそれぞれに CCFinder を適用し算出した各プロジェクトの CVR % LOC を図 1 に示す.

表 2 より，Arduino プロジェクトの CVR % LOC の平均値は Gedit，Glimmer，icewm より大きいことがわかる．また，ESP8266 Library と Apache Ant の CVR の値を比較すると，ESP8266 Library の方が大きいことがわかった.

表 2. 各プロジェクトのコードクローン情報 1

	FILE	LOC	CVR % LOC
Arduino Projects	152	38,795	21.5
Gedit [10]	121	46,877	1.0
Glimmer [10]	124	44,339	2.6
icewm [10]	200	52,689	0.1

表 3. 各プロジェクトのコードクローン情報 2

	FILE	LOC	CVR % LOC
ESP8266 Library	1,464	316,462	37.8
Apache Ant	1,272	272,359	26.6

5.2 RQ2

設問内容:ソースコードのどのような部分にコードクローンが存在するか.

4.2 節で説明した条件に基づきコードクローンを選出したところ，15 種類のコードクローンを得ることができた．また，170 個の Arduino プロジェクト全体の CVR % LOC は 62.3%であった．4.2 節で説明した条件に基つき選出したコードクローンを表 4 に示す．また，SIMD 命令の羅列によるコードクローンとコアの種類によるコードクローンの例をソースコード 3 と 4 に示す.

- **組み込み関数による SIMD 命令の羅列** 組み込み関数として使用されている SIMD 命令がコードクローンとして検出された．各コードクローンの違いは，演算子や型の違いである．15 種類のコードクローンの合計行数のうち 22.9%を占めている.
- **レジスタ処理** ネットワークのパラメータを得るための関数を再利用しているものがコードクローンとして検出された．各コードクローンの違いは，対象となっているレジスタ名の違いである．15 種類のコードクローンの合計行数のうち 5.0%を占めている.
- **コンパイラによる構文解析** RTOS プロジェクト[5][7] のビルドツールによる構文解析のための関数がコードクローンとして検出されている．各コードクローンの違いは，関数名の違いである．15 種類のコードクローンの合計行数のうち 2.7%を占めている.

[5]https://github.com/TrampolineRTOS/trampoline

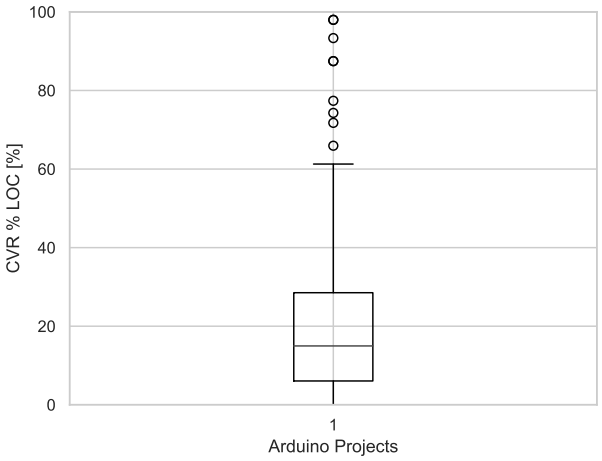


図 1. 170 個の Arduino プロジェクトの CVR % LOC

- **ツールで自動生成されたコード** 基板に非対応のモジュールを接続するための変換アダプタのプロジェクト[6]に含まれているものがコードクローンとして検出された．ソースコードのコメントから，自動生成されたものであることがわかった．各コードクローンの違いは，初期化するパラメータの違いである．15 種類のコードクローンの合計行数のうち 9.0% を占めている．
- **コアの種類によるクローン** SAM マイクロコントローラー [9] や AVR マイクロコントローラー [10] など，異なるマイクロコントローラーに対し，同じライブラリを使っているものがクローンとして検出された．全く同じライブラリを使用しているので，コードクローンの間に違いはない．15 種類のコードクローンの合計行数のうち 7.6%を占めている．

[6]https://github.com/Microsoft/Windows-iotcore-samples

表 4. CVR の大きいコードクローン情報

クローンの種類	LEN	POP
組み込み関数による SIMD 命令の羅列	2,626	46
レジスタ処理	1,511	20
コンパイラによる構文解析	1,081	11
ツールで自動生成されたコード	3,184	15
コアの種類によるクローン	4,037	10

ソースコード 3. SIMD 命令の羅列によるコードクローン例

```
__attribute__( ( always_inline ) ) static __INLINE
            uint32_t __SADD8(uint32_t op1, uint32_t op2)
{
  uint32_t result;

  __ASM volatile ("sadd8 %0, %1, %2" : "=r" (result) :
                                  "r" (op1), "r" (op2) );
  return(result);
}

__attribute__( ( always_inline ) ) static __INLINE
            uint32_t __QADD8(uint32_t op1, uint32_t op2)
{
  uint32_t result;

  __ASM volatile ("qadd8 %0, %1, %2" : "=r" (result) :
                                  "r" (op1), "r" (op2) );
  return(result);
}
```

5.3 RQ3

設問内容:組み込みソフトウェア特有のコードクローンは存在するか.

RQ2 への実験結果で得たコードクローンのうち，組み込みソフトウェア特有であるものの情報を表 5 に示す．

また，各コードクローンの例について説明する．

RQ2 で選出した 15 種類のコードクローンの内，組み込みソフトウェア特有であると思われるものは 12 種類あった．また，15 種類のコードクローンの合計行数の内，組み込みソフトウェア特有のコードクローンの割合は 40.0%であった．

- **コアの種類** 5.2 節で説明した通りである．コアモジュールの種類が異なるが，同じソースコードを用いているので，組み込みソフトウェア特有のクローンであると判断した．
- **ディスプレイの種類** 異なる種類のディスプレイに対し同じ制御ライブラリを使用し，コードクローン

表 5. 組み込みソフトウェア特有のコードクローン情報

	LEN	POP
コアの種類	4,037	10
ディスプレイの種類	2,205	13
ファームウェアの動作モードの種類	3,115	10

として検出され，組み込みソフトウェア特有のコードクローンであると判断した．15 種類のコードクローンの合計行数のうち 5.4%を占めている．

- **ファームウェアの動作モードの種類** ファームウェアの動作モードによって異なる制御ライブラリが用意されているが，ソースコードの内容が等しいためコードクローンとして検出され，組み込みソフトウェア特有のコードクローンと判断した．15 種類のコードクローンの合計行数のうち 5.9%を占めている．

ソースコード 4. コアの種類によるコードクローン例

```
String::String(const char *cstr)
{
  init();
  if (cstr) copy(cstr, strlen(cstr));
}

String::String(const String & value)
{
  init();
  *this = value;
}

String::String(const __FlashStringHelper *pstr)
{
  init();
  *this = pstr;
}

#ifdef __GXX_EXPERIMENTAL_CXX0X__
String::String(String &&rval)
{
  init();
  move(rval);
}
String::String(StringSumHelper &&rval)
{
  init();
  move(rval);
}
```

6 考察

6.1 RQ1

設問内容:Arduino 環境においてどの程度の割合でコードクローンが存在するか.

表 2 より，テキストエディタやウィンドウマネージャよりも Arduino プロジェクトの方が CVR % LOC の値が大きい．R. Al Ekram ら [10] の実験結果から，多くのプロジェクトにおける CVR % LOC はおおよそ 10%から 15%の間となっており，Arduino プロジェクトは平均 21.5%であるので，プロジェクトに対してコードクローンの発生率が高いと言える．

以上のことから，Arduino プロジェクトには組み込み以外のソフトウェアより多くのコードクローンが発生していることがわかる．

6.2 RQ2

設問内容:ソースコードのどのような部分にコードクローンが存在するか.

組み込み関数による SIMD 命令の羅列はコードクローンとして検出された．しかしこれは，組み込み処理の最適化によるものであり，ハードウェアの多様化やオープンソースであることが原因ではないと考えた．

一方で，レジスタを直接参照する記述においてもコードクローンが多く発見されることがわかった．

同様に，5.2 節の結果と神谷ら [2] の実験結果から，ツールによって自動生成されたコードには多くのコードクローンが含まれることがわかる．原因として，自動生成ツールは同じ処理のコードを，使用する変数名のみを変更して生成するためであると考えられる．

以上のことから，SIMD 命令の羅列や，レジスタの処理，自動生成によるコードなどにコードクローンが多く検出されるということがわかる．

6.3 RQ3

設問内容:組み込みソフトウェア特有のコードクローンは存在するか.

4.3 節で述べた方法に基づき，コアの種類，ディスプレイの種類，ファームウェアの動作モードの種類によって生まれるコードクローンを組み込みソフトウェア特有のコードクローンであると判定した．

また，Ekram ら [10] の研究から，RQ2 で検出した変数名のみが違うコードクローンは，テキストエディタやウィンドウマネージャーにも検出されたので，組み込みソフトウェア特有のものではないと考えられる．反対に，組み込みソフトウェア特有であると判断したコードクローンと同じ特徴をもつコードクローンは，Ekram ら [10] の研究においては検出されなかった．

構文解析文におけるコードクローンは，Apache Ant において 3.1%，ESP8266 において 1.6%確認されているため，組み込みソフトウェア特有ものではないと判断した．

本研究で発見した組み込みソフトウェア特有のコードクローンが発生する原因として，伊藤ら [11] の研究から，あるハードウェアを改良して新たにハードウェアを開発する際，開発の効率を上げるためにソースコードを再利用するということや，別種のハードウェアであっても，機能が似ていれば同じ処理のソースコードが使われるということが考えられる．

コードクローンを関数として一つにまとめた場合，呼び出しの際にオーバーヘッドが生まれてしまうため，RQ2 で検出した組み込み関数によるインライン展開を利用するなど，対処する必要がある．Joseph Caldwell ら [12] の研究から，ソースコードの量を増やさずにインライン展開を行い，関数呼び出しの最適化をする手法が考案されている．この研究の結果から，この手法を C++言語プログラムに適用し 29%のオーバーヘッドを軽減できるので，C++言語ベースである Arduino ソフトウェアにも同等の結果が期待できる．

6.4 妥当性への脅威

現在 ArduinoIDE にはライブラリマネージャの機能が存在するため，IDE によって管理されているライブラリがプロジェクトに含まれるものは CVR % LOC が下がる可能性がある．

6.5 課題

今後の課題として以下のようなことが挙げられる．

本研究ではオープンソースである Arduino のプロジェクトを実験対象としたが，他の組み込みソフトウェアではコードクローンはどのように発生しているのか調べる必要がある．

また，本研究で組み込みソフトウェアにコードクローンが発生するいくつかの原因を発見することができたが，発生した場合の対処方法と，コードクローンの発生を未然に防ぐ方法についてより深く考察する必要がある．

7 まとめ

本研究では組み込みソフトウェアにおけるコードクローンの存在について考察する足がかりとして，GitHub から取得した 170 個の Arduino プロジェクトに対し，コードクローン検出ツールである CCFinder を適用した．実験の結果，Arduino プロジェクトには組み込み以外のソフトウェアより多くのコードクローンが発生していることがわかった．発見されたコードクローンのうち，SIMD 命令の羅列や，レジスタの処理，自動生成されたコードにコードクローンが多く見られた．また，組み込みソフトウェア特有と判断できるコードクローンも見られた．今後の課題として，Arduino 以外の組み込みソフトウェアにおけるコードクローンの発生状況を調べ，コードクローンが発生した場合の対処法を考える必要がある．

参考文献

[1] Arduino, （オンライン）, 入手先〈https://www.arduino.cc〉（参照 2019-1-26）.

[2] T. Kamiya, S. Kusumoto, and K. Inoue, “Ccfinder: A multilinguistic token-based code clone detection system for large scale source code,” IEEE Transaction on Software Engineering, vol.28, no.7, pp.1–17, July 2002.

[3] Github, （オンライン）, 入手先〈https://github.com〉（参照 2019-1-26）.

[4] Google, Google BigQuery, （オンライン）, 入手先〈https://cloud.google.com/bigquery〉（参照 2019-1-26）.

[5] The Apache Software Foundation, （オンライン）, 入手先〈http://www.apache.org〉（参照 2019-1-26）.

[6] ESPRESSIF, ESP8266（オンライン）, 入手先〈https://www.espressif.com/en/products/hardware/esp8266ex/overview〉（参照 2019-1-26）.

[7] trampoline, TrampolineRTOS, TrampolineRTOS/-trampoline（オンライン）, 入手先〈https://github.com/TrampolineRTOS/trampoline〉（参照 2019-1-26）.

[8] MICROCHIP, Microchip/sam（オンライン）, 入手先〈https://www.microchip.com/design-centers/32-bit/sam-32-bit-mcus/sam-c-mcus〉（参照 2019-1-26）.

[9] MICROCHIP, Microchip/avr（オンライン）, 入手先〈https://www.microchip.com/design-centers/8-bit/avr-mcus〉（参照 2019-1-26）.

[10] R.A. Ekram, C. Kapser, R. Holt, M. Godfrey, “Cloning by accident: an empirical study of source code cloning across software systems,” International Symposium on Empirical Software Engineering, vol.10, no.7, pp.17–18, Dec. 2005.

[11] M. Ito, Y. Sakai, T. Sato, M. Sato, and T. Matsumoto, “The method of extracting the reusable software assets for improving the quality in embedded software,” Technical report, SQiP, 2006.

[12] J. Caldwell and S. Chiba, “Reducing calling convention overhead in object-oriented programming on embedded arm thumb-2 platforms,” Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, vol.52, no.12, pp.146–156, Oct. 2017.

CNN-BI システムによるプログラムの不具合発見の検証

小川 一彦
放送大学
kgg03771@nifty.com

中谷 多哉子
放送大学
tinakatani@ouj.ac.jp

要旨

ソフトウェアの品質を向上させるため，これまで多くの研究が行われてきた．従来の方法には，ソフトウェアメトリクスを用いてソフトウェアシステムの品質を評価する方法がある．本稿では，プログラムの不具合を有無を推論するために、深層学習技術の *1* つである *CNN*（*Convolutional Neural Network*）を適用する．不具合を起こすプログラムの記述には共通点があるのではないかと考えた．プログラムの記述を目視すると，関数やサブルーチンの集合体でインデントにより模様に見える．記述に共通点があるのであれば，見た目の模様にも共通点があると考えられる．プログラムの構文の文法に従い，文字に色付けすることでより特徴付けられる．プログラムの記述の特徴を，深層学習を用いて，ソースコードの構文の形から不具合を推論することを試みた．これには，ソースコードを画像化し，深層学習により不具合の推論を行った．推論結果を検証するために，不具合のある画像を推論した結果と，推論用プログラムの中で実際に発見された不具合の内容を比較した．この比較実験では，画像化したプログラムを使用して学習用データ及び推論用データを作成した．作成したデータを用いて，学習と検証および推論することで実験を行った．

1. はじめに

近年，高機能なシステムが求められるようになってきた．そのためシステムの規模は大きくなる方向にある．その半面，システム開発は低コストを求められるようになった．それにより，開発費用といったコストを抑える努力が，プロジェクト管理者に求められるようになってきた．プロジェクト開発者は，コストを抑えるために，早期にプログラムの不具合を発見し解決する必要がある．システム開発の手法，開発支援ツールは飛躍的に進歩しているが，システム開発を行うのは人間である．しかし，人間は必ずミスをする．ミスを見逃すとシステムは障害を起こし，システムの供給者及び利用者に多大な損害を発生させる．これまで以上にシステムの品質を向上させることの重要性が増している．研究者や開発者は，分析や設計，実装における人的ミスに対処するための技術を開発した．開発の初期段階で，これらの不具合を見つけて修正することが重要な目標の 1 つである．
ソフトウェアの複雑さを軽減するためのシステム構築方法が提案されている．オブジェクト指向ソフトウェア構築方法論 [1] が効果的であるとしても，信頼できるソフトウェア構築方法は，ソフトウェア内の不具合を見つけることにある．結局のところ，ソフトウェアの構築は人間の能力に依存している．実際には，テストですべてのエラーや欠陥を修正できるわけではない．さらに，未知の不具合によって，致命的な障害が発生することもある．本稿では，ソフトウェアの品質を向上させるために CNN（Convolutional Neural Network）モデルを適用する．このシステムを，CNN based bug inference system（略して CNN-BI system）と呼ぶ．

本稿の目的は，プログラムの不具合の傾向を予測するため，CNN アプローチの有効性を提示することである．CNN は深層学習の 1 つであり，画像認識と画像に基づく推論の分野で成功している．我々は，CNN-BI システムが，深層学習を用いてプログラムの不具合の特徴を抽出することを期待している．CNN によるアプローチの有効性を以下に示す．

- 不具合を引き起こすソースコードは，コードの記述に特徴を持っている．
  CNN のアプローチは，白黒の画像ではなく，文字

に色付けを行ったプログラム画像を用意することで，より特徴を捉えやすくなる．

- CNN モデルがプログラムの不具合の可能性を推測できることで，コーディングの完了・未完了を問わずプログラムの不具合を確認して修正することができる．

プログラムの不具合の傾向を推論するため，ソースコードの画像を利用し，CNN-BI システムを用いて CNN モデルで学習する．

研究の目的は、以下の問いに答えることである．

- 画像を用いた深層学習の結果，不具合の推論は可能であったか？
- 推論ができなかった不具合はどのような内容であったか？
- 不具合の精度は実際に不具合対応した結果を比較してどうであり，また精度を上げるにはどうすれば良いかを今後の課題と併せて提示する．

これらの研究課題に対する答えを得るためには，プログラムの要素を分類し，プログラムを CNN-BI システムが読む画像に変換する必要がある．プログラムのソースコードを色付きの要素を持つ画像に変換し，CNN-BI システムを用いて学習した．学習は，学習データの件数，学習時間などの学習コストを軽減するために，転移学習 [2] を適用した．推論の目的は，プログラムが不具合を起こしやすいかどうかを推論することであり，教師あり学習を CNN モデルの学習方法として選択した．

学習の後，CNN-BI システムを用いて，プログラムの不具合の傾向を推論できるかどうか評価を行った [3]．本稿では，さらに，適合率について精度を向上させるための方策を検討し，検証を行った結果を報告する．

本稿は，以下の内容で構成されている．第 2 節は，関連研究を示す．第 3 節では，研究内容について説明する．第 4 節では，考察を示す．第 5 節は，まとめを述べる．

2. 関連研究

ソフトウェアの品質を向上させる方法には，様々な方法がある．統計的手法はソフトウェアの品質を視覚化することができる．統計的方法を利用する技術が広く適用されている．定性的情報を定量的に提示することができれば，我々はプログラムの定性的特徴を定量的データと比較することができる。ソフトウェアの品質を向上させるために，ソフトウェアの品質を定量的に観測および管理することができる．

Cyclomatic complex は，1976 年に McCabe [4] によって提案された．循環的複雑度と欠陥数の間には強い相関係数がある．したがって，プログラムの循環的複雑度が大きいと測定される場合，プログラムは，循環的複雑度が小さいプログラムよりも複雑であることを意味する．定量的に提示された「比較的複雑な」プログラムは慎重に検討する必要がある．10 年後，複数のパラダイムが適用され始めた．オブジェクト指向ソフトウェアのために，Chidamber と Kemerer [5] は，結合と結合，読みやすさ，理解しやすさ，修正可能性などに基づいて複雑さを測定するための測定基準を導入した．それらのメトリクスのセットは，*CK metrics* として知られている．Zimmermann [6] はまた，欠陥予測モデルを導入した．これらの方法は、ソフトウェアシステムの品質を評価するための定量的アプローチに基づいてる．

定量的アプローチには評価基準が必要である．Lorenz と Kidd [7] は，オブジェクト指向プログラムの品質の基準として典型的な測定基準についての経験的なデータを発表した．彼らの意図は，プログラムの測定値が「通常の」範囲内にない場合，プログラムを見直さなければならないということであった．これは，メトリクスがプログラムの不具合を見つけられないことを意味する．さらに，テストはプログラムの品質を向上させるのに効果的であるが，すべての不具合を発見することはできない．プログラムの品質を向上させる上で困難な点の 1 つは，複雑なプログラムではすべての不具合が発生するわけではないということである．どの部分を集中的に検証するかを決めるために，別のアプローチを採用した．CNN アプローチは，プログラムの複雑さだけでなくプログラムの構造的特徴によっても起こる不具合を見つけることができるかもしれない。

google cat [8] 以降，深層学習の研究と応用例が増えている．ソフトウェアの品質評価に，深層学習を適用した研究がある．たとえば，森崎 [9] はソフトウェアの品質を管理するために，ソフトウェアの読みやすさの評価に深層学習を適用した．近藤ら [10] は，静的コード分析からのフィードバックの遅れに対処するため，深層学習を利用した．Yang ら [11] はまた，ソフトウェアの修正箇所を検証するため，深層学習を適用し，CNN ではなく

W-CNN を提案した．CNN-BI システムが，プログラムの不具合の可能性を予測できることを検証する．

3. 研究内容

深層学習を用いた，プログラムの不具合の推論を評価するために，推論の有効性を定性的および定量的に分析する．本節では，プログラムの不具合を推論するために，CNN が適用された事例について考察する．CNN-BI システムの学習は，OS は Windows10 HomeEdition を使用し，Google から提供されている TensorFlow と KERAS を利用し，Python で学習プログラムの開発を行う．開発環境の構成は，CPU : Core i7-6700K，MainMemory : 8GB，GPU : GeforceGTX980ti をそれぞれ使用する．

3.1. 概要

本稿で学習及び推論に使用したプログラムは，15 名の開発者によるプロジェクトであり，プロジェクトの開発期間は 1 年間であった．プロジェクトで開発されたプログラムは VisualBasic2008 により開発され，プログラム本数は 575 本であった。この 575 本より，CNN-BI システムを訓練するための学習用データ及び学習経過を検証するための検証用データ，訓練後に検証を行うための推論用データを取得する。

学習までの手順を以下に示す．

1. 学習データの収集.
   575 本のプログラムより，27 本をランダムに選択した．

2. プログラムのステートメントの各要素は，次のカテゴリに基づいて色分けを行う．
   - 命令文：青
   - 予約語：オレンジ
   - コメント：緑
   - モジュール名：明るい緑（太字）
   - グローバル変数：青（太字）
   - ローカル変数：黒
   - コントロール名：ピンク
   - ユーザ関数，サブルーチン名：赤
   - ユーザ定義名：ブラウン
   - 文字列：紫

3. 各プログラムを最大 30 行のコードでサブプログラムに分割した．135 文字以上で行を折り返す．フォントは MS ゴシックで 6.8pt，文字色が設定された場合は Bold とした．

4. プログラムの画像は，B5 サイズ横で作成する．
   学習用データとして 1490 枚の画像を取得した．言い換えれば，各画像はプログラムの断片を表している．不具合の傾向は，各画像について学習され推論される．

5. 教師あり学習の学習データは 2 つのグループに分類される．“不具合あり” と “不具合なし” は，それぞれ “BUG” と “OK” でラベル付けされる．

6. CNN-BI システムより学習用データを用いて学習した.
   CNN-BI システムの学習モデルとして，ImageNet の学習モデルである VGG 16（VGG ILSVRC 16 層）を使用した．VGG16 には，13 の畳み込み層と，3 つの全結合層を含む合計 16 の層がある．VGG16 は 1000 のカテゴリの学習済みデータセットである．
   図 1 に，VGG16 の構成を示す．畳み込み層は，CNN-BI システムの学習モデルとして再利用した．プログラムの不具合を推論するために，全連結層を学習用データで訓練した．次に訓練したモデルを使用して，不具合の有無に基づき 2 つのカテゴリーに分類を行い，推論結果を出力した。図 2 に，我々が訓練した全結合層のモデルを示す．このような学習方法を転送学習と呼ぶ．転移学習は，特定の領域から，限られた量のデータしかない別の領域に学習済みモデルを適用することに役立つ．我々は転移学習を利用し，限られた量の学習データでソースコードの不具合を推論した．

7. 同様に，検証データとして 322 枚の画像を作成した．

8. 検証データを用いて CNN-BI システムを検証した．
   学習過程の傾向を検証し，未学習および過度な過学習でないことを確認した．

学習および検証データの数を表 1 に示す．

表 1. 学習および検証データ数

Data Category	Label	#cases
Training data	OK	945
	BUG	545
Verification data	OK	213
	BUG	109

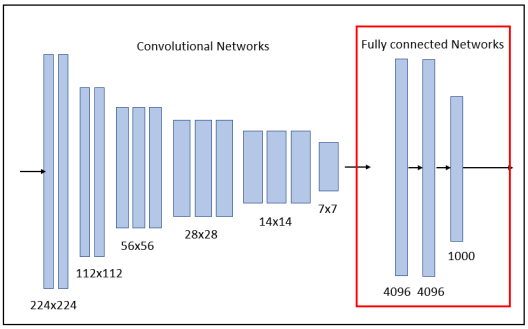


図 1. VGG16 モデルの構成.

3.2. 学習結果

学習データを用いて学習した結果，85%の精度を達成することを目標とする．学習を行う際に，2 つのカテゴリのデータ数が偏りなく学習できるよう．学習データに重み付けを行った． [12] 学習データを用いて実施した学習の過程を図 3 に示す.

左図の x 軸と y 軸は，それぞれエポックと損失を表す．同様に，右図の y 軸は学習の精度を表す．ここで，「エポック」とは、学習のサイクルである．例えば，100 エポックは，学習が 100 回繰り返されたことを意味する．

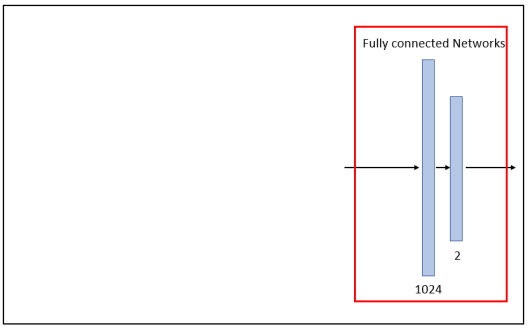


図 2. 全結合層の修正結果

図の青線は学習時の損失と精度を表す．赤線は検証時の損失と精度を表す．我々は，100 エポックの学習を行った．図 3a に示すように，損失は右下に減少している．図 3b は精度を表し，右上に上昇している．学習の結果，損失と精度はそれぞれ 19.04% と 82.97% に達した．

学習結果の精度の目標は当初，85% 以上であった．実際の学習結果の精度は 82.97% で，目標としていた精度より低い．しかし，学習経過より過度な過学習及び未学習ではないと判断し，学習結果のモデルを使用することとした．

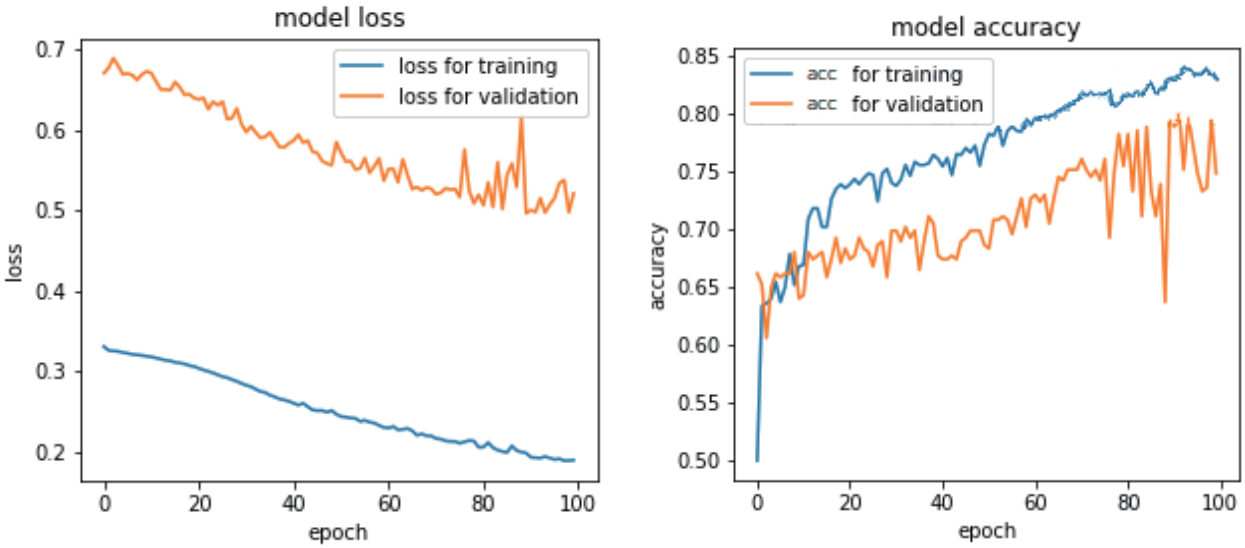
3.3 推論結果

我々は訓練された CNN-BI システムを使用し，プログラムの不具合を推論した．推論の条件は以下の通りであった．

1. CNN-BI システムの学習に使用したソースコードは，不具合の推論には使用しない．学習データと推論データは異なるプログラムを使用し，同じ画像を学習と推論で使わない．
2. 不具合の推論に使用するソースコードは，不具合を解決する前のプログラムを使用する．
3. ソースコード「OK」と「BUG」の画像の数は，学習データの画像の割合から大きく外れていない．この条件は，推論結果を学習データと比較可能にし，かつ検証可能にするためである．

推論を，692 枚の画像に対して行った．推論の結果を図 4 に示す．図 4 は、jpeg ファイルの名称が図の上部に表示される．画像は，不具合を推論した画像のサムネイルである．推論結果は，画像の下に表示される．この例では，“ Result : OK ” は不具合の可能性が低いと推論したことになる．一方，結果が “ Result : BUG ” と表示された場合，その画像の箇所は不具合が発生する可能性が高いと推論したことになる．Result の下の行は，BUG の確率と OK の確率を表している．この例では，BUG の確率は約 0.41、OK の確率は約 0.59 となる．

前述の通り，推論データは不具合を修正する前のプログラムの断片である．不具合の推論結果を評価するため，我々は不具合を解決したプログラムの修正内容を調査した．プログラムが修正されていない場合，プログラムのすべての画像のラベルは「OK」でなければならない．ま



(a) Epoch vs. Loss　　(b) Epoch vs. Accuracy

図 3. 学習経過

た，プログラムが修正されているのであれば，画像のラベルは修正箇所では「BUG」でなければならない．推論結果のラベルが，不具合を解決した結果のラベルと一致した場合，推論の結果は正しいと判断できる．

各プログラムは図 5 に X 軸の画像数と Y 軸の不具合数がプロットされている．画像の枚数はプログラムの大きさを示し，各プロットを原点から結ぶ線の傾きはプログラムの品質を示している．勾配が大きい場合，プログラムの品質は低くなる．

図には 2 つの階層データがある．プログラムを 3 つの層に分類した．層 1 に分類されたプログラムは、層 2 のプログラムよりも不具合が多く発生している．

推論結果を評価するため，我々は推論した不具合と実際に発見された不具合の傾向を検証した．検証した結果を，図 6 に示す．x 軸に画像数，y 軸に「BUG」の数を取る．

ラベル ”BUG”の結果に対応して，適合率と再現率および F 値を算出する．表 2 に結果を示す．

表 2 は左から, Pg. ソースコードに付与した連番, #Positive 推論処理で BUG と判断された画像の枚数，#True 不具合修正結果で BUG と判断された画像の枚数
#TP. 不具合修正結果と推論処理で一致した画像の枚数である．

しかし，CNN-BI システムで推論できない不具合があった．表 3 に推論できなかった不具合を示す．

表 2. 適合率・再現率・F 値

Pg.	#Positive	#True	#TP.	Precision	Recall	F
a	8	9	6	0.75	0.67	0.71
b	14	3	2	0.14	0.67	0.24
c	28	15	7	0.25	0.47	0.33
d	18	13	3	0.17	0.23	0.19
e	91	59	41	0.45	0.69	0.55

SQL 文を含む画像は，障害が発生しやすいプログラムとして推論された．一般に，構文エラーはコンパイラーまたはコーディング・ツールによって検出されるが，実行前に SQL ステートメントのエラーを検出するのは困難である．CNN-BI システムは発見することが難しい不具合を推論することができた．しかし，CNN-BI システムによる推論では指摘が難しい不具合があった．

実際に発見された不具合と推論された不具合を比較したところ，実際に発見された不具合の次の画像に ”BUG” というラベルが付けられているケースが複数あった．「不具合と判定した画像の次画像に不具合の可能性があると推論していた」と仮定したときの適合率と再現率と F 値を，表 4 に示す．

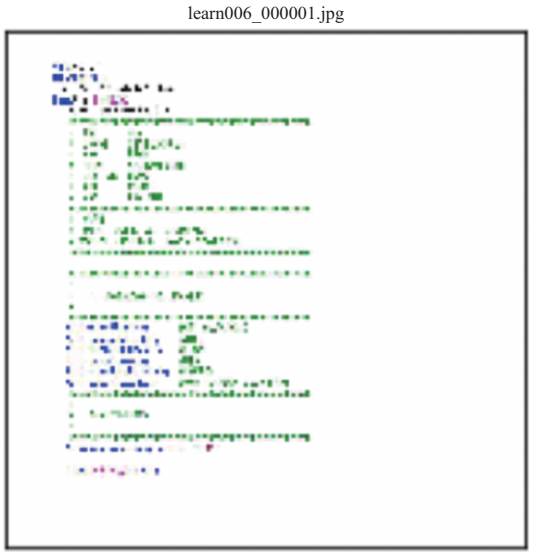


図 4. An example of the results of an inference.

取りこぼしが減少したため，再現率は増加した．しかし，適合率およびF値は減少した．

また、変数およびロジックが削除されたものと追加されたものは CNN-BI システムによる発見が困難であった.「ロジックの削除と追加以外の不具合を発見する」と仮定したときの適合率と再現率とF値を，表 5 に示す．推論修正と同様に取りこぼしが減少したため，再現率は増加した．適合率にはほぼ変化がなく、F値は再現率が増加したことに併せて増加している．変数およびロジックの削除と追加を推論しないことで，再現率が向上し，プログラムによっては，他の全ての不具合を推論できた．

推論の適合率を向上させるため，プログラムの構造に合わせて画像を作成した．プログラムの画像は，決まった行数で分けられているため，プログラムの構造を考慮していないことが原因になるのではないか考えたためである．プログラムの画像を作成する方法を，以下に示す．

1. 各プログラムを A5 横サイズで，最大 30 行のコードに分割した．
2. 関数・サブルーチン単位で切り分けを行った．
3. 30 行以上になる場合は，処理のインデントに合わせて 25～30 行の範囲で分割した．

上記の作成方法に従って画像を作成した．学習結果を

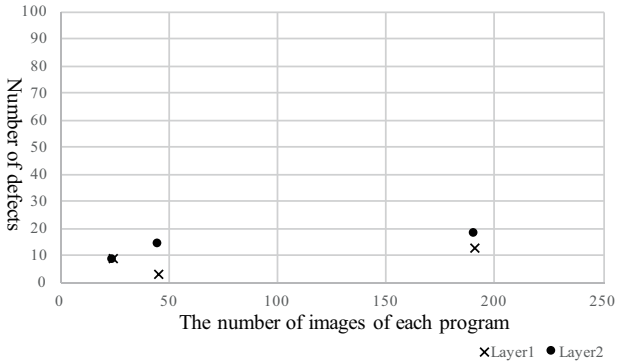


図 5. 各プログラムから作成した画像枚数とプログラムで実際に発見された不具合数の散布図.

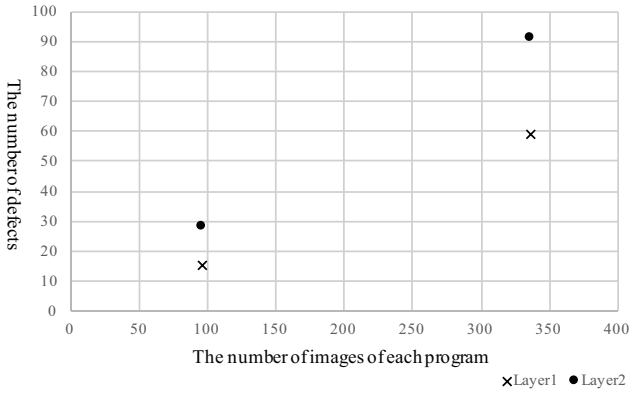


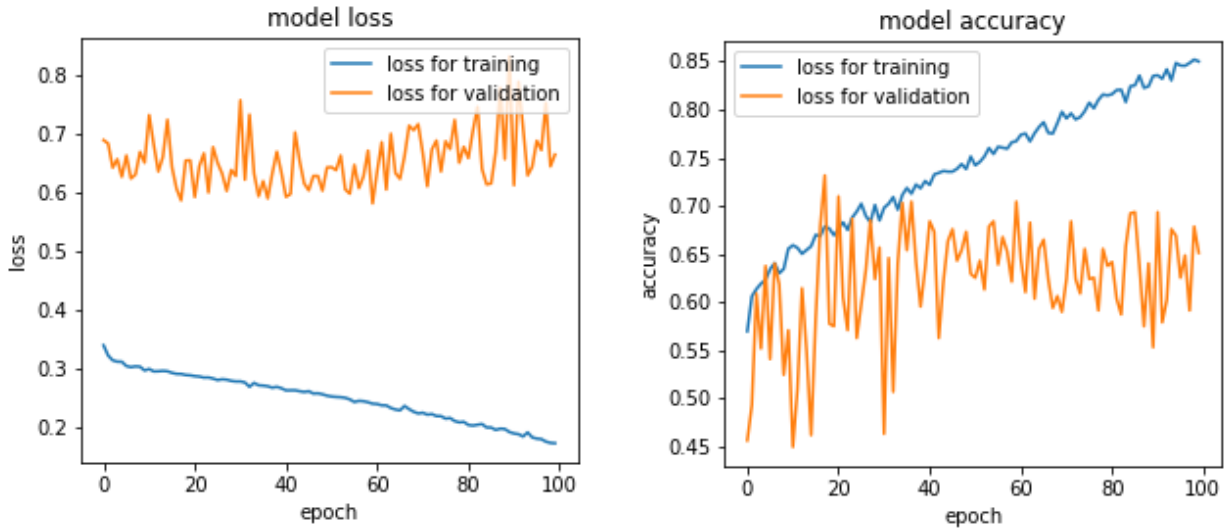
図 6. 画像数と推定された “ BUG ” の数の散布図

図 7 に示す．学習結果を確認したところ，学習の過程で検証データの損失及び精度が向上しない結果となった．これは，画像当たりのソースコード量が減ったため，特徴を捉えにくくなってしまったと考えられる．画像当りのソースコード量を減らさず，ソースコードの切り分けをする方法を考える必要がある．

4. 考察

4.1. プログラムの不具合推論は可能か

本稿では，不具合を起こしやすいプログラムを検出するため，プログラムの画像を CNN モデルに適用し，深層学習を行った．深層学習により，教師あり学習を用い



(a) Epoch vs. Loss

(b) Epoch vs. Accuracy

図 7. 画像作成方法変更後の学習経過

表 3. 推論できなかった不具合の内訳

Pg.	不具合の対応内容	件数
a	変数の削除	1
a	ロジックの削除	2
b	Ｉ Ｆ 文（条件の変数変更）の修正	1
c	呼出し先関数の修正	8
d	ロジックの追加	4
d	ロジックの修正	3
d	ロジックの削除	3
e	変数の追加・削除	13
e	ロジックの追加	1
e	ロジックの修正	2
e	呼出し先関数の修正	2

て不具合と画像の関係を学習した．不具合の学習を行った後，学習結果のモデルを使用して推論を行い，推論結果の検証を実施した．

4.1.1 推論結果

図 5 及び，図 6 は，各プログラムの画像数と実際の/推定された障害の数を用いた散布図を示している．x 軸は各プログラムのサイズを表し，y 軸はプログラムの不具合の発生しやすさを表す．散布図は，プログラムのサイズと不具合の発生しやすさとの間に，強い正の相関関係があることを示している．プログラムのサイズが大きくなるに従って障害の数が増加している．図に示すように，推論された不具合の数は実際の不具合の数よりも多かった．推論で慎重に検討する必要があるプログラムの箇所を指摘することにより，ソフトウェアの品質を向上させることができるため，問題無いと考えられる．

表 4. 推論修正後の適合率, 再現率及び F 値

Pg.	#Positive	#True	#TP.	Precision	Recall	F
a	14	9	7	0.50	0.78	0.61
b	22	3	2	0.09	0.67	0.16
c	51	15	12	0.24	0.80	0.36
d	31	13	4	0.13	0.31	0.18
e	153	59	50	0.33	0.85	0.47

4.1.2 不具合の推論結果

表 2 が示す通り，推定された不具合の再現率は最大 0.68 であった．しかし，表 3 の不具合より，推論処理に

表 5. 変数およびロジックの削除と追加を除いた場合の適合率, 再現率及び F 値

Pg.	#Positive	#True	#TP.	Precision	Recall	F
a	14	6	7	0.75	1	0.85
b	22	3	2	0.14	0.67	0.23
c	51	15	12	0.25	0.47	0.32
d	31	7	4	0.16	0.42	0.24
e	153	43	50	0.45	0.95	0.61

よって推論できなかった不具合を以下に示す.

- 条件文中の変数名が変更された.
- ロジックの追加及び削除.

CNN-BI システムが，上記の不具合を検出することは困難であった．これらの不具合は，画像に基づく推論では検出が難しいと考えられる．一方，SQL ステートメントを含む不具合は，検出ができていた．

我々の研究において最も重要な点は，我々がプログラムの品質を改善するため，この手法を適用する際，CNN-BI システムの弱点を明らかにすることである．我々は，CNN-BI システムによって，検出される不具合のあるプログラムの箇所と同様に，弱点を検討することが可能となる．

4.2. 推論の精度

適合率と再現率と F 値を，表 4 により示した．推論の結果，F 値が示している通り CNN-BI システムは高い適合率および再現率で推論することは期待できない．不具合の取りこぼしをしないため，適合率よりも高い再現率を得ることが重要である．CNN-BI システムによる推論は我々の手法によると，再現率は適合率よりも高い．適合率が再現率よりも高い場合，多くの不具合を見逃す可能性があるため，ソフトウェアの品質を向上させることは期待できない.

4.2.1 画像による不具合の推論

プログラムを画像にすることで，ソースコードの不具合を推測することは，SQL 文を文字列で記述しているプログラムの不具合を検出するのに効果的である．一方で，CNN-BI システムには弱点がある．変数やコードの追加や削除は検出が難しい.

CNN-BI システムの再現率は，すべての不具合を推論するとした場合，最大で 85%であった．我々は，不具合を起こしやすいプログラムの箇所を検出するため，画像による推論は可能であると結論付けることができた．しかし，プログラムの不具合が発生しやすい箇所を推論するための課題がある.

- CNN-BI システムの適合率の向上.

  適合率を向上させるためには，より多くの学習データが必要となる.

  また，画像の作成ルールを変更して学習を行った．関数及びサブルーチン単位で切り分けをしたが，画像 1 枚当たりの情報量を減らしてしまった．情報量が減ったことで，不具合の有る画像と無い画像で有意差も減ったために，特徴を捉えられなくなったと考えられる．画像の作成ルールは再検討が必要である．画像の作成方法を変更することには，CNN-BI システムによる推論の適合率を改善する可能性がある.

- CNN-BI システムの汎化能力の向上.
  本稿では，単一プロジェクトのプログラムソースコードを利用した．CNN-BI システムを他のプロジェクトに適用することは今後の課題である.

5. まとめ

本稿は，ソースコードを画像データに変換することによりプログラムの不具合を推論するため，学習済みデータセットを用いて転移学習を行った．我々のアプローチの鍵は，プログラムのソースコードから画像データを作成する事である．画像データから特徴を抽出するために CNN の長所を利用した.

我々は、以下の手順により，アプローチの有効性を実証した。

- 推論用プログラムに実際に見つかった不具合と比較して，推論結果を分析した.
- ソースコードの不具合について，推論結果を検証した.

検証結果より，完全ではないが，プログラムの不具合を推論することに成功した．本稿の問いに対する答えを以下に示す．

- CNN-BI システムは，プログラムの画像から不具合を起こしやすい特徴を抽出することができた．推論が完全ではないが，我々はシステムを使用して，重点的にプログラムレビューを行う箇所について指摘することができる．CNN-BI システムは，プログラムをレビューするため，どこに集中する必要があるか事前に決定するのに効果的である．
- プログラム中に存在する，いくつかの典型的な不具合の特定に成功した．また，問題のいくつかは，我々のアプローチで見つけるのが難しいことが分かった．
- 推論の精度は，我々の研究では許容可能であった．推論の精度を向上させるための課題を検討した．

本稿では、CNN を少数の学習データに適用した．推論の精度を向上させるため，学習データ（プログラム）を大量に用意し学習する必要がある．

今後，システム開発に於いて不具合の推論を行い，レビュー及びテストで特に注意してみるべき箇所を指摘することで，効率的な品質の向上に貢献したい．

参考文献

[1] B. Mayer. *Object-Oriented Software Construction, 2nd ed.* Prentice Hall, 2000.

[2] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, May 2016.

[3] K. Ogawa and T. Nakatani. Predicting fault proneness of programs with cnn. *11th International Conference on Agents and Artificial Inteligence*, 1:321–328, Feb 2019.

[4] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2:308–320, 1976.

[5] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[6] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, pages 9–15, May 2007.

[7] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics*. Prentice-Hall, 1994.

[8] Quoc V. Le, Marc ' Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, and Andrew Y. Ng. Building high-level features using large scale unsupervised learning. In *Proc. of the 29th International Conference on Machine Learning, Edinburgh*, 6 2012.

[9] Masatoshi Morisaki. Deep learning use cases and their points and tips : 6. review source code with AI. *IPSJ Magazine*, 59(11):985–988, 10 2018.

[10] Masanari Kondo, Keita Mori, Osamu Mizuno, and Eun-Hye Choi. Just-in-time defect prediction applying deep learning to source code changes (in japanese). *Journal of Information Processing Systems*, 59(4):1250–1261, apr 2018.

[11] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun. Deep learning for just-in-time defect prediction. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 17–26, Aug 2015.

[12] Nathalie Japkowicz. Learning from imbalanced data sets: A comparison of various strategies. In *AAAI Technical Report WS-00-05*. AAAI, 2000.

LSTMを用いたソースコード内の演算子推定手法

舟山 優
京都工芸繊維大学 大学院工芸科学研究科
博士前期課程 情報工学専攻
y-funayama@se.is.kit.ac.jp

水野 修
京都工芸繊維大学 情報工学・人間科学系
o-mizuno@kit.ac.jp

要旨

統合開発環境（*IDE*）には関数名や変数名の候補を表示するなどの補完機能が備わっているものが多い．このような機能は，素早くソースコードを記述したり，不具合の混入を抑制したりするなど，ソフトウェア開発の生産性を向上させている．このような生産性に関連する研究は数多く行われている．

本研究ではソースコード中の演算子に関する有用な情報をユーザに提示することを目的とする．有用な情報の例としては，演算子の補完機能や，ソースコード中の不適切な演算子の検出が考えられる．ソースコード中の不適切な演算子とは，例えば ”`if(a > 5)`” とするべきところを ”`if(a >= 5)`” としてしまった場合などである．目的達成のための第一歩として，本研究ではソースコード中のある箇所の演算子の種類が不明な場合に，その箇所に最も当てはまる演算子を推定する手法を提案した．*Java* で記述されたソースコードをトークンの並びに変換し，それをもとに欠落した演算子を推定する *LSTM* を用いた機械学習モデルを作成した．*LSTM* を用いたモデルを *3* つ作成し，それらのモデルを用いた手法と欠落した演算子をランダムに推定する手法で実験し，考察を行った．実験の結果，最大で約 *72%*の正解率を得られた．この結果から，ソースコードには欠落した演算子の特定に有用な特徴が含まれており，*LSTM* を用いることでそれらの特徴を学習できると考えられる．

1. はじめに

統合開発環境（IDE）には補完機能が備わっているものが多い．補完機能とは，文字列を入力しているときに次に連なる字句を推測して候補を提示する機能である．この機能は，開発時にソースコードを記述する速度を向上させたり誤字を減らしたりすることなどを目的として導入される．

ユーザに対して有用な情報を提示することに関して様々な研究が行われている．例えば，Yang ら [1] は，プログラムを修正した際にその修正した要素と類似したものを抽出し，次に修正する可能性がある部分を強調して表示するツール SimilarHighlight を作成した．このツールを使用することで，プログラム開発の生産性が向上したり，レビューをする際に簡単に類似の要素を見つけることができるようになった．また，山本ら [2] は，ユーザが開発作業を中断することなく，必要に応じてソースコードを再利用できる手法を提案した．過去のソフトウェアに含まれる既存のソースコードをコーパス化してデータベースを作成し，ユーザが書いている途中のソースコードの後に続くソースコードとして，過去のソフトウェアの中で確率が高かったものを提示する．

本研究では，ソースコード中の演算子に関する有用な情報をユーザに提示することを目的とする．最終的にはソースコード中の任意の箇所を推定し提示することを目指すが，その第一歩としてまずは推定の対象を演算子のみに限定し，ソースコード中のある箇所の演算子の種類が不明な場合に，その箇所に最も当てはまる演算子を推定する手法を提案する．「演算子の種類が不明なソースコード中のある箇所」を，これ以降便宜的に「空欄」と呼称する．将来的には，演算子の補完機能の開発や，ソースコード中の不適切な演算子の検出などに貢献できると考えられる．

演算子を推定する方法として，本研究では Long short-term memory（LSTM）を用いる．LSTM は自然言語の

分野などで利用されている．本研究で作成した機械学習モデルは，ソースコードの一部をトークン列へ変換したものを LSTM への入力とすることで空欄に当てはまる演算子を推定する．作成したモデルを用いて，空欄より前の部分のソースコードを入力とした場合，空欄より後の部分のソースコードを入力とした場合，その両方を入力とした場合，ソースコードを入力とせず出現頻度で重み付けされた確率に基づいてランダムに推定した場合の 4 つの実験を行い，比較した．

2. 研究の目的

本研究の目的は，ソースコードに関する有用な情報をユーザに提示するため，ソースコード中の任意の箇所を推定し提示することの第一歩として，ファイルに含まれるソースコードの情報をもとに，ソースコード中のある箇所の演算子の種類が不明とされた場合に，その箇所に最も当てはまる演算子を LSTM を用いて推定することである．

本研究では以下に示す研究設問について検証を行う．

RQ1: 空欄より前の情報から空欄に入る演算子をどの程度の精度で推定できるか．

RQ2: 空欄より後の情報から空欄に入る演算子をどの程度の精度で推定できるか．

RQ3: 空欄の前後の情報から空欄に入る演算子をどの程度の精度で推定できるか．

RQ4: LSTM に与える情報によって推定した結果に差異はあるか．

3. 準備

3.1. 演算子

本研究では Java で記述されたソースコードを対象とする．Java には数多くの演算子があるが，その全てを扱うと問題が複雑になるため，二項演算子である

`+`，`-`，`*`，`/`，`%`，`<`，`>`，`<=`，`>=`，`==`，`!=`，`&&`，`||`

の 13 種の演算子を実験対象とした．

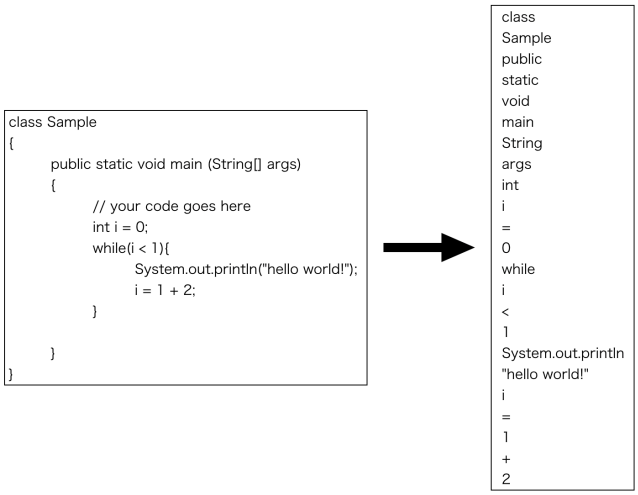


図 1. トークナイザによる変換例

3.2. トークナイズ

トークナイズ（tokenize）とは，ソースコードをトークンの並びに変換することである．トークンとはキーワードや識別子，演算子などを指す．トークナイズを行うプログラム等のことをトークナイザと呼ぶ．Java で記述されたソースコードに対して本研究で用いたトークナイザを使用した例を図 1 に示す．このトークナイザはコメントと括弧，区切り文字 ';' を無視しつつ，先頭から順にトークンを抽出する．

3.3. 深層学習

深層学習とは機械学習の一種であり，人間の脳神経回路を模倣して考案されたニューラルネットワークの層を増やし，階層を深めたアルゴリズムのことである．階層を深くすることで複雑なパターンのデータも学習することができ，モデルの表現力を向上させることができる．

3.3.1. Keras

Keras [3] とは，Python で記述されたニューラルネットワークを扱うライブラリである．TensorFlow などの上で実行できる．Keras は迅速な実験を可能にすることに重点を置いて開発されており，学習コストが低く手軽に使用できることから，事業や研究で幅広く利用されている．本研究では Keras を用いて学習モデルを作成した．

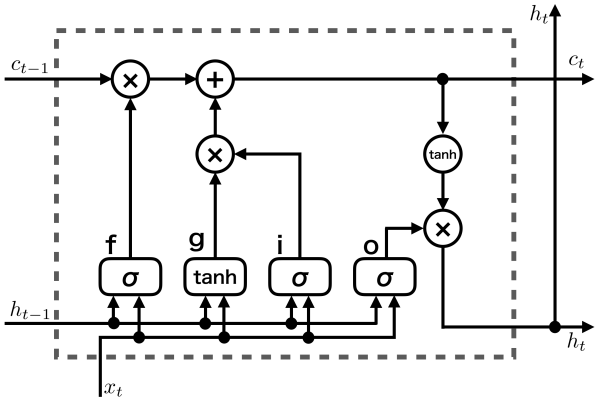


図 2. LSTM の計算グラフ

3.3.2. LSTM 層

Long short-term memory（LSTM）層は，受け取った入力から隠れ状態を計算し結果を出力する層である．再帰型ニューラルネットワーク（RNN）と同じく，出力された隠れ状態を自身への入力とすることで再帰的な計算を行う．

LSTM の中身を計算グラフを用いて表したものを図 2 に示す．この図は時刻 t のときの LSTM の状態を表しており，入力 $\boldsymbol{x}_t$ と，一つ前の時刻 $t-1$ の後述する記憶セル $\boldsymbol{c}_{t-1}$ と隠れ状態 $\boldsymbol{h}_{t-1}$ を受け取り，$\boldsymbol{c}_t$ と $\boldsymbol{h}_t$ を出力する．また，図中の'σ' はシグモイド関数を表している．

LSTM は RNN とよく似ているが，LSTM には記憶セル $\boldsymbol{c}$ と呼ばれるものがある．LSTM 層から出力された記憶セルは自分自身のみへの入力として使用される．記憶セルの計算にはゲートという機能を使用する．ゲートはデータの流れをコントロールするための機能で，0 以上 1 以下の実数で表される．0 はデータを全く流さない閉じた状態を表し，1 はデータを全て流す全開の状態を表す．

ゲートには，忘却ゲート，入力ゲート，出力ゲートの 3 種類がある．

忘却ゲート

図 2 の'f' に相当し，次の式で表される．

$$\boldsymbol{f} = \sigma(\boldsymbol{x}_t \boldsymbol{W}_x^f + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^f + \boldsymbol{b}^f) \tag{1}$$

式中の $\boldsymbol{W}_x^f$ は入力 $\boldsymbol{x}_t$ に対する重みであり，$\boldsymbol{W}_h^f$ は一つ前の隠れ状態 $\boldsymbol{h}_{t-1}$ に対する重みである．$\boldsymbol{b}^f$ はバイアスを表す．

このゲートは，一つ前の記憶 $\boldsymbol{c}_{t-1}$ から不要な情報を捨てる働きをする．

入力ゲート

図 2 の'i' に相当し，次の式で表される．

$$\boldsymbol{i} = \sigma(\boldsymbol{x}_t \boldsymbol{W}_x^i + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^i + \boldsymbol{b}^i) \tag{2}$$

LSTM には前述した忘却ゲートによる不要な情報を捨てる機能だけでなく，新しい情報を得るための機能がある（図 2 の'g'）．これは次の式で表される．

$$\boldsymbol{g} = \tanh(\boldsymbol{x}_t \boldsymbol{W}_x^g + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^g + \boldsymbol{b}^g) \tag{3}$$

この機能に対して制御を行うゲートを入力ゲートと呼ぶ．入力ゲートは追加される新しい情報にどれだけの価値があるのかを判断する．

出力ゲート

図 2 の'o' に相当し，次の式で表される．

$$\boldsymbol{o} = \tanh(\boldsymbol{x}_t \boldsymbol{W}_x^o + \boldsymbol{h}_{t-1} \boldsymbol{W}_h^o + \boldsymbol{b}^o) \tag{4}$$

隠れ状態 $\boldsymbol{h}_t$ は $\boldsymbol{c}_t$ に tanh 関数を適用することで求められるが，その際に出力ゲートは $\tanh(\boldsymbol{c}_t)$ の各要素が次の時刻の隠れ状態としてどれだけ重要であるかを調整する．

これらのゲートを用いて記憶セル $\boldsymbol{c}_t$ と隠れ状態 $\boldsymbol{h}_t$ はアダマール積（$\circ$）を用いて次の式で表される．

$$\boldsymbol{c}_t = \boldsymbol{f} \circ \boldsymbol{c}_{t-1} + \boldsymbol{g} \circ \boldsymbol{i} \tag{5}$$

$$\boldsymbol{h}_t = \boldsymbol{o} \circ \tanh(\boldsymbol{c}_t) \tag{6}$$

ゲートと記憶セルによって，RNN を使用した場合にしばしば問題となる勾配消失を抑制することができる．

LSTM では過去の情報が記憶されるため，過去のデータと現在のデータに関連性があるようなデータ，例えば時系列データを扱う際によく用いられる．文章も単語の並びに関連性があると考えると，時系列データの一種とみなせる．

3.3.3. ドロップアウト層

ドロップアウト層は，学習時にランダムにニューロンを選びそのニューロンを無視することでその先に信号が

伝達するのを止める層である．ドロップアウト層を追加することでランダムな無視が制約となり，過学習を防ぐことができる．過学習とは，学習データに対して特化した学習を行ってしまったために未知のデータに対しては正しい答えを出せないような状態のことである．過学習を防ぐということはニューラルネットワークの汎化性能を向上させるということである．

3.4. 多クラス分類

本研究はソースコード中の空欄に当てはまる演算子を推定するものであり，推定する演算子の候補は 3.1 節で述べたようにようにあらかじめ限定されている．よって，空欄に当てはまる確率が候補の中で最も高いものを選ぶ多クラス分類を行えばよい．

多クラス分類を行うニューラルネットワークでは，ニューラルネットワークが出力するスコアにソフトマックス関数を適用してスコアを確率に変換し，損失関数として交差エントロピー誤差を用いて損失を求めることが多い．ここで，損失関数とはある学習時点における学習結果が正解データとどのくらい異なっているかを示す「損失」を求める関数のことである．学習がうまく行えていると損失は徐々に小さくなり，損失の更新量が一定より小さくなると学習は終了する．

3.4.1. ソフトマックス関数

ソフトマックス関数は，ニューラルネットワークから出力されるスコアを確率に変換する関数であり，次の式で表される．

$$y_k = \frac{\exp(s_k)}{\sum_{i=1}^{n} \exp(s_i)} \tag{7}$$

式（7）はクラスが n 個あるときの k 番目のクラスの出力 y_k を求めている．

ソフトマックス関数を適用した n 個の各出力は 0 以上 1 以下の実数となり，n 個全ての出力を足し合わせると 1.0 になる．このことからソフトマックス関数の出力は確率であるとみなせる．

3.4.2. オプティマイザ

機械学習の分野において，損失関数の値をできるだけ小さくするパラメータの値を見つけることを最適化（optimization）と呼び，その手法のことをオプティマイザ（optimizer）と呼ぶ．タスクによって最適なオプティマイザは異なるが，確率的勾配降下法（SGD）や Adaptive moment estimation（Adam）[4] などがよく利用される．

表 1. 2 クラスの混同行列の例

		予測値	
		Positive	Negative
真値	Positive	TP	FN
	Negative	FP	TN

表 2. 3 クラスの混同行列の例

		予測値		
		A	B	C
真値	A	T_A	F_{BA}	F_{CA}
	B	F_{AB}	T_B	F_{CB}
	C	F_{AC}	F_{BC}	T_C

3.5. 評価尺度

3.5.1. 多クラスの混同行列

説明のため，A，B，C の 3 つのクラスを分類する問題を考える．2 クラスの場合の混同行列は表 1 のようになるが，3 クラスの場合の混同行列は表 2 のようになる．

表 2 の T_A は真値が A で予測値も A であったときの数を表し，F_{AB} は真値は B だが予測値が A であったときの数を表す．以降の説明では，表 2 に基づいて評価尺度を表す．

3.5.2. 正解率 (Accuracy)

正解率とは，正しく予測した割合であり，次の式で表される．

$$Accuracy = \frac{\sum_{i\in\{A,B,C\}} T_i}{\sum_{i\in\{A,B,C\}} T_i + \sum\sum_{i,j\in\{A,B,C\},j\neq i} F_{ij}} \tag{8}$$

正解率は予測の全体的な傾向を把握するには有用であるが，クラスの数の偏りなどに大きく影響を受ける．

3.5.3. 適合率（Precision）

クラス i に対する適合率とは，予測値が i であるもののうち真値が i であったものの割合であり，次の式で表される．

$$Precision_i = \frac{T_i}{T_i + \sum_{j \in \{A,B,C\}, j \neq i} F_{ij}} \tag{9}$$

3.5.4. 再現率（Recall）

クラス i に対する再現率とは，真値が i であるもののうち予測値が i であるものの割合であり，次の式で表される．

$$Recall_i = \frac{T_i}{T_i + \sum_{j \in \{A,B,C\}, j \neq i} F_{ji}} \tag{10}$$

3.5.5. F 値

F 値（F1 値などとも呼ばれる）とは，適合率と再現率のトレードオフに対して最適解を求めるための尺度であり，適合率と再現率の調和平均で計算される．クラス i に対する F 値は次の式で表される．

$$F_i = \frac{2 \cdot Precision_i \cdot Recall_i}{Precision_i + Recall_i} \tag{11}$$

適合率か再現率どちらか一方が大きい値を記録しても良い結果とは言えない．適合率と再現率がどちらも大きい場合に F 値も大きくなる．F 値は各クラスのデータ数に偏りがある場合にも有効な尺度である．

3.5.6. マイクロ平均

上で述べた適合率や再現率，F 値は各クラスごとに計算する方法であり，学習モデル全体の評価を行うにはそれらの平均を取ることなどが考えられる．各クラスごとの指標の平均を求める方法はマクロ平均と呼ばれる．マクロ平均はデータの偏りを考慮していない．

データの偏りを考慮した方法にマイクロ平均というものがある．マイクロ平均はクラスごとに計算するのではなく，多クラスの混同行列を 2 クラスの混同行列に変換して 2 値分類と同様に計算する方法である．表 2 を表 1 のように変換する場合は，

$$\begin{aligned}
TP &= \sum_{i \in \{A,B,C\}} T_i \\
TN_i &= \sum_{\substack{j \in \{A,B,C\} \\ j \neq i}} T_j + \sum_{\substack{j \in \{A,B,C\} \\ j \neq i}} \sum_{\substack{k \in \{A,B,C\} \\ k \neq i}} F_{jk} \\
TN &= \sum_{i \in \{A,B,C\}} TN_i \\
FP &= \sum_{i \in \{A,B,C\}} \sum_{\substack{j \in \{A,B,C\} \\ j \neq i}} F_{ij} \\
FN &= \sum_{i \in \{A,B,C\}} \sum_{\substack{j \in \{A,B,C\} \\ j \neq i}} F_{ji}
\end{aligned} \tag{12}$$

のように計算することで変換できる．式（12）で計算した結果を用いて適合率などを求めることで学習モデル全体の評価を行うことができる．FP と FN が等しくなるため，適合率と再現率は等しくなる．適合率と再現率が等しいとき，式（11）より F 値も適合率および再現率と等しくなる．また，マイクロ平均を用いると正解率も適合率や再現率，F 値と等しい値になる．

4. 提案手法

4.1. 提案手法の概要

提案手法では，ソースコードの情報を入力とする学習モデルを用いて空欄に当てはまる演算子を推定した．LSTM 層などによって構成される 3 種の学習モデルを作成した．

4.2. モデルの定義

空欄より前の部分のソースコードをトークン化したものを t_F，後の部分のソースコードをトークン化したものを t_B とする．

4.2.1. 学習モデル M_F

t_F を用いて空欄に当てはまる演算子を推定する学習モデル M_F を図 3 に示す．バッチサイズは 116 に，埋め込み層の出力の次元数は 100 に，LSTM 層の隠れ状態の数は 128 に，ドロップアウト層のドロップする割合は

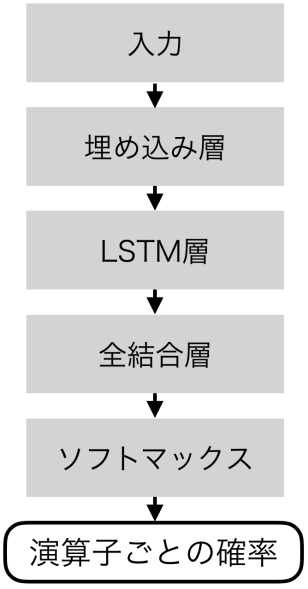


図 3. 学習モデル M_F, M_B の概要図

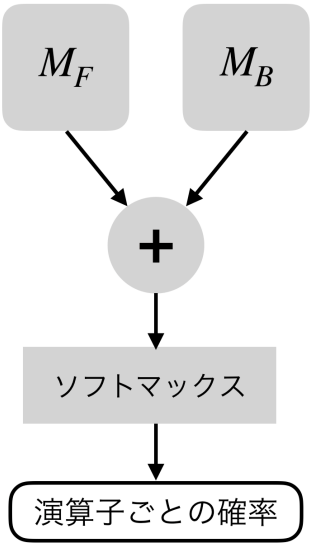


図 4. 学習モデル M_{FB} の概要図

50%に，エポック数は 10 に設定した．オプティマイザには Adam を使用した．また，活性化関数はソフトマックス関数を用い，損失関数は交差エントロピー誤差を用いた．

4.2.2. 学習モデル M_B

t_B を用いて空欄に当てはまる演算子を推定する学習モデル M_B を図 3 に示す．バッチサイズなどのパラメータは M_F と同じものを使用する．

M_B は M_F とほぼ同じであるが，M_F と異なり，入力の順を逆順に並び替える．例えば，”`a = b ?? c + d`” というソースコードの場合は，’`d`’，’`+`’，’`c`’ の順で入力される．

4.2.3. 学習モデル M_{FB}

t_F と t_B を用いて空欄に当てはまる演算子を推定する学習モデル M_{FB} を図 4 に示す．バッチサイズなどのパラメータは M_F と同じだが，エポック数は 20 とした．エポック数が異なるのは，損失が収束するまでの学習回数が異なるためである．

図 4 の左側の入力層に入力された t_F は埋め込み層により分散表現へと変換され，左側の LSTM 層に出力される．同様に，右側の入力層に入力された t_B は埋め込み層により分散表現へと変換され，右側の LSTM 層に出力される．その後，左右それぞれで M_F や M_B と同様の計算を行い，出力を加算する．加算した結果にソフトマックス関数を適用し，最終的な演算子ごとの確率から演算子の推定を行う．

5. 実験方法

5.1. データの前処理

5.1.1. 使用するソースコード

実験には Apache [5] のオープンソースソフトウェアプロジェクトである Ant, Camel, Eagle, James の全 Java ファイルのうち，実験対象の 13 種の演算子のうちいずれかを 1 つ以上含んでいるファイル 11,600 個を使用した．この 11,600 個のデータを，リポジトリを区別せずファイル名でソートし先頭から順に，学習データ 6,960 個，検証データ 2,436 個，テストデータ 2,204 個に分けて実験を行なった．学習データはモデルの学習のために用いられ，検証データは学習が 1 エポック終了する毎にモデルの性能を検証するために用いられる．テストデータは学習が終了したモデルの最終的な評価を行うために用いられる．

5.1.2. 前処理の手順

実験に使用する学習データ，検証データ，テストデータに対して行う前処理について述べる．

1. トークナイザを用いて Java ソースコードをトークンの並びに変換する．変換したものをトークンファイルと呼ぶこととする．
2. トークンファイルの中から実験対象の演算子を無作為に 1 つ選び，”??” に置き換える．”??” は空欄を意味する．この時，置き換えた演算子（つまり教師ラベル）を保存しておく．

5.1.3. 実験データに含まれる演算子

実験データに含まれる演算子を表 3 に示す.「全て」は各データのトークンファイルに含まれる全ての演算子の数を,「空欄」は実験のためトークンファイルの演算子一つを空欄に置き換えたときの置き換えられた演算子の数，つまり教師ラベルの数を表している．例えば，’+’ の行の左端の 31,024 は学習データの全トークンファイルに ’+’ が 31,024 個現れることを表している．また，その右の 2,667 は学習データのうち教師ラベルが ’+’ のものが 2,667 個あることを表している．

5.2. RQ1 の実験手順

本実験では，空欄より前のトークン t_F をモデル M_F の入力とし，空欄に当てはまる演算子を推定する．テストデータを用いて推定した結果を混合行列で表した．

5.3. RQ2 の実験手順

本実験では，空欄より後のトークン t_B をモデル M_B の入力とし，空欄に当てはまる演算子を推定する．テストデータを用いて推定した結果を混合行列で表した．

5.4. RQ3 の実験手順

本実験では，空欄より前のトークン t_F と空欄より後のトークン t_B の両方をモデル M_{FB} の入力とし，空欄に当てはまる演算子を推定する．テストデータを用いて推定した結果を混合行列で表した．

表 3. 各データの演算子の数

	学習データ		検証データ		テストデータ	
	全て	空欄	全て	空欄	全て	空欄
+	31,024	2,677	9,223	1,003	13,347	835
-	7,107	589	2,048	187	2,140	203
*	1,023	133	358	44	539	90
/	217	16	85	5	134	7
%	122	15	39	4	53	6
<	6,141	734	1,994	255	1,818	216
>	5,667	478	1,789	165	1,699	141
<=	358	36	121	13	126	9
>=	400	30	137	13	201	15
==	8,486	839	2,750	298	2,755	261
!=	10,417	947	3,487	277	3,048	289
&&	4,390	325	1,808	115	1,419	85
||	2,058	141	799	57	727	47

5.5. RQ4 の実験手順

LSTM を用いた 3 つの実験の結果と比較するため，本実験ではソースコード内の演算子出現数にもとづいて重み付けされた確率でランダムに推定するモデル M_R を作成した．確率に重み付けをする際は学習データの教師ラベルの数を元に計算した（詳細な数は表 3 の学習データの空欄列を参照）．例えば，学習データの教師ラベル 6,960 個のうち’+’ が正解のものは 2,677 個あるので，’+’ の重み付けされた確率は約 38%となる．テストデータを用いて推定した結果を混合行列で表した．その後，全モデルの推定結果を F 値を用いて比較した．

6. 実験結果

それぞれの実験の推定結果から得られた混同行列のヒートマップを図 5〜8 に示す．ヒートマップとは行列型の数字データの強弱に色をつけることで行列を見やすくしたものである．数字が大きくなると色が濃くなり，反対に数字が小さくなると色が薄くなる．本実験で作成したヒートマップは混同行列の正規化を行い色付けをしているが，表示されている数字は正規化する前のものである．True label は正解の演算子を表し，Predicted label はモデルが推定した演算子を表す．例えば，図 5

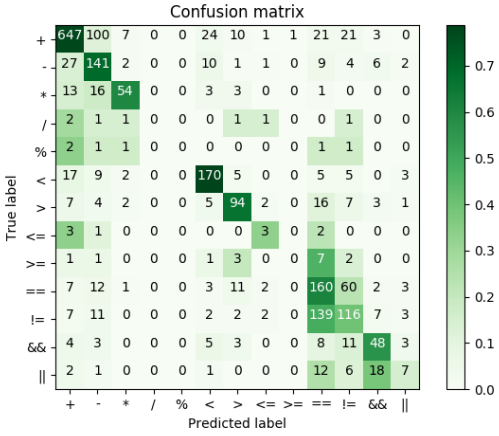


図 5. M_F を用いた実験の結果の混同行列

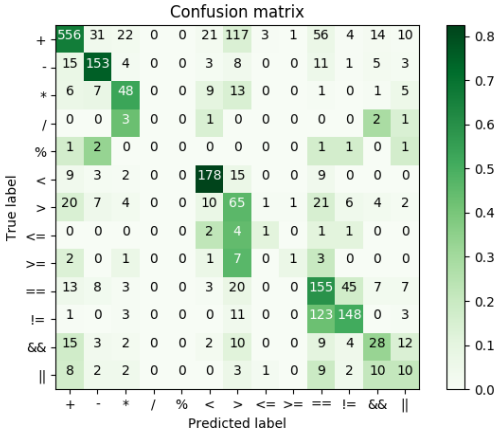


図 6. M_B を用いた実験の結果の混同行列

の 1 行 2 列目の 100 は，空欄に当てはまる正解の演算子は '+' だがモデル M_F が推定した結果が '−' である場合の数が 100 であったことを示している．

各モデルで推定した結果から得られた F 値を表 4 に示す．M_{FB} を用いた推定結果が最も良く，M_R を用いた推定結果が最も悪い結果となった．3.5.6 節で述べたように，マイクロ平均を用いた場合 F 値と正解率は一致するので，表 4 は正解率の表ともいえる．

各モデルで，学習データ全てを用いて学習を行ったときと，テストデータ全てを用いて演算子の推定を行ったときの所要時間を表 5 に示す[1]．M_{FB} の場合，推定時間

[1]Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz を 2 つと 64 GB の RAM を使用した．GPU は使用していない．

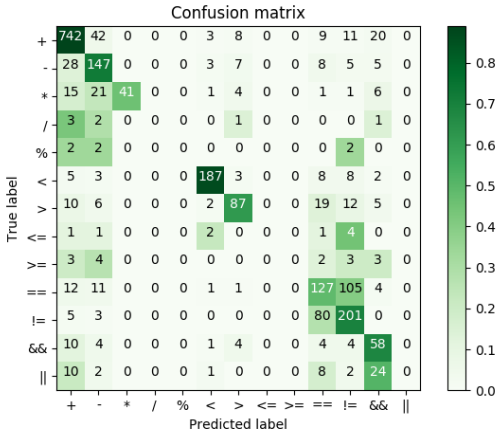


図 7. M_{FB} を用いた実験の結果の混同行列

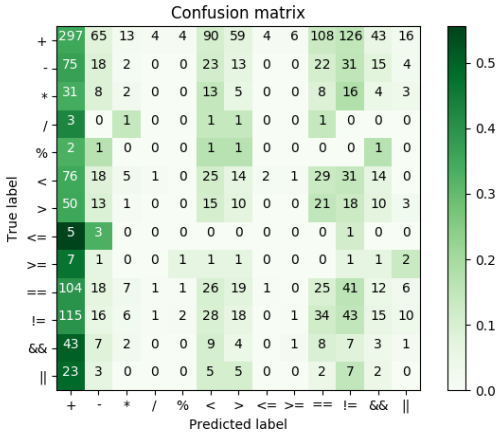


図 8. M_R を用いた実験の結果の混同行列

表 4. 各モデルの実験結果の F 値

	F 値
M_F	0.653
M_B	0.609
M_{FB}	0.721
M_R	0.192

表 5. 各モデルの学習時間と推定時間

	学習時間（秒）	推定時間（秒）
M_F	19,524	174
M_B	9,311	84
M_{FB}	40,468	181
M_R	-	-

は 181 秒でありテストデータのファイル数は 2,204 個であるので，1 つの演算子を推定するのに約 0.08 秒の時間を要することがわかる．また，M_R は学習モデルではないため計測していない．

7. 考察

7.1. 研究設問への回答

実験結果をもとに，研究設問に回答する．

RQ1: 空欄より前の情報から空欄に入る演算子をどの程度の精度で推定できるか．
約 65%の正解率で推定できた．

RQ2: 空欄より後の情報から空欄に入る演算子をどの程度の精度で推定できるか．
約 61%の正解率で推定できた．

RQ3: 空欄以外の情報から空欄に入る演算子をどの程度の精度で推定できるか．
約 72%の正解率で推定できた．

RQ4: 以上の 3 つの方法で推定した結果に差異はあるか．
4〜11%程度の正解率の差があった．

7.2. 結果全体に関して

表 3 より，'+' はソースコードに出現する演算子の約 4 割を占めるが，他の演算子の出現率は 1 割程度以下となっている．'+' を除くと演算子の出現率の偏りが小さいことから，重み付けされた確率からランダムに推定する手法の正解率は低くなることが予想され，その結果は表 4 より約 19%となった．一方，提案手法では最大 72%の正解率を得ることができた．これらの結果から，空欄の前後のソースコードと欠落した演算子の間には何らかの関係性があり，深層学習を用いることでその関係性を学習できると考えられる．

空欄より後の情報から推定した M_B の実験結果の正解率が 61%であった．M_B の推定手法では，例えば "`if(a < b)`" というソースコードがあったとき，トークナイズして演算子を '??' に置き換えると，"`if a ?? b`" というトークン列になり，"`if`" を読まずに "`??`" を推定することになる．しかし，図 6 より，'<' は約 82%，'>' は約 46%の正解率が得られた．また，'<' と '>' の出現回数に大きな差はない．これらのことから，'if' などのキーワードが無くともその後の文からある程度の正解率で空欄に当てはまる演算子を推定できることを示している．空欄より前の部分を使用した場合には劣るが，空欄より後の部分にも演算子の正しい推定に重要な特徴が多く含まれていると考えられる．しかし，'<' と '>' の正解率の差については構文解析を利用するなど，別途考察する必要があると考えられる．

空欄より前の情報から推定する M_F と空欄の前後の情報から推定する M_{FB} の間には約 7%の正解率の差があった．空欄より前の情報だけでは演算子の候補を絞りきれない場合に，空欄より後の情報も用いることによって正しく推定できるようになったと考えられる．演算子の種類や空欄の場所により，空欄の前後の情報のどちらが演算子を正しく推定するための情報として有用であるかが変化することが考えられる．

表 5 より，M_F と M_B の学習時間と推定時間に差があるのは空欄前後のトークンの数が異なるためであると考えられる．空欄より前の部分のトークン t_F には import 文やクラス名などがトークンとなって含まれており，空欄より後の部分のトークン t_B に比べてトークンの量が多くなる傾向があると考えられる．それぞれの合計ファイルサイズを測定した結果，t_F は 26 MB，t_B は 16 MB であった．

7.3. 妥当性への脅威

本研究ではソースコードをトークンに変換するため自作したトークナイザを使用している．しかし，一部のソースコードに対してこのトークナイザを適用する場合に不具合があることが確認されている．例えば，'<' や '>' が演算子として使われていない場合でも演算子として扱ってしまう．そのため，構文解析を利用するなどして，より精度の高いトークナイザを準備する必要があると考え

られる．

空欄より前の情報から推定する M_F は後の情報から推定する M_B に比べて正解率が 4%程度優れていた．しかし，7.2 節にて言及した，トークンファイル t_F と t_B の間にサイズ差があることが M_F と M_B の推定精度の差に影響することが考えられる．そのため，空欄より前の部分と後の部分のどちらがより多くの特徴量を含んでいるか議論するには，トークンファイルに手を加えるなどして検証する必要がある．

7.4. 今後の課題

LSTM 層の多層化

本研究で作成したモデルでは LSTM を 1 層だけ使用している．LSTM を複数使用して層を深くすることによりモデルの表現力が増し性能が向上することが一般的に示されている．ただし，層を深くしすぎると反対に性能が下がる場合もあるので，適切な深さを調整する必要がある．本研究のモデルはそのような調整を行っていないので，LSTM 層の深さを調整する必要がある．

演算子の種類の追加

本研究では推定する演算子の種類を限定した．しかし，実用することを視野に入れるには推定できる演算子の種類を増やす必要がある．そのため，実験対象の演算子の種類を増やし，モデルやパラメータの調整をする必要がある．

他の言語での実験

本研究では Java のみを実験対象とした．他の言語に対応したトークナイザを用意することで，様々な言語のソースコードを対象に実験を行うことができる．それぞれの言語に対応させるため，実験を行いモデルの改良をする必要がある．

8. まとめ

本研究では，ソースコードをトークンに変換して機械学習モデルの入力とすることで空欄に当てはまる演算子を推定する手法を提案した．実験の結果，61〜72%の正解率で推定できた．このことから，ソースコードには空欄に入る演算子を推定するための特徴が含まれており，深層学習を用いることでその特徴を学習できると考えられる．

今後の課題としては，トークナイザやモデルの改良，実験対象となる演算子や言語を追加して汎用性を向上させることが挙げられる．

参考文献

[1] Y. Yang, K. Sakamoto, H. Washizaki, and Y. Fukazawa, “A tool for suggesting similar program element modifications,” 研究報告ソフトウェア工学（SE）, vol.2014, no.20, pp.1–6, jul 2014.

[2] 山本哲男，吉田則裕，肥後芳樹，“ソースコードコーパスを利用したシームレスなソースコード再利用手法,” 情報処理学会論文誌，vol.53，no.2，pp.644–652，feb 2012.

[3] Keras, Home - Keras Documentation, （オンライン）, 入手先〈https://keras.io/ja/〉（参照 2019-2-13）.

[4] D.P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” arXiv e-prints, p.arXiv:1412.6980, Dec. 2014.

[5] The Apache Software Foundation, Welcome to The Apache Software Foundation!, （オンライン）, 入手先〈https://www.apache.org/〉（参照 2019-2-1）.

ソフトウェアエンジニアリングにおけるコミュニケーション技術の活用

艸薙 匠　　田村 朱麗　　藤原 聡子
株式会社 東芝　ソフトウェア技術センター
takumi.kusanagi@toshiba.co.jp　shurei.tamura@toshiba.co.jp　satoko.fujiwara@toshiba.co.jp

要旨

ソフトウェアは未だ人が作るため、コミュニケーションが重要であると言われている。以前から様々なソフトウェアプロセスの中でコミュニケーションについて経験知がまとめられてきた(例:ピアレビュー等)。一方人文科学系ではコミュニケーションについて様々な取り組みが定式化しつつあるが、その成果を整理して、ソフトウェアエンジニアリングの中で十分活かせているとは言い難い。

そこで本発表では、技術者向けのコミュニケーション技術として、特に近年注目されている、ファシリテート(ファシリテーション)、ティーチング、コーチング、カウンセリングの 4 つの方法に着目し整理する。その上でソフトウェアエンジニアリングに活用している社内事例を紹介する。なおそれぞれのソフトウェアエンジニアリング技術は、コミュニケーション以外の側面の方が大きいと思われるが、本報告ではコミュニケーションという切り口で取り扱う。また、「技術」という言葉を本報告では、「物事をたくみに行うわざ(広辞苑)」という定義で用いる。

1. 4 つの代表的なコミュニケーション技術

図 1 及び図 2 に上述した代表的な 4 つのコミュニケーション技術について整理した。

ファシリテート(ファシリテーション)技術とは、話し合う方法である。一般的には会議の方法を指す場合が多く、集団にむけて実施する[1][2]。

ティーチング技術とは、教え伝える方法であり、一般的な企業内研修や個人指導などが含まれる。最近のインストラクショナルデザイン、アクティブラーニング、e ラーニングなどが含まれる[3][4]。

コーチング技術とは、本人に気づかせる方法であり、集団向けに行う場合もあるが、ビジネスコーチングでは 1 対 1 の面談方式で行われる場合が多い[5][6][7]。

カウンセリング技術とは、癒やす方法である。ビジネスの場では、治療というよりも、お互いにお互いの存在を認め合うことによって、組織や個人の活性化する方法として提案されている[8][9][10]。

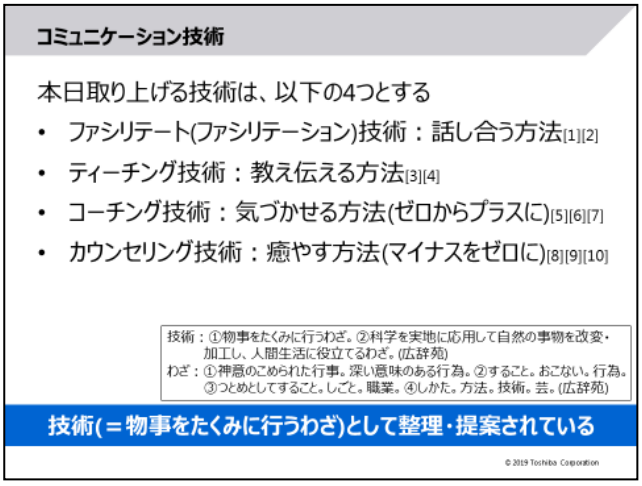
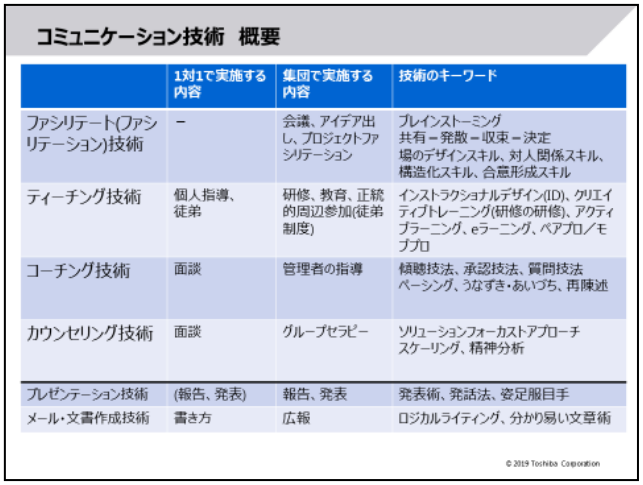


図 1　コミュニケーション技術　代表例



	1対1で実施する内容	集団で実施する内容	技術のキーワード
ファシリテート(ファシリテーション)技術	－	会議、アイデア出し、プロジェクトファシリテーション	ブレインストーミング 共有－発散－収束－決定 場のデザインスキル、対人関係スキル、構造化スキル、合意形成スキル
ティーチング技術	個人指導、徒弟	研修、教育、正統的周辺参加(徒弟制度)	インストラクショナルデザイン(ID)、クリエイティブトレーニング(研修の研修)、アクティブラーニング、eラーニング、ペアプロ／モブプロ
コーチング技術	面談	管理者の指導	傾聴技法、承認技法、質問技法 ペーシング、うなずき・あいづち、再陳述
カウンセリング技術	面談	グループセラピー	ソリューションフォーカストアプローチ スケーリング、精神分析
プレゼンテーション技術	(報告、発表)	報告、発表	発表術、発話法、姿足服目手
メール・文書作成技術	書き方	広報	ロジカルライティング、分かり易い文章術

図 2　コミュニケーション技術　概要

2. ソフトウェアエンジニアリングにおける活用事例

ここでは、図 3 にあるようなソフトウェアエンジニアリング場面での活用事例を紹介する。

アジャイルでは、アジャイルコーチング[11]を参考に、ビジネスコーチングの要素を加味してコーチング技術の基礎的な内容をスクラムマスター向けに教育し、朝会等の運営の改善に活かしてもらった。

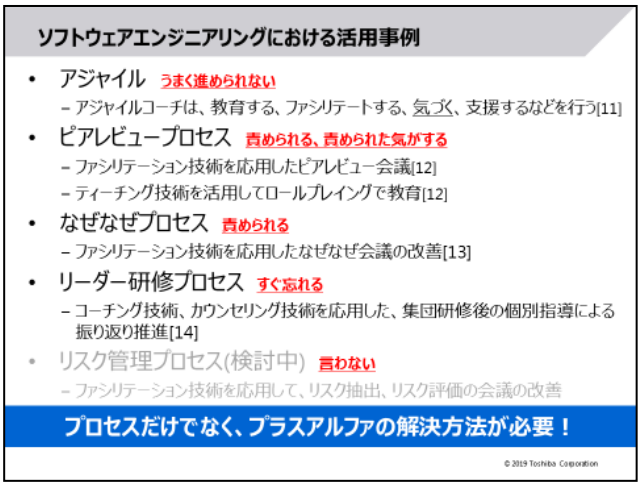


図 3 ソフトウェアエンジニアリングにおける活用事例

ピアレビューでは、ファシリテーション技術を応用してレビュー会議プロセスを改善し、ティーチング技術を活用してロールプレイングでそのプロセスを教育した[12]。

なぜなぜプロセスでは、「責められる」という組織課題に対して、従来のなぜなぜプロセスを参考に[13]、図 4 のようにファシリテーション技術を活用した司会の役割を定義し、管理職から担当まで、職場ぐるみで教育した。

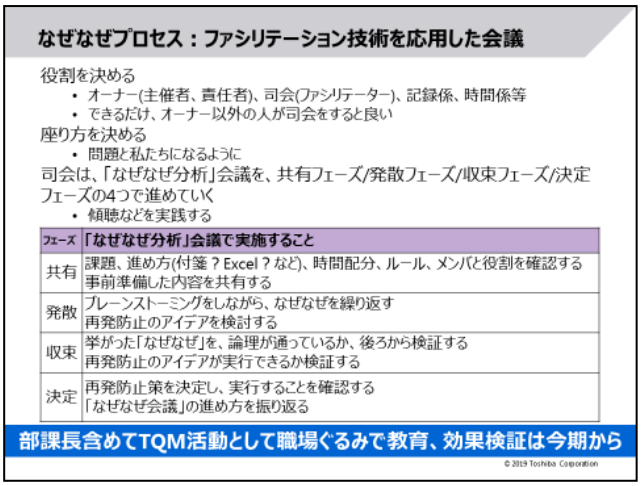


図 4 なぜなぜプロセスでの活用

その他、リーダー研修への応用実践事例[14]があり、またリスク管理などへの応用も検討している。

3. まとめ

ファシリテーションやコーチングなどコミュニケーション技術として代表的なものを整理し、活用事例を紹介した。ソフトウェアは人間が作っているので、ソフトウェア設計開発技術そのもの以外にも、現場を助けるヒントはあるはずだ。適用事例も出てきている[15][16]。定式化が進んでいる「コミュニケーション技術」を上手に活用して、ソフトウェア技術者が十分力を発揮できるようにしていきたい。

参考文献

[1]特定非営利活動法人日本ファシリテーション協会、ファシリテーション基礎講座、2018
[2]堀公俊、ファシリテーション入門、日本経済新聞出版社、2004
[3]中村文子、ボブ・パイク、講師・インストラクターハンドブック、日本能率協会マネジメントセンター、2018/3/2
[4]永谷研一、人材育成担当者のための絶対に行動定着させる技術、ProFuture、2015/8/31
[5]日本コーチ連盟認定コーチ養成プログラム基礎コース/応用コース、日本コーチ連盟コーチアカデミー、2018
[6]伊藤守、コーチング・マネジメント―人と組織のハイパフォーマンスをつくる、ディスカヴァー・トゥエンティワン、2002/7/24
[7]コーチ・エィ、鈴木 義幸((監修)、新版 コーチングの基本、日本実業出版社、2019/1/12
[8]倉成央、谷口祥子、クライアント満足を10倍にする カウンセリングとコーチングの合わせ技、秀和システム、2017/10/4
[9]ピーター・ディヤング、インスー・キム・バーグ、解決のための面接技法―ソリューション・フォーカストアプローチの手引き、金剛出版、 2004
[10]青木安輝、 解決志向(ソリューションフォーカス)の実践マネジメント、河出書房新社、2006
[11]Rachel Davies・Liz Sedley、永瀬美穂・角征典訳、アジャイルコーチング、オーム社、2017/1/15
[12]藤原聡子、ファシリテーション技術を応用したピアレビュー会議の改善、SPI JAPAN 2018
[13]小倉仁志、問題解決力がみるみる身につく 実践 なぜなぜ分析、日本経済新聞出版社、2013/2/1
[14]田中史朗、3 つの行動変化を狙ったプロジェクトリーダ研修(<特集>プロジェクトマネジメント教育)、プロジェクトマネジメント学会誌、Vol.17、No.2、pp.8-12, 2015
[15]弦間健他、指摘を前向きに受け止めてもらうためのレビュー手法提案、SQiP 研究会、2019/2/22
http://www.juse.or.jp/sqip/workshop/report/attachs/2018/2_shiteki_ronbun.pdf
[16]小野瀬昌人、社内講座や社内コミュニティを活用したファシリテータの育成、PM 学会誌 Vol.21 No.2、2019/4/15

コンセプト＆ゴール指向のふりかえりの提案

小楠 聡美
株式会社 HBA
ogusuxs@hba.co.jp

要旨

筆者は，過去に経験した開発現場でのふりかえりについて，「ふりかえりの効果が実感できない．」「ふりかえりをやる意味があるのか？」と思っていた．
そして，その原因として，複数の要因があると考えた．
そこで筆者は，上記の要因を解消して，より効果を実感できるふりかえりを実施できるような手段はないかと考え，実際のものづくりにおけるふりかえり手段として活用した
．
本事例では，「ふりかえりの効果が実感できない」要因を解決できるようなふりかえりの手段と，実際のものづくりにおける改善実践例，およびその効果を報告する．

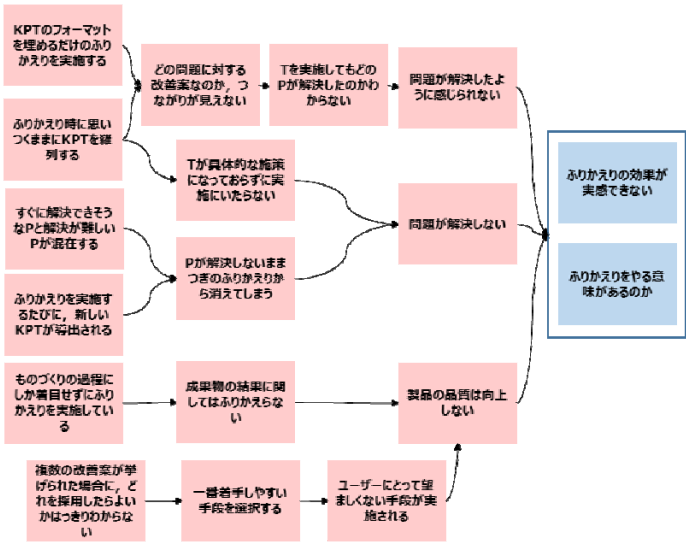
1. はじめに

筆者が過去に経験した開発現場でのふりかえりは，以下のように進められることが多かった．

- KPT[1]の様式に沿って自由に意見を挙げている．
- 前回のふりかえり終了後から，次のふりかえりの期間内の作業をふりかえった KPT を出している．

そのようなふりかえりを実施しながら開発プロセスを進めても，問題と思っていたことが解決されない，あるいは解決したと感じられなかった．そのため，「ふりかえりの効果が実感できない．」「ふりかえりをやる意味があるのか？」と感じていた．なぜそのように感じるのか，その要因は複数あり，以下の構造になっていると考えた（図 1）．

図 1 ふりかえりの効果が感じられない要因



筆者の開発現場と同じように，目先の過程にしか着目していないふりかえりや，KPT などの様式に沿ってただ思ったことを羅列するふりかえりをしている組織も多いのではないだろうか．そのようなふりかえりは効果が薄く，問題がいつまでも解決しないままのチームになったり，最終的にふりかえりすることが目的になってしまったりするのではないかと筆者は考えている．

そのため，図 1 の以下の要因に着目し，それを改善することでもっと効果を実感できるようなふりかえりをできないかと考え，実践してみた．

- どの問題に対する改善案なのか，つながりが見えない．
- P が解決しないまま次のふりかえりから消えてしまう．
- ものづくりの過程にしか着目せずにふりかえりを実施している．
- 複数の改善案が挙げられた場合に，どれを採用したらよいかはっきりわからない．

2. 解決手段と効果（提案内容）

2.1. 解決手段

効果の実感を促進するふりかえりとして，新規製品作成の際に，以下のルールを取り入れた MindMap を用いてふりかえりを実施しながら，何度か改善を試みた（図2）．

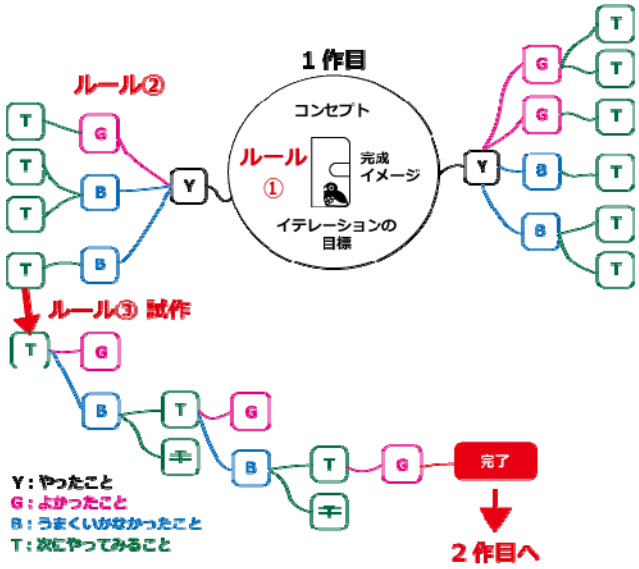
＜ルール＞

① やったこと（Y）に対して，その下層のブランチによかったこと（Good，以下 G とする）／うまくいかなかったこと（Bad，以下 B とする）を挙げ，さらにそれぞれにブランチをきって，次にやること（T）を列挙する．

② MindMap の中心に，完成イメージとコンセプト，そのイテレーションにおける目標を置く．

③ ふりかえりの結果から出た「次にやること（T）」において，うまくいくか試験的に試した方がよい部分や，改善に時間がかかりそうな部分は切り出して，そこだけを深堀りしてうまくできるまでふりかえりながら実施する．うまくいったものは，本番にそのまま組み込み，必ず部分試作をすべて完了させてから次の代を作成するようにする．

図 2. ふりかえり様式イメージ



MindMap にルール①を適用することで，挙げられた T がどの G や B に割り付くものなのか一目でわかるので，「どの問題に対する改善案なのか，つながりが見えない」という要因が改善されるのではないかと思ったのと，一つの B と G に割り付く T がグループ化されるので，どの T を実践するか選びやすくなり，「複数の改善案が挙げられた場合に，どれを採用したらよいかはっきりわからない」要因が改善されるのではないかと考えたためである．KPT の項目ではなく、YGBT の各項目を使用したのは以下の理由による．

- KPT だけだと、どんな行動に対する K や P なのかが明確にならない場合がある．
- 実業務をしていく上で，まったく同じ状況になることはほぼないため，「Keep」という意味でそのまま繰り返されることは少ないため，「Keep」という言葉に違和感を抱いており，「Good」の方が合っていると感じていたため．
- 「Keep」の代わりに「Good」を使うのであれば、その対語として「Problem」には「Bad」の方がよいと考えた．

ただし，「Keep」と「Good」，「Problem」と「Bad」の用途に差はないと考える．

また，ルール②により，製品がコンセプトに合っているかという視点でふりかえりをすることができ，「ものづくりの過程にしか着目せずにふりかえりを実施している」の要因が改善できるのではないかと考えた．

そしてルール③により，解決が難しい B に対する T の部分的な試作を取り入れることで，問題が解決できる見通しが立たない B を残したまま次の本番製作に着手することがなくなり，「P が解決しないまま次のふりかえりから消えてしまう」ことを防ぎ，実施されないまま忘れられてしまう T がなくなるのではないかと考えた．

2.2. 今回の成果物について

今回の解決手段の効果確認として，システム開発ではなく，レザークラフトによるスマートフォンケースの製作で実施した．その理由は以下の通りである．

＜スマートフォンケースの製作で効果確認した理由＞

① スマートフォンケース製作において，市販の型などは参考にせず，設計から筆者自身でやろうと考えていた．

② 筆者はレザークラフト初挑戦で，最初から完璧に作れるとは思っておらず，作ってうまくできないとこ

ろを改善しつつ，何度か作成してみようと思っていた．このプロセスが，ソフトウェアの新規開発→保守・改修のプロセスに似ていると考えたため．

③ ソフトウェアの設計やコーディングにも，レザークラフト製作にもセンスが必要である．そのセンスによって成果物の出来栄えが変わるという点においても，ソフトウェア開発と類似している．

以上のことから，スマートフォンケースの製作の過程に今回のふりかえりを取り入れて実践した．

なお，今回のスマートフォンケース製作のコンセプトは以下の通りである．

「インコに噛まれてももう泣かない」

上記の理由は，筆者はスマートフォンケースを新しく買うたびに，短期間でインコにボロボロにされていた．そのたびに買い替えていたのだが，コストがかかる上に機種が古くなってきたので，なかなか自分が気に入ったデザインのケースが見つからなくなってきた．そのため，安いコストで自分好みのケースを自分で作りたいと思ったからである．

以上の理由から上記コンセプトが生まれ，以下の方針が導出された．

・コストは安く抑える．
・自分好みのデザインにする．
（絵柄，デザイン，色，重さなど）

2.3. 手段実施の流れ

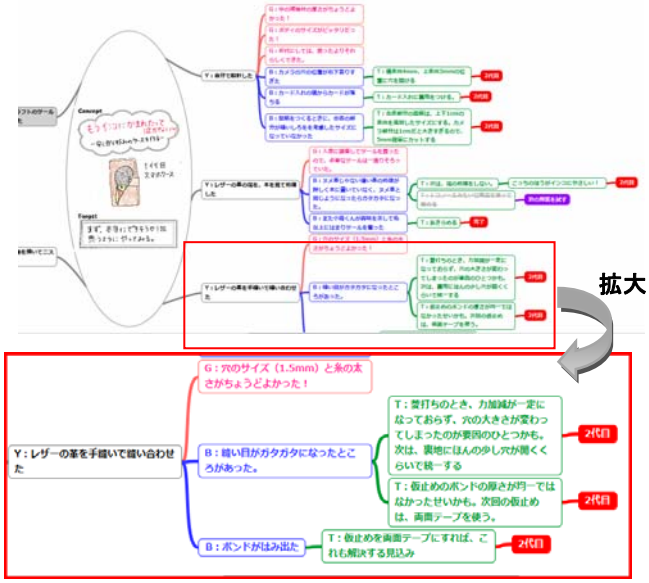
上記でも述べたように，筆者にとって今回レザークラフト，およびスマートフォンケースの製作はいずれも初めての試みである．ふりかえりを実施して改善しながら 1 代から 3 代目まで，以下のコンセプトと各代ケース作成のゴールを立てて 3 つのスマートフォンケースを作成した．

表 1. コンセプトと各代製作のゴール

コンセプト		
インコに噛まれてももう泣かない −安く自分好みのケースを作る−		
ゴール		
1 代目	2 代目	3 代目
まず，本当にできそうか 1 回思うようにやってみる．	1 代目のうまくいかなかったところを改善する．	売れるクォリティで仕上げる．

段階的に改善を進めて製作するにあたり，3 代目で完成させるというゴールを定め，各製作が終わる都度，中央にコンセプトと各ゴールを置いた MindMap のフォーマットを用意し，やったこと(Y)を挙げ，それに対するよかったこと(G)とうまくいかなかったこと(B)，および，それらに割り付く，次にやること(T)を挙げた(図 3)．

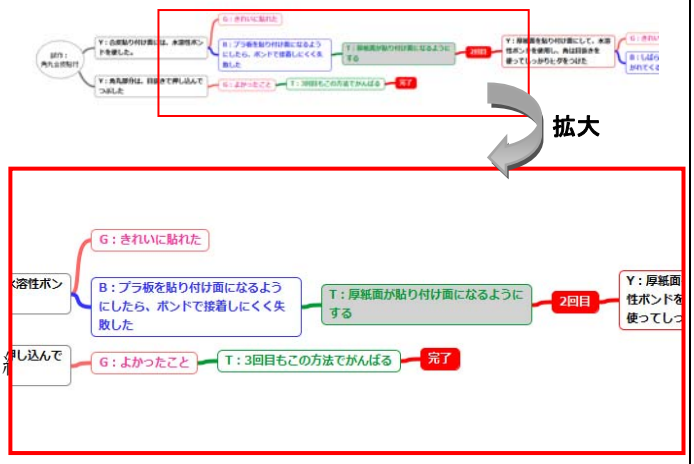
図 3. MindMap によるふりかえりの例



ただし，中央にコンセプトやゴールを入れない場合と入れてふりかえりを実施した場合の効果の違いを確認するために，1 代目のふりかえりの際は，まず中央にコンセプトなどは置かずにふりかえりを行い，その後コンセプトとゴールを置いてふりかえりを再度実施し，両者の違いを確認した．

ふりかえりを行って次代のケース製作を実施する前に，部分的に試作をしてみた方がよいと判断した B-T については，MindMap の該当項目を新規 MindMap に抜き出して，完了するまで試作とふりかえりを同 MindMap 上で繰り返した(図 4)．完了したパーツを組み込み．次代のケースを作成した

図 4. 試作部分のふりかえりの例



3. 結果

今回の試みの結果，この手段の各ポイントとなる点について，以下のことがわかった．

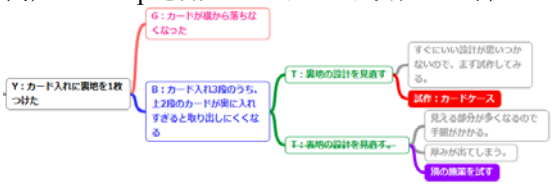
3.1. MindMap 形式にルール①を取り入れたふりかえり

既存のフレームや，表形式などではなく MindMap を使って Y-G-T／Y-B-T を導出したところ，以下のことがわかった．

①全体の俯瞰のしやすさが向上した

構造化されているため，全体を俯瞰しやすく，Y-G-T や Y-B-T のつながりといった各要素のつながりがわかりやすかった．また，T が割り付いていない B がすぐに発見できた．

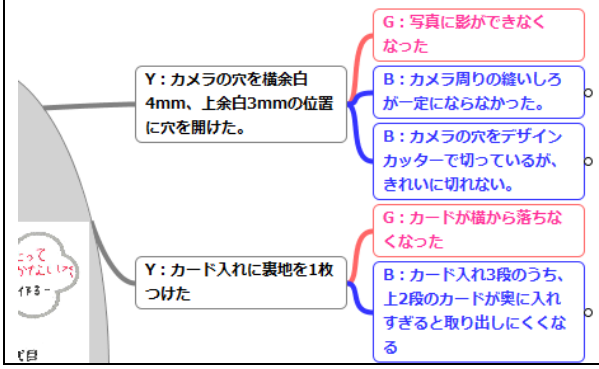
例）MindMap を用いたふりかえり項目の一部



②G の件数と B の件数が比較しやすい

構造化されているのと G と B の色分けにより，1 件の Y に対する G と B の件数の比較，およびそれらに割り付く T の件数の比較し易くなった．

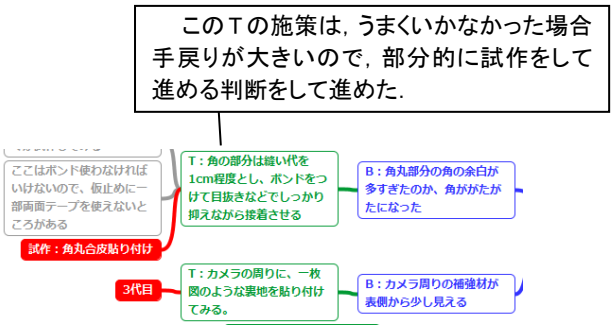
例）Y に割り付く G と B の件数例



③手戻りの抑止となった

1 件ごとの T に自由にメモやさらなる検討事項を追加して残すことができるのとその見やすさから，部分試作が必要な問題と部分試作不要な問題の切り分けが容易になった．これにより，手戻りすることが少なく製作を進めることができた．

例）部分試作を試みたふりかえり項目



④文字と図を同時に扱いにくい

図を挿入して説明したい箇所があったが，思うようなレイアウトにできなかった．また，画像を挿入したファイルを別のコンピューターで表示すると表示できないことがあった．

例）MindMap ツールへの画像取り込み

実際にどう変わったかを画像で残しておきたかったのだが，挿入すると文字が消えてしまった．

3.2. MindMap にルール②を取り入れたふりかえり

MindMap の中央に，「コンセプト」「完成イメージ」「イテレーションの目標」を置いてふりかえりを行った結果として，以下のことがわかった．

⑤異なる視点のふりかえり項目を導出できた

中央にコンセプトとイテレーションの目標を置いてふりかえりをした際，置かずにふりかえりを実施した場合とは異なる視点のふりかえりが出た．

例）1 代目ケースのふりかえりの Y 導出結果

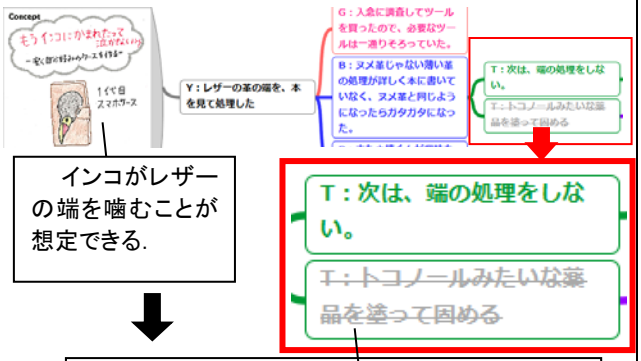
コンセプト配置前	コンセプト配置後
・自分で設計した ・革の端を，書籍を見ながら処理した ・革を手縫いで縫い合わせた ・電気ペンを使って絵を描いてニスを塗った	・自分で設計した ・革の端を，書籍を見ながら処理した ・革を手縫いで縫い合わせた ・電気ペンを使って絵を描いてニスを塗った **・革，布，レザークラフトのツール以外は 100 均で揃えた**

コンセプトに合った製品にするために何をしたかという視点のふりかえり項目が挙がった．

⑥コンセプトに合った施策案を導出できた

1 件の G や B において複数の T 案が出た場合に，コンセプトに立ち戻ってどの T 案がふさわしいかを検討できた．

例）レザーの端処理の改善案検討



インコがレザーの端を噛むことが想定できる．

インコが噛まないのであれば下案のほうが適切だが、コンセプトから噛むことが想定でき，薬品はインコに有害な成分が含まれる可能性があるため，上案のほうが望ましいと判断できた．

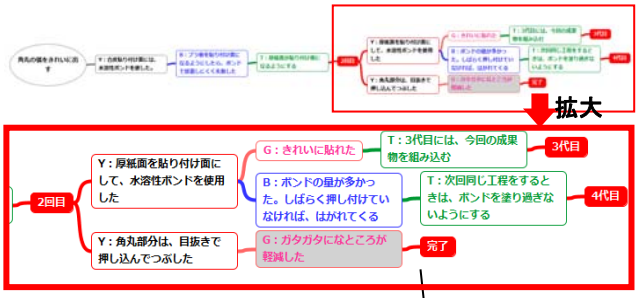
3.3. ルール③を取り入れた MindMap によるふりかえり

試作部分について，新規 MindMap に抜き出して完了するまで，試作とふりかえりを同 MindMap 上で繰り返して改善を行った結果から，以下のことがわかった．

⑦改善必要と判断した B はすべて改善できた

今回 3 代目までのスマートフォンケースの作成とふりかえりを実施中に，その間で改善しようと判断した B で改善できないままとなったものは 1 件もなかった．つまり，改善必要なすべての B において T は完了するまで実施された．

例）部分試作の MindMap



すべての T について「完了」となるか，次の見通しを立てるところまで進めることができた．

⑧次代の目標が部分試作の完了基準となった

部分試作でどの程度まで改善できたら「完了」とするか，次代のゴールから判断することができた．例えば、3 代目のゴールは「売れるクォリティにする」なので，市販されている類似製品と同等の品質になるまで改善する必要があると判断し，試みた．

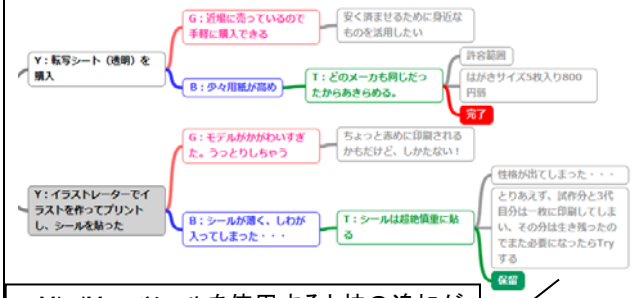
例）ケース留め具の型（角丸）にうまく革を貼り付ける部分試作を繰り返した結果（上：改善後，下：改善前）．

3 代目に組み込むパーツなので，市販品と同じように滑らかな弧になるまで改善を試みた．

⑨施策選択の経緯や実施の過程が見えた

複数の T の候補からその施策を選んだ経緯や注意点などのメモを，MindMap では容易に枝を追加して記載できた．こうすることで，完了後にその MindMap を見直すと，なぜその手段を選択したのか，どのような経緯で現在の状態に至るのかを枝を辿って読み取ることができた．

例）柄プリント部分試作における MindMap



MindMap ツールを使用すると枝の追加が容易なので，該当箇所に置けるメモや注意点などを残すことができる．

4. 考察

今回の結果から，中心に製品のコンセプトとそのイテレーションの目的を置いた MindMap 形式のふりかえりを実施した結果をまとめると以下の通りである．

＜結果＞
①全体の俯瞰のしやすさが向上した．
②G の件数と B の件数が比較しやすい．
③手戻りの抑止となった．
④文字と図を同時に扱いにくい．
⑤異なる視点のふりかえり項目を導出できた．
⑥最適な施策案を導出できた．
⑦改善必要と判断した B はすべて改善できた．
⑧次代の目標が部分試作の完了基準となった．
⑨施策選択の経緯や実施の過程が見えた．

①の結果から，挙げられた T がどの G や B に割り付くものなのか一目でわかるので，冒頭で述べた「どの問題に対する改善案なのか，つながりが見えない」という要因は発生しなくなると考えられる．また，一つの B や G に対する T が同列に列挙される形となるので，どの T を実施するか選びやすくなる．このように，ふりかえり結果が構造体として表現するため，本ふりかえりの様式は，結果が

見やすく，現状把握が容易になる様式であると言える．
また，解決したものについては「完了」の識別メモを付けるようにしたが，これによりメモが書かれていないものは解決していないままとなっているBやまだ完了していないTと判断できる．構造化されて全体が俯瞰しやすいので，このような何も解決していないBや，完了していないTを見つけやすい．つまり，改善項目を漏れることなく実施しやすいと言える．

②の結果から，Gの件数がBの件数よりも多いほど，メンバーの今回の結果に対する満足度が高い状態であり，それが構造化により俯瞰しやすくなっているため，その状況を，ふりかえり結果を見てすぐに読み取れるということがわかる．つまり，このふりかえり結果がプロジェクトの状況を測る一つのメトリクスとなりうる．
また，後述の⑤の結果より，プロジェクトの過程だけではなく製品の品質にも着目したふりかえりができるため，このふりかえりでプロジェクトの状況と製品の品質の両方を見ることができる．

③の手戻りが少なく製造を進めることができた結果からは，このふりかえりを実施しつつものづくりを行うことで，効率的に開発・製造を進められることがわかる．

④の結果は，チームメンバーでふりかえり結果を共有しにくい要因となるため，今後の課題であると考えている．

⑤の結果から，製品がコンセプトどおり作られているかどうかという視点のふりかえりができたと判断でき，本ふりかえり手法により，製品ユーザーに受け入れられるかという視点，つまり，製品の品質に着目したふりかえりもできるようになったと言える．これにより，冒頭で挙げた「ものづくりの過程にしか着目せずにふりかえりを実施している」という問題は解決できる．

⑥の結果からは，本ふりかえりにより，ユーザーの立場に立った最善の改善策を選択できると言える．つまり，本ふりかえりにより，「複数の改善案が挙げられた場合に、どれを採用したらよいかはっきりわからない」問題を解消し，ユーザー目線の開発・ものづくりができるようになる．①の俯瞰しやすさ向上の結果が，この選択をより行いやすくしてくれる．

⑦の結果では，長引きそうなTについては抜き出して完了するまで実践とふりかえりを繰り返すことになるので，完了する前に新しいふりかえり結果が導出されても，実施されないまま見なくなってしまうことはなくなる．これにより，冒頭で述べた「Pが解決しないまま次のふりかえりから消えてしまう」ことを防ぐことができる．

⑧では，段階的に目標を置いたが，それが今回の改善の完了基準になった．ソフトウェアの開発，およびプロセスの改善や品質の改善を試みると，「どこまでやればいいのか」という話がよく挙がるが，この結果から，本手法を取り入れた開発・改善を実施すると，このゴールが，「どこまでやるか」の指標となると言える．例えば，ソフトウェアのテストにおいても「どこまでテストすればいいのかわからない」と思うことがよくある．その際にこの目標が，どの程度詳細なテストをするのがよいかの指標になってくれる．これも無駄な作業をしない，品質が悪いまま次に進まないための手段となる．

⑨の結果では，MindMapの途中に残すメモが，なぜ現状このような結果になっているのかの経緯を示してくれることがわかったが，このような途中経過を残すことで，よく開発現場で聞く「前任者が不在でどうしてこうなったのか誰もわからない．」といった事態も避けることができるのではないだろうか．つまり，チームで開発状況を共有するにも役立つ手法であるともと言える．
また，今回の結果から，この判断の根拠や途中経過を見ながらふりかえりを実施し，その内容を残すことは，チームメンバーの教育にもつながるであろうと感じた．例えば，熟練者が製作・実装を行った場合，初心者がそのふりかえり結果を見ると，熟練者がどう考えてこの結果になったのか道筋が明確に見えるためである．
さらに，教える立場（熟練者）側にとってもメリットがあると感じた．なぜならば，前述で「ソフトウェア設計，コーディング，レザークラフトにおいてセンスが必要」と述べたが，センスを持つ人は時に直感で自分の考えが当たり前として設計や製作等を進めてしまい，なかなかその理由を説明できないことがある．そういった場合にもこのふりかえりの過程を辿り，自分がどうしてそのように考えたのかを相手に伝えることができるようになると考えている．

5. まとめ

今回の結果から，中心に製品のコンセプトとそのイテレーションのゴールを置いたMindMap形式のふりかえりを実施することで，ものづくりにおいて，冒頭で述べた以下の問題に有効であることがわかった．

- どの問題に対する改善案なのか，つながりが見えない．
  - 構造化して表現するので，各要素のつながりが見

えるのと，全体が俯瞰しやすい．

● P が解決しないまま次のふりかえりから消えてしまう
  ・すぐに解決が難しそうな T については，抜き出して完了するまで実践とふりかえりを繰り返すので，完了するまで続けることができる．

● ものづくりの過程にしか着目せずにふりかえりを実施している．
  ・ものづくりの過程だけではなく，製品がコンセプトに合っているかという視点のふりかえりができる．

● 複数の改善案が挙げられた場合に，どれを採用したらよいかはっきりわからない
  ・複数の改善案が挙がった場合，中央のコンセプトに着目し，ユーザー目線の改善策を選択できる．

上記に加えて，以下のこともわかった．

◎ プロジェクトや製品の品質の状態を表すメトリクスとして活用できる．
  ・G と B の件数を比較することで，プロジェクトや製品の品質における現状を探ることができる．

◎ チームで開発状況を共有するにも役立つ手法になる．
  ・前任者不在でも，開発の経緯がわかる．

◎ 初心者スキルアップの教育としても活用できる．
  ・熟練者の開発経緯を見ることで，どう考えて作ればよいのかの思考を探ることができる．
  ・教育指導の際にうまく表現できない場合にもこれを見ながら説明することができる．

今後は他のツールを含め，図の扱いを容易にする手段や他者と共有しやすくするための手段の検討，またはより図も扱いやすいツールがないか検討すると共に，この手法をシステム開発の現場に適用し,より大きなプロジェクトの現場で活用できるか探っていきたい．

参考文献

[1] プロジェクトファシリテーション 実践編 ふりかえりガイド *http://objectclub.jp/download/files/pf/RetrospectiveMeetingGuide.pdf*,

＜違い＞を捉えよう ～違いの分析から始める問題解決～

常盤　香央里
WACATE 実行委員会
caori_t@outlook.com

要旨

組織で発生する様々な“人に起因する問題”を分析すると，その背景にある人の考え方やコンテキストなどの＜違い＞に収束した．この＜違い＞を意識的に捉えることにより，“人に起因する問題”の抑制や対処が可能であると考えられる．

そこで＜違い＞を捉えることを意識的かつ段階的に体験し試行してもらう場として，新たなワークショップを考案し実施した．以下の 2 回のワークショップ実践事例を元に具体的な内容と効果を述べる．

・初回：WACATE 本会（ソフトウェアテストの 1 泊 2 日合宿形式勉強会）での 1 セッションとしての実施

・2 回目：プチ WACATE（WACATE 主催の個別勉強会）での本セッションのみの再演

1. はじめに・背景

ソフトウェア関連業務はさまざまな考え方やコンテキストを持つ人達が集まってモノゴトを作り上げるものである．顧客との調整・折衝,同僚との議論，上司との面談，研修・勉強会への参加など，様々な場面において，対立や意見の食い違いといった問題が発生することがある．

それらの場面を分析すると，以下のような背景が問題発生の起因となっていることが多いと考える．

- 考え方が違う
- 持っている前提が違う
- 期待値が違う
- 同じ目的でも選ぶ手段が違う
- 同じことでも表現が違う

いずれにも共通するのが＜違い＞である．

2. 問題

関係者間でのあらゆる＜違い＞を捉えることにより，問題を解決できる可能性や，あらかじめ発生を予測して対処できる可能性があると考える．

よく言われる対策として「相手の立場になって考える」「相手の視点で考える」といった対策もある．これらの対策をとろうとした場合，そもそもコンテキストが大きくかけ離れている・全体像が見通せないなど「相手の立場が把握しきれない状況にある」場合も多い．また「相手の立場にフォーカスしすぎて自身の意見や考えを見失ってしまう」という別の問題が発生してしまうこともある．

本手法においては,シンプルに限られたスコープ内での＜違い＞にフォーカスすることにより，自身の意見や考えを見失うことなく他者の意見や考えにも向き合うことを推奨する．相手の意見や考えを捉えられれば「相手の立場」といった踏み込んだところまで思考できない状況においても手軽に実践可能であるため，より広範囲に適用可能であると考える．

＜違い＞を捉えることを楽しめるようになれば，様々な問題に向き合いやすくなるが，ワークショップの目的設定においては人の持つ特性も考慮することとした．

人の持つ特性（「ストレングスファインダー」による「34の強み」[1]）の中には，「個別化」のように「人がそれぞれ違うこと」を前提とするパターンと，「公平性」のように「人が平等であること」を前提とするパターンとがある．前者の場合は＜違い＞があることを楽しめる傾向にあるが，後者は＜違い＞があることを楽しみにくい傾向がある．

そこで「＜違い＞を楽しむ」という目的設定ではなく，「＜違い＞を捉える」ことを意識的に行うワークショップのコンテンツを考案することとした．

3. 解決策・工夫した点

コンテンツのワーク部分は大きく 2 部構成とした．

前半は＜違い＞を捉えるワーク，後半は＜違い＞を活用するワークである．

またワークの前後に，「動機づけ」と「ふりかえり」を設けることにより学びがより活かされるように工夫した．

3.1. ＜違い＞を捉えるワークにおける工夫点

参加者によって異なる意見を簡単に出せるように「写真の人物にセリフをつける」お題を準備した．「写真の人物」は参加者の 1/3～半数程度が知っている人物を採用し，コンテキストの＜違い＞が出るように考慮した．

各々の意見を出した後，3 名のチームを構成して「どういうメンバだと思ったか？」「どういう感情に見えたか？」「どうしてそのセリフにしたのか？」といった観点を与え，「共通点」を探して書き出してもらった．その後，メインのワークとなる＜違い＞を探して書き出してもらった．

「共通点」を先に考える構成としたのは，普段＜違い＞を探すことに慣れていない参加者の思考をフォローする役割がある．更に「共通点」との対比も一つの観点とできることで，より多くの＜違い＞を見つけやすくした．

最後にワークを通した「気づき」を共有してもらい，＜違い＞を捉えることをふりかえってもらった．

3.2. ＜違い＞を活用するワークにおける工夫点

班を構成する 6 名をテストチームに見立て，自動運転車のテスト設計を行う場面設定を提示し，個人の意見を出し合ってチームの最終結論を導きだすことをゴールとした．個人の意見を考えて書き出す際に「個人で選択した結論」と合わせて「選択の理由」「選択に至った背景（過去の経験など）」の欄があるフォーマットを用意することで，＜違い＞を捉えやすくした．

最後にワークを通して「＜違い＞を捉えることが役に立ったか？」「どんな理由でどんなことが起きたか？」をふりかえってもらい，各自の学びの定着を図った．

3.3. ワーク構成における工夫点

ワーク前の動機づけとして，実際にあった事例を4パターン紹介することで参加者の近しい事例との結びつけを行ってもらった．

ワーク後の動機づけとして，初回実施時は，本セッションをワークショップ全体の前半に配置し， WACATE 全体を＜違い＞を捉える実践の場として活用しよう！と提言した．2 回目の単独実施時は，活用の段階が各自に委ねられてしまう．そこで「今日の学びをいつ活かす」という問いかけを行い，各自の活かす時期を班内で共有してもらい，活用イメージの定着と活用宣言を行ってもらった．

また，勉強会参加者によくある「学んだことを業務では使えない」「自組織ではできない」といったところで思考を止めてしまわずに，＜違い＞を捉え＜違い＞に対するテーラリングを検討する」ことにより，活かせる場面が増やせる可能性も提言した．

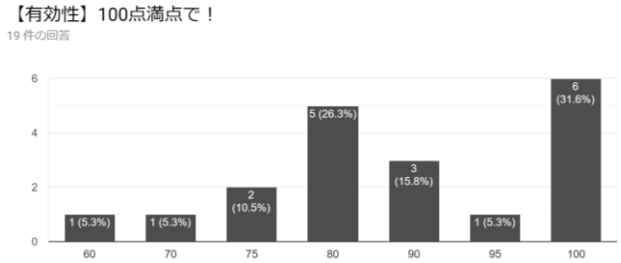
4. 効果

初回の参加者アンケートでは， 9 割以上から「内容を理解できた」「講義レベル・時間がちょうどよかった」と好意的な回答を得られた．一方で，2 割程度であるが「演習レベルが難しい・時間が短い」という回答もあったので,さらなる改善の余地があると考えられる．

また，「前提を疑うことも必要」「個人の経験がアウトプットに強く結びついている」「コンテキストで変わる」といった実感をもった参加者がいた．

後日，自職場にコンテンツ自体を持ち帰り，自職場のメンバで演習含めて実施してくれた参加者もいた．

2 回目の参加者アンケートでは，全員が有効性に 100 点満点で 60 点以上を付け，本手法の有効性が示せた．



また，参加者全員が当日から 3 週間以内に学びを活かす機会が持てそうと回答した．ワーク構成における工夫により，個別開催での 1 時間半のワークでも持ち帰ってもらえることが確認できた．

当初の目的として「楽しむ」はスコープ外としていたが「楽しかった」という感想も複数あった．「共通点」を探す方が難しかったという感想もあったことから，より＜違い＞へのフォーカスが強まった結果の感想と思われる．

「仕事だけでなくて普段人と意見が合わないときにも、自分の気持ちをポジティブに持てそう」「苦手な人の＜違い＞を捉えてポジティブに変換できればいいと思った」といった前向きな反応もあり，目的を上回る効果が得られたと考えられる．

5. まとめと今後の課題

＜違い＞を捉えることにフォーカスすることで様々な問題の解決や発生を予測した対処が行え，課題に向き合うことができると考えた．それらを実践する土台となる練習の場として，体感できるワークショップを構築し実践し，以下のいずれの形式でも活用可能であることがわかった．

・数日間の研修の導入部に配置する 1 セッション

・1 時間半程度の個別テーマ研修ワークショップ

今後も様々な形態の中で実施を繰り返しながら，適用範囲の拡大も試みていきたい．また「ソフトウェアテストに関わるエンジニア」に限定して開催してきたが，参加者層に応じてコンテンツの演習に用いるお題を検討しなおすことで，開発エンジニア向けの開催をはじめ，幅広い層に向けて適用していければとも考えている．

参考文献

[1] マーカス・バッキンガム，ドナルド・O. クリフトン，さあ、才能(じぶん)に目覚めよう―あなたの 5 つの強みを見出し、活かす，日本経済新聞社

なぜなぜ分析とシステム理論に基づくSTAMP/CASTの事例による比較
～ソフトウェアメンテナンスプロジェクトでの問題の分析事例に基づく比較～

日下部　茂
長崎県立大学
kusakabe@sun.ac.jp

三輪　東
SCSK株式会社
azuma.miwa@scsk.jp

要旨

筆者らは，ソフトウェアメンテナンスプロジェクトで問題が発生した事例で，システム理論に基づく事故の説明モデルSTAMPをベースとした分析手法CASTを用いることで失いかけた心理的安全を回復できた．この組織では通常は問題分析においてなぜなぜ分析をベースとした分析を行っており，該当事例でも当初なぜなぜ分析をベースとした分析を行ったが，分析に限界があった．本稿では，システム理論に基づくSTAMP/CASTとなぜなぜ分析の比較について，分析結果に違いが出た上記事例をベースに比較と考察を行う．

1. はじめに

我々はソフトウェアメンテナンスプロジェクトで問題が発生した事例で，システム理論に基づく事故モデルSTAMP (Systems-Theoretic Accident Model & Process)をベースとした分析手法CAST(Causal Analysis using System Theory)[1]を用いることで，失いかけた心理的安全を回復できた例をいくつかの観点で紹介した[2][3][4]．その際，どのようにすれば同様の分析を皆が出来るようになるのかという質問や，必ずしも皆が使えるとは限らないことや適用可能性、使い分けに関する質問などを受けた．現時点では我々もそれらの質問に対する一般的な解は持っていないが，本稿では，部分的な解として，STAMP/CASTの分析と，比較的普及していると推測されるなぜなぜ分析との比較を事例ベースに行う．上記の事例を振り返りながら，分析の過程に焦点を当て比較する．

分析対象のプロジェクトはソフトウェアメンテナンスの元請けと委託先のメンバで実施されたものである．対象事例のドメインは金融関係で，多様で複雑な業務知識を要し，サブシステムごとにチームで担当している．それぞれのプロジェクトは，関係するサブシステムのチームによって実施する．プロジェクトはそれぞれ規模や期間が異なる上，複数が多重並行的に実行されることが多いため，人材配置などに問題が生じやすい．そのため，組織的に人材の定着と多能工化を推進する育成活動にも力を入れている．そのような背景の下，対象事例では，図1の破線で囲んだ箇所を，委託先のメンバが未経験のサブシステムのサブチームリーダを担当していた．

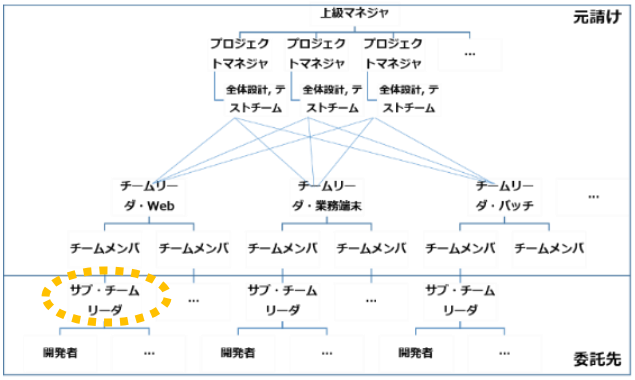


図 1 プロジェクト・チーム構成概要

当該プロジェクトでは，このような例外的な状況に元請け側が配慮し，入念に影響調査を行った上で追加・修正の対象を明確化し委託先に仕事を依頼した．しかしながら，指定範囲外での想定されていない修正が行われた．しかもその修正部分が検証されずに委託先から納品され，欠陥が運用直前の移行フェーズで見つかるインシデントが発生した．このような事態を受け，最初の調査はなぜなぜ分析ベースで行われ，委託先のルール違反，スキルや知識不足が根本原因という結論が報告された．個人非難につながるような結論により，元請けから委託先への信頼関係は崩れ，プロジェクト組織内での心理的安全は損なわれてしまった．しかしながら，システム理論に基づくSTAMP/CASTでモデル化と分析を行った結果，元請け側も含めた関係者の間での想定と実際のギャップに備えた望ましいチェックができてなかったという組織レベルの課題が明らかになった．これにより個人レベルの非難を回避でき，元請けと委託先との間の信頼関係を取り戻しチームの心理的安全を回復させることが出来た．

この事例において，上級マネージャであった著者の一人はなぜなぜ分析的なヒアリングや深堀から開始し，限界を感じたところで STAMP/CAST を導入するという，結果的にはハイブリッドなアプローチで，最終的な結果を導いている．本稿は，このような経緯も踏まえ，分析の結果よりも過程に焦点を当てて振り返りを行いながら，以下の構成で STAMP/CAST となぜなぜ分析の比較を論じる．2 章はなぜなぜ分析と STAMP/ CAST を概説する．3 章は，事例でのポイントとなった心理的安全と暗黙知を説明する． 4 章では，事例での分析過程を振り返りながら，なぜなぜ分析と CAST の比較を行う．5 章ではまとめを行う．

2. 分析法について

ここではなぜなぜ分析と STAMP/CAST の概説を行う．

2.1. なぜなぜ分析

なぜなぜ分析は，複数の要因がありえる現場で発生する問題の，真の原因を迅速に現場で発見して解決することを目的に，トヨタ生産技術の枠組みで提唱された[1]．Root Cause Analysis (RCA)手法のひとつとされ，問題の真の原因に到達するために，問題が発生した理由についての質問を 5 回以上続けて追究する．現場，現物，現実調査による三現主義のもと，推定される要因ではなく事実にもとづいて原因を見つけ，存在しない要因や不明の要因は検討の対象外とする．当初は人的・組織的レベルでの対処が必要な範囲でなく，現場レベルの問題解決手法として提唱されたが現在はより広く用いられている．

なぜなぜ分析は，現場での実際の問題解決が重視されており，著者の知る限り，おおよそ以下の手順を基本にそれぞれにテーラリングされ適用されている．

- まず，分析の対象とする問題と分析の目的を明確化する．また，後述する深堀の過程で全体像を失わないよう初期の段階で全体像を把握しておく
- 時系列に経緯をさかのぼり，最初の「なぜ」として問題事象に至った経緯を分析し要因を列挙する．要因はひとつだけとは限らず，事象に対して論理的なつながりがなければならない．
- 次に，一回目の「なぜ」で得た要因ごとに，それが発生するに至った要因を列挙する．1 回目と同様，要因はひとつだけとは限らず，論理的なつながりがなければならない．
- 同様にして，3 回目の「なぜ」，4 回目の「なぜ」を繰り返していく．

このようななぜなぜ分析の実施には様々な注意事項が提唱され注意には原因を個人に帰さないことも含まれているが，対象事例でのなぜなぜ分析ベースの結論では個人非難のようなものになってしまった．

2.2. STAMP/CAST

STAMP は MIT の Nancy Leveson 教授によって提唱されたもので，従来の解析的還元論や信頼性理論ではなくシステム理論に基づき，システムを構成するコンポーネント間の相互作用に着目した事故モデルである．STAMP のモデルは，安全制約，階層的なコントロールストラクチャ，プロセスモデルという三つの基本要素で構成されている．

- コントロールストラクチャ:システムの構成要素間の構造と，相互作用を表したもの
- プロセスモデル:コントロールする側がその対象プロセスをコントロールするアルゴリズムと対象プロセスを(抽象化して) 表現したもの
- 安全制約:安全のために守るべき制約

STAMP では，コントロールストラクチャとプロセスモデルに対して，システムの安全制約が正しく適用されているかどうかに着目する．STAMP でのコントロールストラクチャとプロセスモデルの基本的な構造を図 2 に示す．

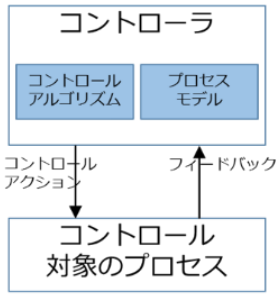


図 2 STAMP モデルの基本構成

STAMP 自身はアクシデントを説明するモデルであり分析技法ではないが，STAMP をベースとして，解析の道具立てやプロセスが複数提案されている．STAMP を用いた安全制約の分析は，作業成果物に対してだけでなく，そのような成果物の開発プロセスや運用プロセスに対しても行うことができ，今回の事例のようなソフトウェアのメンテナンスプロジェクトでも用いることができる．事故分析手法 CAST 以外にも代表的なものとしてハザード分析手法 STPA (System-Theoretic Process Analysis)，セキュリティ向け STPA-Sec (STPA for Security)といった分析法が提案されている．STAMP は安全工学を対象として

提唱され，分析ではアクシデントやハザードを明確にする必要がある．しかしながら安全工学の技法を出来るだけ広く適用する意図で分析するアクシデントも慣習的な安全観点だけでなくミッションの未達なども含めた損失につながりうるイベントとなっている．関連して，ハザードは，環境のある最悪な条件の集合と重なるとアクシデントになり得るシステムの状態もしくは条件の集合，とされている．

今回なぜなぜ分析と比較する CAST 分析のステップは次のようなものである．

1. 損失に関与したシステムレベルのハザードを識別
2. そのハザードに関する安全制約やシステム要件を識別
3. そのハザードを制御するために存在した制御構造を識別
4. 損失につながった近接イベントを究明
5. 損失を物理システムレベルで分析(今回は組織の末端の個人行動等で適宜読み替えて分析)
   (ア) 関与した物理的な制御や設備を識別
   (イ) この事故の防止のための全ての物理的な安全要件と制約を識別
   (ウ) 物理的な装置のあらゆる故障もしくは不適切な制御を識別
   (エ) 物理的な故障もしくは不適切な制御を説明するコンテキスト要因を識別
6. 引き続く各高位のレベルが，どう，なぜ，現レベルの不適切な制御に寄与したか解明するため，高位のレベルを分析
   (ア) 次の高位レベルの制御に対する，安全関連の責務を識別
   (イ) 非安全な決定と制御アクションを識別
   (ウ) 非安全な決定と制御アクションを説明するプロセスモデルの欠陥(誤信)を識別
   (エ) その時点でなぜその振舞いが適切に思えたか説明するコンテキスト要因を識別
7. 損失に寄与した全体的な調整ややりとりを確認
8. 損失に関連したシステムと安全制御構造のダイナミクスや変化，継時的な弱化を割り出す
9. 改善勧告を出す

3. 心理的安全性と暗黙知

3.1. 心理的安全性

心理的安全性は，チームの各メンバがそのチームに対し，気兼ねなく発言でき，本来の自分を安心してさらけ出せると感じられるような環境や場の状態をいう[6]．例えば Google の Project Aristotle の結果が示すように，心理的安全性はソフトウェアプロジェクトの成功にとって重要とされている[7]．チームに明確な目標を設定するなど，他にも重要な基準があるものの，Google の Project Aristotle の結果では，生産性の高いチームは心理的安全性が最も重要であるとされている．また，比較的先進的なソフトウェアプロセスである，モダンアジャイルプロセスでは，心理的安全性もアジャイルプロセスの最も重要な側面の 1 つと見なされており，推進者の中には，心理的安全性に関連する活動を特に「Anzeneering」と呼ぶものもいる[8]．

このようにソフトウェア開発のプロジェクトやプロセスでは心理的安全が重要とされており，今回のプロジェクトの上級マネージャにとっても関心事の一つであった．Google のガイドでは以下の 7 つの質問の中で，ポジティブな回答が多ければ，メンバがチームの中で心理的安全性を高く感じているとされている[9]．

1. チームの中でミスをすると，たいてい非難される．
2. チームのメンバは課題や難しい問題を指摘し合える．
3. チームのメンバは，自分と異なるということを理由に他者を拒絶することがある．
4. チームに対してリスクのある行動をしても安全である．
5. チームの他のメンバに助けを求めることは難しい．
6. チームメンバは誰も，自分の仕事を意図的におとしめるような行動をしない．
7. チームメンバと仕事をするとき，自分のスキルと才能が尊重され，活かされていると感じる

例えば，インシデントの調査で個人非難のような結論が出てしまうとネガティブな意味で上記第 1 項目から当てはまり心理的安全が損なわれる．

3.2. 暗黙知

上級マネージャは経験から暗黙知に関してトラブルの要因となりえる問題点が二つあると考えていた．一つは暗黙知のそのものの問題，もう一つが暗黙知とアジリティ的な価値観が合わさった複雑なスキル不足の問題である．ここでの暗黙知は「もともとのチームではあたりまえであるが，新規参入者には理解しにくい知の集合体と現状．ルール・規律，言語パッケージやフレームワーク構造，作法・判断基準など，普段は言及されないけれど存在するもの全般を含む．」とする．例えば，仕様書に明示的に記載されている情報は暗黙知には含まないが，その設計思想や，様々な実装者が修正してきたソースコードの経緯など，知の集合体を背景に持つファイル自身と，そのファイルを取り扱う前提となる様々な作法や期待される行動などの全般を指す．

こういった暗黙知は長年チームで蓄積されたノウハウや認知的側面[10]が多いため問題は単純ではない．SECI モデル[11]のように歴史と共に変化・進化し，プロセスや成果物フォーマットなどにちりばめられつつ，その現場の慣習や文化になったものは，可視化が難しい． 7S の「ソフトの 4S（Shared value , Style , Staff , Skill）」[12] に染み渡った状況であり，プロセス資産，個人のスキルや意志・行動，文化といった組織の環境要因に相互依存しながら混じり合い，境界線を引くことも難しい．当事者でも，その構造を端的に説明することは難しい場合も多い．新規参入者の場合，既存メンバすら言語化出来ないものを簡単に理解できるはずもなく，その対処は容易でない．

4. モデリングと分析の過程

当時の分析経緯を可能な限りふりかえりながら，事例ベースで STAMP/CAST 分析となぜなぜ分析の比較を行う．心理的安全が脅かされる結論が出た後の上級マネージャの再調査でも，当初はなぜなぜ分析ベースのアプローチから着手した．しかしながら，問題のきっかけとなった逸脱的振る舞いは単純ミスではなく，未経験メンバと暗黙知に起因する部分がありそうだと分かった段階で STAMP/CAST の採用を検討した．

4.1. なぜなぜ分析の深堀とヒアリング

上級マネージャは，まず，次のような視点(以降，視点一と呼ぶ)で元請け側のチームリーダにヒアリングを行った．(WHY/ANS の下線付き強調表示部は詳細を後述)

表 1 視点一によるなぜなぜ分析

WHY:	なぜ，想定外のソースコードに不具合が混入したのか？
ANS:	委託先のミスによる
WHY:	なぜ委託先はミスしたのか？
ANS:	ルール通り行ってくれなかった
WHY:	モラルハザードでも起きているのか？
ANS:	そんなことは無く，今回の件は全員が残念に感じている．

この視点での掘り下げは難しかったため，次のように視点を変えて(以降，視点二と呼ぶ)再度元請け側のチームリーダにヒアリングを行った．

表 2 視点二によるなぜなぜ分析

WHY:	何か他の事情はないのか？
ANS:	未経験のメンバがいて不慣れだったのは事実
WHY:	スキルが大幅に足りなかったということか?
ANS:	他のチームではリーダー経験があり，スキル不足とは言い難い
WHY:	このチームでは初仕事ということで，仕事の仕方を理解していないということか
ANS:	その通り．
WHY:	すると，それをフォローするのは元請け側の責任ではないか？
ANS:	こちらも対象範囲を限定して依頼することで，最大限考慮した．
WHY:	対象範囲を間違ったか否かをチェックするのも元請け側の役目ではないのか？
ANS:	初作業だったので，普段は委託側にお願いしているソースへの影響調査からテスト範囲まですべてこちらで確認したうえで，事故が起きないように「これだけやってください」とお願いした．それを逸脱されたら，こちらもフォローのしようがない．
WHY:	事前に相談や確認は無かったのか？
ANS:	なかった
WHY:	いつも勝手にやるのか？
ANS_1:	まさか，そんなはずはない．こちらが気づかないケースで指示以外のソースを修正すべきケースも勿論ある．その場合は当然，報告・確認があるし，その為に会議をやっている．その会議で相談・確認もなく，「これだけやって」と指示した範囲外を修正されても，こちらは対処のしようがない．

ここで，対象を委託側のリーダ(当事者の上司)に変え，次のような視点(以降，視点三)でヒアリングを行った．

表 3 視点三によるなぜなぜ分析

WHY:	限定した範囲を逸脱したのはなぜ？
ANS:	バックログで該当箇所があり，いつも通り修正をかけた
WHY:	なぜ，事前に相談・確認が無かったのか？
ANS:	いつもなら確認することが漏れてしまった，

	プロセスの問題かもしれない
WHY:	状況を確認する会議体は定期的に行われており，プロセスとして存在していると思われるが，違うか？
ANS:	確かにプロセスとして存在している．やはり，不慣れだったと思われる．
WHY_2:	スキル不足ということか？
ANS:	それは否定できない．

上記のようなやり取りが行われ，対話上では深堀ができているようにも見える．実際の現場で通常行われる深堀はこういった流れであり，表 2 や表 3 のように深堀が STOP してしまうと表 2 の *ANS_1* や表 3 の *WHY_2* で対話を終えることになり，委託先の指示逸脱行為，スキルや知識不足が根本原因と片づけてしまいたくなる．

4.2. STAMP/CAST での再分析

前述の視点二と視点三について図 2 のような STAMP の基本構造を念頭にモデル化と CAST での再分析を試みた．CAST では節 2.2 で紹介した手順 3 にあるように，安全制約を満たす全体的な制御構造を考える必要がある．今回の場合，必ずしも手順通りでない試行錯誤の分析での構築であるが，制御構造は図 3 のようになる(ただし，プロセスモデル等の詳細は略してある)．

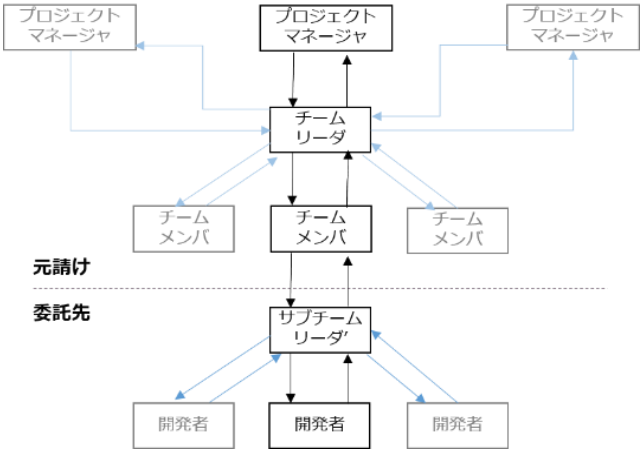


図 3 該当事例でのプロセスの制御構造の概要

STAMP でのモデル表現のために，視点二は元請けから委託先への制御，視点三は委託先内での制御に対応付けて考えると次のようになる．

表 4 視点二での STAMP モデリングの試行

コントローラ	仕事の進め方の理解が不十分と思われた → 範囲を限定して指示
被コントローラ	良くするつもりで指示範囲を逸脱
コントローラ	実際に範囲が限定されているかの確認は，会議体で追加作業の報告が無かったことで問題なしとしてしまった

表 5 視点三での STAMP モデリングの試行

コントローラ	限定された範囲で作業指示をした
被コントローラ	指示された範囲で修正を試みたが，バックログがあったため，指示範囲以外の作業も必要となった
コントローラ	限定された範囲外での作業に気付いたが，被コントローラからの説明でそれが適切だと，納得してしまった

この段階で，視点二と視点三のモデル化の結果をつなげることを試みると，視点三では範囲外の修正について適切と納得してしまっており，視点二のコントローラが想定する「会議体で追加作業の報告」につながらないことに気づく．より上位である視点二では指示した限定範囲が正しいという前提に基づいているが，下位側の視点三では，そもそも範囲の限定自体が不適切と判断したことを表しており，このつながりは構造上で不整合が発生している．何か，大事な要素が抜け落ちていることが分かる．

5. 考察

5.1. 比較概要

なぜなぜ分析は直線的なイベントチェーンやイベントツリー的な分析が当てはまる場合には有効だが，今回のような単純ではない暗黙知が関係する文脈依存の問題では必ずしも有効ではないと思われる．STAMP/STPA のようにモデル要素として制御構造やプロセスモデルが含まれている方がそのような問題でも分析しやすいと思われる．しかしながら，使い慣れていない場合，STAMP /STPA での分析は，着手時点から困難を感じる可能性がある．今回は，なぜなぜ分析をベースにインタビューと深堀を行いながら，STAMP でのモデリングを試行した．制御構造の階層をふまえてハザードを分析するとともに，問題のある構造を探し出すという段階を踏んでいった.その

経験をもとに比較の概要をまとめると以下となる.

表 6 なぜなぜ分析と STAMP/CAST の比較概要

	なぜなぜ	STAMP/CAST
着手しやすさ	易しい	難しい
分析スピード	早い	時間がかかる
制御構造	実施者が意識してつなげる	モデル要素に含まれる
制御決定の背景(プロセスモデル)	実施者が意識して抽出	モデル要素に含まれる

なぜなぜ分析は特にモデルなどを前提とせずに実施できるため着手しやすい. しかしながら実際の分析は, 実施者の知識やスキルに依存した属人性が高くなると考える. STAMP はコントロールの問題に焦点を当てた事故モデルであり, イベントチェーン的なものでない複雑な問題でも広義のコントロールの問題に帰着できる場合は有効なモデリングと感じた. 分析に効果的なモデル要素が含まれているため, STAMP/CAST 分析は強力であり, いったんマスターできればある程度低い属人性で実施できると感じた. しかしながら, STAMP 固有のモデリングに加え, 適切な抽象化と具象化といった一般的な意味でのモデリングに関する能力やドメイン知識が必要であると感じた.

5.2. 事例のなぜなぜ分析での欠落に関する考察

なぜなぜ分析の深堀は, 掘り下げていく性質上, 最後の対話に着目されやすい. 今回の件で言えば, 表 2 の *ANS_1* や表 3 の *WHY_2* のようなものが原因として取り上げられやすい. しかしながら, 表 2 や表 3 中に下線付き太字で強調したような, やり取りの途中で出て来たハザード分析候補(表 7 参照)への考慮が欠落しがちである.

表 7 ハザード分析の候補

WHY (表 2)	このチームでは初仕事ということで, 仕事の仕方を理解していないということか
ANS (表 3)	いつもなら確認することが漏れてしまった, プロセスの問題かもしれない
WHY (表 3)	状況を確認する会議体は定期的に行われておりプロセスに存在していると思われるが, 違うか?
ANS (表 3)	確かにプロセスとして存在している. やはり, 不慣れだったと思われる.

上記のハザード分析候補を STAMP で書き表すと以下のようになり, コントローラと非コントローラの間に食い違いがあり, プロセスの実践上問題があることが分かる.

表 8 コントローラと非コントローラの不一致

コントローラ	確認するプロセスが存在し, その遵守は当然. 不明点は質問するはず.
被コントローラ	確認するプロセスには不慣れで理解は不十分. 不明点を適切に言語化して適切な人に聞けるとも限らない

残念ながら, なぜなぜ分析の対話の中では, プロセスのルールがあれば当然守るべきという前提で対話が進んでしまっている. 今回は発注側と受注側の力関係や権威勾配なども背景にあるかもしれない. そもそもプロセスが存在してもその実践の仕方について理解していない・分かっていないという可能性についての考慮が消えてしまっており, より良い深堀になっているとは言い難い.

5.3. 実際の現場での活用について

現場ではコストや時間に制限があるので, 着手の障壁や実施コストが相対的に低いなぜなぜ分析から始めるのが現実的と思われる. しかし, なぜなぜ分析は最後の対話に着目されやすく, それ以前の(5W まで深掘なら)1W～4W の内容を適切に反映しない結論にする可能性がある. 実施者が最後の対話以外の関連性を適切に取り扱うスキルなどが必要と思われる. このようなある種属人的なスキルを補う意味で, 分析に必要な要素をもとからモデルに含む STAMP/CAST と組み合わせることは現実的な解と思われる. STAMP/CAST のように制約を保つ制御構造とプロセスモデルを対象に分析を行うことで, なぜなぜ分析で欠けている点を補完できると考える.

単純な線形思考の場合, 先に答えを仮定し, その答えに合う情報で肉付けして終わらせてしまうといった, 単純化や確証バイアスが生じやすい可能性もある. STAMP では, 制約とそれを保つ制御ループ構造があるので, ある種のごまかしが難しいと感じた. 仮定がある側面だけから導き出されたものだとすると, フィードバックサイクルの中で矛盾を分析しやすく, それがハザードと関連づくケースが多かった. それらを分析すると新たな矛盾が出るといった繰り返しで全体がつながるようになった.

STAMP/CAST のような分析手法は非常に強力と感じる一方, 誰もが簡単に使えるものでは無いと感じた. 分析

手法が明快に定義され，他で成功事例があったとしても，実際の対象事案に関するさまざまな構成要素や文脈を適切に取り扱える論理な思考や抽象化能力などがないと，簡単には活用できないと思われる．数名にワークショップに参加してもらったが，結局「よく分からなかった」で，実際にモデリングや分析ができる人材は増えなかった．

今回の事例でも最初のモデリングは，表記だけみればコントローラ，被コントロールプロセス，コントロールアクション，フィードバックデータといったSTAMPでのモデル表記ができたものの，制約条件やハザードと種々のモデル要素の関係は当初明確ではなかった．STAMP の良さは，人が関係するプロセス上の事故であっても，誰が・何が悪いのかではなく，なぜ・どのようにコントロール出来なかったのかを明らかに出来ることと感じた．それを明らかにできるまで試行錯誤を繰り返せたのが STAMP の良いところだと感じている．構造的には書けているように見えても，そこで終わらせられず，なぜ・どのようにコントロール出来ていないのかまで明らかにしようとすると，必然的に様々な依存関係まで明らかにする必要があり，そこまで書かないと納得できるものが作成できないのが利点だと感じた．今回のようなソフトウェア開発だけでなく，医療も含め人の振る舞いが関係する事故に対して，対策志向型で取り組み，エラーの背後要因[13]を明らかにするには高いポテンシャルを持つ良い手法と考える．

5.4. 発展的活用に向けて

STAMP は，様々な安全工学の関連領域について議論され提唱された，セーフウェア[14]のコンセプトを具現化するための手法とされている．そのため STAMP の活用においてもそのような関連領域の知識があるとさらに活用の可能性が広がると考える．

例えば，認知心理学の認知情報処理特性の観点から，人の行動を三つのレイヤに分けた Rasmussen の SRK モデルを STAMP/STPA に取り入れた事例もある[15]．今回の事例で重要な切り口であった暗黙知の問題も，STAMP のプロセスモデルの，特にメンタルモデルを認知心理学や社会心理学のモデルを関連付けることができれば，心理学の知見を活用して分析できると考える．

このような関連領域の知見は，なぜなぜ分析などの他の分析法でも活用は不可能ではないと考えるが，分析に効果的なモデル要素を持つ STAMP でのモデリングと分析では、関連領域の系統的な対応付けと活用がより容易になると考える．

6. おわりに

ソフトウェアメンテナンスプロジェクトで問題が発生した事例で，システム理論的な事故説明のモデル STAMP をベースとした分析手法 CAST を用いることで失いかけた心理的安全を回復できた報告に対し，手法の一般性，適用可能性，適用の再現性，使い分けなどの質問を受けていた．本稿では，それらの問いに対する部分的な解として，既報告の事例を分析結果よりも分析過程に焦点を当て振り返りながら，事例ベースで STAMP/CAST と比較的普及していると推測されるなぜなぜ分析を比較した．

主観が入っている可能性があるが，因果関係が直線的もしくはそれに近い場合は，なぜなぜ分析の方が低コストで実施可能であり，暗黙知を紐解くような複雑な要因間の関係がある場合は，相対的に着手の障壁やコストが高くなるが，システム理論に基づく STAMP/CAST を用いると効果的な分析ができるとの感触を持った．教条主義的な使い分けでなく，なぜなぜ分析をベースに分析を開始し限界を感じた場合に STAMP のモデリングを行うといった使い方も有効と考えられる．

STAMP のような特定のモデリング方式に基づくモデルの構築や分析は，分析に効果的なモデル要素を系統的に含むことができる一方で，学習のコストやバリアが存在する．しかしながら，システム的な観点で多層的に関連領域の知見を系統的に活用し強力な分析ができる可能性を感じており，引き続き関連する取り組みを継続する予定である．

参考文献

[1] Nancy Leveson, Engineering a Safer World, MIT press, 2012.
[2] 日下部 茂，三輪 東，ソフトウェアメンテナンスプロジェクトチームの心理的安全確保に向けたCAST適用の事例，第3回 STAMP ワークショップ，2018
[3] 三輪 東，日下部 茂，ソフトウェアメンテナンスプロジェクトチームの心理的安全確保に向けたCAST適用の事例，ソフトウェア信頼性研究会 第 14 回ワークショップ（FORCE2018），2018
[4] 日下部 茂，三輪 東,プロジェクトチームの心理的安全のための STAMP/CAST による事後分析，プロジェクトマネジメント学会 2019 年度春季研究発表大会予稿集，pp.335-340，2019
[5] 大野耐一，トヨタ生産方式:脱規模の経営をめざして，ダイヤモンド社，1978

[6] Amy Edmondson. Psychological safety and learning behavior in work teams, Administrative Science Quarterly, 44:2, 350-383, 1999
[7] Charles Duhigg, What Google Learned from Its Quest to Build the Perfect Team. The New York Times Magazine. https://www.nytimes.com/2016/02/28/magazine/what-google-learned-from-its-quest-to-build-the-perfect-team.html (アクセス 2019-02-25)
[8] Modern Agile. http://modernagile.org/ (アクセス 2019-02-25)
[9] Google re:Work.「効果的なチームとは何か」を知る, https://rework.withgoogle.com/guides/understanding-team-effectiveness/steps/introduction/ (アクセス 2019-02-25)
[10] 知識創造企業, 野中郁次郎・竹内弘高, 梅本勝博訳, 東洋経済新報社, ISBN978-4-492-52081-9
[11] SECI model of knowledge dimensions, https://en.wikipedia.org/wiki/SECI_model_of_knowledge_dimensions, (アクセス 2018-03-12)
[12] 7S,https://mba.globis.ac.jp/about_mba/glossary/detail-12513.html,（アクセス 2018-03-12）
[13] 石橋明, ヒューマンファクターとエラー対策, J. Natl. Inst. Public Health. 51(4):2002
[14] ナンシー・レブソン著, 松原友夫監訳, セーフウェア, 翔泳社, 2009
[15] Nobuyuki Hoshino, Applying Human Mental Model to STAMP/STPA, 3rd MIT STAMP Workshop, 2014

IoT システム開発におけるエッジデバイスソフトウェア開発の難しさ

阪井　誠
株式会社 SRA
sakai@sra.cojp

要旨

IoT システムのエッジデバイスとして Rashberry pi を用いた IoT システムの開発者にヒアリングを行い，12 の具体的な難しさを確認した．

エッジデバイスソフトウェア開発はその構造の複雑さから，一般のアプリケーションの難しさとは異なるものであった．本報告のような知見の蓄積が，IoT システムの開発をより安全で安定したものにしてくれると考えられる．

1. はじめに

IoT(Internet of Things)はモノのインターネット)と訳され，IDC は，IP 接続による通信を，人の介在なしにローカルまたはグローバルに行うことができる識別可能なエッジデバイスからなるネットワークのネットワークと定義されている[1]. IoT デバイスが登場してきたことで，さまざまなモノがインターネットに接続されるようになり，これまで数値が計測できなかった領域のデータも取得可能になった[2].

エッジデバイスとは，通信ネットワークを他の通信ネットワークと接続するために使われる機器の総称とされ[3]，エッジデバイスでデータを収集し，クラウド上で処理されることが多い．近年では，Rashberry pi などの小型 Linux 機が実機や試作用のエッジデバイスとして使われることが多くなっている[4].

しかし，通常のソフトウェア開発にはない，エッジデバイスならではの難しさがある．そこで，本報告では，エッジデバイスソフトウェアの開発経験者にヒアリングし，整理したので報告する．

2. ヒアリング内容

6 名エッジデバイスソフトウェアの開発経験者にインフォーマルにヒアリングし，その内容から難しさを整理した．

ヒアリングの結果，一般のアプリケーションと異なる 12 の難しさが収集でき，以下の 4 つに分類できた．

- ハードウェアそのものの難しさ
- ミドルウェア(OS，言語）の難しさ
- ネットワークの難しさ
- ソフトウェアの難しさ

このようにエッジデバイスソフトウェアの開発には，様々なノウハウが必要である．これらのプロジェクトでは，コミュニケーションを容易にすることで情報を共有し，問題を解決した．

3. おわりに

エッジデバイスソフトウェアの開発には，ノウハウの蓄積が必要であり，本報告のような経験を共有することで，ソフトウェア開発を安全に安定して行うことが可能になる．今後も経験を蓄積・共有するとともに，課題解決を容易にするコミュニケーションについてまとめたい．

なお，本報告は SS2017 の経験論文と対象プロジェクトが一部重複するが[5]，他のプロジェクトを追加した上で，発表済みの開発環境 Node-RED に関する知見を除いた部分について，経験をまとめたものである．

参考文献

[1] EnterpriseZine，国内 IoT 市場におけるソフトウェア／サービス向け支出割合は 2020 年に 6 割に――IDC が予測，http://enterprisezine.jp/article/detail/8057

[2] 袖実樹子，社会を変える IoT，情報処理，Vol.60, No.2, 2019.

[3] NTTPC コミュニケーションズ，用語解説辞典，https://www.nttpc.co.jp/yougo/%E3%82%A8%E3%83%83%E3%82%B8%E3%83%87%E3%83%90%E3%82%A4%E3%82%B9.html

[4] Node-RED UG Japan Osaka, Node-RED UG Osaka 勉強会＆懇親会　vol.1「Osaka キックオフ」，https://node-red-osaka.connpass.com/event/77653/

[5] 阪井誠 Visual 開発ツール Node-RED の導入によるプロセスの変化と考慮点, ss2017

テンソル分解プログラミングの理解支援のための立体パズルの利用

山本 直樹
熊本高等専門学校人間情報システム工学科
naoki@kumamoto-nct.ac.jp

村上 純
熊本高等専門学校人間情報システム工学科
jun@kumamoto-nct.ac.jp

石田 明男
熊本高等専門学校共通教育科
ishida@kumamoto-nct.ac.jp

要旨

テンソル分解の一手法として高次特異値分解(HOSVD)があるが，これは行列の特異値分解(SVD)を高階テンソルの分解に拡張したものである．そのため，HOSVD は SVD と比べアルゴリズムが複雑となる．そこで，我々は，立体パズルを教材に用いた HOSVD アルゴリズムの理解支援の取り組みを行っている．本論文では，立体パズルとしてルービックキューブおよびインスタント・インサニティの 2 つを取り上げ，これらを高階テンソルで表現し，HOSVD アルゴリズムの *n*-モード行列展開を利用して展開図にする原理について述べる．さらに，その展開図のテンソル分解やそのプログラミング教育への応用として，本校学生によるルービックキューブの展開図作成の事例と，主に小中学生を対象として，インスタント・インサニティの展開図の理解度について調査した事例についても言及する．

1. はじめに

ビッグデータのような多次元データを扱いやすくするための低次元化手法にテンソル分解がある．テンソル分解におけるテンソルとは多次元配列で表現される高階テンソルのことを指し，多次元データを低次元の積や和に分解する．テンソル分解としては，Tucker 分解と CP(canonical or parallel factor)分解などがあり，これらの応用として，Fuzzy モデリング，信号処理，画像処理，画像分類，生体信号分析，テキストマイニング，ソーシャルネットワークおよび Web ハイパーリンクの分析等に用いられている[1].

我々は，これまでテンソル分解の数値計算法に関する研究[2, 3]や，それを用いた医療データ分析に関する研究[4, 5]を行ってきた．Tucker 分解を計算する手法としてよく利用されるものに高次特異値分解(HOSVD: higher-order SVD) [6]があるが，これは行列の特異値分解(SVD: singular value decomposition)を 3 階以上の高階テンソルの分解に拡張したものである．したがって，SVD よりもアルゴリズムが複雑となるため，このアルゴリズムの理解支援にも取り組んでいる．理解支援に関する最初の取り組みは，CG を利用したアルゴリズムの可視化[7]であった．次に，立体パズルに着目して，この求解に HOSVD のアルゴリズムを利用することで，アルゴリズムの理解に繋がると考え，研究を行っている[8, 9, 10, 11].

現在は，HOSVD を利用可能な立体パズルの収集を行っているところである．今回，本論文では，よく知られた立体パズルであるルービックキューブ[12]とインスタント・インサニティ[13]を取り上げ，このパズルの展開表現と，それをテンソル分解とそのプログラミングの教育に応用した事例について述べる．

2. 高階テンソル分解アルゴリズムとそのプログラミング教育への応用

2.1. 高階テンソルの概要と定義

高階テンソルとは，1章で述べたように多次元配列のことを指し，1 階テンソルはベクトル，2 階テンソルは行列，3 階テンソルは 3 次元配列にそれぞれ対応する．本論文では，立体パズルを 3 階テンソルおよび 4 階テンソルとして表現する．ここで，図 1 にサイズ$I \times J \times K$の 3 階テンソル$\boldsymbol{\mathcal{A}}$のイメージを示す．いま，$\boldsymbol{\mathcal{A}}$の(i, j, k)要素をa_{ijk}とする

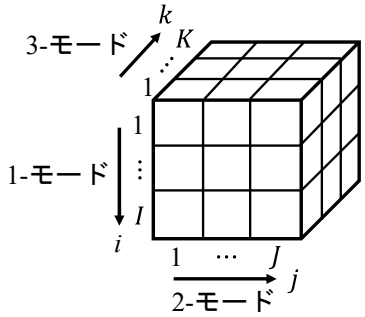


図 1. 3 階テンソルのイメージ

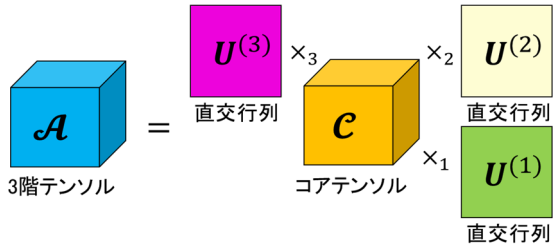


図 2.　3 階テンソルの HOSVD

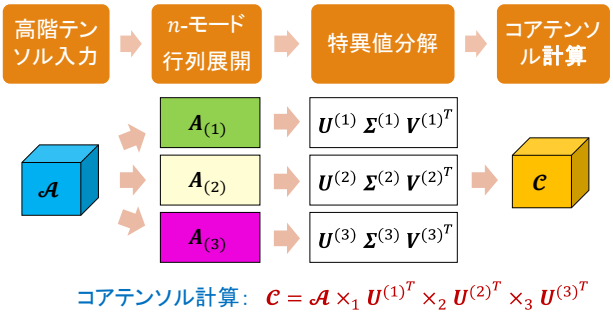


図 3.　3 階テンソルの HOSVD の流れ

と，$\mathcal{A}$は次式で表される．

$$\mathcal{A} = (a_{ijk}),$$
$$(i = 1,2,\dots,I; j = 1,2,\dots,J; k = 1,2,\dots,K) \quad (1)$$

また，図 1 におけるモードとは，図の矢印で示すような高階テンソルの方向を表す．本論文では，縦方向を 1-モード，横方向を 2-モード，奥方向を 3-モードとし，テンソル要素の添字i, j, kとそれぞれ対応させている．

さらに，4 階テンソルは，図 1 に示す 3 階テンソルがベクトル的に配置されたものと考える．すなわち，サイズ$I \times J \times K \times L$の 4 階テンソルを$\mathcal{B}$とすると，$\mathcal{B}$は$\mathcal{A}$と同じサイズの 3 階テンソル$\mathcal{A}_l, (l = 1,2,\dots,L)$を$L$個並べて$\mathcal{B} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_L)$と構成されるものとする．

いま，$\mathcal{B}$の(i, j, k, l)要素をb_{ijkl}とすると，$\mathcal{B} = (b_{ijkl})$,
$(i = 1,2,\dots,I; j = 1,2,\dots,J; k = 1,2,\dots,K; l = 1,2,\dots,L)$
と表され，4 階テンソル要素の添字lが 4-モードに対応する．

2.2. テンソル分解の概要とアルゴリズム

本論文では，立体パズルの展開表現にテンソル分解の一手法である高次特異値分解（HOSVD）のアルゴリズムを利用する．以下に HOSVD の概要とアルゴリズムについて述べる．

図 1 に示す 3 階テンソル$\mathcal{A}$の HOSVD は次式で与えられる．

$$\mathcal{A} = \mathcal{C} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \quad (2)$$

ここで，$U^{(n)}, (n = 1,2,3)$は正規直交行列を表し，SVD の左および右特異行列に対応し，$\mathcal{C}$はコアテンソルと呼ばれ，SVD の特異値行列に対応する．$\times_n, (n = 1,2,3)$はn-モード積と呼ばれ，高階テンソルと行列の積の演算を表す．例えば，$\times_3$は 3-モード積を表す．

以上より，元の 3 階テンソル$\mathcal{A}$は，HOSVD により 3 つの直交行列と 1 つのコアテンソルのn-モード積に分解されることが分かる．この分解のイメージを図 2 に示す．

次に，式(2)を計算する HOSVD アルゴリズムを示す．

（アルゴリズム 1：3 階テンソルの HOSVD）

入力：3 階テンソル$\mathcal{A}$.

出力：正規直交行列$U^{(n)}, (n = 1,2,3)$とコアテンソル$\mathcal{C}$.

ステップ 1：$\mathcal{A}$にn-モード行列展開（詳細は後述する）を適用し，展開された行列$A_{(n)}, (n = 1,2,3)$を求める．

ステップ 2：ステップ 1 で得られた行列$A_{(n)}, (n = 1,2,3)$の SVD をそれぞれ次式で計算する．

$$A_{(n)} = U^{(n)}\,\Sigma^{(n)}\,V^{(n)T}, (n = 1,2,3) \quad (3)$$

ただし，$U^{(n)}$および$V^{(n)}$は，各列に特異ベクトルを持つ左および右特異行列を表し，$\Sigma^{(n)}$は対角成分に特異値を持つ特異値行列であり，記号 T は，行列の転置を表す．

ステップ 3：元のテンソル$\mathcal{A}$とステップ 2 で得られた左特異行列$U^{(n)}, (n = 1,2,3)$から，次式によりコアテンソル$\mathcal{C}$を計算する．

$$\mathcal{C} = \mathcal{A} \times_1 U^{(1)T} \times_2 U^{(2)T} \times_3 U^{(3)T} \quad (4)$$

ステップ 4：$U^{(n)}, (n = 1,2,3)$および$\mathcal{C}$を返す．

（アルゴリズム終わり）

2.3. テンソル分解プログラミング教育への応用

図 3 は，前節で述べたアルゴリズム 1 の流れを示しており，この図の流れに沿ってテンソル分解プログラミングを次のように教えることにする．

（1）まず，このアルゴリズムの全体像を学習者に掴ませるために，既に開発された CG によるアルゴリズムの可視化アプリ[7]を利用する．

（2）次に，図 3 の高階テンソル入力と行列展開の部分は，本論文で提案する立体パズルを利用して理解支援を行い，3.3 節および 3.5 節のスクリプト例に示すような行列展開を求めるプログラムを作成させる課題を与える．

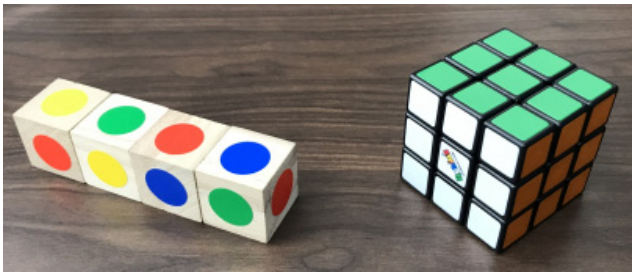
（3）図 3 の特異値分解の部分は，特に左特異行列$U^{(n)}, (n = 1,2,3)$は，高階テンソルの各モードの特徴を表すため，その理解が重要と考えられるので，（2）で作成された行列展開のデータを実際にプログラミング言語で特異値分解させ，$U^{(n)}$の各列ベクトルの変化を可視化させ

ることにより理解を深めさせる．

（4）最後に，コアテンソルの計算部分は，n-モード積の理解が重要であるが，この積の計算は高階テンソルと行列の積として直接計算するもので，理解が難しいと考えられる．そこで，高階テンソルの行列展開と左特異行列 $\boldsymbol{U}^{(n)}$ の行列積から得られた行列を再びテンソルとして畳み込むことと等価であることから，この計算法を用いることにする．そして，本論文の立体パズルを利用して，テンソルの行列展開と畳み込みの理解支援を行い，実際にコアテンソルを計算するプログラムを実装させる．

以上が立体パズルを用いたテンソル分解プログラミング教育の流れであるが，高階テンソルの取り扱いと行列展開および畳み込みがテンソル分解アルゴリズム全体に関連する重要なポイントであるといえる．この教育を実施する対象者としては，プログラミングおよび線形代数の知識があることが前提であり，高等専門学校の学生でいえば，4 年生以上がその対象と考えられる．この取り組みの実際の適用と教育効果の調査については，今後の課題であるが，対象学生を2つのグループに分け，立体パズルを利用するグループと，利用しないグループとの間で理解度の比較を行うことを考えている．

本論文では，実際に開始する前段階として，2種類の立体パズルを利用して2つの試みを行った．1つは，テンソル分解の知識を持つ高専専攻科生に対し，立体パズルの行列展開を CG で表現させるプログラミング課題を実施したもので，もう1つは，オープンキャンパスに参加した小中学生・高専生・一般に対し，立体パズルの行列展開表現を応用したマップ表示アプリと実物のパズルを利用して，マップ表示の理解調査を行ったものである．



（a）インスタント・インサニティ　　（b）ルービックキューブ

図 4.　立体パズル

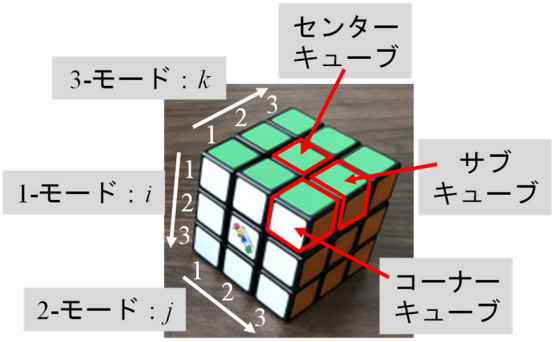


図 5.　ルービックキューブの 3 階テンソル表現

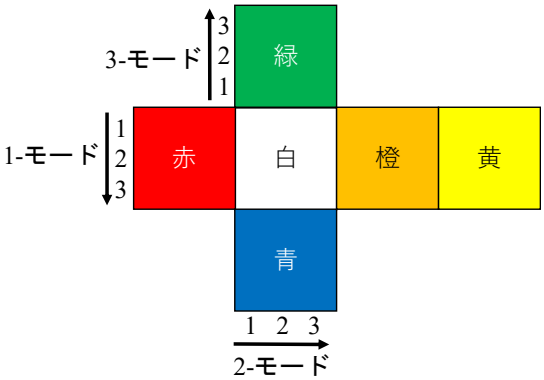


図 6.　ルービックキューブの展開図と配色

3. 立体パズルとその展開表現

3.1. 立体パズル

本論文で取り扱う立体パズルは，図 4 に示すようなインスタント・インサニティおよびルービックキューブである．インスタント・インサニティ（カラーダイスパズルとも呼ばれる[13]）は市販されてもいるが，図 4(a)は 4 個の木製の立方体にそれぞれ色シールを貼り，自作したものである．図 4(b)に示すルービックキューブは市販されているルービックキューブ ver2.0 で，このキューブの配色が世界基準となっている[14]．

3.2. ルービックキューブの高階テンソル表現

ルービックキューブは，サイズ3 × 3 × 3の 3 階テンソルにより表現できる．ルービックキューブにおける各モードの関係を図 5 に示す．

このパズルには，図 5 に示すように 1 色からなるセンターキューブ 6 個，2 色からなるサブキューブ 12 個，3 色からなるコーナーキューブ 8 個がある．さらに，本論文ではパズル内部に色を持たないキューブが 1 個あるとして，これをインナーキューブと呼ぶ．パズルは，このような 27 個のキューブから構成されるものとする．

図 6 に，白色の面を中心として描いたパズルの配色を展開図にして示す．ただし，図 6 で示されていないインナーキューブは無色とする．さらに，表 1 に，図 5 における

表 1. 各キューブの配色情報

キューブ番号	キューブ種類	キューブ配色	図 5 での要素番号
1	コーナー	緑, 赤, 白	(1,1,1)
2	サブ	緑, 赤	(1,1,2)
3	コーナー	緑, 赤, 黄	(1,1,3)
4	サブ	緑, 白	(1,2,1)
5	センター	緑	(1,2,2)
6	サブ	緑, 黄	(1,2,3)
7	コーナー	緑, 橙, 白	(1,3,1)
8	サブ	緑, 橙	(1,3,2)
9	コーナー	緑, 橙, 黄	(1,3,3)
10	サブ	赤, 白	(2,1,1)
11	センター	赤	(2,1,2)
12	サブ	赤, 黄	(2,1,3)
13	センター	白	(2,2,1)
14	インナー	なし	(2,2,2)
15	センター	黄	(2,2,3)
16	サブ	橙, 白	(2,3,1)
17	センター	橙	(2,3,2)
18	サブ	橙, 黄	(2,3,3)
19	コーナー	青, 赤, 白	(3,1,1)
20	サブ	青, 赤	(3,1,2)
21	コーナー	青, 赤, 黄	(3,1,3)
22	サブ	青, 白	(3,2,1)
23	センター	青	(3,2,2)
24	サブ	青, 黄	(3,2,3)
25	コーナー	青, 橙, 白	(3,3,1)
26	サブ	青, 橙	(3,3,2)
27	コーナー	青, 橙、黄	(3,3,3)

各キューブの配置と配色の詳細な情報を示す.

このパズルを 3 階テンソル$\boldsymbol{\mathcal{A}}$で表し, $\boldsymbol{\mathcal{A}}$の(i,j,k)要素をa_{ijk}とすると, 次式のように表現できる.

$$\boldsymbol{\mathcal{A}} = (a_{ijk}),\ (i,j,k = 1,2,3) \quad (5)$$

ただし, $a_{ijk},(i,j,k = 1,2,3)$には, それぞれ対応する表 1 のキューブ番号を与える. 例えば, a_{111}では 1, a_{123}では 6 などとなる. パズルの各キューブの配色は, キューブ番号から, 表 1 を参照して得ることができる.

3.3. ルービックキューブの行列展開表現

次に, 式(5)の高階テンソルで表現されたパズルを展開図として表現する方法について説明する. 本論文では, 2.2 節で述べたアルゴリズム 1 のステップ 1 における n-モード行列展開を利用する. 行列展開とは, 図 3 に示すような元の高階テンソルを行列に変換する操作のことである.

この行列展開の仕方については, 諸論文で様々な方法があるが, ここでは文献[6, 15]の方法を用いた. 行列展開は N 階テンソルまでに適用できるが, 次に示すのは 3 階テンソルの行列展開のアルゴリズムである.

(アルゴリズム 2: 3 階テンソルの n-モード行列展開)

入力: サイズ$I \times J \times K$の 3 階テンソル$\boldsymbol{\mathcal{A}}$.

出力: n-モード行列展開$\boldsymbol{A}_{(n)},(n = 1,2,3)$.

以下のステップ 1 から 3 の操作は, $i = 1,2,\dots,I;\ j = 1,2,\dots,J;\ k = 1,2,\dots,K$の全ての場合について行う.

ステップ 1 (1-モード行列展開): 3 階テンソル$\boldsymbol{\mathcal{A}}$の要素a_{ijk}を行列$\boldsymbol{A}_{(1)}$のi行$(j-1)K+k$列の要素に移す.

ステップ 2 (2-モード行列展開): $\boldsymbol{\mathcal{A}}$の要素a_{ijk}を行列$\boldsymbol{A}_{(2)}$のj行$(k-1)I+i$列の要素に移す.

ステップ 3 (3-モード行列展開): $\boldsymbol{\mathcal{A}}$の要素a_{ijk}を行列$\boldsymbol{A}_{(3)}$のk行$(i-1)J+j$列の要素に移す.

ステップ 4:行列$\boldsymbol{A}_{(n)},(n = 1,2,3)$を返す.

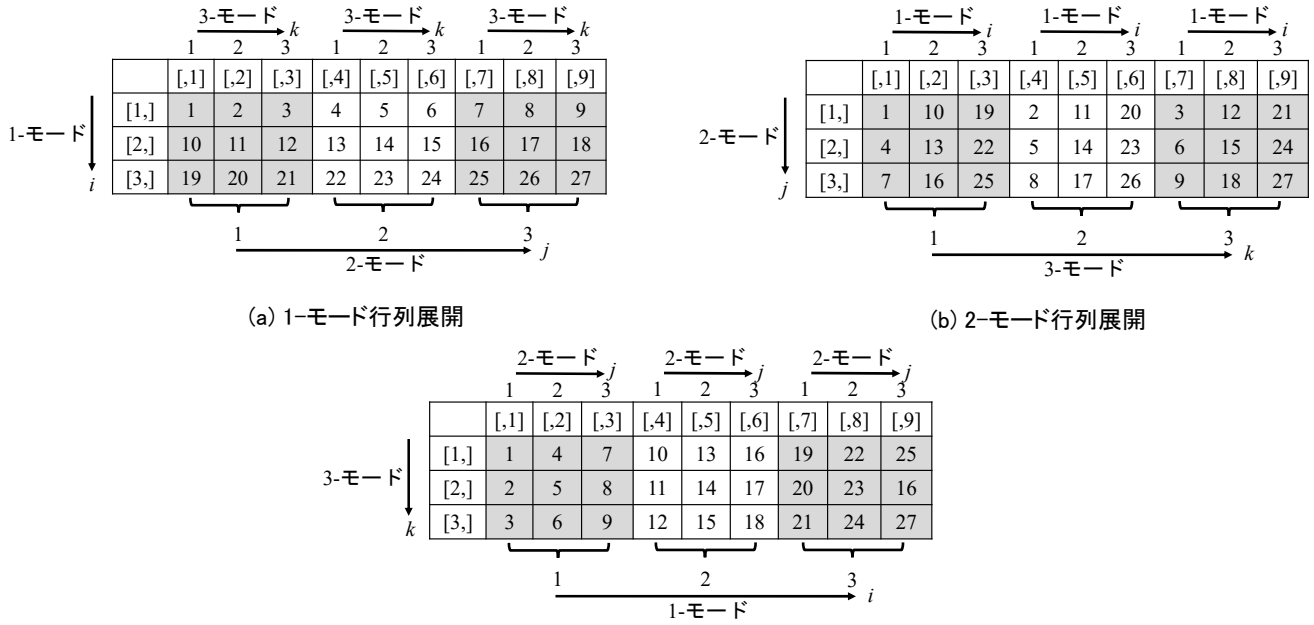
(アルゴリズム終わり)

このアルゴリズムを適用すると, 元のテンソル$\boldsymbol{\mathcal{A}}$の各モードが行列$\boldsymbol{A}_{(n)},(n = 1,2,3)$の各行に移される. 例えば, 1-モード行列展開$\boldsymbol{A}_{(1)}$の行には$\boldsymbol{\mathcal{A}}$の添字 i が対応しており, $\boldsymbol{\mathcal{A}}$の 1-モードが移される. 3 階テンソルの場合のモード数は 3 であるので, 3 通りの行列展開が得られ, N 階テンソルの場合は N 通りとなる.

ルービックキューブの行列展開表現の原理をより具体的に示すために, 統計解析用のプログラミング言語 R[16]を利用し, テンソル計算用のパッケージである rTensor[17]を導入して, このパズルの展開図を作成する R スクリプトを次に示す. R スクリプトは, 実際に R 言語を用いた教育を行う際の参考のために載せたもので, 以降も同様である.

(スクリプト1: ルービックキューブの展開図計算)

```
# パッケージ rTensor の利用
 library( rTensor )
# テンソル A のサイズ指定
 I <- 3; J <- 3; K <- 3
# テンソル A の初期化
 A <- array( 0, c(I,J,K) ); A <- as.tensor( A )
# テンソル A の作成
 A[1, , ] <- 1:9; A[1, , ] <- t( A[1, , ]@data )
```



(a) 1−モード行列展開

(b) 2−モード行列展開

(c) 3−モード行列展開

図 7. *n*−モード行列展開の計算結果

```
  A[2, , ] <− 10:18; A[2, , ] <− t( A[2, , ]@data )
  A[3, , ] <− 19:27; A[3, , ] <− t( A[3, , ]@data )
# テンソル A の n-モード行列展開計算
  A_1mode <− unfold( A, 1, c(3,2) )
  A_2mode <− unfold( A, 2, c(1,3) )
  A_3mode <− unfold( A, 3, c(2,1) )
# n-モード行列展開の結果表示
  A_1mode@data     # 1-モード行列展開
  A_2mode@data     # 2-モード行列展開
  A_3mode@data     # 3-モード行列展開
```

（スクリプト終わり）

スクリプトの中の記号<−は代入操作を表している. 図 7(a)から(c)は, スクリプト 1 で計算されたパズルの *n*-モード行列展開の結果である.

スクリプト 1 において, 行列展開の計算にはパッケージ rTensor の関数 unfold を利用した. スクリプトでは, 関数 unfold の第 1 引数に元のテンソルを, 第 2 引数に行列展開の行に移すモード番号を, 第 3 引数には行列展開の列に移すモード番号を与えている. 例えば, 1-モード行列展開は unfold(A, 1, c(3,2))で求められるが, 行列展開の行に移すモードとして 1-モードを指定し, 列に移すモードとして, 2-モードと 3-モードを指定している. 第 3 引数でベクトル c(3,2)としているのは, 図 7(a)から分かるように, 3-モードの添字は 2-モードの添字よりも先に変化させる必要があるためである.

図 7(a)の 1-モード行列展開は, 図 5 のテンソルの 2-モードに沿って輪切りして得られた行列を横に並べられたものと考えることができる. その他のモードの行列展開についても同様である.

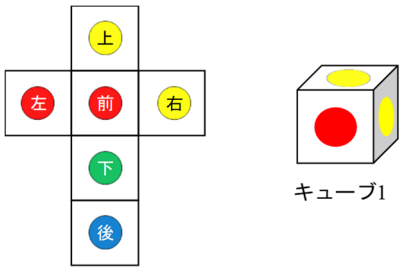
図 7 ではテンソルの要素値として, 表 1 のキューブ番号が格納されている. したがって, 表 1 を参照して各配置に対応するキューブ配色で塗れば, グラフィカルなパズルの展開図を作成することができる.

3.4. インスタント・インサニティの高階テンソル表現

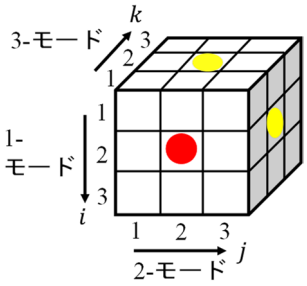
このパズルは, 図 4(a)に示すようにサイズ$1 \times 1 \times 1$の 4 つのキューブからなる. 本論文では, これらのキューブをそれぞれ$3 \times 3 \times 3$の 3 階テンソルにより表現する.

図 8(b)は図 4(a)の一番左のキューブを取り出したもので, これをキューブ 1 と呼ぶことにする. キューブ 1 の各面の配色は図 8(a)のようになっている. ここで, 図 8(b)に示される元のキューブのサイズを$3 \times 3 \times 3$に拡大して, 図 8(c)のような 3 階テンソルで表す. すなわち, 図 8(a)の配色は, この$3 \times 3 \times 3$のテンソルの各面の中央要素に塗られるものとする.

図 4(a)のようにキューブは横に 4 個並べられるため, 3 階テンソルを 4 個並べた$3 \times 3 \times 3 \times 4$ の 4 階テンソルとして, このパズルは表現される. 図 9 に, 図 4(a)の各キューブの配置を 4 階テンソル表現したイメージを示す. また, 図 10 には, 図 4(a)の左から 2〜4 番目のキューブ(キュ



(a) 元のキューブ 1 の配色　　(b) 元のキューブ 1



(c) 拡大されたキューブ 1

図 8. キューブ 1 の 3 階テンソル表現

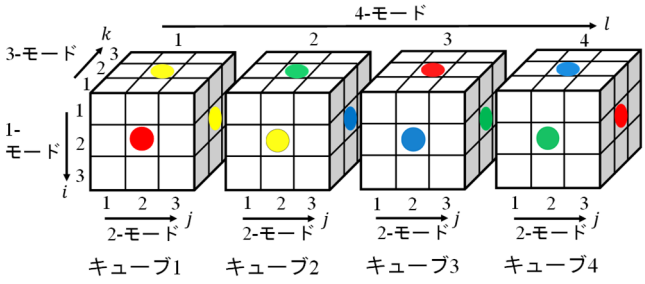


図 9. インスタント・インサニティの 4 階テンソル表現

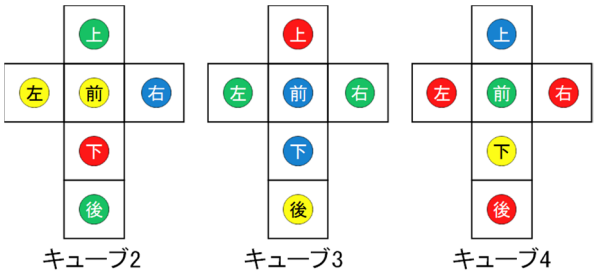


図 10. 元のキューブ 2 から 4 の配色

表 2. 図 9 での各キューブの配色情報

キューブ番号	テンソル要素番号	キューブ配色	キューブ番号	テンソル要素番号	キューブ配色
1	(1,2,2,1)	黄	3	(1,2,2,3)	赤
	(2,1,2,1)	赤		(2,1,2,3)	緑
	(2,2,1,1)	赤		(2,2,1,3)	青
	(2,2,3,1)	青		(2,2,3,3)	黄
	(2,3,2,1)	黄		(2,3,2,3)	緑
	(3,2,2,1)	緑		(3,2,2,3)	青
2	(1,2,2,2)	緑	4	(1,2,2,4)	青
	(2,1,2,2)	黄		(2,1,2,4)	赤
	(2,2,1,2)	黄		(2,2,1,4)	緑
	(2,2,3,2)	緑		(2,2,3,4)	赤
	(2,3,2,2)	青		(2,3,2,4)	赤
	(3,2,2,2)	赤		(3,2,2,4)	黄

ーブ 2～4)の配色を示した. 表 2 には, この 4 階テンソルの配色情報をまとめている.

いま, この 4 階テンソルを$\boldsymbol{\mathcal{B}}$, $\boldsymbol{\mathcal{B}}$の(i,j,k,l)要素をb_{ijkl}とすると, 次式のように表現できる.

$$\boldsymbol{\mathcal{B}} = \left(b_{ijkl}\right),\ (i,j,k = 1,2,3; l = 1,2,3,4) \qquad (6)$$

ただし, 表 2 に記載された各テンソル要素の配色が, 赤のとき$b_{ijkl} = 1$, 黄は 2, 緑は 3, 青は 4 を要素値として与え, 記載のないテンソル要素はすべて 0 とする.

3.5. インスタント・インサニティの行列展開表現

このパズルの展開図を作成するアルゴリズムを次に示す. ここでは, 1-モード行列展開のみを利用したが, 他のモードでも展開図を得ることができる.

(アルゴリズム 3:インスタント・インサニティの展開図作成)

入力: 表 2 の配色を持つサイズ$3 \times 3 \times 3 \times 4$の 4 階テンソル$\boldsymbol{\mathcal{B}}$.

出力: 1-モード行列展開$\boldsymbol{B}_{(1)}$.

ステップ 1: $\boldsymbol{\mathcal{B}}$から 3 階テンソル$\boldsymbol{\mathcal{B}}_l = (b_{***l}), (l = 1,2,3,4)$を取り出す. ただし, b_{***l}は, lだけを固定した$i, j, k = 1,2,3$の 3 階テンソルである.

ステップ 2: $\boldsymbol{\mathcal{B}}_l$, $(l = 1,2,3,4)$に, それぞれアルゴリズム 2 のステップ 1 を適用して, $\boldsymbol{B}_{l\,(1)}, (l = 1,2,3,4)$を求める.

ステップ 3: $\boldsymbol{B}_{l\,(1)}, (l = 1,2,3,4)$を横に並べて, $\boldsymbol{B}_{(1)} = \left(\boldsymbol{B}_{1\,(1)} | \boldsymbol{B}_{2\,(1)} | \boldsymbol{B}_{3\,(1)} | \boldsymbol{B}_{4\,(1)}\right)$とする.

ステップ 4: $\boldsymbol{B}_{(1)}$を返す.

(アルゴリズム終わり)

ここで, この立体パズルの行列展開表現の原理を具体的に示すために, 式(6)の 4 階テンソルに, アルゴリズム 3 を適用して展開図を得る R のスクリプトを次に示す.

(スクリプト 2:インスタント・インサニティの展開図計算)

```
library(rTensor)     # rTensor を利用する
# テンソル B の初期化
 B <- array( 0, c(3,3,3,4) ); B <- as.tensor( B )
# テンソル B の作成
# 色の設定( 赤:1, 黄:2, 緑:3, 青:4 )
 r <- 1; y <- 2; g <- 3; b <- 4
# 各キューブの前・後／上・下／左・右の面の
# 中央に配色
 B[2,2,c(1,3), ] <- rbind( c(r,y,b,g), c(b,g,y,r) )     # 前・後
 B[c(1,3),2,2, ] <- rbind( c(y,g,r,b), c(g,r,b,y) )     # 上・下
 B[2,c(1,3),2, ] <- rbind( c(r,y,g,r), c(y,b,g,r) )     # 左・右
# テンソル B の 1-モード行列展開計算
 B_1mode <- unfold( B, 1, c(3,2,4) )
 B_1mode@data     # 結果表示
```

(スクリプト終わり)

このスクリプトを実行して得られた図 9 のキューブ 1 部分の展開図を図 11 に示す. この図は, 図 8(c)の 2-モードに沿って輪切りにした行列を横に並べたものである. したがって, 一番左の3×3の行列にはパズルの左面の赤色が, 左から 2 番目の3×3の行列にはパズルの前, 後, 上, 下の色, すなわち赤, 青, 黄, 緑の色が, 一番右の3×3の行列にはパズルの右面の黄色が現れていることが分かる. キューブ 2 から 4 部分についても, キューブ 1 の場合と同様にして展開図を得ることができる.

以上より, アルゴリズム3を実行すれば, 各キューブの 1-モード行列展開による展開図が得られるので, 実物のキューブを回転させて正しい並びを揃える際に, この展開図も参照させながら行わせることで, テンソルデータの行列展開や多次元データ処理の理解支援に繋げられるのではないかと考えている.

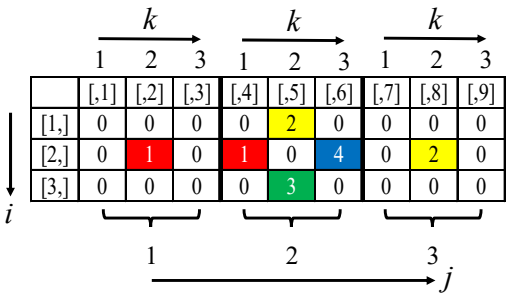


図 11. キューブ 1 部分($l = 1$)の 1-モード行列展開

4. 立体パズルの展開表現の教育への応用

4.1. ルービックキューブの行列展開表現の応用

3.3 節でルービックキューブの行列展開表現のアルゴリズムについて述べた. ここでは, 本校(熊本高等専門学校)においてこの展開表現を学生の教育に応用した事例について記す.

対象とする学生は平成 29 年度の専攻科 2 年生で, 実施期間は 2〜3 週間程であった. この学生は, 本科 5 年次の卒業研究と専攻科 1 年次および 2 年次のシステム工学特別研究を通して, HOSVD および Nonnegative Tucker Decomposition (NTD)などのテンソル分解法を利用した医療データ分析に関する研究に携わっていた. また, CG に関するプログラミング言語は, 専攻科 1 年次の実習科目で Unity や C#を使用してアプリケーションを開発した経験を有していた.

この学生へ与えたアプリ作成の課題は次のとおりである.「ルービックキューブを 3 階テンソルとして取り扱い, n-モード行列展開を求め, その配色を CG で表現すること」と,「ルービックキューブの回転に伴って n-モード行列展開の配置と配色も変化させること」というものであった. 学生には図 4(b)に示すような実物のルービックキューブを貸与した.

この課題の Unity と C#によるアプリの実行例を図 12 に示す. パズルの行列展開と配色の確認のために, 図 12 左上に示したルービックキューブは, 図 5 と同じ配置で, モードの取り方も同じにしている.

これより, 図 7 の計算結果および表 1 の配色情報と,

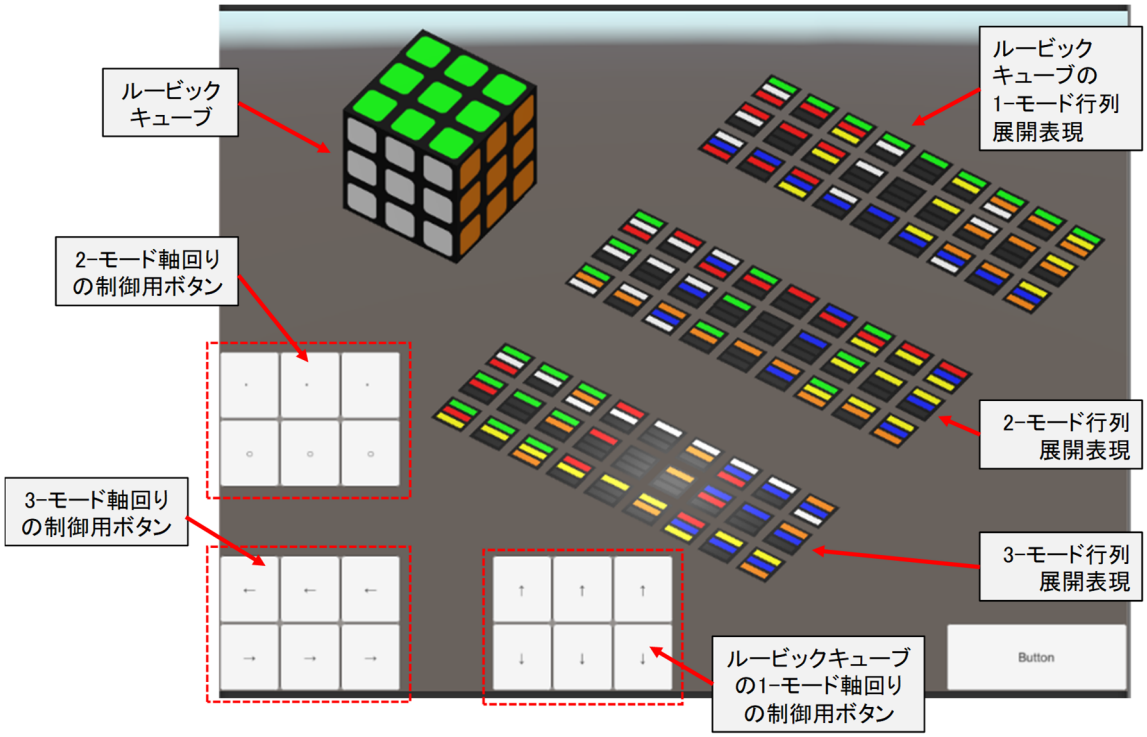


図 12. ルービックキューブのアプリの実行例

図 12 の各行列展開に塗られた色とを比較すると，一致していることが分かる．したがって，このアプリは行列展開の原理に基づいたパズルの展開図を正しく作成しているといえる．

さらに，このアプリでは，図 12 左下側にある制御用ボタンを用いて，各モード軸回りにルービックキューブを回転させることができる．キューブの回転に伴って，各行列展開の配色も変化するようになっている．

学生が作成したこのアプリの実行結果から，本学生は Unity および C#の CG プログラミングのスキルと，研究を通して学んだ高階テンソルの取り扱いや行列展開についてよく理解し，それをプログラミングして応用できる力を身につけたことが分かる．

4.2. インスタント・インサニティの行列展開表現の応用

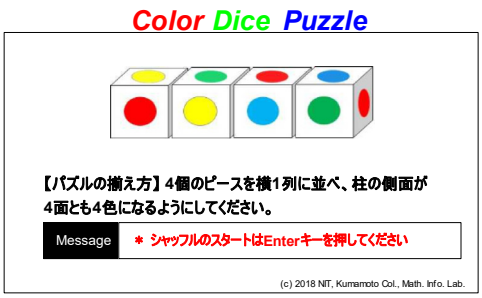
3.5 節で記したこのパズルの行列展開のアルゴリズムに基づいて R を用いて展開図を表示するアプリを作成した．このアプリ（図 13）と実物のパズル（図 4(a)）を利用して，本校のオープンキャンパス参加者に対し，展開図が理解できるかどうかを調査した事例[10]について述べる．

実施時期は平成 30 年 8 月上旬で，我々の研究室展示を見学した小中学生，高専本科生，保護者，教員の計 43 名を対象とした．このときの調査の様子は，次のとおりである．

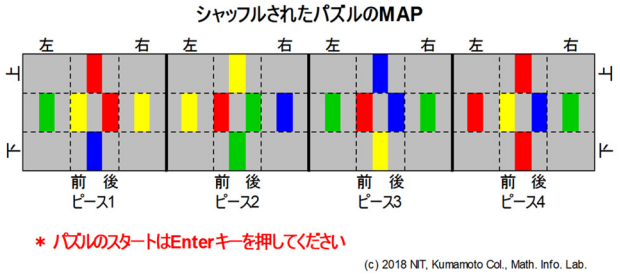
見学者が展示ブースを訪れると，説明者がこのパズルのアプリを起動して，図 13(a)のタイトル画面を表示させ，パズルの揃え方を説明する．その際，実物のパズルも併用する．

次に，アプリ上のパズルをシャッフルさせ，図 13(b)のような展開図を表示させる．このとき，展開図の見方も説明するが，キューブ 1 部分（アプリではピース 1 と表示）について，実物のキューブ 1 を用いて，具体的に図と合わせる．そして，残りのキューブ 2 から 4（アプリではピース 2 から 4 と表示）は，見学者が展開図を確認しながら実物のパズルと同じ色の配置になるように合わせる．図 13(c)は中学生 2 名が実施している様子である．

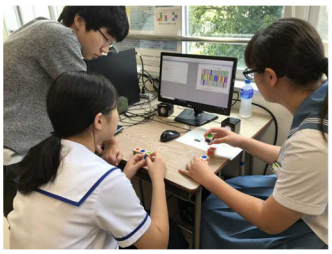
このアプリにはパズルをランダム試行で揃える機能が実装されており，これをスタートすると図 13(d)に示すように展開図の配色が次々に変化し，アプリがパズルを揃えている様子を確認できる．この際，見学者には実物のパズルを解くように指示し，アプリと対戦させる．



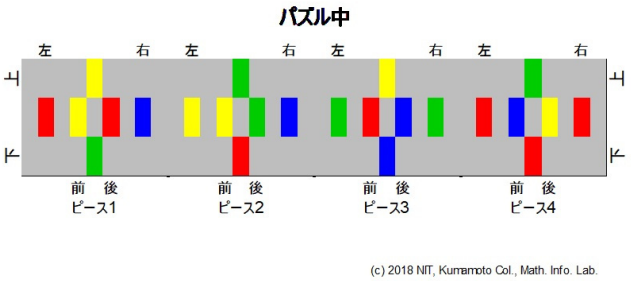
(a) タイトル画面



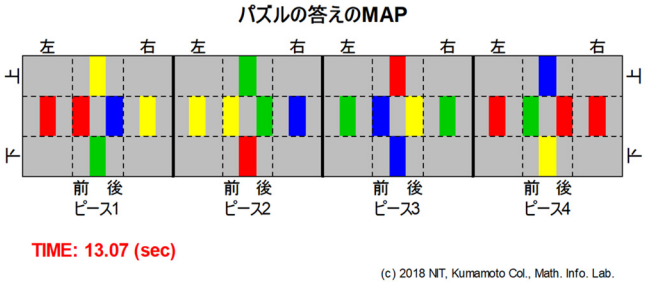
(b) パズルのシャッフル



(c) 展開図を確認しながら実物のパズルを配置



(d) パズルの求解途中



(e) パズルの解の表示

図 13. インスタント・インサニティのアプリの実行例

アプリが最長 2 万回(所要時間は約 2 分)までに解を求められれば，図 13(e)のようにその答えを表示するが，求められなかった場合でも他の解の例を表示する．見学者がアプリの解の表示時に，実物のパズルが解けていない場合は，見学者は図 13(e)の展開図を確認しながら，実物のパズルを配置し，解き終わってから終了する．

パズル終了後，各見学者に図 13 に示したパズルの展開図が理解できたかどうかを質問した．見学者の内訳は，小学生:1 名，中学生:31 名，高専本科生・教員・保護者:11 名であった．質問の回答をまとめると，「理解できた」が 29 名(67%)，「理解しづらかった・できなかった」が 14 名(33%)であり，概ねアプリで表示された展開図が理解できているものと考えられる．

図 13 の展開図は，各キューブの 1-モード行列展開から作成されたものである．アプリのユーザーは，パズルのシャッフル表示時と解の表示時に展開図を見ながら実物のパズルを配置することで，1-モード行列展開の並びを学習する効果があると考えられる．この事例のように，本アプリを利用することにより，テンソル分解や多次元データ処理への関心を持たせることが可能で，卒業研究・講義などに利用すればテンソル分解アルゴリズムやそのプログラミングの理解支援に繋がると考えており，今後も試用を行っていきたい．

5. まとめ

本論文では，立体パズルとしてポピュラーなルービックキューブとインスタント・インサニティを題材として取り上げ，まず各パズルの高階テンソル表現の例を示して，行列展開アルゴリズムにより展開図を作成する方法とそのスクリ

プトについて述べた.

そして, 本校専攻科生にルービックキューブを題材として与えて, このパズルの展開図作成アプリを実装させた事例について述べ, この学生が在学中の実習や研究を通して, テンソル分解を理解しプログラミングができる力のあることを確認した.

また, オープンキャンパスの研究室展示に訪れた小中学生および保護者等に対し, インスタント・インサニティの展開図を表示するアプリと実物のパズルを使ってもらい, 展開図が理解できるかどうか調査した事例を述べ, アプリに表示される展開図が概ね理解できることを確認した.

今後の課題としては, ルービックキューブについては, 同じ位置での同じキューブの回転が考慮されていないため, その高階テンソル表現と展開図表現の改善の必要があることと, インスタント・インサニティについては, より直感的に理解できるような展開図表示方法への改良などが挙げられる. さらに, 本校学生に対して, テンソル分解の教育に本格的に利用して, 効果を評価することも今後行わなければならないことである.

参考文献

[1] L. De Lathauwer “A Survey of Tensor Methods”, Proc. IEEE International Symposium on Circuits and Systems, pp.2773-2776, 2009.

[2] 村上純, 山本直樹, 田所嘉昭 “べき乗法を用いた 3 階テンソル積展開の高速計算法”, 電子情報通信学会論文誌 A, Vol. J82-A, No.8, pp.1351-1359, 1999.

[3] A. Ishida, T. Noda, J. Murakami, N. Yamamoto, and C. Okuma “Calculation of Fourth-Order Tensor Product Expansion by Power Method and Comparison of It with Higher-Order Singular Value Decomposition”, Applied Mechanics and Materials, Vols. 444-445, pp.703-711, 2013.

[4] N. Yamamoto, J. Murakami, K. Fujii, C. Okuma, S. Saito, T. Izumi, and N. Hayashida “Measurement and Analysis of the Functional Independence Measure Data by Using Nonnegative Matrix Factorization Method”, Advanced Materials Research, Vols. 718-720, pp.630-635, 2013.

[5] A. Ishida, K. Kawakami, D. Furushima, N. Yamamoto, and J. Murakami “Analysis of Relationship between Amount of Physical Activity of Patients in Rehabilitation and Their ADL Scores Using Multidimensional PCA”, Advances in Intelligent Systems and Computing, Vol. 690, pp.147-158, 2017.

[6] L. De Lathauwer, B. De Moor, and J. Vandewalle “A Multilinear Singular Value Decomposition”, SIAM Journal on Matrix Analysis and Applications, Vol.21, No.4, pp.1253-1278, 2000.

[7] 大隈千春, 長岡翔, 村上純, 山本直樹, 石田明男 “計算過程の可視化による高次特異値分解の理解支援システムの開発”, 論文集「高専教育」, Vol.38, pp.129-134, 2015.

[8] 山本直樹, 石田明男, 平田将大, 村上純 “立体パズルを利用した多次元データ分解手法の理解支援の試み”, 熊本高等専門学校研究紀要, Vol.9, No.1, pp.67-74, 2018.

[9] A. Ishida, N. Yamamoto, and J. Murakami, and N. Oishi “Solving 3-D Puzzles Using Tensor Decomposition and Application to Education of Multidimensional Data Analysis”, Intl. Journal of Machine Learning and Computing, Vol.8, No.5, pp.447-453, 2018.

[10] 山本直樹, 石田明男, 村上純, 大石信弘 “カラーダイスパズルを利用した *n*-モード行列展開の理解支援アプリの試作”, 熊本高等専門学校研究紀要, Vol.10, No.1, pp.75-78, 2019.

[11] 石田明男, 山本直樹, 大石信弘, 村上純 “多次元データ分解の手法を用いた立体パズルの解法”, 初等数学, Vol. 83, pp.18-22, 2018.（継続して掲載中）

[12] “Rubik’s Official Website”, Rubik’s, https://eu.rubiks.com/, Retrieved Feb. 27, 2019.

[13] 高木茂男 “PLAY PUZZLE パズルの百科”, pp.41-42, 平凡社, 1981.

[14] 日本ルービックキューブ協会, “ルービックキューブ ver.2.0 完全攻略公式ガイドブック”, p.5, 永岡書店, 2016.

[15] 村上純, 日野満司, 山本直樹, 石田明男 “統計ソフト R による 多次元データ処理入門－仮説検定・分散分析・主成分分析”, pp.223-227, 日新出版, 2017.

[16] R Core Team “R: A language and environment for statistical computing”, R Foundation for Statistical Computing, Vienna, Austria.

[17] J. Li, J. Bien, and M. Wells “Package ‘rTensor’”, https://cran.r-project.org/web/packages/rTensor/rTensor.pdf, 2018.

マイクロサービス開発への SOA 開発プロセスの拡張可能性の検討

宗平 順己
Kyoto ビジネスデザインラボ
t-munehira@kyoto-bdl.com

要旨

デジタルトランスフォーメーションの進展に伴い，SoE (Systems of Engagement)と呼ばれるタイプのアプリケーション開発の重要性が急激に高まってきており，先行する欧米では，マイクロサービスの利用がデファクト化してきている．

このマイクロサービス開発にあたっては，サービス設計の難しさが課題として共通認識されている．

このサービス粒度問題については，SOA 開発と同じ課題であることから，筆者らが開発した SOA 開発プロセスをマイクロサービス開発へと拡張できるのではないかと考えた．

本論では，SOA 開発プロセスが BPM からスタートすることから，マイクロサービス開発についても上流プロセスからの定義を行い，マイクロサービスの持つ特徴に留意して，どのように SOA 開発プロセスを拡張すればよいのかを検討する．

1. はじめに

デジタルトランスフォーメーションの進展に伴い，SOE SoE (Systems of Engagement) と呼ばれる新しいタイプのアプリケーション開発の重要性が急激に高まってきている．デジタルトランスフォーメーションの第一の目的は新たな顧客経験の提供にあるが，提供側も利用側も明確な要件が決まっているわけではなく，まずはサービスを提供して，ユーザの反応を見ながら素早く仕上げていくというアプローチがビジネス面，システム面両面において必要となる．

このため，デジタルトランスフォーメーション（以下 DX）へのチャレンジが先行する欧米では，マイクロサービスの利用がデファクト化してきている．

我が国においては，システム開発をアウトソーシングすることから，残念ながらまだまだウォーターフォール型の開発が多く，DX への対応が困難な状況にあるが，SIer の業界団体である JISA が 2018 年度になってマイクロサービス移行への必要性を強く主張する様になってきた．

しかしながら，このマイクロサービス開発にあたっては，サービス設計の難しさが課題として共通認識されており，モノリシックファーストという経験主義的なアプローチが提案されるなど，ソフトウェア工学的なアプローチがなされていない．

ところで，このサービス粒度問題については，SOA 開発においても同様の課題があり，筆者らはこの課題に対応するために SOA 開発プロセスを開発した．

そこで，この SOA 開発プロセスをマイクロサービス開発に拡張できるのではないかと考え，適用可能性について検討したので，その結果を報告する．

第 1 章では．今なぜマイクロサービスが必要となっているのかについて，背景としての DX の概念を整理し，その上で，DX 時代に求められるシステムの特性をアーキテクチャ面から整理し，マイクロサービス適用の必要性について整理する．

第 2 章では．マイクロサービスの開発における粒度問題について整理する．一般に指摘されている下流側からの粒度課題を確認するだけではなく，上流からのアプローチにおける課題についても整理する．DX の開発において，欧米ではサービスデザインから始まることがデファクト化していることから，日本ではまだ馴染みのない，サービスデザインからマイクロサービスへ展開するアプローチを紹介し，そこで課題になっている粒度問題について言及する．

第 3 章では，まず SOA 開発における粒度課題とその解決方法を紹介したのち，マイクロサービスの粒度問題との相違点を整理し，SOA 開発プロセスのマイクロサービス開発への適用に向けての拡張要件を整理する．

本論では実際の案件にて拡張した開発プロセスを検証するところまでには至っていないことから，今後の課題として，必要な検証項目について整理し，まとめとする．

2. 今なぜマイクロサービスが必要なのか

2.1. 背景としての DX

2.1.1 DX の定義

調査会社の IDC は次の様に定義している．「企業が第 3 のプラットフォーム技術を利用して，新しい製品やサービス，新しいビジネスモデル，新しい関係を通じて価値を

創出し，競争上の優位性を確立すること」

この第 3 のプラットフォームというのは IDC が提唱しているコンセプトで，「クラウド」「ビッグデータ」「モビリティ」「ソーシャル」の 4 要素によって形成される情報基盤のことである．

DX の先駆者として良く例に出されている Uber や Airbnb のビジネスモデルをみると，実にうまく第 3 のプラットフォームの 4 要素を使いこなしていることがわかる．

DX の定義については「The best understanding of digital transformation is adopting business processes and practices to help the organization compete effectively in an increasingly digital world.」という主張があり，道具としてのデジタル化ではなく，経営環境としてデジタル化をとらえていくべきであるという考え方がある．[1]

2.1.2 DX への取り組み目的

MIT Sloan が 2015 年秋に Deloitte と共に世界中の 3700 以上の CEO やマネージャに実施した調査において，図-1 に示す様に 90%近くがデジタルによって産業構造に破壊をもたらすと考えており，44%がそれに対する備えを進めていると回答している．[2]

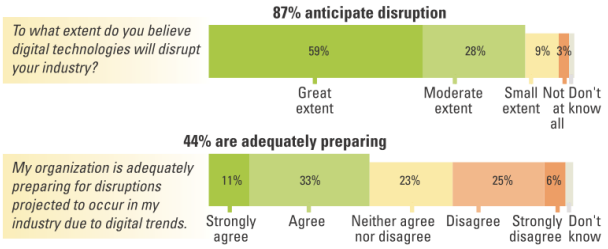


図-1 DX への対応

NoSQL サーバの提供元である Couchbase 社が 2017 年 5 月に米国，英国，フランス，ドイツで DX に積極的に取り組んでいる企業 450 社の CIO や CTO にオンライン調査を実施した．

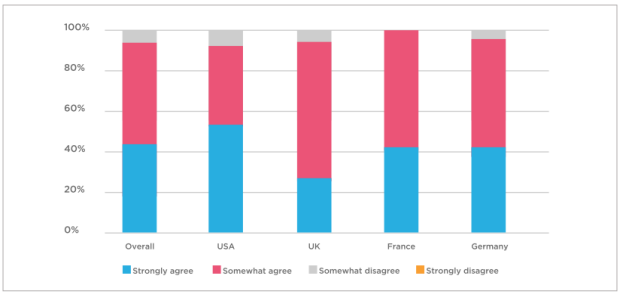


Figure 4: Do respondents agree that the ultimate aim of digital innovation should be to give customers and end-users a truly unique experience?

図-2 DX のゴール

図-2 に示すように，顧客やエンドユーザに今までとは異なる顧客経験を提供することが，DX のゴールであると回答している．[3]先に示した IDC の DX の定義では「新しい価値を創出し」と表現されているが，これはすなわち，新しい顧客経験を提供することであるといえる．

図-3 に示すように，同様の調査結果が，MIT SLOAN チームの 2017 年の調査においても示されており，いかに新しい顧客経験をデザインできるかが，課題といえる．

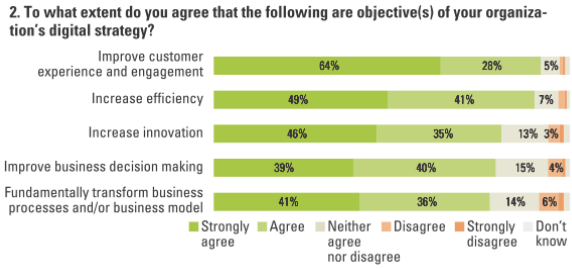


図-3 デジタル戦略の目的[4]

2.2. DX 時代に求められるシステムの特性

図-4 に示すペースレイヤーアプローチはガートナーが 2012 年に提唱したフレームワークで，主にシステムの変更頻度をもとにアプリケーションを分類するもので，ペースレイヤーごとに開発手法が異なる．[5]

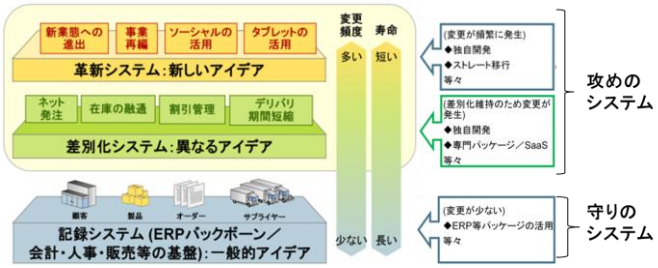


図-4 ペースレイヤーアプローチ 出典 5 を元に筆者が加筆

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

これに対し，革新システムは何が正解かわからず，まずはリリースし，顧客や利用者の反応を見ながら，よりロイ

ヤリティを高めるために，素早く改良を加えるということが求められる．リスクを早期に潰す反復型のアプローチでも反復頻度が間に合わず，アジャイル開発が必須であり，リリース期間を短縮するために DevOps を利用し，スケールアウト／インを迅速に行うためにクラウドが必須となっている．なお，一般に SoE（Systems of Engagement）と呼ばれるシステムはこの革新システムを指す．

図-5 は航空会社の座席予約システムの構成をペースレイヤーアプローチと関連して説明したものである．[6]

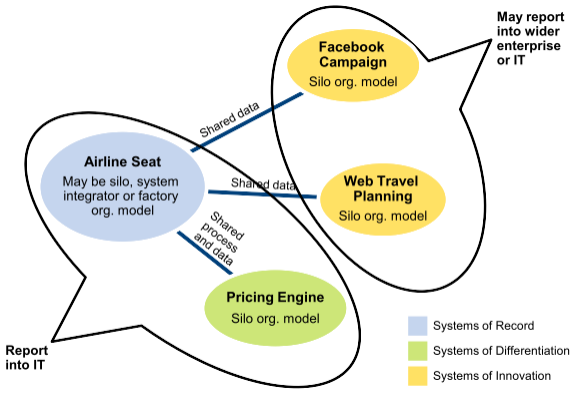


図-5 3つのレイヤーの連係モデル

DX のシステムは SoE であると書かれていることが多いが，実際にはこのように，3つのレイヤーのアプリケーションの組み合わせによって構築されることになる．

2.3. マイクロサービスの特性と DX での適用範囲

素早く改良を加えるということで多くの企業で使われてきているのが，マイクロサービスであり，その活用先進企業である Amazon は，1 年間に実に 5000 万回もシステム更新をかけている．

2.3.1 マイクロサービスの特徴

マイクロサービスの基本方針としては，マーティン・ファウラーが提唱している以下のものが共通認識とされている．[7]

＜マイクロサービスの基本方針＞

- 小規模なサービスを組み合わせてアプリケーションを開発すること
- 各サービスは，独立したプロセス上で稼働すること
- 各サービスは，REST などの軽量プロトコルで通信すること
- 各サービスは，自動的に，それぞれ個別にデプロイできること
- 各サービスは，それぞれ異なるプログラミング言語で実装可能であり， かつ，それぞれ異なる永続データストアを利用可能であること

独立性が高く入替が容易なサービス単位でアプリ ケーションを組み立て，リクエスト数の多いサービスが稼働するプロセスのみをスケールさせることができるようにすることも求められる．

マイクロサービスについてはまだ確立された仕様はなく，マーティン・ファウラーの9つのマイクロサービスの特徴が一種のガイドラインとして使われている．

＜マイクロサービスの9つの特徴＞

- サービスのコンポーネント化 - コンポーネントは独立して交換・更新可能なソフトウェアの単位である．
- ビジネス中心組織 -開発運用チームは技術や開発工程によるチームではなく，ビジネスを中心に機能横断型のチームが編成されている．
- プロジェクトではなくプロダクト - チームは開発完了とともに解散するプロジェクトモデルではなく，製品のライフサイクル全体に責任を持つ．
- スマートエンドポイントとダムパイプ - メッセージ通信は軽量かつシンプルであること．エンドポイントに高い機能を持たせ，通信はダムパイプ（メッセージのルータとしてのみ動作する単純な機構）であること．
- 分散統治 - 各サービスは独立したチーム，開発言語，ツールでアプローチする．
- 分散データ管理 - 概念モデルに関する分散だけでなく，データストレージも分散(サービス間で独立)する．
- インフラストラクチャの自動化 - テスト自動化/継続的デリバリー/継続的インテグレーションの導入
- 障害の設計 - 個々のサービスはいつでも失敗する可能性があるため，互いに障害を迅速に検出し，可能であれば自動的にサービスを復元できることが重要．
- 進化的な設計 - 頻繁に，迅速に，適切に制御されたソフトウェアの変更・廃止・構築を行うことができること．

2.3.2 DX での適用範囲

2.2 の図-5 に示したように DX のシステムは3つの異なる特性のシステムの組み合わせによって構築される．

このうち，マイクロサービスの特徴は，革新システム（Systems of Innovations）に求められるシステム要求に良く合致するものであることから，革新システム部分の開発

にあたってはマイクロサービスを適用することが最善の選択であると考えられている．

顧客起点でサービスを素早くブラッシュアップするAmazonが，マイクロサービスを採用していることがその一つの例証である．

3. マイクロサービスの開発における粒度課題

3.1. 下流側での粒度課題

マイクロサービスの適用にあたって課題となっているのはサービスの粒度である．独立性を高めようとすると，小さな処理単位でサービスを構築することが良いように思われるが，そうすると，各サービスは独立して稼働していることから，サービス間の通信が非常に多くなり，システムのパフォーマンスに深刻な影響を与えることになる．一方で，この問題を避けるために粒度を大きくすると，変更時にサービス内での影響調査の増大とアプリケーション構造の複雑さを生み，結局モノシリックシステムを構築するのと同じことになり，更新速度を著しく低下させることになる．

しかしながら，このサービス粒度については，現在ガイドラインがなく，サービスの粒度がバラバラになり，全体としての保守性を著しく損なうことになる．

この課題に対し例えば Amazon では10人規模で担当できる範囲をサービスの粒度とするとかモノシリックファーストというような考え方も提示されているが，いずれもトライ＆エラー型であり，工学的課題解決にはなっていない．

クラウドネイティブでアジャイル開発の決定打ともいえるマイクロサービスではあるが，このサービス粒度課題についての解決が必須となる．

3.2. 上流から下流への展開における課題

3.2.1 上流プロセスで使われるサービスデザイン

DX の目的は新たな顧客経験（CX）を提供することにある．その顧客経験の設計において広く採用されているのが，サービスデザインと呼ばれるアプローチである．

顧客経験の設計において最も重要なことは顧客インサイトを得ることであるが，具体的にはどのように得れば良いのであろうか．サービスデザインの先駆けであるイギリスのサービスデザイン会社 Engine のプロセスは図-6 の様に紹介されている[8]．このプロセスは一般にダブルダイヤモンドと呼ばれている．

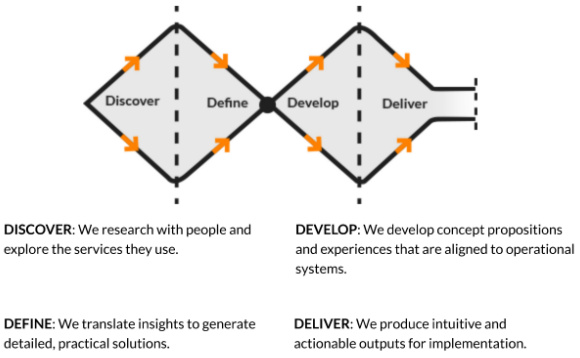


図-6　Engine 社のサービスデザインプロセス

「THIS IS SERVICE DESIGN THINKING [9]」も参考にしながらサービスデザインの具体的な進め方をまとめたのが図-7 である．

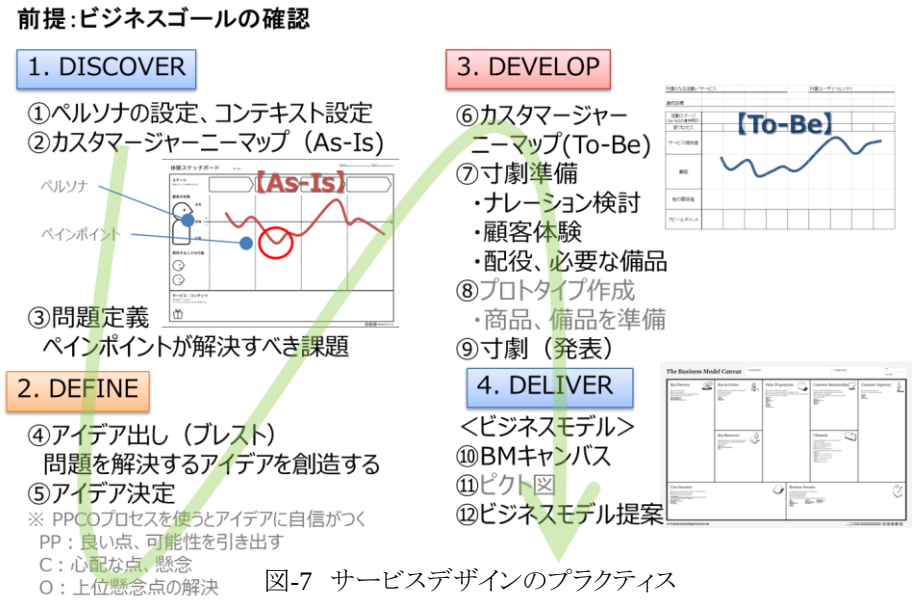


図-7　サービスデザインのプラクティス

まず前提となるビジネスゴールを全員で確認後，1. DISCOVER では，対象となる顧客（ペルソナ）の問題を認識することに専念する．この時に良く用いるのがカスタマージャーニーマップ（As-Is）であり，ペルソナが一連の活動において，どのような良い経験，悲しい経験をしているのかを書き出す．行動観察を行うと第三者でも共感しやすくなる．図にあるように顧客が不愉快，苦痛，不便と感じているペインポイントが解決のターゲットとなる．

2. DEFINE では，ペインポイントを鳥瞰して解決のアイデアだしを行う．最初は個別のペインポイントに着目したアイデア出しとなるが，次の DEVELOP と組合せてアイデアをブラッシュアップする．

3. DEVELOP は試作のステップであるが，まずは創出したアイデアを採用した場合の新しいカスタマージャーニーマップ（To-Be）を作成する．そしてその新しいシナリオをロールプレイやプロトタイピングなどの手法を用いて評価する．ペルソナの代表に参加してもらうのであるが，良い評価が得られない場合は，2. DEFINE に戻ってアイデアの再検討を行い，再び DEVELOP を実施する．

数度の Try & Error を繰り返し，良い評価が得られそうだということになれば，4. DELIVER の事業プランの作成にとりかかる．

その最初に行うのがビジネスモデルキャンバスの作成である．必要に応じてピクト図でビジネスの流れを補足して，スポンサーや関係者に対して事業プランの説明を行う．OK が出れば，具体的な組織設計や業務設計，システム設計など構築フェーズに入ることになる．

3.2.2 サービスデザインからマイクロサービスへの展開

3.2.2.1 サービスブループリント

カスタマージャーニー(To-Be)は顧客接点を中心に設計していることから，background プロセスも含めて実現可能な解とするため，図-8 に示すサービスブループリントを使って，詳細化を行う．

Actions はカスタマージャーニーで設計したものと同じであるが，顧客接点では具体的にどのようなデバイスを用いるのか，それを提供するのは誰（何か），そのためにどのような処理をバックグラウンド側で行うのかなどを設計するようになっている．

ここに示す例でも，フロント部分は SNS やサイネージ，Web サイトなど様々なシステムで，バックグラウンドとして業務システム等が関連しており，異なる Pace layer のシステムが連携している．

DX 時代になって，フロント部分にアプリ等を使ったセルフサービス型のモデルが導入され，パーソナライズ化のために戦略システムが組み込まれるなど，よりシステム依存度が増すようになってきている．

そのため，サービスブループリントの新たな拡張が提案され，実際の適用例もでてきている．

その内容を次に紹介する．

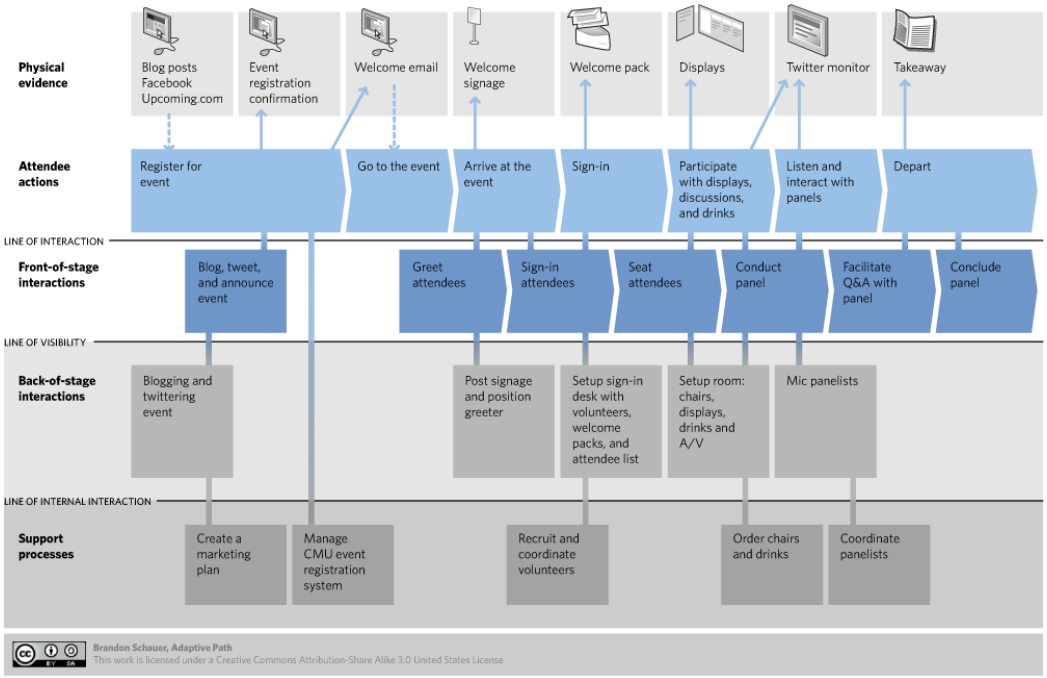


図-8 サービスブループリント(イベントマネジメントでの例)[10]

3.2.2.2 サービスブループリントからマイクロサービスへ

ダブリンの Wipro Digital からマイクロサービス環境を前提としたサービスブループリントの拡張が「Rethinking Service Blueprints for Agile Delivery」として提案されている. [11]

彼らの提案するサービスブループリントの縦方向の構造について図-9 に示す.

最上位にはカスタマーのアクションと関連するサービス提供側に関連するアクター(ロール)がリストアップされており, そのスイムレーンには, カスタマーやサービス提供者等がとるアクションに対して行うアクティビティを記述する.

その下段にある System Functionality がユーザーストーリーを記述するところになる. Develop フェーズでプロトタイプを作成し, おおよそのアーキテクチャ設計も済ませていることから, この例ではそこで定義した具体的なシステムも記述されている(The Hub と Tableau Reports).

記述されたユーザーストーリーを受けて, フロント側の機能を記述し, その実現のために利用するマイクロサービスとその機能をサービスレイヤーに記述する. 既存のマイクロサービスでは機能が不足する場合には, API 層に必要となる新たなサービスに求められる機能を記述し, DATA 層にはその新サービスが DB に対してどのような操作をするのかを記述するようになっている.

3.2.3 サービスマッピングにおける粒度問題

フロント機能とマイクロサービスとのマッピングにおいて課題となるのがサービスの粒度である. マイクロサービスの粒度が大きすぎても小さすぎてもマッピングが難しくなり, 適切な粒度でマイクロサービスが定義・登録されていることが必要となる.

4. SOA 開発プロセスのマイクロサービス開発への適用

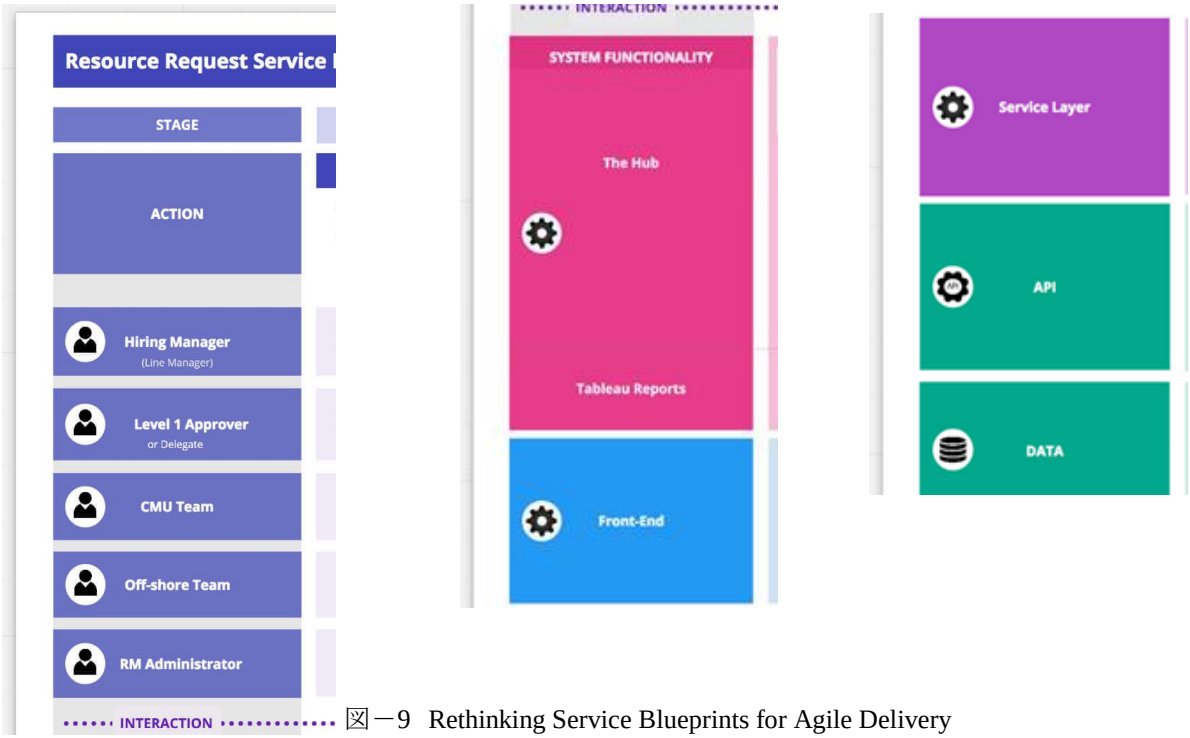
4.1. SOA 開発における粒度問題とその解決方法

4.1.1 SOA 開発における粒度問題

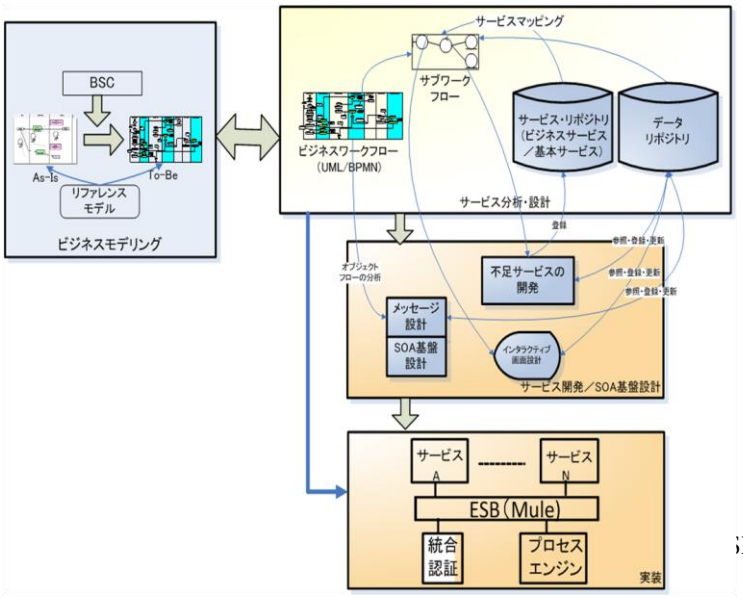
特定の業務領域を対象とした SOA 導入を進めた場合, 粒度問題というものは意識されることはない. 一連の業務フローをひとくくりのサービスとして定義しようと, 認証サービスといったように特定の機能をサービスとして定義しても, 特定業務領域内では問題とはならない.

粒度問題が明確化するのは, 複数の業務領域をまたがるシステムを構築しようとする時である.

業務領域によって, サービスの粒度が異なると事実上プロセス連携の設計はできないといった深刻な問題が生じる. ましてや, 企業間連携となるとこの問題はさらに深刻となる.



図－9 Rethinking Service Blueprints for Agile Delivery



ESB: Enterprise Service Bus

図-10 SOA の開発モデル

4.1.2 考案した開発モデル

4.1.2.1 開発モデルの概要

粒度問題を解決するために設計した SOA(Service Oriented Architecture)の開発モデルを図-10 に示す. [12]

この開発モデルは以下のオーソドックスな SOA アプローチを前提としている.

① プロセス階層を配慮しながら経営戦略を実現する To-Be(あるべき)ビジネスプロセスを設計する.
② レベル 3 のプロセスについて業務フローを作成する.
③ 業務フローのアクティビティをもとにサービスの括りだしをする.
④ 括りだしたサービスに対して既存のサービス群から割り当てができる場合には, 直ちに新しいビジネスプロセスを動かすことができる.
⑤ サービスが不足する場合には, そのサービスを実現するコンポーネントの開発のみを行う.
⑥ アクティビティ間のメッセージをもとにデータ設計を行う.

サービス粒度を揃えるために, 以下の4つの工夫を加えている.

① 詳細プロセスリファレンスを用いてビジネスプロセスモデルを設計して, ビジネスプロセスの粒度を揃える
② システムレーンのアクティビティをサブアクティビティ図を用いて BCE モデルに分解し, コントロールアクティビティに対してサービスをマッピングする)
③ 類似のサービスを構築しないようにリポジトリィによるサービス管理を行う.
④ サービスを, 業務フローをモデル化したプロセスサービス(BPM エンジンの読み込み対象となる), プロセスサービスから呼び出され DB への処理等を行うビジネスサービス, 認証認可などの基本サービスの3つに分類し, リポジトリィに登録し, 再利用の対象とするのはビジネスサービスと基本サービスのみとする. 粒度の大きいプロセスサービスを再利用性は低いからである.

4.1.2.2 サービスマッピングの方法

ビジネスモデラーが行うサービスマッピングは以下の拡張を UML モデリングツールに加えている. (図－11)

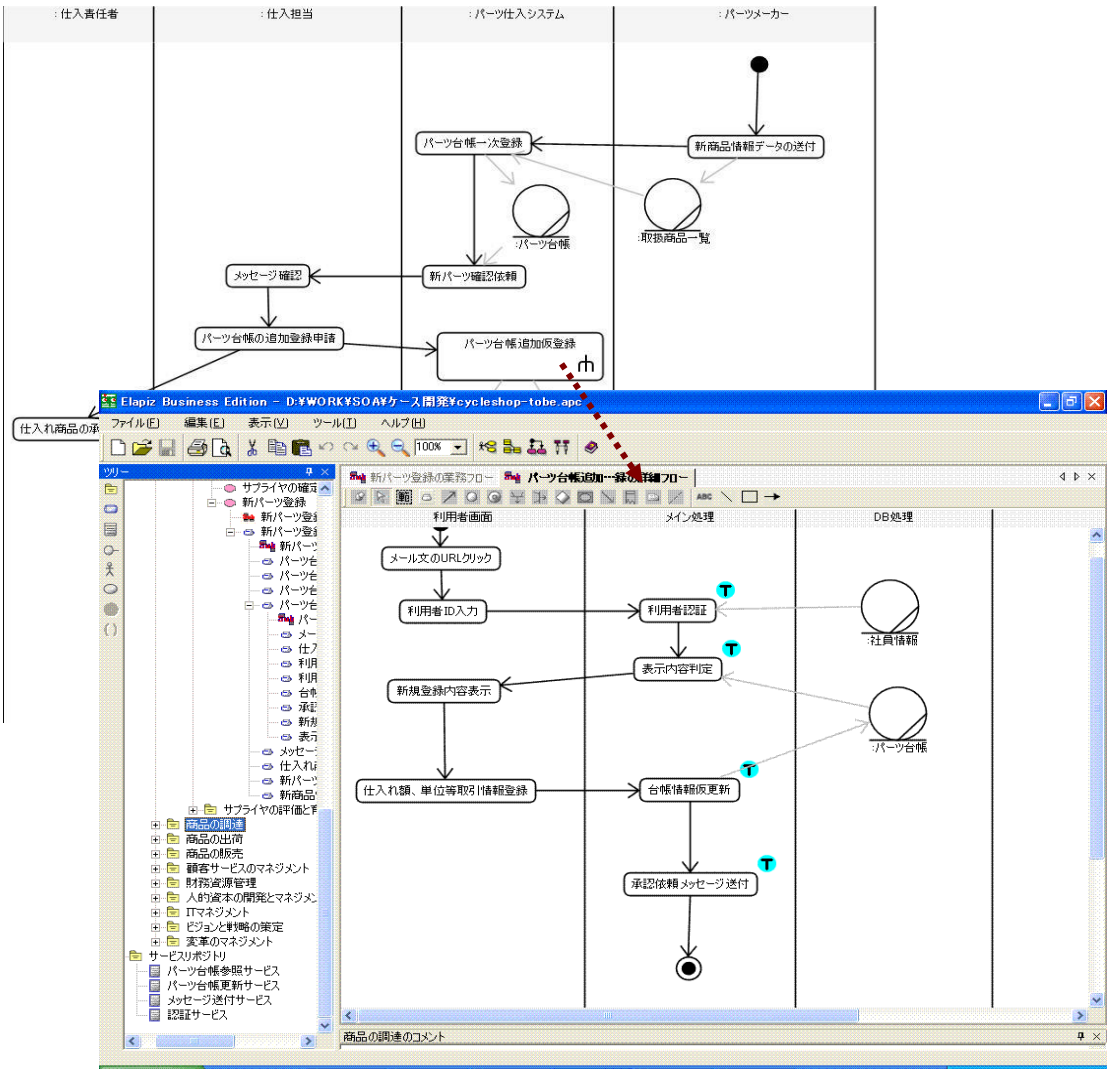
①サブアクティビティ図のフォーマット定義

通常ビジネスプロセスモデルを用いてシステムの要件定義を行う際, アクティビティ図のシステムのスイムレーンに示されるアクティビティをシステムユースケースとし, 分析モデル(BCE クラス図)を作成している.

クラス図ではフローを表記できないので, サブアクティビティ図を用いることとし, 図－11 に示すようにスイムレーンに BCE を割り当てた.

②トレース機能

アクティビティとサービスとの関係定義が必要となる. 図－11 のツリー図に示しているサービスリポジトリに登録されているサービスクラスとサブアクティビティとの結びつきを定義できるようにした. (Tマーク)



図－11　UML モデリングツールを使ったサービスマッピング

4.1.2.3　サービスマッピングからサービス設計の手順

ビジネスモデラーが実施したサービスマッピングを受けで下記の手順でサービス設計を実施する.

1.サービス分析
- サービス候補分析
- 概念モデル設計
- サービス候補の洗練
- サービス分析
- メッセージ分析
- サービスプロトコル分析

2.サービス設計
- サービス設計
- メッセージ設計
- サービス定義書作成

この中でサービスの粒度調整を行っているのが，サービス候補分析である. ここでの作業内容は下記のようになる.

「抽出されたサービス候補を整理する. このときの観点はサービスの粒度を揃えることと，再利用性である. この作業を行う際の指標となるのが，ビジネスエンティティに対する CRUD(create, read, update, delete)操作である. 例えば，複数のサービス操作候補から同じビジネスエンティティへ参照(Read)がある場合は，1 つのサービスにまとめる. サービス操作候補が複数のビジネスエンティティに操作を行っている場合，そのサービス操作候補は 1 つのサービスとし，複数のサービス操作に分割する.」

この過程を経て，工夫の四つ目に示したサービスを3つのグループに分けて定義することとした.

4.2. マイクロサービスの粒度問題との相違点

4.2.1 下流の粒度問題との相違点

4.1.2.3 及び 4.1.2.3 で示した手順は，サービスの独立性を高める手順である．この問題意識は 2.1 に示したマイクロサービスで懸念されている粒度問題と同じものである．

機能単位ではなく，画面フローに展開してからサービス候補が示され，ビジネスエンティティに着目して粒度を決めていることから，適切な粒度で設計するという要求にそのまま適用できるものではないかと考える．

4.2.2 上流の粒度問題との相違点

両者とも下記の共通のアプローチでサービスマッピングを行っている．

- サービスライブラリィを前提としている
- 上流で定義されたアクティビティに対し既存サービスをマッピングする
- 適当なサービスがない場合は，新たなサービスを開発する
- フロント，バックエンド，DB の処理を定義している

マッピングの際に適切な粒度で定義されたサービスライブラリィが欲しいという要求は共通している．

最も異なるのが，SOA はビジネスプロセスモデルを基にしていることであり，マイクロサービスはサービスブループリントをベースにしていることである．

SOA でサービス粒度を揃える起点は，ビジネスプロセスモデルを定義する際に，詳細プロセスリファレンスモデルを参照することである．この詳細プロセスリファレンスモデルは，ポーターのバリューチェーンモデルを第一階層として，ビジネスプロセスを最大第 4 層まで展開し定義したものとなっている．

サービスブループリントは顧客のタッチポイントを切り口としてプロセスを展開しており，この上流のモデリング手法の違いが適用に当たっての検討課題となる．

4.3. 適用に向けての拡張要件

4.3.1 適用に向けての方針

サービスブループリントの元になる To-Be のカスタマージャーニーマップは，複数の業務をまたがるアクティビティ図とほぼ同様のものとなっており，内容的には同一視して良い．

サービスブループリントからのアプローチでは，実はどのようにマイクロサービスをマッピングするのかが不明瞭である．SOA のアプローチでいうとビジネスプロセスモデルのシステムのレーンのアクティビティに対して直接サービスをマッピングしようとしていることと同様のことをしようとしている．

SOA 開発における粒度コントロールの鍵は，リファレンスモデルを参照して，業務フローを作成するビジネスプロセスの粒度を揃えることと，システムレーンのアクティビティを BCE モデルに展開し Control にサービスをマッピングするところにある．

マイクロサービスの設計において，粒度を整えるには，SOA 開発におけるこの2つの粒度コントロールを取り込むところにある．

4.3.2 具体的な拡張方法

前述のように，マイクロサービス様に筆者らの開発した SOA 開発プロセスを拡張するのではなく，粒度コントロールの2つの要素をサービスデザインから始まる開発プロセスに組み込むことが求められる．

筆者らの方法論は実はオリジナルの UML モデリングツールを前提として成立している．

このため，具体的な拡張は，以下の2つの観点で

① リファレンスモデル参照プロセスの組み込み

これについては，サービスブループリントのデザインにおいて，バックエンド側のプロセスを都度設計するのではなく，バックエンド側については，SOA の方法論を用いてビジネスプロセスモデルを予め設計しておくことで対応する．

② BCE 展開プロセスの組み込み

これについては，サービスブループリントを UML のアクティビティ図で記載できるように UML モデリングツールを拡張する．サービスブループリントは独自のレイヤー構造を持つが，これをスイムレーンにおいて表現できるようにすれば，アクティビティの BCE 展開，サービスマッピング，サービス設計のプロセスへと進めることが可能になる．

横書きと縦書きの違いはあるが，構造が類似しているので，拡張は難しくないと考える．

5. まとめ

本論では DX において必要とされるマイクロサービスベースの開発に対して，日本ではほとんど議論されていない上流のプロセスについてサービスデザインを適用し，サービスブループリントを経由して，マイクロサービスにつなげる手法を紹介し，その詳細な検討に筆者らが開発し

た SOA 開発プロセスを組み込むことで，課題となっているサービス粒度問題をも解決できそうであることを示した．

次ステップとして，検討結果を反映した UML モデリングツールの拡張を行い，マイクロサービス開発を実際に実施することでその妥当性を検証することとしている．

なお，マイクロサービスの利用にあたっては，実装環境の整備の課題がある．図-5 に示したように，異なるレイヤーのシステムを相互接続させる必要があるが，この点についての具体的なアーキテクチャの設計が求められる．ベンダーからの提案ではあるが，図-12 に示すような相互接続アーキテクチャが mulesoft 社から示されている[13]．実際の開発にあたっては，実装環境も大きな影響を与えることからジェネリックなモデルの検討も進めたい．

参考文献

[1] Gerald C. Kane, “Digital Maturity, Not Digital Transformation”, http://sloanreview.mit.edu/article/digital-maturity-not-digital-transformation/?article=digital-maturity-not-digital-transformation&post_type=article

[2] Gerald C. Kane, Doug Palmer, Anh Nguyen Phillips, David Kiron, and Natasha Buckley, “Aligning the Organization for Its Digital Future”, MIT Sloan Management Review RESEARCH REPORT SUMMER 2016.

[3] https://info.couchbase.com/2017_CIO_Survey_Report_LP.html

[4] Gerald C. Kane, Doug Palmer, Anh Nguyen Phillips, David Kiron, and Natasha Buckley, “Achieving Digital Maturity”, MIT Sloan Management Review RESEARCH REPORT SUMMER 2017.

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

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

[7] Fowler, Microservices, https://martinfowler.com/articles/microservices.html, 2014/3/25

[8] http://enginegroup.co.uk/approach/, 2015/05/01

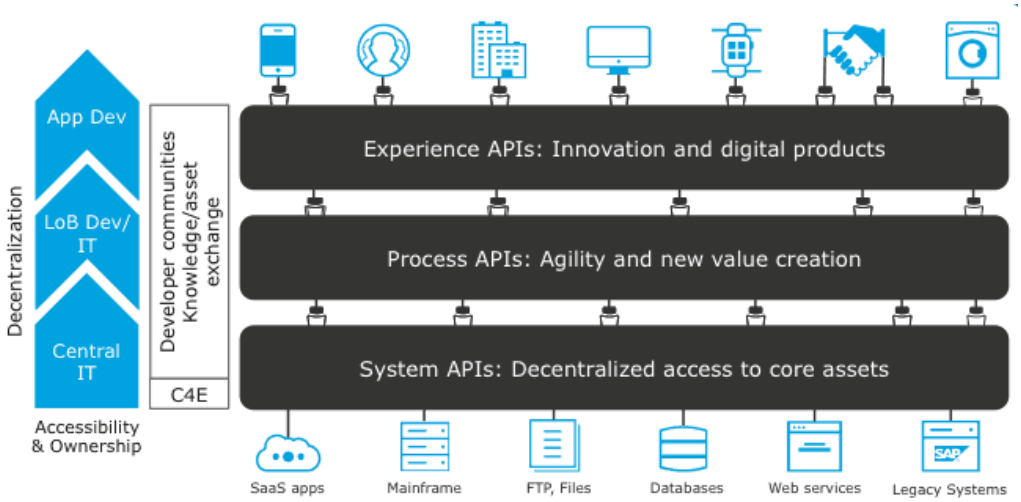
[9] マーク・スティックドーン，ヤコブ・シュナイダー ，長谷川敦士，武山政直，渡邉康太郎（監修），郷司陽子 （翻訳），「THIS IS SERVICE DESIGN THINKING. Basics − Tools − Cases - 領域横断的アプローチによるビジネスモデルの設計」，ビー・エヌ・エヌ新社，2013

[10] https://www.flickr.com/photos/brandonschauer/3363169836/sizes/o/

[11] https://wiprodigital.com/2018/08/30/rethinking-service-blueprints-for-agile-delivery/，2019/3/7 に参照

[12] 齋藤伸也，宗平順己，大場克哉，「UML モデルを活用した SOA の設計・実装プロセスの検証」，研究報告ソフトウェア工学（SE） 2009-SE-166(15)，1-8，2009-10-29

[13] https://blogs.mulesoft.com/dev/api-dev/what-is-api-led-connectivity/



図―12 DX 対応のアーキテクチャ例

データベース・アプリケーションへの実行時モデル検査導入手法の提案

宮永 照二
(株) テクノネット
myngshj3@gmail.com

要旨

情報システムの急速なかつ広範囲な普及に伴って，情報システムの製品の品質とプロセスの品質の両方において，新たな適用方法が提案されている *[2][3]*.

情報システムは開発・運用プロセスにおいてライフサクルを持つが，このライフサイクル内の各工程間でシステムに関する知識が共有されていないために様々な問題を引き起こしている *[2]*. 特にシステムの重要な性質についてテストすら完全にできない状態が存在することも報告されている *[2]*.

このような状況でシステムの重要な性質を検証する技術として，実行時モデル検査 *(On-the-Fly Model Checking)* がある *[7] [9] [10]*.

本稿では，情報システムとくにデータベースアプリケーションについて，実行時モデル検査を効果的に適用する方法について提案する.

1. はじめに

1.1. 情報システムにおける品質保証の状況

情報システムは，研究目的の一時限りの開発 (One-Time) ではなく，ビジネス上の要求にもとづくシステム開発ではその品質とコストと適用効果が考慮され，ある一定のタイムスパンを設けてライフサイクルが設定される (システムに対してライフサイクルという言葉を用いていることについては，システム，ライフサイクル概念，システムライフサイクルステージ，システムライフサイクルの典型について，[4] にその概要が記載されている．). 特に長いライフサイクルをもつレガシーシステムでは，計算機資源やアプリケーションの拡張によりパフォーマンスやユーザビリティの改善をもたらす一方で，ライフサイルのステージ間で知識の共有がなされないために，さまざまな問題を引き起こすことが報告されている [2].

レガシーシステムだけではなく，新規開発のシステムについても，レガシーシステムと同様にステージ間の知識の共有がなされず，1st リリース経過後の 2nd リリース，3rd リリース等の後期リリースで同様の問題が発生することが懸念される.

これらの問題の多くは，大規模化複雑化するシステムに対して，十分な要求や設計を吟味する時間が短くなる傾向があり，そのために，時間が経過するにつれて，ドキュメントとソースコードのギャップ，ドキュメントと実際の実装のギャップやドキュメント自体の非存在などが発生することが原因となっており，多くの場合，テストによる品質の担保すら困難な状況が発生している [2].

これは情報システムの製品としての品質が保たれていないだけではなく，開発プロセスの品質が担保されていないために発生している問題である.

さらに，現在は情報システムが社会インフラの 1 つとして認識されており，その信頼性や安全性という概念は重要な意味をもつ. 単に設計通りに動作するシステムを構築するというだけにとどまらず，フォールトトレラント・フェールセーフなどの運用やメンテナンスのステージにおける信頼性や安全性を含んだトータルな品質が要求される [6].

1.2. モチベーション

このような問題に対して，厳密なプロセスを経て，正しいソフトウェアを作る方法としてフォーマルメソッドがある [1] (ここでは正しいソフトウェアを仕様や想定通りの動作をするソフトウェアという意味で用いている).

フォーマルメソッドは数理的な技法を駆使して厳密なプロセスのもとに正しいプログラムを作成するための総合的な技術であるが，その用途は単に正しいソフトウェアを系統的に作成するだけにとどまらず，システムとしての性質を検証するのにも適用できる [1]．(正確にはシステムの性質を検証するプロセスはソフトウェアの信頼性を保証するためのプロセスに組み込まれており，そのプロセスに数理的な技法を適用している．)

情報システムはそれが大規模化複雑化すればするほど，仕様全体を把握することが困難であり，そのライフサイクルの各ステージ間で同じ知識を共有することはなおさら困難である．

このような状況に対して，システムの安全性などの重要な性質を実行時に検証する方法として，モデル検査 (Model Checking) がある．モデル検査は，検査対象のモデルを記述し，そのモデル上でシステムの重要な性質，例えば起こり得ない状態や安全性を脅かしかねない状態，あるいは重要な問題が発生した状態などを報告することによって，システムの品質向上に役立てることができる．モデル検査は、システムがとりうる状態を網羅的にたどることにより、システムの重要な性質が満たされることを検証する手法であるが、検証対象のシステムの規模によっては大きな状態空間とそれら状態間のネットワークを構築する必要があるため、性能の問題が顕在化する．特にこの性能の問題に対して，On-the-Fly モデル検査 (On-the-Fly Model Checking) [9] [7] [8] が提案されており，現実のシステム開発へ適用する研究もされている [8]．

複雑なシステムをモデル検査するのに，On-the-Fly モデル検査はパワフルで有効な手法である [9]．On-the-Fly モデル検査は，状態空間とそのネットワークを構築しながら，性質の検証を実行する [9] [10]．設計の早期の検証ステージで大量のバグを検出することができるため，本質的に設計プロセスでの適用に向いている [9]．

モデル検査のような数理的な技法による検証プロセスを実際に導入する場合，コストが余計にかかるというデメリットが存在する [3] モデル検査を適用することによる品質の向上という大きなメリットを生かすためには，通常のプロセスに自然と組み込まれているように適用されるのが好ましいと考えられる．たとえば，検証のための追加の労力を極力課さずに設計検証を行う方法として UML により記述される設計をモデル検査する方法の研究があるが [5][8]，これらも検証のための設計と製造のための設計を分離しない，つまりに二度手間を排除し効率的にモデル検査を適用しようとする事例である．

UML の使用の有無に限らず，モデル検査のための設計プロセスは，追加の労力を課さず，なおかつ専門的な知識の習得を極力排除して導入することが望ましいが，一般的なデータベース管理システムの機能を利用すると，比較的容易にシステムの動的な検証が可能である．そこで，データベースアプリケーションにフォーカスを当て，その構成や性質から，検証用のコードを追加する労力を極力抑えて，動的な検証，すなわち実行時モデル検査を実施する方法を考案した．

本稿では，その実現方式について述べる．

2. データベース・アプリケーションの特徴

2.1. リレーショナル・データベースとデータ構造

データベース・管理システムの４つの重要な特性，すなわち，ACID 特性 (Atomicity, Consistency, Isolation, Durability) はデータベース管理システムが提供する最も重要な性質である．これらの性質は，いくつかの構造を保全した情報システムの開発を容易にしている．

データベース管理システムをアプリケーションの中心に位置づける大きな要因は，大きく分けて次の点に分類されると考えている．

1. データについて構造を保全した記録と抽出が可能
2. 大量データの扱いについて，性能のチューニングが容易
3. バックアップ・リストア・レプリケーション等の管理が容易

1. は，リレーショナル・データベースのように何かしら存在する実体（エンティティ）間の関係を保存し，満たすべき関係を破壊する操作をキャンセルできる．この性質は重要な点である．既約なエンティティとそれらの関係からなるオブジェクトを統一的に記述できるアドバンテージは大きい．この構造は通常，静的な構造・リレーションを基本とする値の制約だけでなく，アプリケーションレベルの制約を与えることで構造を破壊するための起こるアプリケーションエラーを回避できる．

2. は，大量データを扱うアプリケーションに対して，特に個別のアルゴリズムの開発を行うことなく，性能を

改善することができる．実際には，その用途に応じて，キャッシュやインデックス等の構成を設定したり，検索のオプションの記述を追加することで，並列化等のアルゴリズムを用いて高速化をはかることができる．昨今のデータベースは大量のデータを保存し再利用する用途が多くなっており，それに対してデータ量に応じたアルゴリズムの切り替えはアプリケーションの仕様に影響を及ぼすため実施しがたい．データ構造やアプリケーションのアルゴリズムに影響を与えず，高速化をはかるためのチューニングの機能は重要である．

3. は，管理面の効率化に寄与する．データベースをバックアップする主な理由は，障害発生時の復旧にある．障害発生時にはバックアップ記録から，システムを安全裏に回復するための機能が必要である．

アプリケーションの品質にもっとも大きく影響するのは，やはり，1. データ構造の保全機能である．リレーショナル・データベースは前述の通り，エンティティ間の関係を保持し，エンティティ間の参照関係や依存関係を記述することができる．さらに，その使い方次第では，参照数の制限等の制約の追加などにより，より高い表現力を与えることができる．

例えば，グラフの構造を考える．グラフは構造 $G = (V, E)$ で与えられるが，最もシンプルな構造，すなわち頂点と辺の定義のみに基づく構造のみでは，任意のノード間に双方向の参照をもつグラフが記述される．しかし一般に，アプリケーションの持つデータ構造は，例えばそれは木構造であったり，サイクルであったり，またループを持たないものであったりする．そこでは，単なるグラフ G に対して，いくつかの制約を与えなければならない．しかもそれは，アプリケーションの実行のタイミングで，部分グラフ $G_1 \subset G$ が，組み込まれた別の構造を持つことがある．単なる静的な制約では実現できないこれらの構造の保全機能は，シンプルなプログラムを組み込むことで事前に実装可能となる．組み込み可能な構造についていくつか例を以下に挙げる．

1. 連結グラフ : 任意の 2 つのノードを結ぶパスが存在する
2. 二分木 : 連結グラフであり，かつ，すべての頂点 v に対して，v を根とする辺は存在しないか，または存在すればその数は 2 である
3. サイクル : ある頂点 v から同じ頂点 v へのパスが存在する

2.2. アプリケーションの仕様と有限オートマトン

データベース・アプリケーションはその名のとおり，そのデータをデータベースに保存するアプリケーションである．アプリケーションはデータベースの ACID 特性の恩恵により，アプリケーションを効率的に開発することができる．特に構造を保全する機能を導入することにより，重要なデータ構造の破壊をもたらす機会を除去することができる．

一方で，アプリケーションは有限状態機械とみることができる．例えば，一般的な Web アプリケーションは複数の画面と画面の背後にある，GUI と分離された機能からなり，ユーザは画面に対して操作し画面を切り替えながら操作する．ある 1 つの画面とその背後にある意味のある機能の組を 1 つの単位として，なになに画面のなになに状態，などと名付けることによって，アプリケーションの状態と関連付けることができる．

アプリケーションを有限状態機械とみなしたとき，それをデータベース内にグラフ構造を保存する機能を用いて実装することはあまりないと考えられる．それは実現可能であり，そのようなニーズも存在するかもしれないが，一般的にはアプリケーションの取りうる状態の全体をそのままデータベースに保存してなにかしらの機能を実現するために用いているとは考えづらい．

その一方で，有限状態機械を採用すれば，システムにその性質に対する高い表現力を与えることができる．ここで，有限状態機械の例として決定性有限オートマトンを挙げる．決定性有限オートマトンの定義は以下のとおりである．

$A = (Q, \Sigma, \delta, q_0, F)$．ここで，$Q$ は状態の集合，Σ は文字の集合，δ は遷移関数，q_0 は初期状態，F は受理状態である．

いま，このオートマトン A に対応するラベル関数 L を以下のように定義する．

$L : Q \cup \Sigma \to S$．ここで S は文の集合である．

すると，このオートマトンは図 1 のような概観をもつ．

いま，関数 L はオートマトンの取りうる状態や受け付ける文字から文への対応を与えるため，文 $\phi, \psi \in S$ にある意味を与えることにより，プログラムのモデルであるこのオートマトンに重要な意味を与えることができる．

$S = L(Q) \cup L(\Sigma)$ とする．$L(s)$ はある状態 $s \in Q$ に対応する文であり，この文が状態 s で常に成り立つ式

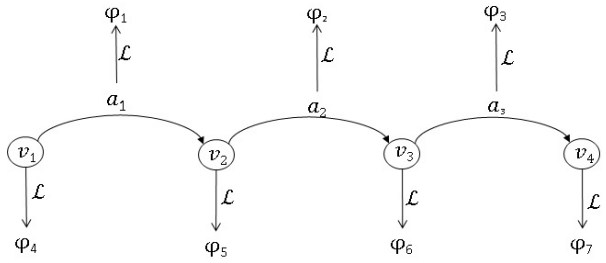


図 1. ラベル付きオートマトンの概念図

であるとする．また，$L(a)$ をある文字 $a \in \Sigma$ に対応する文であり，この文が文字 a に対応するプログラム (命令) であるとすると，オートマトン A はある命令 $L(a)$ を入力として状態遷移する状態遷移システムを表すとみなすことができる．さらに，$L(a)$ を正規言語に限定し，オートマトン A を非決定性オートマトンに拡張すると，このオートマトンは一般化非決定性有限オートマトン (Generalized Non-deterministic Finite Automaton: GNFA) を構成するとみなせる．

そこでいま，この GNFA を構成するための構造をリレーショナル・データベースに与えるために，次のようなテーブルを定義する．

表 1. 状態テーブル (States)

s	$label$
s_1	ϕ_1
s_2	ϕ_2
.	...
.	...

表 2. 文字テーブル (Alphabets)

a	$label$
a_1	ψ_1
a_2	ψ_2
.	...
.	...

このとき GNFA が矛盾なく構成される構造は以下の性質を満たす．

・状態のユニーク性: $\forall s, t \in States \circ s \neq t \rightarrow s.s \neq t.s$

表 3. 遷移テーブル (Transitions)

u	a	v	$remarks$
s_1	$a_{1,2}$	s_2	transition from v_1 to v_2
s_2	$a_{2,3}$	s_3	transition from v_2 to v_3
.	.	.	...
.	.	.	...

表 4. 現在状態テーブル (CurrentState)

s	$remarks$
s_x	$state - x$

・遷移状態の参照制約: $\forall t \in Transitions \circ t.u, t.v \in States \wedge t.a \in Alphabets$

・現在状態のシングルトン性: $\|CurrenState\| = 1$

以上で，GFNA の静的構造は完成する．このオートマトンが入力により状態遷移するには，状態遷移を発生させるための次の機能の記述が必要である．

・遷移実行処理: $doTansition(a \in Alphabets)$

2.3. アプリケーションを実行時に検証する仕組み

前項で説明した構造をもつ GNFA は，ある状態 $s \in S$ のときに成り立つ文 $L(s)$ を s で成り立つ式とみなす．また，ある文字 $a \in \Sigma$ のときに成り立つ文 $L(a)$ を状態遷移の契機となる命令とみなす．この構造を利用すると，図 1 であらわされるオートマトンの実行は，$\psi_1, \phi_1, \psi_2, \phi_2, \psi_3$ という文のシーケンスを表示する．言い換えれば，一連の文 $\psi_1, \phi_1, \psi_2, \phi_2, \psi_3$ はアプリケーションの動作シナリオとして定義される仕様の一つであり，このシナリオに基づいたアプリケーションの動作を検証することができる．

この例では，状態 s_1 にあるときに文 ϕ_1 を与えると，状態 s_2 に遷移し，さらにそのときに文 ϕ_2 を与えると，状態 s_3 に遷移するというシナリオを与えることができる．

これはちょうど，状態 s_1, s_2, s_3 をそれぞれ画面 1, 画面 2, 画面 3 に対応させ，文 ϕ_1, ϕ_2 をそれぞれイベント 1, イベント 2 に対応させると，前述のシナリオは,「画面 1 のときイベント 1 を発生させると画面 2 に遷移し，さらにそのときイベント 2 を発生させると画面 3 に遷移する」と言い換えることができ，操作仕様と対応付けられ

た仕様として記述することができる．この仕様は，オペレーションベースでかつ検証可能なため，先述したシステムライフサイクルのステージ間における知識の共有を促進する．

アプリケーションを実行時に検証する仕組みとしては，状態 s に対応する文 $L(s)$ を状態 s に遷移完了したタイミングで評価し，それが真となるか偽となるかを判定する方法が考えられる．より具体的には，文 $L(s)$ が表す SQL 文を実行し，その結果によって判定する方法が考えられる．

いま，Java や C#のようなオブジェクト指向言語で採用されている try〜catch ブロックによる例外捕捉のメカニズムの記法を導入し，状態遷移に伴う処理を以下のように記述したとする．

$\psi_1, try\{\phi_1, \psi_2\} catch\{\phi_2, \psi_3\}$

この処理のシナリオは ψ_1, ϕ_1, ψ_2 と $\psi_1, \phi_1, \phi_2, \psi_3$ の 2 つである．

2.4. 一階述語論理と時相論理による性質の記述

アプリケーションは一般的に，複数の異なる状態を取るが，各状態においてアプリケーションのもつデータがある性質を示している場合を考えると，

$s \rightarrow \forall o \in Objects \circ o.p$

のような，ちょうど状態 s にあるとき，$Objects$ のインスタンス o の性質 p を満たすというような一階述語論理の記述ができる．

さらに，アプリケーションの性質の意味論としてに有限オートマトンによる定義を与えると，例えば,「ある状態 s から，入力 a によってクリティカルな状態 s_c に到達する関係が存在する」という意味の到達可能関係 $s \overset{a}{\rightsquigarrow} s_c$ を反例とする，時相論理式

$\Box\neg\forall s \forall a \circ s \overset{a}{\rightsquigarrow} s_c$

を，有限オートマトンのもつ構造から検証することができる．この時相論理式の直感的な意味は,「どんな状態にあっても，どんな入力を与えても，状態 s_c に到達することはない」である．

2.5. コンカレントシステムの検証

マルチスレッドやマルチプロセスのシステムでは，複数のタスクが非同期に稼働する．分散処理においては，ひとつのデータベースに複数のタスクが同時に書き込みを行うが，書き込みの衝突を回避するために，同じ領域への書き込みを同時に行わないような制御がなされる．そのために，同一領域への書き込みリクエストを同時に処理しないような方式を採用する．

いま 2 つのプロセスを状態遷移機械で表現し，2 つのプロセスが同時に同じデータへの書き込みをしないかどうかを検証しようとする場合を考える．

プロセス 1 とプロセス 2 の計算モデルをそれぞれ，$A_1 = (Q_1, \Sigma_1, \delta_1, q_{1,0}, F_1)$, $A_2 = (Q_2, \Sigma_2, \delta_2, q_{2,0}, F_2)$ とする．

これらの並行プロセスのモデルをオートマトンの積 $A = \prod_i A_i$ を定義することであらわすことにする．ここで，

$\prod_i A_i = (\prod_i Q_i, \prod_i (\Sigma_i \cup \{\epsilon\}), \prod_i \delta_i, \prod_i q_{i,0}, \prod_i F_i)$,

$\prod_i \delta_i = \prod_i Q_i \times \prod_i (\Sigma_i \cup \{\epsilon\}) \rightarrow \prod_i Q_i$

である．

いま，2 つのオートマトンがそれぞれの状態遷移のシーケンス，s_0, s_1, s_2, s_3, s_0 と q_0, q_1, q_2, q_0 を取るとして，図 2 のようなコンカレントな状態をとる並行システムを考える．

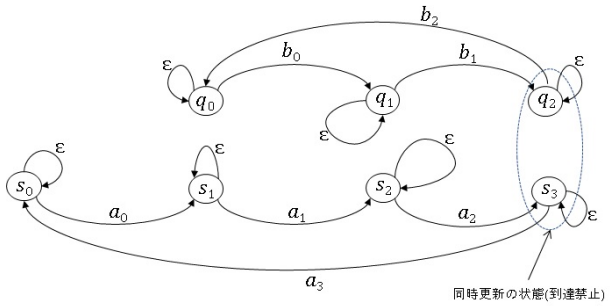


図 2. コンカレントシステムの状態図の概要

いま，A_1 と A_2 の積をとった時，現れる状態の一つ (s_3, q_2) は同時に書き込みが行われる状態であるとすると，この状態に到達しないこと，すなわち $G\neg(s_3, q_2)$ を検証するのが，実行時モデル検査の役割の一つである．

ユーザ $user$ と状態 $state$ の組 $(user, state)$ を保存し，集合 $\{(user, state)\}$ に $state = s_3$ なるエンティティと $state = q_2$ なるエンティティが同時に存在しないこと，すなわち一階述語論理式

$\neg(\exists u \circ u.state = s_3 \wedge \exists v \circ v.state = q_2)$

を満たすかどうかで検証できるし，このような性質を保全するために，状態遷移が確定する直前に，この一階

述語論理式が満たされない場合に，遷移をキャンセルするといった処理を組み込むことで，安全にコンカレントシステムの競合回避機能を実装することができる．

3. ケーススタディ

3.1. 電卓 GUI アプリケーション

トイ問題として，電卓 GUI アプリケーションを例に挙げる．

このアプリケーションの仕様は以下のようである．

画面仕様 : このプログラムは，1 つのスクリーンからなり，数字 : 0,1,..,9 に対応するボタンと演算子:+,-,*,/ に対応するボタンとクリアキー:C に対応するボタンとエンターキー:E に対応するボタンが配置されている．さらに，計算される式を画面に表示するための Widget と計算結果を画面に表示するための Widget からなる．

分析結果 : 電卓は一般的に，複数のキーを組み合わせて連続でキーインすることによって，計算したい式を指定する．さらに計算用のキーをキーインすることによって，計算を実行し，その計算結果を表示する．

計算式を指定するときには，計算式が適切な式になるように，キーインを受け付けるかどうかの制御を行う．例えば，すべてのキーインをすなおに受け付けてしまうと，計算式が例えば「3 + − 2」となるような，算出不可能な式を指定してしまう可能性があるために，入力状態に応じてキーインを受け付けるかどうかの制御を行う．

この分析結果を反映した結果，設計は以下のようになった．まず，このアプリケーションは以下の 3 つの状態を持ち，それぞれの状態からそれ自身または他の状態へ遷移する．

1. 初期状態 s_0 は，アプリケーションが立ち上がって最初の状態．初期状態 s_0 は，ユーザがなんのアクションをしなくても，初期化処理の後に次の状態 s_1 に遷移する．

2. 状態 1 s_1 は，数字キーイン待ち状態．状態 1 s_1 は，ユーザが最初に数字を入力するモードである．ここで数字を入力するとモード変更し s_2 に遷移する．演算キーインは受け付けない．C キーを入力すると s_0 に遷移する．また，このモードに滞在するときは，Enter キーを受け付けない．

3. 状態 2 s_2 は，Enter キーイン待ち状態．状態 2 s_2 は，ユーザが計算を実行可能な状態である．このモードに滞在するときは，演算キーインがあった場合は s_1 へ遷移し，クリアキーが入力されたときには s_0 に遷移する．Enter キーインがあった場合は計算を実行し，s_0 へ遷移する．

このアプリケーションの状態遷移システムは図 3 のようになる．

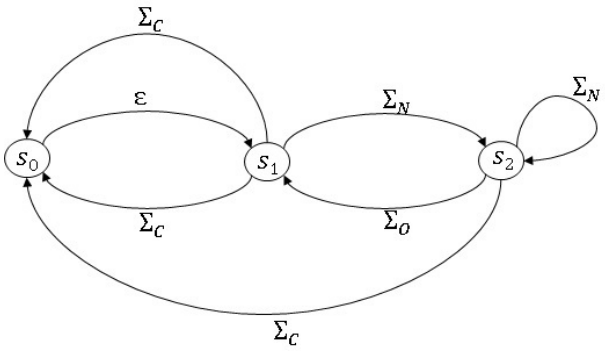


図 3. 電卓アプリの状態図

設計にならって有限オートマトンを構成すると，以下のようになる．

$Q = \{s_0, s_1, s_2\}$

$\Sigma_N = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$\Sigma_O = \{+, -, *, /\}$

$\Sigma_C = \{C, E\}$

$\Sigma = \Sigma_N \cup \Sigma_O \cup \Sigma_C \cup \{\epsilon\}$

$\delta = \delta_{s_0 \to s_0} \cup \delta_{s_0 \to s_1} \cup \delta_{s_1 \to s_0} \cup \delta_{s_1 \to s_2} \cup \delta_{s_2 \to s_0}$

$q_0 = s_0$

$F = \{s_2\}$

このとき検証したい性質は，Enter キーが受け付けられる場合には，計算式が適切であり計算可能な式を表している必要がある．いま計算式を μ とすると，μ は次のような正規表現のクラスに属する必要がある．

$\mu \in ((\Sigma_N^+ \Sigma_O^1)^* \Sigma_N)$

すなわち，このアプリケーションのもつ性質である，計算が正しく行われるための性質は，

$s_2 \to \mu \in ((\Sigma_N^+ \Sigma_O^1)^* \Sigma_N)$

となる．実行時にこのような性質を検証することによって，仕様の検証を行うことができる．

3.2. 既存仕様の記述:個人情報更新処理

トイ問題としてネットバンキングの個人情報更新機能を考えてみる．この処理では図 4 のようなステートチャートが想定される．

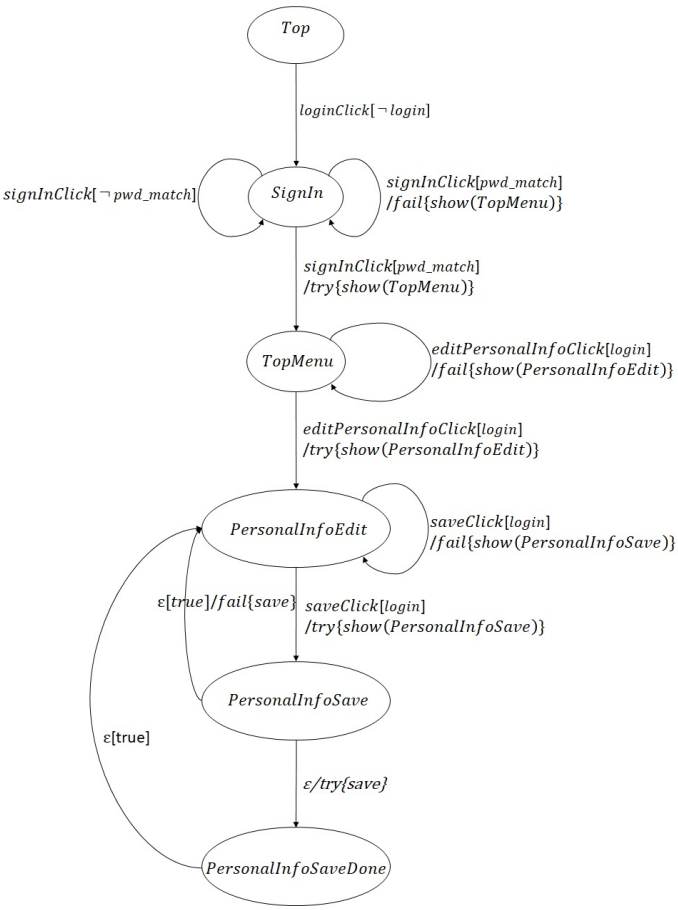


図 4. ネットバンキングのステートチャートの部分図

このログイン (サインイン) に関する性質には以下の性質があげられる．

1. 有効なログイン状態である $\equiv login$
2. 有効なログイン状態にない $\equiv \neg login$
3. Credential がマッチする $\equiv pwd\_match$
4. Credential がマッチしない $\equiv \neg pwd\_match$

このように定め，複雑な性質の記述を回避することにする．次にオートマトンを次のように構成する．

1. Step-1:ステートチャートから導き出される，静的に決定するオートマトンを構成する
2. Step-2:ガード条件のすべての性質を CNF または DNF に変換する
3. Step-3:時相論理に基づいて到達可能性判定を行う
4. Step-4:静的なオートマトンに対して，状態間の新たな遷移を追加したオートマトンを動的に構成する
5. Step-5:新しいオートマトンに対して性質を検証する

Step-4 と Step-5 を繰り返すことによって，On-the-Fly にオートマトンの構成とその上の性質の検証を同時に行う．

複雑なアプリケーションになればなるほど，静的な構造はそれを適切にモデル化することが難しく，実際のプログラムと乖離したモデルを作成してしまうことも往々に存在する．

実行時に，実際に実装された構造を構築するメリットは，本来ある仕様記述の不備や，想定されない振る舞いを検証する機会を与える．

この場合，あるときに成立していた性質が，その後のあるときには成り立たないことが起こりうる（到達可能な経路が現れる）．そのため，オートマトンが構成されたタイミングで検証を行う．

本提案手法の，新規開発ではなく，リプレースや機能追加・改変への有効性・メリットは以下のことがあげられる．

1. プログラムのある状態で取りうる性質が変わることへ対応できる．
2. ステートチャートと状態遷移表で直感的オペレーション仕様は推定できる．
3. 状態を性質ベースで記述するため，プログラマレベルで認識しているステートとユーザレベル認識ステートの乖離を減少させられる．

3.3. 機能追加:多重ログイン許可機能

前述のバンキングシステムではログインユーザ 1 人に対するステートチャートを考慮していた．しかし実際には，複数のユーザが同時に操作する場合があり，場合によっては同時ログインしての作業を許す場合もある．

そこで，多重ログインを許可するデータ構造を導入し，その際に想定される満たすべき性質を検証する例を考える．

現行のログインユーザテーブルを仮定し，そこにユーザ ID とは別のクライアント ID を設けるように拡張し，

以下のような構造を与える．

表 5. ログインユーザテーブル (LoginUsers)

userid	*clientid*	*login_time*	*lastevent_time*
id_1	$client_1$	*login time*	*last event time*
.	.	.	...
.	.	.	...

このような構造を導入しないと，次のようなことが起こる．端末 A で操作しているときに，タイムアウトとなる時間が経過している．端末 B でログインして操作すると，タイムスタンプが更新される．端末 A での操作はタイムアウトのはずなのに，更新がかかっているために引き続き作業ができてしまう．

これを検証するには，以下の式を検証する．

ログインユーザのテーブル $LoginUsers$ があって，ある $userid = id_1$ のユーザが，$clientid = client_1$ で実行している場合を考える．その際のテーブルの構造として，$(userid, clientid))$ は主キーとなるように制約を与えられているとする．

$C = CurrentStates = \{(userid, clientid, state)\}$ とすると，ある $userid$ に対して個人情報の安全な書き込みが満たされる条件は，現在状態がクリティカルセクション $S_c = PersonalInfoSave$ に同時にないことであり，

$\forall u \in LoginUsers \circ$

$\|\{v \in C; v.userid = u.userid \wedge v.state = S_c\}\| \leq 1$

を満たすことである．これが満たされている場合，同時にクリティカルセクション $PersonalInfoSave$ にないために，同時に書き込みに入ることはない．したがって同時保存の状態を回避することができる．万が一クリティカルセクションに同時に入ってしまった場合，実際に入ってしまったことを ASSERTION で確認することもできる．

4. ディスカッションと結論

4.1. ディスカッション

本稿では，データベース・アプリケーションが状態遷移機械とみなせる場合のシステムについて，実行時に時相論理や一階述語論理に基づくモデル検査を行い，システムの安全性などの重要な性質が満たされるかどうかを検証する方法を提案した．

そのなかで，実際のケーススタディとして，電卓アプリケーションとコンカレントな操作が可能なネットバンキングの例を説明し、さらに実際の開発の現場で実施される機能拡張に該当する同時ログイン機能追加を例にして，モデル検査の方法を説明した．いずれにしても，本稿では極めてシンプルで状態の数や遷移パターンが人手で数え上げられるほどであり，現実のシーンではもっと多くの状態や遷移パターンが現れることは必至である．

さらに，検証すべき性質を満たすかどうかの式は一般に，一階述語論理式と時相論理式で表されるが，開発者は，例えば単体テストを行うときには，SELECT 文で必要なデータを抽出し，プログラムの正しい動作を検証するということを行うのが通常であろう．本稿の冒頭で述べたように，アプリケーションのもつ性質についての知識がステージ間で共有されていない現実に対応するためには，適切に入力したり，問題をアラートするためのフロント機能の提供が必要と考える．

4.2. 提案手法のアドバンテージ

本稿で述べた方式による利点は以下のものがあげられる．

1. アプリケーションへの実装の影響を極力抑えて，検証を可能とする
2. アプリケーションの実行時にシステム検証を行うことを可能とする

既に稼働しているシステムや，モデル検査等による設計検証を実施せずに実装しているシステムに対して検証を行う方式であり，まだまだ新しい技術であるモデル検査がコストを要する現状においては，本稿記載の方式はアドバンテージがあるといえる．

また，実装方式としては，静的な状態空間を作成せず，オンデマンドに状態空間とネットワークを生成する

On-the-Fly 方式を採用することにより，アプリケーションの実行と同期的にモデル検査を実施するという方式を提供できる点もアドバンテージがあると考えられる．

この点については，今後試験実装を進めることにより，本稿提案の方式の実用性を検証する必要がある．

本提案手法で懸念される問題は次項以降で説明する。

4.3. 性能面の問題

データベース管理機能を検証のプラットフォームとして利用するというアイデアは，実装の容易性や機能としての有効性とは相反する性能の低下を招くおそれがある．性能面の課題は以下のようなものがある．

1. 状態を記録するオーバーヘッドにどう対処するか
2. COMMIT によるブロックの発生による影響をどう抑えるか
3. ネットワークトラフィックの増加にどう対処するか

状態を記録するためにおこるオーバーヘッド，状態を確実に記録させるための記述量の増加とそのために発生する処理が懸念材料である．また，COMMIT によるブロックの発生頻度も考慮しなければならない．多数のユーザが利用するデータベースアプリケーションでは，ネットワークトラフィックの増加による検証への影響も考慮しなければならない。これらの性能低下を和らげる方策の一つとして，On-the-Fly モデル検査を採用した．On-the-Fly モデル検査は、検証が必要なときに必要な状態空間を生成するため，実行時のコストが低い．また状態遷移のタイミングで検証を行う必要があるが，複雑なアプリケーションや多数のユーザが協業するコンカレントシステムだと，これらのオーバーヘッドは避けられない．仕様理解やテスト等のステージでアプリケーションを Run させることにより性質の検証を網羅的に行い，性能が要求されるステージでは特に重要な性質のみに絞って検証するようにすれば，性能の問題を回避できると考えている．

検証のタイミングでブロックが発生することに関しては、例えば，集合演算を用いたデータベース管理システムの諸機能を，単一エンティティとそれに関連するエンティティに限定して操作するメカニズム，つまり

$\forall r \in Record \circ r.p$

という性質ではなく，

$\forall r \in Record \circ r.id = id \rightarrow r.p$

なる性質の検証を基本的に行うようなメカニズムを提供するようにすれば，性能が向上すると考えられる．

4.4. 性質の認識と記述

一般的にモデル検査は設計を検証するために行われており，既に製造された，あるいは製造中のシステムに対して検証を行うには，別途検証用のモデルを作成する必要がある．本稿で説明したような方式は，運用の点で以下のような課題があげられる．

1. 性質をどこまで適切に記述できるのか
2. 仕様としての正しさと想定外の振る舞いに対する対応をどうするか

性質をどこまで適切に記述できるのかという問題は，モデル検査で検証する性質の表現力の問題である．プログラマ目線で記述される検証式は、データ構造を熟知した状態の表現であり、単体テストで実行する検証式を流用し、ステージ間で共有することを想定している．より汎用で一般的な性質の表現の可能性を探る必要がある．

仕様としての正しさと想定外の振る舞いに対する対応は，完全な検証モデルの記述とトレードオフである．本稿では時相論理式と一階述語論理式で与えらえる検証式を対象とする方針を示したが，時相論理で対象にするオペレータは用途が広く，フルスペックの検証器を独自に実装するには実現するためのコストがかかる．さらに効率よくモデル検査を実装する方法し、高い表現力を誇る方式を検討している．

5. 謝辞

本稿執筆にあたって，アドバイスをいただいたソフトバンク株式会社の瀧本和弘氏，関係部署に調整いただいた株式会社テクノネットの原嶋麻衣子氏，および本稿を査読いただいた先生方にはこの場を借りてお礼申し上げます．

参考文献

[1] 中島 震, “ソフトウェア工学のツールとしての形式手法”, *NII Technical Report*, 2007.

[2] 情報処理推進機構, “システム再構築を成功に導くユーザガイド”, *SEC Books*, 2016

[3] 池田 信之, 今村 紀子, 高田 沙都子, “実用化に向けたモデル検査適用手法の開発”, 東芝レビュー, 2007

[4] IPA/SEC, “システムライフサイクルプロセス説明(案)”, システムズエンジニアリング推進 *WG* 参考 *2(17-SEWG-2)*

[5] 岸 知二, “モデル検査技術にょる UML 設計検証”, 情報処理 *2006* 年 *5* 月

[6] 向殿 政男, “信頼性と安全性”, *SEC journal Vol.10 No.3*

[7] Jean-Michel Couvreur, “On-the-fly Verification of Linear Temporal Logic”, *FM’99, Vol. I, LNCS 1708, pp. 253–271*, 1999.

[8] Stefania Gnesi, Franco Mazzanti, “On the fly model checking of communicating UML State Machines”, *http://fmt.isti.cnr.it/WEBPAPER/onthefly-SERA04.pdf*

[9] Ilan Beer, Shoham Ben-David, Avner Landver, “On-The-Fly Model Checking of RCTL Formulas”, *Lecture Notes in Computer Science*, Decembere 1999.

[10] R.Gerth, D.Peled, M.Y.Vardi, P.Wolper, “Simple On-the-fly Automatic Verification of Linear Temporal Logic”, *Proceedings of the 6th Symposium on Logic in Computer Science*

開発者に着目した Fault-Prediction 技術

高井 康勢
(株) 日立製作所
yasunari.takai.kb@hitachi.com

加藤 正恭
(株) 日立製作所
tadahisa.kato.en@hitachi.com

三部 良太
(株) 日立製作所
ryota.mibe.mu@hitachi.com

林 晋平
東京工業大学 情報理工学院
hayashi@c.titech.ac.jp

小林 隆志
東京工業大学 情報理工学院
tkobaya@c.titech.ac.jp

要旨

ソフトウェア開発では，開発の早い段階でフォールト（バグ）を見つけて修正することが重要となる．そのため，潜在的なフォールトの場所を予測する Fault-Prediction 技術に注目が集まっている．

Fault-Prediction 技術では，ファイルやモジュールごとに，潜在的なフォールトの存在確率を表す「潜在フォールト率」を算出する．算出した潜在フォールト率が高いファイルやモジュールを重点的にテストやレビューすることで，開発の早い段階でのフォールト発見が可能となる．

近年利用が進んでいる Fault-Prediction 技術では，静的解析ツールで検出できるコーディング規約違反や既知のフォールトの情報を基に，潜在フォールト率を算出する．しかしながら，開発の立ち上げ直後は，潜在フォールト率の算出に利用するコーディング規約違反やフォールトの情報が少ないため，従来技術では潜在フォールト率を算出できない．

そこで本研究では，「開発者」が過去に携わったプロジェクトで作り込んだ指摘やフォールトなどの情報を基に，ファイルごとの潜在フォールト率を算出する Fault-Prediction 技術を提案・評価した．本技術では，開発の立ち上げ直後でも潜在フォールト率を算出できる．

企業内のある 1 プロジェクトで提案手法を評価したところ，本手法で算出した潜在フォールト率が高い上位 25％のファイルに，全フォールトの約 6 割が含まれることが分かった．提案手法により，潜在フォールト率が高いファイルから順にテストやレビューを実施することで，開発の早い段階でのフォールトの発見・修正が可能となる．

1. はじめに

近年，大規模で複雑なソフトウェアを，高品質に素早く開発することが求められている．短い期間の中で品質の高いソフトウェアを作るためには，早期に効率よくフォールトを見つけ，修正による手戻りを最小限にすることが重要となる．このようなニーズに対して，潜在的なフォールトの場所を予測する，Fault-Prediction 技術に注目が集まっており，実プロジェクトでの利用も進んでいる．

Fault-Prediction 技術は，ソフトウェアの中から，潜在的なフォールトが存在する可能性が高い部分（モジュール・ファイル・クラス・メソッド・行など）を予測する技術 [1] である．具体的には，モジュールやファイルなどのソフトウェアの部分ごとに，潜在的なフォールトの存在確率を表す「潜在フォールト率」を算出する．Fault-Prediction 技術で算出した潜在フォールト率が高い部分を重点的にレビューやテストすることで，効率よくフォールトを見つけられる．

近年特に利用が進んでいる Fault-Prediction 技術には，既知のフォールトや静的解析ツール [2][3] で検出できるコーディング規約違反 [4][5] などの指摘を利用したものがある．これらの技術では，予測対象のプロジェクトにおける既知のフォールトや指摘を基に潜在フォールト率

を算出する．そのため，開発立ち上げ直後のプロジェクトなどで，既知のフォールトや指摘が少ない場合，潜在フォールト率を算出できないという課題がある．

この課題を解決するため，本研究では，ソフトウェアの品質に影響を与える「開発者」に着目し，開発者が過去に携わった既存のプロジェクトで作り込んだフォールトや指摘などの情報を基にした Fault-Prediction 技術を提案する．提案手法は，既存のプロジェクトにおけるフォールトや指摘を利用するため，立ち上げ直後のプロジェクトであっても利用可能である．

以降，2. では，関連する研究について説明する．3. では，提案手法について説明する．4. では，評価方法，評価対象，及び，結果を説明し，5. で考察を述べる．最後に，6. でまとめと今後の課題について述べる．

2. 関連研究

2.1. Fault-Prediction 技術

1971 年には，ソースコードの行数を指標とする Fault-Prediction 技術の研究 [6] が始まっており，以降，循環的複雑度 [7] などの指標を複数組み合わせた研究 [8] がおこなわれてきた．また，1990 年ごろからは，オブジェクト指向言語を対象とした研究 [9] なども行われている．

近年では，フォールトの分布や修正頻度を利用した研究 [10][11][12] で成果が出ており，実プロジェクトでの活用が進んでいる．代表的な方法は，「既知のフォールトがより多く存在するファイルに，より多くの潜在的なフォールトがある」と予測するものである．この方法では，ファイルごとに，「当該ファイルが原因になったフォールトの数」を集計して潜在フォールト率とする．

さらに，フォールトと静的解析ツールの指摘の関係があるという研究成果 [13][14] に基づき，指摘の分布や修正頻度を利用した研究 [15][16][17] もある．代表的な方法は，「指摘をより多く受けたファイルに，より多くのフォールトがある」と予測するものである．この方法では，まず，開発中のファイルに checkstyle[2] や FindBugs™[*1][3] などの静的解析ツールを適用して指摘を取得する．次に，ファイルごとの指摘数を集計して潜在フォールト率とする．

[*1] FindBugs™ とそのロゴは，the University of Maryland の登録商標です．

2.2. 開発者の特徴

ソースコードなどの成果物の品質には，開発者の影響があることが知られている．例えば，Avgustinov ら [18] は，開発者ごとに受ける指摘に違いがあることを示した．Matsumoto ら [19] は，開発者の特徴を表すメトリクスを提案し，開発者ごとに違いがあることを示した．

3. 提案手法

提案手法である「開発者に着目した Fault-Prediction 技術」は，開発の立ち上げ直後であっても利用できる Fault-Prediction 技術である．本手法は，ソフトウェアの開発者に着目し，開発者が過去に携わった既存のプロジェクトで作り込んだフォールトや指摘などの実績情報を基に，予測対象プロジェクトのファイルごとにフォールトが含まれる可能性を潜在フォールト率として算出することを特徴とする．

以降，3.1 にて実績情報を定義したのち，3.2 で開発者に着目した Fault-Prediction 技術について説明する．

3.1. 実績情報

本研究では，実績情報を「開発者のフォールトの作り込みやすさを直接的／間接的に表すメトリクス」と定義する．例えば，「開発行あたりの平均フォールト数」や「開発行あたりの平均指摘数」は，実績情報である．他にも，Matsumoto ら [19] が提案した「コミットあたりの平均フォールト混入率」など，開発者ごとに測定可能で，かつ，Fault-Prediction 技術で利用可能なメトリクスも実績情報である．

ここで，本論文で実績情報として利用した開発行あたりの平均フォールト数と開発行あたりの平均指摘数の定義を示す．

定義. 開発者 d の開発行あたりの平均フォールト数:Pf*(d)*

$$\begin{aligned}\mathrm{Pf}(d) &= \text{開発者 } d \text{ の開発行あたりの平均フォールト数} \\ &= \frac{\mathrm{FL}(d)}{\mathrm{DL}(d)}\end{aligned}$$

$$\mathrm{FL}(d) = \text{開発者 } d \text{ が開発した行に含まれるフォールト数} = \sum_{\text{開発者 } d \text{ が開発した行 } l} \mathrm{F}(l)$$

$$\mathrm{F}(l) = \text{行 } l \text{ が原因となったフォールトの数}$$

$$\mathrm{DL}(d) = \text{開発者 } d \text{ が開発した行数}$$

定義. 開発者 d の開発行あたりの平均指摘数:Pv*(d)*

$$\mathrm{Pv}(d) = \text{開発者 } d \text{ の開発行あたりの平均指摘数} = \frac{\mathrm{V}(d)}{\mathrm{DL}(d)}$$

$$\mathrm{V}(d) = \text{開発者 } d \text{ が開発した行に含まれる指摘数}$$

$$\mathrm{DL}(d) = \text{開発者 } d \text{ が開発した行数}$$

3.2. 開発者に着目した Fault-Prediction 技術

提案手法である開発者に着目した Fault-Prediction 技術では，以下に示す定義に基づいて潜在フォールト率を算出する．算出した潜在フォールト率が高いファイルを順にレビューやテストすることで，開発の早い段階でフォールトを見つけて修正することが可能となる．

定義. 実績情報に基づく潜在フォールト率:FR*(f)*

$$\mathrm{FR}(f) = \sum_{\text{ファイル } f \text{ の行 } l} \mathrm{P}(\mathrm{Dev}(l))$$

$$\mathrm{Dev}(l) = \text{行 } l \text{ の開発者}$$

$$\mathrm{P}(d) = \text{既存のプロジェクトにおける開発者 } d \text{ の実績情報}$$

尚，$\mathrm{P}(d)$ は，3.1 で定義した $\mathrm{Pf}(d)$，$\mathrm{Pv}(d)$ 等を表す．

4. 評価

開発の立ち上げ直後において，提案手法で算出した潜在フォールト率が高いファイルを重点的にレビューやテストすることで，フォールトを効率的に見つけられるか評価した．

4.1. 評価方法

具体的な評価方法は以下の通りである．

1. 開発がある程度進んでおり，尚且つ，同じ開発者が従事しているプロジェクトを 2 つ用意し，一方を開発者が過去に関わった既存のプロジェクト，もう一方を開発の立ち上げ直後である予測対象のプロジェクトとみなす．
2. 既存のプロジェクトを基に開発者の実績情報を取得し，その実績情報を基にして予測対象のプロジェクトの開発の立ち上げ直後における各ファイルの潜在フォールト率 FR(*f*) を算出する．
3. FR(*f*) が高い順にファイルをレビューやテストした時に発見可能なフォールト数を算出する．
4. ランダムにファイルをレビューやテストした時に発見可能なフォールト数を算出し，3. と比較する．

尚，評価は，3.2 で示した開発者の実績情報に基づく潜在フォールト率算出において，$\mathrm{P}(d)$ として $\mathrm{Pf}(d)$ を用いた方法と，$\mathrm{Pv}(d)$ を用いた方法の 2 通りで行った．以降，前者を FPf, 後者を FPv と呼ぶ．

表 1. 評価対象ソフトウェアの概要

開発者数	9 名
コミット数	17,000
規模（行数）	約 70KLOC
規模（ファイル数）	214
開発期間	約 1 年
フォールト数	224
指摘数	38,187

4.2. 評価対象

企業内の Java ソフトウェアのプロジェクトを対象に，提案手法を評価した．評価対象ソフトウェアの概要は，表 1 の通りである．尚，フォールトには，単体テストで見つけたフォールトと，結合テストで見つけたフォールトが含まれている．

本評価では，1 つのプログラムを機能の観点で 2 つに分割し，2 つのプロジェクトに見立てて評価した．両モジュールの概要は，表 2，表 3 の通りである．以降，両モジュールを，それぞれ「モジュール A」「モジュール B」と呼ぶ．

FPf, FPv 双方に対して，モジュール A を既存のプロジェクト，モジュール B を予測対象のプロジェクトとみなした評価と，その逆の評価を実施した（表 4）．以降，各

表 2. モジュール A の概要

規模（行数）	約 61KLOC
規模（ファイル数）	164
フォールト数	202
指摘数	33,226

表 3. モジュール B の概要

規模（行数）	約 9KLOC
規模（ファイル数）	50
フォールト数	22
指摘数	4,961

評価を，表中の「#」に記載した名称で呼ぶ．

4.3. 評価結果

B→A@FPf 及び B→A@FPv の結果を図 1 に，A→B@FPf 及び A→B@FPv の結果を図 2 に示した．両グラフの横軸は，提案手法で算出した潜在フォールトが高い順にファイルを並べたものであり，縦軸は，累積フォールト率を表している．累積フォールト率は，全フォールト数に対する，潜在フォールト率が高い順にテストやレビューを行った際に発見可能なフォールトの割合である．つまり，「潜在フォールト率が高いファイル上位 x 個の中に，全フォールトの y% のフォールトが含まれているか」を表している．例えば，図 1 では，潜在フォールト率が高いファイル上位 40 個をテストやレビューすることで，最大で，全フォールトの約 6 割のフォールトを発見できることを表している．

また，図 1 と図 2 における，潜在フォールト率が高いファイル上位 25%，50%，75% までの累積フォールト率を表 5 に示した．尚，理論上の最大値は，既知のフォールトが多い順にファイルを並べたものであり，また，ランダムサンプリングは，10 回ファイルをランダムに並べた際の，累積フォールト率の平均である．

図 1，図 2，及び，表 5 をもとに，FPf 及び FPv の結果について述べる．

FPf では，潜在フォールト率が上位 25% のファイルに，B→A@FPf で 64%，A→B@FPf で 55% のフォールトが含まれていた．どちらも，25% のファイルをランダムに選んだ場合と比較すると，倍以上のフォールトを含んでいる．また，潜在フォールト率が上位 50%，75% の場合においても，ファイルをランダムに選んだ場合より，多くのフォールトが含まれていた．

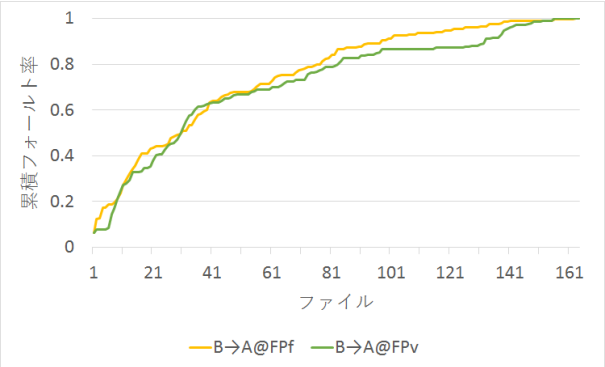


図 1. B→A@FPf 及び B→A@FPv における潜在フォールトの予測結果

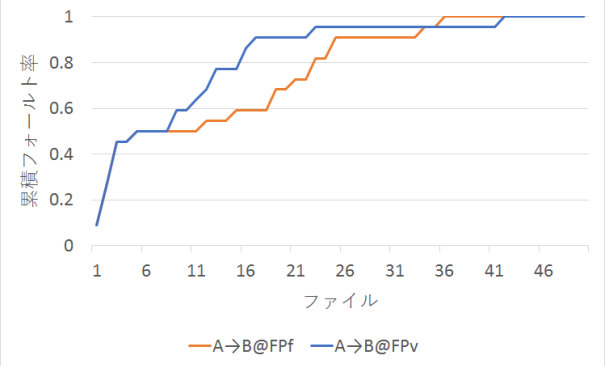


図 2. A→B@FPf 及び A→B@FPv における潜在フォールトの予測結果

FPv では，潜在フォールト率が上位 25% のファイルに，B→A@FPv で 63%，A→B@FPv で 68% のフォールトが含まれていた．どちらも，25% のファイルをランダムに選んだ場合と比較すると，倍以上のフォールトを含んでいる．また，潜在フォールト率が上位 50%，75% の場合においても，ファイルをランダムに選んだ場合より，多くのフォールトが含まれていた．

FPf と FPv を比較すると，潜在フォールト率が上位

表 4. 評価内容

#	対象モジュール		実績情報
	既存のプロジェクト	予測対象のプロジェクト	
B→A@FPf	モジュール B	モジュール A	FPf
B→A@FPv	モジュール B	モジュール A	FPv
A→B@FPf	モジュール A	モジュール B	FPf
A→B@FPv	モジュール A	モジュール B	FPv

表 5. 評価結果

#	上位 25%	上位 50%	上位 75%
A@ファイル数	41	82	123
A@理論上の最大値	81%	100%	100%
B→A@FPf	64%	84%	96%
B→A@FPv	63%	79%	87%
A@ランダムサンプリング	28%	53%	77%
B@ファイル数	13	25	38
B@理論上の最大値	100%	100%	100%
A→B@FPf	55%	91%	100%
A→B@FPv	68%	95%	95%
B@ランダムサンプリング	20%	46%	66%

75% の場合においては，FPf の方が多くのフォールトを含んでいるが，それ以外の場合では，一定の傾向はみられなかった．

5. 考察

5.1. 提案手法の有用性

4.3 で示した通り，立ち上げ直後のプロジェクトであっても，提案手法を利用し，潜在フォールト率が上位 25% のファイルを重点的にレビューやテストすることで，約 6 割のフォールトを発見できる可能性がある．

大規模で複雑なソフトウェアを高品質に素早く開発するためには，早期に効率的にフォールトを発見することが重要となる．しかしながら，従来手法は，立ち上げ直後のプロジェクトでの利用が難しい．一方，提案手法では，立ち上げ直後のプロジェクトであっても，効率的なフォールト発見を実現できると言える．

5.2. FPf と FPv の比較

4.3 で示した通り，本評価では，FPf と FPv の間に優位な差を見出すことはできなかった．但し，特にモジュール B は，ファイル数が 50 ファイルとそれ程多くないため，測定誤差による影響が大きくでている可能性がある．

5.3. 従来手法との比較

従来手法と提案手法を比較しながら，提案手法であれば，開発の立ち上げ直後であっても潜在フォールト率を算出できる理由を考察する．

図 3 は，時間経過に伴う，既知フォールト，指摘，実績情報の変化と，従来手法及び提案手法での潜在フォールト率の算出可否を表している．既存のプロジェクトも予測対象のプロジェクトも，立ち上げ直後には既知のフォールトが存在しておらず，また，指摘も少ない．開発が進むと，まず指摘が多くなり，さらに開発が進むと，既知のフォールトも多くなる．尚，図中の網掛け部分は，従来手法，提案手法で潜在フォールト率を算出できる部分を表している．

従来手法，提案手法ともに，予測に利用する既知のフォールトや指摘の情報（「予測基情報」と呼ぶ）が少ないと，潜在フォールト率の算出ができない．一方で，従来手法と提案手法には，既存のプロジェクトの予測基情報を利用できるか否かの点で差がある．

従来手法は，予測対象のファイルにおけるフォールトや指摘の情報を予測基情報として利用する．既存のプロジェクトで作成したファイルと予測対象のプロジェクトで作成したファイルは異なるため，既存のプロジェクトの予測基情報を予測対象のプロジェクトで利用することができない．それに対し，提案手法では，予測対象の開発者における実績情報を予測基情報として利用する．そのため，同じ開発者が既存のプロジェクトと予測対象のプロジェクトを担当していれば，既存のプロジェクトで得られる予測基情報を予測対象のプロジェクトで利用できる可能性がある．

本評価より，既存のプロジェクトで得られる予測基情報を予測対象のプロジェクトで利用できることが分かった．

6. まとめと今後の課題

近年，ソフトウェアに存在するフォールトの発見を支援できる Fault-Prediction 技術に注目が集まり，実プロジェクトでの利用が進んでいる．しかしながら，従来の Fault-Prediction 技術では，フォールトの予測の基になる情報が少ない立ち上げ直後のプロジェクトでは，利用が難しいという課題がある．そこで本研究では，開発者が過去に関わった既存のプロジェクトを分析し，開発者ごとの実績情報を得たうえで予測の基になる情報として利用し，開発の立ち上げ直後でも利用できる Fault-Prediction 技術を提案した．

提案手法を企業内のある 1 プロジェクトで評価したところ，実績情報として「開発行あたりの平均フォールト数」と「開発行あたりの平均指摘数」を利用した場合，フォールトが存在する確率が高いと予測した上位 25% のファイルに全フォールトの約 6 割が含まれるという結果を得られた．この結果は，ランダムに 25% のファイルを選択した際に含まれるフォールトの割合（20%〜30%）と比較すると，2 倍程度大きい．つまり，提案手法により算出したフォールトの存在確率が高いファイルを重点的にレビューやテストすることで，立ち上げ直後のプロジェクトであっても，早期のフォールト発見を支援できると言える．

尚，本研究では 1 プロジェクトのみを利用して評価したため，他のプロジェクトを含めた評価が今後の課題である．また，今回利用した 2 つの実績情報以外の実績情報でも評価し，予測精度向上の可能性を探る．

参考文献

[1] Rakesh Kumar and D. L. Gupta. Software fault prediction : A review. In *International Journal of Computer Science and Mobile Computing*, Vol. 4, pp. 250–254, Sep 2015.

[2] checkstyle checkstyle 8.18. `http://checkstyle.sourceforge.net/`.

[3] Findbugs - find bugs in java programs. `http://findbugs.sourceforge.net/`.

[4] MISRA-C 研究会. 組込み開発者におくる MISRA - C:2004—C 言語利用の高信頼化ガイド. 日本規格協会, 10 2006.

[5] Pep 8 – style guide for python code — python.org. `https://www.python.org/dev/peps/pep-0008/`.

[6] Fumio Akiyama. An example of software system debugging. In *Proceedings of IFIP Congress*, pp. 353–359, 1971.

[7] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, pp. 308–320, Dec 1976.

[8] V. Y. Shen, , S. M. Thebaut, and L. R. Paulsen.

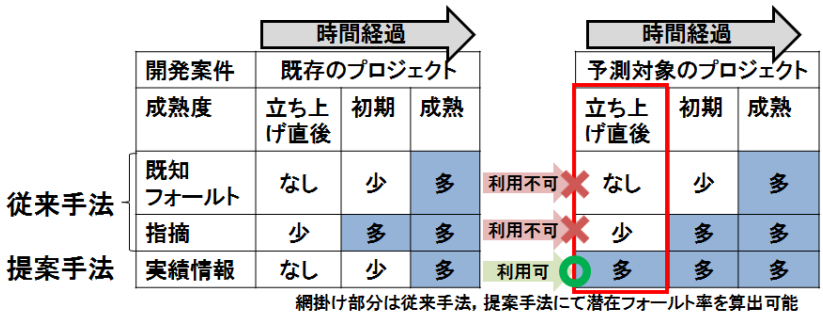


図 3. 既知フォールト，指摘，実績情報の変化と潜在フォールト率の算出可否

Identifying error-prone software - an empirical study. *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 4, pp. 317–324, April 1985.

[9] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *Proceedings of 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pp. 31–41, May 2010.

[10] Foyzur Rahman, Daryl Posnett, Abram Hindle, Earl T. Barr, and Premkumar T. Devanbu. Bugcache for inspections: hit or miss? In *Proceedings of FSE*, pp. 322–331. ACM, 2011.

[11] Chirs Lewis and Rong Ou. Bug prediction at google — google engineering tools. `http://google-engtools.blogspot.com/2011/12/bug-prediction-at-google.html`, 12 2011.

[12] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller. Predicting faults from cached history. In *Proceedings of 29th International Conference on Software Engineering (ICSE 2007)*, pp. 489–498, May 2007.

[13] Nathaniel Ayewah and William Pugh. A report on a survey and study of static analysis users. In *Proceedings of the 2008 Workshop on Defects in Large Software Systems*, (DEFECTS 2008), pp. 1–5, 2008.

[14] Foyzur Rahman, Sameer Khatri, Earl T. Barr, and Premkumar Devanbu. Comparing static bug finders and statistical prediction. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*, pp. 424–434, 2014.

[15] Wojciech Basalaj. Correlation between coding standards compliance and software quality. In *White paper, Programming Research Ltd*, 2008.

[16] C. Boogerd and L. Moonen. Evaluating the relation between coding standard violations and faultswithin and across software versions. In *Proceedings of 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR 2009)*, pp. 41–50, May 2009.

[17] Y. Takai, T. Kobayashi, and K. Agusa. Software metrics based on coding standards violations. In *Proceedings of 2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pp. 273–278, Nov 2011.

[18] P. Avgustinov, A. I. Baars, A. S. Henriksen, G. Lavender, G. Menzel, O. d. Moor, M. Schafer, and J. Tibble. Tracking static analysis violations over time to capture developer characteristics. In *Proceedings of 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE 2015)*, Vol. 1, pp. 437–447, May 2015.

[19] Shinsuke Matsumoto, Yasutaka Kamei, Akito Monden, Ken-ichi Matsumoto, and Masahide Nakamura. An analysis of developer metrics for fault prediction. In *Proceedings of the 6th In-*

ternational Conference on Predictive Models in Software Engineering (PROMISE 2010), pp. 18:1–18:9, 2010.

質問に対する回答者推薦手法に用いられるデータの期間についての一検討

眞鍋 雄貴
熊本大学
y-manabe@cs.kumamoto-u.ac.jp

西原 弘樹
熊本大学
c5846@st.cs.kumamoto-u.ac.jp

要旨

ユーザ間で質問回答を行うプラットフォームとして質問回答コミュニティ（*CQA*）がある．*CQA* では日々多くの質問に対して，回答が行われる．一方で，回答が行われない質問も多くある．この問題を解決するため，回答者推薦手法が提案されている．特に，複数の手法を組み合わせた回答者推薦手法が多く提案され，特に，トピック分析とネットワーク分析を組み合わせた手法が多くを占める．しかしながら，これらの手法の評価では，推薦に用いるデータの期間が考慮されていないこと，また，新たな質問をテストデータにしていないものがある．これらは，回答者推薦手法がユーザにとって有用であるかを評価するには必要な観点である．そこで，本研究では，このような観点から回答者推薦手法を評価する第一歩として，プログラマ向け *CQA* である *Stack Overflow* において，質問や回答を抽出する期間を基準となる日の直前 *3* ヶ月，*6* ヶ月，*9* ヶ月，*12* ヶ月として得られたデータセットを用い，トピック分析とネットワーク分析を組み合わせた回答者推薦手法の結果がどのように変わるのかを調査した．その結果，*3* ヶ月の場合にのみ正しく推薦を行えた質問があった．また，この質問に対して回答者として推薦されたユーザのうち *3* 名は基準日直前 *2* ヶ月間は活動を行っていなかったことが示された．

1. はじめに

ユーザ間で質問, 回答を行うプラットフォームを提供するウェブサイトとして質問回答コミュニティ（CQA）がある．CQA には特定のドメインに依存しないもの（Yahoo 知恵袋など）の一方で, 特定のドメインに特化した CQA が多く設立されている．特定のドメインに特化した CQA のうち，プログラマ向け CQA で最大の CQA として Stack Overflow がある．Stack Overflow では，プログラミングに関する多様なトピックについてユーザ間で質問や回答がなされている．ユーザのアカウント数は 2019 年 2 月 11 日時点で，9,663,406 人，投稿された質問の数は 17,167,300，1 日あたりの質問投稿件数が 73000（2019 年 3 月 14 日時点）となっている．

しかし，CQA には大きく二つの制約がある．一つは，CQA では多くの質問に回答がつかないままである．Stack Overflow では 2019 年 3 月 14 日時点では 29%の質問に回答がついていないことである[1] ．もう一つは，CQA はどのようなユーザが質問に回答するかわからないため，回答の品質も様々に変わることである．

これらの問題に対し，その分野の専門知識を持つユーザが質問に回答できるよう，このようなユーザに適切な質問が提示されるように，CQA における新たな質問に対する適切な回答者を推薦する手法が提案されている．近年では，複数の回答者推薦のアプローチを組み合わせた手法が提案されており，特に，トピック分析とネットワーク分析を組み合わせた手法が多く提案されている [1, 2, 3, 4, 5]．しかし，これらの手法の評価では推薦に用いるデータがどのような期間で取得されたかについての検討が不十分である．回答者推薦手法が適切に働き，また，質問を行うユーザや回答を行うユーザにとって有用なものとするには，以下の 2 点について評価を行うのが望ましい．

- どの程度の期間のデータが効率の良い推薦に必要か．
- 最新の新規の質問に対して，推薦を行うことができるか．

前者は，回答者推薦手法が有効に働くにはどの程度の

[1] https://stackexchange.com/sites?view=list#answers

データが必要となるかの前提を示している．後者は，回答者推薦手法の目的がどの程度達成されているかを示すものである．

本研究では，CQA を用いるユーザの観点から回答者推薦手法がどの程度有用であるかを評価する最初の一歩として，期間が異なる複数のデータセットに対して回答者推薦手法を用いることによって，どのように推薦結果が変化するかを調べた．本研究では，3 ヶ月，6 ヶ月，9 ヶ月，12 ヶ月（1 年）と期間の異なるデータセットを用意した．また，各データセットのうち，最新 1 ヶ月間に行われた質問からランダムに 100 件選び，このうちベストアンサーが設定されていた 47 件をテスト用のデータとした．回答者推薦の手法としては，トピック分析の手法として Twitter-LDA[6] を，ネットワーク分析の手法として HITS[7] を組み合わせた手法を用いた．結果として，3 ヶ月の期間のデータを利用した場合が最も良い精度を示した．

2. 質問回答コミュニティ

質問回答コミュニティ（以後 CQA）は対話的な質問への回答が行われることに基づくコミュニケーションプラットフォームを提供するサービスであり，ユーザが世界中の人々との知識の交換を可能としている [5]．また，数多くの質問や回答を蓄積しているため，ウェブサービスエンジンでは容易に得られない価値のある資源を提供している [8]．現在，多くの CQA が設立されており，多くの CQA をホストしている Stack Exchange では，174 のコミュニティがホストされている．各 CQA では，多くの場合各 CQA では扱う質問の分野が決まっている．例として，Stack Overflow はプログラマの質問に答え，Arqade ではビデオゲーマの質問，Seasoned Advice ではシェフの質問に答えるコミュニティだとしている．

Stack Overflow のスクリーンショットを図 1 に示す．Stack Overflow では質問にタグをつける機能，回答を第三者が編集する機能のほか，その質問もしくは回答に，高評価や低評価をつけたりする機能を提供している．上部には質問の情報が記録されており，上からタイトル，本文，タグ，質問者情報が記載されている．下部には回答の情報が記載されており，上から総回答数，本文，回答者情報，コメントが記載されており，ベストアンサーには左端にチェックマークがついている．

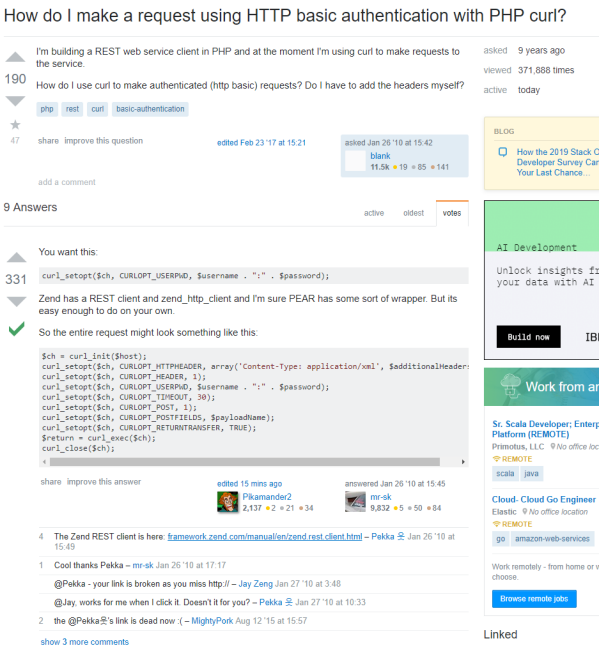


図 1. Stack Overflow のスクリーンショット

3. 本研究で使用するデータ

本研究では，Stack Exchange Explore[1]から取得したデータを用いた．Stack Exchange Explore は Stack Exchange がホストする CQA のデータのアーカイブを保存するサイトである．保存されるデータについては，各 CQA の投稿内容だけではなく，メタデータも保存される．

本研究では，Stack Overflow に焦点を当て，そのうち，User に関するデータ 10097976 件が保存されている StackOverflow-Users.xml と，投稿内容に関するデータが保存されている StackOverflow-Posts.xml を用いた．StackOverflow-Users.xml は各ユーザのデータが xml 形式で格納されている．各ユーザのデータが持つ属性を図 1 に示す．

一方，StackOverflow-Posts.xml は質問と回答についてのデータ 43369425 件が xml 形式で格納されている．質問と回答についてのデータが持つ属性を図 2 に示す．属性には，質問と回答共通の属性，質問のみにある属性，回答のみにある属性がある．特に，回答と質問の対応関係については，回答の ParentId を調べる必要がある．一方，タグは質問のみに Tags として記録されるため，回答がどのタグに関するものかを知るには対応する質問の

[1] https://archive.org/download/stackexchange

属性名	説明
Id	各ユーザの ID
Reputation	ユーザの信用度
CreationDate	アカウントが作成された日時
DisplayName	ユーザ名として表示される文字列
LastAccessDate	アカウントに最後にアクセスした日時
WebsiteUrl	ユーザが登録したウェブサイトの URL
Location	ユーザが登録した場所
AboutMe	ユーザが登録した紹介文
Views	プロファイルの閲覧回数
UpVotes	ユーザが得た高評価の回数
DownVotes	ユーザが得た低評価の回数
ProfileImageUrl	ユーザが登録したユーザの画像
AccountID	ユーザの Stack Exchange Network でのアカウントの ID

表 1. StackOverflow-Users.xml における属性

データを見る必要がある．また，OwnerUserId に着目すると，あるユーザが何度質問もしくは回答を行ったかを調べることができる．

これらのファイルは巨大であり，またユーザや質問回答の数も多いため，データを選択した．その上で，それぞれのファイルを実験で用いることができるように，前処理を行い，データベースへの格納を行った．以下，各ステップについて述べる．

3.1. データの選択

本研究では，[5] での実験設定を踏まえ，'scala' をタグに持つ質問とその回答を用いた．また，使用するデータの期間は，2018 年 12 月 2 日から 1 ヶ月前までの範囲にある質問をテストデータ，2018 年 12 月 2 日から 3 ヶ月前，6 ヶ月前，9 ヶ月前，1 年前までのデータのうち，テストデータの期間を除いた部分を学習データとして用いる．また，テストデータとなる質問は，最新 1 ヶ月の質問のうち 100 の質問を無作為に選んだ．

3.2. データベースへの格納

データの加工の全体像を示す．巨大な XML ファイルを分割し，それを CSV ファイルに変換する．次に CSV ファイルに基づいてデータベースを作成する．最後にそのデータベースから必要な情報を読み出す．

初めに，XML ファイルを 100000 行ごとに分割し，それぞれ CSV ファイルに変換する．次に，各 CSV ファイルをデータベースに格納する．まず，今回使用したタグである'scala' をタグに持つ質問をデータベースに取り込む．その後，その質問に対する回答を取得し，データベースに追加する．3.1 節に示したとおり，本研究では'scala' タグがついている質問を用いるが，表 2 で示されている通り，回答には'scala' タグは付いていない．そのため，回答の ParentId から対応する質問を得て，その上で質問の Tags に'scala' があるかを見ている．

属性名		説明
Id	C	各投稿の id
PostTypeId	C	投稿の種別.
	C	主なものは 1:質問，2:回答.
CreationDate	C	投稿が行われた日時
Score	C	投稿のスコア
Body	C	投稿の本文
OwnerUserId	C	投稿を行ったユーザの ID
OwnerDisplayName	C	投稿を行ったユーザの名前
LastEditorUserId	C	最後に編集したユーザの ID
LastEditorDisplayName	C	最後に編集したユーザの名前
LastEditDate	C	最後に投稿が編集された日時
LastActivityDate	C	最後に投稿に対して変更，回答などが行われた日時
CommentCount	C	投稿に対して行われたコメントの数
CommunityOwnedDate	C	コミュニティ Wiki に投稿された日時
AcceptedAnswerId	Q	質問を投稿したユーザがベストアンサーとしてマークした回答の ID
ViewCount	Q	質問の閲覧回数
Title	Q	質問のタイトル
Tags	Q	質問につけられたタグ
AnswerCount	Q	質問に対してつけられた回答の数
FavoriteCount	Q	質問に対してユーザがお気に入りにした数
ClosedDate	Q	（クローズされた質問について）クローズされた日時
ParentId	A	回答が対応する質問の ID

表 2. StackOverflow-Posts.xml における属性．C: 質問，回答共通属性，Q:質問のみの属性，A:回答のみの属性を意味する

3.3. 実験用データの生成

次に，作成したデータベースからユーザごとの回答履歴を抽出し，ユーザごとにテキストファイルに書き出す．たとえば，ユーザ ID が 100 のユーザの回答のリストは，100.txt となる．同時に，作成したテキストファイルのリストをテキスト形式で出力している．テストデータとして使用する質問についても，テキストファイルをそれらのファイルパスを記載したファイルリストを作成する．最後に，それらのテキストファイルから，<p>のような HTML タグや，ソースコードの部分を除外する．各データセットにおける回答者数，ユーザ数，回答数，質問数を表 3 に示す．

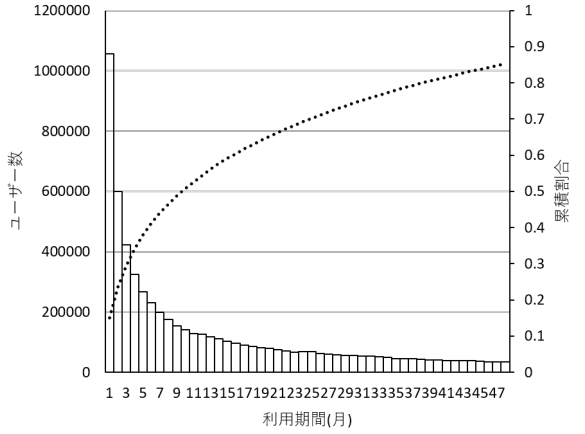


図 2. 利用期間ごとのユーザ数

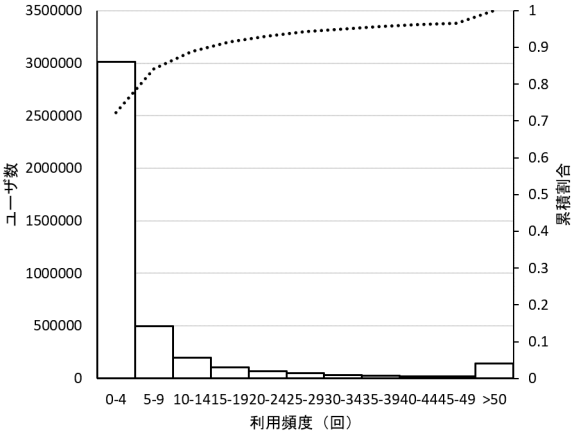


図 3. 利用頻度ごとのユーザ数

4. 予備実験

本研究での問題意識を確認するため，予備実験を行った．予備実験として，Stack Overflow のユーザの利用期間に対する分布を調べた．利用期間は，最後に質問，もしくは回答を行なった日時から，最初にアカウントを作成した日時の差を月単位で表したものとした．利用期間が 48 ヶ月までのユーザの数を図 2 に示す．図 2 は利用期間ごとのユーザ数を表しており，横軸が期間，縦軸がユーザ数である．

図 1 の結果から，ユーザのアカウント利用期間は，3 ヶ月以内が 29.5% ，6 ヶ月以内が 41.2% ，9 ヶ月以内が 48.8% ，12 ヶ月以内が 54.4%となっている．このことから，長期間にわたるデータを用いて推薦を行なったとしてもすでに利用していないユーザを推薦する可能性がある．一方で， 1 ヶ月以内のユーザも 15.0%いるため，推薦に用いるデータに存在せず，推薦できないユーザも要る可能性があることを示している．

表 3. 各データセットにおける回答者数，ユーザ数，回答数，質問数

	3months	6months	9months	1year
answerers	784	1659	2545	3257
users	1820	3952	6119	7880
answers	2204	5545	9794	13361
questions	966	966	966	966

これに加え，利用頻度についても調査した．ユーザの利用頻度に対する分布を図 3 に示す．利用頻度は，各ユーザがアカウントを作成してからの質問の回数と回答の回数の合計とした．図 3 は利用頻度ごとのユーザ数を表しており，横軸が質問や回答を行った回数，縦軸がユーザ数である．図は，50 回未満のユーザが 96.6%を占めていること，また，72.2%のユーザは 4 回以下の活動のみ行っていることを示している．また，50 回以上のユーザが行った投稿について調査したところ，質問，回答全体のうち，55.6%を占めていた．これは一部のユーザが大部分の質問や回答を行っていることを意味する．このことが，推薦結果に影響を及ぼす可能性がある．

5. 関連研究

回答者推薦手法に関するサーベイとして，Wang ら [9] のサーベイがある．本サーベイでは，回答者推薦手法を Simple Method，Language Models，Topic Models，Network-Based Methods，Classification Methods，Expertise Probablistic Models，Collaborative Filtering Methods，Hybrid Methods と分類している．Simple Method は，ユーザに単純な尺度でスコアをつけることによって推薦を行う手法である．尺度の例としては，投票，ベストアンサーの割合，テキストの類似度がある．Language Models は言語モデルによる推薦を行う手法であり，主に query likelihood language model (QLL)[10] とその改良モデルが用いられる．Topic Models は，トピック分析によって推薦を行う手法であり，Probablic

Latent Semantic Analysis(PLSA)[11] や Latent Dirichlet Allocation(LDA) モデル [12] やその改良が用いられる．Network-Based Methods は，ネットワーク分析による推薦を行う手法であり，PageRank や HITS とその改良が用いられている．Classification Methods は，分類により，エキスパートのクラスと，そうでないクラスにユーザを分割する手法である．Expertise Probablistic Models は，Dom ら [13] によって提案されているベイジアン確率モデルを使用してユーザの信頼性の事後推定を取得し，特定の質問の専門家を推薦する手法である．Collaborative Filtering Methods は強調フィルタリングを回答者推薦に利用しようする手法である．HybridMethods は今までに挙げた手法を複数組み合わせて推薦を行うものである．

HybridMethods に属する回答者推薦手法は近年多く提案されており，特に，Topic Models と Network-Based Methods を組み合わせた手法が多い [1, 2, 3, 4, 5]．Zhou ら [1] はトピック分析手法として LDA，ネットワーク分析手法として PageRank を用いた Topical PageRank を提案している．Zhao ら [2] は Topic Model と Network Analysis を組みあわせており，トピック情報を得るために TEL を用いる．TEL は LDA に基づいたモデルで，グラフに基づくリンク分析と内容に基づく意味解析を組み合わせている．Yang ら [3] はトピック分析手法として LDA を，ネットワーク分析手法として HITS を改良した NEWHITS を用いている．Zhou ら [4] はトピック分析手法として LDA を用いるが，質問に対してだけではなく，ユーザに対するトピックを抽出する際にも用いる．また，ネットワーク分析手法としては PageRank を用いる．Li ら [5] の手法がある．Li らは Twitter など短文集合に対応するトピック分析手法である Twitter-LDA[6] を元にした Question-LDA と，ネットワーク分析手法である HITS[7] を改良した NEWHITS を組み合わせた回答者推薦手法を提案している．

本研究では，最も多いトピック分析とネットワーク分析手法の組み合わせが重要な組み合わせであると考え，そのうち，実装が公開され[2]，短い文章でも有効であるとされる Twitter-LDA[6] をトピック分析手法として用い，ネットワーク分析においては，シンプルな HITS を初期段階の検討対象として選択した．

その他の組み合わせとして，ネットワーク分析とクラスタリングを組み合わせた手法 [14]，言語モデル，トピック分析，ネットワーク分析を組み合わせた手法 [15] がある．Bougessa ら [14] はエキスパート推薦のためにネットワーク分析とクラスタリングを組み合わせている．本手法では，ユーザ間のネットワークを質問者とその質問にベストアンサーを提示したユーザ間の関係をグラフとし，グラフの重みは始点ノードに対応するユーザが終点ノードに対するユーザの質問へベストアンサーを示した回数である．Liu ら [15] は言語モデル，トピック分析，ネットワーク分析を組み合わせた CQA ランクを提案している．本モデルは言語モデルである QLL と LDA モデルを関連性を分析するために使用し，それに加えユーザの活動やオーソリティの情報が推薦に考慮される．本研究では，これらの組み合わせについては検討の対象外である．

[2] https://github.com/minghui/Twitter-LDA

6. 実験

6.1. 本実験で用いる回答者推薦手法

初めに，ユーザごとの回答履歴とテストデータとなる質問を Twitter-LDA で処理し，回答者を推薦したい質問のトピック行列（行列 WT）と，ユーザの興味を表すトピック行列 (行列 UT) を取得する．

次に，データベースを利用してユーザ間のリンクを取得し，HITS を実行する．これによって，ユーザのオーソリティスコアを計算する．

最後に，あるユーザにおいて行列 WT と行列 UT を掛け合わせる．その結果に，そのユーザのオーソリティスコアをかける．これによって，あるユーザがある質問に対してどれだけ回答する能力を持っているか計算できる．この結果を行列 QR とし，式 (1) に示す．

$$QR = WT \times UT^T \times AuthorityScore \qquad (1)$$

これを全てのユーザ，全てのテストデータの質問において繰り返すことで，テストデータの質問ごとの推薦者 5 人を取得する．

6.1.1 トピック行列の算出

トピック行列の算出では，行列 UT と行列 WT を得る．本ステップの入力として，すでに推薦対象の質問の集合と，各ユーザの回答履歴を用い，これらの文書に対してどのトピックがどの程度関連があるかを示す行列を

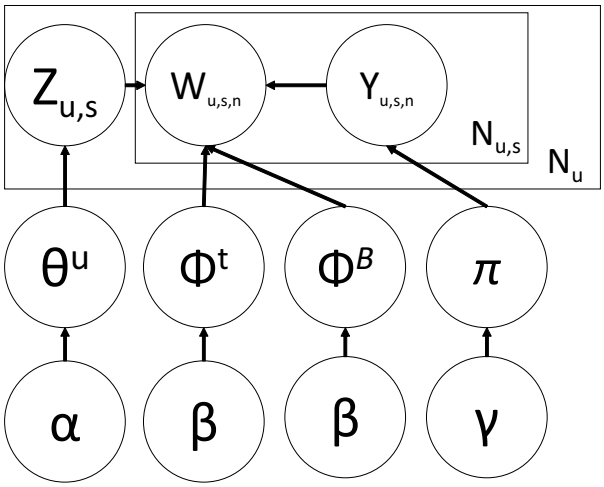


図 4. Twitter-LDA の生成モデル

得る．そして，この行列のうち，推薦対象の質問に対するトピックの対応を示す部分が行列 WT，ユーザの回答履歴に対するトピックの対応を示す部分が行列 UT となる．

推薦対象の質問の数を Q，推薦対象のユーザの集合を U，トピックの集合を T とすると，行列 WT は $|Q| \times |T|$ 行列，行列 UT は $|U| \times |T|$ 行列となる．

Twitter-LDA は Blei ら [12] によって考案された LDA を Twitter のような短文から構成される文書集合に対して適用するために改良したトピック分析手法である [6]．従来の LDA が持つ短文の解析の精度という弱点を，トピックの選定には利用しない単語を作成することで処理している．Twitter-LDA の生成モデルを図 4 に示す．

入力にはテキストの文書データが用いられ，出力は各単語の出現確率である．パラメータは，トピックの選択確率を得るためのパラメータである α，トピックに応じて単語の生成確率を得るためのパラメータである β がある．加えて，Twitter-LDA ではパラメータ γ が導入される．γ は通常の単語とトピックの選定に利用しない単語のベルヌーイ分布である π を導くために使用される．

6.1.2 Authority Score の算出

HITS とは，ネットワーク構造を測るための手法である [7]．本来はインターネットのページの構造を分析するための手法で，Hub Score と Authority Score という二つの値を結果に持つ．優秀な Hub は優秀なページに対して多くリンクを持ち，優秀な Authority は多くのページからリンクを持つ．この考えに基づき，Authority Score は，そのページに対してリンクをしているページのハブスコアの合計値によって求められ，Hub Score は，そのページがリンクを行っているページの Authority Score の合計値によって求められる．グラフの頂点をユーザ，質問したユーザからその質問に回答を行ったユーザへ辺を取ったグラフ (V, E) における Hub Score と，Authority Score の計算式を式 (2) と (3) に示す．

$$HubScore(p) = \Sigma_{q:(q,p)\in E} AuthorityScore(q) \quad (2)$$

$$AuthorityScore(p) = \Sigma_{q:(p,q)\in E} HubScore(q) \quad (3)$$

全ての Hub Score，もしくは Authority Score の二乗和で割ることで，この式を正規化し，値が収束するまで繰り返す．

6.1.3 推薦結果の算出

これまでのステップで得られた，行列 WT，UT，AuthorityScore の積を式 4 で取ることにより，ユーザと文書間の対応関係を示す行列 QR が得られる．この QR において，推薦対象となる質問に対応する部分を見ることで，その質問にどの程度各ユーザが適しているかが値で示されることになる．これを，数値が大きい順にユーザをランク付けし，その上位が推薦結果となる．この式から，QR の値は，回答者の推薦を行いたい質問に関連するトピックの質問に多く答え，また，多くのユーザの質問の答えるユーザに対して大きくなりやすいことが示されている．

$$QR_{mn} = A_m \sum_{t\in T} UT_{mt} WT_{nt} \quad (4)$$

6.2. 実験方法

実験では，3 ヶ月，6 ヶ月，9 ヶ月，12 ヶ月（1 年）分抽出したデータセットに対し，Twitter-LDA と HITS を用いた回答者推薦を行い，精度を評価する．精度の尺度として，precision，recall，F-score を用いる．各尺度の定義を，式 5，6，7 に示す．ただし，正解の予測をして実際に正解だった結果の数を TP，不正解だった結果の数を FP とする．不正解の予測をして，実際に不正解だった結果の数を TN，正解だった結果の数を FN とす

る.[5] 本実験では，ベストアンサーとなった回答をした回答者を正解の回答者とし，節に示した手法を用いて推薦された結果と比較する．

$$\text{precision} = \frac{TP}{TP + FP} \tag{5}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{6}$$

$$\text{F-score} = \frac{2recall \cdot precision}{recall + precision} \tag{7}$$

6.2.1 実験環境

実験のために，Twitter-LDA を入手した．これは，GitHub に共有されていたコードである．Twitter-LDA は，以前述べた通り，[6] によって研究された LDA の派生型で，短文分析もこなすことができる．こちらは，Windows10 および，java，そして java 用の IDE である eclipse を使用した.[3] また，先ほど示した実験用のデータセット加工の部分と，次の実験手順で示すネットワーク分析と各ユーザの質問に対するオーソリティスコアの計算の部分については，python3.6 および Ubuntu on Windows[4]を使用した．これは，Windows 上で Ubuntu のコマンドを利用できるアプリケーションである．

6.2.2 パラメータの設定

LDA のパラメータは [1] に従い，α を 0.1，β を 0.01，トピックの数を 50，iteration を 30 とした．また，γ については，初期値の 0.20 を用いた．

6.3. 結果

以下に実験の結果を示す．テストデータの 100 の質問のうち 53 の質問は，ベストアンサーが指定されていない，回答がついていない，削除されていた，などの理由で無効であった．そのため，有効な推薦は 47 となった．このうち，5 つの推薦結果の中に正解を含んでいたものは，3 ヶ月の期間において 5 つ，それ以外の期間においては 2 つだった．表 4 に，使用データの期間それぞれにおける，precision，recall，F-score を示す．この結果から，本研究で想定したネットワーク分析とトピック分析を組み合わせた手法において，回答者の推薦はできており，また，2 ヶ月分のデータが学習データとして得られていれば一定の精度が得られると言える．

表 4. 期間ごとの precision，recall および F-Score

	3months	6months	9months	1year
precision	0.021	0.009	0.009	0.009
recall	0.106	0.043	0.043	0.043
F-Score	0.035	0.015	0.015	0.015

6.4. 考察

表 5. 質問 53224244 における推薦結果

3months	6months	9months	1year
4993128	2707792	2707792	2707792
6867048	7662670	5880706	4993128
5516172	4204198	6316508	5880706
7662670	4993128	9953316	6316508
2750966	5880706	4204198	7662670

表 6. 質問 53224244 におけるユーザ 5516172 の推薦順位

3months	6months	9months	1year
3	14	24	26

表 5 に 3 ヶ月のデータセットを使ったときのみ正解を推薦した質問 53224244 における推薦されたユーザのリストを示す．それぞれのリストにおいて，重複している id が多く存在する．一方で，正解となる 5516172 は 3 ヶ月のデータセットにのみ存在している．これを深く調べるため，表 7 に各ユーザが期間ごとに何回質問・回答を行ったかを示す．各列は，1 行目に記載されている日から一ヶ月前までの間に各ユーザが行った質問・回答回数を示している．これを見ると，ユーザ 27707792，2750966，4993128，6316508，7662670 はこの 1 年間どの期間においても活動が行われている．また，ユーザ

[5] http://ibisforest.org/index.php?F 値
[3] https://github.com/minghui/Twitter-LDA
[4] https://www.microsoft.com/ja-jp/p/ubuntu/9nblggh4msv6?activetab=pivot:overviewtab

表 7. 推薦されたユーザが期間ごとに行った質問・回答数

Id	11/13-12/12	10/13-11/12	9/13-10/12	8/13-9/12	7/13-8/12	6/13-7/12	5/13-6/12	4/13-5/12	3/13-4/12	2/13-3/12	1/13-2/12	12/13-1/12
2707792	11	18	34	37	60	62	86	49	103	103	120	0
2750966	17	10	13	7	2	3	19	14	15	15	4	0
4204198	0	0	2	10	25	24	6	4	10	10	0	0
4993128	18	27	31	25	16	5	16	28	32	20	32	0
5516172	9	42	20	0	0	0	0	0	0	0	0	0
5880706	0	0	5	10	16	44	51	68	95	95	81	26
6316508	9	21	29	31	20	34	28	33	31	31	22	21
6867048	38	23	37	2	0	0	0	0	1	1	1	2
7662670	20	23	31	34	28	26	22	24	6	6	6	0
9953316	0	2	29	3	50	8	0	0	0	0	0	0

5516172，6867048，9953316 に関しては，一部の期間のみに活動が見られる．

一方で，5880706，9953316 については，テストデータとなる 1 ヶ月間に一度も活動がない．

表 6 に質問 53224244 におけるユーザ 5516172 の推薦順位を示す．3 ヶ月以外のデータでは順位が落ちている．これは，本ユーザが 3 ヶ月以内のみ活動期間があるからと考えられる．

本研究では，トピック分析とネットワーク分析を組み合わせた手法のうちシンプルなものを想定して実験を行っている．しかし，多くの質問に対して回答者推薦を失敗していることから，この手法そのものが実用的とは言えない．しかし，本研究では，ユーザが利用することを想定した評価を行うことを主眼としており，精度そのものの値よりもその変化が重要である．今後，このような観点から新たな回答者推薦手法の評価を行い，より精度の高い手法の構築も重要となる．

7. 結論

本研究では，CQA における回答者推薦手法のうち，複数の手法を組み合わせた手法について，推薦に用いる期間とその推薦結果に関して一検討を行った．実験では，3 ヶ月，6 ヶ月，9 ヶ月，1 年のデータセットを作成し，それぞれについて本研究で想定したトピック分析とネットワーク分析を組み合わせた手法を用いた．その結果として，3 ヶ月のデータセットでのみ正しく推薦された質問があった．これは，正解となるユーザが直近 3 ヶ月のみ活動していたためである．

今後の課題として，本研究では一つの手法を想定して結果を得ていることから結果の一般化が難しい点がある．そのため，既存の手法をより分析し，回答者推薦手法を構成する要素と推薦結果との関係が使用するデータセットの期間によりどのように変わるかを見る必要があろう．また，今回のデータセットの期間は 3 ヶ月区切りとなっている．これをより長期間，また粒度を小さくしたときの変化も重要である．

謝辞　本研究は JSPS 科研費 18H04094 の助成を受けたものです．

参考文献

[1] Guangyou Zhou, Kang Liu, and Jun Zhao. Topical authority identification in community question answering. In *Pattern Recognition*, pp. 622–629, 2012.

[2] Tong Zhao, Naiwen Bian, Chunping Li, and Mengya Li. Topic-level expert modeling in community question answering. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pp. 776–784.

[3] J. Yang, S. Peng, L. Wang, and B. Wu. Finding experts in community question answering based

on topic-sensitive link analysis. In *Proceedings of 2016 IEEE First International Conference on Data Science in Cyberspace (DSC)*, pp. 54–60, June 2016.

[4] Guangyou Zhou, Siwei Lai, Kang Liu, and Jun Zhao. Topic-sensitive probabilistic model for expert finding in question answer communities. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pp. 1662–1666, 2012.

[5] Lin Wang, Bin Wu, Juan Yang and Shuang Peng. Personalized recommendation for new questions in community question answering. *ASONAM '16 Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp. 901–908, 2016.

[6] Ee Peng LIM, Hongfei YAN, Wayne Xin ZHAO, Jing JIANG, Jianshu WENG, Jing HE and Xiaoming LI. Comparing twitter and traditional media using topic models. *European Conference on Information Retrieval ECIR 2011: Advances in Information Retrieval*, pp. 338–349, 2011.

[7] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM, Vol. 46, No. 5, September*, pp. 604–632, 1999.

[8] Fatemeh Riahi, Zainab Zolaktaf, Mahdi Shafiei and Evangelos Milios. Finding expert users in community question answering. *WWW '12 Companion Proceedings of the 21st International Conference on World Wide Web*, pp. 791–798, 2012.

[9] Xianzhi Wang, Chaoran Huang, Lina Yao, Boualem Benatallah, Manqing Dong. A survey on expert recommendation in community question answering. *Journal of Computer Science and Technology July 2018, Volume 33, Issue 4*, pp. 625–653, 2018.

[10] David R. H. Miller, Tim Leek, and Richard M. Schwartz. A hidden markov model information retrieval system. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pp. 214–221, 1999.

[11] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pp. 50–57, 1999.

[12] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research 3*, pp. 993–1022, 2003.

[13] Dom B, Paranjpe D. A bayesian technique for estimating the credibility of question answerers. *In Proc. SIAM International Conference on Data Mining*, pp. 399–409, 2008.

[14] Mohamed Bouguessa, Benoît Dumoulin, and Shengrui Wang. Identifying authoritative actors in question-answering forums: The case of yahoo! answers. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pp. 866–874, 2008.

[15] Liu Yang, Minghui Qiu, Swapna Gottipati, Feida Zhu, Jing Jiang, Huiping Sun, and Zhong Chen. CQArank: Jointly model topics and expertise in community question answering. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pp. 99–108, 2013.

Web サービス・モバイルアプリケーション開発における テスト設計を支援するための標準テスト観点の整備

河野 哲也
株式会社 ディー・エヌ・エー
tetsuya.kouno@dena.com

柏倉 直樹
株式会社 ディー・エヌ・エー
naoki.kashiwagura@dena.com

前川 健二
株式会社 ディー・エヌ・エー
kenji.maekawa@dena.com

菊武 祐輔
株式会社 ディー・エヌ・エー
yusuke.kikutake@dena.com

要旨

Web サービス・モバイルアプリケーション開発におけるテスト設計においてテスト観点を抜け落ちなく網羅的に抽出するために，標準テスト観点を整備し，それをテスト設計で活用する研究を報告する．まず，実際のテスト業務に基づきテストエンジニアの特性を整理し，問題を明らかにする．次に，テスト分析・テスト設計の流れを解析し，標準テスト観点の枠組みとして必要な要件を整理し，その要件に従い枠組みを定める．そして，過去に作成したテストスイートを分析することで様々なテスト観点を導出し，枠組みに従いそれらのテスト観点を整理することで標準テスト観点の実装を行う．最後に，実際の Web サービス・モバイルアプリを対象として標準テスト観点の適用を行い，一定の有効性が確認できた．

1. はじめに

我々は，Web サービスやモバイルアプリケーション（以降，モバイルアプリと略す）分野における機能テスト以降のテスト業務全般を担当している．我々の組織におけるテストプロセスは，テスト分析・テスト設計・テスト実装・テスト実施といった一般的なプロセス[1]を採用している．そのようなプロセスにおいて，テスト分析・テスト設計においてテスト観点を抜け落ちなく網羅的に抽出することが課題となっている．

また我々の組織に限らず，テスト分析・テスト設計領域における業界的認知度が高く，例えばテスト設計の質を競うテスト設計コンテスト[2]や先行事例[3][4][5][6]ではテスト観点にフォーカスした議論がなされている．

以上を踏まえ本稿では，テスト分析・テスト設計領域に焦点を当て議論を進める．

現状，実際のソフトウェアテストの現場では，テスト分析やテスト設計においてテスト観点を網羅的に抽出するために，マインドマップのようなツリー構造による階層化や表構造による一覧化を活用するものの，現実的には経験豊富なテストエンジニアをアサインすることにより解決している．しかしながら，ソフトウェアのビジネス領域の広がりとともに，Web サービス・モバイルアプリ分野の人材の流動性の激しさも相まって，そのような解決では対応できず，経験の浅いテストエンジニアにとってテスト観点を抜け落ちなく網羅的に抽出することが課題となっている．よって，経験の浅いテストエンジニアに対してそれを補うような体系的な仕組み・仕掛けが必要である．

以上を背景として，テスト観点を網羅的に抽出するために枠組みやテンプレートを提案した取組み[7][8]や研究[5][9][10][11][12]がある．これらの取組みや研究は，テスト観点を抽出するための指針やテスト観点を整理するための枠組みであり，多面的なテスト観点を知識として保有するテストエンジニアにとっては有用であるが，経験の浅いテストエンジニアの支援という点で捉えた場合，大きな効果は期待できない．他方，具体的にテスト観点を例示した報告[4][12]もあるが，Web サービス・モバイルアプリ分野に関して言及されていない．

一方，我々の組織ではそのような課題に対して，Web サービス・モバイルアプリ分野で具体的に活用可能なテスト観点を蓄積・整理するような取組みを進めてきた．先行して報告した取組みでは，既存のテストスイートからテスト観点を導出するための手法に言及し，それによって導出されたテスト観点の有効性を確認した[13]．本稿では，この取組みをさらに進め，Web サービス・モバイルアプリ分野における標準的なテスト観点を整理するための枠組みを検討し，標準テスト観点の実装に言及する．

本稿では，第 2 章で本研究が対象とする問題を示し，研究の目的を述べる．第 3 章では，テスト分析・テスト設計の流れを詳細に解析することによりテスト観点を整理

するための枠組みを定める. 第 4 章では, 前章の枠組みに従いテスト観点の実装に関して述べ, さらにそれの使用例を概説する. 第 5 章の適用例では, 前章の実装によって得られた標準テスト観点の有効性を確認する.

2. 問題と目的

2.1. 本研究の問題

本節では, まずテスト分析・テスト設計の流れを概観することで本研究が対象とする標準テスト観点の支援が必要なテストエンジニアの特性を整理し, 本研究の問題を示す. そして, どのような側面のテスト観点の整備が必要なのかを明らかにする.

まず, 文献[1]を援用し, 実際の現場におけるテスト分析・テスト設計の流れを Process Flow Diagram (PFD) [14]によって表現したものを図 1に示す.

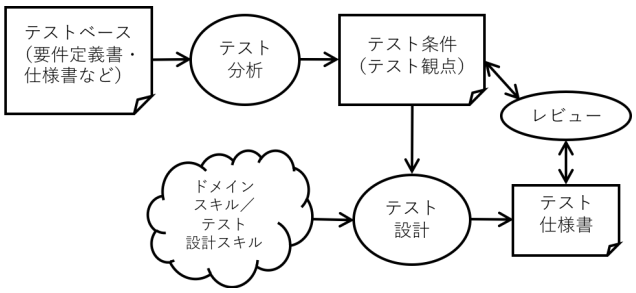


図 1 テスト分析・テスト設計の流れ

テスト分析では, 要件定義書や仕様書などのテストベースを分析することでテスト条件が得られる. 次に, テスト設計では, テスト分析で得たテスト条件をきっかけとして, ドメインスキルを活用しながらテスト条件を補完させつつ, テスト設計スキルをベースとしたテスト技法に従いハイレベルテストケース[1]を作成することでテスト仕様書が得られる. また, テスト条件やテスト仕様書の質を第三者が確認するためのレビューを必要に応じて実施している.

例えば, Web サービスのログイン機能に対する機能仕様書からユーザ ID とその有効・無効やパスワードとその有効・無効, ログイン成功・失敗などがテスト条件として得られる. それらに対して, 例として機能仕様書に「複数回ログイン失敗することでアカウントロック」や「通信不可状態でログイン」などが記載されていない場合は, それらはドメインスキルを活用しながら補完していくことになる. そして, 一例としてユーザ ID の有効・無効とパスワードの有効・無効の組み合わせに対してテスト技法を援用しながらハイレベルテストケースを作成することになる.

以上で述べた補完されたテスト観点は, W モデル[15]の議論でも見られるように開発成果物のレビューで指摘することも可能である. しかし, 本稿では開発成果物の質向上につなげるというレビューの立場ではなく, テスト分析・テスト設計の結果をテスト実装・テスト実行につなげるというテストの立場をとる.

ここで, ドメインスキルは, テストエンジニアが保有するドメイン知識に照らし合わせながら, 必要なテスト条件を抽出するためのスキルとする. テスト設計スキルは, テストエンジニアが保有するテスト技法に関する知識に照らし合わせながら, テスト条件からハイレベルテストケースを設計するためのスキルとする.

また, テスト分析においても一般的にドメインスキルやテスト設計スキルが必要となるが, 本稿では議論を簡単にするために, テスト分析ではテストベースに記述された情報のみからテスト条件が得られるものとする. 加えて, 先に述べた流れでは文献[1]に従いテスト条件という用語を使用したが, 現状のテストの現場では, テスト条件に対応する用語としてテスト観点が広く一般に認知されているため, 辞書的な定義よりも実用性を優先し本稿ではテスト観点という用語で統一する.

以上の流れにおいて, テストエンジニアの技術として極めて重要なのが, ドメインスキルとテスト設計スキルである. そして, ある一定のテスト設計スキルがあれば, テストベースからテスト観点を網羅的に抽出することはできるが, テストベースの情報は常に不足しているという経験則[16]も報告されているようにドメイン知識に関する情報は開発成果物に記載されない傾向が多いため, 一定のテスト設計スキルがあったとしてもドメインスキルが不足している場合, 当該ドメインに関するテスト観点が網羅的に抽出されない. つまり, ドメインスキルとテスト設計スキルという 2 つの側面から捉え, 当該ドメインにおいてテスト観点を抜け落ちなく網羅的に抽出するという点にフォーカスすると, ドメインスキルが重要な位置づけとなる.

一方, Web サービス・モバイルアプリ分野は比較的新しいドメインであるため, テスト設計スキルにおいて習熟しているものの, その分野のドメインスキルにおいては未習熟であるテストエンジニアがアサインされることが我々の組織では多い.

以上で述べたテストエンジニアの特性を表 1に整理する. 表 1が示すように本研究では, テスト設計スキルは習熟しているものの, Web サービス・モバイルアプリ分野のドメインスキルが未習熟なテストエンジニアがテスト設計を行うという制約に対して, 現状, そのようなドメインス

キルの不足を補うような仕組み・仕掛けが体系立って整備されていないことが問題である.

表 1本研究の対象とするテストエンジニアの特性

		ドメインスキル	
		未習熟	習熟
テスト設計スキル	未習熟	そもそもテスト技術不足であるため**対象外**	当該分野では稀な特性であるため**対象外**
	習熟	多くのメンバの特性であるため**本研究の対象**	中長期でアサインできない特性であるため**対象外**

よって, そのような問題を解決するために, Web サービス・モバイルアプリ分野というドメインスキルの側面におけるテスト観点を標準化し, 体系立って利用できるような支援が必要となる.

2.2. 関連研究

テスト観点, もしくはテスト観点と明示しないもののそれに関連する研究は 2 つに大別される.

1 つ目は, テスト観点を抽出・整理するための手法や考え方, 枠組みを提案した研究である. まずツリー構造で多面的にテスト観点を抽出・整理する研究がある[5][7][10]. 次にマトリクス構造で多面的にテスト観点を抽出・整理する研究[3][8][11]がある. これらの研究は, ドメインスキルが十分に習熟しているテストエンジニアにとっては有用であるが, そもそもこのようなスキルが未習熟であると具体的なテスト観点そのものを得ることが難しい. また, 要求仕様書の機能や制約などの要素に対して逸脱の視点でテスト観点を抽出する研究[9]がある. この研究はドメインスキルが未習熟なテストエンジニアにとって, 逸脱の視点でテスト観点を広く抽出することを支援する研究であるものの, 逸脱というレアケースを想定したテスト観点に重点がおかれる.

2 つ目は, テスト観点そのものを提示した取組みや研究である. まず外部機能仕様書の曖昧さに着目して, 具体的なテスト観点を提案した研究[12]がある. 次に, 品質特性や経験則を活用してテスト観点テンプレートを提案し機能テストレベルにおける具体的なテスト観点を例示した研究[4]がある. 加えて, 既出の不具合を分析し複数のテスト観点を例示した研究[3][6]もある. これらの研究では, 部分的に参考になる観点はあるものの Web サービス・モバイルアプリ分野に言及されていないため, その分野のテスト観点を体系的に得ることが難しい.

一方, Web サービスにおけるテストに関して論述した文献[17]がある. これらは一部テスト観点として活用可能なトピックもあるが, Web サービス分野のテストの考え方や手法を体系的に述べたものである.

2.3. 本研究の目的

本研究の目的は, Web サービス・モバイルアプリ分野における標準的なテスト観点を整備し, それを活用することにより 2.1 で述べたような特性を持つテストエンジニアがテスト設計の際に, テスト観点が抜け落ちなく網羅的に抽出できるようにすることである. ここで, 整備された標準的なテスト観点群を標準テスト観点と呼ぶ.

この目的のために, 具体的には, テスト分析・テスト設計の流れを詳細に解析する事でどのような枠組みであればテストエンジニアが有効に活用できるかという点を考察し, 標準的なテスト観点を整理するための枠組みを定める. そして, 過去に作成したテスト仕様書やテストケースから標準的なテスト観点を導出し, 枠組みに従い整理することで標準テスト観点の実装を行う. ここでテスト仕様書やテストケースを総称してテストスイートと呼ぶ. 図 2に本研究の全体像を示す. 図 2は本稿の章・節の構成に対応させているため, 以降, 適宜参照されたい.

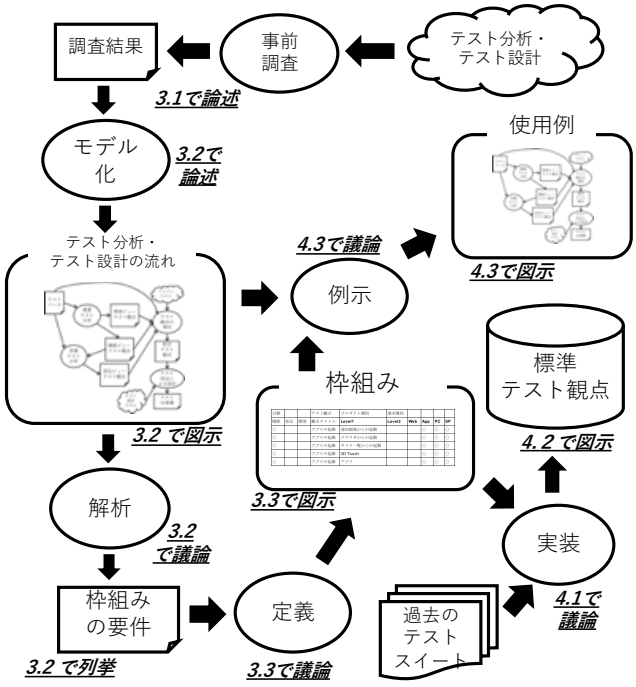


図 2 本研究の全体像

なお, 本稿では標準テスト観点の一部を例示するが,

その全貌を示すことは紙面の都合上難しい．とはいえ，本稿の解析や実装は，本研究の対象とする同様の問題を保有する組織やエンジニアにとって参考になるものと考える．なお，本研究の標準テスト観点のオープン化は今後の課題と考えている．

3. 解析

本章では，テスト観点をどのような枠組みで整理すればテストエンジニアが有効に標準テスト観点を活用できるのかという点でテスト分析・テスト設計の流れを解析し，それに基づき枠組みの要件を整理したのち枠組みを定める．そのために，まず事前調査として経験豊富なテストエンジニアがどのようにテスト分析・テスト設計を行っているのかをヒアリングし，その流れを明らかにする．

3.1. 事前調査

本節では，解析のための事前調査として経験豊富なテストエンジニアにテスト分析・テスト設計の流れに関してヒアリングを行う．

まずテスト業務経験 10～20 年のテストエンジニア 5 名を対象とし，テスト分析・テスト設計の流れに関してヒアリングを実施した．ヒアリングでは 5 名の集合形式の 1 時間程度のミーティングを実施し，2.1 節で示したテスト分析・テスト設計の流れをベースに更に詳細な流れに関して意見交換を行った．

このヒアリングにより，テストエンジニアによっては多少異なる点はあるものの，概ね以下に示すような流れであることが結果として得られた．

- テスト分析では大きく機能と環境の 2 つのビューによって分析されている
- 機能ビューによって抽出された各機能に対して機能を構成する部品のビューによって更に詳細にテスト分析が実施されている
- テスト設計ではテスト分析において機能ビュー・環境ビュー・部品ビューで抽出されたテスト観点をきっかけとして，経験として蓄積したドメインスキルによってテスト観点を補完しつつテスト設計スキルを活用しながらテスト仕様書が作成される

3.2. 調査に基づく解析

本節では 3.1 節のヒアリング結果に基づきテスト分析・テスト設計の流れをモデルで表現し，そのモデルに基づき枠組みの要件を整理する．

まず，3.1 節のヒアリング結果を PFD により表現すると図 3のようになる．以下で，流れの詳細を述べる．

はじめに，概要テスト分析では，テストベースに対して 2 つのビューでテスト分析を行い，環境ビューのテスト観点と機能ビューのテスト観点が得られる．次に，詳細テスト分析では，テストベースと機能ビューテスト観点に対して機能を構成する部品のビューでテスト分析を行い，部品ビューのテスト観点が得られる．なお，概要テスト分析と詳細テスト分析の 2 つを総称してテスト分析と呼ぶ．

ここで部品ビューに関して更に詳細な検討を進めた．部品ビューでは特定の機能に関するテスト観点ではなく複数の機能を横断するような処理やデータ，構造に関するテスト観点である．例えば，メールアドレス入力というテスト観点はユーザ登録機能やログイン機能，問い合わせ機能など複数の機能を横断するテスト観点である．このように複数の機能横断に該当し機能を構成する要素・構造に対してのビューを部品ビューとする．

そして，テスト観点の補完では，環境ビュー・機能ビュー・部品ビューのそれぞれのテスト観点に対して，全てのビューによるテスト観点を統合的に照らし合わせて，ドメインスキルを活用することによりテスト観点の抜け落ちを補完することで統合テスト観点が得られる．また，ハイレベルテストケースの設計を鑑みて，その設計作業を行いやすくするためにテスト観点を整理する作業もここの作業に含まれる．

最後に，テスト技法による設計では，統合テスト観点に対してテスト設計スキルを活用することで，ハイレベルテストケースが記述されたテスト仕様書が得られる．

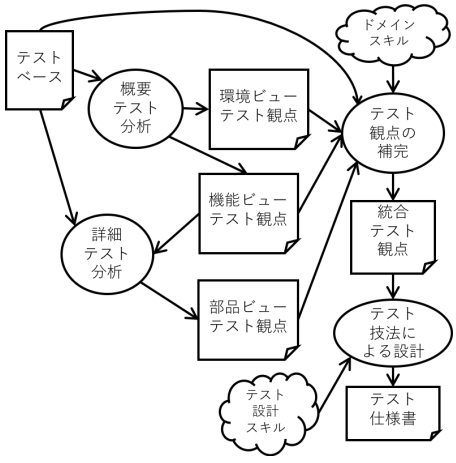


図 3 テスト分析・テスト設計の詳細な流れ

具体的には，ある Web サービスがテスト対象だった場合に，テストベースとしてその機能仕様書が与えられたとすると，概要テスト分析では一例として新規ユーザ登録

やログインといった機能の観点が機能ビューテスト観点に該当し，当該サービスがサポートするOS種別やブラウザ種別といった環境の観点が環境ビューテスト観点に該当する．次に，詳細テスト分析では，ログインといった機能ビューテスト観点に対して一例としてメールアドレス入力やパスワード入力といった機能を構成する部品の観点が部品ビューのテスト観点に該当する．

また，概要テスト分析と詳細テスト分析では，一例として示したこれらテスト観点に対して，機能仕様書には，より詳細な仕様，例えばログイン機能の有効・無効な条件やOS種別・ブラウザ種別の具体的なOS名やブラウザ名及びそのバージョンなどが記載されているため，それらをテスト観点として抽出していく．

そして，テスト観点の補完では，機能ビューテスト観点のログイン機能に対しては，複数端末や複数ブラウザで多重ログインといった仕様は機能仕様書に記載されることは多くないためそれらがドメインスキルを活用することにより，補完されることとなる．一方，環境ビューテスト観点としては，通信状態に関する仕様は機能仕様書に記載されることは多くないためそれがドメインスキルを活用することにより補完され，さらに通信状態という観点に対して弱電波や「通信不可からの復帰」といった詳細な観点が更に補完される．さらに，部品ビューテスト観点のパスワード入力に対しては，「入力不可文字のコピーアンドペースト(以降コピペに略す)」といったテスト観点がドメインスキルを活用することにより補完される．これら補完されたテスト観点を含めて統合的にまとめると統合テスト観点が得られる．

以上のテスト観点の補完のイメージを文献[5]に従い表現した．それを図 4に示す．この図に示すように，「入力不可文字のコピペ」というテスト観点は，以降のテスト技法による設計のためにテスト観点を詳細に整理する．

最後に，テスト技法による設計では，一例として統合された部品ビューテスト観点のパスワード入力に対してテストスキルを活用することで，デシジョンテーブルといったテスト技法を援用し図 5に示すようなハイレベルテストケースが得られる．

以上テスト分析・テスト設計の詳細な流れに基づくと，テスト観点の補完が本研究のスコープとなり，そこにおいてドメインスキルが不足するテストエンジニアに対してテスト観点が抜け落ちなく網羅的に抽出できるようにすることである．よって，ドメインスキルの不足するテストエンジニアでも，図 4に示すような統合テスト観点が得られるようにする必要がある．そのために，先に述べたテスト観点の補完の詳細に基づくと枠組みとして以下に示すような要件が必要となる．

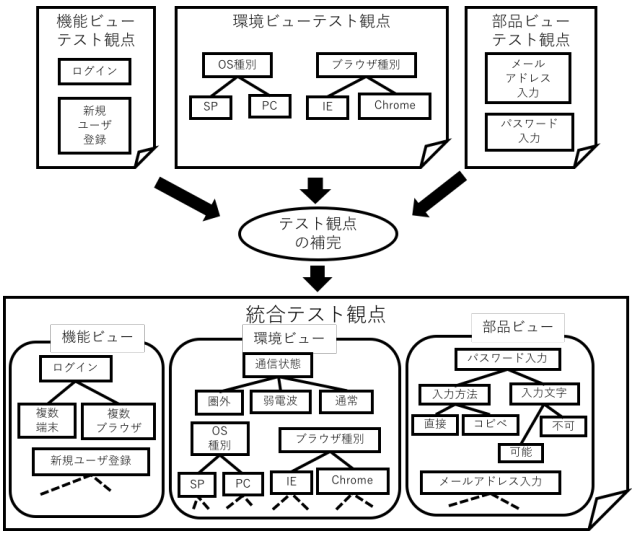


図 4 テスト観点の補完のイメージ図

			1	2	3	4
原因	入力方法	直接	Y	Y		
		コピーアンドペースト			Y	Y
	入力文字	可能	Y		Y	
		不可		Y		Y
結果	入力しログインできること		Y		Y	
	入力できないこと			Y		Y

図 5 ハイレベルテストケースの一例

要件1:標準的なテスト観点群の中から機能・環境・部品のビューでフィルターができること．例えば，機能ビューテスト観点の補完を行っている際に，そのビューのみのテスト観点にフォーカスが当たるようにする必要がある．
要件2:各ビューのトップのテスト観点が一覧から選択できること．例えば図 4の例であれば，ログイン・新規ユーザ登録・アドレス入力・パスワード入力・通信状態・OS 種別・ブラウザ種別がそれに対応する．
要件3:標準テスト観点は階層的に表現さていること．例えば，ログインの配下には複数端末，複数ブラウザなど必要に応じ複数の階層により表現されることによってテスト観点の意図がわかりやすくなるようにする必要がある．

3.3. 枠組み

3.2 節で定めた要件3を満たすためには，標準的なテスト観点を整理するための構造はツリー構造になるが要件1と要件2を満たすためにはスプレッドシートのソート機能を使用したほうが簡便である．また，テスト領域ではスプレッドシートは慣れ親しんだツール[18]であるための，

その導入の容易性を踏まえスプレッドシートで要件を満たすための枠組みを定めることとする.

ビュー				テスト観点		
機能	部品	環境	観点タイトル	Level1	Level2	Level3
	○		ファイルのアップロード	画像ファイル	ファイル種別	JPG
	○		ファイルのアップロード	画像ファイル	ファイル種別	PNG
	○		ファイルのアップロード	画像ファイル	ファイル種別	GIF

図 6 テスト観点を整理するための枠組み

以上まとめると図 6のような枠組みとなる. 枠組みを理解しやすくするために, テスト観点をいくつか例示する.

この枠組では, まず左側に位置するビューの列でテスト観点が該当するそれぞれのビューを○と記載し, フィルター機能を利用することで, それに応じたテスト観点のビューを表示することができる. よって, テスト観点の補完の際に, テストエンジニアがそれぞれのビューでのテスト観点の抜け落ちを確認することができる.

次に, 図 4のそれぞれのツリーの頂点のテスト観点を総称して観点タイトルと呼び, 図6に示す観点タイトルの列で一覧化される. そして, その列から詳細が必要なテスト観点を選択できるようになる. この選択する際も, フィルター機能を使用すれば, 簡便に一覧表示ができる.

加えて, 観点タイトルの配下のテスト観点は, Level1 – Level2 – Level3 といった形式で表し, ツリー構造と比較すると全体の見通しの悪さの弱点はあるものの, 階層的表現は実現されている. それにより, 各階層に位置するテスト観点の意図が理解することができる. ここで図 6では, Level1 から Level3 の階層で示しているが, 第 4 章で述べる実装において, テスト観点を導出し整理した結果, 全てのテスト観点が Level3 までで表現することができたからである. 将来的にはテスト観点を拡充することが考えられるため, 更に階層が深くなることが想定される.

4. 実装

4.1. 実装方法

第3章で定めた枠組みに従い, 標準的なテスト観点の実装を行う. なお, 本節の議論は先行研究[13]で詳述しているため, 本節では実装方法の概要に留める.

まず, 筆者らがテスト業務を担当する当社のプロダクト, 例えば, オートモーティブ関連・ヘルスケア関連のプロダクトに対して過去 2 年間で作成したテストスイートを対象としそれらを分析することでテスト観点を導出する.

なお各プロダクト独自のテスト観点は横断的な汎用性がないため標準テスト観点としては対象外とした. 加えて, テストベースから抽出可能なテスト観点とドメインスキルによって補完されるテスト観点を切り分けながら導出を行っていくことに注意されたい. 例えば図 4のハイレベルテストケースであれば, 入力方法―直接と入力文字―可能文字はテストベースから抽出される観点であるが, 入力方法―コピーアンドペーストや入力文字―不可文字はドメインスキルによって補完が必要なテスト観点となる.

導出したテスト観点を第3章で定めた枠組みに従いテスト観点の一般化・抽象化や詳細化を行いツリー構造で表現し, テスト観点の整理を行う.

以上のテスト観点を 3.1 節で述べた経験豊富なテストエンジニア 5 名によってレビューを行い, 標準的なテスト観点としての十分性やドメインスキルという視点での有効性の確認を行う.

4.2. 実装結果

4.1 節の方法に従い実装を行った結果, 観点タイトルが 90 種類, テスト観点総数は約 700 観点となった. この観点導出から整理までに 340 時間を要した.

標準テスト観点の抜粋を付録に示す. 標準テスト観点はスプレッドシートで実装されているため, そのソート機能を使用しながら, テスト観点の補完する際の関心に合わせて, 必要なテスト観点を検索していくことになる.

4.3. 使用例

本節では, 図 1・図 3のそれぞれのテスト分析・テスト設計の流れに基づき, 標準テスト観点の使用例を示す.

まず, 図 1で示したように実際の現場ではレビューを実施しているため, 図 3にそれを組み込んだ結果を図7に示す. この図に示すように, 標準テスト観点の使用例は大きく 2 つ挙げられる.

使用例1:テスト観点の補完で活用することである. これはテスト分析・テスト設計を行っているテストエンジニア自身がセルフレビューとして活用することを意味する. まず, テスト対象に該当するプロダクト種別・端末種別を選択する. そして, テストベースに基づき機能ビュー・環境ビュー・部品ビューのそれぞれのビューに照らし合わせながら観点タイトルのレベルで抜け落ちが無いかを確認していく. 続いて, 階層構造でテスト観点を捉えた際に, その詳細のテスト観点に抜け落ちがないかを確認する.

使用例2:統合テスト観点のレビューで活用することである. これはテスト分析・テスト設計を行っていないテストエンジニア, もしくはテストマネジャが第三者レビューとして活用することを意味する. 活用方法としては, 使用例1と同様である.

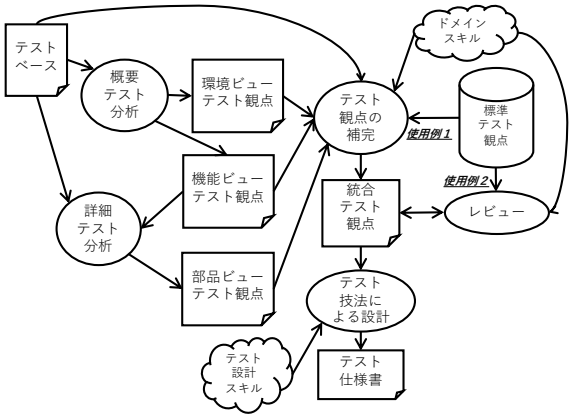


図 7 標準テスト観点の使用例を含むテスト分析・テスト設計の流れ

5. 適用例

本章では, まず概要で適用の方針やテスト対象について述べ, 次に適用方法を示し, 続いて適用結果を示し本研究の有効性を確認する. 最後に本稿の考察を行う.

5.1. 概要

本研究の目的は, Web サービス・モバイルアプリ分野におけるドメインスキルが不足するテストエンジニアに対してテスト観点の抜け落ちなく網羅的に抽出できるようにするための何らかの支援を用意することである.

そのため本適用では, 実際の Web サービスのテスト分析・テスト設計(適用1)とモバイルアプリのテスト分析・テスト設計(適用2)を対象として, 標準テスト観点を活用することで, どの程度テスト観点の補完ができたのかを有効性として確認する. 本適用では, 4.3 節で示した 2 つの使用例を用い, テスト観点の補完を実施する.

適用の概要を表 2に整理する. 適用1では, ある Web サービスの機能追加の開発においてアクセス制御のためのユーザ管理機能をテスト対象として選定し, 表に示すテスト業務経験のテストエンジニア・活用シーンによってテスト業務が遂行されるようにした. 適用2では, あるモバイルアプリの新規開発においてインストール・ログイン・アップデートの機能をテスト対象として選定し, 表に示すテスト業務経験のテストエンジニア・活用シーンによってテスト業務が遂行されるようにした.

ここで適用 1 の担当テストエンジニアを TE1 と呼び, 適用2の担当テストエンジニアをTE2と呼ぶ. また, TE1・TE2 ともに Web サービス・モバイルアプリ分野の経験が豊富とはいえず, ドメインスキルが習熟しているわけではない. よって, TE1・TE2 ともに本研究が対象とするドメインスキルが不足するテストエンジニアに該当する.

表 2 適用の概要

		適用1	適用2
テスト対象		Web	モバイルアプリ
開発形態		機能追加	新規
対象機能		ユーザ管理	インストール ログイン アップデート
担当テスト エンジニア の業務経験	テスト全般	3年	11.5年
	Webサービス ・モバイルアプリ	3年	0.5年
標準テスト観点の活用シーン		使用例 1	使用例 2

5.2. 適用方法

まず, テスト対象機能の機能仕様書をベースにテスト分析を行う.

次に, 適用1ではテスト分析結果に対して TE1 自身が標準テスト観点を活用しテスト観点の補完を行う. 適用2ではテスト分析結果に対して第三者のテストエンジニアが標準テスト観点を活用しながらレビューを行い, テスト観点の抜け落ちを確認することでテスト観点の補完を行う. ここで, 第三者のテストエンジニアは標準テスト観点の使用方法を熟知しているものとし, またレビューの際には, 有効性の確認の目的を踏まえ標準テスト観点のみを利用しテスト観点を補完する.

そして, それぞれの適用において, 標準テスト観点を活用することで, どの程度, テスト観点が補完できたかを適用結果として整理し, 本研究の有効性を確認する.

5.3. 適用結果

5.2 節の適用方法に従い, 2 種類の適用を実施した. 適用 1 のテスト観点の補完では 1 時間を要し, 適用 2 のレビューでは 1 時間を要した. 表 3に適用結果を示す.

適用1では, TE1 がテスト分析を行い 86 のテスト観点が抽出され, TE1 が標準テスト観点を活用し 6 のテスト観点が補完された. また, TE1 およびテスト対象の開発エンジニアが補完したテスト観点の必要性を確認しその結

果，それらのテスト観点全てがテストすべきテスト観点として採用した．補完したテスト観点の一例として，「入力欄に制御文字を含む文字列の入力」や「同一名称のファイルの再アップロード」，「ファイルサイズ0 KB のファイルをアップロード」などが挙げられる．なお参考として，この補完したテスト観点によって不具合は検出されなかった．

表 3 適用結果

	適用1	適用2
テスト分析で 抽出したテスト観点の数	86	36
標準テスト観点活用で 補完したテスト観点の数	6	43
上記のうち テストすべきテスト観点として 採用した数	6	21

適用 2 では，TE2 がテスト分析を行い，36 のテスト観点が抽出され，第三者のテストエンジニアが標準テスト観点を活用し43 のテスト観点を補完した．また，TE2 とテスト対象の開発エンジニアが補完したテスト観点の必要性を確認し，内部仕様上の品質リスクとテスト工数を踏まえて，21 がテストすべきテスト観点として採用した．採用しなかったテスト観点の一例として「端末の日時変更」や「端末の言語設定変更」，「退会済みのアカウントでログイン」などが挙げられる．また採用されたテスト観点として「ストレージ容量不足でインストール」や「通信不可状態からの回復」などが挙げられる．

加えて参考として，適用2 では，採用したテスト観点に基づき作成したテストケースから 3 件の不具合が検出された．これらの不具合は，環境ビューテスト観点に該当する弱電波・通信不可状態でアプリ起動やログインを行うと発生するものであり，このようなテスト観点を補完していなければ見逃した可能性が非常に高い．

適用 1・適用 2 で補完されたテスト観点の数に大きな差が見られたが，TE1とTE2のスキルによる違いではなく，テスト対象の機能仕様書の記載粒度が適用 1 では詳細に記載されており，適用 2 では概要レベルでしか記載されていなかったことが主要な原因である．そのため，適用 1 ではテスト観点を補完する前のテスト分析で詳細にテスト観点が抽出でき，適用 2 ではテスト分析でテスト観点が大まかにしか抽出できなかった．

以上の結果より，適用1・適用2共に標準テスト観点を活用することでそれぞれ6と21のテスト観点がテストすべきテスト観点として採用されたことが分かる．すなわち，これらのテスト観点は，テスト分析においてテスト観点が抜け落ちていたことを意味し，それらが標準テスト観点を活用することで補完でき，テスト観点の抜け落ちに一定の効果があることが分かる．よって，Web サービス・モバイルアプリ分野におけるドメインスキルが不足するテストエンジニアに対してテスト観点の抜け落ちなく網羅的に抽出できるようにするという目的に対して一定の有効性が確認できた．

5.4. 考察

表 3の評価結果が示しているように，補完したテスト観点を含めて統合テスト観点の段階で品質リスクやテスト実行の実現可能性および工数に配慮し，テスト観点の選別が必要なことが分かる．テスト観点の選別では，テスト観点の抽出段階でテストの必要性の有無を判断するのではなく，多面的に抽出された多数のテスト観点に対して必要の有無の判断，もしくは優先順位の設定を行うことが重要となる．本稿で整備した標準テスト観点は，その点において一定の効果が期待できる．また，このようなテストの優先順位付けに関する研究としてはリスクベーステスト（例えば[19][20]など）が挙げられる．

とはいえ，本稿で述べた標準テスト観点は，過去に作成したテストスイートから導出されたテスト観点であるため，それらに該当しているテスト観点に限られる．そのため，十分性という点では，限定的にならざるを得ない．今後は，市場不具合に対して分析を行い，テスト観点を拡充していくことを考えている．

本稿では，テストエンジニアの経験を踏まえたテスト分析・テスト設計の流れに基づき，テスト観点を整理するための枠組みの要件の一つとして，機能ビュー・環境ビュー・部品ビューという構造を提示した．この構造の視点に関する類似の研究として光永らの研究[3]がある．光永らは機能・環境・処理の3つの分類を提案している．これらの違いは，テスト対象のプロダクトや担当しているテストレベルの違いに大きく依存しているものと推察される．

また，本稿で述べた機能ビューと部品ビューは相対的なビューであり，細かく詳細を検討すると部品ビューにも機能に相当するテスト観点，例えば検索やアプリ連携，地図表示などがある．これらの分類に関しては，形式的に一意に定めるのではなく，対象分野のスコープ内（本研究では Web サービス・モバイルアプリ）における実用性を勘案して，分類していくことが重要であると考える．

本稿の 4.3 節では，標準テスト観点を活用するために，テスト観点の補完と統合テスト観点のレビューについて使用例を示した．他の使用例として，開発成果物のレビューにおいて標準テスト観点を活用することが考えられる．この使用例は今後の課題として検討を進めたい．

6. おわりに

テスト分析・テスト設計においてテスト観点を抜け落ちなく網羅的に抽出することが課題となっている. 我々の組織でも同様の課題を抱え, 特に Web サービス・モバイルアプリ分野において, ドメインスキルの不足するテストエンジニアを対象としテスト観点を網羅的に抽出するために, 標準テスト観点を整備し, それを活用することでそれらの課題解決を行うことを本稿で論じてきた.

本稿では, Web サービス・モバイルアプリ分野のドメインスキルが不足するテストエンジニアがテスト設計を行うという制約において, そのようなドメインスキルの不足を補うような仕組み・仕掛けが体系立って整備されていないという問題を取り上げた. その問題を解決するために, テストエンジニア5名にヒアリングを行いテスト分析・テスト設計の流れをモデル化した. そして, そのモデルを解析し, テスト観点を整理するための枠組みの要件を明らかにした. 続いて, その要件に基づき, 枠組みを定め, 枠組に従い標準テスト観点の実装を行った. 標準テスト観点の実装では, 過去に作成したテストスイートを分析し, 多様なテスト観点を導出し, 枠組みに従いテスト観点を整理した. 最後に, 本研究の目的に基づき標準テスト観点の適用を行い, 本適用の範囲に限られるが, 標準テスト観点について一定の有効性を確認することができた.

今後の課題としては, 標準テスト観点のテスト観点そのものの精査・拡充や標準テスト観点の運用方法の構築, 教育を含めた現場導入の検討などが挙げられる.

参考文献

[1] “テスト技術者資格制度 Advanced Level シラバス 日本語版 テストアナリスト Version2012.J01”, http://jstqb.jp/dl/JSTQB-Syllabus.Advanced_TA_Version2012.J01.pdf, (参照 2019-3-15).

[2] ASTER: テスト設計コンテスト, http://aster.or.jp/business/contest.html.

[3] 光永 他, “故障事例によるテスト観点知識ベースの構築とテスト設計への適用”, ソフトウェア品質シンポジウム 2012 (2012).

[4] 稲葉 他, “テスト設計テンプレートを使用したテストケースの充実”, 第 28 回ソフトウェア品質研究会（2013）.

[5] 西, “テスト観点に基づくテスト開発方法論 VSTeP の概要”, http://qualab.jp/materials/VSTeP.130510.color.pdf (参照 2019-3-15).

[6] 吉川, “NGT の記法を応用した不具合分析からのテストの補強”, ソフトウェアテストシンポジウム 2018 東京 (2018).

[7] 池田 他, “マインドマップから始めるソフトウェアテスト”, 技術評論社 (2007).

[8] 湯本, “テストアーキテクチャの具体例” (http://www.jasst.jp/symposium/jasst12tokyo/pdf/A2-6.pdf), ソフトウェアテストシンポジウム 2012 東京 (2012).

[9] 大林 他, “逸脱分析を用いた要求仕様書からのテスト項目抽出手法”, 情報処理学会論文誌, Vol.57, No.4, pp. 1262-1273 (2016).

[10] Grochtmann, M., et al.. “Classification Trees for Partition Testing”, Software Testing, Verification & Reliability (1993).

[11] 吉岡 他, “見通しのよいテストの段階的詳細化の手法”, ソフトウェアテストシンポジウム 2013 東京 (2013).

[12] 河野 他, “ソフトウェア開発における外部機能の仕様記述の曖昧さに着目したテスト項目設計法”, 品質, Vol.41, No.2, pp. 143-151 (2011).

[13] 柏倉 他, “リバースモデリングを用いたテスト観点標準化の取組み”, ソフトウェアテストシンポジウム 2019 東京 (2019).

[14] 清水, “「派生開発」を成功させるプロセス改善の技術と極意”, 技術評論社 (2007).

[15] 鈴木, “W モデルは何か”, ソフトウェアテストシンポジウム 2012 東京 (2012).

[16] 井芹, “品質・仲間・技術と向き合ってテスト設計技術の力を引き出す”, ソフトウェアテストシンポジウム 2018 九州 (2018).

[17] Nguyen, H. Q., “Testing Applications on the Web: Test Planning for Mobile and Internet-Based Systems, 2/Edition”, Wiley (2003). (松村 (監訳), “インターネットアプリケーションのためのソフトウェアテスト”, ソフトバンク パブリッシング (2003).)

[18] The State of Testing Report 2018 (テストの現状調査レポート), https://qablog.practitest.com/wp-content/uploads/2018/10/State_of_Testing_2018_Japanese.pdf (参照 2019-3-15).

[19] 平山 他, “機能モジュールに対する優先度に基づいた選択的ソフトウェアテスト手法の提案”, 電子情報通信学会技術研究報告, ソフトウェアサイエンス, 01, [98], SS2001-6, 1-8 (2001).

[20] 石田 他, “Risk based Testing の実践方法と適用事例からの考察－短期間で要求品質を確保する－”, ソフトウェアテストシンポジウム 2008 東京.

付録 標準テスト観点の抜粋

ビュー			観点タイトル	テスト観点		
機能	部品	環境		Level1	Level2	Level3
○			アプリの起動	通知領域からの起動		
○			アプリの起動	ブラウザからの起動		
○			アプリの起動	3D Touch クイックアクション(iOS)		
○			アプリの起動	アプリショートカット(Android)		
○			アプリの起動	複数回起動		
○			ログイン	複数ブラウザからの多重ログイン		
○			ログイン	複数端末からの多重ログイン	OS別アプリの組み合わせ	Android-Android
○			ログイン	複数端末からの多重ログイン	OS別アプリの組み合わせ	Android-iOS
○			ログイン	複数端末からの多重ログイン	OS別アプリの組み合わせ	iOS-iOS
○			ログイン	複数アカウントの切り替え		
○			ログイン	既存アカウントへのログイン		
○			ログイン	連続認証失敗	失敗回数許容上限の超過	
○			ログイン	ログイン状態の維持		
○			ログイン	キーチェーンアクセス		
○			ログイン	仮登録アカウントでのログイン		
○			ログイン	自動ログイン		
○			ログイン	端末起動直後のログイン		
	○		文字列の入力	文字数	最小文字数未満	
	○		文字列の入力	文字数	最小文字数/最大文字数	
	○		文字列の入力	文字数	最大文字数超過	
	○		文字列の入力	文字数	未入力	
	○		文字列の入力	文字種	半角数字	
	○		文字列の入力	文字種	半角大小英字	
	○		文字列の入力	文字種	半角カタカナ	
	○		文字列の入力	文字種	半角記号	
	○		文字列の入力	文字種	半角スペース	
	○		文字列の入力	文字種	全角数字	
	○		文字列の入力	文字種	全角大小英字	
	○		文字列の入力	文字種	全角カタカナ	
	○		文字列の入力	文字種	全角ひらがな	
	○		文字列の入力	文字種	全角漢字	
	○		文字列の入力	文字種	全角記号	
	○		文字列の入力	文字種	全角スペース	
	○		文字列の入力	文字種	絵文字	
	○		文字列の入力	文字種	機種依存文字	
	○		文字列の入力	文字種	制御文字	
	○		メールアドレスの入力	@マークを含まない		
	○		メールアドレスの入力	@マークの前に文字列がない		
	○		メールアドレスの入力	@マークの後に文字列がない		
	○		メールアドレスの入力	複数@マークを含む		
	○		メールアドレスの入力	カンマを含む		
	○		メールアドレスの入力	コロンを含む		
	○		メールアドレスの入力	セミコロンを含む		
	○		メールアドレスの入力	ドメインが存在しない		
	○		メールアドレスの入力	アカウントが存在しない		
	○		パスワードの入力	入力不可文字のコピーペースト		
	○		パスワードの入力	推測されやすい文字列の制限		
	○		パスワードの入力	登録確認入力エリアとの不一致		
	○		パスワードの入力	強度ポリシー違反		
		○	OS種別	PC	Windows	サポート対象バージョン
		○	OS種別	PC	Windows	サポート対象外バージョン
		○	OS種別	PC	macOS	サポート対象バージョン
		○	OS種別	PC	macOS	サポート対象外バージョン
		○	OS種別	SmartPhone	iOS	最新バージョン
		○	OS種別	SmartPhone	iOS	旧バージョン
		○	OS種別	SmartPhone	Android	最新バージョン
		○	OS種別	SmartPhone	Android	旧バージョン
		○	ブラウザ種別	Chrome	最新バージョン	
		○	ブラウザ種別	Chrome	旧バージョン	
		○	ブラウザ種別	FireFox	最新バージョン	
		○	ブラウザ種別	FireFox	旧バージョン	
		○	ブラウザ種別	Safari	最新バージョン	
		○	ブラウザ種別	Safari	旧バージョン	
		○	ブラウザ種別	Edge		
		○	ブラウザ種別	InternetExplorer	11	
		○	ブラウザ種別	InternetExplorer	10	
		○	ブラウザ種別	InternetExplorer	9	
		○	ブラウザ種別	InternetExplorer	8	
		○	ブラウザ種別	フィーチャーフォン専用ブラウザ		
		○	端末の通信状態	モバイル通信	通信規格	3G
		○	端末の通信状態	モバイル通信	通信規格	4G
		○	端末の通信状態	モバイル通信	通信不可	圏外
		○	端末の通信状態	モバイル通信	通信不可	機内モード
		○	端末の通信状態	モバイル通信	通信不可	SIMカードなし
		○	端末の通信状態	モバイル通信	弱電波	
		○	端末の通信状態	モバイル通信	ローミング	
		○	端末の通信状態	モバイル通信	ハンドオーバー	

システムオブシステムズの現状と課題

落水浩一郎(UIT, Myanmar), 小笠原秀人(千葉工業大学), 日下部茂(長崎県立大学), 艸薙匠(東芝), 熊谷章(金沢諏訪堂の会), 栗田太郎(ソニー株式会社), 塩谷和範(ISO/IEC SC7 エキスパート), 新谷 勝利 (新谷 IT コンサルティング), 鈴木正人(北陸先端科学技術大学院大学), 瀬尾明志(日本ユニシス株式会社),富松篤典(株式会社電盛社), 奈良隆正(コンサル 21 世紀), 羽田裕(日本電気通信システム), 方学芬 (SRA), Huyen Phan Thi Thanh(日立製作所), 堀雅和(株式会社インテック), 矢嶋健一(株式会社 JTB)

要旨

連携する情報システム群(システムオブシステムズ)を対象として, その定義, 問題点, 対処策に対する研究・技術開発の課題を検討する.

1. はじめに

ソフトウェア技術者協会システムオブシステムズ研究会(SIGSoS) は, システムオブシステムズ(以下 SoS と略記)を分類・定義し, 発生する問題を整理し, 対処策を共有する目的で, 2017 年 1 月より, 文献調査, 事例調査を通じて, SoS に関する問題点の整理と解決策の検討 を実施してきた. 本論文では, 活動の成果を基に, SoS に関する今後の研究課題, 技術開発の方向性について検討する.

2. SoS とは

SoS とは, 複数の分散システムや社会技術システムをネットワークで結合したシステムである. 社会技術システムとは, 計算機, ソフトウェア, 装置などの技術的要素に加えて, 人間, プロセス, 規則などの非技術的な要素をシステム設計の対象に加えたものである.

SoS とは, 単なる情報システムの連携ではなく, Maier[1]によれば以下の特徴を有している.

(1) **操作の独立**(Operational Independence)
SoS の構成システムは, 単なるコンポーネントではなくそれ自身一つのシステムとして動作する. それゆえ, 異なる方針や規則に支配され(ガバナンス複雑性), 異なる管理法(管理的複雑性)に支配される.

(2) **管理の独立**(Managerial Independence)
SoS の構成システムは, 異なる組織に所有され, 異なる手段で管理される. それゆえ, 管理や進化に関して異なる規則やポリシーが適用される. この点は SoS の最も特徴的な点の一つである.

(3) **進化的開発**(Evolutional Development)
SoS は単一のプロジェクトによって開発されるわけではなく, 時とともに進化する. このため, システムの異なる部分は異なる技術を用いて構築されがちである.

(4) **創発特性**(Emergency)
創発特性とは, SoS が生成されたあとでのみ明らかになる特性である.

(5) **地理的分散**(Geographical Distribution)
SoS は, しばしば異なる組織にわたって地理的に分散している. システム管理に関する意思決定やセキュリティの維持を困難にする.

Ian Sommerville[3]は, 以下の2点を追加している.

(6) **データ集約**(Data-intensive)
SoS は, 通常のシステムの 10 倍から 100 倍になる, 巨大なデータに依存している.

(7) **異質性**(Heterogeneity).
SoS の構成システムは, 異なる言語や設計法を用いて設計される.

なお,文献[2,3]に、ほぼ同様の SoS の定義がなされている.

3. SoS における問題点

我々は文献[4,5]を基にした調査により, SoS における問題点を以下のように整理した(図 1).

① 各システムが異なる組織によって管理されるため, SoS 構成要素が独自に進化する.
② システム境界が不明確なため要求定義が困難である, また従来の信頼性技術が適用しにくい.
③ Governance Complexity や Managerial Complexity などの新しいタイプの複雑性の制御が必要である
④ 創発特性への対応が困難である.
⑤ また, セキュリティやセフティの問題がこれに絡み, 事態をさらに複雑にしている.

これらの問題は, クローズドなシステム(例えばオペレーティングシステム)を対象として発展してきた伝統的なソフトウェア工学の成果(図 2)では対処しにくい.

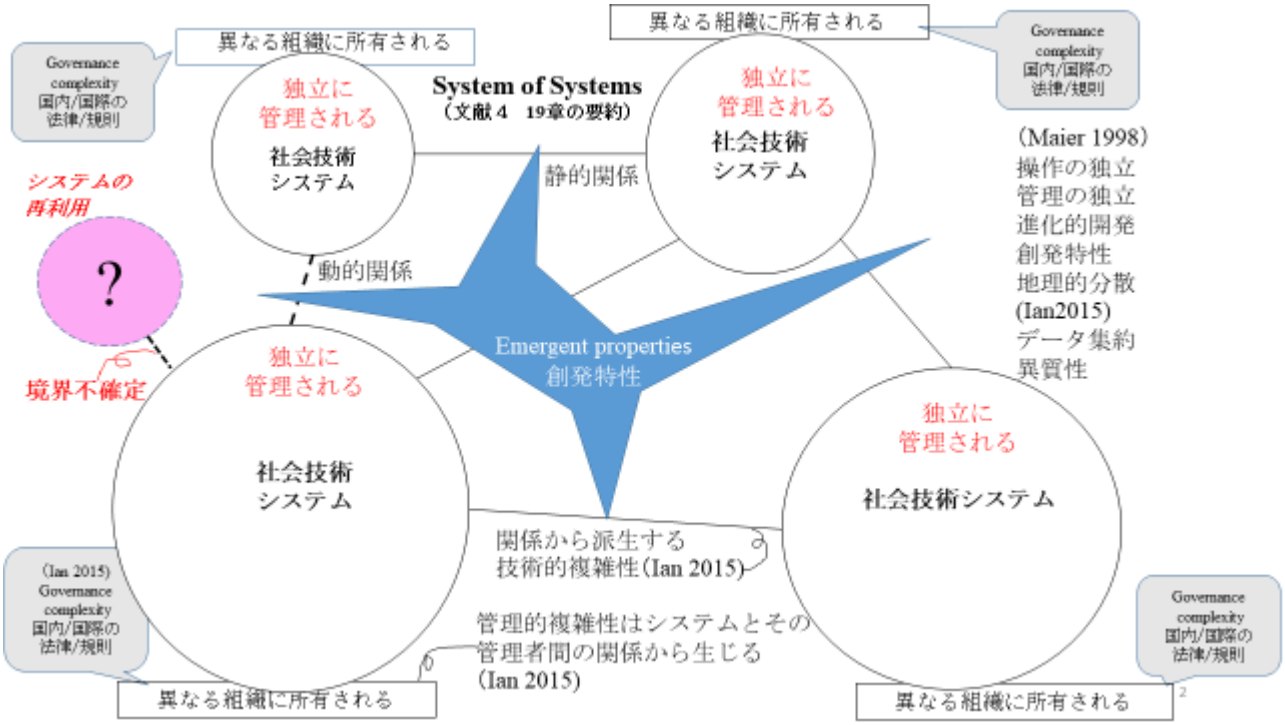
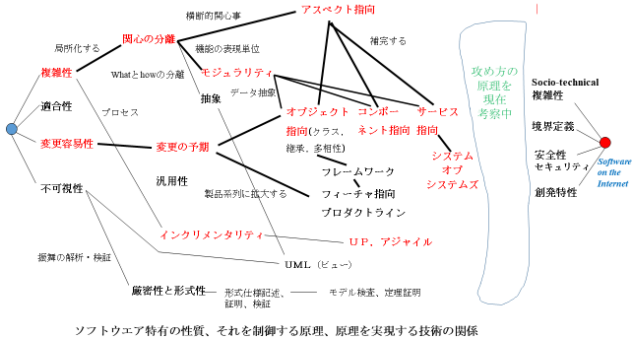


図1　SoS の問題点と課題



ソフトウエア特有の性質、それを制御する原理、原理を実現する技術の関係

図2　伝統的なソフトウェア工学の成果

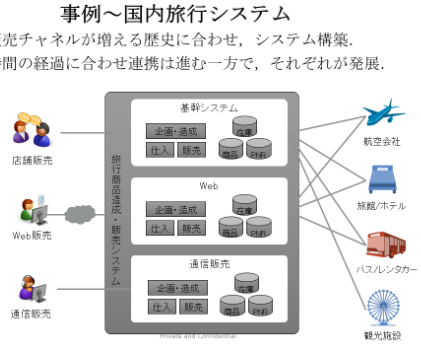


図 3　旅行販売システムの例

4. SoS における問題の一般性と解の探求

現在，世の中の多くの情報システムは SoS が有する問題点を抱えている．例えば，図 3 に示すシステムは、第 2 回研究会で検討した旅行会社の旅行販売システムの例であるが，　航空会社の座席予約システム，ホテルの予約システム，カード会社のシステム等，海外のシステムも含めた多くの情報システムとの連携により実現されている．そこでは，本章の冒頭に示した5つの問題が存在することが確認されている．

SoS　における課題と解決策を手掛かりに，今後，ネットワーク上で連携して稼働する情報システム群に対する課題を整理し，解決策を検討する必要がある．

これまでの調査・検討の結果は以下の通りである．

（1）　初年度は文献[4,5]に基づき，SoS　の定義と問題点を把握した．

（2）　2　年目は創発特性と安全性を対象として，レジリエンスエンジニアリングの研究成果を調査した．レジリエンスエンジニアリングとは，システムの安全性に関して，Safety-II　を目標とするアプローチである．Safety-IIの安全管理では、なぜ平時に業務が成功しているのかを把握し安全活動に活かす[6]．この中でも，システムエンジニアリングズの成果である，システム機能間の不適切な相互作用の抽出を目標とする，エリック・ホルナゲルが提案した FRAM[7]と，システムコンポーネント間の非安全

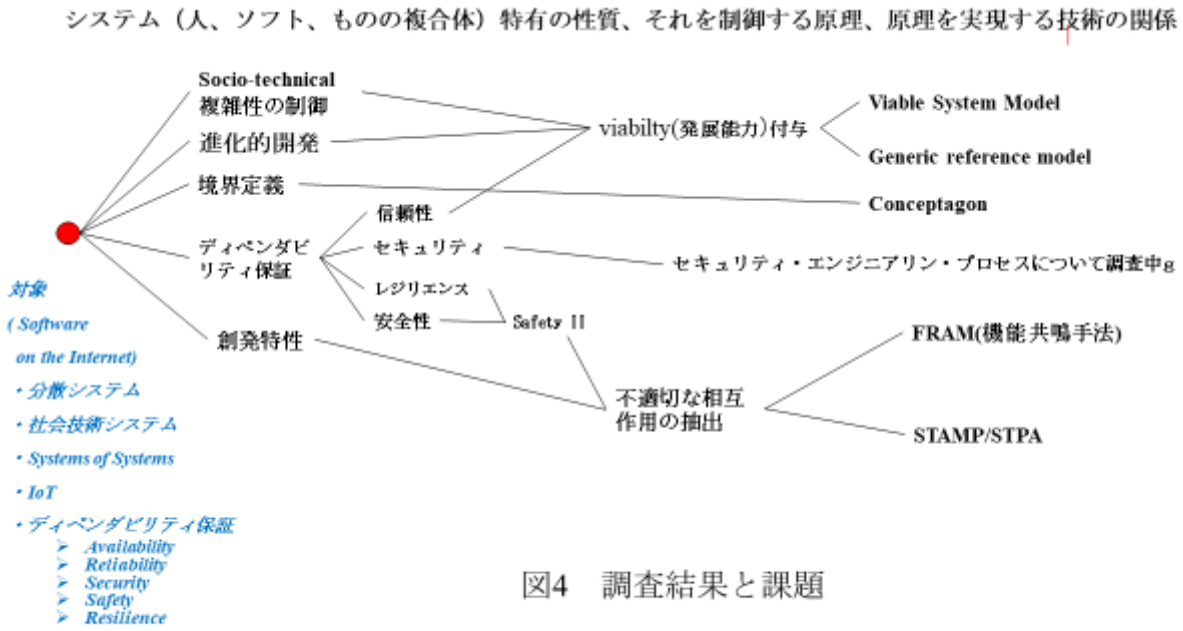


図4　調査結果と課題

な相互作用に注目する STAMP/STPA のアプローチ[8]を集中的に調査した.

5. SoS の課題

ここまでの調査結果に基づき, SoS に関する問題解決のため, 以下の課題の検討は重要である.

(1) **SoS の開発方法とテスト・検証法**: SoS はその構築時から進化する. また, 構成要素システム群の結合時に発生する創発特性を制御する必要がある. さらに, 社会技術システムに特有の社会技術複雑性を制御する手段を開発する必要がある. 環境の変化（例えば法律や規則の変化）に伴って社会技術システムの進化を支持するメタモデルを定義した Viable Systems Model[9]や, Viability 維持など同様の狙いを持つ Derek K Hitchins による Generic Reference Model[10]などを手掛かりに, SoS の進化支援方法論とリファレンスアーキテクチャを開発する必要がある.

(2) **セキュリティとセフティ**: 安全性（とセキュリティ）に関しては, FRAM と STAMP/STPA を基に, 事例研究を推進することにより, SoS への適用法についての知見を深める必要がある.

(3) **コーディネーション支援と境界定義:** SoS の運営には, 組織間のコーディネーションが必須である. 課題 1 と関連づけながら, 組織間のコーディネーション支援の方式を検討する必要がある. また, コーディネーション支援に必要な情報をブロックチェーン[11]により分散共有する技術の開発も興味深い. さらに, 境界定義は SoS の要求を確立するための必須の前提条件である. Conceptagon フレームワーク[12]における 7 つの triplets のうち, 例えば, "システムの内部と外部" などを手掛かりに,オープンシステムの境界定義法を検討することが必要である. なお, Conceptagon, Viable Systems Model, Generic Reference Model については, 文献[13]に解説がある.

6. まとめ

操作・管理の独立, 境界定義, 進化的開発, 創発特性などの特徴を有する SoS に関して,研究技術開発の課題として, SoS リファレンスモデル, 進化支援方法論, 安全性や信頼性の解析手段の必要性を提案した. 提案内容が SoS 固有の問題解決にあたって必ずしも十分であるとはいえないが今後さらに検討を進める予定である.

謝辞

文献[9,10,12]の存在については, 本稿の査読者よりご教示頂いた. 記して謝意を表する

文献

［1］ Maier,M.W.”Architecting Principles for Systems-of-Systems”, Systems Engineering 1(4): pp.267-284, 1998.

［2］ Laura Antul, Judith Dahmann, Aleksandra Markina-Khusid, Ryan Jacobs, “The Systems of Systems(SoS) Primer: A Guide to SoS for all Expertise Levels”, NDIA 20th Annual Systems Engineering Conference, 2017.

［3］ “Systems of Systems(SoS)”, SEBOK

［4］ Ian Sommerville, “Software Engineering – 10thED.”, Pearson, April 2015.

［5］ Edited by Mo Jamshidi, “Systems of Systems Engineering – Principle and Applications”, CRC press, 2009.

［6］ https://resilient-medical.com/risk/resilience-engineering#safety

［7］ エリック・ホルナゲル著,小松原明哲監訳,「社会技術システムの安全分析 FRAM ガイドブック」, 海文堂, 2013 年.

［8］ 日下部 茂,「STAMP/STPA の最新の動向」, SIGSoS 第 5 回例会, 2018 年 5 月

［9］ Janek Richter, Dirk Basten, Applications of the Viable Systems Model in IS Research A Comprehensive Overview and Analysis, 2014 47th Hawaii International Conference on System Science, pp.4589-4599, 2014

［10］ Derek K Hitchins, “The Generic Reference Model”, http://systems.hitchins.net/systems/grm-v4.pdf.

［11］ 赤羽喜治, 愛敬真生 著,「ブロックチェーン仕組みと理論 サンプルで学ぶ FinTech のコア技術」, リックテレコム, 2016.

［12］ The Conceptagon-- A Framework for Systems Thinking and Systems Practice, Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics, pp.3299-3304, 2009.

［13］ 山本修一郎, “システム理論と保証ケース手法の融合に向けた研究課題”, 社会技術システムのモデル化と検証シンポジウム”, 2016.

プログラミング言語横断類似コード断片検出ツールの試作

神谷 年洋
島根大学学術研究院理工学系
kamiya@cis.shimane-u.ac.jp

要旨

複数のプログラミング言語で記述されたプログラム，あるいは，異なるプログラミング言語へ書き換えている途中のプログラムに適用することを目的とした，類似コード断片を検出する動的解析手法を提案する．また，その実装により，*Java* により記述された *300* 行程度のプログラムとそれを *Python* に変換したものの間で類似コード断片を検出した適用例を紹介する．

1. はじめに

ひとつのソフトウェアの実装に複数のプログラミング言語が利用されることが一般的になっている．例えば，Web アプリケーションの場合は，フロントエンドが JavaScript で，バックエンドが Python，Ruby，Java 等で記述されることがある．同様に，ソフトウェアの記述におけるパラダイムシフト（手続き，オブジェクト指向，関数型）やいわゆるレガシーマイグレーションによる別のプログラミング言語（Cobol → Java 等）への書き換えといった要因により，システム全体として見たときに複数のプログラミング言語が混在した記述となる状況が生じている．

複数のプログラミング言語により記述されたソフトウェアの開発・保守の問題として，解析ツールが単一の言語にしか対応していない場合には，システム全体構造を分析したり，品質を評価したりする作業が困難になることが挙げられる．たとえば，コードクローンとは，ソフトウェアのソースコード内に存在する類似コードのことであり，典型的には，開発者がソースコードをコピー＆ペーストすることによって生じる．コードクローンが存在すると，ソースコードを修正する場合には，重複したコード断片のすべてについて修正する（判断を行う）必要が生じるため，ソフトウェア保守上の問題であるとされている．しかし，複数のプログラミング言語により記述されたソフトウェアに対応したコードクローン検出ツールはあまり存在しない（2 節で後述）．

本稿では，著者らが提案した手法 [2] を発展させた，動的解析により異なるプログラミング言語で記述されたコードから類似したコード断片（すなわち，コードクローン）を検出する手法を提案する．さらに，手法の初期的な実装を用いて，Java により記述さた 300 行程度のプログラムと，それを Python に変換したものの間から類似コード断片を検出した試みについても報告する．

2. 関連研究

異なるプログラミング言語で記述されたプログラムの間で類似コードを検出する手法としてこれまでに発表されているものは，[1][4] のような，それらの言語に共通の中間言語を利用して検出するものである．これらの手法を適用するためにはそのような中間言語が存在することが前提となる．これに対して，提案手法の前提は，プログラムを実行して実行トレースを取得できること，および，実行トレースに手続きの呼び出しや戻り，実引数の値や戻り値が記録されることである（3 節で後述）．

複数の言語で記述されたプログラム中の類似コードを求める手法の研究としては，HTML 中で JavaScript で記述された手続きを呼び出すようなものを対象として，呼び出される JavaScript の手続きの類似性を用いるもの [3] がある．

3. 実行トレース中に参照する値の類似性に基づく類似コード断片検出手法

提案する類似コード検出手法のステップを説明する．本節の説明では，プログラムの実行中に参照される値の類似性のみを用いていて，呼び出されているメソッドの名前の類似性等は用いていないことにも留意されたい．

ステップ 1 実行トレースの取得: 対象となる 2 つのプログラム（互いに異なる言語により記述された）を実行し，その際に双方の実行トレースを取得する．実行トレースに記録されるものは，手続き（関数やメソッド）が呼び出されるイベントと戻るイベントについて，それぞれの手続きの名前，ソースコード内での位置，実引数の値（あるいは戻り値）である．

また，補助的に，実行中にリテラル（定数）が参照されるイベントについて，その定数の値が取得可能であれば，それも記録する．

ステップ 2 手続き呼び出しと参照する値の特定: 実行トレースからコールツリーを構築し，コールツリーの節点である手続きの呼び出しのそれぞれ（c）について，その実行の間に参照する値の集合（V(c) とする）を特定する．ここでは，値とみなすことが可能な，真偽値，数値，文字列のみを残し，それ以外は除去した．また，文字列については，文字列を空白や記号で区切ったうえで空白は除去した．

コールツリーの葉に相当する節点の手続き呼び出し c については，その実行中に参照されるリテラルの集合が V(c) となる．それ以外の節点の手続き呼び出し d については，実行中に参照されるリテラル，d が直接あるいは間接的に呼び出す手続き e の実引数や戻り値，および，V(e) の要素を含む集合を作り V(d) とする．

ステップ 3 参照する値が類似した手続き呼び出しの特定: 双方の実行トレース中の手続き呼び出しのペア (c, d) から，V(c) と V(d) が類似しているものを選択する．

選択の基準には様々なものが考えられるが，4 で示した適用例では，ソースコード中のある手続き f について，p の呼び出し c を含むすべてのペア (c_i, d_i) $(i = 1, 2, ..., n)$ のうち，$V(c_i)$ と $V(d_i)$ のコサイン類似度が最大となるペアを選択した．ただし，共通の要素数が 3 に満たないものは取り除いた．

表 1. 類似コード検出例の検出結果集計

正しく特定できた	4 個
呼び出し階層が 1 つ深い（あるいは浅い）	4 個
全く異なる手続きが検出された	1 個
未検出	5 個

表 2. 類似コード検出例の検出結果の内容

Python の手続き (c)	Java の手続き	V(c)	コサイン類似度
main	–	(55)	–
startMatch	←	55	0.528
startGame	←	35	0.478
playerMove	←	20	0.496
put	startGame,-1	17	0.305
display	←	16	0.567
compMove	println*,+1	14	0.546
checkForWin	startGame,-1	10	0.199
move	playerMoved	8	0.208
playerMoved	–	(7)	–
init	–	(6)	–
mark	playerMoved,-1	6	0.178
fixWeights	–	(4)	–
__init__	–	(1)	–

「←」は Python のメソッドに対応する（書き直された元の）Java メソッドが類似コードとして検出されたもの.「–」は類似コードが検出されなかったもの.「+1」（あるいは「-1」）が付されたものは呼び出しの階層が 1 つ深い（浅い）手続きが類似コードとして検出されていたもの.「*」が付された手続きは標準 I/O ライブラリのメソッド.

4. 小規模なプログラムへの適用例

適用対象のプログラムは，ある三目並べのプログラムを Java で記述したもの[1]と，それを著者が Python に書き直したもの[2]である．書き換えはほぼ 1 行ずつ対応するように行われ，双方のプログラムとも，コンストラクタ 1 つを含めて 14 個のメソッド定義を含んでいる．

提案手法の初期的な実装により，これらプログラムの間から類似コード断片である手続きを特定した．表 1 に，検出結果の集計を示す．表 2 に各手続きについての検出

[1] http://www.rosettacode.org/wiki/Tic-tac-toe#Java
[2] https://gist.github.com/tos-kamiya/06e2f6298210001cc54b5b2439cde494

結果，すなわち，Python のプログラムの手続き 14 個のそれぞれについて，提案手法で最も類似度が高いと判定された Java プログラムの手続きを示す．全体的な傾向としては，手続きが参照する値の個数が多いほうが検出が正確になっていた．

4.1. 「未検出」になったメソッドの原因調査

Python のメソッドとその元となった Java のメソッドのペアのうち，表 1 で未検出だった（すなわち，類似コードであると全く判定できなかった）5 個について，`V(c)` およびソースコードにあたってその原因を調査した．

`main`, `playerMoved`: メソッドの処理内容が大きく異なっていた．`main` の場合には，プログラムのエントリポイントにあたるメソッドのため処理の内容はライブラリやクラスのロードであった．ロードすべきクラスの名前などが Java と Python とで大きく異なるために，類似しているとは判定されなかった．`playerMoved` の場合には，このメソッドの処理の内容に int 型の値を文字列に変換して文字列に連結することが含まれていた．プログラミング言語の違いにより，Java では暗黙の型変換，Python では明示的な変換と異なる記述となり，類似しているとは判定されなかった．

`init`, `fixWeights`, `__init__`: メソッドから参照される値の数が少なく類似性が判定できなかった．これらのメソッドの処理の内容の大部分は変数の参照や代入であったが，本実験で用いた実装では変数の参照や代入は実行トレースに記録されていなかった．

いずれも，本実験で用いた実装やアルゴリズムの制約に引っかかったものであり，これらを類似であると判定するためには，（実験に際して用いたパラメータを調整するなどでは対処できず）手法の実装やアルゴリズムの処理を改良すること必要であると考えられる．

5. まとめと展望

実行トレース中で参照する値の類似性により，異なるプログラミング言語で記述されたプログラムの間から類似コード断片を検出する動的解析手法を提案した．三目並べの Java プログラムとそれを Python に書き直したものに適用する例では，参照する値が 10 個以上ある手続きについては，呼び出し階層が 1 つずれたものまで含めれば大多数 (8 個中 `main` を除く 7 個) 特定が可能であった．

今後は，より大きな対象への適用実験による評価を行うとともに，提案手法のアルゴリズムや実装を改善し検出精度・性能の向上を図る．

参考文献

[1] A. Avetisyan, S. Kurmangaleev, S. Sargsyan, M. Arutunian, A. Belevantsev, “LLVM-Based Code Clone Detection Framework”, Proc. 10th International Conference on Computer Science and Information Technologies (CSIT), pp. 100–104, 2015.

[2] Toshihiro Kamiya, “An Execution-Semantic and Content-and-Context-Based Code-Clone Detection and Analysis”, Proc. IEEE 9th International Workshop on Software Clones (IWSC), pp. 1-7, 2015.

[3] 中村 勇太, 崔 恩瀞, 吉田 則裕, 春名 修介, 井上 克郎, “複数プログラミング言語で記述されたソフトウェアからのコードクローン検出”, 情報処理学会研究報告, Vol. 2016-SE-194, No. 7, pp.1–8, 2016.

[4] T. Vislavski, G. Rakić, N. Cardozo, Z. Budimac, “LICCA: A Tool for Cross-Language Clone Detection”, Proc. IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 512–516, 2018.

日本版エンジニアの心理的安全性 ～ トリセツ活動その 2 実践編

増田 礼子
フェリカネットワークス株式会社
Ayako.Masuda@FeliCaNetworks.co.jp

松尾谷 徹
有限会社 デバッグ工学研究所
matsuodani@debugeng.com

要旨

前々回のソフトウェア・シンポジウム *(SS) 2017* では，マイクロマネジメント *(*過介入*)* など，エンジニアにとって危険な職場が実在することを取り上げました．*SS2017* のワーキンググループ *(WG)* では，善良なマネジャーが無意識のうちに，部下に圧力をかける行動に陥る体験を聞き，この問題の根の深さを感じました．その後，*Google* のチームに対する実証研究が公開され，「心理的安全性」がチームの成果や生産性を支配する一番重要な共通因子であることが証明されました．

Google の研究は，データに基づく実証研究であることから，同様に日本におけるこの種の研究をまとめ，さらに，対策として成果がでているケースを調べました．トリセツ第 *2* 弾は，日本版エンジニアの心理的安全性とその対策について提唱します．内容は，*1.* 原因 : 根深い組織文化，*2.* 症状を測る : 客観的な把握，*3.* 対策分類 : なにが実績として成功しているのか，*4.* トレンド : 自立したチーム活動，などです．

1. はじめに

ソフトウェア・シンポジウム (SS) 2017 では，日本のソフトウェア開発現場において，エンジニアにとってネガティブな実態があることをテーマに提唱を行いました．2018 年の ワーキンググループ (WG) では，マイクロマネジメント (過介入) を防ぐためのナレッジを展開し，実践に向けての検討を行って来ました．しかし，現実の世界では，一向に改善される気配はなく，エンジニアにとって心理的に危険な職場が存在します．

その後，Google のチームに対する実証研究 [1] が公開され，「心理的安全性」がチームの成果や生産性を支配する一番重要な共通因子であることが証明されました．Google の研究は，社会正義としてエンジニアの苦境を救う目的ではなく，クリエイティブな成果が求められるチームの成果評価から生まれたものです．この研究では，「圧力型の管理が生産を悪くしている＝マネジメントとして不適切」であることをデータに基づいて実証しました．

日本においても，この種の研究は行われており，日本固有の問題を含め，圧力型管理の副作用についてまとめ，さらに，対策として成果がでているケースを調べました．このトリセツ第 2 弾は，日本版エンジニアの心理的安全性とその対策について提唱します．内容は，1. 原因 : 根深い組織文化，2. 症状を測る : 客観的な把握，3. 対策分類 : なにが実績として成功しているのか，4. トレンド : 自立したチーム活動，などです．

2. 提唱内容

本章では，Future Presentation で取り上げるテーマの概要を示します．

2.1. 圧力型管理の弊害

第 1 の提唱は，経営学や心理学で解明されている，部下に圧力をかける行為の弊害とメカニズムです．この分野の研究は，組織共通の課題であり半世紀以上前から解明されています．心理学者シャートル (C.Shartle) のオハイオ研究 (1957) [3] や，社会心理学者リッカート (R.Likert) のミシガン研究 (1961) [4]，日本では社会心理学者の三隅の PM 理論の研究 [5] です．ミシガン研究のリッカートは，大手保険会社を対象にして調査を行

い，高い生産性をあげる組織は，人間的側面と目標指向的な作業チーム形成を図る「部下中心型」＝関係志向のマネジメント を行っており，低い生産性の組織は，生産を上げるよう圧力をかけようとする「仕事中心型」＝課題志向のマネジメント を行っていたという結論を示しています．

Google の事例は，管理側に圧力源がある旧型の心理的安全より，実務チーム内での圧力に重点を置いています．管理側の圧力に関しては，チームに対する研究とは別に，マネジャーに対する研究としてレポートが公開されています [2].

日本の IT 系における大規模な調査研究であるパートナー満足調査 (PS 調査) は，1,500 名のエンジニアを対象として，仕事満足 (不満足を含む) 調査を行い，主要な因子を示しました [6]．続いて，チーム内の状態を現場力とし，その把握にリーダとメンバの関係性に着目して計測し，プロジェクトの成否を判別することを示しました [7]．この PS 研究の流れと，結論をまとめます．

2.2. 圧力型管理の計測

第 2 の提唱は Tom DeMarco の「測れないものは制御できない」[8] という言葉にあるように，この問題に対するメトリクスを提案します．

心理的安全の加害者となってしまう人の多くは，自身の経験から得た部下支配の価値観が，現代において通用しないことに気付いていません．加害者の同僚は，同僚であるにもかかわらず，心理学における「冷淡な傍観者」となり，被害者を援助することも，加害者に抑制を忠告することもしません．そのため，なかなか組織の中で顕在化せず，顕在化するのは，離職やメンタル問題などで深手を負った後となります．

心理的安全性の課題が顕在化しない原因の一つは，この「冷淡な傍観者」現象にあります．傍観者効果とも呼ばれるこの現象は，ある事件に対して，自分以外に傍観者がいる時に率先して行動を起こさない心理のことで，傍観者が多いほど，その効果は高いとされています [9]. 心理的安全を脅かす圧力型管理の背景には，企業や組織の同僚が傍観者になる現象が生じているのかもしれません．

どのような状態になっているのかを知るには，客観的な計測が必要です．有効な計測方法として PS 研究で用いた「PM ギャップ分析」をベースにした事例を紹介します．PM とは三隅が提案した PM 理論 (Performance function / Meintenance function) に由来します [5]．PM ギャップ分析の調査は匿名であり，圧力源を特定するのではなく，分析から職場風土としての圧力状態を推測します．

2.3. エンジニアのチーム力に注目

第 3 の提唱は，エンジニアの心理的安全性が侵された時の対策です．組織とエンジニア個人の 2 系統が考えられます．

組織としてどのような対策が打てるのでしょうか．発生源が組織文化から生じていれば，「組織文化を変える」が教科書的な対策です．しかし，企業には「利益を追求する」組織の存亡につながる命題があり，対策が遅れる傾向があります．

個人としての対策は，メンタル・マッチョ (コーピング能力強化) になることですが，エンジニア全員をメンタル・マッチョにする対策には無理があります．

現実的な解は，管理的な圧力やメンバ間の圧力からエンジニアを守ることができる実務上のチーム構築です．チームが構築できると，メンバは「冷淡な傍観者」にならないことから，孤立した被害者にならない安心感が生まれます．心理的安全が保たれると，チーム力も相対的に向上します．成功しているチームは，チームビルディングなどによりチームを作る努力をしています [7]．Google の提唱 [1] も，チームを作る役割をチームリーダに期待するものであり，チームリーダが行うべき行動をまとめています．

日本における問題は，多くのエンジニア系管理職やリーダが「良いチーム＝生産性や利益の向上」と概念的に捉えていることです．これは PM 理論の P 機能に偏った概念です．M 機能の充実から生まれる良いチームの体験をしていないと，その捉え方は偏向してしまいます．改善していくには，まず対人スキルとして P 機能だけでなく M 機能が必要であること，M 機能の効果を知り，その実現のために必要なスキルの獲得が必要です．

2.4. これからのチームを GitHub から学ぶ

第 4 の提唱は，激変する IT チームの役割や構成に関するものです．日本の IT 系プロジェクトのチームは，一つの企業だけで構成されることは稀で，多重外注によって構成されています．このような混成チームを「古

い管理概念＝職位」による身分制度で進めることから，さまざまな心理的圧力が生じています．日本型企業システムは成長期を過ぎ，保守や移行が増加し，エンジニアにとって「作る喜び」が減少し,「苦役としての作業」が増加しています．このような環境下では，チームは自立した活動ではなく．指示に従う作業者の孤立した集団となり，現場力が低下します．新興 IT 企業と伝統的な IT 企業とを比較すると，この現象は顕著です．

では，次の時代におけるプロジェクトやチームは，どうなるのでしょうか．背景として，ビジネスにおける価値の主流が，ソフトウェアのプロダクトから，サービスへと移行しているという大きな変化があります．その結果，サービスを生み出し，維持するチームが主流となり，何を作り，維持するかを自立的に実現できることが求められます．

この種の成功事例としては，日本の IT 産業と比較し何桁も大きい成長をした「GAFA」(Google, Amazon.com, Facebook, Apple Inc.) があります．GAFA におけるチームは，どのような運用をしているのでしょうか．GAFA のチームでは，ボトムアップ的に標準化された運用を行っており，GitHub [10] におけるチーム運用と共通しています．なぜ，デファクトスタンダードのような運用になっているのでしょうか．これらのオープン系チームの活性について行われたホットな実証研究を中心に，その驚異的な生産性やスピードのメカニズムや，メンバの貢献評価などについて紹介します．

3. おわりに

日本版心理的安全性について，社会科学的なナレッジと日本における IT 分野における実証研究を基にまとめました．4 つの提唱に関連する情報で，まだ論文等で公開していないものは，順次公開する予定です．

参考文献

[1] Google re:Work, チーム, https://rework.withgoogle.com/jp/subjects/teams/ (accessed:2019/03/10)

[2] Google re:Work, マネージャー, https://rework.withgoogle.com/jp/subjects/managers/ (accessed:2019/05/17)

[3] Halpin, Andrew W and Winer, B James, “A factorial study of the leader behavior descriptions,” Leader behavior: Its description and measurement，pp. 39–51，1957

[4] Likert, Rensis，“New patterns of management,” McGraw-Hill，1961

[5] 三隅二不二,「新しいリーダーシップ」，ダイヤモンド社，1966

[6] 榎田由紀子，松尾谷徹,「Happiness & Active チームを構築する実践的アプローチ: チームビルディングスキルの開発 (＜特集＞ コミュニケーション・マネジメント)」，プロジェクトマネジメント学会誌，Vol.7，No.1，pp.15–20，2005

[7] 松尾谷徹,「現場力は存在するのか: その計測と評価の試み」, ソフトウェア・シンポジウム 2014, pp. 1–8, 2014.

[8] Tom DeMarco, “Software Metrics: Controlling Software Projects,” Yourden Press, 1982.

[9] Wikipedia, https://ja.wikipedia.org/wiki/傍観者効果/ (accessed:2019/05/17)

[10] GitHub, https://github.com/ (accessed:2019/03/01)

ソフトウェア・シンポジウム 2019

日程 ： 2019 年 6 月 5 日（水）～ 7 日（金）
場所 ： 熊本市国際交流会館
主催 ： ソフトウェア技術者協会（SEA）

プログラム [6/5（水） 1 日目]

時間	内容
12:30 13:00	**<受付>** ※1　6F ホール前ロビー 受付場所： 6 階ホール前ロビー
13:00 13:15	**<オープニング>**　6F ホール 実行委員長： 荒木 啓二郎（熊本高等専門学校） プログラム委員長： 栗田 太郎（ソニー）
13:15 14:15	**キーノートスピーチ（1）**　司 荒木 啓二郎 (熊本高等専門学校) ・中西 恒夫 (福岡大学)　6F ホール **講演題目： Evolution of SE Technologies: Looking Back and Looking Forward** 講演者： Professor Emeritus Kyo-Chul Kang, POSTECH（Pohang University of Science and Technology）
14:15 14:30	**<休憩>** Cofee Time　6F ホール前ロビー
	<未来に開く教育革新>　6F ホール 司 小笠原 秀人（千葉工業大学） 米島 博司 (パフォーマンス・インプルーブメント・アソシエイツ)
14:30 15:00	事 **小中学生を対象としたロボット・プログラミング教育とコンテストの実施** 前原 栄輔（NPO 法人 HITO プロジェクト）
15:00 15:30	事 **社会人リカレント教育 enPiT-everi における熊本大学の取り組み** 久我 守弘（熊本大学）
15:30 15:45	**<休憩>**
15:45 16:15	経 **ソフトウェアプロセスの自己改善は自学自習でも可能なのか？** 梅田 政信（九州工業大学）
16:15 16:45	経 **勉強会を活用した組織成長モデル ～参加型勉強会の適用事例～** 伊藤 修司（SCSK）
16:45 17:00	**<休憩>**
17:00 18:00	**キーノートスピーチ（2）**　司 富松 篤典（電盛社）　6F ホール **講演題目： 熊本城の歴史とこれから** 講演者： 津曲 俊博 氏（熊本市経済観光局）
18:00 18:30	**<移動>**
18:30 20:30	**情報交換会** ※2　司 宮田 一平（SHIFT） 会場：熊本市役所 14F　ダイニングカフェ彩（18:15 開場）

時間	<要求抽出・形式仕様記述応用>　5F 大広間 A	<ソースコード解析の応用>　5F 大広間 B
	司 日下部 茂 (長崎県立大学) 張 漢明 (南山大学)	司 神谷 年洋（島根大学） 村上 純 (熊本高等専門学校)
14:30 15:00	研 **要件定義計画を強化するアセスメント項目の提案** 渋谷 公寛（東京海上日動システムズ）	研 **コードクローンへの欠陥混入防止に向けた欠陥混入クローンの特徴分析** 野口 耕二朗（和歌山大学）
15:00 15:30	研 **新規製品開発時の発想支援ツールの提案** 辻脇 優一（国立情報学研究所）	研 **組み込みソフトウェアにおけるコードクローン出現に関する考察** 若林 奎人（京都工芸繊維大学）
15:30 15:45		
15:45 16:15	研 **日本語非機能要件の自動分類における教師あり学習アルゴリズムの評価** 大東 誠弥（和歌山大学）	研 **CNN-BI システムによるプログラムの不具合発見の検証** 小川 一彦（放送大学）
16:15 16:45	研 **軽量形式手法 VDM によるバーチャルマシンの開発** 小田 朋宏（SRA）	研 **LSTM を用いたソースコード内の演算子推定手法** 舟山 優（京都工芸繊維大学）
16:45 17:00		
17:00 18:00		
18:00 18:30		
18:30 20:30		

※1 オープニング以降の受付は、以下の場所となります。
6/5 (水) 6 階ホール
6/6 (木) 12:30 まで : 6 階ホール , 12:30 ～ 14:00: お電話ください , 14:00 以降 : 大広間 B
6/7 (金) 13:00 まで : 大広間 B, 13:00 以降 : お電話ください
※ 事務局の栗田が不在の場合は、070-6429-0240 にお電話ください。(この番号は会期中にしかつながりません。)
※2 情報交換会は 18:15 に開場し、18:30 に “開始” します。お早めにお越しください。アクセスについては P.4 をご覧ください。
※3 2 日目，3 日目はともに、9:00 に開場します。
※4 会議室は 21:30 まで利用可能で、終了時間は WG/TS ごとに異なります。各 WG の開催場所については P.3 をご覧ください。
運営方法・開催時間は、WG ごとに異なります。詳細は各 WG のリーダーまでお問い合わせください。
※5 **ワーキンググループ報告会は、お申し込みいただいた WG と異なるグループに参加されてもかまいません。**

司 : 司会　研 : 研究論文　経 : 経験論文　事 : 事例発表

プログラム [6/6（木） 2 日目]

時間	内容			
	6F ホール	5F 大広間 A	5F 大広間 B	4F 第 1 会議室
	<ふりかえり・問題解決> 司 菅原 広行（ソニー） 中森 博晃 (パナソニック)	**<IoT 関連技術と応用>** 司 石井 宏昌（SCSK 九州） 鈴木 正人 (北陸先端科学技術大学院大学)	**<ソースコード解析の応用>** 司 片山 徹郎（宮崎大学） 奈良 隆正 (NARA コンサルティング)	**<Future Presentation>**
9:10 9:40	事 **ソフトウェアエンジニアリングにおけるコミュニケーション技術の活用** 艸薙 匠（東芝）	事 **IoT システム開発におけるエッジデバイスソフトウェア開発の難しさ** 阪井 誠（SRA）	研 **開発者に着目した Fault-Prediction 技術** 高井 康勢（日立製作所）	9:10-9:50 司 宗平 順己（Kyoto ビジネスデザインラボ） **システムオブシステムズの現状と課題** 落水 浩一郎（UIT，Myanmar）
9:40 10:10	事 **コンセプト&ゴール指向のふりかえりの提案** 小楠 聡美（HBA）	研 **テンソル分解プログラミングの理解支援のための立体パズルの利用** 山本 直樹（熊本高等専門学校）	研 **質問に対する回答者推薦手法に用いられるデータの期間についての一検討** 眞鍋 雄貴（熊本大学）	9:55-10:35 司 小笠原 秀人（千葉工業大学） **プログラミング言語横断類似コード断片検出ツールの試作** 神谷 年洋（島根大学）
10:10 10:20	**<休憩>**			
10:20 10:50	事 **<違い> を捉えよう ～違いの分析から始める問題解決～** 常盤 香央里（WACATE 実行委員会）	研 **マイクロサービス開発への SOA 開発プロセスの拡張可能性の検討** 宗平 順己（Kyoto ビジネスデザインラボ）	研 **Web サービス・モバイルアプリケーション開発におけるテスト設計を支援するための標準テスト観点の整備** 河野 哲也（ディー・エヌ・エー）	10:40-11:20 司 古畑 慶次（デンソー） **日本版エンジニアの心理的安全性～ トリセツ活動 その 2 実践編** 増田 礼子（フェリカネットワークス）
10:50 11:20	経 **なぜなぜ分析とシステム理論に基づく STAMP/CAST の事例による比較 ～ソフトウェアメンテナンスプロジェクトでの問題の分析事例に基づく比較～** 日下部 茂（長崎県立大学）	研 **データベース・アプリケーションへの実行時モデル検査導入手法の提案** 宮永 照二		
11:20 11:30	**<休憩>** Cofee Time 6F ホール前ロビー			
11:30 12:30	**キーノートスピーチ（3）** 司 村上 純（熊本高等専門学校） 6F ホール **講演題目：境界が内部を決めるという数学的世界観** 講演者：古島 幹雄 教授 (熊本大学)			
12:30 14:00	**<休憩>**			
	<ワーキンググループ・チュートリアル>			
14:00	4F 国際会議室 WG1(OS) ：OSS これからの 20 年を想像する 5F 大広間 A WG2(PI) ：働き方改革 ～プロセスを改善して業務効率アップ！ 4F 第 3 会議室 WG3(LE) ：レガシーソフトウェアのブラックボックス化をどう防ぎ，どう解消する？ 4F 第 1 会議室 WG4(OP) ：組織パターンの活用 複数拠点開発を円滑に 5F 小会議室 洋 WG5(QA) ：新しい品質保証のかたちを目指して 5F 大広間 A WG6(FM) ：仕様について考えよう - SPARK/Ada を用いたプログラム検証を題材にして - 5F 大広間 B WG7(PL) ：本当は難しくないソフトウェアプロダクトライン 4F 第 1 会議室 WG8(PT) ：プロセス自己改善手法のビックバン到来？ - PSP/TSP の Creative Commons 化を契機に - 5F 中会議室 WG9(ET) ：エンジニアのターニングポイント：異動，転職，リタイヤ ～ 新たなチャレンジとストレス克服 ～ 4F 第 3 会議室 WG10(RE) ：様々なふりかえり手法の効果的なシチュエーションを考えてみよう！ 4F 第 2 会議室 WG11(ED) ：未来を切り開くソフトウェア教育の可能性を探る 5F 大広間 B WG12(XS) ：ソフトウェア開発の現状と今後の発展に向けたディスカッション 3F 研修室 1 TS1(QU) ：エンジニアのための『質問力』（引き出す力）を伸ばすワークショップ ※ 会議室は 21:30 まで利用可能で、終了時間は WG/TS ごとに異なります。 ※ WG リーダの方へ 13:50 以降に大広間 B に鍵を取りにきてください。 18:15 までに大広間 B に鍵を返しにきてください。遅れる場合は，070-6429-0240 にお電話ください。 (18:00 以降も延長する場合は、当日は大広間 B にいる栗田か富松にお知らせいただくか、お電話ください。)			
18:00				

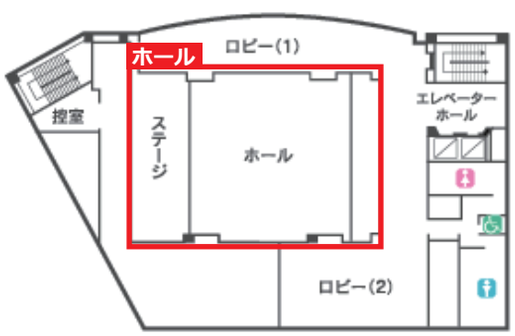
プログラム [6/7（金）3 日目]

時間	内容
	<ワーキンググループ・チュートリアル> ※3 , ※4
09:00 13:30	※ 各 WG と TS の部屋は、2 日目と同じなります。 開始時間・終了時間・お昼休みの時間は、各 WG ごとに異なります。 ※ WG リーダの方へ 9:00 以降に 大広間 B に鍵を取りにきてください。 14:45 までに 6 階ホールに鍵を返しにきてください。
13:30 14:30	**ワーキンググループ報告会** ※5　各 WG の部屋 ソフトウェアシンポジウムでは、興味深いワーキングが毎年開催されておりますが、自身が参加したもの以外はその内容を知ることが困難です。 通常行われる最後の報告の全体セッションは時間の制約もあり、かならずしも躍動感のある議論の内容を肌身で感じることができません。 そこで、2 日間のワーキングに参加された以外の方も 2 日間の議論の最後のまとめの時間帯に参加できれば、ソフトウェアシンポジウム参加者にとって、有益な情報を得れるチャンスが増え、ソフトウェアシンポジウムに参加した意義がより増加するのではと考え、昨年度より実施報告を聞くワーキンググループ報告会のセッションを設けました。 **※ワーキンググループ報告会は、お申し込みいただいた WG と異なるグループに参加されてもかまいません。**
14:30 14:45	**<休憩>**
14:45 15:00	**クロージング**　6F ホール 実行委員長：　富松 篤典 (電盛社) プログラム委員長：　末吉 敏則 (熊本大学)

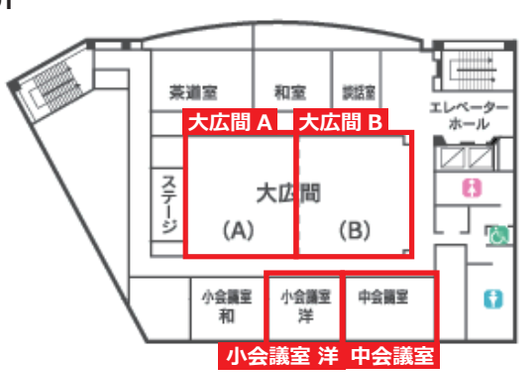
会場案内

熊本市国際交流会館

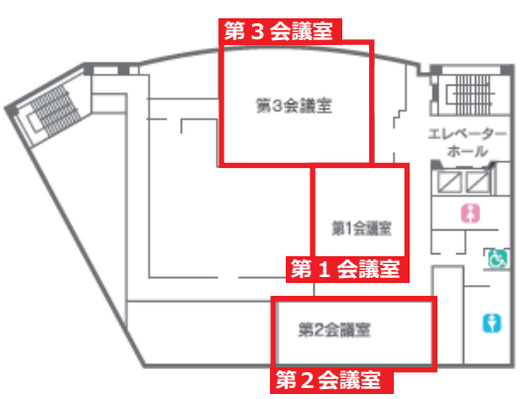
6F



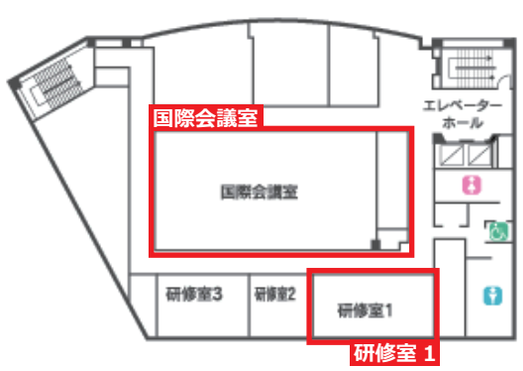
5F



4F



3F



情報交換会（1 日目　18:30 ～）

会場案内

会場：　ダイニングカフェ彩（熊本市役所 14F ）　　**開場 :18:15 ～　開始 :18:30 ～**

※ 情報交換会は 18:15 に開場し、18:30 より “開始” します。お早めにお越しください。
※名札の着用をお願いいたします。

その他

最新情報

最新情報について

SS2019 の最新情報は、随時 Web ページに掲載いたします。公式ページの「新着情報」をご覧ください。
http://www.sea.jp/ss2019/news.php

MEMO

ソフトウェア・シンポジウム 2019
論文集
© ソフトウェア技術者協会

2019 年 6 月 30 日　初版発行

編者　　小笠原 秀人
　　三輪 東

発行者　ソフトウェア技術者協会

〒157-0073 東京都世田谷区砧二丁目 17 番 7 号
株式会社ニルソフトウェア内
ソフトウェア技術者協会 事務局
TEL: 03-6805-8931
https://sea.jp/
ISBN 978-4-916227-25-6

ソフトウェア技術者協会