ソフトウェア・シンポジウム 2017 in 宮崎

論文集

ソフトウェア技術者協会

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 37 回目を迎える 2017 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，SS2016 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．

今回は，“開かれたソフトウェア開発”をテーマに，さまざまな観点からの発表や講演を行います．ソフトウェア開発について，これまでの延長ではない捉え方をしていただく機会として，是非会場に足を運んでいただき，情報交換と深い議論をする場として活用してください！

ソフトウェア技術に関連するさまざまな分野の皆さまからの投稿をお待ちします．参加者の発表にはさまざまな形式があります．研究論文，Future Presentation，経験論文，事例報告です．また，ワーキンググループでの深い議論も予定していますので，ご提案いただきますよう，よろしくお願いします．

※ ソフトウェア開発技術の進化に直接的，間接的に役立つと思われる，「先進的なアイデア」や「さまざまな技術や経験を融合した提案」などを議論するセッション．

開催概要

■ 日程 : 2017 年 6 月 7 日（水曜日）～ 9 日（金曜日）

■ 場所（本会議）: 宮崎市民プラザ

■ 主催 : ソフトウェア技術者協会

■ 後援 : 情報処理推進機構

■ 協賛 : 宮崎県情報産業協会, ソフトウェアテスト技術振興協会, IT 記者会,
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, 電子情報通信学会,
ソフトウェア・メインテナンス研究会, 日本ファンクションポイントユーザ会,
日本ソフトウェア科学会, 組込みシステム技術協会, PMI 日本支部,
日本 SPI コンソーシアム, 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,
アドバンスト・ビジネス創造協会

スタッフ一覧

実行委員会

実行委員長

荒木 啓二郎（九州大学）
小笠原 秀人(東芝)

実行委員

伊藤 昌夫（ニルソフトウェア）
片山 徹郎（宮崎大学）
岸田 孝一（SRA）
栗田 太郎（ソニー）
小松 久美子（帝塚山学院大学）
杉田 義明（福善上海）
鈴木 裕信（鈴木裕信事務所）
萩原 美穂（アートシステム）
中野 秀男（帝塚山学院大学）
奈良 隆正（NARA コンサルティング）
野村 行憲（アイシーエス）
宮田 一平（SHIFT）
三輪 東（SCSK）

プログラム委員会

プログラム委員長

片山 徹郎（宮崎大学）
栗田 太郎（ソニー）

プログラム委員

秋山 浩一（富士ゼロックス）
鰺坂 恒夫（和歌山大学）
安達 賢二（HBA）
天嵜 聡介（岡山県立大学）
伊藤 昌夫（ニルソフトウェア）
岩崎 孝司（富士通九州ネットワークテクノロジーズ）
植月 啓治（パナソニック）
臼杵 誠（富士通）
大平 雅雄（和歌山大学）
落水 浩一郎（University of Information Technology, Myanmar, 金沢工業大学）
小田 朋宏（SRA）
片山 徹郎（宮崎大学）
神谷 年洋（島根大学）
北須賀 輝明（広島大学）
日下部 茂（長崎県立大学）
楠本 真二（大阪大学）
河野 哲也（日立製作所）
後藤 徳彦（NEC ソリューションイノベータ）
古畑 慶次（デンソー技研センター）
阪井 誠（SRA）
酒匂 寛（デザイナーズデン）
佐原 伸（法政大学）
小林 修（SRA）
鈴木 裕信（鈴木裕信事務所）
鈴木 正人（北陸先端科学技術大学院大学）
高木 智彦（香川大学）
田中 康（奈良先端科学技術大学院大学）
張 漢明（南山大学）

角田 雅照（近畿大学）
土肥 正（広島大学）
富松 篤典（電盛社）
中谷 多哉子（放送大学）
中森 博晃（パナソニック スマートファクトリーソリューションズ）
西 康晴（電気通信大学）
野村 行憲（アイシーエス）
野呂 昌満（南山大学）
端山 毅（NTT データユニバーシティ）
本多 慶匡（東京エレクトロン）
松尾谷 徹（デバッグ工学研究所）
松本 健一（奈良先端科学技術大学院大学）
水野 修（京都工芸繊維大学）
宮本 祐子（宇宙航空研究開発機構）
宗平 順己（Kyoto ビジネスデザインラボ合同会社）
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 2017 in 宮崎　目次

■論文・報告　テストとデバック

■論文・報告 コード解析

■論文・報告 要求・安全・品質・メトリクス

■論文・報告 形式手法、プロセス改善

■論文・報告 Future Presentation

■論文・報告 WG にて発表

WG2「テストプロセス改善モデルを理解する」にて発表

WG7「エンジニアリングのトリセツ」にて発表

変更における状態を含むテスト網羅尺度とテストケース抽出法の提案

湯本剛
筑波大学大学院
tsuyoshi.yumoto@gmail.com

松尾谷徹
デバッグ工学研究所
matsuodani@debugeng.com

津田和彦
筑波大学
tsuda@gssm.otsuka.tsukuba.ac.jp

要旨

ソフトウェアの一部に変更を加えた場合，その変更の波及を探る変更波及解析（*Change Impact Analysis*）は実務上の大きな課題である．産業界において，変更にかかる活動は新規開発よりも大きな割合を占めている．ソフトウェアの多様化，複雑化，再利用範囲の増大などから変更波及の範囲が拡大し，かつ安易な変更による弊害など課題が山積している．本論文では，変更波及のうち，状態遷移を持つソフトウェアにおいて，変更波及がデータベースや外部変数などの保持データを介して生ずる場合のテストについて取り上げ，テスト網羅基準とテスト設計手法を提案する．具体的な例を使って手法の適用可能性を確認し，テストケースの数を他の手法と比較し合理的であることを示す．

1. はじめに

ソフトウェアの一部に変更を加えた場合，その変更の波及を探る変更波及解析（Change Impact Analysis）は，実務上の大きな課題である．産業界において変更にかかる活動は，新規開発よりも大きな割合を占めている．ゼロから新規にソフトウェアを開発するケースは稀であり，何らかの流用を基に変更を加える開発が主流となっている．開発方法においてもアジャイルが主流となり，変更の積み重ねによって開発が行われている．

しかし，変更波及を合理的に制御する技術は，ソフトウェア工学とって未完成の分野である [1]．変更の背景は，時代と共に課題を難しくしている．ソフトウェアの多様化と複雑化，再利用範囲の増大などから変更波及の範囲が拡大し，かつ安易な変更による弊害など課題が山積している．これらの課題に対してソフトウェア工学は十分な解を提供できていない状況にある [2]．

本論文では，状態遷移を持つソフトウェアにおいて，変更波及がデータベースや外部変数などの保持データを介して生ずる場合のテストについて考える．課題の一つは，網羅基準である. データフローテストの全使用法（AU 法）[3] を基にして, 変更波及のテスト網羅基準を波及全使用法（IDAU:Impact Data All Used）として提案する．もう一つの課題は，IDAU 法を満たす具体的な変更波及のテストケースの抽出方法である. 変更波及のテストの設計手法として, 順序組合せテストを提案する. 最後に，ここで提案する IDAU 法のコストの評価、すなわちテストケースの数を従来技法である状態遷移テストの S1 網羅基準と比較をして考察を行い, 提案する方法が合理的であることを示す．

2. 変更波及とその解析

本節では，提案する網羅基準やテスト設計手法の前提となる，システムの構成と変更について定義し，変更波及テストの課題について詳細を示す．

2.1. 対象とするアプリケーションの構成

一般的にソフトウェアは，入力 In に対して，何らかの出力 Out を返す．ソフトウェアの機能は，何らかの入

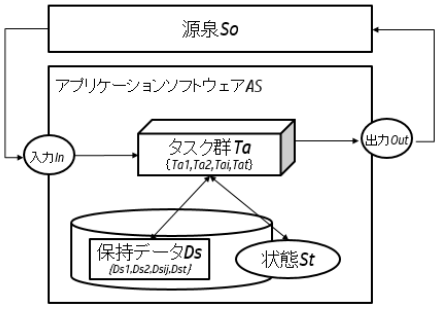


図 1: アプリケーションソフトウェアの構成

力を出力に変換する処理により実現されていると考えられる．この処理を本論文ではタスクと呼ぶ．タスクは，該当のテストレベルからみた入力を出力に変換している 1 つの処理である．そのため，タスクの粒度は，テストレベルによって決まる．ソフトウェアの構成要素であるタスク Ta の出力について考えると，Ta への入力 In だけでなく状態 St と保持データ（データベースや内部メモリに保存されているデータ）Ds の影響を受けると考えられる．例えば，Web アプリケーションにて予約を行うタスクについて考えると，予約が可能か否かを示す状態と，予約オブジェクトの予約状況を示す保持データによって，予約の成否が決まる．

本論文では，対象として図 1 に示すような状態と保持データを持つアプリケーションソフトウェア (AS) について考える．AS の構成要素は，タスク群 Ta と状態 St と保持データ Ds とし，各タスクは外部の源泉 So からの入力 In と So への出力 Out があるとする．タスク群 Ta は，その要素を $Ta = \{Ta_1, Ta_2, \cdots, Ta_i, \cdots, Ta_t\}$ とし，変更タスクを Ta_i とする. 対応する入出力は In_i と Out_i とする．

2.2. 変更と変更波及

AS に対して，何らかの変更を加える場合について考える．変更には，なんらかの意図がある．AS が持つ機能の変更であったり，不具合に対する変更や，性能や保守性の改善のためのリファクタリングであったりする．本論文では，変更の意図については取り扱わず，AS の構成要素（タスク，状態，保持データ）に対する具体的な変更について考える．ただし，テスト実行をするためには，タスクを動かすことが必須となる．そのため，以降の議論はタスクに焦点を絞る．

ひとつの変更 Q について考える．変更 Q は，タスク群 Ta のあるタスク Ta_i に対して行われたとする．変更 Q は，コードの削除や追加を含み，その結果 Ta_i の版 R が $R+1$ に変更される．この変更の結果を Ta_i^R から Ta_i^{R+1} とする．

変更 Q の波及には，3 つのケースが考えられる．

1. 変更波及が無い場合．
2. 変更波及が他のタスクへ波及しない場合．
3. 変更波及が他のタスクへ波及する場合．

3. の変更波及は，タスク間の参照が図 2 に示すように状態と保持データに限られるならば，該当する状態や保持データの参照を介した範囲が限られると考えられる．本論文では，この考え方から波及を受けるタスクを特定し，その合理的なテスト設計について論じる．

変更波及，あるいはその解析に関する研究は古くから行われている．プロダクトラインや UML 図面群を基に依存関係生成モデルを用いて波及解析を行う研究がある [4, 5, 6]．タスク内のデータフローを基に変更波及を詳細に解析した研究がある [7]．データベースなど保有データを基に変更波及解析を行う研究も行われている [8, 9]．一方，状態と状態遷移はマルコフ過程として実装されているので，過去の状態が未来の状態に影響しない．状態遷移に関する波及解析の研究は見当たらないのはそのためだと推測する．

2.3. 状態と変更波及のテスト

変更波及は，保持データを介して変更タスクから他のタスクへ伝達する．状態や状態遷移自身は変更波及に関

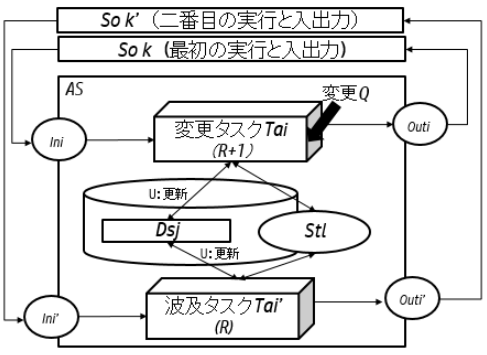


図 2: 変更タスクと変更波及

与しないが，変更波及のテストにおいて与えるデータの順序において状態を考慮する必要が生じる．状態 St において, 考慮する必要がある状態を St_l とする (図 2).

保持データの構成について考える．その要素を $Ds = \{Ds_1, Ds_2, \cdots, Ds_j, \cdots, Ds_t\}$ とし，変更波及を受ける保持データの要素を Ds_j とする．保持データに対する操作は，データのライフサイクルである「C : 生成」「R : 参照」「U : 更新」「D : 削除」を記した CRUD 図で定義する．Ds_j を介して変更波及が生ずるのは，変更タスク Ta_i において「C : 生成」あるいは「U:更新」が行われ，他のタスクでその保持データを利用した場合である．これらのタスクを波及タスクと呼ぶ.

保持データのライフサイクル上の「C : 生成」「R : 参照」「U : 更新」「D : 削除」などの操作は，無条件に行われるのではなく，操作するタスクの制御フローに沿って行われる．制御フローは，図 2 に示すとおり, 2 階層として捉えることができる．上位の制御フローはタスクの実行順序により決定する大きな制御の流れに相当する．個々のタスク内の制御フローが下位にあたる．個々のデータ参照実行文はその制御フロー上の条件文で実行が決定される．条件にはタスクへの入力，保持データ，状態が含まれる．

変更波及を確認するには，変更が波及した保持データを参照するデータフローに沿ってテストを設計することになる．このデータフローを決定するのは，関係するタスクの実行順序とタスク内の制御フローであり，その制御フローを決定する際に状態が影響する．よって，実際のテスト実行においては状態を考慮する必要が生ずる．

2.4. 変更波及テストの網羅基準

テストにおける網羅基準については，その強度を含め制御フローとデータフローの観点から研究が行われ体系が作られている [3]．最も弱い網羅基準は制御フローのみに着目した実行文網羅，次が分岐網羅であり，最も強い網羅基準はデータフローを含めた全パス網羅（All Paths）である．全パステストは，すべての分岐の積であり現実的には実現不可能のため，全使用法（All Uses）が推奨されている [3].

変更波及をテストする場合，波及に関与するデータに着目し，そのデータフローテストを行う．一般的な全使用法は「データを定義したすべての場所から始まり，データを使用するすべての場所に至るまでのパスセグメントを最低限 1 つを含むテストケース」と定義されている [3]．この定義を変更タスクと波及タスクとの関係に置き換え「変更タスクにおいてデータの生成および更新があるデータを使用するすべてのタスク（波及タスク）を 2 つのタスクを実行するまでに経由するルートにかかわらず最低限 1 つ含むテストケース」とし，波及全使用法（IDAU:Impact Data All Used）とする．

3. 順序組合せによるテストケース抽出法

本節では，図 2 で示した変更タスクと波及タスクの組合せを抽出する手法として順序組合せテストを提案する．この手法で抽出するテストケースには，必ず変更タスクを実行した後に波及タスクを実行する，といったように組み合わせるタスクに実行順序があるため，順序組合せと命名した．

3.1. 提案手法に必要となる入力情報

一般的にテストケース抽出のために必要な入力情報をテストベースと呼ぶ [10]．提案手法に必要なテストベースは DFD，ER 図，CRUD 図である．以下，DFD，ER 図，CRUD 図を簡潔に説明する．

- DFD（データフローダイアグラム）

  DFD はシステムにおけるデータの流れを表現した有向グラフであり，要求分析において用いられている．DFD はデータ指向設計の要として用いられ，オブジェクト指向設計においても抽象化する前段階として実践の場で用いられている．

  DFD は，最上位のコンテキストレベルから階層として詳細化され，各階層は 1 枚以上の DFD から成る [11]．テストベースとして用いる場合，テストの範囲は DFD で与えられるとする．DFD の階層はテストレベルに対応し，階層が下がると単体テストとなり，上がると統合テストとなる．

  DFD はノードとエッジからなる．ノードは 3 種類の要素である N 個のタスク（プロセス）Ta と，M 個の保持データ（データストア）Ds と，L 個の源泉（外部エンティティ）So から構成されている．3 種類の要素を一意に特定する際は Ta_i，Ds_j，So_k と表記する．

エッジは，ノードからノードへのつながりを有向線分で表記している．エッジはデータの流れを表しており，制御の流れは表していない．エッジの特定は，起点ノードと終点ノードを用いて行う．ある特定のタスクからデータストアへの入力がある場合のエッジの特定は，Ta_i/Ds_j となり，源泉から出力してタスクで処理をする場合は，So_k/Ta_i と表す．

- ER 図

  ER 図はシステムにおけるエンティティ間の関係を示す図であり，UML のクラス図に対応している．DFD では表現できないエンティティの詳細化やエンティティ間の関係について示しており，DFD と共に用いられている．

  ここでは，DFD のデータストア Ds_j が持つエンティティと，CRUD 図の対応から，後述する拡張 CRUD 図を作成するために用いる．よって，テストベースとしては，システムすべての ER 図を必要とするものではない．

- CRUD 図

  CRUD 図とは，タスク Ta_i からデータストア Ds_j への C：生成，U：更新，R：参照，Ds：削除の操作を表した図である [12]．CRUD 図から，DFD と ER 図では表現されていないタスクのエンティティへの操作を知ることができる．

  本論文では，タスクがデータストアに対して行う操作を特定するために CRUD 図を用いる．タスク Ta_i のデータストア Ds_j に対する操作が U：更新であればタスクによる操作はエッジを介した操作として Ta_i/Ds_jU と表記する．ただし，タスクが操作するデータストアが 1 つだけの場合は，Ta_iU といった省略した表記を使う．

3.2 順序組合せテストの概要

提案する手法は，2 タスク間の順序組合せを対象とする．2 タスク間の順序組合せの抽出は以下のルールを適用する．

- ルール 1：変更タスクの特定

  対象とする DFD 内の変更タスクのうち，データストア Ds へ出力エッジを持つタスクを選択し，順序組合せの変更タスク群 $P\{Ta\}$ とする．変更タスク群からの出力するデータストア群を $P\{Ds\}$ とする．

- ルール 2：波及タスクの特定

  ルール 1 で求めた $P\{Ds\}$ からの入力エッジを持つタスクを波及タスク群 $S\{Ta\}$ として特定する．

- ルール 3：順序組合せテストケースの抽出

  拡張 CRUD 図を基に変更タスク群 $P\{Ta\}$ とそのデータストア群 $P\{Ds\}$ を介する波及タスク群 $S\{Ta\}$ を組合せ，順序組合せのテストケースとする．

以降からは，順序組合せを抽出してテストケースとするまでの実施手順を詳細に説明する．

表 1: 拡張 CRUD 図

タスク	データストア			源泉		
	Ds_1	...	Ds_j	So_1	...	So_k
Ta_1						
...						
Ta_i						

3.3 ルール 1：変更タスクの特定

ルール 1 を用いて変更タスクとそのデータストアを特定し，拡張 CRUD 図の変更タスク部分を作成する．

拡張 CRUD 図とは，テストベースとして与えられた DFD，ER 図，CRUD 図から $P\{Ta\}$ の各 Ta_i と関連する So_k，そして $P\{Ds\}$ となる Ds_j の関係を追加して作成したものである．表 1 に拡張 CRUD 図の表記を示す．拡張 CRUD 図のデータストアに対する情報は C，U，R，D のいづれか，または組合せか空白である．源泉に対する情報は In か Out，または組合せか空白である．空白は関係が無いことを示す．

1. 源泉からの入力エッジを持つ変更タスクの特定

   テストケースは，外部からのテスト対象への入力から，外部への出力結果を確認するものであるため，テスト入力とテスト結果のペアで構成されている．そこで，テスト対象範囲の外からの入力，即ち So_k からの入力エッジを持つ Ta_i を見つける必要がある．この特性を持ったタスクのうち，さらに変更の

表 2: 中間の拡張 CRUD 図の例

タスク	データストア			源泉		
	Ds_1	Ds_2	Ds_3	So_1	So_2	So_3
$Ta_1[St_1]$	CU			In		
$Ta_3[St_1]$		C		In		

あるタスク群を変更タスクの集合となる $P\{Ta\}$ 候補とする．変更が特定の状態でのみ起こり得る場合は，その状態を［St_l］と記載する．

2. データストアへの出力エッジを持つタスク特定

   Ta_i から Ds_j への出力エッジは，C か U か D の操作を行うことを意味する．CRUD 図から該当する出力エッジを持つ Ta_i を選択する．$P\{Ts\}$ 候補の中から，該当する Ta_i を選び，変更タスク群 $P\{Ta\}$ を確定する．

3. 中間の拡張 CRUD 図作成

   拡張 CRUD 図には，変更タスク群 $P\{Ta\}$ に該当する So_k から Ta_i への入力 (In)，もしくは Ta_i から So_k への出力 (Out) の情報を付加する．特定した Ta_i に対して，入力となる So_k に In を記入し，Ds_j については CRUD 図を参照して C か U か D かその組合せかを記入する．中間の拡張 CRUD 図として例示した表 2 では，3 つの源泉 $\{So_1, So_2, So_3\}$ と 3 つのデータストア $\{Ds_1, Ds_2, Ds_3\}$ があり，2 つのタスク $\{Ta_1[St_1], Ta_3[St_1]\}$ が変更タスクである．この段階で作成する拡張 CRUD 図は，作業途中のものである．

表 3: タスク間のデータ共有の組合せパターン

		$P\{Ta\}$			
		C	R	U	D
$S\{Ta\}$	C	×	×	×	○
	R	○	-	○	-
	U	○	-	○	×
	D	○	-	○	×

表 4: 完成した拡張 CRUD 図の例

タスク	データストア			源泉		
	Ds_1	Ds_2	Ds_3	So_1	So_2	So_3
$Ta_1[St_1]$	CU			In		
$Ta_3[St_1]$		C		In		
$Ta_2[St_1]$	R				Out	
$Ta_5[St_1]$		RU				Out

3.4 ルール 2：波及タスクの特定

ルール 2 を用いて波及タスク群を特定し，拡張 CRUD 図へ波及タスク部分を追加し図を完成させる．

1. データストアを介した波及タスク特定

   先に作成した中間の拡張図から変更タスクの操作が C か U か D であるデータストアに着目する．着目したデータストアに対してエッジを持つタスクが波及タスクの候補となる．波及タスクとして選択するタスクは表 3 に示す表の”○”の組合せに該当するタスクである．波及タスクは， C : 生成，U : 更新，D : 削除を選択する．”-”をつけた組合せは，データストアを介した影響が生じないため，組合せテストの対象としない．”×”をつけた組合せは仕様上有り得ない組合せであり，ありえないことの確認は，順序組合せを網羅しなくともテストできるため，組合せテストの対象としない．

2. 拡張 CRUD 図の完成

   波及タスク候補のうち，源泉に出力エッジを持つタスクを波及タスクとして特定する．波及タスクの特性を DFD より読み取り，特定する．特定した波及タスクを拡張 CRUD 図に追記し完成させる．

   完成させた拡張 CRUD 図の例を表 4 に示す．この例では，データストア Ds_1 から源泉 So_2 への流れをタスク $Ta_2[St_1]$ が行い，データストア Ds_2 から源泉 So_3 への流れをタスク $Ta_5[St_1]$ が行っていることを示している．

3.5 ルール 3：順序組合せテストケースの抽出

拡張 CRUD 図を基に順序組合せのテストケースを抽出し，テストケース表を作成する．

表 5: 順序組合せテストによる論理的テストケースの例

No	論理的テストケース	順序組合せ
1	概要	$Ta_1C \rightarrow Ta_2R$
2	概要	$Ta_1U \rightarrow Ta_2R$
3	概要	$Ta_3C \rightarrow Ta_2R$
4	概要	$Ta_3C \rightarrow Ta_2U$

1. 変更タスクと波及タスクの組合せを抽出

   拡張 CRUD 図から変更タスクを選ぶ．先に作成した拡張 CRUD 図の例（表 4 を参照）であれば，$Ta_1[St_1], Ta_3[St_1]$ である．次に変更タスクが操作しているデータストアと，それを操作している波及タスクを対応付ける．例では，$Ta_1[St_1] \xrightarrow[Ds_1]{} Ta_2[St_1]$ と $Ta_3[St_1] \xrightarrow[Ds_2]{} Ta_5[St_1]$ である．

2. データストアに対する操作の組合せ

   操作の組合せとは変更タスクと波及タスクの操作の組合せである．表 4 の例であれば，変更タスクのデータストアに対する操作である Ta_1 は，Ds_1 に対して C と U の操作を行っている．波及タスク Ta_2 の操作は R である．組合せは $C \rightarrow R$ と $U \rightarrow R$ となる．変更タスクと波及タスク間に介在するデータストアが 1 つであれば $\xrightarrow[Ds_1]{}$ を省略して $\rightarrow$ で表してもよい．また変更の発生条件となる状態が 1 つであれは，$[St_1]$ を省略してもよい．表 4 の例における全組合せは，$Ta_1C \rightarrow Ta_2R$，$Ta_1U \rightarrow Ta_2R$，$Ta_3C \rightarrow Ta_2R$，$Ta_3C \rightarrow Ta_2U$ の 4 個である．

3. テストケース表の完成

   変更タスクと波及タスクの操作の組合せをテストケースとしてまとめる．表 5 にその例を示す．概要の部分は，当該組合せが持つ入力の条件や出力の特性を仕様から抜き出して記載する．

以上の手順で，順序組合せテストに必要なテストケースを抽出する．ここで用いたテストケースとは，ISTQB の定義による論理的テストケースに相当する [10]．具体的な値や期待結果，該当の処理までの状態を遷移させていく手順まで定義した記述を具体的テストケースと呼ぶが，本論文では扱わない．

4 順序組合せテストの適用評価

本節では，旅行代理店向けフライト予約システムの仕様を用いて，3 節で述べた実施手順を適用し，順序組合せが抽出できることを確認する．

4.1 題材の概要

フライト予約システムの概要を以下に示す．

＜フライト予約システム概要＞
・旅行代理店用に開発したフライト予約サーバにインターネット経由でアクセスできる専用のクライアントアプリケーション．
・旅行代理店の窓口での利用を想定しており，ユーザ認証されたユーザのみ利用可能である．
・旅行代理店の窓口数（クライアント数）は 50 としており，同時に予約処理を行うことができる．
・旅行代理店にて取り扱う全ての航空会社の飛行機の予約が可能である．
・本システムは，フライト予約サーバを仲介して複数の航空会社のシステムと同期をする．
・チケット情報や残チケット数は同期することで最新に更新される．
・フライトの新規予約、予約内容の更新，削除が可能である．更新と削除は新規予約したユーザのみ可能である．
・以下はシステム範囲外
 –チケット代金の決済（別システムと連携して行うため）．
 –マスタ情報設定（他システムとの共用マスタ設定アプリケーションがあるため）．

図 3: フライト予約システムの概要

題材となるフライト予約システムの仕様は，本研究の一環として評価実験の際に題材として使っているものである [13]．テスト対象の分析と，テストケース設計に関する用語は，国際標準である ISO/IEC/IEEE29119 の定義に従い，テスト対象の論理的なサブセットを機能セット（Feature set）と呼ぶ [14]．本論文では，表 6 の新規フライト予約を変更が入った機能セットとする．(表 6 の最後の列にある○が変更を表す.）新規フライト予約からテストケースを抽出するための前提として用意した仕様は，新規フライト予約に関連する DFD（図 4），ER 図（図 5），CRUD 図（表 7）とする．DFD に含まれるタスク数 N は 6，データストア数 M は 2，源泉数 L は 4 である．

4.2 ルール 1：変更タスクの特定

テストベースである DFD に含まれるタスク数 N は 6 であるが，変更が入った新規フライト予約の変更タスクは，表 7 の CRUD 図を確認するとフライト検索 Ta_1 とフライト予約 Ta_2 であることがわかる．図 4 から，Ta_1 と Ta_2 の外部入力を確認する．Ta_1 は，顧客 So_1 から

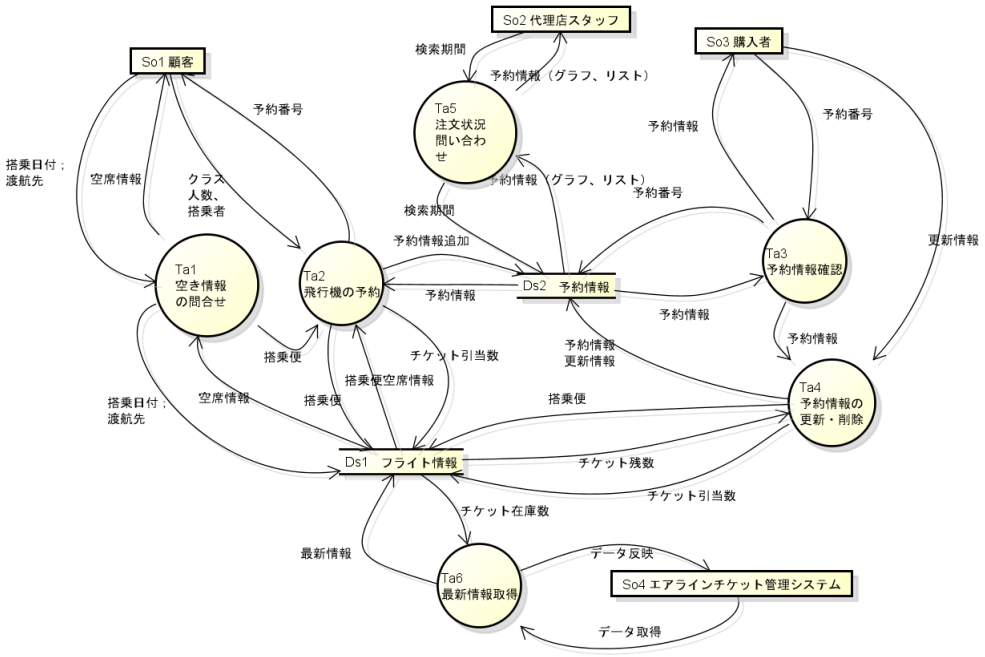


図 4: 新規フライト予約の DFD（一部分）

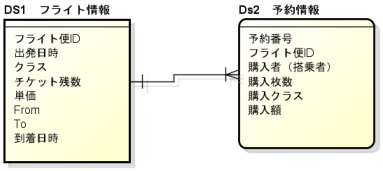


図 5: 新規フライト予約の ER 図（一部分）

搭乗日付と渡航先を外部入力し，Ta_2 は，顧客 So_1 から搭乗便，クラス，人数，搭乗者を外部入力している．

続いて，Ta_1 と Ta_2 の内部出力を確認する．Ta_1 はフライト情報 Ds_1 に対して検索条件を与えているのみで内部入力はしていないため，変更タスク群 $P\{Ta\}$ からは除外する．Ta_2 がフライト情報 Ds_1 で U，予約情報 Ds_2 で C を行っていることが表 7 から読み取れる．これらから，拡張 CRUD 図（表 8）を作る．表 8 から，ルール 1 に適合する Ta_2/Ds_1U，Ta_2/Ds_2C を特定できる．

表 6: フライト予約システムの機能セット一覧

テストアイテム	機能セット	変更
フライト予約システム	メニュー	
	ログイン	
	新規フライト予約	○
	予約変更 & キャンセル	
	予約一覧	
	予約グラフ	
	同期処理&	

4.3 ルール 2：波及タスクの特定

ルール 2 にて波及タスク群 $S\{Ta\}$ を抽出するために，タスクの外部出力を図 4 の DFD から調べる．$P\{Ds\}$ に含まれる Ds_1 と Ds_2 にエッジを持ち，かつ So へ出力するタスク群が $S\{Ta\}$ 候補である．図 4 では，全てのタスクが Ds_1 および Ds_2 からのエッジを持つ．しかし，So への出力に着目すると，Ta_4 は該当するエッジがな

表 7: 新規フライト予約の CRUD 図（一部分）

機能セット	タスク		エンティティ	
			Ds_1 フライト情報	Ds_2 予約情報
新規フライト予約	Ta_1	フライト検索	R	
	Ta_2	フライト登録	RU	C
予約変更	Ta_3	予約情報確認		R
キャンセル	Ta_4	予約情報修正	RU	UD
予約リスト予約グラフ	Ta_5	注文状況		R
同期処理	Ta_6	最新情報取得	CU	

表 8: 新規フライト予約の中間拡張 CRUD 図

タスク	データストア		源泉			
	Ds_1	Ds_2	So_1	So_2	So_3	So_4
Ta_1						
Ta_2	U	C	In			

いため，$S\{Ta\}$ 候補には入らない．

$S\{Ta\}$ 候補のうち、表 3 の ”○” がつく組合せに相当する Ta_i が，ルール 2 で特定したタスクとなる．本節の例の場合，$P\{Ta\}$ での操作は，C と U であるため，$S\{Ta\}$ 候補の中で C の操作をする Ta_i 以外は全てルール 2 で特定したタスクとなる．

これらに該当する Ta_i と Ds への CRUD 操作，そして So への Out を追記し，表 9 を完成させる．

4.4 ルール 3：手順 順序組合せテストケースの抽出

表 9 の拡張 CRUD 図から変更タスクと波及タスクの組合せを抽出する．抽出した変更タスクと波及タスクの組合せに対して，データストアに対する操作を明記したものは以下のとおりとなる．

- $Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_1R$
- $Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_2/Ds_1U$

表 9: 新規フライト予約の拡張 CRUD 図

タスク	データストア		源泉			
	Ds_1	Ds_2	So_1	So_2	So_3	So_4
Ta_1	R		Out			
Ta_2	RU	C	$InOut$			
Ta_3		R			Out	
Ta_5		R		Out		
Ta_6	CU					Out

- $Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_6U$
- $Ta_2/Ds_2C \xrightarrow[Ds_2]{} Ta_3R$
- $Ta_2/Ds_2C \xrightarrow[Ds_2]{} Ta_5R$

これらの変更タスクと波及タスクの操作の順序組合せがテストケースとなる．抽出した順序組合せが持つ入力の条件や出力の特性を仕様から抜き出して論理的テストケースとしてまとめる．表 10 に論理的テストケースとしてまとめた結果を示す．

表 10: 順序組合せテストによる論理的テストケース

新規フライト予約

No	論理的テストケース	順序組合せ
1	フライト予約後の空き情報問合せによる同一フライトの参照	$Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_1R$
2	フライト予約後の再度同一フライトの予約	$Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_2/Ds_1U$
3	フライト予約後の同期処理によって最新のチケット残数の計算	$Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_6U$
4	既存注文開く画面での予約したフライトの参照	$Ta_2/Ds_2C \xrightarrow[Ds_2]{} Ta_3R$
5	注文件数グラフ・注文履歴の一覧への新規予約フライト予約の反映	$Ta_2/Ds_2C \xrightarrow[Ds_2]{} Ta_5R$

表 11: 新規フライト予約の画面遷移表（一部分）

遷移→ 前状態↓	フライト検索ボタン	キャンセル Or	OK	注文を開く	グラフ	新規予約	Or 終了	履歴	XOr 終了
フライト予約画面	[出発日・From・To 入力済] フライトテーブル			注文を開く画面	グラフ	[フライト便・クラス・人数・搭乗者・入力済] フライト予約画面		履歴	
フライトテーブル		フライト予約画面	[フライト選択済] フライト予約画面						
注文を開く画面			[予約番号] フライト予約画面						
グラフ							フライト予約画面		
履歴									フライト予約画面
ログイン			フライト予約画面						

5 考察

次に，提案手法で抽出したタスク間の順序組合せと，*AS* のテストケースを設計する既出の手法である状態遷移テストで抽出されるテストケースの比較を行う. 状態遷移テストのテストベースとなるフライト予約システムの画面遷移表（表 11）を使って, 順序組合せの確認が可能な網羅基準である S1 網羅基準を適用する．表 11 は，4 節の提案手法適用例と適用範囲を合わせるために，新規フライト予約を行うために必要な画面と隣接する画面遷移に該当する表としている．仕様の詳細度合いは，DFD，ER 図，CRUD 図と画面遷移表では同等にしている．それは，画面遷移のイベントでのガード条件に記載したデータが DFD のエッジに記載したデータ，ER 図のエンティティの属性と一致していることから確認できる．S1 網羅基準を適用すると 28 の状態遷移パスとなる．28 の状態遷移パスのうち，対応する提案手法で抽出した順序組合せは，表 10 で示すテストケース No.1，2，4，5 の 4 つであった．これらは, 画面遷移表のフライト予約状態での登録イベントを起点にするもののみであった．順序組合せに該当しない状態遷移パスは，互いのタスクで同一のデータを介して処理をするといったことがない．例えば，フライト検索をした後にキャンセルをするとフライト予約画面に遷移するパスは，前の処理の結果によって影響を及ぼさない．

S1 網羅基準では抽出できないが，本手法によって抽出できたテストケースは，No.3 の $Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_6U$ である．このテストケースは，必要なテストケースと考えられる．4 節の適用評価にて利用したテストベースは，変更が入った機能セットに焦点を絞ったものである．4 節では新規フライト予約が該当する。そのため，表 11 では，新規フライト予約に隣接する画面遷移を該当するテストベースとしている．表 10 のテストケース No3 における波及タスクである Ta_6U は，表 7 から同期処理のタスクであることがわかるが，新規フライト予約とは別の機能セットに含まれるタスクであり，フライト予約画面と隣接する画面からなる画面遷移表には現れない．そのため，S1 網羅基準では抽出することができない．

6 おわりに

本論文では，状態遷移を持つソフトウェアにおいて，変更による変更波及がデータベースや外部変数などの保持データを介して生ずる場合のテストに関して，その網羅基準である IDAU 法と, 順序組合せテストケースを抽出する手法を提案した．DFD，ER 図、CRUD 図をテストベースとして，3 つのルールを適用することでテストが必要となる順序組合せを抽出できることを説明した．提案した手法で組合せが抽出できることを確認するため，フライト予約システムの仕様を具体例にして適用を行った．最後に画面遷移図（画面遷移表）から従来手法である状態遷移テストの S1 網羅基準にて抽出した状態遷移パスと提案手法を比較して，テストケース数の削減ができる効果と，S1 レベルの画面遷移の網羅では抽出できないテストケースが抽出できる効果を示した．

提案手法に対する今後の取り組みは 2 つある．1 つは,

適用範囲の明確化である. 変更のパターン（タスク内の制御ロジックの変更, 新しい要素の追加など）に対して, どこまで適用でき, どこからは適用できないかを明らかにする.

もう 1 つは，今回の提案手法のツール化である．実際の開発プロジェクトで扱う規模の大きいデータ設計文書に対して本手法を適用する際には，本手法のルールをツール化するといった方法での適用が必要になる．これらの準備を行い，実践の場に本手法を適用していく．

参考文献

[1] R.S. Arnold, Software change impact analysis, IEEE Computer Society Press, 1996.

[2] B. Li, X. Sun, H. Leung, and S. Zhang, “A survey of code-based change impact analysis techniques,” Software Testing, Verification and Reliability, vol.23, no.8, pp.613–646, 2013.

[3] B. Beizer, Software Testing Techniques, Van Nostrand Reinhold, 1990. (翻訳 :小野間 彰， 山浦恒央 :ソフトウェアテスト技法, 日経 BP 出版センター (1994)).

[4] H. Gomaa, “Designing software product lines with uml.,” SEW Tutorial Notes, pp.160–216, 2005.

[5] L.C. Briand, Y. Labiche, and L. O’sullivan, “Impact analysis and change management of UML models,” Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference onIEEE, pp.256–265 2003.

[6] 小谷正行，落水浩一郎，“UML 記述の変更波及解析に利用可能な依存関係の自動生成,” 情報処理学会論文誌，vol.49，no.7，pp.2265–2291，2008.

[7] J.N. Campbell, “Data-flow analysis of software change,” Oregon Health & Science University, pp.1–118, 1990.

[8] A. Maule, W. Emmerich, and D.S. Rosenblum, “Impact analysis of database schema changes,” Proceedings of the 30th international conference on Software engineeringACM, pp.451–460 2008.

[9] 加藤正恭，小川秀人，“CRUD マトリクスを用いたソフトウェア設計影響分析手法,” 情報処理学会第 73 回全国大会講演論文集，vol.73，pp.249–250，2011.

[10] ISTQB/FLWG, Foundation Level Syllabus, International Software Testing Qualifications Board, 2011.

[11] T. DeMarco, “Structure analysis and system specification,” Pioneers and Their Contributions to Software Engineering, pp.255–288, Springer, 1979.

[12] A.L. Politano, “Salvaging information engineering techniques in the data warehouse environment,” Informing Science, vol.4, no.2, pp.35–44, 2001.

[13] T.Yumoto， K.Uetsuki， T.Matsuodani, and K.Tsuda, “A study on the efficiency of a test analysis method utilizing test-categories based on aut and fault knowledge,” ICACTCM ’ 2014, pp.70–75, 2014.

[14] ISO/SC7/WG26, Software and Systems Engineering-Software-Testing Part 2:Test Processes, ISO/IEC/IEEE29119 2013(E), 2013.

探索的テストにおける不具合発見率向上に向けた取り組み

中野 直樹
株式会社 LIFULL
NakanoNaoki@LIFULL.com

要旨

探索的テスト（Exploratory Software Testing）は近年様々なソフトウェア開発の現場で用いられている．弊社の開発プロジェクトにて探索的テストの導入を行い，そこで発見された課題を元に不具合発見率向上に向けた取り組みを行った．

1. はじめに

弊社ではウェブサービスの開発，運営をしており，社内開発，外部委託あわせて年間数百の大小様々な要件のプロジェクトが稼働している．その中で品質保証部門では，第三者検証なども行っており，様々な開発プロセスが実践されている中で，特にアジャイル開発プロジェクトにおいて有効とされる探索的テスト[1]に着目し，導入を行った．

2. 解決したい問題

2.1. 探索的テストの多様性

探索的テストを導入するうえで様々な考え方やスタイル[3][4][5]があり，どのようなスタイルを選択するかで探索的テスト実施時の検証内容も変わってくることがわかった．

2.2. 不具合発見率と属人化

探索的テストの問題点としてテスト実行時の不具合発見率に個人差があり，また発見できる不具合の重要度にも偏りがあることがわかった．

3. 工夫した点

3.1. チャーターエレメントの作成

探索的テスト実施者間の不具合発見数の差が小さくなるよう重要なテストのヒントを探索的テストで用いるチャーター[3]作成時のインプットの一部として用いることとした．ここで使用するチャーターのインプットをチャーターエレメントと名付け，プロジェクトで発見した新しい不具合の知見やソフトウェア起因で本番障害につながった問題などを取り込み，過去に発生した問題を元にチャーターを作成できるように情報をまとめ，再利用できるようにした．

3.2. チャーターエレメントの拡張と情報の粒度

探索的テストに用いるチャーターエレメントは，定期的に新しい不具合や障害の情報の追加を行う．その際，最適化する目的で MECE かつ使用しやすい構成に保つことで，過剰な肥大化や，情報の陳腐化の防止を行えるようにした．

4. 今後の課題

チャーターエレメントをデータベース化することを検討しているが，その為には効率の良い情報抽出の仕組みが必要となる．現在はテストデバイス（PC やスマートフォンのウェブブラウザやスマホアプリ）などの種類別に管理しているが，将来的にはタグなどのメタデータを付与することで必要な条件をもとに検索を行えるよう検討を進める予定である

参考文献

[1] 高橋寿一；探索的テストってなんですか？アジャイル時代のソフトウェアテスト http://jasst.jp/symposium/jasst14kyushu/pdf/S3.pdf
[2] Janet Gregory, Lisa Crispin 実践アジャイルテスト テスターとアジャイルチームのための実践ガイド（IT Architects' Archive ソフトウェア開発の実践），2009
[3] Elisabeth Hendrickson Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing ,2013

[4] Cem Kaner, J.D., Ph.D.: A Tutorial in Exploratory Testing
http://www.kaner.com/pdfs/QAIExploring.pdf
[5] James A. Whittaker; Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, 2009

VDM++仕様を対象にしたテストケース自動生成ツールBWDMにおけるif式の構造認識に基づいたテストケース生成手法の提案

立山 博基
宮崎大学大学院工学研究科
tachiyama@earth.cs.miyazaki-u.ac.jp

片山 徹郎
宮崎大学工学教育研究部
kat@cs.miyazaki-u.ac.jp

要旨

ソフトウェア開発において、形式手法を用いることで、仕様に曖昧さが含まれることを防ぐことができる。我々は、形式手法 *VDM* を用いたソフトウェア開発におけるテスト工程の効率化を目的として、テストケース自動生成ツール *BWDM* を試作した。既存の *BWDM* の問題点として、*VDM* 仕様中の *if* 式からテストケース生成を行う際に、ネストや *else if* などの *if* 式同士の構造を認識していない点が挙げられる。そのため、そのような構造の中にある戻り値に対するテストケース生成を行えない。本稿では、この *BWDM* における、テストケース生成処理の問題点を解決するために、*if* 式の構造認識に基づいたテストケース生成手法の提案を行う。提案手法を *VDM* 仕様に適用した結果、従来の *BWDM* ではテストケース生成を行えなかった仕様中の戻り値に対する、テストケースの生成が可能になることを確認した。そのため、*BWDM* に提案手法を実装することで、テストケース生成処理の問題点を解決し、*BWDM* の有用性が向上すると言える。

1. はじめに

ソフトウェアは年々大規模化かつ高機能化を続け、その結果、ソフトウェア上のバグが社会にもたらす影響も近年では甚大なものとなっている [1]。

ソフトウェアにバグが混入する原因の 1 つとして、上流工程のソフトウェア設計段階において、自然言語を一般的に用いていることが挙げられる。自然言語は元来、曖昧さを含んでいる [2]。そのため、プログラマが、仕様書上の表記を、仕様の作成者が本来意図していない意味で捉えてしまうことが起こる。実装者が、仕様書の本来の意図から外れた認識に基づいて実装を行った結果、ソフトウェアにバグが混入されてしまう。

この問題を解決するための 1 つの手段として、形式手法 (Formal Methods)[3] が提案されている。形式手法を用いた開発では、まず、数理論理学を基盤とした形式仕様記述言語 (Formal Specification Language) により、開発対象が持つ特性を仕様として記述する。数理論理学を基にしているため、自然言語を用いた開発と異なり、定理証明や機械的な検査を用いて、記述した内容が正しいことを数学的に証明することが可能である。つまり、自然言語の持つ曖昧さを排除した、厳密な仕様を作成することが可能となる。

一方で、自然言語もしくは形式手法を用いた設計のいずれにしても、実装を行った後は、作成したソフトウェアに対してテストを行う必要がある。テストを行うためには、テストケースの設計作業が必要であるが、人手によるテストケースの設計には手間と時間がかかる。そのため、テストケースの設計作業を効率よく行うことで、テスト工程を効率化できる。また、バグが潜みうる箇所に的を絞ったテストケースの設計を行うことも、テスト実施の効率化のために重要である。

このような問題とソフトウェアテストの必要性を踏まえ、形式仕様を基にしたテストケース自動生成ツール BWDM の試作を行った [4, 5, 6]。BWDM は、形式仕様記述言語 VDM++で記述した仕様とデシジョンテーブルを入力とし、VDM++仕様を基に境界値分析を行い、テストケースを自動生成する。デシジョンテーブルとは、ソフトウェアの入力に対する出力を表形式でまとめたものである [7]。ソフトウェアに対してテストを行う際に、

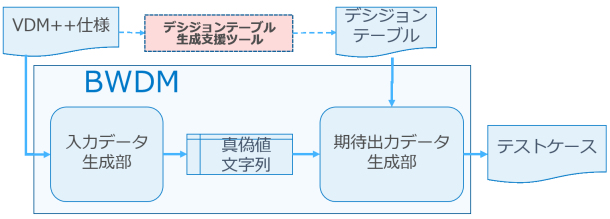


図 1. BWDM の構成

ソフトウェアの期待出力を確認する際に有用である。

既存の BWDM の問題点として、VDM++仕様中の if 式からテストケース生成を行う際に、ネストや else if などの if 式同士の構造を認識していない点が挙げられる。そのため、そのような構造の中にある戻り値に対するテストケース生成を行えない。

そこで本稿では、この問題の解決を目的として、if 式の構造認識に基づいたテストケース生成手法を提案する。具体的には、if 式の構造を認識し、条件式を生成し、それを基にテストケースを生成することによって、既存の問題点の解決を目指す。

2. テストケース自動生成ツール BWDM

本章では、既存のテストケース自動生成ツール BWDM の構成、機能、処理、入出力、及び、問題点について記す。BWDM とは、Boundary Value と Vienna Development Method の頭字語である。W は Value と Vienna の 2 つの V を意味する。

2.1. BWDM の構成・機能・処理

図 1 に、BWDM の構成を示す。BWDM は入力データ生成部と期待出力データ生成部によって構成しており、それぞれの役割は以下の通りである。

- 入力データ生成部
  - VDM++仕様読み込み
  - 構文解析
  - データ抽出
  - 境界値分析
  - 入力データ生成

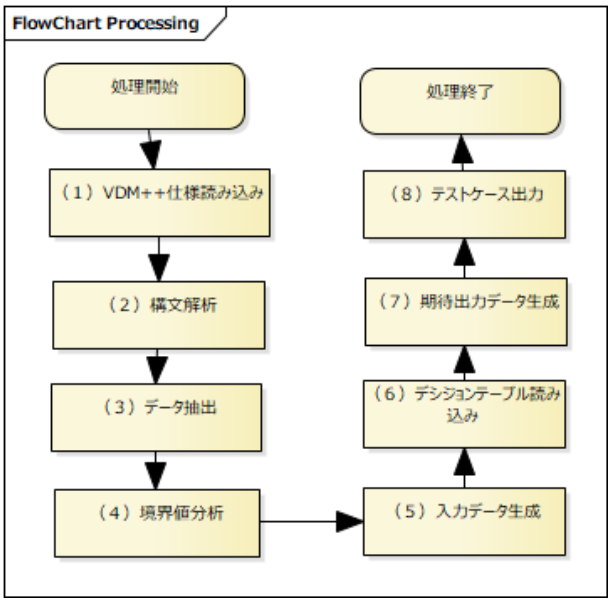


図 2. BWDM の処理の流れ

- 期待出力データ生成部
  - デシジョンテーブル読み込み
  - 期待出力データ導出
  - テストケース生成

BWDM が持つ機能は、以下の通りである。

- VDM++仕様の構文解析及び情報の抽出
- 抽出した情報を基にした境界値分析
- 境界値分析を行った結果を基にした入力データ生成
- デシジョンテーブルを基にした入力データに対応する期待出力データの導出
- テストケースの出力 (入力データ＋期待出力データ)

図 2 に、BWDM の処理のフローチャートを示す。処理は以下の流れで行われる。

1. 指定された VDM++仕様ファイルを読み込む
2. VDM++仕様ファイルを構文解析する
3. 関数定義中の引数型と if 式中の条件式に関する情報を抽出する
4. 抽出した情報に対して境界値分析を行う

```
1 class SampleClass
2
3 functions
4
5 sampleFunction : int*nat − > seq of char
6   sampleFunction(a, b) ==
7   if(a <= 10) then
8     if(-1 > a) then
9       “a <= 10 and -1 > a”
10    else if(b > 0) then
11      “a <= 10 and !(-1 > a) and b > 0”
12    else
13      “a <= 10 and !(-1 > a) and !(b > 0)”
14  else
15    if(b <= -3) then
16      “!(a <= 10) and b <= -3”
17    else if(a < 20) then
18      “!(a <= 10) and !(b <= -3) and a < 20”
19    else
20      “!(a <= 10) and !(b <= -3) and !(a < 20)”;
21
22 end SampleClass
```

図 3. 入力例：VDM++仕様

5. 境界値分析の結果から入力データを生成する
6. デシジョンテーブルを読み込む
7. 入力データとデシジョンテーブルから期待出力データを導出する
8. 入力データと期待出力データをテストケースとして併せて出力する

構文解析は、Nick Battle 氏の開発した VDM 静的解析ツール VDMJ を用いて実現している [8]。境界値分析は、抽出した引数型、if 式中の条件式のそれぞれに対して行い、入力データを生成する。

BWDM は期待出力の導出を、入力データとデシジョンテーブルの照らし合わせによって行う。この際に使用するデシジョンテーブルは、本研究室で開発したデシジョンテーブル生成支援ツール [9] によって生成したものである。最終的に、テストケースを csv(comma-separated values) 形式でファイルに書き出す。

引数の個数:2				
引数型:	第1引数:int	第2引数:nat		
テストケースNo.	入力データ		-->	期待出力データ
No.1	intMin−1	natMin−1	-->	Undefined Action
No.2	intMin−1	natMin	-->	Undefined Action
No.3	intMin−1	natMax	-->	Undefined Action
No.4	intMin−1	natMax+1	-->	Undefined Action
No.5	intMin−1	1	-->	Undefined Action
No.6	intMin−1	−3	-->	Undefined Action
No.7	intMin−1	−2	-->	Undefined Action
No.8	intMin	natMin−1	-->	Undefined Action
No.9	intMin	natMin	-->	aは10以下で、bは0以下です
No.10	intMin	natMax	-->	aは10以下で、bは0より大きいです
No.11	intMin	natMax+1	-->	Undefined Action
No.12	intMin	1	-->	aは10以下で、bは0より大きいです
No.13	intMin	−3	-->	aは10以下で、bは0以下です
No.14	intMin	−2	-->	aは10以下で、bは0以下です
No.15	intMax	natMin−1	-->	Undefined Action
No.16	intMax	natMin	-->	aは20以上で、bは−3より大きいです
No.17	intMax	natMax	-->	aは20以上で、bは−3より大きいです
No.18	intMax	natMax+1	-->	Undefined Action
No.19	intMax	1	-->	aは20以上で、bは−3より大きいです
No.20	intMax	−3	-->	aは10より大きく、bは−3以下です
No.21	intMax	−2	-->	aは20以上で、bは−3より大きいです
No.22	intMax+1	natMin−1	-->	Undefined Action
No.23	intMax+1	natMin	-->	Undefined Action
No.24	intMax+1	natMax	-->	Undefined Action
No.25	intMax+1	natMax+1	-->	Undefined Action
No.26	intMax+1	1	-->	Undefined Action
No.27	intMax+1	−3	-->	Undefined Action
No.28	intMax+1	−2	-->	Undefined Action
No.29	10	natMin−1	-->	Undefined Action
No.30	10	natMin	-->	aは10以下で、bは0以下です
No.31	10	natMax	-->	aは10以下で、bは0より大きいです
No.32	10	natMax+1	-->	Undefined Action
No.33	10	1	-->	aは10以下で、bは0より大きいです
No.34	10	−3	-->	aは10以下で、bは0以下です
No.35	10	−2	-->	aは10以下で、bは0以下です
No.36	11	natMin−1	-->	Undefined Action
No.37	11	natMin	-->	aは10より大きく20未満で、bは−3より大きいです
No.38	11	natMax	-->	aは10より大きく20未満で、bは−3より大きいです
No.39	11	natMax+1	-->	Undefined Action
No.40	11	1	-->	aは10より大きく20未満で、bは−3より大きいです
No.41	11	−3	-->	aは10より大きく、bは−3以下です
No.42	11	−2	-->	aは10より大きく20未満で、bは−3より大きいです
No.43	19	natMin−1	-->	Undefined Action
No.44	19	natMin	-->	aは10より大きく20未満で、bは−3より大きいです
No.45	19	natMax	-->	aは10より大きく20未満で、bは−3より大きいです
No.46	19	natMax+1	-->	Undefined Action
No.47	19	1	-->	aは10より大きく20未満で、bは−3より大きいです
No.48	19	−3	-->	aは10より大きく、bは−3以下です
No.49	19	−2	-->	aは10より大きく20未満で、bは−3より大きいです
No.50	20	natMin−1	-->	Undefined Action
No.51	20	natMin	-->	aは20以上で、bは−3より大きいです
No.52	20	natMax	-->	aは20以上で、bは−3より大きいです
No.53	20	natMax+1	-->	Undefined Action
No.54	20	1	-->	aは20以上で、bは−3より大きいです
No.55	20	−3	-->	aは10より大きく、bは−3以下です
No.56	20	−2	-->	aは20以上で、bは−3より大きいです

図 4. 出力例：テストケース

BWDM は現状、整数値の入力データ生成のみの対応であるため、以降の説明で登場する変数の型は全て整数型であるとする。また、テストケース生成処理は、関数定義中の引数型と if 式のみを対象に行う。

2.2. BWDM の入出力

BWDM の入出力に関して、VDM++仕様ファイルを図 3 に、テストケースファイルを図 4 に、それぞれ示す。

入出力に関係するファイルは以下である。

- 入力
  - VDM++仕様ファイル (テキスト形式)

```
1 class ProblemClass
2
3 functions
4
5 problemFunction : nat -> seq of char
6   problemFunction(a) ==
7     if(a mod 4 = 0) then
8       if(a > 92) then
9         “a mod 4 = 0 and a > 92”
10       else
11         “a mod 4 = 0 and !(a > 92)”
12     else
13       “!(a mod 4 = 0)”;
14
15 end ProblemClass
```

図 5. 現状の課題：ネストした if 式

  - デシジョンテーブルファイル (csv 形式)

- 出力
  - テストケースファイル (csv 形式)

図 4 の入力データ中に存在する $intMin, intMin - 1, intMax, intMax + 1$ などは、図 3 中の関数定義における引数型 (int) に対して境界値分析を行い、生成した入力データである。それぞれ、「引数型 int の最小値 (Minimum)」、「引数型 int 型の最小値より 1 小さい値」、「引数型 int の最大値 (Maximum)」「引数型 int の最大値より 1 大きい値 (Maximum)」を表す。引数型の範囲外の値 (*Min-1, *Max+1) については、桁溢れにより、プログラムの動作が予測不可能であるため、期待出力データは Undefined Action としている。

2.3. BWDM の問題点

既存の BWDM は、構文解析により VDM++仕様中の if 式の抽出を行っている。しかし、ネスト構造や else if などの構造は無視している。それらの構造を持つ if 式が仕様中に存在する場合も、個々の if 式として抽出し、それら 1 つ 1 つに対して境界値分析を行い、入力データを生成している。そのため、複数の if 式が関わりあった構造の中にある戻り値に対する入力データの生成は行えない。

引数の個数:1			
引数型:	第1引数:nat		
テストケースNo. 入力データ --> 期待出力データ			
No.1	natMin-1	-->	Undefined Action
No.2	natMin	-->	a mod 4 = 0 and !(a > 92)
No.3	natMax	-->	!(a mod 4 = 0)
No.4	natMax+1	-->	Undefined Action
No.5	4	-->	a mod 4 = 0 and !(a > 92)
No.6	3	-->	!(a mod 4 = 0)
No.7	5	-->	!(a mod 4 = 0)
No.8	93	-->	!(a mod 4 = 0)
No.9	92	-->	a mod 4 = 0 and !(a > 92)

図 6. 図 5 から BWDM で生成したテストケース

図 5 と図 6 に、現状、問題となる if 式、及び、その if 式から生成したテストケースを示す。図 5 には、a mod 4 = 0 と $a > 92$ の整数値 a に関する 2 つの if 式がネストしており、戻り値は文字列である。文字列中には、その戻り値を返す際に a が満たすべき条件式を表している。

図 6 の期待出力データ中に戻り値 “a mod 4 = 0 and a > 92” に対するテストケースが存在しないことが確認できる。BWDM は境界値分析による入力データ生成処理により、1 つ目の a mod 4 = 0 からは 3, 4, 5、2 つ目の $a > 92$ からは 92, 93 を入力データとして生成するが、それらの中には、“a mod 4 = 0 and a > 92” を出力する入力データは存在していない。戻り値 “a mod 4 = 0 and a > 92” を出力するためには、a mod 4 = 0 and $a > 92$ を満たす入力データを生成する必要がある。

このような if 式による構造関係を認識し、その先の戻り値に対する入力データを生成できるように、BWDM を改良する必要がある。

3. if 式の構造認識に基づいたテストケース生成手法について

本章では、2.3 節で示した問題点を改良するための手法、及び、BWDM 上でその処理を行うための改良方法について述べる。

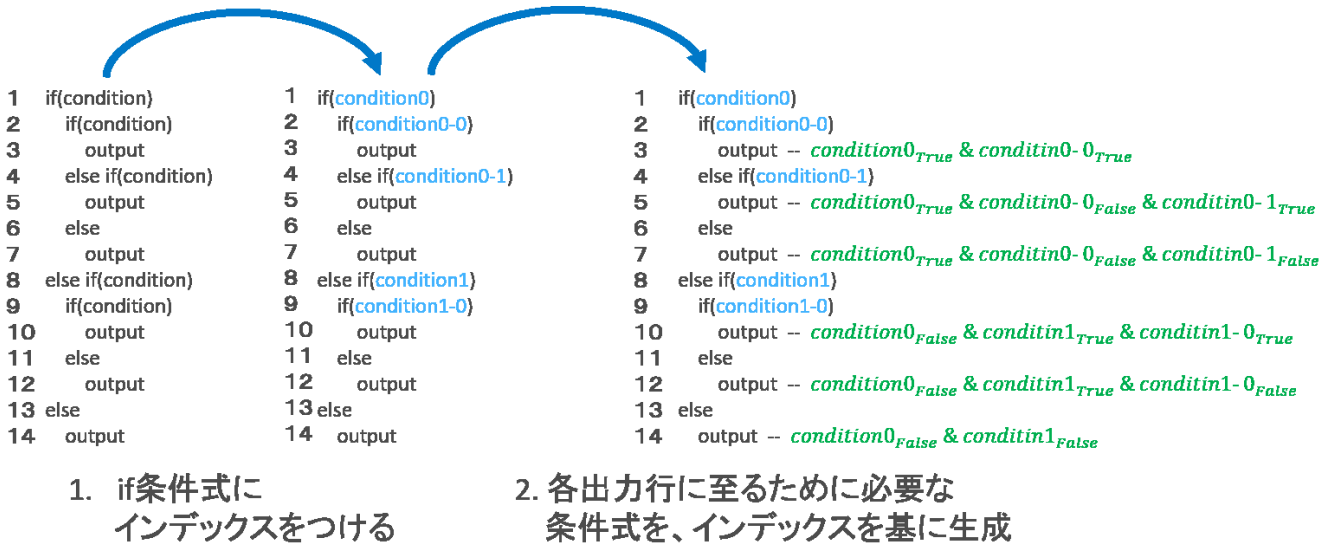


図 7. if 式へのインデックス付け及び満たすべき条件式の生成

表 1. 図 5 における戻り値及びその際満たすべき条件式と真偽値

戻り値	条件式
“a mod 4 = 0 and a > 92”	a mod 4 = 0, a > 92
“a mod 4 = 0 and !(a > 92)”	a mod 4 = 0, a <= 92
“!(a mod 4 = 0)”	a mod 4 != 0

3.1. 提案手法

if 式の各出力行へ至るためには、各出力行へ至る条件式を満たす入力データを生成すればよい。表 1 に、2.3 節で用いた図 5 において、各出力行へ至るために満たすべき条件式を示す。

1 つ目と 2 つ目の出力においては、if 条件式がネストしているため、満たすべき if 条件式は and で繋がったものとなる。また、else 節の後にある出力は、記述してある if 条件式を否定した条件式を満たす必要がある。

次節から、提案手法を実現するための具体的な処理の流れを述べる。

3.2. if 式の構造の認識及び満たすべき if 条件式の生成

提案手法の例を、図 7 に示す。この例を基に、提案手法の説明を行う。手順は、if 式に対するインデックス付けと条件式の生成の 2 つである。

1. if 式へのインデックス付け (図 7 中の手順 1.)

   if 式の構造を認識するために、構文解析を行った後に、各 if 式へインデックスを付ける。インデックスにより、ネストや else if などの構造と、各戻り値に対する条件式を確認できるようにする。図 7 中には、if 条件式が 5 つ存在している。ネスト構造の中に存在している条件式 (図 7 中の $condition0-0, condition0-1, condition1-0$) は、親の条件式 (図 7 中の $condition0, condition1$) のインデックスの数字の後に、更にインデックスとして数字を割り振る。

2. 条件式の生成 (図 7 中の手順 2.)

   各出力行へ至る入力データについて、if 条件式のインデックスを辿りながら、入力データが満たすべき条件式を生成する。インデックスを辿る方法は複数の手法が考えられるが、今回は各戻り値から条件式を辿る。各戻り値ごとに、if 式のインデックスを利用して、その戻り値に至るために満たすべき条件式を導出する。

引数の個数:1			
引数型:	第1引数:nat		
テストケースNo. 入力データ --> 期待出力データ			
No.1	3424	-->	"a mod 4 = 0 and a > 92"
No.2	52	-->	"a mod 4 = 0 and !(a > 92)"
No.3	1217	-->	"!(a mod 4 = 0)

図 8. 図 5 に提案手法を適用して生成したテストケース

5 行目の output の出力に必要な条件式を具体例とすると、まず、4 行目の $condition0-1$ を満たす必要がある。更に、$condition0-1$ は else if の条件式であるため、2 行目の $condition0-0$ は否定する必要がある。そして、1 行目と 2 行目で if 式がネストしているため、1 行目の $condition0$ は満たす必要がある。

そのため、必要となる条件式は、$condition0-1_{True}$ かつ $condition0-0_{False}$かつ $condition0_{True}$ となる。

なお、ここでの $condition*_{True}, condition*_{False}$ とは、$a > 92$ という if 条件式を例にすると、$condition_{True}$ が $a > 92$、$condition_{False}$ は、$\overline{condition_{True}}$ の $a <= 92$ となる。

3.3. 生成した条件式を満たす入力データの生成

満たすべき条件式を生成した後に、その条件式を満たす整数値を生成する。乱数により整数値を生成し、整数値が条件式を満たせば、入力データとして採用する。

条件式を満たす整数が存在しない場合、入力データの生成は不可能である。そのため、入力データ生成に時間制限を設け、一定時間が経過した場合は入力データ生成処理を終了する。

3.4. 提案手法によるテストケース

図 8 に、2.3 節の図 5 に対して、提案手法を適用することで生成できるテストケースを示す。提案手法により新たに生成できるテストケース中に、期待出力データとして “a mod 4 = 0 and a > 92” を出力しているテストケースを確認できる。

よって、提案手法による処理を今後 BWDM に実装することで、if 式構造の中にある戻り値に対するテストケースを生成できる。

4. 考察

4.1. 提案手法の有用性について

本稿では、VDM++仕様を基にしたテストケース自動生成ツール BWDM において、if 式同士の構造の中にある戻り値に対するテストケース生成を行えない問題を解決するために、if 式の構造認識に基づいたテストケース生成手法を提案した。

提案手法は、まず、VDM++仕様を構文解析した後に、関数定義中に登場する if 式にインデックス付けを行った。この際、ネストの中にある if 式については、その親のインデックスに、更にインデックスを加える形とすることで、ネストした if 式の構造を表現した。そして、付与したインデックスを基に、各出力行へ至るための条件式を生成した。最後に、生成した条件式を満たす入力データを乱数によって生成した。

今回提案した手法により、if 式同士の構造の中にある戻り値に対するテストケースを生成できることを確認できた。よって、既存の BWDM のテストケース生成処理に、今回提案したテストケース生成処理を追加で実装することで、if 式同士の構造の中にある戻り値に対するテストケース生成を行えない問題を解決できる。

4.2. 関連研究

記号実行 (Symbolic Execution)[10] を用いることで、今回の提案手法同様に、if 式が入れ子になった場合も全ての戻り値に対してテストケース生成を行った事例として、宮本らの研究 [11] と、高松らの研究 [12] がある。記号実行は、各戻り値に対する制約条件 (条件式) を導出する。

ソースコード中の表明 (Assersion) の動的な自動生成手法は広く研究されているが、その問題点として、最終的に生成する表明は、手法中で対象プログラムの実行データを取得する際に用いられるテストケースに依存するということが知られている。宮本らはこのテストケー

ス依存問題を改善するために、記号実行などの手法を用いたテストケース生成手法を提案した。記号実行を用いて、対象ソースコード内の全ての戻り値に対して、その制約条件 (条件式) を導出し、その条件を満たす引数を生成している。

また、オブジェクト指向プログラミングにおけるソフトウェアテストを行うには、テスト対象のインスタンス生成・状態の変更などの前準備のメソッド実行を、テストケース内で行う必要がある。そのようなテストケース自動生成ツールの 1 つに Seeker[13] があり、高松らは Seeker の機能拡張を行った。Seeker では、記号実行と具体的な値を組み合わせた動的記号実行 (Dynamic Symbolic Execution) を用いて、ソースコード中の条件式を基に、全ての戻り値に対する引数の組を生成している。

本研究で提案する、if 式の構造認識に基づいたテストケース生成手法は、インデックスを用いて戻り値から条件式を導出するため、記号実行と比較して、探索空間を削減できる。そのため、計算効率の面で、記号実行に比べて優位性が有ると言える。

また、VDM 仕様を基にしたテストケースの自動生成法に関する研究の 1 つに、馬場らの研究がある [14]。馬場らの研究では、VDM 仕様記述からコーナーケースに対するテストを行うことを目的として、テストドライバの雛形の作成を行うツールを開発した。馬場らのツールは、形式手法を用いたソフトウェア開発におけるプロセス中の、単体テストにおけるテストを支援する。

馬場らのツールは、現状、VDM-SL 仕様内の操作定義をテストケース作成の対象としている。そのため、VDM++を対象に、関数定義からテストケースを作成する BWDM とは異なる。また、馬場らのツールから出力されるテストケースは、入力データとして記述内の条件式を表示するものである。これに対して、BWDM は条件式から生成した、具体的な数値の入力データを確認することが可能である。そのため、BWDM の出力を使ってそのままテストを実行できることが、BWDM の利点であるといえる。

また，VDM++仕様記述をテストするためのテストケース自動生成ツールとして，TOBIAS[15] がある．TOBIAS は，TOBIAS に入力されたテストパターンに従い，テストケースを自動生成する．テストパターンとは，テスト設計者が正規表現を用いて，出力するテストケースを定義したものである．

TOBIAS と BWDM は，どちらも VDM++を用いたソフトウェア開発のテスト段階を支援する．TOBIAS は，VDM++で記述された仕様そのもののテスト支援を行う．これに対して，BWDM は VDM++で記述された仕様から実際に実装を行ったソフトウェアへのテストを支援する点で，異なっている．

5. おわりに

本稿では、VDM++を基にしたテストケース自動生成ツール BWDM における、if 式同士の構造の中にある戻り値に対するテストケース生成を行えない問題を解決するために、if 式の構造認識に基づくテストケース生成手法を提案した。

提案手法ではまず、VDM++仕様を構文解析した後に、if 条件式にインデックスを付与し、そのインデックスを基に、各出力行へ至るための条件式を生成し最終的に、それらの条件式を満たす入力データを乱数によって生成した。

提案手法により、既存の BWDM では生成できなかった、if 式による構造の中にある戻り値に対するテストケースの生成が可能なことを確認した。故に、既存の BWDM のテストケース生成処理に、今回提案したテストケース生成処理を追加で実装することで、if 式同士の構造の中にある戻り値に対するテストケース生成を行えない問題を解決できる。

以上のことから、今回の提案手法に基づいて BWDM へ入力データ生成処理を実装することで、既存の BWDM の問題点を解決し、BWDM の更なる利便性の向上に繋がると言える。

以下に、BWDM の今後の課題を示す。

- デッドコードへの対応

  戻り値に対する条件式を満たす整数が存在しない場合、戻り値はデッドコードである。この場合、入力データ生成は行えないため、本研究では入力データ生成処理を一定時間で終了する。充足不能な条件式に対しては、入力データ生成処理を行わず、ユーザーにデッドコードを指摘する機能が必要である。

- 未対応の定義部や構文、演算子への対応

  操作定義、型定義などの定義部へは現在未対応である。また、事前条件や事後条件、let 式など

の VDM++に存在する多くの構文にも、既存の BWDM は未対応である。これらの構文からテストケースを生成する手法を考案し、BWDM に実装することで、BWDM の有用性が更に向上すると考えている。

- 整数型以外への対応

  既存の BWDM は、整数型以外の、実数型や合成型には未対応である。構文解析の際に、整数型以外の型の情報を抽出し、境界値分析及び入力データ生成処理を実装することで、これらのテストケースの生成が可能になると考えている。

- 両辺が変数である if 条件式への対応

  if 条件式中の両辺が変数である場合、既存の BWDM は境界値分析を行うことができないため、入力データ生成を行えない。このような if 条件式から境界値を抽出する処理を BWDM に実装することで、BWDM の適用範囲の更なる拡大を見込める。

参考文献

[1] 失敗事例データベース, 失敗事例 – みずほファイナンシャルグループ大規模システム障害, http://condezine.jp/article/detail/2364, 2017/3/8 アクセス

[2] IPA 独立行政法人 情報処理推進機構, なぜ形式手法か, http://sec.ipa.go.jp/users/seminar/seminar_tokyo_20150912-02.pdf, 2017/3/8 アクセス

[3] 荒木啓二郎, 張漢明, プログラム仕様記述論, オーム社, 2002

[4] 立山博基, 片山徹郎, VDM++仕様を基にした境界値テストケース自動生成ツール BWDM の試作について, 平成 28 年度 電気・情報関係学会 九州支部連合大会, p.90, 2016

[5] 立山博基, 片山徹郎, VDM++仕様に対する境界値分析を用いたテストケース自動生成, JaSST'16 九州, p.32, 2016

[6] Hiroki Tachiyama, Tetsuro Katayama, Yoshihiro Kita, Hisaaki Yamaba, and Naonobu Okazaki, *Prototype of Test Cases Automatic Generation Tool BWDM Based on Boundary Value Analysis with VDM++*, The 2017 International Conference on Artificial Life and Robotics(ICAROB 2017), pp.275-278, 2017

[7] SQuBOK 策定部会, ソフトウェア品質知識体系ガイド 第 2 版, オーム社, 2014

[8] Nick Battle GitHub - nickbattle/vdmj, https://github.com/nickbattle/vdmj, 2017/3/8 アクセス

[9] 西川拳太, 片山徹郎, VDM++を用いたデシジョンテーブル生成支援ツールについて, 平成 25 年度 電気関係学会 九州支部連合大会, p.381, 2013

[10] James C. King, *Symbolic Execution and Program Testing*, Communications of The ACM Vol.19, No.7, pp.385-394, DOI: 10.1145/360248.360252 1976

[11] 宮本敬三, 堀直哉, 岡野浩三, 楠本真二, Daikon 生成表明改善のためのテストケース自動生成手法とその評価実験, コンピュータソフトウェア, Vol.28, No.4, pp.306-317, https://www.jstage.jst.go.jp/article/jssst/28/4/28_4_4_306/_article/-char/ja/ 2010

[12] 高松宏樹, 佐藤晴彦, 小山聡, 栗原正仁, 動的記号実行によるメソッドの複雑度を考慮したテストケース自動生成, 情報処理学会研究報告 Vol.2014-SE-185, No.27, NAID: 110009803966 2014

[13] Suresh Thummalapenta, Tao Xie, Nikolai Tillmann, Jonathan de Halleux, Zhendong Su, *Synthesizing Method Sequences for High-Coverage Testing*, SIGPLAN Notices Vol.46, No.10, pp.189-206, DOI: 10.1145/2076021.2048083 2011

[14] 馬場勇輔, 荒木啓二郎, 日下部茂, 大森洋一, 形式仕様記述のプロパティベーステストへの活用, 情報処理学会 火の国シンポジウム予稿集, https://www.ipsj-kyushu.jp/page/ronbun/hinokuni/1005/5C/5C-1.pdf, 2017/3/11 アクセス

[15] Olivier Maury, Yves Ledru, Pierre Bontron and Lydie du Bousquet, *Using TOBIAS*

for the automatic geneeration of VDM test cases, Laboratoire Logiciels, Systemes Reseaux – IMAG, http://vasco.imag.fr/Tobias/Papers/vdm02tobias.pdf, 2017/5/10 アクセス

段階的なシステムテストによる IoT システム開発効率化

西村　治
パナソニック株式会社
nishimura.osm@jp.panasonic.com

要旨

様々な商品が連携し複雑化している当社の IoT システム開発において，従来のシステムテストでは，テスト終盤で不具合が収束せず，不具合対応に計画以上の時間を要し，開発納期に影響するという課題が発生している．

これを解決するために，段階的なシステムテストを導入し，テスト終盤の不具合を大幅に削減できた．この取り組みを本稿で紹介する．

1. はじめに

ネット家電やホームオートメーションなどから始まった家電のシステム化は，昨今 IoT システムと呼ばれるようになった[1][2]．この IoT システムは，構成する商品が多様化し，社内の異なる組織で開発された商品や他社商品を含む構成が一般的となっている．これら多様な組織で開発された商品は，品質が一定でないことに加え，不具合への対応も一律の手順では円滑に進まないケースが多くなってきた．

このため，従来のシステムテストでは非効率な面が顕在化し，新たなシステムテストの仕組みが必要となってきた．この新たな仕組みとして，段階的なシステムテストを導入することした．この導入時の課題とその解決策を事例を踏まえて紹介する．

2. IoT システム開発の変化と課題

2.1. 背景

当社では，独立した複数の商品の組み合わせで価値を提供する商品を「システム商品」と呼んでいるが，従来のシステム商品開発は，システム商品を構成する全ての商品を単一の組織で開発し，その後別の組織が第三者としてシステムテストを実施するという体制をとるケースが多かった．

これに対して，IoT システムのように多様な複数組織が連携して行うシステム商品開発では，従来のブラックボックステスト[3]を行うシステムテストでは非効率な面が顕在化し，新たなシステムテストの仕組みが求められていた．

2.2. IoT システムの例

IoT システムの例として，当社のシステム商品である「スマート HEMS」を紹介する．スマート HEMS は，図 1 に示す通り，コントローラである AiSEG2 を中心に，社内の別組織で開発されたエアコンやレンジフード，他社商品の電動窓シャッターといった多様な商品が無線あるいは有線の通信で接続されており，センサー情報に基づいたエネルギー消費の見える化や，省エネのための家電の最適制御を実現している．また，AiSEG2 はインターネット経由でサーバと接続されており，スマートフォン向けの HEMS サービスを提供している[4]．

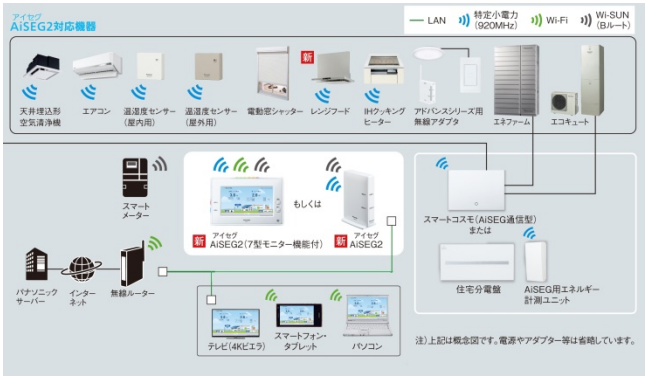


図 1 スマート HEMS システム構成図([1])

2.3. システムテストにおける課題

IoT システムにおける従来のシステムテストでは，不具

[1] 商品カタログより一部抜粋．一部表示されていない対応商品があります．

合の収束に時間がかかるという課題が発生していた.

これは, テスト終盤になっても不具合が断続的に発生している状況である. テストの前半では, 一見すればわかる画面表示に関する不具合が多いが, 後半になると, これら画面表示に関する不具合は減少し, システム内部の振る舞いを通信ログやデバッグログで確認しないと原因がわからない不具合が見つかるようになる.

さらに, これらの不具合が商品間の通信に関するもの(以下, 通信不具合と呼ぶ)である場合, 複数の組織間で整合を図りながら, 不具合対応を進める必要があるため, 時間がかかることになる.

このため, テスト後半に通信不具合が多発すると, 不具合対応に予想以上の時間を要するという問題が発生し, 納期遅延のリスクが高まる.

3. 課題解決に向けて

3.1. 原因

2.3 節で述べた課題である不具合の収束に時間がかかることを解決するために原因を考える.

従来のシステムテストは, ユーザーインターフェースに対する機能ベースのブラックボックステストが中心であった. このブラックボックステストでシステム商品の内部の振る舞いである通信不具合を検出するには, 限界があり, 効率的に不具合を検出できないテストであった.

このため, 商品間の通信において, 詳細レベルを確認するテストが足りていないことが原因と考える.

3.2. 対策案

対策案として, 前節で述べた足りていないテストを追加する. このテストは, システム商品の内部の振る舞いを確認するテストであるので「システム商品のホワイトボックステスト」と位置付け「システム商品結合テスト」と呼ぶ. 一方, 従来のシステムテストで行っていたブラックボックステストは「システム商品適格性確認テスト」と呼ぶ.

システム商品結合テストは, システム商品の内部の振る舞いを確認するテストであるので, 方式設計書に基づき, 商品間の通信の網羅的なテストを行う.

次に, このテストの実施フェーズは,

- 商品開発後に行うシステムテストに位置付ける.
- ブラックボックステストであるシステム商品適格性確認テストの前に実施する.

とする. この理由は以下のとおりである.

一般に, 通信の詳細レベルのテストは, 商品開発におけるプロトタイプ開発や通信の方式評価で行い, 当社でも同様のフェーズで行っている. このフェーズでは, プロトタイプとしてエミュレータやシミュレータで確認しているが, これらの完成度が低い場合が多く, 商品開発時では, 接続に問題がなかったが, 実機どうしで接続すると接続できない, あるいは, 1 台は接続できるが, 複数台の接続はできないといった問題が従来のシステム商品適格性確認テストの開始時で発生していたためである.

以上より, システムテストは, 図 2のような段階的なシステムテストとなる.

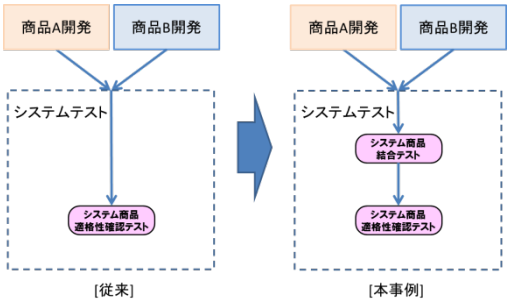


図 2 システムテストの流れ

3.3. 対策案の導入に向けた課題

システム商品結合テストを導入するにあたり, 主な課題を3つ説明する.

3.3.1. テスト担当者のスキルの問題

システム商品結合テスト(以下, 本テストと呼ぶ)の実施にあたり, 以下の観点で問題があった.

[通信の内容の確認スキル]

本テストは, ホワイトボックステストとして, 通信の内容を確認するテストなので, 従来のブラックボックステストで, ユーザインターフェースの動作を確認してきたシステム商品適格性確認テストの担当者が行えるかという問題がある. 具体的には, 通信モニタのログから通信シーケンスや通信データなどを確認し, テスト結果の合否判断を行うことになる. このため, システム商品適格性確認テストの担当者は, テストの実施はできるが, テスト結果の確認ができない可能性があり, 商品開発の担当者が本テストを実

施したほうが良いと考える.

[異なる組織と調整するスキル]

本テストの担当者は, 仕様整合時や不具合対応時などは, 商品毎に異なる組織の担当者との意思の疎通を行い, 整合する必要がある. しかし, 同一の組織内での商品開発のみを行ってきた担当者では, このような経験が少ないので, うまくいかないことが考えられる. さらに, 自組織の利害関係を優先したりすると, 不具合の修正方針を公平に判断できず, 担当する商品の変更をできるだけ少なくする方針で対応したりすることが予想され, システム商品の品質が低下することが考えられる. このため組織間で公平な対応ができる第三者が対応することがより良いと考え, システム商品適格性確認テストの担当者が本テストを担当したほうが良いと考える.

3.3.2. テスト設計に方式設計書が必要

システム商品結合テストでは, 通信の内容の詳細を確認する十分な方式設計書をもとにテスト設計を行う必要がある. この通信に関する方式設計書は, 商品開発側で作成するものであるが, これまでは, システム商品適格性確認テストでは必要ないものであったため, その記載レベルが不明である. このため, 本テストで十分な内容であることを確認する必要がある.

もしも, この設計書の記載レベルが低い場合, 開発担当者に設計書に通信内容を追記してもらう必要がある.

この設計書の確認時期については, 商品開発が始まり, 方式設計フェーズで確認するのが効率的と考える.

3.3.3. ホワイトボックステストのテスト工数増大への対応

一般にホワイトボックステストを実施する場合, テスト項目が増大するという問題がある. しかし, 今回は, 不具合の早期検出を目的として, 不足している通信内容の詳細な確認テストを追加することにしている.

このため, 納期遅延リスクが増えない範囲で, テスト項目の増加によるテスト工数の増加を許容し, テスト実施を効率化する必要がある.

すなわち, テスト工数の増加によるテストカバレッジの向上とテスト実施時間の短縮による効率化を実現することを考える.

次に, システム商品結合テストで行う通信内容の詳細なレベルでの確認方法についてその考え方を説明する.

例えば, 通信に異常が発生した場合のシステム商品の動作を確認するテストで説明する.
この場合, 通信断を発生させてシステム商品の挙動を確認する. この挙動として, 通信失敗がユーザに表示されるか, 通信断が復旧すると, システム商品が正しく動作しているかを確認することになる. 図 3 に示すような, 商品 P が商品 A の 3 つの状態を取得する場合を考える. この場合, 3 往復の通信シーケンスとなり, 商品 P で 1 往復毎にタイムアウトが設定されているとする[5].

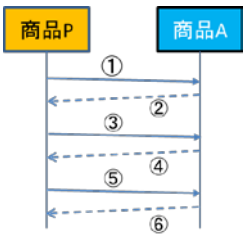


図 3 通信シーケンス

従来のシステム商品適格性確認テストでは, このようなテストは, テスト担当者が通信タイミングをみて, 通信線を抜くといった手順で行うことになる. このため, 確実に発生させることができず, 複数回の実施が必要であり, 時間のかかるテストとなっていた. また, ①～⑥の全てのシーケンスで通信失敗を発生させることは非現実的で, ①～⑥のいずれかのシーケンスで通信失敗となるテストとなっていた. このため, 10 回実施しても通信失敗を 2, 3 回しか発生させることができないといった状況であった.

一方, システム商品結合テストでは, ①～⑥の全てのシーケンスで確実に通信失敗させることを目指し, 任意のシーケンスを破棄させるようなテストツールを作成し, 確実に 1 回で指定したシーケンスを通信断できるようにする. この場合, 6 回のテスト実施で全てのシーケンスで通信断を発生させることが可能となる.

以上のことから, システム商品結合テストのほうがカバレッジの向上と実施時間の短縮が可能となり, 効率的なテスト実施を行い, 不具合を確実に検出できるようになると考える.

4. 事例紹介

3.3 節の課題を解決し, システム商品結合テストを導入した事例について, 当社のシステム商品であるスマート HEMS の開発プロジェクトを紹介する.

4.1. テスト担当者の決定

スマート HEMS の開発プロジェクト発足にあたり，システム商品結合テストのチーム体制を以下のように決定した．

- 商品開発のメンバに，通信に関するスキルがあり，かつシステムテストを経験したメンバがいたので，システム商品結合テストの担当者とした．
- システム商品適格性確認テストを担当する組織には，通信に関するスキルを持つメンバもいたが，他の開発プロジェクトを担当していたので，本プロジェクトにアサインできなかった．

次に，テスト担当者は，他の組織と調整を行う必要があるので，商品の開発には関らず，システム商品結合テスト専任とした．テスト担当者を専任とすることで，開発プロジェクト発足時から参加可能となり，他の組織との関係を早くから築くことができた．さらに，商品開発を担当していないので，個別の商品に肩入れせずに，システム商品全体を見渡し，各商品の開発を行っている様々な組織と公平に調整ができた．

4.2. テスト設計に必要な方式設計書の充実

4.1 節のテスト担当者は，システム商品結合テストの設計に必要な方式設計書を商品開発担当者に開発開始時に作成依頼と提供依頼を行うようにした．さらに方式設計フェーズで方式設計書のデザインレビューにもレビューアとして参加し，内容の確認を行い，不足していると思われる内容については追加の依頼を行った．

実際に，追加した内容としては，通信の異常系に関するものが多かった．正常系の通信に関する方式設計は，問題が少なかった．この結果，方式設計書の内容が充実し，品質が向上した．

想定外の問題としては，システムテストが開始されると商品開発の担当者の一部が他の開発プロジェクトに異動した．この影響で，方式設計書のメンテナンスができなくなった．不具合発生時などで仕様変更となる場合，方式設計書を変更してもらう必要があるが，担当者がいない状況となっていた．このため，緊急対応として，システム商品結合テストの担当者が修正対応を行った．

4.3. テスト工数増大への対応

3.3.3 節で述べたように，テスト工数の増加によるテストカバレッジの向上のとテスト実施時間の短縮による効率化を実現する方法として，テストツールの活用を提案した．

今回，スマート HEMS で開発したテストツールを紹介する．このテストツールは，意図した通信異常を確実に発生させることができるツールである．

従来のシステムテストでは，通信異常に関するテストに多くの時間を要していたので，テストツールを開発し，効率化を実現した．

説明を簡単にするため，スマート HEMS における AiSEG2 とエアコンとの通信動作を考える．この AiSEG2 とエアコンは，特定小電力無線を使って通信を行う．

この場合，図 4 の(a)に示すように，AiSEG2 はエアコンに対して要求パケットを送信するとエアコンが応答を返す通信シーケンスを考える．この通信を図 4 の(b)に示すようにテストツールがパケットを破棄して切断することで AiSEG2 がタイムアウトし，通信異常となる．この時の AiSEG2 の動作を確認するテストが可能となる．

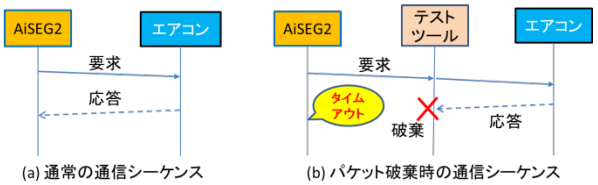


図 4 テストツールによる効率化

システム商品適格性確認テストとテストツールを活用した場合を順に表 1 と表 2 に示す．

表 1 システム商品適格性確認テストの場合

手法	実機 2 台の物理的な距離を離して，パケットの破棄または，遅延を実現する
発生回数 (カバレッジ)	要求か応答のいずれかのパケットを対象とし，破棄か遅延のいずれかを 1 回だけ発生
試行回数	発生するまで何回でも
所要時間	平均 1 時間

表 2 システム商品結合テストでテストツールを活用した場合

手法	実機2台とテストツールを使って机上で容易にパケットの破棄と遅延を実現する
発生回数(カバレッジ)	要求と応答で各 1 回と破棄と遅延の各 1 回で 2 回×2 回の計4回
試行回数	4 回
所要時間	平均15分

以上のようにテストツールを活用することで，通信異常のテストは，テストカバレッジが 4 倍になり，かつ，所要時間を約 1/4 に削減できた．

このことから，通信異常のテストでは，約 16 倍の効率化を実現できた．

5. 成果と検証

システム商品結合テストを導入した成果と検証を行う．

5.1. 不具合の早期発見を実現

システム商品結合テストを実施することで，多くの通信不具合を検出することができた．通信不具合は，システム商品結合テスト全体の不具合の約 3/4 を占めていた．通信シーケンス，通信性能，通信競合に関するものが検出できた．

また，システムテスト全体の通信不具合の約 95%は，システム商品結合テストで検出できた．

このことから，通信不具合の多くをシステム商品結合テストで早期に検出でき，システム商品適格性確認テストの前に検出することができた．

5.2. システムテストの効率化

通信不具合の対応に関しても，方式設計書に基づく指摘であるため，公平な立場で商品開発側に対応をお願いできるようになり，組織の力関係や個別の商品の都合で対応方針が決まることが少なくなり，異なる組織の開発担当者と整合がうまくでき，不具合の収束も想定範囲内で推移し，計画通りにシステムテストを完了することができた．

今回，通信の内容確認ができるスキルを持つ開発経験者を専任の担当者とすることで，開発の初期段階からテスト設計，テストツールを含むテスト環境の準備ができ，システム商品結合テストをスムーズに導入することができた．

さらに，システム商品適格性確認テストの開始をスムーズに始めることができるようなり，システム商品適格性確認テストの担当者は，システム商品結合テストの重要性を理解した．

5.3. 今後の課題

5.3.1. システム商品結合テストの定着化

今回の事例では，開発側にテスト経験のあるメンバをテスト担当者に決めたが，このようなスキルを持ったメンバは少ない．今後，システム商品結合テストを開発側が行った方式設計書に対する検証フェーズと考えた場合，定着化させるためには，開発側メンバに対するテストスキルの育成が必要となる．

また，方式設計書の品質が維持できるように，システムテストフェーズにおいても，開発側の担当者が必要であることを周知し，開発側で修正できる体制の維持が必要と考える．

5.3.2. システム商品結合テストの効率化

IoT システムは，今後も規模が大きくなり，様々な商品が接続されていくと予想される．これに伴い，システム商品結合テストの工数も増加すると考えられる．このため，テスト自動化による効率的なテスト実施を実現し，テスト工数の増加を最小限に抑えたい．

6. おわりに

他社商品や異なる組織で開発が行われている様々な商品で構成されるIoTシステム開発において，テスト担当者の決定，通信に関する方式設計書の充実．テスト工数増大の対応を行い，システム商品結合テストを導入した．この結果，通信不具合の早期検出が可能となり，不具合の収束を早く行えるようになり，システムテストを効率良く進めることができた．

今後の課題としては，システム商品結合テストの定着化と，システム規模拡大に対応した効率化が必要と考える．

また，今回，システム商品結合テストを実施することで，通信品質が向上したと考えるので，今後の市場トラブルを未然に防ぐことができたことを確認したい．これは，商品発売後の初期トラブルの発生数を他の商品と比較することで確認したい．

謝辞

今回の経験論文の作成において，忙しい中，様々なアドバイスをいただいた当社の山本氏，藤村氏，植月氏に感謝いたします．

参考文献

[1] wikipedia, “モノのインターネット”, https://ja.wikipedia.org/wiki/モノのインターネット.
[2] モバイルコンピューティング推進コンソーシアム,“IoT 技術テキスト”, リックテレコム (2016/10/28).
[3] ISTQB テスト技術者資格制度 用語集, JSTQB, http://www.jstqb.jp/dl/JSTQB-glossary.V2.3.J02.pdf.
[4] スマート HEMS，Panasonic, http://www2.panasonic.biz/es/densetsu/aiseg/aiseg2/system.html.
[5] 竹政昭利 他，“かんたん UML 入門”，技術評論社 (2013/6/7).
[6] 独立行政法人 IPA,”【改訂版】組込みソフトウェア向け開発プロセスガイド”，翔泳社(2007/11/19).

例外処理を含むJavaプログラムを対象とした データ遷移可視化ツールTFVISの適用範囲の拡大

佐藤　拓弥
宮崎大学大学院工学研究科
sato@earth.cs.miyazaki-u.ac.jp

片山　徹郎
宮崎大学工学教育研究部
kat@cs.miyazaki-u.ac.jp

水久保　直哉
株式会社スカイコム
mizukubo@skycom.jp

田中　伸英
株式会社スカイコム
tanaka@skycom.jp

要旨

ソフトウェア開発工程において、デバッグは手間と時間のかかる工程である。このデバッグにかかる手間と時間の削減を目的として、我々の研究室ではデータ遷移可視化ツール *TFVIS* を開発した。*TFVIS* は、データ遷移可視化と実行フロー可視化によって、プログラム実行時の挙動把握を支援する。*TFVIS* の可視化により、欠陥を含んだプログラムの実行時の挙動把握を容易にし、プログラムが含む欠陥の特定を支援する。しかし、*TFVIS* は一部の制御構造や式にしか対応しておらず、有用性が高いとは言えない。そこで、例外処理を含む *Java* プログラムを対象とした適用範囲の拡大を行った。これにより、*TFVIS* の *Java* プログラム可視化ツールとしての有用性が向上したと言える。

1. はじめに

ソフトウェアの開発工程において、デバッグは手間と時間のかかる工程である [1]。プログラムの故障により期待した結果を得られない場合、故障の原因である欠陥を特定する必要がある。しかし、この欠陥特定の作業は困難である [2]。

効率よくプログラムの欠陥を特定するためには、プログラムの動的な挙動を理解することが重要である [3]。しかし、プログラムの動的な挙動は、一般的に不可視であるため、把握することが困難である [4]。

この問題を解決するため、我々の研究室では Java プログラムの動的な挙動を可視化するツール TFVIS を開発した [5]。

TFVIS は、データ遷移可視化と実行フロー可視化によって、プログラム実行時の挙動把握を支援する。TFVIS の可視化により、欠陥を含んだプログラムの実行時の挙動把握を容易にし、プログラムが含む欠陥の特定を支援する。また、他の機能として、データ遷移を矢印を用いて示すことができる。これにより、プログラムの不具合から欠陥の特定を容易にする。しかし、TFVIS は一部の制御構造や式にしか対応しておらず、有用性が高いとは言えない。

そこで本稿では、未対応の制御構造の 1 つである、例外処理を含む Java プログラムを対象とした適用範囲の拡大を行う。具体的には、Try Catch 文を含むプログラムを可視化できるように拡張を行う。これにより、TFVIS の Java プログラム可視化ツールとしての有用性の向上を目指す。

2. TFVIS

今回適用範囲の拡大を行う TFVIS の機能と構造について、それぞれ以下で説明する。

2.1. 機能

図 1 に、TFVIS の外観を示す。TFVIS はソースコードからプログラムの構造を解析した構造情報と、実行時

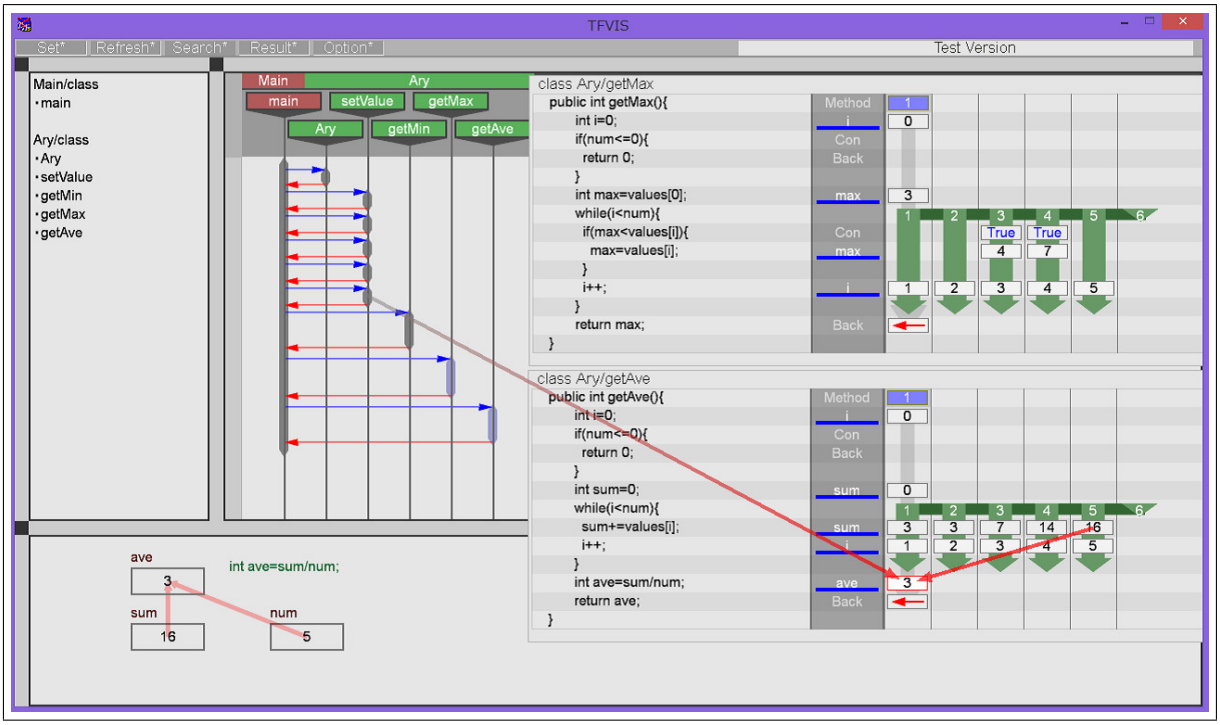


図 1. TFVIS の外観

の情報を基に、データ遷移図を生成する。データ遷移図は、ループ処理や分岐などによる処理の流れの変化、変数の更新によるデータの移り変わりなど、プログラム実行時の各メソッドの詳細な挙動を示す。

図 1 を例に説明する。図の右側に見える個々のウィンドウが、各メソッド内の処理を示すデータ遷移図である。ウィンドウの左側にメソッドのソースコードを、中央に各行で起こるイベントや変数名を、右側に処理の流れを示している。if 文や for 文によって、処理の流れが変化する場合、右側に濃い緑色で処理の流れの様子を描画する。また、メソッド中で変数の更新を行う処理がある場合、更新後の値をボックスに記述し、対応するソースコード右側に描画する。

ウィンドウの左側に見える縦に伸びるラインを描画してある図が、プログラム実行全体の処理の流れを示す実行フロー図である。ライン上にある太い部分がメソッドの活性区間であり、これを選択することで、該当するメソッドのデータ遷移図が確認できる。メソッドの活性区間から伸びる青い矢印はメソッド呼び出しを、赤い矢印はメソッドの処理の終了を示す。

TFVIS のもう 1 つの機能として、よる変数同士の関係を可視化するデータ遷移線がある。データ遷移線は、ある特定の変数が更新された際に、その更新に影響を与えた変数がプログラムのどの時点で生成されたものであるかを示す。例えば、「a=b+c」という式が存在するとき、「a」の値を選択することで、更新に関わった「b」と「c」の値が生成された時点を赤色の矢印を用いて示す。このデータ遷移線を活用することで、特定の変数がどのような変数同士の関係で生成されたかを確認できる。これにより、ユーザがデータ遷移図上で変数の不審な値を発見した場合に、データ遷移線を活用することによって、不審な値を生成した原因の特定が容易になる。

さらに、TFVIS はプログラム全体の流れを、UML のシーケンス図 [6] を基にした実行フロー図によって可視化する。これにより、各クラスのメソッドの使用状況や、メソッド呼び出しの関係を表す。プログラム全体の処理を実行フロー図によって可視化することで、各メソッドの詳細な挙動を示すデータ遷移図の活用を補助する。

2.2. 構造

図 2 に、TFVIS の構造を示す。TFVIS は、解析部と可視化部から成る。また、解析部は、構造解析部、プロー

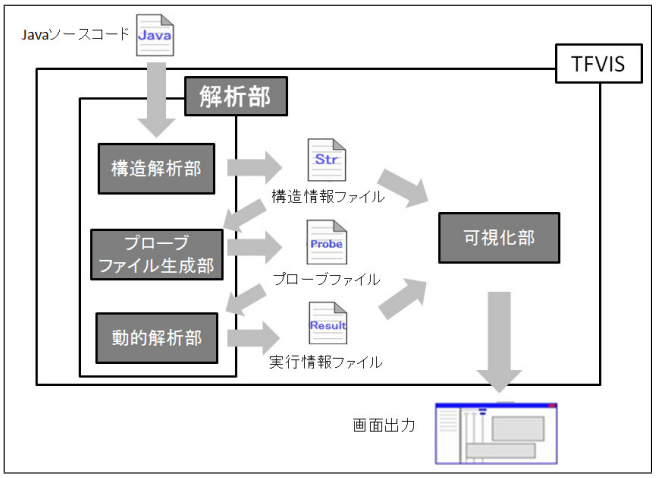


図 2. TFVIS の構造

ブファイル生成部、動的解析部から成る。

構造解析部では、プログラムの構造の解析を行い、解析結果を構造情報としてファイルに出力する。構造情報は、プローブファイル生成部でのプローブ挿入箇所の判断と、可視化部での図表の作成に用いる。構造解析部によって、ソースコードの各行で起こるイベントを取得する。イベントとは、可視化の基準となる特定の処理であり、各イベントはイベント種別の値を持つ。

プローブファイル生成部では、構造情報を基に、対象ソースコードにプローブを埋め込んだプローブファイルを生成する。プローブは、プログラム実行時の挙動の情報を出力する。なお、プローブにはいくつか種類があり、各コードで起きるイベントごとに挿入するプローブが変わる。

動的解析部は、プローブファイルから、実行時の挙動を解析し、解析結果を実行情報として出力する。具体的には、ソースコードにプローブを挿入したプローブファイルをコンパイルし実行することで、プローブが出力する実行情報を取得し、実行情報をファイルに出力する。

可視化部は、解析部が出力する構造情報と実行情報を基に、可視化を行う。

3. TFVIS の拡張

本章では、Try Catch 文への対応のために行った拡張について述べる。初めに、対応のために行った各部の拡張について述べる。次に、改良後の TFVIS のデータの流れについて述べる。

3.1. 各部の拡張点

Try Catch 文への対応のために行った拡張は、以下のとおりである。

- 構造解析部の新たなイベント種別の値の定義
- プローブファイル生成部の Try Catch 文に対する新たなプローブの定義
- プローブファイル生成部の Try Catch 文に対するプローブの挿入
- 可視化部の Try Catch 文のイベントに対する可視化

各拡張の詳細について、以下で述べる。

3.1.1 構造解析部の新たなイベント種別の値の定義

構造解析部において、Try Catch 文に対し出力するイベント種別の値を新たに定義する。

新たに定義したイベント種別の値は、Try ブロック開始の値 (380)、Try ブロック終了の値 (382)、Catch ブロック開始の値 (390)、Catch ブロック終了の値 (392) である。

3.1.2 プローブファイル生成部の Try Catch 文に対する新たなプローブの定義

プローブファイル生成部において、Try Catch 文のイベントに対して挿入するプローブを新たに定義する。

Try ブロックのイベント用のプローブを、Try 処理検出プローブとする。このプローブは、実行時のインスタンスの ID、メソッド ID、メソッド実行番号、行番号を引数とする。そして、可視化で用いる Try イベント ID(380)、インスタンス ID、メソッド ID、メソッド実行番号、行番号を、実行情報ファイルに出力する。

同様に、Catch ブロックのイベント用のプローブを、Catch 処理検出プローブとする。このプローブは、実行時のインスタンスの ID、メソッド ID、メソッド実行番号、行番号を引数とする。そして、可視化で用いる Catch イベント ID(390)、インスタンス ID、メソッド ID、メソッド実行番号、行番号を、実行情報ファイルに出力する。

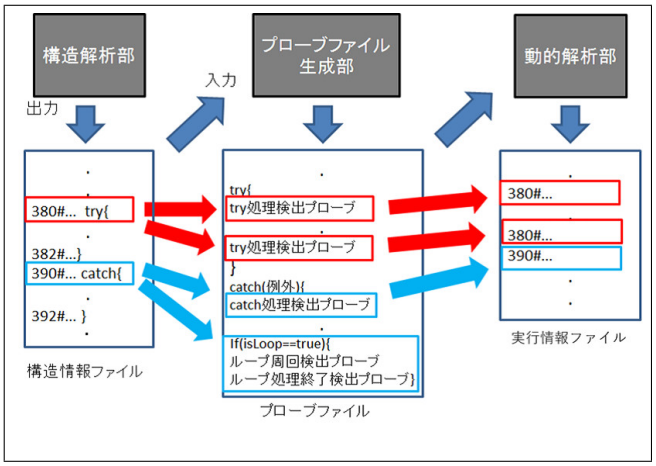


図 3. 解析部のデータの流れ

3.1.3 プローブファイル生成部の Try Catch 文に対するプローブの挿入

プローブファイル生成部において、Try Catch 文のイベントに応じて新たに定義したプローブを挿入する。

Try ブロック開始のイベントを読み込んだ場合、Try ブロック開始から Try ブロック終了までの全ての行の直前に、Try 処理検出プローブを挿入する。このときプローブが得る行番号の値は、直後の行の行番号である。また、Catch ブロック開始のイベントを読み込んだ場合、直後の行に Catch 処理検出プローブを挿入する。

ループ中に Catch ブロックを実行する場合、ループから抜ける処理が存在する。このような処理に対応するため、メソッド開始の行の直後に、ループ中かどうかを示す boolean 型の変 “isLoop”の定義を挿入する。そして、ループ中であれば、“isLoop” が true になるように変更を行った。また、Catch ブロック終了の行の直前に、”isLoop” が true であれば、既存のループ周回検出プローブとループ処理終了検出プローブを実行する if 文を挿入する。

3.1.4 可視化部の Try Catch 文のイベントに対する可視化

例外処理のイベントの発生に対し、データ遷移図の直前の実行の箇所に赤色で “Catch”と記述したボックスを配置する。

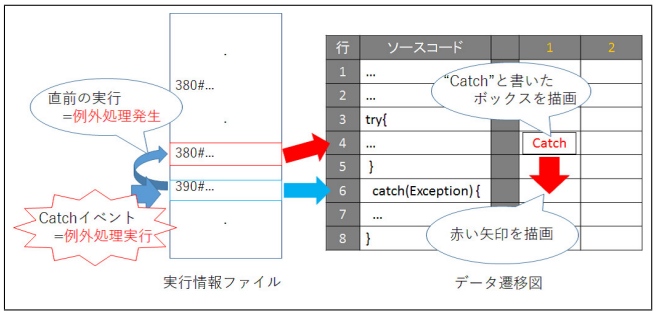


図 4. 可視化の流れ

また、例外処理の発生箇所と実行箇所を結ぶ赤色の矢印を記述する。

3.2. 改良後のデータの流れ

拡張後の TFVIS の解析部と可視化部における、詳細なデータの流れについて、以下で述べる。

3.2.1 解析部のデータの流れ

図 3 に、拡張後の TFVIS に、Try Catch 文を含むプログラムを適用した際のデータの流れを示す。

初めに、構造解析部の拡張によって、Try Catch 文についてのイベントを取得する。図 3 から、構造情報ファイルが Try ブロック開始の値 (380)、Try ブロック終了の値 (382)、Catch ブロック開始の値 (390)、Catch ブロック終了の値 (392) を持っていることが分かる。

次に、構造解析部で取得した Try Catch 文についてのイベントによって、プローブファイル生成部が新たに定義したプローブを挿入する。図 3 から、Try ブロックの全ての行の直前に Try 処理検出プローブを、Catch ブロック開始の行の直後に Catch 処理検出プローブを挿入していることが分かる。また、ループ中であれば、ループ周回検出プローブとループ処理終了検出プローブを実行する if 文を挿入していることもわかる。

最後に、プローブを埋め込んだプローブファイルを動的解析部で実行することで、実行情報を取得する。

3.2.2 可視化部の流れ

図 4 に、拡張後の TFVIS に、Try Catch 文を含むプログラムを適用した際の可視化の流れを示す。

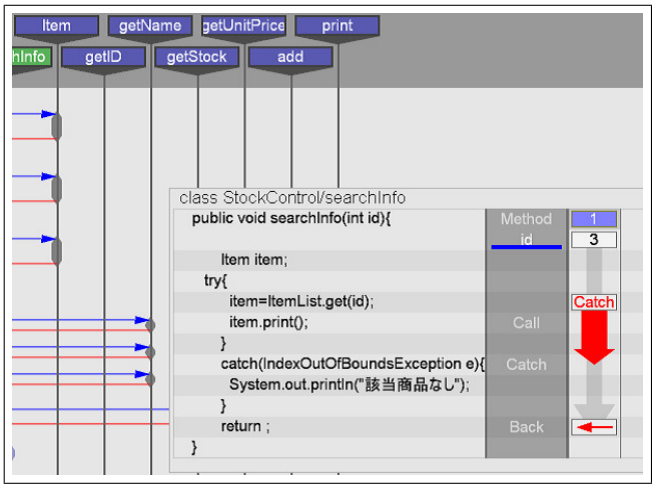


図 5. SearchInfo メソッドの可視化

構造情報に基づき、実行フロー図とデータ遷移図上のソースコードを描画する。また、実行情報に基づき、データ遷移図を描画する。

可視化部が Catch イベント ID を含む実行情報を読み込んだ場合、可視化部は Catch イベントの行を例外処理実行箇所として、その直前の実行の行を例外処理発生箇所として保持する。そして、例外処理実行箇所に赤色で“Catch”と記述したボックスを、例外処理発生箇所から例外処理実行箇所まで赤色の矢印を描画する。

4. 適用例

本章では、3 章で説明した TFVIS の拡張によって、Try Catch 文を含むプログラムが、正しく可視化できることを確認する。適用例として、Java で記述した「在庫管理プログラム」を適用し、例外処理を正しく可視化することを示す。また、「与えられた数の最小、最大、平均値を計算するプログラム」を適用し、TryCatch 文を含むプログラムの可視化において、データ遷移線が正しく機能することを示す。

4.1. 在庫管理プログラム

図 5 に、「在庫管理プログラム」において、Try Catch 文を含む SearchInfo メソッドを可視化したデータ遷移図を示す。このメソッドは、「ID」を入力として、その「ID」を持つ在庫の「ID」と「名前」、「在庫数」、「単価」を表示するメソッドである。

今回作成した在庫は 3 つのみのため、3 つの在庫のインスタンスを生成し、リストに格納した。このときユーザが「ID」を 3 と入力したと仮定すると、リストの 4 番目を参照したことになり、「IndexOutOfBoundsException」という例外が発生する。図 6 から、リストを参照する行に“Catch”のボックスと、このボックスから、発生した例外処理の行までの赤色の矢印を表示しており、データ遷移図で例外処理の発生を正しく可視化していることが分かる。

4.2. 最大、最小、平均値を計算するプログラム

図 6 に、「与えられた数の最小、最大、平均値を計算するプログラム」において、TryCatch 文を含む getAve メソッドを可視化したデータ遷移図を示す。このメソッドは、与えられた数の平均値を計算するメソッドである。

今回の適用例では、このメソッドに故意に例外が発生する処理を記述し、例外処理内の処理でデータ遷移線が機能するか確認を行った。今回基点として選択した更新値は、getAve メソッド内のループ処理の 5 回目のループで更新された変数「sum」の「16」という値である。この「sum」は、「sum+=values[i]」という式からから計算される値である。この更新値の算出に用いた「sum」、「values[i]」、「i」を生成した箇所をそれぞれ赤色の矢印で描画している。このことから、例外処理を含むプログラムにおいても正しくデータ遷移線が機能していることが分かる。

5. 考察

本稿では、未対応の制御構造の 1 つである、例外処理を含むプログラムへの適用を目的とした拡張を行った。具体的には、Try Catch 文を含むプログラムを可視化できるように拡張を行った。本章では、初めに TFVIS の拡張の評価を述べる。次に、関連研究について述べる。最後に、TFVIS の課題について述べる。

5.1. 評価

既存の TFVIS では、Try Catch 文を含むプログラムを適用した場合、コンパイルエラーが発生し、可視化を行うことができなかった。

Try Catch 文を含むプログラムを可視化するため、初めに、構造解析部において、Try Catch 文に対して新た

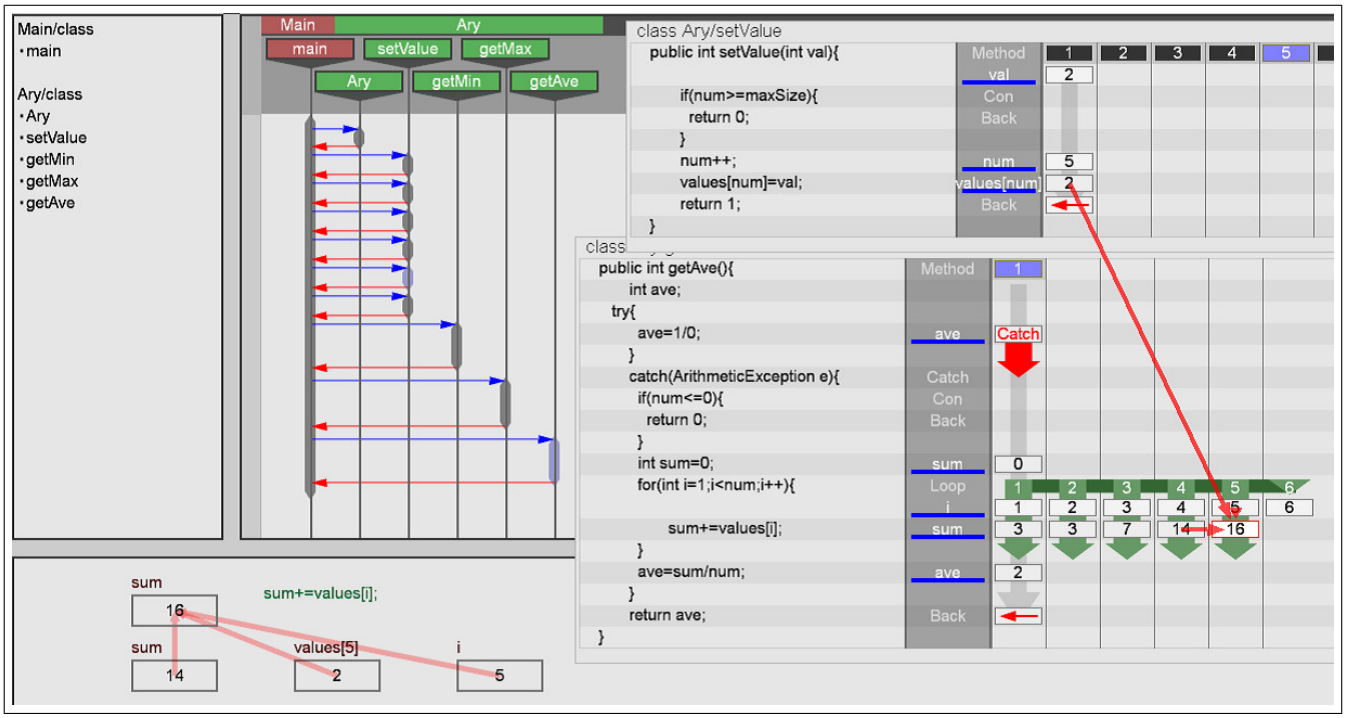


図 6. getAve メソッドの可視化

に定義したイベント種別の値を用いることでイベントを取得する拡張を行った。

次に、プローブファイル生成部において、Try Catch 文についてのイベントに対して、新たに定義した Try Catch 文の実行時の情報を出力するプローブを挿入する拡張を行った。

最後に、可視化部において、Catch イベントに対して、直前の実行に赤色で “Catch” と記述したボックスを描画し、例外処理の発生箇所と実行箇所を結ぶ赤色の矢印を描画する拡張を行った。

以上の拡張から、既存の TFVIS では可視化できなかった Try Catch 文を含むプログラムを、拡張後の TFVIS は可視化できる。また、Try Catch 文を含むプログラムで、既存のデータ遷移線が機能することも確認できた。このことから、Try Catch 文への対応により、TFVIS の実用性が向上したと言える。

5.2. 関連研究

以下に、TFVIS と関連研究との比較を述べる。

- ブレークポイントデバッグ

ブレークポイントは、最も多用されているデバッグ支援手法の 1 つである [7]。ブレークポイントを用いたデバッグでは、プログラムの実行を任意の箇所で停止し、停止した時点での各変数の値など、プログラムの実行状況を確認することができる。

ブレークポイントデバッグには、ブレークポイントを設置する箇所の選定が難しいという問題点が存在する。プログラムの欠陥を特定するのに適当な設置箇所を選定するためには、ユーザの知識と経験が必要である [7]。

これに対して TFVIS は、ユーザが欲する情報を保持するメソッドを選択するだけで必要な情報を得ることができ、ユーザの能力に依存せずに使用できるという点で優れていると言える。

さらに、TFVIS による可視化では、メソッドやプログラム全体の流れを俯瞰することができる。また、データ遷移線を活用することで、変数同士の依存関係を把握することができる。これらの機能から、ブレークポイントによるデバッグに比べ、ある変数がどのような経緯で作られたのか調べることができる、という点で優っていると言える。

- JIVE

  JIVE[8] (Java Interactive Visualization Environment) は、Java プログラムの実行を可視化するツールである。

  JIVE は、実行時の処理から UML のオブジェクト図とシーケンス図を生成する機能を持つ。また、クエリーによる問い合わせに対応しており、例えば、「メソッド “func” が返り値に NULL を返すのはどこか」といった問い合わせが可能である。問い合わせで発見した処理は、シーケンス図上でハイライトされ、プログラム実行時の挙動把握を支援する。

  JIVE と TFVIS を比較した場合、JIVE には、データ遷移のような変数同士の依存関係を示す機能はない。変数更新の問い合わせが可能であるが、データ遷移のような依存関係を調べる場合には、繰り返し問い合わせを行う必要がある。そのため、不審な値を見つけた際に、その原因を探るといった作業には、TFVIS がより効果的であると言える。

- Code Canvas

  Code Canvas[9] は、IDE である Visual Studio のズーム可能なサーフェスである。Code Canvas には、スタックトレースから例外発生箇所とメソッド呼び出し関係を、複数のソースコードにわたって赤い矢印で描画する機能を持つ。

  TFVIS と比較した場合、Code Canvas は例外処理の呼び出し関係の把握をスタックトレースを可視化することで支援するが、例外発生箇所の変数の値や処理の流れなどは調査できない。TFVIS による可視化では、例外発生箇所に不審な値がある可能性が高いため、そこからデータ遷移を辿っていくことで効率よく欠陥を特定できる。この点で、TFVIS による例外処理の可視化は有用であると言える。

5.3. 今後の課題

以下に、TFVIS の課題について述べる。

- レスポンスの遅さ

  TFVIS が可視化を行うためには、読み込む対象のプログラムが終了するまで待たなければならない。また、可視化を行うためには、いくつかの手順を踏まなければならず、1 回の可視化を行うのに、2 分程度の時間がかかってしまう。

- 入力待ち状態の発生を含むプログラムへの対応

  TFVIS はユーザからの入力を受け取る機能を持たない。そのため、入力待ち状態の発生を含むプログラムを適用した場合、解析部の実行が終了せず、実行情報を得ることができない。

- 問い合わせ機能の実装

  TFVIS は問い合わせ機能を持たない。この問題は可視化対象のプログラムの処理が増加するほど、大きな手間となる。そのため、問い合わせ機能を実装する必要があると考えている。

- マルチスレッドプログラムへの対応

  TFVIS はマルチスレッドプログラムを可視化できない。マルチスレッドプログラムでは、スレッド間でデータのやり取りが行われるため、処理が複雑になりやすい。もし、マルチスレッドによる複雑な処理を可視化できれば、TFVIS の有用性がより高まると言える。

以上の課題のうち、レスポンスが遅いという問題点について詳しく述べる。上記で述べたように、TFVIS が可視化を行うためには、読み込む対象のプログラムが終了するまで待たなければならない。また、可視化を行うためには、いくつかの手順を踏まなければならず、1 回の可視化を行うのに、コードの大きさに関わらず 2 分の時間がかかってしまう。これは、プログラムを任意の点で停止し、実行状況を把握できるブレークポイントに比べ、レスポンスが遅いと言える。

しかし、関連研究でも述べたように、TFVIS による可視化には、メソッドやプログラム全体の流れを俯瞰できる、データ遷移線を活用することで変数同士の依存関係を把握できるといった強みが存在する。

そこで、ブレークポイントデバッグを支援するツールとして TFVIS の改良を行う。具体的には、Java 言語の IDE として広く使われており、プラグインの開発が可能な Rclipse[10] のブレークポイント機能と連携を行う。ブレークポイントで停止した時点で、その時点までの実行フロー図やデータ遷移図を TFVIS で表示することで、TFVIS のレスポンスの問題とブレークポイントデバッグの処理の流れの把握が困難であるという問題点を解決

できるのではないかと考える。さらに、ブレークポイント機能と合わせて、データ遷移線による変数同士の依存関係を確認することで、より効率的な欠陥の特定が可能になると考える。

以上の点から、TFVIS の今後の展望として、eclipse プラグインとして改良を行い、ブレークポイント機能と連携することで、ブレークポイントデバッグを支援するツールを目指す。

6. おわりに

本稿では、未対応の制御構造の 1 つである、例外処理を含む Java プログラムを対象とした適用範囲の拡大を行った。TFVIS は、プログラム実行時の挙動を、データ遷移可視化と実行フロー可視化によってユーザに示す。TFVIS の可視化により、欠陥を含んだプログラムの実行時の挙動把握が容易になり、欠陥を効率的に特定できるようになる。また、データ遷移線の機能から、データ遷移を辿ることができ、不具合から欠陥の特定を支援することができる。

しかし、既存の TFVIS では可視化できないプログラムが存在する。Java プログラムが持つ基本的な構文を含むプログラムが可視化できないことは有用性に欠けることを意味している。

そこで本稿では、Try Catch 文を含む Java プログラムを対象とした、TFVIS の適用範囲の拡大を行った。今回の拡張により、TFVIS の Java プログラム可視化ツールとしての有用性が向上したと言える。

今後の課題を以下に示す。

- レスポンスの遅さ
- 入力待ち状態の発生を含むプログラムへの対応
- 問い合わせ機能の実装
- マルチスレッドプログラムへの対応

参考文献

[1] Thomas D. LaToza, Gina Venolia, and Robert DeLine: Maintaining mental models: a study of developer work habits, Proceedings of the 28th international conference on Software engineering, pp.492-501 (2006).

[2] Roger S. Pressman: Software Engineering A Practitioner's Approach, McGraw-Hill Science (2001).

[3] Jonathan Sillito, Gail C. Murphy, and Kris De Volder: Asking and Answering Questions During a Programming Change Task, IEEE Transactions on Software Engineering, Vol.34, No.4, pp.434-451 (2008).

[4] Andreas Zeller, (訳: 中田秀基, 今田昌宏, 大岩尚宏, 竹田香苗, 宮原久美子, 宗形紗織): デバッグの理論と実践-なぜプログラムはうまく動かないのか, オライリー・ジャパン (2012).

[5] Hiroto Nakamura, Tetsuro Katayama, Yoshihiro Kita, Hisaaki Yamaba, Kentaro Aburada and Naonobu Okazaki: TFVIS: a Supporting Debugging Tool for Java Programs by Visualizing Data Transitions and Execution Flows, The 2015 International Conference on Artificial Life and Robotics, pp.376-379 (2015).

[6] Dan Pilone, Neil Pitman, (訳: 原 隆文): UML2.0 クイックリファレンス, オライリー・ジャパン (2006).

[7] Cheng Zhang, Juyuan Yang, Dacong Yan, Shengqian Yang and Yuting Chen: Automated Breakpoint Generation for Debugging, Journal of Software, Vol.8, No.3, pp.603-616 (2013).

[8] Demian Lessa, Bharat Jayaraman and Czyz Jeffrey: JIVE: A Pedagogic Tool for Visualizing the Execution of Java Programs. Technical Report 2010-13, Department of Computer Science and Engineering, University at Buffalo (2010).

[9] R. DeLine and K. Rowan: Code Canvas: Zooming towards better development environments, in Proc. of the 32nd International Conference on Software Engineering, Vol.2, pp.207-210 (2010).

[10] Eclipse Foundation: Eclipse - The Eclipse Foundation open source community website., `https://eclipse.org/`

Data Race Detection Based on Dependence Analysis

Guanqun Wang
Hitachi Automotive Systems, Ltd.
guanqun.wang.ju@hitachi-automotive.co.jp

Masahiro Matsubara
Hitachi Automotive Systems, Ltd.
masahiro.matsubara.td@hitachi-automotive.co.jp

Abstract

Concurrency issues have been a serious problem in the whole software industry. They very often lead to hard-to-reproduce bugs and thus are very difficult to locate and solve. In this paper, we introduce a novel method for detection of one major category of concurrency problems, data race. This method extracts a Variable Dependence Tree from source code and finds data race by analysis on the Variable Dependence Tree. A prototype is built in Java according to the proposed method and tested on production-level source code (C language) with more than 400k LLOC. As a result, the prototype tool successfully detected all known data races including one relying on redundant structure of software and proved the effectiveness and feasibility of our proposed method.

1. Introduction

With the rapid growing complexity of software systems, concurrency issue has been an prevailing problem in software industry, which has drawn a lot of attentions. Since concurrency issues can easily lead to hard-to-reproduce bugs which take a long time to locate and fix, they are one of the most urgent problems to solve for cost control. According to a survey of Microsoft in 2007[1], over 60% of 684 software engineers had to deal with concurrency issues on a monthly basis. Concurrency bugs usually takes several days to detect and debug, and thus most of 684 engineers would welcome additional help on solving concurrency issues. Another survey also showed that about 73% of the examined non-deadlock concurrency bugs were not fixed by simply adding or changing locks, and many of the fixes were not correct at the first try[2], which implies the difficulties and costs software engineers face during solving non-deadlock concurrency problems.

In this paper, we will focus on data race, one main category of non-deadlock concurrency problem. Data race only happens in some certain program states, which makes its localization and fix very expensive in industrial practice because there are too many states in a real product's program to check one by one. To solve this problem, we propose a novel static method for data race detection based on dependence analysis. While a lot of existing static methods suffer from false positives, the proposed method theoretically has few false positives. By focusing on the dependence among data accesses instead of data accesses only, the proposed method can more effectively exclude program states (each corresponds to a combination of data accesses) that are irrelevant to data races so that the false positive rate can be kept in a manageable range. Regarding the target of detection, we choose source code, the relatively downstream output of software development. The main reason is that considering the fact that data races can be introduced into the program in both design and implementation, we want the proposed method to cover different phases of software development as much as possible.

In the rest part of this paper, Section 2 gives a clear definition of data race. Section 3 introduces some related work on data race detection. Section 4 describes the proposed method for data race detection. Section 5 shows a prototype tool we built based on the method introduced in Section 4. Section 6 describes the how the proposed method is evaluated with production-level source code. Section 7 discusses about the proposed method based on the evaluation. Section 8 gives a conclusion.

2. Data Race

Data race has been an important research topic since 1990s[3]. One widely used definition is:

definition 1. *When multiple threads*[1] *that concurrently run without any synchronization access the same location of memory, and at least one access is write, data race occurs.*

In this definition, we can see data race is a bug due to unsynchronized threads concurrently accessing the same memory. Such data access will lead to undefined behavior, and the actual ordering of accesses on the shared memory is unknown because it is not specified by source code. This unknown ordering may finally lead to unintended output. However, there is still a risk of bug when synchronized threads concurrently accessing the same memory. An improper synchronization can also lead to unexpected execution orderings of threads, which makes the final program output unintended. In fact, according to one characteristic study of concurrency bugs, around one third of the examined non-deadlock concurrency bugs are caused by violation to programmers' order intentions, which may not be easily expressed via synchronization primitives like locks and transactional memories[2]. Because both improper synchronization and unsynchronization can lead to an unintended thread execution ordering which finally leads to an unintended program output, we think it is reasonable to treat improper synchronizations the same as non-synchronization in data race detection. Therefore, "without any synchronization" is actually interpreted as "without any proper synchronization" in this paper.

One typical case of data race is shown in Figure 1. Thread A and thread B execute concurrently. One calculation in thread B depends on two readings of variable Var, which is updated in thread A. If the synchronization of thread A and B is improper or does not exist at all, it is possible for thread A to update Var right between two read accesses of thread B, which may finally lead to an incorrect program output.

There is another concept called race condition. Although race condition is used interchangeably with data race in some researches, it is more often considered as a more general concept. In this paper, we adopt the more general definition as follows,

[1]Thread, task and interrupt service routine (ISR) are used interchangeably to represent the general unit of concurrency. It is true that different operation systems or programming languages may have different names for the unit of concurrency, but we are only interested in the general concurrency issues in this paper.

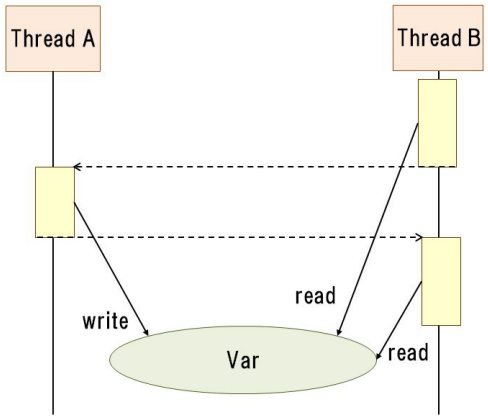


Figure 1. Data race on single variable

definition 2. *When the correctness of a program's output is affected by the potential nondeterminism in timing or ordering of threads' execution, race condition occurs.*

One interesting case is shown in Figure 2. Unlike the case in Figure 1, this race problem happens on two variables instead of one. More specifically, the two variables, $Var1$ and $Var2$ are actually a redundant variable pair, which means $Var1$ and $Var2$ stands for exactly the same thing, and they both exist due to the redundant structure of software which is applied in a lot of critical systems for safety concerns. Thread B reads $Var1$ and $Var2$, then compares their value to check if there is a malfunction in the system. Thread A runs concurrently with Thread B and updates both $Var1$ and $Var2$. If Thread A updates the values of $Var1$ and $Var2$ when Thread B already read $Var1$ but have not read $Var2$ yet, the comparison result of $Var1$ and $Var2$ may be wrong and finally lead to a system failure. This kind of race problem actually happen during development and is sometimes even more difficult to find and fix.

According to definition 1 and definition 2, the race-related problem shown in Figure 2 is actually a race condition instead of data race because Thread A and B are not accessing the same location of memory. However, this problem has almost the same mechanism as data race except there are two variables involved. Therefore, we think it is reasonable to improve definition 1 by changing "the same location of memory" to "memory storing the same information", which will make definition 1 generalize similar problems better. In the rest of this paper, we will use definition 3 instead of definition 1 for data race and consider the problem shown in Figure 2 as a data race.

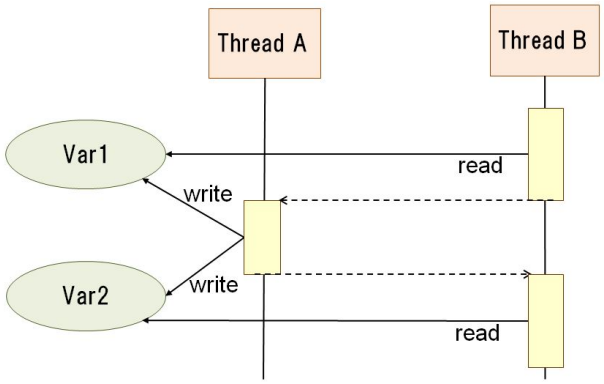


Figure 2. Data race on a redundant variable pair

definition 3. *When multiple threads that concurrently run without any synchronization access memory storing the same information, and at least one access is write, data race occurs.*

Without introducing definition 3, it is also possible to consider a race on a redundant variable pair as a combination of two data races, each of which includes exactly one write access and one read access of one variable, but we chose not to do so due to reduce false positives (details can be found in the 5th paragraph of Section 4.2).

3. Related Work

There are many researches done to solve the data race. The simplest method is Mutual Exclusion Locking discipline, which requires a lock be held on every access to a shared variable. Appropriate use of mutual exclusion locks can eliminate atomicity violations of any data, therefore prevent data races. Another simple method is to use buffer for shared variables, which allows a program to has a copy of certain variable at certain time and can use it at any time necessary. However, when we have a extremely large concurrent program, both methods lead to a very high maintenance cost. In addition, for programs of systems with limited resources (e.g. automotive control systems), it is not practical to use mutual exclusion locks or buffer for all variable shared between threads.

Type System can be used to prevent data race. The basic idea is to make a programming language that cannot realize data race. A lot of early researches on such type systems use locking discipline while some other ideas for the type system construction have also emerged. Recent work can be found in [4][5]. The main problem in type-based solution is the high cost of switching to a new language and the expression limitations in the current race-free languages.

Dynamic Analysis analyzes the runtime behavior of program to find potential data race. Most dynamic race detection tools do lockset-based analysis[6] or happens-before analysis[7] or both[8]. Current dynamic methods for data race detection usually suffer from two main problems. One is that such methods usually require instrumentation in the source code, which can dramatically increase the development cost sometimes. The other problem is that dynamic method can only find data race in the execution paths taken in the detection because the whole analysis is based on runtime behaviors. This makes dynamic detection unpractical for critical systems that run for a long time. For some programs with much longer runtime than the testing time (e.g. one automotive control program can run for thousands of hours in millions of cars), it is much more likely that a data race appearing in actual execution was missed in dynamic testing.

Static Analysis is used for race detection as well. Static method is usually very flexible, therefore the performance varies from one to the other. Among all static methods, some recent flow-sensitive methods showed potentials for industrial use because they have a combination of soundness and completeness. RacerX[9] is one good example. It intentionally sacrificed some soundness for better completeness (fewer false positives) by applying several heuristics in the detection. When elimination of false negative is not required, false positives can be controlled to a low level with this method. Inamori and Yamada proposed a static method for detection of data race caused by improper use of interrupts in automotive systems [10]. It searches for a data access pattern which ensures there is no false negative, and runs several checks to reduce false positives. In the target data access pattern, two read accesses with low priority and one write access with high priority happen on the same variable concurrently. Since this method uses model checking to eliminate false positives in the end, it suffers from the state explosion problem, which makes this method not feasible for large programs.

Model Checking is a method that can find all data race including those hard-to-reproduce ones. Unlike dynamic methods, model checking is able to explore all theoretically possible execution paths by analyze a model of the program instead of the program itself. However, model checking suffers from the state explosion problem. When the size of target program is very large, there are too many possible execution paths to check, which takes a lot of time and memory. In our previous work, we tried to apply a program slic-

ing before model checking to solve the state explosion problem[11]. For some complex programs, the state explosion still appears after slicing, so we also tried a divide and conquer strategy (divided the problem into small parts, do model checking for each and the interfaces between them). This strategy was proved effective, but another question comes up as how to determine which part to check first. When we know there is a bug, it will be possible to find it earlier if we can check the part that is most likely to contain bugs at first. However, this requires some techniques of bug prediction or vulnerability analysis.

4. Detection Method

In this section, we propose a novel method for data race detection based on dependence analysis, which statically analyze source code and point out data race in one part of the code that the user is most interested in. The proposed method is consisted of two main steps. The first step is extraction of variable dependence tree, and the second one is race search.

4.1. Extraction of Variable Dependence Tree

The first step of our proposed method, extraction of Variable Dependence Tree, is a technique originally used for program slicing, which was introduced in our previous work[11]. Program slicing is a popular technique used for locating bugs in large programs. This technique allows the user to focus on the code of interest, which can be the code that is most likely to have bugs, or code related with some observed malfunction. Specifically, it means to take a slice of the program containing all statements that may affect a specific variable at a specific point in the program. It was originally proposed by Weiser [12] and have been quite a hot topic in researches since then[13].

In our proposed method, instead of taking a slice of program related to one variable the user is interested in, we take a slice of variable dependence related to one specific variable the user is interested in and represent it in a tree called Variable Dependence Tree (VDT). Although in [11], VDT is ultimately used for program slicing, the actual program slicing is not necessary for data race detection in this paper because the detection is purely an analysis on VDT. In the rest part of this subsection, we will briefly introduce the concept of VDT and several other concepts it is built on.

The study on dependence representation of program is not a new research field. In 1987, Program Dependence Graph (PDG) was first introduced by Ferrante, Ottenstein and Warren to make explicit both the data and control dependences for each operation in a program. Specifically, PDG is a directed graph. In this graph, nodes are statements and predicates; edges incident to a node represent both the data values on which the node's operations depend and the control conditions on which the execution of the operations depends[14]. Based on PDG, Horwitz, Reps and Binkley proposed System Dependence Graph (SDG) for interprocedural slicing[15]. SDG extends PDG to incorporate sets of procedures rather than just monolithic programs and can be considered as a set of PDGs.

Variable Dependence Graph (VDG)[11] is a concept we developed based on concurrent version of SDG. VDG is a graph derived from SDG in which each node no longer represents a statement. A node in VDG represents a variable or a constant in a specific statement with a specific execution path or a specific control statement (e.g. if statement, while statement) with a specific execution path. For example, the same variable appearing in two different statements is represented by two nodes (one for each statement). The same variable in the same statement that has two possible execution paths is also represented by two nodes (one for each execution path). The label of each node is the name of the variable or the control statement this node represents. Besides the name shown in label, each node also carries trace information of the variable or control statement it represents. An edge in VDG represents data dependence or control dependence between two nodes. The method to extract a VDG from a program is discussed in [11].

Variable Dependence Tree (VDT) is a VDG pruned into a tree. There are two types of VDT, Goal VDT and Start VDT. We will only use Goal VDT in this paper. A goal VDT is a directed tree $T(V, E)$ rooted at node s, where V is the set of nodes, and E is the set of edges. The set of children of node v is denoted by $C(v)$. $\forall v \in V$ depends on $\forall c \in C(v)$. The edge between node v and its child $c \in C(v)$ points from c to v.

Since the transformation from VDG to VDT is already introduced in [11], we will only briefly describe the basic idea about it in this paper. To prune a VDG into a VDT, we search for nodes depending on the same node in VDG. The set of nodes depending on an arbitrary node x is denoted by D. For all edges pointing from x to $d \in D$ except the leftmost one, remove the edge, and add a new leaf node x'_i as the child of $d \in D$. The new leaf node x'_i means that the dependence from x has already appeared elsewhere in this tree and therefore can be ignored, where the subscript i is just an index to prevent redundant node labels. A simple example is shown in Figure 3 and Figure 4. Fig-

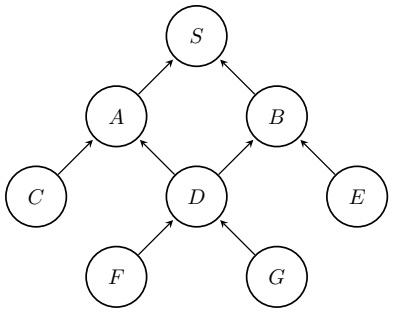


Figure 3. VDG extracted from variable S

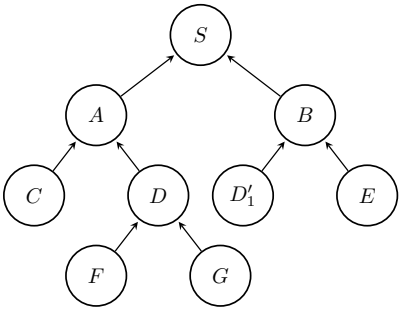


Figure 4. VDT transformed from VDG in Figure 3

ure 3 shows a VDG extracted from variable S, where all nodes represents variables. As we can see, node A and node B both depends on node D. To prune this VDG into VDT, we keep the leftmost edge (A, D) and remove edge (B, D). Then we add a new leaf node D'_1 as a child of variable B. In this way, the VDG in Figure 3 is pruned into the VDT in Figure 4. When the resources (e.g. time, memory) is limited, a partial extraction of VDT is also possible, which is introduced in [11].

By adopting VDT, we are able to eliminate all artificial orderings in the program and make potential parallelism explicit. Since variables are represented as nodes, it is more efficient to use VDT to detect data race happening on specific variable(s). In the real-life application, it is usually efficient and effective to extract a VDT from a certain error flag or output variable the user is interested in. If the user does not have information about such a variable and just wants to reach a full coverage of the dependence in the program, which is not recommended because of the efficiency concerns, the user can extract a VDT from each final output of the program.

4.2. Race Search

In the algorithm of race search, a concept called interference dependence is used. The early definition and application of interference dependence can be found in Krinke's work[16]. According to Krinke, interference dependence occurs when a variable is defined in one thread and used in a parallel executing thread. Specifically, in a VDT, if an edge connects two nodes with different thread information, this edge represents an interference dependence. In [17], Krinke also showed how to analyze interference dependence efficiently.

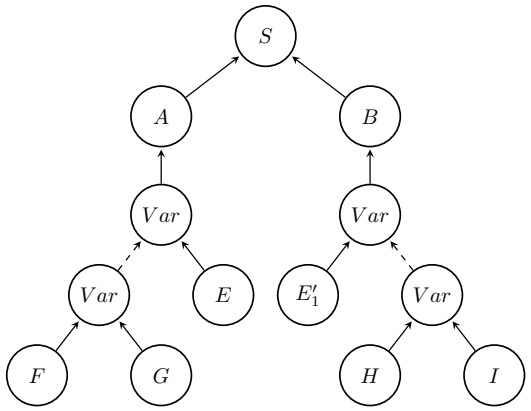


Figure 5. The pattern of data race candidate

In our proposed method, we find data race candidates based on dependence analysis. Specifically, if multiple threads affects the memory storing the same information, this situation is considered as a data race candidate. Then data races are extracted by filtering these candidates through priority and reality checks. After adopting the concept of VDT and interference dependence, the situation considered as data race candidate can be interpreted to "In a VDT, ① there are multiple interference dependences, ② variable(s) storing the same information appear on the descendant side of each interference dependences." Figure 5 shows an example of such situation in VDT. There are two interference dependences (drawn as dotted arrows), which means the value of S is affected by multiple threads. Variable Var appears on the descendant side (bottom side in the tree) of each interference dependence, which means that value of S depends on two readings of Var while these two readings are obtained in a thread different from where S is defined. When the thread where S is defined and the thread(s) where two readings of Var are done execute concurrently, the value of S may be different for a different execution ordering, thus an unintended value becomes possible, which is exactly the data access problem we showed in Figure 1. The details on priority and reality checks will be introduced later in this section.

It is important to notice that the variables storing the same information on the descendant side of interfer-

```
1  int x1,x2,a,b,c;
2  int DI0;
3  int errflg;
4  void main(void){
5      x1=a+1;
6      x2=b+1;
7      if (x1!=x2){
8          errflg=1;
9      }
10 }
11 void calc(void){
12     c=DI0;
13     a=c*2;
14     b=c+c;
15 }
```

Figure 6. Sample code for data race due to indirect influences of data access

ence dependence (Var in the case of Figure 5) are not necessarily the same variable, they can also be a redundant variable pair (two variables storing the same information because of redundant structure), which corresponds to the case in Figure 2. Of course, in order to recognize redundant variable pairs, the user has to collect additional information about redundant variables before data race detection. Fortunately, such information can usually be extracted from design documents.

One interesting fact is that even when the user cannot recognize redundant variable pairs, the data race on redundant variable pairs can still be partially detected. For the case shown in Figure 2, it is possible that $Var1$ and $Var2$ finally depend on the same variable in VDT. For instance, this variable can be some raw sensor data, and $Var1$ and $Var2$ are calculated based on it by different calculation methods. We did not confirm if such case actually exist in evaluation, but it is theoretically possible. Figure 6 shows a sample code of this case. Main() and calc() are two functions running concurrently. a and b are redundant variables used to calculate $x1$ and $x2$ respectively in main(). If $x1$ and $x2$ are not equal, an error flag is triggered. In calc(), a and b are calculated based on variable c (raw sensor data) with different methods. Indeed, data race happens on the redundant variable pair a and b. However, the dependence on the same variable (variable c in the sample code) makes such data race no different from those on single variable. Thus it can be successfully detected even when we do not recognize a and b as a redundant variable pair. Nevertheless, the information on redundant variable pairs is still helpful because it makes the data race detectable at a higher level in VDT.

When locating data race candidates as introduced above, a kind of data race is intentionally excluded in detection. No matter in the typical definition of data race (definition 1) or the extended definition (definition 3) used in this paper, data race happens when only two concurrent accesses of the same variable exist, one of which writes and the other reads. In fact, data race on a redundant variable pair can also be considered as a combination of two such "1 read & 1 write" data races. However, all "1 read & 1 write" data races are excluded in detection for a lower false positive rate. In a large concurrent program such as an engine control program, it's normal to see many such "1 read & 1 write" data races, but they rarely cause real problems in practice since they are easy to find and fix by conventional methods.

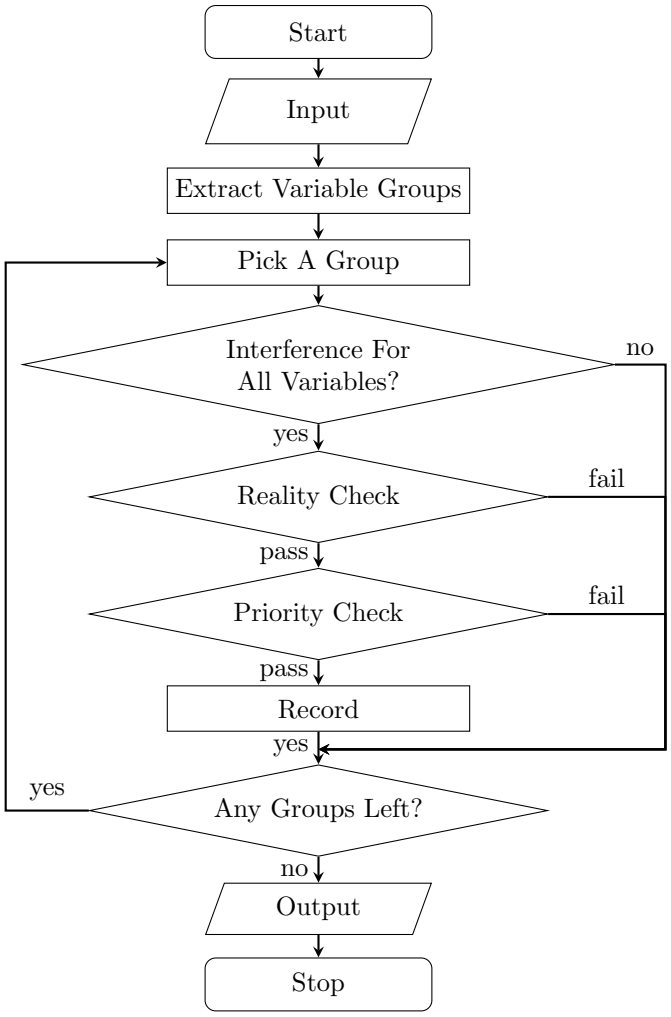


Figure 7. Flowchart of race risk search

The algorithm of potential race search is shown in the flow chart in Figure 7. The input block takes a VDT, information on thread execution priority as well

as redundant variable, and reality information as input. Reality information refers to the information about if some threads can run concurrently in reality. This information can include synchronization information, hardware limitations, limitations of the runtime environment and so on. Then "Extract Variable Groups" process traverses the VDT to extract nodes representing access to the variable that store the same information and divides them into groups by variable (e.g. Var in Figure 5). After extracting variable groups, "Pick A Group" process randomly picks one group for further analysis. Decision block "Interference For All Variables?" checks the paths from picked nodes to a closest common ancestor. If there are interference dependences in multiple paths, a data race candidate is found and the algorithm goes to decision block "Reality Check". If not, the algorithm jumps to "Any Groups Left?" block.

"Reality Check" and "Priority Check" block reduce false positives by checking if the threads involved in data race candidates can actually run concurrently without synchronizations during runtime. "Reality Check" checks concurrent execution limitations that are not determined in the program under test (e.g. external synchronizations, limitations of hardware platform or runtime environment). In a program with optimized resource consumption and complexity, it is normal that only some threads can run in a specific scenario and some threads can never do even if it is not explicitly set in the program. For example, a thread for situation when car speed is equal to 0 and a thread for situation when car speed is higher than 100mph can never run concurrently. "Priority Check" checks the execution priority of threads involved in the data race candidates. Given the OS or hardware platform where the program runs, some threads are not allowed to run concurrently due to execution priorities. For example, in a lot microcontrollers, threads with the same priority are executed sequentially instead concurrently, therefore there can never be data race between them. It should be noticed that rules in both checks are flexible and should be adjusted based on the program under test or the goal of testing.

When both checks pass, the algorithm will go to "Record" block. Otherwise, the algorithm jumps to "Any Groups Left?" block. "Record" block records candidates that passed the checks as potential data races. Decision block "Any Groups Left" checks if there is any extracted variable group left. if there is any, the algorithm goes back to "Pick A Group" block to pick up a new group and search for potential data race. If not, the algorithm ends.

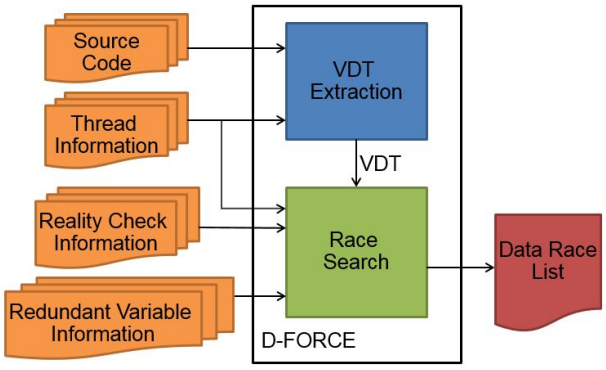


Figure 8. Overview of prototype tool

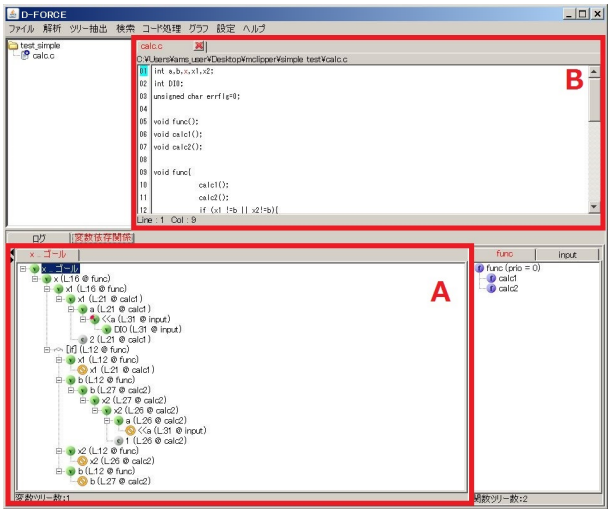


Figure 9. Screen capture of D-FORCE

5. Implementation

We built a prototype tool called D-FORCE in Java for the proposed detection method introduced in Section 4. As shown in Figure 8, it takes inputs including source code, redundant variable information, thread information (priority and entry point of each thread) and reality check information (which threads may run concurrently) to generate data race list. For an efficient implementation, we assumed the target programs are only written in C language without loss of generality. One screen capture of the D-FORCE is shown in Figure 9. Since it is also used in our other researches, only area A and B are used for data race detection. Area A shows the extracted VDT, and area B is a source code reference window, which shows the corresponding source code of selected node in VDT so that the user gets a visual confirmation of actual source code.

Figure 10 shows a sample source code. If we extract

a VDT from variable x, it will be visualized in area A as shown in Figure 11. This is a standard result of the first step of our proposed method, extraction of VDT. Each node has an icon followed by a label. The icon shows data type. If the node represents a variable, a green round icon with a "v" inside is shown, just like the one before label "x (L:16@func)". If the node represents a constant, a gray icon with a "c" inside is shown, just like the one before label "2 (L:21@calc1)". If the node represents a control statement, a diamond shape icon is shown, just like the one before label "[if] (L:12@func)". If a node represents anything that already appeared somewhere else in the VDT (such as the D_1' in Figure 4), the icon is a yellow stop sign, just like the one before "x1 (L21@calc1)". Label gives more details about the node. For example, label "x (L:16@func)" means the node represents variable x on line 16 of function $func$. A label "[if] (L:12@func)" means the node represents the if statement on line 12 of function $func$. When a variable node is at the write side of an interference dependence, the variable icon is marked with a red arrow and a "≪" mark is added to the beginning of the label.

The result of potential race risk search on the VDT is also shown in Figure 4. At the root node, the "(risk=1)" shows that there is one potential data race detected. A "=ic1=" mark is added at the beginning of the label "x (L:16@func)", which indicates that this variable is the common ancestor related with the first detected data race. "=rd1=" and "=wo1=" indicates the corresponding variable are one side of the interference dependence related with the first detected data race. When such a result is available, D-FORCE also automatically generates a file with all formation about locations of interference dependences and the common ancestor, based on which the user can understand how the potential data race can happen and trace back to the exact problematic statements in the source code.

6. Evaluation

The feasibility of the proposed method in industrial use is confirmed in our evaluation. We ran D-FORCE on obsolete version code of two different powertrain ECUs (Electrical Control Unit) developed in Hitachi Automotive Systems. 100% (4 out of 4) data races actually encountered during development were successfully detected while the number of false positives was maintained in a manageable range.

Both sets of source code are written in C and have more than 400k LLOC (Logical Lines of Code). D-FORCE was executed in Windows 7 running on a PC with a Intel Core i7 870 CPU (2.93GHz) and 16GB

```
1  int a,b,c,x,x1,x2;
2  int DI0;
3  unsigned char errflg=0;
4
5  void calc1(void);
6  void calc2(void);
7  void input(void);
8
9  void func(){
10         calc1();
11         calc2();
12         if (x1 !=c || x2!=c){
13                 errflg=1;
14         }
15         else {
16                 x=x1;
17         }
18 }
19
20 void calc1(){
21         x1=a*2;
22         b=x1;
23 }
24
25 void calc2(){
26         x2=a<<1;
27         b=x2;
28 }
29
30 void input(){
31         a=DI0;
32 }
```

Figure 10. Sample source code

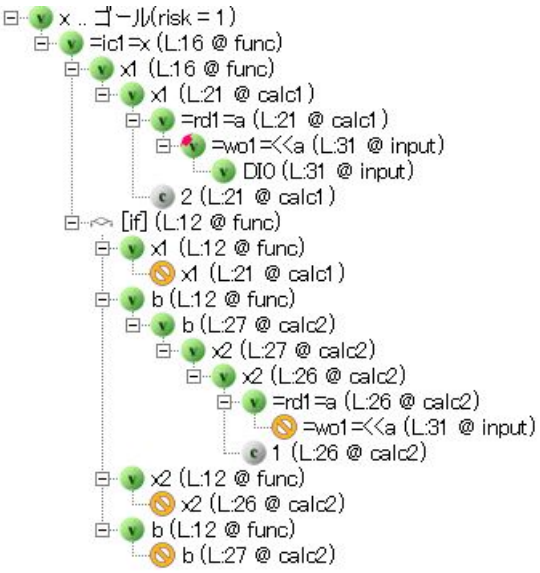


Figure 11. One example of VDT visualization in D-FORCE

RAM. The whole analysis finished on both sets of source code within 15 minutes. Redundant variable information and thread information were obtained manually by reviewing design documents. Since we noticed that some false positives appeared in initialization threads, we manually excluded all initialization threads during data race detection, which can essentially be considered as a kind of reality check.

Among 4 known data races, 3 of them have the same data access pattern, which is exactly the typical case shown in Figure 1. The other one is a data race on a redundant variable pair as shown in Figure 2. All 4 data races were confirmed as bugs leading to system failure in previous development. According to our observation, false positives still exist, so a more accurate measurement on completeness is necessary in the future work, after which we can decide the direction for further improvements of the proposed method.

7. Discussion

In the proposed method, we did not use synchronization information to find data race candidates because we wanted to detect data race due to both "no synchronization" and "wrong synchronization". Obviously, synchronization is partially used in the priority check to reduce false positives, but there are still some synchronization informations that can be helpful but are currently not used. For instance, information on interrupt disables are used in the work of Inamori and Yamada[10] to reduce false positives while it is not used in our method at all. Such underutilization of synchronization information has pointed out one possible direction for us to further reduce false positives.

Dependence analysis in the proposed method allows data race detection to focus on interference in dependence instead of accesses on certain variables. This enables detection of data race resulted from indirect influences of data access. The code in Figure 6 is also an example for this feature. When variable c is updated in one thread, a concurrently running thread does not necessarily have to read exactly variable c multiple times to cause a data race. In the thread where c is updated, variable a and b are defined based on c. A concurrently running thread can also cause a data race by reading a and b and using their values together for some other calculation (a and b might be calculated from different c values). In this case, there are at most two accesses on variables shared among threads (a and b), so it is very hard to detect by conventional methods without generating many positives. In our proposed method, this is not a problem due to dependence analysis.

```
1 int a;
2 void calc(int *pt);
3 void main(void);
4     a=0;
5     calc(&a);
6 }
7 void calc(int *pt){
8     *pt=3  ;
9 }
```

Figure 12. Sample code for data access through pointer

While a lot of data race detection methods have trouble analyzing program with pointers, dependence analysis in our proposed method helps to find out which piece of memory is accessed when pointers are involved in the data access. One example is the sample code in Figure 12. Main() and calc() are two function that run concurrently. They actually access the same variable a, but the write access in calc() is done through a pointer passed to it. Since pointers are also included the dependence analysis, it is very easy to figure out the two threads are accessing the same variable a in our proposed method.

It is also important to notice that our proposed method is naturally compatible with programs containing parallel threads even if we only mentioned concurrent threads so far. For the proposed method, the only difference brought by parallel threads is that the data accesses can not only be concurrent, but also simultaneous (physically at the same time). This difference cannot affect how we identify data race candidates or the correctness of checks applied to filter them, so the proposed method can also detect data races in program with parallel threads as long as we have information about which threads run in parallel.

When a data race happens, the program's output may still be correct after certain processing in the program. Such situations are relatively rare, but still possible. The data race that does not affect the correctness of a program's output is referred to as "benign data race"[18]. Our proposed method cannot completely distinguish benign data race, which seems like a problem. However, recent researches showed that it is impossible to guarantee that a data race never affects the correctness of a program as long as the compiler or hardware can be changed[19]. Therefore, we chose to include "benign data race" in the detection results.

8. Conclusion

In this paper, we proposed a novel method for data race detection based on dependence analysis. A pro-

totype tool is build in Java to realize the proposed method. By running the prototype tool on production-level code of two real industrial products, the proposed method proved to be feasible for data race detection in industrial practice. However, more empirical validation is still necessary in the future. Besides the programs we used for evaluation in this paper, the proposed method should also be tested on more programs for a more accurate measurement of soundness and completeness, after which we may be able to make necessary improvements (e.g. further reducing false positives by a better utilization of synchronization information such as interrupt disable and mutex).

References

[1] P. Godefroid and N. Nagappan, "Concurrency at microsoft: An exploratory survey," in *CAV Workshop on Exploiting Concurrency Efficiently and Correctly*, 2008.

[2] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics," in *ACM Sigplan Notices*, vol. 43-3, pp. 329–339, ACM, 2008.

[3] R. H. Netzer and B. P. Miller, "What are race conditions?: Some issues and formalizations," *ACM Letters on Programming Languages and Systems (LOPLAS)*, vol. 1, no. 1, pp. 74–88, 1992.

[4] E. Castegren and T. Wrigstad, "Reference capabilities for concurrency control," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 56, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[5] C. S. Gordon, M. J. Parkinson, J. Parsons, A. Bromfield, and J. Duffy, "Uniqueness and reference immutability for safe parallelism," in *ACM SIGPLAN Notices*, vol. 47, pp. 21–40, ACM, 2012.

[6] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson, "Eraser: A dynamic data race detector for multithreaded programs," *ACM Transactions on Computer Systems (TOCS)*, vol. 15, no. 4, pp. 391–411, 1997.

[7] A. Dinning and E. Schonberg, "Detecting access anomalies in programs with critical sections," in *ACM SIGPLAN Notices*, vol. 26, pp. 85–96, ACM, 1991.

[8] R. O'Callahan and J.-D. Choi, "Hybrid dynamic data race detection," in *ACM SIGPLAN Notices*, vol. 38, pp. 167–178, ACM, 2003.

[9] D. Engler and K. Ashcraft, "Racerx: effective, static detection of race conditions and deadlocks," in *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 237–252, ACM, 2003.

[10] Y. Inamori and N. Yamada, "Static interrupt-race detection method for c program in vehicle," *IPSJ Transactions on Consumer Devices&Systems (CDS)*, vol. 3, no. 3, pp. 76–84, 2013.

[11] M. Matsubara, K. Sakurai, F. Narisawa, M. Enshoiwa, Y. Yamane, and H. Yamanaka, "Model checking with program slicing based on variable dependence graph," in *First International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2012)*, p. 88, 2012.

[12] M. Weiser, "Program slicing," in *Proceedings of the 5th international conference on Software engineering*, pp. 439–449, IEEE Press, 1981.

[13] B. Xu, J. Qian, X. Zhang, Z. Wu, and L. Chen, "A brief survey of program slicing," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 2, pp. 1–36, 2005.

[14] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 9, no. 3, pp. 319–349, 1987.

[15] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 1, pp. 26–60, 1990.

[16] J. Krinke, "Static slicing of threaded programs," *ACM Sigplan Notices*, vol. 33, no. 7, pp. 35–42, 1998.

[17] J. Krinke, "Context-sensitive slicing of concurrent programs," *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 5, pp. 178–187, 2003.

[18] S. Narayanasamy, Z. Wang, J. Tigani, A. Edwards, and B. Calder, "Automatically classifying benign and harmful data races using replay analysis," in *ACM SIGPLAN Notices*, vol. 42, pp. 22–31, ACM, 2007.

[19] H.-J. Boehm, "How to miscompile programs with" benign" data races.," in *HotPar*, 2011.

N-gram IDF を利用したソースコード内の特徴的部分抽出手法

小林 勇揮
京都工芸繊維大学 大学院工芸科学研究科 情報工学専攻
y-kobayashi@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

従来の大域的な語の重み付け手法である *IDF*（*Inverse Document Frequency*）には，単語 *N-gram* に対して適用できない欠点があった．しかし，近年の研究により，*IDF* を単語 *N-gram* に対して適用する手法が提案された．本研究では，この *N-gram IDF* をソースコードに対して適用し，ソースコード中の特徴的部分の抽出に応用できると考えた．具体的には，局所的重み付けである *TF*（*Term Frequency*）と *N-gram IDF* を利用した語の重み付け手法である $TF\text{-}IDF_{N-gram}$ を用いて，ソースコードごとの特徴語の抽出を行った．そして，その特徴語の行ごとの出現頻度を求めて，ソースコード中の特徴的部分の抽出を行った．まず，サンプルプログラムを用いて特徴語抽出の評価実験を行い，ソースコードにおいても特徴語をある程度抽出できることを示した．次に，*Apache Ant* の公開されているソースコードを用いて特徴的部分抽出を行い，またソースコードの変更による特徴的部分の変化についても調べた．その結果，ソースコードから特徴的部分の抽出をすることができた．また，その抽出した特徴的部分は，ソースコードの変更によってもソースコード全体の相対位置の変化が少ないことを示した．

1 はじめに

語の重み付け手法（term weighting scheme）はテキスト解析において重要な技術であり，情報検索や文書クラスタリングなど幅広い分野で利用されている．そこで，この語の重み付け手法をソースコードに対して適用し，ソースコード中の特徴的部分を特定することによって，開発者がソースコードに対してリファクタリングやメンテナンスをする際の役に立つのではないかと考えた．また語の重み付け手法では IDF（Inverse Document Frequency）という大域的な語の重み付け手法が利用されるが，この IDF には単語 N-gram に対して適用できない欠点があった．しかし，近年の研究により，この IDF を単語 N-gram に対して適用する手法が提案された [14]．

本研究では，この N-gram IDF を利用してソースコード中の特徴的部分を抽出する方法を提案する．この研究の動機としては，今までの IDF では一語の単語に対してしか語の重み付けができず，単語 N-gram に対して重み付けができなかった．IDF をソースコードに対して適用するときに，一語の場合はソースコード中でその語がどのような処理をするときに特徴的なのかが分かりづらい問題がある．例えば，ある語「name」の重みづけが大きくなったときに，その語が「int name」や「name = 0」など，宣言されたときが重要なのか，代入があったときが重要なのかが分からなかった．しかし，この N-gram IDF を利用すると，それを区別することができるため，ソースコードに対して利用すると有効性が得られると考えた．本研究の実施にあたっては，先行研究でのアルゴリズムの公開されている実装[1]を用いた．

2 準備

2.1 語の重み付け手法

語の重み付け手法（term weighting scheme）はテキスト解析において重要な技術である．その中で，代表的なものとして TF-IDF（Term Frequency-Inverse Document Frequency）がある．TF-IDF は，文書中に出現する特定の単語がどのくらい特徴的であるかを識別するための指標で，情報検索 [13]，文書クラスタリング [3]，特徴語抽出 [6]，映像中のオブジェクトマッチング [12] など，幅

[1] https://github.com/iwnsew/ngweight

広いアプリケーションで利用することができる．また，複数の観点 [2] [11] [7] から理論的説明が与えられており，多くの人が TF-IDF を利用する根拠となっている．本研究では，この語の重み付け手法をソースコードに対して利用することにした．次の節で，TF-IDF について詳しく説明していく．

2.2 TF-IDF

TF-IDF のような語の重み付け手法は，一般的に，局所的重み付けと大域的重み付けの二つの要素からなる．局所的重み付けは，ある文書に対する語の出現頻度から計算され，対象とする文書によって重みは変化する．一方，大域的重み付けは，文書集合全体における語の文書頻度から計算され，語の重みは対象とする文書によらず一定である．TF-IDF は具体的には，以下の式（1）により語 t の文書 $d \in D$ における重みを計算する．

$$TF\text{-}IDF(t,d) = tf(t,d) \cdot \log \frac{|D|}{df(t)} \tag{1}$$

ここで，$tf(t,d)$ は文書 d における語 t の出現頻度，$df(t)$ は文書集合 D における t の文書頻度，$|D|$ は D の文書数である．

この式（1）の中で，局所的重み付けは以下の式（2）で表される．

$$TF(t,d) = tf(t,d) \tag{2}$$

また，大域的重み付けは以下の式（3）で表される．

$$IDF(t) = \log \frac{|D|}{df(t)} \tag{3}$$

特に，大域的重み付けである IDF [8] は様々な語の重み付け手法で採用されている．IDF が様々な語の重み付け手法として採用されている理由として，式（3）のような簡潔さとロバスト性が挙げられる．実際に，IDF は様々な文献 [2] [11] [9] において理論的にロバストであることが示されている．

2.3 N-gram IDF

従来の IDF の欠点として，単語 N-gram に対して適用できないことがあった．単語 N-gram に対する IDF は，その連続する単語のつながりが不自然であるほど大きい重み付けをしてしまっていた．例えば，「Osaka University」と「Osaka be」では単語のつながりが不自然な「Osaka be」の方が文書に出現する割合が低くなるため，「Osaka be」の方が大きい重み付けをしてしまう．しかし，大阪大学大学院情報科学研究科の白川真澄らによる研究 [14] により，単語 N-gram に対しても IDF を適用することができることが発見された．この手法は，文字列が出現する Web ページをシャノン・ファノ符号によって表現したとき，空文字からの情報距離が対象の文字列の IDF となることを利用している．具体的には，以下の式（4）によって単語 N-gram g に対する IDF を計算する．

$$IDF_{N-gram}(g) = \log \frac{|D| \cdot df(g)}{df(\theta(g))^2} \tag{4}$$

ここで，$|D|$ は D の文書数，$df(g)$ は文書集合 D における g の文書頻度，$\theta(g)$ は g を構成する各単語の論理積，$df(\theta(g))$ は文書集合 D における $\theta(g)$ を満たす文書数である．また，各単語の論理積とは，文書集合において g を構成する各単語が含まれている文書である．

3 ソースコードに対する N-gram IDF の適用

ここではソースコードに対して N-gram IDF を使用するにあたって，どのような前処理をしたり，注意事項があるのか挙げる．

また，ソースコードに対してのみ N-gram IDF を使用したいので，ソースコード中のコメントがあるならそれを取り除き，その取り除いたソースコードに対し N-gram IDF を使用することにする．

3.1 単語の分割

ここではソースコードの単語の分け方について説明する．英語などのテキストは基本的に空文字によって単語を分割する．しかし，ソースコードを単語に分けるとき，プログラマや開発環境によって空文字の有無が発生する．例えば，「 a = b + c 」と「 a=b+c 」がある．このとき，前者は「a」「=」「b」「+」「c」の 5 文字で認識し，問題はない．一方，後者は空文字がないため，「a=b+c」の 1 文字で認識してしまうと問題が発生する．この問題を解決するために，ソースコードを単語に分ける際の規則は以下のようにする．実験では，C と Java のプログラミング言語に対して単語分けを行った．

- 基本的にスペース区切りで単語を分ける．
- 以下の記号は一単語としてみなす．
  ! ? ” ’ # $ % & | ( ) { } [ ] = < > + - * / \ ~ ^ @ : ; , .

これにより，ソースコードを単語に分けていく．

さらに詳しく説明をするために，この規則を適用してソースコードを単語に分けるときの一つの例を挙げる．例えば，次のようなソースコードがあるとする．

例）`name.name_get()=1+2;`

これを前述の規則に適用すると,「name」「.」「name_get」「(」「)」「=」「1」「+」「2」「;」の 10 単語に分かれる．ここで注意すべきところは「name_get」が 3 語に分かれないことである．これはソースコードにおいて記号の中でも「_」が変数の名前の一部として利用されるためである．特にこの「_」が変数の名前の一部として利用されていることが多いため，この記号を一語として見なさないことにした．また，Java などの言語では変数の名前の一部に「_」以外の記号を使うことがある．例えば,「$」や「@」,「%」などが挙げられる．しかし，そのような場合は先頭で始まったり，特徴的な使い方をされることが多いため 1 語として見なすことにした．

3.2 ソースコード中の特徴語判定方法

本研究では，ソースコード中の特徴的部分を発見するためにソースコード中の特徴語を判定する．そのために，語の大域的重み付けである N-gram IDF と語の局所的重み付けである TF（Term Frequency）を利用した $TF\text{-}IDF_{N-gram}$ を利用する．具体的には，式（2）と式（4）より，N-gram g のソースコード $d \in D$ における重みは以下の式（5）で表される．

$$TF\text{-}IDF_{N-gram}(g,d) = tf(g,d) \cdot \log \frac{|D| \cdot df(g)}{df(\theta(g))^2} \quad (5)$$

ここで，ソースコード $d \in D$ は，D をソースファイル群とし，その中の任意の一つのファイル中のソースコードを d とし，ソースファイル単位で分割する．よって，$tf(g,d)$ はソースコード d 中の g の出現回数，$df(g)$ は g が出現するソースファイル数，$df(\theta(g))$ は $\theta(g)$ を満たすソースファイル数である．

この式（5）を利用して，ソースコード中の特徴語を求めていく．以下に，その手順を示す．

1. 語の重み付けをするためのソースファイルを複数用意する．
2. それぞれのソースファイルのソースコード中のコメントを取り除く．
3. それぞれのソースコードを単語ごとに分割していく．
4. N-gram g とその語の長さ（N-gram の N）と $df(g)$, $df(\theta(g))$ を求める．
5. $df(g)$，$df(\theta(g))$ の値と式（4）を利用して，N-gram IDF を求める．
6. 求めた N-gram g からそれぞれのソースコードごとの $tf(g,d)$ を求める．
7. 5. と 6. の結果より，式（5）から $TF\text{-}IDF_{N-gram}(g,d)$ を求める．
8. 求めた $TF\text{-}IDF_{N-gram}(g,d)$ の値で N-gram g を降順に並び替え，その上位の g を特徴語とする．

ここで $df(\theta(g))$ の値を計算することは，単語の論理積に対する出現頻度を計算する必要があるため計算量が多い．また，N-gram g の種類が非常に多いため，ソースコード中から選ぶ N-gram g を絞ることも必要である．よって，ここでは白川真澄らの文字列解析手法を使った実装方法を用いた．まず，あらゆる N-gram g の中からある程度数を絞るために，拡張接尾辞配列 [1] を用いた極大部分文字列の抽出 [10] を行う．この極大部分文字列を N-gram g とする．次に $df(\theta(g))$ の値を計算するためにはウェーブレット木 [5] を利用することで計算量の短縮を行った．ウェーブレット木というデータ構造が文書頻度の計算処理においても高速に行えることは Gagie ら [4] が示している．

3.3 ソースコード中の特徴的部分抽出方法

本研究では，ソースコード中の特徴的部分を抽出するにあたって，ソースコードを行単位で確認し，特徴語が出現した行を数えていくことにした．そして，特徴語が出現した回数を縦軸に，ソースコードの行数を横軸に取ったヒストグラムを作成する．そして，ヒストグラムの値が高くなっている行に注目して，その部分を特徴的部分として判定する．

また，ソースコード中の特徴的部分を抽出するときにはソースコードを行単位で確認する．そのため，ソースコードを単語ごとに分割していくときには行ごとに単語に分割することにする．

3.4 研究設問

ここでは本研究の目的であるソースコード中の特徴的部分を抽出するにあたって，確かめたい研究設問につい

て説明する．

本研究で利用している N-gram IDF は元々テキスト文書の語の重み付けとして利用されているものである．そのため，それをソースコードに利用した第 3 章の節 3.2 の特徴語判定方法でソースコード中における特徴語を見つけることができるのかという問題がある．また，ソースコード中の特徴的部分を抽出する際に，見つかった特徴語が本当に特徴語である必要もある．よって，以下のような研究設問を設けた．

> RQ1. ソースコード中から特徴語を抽出できるのか

第 3 章の節 3.3 で説明した方法では，作成したヒストグラムを利用して，ソースコード中の特徴語が出現した回数が多い行を特徴的部分として抽出する．そのため，このヒストグラムを利用して，ソースコード中の特徴的部分の特定ができる必要がある．これを確かめるため，以下のような研究設問を設けた．

> RQ2. ソースコード中から特徴的部分を抽出できるのか

最後に，もし特徴的部分が抽出できるなら，その抽出した特徴的部分がソースコードの変更によってどのように変化するのかを調べたい．例えば，抽出した特徴的部分がソースコードの変更によって消えていったなら，その特徴的部分が何らかの変更が必要なソースコードである言えるかもしれない．また，抽出した特徴的部分がソースコードの変更によって消えずにそのまま存在するなら，大幅な変更を加える必要のないソースコードであると言えるかもしれない．したがって，そのような特徴が分かると開発者がソースコードに改良を加える際に何かの手助けになるかもしれない．よって，以下のような研究設問を設けた．

> RQ3. ソースコードの変更によって特徴的部分にはどのような変化があるのか

4 実験方法

第 3 章で述べた研究設問を調べるために，2 つの実験を行った．

```c
/* header files */
#include <stdio.h>
#include <stdlib.h>

/* main */
int main(void) {
 FILE *fp;
 char *filename = "sample.txt";
 int ch;

 /* ファイルのオープン */
 if ((fp = fopen(filename, "r")) == NULL) {
  fprintf(stderr, "%sのオープンに失敗しました.\n", filename);
  exit(EXIT_FAILURE);
 }

 /* ファイルの終端まで文字を読み取り表示する */
 while (( ch = fgetc(fp)) != EOF ) {
  putchar(ch);
 }

 /* ファイルのクローズ */
 fclose(fp);

 return EXIT_SUCCESS;
}
```

図 1. C 言語関数辞典 fgetc のサンプルプログラム[4]

まず，1 つ目の実験ではサンプルプログラムを用いてソースコード中の特徴語抽出の評価実験を行った．この実験結果を用いて RQ1 について答える．

次に，2 つ目の実験ではオープンソースプロジェクトを用いてソースコード中の特徴的部分抽出の実験を行った．この実験結果を用いて RQ2，RQ3 について答える．

4.1 サンプルプログラムを用いた特徴語抽出

「C 言語関数辞典[2]」の Web ページで公開されているサンプルプログラムを用いて特徴語抽出の評価実験を行った．C 言語関数辞典では，C 言語の関数やマクロの使い方に関する説明を主にしている．この Web ページでは，関数やマクロの説明がヘッダファイル別やアルファベット別などによって分類されて，掲載されている．また，関数の説明によってはサンプルプログラムを使って説明をしている場合がある．例えば，関数 fgetc ではサンプルプログラムを使って説明をしている．そのサンプルプログラムは図 1 である．このようなサンプルプログラムからコメントを除いて特徴語抽出を行う．また加えて，このサンプルプログラムでは文字列の出力の際に日本語が使われているので，特徴語を計算する前にすべての日本語を「JAPANESE」に変換した．

[2]http://www.c-tipsref.com/

このサンプルプログラムを複数用いて特徴語抽出の評価実験を行う．具体的には，C 言語関数辞典内のサンプルプログラムを集めてきて，それぞれのサンプルプログラムで説明している関数名をファイル名とし保存する．また，それぞれのサンプルプログラムのファイルの正解語はそのファイルで説明している「関数の名前」を含むものとする．例えば，図 1 のサンプルプログラムの場合，「fgetc」を含む単語 N-gram が正解語となる．さらに，サンプルプログラムは，Web ページ内のヘッダファイル一覧から「stdio.h」「stdlib.h」「string.h」「ctype.h」の 4 つのヘッダファイルを選択し，その中の関数の説明の際にサンプルプログラムを使用している箇所から集めた．集めたサンプルプログラムのファイル数は全部で 94 個になった．

評価実験では第 3 章の節 3.2 で説明したソースコード中の特徴語判定方法を用いて行う．それぞれのサンプルプログラム中の特徴語に順位付けをし，1 位の特徴語にそのサンプルプログラムの関数名が入っていたら正解として数え，その正解率を調べた．正解率 R は正解数を a とし，D をファイル数とすると，以下の式（6）で求める．

$$R = \frac{a}{D} \tag{6}$$

したがって，この正解率の値でソースコード中の特徴語抽出の評価実験を行う．

4.2 Apache Ant を用いた特徴的部分抽出

Apache ソフトウェア財団が開発しているオープンソースのソフトウェアプロジェクトである Apache Ant というビルドツールソフトウェアの Git リポジトリ[5]を用いて特徴的部分抽出の実験を行った．

また，ソースコードの変更による特徴的部分の変化についても調べるため，特定のファイルの変更があった際の N-gram IDF を全て求める．具体的には，以下のようにして特定のファイル（以下ファイル A とする）の変更時のすべての N-gram IDF を求めることにする．

1. 現時点[6]の Apache Ant のソースコードを全て集めてきて，一つのファイル（以下ファイル B とする）にする．また，ファイル B を作成する際は集めてきたソースコードの参照先がわかるように一つ一つ区切る．
2. Apache Ant の全コミットのログを集めてきて，ソースコードファイルに対するすべての変更履歴を調べる．
3. 変更履歴を探索していき，ファイル B のソースコードの一部をその変更があったファイルのソースコードに置き換えていく．もし，変更があったファイルがファイル A なら，ファイル B を書き換えた後に N-gram IDF を求める．

以降，3. を繰り返すことによって，ファイル A の N-gram IDF を全て求めることができる．

この求めた N-gram IDF を用いて，$TF\text{-}IDF_{N-gram}$ をそれぞれのファイルについて求めて，行ごとの特徴語の出現頻度を表したヒストグラムを作成していく．また，ソースコードの変更による特徴的部分の変化については，横軸がソースコードの行数，縦軸が変更回数のヒートマップを作成し，特徴語の出現回数によって行ごとに色を変化させる．そして，その特徴的部分の色の変化によって，ソースコードの変更による特徴的部分の変化について調べることにする．

5 実験結果

5.1 RQ1:ソースコード中から特徴語を抽出できるのか

サンプルプログラムを用いた特徴語抽出の実験を利用して，RQ1 に答えていく．例えば，図 1 に対して特徴語抽出を行い，語の長さが 2 語以上で上位 10 位以内の単語 N-gram を表 1 に示す．表 1 により，最も特徴的な単語 N-gram が「 (ch 」となっていて，次いで「 (fp 」，その後に 4 つの単語 N-gram が同じ特徴度になっていることがわかる．さらに，このソースコードの正解の特徴語は「 `fgetc` 」を含むものとしていたので，それを含む単語 N-gram「 ch = fgetc (fp) 」は第 3 位となっている．このようにして，C 言語関数辞典から集めてきた全 94 個のソースコードのファイルに対して特徴語の抽出を行い，評価を行った．

まず，全てのサンプルプログラムに対して行った特徴語抽出の評価実験の結果をまとめた表が表 2 である．ここで正解数とは，単語 N-gram を順位付けしたときの 1 位の特徴語が正解の語を含んでいるサンプルプログラム

[4]参照元 http://www.c-tipsref.com/reference/stdio/fgetc.html (参照日 2017/02/01)
[5]https://github.com/apache/ant
[6]実験では 2016 年 10 月 18 日時点

表 1. fgetc のサンプルプログラムに対する特徴語抽出の結果

語の長さ	$tf(g,d)$	$df(g)$	$df(\theta(g))$	IDF_{N-gram}	$TF\text{-}IDF_{N-gram}$	単語 N-gram g
2	2	5	5	4.233	8.465	(ch
2	3	20	20	2.233	6.698	(fp
2	1	3	3	4.970	4.970	putchar (
5	1	3	3	4.970	4.970	while ((ch =
6	1	3	3	4.970	4.970	ch = fgetc (fp)
4	1	3	3	4.970	4.970	) { putchar (
6	1	4	4	4.555	4.555	) ! = EOF) {
2	2	20	20	2.233	4.465	fp)
3	2	20	20	2.233	4.465	(fp)
7	1	2	3	4.385	4.385	; } while ((ch =

表 2. ファイル数 94 個に対する特徴語の正解率

語の長さ N 語 [以上]	正解数	正解率
1	16	0.170
2	23	0.245
3	30	0.319
4	27	0.287
5	19	0.202
6	12	0.128
7	8	0.085
8	4	0.043
9	0	0.000

表 3. ファイル数 94 個に対する特徴語のヒット数とその適合率

語の長さ N 語 [以上]	ヒット数	正解数	適合率
1	50	16	0.320
2	47	23	0.489
3	44	30	0.682
4	39	27	0.692
5	29	19	0.655
6	14	12	0.857
7	10	8	0.800
8	7	4	0.571
9	0	0	0.000

の数である．また，正解率は式（6）を用いて計算している．表 2 より，正解率は長さ 3 語以上のときに最も高くなっていることがわかる．しかし，正解率自体は 0.319 とあまり高い値にはならなかった．そこで，サンプルプログラムに対して特徴語抽出を行った際のヒット数と適合率についても調べ，表 3 に示す．ここで，適合率 R-Prec は，ヒット数を h，正解数を a とすると以下の式（7）で表す．

$$R\text{-}Prec = \begin{cases} \frac{a}{h} & (\mathrm{h} > 0) \\ 0 & (\mathrm{h} = 0) \end{cases} \tag{7}$$

ヒット数 h とは，ファイル中のソースコードの単語 N-gram を順位付けした際に，その中に順位は関係なく正解語が入ってるファイルの数である．表 3 より，全ファイル 94 個に対して正解語を含む単語 N-gram が入ってたファイルの数は最大で 50 となっていて，約半数近くのファイルがヒットしていないことがわかる．さらに，語の長さが増えていくたびに値が低くなっていき，語の長さが 9 語以上で 0 になる．このことから，正解率が低かったのはプログラムが単語 N-gram を作成する際に，その単語 N-gram に正解の語が含んでいないものが約半数以上あったからと推測した．そのため，ソースコードの特徴を考えたテキストとは違う単語 N-gram の作成の必要があると考える．

また，適合率は語の長さが 3 語以上から 7 語以上までで 6 割を超え，6 語以上で最大値 0.857 をとる．よって，語の長さが 3 語以上の連続した語に対して特徴語の抽出を行うと，正解率が 0.319 で，適合率が 0.682 となり，どちらの値も比較的高い値になる．このことから，語の長さが 3 語以上の連続した語に対して特徴語の抽出を行い順位付けをすると，上位の語がソースコードにおいて特徴語である可能性が高くなることがわかる．

以上の結果より，ソースコード中から 3 語以上の連続した語なら，比較的高い確率で特徴語を抽出できるといえる．

5.2 RQ2:ソースコード中から特徴的部分を抽出できるのか

ここでは，Apache Ant を用いた特徴的部分抽出の実験を利用して，RQ2 に答えていく．

はじめに，RQ1 の実験の結果を利用して，語の長さが 3 語以上の単語 N-gram に対してのみ特徴語を求め，ソースコードのファイルを代表する特徴語とした． さらに，$TF\text{-}IDF_{N-gram}$ の値で並び替えた上位の単語 N-gram を

特徴語とする時に，上位 100 位以内の単語 N-gram を特徴語として候補を絞り込むこととした．

また，事前の実験で上位 100 位以内の単語 N-gram を特徴語とした結果，同じ行に複数の特徴語が出現することが分かった．行あたりの特徴語の数によって特徴的部分を特定する際に，ある一部の行の特徴語の数が大きくなると，その行以外の値が相対的に低くなり特定が難しくなる． 同じ行に複数の特徴語が出現する理由としては，特徴語の中に似た語の組み合わせで，かつ $TF\text{-}IDF_{N-gram}$ の値が近いからである．例えば，表 4 のような場合がある．この表は「 elementAt (i) 」と「 . elementAt (i) 」が特徴語となっていて，残りの 4 つは 2 つの特徴語を含む単語 N-gram を幾つか挙げたものである．この場合，表の出現箇所を見ると，2 つの特徴語の出現箇所が重なってしまっていることがわかる．この問題に対処するために，特徴語の出現箇所からそれ以降の順位の特徴語を含む単語 N-gram の出現箇所を除くことにした．これを適用すると，2 つの特徴語の出現箇所はそれぞれ「 elementAt (i) 」のとき出現箇所はなくなり，「 . elementAt (i) 」のときの出現箇所は 618 と 840 になる．これ以降，特徴語の出現箇所は以上のようにして求める．

次に，Apache Ant のどのソースコードに対して特徴的部分の抽出を行うか決める．2016 年 10 月 18 日時点で，Apache Ant では全 1222 個の拡張子が java のソースコードファイルが存在する．そのため，ある程度特徴量の高い可能性があるソースコードに絞って実験を行いたい．よって，2016 年 10 月 18 日時点の Apache Ant の全 1222 個のソースコードファイルに対して特徴語抽出を行い，大域的重み付け N-gram IDF の高い値が比較的多いものを探し出した．その結果の比較的多かったファイルの幾つかを表 5 に示した．この中から,「Main.java」と「Javadoc.java」を選んで実験を行った．

まず，2013 年 7 月 17 日時点の「Main.java」に対して特徴的部分抽出を行った結果について説明する．図 2 は特徴語の出現箇所を行ごとに数えていき，行ごとの特徴語の出現頻度をグラフにしたヒストグラムである．グラフから特徴的部分を値が 10 以上のグラフに注目して考えていく，まず 660 行から 700 行あたりが一番高くなり，次に 150 行から 200 行あたり，270 行から 300 行あたりが高くなっている．また，370 行から 400 行あたりも少し高くなっている．

一番値が高くなった 660 行から 700 行あたりは，

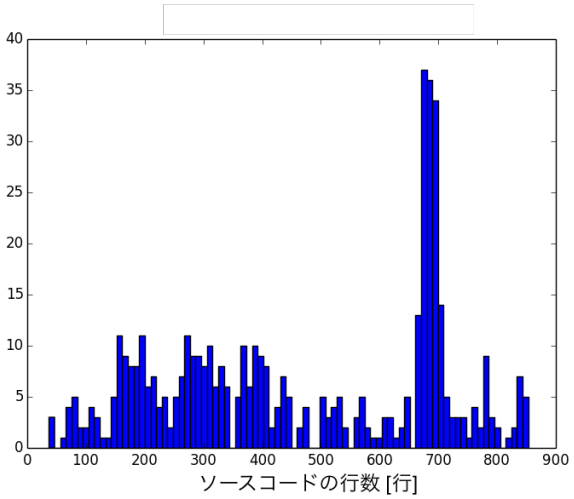


図 2. 2013 年 7 月 17 日における Main.java における特徴語の出現頻度

「Main.java」のソースコードでも見た目からも特徴的なものになっていた．それは，プログラムの引数の使用方法の文字列の出力をする関数（関数名 printUsage）であった．そのため，文字列を表示させるための関数を複数使用していた．次に値が高くなっていた 150 行から 200 行あたりや 270 行から 300 行あたりは「Main.java」のソースコードでは実行時に与えられた引数の処理をする関数（関数名 processArgs）になっていた．さらに詳しく見ていくと，150 行から 200 行あたりは引数の条件分岐のコードとなっていて，270 行から 300 行あたりはビルドファイルがなかったときの条件分岐のコードになっていた．また，次いで値が高くなっていた 370 行から 400 行あたりは，与えられた引数の処理をする際のそれぞれの引数で指定された値を変数に保存したり，その値が無効なものならエラー処理をする関数がいくつか並んでいた．これらの関数は関数 processArgs 内の 150 行から 200 行あたりの引数の条件分岐のコードで利用されていた．

よって,「Main.java」において特徴的部分は大きく分けて 3 つ挙げられる．

- 1 つ目は，660 行から 700 行あたりのプログラムの引数の使用方法の文字列を出力する関数 printUsage
- 2 つ目は，関数 processArgs で具体的には 150 行から 200 行あたりの引数の条件分岐のコードと 270 行から 300 行あたりはビルドファイルがなかったときの条件分岐のコード

表 4. 同じ行に複数の特徴語が出現する理由

語の長さ	$TF\text{-}IDF_{N-gram}$	単語 N-gram g	出現箇所 [行]
4	25.963	elementAt (i)	618,812,832,835,836,840
5	25.769	. elementAt (i)	618,812,832,835,836,840
6	16.942	names . elementAt (i)	812,832,835
8	11.295	(names . elementAt (i))	812,832
6	8.227	. elementAt (i))	812,832,836
6	0.835	. elementAt (i) .	835

表 5. 大域的重み付け N-gram IDF の高い値が多かったファイル

ファイル名	参照先
Javadoc.java	ant/src/main/org/apache/tools/ant/taskdefs/
BlockSort.java	ant/src/main/org/apache/tools/bzip2/
ZipOutputStream.java	ant/src/main/org/apache/tools/zip/
CBZip2InputStream.java	ant/src/main/org/apache/tools/bzip2/
CBZip2OutputStream.java	ant/src/main/org/apache/tools/bzip2/
IntrospectionHelper.java	ant/src/main/org/apache/tools/ant/
Zip.java	ant/src/main/org/apache/tools/ant/taskdefs/
ZipFile.java	ant/src/main/org/apache/tools/zip/
Main.java	ant/src/main/org/apache/tools/ant/

- 3 つ目は，370 行から 400 行あたりの与えられた引数の処理をする際のそれぞれの引数で指定された値を変数に保存したり，その値が無効なものならエラー処理をする関数群

また，2013 年 10 月 27 日時点の「Javadoc.java」に対して特徴的部分抽出を行った結果，図 3 に示すヒストグラムを得ることができた．これに基づき，次のように特徴的部分を抽出できた．

- 1 つ目は，50 行から 710 行あたりの「cmd.createArgument(」というコードを含む関数群
- 2 つ目は，関数 execute 内の 760 行あたりで関数呼び出しをしているソースコード
- 3 つ目は，870 行から 940 行あたりと 1050 行から 1200 行あたりの関数 execute を実行するために何らかの引数に値をセットする関数群

以上より，「Main.java」と「Javadoc.java」のソースコードについて，出力したヒストグラムから特徴語の出現頻度の値が高くなっている部分と低くなっている部分が存在し，その値が高くなっている部分の周辺を特徴的部分として抽出することができた．

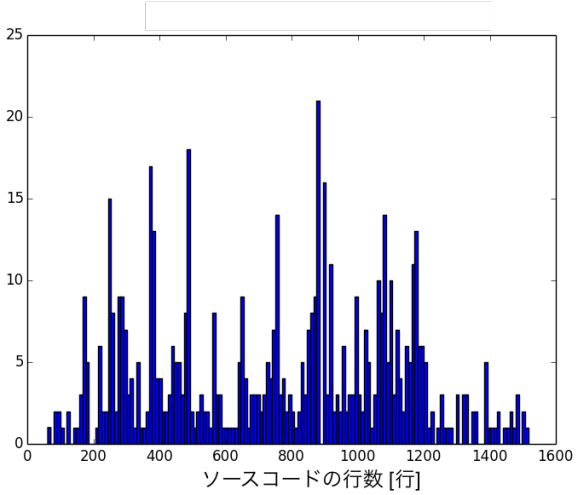


図 3. 2013 年 10 月 27 日における Javadoc.java における特徴語の出現頻度

5.3 RQ3:ソースコードの変更によって特徴的部分にはどのような変化があるのか

Apache Ant を用いた特徴的部分抽出の実験を利用して，RQ3 に答えていく．この実験で求めた「Main.java」と「Javadoc.java」の変更時のそれぞれの特徴語を用いて，10 行ごとの特徴語の出現回数を求めた．その値をヒートマップの値として，横軸を行数，縦軸を変更回数としてヒートマップを作成した．

まず，「Main.java」の全変更時に対して特徴的部分抽出を行い 10 行ごとの特徴語の頻出度をヒートマップにした結果について説明する．その結果を図にしたのが図 4 である．このソースコードの変更回数は 167 回で，行数は最大で 870 行であった．一番上が初期の古いソースコードの特徴量の分布を表していて，下に行くにつれて新しいソースコードの分布になっていく．図 4 を見ると，ソースコードの行数が徐々に増えていき，頻繁に変更がされていることがわかる．図 4 より，5.2 節で説明した

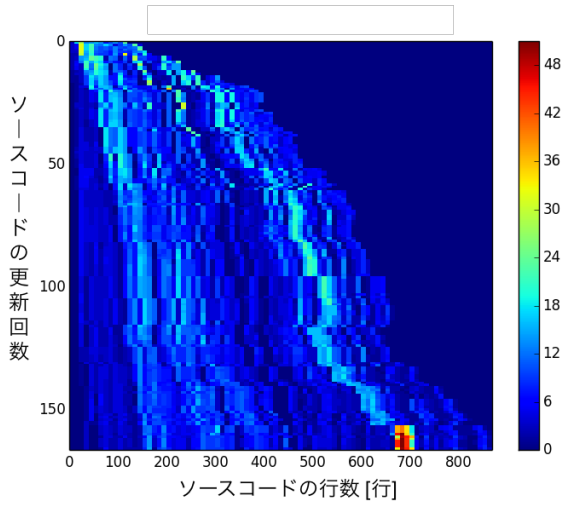


図 4. Main.java における特徴語出現頻度の時系列的推移

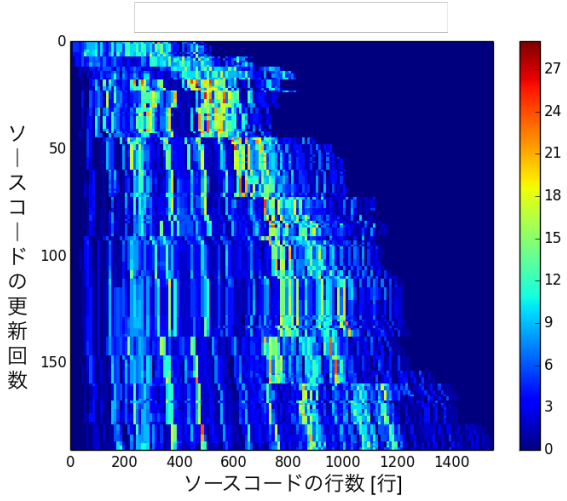


図 5. Javadoc.java における特徴語出現頻度の時系列的推移

特徴的部分の値が高くなっていることがわかり，それぞれのソースコードの特徴的部分はソースコードの変更があってもソースコード全体の相対的な特徴的部分の位置の変化は少ないことがわかる．

同じく,「Javadoc.java」の全変更時に対して特徴的部分抽出を行い 10 行ごとの特徴語の頻出度をヒートマップにした結果 (図 5) について説明する．このソースコードの変更回数は 191 回で，行数は最大で 1552 行であった．図より,「Javadoc.java」においてもソースコードの行数が徐々に増えていき，頻繁に変更がされていることがわかる．また，5.2 節で説明した特徴的部分の値が高くなっていることがわかり，それぞれのソースコードの特徴的部分もソースコードの変更があってもソースコード全体の相対的な特徴的部分の位置の変化が少ないことがわかる．この「Javadoc.java」においては更新回数が 50 回あたりなどでソースコードが大幅に変更があっても，それに合わせて特徴的部分がソースコード全体の位置に合わせて移動している．このことからも，特徴的部分の位置の変化が少ないことがわかる．以上より，ソースコードの変更によって特徴的部分の位置が他に移ることはなく，ソースコード全体の相対的な特徴的部分の位置の変化が少ないことがわかった．

6 まとめ

本研究では，単語 N-gram に対して重み付けができる N-gram IDF をソースコードに対して利用してソースコード中の特徴的部分の抽出を行った．具体的には N-gram IDF を利用した語の重み付け手法である $TF\text{-}IDF_{N-gram}$ を用いて，ソースコードごとの特徴語の抽出を行い，その特徴語の行ごとの出現頻度を求めて特徴的部分の抽出を行った．

適用実験の結果より，N-gram IDF を用いた語の重み付け手法である $TF\text{-}IDF_{N-gram}$ を利用すると，ソースコード中の特徴語を抽出でき，その特徴語の行ごとの出現頻度から特徴的部分を抽出することができた．また，その特徴的部分はソースコードの変更によってソースコード全体の相対位置の変化が少ないということがわかった．

今後の課題としてはサンプルプログラムを用いて特徴語抽出を行った際の，単語 N-gram の特徴語候補のヒット数の改善がある．今回の研究では，複数ある単語 N-gram の数を絞るために拡張接尾辞配列を用いた極大部分文字列の抽出を利用している．しかし，この単語 N-gram の選択方法ではヒット数が少なかったことから，ソースコードに対して有効な単語 N-gram を選択しないのかもしれない．よって，異なるソースコードの特徴を利用した単語 N-gram の選択方法が考えられる．

参考文献

[1] M.I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. In *Journal of Discrete Algorithms*, Vol. 2, pp. 53–86, 3 2004.

[2] A. Aizawa. An information-theoretic perspective of tf-idf measures. In *Information Processing and Management*, Vol. 39, pp. 45–65, 2003.

[3] B.C. Fung, K. Wang, and M. Ester. Hierarchical document clustering using frequent itemsets. In *Proceedings of SIAM International Conference on Data Mining(SDM)*, pp. 59–70, 5 2003.

[4] T. Gagie, G. Navarro, and S.J. Puglisi. New algorithms on wavelet trees and applications to information retrieval. In *Theoretical Computer Science*, Vol. 426-427, pp. 25–41, 4 2012.

[5] R. Grossi, A. Gupta, and J.S. Vitter. High-order entropy-compressed text indexes. In *SODA*, pp. 841–850, 2003.

[6] K.S. Hasan and V. Ng. Conundrums in unsupervised keyphrase extraction: Making sense of th state-of-the-art. In *Coling*, pp. 365–373, 8 2010.

[7] D. Hiemstra. A probabilistic justification for using tfx-idf term weighting in information retrieval. In *International Journal on Dgital Libraries*, Vol. 3, pp. 131–139, 9 2000.

[8] K.S. Jones. A statistical interpretation of term specificity and its application in retrieval. In *Journal of Documentation*, Vol. 28, pp. 11–21, 1972.

[9] D. Metzler. Generalized inverse document frequency. In *CIKM*, pp. 26–30, 10 2008.

[10] D. Okanohara and J. Tsujii. Text categorization with all substring features. In *SDM*, pp. 838–846, 4 2009.

[11] S. Robertson. Understanding inverse document frequency: On theoretical arguments for idf. In *Journal of Documentation*, Vol. 60, pp. 503–520, 2004.

[12] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, pp. 1470–1477, 10 2003.

[13] H.C Wu, R.W.P. Luk, K.F Wong, and K.L. Kwok. Interpreting tf-idf term weights as making relevance decisions. In *ACM Transactions on Information Systems*, Vol. 26, pp. 13:1–13:37, 6 2008.

[14] 白川真澄, 原隆浩, 西尾章治郎. コルモゴロフ複雑性に基づく idf の単語 n-gram への適用. In *DEIM Forum*, 第 A 巻, pp. 3–5, 2015.

ソースコードの品質向上を目的とした特定のコメントを検出するツールの試作

田上諭
宮崎大学 工学研究科
tanoue@earth.cs.miyazaki-u.ac.jp

甲斐秀一
株式会社スカイコム
h-kai@skycom.jp

片山徹郎
宮崎大学 工学教育研究部
kat@cs.miyazaki-u.ac.jp

要旨

TODO コメントと言ったマーキングの役割をするコメントや不適切なコメントを検出できるようになると，ソースコードの品質向上に役に立つ．本稿では，ソースコードの品質向上を目的として，特定のコメントを検出するツールを試作する．試作するツールでは，正規表現を利用し，検出文字列のパターンを *XML(Extended Markup Language)* ファイルに設定できる．これにより，検出したい特定のコメントを検出できる．新たに検出したい特定のコメントが出てきた場合も，検出文字列のパターンとして追加することにより，柔軟に対応できる．株式会社スカイコムの協力で *LOC(Line of Code)* が *5000* 行以上のプロジェクトを対象に試作したツールを適用し， 適合率及び再現率を調査した．調査した結果，検出文字列のパターン調整することにより再現率は *100* %を，適合率は *80* %を，それぞれ達成した．また，一度設定した *XML* ファイルは，他のプロジェクトでもそのまま適用でき，再利用可能なことが確認できた．試作したツールを用いることで，マーキングの役割をするコメントやソースコードに存在する不適切なコメントを検出できる．マーキングの役割をしているコメントや不適切なコメントを減らしていくことで，ソースコードに存在する課題の削減や理解容易性の向上が図れる．

1. はじめに

コメントは，ソフトウェアの動作に影響を与えずに，自然言語で自由に記述できる．そのため，コメントを用いるとソースコードでは表現できない内容も記述できる．例えば，TODO コメントは，ソースコードに存在する課題をコメントとして追加することができ，適切に記述することで有力な情報となる．例えば，Google C++ Style Guide[1] では，TODO コメントを記述することが推奨されている．このようなマーキングの役割をするコメントは，ソースコードに存在する問題点を見つけるのに有力な情報となるので，コメントとして残しておくことが推奨される．

しかし，TODO コメントは，その存在を忘れてしまう可能性があり，TODO コメントが存在したままの状態になってしまうことがある．TODO コメントが表している課題を修正した時点で，TODO コメントは削除されるべきである．すなわち，TODO コメントなどのマーキングの役割をするコメントは，ソースコードに課題が残っていることを調べるために，必要に応じて検出する必要がある．

また，マーキングの役割をするコメントとは別に，ソースコードに存在すると問題のあるコメントが存在する．例えば，処理そのものを説明しているコメントである．このようなコメントは，ソースコードを読めば理解できるので，存在する意味がない．単に読む量が増え，手間となる． 本研究では，このようなコメントを不適切なコメントと呼ぶ．不適切なコメントの詳細については，2 章で説明する．不適切なコメントは，目視による確認で

リスト 1. 改修履歴コメント

```
1 //平成18年6月12日　田上修正
2 /*2016/08/12　田上　修正*/
```

リスト 2. 意図が曖昧なコメント

```
1 //暫定的な処理
2 //とりあえず0で
```

発見可能であるが，人的資源の不足や開発の遅れから，十分な確認作業が行われない可能性があるので，ツールによる支援が必要となる．

そこで本研究では，ソースコードの品質向上を目的として，特定のコメントを検出するツールを試作する．試作するツールは，正規表現を利用し，検出文字列のパターンを XML(Extended Markup Language) ファイル [2] に設定できる．この XML ファイルの検出文字列のパターンにマッチするコメントを検出する． 一度設定した XML ファイルは，その他のプロジェクトでも再利用できる．新たに検出したい特定のコメントが出てきた場合は，検出文字列のパターンとして追加することにより，柔軟に対応できる．本ツールを利用することにより，特定のコメントを検出できる．マーキングの役割をしているコメントや不適切なコメントを減らしていくことで，ソースコードに存在する課題の削減や理解容易性の向上が図れる．

本稿の構成は，次のとおりである．2 章では，不適切なコメントを定義する．3 章では，試作したツールの外観とその機能を説明する．4 章では，ツールの実装方法について説明する．5 章では，ツールの適用例について説明する．6 章では，ツールの考察を行う．7 章では，本稿についてまとめる．

2. 不適切なコメント

不適切だと考えられるコメントの候補をあげ，コメントか不適切かどうかアンケートを行った．アンケートの対象は，株式会社スカイコムの開発者 16 人である [3]．アンケートにおいて 8 割以上が不適切だと答えたものを，不適切なコメントと定義する．以下に，不適切なコメントの分類とその理由を示す．

- 改修履歴コメント

  改修履歴コメントの例を，リスト 1 に示す．リスト 1 に示した改修履歴コメントは，改修履歴が書かれているコメントである．コメントがメンテナンスされずにソースコードの現状とは異なった内容になっている恐れがある．また，改修履歴コメントは改修がされていく度に，コメントが増えていき，ソースコードが読みにくくなる．以上の理由から， 改修履歴のコメントを残すのではなく，バージョン管理ツールを利用するべきである．

- 意図が曖昧なコメント

  意図が曖昧なコメントの例を，リスト 2 に示す．リスト 2 に示した意図が曖昧なコメントは，どのような意図で書かれているのかよくわからないコメントである．意図が曖昧なコメントが存在すると，読み手によって解釈が異なる場合があり，混乱を招いてしまう．

- 処理そのものを説明しているコメント

  処理そのものを説明しているコメントの例を，リスト 3 に示す．リスト 3 に示したコメントは，処理そのものを別の言葉で言い換えただけのコメントである．このようなコメントは，ソースコードの読み手に追加情報を一斉提供せずに，単に読むソースコードの量を増やすだけである．処理そのものを説明しているコメントは，今回試作するツールで検出するのは難しい．本稿では，このコメントは検出対象とはしない．その理由については，7 章の今後の課題で説明する．

リスト 3. 処理そのものを説明しているコメント

```
1 // ここで10回ループする
2 while (i < 10) {
3   i++; // iをインクリメントする
4 }
```

3. 試作ツール

試作したツールは，Java 言語で開発した．Java 仮想マシンがインストールできる環境であれば，任意の OS 上で動作する．

ツールの外観と機能,「コメントを解析」機能の詳細，XML ファイルについて，以降それぞれ説明する．

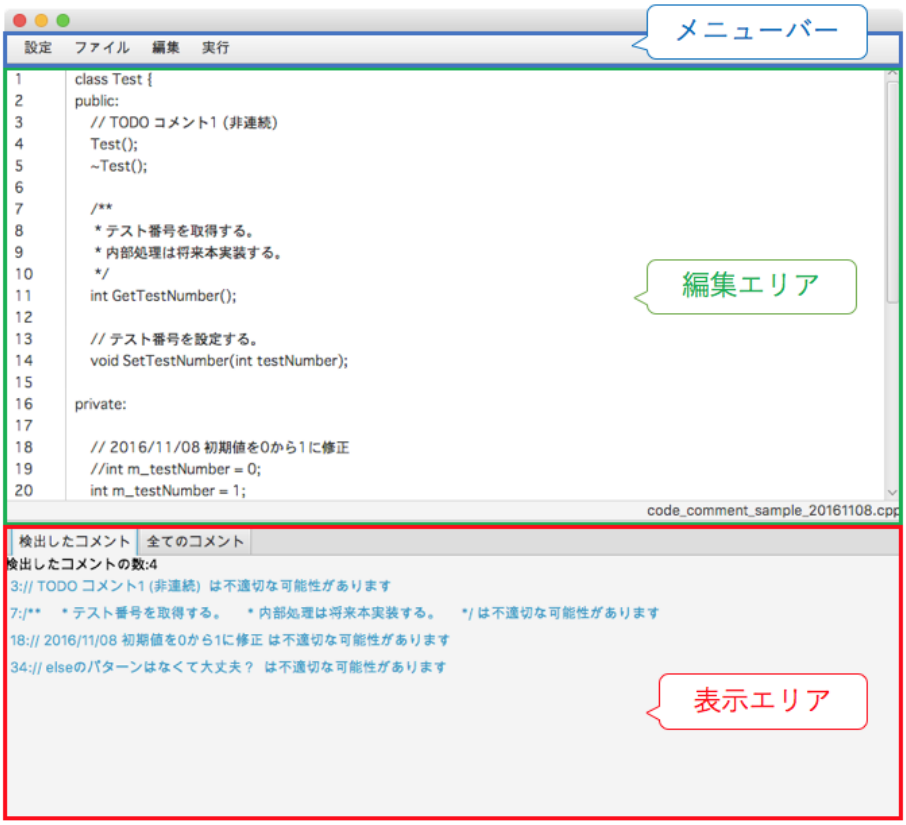


図 1. ツールの外観

3.1. ツールの外観と機能

本研究で試作したツールの外観を，図 1 に示す．試作したツールはメニューバー，編集エリア，表示エリアから構成している．ツールのユーザーは，メニューバーからメニューアイテムを選択することによって，ツールの機能を利用できる．

ユーザーがメニューバーの「設定」をクリックすると，設定メニューを表示する．設定メニューから，下記に示す 1 つの機能を利用できる．

- 文字コード
  ユーザーが「文字コード」にカーソルを合わせると更に 2 つのメニューを表示する.「UTF-8」と「Shift-JIS」である．ユーザーは，どちらかを選択することができ，起動時は「UTF-8」をデフォルトで選択している．ユーザーが文字コードを選択すると，試作したツールは選択した状態の文字コードで編集エリアのコードを開き直し，次回以降は選択した文字コードでファイルを開く．

ユーザーがメニューバーの「ファイル」をクリックすると，ファイルメニューを表示する．ファイルメニューから，下記に示す 3 つの機能を利用できる．

- ファイルを開く
  ユーザーが「ファイルを開く」をクリックすると，ファイルオープンダイアログを表示する．ユーザーが，ファイル―オープンダイアログからファイルを選択すると， ファイルの内容を編集エリアに表示する．

- ファイルを保存
  ユーザーが「ファイルを保存」をクリックすると，ファイルセーブダイアログを表示する．ユーザーがファイルの保存先及びファイル名を入力し，ファイルセーブダイアログの「Save」ボタンをクリックすると，編集エリアの内容を読み込みファイルに保存する．

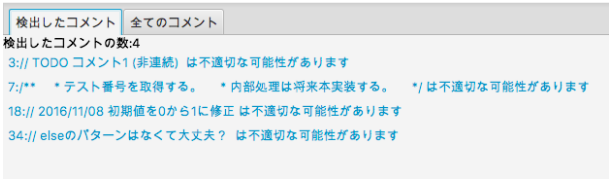


図 2. パターンにマッチしたコメントを表示するタブ

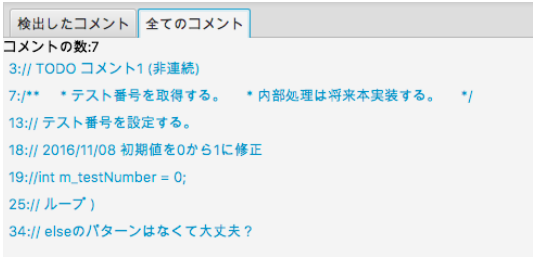


図 3. 全てのコメントを表示するタブ

- 複数ファイルのコメントを解析
  ユーザーが「複数ファイルのコメントを解析」をクリックすると，ファイルオープンダイアログを表示する．ユーザは，ファイルオープンダイアログから複数のファイルを選択できる．ユーザがファイルを選択すると，それぞれのファイルの内容を読み込み，それぞれコメントを解析する．その後，それぞれの解析結果を表示エリアに表示する．

ユーザーがメニューバーの「編集」をクリックすると，編集メニューを表示する．編集メニューから，下記に示す 2 つの機能を利用できる．

- 編集エリアをクリア
  ユーザーが「編集エリアをクリア」をクリックすると編集エリアの内容をクリアする．

- 表示エリアをクリア
  ユーザーが「表示エリアをクリア」をクリックすると表示エリアの内容をクリアする．

ユーザーがメニューバーの「実行」をクリックすると実行メニューを表示する．実行メニューから，下記に示す 1 つの機能を利用できる．

- コメントを解析
  ユーザーが「コメントを解析」をクリックすると編集エリアの内容を読み込み，コメントの解析を行う．試作したツールは，コメントを解析した結果を表示エリアに表示する．詳細については，次節で述べる．

3.2. コメントを解析

コメントは，行コメント (//) とブロックコメント (/* */) に対応している．行コメント及びブロックコメントの表記が (//)，(/* */) であるプログラミング言語であれば，どんな言語でもコメントを解析できる．

「コメントを解析」は，まずソースコードに存在する全てのコメントを抽出する．次に，XML ファイルに記述されている検出文字列のパターンに基づきマッチングする．検出文字列のパターンにコメントの記述内容がマッチした場合，マッチしたコメントとして検出する．検出したコメントが検出対象のコメントかどうかは，最終的には人により判断する必要がある．

コメントの解析結果は，表示エリアで確認できる．表示エリアは，検出文字列のパターンにマッチしたコメントを表示するタブと，全てのコメントを表示するタブの 2 つに分けている．「検出したコメント」タブを，図 2 に示す．このタブでは，パターンにマッチしたコメントを確認できる．解析結果をクリックすると，編集エリアにおいて，そのコメントのある行に移動する．「全てのコメント」タブを，図 3 に示す．このタブでは，ソースコードにある全てのコメントを確認できる．解析結果をクリックすると，編集エリアにおいて，そのコメントのある行に移動する．

3.3. XML ファイル

XML ファイルの例を，リスト 4 に示す．XML ファイルでは，Java 言語が利用できる正規表現 [4] で検出文字列のパターンを記述することができる．

リスト 4 に記述した検出文字列のパターンで，検出可能なコメントは「//TODO メソッドを分かりやすくする」，「//将来は，データをファイルから読み込めるようにする.」といった，「TODO」という単語が含まれているコメント及び「将来」という単語が含まれているコメントである．また，「//2017/3/03 田上 修正」といった「(?=.*\d{2,4}[.\-/年] \d{1,2}[.\-/月]\d{1,2}日

リスト 4. XML ファイルの例

```
<?xml version="1.0" encoding="UTF-8" ?>
<comments>
    <comment type="TODOコメント">
        <regularExpression>TODO</regularExpression>
        <regularExpression>将来</regularExpression>
    </comment>
    <comment type="改修履歴コメント">
         <!-- 検出する日付フォーマットは以下
             yyyy/mm/dd、yyyy.mm.dd、yyyy-mm-dd、yyyy年mm月dd日 (yyyyは2桁以上4桁以下)-->
        <regularExpression>(?=.*\d{2,4}[.\-/年]\d{1,2}[.\-/月]\d{1,2}日{0,1})</regularExpression>
    </comment>
</comments>
```

{0,1})」の正規表現がマッチするコメントである．

「//Todo:インデントを綺麗にする．」というコメントを検出したい場合は，TODO という検索ワードを大文字と小文字を区別付せずに判断するように XML ファイルを変更する必要がある．正規表現により，リスト 4 の 4 行目を，「<regularExpression>(?i)TODO</regularExpression>」に変更する．なお，「?」などの正規表現に使われる特殊文字を検出文字列のパターンとしたい場合は，「?」の前に「\」を追加し「<regularExpression>\?</regularExpression>」とする必要がある．

4. 実装

試作するツールの「コメントを解析」機能は，コメントの抽出及びパターンマッチにより構成する．以降，それぞれについて述べる．

4.1. コメントの抽出

コメントの抽出には，JavaCC(Java Compiler Compiler) を用いる．JavaCC は，Java 言語が利用できるパーサジェネレータとしてよく使われる [5]. EBNF(Extended Backus Naur Form)[6] に似た文法で文法ファイルを記述しコンパイルすると，Java 言語で記述されたパーサを自動生成する．

コメントの抽出に利用している文法ファイルを，リスト 5 に示す．リスト 5 の 23 行目の定義で，行コメント (//) に一致する字句か判断でき，24 行目の定義で，ブロックコメント (/* */) に一致する字句か判断する [7]．それ以外のものは，コメントの抽出には必要ないので，25 行目で<OTHERS>の字句と判断し，34 行目から 41 行目で構文解析を行う．コメントを見つけるだけならば，「((<OTHERS>*)(<LINE_COMMENT>|<BLOCK_COMMENT>)(<OTHERS>*))*」の構文規則の記述ですむ．今回試作したツールでは，コメントを取得して，そのコメントと検出文字列のパターンが一致しているかを調べたい．そこで，構文に一致した際にそのコメントを取得するコードを追加している．

この文法ファイルを JavaCC でコンパイルすると，CommentParser.java などのファイルが生成される．CommentParser.java には，CommentParser というクラスが記述されている．生成されたパーサの使用法を，リスト 6 に示す．リスト 6 の 2 行目で，CommentParser のインスタンスを作成している．コンストラクタに，引数として StringReader 型の文字列を渡す．このパーサに渡された文字列が，構文解析される．構文解析は 5 行目の paser.comment() が実行されるタイミングで実行される．コメントの解析結果は，String を格納する ArrayList インスタンスである comments に保持される．この方法で全ての行コメントと，ブロックコメントを抽出できる．

4.2. パターンマッチ

コメントの解析は，XML ファイルを読み込み解析する．XML ファイルを，リスト 4 に示した．

このファイルから，<comments>の子要素の子要素である<regularExpression>の内容を取得する．取得には，DocumentBuilderFactory クラス [8] を利用する．取得したコメントと取得した内容それぞれのパターンがマッチしているか判断し，パターンにマッチしたものを，「検出したコメント」タブに表示する．

リスト 5. コメントの抽出に使っている文法ファイル CommentParser.jj

```
options {
  STATIC=false;
  UNICODE_INPUT =true;
}

PARSER_BEGIN(CommentParser)
  package CommentParser;

  import java.util.ArrayList;
  import Dictionaly.*;
  import ResultData.*;

public class CommentParser{
}
PARSER_END(CommentParser)

SKIP :{
    <  SPACE:([" ","\t","\r","\f"])+ >
}

TOKEN :
{
      <LINE_COMMENT:"//"(~["\n","\r"])* ("\n"|"\r\n" | "\r")?>
      |<BLOCK_COMMENT: "/*" (~["*"])*("*")+ (~["/","*"](~["*"])*("*")+)* "/">
      |<OTHERS:~[]>
}
ArrayList<String> comment():
{
    Token lineComment,blockComment;
    ArrayList<String> result = new ArrayList<String>();

}
{
    ( (<OTHERS>*)
          (lineComment = <LINE_COMMENT>
          { result.add(lineComment.image); }
          | blockComment =<BLOCK_COMMENT>
          { result.add(blockComment.image); }
          )
         (<OTHERS>*))*
      { return result; }
}
```

5. 適用例

株式会社スカイコムの協力で LOC(Line of Code) が 5000 行以上のプロジェクトを対象に試作したツールを適用し，適合率及び再現率について調べた [3]． 本稿での適合率は，検出結果の中にどれだけ検出すべきコメントが含まれたかを表し，計算式は以下となる．

$$\text{適合率} = \frac{\text{検出できた検出すべきコメント数}}{\text{検出した全コメント数}} \quad (1)$$

また，再現率は検出すべき検出対象のコメントのうち，どれだけ検出できたかを表し，計算式は以下となる．

$$\text{再現率} = \frac{\text{検出できた検出すべきコメント数}}{\text{検出すべきコメント数}} \quad (2)$$

今回の検出対象となるコメントは，TODO コメントと不適切なコメントである．ソースコード分析の結果，理解不足であることを示すコメントが見つかった．理解不足であることを示すコメントの例を，リスト 7 に示す．このような理解不足であることを示すコメントも，不適切なコメントかどうかのアンケート調査は行っていないが，今回の検出対象とした．

計測の方法は，検出すべきコメント及びコメントの総数を目視で確認する．その結果とツールを用いて解析した結果により，適合率及び再現率を計算する．

まず，設定ファイルの調整のためにあるプロジェクトに適用する．このプロジェクトの詳細を， 表 1 に示す．また， 適用した結果を， 表 2 に示す．このプロジェクトの 1 回目は，XML ファイルに記述した検出文字列のパターンに原因があり，再現率，適合率がともに満足な数字ではなかった．その後，検出文字列のパターン調整を行い，3 回目には，再現率が 100 %となり，適合率が 85.2 %と 8 割以上の適合率になった．

リスト 6. 生成されたパーサの使用法

```
public ArrayList<String> comments = new ArrayList<>();
CommentParser paser = new CommentParser(new StringReader(new String("//hello"));
try{
    comments = parser.comment();
}catch(CommentParser.ParseExcepiton e){
    e.printStackTrace();
}
```

リスト 7. 理解不足であることを示すコメント

```
//なぜか動く
//挿入する必要があるかもしれない
```

表 1. 初めに適用するプロジェクトの詳細

LOC	コメント総数	検出すべきコメント数
6854	1990	23

他のプロジェクトでも，設定ファイルが利用できることを確認するために，このプロジェクトで利用した設定ファイルを別のプロジェクトにも適用した．調整した設定ファイルを適用するプロジェクトの詳細を，表 3 に示す．また，別のプロジェクトに適用した結果を，表 4 に示す．実行回数 1 回目で，再現率が 100 %で適合率が 92.3 %と高い結果を得ることができた．良い結果を得られるように調整した設定ファイルが，他のプロジェクトにもそのまま利用できることが確認できた．

6. 考察

不適切なコメントを検出するツールは，他にも開発されている．中村氏 [9] は，Eclipse プラグインとして不適切なコメントを検出するツールを開発している．このツールでは，本論文と同様に XML ファイルを利用し検出文字列のパターンを定義することができる．しかし，パターンマッチによる正規表現は使えない．試作したツールでは，正規表現を利用し検出文字列のパターンの記述ができるので，柔軟に定義できる．

阿萬氏 [10] は，コメントが多くなっているソースコードには，コメントがなければ理解できないようなソースコードが多く存在し，そのようなソースコードがフォールトの温床になり，フォールトが増えているのではとの懸念から, コメントとフォールト潜在との関係に関する定量分析を行った．結果として，コメントの多いソースコードは，フォールト潜在率が高いという傾向が示された．本稿で試作したツールでは，特定のコメントを検出できる．不適切なコメントを検出することによって，フォールトの温床となっているソースコードを見つけることができる．見つけたソースコードを修正することにより，フォールトを減らしていくことができる．

開発者がソフトウェアプロジェクトを短期間でリリースするために，最適でない解決手段を選択することを表す造語として，技術的負債がある [11]．技術的負債には，不注意によって混入するものと，意図的に導入するものがある [12]．意図的に導入された技術的負債は，Self-Admitted Techical Debt と呼ばれる．これは，ソースコードコメントから見つけることができる．従来の研究では，Self-Admitted Techical Debt の検出は，目視に大きく依存していた [13]．そこで，Maldonado 氏は自然言語処理技術を利用し，自動的に Self-Admitted Tchical Debt を検出する方法を提案した [14]．しかし，[14] では再現率を 100 %にできていない．試作したツールでは，文字列検出パターンを用意することで再現率を 100 %にできる．

grep コマンド [15] やテキストエディタのテキスト検索機能を用いても，試作したツールと同様のことができる．しかし，これらは，ソースコードすべてを検索してしまうために，print 文にある自然言語及びメソッド名や変数名として利用されているものまで検出してしまい，適合率が低くなる．本ツールでは 一度全てのコメントを抽出し，抽出したコメントに対して検出文字列のパターンとマッチしているかどうかを判断する．このことにより grep より適合率を高くできる．

本稿の正規表現によるパターンマッチでは，適合率を 100 %にすることは難しい．今回，適用したプロジェクトにはパーサのコードがあり，それを説明するための EBNF(Extended Backus Naur Form) がコメントに記述されていた．このコメントに，なくてもよいという意

表 2. 初めに適用したプロジェクトの適用結果

実行回数	検出すべきコメント数	検出した全コメント数	検出できた検出すべきコメント数	適合率	再現率
1	23	31	20	64.5	87.5
2	23	29	23	79.3	100
3	23	27	23	85.2	100

表 3. 調整した設定ファイルを適用するプロジェクトの詳細

LOC	コメント総数	検出すべきコメント数
9394	1584	24

味を持つ正規表現の「?」があり，曖昧なコメントを検出するための検出文字列のパターンである「?」にマッチしていため，不適切なコメントして検出していた．適合率を 100 %にできるような検出方法の考案が，今後の課題である．

7. おわりに

本稿では，ソースコードの品質向上を目的として，特定のコメントを検出するツールを試作した．試作したツールを用いることで，ソースコードに存在する不適切なコメントやマーキングの役割をするコメントを検出できる．なお， 検出したコメントには，検出対象以外のコメントも含まれてしまうことがあるので，目視による確認が必要となる．また，新たな検出対象のコメントが出てきた場合も，検出文字列のパターンとして追加できる．試作するツールは，正規表現を利用しているので様々なコメントに対して柔軟に対応できる．

株式会社スカイコムの協力で LOC が 5000 行以上のプロジェクトを対象に試作したツールを適用し， 適合率及び再現率について調査した．調査した結果，検出文字列のパターン調整することによって，再現率は 100 %を，適合率は 80 %以上を，それぞれ達成した．また，一度設定した XML ファイルは，再利用可能なことが確認できた．本ツールを利用することにより，特定のコメントを検出できる．マーキングの役割をしているコメントや不適切なコメントを減らしていくことで，ソースコードに存在する課題の削減や理解容易性の向上が図れる．

以下に，今後の課題を示す．

- 適合率の向上
  本稿の方法でも，正規表現の記述方法の工夫や検出対象のコメントによっては，適合率を 100 %にできる．しかし，正規表現によるパターンマッチのみでは限界がある．例えば，学習させたデータに基づいて，適切なコメントか判断する検出手法の提案などが必要となる．

- 処理そのものを説明しているコメントのみの検出
  処理そのものを説明しているコメントに使われる単語は，他のコメントにも頻繁に現れる．そのため，処理そのものを説明しているコメントをパターンマッチで検出しようとすると，多くの適切であるコメントを拾ってしまう．コメントの周辺の情報を取得し，前後の繋がりをもとに，コメントの内容が処理の説明かどうか判断する検出手法の提案などが必要となる．

- メンテナンスされていないコメントの検出
  適切に書かれたコメントでも，メンテナンスされていない場合は，不適切なコメントになりううる．バージョン管理ツールである git などの情報を利用し，コメントの周辺が変更されているにも関わらず，コメントが修正されていないところを検出するなどの方法で，メンテナンスされていないコメントを検出できると考えている．

参考文献

[1] StyleGuide: Google C++ Style Guide. `https://google.github.io/styleguide/cppguide.html`

[2] XML:“Extensible Markup Language (XML) 1.0 (Fifth Edition)”, W3C Recommendation, 2008

[3] 甲斐秀一, 田上諭, 片山徹郎, “ソースコードに存在する不適切なコメントを検出手法の適用事例”, ソフトウェア・シンポジウム *SS2017* 発表予定, 2017.

表 4. 調整した設定ファイルを適用したプロジェクトの適用結果

実行回数	検出すべき検出対象のコメント数	検出した全コメント数	検出できた検出すべきコメント数	適合率	再現率
1	24	26	24	92.3	100

[4] クラス Pattern: Pattern (Java Platform SE8). https://docs.oracle.com/javase/jp/8/docs/api/java/util/regex/Pattern.html.

[5] JavaCC: JavaCC - The Java Parser Generator. https://javacc.org/.

[6] ISO JTC1/SC22: “ISO 14977:1966”, International Organization for Standardization, 1996.

[7] 青木峰郎: “普通のコンパイラをつくろう”, ソフトバンククリエィティブ株式会社, 2009.

[8] クラス DocumentBuilderFactory: DocumentBuilderFactory (Java Platform SE8). https://docs.oracle.com/javase/jp/8/docs/api/javax/xml/parsers/DocumentBuilderFactory.html.

[9] 中村大賀, “ソースコードコメントのテキスト分析”, 社団法人 情報処理学会 研究報告 *SE-163*, pp.287–294, 2009.

[10] 阿萬祐久, “オープンソース・ソフトウェアにおけるコメント記述及びコメントアウトとフォールト潜在との関係に関する定量分析”, 情報処理学会論文誌 *Vol.53 No.2*, pp.612-621, 2012.

[11] W.Counnigham, “The wycash portfolio management system”, *in Addedum to the Proceedings on Object-oriented Programing System, Languages, and Application*, pp.29-30, 1992.

[12] M.Fowler: Techical debt quadrant. https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

[13] A.Potdar, and E. Shihab, “An exploratory study on self-admitted techical debt”, *IEEE International Conference on Software Maintenance and Evolution*, pp.91-100, 2014.

[14] E. d. S. Maldonado, E. Shihab, and Nikolaos Tsantali, “Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt”, *IEEE Transactions on Software Engineering*, DOI:10.1109/TSE.2017.2654244, 2017.

[15] Grep: GNU Grep 3.0. https://www.gnu.org/software/grep/manual/grep.html

ソースコードに存在する不適切なコメントの検出手法適用事例

甲斐 秀一 [1)]　　田上 諭 [2)]　　片山 徹郎 [3)]
1) 株式会社スカイコム h-kai@skycom.jp
2) 3) 宮崎大学 工学研究科 {tanoue, kat}@earth.cs.miyazaki-u.ac.jp

要旨

本研究では、ソースコード（最終的にはソフトウェア）の品質向上を目的とし、ソースコード内に存在する不適切なコメントを検出可能なツール(以下、本ツール)の開発および評価を、宮崎大学と共同で行った。不適切なコメントとして検出した箇所のソースコードを見直すことによって、品質およびメンテナンス時の作業効率の向上を図ると共に、不適切なコメントが及ぼす影響やコメントの重要性を理解するための教育ツールとしても利用可能なものを目指して取り組みを行った。

1. はじめに

ソースコードに用いられるコメントは、プログラムの処理には直接影響を与えないため、内容について重要視されにくい傾向にある。当社でのソースコードレビュー時においても、コードに注視してチェックを行うため、コメント内容の細かいチェックまでは実施しない傾向があった。そのため、不適切なコメントを含んだソースコードが、知らぬ間に徐々に蓄積されていた。読み手にとって理解の妨げとなるような不適切なコメントが存在する場合、メンテナンス時の作業効率を悪化させ、バグの混入に繋がり、結果として製品の品質を低下させる可能性が考えられる。

そこで本研究では、ソースコードの品質向上を目的として、不適切なコメントを容易に検出する仕組みを適用した場合の効果を検証するため、本ツールの開発および評価を行った。

2. 不適切なコメントの検出方法

検出すべき不適切なコメントについては、当社開発部門内でアンケート調査を実施して定義した。アンケート調査の結果、社員が不適切と考えるコメントのほとんどが、単語によってパターン分けできることが判明した。(詳細はスライド参照)

以下に検出したパターン例を挙げる。

■ 意図が曖昧なコメント
(例) 「暫定」「仮」「未定」

■ 周辺処理を理解せず追加したと推測されるコメント
(例) 「なぜか」「おそらく」「かもしれない」

前述のアンケート結果をもとに、不適切になり得る単語を一覧化した定義ファイルを用意し、定義した単語を全コメント中から検出することで不適切なコメントの検出を行うこととした。なお、定義する文字列には正規表現も使用可能にしているため、検出文字列のパターンは柔軟に対応可能である(例えば、日付フォーマットを含むコメントの検出等も可能)。

3. 本ツールの評価

当社開発部門が保有する複数ソースコードに本ツールを適用した結果、主に以下の効果が得られた。

■ 不適切なコメントの目視確認が不要となり、過去の実装も含めて容易に見直しが可能となった。
■ 検出対象コメントの定義を柔軟に拡張できるため、ツールを実行して検出結果の傾向を分析し、定義ファイルを拡張していくことで検出精度が上がった。
■ 検出した不適切なコメントの位置から、その周辺処理に潜在する課題や見直しが必要な可能性のある処理の検出に成功した。

(詳細はスライド参照)

4. おわりに

不適切なコメントを検出し、その周辺箇所を見直すことによって、品質およびメンテナンス時の作業効率の向上に繋がることが確認でき、改めてコメントの重要性を認識した。また、本ツールを利用することによって、効率良く不適切箇所を検出できることを確認した。

今後は、不適切なコメントだけでなく、コードとコメントが一致していない箇所の検出を可能とするなど本ツールを拡張することによって、さらなる開発全体の作業効率化を促進したい。

「あるある診断ツール」による課題の可視化と収集データの分析事例

室谷　隆
TIS株式会社
Muroya.takashi@tis.co.jp

要旨

昨年のソフトウェアシンポジウムにて,『「保守あるある診断ツール」による保守課題の可視化事例』というタイトルで, ツールの検討過程, コンセプト, 使い方の知見などを紹介させて頂いた. この後運用主体の業務を診断する診断項目を作成, 使い勝手の改善を行ない, 2016 年末までに 220PJ(565 名)に適用し, 約 13 万件のデータを得た. この診断結果データの分析を行い, 興味深い知見が得られたため事例の追加報告として発表する.

1. ツールの概要

「あるある診断ツール」はセルフアセスメントツールの一種であり, 以下の特徴を持っている.

1. 短時間（約 30 分）で実施可能
2. 生産性や品質の低下を招いている問題事象をチェックするだけ
3. 保守/運用を特徴付ける 8 つの視点を持つ

2. 設問の特徴

世の中に存在するセルフアセスメントツールは, アセスメントモデルのプラクティスが実施できているかを直接問うものが多く, プラクティスを理解していないと回答が難しく, 時間がかかるものであった.

このため, 設問の内容を,「共通フレーム 2013」のタスクや「SPEAK-IPA」のプラクティスを身近で発生している問題事象に変換することで回答し易くした. [1]

3. 利便性の改善と適用範囲拡大

保守 PJ へ本ツールを展開する過程において, 運用主体の PJ や顧客から, 運用版設問の作成要請があった. 設問内容を検討した結果, ITIL のプロセスを参考にして作成, 展開した. また, 利用者から 30 分で終了できなく, 使いづらいとの意見も多く寄せられたため利便性を向上させるよう変更を行った.

4. データ分析結果

全診断の内, 直近のデータ(2016 年度上期実施分)48PJ, 268 名, 約 6.8 万件を分析した結果, 当初目論見通りの効果が確認でき, 更に以下の知見が得られた.

1. 定性分析結果

経験則として認識されている保守課題が, 分析結果からも裏付けられた.

・属人化, 有識者不足
・ノウハウの蓄積がない
・計画外作業（突発作業）が多発　等

2. 定量分析結果

設問毎の平均値を横軸に, 回答者別の標準偏差を縦軸にして, 散布図を作成し, ゾーン分けして分析.

・A ゾーン:
課題感が低く(高い平均値), PJ 間で差異が大きい
・B ゾーン:
課題感が低く(高い平均値), PJ 間で差異が小さい
・C ゾーン:
課題感が高く(低い平均値), PJ 間で差異が小さい
・D ゾーン:
課題感が高く(低い平均値), PJ 間で差異が大きい

PJ の優れた事例の横展開で, 改善がスムーズに進む D ゾーンに多くの課題が集まっている事が分かった. 今後, この事例の横展開を, 保守/運用の標準プロセスの構築と合わせ, エンハンスメント（保守）革新という活動を進めて行く事となった.

参考文献

[1] 1. 独立行政法人　情報処理推進機構　技術本部　ソフトウェア・エンジニアリング・センター編、
共通フレーム 2013
2.独立行政法人　情報処理推進機構　技術本部　ソフトウェア・エンジニアリング・センター
SPEAK-IPA（Rev.1.0.2.0）

パッケージ製品におけるソフトウェア保守情報の活用事例

加藤 英之
東芝ソリューション株式会社

増井 和也
ソフトウェア・メインテナンス研究会

要旨

開発が済んだソフトウェアは，システム上で稼働を開始すると，ある種のソフトウェア保守[1]対応が必要となる。稼働中ソフトウェアへのこの保守対応（含む利用者サポート）は，利用者にとって無償か有償かに関係なく，稼働が終了するまで何らかの形で継続する。その継続期間が時として 30 年以上となる事例もある[2]。保守対応の間で得られる情報は，処理機能の操作等の Q&A，クレーム，要望，障害情報(事象，原因，回避策，再発防止策等)，修正版リリース情報，保守作業のための運用停止情報，修正に伴う仕様・設計・プログラム・テスト仕様・作業エビデンス・承認過程など，1 件の保守対応にもさまざまで，いわゆる 5W1H 情報を含み，多岐にわたる。ソフトウェアの開発完了時の完成文書が完成時点の状況を示す比較的均一で静的な情報であるのに比べ，保守対応情報は時間軸を持った動的でかつ多様な情報の集まりといえる。

これらの情報には，システム，ソフトウェア，人間系の問題や障害といった瑕疵情報を含み，その量や種類の多さは一般的にシステムの信頼性のあるべき姿から誇らしいことではないとの認識[3]がある。そのため，保守対応情報は社外への開示はもちろんのこと，社内の他部門への開示も統計処理した一部データを除き，生の個別保守情報（特に恥かしい事例と印象付けられるようなもの）が積極的に開示されることは少ない。

本論文は，パッケージ製品の顧客別アドオンソフトウェア保守専任チームが管理する保守情報を，外部への開示は制限しているものの，厳重なセキュリティ権限管理や顧客との機密保持契約遵守の下，可能な範囲で部門間共有を積極的に行い，さまざまな効果が出ている事例を総括的に報告するものである。

1. 対象ソフトウェアシステムの概要

本論文の対象システムは大規模企業グループ向け Web ベースの人財管理パッケージシステム[5]である。対象システムは，単一法人向けとして人事・給与機能のみの初期バージョンが 1999 年にリリースされてから，ライセンス契約数を拡大しつつ 18 年余りになろうとしている（図 1-1）。

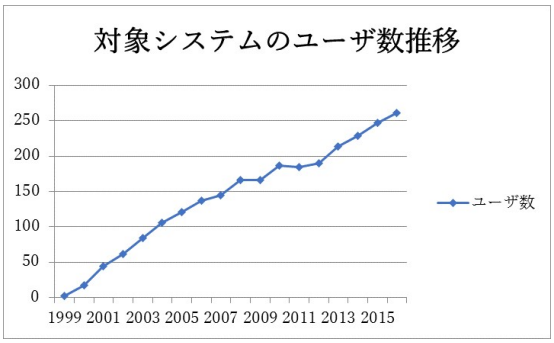


図 1-1　契約顧客数の推移(1999-2016)

本体製品のメジャーバージョンは現在 V6 である。初版より現在まで，グループ企業を統括する複数法人対応の強化，基盤技術や動作可能環境の最新化など，大幅な内部設計更新は複数回行っている。また，企業等の人的資源管理の周辺業務をサポートするオプションサブシステムの製品レパートリの拡張も継続して行っている（図 1-2）。現在のオプションライセンスサブシステムは，従業員の教育管理，能力・目標管理，タレントマネージメント管理，e-ラーニング，人材育成など計 9 のオプション機能を持つまでに進化している。

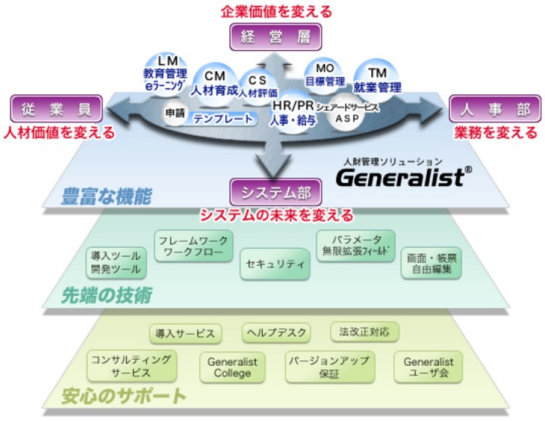


図 1-2 対象システムのサブシステム拡張（進化）

なお，多様な業種に広がる顧客企業等の業態や勤務形態にきめ細かく対応するため，顧客の求めに応じ，個別アドオン機能のライセンス提供の提案と契約締結により顧客別アドオン開発・導入を行う場合がある。アドオンソフトウェア契約自体は，個別顧客の導入効果や利用満足度の向上及び本製品提供側の売上増となり，双方にとってメリットがある。

しかし，製品及びアドオンのそれぞれのソフトウェアを併せた場合は，製品のみの導入に比べ，保守対応の困難性が増す。たとえば，対象システム製品のメジャーバージョンアップ，PC（端末）の OS やブラウザなどのアップグレードなど，周辺環境の変化に個別保守対応が必要となった場合が発生する。アドオンソフトウェアの保守対応を円滑に提供していくには，製品・アドオンの各機能ソフトウェア間の関連を確実に制御できる高度なソフトウェアの保守対応やソフトウェア構成管理技術が必要となってくる。

本論文は，対象システム製品の保守情報管理を通じて保守対応の活用事例および課題とその対策について述べる。

また，このアドオンソフトウェアの保守対応も対象に含めて述べることにする。

2. 対象システムの規模と顧客保守契約

現状の対象システムの規模は，製品本体部分，顧客別アドオン部分，本製品特化した保守作業ツール等を併せ，　1 千万行を超える。なお，この行数はコメント行も含めて数えている。

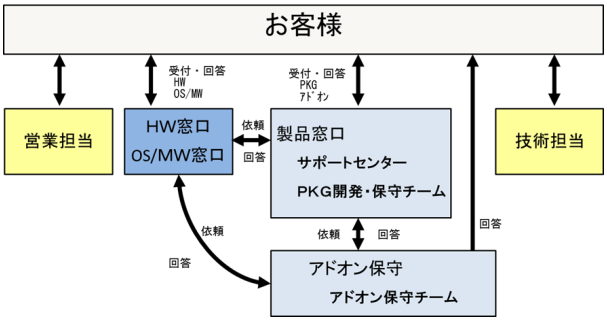
顧客が本製品を導入する場合，本体ライセンス費及び，必要とするオプションライセンス費，アドオン機能提供費を支払う。また，その後の保守費として導入した部分に対応する年間保守費が発生する。しかし，製品戦略上，主要パッケージ製品 [6] の保守サービス費よりも抑えられた保守契約になる傾向がある。さらに既存顧客が今後の IT 予算計画立案を行う際には，当該顧客から保守費減額交渉要求が発生することが珍しくない。

3. 対象ソフトウェアの保守発生状況

対象システムで発生する保守事案（以下，インシデントと呼ぶ）の情報は，保守情報管理システム（以下，単に本管理システムと呼ぶ）に登録する。登録保守情報の範囲は，JIS X0161 で定義された保守プロセスで入出力されるデータや情報だけでなく，製品の仕様・動作環境・顧客情報・一部営業情報，アドオンの機能，操作，制限，注意点，事例，追加・改良見積もりなどの問合せ情報に関するものなども含む。本管理システムは，インシデント情報の登録自体が最終目的ではなく，問合せに対する回答を行う場合に，本管理システムの過去の問合せを検索することで，ワンストップで回答や対応の情報にたどり着けることを目的としている。

4. 対象システムのソフトウェア保守体制

製品本体及び各オプション製品単位に保守担当チームがあり，また顧客別のアドオンを束ねて保守する担当チームが別にある（図 4-1 参照）。以下，アドオン保守統括チームが保守情報をどう活用しているかを中心に報告する。



（注）HW:ハードウェア，OS:オペレーティングシステム，MW:ミドルウェア，PKG:パッケージ製品

図 4-1　対象システムの保守体制のイメージ図

5. 保守情報管理システムの概要

現在，本管理システムには対象システムの全製品，アドオン機能に関するインシデント対応情報やそれに問い合わせに回答するための情報が約 7 万件超が登録されている。これまでの登録件数の推移を図 5-1 に示す。

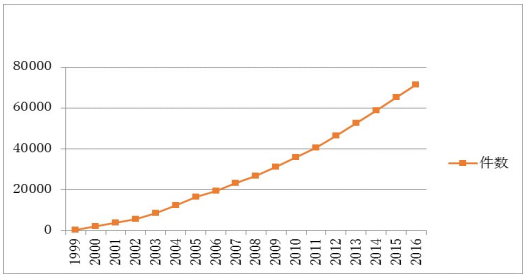


図 5-1　保守情報管理システムの登録件数推移

本管理システムの主な機能は，登録画面，登録済みインシデント情報の検索画面，検索で絞り込んだ情報の CSV 出力機能などがある（図 5-2 参照）。

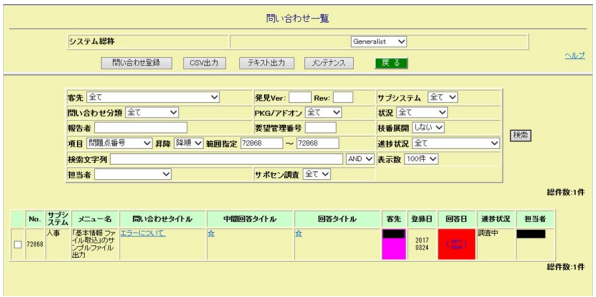


図 5-2　保守情報管理システムの画面イメージ

出力した情報の加工は，CSV ダウンロードした側の業務用 PC で行う。利用上の制限は，ログイン画面の表示が許可されたドメインに限定されている。また，ログインはロックアウト機能をもつパスワード管理され，登録 ID かつ適正なパスワード以外は利用ができないか，権限によって検索範囲が制限される。また，誰がどんな操作をしたかのログも取られている。

6. 保守情報管理システムで管理する項目

本システムで管理する主な項目を表 6-1 に示す。特に，顧客が現在利用している製品リリースバージョンは重要で，各種派生資料（含む CSV 出力後の加工資料）の更新にも注意を払っている。顧客別製品構成管理情報が別にあると，インシデント情報以外の資料も相互に閲覧が必要になり，必要な確認作業が増加する。

7. 本管理システムの情報管理者と入力者

本システムへの登録（初期入力）は，ヘルプデスク，顧客から直接入ってきた情報を登録する保守担当者によるものが 90%以上である。その他全国にいる営業担当者，受注前提案 IT 技術者等が登録する場合がある。また，回答や再質問のような追加情報入力は，それぞれの回答担当者や回答を受けた側が行う。

なお，回答や対応のやり取りは，きめ細かく迅速に行うが，クローズは特に急がない（早期クローズを最大目標とするようなことはしない）。インシデントのクローズ（完了）の目的は「終わったことにしよう」と安心や達成感を与えるものではなく，必要なインシデント対応記録を漏れなく記録できたことを十分確認できたことを証明するためのものと考えている。そのため，いったん追加入力は収まったが，何年かして，不再現の現象が再発して，追加現象発生情報の入力がなされることも珍しくない。クローズされていると（もう済んだインデントだから余計なことをしないという）情報入力の漏れの発生リスクが高くなる（表 7-1 参照）。

また，たとえクローズ状態でも，クローズ案件を除外しなければ，検索対象から外れることはない。また，クローズ状態から非クローズ状態に戻すことや，クローズ後の入力も特に制限なく可能としている。滞留残件（非クローズインシデント）数の増減のみで一喜一憂するようなソフトウェアの保守作業品質の管理発想は，結果的に必要な保守記録をすべて記録しようとする努力を阻害することになり，保守対応プロセスとしての成熟度は高くないと考えている。

表 6-1　保守情報管理システムで管理する主な項目一覧

No.	項目	意味・目的・管理上のポイント
1	問合せ番号_枝番	インシデントの識別
2	インシデント登録日時	発生日時と登録日時に差があるとその原因を確認
3	発生サブシステム	図1-2のサブシステムを識別
4	顧客名（保守契約先）	内部発見の場合は未記入とする
5	保守契約ID	内部発見の場合は未記入とする
6	発生機能名	サブシステム内の機能
7	発生利用法人・部門	保守契約者と利用者法人は1対 n の場合
8	インシデント種別	不具合／要望などの種別
9	発生Ver/Rev	発生バージョンとレビジョンの特定
10	インシデントタイトル	インシデント内容の見出し
11	報告者名	保守顧客の場合は，登録者
12	インシデント内容	詳細なインシデント内容
13	発生条件	インシデントの発生条件／発生を回避できる条件
14	操作手順	インシデントが発生する具体的操作
15	想定している結果	インシデントとならない本来の結果
16	動作環境	主にクライアントの機器/OS/ブラウザ/ネットワークなどの環境
17	製品/アドオン	製品本体機能/アドオン機能かの区分
18	インシデント発生日時	複数の場合は最初の日時
19	インシデント分類	スケジュール処理／随時処理の区別
20	添付文書	画面キャプチャやダンプデータなど，より詳細なインシデント発生状況を示す資料
21	報告者回答希望日	運用上の影響を知るために，報告者には正確に入れるよう理解を促している。
22	修正パッチリリース予定日	保守担当者側が判明したら記入
23	内部パッチ修正予定日	内部検証作業開始可能日
24	顧客先行リリース	あり（インシデント発生顧客の運用影響が大きい場合のみ先行リリースを検討）／なし
25	顧客先行リリース予定日	顧客先行リリースなしの場合は未記入
26	先行内部修正予定日	先行リリースありの場合，顧客を想定した内部検証作業開始可能日
27	不具合混入VER	過去バージョンの潜在原因を区別
28	不具合混入工程	社機開発からの潜在要因の場合は，開発工程内の工程，後からの保守対応の場合は，当該問合せ番号を記入。
29	対応担当者	インシデント対応担当者
30	対応進捗状況	定められた保守プロセスのタスク完了状況
31	要望管理番号	要望は別に管理している
32	重要注意区分	今回の保守対応や今後の保守対応において，特段注意すべき項目
33	要対応VER	リリース中のバージョンで対応要のバージョンをすべて書く
34	備考	インシデント対応で問題となった点を記録に残す
35	中間回答タイトル	中間回答が複数に発生する場合，問合せ番号の枝番で管理
36	中間回答日	
37	中間回答者	
38	中間回答内容	内容のほか，次の回答予定日を必ずいれる。本中間回答への質問も受け付けることを入れる
39	本回答タイトル	タイトルには，原因を簡潔に述べることが多い
40	本回答日時	
41	本回答者	
42	本回答内容	本回答であることをしっかり述べると示し，回答に不満がある時は，再問合せを受け付けることを示す。本回答の再質問は別問合せ番号で管理。
43	内部要員保守担当工数	製品またはアドオンの保守担当で社内要員工数（人・時間）
44	関連ベンダ工数	本インシデントで外部調達した場合の工数（人・時間）
45	本修正内容	最終的な対応内容を記述する
46	本リリース予定Ver	
47	本リリース対応予定	
48	本リリース予定日	
49	本リリース日（実績）	
50	本リリースVer/Rev	
51	本リリース備考	本リリースが根本解決であることの説明等を記入
52	マニュアル反映	済／未完のいずれか
53	FAQ反映	済／未完のいずれか
54	マニュアル反映内容	
55	FAQ反映内容	
56	リリースノート記述内容	
57	HP通知内容	
58	更改資料記載不要	有無の区分
59	外部発見	有無の区分
60	中間回答添付文書	
61	本回答添付文書	

表 7-1　インシデントの早期クローズのデメリット

No.	担当者がインシデントクローズを急ぐ誘惑要因	残インシデントを無理に減らすデメリット	対策
1	未クローズのままだと，インシデント対応を放置していると勘違いされる	発生件数と未クローズ件数の少なさ／多さのみで頑張っている／いないの評価（各案件の対応困難度を度外視）	保守プロセスを共有し，インシデントの対応難易度を共有し，途中進捗や進み具合を定量化
2	インシデント対応残（未クローズ）件数で評価される	残件多で担当者は努力不足と結びつける評価（各案件の対応困難度を度外視）	クローズする条件をインシデント単位に設定
3	長期間クローズしないインシデントはフォローを受けやすい	フォロー者の顔色を窺うだけの対応で，最適対応とならない	（保守対応技術を理解せず）数字だけをフォローするチェック者を作らない
4	直ぐクローズできたインシデントは軽微なインシデントと判断されやすい	軽微なインシデントに安易に分類してしまう	保守プロセスを共有し，インシデントの対応難易度を共有し，途中進捗や進み具合を定量化
5	クローズしてしまえば，フォローを受けることはまずない	クローズ案件は「済んだ」「終わった」と忘れ，同件でも改めて原因調査	クローズしたインシデントも検索時の条件は検索対象となるのが既定値とする
6	管理者も品質保証部も本音は早くクローズすることを望んでいるはずだ	残件の数の少なさ／多さのみで一喜一憂する	（保守対応技術を理解せず）数字だけをフォローするチェック者を作らない
7	細かく記録を残すと後で全体の傾向整理や分類が大変になる	安易な分類をして，原因の深堀がなされず，以降の参考にならない記録となる	分類は区分や分類コードのみでなく，文字データ内のキーワードでも判断する
8	クローズしてしまえば，当該インシデント完了をアピールした報告ができる	完了100点／未完了0点の安直な評価	（保守対応技術を理解せず）数字だけをフォローするチェック者を作らない
9	すでに対応残が多くあるので，残しておきたくない	発生件数とクローズ件数のみで頑張っている／いないの評価（各案件の対応困難度を度外視）	なかなかクローズしないインシデント対応を担当者任せにせず，チームで一緒に残対応を片付ける
10	インシデント対応残の多さで集中対応できず，気分が重くなる	発生件数とクローズ件数のみで頑張っている／いないの評価（各案件の対応困難度を度外視）	なかなかクローズしないインシデント対応を担当者任せにせず，チームで一緒に残対応を片付ける
11	しばらく現象再発やお客様からフォローがないのでいったんクローズしたい	ソフトウェアの障害は自然に解消することはないが，それを期待してしまう	操作ミスなどが確認されるまで安易にクローズしない
12	このインシデントの根本解決は大変な工数が必要，担当者ベースではクローズしたい	担当者一人でできない規模のものを担当者にやらせようとする	エスカレーションして別管理にするが，元インシデントは根本対応が終わるまでクローズにはしない

8. 保守情報の活用事例(課題と活用効果)

（課題１）緊急事態に常に対応できる専任体制は保守コスト増大のリスクがある。

（対策１）顧客から保守サービスの費用対効果の向上（コスト低減とサービス向上の両方を進める）を求められる。コストの多くは緊急障害に備えた体制維持に多くが取られる。緊急障害の発生予測精度を高めるため，過去インシデント情報（情報精度が高いことが前提）を分析し，事前に対応準備をしておく。レビジョンアップ後のデグレード（機能低下），顧客担当者の異動に伴う不慣れ，顧客端末ＰＣ環境の変化，季節的な利用機能の変動など，過去に発生した構成の変更・変化後のインシデントの発生傾向を分析し，緊急インシデントの防止対策（事前に判明した障害や利用制限のアナウンス，障害発生時の利用者対処方法，障害を防止する運用方法や設定変更のアナウンス，初心者が間違いやすい操作の事例紹介，対応の事前準備。）を検討し準備することで発生コストの計画を行い，予定外コストを抑えるようにする(図 8-1)。

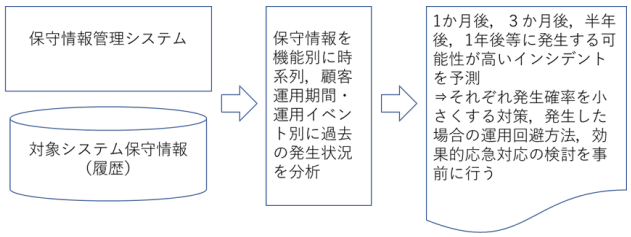


図 8-1　保守情報の活用例１：今後発生するインシデントの予測精度向上

（課題２）保守対応の終わりが不明確。やって当たり前の世界で業務アピール度が低い

（対策２）重大インシデントが発生すると営業部門や品質保証部門が客先窓口になって対応する必要が発生することも珍しくない。この場合，保守対応チームは社内においても，フォローを受けるチームの中心となる。また，障害解消に向け目覚ましい活躍をしても，その評価はやって当たり前というものが多い。特に大きな部門として，開発組織と一緒の場合，顧客，営業部門，品質保証部門から見たとき，そのような障害を発生させた責任追及がメインとなり，努力不足または対応能力が低い要員が対応した場合には復旧不可能であった障害を，素早く復旧させた保守対応能力（技能）に対しての評価を受けることが少ない。

保守担当者が開発の一員（障害を起こした犯人）と見られ，そのような不当な評価を受けることがないよう，過去の障害対応の平均工数，復旧までの期間の短縮（MTTR: Mean Time To Repair の短縮，予防保守による障害発生数の削減（MTBF: Mean Time Between Failure の伸長）の傾向のデータを本管理システムから採取・分析した。そして，従来の同様の障害に対して，どれだけ少ない工数，短い時間で解消できたか，予防保守で同種障害を過年度に比べどれだけ少なくできたかをアピールすることを心掛けた。

さらに，この他に月単位，半期単位，経年で総インシデントの集計，分析を行い，インシデントの状況とオプション製品の保守契約開始の関係等，営業部門への情報提供を行い，既存顧客の営業活動重点分野を決める参考にしてもらう。また，インシデントの時系列分析（顧客別，全顧客）結果と，インシデント対応残件状況報告の定例会開催を顧客に提案する。この時，営業や提案ＳＥも同行を調整し，

顧客部門から次期予算の提案計画などを構想段階から話題として聞くことができ，準備を行い正式決定されたら直ちに最適な提案を提供できる可能性が高くなる(図 8-2)。また，製品に対する顧客の評価や製品やサービスの課題もタイムリに話を聞くことができ，製品やサービス改善を顧客志向で進めることができるようになった。

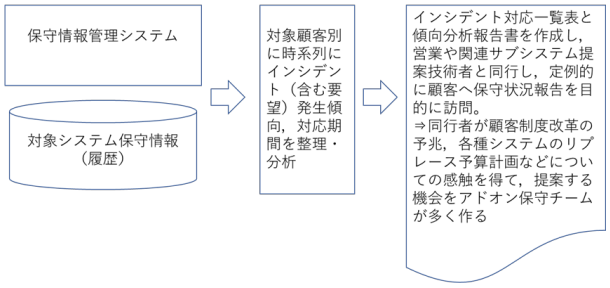


図 8-2　保守情報の活用例 2：保守作業の価値アピールに活用

（課題３）保守作業の作業環境改善という名目での予算化・調達には困難が伴う

（対策３）製品の競争力を高めるための機能強化調達予算化は承認されやすいが，一般的に保守対応コストの削減，保守サービスのタイムリさ向上，保守担当者の作業負荷低減の体制強化など，保守作業整備を達成するための調達予算をあらかじめ入れておくことが難しい場合がある。特に企業業績が思わしくないときはなおさらである。

そこで，本管理システムから，インシデント対応工数やコストデータを分析し，インシデント対応で費用対効果が一番高い調達先や内製対応した時との比較，現行の体制でのバックログの増加状況をデータで示すことによって，調達が必要な根拠を示すことができるようになった(図 8-3)。たとえ，今期予算がなく，新たな調達が難しい場合でも，これらのデータから来期の予算確保の裏付けデータとして活用ができるようになった。

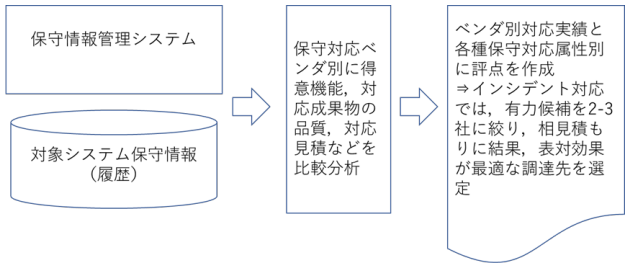


図 8-3　保守情報の活用例３：保守作業改善の予算化に活用

（課題４）障害が発生すると緊急の原因調査・報告・謝罪の作業が発生。モチベーション向上が困難

（対策４）緊急事態の発生は，本来計画的に作業をすることが是として教育されてきたソフトウェア技術者にとっては，教科書にもポジティブに書かれることもなく，悲惨で計画的でない作業に感じることが多い。特に，緊急対応の経験が浅いソフトウェア技術者にとっては，障害対応優先という大きなプレッシャーを感じながらの作業をつらく感じることもある。

一方，中堅クラス以上の技術者の場合は，早期の障害解消対応を求められるだけでなく，障害の原因や再発防止策の報告書の原稿作成対応を迫られる。また，自身が原因でない場合でも，利用者，顧客，営業部門，品質保証部門から厳しい叱責を受けなければならない立場となる場合がある。

このような場合も，本システムに蓄積されている障害対応記録から最適と考えられる対応プロセスを関係者で選択，共有，対応することで，対応の進捗の同一認識と対応作成物の本システムへの登録で，報告資料の定型部分の流用（文章部分のコピペはご法度）ができ，進捗可視化が可能となる。結果として，次にすべことや対応に苦慮する場合の相談先が明確になり，対応ミスが減る。以前の類似障害対応に要した期間や工数に比べ早く・安く・確実に対応できたことを実感できるようになり，障害対応作業においても達成感やチームワーク感を得られるようになり，メンバの士気が向上した(図 8-4)。

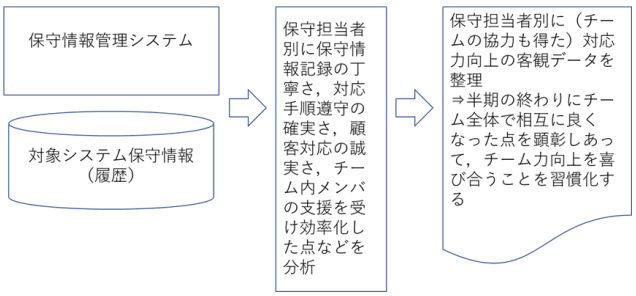


図 8-4　保守情報の活用事例４：保守要員のモチベーション向上に活用

9. 管理システム運営上の課題と対策

9.1. 担当者の初期入力と情報更新の負担感

本管理システムは，インシデント対応や各種問合せ対応の記録が必要十分に行われてこそ意味がある。たとえ，10万件近い保守対応記録があったとしても，それらの入力内容にばらつきや重要な記録項目の入力に漏れがあると利用価値が小さくなる。

そのためには，重い緊急障害対応（入力する時間がない）

や逆に取るに足らないと感じたインシデント対応（こんな些細なことはいちいち入力する必要を感じない）の場合，入力自体を強制されるとソフトウェア技術者にとっては大きな負担感を感じることが多い。

それを解消するため，入力を個人の作業義務として位置付けるのではなく，定例保守打ち合わせ時に，各担当から必要事項を報告し，互いの状況を共有する。突然参加したメンバには無駄や退屈と感じることが多いが，それは保守チームの一体感がないことから発生すると動機づけをする(図 9-1)。

保守技術者の仕事は担当割り当てを受けて，誰ともコミュニケーションをせず，ひたすら対応する個人プレーの仕事ではない（過去対応事例を効率的に活用した保守プロセスの理解）という認識を持ってもらうようリーダが指導する。入力を誰かに任せ（振）れば，もっと短い会議時間になると考えるのは，結局保守情報が共有されず，情報内容の不正確さ，陳腐化，活用の形骸化の要因となりやすい。

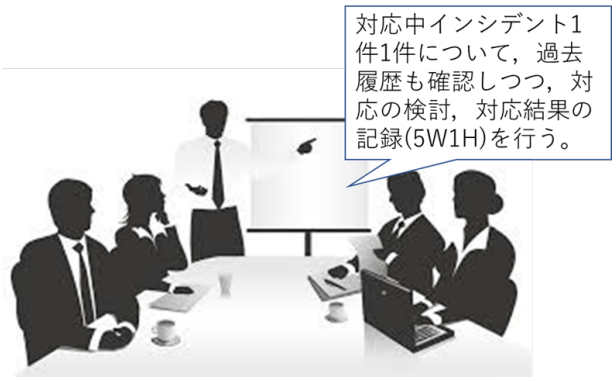


図 9-1　インシデント棚卸ミーティングのイメージ

9.2. コスト障壁

予定外コストの多くはクレームや緊急作業依頼の対応により発生する。クレームや緊急作業依頼は，客先からだけでなく，社内の営業部門，提案ＳＥのほか，品質保証部門，セキュリティ統括部門，知的財産部門，その他監査部門などのスタッフ部門からも来る。

これらの対応記録も本システムに登録し，過去どのような対応をして，どういった対応課題が発生し，どうすれば効率よく対応できるかのノウハウを蓄積するようにする。そして，営業担当者や別システム提案者が顧客訪問する前に，当該顧客のインシデント情報，最近インシデント対応状況を閲覧することを関係部門と調整することで，保守チームの存在価値を高め，営業や提案ＳＥにファンや理解者を増やし，保守チームへの体制強化や作業改善の提案の味方になってもらうよう働きかける。

さらに，顧客が設定する運用系パラメータや開発経験のない他の製品のアドオンソフトウェアの保守も第三者として保守担当を引き受けることを行い，コストメリットをアピールし，コストの問題を削減していこうとしている(図 9-2)。

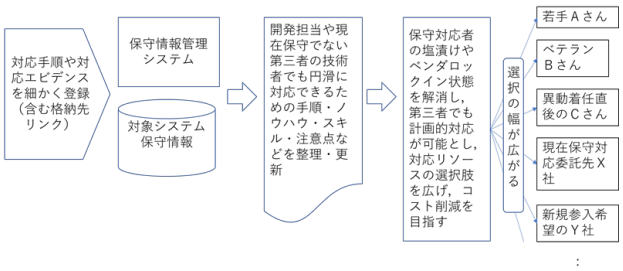


図 9-2　コスト問題への対応

9.3. 製品統括責任者の役割

本対象システムの製品の収益を統括する責任者は，新規導入顧客の増加によるライセンス収入，既存顧客の保守料収入が主な収入となる。保守料収入は安定的な収入と考えられるが，新規導入顧客のライセンス収入はどうしても変動が避けられない。そのため，期初には保守コストを抑えて，本来保守費として計画している以上の利益を出し，変動が避けられない新規ライセンス収入が下振れしても利益予算達成が可能になるようにすることになる。保守コストを抑えた計画を立てることにより，期初に発生したインシデント対応への臨時調達も含めた手厚い対応が手薄になる可能性がある。ソフトウェア保守部門が独立していれば，そんなことはできないはずであるが，現状はソフトウェアに関する部門は販売・開発・保守も一緒になっているケースが少なくない。

このような場合も，本管理システムを活用している営業担当者の活動状況の把握や実際の状況を知るための連絡を取り合い，プロスペクト（受注見込み案件）の受注可能性のリアルな状況を入手する。また，営業活動にも顧客から顔を知られている保守チームのメンバが積極的に協力し受注活動支援も行い，結果として製品統括責任者の防衛的対応を防いでいる。

10. 結論

ソフトウェア保守作業やその作業から発生する記録情報は多くの場合，隠される傾向にある。しかし，そういった保守情報は顧客満足度，顧客サービスレベル，製品品質状況，弱点対策分野絞り込み，周辺分野の製品開発参考データなどに有効で貴重なデータである。本論文では対象となるパッケージシステムで管理している保守情報の内容とその活用方法・課題等について説明した。ソフトウェア保守チームがこの情報を管理し，営業部門，提案部門，品質保証部門，開発チーム，パッケージ保守チーム，ヘルプデスクチームなどと共有することにより，次のビジネス拡大に的確につながる糧の一つとなることは間違いない。

そのためには,保守情報をネガティブにとらえず,入力,開示をためらう発想を捨てる。開示された側も問題を起こしたことに対し，責める側の立場に回ることで，自分に責任がないことを強調するような未成熟な対応に終始しない。

その前提で,保守対応に関連するチームや組織全体がより組織能力向上に向けた保守情報の活用を考えることで,関係部門間や顧客との連携が強まり win-win の関係を築き上げることができる可能性をソフトウェア保守情報は秘めていることが，本論文から分かった。

参考文献

1) JIS X160:2012 ソフトウェアライフサイクルプロセス (ISO/IEC/IEEE/EIA 12207;2008 の一致規格)
2) ユーザ企業ソフトウェアメトリックス調査 2014 第 7 章　保守調査 日本情報システムユーザ協会
3) ソフトウェア開発データ白書 2016-2017 9 章「信頼性の分析」　IPA
4) ソフトウェア開発プロジェクトの生産性評価に関する事例　佐藤浩明 Unisys Technology Review 第 129 号 2017 年 9 月
5) http://www.toshiba.co.jp/cl/sol/gene/
6) http://www.oracle.com/jp/corporate/pricing/e-pl101005-101005a-n-176288-ja.pdf
7) ソフトウェア開発データ白書 2016-2017 3 章「分析について」　IPA

保守と保守開発における Software Evolution の事例報告

三輪　東
SCSK 株式会社
azuma.miwa@scsk.jp

はじめに　15 年間進化を続けている同一 Software に関する事例報告である. Software も機能追加を繰り返し, 大幅に進化した. 加えて, その Software に関わる人々の活動や, それらを支える多様な構成要素も進化を遂げた. 本報告では, Software そのものの変化の歴史では無く, それをとりまく周辺活動の変化に焦点をあて, 進化の推移を報告したものである.

Software Evolution の考え方　私の考える Software Evolution とは「事業を継続するための Software にかかわる活動全般」を言う. Software と直接関わる活動全てを含み, かつ, Software への間接的な活動も含めるのが, 私の考えだ. 間接的なものとは, 教育・組織・営業・事業戦略など周辺の活動全てを言う. Software 周辺には Software そのものに影響を与える様々な活動があり, 複雑に関連している. それらに境界を設け無視することは出来ない. 相互依存[1]の関係である.

タイトルについて　本事例では, 保守と開発は相互依存で一体の関係と捉え, Software Evolution として表現している. しかし, 一般的には理解されにくいと考え, 分かり易さのために「保守と保守開発における」を追加した.

進化の背景　我々の活動は全て相互依存で実現されている. 自身の変化がさらなる変化の連鎖を生む複雑な世界. 加えて, 今は変化の速い VUCA 時代. 短時間で前提が大きく変わるなど, 素早い変化・適応が求められる. 今回の Software Evolution の事例は, 環境が大きく変化する過程で, 我々が生き残るために真剣に取り組んだ結果として, こういうかたちになったという一例である. 気づけば, 全員のセンサーでしなやかに全体最適に向けて進化する, アメーバのような活動体になった. 現在も好循環の成長サイクルが持続できている.

報告上の工夫　Software Evolution には段階があり, それぞれに特徴があった. それらを可視化するために, タックマンモデル[3][4]を用いて説明した.

進化の推移　キーワードは「相互依存」,「共感」,「ソフトの S・ハードの S」[2]の 3 つ. 成立期→動乱期は, 個人仕事が中心であり, チーム同士の相互依存関係が薄かった. そこを壊し, 助け合いを通じて共感を創り, 相互関係を生み出した. その共感を通じてソフトのSに働きかけつつ, ハードの S を生み出す活動を行った. 具体的には, 共感をコーディネートしながらソフトのSを育み, 成果物やプロセスの共通化を通じて, ハードのSを揃える. 大事なことは, ハードのSを現場で生みだしたことであり, 共感の無いトップダウンで与えたハードのSに従わせてはいないことだ. ここではかなりの時間と忍耐を必要とした. 急激な成長は期待できないので, 小さな成果を積み上げていくしかない. あきらめずに改善活動を継続する. 気づくと, 様々な部分で徐々に効果が出てくる. ついに改善の効果を全員が実感できるようになる. さらに改善のアクセルが踏まれる. 共感と改善の好循環が生まれる. ここが動乱期→安定期に移った状態だ. その後も, 改善と共感の好循環を止めない運営を続けた. 特に, 安定期以降は, ツール活用を推進し, さらなる進化を加速した. ふりかえりの時間を増やし, KPT を推進することで, 自然と改善が進んだ. 気づけば, 遂行期に移行していた.

最後に　常に, 成果にはタイムラグがある[1]ことを理解し, 第 5 水準のリーダーシップ[5]やコッターの 8 段階変革プロセス[6]などのリーダーシップを継続することが大事. U 理論[7]の co-evolution プロセスは非常に参考になる. 現在は,「答えのない勉強会」を加えて, 改善と共感の好循環を維持する努力を行っている.

参考文献

[1] 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, 訳者：枝 廣 淳 子・小 田 理 一 郎・中 小 路 佳 代 子, 英 治 出 版, ISBN978-4-86276-101-9

[2] マッキンゼーの 7S
『フリー百科事典 ウィキペディア日本語版』(http://ja.wikipedia.org/) 2017 年 3 月 11 日 18 時 (日本時間) 現在での最新版を取得
https://ja.wikipedia.org/wiki/%E3%83%9E%E3%83%83%E3%82%AD%E3%83%B3%E3%82%BC%E3%83%BC%E3%81%AE7S

[3] PMP 教科書 Project Management Professional 第 5 版, キム・ヘマルドン, 訳者:株式会社トップスタジオ, 翔泳社, ISBN978-4-7981-3729-2

[4] weblio 辞書 (http://www.weblio.jp/) で「タックマンモデル」の検索結果 2017 年 3 月 11 日 23 時 (日本時間) 現在

[5] ビジョナリー・カンパニー② 飛躍の法則, ジム・コリンズ, 訳者:山岡洋一, 日経 BP 社, ISBN978-4-8222-4263-2

[6] 企業変革力, ション・P・コッター, 訳者:梅津祐良, 日経 BP 社, ISBN978-4-8222-4274-9

[7] U 理論 過去や偏見にとらわれず、本当に必要な「変化」を生み出す技術, C・オットー・シャーマー, 訳者:中土井僚・由佐美加子, 英治出版, ISBN978-4-86276-043-2

[8] あなたの話しはなぜ「通じない」のか, 山田ズーニー, ちくま文庫, ISBN978-4-480-42280-4

[9] チームが成長し続ける「ラーニングサイクル」 IT 現場を強くする 究極のチームビルディング, 斉藤 秀樹, 日経 BP 社, ISBN978-4-8222-7186-2

[8],[9]はプレゼン資料にて参照

OSS 開発者の離脱要因理解のための Politeness の質的調査

宮崎 智己
和歌山大学 システム工学部
s181052@sys.wakayama-u.ac.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

近年，オープンソースソフトウェア *(OSS)* は企業のソフトウェア開発でも積極的に再利用されている．*OSS* 開発者の多くはボランティアとして *OSS* プロジェクトに参加するが短期間でプロジェクトを離脱することが知られている．そのため，多くの *OSS* プロジェクトでは，プロジェクトに長期間貢献する開発者が慢性的に不足しており，プロジェクトの安定運営およびプロダクトの品質維持に関する重大な懸念を抱えている．*OSS* プロジェクトに参加する開発者の *Politeness*（コミュニケーション上の配慮）に着目して我々が実施した先行研究 *[18, 19]* では，プロジェクト離脱直前に *Politeness* が急激に変動するコア開発者の存在を明らかにした．

Politeness の急激な変動は開発者の離脱の予兆を検知する手法の構築につながる可能性がある．離脱の予兆検知手法を構築できれば，長期間プロジェクトに貢献する重要な開発者の離脱をプロジェクト管理者等が早期に察知し，離脱を防ぐための方策の立案に役立つことが期待される．しかしながら *[18, 19]* では，開発者の *Politeness* が急激に変動した理由や，開発者の離脱と *Politeness* の急激な変動とにどのような関係があるのかについては詳細な分析をおこなっていない．

本稿では，*Politeness* が急激に変動する理由と離脱原因との関係を明らかにするために，コア開発者が離脱する直前 *3*ヶ月間の開発者メーリングリストへの投稿の内容を目視する定性的な調査を実施する．本調査の結果，離脱直前に *Politeness* が大きく低下するコア開発者はコミュニティ内で他の開発者と意見が対立していたことを確認した．一方，離脱直前に *Politeness* が大きく上昇したコア開発者の離脱理由については明示的な手がかりを見つけることができなかった．

1. はじめに

近年，オープンソースソフトウェア（以降，OSS と呼ぶ）は様々な用途で幅広く普及しており，個人ユーザによるエンドユースのみならず，企業におけるソフトウェア開発にも積極的に再利用されている．しかしながら，再利用する OSS がいつまで開発保守されるのか分からなかったり，不具合が発見された場合に確実に修正されるかどうかが不透明なため [7]，OSS の利用を躊躇するソフトウェア開発企業も少なくない．

OSS の多くはボランティア開発者によって開発保守されている．ボランティア開発者が OSS プロジェクトで中心的な役割を担うようになる（コア開発者になる）には，長期的な貢献を通じてコア開発者としての資質の有無を認められる必要がある [8, 15] が，ボランティア開発者の多くは約 1.5 年以内でプロジェクトから離脱する [2] ため，多くの OSS プロジェクトにおいてコア開発者が慢性的に不足している．特に大規模 OSS プロジェクトでは，ユーザから不具合報告や機能改善要求が多数寄せられるため，少数のコア開発者へ過度な負担がかかることが多い [9]．コア開発者の離脱はプロジェクトの安定運営に支障を来すだけではなくプロジェクトの消滅や品質の低下に繋がる恐れがあるため，ボランティア開発者の参入障壁を取り除く [13, 14]，コア開発者の負担を軽減する [1, 12]，コア開発者候補者を早期に発掘する [5, 16] など，様々なアプローチから OSS プロジェクトの安定運営のための支援方法が研究されている．

我々の研究グループでは，コア開発者の離脱の予兆を検知するための手法の構築に取り組んでいる．手法構築の予備調査として，OSS プロジェクトで開発者が情報交換するために利用する開発者向けメーリングリストを対象に，開発者の Politeness（コミュニケーション上の配

慮）[3, 17] を定量化するツール [4] を用いたケーススタディをおこなった [18, 19]．開発者の Politeness に着目した理由は，OSS 開発は他の開発者との協調作業を基本とするため，(1) 継続的に貢献するコア開発者は協調作業を円滑にするために他の開発者に対して配慮あるコミュニケーションをしているはずであるが，(2) コア開発者が離脱する際には，本業が急に忙しくなる，他の開発者と対立するなどによりコミュニケーションの質が変化する（Politeness が変化する）可能性がある，と考えたためである．

ケーススタディの結果，OSS プロジェクトに長期間参加しているコア開発者の Politeness は，短期間でプロジェクトを離脱した開発者より統計的に有意に低い値を示すことが明らかになった．さらに，OSS プロジェクトに長期間参加しているコア開発者であっても Politeness が急激に変動した場合，その直後にプロジェクトを離脱するコア開発者が存在することを明らかにした．しかしながら [18, 19] では，実際のコミュニケーションの内容については十分に分析をおこなっておらず，なぜ開発者の Politeness が急激に変動したのかや，なぜ Politeness の急激な変動がプロジェクトの離脱を招いたのかなどの理由については不明なままである．

本稿では，Politeness が急激に変動する理由と離脱原因との関係を明らかにすることを目的として，Politeness の急激な変動を示したコア開発者のコミュニケーションの内容を目視により確認する．[18, 19] の知見に加え，本調査により，開発者の Politeness の変動とプロジェクト離脱との関係が明らかになれば，開発者の離脱の予兆を検知する手法を構築する上での重要な手がかりとなることが期待できる．

以降ではまず，2 章において本研究で用いる Politeness の概念について説明する．また，Politeness を定量化するためのツール Stanford Politeness について説明する．3 章では，開発者の Politeness と活動継続性との関係を調査した我々の先行研究 [18, 19] の概要と結果を示す．4 章では，開発者の Politeness の急激な変化とプロジェクト離脱との関係を明らかにするために行った調査結果について議論する．5 章で関連研究を示し，最後に 6 章において本研究の今後の研究方針について議論し本論文をまとめる．

2 Politeness と定量化ツール

本章ではまず，Politeness の概念について説明する．次に，[18, 19] で用いた Politeness を定量化するためのツール Stanford Politeness について説明する．

2.1 Politeness

Politeness[3, 17] とは，日本語の「丁寧さ」や「礼儀正しさ」とはニュアンスが異なる概念であり，円滑な人間関係を確立・維持するための相手への配慮を指す．人は，ポジティブフェイス（他者に認められたい・評価されたいという欲求）とネガティブフェイス（他者から批判されたくない，行動を妨げられたくないという欲求）の二つのフェイスを持っており，コミュニケーションをおこなう相手のポジティブフェイスとネガティブフェイスの両方を保つために Politeness を適切に表現する必要がある．

Politeness Strategy[3] とは，Politeness を示すための具体的な表現方法である．我々は，他者との会話の中で様々な種類の Politeness Strategy を使い分けながらポジティブ・ネガティブフェイスを考慮している．

表 1 に，Politeness Strategy を示す．Politeness Strategy の例をいくつか説明する．相手のポジティブフェイスを保つための Politeness Strategy の 1 つに Gratitude がある．Gratitude は，“I appreciate you.” や “Thank you for ～.” といったように，相手に対する感謝を示すための表現である．Gratitude を受け取った相手は，話し手から評価されていることを明示的に認識することができるためポジティブフェイスが保たれる．一方，Hedges は，相手のネガティブフェイスを保つための Politeness Strategy の 1 つである．Hedges は，“It appears that ～?” や “I suggest ～.” といったように，表現を曖昧にすることで相手のネガティブフェイスを侵害しないようするための表現である．

2.2 Stanford Politeness

本研究では，開発者の Politeness を分析するために，開発者メーリングリストから取得するコミュニケーションデータに対して Stanford Politeness[4][1]を適用する．Stanford Politeness とは，言語表現に込められた Politeness を定量化するツールである．

[1] https://github.com/sudhof/politeness

表 1. Politeness Strategy（影響の強さの順に上から並べたもの）

Politeness 値を上げる Strategy		Politeness 値を下げる Strategy	
Strategy	該当単語や文の例	Strategy	該当単語や文の例
Gratitude	I appreciate thank	Direct start	So 〜. But 〜.
Deference	great, nice, awesome	Factuality	really, actually
Indirect (btw)	By the way	Please start	Please 〜.
Please	〜 please	2nd person start	You 〜. Your 〜.
Counterfactual modal	Could you 〜? Would you 〜?	Direct question	What 〜? Why 〜?
Greeting	hi, hello, hey		
Apologizing	sorry, woops		
Hedges	think, appear, feel		
1st person start	I 〜. My 〜.		
Indicative modal	Can you 〜? Will you 〜?		
1st person pl.	we, our		
1st person	I, my, mine		
2nd person	you, your, yourself		

表 2. Stanford Politeness[4] を適用した例

No.	例文	Politeness 値
1	OK, thanks for your suggestion. I will change that. Regards, *first name*.	0.997
2	Oh dear. My question is already slipping down the list into obscurity... I guess I must be the only person using eclipse 3.4 who would like to be able to do this.	0.507
3	So, what was the solution for this problem then??	0.122

表 2 に，OSS 開発者のメーリングリスト上の議論に対して Stanford Politeness を適用した例を示す．表 2 中の Politeness 値とは，Stanford Politeness を用いて Politeness を数値化した値である．Politeness 値は 0 から 1 の間の値をとり，値が大きいほど相手を配慮する表現が取り入れられていることを示す．例えば，1 番目の文章の Politeness 値は 0.997 と高い値を示している．これは，文中に含まれる “thanks” が Politeness Strategy の Gratitude に，“suggestion” が Politeness Strategy の Hedges に該当しているためである．2 番目の文章は，単語数が多く情報量の多い文章ではあるが Politeness Strategy を含んでいないため，0.507 と中間の値が算出されている．3 番目の文章の Politeness 値は 0.122 と低い値となっている．これは，文中に含まれる “So, what...” の部分が Politeness 値を下げる Politeness Strategy である Direct start に該当しており，相手を不愉快にさせる可能性の高い表現であるためである．

Stanford Politeness は，大きく分けて 2 つの処理からなる．図 1 に処理方法の概要を示す．

まず，入力テキストの構文解析をおこない，テキストに含まれる単語と Politeness Strategy を検出し BOW (Bag of Words) ベクトルを作成する．なお，Stanford Politeness には構文解析機能がないため，本研究では構

図 1. Stanford Politeness における Politeness 値の算出方法

文解析ツールである Stanford Parser[2] を用いて入力テキストの構文解析をおこなう．

次に，作成した BOW ベクトルを機械学習アルゴリズムの 1 つであるサポートベクタマシン (SVM)[6] により学習させる．SVM の学習データには，Stanford Politeness Corpus[4] が用いられている．Stanford Politeness Corpus は，Wikipedia ユーザのトークページ[3]から抽出した 4,353 の質問と Stack Exchange[4]から抽出した 6,604 の質問に対して，人手により Politeness を評価した値からなるデータの集合である．最後に，入力テキストを SVM で判別し Politeness 値を算出する．

3. 先行研究の概要

本章では，[18, 19] における分析内容と結果について概説する．

3.1 対象プロジェクト

[18, 19] では，大規模 OSS である Apache HTTP Server プロジェクトにおける開発者メーリングリストから抽出したメールデータに対して Stanford Politeness を適用し，開発者の Politeness とプロジェクト離脱（あるいは継続性）との関係を理解するためのケーススタディを実施した．分析対象は，プロジェクトが設立された 1995 年 9 月から 2016 年 3 月時点の約 21 年間分のメールデータである．

Apache HTTP Server プロジェクトを選択した主な理由は以下の通りである．

- 20 年以上存続しているプロジェクトでありコア開発者の中にも離脱者が多数存在すること
- 企業とは異なり離脱の要因と考えられる業績や役職等による社会的ストレスの影響を受けにくいこと
- 同じプロジェクトを対象とした先行研究が数多くありケーススタディにより得られる知見の妥当性を確保しやすいこと
- 開発者メーリングリストを通じてやり取りされたメールデータがすべてアーカイブされており，かつ，容易に取得可能であること

3.2 分析内容

ケーススタディでは，以下の 2 点を主に分析した．

(1) プロジェクトに長期間参加した開発者と短期間参加した開発者の Politeness の違い

(2) プロジェクトに長期間参加したコア開発者の Politeness の経時的変化とプロジェクト離脱との関係

(1) の分析を実施するにあたり，まずプロジェクトに参加した開発者を定義する必要があった．開発者メーリングリストでは，投稿を 1 度だけおこなってそれ以降は 1 度も投稿しないような開発者も多数存在するため，そのような開発者を「プロジェクトに参加した」と見なすのは適切ではない．また，開発者の投稿数が 1 度のみといった様な少ない場合に関しても，プロジェクトに貢献していると見なすことは適切ではない．したがって，90 日以内に 1 度以上投稿した，かつ総投稿数 10 件以上の開発者を「プロジェクトに参加した」と見なし,「プロジェクトに参加した」開発者の投稿（他者の投稿に対する返信も含む）を分析の対象とした．次に，プロジェクトに長期間・短期間参加した開発者を区別するために，前述の条件を満たし，かつ，270 日以上開発者メーリングリストに投稿をおこなった開発者を長期開発者，それ以外を短期開発者と定義した．その結果，長期開発者 183 名，短期開発者 173 名を抽出した．

(2) の分析では，長期開発者 183 名の中から投稿数の多い順に上位 10 名をコア開発者と定義し，コア開発者の Politeness の経時的変化とプロジェクト離脱との関係を分析した．

[2] http://nlp.stanford.edu/software/lex-parser.shtml
[3] http://en.wikipedia.org/wiki/Wikipedia:User_pages
[4] http://stackexchange.com/about

3.3 分析結果

3.3.1 Politeness 値：長期開発者 vs. 短期開発者

結果の詳細については紙面の関係上割愛するが，検定の結果，長期開発者の Politeness は短期開発者よりも有意に低い値を示すことが分かった．プロジェクトに長期的に参加する開発者は，他の開発者との良好な関係が構築できているため，Politeness を過度に意識する必要がない，あるいは，他人行儀な改まった態度を示す必要がないということが示唆される．

Ortu らの研究 [11, 10] では，不具合修正プロセスにおける開発者の Politeness と不具合修正時間との関係を分析しているが，高い Politeness 値を示す議論を通じて解決される不具合は比較的長い時間を要することが明らかになっており，本ケーススタディの結果とも共通する現象を捉えていると思われる．

3.3.2 コア開発者の Politeness 値の経時的変化

図 2 に，コア開発者が投稿したメール（他の開発者の投稿に対する返信を含む）の Politeness 値を時系列にプロットしたグラフを示す．青の折線が Politeness 値，赤の折線が投稿件数を示している．なお，単純に月毎にデータを集計してプロットすると投稿件数の少ない期間の外れ値の影響を受けてグラフが読み取りづらくなるため，Politeness 値，メール件数ともに，3 か月間の移動平均をプロットしたグラフであることに注意されたい．

図 2 から見て取れるように，Dev#1 を除いて，コア開発者であっても分析対象である約 21 年間のすべての期間において活動（メール投稿）している開発者は存在せず，どこかの時点でプロジェクトを離脱している．また，コア開発者間で Politeness 値に個人差があること，時期により Politeness 値が変動することが確認された．特に，プロジェクトを離脱する際に Politeness 値が急激に変動する開発者が確認された．表 3 に，プロジェクトから離脱したコア開発者のプロジェクト離脱 6 ヶ月前から離脱 3ヶ月前までの期間とプロジェクト離脱直前 3ヶ月間の期間のメール投稿数と Politeness 値の平均値を示す．表 3 から，Dev#2，Dev#3，Dev#4，Dev#7，Dev#8，Dev#10 の 6 名のコア開発者に関して，プロジェクト離脱直前に 0.11〜0.20 の Politeness 値の変動（上昇，もしくは低下）が確認された．

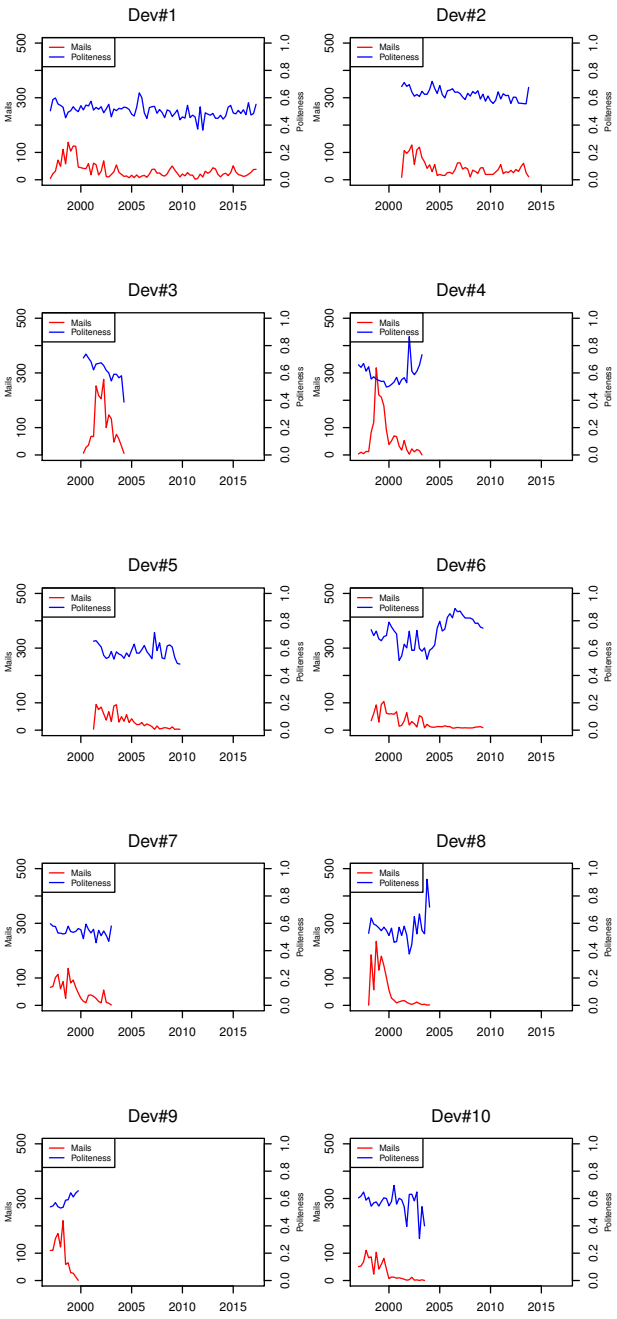


図 2. コア開発者が投稿したメールの Politeness 値の時系列変化（3ヶ月移動平均）

表 3. 離脱したコア開発者のメール投稿数と Politeness 値（月平均）

	(A) 離脱 6ヶ月前から離脱 3ヶ月前まで		(B) 離脱直前 3ヶ月間		Politeness 値の変動 (B-A)
	メール投稿数	Politeness 値	メール投稿数	Politeness 値	
Dev#2	25.7	0.56	11.0	0.68	0.12
Dev#3	32.3	0.58	6.0	0.39	-0.19
Dev#4	15.7	0.66	0.3	0.73	0.07
Dev#5	4.0	0.49	3.0	0.48	-0.01
Dev#6	13.0	0.75	9.0	0.75	0.00
Dev#7	7.7	0.47	1.3	0.58	0.11
Dev#8	1.0	0.92	1.7	0.72	-0.20
Dev#9	13.0	0.64	0.7	0.66	0.02
Dev#10	2.7	0.54	0.3	0.40	-0.14

Politeness の急激な変動が開発者がプロジェクトを離脱する予兆なのであれば，開発者の離脱の予兆を検知する手法を構築する上での重要な手がかりとすることができる．離脱予兆の検知手法を確立できれば，プロジェクト管理者等が開発者の離脱を未然に阻止するための対策を講じることが可能になる．

3.4 先行研究の問題点

図 2 に示したように，数年間プロジェクトに参加しているコア開発者であってもプロジェクトを離脱することが分かった．また，プロジェクトを離脱する直前には Politeness 値が大きく変動する，すなわち，コア開発者のコミュニケーション内容の質が変化していることが分かった．

しかしながら，[18, 19] ではコア開発者の投稿内容までは詳細に分析できておらず，なぜ開発者の Politeness 値が急激に変動したのかについては不明なままである．そのため，Politeness の急激な変動（コミュニケーションの質的変化）がプロジェクトの離脱の予兆と見なすことができるか，すなわち，Politeness の急激な変動と開発者の離脱との間に明確な関係が存在するのかが不透明である．

そのため本稿では，プロジェクト離脱の直前に Politeness の急激な変動を示したコア開発者の中から，Politeness 値が低下した開発者と Politeness 値が上昇した開発者をそれぞれ 1 名選定し，プロジェクト離脱の直前の開発者の投稿内容を目視により確認・分析する．

4 目視による調査

コア開発者がプロジェクトを離脱する直前に Politeness が大きく変動している理由を明らかにするために，開発者メーリングリストに投稿された離脱直前 3ヶ月間分のメールを目視により調査する．

4.1 調査対象

対象開発者の離脱直前 3ヶ月間の以下の項目を調査対象とし，Politeness の急激な変動とプロジェクト離脱理由を明らかにする．

- 開発者が投稿した内容（メール本文）
- 投稿の Politeness 値
- 本文に含まれている Politeness Strategy

次に，3.3.2 頁で述べた Politeness 値が急激に変動した開発者の中から調査対象者を選出する．表 3 より，Dev#4 や Dev#10 は離脱直前 3ヶ月間に投稿したメール数が著しく少ないことがわかる．同じ開発者が投稿したメールであっても，Politeness 値はメールによって大きく異なるため，Dev#4 や Dev#10 は外れ値によって Politeness 値の平均値が変動した可能性がある．そのため調査対象者は，離脱したコア開発者のうち，投稿数の多い Dev#2 および Dev#3 とする．Dev#2 は，離脱直前に Politeness が上昇しているパターンの開発者，Dev#3 は，離脱直前に Politeness が低下しているパターンの開発者である．

次節ではこれら 2 名の開発者を対象とした調査結果について詳しく述べる．

4.2 調査結果

本節では，投稿内容を目視した結果について述べる．まず，離脱直前に Politeness が低下しているパターンの開発者である Dev#3 の調査結果を示す．Politeness が急激に低下するということは，他の開発者のネガティブフェイスを侵害する表現が増えていることを意味しており，Dev#3 とその周辺の開発者の間に何らかの対立や緊張状態が生まれている可能性がある．

次に，離脱直前に Politeness が上昇しているパターンの開発者である Dev#2 の調査結果を示す．Politeness の上昇は，コミュニティに新たなメンバを迎える際，新人に友好的であることを古参メンバが示すための態度として我々の社会生活でも経験する事象であるが，Dev#2 の場合は直後に離脱している．これまで友好関係を築いていた他の古参メンバとの対立が生じたり，何らかの理由でコミュニティに一定の距離を取り始めた結果,「よそよそしく」なっている可能性がある．

4.2.1 Dev#3

表 4 に，Dev#3 の投稿内容の例，本文に含まれる Politeness Strategy，および，本文の Politeness 値を示す．

プロジェクト離脱直近 3ヶ月分の Dev#3 の投稿をすべて目視した結果，この時期の Dev#3 は他の開発者と意見が折り合えず，口論にまで発展していることが分かった．口論の原因は，他の開発者が変更したコードに不具合が含まれていることを Dev#3 が指摘しているが，他の開発者からは理解が得られないといった状況によるものである．

例えば，表 4 の 1 つ目の投稿は，コードの変更によりメモリリークを生み出す可能性をその理由とともに指摘している．本文中に含まれる “actually” や “In fact” は，Politeness 値を大きく下げる Politeness Strategy である Factuality（表 1 参照）に該当する．Politeness 値を上げる Politeness Strategy である 1st person や 1st person start など（I, my, we など）も数多く含むが，Factuality の効果を相殺するほどではない．Politeness Strategy 以外にも，“VERY big mistakes” や “too dangerous” といった相手のネガティブフェイスを損害する直接的な表現が目立つ．Stanford Politeness Corpus には，Politeness Strategy 以外にも単語単位で人手による評価が加えられているため，ネガティブな文脈で使われる単語等にも Politeness 値を大きく下げる効果があると思われる．

2 つ目の投稿は，Politeness 値を大きく下げる Politeness Strategy が含まれておらず，1 つ目の投稿と比較すると高い Politeness 値を示している．しかし，1 つ目の投稿と同様に，メモリリークを引き起こす変更に対して強く反対していることが読み取れる．また，1 つ目の投稿と同様に，相手のネガティブフェイスを明示的に侵害する表現がいくつか見られる．

3 つ目の投稿では，1st person よりも 2nd person の Politeness Strategy が多用されている．また，Politeness 値を大きく下げる Politeness Strategy の Factuality (“in fact”) も使用している．本文からは，メモリリークを引き起こす問題について強く相手を批判している様子が読み取れる．

これらのことから，この時期に Dev#3 には納得できない変更があり強く反対意見を述べた結果，Politeness 値を大きく下げることに繋がっていることが分かった．したがって，この変更に関するプロジェクトの方針に賛同できなかったため，Dev#3 は直後にプロジェクトを離脱するという選択を取ったと思われる．

4.2.2 Dev#2

表 5 に，Dev#2 の投稿内容の例，本文に含まれる Politeness Strategy，および，本文の Politeness 値を示す．

プロジェクト離脱直近 3ヶ月分の Dev#2 の投稿をすべて目視した結果，Dev#2 がおこなった変更に関してフィードバックした他の開発者に対する返信や，他の開発者の投稿したパッチに対して賞賛する内容の投稿が確認できた．

例えば，表 5 の 1 つ目の投稿は，パッチの投稿に感謝の意を伝えるためのものである．本文中に含まれる Politeness Strategy は Deference の “great” と Gratitude の “thanks” のみであるが，それ以外の情報が少ないため高い Politeness 値を示している．

2 つ目の投稿は，他の開発者が作成したパッチに対して感謝の意を伝えるとともに，さらなるパッチの検証を他の開発者に依頼するものである．Hedges や Gratitude など Politeness 値を大きく上げる Politeness Strategy が複数含まれている．一方，Politeness 値を大きく下げる Factuality（“really”）も含まれているが，前向きな文脈の中で使用されている（“I’d really prefer we ∼.”）

表 4. Dev#3 の投稿内容の例（本文中の太字は Politeness Strategy に該当）

id	本文	Politeness Strategy	スコア
1	**I** am **actually** pretty **sure** that allocating brigades out of the bucket_allocator is a VERY big mistake. **In fact**, **I** am close to asking that change to be backed out because it is too dangerous. When **we** first designed buckets and bucket brigades, **we** made one VERY clear distinction. Bucket_brigades are allocated out of a pool, because that stops us from leaking memory. Buckets shouldn't be allocated out of a pool, because nobody knows which pool to allocate them from. Basically, **we** said that in Apache, it is safe to just drop a bucket brigade because the pool cleanups will free the memory. It is not safe to just drop a bucket, it must either be left on the brigade to be cleaned up by the brigade_cleanup function, or it must be destroyed when it is removed from the brigade. By allocating bucket_brigades out of the bucket_allocator, **we** have removed the protection that the pool cleanups used to give us. **I** may be wrong about how the bucket_allocator works, but **I** don't believe **I** am. Basically, this change will make Apache leak memory on some requests, and it goes against the original design of buckets and bucket brigades. **In fact**, allocating the brigades out of a pool was a requirement, because some people at the meeting were afraid of memory leaks with bucket brigades.	Factuality, 1st person pl., 1st person, 1st person start	0.08
2	**I** am vetoing this change until **I** get a clear description of the problem it is solving. **I** have explained multiple times why it is a bad change, but **I** can't offer any other solutions until **I** know why it is needed. It will not. If **you** _ever_ create a brigade with a NULL pool, **you** will have memory leaks. **I** dare say that every filter ever written will leak if the feature implemented with this patch is ever used. **I** have already posted one code segment in the core_output_filter that proves the memory leak exists. Trust me when **I** tell **you** that there are hundreds more, and some of them are in 3rd party modules that **you** don't control. ++1, but they should go into an external library so that projects like serf have access to them. Happily, **I** don't have to worry about that.	1st person, 1st person start, 2nd person	0.38
3	No, that is a solution, **in fact**, it was part of the original design. What **you**'ve done is said "The brigade may not live long enough", but **you** haven't explained how that is the case. **You** also haven't explained why moving the buckets to another brigade won't work. Which **I** have a very hard time believing, because that is how Apache works today. **You** also haven't answered the problem that even _WITH_ this change, **you** still need to move the buckets to a new brigade. At the end of the day, this change doesn't get **you** anything, because **you** _still_ have to do the brigade migration that **you** say **you** can't do.	Factuality, 1st person, 2nd person, 2nd person start	0.05

ため大きな影響はなかったと考えられる．結果として，Politeness 値が 0.99 と非常に高い値を示しているものと思われる．

3 つ目の投稿では，メールクライアント Thunderbird とプロバイダの相性から問題が生じ，メールの復旧作業に入るために活動をできないことをコミュニティに伝えている．また，他の開発者へのアドバイスも含まれており，Gratitude や Apologizing，Hedges といった Politeness を上げる Politeness Strategy を多く含んでいる．

このように，表 5 以外のものを含めても，Dev#3 とは異なり Dev#2 の投稿内容からはプロジェクト離脱の予兆を示す言動を読み取ることはできなかった．しかしながら，Politeness の上昇後に離脱するという現象は不可解である．何らかの理由によりすでに離脱を決めている，あるいは，離脱しようとしている過程での人間の意識の現れとも考えられるが推測の域を出ない．

コア開発者の離脱直前の 3ヶ月分のメールデータは 1 名で数十件を目視する必要があった．また，メールによって非常に長い本文を含む場合もあったため，本稿ではまず，Politeness が上昇したパターンを持つ Dev#2 のみを分析対象とした．同様のパターンを持つ開発者は Dev#2 以外にも，Dev#4 や Dev#6，Dev#8 が該当するため，今後はこれらの開発者の分析を進め離脱理由を明らかにしたい．ただし，いずれの開発者の離脱理由は投稿内容からは読み取れない可能性もある．その場合は，他のプロジェクトを対象に追加分析をしたり，可能であれば離脱した開発者へのインタビューを実施し，Politeness の大きな上昇とプロジェクト離脱との関係を明らかにしたい．

5 関連研究

Politeness 分析に関しては Ortu らの研究 [11, 10] が関連研究として挙げられる．OSS 開発で利用されている不具合管理システムのコメントデータに対し Politeness 分析を適用し，開発者の Politeness と不具合修正時間の関係を分析したものである．分析の結果，Politeness の高いコメントが寄せられた不具合は，Politeness の低いコメントが寄せられた不具合よりも修正時間が長く必要となることを明らかにしている．3 章において述べたよう

表 5. Dev#2 の投稿内容の例（本文中の太字は Politeness Strategy に該当）

id	本文	Politeness Strategy	スコア
1	Looks **great**! **Thanks** for the patch :)	Gratitude, Deference	0.90
2	**Thanks** for the feedback, Kaspar. Other than shifting the one #ifdef case to the correct line of code, **I believe** the patch is complete, **I**'ve tried to state **my** case for no further changes. If **you would like** to **propose** further changes, **I**'d **really prefer we** defer these to 2.2.24-dev so we can T&R already. **I**'m pretty **sure you** already agree with me that the flexibility to disable a particular cipher in light of exploit research in the very fresh openssl code base makes this patch pretty critical to release for legacy, as well as stable. Even if **you** can give the patch **your** +1, we still need one more individual to chime in before **we** can finish this backport, and as **I** mention, OpenSSL's CVE-2012-2333 suggests that this patch is a showstopper. Can one more person take a pass through the code, since Stefan didn't have a chance to re-review this week?	Gratitude, Hedges, Factuality, 1st person pl., 1st person, 2nd person	0.99
3	**Sorry**, **I**'m not ignoring the list (entirely). Seems Thunderbird and **my** ISP have decided not to dance anymore and it looks like **I**'m spending Thursday doing some fundamental email restructuring (sigh). Hopefully **I**'ll have the list traffic back sometime by Friday, **thanks** to Jeff, Reindl, and Steffen for **your** review, **I** didn't read Ruediger, Rainer, or Jim as voting one way or another - even with **my** vote there is still a missing PMC +1 for release. Rainer, are **your** observations a regression? Because the security issues are relatively minor, **I** took this opportunity to update to a slightly refreshed autoconf (not a radically new version, nor the last in that version minor which barfs on our configure. in and m4 files)... so it's possible **I** had introduced this. If not a regression, please confirm.	Gratitude, Hedge, Apologizing , 1st person pl., 1st person, 2nd person, Direct start	0.87

に，Politeness の高いコミュニケーションは一見望ましいように思われるが，問題を効率的に解決するという観点では必ずしも必要ではなく，チームワークの高いチームでしばしば見られるフランクな会話の方が良い場合があることを示唆するものである．我々の先行研究においても [18, 19]，プロジェクトに長期間参加する開発者の Politeness はその他の一般開発者に比べて統計的に有意に低いことを示しており，Ortu らの研究とは目的が異なるものの得られた知見は矛盾するものではない．

OSS プロジェクトへ参加する開発者の離脱阻止の支援という観点では，Steinmacher らの研究 [13, 14] が挙げられる．Steinmacher らの研究は，プロジェクトへこれから参加したいと考えている開発者の参入を妨げる要因について分析したものである．ドキュメントの不足や不親切な対応など，新人にとって障壁となる計 58 種類の要因を明らかにしている．本研究は古参メンバの離脱要因を理解し離脱の予兆を検知する手法を構築することを目的としており，アプローチが大きく異なるものの OSS プロジェクトの安定運営の支援という観点では互いに補完しあうものである．

6. まとめと今後の課題

本研究では，開発者のプロジェクト離脱と Politeness の変動の関係を明らかにするために，Politeness の急激な変動を示したコア開発者のメールを目視した．目視の結果，Politeness が大きく低下した開発者はコミュニティ内で意見衝突をしていることが確認された．開発者間の意見衝突は開発者のプロジェクト離脱を促す要因の 1 つとして考えられるため，Politeness の大きな低下は開発者のプロジェクト離脱の予兆とみなすことができる．プロジェクト管理者は Politeness が大きく低下した開発者に対してなんらかの処置をすることで，その開発者の離脱を未然に防止することができると考えられる．一方で，離脱直前に Politeness が大きく上昇した開発者は他の開発者に感謝をしていることが確認されたが，明確な離脱理由を見つけることはできなかった．そのため，プロジェクトの離脱と Politeness の上昇に普遍的関係が存在するかを明らかにするために，さらなる分析をおこなう必要がある．

今後の課題としては，本研究で分析した対象開発者以外の開発者に対しても分析をおこない，開発者のプロジェクト離脱と Politeness の変動の関係についての知見を得たいと考えている．本研究では開発者の離脱直前の 3ヶ月間を分析対象としたが，開発者の離脱の予兆を察知する期間として適切であるかは確かではない．今後より詳細な分析をおこない，開発者の離脱の予兆を察知するための適切な期間を明らかにしていきたい．

7. 謝辞

本研究の一部は，文部科学省科学研究補助金（基盤(C): 15K00101）による助成を受けた．

参考文献

[1] John Anvik, Lyndon Hiew, and Gail. C. Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering (ICSE '06)*, pp. 361–370, 2006.

[2] Christian Bird, Alex Gourley, Prem Devanbu, Anand Swaminathan, and Greta Hsu. Open borders? immigration in open source projects. In *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR '07)*, 2007.

[3] Penelope Brown and Stephen C Levinson. *Politeness: Some Universais in Language Usage*. Cambridge University Press, Cambridge, UK, 1987.

[4] Cristian Danescu-Niculescu-Mizil, Moritz Sudhof, Dan Jurafsky, Jure Leskovec, and Christopher Potts. A computational approach to politeness with application to social factors. In *Proceedings of 51st Annual Meeting of the Association for Computational Linguistics (ACL '13)*, 2013.

[5] Mohammad Gharehyazie, Daryl Posnett, Bogdan Vasilescu, and Vladimir Filkov. Developer initiation and social interactions in oss: A case study of the apache software foundation. *Empirical Software Engineering*, Vol. 20, No. 5, pp. 1318–1353, 2015.

[6] Steve Gunn. Support vector machines for classification and regression. Technical Report, School of Electronics and Computer Science, University of Southampton, 1998.

[7] IPA（独立行政法人情報処理推進機構）. 第 3 回オープンソースソフトウェア活用ビジネス実態調査（2009 年度調査）, 2009.

[8] Chris Jensen and Walt Scacchi. Role migration and advancement processes in ossd projects: A comparative case study. In *Proceedings of the 29th International Conference on Software Engineering (ICSE '07)*, ICSE '07, pp. 364–374, 2007.

[9] Audris Mockus, Roy T Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 11, No. 3, pp. 309–346, 2002.

[10] Marco Ortu, Bram Adams, Giuseppe Destefanis, Parastou Tourani, Michele Marchesi, and Roberto Tonelli. Are bullies more productive?: Empirical study of affectiveness vs. issue fixing time. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR '15)*, pp. 303–313, 2015.

[11] Marco Ortu, Giuseppe Destefanis, Mohamad Kassab, Steve Counsell, Michele Marchesi, and Roberto Tonelli. Would you mind fixing this issue? an empirical analysis of politeness and attractiveness in software developed using agile boards. In *16th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP 2015)*, pp. 129–140. Springer International Publishing, Cham, Switzerland, 2015.

[12] Jin-woo Park, Mu-Woong Lee, Jinhan Kim, Seungwon Hwang, and Sunghun Kim. Costriage: A cost-aware triage algorithm for bug reporting systems. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-11)*, pp. 139–144, 2011.

[13] Igor Steinmacher, Tayana Uchoa Conte, Christoph Treude, and Marco Aurélio Gerosa. Overcoming open source project entry barriers with a portal for newcomers. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*, pp. 273–284, 2016.

[14] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '15)*, pp. 1379–1392, 2015.

[15] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation open source software developers. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*, pp. 419–429, 2003.

[16] Minghui Zhou and Audris Mockus. What make long term contributors: Willingness and opportunity in oss community. In *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*, pp. 518–528, 2012.

[17] 宇佐美まゆみ. ポライトネスという概念. 月刊 言語, Vol. 31, No. 1, pp. 100–105, 2002.

[18] 宮崎智己. OSS 開発における活動継続性と Politeness の関係. Technical report, 和歌山大学システム工学部 2016 年度卒業論文, 2017.

[19] 宮崎智己, 山谷陽亮, 東裕之輔, 大平雅雄. OSS 開発コミュニティの進化の理解を目的としたコミュニケーション分析：Politeness 分析適用の試み. 情報処理学会 マルチメディア，分散，協調とモバイル (DICOMO2016) シンポジウム論文集, pp. 703–713, 2016.

OSS 事前評価による開発リスク特定の取組み

岩崎 孝司　高山 修一　岩永 裕史
富士通九州ネットワークテクノロジーズ㈱
Iwasaki.takashi@jp.fujitsu.com
takayama_@jp.fujitsu.com
iwanaga_hiroshi@jp.fujitsu.com

鵜林 尚靖　亀井 靖高
九州大学
ubayashi@ait.kyushu-u.ac.jp
kamei@ait.kyushu-u.ac.jp

要旨

昨今のソフトウェア開発において、OSS を利用した開発が徐々に増えてきている。また、利用した OSS に起因する問題が開発後半で発生し、開発コスト増や納期遅延といった問題が発生することも珍しくない。OSS を利用する前に OSS の利用リスクを判断する事前評価の技術や開発プロセスは既存の研究でも実施されてきたが、開発現場で一般的に活用されるには至っていない。本研究では開発現場への導入を主眼に置き、開発者が OSS を利用する際に意識している品質要求から OSS の利用リスクを判断できる OSS 事前評価手法を考案した。本手法は、品質要求と OSS を事前に評価した事前評価指標を関連付け、開発者にリスク評価結果を示す事前評価レポートを作成する手法とした。考案した手法を、当社内で実際に開発を行ったプロジェクトに対して適用し、リスクの事前評価結果と実際の開発で発生した OSS 利用時の問題を比較した。結果、今回試行の対象としたプロジェクトで実際に発生した問題に対しては、本手法を用いてリスクを検知できる事を確認した。

1. 背景

(1) 企業での OSS 利用状況

近年のソフトウェア開発において、OSS(Open Source Software)を利用した開発が一般化しつつある。全国のソフトウェア関連企業の 50%を超える企業で OSS を利用した開発が行われている[1]。WEB系システム構築や企業向けの情報システムの様な一般的な情報システムの構築に加えて、通信ネットワークシステムやIoT（Internet of Things）システムの様な組込みシステムの開発においても、OSS を用いた開発が急増している。富士通九州ネットワークテクノロジーズ株式会社（以降、富士通 QNET）社内の場合でも年々OSS の利用率が上昇し、最近では約 50%のプロジェクトで OSS を利用した開発が行われている[11]。

OSS を利用したプロジェクトの問題として、利用した OSS の問題が開発工程後半で発覚し、納期遅延や開発コスト増大等のプロジェクト全体の問題になってしまう様な事態も発生しており、開発初期のリスク管理が重要視されている。

その様な OSS 利用時の問題に対応する技術として、開発開始前や開発工程前半で OSS 利用時のリスクを判断する OSS 事前評価手法[2-7]がある。OSS 事前評価手法は企業向け情報サービスやITソリューションを提供する企業では普及しつつあり、世の中では事前評価を含むコンサルタント業務をビジネスとして行う会社もある。一方、富士通 QNET を含む設計開発会社では事前評価手法は存在するものの十分に普及、運用されていない状況である。本研究では、OSS 事前評価指標を開発現場に浸透させるため、普及、運用を考慮した OSS 事前評価手法を考案した。普及、運用の課題については第二章で詳細を述べる.

(2) OSS 研究の状況

OSS を実際に利用する前に評価する事前評価技術の研究は2010年までに多く取り組まれ、その有効性が評価されてきた[2-7]。同時に事前評価を正しく実施するための開発プロセスの研究やツール開発も積極的に取り組みが行われた[8-10]。 現在ではソフトウェア工学の1つの独立した研究分野として「オープンソフトウェア工学」が提唱されソースコードの評価やプロセスに留まらず、OSS コミュニティの人的リソースの相互関係やビジネスモデルの研究が行われており、事前評価技術を含む、多面的な観点での研究となっている[13]。

(3) 論文の構成

以降、本論文の第二章では OSS の事前評価の課題を、開発現場での普及、運用の観点を含めて述べる。第三章では開発現場への普及を目的に、本手法を開発者が持つ品質要求を元に OSS 事前評価を行う手法とした事を述べる。第四章では、事前評価指標を品質要求に結び付けたやり方について、本手法での実現方法を述べる。第五章では本手法の全体像を実施手順、スコアリングルール、事前評価の結果作成されるレポートの例で紹介する。第六章では本手法を既存の開発プロジェクトを元に検証した結果、手法により、抽出したリスクがプロジェクトで発生した問題に対応しており、評価手法が有

効であったことを述べる。第七、八章にて、本研究のまとめと今後の課題を述べる。

2. OSS 事前評価手法の課題

OSS を利用する前に OSS の利用リスクを予測する OSS 事前評価手法は、OSS 利用の拡大とともに重要視されてきている。OSS 事前評価技術やプロセスの適用は、一部では普及しているが、組込み開発を中心として、設計開発を実施する会社では、必ずしも普及しておらず組織的な運用に至っていないのが現状である。

本研究で、現在の評価方法の問題を確認するため、OSS 事前評価を実施する既存の評価モデルである、QSOS[2]、OpenBRR[8]、RepOSS[4]、 OMM[10] の調査を実施した結果、「用語がおおまかで曖昧」,「スコアリングルールが曖昧な項目が半数」,「公開されたリポジトリがない」といった課題を認識した[11]。また、開発現場に事前評価手法が普及しない理由を以下の様に考察した。

・開発者は、提示された事前評価指標を1つ1つ解釈し、開発者自身が持つ品質要求と結び付けてリスク分析をしなければならない。そのため、リスク評価が開発者の能力に依存する。

・評価方法が曖昧で理解しづらい。例えば、既存研究では「ユーザビリティ」(使用性)という項目があるが、お客様の使い勝手を指すのか、OSS 自体の使い勝手を指すのか、一見して分かりづらい。

・事前評価指標が多種であり、全評価指標を調査し、検討するにはコストがかかり過ぎる。

・開発する製品の要求性能や要求品質により OSS への要求性能や要求品質が変わるため、事前評価が一意には評価できない

これらの課題から、OSS 事前評価手法開発の課題を以下とした。

(1) 開発現場が理解しやすい手法の確立

OSS 事前評価時の観点を開発者側から見た品質要求で定義して、開発者が理解しやすい評価手法を開発する事にした。例えば、「ユーザビリティ」は、開発者からみた OSS の使い勝手と、システムの利用者から見た使い勝手の品質要求に分けて定義し、開発者が理解しやすい評価手法とした。詳細は表 2 を参照、ユーザビリティの例は使用性に対応している。

また設計開発の場合は、開発者の利用する OSS に対する品質要求は開発するシステムや製品によって異なるため、品質要求のレベルも異なってくる。これらの異なる品質要求レベルに対応できる手法とする。

表1. 品質要求カテゴリと品質要求

カテゴリ	品質要求
機能適合性	利用するOSSの機能に関する品質要求
性能効率性	利用するOSSの性能に関する品質要求
使用性	利用するOSSのインストールやシステムへの組み込み等の利用容易性に関する品質要求
信頼性	利用するOSSの開発コミュニティやOSS自体の保守に関わる品質要求
セキュリティ	利用するOSSのセキュリティに関する品質要求
保守性	利用するOSSのプログラム構造や可読性に関する品質要求
移植性、互換性	利用するOSSを他システムや他OSで利用した場合を想定した移植性、互換性に関する品質要求

(2) 事前評価作業のコストの明確化と削減

OSS を利用する開発者は開発コスト削減や開発スピード向上を目的としており、事前評価作業や指標取得を実施する事に大きな負担を感じている。開発者の負担を軽減するためにも、事前評価作業におけるコストの明確化を図る共に、不要な作業を除去して、より小さなコストで評価を実施できる手法とする。

(3) 個人のスキルに依存しない事前評価作業の実現

従来研究の評価モデルでは、事前評価指標と OSS の品質要求を、開発者が自身の知識やスキルを元に対応させ評価する必要があった。本手法は開発者の知識やスキルに依存せず、一定の精度で利用する OSS のリスクを予測できる手法とする。

3. OSS 利用者の品質要求について

本手法を、開発者の品質要求から OSS の利用リスクを判断できる手法とするため、開発者の OSS 品質要求としてどの様な要求事項があるかを抽出した。

品質要求を抽出する際の観点としては、システム・ソフトウェア製品品質(JIS X 25010:2013)の品質要求のカテゴリを参考として、表 1 の7つの分類を採用し、OSS 利用の観点から各カテゴリに対応する品質要求を定義した。

具体的な品質要求の抽出に当たっては、研究チームの企業と大学側で、役割分担してまずは素案を作成した。企業側からは、OSS を利用した開発を行っている開発者(開発経験 10 年前後)複数名で実際の開発現場での品質要求をカテゴリ毎に抽出した。大学側は、先例の研究により明確になっている事前評価指標(表3参照)[11]から類推される品質要求を抽出して品質要求の素案とした。

次に、作成した品質要求の素案を元に当社の開発プロジェクトのメンバにヒアリングを実施した。ヒアリングの範囲としては、8プロジェクトの開発リーダ、開発担当の技術者を対象とした。ヒアリングにより追加された品質要求は 25 項目挙がったが、各カテゴリにより、項目数のばらつき

表2. カテゴリ毎の品質要求項目

カテゴリ	品質要求
機能適合性	OSS自身が実現したい機能を満たしている
	OSSのリリースが迅速で早期に機能を利用できる
	正常系が正しく動作しており，早期に機能を確認できる
性能	OSS自身が想定されるリソースで動作する
	OSSが指定された性能を満たす事が出来る
	性能測定のための環境作りが容易にできる
互換性・移植性	OSSが他のOSSやシステム内の機能と併存して利用可能
	複数種別のＯＳやハードに対応している。
	他の代替可能なOSSがある
使用性	OSS自体の操作性が利用者の要求を満たしている
	OSS自体にドキュメントや書籍が揃っていて，十分な情報が得られる
	OSSのインストールや組み込みが比較的容易にできる。
	OSSの利用例が多い
信頼性	サポートするコミュニティがしっかりしていて，保守されている
	OSSのバグ情報や対応状況が把握できる
	OSS自体の品質が良い
セキュリティ	脆弱性に関する報告が少ない
	信頼できるサイトで公開されている
保守性	OSSのプログラム構造が容易でホワイトボックス化しやすい
	ソースコード中のコメントが豊富で読解性が良い

があり、1 項目しか品質要求が挙がらなかったカテゴリもあれば、7 項目以上品質要求が挙がったカテゴリもあった。項目数を極力そろえるため、類似項目の統合や有識者による再検討を経て、各カテゴリの品質要求項目を 2～4 項目に見直し整理した。整理後の品質要求を表 2 に示す。

抽出した品質要求について当社において過去1年間で OSS を利用した開発リーダ（述べ 50 名）に対して、品質要求の重視度のアンケート調査を実施した。アンケートは、各品質要求に対しての重視度を、5:重視する 4:やや重視する 2:やや重視しない 1:重視しないの4段階で開発リーダに選択させた。

アンケート結果を図1に示す。アンケート時点の品質項目数は 23 件（後の整理統合により、最終的には表 2 に示す 19 件となった）であり、23 件中、ほとんどの項目は 3 点以上の評価結果となり、半分以上の項目は 4 点以上（重視する、やや重視する）という結果になった。

これらの結果により、定義した品質要求は、開発現場での品質要求を適切に捉えたものであると判断した。

一方、平均値として相対的に低い値となった品質要求

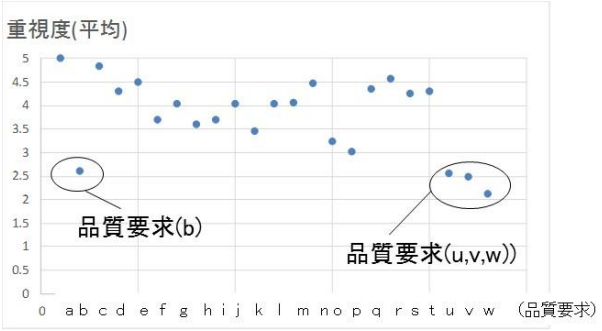


図1 品質要求のアンケート結果

が 4 項目存在した。それぞれ「OSS のリリースが迅速で早期に機能を利用できる」（品質要求 b）、「OSS のプログラム構造が容易でホワイトボックス化しやすい」（品質要求 u）、「ソースコード中のコメントが豊富で読解性が良い」（品質要求 v),「他の代替可能な OSS がある」(品質要求 w)であった。

品質要求 b については、既に OSS を開発母体に使用しており、早期の機能提供より OSS も安定を重要視するプロジェクトで低く評価された。当社の開発の多くはこの様な派生開発であることが平均値を低くした要因として挙げられる。ただし、先進分野(IoT 等）の新規開発においては、技術の進化速度が速く、リリースの迅速性を重視する開発も存在するため、品質要求として有益であると判断した。

品質要求 u,v については、OSS の利用方法によりプロジェクトの回答が分かれており、OSS をブラックボックスで利用するプロジェクトは重視度低く評価する傾向があり、改造して利用するプロジェクトは重視度を高く評価する傾向があった。OSS を改造して利用するプロジェクトが存在する事に加え、元々ブラックボックスで利用を計画していたプロジェクトでも結果的にコード解析や改造が必要になることを鑑み、品質要求としては有益であると判断した。

品質要求 w については、そもそも OSS の代替が念頭にないプロジェクトが多く存在した事が原因と考えられる。これは、派生開発の特徴に加え OSS の選択を顧客からの指定で決定されるプロジェクトが、品質要求の重視度を低く評価した事が原因であった。ただし、複数の OSS 候補に対して可能性を配慮することは、選択した OSS に大きなリスクが発生した場合には不可欠な品質要求となりえるため、開発者へ啓蒙の意味があると判断してそのまま採用した。

表3　RepOSS による事前評価指標の分類

OSS事前評価指標の分類
ライセンス情報
開発組織情報（コミュニティ、企業、主たる開発者、開発者の分布）
提供情報（メジャーリリース、メジャーリリース提供間隔）
コミュニケーション方法（オフィシャルサイト、ソースコードリポジトリ、バグトラッキングシステム、メーリングリスト）
仕様関係情報（標準準拠、ローカライゼーション）
構成（開発言語、プラグイン構造、GUIツール）
品質（バグ数、バグ修正率、平均バグ修正時間）
普及度（ダウンロード数、ユーザマニュアル、Webサイト数、書籍数、ユーザグループ、賞）
ポータビリティ（ディストリビュータパッケージ情報、インストーラ情報）
ビジネス（他ソフトでの利用、ベンダーによるサポートサービス）
事例

表3は日本OSS推進フォーラムプレスリリースを出展に作成　http://ossforum.jp/node/126

4. 品質要求項目と事前評価指標の対応

抽出した品質要求項目と、品質要求を評価可能な事前評価指標の対応付けを行った。対応付けの母集団として採用した事前評価指標は、過去の研究で提案された事前評価指標とソースコードの静的解析ツールで計測したソースコードメトリックス値とした。今回の研究ではコード解析ツールとし TechMatrix 社の Understand を採用した。

過去の研究で提案された事前評価指標は、日本OSS推進フォーラムが提供するオープンソースソフトウェア評価情報リポジトリ（RepOSS）の事前評価指標を採用した。これは他の研究組織の提唱する事前評価指標と比較し多種であり、系統立てて事前評価指標が定義されていたためであった。

加えて、Understand を用いて、オープンソースコードのソフトウェアメトリックスを計測した。Understand は、ソースコードを解析し、100 種類に及ぶソフトウェアメトリクスを計測可能である。ソフトウェアメトリックスには、複雑度を表すサイクロマチック複雑度や、結合性を表わす LCOM(結合性の欠如)、CBO（結合されクラスの数）、規模を表す LOC(コード行数)などが含まれる[11]。

これらの事前評価指標と抽出した品質要求の関連付けを実施した。表 4 に対応付けを行った品質評価指標の例と品質要求のカテゴリを示す。

表4. 品質要求カテゴリと採用した事前評価指標

品質特性	採用した事前評価指標（例）
機能適合性	書籍数・サイト数 マニュアル システムの開発事例 開発主体
性能（効率）性	動作環境:OS 動作環境：必要スペック 書籍数・サイト数
使用性	マニュアル パッケージインストーラ 書籍数・サイト数 他ソフトウェアへの適用
信頼性	開発主体 バグトラッキングシステムの有無 メーリングリストの流量 バグ数・フィックス率 Cyclomatic複雑度
セキュリティ	CVEEスコア 公開サイト
保守性	Cyclomatic複雑度 ネスト数 コード行に占めるコメント行の割合
互換性・移植性	ライセンス バージョンの互換性 マルチOS等への対応

特に、本研究の先行調査で、書籍数、Web サイト数、MaxNesting（ネスト数）、複雑度に関するソフトウェアメトリックス(5 種)の事前評価指標は OSS の品質と相関関係が深い事が確認されており[11]、該当する品質カテゴリの評価に結び付けることにした。

複雑度については、 5 種類の指標の相関関係がほぼ同一であったため、代表的な指標である AvgCyclomatic のみを採用した。

先行研究で研究された事前評価指標(RepOSS)とソフトウェアメトリックス(Understand)の事前評価指標を品質要求と対応付した結果、使用性の中にある書籍やドキュメントに関する品質要求や、信頼性にある OSS 自体のバグに関する情報やコミュニティの活動状況に保守性に関するリスク評価は、RepOSS の事前評価指標やメトリックス値と対応付ける事ができた。

しかし、品質要求によっては、品質要求を満たせる事前評価指標が存在しないものも発生した。性能や使用性のカテゴリに、そのままでは、事前評価指標が対応できない品質要求が存在した。

性能では、「OSS が指定された性能を満たす事ができる」や「性能測定のための環境作りが容易にできる」の品質要求がこれに相当し、使用性では「正常系が正しく動作しており，早期に機能を確認できる」が相当する。本来、対応する指標がないという事はリスクを判断する材料がなく、リスク有との判断をしてリスク管理の対象とするやり方が一般的である。今回の手法では極力、事前評価指標を品質要求に結び付ける事にした。例えば、「正常系が正しく動作しており，早期に機能を確認できる」について

は、多くのユーザに利用される OSS であれば、正常系動作が保証されている可能性は高いと考え、ダウンロード数や世の中の利用実績を関連指標として対応付けた。

しかしながら、ダウンロード数や世の中の利用実績が示す結果が良好であっても、品質要求が満たされるとは限らないため、結び付けた品質要求に対する事前評価指標の関連度を設定した。

品質要求に評価指標が直接関連する場合は関連度大(5点)とし、直接的に関連しないが、品質要求に何らかの関連があるものは、関連度を関連中(3 点)、関連小(1 点)として対応付けを実施した。

性能については、既存の事前評価指標には対応する評価指標は見つからなかった。例えば、他の導入事例等で性能に関する情報があっても、システムが動作する環境が違っていたり、性能評価のデータ自体が大変古いものであったりと有効に利用できないものが多かった。そこで、社内の利用実績やノウハウといった社内情報を評価指標に組み入れる事にした。これによって、実績の有無、件数に加えて、社内情報ならではの OSS を利用したシステムの特性や動作環境等の情報が把握でき、定性的に性能のリスクを検討できる可能性がある考えた。

最終的に採用した評価指標を表 5 に示す。以上の様に、当初 200 個強あった事前評価指標を今回抽出した品質要求に関連するもののみに絞り込んで、事前評価の仕組みに盛り込んだ。

5. OSS 事前評価手法について

本研究の OSS 事前評価手法は、開発者が利用する OSS への品質要件により変動する開発リスクを予め判断し、開発計画に反映するための手法である。従来手法と比べて、以下の特徴を持つものとした。

・開発者が OSS 利用時に意識している品質要求毎にリスク予測が可能となり、開発者の視点に近いリスク特定が可能となる。

・開発者が特別な知識やトレーニングなしに、一定のリスクを予測可能となる。

・開発者が複数の品質要求側面から網羅的に品質リスクを判断できる様になる。

表 5 採用した事前評価指標

事前評価指標	採用した評価指標
OSS評価指標 (REPOSS)	31種類 (詳細は表4参照)
ソフトウェアメトリックス	5種類 (Cyclomatic複雑度、コード行数、ネスト数 ファイル数、コメント行数、コメント行の割合)
社内利用実績	3種類 (社内の利用実績、トラブル報告数、ノウハウ数)

5.1. 手法の実施手順

OSS を利用して開発を行うプロジェクトでは、開発開始前に以下の手順で OSS の事前評価を行う。

(1) 品質要求アンケートの作成

OSS への品質要求の重視度を開発者の判断で記載する。重視度としては、重視する。やや重視する。やや重視しない。重視しない。の 4 段階の評価とした。

(2) 事前評価指標の算出

研究に先駆け当社で過去利用した OSS 約 120 種について、大学側で一般的な OSS 事前評価指標(RepOSS)とソフトウェアメトリックスについて、算出を行い、データベース化を実施した。

評価したい OSS の事前評価データがデータベース中にあれば、算出作業は不要であるが、データベース中になければ、新たに調査を実施して事前評価データを算出する。

(3) スコアリング

品質要求毎に予め事前評価指標を対応付けており品質要求毎のスコアが算出される。

(4) 事前評価レポートの作成

品質要求毎に品質要求の重視度と事前評価指標のスコアを開発者にレポート形式で通知する。開発者は品質要求の重視度に対して、事前評価のスコアが下回る場合、品質要求にリスク有と判断できる。

5.2. スコアリングルール

スコアリングを実施するに当たり、開発者が理解し易い様に、5 点満点で評価する様にした。

スコアは品質要求とそれらのカテゴリ毎に集計される。ここで品質要求 Qi(i=1～n)は、そのカテゴリ内で、図1に示したアンケート結果によって重みづけられる。重み付けの方法は、カテゴリに属する品質要求に対して、アンケートの結果から各品質要求の重みの合計が 100%となる様に、重み付けを行った。ただし重み付けについては厳密

には計算せずに、5%もしくは 10%単位で重み付けを実施した。

例えば、「機能適合性」の 3 つの品質要求の重視度の平均値は、「OSS 自身が実現したい機能を満たしている」は5、「OSS のリリースが迅速で早期に機能を利用できる」は 2.6、「正常系が正しく動作しており，早期に機能を確認できる」は 4.8 であった。これらの平均値からそれぞれの項目の重み(Bi)は、それぞれ 40%,20%.40%とした。

品質要求のカテゴリ毎の品質要求値(QV)は、品質要求の値(Qi)に重みを考慮し,以下の式で算出した。また、図2にスコアリングルールを示す。

$$QV= \sum_{i=1}^{n}(Qi * Bi)$$

品質要求に対するカテゴリ毎の事前評価指標のスコア(SV)も品質要求値のカテゴリ値(QV)と同様の重み(Bi)で算出した。品質要求毎の事前評価データのスコアを(Si)とするとき、品質要求カテゴリでの事前評価データのスコア(SV)は、以下の式で算出した。

$$SV= \sum_{i=1}^{n}(Si * Bi)$$

ここで各々の品質要求に対する事前評価データのスコア値(Si)は、複数の関連する事前評価指標より、算出される。品質要求に対応するそれぞれの事前評価指標に対して、品質要求に対する関連度(Ri)を決定した。関連度は、5 点満点として、関連大 5 点、中 3 点、小 1 点とした。それぞれの事前評価指標のスコア(Ei)は、5 点満点とし、指標毎にスコアの決定方法を定めた。例えば、書籍数や WEB サイト数の様に数値化可能なデータは、相対的に 5 段階評価でスコアを決定した。マニュアルの有無等の2極値であるものは、1 点、5 点とした。またデータ取得ができなかった場合は、スコアを 0 点とした。
事前評価指標の品質要求ごとのスコアは以下の式とした。また、図 3 にスコアリングルールを示す。

$$Si = \frac{\sum_{1=1}^{m}(Ei*Ri/5)}{(m-m_0)}$$

※m は品質要求に結びついた事前評価データの数、m_0 は Ei=0 データ取得不能で評価できなかった事前評価指標の数を示す。

5.3. 事前評価レポート

開発者から見た品質要求と事前評価指標によるスコアリング結果から、OSS の事前評価レポートを作成し、開発

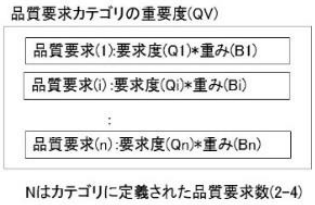


図 2 品質要求カテゴリのスコアリングルール

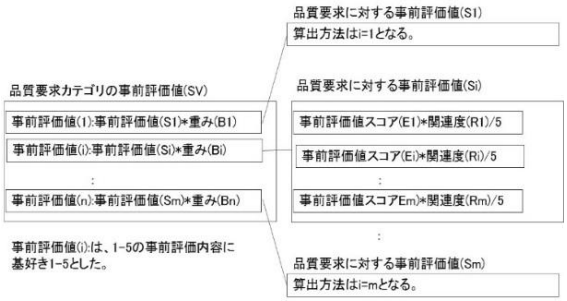


図 3 事前評価値のスコアリングルール

前にOSS利用時のリスクを特定できる様にした。事前評価レポートは、OSS の全体像を示す総括パートと個別の品質要求のスコアを示す品質要求評価パートの 2 つで構成される。

総括パート（図 4 参照）では、OSS 基本情報①として、OSS の公開ディレクトリやライセンス情報や機能概要が示される。総合判定チェック②として、各品質要求カテゴリのレーダチャートが記載される。レーダチャートでは、品質要求に対する期待度と事前評価指標によるスコアがカテゴリ毎に正規化され、レーダチャートの形状から、要求に対する期待度とスコアの関係が判断できる様になっている。ここで、事前評価指標のデータが一切取得できず、事前評価の評価値が決定できなかったカテゴリについては、スコア値を 0 としてレーダチャート上に表示した。レーダチャート上は要求を大きく下回る結果となり、リスク項目として表示される。

③に社内での OSS の利用状況、問題発生状況、社内ノウハウの一覧が表示される。

品質要求パート(表 6 参照)では、品質要求カテゴリ④毎に、品質要求⑤に対して、開発者が選択した重視度⑥と事前評価データによりスコアリングした事前評価スコア⑦（評点）が記載される。これらの品質要求毎のスコアを品質要求の重み付けを加味してカテゴリの重視度と価値を求め、（算出方法は 5.2 スコアリングルールを参照）、カテゴリ評価⑧とした。総合評価⑨は。カテゴリ評価の重視度と評価値を比較し、評価値が重視度以上であれば○:リスク低、評価値が重視度より低く差が 2 点以下であれば△:リスク中、2 点より大きければ、×;リスク大とした。

図 4. 事前評価レポートの例　総括パート

ここで△と×の閾値 2 点は 5 段階評価の中間点(1,3,5)の間隔から設定した。閾値の正当性については検討の余地があったが、後のトライアル結果を見て判断する事にした。総合評価が△、×の品要求項目については、評価結果に関する理由をコメント⑩として記載し、特に原因となった事前評価指標の内容を記載した

6 事前評価手法のトライアルと評価結果

研究した事前評価手法を用いて、事前評価手法の効果を評価した。評価対象として、2015 年度下期から 2016 年度上期に開発を完了したプロジェクト 12PRJ の内、任意の 4PRJ を対象とした。1 プロジェクトで複数の OSS を利用している場合があり、評価対象となった OSS は 6 種類であった。

トライアルは、開発メンバにより、品質要求の重視度を回答してもらい、トライアル実施チームにより、事前評価レポートを実施し、作成した事前評価レポートを元に実際に開発を実施した開発メンバとレビューを行い、事前評価レポートと開発実績を比較する事で実施した。

開発実績で問題が発生した項目は、事前評価時のリスクがどの様に検知されたかを確認するため、原因のレビューを実施した。また、カテゴリ毎に評価として、事前評価レポートを元に、カテゴリ毎にリスク予測と開発実績を比較した。開発実績は、品質要求のカテゴリ毎に〇：問題発生はなかった。△：問題までは至らないが、OSS導入に課題があった。×：OSS利用の問題が発生した。の三段階で評価した。

6.1 発生した OSS 利用時の問題とリスク検出の調査結果

今回のトライアルで評価を実施した OSS6 種について、延べ 5 件の OSS 利用時の問題が顕在化した。ここで挙げる問題については、OSS の機能的な障害(バグ)に加えて、OSS 利用時の問題を含んでいる。例えば、OSS の機能自体に問題はないが OSS の利用方法調査に想定外

表 6　事前評価レポート　品質要求パート

④カテゴリ	⑤品質要求	⑥重視度	⑦評価値	⑧カテゴリ評価 重視度	⑧カテゴリ評価 評価値	⑨総合評価	⑩コメント
使用性	OSS自体の操作性が利用者の要求を満す。	5	1.7	4.8	2.8	△	操作性とドキュメントの充実にリスクがある。 ・Webサイト数が少ないためOSS自体の情報が十分に得られない可能性がある。 ・社内ノウハウも未登録である
	ドキュメントや書籍が揃っている。	5	3.0				
	OSSの操作が容易にできる。	4.3	2.6				
	OSSの利用例が多い	5	4.0				
セキュリティ	脆弱性に関する報告小	2	3.0	2.0	3.0	〇	
保守性	プログラム構造がホワイトボックス化しやすい	5	2.1	5.0	2.0	×	ソースコードの構造、可読性ともリスクが高い。 ・複雑度の評価(スコア1) ・コメント率の評価(スコア2) ソースコードの解析のための工数を開発工程に盛り込む事を推奨する。
	ソースコードの読解性が良い	5	2.0				

表 7　発生した OSS 利用時の問題

発生OSS	問題内容	OSSの障害	利用時の問題
OSS1. OSS2	問題1：不用処理による性能問題	一部該当	該当する
OSS1 OSS2	問題2：情報不足による利用方法不明	該当しない	該当する
OSS3	問題3：ハードウェアの互換性不足	該当しない	該当する

の工数が発生した問題（問題 2）や初期調査が不完全であったため、開発後半でハードウェアの互換性が発生した問題（問題 3）がそれに当たる。これらの問題も何らかのリスク管理ができていれば、開発工程遅延や予定外の対応工数が不要であったと考え、問題として抽出した。

問題 1,2 は、問題の原因が OSS1,OSS2 のインターフェースや処理手順にあり、問題の解決には、OSS1,OSS2 双方の修正が必要となったため、両 OSS の問題とした。

問題 1 の内容は、OSS の処理に不要な処理が入っており、動作は正常に動作するが、システムの要求性能に至らかったという事象であった。また不要な処理を除去するために OSS の改造を実施したが、OSS の内容理解に多くの時間を費やすことになり、工程遅延を招いた。

問題 2 は、OSS の機能を利用する際、必要なパラメータ設定などのドキュメントやサンプルコードが不足しており、利用方法が不明であった。利用方法を理解するため、OSS 内部の解析に多くの時間を費やす事になった。

問題 3 は、OSS を利用した際、システムで利用するハードウェアに対応していないことが工程途中で判明し対応を実施しなければならなかった。この問題も急遽 OSS を解析して、独自の改造を実施した。

各問題ともに、直接の原因に対応する品質要求に加えて、「実現したい機能を実現できる。」とソースコードの解析で苦労をしたことから、ドキュメントやホワイトボックス化に対する品質要求が問題発生の品質要求として挙げられた。

表 8 発生問題と品質要求の対応

問題	OSS	カテゴリ	品質要求	要求スコア	評価スコア	事前評価
1	OSS1	機能適合性	実現したい機能を実現できる。	5	2.4	×
		性能	指定された性能を満たすことができる。	4	1.2	×
		保守性	構造が容易でホワイトボックス化しやすい。	5	2.1	×
2	OSS1	使用性	ドキュメントが整っていて情報が得られる	5	3.0	△
		保守性	構造が容易でホワイトボックス化しやすい。	5	2.1	×
1	OSS2	機能適合性	実現したい機能を実現できる。	4	1.6	×
		性能	指定された性能を満たすことができる。	4	0.6	×
		保守性	構造が容易でホワイトボックス化しやすい。	5	2.2	×
2	OSS2	使用性	ドキュメントが整っていて情報が得られる	5	2.5	×
		保守性	構造が容易でホワイトボックス化しやすい。	5	2.2	×
3	OSS3	機能適合性	実現したい機能を実現できる。	4.2	1.4	×
		互換性	複数種類のOSやハードに対応している	4	1.5	×
		使用性	ドキュメントが整っていて情報が得られる	5	3.2	△
		保守性	構造が容易でホワイトボックス化しやすい。	5	2.0	×

表 8 に問題発生に関連があった品質要求の一覧と事前評価の結果を示す。延べ 14 種の大半にあたる 12 種の品質評価指標でリスク大（×）の評価であった。

問題1の直接原因と対応する品質要求の評価詳細を表 9 に示す。性能については、社内での利用実績とノウハウで評価しており、利用実績無しの場合の評価は1、ありの場合は 3、類似システムでの利用有りの場合 5 の評価とした。ノウハウも同様に 1、3 として、ノウハウに性能の記述がある場合を 5 とした。これらのデータに品質指標との関連度を 5 点満点で設定し、5.2 項に示すスコアリングルールで計算した結果が関連度反映後のスコアとなる。いずれの OSS もノウハウ、実績もほぼないため、関連度反映後のスコアは 1.2、0.6 と大変低くなっており、要求の重視度 4 を大きく下回っている。これにより、リスク大（×）との評価結果となり、問題 1 の発生リスクを抽出できた。

問題 2 の直接原因と対応する品質要求の評価詳細を表 10 に示す。品質要求は、5 つの事前評価指標で評価される。WEB サイト数、書籍数、SLIDESHARE は、当社でデータを収集した全 OSS の事前評価指標から相対的に 1～5 点でスコアを算出している。マニュアルやユーザ会については有無で 1 点もしくは 5 点の評価としている。問題 1 の例と同様に関連度反映後のスコアを算出した結果は、それぞれ、3、2.5 であった。要求の重視度は 5 であり、OSS2 はリスク大、OSS1 はぎりぎりの点数リスク中と評価された。OSS1,2 共にほぼ問題 2 の発生リスクを抽出できた。

問題 3 の直接原因と対応する品質要求の評価詳細を表 11 に示す。互換性については、OSS のバージョン互換性と複数ハードウェアの対応で評価した。評価の詳細を表 11 に示す。複数バージョンの対応は、対応していたため 5 点となった。複数ハードウェア対応は、OSS3 は対応していないため、1 点となった。関連度は 3 であり、スコアは 1.5 点となった。結果、品質要求の重視度 4 に対して下回っており、問題 3 の発生リスクを抽出できた。更に、機能適合性や保守性についても、同様に事前評価結果を確認して、問題に対するリスクを抽出できたことを確認した。

表 9 性能の品質要求評価の詳細

事前評価指標	関連度	評価 (OSS1)	評価 (OSS2)
社内のノウハウの有無	3	1	1
社内での利用実績	3	3	1
スコア（関連度反映後）	—	1.2	0.6

表 10 使用性の品質評価の詳細

事前評価指標	関連度	評価 (OSS1)	評価 (OSS2)
WEBサイト数	5	1	1
書籍数	5	3	2
SLIDESHARE	5	–	2
マニュアルの有無	5	5	5
ユーザ会の有無	5	–	–
スコア（関連度反映後）	—	3	2.5

表 11 互換性の品質評価の詳細

事前評価指標	関連度	評価 (OSS3)
複数Version対応	3	5
複数ハードウェア対応	3	1
スコア（関連度反映後）	—	1.5

6.2 品質カテゴリ毎の評価

次に、作成した事前評価レポートを元にトライアルチームが開発チームに開発時の問題発生状況をヒアリングした。ヒアリング時には、開発チームの回答がリスク予測結果に左右されない様、カテゴリ毎のリスク予測結果は伏せて、カテゴリ毎の問題発生状況を、問題発生、多少問題あり（多少問題があったのもの開発に大きな支障はなかった）、問題なしの 3 択で回答して貰った。表 12 に事前評価レポートでのリスク判断件数と開発チームでのヒアリング結果を示す。

リスク予測と開発チームの評価を比較した件数の集計結果を以下に示す。

- リスク予測が実績評価と一致した件数　17 件
- リスク予測より実績評価と悪化した件数　2 件
- リスク予測より評価と好転した件数　22 件

(1) リスク予測が実績評価と一致したカテゴリの分析

リスク予測が評価と一致したカテゴリは、6.1 項で述べたリスク予測が問題として顕在化したカテゴリ 10 個に加え

表 12 品質カテゴリ毎のリスク予測と開発チームでの評価

事前評価レポートの評価結果		開発実績に伴う開発チームの評価	
事前評価	カテゴリ件数	実績	カテゴリ件数
リスク低	5	問題なし	5
		多少問題あり	0
		問題発生	0
リスク中	13	問題なし	9
		多少問題あり	2
		問題発生	2
リスク大	23	問題なし	11
		多少問題あり	2
		問題発生	10
評価不能	1	問題なし	1

表 13 リスク低で、問題が発生しなかったカテゴリ

OSS	カテゴリ	重視度	評価値
OSS1	セキュリティ	2.0	3.0
OSS3	セキュリティ	1.0	1.0
OSS4	保守性	1.0	2.4
OSS5	保守性	1.0	2.4
OSS6	保守性	1.0	2.5

て、リスク低で問題が発生しなかったカテゴリ 5 個とリスク中で多少問題があったカテゴリ 2 個であった。

リスク低で、問題が発生しなかったカテゴリの評価を表 13 に示す。いずれもカテゴリについての重視度が低く、開発者から見ても問題の顕在化の可能性が薄い品質要求であった。予測通り、問題は発生せず、リスク予測と結果が一致した。

リスク中で多少問題があった2カテゴリは、問題 2 に関連するカテゴリであった。

開発チームは、OSS1,2 の信頼性のカテゴリを多少問題ありと評価した。これは問題 2 の対応中にソースコードを解析し、ソースコードの構造が大変複雑であった事から開発上では問題ないものの、OSS の品質に開発チームが懸念を持った事から、多少問題ありの評価となった。信頼性に対する事前評価結果もリスク中との結果となっており、正しくリスク予測ができたと判断した

(2) 事前評価より結果が悪化したカテゴリの分析

事前評価より、結果が悪くなってしまった 2 項目については、リスクが高いにも関わらず、リスクに対する配慮が低下してしまう可能性があるため、重要な問題と認識した。その原因として、問題発生に対応した品質要求のリスク評価結果はリスク大であるものの、同一カテゴリの他の品質要求の評価結果がさほど悪くなかったため、カテゴリとしてのスコアが良くなってしまった事が挙げられる。

表 14 に該当カテゴリの 1 つである機能適合性カテゴリの事前評価詳細を示す。問題発生に関連する品質要求は「実現したい機能が実現できる。」であり、これはリスク大との判定結果であったが、残る 2 つカテゴリの内、1個がリスク低であり、カテゴリ全体としては、リスク中の判定となった。

表 14 機能適合性カテゴリの事前評価詳細

品質要求	要求の重み	重視度スコア	スコア	結果
実現したい機能を実現できる。	0.4	5	2.4	×
リリースが迅速である。	0.2	2	2.5	○
正常系が正しく動作する、	0.4	5	2.4	×
合計（カテゴリ評価）	1.0	4.4	2.4	△

表 15 OSS 毎のリスク予測＞実績のカテゴリ件数

OSS	問題あり	リスク予測＞実績	社内での利用実績
OSS1	有り	1	1
OSS2	有り	0	0
OSS3	有り	2	0
OSS4	無し	6	3
OSS5	無し	6	7
OSS6	無し	6	5

現在のカテゴリ評価は品質要求の重み付けでスコアを計算し判定しているが、これに加えて、カテゴリ内品質要求のリスク大の有無を考慮して、リスク大の項目がある場合、カテゴリ評価をリスク大にするといったカテゴリ評価方法の改善が必要と考えた。

(3)事前評価結果より結果が良好なカテゴリの分析

事前評価でリスク有と判断したが、問題なしとなったカテゴリが 22 個存在した。

リスク管理において顕在化しないリスクも当然あり得るため、事前評価結果より結果が好転することは止むを得ないが、不要なリスク項目が数多くある場合、開発者に余計な負担がかかってしまう。また多数のリスクがある場合、重点的に扱われるリスクを明確にするためにも、リスクの検出の精度を向上させる必要があると判断し、リスク予測より結果が向上した品質カテゴリの分析を行った。

分析の結果、大きく 2 つの原因があると分析した。

1 つ目は、OSS 評価方法の問題である。表 15 に、OSS 毎の問題発生の有無、リスク予測より実績が向上したカテゴリ数ならびに、事前評価レポートに記載された OSS の利用実績件数を示す。

OSS(OSS1～3)は、リスクが顕在化して、問題が発生した OSS であり、当然リスク予測の精度が上がっている。一方問題発生がなかった OSS は、ほとんどの品質カテゴリでリスク予測より、実績が向上しており、過剰にリスクを予測した結果になった。

OSS1～3 は、社内では新規導入（もしくは導入実績が少ない）が OSS4-6 は、社内の利用実績が比較的多く、複数のプロジェクトで使われており、従来の使い方であれば新たな問題が発生することは稀である。今回のトライアルでも OSS4-6 について問題発生はなく、予測したリスクも顕在化することはなかった。

利用実績が少なからず OSS リスク予測に有効な事は理解していて評価指標に盛り込んでいるが、利用実績がスコアに関与する影響は必ずしも高くない。利用実績が

スコアに与える割合を補正して、事前評価のスコアに与える影響を上げればリスク予測の精度向上に貢献できると分析した。

2 つ目は、品質要求に対する重視度の定義である。リスク予測より実績が向上した品質要求カテゴリの多くに重視度が想定よりも高いものが多く見受けられた。現手法では、重視度を開発プロジェクトの主観で決定している。

例えば、性能での「OSS が指定された性能を満たす事ができる」という設問は、文言の捉えようによっては、顧客から要求される性能とは無関係に、「重視する」と回答される可能性があり、今回のトライアルでもその様に回答したプロジェクトも見受けられた。品質要求の重視度についても、設問内容を含めて、改善の余地があると判断した。

7 まとめ

本研究では開発現場への普及や運用の容易性を目的に開発者の持つ品質要求を元に OSS 利用リスクを特定する手法を考案し、トライアルを実施した結果、以下の評価結果を得た。

・OSS 利用時発生した問題 5 件について、品質要求単位、品質カテゴリ単位の双方で、リスク高、中として検知できた。

・リスク予測と開発チームの評価実績が一致したカテゴリ数は 17 個であり、全カテゴリ数 42 個に対して約 40%の確度でリスク予測がトライアル結果と一致した。

以上により、本手法は予測精度向上の余地はあるが、リスク評価の手法として有用であると判断した。

8 今後の課題

(1) リスク予測精度向上の課題

リスク予測において、問題発生を検知できた反面、過剰にリスクを検知している可能性がある。品質要求の入力のやり方やスコアリング方法など、更なる改善の余地が残っている。本稿の執筆時、トライアルは継続中である。トライアル終了時に、データを再度集計、分析して、手法にフィードバックしリスク予測精度を向上させていきたい。

(2) 予測コストの明確化と更なる削減

本手法の実行において、大きく 2 つのコストが必要となる。1 つは、評価のための事前評価情報の収集コストであり、これは共同研究者である九州大学の研究にて、1OSS 当たり、30 分～2 時間とのベンチマーク結果が得られている[12]。取得手順マニュアルの整備や自動化を進めて、更なる運用コスト削減を進めていきたい、

2 つ目は、事前評価レポートの作成時間である。今回手法の検証作業も併せて、評価レポート作成を行ったため、最初のレポート作成には 2 週間、作業が慣れてきた後半でも 2-3 日を要している。正確な作成時間の計測を進めると共に、スコアリング等、一部ツール化、自動化を図り、コストの削減を実施していく。

謝辞

当社が過去に利用した OSS 約 170 種について、延べ 3 万種に及ぶ OSS の事前評価データを収集して頂いた九州大学）鵜林研究室の学生の皆様に感謝致します。

参考文献

[1] 独立行政法人情報処理推進機構：第3 回オープンソースソフトウェア活用ビジネス実態調査（2010）.

[2] Atos: Quali_cation and selection of open source software (QSOS), version 2.0. (2013).

[3] Duijnhouwer, F.-W. and Widdows, C.: Open Source Maturity Model, *Capgemini Expert Letter* (2003).

[4] Northeast Asia, O.: RepOSS: A Flexible OSS Assessment Repository (2012).

[5] Petrinja, E., Nambakam, R. and Sillitti, A.: Introducing the opensource maturity model, *Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009. FLOSS'09. ICSE Workshop on*, IEEE, pp. 37{41 (2009).

[6] Taibi, D., Lavazza, L. and Morasca, S.: OpenBQR: a framework for the assessment of OSS, *IFIP International Conference on Open Source Systems*, Springer, pp. 173-186 (2007).

[7] Wasserman, T. and Das, A.: Using FLOSSmole Data in Determining Business Readiness Ratings, *WoPDaSD* (2007).

[8] Deprez, J.-C. and Alexandre, S.: Comparing assessment methodologies for free/open source software: OpenBRR and QSOS, *International Conference on Product Focused Software Process Improvement*, Springer, pp. 189-203 (2008).

[9] Petrinja, E., Sillitti, A. and Succi, G.: Comparing OpenBRR, QSOS, and OMM assessment models, *IFIP International Conference on Open Source Systems*, Springer, pp. 224{238 (2010).

[10] Petrinja, E. and Succi, G.: Assessing the open source development processes using OMM, *Advances in Software Engineering*, Vol. 2012, p. 8 (2012).

[11]情報処理学会：第190回ソフトウェア工学研究会製品開発におけるOSS導入のためのOSS事前評価に向けた初期調査　松本、山下、鵜林、亀井、深海、岩崎(2016)

[12]日本ソフトウェア科学会：第23会ソフトウェア工学の基礎ワークショップ(FOSE2016) 開発現場への導入を想定したOSS事前評価手法確立に向けた調査　松本、山下、鵜林、亀井、大浦、高山、岩崎(2016)

[13] コンピュータソフトウェア,Vol33.No1(2016) pp28-40,:Open Source Software Engineering 伊原　大平

開発者の所属企業規模を考慮したソフトウェア工学の有用性評価

村上　優佳紗
近畿大学大学院総合理工学研究科
m.yukasa@gmail.com

角田　雅照
近畿大学大学院総合理工学研究科
tsunoda@info.kindai.ac.jp

要旨

ソフトウェア工学の研究目標を設定する際，開発者の総数が多いボリュームゾーンのニーズを考慮する必要がある. 日本では 300 人未満の中小企業及びそこに所属する開発者が多数を占めており, ここがボリュームゾーンといえる. ただし，ソフトウェア工学の最先端研究が, これらの企業に所属する開発者のニーズに応えられているかどうかは明らかではない. そこで本研究では，ソフトウェア工学の最先端研究が，中小企業に所属する企業のニーズに対応できているかどうかを分析した. 具体的には，ソフトウェア開発企業に所属する開発者に対し, 最先端研究が自分の業務に有用であるかどうかを評価してもらった. その後，開発者が所属する企業の規模によりデータを層別して分析を行った. その結果，所属する企業の規模によって, 業務に関連する研究カテゴリに差異が見られたが，研究の有用性及び興味の評価については，規模による差異は見られなかった.

1. はじめに

近年, ソフトウェアは企業の基幹業務システムや機器に組み込まれて広く利用されており, 社会基盤として欠かせないものとなっている. このため，ソフトウェアに高い信頼性が求められるようになっているが, その一方で, 大規模なソフトウェアを短期間で開発することも同時に求められている. このようなソフトウェア開発が成功する確率を高めるために, ソフトウェア工学による開発の支援が求められる.

ソフトウェア工学による開発の支援を検討する際，開発者の総数が多いボリュームゾーンのニーズを考慮する必要がある. 中小企業関連立法では，ソフトウェア業・情報処理サービス業について，資本金 3 億円以下または従業員 300 人以下を中小企業としている[2]. 従業員 300 人未満の企業の割合は 99%，それらの企業においてソフトウェア開発に従事している者の割合は 69%であり[7], 300 人未満の中小企業及びそこに所属する開発者が, 日本のソフトウェアベンターのボリュームゾーンであるといえる.

ただし，ソフトウェア工学の最先端研究が，ボリュームゾーンである, これらの企業に所属する開発者のニーズに応えられているかは明らかでない. ソフトウェア工学の研究の有用性や方向性について検討した研究がいくつか存在するが[8][11]，これらの企業は Facebook や Microsoft など，規模の大きなソフトウェア開発会社に所属する開発者を対象としている. 中小企業と大企業では, 各ソフトウェアプロジェクトに従事するメンバー数や, 開発対象のソフトウェアの規模が異なることが多いと考えられる. このため，所属する企業の規模が異なれば, 技術者のニーズも異なる可能性がある.

本研究では，ソフトウェア工学の最先端研究が，中小企業に所属する企業のニーズに対応できているかどうかを分析する. ソフトウェア工学の最先端研究が大企業のニーズに応えることも重要であるが, ボリュームゾーンである中小企業のニーズの考慮も怠るべきでないと考える. 分析では，ソフトウェア開発企業に所属する開発者に対し, ソフトウェア工学の最先端研究が自分の業務に有用であるかどうかを評価してもらった. その後，開発者が所属する企業の規模によりデータを層別して分析を行った. また，開発者による研究の有用性評価を支援するために, 開発者の業務に関連する研究カテゴリとそれら属する研究の評価を，協調フィルタリングを用いて予測し，精度を評価した.

2. 関連研究

ソフトウェア工学の研究がソフトウェア開発に対して与えた影響について分析した取り組み（Impact Project）[9]では，プログラミング言語や構成管理ツー

表 1　回答者の概要

従業員数	回答者数(人)	経験年数 3 年未満(人)	平均経験年数(年)	修士号取得者(人)
300 人未満	6	4	6.7	2
300 人以上 1000 人未満	5	5	1.2	2
1000 人以上	5	5	1.2	1

表 2　回答者が関連する開発工程

従業員数	要求分析	基本設計	詳細設計	テスト	プログラミング	開発支援	プロジェクト管理
300 人未満	33%	67%	67%	100%	83%	33%	17%
300 人未満(経験年数3年以上除く)	0%	50%	50%	100%	75%	0%	0%
300 人以上 1000 人未満	20%	60%	60%	80%	80%	20%	0%
1000 人以上	20%	60%	60%	100%	80%	0%	20%

ルなどに対して，ソフトウェア工学がそれらの発展にどのように寄与してきたかを分析している[3][12]．その結果，この取り組みでは，研究が実際の開発方法に影響を与えていることを結論づけている．ここから，これまでのソフトウェア工学の研究に有用性があることはわかるが，最先端の研究も同様に有用なのか，また，どの程度の割合の研究が有用なのかは明らかではない．

ソフトウェア工学の研究の方向性について示唆した研究として，Rubin らのもの[11]が挙げられる．Google や Facebook などの先端のソフトウェア開発会社に所属する開発者にインタビューし，彼らが直面する問題を明らかにするとともに，その問題に基づき，開発者が必要な情報に素早くアクセスすることをサポートする研究が望まれることなどを示している．ただし，この研究では，先端のソフトウェア開発企業以外の開発者のニーズは明らかにしていない．

本研究と類似した取り組みとして，2 つの従来研究[1][8]が挙げられる．これらは同じ研究グループによって実施されたものであり，ソフトウェア開発企業に所属する開発者が，最先端の国際会議の論文の要約（1-2 行）を読み，それに対して研究の価値があるかどうかを 5 段階で評価している．その結果，7 割前後の研究が，価値があるという結果となっている．調査方法は基本的に Zimmermann ら[1][8]と同様の方法であるが，本研究とこれらの研究との違いは 3 つある．1 つ目は各研究に対する評価の観点である．本研究では，「自分の業務に有用であるか」という観点で評価してもらっていることであり，これが従来研究との最も大きな違いである．2 つ目は，所属する企業の規模の違いに着目して分析していることである（従来研究ではマイクロソフトなど巨大企業に所属する開発者が回答者の主体となっている）．3 つ目は，個別の研究内容に対して，より詳細な説明資料を読んでもらった上で評価してもらったことである．その結果，後述するように，上記従来研究と異なり，最先端の研究に対する有用性の評価は低くなっていた．

3. データ収集

分析で用いるデータは，ソフトウェア開発企業に所属する開発者から，アンケートフォーム（Google フォーム）を用いて収集した．研究に対する有用性などの評価について，以下の手順に従って回答してもらった．

1. 業務に関係する研究カテゴリを最大 5 つ（3 つは必須）選ぶ．
2. 上位 3 件の研究カテゴリに関する論文の資料を読む．
3. 「業務に役立つか」，「興味を持ったか」について，それぞれ 5 段階（1 が最低，5 が最高）で評価する．

研究カテゴリの分類や，各カテゴリに含まれる研究論文については，ソフトウェア工学の国際会議で最も権威のある ICSE (International Conference on Software Engineering)の，2016 年開催分のものを参考にした．具体的には，会議の各セッションを研究カテゴリとした．研究カテゴリの詳細については，4.2 節で後述する．各カテゴリ（セッション）には論文が 4 本程度含まれる．回答者の負担を減らすため，それらを直接読んでもらうのではなく，日本語の要約資料[6]を回答者に読んでもらった．資料は一本の論文につき 2，3 ページのスライド形式となっており，比較的読みやすい内容となっている．

また，各開発者について，以下の項目について回答してもらった．

表 3　研究カテゴリの概要及び回答者の業務に関連する割合

No.	カテゴリ名	論文数	総計	300 人未満	300 人未満 3 年未満	300-1,000 人	1,000 人以上
1	Android (android で動くソフトウェアに関する研究)	4	2%				8%
2	Performance (ソフトウェアの性能)	4	4%			7%	8%
3	Empirical (ソフトウェア開発データの分析)	3					
4	Symbolic Execution (シンボリック実行によるソフト解析)	4					
5	Compilers and Emerging Trends (コンパイラ・最新トピック)	4					
6	Energy and Videos (ソフトウェア省電力化・ビデオの活用)	4					
7	Open Source (オープンソースソフトウェア)	2	4%	11%	8%		
8	Defect Prediction (ソフトウェアの欠陥予測)	4					
9	Synthesis (自動プログラム生成)	4	4%			7%	8%
10	API (Application Programming Interface)	3	4%	11%	8%		
11	Code Smells (コーディングの方法論)	4	2%				8%
12	Architecture (ソフトウェアの構造)	3	4%	6%		7%	
13	Testing (ソフトウェアのテスト)	8	20%	17%	17%	21%	23%
15	Effort Estimation and Search (開発工数見積もり)	3	2%	6%	8%		
16	Product Lines (プロダクトラインによるソフト再利用)	4					
17	Repair and Model Synthesis (プログラムの自動修正)	4					
18	Languages (ソフトウェア開発言語)	4	4%	6%	8%	7%	
19	Debugging (ソフトウェアのデバッグ)	4	9%	11%	8%	7%	8%
20	Requirements (ソフトウェアに対する要求)	4	7%	6%	8%	7%	8%
21	Dynamic Analysis (ソフトウェアの動的な解析)	4	2%			7%	
22	Security (セキュリティ)	4	2%	6%			
23	Collaborative (ソフトウェア開発の人的要因)	4	4%			7%	8%
24	Software Quality (ソフトウェアの品質)	3	9%	6%	8%	7%	15%
25	Program Analysis (プログラムの解析)	4					
26	Concurrency (ソフトウェアの並行実行)	4					
27	Maintenance (ソフトウェアの保守)	4	13%	17%	25%	14%	8%

- 取得学位（学士，修士など）
- 実務経験年数
- 所属する企業の従業員数（300 人未満，300 人以上 1,000 人未満，1000 人以上）
- 担当する開発工程

4. 分析

分析の目的を明確にするために, 以下の 4 つのリサーチクエスチョンを設定した.

- RQ1: 業務に関連する研究カテゴリに偏りはあるか?
- RQ2: 所属する企業の規模によって，関連する研究カテゴリに差異は生じるか?
- RQ3: 最先端研究の内容は業務に有用か?
- RQ4: 所属する企業の規模によって，研究の有用性に差異は生じるか?

4.1. 回答者の概要

アンケートの回答者はソフトウェア開発企業に勤務する 16 人である．回答者の概要について，所属企業の従業員数別に層別したものを表 1 に示す．従業員数 300 人未満の会社の回答者グループでは, 開発経験年数が 3 年を超える回答者が 2 名いたが, その他のグループでは経験年数が 3 年未満であった. このため, 以降の分析において企業の規模別に回答傾向などを確かめる場合, 経験年数が 3 年以上の回答者を除外した場合についても示す.

業務に関連する工程: 回答者の業務に関連する工程について, 所属企業の従業員数別に層別したものを表 2 に示す. テストとプログラミングに関してはほとんどの回答者が関連し, 設計に関しても半数以上の回答者が関連している. 要求分析や開発支援などについては，300 人未満の従業員数の企業の場合，経験年数が浅い場合（表の 2 行目）は関連していないが，それ以外の規模の企業の場合, 割合は低いが関連している場合があった.

4.2. 業務に関連する研究カテゴリ

ICSE での研究カテゴリを表 3 に示す．会議での各

セッションを各研究カテゴリとした．カッコ内は，セッションに含まれる研究の内容を考慮して筆者らが日本語に訳したものである．Testing のみ 2 つのセッションがあったため，合わせて 1 つのセッションとして扱った．各セッションにつきおおむね 4 本程度の研究論文が含まれる．

回答者全体の傾向(RQ1 に関する分析)：業務に関連すると回答のあった研究カテゴリについても表 3 に示す．総計は従業員数別に層別しない場合に関連があると答えた割合を示す．3 章で説明したように，少なくとも 1 人あたり 3 つのカテゴリを選択しており，その総選択数(少なくとも回答者 16 人×3 カテゴリ）を分母として割合を算出している．その他の列は層別して同様の方法により割合を算出している(各列で合計 100%としている）．

全体でも 36%（26 カテゴリ中 9 カテゴリ）のカテゴリが業務と関連があると答えられていなかった．これは，一部の研究が現時点では一般的な開発と関連が薄いためであると考えられる．例えば Compilers and Emerging Trends (コンパイラ・最新トピック)や Energy and Videos (ソフトウェア省電力化・ビデオの活用)などが該当する．

開発支援を業務とする回答者は若干いたにも関わらず， Empirical (ソフトウェア開発データの分析)などの分析関連のカテゴリは選択されていなかった．このことから，従業員数が 300 人未満の場合（このグループには経験年数が 3 年以上の回答者が含まれる），または，経験年数が 3 年未満の場合，開発支援の業務においてデータ分析は関連が弱い（優先順位が低い）可能性がある．

上記の分析結果より，RQ1 に対する答えは，「偏りがある(業務と関連が弱い研究カテゴリがいくつか存在する)」となる．

企業規模別の傾向(RQ2 に関する分析)：Performance (ソフトウェアの性能)や Collaborative (ソフトウェア開発の人的要因)などについては，従業員数 300 人未満の会社の回答者グループでは，業務との関連が弱かった．なお，300 人以上の企業で上記それぞれに関連があると答えている回答者は 4 人であり，例外的な回答者 1 人が答えたわけではない．表 2 で示したように，回答者が関連する工程に大きな違いがないことから，それらがこの結果に影響している可能性は低い．300 人以上の企業に所属する回答者の経験年数は 3 年未満であるが，300 人未満の企業に所属する回答者は 3 年以上の経験を持つ開発者も含まれていることから，経験年数についても影響している可能性が低い．この結果は，開発対象とするソフトウェアの性質の違いや，開発体制の違い（プロジェクトあたりの開発者数の違いなど）が影響している可能性がある．これらは企業の規模との関係が強いと思われる．

表 4　研究の有用性および興味（従業員数別）

従業員数	有用性	興味
300 人未満	2.2	3.6
300 人未満 (経験年数 3 年以上除く)	2.2	3.1
300 人以上 1000 人未満	2.4	2.9
1000 人以上	2.4	2.7

表 5　研究の有用性および興味（学位別）

最終学位	有用性	興味
学士他	2.3	3.0
修士号	2.3	3.5

API (Application Programming Interface) や Open Source (オープンソースソフトウェア)については，従業員数 300 人未満の企業の回答者については業務との関連が強く，それ以外の企業の回答者の業務とは関連が弱かった．経験年数 3 年未満と 3 年以上両方の開発者がこれらを回答していたことから，経験年数が結果に影響している可能性は高くないと考えられる．また，前述のように関連する工程に大きな違いはないことから，開発対象のソフトウェアの規模が比較的小さく，API や OSS の影響が相対的に大きいことが，このような結果につながった可能性がある．

これらの結果より，関連する（担当する）工程に大きな差がなくても，所属する企業の規模が異なる場合，業務に関連が強い研究カテゴリが異なってくる可能性があるといえる．よって，RQ2 に対する答えは「差異が生じる」となる．

4.3. 研究の有用性などに対する評価

本節では RQ3，RQ4 に関する分析結果を述べる．各研究カテゴリに含まれる研究論文に対して，業務に有用かどうか，研究内容に興味があるかどうかを評価してもらった結果を表 4 に示す．所属する企業の従業員数別に層別したが，規模によって有用性の評価に大きな違いはなかった．評価の平均値は 3 を下回っており，ソフトウェア工学の最先端研究は，一般的なソフトウェア開発に対して業務改善などの即効性は強くない可能性がある．

表 4 において，研究の有用性と比較して，研究に対する興味は低くなかった．ただし，興味の平均値は回答グループによって異なっていた．研究に対する興味は，大学院などで研究に取り組んだ経験があるかどうかが影響している可能性がある．そこで回答者を取

表 6　研究カテゴリ別の有用度の平均値

No.	1	2	7	9	10	12	13	18	19	20	23	24	27
有用度	2.0	**3.0**	2.5	**3.0**	2.5	2.5	2.2	1.5	2.3	2.3	2.5	2.3	2.2
回答数	2	3	2	2	2	2	7	2	4	3	2	4	6

表 7　研究カテゴリ別の興味の度合いの平均値

No.	2	7	9	10	12	13	18	19	20	23	24	27
興味	**4.0**	**4.0**	**3.0**	**4.5**	**3.5**	2.7	1.5	**3.3**	2.7	**3.0**	**3.3**	2.7
回答数	2	2	2	2	2	7	2	4	3	2	4	6

得学位別に層別した結果を表 5 に示す．修士号取得者のほうが研究に対する興味が平均的に高かった．それぞれの研究論文に対する興味の強弱は，企業規模とは関係が弱く，取得学位との関連が強いと考えられる．これらの結果より，RQ4 に対する答えは「差異は生じない」となる．

カテゴリ別の，研究の有用度に対する評価の平均値を表 6 に，興味の度合いの平均値を表 7 に示す．データ件数が少ないため，ここでは層別せずに平均値を算出した．表では，回答が 2 件以上あったカテゴリのみを示し，平均値が 3 以上となったものを太字で示している．表中の No.は表 3 のものに対応する．個別に見た場合でも，研究の有用性の評価は全体的に低く，Performance (ソフトウェアの性能)と Synthesis (自動プログラム生成)のみ平均値が 3 以上となった．

興味の度合いについては，半数以上の研究カテゴリで平均値が 3 以上となっていた．ただし，Languages (ソフトウェア開発言語)に関しては，有用性，興味の度合いとも 2 を下回っており，全カテゴリで最も低い値となっていた．研究内容は C 言語に関するものが多かったが，回答者と C 言語の関連が弱かった可能性も考えられる．なお，カテゴリ別の有用度と興味の度合いについて相関係数を算出すると 0.47 となった．このことから，有用性と興味の度合いはある程度関連があると考えられるが，回答者は有用性と興味の度合いをある程度独立して評価していると考えられる．

これらの結果より，RQ3 に対する答えは「一部の研究カテゴリを除き，有用であるとはいえない」となる．

5. 研究カテゴリの推薦

いくつかの最先端研究は回答者（開発者）に興味を持たれていた．開発者が興味を持ちそうな研究カテゴリを，あらかじめ推薦システムによって提示することができれば，開発者は読む論文を絞り込むことができる．例えば，推薦システムはある開発者に対し，「業務と関連のある研究カテゴリは保守とテストである．そのうち，テストの研究に対して興味を持つと予測される」などと提示する．興味を持ちそうな研究を容易に知ることができれば，最先端研究を知ろうとする技術者も増え，そのような技術者の一部が研究者に対して有用性に対するフィードバックを行えば，それぞれの研究の有用性に対する再考察が進む可能性がある．

そこで本章では推薦システムによって，開発者が興味を持ちそうな研究カテゴリを，高い精度で推薦可能かどうかを実験により確かめた．同様の推薦システムは文献[14]でも提案しているが，文献[14]ではソフトウェア開発技術を推薦しているのに対し，本研究ではソフトウェア工学の研究を推薦している．そのため，必ずしも同様の精度が得られるかが明らかではないため実験により精度を確かめた．

実験では，業務に関連するカテゴリの推薦と，興味の度合いの予測を，協調フィルタリングを用いてそれぞれに行った．4.2 節で示したように，業務に関連するカテゴリは開発者によって異なるため，まず関連するかしないかを判別予測した．例えば「開発者 A の業務に関連する研究カテゴリは，保守，テストである」などと予測する．次に，興味の度合い（5 段階評価）の数値を予測した．例えば，「開発者 A は保守の研究に対する興味は低く，テストの研究に対する興味は高い」などと予測する．

上記の推薦における説明変数は，開発者のプロフィール，すなわち経験年数，関連する開発工程，所属する企業の規模とした．目的変数は関連の強い工程と，研究に対する興味の度合とした．関連する開発工程，所属する企業の規模についてはダミー変数化した．ダミー変数化とは，例えば要求工程が関連している開発者では 1，そうでない開発者では 0 となるような変数を設定することを指す．判別予測，数値予測の精度評価時にはリーブワンアウト法を適用した．リーブワンアウト法では，全データのうちの 1 件を予測対象とし，それ以外を過去（既知）データとすることを，繰り返して行う．

表 8　協調フィルタリングで用いるデータ

	p_1	p_2	…	p_i	t_k
u_1	v_{11}	v_{12}	…	v_{1j}	r_{1k}
u_2	v_{21}	v_{22}	…	v_{2j}	r_{2k}
…	…	…	…	…	…
u_i	v_{i1}	v_{i2}	…	v_{ij}	r_{ik}
…	…	…	…	…	…
u_m	v_{m1}	v_{m2}	…	v_{mj}	r_{mk}

5.1.　協調フィルタリングの概要

協調フィルタリングは，ユーザにとって好ましい，または役立つと考えられるアイテム（書籍，音楽など）を推薦するための手法として用いられている[4][10][13].「協調」とは，ユーザの知識を利用することを意味し，「フィルタリング」とは，大量のアイテムの中から，役立つアイテムだけを選び出して推薦することを意味する．一般的な協調フィルタリングで推薦を行う場合，各ユーザが各アイテムを（5 段階の数値などで）評価していることが前提となる（システムによっては，ユーザがそのアイテムを閲覧したかどうかを評価の代わりに用いることもある）．あるユーザが未評価のアイテムが，そのユーザにとって役立つと考えられる場合，そのアイテムを推薦する．

協調フィルタリングの主なアルゴリズムとして，ユーザベース手法とアイテムベース手法の 2 つがある．ユーザベース手法は，「アイテムの評価（好み）が似たユーザは，どのアイテムに対しても似た評価を行う」と仮定し，推薦を行う．具体的には，各ユーザの各アイテムに対する評価を要素とするベクトルを，ユーザごとに作成し，そのベクトルのなす角をユーザの類似度とする．そして推薦対象のユーザが未評価で，かつ類似したユーザの評価が高いアイテムを推薦する．ユーザベース手法を用いた推薦システムとして，Resnick ら[10]の GroupLens が挙げられる．GroupLens は，Usenet にある多数のニュース記事から，ユーザの好みに合うと予測される記事を選び出して推薦するシステムである．

アイテムベース手法は Sarwar ら[13]によって提案されたアルゴリズムであり，アイテム間の類似度に基づいて推薦を行う．アイテムベース手法の場合も，各ユーザの各アイテムに対する評価を要素としてベクトルを作成するが，ユーザごとにベクトルを作成するのではなく，アイテムごとに作成し，類似度を計算する．すなわち，「あるグループのユーザに高評価されるアイテムは，類似の性質を持っている」と仮定し，推薦対象のユーザが高い評価を行っているアイテムと類似度の高いアイテムを推薦する．

協調フィルタリングを用いた推薦では，表 8 のような $m \times n$ のマトリクス形式のデータを想定する．$u_i \in \{u_1, u_2, ..., u_m\}$ は i 番目の開発者を表し，$p_j \in \{p_1, p_2, ..., p_j\}$ は開発者のプロフィール（経験年数など）を表す．また，$t_k \in \{r_{1k}, r_{2k}, ..., r_{mk}\}$ はある研究カテゴリ k の評価結果を表す．

表 9　協調フィルタリングによる判別予測精度

	適合率	再現率	F1 値
ユーザベース	30%	24%	27%
アイテムベース	20%	22%	21%

表 10　協調フィルタリングによる数値予測精度

	絶対誤差平均値
ユーザベース	1.16
アイテムベース	1.23

5.2.　評価尺度

判別予測の評価尺度として，適合率，再現率，F1 値を用いた．これらは，推薦システムの評価尺度として広く用いられているものである[5]．適合率は，「業務に関連がある」と予測されたうち，実際に関連があった割合を示す．再現率は，全ての関連があった研究カテゴリのうち，「関連がある」と予測されたものの割合を指す．F1 値は，適合率と再現率のバランスを考慮して一つの指標としたものである．これらの値が大きいほど，判別予測の精度が高いことを示す．

数値予測の評価尺度には絶対誤差を用いた．絶対誤差は予測値と実測値（実際の評価値）との差の絶対値であり，値が小さいほど数値予測の精度が高いことを示す．

5.3.　予測結果

判別予測の結果を表 9 に示す．アイテムベースよりもユーザベースのほうの精度が高かったが，F1 値が 27%と非常に低い判別予測精度となった．数値予測の結果を表 10 に示す．こちらについてもユーザベースの予測精度がわずかではあるが高かった．ただし，絶対誤差の平均値が 1 を少し超えており，高い精度であるとまではいえなかった．

この結果より，業務との関連が強い研究カテゴリを判別予測することは容易ではないといえる．ただし，データの件数が増えれば予測精度が高まる可能性もある．実験で用いたデータにおいては，業務に関連する研究カテゴリの絞り込みについては，各開発者で行う必要があるといえる．それらに対する興味の度合いの高低については，ある程度は協調フィルタリングにより予測可能であるといえる．また，推薦に用いるア

ルゴリズムにはユーザベースが適しているといえる.

6. 考察

ソフトウェア工学は特に他の工学分野と比べて開発現場と密に関わっているため, 理論中心で現場への適用が想定されていないような研究は, あまり推奨されない. ICSE はソフトウェア工学の理論中心というよりも, 実験を考慮した研究が比較的多いとみなし, 本研究の実験題材として採用した.

研究論文のグルーピングを行う際には, 以下の 3 点をクリアできるよう考慮する必要がある.

- グループの定義が正しい
- 論文が所属するグループが正しい
- 論文数に偏りが出ない

研究論文を分類するために, 著者らで新たなグループを考慮することもできたが, 論文によってはどのカテゴリに何が属するのかの判断が難しく, 論文の数にも偏りが出る可能性があり, 上記 3 点が満たせない可能性があった. 各セッションに基づいて論文を分類しても特に違和感はなかったため, やや乱暴ではあるが会議のセッションを研究カテゴリとして用いた.

Zimmermann ら[1][8]の研究では, 「研究の方向性が正しいか」を聞いており, この観点からは各研究の評価は高かった. それに対し本研究では, 実際の業務でソフトウェア工学が有用であるかどうかが重要であると考え, 評価してもらった. 一部の回答者は業務経験年数が長かったが, 有用性の評価は低かった. このことから, 業務経験の短さが必ずしも有用性の評価につながっていないと考える. また, 研究内容に対する興味については低くなかったため, 回答者が内容を理解していないとはいえないと考える.

著者の一人の実務経験に基づくと, 実務経験が 1 年ほどあれば, 技術の有用性が理解できる程度の知識は習得しており, 少なくとも, 自分が担当する業務に関しては, 研究の有用性に対する判断はある程度正しく行えると考える. ただし, 経験年数が少ないと, 自分が担当している工程が少ないため, 担当した経験の少ない分野については有用性が低くなる可能性がある. 特に上流工程に関する研究の評価については, 経験年数のより長い被験者を対象に調査をする必要があると考える.

7. おわりに

本研究では, 最先端のソフトウェア工学の有用性に対するソフトウェア開発者の評価を, 開発者が所属する企業規模に着目して分析した. 企業に所属するソフトウェア開発者に対し, 業務に関係する研究カテゴリを最大で 5 つ選択してもらい, そのカテゴリに属する研究（1 カテゴリにつき 4 つ程度の研究）が自分の業務に有用かどうか, 興味を持ったかどうかを 5 段階で評価してもらった. その結果, 以下の傾向が見られた.

- 業務と関連が弱い研究カテゴリがいくつか存在した. 例えば Compilers and Emerging Trends (コンパイラ・最新トピック)や Energy and Videos (ソフトウェア省電力化・ビデオの活用)などのカテゴリは関連があるとの回答がなかった.
- 所属する企業の規模によって, 関連する研究カテゴリに差異が見られた. 例えば API や OSS は, 従業員数 300 人未満の企業に所属する開発者の業務と関連が強く, それ以外の関連が弱かった.
- Performance (ソフトウェアの性能)と Synthesis (自動プログラム生成)に関する研究を除き, 最先端の研究は業務に有用であるとはいえなかった.
- 開発者が所属する企業の規模によって, 研究の有用性及び興味の評価に差異は生じなかった. ただし, 興味については開発者の取得学位が影響している可能性がある.
- 業務に関連する研究カテゴリを予測した場合の精度が低かった. ただし, 研究に対する興味の度合いについては, ある程度の精度で予測可能であった.

今後さらに分析を行う必要があるが, 今回の分析では有用性の評価が高くなく, ソフトウェア工学の最先端研究が中小企業のニーズに応えられているとは必ずしもいえなかった. 今後の予定は, 回答者を増やすとともに, 入社年数の長い回答者を増やして分析結果の信頼性を高めることである.

謝辞 本研究の一部は, 文部科学省科学研究補助費（基盤 C : 課題番号 16K00113）による助成を受けた.

参考文献

[1] Carver, J., Dieste, O., Kraft, N., Lo, D., and Zimmermann, T.: How Practitioners Perceive the Relevance of ESEM Research, *In Proc. of International Symposium on Empirical Software Engineering and Measurement (ESEM)*, no.56, 10pages, 2016.

[2] 中小企業庁 : 中小企業の定義について, http://www.chusho.meti.go.jp/pamflet/g_book/h22/teigi.html.

[3] Estublier, J., Leblang, D., Hoek, A., Conradi, R., Clemm, G., Tichy, W., and Wiborg-Weber, W.:

Impact of software engineering research on the practice of software configuration management, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol.14, no.4, pp.383-430, 2015.

[4] Goldberg, D., Nichols, D., Oki, B., and Terry, D.: Using Collaborative Filtering to Weave an Information Tapestry, *Communications of the ACM*, vol.35, no.12, pp.61-70, 1992.

[5] Herlocker, J., Konstan, J., Terveen, L., and Riedl, J.: Evaluating collaborative filtering recommender systems, *ACM Transactions on Information Systems (TOIS)*, vol.22, No.1, pp.5-53, 2004.

[6] 情報処理学会 ソフトウェア工学研究会 国際的研究活動活性化 WG：ICSE2016 勉強会, https://sites.google.com/site/sereadings/icse2016.

[7] 経済産業省:平成 27 年特定サービス産業実態調査, 2016.

[8] Lo, D., Nagappan, N., and Zimmermann, T.: How practitioners perceive the relevance of software engineering research, In Proc. of Joint Meeting on Foundations of Software Engineering (ESEC/FSE), pp. 415-425, 2015.

[9] Osterweil, L., Ghezzi, C., Kramer, J., and Wolf A.: Determining the Impact of Software Engineering Research on Practice, *Computer*, vol.41, no.3, pp.39-49, 2008.

[10] Resnick, P. Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J.: GroupLens: An Open Architecture for Collaborative Filtering of Netnews, *In Proc. of Conference on Computer Supported Cooperative Work (CSCW)*, pp.175-186, 1994.

[11] Rubin, J., and Rinard, M.: The Challenges of Staying Together While Moving Fast: An Exploratory Study, *In Proc. of International Conference on Software Engineering (ICSE)*, pp.982-993, 2016.

[12] Ryder, B., Soffa, M., and Burnett, M.: The impact of software engineering research on modern programming languages, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol.14, no.4, pp.431-477, 2005.

[13] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J.: Item-Based Collaborative Filtering Recommendation Algorithms, *In Proc. of International World Wide Web Conference (WWW)*, pp.285-295, 2001.

[14] 秋永知宏, 大杉直樹, 柿元健, 角田雅照, 門田暁人, 松本健一:協調フィルタリングに基づくソフトウェア開発技術の推薦, 電子情報通信学会技術報告, ソフトウェアサイエンス研究会, SS2005-13, pp.7-13, June 2005.

実験参加協力のお願い

我々は, ソフトウェア開発企業で開発業務に従事したことのある方で実験参加の協力をしていただける方を募集しています. 詳細は以下 google フォームのサイトに記載しております. お問い合わせは, 本研究の著者までお願いします.

https://goo.gl/forms/n5UKrffJRAfkKFXz1

自分事化影響要因に着目した中期経営計画立案・展開への共創アプローチ[現状分析～計画立案編]

安達　賢二
株式会社 HBA
adachi@hba.co.jp

要旨

弊社のこれまでの経営計画立案・展開は、経営企画部門と役員クラスの事前調整に基づくトップダウンアプローチであった. その結果、実務層は実施すべき施策が指定された全社計画に合わせて部門・部署計画を立案、展開していた. 結果的にやらせる／やらされる運営が中心となり、躍動感やワクワク感のない、当事者意識・参画意識が薄い（あるいは、ない）、“こなす”仕事が多いと感じていた. また、全社レベルの計画・展開と実務レベルの実質的な実践内容が乖離している場合もあった.
2016 年に新社長が就任し、創立 50 周年(2014 年)に掲げた新テーマ“IT で幸せに挑む”の実践・実現に向け、社員一人ひとりが当事者となり事業を進めることで“仕事を通じた幸せ”を獲得してもらうため、人がものごとを自分事として継続的に実施することに影響を与える要因（自分事化影響要因、とする）に着目した社員参画型共創アプローチを採用し、中期経営計画立案を進めることとした. 現時点は中期経営計画が発行される直前であるため、まずは現状分析～計画立案に至る過程とその中で得られた効果、および今後の課題を明確にする.

1. はじめに

企業はその組織方針や中長期計画、および年度計画に基づき運営されるケースが多い. 弊社は 1964 年に創業（現在 52 周年）、社員数約 800 名、札幌に本社とデータセンター、北海道内 6 営業所、東京支社、関西にも拠点を置く（北海道では老舗の）SIer である. 弊社におけるこれまでの計画立案・展開は、経営企画部門と役員クラスの事前調整に基づくトップダウンアプローチが主流であった. その結果、実務層は「年度で実施すべき施策は与えられるもの」との認識下で展開された全社計画内容に依存し、それに合わせる内容で部門部署計画を立案、展開していた. 結果的にやらせる／やらされる運営となり、躍動感やワクワク感のない、当事者意識・参画意識が薄い（あるいは、ない）、どこか他人事のように“こなす”仕事が多かったと思われる. それでいて当該領域に対する社員の問題意識は薄く、慣習の恐ろしさをも感じてもいた.
2015 年度に就任した新社長の意向もあり、創立 50 周年に掲げた新テーマ“IT で幸せに挑む”の実践・実現に向け、社員一人ひとりが当事者となり事業を進めることで“仕事を通じた幸せ”を社員に獲得してもらう中期経営計画とすることを目指すことになった。
その命を受けた経営企画部門の主担当者が、これまで社内外問わず、関係者が自ら現状分析～改善計画立案とその実践を当事者として取り組む場を作りファシリテートするサービス（SaPID(*1)）を提供してきた筆者に（2016 年 5 月初旬）支援依頼してきたことから当取り組みが動き始めた.

2. 活動内容

2.1. 当初のアプローチ提案

中期経営計画とは、弊社の経営を 3 ヵ年単位で計画し、最終年度（3 年後）の目標達成に向けて施策を実践する基盤となるものである.
次期中期経営計画発行予定時期が 2016.12～2017.1 頃と期間的な余裕があること、実態に即した実効性のある計画にしたいこと、社員との直接的なコミュニケーションを重視する方針であった等の理由から、中期経営計画立案に対し、自分事化影響要因に着目した社員参画型共創アプローチを提案した.

社員参画型共創アプローチとは、社員一人ひとりが普段の実務での出来事や感じていることを自分の言葉で（付箋等に）記述し、その情報をきっかけにして主に以下の内容を明らかにすることを意図している.

(1)弊社らしさや実務で大事にしていること
(2)こういう会社になってほしいという将来像

(3)弊社の強み:他社より秀でた点、ウリ
(4)弊社の弱み:実存する課題や問題、困り事

(1)～(4) の情報を可能な限り社員から直接収集し、その結果を分析することにより、事実に基づく弊社の実情や社員の想い、そして社長を中心としたトップマネジメントの考えを融合させ、実効性ある(段階的な)施策を反映した中期経営計画を立案し、一人ひとりの社員が当事者意識を持ちながら実践することにつなげる方法として提案した.
このようにトップダウン偏重型のアプローチから脱却し、トップから実務を行う社員までが一体となり、共に新しい組織や事業を創りあげる、という意味から“共創アプローチ”と命名した.

2.2. 社員による関連情報収集と分析

それぞれの社員からの意見、コメント収集やその結果共有による Q&A の実施実績は以下の通りである.

表 1:社員の意見、コメント収集～結果共有活動実績

実施日 2016 年	実施内容	参画者数
5 月初旬	新入社員ワークショップ	25
5/18	新任課長代理ワークショップ	16
6/20	管理職選抜ワークショップ	17
7/8	新任主任ワークショップ	22
7/15	串刺懇談会ワークショップ	7
8/3	経営管理部門管理者ワークショップ	10
8 月末	経営管理部門ワークショップ	40
9/1	部門長ワークショップ	16
9/12-13	役員合宿ワークショップ(全員出席)	12
9/29	データセンター部門ワークショップ	15
11/25	技術部門管理者ワークショップ	11
11/26	公共部門管理者ワークショップ	15
12/6-7	技術部門意見交換会	15
12-1 末	社長による Town Meeting(計 5 回)	224
	計 445 名／800 名(56%) 課長以上の管理層参画率＝80%超	

社員からの意見、コメント収集の手段は、主にワークショップ形式を採用した。参加者に各 5 人前後のチームになってもらい、2.1.に示した(1)～(4)それぞれに対する各自の意見、コメントを付箋に記述し、その内容をきっかけにして相互に対話しながら本当に言いたいこと、伝えたいこと、背後に存在する事象や問題・課題などを明らかにする.

当初は筆者がファシリテータとなりワークショップを運営した. 抽象度が高い内容、事実かどうか不明な事項、真意が不明の表現などを“質問を契機にした対話”により掘り下げ、周囲のメンバー(他者)にも理解できる具体的な内容として書き換えて共有する.

写真 1:ワークショップの様子

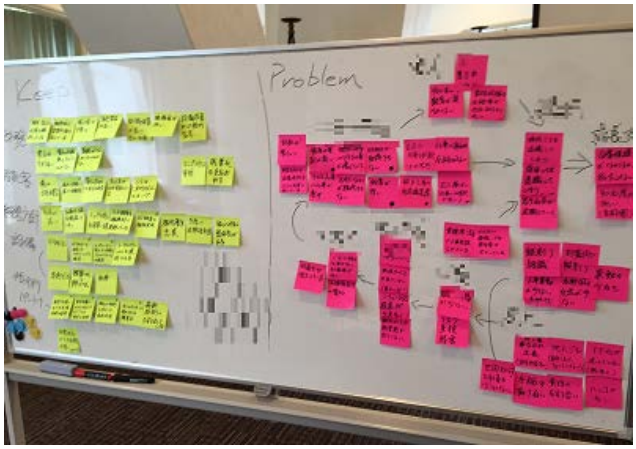
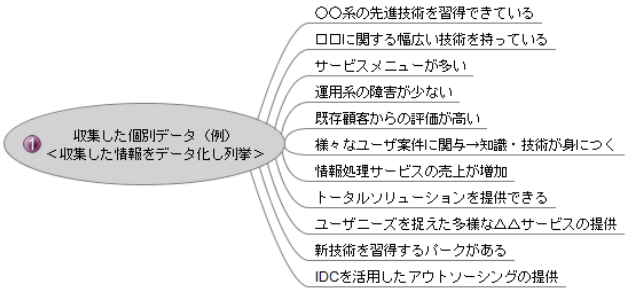


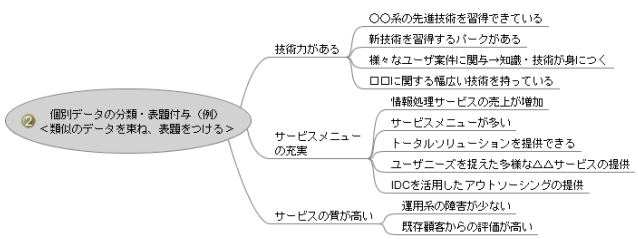
写真 2:ワークショップ中の分析過程

主な付箋(要素)の処理(洗練)が終わったところで、それぞれのチーム、もしくは参加者全員で収集(列挙)した個別要素を分類し束ねる、表題を付与する、因果関係が成り立つ要素間を矢印で結ぶ(原因→結果)ことによるモデ

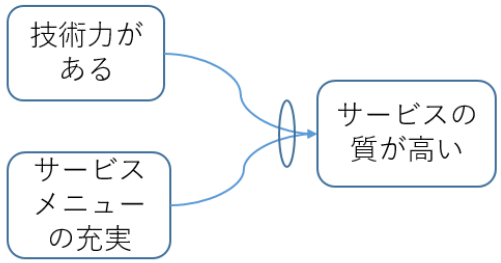
ル化・構造化を行う。これを関係性分析と呼ぶ.（図 1）



①収集した情報（要素）を列挙する



②類似の要素を束ね、表題をつける



③要素間の因果関係があれば→で連結し構造化する
（因果関係分析によるモデル化・構造化）

図 1:関係性分析の流れ(例)

分析対象は以下の 2 つである.

- (1)＋(2)から
  弊社の特長を統合した
  こういう会社にしたいよねモデル
- (3)＋(4)から
  わが社のよい点＋問題・課題構造図

ファシリテータは、要素の洗練や分析、モデル化・構造化に対しての答えを提供するのではなく、これらの進行促進と答えの出し方をガイドする.見落としや異なる見方などのヒントを提供することで、参加者自らが要素を洗練し、構造化した結果を導き出すことを促進する. 参加者自らが記述した要素を、自ら分析し、構造化することで、メンバー全員の総意である結果が導出されるため、参加者全員が腹落ちしやすく、自分事化しやすい結果を獲得することができる.

写真 3:ある部門のこういう会社にしたいよね要素群

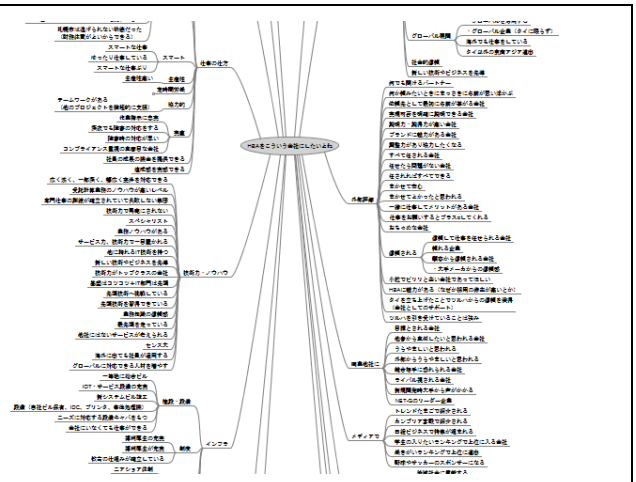
なお、筆者がファシリテータを自ら務めたのは当初から 2016/9/1 に実施した部門長ワークショップまである. それまでのワークショップにて進行方法やファシリテートの仕方を確立し、主催する経営企画部門に技術移転し終えた. 以降のワークショップ運営、ファシリテートは経営企画部門に委ね、筆者は次の段階となる収集済み全データの分析・構造化に軸足を移した.

2.3. 収集した全データによるモデル化・構造化

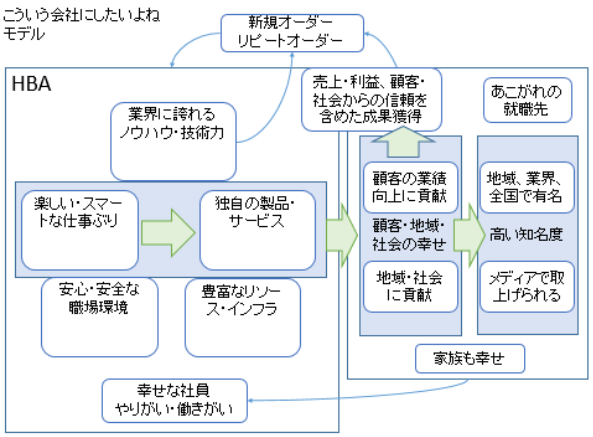
全社中期経営方針、および計画を立案するため、それぞれのワークショップにより収集した(1)～(4)の情報（要素）をすべて電子化し、関係性分析を行い、以下のモデル化・構造化を行った.（図 2）

- こういう会社にしたいよねモデル
- わが社のよい点＋問題・課題構造図

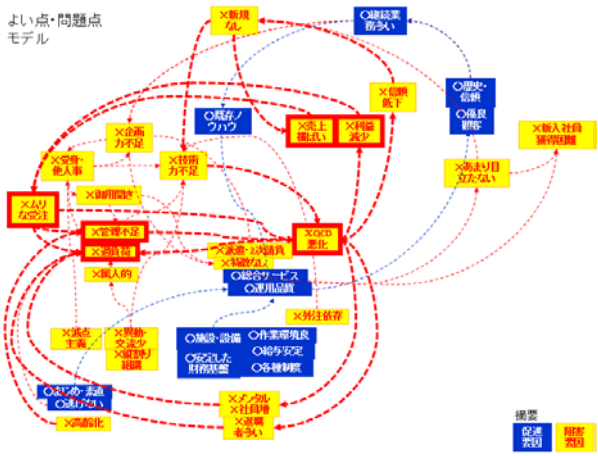
これまで実施してきた個別ワークショップでも同じモデル・構造図を構築しているが、それらは“参加者層”から見えている限定した情報に基づくものである. 情報の抜け漏れ、偏りが存在し、全社経営方針・計画立案の入力情報として相応しくない. よって中期経営計画立案のために全データによるモデル化・構造化が必要との判断した.



①収集した情報のデータ化＆分類（一部）



②会社の特長とこういう会社にしたい要素の因果関係分析から構築した「こういう会社にしたいよねモデル」



③わが社のよい点＋問題・課題要素の因果関係分析から構築した「よい点＋問題・課題構造図」

図 2:こういう会社にしたいよねモデル／
よい点＋問題・課題構造図の構築

実際には 2016 年 10 月中旬にそれまでに実施・収集した情報をデータ化し、分析・モデル化した上で、その後のワーク結果により都度更新することで対応した.

モデル化、構造化する理由・利点は以下のとおりである.

- 事業活動のよい循環・悪い循環が俯瞰できる
- 解決すべき問題・課題を特定しやすくなる
- 施策による効果目標を設定しやすくなる
- 施策による効果をシナリオ化でき、予測しやすくなる

2.4. 解決すべき問題・課題（候補）の定義

収集したさまざまな情報（要素）の洗練は（ワークショップの時間・工数的な制約から）ファシリテーション過程で検出した要素に絞って対応したため、分析結果と上記モデルには解決手段の裏返し事項などが残存していた.
例えば「～不足」（例：プロジェクトマネージャ不足）がそれにあたる. この情報は「～（例：プロジェクトマネージャ）を増やすと（ある問題が）解決する」という意図を持った情報であり、解決したい問題・課題ではなく解決手段の提案となっている. この情報をそのままにすると、本当に解決したい問題・課題に対する解決手段の他の選択肢を排除してしまい、実効性のある施策を導出しにくい状態となる.

よって、この段階で解決すべき問題・課題は何か？を再整理し、解決すべき問題・課題の候補として定義することとした.（例：ロスコンプロジェクト化の防止）

2.5. 中期経営計画のテーマの再確認

限られたリソースや制約条件の中で、最も有効な施策を実践するためには、沢山の解決すべき問題・課題の候補群の中から選りすぐった問題・課題を特定し、限りあるリソースを集中投下する必要がある.
この段階でその判断基準となる中期経営計画のテーマを今一度再確認した.

2.6. 社長主催タウンミーティングの実施

以上の情報収集、分析結果から社長、役員、および経営企画部門が中期経営計画の骨子を検討した. その結果を携えて社長自らがタウンミーティング（札幌・東京に

て計5回）を開催し、ワークショップに参加できなかった社員などを中心にこれまでの取り組みの経緯や情報収集・分析結果、今後の方向性（中期経営計画の考え方など）などを説明した.
また、説明後には参加者からの質問一つひとつに社長が自らの考えを伝達した.

2.7. 中期経営方針と中期経営計画の立案

以上のように収集した情報（要素）を分析・構造化し、解決すべき問題候補と中期経営計画のテーマに基づいて、解決すべき問題・課題を特定し、役員と経営企画部門が中期経営方針を策定、中期経営計画を立案した.

3. ワーク設計のポイント

当初の支援依頼を受けてワークショップをデザインした際に重視した主なポイントを記す.

3.1. 社員の参画・自分事化を促進

「実務の自分事化」「仕事を通じた幸せの獲得」「幸せな社員が顧客を幸せにする」を目指す上で、社員の参画は不可欠である. その意味で、アンケート収集などの間接的な手段ではなく、あえて直接対話の機会となる（社員間交流も促すことを目指して）ワークショップやワールドカフェ形式（Town Meetingで実施）を採用した.
自らの表現で記載した付箋や他者が記載した付箋をメンバーと相互に理解し、一緒に全体像を創りあげる.
構成要素、全体像の両面に納得し、合意を形成する.
以上の過程を踏むことで成果物の内容に対する自分事化を促進する.
また、可能な限り定例開催される昇格者向け社内研修の場を活用するなど社員への負担軽減に考慮した.

3.2. 安全な場の提供

テーマに対して参加者が忌憚のない意見、コメントが出せるように安心、安全な場を提供することが重要である. そのような場を提供するために実施した事項を列挙する.

・役員、管理者、実務者のワークの分離
・チームメンバーはあらかじめ運営側で設定することで、不仲、不毛な争い等が発生するリスクを最小化する
・情報収集・共有時の匿名性の確保
・ファシリテータ自らぶっちゃけ話を提供する
・感情的な内容であってもその背景を常に冷静に掘り下げ、真意を把握する

3.3. 事実情報の収集

実存しない情報を基に計画を立案するようなことはあってはならない. しかし、多くの社員から意見・コメントを収集すると事実かどうか把握できないもの、疑わしいものが混在していることが多い.
一方で、収集前に「事実でなければなりません」などの注意を述べてから収集を始めると、とたんに参加者の手が止まり、大事な情報も出てこなくなる傾向がある.

よって、意見・コメント収集時には“テーマ”以外の制約を可能な限り排除した上で記載してもらい（大事な情報を逃さないように）、その内容をヒントに掘り下げる（事実かどうかを把握して実存しない場合は排除する）対応を行った.
また、“問題・課題”という表現が硬い、難しいと受け取ると手が止まることを考慮して“普段の実務における困り事”を書いてもらうようにした.

3.4. 明るい未来とよい点から始める

意見・コメントを収集する順番は、以下のとおりである.

(1)弊社らしさや実務で大事にしていること
(2)こういう会社になってほしいという将来像
(3)弊社の強み：他社より秀でた点、ウリ
(4)弊社の弱み：実存する課題や問題点、困り事

計画立案などを目的とした現状把握情報として一般的に収集されるのは問題・課題関連情報である. 存在する問題や課題を捉えることでその対策としての計画を立案することが可能である. しかし、問題・課題情報から収集し始めると、参加者の顔が曇り、後ろ向きな場が出来やすい. それでは“共に創る：共創”は実現しにくくなってしまう.
よって、前向きで発展的な場を作り、新しい組織を一緒に創りあげる、参加者が前を向いて取り組んでもらえるように、(1)～(3)に関連する情報を先行して収集し、前向きな場を作り上げてから(4)を取り扱うこととした.

3.5. 自分事化に促すアプローチ

以上のワークは、自分事化影響要因を考慮した対策を、適切なタイミング、必要な箇所に配置し、構成した.

（表 2）
この対策群（アプローチ）により、ワークショップの参加者である社員が自ら現状を把握し、ありたい姿とのギャップを埋めるために必要な施策を自分事として実践することを促進する.

表 2: 自分事化影響要因とその対策

自分事化影響要因	今回の対策
自分が言ったこと、決めたことに責任を持ちやすい. 誰かが言ったことは自分事にしにくい.	・ワークショップにて自らの表現で書き出した要素を基に分析・構造化を実践してもらう.
自らのことを周囲に理解してもらいたいと（内心）思っていることが多い→理解してくれるとうれしい. 自分の困り事を解消するためなら積極的になりやすい.	・自分の困り事を書いてもらうところから始める. ・その内容と真意をチームメンバーがリアルに理解するワークを行い、その結果を分析・構造化に活用する.
一人だと継続できないことも仲間とならやれる.	ワークショップではチームとして取り組んでもらう.
価値を感じていれば、自分でもできる/できそうだと思うことには挑戦する.	・自分が考える“こうなりたい姿”を表明してもらうところから始める. ・自分の困り事を書いてもらう. ・なりたい姿と問題構造のギャップを埋めるために、チーム全員で必要な事項を決定（合意）する.
できなかったこと・わからなかったことができる・わかるようになると楽しい.	沢山の問題・課題のどれに取り組めば合理的、現実的なのかを構造化により明らかにする（自ら答えを導く）.
自らのニーズ・要求がよくわかっていない. 問題や要求ではなく（自分が知っている）手段を提示してしまう.	実務での困り事、感じていることを書いてもらい、その情報から（適切な質問により）背後に存在する問題・課題を明らかにする.
場に行動が左右されやすい.	安心、安全な場を提供し、(1)～(4)の情報を生の声として獲得する.

4. 当アプローチの効果

以上のアプローチにより、このあと 2017.4.1 からの 3 か年計画となる中期経営方針・計画を発行し、その実践に移行する. これまでの実践結果と現時点での効果を記載する.

4.1. 社員の積極的な参画を獲得

＜実施前の不安＞
ワークショップを実施しても冷めた態度を取る社員や管理者が多数存在することになるのではないか.

＜実施結果＞
いざワークショップを開催してみると、ほぼ例外なく参加者がみな積極的に参画し、実践していた. 特に意外だったのは、普段は強権発動的な行動・言動が目立つ社員や、一匹狼的なリーダが、ワーク時のチームメンバーと共にバランスのよい真っ当な意見・コメントを出しつつ分析や構造化も積極的に進めていたことである.
普段は内弁慶的に振舞っている社員であっても、同じ役割層等他者の目に晒されること、およびファイシリテータにより極論やバランスを欠くコメントには別の見方や事実確認を行うため、出過ぎた行動が融和・抑制された可能性がある. このアプローチには極論や自分勝手な筋違いな内容等が除去される効果があるのではないか. 一方で、（誰でも自分は正しいと思っているので）自分の意見・コメントが通らないことを心の奥底に溜め込んだ可能性もあるため、今後も継続観察していく必要がある。

4.2. 長所・強みが悪循環に加担していることに気づいた

＜実施前＞
事業、業務上の好循環、悪循環は、それぞれ長所・強み・よい点／短所・弱み・悪い点だけで構成されていると思っていた.

＜実施結果＞
今回は長所・強み・よい点／短所・弱み・悪い点すべての要素による関係性分析を行った結果（図 2 を参照）、弊社の誰もが認める長所・強みの一部が、事業の悪循環に加担している可能性が把握できた.（表 3）
このことは、今後弊社が目指す姿をあらためて考え直すきっかけとなった.

表 3:強みが悪循環要因へ加担する(例)

例: 強み「トータルサービスを提供している」 →解釈「何でも出来る」 →受取「ぼんやりしていて特長がない」 →結果「新規顧客獲得がなかなか進まない」 ＝悪循環要因の一つ

4.3. 社員の想いとトップの想いの融合を促進

＜実施前の不安＞
実務層の想いとトップの想いが折り合わない可能性が高いのではないか.

＜実施結果＞
実務を通じて社員が何を見ているのか、どう感じ、何を想っているのかを把握したうえで、トップの想いと融合することができた. これまでも現状評価に基づき計画を立案してきたが、社員の生の声を取り入れ、その背後にある事実を把握した上でトップの想いとつなげたのは初めてのことである.
今後「実務の自分事化」「仕事を通じた幸せの獲得」「幸せな社員が顧客を幸せにする」を目指す上で大事な階段を一つ昇ったと言えるのではないか.

このような結果となったのはこの取り組みが、「社員の(仕事を通じた)幸せ」と「顧客の幸せ」の実現を目指しているからではないかと思う.

4.4. 解決すべき問題・課題(候補)の明確化による実効性ある施策導出

＜実施前の状態＞
これまでの中期経営計画、年度事業計画で打ち出されてきた施策は、弊社の(そのときどきの)現状に対して間違いではないものの抽象的、一般的な内容であったり、表現は異なるものの何年経っても同類の内容が繰り返されることが多かった. そして実務層や管理層からの生の声を反映したとは言いがたい状態であったため、施策に対する社員の当事者意識は薄く、実務とは別物的に扱われている印象もあった.

＜実施結果＞
今回のアプローチでは、実務層・管理層から直接声が上がった内容を基に、解決すべき問題として捉えなおし定義した(→2.4.参照)候補群から、中期経営方針に沿って選りすぐった事項に対して施策を打ち出されている.
仮に、これまでと同じ(同類の)施策が存在していたとしても、自分事として捉えて取り組む可能性が高いと言える.
現在、各部門・部署の中期経営計画立案中であるため、今後の取り組みで自分事としての実践をさらに促進し、その成果を確認する必要がある.

5. 当アプローチの課題

当アプローチの主な課題とその対策を明示する.

課題 1:全社事業計画はどの部門にも適用可能な施策として提示しているため組織の状況や特長に適応できるアプローチを設計する必要がある.
↓
その対策
各部門による施策具体化検討開始時に、施策の目的、背景、現状、体制、制約条件などを共有し、効果的かつ現実的なアプローチを設計してもらう. そのうえで小さく試行した結果を活用してから部門内展開につなげるよう促す.

課題 2:人間の感情論をも取り扱うため、比較的高度なファシリテーション技術が必要になる.
↓
有識者が先行実施しながら、進行方法やファシリテートの仕方を確立し、ファシリテータ候補者に技術移転する. (→参照 2.2.)

課題 3:人の心を動かすのは簡単ではないため、継続して手間／時間をかけていく必要がある.
↓
その対策
即効性のある手法は存在しない. わかってもらえるまで粘り強く、一貫して対話を重ね、以降もさまざまな局面で繰り返しミッション・ビジョン・価値観を伝え、その実現を目指す実践を継続していく.

6. 今後の活動予定

2017年4月からいよいよ中期経営計画の実践に入る.
しかし市場環境の変化は激しいため、計画した施策を実践するだけではなく、併せて中期計画に対する四半期単位のふりかえりと計画見直しを実践していく.

また、今後の実践課程で社員が仕事を通じた幸せを実感できるようになったのかを 2011 年から隔年で実施している社員満足度（ES）調査の幸福 5 指標（心身の健康・経済的安定・良好な人間関係・仕事への情熱・周囲への貢献）などの指標を活用して評価していく予定である.

今回立ち上げた中期経営方針・計画はあくまでも弊社の全社員が一体になり“ITで幸せに挑む”＝「実務の自分事化」「仕事を通じた幸せの獲得」「幸せな社員が顧客を幸せにする」“を実現するための準備でしかない. 実現に向けた実践はこれからが本番である.
これからスタートする各種施策の実践においても、それぞれの社員が当事者意識を持って日々の業務に邁進し、仕事を通じた幸せを実感してもらえるように、そして変化が激しい市場環境に適応しながら弊社が目指す姿をより効果的に、効率的に実現できるように運営していきたい.

参考文献

[1] ソフトウェアプロセス改善カンファレンス 2012（SPI Japan 2012）システムズアプローチに基づくプロセス改善メソッド：SaPID が意図するコト（Systems analysis ／ Systems approach based Process Improvement methoD）～プロセスモデルをより有効活用するために／そして現場の自律改善運営を促進するために～
http://www.jaspic.org/event/2012/SPIJapan/session3A/3A4_ID023.pdf

[2] SaPID 実践事例より～改善推進役がやるべきこと／やってはいけないこと
現場が自らの一歩を踏み出すために
http://www.jaspic.org/event/2013/SPIJapan/session2B/2B3_ID011.pdf

コンテンツ開発プロジェクトとソフトウェア開発プロジェクトのマネージメント知見の類似性探究の試行報告

日下部　茂　　片平　梓　　青木　研
長崎県立大学　　長崎県立大学　　長崎県立大学

要旨

創造性にもとづいて知的作業成果物を作成するという観点から，ソフトウェアの開発と映像のようなコンテンツの開発には共通点があると考える．例えば，開発支援技術の進歩でルーチン作業の負荷は削減される傾向があると同時に，大規模複雑化が進み上流工程では明確な見通しを得にくいけれども，上流工程での欠陥は後工程で大きな問題となってしまう可能性が高い．本発表では，類似性がある領域のプロジェクトのマネージメントには類似性があるとの仮定の下，大学での教育研究活動においてコンテンツ開発プロジェクトにソフトウェア開発プロジェクト管理の知見を適用する試みと，その予備評価について述べる．

1. はじめに

著者らが所属する長崎県立大学の情報システム学部・情報システム学科は，平成 28 年度から改組新設された学科である．一般の情報工学系の学科と比べ，デザイン・コンテンツ系の教員の比率が高く，IT 分野における「アイデアを形にする能力」を重視し,「デザインもできる技術者」や「 技術もわかるデザイナー」の育成を目指している．創造性を発揮して知的作業成果物を作成するという点から，ソフトウェアの開発と映像を含む多様なメディアコンテンツの開発には共通点があると考え教育研究を進めている．

教育研究内容については，要素技術の知識やスキルだけでなくシステム開発やプロジェクトマネージメントの実践力も重視し，カリキュラム内だけでなく課外の活動も含めて継続的な改善に取り組んでいる．しかしながら，様々な制約から，すべてに関して，座学だけでなく実際の経験をもとにした段階的な実践力の養成をカリキュラム内だけで実施することは困難である．図 1 に情報システム学科のカリキュラムツリーを示す．例えばプロジェクトマネージメントに関して，座学が二年の後期，プロジェクト型のチーム型演習が三年開講されるものの，カリキュラム内で以前の経験の結果の振り返りを行いながらプロジェクトマネージメントの実践力を向上させるサイクルを複数実施することは難しい．

我々はこのような問題に対して，課外での開発・創作活動を積極的に支援すると同時に，その知見の蓄積や共有も推進している．本発表では，そのような取り組みの一つとして行っている，大学内でのコンテンツ開発プロジェクトにソフトウェア開発のプロジェクトマネージメントの知見を適用する試みとその予備評価について述べる．

本稿の構成は以下の通りである．第 2 節で，今回の取り組みで対象とした映像コンテンツ作成プロジェクトについて説明する．第 3 節で，適用を試みたソフトウェア開発のプロジェクトマネージメントの知見を説明する．第 4 節は，学生視点での振り返りを行う．また，第 5 節は，考察と今後の課題を述べる．第 6 節でまとめを行う．

2. 映像コンテンツ作成プロジェクト

本節では，ソフトウェア開発のプロジェクトマネージメントの知見を適用した，大学内でのコンテンツ開発プロジェクトについて説明する．

2.1. プロジェクトの目的

映像制作をおこなうには様々な技術やノウハウが必要とされ，そのすべてを横断的に網羅して一度に教育することは困難である．そこで，PBL(Project-Based Learning)

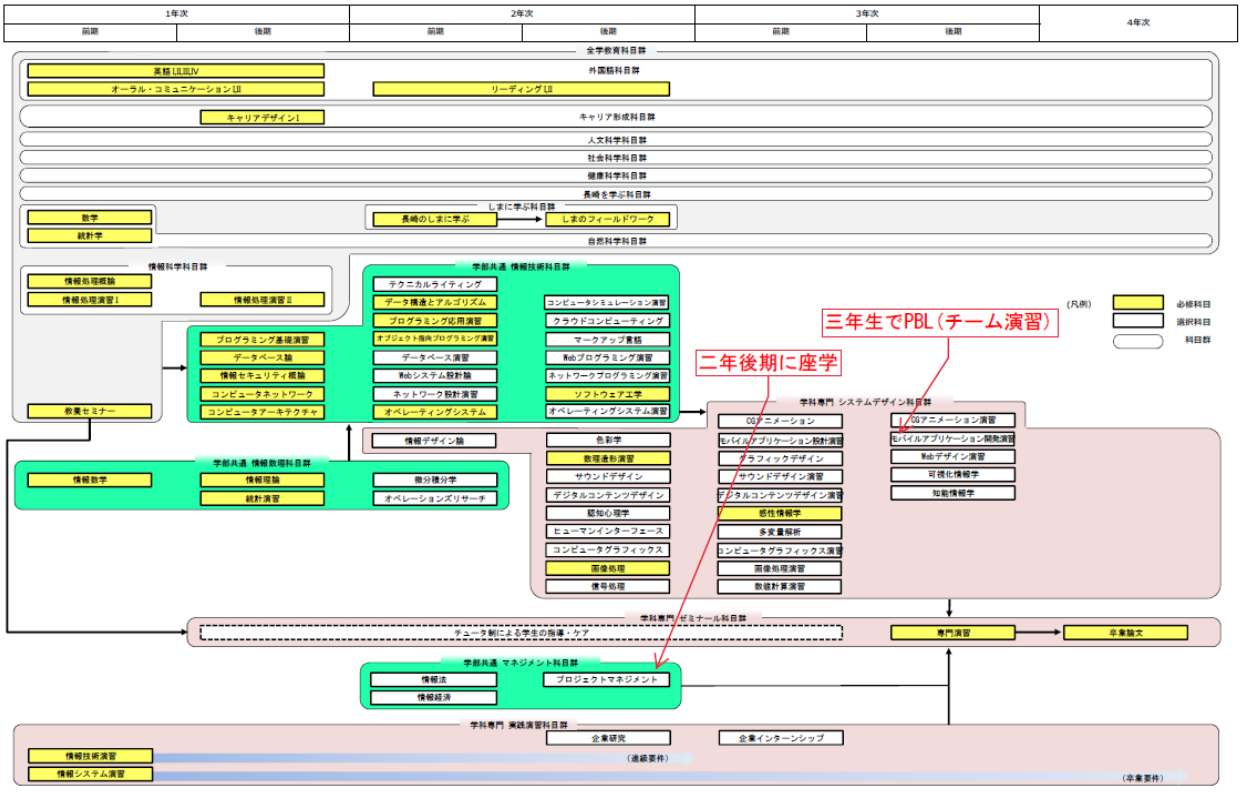


図 1. 情報システム学科のカリキュラムツリー

の手法を用い，短編映画製作を企画して，学生と教員，映像制作の専門家の共同作業による実践的な映像制作を試行する課外活動プロジェクトを実施している．その過程では以下を得ることを目指す．

- 参加学生に対する教育効果
- PBL による教育ノウハウの蓄積
- 教育のための教員の実践的技能の向上
- 公的なコンペティションによる作品評価と実績

教員や専門家との PBL 型の共同作業を通して学生が主体的に学び，コンペティションで採択されるレベルの作品作りを目指しながら，定型的な教育が難しい映像制作の技術とノウハウの模索と蓄積を試行する．

現在，映像系コンペティションは，海外においては戦略的な国家予算をつぎ込んだアジア諸国の大学の作品，国内においてはプロモーション目的で映像プロダクションが制作した作品といったものが，インターネットを通して広く応募されており，極めてレベルが高くなってきている．このような環境の中で，これらの作品に対抗しうる作品作りのノウハウを，学生自ら経験的に学んでいく教育方法の確立を目指す．

2.2. プロジェクトの概要

映像制作を後述のように，プリプロダクション（企画，脚本，ロケハン），プロダクション（撮影），ポストプロダクション（編集，整音，色補正，CG 等）のフェーズに分けて考えた上で，プロジェクトを実施する．さらに，用いる要素技術を段階的に高度化し，プロジェクトを段階的に高度化，複雑化していく．

プリプロダクション

- 短編映画作品企画：参加学生を募り，短編映画の作品企画を立てる．
- 脚本制作：企画に沿って脚本制作をおこなう．
- ロケハン 脚本推敲：脚本から撮影場所を決定する．

ロケハンによる撮影条件の検討から，脚本の推敲をおこなう．

プロダクション

- 出演者決定 リハーサル 美術製作 撮影準備 : 出演者を決定し，脚本の読み合わせなどリハーサルをおこなう．出演者には，一部プロを援用する．美術の設計と製作をおこなう．美術製作には一部プロを援用する．カメラテストや撮影許可の申請など，撮影準備をおこなう．
- 撮影 : 撮影をおこなう．撮影にあたっては，機材運搬車，出演者とスタッフの移動用のロケバス等の車両をレンタルする．照明には一部プロの援用と機材レンタルをおこなう．

ポストプロダクション

- 撮影後の編集，整音，色補正，CG 制作等をおこなう．
- コンペティション応募準備 上映会（公開講座）準備 : コンペティションのリサーチや英語字幕の制作など，コンペティション応募準備をおこなう．上映会（公開講座）用のプレゼン準備と，開催場所の確保，上映機材の手配をおこなう．
- コンペティション応募 上映会（公開講座）開催

3. 探究アプローチ

この節では，コンテンツ開発プロジェクトでも適用可能な，ソフトウェア開発プロジェクトのマネージメントの知見を探究する今回のアプローチについて説明する．

我々は，類似性がある領域ではプロジェクトの効果的なマネージメントにも類似性があるとの仮定にもとづいて取り組みを行っている．ソフトウェアの開発と映像のようなコンテンツの開発では，創造性にもとづいて知的作業成果物を作成するという点だけでなく様々な類似性があると考える．開発支援技術の進歩でルーチン作業の負荷は削減される傾向があると同時に，大規模複雑化が進み上流工程で明確な見通しを得にくい上に上流工程での欠陥は大きな問題となってしまう，といった点など共通点が多いと考える．ソフトウェア開発では，効率の向上や品質の向上には要素技術だけでなく，ソフトウェアの開発プロセスも含めたライフサイクルプロセスが重要とされ，上流工程が重要視され上流工程からモデルを活用することも普及している．同様に，映像分野でもVFX(ビジュアルエフェクト) を用いた工程の複雑化が進む一方，上流工程からプレビズを活用するといった上流重視の傾向が強くなっている．さらに，どちらも，プロセスを一旦構築すればそれで終わりということはなく，状況に応じてテーラリングや改善を継続的に行うことが重要という点でも共通点がある．

3.1. アプローチの概要

今回のアプローチでは，要素技術もマネージメント技術も段階的に高度化させていくという前提で，最初の段階では，プロジェクトで用いる要素技術も適用するマネージメントの知見も高度なバックグラウンドがなくとも取り組みやすいものを中心とした．まず，入門的な要素技術を用いて映像コンテンツを作成するプロジェクトを実施，その後プロジェクトマネージメントの担当者が，ソフトウェア開発プロジェクトのマネージメントの入門的な教材を読み，自身の体験の振り返りを行い適用可能な知見について考察した．

3.2. 教材の概要

今回は，入門的な教材として，”トム・デマルコの「プロジェクト管理」がわかる本”[1] を採用した．その書籍では，章の見出しレベルで，以下の内容が紹介されている [1]．

1. プロジェクト管理は心の管理
2. 生産性を向上させるオフィス環境
3. チーム力を上げる
4. プロジェクトを成功させる組織
5. デマルコのプロジェクト管理術
6. ソフトウェア開発におけるリスク管理
7. リスク管理の手順

プロジェクト管理については，プロジェクトマネジメント協会 (PMI: Project Management Institute) が発行している PMBOK ガイド (A Guide to the Project

[1] 節レベルの見出しは付録参照

Management Body of Knowledge)[2] をはじめとしてガイドや入門書が多数あるが，高度なバックグラウンドがなくとも読みやすいものを今回の教材とした．

4. 事後分析

この節では，学内プロジェクトのマネージメントの担当者による振り返りを紹介する．また，各大項目ごとに，共感を感じた小項目と，共感しなかった小項目をカウントした結果を示す．特に学生が共感したものについてはその内容を引用 (一部改編) して示した．

4.1. 「プロジェクト管理は心の管理」

「プロジェクト管理は心の管理」の 8 項目中，7 項目について共感し，1 項目は共感しなかった．

管理者が陥る七つの錯覚　「『気をつけたい 7 つの錯覚』のような状態に実際に陥った．進捗管理をまめに行い，各連絡はもっと確実に頻繁に取れ，もっと簡単により自動化したシステムでスケジュール管理や進捗管理が行え，多少厳しいことを言った方がいいはずだ，などをいつも考えていた．”部下を働かせるのではなく，働く気にさせる”とあったことに気づきを得た.」

品質第一主義は開発者の自尊心　「『マーケットの要求』に対するものとして，プロジェクトチームの実績をはやく作りたい，という自身の気持ちや，後に延ばせば延ばすだけ結局雑になるのでは，という疑問から，品質やメンバーの意向を無視して『自尊心』を満たせていないと感じた．作品でも，結果として詰め込んだスケジュールでの撮影で疲労や焦りで後半はあまりいい演出を行えなかった.」

生産性向上に関する法則とリスク管理　「リスク管理が足りてないということは痛感した．無駄な時間が多い．しかし，映画制作から無駄をなくしていいのだろうかとの懸念もある．楽しんでやること，遊び心が，より良い作品を生み出すとも考えられるので，バランスを意識しなくてはいけないと考える.」

4.2. 生産性を向上させるオフィス環境

「生産性を向上させるオフィス環境」については，4 項目中，3 項目について共感し，1 項目は共感しなかった．

職場を楽しいところにするためには　「職場に小さな混乱を導入」ということで，コンテストの開催やブレインストーミングが提唱されているが，これは積極的に行っていきたい．12 月に開催したクリスマス会は，そのいい例だと感じた．そのときに行った「青春選手権」という企画は，元々はまだあまり話したことないメンバー同士や，初めて参加したメンバーとの仲を深めることが目的だったが，私自身を含め，とても刺激になった．また，プロジェクトチームの今後の方針や動きについてのブレインストーミングを行ってみたいと思った．コンテスト，というほどのことではないが，脚本，編集，撮影，演出などで競うということも今後やっていきたい.」

4.3. チーム力を上げる

「チーム力を上げる」に関しては，10 項目中，3 項目について共感し，4 項目は共感しなかった．3 項目についてはどちらでもなかった．

4.4. プロジェクトを成功させる組織

「プロジェクトを成功させる組織」に関しては，6 項目中，2 項目について共感し，2 項目は共感しなかった．2 項目についてはどちらでもなかった．

伝染する管理者の怒り　「怒りは恐怖の別の表現，ということであるが，確かに私自身に当てはまる．このまま完成できないのではないか，今更このような変更をしてうまくいくのだろうか，今の段階でこの話合いをして撮影までに間に合うのだろうが，というような不安からついイライラすることは頻繁にある．実際にその場で怒りを表現したことはあまりないが，恐らくメンバーは察知して気を遣ってくれていると思う．私が感情的になっていては，円滑に動かないことも考えられるので，むしろ私がメンバーの心情を汲んで話合い等を進めていけたらと思う.」

4.5. デマルコのプロジェクト管理術

「デマルコのプロジェクト管理術」に関しては，10 項目中，1 項目について共感し，7 項目は共感しなかった．2 項目についてはどちらでもなかった．

計画の変更を恐れてはならない　「今回の撮影でもあったが，撮影の日になって突然脚本の内容や演出を変更することがある．もちろん，必要なことであり，よりよい作品とするため，その変更についてはむしろプラス方向で捉えている．しかし，その変更により，撮影時間が伸び，スケジュール通りに動かない．そこで，撮影も後日，話合いも後日，と，日程が延期されていく．自分自身の，実績を得なければ，もっと作品を作らなければ，という焦りも相まって，延期されていく（予定のスケジュールを変更する）ことにとても抵抗がある．抵抗はあるが，仕方ない，私の計画ミスだ，と考えてきたが，この節を読んで腑に落ちた．実際，撮影のスケジュールは，限られた時間で撮影を終えるため，詰め込んだスケジュールになっていた．疲労や空腹から，全体の空気も悪くなっていた．私のミスだ，と自分を責めることは簡単だが，それは責任と向き合っていない．より柔軟な対応を心掛けたい.」

プロジェクト初期は少人数で　「設計段階では少数精鋭で会議を行う，ということは取り入れてみたい．各部での責任者が全体の現状を把握できてないことで，部と部での連携が取れていない．どの段階で少数でのミーティングを行うかは考える必要があるが，作品の初期段階で行いたい.」

無駄な会議の減らし方　「よく『参加できなくてごめん』とメンバーから言われる．プロジェクトチームの方針として，来られるとき，来られるメンバーが来る，ということになっている．しかし，参加してもらわないと困る人ももちろんいる．その一方，実際，無駄だと思うミーティングの時間もある.「次のミーティングで何をする」という宣言により，誰が来なくても大丈夫か，ということをはっきりさせたい．参加できないことを申し訳なく思わせることを減らすことで，より気軽な集まりにできたらいい.」

4.6. ソフトウェア開発におけるリスク管理

「ソフトウェア開発におけるリスク管理」に関しては，12 項目中，共感する項目はなく，8 項目は共感しなかった．4 項目についてはどちらでもなかった．この結果は，ソフトウェア固有の事項が該当しないといった，領域の違いによるものなのか，まだリスク管理を考える域まで達していないためなのか，といったいくつかの理由が考えられるため，今後より詳細な分析を続ける予定である．

「大人」のプロジェクト管理　「リスクについては本当に管理できないと感じている．起きてほしくないことを放置，または，なんとかなる，で片づけている．書き出してまとめて，対策を考えることも行いたい.」

4.7. リスク管理の手順

「リスク管理の手順」に関しては，共感する項目はなかった．この結果に関しては，領域の違いによるものなのか，まだリスク管理を具体的に考える域まで達していないためなのか，今後より詳細な分析を続ける予定である．

5. 考察と今後の課題

ソフトウェア開発の領域では，例えば CMMI(Capability Maturity Model Integration)[3][2] のようなプロセスモデルや TSP(Team Software Process)/PSP(Personal Software Process)[4] [5] [3] のようなプロセステンプレートが提案され，実際に活用もされている．当初はそのようなものを利用してコンテンツ系のプロジェクトの教育を実施できないかとの検討も行った．しかしながら，ソフトウェアの開発とコンテンツ系の開発では考慮すべきことは必ずしも同じではない一方，コンテンツ領域において今回の趣旨にそった形で利用できるテンプレートや参照モデルを見つけることができなかった．

また，仮にそのようなものがあったとしても，プロジェクト管理やプロセスの教育は，学生自らがある程度の開発経験を経てその必要性を感じてからでないと，効果的に実施することが困難である可能性がある．著者の

[2]CMM & CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

[3]TSP & PSP are service marks of Carnegie Mellon University.

一人は，個人レベルのソフトウェア開発 PSP(Personal Software Process) のインストラクタとして大学でそのトレーニングコースを講義・演習として行った経験がある．PSP は，CMM(Capability Maturity Model, CMMI の前身) のレベル 5 相当の組織でのエンジニアを想定して提案されており [6]，学生にとっても有用でないかと考えた．しかしながら，実際には該当科目を選択する学生数も多くなく，トレーニングコースの完了率も高くなかった [4]．そういった経験もあり，モデルやテンプレートがあったとしても，学生がもつ知識やスキルのレベル，プロジェクトでの学生の役割やプロジェクトの特性などに応じて，プロジェクト遂行時に学生が実際にどのようなことに必要性や重要性を感じるのか，といった観点で知見を集めることは重要と考えている．

学生が納得しやすいという観点で，4 節のような振り返りは有用である一方，系統だった整理・分類という観点では十分ではない．最終的には，多数の初学者を念頭に置いた学内の教育研究だけでなく，一定の経験をもつソフトウェア開発者やコンテンツ開発者の双方にとって有用な一般化された知見の構築も目指すしている．そのような観点では，ソフトウェア開発領域で一定の評価を得ている CMMI-DEV(開発のための CMMI) といった参照モデルや PSP のようなテンプレートのフレームワークは有効と考えている．

PSP では図 2 に示すようなベースのプロセスフローと，プロセス要素が段階的に高度化されたプロセスのテンプレート 3 が定められている．また，CMMI では表 1 のように複数のプロセス領域が系統的にモデル化されている．このようなソフトウェア開発領域で一定の評価を得ているものを参考にすることで，ソフトウェア開発とコンテンツ開発での類似点や相違点も含め，双方の領域の開発者にとって有用な知見を取りとめることができる考えている．

このような取り組みが成功すれば，ソフトウェア開発において例えば UX(ユーザーエクスペリエンス) が重視される場合はコンテンツ開発の知見が有用になり，コンテンツ開発において例えばデジタル化が進展するとソフトウェア開発の知見の有用になる，といったことが予想される場合に，どのようなプロセス領域を重視し，どのようにチームや個人レベルのプラクティスに反映すればよいか系統的に取り組めると考えている．

[4] このような傾向は，PSP を採用している他大学でも類似の傾向があると聞いている．

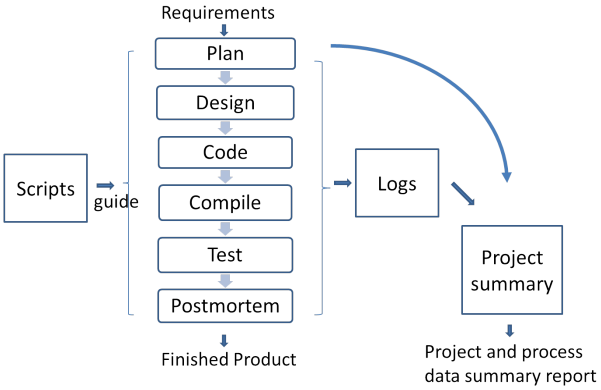


図 2. PSP のプロセスフロー

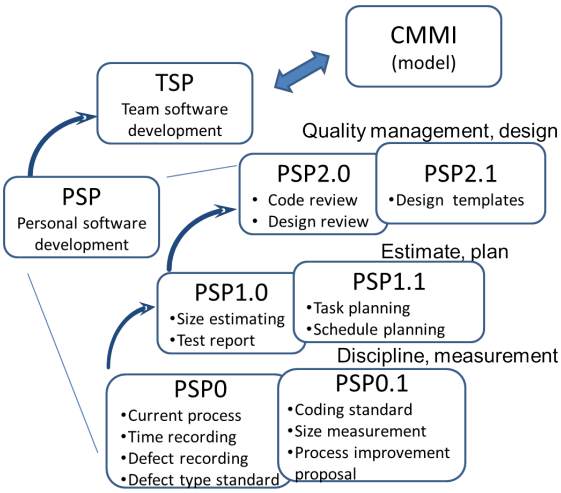


図 3. PSP の段階的高度化

6. おわりに

創造性にもとづいて知的作業成果物を作成するという観点から，ソフトウェアの開発と映像のようなメディアコンテンツの開発には共通点があると考え，大学内でのコンテンツ開発プロジェクトにソフトウェア開発のプロジェクトマネージメントの知見を適用する試行とその予備評価について述べた．ソフトウェア開発のプロジェクトマネージメントの知見であっても，おおむね，領域に非依存ともいえるものは，メディアコンテンツの開発プロジェクトのマネージメントにも適用できる可能性が高いという評価を得た．しかしながら，ソフトウェア固有の事項に関連付けられていたり，種々の想定を必要とす

表 1. 開発のための能力成熟度モデル統合 (CMMI-DEV) のプロセス領域

段階表現での成熟度レベルとプロセス領域	連続表現での区分
2: 要件管理 (REQM: Requirements Management)	プロジェクト管理
2: プロジェクト計画策定 (PP: Project Planning)	プロジェクト管理
2: プロジェクトの監視と制御 (PMC: Project Monitoring and Control)	プロジェクト管理
2: 供給者合意管理 (SAM: Supplier Agreement Management)	プロジェクト管理
2: 測定と分析 (MA: Measurement and Analysis)	支援
2: プロセスと成果物の品質保証 (PPQA: Process and Product Quality Assurance)	支援
2: 構成管理 (CM: Configuration Management)	支援
3: 要件開発 (RD: Requirements Development)	エンジニアリング
3: 技術解 (TS: Technical Solution)	エンジニアリング
3: 成果物統合 (PI: Product Integration)	エンジニアリング
3: 検証 (VER: Verification)	エンジニアリング
3: 妥当性確認 (VAL: Validation)	エンジニアリング
3: 組織プロセス重視 (OPF: Organizational Process Focus)	プロセス管理
3: 組織プロセス定義 (OPD: Organizational Process Definition)	プロセス管理
3: 組織トレーニング (OT: Organizational Training)	プロセス管理
3: 統合プロジェクト管理 (IPM: Integrated Project Management)	プロジェクト管理
3: リスク管理 (RSKM: Risk Management)	プロジェクト管理
3: 決定分析と解決 (DAR: Decision Analysis and Resolution)	支援
4: 組織プロセス実績 (OPP: Organizational Process Performance)	プロセス管理
4: 定量的プロジェクト管理 (QPM: Quantitative Project Management)	プロジェクト管理
5: 組織実績管理 (OPM: Organizational Performance Management)	プロセス管理
5: 原因分析と解決 (CAR: Causal Analysis and Resolution)	支援

るリスク管理に関する知見は，今後の取り組みでより詳細な分析を行う必要があると考える．

どちらの領域の開発でも，今後も引き続き支援技術が進化しルーチンワークは省力されると同時に，より一層大規模複雑化が進み，プロジェクトマネージメントの重要性も増すためこのような取り組みの重要性も増すと考えている．また，外部環境の変化，要素技術の進展，開発プロセスの上流重視の傾向などもあり，一定の知見が確立したとしても継続的な見直しが必要と考えている．

今後は，カリキュラム内外でのプロジェクト経験,「企業研究」や「インターンシップ」といった科目での実務者との交流や現場体験からのフィードバックなども通して，蓄積，共有される知見の高度化を目指す予定である．

参考文献

[1] 吉平健治，トム・デマルコの「プロジェクト管理」がわかる本，秀和システム，2007

[2] Project Management Institute, PMBOK (Project Management Body of Knowledge) Guide, http://www.pmi.org/pmbok-guide-standards/foundational/pmbok (2017 年 5 月 19 日参照)

[3] CMMI, http://cmmiinstitute.com/ (2017 年 5 月 19 日参照)

[4] Team Software Process, http://www.sei.cmu.edu/tsp/ (2017 年 5 月 19 日参照)

[5] Watts S. Humphrey, A Self-improvement Process For Software Engineers, Addison-Wesley Pub, 2005.

[6] Watts S. Humphrey, Software Process Improvement A Personal View: How it Started and Where it is Going, Softw. Process Improve. Pract. 12: 223-227, Wiley InterScience, 2007.

付録 :”トム・デマルコの「プロジェクト管理」がわかる本 ”の知見 (大項目と小項目)

1. プロジェクト管理は心の管理
   (a) プロジェクト失敗の本当の原因は?
   (b) 管理者が陥る七つの錯覚
   (c) 「管理ごっこ」をしていませんか?
   (d) 品質第一主義は開発者の自尊心
   (e) パーキンソンの法則は当てはまらない
   (f) プログラムは夜できあがる

(g) 生産性向上に関する法則とリスク管理
(h) プレッシャーと生産性の関係

2. 生産性を向上させるオフィス環境

(a) 仕事に集中できるオフィスとは
(b) オフィス環境進化論
(c) 職場を楽しいところにするためには

3. チーム力を上げる

(a) 最高の人材を揃える
(b) 退職にまつわる無駄なコスト
(c) 結束したチームがもたらす利益
(d) 自己修復システムを使う
(e) 結束させるためのチームの潰し方
(f) チームを形成するための不思議な作用
(g) 部下，チームメンバを好きになれ
(h) 体を使った採用術

4. プロジェクトを成功させる組織

(a) 脅迫は効かない
(b) 匿名性を使い「悪い情報」を伝達
(c) 病んだ政治は避けられない
(d) 伝染する管理者の怒り
(e) 対立が仕様書をあいまいにする
(f) 対立は仲裁によって解決する

5. デマルコのプロジェクト管理術

(a) 失敗プロジェクトを打ち切る勇気が必要
(b) 管理者の直観をモデル化する
(c) 計画の変更を恐れてはならない
(d) ソフトウェアのサイズを測定する
(e) 開発プロセスに対しての考察
(f) デバッグ時間を削減するには?
(g) 間違いを知らないのは一番怖い
(h) プロジェクト初期は少人数で
(i) 無駄な会議の減らし方

6. ソフトウェア開発におけるリスク管理

(a) リスクのないプロジェクトには手を付けるな
(b) 「大人」のプロジェクト管理
(c) リスク管理と危機管理
(d) リスク管理を行う理由
(e) リスクを無視すると
(f) 不確定性を数量化する
(g) 不確定な日程の見積もり
(h) リスク図による日程の解釈
(i) 不確かなものへの不安

7. リスク管理の手順

(a) リスク管理の五つのステップ
(b) リスク発見の方法
(c) リスクを隠してしまう会社文化
(d) 破滅シナリオの提案
(e) 見えないリスクを発見する
(f) ソフトウェア開発でのリスク
(g) リスク・エクスポージャー
(h) インクリメンタル手法によるリスク軽減
(i) EVR による進捗管理
(j) 感度分析
(k) 臆病なマネージメント

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
SCSK 株式会社
Shuuji.Itou@scsk.jp

山口 真
SCSK 株式会社
Makoto.Yamaguchi@scsk.jp

豊田 圭一郎
SCSK 株式会社
Keiichirou.Toyoda@scsk.jp

要旨

1. はじめに

基幹系システムを保守・運用する組織やチームは,そのシステムが稼働し続ける限り,もしくは契約が続く限り,一定の経験を有するメンバーを中核に据えた体制を維持する必要がある.今回の事例報告では,機能期の段階に突入した組織が,メンバーを入れ替えずに更に成長していくための手法として,「参加型勉強会」を適用した事例を紹介する.

2. 事例概要

(1) 「参加型勉強会」について

座学によるスキルや知識の習得を目的とした「通常の勉強会」とは異なり,グループワーク中心でアウトプット重視の勉強会である.この会の共通キーワードは「正解はありません」.物事の捉え方や考え方をバージョンアップする気づきやヒントを得るための場で,参加者全員が主役になることができるのがポイントである.

(2)「参加型勉強会」で工夫した点

運営にあたり,以下の 3 つの点を工夫した

①こまめにふりかえりを実施

勉強会後やプロジェクトの節目に,KPT でふりかえりを実施し,各自の考え方の変化を可視化した.回を重ねるごとに学びの意識の高まりや前向きな意見が増えてきた.

②報告レポートで共有

毎回,報告レポートを作成して勉強会に参加できなかったメンバーとも内容を共有し,次回参加のきっかけとした.

③ランチ開催

多様な参加者を募るためランチ形式での開催を優先.業務後は参加が厳しいメンバーも参加しやすくなり,発言や考え方に多様性が生まれた.

(3)「参加型勉強会」の成果

この活動の大きな成果は 4 つある

①多様なメンバーの参加

多様なメンバーで議論することで,メンバーの考え方や発言に新たな可能性がみえた.

②現場のコミュニケーションが活性化

勉強会の中でチームの強みと実例を共有することで,チームの枠組みを超えた会話や連携が生まれた.

③積極的な発言を引き出す

全体会議で対立を恐れない積極的な発言が増えた.

④新しい知識やスキルへの意欲

旬のテーマや普段使わない手法を体験することで,組織やチームの外側にも目が向くようになった.結果,現場を改善しようとするや行動が増えた.

(4)今後の課題

今後の課題として考えているのは 3 つ

①全体とチームで中長期ビジョンと戦略の向き先を合わせる

②プロジェクト内で柔軟な配置転換と育成

③コミュニケーション量と質の可視化

組織成長モデルとして,今回示した勉強会の効果はごく一部分である.今後も活動を継続することで,有益な成果,知見として発表できるように努めたい.

参考文献

[1] 翔泳社 Project Management Professional 第 5 版 P530-P534 （ISBN978-4-7981-3729-2）
[2] チームが成長し続ける「ラーニングサイクル」 IT 現場を強くする 究極のチームビルディング P18-P19 斉藤 秀樹著 日経 BP 社 （ISBN978-4-8222-7186-2）
[3] すばる舎リンケージ これだけ！,KPT P103-P106 （ISBN978-4-7991-0275-6）

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

久木田 雄亮
和歌山大学大学院 システム工学研究科
kukita.yusuke@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

本稿では，コードクローン追跡ツール *CCT (Code Clone Tracer)* を用いて，コードクローンの変更に関与した開発者らとソフトウェア品質との関係について分析する．*CCT* は，コードクローンの作成・利用過程における人的影響を調査することを意図して設計されたコードクローン追跡ツールである．分散型バージョン管理システム *Git* を対象としたコードクローン追跡機能に加え，不具合管理システムと連携することで不具合を混入したコミットとその作成者を特定し分析するための機能を備えている．本稿では，*3* つのオープンソース開発プロジェクト（*RxJava, c:geo, Jackson databind*）を対象として行った社会ネットワーク分析の結果について報告する．

1. はじめに

長きに渡るソフトウェアシステムの改良・保守の過程では，生産性の向上を目的として既存コードが頻繁に再利用されることがある．その結果,「同一の」または「類似する」コード片（本稿ではコードクローンあるいは単にクローンと呼ぶ）がソースコード全体に遍在することも少なくない．ソースコード全体にコードクローンが多数存在する状況においてコードクローンに変更を加える必要性が生じた場合，変更箇所が多岐に渡ることから多くコストが必要となったり，変更漏れにより新たな不具合を混入してしまう恐れがある．すなわち，改良・保守の初期段階では有益な手段であった既存コードの再利用が，結果的にその後の生産性や品質を低下させる原因となる可能性がある．

このような懸念を払拭するために，コードクローンの**検出手法** [1, 5, 10] や**分析手法** [6, 7, 9] がこれまで盛んに研究されてきた．これら既存研究の多くは主に，ある時点のソフトウェアシステムに含まれるコードクローンとソフトウェアシステムの品質，あるいは，コードクローンの変更がソフトウェアシステムの品質に与える影響に着目している．コードクローンの存在により，あるソースコードに対する変更が他のソースコードに対してどのように影響するかを把握することが困難になるため，結果として品質が低下する可能性が高いと考えられていたためである．しかしながら，これまでの膨大な量の研究を通じて，コードクローンの存在と品質に明確な関係が認められる事例とそうでない事例が混在して報告されており，未だ統一的なコンセンサスは得られていない [15]．

そのため，特に近年では，ソフトウェアシステムの改良・保守の過程でコードクローンがどのように拡散していくのかを詳細に分析するための**クローン追跡**手法 [4, 8, 14] に注目が集まっている．ソフトウェアシステムの開発において広く利用されているバージョン管理システムに記録された全変更履歴に基づいて，リビジョンを 1 つずつ辿りながらコードクローンの変更により不具合が混入される過程を正しく理解し，不具合の混入を未然に防ぐための対策につながると期待されている．

本稿では，我々[12, 16] が提案したコードクローン追跡ツール CCT (Code Clone Tracer) を用いて，コードクローンの変更に関与した開発者らとソフトウェア品質との関係について分析を行い，プロジェクトにおいてコードクローンを作成することは悪影響を与えるものであるかをコードクローンに関係する Issue や開発者間の繋がりから確認する．CCT は，コードクローンの作成・利用過程における人的影響を調査することを意図して設計さ

れたコードクローン追跡ツールである．分散型バージョン管理システム Git を対象としたコードクローン追跡機能に加え，不具合管理システムと連携することで不具合を混入したコミットとその作成者を特定し分析するための機能を備えている．本稿では，3 つのオープンソース開発プロジェクト (RxJava, c:geo, Jackson databind) を対象として行った社会ネットワーク分析の結果について報告する．

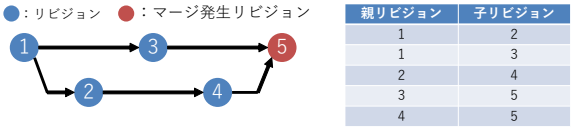
本論文の構成は以下の通りである．続く 2 章では，まず，コードクローン追跡ツール CCT (Code Clone Tracer) の実装について詳述する．3 章では，CCT の処理結果を用いて，コードクローン変更過程における開発者のインタラクションとソフトウェア品質との関係を調査するための分析方法について述べる．4 章では，3 つのオープンソース開発プロジェクト (RxJava, c:geo, Jackson databind) を対象として行った社会ネットワーク分析の結果を示し，分析結果を考察する．最後に 5 章において，本稿をまとめるとともに本研究の今後の課題を示す．

2　コードクローン追跡ツール CCT

本章では，コードクローンの作成・利用過程における人的影響を調査するため実装したコードクローン追跡ツール CCT (Code Clone Tracer) について述べる．CCT は，数千から数万リビジョンからなるソフトウェアシステムに存在するコードクローンの追跡を想定しており，高速にクローンを特定することができる粗粒度なクローン特定方法 [13] を用いてクローンを追跡する [1]．また，不具合管理システムと連携することで不具合を混入したコミットとその作成者を特定する機能を備えている．以降では，コードクローン追跡機能とコードクローン分析機能のそれぞれについて詳細に説明する．

2.1　コードクローン追跡機能

分散バージョン管理システム Git を対象としたコードクローン追跡を実現するために，リポジトリに保存されているすべてのリビジョンを対象に，(1) リビジョンの親子関係の特定，(2) 各リビジョンに含まれる全コード片の特定，(3) コードクローンの特定，(4) コードクローンの追跡の 4 種類の処理を順に行う．以下ではそれぞれの処理について説明する．



親リビジョン	子リビジョン
1	2
1	3
2	4
3	5
4	5

図 1: リビジョンの親子関係の特定

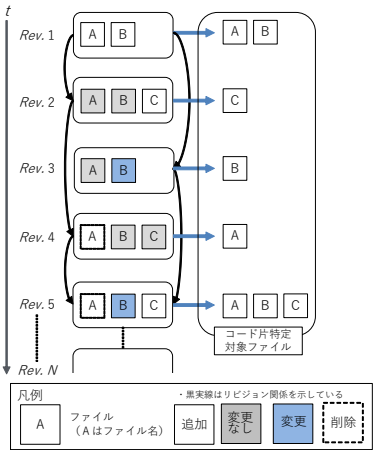


図 2: コード片特定対象ファイルの決定（リビジョンの親子関係は図 1 と同じ）

(1) Git におけるリビジョンの親子関係の特定：CCT は分散型バージョン管理システム Git を対象とするため，従来の集中型バージョン管理システムを対処としたクローン追跡 [4, 8, 14] のように時系列でリビジョンを 1 つ 1 つ順に辿る方法は適用できない．そのため，以下の手順でリビジョンの親子関係を特定する（図 1）．まず，分析対象リポジトリに含まれるリビジョンの情報を取得する．次に，リビジョン毎に親リビジョンと子リビジョンを特定し（図 1 左），各リビジョンの親子関係を記録する（図 1 右表）．リビジョンの親子関係を整理しておくことで，図 1 の *Rev.*1 や *Rev.*5 のように，複数の親リビジョンや子リビジョンを持つリビジョンが参照可能になる．

(2) 各リビジョンに含まれる全コード片の特定：ファイル変更情報を基にして各リビジョンのコード片特定対象ファイルを決定し（図 2），[13] と同様にブロック単位でコード片を特定する（図 3）．

まず，分析対象リポジトリのコミット履歴から，リビジョン番号，コミット日時，コミッタ名，追加・変更・削

[1] 実際，クローンの特定までの処理（コード片抽出，正規化，ハッシュ化）は，CRD (Clone Region Descriptors)[4] ベースのクローン追跡ツール ECTEC (Enhancement of CRD-Based Clone Tracker for Evolution of Clones) [8] の一部を再利用して実装している．

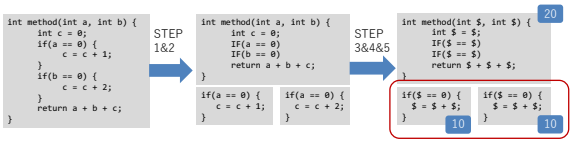


図 3: ブロックの検出・正規化・ハッシュ化の流れ（[13] より抜粋）

除されたファイルについての情報（コミット情報と呼ぶ）を取得する．次に，分析対象リポジトリの最初のリビジョン（リポジトリに初めてコミットされたファイル群から構成されるリビジョン）を調査し，最初のリビジョンを構成する全ファイルを特定する（図 2 における *Rev.*1）．コミット情報を用いて，各リビジョンに加えられたすべての変更（ファイルの追加・変更・削除）を，最初のリビジョンを起点として，リビジョンの親子関係に従って順次遡り最新のリビジョンまでコード片特定対象ファイルを決定する．マージコミット（図 2 における *Rev.*5）では，変更の衝突が発生する場合がある．衝突がない場合，マージ元の 2 つのコミットとの差分情報を取得し，変更が加えられたファイルをコード片特定対象ファイルとする．衝突が発生していた場合，衝突を解消するまでの履歴が抜け落ちていることがあるため，そのリビジョンに存在するファイル情報を基にしてすべてのファイルをコード片特定対象ファイルとする．なお，ソースコード以外のファイルは分析対象ではないため除外する．

コード片特定対象ファイルに対して行うコード片特定処理について以下で説明する．まず，最初のリビジョンを構成する全ファイルに対してブロック単位でコード片を特定する．図 3 の STEP1 でソースコードに対して字句解析，構文解析を行い，STEP2 でブロックの種類，条件式，シグネチャ，クラス名を特定する．ブロックとは，クラスやメソッド，for 文や if 文などのブロック文を指す．ブロック単位でコード片を特定した後，最初のリビジョンを起点として以降のリビジョンを構成するファイルについてもコード片を順次特定する．このとき，最初のリビジョン以降のリビジョンでは，ファイルの変更情報に基づいて，変更が加えられたファイル（図 3 のコード片特定対象ファイル）のみに対してコード片を特定し，変更がなかったファイルのコード片は前のリビジョンのコード片情報を再利用する．

(3) コードクローンの特定：前述の前処理により，すべてのリビジョンに対してブロック単位ですべてのコード片が特定される．コードクローンの特定処理においては，まず，[13] と同様の方法（図 3 の STEP3&4&5）で，特定されたブロックを定められたフォーマットに整形する．STEP3 において，ブロックに含まれる変数名とリテラルは特殊文字にすることで正規化し，変数名やリテラルのみが異なるようなクローンを特定できるようにする．さらに，STEP4 において，正規化されたブロックの文字列をもとにハッシュ値を算出する．そして，STEP5 において，ハッシュ値の一致するブロックをクローンとして特定する．このコードクローン特定処理を前述の前処理によって特定できた全てのリビジョンのコード片に対して行うことで，各リビジョンに存在するコードクローンを特定できる．

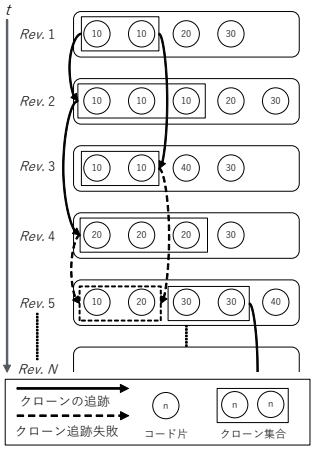


図 4: コードクローンの追跡（リビジョンの親子関係は図 1 と同じ）

(4) コードクローンの追跡：コードクローンの追跡処理では，最初のリビジョンに含まれるすべてのコード片のハッシュ値を比較し，同じハッシュ値を持つコード片の集合をクローン集合としてまとめ，以降のリビジョンでのクローン集合の前後関係を特定できるようにする．具体的には，以下の方法でクローン集合の前後関係を特定しクローン集合をリビジョン間で追跡する（図 4）．

まず，次の子リビジョンに存在しているクローン集合に含まれるコード片のハッシュ値を調べ，直前のリビジョンのクローンのハッシュ値と比較する．ハッシュ値が一致した場合は，前のリビジョンに存在するクローン集合と同じクローン集合であるので要素数を前後のリビジョンで比較する．要素数の増加によるものか，ハッシュ値が変化しない変数名やリテラルの変更によるものかを判

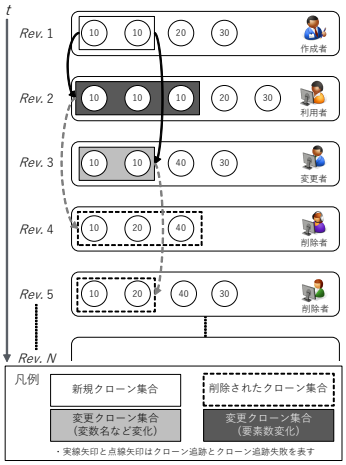


図 5: コードクローンの追跡と開発者の分類（リビジョンの親子関係は図 1 と同じ）

別後，前のリビジョンのクローン集合を親集合，次のリビジョンのクローン集合を子集合と解釈し，クローン集合間にリンクを張る．ハッシュ値だけの比較では追跡漏れが起きてしまうため，ハッシュ値の比較でリンクを張ることができなかったクローン集合に対して [8] で用いられている手法と同様に CRD の類似度比較（CRD はコード片のブロックの種類やソースコードにおける位置情報を含む表記形式で，この文字列が類似するかを比較）によってクローン集合間にリンクを張る．リンクが張られることがなかったクローン集合は，新たなクローン集合が作成されたものと解釈し，該当クローン集合を以降のリビジョンでの追跡対象とする．最後に，前のリビジョンにのみ存在するクローン集合は削除されたものと解釈して追跡対象から除外する．

従来の追跡ツールである ECTEC[8] では全てのコード片に対してリンクを張ることによって，コードクローンの追跡を行っていた．本ツールではコードクローンのみにリンクを張ることによってコードクローンの追跡を実現しているので，コードクローンとは関係のないコード片の追跡判定処理をする必要がなくなり，コードクローンの追跡にかかる処理時間を削減できる．

2.2 コードクローン分析機能

CCT はコードクローン追跡機能に加えて，コードクローンの作成・利用に関与した開発者とコードクローンの品質との関係を時間を遡って分析するための機能を以下の処理によって実現している．

(1) 開発者の特定処理：クローンの作成・利用過程に関与した開発者を作成者（オリジナルのクローンを作成した開発者）・利用者（既存のクローンを他の場所に移植した開発者）・変更者（ハッシュ値の変更を伴わないクローンに対する変更をおこなった開発者）・削除者（ファイルの削除などによりクローンを削除した開発者）の 4 種類に分けて特定し，リビジョンと紐づける（図 5）．

(2) 不具合混入リビジョンとファイルの特定処理：これまで紹介した機能はすべて Git リポジトリから得られる情報に基づいた処理であり，クローンの作成・利用過程に関与した開発者が混入した不具合に関する情報は別途，不具合管理システム（Bugzilla や Jira など）から取得する必要がある．本ツールには，SSZ アルゴリズム [11] を用いて不具合混入コミットを特定し，不具合を混入した開発者を分析する機能を実装してる．

(3) コードクローンと不具合紐づけ処理：本ツールで行ったコードクローン追跡処理と不具合混入リビジョンとファイルの特定処理の結果から得られる開発者と不具合の関係を用いて，コードクローンと不具合を紐づけることによって，クローン・人・不具合の紐付けできる．

3 分析：開発者のインタラクションとソフトウェア品質

本章では，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を調べるために CCT を用いて行う分析と，分析で用いたデータセットについて述べる．

3.1 目的

ファイルの作成・変更に関与する開発者と品質を分析した研究 [3, 2] から，開発者のインタラクションとソフトウェア品質には密接な関係があることが知られている．例えば文献 [3] では，安定して成長している OSS 開発プロジェクトでは潜在的に密なコミュニティが存在しており，そのコミュニティに属する開発者同士が同じファイルの変更に関与していることを明らかにしている．すなわち，特定の開発者からなるグループでの密なインタラクションがソフトウェア品質に寄与することを示唆するものである．一方，文献 [2] では，一人の開発者によって変更されてきたファイルよりも多数の開発者によって変

更が加えられてきたファイルの方が欠陥が含まれやすいことを明らかにしている．文献 [3] とは反対の結果ではなく，開発者の疎なインタラクションを通じて同一ファイルに対する変更が行われるとソフトウェア品質の低下を招くということを示唆したものである．

これらの研究成果は，ファイル単位での分析によりファイルの作成・変更に関与した開発者とソフトウェア品質の関係を示したものであった．そこで本稿では，コードクローン単位の分析でも同様の結果を導くことができるのかどうかを確かめることを目的とする．コードクローンとソフトウェア品質の関係については多数の研究が存在しているが，コードクローンがソフトウェア品質にとって有害である場合とそうでない場合が混在しており，統一的なコンセンサスは未だ得られていない．さらに，コードクローン変更過程における開発者とソフトウェア品質の関係について調べた研究はまだ多くないため，既存研究では示されていないコードクローンの長期的なメンテナンスのための新たな知見を提供できる可能性がある．

3.2 分析方法

本稿では，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を Pajek[2] により可視化し分析する．Pajek とは人と人，人とモノの関係などをネットワークとして表現し分析するためのネットワーク分析ツールの 1 つである．分析では，課題管理システムに報告された課題 (Issue) に関係しているコードクローン（Issue 混入クローン）と Issue 解決のためにコードクローンの変更に関与した開発者との関係を Pajek により可視化する．また，対比のため，Issue の解決とは全く関係のないコードクローン（Issue 未混入クローン）とその変更に関与した開発者との関係を可視化する．すなわち，品質に問題のあるコードクローンの変更に関与した開発者のインタラクションと，品質に問題のないコードクローンの変更に関与した開発者のインタラクションとを比較する．具体的には以下の手順で分析を行う．

(1) Issue 混入クローンと未混入クローンの分別：まず初めに，CCT の処理結果から得られる情報を基に，Issue と紐づいた Issue 混入クローンと，Issue と紐づいていない Issue 未混入クローンとに分ける．

(2) 開発者の特定：次に，各コードクローンを変更（追加，修正，削除）した開発者の情報を CCT の処理結果から取り出す．

(3) ネットワーク情報の作成：(1), (2) の情報を用いてコードクローンとコードクローンの変更に関与した開発者を紐付け，Pajek 形式のネットワークデータを作成する．

(4) コードクローンと開発者の関係の可視化：(3) により Pajek を用いてコードクローンと開発者の関係を無向グラフとして可視化する．

(5) 開発者のインタラクションの可視化：さらに，(4) から同一クローンの変更に関与した開発者のみを紐づけることにより開発者のインタラクションを無向グラフとして可視化する．

3.3 データセット

本稿では，GitHub[3] で活発に活動を行なっている 3 つのオープンソース開発プロジェクト（RxJava, c:geo, Jackson databind）を対象として分析を行う．RxJava はリアクティブプログラミングを支援するライブラリである．c:geo は Android 向けの GPS 機能を使ったゲームのアプリケーションである．Jackson databind は JSON 形式のデータを扱うことを支援するライブラリである．表 1 に各プロジェクトの基本統計量を示す．

表 1 中の「Issue 混入 CC 変更」,「Issue 未混入 CC 変更」とはそれぞれ，Issue 混入クローンの変更に関与した開発者の数と Issue 未混入クローンの変更に関与（Issue とは関連のない理由でコードクローンを変更）した開発者の数を指す．プロジェクト間でばらつきがあるものの，プロジェクトの開発者の約 1 割〜3 割程度がコードクローンの変更に関与していることが見て取れる．なお，Github では Issue の種類をプロジェクト毎に自由に定義するタグで管理する仕様になっておりプロジェクト間に共通のタグが存在しないため，本データセットにおける Issue には不具合に関するものから機能拡張（エンハンスメント）に関するものまで含まれている．特定の種類の課題のみを指すものではないことに注意されたい．Issue の種類を厳密に区別するためにはすべての Issue を目視する必要があるため，本データセットでは区別しないこととしたが，分析結果に影響を与える可能性があり本研究の今後の課題とする．

[2] Pajek, http://vlado.fmf.uni-lj.si/pub%20/networks/pajek/

[3] https://www.github.com/

表 1: 対象プロジェクトの統計量

プロジェクト名		RxJava	c:geo	Jackson databind
対象期間		2014/08/30 ∼2016/06/29	2011/07/11 ∼2016/07/21	2011/12/23 ∼2016/07/21
リビジョン数		1,397	9,821	3,260
開発者数（人）(∗)	開発者総数	73	107	114
	Issue 混入 CC 変更者	18	28	17
	Issue 未混入 CC 変更者	17	27	11
コードクローン（セット数）	CC 合計	289	2,660	1,610
	Issue 混入 CC	104	2,008	1,223
	Issue 未混入 CC	185	652	387
Issue 数（件）	Issue 合計	1,892	5,849	1,307
	CC 関連有 Issue	35	405	241
	CC 関連無 Issue	1,857	5,444	1,066

(∗) 開発者総数とは，プロジェクトでコードを変更したことのあるすべての開発者の数である．直下の 2 項目（Issue 混入/未混入 CC 変更）はコードクローン (CC) の変更に関与した開発者の数を示すもので，それら 2 項目の合計とはプロジェクト全体の開発者の合計は一致しない．

表 1 中,「Issue 混入 CC」,「Issue 未混入 CC」とはそれぞれ，変更されたコードクローンの内，Issue が混入していたものと未混入だったもの数を意味している．それぞれを足し合わせたものが合計と一致する．なお，コードクローンは，クローンとなっているコード片の総数ではなくクローン関係にあるコード片集合（セット数）を示すものである．コードクローンはリビジョン数の違いを考慮してもプロジェクト間で大きな差異があることがわかる．特に，c:geo では 2,008 セットの Issue 混入クローンが存在しており，コードクローンに対する変更の多くは Issue に関連したものである．Jackson databind も同様に，コードクローンの変更の多くは Issue に関連していることが見て取れる．

表 1 中,「CC 関連有 Issue」,「CC 関連無 Issue」とはそれぞれ，報告された Issue を解決するためにコードクローンの変更を伴ったものとコードクローンの変更は伴わなかったものを意味している．それぞれを足し合わせたものが Issue 合計と一致する．RxJava については，報告された Issue に占める CC 関連有 Issue は少ない (35/1,892) が，c:geo および Jackson databind については比較的多くの Issue がコードクローンの変更に関係していることが見て取れる．

4 分析結果と考察

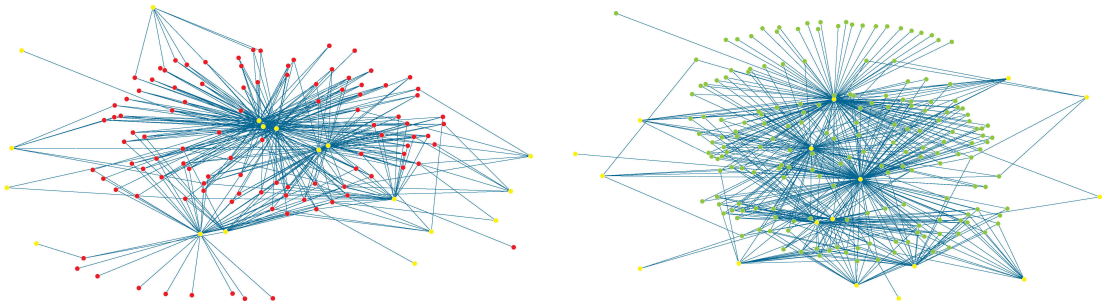
本章では，前章で述べたデータセットに対して Pajek を用いて可視化した開発者のインタラクションをプロジェクト毎に示し結果を考察する．

4.1 概要

Pajek で可視化した開発者のインタラクションを可視化した結果を図 6∼図 11 に示す．すべての可視化結果において，赤色のノードは Issue 混入クローンを，緑色のノードは Issue 未混入クローンを表す．黄色のノードが開発者である．

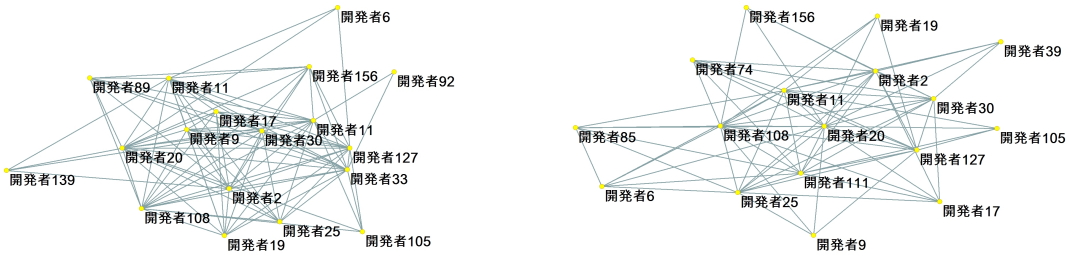
図 6, 8, 10 は，コードクローンを変更した開発者と変更されたコードクローンの間をエッジで結び，コードクローンとコードクローンの変更に関与した開発者の関係（開発者とコードクローンのインタラクション）を，Issue 混入クローンと Issue 未混入クローンとで対比できるように可視化したものである．

図 7, 9, 11 は，図 6, 8, 10 のグラフから，同じコードクローンを変更したことのある開発者間をエッジで結び，コードクローンの変更に関与した開発者同士の関係（開発者間のインタラクション）のみを抽出し可視化したものである．開発者とコードクローンのインタラクションと同様に，Issue 混入クローンと Issue 未混入クローン



(a) Issue 混入（密度：0.603）　　**(b)** Issue 未混入（密度：0.394）

図 6: コードクローンと開発者の関係 (RxJava)



(a) Issue 混入（密度：0.614）　　**(b)** Issue 未混入（密度：0.529）

図 7: 開発者の関係 (RxJava)

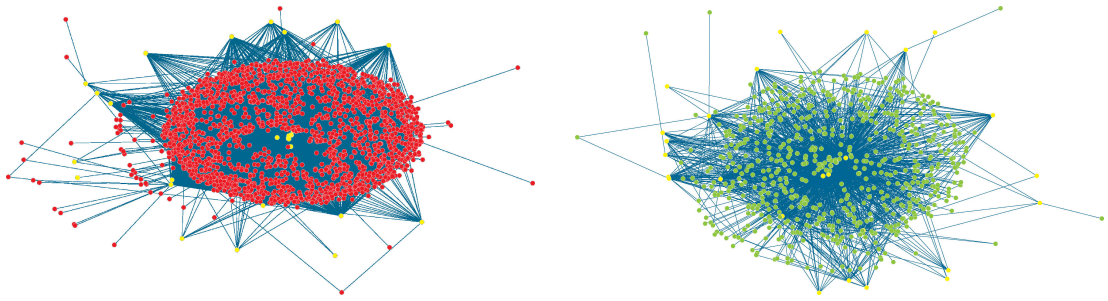
とで対比できるようにしている．開発者間のインタラクションは，開発者同士が同じコードクローンを共同で作成・変更・利用したことを意味する．つまり，図 7, 9, 11 のグラフが密になっているということは，開発者同士が同じコードクローンを共同で作成・変更・利用したことが多いということを示している．各グラフの密度はあるグラフの辺の数とそのグラフの可能な辺の数の最大数で割ったものである．

次節では，これらの可視化結果をプロジェクトごとに考察する．

4.2 RxJava

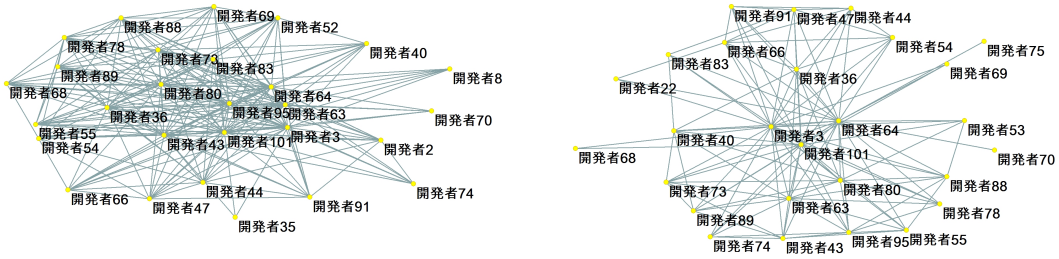
表 1 より，RxJava は他の 2 プロジェクトに比べて変更されたクローンセット数（CC 合計: 289, Issue 混入 CC: 104, Issue 未混入 CC: 185）が少ないことが分かる．図 6 からも変更されたコードクローンの数に違いがあるものの，開発者とコードクローンのインタラクションの基本構造に大きな違いがないことが見て取れる．実際，図 6(a), (b) のどちらも 5, 6 名の開発者が数多くのコードクローンの変更に関与していることが分かる．

一方，図 7 からは Issue 混入クローン数 (104) は Issue 未混入クローン数 (185) よりも少ないにもかかわらず，図 7(a) の方が若干ではあるが密な構造になっていることが分かる．コードクローンは開発担当者の異なる複数のファイルに散在していることが多いが，いずれかのファイル中のコードクローンを変更した場合にはその他のファイル中のコードクローンも変更する必要が生じやすい．特に Issue 混入クローンでは，コードクローンの変更により Issue を混入させる，あるいは，Issue を解決するためにコードクローンを変更する必要があるため，コードクローンを含むファイル間の依存関係が Issue 未混入コードクローンの変更よりも強く現れやすいものと考えられる．結果的に，Issue 混入クローンの変更に関与した開発者同士のインタラクションは，Issue 未混入クローンの変更に関与した開発者同士のインタラクションよりも密になるものと思われる．



(a) Issue 混入（密度：0.891）　　**(b)** Issue 未混入（密度：0.333）

図 8: コードクローンと開発者の関係 (c:geo)



(a) Issue 混入（密度：0.597）　　**(b)** Issue 未混入（密度：0.351）

図 9: 開発者の関係 (c:geo)

4.3 c:geo

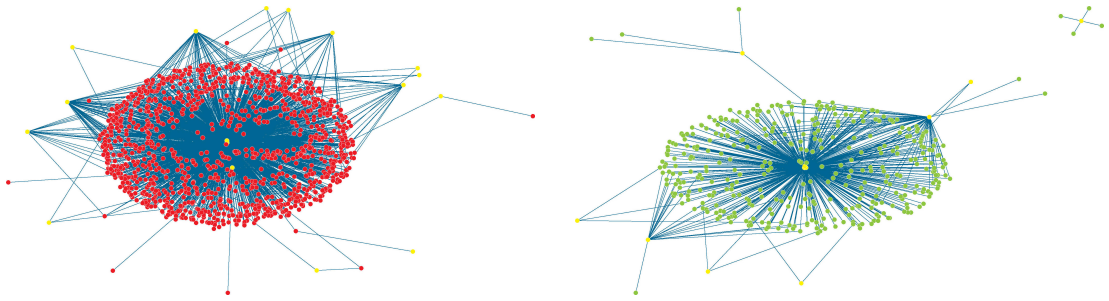
表 1 より，c:geo は他の 2 プロジェクトに比べてリビジョン数が最も多く，変更されたクローンセット数（CC 合計: 2,660, Issue 混入 CC: 2,008, Issue 未混入 CC: 652）が非常に多いことが分かる．一方，コードクローンの変更に関与した開発者自体は十数名多い程度（Issue 混入 CC 変更者: 28, Issue 未混入 CC 変更者:27）であるため，図 8 のグラフではコードクローンのノードとそのエッジがグラフ要素の大半を占める．その結果，開発者とコードクローンのインタラクションの詳細を読み取ることができないが，少なくとも少数の開発者が多数の Issue 混入/未混入クローンの変更に関与していることは見て取れる．RxJava と同様，少数の中心的な開発者がコードクローンの変更において重要な役割を担っていると思われる．

図 9 からは，コードクローンの変更に関与した開発者の数はほぼ同じであるが，Issue 混入クローン数 (2,008) が Issue 未混入クローン数 (652) よりも相当多いため，Issue 混入クローンを変更する場合の方が開発者同士のインタラクションが密になっていることが分かる．Issue 未混入クローンの変更では，Issue 混入クローンの変更とは異なり数名の開発者がより多くの繋がり（エッジ）を持っているが全体としては比較的疎な構造担っていることが分かる．

4.4 Jackson databind

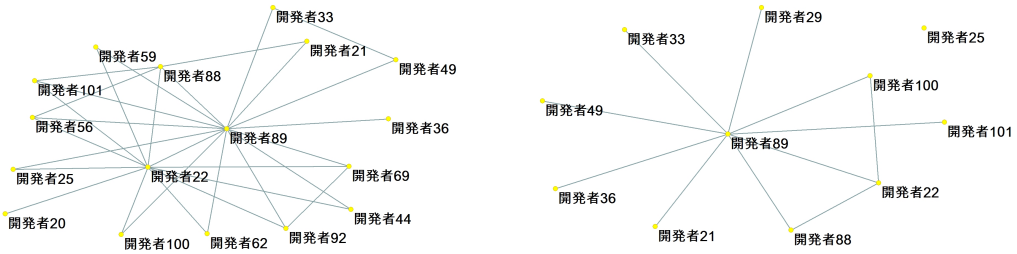
表 1 より，Jackson databind のリビジョン数 (3,260) は c:geo (9,821) の 1/3 程度であることが分かる．一方，Issue 混入 CC および CC 関連有 Issue は 1/2 程度であるため，Jackson databind におけるソースコードの変更は 3 つのプロジェクトの中で最もコードクローンの変更によるものであることが読み取れる．図 10 からは，Issue 混入クローンの変更では 2 名，Issue 混入クローンの変更では 1 名の開発者がほとんどのコードクローンの変更に関与していることが分かる．

図 11 からも，同様のことが見て取れる．Jackson databind では，1,2 名の中心的な開発者が作成したソー



(a) Issue 混入（密度：0.855）　　**(b)** Issue 未混入（密度：1.223）

図 10: コードクローンと開発者の関係 (Jackson databind)



(a) Issue 混入（密度：0.235）　　**(b)** Issue 未混入（密度：0.231）

図 11: 開発者の関係 (Jackson databind)

スコードには多くのコードクローンが含まれており，その他の開発者がそれらコードクローンを利用した結果を表しているものと思われる．Jackson databind においても，Issue 混入クローンの変更の場合の方が 2 名以外の開発者のインタラクションも若干密な構造になっている．Issue 未混入クローンの変更の場合は，1 名の開発者のみが全ての開発者とインタラクションが存在する構造になっており，他の開発者同士の繋がりはほとんど存在しない．

5. まとめと今後の課題

本稿では，コードクローンの編集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を理解する事を目的として，我々が提案しているコードクローン追跡ツール CCT (Code Clone Tracer) を用いて，3 つのオープンソース開発プロジェクト（RxJava, c:geo, Jackson databind）を対象とした社会ネットワーク分析を行った．

社会ネットワーク分析ツール Pajek によって可視化した結果，以下の知見が得られた．

- いずれのプロジェクトにおいても少数の開発者のみがコードクローンの編集の大多数に関与しており，Issue の存在の有無に関わらず各プロジェクトにおける開発者とコードクローンのインタラクションの全体構造に大きな違いはなかった（図 6, 8, 10）．
- Issue 混入クローンの変更に関与した開発者同士のインタラクションは比較的密な構造になる（図 7, 9, 11），すなわち，コードクローンに何らかの問題がありそれを解決するためには多くの利害関係者の協力を必要とする．問題のあるコードクローンの編集は多数のソースコードの変更という観点だけではなく，それぞれの担当者との調整や協力という観点からも影響範囲がより大きくなる可能性がある．

本稿では，CCT の処理結果をもとに社会ネットワーク分析を行ったが，まだ基本的な分析を終えた段階であり，より有益な知見を提供するためには今後はさらに分

析を精錬する必要がある．本研究の今後の課題としては以下のようなものが挙げられる．

- Issue の種類を分別する（本稿では不具合なのか機能拡張なのか等を区別していない）．
- Issue の内容を識別するとともに変更されたコードクローンの内容を分析する（本稿では Issue の内容やコードクローンの変更内容を詳細に把握しておらずコードクローンの変更理由は不明なままである）．
- Issue の内容を識別する（本稿では Issue の内容を詳細に把握していないのでコードクローンの変更理由は不明なままである）．
- ファイル単位の従来の分析結果 [3, 2] と比較して，コードクローンの編集過程においてのみ特徴的な現象を抽出する（本稿ではファイル単位での分析は行っていないため，前述の知見はコードクローンの変更に関してのみ観察される事象でない可能性がある）．
- 開発者とコードクローンのインタラクションを時系列に可視化する（本稿では，過去のデータを一度に可視化しているためコンテキストが失われている）．
- 分析対象とする OSS プロジェクトデータを増やす，あるいは，企業のプロジェクトデータを対象にする（本稿では，Github 上の 3 つの OSS プロジェクトを対象にしたのみ）．

謝辞

本研究の一部は，文部科学省科学研究補助金（基盤 (C): 15K00101）による助成を受けた．

参考文献

[1] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proc. of the Int'l Conf. on Software Maintenance (ICSM '98)*, pp. 368–377, 1998.

[2] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. Don't touch my code!: Examining the effects of ownership on software quality. In *Proc. of the Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC/FSE '11)*, pp. 4–14, 2011.

[3] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects. In *Proc. of the 16th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE '08)*, pp. 24–35, 2008.

[4] E. Duala-Ekoko and M.P. Robillard. Clone region descriptors: Representing and tracking duplication in source code. *ACM Trans. on Software Engineering and Methodology (TOSEM)*, Vol. 20, No. 1, pp. 3:1–3:31, 2010.

[5] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. In *Proc. of the IEEE Int'l Conf. on Software Maintenance (ICSM '99)*, pp. 109–118, 1999.

[6] N. Göde and R. Koschke. Frequency and risks of changes to clones. In *Proc. of the 33rd Int'l Conf. on Software Engineering (ICSE '11)*, pp. 311–320, 2011.

[7] J. Harder. How multiple developers affect the evolution of code clones. In *Proc. of the 2013 IEEE Int'l Conf. on Software Maintenance (ICSM '13)*, pp. 30–39, 2013.

[8] Y. Higo, K. Hotta, and S. Kusumoto. Enhancement of crd-based clone tracking. In *Proc. of the 2013 Int'l Workshop on Principles of Software Evolution (IWPSE '13)*, pp. 28–37, 2013.

[9] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *Proc. of the 31st Int'l Conf. on Software Engineering (ICSE '09)*, pp. 485–495, 2009.

[10] 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.

[11] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *Proc. of the 2005 Int'l Workshop on Mining Software Repositories (MSR '05)*, pp. 1–5, 2005.

[12] 大平雅雄, 久木田雄亮. コードクローンの作成・利用過程における人的影響を調査するための追跡ツールの試作. 情報処理学会 ソフトウェア工学研究会, 第 2016-SE-193 巻, pp. 1–25, 7 2016.

[13] 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.

[14] 堀田圭佑, 肥後芳樹, 楠本真二. Crd を用いたコードクローンの生存期間と修正回数に関する調査. 情報処理学会論文誌, Vol. 55, No. 2, pp. 947–958, 2013.

[15] 堀田圭佑, 肥後芳樹, 楠本真二. 生成抑止，分析効率化，不具合検出を中心としたコードクローン管理支援技術に関する研究動向. コンピュータソフトウェア, Vol. 31, No. 1, pp. 14–29, 2014.

[16] 久木田雄亮, 大平雅雄. コードクローンの編集過程における開発者の潜在的社会構造に関する分析. 2016 年度情報処理学会関西支部大会 支部大会講演論文集, 第 2016 巻, 9 2016.

Convolutional Neural Network を用いたフォールト数予測手法

小川 直記
株式会社 日立製作所
naoki.ogawa.od@hitachi.com

坂井 伸圭
株式会社 日立製作所
nobuyoshi.sakai.wx@hitachi.com

横内 弘
株式会社 日立製作所
hiroshi.yokouchi.dn@hitachi.com

要旨

ソフトウェア開発のテスト工程において，いかに品質を確保しつつ，テストの効率を上げるかは重要な課題の一つである．そのために，品質状況を判断し，品質と効率のバランスを最適化する必要がある．そこで，我々は品質状況を判断するための指標の一つとして，「摘出フォールト数の目標値」をテストの十分性を判断するために用いている．この際，高い精度で予測出来なかった場合は，過剰なテストを実施してしまう．しかし，高い精度の予測には開発者やテスト担当者が長年培ってきた経験が必要であるため，有識者に作業が集中してしまい全体のボトルネックとなってしまっていた．本研究では，過去のソフトウェア開発プロジェクトのデータから Convolutional Neural Network（以下 CNN と略す)を用いてフォールト数の予測を行い，定量的評価を行う．評価の結果，実際のプロジェクトで使用可能な精度でフォールト数の予測が可能なことがわかった．

1. はじめに

ソフトウェア開発において，テスト工程の工数が全工程に占める割合は大きく，例えば，新規開発の場合では結合テストと総合テストを合わせると，全工程の 28.6%を占め，改良開発においては 33.2%を占めるという報告[19]がある．このような中で，ソフトウェア品質を確保するためのテスト工程においては，実運用に耐えられる品質を確保しつつ，効率的にテストを実施することが必要である．

しかし，効率のためにテスト工数を減らすと，摘出フォールト数が少なくなるため，品質が下がり，品質確保に注力してテスト工数を増やすと，摘出フォールト数は頭打ちになるため，テストの効率が下がるというトレードオフの課題がある．

そこで，テスト工数を適切に割り当てるテスト戦略を立て，効率的に多くのフォールトを発見することが重要である[15]．

我々のユースケースベースのソフトウェア開発[9]でも，品質指標を用いてテスト戦略を立てており，テストの十分性を判断するための品質指標の一つとして，「摘出フォールト数の目標値」を用いてきた．「摘出フォールト数の目標値」は，テストの計画時に予測し，テスト後に実際にテストで摘出されたフォールト数と比較して，品質状況の判断に用いる．また，値の算出は，過去のプロジェクト情報や開発対象のユースケースごとの仕様書を基に，有識者の経験から推定する必要がある．そのため，有識者に作業が集中することでプロジェクトのボトルネックとなり，さらにフォールト数の予測値が有識者の経験に依存してしまう．

本研究では，ユースケースベースのソフトウェア開発を対象として，仕様書の情報と過去のプロジェクト情報から，ユースケースごとのフォールト数を予測する手法を提案する．過去のフォールト数と仕様書の文章データ，仕様書のページ数等の数値データを CNN[3]で学習させることで，フォールト数の予測を行う．本研究では，我々が実際に行った過去のソフトウェア開発プロジェクトのデータからフォールト数の予測を行い，定量的評価を行う．また，評価の結果，実際のプロジェクトで使用可能な程度の精度でフォールト数の予測が可能なことがわかった．

2. 関連研究

フォールト数の推定では，開発プロジェクトの工程ごとに，フォールト数を推定するモデルが提案されている．例えば，フォールト数の推移を時間や工数の関数として定義したレイリーモデルを使用する方法がある[12]．レイリーモデルでは開発工程ごとのフォールト数が予測できる利点はあるが，モデルの形状パラメータの設定が必要であり，モデルに即していない場合は利用できない．また，

形状パラメータを過去のプロジェクト実績から設定する方法も提案されている[16]が，ユースケースベース単位で設定する場合，過去のプロジェクト実績から適切なデータを抽出するのは難しい．ファイル単位でコードメトリクスや変更履歴から，フォールト数やフォールト密度を予測する手法[5]も存在するが，ファイルがどのユースケースで使用されるか推定することは難しいため，ユースケースベースでのフォールト数予測に適用することは難しい．他にも製品のフォールト数が推定できるシステムとして，SQE[1]が提案されている．しかし，ウォーターフォール型の画一的なソフトウェア開発時に製品全体への適用を想定されており，反復的かつユースケースベースの開発に適用するのは難しいことや，初期値の設定に経験が必要である．

また，品質を確保し，テストやレビューを効率的に行うために，フォールトを含んでいそうなモジュール(fault-prone module)を推定する研究は数多く行われている[2][18]．例えば，コードメトリクスや開発履歴のメトリクス，プログラムをテキストマイニングした結果を用いて，分類木[4]やロジスティック回帰，機械学習[6]で推定する手法が提案されている．また，fault-prone module の中でも Security[7][14]や Buffer Boundary Violation[17]等の特定の分野に特化した研究もおこなわれている．ただし，モジュール単位での推定であり，実際のプログラムが既に存在することが前提であることや，フォールト数の推定は行っていない．

3. CNN を用いたフォールト数予測手法

本研究では，有識者が仕様書の内容を読み，フォールト数の予測をすることを，CNN を用いて模倣する．

3.1. 従来のフォールト数予測

従来は，有識者がユースケースの規模と仕様書の内容から，経験をもとにフォールト数を予測していた．有識者がフォールト数を予測する場合は次のような作業を行う．まず，予め過去のユースケースの規模とフォールト数の実績値から，統計的に係数を算出しておく．予測する際に，算出された係数を用いて，新しいユースケースの規模に係数を掛け，予測値のベースとなる値を算出する．次に，有識者は仕様書の内容を読み，ユースケースの難しさを考慮し，経験によってベースとなる値を補正することでフォールト数を予測する．

3.2. CNN を用いた有識者の思考のモデル化

本研究では，ユースケースの規模と仕様書の内容から，経験をもとにフォールト数を予測していた有識者の思考を，CNN で模倣する．ユースケースの規模の情報として仕様書のページ数，ドキュメント数，単語数の数値データを使用し，ユースケースの難しさの情報して仕様書の文章データを使用する．ユースケースの規模の情報としては，他にも開発予定工数やコード量を使用することが考えられる．しかし，開発予定工数は有識者の経験で予測しているため，経験に依存する．また，コード量は異なるユースケースで使用されているコードがあることや，機能単位ではないため，コードとユースケースの対応付けが難しい．そのため，ユースケースの規模の情報として仕様書の数値データを用いる．有識者の思考を模倣するために，CNN を用いてユースケースごとの仕様書の数値データと仕様書の文章データの特徴を学習させる．ここで，本研究で入力として扱う仕様書とは，ユースケースごとに概要，外部仕様，内部仕様が記載されたものを指す．

3.3. CNN の構造

図 1 に本研究で用いた CNN の構造を示す．但し，各 Convolution Layer 及び Fully Connected Layer に対しての Dropout Layer は省略している．また，各レイヤの上にレイヤの名称と行，列，深さの順で次元数を記述する．但し，列，深さについて，存在しない場合は省略して記述する．本手法では活性化関数は全て ReLU 関数を用いている．Output Layer で ReLU 関数を用いた理由として，フォールト数は 0 件が最小値となるからである．また，最適化アルゴリズムには Adam[11]を使用し，コスト関数には最小二乗誤差を用いた．

3.4. 入力値の前処理

本手法では Input1，Input2 の入力値に対して，それぞれ次の前処理を行う．

- Input1

  入力する仕様書は，MeCab[20]を用いてわかち書きし，単語単位に区切る．仕様書の類似性を CNN に判別させるために，Word2Vec[8]を用いて 196 次元の単語の分散表現とする．また，仕様書を 500 単語ずつに分け，1 つの仕様書を複数の仕様書として扱う．例えば，2000 単語の仕様書があった場合，4 つの仕様書として扱う．また，500 単語に満たない場合，足りない部分は零ベクトルでパディングをする．CNN は主に画像処理で用いられる技術

であり，隣接情報を考慮した特徴を抽出することが出来る．自然言語処理では隣接した単語の特徴を抽出することで，単語の並び順を考慮した文脈の特徴を抽出が可能となることが期待できる．

- Input2

学習データに偏りが出ないように，仕様書のページ数，ドキュメント数，単語数の 3 つの数値データについて，平均 0，分散 1 となるように正規化を行う．Input2 は 1 つのユースケースに対して 1 つが定まるが，Input1 で 1 つの仕様書が複数の仕様書に分割される．この際には分割された仕様書に対して，Input1 は全て同じ値を入力とする．一般に CNN では多数の学習データが必要だが，多数のデータを得ることが難しい場合，疑似的にデータを増やすことが行われる．例えば，画像処理の分野では 1 枚の画像の明るさを変更したり，拡大，縮小，回転変形を加えたりし，データ数の問題を解決する．また，このようにデータを増やすことで，ノイズに強い学習モデルが生成できる．本研究では 1 つの仕様書を複数の仕様書に分割することで，同等の効果を得ることを目指す．

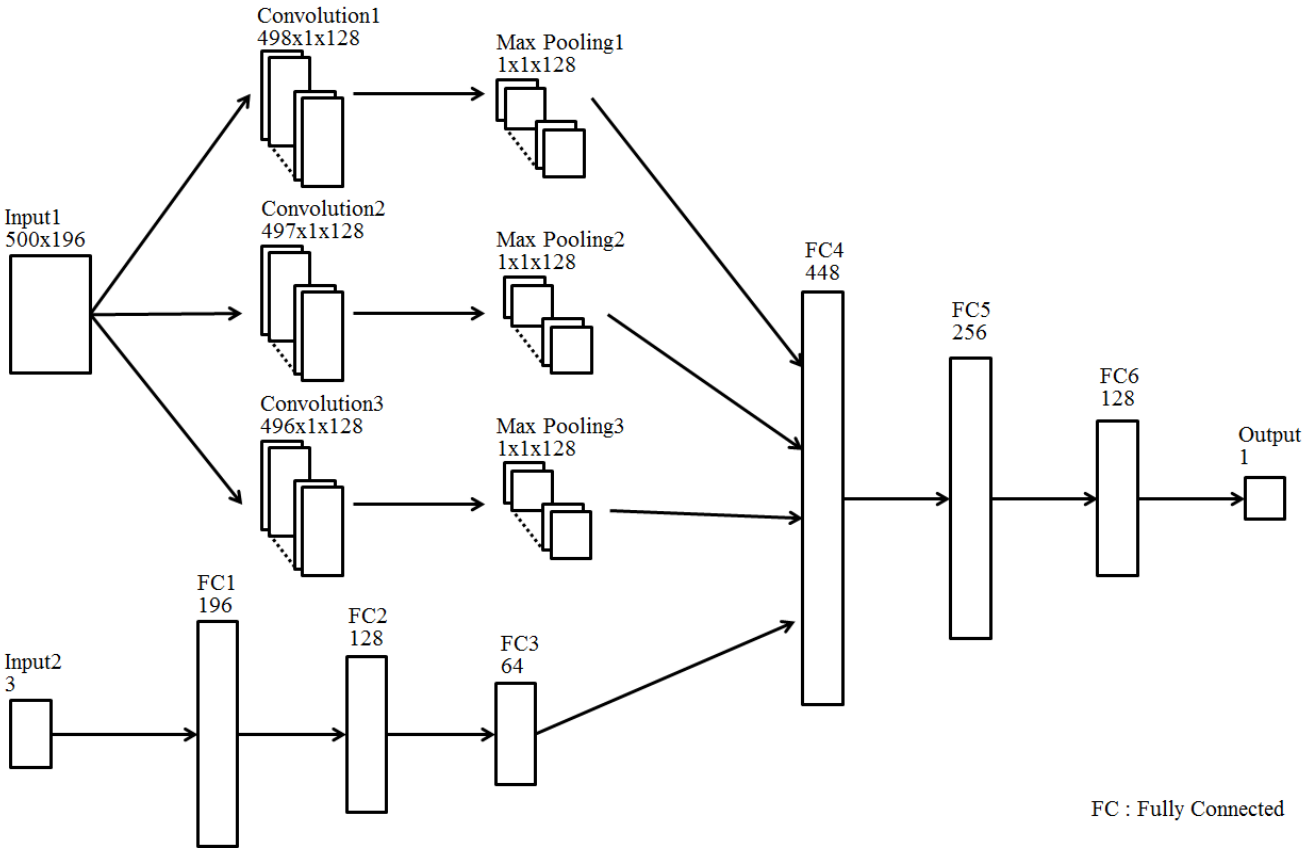


図 1：本手法における CNN の構造

4. 評価

4.1. 評価尺度

フォールト数予測精度の評価尺度として，MAE(Mean Absolute Error)，SD(Standard Deviation)の 2 つの指標を用いた．フォールト数の実測値をx，予測値を$\hat{x}$，データ数を N としたとき，それぞれの値は以下の式(1)，(2)で求められる．

$$\mathrm{MAE} = \frac{\sum |x - \hat{x}|}{N} \quad (1)$$

$$\mathrm{SD} = \sqrt{\frac{\sum (x - \hat{x})^2}{N}} \quad (2)$$

MAE は予測値のずれを表し, SD はユースケースごとにフォールト数の予測値がばらつかないかを示す指標であり，MAE，SD 共に小さい値は精度がよいことを表す.

4.2. 評価データ

本研究では, 開発プロセスや開発対象が類似している 2 つの製品 A, B を用いて検証を行った. 検証のために過去のプロジェクトのデータを学習データ, テストデータの 2 つに分けて精度の検証を行った. ここで, 学習データのバージョンはテストデータのバージョンより前のバージョンである. 学習データ, テストデータに用いたバージョン数とユースケース数をそれぞれ表 1, 表 2 に示す.

表 1 : 学習データのバージョン数とユースケース数

製品	バージョン数	ユースケース数
A	9	75
B	4	24

表 2 : テストデータのバージョン数とユースケース数

製品	バージョン数	ユースケース数
A	2	11
B	2	12

5. 評価結果

5.1. 結果

学習回数を 1500epoch とし, 1500epoch 目のモデルで製品 A, B のフォールト数を予測, 評価した結果を表 3 に示す. また, 仕様書の文章データと数値データを合わせた効果の検証のために, 仕様書の文章データのみから予測した結果と, 仕様書ページ数, ドキュメント数, 単語数の 3 つの数値データのみから予測した結果も併記し, それぞれ(仕), (数)と付記する.

表 3 : 同じプロジェクトの過去バージョンの学習データからフォールト数を予測した結果

学習製品	予測製品	MAE	SD
A	A	1.22	0.89
A(仕)		3.04	1.80
A(数)		7.35	11.58
B	B	3.72	2.16
B(仕)		4.09	2.51
B(数)		1.72	1.51

さらに, A と B を混ぜて学習させることで, 学習データ数を増やし, 予測精度が上げられないか検証した結果を表 4 に示す. ここで, A と B を混ぜたデータを A+B と表記する.

表 4 : 別プロジェクトのデータを混ぜた学習データからフォールト数を予測した結果

学習製品	予測製品	MAE	SD
A+B	A	1.27	0.85
A+B(仕)		3.61	2.29
A+B(数)		7.74	21.72
A+B	B	1.40	1.04
A+B(仕)		5.54	3.37
A+B(数)		2.37	4.48

次に, 学習の収束状況を評価するために, 製品 A, B のフォールト数予測における, 5epoch ごとの MAE, SD の値を図 2～5 に示す. ここで A+B(学)は本手法のモデルがフォールト数を予測するための表現力を有しているかを示すため, 学習データを用いてフォールト数を予測したものである.

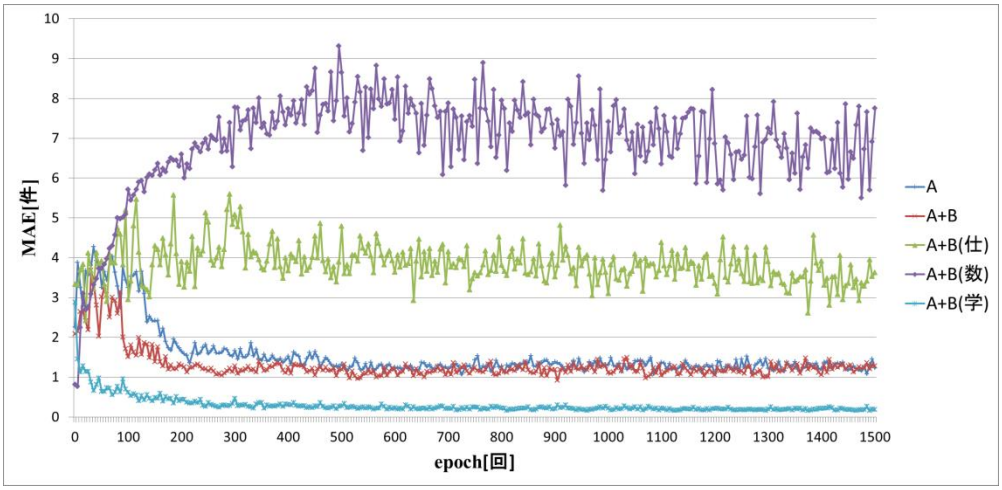


図 2：製品 A のフォールト数を予測した時の MAE の 5epoch ごとの推移

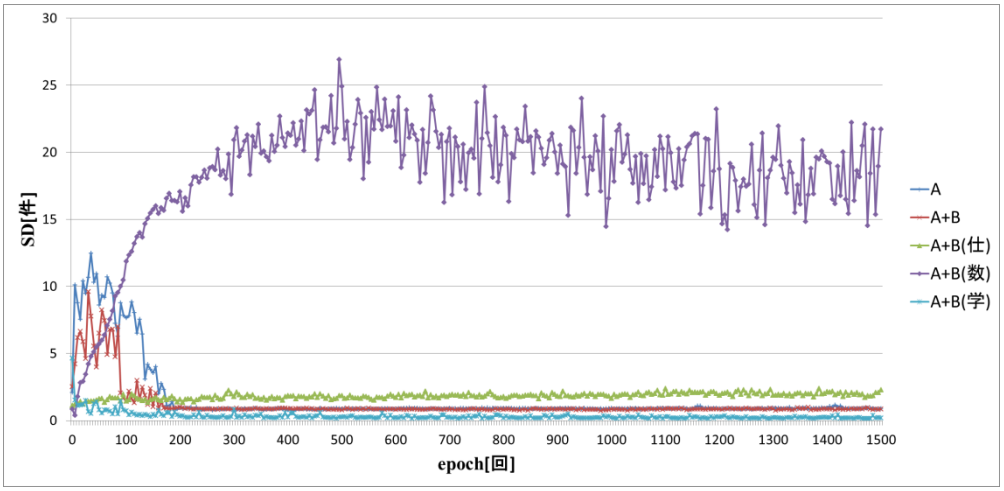


図 3：製品 A のフォールト数を予測した時の SD の 5epoch ごとの推移

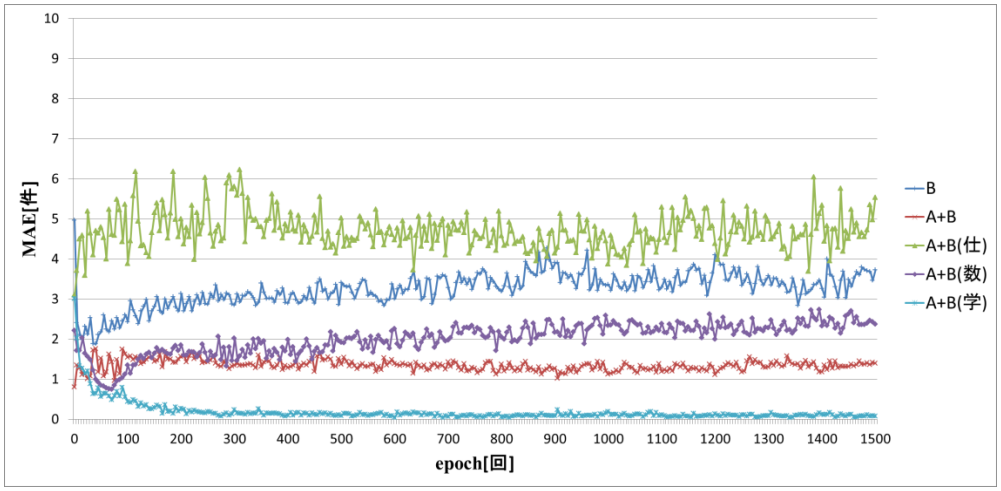


図 4：製品 B のフォールト数を予測した時の MAE の 5epoch ごとの推移

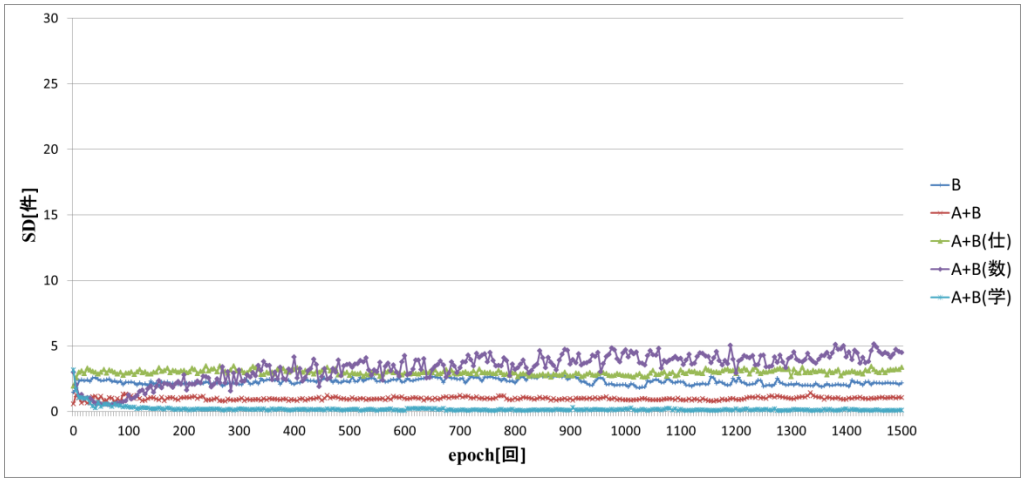


図 5：製品 B のフォールト数を予測した時の SD の 5epoch ごとの推移

6. 考察

6.1. 本手法の予測精度

表 3 より，製品 A では本手法の予測精度が最も高く，仕様書の文章データと数値データを合わせることで精度がよくなる効果が見て取れる．一方，製品 B では数値データのみが最も精度が高くなっている．製品 A，B では表 1 より，学習したユースケース数が 3 倍程度異なっているため，学習データ数が異なる．製品 B では学習データが十分でないため，学習しきれていない可能性がある．表 4 より，開発プロセス，開発対象が似ている A，B のデータを混ぜ，学習データ数を同じにして学習させた場合，A は予測値がほぼ同等であるが，B の予測値は改善が見られ，A，B 共に本手法の予測値が最も高くなっている．一方，仕様書のみや数値データのみの場合，A と B を混ぜると精度が悪くなっている．以上により，仕様書の文章データと数値データを合わせることで，どちらか一方だけでは表せないユースケースの特徴が存在し，本手法の CNN モデルで学習できていると考えられる．また，ユースケースの特徴は，開発プロセス，開発対象が似ている場合，共通している部分があると推測できる．そのため，製品 B は製品 A と比べ，学習ユースケース数が少なかったが，本手法のモデルを使用することで，少なかったデータを別プロジェクトのデータで補って，精度の高い予測ができたと考えられる．この結果は，異なるプロジェクト間で汎用的なモデルが生成できるということを示唆している．

ここで，参考までに，我々の有識者による予測値のデータから MAE と SD を算出した所，それぞれ 1.50[件]，1.98[件]であり，A と B を合わせて学習したモデルの場合の予測値では，有識者の精度以上となった．

6.2. 1500epoch での評価値の妥当性

本研究では，学習を 1500 回で打ち切って評価を行っている．しかし，固定回数での学習打ち切りでは，偶然精度の高い値になっただけの可能性がある．そこで，図 2，4 において MAE の値に注目すると，A+B を学習データとした時，200epoch 程度で小さな振動はあるが，一定値に収束していることがわかる．予測値の小さな振動は，過学習を抑えるために Dropout を実施していることが原因と考えられる．また，図 3，5 より SD の値に関しても，一定 epoch で収束している．よって，適当な回数で打ち切っても予測精度への影響は少なく，固定回数での打ち切りは実用上問題ないと考えられる．

6.3. モデルの妥当性

学習データに対して，MAE と SD が 0 に近づかなかったり，振動し続けていたりする場合，入力値に対して出力値を表現することが出来ないモデルであり，学習モデルとして妥当ではない．図 2～5 において，MAE，SD の値は 0 付近に収束しており，本研究における仕様書からフォールト数を予測するために妥当なモデルであるといえる．

6.4. 本手法の限界

本研究では，ある製品プロジェクトの過去のバージョンでフォールト数の予測を検証した．さらに，学習に使用していないプロジェクトのフォールト数を予測出来れば，適

用範囲が広がる. そこで, 製品 A, B をそれぞれ学習データとし, 製品 B, A のフォールト数をそれぞれ予測できるか検証を行った. しかし, 表 5に示す通り, 学習に使用していないプロジェクトへの適用は実現できなかった. 本研究では, 仕様書の文章データと, 仕様書のページ数, ドキュメント数, 単語数からフォールト数の予測を行ったが, コードメトリクス等の導入していないデータをいれることで, 学習に使用していないプロジェクトのフォールト数を予測できる可能性はあり, さらなる検証が必要である.

表 5: 学習に使用していないプロジェクトのフォールト数予測精度

学習製品	予測製品	MAE	SD
A	B	8.57	9.56
B	A	3.94	2.35

7. おわりに

本研究では, CNN を用いてフォールト数の予測モデルを構築し, 実際の開発プロジェクトのデータを用いて, ユースケースベースでのフォールト数予測を行い, 評価した. 評価の結果, 実際のプロジェクトで使用可能な精度でフォールト数の予測が可能ことを明らかにし, 有識者の経験に頼っていた部分を機械で置き換えられる可能性を示した.

また, CNN に学習させるデータが足りない場合, 開発プロセス, 開発対象が似ている別プロジェクトの製品のデータを混ぜて学習させることで, 精度を上げられることがわかった. 一方で, 学習に使用していないプロジェクトの予測へは適用できなかったため, さらなるモデルの改良や別の手法の適用を考える必要がある.

今後の課題として, 他プロジェクトや, ユースケースベースの開発以外でのプロジェクトでの検証を行うことが考えられる. 本研究ではユースケースベースの開発以外の仕様書を読み込ませ, フォールト数の予測ができるかは検証していないため, より汎用的に使える手法かの検証をしていく必要があると考えている.

また, 本手法では仕様書のみからのフォールト数の予測を行った. そこで, コードメトリクスや開発履歴[13]の使用していないデータを導入し検証することや, 仕様書等のドキュメントがない場合にでも適用できるフォールト数予測手法を考案していく必要があると考えている.

参考文献

[1] 橋本 弥一郎, 平島 俊一, 古賀 恵子, 横山 陽一, 森 文彦, ソフトウェア品質評価システム"SQE”, *日立評論*, vol. 68, No. 5(1986-5), pp.55-58 , 1986.

[2] Ohlsson, N., Alberg, H. Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*. 886-894. 1996.

[3] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 2278-2324. 1998.

[4] Khoshgoftaar, T. M., Allen, E. B., Deng, J. Using regression trees to classify fault-prone software modules. *IEEE Transactions on Reliability*. pp. 455-462. 2002

[5] Ostrand, Thomas J., Elaine J. Weyuker, Robert M. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*. pp. 340-355. 2005

[6] Kim, S., Whitehead Jr, E. J., Zhang, Y. Classifying software changes: Clean or buggy?. *IEEE Transactions on Software Engineering*. pp. 181-196. 2008.

[7] T. Zimmerman, N. Nagappan, L. Williams. Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista. In *Proceedings of the 2010 Third International Conference on Software Testing , Verification and Validation (ICST '10).* pp. 421-428. 2010.

[8] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., Dean, J., Distributed Representations of Words and Phrases and their Compositionality. In *Advances in neural information processing systems*. pp. 3111-3119, 2013.

[9] Sakai, N., Nakanishi, M., Yokouchi, H., Improving User Experience in a Large-scale Software Development Project. In *Proceedings of 6th World Congress for Software Quality (6WCSQ)*. pp. 106-116. 2014.

[10] Kim, Y. Convolutional Neural Networks for Sentence Classification. *arXiv preprint arXiv:1408.5882*. 2014.

[11] Kingma, D., Ba, J., Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2014.

[12] 安藝優子, 野中誠. 摘出済み欠陥を考慮したレイリーモデルに基づくソフトウェア欠陥予測手法. In *経*

営情報学会 全国研究発表大会要旨集 経営情報学会 2010 年秋季全国研究発表大会. pp. 27-27. 2010.

[13] 畑秀明, 水野修, 菊野亨. 不具合予測に関するメトリクスについての研究論文の系統的レビュー. *コンピュータ ソフトウェア*, pp.106-117. 2012

[14] Scandariato, R., Walden, J., Hovsepyan, A., Joosen, W. Predicting Vulnerable Software Components via Text Mining. *IEEE Transactions on Software Engineering*. pp. 993-1006. 2014.

[15] 中野大輔, 門田暁人, 亀井靖高, 松本健一. バグモジュール予測を用いたテスト工数割り当て戦略のシミュレーション. *コンピュータ ソフトウェア*. pp. 118-128. 2014.

[16] 田中公司, 生駒卓志, 津田和彦. ワイブル分布を用いたソフトウェア欠陥予測手法の検討. In *経営情報学会 全国研究発表大会要旨集 2015 年秋季全国研究発表大会* . pp. 21-24. 2015

[17] Meng, Q., Zhang, B., Feng, C., Tang, C. Detecting Buffer Boundary Violations Based on SVM. In *Information Science and Control Engineering (ICISCE), 2016 3rd International Conference on*. pp. 313-316. 2016.

[18] Kaur, I., Bajpai, N. An Empirical Study on Fault Prediction using Token-Based Approach. In *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*. P.32 2016.

[19] ソフトウェア開発データ白書 2016-2107. 独立行政法人情報処理推進機構(IPA) 技術本部 ソフトウェア高信頼化センター(SEC). 2016

[20] MeCab: Yet Another Part-of-Speech and Morphological Analyzer, (http://taku910.github.io/mecab/)

事例報告：AADL を用いた STAMP/STPA 支援

岡本圭史，力武克彰
仙台高等専門学校
{okamoto, yoshiaki}@sendai-nct.ac.jp

大友楓雅
仙台高等専門学校情報電子システム工学専攻
a1602006@sendai-nct.jp

要旨

本稿では，アーキテクチャ記述分析言語 AADL による安全分析手法 STAMP/STPA 支援事例として，特にハザード誘発要因識別の支援事例を報告する．

1. はじめに

複雑化したシステムの安全分析に対応するため，システム理論に基づく事故モデル STAMP(Systems-Theoretic Accident Model and Process)及び STAMP に基づく安全分析手法 STPA(STAMP based Process Analysis)が提唱された[1]. STPA では，ハザードを識別，コントロールストラクチャー（Control Structure，CS）を構築し(Step1)，それを基に，コンポーネント間の非安全制御動作(Unsafe Control Action，UCA)を識別し(Step2-1)，得られた UCA の誘発要因（Causal Factor，CF)を識別する(Step2-2).

STAMP/STPA では分析者の発想に重きを置くことで，従来法では気づきにくい要因に気付きやすくしている．他方 AADL(Architecture Analysis and Design Language)を用いて，人と機械支援の相補的分析の提案[2]や，STPA の一部自動化[3]が行われている．そこで，AADL を用いた STAMP/STPA 支援法の提案を目標とし，AADL による STAMP のモデル化と STPA Step2-2 支援の事例を報告する．

2. STAMP/STPA と AADL

AADL はアーキテクチャ記述・分析言語であり，AADL モデルにアノテーションを追記することで分析が可能になる．安全分析用アノテーションを定義した拡張に Error Model Annex があり，安全分析をツール実行できる．

AADL と STAMP はともにコンポーネントベースであるため，STAMP の CS は AADL モデルとして記述できる[2,3]. また STPA Step2-1 のタイプは Error Model Annex のエラータイプを用いて定義できる．[2]では，Fault Impact Analysis(FIA)を用いたハザード分析を行っている．また[3]では，手動分析と自動分析を組み合わせることで分析結果を充実させるというアプローチを提案している．本稿では，[2]の分析法を STPA の手順に組込み，[3]の提案を試みる．

3. 事例：AADL を用いた CF 識別支援

踏切への STPA 適用事例[4]を参考に CS を AADL モデルとして記述した．さらに FIA を用いて UCA 候補や CF 候補の影響範囲を指摘し，その結果を参考に STPA を実施した．なお FIA を実施するために，Step1 のタイプをエラーソースとして，コンポーネント内やコンポーネント間のタイプ伝搬をエラー伝搬として，AADL モデルにそれぞれ追記した．

STPA Step2 支援に FIA を用いることで，分析対象 UCA を引き起こす CF を識別できた．具体的には，CF の影響範囲に分析対象 UCA が含まれることをツールにより指摘できた．また，CF が別のコントロールループの UCA を引き起こすことも指摘できた．

以下の課題を確認した．1）分析項目に応じた AADL モデル要素の詳細化が必要である．2）Step2-1 支援を実施するには，ハザードへの到達判定が必要となる．しかしハザードはシステムの状態であるため，モデルの改善が必要である．3)FIA は単一要因からの影響範囲のみ指摘可能であるため，複合要因の識別には FIA 以外あるいはモデルの改善が必要となる．

参考文献

[1] Leveson, N. Engineering a Safer World: Systems Thinking Applied to Safety. MIT Press, 2011
[2] Procter, S. and Hatcliff, J. An Architecturally-Integrated, Systems-Based Hazard Analysis for Medical Applications, Formal Methods and Models for Codesign (MEMOCODE), 2014
[3] Feiler, P. et al., Improving Quality Using Architecture Fault Analysis with Confidence Arguments, 2015
[4] システム安全性解析 WG，はじめての STAMP/STPA，情報処理推進機構，2016

ゴール指向分析ＫＡＯＳにおける依存性を考慮した要求抽出法の考察 -酒屋倉庫問題の場合-

岡野道太郎
筑波大学大学院ビジネス科学研究科
okano@gssm.otsuka.tsukuba.ac.jp

中谷 多哉子
放送大学 情報コース
tinakatani@ouj.ac.jp

要旨

本稿は，要求仕様書に書かれるべき，設計・実装・テストに必要な情報を獲得するためにはどのような手法を用いればよいかの知見を得ることを目標とする．この目標へのアプローチとして，ゴール指向要求分析の一手法である *KAOS* を用いる．この手法はシステムの目標であるトップゴールから，ゴールの詳細化を繰り返すことにより要求を獲得する．本稿ではゴール $P \rightarrow Q$ の詳細化手法として，状態遷移→部分を詳細化するマイルストーン分解と，状態 Q を分解する要素分解を用いる．その際，状態 P については分解していないが，状態 P を分解する必要があるかについて考察した．考察を行うために「酒屋倉庫問題」のゴール分解を行った．すると，状態Ｐを分解しないと依存性に関して問題が生じることが判明した．そこで本稿では状態Ｐの部分を分解し，依存性を明記する表記法を提案する．関連研究では本稿の手法と他の要求抽出手法との関連について議論し，今後の研究において本稿で取り上げた達成ゴール以外のゴールについて述べる．

1. はじめに

ソフトウェア開発は，要求分析の後，設計・実装・テストの各工程が続く．そのため要求分析では，設計・実装・テストに必要な情報が獲得される必要がある．この「設計・実装・テストに必要な情報」は何であり，その情報を要求分析の工程で獲得するには，どのような手法を用いればよいかの知見を得ることが本稿の目標である．この目標へのアプローチとして，筆者はゴール指向分析の一手法である KAOS[1] を用いる．KAOS については２章で詳述するが，簡潔に述べると，KAOS はシステムの目標であるトップゴールからゴールの詳細化を繰り返すことで要求を獲得する手法である．我々の今までの研究ではゴールを「状態 P から状態 Q への状態遷移」とし，詳細化手法として，状態遷移を詳細化するマイルストーン分解 [1] と，状態 Q を詳細化する要素分解 [2] を利用してきた．つまり，状態 P は詳細化しなかったが，詳細化する必要がないのだろうかということについて，本稿では考察する．その結果，分解を行わないと依存性に関して問題が生じ，設計に必要な情報が欠けることが判明したので，依存性に対する提案を行い,「酒屋倉庫問題」の事例を用いて提案を評価する．

本稿の構成を以下に述べる．まず第二章で用語の説明を行う．第三章で，上述した状態Ｐの分解について「酒屋倉庫問題」を利用して考察する．第四章では，考察時に生じた問題を解決する手法を提案し，その手法を酒屋倉庫問題に適用する．第五章では，提案手法について考察する．第六章では関連研究として，本稿と i*[3],UML[4] の関連について述べる．第七章では今後の研究について述べ，第八章でまとめる．

2. 用語の説明

2.1. KAOS について

KAOS は，ゴールを要求獲得に活用する，ゴール指向要求分析の一手法である．はじめにシステムが満たすべきゴールをトップゴールとして１つ挙げ，そのゴールを詳細化して，いくつかのゴールに分解し（この「詳細化により分解されたゴール」をサブゴールと記す），サ

ブゴールを再度詳細化してサブゴールのサブゴールへ分解するというように，詳細化を繰り返す．すると，トップゴールでは，多くの人が関わり，実現に複雑な過程を経るとしても，最終的には 1 人あるいは 1 システム（これをエージェントと呼ぶ）が満たせる程度のゴールとなる．このとき分解を終了し，末端ゴールをエージェントに対する要求とする．このようにＫＡＯＳを用いれば，要求はゴールの詳細化によって抽出できるが，そもそも，「ゴールとは何か,「設計・実装・テストに必要な情報は要求だけなのか」,「詳細化はどのように行うのか」については説明していない．そこで，まず,「設計・実装・テストに必要な情報は要求だけなのか」について説明する．

2.2. 設計・実装・テストに必要な情報と要求の関係

「設計・実装・テストに必要な情報」は，要求仕様書に文や図としてまとめられる．図は文章で表現することも可能なので,「設計・実装・テストに必要な情報」は文で表現することが可能であるとみなすことが出来る．そして文を Lamsweerde は図 1 のように分類している [1].

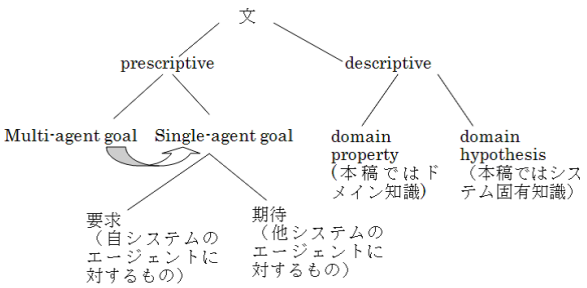


図 1. 文の分類（[1] を元に作成）

文は意図を含むか否かにより，prescriptive な文と，descriptive な文に分けられる．本稿では前者を規範的な文，後者を記述的な文と記す．記述的な文は，自然法則のようなシステム開発者の意図とは関係なく従わなければならない domain property と，本システム開発上，そのようになっているという（例えば，本社は東京にあるなど）domain hypothesis がある．本稿では，前者をドメイン知識，後者をシステム固有情報と記す．そして規範的な文はゴールであり，このゴールは多くの人が関わる Multi–agent goal から，詳細化を繰り返すことにより，一人または一システム（本稿ではこれらの人とシステムをまとめてエージェントと記す）が実現できる Single–agent goal に分解される．この Single–agent goal は，対象システムのエージェントが実現する場合「要求」となり，それ以外のエージェントが実現する場合「期待」となる．

つまり，文は意図を含むか否かにより，記述的な「ドメイン知識」,「システム固有情報」と，規範的な「要求」,「期待」とに分かれる．よって,「設計・実装・テストに必要な要求仕様書で表現されるべき文」は，対象システムに関する文なので（「期待」を除いた）「要求」,「ドメイン知識」,「システム固有情報」であるといえる．すなわち要求以外にも必要な情報はあるが，少なくとも要求は必要であると言えるので，まず要求について分析する．

そして「要求」はゴールを分解することによって得られる．次に「ゴールとは何か」について説明する．

2.3. ゴールとは何か

ゴールとは，Lamsweerde[1] によると,「エージェントの協調によって満足すべき意図を示した規範的な文」であるとしている．ゴールにはいくつかのタイプがある（詳細は「６．今後の研究」にて述べる）が，本稿では達成ゴールを取り扱う．達成ゴールは，振る舞いを表したゴールの一種で，状態遷移の連続であると [1] では定義している．よって，達成ゴールは「状態 P から状態 Q へ状態への遷移が完了している」のように記述できる．このような状態遷移を本稿では P → Q と表現する．P が状態変化が起こる前の状態（本稿では事前状態と呼ぶ），Q は状態変化が起こった後の状態（本稿では事後状態と呼ぶ），→は P から Q へ状態が遷移する過程について表している．なお達成ゴールを [1] では，様相論理を用いて P →◇ Q で表現している．本稿ではそこまで厳密な定義をせず，ある状態から別の状態に遷移すれば，その状態遷移を P → Q と表現し，それを達成ゴールとする．

よってトップゴールも達成ゴールの形式で書かれる．そして詳細化すると最終的に要求が達成ゴールの形で得られるが，どのように詳細化したら，要求が得られるかについては説明していない．そこで次に詳細化の方法について述べる．

2.4. 詳細化はどのように行うのか

達成ゴール P → Q の詳細化を，本稿では 2 つの方法で行う．一つは P → Q のうち，状態遷移（→）を分解

するマイルストーン分解であり，もう一つは Q の部分を分解する要素分解である．

マイルストーン分解は [1] で述べられている．マイルストーン分解とは，P から Q への状態遷移が起こる過程において，マイルストーン M の状態を経るのであれば P → M と M → Q に分解できるとする分解法である．

要素分解 [2] は，P から Q への状態遷移が起こったとき，事後状態 Q がいくつかの要素 $\{Q_1, Q_2, \cdots Q_n\}$ から成り立つとき，状態遷移 P → Q を事前状態 P，事後状態を各要素として分解する方法である．すると，P → Q_1,P → $Q_2, \cdots$ P → Q_n に分解できるが，それだけでは $\{Q_1, Q_2, \cdots Q_n\}$ という各要素が存在する状態になるだけで，それが結合した状態にならないので，$\{Q_1, Q_2, \cdots Q_n\}$ → Q という状態遷移も付加することもあるとする分解法である．例えば,「お皿の上にスパゲティが盛り付けてある」を分解するとき,「お皿がある」「スパゲティがある」だけでは，お皿の上にスパゲティがあるかどうかわからない．そこで「お皿とスパゲティがある状態から，スパゲティをお皿の上に盛り付けられている」というゴールを追加して分解する．なお要素分解における要素は，分解時に着目する視点によって様々に変わる．例えば，ユースケースに着目して，ユースケースごとに分解することも考えられるし，構成部品に着目し，部品ごとに分解することも考えられる．さらに機能に着目し，機能ごとに分解することもあり得る．

2.5. 詳細化と本稿での考察との関係

我々は既に，達成ゴールをマイルストーン分解と要素分解を用いて詳細化し，要求を導き出し，それとドメイン知識・システム固有情報をもとに実装が行えるものがあることを，電気工事の事例により示した [2]．

しかしここで疑問がある．状態遷移→と状態 Q は分解したが，事前状態の P は分解する必要はないのだろうか．本稿ではその事前状態 P の分解について考察する．

3. 事前状態 P の分解の必要性 「酒屋倉庫問題」による考察

酒屋倉庫問題とは，情報処理 [5] に記載されている「ある酒類販売会社」（以下酒屋と記す）における「受付係の仕事（在庫なし連絡，出庫指示書作成，在庫不足リスト作成）のための計算機プログラムを作成する」という問題である．酒屋倉庫問題における本稿に関係のある事項についてのみ，以下に記載する．

- 酒屋について
  - 酒屋には倉庫係，受付係がある
- 入庫について
  - 倉庫係はコンテナ入りの酒を倉庫に保管し，積荷票を受付係へ渡す
  - 積荷票の項目はコンテナ番号と搬入年月，日時，内蔵品目と数量の繰り返しである
- 出庫について
  - 受付係は出庫依頼を受ける
  - 出庫依頼を基に倉庫係へ出庫指示書を出す
  - 出庫指示書の項目は注文番号，送り先名，コンテナ番号，品名，数量である
  - 出庫依頼の項目は，品名，数量，送り先名である
  - 在庫がない場合は依頼者に電話連絡し，在庫不足リストに記載する
  - 在庫不足リストの項目は，送り先名，品名，数量である
  - 空になる予定のコンテナを倉庫係に知らせる（出庫指示書に記載）
  - 倉庫内のコンテナ数はできる限り最小にする
  - 出庫指示書の項目は，注文番号，送り先名，コンテナ番号，品名，数量，空コンテナ搬出マークである

なお,「在庫なし連絡」のための計算機プログラムとは，本稿では，以下の項目の「在庫なし連絡票」を作ることとする．

- 在庫なし連絡票：注文番号，送り先名，品名，注文数量，不足数量

ここで,「帳票出力が行われている」というゴール分解を行うと，図 2 のような分解を一例として挙げることができる．これ以外の分解方法も可能ではあるが，本稿では問題点が明示しやすいため，この分解を元に議論を進める．

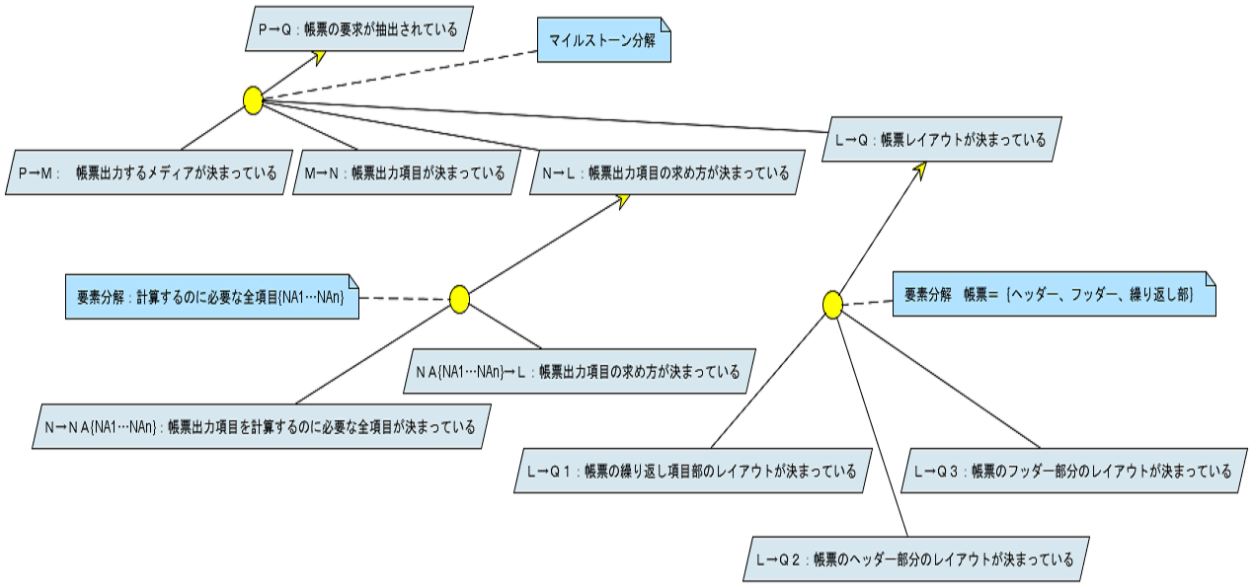


図 2. 一般的な帳票作成のゴール分解

ここで図 2 の表記法について述べる．この表記は KAOS を基にしている．すなわち，始めにトップゴールを挙げ，トップゴールをマイルストーン分解か要素分解を行ってサブゴールを挙げる．この際，マイルストーン分解を行ったか，要素分解を行ったかについては，注釈として記述している．さらに各サブゴールに対し，マイルストーン分解または要素分解を適用して詳細なサブゴールを導いている．このとき図 2 のように，ゴールとサブゴールの間にある円と，各サブゴールが線で結ばれている場合がＡＮＤ分解，ゴールから各サブゴールに線が出ている（この場合，１本の線の間に１つの円がある）場合がＯＲ分解である．ＡＮＤ分解は，サブゴールの全てが成立しないとゴールが成立しないものであり，ＯＲ分解は，サブゴールの一つが成立すればゴールが成立するものである．ゴールの記号の内部に書かれている文字列をラベルと呼ぶ．本稿では，ラベルの前半にＰ→Ｑ等，状態遷移が始めに記されている．そして，その後に「帳票の要求が抽出されている」などの状態が記されている．つまり，ラベルの前半にゴールの状態遷移，後半に最終状態Ｑの状態を表すものとする．したがって，状態Ｐの状態はＰを最終状態に持つゴールのラベルに記されている．本来，最終状態が同じゴールの場合，ラベルの後半も同じ表現になるはずであるが，同じ状態を指しても視点により表現が異なる場合がある．このため同じ最終状態のゴールでもラベルの表現が違うことがある．

図 2 に対し，帳票が「在庫なし連絡票」,「出庫指示書」,「在庫不足リスト」であることをトップゴール直下のサブゴールにそれぞれ適用すると，図 3 のようになる．

ここで「Ｎ→Ｌ１：在庫なし連絡票の出力項目の求め方が決まっている」について議論する．このゴールは事前状態Ｎ，すなわち「帳票出力項目が決まっている」を満たしている必要がある．このゴールで示す「帳票」とは「在庫なし連絡票」,「出庫指示書」,「在庫不足リスト」の３つを指し，これらすべてが完了していないと事前状態Ｎを満たしていないことになる．しかし，Ｌ１の状態「在庫なし連絡票の出力項目の求め方が決まっている」に遷移するためには「Ｍ→Ｎ１：在庫なし連絡票の出力項目が決まっている」は満たす必要はあるが,「Ｍ→Ｎ２：出庫指示書の出力項目が決まっている」を満たす必要は必ずしもない．つまり，現状の表記法ではＮ→Ｌ１を実行するためには，Ｍ→Ｎ１，Ｍ→Ｎ２，Ｍ→Ｎ３が必要となってしまっているが，Ｍ→Ｎ２，Ｍ→Ｎ３は必ずしも必要ではなく，その分，成立条件が厳しくなってしまっている．この問題を以下「問題１」と記す．

また「Ｌ→Ｇ１：在庫なし連絡票のレイアウトが決まっている」について議論すると，在庫なし連絡票のレイア

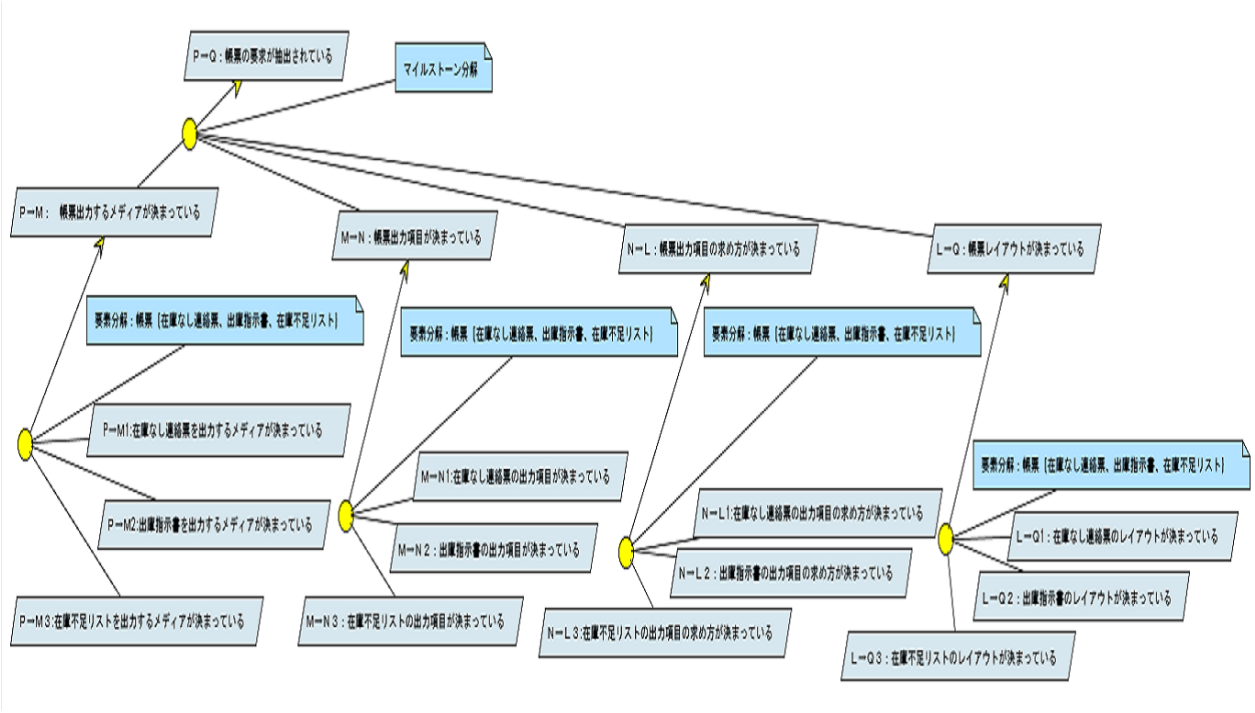


図 3. 酒屋倉庫問題のゴール分解

ウトを決めるためには,「在庫なし連絡票の出力項目が決まっている」とともに,「在庫なし連絡票を出力するメディアが決まっている」必要がある（ここでメディアとはＡ４の大きさのＰＤＦ等，出力先を示す）．そこで，Ｌ→Ｇ１のゴールの事前状態としては,「Ｎ→Ｌ１：在庫なし連絡票の出力項目の求め方が決まっている」と「Ｐ→Ｍ１：在庫なし連絡票を出力するメディアが決まっている」の両方が満たされなければならない．ただし,「Ｐ→Ｍ１：在庫なし連絡票を出力するメディアが決まっている」はレイアウトを決める直前までに決まっていればよいのであって,「Ｎ→Ｌ１：在庫なし連絡票の出力項目の求め方が決まっている」と同時並行で行ってもかまわない．一般的に出力先等，大まかな事項を決定してから出力項目等の詳細項目を決めるので，Ｐ→Ｍ，Ｍ→Ｎ，Ｎ→Ｌという遷移にしただけであり，順序は必須ではない．ところが図の表記法だと，同時並行を認めないように見えてしまい，その分，事前状態の成立条件が厳しくなってしまっている．この問題を以下「問題２」と記す．

これら問題１，２は依存性の問題といえる．本稿において依存性とは，ゴールＰ→Ｍ，Ｍ→Ｑに対し，Ｍ→Ｑを達成するには事前状態Ｍが成立しなければならず，Ｍが成立するためにはＰ→Ｍが成立しなければならない場合，ゴールＭ→Ｑが達成するか否かは，ゴールＰ→Ｍが達成するか否かの結果に影響されるといえる．このように，ゴールＭ→Ｑの事前状態の成立が，ゴールＰ→Ｍの達成結果による場合，ゴールＭ→ＱはゴールＰ→Ｍに依存しているといい，この関係を依存性という．

問題１は事前状態Ｐを分解しないことにより生ずる問題であり，事前状態の範囲，すなわち依存性の範囲を広く取りすぎている問題である．問題２は事前状態を分解したり，並行処理を認めたりした場合，それを認めなければ必ず成立していたが，認めることにより成立するかどうか確かではなくなる可能性がある場合の問題である．

依存性は,「設計・実装・テストに必要な情報」である．なぜならば，もし依存性の情報がなく要求Ｐ→Ｑが成立しているかテストを行おうとすると，事前状態Ｐが成立していない状態でテストを行う可能性もあり，その場合正しく実装されていても，テストに失敗する．したがってテストを行う者にとって，テスト対象の依存性は必要な情報といえる.「設計・実装・テストに必要な情報」を獲得することが本稿の目的なので，依存性の問題を解決しないと，本稿の目的は達成できないことになる．そこ

で，問題１，２の解決法を次の章で提案する．

4. 提案手法

ここでは，以下の表記法を提案する．
ゴールＰ→Ｑにおいて

- （問題１解決のための表記法）事前状態Ｐがいくつかの要素｛Ｐ１，Ｐ２，Ｐ３ … Ｐｎ｝に分解可能で，それらの要素の一部が成立すれば事前状態が成立したとみなせる場合，事前状態Ｐの後にっをつけ，その要素を列挙することにより，その列挙した状態が成立すれば，事前状態が成立したとみなす．例えば，事前状態Ｐが｛Ｐ１，Ｐ２，Ｐ３｝に分解可能で，Ｐ１を満たせば事前状態が成立したとみなせる場合は，Ｐっ Ｐ１→Ｑと記述する．

- （問題２解決のための表記法）事前状態の分解を認めなければ成立した状態で，事前状態の分解を認めることにより，成立するかどうかが確かではない場合や，その成立を明示したい場合は，事前状態Ｐの後に＞をつけ，その状態を列挙することにより，その列挙した状態が成立しなければ，事前状態が成立しないものとすることができる．例えば，Ｐ→Ｑが，Ｐ→Ｍ，Ｍ→Ｑと分解でき，Ｐが｛Ｐ１，Ｐ２，Ｐ３｝，Ｍが｛Ｍ１，Ｍ２，Ｍ３｝，Ｑが｛Ｑ１，Ｑ２，Ｑ３｝に分解可能な時，Ｍっ Ｍ１→Ｑ１の事前状態Ｍ１が成立する際に，Ｐ１，Ｐ２が成立する必要がある時（Ｐ３は必須でない）Ｍっ Ｍ１，Ｍ＞｛Ｐ１，Ｐ２｝→Ｑ１と記述する．

この表記法を基に図 3 を分解すると，図 4 のようになる．

本提案によって変わった点は，ゴールラベルの前半に記述される状態遷移を表す部分である．

- Ｍ→Ｎのサブゴールにおいて，このゴールはＰ→Ｍと並行して行っても問題を生じないため，Ｍ＞Ｐ→Ｎ１等，Ｐの時点すなわちはじめから実行可能であることを明記した．

- Ｎ→Ｌのサブゴールにおいて，Ｎは，それぞれの帳票に対する事前状態が成立していればよいので，Ｎっ Ｎ１→Ｌ１等，該当する帳票の状態のみを事前状態に挙げるようにした

- Ｌ→Ｑのサブゴールにおいて，このときまでに，各帳票の出力項目とその求め方だけでなく，メディアも決まっていないといけないため，Ｌっ Ｌ１等の他にＬ＞Ｍ１等も加え，Ｌっ Ｌ１，Ｌ＞Ｍ１→Ｑ１等とした

なお，事前状態を分解したとき，その分解した状態が，他のゴールの最終状態として存在するときは，その状態名を使う．たとえば，Ｎっ Ｎ１→Ｌ１の分解された事前状態Ｎ１は，Ｍ＞Ｐ１→Ｎ１の最終状態Ｎ１と同じ状態なのでＮ１という状態名を使っている．もし分解された事前状態が，他のゴールの事後状態にも表れず，初期状態Ｐでもない場合は，別の状態名をつけ，その状態の内容を注釈に記載する．

5. 提案手法に対する考察

まず，この提案した表記法により，問題が解決したかを確認する．

- 問題１については，「在庫なし連絡票の出力項目の求め方が決まっている」のゴールはＮっ Ｎ１→Ｌ１となり，このゴールは在庫なし連絡票以外には依存しなくなっている．すなわち，依存する範囲を的確に示していることなり，問題は解決した

- 問題２については，「在庫なし連絡票のレイアウトが決まっている」のゴールはＬっ Ｌ１，Ｌ＞Ｍ１→Ｑ１となり，このゴールは在庫なし連絡票のメディア (出力先等）と出力項目の求め方に依存していることを明確に示している．すなわち，依存する範囲を的確に示していることなり，問題は解決した

つまり，問題１，２は本提案の表記法により適切になり，依存性が的確に示せるようになった．本稿のゴール分解手法は KAOS を基にしているが，KAOS では依存性が明白に示せなかったことは，問題１，２が生じたことからもわかる．新たに本稿が提案する記法を導入することによって，依存性が明確に記述できるようになった点に本稿の新規性，有用性がある．

ただし，本稿では事前状態Ｐを分解しているが，その際，分解の妥当性は保障していない．つまり，事前状態ＰをＰ１，Ｐ２に分解した場合，Ｐ１は，本当に事前状態Ｐに含まれているかということを保障するのは，その分解を行った者の責任である．

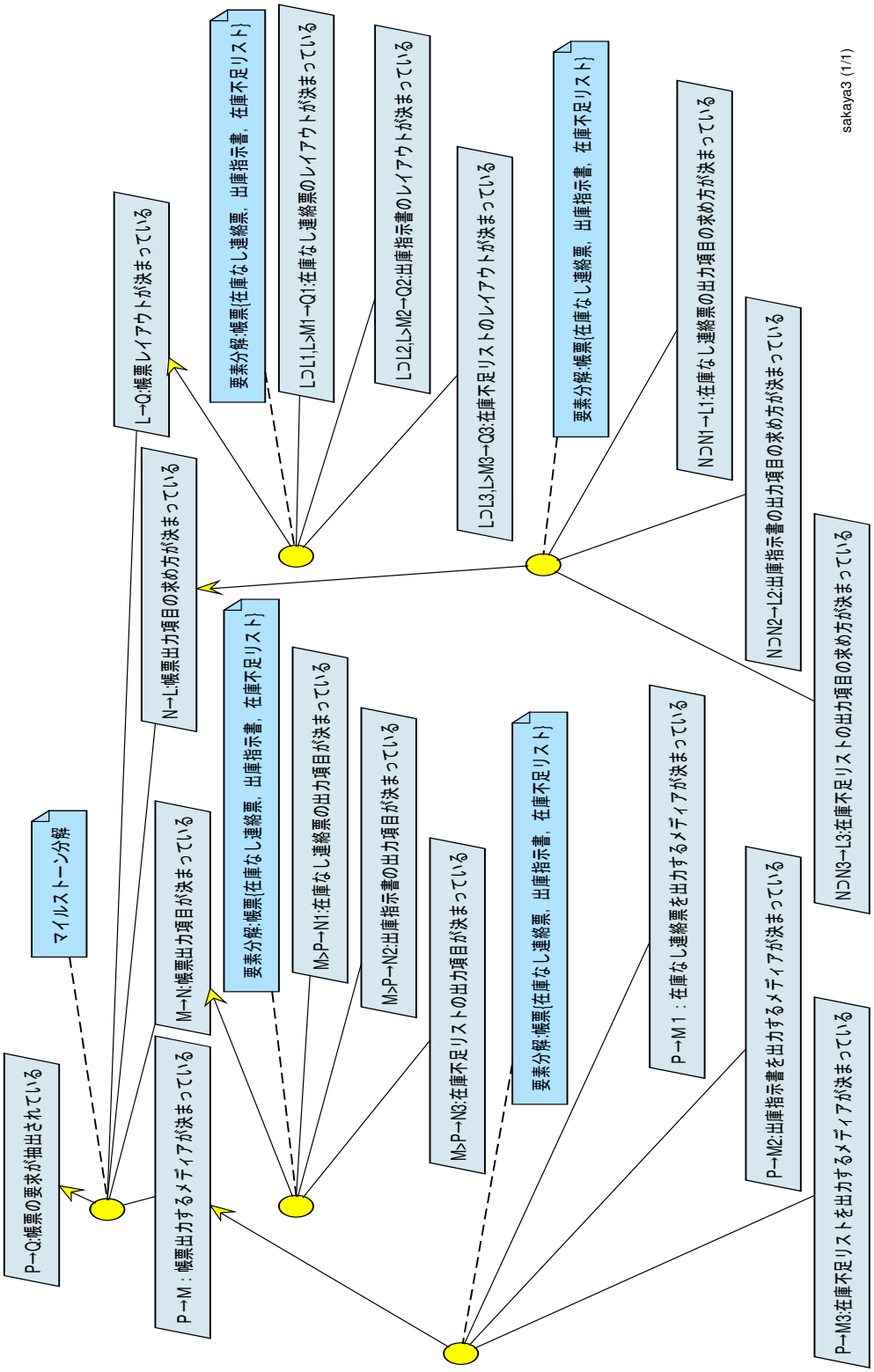


図 4. 提案手法による酒屋倉庫問題のゴール分解

本稿で行っているマイルストーン分解は，状態遷移に基づく分解であるが，状態遷移はプロセスによって実現されるので，プロセス分解を行っているといえる．また，要素分解は，最終状態の構成要素に着目して分解を行っているが，それは構造を分解していることになる．そして,「酒屋倉庫問題」の事例では，マイルストーン分解と要素分解の両方を用いている．したがって，今回の事例はプロセス分解のみを行っているのでも，構造による分解のみを行っているのではなく，両方の分解を適切に切り替えることによって分解が実現できているといえる．一般にシステムは，一連の情報処理プロセスが続いて実行される．この際，各プロセスの入出力データにはデータ構造が存在する．よってシステムを分析するには，プロセスの分析と各プロセスの入出力データのデータ構造分析の両方が行える必要がある．マイルストーン分解と要素分解を適切に組み合わせることにより，このプロセス分析とデータ構造分析が適切に行える．

次の章では，この依存性とプロセス分解を表現する振る舞い図や構造による分解を表現する構造図について，既存の他の研究と本稿の関係について議論する．

6. 関連研究

本稿はゴール指向要求分析のＫＡＯＳを基にしている．ゴール指向要求分析では，この他に i*[3] がある．i* では，ゴールとアクター間の依存性をＤの記号を用いて表すことにより明示している．ＫＡＯＳでは要求をエージェントに割り付けることにより，要求 (ゴール）とアクターを含むエージェントの関係を明示することができる．さらに本稿の研究により，要求（ゴール）間の依存性も明記できるようになった．i*は KAOS のようにトップゴールと要求の関係を明示できないので，本稿の表記法は，i*で表記できる依存性に加え，トップゴールからの関係もゴールの詳細化により明示できる点で i*よりも多くの情報を表記できるといえる．ただし，as–is 分析の場合，各エージェントが行っているゴールがトップゴールを意識しているかどうかは定かではないため，この場合，トップゴールを明示しない i*のほうが描きやすい可能性はある．しかし to–be 分析においては，各エージェントのゴールはトップゴールに貢献すべきであるので，トップゴールから各自のゴール，要求との関係が示せる本稿の表記法のほうが望ましいといえる．

また，本稿ではマイルストーン分解と要素分解によりゴール分解を行っている．マイルストーン分解は，振る舞いゴールである達成ゴールを分解していることから，振る舞いの分解といえる．そして本稿の記述法により振る舞いゴール間の依存性を示すことができるようになった．そこで，依存性を辿っていくことにより，一連の振る舞いの流れを辿ることができる．このような振る舞いの流れを記述した図が「振る舞い図」であり，ＵＭＬ [4] においては，アクティビティ図等が，振る舞い図に相当する．

要素分解は最終状態を各要素に分解している．その際，最終状態のデータ構造に着目し，データを構成する各要素に分解して，データ構造を表すことも出来る．そのような構造を表す図には，ＥＲ図やＵＭＬのクラス図等の構造図がある．

つまり本稿の表記法は，構造図と振る舞い図の両方を１つの図の中に統合している．実際の要求分析は，本稿の酒屋倉庫問題の事例で示したように，プロセス分析と構造分析が適度に切り替えられて行われる．したがってこの分析結果は別々の図ではなく，統合した１つの図に書かれることによって，その関係性が明示できる．その点で，本稿の表記法は優れている．ただし，実際にデータベースのテーブルを定義するときは，プロセスは不要であり，データベースの構造のみが必要となる．そのような場合は，テーブル構造のみを明記できるＥＲ図がふさわしい．

関連研究と本稿の関係についてまとめると，本稿はＫＡＯＳに新たな表記を加えることにより，依存性が明記できるようになり，これとマイルストーン分解，要素分解により，依存性，振る舞い，構造を１つの統合した図で示せるようになった．前章で述べたとおり，システムを分析するには，プロセスの分析と各プロセスの入出力データのデータ構造分析の両方が行える必要がある．プロセスの振る舞いと依存性，入出力データの構造を１つの図で示せる点で，本稿の提案は優れており，有用性はある．ただし，場合によっては，それらのうちの一部の情報のみが必要な場合があり，その場合には，依存性を示す i*や振る舞いを表すＵＭＬの振る舞い図 (アクティビティ図等)，構造を示すＥＲ図やＵＭＬの構造図（クラス図等）のほうがふさわしい．

7. 今後の研究

本稿では達成ゴールのゴールの詳細化について議論した．しかしゴールには達成ゴール以外のゴールもある．ゴールは，図 5 に示すように分類される [1]．すなわち，

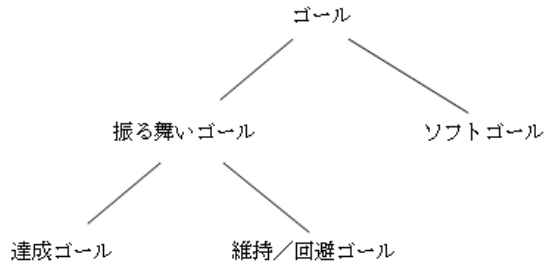


図 5. ゴールの種類（ゴールタイプ）（[1] を元に作成）

まずはじめにソフトゴールと振る舞いゴールに分かれ，振る舞いゴールは本稿で取り上げた達成ゴールの他に，維持ゴールや回避ゴールがある．ソフトゴールは,「レスポンスが 3 秒以内にあること」のような望ましい状態を示したゴールであり，非機能要求の中には，ソフトゴールで表現できるものもある（セキュリティ等は回避ゴールで示すものもある）．また，達成ゴールは最終状態である「よいことがいつかは起こる」形のゴールであるが，回避ゴールは「悪いことは起こらない」という形のゴールであり,「不正アクセスが起こらない」といったセキュリティ要求を表現することができる．維持ゴールは「よいことが起こり続ける」であり,「システムが稼働し続けている」という運用・保守に関する要求を表現することができる．非機能要求やセキュリティ要求，保守・運用の要求は，設計上必要な情報なので,「要求仕様書に書くべき内容」であるが，本稿ではそれらについて触れられていない．今後研究する必要がある．その際に，達成ゴールと i*,UML 等は関連があったが，他のゴールとの関連性はどうなのかについても研究する必要がある．達成ゴール以外のゴールもＫＡＯＳでは表現できるが，もし i*やＵＭＬでは表現できない場合，ゴール抽出をＵＭＬのみで行うと，要求仕様書に書くべき要求が抜けることになるからである．

また，本稿では酒屋倉庫問題の一部しか取り上げていない．酒屋倉庫問題のゴール分解を完成させ，トップゴールから要求が抽出できることを確認する必要がある．我々は [6] において，酒屋倉庫問題は，以下の点が抜けていることを指摘した．

- 注文番号は出庫指示書の出力項目として挙げられているが，どのように生成されるか明示されていない
- 積荷票からどのように出庫を割り当てるか，明示されていない

ゴール分解中に，これらの点が抜けていることを発見できるか確認することも必要となる．

8. まとめ

本稿は，要求仕様書に書かれるべき，設計・実装・テストに必要な情報を獲得するためにはどのような手法を用いればよいかの知見を得ることを目標とした．要求仕様書に書かれるべき情報は，要求，ドメイン知識，システム固有情報であるが，本稿ではこのうち要求を獲得する手法について対象とした．要求を獲得するアプローチとして，ゴール指向要求分析の一手法である KAOS を用いた．この手法はシステムの目標であるトップゴールから，ゴールの詳細化を繰り返すことにより要求を獲得する．本稿ではゴール Ｐ → Q の詳細化手法として，状態遷移→部分を詳細化するマイルストーン分解と，状態 Q 部分を分解する要素分解を用いた．その際，状態Ｐについては分解していないが，状態Ｐを分解する必要があるかについて考察した．考察を行うために「酒屋倉庫問題」のゴール分解を行った．すると，Ｐの部分を分解しないと，依存性に関して問題が生じることが分かった．そこで本稿では状態Ｐの部分を分解し，依存性を明記する表記法を提案し，酒屋倉庫問題で確認した．本提案により，依存性，振る舞い，構造を 1 つの統合した図で示せるようになった．既存の手法でも i*は依存性，ＵＭＬは振る舞いや構造を示せるが，依存性・振る舞い・構造の関連を 1 つの図で示せる点で，本稿の提案は優れており，有用性がある．本稿で取り上げなかったゴールから，非機能要求やセキュリティ要求，保守・運用の要求が抽出できる．これらも設計･実装･テストに必要な情報なので，今後研究する必要がある．また，本稿は酒屋倉庫問題の一部しかゴール分解していないので，ゴール分解を完成させ，設計・実装に必要な要求が欠けていることが検出できるかどうか確認することも今後の課題である．

参考文献

[1] Axel van Lamsweerde,“Requirements Engineering: From System Goals to UML Models to Software Specifications”,Wiley,2009.

[2] 岡野道太郎・中谷多哉子, 状態と状態遷移に着目したゴール指向要求分析手法の考察, 知能ソフトウェア工学研究会 (KBSE) , 電子情報通信学会技術研究報告 : 信学技報 Vol.116 No.493,2017,pp25–30

[3] *i* homepage*, http://www.cs.toronto.edu/km/istar/(2017/3/12)

[4] *WHAT IS UML*, http://www.uml.org/what-is-uml.htm(2017/3/12)

[5] 山崎利治, 共通問題によるプログラム設計技法解説, 情報処理 Vol25 No9(1984).

[6] 岡野道太郎・中谷多哉子, 出力データに着目したゴール指向分析による機能要求抽出方法の提案-酒屋倉庫問題のケース- , ソフトウェア エンジニアリング シンポジウム 2015, 情報処理学会 ソフトウェア工学研究会,pp107 – 114

要件定義書からの ファンクションポイント自動計測の試み

山田涼太 山田悠斗 楠本真二 柗本真佑 肥後芳樹
大阪大学大学院情報科学研究科
{r-yamada, y-yuto, kusumoto, shinsuke, higo}@ist.osaka-u.ac.jp

要旨

一般にソフトウェア開発プロジェクトの見積りでは，まず開発規模が見積もられ，それを用いて工数や予算の見積りが行われる．開発規模の尺度として，最近ではファンクションポイント *(FP)* の利用が重要視されている．一方で，*FP* を導入する上での幾つかの課題があり，それらの対策として計測の自動化が期待されている．

本稿では，開発の上流工程で作成される要件定義書を対象とした，*FP* 自動計測の試みについて報告する．提案手法では，要件定義書のうち機能仕様についての記述を提案するテンプレートを用いて書き換え，書き換えられた仕様を解析し，計測に必要な要素を抽出することで *FP* 計測を行う．*4* 種類の小規模な要件定義書を対象に提案手法を適用し，手動での計測と比較することで精度を確認した．その結果，*3* つの要件定義書では手動計測と同様の値が，残りの *1* つの要件定義書では手動計測より若干大きな値が計測された．

1. はじめに

一般に，ソフトウェア開発においては初期段階でのソフトウェア開発規模の見積りが重要である [1] ．不正確な見積りはプロジェクト失敗の原因になることも報告されている [2]．正確な見積りの実現を目指し，ソフトウェア開発規模の見積りに関する研究が盛んに行われている．

見積りの基となる開発規模の尺度としてはソースコードの行数がよく使われていたが，最近ではファンクションポイント (FP) [3] の利用が重要視されている．政府情報システムの開発においても，要求内容に設計または開発に関する工程が含まれる場合には，原則 FP の見積り及びその根拠を示すことが必須となりつつある [4]．その一方，FP の導入における様々な課題のため，ベンダー・ユーザ企業ともに FP は十分に普及していない．1 つ目の課題は，見積りのための基礎データが必要であることである．新規プロジェクトに対する FP を計測したとしても，その値を工数に変換するためには，ある程度，過去プロジェクトの FP 値と工数の情報が必要となる．次に，計測者の判断による誤差が生じる点である．詳細な部分の FP の計測には測定者の判断が必要となる．同一のソフトウェアからの FP 計測であっても，計測者によって誤差が生じてしまうという問題点が指摘されている [5]．例えば，同じ組織の人間が同じアプリケーションに対して測定した場合は 12%，違う組織の人間が即していた場合は 30%以上の誤差が出るという報告もされている [6]．最後に，導入教育が必要である．FP 法を用いるためには，FP 計測方法や，見積りについての知識を身につけなければならない．それらの課題の 1 つの対策として，計測の自動化が考えられる．

FP 計測の自動化に関する研究としては，UML(Unified Modelling Language) 図で記述された設計図からの計測 [7, 8]，Web アプリケーション・プログラムからの計測 [9, 10] 等があり，幾つかの制約を置いた上である程度正確な FP 値の計測が可能となっている．しかし，開発の上流工程で作成される主に自然語で記述された要件定義書を対象とした FP 自動計測手法は著者らの知る限りは存在しない．

本研究では，要件定義書 [11] を対象とした，FP 自動計測手法について検討する．まず，小規模な自然語で書かれた仕様に対して，仕様がどのように記述されていれ

ば計測が可能かを確認・調査する．その結果に基づいて，自然語記述からの FP 計測方法を提案する．提案手法では，機能仕様を FP 計測のためのテンプレートを用いた書き換え，書き換えられた仕様を解析して FP を計測する．更に提案手法に基づいた計測ツールを試作し，様々なプロジェクトの仕様書で検証することで評価を行った．以降，2 章では研究の背景となる諸用語や関連研究について述べる．3 章では今回提案する自動計測の手法について述べる．4 章では実際に自動計測を行った結果と考察について述べる．最後に 5 章で本報告のまとめを述べる．

2. 準備

本章では研究の背景となる諸用語や関連研究について簡単に述べる．

2.1. ファンクションポイント法

ソフトウェアの規模を見積もる手法の 1 つにファンクションポイント (FP) 法 [3] がある．FP 法は，ユーザから見た機能の量を計測する手法で，A.J.Albrecht によって 1979 年に提案された．画面や帳票，ファイルなどを通じた情報の入出力に着目し，それらを種類別に数え上げ，種類数を加重合計した値を機能量としている．以降，IFPUG 法 [12]，MarkII 法 [13]，COSMIC 法 [14] 等，様々な FP 法が提案されてきている．本研究では IFPUG 法を対象とし，以降，FP 法とは IFPUG 法を意味することとする．IFPUG 法は，Albrecht 版の使い勝手や曖昧な部分を改良したバージョンであり，ビジネスアプリケーションソフトウェアを対象として欧米で広く使用されている．IFPUG 法では，ソフトウェアの持つ機能から 5 種類の基本機能要素を以下に示す [15].

内部論理ファイル (ILF)
: 計測対象のアプリケーション内でデータが更新される論理的な関連を持ったデータの集合

外部インターフェイスファイル (EIF)
: 計測対象のアプリケーションによってデータが参照されるデータの集合 (データは更新されない)

外部入力 (EI)
: 計測境界外からのデータ入力によって ILF の更新を行う処理

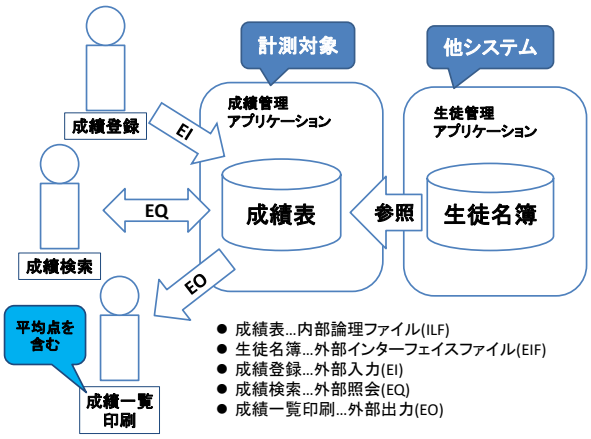


図 1. 基本機能要素の例

外部出力 (EO)
: 計測境界外へのデータ出力を含む処理のうち，出力データに導出データを含むもの

外部照会 (EQ)
: 計測境界外へのデータ出力を含む処理のうち，出力データに導出データを含まないものであり，処理が ILF を更新しないもの

例として，図 1 に学生の成績管理アプリケーションとそこに含まれる基本機能要素を示す．計測対象である成績管理アプリケーション内でデータが更新される成績表は ILF，成績管理アプリケーション内にはなく，成績管理アプリケーションから参照される生徒名簿は EIF となる．また，データ入力によって成績の登録を行う成績登録処理は EI，成績表から平均点などの派生データを含まないデータを出力させる成績検索は EQ，成績表から平均点などの派生データを含むデータを出力させる成績一覧印刷は EO となる．上記の 5 種のうち，ILF と EIF はデータファンクション (以降，DF)，EI と EQ と EO はトランザクションファクション (以降，TF) と呼ばれ，それぞれ計測に必要な要素が異なる [15].

- DF
  - 種別…ILF，EIF
  - DET(Data Element Type)…アプリケーション上で参照される項目の数
  - RET(Record Element Type)…データ群の属性内のサブグループの数

- TF
  - 種別…EI，EQ，EO
  - DET…重複を除いた入出力項目の数に，該当処理において下記のものがあれば 1 ずつ増やしたもの
    - きっかけ (処理が行われるトリガー．ボタンなど)
    - メッセージ (エラーメッセージなど)
  - FTR(File Type Reference)…TF に関わる DF の数

上述した要素を用いて各機能の複雑度を計測し，その上で FP の値を求める．

- DF の複雑度

  表 1 を参照し DF の複雑度を決定する．例えば RET が 1，DET が 26 の ILF があった場合，表 1 より複雑度は低となる．

- TF の複雑度

  EI，EO，EQ のそれぞれに対して用意されている対応表を用いて，TF の複雑度を決定する．例えば，EI は，表 2 を参照し複雑度を求める．FTR が 2，DET が 14 の EI の場合，複雑度は中となる．

- 複雑度からの FP 計測

  抽出された基本要素の複雑さ毎の個数を表 3 に代入し，それぞれに設定されている重み数を掛けて足し合わせた値が FP となる．前述の例であれば，複雑度「低」の ILF(RET : 1，DET : 26) であれば重みは 10 となる．また，複雑度「中」の EI(FTR : 2，DET : 14) であれば重みは 4 となる．

2.2 自然言語解析ツール cabocha

cabocha は Support Vector Machines(SVM) に基づく日本語係り受け解析器である [16]．入力として与えられた文中の各単語の品詞と各文節の係り先を取得する．

図 2 に解析例を示す．矢印はその文節の係り先を表している．図 2 からは,『このデータ群は、』という文節は『データ群です。』に係り,『依頼番号と』という文節が間接的に『持つ』という文節に係っていることが分かる．

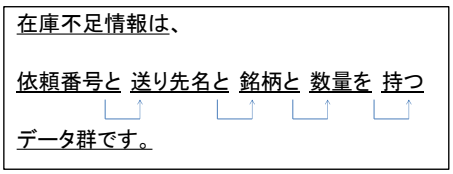


図 2. cabocha による解析例

3. 提案手法

本章では今回提案する計測手法について述べる．

3.1. 提案する FP 計測の流れ

2. で述べた通り，FP 法はユーザから見た機能の量を計測する手法であるため，要件定義書に記載されている情報を用いることで，計測することは可能である．しかし，一般には，必要な情報が省略されている場合も考えられ，必ずしも全ての情報が記載されているとは限らない．そこで，幾つかの小規模な仕様書に対して，どの程度詳細に記述されていれば，自動計測が可能かどうかを確認した．その結果，以下のような仮説を仮定した．

A1 DF 計測の基となるアプリケーションで扱われるデータは，アプリケーションが行う処理（機能）での入力や処理の中で使われる項目として表れている．また，表の形で記載されることもある．

A2 DF において，DF が持つ項目についての情報は仕様に明記されている必要がある．

A3 DF の種別は，TF において参照する DF が明記されていれば更新されるか否かが分かる．

A4 DF の DET は，TF において参照する DF と入出力項目が明記されていれば特定することができる．

A5 TF 計測の基となるアプリケーションが提供する機能は，処理を表す文章や図を用いて表現されることが多い．

A6 TF の種別は，仕様に含まれた特定の用語および出力項目と参照する DF から判別できる．

A7 TF において，全ての TF に「きっかけの有無」が存在する．

表 1. ILF と EIF の複雑度

RET \ DET	1-19	20-50	＞ 50
1	低	低	中
2-5	低	中	高
＞ 5	中	高	高

表 2. EI の複雑度

FTR \ DET	1-4	5-15	＞ 15
0-1	低	低	中
2	低	中	高
＞ 2	中	高	高

表 3. FP の算出のための計算表

	低	中	高	合計
ILF	□× 7=□	□× 10=□	□× 15=□	
EIF	□× 5=□	□× 7=□	□× 10=□	
EI	□× 3=□	□× 4=□	□× 6=□	
EO	□× 4=□	□× 5=□	□× 7=□	
EQ	□× 3=□	□× 4=□	□× 6=□	
			FP	

A8 TF において，DET の根拠となる「入出力項目」「メッセージの有無」および FTR の根拠となる「関わる DF」についての情報は仕様に明記されている必要がある．

以上の仮説に基いて，以下の 4 つのステップを提案する．また，概要を図 3 を示す．

1. 要件定義書における機能を説明する部分の文章を自動計測用テンプレートに合わせて計測用の仕様へと変換する．
2. 変換した仕様から計測に必要な要素を抽出する．
3. 解析結果を用いて FP を計測する．

3.2. 提案手法の前提

今回の提案手法は，3. の最初で述べた小規模な仕様書に対する試行結果に基づくものであるので，いわゆる在庫管理等のビジネスアプリケーションに関する要件定義書が対象となる．また，要件定義書では，機能を説明する部分の文章において以下の条件を満たすものとする．

- 主語，述語，目的語が整った誤解の生まれない文であること．

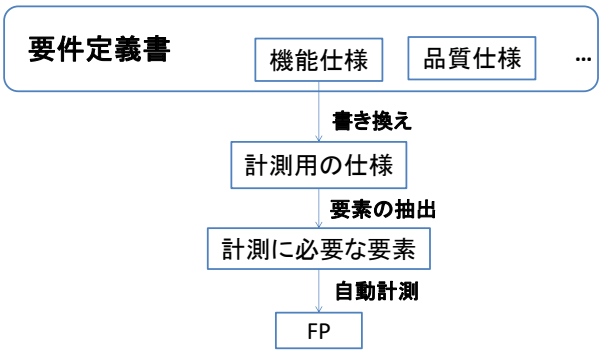


図 3. 提案する FP 計測手法の概要

- 一般には図表参照と記載されている部分について，文章で記述されていること．

図 4 にて例を示す．例えば「ログインは図 4 の画面を用いて行う.」と記載されている場合,「ログイン画面では、名前とパスワードを入力し、ログインボタンを押すことでログインを行う。」と図を参照せずに，文章として記載する．

3.3. テンプレートを用いた書き換え

計測には大きく分けて，DF と TF の計測が必要であり，それぞれの計測において必要な要素が存在する．そ

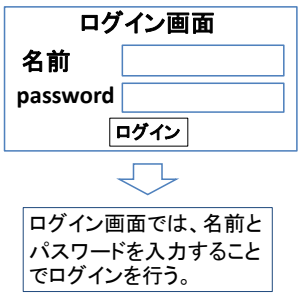


図 4. 図から文章への変換

れらの要素を持った仕様であれば，理論的には自動計測は可能であるが，前述の「主語，述語，目的語が整った誤解の生まれない文」であっても自由度が存在するため，今回はテンプレートを用いて書き換えを行う．

3.3.1 TF のテンプレート記述

TF に必要な要素は以下の通りである．

- 種別…EI，EQ，EO
- DET…入出力項目の数 (重複は除く)+1(きっかけの有無)+1(メッセージの有無)
- FTR…関わる DF の数

今回の計測では全ての TF にきっかけがあると仮定しているので，テンプレートに沿って書き換えられた文章から FP 計測を行うためには,「種別」「入出力項目」「メッセージの有無」「関わる DF の数」といった情報が得られなければならない．計測に必要な要素を含み，かつ判別の容易な文章を作るためのテンプレートが図 5 である．

図 5 のテンプレートに変換する際，処理に入力項目や参照，メッセージが存在しない場合はそれぞれ「(入力項目) を入力することによって、」や「(DF 名) を参照して、(項目) を得ます。」,「(メッセージ名) というメッセージが表示されることがあります。」の部分を記載しない．

実際に要件定義書内の文章をテンプレートへと書き換える．交通費登録内容の照会という機能 [17] を用いて説明する．当該機能は，図 6 のような文章で記述されている．これをテンプレートに変換した場合，図 7 のようになる．上記の通り,「UI に」などのような FP 計測に関係ない部分が取り除かれ，計測に必要な要素のみが残ったことが分かる．また，基本的に「何を入力したか」「何

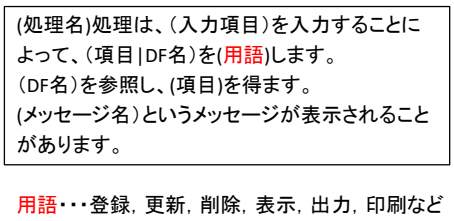


図 5. TF のテンプレート

を出力したか」「何を参照したか」のみをテンプレートに合わせて記せばよいため，FP 計測の知識がない者であっても変換は容易である．

3.3.2 DF のテンプレート記述

DF の計測に必要な要素は以下の通りである．

- 種別…ILF，EIF
- DET…アプリケーション上で参照される項目の数
- RET…データ群の属性内のサブグループの数

RET とは，1 つの DF の中に混在している異なる意味合いを持つデータのまとまりの数を指す．しかし，仕様から得られる要素を用いて対象の DF がどの DF のサブグループであるかを判別することは，仕様の中で直接的な言及が行われていない限り不可能である．そのため，本研究では RET を常に 1 として数えることとした．よってテンプレートに沿って書き換えられた文章のみで FP 計測を行うためには，その文章から「種別」「アプリケーション上で参照される項目の数」といった情報が得られなければならない．計測に必要な要素を含み，かつ判別の容易な文章を作るためのテンプレートが図 8 である．実際に図 8 のテンプレートを用いて変換を行うと図 9 上部にある文章は，図 9 下部のように変換される．

3.4. 自然言語解析による要素の抽出

先述したテンプレートに沿って書き換えた仕様から FP 計測に必要な要素を以降で述べる方法で抽出する．

3.4.1 DF からの要素抽出

図 8 のテンプレートにおける (項目) は当該 DF が持つ全項目である．しかし DF の FP 計測に必要なものは

「所有する項目の数」ではなく「アプリケーション上で参照される項目の数」であるため，このテンプレートからではDETが数えられない．また，種別についての言及もない．そこで，前述したTFの情報を用いて以下のように判別を行う．

- DETの判別
  当該DFのテンプレートにて示された項目のうち，TFの中で一度以上使用された項目のみをDETとして数え上げる．
- ILFとEIFの判別
  当該DFが一度でもEI内にて登録や更新が行われていた場合はILF，そうでない場合はEIFとして判別する．例として「Aを更新する。」と記述されていればAはILFと判別される．

3.4.2 TFからの要素抽出

図5のテンプレートにおける(入力項目)，(出力項目)はDETの一部である入出力項目に相当し，(データ群)は処理に関わるDFを示している．また,「～というメッセージが表示されることがあります」という文によってメッセージの有無が判別できる．従って，このテンプレートで,「入出力項目」「メッセージの有無」「関わるDFの数」は文章から直接読み取れる．

一方で「種別」については明記が行われていない．そこで，このテンプレートから該当TFの種別がEI，EQ，EOの3種の内どれに該当するのかを以下の2段階で判別する．

- EIとEQ，EOの判別方法
  EIとEQ，EOの判別を使われている用語で行う．具体的には,「登録」,「更新」といった外部入力に関する用語であればEI,「表示」,「出力」といった用語が用いられているならEQ，EOといった判別を行う．判別のための用語は，辞書に別途登録しておく．
- EQとEOの判別方法
  EQとEOの判別には，動詞の目的語となる項目を用いて判別を行う．具体的には，FP計測規則[12]に則って，DFに含まれていないデータ（項目）が出力されていれば，トランザクションにおいて何らかの処理を行っていると判断してEOと判別し，

精算データの照会は、UIに社員番号と登録日付を
入力することで、画面中央に利用日と目的と金額
と交通機関とFromとToを表示する。
エラーが起きた際にエラーメッセージが表示される。
社員番号と登録日付を基に、交通費精算ファイル
を参照することで利用日と目的と利用金額と
交通機関とFromとToを得る。

図 6. 変換前の文章

精算データの照会処理は、社員番号と登録日付を
入力することによって、利用日と目的と利用金額と
交通機関とFromとToを表示します。
交通費精算ファイルを参照し、利用日と目的と利用
金額と交通機関とFromとToを得ます。
また、入力データに誤りがあった際エラーという
メッセージの表示を行います。

図 7. 変換後の文章

そのようなデータがない，つまりDFの項目をそのまま出力していればEQと判別する．

3.4.3 TFでない処理について

今回の計測上，仕様の書き換えの際にTFではない処理をTFのテンプレートを使って変換してしまう場合があり得る．今回の手法では，EI, EQ, EOの判断のため辞書に登録している「更新」や「表示」「出力」等の用語が存在しない場合，EI，EQ，EOではないとしてTFとしては扱われないため，計測には問題ない．

3.4.4 重複処理について

FP計測において，処理内容が同一の処理は重複処理と呼ばれる．この重複処理は複数存在しても1つしか計測しないルールとなっている．そのため，図10のように同一の項目を持つ処理が複数存在した場合，それらを重複処理と判断し，1つしか数えないようにする．

3.5. FPの計測

ここまでのステップにてFP計測に必要な要素が全て揃うため，2．1．で記述したように要素から複雑度およびFPを算出する．

(DF名)は、(項目)を持つデータ群です。

図 8. DF のテンプレート

データベース名	データ項目
図書情報	出版社，題名，著者名
利用者情報	氏名，利用者番号

図書情報は、出版社と題名と著者名を持つデータ群です。
利用者情報は、氏名と利用者番号を持つデータ群です。

図 9. DF のテンプレート記述への変換

3.6. 計測ツールの実装

これまでに述べた方針に基づく FP 計測ツールを Java 言語を用いて試作した．計測ツールの役割は「cabocha を用いた自然言語解析による計測に必要な要素の抽出」および「DF や TF の種類などの cabocha では抽出できない要素の判別」，そして「要素を用いた FP の算出」である．図 11 に本研究における FP 計測の流れと計測ツールの関係を示す．

計測の流れは以下の通りである．

1. 要件定義書のうち，機能に関わる部分の文章を意味内容が明確となるよう手動で変換する．
2. 書き換えた文章をテンプレートを用いて計測用の仕様に手動で変換する．
3. 計測用の仕様に cabocha による自然言語解析を行い，計測に必要な要素を抽出する．
4. 3. で抽出した要素を用いて，FP 計測を行い結果を出力する．

4. ケーススタディ

本章では，対象となる 4 つの仕様書に対して自動計測を行った結果及び考察をまとめる．

処理名	種別	入力項目	出力項目	参照したDF	メッセージの有無
図書検索A	EQ	図書番号	題名，著者名	図書情報	有り
図書検索B	EQ	図書番号	題名，著者名	図書情報	有り

処理名以外の全項目の内容が一致
→「図書検索A」と「図書検索B」は重複処理と判断

図 10. 重複処理となる例

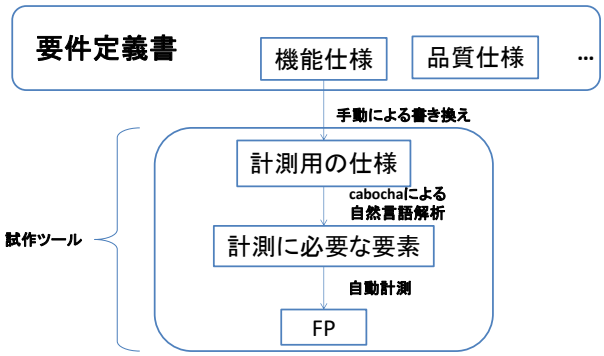


図 11. 計測フローと計測ツールの役割

4.1. 対象システムの概要

今回自動計測の対象とした 4 つのシステムの概要を記す．

1. 旅費精算システム
   交通費の管理を行うシステムである．社員が出張にて支払った交通費を登録しておくことで上司がそれを確認，承認できる．また，承認されたものについては銀行への支払い命令が行われる [17].
2. 図書システム
   図書の管理を行うシステムである．図書の登録や増冊，検索や削除などが行える．ある企業の社内勉強用資料に記載されているものである．
3. 酒屋問題
   酒の在庫管理を行うシステムである．積荷として運ばれてきた酒の銘柄や本数の登録や，出庫指示などが行える [18].
4. 健康情報サイト
   病院や健康情報について扱うサイトである．健康診断が行われている病院の検索や健診情報の登録が行える [19].

表 4. 計測結果

		旅費精算				図書システム				酒屋問題				健康情報			
		手動		自動		手動		自動		手動		自動		手動		自動	
DF	ILF	17	7	17	7	14	14	14	14	14	14	14	14	21	21	21	21
	EIF		10		10		0		0		0		0		0		0
TF	EI	52	12	52	12	40	13	40	13	24	9	27	12	75	30	75	30
	EQ		20		20		7		7		3		3		41		41
	EO		20		20		10		10		12		12		4		4
FP		69		69		44		44		38		41		96		96	

4.2. 提案手法の計測結果と手動計測結果の比較

先述した 4 つのシステムの要件定義書に対して，FP 計測の熟練者 (一部は FP 計測における国際資格である CFPS[20] の有資格者）による手動計測結果および計測ツールによる自動計測結果を表 4 に示す．表 4 より，手動で計測した結果と比べ，酒屋問題において TF と FP が 3 ずつ多く計測されていることが分かる．

4.3. 考察

1. 誤差についての分析

   旅費精算システム・図書システム・健康情報サイトの要件定義書に対しては，手計算で得られた FP と同様の値を得ることができたが，酒屋問題に関しては誤差が生じた．この原因について考察する．

   酒屋問題における誤差の原因を調べたところ，本来無視すべき付随処理を計測していることが原因であった．付随処理とは TF として扱われる処理において，本来の処理に付随して行われる処理の事を言う．酒屋問題においては酒の在庫情報を更新する「在庫情報更新処理」が相当し，この処理は本来出庫の指示を担う出庫指示票を出力する処理である「出庫指示票出力処理」に付随して行われる処理であったが，今回の提案手法での計測では別々の処理として計測された．FP 計測においてこの付随処理は無視される，つまり計測されないものであるが，その処理が付随処理であるかどうかは FP 計測の知識がある者でしか判断できず，またツール側から判断することもできない．そのため，FP 計測の知識があまりない者がテンプレートへの変換を行った場合，この付随処理も含めて記述してしまう可能性が高く，結果として今回のように付随処理の分多く計測される可能性も高い．この問題に関しては，自動計測後の結果をファイル出力し，利用者の手によって修正・変更を行う方法による解決が望ましい．

2. DF が明記されていない場合の対応

   今回は DF や TF について記述が正確に行われているという条件で FP 計測手順を提案したため，有資格者が計測した際の値と近いものを得ることができた．しかし，実際の要件定義書の中には DF について明示的に記載されていないケースが存在する．その場合，今回の研究にて提案した手法をそのまま適用できないため，一つのアイデアとして，TF で扱われているデータ項目の集合を DF の候補とするということを考えており，今後その詳細を検討する予定である．

5. あとがき

本研究では，自然言語で書かれた仕様書からの FP 自動計測の実現を試みた．その方法として，テンプレートを用いて仕様を書き換えた上で，自然言語解析を用いて FP 計測に必要な要素を抽出し FP 計測を行う手法を提案した．4 つの仕様書を対象とし，提案手法を用いて自動計測を行った．その結果，3 つの仕様書においては手計算で見積りが行った場合と同様の FP を算出することができた．残り 1 つの仕様書では手計算で得られた FP と異なる値が得られたことから，その原因に対する考察を行った．

本研究の今後の課題としては，下記のような事が考えられる．

1. 適用事例の追加

今回取り扱った 2 つの要件定義書は最大でも FP が 100 程度のものであり，また，FP が小さいために要件定義書内で扱われる処理も種類が少ない．そのため，FP が大きく，また，多種多様な要件定義書への適用を行いそこで生じた問題点を解決することで，本研究の実用性を高める．

2. 手法の改善

テンプレートを用いた書き換えコストの調査やテンプレート記述以外のより計測に適した記述方法の提案，および穴埋め式記述を用いたツールの開発によるより記述量の少ない計測手法の提案などによって本研究における手法そのものの改善を行い，本研究およびツールの有益性を高める．

謝辞

本研究を進めるにあたって，貴重なご助言を頂きました株式会社 NTT データ　鵜澤仁氏，一般財団法人経済調査会　大岩佐和子氏，押野智樹氏に深く感謝申し上げます．

本研究は一部，日本学術振興会科学研究費補助金基盤研究（S）（課題番号：25220003）の支援を受けている．

参考文献

[1] B. Boehm, C. Abts, S. Chulani. Software de- velopment cost estimation approaches - A survey. Annals of software engineering, Vol. 10, No. 1-4, pp. 177-205, Springer, 2000.

[2] R. L. Glass. Facts and fallacies of software engineering, Addison-Wesley Professional, 2002.

[3] A. J. Albrecht. Function point analysis. Encyclopedia of Software Engineering, Vol. 1, pp. 518-524,Addison-Wesley Professional, 1994.

[4] 総務省行政管理局. 政府情報システムの整備及び管理に関する標準ガイドライン実務手引書. 政府情報システムの整備及び管理に関する標準ガイドライン実務手引書, p. 122, 内閣官房情報通信技術 (IT) 総合戦略室,2015.

[5] G. C. Low, D. R. Jeffery. Function points in the estimation and evaluation of the software process. IEEE Transactions on Software Engineering, Vol. 16, No. 1, pp. 64-71, IEEE, 1990.

[6] B. Kitchenham. The problem with function points. IEEE software, Vol. 14, No. 2, p. 29, IEEE Computer Society, 1997.

[7] T. Uemura, S. Kusumoto, and K. Inoue. Function-point analysis using design specifications based on the united modelling language.Journal of software maintenance and evolution:Research and practice, Vol. 13, No. 4, pp. 223-243, IEEE Computer Society, Wiley Online Library, 2001.

[8] G. Cantone, D. Pace, and G. Calavaro. Applying function point to united modeling language: Conversion model and pilot study.Software Metrics, 2004. Proceedings. 10th International Symposium on, pp. 280-291. IEEE, 2004.

[9] T. Edagawa, T. Akaike, Y. Higo, S. Kusumoto, S. Hanabusa, and T. Shibamoto. Function point measurement from web application source code based on screen transitions and database accesses. Journal of Systems and Software, Vol. 84, No. 6, pp. 976-984, Elsevier, 2011.

[10] P. Fraternali, M. Tisi, and A. Bongio. Automating function point analysis with model driven development. In Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research, p. 18. IBM Corp. , 2006.

[11] IEEE Computer Society. Software Engineering StandardsCommittee, IEEE-SA Standards Board. IEEE recommended practice for software requirements specifications, IEEE, 1998.

[12] IFPUG: Function Point Counting Practices Manual, Release 4. 3. International Function Point Users Group, 2010.

[13] C. R. Symons. Software sizing and estimating:Mk II FPA (function point analysis), John Wiley & Sons, Inc. , 1991.

[14] 山口正明 (著) 調重俊 (監修) . ファンクションポイント cosmic- ffp 法実践ガイド―組込み系・リアル

タイム系に最適なソフトウェア規模・工数の見積り方法, 日科技連出版社, 2007.

[15] 柏本隆志, 楠本真二, 井上克郎, 鈴木文音, 湯浦克彦, 津田道夫. イベントトレース図に基づく要求仕様書からのファンクションポイント計測手法. Vol. 41,No. 6, pp. 1895-1904, 2000.

[16] Cabocha/南瓜 https://taku910.github.io/cabocha/.

[17] 日本ファンクションポイントユーザ協会. ファンクションポイント計測コース演習編課題テキスト,2014.

[18] 山崎利治. 共通問題によるプログラム設計技法解説. 情報処理, Vol. 25, No. 9, p. 934, 1984.

[19] 経済調査会調査研究部第二調査研究室. Web サイト制作費の見積りの現状と課題, 2015.

[20] CFPS 概要. http://www.jfpug.gr.jp/app-def/S-102/wp/?page_id=1374.

ソフトウェア開発発注者育成のための形式手法を取り入れたプログラミング教育

伊藤 栄一郎
山梨学院大学
e-itoh@ygu.ac.jp

小田 朋宏
株式会社 SRA
tomohiro@sra.co.jp

荒木 啓二郎
九州大学
araki@ait.kyushu-u.ac.jp

要旨

品質の高い仕様記述は開発における手戻りを減らし最終成果物であるソフトウェアシステムの品質を高めるために重要な工程である．仕様記述は開発を計画するドメイン専門家とその実現を担当する開発者の双方が記述やレビューを通して関与すべき工程である．したがって，品質の高い仕様記述を得るためにはドメイン専門家と開発者の双方において仕様記述に関する知識と技能を高める必要がある．我々は，品質の高い仕様記述を得るための手法として形式仕様記述に着目し，将来企業においてソフトウェアシステムの発注者側として開発に携わることが予想される大学生へのプログラミング導入教育において，形式仕様のエッセンスであるプログラムの正当性，手続きの事前条件と事後条件について講義を行い，ビジュアルプログラミング環境上での演習およびペーパーテストを実施した．本論文では，講義の内容と結果，および講義向けに開発したビジュアルプログラミング環境への拡張機能を説明し，議論する．

1. はじめに

ソフトウェア開発において仕様記述をはじめとする上流工程は開発の経済性や成果物の品質を高める上で重要な工程である．専門教育においてもソフトウェア工学などを通して多くの大学においてソフトウェア開発のライフサイクルや分析，仕様記述，設計に関する技術教育がカリキュラムに組み込まれている．

近年では，PBL (Problem Based Learning) を通して，実社会で求められているソフトウェアの開発を経験することで技術だけでなくシステム開発者としての基本的な姿勢を学ぶ訓練が行われている [1]．それらの多くはソフトウェア開発者の視点に立ったものである．PBL においても学生は主に開発者として模擬プロジェクトに参加する．しかしながら，ソフトウェア開発における仕様記述工程は開発者のみが行う工程ではない．システムが解決すべき問題に対してシステムの仕様が妥当であることを確認するためには，ドメイン専門家が仕様記述を理解し適切なフィードバックを行うことが必要である [2]．したがって，ソフトウェアの仕様記述に関する基本的な知識および技術は，開発者のみならずドメイン専門家も習得することが望ましい．

仕様記述の品質を高める技術として形式仕様がある [3]．形式仕様とは，構文および意味論が厳密に定義されている記法によって，解釈に曖昧性のない仕様を記述する技術である．モデル規範型の仕様記述言語では手続きや関数の事前条件および事後条件を表明することで，手続きや関数が満たすべき仕様を高い抽象度で記述することができる．経済性や成果物の品質の観点から多くの成功事例があるが [4, 5, 6]，ソフトウェア開発全体においてはまだ適用は一部に止まっている．Hall は [7] において，形式手法に関する誤解の 6 番目に「形式手法はユーザに受け入れられない」という誤解を挙げて，ソフトウェアシステムの受託開発におけるユーザ企業側に受け入れられないという誤解から形式手法の普及が遅れていると指摘している．この誤解は現在においても改善されていない．

我々は，システム発注者のための形式仕様の基本的な技術として，事前条件および事後条件の表明に着目した．表明は，プログラミングの入門段階から導入することができ，適切な課題設定によって必ずしも高度な数学知識に依らなくても記述することが可能である．本研究では，小規模なプログラミングの演習で事前条件および事後条件を扱うことによって，形式仕様のエッセンスを学ぶこ

とができるのではないかという仮説を立てた．プログラミング導入教育を目的とした講義において，形式仕様の基本的な要素である事前条件および事後条件の表明についての知識をレクチャーし，演習として簡単な問題をプログラミング入門向けのビジュアルプログラミング環境上で表明を行う課題を与え，ペーパーテストを実施した．

第 2 節では演習のために開発した表明付きビジュアルプログラミング環境 Assertch について紹介する．第 3 節ではプログラミング導入教育で実施したレクチャーや演習，試験を説明し，その結果を示す．第 4 節では実施した結果について考察を行い，最後に第 5 節ではまとめと今後の課題を示す．

2. 表明付きビジュアルプログラミング環境 Assertch

プログラミング教育を目的として Scratch[8] や Viscuit[9] を始めとするビジュアルプログラミング環境が開発され，小学生から大学生を対象に適用されている．ビジュアルプログラミング環境は，構文を覚えるなどの学習者の負担を最小限に止めながら，プログラムを構築することのエッセンスを実践を通して学ぶことを可能にしている．

我々は，プログラミング導入教育の中でプログラムの正当性や表明の概念を導入するために，Scratch に類似したブロック型のビジュアルプログラミング環境である Phratch に表明ブロック等を追加する拡張機能として，Assertch を開発した [10]．Phratch は オープンソースとして公開されており[1]，2017 年 3 月現在も活発に開発が続けられている．

図 1 に Phratch 上で簡単なプログラムを作成した画面例を示す．画面は大きく左側，中央，右側に 3 分割されている．画面左にはプログラムを構成するための「ブロック」と呼ばれる部品がカタログとして分類されて提供されている．ユーザがカタログ上部で分類を選択すると，カタログ下部にその分類に含まれるブロックが縦方向に並べられて提供されている．ユーザはブロックをカタログ上から画面中央のスクリプトエリアにドラッグ＆ドロップによって配置する．図 1 の例では，スクリプトエリアには，いわゆる FizzBuzz 問題のプログラム（1 から 100 までの数を数え上げ，もし 15 の倍数であれば FizzBuzz あるいは 5 の倍数であれば Buzz，3 の倍数であれば Fizz，いずれでもなければその数を表示する）がブロックを使って組み立てられている．画面右側の上部では，各変数の値や，タートルグラフィックスやユーザとの対話を行うエージェントとしてロボットが表示されている．

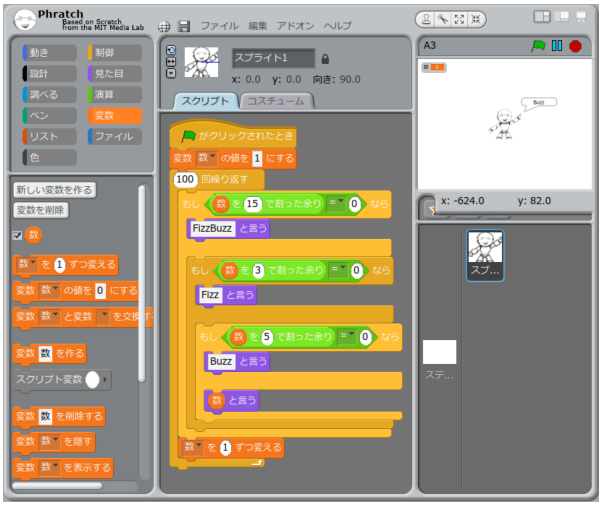


図 1. Phratch 環境の画面例

Assertch は，Phratch に表明を始めとする仕様記述に有用なブロックを追加する拡張モジュールである．現在 Phratch の標準アドオンとして提供されており，AddOn メニューから Assertch を選択することでネットワーク経由で自動的にインストールすることができる．図 2 に，表明ブロックを収めた設計カタログを示す．

「を確認するブロック」は，一般のプログラミング言語における assert 文に相当し，引数として与えられたブロックを評価実行し，true であれば実行を継続し，false であれば表明エラーとして停止するブロックである．「事前条件ブロック」は，内側にブロックを抱え込む構造を持つブロックで，内側のブロックに対する事前条件を表明するブロックである．引数として与えられたブロックを評価実行し，true であれば内部に抱えたブロックを実行し，false であれば表明エラーとして停止する．「事後条件ブロック」は，内側のブロックに対する事後条件を表明するブロックである．内部に抱えたブロックを実行した後で，引数として与えられたブロックを評価実行し，true であればプログラムの実行を継続し，false であれば表明エラーとして停止する．事前条件と事後条件の両方を表明する「事前条件事後条件ブロック」も提供されている．いずれの表明ブロックも，表明エラーが発生した

[1] http://www.phratch.com/

図 2. Assertch で拡張された表明ブロック

ブロックの色を黒色から赤色に変化させることで，どの表明が守られなかったかをユーザに示す．エラーとなった表明ブロックの色は，「表明エラーをリセットするブロック」によってリセットすることができる．

これらの表明ブロックを利用することで，プログラミング学習者であるユーザは仕様記述とプログラムの関係について以下のことを学ぶことができる．

- 試行錯誤の結果たまたま動くプログラムができるのではなく，プログラムの目的と前提条件を明確にした上で，ブロックを合理的に選択しプログラムを組み立てること
- プログラムに誤りがあった時に，各部分で期待した条件に照らし合わせて，どのブロックに問題があるかを特定すること
- プログラムを作るだけでなく，意図した通りの動作が行われるかをテストすること
- プログラムを拡張する時に，元のプログラムの設計意図に適合した改変を行うこと

これらの技術を学ぶことによって，仕様記述がプログラムの開発においてどのような役割を担っているかを体得するための環境を提供することが，Assertch の開発目的である．

3. プログラミング導入教育への適用

現在，多くの大学の様々な課程においてプログラミング導入教育が行われている．山梨学院大学経営情報学部は，経営学と情報科学をまたがる専門的な知識と技能の習得によって，新しい時代を担う情報技術者・情報管理者を育成することを目標の 1 つにしている．ソフトウェアシステムのユーザ企業側においてシステム開発に携わる人材を育成することが期待されていることから，プログラミング教育においては開発者としての視点だけでなく発注者として求められる知識や技能を学習することが，プログラミング導入教育を行う講義において求められる．

平成 28 年度後期に「情報処理論」として以下の内容の講義を実施した．

- アルゴリズムとプログラミング言語
- グラフアルゴリズム

- プログラムの正当性

うち，アルゴリズムとプログラミング言語は Phratch，プログラムの正当性は Assertch を利用して講義および演習を行った．

プログラムの正当性の演習では，二次方程式の解の 1 つを求めるプログラムを表明ブロックを使って作成し，レポートとして提出する課題を出した．受講者に対して，二次方程式 $ax^2+bx+c=0$ の解法として，$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$ の公式を示した上で，この式が多くの前提条件，すなわち事前条件を持っていることを説明し，その事前条件や事後条件を洗い出しながらプログラムを構成するように指示した．

形式仕様記述工程では，ドメイン知識や実現する機能の特性から様々な事前条件や事後条件を発見し，それらを適切な機能単位に記述する作業が行われる．これを簡単なプログラムの作成で模擬的に体験させるために，二次方程式の解法を課題として選んだ．二次方程式の解の 1 つを求めるプログラムを構成するためには，これらの事前条件および事後条件を発見し，適切に表現し，それらの事前条件および事後条件を満たすようにブロック群を合理的に構成しなければならない．

プログラム全体の事後条件は $ax^2+bx+c=0$ だが，$a=0$ の場合には分母が 0 となるので，上記公式は単純に適用できない．そこで，$a=0$ を事前条件とした副プログラムを構成する必要があるが，その場合には $a=0$ により事後条件は $bx+c=0$ となり，一次方程式の解法を使うことができる．さらに，一次方程式の解法において，$a=b=0$ の場合には事後条件が $c=0$ となるため，$c=0$ ならば任意の数が解となり，$c\neq 0$ ならば解なしとなる．また，$a\neq 0$ の場合には，判別式によって実数解があるかどうかを判定しなければならない．

実習において，受講者は以上のような事前条件および事後条件の抽出を行い，表明ブロックによってそれらを表現し，表明ブロックに内包されるブロック群を合理的に構成することで課題を遂行する必要がある．図 3 に Assertch での正答例を示す．

学期の最後に定期試験としてペーパーテストを実施した．ペーパーテストでは，アルゴリズム，プログラミングおよび正当性それぞれについての基本的な用語の理解，プログラムやアルゴリズムの実行に関する穴埋め，そして，事前条件や事後条件がプログラム開発の委託や受託にどう関係するのかを自由形式で記述する問題を出題した．

3.1. 結果

授業では全受講者 79 人のうち最終試験を受験した 72 人を評価の対象とした．そのうち 61 人が演習後のレポートを提出し，9 人は課題において表明ブロックを使いこなしていた．27 人は表明は適切でなかったが求められたプログラムを完成させた．

演習の結果とペーパーテストの結果の関係を表 1 に，演習の結果と表明に関する設問の正解/不正解による分布を表 2 に示す．演習において適切な表明を付けてプログラムを完成させた受講者は，表明がプログラムの委託および受託それぞれにおいてどのような役割を果たすかという設問に対する正答率は 0.67 であり，その他の設問，すなわちプログラミング言語およびアルゴリズムに関する設問に対する正答率は 0.77 であった．表明に不足はあったがプログラムを完成させた受講者はそれぞれ正答率が 0.63 および 0.77 と，適切な表明を付けた場合と大きな差はなかった．プログラムが未完成だった受講者は，表明と委託/受託に関する設問では正答率が 0.53 と，プログラムを完成させた受講生グループと比較して低い正答率にとどまった．その他の設問においては正答率に大きな差はなかった．レポート未提出の場合には，いずれの設問についても低い正答率になった．結果として，表明の意義に関する理解は，演習において表明そのものを適切に使いこなしていたかに関わらず，プログラムを完成させたかどうかで差がついた．

4. 議論

表明ブロックを用いた演習結果とペーパーテストでの表明に関する設問での正答率の関係について，プログラムが完成したか否かが差をつける因子であることがわかった．考えられる説明としては，表明ブロックの利用を意識して課題に取り組んだことによって，たとえレポートとして提出したプログラムでの表明が不完全であっても，表明という概念について適切な理解を得ることに繋がったと考えられる．

受講生の一部が既に一定のプログラミング技術に習熟していたことの影響も考えられるが，表 2 から，アルゴリズムやプログラミング言語に関する設問では得点が低いが表明の設問には正答できている受講者や，逆に表

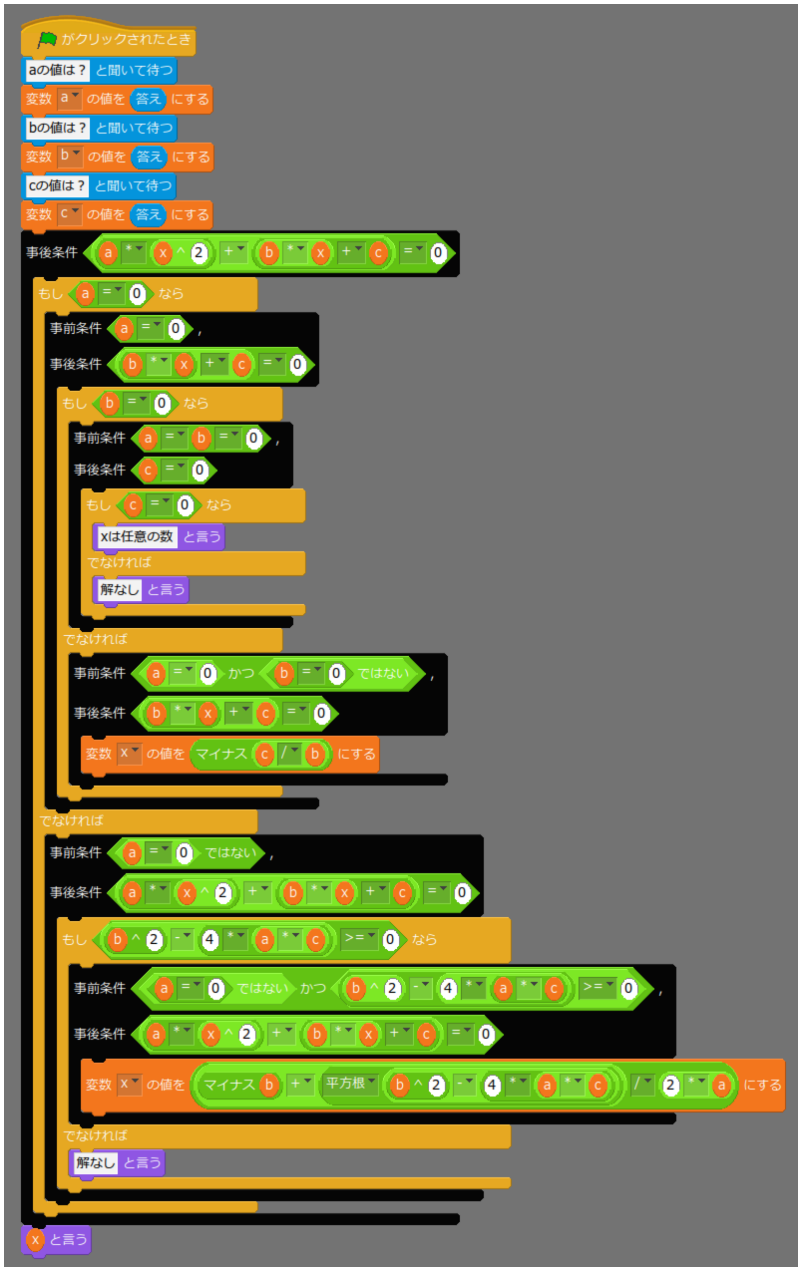


図 3. Assertch での二次方程式解法の正答例

表 1. 演習結果とペーパーテストの正答率

演習結果	「表明と委託/受託」 正答率	その他の設問の 正答率
表明付きでプログラム完成	0.67	0.77
表明なしでプログラム完成	0.63	0.77
プログラム未完成	0.41	0.71
レポート未提出	0.33	0.57

表 2. ペーパーテストにおける表明以外の設問の得点と表明設問の正答の人数分布

表明以外の得点	表明設問 正解	表明設問 不正解
-31	9	4
32-35	9	6
36-39	11	15
40-	6	12

明の設問には正答できなかったがアルゴリズムやプログラミング言語の設問では得点が高い受講生が少なくないことが示されており，プログラミングの知識や技能は表明や仕様記述に関する理解とは必ずしも直結していないと考えられる．すなわち，将来ソフトウェアシステムのユーザ企業において発注者としての役割を担う技術者を育成する上で，アルゴリズムやプログラミング言語の知識や技能だけで適性を判断するのではなく，表明に関する知識や技能によって適切な仕様記述を行うための教育が必要である可能性を示唆している．

演習に用いたプログラミング環境については，多くの改良の余地があることがわかった．特に，演習において表明に記述する式が複雑になり，受講生の時間や労力の多くが式をブロックで組み立てることに費やされてしまっていた．ブロック式のビジュアルプログラミング環境であることの欠点であり，条件式をテキスト形式で入力可能なブロックを導入するなどして解決する必要がある．

5. おわりに

ソフトウェアシステムを開発するための技術者の不足が叫ばれて久しい．情報システム開発の要求およびその複雑さは，これからも増えていくことが予想されている．そこで求められる生産性と品質を獲得するためには，品質の高い仕様を記述することができる技術者が必要である．従来のソフトウェア開発を受託する技術者の育成により開発者の数を増やすだけでなく，仕様を策定するユーザ企業側においても，品質の高い仕様を記述する知識と技能を持った技術者を確保することが求められる．

本研究では，将来ユーザ企業またはソフトウェア開発企業において技術者として開発に携わることが期待される受講者に，プログラミング教育向けのビジュアルプログラミング環境に表明機能を追加することで，正しいプログラムについて学ぶ機会を提供することを試みた．演習の中で，プログラムが期待した動作をしなかった時に，プログラムの誤りなのか仕様の誤りなのかを考え修正する受講生の姿を観察した．

「情報処理論」にプログラムの正当性および表明に関する講義や演習を取り入れて 2 期目が経過した．今後もプログラミング環境の改善を行いつつ，教材や演習内容の改善に務める．

6. 謝辞

本研究を遂行するにあたって Jannik Laval 氏，阿部和広氏，中小路久美代氏，山本恭裕氏より多くの助言を受けた．ここに謝意を記す．有益なご指摘を頂いた査読者の方々に感謝する．本研究の一部は, JSPS 科研費 (26330099) の助成を受けたものである．

参考文献

[1] 井上明, 金田重郎. “実システム開発を通じた社会連携型 PBL の提案と評価.” **情報処理学会論文誌**, Vol. 49, No. 2, pp. 930-943, 2008.

[2] Oda, T., Yamomoto. Y., Nakakoji, K., Araki, K., and Larsen, P. G. “VDM Animation for a Wider Range of Stakeholders”. In Proceedings of the 13th Overture Workshop, pp 18-32, 2015.

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

[4] Fitzgerald. J, Larsen, P. G., and Sahara, S. “VDM-Tools: Advances in Support for Formal Modeling in VDM”, *ACM Sigplan Notices*, Vol. 43, No. 2, pp. 3-11, 2008.

[5] Kurita, T., and Nakatsugawa, Y. “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.

[6] IPA/SEC. “厳密な仕様記述における形式手法成功事例調査報告書”, 独立行政法人情報処理推進機構 技

術本部 ソフトウェア・エンジニアリング・センター, 2013.

[7] Hall, A. “Seven myths of formal methods.” *IEEE software*, Vol. 7, No. 5, pp. 11-19, 1990.

[8] Maloney, J., et al. “The scratch programming language and environment.” *ACM Transactions on Computing Education (TOCE)*, Vol. 10, No. 4: 16, 2010.

[9] Harada, Y., and Potter, R. “Fuzzy Rewriting.” *End User Development*, Springer Netherlands, pp. 251-267, 2006.

[10] Oda, T., Araki, K., and Larsen, P. G. “ViennaTalk and assertch: building lightweight formal methods environments on pharo 4.” in Proceedings of the 11th edition of the International Workshop on Smalltalk Technologies (IWST16), pp. 4:1-4:7, 2016.

「VDM++仕様記述と分析」の定石と手筋

佐原　伸
法政大学情報科学研究科
ss@shinsahara.jp

要旨

形式仕様記述言語VDM++で仕様を記述し，それを分析できる産業界の技術者を育てることは簡単ではない．

しかし，長年にわたって囲碁や将棋を学習し，多くの技術者や学生に，VDM++による仕様記述方法を教えた経験から，「仕様記述と分析」を定石や手筋を使って教育することが，囲碁や将棋の教育と同様に有効だと考えた．特に，仕様記述に重要な不変条件(invariant) ・事後条件(post-condition)・事前条件(pre-condition)の記述を容易にできるよう，定石と手筋を集めて検索可能にすれば比較的容易に「仕様記述と分析」行うことができる．

このような「定石集」を試作し，評価した結果，3節で述べるように，かなり役立ちそうである事が分かった．また，これらの定石をコンピュータ化すれば，検索や「仕様の一部」の自動生成に活用できる可能性もある．

0. はじめに

産業界の技術者は要求仕様や設計仕様のVDM++による仕様記述に困難を感じる人が多い．特に，不変条件，事後条件，事前条件の記述ができない．手続き的記述には慣れているが，状態を宣言的記述で表現することに慣れていないため，記述することが難しいようである．

例えば，顧客のデータベースをインスタンス変数「i客DB」で表し，そこに新しい客「a新しい客」を追加する操作(operation)では，事後条件を

post i客DB = i客DB~ union {a新しい客}

と記述すればよいのであるが，これがなかなか書けない．ここで，~は，インスタンス変数の旧値（操作が実行される前の値）であり，集合の合併演算子unionを使って，引数で与えられた「a新しい客」を要素に持つ集合「{a新しい客}」と合併した集合が，「i客DB」（操作が終わった後のインスタンス変数「i客DB」）と等しいと書けばよい．

このような「手筋」（仕様中のいくつかの場所で役立つ方法）を「集合に要素を追加する手筋」として整理し，仕様記述例と説明を与えておけば，仕様記述の上達に役立つ．囲碁や将棋では，このような手筋や定石が整理され蓄積されていて，囲碁の場合，アマチュア初段程度の実力でも2万個ほどの定石と手筋を使いこなさなければならないとされている．逆に言えば，定石や手筋を使いこなせれば，アマチュア初段程度の実力を発揮できるということであり，VDM++仕様記述の定石と手筋を整理して蓄積すれば，VDM++アマチュア初段レベルの技術者を養成できることになる．

しかし，囲碁や将棋のように仕様記述と仕様検査のための定石と手筋を集めようとして，文献1, 2, 3, 4など，関連しそうな文献を調べたが，いずれも証明が中心[1]であり，産業界向けの例題としては適当なものがなかった．そこで，筆者の作成した実用的なモデルをセミナー用に縮小したモデル群と，文献5を中心に定石と手筋を収集して，これを整理し，評価を行った．

[1] 筆者たちは，証明を使わずに仕様をテストする手法を中心に産業界へのVDM普及を行っている．現存するVDMのツールも，すべて仕様をテスト実行する仕様アニメーション機能しか提供していない．

評価は，「定石と手筋集」作成では参考にしなかった既存文献（文献5と6）上の例題を使って，どの程度定石と手筋が使われているかを調べた．その結果．すべての不変条件，事後条件，事前条件の記述に，この「定石と手筋集」の定石と手筋が使われていた事が分かった． 3節で．収集した定石・手筋がどの程度有効であるかの実験結果を示した．

収集した定石・手筋の構成は以下のとおりである:

1. 仕様記述
2. 分析
3. ツール
4. マニュアル

分析を追加したのは，記述した仕様を分析するための重要な技術である検証と妥当性検査について，開発現場の技術者に 知識があまり無いからである．
ツールとマニュアルの章を追加したのは，これらの知識がないと実際の普及の上で大きな障害になることが分かったからである．ただし，本論文では紙幅の関係で4.マニュアルについては省略した．

1. アイデア

1.1 仕様記述についてのアイデア

「1. 仕様記述」では，次の観点で定石・手筋を整理した．すなわち，VDM仕様の特定の場所でよく使う記述方法を定石と呼び，VDM仕様のいろいろな場所でよく使う記述方法を手筋と呼ぶ．

定石は，事前条件，事後条件，不変条件でよく使う仕様記述方法であるので，下記のように分類した．

- 1.1.1 定石
  - 1.1.1 事前条件
  - 1.1.2 事後条件
  - 1.1.3 不変条件

1.1.1, 1.1.2, 1.1.3 は「規則」と「定石」を記述している．ここで，「規則」とはVDM++の文法規則のことである．例えば，関数の場合の事前条件式は，式中で局所名以外の名前はパラメータ名しか使えない．これも一種の「定石」ではあるので「規則」として記述した．
「1.1.3 不変条件」定石は，下記のように分類した．

- 1.1.3.1 型の不変条件
- 1.1.3.2 インスタンス変数の不変条件
  - 1.1.3.2.1 インスタンス変数の正当性
  - 1.1.3.2.2 インスタンス変数同士の制約

「1.1.3.1 型の不変条件」中には，例えば「1.1.3.1.1.1 データが意味的に正当か確認する」定石(IDは**JIT_wf_1**)があり，例えば以下のように，レコード型である予約明細型は予約明細を表しているの，属性**f座席集合**は空集合ではないという制約を記述している．この定石は，3節の「定石・手筋の評価」の結果，かなり使われていることが分かった．

```
types
public 予約明細 ::
  f顧客名 : 顧客名
  f列車番号 : 列車番号
  f乗車駅 : 駅番号
  f降車駅 : 駅番号
  f座席集合  : set of 座席番号
inv
```

```
w予約明細 == w予約明細.f座席集合 <> {};
```

「1.1.3.2.2 インスタンス変数同士の制約」中には，例えば「1.1.3.2.2.2 写像の定義域・値域[2]同士の部分集合関係を記述する」定石(IDはJII_ibV_subset2)があり，例えば以下のように，インスタンス変数同士の制約条件を記述する．

```
types
public 管制官 = token;
public 空域 = token;
public 航空機 = token;
public 当直集合 = set of 管制官;
public 管制 = inmap 空域 to 管制官;
public s位置 : 位置 := {|->};

instance variables
  public s当直集合 : 当直集合 := {};
  public s管制 : 管制 := {|->};
  public s許容数 : 許容数 := {|->};
  public s位置 : 位置 := {|->};

inv
   rng s管制 subset s当直集合 and
   dom s管制 subset dom s許容数 and
   rng s位置 subset dom s管制
```

ここでは，3つのインスタンス変数同士の部分集合関係を記述しているが，例えば，この内の3番目は，インスタンス変数「s位置」の値域「rng s位置」は，インスタンス変数「s管制」の定義域「dom s管制」の部分集合であるという制約を記述している．

もちろん，インスタンス変数同士の制約は「対象問題分野」の知識がなければ記述できないのであるが，定石に基づいて「写像型を使ったインスタンス変数の定義域や値域に部分集合関係はないか?」という疑問を持ち，調べることによって制約を発見して記述するきっかけになる．この定石も，3節の「定石・手筋の評価」の結果，使われていることが分かった．

手筋は，下記のように分類した．

1.2手筋
- 1.2.1 成功手筋
- 1.2.2 失敗手筋

1.2には手筋として成功手筋と，失敗手筋を記述した．本稿では，成功することが多い手筋を成功手筋と呼び，失敗することが確実，または失敗することが多い手筋を失敗手筋と呼ぶ．

この中には，定石と呼んでよいものもかなりあるが，仕様中のほとんどの個所で使えるパターンが多く，手筋とした．

以下の定石・手筋の説明では，紙幅の都合上ごく一部しか紹介していないが，全部で定石・手筋は129個ある．このため，定石や手筋を一意に識別するため，VDMの証明に関する教科書(文献9)の証明IDの命名法を参考にして，定石と手筋のIDを付けることにした．この命名法で付けられたID名は，人間に分かりやすい名前を短縮したものなので，人間が読むにはやや分かり難くなるが，証明IDや囲碁の定石名などと同様に，最終的に3千から2万個ほどになる可能性があるため，唯一性と検索容易性を優先した．

成功手筋は，さらに以下のように分類し，整理した．すなわち，

[2]写像の定義域・値域は，キーとデータの対応を持つ表だとイメージすればよい．キーの集合が定義域であり，データの集合が値域である．

1.2.1.1　物の集まりの型｛集合，写像，列｝を使う手筋
1.2.1.1.1集合を使う手筋
1.2.1.1.2写像を使う手筋
1.2.1.1.3列を使う手筋

さらに，集合，写像，列を使う手筋は，内包式[3]を使う手筋と，限量式[4]を使う手筋と，その他の手筋に分けた．

「1.2.1.1.2写像を使う手筋」の中には，「1.2.1.1.2.1.2 写像に要素を追加する」手筋(IDTSCMCR_Append_++)があり，写像の上書演算子++と写像の旧値を使用し，写像の旧値と追加する写（maplet）を写像化したもの(写を'｛'と'｝'で囲うと写像になる)を上書(定義域のキーが重複した場合，演算子++の右辺により上書き)した写像が，追加後の写像と等しいことを示している． この手筋に従えば，事後条件の記述と，操作本体の記述が容易になる．また，この定石は，3節の「定石・手筋の評価」の結果，かなり使われていることが分かった．

```
instance variables
public s許容数 : 許容数 := {|->};

public 委託する: 空域 * nat ==> 許容数
委託する(a空域, n) == (
    s許容数 := s許容数 ++ {a空域 |-> n};
    return s許容数
)
pre
   a空域 not in set dom s許容数
post
    s許容数 = s許容数~ ++ {a空域 |-> n};
```

失敗手筋は，以下のように分類した．

1.2.2.1 ブール演算で注意すべき失敗手筋
1.2.2.2 集合で注意すべき失敗手筋

「1.2.2.1 ブール演算で注意すべき失敗手筋」の中の「1.2.2.1.2 含意演算子と全称限量子 forall の組合せで注意すべき失敗手筋．」は「forall e in set S & false となる式で，S が空集合の場合は，forall 式は常に true になる．」(手筋 ID TFB_tautology)というものだが,VDM が使用している一階述語論理の日本語テキスト中で，筆者の知る限り,この定理に言及したものは無いため,VDM++仕様記述の初心者は必ず間違えるため手筋とした．

1.2 分析についてのアイデア

分析は，仕様を分析し欠陥を修正する．VDM++の分析で利用できるツールの機能は，内部矛盾がないことを検証する検証機能として組合せテストと証明課題生成があり，仕様が顧客の要求を満たしているかを検査するため回帰テスト機能がある．このため，次のように分類した．

2 分析
2.1 検証
2.1.1 組合せテスト
2.1.2 証明課題生成
2.2 妥当性検査

[3] 内包式はデータの範囲と条件を指定して物の集まりを作り出す式である．
[4] 限量式は，物の集まりのすべての要素が条件を満たすか，ある要素が条件を満たすかを判定する式である．

2.2.1 回帰テスト

例えば,「2.1.2 証明課題生成」の定石(ID Veri_PO)は, 教科書には記述されていないツールの証明課題生成結果をどう使うかを, VDMTools 設計者の Peter Gorm Larsen 教授にインタビューして得た結果を記述している.
すなわち,**「ツールで生成された証明課題は, VDM 本来の手法としては証明するのであるが, 産業界の技術者向けに作られた VDM のツール群は証明機能を支援していない. そのため, 生成された証明課題をレビューすることにより, 未記述の事前条件や不変条件を見つけ, 仕様の品質を高める. 」**

1.3 ツールについてのアイデア

VDM++を支援するツールは, Overture ToolsとVDMToolsとvdmjの3つあり, 各々の長所を組合せて使用すると効果的である. そこで, 以下のように分類した.

3. ツール
  3.1 Overture Tools と VDMTools と vdmj の協働
    3.1.1 ツールによるエラーの修正
      3.1.1.1 エラー特定の手筋
      3.1.1.2 テスト実行の手筋
      3.1.1.3 テスト結果の比較の手筋
    3.1.2 ツールによる回帰テスト
    3.1.3 ツールによる組合せテスト

例えば「3.1.1.2 テスト実行の手筋」(ID TU_EC_PE_VT)は,**「Overture Toolsは, エラーがあるとエラーの無い部分の実行もできない. VDMToolsは, 完成した部分からテスト実行できる. 構文エラーや型エラーがある時でも, エラーの無い実行可能な部分をテスト実行したい場合はVDMToolsを使う. 」**であり, アジャイル的に仕様を組み立てていく時はVDMToolsの方が有効であるが,「3.1.2 ツールによる回帰テスト」手筋の中の「3.1.2.1 ツールによる回帰テスト使い分けの手筋」(ID TU_RT_OT_VT)によると,**「Overture Toolsは比較的簡単に回帰テストケースを作成できるので, テストケースがさほど多くない回帰テストはOverture Toolsで行い, テストケースが非常に多い場合はVDMToolsを使う. Overture Toolsの回帰テストライブラリーは, VDM++の名前がtest(大文字も可)で始まる操作をテストケースとして自動的に探索し実行してくれるため, 回帰テストの行数が少なくて済む. ただし, Overture ToolsとVDMToolsの回帰テスト・ライブラリを切り替え, テストケース記述もVDMTools用に追加しなければならない. 一方, VDMToolsの方が, 大量の回帰テストケースを実行できた実績がある. 」**であり, アジャイル的な仕様作成が大体終わり, 回帰テストを使う最初の段階では**Overture Tools**を使う方が有効であり, 大量のテストケースによる回帰テストを行う時は**VDMTools**の方が安心であることが分かる.

1.4 マニュアルについてのアイデア

マニュアルの手筋の分類は以下のとおりである.

4.マニュアル
  4.1 Overture Tools のマニュアル
  4.2 言語仕様の要約版
  4.3 VDMTools のマニュアル
  4.4 vdmj のマニュアル

これらを要約すると,VDM++の初心者は「日本語マニュアルのため読みやすい」手筋「4.3 VDMTools のマニュアル」(ID MU_VT)を使い,実際に仕様を書く時は「4.2 言語仕様の要約版」(ID MU_OT_Quick)を参照すればよい.
VDM10 などの最新の情報を調べるには「4.1 Overture Tools のマニュアル」(ID MU_OT)が必要になり,「4.4 vdmj のマニュアル」は「GUI 版はなく,コマンド版のマニュアルであり,プロ向きの記述である」(ID MU_VJ)は,VDM++初心者には必要ないが,VDM++上級者には必要になることがある.

2. 主要な定石と手筋

1節で紹介した定石と手筋以外の、3節「定石・手筋の評価」で使われている主要な定石と手筋を,紙幅の許す範囲で紹介する.

1.2.1.1.2.1.3 写像から削除する手筋

種類による分類	式による分類	式に使える名前	**定石 | 手筋 | 規則 | 注意事項**}ID	{**定石 | 手筋 | 規則 | 注意事項**}ID**説明**
手筋	**<-: dom**	-	TSCMCR_Delete	

写像から要素を削除する手筋で，写像型の定義域削減演算子**<-:**と，削除する写像の定義域集合を得る演算子**dom**を使用して，写像の旧値から削除本の要素を削除したものが，削除後の写像と等しいことを示している。

例 *1.2.1.1.2.1.3.1:* 写像「*i*蔵書」の旧値から削除本の要素を削除したものが、削除後の写像*i*蔵書と等しいことを示している例。

```
types
public 蔵書 = map 蔵書ID to 本;
public 貸出 = inmap 利用者 to 蔵書;

instance variables
private i蔵書 : 蔵書 := {|->};
private i貸出 : 貸出 := {|->};
inv 蔵書に存在する(merge rng i貸出, i蔵書);

operations
public 蔵書を削除する: 蔵書 ==> ()
蔵書を削除する(a削除本) == is not yet specified
post i蔵書 = dom a削除本 <-: i蔵書~;
```

例 *1.2.1.1.2.1.3.2:* 写像「*s*許容数」の旧値から空域集合に対応する要素を削除したものが削除後の写像「*s*許容数」と等しいことを示している。

```
public 使用をやめる : 空域 ==> () --TSCMCR_Delete
使用をやめる(a空域) ==
   s許容数 := {a空域} <-: s許容数
pre a空域 in set (dom s許容数 ¥ dom s管制)
post s許容数 = {a空域} <-: s許容数~;
```

1.2.1.1.1.1 集合に要素を追加する手筋。

種類による分類	式による分類	式に使える名前	定石 | 手筋 | 規則 | 注意事項}ID	{定石 | 手筋 | 規則 | 注意事項}ID説明
手筋	union	旧値が使えるのは、事後条件式の中だけである。	TSCSCR_Append	

集合の合併演算子unionを使用してインスタンス変数に1要素を追加する手筋である

例 *1.2.1.1.1.1.1:* 集合の合併演算子unionを使用してインスタンス変数に*1*要素を追加する式の例

```
--操作本体で使う場合の例
i客DB := i客DB union {a新しい客}

–事後条件で使う場合の例
post i客DB = i客DB~ union {a新しい客}

public 出勤する : 管制官 ==> () --TSCSCR_Append
出勤する(a管制官) ==
    s当直集合 := s当直集合 union {a管制官}
pre a管制官 not in set s当直集合
post s当直集合 = s当直集合~ union {a管制官};
```

1.2.1.1.1.2 集合から要素を削除する手筋

種類による分類	式による分類	式に使える名前	定石 | 手筋 | 規則 | 注意事項}ID	{定石 | 手筋 | 規則 | 注意事項}ID説明
手筋	¥(集合の差)	旧値が使えるのは、事後条件式の中だけである。	TSCSCR_Delete	

集合の差演算子¥と集合の旧値を使用し，旧値と削除する要素(を集合化したもの)の差が集合の値と等しいことを示した例である。

例 *1.2.1.1.1.1.2:* 集合の差演算子¥と集合の旧値を使用し、旧値と削除する要素(を集合化したもの)の差が集合の値と等しいことを示した例である。

```
--操作本体で使う場合の例
i客DB := i客DB ¥ {a削除する客}

–事後条件で使う場合の例
post i客DB = i客DB~ ¥ {a削除する客};

public 退勤する : 管制官 ==> () --TSCSCR_Delete
退勤する(a管制官) == s当直集合 := s当直集合 ¥ {a管制官}
pre a管制官 in set s当直集合 ¥ rng s管制
post s当直集合 = s当直集合~ ¥ {a管制官}
```

3. 定石・手筋集の評価

ここでは，本定石・手筋集がどの程度有効であるかを，定石と手筋の収集に使っていなかった参考文献5と6を使って評価した．

定石と手筋	文献5 化学プラント仕様	文献6 航空管制システム 最上位仕様
定石・手筋使用率	100%	100%
写像に上書演算子を使って追加 TSCMCR_Append_++		6
写像から定義域削減演算子<-: を使って削除 TSCMCR_Delete		3
集合の合併演算子unionを使って追加 TSCSCR_Append		1
集合の差演算子¥を使って削除 TSCSCR_Delete		1
データが意味的に正当か確認 JIT_wf_1	3	
写像のドメイン同士に部分集合関係がある JII_ibV_subset2	1	
集合内に存在することを確認 TSCSQ_exsists_set	3	

表6. 定石・手筋集の評価

上記2つのVDM++仕様については，すべての不変条件，事後条件，事前条件が，本定石・手筋集に記述されたものであった．また，全部で19個所ある定石と手筋の適用個所で使われている定石と手筋はわずか7種類であり，今後より多くのモデルのデータを集めることにより，主要な定石と手筋を特定できる可能性が強いことが分かった．

当然のことながら,集合型を中心とした仕様記述は集合に関する手筋を使い,写像型を中心とした仕様記述は写像と集合に関する手筋を多用するので,成功手筋の中で集合と写像と列[5]で手筋を分類したのは有効だった．

これらのVDM++仕様は定石と手筋の使用率が高いものを選んだわけではなく，たまたま最初に評価した2つのVDM++仕様だった．

現在のデータ数では，まだ「本定石・手筋集の有効性が高い」と言い切ることはできないが，一応定石と手筋集は役立ちそうだといえる．

4. 関連研究

文献 1. 2.など,関連ありそうな情報を探したが,今のところ囲碁や将棋やチェスの定石と手筋という観点から「定石と手筋集」を作成・整理しようという関連研究は無かった．

[5] 列型を使った仕様はまだ評価していないが,過去の経験から,確実に有効であろうと考えている．

文献 3. 4. 6.7.などの VDM の証明に関する教科書は,本研究のヒントになる情報は多いが,VDM 仕様の証明が主題であり，仕様記述の際に活用しようというものではない．文献 12.も,囲碁などの棋譜や音楽の楽譜からの発想で，証明譜を使い,それを証明に活用しようというものであり,やはり仕様記述の際のヒントにしようというものではない.

5. 結論と追加研究

産業界でのVDM++仕様記述で一番困難な，不変条件，事後条件，事前条件のパターンを定石・手筋集として試作した.

本定石・手筋集の有効性はある程度確認できたので，今後，実用的なモデルの定石・手筋使用率をさらに調べるていく予定である．調査対象モデルとしては，文献5の原子力追跡機モデル，筆者のカレンダー・ライブラリ， 文献9のNetwork Security Policy Modelなどを考えている.

VDMの証明についての教科書である文献7と文献9は，不変条件，事後条件，事前条件例を見つけ出すためにかなり役立つことが分かったので，重点的に調べていく予定である.

定石・手筋から仕様の骨格を生成でき,ツールに組み込んでいくことも可能である.例えば,Overture ToolsのEdit Template機能(文献 13.参照)で,下記のような「写像に要素を追加する．」手筋(ID TSCMCR_Append_++)用の”appendpp”という名前のテンプレートを記述しておくと,

```
types
public ${TypeName} = ${TypeName0};

instance variables
public ${instancevariable} : ${TypeName} := ${statement};

operations
public ${operationname}: ${argTypes} ==> ${resultType}
${operationname}(${arg1}, ${arg2}) == (
        ${instancevariable} := ${instancevariable} ++ {${arg1} |-> ${arg2}};
        return ${instancevariable}
)
pre
        ${arg1} not in set dom ${instancevariable}
post
        ${instancevariable} = ${instancevariable}~ ++ {${arg1} |-> ${arg2}};
post
        ${instancevariable} = ${instancevariable}~ ++ {${arg1} |-> ${arg2}};
```

エディター画面でappendppとタイプしてからコントロール・キーとスペース・キーを同時に押すことにより,テンプレートのテキストが挿入され,エディターで${instancevariable}部分を**s許容数**に変えると,すべての${instancevariable}が**s許容数**に変わり,下記のようになる.

```
types
public TypeName = TypeName0;

instance variables
public s許容数 : TypeName := statement;

operations
public operationname: argTypes ==> resultType
operationname(arg1, arg2) == (
        s許容数 := s許容数 ++ {arg1 |-> arg2};
```

```
        return s許容数
)
pre
        arg1 not in set dom s許容数
post
        s許容数 = s許容数~ ++ {arg1 |-> arg2};
```

作成した手筋テンプレートはまだ1つであるが,今後,徐々に追加していく予定である.

なお, 現在コンサルティング中のプロジェクトで本定石・手筋集の一部を教育し, 実プロジェクトで試行していく予定である.

参考文献

1. JÖRG ACKERMANN, FORMAL DESCRIPTION OF OCL SPECIFICATION PATTERNS FOR BEHAVIORAL SPECIFICATION OF SOFTWARE COMPONENTS, 2005
2. ANDREW IRELAND AND BILL J. ELLIS AND TOMMY INGULFSEN, INVARIANT PATTERNS FOR PROGRAM REASONING, 2004
3. CLIFF B JONES, CASE STUDIES IN SYSTEMATIC SOFTWARE DEVELOPMENT SECOND EDITION, 1990
4. CLIFF B JONES, SYSTEMATIC SOFTWARE DEVELOPMENT USING VDM, SECOND EDITION , 1990
5. ジョン・フィッツジェラルド他著, 酒匂寛 訳, VDM++によるオブジェクト指向システムの高品質設計と検証, 2010
6. JIM WOODCOCK, MARTIN LOOMES, SOFTWARE ENGINEERING MATHEMATICS: FORMAL METHOD DEMYSTIFIED, 2007
7. JUAN C. BICARREGUI, JOHN FITZGERALD), PETER A. LINDSAY, RICHARD MOORE, BRIAN RITCHIE, PROOF IN VDM: A PRACTITIONER'S GUIDE, 1995
8. 佐原伸, IPA, 対象を如何にモデル化するか?, 2013
9. JUAN C. BICARREGUI (ED.) PROOF IN VDM: CASE STUDIES, 1998
10. VDM — THE VIENNA DEVELOPMENT METHOD, ANDREAS MULLER, APRIL 20, 2009
11. DANIEL JACKSON. SOFTWARE ABSTRACTION : LOGIC, LANGUAGE, ANALYSIS . THE MIT PRESS, CAMBRIDGE, 2006
12. Kokichi FUTATSUGI1, Joseph A. GOGUEN2, and Kazuhiro OGATA: Verifying Design with Proof Scores, 2008
13. PETER GORM LARSEN, KENNETH LAUSDAHL, PETER TRAN-JØRGENSEN, JOEY COLEMAN, SUNE WOLFF, LU´IS DIOGO COUTO AND VICTOR BANDUR: OVERTURE VDM-10 TOOL SUPPORT: USER GUIDE, VERSION 2.4.6, 2017

日立グループにおける働き方改革の「カタ」
～マルチタスクに注目した設計・開発の改善プログラム～

八木 将計
株式会社日立製作所
masakazu.yagi.zd@hitachi.com

西原 隆
ゴール・システム・コンサルティング株式会社
nishihara@goal-consulting.com

真道 久英
ゴール・システム・コンサルティング株式会社
shindo@goal-consulting.com

村上 悟
ゴール・システム・コンサルティング株式会社
murakami@goal-consulting.com

渡辺 薫
株式会社日立製作所
kaoru.watanabe.lm@hitachi.com

要旨

本論文では，日立グループにおける働き方改革の「カタ(標準型)」を報告する．具体的には，ソフトウェアを含む設計・開発において，日立グループでの改善活動で得られた知見を整理・標準化したチームマネジメントA3というフォーマットとそれに基づく改善プログラムを提案する．提案手法では，QCD に様々な悪影響を及ぼすマルチタスクの低減を改善目標とし，マルチタスクを見える化するタスクボードと毎朝のスタンドアップミーティングで問題をあぶり出し，KPT による解決志向アプローチと TOC 思考プロセスによる原因追及アプローチの組合せで改善を推進する．このチームマネジメント A3 により，助言を得やすくなったなどの定性的効果だけではなく，残業時間 30%減や納期遵守率 27%増などの効果も得た．また，働き方改革の展開が容易になり，現在，日立グループにて1,000 人以上が関わる規模に拡大している．

1. はじめに

製品の短納期化・多機能化といった市場要求の変化などの背景により，ソフトウェアを含む製品の開発・設計において，従来の開発スタイルの維持では，会社の競争優位性の確保が難しくなってきている．そのため，ソフトウェア開発においても，常に「改善」が求められる．また，近年，「働き方改革」として，様々な業種・業態にて，働き方の見直しとそれによる生産性向上が求められており[1]，ソフトウェア開発でも例外ではなく，さらなる改善が必要になってきている．

このような流れの中，ソフトウェア開発の改善活動に取り組んでいる組織が多くあるものの，必ずしも十分な改善効果を上げているわけではない．なぜならば，ソフトウェア開発では，製品の特性や開発者のスキル，ビジネス環境など，様々な要素が絡んでおり，その改善効果は，SEPG，SQA，コンサルタントなどの支援者や改善推進者の力量に依存してしまっている．

一方，ソフトウェアを含む開発・設計の現場は，多くの場合，一つの開発・設計業務に集中しているわけではなく，さまざまな業務を切り替えながら仕事をする「マルチタスク」となっている．このマルチタスクは，ソフトウェア開発における QCD に悪影響を及ぼすことが多い[2][3][4]．

日立グループでは，2011 年から開始した企業変革である Hitachi Smart Transformation Project (通称：スマトラ)の一環として，設計・開発の「働き方改革」に取り組んできた．その働き方改革では，上述のマルチタスクの低減に注目することで十分な改善効果を上げることができることがわかってきた．そこで我々は，このマルチタスクに注目した改善手順を日立グループの働き方改革の「カタ(標準型)」として整理・標準化した．

本報告では，この働き方改革の「カタ」である「チームマネジメント A3」と，それに基づく改善プログラムを提案する．このチームマネジメント A3 に基づく改善プログラムを用いることで，ある程度，改善支援者や推進者の力量に依存せずに改善活動を進めることが出来ており，現在まで日立グループにて 1,000 人以上が関わる規模に拡大している．

2. 問題解決手法における課題

組織の問題解決手法や改善手法は，さまざま存在しているが，概ね QC ストーリー(表 1)のような手順となる．

表 1 問題解決型 QC ストーリー

1)テーマの選定	あるべき姿と現状のギャップから改善のテーマを決定する
2)計画の立案	改善の計画を立案する
3)現状の把握	事実のデータを収集し，現状を把握する
4)目標の設定	把握した現状と選定したテーマに基づき，具体的な目標値を決定する
5)要因の解析	より深い分析により，問題の原因を特定する
6)対策の立案・選定	特定した問題の原因を取り除く対策を立案し，選定する
7)効果の確認	対策を実施し，その有効性を確認する
8)歯止め	問題が再発しないように歯止めを行う

我々は，上記の手順について，下記のような課題があるため，改善の効果が改善支援者や推進者の力量に依存してしまうと考えている．

【課題 1】:テーマに何を設定してよいかわからない
【課題 2】:現状把握が主観的になりやすい
【課題 3】:情報過多となり，根本問題がわからない
【課題 4】:根本問題に対する解決策を導出できない

3. 日立グループにおける働き方改革の「カタ」

3.1. 働き方改革の「カタ」:チームマネジメント A3

日立グループでは，全社の企業改革であるスマトラ活動の一環として，「働き方改革」を推進してきた．その経験から，特に設計・開発においては，以下のような進め方をすることで，前章に示す問題解決の課題を解決することができ，結果，改善支援者や推進者の力量にある程度依存しない形で改善効果を得ることが出来るという知見を得た．

【課題 1 の対策】:マルチタスクの低減
ソフトウェア開発の QCD に悪影響を与える「マルチタスク (詳細は 3.3 節)」の低減をテーマとして固定する．

【課題 2 の対策】:タスクボードとスタンドアップミーティング[5][6]
一般的に，上述のマルチタスクは感覚的には認識されているが，客観的にどのようにマルチタスクになっているかが分かっていないことが多い．そこで，設計・開発チーム内のタスクを見える化・共有するツールであるタスクボードとスタンドアップミーティング(詳細は 3.4 節)を実施することで，客観的にマルチタスクの状況把握を行う．

【課題 3 の対策】:解決志向アプローチ
タスクボードとスタンドアップミーティングにて，マルチタスクの状況を見える化すると，様々な問題点が浮き彫りになってくる．しかし，改善活動開始当初は，明らかになる問題の数が多すぎるため，本腰をいれて取り組むべき根本問題が簡易な問題に埋もれてしまっている場合が多い．そこで，それら簡易な問題について，解決志向アプローチを用いた改善を行うことで，枝葉を取り除く．我々は，この解決志向アプローチとして，KPT (良い点:Keep，問題点:Problem，改善案:Try) を用いた週次のふりかえりを用いている[7][8](詳細は 3.5 節)．

【課題 4 の対策】:原因追求アプローチ
上記の解決志向アプローチを進めて行くと，どうしても解決できない根の深い問題が残ることになる．その問題について，原因追求アプローチで原因の深堀りし，根本的な問題解決を行う．
我々は，この原因追及アプローチとして，組織の問題解決に用いられる，TOC (Theory of Constraints:制約理論) 思考プロセス[10][11]を応用している(詳細は 3.6 節参照)．

上述の知見に基づき，我々は日立グループにおける働き方改革の「カタ(標準型)」として，整理・標準化している．具体的には，「チームマネジメント A3」と呼ぶ A3 一枚(PowerPoint で 8 スライド)の標準的な問題解決フォーマットを規定した(図 1)．チームマネジメント A3 フォーマットの概要を以下に示す．

【1 枚目】:タスクボード&スタンドアップミーティングの運用ルール
【2 枚目】:よかったこと・工夫点 → **解決志向アプローチ**
【3 枚目】:現状の問題分析
【4 枚目】:重点改善項目
【5 枚目】:根本原因分析
【6 枚目】:対策案検討
【7 枚目】:対策の実行計画
【8 枚目】:実施評価と横展開
原因追求アプローチ

図 1 チームマネジメント A3 フォーマット

3.2. チームマネジメント A3 による働き方改革の進め方

前節に示すチームマネジメント A3 を用いた改善プログラムは，4〜8 名程度のチームによる活動を基本とし，チームの改善推進者が改善支援者(TOC の有識者)の支援の下，チームマネジメント A3 にしたがって，自チームの改善を推進していく．活動の全体像は，以下のようになる．

① 準備(期間:1〜3 ヶ月)
2〜3 回のワークショップを通じて，チームマネジメント A3 の手法の学習する．

② チームマネジメント A3 導入(期間:6 ヶ月)
6 ヶ月間，チームにてタスクボードを運用する．また，隔週 1 時間，チームの改善推進者が TOC の有識者と面談しながら，チームマネジメント A3 を 1 枚目から順番に作成する(チームよって進捗は異なるが，典型的には隔週で 1 枚のペースとなる)．

- 導入〜定着(期間:前半 3 ヶ月程度)
  タスクボードの運用をチーム内に定着させ，以下の定性的効果を得る．
  - ✓ 仕事内容の共有／コミュニケーションの円滑化
  - ✓ トラブルの早期発見と早期対策／相互支援
  - ✓ 仕事の計画ミス・抜け漏れ防止
- 根本原因分析〜チーム個別改善施策検討(期間:後半 3 ヶ月程度)
  タスクボードの運用のふりかえりから，チームマネジメント A3 に則り以下を実施する．
  - ✓ 十分な設計時間を確保する上での重要な阻害要因の特定と重点改善項目の決定
  - ✓ 根本原因分析とチーム個別改善施策および期待効果の検討・合意
    (改善施策には，TOC の手法を始め，アジャイル開発の手法，XDDP (eXtreme Derivative Development Process)[12][13]，リーン製品開発[14]などの各種手法を適宜取り入れる)
  - ✓ (可能な場合)チーム個別改善施策の実行〜成果の確認

③ チーム個別改善活動(期間:6 ヶ月〜12 ヶ月)
上記のチーム個別の改善施策を本格的に実行し，定着させる．

④ 継続的改善
上記のサイクルをチーム単独で(もしくは最小限の TOC 有識者の支援で)継続実行する．

3.3. マルチタスクの QCD に対する影響

ソフトウェアを含む開発・設計の現場は，多くの場合，一つの開発・設計業務に集中しているわけではなく，複数の開発を抱えていたり，突発に発生する不具合に対応したり，問い合わせ対応や会議などのその他さまざまな業務を切り替えながら仕事をする「マルチタスク」となっている．ここでの「マルチタスク」とは，目的や内容の異なる複数の作業タスクを切り替えながら仕事をすることとする．つまり，内容的に一つの塊になっているタスクを中断して，別のタスクを実行し，もう一度中断したタスクに戻ってくるような仕事の仕方のことをさす．

このマルチタスクは，タスクの切り替えに段取り時間などのオーバーヘッドがあるため，全てのタスクをシングルタスクでシーケンシャルに実行するよりも多くの時間がかかってしまい，工数圧迫や遅延の要因となり，開発コスト増大にもつながる[5][15]．また，一般的に人間の思考はシングルタスクであるため，マルチタスクになると生産性が 40%低下するとも，IQ が 15 ポイント低下するともいわれ

ている[2]. 加えて，タスク切り替えで集中力が削がれることで，ケアレスミスなどの作業品質の低下を招き，工数圧迫によるプレッシャーなどで，成果物や製品の品質を低下させる.

さらに，過度なマルチタスクは，精神疾患を生み出すともいわれており，QCD の側面だけでなく，組織全体の能力が奪われることにつながる[3][4].

チームマネジメント A3 による働き方改革では，このマルチタスクの低減を改善の目標と設定している.

3.4. タスクボードとスタンドアップミーティング

マルチタスクを見える化する有効な方法として，タスクボードとスタンドアップミーティングがある[5][6]. タスクボードとは，その週の予定タスクを貼る「ToDo」欄，本日実施しているタスクを貼る「Doing」欄，完了したタスクを貼る「Done」欄を最小限とした，ボードである (図 2). スタンドアップミーティングは，毎日 15 分程度のチームでの作業タスクの確認・共有を目的とした立って行うミーティングである. チームメンバー各自のタスク(付箋など)をタスクボードに貼り，毎日のスタンドアップミーティングで共有することで，マルチタスクも含め，チーム内のタスクの状況を見える化することが出来る.

タスクボードの運用は，以下のようなサイクルとなる.

① 週に一度，自分自身の一週間分の作業タスクを一日未満に分解して計画し，付箋に書き，タスクボードに貼り付ける.

② その計画に対して，毎日のスタンドアップミーティングにて，付箋を動かしながら，作業状況や問題点をチーム内で共有，問題発見を迅速にし，解決を早める.

チームマネジメント A3 では，まず，このタスクボード&スタンドアップミーティングで現状を見える化，特に，多くの場合にマルチタスクの大きな要因となっている「飛び込みタスク(計画外タスク)」がわかるような運用を行う. ここで，飛び込みタスク(計画外タスク)とは，週次計画で計画しておらず，週中でやならければならなくなったタスクであり，他者からの依頼の他に自分自身の計画漏れも含むタスクである.

担当者	TO－DO （今週の予定）	Doing （本日のタスク）	Done （完了）
AAAさん	･･･資料作成 ･･･資料作成	･･･資料作成 ･･･打合せ	･･･資料作成 ･･･打合せ
BBBさん	･･･資料作成 ･･･打合せ	･･･打合せ	･･･資料作成
CCCさん	･･･資料作成 ･･･打合せ	･･･資料作成	･･･打合せ

図 2 タスクボードの例

3.5. 解決志向アプローチ

解決志向アプローチとは，問題やその原因を追求するのではなく，改善策とその効果に焦点を当てる心理療法を基礎とした手法であり，トライ&エラーでふりかえりながら簡単に改善できるものは改善してしまおうというものである.

チームマネジメント A3 では，タスクボード&スタンドアップミーティングで明らかになってきたマルチタスク，特に飛び込みタスクの要因などさまざまな問題が明確になるので，KPT などを用いた週次のふりかえりで手をつけやすい問題から改善していく[7][8].

3.6. 原因追求アプローチ

原因追求アプローチとは，なぜなぜ分析に代表される，問題の原因を深堀して追求し，その根本原因を解決する方法である.

チームマネジメント A3 では，解決志向アプローチを繰り返しても解決できない深い根本問題について，原因追求アプローチを用いる. 具体的には TOC 思考プロセスを原因追及アプローチとして用いる.

TOC とは，エリヤフ M ゴールドラットが開発した企業利益を決定づける「制約条件」にフォーカスすることによって，全体最適を実現し，最小の努力で最大の効果をあげる経営管理手法である[16]. TOC 思考プロセスは，TOC における組織の問題解決のための手法である[9][10][11]. 表出している症状に個別対処するのではなく，組織の目標達成を阻害している本質的な中核問題(悪い方針や評価基準)を明かにし，それを解決した“あるべき姿”を描き，それを実現するためのプランを策定する体系的な原因追及アプローチの問題解決手法である.

3.7. チームマネジメント A3 の各項目内容

【1 枚目】:チームマネジメント運用ルール
チームマネジメント A3 フォーマットの 1 枚目は，タスクボードやスタンドアップミーティングの運用ルールである．運用ルールは各チームの個別事情があるため，それぞれの事情に合わせたカスタマイズをしてもらう．

【2 枚目】:チームマネジメント導入の工夫と成果
チームマネジメント A3 フォーマットの 2 枚目は，タスクボード運用での週次ふりかえりで出た解決志向アプローチの結果における良い点(KPT の Keep，Try)をまとめたものである．また，チームごとで独自に工夫・改善した点についてもまとめる．
まずは，タスクボードによるチームマネジメント導入の定性的効果を感じることで，本格的な業務改善のモチベーションを高める．

【3 枚目】:現状の問題分析
チームマネジメント A3 フォーマットの 3 枚目は，週次ふりかえりで議論し，2 枚目の解決志向アプローチでは改善しきれない，現状の問題点(KPT の Problem)を纏めたもの．できるだけ客観的にチーム内の現状を把握するために，定量的に纏める方がよい．

【4 枚目】:重点改善項目
チームマネジメント A3 フォーマットの 4 枚目は，3 枚目の結果を受けて，チーム内で取り組む改善項目とその理由である．3 枚目同様，できるだけ客観的にチーム内の現状を把握するために，定量的に纏める方がよい．ただし，客観的に見た場合に合理性に欠いていても，チーム内での合意形成を重視する．そのため，3 枚目とは直接関係のない改善項目でも構わないが，その後の進め方は慎重に行う必要がある．

【5 枚目】:根本原因分析
チームマネジメント A3 フォーマットの 5 枚目は，4 枚目の重点改善項目に対して，その根本原因を分析したものを纏める．フォーマットは，TOC 思考プロセスで根本原因分析に用いる対立解消図[9][10][11](図 3)などを用いて行う．この対立解消図は，「改善したいけどできない」というジレンマを整理し，そこに潜んでいる仮定(Assumption)を明かにする．

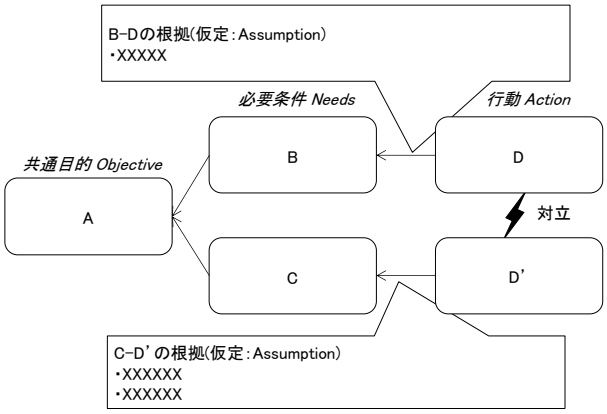


図 3　対立解消図

【6 枚目】:対策案検討
チームマネジメント A3 フォーマットの 6 枚目は，5 枚目の根本原因分析に対して，考えられる対策案を列挙し，効果と難易度から実行する対策を決定する．5 枚目にて対立解消図を用いている場合，ジレンマを発生している仮定に対する対策を考えることで，ジレンマを取り除く方法を検討する．

【7 枚目】:対策案実行計画
チームマネジメント A3 フォーマットの 7 枚目は，6 枚目で決定した対策案の実行計画である．まず，実行にあたり改めて，対策の目的，対策内容，成功基準を明確にする．ここでは，対策が実際に目的を達成し，成功したことを認識するために，成功基準が重要となる．ハッキリとそうであると判定できる基準でなければならず，できる限り定量的に記述するようにする．次に，その目的，対策内容，成功基準を実行する計画を期間，担当者なども含めて立案する．

【8 枚目】:実施評価と横展開
チームマネジメント A3 フォーマットの 8 枚目は，7 枚目を実際に実行した結果の評価を纏めたものである．定量的なデータとして成果を纏める．また，データの収集は追加負荷のかからないものを選ぶと良い．例えば，タスクボードの付箋のカウントなどである．また，残業低減などの目的に対応する「効果」に関するデータと，対策の適用累計などの「実施」に関するデータを取るとモチベーションにも繋がるためよりよい．ただし，実際には，チームマネジメント導入の 6 ヶ月では，成果が出ていないことが多いため，「こうなればよい」という予測の特性となる．

4. チームマネジメント A3 による働き方改革の効果

4.1. 飛び込みタスクの低減例

本手法による働き方改革の具体的な改善内容は，チームの特性などによっても全く違っている．本節では，個別全てのチームを示すことはできないので，典型的な改善パターンを述べる．特に，マルチタスクの大きな要因の一つである飛び込みタスクの低減について述べる．

本手法の導入初期の段階では，見える化と解決志向アプローチによって飛び込みタスクが減る．具体的には，初期はチームメンバーそれぞれがタスクをばらすこと(週次計画)に慣れていないため，自分自身の計画漏れによる飛び込みが多く(図 4)，解決志向アプローチによる改善を行うことで，その自分起因の飛び込みタスクが減り始めることが多い．良く用いられる飛び込み削減のための施策は，以下のようなものがある．

- ペアで週次計画をチェック(計画のピアレビュー)
- 月曜日に計画していたのを，金曜日に計画し，月曜日に見直す
- リーダーによる週次計画レビューの徹底
- タスクの不明点リストの導入
- リーダーからの作業指示のフォーマット化

その後，自分起因以外も含めた飛び込み削減のために解決志向アプローチで改善をしていくが，それだけでは，大きな改善効果が出なくなってくるので，その状況下で見えてきたチームの根本的な問題について，TOC 思考プロセスに原因追及アプローチを用いて，チーム個別の改善施策を検討し，実行する．典型的な飛び込みタスクの低減は，図 5 に示す形となる．

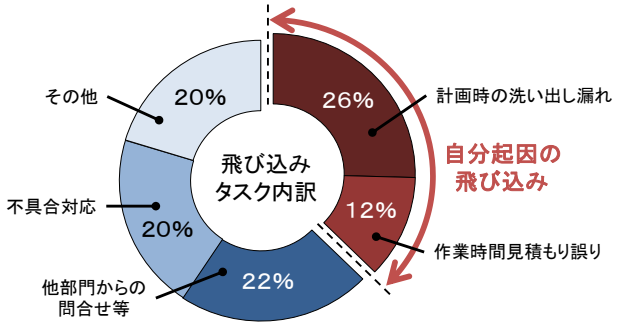


図 4 飛び込みタスクの内訳実績例

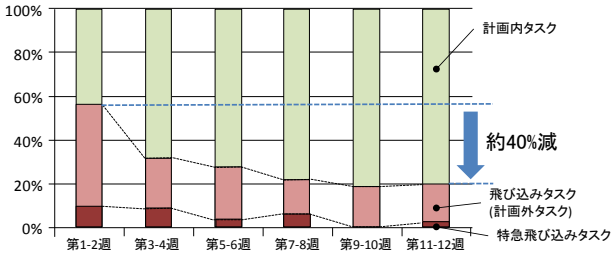


図 5 飛び込みタスク低減の実績例

4.2. 定性的効果

本手法による働き方改革のヒアリングに基づく定性的効果を以下に示す．主に，タスクボードを導入したことによる効果と，解決志向アプローチの効果が多い．

① 情報共有について

- 自分以外の担当者がどのような業務を実施しているのか分かるようになった．
- タスクボードへ貼り出して見ることにより，業務の優先順位を整理し易くなった．
- メンバー個人の作業量の偏りが把握し易くなってきた．
- 毎日実施することで，大きい項目を細分化する意識，および一つ一つ片付けて後戻りが無いようにする意識が強くなった．

② ミス防止について

- 業務計画時，各業務を細分化して遂行時間を見積もって仕事を進める行動ができるようになってきた．
- 数週間停滞していると忘れがちであったが，相手をフォローアップできるようになったとともに，予定作業として準備できるようになった．
- 保留になっている業務をリーダーが迅速に把握して，保留原因の問題解決が早くできるようになってきた．
- 助言を得やすくなった．
- 業務時間を正確に見積もるため，業務を細分化して本人が意識しない問題を指摘され，問題発生前に問題を解決できた．

③ 課題抽出について

- 飛び込み作業について件数，内容が分析でき，チーム内の飛び込み業務に対する考え方を整理できた．
- 設計手配業務の不定期な割込が，業務遅延要

因となっていることが判明した．手配業務は，専任者を決め，各設計者より適宜手配業務を移管することで手配業務による遅延時間は低減している．

- 作業内容を先送りした事を忘れてしまい，次のプロジェクトで問題になる事がある．忘れないようにするため欄を設置して注意を促すようにした．
- 業務遅延の要因（内容，自責，他責）を把握し，発生頻度を記録することで，頻度の高い問題や遅延要因に焦点を当てた改善活動ができた．

④ その他の成果

- 職場での会話が格段に増えた／今まであまり話さないと思っていた若手が実は結構積極的にキチンと話せることがわかった．
- 従来は打ち合わせの雰囲気が悪く・対立的な会話が顕著だったが，毎朝笑顔で情報共有し，気持ちよく仕事を開始できるようになった．
- 個人・家庭の事情を共有し，助け合う雰囲気が，より強くなった．時短勤務や家庭の事情による遅刻・早退等を責める雰囲気や気持ちが軽減された．
- 若手が仕事の段取りを覚えて自分で仕事をこなせるようになった．また報告や連絡の内容・タイミングが的確になってきた．

4.3. 定量的効果

本手法を全展開した部署（18 チーム）の 1 年間の活動にて，下記のような大きな定量的な効果を得ている．ただし，当該部署では本手法以外の施策にも取り組んでおり，それらの総合的な結果である．また，それらの相乗効果があるため，本手法の効果のみを切り出すのは困難と考える．

- 設計 Fix 以降の変更手戻り時間：70%削減
- 納期遵守率：71% → 98% (内部管理指標)

上記以外の部署においても，下記のような定量的成果を得ている．ただし，仕事の変動による影響が大きいこと，着手まもないチームも多く，全てのチームで定量的効果を検証できているわけではない．また，本手法に加え，並行して実施した他施策との相乗効果が含まれる．

- 残業時間(休出含)：720 時間/月 → 480 時間/月 (33%減)
- 納期遵守率：52% → 75% (内部管理指標)
- 納期遵守率：平均遅延日数を半減
- リードタイム：仕様決定までの問い合わせ数 20%削減による期間短縮
- リードタイム：5 日 → 1 日 (仕様変更対応日数)
- クレーム件数：6 件/案件数 70 件 → 0 件/案件数 135 件
- 生産性：1.5 倍 (対応案件数) ※人員および残業の増加無しに達成
- チームワークに関するアンケート Ocapi[17]：10 点/10 点満点 ※平均は 5～7 点程度．

4.4. 適用チーム数の推移

本手法による働き方改革を取り組んでいるチーム数の推移を図 6 に示す．本手法は，2014 年下期から検討を開始，試行を経て，2015 年下期から本格的に展開を開始した．図 6 に示すとおり，口コミでの評判も手伝って，2015 年下期から取組みチーム数が倍増している (現時点で，のべ 1,000 名以上が改善活動に参加している)．また，展開規模が拡大しているが，その間 TOC の有識者は 3～4 名と大きくは変動していない．そのため，チームマネジメント A3 で標準化したことによって，支援者の負荷を増やさずに，効率的に業務改善を進めることができると考えられる．

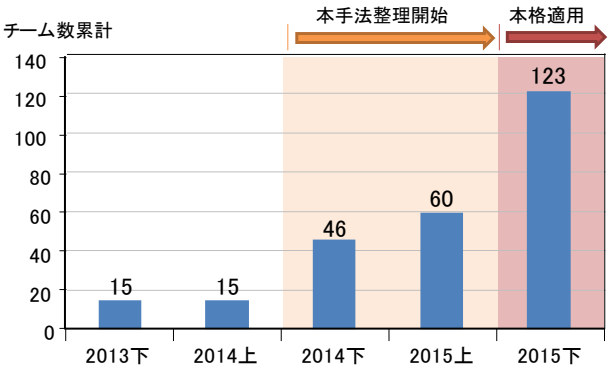


図 6 本手法適用チーム数累積の推移

5. まとめ

本報告では，日立グループにおける働き方改革の知見と経験を整理・標準化した「カタ(標準型)」として，チームマネジメント A3 と，それに基づく改善プログラムを提案した．提案手法では，ソフトウェアの QCD に悪影響を与えるマルチタスクに注目し，その状況をタスクボードとスタンドアップミーティングで明らかにする．そこから浮き彫りになる問題点について，まず KPT による解決志向アプローチでふりかえりながら，改善できるところから継続的に

改善を行い，解決志向アプローチでは解決しきれない根の深い問題について，TOC 思考プロセスによる原因追及アプローチを用いることで，問題原因の深堀を行って，根本原因の改善を行う．チームマネジメント A3 はこのステップをまとめたフォーマットであり，改善支援者や改善推進者の力量に依存しやすい改善活動において，ある程度属人性を排除し，改善効果を生みやすい構成としている．

結果，チームマネジメント A3 という標準フォーマットを用いることで，改善チーム数を倍増，のべ 1,000 名規模の活動に拡大することに成功した．

また，実際の改善効果として，このチームマネジメント A3 による改善プログラムの結果，「メンバー個人の作業量の偏りが把握しやすくなった」「タスクを一つずつ片付けることで後戻りを減らすようになった」「助言を得やすくなった」などの定性的効果に加え，残業時間 33%減，設計変更手戻り時間 70%減，納期遵守率 27%増，などの定量的効果もあがっている．

参考文献

[1] “首相官邸「働き方改革の実現」” http://www.kantei.go.jp/jp/headline/ichiokusoukatsuyaku/hatarakikata.html

[2] “Multitasking: Switching costs.” Research in Action. 20 March 2006. http://www.apa.org/research/action/multitask.aspx

[3] エドワード M. ハロウェル，” マルチ・タスクが「脳力」を奪う,” ハーバードビジネスレビュー，2005.

[4] 八木将計，西原隆，真道久英，村上悟，渡辺薫，“マルチタスクが「QCD」を奪う！ ～日立グループ 1,000 人に広がる 働き方改革 6 ヶ月プログラム～,” ソフトウェア品質シンポジウム 2016(SQiP2016)，2016.

[5] 西原隆，栗山潤，TOC/CCPM 標準ハンドブック―クリティカルチェーン・プロジェクトマネジメント入門，秀和システム，2010.

[6] “プロジェクトファシリテーション実践編:見える化ガイド,” http://www.objectclub.jp/download/files/pf/ManagementBySeeingGuide.pdf

[7] 天野勝，これだけ！KPT，すばる舎，2013.

[8] “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

[9] エリヤフ・ゴールドラット，ザ・ゴール 2 ―思考プロセス，ダイアモンド社，東京，2002.

[10] 村上悟，問題解決を「見える化」する本，中経出版，東京，2008.

[11] 岸良裕司，全体最適の問題解決入門－「木を見て森も見る」最強の思考プロセスー，ダイアモンド社，東京，2008.

[12] 清水吉男，失敗しない派生開発(Software People Vol.8)，技術評論社，東京，2006.

[13] 清水吉男，「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，東京，2007.

[14] 稲垣公夫，開発戦略は「意思決定」を遅らせろ! ―トヨタが発想し、HP で導入、ハーレーダビッドソンを伸ばした画期的メソッド「リーン製品開発」，中経出版，2012.

[15] エリヤフ・ゴールドラット，クリティカルチェーン―なぜ、プロジェクトは予定どおりに進まないのか?，ダイアモンド社，東京，2003.

[16] エリヤフ・ゴールドラット，ザ・ゴール―企業の究極の目的とは何か，ダイアモンド社，東京，2001.

[17] “組織変革プロセス指標 Ocapi,” https://ocapi.jp

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

阪井 誠
株式会社 SRA
sakai@sra.co.jp

要旨

本論文では Node-RED を導入したソフトウェア開発プロジェクトの経験者にアンケートを行い，ソフトウェア開発にどのような変化があったのかを調査した．経験者 8 人に対するアンケートの結果，品質と開発期間の評価が高かったものの，ツールの導入だけでは必ずしも効果が得られず，(1)ツールの知識やノウハウを共有すること，(2)特性を活かした設計で品質を作りこむこと，(3)実装を繰り返して常に確認すること，(4)上流から利用すること，が効果的なソフトウェア開発につながることが分かった．

プロセス改善を目的として多くの企業でツールが導入されているが，必ずしも成功していない，本研究の知見を参考に，より効果的なプロセス改善が行われることが期待される．

1. はじめに

「プロセスやツールよりも個人と対話を」とアジャイルソフトウェア開発宣言には書かれている[1]．これはソフトウェア開発におけるプロセスやツールの重要性を認めたうえで，個人との対話の重要性を宣言したものである．特に開発言語のように成果物と直接関わるソフトウェア開発ツールは，プロジェクトの成否に大きく影響する．このため，ツールを先に決定しておいて，経験のある開発者を採用することも多く行われている．このようにソフトウェア開発に大きな影響を与えるので，新しいツールを導入する機会は少なく，開発プロセスに大きな影響を与えるにも関わらず，その知見はあまり多くない．

その一方で，計算機の性能向上や技術の発展によって，効率的なソストウェア開発が可能になった．上流からのアプローチとしてはデータモデリングや画面・帳票まで自動生成する GeneXus[2]，モデルをベースにプログラムのパターンを定義する事で仕様書を実行する X-TEA Driver [3]などがある．逆に下流からのアプローチとしては，本論文で扱う Node-RED をはじめとして，高機能なシステムを簡単に作成するビジュアル開発ツールが数多く提案されている．

Node-RED は Visual IoT ツールと呼ばれており[4]，サーバーサイド Javascript 環境である Node.js およびそのミドルウェアである Express.js 上で動作する．Node.js はイベント駆動で動作し，いわゆるクライアント 1 万台問題が生じないと言われるほど高性能であるが，非同期処理プログラミングは容易ではない．Node-RED はプログラムコンポーネントであるノードを線でつなぐ事で順次処理と非同期処理を簡単に記述できる．また，多様で多機能なノードが豊富にある，一瞬でデプロイが可能，といった特徴から快適で効率的な開発を実現している．すでに日本でもユーザ会のほか[5]，書籍や雑誌の記事もあり[6]，IoT ツールとしてだけでなく，Web サービスなど様々な開発が行われている

これらのツールを導入するとソフトウェア開発プロセスは大きく変わると思われる．しかし，ツールの技術情報は多いものの，プロセスの変化に関する報告は GeneXus に関して開発プロセスを変えるものとして報告があるなど[2]，ごく一部に限られている．また，上流からのアプローチであれば開発規模も比較的大きく，コンサルタントを依頼できる可能性があるが，下流からのアプローチのツールを導入する場合は，当初は規模が小さいことが多いので外部に委託することが難しい．このことが，プロセスを大きく変えるツールの導入を難しくしている．

本論文では Node-RED によるソフトウェア開発の経験者にアンケートを行い，ソフトウェア開発のプロセスにどのような変化があったのかを調査し，その原因を考察した．アンケートを実施した経験者の開発対象は，社内サービス，プロトタイプ，テストダブル(ドライバ，モック，スタブ)，自社パッケージ，ユーティリティなどである．また，アンケートで調査した内容は，品質，コスト， 開発期間に対する選択式の評価および自由記述，要件定義，設計，プログラミング，テスト，リリースに対して従来との違いについて，自由記述で行った．

以降の章では，Node-RED の概要と長所・短所，アンケートの方法と結果，考察，そして最後にまとめの順に報告する．

2. Node-RED

2.1. Node-RED の基本

Node-RED は Visual IoT ツールと呼ばれ，Web ブラウザ上のエディタでプログラミングする．左の領域にある長円のプログラムモジュールをノードと呼び，中央の編集領域に配置し，ノード間を接続してフロー（処理）を作成することでプログラミングする．ノードに名前を付加できるが，単に配置するだけでも設定に応じた内容が表示される．

最もシンプルなフローは図1のようになっている．左端がインジェクトノードでデフォルトの timestamp がセットされている，このノードの左にあるボタン部分をクリックすると，現在時刻が msg オブジェクトとして次のノードに送られる（クリックでなく，定期実行することもできる）．右端にあるのがデバッグノードで，受け取った msg オブジェクトの一部またはすべてを右側のデバッグタブに表示する（コンソール出力も可能）．インジェクトノードに文字列等をセットすることも可能だが，ここでは中央のファンクションノードで“Hello world!”を msg.payload に代入している．

ここで，インジェクトノードの代わりに http ノード，デバッグノードと並列に http response ノードを接続すると，レスポンスボディに“Hello world!”を返す web API になる（図2）．デバッグノードの出力であるmsgオブジェクトは JSON 形式なので，コピーして別のインジェクトノードのデータとしてペーストすれば，後続の処理を手分けして開発できる．インタフェースがすべて msg オブジェクトに統一されているので，処理の変更や追加が容易なのである．

編集領域は複数のタブで構成されていて，複数のフロー間をリンクでつなぐこともできるので，大きなシステムは，タブを切り替えて開発することが可能である．フローを修正した場合は右上のデプロイボタンが赤くなり，これを押すことで瞬時に実行が可能になる．

標準で用意されているノードには，データをセットする

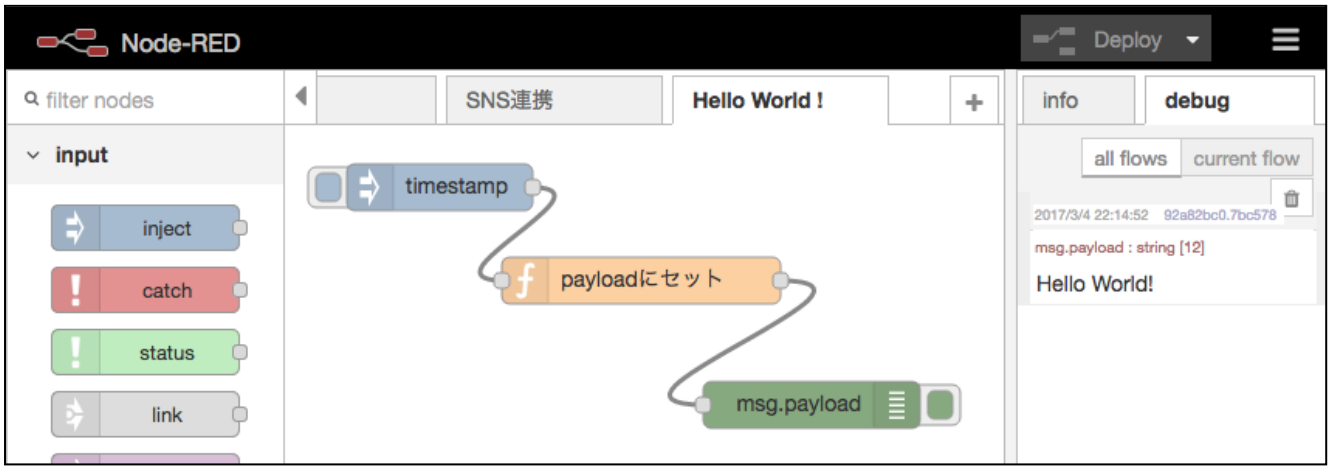


図 1 基本構造

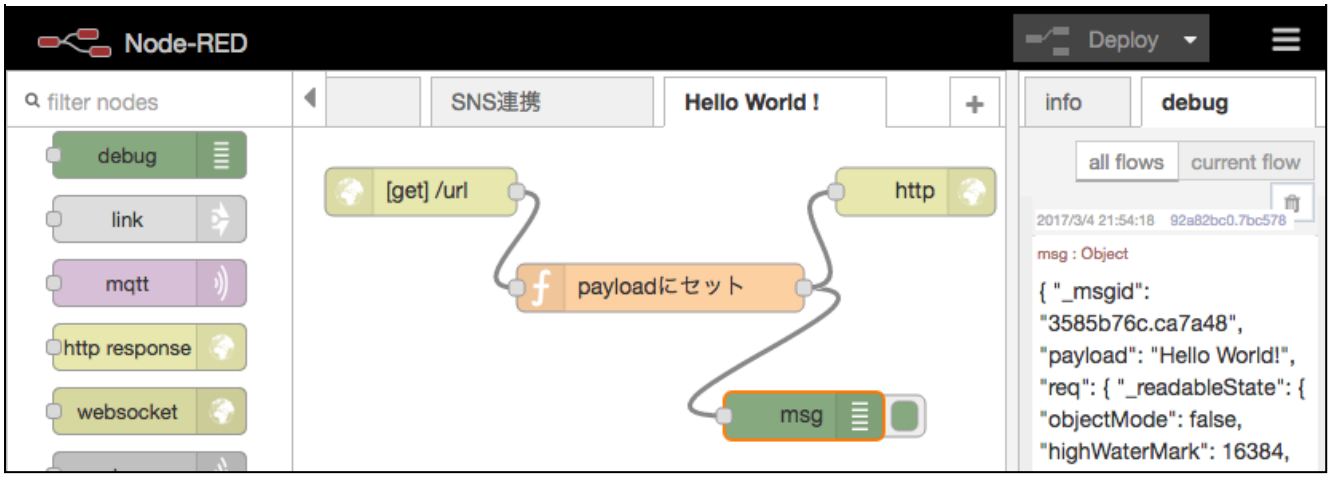


図 2 Web サービスの例

チェンジノードや，分岐処理をするスイッチノードなどがあり，簡単なプログラムであればプログラムコードを書かずに設定だけ実装できる．また，標準ノードのファンクションノードでは javascript でプログラムを書くことができるほか，フローの一部のノード群をサブフローとして一つのノードにまとめることができる．サブフロー以外にも Javascript と HTML でカスタムノードを作成できる．サブフローには入口，出口，名称以外は設定できないが，カスタムノードは標準ノードのように設定用の UI を持ち，多くのカスタムノードがライブラリとして npm(Node Package Manager)で公開されている．

2.2. 長所

Node-RED の長所をまとめると以下のようになる．

非同期処理が簡単に扱える：図1の様にノードを一列に並べると順次処理になる．図 2の http と表示されている http レスポンスノードと，msg と表示されているデバッグノードの様に，ノードの出力を複数のノードに接続すると非同期に処理が行われる．また，配列やオブジェクトを分割して非同期処理する split ノードがあるほか，ファンクションノードから node.send()メソッドで非同期に msg オブジェクトを送出することも可能である．

アルゴリズムが可視化される：ノードの名称とその順序で処理が可視化される．ノードに和名を付けたり，コメントノードでドキュメントを記述することも可能である．

多機能なノード（モジュール）：Node-RED には HTTP，TCP，UDP，WEBソケット，MQTT，Twitterなど各種通信が簡単に行えるノードが標準であるほか，さらに多くの通信やデバイス制御，DB アクセス，SNS，BOT などが数多くノードがコントリビュートされている．中でも Dashboard というライブラリは，Web 画面を簡単に作成でき，デプロイなしにデータの入出力ができるので，テスト用のドライバ，スタブ，モックなどいわゆるテストダブルの開発が簡単にできる．

デプロイが一瞬：コンパイルやビルドが不要で，デプロイが 1 瞬で実行できる．常に動作を確認しながら開発できるので，仕様通りに動作する部分を徐々に増やしながらインクリメンタルに開発することで，品質の高いソフトウェアを開発可能である．

再利用が容易：カスタムノードやサブフローによるモジュール化のほか，Node-RED 内でフローの一部をコピー＆ペーストできる．また，フローの一部や全体をエクスポートして，他の Node-RED 環境にクリップボードを介してインポートして簡単に利用できる．さらに，作業分担の界面にデバッグノードをつないで msg オブジェクトをダンプし，その値を設定したインジェクトノードで開始すれば繋がっていないフローを別々に開発できる．これらの方法を組み合わせると，複数人で効率的に開発できる．

2.3. 短所

Node-RED の長所をまとめると以下のようになる．

単体テスト：カスタムノードにユニットテストを組み込むことは可能だが，フローの一部をテストする標準的な方法がない．このため，フロー全体をブラックボックステストする必要がある．

発展途上：Node-RED の機能は日々進化しているが，（改善しつつあるものの）複数人で同時編集できないなど，まだ不十分なところがあるほか，一部のノードに不具合が残っている．

方式設計が重要：小規模なシステムは実装を優先しても問題ないが，多機能なシステムを小さな工数で作れてしまうので気づかないうちに複雑になり混乱することがある．規模が大きくなる場合は，従来のソフトウェア開発と同じように，msg オブジェクトや永続化対象のデータ構造や，ソフトウェアアーキテクチャをあらかじめ設計する必要がある．

ループが特殊：複数のデータを処理する場合，ファンクションノード内のループ，ファンクションノード＋node.send()，split ノード＋join ノードなどの方法があるが，複数のノードで順次処理を繰り返す場合は，フローがループ状になるので，慣れるまではわかりにくい．

マージ・保守に工夫が必要：従来の言語のように diff で差分をとったものを patch でマージすることができない．このためバージョン管理が容易でない．開発環境のコードを本番環境にマージするには手作業が必要になる．

3. アンケート

3.1. アンケート方法

Node-REDを用いた開発を行っている8人の経験者にアンケートを行った．

各経験者は Node-RED を用いて社内サービス，プロトタイプ，テストダブル（ドライバ，モック，スタブ），自社パッ

ケージ，ユーティリティなどを開発している．全てのプロジェクトはいわゆるウォーターフォール型開発の工程を持っていたが，ばらつきはあるものの厳格な工程完了の審査は行われていない．Node-RED は生産性が高いことから選択された．

アンケートはメールで依頼し，いわゆるQCDとプロセスの変化をアンケートした．品質，コスト， 開発期間に対しては 4 段階の選択式（重複選択可）の評価と自由記述でアンケートした．従来のプロセスとの違いは，要件定義，設計，プログラミング，テスト，リリースに対して自由記述でアンケートした．

3.2. アンケート結果

品質，コスト，開発期間のアンケートは選択肢毎の比率を集計し（図 3），コメントを確認した．複数選択を許しているので，合計は 100%でない．評価は総じて良かったが，複数選択なので評価の良かった点だけでなく，Node-RED の問題点も明らかにすることができた．

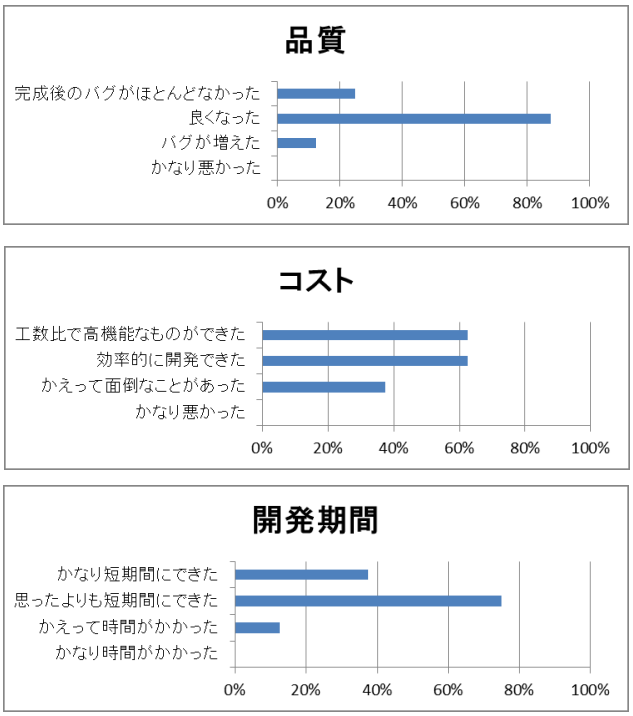


図 3 品質，コスト，開発期間のアンケート結果

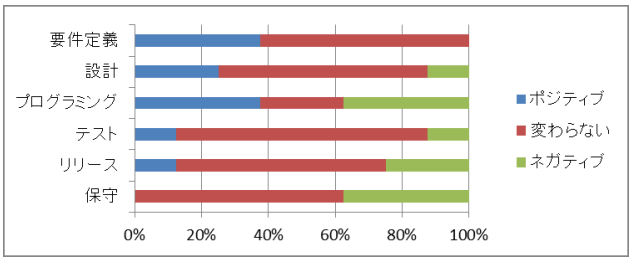


図 4 プロセスの変化のアンケート結果

品質に対して評価が良かった理由は以下のような内容だった．

- サクサクと実装，実行，確認・修整の作ってのループが良かった
- 内製にこだわるよりも品質が良い
- コード量が減った
- 試作に有効
- 非同期処理が容易（promise は間違いやすい）
- フローを意識してシンプルな作りになった

このようにデプロイが一瞬，高機能，ビジュアルなど NodeRED の長所が品質に対して評価が高かった．逆に評価が低かった理由は以下のような内容だった．

- 単体テストができない
- コード検索ができないのでバグが見つけにくい

これらはマイクロサービス化などの工夫はできるものの，Node-RED の短所が品質に影響を与えていたと考えられる．バグがほとんどなかったとしている回答者も 2 名いるので開発プロセスの違いが大きいと考えられる

コストに対して評価が高かった理由は以下のような内容だった．

- 非同期処理が容易
- 高機能なコンポーネントが多い
- 設計からテストまでシームレスにでき効率が良い
- 処理部に注力できた

このように長所が評価された．逆にコストに対して評価が低かった理由は以下のような内容だった．

- diff が取れない
- ドキュメントが少ない
- 必要なノードを探すのに時間がかかった
- 繰り返し処理に苦労した

Node-RED の短所が指摘されたが，下の 2 つは知識やノウハウを共有することで改善できると考えられる．

開発期間に対して評価が高かった理由は以下のような内容だった．

- 非同期処理が容易

- カスタムノード作成で効率化できた
- 開発のスピード感が半端ない
- 他の人のフローを簡単にインポートできる
- 実装にいきなり入れる
- 効率よく開発できる
- 大きな手戻りがなかった

ここでは品質で挙げられた点のほか，インポート・エクスポート，常に動作させることで高品質ななること，が挙げられた．逆に開発期間に対して評価が低かった理由は以下のような内容だった．

- 自作部分の作りで効率や保守性が変わる
- ドキュメントが少ない
- 品質の悪いノードがあった
- 必要なノードを探すのに時間がかかった

ここではサブフローやカスタムノードなど再利用の仕組みをどう使うか，あるいは全体の作りなど，Node-RED に適した設計が重要であること，発展途上のツールであることなどが指摘された

プロセスの変化は，自由記述で書かれた内容からネガティブ評価，ポジティブ評価，変化なし（およびどちらともいえない）の 3 段階を判断して集計した（図 4）．上流においてプロトタイピングが可能な点に対して評価が高かった反面，プログラミング，テスト，リリース，保守に関しては管理面の課題が多く挙げられていた．プロセスの変化の詳細は考察で検討する．

4. 考察

4.1. 品質，コスト，開発期間

アンケートの結果，特に品質と開発期間に対しての評価が良かった反面，コスト面ではある程度の経験が必要であることが指摘された．

Node-RED は高機能なノードを組み合わせて，とりあえず動かすことが簡単．デプロイも一瞬なので作りながら設計と実装を進めると，品質が確認されたフロー（処理)を増やしていくことができる．その反面，単体テストができない，コード管理が困難であるなどツールの特性を理解する必要があり，それがアンケートに反映されたと考えられる．

プロジェクト毎の特性で見ると，知識が不足している回答者を除くと，要件が曖昧で実装しながら詳細な仕様を確認していたプロジェクトの評価が高く，そのような実装を進めないと詳細仕様を決め難い開発に向いていると考えられる．ただし，データの定義やアーキテクチャはしっかりと設計する必要がある．

4.2. プロセスの変化

プロセスの変化に関しては，上流はポジティブな評価だったが，プログラミング・テスト・リリースに関しいては評価が拮抗するかやや低評価，保守に関してはネガティブな評価のみがあった．

上流に関してのポジティブな意見は以下のようなものであった．

- プロトタイプが早くできると説明も早く，意見を貰いやすい
- 曖昧な要求でもとりあえず作り始めることができる
- 作ったものから要件を確定していくことが可能
- 大まかな処理の流れをすぐにフローとして実装可能
- 設計とプログラミングのイテレーションが容易

このようなコメントをした回答者は，設計以降の問題を指摘しながらも，全体としては好意的な表現であった．具体的には，以下のような内容だった

- Inject ノードにより，実装したフローをすぐに動作させやすく，デバッグノードでの確認も容易
- flwos.json（Node-RED の保存ファイル）などほんの少数のファイルをリリースするだけでよく，管理しやすい
- ユニットテストはカスタムフローや API 単位しかできないが，不安は少なかった
- コード管理系だけどうすれば・・・Node-RED だけで開発環境になればすばらしいものになりそう
- （保守は）基本的に容易だが，開発環境を残しておかないと詳細な内容を確認し辛かった
- ユニットテストはカスタムフローや API 単位しかできないが，不安は少なかった

それ以外の設計以降でネガティブな表現をした回答者は，上流のプロセスに変化がないとしていた．また，QCD の項目で良い評価とされていた内容を書きながらも，Node-RED 流の開発スタイルをつかみ切れていない様子であった．

- 単体テストをどのように行うのかわかりませんでした
- 複数人数での開発が少し手間取る
- 外部リソースのノードの設定が外だしに出来なくて，

リリース後にとても煩わしかった
- Node-RED の癖にあわせた設計は必要．設計次第
- 簡単な反面，リリース後の不具合も増える可能性がある
- コード全体の検索が出来ないため，複雑なシステムは保守しにくくなる

これらから，Node-RED を利用する場合は，上流から実装を開始する．特性を理解して設計する．動作をきちんと確認しながら作り上げる．といったことが必要だと考えられる．

4.3. Node-RED の特性と結果

以上のことをまとめると，実装を進めないと詳細仕様が決め難いプロジェクトに有効であり，その実施には以下のような点が重要であると考えられる．

- ツールの知識やノウハウを共有すること
- 特性を活かした設計で品質を作りこむこと
- 実装を繰り返して常に確認すること
- 上流から利用すること

プロセスの視点でいうならば，Node-RED を導入する場合は，他のツールと同じように情報共有や教育が重要であるだけでなく，既存のプロセスをそのまま適用するのではなく，積極的に変更することが，プロセス改善につながると考えられる．

5. おわりに

アジャイルソフトウェア開発宣言には「計画に従うことよりも変化への対応を」と書かれている[1]．新しいツールや技術を導入するには，従来通りのプロセスのままでは難しく，適切に変化させなくてはならない．

本論文では Node-RED によるソフトウェア開発の経験者 8 人にアンケートを行い，ソフトウェア開発プロセスにどのような変化があったのかを調査し，その原因を考察した．アンケートの結果，新しいツールを導入する際には単にツールを利用するだけでなく，ツールの知識やノウハウを共有して特性を活かした設計を行い，実装と確認によって品質を上流から作りこむなど，主体的にプロセスを変更することが，プロセスの改善につながっていたことが分かった．また，Node-RED は発展途上であるので，バージョンアップのフォローも必要になるであろう．

書籍「ソフトウェア開発 55 の真実と 10 のウソ」には，“ソフトウェアの世界には，「一つのツールや技法が何にでも当てはまる」と信じる人が実に多い．”と書かれている[7]．ソフトウェア開発ツールにはそれぞれの特性があり，よく理解したうえでふさわしいプロセスで利用しないとその効果を十分に得ることはできない．本論文の結果は，要件が曖昧で高い生産性が求められるプロジェクトに適した Node-RED という 1 ツールの導入に関しての調査結果から得られたものであるが，このような視点は他のツールにも重要である．

近年，アジャイル開発の再定義の一つとして注目されているモダンアジャイル[8]は次の 4 つの基本理念によって定義されている．

- 人々を尊重する
- 安全な状態を前提とする
- 素早い実験と学習
- 価値を継続的に届ける

今回調査したプロジェクトは一定期間のタイムボックス管理を行う一般的なアジャイル開発ではない．しかし，アンケートの結果から得られた知見は，モダンアジャイルの基本理念と通じるものであり，Node-RED がソフトウェア開発のアジリティ（機敏さ）を高めるものであると考えられる．

今後も Node-RED を利用するプロジェクトを増やして，そのノウハウを蓄積するとともに，より良いプロセスに関する知見も増やしていきたい．

謝辞

アンケートのご協力いただいた西根さん，陣内さん，奥嶋さん，澤本さん，畑中さん，中島さん，佐々木さんに感謝いたします．

参考文献

[1] Beck 他，アジャイルソフトウェア開発宣言，http://agilemanifesto.org/iso/ja/manifesto.html，2001.
[2] 松山，鯵坂,飯ヶ谷，情報システム開発におけるソフトウェア資産の上流シフトへの対応，ソフトウェア・シンポジウム 2016 in 米子, 2016.
[3] 渡辺，X-TEA Driver，ディービーコンセプト，http://dbc.in.coocan.jp/xeadDriver.html，2004..
[4] JS Foundation，Node-RED is a visual wiring tool for the Internet of Things，https://nodered.org/.
[5] Node-RED User Group Japan https://nodered.jp/,

[6] 桑野, ブラウザでお絵描き I/O!Node-RED で極楽コンピュータ・プログラミング (インターフェース SPECIAL), CQ 出版, 2016.
[7] Glass, ソフトウェア開発 55 の真実と 10 のウソ, 日経 BP 社, 2004.
[8] Smith, Agile 2016 Keynote: Modern Agile, https://www.infoq.com/news/2016/08/agile2016-modern-agile, 笠原, Agile 2016 の基調講演: モダンアジャイル , https://www.infoq.com/jp/news/2016/08/agile2016-modern-agile, 2016.

テストの遊び場を作ったら、一緒に遊びますか？

浅井 真樹子
株式会社 ワークスアプリケーションズ
asai_m@worksap.co.jp

要旨

テストの遊び場は，ソフトウェアテストに関する色々な技術のスキルアップを目的に，泥まみれになって遊べる体験型の場所があったら楽しそうだというアイデアである．

ソフトウェアテストの発展への貢献と世界中のエンジニアと交流できる可能性を秘めている，私にとっての野望でもある．

1. はじめに

テストはとてもクリエイティブでプロフェッショナルな楽しい仕事なので，もっとスキルアップしたい！しかしお勉強スタイルで学習するのはツラいし限界がある…．と考えていたとき，ふと思いついたのが泥んこになれるテストの遊び場である．

テストの遊び場は，リスクフリーで，自分で考えたことや色々なメソッドを実験でき，みんなで交流することでフィードバックがもらえる場所．

ソフトウェアテストの活動は，要求分析，計画，設計，実施，バグ報告，管理や分析，自動テストなど，多岐にわたるが，テストの遊び場ではテストに関するあらゆるものを自由に扱いたいので，敢えて「テスト」とだけ称している．

扱う対象は幅広く自由に選択するが，現物を入手しやすい対象を 1 つ選択して複数人で扱うことを想定している．これは 1 つの対象を共通に扱うことで，比較したり客観的に評価したり具体的にアドバイスしあうことが可能になり，オンライン・オフラインの垣根なく議論でき，空中戦になりにくいと考えたからだ．

参加者は初心者から有識者まで，色々な立場でソフトウェアとその品質に携わる有志に集ってほしい．

自分の腕を試し，色々なエンジニアと交流して腕を磨き，開発者や製品に貢献できるようになる．そういう活動ができる場所を作ってみたら楽しく学べ，ソフトウェアテストとソフトウェア製品の役に立つのではないかというのが今回提案するアイデアだ．

具体的な活動イメージの一つを紹介する．

2. こんな悩みはないだろうか？

テスト設計や自動テストを始めたいが，どのように始めたら良いか分からない．

テスト設計について情報はたくさんあるが，どの手法が自分に適しているのか分からない．

網羅的にテストして保証したいが，どこまでやればいいのか分からない．

テストの目的に対して，実施しているテストが妥当なのか，十分なのか心配だ．

品質指標の妥当性が分からない．

新たな手法に取り組んでみたいが，失敗したら困るのでチャレンジできない．

他社の事例を参考にしたが上手く扱えなかった．

自社の状況を公開できないので適切なアドバイスをもらえない．

などといったことに悩んでいないだろうか？今現在はなくても，過去に悩んだことはないだろうか？

悩みや疑問を解消したくても，時間がない，知識がない，相談できる有識者がいない，予算がない，試しにやってみて失敗できる状況ではない，など色々な障壁があり，なかなか解決の糸口を見い出せないということはないだろうか？

こうした悩みに対して，失敗してもリスクがなく自由に実験でき，体験を通して自らに適した解決の方法を見つけ出せるのがこの遊び場だ．

3. テストの遊び場での遊び方

テストの遊び場でテストに関わるあらゆるものを自由に扱うことは前述の通りだが，具体的にイメージしやすいように，テスト設計を対象に例を示す．

まずはテストの対象を決める．これは既に公開されていて，誰もが入手して使えるものにする．例えば「ブラウザのテスト」をすると決めた場合，どのブラウザを対象にするのかを検討して選択する．今回は Google Chrome を対象にすると決めたとする．

次にテストの手法やテストする範囲を決める．既に出荷されている製品なので，外部仕様に対するブラックボックステストをやることにしてみる．

さらに Google Chrome のどの機能を対象にするのか検討する．Google Chrome には設定や画面の表示や検索などの他に Google Chrome 上で動作する色々なサービスやアドオンが提供されているので，各機能をどのような単位で集めるかまたは分解するかを検討する．例えば検討の結果を元に機能一覧を作ってみるだけでも対象製品の特性に対する理解が進み，各自のこだわりや思考パターンにも気づけることだろう．

テストするのに扱い易い単位が作れたら，どういうメソッドを使ってテストを設計してみるかを検討する．自分が普段使い慣れている手法を披露したり，新たな手法を取り入れてみたり，試してみたい手法について参加者同士でアドバイスしたりすることもできるだろう．

こうしてテストを設計する過程で色々な学びを得られるのはもちろんだが，実際にバグを見つけたら当該製品の開発者にフィードバックすることもできるし，バグをナレッジとして設計にフィードバックすることもできる．

テスト設計に入る前に目的を明確化したり，計画を作成したり，メトリクスを定義しても良い．テスト設計の成果物を元に自動テストを書いてみることもできるし，テストケース作成を練習してみても良いし．また作成したテストケースを用いてテストを実行してみるのも良い．

どういう遊び方をするにしても，手元で扱える対象をみんなで共通に扱うので，個別具体的に議論できるので成果を得やすいのがポイントだ．

4. さいごに

遊び場での体験を通して，自分の悩みが解消し，自分の強みを確認でき，興味の対象や思考パターンなどが分かる．色々なメソッドの使い方や優れている点を，実際に体験しながら，ときに失敗しながら学べる．悩みを率直に語り合うことができ切磋琢磨できる．

活動を通して生まれた成果物は共有でき，ナレッジとして蓄積されることで色々な立場でテストに携わる人々のテスト活動に貢献できるし，開発にフィードバックすることもできる．どんな学びをどれほどたくさん持ち帰るかは参加者の自由だ．

会社や対象ドメインの垣根を超えて，より具体的な体験を通して学びを得る．

そういう遊び場があったら，一緒に遊びたくなるだろうか？どういうテストだったら一緒に遊びたくなるだろうか？

遊び心をもって真剣に取り組む．そこからクリエイティブでプロフェッショナルな楽しいテストに出会えるのではないだろうか．

参考文献

なし

エンジニア人材を死滅させるマイクロマネジメントの打破 ～エンジニアを活かし育てるトリセツ活動の提案～

松尾谷徹
デバッグ工学デバッグ工学研究所
matsuodani@debugeng.com

要旨

企業の *IT* 関連職場において，エンジニアの悲痛な叫びを聞くことが増えている．叫びの多くは，日々の仕事として，その活動の意味が見いだせない無駄な作業に忙殺されていることから生じている．現代の若者に我慢が不足していると切り捨てる管理職も存在するが，決してそうではない．

エンジニアは，製品やシステムやサービスにおいて，維持するだけでなく付加価値を生み出すことが求められている．その原動力はクリエイティブな人的資源による創造的な工夫や変革であり，規範的なプロセスの遵守だけでないことは明白である．

しかし，多くの職場レベルのマネジメントは規範的な側面が詳細に強化され，厖大な手続き的な作業を押し付けるマイクロマネジメントが蔓延している．結果的に，付加価値を生まない活動にクリエイティブ人材を投入することになり，付加価値が生まれないばかりか，人材を疲弊させ，組織自身をも衰退させている．この現状を明らかにし，この流れを断ち切るための提案を行う．

1. はじめに

この主張は，エンジニアの地位獲得を進めようとするものであるが，経済的な地位など利己的なものでは無い．現代のエンジニアに求められているパフォーマンスを発揮し，社会や組織に貢献するために，雇用主や社会に求めるものである．エンジニアが付加価値生産性を高めるクリエイティブな成果を出すには，自由な発想と試行錯誤が必須であり，それをサポートするエンジニアの処遇と扱い方＝「トリセツ」(取扱説明書) を明らかにする必要がある．

ソフトウェア以外の伝統的な機械や電気の分野においては，エンジニアを貴重な人的資源とし，雇用側はその活躍の場と処遇を一般的な雇用者とは区別して提供し，育成している．何故ならば，彼等の貢献によって製品やサービスの価値を高め，企業や組織を維持発展させているからである．

日本におけるソフトウェア産業は，歴史的な経緯からエンジニアも含め人的資源を消費原料的な作業工数として取り扱ってきた．管理（監督）の現状は，非熟練者によるマニュアルに沿った作業の監督と位置付け，量的生産性，品質の底上げ，納期遵守といった低レベルの管理に留まっている．

では，どのようにしてこの慣習的な問題を解決して行くのか．ここは次の項目に分けて考える．

1. 現状の認識と対策としての見える化
2. マイクロマネジメントの原因と対策
3. あるべき姿に対するコンセンサス
4. 現実的なトリセツのテーラリング

2　現状の認識と見える化

大手広告企業における悲しい事件をきっかけに，企業の職場における管理の問題が明らかになりつつある．顕在化した問題とは労務管理の最低限のレベルである労働時間ですら長年に渡って違法状態であり，それを自律的に改善出来なかったことである．事件は，超過した労働時間が原因ではなく，クリエイティブな人材に対する管

理規範＝トリセツが未成熟で，封建時代の滅私奉公のような主従関係の考え方に近い人的資源管理に本質的な原因がある．

識者が指摘するように，本質的な問題は２つに分類できる．１つは組織の末端における職場が閉鎖的であり，未熟な管理が行われてもそれを検知し回避することができないことにある．逆に隠ぺいし悪い方向に強化されて事例もある．

もう１つは，末端の職場で異常な管理を行う者の管理規範にある．多くの場合は，管理に関して未熟な者が独善的な考えで管理行為を行うことにあるが，その職場や組織において慣習化している場合もある．

この項では，職場の閉鎖性について考える．問題が顕在化するのは，事件となった場合や，パワハラとして強い被害を受けた者が訴えた場合など，かなり悪い状況である．この現象は，大学ではアカハラであり，家庭ならDV である．現実として，これらのパワハラ類は顕在化に至らないが潜在的な状況を含めると，閉鎖的な主従関係において多発しており，根底には上司部下の関係を封建的な身分関係と誤解している国民的な未熟から生じており，簡単に解決できない問題である．

よって，国民性を変える前に，職場における過剰な閉鎖性を減らし，問題の兆候を検出する仕組みが必要である．いじめ事件などの事例では，組織側は隠ぺいの意図が強く，組織側の調査結果はでは「いじめは無かった」となり，後から第三者による調査（アンケートや聞き取り）で深刻な状況と兆候があったことが明らかになることが多い．

見える化の手段としては，簡便な調査紙によるアンケート調査から，心理測定尺度の研究で明らかになった構成概念を用いてその構成因子ごとの得点と，ビッグデータを用いた判別分析（AI 技術）を行う方法が考えられる [1, 2]．

IT 分野においては，JISA の援助を基に 2001 年に実施したエンジニアのパートナー満足調査（PS 調査）がある [3]．この調査は IT 系のエンジニア約 1500 名に対して主に「仕事満足」の観点から分析した [4]．閉鎖性の打破のためには，これに加えて「不満足」の観点を追加する必要がある．

もう一つのひな形は，職場のリーダシップ研究で良く知られている「PM 理論」から，エンジニアが強い「不満足」を生む因子を探る方法である [1, 5].「苦手な上司／良かった上司」の比較研究も行われており，この方法の実現性は高い．

職場の状況を客観的に見える化するための計測手段として，質問紙，分析ツールなどを作成するのはさほど困難ではなく，実績もある．実際の運用方法については検討が必要で，匿名で第三者が行い，人事評価など意図しない利用による弊害を防ぐ必要がある．

3　マイクロマネジメントの原因と対策

なぜマイクロマネジメントのような不適切な管理が自然発生的に多発するのかについて考える．マネジメントの中に人的資源のパフォーマンスを高め育成する機能が，重要な要素として含まれていることを否定する者はいない．しかし，末端の職場において大義名分的な育成機能より，日常の習慣的な管理機能が優先され結果として逆に人的資源のパフォーマンスを抑圧するケースが生じている．習慣的な管理機能とは，進捗報告や予実管理，コンプライアンスや QCD 関連のレビューなどであるが，チャレンジ的ではな実態は実効性の低い文書作成と打ち合わせに忙殺される．

問題を起こす管理・監督者の多くは，人的資源管理の職務に対して未経験でかつ専門的な知識も訓練も受けていない場合が多い．具体的には，エンジニアが職位が上がり，未経験の管理・監督の職務に就く場合である．常識的に考えると，高学歴で社会人となりそれなりの実務を経た者が，マイクロマネジメントのような稚拙な行動に走るとは考え難い．何らかの原因が存在するのではないか．

原因として考えられるのは，管理・監督の職務を身分的な地位としてとらえ，君主的で指示命令すること＝管理者と考える時代錯誤的な勘違いである．経営学における人的資源管理の基礎知識からしても，現代ではあり得ない管理規範ではあるが，パワハラやアカハラの原因因子としては強い．

では，何故このような稚拙な思い込みに走るのか．その原因は，職場の閉鎖性にあるのではないか．配属後，その職場一筋ですごし，社外の学会や業界どころか，社内の他部署とも社会的なつながりを持たないため，常識的な社会経験が無く，未成熟で独善的な行動に陥ると考えられる．

次項のあるべき姿とは分けて，人的資源管理の前提としての「トリセツ」が必要である．べからず集のような事例と対策的なものや，人を扱う管理への入門書的なま

とめ方が考えられる．

4　あるべき姿を求めて

マイクロマネジメントのようなとんでもない状況とは別に，目指すべきトリセツについて考える．理想ではなく，現実的な姿としては，先行する成功例から学ぶ必要がある．理想とは演繹的な規範から導かれるもので，教科書的な美しい姿である．現実的な姿とは，事例などで実証されたものを意味する．具体的には，客観的な計測や，実際のチームやプロジェクトの事例を基にした取り組みである [6, 7]

ソフトウェアエンジニアのドメスティックな特徴もあるが，大きな括りでこの問題を捉えると，クリエイティブな活動と生産活動との対比問題と見ることができる．クリエイティブな活動が 21 世紀の産業を牽引していることは，多くの賢者が述べており，産業界においても Google，Amazon など 21 世紀に躍進した多くの企業で実証されている．この分野における先進的な考え方は，リチャード・フロリダによるクリエイティブ・クラスの中の Super-Creative Core が近いと考えている [8].

「トリセツ」が目指す方向は，組織がクリエイティブな成果を出すための人的資源管理をステークホルダに解りやすく示すことにある．ステークホルダとは職場を含め経営層や関係する顧客などを指す．具体的な「トリセツ」は，時代や状況によって変化するので，変化に対応して行く必要がある．あるべき姿は，抽象的であるが基本となる考え方に他ならない．

経済成長の実証研究から，成長に必要な要素が明らかになっており，それらのほぼすべては人的資源に関するものであり「才能」「寛容性」「経験への開放性」などが指摘されている [8].「才能」とは，スキルと野心であり，具体的には高学歴でかつチャレンジ精神を持つ人的資源である．多くの企業の採用基準であり，これは達成されている．

「寛容性」とは，多様性を認めることであり，エンジニア側からすると自由度である．陳腐化の早い分野において技術を維持して行くには，論文を読み，学会などの情報源に参加し，考え試して技術習得を自律的に行うための自由が必要不可欠である．管理・監督側は，その自由度を認め確保する寛容性が必要である．マイクロマネジメントは，これに逆行している．

「経験への開放性」とは，新しい技術や経験に対する好奇心であり，これが欠如している職場では，チャレンジ精神が疲弊しクリエイティブ活動が芽生えない．この「寛容性」「開放性」を柱にして，日本的な価値観である「育成」「チーム」などを加えた概念を想定している [9]

5　現実的なトリセツ

トリセツについて，3 項ではトリセツ以前の課題について考え，4 項ではあるべき姿について考えた．ここでは，実際にトリセツを展開する局面について考える．展開する局面とは，展開対象の職場や組織の状況を考慮し，さらに既存の目指す考えも取り入れテーラリングを行う局面である．

テーラリングの概要を図 1 に示す．テーラリングのための情報は，トリセツを展開する職場や企業に適合させるためのもので，２つの要素が考えられる．1 つは，貢献目標と示したもので，企業や社会で既に認められたエンジニアの倫理やコンピテンシー（行動特性）である．雇用側からすれば，トリセツの前提となるエンジニアの義務や組織への貢献に関する期待（要求）である．

具体的な内容としては，企業であれば採用要件として求めるエンジニアの資質であり，その内容は「チャレンジ精神」「やる気」などと表明されている．これらを汎用化すると，エンジニア倫理であり，現代において高い支持をうけているのは IEEE の ”Software Engineering Code of Ethics ” である [10, 11].

現実的なトリセツへの反映は，エンジニアの義務の確認である．これらの倫理が達成されているからこそ，エンジニアにクリエイティブな成果を期待し，良い処遇

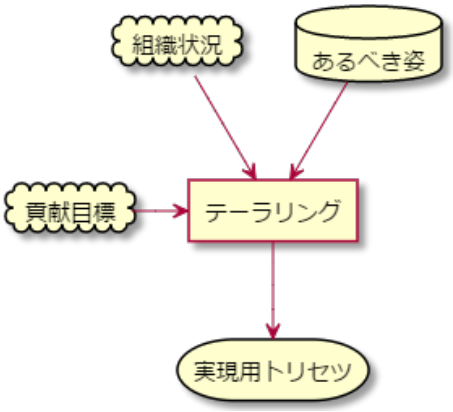


図 1: トリセツのテーラリング

（トリセツ）を提供する．

もう 1 つは，対象とする職場の成熟状態である．マイクロマネジメントなど未成熟な状態と，クリエイティブな活動が進行している状態では掲げるトリセツの内容は変える必要がある．具体的な内容は，テーラリングにより職場の状況に合わせて解りやすく現実的な表現にする必要がある．

トリセツの内容自身は職場によって多少変化するが，エンジニアが主体的に働く原動力は，外発的報酬ではなく，内発的報酬であることを理解する必要がある．IT 系エンジニアの内初的報酬は，満足度調査の結果から次の通りである [4]．

- 仕事から得られる「やりがい」
- 自律的な工夫や進め方の自由「柔軟性」
- キャリアパスや自己成長「安定性」
- 同僚からの敬意や良いチーム「仲間」

6 おわりに

日本のソフトウェア産業は，1980 年代から高学歴のエンジニアを長年に渡って雇用し活動してきた．しかし，世界的な IT 技術やサービスの進歩と比べると，相対的には衰退していると認めざるを得ない状況にある．その一つの原因は，極端な例であるがここで取り上げたマイクロマネジメントのような，エンジニアに対する人的資源管理の稚拙さにある．

その原因は，職場の閉鎖性や封建的な主従関係と真面目な国民性から生じており，改善されるどころか維持され強化すらされている．海外では，エンジニア倫理とそれに答える雇用側の処遇が，社会習慣として定着している．エンジニア倫理については，我が国でも多くの識者が論じているが，雇用側の処遇＝人的資源管理についてはあまり議論されていない．

ここでの提案は，エンジニア倫理を持つエンジニアに対する，企業や社会のエンジニアに対する処遇改善である．どっちが先かと言う鶏と卵の議論もあるが，後者が不足していると考えている．ここでの提案は，閉鎖性を打ち破るための見える化，エンジニアを指示命令で使うリーダーシップは長期的な成果を生まない，良い人的資源管理のプラクティスを実証主義で積み上げる，実践には職場の現状に応じてカスタマイズが必要，などである．

参考文献

[1] 堀洋道，吉田富二雄，“心理測定尺度集 ii,” 2001.

[2] 松尾谷徹，“It に現場力は存在するのか: その計測と評価の試み,” ソフトウェア・シンポジウム，pp.1–8，2014.

[3] 松尾谷徹，“パートナー満足 (ps) と人的リソースのパフォーマンス,” プロジェクトマネジメント学会誌，vol.4，no.1，pp.3–8，2002.

[4] 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.

[5] 三隅二不二，“組織におけるリーダーシップの研究 (リーダーシップ–集団過程の社会心理学 (特集))–(現実の集団におけるリーダーシップの研究),” 年報社会心理学，vol.14，no.11，pp.63–90，1970.

[6] “感動するチーム,” http://itpro.nikkeibp.co.jp/article/COLUMN/20090525/330593/.

[7] 増田礼子，“混成チームにおけるチーム力向上のための三者ヒアリング活用事例,” https://www.juse.jp/sqip/symposium/archive/2015/day2/files/ronbun_B3-4.pdf.

[8] 井口訳リチャード・フロリダ，“新クリエイティブ資本論,” 2014.12.

[9] 森本千佳子，“プロジェクト成功のためのコミュニケーション・ファシリテータ活用,” プロジェクトマネジメント学会誌，vol.14，no.4，pp.23–28，2012.

[10] “Software engineering code of ethics,” https://www.computer.org/web/education/code-of-ethics.

[11] “ソフトウェアエンジニアの倫理（翻訳）,” http://ethics.acm.org/wp-content/uploads/2016/07/SE-code-jpn.pdf?60549e.

TPI NEXT による現場主導のテストプロセス改善を支援するための手法の提案

高野 愛美　河野 哲也
株式会社日立製作所
{manami.takano.pj, tetsuya.kouno.cb}@hitachi.com

山﨑 崇
株式会社ベリサーブ
takashi.yamasaki@veriserve.co.jp

佐藤 徳尚
日本ナレッジ株式会社
norihisa-sato@know-net.co.jp

要旨

本報告では，TPI NEXT[1]による現場主導のテストプロセス改善を進める際に現場のチームが陥りやすい問題に対して支援するための手法を提案する．TPI NEXT による現場主導のテストプロセス改善で陥りやすい 2 つの問題，1）チーム内で現状のテストプロセスを共通理解していないため，改善策の検討が難しい，2）改善項目が多く，優先順位の判断が難しい，に対して支援するための 2 つの手法，1）現状のテストプロセスを可視化するための PFD[2]の活用，2）一定の指針による改善項目の絞り込みを提案する．また，現状のテストプロセスを可視化する対象を判定するために，TPI NEXT のチェックポイントがプロセス表現できるかどうかを整理した結果を報告する．

1. はじめに

近年，テストプロセス改善技術への関心が高まっており，導入する企業が増加している[3]．我々ソフトウェアテスト技術振興協会 テストプロセス改善研究会では，テストプロセス改善モデルである TPI NEXT を取り上げ，現場主導のテストプロセス改善を支援するための研究を行っている．

本報告では，TPI NEXT による現場主導のテストプロセス改善を進める際に陥りやすい問題を支援する手法を提案する．

2. TPI NEXT による現場主導のテストプロセス改善で陥りやすい問題

TPI NEXT は，現場主導で取り組みやすいテストプロセス改善モデルであるが，コンサルタントなどの支援を受けずに現場のチームだけで，テストプロセス改善に取り組む場合，スムーズに改善を進めることは難しい．TPI NEXT によるテストプロセス改善を進める際に陥りやすい 2 つ問題を示す．

1 つ目は，現状のテストプロセスを共通理解していないため，改善策の検討が難しい，という問題である．TPI NEXT は，改善策の検討において，現状のテストプロセスの可視化には言及していないため，TPI NEXT の枠組みのみでは，現状の共通理解ができずに，改善策の検討が難しい．

2 つ目は，改善項目が多く，優先順位の判断が難しい，という問題である．TPI NEXT が改善の指針として提供するクラスタセットにより大まかな改善スコープを特定できるが，詳細な改善の優先度は現場チームの状況により判断が必要である．そのため，同じスコープ内の改善項目が多い場合，優先順位の判断が難しい．

3. 問題解決のための手法の提案

2 章で述べた 2 つの問題に対して，2 つの手法を提案する．

1 つ目は，現状のテストプロセスを可視化するための PFD の活用である．PFD により現状テストプロセスを可視化することで，チームが現状のテストプロセスに対して共通理解し，改善策の検討を行えるようになる．また，可視化する対象は，チェックポイントを基準にプロセス表現しやすいかどうかで判断する．このため，各チェックポイントがプロセス表現できるかを整理し，その結果を PFD 化の指針として提供する．

2 つ目は，一定の指針による改善項目の絞り込みである．各チームメンバが TPI NEXT によるアセスメントを行い，その結果のばらつきを指針として改善項目を絞り込むことで，優先順位の判断ができるようになる．チーム間で評価結果がばらついたチェックポイントは，プロセスが暗黙的であると考えられ，これらは可視化することで作業や成果物の抜け漏れの防止などの改善が見込めるため，優先的な改善対象とする．

参考文献

[1] Vries,G. D. et al., TPI Next Ⓡ :Business Driven TestProcess Improvement, UTN Publishers. (薮田ほか(翻訳), TPI NEXTⓇビジネス主導のテストプロセス改善，株式会社トリフォリオ)

[2] 清水吉男，PFD(Process Flow Diagram)の書き方，http://soft-koha-hp.la.coocan.jp/process/PFDform3.pdf

[3] 河野ほか，パネルディスカッション：普及が始まったテストプロセス改善技術導入・実践着手のために考えるべきこと，ソフトウェアテストシンポジウム 2017 東京，2017

[Future Presentation] 道徳性向上後の行動におけるアラートシステム

阿部 敬一郎[*1*2]
[*1]東筑紫短期大学

中谷 多哉子[*2]
[*2]放送大学大学院

要旨

道徳教育支援システムの一部である，高次な道徳的行為の発現を促すためのタスク管理システムを提案する．既存のクラウドサービスを使用し，教師と学生，学生と学生とで学内外での実践とその達成率を可視化する．

1. はじめに

「情報倫理」の研究は我々の生活で一般的なことである．しかしながら文書化されたルールや倫理では，世界のあらゆるシチュエーションをフォローしきれない．規則にないことは何をしても許されるという考えは危険である．ルールや倫理のリストを作ることがなくても，物事の善し悪しが議論できるためのサポートに，成人向け道徳教育が必要であるとする．

しかし成人に道徳の教育をすることは容易ではなく，成人向けの道徳教育の成果を研究する例も殆ど無い．本研究は卒業までという時間の中、社会に出る前の大学生の道徳教育方法を作る挑戦をする．大学生への道徳教育の定義を，『①学生自身が高い道徳性の考え方を理解し，②高い道徳性で議論し，③高い道徳性を実社会で発動し，④それを認識するまでが含まれる』とした．論文[1]の中で①～②は成功することができたが，③の道徳性の高い行動の発動と④の認識に課題が残った．

ジャック J．フィリップス[2]が教育・研修の成果をまとめるシート類を参考にし，「アクションプランワークシート」を作成した．高次の道徳性を意識して行動の計画を立て，振り返りによって高次の道徳性の発動を認識できる．目標を立てるのは上記①②の直後の教室なので，教師や学生同士のアドバイスにより高次の道徳的意識はある．つまりタスクも計画された時点で，高次な道徳性によるものと言える．

タスク管理は次の 2 点で道徳教育に効果的である．1 つは道徳的行動には発動の制限が厳しくないため，忘れがちである点である．公の場において発動目標を立てることで，意図的に行動を取れる状況を設定できる．2 つは高次な道徳的意識を促せることである．道徳的行動は意識の持ち方で高低が変わる．タスクを先生や学生同士で共用することで，決めた行動を取るだけでなく，行動意識に高い道徳性を気づかせる環境に置ける．

具体的な道徳的行為であるが，現段階において判断できる行為は限られている．執筆者の勤務する大学における「建学の精神」の一部，「勇気の心，親和の心，愛の心」は実験対象の学生に行動が促しやすい．具体的な行為は挨拶・礼と掃除が例に上がる．挨拶・礼は場の雰囲気を正し，感謝の気持ちや相手を思いやる気持ち，敬愛の心を持って行うと高次な道徳性になる．挨拶の言葉やおじぎをするという行為ではわからない．行為を起こす前に高次の道徳性を気づかせ，発現したらすぐ記録することが望ましい．その時の気持ちがわかるのは本人だけだからだ．

頭で理解していても高い道徳性を発現させ行動に移れるかは別問題である．この度のアラートシステムは道徳的目標の内容と意識の気づき，計画時にアラート(アラームを鳴らす・バイブレーションで知らせる)を出すこと，その行動の達成度を教師と学生，学生同士とで共有する手法について提案する．

2. クラウドシステムの準備と管理手法

2.1. “Wunderlist”の紹介

Wunderlist はクラウド型タスク管理 Web アプリケーションである．機能概要は以下の通り．

(1) 5 人までタスクを共有することができ，共有者はお互いにタスクを振ることができる．

(2) タスクは開始時間とアラートとして注意を促す時間，タスク完了見込み時間を設定でき，タスクを始めてから終わるまでの時間をカウントすることができる．毎日，毎週のように繰り返すタスクも設定が可能．

(3) 目標の設定，アラート，タスクの完了ボタンを押したとき，プッシュメールでグループメンバに自動で告知がされる．

(4) スマホアプリケーションでも Web サイトでも

タスク管理が行えるサービスである.

以上の機能から，道徳的行動の管理方法を紹介する.

2.2. 道徳的行動の管理方法と評価

高い道徳性を得た学生が，高次の道徳性を得た同級生や先生のアドバイスを受けながら行動目標を自ら設定しなければならない. 道徳性を評価することを考えると，人が集う場所に赴く時間が行動を発現しやすい. 例えば通学中や休み時間，授業後の時間や掃除当番のような他人と接点を持てる時間や公共的義務の執行中が挙げられる. 高い道徳性を発現できる時間と場所を特定したら，タスクを自ら課し，教師とある時は学生にも目標を共有する. 学生同士で目標を共有することは，高次の道徳性を発現する補助として期待できる. また学内の当番のように，学生が一緒に活動するときは，お互いが目標を立ててもよいし，共通の 1 タスクを代表者が管理してもよい.

目標とした行為が一定の時間を要する場合は，開始時と終了時にスマホの管理画面のボタンを押す. 操作が複雑に感じる場合は終了したときのみでもよい. そのときの行為または意識が目標に届かなかった，あるいは予想以上だった場合は，理由をコメントとして残す. 入力が面倒なら音声で残してもよい. タスクが完了すると，情報の共有者にメールが自動で届く. 計画して初日に当たるときは学生や先生から「いいね」などのコメントで褒めたり，励ましたりして計画の達成をサポートできる. 報告がないときはリマインダーを送り，行動を促してもよい. すべての報告に先生がすべて答えることは難しいので，学生同士の双方向教育を効果的に活用することが望ましい. ただしコメントを出し合える，学生同士の仲の良い雰囲気を構築する課題は残る.

計画した行動をリアルタイムに把握し，タイミングよく人による評価や励ましをすることで，高次の道徳的な行為にやりがいをみつけやすくなる. 繰り返すことで行為と考え方が習慣化し，大学生への道徳教育の定義の達成に近づく. 目標とする期間の終了後にタスクの成否を集計し，行為を発動するときの気持ちは教師が L.コールバーグの道徳性の発達段階[4]に照らして 6 段階で評価する.

3. ディスカッションと考察

教材，授業を可視化することで道徳教育レベルを合わせること，行動の達成度合いを可視化することで，母集団の道徳性の発現を比較することを目指した. 高次の道徳性を発現できない母集団が生まれたとしたら，児童向けの道徳教育とは違い，対象者が大学生や成人ゆえの問題が見えるはずだ. その問題を埋める教育や支援システムを構築，実験することでより完成度の高い，成人・大学生向けの道徳教育を目指す.

参考文献

[1] 阿部敬一郎，辰己丈夫，村上祐子，中谷多哉子，“道徳教育支援システム化に向けたモラルジレンマ”，信学技報，vol. 116, no. 493, KBSE2016-44, pp. 31-36, 2017 年 3 月.

[2] J. J. Phillips, How to Measure Training Results: A Practical Guide to Tracking the Six Key Indicators, New York: McGraw-Hill, 2000.

[3] Wunderlist, https://www.wunderlist.com/ja/download/(2017 年 3 月 16 日アクセス)

[4] Anne Colby, F., Lawrence Kohlberg, S.: The Measurement of Moral Judgment. Volume I. Cambridge University Press, New York (1987).

ソフトウェア保守 隠蔽の構造　～なぜ，保守作業は隠されてきたのか？～

増井　和也　ソフトウェア・メインテナンス研究会

masui@ye4.fiberbit.net

１．　はじめに

本報告は，ソフトウェア保守作業が現場の実作業量に比べ，大幅に少なくなるよう隠蔽される原因，構造，弊害，対策提言について一般事例を基に総括的に報告するものである。なお，本報告で使う「ソフトウェア保守」（または単に「保守」）は，JIS X0160[2]の保守プロセス規格 JIS X0161[3]が示す定義とする。

２．　現場の保守作業量と周辺対応の乖離

ソフトウェア技術者が参画するプロジェクトで新規開発 36%，保守(メインテナンス)に類するもの過半数との報告がある [1]。40年超ソフトウェア現業経験を持つ報告者も「保守作業が新規開発作業よりはるかに多い」に違和感はない。

国立国会図書館登録和書書籍名で「開発」を含むソフトウェア関連図書の最近登録数は 1,500 件を超える [4]。しかし，現場の作業量とは相関せず，「保守」を含む同登録数は 40 と僅かである。

また，IT 教育機関の教育内容はソフトウェアの新規制作想定が大半に見える。現場で多数発生するはずの作成済みの設計やプログラムの見直し技法，修正デグレード防止や回帰テスト効果的・効率的実施方法，ソフトウェア稼働後の構成管理要点等，保守技術を向上させる教育実践は聞かない。保守対応もあるだろう IT 企業の新人集合教育で，完成済みソフトウェアに対し，事前予告なしの保守課題を次々と与え，適切なソフトウェア保守の教育を積極実施している企業は，少なくとも広報情報等からは見えない。

情報処理推進機構が取りまとめた IT 担当者として持つべきスキル・知識項目 [5][6]の中に，JIS X0161 を詳細に意識した「保守」は少ない。最近の改定で少し増えたようだが，情報処理技術者試験のシラバスにもソフトウェア保守は概要・付け足し程度である。

IT 業界誌の法人対応事例記事では，システムを新規に再構築・リプレースし，ソフトウェアの保守・運用コストを劇的に削減できたとの記事は多い。しかし，実態として増え続けるソフトウェア保守作業を，リプレースが困難な状況で，保守担当が技術面・プロセス面で保守の課題をどう克服してきたかの記事は少ない。

３．　現場保守作業を隠蔽させてきたレガシーパラダイム

報告者の経験や研究から，乖離の最大の原因はソフトウェアを扱う企業・研究・教育・行政の全体が「開発善／保守悪」または「保守は開発の付け足し」のレガシーパラダイムに長年固執してきたためと考える。その結果，保守作業自体の存在や量の多さを不当に悪者扱いし，実態より極端に少なく見せる保守隠蔽を助長してきた。現場の部門や技術者自身も保守作業は必要悪な付随作業と考え，誇らしげに公表することが憚られてきたと分析できる。

４．　ソフトウェア保守隠蔽の構造例

ソフトウェア保守作業はどのようにして隠蔽されてきたのか，その構造的な要因 3 例を示す。

(1)例 1)：開発の範囲を最大限広げ，その分保守の範囲を縮小

国際規格 [3]が示す「保守」の範囲は修正規模に関係なく，改良(Enhancement)，訂正(Correction)ともに保守で，同じ保守プロセスをガイドする。しかし，ある公的な調査 [7]では，保守は「改良開発」という言葉を使い，改良はあくまで開発の一種とし，保守作業を隠すか小さく見せている。また，一括に「運用保守フェーズ」等，あえて「保守」の存在を薄める表現も頻繁に見かける。

(2) 例 2)：次期開発に保守を紛れ込ませる

企業等において，新規開発が低品質のまま稼働され，投資計画上の保守費より稼働後保守費が大幅に増大しそうな場合がある。計画差（失敗）を隠すため，当該保守作業を次期開発体制の中で行い，前開発の保守費がゼロか，想定以下のように見せる。

(3)事例 3)：アウトソーシングによる隠蔽

稼働後のソフトウェア保守作業を外部委託（保守対応社員の転籍も行う）し，低価格の成果物請負契約とする。保守案件で複数更改したソフトウェアを「開発成果物」とし保守費を隠す。

５．　保守作業隠蔽の弊害

保守作業の隠蔽は現場疲弊を起こす。隠すためには，担当者に対し，保守作業を他作業とは別に対応させる必要がある。当該担当者はキャリアアップ，作業報告・記録といった，業務として当然与えられるべき時間を隠し保守に充てることになる。それでも不足する場合，さらなる残業・休日出勤での対応となる。IT 部門の構造的多忙要因の一つに保守の隠蔽があると報告者は考える。

6.　是正すべき点と研究者や現場担当者への期待

現場の疲弊をもたらす保守作業隠蔽の解消を進める対策として，開発と保守の明確な組織的分離が必要と報告者は考える。両者が分離され，相互牽制が機能することで，それぞれの作業量や品質が客観視できる。ソフトウェア関連の研究者，スタッフ，担当者，教育者が，保守そのものの研究・改善・教育から逃避せず，連携・情報共有し，積極的に保守についての実態把握を期待する。

7.　結論

ソフトウェア作成作業誕生以降，保守は隠蔽されてきた。その隠蔽の弊害がソフトウェア現場の技術者に対する過重労働増大の要因となっている。ソフトウェアの新規開発が減る時代に入り [8][9]，今後も増大し続ける保守作業の存在を明確に認知し，保守組織の開発組織からの分離，そのあるべき姿探求が必要な時が来ている。

参考資料・コンテンツ等

1) 「エンタプライズ系ソフトウェア技術者 個人の実態調査 2008」 情報処理推進機構 3.3 (3) (g)
2) JIS X160:2012 ソフトウェアライフサイクルプロセス (ISO/IEC/IEEE/EIA 12207:2008 の一致規格)
3) JIS X0161:2008 ソフトウェア技術-ソフトウェアライフサイクルプロセス-保守 (ISO/IEC/IEEE 14764:2006 の一致規格)
4) 国立国会図書館サーチ;’17 年 5 月時点，書名に「ソフトウェア開発」を含む書籍検索ヒットは 1,500 件
5) 「IT スキル標準 V3 2011」 情報処理推進機構 2012
6) 「i コンピテンシ ディクショナリ」情報処理推進機構 2015
7) 「ソフトウェア開発データ白書 2016-2017」情報処理推進機構 2016
8) ～ISO14764 による～　ソフトウェア保守開発　 増井和也，馬場辰男他　2007 ソフト・リサーチ・センター
9) 「ソフトウェアの少子高齢化が急加速 ～開発中心パラダイムに未来はあるか？～」増井和也 2016 ソフトウェア技術者協会　SS2016　Future Presentation

ソフトウェア・シンポジウム 2017

日程: 2017 年 6 月 7 日 (水曜日) ～ 9 日 (金曜日)
場所: 宮崎市民プラザ

SS2017プログラム

日付	時間	内容		
6/7 (水)	12:30-13:00	**受付** [ギャラリー付近] ※1		
	13:00-13:15	**オープニング** [ギャラリー1] 実行委員長 荒木啓二郎（九州大学） プログラム委員長 片山徹郎（宮崎大学）		
		[ギャラリー1] **< テストとデバック >** 司会：松尾谷徹（デバッグ工学研究所） 西康晴（電気通信大学）	[ギャラリー2] **< 保守、OSS >** 司会：日下部茂（長崎県立大学） 阪井誠（SRA）	[大会議室] **< 要求・安全・品質・メトリクス >** 司会：佐原伸（法政大学） 小田朋宏（SRA）
	13:20-13:45	研究論文 変更における状態を含むテスト網羅尺度とテストケース抽出法の提案 湯本剛（筑波大学）	事例報告 「あるある診断ツール」による課題の可視化と収集データの分析事例 室谷隆（TIS）	研究論文 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 久木田雄亮（和歌山大学）
	13:50-14:15	事例報告 探索的テストにおける不具合発見率向上に向けた取り組み 中野直樹（LIFULL）	経験論文 パッケージ製品におけるソフトウェア保守情報の活用事例 加藤英之（東芝ソリューション）	経験論文 Convolutional Neural Networkを用いたフォールト数予測手法 小川直記（日立製作所）
	14:20-14:45	研究論文 VDM++仕様を対象にしたテストケース自動生成ツール BWDMにおけるif式の構造認識に基づいたテストケース生成手法の提案 立山博基（宮崎大学）	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）	事例報告 AADLを用いたSTAMP/STPA支援 岡本圭史（仙台高等専門学校）
	14:45-14:55	**休憩**		
	14:55-15:20	経験論文 段階的なシステムテストによるIoTシステム開発効率化 西村治（パナソニック）	研究論文 OSS開発者の離脱要因理解のためのPolitenessの質的調査 宮崎智己（和歌山大学）	研究論文 ゴール指向分析KAOSにおける依存性を考慮した要求抽出法の考察 -酒屋倉庫問題の場合- 岡野道太郎（筑波大学）
	15:25-15:50	研究論文 例外処理を含むJavaプログラムを対象としたデータ遷移可視化ツールTFVISの適用範囲の拡大 佐藤拓弥（宮崎大学）	経験論文 OSS事前評価による開発リスク特定の取組み 岩崎孝司（富士通九州ネットワークテクノロジーズ）	研究論文 要件定義書からのファンクションポイント自動計測の試み 山田涼太（大阪大学）
	15:50-16:20	**コーヒータイム** [ギャラリー付近]		
	16:20-17:30	**キーノートスピーチ（1）** [ギャラリー1] **講演題目：「夢をかたちに」** 講演者：霧島酒造株式会社 企画室 副部長 福田 達之氏		
	18:30～	**情報交換会**（18時15分開場）※2 [宮崎観光ホテル 西館10Fスカイホール]		

※1：オープニング以降の受付場所は以下になります．
　　ワーキンググループ中：「ギャラリー（TS1）」/ それ以外：「ギャラリーもしくはギャラリー付近」
　　誰もいない場合は，070-6429-0240 にお電話ください （この電話番号は，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．お早めにおいでください．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，8:55に開場します.
※4：時間は各会場が利用できる時間帯を示しています．各 WG の開催会場については p.3 をご覧ください.
　　運営方法は WG ごとに異なります．詳細は各 WG のリーダまでお問い合わせください.

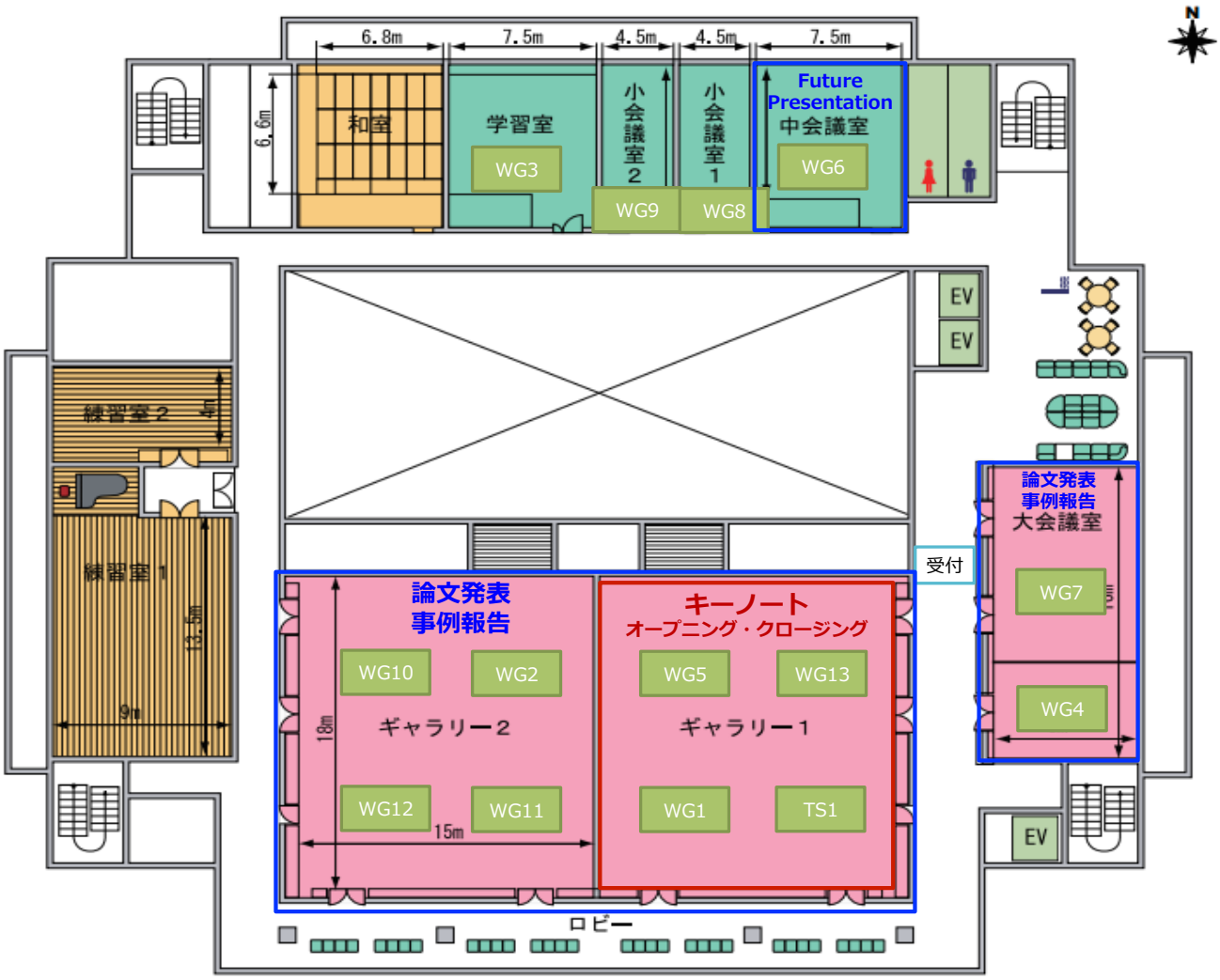
SS2017プログラム

日付	時間	内容			
		[ギャラリー1] **< コード解析 >** 司会：神谷年洋（島根大学） 中森博晃（パナソニック）	[ギャラリー2] **< マネジメントとチーム >** 司会：古畑慶次（デンソー技研センター） 田中康（奈良先端科学技術大学院大学）	[大会議室] **< 形式手法、プロセス改善 >** 司会：岩崎孝司（富士通九州ネットワークテクノロジーズ） 鈴木正人（北陸先端科学技術大学院大学）	[中会議室] **<Future Presentation>** 司会：西康晴（電気通信大学） 大平雅雄（和歌山大学）
6/8 (木)	9:00- 9:25	研究論文 Data Race Detection Based on Dependence Analysis 王冠群（日立オートモティブシステムズ）	研究論文 開発者の所属企業規模を考慮したソフトウェア工学の有用性評価 村上優佳紗（近畿大学）	研究論文 ソフトウェア開発発注者育成のための形式手法を取り入れたプログラミング教育 小田朋宏（SRA）	Future Presentation テストの遊び場を作ったら、一緒に遊びますか？ 浅井真樹子（ワークスアプリケーションズ）
	9:30- 9:55	研究論文 N-gram IDFを利用したソースコード内の特徴的部分抽出手法 小林勇揮（京都工芸繊維大学）	経験論文 自分事化影響要因に着目した中期経営計画立案・展開アプローチ その１：現状分析～計画立案編 安達賢二（HBA）	研究論文 VDM仕様記述と分析の定石と手筋 佐原伸（法政大学）	
	9:55- 10:05	**休憩**			
	10:05- 10:30	研究論文 ソースコードの品質向上を目的とした特定のコメントを検出するツール 田上諭（宮崎大学）	経験論文 コンテンツ開発プロジェクトとソフトウェア開発プロジェクトのマネージメント知見の類似性探究の試行報告 日下部茂（長崎県立大学）	経験論文 日立グループにおける働き方改革の「カタ」 ～マルチタスクに注目した設計・開発の改善プログラム～ 八木将計（日立製作所）	Future Presentation エンジニア人材を死滅させるマイクロマネジメントの打破 ～エンジニアを活かし育てるトリセツ活動の提案～ 松尾谷徹（デバッグ工学研究所）
	10:35- 11:00	事例報告 ソースコードに存在する不適切なコメントの検出手法適用事例 甲斐秀一（スカイコム）	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（SCSK）	経験論文 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 阪井誠（SRA）	
	11:10- 12:30	**キーノートスピーチ（2）** [ギャラリー1] **講演題目：プログラミングと教育** 講演者：筧 捷彦 教授（早稲田大学名誉教授、 NPO 情報オリンピック日本委員会理事長、 公益財団法人 情報科学国際交流財団理事長）			
	12:30- 14:00	**昼食**			
	14:00- 18:00	**ワーキンググループ・チュートリアル** [ギャラリー，会議室，学習室] ※4 次の Future Presentation と事例報告は、ワーキンググループの中で発表します。 <WG2> 事例報告 (14:40-15:10) TPI NEXT による現場主導のテストプロセス改善を支援するための手法の提案 河野哲也，高野愛美（日立製作所/ASTERテストプロセス改善研究会） <WG7> 事例報告 (17:00-17:15) ソフトウェア保守 隠蔽の構造 ～なぜ，保守作業は隠されてきたのか？～ 増井和也（ソフトウェア・メインテナンス研究会） <WG7> Future Presentation (17:15-17:30) 道徳性向上後の行動におけるアラートシステム 阿部敬一郎（東筑紫短期大学） WG1 (AJ)：アジャイルの振り返りを、自分達の強みで進化させよう！ WG2 (TP)：テストプロセス改善モデルを理解する - TPI NEXT を例に - WG3 (FM)：仕様で書きたいことと形式仕様で書けること - 道具としての形式手法 - WG4 (AI)：人工知能技術のソフトウェア開発プロセスへの応用 ～AIはソフトウェア開発を変えるか？～ WG5 (ED)：AI時代のソフトウェア技術者教育 WG6 (DE)：ソフトウェア欠陥エンジニアリング WG7 (ET)：エンジニアのトリセツ ～やりがいのある職場と育成～ WG8 (SE)：Software Evolution，保守こそもっと自由に・柔軟に！ WG9 (RD)：要件開発を開発する WG10 (QU)：ソフトウェア品質保証の温故創新 WG11 (ST)：システム理論に基づくSTAMPでのモデリング WG12 (FR)：機能共鳴分析法とシステムズオブシステム WG13 (SD)：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 TS1 (CM)：開発における情報伝達を見直す ※ 会議室は 22:00 まで利用可能。終了時間は、WG/TS毎に変動する事もあります。			

SS2017プログラム		
日付	時間	内容
6/9 (金)	9:00- 14:00	**ワーキンググループ・チュートリアル** [ギャラリー，会議室，学習室]
	14:00- 14:30	**コーヒータイム** [ギャラリー付近]
	14:30- 15:30	**キーノートスピーチ（3）** [ギャラリー1] **講演題目：Software Safety and Dependable Systems** 講演者：Professor Sungdeok (Steve) Cha（Korea University）
	15:30- 16:00	**クロージング** [ギャラリー1] 実行委員長 小笠原秀人（東芝） プログラム委員長 栗田太郎（ソニー）

会場案内図

宮崎市民プラザ 4F



情報交換会

宮崎観光ホテル 西館10Fスカイホール

※ 情報交換会は 18:15に開場し，18:30 に“開始”します．お早めにおいでください．
※ 会場へは，西館の東側エレベータをご利用ください．
※ 名札の着用をお願いします．

その他

最新情報について
SS2017 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://sea.jp/ss2017/news.html

Memo

ソフトウェア技術者協会（SEA）

http://sea.jp/ss2017/

6月に宮崎で「開かれたソフトウェア開発」について話しませんか

ソフトウェア・シンポジウム 2017（SS2017 in 宮崎）

第 37 回目を迎える 2017 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，SS2016 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．

SS2017プログラム委員長
片山 徹郎 (宮崎大学)・栗田 太郎 (ソニー)

【開催要領】
◆日程: 2017年6月 7日 (水)～10日 (土)
6月 7日 (水)～9日 (金):本会議
6月 10日 (日):併設イベント
◇場所 (本会議):宮崎市民プラザ
◆場所 (併設イベント):未定

【論文投稿スケジュール】
◆投稿締切: 2017年 3月3日 (金) ※この場で応相談
◇採否通知: 2017年 4月中旬 (予定)
◆最終原稿締切: 2017年 5月19日 (金)

【募集要領】
◆研究論文 (A4版 5ページ以上 10ページ以内)
◇Future Presentation (A4版 1～2ページ)
◆経験論文 (A4版 5ページ以上 10ページ以内)
◇事例報告 (要旨 A4版 1ページと，
スライド原稿 12枚～18枚)

SSこれまでの開催地

ソフトウェア・シンポジウム（SS）とは

第１回目のソフトウェア・シンポジウムは，1980 年の末に開催された．主催は，ソフトウェア産業振興協会（当時）の技術委員会 (SIA/TC) であった．通常のアカデミックなコンファレンスとは違って，開発現場で働くソフトウェア技術者たちが，自らの経験を通じて獲得した実践的な技術や知識を交流する場が必要だという認識は，この委員会の発足当時から強く存在していたのだが，ある技術調査のためのワーキング・グループ活動をきっかけとして，その成果発表を兼ねたイベントとして，このシンポジウムが企画されたのである．それからほぼ１年後の 1982 年冬には，第２回目がやはり同様なコンテクストで開催された．この２回のプロトタイピングの成功にもとづいて，翌 1983 年以降のソフトウェア・シンポジウムは，毎年６月に定期的に開催するようになった．

ソフトウェア技術者協会 (SEA，「シー」) は，ソフトウェアハウス，コンピュータメーカ，計算センタ，エンドユーザ，大学，研究所など，それぞれ異なった環境に置かれているソフトウェア技術者あるいは研究者が，そうした社会組織の壁を越えて，各自の経験や技術を自由に交流しあうための「場」として，1985 年 12 月に設立しました．

分科会活動

- 環境分科会 (SIGENV)
- 教育分科会 (SIGEDU)
- ソフトウェアプロセス分科会 (SPIN)
- フォーマルメソッド分科会 (SIGFM)
- ソフトウェア品質保証分科会 (SigSQA)
- システムオブシステムズ分科会 (SIGSoS)
- ソフトウェア信頼性分科会 (SIGForce)

支部活動

- 名古屋支部
- 関西支部
- 九州支部
- 上海支部
- 横浜支部
- 東北支部
- 北陸支部
- 広島支部
- 盛岡支部
- 北海道支部（予）

ソフトウェア・シンポジウム 2017
論文集

2017 年 6 月 15 日　初版発行

編者　　小笠原秀人
　　三輪東

発行者　　伊藤昌夫

発行・発売　ソフトウェア技術者協会

〒157-0073 東京都世田谷区砧二丁目 17 番 7 号
株式会社ニルソフトウェア内
ソフトウェア技術者協会 事務局
TEL: 03-6805-8931
http://sea.jp/

ISBN978-4-916227-23-2

ソフトウェア技術者協会