Vol.2015-SE-187 No.5
2015/3/12

# 開発履歴による開発者特性とアンケートによる特性の自己診断の関連分析

五田 篤志[1]　山崎 尚[2]　玉田 春昭[2]　畑 秀明[3]　角田 雅照[4]　井垣 宏[5]

**概要**：開発者特性を定量的に分析するために，我々は開発履歴を元にソフトウェア開発者の能力分類を行う手法を提案した．この手法によると，おおよそではあるが開発者特性を分析出来た．ただし，この手法は開発終了後の分析に限る．開発者特性を活かしてチーム構築をするためには，事前に開発者特性を把握することが望ましい．そこで，本稿は事前に開発者特性を特定するために調査を行った．調査内容は，1. 事前アンケートから開発者特性を特定できるか．2. 開発後に収集したメトリクスと事後アンケート結果は関係するか．3. 開発者特性が成果物の品質に与える影響は何か．である．

## 1. はじめに

ソフトウェア開発は，複数の開発者がそれぞれの役割を遂行し，かつ，互いにコミュニケーションを取らなければならない．また，開発者は人間であるため，開発において，得意なタスクと，不得意なタスクがあるのは当然である．そのため，開発者の得意分野がわかっていれば，より効率的なチーム編成が可能となる．一般的には，互いに得意な分野が異なっていれば，開発者が相互にタスクを補い合えるため，理想的なチームの 1 つとなる．また，一方で，プロトタイプ開発や，リファクタリング開発など，特定の目的の開発においては，その目的に合う開発者を割り当てたい．このことから，開発者の得意・不得意を踏まえたチーム編成が行えると，より良い開発に繋がると考えられる．

我々は先行研究として，開発履歴を利用した風林火山モデルに基づく開発者の特性分析 [1] を行った．これは，ソフトウェア開発終了後にリポジトリやチケット管理システムに蓄積された開発者個人の履歴から，小野が提案する風林火山分類 [2] で開発者の特性を特定するフレームワークの提案と，その妥当性を検証したものである．この研究では，2013 年度に実施された教育プログラムである CloudSpiral で実施されたプロジェクトでの開発履歴を対象に実施した．そして，開発履歴から収集したメトリクスに基づいて，開発者の特性を特定できることが確認できた．

先行研究では，おおよそ開発者特性を特定出来るため，チーム編成に役立てられると考えられる．一方で，プロジェクトの開始時には開発者特性を知ることが出来ない側面を持ち，一度しか実施しないプロジェクトメンバーやスタートアップのようなプロジェクトには適用出来ない．このような状況でのチーム編成には，事前に開発者特性を知ることが望ましい．よって，事前に開発者特性を特定することが課題である．また，風林火山への割り当てが第一著者によるものであったため，根拠を改善することも課題である．

そこで本稿は，プロジェクトの開始前に実施する事前アンケート (表 4) から開発者特性を特定出来るか，2014 年度の CloudSpiral の受講生である大学院生を対象に行ったプロジェクトを対象とし調査した．本稿では，事前アンケートは先行研究に従い開発者特性として風林火山分類を用いた内容とした．プロジェクト後に開発履歴から収集した開発者特性と同者のアンケート結果を比較し，同じ傾向が見られた場合は，事前アンケートによって開発者特性が特定できると考えられる．

また，事後アンケートによって風林火山分類に対するメトリクスの割り当てを見直すために，同様に比較した．これは先行研究では割り当てが第一著者の主観によるものであったが，開発終了後の当事者の主観も取り入れることで，割り当ての根拠を改善する試みである．

[1] 京都産業大学大学院 先端情報学専攻
Division of Frontier Informatics, Graduate school of Kyoto Sangyo University , Kyoto, Japan
[2] 京都産業大学コンピュータ理工学部
Faculty of Computer Science and Engineering, Kyoto Sangyo University, Kyoto, Japan
[3] 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan
[4] 近畿大学 理工学部
Department of Informatics, Kindai University, Osaka, Japan
[5] 大阪大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Osaka, Japan

## 2. 関連研究

角田らは，ソフトウェア開発において人間的側面が与える影響についての調査を行った [3]．調査は開発メンバのパーソナリティ（性格）に特に着目して行われた．調査結果によると，ソフトウェア開発において，チームメンバの性格は無視できない要因であると述べられている．ただし，(1) 調査方法がソフトウェア工学に特化していないこと，(2) アンケートでの調査であることの 2 点が課題として提示されている．(1) は，他の分野に応用できる反面，チーム編成のためには，より効率的な方法の可能性がある．(2) は，客観的指標に基づいてチーム編成を行いたいにもかかわらず，アンケートという個人の主観に基づいた指標を元にしている点が問題である．

Onoue らは，アクティブなソフトウェア開発において，どのようなタイプの開発者がいるのかを調査した [4]．オープンソースソフトウェア開発での開発者の活動から能力診断をすることに注目し，GitHub の node[*1] と jQuery[*2] プロジェクトを対象に調査を行った．その結果，それぞれの開発者の能力によって行動が異なることが明らかになった．

## 3. 準備

### 3.1 風林火山分類

本稿では具体的なメトリクスの集約方法について，1 つの方法として小野が提案している開発者の風林火山分類に着目する [2]．このモデルは開発者には，風林火山の 4 属性に分けられた能力がそれぞれ必要であるという，能力の分類法である．それぞれの属性は表 1 に示す特徴を持つ．風は実装能力を表し，林はプロジェクト管理能力を表している．火はプロダクトとプロセスの価値を向上させる能力，そして，山はプロダクトの品質を向上させる能力を表している．どれもが開発者にとって必要な能力であるため，4 つの属性を兼ね備える開発者が理想的な開発者であると言える．

また，この風林火山は，追加されること，削除されることもないと考えられる．例えば，[5] では，カテゴリが職業で表されているため，新たなカテゴリを追加することは非常に容易である．しかし，風林火山は 4 つのカテゴリに限定されており，カテゴリ数は固定される．

そこで，本稿は，第 2 の問題点を解決するため，開発データを使って，風林火山モデルによる開発者の能力測定手法を提案する．

## 4. ケーススタディ

### 4.1 対象プロジェクトの概要

本稿で対象とするプロジェクトについて述べる．本稿で対象とする開発プロジェクトは，大学院生を対象とした教育プログラムの中で行われた開発プロジェクトである．この教育プログラムは，神戸大学と大阪大学が中心となって行っている．そして，神戸大学，大阪大学に加え，7 大学の大学院 1 年次生を対象に，以下の 3 点を考慮して開発されている [6]．

- クラウド技術の本質を理解し，活用できること．
- ソフトウェア工学に基づいた PBL を行うこと．
- リアリティのある題材で学習すること．

この教育プログラムにおいて，1 週間の間に集中的に開発を行うプロジェクトを実施した．仕様は与えられ，1 チーム 5，6 人体制で，Web アプリケーションを開発するものである．このプロジェクトを本稿で得られた開発データの分析対象とした．

表 1　風林火山モデル

| 分類 | 特徴 |
|---|---|
| 風 | 迅速な設計・実装によってチームを加速させる． |
| 林 | 突発的なトラブルに冷静に対処し，チームに乱れぬペースを提供する． |
| 火 | 新しい技術・方法・ツールの積極的な導入によって，チームの成果物の競争力を高める． |
| 山 | 厳密なエラー・チェックと堅牢なプログラミングによって成果物の安定性を高める． |

### 4.2 対象プロジェクトの進め方

対象プロジェクトは，Scrum[7] を適用したアジャイル開発を行った [8]．Scrum では，スプリントと呼ばれる短い期間に，１つのまとまった単位の開発を行う．そして，スプリントの最初に計画を立て，最後に振り返りを行う．本来の Scrum であれば，1 スプリントは 1 週から 4 週程度であるが，対象プロジェクトは 4 日間の開発であったため，1 日を 1 スプリントとしている．ただし，この開発合宿に先立ち，１日だけ練習としてスプリントを実行している．そのため，合計 5 スプリントを実行したことになる．

各チームは，いずれのスプリントにおいても，最初に開発するコンポーネントを決め，見積もり後，役割分担の上開発に臨む．役割は，プロダクトオーナー 1 名，スクラムマスター 1 名，そして残りのメンバが開発者の計 3 役である．スクラムマスターはチームが円滑に開発できることを最優先事項としているため，必ずしも開発を行うわけではない．

また，開発を進める上で，Trac と Subversion を使った TiDD[9] を採用しており，各チケットに，見積もり時間，開始・終了時刻，所要時間を記録した．チケットにはそのチケットの性格を表すチケットタイプが付随している．チケットタイプは，ソースコード作成 (Java)，ソースコード作成 (HTML，JavaScript)，テストコード作成，レビュー，結合テスト作成，結合テスト実施，バグ修正の 7 種類に分

[*1] https://github.com/joyent/node
[*2] https://github.com/jquery/jquery

けられる．

各チームに CI ツールとして，Jenkins[*3] を用意しており，Subversion へのコミット毎に Jenkins でのフルビルドが行われるようになっている．

### 4.3 対象プロジェクトの教育上の制約

対象プロジェクトにおいては，教育上の観点から様々な制約を設けている．制約がなければ，ソースコード編集に多くの時間が費やされ，目的の１つであるソフトウェア工学に基づいた PBL が達成できない．同じく，制約がなければ，徹夜での作業を行うチームも起こり得る．更に，少数のプログラムを得意とする学生だけがアプリケーションを開発し，そうでない学生が何もしない時間を持つことも考えられる．このような理由から，対象プロジェクトにおいては，時間的な制約とタスク割り当ての制約を設けた．

時間的な制約は，開発時間は午前 10 時半から，午後 6 時までと決めたことである．この時間以外には打ち合わせしか認められていない．また，タスク割り当ての制約は，アサインメント制約と呼んでおり，次の通りである．

A. 開発の期間中ですべての役割 (プロダクトオーナー，スクラムマスター，開発者) を担当すること．
B. ソースコード作成（Java），ソースコード作成（HTML，JavaScript），テストコード作成，レビューの各メンバの担当数がチーム平均値の 0.8 倍から 1.2 倍以内に収まること．

このアサインメント制約はチームの自己組織化や個人の能力の向上，また，チームとしての成長を狙ったものである．例えば，スクラムマスターはチームを健全に保つという，Scrum を実践する上で不可欠な要素である．また，プロダクトオーナーも，チームに明示的な支持をせず，チームの方向性を決めるという，非常に難しい役割が求められている．そこで，短い期間ではあるが，実際に行ってみることで，それぞれの学生が，自らの感覚でそれぞれの役割を掴むことを狙っている．つまり，同じチームで 5 スプリントを実施するため，各スプリントでスクラムマスターとプロダクトオーナーが異なることになる．

### 4.4 収集したメトリクス

本稿ではソフトウェア開発のログに基づいて開発者の特性を定量化する．その対象は 4.1 節で述べたように，大学院生を対象とした教育プログラム中に行われた開発プロジェクトとする．2014 年度のプロジェクトからメトリクスを収集した．

対象プロジェクトでは, Trac, Subversion, Jenkins, Eclipse を利用しており，すべて開発ログを収集している．また，一方で，スプリント終了後，アンケートに回答してもらい，スクラムマスターの評価も行っている．そこで，表 2 に示す 8 つのメトリクスが，個人毎に収集できるようになっている．

スクラムマスターランクは，各スプリントにメトリクス値が導出される．しかし，各スプリントにスクラムマスターが異なる．そのため，当該スプリントでのスクラムマスターを担当した個人のメトリクス値とする．

メトリクスを収集出来る開発状況の補足として, 実際の開発で行われていた Trac を用いたチケット管理と Subversion をリポジトリとして利用する実装段階の概略図を図 1 及び図 2 に示す．また，実装段階では以下のタスクについてルールが規定されていた．

タスクの種類は次の通りである．

(1) 作成 (ソースコード)
(2) 作成 (単体テスト)
(3) レビュー
(4) 作成 (結合テスト)
(5) 結合テスト

一方，タスクに規定されていたルールは次の通りである．

(1) すべての実装は図 2 に従ってタスクが実施されなければならない．
(2) 作成 (ソースコード)or バグ修正 (ソースコード) →レビューの順に連続して実施されたタスクにおいて，各タスクの担当者は異なっていなければならない．
(3) 作成 (単体テスト)or バグ修正 (単体テスト) →レビューの順に連続して実施されたタスクにおいて，各タスクの担当者は異なっていなければならない．
(4) 作成 (ソースコード)or バグ修正 (ソースコード) 完了時には，コンパイル可能でかつすべての単体テストが正常に通る状態で成果物をリポジトリにコミットすること．
(5) 作成 (単体テスト)or バグ修正 (単体テスト) 完了時には，コンパイル可能な状態で成果物をリポジトリにコミットすること．(必ずしも単体テストが正常に通る状態である必要はない)
(6) すべてのタスクのすべての成果物をタスクの完了時にリポジトリにコミットすること (フォルダ構成やファイル形式は別途定める)

### 4.5 風林火山モデルへの提案フレームワークの適用

風林火山モデルに，提案フレームワークを適用する模式図を図 3 に示す．図 3 に示す通り，まず，利用する開発ツールやアンケートから，様々なメトリクスを収集する．次に，それらを風林火山モデルのそれぞれの属性に割り当てる．

どのメトリクスをどの属性に割り当てるか，割り当て後，各属性にどのように計測結果を導出するかは，プロジェクトに最適化させる必要がある．本稿においては，どのメト

[*3] http://jenkins-ci.org/

表 2　収集したメトリクスと収集方法

| ID | メトリクス | 概要 | 取得方法 |
|---|---|---|---|
| M1 | スクラムマスターランク | メンバからのスクラムマスターとしての評価 (５段階) の平均 | アンケート |
| M2 | 個人の総実装行数の割合 | 個人の総実装行数/チームの総実装行数 | Subversion |
| M3 | チケット対応コミットの割合 | チケットとコミットが対応しているコミット数/全コミット数 | Trac |
| M4 | レビュー回数 | チケットタイプがレビューであるチケットの解決数 | Trac |
| M5 | レビュー時間／回 | レビュー時間/レビュー回数 | Trac |
| M6 | バグ修正回数 | チケットタイプがバグであるチケットの解決数 | Trac |
| M7 | バグ修正時間／回 | バグ修正時間/バグ修正回数 | Trac |
| M8 | Jenkins ビルド成功率 | Jenkins でのビルド成功数/Jenkins でのビルド数 | Jenkins |

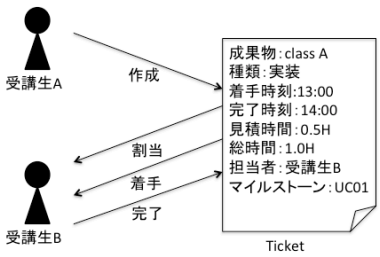


図 1　チケット管理の概要図

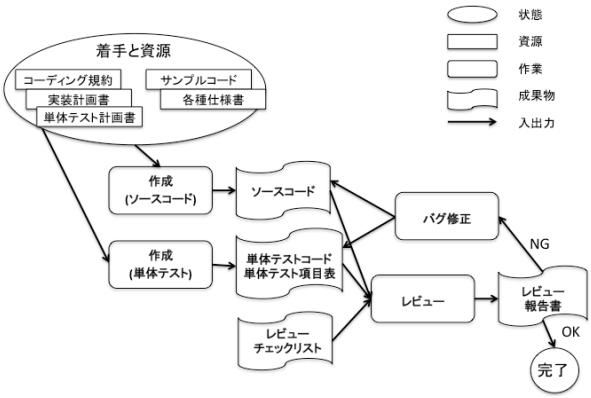


図 2　実装段階のフロー図

表 3　メトリクスの風林火山モデルへの割り当て

| 分類 | メトリクス |
|---|---|
| 風 | 個人の総実装行数の割合 |
| 林 | チケット対応コミットの割合, バグ修正回数, バグ修正時間／回 |
| 火 | スクラムマスターランク |
| 山 | レビュー回数, レビュー時間／回, Jenkins ビルド成功率 |

リクスをどの属性に割り当てるかは，第一著者の主観により決めた．

先行研究では，風林火山モデルの妥当性を確認するために，メトリクスを風林火山に割り当て，各開発者の風林火山値を算出した．しかし，本稿では，第一著者の主観によって決めていた割り当てを検証することに着目するため，割り当てのみを実施した．

### 4.6　メトリクスの風林火山モデルへの対応付け

本節では，収集したメトリクスそれぞれを，風林火山で表した能力モデルに対応付ける．これにより，風林火山のそれぞれの値が算出できるようになる．なお，対応付けには，以下の点を考慮して筆者の主観により割り当てた．割り当てを表 3 に一覧で示す．

風には，迅速な設計・実装という風の特徴から，開発速度や納期に関連するメトリクスを割り当てる．具体的には，個人の総実装行数の割合である．林は，チームに乱れぬペースを提供することから，チームがしっかりと運営されているか，決められたルールを遵守しているかを表すメトリクスを割り当てる．そのため，チケット対応コミットの割合，バグ修正回数，バグ修正時間の 3 つのメトリクスを割り当てた．火は，チームの成果物の競争力を高めることから，より良い成果物を生み出すため，チームのタスク遂行力を高めるメトリクスを割り当てる．具体的にはスクラムマスターランクを割り当てた．山は，成果物の安定性を高めることから，成果物の品質を高めるために必要なメトリクスとした．品質向上のために行うレビュー関連のメトリクスとビルド成功率を割り当てた．

## 5.　リサーチクエスチョン

### 5.1　概要

先行研究での課題は，開発者特性を算出するために開発履歴が必須であることである．そのため，開発履歴が存在しない状態で，開発者特性を算出することは不可能である．このことから，例えば，授業に於いて，開発者能力が

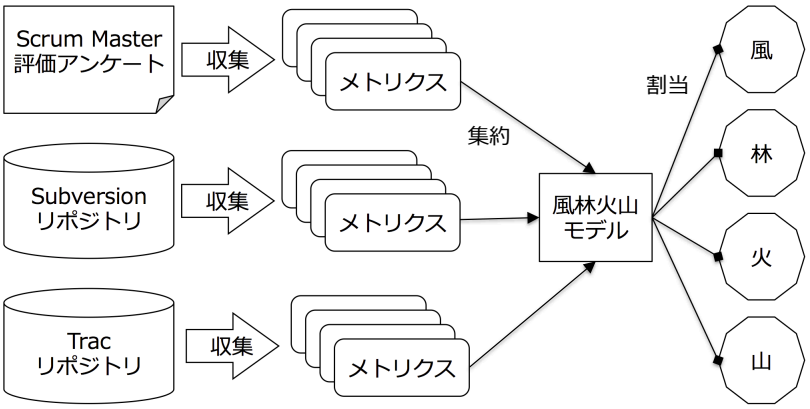


**図 3** 風林火山の概観図

均等になるようなチーム編成は行えない．そこで，事前に開発者特性を何らかの方法で導出できれば，その特性値に従ったチーム編成が行える．そして，開発を通して得られたメトリクスを特性値にフィードバックし，特性値の改善に繋がっていくと考えられる．そこで，開発者特性に関わるアンケートをプロジェクト開始前と終了後に実施し，アンケートが開発者特性を知る上で有意かどうかを調べることにした．さらに，事後アンケートを利用して開発者特性が成果物に与える影響を分析した．これらは以下の３つのリサーチクエスチョンである．

**RQ1** プロジェクト開始前に事前アンケートを用いることで開発者特性を知ることができるのか．

**RQ2** 開発者特性の風林火山への割り当ての妥当性を事後アンケートで評価できるか．

**RQ3** 開発者特性が成果物の品質にどのような影響を与えるのか．

RQ1 の意義は以下の通りである．本研究の目的は開発者特性を知ることで合理的なチーム構築を提案することである．一方で，本手法は開発履歴を用いた開発者特性の分析を行っているため，事前に合理的なチーム構築を提案することは困難である．1 つの事前に開発者特性を知る方法として，プロジェクト開始前に風林火山分類による開発者の能力を知るためのアンケートを実施することがある．ただし，このアンケートが示す結果は開発履歴と比較して人の思考に頼るため定性的であるため必ずしも本人の能力と直結しない．そこで，プロジェクトの結果として定量的な値で示せる開発者特性のメトリクスとアンケートの結果を比較し，どの程度事前アンケートが開発者特性を事前に知る上で有意 ($p < 0.05$) であるかを調査する．

RQ2 の意義は以下の通りである．事後アンケートは被験者らがプロジェクトの経験を踏まえた上での自己評価であるため，プロジェクト終了後の開発履歴から測定した開発者特性のメトリクスと相関を示すことが望ましい．そこで本稿の風林火山分離へのメトリクスの割り当てが妥当かの確認がとれる．

RQ3 は，RQ2 で得られた結果をもとに開発者特性が成果物の品質に与える影響を調査する．

アンケートでは，開発者が自分自身の特性をどのように理解しているのかを調査した．開発者それぞれに，アンケートを用いて，風林火山のそれぞれの特性を５段階評価で回答してもらった．また，それぞれの特性を目指したいかを５段階で回答してもらった．なお，アンケートの対象者は，2014 年度の CloudSpiral 受講生である．

### 5.2 RQ1 プロジェクト開始前に事前アンケートを用いることで開発者特性を知ることができるのか

開発者単位で収集した風林火山分類のアンケート結果と，開発者特性を相関分析し，こちらが想定している風林火山を示す特性とどれほどの相関があるかを調査する．強い正の相関を示した場合は，事前に開発者特性を知る上でアンケートは有意である可能性がある．弱い正の相関を示した場合は，ある程度アンケートで開発者特性を知ることができるとする．ただし，相関が弱かったことにはなんらかの原因が考えられる．例えば，開発者が考える風林火山の基準 (メトリクスの値やメトリクスそのもの) が本稿で抽出しているものでは十分で無いことや，まったく異なっている原因などが考えられる．十分で無いと仮定した場合は，他のメトリクスを風林火山の属性に併用 (集約) させることで精度を高められる可能性はある．この章では，問題点を単純化するために敢えて開発者特性を集約する前の段階で相関分析を行った．

相関分析の結果を表 4 に示す．有意な相関結果には網掛けを施している．表の縦には，開発者特性とこちらが想定している風林火山分類への集約が示されていて，表の横には，アンケートの項目が示されている．wind とは自分を風のエンジニアであるかどうかを 1 から 5 段階で評価したもので，数値が高いほど自分は風であるという評価である．want_wind は風のエンジニアになりたいかどうかの結果で 1〜5 段階評価したもので，数値が高いほど強くそう思っていることを示している．

**表 4** 事前アンケートと開発者特性の相関

| | wind | want_wind | woods | want_woods | fire | want_fire | mountain | want_mountain |
|---|---|---|---|---|---|---|---|---|
| 火 スクラムマスターランク | 0.224 | 0.025 | -0.081 | -0.053 | 0.167 | 0.115 | 0.175 | 0.129 |
| 風 個人の総実装行数の割合 | 0.205 | 0.078 | -0.276 | -0.011 | 0.073 | 0.033 | 0.095 | 0.175 |
| 風 総実装行数 | 0.186 | 0.086 | -0.285 | -0.080 | 0.028 | 0.031 | 0.103 | 0.131 |
| 林 チケット対応コミットの割合 | -0.039 | -0.006 | 0.033 | 0.027 | -0.330 | -0.257 | -0.151 | 0.106 |
| 山 レビュー回数 | 0.053 | 0.310 | -0.061 | 0.048 | -0.026 | 0.150 | 0.090 | 0.330 |
| 山 レビュー時間/回 | -0.062 | -0.179 | -0.293 | -0.236 | -0.108 | -0.272 | -0.069 | -0.110 |
| 林 バグ修正回数 | 0.104 | 0.191 | 0.213 | 0.168 | 0.029 | -0.050 | 0.091 | 0.166 |
| 林 修正時間/回 | 0.312 | 0.136 | -0.022 | 0.060 | -0.003 | -0.116 | 0.112 | 0.152 |
| 山 Jenkins ビルド成功率 | 0.114 | -0.015 | 0.121 | -0.072 | -0.168 | 0.025 | -0.311 | -0.172 |

結果は，こちらが想定していた開発者特性の属性で有意な相関は 1 つだけ見つけられた．山になりたい被験者 (want_mountain) はレビュー回数と弱い正の相関がある．今回の結果からは，山になりたいという評価をした被験者は，レビュー回数が多くなる傾向が見られた．現段階で山かどうかは不明だとしても事前にアンケートをとることで，レビューを複数回実施する開発者はある程度特定できるということである．

その他の結果は，事前アンケートによる結果は割り当てたメトリクスと有意な正の相関は得られなかったため，アンケートを事前に開発者特性を知る方法としては利用出来ない．

負の相関については，自身は林であると自己評価した被験者は風に割り当てた実装行数と個人の総実装行数の割合が低くなる傾向が見られた．風が低くなる可能性を発見する方法として利用出来ると考えられるが，風が高い開発者を発見する方法ではない．

同様に，自身が火であると自己評価した被験者はチケット対応コミットが低い傾向にあるので，林ではないということが言える．よって，事前アンケートでは開発者特性である風林火山の内，山を知ることが出来ることが分かった．

### 5.3 RQ2 開発者特性の風林火山への割り当ての妥当性を事後アンケートで評価できるか

RQ1 との違いは，被験者が風林火山を自己評価する際の判断材料として実施したプロジェクトの経験が入ってくることである．そのため，事前アンケートとは異なる結果を示す可能性は低くない．また，事前アンケートと開発者特性の比較結果は事前アンケートの有意性を示すものであるが，事後アンケートとの比較はどちらかというと開発者特性をどう風林火山分類に集約するかの見直しに繋がると考えられる．つまり，アンケート結果と開発者特性の相関が，強い正の相関かつ属性が一致している場合は開発者特性は開発者の能力を風林火山で測るには十分なメトリクスであると言える．弱い正の相関でかつ属性が一致している場合は，ある程度メトリクスは有用といえるが，RQ1 と同様に弱い値を示した原因がいくつかあると考えられる．

結果を表 5 に示す．有意な相関結果には網掛けを施している．表の縦は，RQ1 と同様であるが，横は自身が各属性であると思うかどうかではなく，自身が各属性であったかを 1 から 5 段階評価するものに置き換わっている．山と林は割り当てに合った有意な弱い相関が得られ，風と火は割り当てとは異なる部分で負の弱い有意な相関が見られた．表の各行について結果と考察を述べる．理想的な結果を示したのは，山と林に割り当てたメトリクスである．

火属性に割り当てたスクラムマスターランクは負の弱い相関が見られた．これは，被験者がスクラムマスターが出来たかどうかと火属性は相反するという意見が得られたということである．2 つの共通部分は 4.6 節で述べたように成果物の競争力を高めることと本論文では考えたが，被験者の着目点はそこでは無かったと考えられる．

個人の総実装行数の割合及び，総実装行数は有意な基準である ($p < 0.05$) すなわち相関係数の絶対値が 0.268 以上を満たしたものは無かった．一方，自身を風と評価した被験者と個人の総実装行数の割合には 0.072 とわずかに相関があった．風とは実装の速度のことであり，量ではないためこのような有意ではない結果になったと考えられる．同様にコード量という観点である総実装行数で-0.059 を示した理由は，個人の総実装行数はチーム内における自身の相対量であるが，総実装行数はチーム外を含めた 54 人の量だからだと考えられる．すなわち，両者の相関でこのような差がついた原因は，自身が風であるかを聞かれた際，チーム内でどれだけ貢献したかの経験を重視したのではないかという予想である．補足として，今回のメトリクスに実装速度が無い理由は，実装速度を算出するためのデータに Trac の手入力部分であるチケット完了までにかかった時間を利用していたが，2013 年度では特に問題は無かったが，2014 年度では作成 (ソースコード) のタスクにおいて記入漏れがいくつも観測されたため，不正確な値が算出されたので比較対象から外した．

チケット対応コミットは林に割り当てていたが，自身が林であったかどうかとは無相関になり，林でありたいかについてはわずかに有意な値に届かなかった．どちらかというと山属性の方がいずれの場合も相関で指示されていた．

**表 5** 事後アンケートと開発者特性の相関

| | were_wind | want_wind | were_woods | want_woods | were_fire | want_fire | were_mountain | want_mountain |
|---|---|---|---|---|---|---|---|---|
| 火 スクラムマスターランク | -0.099 | -0.227 | 0.100 | -0.090 | -0.300 | -0.167 | 0.114 | -0.030 |
| 風 個人の総実装行数の割合 | 0.072 | 0.036 | 0.256 | 0.267 | 0.140 | -0.016 | 0.122 | 0.070 |
| 風 総実装行数 | -0.059 | 0.073 | 0.173 | 0.248 | 0.093 | -0.054 | 0.050 | 0.030 |
| 林 チケット対応コミットの割合 | -0.058 | 0.330 | 0.024 | 0.234 | 0.015 | 0.134 | 0.224 | 0.210 |
| 山 レビュー回数 | 0.058 | 0.110 | 0.240 | 0.184 | -0.011 | 0.064 | 0.461 | 0.156 |
| 山 レビュー時間/回 | -0.006 | 0.244 | -0.033 | -0.071 | 0.068 | -0.036 | -0.203 | -0.134 |
| 林 バグ修正回数 | -0.028 | 0.226 | 0.269 | 0.314 | 0.269 | 0.134 | 0.246 | 0.127 |
| 林 修正時間/回 | -0.077 | -0.223 | 0.065 | -0.360 | 0.017 | -0.165 | 0.028 | -0.282 |
| 山 Jenkins ビルド成功率 | -0.179 | -0.025 | -0.021 | 0.098 | -0.039 | -0.126 | 0.090 | 0.033 |

**表 6** テストコードの品質と山と林のメトリクス

| team | instruction | branch | ave | 命令ミス | 命令カバー | 分岐ミス | 分岐カバー | ave_M9 | max_M9 | ave_M7 | max_M7 | ave |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| B1 | 84.9% | 85.2% | 85.1% | 573 | 3216 | 18 | 104 | 9 | 24 | 13 | 28 | 19 |
| B7 | 77.4% | 86.0% | 81.7% | 886 | 3027 | 16 | 98 | 8 | 19 | 8 | 19 | 13 |
| B4 | 76.1% | 86.6% | 81.3% | 1111 | 3528 | 18 | 116 | 8 | 13 | 11 | 22 | 13 |
| B8 | 84.9% | 77.2% | 81.0% | 599 | 3363 | 26 | 88 | 9 | 13 | 12 | 16 | 12 |
| B2 | 80.1% | 81.5% | 80.8% | 724 | 2920 | 23 | 101 | 6 | 13 | 11 | 14 | 11 |
| B3 | 78.6% | 78.7% | 78.7% | 787 | 2895 | 23 | 85 | 5 | 10 | 7 | 12 | 9 |
| B6 | 78.6% | 76.9% | 77.7% | 1039 | 3810 | 30 | 100 | 4 | 8 | 7 | 16 | 9 |
| B9 | 80.4% | 74.0% | 77.2% | 713 | 2926 | 27 | 77 | 3 | 7 | 9 | 12 | 8 |
| B5 | 60.0% | 62.3% | 61.1% | 1135 | 1703 | 40 | 66 | 5 | 11 | 5 | 10 | 8 |

山と林の区別についてはプロジェクトに対する能力かプロダクトに対する能力かといった観点で行っていたが，チケット対応コミットは Trac のチケットのように成果物としても表れ，ホワイトボードなどによるプロジェクトの進行管理にも表れる重複したメトリクスであったと考えられる．その結果，山と林両方でわずかな正の相関がでたのであろう．

レビュー回数は山に割り当てており，さらに自身が山であったかの評価と有意な相関が得られた．しかし，レビュー時間/回とは負の相関が得られた．レビューを厳密に実施したことが山であるかどうかの判断材料になると予想出来るが，比較結果は回数を重視したものとなった．被験者の体験を考えると，レビューにかけた時間よりも，成果としてレビューのチケットを実施した回数 (レビュー回数) の方が印象に残ることや，直感で定量的に判断出来るか否かが影響されたと考えられる．このレビュー回数というメトリクスについては RQ3 で更に分析を深める．

バグ修正回数は林に割り当てており，さらに自身が林であったか及び林になりたいかの評価で正の有意な弱い相関が見られた．林は突発的なバグへ対処する能力が求められているため，意味がメトリクスと直接繋がったから正の相関がでたと考えられる．弱い相関である理由は，プロジェクトの期間中にバグ修正を体験できるかは一定数あるわけではないため体験していない人は自身はあてはまらないと判断したと考えられる．他の属性でも相関がでている理由はバグ修正することのより潜在的な部分で共鳴したと考える．被験者の視点で考えると，バグ修正という突発的なタスクはアサイメント制約に影響しないため，個人の評価には繋がりにくい．それを踏まえた上で，チームのために実施するタスクであり，被験者の積極性が試されるタスクであると言える．火は積極的にツールを導入し成果物の競争力を高める能力であるため，その点が共通している．本稿では，分析結果を複雑にしないために出来限り単純明快な理由で割り当てるポリシーであるため，バグ修正回数は引き続きは林であるとする．このバグ修正回数もレビュー回数と同様に RQ3 で利用する．

Jenkins ビルド成功率は，チームのリポジトリの状態に依存するため個人がいくら正当な成果物をコミットしても，失敗する危険がある．Eclipse ビルド成功率が収集出来なかったため代用として比較対象にいれたが 0.090，0.033 と山との相関は出なかった．ただし，自身を山であると評価したわずかな正の相関 0.090 は少なからず，手元でビルドが通りまた，テストも通った開発者がいたのではないかと考えられる．他の原因は，レビュー回数と山の関係が 0.461 と有意であったため，そちらに回答が引き寄せられたとも考えられる．なぜなら回答者は，すべてのメトリクスを把握している訳ではなく，経験を辿り独自の思考で評価軸 (本稿ではメトリクス) を決めるため，回答者の中にあるメトリクスの数は一定ではない．また，多角的にに自己評価をする回答者である保証はなく，1 つのメトリクスに引き寄せられることはプロジェクトの評価には関係しないアンケートであることからも考えられる．よって，山と林の割り当ては，第一著者の主観による割り当てと回答者のアンケートで有意な相関がでたため妥当であると考えた．

### 5.4 RQ3 開発者特性が成果物の品質にどのような影響を与えるのか

RQ2 の結果から，山属性はレビュー回数からある程度測定でき，林属性はバグ修正回数から測定できることが分かった．山と林はいずれも成果物の品質に関わる属性である．そこでこの章では，山と林が与える成果物の品質について調査する．

調査する目的は，山や林の高いチームメンバーを含んだ場合に成果物の品質向上が見込めるかどうかを確認することである．

調査方法は，各チームのテストコードの定量的な品質と各チームの山 (レビュー回数) と林 (バグ修正回数) の平均値と最大値を比較することである．

比較結果の理想としては，山や林が高いほどテストコードの品質は高くなるということである．以下にテストコードの品質を定量化するためのメトリクスを挙げる．

(1) 命令のミス数 ($O_m$)
(2) 命令の網羅数 ($O_c$)
(3) 分岐のミス数 ($B_m$)
(4) 分岐の網羅数 ($B_c$)
(5) 命令の網羅率 ($O_{cr} = O_c/(O_c + O_m)$)
(6) 分岐の網羅率 ($B_{cr} = B_c/(B_c + B_m)$)

結果を表 6 に示す．テストコードの品質と山と林のメトリクスをチームごとに分けている．ave はチーム内の平均で，max はチーム内の最大値を表す．テストコードの品質の平均値が高いほど，山と林の平均値が伸びていることが分かる．また，各特性を示す値の最大値などもほぼそれに従っていることが分かる．以上のことから，山と林が高ければ高品質な成果物が作成されることが分かった．

## 6. まとめ

本稿は，2013 年度の開発履歴を対象に行った先行研究の課題を解決するために，2014 年度の開発履歴を用いて調査を行った．課題は 2 つあり，1. 事前に開発者特性を特定すること，2. 風林火山分類へのメトリクスの割り当ての根拠が第一著者の主観による．であった．

そこで，1 を解決する方法として，事前アンケートによって開発者特性を特定する方法がある．そこで，事前アンケートの有効性を確かめるために開発履歴による開発者特性との関連を調査した．調査によると，山の特性のみアンケート結果と開発履歴による開発者特性で正の弱い相関がみられた．つまり，自己評価を山であるというアンケート結果はその後の開発で，山の傾向が見られるということである．

また，2 を解決する方法として，事後アンケートを利用する方法がある．事前に比べ，プロジェクトの経験を踏まえた自己の特性分析結果が得られるためより信用出来ると考えられる．メトリクスの割り当てについて，第一著者の主観とプロジェクトの経験者の自己評価を比較し調査した．調査によると，山と林であると自己評価した被験者は，こちらが山に割り当てたレビュー回数と，林に割り当てたバグ修正回数と正の有意な相関が得られた．この結果により，山と林が高いチームの成果物の品質は高いことが予想出来るため，プロジェクトの成果物にあたるテストコードの品質と被験者の山および林特性を比較した．結果は，山や林が高くなるにつれてテストコードの品質も向上していることが確認できた．

## 謝辞

本研究の一部は科研費 (萌芽研究 26540029) の助成を受けたものである。

### 参考文献

[1] 五田篤志，山崎　尚，玉田春昭，畑　秀明，角田雅照，井垣　宏：開発履歴を利用した風林火山モデルに基づく開発者特性の分析，研究報告ソフトウェア工学 (SE), volume 2014-SE-185, number9 (2014).
[2] 小野和俊：Developer のための 5 つの習慣 —日本をソフトウェア輸出大国にしていくために，デベロッパーサミット 2006 (2006).
[3] 角田雅照，玉田春昭，畑　秀明：ソフトウェア開発チームにおけるパーソナリティの影響に関する調査，SES2013 併設ワークショップ「開発マネジメントにおける産学の問題共有と連携強化」(2013).
[4] Onoue, S., Hata, H. and Matsumoto, K.: A Study of the Characteristics of Developers' Activities in GitHub, *5th International Workshop on Empirical Software Engineering in Practice*, pp. 1–6 (2013).
[5] James, J.: 10 types of programmers you'll encounter in the field (2007). http://www.techrepublic.com/blog/10-things/10-types-of-programmers-youll-encounter-in-the-field/ (Last Access: 2014/1/21).
[6] 中村匡秀，井垣　宏，佐伯幸郎，まつ本真佑，楠本真二，上原邦昭，井上克郎：Cloud Spiral の取り組み，日本ソフトウェア科学会 第 30 回大会 講演論文集 (2013).
[7] Rising, L. and Janoff, N. S.: The Scrum software development process for small teams, *IEEE Software*, Vol. 17, pp. 26–32 (2000).
[8] Fukuyasu, N., Saiki, S., Igaki, H., Matsumoto, S. and Kusumoto, S.: A Case Study of Cloud-enabled Software Development PBL, *2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp. 499–504 (2013).
[9] 小川明彦，阪井　誠：Redmine によるタスクマネジメント実践技法，翔泳社 (2010).