ソフトウェア・シンポジウム 2015 in 和歌山

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 35 回目を迎える 2015 年のソフトウェア・シンポジウムでは，特定のテーマで議論を深めるフォーラムセッションを追加する予定です．また，Future Presentation という発表形式を新設しました．ここでは，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」などを期待しています．このほか，SS2014 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．皆さま，ふるってご参加ください．

今回は，「ソフトウェア開発の未来」をテーマに，皆さま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．

今回，いくつかの新たな試みを企画しています．今回のソフトウェア・シンポジウムが，将来ふり返ったときに，重要な転換点であったことに気がつく会議にしたいです．

SS2015 の開催地は和歌山県です．気候温暖，海の幸・山の幸豊富な桃源郷で，少しネジを緩めた「待ったあり」の議論のなかから，自然な新展開が出てくることを期待できます．

開催概要

■日程 ： 2015 年 6 月 14 日（日曜日）～ 17 日（水曜日）
6 月 14 日（日曜日）： 併設イベント
6 月 15 日（月曜日）～ 17 日（水曜日）： 本会議
■場所（本会議）： 県民交流プラザ・和歌山ビッグ愛
■場所（併設イベント）： 和歌山大学
■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 ： 和歌山情報サービス産業協会，近畿情報システム産業協議会，
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，ソフトウェアテスト技術振興協会，
ソフトウェア・メインテナンス研究会，電子情報通信学会，
日本情報システム・ユーザー協会，日本ソフトウェア科学会，
組込みシステム技術協会，日本 SPI コンソーシアム，
日本ファンクションポイントユーザ会，派生開発推進協議会

スタッフ一覧

実行委員会

実行委員長

鯵坂 恒夫（和歌山大学）

新谷 勝利（新谷 IT コンサルティング）

実行委員

伊藤 昌夫（ニルソフトウェア）

小笠原 秀人（東芝）

落水 浩一郎（金沢工業大学）

岸田 孝一（SRA）

栗田 太郎（ソニー，フェリカネットワークス）

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）

田中 一夫（アーティス情報システム）

田中 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）

プログラム委員会

プログラム委員長

小笠原 秀人（東芝）

西 康晴（電気通信大学）

プログラム副委員長

大平 雅雄（和歌山大学）

プログラム委員

秋山 浩一（富士ゼロックス）

鯵坂 恒夫（和歌山大学）

安達 賢二（HBA）

天嵜 聡介（岡山県立大学）

荒木 啓二郎（九州大学）

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）

臼杵 誠（富士通）

落水 浩一郎（金沢工業大学）

小田 朋宏（SRA）

片山 徹郎（宮崎大学）

北須賀 輝明（熊本大学）

日下部 茂（九州大学）

楠本 真二（大阪大学）

栗田 太郎（ソニー，フェリカネットワークス）

小嶋 勉（DNV ビジネス・アシュアランス・ジャパン）

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）

酒匂 寛（デザイナーズデン）

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）

高橋 芳広（トリプル・アイ企画）

田中 康（東京工業大学）

張 漢明（南山大学）

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 ファクトリーソリューションズ）
野村 行憲（アイシーエス）
野呂 昌満（南山大学）
端山 毅（エヌ・ティ・ティ・データ）
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（ロックオン）
村山 優子（岩手県立大学）
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）
山本 英之（エヌ・ティ・ティ・データ）
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）

事務局

伊藤 昌夫（ニルソフトウェア）
小笠原 秀人（東芝）
栗田 太郎（ソニー，フェリカネットワークス）

ソフトウェア・シンポジウム 2015 in 和歌山 目次

■論文・報告 C1-1「要因分析/チームビルド」

[研究論文]『反復プロセスと欠陥モデリングによるソフトウェア要因分析の改善
～アジャイルな RCA の導入とその効果～』
永田 敦 (ソニー) …… 1

[経験論文]『職種を超えた連携におけるチームビルディング適用とその効果評価』
山川 紘明 (デンソーテクノ) …… 11

■論文・報告 C1-2「形式手法」

[研究論文]『VDM-SL 実行可能仕様による Web API プロトタイピング環境』
小田 朋宏 (SRA), 荒木 啓二郎 (九州大学) …… 20

[事例報告]『大規模複雑化した組込みシステムのための障害診断における
形式手法の適用事例報告』
岡野 浩三 (信州大学), 北道 淳司 (会津大学) …… 26

■論文・報告 C1-3「開発管理」

[研究論文]『ソフトウェア開発状況の把握を目的とした変化点検出を用いた
ソフトウェアメトリクスの時系列データ分析』
久木田 雄亮, 柏 祐太郎, 大平 雅雄 (和歌山大学) …… 27

[研究論文]『スパムフィルタに基づく即時バグ予測ツールの試作』
森 啓太, 水野 修 (京都工芸繊維大学) …… 37

■論文・報告 C1-4「テスト」

[研究論文]『Concolic Testing を活用した実装ベースの回帰テスト
人手によるテストケース設計の全廃』
松尾谷 徹 (デバッグ工学研究所), 増田 聡 (日本 IBM),
湯本 剛 (日本 HP), 植月 啓次 (フェリカネットワークス),
津田 和彦 (筑波大学) …… 47

[経験論文]『安全系組込ソフトウェア開発におけるユニットテストの効率化』
岸本 渉（デンソー） …… 57

■論文・報告 C1-5「プロセス」

[経験論文]『標準プロセスを肥大化させない補完型チケット駆動開発の提案』
阪井 誠（SRA） …… 64

[研究論文]『機械学習を用いたテキスト分類によるライセンス特定のための
ルール作成プロセス支援』
東 裕之輔，大平 雅雄（和歌山大学），
眞鍋 雄貴（熊本大学） …… 70

■論文・報告 C2-1「不具合分析/信頼性」

[研究論文]『バイトコードを用いたテキスト分類による不具合予測』
藤原 剛史，水野 修（京都工芸繊維大学） …… 80

[研究論文]『Blocking Bug の発生原因を理解するための依存関係分析』
金城 清史，山谷 陽亮，松本 明，
大平 雅雄（和歌山大学） …… 89

[研究論文]『報酬構造を考慮したテストケース生成と信頼性評価の効率性』
小澤 公貴，土肥 正（広島大学） …… 99

■論文・報告 C2-2「要求/計画・見積もり」

[研究論文]『ユーザ視点に基づいたソフトウェアアップデート計画に関する一考察』
岡村 寛之，中原 良太，土肥 正（広島大学） …… 109

[研究論文]『ソフトウェア開発工数見積もりにおける外れ値の実験的評価』
小野 健一 ，松本 健一（奈良先端科学技術大学院大学），
角田 雅照（近畿大学），門田 暁人（岡山大学） …… 115

[研究論文]『要求のヌケ・モレを防ぐためのゴール分解方法の提案と実験
－ソフトウェア・シンポジウム 2014 WG3 の事例－』
岡野 道太郎（筑波大学），中谷 多哉子（放送大学）…… 124

■論文・報告 C2-3「プロセスの実践と評価」

[経験論文]『チームビルディング活動の効果測定 ～事例 10 年間の活動成果～』
増田 礼子（フェリカネットワークス），森本 千佳子（東京工業大学），
松尾谷 徹（デバッグ工学研究所），
津田 和彦（筑波大学） …… 134

[経験論文]『大学機関調査研究 IR へのデータ管理成熟度モデル DMM
の軽量な適用』
日下部 茂，大石 哲也，森 雅生，
高田 英一（九州大学） …… 141

[経験論文]『オンデマンド受注生産システム開発へのサービスデザインの適用』
宗平 順己（ロックオン） …… 148

■論文・報告 C2-4「教育/要件開発/欠陥予測」

[経験論文]『ソフトウェアアーキテクチャの授業での取り上げ方について』
小林 洋（東海大学） …… 154

[経験論文]『要件開発プロセスへの PReP モデルの適用』
田中 康（ケイプラス・ソリューションズ/東京工業大学），
後神 義規（日立製作所），光井 邦雄（クラリオン） …… 162

[研究論文]『ソフトウェア欠陥数予測におけるトービットモデルの適用』
村上 優佳紗，角田 雅照（近畿大学），
戸田 航史（福岡工業大学） …… 172

■Future Presentation 1
『ソフトウェア工学に関する研究を発展させるためには』
落水 浩一郎 (金沢工業大学) …… 175

■Future Presentation 2
『Agile XDDP』
八木 将計 (日立製作所), 会田 圭司 (テクマトリックス), 石川 宏保,
山田 謙輔(派生開発推進協議会 T6 研究会),
斎藤 賢一 (エクスモーション),
永田 敦 (ソニー), 星野 充史 (アンリツエンジニアリング) …… 178

■Future Presentation 3
『クラウド・ビックデータとソフトウェアエンジニアリング』
鯵坂 恒夫 (和歌山大学) …… 181

反復プロセスと欠陥モデリングによるソフトウェア要因分析の改善

アジャイルな RCA の導入とその効果

永田　敦
ソニー株式会社
atsushi.nagata@jp.sony.com

要旨

RCA（Root Cause Analysis）やその手法の一つであるなぜなぜ分析は，障害の要因を明らかにし，再発防止，未然防止を行い品質改善するために非常に有効な手法である．その一方で，実践の際，時間がかかり，障害の担当者が時に責められることがあった．そうなると，行う回数は少なくなり，分析のスキルは上達しない．結果として，期待した結果が得られないことがしばしばある．本論文では，欠陥を表現するモデルを採用して分析の見える化を行い，短い分析をイテレーティブに行うことで，負荷を緩和することで，より効果的，効率的に障害のメカニズムを明らかにし，そのモデルから対策を策定するプロセスを提案し，その結果を紹介する．

1. はじめに

RCA はソフトウェア品質を改善するもっとも有効な手法の一つである．その目的の一つは，ソフトウェアの欠陥による問題を分析して，問題発生のメカニズムおよび欠陥が成果物に入り込むメカニズムを明らかにし，その問題の再発や未然防止をすることである．また，RCA の結果を，fault-prone の箇所の推定に使っていくことによって，優先的および重点的にレビューやテストをすることにより，レビューやテストの効率や効果を上げることが期待できる．当初，我々は，RCA の一つであるなぜなぜ分析[2] を使って要因分析を行ってきた．そのテンプレートを図1に示す．

2. RCA の課題

なぜなぜ分析はその期待結果がある一方で，我々がソフトウェアの障害に対して実行した際に，以下のような問題があった．

- 実施に時間がかかる
- 分析の際，障害に関係する担当者（以下，担当者と呼ぶ）を責めてしまう傾向がある．
- 分析のスキルが上がらず，期待する効果が出ないことがある

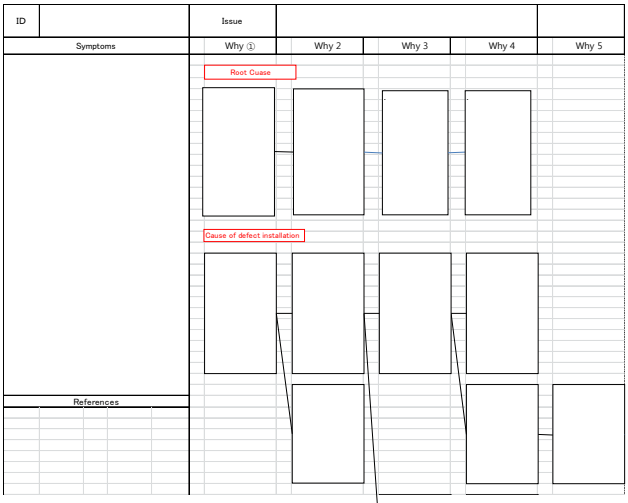


図1　なぜなぜ分析テンプレート

2.1. 実施に時間がかかる

なぜなぜ分析では，二つのプロセスを持っていた．

一つは，分析の準備で，担当者に分析のための説明資料や，場合によってはなぜなぜ分析まで行い，その結果を準備してもらう．

もう一つは，用意された資料を基に，関係者を集め，レビューするプロセスである．

まず，分析の準備で，数時間の時間がかかっていた．そして，レビューには，2時間かかっていた．しかし，1回のレビューでは終わらず，追加の資料作りを含めて，複数回のレビューが必要だった．それにより，担当者にかかる負担はトータルで8時間以上かかっていた．

2.2. ストレスのある分析

なぜなぜ分析は，要因分析の中で非常によく知られ，行われているプラクティスである．“なぜなぜ”分析という名の通り，要因分析のカギになるのは”質問の技術”である．Eric E. Vogt [3] は，”なぜ”という問いは，他の疑問詞よりも，よりよく考え，深いレベルの議論を引き出そうとする傾向があると言っている．それは，”強力な問いかけ”だからであり，それは，思慮深い検討を引き起こし，

創造的な思考を喚起するからである．図2に示すように，”なぜ”は，他の疑問詞に比べ最も強力な疑問詞である[3]．よって，なぜを使えば，質問の力も強い．

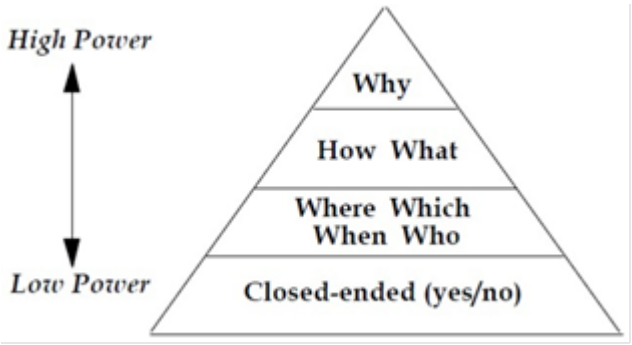


図2　The Art and Architecture of Powerful Questions

もともと，なぜなぜ分析は，トヨタ生産システム(TPS)における生産ラインの要因分析のために生まれたものである TPS では，この強力な質問を用いて工場のラインの要因分析に用いて，非常な効果を上げてきた．（図3参照）

図3　TPS でのなぜなぜ分析

生産ラインにおける欠陥は，物理現象，化学現象による要因で物理的な問題で，分析の対象は，現物である．もちろん，その要因には人間のミスもあるが，それをポカと呼び，ポカよけと言って，ミスを起こさない工夫をしている．

2.3. 分析のスキルが向上しない

一方で，ソフトウェアの欠陥においては，その調査の対象は人間の過ちである．ポカよけを考えるには，人間の過ちそのものを，より分析しなくてはならない．そこで，人間に“なぜ”を投げかけることになる．Eric E.Vogt は，次のように述べている．”なぜ”という疑問文を人に使うときは，気を付けないと，簡単に防御的な反応を引き起こしてしまう(図4参照)．それは，調査をしていこうという意思よりも，彼らの答えを正当化しようとするからである．たとえば，”なぜ思っていることをちゃんと言ってくれないんですか”とか”なぜ，そのようにしたのですか”という問いは，新たな情報を得る可能性を広げる代わりに，相手をして自らの保身と過去に下した決断を正当化することに駆り立ててしまう．[3]

図4　単純にソフトウェア障害でなぜなぜをすると

つまり，防御モードになった時に出される答えは，図5のように，本当に求めている方向からずれてしまう恐れがでてくる．そのずれた答えに対し，さらに機械的に”なぜ”という質問をしまうと，さらにずれてしまう．強力な質問のために，分析の方向が制御できていないのだ．そこから出てくる結果は，真の原因をとらえていないため，そこから出てくる施策も効果的なものが出てこないことがしばしば起こっている．

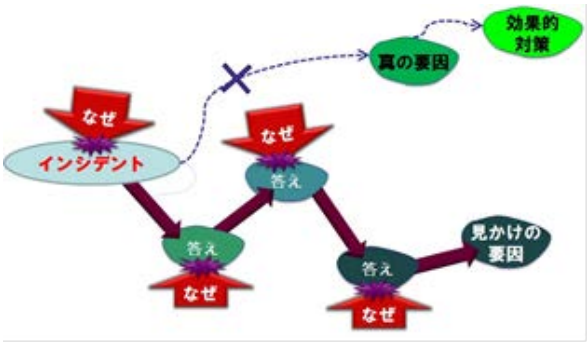


図5　防御モードでのなぜなぜ分析

つまり，RCA で必要なのは，”なぜ”という強力な疑問詞を使った質問を多用するのではなく，状況にあった適切な質問を考え出すことだ．これは，”なぜ”を使ってはいけないということではない．いかにして，人の心をオープンにしながら，本当に知りたいことを聞き出すことができるかということだ．それでは，どうしたら適切な質問を生み出すことができるだろうか．それには，知識，

知見や分析の見える化の仕組みも必要だが，質問のスキル，ファシリテーションスキルも必要になってくる．

2.4. 分析のスキルが向上しない

RCA の実行に非常に時間がかかるということは，RCA を頻繁に行うことができなくなる．ましてや，そのたびに責められるストレスを担当者が受けるのであれば，モチベーションが上がらないため，ますますRCAを行う頻度は少なくなる．2．2節で述べているように，RCA のカギになるのは，適切な質問を作り出すことである．いかに深く分析できるかは，この質問力にかかっている．しかし，”適切”な質問を出していくことは難しく，机上で向上できるものではない．つまり，できるだけ訓練する場が必要なのだが，肝心の RCA を実施する機会が少なければ，実践をし質問のスキルを磨くことができない．そうなると，RCA をしても期待する効果，結果が出ないのでますますやらなくなってしまう．

3. 改善1　適切な質問を出していく工夫

オリジナルのなぜなぜ分析テンプレート（図1）では，表現するのは答えだけである．質問は，”なぜ”がデフォルトだからである．もちろん，なぜを使ってはいけないというのでは決してない．その時，その状況で最適な質問は何が良いのかを考えることが重要なのだ．回答者から多くの情報を引き出すためには，回答者がオープンな気持ちになっている必要がある．そのためには，質問する我々は相手から信頼されていなければならないだろう．そのような関係を保ちつつ，核心に掘り進んでいくためには，ファシリテーションの中心である質問を吟味していかなければならない．質問をテンプレートに明記することは，質問が適確なものかをレビューすることを可能にし，その質問自体にフィードバックをかけ，質問の改善ができるのだ．もちろん，回答する人にとっては，質問を正確に理解することができる利点もある．

そこで，テンプレートを図6のように，質問と回答とを組にして表現するようにした．質問は，できるだけ”なぜ”を使わないようにし，前の質問の答えから，戦略を立て，狙いをつけて質問を考えるようにした．つまり，強い質問によって翻弄されることなく，論理の通った質問と答えの連鎖ができるよう心掛けた．

ただ，適確な質問を立て続けに出していくことは難しい．質問と答えの連鎖が長く伸びてしまうと，その論理性を保ちながら，さらに次の質問をリアルタイムに出していくことは次第に困難になる．それを無理に続けようとしても，質問の質が落ちれば分析を掘り進めることができない．しまいには結論を焦り，生煮な状態で対策案

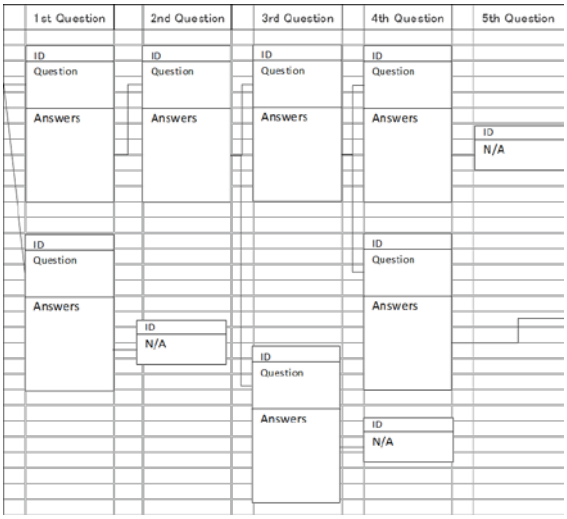


図6　改善1：質問を明示的に表現する

を考えたり，テストで漏れてしまった要因を探ろうとしてしまうことがある．このような状態では，満足する分析結果は得られない．

4. 改善2：分析ミーティングを短くし，繰り返し行う

4.1. 改善1での問題

テンプレートを担当者に渡して分析をまかせ，その結果をレビューするというプロセスをとる場合がある．すると担当者は，横方向にある一つの枝をいっぺんに掘り進めようとする傾向がある．それでは改善1で質問を見える化の工夫をしても，最初の質問が適切でなかったばあい，後の分析は無駄になってしまう．

4.2. イテレーティブ RCA

この改善は，ミーティングの時間を短くし，それを定期的に繰り返すようにしたことである．この理由は二つあり，一つは，回答者の負荷を軽くするためで．もう一つは，前の節で述べた，適確な質問を出していくためである．

ミーティングは，1日1回で長さは15分から最長30分，なるべく，毎日同じ時間に行う様にする．

事前資料を要求せず，最初のイテレーション内で，何が起こったかのみを話してもらう(インシデント分析)．それをシートに書いて，次の質問を2回目のイテレーションまでに考える．

2回目のイテレーションから，考えてきた質問を皮切りに時間内に分析を進めていく．イテレーション内では，質問の答えに対して，不明点やその背景を聞き，回答を整理しまとめていく．3回目以降，これを繰り返してい

く. そのイメージを図7に表す.

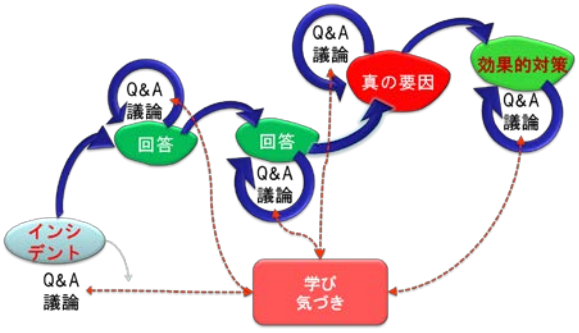


図7 イテレーティブ RCA の流れ

4.3. 改善点と課題点

この施策は，担当者にもこの施策により，回答者の負担が改善したが，より大きなことは質問の改善だった. 分析ミーティングでのレビューアは，質問を出し，その答えを吟味して適切な次の質問を繰り返していかなければならない.

適切な質問を策定する方法は二つ考えられる. 一つは Heuristic な方法で，もう一つは分析的な方法である. 前者はすぐに質問を出せるが的が外れる場合もある. 後者は的を外すことは少なくなるが，それなりの時間がかかる場合がある. 分析ミーティングでは，レビューアは担当者を待たせてじっくり分析をする余裕はないので，Heuristic な方法をとってしまうことが多い. しかし，ミーティングが 30 分を過ぎると，レビューアは集中力を失なっていき，そう簡単に Heuristic に適確な質問を続けられなくなる. やがて質問の質が落ちていき，的をはずしたり真の原因にたどり着く前に質問が出せなくなったりすることがしばしば起こっていた. すると，矛先を原因追求から，そのトラブルがなぜ漏れたのかというテストのほうに向けることになる. しかし，真の原因にたどり着いていないので，結局適切な対策にたどり着かない.

それに対し，分析ミーティングにタイムボックスを使うことによって，質問の質を落とさず集中した分析で終え，そのあとに，分析的な方法で質問を考える時間をとることができる. タイムボックスでイテレーティブに行うことにより，質問の質も改善することができた. さらに，質問と回答の場は，参加者に気づきや知見の伝達ももたらしていた.

一方，このように質問に時間をとっても適切な質問が出せないことがしばしばあった.

その原因として，分析の見える化の仕組みを考えた. 図6のようなテンプレートでは，原因と結果が木構造で

ために利用しているのだ.

連鎖したモデルになる. しかし，実際は，背景や，引き金になるものなど，属性と関係を持っており，その組み合わせで障害のメカニズムを表す必要を感じていた. まとめると次の 3 つのようになる.

- 障害のメカニズムは思ったより複雑である.
- 質問から得られる応答は，要因ばかりでなく，いくつかの属性を持った因子になる.
- それぞれの因子は，関係を持ち影響を与えている.

5. 欠陥モデリング

2013年，ソフトウェアテストシンポジウム東京2013（JaSST Tokyo 2013）において，細川 宣啓氏，野中誠氏，西 康晴氏，原 佑貴子氏，嬉野 綾氏によるプロジェクトファーブル(Project Fabre)によって，欠陥エンジニアリングの紹介があり，そこで欠陥モデリングが提案された. [4]

なぜなぜ分析では，障害の真の原因を求めるために，表現されているのは質問とその答え（その質問に対する原因）であった. 欠陥モデルは，真の原因の代わりに図のような”欠陥”とそれに関連する因子で組み合わせたモデルで，障害の起こるメカニズムと，それを引き起こす欠陥が入り込むメカニズムを表している.

5.1. 欠陥モデリングの要素

Project Fabre では，欠陥モデルの要素として以下のように定義されている.

- 表出現象
  欠陥によって引き起こされる不具合・障害
- 欠陥
  成果物に含まれた，人間の思考の過ちが具現・表出化したもの
- 過失因子
  人間の思考や判断の誤りそのもののこと.
- 誘発因子
  成果物の中に含まれる，人間の思考の誤りを誘発する“トリガー”となる要素のこと
- 増幅因子
  過失の連鎖を助長し，欠陥の混入確率を増幅させる要素

ここで注意しておきたいのは，アジャイル RCA の成果物として作られた欠陥モデルは，Fabre で求めている欠陥モデルではないということである. このモデルの機能の一部を使って，あくまでも障害分析の見える化のために利用している.

6. アジャイル RCA

4.3 で述べた，イテレーティブ RCA の問題を解決するため，図7のようにそれぞれのイテレーションの間にモデリングプロセスを置いた．これをアジャイル RCA と呼ぶことにした．図8はそれをフローダイアグラムとしてあらわしたものである．

プロセスには4つのループがある．

- インシデント分析ループ
- 探索的分析ループ
- アジャイル RCA ループ
- 対策ループ

である．

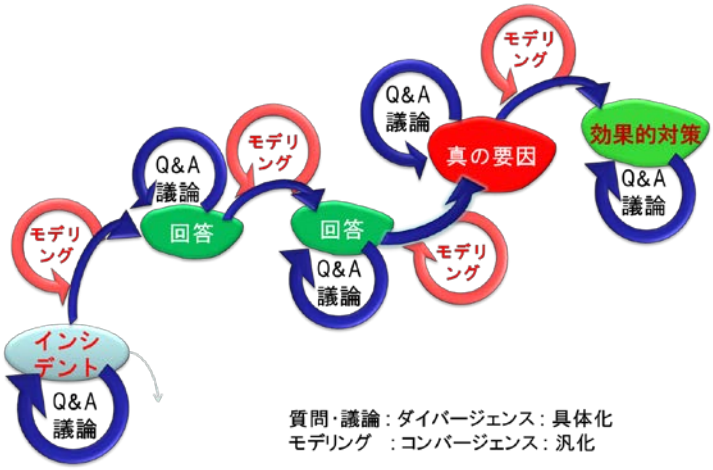


図 7　アジャイル RCA のイメージ

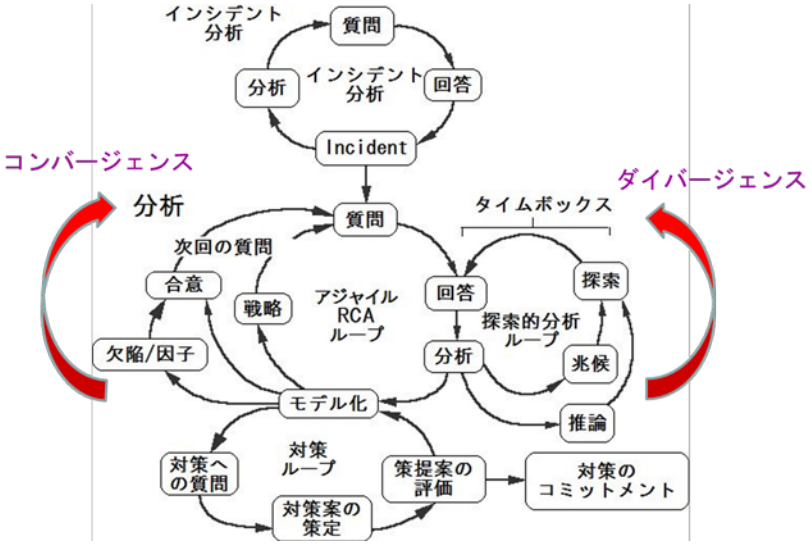


図8　アジャイル RCA プロセス

6.1. ロール

アジャイル RCA のプロセスの説明の前に，プロセスでのアクターであるロールを説明する．3つのロールがある．

- モデレータ
- 担当者
- レビューア

モデレータは，アジャイル RCA のプロセスとルールを理解している．質問を作成し実際に質問をし，分析，モデルの作成を行っていく．

担当者は，障害に関係する直接の担当者である．

レビューアは，モデレータとともに，質問の作成や質問をし，モデル作成を行う人である．

6.2. インシデント分析ループ

最初のイテレーション（初日）は，イテレーティブ RCA と同様にインシデント分析を行う．担当者から，インシデントの情報を正確に聞き出す．図9参照．

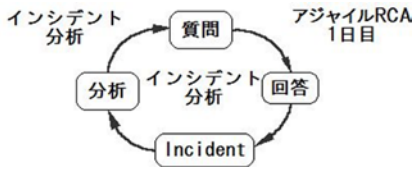


図9 インシデント分析ループ

注意することは，ここでは分析に踏み込まないことだ．インシデントを明確にしていくと，なぜそうなったのか，分析の強い疑問が湧いてしまうことがある．そこで，分析の質問を始めてしまうと，その質問が適切であるかのチェックなしに進んでしまう．それでは，以前と同じことになってしまい，効果がでない．しかも，インシデント自身の時間が短くなり，インシデント分析が不十分になるかもしれない．もし不十分な場合，例えば，その障害の起きた背景，必要なドメイン知識の詳細，そこで使われているプロセス，リソースの状態などが理解されていない場合には，分析者の経験や知見でのバイアスが影響してしまい，欠陥の探索のスコープが，狭まったり違ったりするために，適切な質問ができなくなってしまう．

6.3. 初回の探索的分析ループ

初回の探索的ループ，つまり2日目のアジャイル RCA のイテレーションである．

分析チーム，つまりレビューア，担当者，モデレータが集まり，15分ないし30分のタイムボックスの中で質問と回答，議論を繰り返して分析を行うプロセスである．（図10参照）モデレータやレビューアは，担当者の答から，欠陥やそれに関係する因子の兆候をとらえ，または，それらに対する推論をして，次の質問をしていく．イテレーションの間に，質問と応答の連鎖をできるだけ多く回して，図8にあるようにダイバージェンスしていく．

たとえば，ツールを使い，モデル上に，どんどん書き込んでスクリーンに映し出したり，模造紙に，付箋で質問と答えを書きながら貼り付けたりして見える化いくとよい．これにより，耳で聞いて認識，理解をするだけでなく，目でイメージから認識をして理解していくので，記憶を思い出し，思考を助長する効果があると考える．また，記録として残せることにもなる．

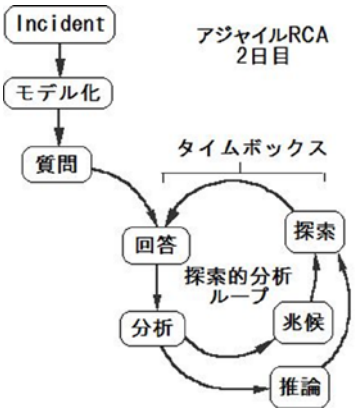


図10 探索的分析ループ

6.4. アジャイル RCA ループ

探索的分析ループで出た出力を持ち帰り，それらを因子に分類整理し，関係をリンクで表現してモデルを作っていく．因子どうしの論理性と，全体の論理の流れを確認することでより根本的な問題の兆候が見えてくる場合がある．その問題を仮説として，もしそうだったらどのような兆候がほかに出てくるかを推論して，それを質問事項としていく．図11参照

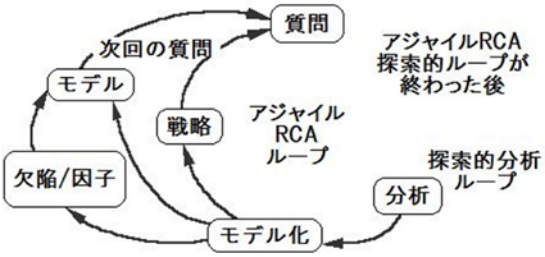


図11 アジャイル RCA ループ

探索的分析ループでは，回答をできるだけ漏らさぬようにメモをしていくのがよいが，内容としては重複していたり冗長であったり重要でないものもあり，それらを整理して，できるだけ少ない因子でシンプルに表現する．因子の内容表現も，より簡潔で正確なものにしていく．図8のように，モデリングをするときは，コンバージェンス，つまり収束させる方向でまとめていく．

6.5. 2回目以降の探索的分析ループ

アジャイル RCA ループで作られた欠陥モデルや質問を用意し，2回目の探索的ループに入る．2回目以降はこれと同じである．

まず，担当者にモデルを見せて，その合意をとることから始まる．違いの指摘を受けることもあるが，すぐに終わる．この欠陥モデルの合意をしたことにより，新たな視

点で分析をすることができる. 図12参照

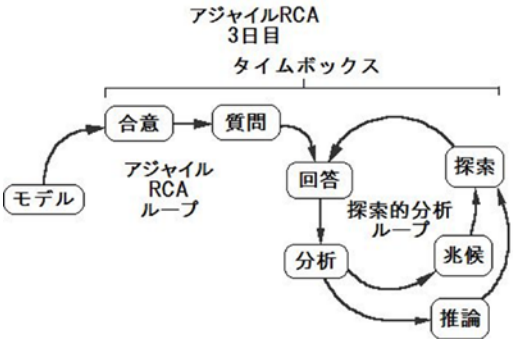


図12　2回目以降の探索的分析ループ

それ以降は，用意していた質問をきっかけに，1回目と同様に探索的分析ループを実行していく.

6.6. 対策策定ループ

何回か，探索的分析ループとアジャイル RCA を交互に行った後，モデルは障害のメカニズムを表すようになる. そこで担当者に対し，対策の質問をして，対策案を提案してもらう. その妥当性を議論しながら，実行していく対策を担当者にコミットしてもらい，プロセスを終える. 図13参照.

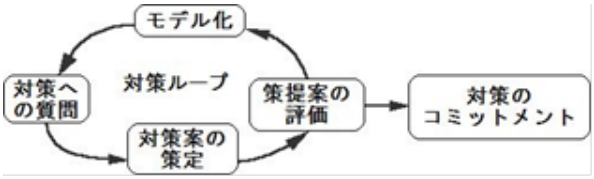


図13　対策ループ

7. 事例

6 章で示したプロセスによって得られる欠陥モデルが，イテレーションごとに変化しながら，欠陥のメカニズムを表し，対策策定まで持っていく様子を，事例を用いて日別に表したものを以下に示す.

7.1. 一日目

図14に示すように，インシデントとそこからわかる二つの欠陥を示している. この例では，インシデント分析をやっていく中で，欠陥がわかってしまった. 設計は最初 QA の報告を，仕様であると返したが，仕様を作っているチームが，これは仕様には載っていないうえに，設計の実装が違っていることを指摘していた.

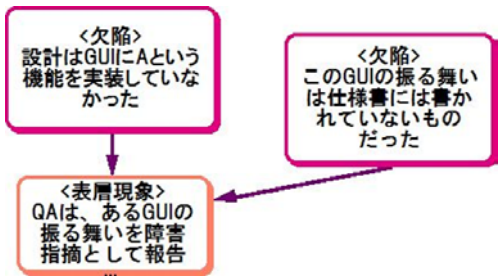


図14　第1日目の欠陥モデル

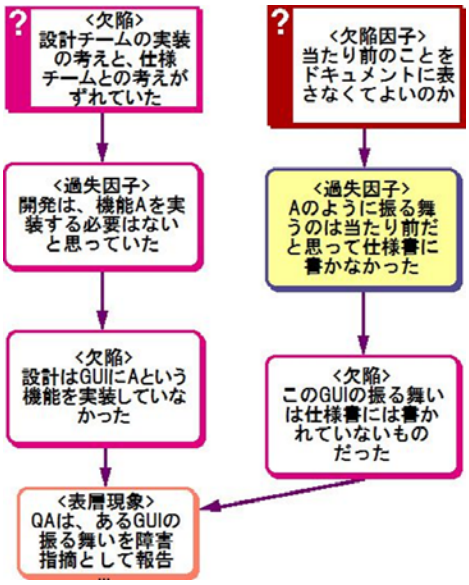


図15　二日目の欠陥モデルの結果

7.2. 二日目

図 15 のように，“当たり前のことは仕様書には書かない”という話が得られた. これは暗黙知で共有している知識や判断基準は，あえて書く必要がないという心理から来ているが，それにより，認識の齟齬が起こっていることに気づかないところを過失とした. その先に，このような“当たり前“という暗黙知をどうした良いかを質問としてモデルに書いている.

7.3. 三日目

図16のように，さらに分析が進んだ. 暗黙的な仕様について共有出来でいないことがわかり，設計と仕様チームとの判断基準が違うことが分かった.

7.4. 最終日

最終的には図17のようになった. 担当者は，振る舞いについてのポリシーがないことがわかり，施策として，仕様チームがポリシーを書くことにした. また，仕様についての理由や背景も書くことをコミットした.

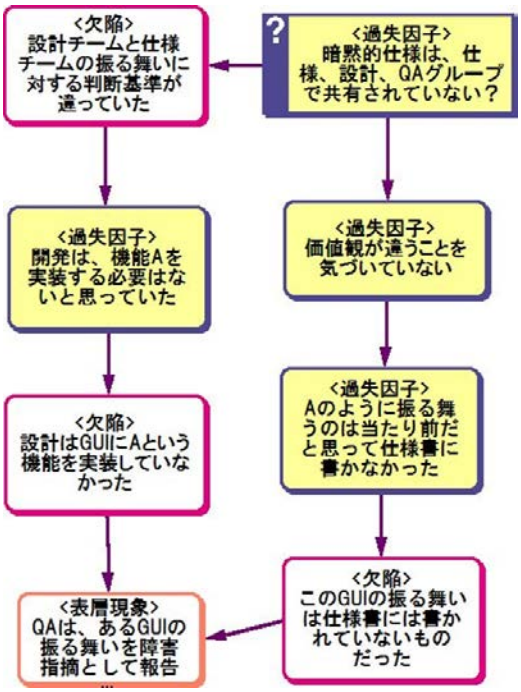


図16　三日目の欠陥モデルの結果

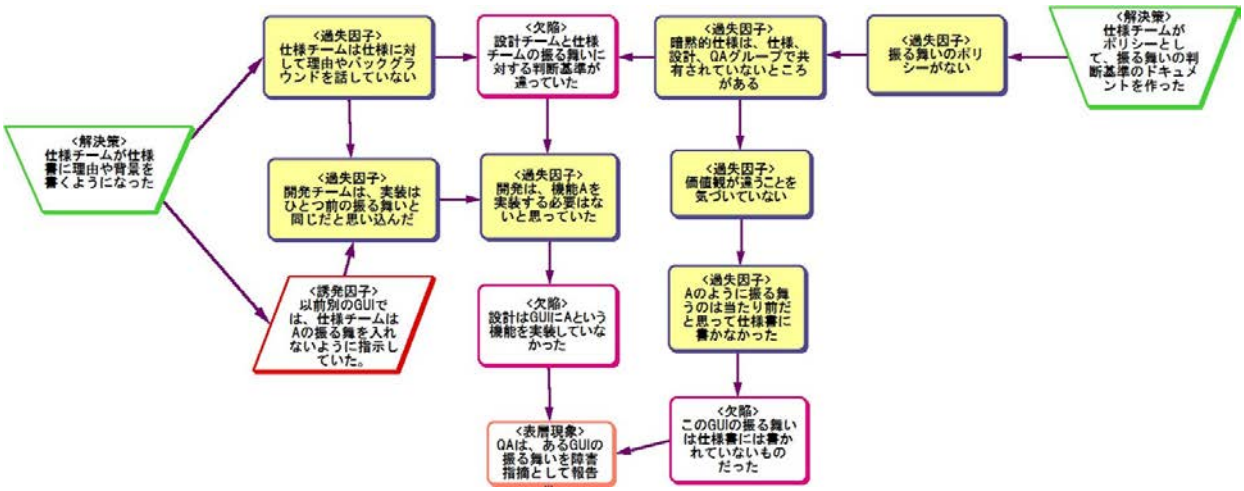


図17　最終日の欠陥モデル

8. 結論

8.1. 旧方式の方法での結果

図1のテンプレートを使っていたころの分析結果は表1のようになっていた.

表1　実行結果

件数	真の原因	分析ミーティング（平均　分）	準備時間（平均　分）	トータル時間（分）
6	1	340	200	540

分析ミーティングは，関係者が集まり，なぜなぜ分析を行い，質問の回答をテンプレートに書き込んでいく会議である．その当時は1回2時間の会議を，3回程度行う必要があった．それに伴い，担当者の準備時間が増えていった．トータル時間は，分析のために担当者がかかわったすべての時間で，平均して 9 時間に及んでいた，

8.2. 実行時間の削減

対象は，アジャイル開発をしている A,B の2チームで行った．A チームは午後決まった時間，B チームは，夕食後決まった時間に行った．

まず，担当者が事前に準備時間は，必要なくなった．報告のために簡単な説明を別途作っている場合は，それを参考に使わせてもらう程度だった．

表2は，二つのチームで行った時の結果である．

表2　実行結果

	課題数	イテレーション	欠陥数	因子数	トータル時間（分）
チームA	6	4	2	15	103
チームB	7	4	4	21	133

課題数　　：扱った課題の数
イテレーション：一つの課題に対するイテレーション回数の平均
欠陥数　　：一つの課題に対する欠陥の数の平均
因子数　　：一つの課題に対する因子の数の平均
トータル時間：アジャイルRCAのトータルの時間の平均．

4回から6回のイテレーションで済んでおり（対策ループを入れると，もう1回増える）トータルで2時間ないし3時間で終わっている．これは，今までの方法より，3倍から4倍速く終わっている．

欠陥は，平均して2個ないし4個程度であるが，それに関係する因子数の平均は，15および21であった．複数の欠陥（真の原因）が確実に見つかっており，それに伴う多くの因子がメカニズムを表していることがわかる．

8.3. 継続性

両チームとも，6か月以上継続して行っていった．半年で，13の障害をトータル56のイテレーションで分析した．大体，月あたり1回ぐらいのペースに見えるのは，相手のスケジュールの都合で休みが入ったりした為である．しかし，開発中の忙しい中で，このように継続できたのは，アジャイル RCA の軽量性と，確実に結果を出していく実績と，それによる信頼関係の構築によるものである．A チームはその後，自分たちで行うようになり，私の手を離れた．

8.4. 欠陥メカニズムの気づき

欠陥モデルにより，欠陥のメカニズムは，より複雑であることが理解できた．これは，最初から過失を防ぐことが容易くできるものではないことを，現場の人たちと共に学んだ．もし同じ条件，同じシチュエーションになったときは，同じように間違えを犯してしまうだろう．

8.5. 学習，伝達および記憶の改善

たとえ数日間，アジャイル RCA のイテレーションの間隔が空いたとしても，担当者に欠陥モデルを見せながら数分説明するだけで，状況を思い出して分析を再開することができた．また，途中から入ってきた人に対しての説明にも有効に使うことができた．

アジャイル RCA で得られる知見は，他の組織のチームにおいて，警告としてガイドすることができた．たとえば，ある障害から，構成管理による問題の欠陥モデルを分析結果として得た．そのモデルの誘発因子から導かれる予兆の情報を使って，他の組織のチームにおける構成管理の欠陥の予兆を進捗会議でつかんだ．そこで，その欠陥から推定されるインパクトをもって警告することにより，構成管理の改善を促すことができた．

9. 考察

9.1. 分析の二つのタイプ

アジャイル RCA のプロセスでは，探索的分析ループとアジャイル RCA ループをイテレーションごとに交互に使っている．

図8のように探索的分析ループは，返ってきた答えに対して，直感的に予兆を探り，推測と仮説を使って次の質問を出して，短い時間に集中して情報を得ていく．そのため，発散的なダイバージェンスタイプの分析になっている．一方，アジャイル RCA ループは，発散した情報をじっくりとモデルに変換し，インスタンスとのリンクを取りながら汎化，抽象化していく．その際，重複したり，重要ではないものはできるだけそぎ落として，エッセンスを残していく．つまりコンバージェンスタイプの分析になっている．次の探索的分析ループを始める前に，ここでできた欠陥モデルを使って，担当者と合意をすれば，より分析するポイントが明確になり，より深い分析をすることが期待でいる．これが，従来のなぜなぜよりも早くできる要因の一つだと考えられる．

9.2. 欠陥モデル

Project Fabre [4] で本来求められている欠陥モデルは，インスタンスを含まず，抽象的な表現になっている．しかし，現場における障害の分析というアジャイル RCA の活動の目的から，欠陥モデルにインスタンスも便宜上入れざるを得なくなっている．これは，欠陥モデルに対する目的が異なっているからである．

9.3. なぜ，アジャイルというか

図8のように，アジャイル RCA は，探索的分析ループと，アジャイル RCA ループが，欠陥モデルを用いてフィードバックを回しながら，分析をイテレーションごとに改善しながら，ゴールに向かっているプロセスといえる．この考えから，アジャイル RCA と名付け提案している．

10. 課題

10.1. 欠陥モデル作成がまだ時間がかかる

30分の探索的分析ループでの結果をまとめるのに1時間ほどかかることがあり，さらなる工夫が必要である．

パターンを使うとより早くなるので，パターンを作っていくことが次の課題になっている．また，欠陥モデルや因子の整理をし，アジャイル RCA 用の欠陥データベースを考える必要もある．

10.2. 欠陥モデルの表現と剪定

剪定の技術がまだ不足で，欠陥モデルが複雑になりすぎてしまう．一見するとマインドマップのように見えてしまい，担当者との間では，理解できても分析に参加していない人が見ると理解しづらくなっている．これは，一緒に共有しながら分析を行っているときに暗黙的な知識が共有され，補っていると考えられる．しかし，この欠陥モデルは，第三者にも知見として伝えていかなければならない．そのためには，もっと理解しやすい，よりシンプルな表現にしていかなければならない．

10.3. モデレータ，レビューアのドメイン知識やプロセスの知識

ドメイン知識やプロセスが違うところでアジャイル RCA をやる機会があったが，そのドメイン知識やプロセスの知識を知らないために，バイアスがかかってしまい，誤った方向の質問を出して分析がはずれてしまい，手戻りが起こってしまった．これは，インシデント分析の不足とも言える．違うドメイン，プロセスでは，インシデント分析でより多くの情報を聞き出すか，事前に調べて，バイアスがかからないようにしていかなければならない．

11. まとめ

何段階かの改善を経て，アジャイル RCA にたどり着き，一定の改善効果を得ることができた．

現在弊社では，一部のアジャイル開発（SCRUM）の振り返りで分析が必要な問題に対して，アジャイル RCA を使っている．現場の設計者に負荷をかけずに分析を行い，適切な施策をフィードバックできるので，振り返りをイテレーションごとに行う SCRUM では，アジャイル RCA を継続して実施するようになる．これにより，開発の時点から，効果的な改善を継続的に行うために，アジャイル RCA を行うモチベーションが生まれた．

また，従来よく使われている市場問題の分析にも，アジャイル RCA が使われるようになってきた．人を責めず，効果のある分析を早く行うことが認められ，ソフトウェアばかりでなく，システム，ハードウェアの問題でも使われ，効果を上げている．

一方，欠陥モデリング自身は，因子や表現方法などに改良の余地が多くある．分析するための欠陥モデルと，報告のため，蓄積のため，第三者に知見として使うための欠陥モデルは，使う人によって変わっていかなければならないかもしれない．

また，モデレータの育成，適切な質問をどのように作っていくかということも，今後考えていかなければならない課題である．

12. 謝辞

アジャイルの本質を教えていただいた，Tom Gilb, Kai Gilb に感謝する．

欠陥エンジニアリングおよび欠陥モデリングの知見を教えていただいた，細川 宣啓氏，西 康晴氏，野中 誠氏，嬉野 綾氏，原 佑貴子氏に感謝する．

参考文献

[1] Ram Chillarege., Orthogonal defect classification-a concept for in-process measurements, Software Engineering, IEEE Transactions on (Volume:18 , Issue: 11), http://www.chillarege.com/articles/odc-concept, 1992.

[2] 小倉仁志，なぜなぜ分析 10 則―真の論理力を鍛える” 日科技連出版社 (2009/03).

[3] Eric E. Vogt, Juanita Brown, and David Isaacs, THE ART OF POWERFUL QUESTIONS: Catalyzing Insight, Innovation, and Action, Whole Systems Associates, CA, USA, 2003

[4] Nobuhiro Hosokawa, Yasuhiro Nishi, Aya Ureshino, Makoto Nonaka, Yukiko Hara, 過失に着目した欠陥のモデリング, JaSST 2013 Tokyo – Project Fabre, 2013

[5] Tom Gilb, Fundamental Principles of Evolutionary Project Management, INCOSE, 2005

[6] Andersen, B., Fagerhaug, T.: Root Cause Analysis: Simplified Tools and Techniques. ASQ Quality Press (2006)

[7] Tomomi KATAOKA*, Ken FURUTO and Tatsuji MATSUMOTO, The Analyzing Method of Root Causes for Software Problems, SEI TECHNICAL REVIEW · NUMBER 73 · OCTOBER 2011

[8] Duke Okes, Root Cause Analysis, The Core of Problem Solving and Corrective Action, ASQ Quality Press, Wisconsin, 2009

[9] Isao Hayakawa et al., Software Quality Symposium 2008, ‘Applying“ask why five times” method on software development,’ pp. 185-194

職種を超えた連携におけるチームビルディング適用とその効果評価

山川　紘明
デンソーテクノ(株)
yamakawa@densotechno.co.jp

要旨

ソフトウェア開発は人的資源の影響を強く受ける. スキル面だけでなく現場力に当たるチームの状態が注目されている. この経験論文は, デザイナーとソフトウェア技術者による職種を越えたチームの現場力向上について実践した報告である. 技術, 人の両側面からチームビルディングに取り組み, その効果評価を行った. 評価には, 先行研究で使われた計測方法を用いた. 社内, グループ会社, IT 業界において計測結果の統計的な分析を行い, 統計値にてチームビルディングによって非常に大きな差が出ていることを確かめた. チームビルディンが現場に与える影響を考察し, 技術, 人に基づくチームビルディングが職種を超えた連携に有効であることを論ずる.

1. はじめに

「現場力」をテーマとする製造業における研究では, 経験的な現場力の定義として, チームの自立的な問題解決能力, OJT(On-the-Job Training) など技能の伝承力, チームメンバが共有する強いモチベーションなどが上げられている. IT 業界における現場力についても近年研究が行われており,「仲間意識」や「役割意識」などが該当することが分かってきている. また, この現場力がプロジェクトの成否に大きく影響することが分かっている. しかし, 製造業の現場力と比較すると IT 業界の現場力は 30 点と言われている[1], [2].

我々は, カーナビゲーションシステム(ナビゲーション)の開発を行っている. ナビゲーションは, 多機能化, 高機能化と価格競争が激しく, 車の開発期間短縮に伴い開発期間も短くなっているばかりか, 車種, 仕向地(国や地域)など多くのバリエーションにも対応しなくてはならない. こうした背景から, ナビゲーション開発においては, 従来の品質を保ち, かつコスト低減と短納期厳守でリリースしていく必要がある. 本稿では, この課題に対し, 新たなリソースの投入ではなく, チームの現場力を向上させるアプローチを行ったので報告する.

ナビゲーションの特徴の一つは, 高度のGUIと多機能の連携にあり, その仕様を決めるデザイナーとソフトウェア技術者の連携が, 生産性や納期に大きな影響を与える. これまでは, デザイナーとソフトウェア技術者の仕事は独立したチームで役割分担し, 開発を進めていた. この方式では, 高度の GUI と多機能の連携を達成できないことから, デザインを意識した UX(User eXperience)指向を実現するためデザイナーとソフトウェア技術者が一体となって開発するスタイルを目指した. 具体的には, チームを 1 つにし, 両者の価値を高めることが現場力の向上に繋がるのではという着想のもと, 職種を超えた連携に取り組んでいる.

本稿は, 以下の構成になっている. 次の 2 章では, 職種を超えた連携の必要性とその実現における問題点を論じる. 3 章では, チームビルディングの取組を紹介する. 4 章では, 取り組んだ活動に対する結果を示し, 5 章で考察, 効果を議論する. 最後の 6 章でまとめる.

2. 職種を超えた連携の必要性と問題点

2.1. 連携の必要性

iPhone やそのほかのタッチ・デバイス(指でタッチすることで操作できる端末)に代表されるように, UX および UI を起点とした情報技術(IT)の革命が起きている.「システムが優れたUXを提供するUIか否か」, によって, 情報システム自体の資産価値も変わってしまう時代になってきている[7]. ナビゲーションも例外ではない.

我々が担当するソフトウェア開発は, デザイン情報をソフトに取り込むなど, デザイン要素が大きく関係する開発となっている. これまでは, デザイナーとソフトウェア技術者の仕事は独立したチームで役割分担し, 開発を進めていた. しかし, 上述したように, 与えられた仕様からプログラムを作るという考え方から, デザインを意識した UX 指向で作る考え方に変化させる必要がある. そのために

は，両者が独立したチーム構成で，それぞれ開発し，デザインとソフトウェアの結合時に初めて UX を意識するのでは遅い．開発当初からデザイナーとソフトウェア技術者が連携し，UX を意識して開発を進めることで，生産性や納期に大きな効果をもたらす．

また，デザイナーとソフトウェア技術者がお互いを知ることで，技術の幅が広がり，両者の価値が向上し，結果として現場力の向上にもつながる．

これらのことから，デザイナーとソフトウェア技術者の連携を開始した．

2.2. 連携での問題点

職種を超えたチーム連携を開始し，ソフトウェア技術者，デザイナーそれぞれにいたマネージャーも一人とし，組織としても 1 つとした．また，作業スペースも席を隣接させ，物理的な距離を縮めた．

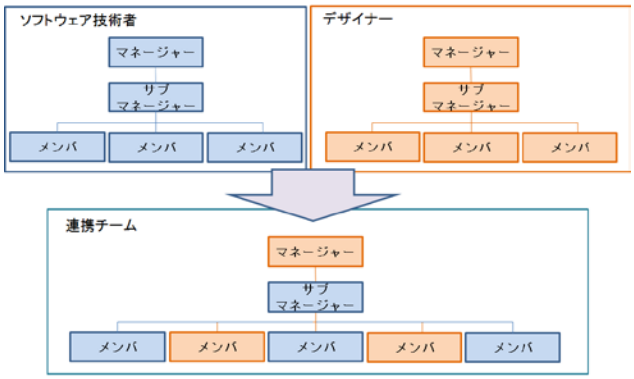


図 1. チーム連携前後の組織構成

しかし，組織編成が変わり，物理的な距離が縮まっても容易に連携が行えるわけではない．次に示す，連携を阻む大きな 3 つの壁が考えられる．

(1) 技術の壁

ソフトウェア技術者とデザイナーでは，今まで作業の進め方や，ツール，成果物等，大きく異なっている．また，コミュニケーションを図ろうとしても，互いの専門用語が理解出来ない．双方，相手が何を知らないのかも把握できない状態であった．このような状態でお互いの技術を習得することは困難である．

(2) 機会の壁

これまで，それぞれ独立したチームで開発を進めており，話す機会は少なかった．技術の壁とも関連するが，組織を1つにしたところで，話すテーマ，議題がなければ両者の話す機会は増えない．

(3) 心の壁

今までのやり方を変えるということは，少なからず負担が発生する．今までのやり方で開発を行ってきた実績があるデザイナー，ソフトウェア技術者にとっては「やりたくない」「なぜそこまでしないといけないのか」という感情が生まれると想定された．

3. チームビルディングの取り組み

チームで新しいことに挑戦する場合，「技術の壁」「機会の壁」にのみ着目して活動しがちである．世間で良いとされている手法や技法を取り入れ，アプローチを行うことが多い．もちろん，このアプローチも有用ではあるが，やりたくない，知りたくないという「心の壁」が残っている状態での活動は「やらされ」となり，活動に身が入らない，活動の本質を理解しないといった状況に陥ることが想定される．そのような状況で連携を行っても，無意識のうちに自身の担当領域に線を引き，いずれ「技術の壁」「機会の壁」が再び現れることになる．1 章でも記載したが，「仲間意識」や「役割意識」のような，人の心理がプロジェクトの成否に大きく影響することが先行研究でも証明されている．そのため，デザイナー，ソフトウェア技術者の連携は技術，機会に着目した活動だけではなく，人に着目した活動も必要であると考えた．

我々は適応的開発手法であるアジャイル（スクラム）開発に挑戦することとした．それは，スクラム開発の「チームを単位に物事を考える」「チームメンバ全員がリーダー」という考え方が，技術，機会，心の壁全てを乗り越えるのに適しているのではと考えたからである．しかし，スクラム開発はあくまで仕組みであり，この仕組みを使ってどう自分たちの活動に落とすかが大事である．そのため，我々のチームビルディングでは，スクラム開発の良さを活かしつつ，デザイナー，ソフトウェア技術者の連携を行うためにはどうすればいいかを考えて活動を行った．

3.1. 技術的側面からのアプローチ

3.1.1. 相互の技術習得

一般的に，数（量・規模）が増えると指数関数的に複雑度が増すが，数が少ないうちは比較的容易に理解で

きるという原理がある. そこで,「スプリント」という考え方に着目した. スプリントは小さい範囲を短い期間で繰り返すという考え方である. 小さい範囲であれば, ソフトウェア技術者はデザイナーの仕事, デザイナーはソフトウェア技術者の仕事に挑戦することが出来るのではないかと考えた. お互いの仕事を体験すれば, 成功, 失敗体験から学びを得て, 他職種の仕事を理解できるようになる. また, 両者が互いに教える, 教えられる立場になることで会話も増えると共に, 体験を共有することで共通の話題も生まれると考えた.

ただし, 両者は他職種の技術が何も分かっていない状態であったため, デイリースクラムにて, 問題や困っていることを積極的にヒアリングし, メンバ全員で共有することでサポートできる体制も充実させた. これにより, 相手が何を知らないのかを理解にし, 必要な時に即座に意思疎通ができる機会を増やし, 両者の知識共有を図ることを行った.

3.2. 人的側面からのアプローチ

心の壁はなぜ生まれるのかをチーム全員で話し合った. そこでは下記のような意見が聞かれた. 大きく分類すると 3 つに分けられる. 1 つ目は,「なぜ連携しないといけないのか」「メリットがよく分からない」という目的が不明確であるという点である. 2 つ目は,「どこまで考えればいいのか分からない」「相手が何をしてほしいのかが分からない」という役割が不明確という点である. ここについては, 相手の技術を理解できていないという技術の壁も関連している. 3 つ目は,「仕事が増えるから嫌だ」「面倒くさい」といったモチベーションによるものである.

そのため, 目的, 役割, モチベーションの 3 つに着目した活動が必要だと考えた.

組織行動論でも, チームとして活動するためには, 目的の共有, 役割の共有, モチベーションの 3 つが必要であるとされている[4][5]. 社会科学の知見からも, 3 つに着目することは有用だと考えることが出来る.

3.2.1. 目的の共有

チームの目的の共有は, チームが形成されて行く「チーム立ち上げ期」と, 本格的にチームが機能する「チーム活動期」では大きく異なると考え, 取り組みは分けて行った. 価値観や文化の異なる両者の連携では, 意見が食い違うことが容易に想像できた. そのため, チーム立ち上げ期に正しくチームの目的を理解し, 意見が食い違った際に自分たち自身が立ち戻る先を明確にしておく必要があると考えた. ただし, 最初に話し合いを設けただけではなかなか定着はしないことが考えられた. そのため, チームで仕事をしていく中で, 目的を腹落ちさせる取り組みが必要であった.

取り組みの着眼点は,「当事者意識」である. チームの目的をより深く理解するためには, 上司から目的を伝えられるよりも, 何故 1 つのチームになったのかを自分たち自身で考えることが有用だと考えた. 人から言われたことは, 他人のせいにできるため受け入れ難いこともある. しかし, 自分のこととして, 自ら考えたことは当事者意識が生まれ, 行動にも変わりやすいと考えた.

(1) チーム立ち上げ期の目的共有

チーム立ち上げ期の取り組みとして, インセプションデッキの作成を行った. チームの目的や役割を考え, 我々はどのようなチームになりたいのかを全員で話し合った. 我々の目指すチームは,「ナビゲーション開発での吉野家になる」と定めた. 吉野家のコンセプトといえば「旨い, 安い, 早い」ということで知られているが, 我々の開発も, デザイナー, ソフトウェア技術者の連携によってうま味, 価値を出す(旨い), 連携により無駄なものを排除しコストを抑え(安い), かつ開発スピードを上げる(早い)という考えから上述の目標とした. また, 目標を達成するための, 職場風土についてもこの場で議題として挙げ, どのように仕事をしたいかを話し合った. ここでは, チームの成長を第一とすることを全員の共通認識とし, 今までの個人で仕事をするスタイルからチームで仕事をするスタイルに変えようということを合意した.

(2) チーム活動期の目的共有

チーム活動期の取り組みとして, スプリント計画を活用した. スプリント計画は, ゴールを正しく捉えているか, 出来そうな道筋なのか, なぜやるのか等を確認し, 仕事の目的を共有する場である. この場で今回のスプリントでは何を旨みにするのかコストやスピードはどうアプローチするかを考え, チームの目的, 方向性を小さな仕事単位で繰り返し感じることができるようにした.

3.2.2. 役割の共有

役割についても「目的の共有」と着眼点は同一である. 自ら考え, それを小さい仕事単位で共有することが必要であると考えた. 理由は目的の共有でも述べたが, 当事者意識を持つことと腹落ちさせることである. そのため, 役割の共有には, 仕事の入り口の活動としてスプリント計

画，日々の活動としてデイリースクラムの場やタスクかんばんを活用した．

(1) 仕事の入口での役割共有

スプリント計画時に，仕事の課題，リスク，懸念点，心配点を考え，それぞれの役割で何をしないといけないかの作戦会議を行った．連携によって解決できる箇所を，仕事の入り口でイメージできるようにし，お互いの役割を共有する場とした．

(2) 日々の仕事での役割共有

仕事の入口で，どのような箇所が注意点なのかを共有できているため，日々のデイリースクラムの場やタスクかんばんを見て，お互いにサポートをでき，ゴールへのアプローチが問題ないことをそれぞれの役割で考え，共有できるようになった．

3.2.3. モチベーション

モチベーションアップのためには，成功体験を持つことが必要だと考えた．自分たちで考え，工夫した行動が成功に繋がれば，喜びに変わり愉しくなる．この着眼点に基づき，2 つの活動に取り組んだ．我々の取り組んだ 2 つの活動は，先行研究の観点からも有効であることが判断できる．IT 業界における仕事満足の実証研究ではモチベーションに影響を及ぼす 7 つの要因[6]が報告されている．我々の活動は，①自己実現・スキルアップの可能性，②自分への評価，④コミュニケーションの状態，⑤プロジェクト運営の体制に該当する．

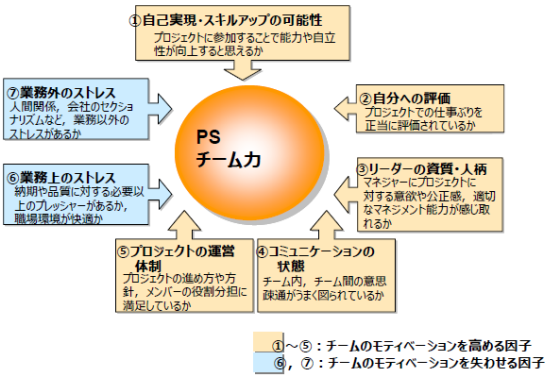


図 2.モチベーションに影響を及ぼす 7 つの要因

(1) KPT 法を用いた振り返り

振り返りは，スプリント単位で継続的に行う活動であり，ここに着目した．連携活動を行っていく中で，自分たちの活動が上手くいったと感じ，それを他者にも認められることが重要である．継続的に振り返りを実施することで，自分たちが出来ていなかった部分が改善出来ていることが実感できるようになる．技術的な部分はもちろんであるが，コミュニケーションや，プロジェクト運営についても振り返り，良いことは「良い」と認め合うことで個人の成長とチームの成長を確認できるようにした．

(2) チーム状態の見える化

自分たちの状態を定量的に表す指標を持つことである．チームの状態を定量的に示すことが出来れば，客観的に自分たちを判断することができるようになる．見えることで，改善意識や向上心も生まれてくるのではないかと考えた．指標は，先行研究「IT に現場力は存在するのか」で示されている心理尺度を利用した質問紙法を用いたメトリクス[1]を使用した．チームの状態を「仲間意識」，「役割意識」，「規範意識」，「成果意識」，「仕事の難易度」の 5 つに分類し，定量的に示した．結果については 4 章で詳細に述べる．

4. 効果検証

4.1. 検証方法

職場におけるチームビルディングの効果を検証する方法として考えられるのは，以下の2つがある．

(1) 職場において，実施前の生産性や納期と比較する．あるいは，同種の他のチームと比較する．
(2) チーム状態を何らかの方法で測り，社内外のチームの状態と統計的な比較を行う．ただし，チームの状態を測るためのメトリクスが必要である．

具体的には，図 3 に示す．

① 成果比較 A-B：
当該チームの対策前と後でチームの成果，例えばトラブルの件数や納期を比較する．
② 成果比較 B-C：
当該チームと同様なチームを探し，その成果を比較する．同じ仕事を行っているチームを探すのが困難であり，非現実的である．
③ チームの状態の比較：A-B：
時系列的に計測し比較するが，メトリクスで使用

する同じ質問紙を使用すると，時間的なバイアスが掛かるので，測定が困難である．

④ チームの状態の比較：B-C：
同じメトリクスを使えば，比較が可能である．

以降の図，表では，チームビルディングをTBと記載する．

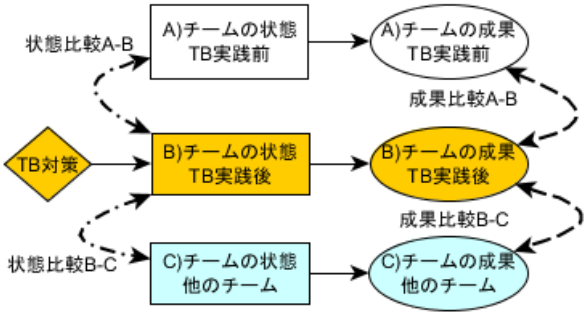


図 3.チームビルディング効果の調べ方

結果として，(1)-①，(2)-④を使って成果，状態の両観点から検証を行うことが出来る．(2)-④の状態比較については，3 章でも述べた「IT に現場力は存在するのか」で心理尺度を利用した質問紙法を用いたメトリクスがある[1]．このメトリクスを使って，先行研究で計測されたデータを(a)社外との比較に用い，それ以外の比較として(b)グループ会社との比較，(c)社内（事業部内）の計測を行い，本稿の検証とする．

4.2. チームの成果比較

4.2.1. 比較検証の対象

成果比較では，一般的に不具合件数の増減や，生産性を比較することが多い．しかし，本活動はプロジェクトの初期フェーズ(作り込み時期)での活動であり，不具合件数の比較には適していない．また，他職種の業務を知るために他職種を体験するなど，生産性を目的とした活動ではないため，生産性の比較も適さない．

連携という観点では，デザイナーとソフトウェア技術者がどれだけチームとして活動できているかを検証することが有用であると考える．

デザイナーとソフトウェア技術者は，従来，不明点がある場合は，組織で定められた質問回答手段である QA 票を用いてやり取りをしていた．この QA 件数がどのように推移したかを確認することで，デザイナーとソフトウェア技術者がチームとして活動できているかが分かる．

また，連携したチームが自立的な改善ができているかは，KPT 法であがった「Try」が「Keep」へ移行できているかを確認することで判断できる．この2点で検証を行った．

4.2.2. 比較結果

連携前はプロジェクト立ち上げ時であれば 100 件近く QA が発生し，回答を得るまでに2～3 日を要していた．これが連携後は，QA での確認ではなく即座に認識合わせをするようになり，待ち時間が発生することはなくなった．QA 件数だけでは判断が断定すること難しいが，両者が他職種の業務領域を理解できるようになったことも要因の1つとして挙げられる．

また，KPT 法であがった「Try」が「Keep」へ移行している件数については，連携直後では数は多くなかったが，連携を進めるうちに改善件数が増えてきていることが確認できた．最初は何を改善すべきかが分かっていなかったチームが活動を続ける中で，改善点を見つけ，改善に向けて活動を行っていくことができるようになったと判断することができる．

4.3. チームの状態比較

4.3.1. 比較検証の対象

現場力の比較は，(a)社外，(b)グループ会社内，(c)社内（事業部内）の 3 つと本稿での活動を行ったチームとの比較を行った．

社外については，「IT に現場力は存在するのか」の実証研究にて，デバッグ工学研究所の松尾谷徹氏が行った，特定非営利活動法人日本プロジェクトマネジメント協会が主催する PM シンポジウム 2013 に集まったプロジェクトマネジャーを対象として行った調査結果を借用した．このデータは，様々なプロジェクトのデータであり，日本の IT 業界の平均的な値と想定できる．

グループ会社内は，デンソー技術研修のハイタレント研修を受講する中堅リーダーを対象とした．ハイタレント研修を受講する人材は各部門のキーマンであり，グループ会社の平均的な値よりは高いと想定できる．

社内は，チームビルディングを実施したチームの他 5 チームに対し計測を行った．

4.3.2. 比較結果

職種を超えた連携とチームビルディングを始めて 6 ヶ月経過したタイミングで，現場力の計測を行ったもの(d)と，前述の(a)(b)(c)とを比較した．

(1) 現場力メトリクスの構成要素による比較

現場力のメトリクスは，質問項目を5つの構成要素に分け，それらについて主成分分析を行い，正規化した得点を算出している．この手法に従い，次の手順で比較を行った．

i. 比較する質問紙の回答群を(d)の回答に対して，構成要素別に主成分分析を行い，正規化した得点を求めた．

ii. 構成要素別に，分布と平均値が異なっているか，箱ひげ図によって比較した．

統計処理は，統計ソフト R を用い行い，その結果を以下の図 4, 5, 6 に示す．

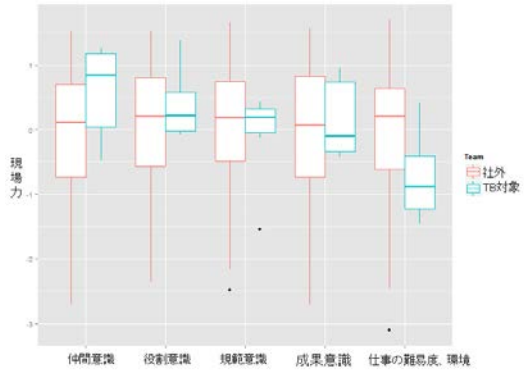


図 4.社外との比較

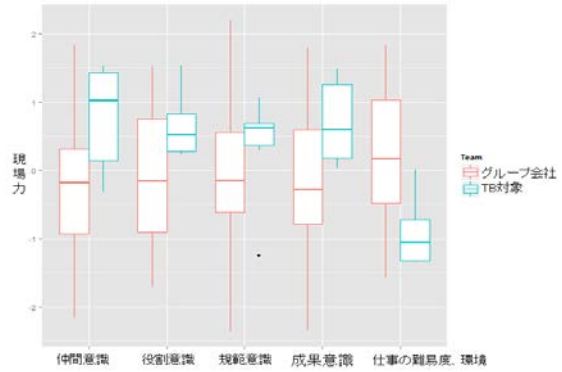


図 5.グループ会社内との比較

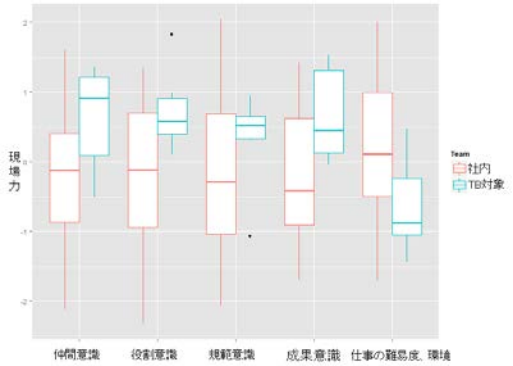


図 6. 社内（事業部内）との比較

現場力メトリクスの構成要素の値の「高い/低い」が，能力の「高い/低い」を表すとは限らない．統計的な主成分分析によって抽出された，違い（分散）を説明するための値である．検証の目的も，「高い/低い」の比較では無く，平均値や分散に差があることを明らかにするためである．

比較結果として，構成要素の「仲間意識」「仕事の難易度意識」については，同じような差が観測された．「役割意識」「規範意識」「成果意識」については，社外との比較では差が認められない結果となった．

(2) 線形判別分析

構成要素別に比較する方法では，多くの構成要素で差があることが明らかになったが，構成要素間の違いや，その違いの差が，全体の差にどのように影響しているのか明らかではない．そこで，現場力メトリクスの項目すべてを使って，判別分析の1つである線形判別分析による比較を行った．判別分析とは，統計的な手法の一つで，与えられたデータ群が，その構成要素である群に分けることができるかを判別する．線形判別分析は，その判別式に線形式を用いたもので，それぞれの観測値に対して判別得点と呼ばれる値を計算する．

比較する対象(a)(b)(c)それぞれに(d)を加え，判別は(d)か，それ以外か，で行った．その結果を表と，箱ひげ図で順に示す．

表 1.全データの線形判別結果

			予測グループ		合計
実測グループ	結果		TB 対象	その他	
	度数	TB 対象	5	1	6
		その他	1	100	101
	%	TB 対象	83.3	16.7	100
		その他	1.0	99.0	100

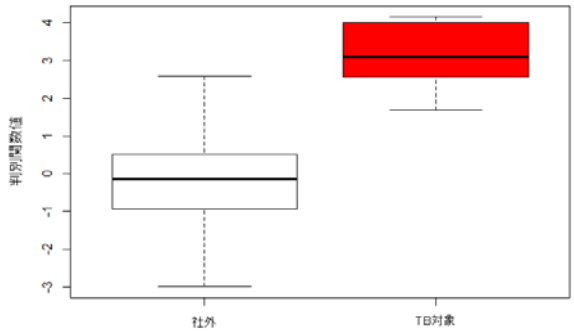


図 7. 全データでの線形判別結果

表 2.グループ会社内の線形判別結果

			予測グループ		合計
実測グループ	結果		TB 対象	その他	
	度数	TB 対象	6	0	6
		その他	0	46	46
	%	TB 対象	100	0	100
		その他	0	100	100

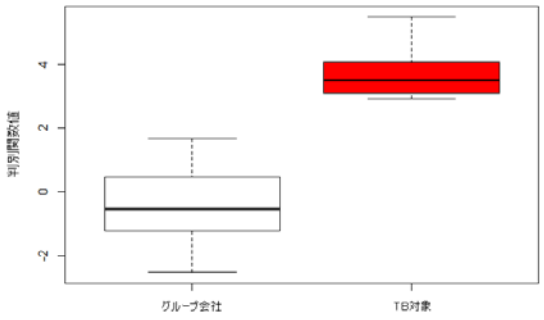


図 8.グループ会社内の線形判別結果

表 3.社内の線形判別結果

			予測グループ		合計
実測グループ	結果		TB 対象	その他	
	度数	TB 対象	6	0	6
		その他	0	20	20
	%	TB 対象	100	0	100
		その他	0	100	100

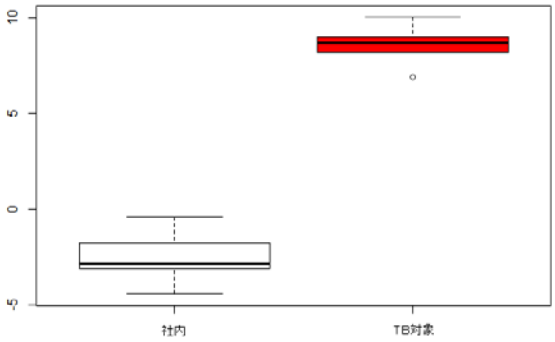


図 9.社内の線形判別結果

表の判別結果は，実測された2つのグループ，「TB 対象」と「その他」に対し質問紙の回答値からグループの区別を計算で予測した結果とを比較している．表1は，実測された TB 対象が 6，その他が 101 であり，これに対し，質問紙の回答から計算した判別では，TB 対象で1件，その他でも1件，判別が誤ることを示している．正解率は，TB で 83.3%　その他で 99%であり，明らかに TB 対象とその他を区別するこができる．図 7 から図 9 は，判別得点の分布を箱ひげ図で示した．この箱ひげ図から，いずれの場合も，TB 対象とその他では，中央値と分散が異なっていることが明らかである．

(3)　開発現場での定性的成果

チームの状態が異なっていることを(1)(2)で示した．どのように異なっているのかについて，定性的ではあるが調べた．分析結果を裏付けるように，チーム内の振り返りでは以下の声が上がっている．

○最良の選択が採れるようなった．

・デザイナーが楽な方法，ソフトウェア技術者が楽な方法を主張せず，お互いの観点で最適解を検討す

るようになった.
・目的とゴールを明確にし，共有することで問題を個人任せでは無く，チームとして扱えるようなった.
○モチベーションが上がる.
・3 週間（スプリント）単位で達成感が感じられる.
・定期的な振り返りにより改善意識が高まる.
・技術の幅が広がり，自身の成長を感じられる.（他人にも認められる）
○問題点の打上げ，補正が早い
・スクラム会議にて，こまめに情報共有が可能となり，問題に対する対応が早くなった. 一人で悩まなくなった.
○開発効率が上がった
・何が本当に必要なのかを考えるようになり，不要な成果物作成，無駄な会議をしないようになった.

5. 考察

検証結果より，デザイナー，ソフトウェア技術者の連携を，人，技術の両面からのアプローチしたことは有用であったと考えることができる. 特に「仲間意識」は高いレベルにあり，個人ではなくチームとして力を合わせ，共に1つの目標を目指すというチームになっている. 現場の声からも，目標に向かって本当に必要なことが何なのかを自主的に考え，チームでコミュニケーションを図ることができるようになったことがわかる.

これはチームとしてのメリットがあるのみでなく，個人にも大きな効果をもたらした. チームのメンバで互いに支えあうことにより，個人の技術の幅が広がり，成長につながる. それを他人に認めてもらい，チームに貢献していると認識することでモチベーションの向上にもつながった. モチベーションが向上したことによりメンバの意識が変わり，改善活動のような非生産業務を始め，プロジェクトの様々な活動に挑戦するようになった. その結果，デザイナーとソフトウェア技術者が共に成長し，互いに支えあう領域が拡大してくというプラスのサイクルにすることが出来た. これは，チーム，個人の両方でパフォーマンスの向上に繋がっていくと考える.

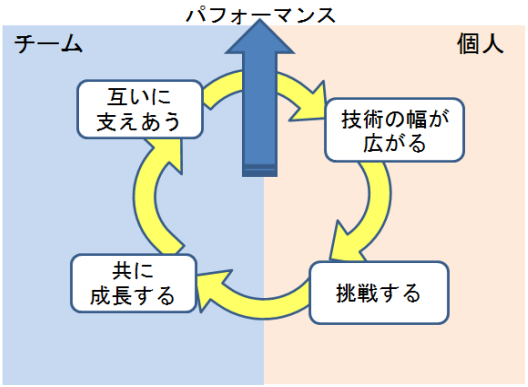


図 10.チームと個人の相互成長サイクル

IT 業界において，組織行動論のような社会科学の知見を適用することは難しいと言われている. その理由は，IT 業界におけるチームの挙動が，一般的な組織（企業）やチームと異なっていることにある[1]. しかし，今回のようなお互いをよく知らないメンバが共通のゴールを目指し，自主自律したチームを作るという観点では，社会科学の適用は十分に可能であると判断することができる.

チームビルディングは，成長を続ける「練習の場」（長期的）とすることも重要である[3]. 本稿では連携を始めて，6 ヶ月のタイミングでの現場力について纏めたが，この状態を維持していくために今後もチームの発展のため活動を行っていく必要があると感じている. そこには今回と同様に，「人間の理論」に着目したアプローチも必要であろう.

6. おわりに

デザイナーとソフトウェア技術者の職種を超えた連携において，技術的なアプローチのみでなく，「人間の理論」に着目したアプローチが十分適用できることが分かった. 社会科学の観点からすれば，当然の結果であるが，IT 業界での適用研究はあまり報告されていない. 今回の適用事例が同様の取り組みをする方々の参考になれば幸いである.

謝辞

本研究の調査実施，及び論文執筆に当たり，デバッグ工学研究所 松尾谷 徹氏，堀田 文明氏，デンソー電子 PF 開発部 足立久美担当次長，デンソー技研センター 技術研修部 古畑 慶次担当課長，上杉 卓司担当

課長に丁寧かつ熱心な指導を賜りました. また調査にご協力いただいたハイタレント研修受講生の皆様, 及び, デンソーテクノ 情報通信 5 部の皆様に感謝いたします.

参考文献

[1] 松尾谷徹 IT に現場力は存在するのか:その計測と評価の試み,ソフトウェア・シンポジウム 2014.
[2] 谷古宇浩司 IT 業界の「現場力」はたった 30 点, http://www.atmarkit.co.jp/news/200607/20/mieru.html
[3] 増田礼子 チームビルディングから組織文化へ,SQiP2014.
[4] 上田泰 組織行動研究の展開, 白桃書房, 2003
[5] 関本浩矢 入門組織行動論, 中央経済社, 2007
[6] 松尾谷徹 7つの要因を知りモチベーションを管理, 日経 IT プロフェッショナル, 2003 年 8 月号
[7] 橋本圭一, UX とは何ぞや？UX を高める武器を手に入れよう, http://www.atmarkit.co.jp/ait/articles/1008/31/news096.html

VDM-SL 実行可能仕様による Web API プロトタイピング環境

小田 朋宏
株式会社 SRA
tomohiro @ sra.co.jp

荒木 啓二郎
九州大学
araki @ csce.kyushu-u.ac.jp

要旨

ソフトウェアシステムの仕様を厳密に記述することで仕様記述の問題点をより早期に発見し修正する開発手法として，形式手法が注目されている．本稿では，形式的仕様記述言語 VDM-SL の実行系を使って実行可能仕様を WebAPI として提供するウェブサーバ Webly Walk-Through を紹介する．

1. はじめに

軽量形式手法[1][2]はソフトウェアを経済的に開発するための形式手法の実践として注目されている．VDM-SL[3][4]は軽量形式手法に適した仕様記述言語として多くの成功事例が報告されている．それらの成功事例では，VDM-SL で記述された仕様が多くの工程の技術者によって信頼できる文書として利用されたことが報告されている[5][6].

軽量形式手法では，形式手法が持つ厳密さおよびツール支援を，開発工程全体のうちより効果的な部分に適用する．自然言語が持つ曖昧さを軽減できることが挙げられる．構文検査や型検査，証明課題の自動生成によるツール支援も大きな利点として支持されている．VDM-SL は実行可能なサブセットを持ち，実行系を利用してインタプリタ実行することができる.

複数の VDM の成功事例で実行系が有効に用いられている．仕様をインタプリタ実行することは，要求獲得やテストやマニュアル作成などに効果的に利用されている[6][7].

我々は VDM-SL 実行系を利用したツールを開発してきた．Smalltalk[8]上の外部インタプリタ接続ライブラリ SOMETHINGit[9][10]を使ったライブ UI プロトタイピング環境を構築し，また，仕様記述の初期段階での試行錯誤を支援するウェブベース仕様記述環境 VDMPad[11][12][13]を開発した.

我々はソフトウェア開発全体における実行系の利用シーンを増やすことにより，形式的仕様記述をより効果的に導入することができると考えている．本稿では，VDM-SL による実行可能仕様の新たな利用シーンとしてウェブアプリケーションに注目し，ウェブアプリケーションの開発において重要な Web API のプロトタイピング環境として開発した Webly Walk-Through を紹介する.

2. Web API 仕様記述言語としての VDM-SL

VDM-SL はモデル規範型の形式的仕様記述言語で，開発対象となるシステムを型，定数値，関数，状態，操作を定義することで記述する．ここでは VDM-SL の言語機能の概要を紹介し，Web API の仕様記述の具体例を紹介する.

2.1. VDM-SL の概要

VDM-SL ではシステムをモジュールに分割し，各モジュールに型，定数値，関数，状態変数，操作を定義する．ここで，関数は参照透明な関数で，操作は状態変数への参照および状態変数への代入を伴う手続きである．関数および操作は，事前条件と事後条件のみを定義した陰仕様と，事前条件および事後条件に加えて処理本体を明示的に定義した陽仕様のいずれかで定義する．陽仕様で定義された関数および操作は実行系によりアニメーション実行することができる．各モジュールにおいて imports 宣言および exports 宣言によりモジュールのインターフェイスを定義し，モジュールをまたがる参照関係を記述する．インターフェイスでは型，定数値，関数，操作を公開することができるが，状態変数を公開することはできない.

VDM-SL 仕様の例として，メールアドレス管理システムの実行可能仕様を示す．メールアドレス管理システムは個人の名前とメールアドレスを管理する．また，各個人はグループを作ることができる．グループはグループ名とメンバーからなるものとする．1 人の個人は任意の数のグループに所属することができるものとする．ユーザは自分と同じグループに属するメンバーの名前とメールアドレスに

```
module AddressBook
exports
  types ID; NAME; EMAIL;
  operations add:NAME*EMAIL==>(); id:NAME==>[ID];
             name:ID==>[NAME]; email:ID==>[EMAIL];
             names:()==>set of NAME;
definitions
types
  ID = nat;
  NAME = seq of char;
  EMAIL = seq of char;
  ITEM :: name:NAME email:- EMAIL
state AddressBookState of
  book : inmap ID to ITEM
  next_id : nat
  init s == s = mk_AddressBookState({|->}, 0)
end
operations
  add : NAME * EMAIL ==> ()
  add(name, email) == let item = mk_ITEM(name, email) in
    book(if item in set rng book
         then (inverse book)(item)
         else get_next_id()) := item;
  name : ID ==> [NAME]
  name(id) ==
    return if id in set dom book then book(id).name else nil;
  email : ID ==> [EMAIL]
  email(id) ==
    return if id in set dom book then book(id).email else nil;
  id : NAME ==> [ID]
  id(name) == return let item = mk_ITEM(name, "") in
    if item in set rng book then (inverse book)(item) else nil;
  names : () ==> set of NAME
  names() == return {name | mk_ITEM(name, -) in set rng book};
  get_next_id : () ==> nat
  get_next_id() == let id = next_id in (next_id := id + 1; return id);
end AddressBook
module Groups
imports from AddressBook types ID
exports all
definitions
types
  NAME = seq of char
state GroupsState of
  groups : map NAME to set of AddressBook`ID
  init s == s = mk_GroupsState({|->})
end
operations
  add : NAME * AddressBook`ID ==> ()
  add(name, id) == groups(name) :=
    (if name in set dom groups then groups(name) else {})
    union {id};
  remove : NAME * AddressBook`ID ==> ()
  remove(name, id) ==
    if name in set dom groups
    then (groups(name) := groups(name) \ {id};
          if groups(name) = {} then groups := {name} <-: groups);
  member : NAME * AddressBook`ID ==> bool
  member(name, id) ==
    return name in set dom groups and id in set groups(name);
  friends : AddressBook`ID ==> set of AddressBook`ID
  friends(id) ==
    return dunion {s | s in set rng groups & id in set s} \ {id};
end Groups
```

リスト 1:メールアドレス管理システムの VDM-SL 仕様

アクセスすることができるものとする. リスト 1 に VDM-SL による仕様記述の例を示す.

上記のメールアドレス管理システムは 2 つのモジュールから成る. AddressBook モジュールは, 個人に ID を割り当てて名前とメールアドレスを管理する. 個人を登録する add 操作および ID から名前をひく name 操作, ID からメールアドレスをひく address 操作を提供する.

Groups モジュールはグループに関する管理をおこなう. 個人をグループに登録する add 操作, 個人の登録をグループから取り消す remove 操作, 個人がグループに属するかを検査する member 操作, 指定された個人と 1 つ以上の共通グループに属する個人の集合を求める friends 操作が提供される.

上記の VDM-SL 仕様は実行系によってインタプリタ実行することができる. 実行を開始すると, 個人およびグループがそれぞれ空の状態でスタートする. 開発者はインタプリタ上で

AddressBook`add("Adam", "adam@example.com")

を実行することで Adam のメールアドレス adam@example.com を登録するなどして記述した仕様が求められる機能を提供しているか妥当性の確認をすることができる.

2.2. Web API 記述の例

Web API の仕様では, 各 API の入力および出力のデータ型, 入力への制約条件, 処理内容が明示されることが重要である.

前述のシステムの VDM-SL 仕様はレポジトリとしての機能を記述したものである. 次にウェブブラウザをクライアントと仮定して, HTTP リクエスト/レスポンスを通してレポジトリの機能をクライアントに提供する Web API の仕様を VDM-SL で記述する. 前節で定義したレポジトリの VDM-SL 仕様から必要な機能を参照することで, 実行可能な Web API 仕様を記述することができる.

リスト 1 の仕様に追加する Web API の仕様をリスト 2 に示す. リスト 2 の API モジュールは, API で利用するデータ型 ENTRY および MEMBER を定義し, それらデータ型を入力および出力とする操作として register, join, leave, list の 4 つの公開操作を定義する. 各 API 仕様は入出力のデータ型および入力値への制約(pre に続いて記述された事前条件)が明示され, 条件を満たした入力が AddressBook モジュールおよび Groups モジュールが提供するどの操作に与えられるかが明示される. API モジュールが定義する 2 つのデータ型は JSON 形式に対応

```
module API
imports
  from AddressBook operations add; id; name; email; names,
  from Groups operations add; remove; member; friends
exports
  types ENTRY; MEMBER;
  operations register:ENTRY==>(); update:ENTRY==>();
             join:MEMBER==>(); leave:MEMBER==>();
             list:seq of char==>seq of ENTRY;
definitions
types ENTRY :: name : seq of char email : seq of char;
types MEMBER :: name : seq of char group : seq of char;
operations
register : ENTRY ==> ()
register(mk_ENTRY(name, email)) ==
  AddressBook`add(name, email)
  pre name not in set AddressBook`names();
update : ENTRY ==> ()
update (mk_ENTRY(name, email)) ==
  AddressBook`add(name, email)
  pre name in set AddressBook`names();
join : MEMBER ==> ()
join(mk_MEMBER(name, group)) ==
  Groups`add(group, AddressBook`id(name))
  pre name in set AddressBook`names();
leave : MEMBER ==> ()
leave(mk_MEMBER(name, group)) ==
  Groups`remove(group, AddressBook`id(name))
  pre name in set AddressBook`names();
list : seq of char ==> seq of ENTRY
list(name) == return
  let s:set of nat = Groups`friends(AddressBook`id(name)) in
    [mk_ENTRY(AddressBook`name(i), AddressBook`email(i))
    | i in set s]
  pre name in set AddressBook`names();
end API
```

リスト 2:VDM-SL による Web API 仕様

させることができる. 例えば ENTRY 型は name および email の 2 つの文字列のフィールドを持つレコード型として定義されているが, JSON フォーマットでは name および email の 2 つのタグを持つオブジェクトデータに対応させることができる.

VDM-SL の陽仕様で記述された Web API の仕様はインタプリタ実行が可能であり, 具体的な入力データを与えて実行することにより妥当性を確認することができる.

3. Webly Walk-Through

インタプリタ実行により妥当性が確認された Web API 仕様は, Web API サーバを実装する開発者と Web API のクライアントを実装する開発者により Web API 仕様書として参照される. Web API サーバのホワイトボックステストにおいて Web API 仕様書がテスト技術者に参照されるだけでなく, Web API 仕様書が実行可能であることから, Web API 仕様の実行結果を Web API サーバの実装のテストに利用することができる.

Webly Walk-Through は, HTTP サーバと VDM-SL 実行系を組み合わせ, HTTP リクエストから VDM-SL の評価式に変換し, 評価結果から HTTP レスポンスを生成しクライアントに応答することで, JavaScript 等で記述されたクライアントプログラムから呼び出すことを可能にする Web API プロトタイピングシステムである. Webly Walk-Through によって WebAPI 仕様をクライアントプログラムから利用可能にすることで, クライアント技術者は Web API の実装完了を待たずに並行してクライアントプログラムの開発に Web API を利用することができる.

3.1. アーキテクチャ

図 1 にシステム構成を示す. Webly Walk-Through は MacOSX および Linux 上で動作する HTTP サーバである. 実装言語は Squeak Smalltalk[8]であり, VDM-SL 実行系として VDMJ[14]を利用する. Squeak Smalltalk と VDMJ のブリッジには, 多言語 UI プロトタイピングライブラリ SOMEHTINGit[9][10]を利用した.

3.2. HTTP サーバおよびユーザインターフェイス

Webly Walk-Through は HTTP サーバとして以下の 3 種類のサービスを提供する.

- ウェブベース開発環境
- 静的ファイルサービス
- Web API サービス

Webly Walk-Through は VDM-SL 仕様で記述された Web API をインタプリタ実行するだけでなく, それ自体がウェブベースの VDM-SL 開発環境およびウェブコンテンツ編集環境を提供する. Webly Walk-Through のウェブベース開発環境は以下の 3 つの画面を持つ.

- VDM-SL 仕様および VDM-SL/JSON 相互変換規則を記述し, 状態変数を監視し, VDM-SL 評価式を実行する VDM-SL 開発画面(図 2)

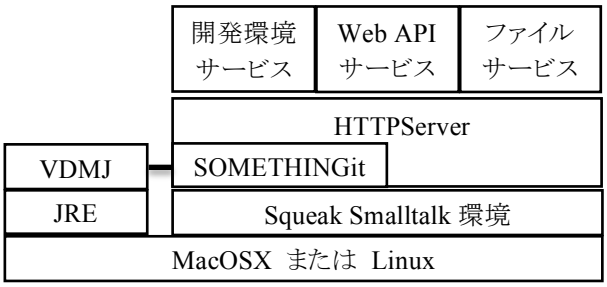


図 1: Webly Walk-Through のシステム構成

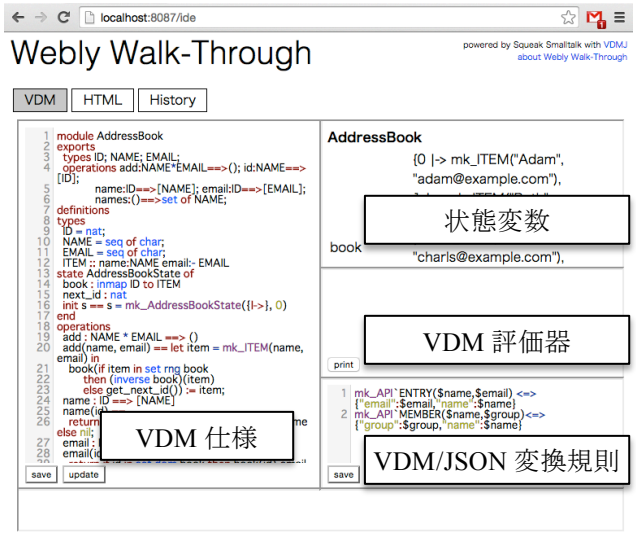


図 2: VDM-SL 開発画面

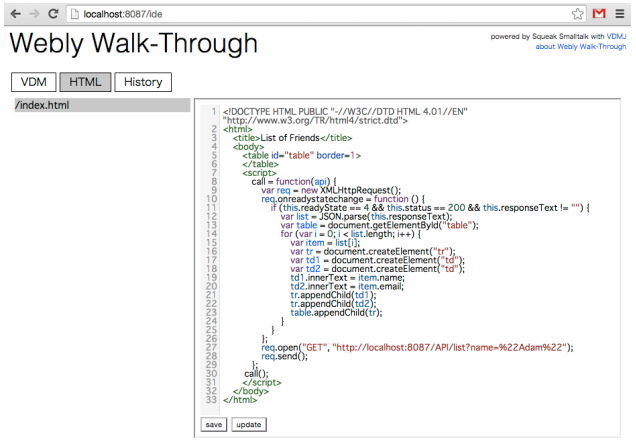


図 3:ファイル編集画面

図 4: 実行履歴画面

- ◆ 静的ファイルを編集するファイル編集画面 (図 3)
- ◆ Web API 仕様の実行履歴を表示する履歴画面 (図 4)

図 2 に示した VDM-SL 開発画面の VDM-SL 仕様部分では，Web API を提供している VDM-SL ソースを編集することができる．ただし，Webly Walk-Through 上での編集は細かな修正を想定しており，第 2 章で紹介したような VDM-SL 仕様記述をスクラッチから記述するのは VDMTools[15][16]や Overture tool 等の VDM 統合環境上で行うことを想定している．

VDM-SL 開発画面では，インタプリタ実行中の状態変数を監視することができる．各モジュールの各状態変数の現在の値が表示される．VDM-SL 評価機部分では，VDM-SL の表現式を評価することで，Web API のクライアントプログラムについて特定の状態をサーバ側に作り出してテストすることができる．

Web API は VDM-SL で記述されているが，クライアントプログラムで Web API に入出力するデータフォーマットとして JSON を利用することができる．VDM-SL/JSON 変換規則を簡単なパターン言語で記述することができる．パターン言語の詳細は次節で説明する．

3.3. Web API の処理

Webly Walk-Through は与えられた VDM-SL 仕様の全モジュールの全操作および全関数を URL の path 部

/ <モジュール名> / <操作・関数名>

で公開する．操作や関数への引数は URL の query 部もしくは POST メソッドのボディ部にフォーム送信フォーマット (application/x-www-form-urlencoded) で送られる．各引数値は JSON フォーマットで表現される．

Webly Walk-Through は Web API クライアントから送信された HTTP リクエストからモジュール名，操作名，引数を取得し，各引数を JSON フォーマットから VDM-SL の表現式に変換し，下記の形式の VDM-SL 評価式を合成する．

<モジュール名>`<操作・関数名>(<引数 1>,…,<引数 n>)

VDM-SL 実行系 VDMJ が上記評価式を実行し，返り値を JSON フォーマットに変換し，HTTP レスポンスのボディ部に格納し，Web API クライアントに送信する．

リスト 2 で定義された API モジュールの list 操作を Web API を通して実行する例を図 5 に示す．

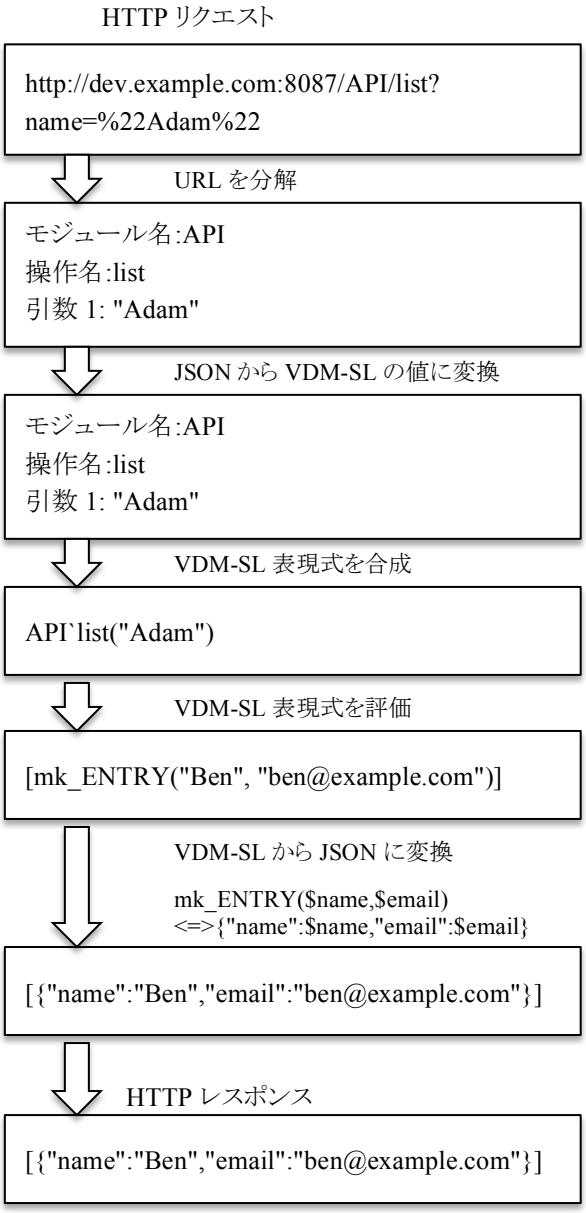


図 5: Web API **の処理の流れ**

4. まとめ

軽量形式手法は高信頼性を求められるソフトウェア開発のみでなく，経済性を求められる開発にも適用されてきた．VDM-SL は仕様に関する検証だけでなく，実行系によるインタプリタ実行が可能であり，仕様書として読むだけでなく，テストオラクルやプロトタイプとして利用することも可能にする．Webly Walk-Through は VDM-SL 仕様の応用をウェブアプリケーション開発における Web API を VDM-SL で記述し，それを Web API としてクライアントプログラムから利用可能にするものである．

ウェブアプリケーション以外にも VDM-SL による実行可能仕様を応用できる領域は多く存在している．Webly Walk-Through は実開発プロジェクトには使用されていない．今後は Webly Walk-Through をはじめとする実行可能仕様を応用したツールの実プロジェクトへの適用を図るとともに，さらなる新しい応用領域のためのツールの開発を継続する．

5. 謝辞

本研究を遂行するにあたって中小路久美代氏，山本恭裕氏，大森洋一氏，日下部茂 氏，佐原伸氏，酒匂寛氏，栗田太郎氏より多くの助言を受けた．ここに謝意を記す．本研究の一部は，JSPS 科研費(24220001) の助成を受けたものである．

参考文献

[1] J. Fitzgerald, P. G. Larsen, and S. Sahara, “VDMTools: Advances in Support for Formal Modeling in VDM,” *ACM Sigplan Notices*, vol. 43, no. 2, pp. 3–11, February 2008.

[2] S. Agerholm and P. G. Larsen, “A Lightweight Approach to Formal Methods,” in *Proceedings of the International Workshop on Current Trends in Applied Formal Methods*. Boppard, Germany: Springer-Verlag, October 1998.

[3] J. Fitzgerald and P. G. Larsen, *Modelling Systems - Practical Tools and Techniques in Software Development*. The Edinburgh Building, Cambridge CB2 2RU, UK: Cambridge University Press, 1998.

[4] P. G. Larsen, B. S. Hansen et al., "Information technology - Programming languages, their environments and system software interfaces - Vienna Development Method - Specification Langauge - Part 1: Base language," December 1996, International Standard ISO/IEC 13817-1.

[5] J. Woodcock, P.G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal Methods: Practice and Experience," ACM Computing Surveys, vol. 41, no. 4, pp. 1-36, October 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
http://www.ipa.go.jp/sec/softwareengineering/reports/20130125.html

［7］ T. Kurita and Y. Nakatsugawa, "The Application of VDM++ to the Development of Firmware for a Smart Card IC Chip," Intl. Journal of Software and Informatics, vol. 3, no. 2-3, pp. 343-355, October 2009.

［8］ D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, and A. Kay, “Back to the future - the story of squeak, a practical smalltalk written in itself,” *ACM SIGPLAN Notices*, vol. 32, no. 10, pp. 318–326, 1997.

［9］ T. Oda, K. Nakakoji, and Y. Yamamoto, “SOMETHINGit: A Prototyping Library for Live and Sound Improvisation,” in *Proceedings of the 1st International Workshop on Live Programming*, ser. LIVE ’13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 11–14.

［10］小田朋宏, 中小路久美代, 山本恭裕, ライブ UI プロトタイピングに向けたマルチ言語環境 SOMETHINGit, ソフトウェアシンポジウム2013論文集, 2013

［11］T. Oda and K. Araki, “Overview of VDMPad: An Interactive Tool for Formal Specification with VDM,” in *International Conference on Advanced Software Engineering and Information Systems (ICASEIS) 2013*, Nov 2013.

［12］小田朋宏, 荒木啓二郎, 形式的仕様スケッチのためのウェブベース開発環境 VDMPad, ソフトウェアシンポジウム 2014 論文集, pp.139-146, 秋田, Jun 9-11, 2014

［13］T. Oda, K. Araki, P. G. Larsen, "VDMPad: a lightweight IDE for Exploratory VDM-SL Specification", *in FME Workshop on Formal Methods in Software Engineering (FormaliSE) 2015 (to appear)*

［14］N. Battle, “VDMJ User Guide,” Fujitsu Services Ltd., UK, Tech. Rep., 2009.

［15］P. G. Larsen, N. Battle, M. Ferreira, J. Fitzgerald, K. Lausdahl, and M. Verhoef, “The Overture Initiative – Integrating Tools for VDM,” *SIGSOFT Softw. Eng. Notes*, vol. 35, no. 1, pp. 1–6, January 2010.

［16］K. Lausdahl, P. G. Larsen, and N. Battle, “A Deterministic Interpreter Simulating A Distributed real time system using VDM,” in *Proceedings of the 13th international conference on Formal methods and software engineering*, ser. Lecture Notes in Computer Science, S. Qin and Z. Qiu, Eds., vol. 6991. Berlin, Heidelberg: Springer-Verlag, pp. 179–194, October 2011

大規模複雑化した組込みシステムための障害診断における形式手法の適用事例報告

岡野浩三
信州大学工学部情報工学科
okano@cs.shinshu-u.ac.jp

北道淳司
会津大学コンピュータ理工学部
kitamiti@u-aizu.ac.jp

要旨

IPA/SEC ソフトウェア高信頼化推進委員会 障害原因診断 WG では，大規模・複雑化した組込みシステムで発生した障害に対する初動調査から障害原因診断分析，整理に至るまでのプラットフォームとして事後 Verification & Validation(V&V)を提案している(発表時には参照先を示す)．本稿では，事後 V&V において，形式手法による障害原因診断の例を述べる．

1. はじめに

大規模・複雑化した組込みシステムでの障害は社会への影響が大きく，多くの設計技法や国際規格が提案されているがそれでも障害が発生する．本稿では，形式手法を用いて障害診断を行った例を述べる．形式手法は，数学的な意味定義が行われた言語および手続きによりシステムのモデリングおよびモデルに対する議論(ある性質が成り立つかどうかなど)が行える．形式手法の一つであるモデル検査手法を化学プラントに適用し，プラントに内在する障害を検出し，モデルを修正するまでを述べる(詳細はスライド参照)．

2. 形式手法を用いた障害診断例

対象は図 1 に示す化学プラントである．2 つの水槽があり，センサ，操作員による操作を入力として，水槽があふれないように制御する．このプラントでは，ある条件において水槽があふれるという障害が発生した．プラントの制御部に着目し，形式手法の 1 つである UPPAAL[1]を用いて文献[2]を参考にモデル化した(図 2)．

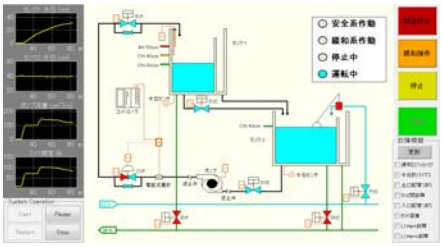


図 1対象とした化学プラント

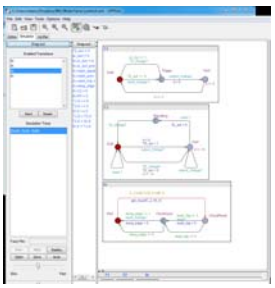


図 2 UPPAAL を用いたモデル

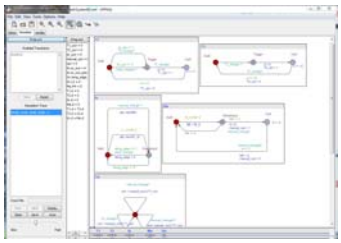


図 3 修正して得られたモデル

一般に，事後 V&V では事故などが発生したあとシステムの資料などからシステムモデルを作成し，そのモデルを解析して事故の原因を究明する．このケーススタディでは図1のシステムの要求記述をもとに図 2 のモデルを作成した．「水槽はあふれない」ことを時相論理式で表し，モデル検査を行った結果，性質が成り立たないことと反例が示された．反例を解析した結果，モデル(すなわち，当初の制御部の設計)にバグがあることが分かり，それを修正したモデル(設計)を作成し，性質が成り立つことを示した．

3. まとめ

組込みシステムの障害に形式手法を適用し，その障害を検出し，取り除いた事例を述べた．形式手法のこのような利用に関する質問，ご意見を伺いたい．

[1] Gerd Behrmann, Alexandre David and Kim G. Larsen,” A Tutorial on Uppaal,” LNCS Vol. 3185, pp 200-236, 2004.
[2] Kim Björkman, Juho Frits, Janne Valkonen, Jussi Lahtinen, et al., “Verification Of Safety Logic Designs By Model Checking,” Sixth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies, NPIC&HMIT 2009.

ソフトウェア開発状況の把握を目的とした 変化点検出を用いたソフトウェアメトリクスの時系列データ分析

久木田　雄亮
和歌山大学 システム工学研究科
s161018@sys.wakayama-u.ac.jp

柏　祐太郎
和歌山大学 システム工学研究科
kashiwa.yutaro@g.wakayama-u.jp

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

要旨

大規模ソフトウェア開発では，進捗および品質の状況を随時把握する必要がある．開発状況を把握するためのモニタリングツールは多数存在しているが，異変が起こっているかの判断はプロジェクト管理者の勘と経験による部分が大きい．そこで本研究では，変化点検出アルゴリズムを用いて，リアルタイムに計測したソフトウェアメトリクスの値から開発状況の異変を機械的に検出するための手法を提案する．*Eclipse Platform* プロジェクトを対象としてケーススタディを行った結果，変化点前後で開発の活動内容が別の活動内容にシフトしていく様子を見てとることができた．また，トピック分析を併用した場合，プロジェクト管理者が変化の内容を理解しながら開発状況を把握できることが分かった．

1. はじめに

ソフトウェア製品の大規模化・複雑化に伴って，高品質なソフトウェアを安定して開発することが困難になってきている．ソフトウェアの品質を維持しつつ開発を行うためには，開発・品質状況を随時把握する必要がある．そのため，ソフトウェアメトリクスのリアルタイム収集および定量的な進捗・品質計測を支援するモニタリングツールが多数提案されてきた [9][3][5]．ソフトウェアメトリクスとは，ソフトウェア開発データの様々な特性から計測可能な定量的尺度 [11] のことであり，ソフトウェア開発における進捗・品質管理にとって必要不可欠な存在である．

既存のモニタリングツールを用いた品質管理では，プロジェクトが計画通りに進んでいるか否か，品質が一定の水準に保たれるかどうかの判断は，未だにプロジェクト管理者の勘と経験による部分が大きいとされる．そのため，開発の遅延やソフトウェアの障害につながる可能性のある異変をいち早く検知するための支援が求められている．

例えば，コード行数（LOC） を計測してソースコードの規模推移をリアルタイムにモニタリングする状況を考える．実装開始直後は通常，LOC の変化（増加量）は日々比較的大きな値をとり，出荷に向けて LOC の変化は小さな値をとる．テスト工程での欠陥修正による LOC の変化は，実装工程での LOC の変化に比べ小さなものであることが多いため，テスト工程でのコード修正にまつわるなんらかの異常を LOC の変化から見て取ることは困難であると考えられる．すなわち，ソースコードの規模推移がすべて見渡せる環境があるが故に，分析者（プロジェクト管理者）は変化の大きさを相対的に認知してしまい，工程や期間毎に意味の異なる変化を見落としてしまう可能性が高い．プロジェクトの異変を，相対的な変化ではなく工程毎のコンテキストに沿った意味的な変化として機械的に検知することができれば，早くに対処することができる可能性がある．

本研究の目的は，変化点検出アルゴリズム [10] を用いて，プロジェクトの進捗・品質状況の悪化などといった

変化を，リアルタイムに計測したソフトウェアメトリクスの値から早い段階で捉えるための手法の構築することである．本論文では，オープンソースプロジェクトから取得したデータに対して変化点検出を行うとともに，トピック分析 [2] を用いて変化の要因を理解することが可能かどうかを確かめる．

2. モニタリングツールによるソフトウェア管理

大規模かつ複雑な作業の組み合わせを経て開発される近年のソフトウェアを，定められた納期に定められた品質で開発するのは容易ではない．ソフトウェアをより安定して開発するためには，開発者の能力や開発対象によって品質が左右されないようにリソース割当計画や開発プロセスを策定し，進捗・品質を管理することが必要となる．プロジェクトの状況を把握し問題点を発見・対処するために，プロジェクトの状況を可視化するモニタリングツールが多数存在している．本章では，既存のモニタリングツールをいくつか紹介し，本研究の立場の違いを説明する．

2.1. プロジェクト管理を目的としたモニタリングツール

プロジェクトモニタリングツールのひとつとして，大平らの EPM(Empirical Projet Monitor) [9][4] がある（図 1）．EPM は，リアルタイムのプロジェクトモニタリングを目的としたツールであり，より安定したソフトウェア開発のために開発状況を随時把握することを支援する．EPM は，バージョン管理システム，メーリングリスト管理システム，障害管理システムのリポジトリに保存された履歴データを収集して利用する．収集したメトリクスデータから，ソースコードの規模推移やチェックインとチェックアウトの関連など，グラフィカルにプロジェクトの状況を表現することができる．

Teamscale[3] は，リアルタイムにソフトウェアの品質を管理することを目的として開発されたツールである（図 2）．ソフトウェアの品質状況を示す様々な指標を可視化する．バージョン管理システムに開発者が差分ソースコードをコミット（登録）した直後に，分析結果をフィードバックすることができる．開発者が普段使う開発ツール（Eclipse, や Visual Studio など）に組み込むことができる．

Cisco の Software Quality Dashboard(SWQD)[5] は，顧客に関するメトリクスを取得しモニタリングする（図 3）．例えば，取得しているメトリクスの 1 つに CFD (Customer-Found Defect) があり，顧客が発見した欠陥の数を取得して表示していることが見て取れる．

Hackystat[1] は，ソフトウェアのプロダクトデータとプロセスデータを収集・分析し，結果を可視化するツールである（図 4）．Hackystat は，プロジェクトの開発者や管理者がプロジェクトの開発状況を把握することに用いられる以外に，教育分野でも用いられている．

2.2 既存のモニタリングツールの問題点

既存のモニタリングツールは基本的に，取得・計測した値のみを可視化する．現在プロジェクトが順調に進んでいるのか停滞しているのかといった判断は，管理者や開発者の経験や勘に頼る部分が大きい．特に，問題発見は管理者および開発者の判断で行われるため，プロジェクトの遅延につながる問題や，後に大きな障害となりそうな可能性のある変更（code change）が混入したかどうか，などを時系列のグラフや値から読み取ることは難しい．プロジェクト遅延や品質低下を招く原因となりうる変化を機械的にできるだけ早く見つけることができれば，手戻りコストを減らすことができると言える．

本研究では，プロジェクトの進捗・品質管理において今後大きな問題につながる変化を，開発者や管理者の勘や経験に頼らず，自動的にリアルタイムで抽出する手法を提案する．

3. 変化点検出を用いたソフトウェアメトリクスの時系列データ分析手法

本章では，変化点検出を用いたソフトウェアメトリクスの時系列データ分析手法を提案する．提案手法は，大きく分けて，変化点検出アルゴリズム [10] をソフトウェアメトリクスに適用するための処理と，検出した変化点の前後でどのような話題が議論されていたかを分析するためのトピック分析 [2] からなる．

3.1 変化点検出

3.1.1 データ収集とメトリクス集計

版管理システム，不具合管理システム，メーリングリスト管理システムなどのリポジトリからメトリクスデー

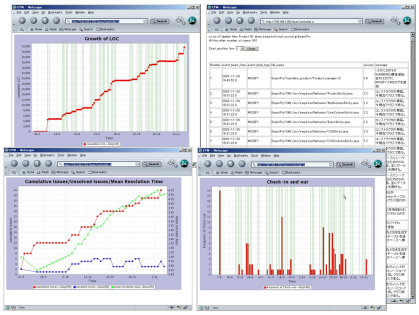


図 1. EPM[9] の使用例

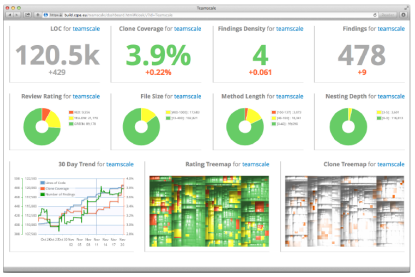


図 2. Teamscale[3] の使用例

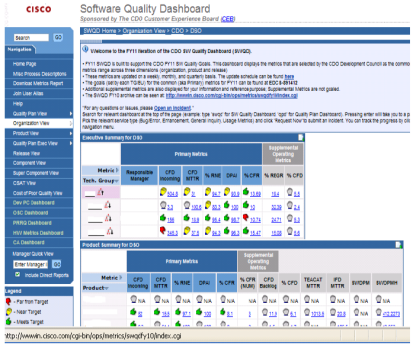


図 3. SWQD[5] の使用例

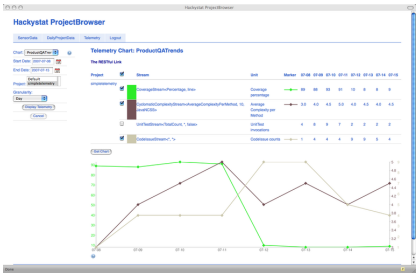


図 4. Hackystat[1] の使用例

タを時系列データとして取得する．リポジトリの種類によって，時系列データの整形方法は異なる．

版管理システム：コミットログに従ってリビジョン毎にソースコードからメトリクスを抽出する．さらに，各リビジョンのタイムスタンプに従って，データを 1 日単位にまとめて集計する．例えば，1 日に 2 回のコミットがあり，それぞれ異なるファイルにコードを追加している場合は，二つのファイルの追加行数の合計がその日のソースコード増加行数として集計される．

不具合管理システム：不具合管理システムに登録された不具合票には，報告日時，割当日時，修正完了日時などの異なるタイムスタンプが記録されているため，取得するメトリクス毎にタイムスタンプを区別して集計する．例えば，不具合報告数は，すべての不具合報告の報告日時を計測し，1 日毎に集計して算出する．

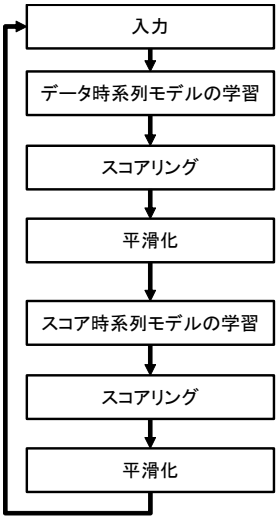
メーリングリスト管理システム:メーリングリストデータからは送信（受信）日時を用いて，メール送信数や送信者数を 1 日毎に集計する．

なお，本稿のケーススタディでは時間の最小単位を 1 日としたが，上述のデータをまとめることで 1 週間単位や 1ヶ月単位での時系列データを作成することもできる．

3.1.2 変化点検出

集計したメトリクスの時系列データに対して変化点検出アルゴリズム [10] を適用する．変化点検出とは，データマイニングの分野で用いられている異常検知手法の 1 つである．変化点とは，時系列データのモデル（トレンド）の変わり目を指す．一時的なデータの大きな変動（外れ値等）は変化点とはみなさず，あるデータの大きな変動がある程度継続して続く場合，本質的なデータの変動を変化点としてみなす．

本研究では，ChangeFinder[6] を用いた変化点検出を行う．ChangeFinder とは時系列モデルの 2 段階学習に基づいている．時系列モデルとは，過去の時系列データをもとにして将来の時系列データを予測するために定式化されたものである．時系列モデルには様々な種類のモデルが存在しており，自己回帰モデル（Auto Regressive Model），移動平均モデル（Moving Average Model），自己回帰移動平均モデル（Auto Regressive Moving Average Model）などがある．ChangeFinder では，時系列データに対して自己回帰モデル（AR モデル）を用いてモデル化し，SDAR アルゴリズムを適用し，学習する．AR モデルは，時系列データの定常性を仮定した下でし



$$x_t \quad (1)$$

$$P_{t-1}(x) \quad (2)$$

$$Score(x_t) = -\ln P_{t-1}(x_t) \quad (3)$$

$$y_t = \frac{1}{T}\sum_{i=t-T+1}^{t}(-\ln P_{t-1}(x_i)) \quad (4)$$

$$Q_{t-1}(y) \quad (5)$$

$$-\ln Q_{t-1}(y_t) \quad (6)$$

$$Score(t) = \frac{1}{T}\sum_{i=t-T+1}^{t}(-\ln Q_{t-1}(y_i)) \quad (7)$$

図 5. 時系列モデルの 2 段階学習

か扱うことができないが，SDAR アルゴリズムを適用することにより 非定常な時系列データに対応することができる．SDAR アルゴリズムとは AR モデルの忘却学習アルゴリズムのことである．忘却学習とは，過去のデータが現時点からさかのぼればさかのぼるほどモデルの学習に与える影響が減少する学習のことである．SDAR アルゴリズムでは忘却パラメータを設定することにより忘却型学習を行っている．この忘却型学習によって，AR モデルを用いて非定常な時系列データを学習することができる．

図 5 に，時系列モデルの 2 段階学習のアルゴリズムを示す．式 (1) で時点 t におけるデータ x_t を入力として受ける．式 (2) の $P_{t-1}(x)$ は $x^{t-1} = x_1, ..., x_{t-1}$ から SDAR を用いて学習 (第 1 段階の学習) をした確率密度関数を示している．式 (3) の $Score(x_t)$ は対数損失関数である．つまり，入力されたデータ x_t が P_{t-1} からどの程度外れているかを表している．式 (4) では，式 (3) で求めた $Score(x_t)$ の移動平均を求めることで平滑化し，y_t とする．平滑化することにより，一時的なデータの変動に対して反応した外れ値を除去する．T は移動平均のウインドウサイズで，与えられた正数である．式 (5) の Q_{t-1} は新しく得られた時系列データ $\{y_t : t = 1, 2, ...\}$ の確率密度関数で，式 (2) と同様に $y^{t-1} = y_1, ..., y_{t-1}$ から SDAR を用いて学習 (第 2 段階の学習) をして得る．式 (6) は式 (3) と同様に，y_t が Q_{t-1} からどの程度外れているかを表している．式 (7) で得られる $Score(t)$ が変化点スコアである．第 1 段階の学習の時点では，外れ値と変化点が両方がスコアとして高い値が出力されるが，式 (4) で平滑化を行うことで，外れ値を除去できる．そして，第 2 段階の学習によって本質的な変動のみが $Score(t)$(変化点スコア) の値が高くなる．

3.2 トピック分析

変化点検出によって検出された変化点の前後でどのような議論がなされていたのかを確認できれば，変化点の意味を理解するために役立つと思われる．提案手法では，変化点として観察される事象が発生した要因を特定するための方法としてトピック分析を用いる．

3.2.1 トピック分析

提案手法では，後述する LDA に基づくトピック分析を，メーリングリストや不具合報告のコメントデータに適用する．トピック分析を用いることで，それぞれのコミュニケーションメディアで行われた議論の要点をトピック（話題）の分布として捉えることができる．提案手法では，変化点が検出された前後のトピックを，設定した期間（1 日や 1 週間など）毎に集計する．トピックの移り変わりを見ることにより，変化点の前後でプロジェクトに何が起きていたのか理解することを支援する．

3.2.2 潜在的ディリクレ配分法（LDA）

潜在的ディリクレ配分法は，文書の生成を確率的にモデル化したトピックモデルである．トピックモデルとは，文書内に潜在しているトピック（話題）を推定するモデルである．本手法でのトピック分析において潜在的ディリクレ配分法（Latent Dirichlet Allocation）[2] を用いる．ひとつの文書には複数のトピックが存在し，それぞれのトピックがある確率によって文書内に生起するという考えのもと，それぞれのトピックの確率分布を求める手法である．文書に含まれる可能性の高いトピックを抽出することで，文書内に潜在しているトピックを推定できる．

近年では，LDA はリポジトリマイニングの分野で使われることが多い [7][8]．主にリポジトリのデータに対する開発者の理解を手助けする目的などで用いられている．

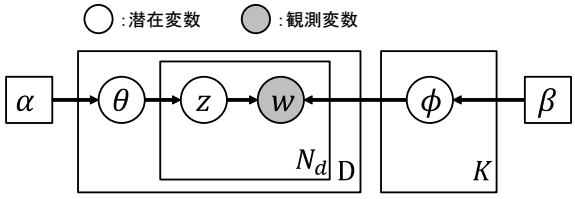


図 6. LDA のグラフィックモデル

図 6 に，潜在的ディリクレ配分法のグラフィックモデルを示す．文書毎にトピック分布 θ を持ち，文書上の各単語について，θ によってトピック z が選ばれ，トピック z の単語分布 ϕ に従って，単語 w が生成される．K はトピック数，D は文書数，N_d は文書 d 上の単語の出現回数を表している．また，α は θ が従うディリクレ分布のパラメータで，β は ϕ が従うディリクレ分布のパラメータである．

4. ケーススタディ

提案手法の有用性を確認するために，本論文ではオープンソースプロジェクトの 1 つである Eclipse Platform プロジェクトを対象としたケーススタディを行う．

4.1 分析対象データ

分析対象とするデータは，Eclipse Platform プロジェクトのソースコード，ソースコードの変更履歴，不具合報告，メーリングリストである．対象とする期間は 2003 年 7 月から 2008 年 7 月までの 5 年間とする．

4.2 分析方法

本ケーススタディにおける分析の手順を説明する．

1. **メトリクスの抽出**：まず，Eclipse Platform プロジェクトが利用している版管理システムから，ソースコードおよびソースコードの変更履歴（コミットログ）をすべて取得する．次に，コミットログに従い，リビジョン毎にソースコードを取り出す．その後，ソースコード解析ツール Understand を用いてコードメトリクスを抽出する．本ケーススタディでは，コード行数（LOC）およびサイクロマティック数をリビジョン毎に抽出した．また同様に，不具合管理システムから不具合報告に関するメトリクスを抽出する．本ケーススタディでは，不具合報告数，不具合解決数，修正待ち不具合数を 1 日毎に抽出した．
2. **コードメトリクスの時系列データの整形**：コードメトリクスデータを時系列データにまとめる．各リビジョンのタイムスタンプに従い，メトリクスデータを 1 日単位の時系列データに整形する．不具合メトリクスは，抽出時点で時系列データとなっているため，本ステップでは何も行わない．
3. **時系列データを用いた変化点検出**：コードメトリクスおよび不具合メトリクスの時系列データを入力として，変化点検出を行う．ChangeFinder を使う際に設定するパラメータには，AR モデルの次数，平滑化のウィンドウサイズ，忘却パラメータの 3 つがある．本ケーススタディでは，それぞれ 4，7，0.01 として変化点検出を行った．
4. **トピック分析**：開発者のメーリングリストおよび不具合報告に含まれるのコメント（テキストデータ）に対して，1 週間毎に LDA を適用する．設定するトピック数が多すぎると分析が困難になるため，本ケーススタディでは，トピック数を 10 個に設定する．
5. **変化点の要因特定**：高い変化点スコアが現れた時点の前後 1 週間のトピック分析の結果を抽出し，各トピックに含まれる単語を確認する．変化点前後におけるプロジェクト内での議論内容を確認できるため，大きな変化点が計測された要因が特定しやすくなる．本ケーススタディでは，変化点スコア 5 以上の変化点の前後 1 週間のトピックを抽出することとした．変化点スコアを 5 に設定した理由は，5 以上の変化点スコア計測されることは本データセットでは稀であったことから，コード行数およびサイクロマティック数に大きな変化をもたらすイベントが生じた可能性が高く，それらに関連する議論がなされている筈と考えたためである．

4.3 分析結果

4.3.1 ソースコードメトリクスでの変化点検出結果

コード行数（LOC）およびサイクロマティック数に対して変化点検出を行った結果を次の図 7，図 8 に示す．横軸は 1 日単位の時系列を表し，左縦軸はそれぞれコー

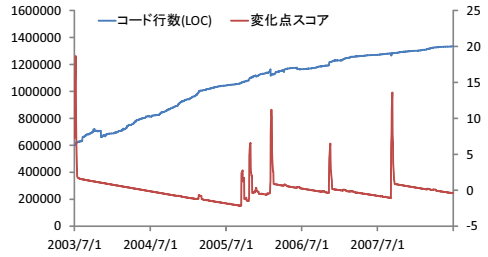


図 7. コード行数に対する変化点検出の結果

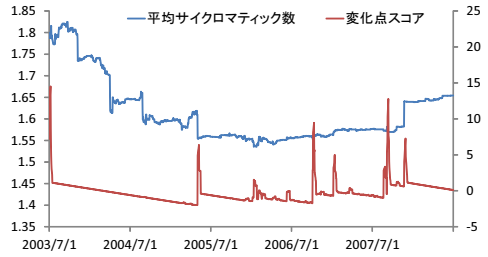


図 8. サイクロマティック数に対する変化点検出の結果

ド行数，サイクロマティック数の平均値を表し，右縦軸は変化点スコアを表している．

コード行数では，変化点スコアが 5 以上であったものは，2005 年 10 月，2006 年 11 月，2006 年 2 月，2007 年 9 月の計 4 箇所であった．コード行数が急増あるいは急減したことを示している．

サイクロマティック数では，2005 年 5 月，2006 年 10 月，2007 年 9 月，2007 年 11 月の計 4 箇所で 5 以上の変化点スコアが計測されており，サイクロマチック数の大きな増加あるいは減少があったことが見て取れる．

4.3.2　不具合メトリクスでの変化点検出結果

不具合管理システムから抽出したメトリクスである不具合報告数，不具合解決数，修正待ち不具合数に対して変化点検出を行った結果を図 9，図 10，図 11 に示す．横軸は 1 日単位の時系列を表し，左縦軸はそれぞれ累計不具合報告数，累計不具合解決数，修正待ち不具合数を表し，右縦軸は変化点スコアを表している．

不具合報告数に関しては，5 以上の高い変化点スコアは見ることができなかった．

不具合解決数と修正待ち不具合数では変化点スコアが

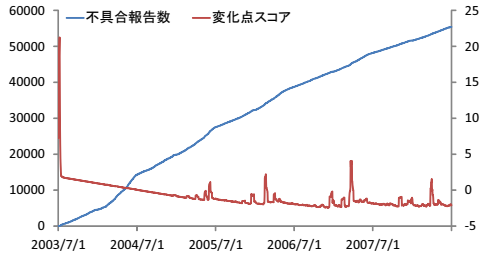


図 9. 不具合報告数に対する変化点検出の結果

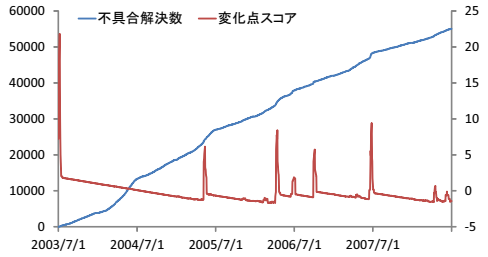


図 10. 不具合解決数に対する変化点検出の結果

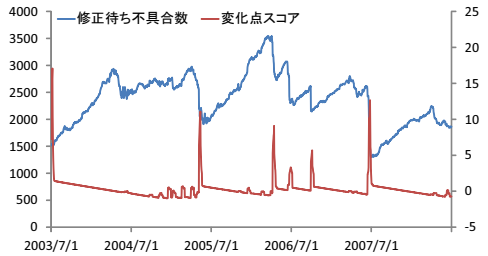


図 11. 修正待ち不具合数に対する変化点検出の結果

5 以上の時点は共通しており，2005 年 5 月，2006 年 4 月，2006 年 10 月，2007 年 6 月 2 の計 4 箇所でそれぞれ計測された．共通する 4 時点は，不具合解決数の急増と修正待ち不具合数の急減に対応しており，妥当な結果といえる．

4.3.3　トピック分析結果

設定した期間のメーリングリストと不具合報告のコメントに対してトピック分析を行った結果，それぞれ表 1，表 2 のように 10 種類のトピックが分類された．各トピックに含まれる単語からトピック（議論の内容）を推測するために利用できる．

表 1. メールのトピック分類

トピック番号	トピックに含まれる単語
M1	table tree swt event public column object viewer item tom selection text string int jface return tableviewer void method
M2	eclipse wrote problem code don application cvs swt java work ve windows time user project make find rcp bug
M3	view menu eclipse ui action org extension editor id add perspective views plugin class actions page point rcp shellbtn
M4	eclipse plugin file project jar build plugins xml plug ant org application wrote java run rcp directory class files
M5	java org eclipse internal core ui main osgi run framework runtime lang swt workbench invoke reflect sun launcher method
M6	backslash nbsp font br face size courier serif sans div wb ah wc ep wd mj ad yt ai
M7	eclipse org http news wrote message www bugs html swt bug platform cgi index browser show id https wiki
M8	editor file class method code wrote null line thread public string eclipse return object text resource plugin run problem
M9	swt shell display composite org eclipse import public text image button widgets void layout griddata label parent settext int
M10	backslash eclipse org lib jar java plugins ui usr dll core bundle message swt system subentry missing required home

表 2. 不具合コメントのトピック分類

トピック番号	トピックに含まれる単語
B1	lib eclipse usr jar plugins xp java home dll org system opt rw aab jre gtk rwxp javac bin
B2	eclipse org ui core plugins file jar update platform xml http java plugin jdt id plug osgi version feature
B3	swt shell display line int public string event void import class org eclipse null table os text item return
B4	file project cvs workspace eclipse build files problem error run ant log user set case projects dialog view laun
B5	eclipse swt problem gtk java version windows linux running build browser time vm run comment work code system os
B6	java eclipse org internal core ui swt run widgets main lang workbench jface runtime method display invoke object reflect
B7	bug duplicate marked fixed fix verified build problem head released bugs patch comment verify issue closing marking rc report
B8	api code comment change don method make case class work content add support point ui implementation part extension type
B9	created details attachment patch test screenshot fix tests file log screen attached updated problem added applied error shot image
B10	view editor text menu dialog window open problem key button selection tab windows perspective ctrl show don click set

5. 考察

本稿では紙面の都合上，サイクロマティック数の推移（図 8）に対して大きな増減を捉えた 4 時点の変化点の内 2 時点（2005 年 5 月と 2007 年 11 月），および，不具合解決数と修正待ち不具合数の推移（図 10 と図 11）対して大きな増減を捉えた 4 時点の変化点の内 2 時点（2005 年 5 月と 2007 年 6 月）のみに着目する．それぞれの変化点の前後のトピックの変化を確認することにより，大きな変化点の原因となった要因を理解できるかどうか，すなわち，提案手法がプロジェクト内に発生した何らかの変化の原因を理解するために有用かどうかを議論する．

5.1 コードメトリクスに対する変化点検出

サイクロマティック数の減少に対して大きな変化点が計測された 2005 年 5 月における変化点の前後 1 週間のメールトピックおよび不具合コメントトピックの確率分布を比較した結果を，図 12 および図 13 に示す．同様に，サイクロマティック数の増加に対して大きな変化点が計測された 2007 年 11 月における変化点の前後 1 週間のメールトピックおよび不具合コメントトピックの確率分布を比較した結果を，図 14 および図 15 に示す．

2005 年 5 月におけるメールトピックでは，トピック M2 が増加している．一方，不具合コメントトピックでは，トピック B7 が増加していることが見て取れる．メールトピック M2 には，“eclipse”, “wrote”, “problem”, “code”, “bug” などの単語が含まれており，不具合に関する議論が行われていると推察できる．また，不具合コメントトピック B7 には，“bug”, “duplicate”, “marked”, “fixed”, “fix” などの単語が含まれており，リリースに向けた活動に関する議論がなされていたものと思われる．これらのことから，サイクロマティック数の大幅な減少に対する理由としては，以下のようなことが考えられる．2005 年 5 月に大規模なリファクタリングが行われた結果，不具合が増加したと考えられる．そして，不具合が増加したため，リリースに向けた修正活動が活発になされていたものと推測される．

一方，2007 年 11 月におけるメールトピックは，トピック M3 が増加している．不具合コメントトピックでは，トピック B5 が減少していることが見て取れる．メールトピック M3 には，“jar”, “view”, “menu”, “ui”, “action” といった単語が含まれており，UI に関する議論が行われていると推察できる．また，不具合コメントトピック B5 には，“problem”, “version”, “windows”, “linux”,

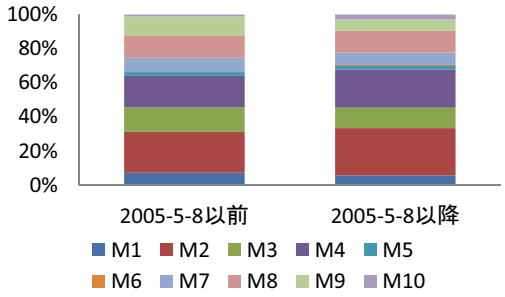


図 12. 2005-5-8 の前後 1 週間のメールトピックの確率分布

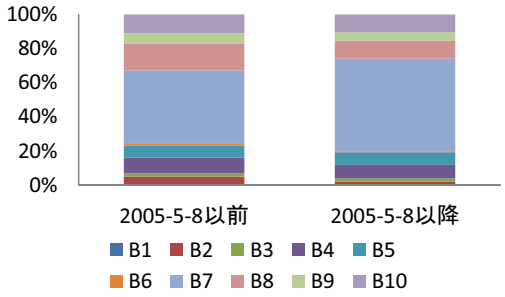


図 13. 2005-5-8 の前後 1 週間の不具合コメントトピックの確率分布

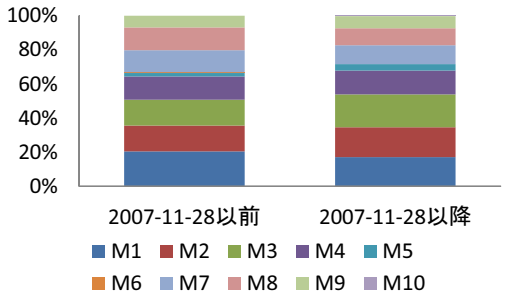


図 14. 2007-11-28 の前後 1 週間のメールトピックの確率分布

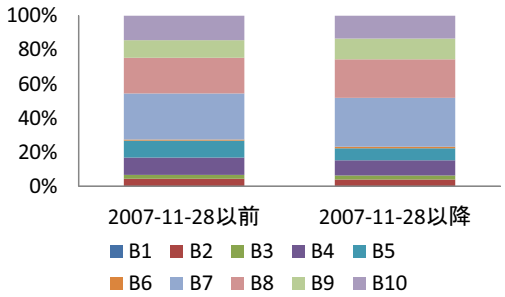


図 15. 2007-11-28 の前後 1 週間の不具合コメントトピックの確率分布

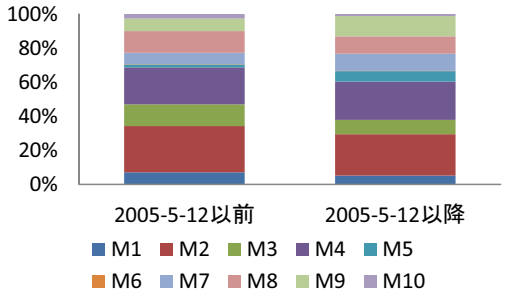


図 16. 2005-5-12 の前後 1 週間のメールトピックの確率分布

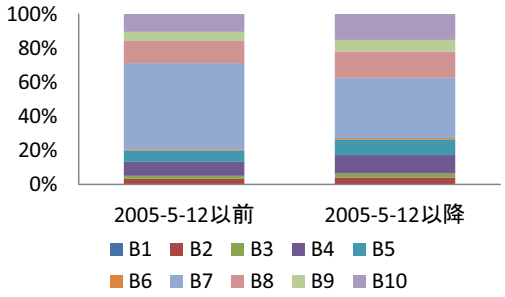


図 17. 2005-5-12 の前後 1 週間の不具合コメントトピックの確率分布

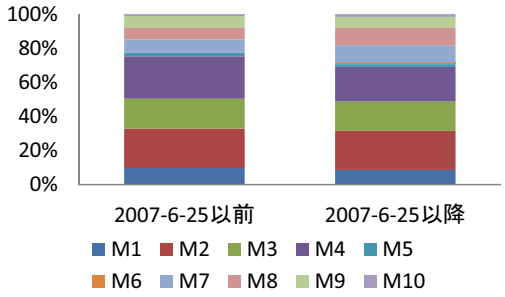


図 18. 2007-6-25 の前後 1 週間のメールトピックの確率分布

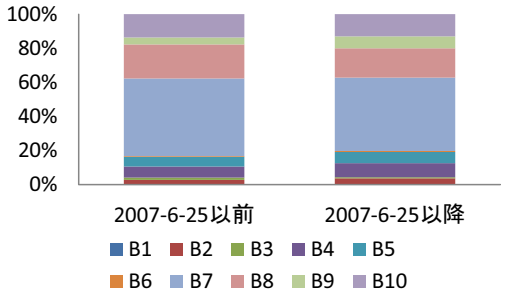


図 19. 2007-6-25 の前後 1 週間の不具合コメントトピックの確率分布

“running” などの単語が含まれており，テスト活動に関する議論がなされていたものと思われる．これらのことから，サイクロマティック数の大幅な増加に対する理由としては，以下のようなことが考えられる．2007 年 11 月に UI に関する大きな修正が必要であったとテスト活動によって判明した思われる．その結果，サイクロマティック数が増加したと推測される．

5.2 不具合メトリクスに対する変化点検出

不具合解決数の増加と修正待ち不具合数の減少に対して大きな変化点が計測された 2005 年 5 月における変化点の前後 1 週間のメールトピックおよび不具合コメントトピックの確率分布を比較した結果を，図 16 および図 17 に示す．同様に，不具合解決数の増加と修正待ち不具合数の減少に対して大きな変化点が計測された 2007 年 6 月における変化点の前後 1 週間のメールトピックおよび不具合コメントトピックの確率分布を比較した結果を，図 18 および図 19 に示す．

2005 年 5 月におけるメールトピックでは，トピック M3 が減少している．一方，不具合コメントトピックでは，トピック B7 が減少していることが見て取れる．メールトピック M3 には，“jar”, “view”, “menu”, “ui”, “action” といった単語が含まれており，UI に関する議論が行われていると推察できる．また，トピック B7 には，“bug”, “duplicate”, “marked”, “fixed”, “fix” などの単語が含まれており，リリースに向けた活動に関する議論がされていると思われる．このことから，修正待ち不具合数の大幅な減少，不具合解決数の大幅な増加に対する理由としては，以下のようなことが考えられる．UI に関する不具合を解決する方法が見つかった結果，多数の不具合修正が行われたと推測される．

一方，2007 年 6 月におけるメールトピックでは，トピック M4 が減少している．不具合コメントトピックでは，トピック B7 が減少していることが見て取れる．メールトピック M4 には “jar”, “build”, “plugins”, “plug”, “ant” といった単語が含まれており，プラグインに関する議論が行われていると推察できる．また，不具合コメントトピック B7 には，“bug”, “duplicate”, “marked”, “fixed”, “fix” などの単語が含まれており，リリースに向けた活動に関する議論がされていると思われる．このことから，修正待ち不具合数の大幅な減少，不具合解決数の大幅な増加に対する理由としては，以下のようなことが考えられる．プラグインに関する不具合を解決する方法が見つかった結果，多数の不具合修正が行われたと推測される．

5.3 本論文の制約

本論文では変化点のスコアが高くなっている要因を特定するために LDA を用いて前後 1 週間のトピック分析を行った．トピックモデルの LDA を適用して得られた結果として，不具合コメントとメールで 10 個づつのトピックがある．それぞれのトピックの議論の内容をトピックに含まれる単語から推測していた．そのため，今後は分類されたテキスト（メールや不具合コメント）も含めて確認し，総合的に議論の内容を推測する必要があると考えられる．

本論文で用いた変化点検出には事前に設定するパラメータ，AR モデルの次数，平滑化のウィンドウサイズ，忘却パラメータの 3 つがある．AR モデルの次数については参考文献で使われていた 4 次に設定した [6, pp.57-58]．平滑化ウインドウサイズは小さすぎると移動平均の値の動きが大きすぎるため毎日変化点が検出されることになるので，平滑化ウィンドウサイズについては 1 週間つまり 7 日毎の移動平均を取るために 7 に設定した．忘却パラメータについては，0.01 に設定した．これ以上小さくすると変化点を計測することができなくなるため，変化点検出可能な忘却パラメータのうち最も小さい値を設定した．

6. まとめ

本論文では，プロジェクトにおけるソフトウェア開発状況の異変を機械的に早い段階で検知することを目的としたトピック分析を併用した変化点検出手法を提案した．提案手法は，変化点検出アルゴリズムをソフトウェアメトリクスの時系列データに適用し変化点を検出する．検出された時点のメールや不具合コメントにトピックモデルの LDA を用いてトピック分析を行うことで，変化点が検出された要因を特定する．

提案手法の有用性を確かめるために，オープンソースプロジェクトの Eclipse Platform プロジェクトを対象とするケーススタディを行った．ケーススタディではソースコード，不具合管理システムの 2 つのデータソースから取得できるメトリクス（コード行数，サイクロマティッ

ク数の平均値，不具合報告数，不具合解決数，修正待ち不具合数）の時系列データを用いて変化点検出を行った．その結果，不具合報告数の時系列データ以外で高い変化点スコアの持つ変化点を検出することができた．また，不具合管理システムから取得できる不具合コメントと開発者メーリングリストから取得できるメールに対してトピック分析を行った．そして，変化点が検出された前後1週間の不具合コメントとメールトピックの割合を比較した．

ケーススタディを行った結果，ソースコードから取得したメトリクスを使った変化点検出と不具合管理システムから取得したメトリクスを使った変化点検出において変化点を検出することができた．そして，変化点の前後1週間のメールトピックと不具合コメントトピックの比較をすることによって，変化点の前後で，開発者間で議論がシフトしている様子を確認することができた．したがって，管理者が開発活動の状況を把握することに役立つと言える．

今後は複数のプロジェクトに対して同じ分析手法を適用し，より高い一般性のある結果を得る必要がある．変化点検出に用いたメトリクスについてもソースコードと不具合管理システムから取得したメトリクスだけでなく，別の視点でのメトリクスを対象とすることで，プロジェクトにおける開発状況の異変を別の観点から検知できると考えている．

謝辞

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

参考文献

[1] Hackystat. https://code.google.com/p/hackystat.

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research (JMLR2003)*, Vol. 3, pp. 993–1022, 5 2003.

[3] Lars Heinemann, Benjamin Hummel, and Daniela Steidl. Teamscale: Software quality control in real-time. In *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE2014)*, pp. 592–595, 7 2014.

[4] Masao Ohira, Reishi Yokomori, Makoto Sakai, Ken-ichi Matsumoto, Katsuro Inoue, and Koji Torii. Empirical project monitor: a tool for mining multiple project data. In *Proceedings of the 1st International Workshop on Mining Software Repositories (MSR2004)*, pp. 42–46, 5 2004.

[5] Pete Rotella and Sunita Chulani. Implementing quality metrics and goals at the corporate level. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR2011)*, pp. 113–122, 5 2011.

[6] Jun-ichi Takeuchi and Kenji Yamanishi. A unifying framework for detecting outliers and change points from time series. *Transactions on Knowledge and Data Engineering (TKDE2006)*, Vol. 18, No. 4, pp. 482–492, 4 2006.

[7] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. Modeling the evolution of topics in source code histories. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR2011)*, pp. 173–182, 5 2011.

[8] Shamima Yeasmin, Chanchal K. Roy, and Kevin A. Schneider. Interactive visualization of bug reports using topic evolution and extractive summaries. In *Proceedings of International Conference on Software Maintenance and Evolution (ICSME2014)*, pp. 421–425, 9 2014.

[9] 大平雅雄, 横森励士, 阪井誠, 岩村聡, 小野英治, 新海平, 横川智教. ソフトウェア開発プロジェクトのリアルタイム管理を目的とした支援システム. 電子情報通信学会論文誌 D-I, Vol. J88-D-I, No. 2, pp. 228–239, 2 2005.

[10] 山西健司. データマイニングによる異常検知. 共立出版, 東京, 2005.

[11] 阿萬裕久, 野中誠, 水野修. ソフトウェアメトリクスとデータ分析の基礎. コンピュータソフトウェア, Vol. 28, No. 3, pp. 12–28, 8 2011.

スパムフィルタに基づく即時バグ予測ツールの試作

森 啓太
京都工芸繊維大学 大学院工芸科学研究科 情報工学専攻
k-mori@se.is.kit.ac.jp

水野 修
京都工芸繊維大学 大学院工芸科学研究科 情報工学部門
o-mizuno@kit.ac.jp

要旨

ソフトウェア工学において，*Fault-prone* モジュールの予測モデルの研究は数多く行われてきた．これらの研究の最終的な課題は，*Fault-prone* モジュール予測を実際のソフトウェア開発の現場に適用していくことである．

そこで本稿では，その中でも構築が簡単で高い予測性能をもつ「*Fault-prone* フィルタリング」という予測モデルを利用して，*Fault-prone* モジュール予測ツールの試作を行う．本試作ツールでは，ソフトウェアリポジトリを監視して新しい変更があるごとにモジュールがバグを含む確率を予測し，結果をツール使用者に *Web* ページ上で提示する．変更ごとに予測するため，バグを含むファイルを変更した開発者を特定でき，さらにその時のコード片を特定することで，ソフトウェアの品質確保活動にかかる工数の削減とソフトウェア自身の品質の向上が期待される．

また，ツールの性能評価のため，*3* つのソフトウェアリポジトリを対象に評価実験を行ったところ，ある程度の予測性能を伴って予測結果をツールの利用者に *Web* ページで提示することが可能であるとわかった．

1. はじめに

ソフトウェア開発において，品質確保活動であるテストやレビューといったソフトウェアのメンテナンスは重要であり，かつ工数をかけすぎず効率的に行うことが期待されている．そのため，現在のソフトウェア開発ではバージョン管理システム，バグトラッキングシステム，さらに工数見積りツールなどといった様々な開発支援のためのツールが開発され利用されている．

上記で述べた開発支援方法以外に，品質確保活動の工数を削減する手段として Fault-prone モジュール (バグを含むかもしれないモジュール) を予測する手法の研究が数多く行われている [1, 2, 3, 4]．この手法は，バグの含む可能性が高いと予測されたモジュールから重点的に処理できること，さらに早期にバグを発見し除去できることから，品質確保活動の工数を削減する効果や，ソフトウェア自身の品質向上への効果が期待されている．また Kamei らは，モジュールの粒度を変更レベルとした予測モデル [5] を提案し，テストすべき変更を順序付けることにより工数を削減することができたと述べている．さらに Kamei らの予測モデルでは，変更レベルで予測をするため，バグの早期発見やバグの混入者の特定も可能であった．そして，こういった予測モデルを実際に開発支援のためのツールとして現場に適用させていくことが Fault-prone モジュール予測の研究の最終的な課題となっている．

そこで本研究では，構築が簡単で高い予測性能をもつ「Fault-prone フィルタリング」[6] という予測モデルを利用して Fault-prone モジュール予測ツールの試作を行う．この予測モデルは，その他の予測モデルが構築に様々なメトリクスの収集環境を必要とするのに対し，ソースコードのみで構築が可能である．また，試作するバグ予測ツールでは，Kamei らの提案した変更レベルの予測の利点に注目し，ソフトウェアリポジトリからコミットごとに即時的にバグを予測する．本ツールの性能の評価実験は，学習で用いた Git リポジトリの履歴を参照することで，バグが含まれるかどうかがわかっているモジュールと変更情報を関連付け，それらを新しい変更があったと想定してツールに与えることで行う．

2. 目的

本研究の目的は「Fault-prone フィルタリング」という予測モデルを用いて，Git リポジトリから変更ごとに即時的にバグを予測するツールを試作し，「Fault-prone フィルタリング」を用いたバグ予測ツールの有効性を考察することである．

この目的のため，以下の研究設問を設定し，ツールの有効性について考察する．本研究における研究設問は次の通りである．

RQ1: Fault-prone フィルタリングによる即時バグ予測ツールは構築できるか．

RQ2: 提案する即時バグ予測ツールによるバグ予測では，どの程度の精度が得られるのか．

RQ3: 提案する即時バグ予測ツールは開発に有用な情報を提供できるか．

3. 試作の準備

3.1. SZZ アルゴリズム

SZZ アルゴリズム [7] とは，バグトラッキングシステムのバグデータベースとバージョン管理システムのバージョンアーカイブを相互に結びつけて解析することでバグを混入したモジュールを特定するアルゴリズムである．

本研究では，研究室において開発・保守されているツール (szz_tools) を利用した．このツールは，バージョンアーカイブとして Git リポジトリ，バグデータベースとしてバグ情報の記述された csv ファイルを用いて SZZ アルゴリズムを適用し，バグが含まれるモジュールとバグの含まれないモジュールを特定してその情報をリポジトリにタグとして添付するものである．このツールでのモジュールの粒度は一つのファイルである．

3.2. Fault-prone フィルタリング

Fault-prone フィルタリングとは，スパムメールの判別を行うスパムフィルタ (CRM114) の技術をソフトウェアのソースコードに適用し，コードのテキスト情報のみから Fault-prone モジュールの予測を行うものである．スパムフィルタは，スパムメールには特定の単語群や頻繁に含まれているという事実に基づき，過去に受信したメール内の単語群を学習してスパムメールかハムメールかを分類するための識別辞書を作成する．そして新しいメールについて，識別辞書とベイズ識別を利用してスパムメールかどうかの分類を行う．これをソフトウェアに関しても，バグが存在するところには特定のコード片が存在するという類推のもと適用する．テキスト情報のみで予測を行うため，複雑なソフトウェアメトリクスを使用した従来の予測モデルよりも容易にプロジェクトに適用することができる．

3.3. 欠陥の変更時の即時予測

従来の Fault-prone モジュールの予測モデルはファイルやパッケージレベルでの予測のものがほとんどであった．それらの欠点として以下のことが挙げられる．

- 予測の粒度が粗い
- 予測が出たとしても，バグを混入した開発者が特定できない
- 予測が遅すぎて，開発サイクルと咬み合わない

これらの欠点のため，変更 (git などにおけるコミット) レベルでの即時予測を行える予測モデルが注目されている．それらの利点として以下のことが上げられる．

- 予測の粒度が細かい
- バグを混入した開発者は，変更を行った開発者であると簡単に特定できる
- 予測が早いので，開発サイクルへのフィードバックが容易

「予測の粒度が細かい」というのは，変更はファイルの差分の集合であると考えられるからである．本研究では，この変更レベルでの即時予測の利点に注目する．

しかし，本研究で使用する Fault-prone フィルタリング自体の予測粒度は第 3.1 節で述べたツールの粒度から，ファイルレベルである．そこで本研究では新しい変更ごとにその変更で修正されたファイルに対して Fault-prone フィルタリングを実行することで変更レベルでの予測とする．これを実装した本ツールに期待される利点は次の通りである．

- 新しい変更に対して即時的にファイルのバグの確率を提示できる．

- コミット情報を保持しているのでバグの確率を上げた開発者及び日付などを特定できる．
- 確率はファイルごとに予測されるが，Git を用いてバグの確率が変動した前後で差分を取ることにより，どういったコード片がバグの確率を上げているのかがわかる．

4. 即時バグ予測ツールの試作実験

本章では，本試作ツールが特定の Git リポジトリを継続的に監視して即時的にバグを予測していく際のツール内部の動作ついて説明する．また，ツールとしてのユーザインタフェース部分である Web ページ部分の表示に関する内部動作，操作方法ついてもこの章で述べる．最後に，予測性能の評価手法について説明する．

4.1. 予測の準備

4.1.1 リポジトリとバグ情報の取得

本実験ではツールでの予測対象として次の 3 つの Java で開発されているオープンソースプロジェクトのリポジトリとそのバグ情報を取得した．

- Apache OpenJPA
- Apache James
- Eclipse Birt

4.1.2 バグ情報との統合

本実験ではリポジトリをバグを予測するための学習のデータセットとして用いるために，これらのリポジトリに対して研究室で開発・保守されている SZZ アルゴリズムを用いたツール (szz_tools) を適用してリポジトリにバグ情報を与える．このツールを用いると，Git の tag コマンドを利用してバグ除去時点のコミットには FIX タグを，バグの混入時点のコミットには BUG タグを添付することができ，さらにバグが含まれるモジュール (ファイル) を特定し，FAULT タグを添付した後，それ以外に INNOCENT タグを添付することができる．

4.2. 予測の手順

本実験のツールでは，前節で用意した Git リポジトリへの新しい変更に対して継続的に予測を行っていく．予測の手順は，以下の項の通りである．

4.2.1 学習

新しい変更に対して予測を行うためには，Fault-prone フィルタリングによる初期学習が必要である．前節で用意したバグ情報を統合した各 Git リポジトリに対して次の手順で学習を実行し，識別辞書を作成する．

1. FAULT タグと INNOCENT タグをリポジトリから取得する．
2. これらのタグを用いて，リポジトリからモジュールのソースコードを読み出し，FAULT であるか INNOCENT であるかの情報と共に記録していく．
3. それらすべてをトレーニングセットとしてテキストフィルタを用いて学習させ，FAULT と INNOCENT の識別辞書をそれぞれ作成する．

本実験のツールは試作であるため，スパムフィルタリングで用いられる手法である「誤判定時のみ学習」による学習の強化を実装していないので，この学習の処理は予測を開始する前の 1 回のみ行われる．

4.2.2 変更の取得

変更の取得と解析の手順は次の通りである．

1. git fetch コマンドを使用して差分を取得する．
   このコマンドにより，Git はリモートリポジトリからローカルリポジトリとの差分を取得する．このとき，フェッチしてきたコミットの中で最新のものは Git 内で FETCH_HEAD という名前で参照できる．
2. git log HEAD..FETCH_HEAD –name-status コマンドを使用して差分に対するログを取得する．
   「HEAD..FETCH_HEAD」という引数を指定をすることにより，HEAD(ローカルリポジトリの最新のコミット) から FETCH_HEAD(リモートリポジトリの最新のコミット) までの範囲を指定して Git からログを取り出す．さらに「–name-status」とい

う引数を指定することにより，各コミットで変更されたファイルのファイル名とその変更情報を加えて取得することが出来る．

3. ログを解析する．
   得られたログから各コミットごとにそのコミットの情報と，予測する対象のモジュールとして追加，変更，コピー，または修正のあったファイルの名前を記録していく．

4. git merge FETCH_HEAD コマンドを使用してローカルリポジトリを更新する．
   このコマンドにより，差分のログの解析が終わったローカルリポジトリをリモートリポジトリと同じ状態にする．つまり HEAD が FETCH_HEAD の指すものとなる．これにより次回の変更の取得に備えることができる．

4.2.3 分類

前節でログを解析して得られた各モジュールをスパムフィルタによる分類にかけ，得られた確率，ファイル名，及びコミットの情報をデータベースに格納していく．

4.3. リポジトリの監視

本実験のツールで，リポジトリの監視をしていくためには以下の操作を行う．

- 登録用プログラムによるツールへのリポジトリの登録
  前節第 4.2.1 項で述べたように，予測をする前には学習が必要である．しかしこの学習には時間が大きくかかるため，リポジトリ登録用プログラムをツールの一部として作成した．このプログラムは，学習が終わった時にツールへとリポジトリの情報の登録を行う．新しいリポジトリを監視したい時は，この登録用プログラムの引数として新しいリポジトリを指定することによってツールへ監視対象として登録ができる．

- cron によるリポジトリの監視の設定
  本実験ではリポジトリの動きを監視するために，Unix 系 OS においてコマンドの定時実行のスケジュール管理を行うために用いられる cron というプログラムを利用する．cron に前節の第 4.2.2 項から第 4.2.3 項までの処理を行うプログラムを指定し，任意の時間間隔で自動的に実行させる．これにより，定期的に変更があったかどうかを確認し，即時的な Fault-prone モジュールの予測を行うことが出来る．

4.4. Web ページへの表示

それぞれのページについての出力や操作方法について以下に説明する．

- main.cgi
  本ツールのメインページであり，図 1 で示すように予測結果の検索フォームによる検索と，cron での実行のログの確認ができる．検索フォームで入力できるものは次の通りである．
  - Repository's name (セレクトボックス)
    結果を表示したいリポジトリの名前を指定する．
  - Omit commits which have no java files (ラジオボタン)
    Java ファイルを持っていないコミットを表示するかどうかを指定する．
  - Order of commit's date (ラジオボタン)
    コミットの日付が新しい順番で表示するか古い順番で表示するかを指定する．
  - Date (日付入力欄)
    コミットの期間を指定する．
  - Commit ID (テキストボックス)
    コミットの ID(40 桁の SHA-1 ハッシュ) を指定する．
  - Author (テキストボックス)
    コミットした開発者名を指定する．
  - Commit message (テキストボックス)
    コミットメッセージを指定する．
  - File name (テキストボックス)
    ファイル名を指定する．
  - show (ボタン)
    上記の条件で検索を開始する．検索結果は show_result.cgi で表示する．

- show_result.cgi
  main.cgi の検索フォームからの入力を受け取り，そ

れらを条件としてデータベースから予測結果を検索し，Web ページにコミットごとに予測結果をまとめて図 2 のように表示する．

- about.html
  本ツールの使い方について記述されているページ．
- setting.cgi
  監視しているリポジトリに対して，データの削除などの設定を行う．
- setting_apply.cgi
  setting.cgi からの入力を受け取り，実際にデータベースやデータファイルの内容を変更し，設定結果の旨を表示する．

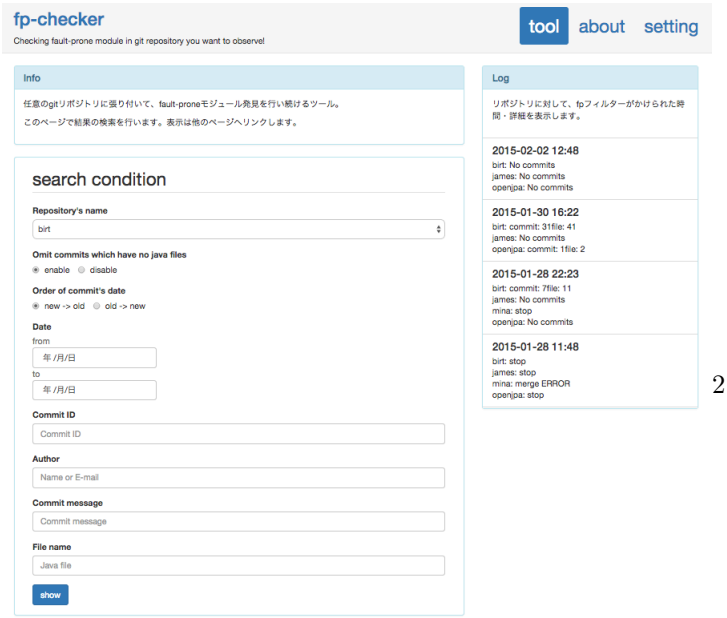


図 1. ツールのメインページ

4.5. 予測性能の評価実験

評価実験の概要

最後に，本研究において本ツールの性能・信頼性の評価の実験は，研究設問に回答し，ツールの有効性を考察するために必要不可欠なものである．

この実験では，実際のツールの使用を想定して，学習時に用いた第 4.1.2 節で FAULT，または INNOCENT タグを添付されたモジュール，つまりバグかどうか予めわかっているものを，Git リポジトリの履歴を参照し変更と関連付け，それらの変更を新しい変更があったと想定してツールに与えることで評価する．また，ただデータを集計するだけでなくツールとして Web ページ上で，どの程度予測の精度があるのかを確認できるようにする．

以下の手順で評価実験を行う．

1. テストデータの準備
   本ツールで行う予測では，テストのために利用するデータはコミットごとにそのコミットの情報と関連付けられていているモジュールでなければならない．しかし，モジュールに添付された FAULT タグと INNOCENT タグにはコミットの情報が入っていないため，リポジトリの履歴を参照し，タグを逆算することによりコミットごとのモジュールを取得する．取得するコミットは，バグを含むモジュールと含まないモジュールの数のバランスを考え，BUG タグがついているコミットを選択する．SZZ アルゴリズムの特性上，これをしなければバグを含まないモジュールの数が多くなってしまうためである．取得したコミットとモジュールを関連付けたものをテストデータとする．
2. 分類テスト
   前述で準備したテストデータを用いて，分類テストを以下の手順で行う．
   - 各モジュールに対してバグである確率を予測して，モジュールが実際にバグであるかの情報と共に予測した確率をデータベースに登録する．
   - データベースから前述で登録した情報を取り出し，閾値を 0.5 として分類して結果を集計する．

5. 予測の結果

5.1. 対象とした Git リポジトリの学習時のデータ

前章で述べたように 3 つの実験対象としたリポジトリに対して，SZZ アルゴリズムを利用したタグの添付によりモジュールがバグを含むかどうかを特定した．その結果の集計を表 1 に示す．

Result Acc: 0.7868 Pre: 0.8044 Rec: 0.8381 F1: 0.8209

author	commit	message	date
Richard G. Curtis <curtisr7@apache.org>	e48cd3e59d82ce352d161109987f283ecc369b66	OPENJPA-2508 : Account for JOIN FETCH statements when loading from the Query Cache. git-svn-id: https://svn.apache.org/repos/asf/openjpa/trunk@1600757 13f79535-47bb-0310-9956-ffa450edef68	2014-06-05 20:49:01
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Department_.java			45%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Department.java			0%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Employee.java			0%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/TestJoinFetchWithQueryDataCache.java			100%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Employee_.java			92%
openjpa-kernel/src/main/java/org/apache/openjpa/datacache/QueryCacheStoreQuery.java			100%
Richard G. Curtis <curtisr7@apache.org>	b95e0d1c6c2e2cc8f6f9d422058dfe232f6091d5	OPENJPA-2502 : Update accessPath metas in CriteriaExpressionBuilder. Merged changes from 2.2.1.x. Patch contributed by Albert Lee. git-svn-id: https://svn.apache.org/repos/asf/openjpa/trunk@1600682 13f79535-47bb-0310-9956-ffa450edef68	2014-06-05 15:48:54
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/TestJoinFetchWithQueryDataCache.java			100%
openjpa-persistence/src/main/java/org/apache/openjpa/persistence/criteria/CriteriaExpressionBuilder.java			100%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Department.java			0%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Department_.java			34%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Employee.java			0%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Employee_.java			89%

図 2. 予測結果表示ページ (対象は Apache OpenJPA)

これらのモジュールは，ツールでの即時予測と評価実験を行うための学習でデータセットとして用いた．

表 1. 各 Git リポジトリの学習時のモジュール数

	Apache OpenJPA	Apache James	Eclipse Birt
コミット数	4383	11866	29501
バグを含まないモジュール	9273	21600	41794
バグを含むモジュール	12518	3794	35398
合計	21791	25394	77192

5.2. 評価実験

5.2.1 評価指標

試作したツールの評価実験で得られた結果の評価指標として精度 (Accuracy)，再現率 (Recall)，適合率 (Precision)，F_1 値を用いる．分類結果の凡例を表 2 に示す．

表 2. 分類結果の凡例

実測 \ 予測	バグを含まない	バグを含む
バグを含まない	TN	FP
バグを含む	FN	TP

- TP (True Positive): バグと分類したもので，実際にバグであったモジュールの数．
- FP (False Positeve): バグと分類したもので，実際はバグでなかったモジュールの数．
- TN (True Negative): バグでないと分類したもので，実際にバグでなかったモジュールの数．
- FN (False Negative): バグでないと分類したもので，実際はバグであったモジュールの数．

精度 (Accuracy)

精度は，分類結果全体の中で分類成功であった割合を示す．よって以下のように定義される．

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

全体的な傾向を把握することができるがデータの隔たりなどに大きく影響を受けるので，再現率と適合率を併用する．

再現率 (Recall)

再現率は，実際にバグであったモジュールの中でバ

グであると分類したものの割合，つまり網羅率を示す． よって以下のように定義される．

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

適合率 (Precison)

適合率は，バグであると分類したモジュールの中で実際にバグであったものの割合，つまり無駄なモジュールをどれだけ調べる必要があるかというテストのコストを示す．よって以下のように定義される．

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

F_1 値

F_1 値は，再現率と適合率がトレードオフの関係にあるため，両者を総合的に判断するための指標であり，再現率と適合率の調和平均によって示される．よって以下のように定義される．

$$F_1 = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{4}$$

5.2.2 評価結果

第 4.5 節の手順の通りに結果を示す．

1. 用意したテストデータ
   用意したテストデータの集計を表 3 に示す．この表で示すデータには，本研究の変更ごとの即時バグ予測を行うという特性上から，複数のコミットが同じ状態のモジュールを指すことがあるため，テストセットとして同じモジュールがいくらか含まれている．そのため独立したモジュールの数も集計した．

表 3. 評価実験で用いたテストデータ

	Apache OpenJPA	Apache James	Eclipse Birt
コミット数	1828	1513	7098
バグを含まないモジュール	6375	9481	18732
独立したモジュール	6160	5812	18489
バグを含むモジュール	8909	4947	24253
独立したモジュール	8811	2347	24049
モジュールの合計	15284	14428	42985

2. 分類テストの結果
   各リポジトリの分類結果を表 4 から表 6 に示す．次に前節で定義した評価指標を用いた分類テストの評価を表 7 に示す．

またツールとして，Web ページでは，図 3 に示すようにファイル名の右側に実際にバグを含む ([B]) か含まないか ([N]) という情報を加えて表示することで，どの程度予測が正しいかツール上でもおおまかに確認することが出来た．また，計算した評価指標を Web ページ上でも示すことによって，ツールの予測性能の状態を把握できるようにした．

表 4. 分類結果 (Apache OpenJPA)

	予測	
実測	バグを含まない	バグを含む
バグを含まない	4559	1816
バグを含む	1442	7467

表 5. 分類結果 (Apache James)

	予測	
実測	バグを含まない	バグを含む
バグを含まない	3657	5824
バグを含む	233	4714

表 6. 分類結果 (Eclipse Birt)

	予測	
実測	バグを含まない	バグを含む
バグを含まない	6498	12234
バグを含む	2519	21734

6. 考察

6.1. 本ツールの有効性

表 4 から表 7 に示した評価実験の結果を踏まえて，第 2 章で定めた研究設問に答えることで本ツールの有効性について考察する．

RQ1: Fault-prone フィルタリングによる即時バグ予測ツールは構築できるか．

本予測モデルの構築に必要となるのはソースコードのみであるので，ツールとしての実装は容易である．この点に関しては，様々なメトリクスを要求する予測モデルを用いたツールよりも優れていると考えられる．しかし，FindBugs[8] のような静的コード解析ツールと比較すると，機械学習のコストが構築の際にかかってしまう．

Result Acc: 0.5802 Pre: 0.4473 Rec: 0.9529 F1: 0.6088

author	commit	message	date
Stephen K. Brewin <sbrewin@apache.org>	d425aebb99aa790ec245ab5824f041ec27e4c26d	JAMES-1355 Switched to using the FileSystem service to obtain the 'sieve' sub-directory of the root directory of the application git-svn-id: https://svn.apache.org/repos/asf/james/server/trunk@1222684 13f79535-47bb-0310-9956-ffa450edef68	2011-12-23 14:09:28
/mailets/src/main/java/org/apache/james/transport/mailets/ResourceLocatorImpl.java [N]			0%
/mailets/src/main/java/org/apache/james/transport/mailets/LocalDelivery.java [N]			100%
/mailets/src/main/java/org/apache/james/transport/mailets/SieveMailet.java [N]			0%
Norman Maurer <norman@apache.org>	00f75520dea733f81b8f9f470e1a4f3174711d41	Add protocols-lmtp as dependency as we factor out the lmtpserver code to protocols. See PROTOCOLS-1 git-svn-id: https://svn.apache.org/repos/asf/james/server/trunk@1176509 13f79535-47bb-0310-9956-ffa450edef68	2011-09-27 18:40:48
/lmtpserver/src/main/java/org/apache/james/lmtpserver/CoreCmdHandlerLoader.java [B]			100%
/lmtpserver/src/main/java/org/apache/james/lmtpserver/hook/MailboxDeliverToRecipientHandler.java [N]			0%
/lmtpserver/src/main/java/org/apache/james/lmtpserver/DataLineLMTPHandler.java [N]			0%
Norman Maurer <norman@apache.org>	528d39591513fbc86d5675609c1f185514ebd668	Add file based queue implementation. See JAMS-1316 git-svn-id: https://svn.apache.org/repos/asf/james/server/trunk@1171244 13f79535-47bb-0310-9956-ffa450edef68	2011-09-15 19:52:51
/queue-file/src/main/java/org/apache/james/queue/file/FileMailQueue.java [N]			100%
/queue-file/src/main/java/org/apache/james/queue/file/FileMailQueueFactory.java [N]			0%

図 3. ツール上での評価結果の表示 (対象は Apache James)

表 7. 分類テストの評価

	精度	再現率	適合率	F_1 値
Apache OpenJPA	0.7868	0.8381	0.8044	0.8209
Apache James	0.5802	0.9529	0.4473	0.6088
Eclipse Birt	0.6568	0.8961	0.6398	0.7466

RQ2: 提案する即時バグ予測ツールによるバグ予測では，どの程度の精度が得られるのか．

評価実験により，表 7 に示すような精度が得られた．この結果に対して以下のことが考察できる．

- このツールでのバグの網羅性
  得られた再現率は 8 割から 9 割であり，これは本ツールでチェックしたコミットに属するファイルのうち，実際にバグであるもののほとんどを網羅できていることを示す．
- このツールでの確率の提示による品質確保活動のコスト
  得られた適合率は，4 割から 8 割とかなりばらつきがあった．4 割であった場合，1 つのバグを含むモジュールを発見するためには，1 つか 2 つの実際はバグを含まないモジュールを調べなければならないことになる．しかし，このツールではバグの確率を変更ごとに予測することができるため，Git を用いてバグの確率が変動した前後で差分を取り，その差分のみをレビューするなどの工夫をすることで，工数を削減できる．また，予測のタイミングが早く，開発者の特定も可能なこともコードのメンテナンスにかかる工数を減らす要因となるだろう．

RQ3: 提案する即時バグ予測ツールは開発に有用な情報を提供できるか．

本ツールは，変更ごとにモジュールがバグを含む確率の情報を与えることでコストをかけるべきファイルを選択することが出来る．評価実験では，バグであると判断する閾値を 0.5 としたが，この値を変更することで品質確保活動のコストの調整がある程度可能となるだろう．例えば，本実験結果では再現率が十分に高いので，閾値を上げることでバグの判断を厳しくし，レビューするモジュールを減らすことができる．

また，開発者へ開発のフィードバックを与えること

が出来る．本ツールで実際にリポジトリを監視していたところ，図 4 のように「Refactor code.」というメッセージを持つ変更の前後でファイルに対するバグである確率の変動が見られた．この図のように確率が下がった場合は，達成度を開発者に与えられるだろう．また，このようなときに Git を用いて差分を取ればどういったコード片がバグの可能性を高めているのかを知ることが出来る．

6.2. 妥当性の検証

本研究の妥当性について以下に示す．

- オープンソースプロジェクトに対する実験
  実験で用いたのは，全てオープンソースのプロジェクトである．このため商業のプロジェクトに今回の実験と同じ手法を適用しても，同様の結果が得られるとは限らない．

- SZZ アルゴリズム
  本研究でモジュールにバグを含むかどうかのタグを添付するのに用いた SZZ アルゴリズムは，完全にバージョン管理システムとバグ情報を結び付けられるものではない．

  また，最近のコミットで変更されたファイルについては，バグ情報が不足しているのでほとんどのものに INNOCENT タグがつけられてしまう．

- 評価実験におけるテストデータの偏り
  本実験では，上述した SZZ アルゴリズムの特性から，INNOCENT タグが添付されたモジュールが多くならないように，BUG タグのついているコミットのみを実験時のツールの入力として使用した．しかし，BUG タグが付けられているコミットの期間，つまりバグ情報が存在している期間を抽出し，その期間での BUG タグがついていないコミットもテストデータとして含めた方が実験における評価の妥当性が上がる可能性がある．

  また，結果の表 3 に示すように Apahce James においては，独立したモジュールの数が全体のモジュールに対して半分ほどであったため，本実験の評価の妥当性は低い可能性がある．

6.3. 今後の課題

本研究で試作したツールにおける今後の課題は以下の通りである．

- 継続的な学習の実装
  本ツールを実際の現場へと適用するためには，予測した結果が正しかったかどうかの情報を後から入力することにより，誤判定ならば学習するという「誤判定時のみ学習」による状況に合わせた学習の強化は必須である．また，そのたびに Web ページ上で提示する精度などの評価指標も更新すると良いと思われる．

- 予測結果の Web ページへの表示速度の改善
  予測結果の絞込をする際，データベースの検索に時間が掛かる場合があった．このためデータベースのスキーマやプログラムを改善する必要がある．

7. 結言

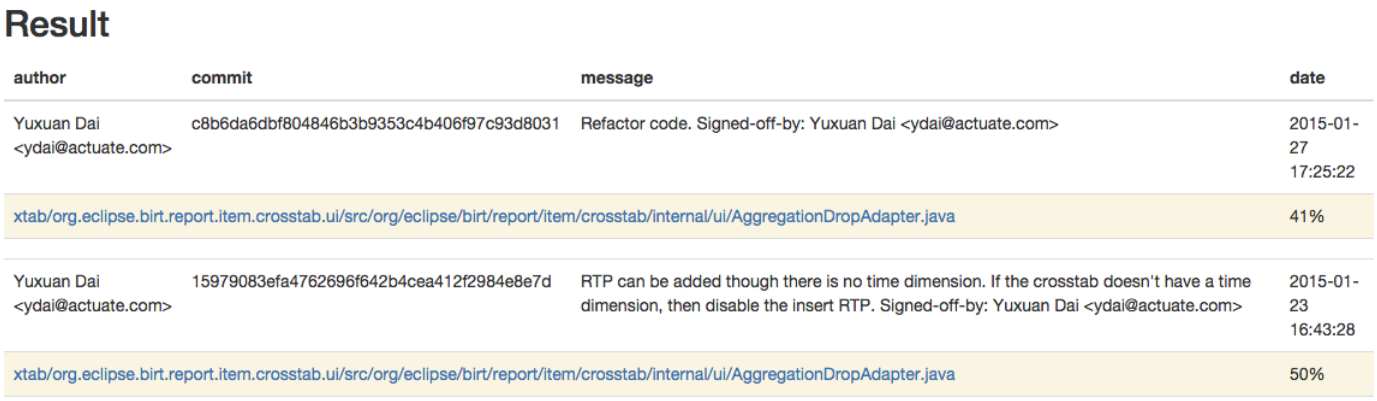
本稿では,「Fault-prone フィルタリング」という予測モデルを利用して Fault-prone モジュール予測ツールの試作を行い，このツールの理論，内部動作，操作方法，評価手法およびその結果，考察を述べてきた．3 つのソフトウェアリポジトリを対象にした評価実験の結果をもとに考察を行った結果,「Fault-prone フィルタリング」を用いたバグ予測ツールは構築可能であり，本ツールはある程度の予測性能を伴って予測結果をツールの利用者に Web ページで提示することが可能であるとわかった．

参考文献

[1] 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, vol.29, no.1, pp.106-117, Feb. 2012.

[2] P. Bellini, I. Bruno, P. Nesi, and D. Rogai, “Comparing Fault-Proneness Estimation Models“, *Proc. of 10th IEEE International Conference on Engineering of Complex Computer Systems*, pp.205-214, 2005.

[3] Tim Menzies, Jeremy Greenwald, and Art Frank, “Data Mining Static Code Attributes to Learn Defect Predictors“, *IEEE Trans. on Software Engineering*, vol.33, no.1, pp.2-13, Jan. 2007.



Result

author	commit	message	date
Yuxuan Dai <ydai@actuate.com>	c8b6da6dbf804846b3b9353c4b406f97c93d8031	Refactor code. Signed-off-by: Yuxuan Dai <ydai@actuate.com>	2015-01-27 17:25:22
xtab/org.eclipse.birt.report.item.crosstab.ui/src/org/eclipse/birt/report/item/crosstab/internal/ui/AggregationDropAdapter.java			41%
Yuxuan Dai <ydai@actuate.com>	15979083efa4762696f642b4cea412f2984e8e7d	RTP can be added though there is no time dimension. If the crosstab doesn't have a time dimension, then disable the insert RTP. Signed-off-by: Yuxuan Dai <ydai@actuate.com>	2015-01-23 16:43:28
xtab/org.eclipse.birt.report.item.crosstab.ui/src/org/eclipse/birt/report/item/crosstab/internal/ui/AggregationDropAdapter.java			50%

図 4. バグである確率の変動 (対象は Eclipse Birt)

[4] Naeem Seliya, Taghi M. Khoshgoftaar, and Shi Zhong, “Analyzing Software Quality with Limited Fault-Proneness Defect Data“, *Proc. of 9th IEEE International Symposium on High-Assurance Systems Engineering*, pp.89-98, 2005.

[5] Yasutaka Kamei, Emad Shihab and Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi, “A Large-Scale Empirical Study of Just-in-Time Quality Assurance“, *IEEE Trans. Softw. Eng*, vol.39, no.6, pp.757-770, June 2013.

[6] O. Mizuno and T. Kikuno, “Training on Errors Experiment to Detect Fault-Prone Software Modules by Spam Filter”, *The 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE2007)*, pp.405-414, 2007.

[7] J. Śliwerski, T. Zimmermann, and A. Zeller, “When do changes induce fixes?”, *Proceedings of the 2005 international workshop on Mining software repositories*, pp.1-5, 2005.

[8] (2015, Mar.) Findbugs™- find bugs in java programs. [Online]. Available: `http://findbugs.sourceforge.net/`

Concolic Testingを活用した実装ベースの回帰テスト ～人手によるテストケース設計の全廃～

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

増田聡
日本IBM
SMASUDA@jp.ibm.com

湯本剛
日本HP
tsuyoshi.yumoto@hp.com

植月啓次
フェリカネットワークス
Keiji.Uetsuki @ FeliCaNetworks.co.jp

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

要旨

ソフトウェアテストにおける回帰テストは，主に以前のリリースとの互換性を確認するために行われる．当該ソフトウェアが良く使われ利用価値が高まるほど，頻繁に行われる傾向がある．テストを行う側は，派生開発の頻度と規模が増えるほど回帰テストの困難さが増加し，それに比例してコストと期間が増加する問題を抱えている．

この問題の原因を既存の回帰テストが仕様ベースのテストを基にしていることにあると考え，ツールを用いた実装ベースのテスト方法を提案する．具体的な実現方法として *Concolic Testing* を利用し，回帰テストにおいて人手によるテストケース設計を全廃する実験例を示す．

1. はじめに

ソフトウェア開発コストの中で，テストに係るコストが大きな割合を占めている．その中でも，既存のソフトウェアに変更を加えた時，意図しない影響が無いことを確認する回帰テストは，我が国のソフトウェア開発の多くが派生開発であることから，テストの中で大きな割合を占めている [1].

回帰テストとは，テストの団体 International Software Testing Qualifications (ISTQB) による定義では「変更により，ソフトウェアの未変更部分に欠陥が新たに入り込んだり，発現したりしないことを確認するため，変更実施後，すでにテスト済みのプログラムに対して実行するテスト」とされている [2]．回帰テストは，変更行為に対して，想定される機能の変化を確認する「変更に伴う確認テスト」とは区別されている．

回帰テストの問題は，テスト対象範囲が未変更の部分であり，変更量とは無関係に広いため，テストの漏れを少なくするには膨大な量のテストが必要となり，人手によるテストでは対応できない．しかし，自動化の現実的な方法が知られていない課題である．自動化が難しい問題は，テストで発見すべき対象が，新たな欠陥が入っていないこと，潜在する欠陥が発現しないことであり，テスト結果の判定条件が曖昧なことにある．

我々の研究は，回帰テストの目的を未変更部分の互換性を確認するテストと捕え，その自動化を進めることにより，回帰テストの問題を解決する．提案する方式は，実装ベースのテストの考え方を用いて，人手によるテストケース作成を不要とし，それまでのテストケース蓄積にも依存しない，回帰テスト自動化の方式を提案する．この方式は，近年，研究が進んでいる静的実行 (Symbolic execution) や Concolic testing におけるツールを活用することにより実現している [3, 4, 5].

本稿では，回帰テストをプログラムの二つの世代（Rm と Rm+1) における互換性の確認と考える．互換性の確認とは，プログラムの挙動を入力と出力の関係で捕え，同じ入力に対して同じ結果を出力することを確認する．変更とは，バグの修正や機能の追加・変更・削除などを

意味し，その変更の想定される影響を確認するテストは，回帰テストとは別のテスト（変更に伴う確認テスト）とし，回帰テストには含めない．また，Rm に潜在する欠陥があったとして，回帰テストは，その潜在する欠陥を見つけるテストではなく，発現しないことを確認する．発現しないとは，二つの世代において同じ入力に対し同じ値を返すことと考える．

一般的に行われているテストは，仕様を基にプログラムの正しい動作に対応する，テスト入力値と期待結果のセットを人手によって求める．このセットをテストケースと呼んでいる [2]．回帰テストの自動化は，テスト入力値と対応する期待結果を人手に頼らずに得ることである．ここでは，プログラムが持つ制御パスを網羅することができる入力のセットを自動的に見つける手段と，そのテスト入力値に対応する版 Rm における出力を自動的に取得し，回帰テストの期待結果として用いる手段によって，回帰テストの自動化を行う．

本論文 2 章で仕様ベースの回帰テストが持つ原理的な問題を明らかにし，提案する方式と比較する．本論文 3 章で実装ベースの回帰テストを実現する方法について提案し，本論文 4 章で実験的に実装ベースの回帰テストを評価する．

2. 回帰テストの背景技術と課題

現在の主流である仕様ベースの回帰テストの問題について考える．次に，理想的なモデルベース開発によって解決できる部分を示し，提案する実装ベースによる方式を示す．

2.1. テストに関する定義と問題

回帰テストの課題について考える前に，ソフトウェアテストにおける定義と問題について示す．

テストの基本課題は，漏れの無い網羅性の高いテスト入力値の生成と，その入力に対する正しい期待結果の生成である．先ず，テスト入力値の網羅性について考える．

テストの性能の一つは，テストの網羅性である．回帰テストの場合，未変更部分に対する網羅が求められている．未変更部分とは，変更部分の補集合であるが，何が全体なのかを定義する必要がある．ここでは，全体を回帰テスト対象のプログラム（ソースコード）とする．

回帰テストの網羅性を対象とするプログラムに対するテストの網羅性と定義し，プログラムが持つ制御構造の網羅尺度を使う [2]．プログラムの網羅尺度研究は，ほぼ完成しており，網羅基準の順位についても定義がある．データフローを基準にするものと，制御フローを基準にするものがあり，制御フローの順位は，もっとも低い基準が実行文網羅 (SC: Statement coverage）や条件網羅 (BC: Branch condition coverage) であり，最も高い基準は分岐条件組合せ網羅（BCC: Branch condition combination coverage）であり，次が MC/DC(Modified condition/decision coverage) とされている [6, 7]．ここでは，テストで設定可能な変数群に着目した BCC を可達パス網羅と定義し回帰テストにおける最も高い網羅基準と考える．設定可能とは，プログラムでは関知出来ないマシン割込みなどの変数を除くことを意味する．

テストの性能を決めるもう一つは，期待結果の正しさである．膨大な量のテスト入力に対して期待結果を求める問題は，オラクル問題（Oracle Problem）と呼ばれ，テスト入力値の網羅性と共にテストの大きな課題である．関数レベルの単体テストなど人手により仕様から期待結果を求めることが容易な場合では問題にならないが，複雑なプログラムやシステムに対して，変数間の組合せを含む自動テストを行う場合に問題となる [8]．

テストの問題をまとめると，

1). テスト入力値の生成問題 : 網羅性の高いテスト入力値のセットを合理的に生成する方法を見つける．

2). 期待結果の生成問題 : オラクル問題と呼ばれ，変数間の組合せを含む膨大な期待結果を正しく生成する方法を見つける．

この二つの問題が，回帰テスト問題の背景にある．

2.2. 仕様ベースのテスト

仕様ベースのテストとは，仕様を基に実装されたプログラムをテスト方法である．仕様とは，テスト対象となるプログラムが達成すべき機能とその入力や事前条件について定義したものを意味する．仕様ベースのテストは，二つの活動，すなわち，テスト設計とテスト実行に分けて行われる．テスト設計とは，仕様に基づき，テストケース（テスト入力値と期待結果のペア）を設計とする活動である．テスト実行とは，テスト入力値を基にプログラムを動作させ，その出力結果を求め，期待結果と比較し，良否を判定する活動である．

次に，仕様ベースの回帰テストを考える．単純化してプログラムの二つの世代，版 Rm と版 $Rm+1$ だけで考える．版 Rm の仕様を S^{Rm}，版 $Rm+1$ を S^{Rm+1}，版 Rm から版 $Rm+1$ に加えた変更仕様を $S^{\triangle m}$ とし，実装されたプログラムを P^{Rm}，P^{Rm+1} とする．

回帰テストの前に行う，変更に伴う確認テストは，変更仕様 $S^{\triangle m}$ が P^{Rm+1} に正しく実装されたことをテストする．そのためのテストケース $TC^{S^{\triangle m}}$ を作成し，P^{Rm+1} をテストする．

その後，仕様 S^{Rm} において変更仕様 $S^{\triangle m}$ により影響を受けなかった部分，$S^{Rm} - S^{\triangle m}$ に対して回帰テスト行う．記号で影響を受けなかった部分，$S^{Rm} - S^{\triangle m}$ を表すのは簡単だが，自然言語で書かれた実際の仕様書において，この操作を行うことは非常に困難である．

仮に回帰テストの範囲を定義した仕様 $S^{Rm} - S^{\triangle m}$ が作れたとしても，この仕様から，テストケースを人手で作成することは，単体テストなどを除いて困難である．そうすると，S^{Rm} に対応するテストケース TC^{Rm} が既存であることが必要になる．

仕様ベースの回帰テストは，ベースとなる版の仕様とその仕様に対応するテストケースがしっかり維持されていることが必要である．実態として，これは難しい要件であり，変更仕様 $S^{\triangle m}$ から実装上の変更箇所を特定する活動ですら仕様をベースに行うことができず，ソースコードを調べたり，派生開発のための特別な技術が開発され活用されている [9].

2.3. モデルベースによる対策

仕様ベースのテストの問題は，自然言語による仕様記述では，厳密な仕様内容を変更とマージして保持できないことが原因と考えられる．前述の問題を解決する方法として，自然言語による仕様に代わり，定義した挙動をシミュレーションなどで返すモデルを作る方法が考えられる．モデルを作る方法は幾つかあり，システムの物理的な挙動を記述しシミュレーションを行う「MATLAB/Simulink」などを使ったモデルベース開発や，VDM++を使って仕様を詳細に記述し疑似実行する形式手法などである．究極のモデルとして，一部ではプロットタイプ用のソースコードまで生成することができる．

モデルベースのテストは，モデル自身の検証とは別に，プログラムの挙動がモデルの定義と合致していることを

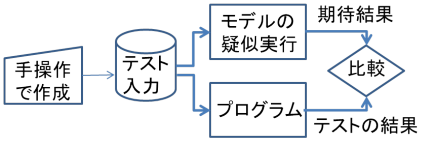


図 1. モデルを使ったテスト

確かめる．モデルが疑似実行できるなら，図 1 に示すように，プログラムとモデルに同じテスト入力を与え，結果を比較することと等価である．モデルベースによるテストの特徴は，期待結果を新たに設計する必要が無い．モデルを疑似実行することによって得られる．テストの問題の一つである，オラクル問題を解決できる．

もう一つの課題である入力値の生成問題については，幾つかの研究が行われているが，実用化には達していない [10, 11]．現時点においては，テスト入力値を作成し，網羅性を評価する必要がある．

回帰テストについて考える．版 Rm と版 $Rm+1$ に対応するモデルを M^{Rm}，M^{Rm+1} とし，プログラムは P^{Rm}，P^{Rm+1} とする．テストは，モデル M^{Rm+1} と P^{Rm+1} の挙動を比較する．モデルが持つ挙動のすべてを網羅する，テスト入力値については，モデルの記述から作成する必要がある．その方法やコストについては，ここでは検討しないが，容易では無い．

モデルベースの変更においては，変更に伴う確認テストと回帰テストを分離できない．版 Rm におけるモデル M^{Rm} に，変更を加えたモデル P^{Rm+1} が，モデル上，未改造部分に欠陥が入り込んだり発現しないないかは，モデル自身の回帰検証の問題であり，プログラムに対する回帰テストでは検出できない．

モデルベースにおける回帰テストをまとめると．モデルが正しく作られていれば，オラクル問題は解決できる．テスト入力の生成については，実用的なツールは出現していない．回帰テストの主役は，モデルの変更におけるモデル自身の検証に依存する．現実的な問題として，モデル化されていない既存のプログラムから，疑似実行可能なモデルを作るアプローチは非常に難しい．

2.4. 実装ベースのテスト

実装ベースのテストとは，実装物であるプログラムを静的，あるいは動的に解析し，テストを行う方法である．テストと言っても，本質的に仕様との合致を確認する期

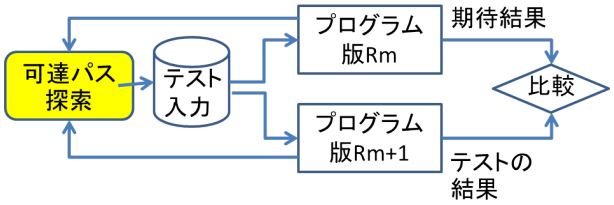


図 2. 実装ベースのテスト構成概要

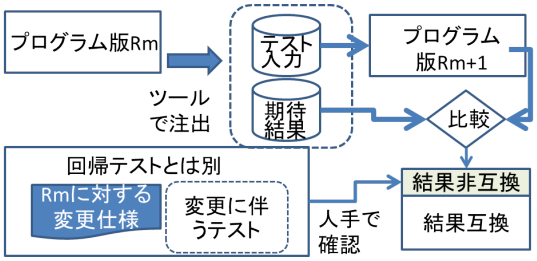


図 3. 実装ベースの回帰テスト構成概要

待結果を実装から作ることは不可能である．合否を判断するためには，何らかの期待結果を決める事後条件を指示する必要がある．例えば，メモリーリークやオーバーフローが生じない，セキュリティホールが無いことを確認するために活用されている．この種のテストは実装ベースのテストでは実現できない．

仕様との比較を行う先行研究の一つに，プログラムが持つ制御構造をツールを用いて探索し，プログラムに実装されている論理を決定表で表し，仕様から求めた決定表と比較する方式が提案されている [3, 4, 5]．この方法は，マシンを使ったテスト実行を省略できることが大きな特徴である．

プログラムが持つ制御構造を探索する方法は，対象とする変数を指定し，その変数が関与するすべての条件判断に対して，チェックポイントを埋め込み，すべてのチェックポイントを通過する変数値を探索する．チェックポイントを埋め込む前に，ソースコードは抽象構文木に展開している．探索の方法は，制約ソルバーを使って分岐条件を探り，解を求める．制約ソルバーで解けない場合は，乱数を使うなどヒューリスティックな手段を併用する．2008 年頃から，実用的なツールが Java,C,C++などの言語に対して公開されている [12, 13, 14, 15, 16, 17].

この方法によって可達パスが求まる．このツールをここでは，可達パスの探索ツールと呼ぶ．可達パスを求めることは，同時に可達パスを駆動する変数（複数）の入力値が決まるので，この値をテスト入力値として用いる．4 章の実験では，Concolice test の CREST を可達パスの探索ツールとして用いた．

実装ベースの回帰テストについて考える．テスト入力値については，プログラムから探索して求める．オラクル問題の解としては，前世代の版を利用することにより実現する．そのその構成概要を図 2 に示す．

実際の回帰テストでは，版 Rm と版 $Rm+1$ の挙動には差が有り，変更 $S^{\triangle m}$ が加えられているため，単純に図 2 に示すような比較では，回帰テストの目的を達成できない．変更が行われてい部分に対するテスト入力値に限定する必要があるが，これを求めることができない．

理由は，P^{Rm} は互換を確認するベースとして，正しいと仮定できる．しかし，変更が行われた結果である P^{Rm+1} には，変更を加えたため欠陥が含まれている可能性がある．その欠陥には，変更仕様を逸脱したものも考えられるため，仕様から範囲を決めることができない．そこで，図 3 に示すように，P^{Rm} から抽出した TC^{Rm} を用いて P^{Rm+1} をテストし，得られたテスト結果から，結果として非互換の部分（結果非互換）と互換の部分（結果互換）を識別する．

得られた結果非互換と，変更仕様 $S^{\triangle m}$ の内容，あるいは，それに基づく変更に伴う確認テストのテストケースから，結果非互換の項目について消込を行い，残った項目が有れば，デグレードの可能性がある．結果互換に欠陥があるとすれば，変更仕様 $S^{\triangle m}$ で定義された変更の漏れであるが，回帰テストの目的を超えるので議論しない．結果非互換の項目について消込は，人手による活動である．

仕様ベースのテストが，正しい結果を想定してテストケースを漏れなく設計するのに対し，実装ベースのテストは，プログラムが持つ非互換の部分を抽出し，それが仕様を満たしているか否かは，テスト結果を基に後から解析する．詳細については 3 章で示す．

2.5. 実装ベースを支える可達パスの探索ツール

実装ベースのテストでは，実装が持つすべての入力域を探索するツールが必要になる．プログラムが持つ可達パスを探索する研究は，モデル検査から発達した Symbolic Execution として活発に研究が進んでいる [18, 19]．近

年，抽象構文木で中間言語生成するコンパイラーが出現し，疑似実行の環境が開発されたこと，及び，制約ソルバーの発達から実用的な可達パスの探索ツールが開発され，普及が進んでいる．

ここでは，Symbolic Execution の応用である Concolic Testing を使用する．Concolic Testing とは，Sen, Koushik と Agha, Gu が 2006 年に発表した論文の中で定義した Concreat と Symbolic を合わせた造語である [16]．Concolic testing のためのオープン系可達パスの探索ツールとしては，CREST が公開されている [20]．ここでは，この可達パスの探索ツールを使た実装ベースの回帰テストについて示す．

CREST の探索能力については，多くの応用研究が行われ確認されている．代表的なものでは，Linux の core util を対象に行われた検証で，人手による十数年間のテストで発見できなかった欠陥を見つけている．grep など再帰呼び出しに対しても効率的にパスを探索することができる．規模としては 15 万ステップのプログラムに対する例が報告されている．

全ての場合において，制約ソルバーで解析的にパスを解けないため，ヒューリスティックな手段を併用する．よって，探索オプションなどを調整する必要がある．それでも完全に解けない場合もあるが，人手による探索と比べ高い網羅性を達成している．

3. 実装ベースの回帰テスト

実装ベースの回帰テストにおいて必要なものは，変更前と後のソースコードと，変更に関する仕様（変更仕様）である．ここでは変更前のプログラムを P^{Rm}，変更後を P^{Rm+1} と表示し，それがソースコードなのか実行形式なのかは文脈で表す．変更に関する仕様は，変更に伴う確認テストを設計できる詳しさで記述されているとする．変更に伴う確認テストとは，変更によって意図した変化が正しく動作していることを確認できることである．変更に伴う確認テストは，回帰テストの前に実行されてれいることを前提とする．

このような条件下で，実装ベースの回帰テストとして二つの方式「可達パス網羅の回帰テスト」と「同値組合せ網羅の回帰テスト」を提案する．

3.1. 可達パス網羅の回帰テスト

可達パス網羅の回帰テストは，回帰テストのテスト入力値の網羅基準として，可達パス網羅を用いる．

図 3 に示した実装ベースの回帰テスト構成概要を基に，その手順について以下に説明する．

(A1) 可達パスの探索 可達パスの探索ツールを用いて Rm のソースコードに対し，可達パスに関する情報を得る．

ここでの入力は，Rm のソースコード，出力は，テストケース TC^{Rm} である．

(A2) クロステストの実行 通常のテストハーネスを用い，Rm から得られた TC^{Rm} を用いて P^{Rm+1} をテストする．このクロステストは Rm から見た $Rm+1$ との互換/非互換を検出する．互換であるか非互換であるかは，Rm から得られた期待結果と比較することにより判断している．

ここでの入力は，テストケース TC^{Rm} と TC^{Rm+1}，出力は各テストケースの判定情報，すなわち合格＝互換，不合格＝非互換の一覧である．

(A3) 解析 この部分は，人手によって行う．ここでの入力は (A2) の出力であるテスト実行の結果と変更仕様である．出力は，非互換部分が変更仕様に基づくものか否かの判断である．

人手による判断の難易度は，主に変更の量に依存する．(A1)(A2) は，自動化出来ることから，変更を小さな単位で区切り，繰り返し確認することにより，判断の難易度を下げることができる．

以上に示した，可達パス網羅の回帰テストの特徴は，人手によるテストケース作成を一切行わない．可達パスの探索ツールの設定やテスト環境が出来上がれば，テストケースを人手により作成する作業から解放される．可達パスを探索するスピードと網羅性は，圧倒的に人手による作業能力を越えている．

解析者は，デバッグを除けば，プログラムのロジックを見ることなく，プログラムの入出力の関係から変更仕様に合致しているか否かを確認できる．但し，変更仕様により変更されたロジックについては解析し，非互換の精査を行う必要であるが，これは，仕様ベースのテストでも同様である．

この方式で回帰テストが出来るか否かは，入力変数の値に依存しない出力を生成するプログラムを除けば，用いる可達パスの探索ツールでテスト入力値を漏れなく検出できるか否かにかかっている．入力変数の値に依存しない出力とは，乱数生成に基づくロジックなどを意味する．また，用いる可達パスの探索ツールから，ソースコードを駆動するため，変数を指定して接続するためのインタフェースを作成する必要がある．

可達パスの探索ツールは，取り扱える変数（整数，ビット列，浮動小数点，など）と組込まれたソルバーによってその能力が決まる．解析的に解けない場合は，ヒューリスティックな探索を行うため，プログラムの規模が大きい場合や，再帰呼び出しが多いと，解析に時間がかかる．

ここでの提案は，実装ベースの回帰テストに関する方式であり，可達パスの探索ツールの能力そのものについては対象としていない．

3.2. 同値組合せ網羅の回帰テスト

同値とは，テスト技法で用いられる用語であり，テストの入力となる変数の変数域に着目し同値分割を行い得る．変数域をテストのために同値クラスに分割し，その中で選んだある値である．変数は複数の同値を持つ．テストの規格 (ISO/IEC 29119-4) では，変数と選んだテスト入力値を P-V pair(Parameter - Value) と呼び，テスト設計の基本的な成果物である．Value に同値を用いることにより，少ないテスト入力で効果的な網羅基準を達成することができる．

同値組合せ網羅の回帰テスト構成を図 4 に示す．可達パス網羅の回帰テストとの違いは，ツールが生成するテスト入力値ではなく，一度，変数の持つ同値を求め，その後，変数の同値を組合せてテスト入力値とする．組合せを指定することにより，テストの網羅性基準を調整することができるのが特徴である．その手順を次に示す．

(B1) パスの探索と変数の同値抽出 ツールを使う点は，可達パス網羅の回帰テストにおける (A1) と同じであるが，出力は変数に対する同値のリストである．

(B2) 同値の組合せからテスト入力値作成 変数間の組合せを指定してテスト入力値を作成する．変数は同値の数だけバリエーションを持つので，すべての組合せだと膨大な数になる．回帰テストのリソースや重要性から，組合せを指定する．例えば All-pair 法などを用いて網羅基準を緩和することが考えられるが，マシンパワーがあれば，その必要は無い．

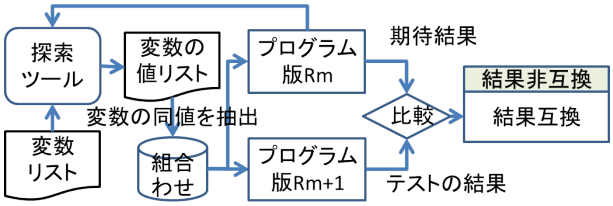


図 4. 同値組合せ網羅の回帰テスト構成

(B3) 期待結果の取得 (B2) で作成したテスト入力を使って期待結果を得る．

(B4) テストの実行 (B2) で作成したテスト入力を変更後のプログラムに与えテストを行い結果を得る．結果は (B3) で得た期待結果と比較し，非互換か互換に分割する．

(B5) 解析 この活動は，可達パス網羅の回帰テストと同じである．

同値組合せ網羅の回帰テストも，可達パス網羅の回帰テストと同様な特徴を持っている．異なるのは，網羅の基準を調整することができる点である．

4. 回帰テストの実験

先に示した実装ベースの回帰テストについて実験し評価を行う．実験に用いたプログラムは，ロジックの設計やテストを行うための演習問題を使った．簡単な問題ではなく，人手で解くには複雑な組合せが生ずる問題である．

4.1. 用いた例題：割引問題の仕様

このプログラムは，ある施設の入場割引に関するもで，入力として割引の条件が与えられており，それぞれの条件に対して，正規料金に対する割引後の値を%で返す．その仕様は次の通り．

1). 3 歳以下の場合には無料とする．(正規の 0%)
2). 水曜日なら，正規の 90%とする．
3). 60 歳以上なら，正規の 60%とする．
4). 女性の場合，50 歳以上なら正規の 65%とする．

5). 施設の記念日なら，正規の 80%とする．
6). 施設の地域住民なら，正規の 50%とする．
7). 15 時以降の入場なら正規の 70%とする．
8). 12 歳以下なら正規の 40%とする．
9). 但し，条件が重複する場合には，割引の大きい方を選択する．

この仕様を満たすプログラム waribiki_Rm.c があり，これを変更前のプログラムとする．このプログラムに次の仕様を追加したものを waribiki_Rm+1.c とする．
10). 冬季（1 月，2 月）なら，67%とする．

4.2. 回帰テストの条件

4.1 で定義した仕様から作られたプログラムに対して回帰テストを試行する．テスト側の条件としては以下を設定する．

- コード waribiki_Rm.c と同_Rm+1.c は与えられる．解析してもよい．
- コード waribiki_Rm.c に対するテストケースは引き継がれていない．
- 変更仕様として「冬季（1 月，2 月）なら，67%とする.」が waribiki_Rm+1.c に加えられたことを知っている．
- 割引全体に関する仕様（1. から 9.) は知らない．

4.3. 仕様ベースの回帰テスト

比較のために，仕様ベースの回帰テストについては机上考察を行う．回帰テストの条件で，元の仕様（1. から 9.) は解らないが，変更仕様は入手しているので，そこから変更に伴う確認テストを作る．そのテストケースとしては，次のものが考えられる．

1). 2 月なら 67%
2). 1 月なら 67%
3). 3 月から 12 月は 67%にならない，他は今までと同じ

他の影響範囲を推測するには，ソースコードを解析する必要があるが，ここでは考えない．もし，ソースコードを正確に理解すれば，この例題の仕様は，9 番目の仕様に「条件が重複する場合には，割引の大きい方を選択する」とあることから，10 番目に追加された仕様の意味は,「12 歳以下では無くて，地域住民では無くて，50 歳以上の女性ではなくて，60 歳以上でもない」場合において，冬季（1 月，2 月）なら，67%とする処理が追加されることである．同様な依存関係から，他の割引条件にも影響が生じる．仕様を正しく伝えても，この影響に気付くことは難しく，仕様を基に漏れの無い回帰テストを行うことは，誰にでも簡単にできることではない．

4.4. 可達パス網羅の回帰テスト

waribiki_Rm.c から waribiki_Rm+1.c へ変更が行われたとして,「可達パス網羅の回帰テスト」を実験する．ここでは次の手順で回帰テストを進める．

1). テストドライバの開発
2). Concolic Testing によるテストケース探索
3). クロステスト
4). 結果の差分から回帰テストの判定

(1) テストドライバの開発 二つのテストドライバを準備する．一つは Concolic Testing により可達パスを探索し，テスト入力値を求める．ここで用いたツールは，可達パスに対して実行結果も出力することができる．ここでは Concolic ドライバと呼ぶ．

もう一つは，テスト入力値をファイルから読込，結果を出力する普通のテストドライバである．Concolic ドライバは，簡単で次のようなコードである．

```
//Concolic のためのドライバTesting
#include <crest.h> /* Concolic header */
#include <stdio.h>
#include <stdlib.h>
int main(){
int sex,age,dayofweek,citizen,month,
    memorialday,intime;
  CREST_int(sex); /* variable to crest */
  CREST_int(age);
  CREST_int(dayofweek);
  CREST_int(citizen);
  CREST_int(month);
  CREST_int(memorialday);
  CREST_int(intime);
  int t_no=1;
/*（functionwaribiki)とテストケースの出力call*/
printf("%d %d %d %d %d %d %d \t\t %d\n",
sex,age,dayofweek,citizen,month,memorialday,
intime,waribiki(sex,age,dayofweek,
citizen,month,memorialday,intime));
```

(2)Concolic Testing によるテストケース探索
CREST を使って，P^{Rm} と P^{Rm+1} の探索を行う．P^{Rm+1} を探索するのは，変更が削除の場合を $P^{Rm+1} \rightarrow P^{Rm}$ として実験するためである．探索オプションは-dfs: BoundedDepthFirstSearch を用いた．ツール CREST は，整数の場合 long で膨大な範囲を探索するので，事前条件として範囲を限定する処理を Concolic ドライバに入れてある．探索されたテストケースを表 1 に示す．可達パス網羅の計測方法が無いので gcove を使った分岐網羅を参考に示している．

表 1. Concolic ドライバの結果

版	ケース数	実行時間	分岐網羅
Rm	48	0.274s	100%
$Rm+1$	72	0.522s	100%

表 2. Rm の結果から $Rm+1$ をテストした差異

性	歳	曜日	住居	月	記念	入場	改造後	改造前
1	13	1	0	1	0	10	67	100
1	13	1	0	1	0	16	67	70
1	13	1	0	1	1	10	67	80
1	13	1	0	1	1	16	67	70
1	13	1	0	2	0	10	67	100
1	13	1	0	2	0	16	67	70
1	13	1	0	2	1	10	67	80
1	13	1	0	2	1	16	67	70
1	13	6	1	1	0	10	67	70
1	13	6	1	1	0	16	67	70
1	13	6	1	2	0	10	67	70
1	13	6	1	2	0	16	67	70
2	13	1	0	1	0	10	67	100
2	13	1	0	1	0	16	67	70
2	13	1	0	1	1	10	67	80
2	13	1	0	1	1	16	67	70
2	13	1	0	2	0	10	67	100
2	13	1	0	2	0	16	67	70
2	13	1	0	2	1	10	67	80
2	13	1	0	2	1	16	67	70
2	13	6	1	1	0	10	67	70
2	13	6	1	1	0	16	67	70
2	13	6	1	2	0	10	67	70
2	13	6	1	2	0	16	67	70

(3) クロステスト P^{Rm} から得られたテストケース 48 項目を P^{Rm+1} に与え，Rm の結果と異なるものを比較したところ，48 ケース中 18 ケースで結果が一致しない非互換が見つかった．非互換である 18 のテストケース（入力と結果）を表 2 に示す．$Rm+1$ から得られたテストケース 72 項目を Rm に与え，$Rm+1$ の結果と異なるものを比較したところ，72 ケース中 24 ケースが異なっていた．

(4) 結果の差分から回帰テストの判定 クロステストの結果から，異なっていたものの共通点は，$Rm+1$ 側の結果である割引がすべて 67%であった．これは，変更仕様で追加した割引と一致している．他の割引で非互換のものは見つからなかった．可達パスによる回帰テストの網羅は，変数のすべての組合せを網羅しているので，挙動上の違いをすべて見つけている

この程度の差であれば，人がチェックすることも出来るが，差の数が増加すると比較が困難になる．対策としては，実装ベースのテストの先行研究で提案されている決定表で表現する．その例を表 3 に示す．テスト結果と比較する仕様側のロジックも決定表で表現できれば，比較が簡単である．表 4 にその例を示す．仕様で用いる決定表はビジネスルールとも呼ばれている．

なお，逆の変更として P^{Rm+1} のテストケースで PP^{Rm} をテストし他場合，18 件の非互換を検出し，変更仕様後と前の差を検出できた．

表 3. 決定表によるテストケースの表現

	テストケース	1	2	3	...	9	10
原因	性別 {1,2}	1	1	1		2	2
	齢 {0,4,13,50,60}	0	4	4		13	13
	曜日 {1,3,6,7}	1	1	1		3	1
	住居 {0,1}	0	1	0		0	0
	月 {1,2,3}	1	1	1		1	1
	記念日 {0,1}	0	1	0		0	0
	入場時 {10,16}	10	10	10		10	10
結果	0	X					
	30		X				
	40			X			
	80						
	90					X	
	100						X

表 4. 仕様の決定表（ビジネスルール）

	規則	1	2	3	4	...	9	10	11	12
条件	性別					省略				ELSE
	年齢	3 以下		12 以下						
	曜日				平日				水曜	
	住居		真		住民		真			
	月									
	記念		真					真		
	入場時間									
動作	割引後%	0	30	40	50		70	80	90	100

以上の実験の結果，提案した「可達パス網羅の回帰テスト」を使って，この例題について，人手を介さず互換/非互換となるプログラムに対する入力値と結果を洗い出せた．全てのプログラムに対して有効か否かは，この実験だけでは解らないが，この方式で回帰テストが実現できることは明らかになった．

4.5. 同値組合せ網羅の回帰テスト

次に，同値の組合せからテスト入力値を作成して回帰テストを行う「同値組合せ網羅の回帰テスト」について実験する．用いるのは，先の実験と同じ waribiki_Rm.c から waribiki_Rm+1.c である．手順的には，Concolic Testing で得られたテストケースから，(1) 同値を抽出し，(2) 同値組合せを作成し，(3) 期待結果の取得を行う．その後は，可達パス網羅と同様である．

(1) 同値の抽出 4.4 の (2) の結果から入力の同値を抽出する．その方法は，可達パスの探索ツールが生成した各入力変数のユニークな値である．項目別に sort し uniq コマンドで簡単に入手できる．抽出され各変数の同値の数は，Rm に対し性別:2, 年齢:5，曜日:4，住居:2，月:1，記念:2，入場時間:2 であった．変更後の $Rm+1$ については，変数月に対して同値が 1 から 3 に増加する．

(2) 同値の組合せ 同値から組合せを作成する．人手で組合せるのは大変なので all-pair を生成するツールを使って生成した．2 変数間の全同値の組合せだけでなく，全変数間の全同値の組合せまで生成できる．ここでは，全変数間の全同値の組合せを行い Rm において 480，変更後の $Rm+1$ において 1440 であった．

(3) 期待結果の取得 (2) で得られた同値組合せを入力として，Rm と $Rm+1$ を動作させて得る．実行時間は 0.003 秒と 0.005 秒であり，表 1 に示した concolic Testing と比べれば，テスト数が 10 倍であるが，実行時間は 100 倍ほど早い．

(4) クロステストの実行と解析 クロスでテストを実行した結果，Rm から得られたテストケース 480 項目を $Rm+1$ に与え，Rm の結果と異なるものを比較したところ，86 ケースが異なっていた．$Rm+1$ から得られたテストケース 1441 項目を Rm に与え，$Rm+1$ の結果と異なるものを比較したところ，165 ケースが異なっていた．可達パス網羅と同様，差が生じたテストケースの $Rm+1$ 側の出力結果は，67%であり変更仕様を満たしていた．

同値組合せ網羅の実験において，Rm と $Rm+1$ のクロステストにより見つかった非互換は，86 件と 165 件であった．この非互換としたテストケースで共通のものを除外した 42 件がユニークな非互換であった．この 42 件は，すべて変更仕様を満たしていた．

変更仕様から，机上でこの 42 件を予見するのは難しいと思われる．変更を論理的に考え，プログラマーが機能やロジックを変更することと，その結果として，利用者から観た入力と結果の挙動の違いを数え上げることとは，本質的に違う活動である．利用者には，変更が正しく行われたか否かより，生ずる変化に対して受け入れられるか否かが問題となる．実装ベースの回帰テストは，利用者の視点から，何が変化したのかを明らかにすることができる．

4.6. 実験のまとめ

この実験で用いたソースコードを示す．

```
// waribiki_Rm
int waribiki(int sex,int age,int dayofweek,int citizen,
int month,int memorialday,int intime) {
int  discount=100;

if ( age <= 3 ) discount=0;
else if (memorialday == 1 && citizen == 1) discount=30;
else if ( age <= 12 ) discount=40;
else if ( citizen == 1 && (dayofweek >=1 && dayofweek
        <6) ) discount=50;
else if ( age >= 60 ) discount=60;
else if ( sex == 2 && age >= 50 ) discount=65;
//else if ( month ==1 || month ==2 ) discount=67;
// Rm+1 では，この部分を追加
else if ( intime >= 15 ) discount=70;
else if ( memorialday == 1 && (dayofweek ==6 ||
      dayofweek ==7) ) discount=70;
else if ( citizen == 1 ) discount=70;
else if ( memorialday == 1 ) discount=80;
else if ( dayofweek == 3 ) discount=90;
else discount=100;

/* 後処理 */
  return discount;
}
```

仕様上は一つの条件と処理を追加するだけの変更であり，実際のコーディングも else if (month ==1 —— month ==2) discount=67; を一行追加するだけである．しかし，仕様ベースの回帰テストでは十分なテストが出来ないことは明らかである．

一方，実装ベースではテストケースを作成せずに，正確に互換/非互換を検出した．可達パス網羅では，プログラムの制御構造上の特性から，同値組合せ網羅より少ないテストケースとなった．

どちらも自動実行であり，その処理時間も小さいことから優劣は無いと考える．利用者に，挙動上の差を説明する．あるいは，挙動上の差から派生開発の良否を判断するには，同値組合せの方が漏れが少ないと思われる．

この実験では，特別な欠陥を挿入していないが，プログラムに存在するすべての条件分岐を検出するので，定義された変数に対して仕様外の条件分岐があれば検出できる．しかし，定義されない入力変数については検出できない．

5. おわりに

現状の回帰テストの課題に対して，その原因が仕様ベースのテストを使っていることから生じていることを示し，解決案として，実装ベースの回帰テストを提案した．実現する方法として，可達パス網羅と同値組合せ網羅を示し，その方法について実験を行い，実現できることを示した．

参考文献

[1] 深谷直彦，古川善吾，西康晴，他，“特集 ソフトウェアテストの最新動向,” 情報処理，vol.49，no.2，pp.126–173，2008.

[2] JSTQB 技術委員会，“ソフトウェアテスト標準用語集 日本語版,” http://www.jstqb.jp/dl/JSTQB-glossary.V2.3.J02.pdf.

[3] 植月啓次，松尾谷徹，津田和彦，“効率的なテスト設計を実現するデシジョンテーブルの拡張法,” ソフトウェア・シンポジウム 2013 論文・報告，2013.

[4] 植月啓次，“ソフトウェアの実装情報に基づく決定表を活用した論理検証手法,” ソフトウェア・シンポジウム 2013 論文・報告，2013.

[5] K. Uetsuki, T. Matsuodani, and K. Tsuda, “An efficient software testing method by decision table verification,” International Journal of Computer Applications in Technology, vol.46, no.1, pp.54–64, 2013.

[6] ボーリス・バイザー，ソフトウェアテスト技法，日経 BP 出版センター，1994.

[7] J. SC7/WG26, Software and systems engineering — Software testing Part: 4 Test techniques, ISO/IEC/IEEE，1994.

[8] E.T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, “The oracle problem in software testing: A survey,” IEEE SE, vol.41, no.5, pp.507–525, 2015.

[9] 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，2007.

[10] A.S. Santos, “Vdm++ test automation support,” Master’s thesis, Minho University with exchange to Engineering College of Arhus, 2008.

[11] M. Utting, A. Pretschner, and B. Legeard, “A taxonomy of model-based testing approaches,” Software Testing, Verification and Reliability, vol.22, no.5, pp.297–312, 2012.

[12] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, “Model checking programs,” Automated Software Engineering, vol.10, no.2, pp.203–232, 2003.

[13] G. Stergiopoulos, B. Tsoumas, and D. Gritzalis, “Hunting application-level logical errors,” Engineering Secure Software and Systems, pp.135–142, Springer, 2012.

[14] C. Lattner and V. Adve, “Llvm: A compilation framework for lifelong program analysis & transformation,” Code Generation and Optimization, 2004. CGO 2004. International Symposium onIEEE, pp.75–86 2004.

[15] C. Cadar, D. Dunbar, and D.R. Engler, “Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs.,” OSDI, vol.8, pp.209–224, 2008.

[16] K. Sen and G. Agha, “Cute and jcute: Concolic unit testing and explicit path model-checking tools,” Computer Aided VerificationSpringer, pp.419–423 2006.

[17] J. Burnim and K. Sen, “Heuristics for scalable dynamic test generation,” Proceedings of the 2008 23rd IEEE/ACM international conference on automated software engineeringIEEE Computer Society, pp.443–446 2008.

[18] J.C. King, “Symbolic execution and program testing,” Communications of the ACM, vol.19, no.7, pp.385–394, 1976.

[19] C. Cadar, P. Godefroid, S. Khurshid, C.S. Păsăreanu, K. Sen, N. Tillmann, and W. Visser, “Symbolic execution for software testing in practice: preliminary assessment,” Proceedings of the 33rd International Conference on Software EngineeringACM, pp.1066–1071 2011.

[20] CREST, “Concolic test generation tool for c,” http://jburnim.github.io/crest/.

安全系組込ソフトウェア開発におけるユニットテストの効率化 ～Concolic Testingの活用事例～

岸本 渉
株式会社デンソー
wataru_kishimoto@denso.co.jp

要旨

機能安全など，安全系のソフトウェアに対する詳細なテスト要件は *MC/DC* カバレッジなど具体的な要求として浸透し始めている．人手に頼ったテストでは抜け漏れ等の問題と，コスト増の問題が生じ対応できないため自動化が進んでいる．しかし，現有のツールでは *C0*，*C1*，*MC/DC* カバレッジ *100* %を達成することが出来ず，その部分を人手で対処するため大きな損失が生じている．

本稿では，ソフトウェア工学における最新の研究成果が反映しているオープン系のツール *Concolic Testing* を活用してこの問題を解決したので報告する．オープン系のツールは導入時に，技術的なスキルが求められるが，様々な工夫を行うことが出来るので，柔軟に対応し，導入時の問題を解決したので報告する．

1. はじめに

我々は，センサで車両の状態を検知し，その結果を用いて安全系装置を制御する組込ソフトウェアの開発を担当している．このシステムは，車種やグレードにおけるオプション搭載ではなく標準搭載が進み，他サプライヤとの価格競争が激化している．高い品質と自動車機能安全規格 ISO26262 に準拠するなど安全性を担保しつつ，一層のコスト削減が求められている．さらに，車両の安全系機能が多様化し，従来以上に開発短納期化の要求が強くなり，コスト削減以上に開発サイクルの短期化が求められている [1].

このような状況に対応するには，従来のテスト方式を基に要員の増強や方式改善では限界がある．当然，自動化を進める必要があるが，従来の自動化のように人手で出来ることをスクリプトや記述言語で開発，蓄積して実行を自動化する方式ではやはり限界がある．

ソフトウェア工学では，近年，モデル検査の技術を拡張し，ソースコードを探索する技術が急激に発達している [2]．静的解析では，有力なプロプライエタリ系のツールが出現し，この分野では人手による方式を完全に駆逐しつつある．しかし，仕様ベースのテストについては，様々な研究が行われているが，有力な自動化ツールは存在しない．

現状では，我々が求める自動化を達成できる統合ツールは存在しないが，部分的には利用できる技術が出現し，それらの技術は現在も発達中である．具体的には，Symbolic Execution や Concolic testing など，ソースコードの高度な解析を行う技術を指す [3, 4, 5]．この分野では，NASA の JPF/SPF や LLVM/KLEE や CIL/CREST などオープン系のツールが先行しており，これらを超えるプロプライエタリ系のツールは出現していない [6, 7, 8].

本論文では，高度な解析技術を実際のユニットテストに活用する試みについて報告する．この種の技術は，最先端のオープン系技術を使いこなす高度なエンジニアを育成しないと利用することが困難である．将来に向けて新たなソフトウェア生産技術を構築する試みとして評価を兼ねて行った．

本論文の構成と主な内容は以下の通りである．2 章では，現行のユニットテスト方式における課題を明確にし，3 章では，その対策と導入するツールについて説明する．4 章では，我々のソースコードに対してツールは有効に機能するのかを確認した．その方法は，あるプロジェクトのソースコードのなかからサンプリングしたものを評価用に書きなおしたコードを使って行った．その結果，

ツールを使うための工夫を行えば十分使えることが判った．5 章では，実際にプロジェクトにおけるユニット群に対し，現行のツールでは MC/DC カバレッジ 100%が未達であるユニットに適用し，達成できることを確認できた．6 章では，この結果からテスト方式を変更することについて検討した．

2. 現行ユニットテスト方式の分析

安全系組込ソフトウェアは，自動車機能安全規格 ISO26262 に準拠することが求められている．そのため，我々の職場では，ユニットテストにおいて，実装マイコンコードを評価対象としてマイコンシミュレータでユニットテスト実行を行う仕組みを持つツールと，そのツールを効率よく用いるための内製ツールを採用している（図 1）．このユニットテストツールはソース解析を行い C0，C1，MC/DC カバレッジを 100%達成するテストケースを生成する機能も持っている．

これらのツールを用いて大部分の作業を自動化しているが，実際のところ C0，C1，MC/DC カバレッジが 100%未満のテストケースが生成される場合がある．安全系のテスト対象ユニット，すなわち ASIL A 以上が割り当てられたユニットでは，人手により C0，C1，MC/DC カバレッジを 100%達成するテストケースを追加する作業が発生する．これは，単にカバレッジが悪い所を見れば追加できる容易な作業ではなく，カバレッジが達成できていない箇所に至る制御経路を解析する必要がある．そのため，安全系のユニットテストは，多くの工数を必要とし，生産性を低下させ，納期を圧迫している．

3. 対策と CREST の導入

3.1. 対策の考え方

前章で分析した通り，現在使用しているユニットテストツールだけでは十分なテストケースの生成ができない．安全系以外のユニットで C0，C1，MC/DC カバレッジが 100%未満のユニットに対しては，実機を用いた統合テストにおいてユニットテストレベルの詳細なテストを大きな工数をかけて行っており，システムテストを含めたテスト工程全体の効率化の妨げとなっている．

したがって，十分なテストケースが自動で生成できるようになれば，MC/DC カバレッジ 100%のユニットテスト対象ユニットを安全系かどうかに関わらず全ユニットに拡大することができる．それによって，統合テストにおけるユニットテストレベルの詳細なテストが不要となり，生産性と品質の両方を向上できる．

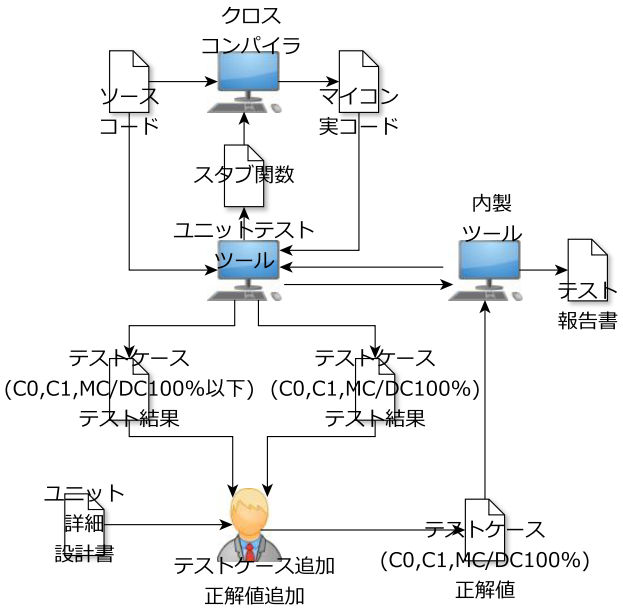


図 1. 現行ユニットテスト方式

3.2. ツールの選択

ユニットテストのテスト設計に新しいテストツールの適用を検討することとした．ユニットテストの自動化については様々なアプローチが研究されているが，本稿では，現場にて実用化するため，容易に試用でき，事前評価できることが求められることから，研修用に提供されている環境 (RAM は 2GB，CPU は Xeon E5520 2.26GHz，OS は CentOS) で CREST を用いることとした．CREST とはオープンソースとして公開されている C 言語向け concolic testing ツールである [9].

3.3. CREST 概要

CREST は Concrete Execution と Symbolic Execution を組み合わせた Concolic Testing を行うツールである．ここでは，まず Symbolic Exectution と Concolic Testing について説明する．

(1)Symbolic Execution:

Symbolic Execution は，プログラムの制御フローを解析する手段である．通常のプログラム実行は，変数に代入された値を使って実行するが，Symbolic Execution は変数を抽象解釈 (abstract interpretation) と呼ばれる方法で取扱い，各条件分岐の組合せを制約式として得る．得られた制約式を制約ソルバー (constraint solver) を用いて解く．

(2)Concolic Testing:

Concolic Testing は固定の具体的な値を使ったテストと Symbolic Execution を連携させて実行することにより，プログラムが持つすべての可達経路を網羅するテストデータを生成することを目標としている．経路探索は，Symbolic Execution により分岐点における分岐条件を求め，実行されなかった経路を選択するための入力値を制約ソルバーで求め，その具体的な入力値でプログラムを実行する，という流れである．

Symbolic Execution との違いは，すべてを疑似実行するのではなく，実コードでの実行とハイブリッドで行う点にある．この特徴を生かし，テスト対象の範囲を広く取り扱うことができるし，ランダム探索におけるスピードも改善されている．制約ソルバーでは解析的に解けない場合，乱数を用いたヒューリスティックな解を求めるが，そのランダム探索におけるスピードも改善されている．

4. CREST の網羅性確認

現行ユニットテスト方式の欠点を補うため CREST を導入するにあたり，我々のソースコードにおいて CREST により全可達経路のテストケースを生成できるかを確認する必要がある．そこで，いくつか関数を抽出し，CREST で確認を行った．確認を行う上で，問題点を評価するため，選んだ関数の持つロジックを抽象化したサンプル関数を作成して試用した．その理由は，色々なバリエーションを試すことと，問題点について外部からコメントをもらうとき，製品そのものが持つ情報を隠ぺいするためである．

サンプル関数の抽出方法は循環的複雑度（以下，CCN）を基に選択することとした．あるプロジェクト A で作成したソースコードの各関数の CCN の分布状況を確認すると大部分が 10 以下であることがわかった（図 2）．よって，サンプル関数として CCN が 2(小)，4(中)，8(大) の 3 ユニットを抽出して試用することとした．

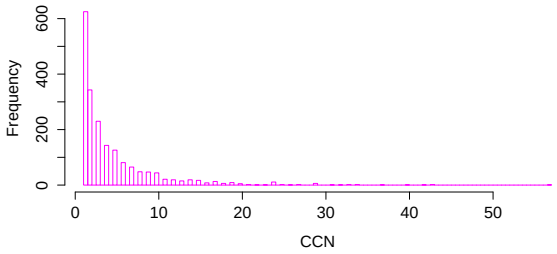


図 2. プロジェクト A で作成したソースコードの循環複雑度の分布

```
1 unsigned char ccn2( unsigned char a ){
2   unsigned char result ;
3   if ((a & 0x01U) == 0x01U){
4     result = 1U;
5   }else{
6     result = 0;
7   }
8       return ( result );
9 }
```

図 3. CCN=2 のサンプル関数

まず，CCN=2 のサンプル関数 (図 3) に CREST を試用した．探索オプションとして DFS(Depth-First Search) では，カバレッジ 100%のテストケースが生成されなかったが，その他の探索方法ではカバレッジ 100%のテストケースが生成された．DFS オプションの場合，例題のようなビット演算に対して，CREST が使っている制約ソルバー（yices）が対応していないことが原因である．韓国の SAMSUNG 社が携帯電話の OS に対して行った論文では，ビット演算のための機能追加を行ったことが書かれている [10]．ここでは，追加機能を用いず，DFS 以外の探索オプションとして CFG(Control-Flow Directed Search) を用いて解を求めた．

次に，CCN=4 のサンプル関数 (図 4) に CREST を試用した．CCN=2 のサンプル関数と同様に DFS 以外の探索方法でカバレッジが 100%のテストケースが生成されたが，引数 a が 5 行目で配列 c の領域外アクセスとな

```
 1   unsigned char ccn4( unsigned char a,
                         unsigned short b ){
 2     static const unsigned short c[2]=
                               { 510U, 514U };
 3     unsigned short d;
 4     unsigned char result;
 5     d = c[a];
 6     if ( b == 0U ){
 7       result = 2U ;
 8     }else if ( (d <= b) && (b <= 1000U) ){
 9       result = 1U ;
10    }else{
11      result = 0U ;
12    }
13    return ( result ) ;
14  }
```

図 4. CCN=4 のサンプル関数

```
1  int main( void ){
2    unsigned char a;
3    unsigned short b;
4    while ( 1 ){
5      CREST_unsigned_char( a ) ;
6      if ( a < 2U ){
7        break ;
8      }
9    }
7    CREST_unsigned_short( b );
8    printf( "%d %d\t\t %d\n", a, b, ccn4( a, b ) );
9  }
```

図 5. 入力値を制限する処理を追加したテストドライバ

```
 1 void func( void ){
 2  if ( Dat6 > 0U ){
 3      Dat6--;
 4      if ( Dat6 == 0U ){
 5        AAA[ Dat5 ]._dat2--;
 6        if ( AAA[ Dat5 ]._dat2 > 0U ){
 7          Dat6 = AAA[ Dat5 ]._dat1;
 8          if ( Dat1 == 0x01 ){
 9            Dat1 = 0x00;
10          }
12       } else {
13         Dat5++ ;
14         if ( Dat5 < 6 ){
15           Dat6 = AAA[ Dat5 ]._dat1 ;
16           Dat1 = AAA[ Dat5 ]._dat3 ;
17         } else {
18           Dat4++ ;
19           if ( Dat4 < Dat7 ){
20           } else {
21             if ( Dat3 == 0x01 ){
22               Dat4 = 0U ;
23             } else {
24               Dat2 = 0x00 ;
25             }
26           }
27         }
28       }
29     }
30    }
31  }
```

図 6. CCN=8 のサンプル関数

る値を生成していた．CREST はあくまでプログラムを実行して入力を求めているため，配列の領域外にアクセスして得られた値により分岐条件が成立すれば，それがテストデータになってしまう．

この問題を回避するために CREST が生成する a の値が 2 未満となるように条件文を追加した (図 5 の 4 行目，及び，6 行目から 9 行目)．その結果，適切なテストデータによりカバレッジが 100%となるテストケースが生成されることを確認した．

ここでテストドライバに print 文を使用しているが，CREST 以外の Symbolic Execution ツールでは，テストドライバで使用する print 文自身も経路探索の対象となり，ソースコードを与える必要が生じるが，CREST の場合は，その必要が無く使い勝手が良い．

最後に，CCN=8 のサンプル関数 (図 6) に CREST を試用した．CCN の増加だけでなく入力変数が 24 もある関数のため，何の工夫もせずに CREST によりテストケースを生成すると 14 分岐あるうちの 7 分岐まで網羅するテストケースは生成されたが，残り 7 分岐を網羅するテストケースは約 1 時間探索を繰り返しても生成されなかった．ここで，この CCN=8 のサンプル関数も変数 Dat5 が配列 AAA のインデックスとなっているため入力値を制限する処理を追加した．それにより生成されたテストケースによるカバレッジは増加したが 100%には至らなかった．そこで，可達経路を網羅する入力値の候補が多いために探索に時間がかかっていると考え，配列のインデックスとなる変数以外で取り得る値の範囲が決まっている変数に対しても入力値を制限する処理を追加した．その結果，約 1 分の探索時間でカバレッジが 100%となるテストケースが生成された．ここで行った

表 1. ユニットテストツールで生成したテストケースのカバレッジ

関数 No.	C0 カバレッジ	C1 カバレッジ	MC/DC カバレッジ	CCN
1	100.0%	100.0%	100.0%	4
2	100.0%	100.0%	100.0%	11
3	100.0%	100.0%	100.0%	12
4	100.0%	100.0%	100.0%	10
5*	75.0%	50.0%	33.0%	4
6	100.0%	100.0%	100.0%	1
7	100.0%	100.0%	100.0%	1
8	100.0%	100.0%	100.0%	2
9*	45.0%	50.0%	42.0%	8
10	100.0%	100.0%	100.0%	12
11*	97.0%	80.0%	50.0%	15
12*	52.0%	42.0%	20.0%	13
13	100.0%	100.0%	100.0%	1
14	100.0%	100.0%	100.0%	1
15	100.0%	100.0%	100.0%	1

表 2. CREST で生成したテストケース追加後のカバレッジ

関数 No.	C0 カバレッジ	C1 カバレッジ	MC/DC カバレッジ	CCN
1	100.0%	100.0%	100.0%	4
2	100.0%	100.0%	100.0%	11
3	100.0%	100.0%	100.0%	12
4	100.0%	100.0%	100.0%	10
5°	100.0%	100.0%	100.0%	4
6	100.0%	100.0%	100.0%	1
7	100.0%	100.0%	100.0%	1
8	100.0%	100.0%	100.0%	2
9°	100.0%	100.0%	100.0%	8
10	100.0%	100.0%	100.0%	12
11°	100.0%	100.0%	100.0%	15
12*	97.0%	95.0%	90.0%	13
13	100.0%	100.0%	100.0%	1
14	100.0%	100.0%	100.0%	1
15	100.0%	100.0%	100.0%	1

対策は，形式仕様で用いられている「事前条件」に相当すると考えられる．

以上のことから，CREST を我々のソースコードに適用できる目途付けを行うことができた．次章では，実在するプロジェクト A で作成したソースコードにおいて C0，C1，MC/DC カバレッジ 100%のテストケースが生成できなかった関数に対して，実際に CREST を適用し現行ツールの問題を解決できるかを確認する．

5. 現行ユニットへの CREST 適用結果

ここでは，プロジェクト A で作成した実際のソースコードに対して，CREST が現行ツールより優れているか否かを比較評価する．

まず，プロジェクト A のライブラリから無作為に C ソースファイルを抽出し，そこに含まれる関数に対して，現行ユニットテストツールによりテストケースを生成させた．その結果を表 1 に示す．対象とするソースファイルに含まれる 15 関数の内，* 印の 4 つの関数において C0，C1，MC/DC カバレッジ 100%に未達であった．

それらの関数に対して CREST によりテストケースを生成し直した結果，4 つのうち ° 印の 3 つの関数において C0，C1，MC/DC カバレッジ 100%となるテストケースが生成された (表 2)．残り 1 つの関数においてカバレッジが 100%とならなかった理由は 2 つあり，1 つは防衛的プログラミングによりメモリ破壊が発生しない限り通らない経路である．もう 1 つは図 7 のように 1 行目の 2 つ目の条件式が成立しない場合は 3 行目の 2 つ目の条件式が必ず成立するため無駄な処理となっていたためであった．

```
1 if ( (a == FALSE) && (b == TRUE) ){
2 ……
3 }else if ( (a == FALSE) && (b != TRUE) ){
4 ……
5 }
6 else {
7 }
```

図 7. 未達経路 の例

以上のことから，現行のユニットテストツールで全可達経路のテストケースを生成できなかったユニットに対して，この実験では CCN の大小に関わらず CREST により全可達経路のテストケースを生成することが確認できた．

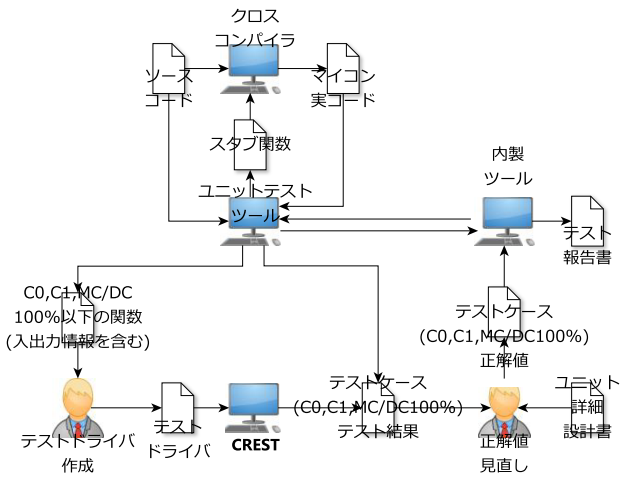


図 8. CREST 導入後のユニットテスト方式

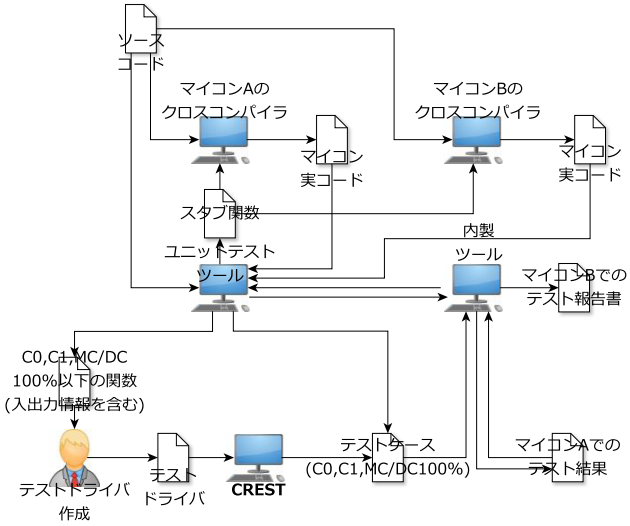


図 9. CREST 導入後のデグレードテスト方式

6. CREST によるユニットテスト方式の改善

本章では，2 章で示した現行のユニットテスト方式に対して，どのように CREST を導入するかを述べる．

まず，現状の方式ではユニットテストツールが出力するテストケースに対して，カバレッジ 100%となるテストケースと正解値を設計書を基に追加する．CREST 導入により，設計書を基にテストケースを追加する作業が，ユニットテストツールが生成した関数の入出力情報から CREST 用にテストドライバを作成する作業に置き換わる．この作業は，設計情報を知らなくてもできる作業のため，設計者のスキルがある人以外でも担当することができる．よって，設計情報を知らないとできない作業は正解値を設計書を基に見直す作業だけとなる (図 8)．これにより，ユニットテストでかかる工数は大幅に削減されると考えられる．

また，自職場ではマイコン変更，及び，コンパイラ変更を伴う派生開発も多々あり，その際に，実施するデグレードテストには，CREST の適用が更に有効である．その方式を図 9 に示す．現状，派生元のプロジェクトではユニットテストで実施可能なテストケースを実機を用いた結合テストに含めて実施していたため，デグレードテストにおいてユニットテストを実施する場合，一からテストケースの作成を余儀無くされる．しかし，CREST を導入すると可達経路を通るテストケースを得られるため，派生元のマイコン実コードと得られたテストケースを使いテスト結果を生成し，それをデグレードテストの期待値とすることで，ほとんど人手を必要としなくなる．但し，4 章で説明したようにただ入力を与えるだけでは可達経路を通るテストケースを得られない場合があり，入力値を制限する処理の追加は必要となる．また，CREST の特性やパラメータの理解と，テスト対象ユニットが持つドメイン固有のレジスタ操作などの特殊性を理解するスキルも求められる．

7. おわりに

C 言語向け Concolic Testing ツールである CREST が我々のソフトウェアに適用可能であることをサンプル関数，及び，実際の製品ソフトウェアを使って確認できた．これにより，ユニットテストにおいて課題の一つであったテストケース作成の効率化に CREST を用いることで解決の目途がついた．また，ユニットテストにおけるもう一つの課題である正解値の作成も派生開発におけるデグレードテストでは解決可能であることが確認できた．

当初，オープン系のツール CREST は，商用ツールと比べ入門向けの情報やノウハウが少なくとても困惑した．特に，動作環境を構築するのは，CREST を構成している様々なオープン系ツールの依存関係があり難しい．実際に CREST が動作する環境があれば，その環境下で試行的に使いながら問題を解決して行くのは，それほど難

しくなく，短期間で進めることができた．

今後の課題として，デグレードテストは直ぐにでも改善すべき方式であるため，現場の担当者が容易に使えるような環境の整備と保守方式を検討する．また，CREST でテストケースを生成する上で，変数の各ビットが ON/OFF のような意味を持つ入力のテストケース生成は変数のサイズが大きくなるにつれて長くなる生成時間の短縮が課題である．

最後に，この研究は，デンソー技研センターの 2014 年度ソフトウェア工学ハイタレント研修の中で行った．講師の先生と，研究の機会を与えていただいた職場に感謝する．

参考文献

[1] 大須賀竜治，“ISO26262 の日本自動車業界での活動と今後について (< 連載 > ISO 26262-自動車業界における機能安全に対する取り組み-),” 品質，vol.43，no.2，pp.212–217，2013.

[2] 梅村晃広，“SAT ソルバ・SMT ソルバの技術と応用,” コンピュータ ソフトウェア，vol.27，no.3，pp.24–35，2010.

[3] K. Sen, D. Marinov, and G. Agha, CUTE: a concolic unit testing engine for C, vol.30, ACM, 2005.

[4] J. Burnim and K. Sen, “Heuristics for scalable dynamic test generation,” Proceedings of the 2008 23rd IEEE/ACM international conference on automated software engineeringIEEE Computer Society, pp.443–446 2008.

[5] J. Foster, “Symbolic execution,” http://www.seas.harvard.edu/courses/cs252/2011sp/slides/Lec13-SymExec.pdf.

[6] W. Visser, C.S. Psreanu, and R. Pelánek, “Test input generation for java containers using state matching,” Proceedings of the 2006 international symposium on Software testing and analysisACM, pp.37–48 2006.

[7] C. Cadar, D. Dunbar, and D.R. Engler, “Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs.,” OSDI, vol.8, pp.209–224, 2008.

[8] J. Burnim and K. Sen, “Heuristics for scalable dynamic test generation,” Proceedings of the 2008 23rd IEEE/ACM international conference on automated software engineeringIEEE Computer Society, pp.443–446 2008.

[9] CREST, “Concolic test generation tool for c,” http://jburnim.github.io/crest/.

[10] Y. Kim, M. Kim, Y.J. Kim, and Y. Jang, “Industrial application of concolic testing approach: A case study on libexif by using crest-bv and klee,” Software Engineering (ICSE), 2012 34th International Conference onIEEE, pp.1143–1152 2012.

標準プロセスを肥大化させない補完型チケット駆動開発の提案

阪井　誠
SRA
sakai @ sra.co.jp

要旨

本論文では標準プロセスを肥大化させないプロジェクト運営のノウハウの蓄積方法として，補完型チケット駆動開発の利用を提案する．補完型チケット駆動開発では計画と異なるタスクがチケットとして記録されるので，チケットを参照することで，過去の想定外の事象に対してどのような対処を行ったか先人のノウハウを知ることができる．しかし，チケットからどのような情報が得られるか明らかになっていなかったことから，導入が容易であるにも関わらず広く普及しているとは言い難い．

そこで，補完型チケット駆動開発を実施したプロジェクトの追加タスクのチケットを分析した結果，７種類に分類が可能で，仕様変更，運用データ，準備作業など客先業務に固有の経験や，基盤の入れ替え，環境依存のテスト，追加の管理作業，障害の分析といった汎用的な経験が蓄積されており，標準プロセスに組み込まなくても類似プロジェクトの計画時やトラブル解決時のノウハウとして利用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組織の成熟は，ソフトウェアの品質や生産性の向上に重要である．日本においても古くから開発標準の制定や QC 活動などを通じて改善活動が行われてきた．特に 1990 年代中頃の CMM／CMMI ブームからは，標準プロセスとそれを通じた組織レベルの改善が行われてきた[1].

CMM／CMMI は広範な調査に基いており，より良い組織から集められた汎用的なベストプラクティスの集合でもある．このベストプラクティスを標準プロセスに組み込めば，ソフトウェア一般に生じる問題に対する解決策を標準プロセスに組み込むことが可能である．標準プロセスは組織内の標準として構築され，必要に応じてプロジェクトごとにテーラリングして適用される．すべてのプロジェクトは標準プロセスに基づいて実施されるので，各組織の顧客や文化に応じて標準プロセスを改良し，技術移転することで，継続的な改善を進めることが可能な仕組みである．特定の業務ドメインや顧客に特化した組織には効果的な方法である．

その一方で，ソフトウェアの適用分野は急速に多様化している．経験の少ないプロジェクトでは想定外の事象が発生してプロジェクトは混乱を招きやすい．このような想定外の事象の対処を目的として発生した作業は，実行時のダイナミックなプロセスの変更と言われ，ソフトウェア開発によくある事象としてソフトウェアプロセスモデリングのための例題として扱われている[2]．ダイナミックなプロセス変更の履歴は想定外の問題があったことと，その具体的な解決策を示す，いわゆるノウハウの源泉である．新しい業務ドメインなどで発生した想定外の事象のうち，頻度が高く影響の大きい問題は，問題の発生を防ぐ方策をなるべく前工程に組み込んでフロントローディングすると，より良いプロセスに改良できると考えられる（図 1 組織的な改善）．

このように想定外の事象が発生し，ダイナミックなプロセスの変更が行われたプロジェクトが終わると，本来ならその経験を今後に生かす目的で標準プロセスにそのノウハウを組み込んでいくことになる．しかし，このような過去に経験の少ないプロジェクトでは，以下のような問題があるので，組織レベルの改善活動では必ずしもうまく活用できない．

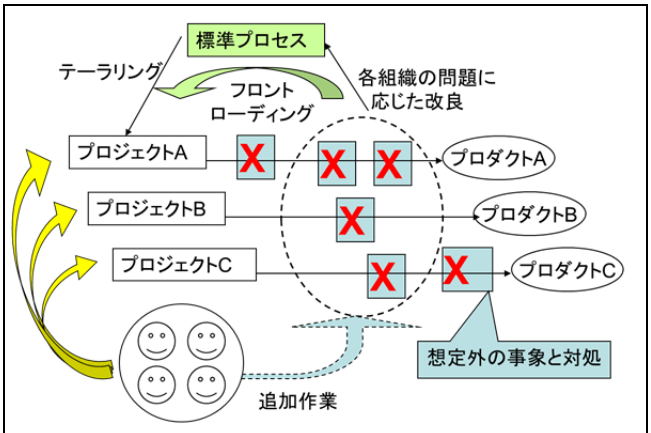


図 1 組織的な改善

- 新しい分野が増えるごとに情報を追加すると標準プロセスが肥大する
- 類似のプロジェクトが予想できない場合は標準化のモチベーションが高くなりにくい
- 経験の蓄積が必要なので，開発中の問題に対する支援ができない

特に標準プロセスの肥大化は，ウォータフォール型開発などの計画駆動型開発の問題点として，「最初にありとあらゆるものを包括した手法を開発し，（中略）たいていはかなりの額の浪費をする羽目になった」[3]と書かれており，従来から問題とされていた．

このような問題を解決する方法として，ダイナミックなプロセスの変更をすべて標準プロセスに組み込むのではなく，直接参照して活用することが考えられる．ダイナミックなプロセスの変更を収集した研究としては文献[4]がある．この文献では作業報告書から手作業でダイナミックな変更を抜き出して蓄積している．この研究ではノウハウの蓄積よりも標準プロセスの改善に注目しており，問題の原因に応じた対策を前工程で実施しようとしている．この方法は組織のプロセス改善の方法としては有効であるが，上に挙げた標準プロセスの問題を解決できるものではなかった．また，作業報告書から経験を抜き出す作業が必要なことから，標準化のモチベーションが高くなりにくい方法であった．

そこで，本研究では補完型チケット駆動開発で蓄積されるチケットに注目し，その可能性を検討した．チケット駆動開発とは，Redmine や Trac など ITS（課題管理システム）のチケットでタスクを管理する開発方法である[5]．全てのタスクを管理する完全型チケット駆動開発に対して，補完型チケット駆動開発は計画にない作業が発生した際にチケットが作成される．このため，補完型チケット駆動開発では，ダイナミックなプロセスの変更だけがチケット化されており，想定外の問題とその対策であるノウハウが自動的に記録されると考えた．

以下の章では，補完型チケット駆動開発を説明し，実際のプロジェクトのデータで検証したケーススタディについて述べ，その結果から，補完型チケット駆動開発によるノウハウの蓄積とその利用に関して考察する．

2. 補完型チケット駆動開発

本章では「チケット駆動開発」とチケット駆動開発の一種である「補完型チケット駆動開発」について述べる．

2.1. チケット駆動開発

チケット駆動開発は ITS のチケットを作業管理(タスクマネジメント)に用いる開発法である．障害票や課題票にあたるチケットを用いてタスクを管理することで，障害の管理と同じように，現状でどのようなタスクがあり，だれが担当し，どのような議論が行われ，どのような状態であるかをチケットで可視化する．またチケットには，今後のタスクの予定，作業中のタスクの状況，過去のタスク履歴を示しており，様々な条件で検索して利用可能である[5]．

チケット駆動開発は，たくさんの小さな修正を加えるシステムを開発されている中で，ITS の一つである Trac のチケット(障害票)を用いて開発プロセスを改善したことから始まった[6]．障害管理票であるチケットなしに，構成管理ツールにコミット(更新)してはいけない(No Ticket, No Commit!)というシンプルなルールで運用される．チケットによって情報共有されることでプロジェクトの柔軟性が高まるほか，膨大な作業はプロジェクトのリリース計画に従って構造化され，プロジェクト内で見える化される．また，作業やソースコードと関連付けられた多くの履歴が管理できるようになる．

このようなチケット駆動開発を導入すると，各担当者の日々の活動は，担当チケットの確認，作業実施，進捗更新という繰り返しのパターンになり，担当作業に集中することができる．また，リリースに向けても，リリース計画，チケット登録，チケット解決，リリース，ふりかえり，といった一連のプロセスが繰り返されるようになる．このような繰り返しの中で，プロジェクトにリズムが生まれ，繰り返しごとにプロセスが改善される[7]．

チケット駆動開発はこのように Trac ユーザから生まれたが，その後，Redmine ユーザにも広がり[8]，従来型の開発やアジャイル開発でも利用されている[9][10]．また，情報共有が容易でリアルタイムにメトリクスが収集可能なことから教育にも利用されている[11]．

2.2. 補完型チケット駆動開発

チケット駆動開発は，全てのタスクをチケット化する完全型チケット駆動開発と，WBS に基づく線表など従来型の管理で開発を進め，想定外の作業が発生したときに備忘録的にチケットを用いる補完型チケット駆動開発がある．

完全型チケット駆動開発は全てのタスクをチケット化するので，プロジェクトの進捗管理を一元化することができる．近年の ITS では標準あるいはプラグインでガントチャートをサポートしており，プロジェクトの管理にも用いること

が可能である. このようにすべてのタスクがチケット化され, チケットを中心にプロジェクトが運用される状況が本来のチケット駆動開発の姿であるが, 既存の標準プロセスに定められた管理方法を見直す必要があり, 導入は必ずしも容易ではない.

補完型チケット駆動開発は, 従来型の開発で想定外の問題が次第に明らかになる中で生まれた[5]. WBS に基づく線表など従来の管理手法で開発を進めるので, 標準プロセスを変更する必要がない. また, 想定外の作業が発生したときにのみ備忘録的にチケットを用いるので, 管理対象となるチケットの総数が少なくなるというメリットがある.

完全型チケット駆動開発はトップダウンに導入されることが多く, どちらかというとプロジェクト管理の方法となりがちである. これに対し, 補完型チケット駆動開発は社内的なプロジェクト管理から独立してチケットが運用されるので, 開発現場の情報共有や備忘録として活用される傾向がある[12].

組織的な改善と補完型チケット駆動開発の関係を図 2 補完型チケット駆動開発に示す. 組織的な改善方法であるフロントローディングの考え方では想定外の事象を生じさせた問題が起きないように標準プロセスを改良するのに対し, 補完型チケット駆動開発では, 想定外の事象の経験をチケットとして ITS に蓄積し, 類似プロジェクトの計画時や問題が発生した際に参照される. このような補完型チケット駆動開発のチケットは徐々に広がっていると思われるが, どのような情報が蓄積されており, どのように利用できる可能性があるかはあまり明らかにされていなかった. このような状況が, ITS すら使用しない前近代的なプロジェクト管理が残っている一つの要因であると考えられる.

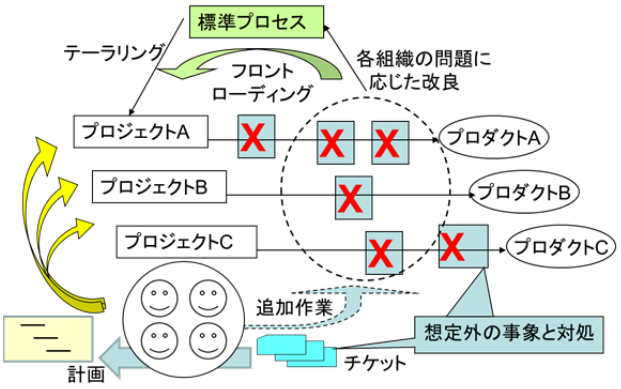


図 2 補完型チケット駆動開発

3. ケーススタディ

文献[12]に示す補完型チケット駆動開発を実施したプロジェクトのチケットを分類した. 文献[12]では補完型チケット駆動開発でプロジェクトがどのように活性化したかを報告しているが, 本研究ではチケットにどのようなノウハウが記録されているか検証することを目的とし, タスクチケットの内容に応じて分類した.

3.1. 対象プロジェクトの業務

検証の対象は Struts フレームワークをベースとする統合文教パッケージ UniVision のカスタマイズプロジェクトである. オープンなフレームワークを利用し, 高機能なソフトウェアを組み合わせることで, 少人数で様々な顧客に対応可能なシステムである. Struts フレームワークによってシステム内に同じようなコードが少なくなり, 個別の業務要件に対応する固有の開発をするだけで良いのである.

UniVision では, 対象業務ごとのサブシステムを複数組み合わせてシステムを構成できるほか, 個別要件にも柔軟に対応が可能である. オープンソースだけでなく商用 DB にも対応しており, 規模, 予算に合わせて, ハードウェアやミドルウェアの構成が選べるようになっている.

その一方で, カスタマイズ作業には固有のルールが多く, 複雑で難しい作業になりがちである. また, 少人数での部分的なカスタマイズを実施するだけで顧客専用の大規模なシステムが作れてしまうので, 工夫できる余地が少ない作業でもある. さらには, オープンソース実行環境の構成はバージョンアップによって日々変化するので, リリースまでに最新版を取り込んで動作を確認しないといけない. 一言で言うと, 複雑で大変な作業である.

3.2. プロジェクト概要

プロジェクトは最大で 8 人, 仕様変更の追加作業を含めて約 1 年間のプロジェクトであった. 全体の期間は長かったものの要件定義の期間が長い上に, 仕様がなかなか決まらず, 製造の期間が圧迫されていた.

プロジェクトメンバーのスキルは高いものの, 業務経験者が少なく, パッケージやミドルウェアの構成も UniVision としては初めて利用されるものがあるというリスクの高い開発であった. メンバーはプレッシャーの中で不安を感じ, プロジェクト全体に重苦しい雰囲気があった. 守りに入るメンバーもおり, コミュニケーションの悪いプロジェクトであった. また, プロジェクトへの参加時期や協力会社との関

係から経験の少ない者がサブリーダを担当していたことから，作業指示がうまくいかないこともあった．

システムテストの時期が近づいてくると，計画外の環境構築やリリース準備作業が必要であることが明らかになった．さらには環境に関連するバグまでが明らかになり，急激な作業の増加によってプロジェクトのコントロールが難しくなった．

このようにコミュニケーションが悪く，ゴールが見えない状況だったので，補完型チケット駆動開発を導入した．すでに ITS を障害管理に利用していたことから特に ITS の教育は行わず，「バグだけではなく，ソースを触るときや，WBS にない作業をするときは，チケットを登録してください！」と宣言して補完型チケット駆動開発を開始した．気づいた作業を備忘録的にチケットに記入し，共有することで，プロジェクト内のコミュニケーションが大きく改善した．作業量は少なくなかったが，全体の作業量が見えていたので効率的に作業を分担して何とかリリースすることができた．

3.3. 検証方法

上記プロジェクトのチケットを分類した．プロジェクトの計画時や問題発生時に参考にすることを想定して，障害チケットや未分類チケットは対象とせず，タスクチケットのみを対象として分類した．ITS のレポート機能でタスクチケットのみを抽出して一覧を作成した．分類にあたっては，類似しているチケットを順次まとめていく方法で分類した．

4. 分類結果と考察

4.1. チケットの分類結果

補完型チケット駆動開発を実施した対象プロジェクトの全 265 チケットのうち，計画以外の作業として追加されたタスク 77 チケットを分析した．チケットの内容を確認して，類似チケットごとに分類した結果， 客先業務に固有のチケット，汎用的な内容のチケットに大きく分けることができた．また，全体では7種類に分類が可能だった（表 1 チケットの分類結果）．表中にある各項目は以下の分類を示している．

客先業務に固有のチケット
- ・仕様：顧客の要望に基づく仕様変更
- ・データ：連携システムからの運用データ移行など
- ・準備：客先固有の資料作成などの準備作業

表 1 チケットの分類結果

客先業務固有のチケット			汎用的な内容のチケット			
仕様	データ	準備	基盤	テスト	管理	障害
11	25	6	17	12	3	3

汎用的な内容のチケット
- ・基盤：最新のシステムや DB などへの入れ替え
- ・環境：構築した環境依存のテストなど
- ・管理：リリース前の確認作業など追加の管理作業
- ・障害：障害に関連する分析作業

対象のプロジェクトは既存システムからの移行であり，外部システムとの連携もあったので，データに関するタスクが多かった．また，カスタマイズ元の統合文教パッケージ UniVision はオープンソースを多用したシステムであることから，パッケージやミドルウェアの更新などの基盤のタスクが多かったと考えられる．

4.2. 考察

補完型チケット駆動開発を実施したプロジェクトのチケットを分類した結果，チケットはプロジェクトの特性をあらわしており，客先業務に固有のチケットのほか，汎用的な内容のチケットが蓄積されていることがわかった．客先業務固有のチケットは，同一顧客の他システム構築時に参考になると考えられ，汎用的な内容のチケットは他の顧客に同一システムを導入する際やトラブル発生時に参考になると思われる．もし，同一システムの導入が継続するのであれば，標準プロセスにチケットの内容を一般化して組み入れると良いと考えられる．

蓄積されたチケットは，そのタイトルを見るだけでもどういった作業が必要になるかなど経験の少ない管理者・開発者には大いに参考になると思われた．その反面，技術的な TIPS は詳細情報を読まないとわからないものがあった．具体的には，デプロイ時のサービス停止時間を最短にする目的で，あらかじめ war ファイルを配置しておき，予定時刻になってから TOMCAT の再起動を実施していた．このような TIPS は仕組みを知っていれば明らかであり，詳しくは書かれていなかったと考えられる．社内教育や OJT による技術移転が必要と考えられる．

ダイナミックなプロセスの変更として分類結果を見ると，追加タスクの収集はできたが，削除や修正に関しては収集できなかった．これは，補完型チケット駆動開発は，既存の管理方法に基づくので，元々のタスクの削除は管理されないからである．また，元々のタスクを変更した際に

作業が詳細化されればチケット化される可能性はあるが，基本的には記録されないと考えられる．これらは今後のプロセスのノウハウあるいはバッドノウハウとして，標準プロセスで支援する必要がある．

もし，このプロジェクトを完全型チケット駆動開発で実施していたなら，ベースとなるプロセスがチケット化されているので，その削除や修正が記録できる可能性がある．その反面，チケットはWBSよりも粒度が細かいことが多いので，安定したプロジェクトでないと差分がとりにくい．今回のプロジェクトのように仕様が確定せずに段階的に確定する状況では，常にチケットが更新されてしまい，ベースとなるチケットと差分のチケットが区別できないからである．その点では，削除と変更の情報は得られないものの，補完型チケット駆動開発の方がノウハウの収集が容易であるといえる．

補完型チケット駆動開発のチケットは数も少なく，容易に検索ができるので，過去のプロジェクトを参考にして，リスク見積，計画，問題が発生した際の対策，などの参考にすることが可能である．プロジェクトのノウハウを個人に閉じることなく，類似プロジェクトや将来の保守の際に効果を発揮すると考えられる．

今回は補完型チケット駆動開発をシステムテスト以降で実施したが，より早いフェーズから実施した場合や，規模や対象業務が異なる場合には，チケットの比率が異なる可能性がある．しかし，プロジェクト毎に固有の問題は存在すると考えられるので，一定の効果は期待できると考えられる．

5. まとめ

標準プロセスを肥大化させないプロジェクト運営ノウハウの蓄積方法の一つとして，補完型チケット駆動開発の利用を提案した．補完型チケット駆動開発を実施したプロジェクトの追加タスクのチケットを分析した結果， 7 種類に分類が可能で，仕様変更，運用データ，準備作業など客先業務に固有の経験や，基盤の入れ替え，環境依存のテスト，追加の管理作業，障害の分析といった汎用的な経験が蓄積されており，類似プロジェクトの計画時の作業項目漏れの確認や，トラブル解決時にキーワードで類似障害を検索して対応策を検討するといった利用方法が可能なことがわかった．

今回のチケットの内容のうち，汎用的な経験は考慮漏れのチェックリストなどに一般化して標準プロセスの改良にも用いることができる．しかし，発生頻度の低い問題の対策を無秩序に追加していくと，標準プロセスはどんどん肥大化して利用が困難になってしまう．そこで，補完型チケット駆動開発を利用して必要な時に類似プロジェクトを検索すれば，実際に起きたことを具体的に確認でき，参考にすることが可能である．

プロジェクトには予見できないことが数多く存在している．Humphrey はこれを要求の不安定さと呼び，使わないとわからない未知の要求，細目が流動的な不安定な要求，実現方法の詳細が理解されていない誤解された要求，の 3 つを挙げている[14]．今回のチケット分類を要求の不安定さにあてはめると，それぞれ仕様変更，運用データ，準備作業に相当する．Humphrey は要求の不安定さの解決策としてプロトタイピングを薦めているが，Humphrey 自身が問題点を挙げているように，プロトタイピングには正解がなく，そのコントロールは難しい．

補完型チケット駆動開発で得られるタスクは標準プロセスに追加されたタスクであり，そのプロジェクトに特有のノウハウである．経験を蓄積する目的でプロトタイピングを実施しなくても，チケットからノウハウを得ることで発生しうる事象の予想がある程度可能である．また，想定外の事象が発生した際の対策の検討にも有効で，プロジェクトの運営を堅牢にするものである．しかし，効率化などを目的に削除や修正が行われた作業の記録は含まれておらず，また汎用的な TIPS は記録されにくい．標準プロセスの改良や社内教育などと組み合わせて利用すると良いと思われる．

チケット駆動開発を実施するには ITS などのチケットを管理するツールが必要になる．しかし，近年のトレンドとして知っていても，従来の標準プロセスが制約になって実施できない組織が未だに多い．このため，標準プロセスに取り込まれなかった多くのノウハウが，個人の経験として蓄積されてしまい，多くのノウハウはプロジェクトが終わるたびに忘れ去られている，今回の検証のようにチケットには多くのノウハウが含まれており，このような検証を通じて補完型チケット駆動開発を普及させていきたい．

参考文献

[1] フュジェッタ，ウルフ，“ソフトウェアプロセスのトレンド”, pp.212-213, 海文堂, 1997.
[2] 井上，松本，飯田，“ソフトウェアプロセス”, pp.209-221, 共立出版,2000.
[3] ベーム，ターナー，アジャイルと規律，p.189，日経 BP 社, 2004.
[4] Sakai, Matsumoto, Torii, “A new framework for improving software development process on small

computer systems,” International Journal of Software Engineering and Knowledge Engineering, vol.7, no.2, pp.171-184, 1997.
[5] 小川，阪井，“チケット駆動開発”，翔泳社, 2012.
[6] まちゅ，“チケット駆動開発 … ITpro Challenge のライトニングトーク (4) –”，まちゅダイアリー，http://www.machu.jp/diary/20070907.html, 2007.
[7] 小川，阪井,「チケット駆動開発- BTS で Extreme Programming を改善する-」，ソフトウェア品質シンポジウム 2009,
http://www.juse.or.jp/software/83/attachs/ippan_2-1.pdf
[8] 小川，阪井，“Redmine よるタスクマネジメント実践技法”，翔泳社, 2010.
[9] 岡，三宅，“本当に使える開発プロセス”，日経 BP 社 2012.
[10] 前川，西河 誠，細谷，“わかりやすいアジャイル開発の教科書”，ソフトバンククリエイティブ, 2013.
[11] Igaki, Fukuyasu, Saiki, Matsumoto, Kusumoto, "Quantitative Assessment with Using Ticket Driven Development for Teaching Scrum Framework," In Proceedings of the 2014 International Conference on Software Engineering(ICSE2014), 2014.
[12] 阪井，“チケット駆動開発によるプロセス改善 現場重視，管理重視，それとも情報共有重視”，SPI Japan 2013, SPI コンソーシアム, 2013.
[13] 阪井，“チケット駆動開発によるプロジェクトの活性化 ―見える化と運用ポリシーがプロジェクトを変えた”, SPI Japan 2010, SPI コンソーシアム, 2010.
[14] Humphrey，“ソフトウェアプロセス成熟度の改善”，pp.276-295，日科技連, 1991.

機械学習を用いたテキスト分類による ライセンス特定のためのルール作成プロセス支援

東　裕之輔
和歌山大学 システム工学研究科
s151039@sys.wakayama-u.ac.jp

眞鍋　雄貴
熊本大学 大学院自然科学研究科
y-manabe@cs.kumamoto-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

近年では，*OSS* を利用したソフトウェア開発が盛んに行われており，*1* つのソフトウェアに多数のライセンスが含まれている場合がある．ライセンス違反を回避するために，ソースファイル単位でのライセンス特定を行うことが必要である．最も精度の高いライセンス特定ツールである *Ninka[5]* は，未知ライセンスの存在を検出することができる．しかし，未知のライセンスが何であるかは利用者が目視により確認する必要がある．また，大量の未知ライセンスが出力される場合もある．このような未知ライセンスを特定するには，ライセンスルールを手作業で作成する必要があり，複雑で労力がかかる．本研究では，ライセンスルール作成支援を目的とし，*Debian v7.8.0* のソフトウェアパッケージを入力として *Ninka* が未知ライセンスと判定したファイル群に対して *K-means* 法を適用し，ライセンス記述の分類精度を定性的および定量的評価を行った．また，*Random Forest* 法を用いて，類似ライセンスの推定をおこなった．実験の結果，ライセンスのルール作成プロセスにおいて，機械学習を用いたテキスト分類が有用であることがわかった．

1. はじめに

生産性向上の手段として，ソフトウェアの再利用は必要不可欠なものとなっている [9]．特に近年では，オープンソースソフトウェア（以降，OSS と呼ぶ）を利用したソフトウェア開発が盛んに行われている．ソフトウェアを再利用する際には，著作者からライセンスを得ることが義務付けられている．OSS の場合，一般的には，ソースコードのコメントアウト部分に明示されている，ライセンスを遵守することで OSS の再利用が可能となる．

ただし，一つのソフトウェアが複数の異なるソフトウェアを再利用し構成される場合，複数のライセンスが，一つのソフトウェアに共存することが想定される．OSS を含めたライセンスには不整合問題というものが存在する．ライセンス不整合問題とは，異なるライセンスの条項間に矛盾した条項が存在し，両方のライセンスに違反している状態のことを指す．ライセンスに違反すると，最悪の場合，訴訟問題に発展する恐れがある．例えば，2007 年に GPL 違反で提訴され VoIP 電話機の仮販売差止め処分に至った Skype 社などの事例がある [1]．ライセンス違反の回避を目的として，ライセンスをソースファイル単位で機械的に特定する手法やツールが複数提案されている．German らの研究 [5] では，Ninka と呼ばれるライセンス特定ツールが，現時点で最も精度の高いツールであることを示している．ただし，ライセンス記述が検出された場合に，他の手法やツールは何らかの既知ライセンスを出力として返す（誤判別）するのに対して，Ninka は，未知ライセンスの存在が検出されたことを出力する．しかし，未知ライセンスそのものは，目視により確認する必要がある．

本研究では，まず，特定を行ったファイルの内，Ninka

[1] OSS ライセンスの比較および利用動向ならびに係争に関する調査:http://www.ipa.go.jp/files/000028335.pdf

が未知ライセンスと判別したソースファイルが，どの程度の割合を占めているかを予備的に調査する．調査の結果，特定を行ったソースファイルの内，未知ライセンスファイルが大量に含まれていることがわかった．すなわち，Ninka は誤検知そのものは少ないが，未知ライセンスとして判定したライセンスを多く含むことがわかった．次に，大量の未知ライセンスファイルをすべて目視により調査しライセンスを特定するには相当な労力がかかるため，未知ライセンスファイルをクラスタリング（分類）し，クラスタの代表的なファイルを目視するのみでライセンスを特定できるかどうかを調査した．本研究では，Debian v7.8.0 のソフトウェアパッケージを入力として，Ninka が判別した未知ライセンスファイル集合に対して K-means 法を適用し，ライセンス記述の分類精度を定性的および定量的評価を行った．また，Random Forest 法を用いて，類似ライセンスの推定をおこなった．実験の結果，ライセンスのルール作成プロセスにおいて，機械学習を用いたテキスト分類が有用であることがわかった．

表 1. 本論文で使用する一般的なライセンスとその略称

略語.	説明
Apachev2	Apache License version 2.0
BSD4	初期の BSD License，BSD4clause License
BSD3	BSD3clause License
BSD2	BSD2clause License
CeCill	Cea Cnrs Inria Logiciel Libre License
CDDL	Common Development and Distribution License
CPLv1.0	Common Public License version 1.0
EPLv1.0	Eclipse Public License version 1.0
FreeType	FreeType License
GPLv1	General Public License version 1
GPLv2	General Public License version 2
GPLv3	General Public License version 3
GPLnoVersion	GPL with no version indicated
LGPLv2.1	Lesser General Public License version 2.1
LGPLv3	Lesser General Public License version 3
LibraryGPLv2.0	Library General Public License version 2.0
MIT/X11	MITLicense，MIT が X11 のためにリリースしたライセンス
MITold	昔の MIT/X11
MPLv1.1	Mozilla Public License version 1.1
Python	Python License
PostgreSQL	Postgre SQL License
SameAsPerl	Perl に適応されているライセンス
SeeFile	別のファイルへの参照
ZLIB	ZLIB/libpng license
publicDomain	パブリックドメインとされているライセンス

2. 用語

本章では，本論文で用いる用語を定義する．

ソフトウェアライセンス ソフトウェアの利用許諾である．ソフトウェアの利用者は，ライセンスを遵守することで，ソフトウェアを利用することができる．ライセンスはソフトウェア全体だけではなく，ソースファイルについても決まっている．ライセンスには条項の修正を受けた，複数のバージョンが存在する．例えば，GPL には現在 3 種類のバージョンが存在するが，全て別のライセンスである．本論文では，ライセンスが複数のバージョンを持つ場合，特定のバージョンを示すために接尾辞 v を用いる．また本論文では “or later” を接尾辞+で表現する．“or later” とは，例えば，GPL のライセンス記述に “either version 2 of the License, or (at your option) any later version” と記載されている場合，version 2 以降のバージョンのうちのどれかでの再頒布が可能となる．表 1 に本論文で使用するライセンスの名前と略称を示す．

ライセンス記述 ライセンスの明示を行っている文章である．主にソースコードのコメントアウト部分にて明示される．ライセンス記述には，ライセンス全文が載っているものと，ライセンス全文への参照が記述されているものがある．

未知ライセンスファイル Ninka がライセンス特定の際，未知のライセンスと判別したソースファイルである．

未知ライセンス記述 未知ライセンスファイルのライセンス記述である．

類似ライセンス 既存ライセンスをベースに作られたライセンスであり，ライセンス記述が類似しているライセンスである．

ライセンスルール 正規表現などのルールベースによるライセンスの特定に必要な知識．

3. ライセンス特定手法

ソフトウェアが複数の異なるソフトウェアを再利用し構成される場合，複数のライセンスが，一 つのソフトウェアに共存することが想定される．ライセンスには，不整合問題が存在する．ライセンス不整合問題とは，異なるライセンスの条項間に矛盾した条項が存在し，両方のライセンスに違反している状態である．

ライセンスの不整合問題を回避するため，ソースファイル単位でのライセンス特定を行う必要があるが，全

てのソースファイルのライセンスを目視で確認するのは相当な労力がかかる．そのため，正規表現などのルールベースでソースファイルのライセンスを機械的に特定する手法やツールが複数提案されている．Ninka[5] は，ライセンス記述を複数の文に分解したライセンス文を特定し，ライセンス文で構成される既知ライセンスを特定結果として出力する．Fossology[6] は，b-SAM[2] アルゴリズムを用いて既知のライセンス記述とソースコード中のコメントを比較し，最も類似している既知ライセンスを特定結果として出力する．他にも，単純な正規表現を利用したソフトウェアライセンス特定ツールも存在する．OSLC[3] はソースコード中のコメントとライセンス記述間での最長一致列を類似度とする．次に最も類似度が高かったライセンスを特定結果として出力する．Ohcount[4] は各ライセンス記述に対応した正規表現を用いて，ソースコード中のコメントにマッチした正規表現からライセンスを特定する．

現時点で，どの手法が最も有用なのか検証するため，German ら [5] は各ツールを Debian v5.0.2 からランダムに選出した 250 のソースファイルのライセンスを各ツールで特定し，その性能の評価を F-尺度法を用いて行った．その結果，最も F 値が高かったツールは Ninka であった．ただし，ライセンス記述が検出された場合に，他の手法やツールは何らかの既知ライセンスを出力として返す（誤判別）するのに対して，Ninka は，未知ライセンスが存在が検出されたことを出力する．しかし，未知のライセンスそのものは手作業によりライセンスの確認を行わなければならない．そこで本研究では，未知ライセンスファイルに着目した分析を行う．

4. 予備調査

Ninka が特定したソースファイルの内，未知ライセンスファイルがどの程度の割合を占めているかを予備的に調査する．データセットとして，Firefox v25.0[5] のソースファイルを選んだ．対象とする言語は Java (.java), C (.c , .h), C++ (.cpp, .cxx, .cc, .hxx), Lisp (.el), Perl (.pm and .pl), Python (.py), Ruby (.rb), ShellScript (.sh), Makefile (.mk) である．これらの拡張子を条件にソースファイル集合を作成した結果，Firefox v25.0 では 22,413 ファイルとなった．表 2 に作成した Firefox v25.0 のソースファイル集合のライセンスを Ninka で特定したライセンスの内，上位 10 件のライセンスを示す．

表 2. FireFox v25.0 のライセンス（上位 10 件を表示）

Ninka 特定結果	ファイル数
NONE	1,390
spdxBSD3	959
Apachev2	752
FreeType	481
BSD3	240
MPL1_1andLGPLv2_1,MPLv1_1	204
SeeFile	143
MITX11	135
spdxBSD2	90
MITmodern	60
UNKNOWN	17,678

特定に成功したライセンスのうち，未知ライセンスファイルは 17,678 ファイル存在し，特定を行ったソースファイルの内，約 79%を占めていた．予備調査の結果，Firefox v25.0 は，大量の未知ライセンスファイルが含まれていることがわかったが，大量の未知ライセンスファイルのライセンスを目視で特定するのは相当な労力がかかる．そのため，未知ライセンスファイルのライセンスを特定できなければ，実用的とはいえない．未知ライセンスを特定するには，新しくライセンスルールを作成する必要がある．

5. ライセンスルール作成プロセスの問題点

5.1. ライセンスルール作成プロセス

Ninka のライセンスルール作成プロセスは以下の 4 つの手順から成る．

1. 未知ライセンスファイルを調査し，追加する未知ライセンス記述を決定する．
2. 未知ライセンス記述を複数のライセンス文に分解する．
3. 既知なライセンス文が存在しないか調査し，ユニークなライセンス文を決定する．

[2] http://fossology.org/symbolic_alignment_matrix
[3] http://sourceforge.net/projects/oslc/
[4] http://github.com/blackducksw/ohcount
[5] ftp://ftp.mozilla.org/pub/mozilla.org/firefox/releases/25.0/source/irefox-25.0.source.tar.bz2

4. 新規に追加するライセンス文の正規表現を作成し，ライセンスルールを追加する．

手順 1 では，ライセンス特定ツールに追加すべきライセンスを決定するため，未知ライセンスファイル集合を調査する．ここでは，未知ライセンスファイル数に対して多くを占めているライセンスがライセンス特定ツールに追加すべきライセンスである．

手順 2 では，未知ライセンス記述を複数のライセンス文に分解する．このプロセスは Ninka を用いて行う．Ninka はライセンス特定時にオプションを指定することで中間生成ファイル “ファイル名.goodsent”（以降では，goodsent ファイルと呼ぶ）を出力する．goodsent ファイルはライセンス記述を抽出し，Ninka によってライセンス文に分解されたライセンス記述が出力されているファイルである．追加すべきライセンスのソースファイルに対して Ninka を実行し，goodsent ファイルを出力させることができれば，手順 2 は完了する．

手順 3 は，手順 2 で分解したライセンス文の一部は，既知である場合がある．同じライセンス文が違う名前で登録されることになってしまい，正しいライセンス特定が行えなくなるため，ライセンス特定ツールにとって既知であるライセンス文を調査し，追加するライセンス文と，同一のライセンス文を明らかにする必要がある．Ninka の場合，ライセンス文は表記ゆれを考慮した拡張正規表現で登録されているため，単純な検索だけでは，このプロセスは行うことはできない．

最後に，手順 4 では，新しく追加するライセンス文の正規表現を作成し，ライセンスルールを追加する．

5.2. ライセンスルール作成プロセスの問題点

ライセンスルール作成プロセスにおける問題点として，ライセンスルール作成プロセスは労力がかかると考えられる．特に労力がかかる手順は，手順 1，手順 3 がある．手順 1 は，未知ライセンス集合のライセンスを目視で確認し，未知ライセンスファイル集合に存在するライセンスの内訳を明らかにする必要がある．手順 3 は，ライセンス特定ツールの知識の調査を行わなければならない．ライセンスには，ライセンス記述が酷似している類似ライセンスが多数存在する．図 1 の Python と PostgreSQL のような類似ライセンスは他にも存在するため，ライセンス記述の違いを見極めなければならないライセンスを全て探す必要がある．

（PostgreSQL の類似箇所）
Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

（Python の類似箇所）
Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

図 1. 類似ライセンス例（Python と PostgreSQL の例）

以上により，ライセンスルール作成プロセスを自動化できることが望ましい．しかし，自動化のためには，未知ライセンスファイル集合からライセンスルールに追加すべきライセンスを抽出することがまず必要になる．追加すべきライセンスを抽出する手段の 1 つとして，機械学習を用いたテキスト分類が挙げられる．テキスト分類を行うことで，類似する未知ライセンス集合を抽出することが容易になると考えられる．ただし，機械学習でのテキスト分類をライセンスに対して行った研究はまだない．ライセンスルール作成プロセス自動化の支援を行うには，まず，機械学習を用いたテキスト分類の限界を明らかにする必要がある．そこで本研究では，機械学習を用いたテキスト分類が，既存 OSS ライセンス特定手法のルール作成プロセスの支援にどれくらい有用かを明らかにすることを目的とした分析を行う．

6. 分析方法

本章では，ライセンスルール作成プロセスである手順 1，手順 3 の支援に，機械学習を用いたテキスト分類がどれくらい有用であるかを明らかにするための分析方法を説明する．

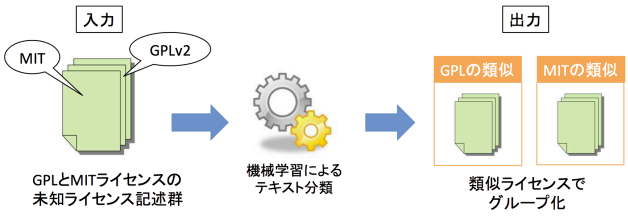


図 2. 分析方法の概要

具体的には，図 2 のように，GPL や MIT で構成されている未知ライセンス記述群を入力とし，GPL と MIT の 2 つの類似ライセンスのグループにクラスタリングすることがどれくらいできるかどうかを明らかにする．

機械学習を用いたテキスト分類を行うには，まず，未知ライセンス記述を単語文書行列 (TDM) を用いて数量化する前処理が必要である．

次に，手順 1 を支援するため，前処理後の未知ライセンス記述群に対して K-means 法によるクラスタリングを行う．クラスタリングにより，一つのライセンスから成るクラスタを生成することができれば，追加すべきライセンスを探すための調査が容易に行えると考えられるため，手順 1 を支援することができる．

次に，手順 3 を支援するため，前処理後の未知ライセンス記述に対して Random Forest 法による類似ライセンスの推定を行う．未知ライセンス記述の類似ライセンスを推定することができれば，ライセンス特定ツールの知識内に存在する既知のライセンス文を把握することが容易に行えるようになると考えられるため，手順 3 を支援することができる．

本実験で用いる K-means 法のクラスタ数，Random Forest で推定を行う類似ライセンスは，OSI が発表している主要なライセンスを参考に決定した．本実験で推定を行う類似ライセンスを表 3 に示す．OSI が発表している主要なライセンスとは，人気でかつ広く使用されているライセンス [6]13 種類と MPLv1.1 を含めた 14 種類である．MPLv1.1 を含めた理由は，MPLv1.1 はマルチライセンスを採用した古くから用いられているライセンスであり，ソフトウェア企業が，独自のライセンスを作成する際の雛形とすることが多いからである．K-means 法の最適クラスタ数は不明なため，本研究で用いる主要なライセンスに従って 14 とし，Random Forest 法の学習データは表 3 の OSS プロジェクトから，100 ファイル分のライセンス記述をコメントアウト単位で抽出した．

表 3. 推定を行う類似ライセンス及び取得先

ライセンス	ライセンス記述の取得先	取得した日付
Apachev2	OpenOffice v3.4.0	2014/11/16
BSD3	j-gitv3.5.1	2014/10/14
BSD2	OpenSSh v6.7p1	2014/11/19
GPLv1	Gcc v2.9.0	2014/11/21
GPLv2	Gcc v3.0.1	2014/11/29
GPLv3	Gcc v4.9.2	2014/11/16
LGPLv2.1	Glibc v2.20	2014/11/20
LGPLv3	OpenOffice v3.3.0	2014/10/18
LibraryGPLv2.0	Gcc v4.9.2，Glibc v2.2.0，FireFox v9.0	-
MIT	FireFox v33.0	2014/11/15
MPLv2.0	FireFox v33.0	2014/11/15
MPLv1.1	FireFox v9.0	2014/11/20
CDDLv1.0	zfs-fuse v0.6.9	2015/2/2
EPLv1.0	Eclipse MARS	2014/11/19

以降では，分析方法の各処理（前処理，クラスタリング，類似ライセンス推定）の詳細を説明する．

6.1. 前処理：TDM の構築

文章を数量化する方法の一つとして，単語文書行列 (Term-DocumentMatrix: TDM) を構築する方法がある．TDM とは全文書に現れた単語一つ一つが行，各文書を列とした行列であり，総単語数を N，総文書数を M とすると $N \times M$ の行列となる．その [i , j] 番目の要素は，単語 i が文書 j に現れる回数に対応する．次に TDM を構築するまでの過程と処理を順に説明する．

最初に改行の除去を行う，未知ライセンス記述の改行を空白に置換する．次に，未知ライセンス記述の数値の英単語化を行う．ライセンスには複数のバージョンが存在することを 2 章で述べた．このバージョンの値を学習させたいため，数値を英単語として扱う．1 → one，2 → two のように 1–9 までの数値を英単語に置換する．

次に，未知ライセンス記述から英語の文書によく出てくる単語，例えば，“i”，“me”，“my” などの人称代名詞や “and”，“but” などの接続詞などを除去するため，ストップワードの設定を行う．ストップワードには，R のテキストマイニングパッケージ [7] で利用でき

[6] http://opensource.org/licenses/category

[7] http://cran.r-project.org/web/packages/tm/tm.pdf

る $stopwords(kind = "en")$ で表示される単語 174 語を設定する．最後に，R のテキストマイニングパッケージで利用できる関数 $TermDocumentMatrix$ と設定したストップワードを用いて TDM を構築する．ただし，Random Forest 法においては，実験に使用した計算機環境 [8] では，計算時間が膨大となったため，学習データと未知ライセンスファイルから構築した TDM から，1 回のみ出現した単語を削除し，再構築した TDM を用いる．

6.2. K-means 法によるテキスト分類

未知ライセンス記述群のテキスト分類に，教師なし機械学習アルゴリズムである K-means 法 [8] を用いる．テキスト分類に K-means を用いた理由は，未知ライセンス記述に対して機械学習を用いたテキスト分類を行った例がなく，クラスタリング結果が予想できないことから，複雑なクラスタリングアルゴリズムを使用すると結果の考察が困難になると考えたからである．本研究で用いる K-means 法は，クラスタ数を 14，類似度はユークリッド距離とした．

6.3. Random Forest 法による類似ライセンス推定

クラスタの類似ライセンスの推定として Random Forest 法 [2] を用いる．テキスト判別手法に Random Forest を用いた理由は，多値分類アルゴリズムの中で精度が最も高いことで知られているためである．クラスタの類似ライセンスは，各クラスタの未知ライセンス記述の類似ライセンス推定結果を多数決することにより決定する．Random Forest 法により，未知ライセンス記述群から構築した TDM を，表 3 から取得した学習用のライセンス記述から構築した TDM を学習し，類似しているライセンスを出力する．

7. 実験

本実験は，機械学習を用いたテキスト分類がライセンスルール作成プロセスの支援にどれくらい有用であるかどうかを明らかにすることが目的である．6 章で述べた分析方法を用いて，OSS ライセンスルール作成支援プロセス手順 1，手順 3 にどれくらい有用かどうかを明らかにするための実験を行う．本実験では Debian v7.8.0 の各ソフトウェアパッケージから 1 つのソースファイルをランダムに選び，Ninka でライセンスを特定できなかった未知ライセンス記述群をデータセットとして抽出する．

7.1. 評価基準

本研究では特に労力がかかるプロセスは，手順 1, 手順 3 であると述べた．以下の 2 点が確認できれば，手順 1 を支援できるとする．

- 評価基準 A. ライセンス記述が存在しないクラスタは存在するか
- 評価基準 B. 一つのライセンスで 75%以上を占めているクラスタは存在するか

評価基準 A は，ライセンス記述が存在しないクラスタが存在すれば，明らかに調査の対象外な未知ライセンス記述を除外できる可能性があるということである．評価基準 B は，一つのライセンスで 75%以上を占めているクラスタが存在すれば，クラスタの選別などを行うことで，容易に追加すべきライセンスを抽出できそうであると考えられる．評価基準 B の判定を行うとき，未知ライセンス記述数が 10 以下のクラスタは除外する．評価基準 A,B を満たしていれば，テキスト分類が OSS ライセンス特定ツールのライセンスルール作成プロセス支援に有効であるとする．

次に，以下を確認できれば，手順 3 を支援できるとする．

- 評価基準 C. 目視により特定したクラスタのライセンスの類似ライセンスであったクラスタが 50%以上存在したか

Random Forest で推定した，クラスタの類似ライセンスが，目視により特定したクラスタのライセンスの類似ライセンスであったクラスタが 50%以上存在すれば，類似ライセンス推定を全く行わない状況と比べると，違いを見極める必要のある類似ライセンスを探すのに有用であるとする．なお，類似ライセンスかどうかは，ライセンス記述と一致している文章が存在すれば類似ライセンスとする．

7.2. データセット

本実験では Linux ディストリビューションである Debian v7.8.0 のソフトウェアパッケージのソースファイル [9]

[8] OS:OS X Yosemite, CPU:2.3GHz Intel Core i7, メモリ:16GB

[9] http://ftp.riken.jp/pub/Linux/debian/debian-cd/7.8.0/source/iso-dvd/

の未知ライセンス記述群をデータセットとして抽出した．Debian には 48,559[10] のソフトウェアパッケージを含んでいるため，多数のライセンスを含んでいる．本実験では “.zip”， “.tar.gz” ， “.tar.xz” ， “.tar.bz2” をソースコードアーカイブと見なし解凍した．ソースファイルの拡張子は，Gonzalez ら [7] によって報告された Debian でよく使われているプログラミング言語 8 種類のうち，シェルスクリプトと PHP を除く，6 種類のプログラミング言語の拡張子，Java (.java), C (.c), C++ (.cpp , .cxx , .cc), Lisp (.el ， .jl), Perl (.pm , .pl), Python (.py) を選んだ．これらのプログラミング言語で書かれている 12,725 パッケージから各 1 ファイルをランダムで抽出し，Ninka でライセンス特定を行った結果，ソースファイル 12,725 ファイルのうち未知ライセンスファイルは 2,838 ファイルであった．この未知ライセンスファイル集合から goodsent ファイルに書かれている文章を未知ライセンス記述群として取得しデータセットとする．

7.3. 実験結果

7.3.1 K-means のクラスタリング結果

未知ライセンス記述群を 14 のクラスタに分類した．クラスタの未知ライセンスを目視で特定し，各クラスタで最も多かったライセンスのうち上位 3 つと，クラスタのデータ数をまとめたものを表 4 に示す．表 4 において，ライセンス記述が無かったクラスタは，クラスタ 1，9，13 の 3 クラスタで確認できた．一つのライセンスで 75%以上を占めているクラスタは，クラスタ 4，7，12 の 3 つのクラスタで確認できた．特に，クラスタ 7 に関しては，ZLIB が 95%以上を占めている．以上から，K-means 法によるクラスタリングは，評価基準 A，B を満たしているため，ライセンスルール作成プロセスに有用であるとする．

7.3.2 Random Forest 法の類似ライセンス推定結果

Random Forest 法による各クラスタの類似ライセンス推定結果を表 5 に示す．各クラスタのライセンスは，各クラスタの未知ライセンス記述のライセンス特定，あるいは推定した結果から多数決を行い決定する．目視により特定したクラスタのライセンスと Random Forest 法により推定したライセンスが類似していた場合，推定に成功したとし，クラスタのライセンスが Random Forest 法が推定可能なライセンスではなかった場合とクラスタのファイル数が 10 以下の場合は推定不可とし評価には含めない．推定が可能であった 6 クラスタのうち，4 クラスタは類似ライセンスの推定に成功した．その精度は 4/6 = 約 67%（小数点第 2 位四捨五入）である．クラスタ 12 に関しては Random Forest 法により推定したクラスタの類似ライセンスは，目視で確認したクラスタの類似ライセンスと一致している．

次に，類似ライセンスが一致しているクラスタ 12 以外の類似ライセンスの推定が成功したものに関して類似箇所を示す．

LGPLv3+ と GPLv2+の類似箇所

LGPLv3 と GPLv2 は大部分が同一の記述であり，異なる箇所は，バージョンの数字の違いと，“GNU General Public License” と “GNU Lesser General Public License” の違いの 2 箇所のみである．

MIT と ZLIB の類似箇所

MIT と ZLIB の類似箇所はいくつか存在する．MIT の免責条項 “THE SOFTWARE IS PROVIDED ‘AS IS’, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,” と ZLIB の冒頭にある免責条項 “This software ‘as-is’, without any express or implied warranty.” の箇所では，大文字表記と小文字表記の点に違いがあるが，文章は類似している．もう一つは MIT の冒頭にある “Permission is hereby granted” と ZLIB の “Permission is granted” である．

BSD2 と BSD3 の類似箇所

表 4. K-means 法によるクラスタリング結果

クラスタ	目視によるライセンスの内訳	データ数
1	NONE	23
2	GFDL ライセンス全文	1
3	NONE，GPL，GPLv2	382
4	SameAsPerl（248/312），GPL，GPLv2	312
5	Apachev2，Cecil，LGPLv2.1+	55
6	GPLv2+，GPL，LGPLv2.1+	278
7	Zlib（41/46），SOFA，Other	43
8	BSD3，BSD2，BSD-style	85
9	NONE	2
10	CC の全文	1
11	CPL のライセンス全文	1
12	MIT（105/118），BSD4	118
13	NONE	1
14	Copyright，SeeFile，ライセンス名のみ	1536

[10] https://packages.debian.org/stable/allpackages?format=txt.gz

表 5. Random Forest 法による類似ライセンス推定

クラスタ	目視の結果	正解集合	推定結果	推定の成否
1	NONE	なし	MIT	-
2	GFDL	GFDL	MPLv1.1	-
3	NONE	なし	MIT	-
4	SameAsPerl	GPLv1，GPLv2，GPLv3，LGPLv2.1，LGPLv3	MIT	失敗
5	Apachev2	Apachev2，BSD2，BSD3	MIT	失敗
6	GPLv2+	GPLv1，GPLv2，GPLv3，LGPLv2.1，LGPLv3	LGPLv3	成功
7	ZLIB	MIT	MIT	成功
8	BSD3	Apachev2，BSD2，BSD3	BSD2	成功
9	NONE	なし	MIT	-
10	CC	CC	BSD2	-
11	CPLv0.5	なし	BSD2	-
12	MIT	MIT	MIT	成功
13	NONE	なし	MIT	-
14	Copyright	なし	MIT	-

BSD2 と BSD3 は大部分が同一の記述であり，この 2 つのライセンス記述の異なる箇所は “3.Neither〜” 条項が存在するか否かのみである．

以上により，評価基準 C を満たしているので，Random Forest 法を用いた類似ライセンスの推定は，ライセンスルール作成プロセス支援に有用ではないかと考えられる．

8. 考察

8.1. ライセンスルール作成プロセス支援の可能性

データセットのクラスタリングを行った結果，1 つのライセンスで 75%以上を占めているクラスタが 3 つのクラスタで確認できた．特にクラスタ 7 においては，ZLIB ライセンスが 95%以上を占めていた．しかし，各クラスタのライセンスの内訳は，クラスタ内の未知ライセンス記述を目視で確認しなければ，把握することができない．各クラスタのライセンスの内訳を把握するには，同一ライセンスであるライセンス記述の定義を行い，ライセンス特定ツールに追加すべきライセンスを各クラスタから抽出することができれば，ライセンスルール作成プロセスの手順 1 を支援することが可能である．また，クラスタ 14 に 1536 の未知ライセンス記述がクラスタリングされた．クラスタ 14 は，Copyright のみや，ライセンス名のみなど，比較的短い文章が多数存在した．ライセンス名のみの記述については，ライセンス名が未知であった場合，ライセンスルールに追加するなど，未知のライセンス名の抽出にも用いることができる．

各クラスタの類似ライセンスの推定を行った結果，約 67%の精度で，類似ライセンスを推定することができた．本実験では，推定を行うライセンスを 14 種類に設定し，類似ライセンスの推定を行った．しかし，未知ライセンスファイル集合には，14 種類以外のライセンスが多数含まれていることが考えられる．推定を行うライセンスの種類を増やすことで，精度の向上が期待できる．また，ライセンス記述がないクラスタが確認できたことから，ソースコードを学習し，ライセンス記述がないクラスタの推定が可能であると，ライセンスルール作成プロセスの手順 1 の支援においても有用であると言える．

8.2. 類似ライセンスの推定結果

本実験で行った類似ライセンスの推定では，Random Forest 法の予測結果は MIT とされることが多かった．クラスタ単位では 14 クラスタ中 9 クラスタに，2,838 の未知ライセンス記述の内，2,415 ファイルが MIT ライセンスと推定された．MIT は類似ライセンスが多いことでも知られている．Fedora Project はソースファイル上で見つかった 36 種類の MIT の類似ライセンスを報告している [11]．また MIT ライセンスには，BSD，Apachev2，ZLIB などにも記述される免責条項が存在する．以上のことから MIT ライセンスの類似ライセンスであると推定されるライセンスが多いのではないかと考えられる．

[11] https://fedoraproject.org/wiki/Licensing:MIT

8.3. 企業におけるライセンス特定ツールの利用シナリオ

ライセンス特定ツールは，特定精度が 100%でない限り，企業の法務部門などではすべてのライセンスを目視で確認する必要があり，ライセンス特定ツールの利用自体の意義が不明確な場合がある．

我々は，現時点でライセンス特定ツールを利用せず目視ですべてのライセンスを確認している企業にも，ライセンス特定ツールを利用することに意味があると考えている．すべてのライセンスを目視で確認する場合，本論文で紹介したように軽微な表記ゆれを見落とす可能性があり，目視であっても特定精度が 100%になるとは限らない．そのため現状では，2 名以上の担当者でダブルチェックするなどして対応していると思われる．

ライセンス特定ツールによる特定結果は，人為的ミスを防ぐためのダブルチェックとしても用いることができると我々は考えている．ライセンス特定ツールと目視による特定結果に差異があれば，いずれかの方法に誤りがある可能性が高い．特に，本論文の実験で用いた Debian のように，膨大な数のライセンスを確認しなければならない状況においては，目視による確認にもミスが増えると考えられる．ライセンス特定ツールをダブルチェックの手段として利用することで，作業コストの大きな削減にもつながると期待できる．

8.4. 制約

本研究では，OSI の主張する主要なライセンスと MPLv1.1 を含めた 14 ライセンスのみを用いて分析を行った．そのため本研究の実験結果は一般性が高いとは言えない．K-means 法によるテキスト分類時のクラスタ数を，それぞれのプロジェクトのライセンス分布に基づいたクラスタ数にするとクラスタリングや Random Forest 法の類似ライセンス推定の精度向上が期待できる．

本研究では，支援すべきライセンス特定ツールのルール作成支援プロセスとして Ninka のみを選んでいる．そのため本研究の実験結果は一般性が高いとはいえない．特に，Ninka が未知ライセンスを判別する仕組みに着目しているため，他のライセンス特定ツールのルール作成支援プロセスと異なる部分が生じる場合がある．しかし，どのようなライセンス特定ツールもライセンス記述を調査する部分は共通しているはずなので，本実験結果が部分的には当てはまると考えられる．

9. 関連研究

9.1. ライセンス特定に関する研究

本論文でも紹介したライセンス特定に関する研究は，他にも行われている．例えば，Tunnanen ら [10] は，ライセンスを指定する記述と適合する正規表現を作成し，どの正規表現に適合するかによりライセンスを特定する方法を提案している．ただし，特定されていないソースファイルについては，利用者自身がライセンスの正規表現を作成しなければならない．眞鍋ら [11] は，ライセンス特定支援に関する研究を行っている．ライセンス記述には共通した記述が用いられることに着目し，頻出文字列を提示することで作成すべきライセンス記述のための正規表現を作成支援する手法を提案している．本研究では，K-means 法による単語の出現頻度による類似度に基づいたテキスト分類を行っている部分で，眞鍋らのアプローチとは異なっている．

9.2. ライセンス不整合問題に関する研究

ライセンス不整合問題を解決するために，Agawal ら [1] は，ライセンスの条項をタプルで表現したライセンスメタモデルを構築し，ArchStudio4 上で義務の衝突，利用できる権利を計算する手法を提案している．また，German ら [4] は，ライセンス不整合を起こした場合，ライセンス不整合をどのように解消するか，ライセンサーとライセンシの観点から解決方法をまとめている．これらの研究と本研究は，ライセンス不整合問題を問題意識として持っているが，本研究では，ライセンス特定ツールにおけるライセンスルール作成支援を行うことで，ライセンス不整合問題を解決しようとしている．

9.3. ライセンスとソフトウェア開発者に関する研究

ソフトウェア開発者のライセンスに対する認識を分析した研究がいくつか行われている．例えば，Colazo ら [3] は，コピーレフト性のある OSS ライセンスを用いる OSS プロジェクトはコピーレフト性のない OSS ライセンスの OSS プロジェクトと比べて，開発者の地位やコーディング量，活動の永続性が高く，開発期間が短くなることを示している．Sojer ら [9] は，ソフトウェア開発者にソフトウェアの再利用に関するインタビューとライセンスに関するクイズを実施したところ，65%のソフトウェア開発者は，ソフトウェアの再利用に関する知識を，

主にインターネットのような非公式な情報を元に勉強していることがわかった．また，ライセンスに関するクイズから開発者自身のライセンスに関する知識を過剰評価していることがわかった．この結果から，効果的なソフトウェア再利用に関する勉強方法が確立されていないことを示した．これらの分析結果を踏まえ，本研究では，ソフトウェア開発者のライセンスに関する知識が不十分であることを前提とし，ソフトウェア開発者が容易にライセンスルールの作成を行えるようにするための支援を最終目標にしている．

10. おわりに

本研究では，機械学習を用いたテキスト分類がルール作成支援にどれくらい有用かどうかを明らかにすることを目的として，K-means 法によるテキスト分類，Random Forest 法による類似ライセンスの推定を Debian v7.8.0 のソースファイルから抽出した未知ライセンス記述群に対して行い結果を分析した．本研究ではルール作成支援プロセスを 4 つに分割して定義し，手順 1 の追加すべきライセンス記述の調査，手順 3 の既知ライセンス文の調査を支援対象とした．ライセンス記述を K-mean 法を行うと追加すべきライセンスルールを容易に発見できる可能性があることを述べた．Random Forest 法でクラスタの類似ライセンスの推定を行うと，約 67%の精度で類似ライセンスを推定できることを述べた．

今後の課題としては，実用的な手法を提案するため，分類するクラスタ数や，推定を行う類似ライセンスの種類を増やすことで，より細分化が可能なクラスタリングや，より精度の高い類似ライセンス推定を可能にすること．また，同一ライセンスであるライセンス記述の定義を行い，ライセンス特定ツールに追加すべきライセンスを各クラスタから抽出可能にすることなどが挙げられる．

参考文献

[1] Thomas A. Alspaugh, Hazeline U. Asuncion, and Walt Scacchi. Analyzing software licenses in open architecture software systems. In *Proc. of ICSE '09 Workshop on Emerging Trends in FLOSS '09*, pp. 54–57, May 2009.

[2] Leo Breiman. Random forests. *Machine Learning*, Vol. 45, No. 1, pp. 5–32, Oct. 2001.

[3] Jorge Colazo and Yulin Fang. Impact of license choice on open source software development activity. *American Society for Information Science and Technology*, Vol. 60, No. 5, pp. 997–1011, May 2009.

[4] Daniel M. German and Ahmed E. Hassan. License integration patterns: Addressing license mismatches in component-based development. In *Proc. of ICSE '09*, pp. 188–198, May 2009.

[5] Daniel M. German, Yuki Manabe, and Katsuro Inoue. A sentence-matching method for automatic license identification of source code files. In *Proc. of ASE '10*, pp. 437–446, Sep. 2010.

[6] Robert Gobeille. The fossology project. In *Proc. of MSR'08*, pp. 47–50, May 2008.

[7] Jesus M. Gonzalez-Barahona, Gregorio Robles, Martin Michlmayr, Juan José Amor, and Daniel M. German. Macro-level software evolution: A case study of a large software compilation. *Empirical Software Engineering*, Vol. 14, No. 3, pp. 262–285, June 2009.

[8] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Royal Statistical Society*, Vol. 28, No. 1, pp. 100–108, 1979.

[9] Manuel Sojer and Joachim Henkel. License risks from ad hoc reuse of code from the internet. *Communications of the ACM*, Vol. 54, No. 12, pp. 74–81, Dec. 2011.

[10] Timo Tuunanen, Jussi Koskinen, and Tommi Krkkinen. Retrieving open source software licenses. In *OSS '06*, pp. 35–46, June 2006.

[11] 眞鍋雄貴, 市井誠, 早瀬康裕, 松下誠, 井上克郎. コメント中の頻出文字列を用いたソフトウェアライセンスの特定支援. ソフトウェア工学の基礎 XIV, 日本ソフトウェア科学会 FOSE2007, pp. 105–114, Nov. 2007.

バイトコードの機械学習に基づく不具合予測手法の提案

藤原 剛史
京都工芸繊維大学 工芸科学部 情報工学課程
m5622042@edu.kit.ac.jp

水野 修
京都工芸繊維大学 大学院工芸科学研究科 情報工学部門
o-mizuno@kit.ac.jp

要旨

不具合を含んでいそうなモジュール（*Fault-prone* モジュール）の予測には，複雑度メトリクスに基づいたモデルが用いられることが多い．しかし，それらのモデルを構築するには，メトリクスを測定するための環境整備やツールへの慣れが必要であり，現場への適用は簡単ではない．そこで，メトリクス測定を行わない *Fault-prone* モジュール検出手法として，「*Fault-prone* フィルタリング」というものが提唱されている．この手法は，スパムフィルタリングの理論を用いたものであり，ソースコードへの簡単な適用のみによって *Fault-prone* モジュールを検知できる．

本研究では，*Fault-prone* フィルタリングによる *Fault-prone* モジュール検出のより高い効果を得ることを目的として，*Fault-prone* フィルタリングをバイトコードへ適用した場合とソースコードへ適用した場合の比較実験を行う．具体的には，対象とするプロジェクトのバイトコードおよびソースコードから単語を抽出し，スパムフィルタに通して結果を得たのち，比較を行う，

本研究ではこの実験を通して，バイトコードによる不具合予測が従来の不具合予測と比べて同等以上の精度を得ることが可能であることを示した．

1. はじめに

今やソフトウェアの果たす社会的な役割は大きく，ソフトウェアの品質を高く保つことはソフトウェア工学における重要な目標である．しかし実際には，時間的・人的な理由が障害となって，それらの品質を高く保持するのは難しく，より効率的かつ簡易な手法が求められている．コードを作成した時点で不具合を含んでいそうなモジュール（Fault-prone モジュール）の検出が可能であることは，レビューやデバッグに費やすコストの削減につながる．そのため，これまでも Fault-prone モジュールを予測すべく，多くの研究が行われてきた [1][2][3][4][5][6][7]．従来の手法では，主にモジュールの複雑さや変更頻度等のソフトウェアメトリクスを用いて予測モデルを構築している．しかし，こうしたソフトウェアメトリクスを測定するためには，メトリクスの測定環境が必要であり，現場への適用を難しくしている一因となっている．

こうした状況の中，メトリクス等の測定の必要なく，ソースコードのみを入力として Fault-prone モジュールの予測が可能である「Fault-prone フィルタリング」と呼ばれる手法が水野らによって提唱されている [8][9]．この手法は，迷惑メール検出に用いられるスパムフィルタリングの技術をソフトウェアのソースコードに対して適用し，純粋にコードのテキスト情報のみから Fault-prone モジュールを予測する．そして，適用が簡便であるにも関わらず，従来の手法と比較しても劣らない程の高い予測精度が示されている手法である．

先に示した Fault-prone フィルタリングの研究では，ソースコードを入力として Fault-prone モジュールの予測が行われているが，実は Fault-prone フィルタリングに用いる入力は，テキスト情報であればソースコードでなくてもよい．そこで，本研究では，この Fault-prone フィルタリングをバイトコードへ適用する．バイトコードを用いる理由としては，ソースコードに比べ開発者のコードの書き方等に依存しにくく，精度が保証されれば有用であるだろう、という考えのもとに採用した．そして，ソースコードによる予測と比べてどのように精度が違うのかを，ソースコードへ適用した場合との比較実験を通じて調べる．

本報告書の構成を以下に示す．２章では，本研究を行

う目的について述べる．３章では本実験への準備として，前提知識として必要な事項を説明する．４章では，本実験の対象となるプロジェクトおよび本実験の準備の手順を述べるとともに，実験の方法について説明する．５章では，予測精度の評価尺度について言及し，本実験の結果を述べる．６章では，本実験の結果を基に考察を行う．そして７章では，本研究のまとめと今後の課題について述べる．

2. 研究の目的

これまで行われてきた Fault-prone フィルタリングの研究においては，ソースコードを入力とした不具合予測で高い成果を収めてきた．しかし，Fault-prone フィルタリングの入力はテキスト情報であればソースコードである必要はない．何か別のテキスト情報を入力とすることによって，より良い結果が得られる可能性がある．

そこで本研究では，Fault-prone フィルタリングの入力をバイトコードにすることによって不具合予測を行い結果を得る．バイトコードを用いる理由としては，バイトコードは最低限の命令列で記述されているため，開発者のコードの書き方等に依存しにくいので，精度が保証されれば有用であることが挙げられる．また，ソースコードを入力とした不具合予測との比較を行い，どのような違いが生じるのかを調べる．

3. 準備

本章では実験を行う前に，実験の前提となる事項を紹介する．

3.1. Fault-prone フィルタリング

迷惑メール検出に用いられるスパムフィルタで利用される技術を，ソフトウェアのソースコードに対して適用し，純粋にコードのテキスト情報のみから Fault-prone モジュールを予測する手法である．

3.1.1 スパムフィルタ

現在使用されている一般的なスパムフィルタは，その仕組みにベイズの定理を利用しているものが多い．これらのスパムフィルタでは受信したメールの特徴を学習し，新たにメールを受信した際に，それらの学習に従ってメールを本人に有用なメール (ham) と迷惑なメール (spam) に分類するといった仕組みを取っている．基本的なスパムフィルタの動作は，メールを学習するステップと，学習した結果を用いてメールが ham メールか spam メールかを分類するステップの２ステップで構成されている．

3.1.2 CRM114

Yerazunis らによって開発されたスパムフィルタであり [10] ，Fault-prone フィルタリングに使用されている．主にスパムフィルタとして開発されているが，汎用的な用途，例えば計算機のログ監視やネットワークのトラフィック監視等にも活用できるとされている．また，現在開発されているメールフィルタの中でも高い予測精度をあげているものの１つである．

CRM114 は基本的にはベイズ識別を利用したテキスト分類フィルタであるが，複数の単語を組み合わせたものをトークンと呼び，学習・分類の単位として利用することが大きな特徴である．従来のテキスト分類フィルタは１単語をトークンとしているのに対し複数単語の組をトークンとすることで，より複雑な学習が可能となっている．

本研究では，CRM114 のデフォルトの分類手法である ”Othogonal Sparce Bigrams Marcovmodel (OSB)”を使用する．OSB は任意の連続する５単語の組み合わせのうち，２単語からなるものだけをトークンとする手法である．

3.2. PROMISE

ソフトウェア工学のためのデータセットを公開しているウェブサイトである．PROMISE 内で公開されているデータセットの中には，不具合情報や，提案手法との比較に利用することができる，

3.3. Lex

Lex は，レキシカルアナライザ（字句解析プログラム）を生成するプログラムである．テキスト中の文字列の変換，カウント，抽出などさまざまな目的に使われ，その応用領域は，コンパイラやコンバータの作成を筆頭に，自然言語処理や簡単な整形まで幅広い．

本研究では，バイトコードおよびソースコードの単語抽出に使用する．

3.4. 誤判定時のみ学習 (TOE)

これは，スパムフィルタにおいても利用される方式であり，対象の分類を行い，その分類結果が実際の結果と異なっていたと判断した時点でのみ学習を行う手法である．この手法を用いることで，実際の環境に近い状況，すなわち事前の知識が全くない状態からの予測モデル構築が可能となる．

本研究では，実際は Fault-prone であるバイトコードおよびソースコードで示されたモジュールを Non-Fault-prone であると判断したとき，または実際は Non-Fault-prone であるモジュールを Fault-prone であると判断したときのみに学習する．

4. 実験

本章では，実験の対象・準備・方法について説明する．

本実験では，Fault-prone フィルタリングの入力をバイトコードとすることで，Fault-prone モジュールの予測を行う．また，ソースコードを用いた場合の予測との変化を確認するため，入力をソースコードとした Fault-prone モジュール予測との比較を行う．

以下，この実験においてはクラスをモジュール単位として扱う．

4.1. 実験対象

本実験では，次に示すオープンソースプロジェクトを対象とする．

- **Apatch Ant**

  Apache Ant は，GNU make の Java 版ともいえるものであり，OS など特定の環境に依存しにくいビルドツールである．記述言語は Java である．

  今回の実験では rev.1.5，rev.1.6，rev1.7 の３つのリビジョンを対象とする．

4.2. 実験の準備

実験を行う前にまず, ”THE APACHE ANT PROJECT”[1] から Apache Ant のバイトコードとソースコードをリビジョン毎に用意する．また，Apache Ant に関するリビジョン毎のデータセットを PROMISE から取得し，以降の操作を行う．

4.2.1 jar ファイル展開

用意したバイトコードは jar（Java アーカイブ）形式で圧縮されているため，クラスファイルに展開する必要がある．jar 形式のファイルを展開する手段は JDK （Java Development Kit）で提供されており, jar コマンドの -x オプションと-f オプションを使用することで，バイトコードが記述されたクラスファイルを取得することが可能である．

4.2.2 逆アセンブル

クラスファイルの状態だと学習に利用できないため，オペコードを取得する必要がある．しかし, バイナリデータからオペコードを抽出するのは困難であるため，クラスファイルに逆アセンブル[2] を行う．クラスファイルを逆アセンブルする手段は JDK で提供されており，javap コマンドの -c オプションをクラスファイルに適用するだけで，アセンブリコードの命令列（以下アセンブリコードと呼ぶ）を得ることが可能である．

4.2.3 ソースファイル分割

１つの Java ソースファイル（以下ソースファイルと呼ぶ）には複数のクラスが定義されていることがある．クラスを１つのモジュールとして扱うと，１つのソースファイルに複数のモジュールが存在することとなり，実験を行うにあたって不都合である．よって，１つのソースファイルには１つのモジュール（クラス）のみが存在するようにソースファイルを分割する．

4.2.4 タグ付け

PROMISE のデータセットに準拠して，各モジュールが Fault-prone であるか Non-Fault-prone であるかを決定する．本実験において Fault-prone なモジュールとは，バグを少なくとも１つ含んでいるモジュールであると定義

[1] http://ant.apache.org/git.html

[2] 実行可能なバイナリデータを人間が可読なアセンブリ言語に変換すること

し，Non-Fault-prone なモジュールとは，バグを１つも含んでいないモジュールであると定義する．

タグ付けの結果, Fault-prone なモジュールが 291, Non-Fault-prone なモジュールが 1,096, 合計 1,387 のモジュールが得られた．

4.2.5 トークナイザ作成

Fault-prone フィルタリングを行うにあたって，テキスト情報を単語に分割するトークナイザが必要となる．本実験では，Lex を用いて作成した字句解析プログラムをもってトークナイザとする．具体的には，正規表現を用いて，アセンブリコードからニーモニック [3] を抽出するトークナイザとソースコードから単語 [4] を抽出するトークナイザの２つを作成する．

4.3. 実験方法 1

本実験では，Fault-prone フィルタリングを適用するにあたって，誤判定時のみ学習（TOE）を採用する．TOE を用いた Fault-prone フィルタリングの具体的な手順を以下に示す．

(1) 当該プロジェクトのモジュール群を古いリビジョンのものから順にソートする．ただし，同リビジョン内のモジュール群の順に関しては，ランダムに決定する．

(2) 並び替えたモジュール群から１つ取り出して適当なトークナイザを適用し，Fault-prone であるか Non-Fault-prone であるかをスパムフィルタにより予測する

(3) モジュールの予測が完了した時点で，その結果とタグを比較し，予測が正しければ何もせずに (2) へ戻る．

(4) 予測が正しくなければ，正しい結果を学習させ，その後 (2) へ戻る．

以上の (1)〜(4) の手順を, アセンブリコードおよびソースコードで記述された，それぞれのモジュール群に対して適用することをもって, 実験の方法とする．このとき，モジュールが Fault-prone であるか Non-Fault-prone であるかの予測を決定するための確率の閾値は，0.5 に設定している．

[3] オペコードと１対１対応している人間が可読な文字列

[4] ここで述べる「単語」とは，変数名，演算子，キーワード等のソースコードを構成する要素のことである．ただし，本研究では区切り文字を示す” ; ”は含まない．

4.4. 実験方法 2

より具体的な比較をするため，アセンブリコードおよびソースコードで記述されたそれぞれのモジュール群に対して，閾値を動かして Fault-prone モジュールを予測し，結果を評価する．予測の手順に関しては，先に示した TOE を用いた Fault-prone フィルタリングの手順 (1)〜(4) を用いる．また，閾値は 0.05 から 0.95 まで 0.05 刻みで動かして実験を行う．このとき，TOE という手法は予測の順への依存が大きいため，閾値を変えて測定し直す際に，最初にランダムに決定した同リビジョン内のモジュール群の順を変えないように注意する．

5. 実験結果

本章では，実験によって得られた結果を示す．

5.1. 実験の条件

本実験に行う上での条件を次に示す．

5.1.1 スパムフィルタ

本実験では，スパムフィルタとして CRM114 を使用する．また分類の手法として，CRM114 のデフォルトの”Othogonal Sparce Bigrams Marcovmodel（OSB）”を使用する．

5.1.2 閾値（Threshold）

対象となるモジュールに CRM114 を適用すると，0〜1 の値が得られる．この値はベイズ識別によって得られた，そのモジュールが Fault-prone である確率である．閾値は，そのモジュールが Fault-prone か Non-Fault-prone かの予測を決定するための境界値であり，Fault-prone である確率が閾値より大きければ Fault-prone と予測され，閾値以下であれば Non-Fault-prone と予測されるということである．

当実験においては，実験方法１では閾値を 0.5 に設定し，実験方法２では閾値を 0.05〜0.95 間を 0.05 刻みで動かしながら実験を行う．

5.2. 評価尺度

表 1 は本実験で得られる結果の凡例である．N_1，N_2，N_3，N_4 は横に示す予測と縦に示す実測にそれぞれ該当する例数を表す．またこの表 1 では便宜上，Fault-prone を FP，Non-Fault-prone を NFP と省略する．本実験で得られた結果の評価尺度として以下の，精度（Accuracy），適合率（Precision），再現率（Recall）の３つの指標を使用する．

表 1. 実験結果の凡例

		予測	
		FP	NFP
実測	FP	N_1	N_2
	NFP	N_3	N_4

5.2.1 精度（Accuracy）

精度は，全モジュールのうち，予測が正しかった割合を示す．よって，精度は以下のように定義される．

$$Accuracy = \frac{N_1 + N_4}{N_1 + N_2 + N_3 + N_4} \tag{1}$$

精度は，予測の全体的な傾向を把握するには便利であるが，実測値の偏り等に大きく影響を受ける指標である．そのため，この値のみで予測の良さを判断するのは危険である．

5.2.2 適合率（Precision）

適合率は，予測が Fault-prone であるモジュールのうち，実測が Fault-prone であったものの割合を示す．よって，適合率は以下のように定義される．

$$Precision = \frac{N_1}{N_1 + N_3} \tag{2}$$

適合率は，直感的には１つの不具合を見つけるのにどのくらい無駄なモジュールを調べる必要があるかを示す指標である，すなわち，テストのためのコストを示している．

5.2.3 再現率（Recall）

再現率は，実測が Fault-prone である全てのモジュールのうち，正しく Fault-prone と予測できたものの割合を示す．よって，再現率は以下のように定義される．

$$Recall = \frac{N_1}{N_1 + N_2} \tag{3}$$

再現率は，直感的には予測によって実際の不具合をどれだけ網羅できるかを示す指標である．そのため，Fault-prone モジュール予測にあっては，不具合を未然に防ぐという観点から最も重視すべき指標といえる．

5.3. 実験結果１

[**4.3　実験方法１**] によって得られた Fault-prone フィルタリングの経過のグラフを図 1，図 2 に，最終的な予測結果を表 2，表 3 に示す．

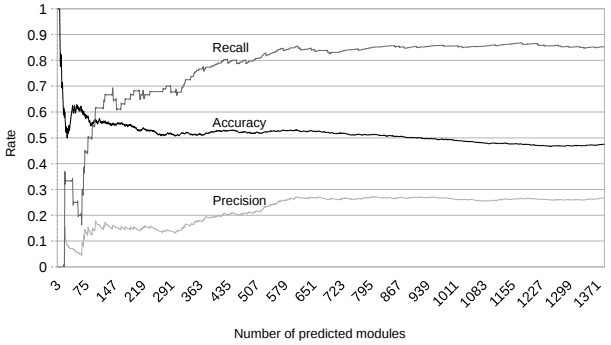


図 1. バイトコード入力時における各指標の推移

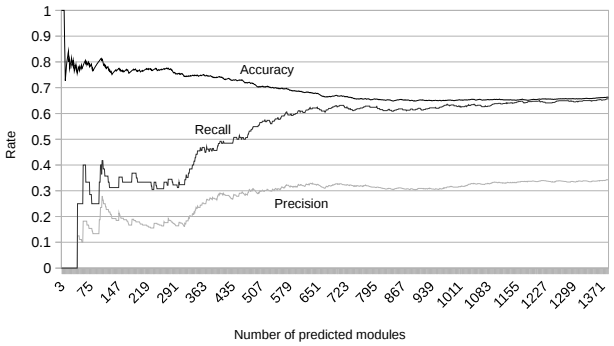


図 2. ソースコード入力時における各指標の推移

表 2. バイトコードモジュールの最終予測結果

		予測	
		FP	NFP
実測	FP	248	43
	NFP	686	410
精度（accuracy）		0.474	
適合率（precision）		0.266	
再現率（recall）		0.852	

表 3. ソースコードモジュールの最終予測結果

		予測	
		FP	NFP
実測	FP	192	99
	NFP	388	708
精度（accuracy）		0.649	
適合率（precision）		0.331	
再現率（recall）		0.660	

図 1，図 2 のグラフについて説明する．図 1 のグラフは，バイトコードモジュールに Fault-prone フィルタリングを適用したときの各指標の推移経過を，また図 2 のグラフはソースコードモジュールに Fault-prone フィルタリングを適用したときの各指標の推移経過を示している．TOE では無学習の状態から 1 つずつモジュールを予測していき，その予測結果が間違っていればその時点で学習を行うため，予測し終えたモジュールの数によって予測精度が変わる．これらのグラフは，第 n 個目までのモジュールの予測が完了した時点での 3 つの指標の値をプロットしたものである．このグラフの縦軸は精度（Accuracy），適合率（Precision），再現率（Recall）の各指標の値を表す．また，横軸は精度，適合率，再現率の各指標を算出した時点での，既に予測が完了しているモジュールの数を表している．

次に，表 2，表 3 について説明する．これらの表は，対象としたプロジェクトの全てのモジュールを予測し終わった時点での予測と実測をクロス集計したものである．特に，表 2 はバイトコードモジュールを予測したときの予測結果であり，表 3 はソースコードモジュールを予測したときの予測結果である．

これらの結果からは，バイトコードモジュールの Fault-prone フィルタリングによる予測は，ソースコードモジュールの予測に比べて精度，適合率が低いことが確認できる．特に適合率は 0.266 と，1 つの Fault-prone モジュールを見つけるのに余分なモジュールを 3 つも調べなければならないことを示している．しかし再現率は，ソースコードモジュールのものと比べても 0.852 と著しく高く，これはおよそ 85 %の Fault-prone モジュールを検出できていることを意味する．

5.4. 実験結果 2

[**4.4　実験方法 2**] によって得られた閾値を動かして行った Fault-prone フィルタリングの結果のグラフを図 3，図 4 に示す．

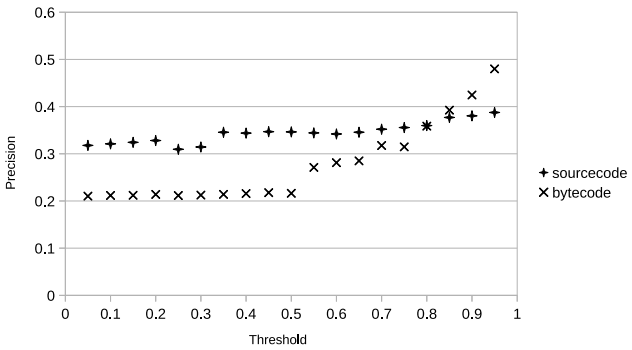


図 3. 閾値の変化における適合率の推移

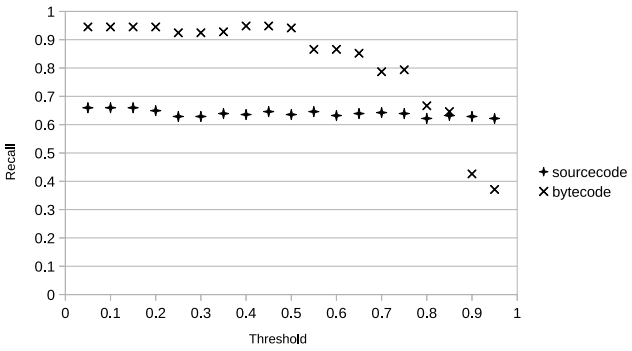


図 4. 閾値の変化における再現率の推移

図 3，図 4 のグラフについて説明する．このグラフは，閾値（Threshold）を変化させた場合の Fault-prone フィルタリングの適合率（Precision）と再現率（Recall）の推移について示している．図 3 のグラフの縦軸は適合率の値，図 4 のグラフの縦軸は再現率の値を表し，これらのグラフの横軸は閾値を表している．

ここで注意すべきは図 4 のグラフにおいて，閾値の増加に伴い再現率の値が増加することがあることである．通常，すべてのデータを学習する場合は閾値の増加によって再現率が上がることはない．しかし，本実験で

は TOE という手法を使っており，閾値を変えることによって予測が変われば，今までの閾値で学習されていたモジュールが学習されなくなる，また逆に学習されていなかったモジュールが学習されるようになるということが起こりうる．よって，ある程度の再現率の揺れは十分にあり得るものであると考えられる．

図 3，図 4 では，閾値を 0.8 あるいは 0.85 としたときにバイトコードモジュールの予測結果による適合率と再現率双方の値がソースコードモジュールの予測結果の値を上回っている，あるいは同等であることが確認できる．このことからバイトコードモジュールによる予測精度は，閾値を変えることによってソースコードモジュールの予測精度を上回ることが可能である．

6. 考察

本章では，実験に対する考察を行う

6.1. 妥当性の検証

本実験で得られた結果の妥当性を検証する．

- **対象プロジェクトの不足**

  本実験では，Apache Ant を対象のプロジェクトとして特定の結果を得た．しかし，対象としたプロジェクトが 1 つしかなく，本実験で示した結果が他の多くのプロジェクトに対して，一般的に当てはまるとは限らない．また，Apache Ant はオープンソースプロジェクトであり，商業プロジェクトに今回の手法を適用しても同等の結果が得られる確証はない．

- **Perl スクリプトの不具合**

  本実験を行うにあたって，当環境では複数の処理を一括して行うために，Perl スクリプトを作成して実験を行っている．得られた実験結果に関してはある程度の根拠があり，目下のところ特別な問題は見受けられないが，作成した Perl スクリプトに何らかの不備が存在して実験の処理の手順を誤っている可能性がある．

- **トークナイザの不具合**

  本実験において，Lex を用いて作成した字句解析プログラムをもってトークナイザとしたが，この字句解析プログラムに不具合がある可能性がある．いくつかのモジュールデータに対して適用して確認をしてみてはいるが，人手で確認するにはデータの量が膨大なためすべてのデータに対して確認ができておらず，字句解析プログラムが不具合なく動いているという確証はない．

- **モジュール群の適用順**

  本実験を行うにあたって，同リビジョン内のモジュール群の順をランダムにソートした．これは，ダウンロードしてきたバイトコードおよびソースコードからは詳細な作成日時が不明であったためである．今回実験に用いた手法である TOE においては適用順によって学習内容が変わるので，最悪の場合と最良の場合とで大きな差が出る可能性がある．

6.2. 実験結果の考察

まず，実験方法 1 の結果を考察する，図 1，図 2，表 2，表 3 はバイトコードモジュールおよびソースコードモジュールの fault-prone フィルタリングを用いた不具合予測における，精度，適合率，再現率の各指標の推移と最終結果を示したものである．図 1 と表 2 からはバイトコードモジュールの不具合予測の結果を知ることができる．バイトコードモジュールの不具合予測は精度と適合率はあまり高くないが再現率だけは非常に高い，このことの理由がこれらの図表から確認できる．その理由とは，バイトコードモジュールの不具合予測ではモジュールの多数を Fault-prone であると予測しているということである．特に，実測が Non-Fault-prone なモジュールでは 1,096 モジュール中 686 ものモジュールが Fault-prone と誤判定されていることがわかる．しかし一方，Fault-prone モジュールは 291 モジュール中 248 ものモジュールの予測に成功しており，図 1 のグラフからもわかるように，TOE による学習は一定の成果を上げている．このようなことが起こる理由として考えられるのは，バイトコードモジュールでは Fault-prone なモジュールは Fault-prone であると識別できる何か決定的な特徴を持ちながら，その他の特徴は Non-Fault-prone なモジュールと似通っている，そして，Non-Fault-prone なモジュールは Non-Fault-prone であると識別できる決定的な特徴を持っていないということである．一方，図 2 と表 3 からはソースコードモジュールの不具合予測の結果を知ることができる．これらの図表からは，ソースコードモジュールの不具合予測は精度と再現性が比較的高いこ

とが確認できる．バイトコードモジュールの予測と反して，$N_1 : N_2$ と $N_4 : N_3$ がともにおよそ 2 : 1 の比になっていることからも，バランスの良い安定した学習と予測が行われていると考えられる．

次に，実験方法２の結果を考察する．図 3，図 4 は確率の閾値の変化における適合率と再現率の推移を示している．図 3，図 4 を見てまず注意すべきことは，バイトコードモジュールの不具合予測では閾値を変えると適合率と再現率が大きく変化するのに対し，ソースコードモジュールの不具合予測では閾値を変えても再現率がほぼ変わらず，適合率もさほど変わらないことである．この結果が示すことは，バイトコードモジュールの不具合予測では，Fault-prone であるか Non-Fault-prone であるか明確でないモジュールが多いのに対して，ソースコードモジュールの不具合予測ではモジュールが Fault-prone であるか Non-Fault-prone であるかが明確であるということである．特に，閾値を上げても再現率がほぼ変化していないことから，Fault-prone であると予測されたモジュールのほぼすべてにおいて Fault-prone である確率が１に近いことを表している．これは，バイトコードから抽出された単語（ニーモニック）は種類が少なく，ユニークなトークンが乏しくなる一方で，ソースコードから抽出された単語は，変数名や記号が含まれることから種類も多く，トークンがユニークになりやすいことに起因すると考えられる．つまり，バイトコードモジュールでは特徴が似たようなものが多くなり，Fault-prone である確率が中程の値を取りやすいのに対して，ソフトコードモジュールでは特徴がユニークであるので，多くのモジュールにおいて Fault-prone である確率は 0 か 1 に近く，閾値を変えても予測がほとんど変わらないものであると予想される．

実験方法１，実験方法２の結果を通して述べた通り，バイトコードモジュールの不具合予測においては Fault-prone である確率がどっち付かずとなることが多く，実践的な適用に向いてないように思われるかもしれない．しかし，図 1 と表 2 が示すように，閾値が 0.5 のときに高い再現率を持つことから決定的な識別要因が存在しているものと考えられる．それを裏付けるように，バイトコードモジュールの不具合予測では適切な閾値を設定してやれば，ソースコードモジュールの不具合予測の精度を上回る可能性があることが図 3，図 4 からわかる．さらに言うならば，実は適合率と再現率はトレードオフの関係にあるので，閾値の変化によって適合率と再現率の大きく変わるバイトコードモジュールの不具合予測は状況に合わせた使用が可能であると考える．

今回の実験は単一のプロジェクト内の複数リビジョンで行っているためこのような結果が出たが，複数のプロジェクト間で不具合予測を行った場合，バイトコードモジュールによる不具合予測が勝るものと予想できる．根拠としては，別プロジェクトにおいては開発者が違えば文脈も違い，また変数名や関数名も違うためユニークな情報が役に立ちにくいと考えるからである．一方，バイトコードは余分な情報を排除した純粋な命令列であるため，違うプロジェクト間においても機能すると考えられる．もしこの考えが成り立つならば，あるプロジェクトで蓄積した学習情報を違うプロジェクトの初期のリビジョンでも適用することも可能となり，さらに適用も比較的簡単であるので，今後への大きな貢献となることが期待できる．

7. 結言

本研究では，スパムフィルタ CRM114 を用いた Fault-prone フィルタリングの手法を用いて，バイトコードモジュールの不具合予測を行う実験を行った．また，ソースコードモジュールに対しても同様の不具合予測を行い，バイトコードモジュールの予測結果と比較することで予測の精度の違いに関して調べた．実験の結果，バイトコードモジュールを Fault-prone フィルタリングに適用して不具合予測を行うことによって，従来の予測をわずかに上回る，もしくは同等な精度を得ることが可能であることが示された．

今後の課題としては，より多くのプロジェクトに対して実験を行い，バイトコードモジュールによる不具合予測が他のプロジェクトにおいても高い精度を得ることができるかを調べることが考えられる．また，プロジェクト間に対しても実験を行うことで，あるプロジェクトのバイトコードモジュールの学習結果を他プロジェクトに適用できるかどうかを調べることを今後行うべき課題としたい．

参考文献

[1] P. Bellini and I. Bruno and P. Nesi and D. Rogai “Comparing Fault-Proneness Estimation Models”, *Proc. of 10th IEEE International Conference on Engineering of Complex Computer Systems*, pp. 205–214, 2005.

[2] Lionel C. Briand and Walcelio L. Melo and Jurgen Wust “Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects”, *IEEE Trans. on Software Engineering, Vol. 28, No. 7*, pp. 706–720, 2002.

[3] Giovanni Denaro and Mauro Pezze “An Empirical Evaluation of Fault-Proneness Models’, *Proc. of 24th International Conference on Software Engineering*, pp. 241–251, 2002.

[4] Lan Guo and Bojan Cukic and Harshinder Singh “Predicting Fault Prone Modules by the Dempster-Shafer Belief Networks”, *Proc. of 18st International Conference on Automated Software Engineering*, pp. 249–252, 2003.

[5] Taghi M. Khoshgoftaar and Naeem Seliya “Comparative Assessment of Software Quality Classification Techniques: An Empirical Study”, *Empirical Software Engineering*, pp. 229–257, 2004.

[6] Tim Menzies and Jeremy Greenwald and Art Frank “Data Mining Static Code Attributes to Learn Defect Predictors”, *IEEE Transactions on Software Engineering*, pp. 2–13, 2007.

[7] Naeem Seliya and Taghi M. Khoshgoftaar and Shi Zhong “Analyzing Software Quality with Limited Fault-Proneness Defect Data”, *Proc. of 9th IEEE International Symposium on High-Assurance Systems Engineering*, pp. 89–98, 2005.

[8] Osamu Mizuno and Shiro Ikami and Shuya Nakaichi and Tohru Kikuno “Fault-Prone Filtering: Detection of Fault-Prone Modules Using Spam Filtering Technique”, *Proc. of 1st International Symposium on Empirical Software Engineering and Measurement*, 2007.

[9] Osamu Mizuno and Tohru Kikuno “Training on Errors Experiment to Detect Fault-Prone Software Modules by Spam Filter”, *Proc. of 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the founda- tions of software engineering*, pp. 405–414, 2007.

[10] S. Chhabra and William S. Yerazunis and C. Siefkes “Spam filtering using a Markov random field model with variable weighting schemas”, *Proc. of 4th IEEE International Conference on Data Mining*, pp. 347–350, 2004.

Blocking Bugの発生原因を理解するための依存関係分析

金城 清史
株式会社サイバーリンクス
k-kinjo@cyber-l.co.jp

山谷 陽亮
和歌山大学 システム工学研究科
s151049@sys.wakayama-u.ac.jp

松本 明
和歌山大学 システム工学研究科
s151045@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

影響の大きい不具合の 1 つとして *Blocking Bug* がある．*Blocking Bug* とは他の不具合の修正を妨害する不具合のことである．先行研究では *Blocking Bug* の特徴を明らかにしている．しかし，発生の原因を分析したものではない．本論文では *Blocking Bug* の早期発見，発生の予防を目的として，*Blocking Bug* の実態把握と発生原因を理解するための分析をおこなった．分析の結果，*Blocking Bug* は通常の不具合に比べて修正のために変更するファイルが多いことや，多くのファイルに参照されていると *Blocking Bug* の発生の原因になることを明らかにした．

1. はじめに

ソフトウェア開発では，ソフトウェア保守に要するコストがソフトウェアライフサイクル全体に要するコストの大半を占めることが知られている [1, 2]．そのため，ソフトウェア保守の改善支援を目的とする研究が盛んにおこなわれてきた．特に，Mining Software Repositories (MSR) 分野では，主に不具合管理システムに蓄積された不具合データの解析結果に基づいて，不具合修正および不具合管理プロセスを支援する手法が提案されている．例えば，不具合修正タスクの最適割り当てを支援する手法 [3] や不具合修正タスクの優先順位や重要度を推定する手法 [4, 5] ，重複して報告された不具合を検出する手法 [6, 7] などが挙げられる．

先行研究の多くは不具合修正作業および管理プロセスの効率化を目的とするものの，個々の不具合の特徴や影響範囲を十分に考慮したものではない．しかし，1 つの不具合がシステム全体の挙動を不安定にさせたり，場合によっては致命的なエラーにつながることもあるため，重大な問題を招く可能性のある不具合を考慮することが重要である．

これまでの反省もあり，特に最近では，個々の不具合がユーザの満足度や開発スケジュールの変更などに与える影響の大きさを考慮する研究が増えている．保守プロセスに充てられるコストやリソースには限りがあり，全ての不具合を同等に扱うのではなく，High Impact Bug[8] と呼ばれる影響度の大きな不具合 [9, 10, 11] を優先的に修正することの必要性が認識され始めたためである．

High Impact Bug の内，本研究では特に，Blocking Bug[12] を対象にする．Blocking Bug とは，他の不具合修正を妨害する不具合である．Blocking Bug が修正されない限り，関連する他の不具合 (Blocked Bug) も修正することができないため，Blocking Bug は優先的に修正されるべき不具合であるとされている．

Garcia ら [12] の研究では，新たに報告された不具合が Blocking Bug かどうかを予測するモデルを提案しているが，Blocking Bug が生じる原因や予防する方法については述べられていない．そのため，本研究では，不具合報告の依存関係，モジュールの依存関係，開発者の依存関係といった Blocking Bug の発生原因となり得る依存関係を分析し，Blocking Bug の早期修正と予防を支援するための知見を得ることを目的とする．

2. 修正活動における Blocking Bug の問題点

2.1. Blocking Bug 定義

Blocking Bug とは，他の不具合修正を妨害する不具合である．例えば図 1 に示すように不具合 A および不具合 B が存在する状況を考える．不具合 A は不具合 B を修正しないと

不具合管理システムに報告された不具合が Blocking Bug であることが判明した場合，Blocking Bug と不具合修正を妨害されている側の不具合 (Blocked Bug) との間に関連付けがおこなわれる．この際，不具合管理システムでは，二つの不具合間の関連付けのために，Blocking Bug の内容が記述された不具合報告には “blocks” というタグが付与される．同様に，Blocked Bug の内容が記述された不具合報告には “is blocked by” というタグが付与される．なお，タグ付けは自動的におこなわれるのではなく，不具合管理システムの管理者らによっておこなわれる．

多くの不具合管理システムには，不具合修正に際して依存関係のある不具合同士を関連付けて管理するための機能が備わっている．本研究では，依存関係のある Blocking Bug と Blocked Bug のことを Blocking Set と呼ぶこととする．また，ソフトウェアの不具合は，複数のファイルが関連しあって 1 つの不具合の症状として現れる場合や，1 つのファイルが原因となって複数の不具合の症状を引き起こす場合があり得るが，本研究では，1 件の不具合報告に記録された症状を 1 つの不具合として取り扱う．したがって，本論文中で特に断りなく Blocking Bug という用語を使う場合は，不具合報告に記載された Blocking Bug の症状，あるいは，不具合報告そのものを指すこととする．

2.2. 問題点

Blocking Bug を予測するための Garcia ら [12] の研究では，いくつかのオープンソースプロジェクトを対象とした予備調査から，Blocking Bug は Blocking Bug ではない不具合（非 Blocking Bug）よりも多くの修正時間を必要とすることが示されている．リリース直前やプロジェクトの繁忙期であったり，緊急な対応を要する不具合修正をおこなっているときなど，十分な時間が確保できない場合に Blocking Bug が発見されると，ソフトウェアの品質を改善（あるいは維持）することができない可能性がある．これら Blocking Bug の問題を，図 1 を用いて整理する．

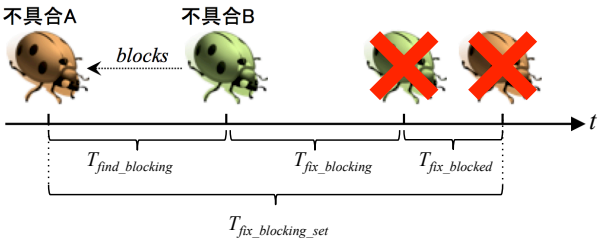


図 1. 不具合修正を阻害する Blocking Bug

報告された不具合 A は，不具合 B によって修正を妨害されており，不具合 B が先に修正されない限り不具合 A を修正することができない状態にある．つまり，不具合 A が甚大な影響を与える不具合であっても，不具合 B が修正されない限り不具合 A を修正することができない．したがって，不具合 A を修正するには，本来必要な時間 ($T_{fix\_blocked}$) よりも多くの時間 ($T_{fix\_blocking\_set}$) が必要となる．不具合修正の遅延は顧客満足度の低下に繋がるため，できるだけ早く Blocking Bug を検出する必要がある．

しかしながら，不具合 A の担当者は不具合 B を修正するためのコードやモジュールについて熟知しているとは限らず，不具合 B，すなわち，Blocking Bug の検出自体に時間 ($T_{find\_blocking}$) がかかってしまう場合もある．また，Blocking Bug である不具合 B を検出できたとしても，不具合 A と不具合 B の担当者が異なる場合は互いに情報交換しながら修正作業をおこなう必要があるため，通常の不具合よりも多くの時間 ($T_{fix\_blocking}$) を必要とする．実際，6 つのオープンソースプロジェクトを対象とした Garcia らの研究 [12] では，Blocking Bug は，非 Blocking Bug (Blocking Bug 以外の不具合）に比べ 15–40 日程度の修正時間が長くなるとされている．なお，Blocking Bug が検出されるまでの時間 ($T_{find\_blocking}$) は短い場合もあれば長い場合もあり (3–18 日)，プロジェクト毎に傾向は異なる．プロダクトの複雑度やコンポーネント間の結合度に関連があるように思われる．Garcia らの分析結果を受けて，Blocking Bug の発生原因を理解することが Blocking Bug を早期に検出したり予防したりするのにより重要であると考えるに至った．

3. 先行研究

3.1. Blocking Bug 予測

Garcia らの研究では Blocking Bug に関する特徴を分析し，新たに報告された不具合が Blocking Bug か否かを予測するモデルを構築している [12]．6 つの OSS プロジェクト（Chromium, Eclipse, FreeDesktop, Mozilla, NetBeans, OpenOffice）を対象におこなった分析では，Blocking Bug は非 Blocking Bug と比較して 2 倍から 3 倍ほど修正時間が長くなる（15 日から 40 日程度修正時間が長くなる）ことや，プロジェクトによって差はあるものの，検出にかかる時間は 3 日から 18 日程度であると述べている．

また，Blocking Bug かどうかを予測するモデルでもっとも高く寄与したメトリクスは不具合票に投稿されたコメントであることを明らかにしている（Mozilla のみ，その不具合票に関わりを持つことを望んだユーザ (CCList) の数がもっとも寄与している）．

先行研究の分析は，Blocking Bug の検出を目的としており，Blocking Bug の発生原因や発生の予防法については述べられていない．

本研究の目的は Blocking Bug の早期修正，発生の予防をおこなうことである．そのために Blocking Bug が発生する原因を理解する必要があり，発生原因を理解するための分析をおこなう点で先行研究と異なる．

3.2. 依存関係分析

ソフトウェア保守における依存関係の分析は盛んにおこなわれている．Biazzini ら [13] は， 87 プロジェクトを対象として，分散管理システムにおけるコミット履歴を基に共同開発における特徴を分析している．分析の結果，開発者の人数が多くなると複雑なブランチを形成することやブランチごとのコミット回数と開発者の共同作業には関係がないことを示している．

Bhattacharya ら [14] は，11 プロジェクトを対象にソースコードを基に作成したグラフと開発者を基に作成したグラフを用いて，ソフトウェアの開発支援のための分析をおこなっている．分析の結果，ソースコードの参照関係を基に作成したトポロジグラフのうち，入次数が多い，つまり多くのファイルから参照されているほど，そのファイルから発生するバグの重要度が高いことや，不具合の担当者とその不具合を修正できずに新たな担当者として不具合を担当した開発者間にエッジを結び作成したトポロジグラフにおいて，トポロジの変化が大きいほど不具合が発生しやすいことが確認された．

いずれもモジュール間でのみの分析や人のみの分析であり，本研究のように，不具合，モジュール，人を結びつけて 1 セットで分析した研究はまだされていない．

4. Blocking Bug の発生原因理解のための依存関係分析

4.1. 概要

プロジェクトに報告される不具合のうち Blocking Bug に特化した研究は，Garcia らの研究 [12] 以外には存在しない．また，Garcia らの研究は，Blocking Bug の予測を目的としたものであり，Blocking Bug の実態や発生原因を明らかにはしていない．

そこで本研究ではまず，Blocking Bug の実態を調査し，その後，Blocking Bug の発生原因理解のための分析を行う．Blocking Bug の発生原因理解のための分析にあたり，本研究では 3 つの分析視点を導入する．

不具合管理システムに報告された不具合は，まず適任の修正担当者（多くの場合，修正対象物の作者）に割り当てられる．次に修正担当者は，その不具合を発生させている原因を調査し，修正対象物（パッケージ，コンポーネント，ファイルなど）を決定する．この不具合解決プロセスは，Blocking Bug を対象にした分析においては非常に複雑なものとなる．複雑な関係は，修正すべき不具合が単独で発生したものであれば（Blocking Bug でなければ）発生しない．この点からも Blocking Bug の解決は通常の不具合修正に比べて容易なものでないことが分かる．したがって，分析の切り口としてまず，以下の 3 つの視点で複雑な依存関係を分解する．

- 不具合報告間の依存関係（Blocking Bug と Blocked Bug の依存関係）
- 修正担当者間の依存関係（修正作業を進める上での依存関係）
- 修正対象物間の依存関係（call-caller 関係などの依存関係）

これらの依存関係をそれぞれ分析することで，Blocking Bug の発生原因をある程度絞り込むことができ，Blocking Bug の予防する上での知見が得られるものと期待している．さらに，補足的な分析視点として，それぞれの分析観点の重畳的な分析を行う．

Blocking Bug が特定の開発者に割り当てられている状況や特定の修正対象物が Blocking Bug を引き起こす可能性などについて調査することで，Blocking Bug の発生原因をより正確に捉えることができると期待される．

4.2. Blocking Bug の実態把握のための分析

- 不具合報告のメトリクス

1. **不具合の数**：Blocking Bug と非 Blocking Bug の不具合票の数を調査する．
2. **修正時間**:Blocking Bug，非 Blocking Bug，Blocked Bug の修正時間を調査する．本研究では不具合票の Status が “Resolved” または “Closed” のものを解決済みの不具合として扱うことにし，不具合が報告された日時から解決した日時までを修正時間と定義する．
3. **Blocking Bug の検出時間**：Blocking Bug が検出されるまでの時間を調査する．本研究では不具合が報告された日時から “blocks” のタグがつけられるまでの日時を Blocking Bug の検出時間と定義する．
4. **優先度別の不具合票の内訳**優先度別に Blocking Bug と非 Blocking Bug の不具合票の数を調査する．
5. **各 Blocking Set の構成不具合数**：各 Blocking Set が持つ不具合票の数を調査する．本研究では Blocking Set に含まれる Blocking Bug と Blocked Bug の合計を Blocking Set を構成している不具合票の数とする．
6. **Blocking Set の発生順序**：Blocking Bug と Blocked Bug のどちらが先に報告されたのかを調査する．Blocking Set の中でもっとも早く報告されたものが Blocking Bug である場合 Blocking Bug が先に報告されたものとし，1 つでも Blocked Bug が先に報告されていれば Blocked Bug が先に報告されたものとしてカウントする．

- 修正担当者のメトリクス

1. **各 Blocking Set の担当者の数**：各 Blocking Set に含まれるそれぞれの不具合の修正をおこなった修正担当者の数を調査する．
2. **修正担当者の経験**：Blocking Bug を埋め込んだ開発者の経験がどの程度であったかを調査する．開発者の経験はコミットの回数から測ることにする．まず，開発者を経験の浅い開発者と豊かな開発者に分けるために対象期間中の各開発者の総コミット回数を求める．対象期間中のコミット回数が第一四分位数以下である時期は「経験が浅い開発者」とし，第一四分位数以上である時期は「経験が豊かな開発者」であると定義する．

- 修正対象物のメトリクス：

1. **修正ファイル数**：Blocking Bug と非 Blocking Bug の修正ファイル数を調査する．本研究では不具合修正の際に変更されたファイルを修正ファイルとみなす．
2. **各 Blocking Set の修正ファイル数**：各 Blocking Set に含まれる不具合を修正するために変更したファイルの総数を調査する．

4.3. Blocking Bug の発生理解のための依存関係分析

本節では Blocking Bug の発生原因を調査するための取り組む RQ について述べる．不具合報告間の依存関係の観点から RQ1 を立てる．修正担当者の依存関係の観点からは RQ2 を，修正対象物の依存関係の観点から RQ3，分析視点間の依存関係の観点から RQ4 を立てる．

RQ1–1：優先度が高い Blocked Bug の発生によって Blocking Bug の修正は活発になるか

動機：報告されたすべての不具合をリリースまでに修正することは困難である．そのため各不具合には優先度が付与される．優先度は不具合が報告された時点で決定される（途中で変更することも可能）．不具合単体としては優先度が高いない不具合が Blocking Bug である場合や，Blocking Bug が重要でないとしても Blocked Bug が重要な可能性もある．単体では重要でないと判断され，修正作業を後回しにされた Blocking Bug によって，早期に修正しなければならない Blocked Bug の修正に取り掛かれないという状況が発生してしまう．今まで，修正作業が後回しにされていた Blocking Bug が重要であ

ることが確認されれば，不具合の優先度の割り当てをより適切なものにすることができる．結果，Blocking Bug の早期修正に役立つと考えられる．

アプローチ: 本研究では Blocked Bug の優先度が高く，修正作業が後回しにされた Blocking Bug が何件存在するかを確かめる．本研究では不具合票の Priority が “Blocker” または “Critical” のものを優先度が高い不具合とする．また，修正が後回しにされたかの判断については，不具合報告後，Blocking Bug の検出時間の中央値より長い時間，コメントやパッチ投稿を不具合管理システムにおこなわなかった不具合票を修正作業を後回しにされたと定義する．

RQ1–2 : 優先度が高い Blocked Bug の発生が Blocking Bug の発見に寄与するか

動機： RQ1–1 で述べた仮説では優先度の高い Blocked Bug の修正に取り掛かかることで Blocking Bug の存在が判明するケースも考えられる．RQ1–1 と合わせて，優先度の高い Blocked Bug の修正が Blocking Bug に発見に寄与するかどうかを RQ1–2 として調査する．

アプローチ： 優先度が高い不具合の定義は RQ1–1 と同様とする．報告された不具合票には不具合に関してのコメントや依存関係にある不具合の情報を追加登録（変更）することができる．Blocking Bug の不具合票において最初の追加登録（変更）が “blocks” タグをつける（Blocking Bug であると定める）ことであった場合を，優先度が高い Blocked Bug が Blocking Bug の発見に寄与したと考える．

RQ2 : 修正担当者が多いほうが Blocking Bug が発生しやすいか

動機 : OSS 開発現場では不特定多数の人々が開発に参加している．ファイルの変更が他のファイルへ影響する場合，修正担当者が複数人いることで変更されたファイルからの影響をよく理解しないまま修正をおこなう可能性がある．結果として Blocking Bug が発生する頻度が高まる．修正担当者が多いほうが Blocking Bug が発生しやすいということが確認できれば，不具合の修正担当者を割り当てる際に関連がありそうな不具合は 1 セットと考えて，1 人の担当者に任せることで Blocking Bug の発生を予防することが可能であると考えられる．

アプローチ： RQ2 に答えるために，修正担当者数別に Blocking Bug の件数を示す．修正担当者は Blocking Set の不具合の修正をおこなった開発者とする．

RQ3 : ファイルが参照されているほど Blocking Bug が発生しやすいか

動機 : 大規模なソフトウェアは複数のソフトウェア部品によって構成されている．複数のファイルを参照し，組み合わせることでソフトウェアを構築している．これはファイルごとに機能をまとめることでファイル管理の観点からは有効であると言える．しかし，ファイルを参照していることで予期しない不具合が発生することがある．例えば，機能追加等で他のファイルから参照されているファイルを変更するとする．このとき，ファイルの変更によって他のファイルに与える影響を十分に考慮する必要があるが．このように，ファイルの参照関係が多いほど同時に変更すべきファイルが多くなり，不具合が発生する可能性も高くなる．ファイルの被参照数が多いほど Blocking Bug になりやすいことが確認されれば，被参照数を減らすことで Blocking Bug の発生を予防することができる．

アプローチ： RQ3 に答えるために，Blocking Bug と非 Blocking Bug のそれぞれのファイルが参照されているファイル数を比較する．比較には U 検定を用いる．参照されている状態について定義する．本研究ではインポートや継承によって他のファイルから利用されているとき，ファイルが「参照されている」と定義する.「参照されている」状態かを分析するためにはソースコード解析ツールである Understand[1] を用いる．

RQ4 : 経験が豊富な開発者によって Blocking Bug の発生を予防できるか

動機 : ファイルの変更権限が与えられた開発者（コミッタ）のコードやプロジェクトに対する理解は開発者ごとに異なる．経験が浅い開発者は経験が豊かな開発者に比べてファイルの依存関係の理解が浅い．ファイルの依存関係によって発生するような Blocking Bug は経験が浅い開発者のほうが発生させやすいと考えられる．経験の豊かな開発者がいる場合，こうした Blocking Bug の発生を予防することができる可能性がある．経験の豊かな開発者が Blocking Bug の発生を予防することが確認できれば，Blocking Bug が発生する可能性の高いファイルは経験豊富な開発者にチェック依頼することで発生を予防できると考えられる．

アプローチ : RQ4 に答えるためのアプローチについて述べる．経験豊かな開発者がいることで Blocking Bug の発生を予防できるかどうかを確かめるために，チームを

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

作成する．Blocking Bug が埋め込まれたファイルに変更を加えた経験をもつ開発者群をチームとする．開発者の経験については 4.2 節の不具合担当者間のメトリクスで述べた定義を使うことにする．RQ4 に答えるために，経験豊かな開発者のみ，経験が豊かな開発者と浅い開発者の混合，経験の浅い開発者のみの 3 つのグループに分けて，各チームが発生させた Blocking Bug の件数を比較することにする．

5. ケーススタディ

5.1. 対象プロジェクト

本論文では，Apache Hadoop プロジェクトをケーススタディの対象とする．Hadoop プロジェクトは，大規模データの分散処理を可能とする OSS である．主に，Hadoop Common, HDFS, Hadoop YARN, Hadoop Map/Reduce の 4 つのコンポーネントで構成される．Hadoop を選定した理由は，4 つのコンポーネントが互いに密接な関係にあるため Blocking Bug が発生しやすいと考えたためである．

5.2. 対象データ

依存関係分析の対象データとして，不具合データ，ソースコード及びソースコードの変更履歴のデータを用いる．Hadoop プロジェクトでは，不具合データは不具合管理システム JIRA から，ソースコード及びソースコードの変更履歴は Git から取得することができる．対象データを作成するまでの 3 つの手順を以下に示す．

Step1：取得した不具合データから解決済みの不具合データのみを抽出する．解決済みの不具合データのみを抽出するのは，不具合の修正がおこなわれるまでは修正対象物（ソースファイル等）を特定することができないためである．本研究では不具合票の Resolution が “Fixed” のものを解決済みの不具合とする．

Step2：Blocking Bug のうち Blocking Set を作成することができるものを抽出する．本研究では Blocking Bug の依存関係を分析したいと考えているため Blocking Set の構築が可能である Blocking Bug に限定する．

Step3：抽出した不具合データのうち，不具合データとコミット履歴の紐付けができるものを抽出する．コミットメッセージに不具合 ID が含まれていれば，該当する不具合を修正するためのコード変更を特定することができる．また，SZZ アルゴリズム [15] を併用することで，該当する不具合が混入したリビジョンを特定することができる．特定ができた Blocking Bug と関係のない不具合を本研究で扱う「非 Blocking Bug」とする．また，Blocking Set のすべての不具合が特定できたものを本研究で扱う「Blocking Bug」とする．

表 1. データセットの概要

コンポーネント	Blocking Bug	非 Blocking Bug
Hadoop Common	107	1,223
HDFS	57	1,851
Hadoop YARN	10	134
Hadoop Map/Reduce	60	1,575
全体	**234**	**4,783**

表 2. 修正時間

	Blocking Bug	非 Blocking Bug	Blocked Bug
修正時間（日）	10	11	52

5.3. Blocking Bug の実態把握のための分析

5.3.1 不具合報告のメトリクス

1. 不具合の数

   Blocking Bug と非 Blocking Bug の不具合数についての分析結果を表 1 に示す．

   234 件の Blocking Bug と 4,783 件の非 Blocking Bug が確認された．

2. 修正時間

   修正時間についての分析結果を表 2 に示す．

   表 2 では，Blocking Bug，非 Blocking Bug，Blocked Bug の修正時間の中央値を日数単位で示している．Blocking Bug と非 Blocking Bug の間で修正時間に大きな差はみられなかった（U 検定の結果，$p = 0.46$ であった）．また，Blocked Bug は修正時間が中央値で 52 日であることが確認された．

3. 優先度別の不具合票の内訳

   優先度別に不具合票の内訳を表 3 に示す．

   高い優先度 (Blocker と Critical) が割り当てられた不具合の割合については，Blocking Bug と非 Blocking Bug の間で大きな違いはみられなかった．低い優先度（Minor や Trivial）が割り当てられた不具合の

表 3. 優先度別の不具合票の内訳

優先度の割合	BlockingBug	非 BlockingBug
Blocker	25 (10.7%)	426 (8.9%)
Critical	12 (5.1%)	319 (6.7%)
Major	177 (75.6%)	2,965 (62.0%)
Minor	18 (7.7%)	854 (17.9%)
Trivial	2 (0.9%)	219 (4.6%)

表 4. 各 Blocking Set の基本統計量

	最大値	最小値	平均値	中央値	標準偏差
不具合票の数	4.0	2.0	2.2	2.0	0.2
修正担当者の数	3.0	1.0	1.4	1.0	0.3
修正ファイル数	9405.0	1.0	85.8	13.0	661.1

表 5. 構成不具合票数別の Blocking Set の件数

構成不具合票数	件数
2 件	187 (80.3%)
3 件	37 (15.9%)
4 件	9 (3.9%)

割合については，非 Blocking Bug に比べて Blocking Bug のほうが少ないことが確認された．

4. **Blocking Set の構成不具合数** Blocking Set の構成不具合数を表 4 の 1 行目に示す．表 4 に，Blocking Set の統計量（1 つの Blocking Set に含まれる不具合数，修正担当者数，修正ファイル数）を示す．Blocking Set に含まれる不具合の数の最小値と中央値が 2 件であることから，半数以上の Blocking Set は Blocking Bug と Blocked Bug が 1 件ずつであることがわかる．

   表 5 に Blocking Set の構成不具合数別の件数を示す．約 8 割の Blocking Set は，構成する不具合の数が 2 つである．つまり，約 8 割の Blocking Bug は 1 つの Blocked Bug の修正を妨害していることが確認された．

5. **Blocking Set の発生手順**

   Blocking Bug と Blocked Bug のどちらが先に報告されたかを表 6 に示す．

   Blocking Bug の約 7 割が Blocked Bug のほうが先に報告されていることが確認された．

5.3.2 修正担当者のメトリクス

1. **Blocking Set の担当者の数**

表 6. 先に報告された不具合の数

	Blocking Bug	Blocked Bug
先に報告された不具合の数	69（29.5%）	165（70.5%）

表 7. 開発者の経験による Blocking Bug の発生件数

開発者の経験	件数	コミット回数
浅い	13	614
豊か	173	6066

表 8. 修正ファイル数（中央値）

	Blocking Bug	非 Blocking Bug
修正ファイル数	4	2

Blocking Set に含まれる不具合を修正した担当者の数を表 4 の 2 行目に示す．修正担当者の中央値は 1 人であった．

2. **修正担当者の経験**

   経験の違いによる Blocking Bug の発生件数とコミット回数を表 7 に示す．234 件の Blocking Bug のうち，新たなファイルの作成のみで修正を完了させた Blocking Bug が 48 件存在した．この 48 件の Blocking Bug を除いた，186 件の Blocking Bug に分析をおこなった結果，経験が浅い開発者は 47 回に 1 回の割合で Blocking Bug を発生させることが確認された．また経験が豊かな開発者は 35 回に 1 回の割合で Blocking Bug を発生させていた．

5.3.3 修正対象物のメトリクス

1. **修正ファイル数（中央値）**

   Blocking Bug と非 Blocking Bug の修正時に変更したファイルの数を表 8 に示す．分析の結果，修正ファイル数の中央値は Blocking Bug の方が多いことが確認された（U 検定をおこなった結果，$p < 0.00$）．つまり，Blocking Bug の修正には非 Blocking Bug と比べてより多くのファイルを修正する必要があることが確認された．

2. **各 Blocking Set のファイル数**

   各 Blocking Set に含まれる不具合を修正するために変更したファイルの総数を表 4 の 3 行目に示す．表 4 から修正総ファイルは最大で 9405 ファイル，最小で 1 ファイルであることが確認された．また中央値は 13 ファイルであった．

表 9. 修正担当者の人数の内訳

修正担当人数	件数
1	145（62.0%）
2	79（33.8%）
3	10（4.2%）

5.4. Blocking Bug の発生理解のための依存関係分析

RQ1–1：優先度が高い Blocked Bug の発生によって Blocking Bug の修正は活発になるか

243 件の Blocking Bug のうち，優先度が高い Blocked Bug を持っている不具合は 45 件確認された．45 件の Blocking Bug のうち，Blocking Bug の検出時間の中央値である 2 日を超えた後に修正が開始された Blocking Bug は 3 件のみであった．

RQ1–2：優先度が高い Blocked Bug の発生が Blocking Bug の発見に寄与するか

45 件の Blocking Bug のうち，不具合管理システムでの最初の変更が “blocks” のタグの挿入であるものは 4 件のみであった．

RQ1–1 と RQ1–2 の結果から優先度が高い Blocked Bug であっても Blocking Bug の発見にはつながらないことが確認された．

RQ2：修正担当者が多いほうが Blocking Bug が発生しやすいか

表 9 に Blocking Set の修正担当者の人数に関する詳細な内訳を示す．

全 Blocking Set のうち，約 62%が 1 人の担当者によって修正されていることが確認された．RQ2 の結果から，修正担当者が多くても Blocking Bug が発生するとは言えないことが確認された．

RQ3：ファイルが参照されているほど Blocking Bug が発生しやすいか

表 10 に Blocking Bug と非 Blocking Bug ごとの参照数の中央値を示す．

表 10 は Blocking Bug と非 Blocking Bug のそれぞれの参照数の中央値と，U 検定をおこなった際の p 値を示している．分析の結果，将来 Blocking Bug になるファイルのほうが，将来 Blocking Bug とならないファイルと比べてより多くのファイルから参照されていることが確認された．RQ3 の結果から，多くのファイルから呼ばれているほうが Blocking Bug になりやすいことが確認された．

表 10. 参照ファイル数（中央値）

	Blocking Bug	非 Blocking Bug	p 値
参照数	1	1	0.39
被参照数	4	3	0.00**

表 11. チーム別の Blocking Bug 発生件数

チーム	件数
経験豊富な開発者のみ	81
経験豊富な開発者と浅い開発者の混合	102
経験の浅い開発者のみ	3

RQ4：経験が豊富な開発者によって Blocking Bug の発生を予防できるか

開発者の経験を基に分けたグループの Blocking Bug の発生件数を表 11 に示す．表 11 に示すように経験の浅い開発者のみのチームによって埋め込まれる Blocking Bug は全体の 1.6%（3/186）であり，98.4%の Blocking Bug は経験豊富な開発者が存在しているチームによって埋め込まれていることがわかる．したがって，経験が豊富な開発者がいても Blocking Bug の発生を予防できるとは言えないことが確認された．

6. 考察

本章では，まずケーススタディの結果に基づいた考察をおこなう．その後，本研究の制約について述べる．

6.1. Blocking Bug とファイルの関係

本研究では，Blocking Bug が埋め込まれるファイルは他のファイルに比べて被参照ファイルが多いことが確認された．また，Blocking Bug は非 Blocking Bug に比べて修正するために変更しなければならないファイルが多いことも確認された．Blocking Bug が埋め込まれるファイルは参照されているファイルが多いので，ファイルの変更によって与える影響が大きく修正すべきファイルも多くなったと考えられる．また，これらの結果から，Blocking Bug は非 Blocking Bug に比べて修正にコストがかかると言えるので，早期修正と発生の予防の両方の観点から参照されているファイルを減らすことは重要であると考えられる．ファイルの被参照数を減らすための

方法としては，メソッドの利用を継承に変更する方法や子クラスの共通メソッドを親クラスに引き上げるなどのリファクタリング方法が考えらえれる．

実際に Hadoop プロジェクトで修正されたファイルの依存関係の一部を用いて被参照ファイルを減らす方法を説明する．図 2 において “Writable” ファイルは Blocking Bug として修正されたファイルである．“Writable” ファイルは “Configuration” ファイルから継承のために，また，“WritableUtils” ファイルからはメソッドの利用のために参照されている．“WritableUtils” ファイルは “Configuration” ファイルからメソッド利用のために参照されている．このとき，“Configuration” ファイルは “Writable” ファイルを継承しているが，図 2 の下図のように “WritableUtils” ファイルが “Writable” ファイルを継承することで “Configuration” ファイルは “Writable” ファイルを継承する必要がなくなる．結果として，“Writable” ファイルは “Configuration” ファイルから参照されなくなり，参照されるファイルの数は 2 ファイルから 1 ファイルへ減少する．

6.2. Blocking Bug と修正担当者の関係

本研究では，約 62%の Blocking Set は 1 人で修正することが確認された．これは修正の効率化を重視した結果であると言える．例えば，修正担当者が複数人である場合，各修正担当者はそれぞれの不具合に関する情報を交わしながら担当する不具合を修正する必要がある．そのため，非 Blocking Bug に比べて修正に時間がかかる．しかし，修正担当者 1 人で Blocking Set に対応すれば，不具合に関する情報を交わす必要がなくなるので非 Blocking Bug 同様に議論のための時間は必要なくなり，

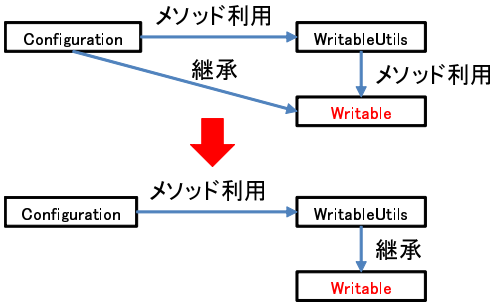


図 2. Hadoop プロジェクトの実態例

修正が効率化されると考えられる．実際に，修正担当者が 1 人の場合と複数人の場合で Blocking Bug の修正時間の差を比較した結果，修正時間の中央値は，修正担当差者が 1 人である場合は 7 日であるのに対し，複数人である場合は 20 日であった（U 検定の結果 p 値 $= 0.02*$）．つまり，Blocking Bug の早期修正をおこなうためには Blocking Set を 1 人で修正する方が効率的であると言える．

6.3. 本研究の制約

本研究ではファイルを変更する際のコメントを用いて，不具合，モジュール，人の 3 つのデータを結びつけている．しかし，コメントの内容によっては結びつけができないこともあるため，結びつけができなかったデータは対象から除外している．また，不具合の解決方法によっては，結びつけができないために対象から除外しているものもある．これは，不具合の解決方法の中にはファイルの変更を伴わないものがあるためである．例えば，“Won’t Fix” とつけられている不具合表では，ファイルの変更がおこなわれる可能性は低い．よって，本研究で扱う Blocking Bug は Blocking Set 中のすべての不具合票で最低 1 ファイル以上の修正がおこなわれた Blocking Bug に限定される．

7. おわりに

本論文では Hadoop プロジェクトを対象に Blocking Bug の実態把握と発生原因を理解するための分析をおこなった．Blocking Bug の発生原因を理解するため，不具合間の依存関係，修正担当者間の依存関係，修正対象物の依存関係の 3 つの視点に加えて，3 つの視点の重畳的な依存関係の 4 つの視点から依存関係分析をおこなった．Hadoop プロジェクトを対象におこなったケーススタディの結果，Blocking Bug は Blocked Bug が先に報告されてから報告されることが多いことや，非 Blocking Bug に比べて修正のために変更を加えるファイル数が多いことが確認された．また，多くのファイルに参照されていると Blocking Bug になりやすいことが確認された．今後の課題としては，本研究で得られた知見を基に Blocking Bug の発生をどの程度予防できるかを確かめる必要があると考えられる．また，本論文で得られた知見は Blocking Bug 検出の予測やファイル変更時の影響分析にも活用できると考えられる．

参考文献

[1] Page-Jones, M.: *Practical Guide to Structured Systems Design (2nd Edition)* (1988).

[2] April, A. and Abran, A.: *Software Maintenance Management: Evaluation and Continuous Improvement* (2008).

[3] Shokripour, R., Anvik, J., Kasirun, Z. M. and Zamani, S.: Why So Complicated? Simple Term Filtering and Weighting for Location-based Bug Report Assignment Recommendation, in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR' 13)*, pp. 2–11 (2013).

[4] Menzies, T. and Marcus, A.: Automated severity assessment of software defect reports, in *24 th IEEE International Conference on Software Maintenance (ICSM '08)*, pp. 346–355 (2008).

[5] Tian, Y., Lo, D. and Sun, C.: DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis, in *29th IEEE International Conference on Software Maintenance (ICSM '13)*, pp. 200–209 (2013).

[6] Runeson, P., Alexandersson, M. and Nyholm, O.: Detection of Duplicate Defect Reports Using Natural Language Processing, in *Proceedings of the 29th international conference on Software Engineering (ICSE '07)*, pp. 499–510 (2007).

[7] Sun, C., Lo, D., Wang, X., Jiang, J. and Khoo, S.-C.: A discriminative model approach for accurate duplicate bug report retrieval, in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE '10) - Volume 1*, pp. 45–54 (2010).

[8] Kashiwa, Y., Yoshiyuki, H., Kukita, Y. and Ohira, M.: A Pilot Study of Diversity in High Impact Bugs, in *Proceedings of 30th International Conference on Software Maintenance and Evolution (ICSME2014)*, pp. 536–540 (2014).

[9] Chen, T., Nagappan, M., Shihab, E. and Hassan, A. E.: An Empirical Study of Dormant Bugs, in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*, pp. 82–91 (2014).

[10] Shihab, E., Mockus, A., Kamei, Y., Adams, B. and Hassan, A. E.: High-impact Defects: A Study of Breakage and Surprise Defects, in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*, pp. 300–310 (2011).

[11] Nistor, A., Jiang, T. and Tan, L.: Discovering, Reporting, and Fixing Performance Bugs, in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*, pp. 237–246 (2013).

[12] Garcia, H. V. and Shihab, E.: Characterizing and Predicting Blocking Bugs in Open Source Projects, in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*, pp. 72–81 (2014).

[13] Biazzini, M., Monperrus, M. and Baudry, B.: On Analyzing the Topology of Commit Histories in Decentralized Version Control Systems, in *Proceedings of the 30th International Conference on Software Maintenance and Evolution* (2014).

[14] Bhattacharya, P., Iliofotou, M., Neamtiu, I. and Faloutsos, M.: Graph-based Analysis and Prediction for Software Evolution, in *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pp. 419–429 (2012).

[15] Śliwerski, J., Zimmermann, T. and Zeller, A.: When Do Changes Induce Fixes?, in *Proceedings of the 2005 International Workshop on Mining Software Repositories*, MSR '05, pp. 1–5 (2005).

報酬構造を考慮したテストケース生成と信頼性評価の効率性

小澤 公貴
広島大学 大学院工学研究科
ozawa@s.rel.hiroshima-u.ac.jp

土肥 正
広島大学 大学院工学研究院
dohi@rel.hiroshima-u.ac.jp

要旨

本稿では, マルコフ使用モデル (Markov Usage Model) とシステムの運用プロファイル (Operational Profile) に基づいてテストケースを自動生成するための統計的方法について考察する. 特にソフトウェアの設計情報を報酬構造として導入することでマルコフ報酬過程を定義し, ソフトウェアテストの効率化とソフトウェアの信頼性評価尺度の算出を行う．最終的に, 実際のソフトウェアプログラムにフォールトを埋め込んでテスト実験を行った結果，ソフトウェアメトリクスを報酬として与えたテストケース生成アルゴリズムは, 従来の報酬なしのアルゴリズムと比較して数多くのフォールトを検出でき，しかも規定の信頼度に達する為に必要とされるテストケース数を効率的に削減できることが示される．

1. はじめに

ソフトウェア開発は主に,「要件定義」,「基本設計」,「詳細設計」「コーディング」,「テスト」の各工程から構成されている. ソフトウェアテストはテスト工程で行われ，ソフトウェア内に潜在するフォールトの発見・除去を通じて，要求される機能を満たしているかどうかを確認する．よって，テスト工程は製品の信頼性・品質を保証するために必要不可欠であり, 他の工程と比較して最も多くのコストがかかることが知られている. 特に, 大規模・複雑化するソフトウェア開発の現状において，より安価でかつ効果的なテストを実施することでテストの質を向上させることが要求されている.

一般に, プログラムを実際に実行するテストを動的テストと呼び，コードレビューのように実行を伴わないテストを静的テストと呼ぶ. 動的テストを行う際，何も判断基準を持たずにプログラムを実行するだけでは入出力の正誤判定が行えないため，ソフトウェアへの入力と正しい期待出力の組から構成されるテストケースを投入することで入出力の正誤判定を行う. 特別に大規模なソフトウェアでなくても, この入出力関係を規定する組合わせ数は膨大となり, ソフトウェアの全ての実行パスを網羅するテストケースを生成することは現実的に不可能である．よって動的テストでは，テストケースの効率的な自動生成が重要な問題となっている.

動的テストは，さらにホワイトボックステストとブラックボックステストに大別される．ホワイトボックステストとは，プログラムの内部構造を理解した上で，それら一つ一つが意図した通りに動作しているかを検証するテスト技法である．ホワイトボックステストにおけるテストケースは，命令の網羅性や判定条件の網羅性などの内部構造を注意深く検査しながら設計されるため, テストケースの生成には非常にコストがかかる．一方, ブラックボックステストはプログラムの内部構造には着目せず, 与えられた入力に対して正しい出力がなされたかどうかの確認のみを行うテスト技法である．ブラックボックステストに基づいたテストケースは，同値分割や限界値分析を通じて, 入力データを考慮しながら設計される.

ブラックボックステストの興味深い研究テーマとして, 設計情報をテストケース生成に活用するモデルベーステスト (Model-based Test; MBT) が挙げられる. これは, テスト対象となるソフトウェアの振る舞いや機能を抽象的モデルを用いて表現し，モデル上で表現されたソフトウェア・アーキテクチャの情報をテストケース生成に利用するものである．このような形式的モデルを用いることの利点として, 設計されたテストケースがモデル上で規定された各状態や遷移をどれだけ網羅できたのかを明確にすることが挙げられる．形式的モデルの例として,

UML (Unified Modeling Laungage) などの設計記述言語を用いるものや, 木構造や有限状態機械 [3],[5] などのグラフモデルを利用するもの, マルコフ連鎖 [16], [17] や確率オートマトンなどの確率モデルを用いるものが代表的である.

確率モデルを用いたテストケース生成について，今富ら [7] は，MBT の一種でありソフトウェアの運用プロファイル（Operational Profile）を用いてテストケースを生成する運用プロファイルベーステスト (Operational Profile-based Test; OPBT) に着目した．MBT がテスト網羅性のみを志向してテストケースを設計するのに対し，OPBT では運用プロファイルを用いることによりテスト網羅性とユーザの使用環境を同時に考慮してテストケースを設計する．本稿では，今富ら [7] の方法を拡張し，規定のソフトウェア信頼度に到達したらテストを打ち切る方策の下で，報酬構造を考慮したテストケース生成法の有効性を評価する．

2. 関連研究

テストケース生成法に関する従来研究として，Whittaker and Thomason [17] により提案され, 後に Avritzer and Weyuker [1], Gutjahr [6], Kallepalli and Tian [8], Poore ら [13], Whittaker ら [18] によって精緻化されたマルコフ連鎖に基づく統計的テストケース自動生成法がある．これは，ソフトウェアの機能や構造に着目した振る舞いを確率モデルによって表現し, それをシミュレートすることでテストケースの自動生成を可能にしている. プログラムの挙動をモデル化するための表現能力に関して, 有限状態機械 [3],[5] やマルコフ性を持たないより一般的な確率モデル [4] など他にも優れた方法が存在するが, パラメータの物理的性質の同定や統計的推論の簡便性から, マルコフ連鎖に基づく方法は最も汎用性が高いと言える.

テストケースの自動生成とは離れて, マルコフ連鎖に基づく統計的方法では, テストケース投入後のフォールト検出に関する実績データからソフトウェアの信頼性を定量的に評価できることが特徴として挙げられる. Avritzer and Weyuker [1], Gutjahr [6], Kallepalli and Tian [8], Prowell and Poore [14], Weyuker [15], Whittaker ら [18] はプログラムの振る舞いを表すマルコフ連鎖の状態推移図に故障状態を導入し, テストケース実行中にフォールトが検出される確率を推定し, さらにプログラムを任意回数実行中に障害に至る確率を推定する方法について議論している. これより, プログラムの実行中にソフトウェア障害が発生しない確率であるソフトウェア信頼度を, テストケースの投入履歴から統計的に推論することができる.

以上の統計的テストケース自動生成法を踏まえ，今富ら [7] は，従来までの多くの研究ではプログラムの振る舞いを表現するためにモデルの表現能力の向上だけに注力しており, ソフトウェアの設計情報や各モジュールを構成するソースコードの定量的性質に着目していないことを指摘した．そこで，パフォーマビリティ [12] もしくは報酬過程の概念を導入することで, より集中してテストを行うべきモジュールをソフトウェアメトリクスによって特徴づけた上で，テストケースの効率的な生成方法を提案している．最近 Avritzer ら [2] は, この報酬過程の概念を導入し，実在するミッションクリティカルなシステムに対するテストケースの自動生成問題を取扱い，ソフトウェア信頼度の推定問題を考察している.

一方で, Avritzer ら [2] の論文は連続時間マルコフ連鎖 (Continuous-time Markov Chain; CTMC) の過渡解析の結果を用いており, 連続時間で駆動するシステムに対するテストケース生成と信頼度推定に主眼を置いている．今富ら [7] はテスト工程においてより取扱いの簡便な離散時間マルコフ連鎖 (Discrete-time Markov Chain; DTMC) を仮定し, さらに Avritzer ら [2] が論文中に明示的に述べていなかった報酬の決め方について実証的なアプローチを展開している．すなわち，ソフトウェアの各モジュールを構成するプログラムを, ソフトウェア設計メトリクスを用いて報酬の形で与えることを提案している．これにより, プログラムの振る舞いを表す形式モデルは離散時間マルコフ報酬過程 (Discrete-time Markov Reward Process; DTMRP) として表現される.

本稿では今富ら [7] の手法をもとに，単にテストケースを生成するだけでなく，ソフトウェア信頼度の定量的な推定も併せて行う．最終的に, 実際のソフトウェアプログラムにフォールトを埋め込んでテスト実験を行った結果, ソフトウェアメトリクスを報酬として与えたテストケース生成アルゴリズムは，従来の報酬なしのアルゴリズムと比較して数多くのフォールトを検出でき，しかも規定の信頼度に達するために必要とされるテストケース数を効率的に削減できることを示す.

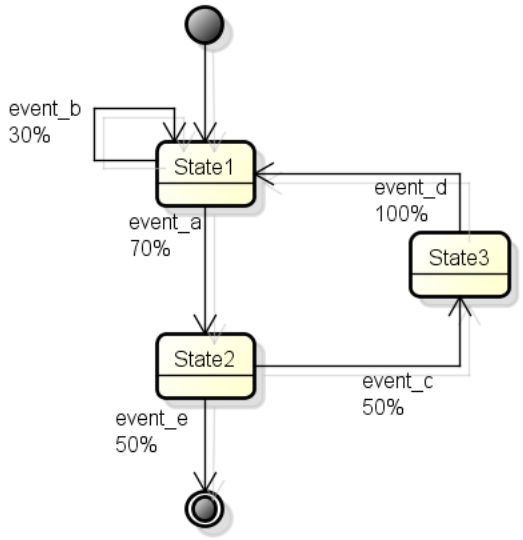


図 1. 有限状態機械によるモデル表現の一例.

3. 運用プロファイルベーステスト

運用プロファイルとは, 開発対象であるソフトウェアが実際の運用環境においてどのように使用されるかを規定した仕様であり, ソフトウェアの各機能や命令を使用する頻度を予め想定することでユーザの運用環境を予測するために用いられる. Musa [10], [11] は, ソフトウェアの各機能の使用確率を要求仕様段階において仮定することで, ユーザの運用環境を考慮した設計, テスト, 検証を行うべきであることを主張している. 運用プロファイルの定量的な取扱いでは, ソフトウェアにおける各機能の使用確率 (Usage Probability) を過去のバージョンの使用履歴や類似ソフトウェアの開発データから推定したり，ユーザの使用振る舞いを開発者が主観的に見積もることにより割当てられる．この使用確率は有限状態機械の推移確率に対応しており, 運用プロファイルを規定することで形式的モデルの振る舞いが定量的に表現できる. 運用プロファイルを MBT に適用するテストである OPBT は, 従来の MBT で作成したモデルに対して, 運用プロファイルから各状態の遷移確率を付加することでユーザの運用環境を記述した動的モデルであると解釈できる.

図 1 に, 有限状態機械によって表現されたソフトウェアの振る舞い図の例を示す．ここで, 図中のノードはソフトウェア機能の種類を表す状態を意味し，アークは状態間の遷移をそれぞれ表している．各遷移に付加された event は遷移の発火条件を表し，黒丸は実行開始状態，二重丸は実行終了状態をそれぞれ表す．図中のアーク上に割当てられた数値は状態推移確率を表しており, 例えば “State2” に対応するソフトウェア機能を実行した後, 確率 50% で “event_e” が生じるとプロセスの実行は終了し, 確率 50% で “event_c” が生じると状態は “State3” に移行し, さらに 100% の確率で状態は “State1” に移る. テストケースの生成は, この状態機械を実行（シミュレート）することで実現される. すなわち, 実行開始状態から実行終了状態に至るサンプルパスをモンテカルロ・シミュレーションを通じて生成し, 各テストケースを構成する入出力値を定める.

4. マルコフ報酬過程

4.1. 離散時間マルコフ連鎖

本論文では, 有限状態機械のサブクラスである DTMC を用いる. 状態の表現能力に関してはオートマトンなど他にも有力な有限状態機械が存在するが, マルコフ連鎖を用いる利点としてモデルパラメータの物理的意味が明確であり，かつ実際のデータからパラメータを統計的に推定することが容易である点が挙げられる．また, ソフトウェア信頼度に代表される信頼性評価尺度を算出するためには, 必然的に解析的取扱いが可能な確率過程を適用する必要がある.

今, 離散時刻 $n = 0, 1, 2, \cdots$ において非負値状態空間上で値をとる確率過程 $\{X(n), n = 0, 1, 2, \cdots\}$ を考える．任意の状態 i, j, i_n に対して，

$$\begin{aligned} P\{X(n+1) = j \mid X(0) = i_0, X(1) = i_1, \cdots, \\ X(n-1) = i_{n-1}, X(n) = i\} \\ = P\{X(n+1) = j \mid X(n) = i\} = p_{ij} \end{aligned}$$

のような関係が成立するならば, $X(n)$ はマルコフ性を有し, DTMC に従うと言う．ここで状態 i から状態 j に推移する条件付き確率 $p_{i,j}$ は推移率と呼ばれる．図 1 の実行開始状態を除いた 4 つの状態の内，実行終了状態のようにこれ以上推移が起こらない状態を吸収状態, 以降も推移可能な状態を過渡状態と呼ぶ. 過渡状態の数が

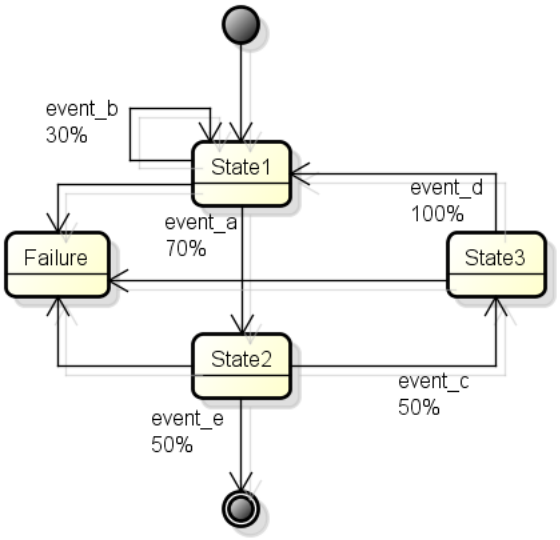


図 2. 障害状態を追加したモデル表現.

m で吸収状態の数が l の DTMC において, 推移確率行列を

$$\mathbf{P} = \left(\begin{array}{c|c} \mathbf{Q} & \mathbf{C} \\ \hline \mathbf{O} & \mathbf{I} \end{array} \right) \tag{1}$$

によって定義する. ここで, $\mathbf{Q}$, $\mathbf{C}$ は, 過渡状態から過渡状態への推移と, 過渡状態から吸収状態への推移を表す部分行列であり, $\mathbf{O}$, $\mathbf{I}$ はそれぞれ零行列と単位行列を表している. よく知られた DTMC の結果より, DTMC の n ステップ推移確率行列は

$$\mathbf{P}^{(n)} = \mathbf{P}^n = \begin{pmatrix} \mathbf{Q}^n & \mathbf{C}_n \\ \mathbf{O} & \mathbf{I} \end{pmatrix} \tag{2}$$

によって与えられる. ここで部分行列 C_n は

$$\mathbf{C}_n = (\mathbf{I} + \mathbf{Q} + \cdots + \mathbf{Q}^{n-1})\mathbf{C} \tag{3}$$

となる. $\mathbf{Q}$ は列ベクトルの総和が 1 となる非負値行列であり,

$$\lim_{n\to\infty} \{\mathbf{I} + \mathbf{Q} + \mathbf{Q}^2 + \cdots + \mathbf{Q}^{n-1}\} = (\mathbf{I} - \mathbf{Q})^{-1} \tag{4}$$

の関係を満足する. この $\mathbf{M} = (\mathbf{I} - \mathbf{Q})^{-1}$ は基本行列 (fundamental matrix) と呼ばれる. テスト対象とするソフトウェアには m 個の機能が備わっており, フォールトが全く含まれていない場合には, 1 回のテスト実行で状態 1 から任意の過渡状態を経由して実行終了状態である吸収状態に至る. もし各機能を実現するモジュールにフォールトが潜在的に含まれるならば, 各状態からもうひとつの吸収状態である “Failure” 状態に推移が発生する. 図 2 は $m = 3, l = 2$ の場合の DTMC の推移図の例を表している. OPBT においてテストケースを生成する場合, 図 1 のようにフォールトの検出を想定することなく, 実行開始状態から実行終了状態までの経路をシミュレートし, 各機能が順に実行される遷移系列を求める. 求められた遷移系列の入力値と期待出力値を予め用意することで, 当該テストケースによる正誤判定を行うことができる. 十分なテストケースが投入された後, 図 2 の過渡状態 j への累積訪問回数を s_j $(j = 1, 2, \cdots, m)$, 各過渡状態においてフォールトが検出された回数を f_j とすれば, モジュール j の信頼度は

$$R_j = 1 - (f_j/s_j) \tag{5}$$

によって求められる. よって, 任意回数のテスト実行でフォールトが検出されることなく終了する確率であるソフトウェア信頼度を, 基本行列の要素から容易に算出することができる. これにより, DTMC を用いてテストケースの自動生成とソフトウェア信頼度の推定を同時に行うことが可能となる.

4.2. マルコフ報酬過程とソフトウェアメトリクス

先に述べた DTMC に基づいた OPBT の問題点として, 仕様の段階で類推される運用プロファイル（DTMC における過渡状態間の推移確率）を予め正確に知ることが困難である点が挙げられる. 一般に, 不特定多数のユーザがどの機能をどのような頻度で使用するかを正確に予測することは容易でなく, 運用プロファイルを利用した定量化技術にも自ずから限界がある. また, 運用プロファイルはあくまでユーザの動作環境を想定したものであり, ユーザによるソフトウェアの使用環境を正確にモデル化できたとしても, 必ずしも潜在するフォールトが多数検出されるテストケースを生成できる保証はない. 換言すれば, 頻繁に使用される機能を実現したモジュールに常に多くのフォールトが含まれているわけではないため, なるべく多くのフォールトを検出したいというテスト効率の観点から通常の OPBT が適しているとは言えない.

ここでは, DTMC の各状態を訪問する度に r_i $(i = 1, 2, \cdots, m)$ の報酬が与えられるものとし, DTMC の拡張である DTMRP を導入する．マルコフ報酬過程はパフォーマビリティの概念と共に Meyer [12] によって導入され, 最近 Avritzer ら [2] によってテストケースの自動生成に応用されている．テストにおいて探索すべき（フォールトプローンな）モジュールが存在し, その探索優先順位が予め定められているならば, 優先順位ごとに大きくなる報酬値を割振り, 1 回のテストケースの実行が終了するまでの単位ステップ当たりの累積報酬を求め, その値が大きいテストケースから順に実行すればよい．しかしながら, 問題は各モジュールをテストする際に割与えられる報酬値をどのように決定するかにある．最も簡便な方法は, 単体テストにおいてフォールトプローンモジュールの相対順位を決定し, 優先順位の低い（フォールトがより多く含まれない傾向にある）モジュールから, $1, 2, \cdots$ のような整数値を割振ることであろう．しかし, 単体テストの成果として類推されたフォールトプローンモジュールの相対順位さえも, 必ずしもシステムテストにおいて重点的に探索されるべきモジュールであるかどうか定かではない．

本稿では, ソフトウェアの各モジュールを特徴付ける報酬値として，ソフトウェアの複雑性メトリクスの相対順位を用いることを提案する．複雑性メトリクスとは, プログラムを構成するソースコードから計測される形式的構造の複雑性を表す評価尺度である．本研究では単純なコード行数， Halstead によるソフトウェアサイエンス理論 (例えば [19] を参照) に基づいたボリュームメトリクス，および McCabe [9] によるサイクロマチック数を採用した．コード行数は, 開発規模の見積もり方法として最も計測が簡単な尺度である．一方, ソフトウェアサイエンス理論とは, 複雑性の尺度をオペレータおよびオペランドの数による関数によって定義する複雑性メトリクスの枠組みである．ここで, オペレータとは算術演算子, 比較演算子, 理論演算子, 関数名, コマンド名, 句読点, 区切り符号などで構成され，オペランドとは変数, 定数, ラベルなどを含む．オペレータとオペランドの種類および総出現回数を計測することにより，プログラム開発に要する時間やプログラムがメモリ上に占める大きさなどの指標が計算できる．

Halstead によるソフトウェアサイエンス理論における基本的尺度は以下の 4 つである．

- n_1 : プログラム内のオペレータの種類数
- n_2 : プログラム内のオペランドの種類数
- N_1 : プログラム内でのオペレータの総出現数
- N_2 : プログラム内でのオペランドの総出現数

これらはプログラムのソースコードから容易に計測可能であり, これら 4 つの基本的尺度を用いてプログラムのボリュームメトリクスを算出する．

報酬値として用いる評価尺度の例を表 1 に示す．表中のプログラムの大きさ V^* は予測値であり，正確な値が求められないため, 本稿では

- プログラムの長さ N
- プログラムの大きさ V
- プログラムの水準の推定値 $\hat{L}$
- プログラミングの困難さの推定値 $\hat{D}$
- プログラミング労力の推定値 $\hat{E}$
- プログラミング言語水準の推定値 $\hat{\lambda}$

の 6 つを用いた．McCabe によるサイクロマチック数は，プログラムの複雑度をソースコードから測定する手法である．ソースコードから制御フローを有向グラフで表現することで，グラフを網羅するパスの数がサイクロマチック数として算出される．制御フローグラフが e 個の枝，n 個の節点を持つとき，サイクロマチック数 M は

$$M = e - n + 1 \tag{6}$$

と求めることができる．各モジュールごとにこれら 6 つのボリュームメトリクスもしくは複雑性メトリクスを算出し, 優先順位の低い (メトリクスの値が小さい) 順に $1, 2, \cdots$ の整数値を割振った相対順位を報酬として与える．DTMC に基づいて生成された各テストケースごとに単位ステップ当たりの累積報酬を計算し, その値が大きいものから優先的にテストに使用する．

5. 実験

ここでは実際のプログラムに予めフォールトを挿入し, 生成されたテストケースに基づいてフォールトを発見・除去する実験を行うことで報酬に基づいたテストケース

表 1. ソフトウェアサイエンス理論における評価尺度 (文献 [19] から抜粋).

分類	尺度	公式
プログラムの長さ	プログラムの長さ	$N = N_1 + N_2$
	プログラムの長さ N の推定値	$\hat{N} = N_1 + N_2$
プログラムの大きさ	プログラムを実現したときの大きさを記述に必要とされるビット数により表したもの	$V = N\log_2(n_1 + n_2)$
	アルゴリズムを表現したときの最小の大きさ	$V^* = (2n_2^*)\log_2(2 + n_2^*)$ $(n_2^*) =$ アルゴリズムの入出力パラメータ数
プログラムの水準	プログラムの水準を実現された大きさ V と最も簡潔な大きさ V^* との比で表したもの	$L = V^*/V$
	プログラムの水準 L の推定値	$\hat{L} = 2n_2/(n_1N_1)$
プログラミングの困難さ	プログラミングをするのに伴う困難さ	$D = 1/L$
	プログラミングの困難さ D の推定値	$\hat{D} = 1/\hat{L} = 2n_2/(n_1N_1)$
プログラミング労力	アルゴリズムを実現するときの理解に必要とされる労力	$E = V/L = V^2/V^*$
	プログラミング労力 E の推定値	$\hat{E} = V/\hat{L}$
プログラミング言語水準	使用するプログラミング言語の言語水準	$\lambda = LV^*$
	言語水準 λ の推定値	$\hat{\lambda} = \hat{L}^2V$

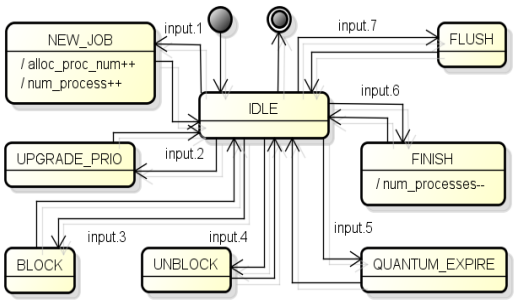


図 3. ‘schedule’ の状態遷移図.

生成法の性能評価を行う. 実験対象とするソフトウェアは ‘schedule’ と呼ばれるプロセス管理ソフトウェア [1] であり, C 言語で記述され, コード行数は約 400 行, 18 の関数から構成されているプログラムである. このプログラムを起動するとまず待機状態になり, 次に入力を受け付ける. 入力された値によって各機能を実行し，機能実行が終了すると再び待機状態となる. 終了命令が入力されるまで上述の動作を繰り返す. テスト対象ソフトウェアの振る舞いを有限状態機械で表したものを図 3 に示す. プロセスの開始, 終了, 優先度の変更など 7 つの機能によって状態が構成されていることが分かる. このソフトウェアに実際のテスト工程で発見された 7 つのフォールトを事前に埋め込み，用意されたテストケースを実行することで実際のソフトウェアテスト環境を再現した. ここでフォールトはプログラム中の 7 箇所に対応しており，1 つの故障が複数箇所の原因から引き起こされた場合，原因箇所それぞれが発見されたフォールトとしてカウントされる. 本稿では以下のように運用プロファイルを想定し，テスト実験を行った.

- DTMC における状態推移確率を確率の公理を満たすようランダム (一様乱数) に設定:
$$\begin{pmatrix} p_{12} = 0.13 & p_{13} = 0.14 & p_{14} = 0.15 & p_{15} = 0.11 \\ p_{16} = 0.13 & p_{17} = 0.15 & p_{18} = 0.10 & p_{1end} = 0.09 \end{pmatrix}$$

ここで, 各モジュール IDLE, NEW_JOB, UPGRADE_PRIO, BLOCK, UNBLOCK, QUANTUM_EXPIRE, FINISH, FLUSH にそれぞれ 1 ∼ 8 までの状態番号を割当て, 推移率 $p_{1,j}$ $(j = 2, \cdots, 8)$ と $p_{1,end}$ を与えている. また, $p_{j,1}$ $(j = 2, \cdots, 8)$ および

[1] http://sir.unl.edu/content/sir.php

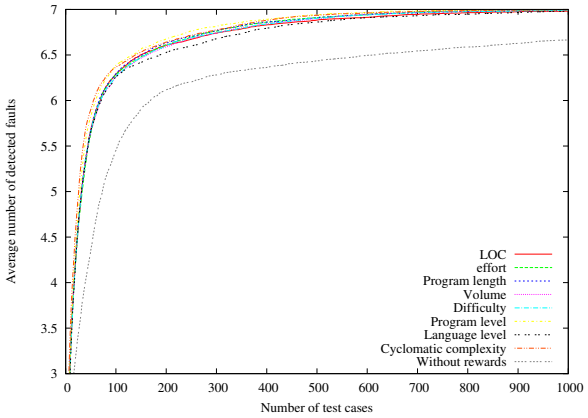
$p_{end,1}$ に関しては

$$\begin{pmatrix} p_{21}=1 & p_{31}=1 & p_{41}=1 & p_{51}=1 \\ p_{61}=1 & p_{71}=1 & p_{81}=1 & p_{end1}=0 \end{pmatrix}$$

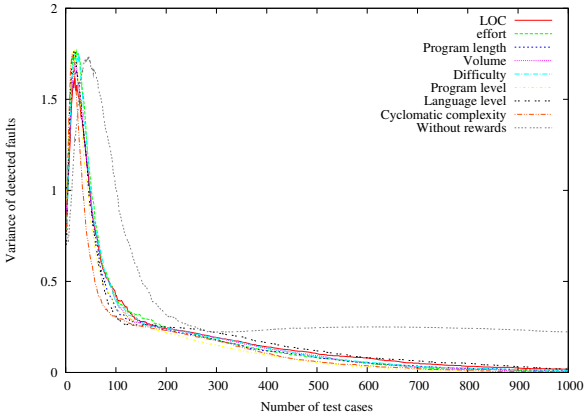
のように設定した．一旦運用プロファイルが定まると，次にソフトウェア設計メトリクスに基づいて報酬としての相対順位を決定する．表 1 において，プログラムの水準およびプログラミング言語水準に関しては，それらのメトリクス値が大きいものほどフォールトは検出されにくいと考えられる。そのため各機能のメトリクス値を算出したのち逆数をとったものを用いることで，フォールトが潜在すると想定されるモジュールをテストするためのテストケース生成を行った。各テストにおいて，実行開始状態から実行終了状態に至るステップ数（推移の回数）と，その間に訪問するモジュールのメトリクス値に対する相対順位の累積値をカウントする．次に相対順位の累積値をステップ数で割ることで, 単位ステップ当りの報酬を求める．DTMRP を 10,000 回シミュレートすることで 10,000 本の状態遷移列をテストケースとして生成し，全てのテストケースの中で単位ステップ当たりの報酬値が高い順に並び換えた 1,000 本のテストケースを，1 つのテストスイートとする．この実験では上述のテストスイートを独立に 1,000 本用意し，それらを順次実行することで，1,000 本のフォールト検出過程を観測する．各テストケース実行中にフォールトが検出されると, それを修正した後に同じテストケースが再度実行され, 終了するまで繰り返される．ここで，フォールトの修正によってメトリクス値が変化することが考えられるが，本実験では修正による報酬値の変化を考慮しないものとする．

図 4 は，想定した運用プロファイルを仮定した 1,000 本のテストスイートに対して，発見フォールト数の平均値と分散を消化テストケース数ごとにプロットしたものである．発見フォールト数の平均値はテストケースが消化されるにつれて指数関数的に増加し，潜在フォールト数の値である 7 に近づく傾向がある．しかしながら，報酬を伴わない方法では，報酬を伴う場合と比較して平均発見フォールト数が若干少なくなっている．分散に関しては，テスト初期段階で大きくなり，その後で徐々に減少する傾向を読み取ることができる．報酬を伴わない場合の分散は他と比較して大きくなり，その結果かなりのバラツキが見える．これより，何らかの報酬に基づいてテストケースを自動生成する方が，多くのフォールトを

図 4. 各テストケース生成アルゴリズムに基づいて算出された累積発見フォールト数の比較.



(i) 平均値.



(ii) 分散.

早い段階で安定して検出することが可能であることが分かった．

次に, より詳細に報酬値の種類に応じたテスト効率の比較を行う．表 2 はテスト効率の比較結果を示しており，1,000 本のテストスイートの中ですべてのフォールトを発見・除去することができたテストスイート数と，各テストスイート内で 7 つすべてのフォールトを発見・除去するまでに要したテストケース数の最小値, 平均, 分散をそれぞれ求めたものである．これより，報酬を伴わない方法で生成したテストスイートでは，全体の 67% しかすべてのフォールトを発見・除去することができなかった．一方，報酬を伴った方法で生成したテストスイートではすべて 98% 以上となり，報酬としてプログラム水準を導入した場合はすべてのフォールトを検出できたテ

表 2. 報酬値の種類に応じたテスト効率の比較.

	全フォールトを発見したテストスイート数	全フォールトを発見できなかったテストスイート数	全フォールトを発見するまでのテストケース数の統計量		
			最小	平均	分散
報酬なし	668	332	2	408.8	73343.3
コード行数	982	18	8	215.0	38800.7
プログラミング労力	994	6	3	205.2	33069.0
プログラムの長さ	994	6	6	204.2	32846.2
プログラムの大きさ	995	5	5	216.9	34648.8
プログラムの水準	998	2	4	179.0	26386.6
プログラミングの困難さ	990	10	4	212.2	34879.7
プログラミング言語水準	980	20	7	244.0	43767.7
サイクロマチック数	997	3	4	192.3	29035.1

ストスイート数は 98.8% となった．さらに，この傾向は 7 つすべてのフォールトを発見・除去するまでに要したテストケース数に対しても同様に観測され，特にプログラム水準をメトリクスとして用いた場合にテストケース数が最小となっている．以上の結果より，どのメトリクスを報酬値として用いた場合でも，報酬値を用いないテストケースよりフォールト検出力は格段に向上することが分かった．

図 5 は，生成したテストケースを用いてソフトウェアテストを実行する際に算出されるソフトウェア信頼度の平均値と分散を消化テストケースごとにプロットしたものである．信頼度の平均値はテストケース消化と共にテストの初期段階から大きく増加し，上限である 100 に近づいてゆくことがわかる．しかし，報酬を伴わない方法では報酬を伴うテストケース生成法と比較して，平均ソフトウェア信頼度の値が小さいことが示される．ソフトウェア信頼度の分散では，テスト開始時点では分散値は極端に大きいが，テストの進捗が進むにつれて 0 に収束する様子を確認できる．分散に関する結果から，報酬を用いないテストケース生成法は，報酬を用いた場合と比較して，バラツキが収束するまでにより多くのテストケース数が必要であることが示されている．この結果より，いずれかの報酬を用いてテストケースを生成することにより，テストの早期からフォールトを効率的に検出することが可能であり，ソフトウェア信頼度の値が早い段階で安定することが分かる．

次に，ソフトウェア信頼度が規定の値に達した時点でソフトウェアテストを打ち切る状況を考える．表 3 は目標信頼度を 99%とした場合に，目標を達成するのに要したテストケース数の平均と分散を比較したものであり，目標を達成した時点での発見フォールト数の平均と分散をそれぞれ求めた結果である．ソフトウェア信頼度が 99%に達するまでに必要なテストケース数について，いずれかのメトリクスを報酬として用いた場合の方が，用いない場合と比較して平均と分散が共に小さいことが確認された．中でも，サイクロマチック数を報酬として用いた場合，平均と分散が共に最小となり，目標信頼度に到達するためにはサイクロマチック数を報酬として与えることでテストケースを生成すれば良いという結論を得た．一方，ソフトウェア信頼度が 99%に達成した時点で最も多くのフォールトが発見できているメトリクスはプログラムの水準であった．しかし，99%の目標水準に達した時点での平均発見フォールト数は，どのメトリクスを用いたとしてもほぼ全てのフォールトを検出できているため，メトリクスの種類による違いはほとんどないと言ってよい．また，報酬なしの場合のテストケース生成法で発見された平均発見フォールト数は，報酬を用いたものと比較すると若干小さいことが分かる．本稿で提案した信頼度基準に基づいてテストの打ち切りを行う場合，どのメトリクスを報酬として用いたとしてもフォールト検出に有効なテストケースを生成できることが分かる．

詳細な実験結果は割愛するが，DTMC における状態推移確率を変更した実験において，全ての場合において

表 3. 信頼度基準に基づいたテスト効率の比較.

	信頼度が 99% を超えるまでにに要したテストケース数		信頼度が 99% を超えた時点での発見フォールト数	
	平均	分散	平均	分散
報酬なし	645.4	3146.3	6.5	0.2
コード行数	344.0	641.1	6.8	0.2
プログラミング労力	343.1	464.7	6.8	0.2
プログラムの長さ	375.9	517.2	6.8	0.1
プログラムの大きさ	344.6	628.7	6.8	0.2
プログラムの水準	402.6	542.9	6.8	0.2
プログラミングの困難さ	342.2	492.0	6.9	0.1
プログラミング言語水準	352.7	970.6	6.7	0.2
サイクロマチック数	286.1	368.9	6.7	0.2

報酬なしと比較した結果，報酬を与えたテストケース生成が良好な結果をもたらすことが示された．その中でも，プログラム水準とサイクロマチック数を報酬として用いた場合が，概ね良い結果を示していることが観測できた．

6. おわりに

本稿では OPBT に報酬構造を導入することで，より効率的なテストケースの自動生成が可能であることを示した．特に，報酬として与えたボリュームメトリクスおよび複雑性メトリクスの中でも，プログラム水準とサイクロマチック数の意味で，複雑度の高いモジュールを優先的にテストするようなテストケースを生成することが効率性の観点から好ましいことが分かった．今回得られた結果は一つのプログラムに対してのみ実験を行ったものであるため，実験結果が他の種類のソフトウェアに対しても成立するか調べる必要がある．よって，どのような報酬構造がテストケースの効率性に寄与するかを包括的に調べるためには，今後さらに他の種類のプログラムを用いたテスト実験を行う必要がある．また，テストケースの自動生成を通じてソフトウェア信頼度を推定することは，テスト技術者がテスト進捗状況を逐次把握するためには有益な情報に繋がるものと考えられる．その際，ソフトウェア信頼性だけでなく，期待報酬に基づいた信頼性評価尺度を求めることにより，異なる視点からテスト終了後の品質評価を行うことは興味深いものと思われる．

参考文献

[1] A. Avritzer and E. J. Weyuker, The automatic generation of load test suites and the assessment of the resulting software, *IEEE Transactions on Software Engineering*, vol. 21, no. 9, pp.705–716 (1995).

[2] A. Avritzer, E. de Souza e Silva, R. M. M. Leao and E. J. Weyuker, Automated generation of test cases using a performability model, *IET Software*, vol. 5, no. 2, pp.113–119 (2011).

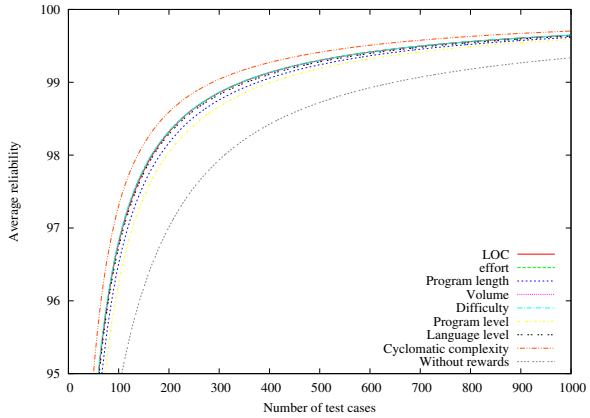
[3] T. S. Chow, Testing design modeled by finite-state machines, *IEEE Transactions on Software Engineering*, vol. 4, no. 3, pp.178–187 (1978).

[4] K. Doerner and W. J. Gutjahr, Representation and optimization of software usage models with non-Markovian state transitions, *Information and Software Technology*, vol. 42, no. 12, pp. 815–824 (2000).

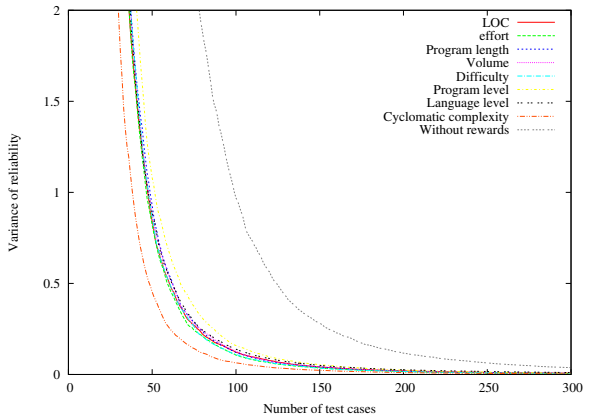
[5] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou and A. Chedamsi, Test selection based on finite state models, *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp.591–603 (1991).

[6] W. J. Gutjahr, Software dependability evaluation based on Markov usage models, *Performance Evaluation*, vol. 402, pp. 199–222 (2000).

図 5. 報酬値の種類に応じた信頼度のモーメント比較.



(i) 平均値.



(ii) 分散.

[7] 今富政喜，土肥正,“運用プロファイルと報酬構造を考慮したテストケース生成法,” ソフトウェアシンポジウム ’14 論文集, ソフトウェア技術者協会, 10 pages, 秋田 (June 9–11, 2014).

[8] C. Kallepalli and J. Tian, Measuring and modeling usage and reliability for statistical web testing, *IEEE Transactions on Software Engineering*, vol. 27, no. 11, pp. 1023–1036 (2001).

[9] T. J. McCabe, A complexity measure, *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308-320 (1976).

[10] J. D. Musa, Operational profiles in software-reliability engineering, *IEEE Software*, vol. 10, no. 2, pp. 14–32 (1993).

[11] J. D. Musa, Software-reliability-engineered testing, *IEEE Computer*, vol. 29, no. 11, pp. 61–68 (1996).

[12] J. Meyer, On evaluating the performability of degradable computing systems, *IEEE Transactions on Computer*, vol. C-29, no. 8, pp. 720–731 (1980).

[13] J. H. Poore, G. H. Walton and J. A. Whittaker, A constraint-based approach to the representation of software usage models, *Information and Software Technology*, vol. 42, pp. 825–833 (2000).

[14] S. J. Prowell and J. H. Poore, Computing system relaibility uising Markov chain usage models, *Journal of Systems and Software*, vol. 73, pp. 219–225 (2004).

[15] E. J. Weyuker, Using failure cost information for testing and reliability assessment, *ACM Transactions on Software Engineering and Methodology*, vol. 5, pp. 87–98 (1996).

[16] J. A. Whittaker and J. H. Poore, Markov analysis of software specifications, *ACM Transactions on Software Engineering and Methodology*, vol. 2, no. 1, pp. 93–106 (1993).

[17] J. A. Whittaker and M. G. Thomason, A Markov chain model for statistical software testing, *IEEE Transactions on Software Engineering*, vol. 20, no. 10, pp. 812–824 (1994).

[18] J. A. Whittaker, K. Rekab and M. G. Thomason, A Markov chain model for predicting the reliability of multi-build software, *Information and Software Technology*, vol. 42, pp. 889–894 (2000).

[19] 山田茂，高橋宗雄，ソフトウェアマネジメントモデル入門－ソフトウェア品質の可視化と評価法－，共立出版，東京，1993.

ユーザ視点に基づいたソフトウェアアップデート計画に関する一考察

岡村 寛之
広島大学大学院工学研究院
okamu @ rel.hiroshima-u.ac.jp

中原 良太
広島大学工学部第二類
（電気・電子・システム・情報系）

土肥 正
広島大学大学院工学研究院
dohi @ rel.hiroshima-u.ac.jp

要旨

本研究では，ユーザ視点からのソフトウェアの運用保守において「いつソフトウェアをアップデートするか」と言う問題を考える．特に，ソフトウェアアップデートのアクティビティを残存バグ数から数理モデルで表現し，ある種の決定問題として定式化および最適解の導出手法を議論する．さらに，実データを用いた例を通じて，最適なソフトウェアアップデート計画の特徴について議論する．

1 はじめに

情報システムの社会における重要度は年々増加しており，システムの停止により多大なコストが発生することがある．近年では OpenStack などの仮想化基盤ソフトウェアを用いたクラウドシステムが企業を中心に盛んに用いられているため，そのような基盤ソフトウェアの信頼性や可用性の確保が重要な課題となっている．

一般にソフトウェアの信頼性は，テスト工程におけるバグの発見および除去により改善されることが知られている．仮に，ソフトウェアに内在するバグや仕様上の欠陥が完全に取り除けた場合，そのソフトウェアの障害発生確率は 0 となる．しかしながら，現実問題としてバグや仕様上の欠陥を完全に取り除くことは不可能である．また，ソフトウェア開発におけるテスト費用や納期の制約により，不完全な状態でソフトウェアをリリースすることも現実には起こりえる．そのため，ベンダー（開発者）視点では，ソフトウェアをリリースした後，バグフィックスやセキュリティアップデートを提供するなどの「保守作業」が実際的に製品としてのソフトウェアの信頼性に大きく寄与している．

一方で，ユーザ視点でソフトウェアシステムの運用に注目した場合，システムの停止をできる限り少なくし，システムの可用性を高めることが望ましい．一般的に，システム停止の要因として，ソフトウェアに内在するバグが引き起こすシステム障害によるものが注目されるが，現実的に保守によるシステムも可用性を低下する要因として認識する必要がある．先に述べたように，ソフトウェアの保守作業はリリース後に提供されるバグフィクスやセキュリティアップデートなどを適用するソフトウェアのアップデート作業であり，それらを行うことでソフトウェアに内在するバグや脆弱性を取り除き，システムの障害確率を低下させることができる．しかしながら，ソフトウェアのアップデートでは再起動などの一時的なシステム停止を要する場合が多い．アプリケーションソフトウェアなどの上位層では，そのような保守による停止は大きな問題とならないが，仮想化基盤ソフトウェアなどでは，その上で稼働するすべてのアプリケーションを一旦停止する必要があるため，その影響を無視することはできない．

本論文では，ユーザ視点からシステム運用時において可用性の観点から最適な保守計画を考えるための数理モデルについて考察する．ソフトウェアの保守では二つの視点から二つの問題が存在することに注意すべきである．一つは，ベンダー視点で，ソフトウェアのバグフィックスはセキュリティアップデートを「いつリリースするか」

と言う問題である．これはパッチマネジメントの問題として，いくつかの文献において議論されている．文献 [1] では，ベンダー視点で最適なパッチマネジメントを行うための数理モデルの考察を行っている．そこでは，リスクとパッチリリース費用のトレードオフを考慮して，最適なパッチ配布スケジュール問題を取り扱っている．もう一つは，ユーザ視点から，「いつアップデートを行うか」と言う問題である．Luo ら [2] はオープンソースなど開発リポジトリからユーザがいつでも最新版を入手できる状況における最適なソフトウェアアップデート計画のためのモデルを考察している．一方で，上記の問題はベンダーが支配的な社会的なゲームととらえることも可能であり，実際，Cavusoglu ら [3] はベンダーとユーザによるゲームとして，単純な仮定の下でパッチマネジメントの議論を行っている．

本論では，Luo ら [2] と同様にユーザ視点からいつアップデートを行うかと言う問題を考える．Luo ら [2] は任意時刻において，いつでも最新のものが利用できるという状況で議論を行っているが，一般的に，プロプライエタリなソフトウェアにおいては，ベンダーから提供された最新版を利用できるだけであり，誰もが開発版を利用できるわけではない．換言すると，最新版はある固定されたタイミングで（離散的に）利用可能である．この状況を考慮すると，Luo ら [2] とは異なったソフトウェアアップデート計画のための数理モデルが考えられる．

本論文は以下の構成となる．2 節では，ソフトウェア運用と保守を単純化した数理モデルについて，モデルの仮定とふるまいを述べる．特に，ソフトウェアに内在する残存バグ数によって，ソフトウェア保守作業がモデル化されることについて言及する．3 節では，モデルにもとづいて，費用の観点から最適なソフトウェアアップデート計画を導出するための手順について説明する．4 節では，実際のソフトウェアデータを用いた数値例を通じて最適アップデート計画の特徴を示す．

2　モデルの記述

ユーザは時刻 $T_0 = 0$ から，あるソフトウェアの利用を開始し，時刻 $T_L(> T_0)$ までの利用を行うものとする．ベンダーはその期間 $[T_0, T_L]$ において N 回のソフトウェアリリースを行う．ソフトウェアリリースはバグフィクスのみであり，N 回のリリースの時刻を $T_1 < \cdots < T_N$ として表現する．ただし，$T_0 \geq T_1$ および $T_N \geq T_L$ である．ユーザは期間 $[T_0, T_L]$ における任意時刻でソフトウェアのアップデートを何回でも行うことができる．アップデートによってソフトウェアをリリースされている最新の版にすることができる．簡単のため，ここではデグレードはないものとする．

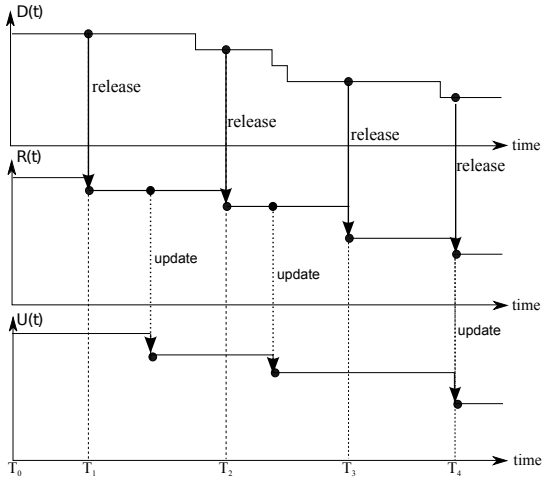


図 1. リリースおよびアップデートによる残存フォールト数のふるまい．

ソフトウェアのリリースやアップデートを残存バグ数によってモデル化する．いま，次の三つの異なる環境における残存バグ数を定義する．

- $D(t)$: 開発者が管理しているソフトウェア（開発版）に内在する残存バグ数を表す確率過程
- $R(t)$: リリースされたソフトウェア（リリース版）の残存バグ数を表す確率過程
- $U(t)$: ユーザが利用しているソフトウェア（ユーザ版）の残存バグ数を表す確率過程

開発版の残存バグ数は，バグ報告などにより継続的に減少していく．一方，リリース版は開発者が最新版をリリースするタイミングのみで残存バグ数が減少し，$D(T_i) = R(T_i),\ i = 1, \ldots, N$ の関係が成り立つ．また，ユーザ版も同様に，アップデートのみで残存バグ数が減少しする．例えば，時刻 t でユーザがアップデートを行ったとすると $R(t) = U(t)$ の関係が成り立つ (図 1 を参照)．

システムの障害はユーザ版における残存バグ数に比例した発生率 $\mu U(t)$ で発生するものと仮定する．ここで，

μ は単位時間単位バグあたりの障害発生率を表す．一方で，障害からの復旧について次の二つの方法を考える．

- 方法 1：障害が発生した場合，システム再起動で利用環境を変化させることにより障害を回避することを試みる．
- 方法 2：より最新のソフトウェアがリリースされている場合は緊急アップデートを行う．リリースされていない場合は方法 1 と同じ．

システム障害はソフトウェアに内在するバグによって引き起こされるが，再現性のある障害と再現性のない一時的な障害に分類できる．方法 1 は，メモリリークなど特定が難しいバグによる再現性のない一時的な障害に対応した回避方法である．実際,「再起動したら動くようになった」と言うのは運用においてもよく観測される事象である．一方，方法 2 は再現性のある障害の一部に効果のある手法であり，観測された再現性のある障害が最新版で修正されていれば本質的な回避が行える．ただし，モデルの簡単化のため，いずれの方法も，どのようにして障害を回避するかと言う詳細については議論しない．つまり，もし行いたい作業が再現性のある障害で，何度やっても行えない場合はその作業を諦めるような回避も含んでいることに注意する．

3 最適ソフトウェアアップデート計画

最適なソフトウェアアップデート計画を考えるため，次の費用パラメータを考える．

- c_u: 一回のアップデートに要する費用
- c_e: 一回の緊急アップデートに要する費用
- c_f: 一回のシステム再起動による障害回避に要する費用

先に述べたように，ソフトウェアアップデートではシステム停止を伴う場合があるため，c_u はシステム停止による機会損失とアップデートそのものの労力による費用を含む．また，c_f には再起動によるシステム停止による機会損失に加えて，作業を諦めることによる回避のペナルティ費用も含む．上記の費用を，アップデートによるシステムの停止時間，緊急アップデートによるシステムの停止時間，システム再起動によるシステム停止時間として，それらの停止時間を最小化する，本質的に可用性を最大化する問題も取り扱うことができる．

上記の費用構造のもとでは，数学的に総費用を最小にする観点から，ユーザは必ず開発者が最新版をリリースしたと同時にアップデートを実行するかしないかを決定し，それ以外の時刻ではアップデート（緊急アップデートを除く）を行わないことがわかる．つまり，アップデート計画問題は，最新版のリリース時刻列 $T_1, T_2, \ldots, T_N$ でアップデートを行う／行わないと言う行動を決定するマルコフ決定過程により記述できる．

費用を最小にする最適なアップデート計画を求めるために，開発者が x 番目のリリースを行った時刻 T_x において，ユーザが $y(< x)$ 番目にリリースされたソフトウェアを利用している状況を考える．このとき，次の二つの費用を定義する．

- $W_P(x|y)$: x 番目にリリースされたソフトウェアへのアップデートを行い，それ以降は最適なアップデート計画に従ったときの時刻 T_L までの総期待費用．
- $W_{\overline{P}}(x|y)$: x 番目にリリースされたソフトウェアへのアップデートを行わずに，それ以降は最適なアップデート計画に従ったときの時刻 T_L までの総期待費用．

障害回避を方法 1 によって行う場合，最適性原理から $y = 0, \ldots, N$, $x = y+1, \ldots, N$ に対して以下の最適性方程式を得る．

$$V(x|y) = \min\{W_P(x|y), W_{\overline{P}}(x|y)\}, \tag{1}$$

$$W_P(x|y) = c_u + c_f\mu\Delta T_{x+1}\xi_x + V(x+1|x), \tag{2}$$

$$W_{\overline{P}}(x|y) = c_f\mu\Delta T_{x+1}\xi_y + V(x+1|y). \tag{3}$$

ここで $\Delta T_{x+1} = T_{x+1} - T_x$ および $\xi_x = \mathrm{E}[R(T_x)]$ である．さらに，$T_{N+1} = T_L$ および $V(N+1|\cdot) = 0$ であることに注意する．

一方，障害回避を方法 2 によって行う場合，最適性原理から $y = 0, \ldots, N$, $x = y+1, \ldots, N$ に対して以下の最適性方程式を得る．

$$V(x|y) = \min\{W_P(x|y), W_{\overline{P}}(x|y)\}, \tag{4}$$

$$W_P(x|y) = c_u + c_f\mu\Delta T_{x+1}\xi_x + V(x+1|x), \tag{5}$$

$$\begin{aligned} W_{\overline{P}}(x|y) = {} & \int_0^{\Delta T_{x+1}} \left(c_e + c_f\mu\left(\Delta T_{x+1} - t\right)\xi_x\right) dF_y(t) \\ & + V(x+1|x)F_y(\Delta T_{x+1}) \\ & + V(x+1|y)(1 - F_y(\Delta T_{x+1})). \end{aligned} \tag{6}$$

表 1. IE9 のバグ発見数データ.

デバッグ期間（月）	その期間でのバグ発見数
2	11
2	7
2	8
2	3
2	4
2	5
2	13
1	2
1	4
1	5
1	3
1	3
1	13
1	9
1	2
1	11
1	19
1	18
1	11
1	10
1	9

ここで $F_y(t) = 1 - e^{-\xi_y t}$ である.

上記の二つの最適性方程式はいずれも $y = N, x = N$ から後ろ向きに $V(x|y)$ を算出することができ，得られた $V(x|y)$ をもとにして，リリース時刻列 $T_1, \ldots, T_N$ でアップデートを行うかどうかの最適な行動を導出できる.

4 数値例

ここでは，Microsoft 社の Web ブラウザである Internet Explorer 9 (IE9) の 28 ヶ月間（2011/6/14 から 2013/10/8 まで）のバグ修正レポート [1] から収集したデータを用いた例を示す.

表 1 は収集した月ごとのバグ発見数データを示している．これを用いて，開発版における残存バグ数を表す確率過程の推定を行う．いま，開発版における累積バグ発見個数を表す確率過程を $N(t)$ を次の非同次ポアソン過程（NHPP）と仮定する.

$$P(N(t) = n) = \frac{(\omega G(t))^n}{n!} e^{-\omega G(t)}. \tag{7}$$

ここで，$\mathrm{E}[N(t)] = \omega G(t)$ は平均値関数と呼ばれ，時刻 t で発見されるバグの期待値を表す．実際，式 (7) は NHPP に基づいたソフトウェア信頼度成長モデル（以下，NHPP モデル）を表しており，リリース判定などで利用されている [4]．また，ω はソフトウェアに内在する初期バグ数の期待値，$G(t)$ は個々のバグの発見時刻に対する累積確率分布関数を表している．いま，開発版の残存バグ数 $D(t)$ に対して $N(t) + D(t)$ がソフトウェアに内在する総バグ数（初期バグ数）であるため,

$$P(D(t) = n) = \frac{(\omega \overline{G}(t))^n}{n!} e^{-\omega \overline{G}(t)} \tag{8}$$

であることが示される．ここで $\overline{G}(t) = 1 - G(t)$ であり，$G(0) = 1$ から $G(\infty) = 0$ まで単調に減少する関数である．つまり，データから NHPP モデルを推定することで，開発版の残存バグ数の平均値を推定することができる.

NHPP モデルに対しては，伝統的に最尤法および情報量規準によるモデル選択が行われる．NHPP モデルでは，バグ発見時刻分布 $G(t)$ にどのような確率分布を適用するかで様々なモデルが表される．つまり，具体的な手順は，まず，候補となるいくつかの NHPP モデルに対して最尤推定によるパラメータ推定を行い，次に，以下の赤池情報量規準（AIC）[5] を算出し，最も AIC が小さくなるモデルを最良のモデルとして選ぶ.

$$\mathrm{AIC} = -2(\text{最大対数尤度}) + 2(\text{自由パラメータ数}). \tag{9}$$

上述のデータに対して，これらの推定作業を SRATS [2] で行った．SRATS では 11 種類の候補となるモデルに対する最尤法と情報量規準の算出を行う．表 2 は SRATS によって得られた各モデルの AIC を示している．この結果から AIC を最小にする切断最小値分布モデルを利用し，以下の平均残存バグ数を推定した.

$$\mathrm{E}[D(t)] = \omega H(-t)/H(0) \tag{10}$$

$$H(t) = \exp\left(-\exp\left(-\frac{t-\beta}{\alpha}\right)\right), \tag{11}$$

$$\omega = 1817.9, \quad \alpha = 14.99, \quad \beta = 60.28. \tag{12}$$

また，図 2 に推定した残存バグ数の平均値を示す.

[1] マイクロソフトセキュリティ情報
https://technet.microsoft.com/ja-jp/security/bulletin

表 2. 推定結果.

NHPP モデル	AIC
指数分布モデル	170.09
ガンマ分布モデル	155.31
パレート分布モデル	178.58
切断正規分布モデル	134.82
対数正規分布モデル	167.85
切断ロジスティック分布モデル	133.61
対数ロジスティック分布モデル	154.85
切断最大値分布モデル	135.93
対数最大値分布モデル	177.82
切断最小値分布モデル	133.42
対数最小値分布モデル	154.45

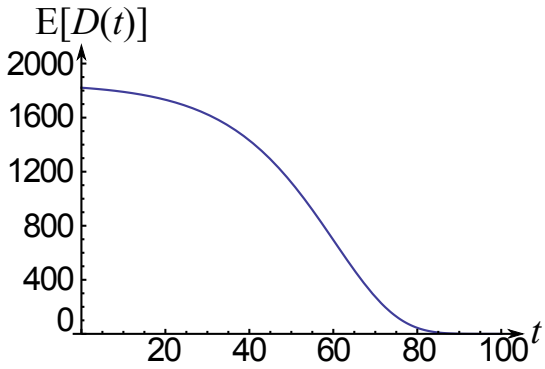


図 2. 推定された期待残存バグ数のふるまい.

残存バグ数の減り方が最適なアップデート計画に与える影響をみるため，ユーザが IE9 を利用する期間として次の三つの期間に対する最適なアップデート計画をそれぞれ算出する.

- 期間 1：0ヶ月から 24ヶ月目までの 24ヶ月間（$T_0 = 0, T_L = 24$）.
- 期間 2：32ヶ月から 56ヶ月目までの 24ヶ月間（$T_0 = 32, T_L = 56$）.
- 期間 3：64ヶ月から 88ヶ月目までの 24ヶ月間（$T_0 = 64, T_L = 88$）.

また，IE9 は，一ヶ月あるいは二ヶ月毎に定期的にアップデート用の修正プログラムがリリースされているため，上記のいずれの期間においても $\Delta T_x = 1, x = 1, \ldots, 23$ とした．また，単位時間単一バグ当たりの障害発生率を $\mu = 0.1$，アップデート一回当たりに要する費用を $c_u = 1.0$，障害を再起動による回避に要する費用を $c_f = 1.0$ とした．また，緊急アップデートに要する費用は $c_e = 1.5, 2.0, 2.5$ の三つの場合を考える.

表 3 は，方法 1 と方法 2 それぞれの障害回避方法のもとで，最適なアップデート計画を行った時の最小総期待費用を表している．表中の括弧内の値は対応する期間で行う計画されたアップデート回数を表している．各期間における総期待費用を比較すると，残存バグ数が少なく且つその期間で発見されるバグ数が少ない期間（期間 3）が最も少ない費用であることがわかる．また，方法 1 でアップデートを行う回数に着目すると，期間 2 が最も多く，その期間で発見されるバグ数とアップデート回数に強い関連があることがわかる．一方で，方法 2 に着目すると，緊急アップデートに要する費用 c_e が一回のアップデートと再起動による障害回避に要する費用の和 $c_u + c_f$ より小さい場合，計画されたアップデート回数が極端に少ないことが読み取れる．これは，アップデートのほとんどを緊急アップデートで対応することを意味している．方法 2 では，緊急アップデートを行う機会があるため，方法 1 ほど発見バグ数とアップデート回数の間に強い相関が見られなかった．また，方法 1 と方法 2 の総期待費用を比較すると，c_e が $c_u + c_f$ より小さい場合にだけ，方法 2 の総期待費用が方法 1 の総期待費用より低くなっている．つまり，緊急アップデートは，通常のアップデートと障害回避に要する費用が緊急アップデートに要する費用より小さい場合だけ効果的であり，その他の場合は緊急的にアップデートする必要性がないことも読み取れる.

表 3. 最小総期待費用.

		期間 1	期間 2	期間 3
方法 1		4273(10)	3155(23)	435(19)
方法 2	$c_e = 1.5$	4267(0)	3144(0)	427(4)
	$c_e = 2.0$	4279(23)	3155(23)	434(19)
	$c_e = 2.5$	4279(23)	3155(23)	435(21)

5 まとめと今後の課題

本研究では，ユーザ視点からソフトウェアアップデートに関する数理モデルを構築し，最適なアップデート計

[2] http://www.srat-app.com/SRATS/

画を導出するための問題設計および最適解の導出を行った．また，数値例では実データに基づいた分析を行い，最適アップデート計画に対する特徴について議論した．

本研究では，いくつかの重要な点をモデル化のために簡単化している．一つ目は，デグレードの問題であり，実際のアップデート作業では，アップデートすることにより引き起こされる障害が存在する．ユーザ視点で，すぐにアップデートを適用しない理由の一つとしてあげられるため，実際の計画においては重要な要因となり得る．もう一つは，アップデート内容に関する問題である．例えば，重大なセキュリティ上のアップデートであれば，緊急的かつ優先的に対応する必要がある．しかしながら，ここで行ったモデル化はこれらの要因を考えないこととしており，今後の課題として，これらの要因を考慮した上で，アップデート計画を行うための意思決定モデルの構築を行う予定である．

参考文献

[1] H. Okamura, M. Tokuzane and T. Dohi, “Optimal security patch release timing under non-homogenous vulnerability-discovery processes”, Proc. Int. Symp. on Software Reliab. Eng. (ISSRE’09), 120–128, 2009.

[2] C. Luo, H. Okamura and T. Dohi, “Optimal planning for open-source software updates”, in submission.

[3] H. Cavusoglu, H. Cavusoglu, and J. Zhang, “Security patch management: Share the burden or share the damage?” *Management Science*, vol. 54, no. 4, pp. 657–670, 2008.

[4] 山田，ソフトウェア信頼性モデル，日科技連，1994.

[5] H. Aiakike, Information theory and an extension of the maximum likelihood principle, Proc. 2nd Int. Symp. on Information Theory (PN. Petrov and F. Csaki, eds.), 267–281, 1973.

ソフトウェア開発工数見積もりにおける外れ値の実験的評価

小野 健一
奈良先端科学技術大学院大学
ono.kenichi.ob4@is.naist.jp

角田 雅照
近畿大学
tsunoda@info.kindai.ac.jp

門田 暁人
岡山大学
monden@okayama-u.ac.jp

松本 健一
奈良先端科学技術大学院大学
matumoto@is.naist.jp

要旨

大規模なソフトウェア開発においてプロジェクト管理は必要不可欠であり，そのためにはソフトウェア開発工数を正確に見積もることが重要となる．工数見積もりにおいて外れ値は除去されることが多い．ただし，外れ値がどの程度見積もり精度に影響するのかは明らかではない．そこで本研究では，データセットに外れ値を実験的に追加し，外れ値の見積もり精度に対する影響を分析した．外れ値の割合，外れ値の程度，外れ値の存在する変数，外れ値の存在するデータセットの 4 つを変化させて見積もり精度を評価した結果，開発規模に外れ値が含まれていても，外れ値の割合が 10%で，外れ値の程度が 100%の場合，あまり見積もり精度が低下しないことなどがわかった．

1. はじめに

近年，ソフトウェアはより高機能が求められており，その要求にこたえるためにソフトウェアの規模が大規模化している．大規模なソフトウェアを開発するためには，多くの工数（コスト）が必要となる．このようなソフトウェアを開発するプロジェクトでは，スケジュールやコストに関する管理を行わずにプロジェクトを成功させることは困難である．よって，大規模な開発プロジェクトにおいてプロジェクトの失敗を避けるためには，スケジュールやコストの管理が必要不可欠となる．ソフトウェア開発工数の見積もりは，それらの管理の基礎となるものである．

スケジュールやコスト管理を正確に行うためには，ソフトウェア開発の工数を高い精度で見積もることは非常に重要であり，そのため，これまでさまざまな定量的工数見積もり手法が提案されてきた[1][14][17]．定量的工数見積もり方法として，重回帰分析やプロジェクト類似性に基づく工数見積もり[13]などがあげられる．定量的に工数を見積もるためには，過去のプロジェクトで収集されたデータを見積もりの根拠データとし，見積もり対象のプロジェクトにおいて既知のデータ，例えば開発言語や開発規模などを用いて，開発工数を見積もる．

ただし定量的工数見積もりを行う場合に，見積もりの根拠データなどに外れ値と呼ばれるデータが存在する可能性があり，それにより，工数見積もりの精度が低下する可能性がある．例えば，非常に大量の作業手戻りが発生したプロジェクトの場合，同規模の他のプロジェクトに比べると，開発工数が極端に大きくなる．また，データの収集や集計時にミスが発生したプロジェクトの場合，実際のデータと異なるものが記録される．

外れ値により工数見積もりの精度が低下することを防ぐため，外れ値除去法が適用される場合がある．外れ値除去法とは，見積もりの根拠データに含まれるプロジェクトが外れ値であるか検証し，外れ値と判断されたものをデータから除去する手法である．例えば，重回帰分析によりソフトウェア開発工数を見積もる場合，Cook の距離を用いて外れ値を除去すること多い（[7]など）．類似性に基づく見積もり方法に適した外れ値除去法もいくつか提案されている[6]．

本研究では，定量的にソフトウェア開発工数を見積もる場合に，データに含まれる外れ値がどの程度影響を与えるのかを分析する．そのために，本研究ではデータセットに外れ値を実験的に追加することを行う．具体的には，目的変数である工数と，説明変数として最も重要な開発工数に外れ値を含める．実験では以下の 2 つのパラメータを変更した場合の見積もり精度の変化を確かめる．

- 外れ値が含まれる割合
- 外れ値の程度

外れ値が含まれる割合とは，例えばこのパラメータを 10%とした場合，根拠データ 100 件のうち 10 件に外れ値

を含めることである. 外れ値の程度とは, 例えばこのパラメータを 100%とした場合, 工数が 100 人時のデータを 200 人時とすることにより外れ値を含めることである. これにより, 工数見積もりを行う際, 各変数における外れ値の影響をどの程度考慮すべきかの参考になると考える.

さらに, データに外れ値が含まれているパターンについても変更して分析する. 具体的には以下のパターンでデータに外れ値を含める.

- 見積もり根拠データ:外れ値あり, 見積もり対象データ:外れ値なし
- 見積もり根拠データ:外れ値あり, 見積もり対象データ:外れ値あり
- 見積もり根拠データ:外れ値なし, 見積もり対象データ:外れ値なし

外れ値は見積もり根拠データだけではなく, 見積もり対象プロジェクトにも含まれている場合がある. 一般に, 見積もり対象プロジェクトの外れ値を除去することは難しいため, これまで見積もり対象プロジェクトに外れ値が含まれている場合の見積もり精度への影響は考慮されていない. ただし, 実際に定量的見積もりを行う場合, 見積もり対象データに外れ値が含まれることがあるため, これを考慮せずに見積もりモデルを評価することは, モデルの見積もり精度を過大に評価している可能性がある. この分析により, より現実的な見積もり精度を評価するためにはどうすべきかを考察することができると考える.

実験では, 定量的工数見積もり方法として重回帰分析とプロジェクト類似性に基づく工数見積もりを採用する.

2. 定量的工数見積もり

本研究では, 定量的工数見積もり方法として用いられる重回帰分析と類似性に基づく工数見積もりにおける, 外れ値の影響を評価する. 以降では, それぞれの見積もり方法について説明する.

2.1. 重回帰分析に基づく工数見積もり

重回帰分析による見積もりは, ソフトウェア開発工数を定量的に見積もる際に広く用いられる方法である. 重回帰分析では, 過去のプロジェクトにおいて記録されたデータを見積もりの根拠データとして, 最小二乗法により見積もりモデルが1つ構築される. 開発工数をy, 開発規模などの説明変数を$x_1, x_2, \ldots, x_k$とすると, 重回帰分析よる積モデルは以下のようになる.

表 1 Analogy 法による工数見積もりにおいて用いるデータセット

Table 1 A dataset used on analogy based effort estimation

	変数 1	変数 2	…	変数 j	…	変数 l
p_1	m_{11}	m_{12}	…	m_{1j}	…	m_{1l}
p_2	m_{21}	m_{22}	…	m_{2j}	…	m_{2l}
…	…	…		…		…
p_i	m_{i1}	m_{i2}	…	m_{ij}	…	m_{il}
…	…	…		…		…
p_k	m_{k1}	m_{k2}	…	m_{kj}	…	m_{kl}

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \varepsilon \quad (1)$$

ここで, β_0は回帰定数, $\beta_1, \beta_2, \ldots, \beta_k$は偏回帰係数, εは誤差項である. 重回帰分析により見積もりモデルを構築する場合, 経験的に説明変数の5から10倍のプロジェクト数が必要であるといわれている.

重回帰分析を用いてソフトウェア開発工数を見積もる場合, モデルの構築前に比例尺度の説明変数, 目的変数に対し, 対数変換が適用される場合がある. 対数変換を適用した重回帰分析はソフトウェア開発のデータの特徴を表すのに適していることが指摘されているため[10], 本研究でも対数変換を適用した重回帰分析により見積もりモデルを構築する.

2.2. プロジェクト類似性に基づく工数見積もり

プロジェクト類似性に基づく工数見積もり(Analogy 法)は, 事例ベース推論(Case Based Reasoning; CBR)に基づいた見積もり方法である. CBR は人工知能の分野で研究されてきた問題解決のための方法であり, Shepperd ら[13]が CBR をソフトウェアプロジェクトの開発工数見積もりに適用することを提案した. CBR では, 蓄積された過去の事例の中から, 問題を解決したい現在の事例と類似したもの抽出し, その解決方法を適用する. CBR の基礎となる考え方は「類似した問題は類似した解決方法を採っている」というものである. Analogy 法では, 類似するプロジェクト(開発規模や対象業種などの特徴が互いに似たプロジェクト)は, 工数も互いに似た値を取るであろうと仮定し, 類似プロジェクトの工数に基づいて対象プロジェクトの工数を見積もる.

Analogy 法による工数見積もりでは, 表 1に示す $k \times l$ 行列で表されるデータセットを入力として用いる. 図中, p_i は i 番目のプロジェクトを表し, m_{ij} はプロジェクト p_i の j 番目の変数を表す. すなわち, 行がプロジェクト, 列が変数を表している. ここで p_a を見積もり対象のプロジェクト,

$\hat{m}_{ab}$を m_{ab} の見積もり値とする. Analogy 法による工数見積もりは, 以下の手順に従って行われる.

1. 互いに異なる各変数の値域を統一する. 本研究では変数 m_{ij} の正規化された値 m'_{ij} を計算する際, 以下の式を用いて, 変数の値域を[0, 1]に揃える.

$$m'_{ij} = \frac{m_{ij} - min(m_j)}{max(m_j) - min(m_j)} \quad (2)$$

ここで, $max(m_j)$ と $min(m_j)$はそれぞれ j 番目の変数の最大値, 最小値を表す. この計算方法は, 値域を変換する際に広く用いられる方法の 1 つである[15].

2. 見積もり対象プロジェクト p_a と他のプロジェクト p_i との類似度 $sim(p_a, p_i)$を求める. p_a と p_i が持つ変数を要素とする 2 つのベクトルを作成し, ベクトルのなす角のコサインを用いて類似度を計算する. 見積もり対象プロジェクト p_a と他のプロジェクト p_i との類似度 $sim(p_a, p_i)$を次式によって計算する.

$$sim(p_a, p_i) = \frac{\sum_{j \in M_a \cap M_i} (m'_{aj} - avg(m'_j))(m'_{ij} - avg(m'_j))}{\sqrt{\sum_{j \in M_a \cap M_i} (m'_{aj} - avg(m'_j))} \sqrt{\sum_{j \in M_a \cap M_i} (m'_{ij} - avg(m'_j))}} \quad (3)$$

ここで M_a と M_i はそれぞれプロジェクト p_a と p_i で計測された(未欠損の)変数の集合を表し, $avg(m'_j)$は j 番目の変数の平均値を表す. $sim(p_a, p_i)$の値域は[-1, 1]である.

3. 類似度 $sim(p_a, p_i)$を用いて, プロジェクト p_a の変数の見積もり値 $\hat{m}_{ab}$を計算する. 本研究では, 見積もり値計算時に, プロジェクトの規模を補正する$amp(p_a, p_i)$を乗じた値で加重平均を行う[17].

$$\hat{m}_{ab} = \frac{\sum_{i \in k-nearestProjects} (m_{ib} \times amp(p_a, p_i) \times sim(p_a, p_i))}{\sum_{i \in k-nearestProjects} sim(p_a, p_i)} \quad (4)$$

$$amp(p_a, p_i) = \frac{fp_a}{fp_i} \quad (5)$$

ここで, fp_a, fp_i はそれぞれプロジェクト p_a, p_i におけるソフトウェアの開発規模を示す.

3. 外れ値除去法

外れ値除去法とは，データセット中のプロジェクトが外れ値であるか検証し，外れ値と判断されたプロジェクトをデータセットから除去する処理である. これまで，ソフトウェア開発工数見積もりにおいて，重回帰分析適用時にCookの距離を用いて外れ値を除去することが行われている（[7]など）．Cook の距離を用いた除去法は，重回帰分析を行い，モデルの係数に大きな影響を与えるプロジェクトを外れ値とみなす．Cook の距離は，あるプロジェクトをモデルの推定の計算から除外した場合に，すべてのプロジェクトの残差がどの程度変化するかを示す距離であり，Cook の距離が 4/n（n はデータ件数）以上となるプロジェクトを外れ値とみなし[16], データセットから除外する．Cook の距離が大きいときは，回帰統計量の計算からプロジェクトを除外したことが係数を実質的に変化させたことを示す[16].

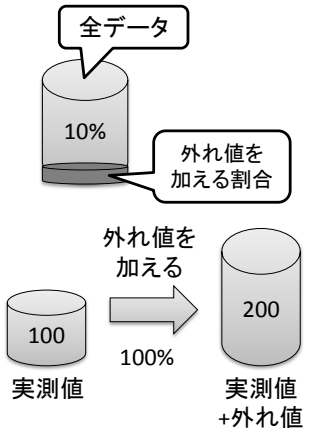


図 1 外れ値の割合, 外れ値の程度
Figure 1 Ratio of Outliers, Degree of Outliers

Keung ら[6]は，Analogy 法への適用を前提とした，Mantel 相関を用いた除去法を提案している．Mantel 相関を用いた除去法は，説明変数が類似しているにもかかわらず，目的変数が類似していないケースを外れ値とみなす手法である．通常の相関係数は 2 つの変数の関連の強さを表すのに対し，Mantel 相関は（説明変数のセットと目的変数など）2 つの変数セット間の関連の強さを表す．Mantel 相関により「プロジェクトの開発規模や開発期間（説明変数）が似ていると，開発工数（目的変数）も似ているかどうか」を調べることができる．Mantel 相関は，各プロジェクトのペアについて，説明変数に基づくユークリッド距離と目的変数に基づくユークリッド距離を計算し，それらの相関係数を計算することにより求められる.

本研究では，これらの外れ値除去法を適用しない場合における，外れ値の影響を評価する.

4. 実験

4.1. 外れ値に関する 4 つの観点

本研究では，(1) 外れ値の割合，(2) 外れ値の程

度，（3）外れ値の存在する変数，（4）外れ値の存在するデータセットの 4 つに着目し，これらが異なると工数見積もり精度に対する外れ値の影響も異なると仮定した．そこで，これらを変化させて外れ値の影響を分析した．

（1）外れ値の割合：全データ件数に対する外れ値の割合を示す．例えば，データセットに含まれるプロジェクトの件数が 100 件あり，うち 10 件を実験的に外れ値に置換する場合，外れ値の割合は 10%となる（図 1）．

（2）外れ値の程度：変数に記録されている値が，実際の値よりもどの程度ずれているかを示す．例えば，あるプロジェクトの工数が 200 人月と記録されており，これを実験的に 400 人月に置換した場合，外れ値の程度は 100%（|400-200| / 200）となる（図 1）．

（3）外れ値の存在する変数：外れ値は開発工数などの説明変数と目的変数である開発工数の両方に含まれると考えられる．そこで実験では，目的変数と，目的変数に最も影響が強いと考えられる説明変数である，開発規模に外れ値を加える．

（4）外れ値の存在するデータセット：一般に，外れ値除去法は工数見積もりの根拠データに含まれる外れ値を除去することを想定しているが，見積もり対象データにも外れ値が含まれる（見積もり対象データが外れ値となる）可能性がある．見積もりの根拠データに外れ値除去法を適用する場合などを考慮すると，外れ値の存在するパターンは以下の 3 つに分類されると考えられる．

- 見積もり根拠データ:外れ値あり，見積もり対象データ:外れ値なし
- 見積もり根拠データ:外れ値あり，見積もり対象データ:外れ値あり
- 見積もり根拠データ:外れ値なし，見積もり対象データ:外れ値なし

4.2. 実験手順

データセットに含まれる外れ値が工数見積もりの精度に与える影響を，以下の手順により分析した．

（1）データセットを無作為に半数ずつ分け，一方をフィットデータ，もう一方をテストデータとする．フィットデータは見積もりの根拠とするデータセット（過去プロジェクトに該当する），テストデータは見積もり対象とするデータセット（開発中のプロジェクトに該当する）である．

（2）実験的にデータセットに外れ値を加える．外れ値は4.1節で述べた方法により追加する．予備分析の結果より，開発規模に対しては外れ値の割合を 10%及び 20%, 外れ値の程度を 100%として分析した．工数に対しては外れ値の割合を 10%，外れ値の程度を 50%及び 100%として分析した．外れ値の割合を 10%から 20% とした理由は，一般的に想定される外れ値の割合よりも若干多めに設定することにより，最悪の場合における外れ値の影響を確かめるためである．外れ値の程度を 50%から 100%とした理由は，これよりも大きい外れ値はデータの転記ミス以外では発生する可能性が低いと想定したためである．

（3）外れ値が含まれたデータを用いて，見積もりモデルを構築し，工数を見積もり，見積もり精度の評価指標を算出する．

（4）（1）～（3）を 10 回繰り返し行い，評価指標

表 2　重回帰分析における, 外れ値と見積もり精度の関係（工数, 10%, 50%）
Table 2　Relationships between outliers and estimation accuracy on multiple regression analysis (effort, 10%, 50%)

フィット データ	テスト データ	*AE* 平均	*AE* 中央値	*MRE* 平均	*MRE* 中央値	*BRE* 平均	*BRE* 中央値
外れ値あり	外れ値なし	-169.99	-173.69	**-5.11%**	-1.90%	**-6.83%**	-4.46%
外れ値なし	外れ値あり	-179.67	-86.01	0.17%	-0.69%	-3.51%	-1.70%
外れ値あり	外れ値あり	-342.93	-231.78	-4.54%	-2.19%	**-10.02%**	**-5.94%**

表 3　重回帰分析における, 外れ値と見積もり精度の関係（工数, 10%, 100%）
Table 3　Relationships between outliers and estimation accuracy on multiple regression analysis (effort, 10%, 100%)

フィット データ	テスト データ	*AE* 平均	*AE* 中央値	*MRE* 平均	*MRE* 中央値	*BRE* 平均	*BRE* 中央値
外れ値あり	外れ値なし	-260.37	-207.64	**-10.80%**	-3.74%	**-13.55%**	**-7.86%**
外れ値なし	外れ値あり	-469.14	-151.18	0.82%	-1.68%	**-7.55%**	-2.58%
外れ値あり	外れ値あり	-711.48	-430.79	**-8.82%**	**-6.50%**	**-19.77%**	**-10.98%**

表 4　重回帰分析における，外れ値と見積もり精度の関係（FP，10%，100%）
Table 4　Relationships between outliers and estimation accuracy on multiple regression analysis (FP, 10%, 100%)

フィット データ	テスト データ	*AE* 平均	*AE* 中央値	*MRE* 平均	*MRE* 中央値	*BRE* 平均	*BRE* 中央値
外れ値あり	外れ値なし	-18.03	19.90	4.00%	-1.23%	-0.09%	-0.70%
外れ値なし	外れ値あり	-110.67	-109.37	-3.94%	-2.00%	-2.58%	-3.59%
外れ値あり	外れ値あり	-107.91	-60.70	1.28%	-2.40%	-1.97%	-2.17%

表 5　重回帰分析における，外れ値と見積もり精度の関係（FP，20%，100%）
Table 5　Relationships between outliers and estimation accuracy on multiple regression analysis (FP, 20%, 100%)

フィット データ	テスト データ	*AE* 平均	*AE* 中央値	*MRE* 平均	*MRE* 中央値	*BRE* 平均	*BRE* 中央値
外れ値あり	外れ値なし	-101.19	-99.18	1.03%	-2.75%	**-6.97%**	-3.53%
外れ値なし	外れ値あり	-135.70	-176.39	**-8.09%**	-1.55%	**-5.48%**	0.65%
外れ値あり	外れ値あり	-100.37	-138.22	-0.17%	-2.52%	**-7.00%**	-4.61%

の平均値と中央値を算出し，外れ値を全く加えなかった場合と比べて，どの程度評価指標の値が変化したのかを確かめる．

4.3. データセット

Desharnais データ[3]を用いて工数見積もりの精度評価した．Desharnais データは，Desharnais によって収集されたカナダのソフトウェア開発企業における 80 年代のデータセットである．このデータセットは，ソフトウェア開発工数見積もりの研究において広く用いられており[6][13]，かつ無料で入手できるデータセットの中ではデータ件数，説明変数の数ともに多いために利用した．データの収集時期は古いが，例えば近年に発表された研究[6]でも用いられている．

データセットに含まれる 81 件のプロジェクトのうち，欠損値が含まれるプロジェクトを除いた 77 件を用いた．説明変数は，開発年度と調整済み FP（ファンクションポイント），開発期間を除いた 7 つの変数（開発チームの経験年数，プロジェクトマネージャの経験年数，トランザクション数，未調整 FP（以降 FP と略す），調整係数，開発言語，エンティティ数）である．このうち開発言語はカテゴリ変数のため，ダミー変数化して用いた．

4.4. 評価尺度

工数見積もりの精度評価指標として，*AE*，*MRE* (Magnitude of Relative Error)[2]，*BRE* (Balanced Relative Error)[8]の 3 つの指標の平均値と中央値を用いた．工数の実測値を x，見積もり値を $\hat{x}$ とするとき，それぞれの指標は以下の式により求められる．

$$AE = |x - \hat{x}| \tag{6}$$

$$MRE = \frac{|x - \hat{x}|}{x} \tag{7}$$

$$BRE = \begin{cases} \dfrac{(\hat{x} - x)}{x}, \hat{x} - x \geq 0 \\ \dfrac{(x - \hat{x})}{\hat{x}}, \hat{x} - x < 0 \end{cases} \tag{8}$$

それぞれの指標の値が小さいほど，工数見積もりの精度が高いことを示す．直感的には *MRE* は実測値との相対誤差であるといえる．ただし，*MRE* は過大見積もりに対し，アンバランスな評価になるという問題がある．（見積もり工数が 0 以上の）過少見積もりの場合，*MRE* は最大でも 1 にしかならない（例えば実測値が 1000 人時，見積もり値が 0 人時の場合，*MRE* は 1 となる）．そこで本研究では，*MRE* に加え，過大見積もりと過少見積もりをバランスよく評価する指標[9]である *BRE* を評価指標に用いた．

フィットデータ，テストデータともに外れ値を含んでいないモデルを基準モデルとし，基準モデルと評価対象のモデルの評価指標の差により外れ値の影響を評価した．評価指標が負の値の場合，見積もり精度が基準モデルよりも低下したことを示す．

5. 実験結果

表 2から表 9に，各評価指標の基準モデルとの差を示す．表見出しのカッコ内に，外れ値が含まれる変数，外れ値の割合，外れ値の程度を示す．表における評価指標の値は 10 個のテストデータから得られた値を平均したものである．基準モデルよりも *MRE*，*BRE* が 5%以上悪化した場合は無視できない見積もり精度の悪化であると考え，これらを太字で示した．なお，ここで 5%を基準と

表 6　Analogy 法における，外れ値と見積もり精度の関係（工数，10%，50%）
Table 6　Relationships between outliers and estimation accuracy on analogy based estimation (effort, 10%, 50%)

フィット データ	テスト データ	*AE* 平均	*AE* 中央値	*MRE* 平均	*MRE* 中央値	*BRE* 平均	*BRE* 中央値
外れ値あり	外れ値なし	-111.65	-25.49	**-5.67%**	-2.07%	**-5.00%**	-2.16%
外れ値なし	外れ値あり	-106.61	26.30	2.80%	1.32%	0.91%	1.32%
外れ値あり	外れ値あり	-199.98	-13.79	-2.41%	0.20%	-3.52%	-0.39%

表 7　Analogy 法における，外れ値と見積もり精度の関係（工数，10%，100%）
Table 7　Relationships between outliers and estimation accuracy on analogy based estimation (effort, 10%, 100%)

フィット データ	テスト データ	*AE* 平均	*AE* 中央値	*MRE* 平均	*MRE* 中央値	*BRE* 平均	*BRE* 中央値
外れ値あり	外れ値なし	-395.67	-342.97	**-14.26%**	**-8.23%**	**-12.93%**	**-8.07%**
外れ値なし	外れ値あり	-410.08	-102.81	2.26%	-1.11%	-3.71%	-1.99%
外れ値あり	外れ値あり	-738.22	-393.27	**-10.59%**	**-8.55%**	**-13.95%**	**-10.66%**

した理由は，ソフトウェア開発における利益（売上 - コストで計算される）を考慮した場合，コスト（工数）の誤差が 5%以上存在すると，利益が赤字になるリスクが無視できない程度に高まると想定したためである．

5.1. 重回帰分析における外れ値の影響

目的変数に外れ値を加えた場合

外れ値の程度が 10%の場合（表 3），見積もり精度が全体的に悪化していた．外れ値の程度が 50%の場合（表 2）でも，フィットデータに外れ値あり，テストデータに外れ値なしで *MRE* 平均と *BRE* 平均が 5%以上悪化していた．フィットデータ，テストデータ両方に外れ値ありだと，*BRE* 平均が 10%，*BRE* 中央値が 5%以上悪化していた．よって，外れ値の影響は決して無視できないといえる．

当然の結果ではあるが，フィットデータ，テストデータ両方に外れ値ありの場合の見積もり精度が最も低くなっており，外れ値の程度に関わらず，*BRE* 平均と *BRE* 中央値が-5%を越えて悪化していた．

説明変数に外れ値を加えた場合

全体的に見積もり精度が悪化しているものの，工数に外れ値を加えた場合よりは精度の低下が低かった．外れ値の割合を 10%，外れ値の程度を 100%にした場合（表 4），*MRE*，*BRE* の各評価値で 5%を越える悪化はなかった．ただし，外れ値の割合を 20%にした場合（表 5）は各評価値の悪化が見られた．特に *BRE* 平均が，どのデータセットに外れ値が存在する場合においても 5%以上悪化していた．

このことから，FP の誤差が大きいプロジェクトが見積もり根拠データに多数含まれている場合，外れ値に注意する必要があるが，外れ値の割合が 10%，外れ値の程度が 100%程度であるならば，あまり外れ値の影響を考慮しなくてもよいといえる．

目的変数に外れ値を加えた場合では，フィットデータに外れ値があり，かつテストデータに外れ値があると，テストデータに外れ値がない時と比べて精度の低下が大きかった．説明変数に外れ値を加えた場合，フィットデータにその外れ値があれば，テストデータにおける外れ値の有無は見積もり精度にそれほど影響していなかった．

5.2. 類似性に基づく見積もりにおける外れ値の影響

目的変数に外れ値を加えた場合

外れ値の割合が 10%，外れ値の程度が 100%の場合（表 7），見積もり精度が大きく低下しており，フィットデータに外れ値なし，テストデータに外れ値ありの場合を除き，*MRE*，*BRE* の平均と中央値が 5%以上悪化していた．ただし，外れ値の割合が 10%，外れ値の程度が 50%の場合（表 6），フィットデータに外れ値あり，テストデータに外れ値なしの場合を除き，*MRE*，*BRE* の平均と中央値が 5%以上悪化していなかった．

説明変数に外れ値を加えた場合

全体的に評価指標の値が大きく悪化していなかった（表 8，表 9）．外れ値の割合が 20%の場合（表 9）においても，*MRE*，*BRE* の平均と中央値が 5%以上悪化することがなかった．よって，工数見積もりに Analogy 法を用いる場合，開発規模に外れ値が含まれていても，あまり影響を考慮しなくてもよいと考えられる．

表 8　Analogy 法における，外れ値と見積もり精度の関係（FP，10%，100%）
Table 8　Relationships between outliers and estimation accuracy on analogy based estimation (FP, 10%, 100%)

フィット データ	テスト データ	*AE* 平均	*AE* 中央値	*MRE* 平均	*MRE* 中央値	*BRE* 平均	*BRE* 中央値
外れ値あり	外れ値なし	7.98	10.59	0.99%	0.49%	0.86%	0.43%
外れ値なし	外れ値あり	-38.52	-9.64	-1.18%	-0.89%	-0.78%	0.40%
外れ値あり	外れ値あり	-32.39	32.21	-0.15%	-0.91%	0.05%	0.64%

表 9　Analogy 法における，外れ値と見積もり精度の関係（FP，20%，100%）
Table 9　Relationships between outliers and estimation accuracy on analogy based estimation (FP, 20%, 100%)

フィット データ	テスト データ	*AE* 平均	*AE* 中央値	*MRE* 平均	*MRE* 中央値	*BRE* 平均	*BRE* 中央値
外れ値あり	外れ値なし	-16.57	3.15	0.99%	-1.02%	0.41%	-0.88%
外れ値なし	外れ値あり	-75.97	-55.83	-4.10%	-1.71%	-3.20%	-2.18%
外れ値あり	外れ値あり	-69.30	-77.98	-1.99%	-2.24%	-2.03%	-3.13%

5.3. 重回帰分析と類似性に基づく見積もりとの比較

目的変数に外れ値を加えた場合において，重回帰分析と Analogy 法を比較すると，見積もり精度の悪化は，重回帰分析のほうが大きかった．特にフィットデータとテストデータに外れ値が含まれる場合の差が大きかった．例えば，外れ値の程度が 50%の場合，Analogy 法では *MRE*，*BRE* の平均と中央値が 5%以上悪化していなかったが，重回帰分析では 5%以上悪化していた．

Analogy 法のほうが，テストデータに外れ値が含まれていた場合の影響が小さい可能性がある．外れ値の程度が 100%でフィットデータに外れ値がない場合，*BRE* 平均値の悪化は-3.71%であり，重回帰分析の同じ場合よりも精度の悪化が小さかった．ただし，どの実験結果でも同様の傾向であるとまではいえないため，結論を下すためには，今後さらなる分析が必要である．

説明変数に外れ値を加えた場合においても，Analogy 法のほうが重回帰分析よりも見積もり精度の低下が小さかった．Analogy 法の場合，外れ値の割合が 20%の場合においても，*MRE*，*BRE* の平均と中央値が 5%以上悪化することがなかった．

これらの結果から，重回帰分析よりも Analogy 法のほうが外れ値の影響を受けにくいと考えられる．ただし，フィットデータとテストデータに外れ値が含まれていない基準モデル同士を比較すると，Analogy 法のほうが重回帰分析よりも見積もり精度が低かった．Analogy 法の基準モデルの見積もり精度が低かったために，外れ値によるさらなる精度低下が起きにくかった可能性もあることに留意する必要がある．

5.4. 考察

実験結果を，ソフトウェア開発の現場でどのように活用すべきかについて考察する．ソフトウェア開発工数の見積もりモデルを構築する際，ソフトウェア開発データセットに含まれうる外れ値の影響を避けるためには，以下の 2 つの方法がある．

- データセットに外れ値が含まれないように，データ計測及び収集を正確に行う．
- 外れ値と考えられるデータを除去する．

ただし，上記 2 つの方法を厳格に適用しなくとも，すなわちデータを正確に収集することにコストを掛け過ぎたり，外れ値を完全に除去したりすることをしない場合でも，外れ値の影響が極端に大きくならない可能性がある．実験に用いたデータセットにおいては，例えば開発規模の計測値に 50%の誤差が含まれているケースが 10%存在した場合でも，誤差（*BRE* 平均など）が 50%以上変化するなどといった，極端に見積もり精度が悪化するということはなかった．

本研究の分析結果は 1 つのデータセットを用いただけであり，今後さらに他のデータセットで同様の実験を行い，結果の信頼性を高める必要がある．また，外れ値の除去を行うことにより見積もり精度は改善されるため，外れ値を除去する必要性がないというわけではない．ただし，開発現場においてデータを収集，活用する際には，外れ値に対して厳格に対応する必要性は必ずしも高くない可能性がある．

なお，ソフトウェア工数見積もりの研究において，Desharnais データは広く用いられており[6][13]，

Desharnais データ 1 つだけを用いて実験を行っている研究も存在する[6]. 本研究の実験結果を過度に一般的することは危険であるが, 他のデータセットで同様の実験を行った場合でも, 外れ値の工数見積もり精度に対する影響が非常に大きいなど, 結果が極端に異なる可能性は低いと考えられる. ただし, 外れ値の程度と割合が見積もり精度にどの程度影響を与えるか(*BRE* 平均値が何%悪化するか)は, データセットにより異なると考えられるため, 今後さらに別のデータセットにおいて同様の実験を行い, 詳細に両者の関係を分析する必要がある.

6. 関連研究

Seo ら[11]は, ソフトウェア開発工数見積もり時に LTS 法を用いた除去法と k-means 法を用いた除去法を適用することを提案し, 重回帰分析, ニューラルネットワーク, ベイジアンネットワークそれぞれによる見積もり時に外れ値除去法を適用し, 外れ値除去の効果を確かめている. ただし, 本研究のように実験的に外れ値を追加することは行っておらず, 外れ値の程度, 外れ値の割合などと見積もり精度との関係は明らかにしていない.

Shepperd ら[12]は, 重回帰分析や類似性に基づく見積もりなどの複数の見積もり手法の性能を評価するために, 外れ値や多重共線性を持つデータセットを実験的に生成し, 分析に用いている. Shepperd らは説明変数に外れ値を加えているが, 外れ値の存在するデータセットを変えるなどはしておらず, 本研究のように複数の観点から外れ値の影響を評価することは行っていない.

7. おわりに

本研究では, 工数見積もりにおいて外れ値が与える影響について明らかにするために, データセットに外れ値を実験的に追加して分析を行った. 分析では, 外れ値の割合, 外れ値の程度, 外れ値の存在する変数, 外れ値の存在するデータセットの 4 つを変化させて見積もり精度を評価した. 工数見積もり手法として, 重回帰分析と類似性に基づく見積もり方法を採用した. その結果, 開発規模に外れ値が含まれていても, 外れ値の割合が 10%で, 外れ値の程度が 100%の場合, あまり影響が大きくないことなどがわかった. また, 類似性に基づく見積もり方法の場合, テストデータに外れ値が含まれていても, 比較的影響が小さい可能性があることがわかった.

今後の課題は, さらにデータセットを増やすとともに, 外れ値の割合や外れ値の程度をさまざまに変化させて実験することにより, 分析結果の信頼性を高めることである. 特に, 一般に広く知られている IPA/SEC が収集したデータセット[5]や, 工数見積もりモデルの評価で広く用いられる ISBSG データセット[4]などを用いて同様の評価を行い, 結果の信頼性を高める必要がある.

謝辞 本研究の一部は, 文部科学省科学研究補助費(基盤 C:課題番号 25330090)による助成を受けた.

参考文献

[1] Boehm, B.: *Software Engineering Economics*, Prentice Hall (1981).
[2] Conte, S., Dunsmore, H., and Shen, V.: *Software Engineering, Metrics and Models*, Benjamin Cummings (1986).
[3] Desharnais, J.: *Analyse Statistique de la Productivitie des Projets Informatique a Partie de la Technique des Point des Function*, Master Thesis, University of Montreal (1989).
[4] International Software Benchmarking Standards Group (ISBSG): *ISBSG Estimating: Benchmarking and research suite*, ISBSG, 2004.
[5] 情報処理推進機構 ソフトウェア・エンジニアリング・センター:ソフトウェア開発データ白書 2010-2011, 情報処理推進機構 (2010).
[6] Keung, J., Kitchenham, B., and Jeffery, R: Analogy-X: Providing Statistical Inference to Analogy-Based Software Cost Estimation, *IEEE Trans. on Software Eng.*, Vol.34, No.4, pp.471-484 (2008).
[7] Mendes, E., Martino, S., Ferrucci, F., and Gravino, C.: Cross-company vs. single-company web effort models using the Tukutuku database: An extended study, *Journal of Systems and Software*, Vol.81, No.5, pp.673–690 (2008).
[8] Miyazaki, Y., Terakado, M., Ozaki, K., and Nozaki, H.: Robust Regression for Developing Software Estimation Models, *Journal of Systems and Software*, Vol.27, No.1, pp.3-16 (1994).
[9] Mølokken-Østvold, K., and Jørgensen, M: A Comparison of Software Project Overruns-Flexible versus Sequential Development Models, *IEEE Trans. on Software Eng.*, Vol.31, No.9, pp.754-766 (2005).
[10]門田暁人, 小林健一:線形重回帰モデルを用いたソフトウェア開発工数予測における対数変換の効

果，コンピュータソフトウェア，Vol. 27，No. 4，pp.234-239 (2010).
[11] Seo, Y., Yoon, K., and Bae, D.: An Empirical Analysis of Software Effort Estimation with Outlier Elimination, *Proc. the international workshop on Predictor models in software engineering (PROMISE)*, pp.25-32, Leipzig, Germany (2008).
[12] Shepperd, M., and Kadoda, G.: Comparing Software Prediction Techniques Using Simulation, *IEEE Trans. on Software Eng.*, Vol.27, No.11, pp.1014-1022 (2001).
[13] Shepperd, M. and Schofield, C.: Estimating software project effort using analogies, *IEEE Trans. on Software Eng.*, Vol.23, No.12, pp.736-743 (1997).
[14] Srinivasan, K., and Fisher, D.: Machine Learning Approaches to Estimating Software Development Effort, *IEEE Trans. on Software Eng.*, Vol.21, No.2, pp.126-137 (1995).
[15] Strike, K., El Eman, K., and Madhavji, N.: Software Cost Estimation with Incomplete Data, IEEE Trans. on Software Eng., Vol.27, No.10, pp.890-908 (2001).
[16] 田中豊，垂水共之（編）：Windows 版 統計解析ハンドブック 多変量解析，共立出版 (1995).
[17] 角田雅照，大杉直樹，門田暁人，松本健一，佐藤慎一：協調フィルタリングを用いたソフトウェア開発工数予測方法, 情報処理学会論文誌, Vol.46, No.5, pp.1155-1164 (2005).

要求のヌケ・モレを防ぐためのゴール分解方法の提案と実験 －ソフトウェア・シンポジウム 2014　WG3 の事例－

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

システムの要求獲得でゴールモデルを利用する際に，できる限り要求のヌケやモレがなく，誰でも実施可能なゴール分解を可能とする手法・質問方法を提案するのが本研究の目的である．

目的を達成するために，本研究では，ゴール分解手法として *Lamsweerde* の「ゴール・カテゴリ」を参考にゴール分解方法を提案し，そのゴール分解を誰でも行えるように，ゴール分解方法に対応する質問方法を提案する．

そして，この提案の有効性を検証するために,「交通安全」の事例を用いて，ソフトウェア・シンポジウム *2014* ワーキンググループ *WG3*「要求変更の要因分析」で実験した．その結果，提案する質問を行うと，詳細化したゴールが多くあらわれることが明らかとなった．

1. はじめに

1.1. 研究の背景

ゴール指向分析では，はじめにシステムの目標であるトップゴールを抽出し，そのトップゴールを AND 分解，OR 分解を用いて詳細化を行っていく．しかし，詳細化を行う際に，さまざまな観点からゴールを分解できてしまうため，出来上がったゴールグラフが，必ずしも同じゴールグラフになるとは限らない．その結果，ステークホルダの要望を十分に反映しないゴールグラフが出来たり，考慮すべき状態が挙げられていないゴールグラフが出来てしまうこともある．このようなゴールグラフを元に要求仕様を作成し，後工程を実行すると，要望や考慮すべき事項のヌケ・モレが発生する可能性がある．ここでモレ・ヌケとは，要求者が開発者に対して要求を提示し，開発者が設計・実装するとき，設計・実装上必要な事項が開発者に伝わっていなかった状態と本研究では定義する．モレ・ヌケがあると，要求者と開発者の間で誤解を生じ，その結果，後工程でそのヌケ・モレに対応するための要求変更が起こり，開発コストが高くなってしまう可能性がある．したがって，要求のヌケ，モレがなく，分析者が同程度の品質のゴールモデルを作成できるようにするためのゴール分解手法が必要となる．

1.2. 研究の目的

本研究は，システムの要求獲得時に要求のヌケ，モレを削減するためのゴール分解手法を提案する．また，その手法を技術者に質問するための方法を開発する．さらに，提案する手法および質問方法の有効性を評価するためにゴール分解実験を行う．

2. 関連研究

2.1. ゴール指向要求分析

ゴール指向要求分析は，ゴールに基づいて要求を分析する手法である．主なゴール指向要求分析手法としては i*，KAOS 等がある．

i*[1] は要求分析を行う際に，アクター，ゴール，タスク，ソフトゴール，資源という要素を用い，アクター間の依存関係を他の 4 つの要素を用いて分析する．これによって，各アクターが達成すべきゴールを抽出できるようになる．

KAOS[2] は，上位ゴールを AND 分解と OR 分解のいずれかを用いて，サブゴールへと繰り返し詳細化してい

く．AND 分解は，詳細化されたサブゴールのすべてが達成されないとゴールが達成されない分解方法であり，OR 分解は，ゴールのうち 1 つでも達成されればゴールが達成される分解方法である．本研究の議論の出発点はゴールモデルのトップゴールである．トップゴールをどのように分解すればゴールモデルの品質を確保できるかを論ずるために，i*におけるアクター間の依存関係から議論を始めるのではなく，KAOS のゴールの詳細化における課題を定義し，その解決策を考案することから始める．

2.2. ゴール・カテゴリ

Lamsweerde は，[2] の中で，ゴールをカテゴリ化している．ゴールは大きく,「機能ゴール」と「非機能ゴール」に分かれ,「機能ゴール」はさらに,「満足」,「刺激－反応」,「情報」の 3 つに別れる．それぞれの意味を，以下のように説明している．

- 満足 : エージェントの要求を満足させることに関する機能ゴール
- 刺激－反応 : 特定のイベントに対する適切な反応に関する機能ゴール
- 情報 : 重要なシステムの状態に関してエージェントに知らせることに関する機能ゴール

本研究では，ゴール分解の方法を論じるにあたり，ゴール・カテゴリのうち，特に機能ゴールの分解方法に着目する．なぜならば，機能ゴールの分解に関しては，未だ，属人的な手法となっていると著者らは考えるからである．

Lamsweerde も [2] の中でゴール獲得方法やゴール洗練方法を述べている．特に,「8.8.1 Eliciting preliminary goals」では，経験則として，ゴール抽出方法を紹介している．ゴールカテゴリに関しては,「 (H3)Instantiate goal categories 」の中で，ゴールカテゴリの葉が，システム固有の要求になるため，具現化する必要があると述べている．さらに，ゴールカテゴリ以外のゴール抽出・洗練方法も紹介されており，下位ゴールに洗練するための HOW/WHY の質問，責任による分割，ソフトゴールの認識，エージェントの願望の識別，障害や脅威の分析などが紹介されている．

しかし，本研究で焦点を当てている設計・実装のためにモレ・ヌケを削減するためには何が必要で，どのようにゴール分解していくのがよいかについては，具体的な手法が紹介されていない．

3. 提案する手法

ここでは，本研究で提案する手法の概要と，手法を質問するための，指示の内容と順番を述べる．

3.1. ゴール分解方法へのアプローチ

モレ・ヌケは「設計・実装上必要な事項が，開発者に伝わっていなかった状態」と定義したので，モレ・ヌケをなくすには，設計・実装に必要な情報が開発者に伝わればよいこととなる．ここで，機能要件における設計・実装に必要な情報とは，プログラミングを記述するうえで必要な情報，すなわち「どのような関数・メソッド」を「どのような引数」で,「いつどの順番で呼び出すか」であると考えられる．これらの情報をゴールカテゴリを活用して引き出す方法を提案すれば，本研究の目的は達成される．

3.2. ゴール分解方法の提案

本提案手法は，Lamsweerde のゴールカテゴリの研究を参考にしたもので，トップゴールが与えられたとき，以下の手順でゴール分解を行うといったものである．

- ステップ 1 : そのシステムに関するステークホルダ X を挙げる
- ステップ 2 : ステークホルダ X は，最終的にどのような状態になっていれば満足するのかを挙げ,「ステークホルダー X にとって，対象 S が状態 Q になっている」というゴールを作成する
- ステップ 3 :「最終的に満足な状態 Q」になるために,「事前に達していなければならない状態 P が起きた」というサブゴールと「イベントがおき，変換プロセス A が起動され終了している」と AND 分解し，状態 P を探し，変換プロセス A を割り当てる．
- ステップ 4 : 最終的に満足する状態 Q になるために必要な情報を挙げる

Lamsweerde のゴールカテゴリでは，満足ゴールは「エージェントの要求を満足させることに関する機能ゴー

ル」である．ここではエージェントは満足させる対象，すなわちステークホルダーと考えられる．そこでシステムのステークホルダーに対して着目しゴール分解を行う．

ステップ 2 では，Lamsweerde の満足ゴールの「満足させること」の部分を具体化している．つまり，ステークホルダーが求める最終的状態，すなわち最終的な出力に着目して分解を行う．このとき，何が（対象 S) どのような値（状態 Q) になっていれば良いかを明確にする．また状態 Q で満たさなければならない制約についても挙げる．

ステップ 3 では，Lamsweerde の刺激 - 反応ゴールを具体化する．刺激 - 反応ゴールは「特定のイベントに対する適切な反応に関する機能ゴール」であった．ここで「適切な反応」とは「イベントが起きたときの状態 P から，満足ゴールの状態 Q に至る反応」と考える．この反応がプログラムを実装したときの「関数・メソッド」に対応する．この刺激ー反応，すなわち刺激を与える状態 P から反応が起きた状態 Q に至るプロセスを「変換プロセス」と定義する．この状態 P が見つかり，変換プロセスが記述できるかが詳細化できるかどうかのポイントであり，これが開発者に伝わらなければならない．

また，ある変換プロセス A の状態 Q が別の変換プロセス B の状態 P になっていて，変換プロセス B の状態 Q が，さらに別の変換プロセス C の状態 P になっていた場合，A → B → C のように遷移が起こり得る．このように状態 P，Q の関係により「いつどの順番で呼び出すか」が判明する．

ステップ 4 では Lamsweerde の情報ゴールを具体化する．状態ゴールは,「重要なシステムの状態」を問題とする．ステップ 3 のイベントが起きるときの状態 P と終了時の状態 Q の状態がわからないと変換プロセスが生ぜず，ゴールが達成できないため，状態 P と状態 Q は重要なシステムの状態と言える．ここで状態とは複数オブジェクトの属性値と考える．すると，ここで状態 Q になるために必要な情報は，変換プロセス起動時の状態 P を構成するオブジェクトの属性値と，終了時に満たすべき状態 Q を構成するオブジェクトの属性値となる．状態 P を構成するオブジェクトの属性値のうち，変換プロセスが直接参照できない値は，変換プロセスの「引数」等で渡される．これら状態を構成するオブジェクトの属性値が詳細化でき，開発者に伝わらないと，設計・実装上,「関数・メソッド」をどのような引数で呼び出せばよいかわからなくなる．

このようにオブジェクトの属性値を明確にした後，それらの属性値が，オブジェクトのとり得る値として可能かどうか，また満足ゴールを満たしているかどうかを検証しなければならない．具体的には，オブジェクトの属性値を変えて，制約に違反していないか，満足ゴールを満たしているかの評価が必要である．

3.3. 手法を利用するための質問方法

前で述べたように，本手法は 4 ステップのゴール抽出から構成されている．そこで，これらのプロセスを遂行しやすくするための質問方法を以下に述べる．

提案する質問方法では，ゴール抽出のプロセスを遂行しやすくするために，以下の質問を網羅する必要がある．

1. ステップ 1 の質問 :「このシステムを利用する人と，関係者を挙げてください」
2. ステップ 2 の質問 :「このシステムを利用する人は，どうなっていればよいと思うか，列挙してください」
3. ステップ 3・4 の質問 :「最終的に満足する状態 S になるためには，その前提として，何が，どのような状態になっていなければなりませんか」

実際の質問を行う場合,「このシステム」の部分に，具体的なシステム名が入る．

ステップ 3・4 の質問は 2 ステップで 1 つの質問になっている．これは最終的な状態 Q はわかっても，その直前の状態 P はわかりにくいので，状態 Q の前の状態をすべて挙げてもらい，要求をまとめる際に，状態間の遷移と必要な情報を分析し，直前の状態 P を見出そうとしているからである．

この質問法の有効性を評価するために，我々は，ソフトウェア・シンポジウム 2014 のワーキンググループ WG3「要求変更の要因分析」において，この質問を行う場合と行わない場合について，具体的にゴールモデルがどのように変化するかを実験した．その実験と成果について，以降で議論する．

4. 実験

4.1. 実験計画

ゴールグラフ作成に対する本提案の有効性を確認するために，本実験を行う．本実験は，2 フェーズに分かれ

ている．第一フェースでは，質問なしにゴールグラフを作成する．その後行われる第二フェーズでは，はじめに本提案の質問を行い，その後ゴールグラフを作成する．その後，第一フェーズと第二フェーズ間で違いがあるかどうかを評価する．

実験は，交通安全に関するゴールに関して行ったものである．我々は，本提案の手法は，社会システムの改善にも情報システムへの適用にも，どちらも可能であると考えている．そこで今回は社会システムの例として交通安全を取り上げる．交通安全の目的について扱った法律である道路交通法 [3] や関連法規と比較することにより評価する．

具体的には，道路交通法等に出てくる用語と実験で出現した用語を詳細化のレベルを考慮して比較することによって効果を測定する．

道路交通法において，どのような利用者 の詳細化がなされているかを図 1 に示す．この図と，本実験で挙げられたゴールと対応付けることによって，被験者が，どの程度ゴールを詳細に分解出来ていたかを測定する．

なお，本実験は，ソフトウェア・シンポジウム 2014 のワーキンググループ WG3「要求変更の要因分析」として行なった．

4.2. 実験：第一フェーズ (ワーキンググループ 1 日目)

4.2.1 実験内容

大学生 5 人と，システム開発経験者，本稿の筆者の 1 人が参加して行った．ゴール指向分析と AND 分解，OR 分解を説明した後，トップゴールである「利用者が交差点で安全快適に通行できている」を挙げ，それをゴール分解するように指示した．ただし，トップゴールから直接結びつかないゴールを挙げてもかまわないものとし，ゴールを付箋紙に記述し，模造紙に貼った．この作業を 90 分間行った．

4.2.2 結果

第一フェーズで作成したゴールグラフは図 2 のとおりである トップゴールからつながなくてよいとしたため，トップゴールと繋がらないゴールが多数，出来ている．実験中，大学生の 1 人が「交通システム」の授業で「歩車分離」等の概念について学んだとの発言があったが，それらはゴールとして挙がっていない．

また，図 1 の章レベルで挙げられている「歩行者」，「車両」の 2 つと，条・項レベルで挙げられている「目の不自由な人」,「自動車」の 2 つはゴールやエージェント中に現れているが，そのほかの利用者については出てきていない.「人」,「車」という抽象的な概念で表現しているゴールもある．

4.3. 実験：第二フェーズ (ワーキンググループ 2 日目)

4.3.1 実験内容

第一フェーズの実験から 2 日後に行った．大学生 2 人と本稿の筆者 2 人が参加した．大学生は 2 人とも第一フェーズの参加者であった．本稿の筆者 1 人（第一フェーズに参加していない．以下実験者と記す）が質問を行い，大学生から挙げられたゴールを付箋紙に記載し，模造紙に貼っていった．そして付箋紙に記載されたゴールをサブゴールに分類した．この作業を 90 分間行った．

以下，どのように質問を行ったかを説明する．

1. 実験者が，トップゴールである「利用者が交差点で安全快適に通行できている」を挙げた
2. 実験者がさらに，トップゴールを,「利用者が交差点で安全に通行できている」,「利用者が交差点で快適に通行できている」に分解した
3. 実験者が，第一の質問「交差点を利用する人と，関係者を挙げてください」を行った．この質問を受けて，参加者の大学生 2 人は,「利用者が交差点で安全に通行できている」,「利用者が交差点で快適に通行できている」のゴールの下に，利用者を挙げ，ゴールを詳細化していった．
4. 実験者が「安全快適に通行できるために関与している」というゴールを挙げ，参加者は，関係者であり利用者でないものを挙げて，そのゴールを詳細化していった．
5. これら利用者，関係者のサブゴールが挙がった後，実験者は，第二の質問「交差点を利用する人は，どうなっていればよいと思うか，挙げてください」との質問を行った．ただし，この質問は，被験者の理解性を向上させるため「交差点に信号機があるとして，歩行者は，待っているとき，どうなっていてほしいですか，挙げてください」という質問に変えた．

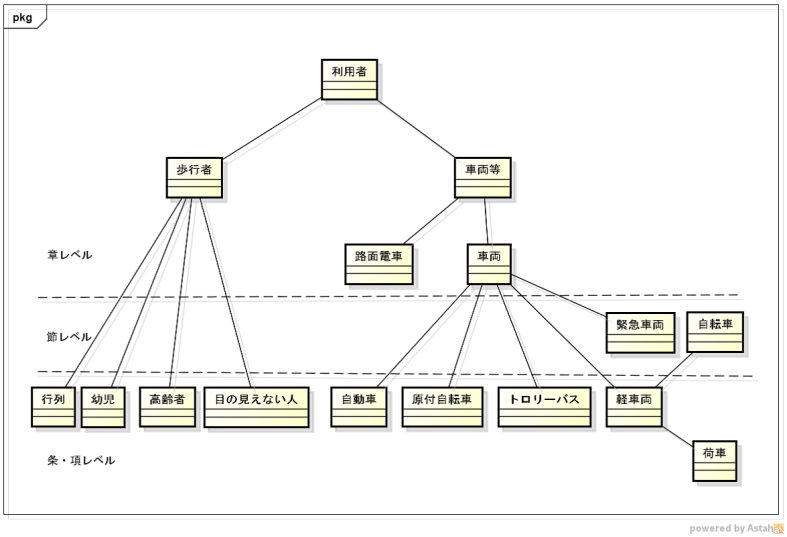


図 1. 道路交通法の交通方法に関する利用者

6. その後,「歩行者が渡っているとき」,「自動車が待っているとき」と，利用者と状況を変えて質問を行った．質問を行うものが，ゴールを付箋紙に記入し，大学生の 2 人がゴールを挙げていった．

予定では，信号機に関して，被験者が一通りサブゴールを挙げたら，歩道橋についてもゴール分解の作業を行ってもらうことにしていたが，時間がなかったため，行えなかった．また「安全快適に通行できるために関与している」ゴールについても，ステークホルダを挙げるところ，すなわち第一の質問までは行ったが，それ以上は時間がなく行えなかった．第三の質問も行えなかった．

4.3.2 結果

上述の作業で作成したゴールグラフは図 3（「安全に通行できている」のゴールを基に作成），図 4（「快適に通行できている」のゴールを基に作成），図 5（「安全快適に通行できるために関与している」のゴールを基に作成）である．

なお，利用者には「右折する自動車」,「左折する自動車」があるのにもかかわらず,「どうなっていてほしいか」に関しては右折のみしかない．これは,「どうなっていてほしいか」をたずねたとき，大学生の 1 人が,「左折はどうでもいいけど，右折がしたい」という意見が出たため，左折のゴールをあえて作らず，右折のみにしたためである．

図 1 の章レベルは「歩行者」1 つであるが，条・項レベルは,「子供」（幼児）,「目の不自由な人」,「老人」（高齢者）,「自動車」,「バイク」（原動機付自転車）等，フェーズ 1 より多くのゴールが詳細に挙がっている．さらにフェーズ 1 では挙がっていなかった節レベルの「緊急車両」（緊急自動車等）,「自転車」も挙げられている．このほか,「三輪車」,「せっかちな歩行者」,「横断する気のない歩行者」,「暴走族」,「右折する車」,「左折する車」,「直進する車」,「右折する自転車」,「左折する自転車」,「直進する自転車」といった，道路交通法にないものまでも詳細化されている．

4.4. 評価

今回は，第一の質問しか終わっていないため（第二の質問は作業途中），第一の質問であるステークホルダーの出現数について比較する．

第一フェーズと第二フェーズを比較した表が，表 1 である．第一フェースは，章より詳細化された「節レベル」「条・項レベル」で出てくる用語が，第二フェーズよりも少ない．このことから，第二フェーズのほうが，詳細化したゴールが多く現れているといえる．一方，章に関しては，第一フェーズのほうが 1 つ多い．これは,「車両」という言葉が第二フェースで出てこなかった為である．た

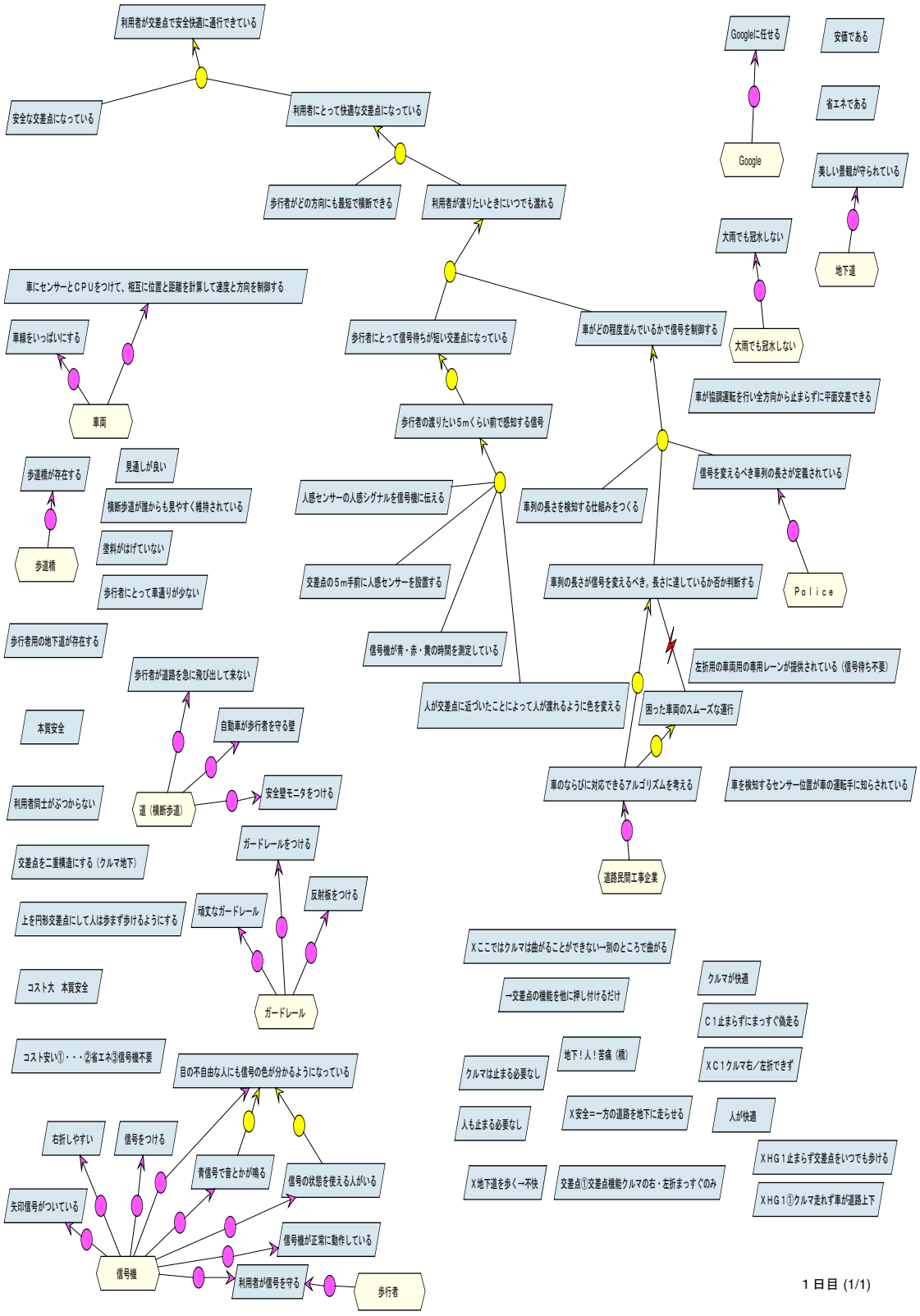


図 2. 1 日目のゴールグラフ

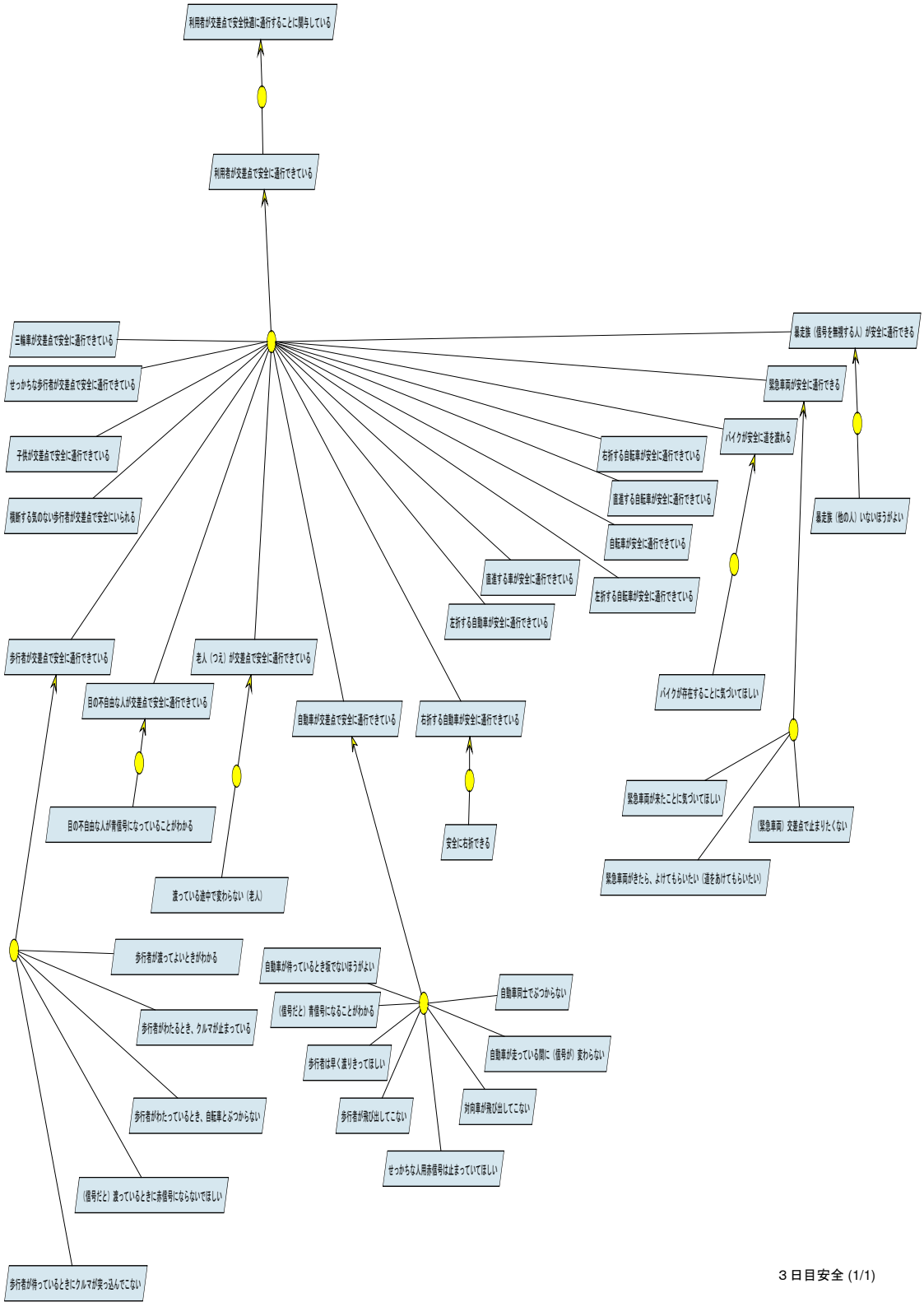


図 3. 2 日目のゴールグラフ（「安全に通行できている」）

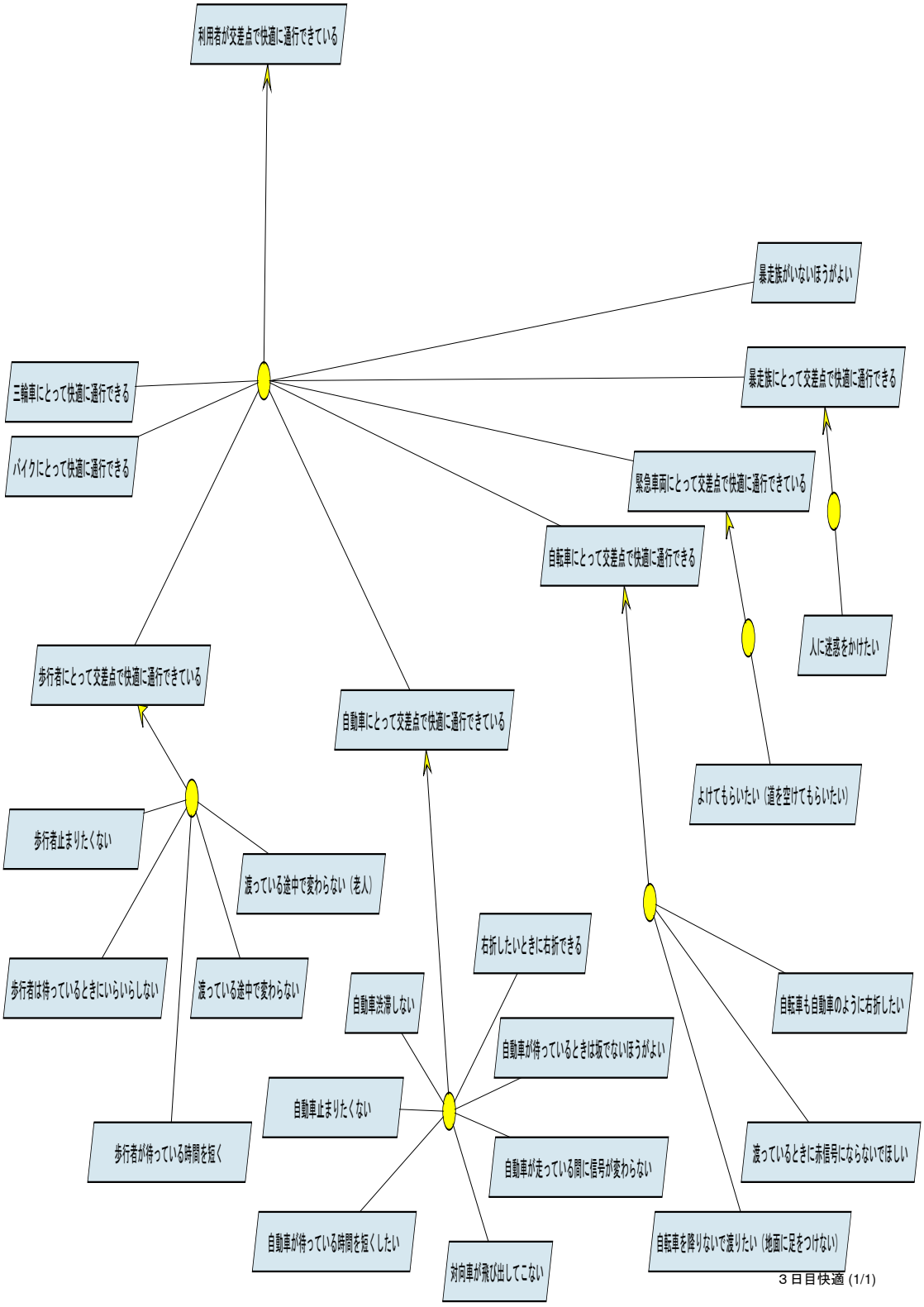


図 4. 2 日目のゴールグラフ（「快適に通行できている」）

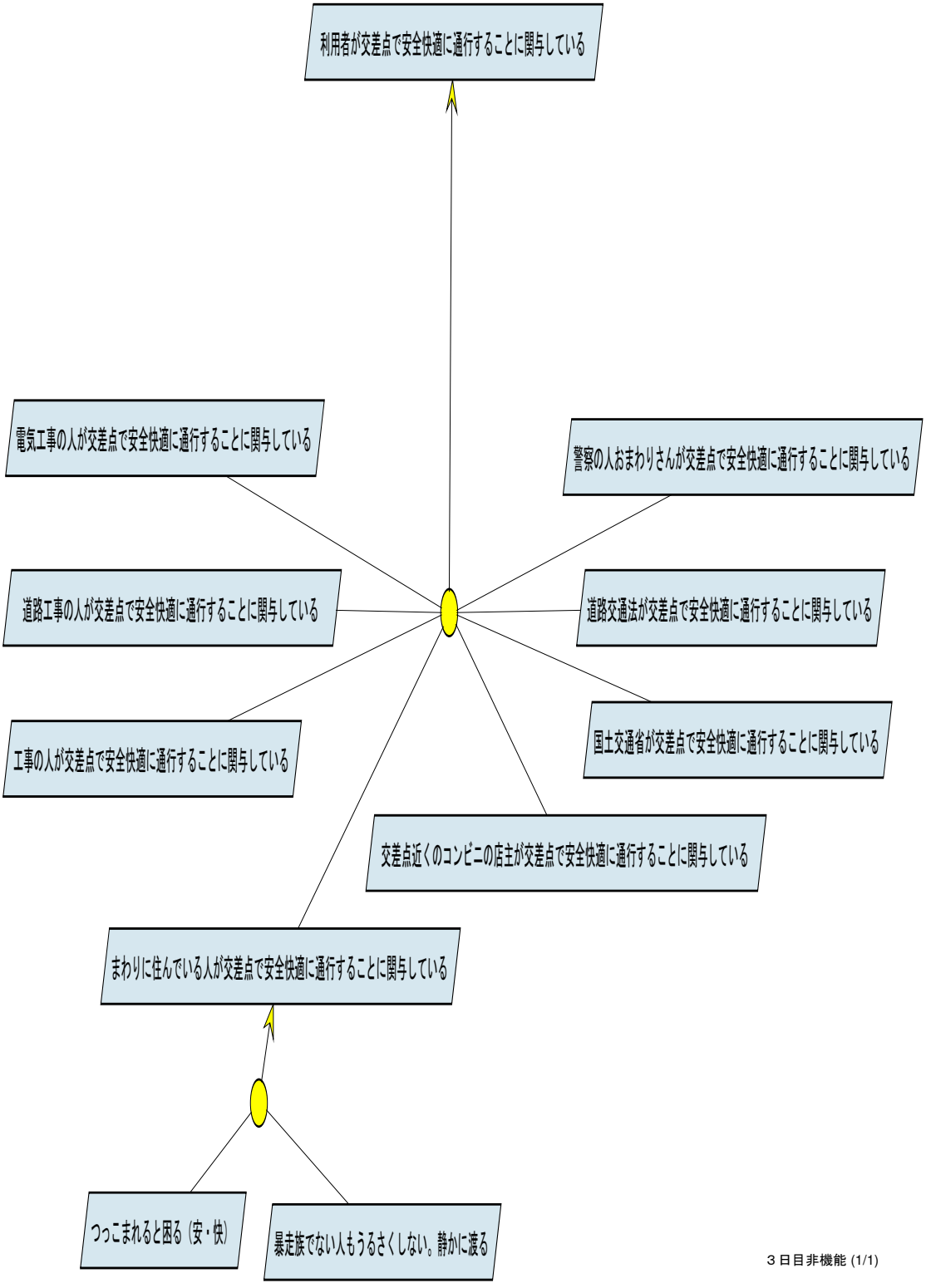


図 5. 2 日目のゴールグラフ（「安全快適に通行できるために関与している」）

表 1. 第一・第二フェース結果比較

レベル	道路交通法	第一フェーズ	第二フェーズ
章レベル	3	2	1
節レベル	2	0	2
条・項レベル	9	2	5

だし第二フェーズでも車両を詳細化した「自動車」という言葉は出てきている．したがって，車両が満たすべきゴールは，自動車が満たすべきゴールとして第二フェーズで」も挙げられている．

また，道路交通法にないものについては，第一フェーズが「人」「車」等，章レベルの概念と同等か，それよりも大きい概念が出ているのに対し，第二フェーズでは，「三輪車」,「せっかちな歩行者」など,「条・項レベル」より詳細化したゴールが多くあらわれている．

5. 考察

5.1. 本提案の有効性

質問を行わなかった第一フェーズよりも，質問を行った第二フェーズのほうが，詳細化した用語を含むゴールが多くあらわれているということは，質問によって，詳細化が促され，幅広い利用者の要望が抽出できたことを意味する．特に第二フェーズにおける「左折はどうでもいいけど，右折がしたい」という発言は，第一フェーズより具体的に状況を考察した上で現れた要望といえる．この点で，本提案による第一の質問（ステークホルダーの抽出）「このシステムを利用する人と，関係者を挙げてください」によって，多くのゴールを出現させ，ヌケ・モレの削減という本研究の目的に対する有効性が示されたといえる．

5.2. 今後の課題

今回は，第二の質問「このシステムを利用する人は，どうなっていればよいと思うか，列挙してください」は途中までしか行えず，第三番目の質問「最終的に満足する状態 S になるためには，その前提として，何が，どのような状態になっていなければなりませんか」は全く行えなかった．この第二，第三の質問を行った場合どうなるかについて，今後実験をする必要がある．また，今回の実験では，大学生だったため，経験の浅い技術者を対象として実験したといえる.「誰でも」といった場合，経験豊富な技術者だった場合，質問に効果があるかどうかを研究する必要がある．

6. まとめ

システムの要求獲得時に，できる限り要求のヌケ，モレがなく，誰でも実施可能なゴール分解手法が可能となる質問方法を提案するのが本研究の目的であった．

この目的を達成するために，質問を用いたゴール指向分析手法を提案した．提案した手法の有効性を評価するために，質問の有無による実験を行った．その結果，質問によって分析を詳細に行うことができることが明らかとなった．

参考文献

[1] “i* homepage”, http://www.cs.toronto.edu/km/istar/(2015/3/14 アクセス)

[2] Axel van Lamsweerde “Requirements Engineering: From System Goals to UML Models to Software Specifications”, Wiley,2009.

[3] 道路交通法, http://law.e-gov.go.jp/htmldata/S35/S35HO105.html(2015/3/14 アクセス)

チームビルディング活動の効果測定 ～ 事例 10 年間の活動成果 ～

増田 礼子
フェリカネットワークス
Ayako.Masuda @ FeliCaNetworks.co.jp

森本 千佳子
東京工業大学
morimoto @ cs.titech.ac.jp

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

津田 和彦
筑波大学大学院
tsuda @ gssm.tsuka.tsukuba.ac.jp

要旨

ソフトウェア開発をはじめとする IT 分野において，チームの構築及び活性化は，プロジェクトの成功 / 失敗の大きな要因である．このような背景の中，2014 年のソフトウェア・シンポジウムでは，9 年間に渡るチーム活性化活動について事例報告としてまとめた．今回は，このチーム活性化活動の効果を明らかにするため，他のチームとの比較を行ったので報告する．

チーム活性化活動の効果を客観的に示す方法として，チーム活性化活動を実施していないチームとの比較を行うことが考えられる．ここでは，それぞれのチームのメンバに対して質問紙調査を行い，その調査データを基に統計分析によって比較を行う方法を用いた．効果測定の事例として，2014 年の事例報告で示したチーム活性化活動を行ったチームとの比較結果を示す．

質問紙調査は，事例のチームと，他の約 70 チームに対し，「IT の現場力調査」として使われている質問紙を用いて行った．この調査で集めたデータを分析した結果，事例のチームと他のチームとの間に明確な違いがあることが判った．さらに，違いを生じさせた主成分の確認を行った結果，両チームの違いは 4 つの因子で説明できることが判った．主成分の考察により，この 4 つの因子は，「自分たちのチームに対する自信や信頼」，「役割に対する納得感や態度」，「仕事に対する技術的な誇り」，「仕事に対する自由度」であると推測する．そして，これらの因子が，事例のチームがチーム活性化活動によって培ったチーム力の主要な構成要素だと考える．

1. はじめに

近年，ソフトウェア開発をはじめとする IT 分野では，プロジェクト型組織が多くなっている．プロジェクト型組織とは，共通の目標の達成を目指して集められたメンバによって構成された有期限の組織である．組織行動学では，共通の目的を達成すべく互いに相互作用関係をもって行動する複数の人間全体を集団という．その集団の中でも，仲間意識が強く，相互信頼感が高く，少数の構成員からなる集団をチームと呼ぶことが多くなってきている[1]．プロジェクトの規模により，プロジェクトの中に複数のサブチームがある場合もあるが，ここでは，プロジェクト全体をひとつのチームとして考えることとする．内閣府が作成した平成 19 年度版国民生活白書[2]によると，仕事に対する意欲を向上させるために重要なこととして「良好な人間関係」と回答した者は，54.2% と半数以上を占めている．これは，賃金や労働時間といった主要な労働条件に関する項目よりも割合が高く，職場における人間関係は，仕事に対する個人の意欲に影響を及ぼすと考えられる[2]．チームとして業務を進めていくためには，まず，メンバ同士の良好な人間関係の醸成が重要であるといえる．チーム活性化活動の効果測定には，良好な人間関係を築くことができているか，仲間意識を持つことができているか，といった観点が含まれると想定する．

ソフトウェア開発の工数や開発期間の見積もり手法のひとつである COCOMO[3]では，ソフトウェアの生産性に影響を与える 5 つのスケール・ファクタのひとつに「チーム強度」が挙げられている．これは，ユーザ，開発者，発注者などといった利害関係者間の連携，意思の疎通，目標やビジョンの共有についての度合いを示す指標である．この研究により，ソフトウェアの生産性にチームの状態，すなわち，ここで対象とするチーム活性化が影響を与えることが明らかになっている．

事例として扱った，モバイル FeliCa IC チップのファームウェア開発チームでは，プロジェクトの円滑な推進とチームのコミュニケーション活性化を目指し，10 年間に渡りチームビルディング活動を推進してきた[4][5]．本チームでは，まず，メンバ同士がお互いを知り合うといった関係性作りを目的としたチームのブートから始めた．そして，業務を推進していくために必要となるコミュニケーションとその活性化方法について，チームとして議論を重ねる機会を継続的に持ち続けてきた．具体的には，業務において必要なコミュニケーションとはどのようなものか，情報を正しく伝えるためにはどのようにしたら良いかなどに

ついて，プロジェクトやチームの状況に合わせて議論を重ねた．ここでは，チームビルディング活動を，プロジェクトの初期段階のチームのブートを目的とした活動だけではなく，チームを取り巻く状況に合わせた自律的なチーム力の維持を目指す継続的な活動ととらえる．チーム力とは，前述の COCOMO の研究[3]で定義されたチーム強度の構成要素であるメンバ同士の連携力，意思の疎通力，目標やビジョンの共有度合いとほぼ同義といえる．つまり，メンバそれぞれが持つさまざまな知識，経験，スキルなどにくわえて，それぞれの強みや弱みをメンバ同士で補完し合いながら，チーム全体で目標達成に向かう連携力と推進力である．

しかし，この活動により培われたチーム力がどのようなものなのか，その成果を定量的に評価することができていない．そこで，10 年に渡り継続してきたチームビルディング活動が及ぼしたチームへの影響を客観的に評価することが不可欠と考えた．

チーム力（Team Performance）に関する先行研究として，スポーツの世界においては，リーグ成績を基に Input-Process-Output model (I-P-O モデル)を用いた評価が報告されている[6][7][8]．しかし，ソフトウェア開発をはじめとする IT 分野においては，リーグ成績のようにチームの優劣を客観的に表す尺度の設定が難しい．リーグ成績に相当するものとして，プロジェクトの成功 / 失敗が考えられる．しかし，プロジェクトの成功 / 失敗にはチーム力以外の要因も強く関わっていること，プロジェクトの成功 / 失敗の定義が見方によってさまざまであることが尺度の設定が難しい原因である．また，プロジェクトの内容や目標によって求められるチーム力が異なり，多様なチーム力を評価する尺度もまだ開発されていない．

本研究では，IT 分野におけるチーム力研究の入り口として，チーム力の詳細な構成要素の分析ではなく，そもそも「チーム間で違いがあるのか」，「違いがあるとすればどのような違いなのか」をテーマに比較調査を行い，判別分析を行った．

チーム力の比較調査は，IT 分野におけるチーム力に関して調査・分析した松尾谷[9]の調査データと，本研究で事例としたチームのデータを基に行った．松尾谷[9]は，現場力を役割意識，仲間意識，規範意識，顧客との関係性の 4 つの下位尺度からなる構成モデルで定義し，各下位尺度単位で因子を用いた分析を行っている．本研究では，比較対象である他チームとの違いを調べるため，松尾谷が定義した構成モデルによる因子を用いた分析ではなく，元データである質問紙の全回答を使った判別分析から始めた．その結果を基に主成分分析を行い，差異が発生している因子を明らかにした．分析の結果抽出された 4 因子の中で，影響の大きな 2 因子は，先行研究[9]で抽出された 8 因子中の 3 因子と同じものであり，構成モデルにおいて示された 2 つの構成要素に近いものであった．分析のアプローチを変えても同じ要素が抽出されたということは，これらの要素がチーム力に与える影響が大きいことを示している．また，分析のアプローチを変えたことで，先行研究[9]では明らかにされていない因子が明らかになった．本分析の詳細を 3 章に示す．

2. チーム力の測定

スポーツ科学においては，チーム力を計測し客観的な評価を行う研究がある．目的変数にチームの勝敗などチームの優劣を表す尺度を用い，チームの構成やメンバの特性を説明変数として分析を行う．大学や高校のリーグ戦の事例では，メンバの構成などに大きな差がないことから，チーム力の特性を詳細に調べることができ，コミュニケーションやリーダシップに関する要因が強く作用していることが明らかになっている[7][8]．

ビジネスや IT 分野におけるチーム研究においては，何をもってチーム力が優れているのかを定義するのかが難しいため，チームの優劣を客観的に表す尺度が問題になる．ビジネスであれば，ビジネス上の成功や利益，プロジェクトマネジメントであれば，QCD の達成や向上，契約の達成といったプロジェクトの成功による評価は存在するが，その評価にはチーム力とは別の要因の影響も関わっているため，チーム力を正しく評価することが難しい．

IT 分野では，客観的なチーム力の優劣比較が出来ないことから，2 つの視点から研究が行われている．一つは，従業員満足のように，チームを構成するメンバから見たチームの評価である．この研究としては榎田らのパートナー満足の研究[10]がある．もう一つは，プロジェクトの主観的な判断により成功 / 失敗を 5 段階で測り，チーム力のどのような因子がプロジェクトに影響しているのかを明らかにする松尾谷の研究[9]である．

本研究の課題は，松尾谷が示した，チーム力の優劣を目的変数として，要因を説明するものではない．長期に渡り進めてきたチーム活性化活動の評価として，他のチームと客観的な違いの有無と，違いがある場合には，その違いが何かについて明らかにすることである．

3. チーム力の比較対象と比較方法

事例のチーム力がどのような状態にあるのかを明らかにするため，チーム力の測定に関する松尾谷の研究[9]で蓄積されたデータを利用した．データは，質問紙調査の回答である．本章では，チーム力活性化活動を行ってきたチームの比較対象としたデータの概要，分析対象データの基となる質問紙の概要，チーム力の比較方法について述べる．

3.1. 比較対象としたデータの概要

比較対象は，先行研究[9]のデータ 55 件に，新たに 44 件を加えた，合計 99 件のデータである．このデータの調査期間は 2013 年秋から 2014 年末までの期間であった．調査対象のドメインには，エンタープライズ系，Web 系，組込み系が含まれていた．また，規模も数名から 100 名を超えるものまで多様であった．このデータと，チーム力活性化活動を行ってきた事例チームのデータを比較する．事例チームにも，先行研究[9]と同じ質問紙を使って調査を行い，28 件のデータを取得した．

3.2. 質問紙の概要

質問紙[9]は，強い否定，否定，判断できない，肯定，強い肯定という選択肢による 5 件法で，チーム力の構成モデルに基づき，仲間意識，役割意識，規範意識，成果意識，環境意識の 5 つの構成要素でカテゴリ化された，合計 24 項目の質問で構成されている．各構成要素の質問項目の概要を以下に示す．

仲間意識 ： チームメンバが目的を共有し，その目的に向かって行動することなどについて問う
役割意識 ： チームの中で自分の役割を認識して行動したか，あるいは，他者が自分の役割を認識し行動しているかなどについて問う
規範意識 ： チームで共有しているチーム文化の有無と，その受容について問う
環境意識 ： チームが外部から受ける影響に関して問う
成果意識 ： メンバのスキルや頑張りに対して問う

3.3. 比較方法の概要

事例チームのデータは，チームメンバ 28 名に対して行ったものである．データは，1 チーム分として一つにまとめて比較するのではなく，個々のデータとして， 99 件のデータと比較を行った．これは，チームメンバによって，チームの受け止め方に差があると考えたことによる．

先行研究[9]では，構成モデル単位で主成分分析による尺度を持ち，その値を使ってチーム力の判別を行っていたが，本研究の目的はチーム間の違いを明らかにすることであり，チームの優劣には着目しないため，先行研究[9]の構成モデルは用いないことにした．

評価に用いた統計ツールは，サーバに設置した R (V3.1.2) を RStudio 経由で用いた．ライブラリは，library (psych) と library (MASS) を用いた．他に特別なツールは用いていない．

チーム力比較方法の概略は，次の通りである．

1) データの確認 ： 使用する質問紙の回答データを調べ，比較するデータの範囲を定める
2) チーム間の差の調査 ： 調査項目の全てを使い，線形判別分析で違いの有無を調べる
3) 差が発生している項目の調査 ： どのような質問項目に差があるのかを共分散分析で調べる
4) 差の抽象化 ： 差があった質問項目が意味することは何かを調べるために，主成分分析を行い，明らかにする
5) 抽象化の検証 ： 求めた主成分のみで，事例チームと，他のチームを判別できるのか確かめる

この順序に従って比較分析を実施した．その結果を 4 章で示す．

4. チーム力の比較結果

本章では，3 章で述べた比較方法に基づき，各ステップで実施した分析の説明とその結果を述べる．

4.1. データの確認

まず，データを 2 群に分けて群間で比較を行う．事例のチームのデータを X 群（group X），他チームのデータを Y 群 (group Y) とする．X 群のデータ数は 28 件，Y 群は 99 件である．質問紙の全質問は 24 項目あるが，欠損のある 2 項目を除外した 22 項目(以降，22 項目を全項目と表記する) を対象に X 群と Y 群の差異を調べる．

4.2. チーム間の差の調査

線形判別分析 (LDA) の目的は，群を最もよく分離

する変数の線形結合を求めることにある. LDA は, 正準判別分析とも呼ばれている. ここでは R の LDA として library (MASS) を用いた.

判別分析は, 線形判別式の係数を求め, その式に基づいて X 群と Y 群のそれぞれの判別値を計算する. その計算結果を図 1 に示す. このグラフの y 軸は, 判別分析によって求めた判別値であり, たとえばプラス側が優秀といったような特性を表すものではなく, 統計的な差を調べるための尺度である. この値は平均を 0 として基準化している. x 軸は観測データを順に並べた index である. グラフの ○ 印が X 群を示し, △ 印が Y 群を示す.

図 1 に示した通り, X 群の要素と Y 群の要素は全く異なった値ではないが, 群として見ると明確な差がある.

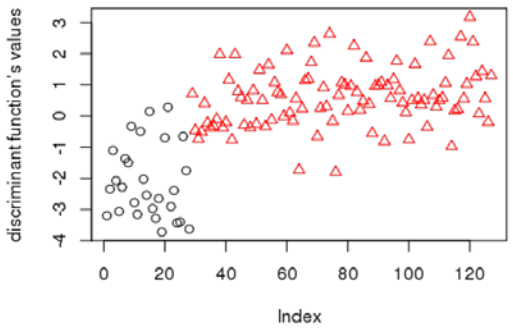


図 1 質問項目を使った判別値の散布図

図 2 は, 箱ひげ図を使って X 群と Y 群の差を示したものである. この図から両群を判別することは容易である. このグラフの y 軸も優劣を示すものではなく, 図 1 と同様, 判別値である.

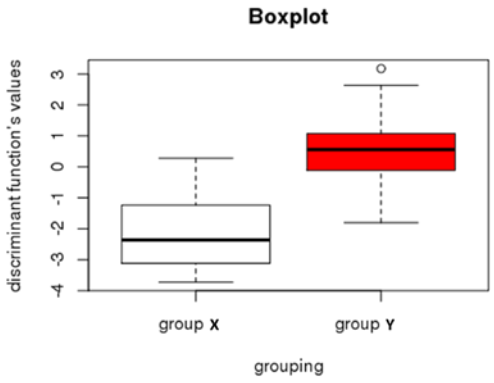


図 2 X 群と Y 群の箱ひげ図

判別式を用いて, X 群と Y 群の各要素 (メンバ) について判別を行ったのが表 1 である. この表の見方は, X 群(表では group X)の 28 要素のうち 21 要素は X 群と判別できるが 7 件は Y 群と判別したことを示している. Y 群については 99 要素中 2 要素だけ X 群と判別したが 97 要素は正しく判別できている. この 75% や 98% を判別率と呼んでいる.

表 1 質問紙全項目による判別分析結果

			予測番号		合計
	結果		group X	group Y	
実測番号	度数	X	21	7	28
		Y	2	97	99
	%	X	75	25	100
		Y	2	98	100

以上のことから, X 群と Y 群では明確な差が存在することが明らかになった.

4.3. 差が発生している項目の調査

全項目に対し, X 群と Y 群の回答に差があるかどうかを質問項目ごとに共分散分析を使って検定した. その結果を表 2 に示す.

表 2 質問項目毎の回答の共分散分析結果

codes	有意水準 (p値)	項目数
***	0 – 0.001	7
**	0.001 – 0.01	9
*	0.01 – 0.05	0
.	0.05 – 0.1	2
N/A	0.1 –	4
	合 計	22

22 項目のうち, 有意水準 (p 値) が 0.01 以下である 16 項目において, X 群と Y 群で統計的な差があることが判った. これにより, 差がでている質問項目が, X 群と Y 群のチーム状態に影響を及ぼしている何らかの要因を含んでいると考えた.

4.4. 差の抽象化

前項の共分散分析により, 有意な 16 項目を使って, それらを構成する何らかの隠れた成分があるかを調べるために主成分分析を行った. その時のスクリープロットを図 3 に示す. スクリープロットとは, 因子分析や主成分分析において, 因子数を決める時に使われる.

図 3 では, 2 つのグラフがあり × 印が主成分分析, △ 印が因子分析を表している. ここで用いたの

は主成分分析であるが，参考として因子分析も示した．図 3 の水平方向の線は固有値と呼ばれる値であり，因子数を決める一つの基準として使われている．因子数を決める方法には，他にスクリープロットが大きく落ち込むところまでを選択することがある．ここでは，固有値を用いて因子数を選んだ．

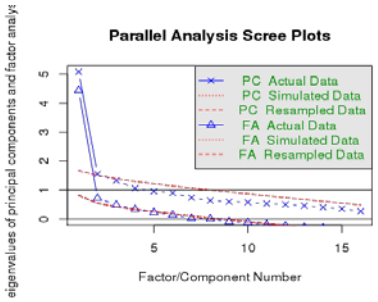


図 3　16 項目に対するスクリープロット

主成分分析により，4 つの主成分が抽出された．以降，抽出した第 1 主成分を PC 1，第 2 主成分を PC 2，第 3 主成分を PC 3，第 4 主成分を PC 4 と記す．

4.5. 抽象化の検証

主成分分析により 4 つの主成分を抽出した．この 4 つの主成分によって，X 群と Y 群の違いを説明できるのか検証を行う．検証の方法は，この主成分によって，X 群と Y 群を用いて判別を行い，4.2 項で行った 全項目による判別と比較する．

検証は，3 つのパターンで実施した．パターンの違いは，判別に用いる主成分の数である．表 3 に，用いた主成分の数を 4 つ， 3 つ，2 つとした 3 パターンと，それぞれの判別結果を示す．

表 3　主成分数毎の判別分析結果

	主成分数			予測番号		合計
		結果		group X	group Y	
実測番号	4	度数	X	21	7	28
			Y	0	99	99
	(PC1 - PC4)	%	X	75	25	100
			Y	0	100	100
	3	度数	X	21	7	28
			Y	0	99	99
	(PC1 - PC3)	%	X	75	25	100
			Y	0	100	100
	2	度数	X	21	7	28
			Y	0	99	99
	(PC1 - PC2)	%	X	75	25	100
			Y	0	100	100

判別を行った結果，主成分の上位の 2 主成分（PC 1，PC 2）でも 4 主成分でも判別率は変わらなかった．4.2 項で行った　全項目による判別率と比べると，X 群については同じだが， Y 群については 98% から 100% に向上していた．図 4 に，上位 2 主成分による判別値の散布図を示す．

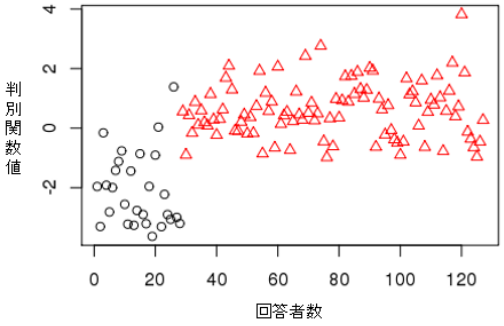


図 4　PC 1 と PC 2 による判別値の散布図

結果として，主成分数を減らしても判別率に変わりはなく，PC 1 と PC 2 の 2 主成分で判別可能であることが判った．

5. 考察

4 章での分析により，質問紙調査のデータを使った X 群と Y 群の比較において，有意な差があることが判った．そして，主成分分析により，その差が 4 つの主成分で説明できることを明らかにした．また，判別式による検証の結果，上位 2 つの主成分が両群の差に大きな影響を持っていることが判った．そこで，本章では，まず，抽出した主成分がどのような意味をもつのか，チーム力の構成要素はどのようなものなのかについて考察する．そして，本研究の目的であるチーム力の比較結果についてまとめる．

5.1. 主成分の考察

4.4 項で抽出した 4 つの主成分に対して強く影響を与えた各質問項目の意味から，それぞれの因子に名前を付けた．各因子の名前を表 4 に示す．

表 4　16 の質問項目に対する主成分分析結果

因子	因子の名前
PC 1	自分たちのチームに対する自信や信頼
PC 2	役割に対する納得感や態度
PC 3	仕事に対する技術的な誇り
PC 4	仕事に対する自由度

各因子に影響を及ぼした質問項目の意図は以下の通りである．

・PC 1: 自分たちのチームに対する自信や信頼
同じチームでこれからも仕事を続けたいなど，仕事の満足感などが含まれる
・PC 2: 役割に対する納得感や態度
チームにおける仕事の分担や役割について自覚があり，積極的な態度があることが含まれる
・PC3 : 仕事に対する技術的な誇り
仕事から受ける自己成長や成長への刺激が含まれる
・PC 4: 仕事に対する自由度
仕事の進め方などについてやらされ感がなく，工夫などが出来ることが含まれる

5.2. チーム力の構成要素

両群の差に影響を与える因子は，「自分たちのチームに対する自信や信頼」，「役割に対する納得感や態度」，「仕事に対する技術的な誇り」，「仕事に対する自由度」の 4 つである．今回の比較の結果から，これらの因子が，事例チームがチームビルディング活動によって培ったチーム力の主要な構成要素であると考える．

また，表 4 で示した通り，抽出した主成分を用いて行った判別分析では，主成分数を減らしても判別率に変わりはなく，PC 1 と PC 2 のみで判別可能であった．この主成分分析の中で両群の差に対して特に大きな影響を持つ PC 1 と PC 2 は，「自分たちのチームに対する自信や信頼」と「役割に対する納得感や態度」である．この 2 因子は，先行研究[9]で示された構成モデルの「仲間意識」と「役割意識」とほぼ同じものである．分析のアプローチを変えても，チーム力に影響を与える因子として同じ要素が抽出された．このことから，チーム力に特に大きな影響を与える要素は，「仲間意識」と「役割意識」の 2 つであるといえる．

5.3. チーム力の比較結果のまとめ

この研究の目的は，事例としたチーム活性化活動の効果を，他のチームとの比較により明らかにすることである．質問紙調査の回答データを用いて，28 件の事例チームのデータと，99 件の他のチームのデータとを次の分析方法により比較し，結果を示した．以下に，チーム力の比較結果についてまとめる．

1) 全データ項目を使った線形判別分析
全回答データを用いて行った判別分析の結果，事例チームの判別率は 75 % ，他チームの判別率は，98% であり，両チームの間には判別可能な違いがあることが明らかになった．

2) 判別分析に寄与している質問項目の抽出
質問項目ごとに共分散分析を使って検定を行った結果，22 項目中の 16 項目において，両チームの間で統計的な差があることが判った．この 16 項目が，判別分析に寄与している質問項目である．

3) 質問項目の主成分を用いた線形判別分析
両チーム間で統計的な差がある 16 の質問項目を用いて，主成分分析を行った結果，4 つの主成分を抽出した．抽出した主成分を用いた判別分析の結果，事例チームの判別率は 75 % ，他チームの判別率は，100% となった．全データ項目を使った場合の判別率と比べると，事例チームは変わらないが，他チームは 98 % から 100% へと向上した．これにより，抽出した主成分で両チームの差を説明できることが明らかになった．

4) 主成分の考察
各主成分に影響を与えた質問項目から考察し，抽出した 4 つの因子を「自分たちのチームに対する自信や信頼」,「役割に対する納得感や態度」,「仕事に対する技術的な誇り」，「仕事に対する自由度」と名付けた．この 4 つの因子が，チーム力に影響していると推測する．

6. おわりに

これらの分析により,「チーム間で違いがあるのか」については，明確な違いがあることが明らかになった．また,「違いがあるとすればどのような違いなのか」については，違いに影響を与える 4 因子を抽出した．さらに，特に大きく影響する因子は，先行研究[9]で示された「仲間意識」と「役割意識」に相当する 2 因子であることが判った．

本研究では，両チームの間に違いがあることと，その違いを構成する要素を明らかにすることができた．しかし，データ数はまだまだ少なく，IT 分野における他のチームでも同様の分析結果が出るのかどうか確認できていない．チーム活性化活動の事例を増やすとともに，質問紙を公開し，分析の確度を上げていくことが今後の課題である．

謝辞

本研究にあたり, データをご提供いただいた PS 研究会 MM4 のみなさま, 対象チームのみなさまに深く感謝いたします.

参考文献

［1］上田泰，組織行動研究の展開，白桃書房，2003/01. 172-174
［2］内閣府，平成 19 年度版 国民生活白書 つながりが築く豊かな国民生活，http://www5.cao.go.jp/seikatsu/whitepaper/h19/10_pdf/01_honpen/index.html
［3］Boehm, B. Clark, B. Horowitz, E. Westland, C. (1995). Cost models for future software life cycle processes: COCOMO 2.0. Annals of software engineering 1.1: 57-94.
［4］増田礼子，人を育て技術を実現するチーム力 ～ 9 年間の活動から得られた TIPS ～，ソフトウェア・シンポジウム 2014.
［5］増田礼子, チームビルディングから組織文化へ – チームビルディング継続実施の効果 - , ソフトウェア品質シンポジウム 2014.
［6］Hackman, J. Richard, and Ruth Wageman. "A theory of team coaching." Academy of Management Review 30.2 (2005): 269-287.
［7］池田浩, et al. 「チーム・メンタルモデルおよびチーム・パフォーマンスを規定する要因に関する検討―チーム力およびチーム・リーダーシップの効果―」 福岡大學人文論叢 44.2 (2012): 293-309.
［8］河津慶太, 杉山佳生, and 中須賀巧. "スポーツチームにおける組織市民行動, チームメンタルモデルとパフォーマンスの関係の検討―大学生球技スポーツ競技者を対象として―." スポーツパフォーマンス研究 4 (2012): 117-134.
［9］松尾谷徹,「IT に現場力は存在するのか:その計測と評価の試み」, ソフトウェア・シンポジウム 2014.
［10］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～」, プロジェクトマネジメント学会誌 Vol.7, No.1 pp.15-20.

大学機関調査研究 IR へのデータ管理成熟度 DMM モデルの軽量な適用

日下部　茂
九州大学大学院システム情報科学研究院
kusakabe@ait.kyushu-u.ac.jp

大石 哲也　　森 雅生 *　　高田 英一
九州大学大学評価情報室
oishi, mori, takata@ir.kyushu-u.ac.jp

要旨

IR とも称される大学機関の調査研究活動では，大学における計画策定や政策立案，意思決定を支援する情報の提供に必要な調査研究活動をデータにもとづいて行う．日本においても近年特にデータに基づく評価・改善が求められ *IR* に対する期待は大きい．その一方で，日本の大学に適した *IR* の導入・運用・改善などはまだ確立しているとは言い難い．我々は，データ管理に関する先進的な実践に向けたプロセス改善を行うための包括的な参照モデルとして提唱されている，*Data Management Maturity (DMM)* モデルの *IR* での活用を試みている．*IR* の活動内容は機関ごとに異なる可能性があり，また時代とともに変化するもの多いものの，データの収集・管理は共通して重要であり，その点に着目したモデルの活用は有用と考える．本稿では，九州大学の大学評価情報室を中心とした *IR* 活動における，データ管理に関する改善の一手法として *DMM* モデルを軽量に適用したケースについて述べる．

1. はじめに

Institutional Research (以下 IR) とも称される大学調査研究は，大学における計画策定や政策立案，意思決定を支援する情報の提供に必要な調査研究活動をデータに基づいて行う [9]．調査研究活動の分析結果を組織の改善につなげる重要性はますます高まる一方，大学を取り巻く社会情勢や関連する技術は変化を続けている．IR の効果的な導入と継続的な改善のためには，IR の具体的な活動内容を，導入時だけでなく，継続的にカスタマイズすることが必要である．例えば，IR の活動のうち，重要な基盤に相当するデータの収集・管理に関しては，導入時の目的に応じた調査研究項目の選定に加え，ICT 技術の進展によるデータの多様化・大規模化にも継続的に対応する必要がある [10]．

ICT 技術の進展により，様々なシステムやデバイスからビッグデータと呼ばれるデータを取得しその分析を行うデータサイエンスの取り組みも盛んになっており，IR に関しても機関内で取得可能なデータの種類や規模も増加している．例えば，学生の成績データや教員の業績データに加え，講義の出席状況，図書館や食堂の利用状況といった活動のデータも取得可能となってきている．しかしながら，IR の目的に沿った解析だけでなく，その前提となるデータの取得や管理といった活動の実践も必ずしも容易ではない [6]．そのようなプロセスの確立や改善に着目することも重要である．

IR はもともとはアメリカ合衆国で提唱されたものであるが，歴史的な経緯も反映して複数の定義があるとされており，必ずしも統一的な定義はないとされている．国レベルの観点では，日本の高等教育機関の評価方法は合衆国のものとは同じではなく，合衆国での個別の成功事例をそのまま導入しても効果的なものになるとは限らない [6][8]．機関レベルの観点でも，実践の具体的な詳細レベルでは各機関ごとの状況に依存した様々なバリエーションがあり得る．このような理由で，IR 導入や改善の方法を一律に述べることは事実上不可能であり，既報告の導入事例があったとしても，自らのものと直接比較することは必ずしも容易ではない．

実際に IR を導入・改善するとなった場合，その実践にあたっては自機関のデータ資産に関するプロセスを中心に何らかのプロセスを確立・改善する必要がある点は共通する．このような観点から，IR というドメインにおけるベストプラクティスを反映したプロセスモデルを

＊ 2015 年 4 月より東京工業大学・情報活用 IR 室

適切な抽象レベルで設定し参照可能とすることで，モデルを指針としたIRの導入・改善の推進を目指す．

本稿では，データ管理に関する先進的な実践に向けたプロセス改善を行うための包括的な参照モデルとして提唱されている，Data Management Maturity (DMM) モデルをIRで活用する試みについて述べる．第2節で，IRの概略を説明し，第3でDMMモデルの紹介を行う．第4節では，九州大学の大学評価情報室を中心としたIR活動の，データ管理に関する改善の一手法としてDMMモデルを軽量に適用したケースについて述べる．

2 IRについて

2.1 IR概要

現在広く用いられているIRの定義 [3] では「機関の計画立案，政策形成，意思決定を支援するための情報を提供する目的で，高等教育機関の内部で行われる研究」とされている．そのようなIRを担当する部門の学内での役割はおおよそ次のようなものといわれている [9]．

1. 機関の業績についてのデータを収集する，
2. 機関の環境についてのデータを収集する，
3. 収集したデータを分析，解釈する，
4. データの分析と解釈を機関計画策定，政策策定と意思決定のもとになるような情報に変換する

IRについてまとめたものとして他に以下のようなものもある．現代のIR組織のミッションレベルでの活動内容を分析した研究では，現代のIR組織における機能が類型化されている [4]．大学はめまぐるしく変化する外部環境に適応する必要があるとされ，IRはそこで重要な役割を果たすことが期待されている．そのような観点からのアメリカの高等教育機関の歴史的変遷とIRの特色もまとめられている [5]．

近年，日本の高等教育関係者の間でも，IRが注目を浴びるようになってきている．教員個人の主観や経験値に基づくのではなく，各大学における様々なデータを収集し，客観的に分析，数値化・可視化し，評価指標と関連付け，その結果を教育・研究，学生支援，経営等に活用することの重要性が認識されてきている．

2.2 データの重要性

IRが関係する領域は，教育や経営を含めかなり広範囲にわたるが，大学の諸活動に関するデータの収集と分析をITシステムによって効果的に行うことがその基盤的な活動になる．データに基づいて戦略立案を行うには，データの収集や解析なしには不可能である．より精緻な戦略立案を行うにしても，収集するデータの規模の拡大や精度の向上も必要となる．具体的な実践においては，データをどのように集積し，測定し，そしてそれらの分析の結果を改善につなげていくかということは大きな課題となっている．

高等教育機関の管理運営における意思決定の過程のモデル [2] にも，組織内で問題が特定されたとき，その解決策を検討するためには「データの収集」という段階が必要とされている．このようにデータを収集・管理する能力は，これまでのIRの変遷でもIR担当者に求め続けられており，それは今後も継続するものと考える．

2.3 IRとプロセス改善モデル

具体的なIRの実践には何らかのITシステムの利用が事実上必須であり，データ収集・管理を含め，IRの実践にソフトウェアの果たす役割は大きく今後もますます大きくなると予想する．IRの効果的な導入や継続的改善に関してソフトウェア工学の果たす役割も大きく，ソフトウェア工学分野でのプロセス改善モデルの活用も有用と考える．

代表的な改善モデルのフレームワークに能力成熟度モデルCMMIがある．開発のためのCMMI-DEV，サービスのためのCMMI-SVC，獲得のためのCMMI-AQCはこのCMMIのフレームワークを基にしている．IRは多様な側面があり，着目する観点によっては，これらのモデルをIRに活用できる可能性がある．アメリカの大学にはいわゆる情報システム部門のようなものがありIR部門と役割分担ができていることが多いとされる．一方，日本の場合はデータ収集，ITシステムの開発からIR部門がかかわることが多いとされており [10]，そのような場合はCMMI-DEVも有用な可能性がある．

本稿では，IRがデータに基づく調査研究であることから，データに着目した改善モデルが有効と考え，DMMモデルに着目する．参照モデルを活用することにより，導入や改善の指針を立てやすくでき，直接的な比較が困

難な個別の事例の知見も，標準的なモデルを介することで，参照や比較が容易になると考える．

IR に関連する業務プロセスの確立の度合いや，プロセスのテーラリング能力のばらつきもあるため，データの取得や管理に限ったとしても，事例間の一律の評価・比較が難しく，他機関での成功事例があったとしてもその知見を活用するのは容易でない．そのようなばらつきの問題がある中で，モデルを介することで，モデルレベルでの評定を可能とし，IR の導入や改善を見通しよく円滑に実施し，直接的な比較が困難な他組織の事例の知見を活用した IR のテーラリングを容易にすることを目指す．また，様々な機関で多様なデータが収集され，分析されてきている中で，機関ごとの境界を越えて IR を推進する際にも参照モデルの利用は有効だと考える．

3. DMM モデル

3.1 概要

DMM モデルは，データ管理に関する基本的なビジネスモデルと，段階的な成熟度に結び付けられた固有の能力を定めている．データ管理のプラクティスを評価するための包括的なフレームワーク，データ統治の漸増的なスコープと取り組み，あらゆる組織に採り入れられ適用されるためのベストプラクティスの実装，といったものを提供している．利用者は文書化されたベストプラクティスに対して自らを評価し，理想と実際のギャップを明らかにし，データ資産の管理を改善を行う．

DMM モデルのプロセス領域と，インクリメンタルな能力測定基準であるプラクティスの記述文は，組織におけるデータ資産の効果的な管理を達成，維持するために必要とされているような，実践的で効果が実証済みのアクティビティに基づいている．DMM モデルは， 組織のデータ資産の管理， 関連するアクティビティに対する，プロセス改善と能力成熟度のモデルであり，データの生成から納品，維持管理，アーカイブまでのデータのライフサイクルにまたがる，効果的なデータ管理の構築，維持および最適化，といったものに対するベストプラクティスを含んでいる．

モデルの全体的なゴールは，組織の重要なデータ資産の管理における熟練さを改善する手助けをすることで，継続的な改善，法令順守，監査に対する適切なベンチマークを提供する．後述するプロセス領域の集合を持ち，業種を問わず，どのようなデータ管理の目的にも適用できるよう考えられている．漸増的な能力度と規律あるプラクティスを通して，データを重要なインフラストラクチャとして組織が理解して取り扱うようにする．能力度の現在の状態を評定するためだけでなく，データ管理の実装に対するカスタマイズされたロードマップを作るためにも使えるように構成されている．しかしながら，DMM モデルは，効果的なデータ管理のための要件とアクティビティを定義する一方，これらの能力度を各組織がどのように達成するかの処方はしない．

DMM モデルに関して，以下のものが CMMI Institute より提供されているが，今回はモデルのドキュメントのみを用いた自組織内での軽量な適用を行った．

- 組織のデータ管理プログラムの客観的な評価を実施するための標準的な方法
- 組織レベルのパートナープログラム
- Enterprise Data Management Expert (EDME) の証明書獲得につながる一連のトレーニングコース

3.2 DMM モデルの構成要素

CMMI のプロセス領域にもとづき，20 個のデータ管理プロセス領域と，5 つの支援プロセス領域がある (表 1 参照)．各プロセス領域は以下の構成要素からなる．

- 目的の記述文
- 導入説明
- ゴール
- 核となる質問
- 関連プロセス領域
- 機能面でのプラクティス (Levels 1-5)
- 作業成果物の例

これらのうち，目的の記述文，ゴール，核となる質問，関連プロセス領域，作業成果物の例，が「説明のためのモデル構成要素」で，機能面でのプラクティスおよび，個別のプロセス領域と別に設定されているインフラストラクチャの支援プラクティス (後述) が「モデル遵守のために必要とされるもの」になっている．

各組織では，固有のニーズに合わせて，単一もしくは複数のプロセス領域，単一のカテゴリもしくは複数のカテゴリなどの任意の組み合わせを選んで実装，評定などができる．各プロセス領域ごとのプラクティスの達成を評価することで能力度の評定を得ることができる．

また，成熟度の評定に使う，以下のようなインフラストラクチャ支援プラクティスを定めており，プロセス領域のプラクティスの達成と，これらの達成を同時に評価することで，成熟度の評定を得ることができる．

- IS1 機能に関するプラクティスを実行する
- IS2 管理されたプロセスを実装する
- IS3 組織の標準を定着させる

表 1. DMM モデルのプロセス領域

Data Management Strategy カテゴリ
- Data Management Strategy
- Communications
- Data Management Function
- Business Case
- Program Funding
Data Governance カテゴリ
- Governance Management
- Business Glossary
- Metadata Management
Data Quality カテゴリ
- Data Quality Strategy
- Data Profiling
- Data Quality Assessment
- Data Cleansing
Data Operations カテゴリ
- Data Requirements Definition
- Data Lifecycle Management
- Provider Management
Platform and Architecture カテゴリ
- Architectural Approach
- Architectural Standards
- Data Management Platform
- Data Integration
- Historical Data, Archiving and Retention
Supporting Processes カテゴリ
- Measurement and Analysis
- Process Management
- Process Quality Assurance
- Risk Management
- Configuration Management

4 事例

ここでは，九州大学の大学評価情報室を中心とした機関別認証評価 [11] のような IR 活動における、データ管理に関する改善の一手法として DMM モデルを軽量に適用したケースについて述べる．DMM モデルでは，能力・成熟度レベルの評定が可能であるが，今回の試行はレベル獲得ではなく，業務の PDCA を意図したものである．

4.1 国立大学法人の IR

日本の国公私立大学は，認証評価機関による評価を 7 年に一度受ける必要がある．また，国立大学法人は，中期目標・中期計画の業務実績の評価（国立大学法人評価）を 6 年に一度 (年度計画については毎年度) 受ける必要があり，大学の戦略的計画である中期目標・中期計画の策定とその法人評価において，教育研究活動に対する有効性に基づいた説明責任が強く求められている．評価結果が運営費交付金の配分において考慮されることとなり，大学の管理運営にあたっては，大学の諸活動に関する現状把握に向けた様々な情報の収集と分析を行い，大学内部で評価・改善を進めることが必要となっている．そのため，各大学内で，部局ごとに散在している財政，学生，教学などに関するデータをどのように収集，管理して活用していくかに関する取り組みがなされ，IR に対する期待も高くなっている．

文献 [7] では IR 組織を有する名古屋大学，愛媛大学，九州大学の設置目的と実際の業務内容との比較を行っている．表 2 は三大学のミッション分析と実際の活動状況分析を示している．○がミッションとしての該当項目，□が実際の活動状況の領域を示している．分類としては文献 [4] の九つの分類を用いている．

これらの大学の IR 組織の設置規定の共通する特徴は，業務の中心が大学評価に関わる業務への支援で，次にその業務に関わるデータの収集・管理・分析である．このような背景の下，九州大学の評価情報室の取り組みを中心に，特にそのデータ管理に関して DMM モデルによる評価と改善方針の検討を行った．

4.2 DMM モデルの軽量な適用

DMM モデルを用いることで，標準的な評定の方法論により，ベストプラクティスと比較して組織が自らを測

表 2. 設置規定と活動内容の比較

	名古屋大学	愛媛大学	九州大学
計画策定支援	○□	○□	
意思決定支援		□	
政策形成支援		□	□
評価活動への支援	○□	○□	○□
個別の調査研究		○	
データ管理	○□	○□	○□
データ分析	○□	○□	○□
外部へのデータ提供と報告			○□
内部向けレポート作成	○□	○	○□

定できる．今回着目する，九州大学の大学評価情報室は，自己点検・評価体制の整備・確立の一環として，平成 13 年に設置された評価情報開発室をその前身としており，以下の趣旨により平成 16 年に大学評価情報室となった．

- 国立大学法人化により評価が大学財政とリンクされ評価の重要性がさらに高まった状況に対応すること．
- 国立大学法人化により自由度ともに責任も増大した大学運営に資する情報の開発・提供を機能的・機動的に行うこと．

このような評価室の主要関係者へのインタビューやドキュメントの参照などを通して，能力度や成熟度のレベル達成ではなく，実務的な PDCA サイクル実施，暗黙的な知見の明示化といった観点で，以下に述べるようなカテゴリのプロセス領域に対して評価を行った．

4.2.1 Data Management Strategy カテゴリ

このカテゴリのプロセス領域は，データ管理に対する共同のビジョンを確立，伝達，正当化し，資金を供給するためのベストプラクティスに関するものである．

Data Management Strategy データ管理プログラムのためのビジョン，目標，目的を定義し，関連するすべての利害関係者が優先事項とプログラムの実装と管理の上に位置づけられることを確実にする．

Communications ポリシー，標準，プロセス，進捗の公表，およびその他のデータ管理コミュニケーションがフィードバックに基づいて，公表され，制定され，理解され，調整されていることを確実にする．

Data Management Function データが組織の資産として管理されていることを確実にするための，データ管理のリーダーシップとスタッフに向けたガイダンスを提供する．

Business Case どのデータ管理の取り組みに資金提供すべきかを決定するための理論的根拠を提供し，組織への財政上の配慮と利益に基づいた意思決定を行うことで，データ管理の持続可能性を確実にする．

Program Funding データ管理プログラムをサポートするための十分かつ持続的な資金調達の可用性を確実にする．

4.2.2 Data Governance カテゴリ

このカテゴリには，実践への幅広い参加と，データ管理の有効性を確実にするためのベストプラクティスに関するプロセス領域が属している．

Governance Management 組織のデータが重要な資産として管理され，効果的で持続可能な方法で実装されることを確実にするために必要な，所有権，管理，および運用構造を構築する．

Business Glossary 全利害関係者のためのビジネスプロセスを支援する構造化データと非構造化データに関する用語と定義の共通の理解をサポートする．

Metadata Management メタデータ管理は，管理下にある構造化および非構造化データ資産について，明確かつ組織的な情報を特定，拡張し，データ共有を育成，支援し，データの法令に遵守した利用を確実にし，ビジネスの変化への応答性を改善し，データ関連のリスクを低減するための，プロセスとインフラを確立する．

4.2.3 Data Quality カテゴリ

事業の運営，意思決定，および計画策定における，意図に沿った利用に対する適合性の確保の目的での，データの欠陥を検出，評価，洗浄するための協調的なアプローチを定義し実現するためのベストプラクティスに関するプロセス領域が含まれる．

Data Quality Strategy ビジネスの目標と目的をサポートするために必要なデータ品質のレベルを達成し，維持するための統合された組織全体の戦略を定義する．

Data Profiling 管理下のデータの指定されたセットの内容，品質，およびルールの理解を深める．

Data Quality Assessment プロセス，技術に従い，データの品質ルールに照らし合わせてデータの品質を測定，評価するための系統的アプローチを提供．

Data Cleansing データクレンジングは，事前に定義されたビジネスルールに従って，データの正当性を検証し，修正するために使われる，メカニズム，ルール，プロセス，方法を定義する．

4.2.4 Data Operations カテゴリ

データの要件を指定し，サプライチェーン全体で実装されたデータを管理するためのベストプラクティスに関するプロセス領域が属する．

Data Requirements Definition 生産され消費されるデータが，ビジネス目標を満足させ，すべての関係者に理解され，データを生成，消費するプロセスと一貫性がとれていることを確実にする．

Data Lifecycle Management データの生成または取得から廃棄までのデータのライフサイクル全体を通して業務プロセスを流れるデータフローを，組織が理解し，マップし，目録を作り，制御していることを確実にする．

Provider Management ビジネス要件を満たし，データのプロビジョニングの合意を一貫して管理するための，データの内部および外部の供給を最適化する．

4.2.5 Platform and Architecture カテゴリ

このカテゴリのプロセス領域は，実装されたデータ管理プラットフォームが業務の目的をサポートするための組織のデータ資産を統合，アーカイブし，保持することを確実にするような，方法や基準を確立するためのベストプラクティスに関するものである．

Architectural Approach 業務および技術目標を達成するための，データの取得，生産，保管，および配信を可能にするような，最適なデータ層を実装する．

Architectural Standards データ資産コントロールと情報の効率的な使用と交換にとって基本的な，公認されたデータ表現，データアクセスとデータ配布を支援するアーキテクチャ上の要素を統治するための，公認された標準を提供する

Data Management Platform ビジネス・ニーズを満たすために，効果的なプラットフォームが実装，管理されることを確実とする．

Data Integration 複数の情報源からデータを獲得し，データ分析のような，データの統合や集約を必要とする業務プロセスにおけるデータの可用性を改善する必要性を削減する．データ統合は，ソースデータの最適化，集中化によるコスト削減の実現，改善されたデータ品質を可能にする．

Historical Data, Retention and Archiving データの保守が，履歴データの可用性に対する組織と規制の要件を満たすこと，データのアーカイブと保持に対する法と規制の要件が満たされていることを確実にする．

4.2.6 Supporting Processes カテゴリ

すべてのプロセス領域における，データ管理の有効性の評定および実装に対して求められる，業務プロセスおよび能力の定義に関するプロセス領域が属している．

Measurement and Analysis データ管理活動を管理，改善することを支援するための，測定能力と分析技術を開発し維持する．

Process Management 組織プロセス資産の有用な集合を確立，維持し，ビジネスの目標と目的と組織のプロセスにおける現在のギャップによって把握される，組織プロセス改善を，計画，実装，展開する．

Process Quality Assurance スタッフと経営陣にプロセス実行と関連作業成果物に対する客観的な洞察を提供する．

Risk Management 目的の達成を確実にするような適切な行動を取るために潜在的問題を識別して，分析する．

Configuration Management 構成の識別，制御，状態の説明，および監査を使用して運用環境の整合性を確立して維持する．

4.3 考察

認証評価の報告書を作成するために必要なデータ管理戦略は，取り組みの当初から重要性が認識され，親委員会から部局に至るまでの戦略に関係する組織が構成され，関連するプラクティスが実施されていた．しかしながら，過去の経緯もあり，後述する組織レベルのデータの統治や品質に関する弱みなどがあった．

大学評価情報室の前身は，教員レベルの自己点検・評価体制の整備・確立のためのものであったため，組織レベルではデータ統治と品質に関しては必ずしも理想的なものとはなっていない．中期目標の達成状況の評価は法人を構成する学部・研究科等の現況分析結果を踏まえて実施される一方，構成組織レベルではデータ管理に関する戦略，統治，品質などにばらつきがある．また，縦割りの弊害，慣習的で不統一な用語，必ずしも網羅的な想定ができなかったことによるアドホックな運用，といった問題があった．データの品質に関しても，データの要求定義に関するプロセス領域などと関連して，改善が望ましい項目があった．

軽量の適用であっても，DMM モデルによってデータ管理プロセスの強みや弱みを評価できた．しかしながら，DMM モデルでは，どのようにデータ管理を立ち上げ，改善していくかの処方箋は示されない．他の改善モデル，例えば開発のためのプロセス改善モデル CMMI-DEV では，モデルのインスタンスとして，チームレベルプロセス TSP や個人レベルプロセス PSP がある．IR 分野でも，DMM モデルのような改善モデルをふまえた，カスタマイズ可能な実装用のプロセステンプレートを実現できれば，具体的な改善活動の推進が容易になると考える．

5 おわりに

IR の導入･改善のための評価の参照モデルとして DMM モデルに着目し，九州大学の大学評価情報室を中心とした IR 活動におけるデータ管理活動の改善の一手法として，DMM モデルを軽量に適用した．系統的に強みと弱みを評価でき，改善のための指針を立てることができた．しかしながら DMM モデルでは処方箋は示されないので，改善の実現方法については今後具体化する必要がある．また，汎用である DMM モデルを IR ドメインに向けにカスタマイズするなどして取り組みをさらに発展させる予定である．

参考文献

[1] Data Management Maturity (DMM) Model, cmmiinstitute.com/data-management-maturity

[2] Hughes, R. and Miller, B. W., The Administrator's Task in Goal Setting, Planning, Programming, Budgeting, and Decision Making: The Scientific Decision-Making Process as a Basis for Planning and Problem Solving, Leadership in Higher Education: A Handbook for Practicing Administrators, Miller, Hotes, Terry eds., Greenwood Press, Westport, CT, 1983 年

[3] Saupe, J. L., The Functions of Institutional Research 2nd, Association of Institutional Research, 1990 年, http://files.eric.ed.gov/fulltext/ED319327.pdf

[4] Thorpe, S. W., The Mission of Institutional Research. 26th Conference of the North East Association for Institutional Research, 1999 年

[5] Peterson, M. W., The Role of Institutional Research: From Improvement to Redesign, New Directions for Institutional Research, No.104, pp.83-103, 1999 年

[6] 中井俊樹，鳥居朋子，酒井正彦，池田輝政，名古屋大学における経営情報システムの構築，名古屋高等教育研究，No.3，pp47-65, 2003 年

[7] 小湊卓夫，中井俊樹, 国立大学法人におけるインスティテューショナル・リサーチ組織の特質と課題，大学評価・学位研究, 第 5 号, 2007 年

[8] 加藤毅, 鵜川健也, 大学経営の基盤となる日本型インスティテューショナル・リサーチの可能性, 広島大学高等教育研究開発センター, 大学論集第 41 集, pp.235 － 250, 2010 年

[9] 山田礼子他，高等教育における IR（Institutional Research）の役割，私学高等教育研究叢書, 日本私立大学協会附置, 私学高等教育研究所，2011 年

[10] 森雅生，実践的な機関調査とは，大学職員論叢, No.2, pp.69-77, 2014 年

[11] 高田英一，森雅生，機関別認証評価を機会とした組織単位の IR の機能の実現の取組，第三回大学情報・機関調査研究会，2014 年

オンデマンド受注生産システム開発へのサービスデザインの適用

宗平　順己
株式会社ロックオン
Toshimi_munehira@lockon.co.jp

要約

3D プリンタに代表されるオンデマンド型の受注生産システムの開発にあたって，受注部分の効率化のためには不特定多数の顧客に BtoC 型の EC(E-Commerce)システムを構築する必要がある．本発表ではその要件定義に国際的な潮流であるサービスデザインを適用し，素晴らしい顧客体験を生むシステムを構築することができたので，その結果を報告する．

1. オンデマンド型受注生産システムとその課題

Maker の発刊により 3D プリンタが世の中に知られるようになり，ネットを用いたオンデマンド型の受注生産システムが試作だけでなく，一品受注製品生産にも利用されるようになった．そのわかりやすい例がオンラインプリントである．

ネット受注の場合，試作とは異なり製品製作の場合は，ネットで顧客がリクエストしたものと，実際の製品との間で違いが発生することは許されない．これまでの商流では，営業マンが印刷見本を届け，内容を確認した上で本工程に着手するというアプローチが採用されていたが，ネット受注では，この確かさを担保しつつ，顧客側，事業側双方のコスト削減を図る必要がある．

2. サービスデザインの適用

顧客側のコストも削減する BPM のアプローチとしては「リーン消費」というものがある．

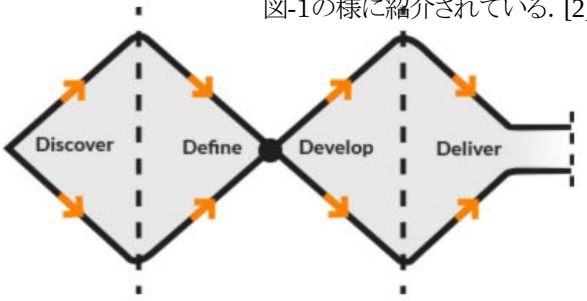
しかしながら，ネットでのビジネスは効率化だけでなく UX（User eXperience）をも考慮したサイトを構築しなければ，事業としては成立しえない．加えて，今回のテーマでは実物の品質保証というコスト削減のみを目的とした検討では対応できない目標もあり，ビジネスモデル全体の再設計が必要となる．

そこで，顧客経験を起点にビジネスを設計するために，近年欧米を中心に取り組みが盛んになっているサービスデザインのアプローチを適用することとした．

3. サービスデザインの特徴とそのプロセス

3.1. エクスペリメントデザインのプロセス

顧客経験（CX：Customer eXperience）の設計において重要なことは顧客インサイト（顧客の洞察から引き出された顧客の行動や態度の根底にある本音、核心）を得ることである[1]が，具体的にはどのように得れば良いのであろうか．エクペリメントデザインを受託しているイギリスのサービスデザイン会社 Engine 社のプロセスは図-1の様に紹介されている．[2]



DISCOVER: We research with people and explore the services they use.

DEVELOP: We develop concept propositions and experiences that are aligned to operational systems.

DEFINE: We translate insights to generate detailed, practical solutions.

DELIVER: We produce intuitive and actionable outputs for implementation.

図-1 Engine 社のエクスペリエンスデザインプロセス

DICOVER：顧客とともに現在のサービスについて調査をする.
→隠れている問題を見つけ出していく. そのため矢印が広がっている.
DEFINE：DISCOVER で得られたインサイトを, 実際的かつ詳細なソリューションへと昇華させる.
→問題解決のアイデア出しをし選択する. このため矢印がいったん収束する.
DEVELOP：新しい価値提案と顧客経験を業務で実行できるものとして開発する.
→集約されたアイデアが業務レベルに展開される. このため矢印が広がっている.
DELIVER：実際の業務に組み込まれ直感的で次のアクションを起こすことのできる成果を生み出す.
→実際の業務に導入し、具体的な成果がでるように調整をかけていく.

また、同社は上記のプロセスの特徴を以下のように示している.

- Highly collaborative
- Led by exploratory research
- Detailed and rigorous
- Strategic in nature
- Creative
- Grounded in delivery
- Visual and tangible
- Customer-centered
- Designed for multichannel
- Aimed at simplifying complexity
- Focused on finding and creating measureable value.

「Highly collaborative」, すなわち「密接に強調して」いることを一番に挙げられているが, これは顧客インサイトを得るのに有効な姿勢とされており,「参加型デザイン」とか「コ・クリエーション」と呼ばれている創造の方法として実践されている.

3.2. IDEO 社のイノベーションプロセス

有名なデザイン・ファームである IDEO 社のイノベーションプロセスは次の様に紹介されている. [3]

1.理解　　4.評価とブラッシュアップ
2.観察　　5.実現
3.視覚化

一方, IDEO と連携しているスタンフォード大学のデザインスクール「d.school」の指導するデザイン思考の 5 ステップは一般に広く知られており, 以下のような内容となっている.

・【ステップ 1】Empathize：共感
・【ステップ 2】Define：問題定義
・【ステップ 3】Ideate：創造
・【ステップ 4】Prototype：プロトタイプ
・【ステップ 5】Test：テスト

IDEO 社のプロセスとは若干異なるが, まずは, 顧客インサイトを得るところからスタートしており, そのキーワードは「共感」である.「共感」は, Sympathize ではなく Empathize であることに注意が必要である.

Sympathize は供給側から顧客をみているのに対し, Empathize は顧客側に立って, 何が困っているのかを顧客と同じように感じることを意味している.

よく言われる顧客志向は Sympathize でしかなく, デザイン思考は Empathize が重要であるとしており, 石井先生の指摘[1]ともよく一致している.

図-1 との比較では DISCOVER, DEFINE をより詳細化したものがデザイン思考のプロセスであると考えることができる.

3.3. サービスデザイン／デザイン思考で使用するツール

表-1 は「THIS IS SERVICE DESIGN THINKING. Basics[4]」に記載されていた内容も参考にしつつ, 使われるツールを一覧表にまとめたものである.

表-1　サービスデザイン／デザイン思考のためのツール

This is service design thinking	d.school	This is service design thinking	d.school	その他
DISCOVER	共感(Empathize)	○ステークホルダー(利害関係者)・マップ	インタビュー	
		サービスサファリ	○ 行動観察	エクストリームユーザ
		シャドーイング		
		○文脈的インタビュー		
		モバイル・エスノグラフィー		
		カルチュラル・プローブ(文化探索)		
		アーティ・イン・ザ・ライフ(ある一日)		
		○ペルソナ		
		○カスタマージャーニーマップ(As-Is)		
	問題定義(define)	共感マップ		
			○ エンパシーマップ	
			リフレーミング	
DEFINE	創造(ideate)	アイディア創造	○ ブレーンストーミング	
		What-if分析		
		デザインシナリオ		
		ストーリーボード		
		○カスタマージャーニーマップ(To-Be)		
DEVELOP	プロトタイプ(prototype)	デスクトップ・ウォークスルー	プロトタイプ	
		○サービス・プロトタイプ		ビジネス折り紙
		○サービスロールプレイ		
		○アジャイル開発		○ ペーパープロトタイプ
	テスト(test)		フィードバックマップ	
DELIVER	–	ストーリーテリング		
		サービスブループリント		
		顧客ライフサイクルマップ		
		○ビジネスモデル・キャンバス		
		サービス・ステージング		

○は比較的良く使われるもの.

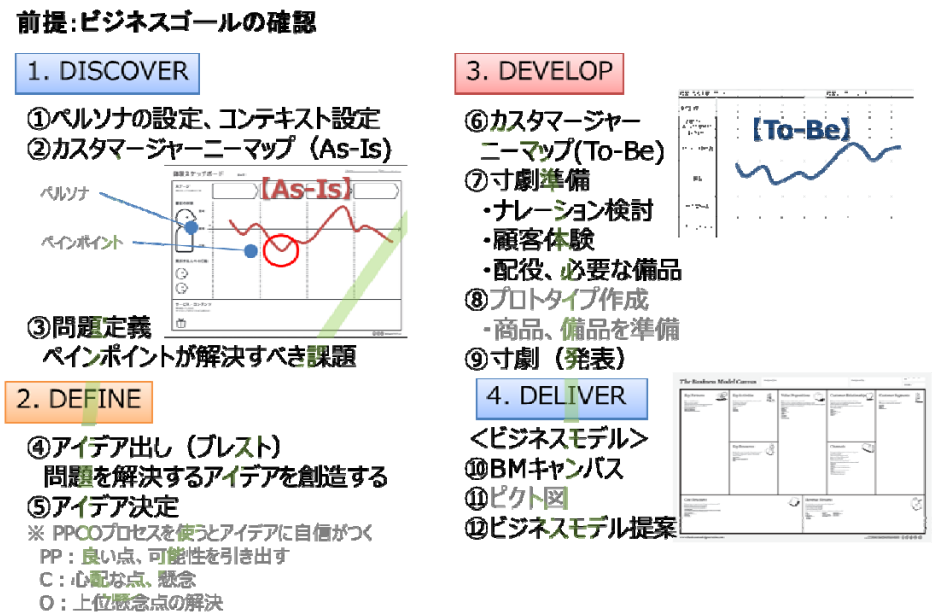


図-2 サービスデザインのプラクティス例

4. サービスデザインのプラクティス

表-1 に示したツールを用いてサービスデザインを具体的にどのように進めればよいのか，その一例をまとめたのが図-2 である．

大前提として，これらのプロセスはデザイナーとステークホルダーが一緒になってグループワークすることが重要である（前述の「参加型デザイン」）．

1. DISCOVER では，対象となる顧客（ペルソナ）の問題を認識することに専念する．この時に良く用いるのがカスタマージャーニーマップ（As-Is）であり，ペルソナが一連の活動において，どのような良い経験，悲しい経験をしているのかを書き出す．行動観察を行うと第三者でも共感しやすくなる．

図にあるように顧客が不愉快，苦痛，不便と感じているペインポイントが解決のターゲットとなる．

2. DEFINE では，ペインポイントを鳥瞰して解決のアイデアを創造する．最初は個別のペインポイントに着目したアイデア出しとなるが，それをブラッシュアップする．

3. DEVELOP は試作のステップであるが，まずは創出したアイデアを採用した場合の新しいカスタマージャーニーマップ（To-Be）を作成する．そしてその新しいシナリオをロールプレイやプロトタイピングなどの手法を用いて評価する．ペルソナの代表に参加してもらうのであるが，良い評価が得られない場合は，2. DEFINE に戻ってアイデアの再検討を行い，再び DEVELOP を実施する．

数度の Try & Error を繰り返し，良い評価が得られそうだということになれば，4. DELIVER の事業プランの作成にとりかかる．

その最初に行うのがビジネスモデルキャンバスの作成である．スポンサーや関係者に対して事業プランの説明を行う．OK が出れば，具体的な組織設計や業務設計，システム設計など構築フェーズに入ることになる．

5. 企業システムへの取り込み

以上のサービスデザインのプラクティスは，「製品へのサービスの付加による新事業の立案」において有効である．試行したところ KOMTRAX や SNCS（シスメックス・ネットワーク・コミュニケーション・システム）などのサービスを導き出せた．

このような大掛かりな取り組みだけでなく，2013/11/19-20 に英国 Cardiff で開催されたサービスデザインのカンファレンス sdnc13[5]では，金融の顧客窓口システムに適用して顧客満足度を向上させた事例や，スマートデバイス向けのアプリケーションに適用した事例など，多様な範囲で適用された成果が報告されていた．

また，翌 11/21-22 に Dublin で開催された European Design Science Symposium 2013 では，DELL から以下の様な発表が社内システム構築へのデザイン思考適用の報告がなされていた．

「Customer/human-centered design can positively affect insight and idea generation in a natural and meaningful way by helping operations team members review chronic or open-ended problems with a new lens.

Dell’s experience has been consistent with other organizations in that Design Thinking as a methodology can be applied to many problem spaces to come up with innovative solutions.」

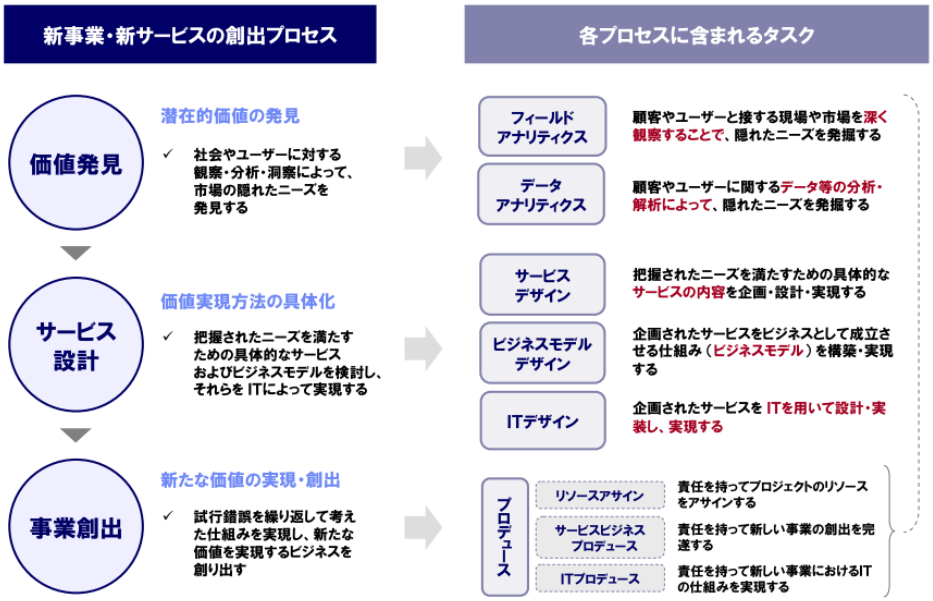


図-3　新製品・新サービスの創出プロセス

図-3 に示す「次世代高度 IT 人材」の新製品・新サービスの創出プロセスについて，経産省のレポートでは，現在の延長線上ではない新たな人材が必要となるとしていたが[6]，今回のプラクティスを整理したところ，十分に現在のエンジニアでも対応できるとの感触を得ることができた．

以上の検討を踏まえて，オンデマンド受注生産に適用することとした．

6. オンデマンド受注生産への適用

6.1. 対象ビジネスの概要

今回適用対象とする事業は BtoB 型のプリントサービスを実施していた企業が，オンデマンド型のプリント機械の導入に伴い，少量個別受注生産領域に乗り出したものである．この場合のプリントは印刷ではなく，捺染と同義で，複数の版を組み合わせることで印刷が仕上るものである．

6.2. 事業化にあたっての課題

BtoB 型企業が，BtoC ほど不特定多数ではないが，現在の営業体制ではカバーしきれない多数の顧客を対象とする場合，以下のような課題が生じる．

①営業マンによる訪問営業では回りきれない
②受注ロットが小さく，従来型の営業スタイルではまったくコストが合わない
③代金回収の確実さを高めないと，債権回収の手間もかけられない

この課題を解決するために BtoC で使われている EC システムを導入することとした．上記①，②が解決することは容易に想像できるが，③については一般の EC のシステムとして確立しているクレジット払いをあわせて導入することとした．

6.3. サービスデザインの適用

BtoB に BtoC 型 EC を導入する場合，これまでの商取引と異なって戸惑うのは提供側だけでなく，顧客側も同様である．

したがって，事業を開始するに先立って，CX のデザインから始める必要があった．加えて，EC システムを導入する必要があることから，システムの UX/UI デザインにまで展開する必要があった．

6.4. UI/UX への展開

UI の目的は UX が実現できるようにすることである．UX は CX のうち，システムが関わる部分を指す．すなわち，CX→UX→UI の順に詳細化されることとなる．適用した UX のプラクティスを図-4 に示す．[7]

サービスデザインで設定したペルソナ情報を引き継ぎ，ユーザーの視点ユースケースの代わりに，どういう操作をさせるのかをストーリーボード上に記載する．加えて，ストーリーボードに定義した各アクションについて，ユーザーにどういう経験をさせたいのかをその理由と共に付記していく．これをメンタルモデルという．次にそのストーリーボードとメンタルモデルをもとにワイヤーフレームを作成する．このワイヤーフレームで可視化されたストーリーボード上の各アクションは「機能とデータの

図－4 UX デザインのプラクティス

定義」の元となり，メンタルモデルに記載された経験が「UI 作成」の定義に大きく影響を与えるものとなる．

ただし，このプロセスは一度で完了するものではなく，ステークホルダーが参加し，数回の繰り返しを必要とする．

上記のサービスデザインならびに UX デザインのプラクティスを実施するにあたって，営業，デザイン，製造の各部門からの参加者による毎週のワークショップを開催した．

サービスデザインを終え，UX のメンタルモデル作成までは，ポストイットを使った参加型で実施しワイヤーフレームについては，草案をコンサル側で作成し，それに対し，参加者が意見の述べる方法に切り替えた．

加えて，技術的にチャレンジ要素が多い開発であったため，Develop の位置づけで PoC も実施し，その後一気に開発を進めるという方式を採用した．

結果，サービスデザイン+UI/UX で2か月，システム開発で1か月という短期間でシステムを稼働させることができた．

7. まとめ

以上のプラクティスを経て開発した EC サイトのうち UX の中心となるページを図-5 に示す[8].「直感でプロ並みの品質のパッケージを簡単かつ快適にデザインできる」という顧客経験を体現したページを構築することができた．サービスデザインを適用しなければ実現は難しかったと考えている．

参考文献

[1] 石井淳蔵，“消費者の生活に深く入り込む「経験価値マーケティング」，http://www.president.co.jp/pre/backnumber/2006/20060130/1063/，（2015.03.5 閲覧）

[2] Engine 社 HP，http://enginegroup.co.uk/approach/，（2015.03.5 閲覧）

[3] Tom Kelley, Jonathan Littman, (訳) 鈴木 主税, 秀岡 尚子,「発想する会社! ― 世界最高のデザイン・ファーム IDEO に学ぶイノベーションの技法」，早川書房, 2002.7

[4] マーク・スティックドーン，ヤコブ・シュナイダー ，長谷川敦士，武山政直，渡邉康太郎 (監修)，郷司陽子 (翻訳),「THIS IS SERVICE DESIGN THINKING. Basics - Tools - Cases － 領域横断的アプローチによるビジネスモデルの設計」，ビー・エヌ・エヌ新社, 2013

[5]http://conferences.service-design-network.org/sdnc13/programme-sdnc13/，（2015.03.5 閲覧）

[6] 経済産業省,「次世代高度次世代高度IT人材モデルキャリア開発計画事業報告書」，2013

[7] 金成哲，IT エンジニアに易しい UI/UX デザイン，http://www.slideshare.net/ksc1213/ituiux-16732374?related=1，（2015.03.5 閲覧）

[8] みんなのパッケージ，https://www.minpake.com/，（2015.03.5 閲覧）


みんなのパッケージ
オリジナルパッケージを作りましょう。
背景を決める
スタンプを選択
画像をアップ
文字を入力
裏面へ
TOPに戻る
校正確定
タイプA
タイプB
タイプC
その他
光沢
ORGANIC COFFEE
No. 3
coffee roasters
ORIGINAL
←ひとつ前に戻る
ひとつ前に進む→
保存する
復元する
頭シール /10~15mm
切り取り線
極まれにインキ変色の恐れあり
チャック
サイドシール
Original Coffee
※シール部分
商品名(自由にお決め頂けます)
※必須入力項目 16文字以内
拡大縮小
比率:
%
変化させる
デフォルト
回転角度
度
変化させる
デフォルト
重ね順
△一つ上へ
▽一つ下へ
▲最前面へ
▼最背面へ
位置揃え
左揃え
中央揃え
右揃え
ガイドラインのオンオフ
オン
オフ
パーツを削除
全てリセット

図-5 開発した UX

ソフトウェアアーキテクチャの授業での取り上げ方について

小林　洋
東海大学
koba @ tokai.ac.jp

要旨

最近，ソフトウェア開発において重要性の認識が高まって来ているソフトウェアアーキテクチャを大学の授業で教えようとする，一つの取り組みについて述べる．まず，開発経験が無いといってもいい学生に，ソフトウェアアーキテクチャについて講義しようとする場合の問題点について述べる．次に，講義や実習の授業でソフトウェアアーキテクチャを取り上げようとした場合の問題点と課題について，授業での今までの試みを踏まえて述べる．なお，ソフトウェア開発の実習においては，ゲーミフィケーションの導入は，学生の興味をひくという点から効果的であった．

1. はじめに

ソフトウェア開発におけるアーキテクチャの重要性が認識されるようになって来ているため，情報系の学科では，何らかの形でそれを意識した授業を行っているものと思われる．しかしながら，通常，発表されるものは，どうしても，そのコンテキストも含め優れた事例ばかりになってしまい，一般的な状況が正しく伝わっていない恐れがあるように思われる．本稿では，一事例に過ぎないかもしれないが，比較的一般的と考えられる状況下で，ソフトウェアアーキテクチャを教えようとする取り組みを踏まえた上で，問題点や課題について述べる．問題点としては，まず，ソフトウェアアーキテクチャという用語には多義性があるため[1]，列挙して解説するだけでは，理系の学生の多くは興味を示さない上，混乱を招く恐れがあることが上げられる．学生には実際の開発経験など無いといってもいいため，単なる用語の羅列や，ボックス型の図形に単語が書かれただけのものと受け止められてしまうところもある．次に，大学と言う特性上，学問的にも扱いたいところだが，定量的評価方法が未だ確立されていないために，これが難しいことが上げられる．更に，単なる概念的な知識としてだけではなく，理解を深めてもらうためには実習が効果的であるため，最近，開発の実習において効果的な方法として注目を集めているプロジェクトベーストラーニング(Project-Based Learning :PBL)で行い，この中にソフトウェアアーキテクチャを取り入れることが考えられるが，PBL を実現させるためには，現状，予算や人員の確保等いくつかの課題がある．また，学部生の段階では，多くの学生は PBL よりも個人レベルでのソフトウェア作成力の向上の方が優先事項のように見受けられる．その場合，個人レベルの実習の問題となると，どうしても極めて小規模な問題にならざるを得ないが，その中でアーキテクチャをどのように取り入れるかは継続した課題である．なお，ソフトウェア開発の実習では，ゲームの要素を取り入れ，段階的にクリアして行けるようにすると，興味を持って取り組んでくれることが解った．この試みの成果としては，特に三層モデルでの開発により，ソフトウェア開発では動くものをただ作れば良いという訳ではなく．アーキテクチャを考慮する必要のある事を，認識してもらうことが出来たことではないかと考えている．

ここで取り上げた科目は，「情報システム開発・同演習」という3年生対象の週2コマ，概ね1コマずつ講義と実習を行う授業であり，単なる実習だけの授業ではない[2]．受講者は，前提条件としてオブジェクト指向と Java 及び SQL は履修済みであることとしている（但し，拘束力はない）．受講者数は，選択科目ということもあり，年度によって30名程度から80名程度と大きく変動する．授業は教員1名と大学院生の教育補助学生（Teaching Assistant: TA）2名で行っている．本稿は，主にこの授業での試みとその結果を基にした所見である．以下，2.では，ソフトウェアアーキテクチャの問題点について，3.では授業の取り組みとその問題点と課題について述べる．

[1] 観点の多様性と捉えても良いが，多くの学生から見れば実質同じ事かもしれないと思われる．

[2] 実習だけの授業では2コマで2単位，同演習とついている科目は，講義と実習からなり，2コマで4単位となる．TA の人数は，大学内の規定で定められている．なお，カリキュラム策定者と授業実施者は同じとは限らない．

2. ソフトウェアアーキテクチャの問題点

2.1. 用語の多義性

ソフトウェアアーキテクチャは，ソフトウェアの開発規模が大きくなり，ソフトウェア危機が提唱されるようになった 1960 年代後半頃から研究され始めたようであり，その定義は対象とする範囲も含め研究者や文献により様々である[3] [1]. これらの定義を列挙したものが，例えば SEI の Software Architecture の Web ページ[2]に記載されている. 狭義にはソフトウェアアーキテクチャは，プログラムの静的構造のみを指す場合もあるが，一般的には動的構造（振る舞い）も併せて指す. これらの表現方法としては図式が良く用いられ，前者は機能分割図や UML のクラス図等が，後者は UML のシーケンス図等が良く用いられる. これらに加え，表形式が補助的に使われ，厳密性を要求される場合には数学的表現が用いられる場合もある. これらの構造で，ソフトウェア開発者によって良く使われるものをパターン化したものに，細かい粒度では GOF のデザインパターン[3]等がある. また，より粗い粒度のミドルウェア的な構造としては MVC(Model-View-Controller) モデルや三層(Tree Tier [Layer])モデルなどがあり，ブッシュマンのソフトウェアアーキテクチャ[4]等に示されている. 他方，業務アプリケーションのパターンに対応したものとしては，ピーター・コードのビジネスオブジェクトモデリング[5]等に示されている.

ソフトウェアアーキテクチャについて定義した IEEE1471[6]（ANSI/IEEE 1471-2000：現在では ISO/IEC/IEEE42010:2011[7]）では，「ソフトウェアアーキテクチャとはコンポーネント，コンポーネント間およびコンポーネントと環境との関係，並びにその設計や発展性を導く原理によって具現化されるシステムの基本構成」と記されており，更に，アーキテクチャでは，利害関係者の関心である，性能，信頼性，セキュリティ，分配，発展性等を考慮する必要があるとしている. IEE1741 での定義の後半の部分については広義に解釈すると，開発の方法論，開発環境，技術標準，オントロジ（語彙体系）や原則，ガイドラインまで含むことになり，TOGAF（The Open Group Architecture Framework）[8-9]などではそのような観点で記述されている. また，情報システムの構築を策定しようとする場合，利害関係者（ステークホルダ）の立場により，情報システムを複数の観点（ビュー）から捉えることになる[10]. 対象とする業務（事業）の構造ということを名称にも明示したエンタープライズアーキテクチャ（EA）[11-13]では，図1に示すように，ビジネスアーキテクチャ，データアーキテクチャ，アプリケーションアーキテクチャ，及びテクノロジーアーキテクチャの4つで構造を捉えていて，対象業務の分析やシステム方式設計等を含んだアーキテクチャとなっている. 表記方法については，ビジネスアーキテクチャは，組織構造図と業務（ワーク）フロー（UML で表すなら，アクティビティ図やユースケース記述），データアーキテクチャは ER 図や DFD[4] ，アプリケーションアーキテクチャはモジュール構造図と振る舞い図（UML では，クラス図，コミュニケーション図，シーケンス図），テクノロジーアーキテクチャは構成図（UML では，配置図）などが使われている. なお，最近話題になった自動車向けの機能安全規格である ISO26262[14]では，ソフトウェアアーキテクチャについては，設計における手法として表記法，エラー検出と処理，および検証が示されており，安全性のための開発手法やシステムに必要な安全機構（機能）という観点から捉えている. このように，ソフトウェアアーキテクチャの定義は，狭義のものから広義のものまで様々なものがあり，コンテキストによって使い分けられている.

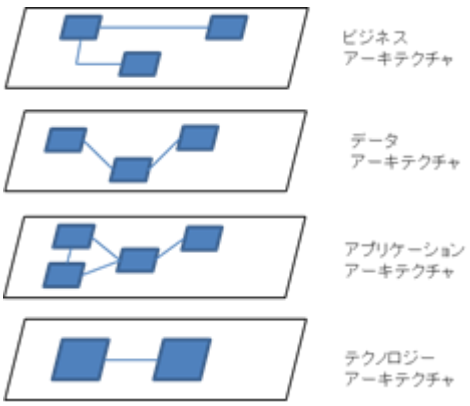


図1 エンタープライズアーキテクチャ

[3]用語の定義というものは時代と共にだんだん広義なものになって行く傾向があるが，それと共に曖昧性が増す. そのため，場合によってはやがて別の用語が用いられるようになる.

[4] DFD は，データアーキテクチャを表すとされる場合と，アプリケーションアーキテクチャを表すとされる場合があり，これは観点の違いと考えられる.

2.2. 評価方法

ソフトウェアアーキテクチャの評価の必要性については，連邦エンタープライズアーキテクチャ FEA（Federal Enterprise Architecture ）でも，FEA consolidated Reference Model Document Version2.3(2007)[12]で，Performance reference Model(PRM)に記載されている．但し，この文書に示されている評価尺度（Measurement indicator）については，カテゴリとグループで分類しているのみであり，具体的な評価尺度は各機関で作成する必要があるとされている．従来，ソフトウェアアーキテクチャの評価手法としては，いくつかのものが提案されており[15-22]，SAAM（Software Architecture Analysis Method）[21]や ATAM（Architecture Tradeoff Analysis Method）[22]が比較的良く知られている．SAAM では，アーキテクチャの修正容易性，拡張性等の開発の際の特性を，変更のシナリオを用意し，変更させるモジュール数，修正コスト等をスコアリングにより評価している．SAAM を拡張した ATAM では，アーキテクチャが性能や信頼性等の品質特性に関する要求を満たし得るかを，品質特性（性能，稼働率，修正容易性等）を列挙したユーティリティツリーに基づき，品質特性に関わるシナリオを用意し，要求を満たすかを評価している．いずれも，アーキテクチャを，シナリオに基づくシステムの振る舞いの妥当性によってスコアリングにより評価するという方法である．しかしながら，これらの手法は，人的・時間的コストの問題から，開発現場において用いる場合においては，簡略化した手法が用いられることが多いようである[23].

3. 授業の取り組みとその問題点と課題

3.1. 授業の主な内容

授業の主な内容は，以下の通りで，（1）から（6）は，（7）を行うための準備教育である．

（1）ソフトウェアアーキテクチャと UML による設計
（2）部品化を考慮したプログラミング
（3）GUI プログラミング
（4）Java と SQL の連接
（5）GUI から DB までの連接
（6）Web プログラミング
（7）ロバストネス分析手法による総合演習

3.2. 講義での問題点と課題

ソフトウェアアーキテクチャを広義にとらえると，情報システム開発に関連する事柄の，あらゆる構造的なものとなってしまう．このような内容は，極めて小規模の練習問題程度のプログラムしか開発経験の無い学生にとっては，実感の伴わない概念だけの話になってしまう．ソフトウェア開発の分野でしばしば用いられる前述の図1のような箱型のモデル記述は，一時限りの暗記事項で終わってしまう可能性がある．また，ソフトウェア開発に関わるアーキテクチャの定義はいろいろあると言って，列挙して示したりすると学生の多くは混乱するだけのようである．初学者にとっては，まず，講義では定義のごく数種類のみを示し，多義性については，最後に軽く触れる程度に留めるのが良いようである．しかしながら，例えば MVC モデルに限定した話でも，図２に示すように，Smalltalk 流の MVC と Web アプリケーションでの MVC2 では振る舞いも異なるし，C や M の範囲ついては，対象や研究者によって解釈の違いが見られる．MVC2 の捉え方では，C は単純な振り分け処理のみで，M はアプリケーションのみ，或いは永続データへのアクセスの連接も加える場合があり，更には永続データも含める場合がある．MVC モデルと共に図3に示す三層モデル（プレゼンテーション層：P，ビジネスロジック層：B，データ層：D）を教えようとする場合，この二つについては，単に観点の違いであると説明する方法があるが，それで納得してもらえるかは疑問である．そこで，図4のように，V+C＝P，M=B+D（又は M=B）であると説明する方法も考えられるが，このように限定したケースであっても，学生に理解してもらうには，いかにしたら良いかは，課題であろう．

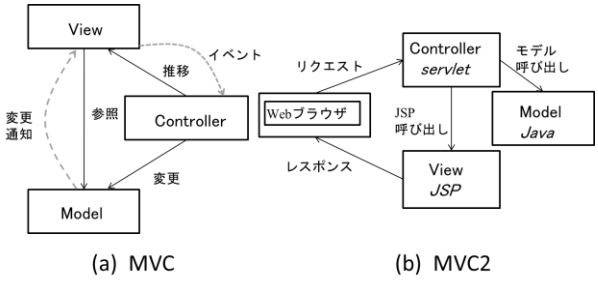


図2 MVC モデル

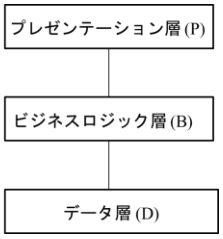


図3 三層モデル

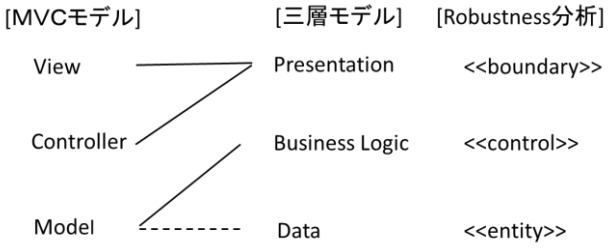


図4 MVC と三層（PBD）モデルとの対応？

3.3. 実習での問題点と課題

(1) 開発工程に沿った実習

a. PBL と個人のスキル

ソフトウェアアーキテクチャを学生により深く理解してもらうには，具体的なソフトウェア開発の問題により，開発工程に沿って要求分析からプログラミング[5]までの実習の中で理解してもらうのは効果的であると考えられる．このような実習の方法として，最近，数人（例えば5～6人程度）の学生でグループを組んである程度の規模のソフトウェアを開発するというプロジェクトベーストラーニング（Project-Based Learning: PBL）[26-28]が注目を集めている．PBL の実施例を見ると，グループのリーダの力が重要となるということで，企業等から招へいした実開発経験者を各グループに一人ずつ配置し，プロジェクトの全般管理の他，業務の適切な割り当てや，ソフトウェアの不具合についての適切なサポートを行ってもらい，更に，各グループに大学院生の TA も配置しているケースが多く見受けられ，それが理想なのであろうと思われる．しかしながら，現実には，大学によっては，まず予算や大学院生の数の問題により，そのような十分なサポート体制が取れない場合がある．また，学部生の多くは3年生であっても，未だソフトウェア開発についての個人の知識・技能の向上を行う段階にあり，プロジェクトを行える段階ではないように見受けられる．現状では，大学により状況は異なるかもしれないが，学部生の内は，各人のスキルの向上を図るために，個人ごとに設計からプログラム作成まで行う方が適しているように思われる． この場合，現在の大学では，基本的には「ほとんどの者がほぼ達成できるような問題」を出題することが求められているようなので，授業時間という制約もあり，トイレベルの極めて小規模な問題にせざるを得ない．但し，学生数だけ異なる問題を用意するのは，現実的に不可能であるので，共通又は数種類の問題で実習を行わざるを得ない．いずれにせよ，従来型の個人ベースの実習においてもであるが，特に，PBL の実施にあたっては，予算と人員の確保が第一の課題である．

b. 開発対象とする問題の選定

学生を対象としたソフトウェア開発の実習においては，上流工程の要求分析やシステム設計においては，対象業務についての知識が必要となる．すると，教育の時間的制約からどうしても履修登録システムや図書館システムなど，学生になじみの深いもので行わざるを得なくなる．但し，ここで問題になるのは，このようなシステムに限らず，今の時代，日常的な業務系システムにおいてはほとんど既存のものが存在するため，システム設計においては，学生は既存のシステムの処理手順やそれに伴う画面の流れの影響を受けてしまい，単にそれらの一部の再現になってしまう可能性が大きいという問題がある．履修登録システムや図書館システムも含め適切な対象業務を選び，適切な規模の問題を作成することは，課題である．

(2) アーキテクチャを意識した開発の実習方法

学生に，小規模の問題にせよ，上流工程からプログラミングまで通して修得してもらい，更にその中で三層モデルの一種についても理解してもらうために，次のような手順の実習を実施している．但し，受講者は，前提としてオブジェクト指向と Java 及び SQL は履修済みであることとしているが拘束力はないので，最初の数回の授業で，確認のための実習を行っている．また，実習の効率化のために，GUI や JDBC 等のサンプルプログラムをファイルの形で与え，これを活用してもらっている．なお，コーディング規約については，学生がうんざりしない程度の簡単なことのみに留めている．

まず，図5に示すように，要求仕様に相当する文書（あまり長くないもの）を与え，段階的に成果物を作成して行けるように，ロバストネス分析手法[24-25]を簡略化した次のような手順で実習を試みている[6]．最初は，①UML のユースケース（図と記述）を書いてもらう[7]．（時間的余裕があれば，更に業務フローを表すためのアクティビティ図

[5]授業時間という制約を考えると，テスト工程まで十分に行うのは困難なためテスト工程は簡略化せざるを得ない．また，学生の多くが興味を抱くのは，作成の部分である．

[6] ロバストネス分析手法を採用した理由は，画面，ロジック，データの分離を教えるのに効果的と思われたためである．

[7]実務においては，ユースケースをいきなり書くようなことは，あまり有り得ない事かもしれないが．

及び粗いシーケンス図を書いてもらう.)次に,②画面のデザインのスケッチと画面フロー図(画面遷移図)を書いてもらう.更に,③文献[5]等に示されている概念モデル(ドメインモデル)を参考に必要なクラスを考えてもらい,ロバストネス分析手法での boundary(View), control(ロバストネス分析では,アプリケーションロジックを指す), entity(Model:ロバストネス分析では,永続的データを指す)に分けてクラス図を作成してもらう.その結果を,④画面デザインと画面フロー図にフィードバックし,必要な修正を行ってもらう.

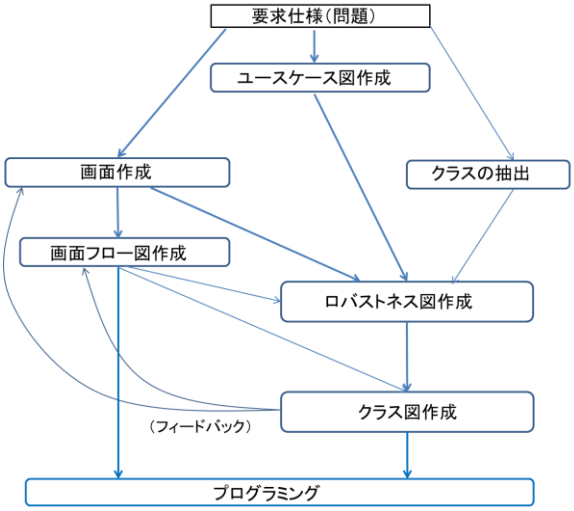


図5 ロバストネス分析の簡略手法

すると,画面デザインや画面フロー図については,インターネットやスマートフォンが普及した現在,Web 画面や GUI 画面についてのイメージが湧きやすいのか,比較的容易に考えられるようである.

クラス図の作成については,参考となるクラス図のカタログがビジネスオブジェクトパターンとして文献[5]等に示されているので,これらを参考にして考えてもらうのが実際的なように思われる.初学者には,いきなり,「販売」や「注文」がクラスになるということなどは,まず思いつかない.

最後に,⑤実装を行ってもらう.実装は,boundary, control, entity に分けて記述したクラス図と画面デザイン及び画面フロー図を参考に Java と SQL(本授業では SQL Server)で行っている.ところで,永続データ部分の実装については,リレーショナルデータベースの使用が一般的であろうが,この際,クラス図からテーブルへの変換が必要となる.この場合,クラス図が基本的な記述のみの単純なものであれば,正規化がうまく行われるかは別として,テーブルへの変換は比較的容易に行える.

もちろん,このような小規模の仕様の場合には,実際の開発では,三層への分離など不要かもしれないが,学習としては行っても良いと思われる.

以上については,現在,学生の学習状況を見ながら試行錯誤して行っていることである.学生の実習の場合,学生の習熟度や授業の時間,更には予算の問題もあり,企業で行われているような実際的な手法の一つを,そのまま導入すれば良いという訳ではなく,学生の状況に合わせた修正が必要と思われる[8].

(3) 設計書とプログラムのアーキテクチャのギャップ

プログラムの開発経験の乏しい学生には,設計図を書くだけでは,その設計図が実際に動くプログラムに繋がるという実感が湧きにくく,プログラム作成までを経験するのが望ましいと思われる.ところで,学生が上流工程からプログラミングまでを通して行おうとすると,設計図は何とか出来ても,プログラミングの段階になると,設計図のとおりではプログラミング困難ということで,設計図を棚上げして,プログラミング容易なものを作成してしまうことが起こり得る.そして,完成したプログラムと設計図と突き合わせてみると,図6のように,ソフトウェアアーキテクチャ的にもかなり違ったものが出来ているということが起こる.しかし,この事もまた貴重な体験なのであろうから,とにかく,一度,上流工程からプログラミングまで通して開発してみること自体が重要なのだとも思われる.但し,学生の多くは,問題には唯一の解答があり,それに向けて一直線に開発を進めたいと考える傾向があり,試行錯誤を嫌がる傾向にある.

図6 仕様とプログラムの不整合

[8] 企業から大学に移った教員が,当初,「このような授業の内容は現場ではやっていない.」と言うのを時々耳にするが.

また，最近では，画面設計自体は，GUI ツールの発達により，見栄えの良いものが手軽に出来る．しかし，そのまま，プログラムの設計や作成を行ってもらおうとすると，作成できなかったり，又は，できたとしても，画面のプログラムにロジックやデータベースとのアクセスまで張り付いたものになり易い．GUI の作成だけが先行してしまうのを，いかにして解消するかは課題と言えよう．

(4) デザインパターンの初学者にとっての解り難さ

最近では書籍の付録やネットに，プログラムのサンプルが多数掲載されている．プログラムの問題を考える前に，サンプルを捜すというアプローチをとる学生も少なくない．但し，これらのプログラムは，お手本として載せているせいか，ある程度の熟練者が作ったものが多く，巧みに部品化され，デザインパターンが用いられているものも多い．ところが，初学者にとっては，これがプログラムの理解の妨げになってしまうことがある．例えば GOF の Abstract Factory パターンなど，初学者にとっては，面倒なだけのように思えるし，Composite パターンなどもプログラムを見ると，戸惑うようである．一般的に，熟練者の作った筋の良いプログラムは，初学者にとって必ずしも解り易いプログラムではない．もちろん，熟練者が巧みに用いているデザインパターンを学習することによって，プログラミングの技能向上を図るというメリットはあるだろうが，多くの学生にとっては，プログラミングが難しく思えてしまい，苦手意識を持つことになるというデメリットも大きいように思える．デザインパターンをどのように扱って行くかは課題であると考えられる．

なお，初学者にとっては，パターンを用いた場合のメリットも解り難いようであるが，一つの解決策として，結果の可視化が考えられる．例えば Singleton パターンの効果については， 図7のように，画面遷移において，ボタンを複数回クリックすると画面が複数個作成されてしまうのを Singleton パターンで防止することが出来ることなどは，学生にとっては直観的で解り易いようである．

図7 シングルトンパターンの振る舞いの可視化

(5) アーキテクチャの評価方法とその実習方法

大学では，学生に対象を工学的に評価する手法，できれば定量的評価する手法を取り上げて教えたいところである．ソフトウェア開発における，広い意味でのアーキテクチャの評価手法を上げるならば，現在は ATAM が良く知られており，これから派生したような研究も多い[15-22]．しかし，これらの手法については，そもそも，企業等が実開発で用いる場合にも課題があるようである．まず，アーキテクチャの評価は開発前に行いたいが，詳細な設計もされていない段階で，妥当な評価をいかにして行うかということが上げられている．次に，ATAM 等を実施するには，マンパワー（コスト）がかかるので，それをいかに低く抑えるかという点であり，実開発においては，より簡略化した手法を用いているのが実情のようである[23]．このような状況にある評価手法を授業で取り上げるのが適切かどうかも疑問のあるところだが，授業で取り上げた場合，学生に理解してもらうには，適当な規模の具体的問題の実習により行う必要があると考えられ，問題の作成と併せて，良い解答例の作成も課題となる．

(6) ゲーミフィケーション

実習教育においては，ゲームになじんでいる学生が多いことから，ゲームのステージ達成という要素を取り入れた，インクリメンタルな開発という方法が効果的なようである（図8参照）．

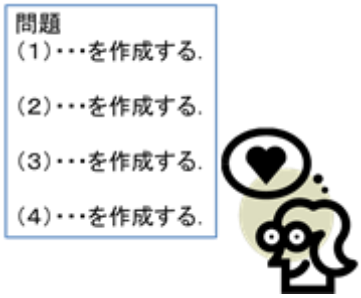


図8 ゲーミフィケーション（小問（ステージ）への分割）

本稿で対象とした授業の場合には，3.1.で示した授業の内容の実習において，各問題では可能な限り小問に分割することを試みた．また，各問題自体が総合演習で，そのままではないにせよ，活用できるような問題となるようにした．実習においては，問題を分割し，段階的に作って動くのを確認しながら，徐々に拡張して行くように問題を作っておくのは効果的であった．ゲーム慣れしている学生にとっては，ソフトウェア開発において，ゲームのス

テージのようなものを設けて，適宜与えられたヒントを参考にしながら，1段階ずつクリアして行くという方法，つまりプログラム開発のゲーム化（ゲーミフィケーション）[29]という方法が，興味を持ってもらえ有効であった．但し，このように小問に分割する方法では，どうしても各小問については，解釈が限定されるような問題になってしまう．本来は，上流工程で仕様が曖昧な部分は各人の判断に任せ，出来るだけ自由に作成してもらいたいところだが，小問の連続にすると自由度が失われ創造性が妨げられてしまう恐れがあるという問題がある．しかし，プログラム開発での不具合に教員やTAが対処可能で，ほとんどの学生がほぼ達成できるようにするためには，やむを得ないと考えている．開発の実習のゲーミフィケーションについては，他にもいろいろな手法の導入が考えられるだろうが，これは今後の課題と考えている．

4. おわりに

本稿では，ソフトウェアアーキテクチャを，大学で教えようとする場合の問題点と課題を，現在の授業の状況を踏まえて述べた．

問題点としては，そもそも，ソフトウェアアーキテクチャという用語があまりにも多様に使われていることと，定量的評価法が確立していないことが上げられる．

授業を講義として行う上での課題としては，実開発の経験の無い学生に，対象や研究者によって解釈の異なる場合のある抽象化・モデル化した内容をどのように理解してもらうかということが上げられる．

実習として行う上での課題としては，まず，実習の一般的な課題として，予算と実習補助者をいかに確保するかということが上げられ，特に，PBL においては，切実な問題となる．なお，PBL については，現状，一般的な状況においては，学部生の段階では，個人の知識・技能の向上を行う段階にあるように思われるため，卒業研究や大学院あたりでの実施の方が適切なように思われる．

実習を，アーキテクチャを考慮したものにしようとする場合，実習では，問題を極めて小さなものにせざるを得ないため，三層モデルの場合，三層モデルを適用した小さな適当な問題の作成ということが，まず課題として上げられる．また，作成するシステムは，時間的制約やサポート体制のことも考えると，現状，学生にとって仕様が理解容易な，限られた種類のものになってしまう傾向があり，その意味でも適切な問題の作成というのは課題である．

開発においては，最近では，GUI の作成が容易になったために，GUI の作成だけが先行してしまう傾向があり，これをいかに解消するかも課題であると考えられる．また，学生が書籍やネットのプログラムのサンプルを参考にするのは止むを得ないと思われるが，熟練者が作成した巧みに部品化したデザインパターンを使ったプログラムは，学生には理解しにくい事が多い．これにより，プログラミングに対して苦手意識を持ってしまう学生も多いようなので，デザインパターンを実習でどのように取り扱っていくかは課題と考えている．

更に，大学の授業においては，教えたことは理解度をテストし評価を行う必要があるため，実習において，どのようなテスト問題で理解度を評価するのが妥当であるかということも，大きな課題である．

以上のように，ソフトウェアアーキテクチャを取り上げようとした授業の試みは，今の所，課題が多く残されているが，成果として上げられる最大のことは，ソフトウェアの開発においては，ただ動くものを作成すれば良いというだけではなく，アーキテクチャを考慮する必要があるということを，学生に認識してもらい，多少なりとも納得してもらう事ができたことであろうと考えている．なお，実習で用いたロバストネス分析手法は，三層の分離を強調するという意味では，有用であると思われる．

また，実習においては，問題を小問に分け，段階的にクリアして行くようにするという，一種のゲーミフィケーションの手法の導入は，実習の促進に効果的であった．ゲーミフィケーションについては，他の手法の導入の検討が今後の課題として残されている．

授業の工夫とその効果については，授業アンケートや実習問題の提出状況等を基に定量的評価の形で示した方が良いのであろうが，公表できるような形にはなっていないので，本稿では，授業の状況を基にした所見のみに留めた．社会的な影響も考慮した上で，外部へ公開できる形での客観的データの収集と分析評価については，今後の課題と考えている． 但し，最終的には，学生がソフトウェア開発に悪いイメージを持たず，ソフトウェア開発業界を避けないようにすることが，大きな課題であると考えている．

参考文献

［1］ 岸知二，野田夏子，深澤良彰：ソフトウェアアーキテクチャ，共立出版，2005.
［2］ SEI Software Architecture:

http://www.sei.cmu.edu/architecture/start/glossary/community.cfm
[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides: Design Patterns, Addison-Wesley,1995. [邦訳]本位田真一, 吉田和樹監訳, オブジェクト指向における再利用のためのデザインパターン,ソフトバンク, 1995.
[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal: Pattern-Oriented Software Architecure: A System of Patterns, John Wiley & Sons, 1996. [邦訳]金澤典子他訳,ソフトウェアアーキテクチャ:ソフトウェア開発のためのパターン体系, 近代科学社, 2000.
[5] P. Coad, D. North, and M. Mayfield: Object Models: Strategies, Patterns, and Applications 2nd Edition, Yourdon Press, 1997. [邦訳]依田光江訳, 戦略とパターンによるビジネスオブジェクトモデリング, ピアソン, 1999.
[6] ANSI/IEEE1471-2000:http://standards.ieee.org/findstds/standard/1471-2000.html/.
[7] ISO/IEC/IEEE42010:2011: http://www.iso-architecture.org/42010/.
[8] TOGAF: http://www.opengroup.org/togaf/.
[9] オープングループジャパン: http://www.opengroup.or.jp/togaf.html/.
[10] N.Rozanski, and E. Woods: Software Systems Architecture: 2nd Edition,2012.[邦訳]榊原彰監訳, 牧野祐子訳, ソフトウェアシステムアーキテクチャ構築の原理:第2版, SB クリエイティブ, 2014.
[11] FEA: A Practical Guide to Federal Enterprise Architecture Version1.0 :http://www.enterprise-architecture.info/Images/Documents/Federal Enterprise Architecture Guide v1a.pdf,2001.
[12] FEA consolidated Reference Model Document Version2.3:http://www.whitehouse.gov/sites/default/files/omb/assets/fea_docs/FEA_CRM_v23_Final_Oct_2007_Revised.pdf ,2007.
[13] 萩原正義:アーキテクトの審美眼, 翔泳社, 2009.
[14] ISO26262: http://www.iso.org/iso/catalogue_detail?csnumber=43464.
[15] U. v. Heesch, V-P. Eloranta, P. Avgeriou, K. Koskimies, and N. Harrison: Decision-centric Architecture Reviews, IEEE Software,Vol.30, No.1, pp.69-76, 2014.
[16] L. Bass, and R. L. Nord: Understanding the Context of Architecture Evaluation Methods, Proc. 2012 Joint Working Conference on Software Architecture & 6th Europian Conference on Software Architecture, pp.277-281, 2012.
[17] L. Dobrica, and E. Niemela: A Survey on Software Architecture Analysis Methods, IEEE Trans. Software Engineering, Vol. 28, No.7, pp.638-653,2002.
[18] J. Knodel, M. Lindvall, D. Muthig, and M. Naab: Static Evaluation of Software Architectures, Proc. Conference on Software Maintanance and Reengineering(CSMR'06),pp.284-294, 2006.
[19] N. Harrison, and P. Avgeriou: Pattern-based Architecture Reviews, IEEE Software, Vol.27, No. 6, pp.66-71, 2011.
[20] P. bengtsson, N. Lassing, J. Bosch, and H. v. Vliet: Architecture-level modifiability analysis (ALMA), The Journal of Systems and Software, No. 69,pp.129-147, 2004.
[21] R. Kazman, L. Bass, M. Webb, and G. Abowd; SAAM: A Method for Analyzing the Properties of Software Architectures, Proc. 16th ICSE, pp.81-90,1994.
[22] R. Kazman, M. Barbacci, M. Klein, and S.J. Carriere: Experience with Performing Architecture Tradeoff Analysis, Proc. 21st ICSE, pp.54-63, 1999.
[23] T. エンゲルバーク著, 長谷川裕一, 土岐孝平訳: 間違いだらけのソフトウェア・アーキテクチャ:非機能要件の開発と評価, 技術評論社, 2010.
[24] D.Rosenberg, and K.Scott: Use Case Driven Object Modeling with UML: A Practical Approach, Addison-Wesley, 1999.[邦訳] 長澤嘉秀, 今野睦訳:ユースケース入門ーユーザマニュアルからプログラムを作る, ピアソン, 2001.
[25] C. T. Arrington: Enterprise Java with UML, Wiley, 2002.[邦訳]平川章他訳:UML によるエンタープライズ Java 開発, 翔泳社, 2002.
[26] B.J.S. Barron, D. L. Schwartz, N.J. Vye, A. Moore, A. Petrosino, L. Zech, and J.D. Bransford: Doing with understanding: Lessons from research on problem- and project-based learning, The Journal of the Learning Sciences, Vol.7, pp.271-311, 1998.
[27] 松澤芳昭, 大岩元:産学共同の Project-based Learning によるソフトウェア技術者教育の試みと成果, 情報処理学会論文誌, Vol.48, No.8,pp.2767-2780, 2006.
[28] 鶴保征城, 駒谷昇一:ずっと受けたかったソフトウェアエンジニアリングの授業1, 2[増補改訂版], 翔泳社, 2011.
[29] 井上明人:ゲーミフィケーション―"ゲーム"がビジネスを変える, NHK 出版, 2012.

要件開発プロセスへのPRePモデルの適用

田中 康† 後神 義規†† 光井 邦雄†††
†有限会社ケイプラス・ソリューションズ †東京工業大学 ††株式会社日立製作所 †††クラリオン株式会社
†ytanaka@kplus-solutions.com †ytanaka@cs.titech.ac.jp ††yoshinori.goko.yv@hitachi.com †††Kunio_Mitsui@clarion.co.jp

要旨

PReP（プレップ）モデルは，ソフトウェア開発プロセス改善のために考案した成果物観点によるプロセスのモデル化方法である[1]．業務システム開発の現場で認識されていた従来方法による業務プロセスモデリングの難しさと，業務を超上流から改善する際の有用性の問題に関して，成果物観点のプロセスモデルが有効であるとの仮説に基づき，PRePモデルを業務システム開発の超上流工程に適用した．適用結果を，プロセス改善のためのプロセスモデル要件に沿って評価した．また，本手法を適用する際に重要となる観点に関して，適用経験から報告する．

1. はじめに

大規模なシステムをサービスの集まりとして構築するSOA（サービス指向アーキテクチャ）の普及に伴って，業務プロセスのモデリング方法であるBPMN，BPEL，WfMC，UMLなどの標準が整備されてきた．特に，BPMNの標準化がOMGに移ってUMLとの統合が図られたことによって，BPMNでモデリングをしBPELでプロセスを記述してサービスを動作させ連携させるといった上流工程の流れが整備されてきている．

1.1. 現在の業務プロセスの記述の問題

その一方で，業務プロセスを記述する際に，「適切な粒度でのプロセスのモデル化が難しい」，「抽象度のレベルが混在してしまう」といった問題や，「どこまで描けばすべて描いたことになるのかわからない」といった網羅性の問題など，モデル化自体の困難さに関する声が現場から上がっていた．

さらに，「業務プロセスを記述しているつもりがシステムの仕様記述になってしまい、業務自体の見直しが難しかった」，「現状の業務プロセスを描いてはみたものの，どこをどのように改善すれば良いかが見えてこない」といった業務改善における有効性の問題もあがっていた．

1.2. 超上流工程での有用性の問題

IPA（独立行政法人情報処理推進機構）は，ITシステムの要件を，経営と業務の視点を含めて定義する過程を「超上流」と呼び，その重要性を訴えている[2]．日立製作所では，顧客経験価値の視点から業務を見直すことによって，この超上流工程からシステム要求を考える取り組みを行っている[3]．しかし，現状の業務プロセスのモデル化方法では，前項で述べた問題に加え，経験価値視点から業務を見直すために，対象業務の本質的な意味や理由を理解することが難しいという声があった．

ビジネスプロセスモデリングの例を図1に示す．図1は，旅行予約プロセスをモデル化したものである[4]．プロセスモデルの各ノードは「アクティビティ」または「タスク」と呼ばれている．

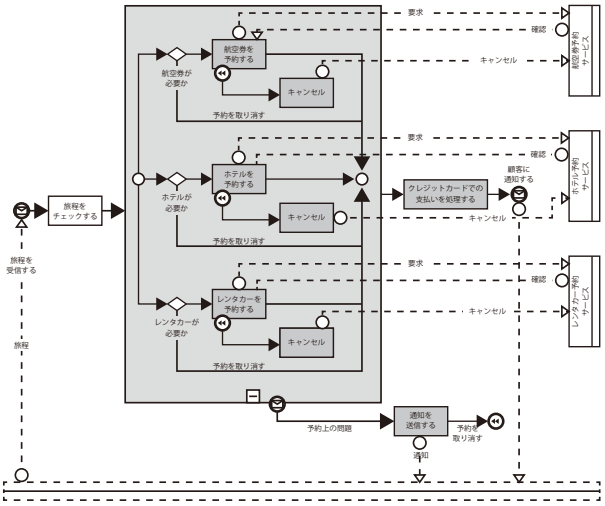


図1. 旅行予約プロセスの例

このプロセスの例では，旅行の予約業務は，取得した顧客の旅程をチェックし，航空券を予約し，ホテルを予約し，レンタカーを予約するといったタスク（「〜する」と表現される）の流れとなって定義されている．一見すると順当なプロセスであると思われる．このように，現在の業務プロセスモデルでは，「何をするのか」と，それに対応するサービスを定義することはできる．

しかし，我々が経験価値の視点から考えたいことは，何故，「旅程をチェックする」，「航空券を予約する」，「ホテルを予約する」というタスクがプロセスモデルのノードとして表出されたのか，そして，されるべきであったのか，という「何故」という問いである．

旅行者の目的は，航空券を予約することではない．ホテルを予約したいから旅行に行くわけではない（そのよ

うな旅行者もいるかもしれないが）．多くの旅行者は，安心して快適に旅行に出発したいのである．面倒な航空券やホテルの予約などはできれば避けたい．旅行者が望むことは，例えば，「これからはじまる旅行を，ワクワクした気持ちでシミュレーションしたい」かもしれないし（その過程を通して，いつの間にか必要な予約ができているといった方法も十分考えられる），人によっては，「旅行先で待ち受ける未知の体験を大切にしたいので，泊まるホテルの情報はあまり知りたくない」といったようなドキドキ感を期待する人もいるかもしれない．

我々が行いたいことは，顧客の経験価値を最大化するプロセスを設計することである．旅行者が旅行の準備をする過程で，「旅行者にとって，何が嬉しいのか」，「それが，どのように嬉しいのか」，「それは何故なのか」といった問いを，旅行者の視点から分析・理解しながら業務プロセスを設計したい．そのために，「要するに旅行の予約をするということはどういうことなのか」という本質を理解したい．しかし，現状の業務プロセスのモデル化方法では，そもそも，何故そのようなプロセスになっているのかといった，本質的な意味や理由を理解することが難しという声があった．

1.3. 現状の業務プロセスモデルの問題の原因仮説

確かに，現行の方法は，業務プロセスモデルからサービスを定義・特定し，業務を支援するシステムとその連携を定義するには有効であると考える．しかし，前項で述べた下記の3項目の問題は，どこから来るのだろうか

1) モデル化自体の困難さの問題
2) 業務改善のための有効性の問題
3) 業務プロセスの本質的な意味・理由の理解の困難さの問題

プロセスモデルの分類にはいつかの観点がある．1990年に開催された第6回ソフトウェアプロセスワークショップ（6th ISPW）では，記述モデルの主な目的，記述表現，記述要素，動作制御の方法，そして基礎となる言語による分類等によって，18種類のプロセスモデルが比較検討された[5]．その結果，Humphreyらは，プロセスモデルを，表1に示すような抽象度によって，「Universal」，「Worldly」，「Atomic」の3 段階に分類・定義できるとした[6]．

Universalはプロセスの概念を提供する抽象度の高いレベルであり，プロセスを目的的に概観するために有効である．Universalとは逆に，抽象度が低く特定の作業手順をモデル化するレベルがAtomicである．Universal レベルとAtomicレベルとの間の抽象度で，実際の開発活動をモデル化するレベルがWorldlyレベルである．Worldlyレベルは，実際に行われている活動の実体をモデル化するものであり，プロセス改善の検討や再利用に適しているとしている[6]．

また，プロセスをモデル化する観点からの分類として，タスク観点と成果物観点の区分けがある[7]．作業行為に着目してプロセスをモデル化する方法がタスク観点であり，作業成果物（以下「成果物」）の関係に着目してモデル化する方法が成果物観点である．UniversalとAtomicレベルのプロセスのモデル化方法は主にタスク観点によるモデル化が取られており，Worldlyレベルのモデル化は，成果物観点によるモデル化が有効であるとされている[6]．プロセスモデルの抽象度とモデル化の観点を表1に整理する．

表1. プロセスモデルの抽象度による分類

抽象度	モデルの利用	モデル化方法
Universal	プロセス概念の理解	タスク観点
Worldly	実際の開発活動の理解・改善・管理・再利用	成果物観点
Atomic	特定の作業手順の記述	タスク観点

現状の業務プロセスのモデル化方法は，主にタスク観点によるモデル化方法がとられている．そのために，モデル化されたプロセスは概念のレベル（Universal）か，もしくは，特定の作業手順の記述のレベル（Atomic）となる．「適切な粒度でのプロセスのモデル化が難しい」，「抽象度のレベルが混在してしまう」といった問題は，現行のプロセスモデルがタスク観点によるモデル化を行っているために，概念レベルと作業手順のレベルとが混在してしまうのではないかと考えられる．

例えば，図1に示した旅行予約プロセスにおいても，「旅程をチェックする」が，概念のレベル，つまり，旅程をチェックするという行為のレベルともとれるし，具体的な作業，つまり，旅程をチェックするという作業のレベルとも解釈できる．現在の業務プロセスのモデル化方法は，タスク観点によるモデル化方法がとられているため，目的と手段の混在や，抽象度や粒度の定義の困難さが生じ，モデル化自体の困難さの問題の原因となっているのではないかと仮説する．

また，Humphrey らは，プロセスのモデル化はプロセス改善のための重要な要素であり，プロセス改善のためのプロセスモデルは次に示す目的のもとに使用されるとしている[8]．

- プロセスに関する効果的なコミュニケーションの実現
- プロセス改善の支援
- プロセスの管理

さらにHumphrey らは，上記3項目を実現可能とするプロセスモデルの要件として下記の3 項目を定義し，これらの要件を満足するためには，Worldlyレベルでのプロセスのモデル化が有効であるとしている[6),8)]．

1) 現実に行われている，または行われるべき活動をモデル化できること
2) プロセスのモデル化と改善を行うために十分であるとともに柔軟で理解が容易であること
3) 必要とする粒度でのプロセスの洗練が可能であること

すなわち，業務プロセスの本質的な意味や理由を顧客の視点から理解・分析し，利害関係者間で効果的なコミュニケーションをとりながら，顧客の経験価値を最大化するプロセスを設計するためには，Worldlyレベル，すなわち，成果物観点によるプロセスのモデル化方法が有効であるとの仮説を立てた．

ところで，「プロセス」という語彙は曖昧性を含んでおり，分野によって解釈が異なる．現状の業務プロセスモデルでは，主にタスク観点がとられている．その理由としては，例えば，システム開発の分野では，「時系列に並んだイベントの集まり」が一般的に「プロセス」として定義されており[9]，逐次的な処理として業務の場合の「タスク」や「アクティビティー」が想起されるためではないかと思われる．一方で，プロセス改善の分野では，例えばCMMの場合，「プロセス」は「一連の活動，手法，プラクティス，変換」と定義されている[10]．ある目的を持った一連の活動であり，必ずしも手順やフローとして記述されるものではない．

2. PRePモデルの業務プロセスへの適用

PRePモデルは，Worldlyレベルでのプロセスのモデル化方法として，ソフトウェア開発プロセス改善のために考案した成果物観点のプロセスのモデル化方法である．Worldlyレベルの抽象度のモデルを提供することによって，プロセスの改善や，ソフトウェアとハードウェアの協調開発プロセス設計のために考案した方法である[1]．

前節の仮説を検証するために，ソフトウェアプロセス改善のために開発した成果物観点によるプロセスのモデル化方法を，実際の業務プロセス開発へ適用した．さらに，業務プロセスへ適用する際に重要であった観点に関して適用経験から報告する．

2.1. PRePモデルの特徴

成果物観点によるプロセスモデルは，何を成果物として定義するかがモデリング時の主要な観点となる．成果物の定義を含め，PRePモデルは主に次の特徴を持つ．

1）成果物の定義方法
- 「プロセスの目的から見た意味的なチャンク」を成果物ノードとしてPRePモデルでは「成果物」と呼ぶことにする（「チャンク」とは，人が情報を理解・認識する際の情報のまとまりであり，単位が固定されない）．「◯◯帳票」といった物理的な実体ではなく，意味的なまとまりを定義する（そのため，物理的なひとつの帳票は複数の成果物として定義される場合もある）．
- プロセスモデルを構成する成果物を「定義する業務プロセスにおいて，共有されかつ管理されているもの」のみとして定義
- 成果物を，プロセスの品質目標を管理する視点から「中間成果物」と「マイルストン成果物」に分類し，外部プロセスへ引き渡される成果物を「最終成果物」として定義
- 成果物は状態を持つ．例えば，「申込書」という成果物に関して，「受け付けられた」と「確認された」といった状態に対応して，２つの異なる成果物として定義する

2）成果物の関係の定義方法
- 成果物間の関係を，「入力関係」と「同期関係」の2種類で表現

3）Backwardに検証する
- 描いたプロセスの検証を行う場合，後ろから前方向に，成果物の関係を逆方向に辿る

2.2. 業務プロセスのモデル化のための拡張

PRePモデルを業務プロセスモデルへ適用するために，次項に示す拡張を行った．

2.2.1. 基本フレームの定義

業務プロセスのモデル化を行うために，新たに「業務スコープ」，「業務ゴール」，「最終成果物」，「サブプロセス」の概念を追加し，図2に示す業務プロセスの基本フレームを定義した．

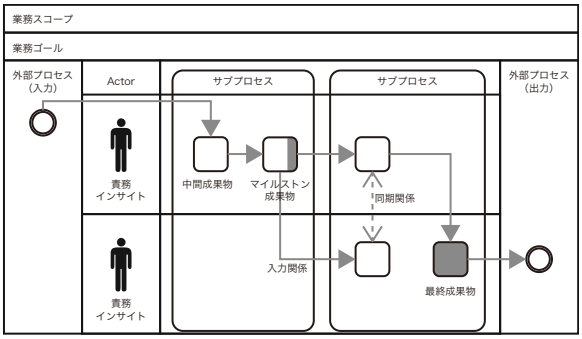


図2. 業務プロセスの基本フレーム

また，基本フレームは，以下の関係を有するものとして定義した．
- 業務プロセスは，経営視点から定義される業務の目的（ゴール）を持つ
- 業務プロセスのスコープは，最終成果物によって定義される
- 最終成果物は，その成果物が利用される外部プロセスによって意味が定義される
- 最終成果物は，業務の目的と一貫した対応を持つ

上記の定義から，業務プロセスのスコープ定義が確定される．すなわち，業務プロセスは，経営視点から見た関連する外部プロセスとの関係の上で，最終成果物を媒介としてそのスコープと目的的意味（経営的視点から見た業務の意味）が定義される．そして，業務プロセスのゴールと品質目標が定義される．

2.2.2. 業務プロセス管理のための構造の定義

さらに，上記で定義される業務プロセスのゴールと品質目標を実現・管理するために，個々の業務プロセスは，以下のプロセス管理構造を持つとした．

- 業務プロセスのゴールと品質目標を実現・管理するための管理ゲートのための成果物として，業務プロセスにはマイルストーン成果物が定義される
- マイルストーンで区切られる一連のプロセスは，業務プロセスのゴールと品質目標を実現・管理するという目的で認識される．これを「サブプロセス」と定義する

2.2.3. 経験価値視点による分析要素の追加

また，アクターに「責務」属性と「インサイト」属性を追加した．インサイトとは，アクターの本音であり，例えば，アクターの責務が業務上では「適正数量を発注すること」と定義されるも，本音の部分では，「欠品になったら大変なので適正量よりも多めに注文しておこう」と動いてしまうようなことである．このようなインサイトは，業務リスクを分析する上で重要な観点になることが，今回の適用事例でも確認された．

2.3. PRePによる現状プロセスの分析方法

PRePモデルを用いて現状（As-is）のプロセスをモデル化し分析する場合，基本的に以下の考えかたと順番となる．ただし，実際の作業は反復的に進められる．

1）対象業務ドメインの理解
- プロセスをモデル化する対象の業務スコープを，登場する組織，アクター，大枠の業務概念と業務間でやり取りされる（インタフェースとなっている）成果物をもって把握する

2）業務フレームの理解
- モデル化対象業務の枠組みを理解するために，最終成果物に着目し，最終成果物が関連する外部プロセス，最終成果物と業務プロセスのゴール，当該業務のアクターとその責務を把握する

3）業務構造の理解
- 当該業務プロセスを，業務の品質管理のゲートとなっているマイルストーン成果物に着目しながら入力関係と同期関係の構造によって理解していく
- また，何故そのような業務構造になっているか，マイルストーン成果物および最終成果物から関係を逆方向に辿りながら，その理由を，技術モデルとリソースモデルを適宜理解し，モデルを検証・精緻化していく

4）業務間の関係の理解
- 業務全体の関係を，業務間の入出力となっている成果物をたどりながら，その繋がりを検証する

5）問題の把握と原因の特定
- まずはじめに，業務上のQCDの問題が現象している成果物を特定する
- 問題は，原因によって引き起こされる．原因となっている成果物は，問題が現象している成果物の入力側にある場合が多い．PRePモデルでは，問題が現象している成果物に対して，その入力側の成果物と同期している成果物の関係から理解することによって，原因を特定する
- 原因となっている成果物に着目し，担当アクターの責務やインサイトなどを考慮しながら本質的な問題を特定していく

6）ボトルネックの特定
- 各業務の問題構造とその原因の関係理解ができたら，モデル化された業務全体を俯瞰し，スコープとなっている業務全体の中で，一番の障害の根本原因とその理由を理解する

2.4. PRePモデルを用いた改善プロセスの設計

プロセスの改善，すなわちTo-beプロセスの設計は，As-isプロセス全体の中の最大のボトルネックに着目して，それを解消することに集中して改善方法を考える．その場合，適用技術制約やリソース制約の見直しなども考慮する．リソース制約の見直しでは，アクターの責務の見直しも含め，組織レベルの再構築も考慮する必要もある．

2.5. 業務プロセスモデルからシステム要求へ

PRePモデルでは，個々の成果物を「プロセスの目的から見た意味的チャンク」として定義している．そのため，個々の成果物ノードは，業務を記述する際の適切な概念の単位となっていると考えられる．例えば，PRePモデルでは，「申込書」という帳票実体を経営視点から見た業務の意味チャンクとして再定義する．その結果，「申込書」は，業務の意味からとらえると「申込者情報の取得」と「申し込み証票の取得」の目的に分解された別々の成果物として定義され，それぞれが業務機能と対応する場合がある．「申込書」という帳票は，2つの機能を実装するために考えられた実体，すなわち設計の結果である（一般的な業務プロセスモデリングでは「申し込みをする」と一つのタスクとして定義されてしまう場合が多い）．このように定義された意味的チャンクとしてのPRePの「成果物」は，データモデリングでのひとつのエンティティーを業務観点から定義する際の主要な観点になると考えられる．

PRePモデルの業務プロセスモデル拡張で定義した，業務，サブプロセス，成果物という構造は，それぞれ，業務の目的と品質，業務目的と品質実現のための業務のリスク管理ゲート，業務を実行管理するための意味的概念要素に対応する．この構造に合わせて業務を支援するITシステムへの要求を整理することによって，ITシステムの支援目的，業務リスク管理目的，業務実行管理目的といった構成でITシステムへの要求を整理することができ，支援する業務との対応が取りやすくなると考えた．

2.6. ワークショップ形式による業務プロセスの改善

筆者らは，超上流工程を進めるための前提条件として下記を定義している[3)]．

- ユーザ企業内のステークホルダによって，ビジネス要求とシステム要件とが並行に開発されていること
- ビジネス要求とシステム要件とが，各ステークホルダによって自発的に合意され，ユーザ企業の意思決定を構成するものになっていること．すなわち，システム要件のレビュープロセスが，企業内での経営的意思決定プロセスと連携して実行されていること
- ビジネス要求とシステム要件の記述内容が，抽象的なレベルではなく，判断・行動の指針となる具体的なレベルで記載されていること

上記条件を満足するためには，業務プロセスの改善とシステム要求定義を，業務側と情報システム側両者のステークホルダ参加によるワークショップ形式による共同作業として進めることが有効であると考えている．

3. PRePモデル ツール

業務プロセス改善のための初期の探索的なモデル化作業から，To-beの具体的な業務とシステムの要求を詳細に定義するまで，モデルを洗練させ成長させていく必要がある．そのための一貫した手段を提供するために支援ツールを開発した．

本ツールは，Microsoft社のVisio 2013のAdd-inツールとして開発し，ワークショップ形式による業務プロセスのモデル化作業を支援するとともに，定義した業務プロセスモデル上でシステムスコープ定義と要求の整理を行うことができる．そして，最終的にシステムスコープ・要求書と業務定義書とをExcel形式で出力することができる．図3にプロセスモデル画面，図4に，Excel形式で出力されたシステムスコープ・要求定義書の代表的な画面を示す．

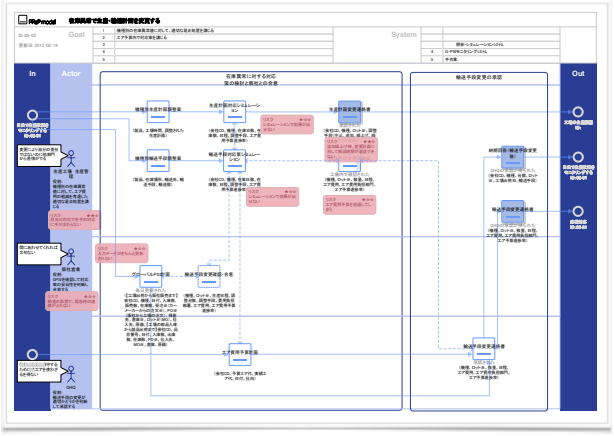


図3. PRePモデルツールでのプロセスモデル画面の例

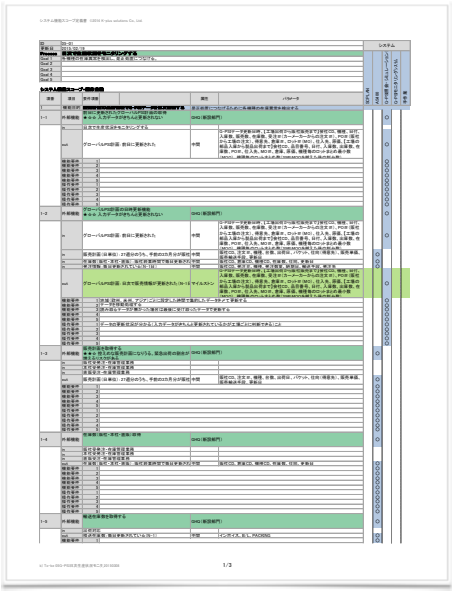


図4. PRePによる業務プロセスモデルからExcel形式で出力されたシステムスコープ・要求定義書画面の例

4. グローバルPSIシステム構築への適用

クラリオンは，主に車載機器を中心に，日本，アジア，欧州，北中南米とグローバルな販売，生産，流通の拠点を持つ．今回のPSI 業務とシステムの改善では，これまでそれぞれの地域で行われてきたPSI業務を，グローバルな観点から全体最適化し，経営目標として定めた在庫回転率を達成しようというプロジェクト「グローバルPSIプロジェクト」である．

PRePモデルをグローバルPSIプロジェクトに適用するにあたって，前節で述べた「超上流工程を進めるための前提条件」に従って下記を考慮した．

1）ユーザ企業内の有識者に参画してもらうこと：現在の業務とシステムを意味的に理解しているメンバであり，定義した業務とシステムの要件の実装時のリーダーレベルのメンバを「有識者メンバ」として参画してもらうことを条件とした

2）ビジネス要求とシステム要求とを並行に開発すること：業務有識者とシステム有識者が一緒になってワークショップを進めることを条件とした
3）超上流工程が，企業内での経営的意思決定プロセスと連携すること：適切なタイミングで，本テーマのステークホルダのレビュープロセスを組み込むとともに，経営的判断が必要となる局面（業務とシステムのあるべき姿の方向性の判断）においては，経営ステアリングコミッティの合意形成プロセスを組み込んだ
4）ビジネス要求とシステム要件の記述内容を，業務とシステムの実装がイメージできる具体的なレベルで定義すること：業務改革プロジェクトとシステム構築プロジェクトへのスムーズな引き継ぎを実現する

また，PRePモデルの本事例への適用を通して下記が行えることと，その効果を検証した．

- 業務プロセスの改善課題の特定ができること
- 改善課題を解決するための業務プロセスと業務機能の設計（業務改革への入力となる）
- 業務プロセスから出力されるシステム要求定義の有用性（システム構築への入力となる）

4.1. 実際の適用過程

今回のプロジェクトは表2および図7に示すように進んだ．本プロジェクトに適用した超上流工程の基本プロセスは，現状の業務プロセスを外在化し理解するというリバースエンジニアリングの過程と，定義した改善課題を解決するための業務プロセスを設計し，ITシステムの要件を定義するデザインの過程からなる「逆Vモデル」[3] を基にした．

写真1. As-is業務の理解ワークショップ

写真2．PRePモデルツール利用

写真3. ボトルネックの分析

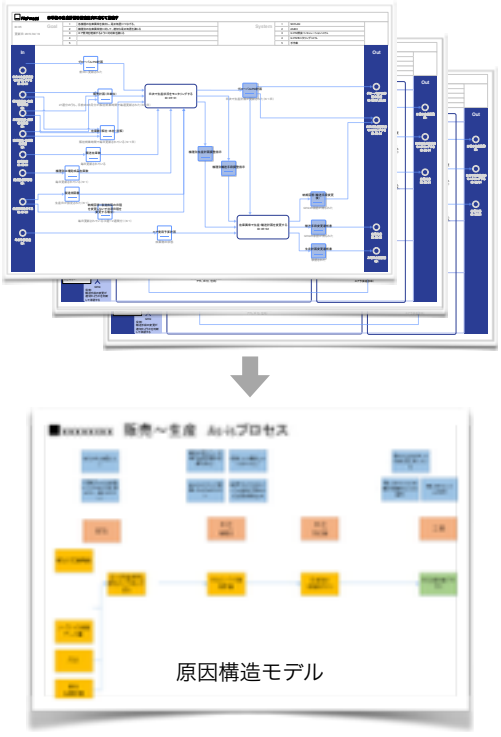


図5. As-isプロセスの分析からボトルネックを特定し，原因構造モデルを作成（ポストイットを使用）

図6. 原因構造モデルからTo-beの主要構造を設計（ポストイットを使用）したのち，各プロセスの詳細設計に入る

表2. クラリオン グローバルPSIへのPRePモデルの適用過程

項番	ワークショップ回数	目的	実施項目	関与者	実施内容
1	2	理解ベースラインの形成	PRePモデル研修	情報システム部門全員	情報システム部門全員 に対して1.5日のPRePモデル研修を実施.
			プロジェクトキックオフ	業務部門と情報システム部門のマネージャ	参加メンバ要件（有識者の参画を条件とすること）とプロジェクトの基本計画の合意．プロジェクトの基本情報共有．目的と成功基準をODSCに従って確認．ステークホルダマップ（誰が嬉しいのか）と競争力モデル（どのように嬉しいのか）の確認
2	5	As-isプロセス分析とボトルネックの特定	As-is業務プロセスモデル化	有識者	ホワイトボードを使用し，現在の業務の基本概念を理解．模造紙版PRePモデルテンプレートに，基本概念理解の過程で見えてきた業務スコープや成果物をポストイットで貼り出し，主要業務の大枠をつかんだ（写真1）．現状の業務プロセスのモデル化がある程度見えてきたところで，PRePモデルツールに落とした（写真2）．また，必要に応じてAs-is業務プロセスモデル化とあわせて業務プロセスMapを作成し，業務間の関係を確認した.
			技術・リソースモデル分析 リスク・インサイト分析	有識者	現行のPSIの管理技術とリソースをモデル化して理解したうえで，As-is業務プロセスに対して，リスク分析とインサイト分析を行い，重要なボトルネックを特定した．PRePモデルによる分析では，業務の構造をプロセスの後ろ（最終成果物）から前方向に，プロセス構造の理由に着目してレビューを行う．特定されたボトルネックの原因部分において，プロセスの構造とアクターのインサイトが相互に関連していることがわかった．写真3 はボトルネックが見えた瞬間
3	1	改善ポイントの合意	経営方針確認	ステークホルダ	As-isとして分析された業務プロセスから改善モデルを特定するために，あらかじめ収集していた経営目標要求と経営制約を再度確認
			改善ポイント合意	ステークホルダ	ボトルネックの発生原因の構造を取り出し，分かりやすくモデル化してステークホルダに説明．潜在する制約などをステークホルダから引き出しながら改善すべきポイントの合意形成を図った.
4	8	To-be業務プロセス基本設計	As-is改善ポイントのモデル化	コアメンバ	To-beプロセスの設計は人数を5人（業務部門と情報システム部門から構成）に絞って進めた．To-be設計にあたり，原因構造をポストイットを用いでモデル化（図5）.
			To-be主要構造設計	コアメンバ	原因構造からTo-beの主要構造をポストイットを使ったワークショップ形式で設計（図6）.
			To-beプロセス基本設計	コアメンバ	To-beの基本構造が見えてきたところで，PRePモデルのツールを使ってプロセスの設計を進めた（図6）．必要に応じて印刷し，プロセス間の関係などを確認した．業務プロセスの設計では，意識的に，実現方法（How）は意識的に考慮から外した．As-is分析で特定されたボトルネックの原因を構造的に分析することによって，PSI業務の全体最適化を実現するためのTo-beプロセス構造が見えてきた．To-beプロセスの設計に伴い，新たな組織構造や責務要件が見えてきた.
			改善方針のステークホルダ合意	ステークホルダ	To-be業務プロセスの検証と，あとに続く業務改革とITシステム構築をスムーズに進めるために，ステークホルダのレビュー受け，基本合意を得た.
			ステアリングコミッティからのコミットメント獲得	コアメンバ ステークホルダ ステアリングコミッティ	設計したTo-be業務プロセスとその効果を説明し，あとに続く業務改革プロジェウトとITシステム開発に関する経営判断を仰ぎ，コミットメントを獲得した．ステアリングコミッティー向けの説明資料は別途作成し，ステークホルダによるレビューを受けた.
5	8	To-beプロセス詳細設計とシステムスコープ・要件定義	To-beプロセス詳細設計	コアメンバ	設計した業務プロセスをもとに，業務とシステムの実装を具体的にイメージしながら，ITシステムの要件を定義し，To-beのプロセスを完成させた.
			システムスコープ・要件定義	コアメンバ	PRePモデルでは，業務プロセスとの関係の上で成果物パラメータの精緻化，システム割り振り，機能要件・操作要件定義を行うことによってシステムスコープ定義・要件定義書がExcelフォーマットで出力される．プロセスモデルが実装レベルで具体化する過程で，それまで見えていなかった業務とシステムに関する様々な制約が見え，それにともなって，To-beプロセスも修正が加えられた.
			To-beプロセスの検証	コアメンバ，業務ステークホルダ	設計したTo-be業務が正しく動くかを検証するために，実際に業務を行っているステークホルダを交えたステージプロトタイピングを用いたウォークスルー検証を行った.
			モデルの精査とまとめ	コアメンバ	検証結果をTo-be業務プロセスへ反映させ，To-be業務プロセスとシステムスコープ・要件定義のベースラインをリリース．あとに続く業務改革プロジェクトとITシステム開発プロジェクトへ引くつぐ課題を整理した.

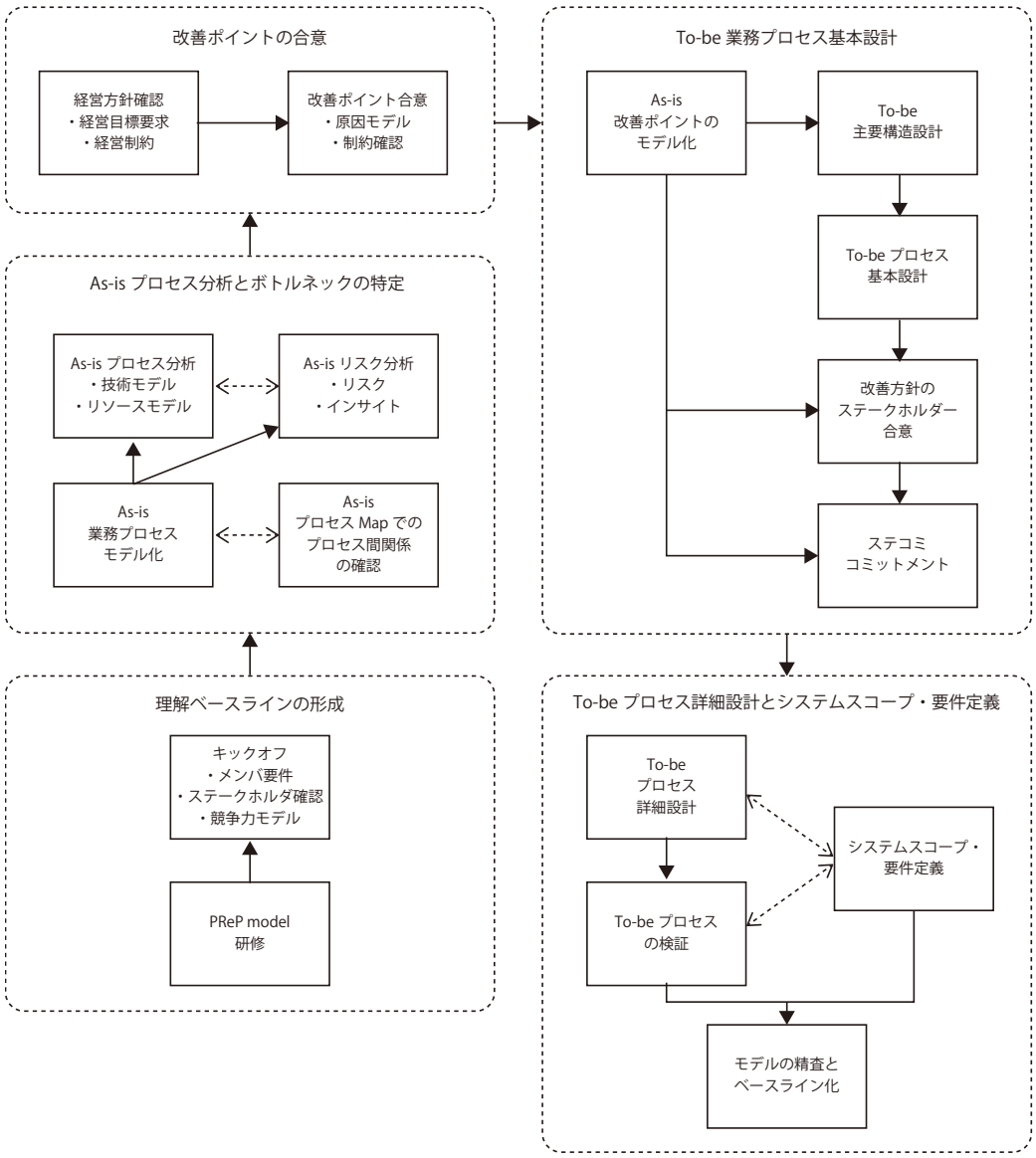


図7. クラリオン グローバルPSIへのPRePモデルの適用プロセス

4.2. 適用結果

今回のプロジェクトでは，PRePモデルを適用することによって，As-is業務プロセスのモデル化から改善課題を特定し，To-beの業務設計と，システムスコープ定義，システム要求の整理を行った．また，あとに続く業務改革とシステム開発プロジェクトそれぞれの実行計画までを，参加有識者とともに策定した．その結果，今回の適用を通して，下記の効果を確認した．

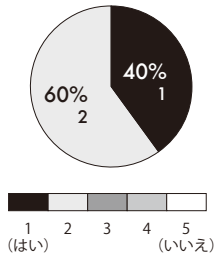
- 業務部門のメンバーと情報部門のメンバーとが非常に良い協力関係でワークショップを進めることができた
- 特に，業務部門のメンバーが率先して業務プロセスのモデル化をリードすることができた
- 現状の業務プロセスの最大のボトルネックとその原因を，問題が顕在化している成果物とその原因成果物の観点，さらに，関係アクターのインサイト分析の結果から特定することができた
- ボトルネックの原因を成果物の関係から理解することによってTo-beの基本構造を設計することができた
- その結果，業務改善では，アクターの責務定義の大きな見直しと，組織の大幅な変更の必要性を特定することができた
- 後続するシステム開発とに引き渡ためのシステムスコープ・要求定義書を出力することができた

さらに，今回のプロジェクトにおけるPRePモデルの適用に関して，参加メンバーアンケートを行った．アンケート内容は，Humphreyらがプロセス改善においてプロセスモデルが持たなければならないとして定義したプロセスモデルの下記の3要件[8] にもとづいて行った．

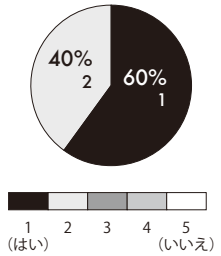
- 現実に行われている，または行われるべき活動をモデル化できること
- プロセスのモデル化と改善を行うために十分であるとともに柔軟で理解が容易であること
- 必要とする粒度でのプロセスの洗練が可能であること

アンケート項目とその結果を以下に示す.

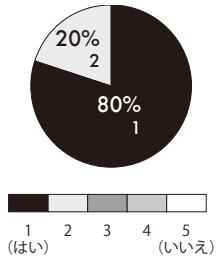
1. As-isの業務プロセスとして，現実に行われている活動をモデル化することができましたか



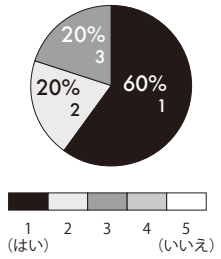
2. To-be の業務プロセスとして，Global-PSIの実現に向けて実現すべきことをモデル化することができましたたか



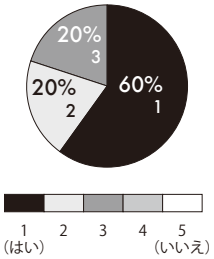
3. To-be の業務プロセスを検討するためのモデル化方法として，柔軟で理解が容易でしたか



4. To-be のシステム要件を検討するためのモデル化方法として，柔軟で理解が容易でしたか



5. To-be の業務プロセスを検討するためのモデル化方法として，必要とする粒度でのプロセスの洗練が可能でしたか



6. 総合的に良かった点を教えてください（自由記述）
   - 関係部門の人達と一緒にAs-is及びTo-beを整理したことで情報の共有化が図られた
   - 進むべきベクトルの向きボリュームが明確になった
   - 資料を整理するツールにより，統一された分かり易い資料が作り上げられた
   - 今回経験した手法，ツールが今後のシステム開発に活用できる
   - 普段のシステム企画や設計では，このように時間をかけてじっくり作りこめる機会は少ない（余裕が無かったということでしょうか．．．）ため，じっくりと纏められたことは良かったと思っています．特に組織が硬直していたり，マンネリ化している日本では，時間をかけて空気を盛り上げて行くことは必要かもしれず，，，ただし，そういったアプローチは社内の力だけでは難しいので，コンサルタントの話の進め方に非常に助けられたと思っています．本来であれば，若手が参加すべき内容であったと思いますが，，，なかなか余裕がないですね
   - 現状業務を再理解することができた（知らなかったことも含めて）
   - プロセスを検討する上で，現状困っている事に対しての改善策を導き出す方法など，理解できた
   - 第三者的な観点で進めていただいたことが良かった
   - まとめや進行など全体的に俯瞰して見ていただいたことが良かった
   - 核となる業務単位にフローを整理することで業務の流れが理解できる
   - 業務間の連携，関連度が検証できた
   - プロセスを検討する上で，現状困っている事に対しての改善策を導き出す方法などが理解できた
7. 改善点を教えてください（自由記述）
   - 核となる業務を繋ぎ合わせた全体像が確認できると良いのでは．
   - 最終アウトプットまでのステップ感や手順が見えなかった．
8. 今回の取り組みが成功するための条件として考えられることは何でしたか（自由記述）
   - 無理と思っていてもやってみようと思うこと
   - 業務に精通した関係部門の人の参加，及び現状を変えたいという意思とあるべき姿を持っている人の参加
   - 会議の進行をコントロールするコンサルタントの参加

- 業務全体の流れを知らないと踏み込めない，また，踏み込んでいかないと改善策がうまく出来ない
- 参加人材と意識改革と目的・目標の共有
- 活動計画と，そのチェックポイント
- ITと業務の連携
- プロジェクトメンバを中心とした意識の改革が必要と感じている
- 若手を中心とした人材の育成も重要

9. まとめ

経営視点から業務改善を行い，システムのスコープと要求を特定する超上流工程において，現行の業務プロセスのモデル化に関して，現場から下記の声が上がってきていた．

- モデル化自体の困難さの問題
- 業務改善のための有効性の問題
- 業務プロセスの本質的な意味・理由の理解の困難さの問題

上記の問題に対して，成果物観点によるプロセスのモデル化方法が有効であると仮説し，ソフトウェア開発プロセス改善のための成果物観点のプロセスのモデル化方法であるPRePモデルを，業務プロセスのモデル化のために拡張し，実プロジェクトへ適用した．

適用結果を，Humphreyらが定義したプロセス改善においてプロセスモデルが持つべき3要件にもとづいて評価を行った．その結果，3要件を概ね満足していることがわかった．また，実際の適用の中で下記の効果を認識した

- SEではなく，業務部門のメンバーが率先して業務プロセスのモデル化をリードすることができた
- 現状の業務プロセスのボトルの原因を特定する際に，インサイト分析が効果的であった

10. 適用から見えた課題

現状の業務プロセスをモデル化を通して理解・分析し，さらにTo-beを設計するという過程は，探索型であり反復的な学習のプロセスでもあった．そのため，我々が何を発見し，それをどのように解決できるのか（もしくは，できないのか）を予測することが難しく，初期段階で精度の高い計画を立てられなかった．チームメンバからは「全体感が持ちにくい」という意見が出た．

探索型の学習プロセスは，そもそも，進行に伴ってすべきことや進めかたが調整されていく．作業を定型化し，標準的なプロセスやワークフローを適用しようとすると，逆に，効率や品質が低下する場合もある．このようなプロセスを成功させるためには，より変動的な状況下で活動する状況適合的なチーム作りが必要である．今回のプロジェクトの大きな成功要因は，意識の高いメンバーが参加し，非常に良いチームワークのもとで作業を進めることができたことにあると考えている．一方で，適用するモデル化手法や支援ツール以前に，今回のようなチームが構成できなければ，プロジェクトの成功はとても困難であったと思われる．

参考文献

1) 田中, 飯田, 松本: 成果物間の関連に着目した開発プロセスモデルPReP:(PReP: Product-Based Modeling Method for Software Process,) 情報処理学会論文誌, 2004 年「社会人学生」論文, 2005.
2) ソフトウェアエンジニアリングセンター（編）：経営者が参画する要求品質の確保～超上流から攻めるIT化の勘所～, 第２版, （独）情報処理推進機構, p.35（2006）.
3) 高田, 豊田, 渡辺, 曽我, 中川, 田中: 超上流工程における合意形成手法「Exアプローチ」, 情報処理学会デジタルプラクティス, Vol.4 No.2(Apr. 2013), pp.133-140, 2013.
4) Michael Havey: 詳説ビジネスプロセスモデリング, p.4, 2006.
5) M. Kellner, D. Rombach: Session summary, Comparisons of Software Process Descriptions, Proc, 6th International Software Process Workshop, IEEE Computer Society Press, pp.7-18, Hakodate, Japan, October 1990.
6) Watts S. Humphrey. : Managing the Software Process, Addison-Wesley, ISBN-0201180952, 1989. 藤野喜一（監訳）：ソフトウェアプロセス成熟度の改善, 日科技連, pp.268-274, pp.298-300, 1991.
7) 井上, 松本, 飯田. :ソフトウェアプロセス, 共立出版, p.27, 2000.
8) Watts S. Humphrey, Marc I. Kellner. : Software Process Modeling: Principles of Entity Process Models, CMU/SEI-89-TR-002, 1989.
9) マイケル・ジャクソン. : プロブレムフレーム, p.382, 2006.
10) M. Paulk, B. Curtis, M. Chrissis, C. Weber. : Capability Maturity Model for Software (Version1.1), CMU/SEI-93-TR-024, 1993.

ソフトウェア欠陥数予測におけるトービットモデルの適用

村上 優佳紗
近畿大学理工学部情報学科
m.yukasa@gmail.com

角田 雅照
近畿大学理工学部情報学科
tsunoda@info.kindai.ac.jp

戸田 航史
福岡工業大学情報工学科
toda@fit.ac.jp

要旨

本研究では，ソフトウェア欠陥数予測に対しトービットモデルを適用し，その効果を確かめた．実際のソフトウェア開発プロジェクトにおいて収集されたデータを用いてソフトウェア欠陥数を予測した．最小二乗法とトービットモデル，およびそれぞれに対数変換を適用した 4 つのモデルを評価した．その結果，対数変換を適用したトービットモデルは検討すべきモデルであるといえる．

1. はじめに

近年，プロジェクトは大規模化しており，納期の遅れ，品質の低下，コストの超過などを防止する重要性が高まっている．それらを防止するためには，定量的なプロジェクト管理は欠くことができない要素の一つである．例えば，開発言語やソフトウェアの規模に基づき，ソフトウェアリリース後に発見される欠陥数を予測し，欠陥数の予測値に基づき，テスト期間やテストケース数，テスト人員が適切となるように計画を立案する．

予測するために用いられる数学的モデルとして重回帰モデルなどがあり，過去に得られたデータを基に予測モデルが構築される．しかし，ソフトウェアプロジェクトのデータをそのまま用いた場合，ソフトウェアの欠陥数など，最小値が0となる変数がある．このため，最小二乗法に基づく重回帰モデルでは適切なモデルが構築されず，欠陥数や開発工数の予測値が負の値となる可能性がある．例えば，開発規模を説明変数とし，開発規模が100FP（ファンクションポイント）で欠陥数が0というデータが含まれるデータセットを用いて（対数変換を行わずに）モデルを構築したとする．この場合，予測対象プロジェクトの開発規模が100FP未満，例えば50FPの時，欠陥数の予測値が負の値となる可能性がある．この問題を解決するために，本研究ではトービットモデルをソフトウェア欠陥数予測に適用することを提案する．

2. トービットモデル

トービットモデルとは，データの分布の偏りを考慮したモデルである．データの偏りの分布は以下の3つに分類される[4]．

- 打ち切り（censored）
- 切断（truncated）
- 付随的切断（incidental truncation）

打ち切りとは，欠陥数など，目的変数の最小値が 0 以上など，値域があらかじめ決まっている場合である．切断とは，欠陥数が 0 のデータを除外するような場合である．付随的切断とは，レビュー指摘数の省略などが理由で本来 0 でない数値が 0 となる場合である．

ソフトウェア欠陥数は打ち切りに該当するため，本研究ではタイプ I のトービットモデルを適用する．タイプ I トービットモデルの打ち切り回帰モデルは以下のようになる[4]．

$$y^* = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \varepsilon \qquad (1)$$

$$y = \begin{cases} y^* & y^* > 0 \\ 0 & y^* \le 0 \end{cases} \qquad (2)$$

ここでβ_0は回帰定数，$\beta_1, \beta_2, \ldots, \beta_k$は偏回帰係数，$\varepsilon$は誤差項である．最小二乗法では$y$が 0 となっているケースについても 1 つのモデル（式(1)のみ）で扱うが，トービットモデルではそれらを別の式（式(2)）で扱うことにより，モデルが適切に構築される．トービットモデルはかなり以前に提案されたものであるが，我々の知る限り，これまでソフトウェア欠陥数の予測モデル構築に適用されていない．

図 1 に，説明変数を開発規模，目的変数をソフトウェア欠陥数として予測モデルを構築した場合のトービットモデルと最小二乗法に基づく重回帰モデル（OLS）のイメージを示す．グラフ内にプロットされている点は各プロジェクトのケースを表す．最小二乗法に基づく重回帰モデルと比べて，トービットモデルは欠陥数が 0 のケースを適切に扱えていることがわかる．

ソフトウェア欠陥数の予測は，ソフトウェアのテスト計画を立案する際に，ソフトウェアの品質を予測するために行

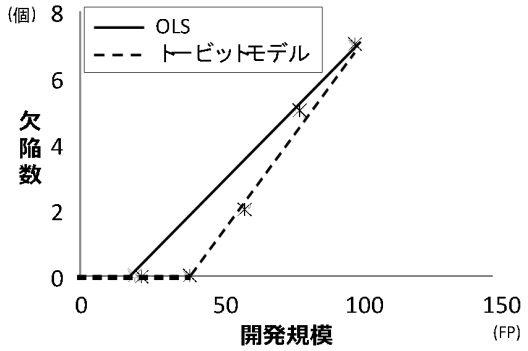


図 1 トービットモデルと最小二乗法に基づく重回帰モデル

われる. 例えば品質が低い, すなわち欠陥数が多いと予測された場合, テストケース数の計画数を増加させるなどし, 欠陥の発生を抑える.

なお, ソフトウェアの欠陥数が0となっているケースが多く存在するという, データの偏りが存在する理由は, データの収集方法などに問題があるためではない. 一般的なソフトウェア開発において, リリース後の欠陥数は0となる場合が多い. このために上述の偏りが発生する. 従来の最小二乗法に基づく重回帰モデルでは誤差には偏りがないという前提のため, この偏りが考慮されないが, トービットモデルでは偏りを考慮したモデルが構築される. そのため, 予測精度の向上が期待される.

3. 実験

最小二乗法に基づく重回帰モデル(OLS), トービットモデル(Tobit), 対数変換を適用した最小二乗法に基づく重回帰モデル(OLS_ln), 対数変換を適用したトービットモデル(Tobit_ln)の計4つを用いてソフトウェア開発プロジェクトにおける欠陥数の予測を行い, 精度を比較した. 0は対数変換できないため, 各変数にあらかじめ1を加えた上で対数変換した. 欠陥数は0以上の整数であり, その最大値は149であったため, 1を加えることとした. なお, これは対数変換をする場合のみの処理であり, 対数変換後の値は0となる.

提案手法の有用性を評価するために, ISBSG (International Software Benchmarking Standards Group)が収集したプロジェクトデータ(ISBSGデータ[1])を用いて分析を行った. ISBSGデータは, 世界20ヶ国のソフトウェア開発企業から収集されたものであり, 予測手法の有効性を評価する研究で広く用いられている[2][3][5]. 用いたデータセットはRelease 9と呼ばれるバージョンであり, 1989年から2004年に実施されたソフトウェア開発プロジェクトが3026件, 変数が99個含まれている. データには欠損値が含まれている. 欠損値とは値が記録されていないことを指す.

分析に用いた変数は総欠陥数, 未調整FP, 開発種別(新規開発など), 業種(製造業など), 開発プラットフォーム(メインフレームなど)であり, 本研究では総欠陥数を目的変数とした. 開発種別などのカテゴリ変数はダミー変数化した. 予備分析において, OLSとAIC(Akaike Information Criterion)による変数選択を行い, モデルにおいて説明変数として用いる変数を絞り込んだ. 具体的には, 未調整FP, 新規開発, 金融業, メインフレーム, ミッドレンジを説明変数として採用した.

分析対象のプロジェクトの条件を整えるため, データ品質評価がAまたはB, FP計測法がIFPUG(International Function Point Users Group)のデータを抽出した. また, 分析で用いた変数に欠損が含まれているデータを除去した. その結果, 221件のプロジェクトが分析対象となった. 欠損が含まれているデータを除去することはリストワイズ除去と呼ばれ, 広く用いられており, 予測モデルの構築時に適用することが妥当であることが指摘されているため[6], これを採用した.

モデルの予測精度を評価するために, 5-fold cross validationによる実験を行い, 評価指標は, 絶対誤差(*AE*), Balanced Relative Error(*BRE*)それぞれの平均値, 中央値とPred25を用いた. *AE*, *BRE*は値が小さいほど精度が高いことを示す. Pred25は, 誤差25%以内のデータの割合を指し, 数値が高いほど精度が良いことを示す.

表1 各モデルの予測精度

	*AE*平均値	*AE*中央値	*BRE*平均値	*BRE*中央値	Pred25
OLS	10.74	4.11	284%	165%	12%
Tobit	9.66	2.04	378%	115%	8%
評価値	**1.08**	**2.06**	**-94%**	**49%**	**-4%**
OLS_ln	8.24	1.86	226%	107%	21%
Tobit_ln	8.45	1.53	260%	93%	28%
評価値	**-0.21**	**0.33**	**-34%**	**14%**	**7%**

実験結果を表1に示す．評価値は，OLSの各評価指標から，Tobitの各評価指標を引いた値である．ただし，Pred25のみTobitからOLSを引いた値とする．これはそれぞれの評価指標の評価値をみた際に正の値ならばTobitモデルの方が優れている（負の値ならばOLSの方が優れている）ということを分かりやすくするためである．実験結果では以下の傾向が見られた．

(1) OLSとTobitの比較：*AE*平均値，*AE*中央値，*BRE*中央値は改善し，*BRE*平均値，Pred25は悪化した．
(2) OLS_lnとTobit_lnの比較：*BRE*平均値，Pred25は大幅に改善し，*AE*平均値，*AE*中央値は大きな差がなかった．

（1）については，特に*BRE*中央値を見るとOLSと比べ100%近く悪化していた．また，Pred25をみると，誤差の大きなデータ件数もOLSより多かった．（2）については，*BRE*平均の評価値には大幅な改善がみられた．Pred25に関してはOLS_lnよりも高い精度となった．

これらにより，対数変換を適用しない場合のトービットモデルは*BRE*平均値とPred25に問題を残すが，対数変換後はこれらの指標を改善することが可能となり，有効性を示すことができたと考える．

4. おわりに

本研究では，ソフトウェア欠陥数の予測にトービットモデルを用いることを提案した． 実験結果では，対数変換を適用したトービットモデルは，対数変換を適用した最小二乗法に基づく重回帰モデルよりも5つある評価尺度のうち3つに対し予測精度が高いと評価された．このことから，対数変換を適用したトービットモデルは検討すべきモデルであるといえる．対数変換を適用したトービットモデルでは，最小二乗法に基づく重回帰モデルよりも*BRE*平均値が悪化する傾向があるが，その理由の分析と改善方法について，他のデータセットを用いて検討することは今後の課題である．

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 25330090）による助成を受けた．

参考文献

[1] International Software Benchmarking Standards Group, “ISBSG Estimating Benchmarking and Research Suite Release 9,” ISBSG (2004).
[2] Jeffery, R., Ruhe, M. and Wieczorek, I., “Using Public Domain Metrics to Estimate Software Development Effort,” Proc. International Software Metrics Symposium (METRICS), pp.16-27(2001).
[3] Mendes, E., Lokan, C., Harrison, R. and Triggs, C., “A Replicated Comparison of Cross-company and Within-company Effort Estimation Models using the ISBSG Database,” Proc. International Software Metrics Symposium (METRICS), p.36 (2005).
[4] 水落正明，“打ち切り・切断データの分析”，理論と方法（数理社会学会 機関誌），Vol. 24, No. 1, pp. 129-138, (2009)
[5] Sentas, P., Angelis, L., Stamelos, I. and Bleris, G., “Software productivity and effort prediction with ordinal regression,” Information and Software Technology, Vol.47, No.1, pp.17-29(2005).
[6] Strike, K., El Eman, K., and Madhavji, N., “Software Cost Estimation with Incomplete Data,” IEEE Transactions on Software Engineering, Vol.27, No.10, pp.890-908 (2001).

ソフトウェア工学に関する研究を発展させるためには

落水浩一郎
金沢工業大学　客員教授

要旨

「ソフトウェア工学は経験科学である」とする立場から，ソフトウェア工学に関する研究を発展させるための手段を，カール・ポパーの反証主義に基づいて検討する．

1. はじめに

ソフトウェア工学は経験科学である．経験科学とは「純粋に理論を探求する科学に対して，経験的事実を対象として実証的に諸法則を探求する科学（大辞林）」である．カール・ポパーの哲学，反証主義「客観的真理にせまるための試行錯誤は，問題の発生，試験的理論，誤りの排除，新たな問題の発生，という推測と反駁の過程の連鎖である」に基づいて考察する．なお，ソフトウェア作りとソフトウェア工学をここでは以下のように区別し，②のみを考察の対象とする．

① ソフトウェア作りは創作活動であり芸術活動の一種であろう
② ソフトウェア工学は，ソフトウェア開発・保守にあたっての労力を緩和するものであり，作成されるソフトウェアの品質を高めるためのものである．

2. 経験論（経験主義）の歴史

経験科学の方法論は長い歴史をもつ[1].

① フランシス・ベーコンによる帰納法：感覚的観察を無条件で信用せず，実験という方法を駆使して，少しずつ肯定的な法則命題へと上がっていく
② 論理実証主義：実験による調査研究，帰納的推論，演繹的検証
③ カール・ポパーによる反証主義

筆者は長い間，観察を基に，帰納法によって理論に到達するというベーコンの帰納法を漠然と意識してきたが，仮説を立てそれを実験に基づき反証することによって理論を進化させていくという立場のカール・ポパーの著書を読み，ごく自然な考え方であるということで彼の考え方にそって考察することとした．

3. 科学の課題

以下，３章および４章の内容はすべて，カール・ポパー著書の訳書[2，3，4，5]を引用することにより記述する．

カール・ポパーによれば，科学の課題は，一部は理論的なもの－**説明**－であり，一部は実際的なもの－**予測および技術的応用**－である．

説明とは図１に示すように，被説明項Ｅが与えられており説明項を求めることである．ここで，説明項は，普遍法則Ｕ（または，仮説，または理論）と特殊な初期条件Ｉからなる．被説明項とは観測された事実であり，複数の被説明項（反復しておこる同じ事実）からそれを説明する一般性（普遍性）を導出してよいとする立場が帰納法であり，カール・ポパーはこの立場を否定する．

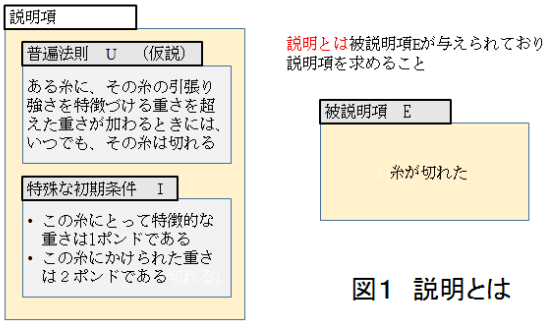


図１　説明とは

反証法においては，被説明項は仮説（または理論）のテストに使用される．すなわち，現実世界においておこる問題現象に対してそれを説明する仮説を導く，それを観測によって得られた事実（被説明項）とつきあわせて確認する．仮説に反する事実が観測された場合は仮説（または理論）か特殊初期条件が間違っているので，双方またはどちらかを作りなおす．このように，初期の仮説を観測によって確認し，反駁によって仮説（理論）を進化させていく立場が反証主義である．以下のような手順を踏む．

P1(出発点となる問題)→
TT(暫定的解決・暫定的理論)→
EE(誤りの排除)→
P2(より大きな深さをもち，豊沃性に富む，新しい問題).

説明には**理論的説明**と**歴史的説明**がある．理論家は普遍法則を見出し，テストすることに関心を持つ．歴史家は，特殊初期条件の適切さまたは正確さをテストする(図2)．

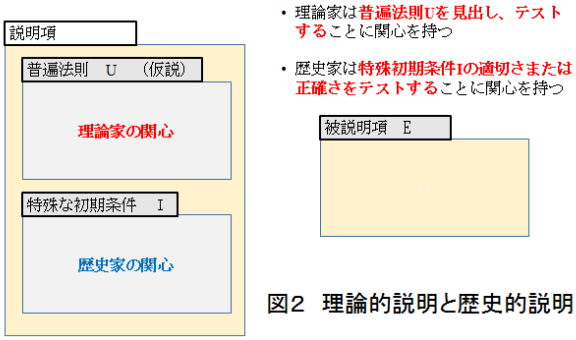


図2 理論的説明と歴史的説明

予測(図3)とは，理論Uが所与または既知であると仮定されており，また，特殊初期条件Iも観察によって知られているあるいは，知られ得ると仮定されている．見出されるべきものは論理的帰結，すなわち予測Eである．

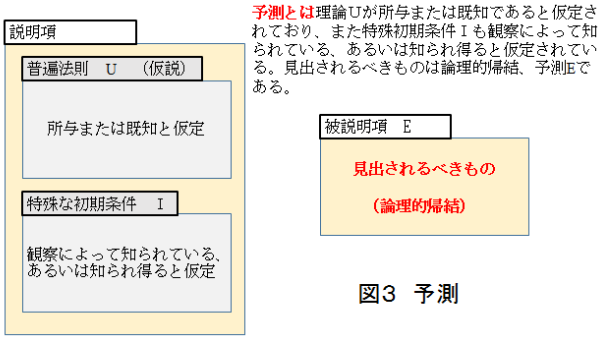


図3 予測

技術的応用(図4)とは，Eに相当する仕様明細書(顧客の仕様書)があり，これには一定の要求された事態が記述されている．ある種の実用的な目分量のやり方も含めて，関連のある物理学的諸理論Uも与えられている．われわれが見出さなければならないものは，技術的に実現でき，また理論と一緒になって仕様明細書Sが導出できるような初期条件Iである．

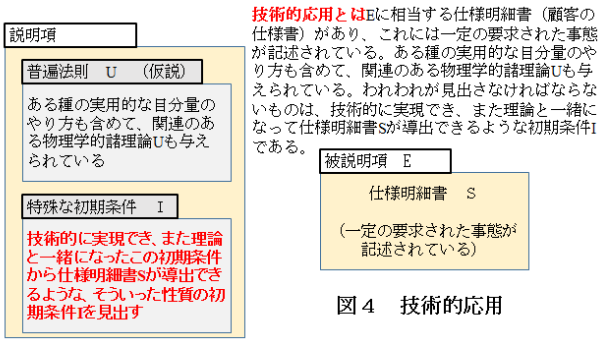


図4 技術的応用

4. テストの手続き

反証法においては，テストは重要な役割をもつ．テストの手続きは以下の通りである．

① テストとは，説明項からの予測 P の導出と，その予測を現実に観察可能な事態と比較することである．
② もし予測が観察された事実と一致しなければ，説明項は偽なることが立証される．つまり反証される．
③ この場合，反証されたのが普遍理論であるのか，それとも初期条件が偽であるのかはわからない
④ 予測の反証は説明項が偽であることを立証するが，しかしその逆は成り立たない．つまり予測の実証をもって説明項を実証するものと解釈できると考えるのは間違いである．

5. ソフトウェア工学への適用

3章で紹介した考え方をソフトウェア開発方法論に適用してみる．**説明**，**予測**，**技術的応用**のどれを基本的な考え方として採用すべきかがまず問題である．ここでは，**予測**をもとに考察する．**技術的応用**の適用については今後の課題とする．また，ソフトウェア工学の種々の分野のうち，ソフトウェア開発方法論を例としてとりあげる．

図５に示す図式に従って，仮説や特殊な初期条件の妥当性を検証する手段を検討する．仮説そのものの妥当性を議論したいのではないことに注意して欲しい．仮説には，方法論の効用としてわれわれが期待している事柄を記述し，初期条件には，

該当する方法論の適用条件を記述する．被説明項には，実際の開発活動の結果得られた事実を記述する（図５）．

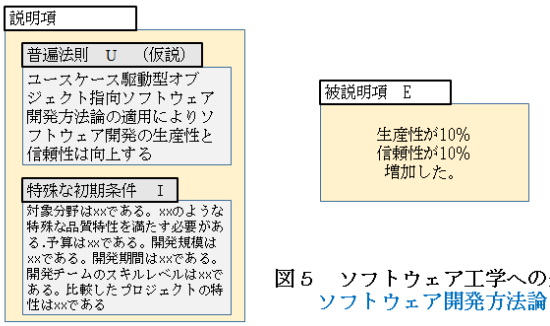


図５　ソフトウェア工学への適用
ソフトウェア開発方法論

予測においては，被説明項は説明項の論理的帰結でなければならない．ソフトウェア工学への適用の図式においては，説明項から被説明項を論理的に帰結するのは困難であるので，　論理的帰結（予測）は実際の開発活動の結果得られたものとする．

実際の活動は，仮説が成立する特殊な初期条件を精錬する活動になるものと思われる．

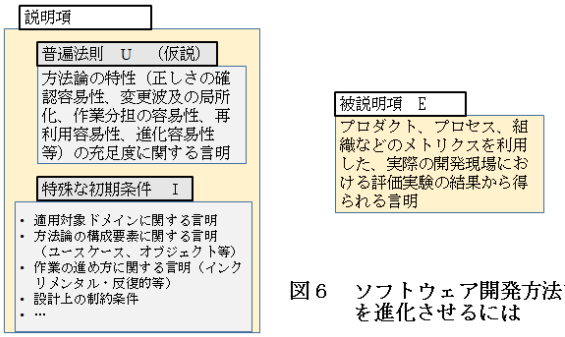


図６　ソフトウェア開発方法論
を進化させるには

図5の記述はひとつの考え方であるが，成果が特定の方法論・ドメインに依存しすぎるかもしれない．ソフトウェア開発方法論の発展のためには，**「ソフトウェア開発方法論の進化の手段を提供する」**という立場のモデル化がより重要であると考える．すなわち，現在利用中の方法論を改善する立場ではなく，よりよい方法論を求める立場である．図6に一つの考え方を示す．

もちろん図6の定義内容そのものは，いまだ十分なものとはいえず，今後の課題である．

6. まとめと課題

経験にもとづいて，皆にとって有益であるような法則性を発見するにはどのように思考すべきであろうか．「ちょっと考えてうまくいった，みんなにひろめよう」，「ちょっとやってみたらうまくいった，みんなにまねさせよう」はレベルの低いアプローチである．これに対して，カール・ポパーの哲学をソフトウェア工学研究の背景哲学として採用することは有益であると考える．すなわち，「ちょっと考えてみてうまくいった，確かめてみよう」，「ちょっとやってみたらうまくいった，本当にそうか確認してみよう」のように考え方を変えれば，誰かが定義した方法論の良し悪しを論議するだけの立場から脱却できる．すなわち，

①．　問題状況を解決する仮説を立てる(初期の説明項)
②．　初期条件を明らかにすることにより，理論の適用範囲を明確にし
③．　予測と実験による反駁を繰り返すことにより，よりよい理論に仕上げていく

ことが可能になる．よく考えてみると以上の事柄は皆様が日常行われていることではないかと推察する．それぞれが関心のある分野(形式手法，要求定義，設計，プログラム言語，プロジェクト管理，メトリクス等)で考え方(攻め方)を明示的に整理し，皆で共有することは有用であると考える．

このとき，テストは，組織や会社の枠組みを超えて実施する必要があろう．テストを実施するための場の提供と共有は，ソフトウェア技術者協会の一つの役割ではなかろうか．

参考文献

[1]　WikiPedi　「経験論」http://ja.wikipedia.org/wiki/**経験論**
[2]　カール・ポパー著，大内　義一・森　博　訳，「科学的発見の論理（上）」恒星社厚生閣，2010.
[3]　カール・ポパー著，大内　義一・森　博　訳，「科学的発見の論理（下）」恒星社厚生閣，2010.
[4]　カール・ポパー著，森　博　訳，「客観的知識　－進化論的アプローチー」，木鐸社，1999.
[5]　カール・ポパー著，藤本　隆志，石垣　壽郎，森　博　訳，「推論と反駁　科学的知識の発展」，法政大学出版局，2014.

AgileXDDP

八木 将計
株式会社日立製作所
masakazu.yagi.zd@hitachi.com

会田 圭司
テクマトリックス株式会社
aita@techmatrix.co.jp

石川 宏保
派生開発推進協議会 T6 研究会

斎藤 賢一
株式会社エクスモーション
kenichi.saito@exmotion.co.jp

永田 敦
ソニー株式会社
inagworld@gmail.com

星野 充史
アンリツエンジニアリング株式会社
Atsushi.Hoshino@anritsu.com

山田 謙輔
派生開発推進協議会 T6 研究会
yamaken1979@gmail.com

要旨

本報告では，広がりを見せている Agile 開発[1][2][3]と，派生開発手法である XDDP (eXtreme Derivative Development Process) [4][5]の二つの手法において，双方の特徴を組合せることで，それぞれの課題を解決する「AgileXDDP」を提案する．AgileXDDP は，Agile 開発に XDDP の考え方を導入したものと，XDDP に Agile 開発の考え方を導入したものの 2 つを提案する．

1. はじめに

近年，ソフトウェア開発において，スクラムなどの Agile 開発が一般的に広まっている[1][2][3]．Agile 開発の多くは，イテレーション(反復開発)することで，リスクを最小限にしつつ，迅速で柔軟な開発を実現する．そのため，自動テストなどの反復によるデグレードなどのトラブルを防止する仕組が含まれる．しかし，多くの反復を繰り返していくうちに，自動テスト等の仕組の保守が難しくなり，デグレードなどのトラブルが発生することがある．また，自動テスト等の仕組がないソフトウェアに機能追加をする場合などに，導入が難しくなりやすい．

一方，ソフトウェアの派生開発の手法として XDDP (eXtreme Derivative Development Process)が注目されてきている[4][5]．XDDP は，派生開発における変更点に注目し，それに特化したドキュメントを作成・レビューすることで，デグレード等のトラブルを防止する手法である．しかし，この変更に関するドキュメントは作成スキルが必要であり，担当者のスキルが未熟な場合，通常のレビュープロセスでは，ドキュメント作成で手戻りが発生するという課題がある．

派生開発推進協議会 T6 研究会では，「Agile 開発との連携」というテーマで，XDDP と Agile 開発の連携を研究している[6]．T6 研究会では，本報告にて，上記の Agile 開発での課題と XDDP での課題を解決する手法として，「AgileXDDP」を提案する．提案手法は，Agile 開発に XDDP の要素を持ち込んだ「機能追加の AgileXDDP」と XDDP に Agile 開発の要素を持ち込んだ「変更の AgileXDDP」の二つがある．

2. Agile と XDDP の課題

2.1. Agile 開発の課題

Agile 開発では，構成管理や継続的インテグレーションによる自動テストによって，通常よりもデグレードなどのトラブルは発生しづらいが，イテレーションを繰り返しいくなど，ソフトが増大化していくと，デグレードなど，一般的な派生開発と同じような問題が発生する．

また，Agile 開発は，開発の成果物が刻々と変化するような新規開発に適した手法であり，既に動いているものに機能追加するには，困難な場合がある．特に，仕様やテストケースが失われている場合，デグレードを防ぐのが難しく，適用が困難になる．

2.2. XDDP の課題

XDDP は，機能追加プロセスと変更プロセスを分離しており，特に変更プロセスに「変更 3 点セット(変更要求仕様書，TM，変更設計書)」というドキュメントを作成・レビュ

ーすることで，変更によるデグレードといった手戻りを防止する方法である．しかし，この変更 3 点セット，特に，変更要求仕様書に用いられる USDM (Universal Specification Describing Manner)[7]の記載にスキルが必要になる．

そのため，担当者のスキルが未熟な場合，変更 3 点セットが完成してからレビューを行うと手戻りが発生しやすいという課題がある．また，レビューで充分に精査できていない状況で開発を進めると，後工程での大きな手戻りになるとともに，担当者のスキル向上に繋らず，最終的に変更 3 点セットが形骸化する可能性もある．

3. 二つの「AgileXDDP」

3.1. 機能追加の AgileXDDP

2.1 節に示す Agile 開発の課題を解決するため，Agile 開発に XDDP の考え方・やり方を導入する．本報告では，主に機能追加プロセスに関する手法なので，本手法を「機能追加の AgileXDDP」と呼ぶ．

機能追加の AgileXDDP は，Agile 開発の一つの機能追加において，図 1，図 2 に示す通り，一度 Master から Feature ブランチを切り，機能を実現し，顧客等とレビューを行う．結果，よければそれを母体である Master に反映することになるが，その際に，XDDP の「移植」を用いる．

Feature ブランチを試作台として用いることで，Agile 開発特有のスピードを実現しつつ，XDDP の移植を用いることで Master に対する影響分析し，デグレードなどの手戻りの悪影響を防ぐ方法となる．一見，移植部分が無駄なように感じられるが，「移植」はそもそも別のソースコードから別のソースコードへ機能を持ってくる作業なので，Develop などで Master が違っていたとしても，その差による影響を最小化することができると考えられる．

機能追加の AgileXDDP では，Master が異なっていても対応可能になるため，図 3，図 4 のように一部の作業を並行にする応用も可能になると考えられる．また，Master への悪影響を最小化できるので，複数チームで同一の Master に対して機能追加することも可能となると考えられる．

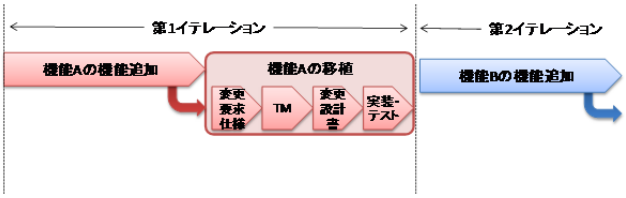


図 1 機能追加の AgilXDDP

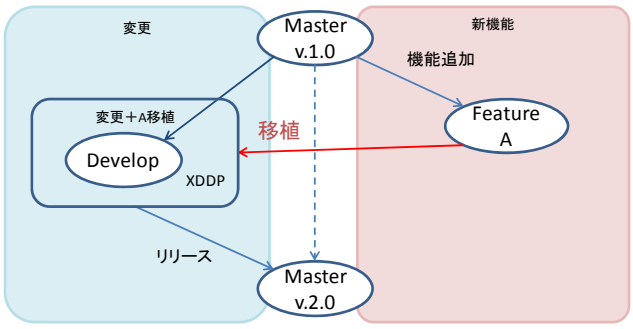


図 2 機能追加の AgileXDDP の構成管理

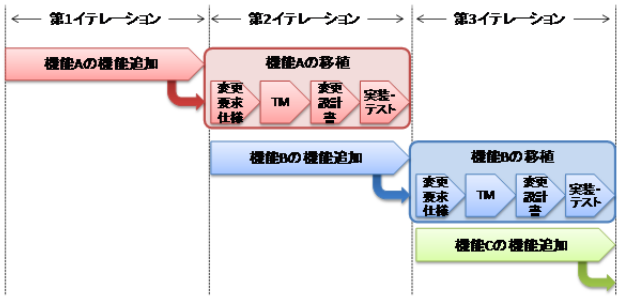


図 3 機能追加の AgileXDDP ver.2

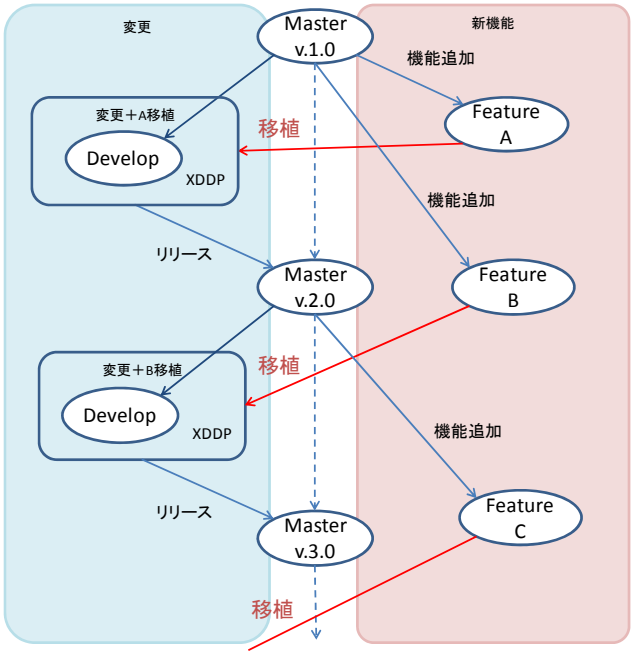


図 4 機能追加の AgileXDDP ver.2 の構成管理

3.2. 変更の AgileXDDP

2.2 節に示す XDDP の課題を解決するため，XDDP に Agile 開発の考え方・やり方を導入する．本報告では，主に変更プロセスに関する手法なので，本手法を「変更の AgileXDDP」と呼ぶ．

変更の AgileXDDP は，XDDP の変更プロセスの変更 3 点セットの作成において，タイムボックスを用いて，イテレーティブに成果物を作成する．特に，変更要求仕様書である USDM では，記載技術が必要になるので，「アジャイルインスペクション[1]」を用いて，担当者のスキル向上を含める．また，このアジャイルインスペクションを用いることで，全ての仕様を記述してからレビューするのではなく，要求定義の早期の段階から要求者へフィードバックすることで手戻り防止の効果を狙う．

この変更の AgileXDDP を用いることで，変更 3 点セットの作成技術を醸成しながら，後工程での手戻りも防ぐことができると考えられる．

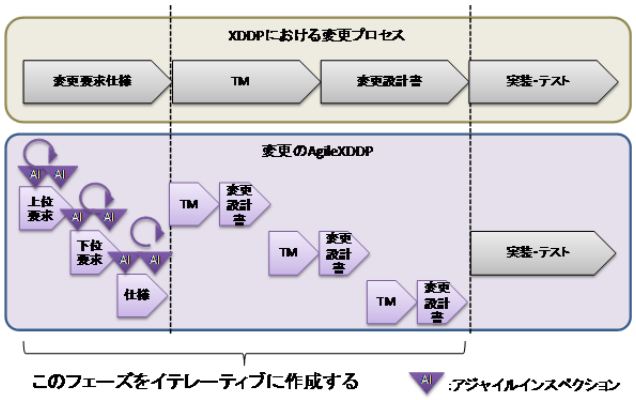


図 5 変更の AgileXDDP

4. 今後の課題

本報告で示した，二つの AgileXDDP は，まだ構想の段階であり，実際に適用することが今後の一番大きな課題となる．

また，この二つの AgileXDDP は，それぞれ機能追加と変更に注目しており，両方をハイブリッドすることも可能になると考えている．そちらの検討も進めていきたい．

[1] アジャイルインスペクションとは，ドキュメント作成途中でサンプリング，短いタイムボックスでレビューし，決められた品質になるまで修正／サンプリング／レビューを繰り返すことで，ドキュメントと書き手の品質レベルを向上する手法である[8][9].

5. まとめ

本報告では，派生開発推進協議会 T6 研究会の研究成果として，Agile 開発と XDDP の特徴を活かした二種類の「AgileXDDP」を提案した．まだ，実証ができていない方法ではあるが，この二つの AgileXDDP により Agile 開発，XDDP 双方における課題が解決できることを期待している．

参考文献

[1] Beck, et. al., Manifesto for Agile Software Development. http://www.agilemanifesto.org/
[2] 独立行政法人情報処理推進機構，アジャイル型開発におけるプラクティス活用事例調査 調査報告書，東京，2013.
[3] 平鍋健児，野中郁次郎，アジャイル開発とスクラム，翔泳社，2013.
[4] 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，2007.
[5] K. Kobata, E. Nakai and T. Tsuda, “Process Improvement Using XDDP: Application of XDDP to the Car Navigation System,” Proc. 5th World Congress for Software Quality, (Nov. 2011).
[6] 派生開発推進協議会 AFFORDD http://www.affordd.jp/
[7] 清水吉男，改訂第 2 版 [入門+実践]要求を仕様化する技術・表現する技術～仕様が書けていますか？，技術評論社，2010.
[8] T.Gilb, “Inspection Facts: To Help You Make Good Decisions to Do Software Inspections and Reviewers Properly,” ソフトウェア品質シンポジウム 2008 (SQiP2008), 2008.
[9] 永田敦，森崎修司，“アジャイルインスペクションの実際，” ソフトウェアテストシンポジウム 2010 (JaSST’10 Tokyo), 2010.

クラウド・ビッグデータとソフトウェアエンジニアリング

鯵坂恒夫
和歌山大学
ajisaka@sys.wakayama-u.ac.jp

要旨

クラウド・ビッグデータと高頻度で抱き合わせにされるこれら二つのバズワードについて，その本質をさぐりながら，ソフトウェアエンジニアのとるべき姿勢と態度について議論する．

1. はじめに

クラウドとビッグデータは，今後も当面は情報システム技術のトレンドであろうということが共通点であり，現象としても共起する傾向にはあるだろうが，対応すべき技術的内容は必ずしも相似していない．したがって，議論展開は実はオムニバス的になる．

2. クラウドによる変化

利用者と開発者のギャップはこれまでも様々に言及されてきたが，クラウドは両者のものの見方をさらに分離する．利用者は使いたい機能を使いたいときだけというスタンスになるのに対して，開発者は情報統合の視点を強化しなければならなくなる．利用者側について運用技術を支援する役割と，クラウドシステム自体を開発する役割に明確に分かれる．これまでソフトウェアエンジニアが一人二役で両方を担ってきたが，頭の使い方やネゴシエーションのしかたが相当に違うので，担当者が分かれる方向に向かうと予想され，それによって利用者・開発者ギャップ問題が思わず解決するのかもしれない．

利用者側の技術も開発者側の技術もこれまで以上に業務の内容とプロセスに密着することになる．利用側は業務とシステムとのマッチング（機能選択と連結）である．エンジニアは対象業務のビジネスモデルやワークフローをよく理解していなければならない．エンドユーザコンピューティングの指南役も求められるかもしれない．一方，開発側のほうは，情報系クラウドが主流の現状ではその業務結合性は明確でないかもしれないが，基幹系クラウドがこれから進展してくれば，情報の要素認識と意味的関係性の把握が重要になってくる．単体業務ではなくグッズとサービスのチェインマネジメントを見通していなければならず，それによる規模と複雑さ・不確定性の拡大に対抗する手だてが肝要となる．

空想的にはクラウドがどんどん結着して「ザ・情報システム」がひとつあればよい．組織活動にはすべからく inbound logistics と outbound logistics がある（cf. Michael Porter の Value Chain, 1985）ので，その鏡像関係がクラウド内で縮退すれば，全体としての規模と複雑さは半減する．問題は不確定性である．規格化に人々が反抗するということではない．社会の進歩により概念要素や規則が変化するということである．ソフトウェアの問題解決はパターン化しかないが，進歩によって生じるバリエーションを抑圧することはできない．この二律背反を解消するのはバリエーションの生え方をパターン化する技である．パターンは還元論的であり，バリエーションは全体論的であるので，これら不倶戴天の両者を協調させる超絶技巧が求められる．

ザ・情報システムが空想であるとすれば，空想から科学へ移行するには何が必要かを考えることができる．ひとつ明らかなことに，ザ・情報システムは一日にしては成らないので，そこに至るステップはいかにあるべきかという課題がある．あるいは IoT がそこへ向かうのを加速するかもしれない．何か多くの局面で有用な thing がネット化されれば，それを共有しようといろいろなサブシステムが寄り集まり融合をはじめる．

3. ビッグデータというブルートフォース

クラウドはソフトウェアエンジニアリングや情報システム技術のコンテクストから出てきた由緒正しいトレンドだが，ビッグデータは必ずしもそうではない．クラウド化すれば扱うデータ量も巨大化するので，ビジネスデータという限られたジャンルのビッグデータが連想されるというだけである．ビッグデータはむしろ WWW やセンサネットワーク，IoT から溢れ出てくるものが主流であって，その処理のさ

れかたはかなり武骨である. ビッグさは半端でないので, 精緻な扱いはもとよりできない. 何種類かの特徴値の組合せを分類するだけである. 言語処理になぞらえていえば, 構文解析以上の高級な処理は一切なし, 字句解析のレベルのみという, すでにインターネット検索で成功した手法である.

そうして得られた結果(予測)には理由がない. Singularity (cf. Kurzweil, 1999,2005)をもたらすという「新型人工知能」はこの方式である. これはコンピュータの本分をわきまえたという点では妥当であり, それによってかつての(人間を模倣しようとした)人工知能の失敗を乗り越えている, というより別物である. そこで人間が why の追求をやめるか続けるか, いいかえれば「知性」の定義そのものを変えるか変えないかによって, 人間の尊厳の維持・存続が決まる.

ビッグデータの処理技術は特徴値の種類の選択と各特徴値の閾値の設定にかかっている. さらにソフトウェアがそれらを動的に最適化できることが必要だと思われるが, こういうメタサーキュラーなしくみはなかなかうまく動かない. したがって, 得られた予測を見た人間の対応行動が早ければ(例えば金融商品売買や混雑状況対応など), 次の予測は狂うか大きく変動して, 意思決定の拠りどころにならなくなる. 新型人工知能に人間の理解する意味に基づいた調整を混ぜると, おそらくそれ本来の威力を発揮しなくなるので, 未来予測が際限なく進歩することはないと思いたい.

4. おわりに

クラウド・ビッグデータとソフトウェアエンジニアリングという標題を受けた結論は何かといえば, 不確定性 (uncertainty) への対応強化である. これまでソフトウェア工学論文の前振り定番であった大規模化・複雑化は, 分析・設計手法の進歩とプラットフォーム・フレームワーク・パッケージ化技術の進展により一定の対応がなされてきた. ソフトウェアの生産性と品質を向上させる決定打は, できるだけソフトウェアを作らない(再利用する)ことであるという信条が浸透してきた. しかし, これまでも保守の問題の多くが単純な修正保守より適応保守・改善保守の要求であろうと思われ, クラウド化によりますますその傾向は強まる. このように不確定性の増えた状況でも, それに対応していちいちソフトウェアを作らないですますには, 先述のとおりバリエーションのパターン化という形容矛盾を実現しなければならない.

これはおよそ人間の活動とはどのようなものかという, 神の発するような質問にこたえようとするものである. 行動観察のようなボトムアップ的手法や, Luhmann の社会システム理論, Latour の Actor-Network Theory, Vygotsky に始まる Activity Theory, Bertalanffy の一般システム理論など, 社会学的・哲学的論考もあるが, どれもまだ大成功しているようにはみえない. しかし, 相手にしているのが全体論的枠組みであるから, ゲーデルの不完全性定理のような明解な否定的解決も確立するとは思えず, 挑戦を続けるしかない.

上記「社会学的・哲学的論考」に関する補足:筆者も原書をあたってきたわけではありません, ネットワーク経由でかじったものです. ニクラス・ルーマンについては最も日本語の情報や訳書が豊富ですので, 容易にいろいろ参照できます. Actor-Network Theory と Activity Theory については英語版 wikipedia がしっかり書けていると思われます. ベルタランフィについては,「松岡正剛の千夜千冊」(http://1000ya.isis.ne.jp/0521.html)からあたり始めるのも一興かもしれません.