情報処理学会論文誌 Vol.59 No.2 486–495 (Feb. 2018)

ソーシャルコーディングにおける有益提案の抽出

江見 圭祐[1] 乃村 能成[1,a)] 谷口 秀夫[1]

受付日 2017年5月9日, 採録日 2017年11月7日

概要：ソフトウェア開発において，ソーシャルコーディングと呼ばれる手法が広がりつつある．ソーシャルコーディングで進められるプロジェクトには，様々なユーザからソースコードの改善に関する提案が寄せられる．これら提案の採否の判断には，提案の内容だけでなく，提案を発案したユーザの評判も影響を与えている．本論文では，ユーザの評判によるバイアスを排除した提案の良し悪しの指標を述べ，提案にまつわるユーザの行動から，その指標を算出する手法を示した．次に，本手法が算出する指標を評価し，指標が有効に機能することを示した．最後に，評価結果を分析し，実際の提案の採否の判断にユーザの評判によるバイアスが存在する可能性を確認した．

キーワード：ソーシャルコーディング，仕事効率化

Extraction of Significant Contributions in Social Coding

Keisuke Emi[1] Yoshinari Nomura[1,a)] Hideo Taniguchi[1]

Received: May 9, 2017, Accepted: November 7, 2017

Abstract: Social coding is getting widespread among software development societies. Various users propose contributions improving source code for social coding projects. Not only the proposal itself but also the contributor's reputation affects the decision on acceptance/rejection of these contributions. In this paper, we described an indicator of the quality of contributions excluding the bias due to contributor's reputation, and showed a method to measure the indicator by learning user's behavior related to the contribution. Then, by evaluating the indicator calculated by this method, we showed that the indicator performs effectively. Finally, we confirmed that the bias due to contributor's reputation may exist in actual decision on acceptance/rejection of contributions by analyzing evaluation results.

Keywords: social coding, productivity

1. はじめに

ソフトウェア開発において，ソーシャルコーディングと呼ばれる手法が広がりつつある [1]．ソーシャルコーディングとは，ソーシャルネットワークの考え方をソフトウェア開発に適用した試みである．

ソーシャルコーディングは，ソフトウェアプロジェクトの開発を通して，ユーザ間のコミュニティの形成の支援とコミュニティ内，またはコミュニティ間での交流を促進する．ユーザ，もしくはユーザによって構成されるグループは，自身が管理するソフトウェアプロジェクトのソースコードと開発の様子を公開する．外部のユーザは，公開されているプロジェクトに対して，自由に「イイネ」を付けたり，ソースコードレビューや改善案を掲示板に書き込んだりできる．また，これらのやりとりが公開されていることによって，誰がどのソフトウェアプロジェクトに興味があり，どの程度のソースコード量をどのプロジェクトに提供しているのか可視化され，ユーザのコーディングスキルやプロジェクトでの貢献度，興味のある分野が分かる．このようなユーザ間のやりとりを通して，ユーザの評価や評判が形成され，ユーザ間の交流と協調につながる [2]．

こうしたソーシャルコーディングの手法を取り入れた代表的なサービスとして GitHub [1] があり，2017 年 5 月現在で 2,100 万人のユーザと 5,700 万のソフトウェアプロジェクトを擁するまでとなっている [3]．GitHub は，ソー

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University, Okayama 700–8530, Japan
[a)] nom@cs.okayama-u.ac.jp

スコードレビュー，バージョン管理，チケット管理，およびパッチ提示といった作業を支援する Web システムを提供している．そして，これらの作業履歴は記録され，可視化されている．

ソーシャルコーディングで開発が進められるプロジェクトは公開されているため，プロジェクト内外の様々なユーザからソースコードの改善に関する提案が発案される．このような提案を積極的にプロジェクトに採用することは，コミュニティの活性化とプロジェクトの成功につながる [4]．しかし，提案によって内容の良し悪しは大きく異なるため，プロジェクトオーナは，発案された提案の内容を吟味し，提案の採否を判断する必要がある．このため，大量に提案が発案されるようなプロジェクトでは，有益そうな提案から優先的に内容を吟味したいという要求がある．しかし，提案の良し悪しは提案を吟味した後でなければ判断できない．このため，提案を吟味する前に提案の良し悪しを測る尺度が必要である．

ソーシャルコーディングは，提案をしたユーザが「誰なのか」という情報が重要な要素となる．なぜならば，ソーシャルな環境における人間関係を強く反映しており，プログラミングという行為が人に強く依存するためである．このような要素は，プロジェクトでの実績や貢献度と直結しており，ユーザの評判と見なすことができる．プロジェクトオーナは，このようなユーザの評判を提案の採否の判断基準として用いることもある [2], [5]．しかし，ユーザの評判を基に提案の優先順位を決めた場合，プロジェクトに初めて参加するようなユーザの発案した提案は優先順位が低くなり，コアなユーザが発案した提案ばかり優先的に対応される．これはプロジェクトの公平性や健全性を損ない，コミュニティの活性化を妨げる要因となる．

プロジェクトオーナが人間である以上，提案の採否の判断にユーザの評判によるバイアスがかかることは避けられない．そこで，本論文ではユーザの評判によるバイアスを排除した提案の良し悪しの指標を述べる．この指標を用いることで，プロジェクトオーナは提案の採否を公平に判断でき，プロジェクトの健全性の確保につながる．また，算出された指標と実際の採否の判断を比較することにより，プロジェクトオーナの提案の採否の判断にユーザの評判によるバイアスが存在するか否か考察する．具体的には，プロジェクトごとに提案にまつわるユーザの行動を学習し，新規に発案された提案について，指標を算出する手法を提案する．そして，提案手法で算出した指標と実際の採否が大きく異なっている提案について，その原因を提案に関するやりとりなどから定性的に分析する．

2. ソーシャルコーディングによるソフトウェア開発

2.1 GitHub におけるソーシャルコーディング

代表的なソーシャルコーディングサービスである GitHub における開発の流れを説明して，ソーシャルコーディングの特徴を説明する．GitHub のユーザには，自身のプロフィールページが用意され，自己紹介や GitHub での開発活動の履歴などが公開される．

ユーザは，自身がプロジェクトオーナとなってソフトウェア開発プロジェクトを開設できる．プロジェクトは他ユーザにも公開されており，プロジェクトオーナだけでなく，他ユーザも開発に関わることができる．他ユーザは，公開されているプロジェクトから自身のプロジェクトを派生させ，自由にプロジェクトを改変できる．このような開発の流れは，ソーシャルコーディングの特徴的な開発形態である．以上の開発の流れを容易にするために，プロジェクトには以下の機能が提供される．

(1) ソースコードの共有スペース（repository）
repository は，バージョン管理機能を備えるソースコードとドキュメントの共有スペースである．repository は，commit と呼ばれる変更の単位で開発される．commit は commit id によって一意に識別可能で，GitHub ではしばしば特定の commit を言及する際に commit id を用いて参照することがある．また，ブランチと呼ばれる機能によって開発の流れが管理され，任意の時点で自由に分岐させることできる．そして，分岐したブランチは他のブランチとは独立して開発できる．多くのプロジェクトは，開発のためにブランチを派生させ，本流のブランチに統合することで開発を進める．このようなプロジェクトの開発モデルの代表例として，gitflow がある [6]．

(2) ソースコードの変更の取り込み要求（pull request）
pull request は，ユーザが施したソースコードの変更をプロジェクトに取り込みやすくするための機能である．repository は全ユーザが自由に複製可能で，プロジェクトに独自の変更を施すことができる．repository を複製し，派生 repository を作成することを fork と呼び，プロジェクトの派生関係はシステムで管理されている．pull request を作成することで，ユーザの施した変更を改善案として，派生元のプロジェクトに提示できる．派生元のプロジェクトオーナは，提示されたソースコードを認めれば，取り込み操作（merge）をするだけで変更を簡単に取り込める．こうして，ユーザの施したソースコードの変更は派生元のプロジェクトの一部として取り込まれる．

これまでに述べた GitHub 上での行動は，システムのデータベースで管理され，公開されている．たとえば，プ

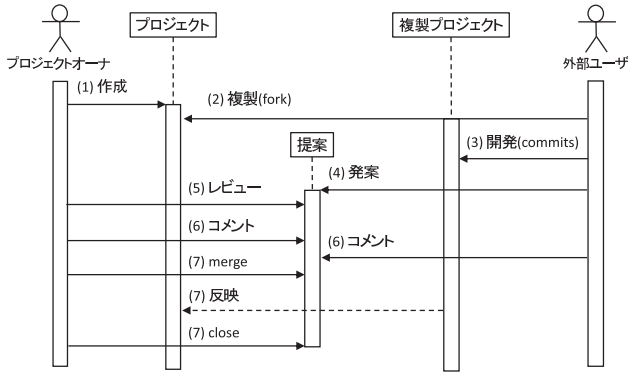

図 1 GitHub で提案が発案されてから対応が終了するまでの流れ
Fig. 1 Flow of contribution in GitHub.

ロジェクトが fork された回数も公開されており，プロジェクトの知名度の指標となっている．また，あるユーザが発行した pull request の回数や一覧を調べることも可能である．このため，公開データは，プロジェクトやユーザを評価する指標として使用される [7]．GitHub で公開される様々な指標によって，プロジェクトやユーザの評判が形成される．たとえば，あるユーザがプロジェクトにどれほど貢献してくれるかを予測するには，そのユーザが過去に関わったプロジェクトやそこでの活動履歴を見ればよい [8]．またプロジェクトに必要な人材を GitHub のユーザのプロフィールから選定することも可能である [9]．

2.2 GitHub における提案

GitHub では，他ユーザの管理するプロジェクトを複製し，自身のプロジェクトとして自由に変更できる．自身の変更は，pull request を用いることで，ソースコードの改善案として複製元のプロジェクトに提示できる．本論文では，このようなソースコードの変更をともなう改善案を提案と呼ぶ．

図 1 を用いて，GitHub で提案が発案されてから対応が終了するまでの流れを説明する．

(1) プロジェクトオーナがプロジェクトの開発を行う．
(2) 外部ユーザは，プロジェクトオーナのプロジェクトを自由に複製（fork）できる．
(3) 複製されたプロジェクトは元のプロジェクトとは独立しており，外部ユーザが自由に開発できる．
(4) 外部ユーザが提案（pull request）を発案することで，複製プロジェクトに施した変更を複製元プロジェクトに取り込むように依頼する．
(5) プロジェクトオーナは提案の中身を吟味し，レビューやテストを行う．
(6) プロジェクトオーナは，プロジェクトの参加者と提案について議論を交わし，提案の採否を判断する．
(7) 提案を採用する場合，プロジェクトオーナは取り込み操作（merge）を行い，プロジェクトに変更を反映させ，提案を終了（close）する．一方，拒否する場合，外部ユーザに提案を拒否する理由を説明し，提案を終了する．

2.3 ソーシャルコーディングでの課題

ソーシャルコーディングで進められるプロジェクトの中には，プロジェクトのコアメンバからだけでなく，世界中の様々なユーザから提案が寄せられ，提案の内容がプロジェクトに反映されるものもある．これは，ソーシャルコーディングの利点であるが，欠点にもなる．プロジェクトの参加者の中には，プロジェクトへの理解度の低い者も多いため，プロジェクトに取り込むべきではないような内容の提案も数多く発案される．本論文では，このような提案を無益提案と呼ぶ．無益提案は，プロジェクトに取り込まれないばかりか，提案を採用しない理由の説明などの負担をプロジェクトオーナに強いる．対して，本論文では，プロジェクトに取り込まれる提案を有益提案と呼ぶ．開発に参加するユーザ数の多いプロジェクトでは，無益提案が増加すると示されている [10]．

無益提案が多く発案されるような状況では，プロジェクトオーナには，なるべく有益な提案から対応し，提案の内容をプロジェクトに反映させたいという要求がある．しかし，提案を吟味しなければ，提案の良し悪しは判断できない．プロジェクトオーナは，提案を発案したユーザの評判を用いて，提案を吟味する前に提案の良し悪しを測ることもある．しかし，提案を発案したユーザの評判を基に提案の優先順位を決めると，プロジェクトに初めて参加するようなユーザの発案した提案が後回しにされる．これは，プロジェクトの公平性や健全性を損ない，コミュニティの活性化を妨げる要因になると考えられる．

以上のことから，ソーシャルコーディングでの課題として以下があげられる．

課題 ユーザの評判に代わる提案の良し悪しを判断する指標の定義

ユーザの評判に代わる指標を基に，膨大な提案の中から有益な提案を優先的に反映できれば，プロジェクトオーナの負担の軽減とプロジェクトの活性化につながる．

2.4 目的

本論文では，2.3 節で述べた課題への対処として，提案を発案する際のユーザの行動を基に提案の良し悪しの指標を算出することを目的とする．ユーザの評判の代わりに，算出した指標を提案の優先順位付けに用いることで，プロジェクトオーナの負担の軽減だけでなく，コミュニティの拡大と活性化につながる．具体的には，プロジェクトごとに提案にまつわるユーザの行動を学習し，提案が発案された時点での提案の良し悪しの指標を算出する．プロジェクトオーナは，算出された指標を基に有益だと考えられる提

案を抽出することで，優先的に有益提案に対応できる．

3. 提案の採否とユーザの評判の関係の調査

3.1 調査目的

初めてプロジェクトに参加するユーザが有益提案を発案する割合が低ければ，初めてプロジェクトに参加するユーザが発案した有益提案を取り出し，内容を反映させる負担は大きい．しかし，プロジェクトの活性化のためには，このようなユーザからの提案を積極的に採用し，プロジェクトに参加するユーザの多様性を高める必要がある．そこで，提案を発案したユーザの評判によって，有益提案の割合に違いがあるかを調査する．具体的には，初めてプロジェクトに提案を発案するユーザと過去に提案を発案したことのあるユーザを区別した場合の有益提案の割合を調査する．

3.2 調査対象と有益提案の条件

提案を有益なものと無益なものに分別するために，まずは調査対象プロジェクトの選定と有益提案の条件を設定する．なお，調査するデータは，GHTorrent プロジェクト [11] のデータベースから取得できるものを使用する．GHTorrent プロジェクトは，GitHub データのミラーを提供しており，GitHub 上のプロジェクトやイベントの情報が蓄積されている．

調査対象の選定のために，以下の 3 つの条件を設定する．

(1) プロジェクトの参加者の多いプロジェクトである
2.3 節で述べたような問題は，参加者が多く，発案される提案の質にばらつきがあるようなプロジェクトで深刻となる．ここでは，プロジェクトの fork 数をプロジェクトに参加しているユーザ数の指標として扱う．

(2) 提案が多く発案されているプロジェクトである
プロジェクトに多くの提案が発案されていれば，プロジェクトが提案を積極的に用いて開発されていることを示す．

(3) ソフトウェア開発を目的とするプロジェクトである
GitHub には，文書管理のためのプロジェクトも存在する．我々が興味を持つのは，ソフトウェア開発を行っているプロジェクトで発案される提案の良し悪しを予測することである．

これらをすべて満たすプロジェクトとして，angular.js, bootstrap，jquery，および rails の 4 つのプロジェクトを選定する．

次に，議論が終了した過去の提案が有益か否か判定するための条件について検討する．本論文での有益提案は，内容がプロジェクトに反映されている提案，つまり，プロジェクトに採用された提案のことを指す．

GitHub では，提案の内容をプロジェクトに反映させる仕組みとして，取り込み操作（merge）以外にも様々な方法がある．たとえば，提案で示された commit の中から特定の commit を拾い上げ，プロジェクトに取り込むことができる．一方，提案の内容を直接取り込むのではなく，プロジェクトオーナが手直しをして commit をすることで提案の内容をプロジェクトに反映させることもある．GitHub では，プロジェクトに commit を加える際に，commit で対処した提案について commit message に記述することで，自動的に提案を close できる．また，プロジェクトオーナは，提案についての議論の中で，特定の commit へのリンクを貼り，言及することで，当該 commit により提案をプロジェクトに反映させたことを示すこともある．

以上のことから，本論文では，提案がプロジェクトに採用されたか否かを機械的に判別するための条件として以下の 4 つを設定する．これらの条件のうち，1 つでも満たす提案を有益提案とする．

条件 A 提案が merge されている．

条件 B 提案由来の commit がプロジェクトのメインブランチに含まれている．

条件 C 提案が commit message により close されている．

条件 D 提案の議論の中に commit id が含まれる．

3.3 調査結果

提案の採否とユーザの評判の関係の調査として，提案を発案したユーザを一見ユーザとコアユーザに区別し，ユーザの種類によって発案される提案の有益提案と無益提案の割合が異なるか調査する．ここで，一見ユーザとはプロジェクトに初めて提案を発案するユーザのことを指しており，このようなユーザの評判はプロジェクト内では未知である．一方，コアユーザとは過去に提案を発案したことがあるユーザを指す．

3.2 節であげたプロジェクトについて，ユーザの種類ごとのユーザ数，提案数，および有益提案数の割合を**表 1** に示す．ここで，表 1 の一見ユーザが発案した提案数は，プロジェクトに参加したユーザが初めて発案した提案を集計している．つまり，プロジェクトに参加したすべてのユーザは少なくとも 1 回は一見ユーザとして扱われる．一方，表 1 の一見ユーザのユーザ数は，最終的に 1 個の提案しか発案しなかったユーザを集計している．このため，一見ユーザの発案した提案数は，一見ユーザのユーザ数よりも多く，プロジェクトに提案を発案したユーザ数の合計となる．

表 1 から，すべてのプロジェクトにおいて，一見ユーザの数はコアユーザよりも多い．これは，プロジェクトに 1 度提案を発案しても，その後継続的にプロジェクトに参加するユーザが少ないことを表す．一方，ユーザの種類ごとの提案数から，一見ユーザが発案した提案数よりもコアユーザが発案した提案数の方が多い．さらに，一見ユーザが発案した提案よりもコアユーザの発案した提案の方が有益提案の割合が高い．このことから，少ないコアユーザが

表 1 ユーザの種類とプロジェクトごとのユーザ数，提案数，および有益提案の割合の内訳
Table 1 Number of users, number of contributions, and acceptance ratio for each user type and projects.

	ユーザ数		提案数		有益提案の割合	
プロジェクト名	一見ユーザ	コアユーザ	一見ユーザ	コアユーザ	一見ユーザ	コアユーザ
angular.js	1,958	691	2,649	3,857	49.3%	55.2%
bootstrap	1,665	550	2,215	3,074	36.6%	71.4%
jquery	502	165	667	1,219	17.2%	36.6%
rails	2,029	1,451	3,480	11,411	59.8%	77.4%

多く有益提案を発案し，プロジェクトに貢献していることが分かる．コミュニティの活性化のためには，一見ユーザが発案した提案を積極的に採用し，継続的にプロジェクトに貢献するユーザを増やす必要があるが，一見ユーザの発案する提案の有益な割合は，コアユーザと比べて低く，有益提案を見つけ出す負担は大きい．そこで，本論文では，一見ユーザが発案した提案に焦点を当て，提案の良し悪しの指標を算出することを目指す．

4. ユーザの行動を基にした提案の良し悪しの予測手法

4.1 概要

2.4 節で述べた目的を実現するために，提案が持つ性質を学習することで提案の採否を予測する手法を提案する．具体的には，すでに議論の終了した過去の提案にまつわるユーザの行動を学習し，新しく発案された提案が有益提案である確率を算出する．プロジェクトオーナは，確率の高い順に提案を対応することで，効率的に有益提案に対応できる．そこで，まず提案にまつわるユーザの行動について調査し，学習する素性について述べる．

4.2 学習する素性

提案をするユーザは，プロジェクトのブランチを指定し，提案をどのブランチに取り込んでほしいかを示す．無益提案の中には，指定したブランチが適切でないことが理由で拒否されたものもある．たとえば，開発中のブランチを無視して古いバージョンのブランチを指定した提案や最新のブランチで問題が解決していることに気づかずに別ブランチを指定した提案などである．このことから，提案の指定先ブランチと採否には関係があると考える．

提案の指定先ブランチと採否の関係を把握するため，プロジェクトに発案された提案の指定先ブランチの情報を基にブランチごとの特徴を調査する．今回調査するプロジェクトは bootstrap であり，使用する提案の期間は 2011 年 8 月から 2016 年 9 月までの期間である．

図 2 に bootstrap の master ブランチと開発用ブランチで発案された提案数の推移を示す．図 2 の濃淡が各ブランチで発案された提案数を示しており，濃い部分ほど提案数が多く，薄い部分ほど提案数が少ない．図 2 から，master

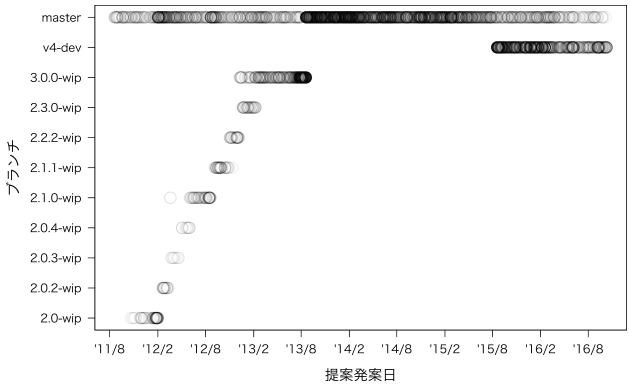

図 2 bootstrap のブランチごとの提案の推移
Fig. 2 Timelines of contributions each branch in bootstrap.

ブランチは全期間を通して多くの提案が発案されていることが分かる．一方，開発用ブランチの多くは，以前のバージョンと入れ替わるように提案が発案される期間が推移している．また，他の開発用ブランチと提案が発案される期間が重なっている開発用ブランチも存在する．たとえば，2.1.0-wip ブランチと 3.0.0-wip ブランチは，それぞれ 2 つの開発用ブランチと提案が発案された期間が重なっており，並行して開発が行われている．このように，開発用ブランチは時に開発期間が重なりながら，ほとんど隙間なく存在している．しかし，3.0.0-wip ブランチから v4-dev ブランチの間は，開発用ブランチに提案が発案されていない．この期間の提案のほとんどは master ブランチに発案されている．このため，開発用ブランチが存在する期間と存在しない期間で master ブランチに発案された提案の傾向は異なる可能性がある．そこで，master ブランチについて，提案が発案された時期とその提案の傾向について分析する．

master ブランチに発案された提案数と有益提案数の割合の推移を**図 3** に示す．横軸に提案が発案された日付をとり，ヒストグラムの縦軸（左側）は提案数，折れ線グラフの縦軸（右側）は有益提案数の割合をとる．図 3 から，提案が発案された時期によって発案される提案数と有益提案数の割合に大きな偏りがあることが分かる．開発用ブランチが存在している 2011 年 8 月から 2013 年 7 月の期間の有益提案数の推移は 20%から 40%前後であるのに対し，3.0.0-wip が統合された 2013 年 8 月から 2015 年 7 月の期間は有益提案数の割合は 60%まで上昇し，総提案数も増加

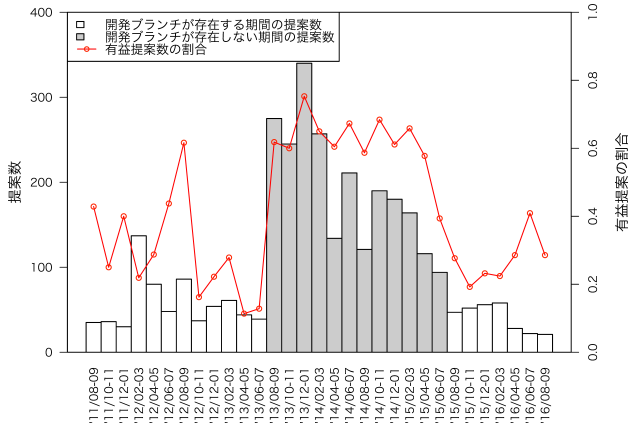

図 3 bootstrap の master ブランチの提案数と有益提案数の割合の推移

Fig. 3 Number of contributions and its acceptance ratio on master branch in bootstrap.

している．また，2015 年 8 月に v4-dev ブランチが発案され，新バージョンへの過渡期が終了すると総提案数と有益提案数の割合は減少している．新バージョンへの過渡期においては，開発用ブランチが存在しないため，master ブランチを開発用ブランチとして開発を進めていたと考えられる．なお，2012 年 2 月から提案数が一時的に増加し，その後 2012 年 9 月まで有益提案の割合が上昇しているのは，2012 年 1 月 31 日に bootstrap の version 2.0.0 がリリースされたことに起因していると考える．このように，あるブランチが開発用のブランチかどうかは時期によって変化するため，提案発案時にブランチがどれほど活発に開発されているかが重要だと考える．

4.3 学習に用いる素性とアルゴリズム

4.2 節での調査結果と既存研究から，以下の 5 つの素性を学習に用いて，提案が採用される確率を予測する．つまり，これが提案の良し悪しの指標になる．なお，学習アルゴリズムとしてランダムフォレストを使用する．

素性 1 提案が発案されたブランチの活発度
素性 2 提案で変更されたファイルの活発度
素性 3 提案で変更されたソースコードの行数
素性 4 提案で変更されたファイルの数
素性 5 提案が含む commit の数

（素性 1）は，提案が発案されたブランチがどの程度活発に開発されているブランチかを表現する．指定したブランチが活発に開発されていれば，そのブランチでは提案は受け入れられやすい．一方，活発に開発されていないブランチは，安定版のブランチだったり，プロジェクトオーナにとってもはや興味のないブランチだったりする可能性がある．このようなブランチに提案を発案しても受け入れられにくい．（素性 1）は，対象となる提案発案日時を t_{pr}，提案の発案先のブランチを対象として，t_c から T_w 時間分だけ遡った期間の commit の集合を C_w とすると，式 (1) の $activity$ となる．

$$activity(t_{pr}) = \sum_{c \in C_w} \frac{T_w - (t_{pr} - t(c))}{T_w} \quad (1)$$

ここで，$t(c)$ は commit c の作成日時を表す．式 (1) では，$t(c)$ が提案発案日時 t_{pr} から離れているほど 0 に近づき，t_{pr} 直前であるほど 1 に近づくように重み付けを行っている．本手法では，T_w を 3 カ月として $activity$ を求める．

（素性 2）は，提案で変更されたファイルがどの程度活発に開発されているファイルかを表現する素性である．プロジェクトでのファイルの変更は時期によって偏るため，活発に変更されているファイルは変更されやすく，提案でこのようなファイルを変更しているか否かは提案の採否に強く影響を与える [12]．（素性 2）は，（素性 1）の C_w の commit のうち，提案で変更されたファイルを変更している commit のみを対象として，式 (1) と同様の計算を行うことで求められる．

（素性 3）から（素性 5）は，提案の変更の規模を表現する素性である．関連研究では，パッチの規模がパッチの採否に影響を与えると述べられている [13]．

ここで，重要なのは提案手法で利用する素性はすべてユーザの行動に起因するものであり，ユーザの評判に関係する素性は使用していない．これにより，ユーザの評判に影響を受けず提案の良し悪しの指標を算出できる．

4.4 学習する提案の選別

2.3 節で述べたように，提案の採否の判断にユーザの評判によるバイアスがかかることは避けられない．このため，ユーザの評判に関する素性を学習から排除しても，学習する正解データそのものに存在するバイアスによって提案の良し悪しの指標が影響される．そこで本手法では，一見ユーザが発案した提案のみを学習する．一見ユーザは，プロジェクトオーナにとって評判が未知のユーザであり，提案の採否の判断にユーザの評判によるバイアスがかかりにくい．このため，ユーザの評判によるバイアスを排除した学習が可能だと考える．

5. 評価と分析

5.1 手法の評価

5.1.1 評価

本論文では，提案手法を用いることで，プロジェクトオーナがどれだけ多く有益提案を抽出できるか評価する．提案手法では，議論が終了した提案を対象に学習し，議論が終了していない提案の良し悪しの指標を算出する．算出した指標の降順で提案を整列し，プロジェクトオーナに提示する．そして，プロジェクトオーナが提示された提案の上位から順番に提案の中身を吟味し，提案の採否を判断す

ることを想定する．このとき，提案の順位付けの精度が高ければ高いほど，優先的に有益提案を処理でき，より多くの提案の内容をプロジェクトに反映できる．

そこで，提案手法を用いることで有益提案がどの程度上位に集中するかを評価する．具体的には，無秩序に提案を並べた場合と提案手法から得られる指標で提案を整列した場合の順位付けの精度を比較する．

5.1.2 評価環境

本論文では，評価を行うプロジェクトとして，表 1 に示した 4 つのプロジェクトを選定する．表 1 に示した一見ユーザの発案した提案から，無作為に選んだ 9 割の提案を学習データとし，学習データを除いた 1 割をテストデータとする．

5.1.3 評価方法

本論文では，提案の順位付けの評価指標として，平均適合率（Average Precision: AP）を使用する．平均適合率とは，テストデータを上位から走査し，有益提案が現れた時点での適合率の平均値である．平均適合率は，提案の順位付けを再現率と適合率の観点から総合的に評価する指標である．平均適合率が高いほど，有益提案が上位に集中していることを示す．テストデータ中に含まれる有益提案の総数を N，整列された提案の上位から i 番目の有益提案時点での適合率を P_i とすると，平均適合率 AP は，式 (2) で求められる．

$$AP = \frac{1}{N}\sum_{i=1}^{N} P_i \tag{2}$$

また，テストデータによる評価結果の偏りを緩和させるため，ランダム関数のシード値を変えながら評価を 1,000 回試行し，平均適合率の平均値を求める．そして，提案を無秩序に並べた場合と提案手法から得られる指標で整列した場合のそれぞれについて平均適合率の平均値を求め，結果を比較する．

5.1.4 評価結果

5.1.3 項で述べた評価方法に従って，表 1 に示したプロジェクトについて平均適合率を求める．一見ユーザの発案した提案を無秩序に並べた場合と提案手法を用いて並べた場合の平均適合率を**図 4** に示す．図 4 から，すべてのプロジェクトについて，提案手法を用いて提案を並べた場合の平均適合率は，無秩序に提案を並べた場合と比べて向上していることが分かる．平均適合率は，bootstrap の場合で 0.16 と最も向上しており，最低でも jquery の場合で 0.04 向上している．プロジェクトによって，平均適合率の向上の幅に差異はあるが，膨大な提案の中から，少しでも多くの有益提案を優先的に吟味できることが重要である．提案手法に用いることで，すべてのプロジェクトについて平均適合率が向上していることから，提案手法の適用はプロジェクトオーナの負担の軽減につながるといえる．

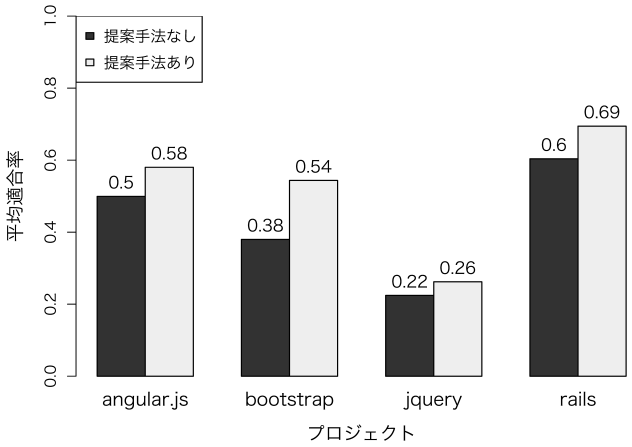

図 4 提案手法を用いた場合と用いない場合の提案の平均適合率（一見ユーザ）

Fig. 4 Average precisions for ranked contributions with/ without our method (novice users).

ここで，本評価結果を具体的なプロジェクトオーナの行動にあてはめて解釈する．たとえば，bootstrap では，1 日に処理した提案の最大数が 48 個であることが調査から分かっている．この個数を無秩序に並んだ提案から取り出した場合，含まれる有益提案数は 17.3 個，提案手法を用いて整列した提案から取り出した場合，26.9 個となり，提案手法を用いることで取り出せる有益提案は 9.6 個増加する．一方，jquery では，1 日に処理した提案の最大数は 25 個であることが分かっている．この個数を無秩序に並んだ提案から取り出した場合，含まれる有益提案数は 4.2 個，提案手法を用いて整列した提案から取り出した場合，5.4 個となり，提案手法を用いることで取り出せる有益提案は 1.2 個増加する．

図 4 から，jquery の平均適合率の向上は，他プロジェクトと比べ小さいことが分かる．これは，他の対象プロジェクトと比べて jquery はユーザ数や提案数が少なく，学習データが不足していることが原因となっている可能性がある．このため，提案手法を適用するべきプロジェクトは，参加するユーザ数や提案数ができるだけ多いプロジェクトであることが望ましい．ここで，我々の目的は，プロジェクトオーナの負担の軽減だけでなく，ユーザの評判に代わる提案の良し悪しの指標を定義することでコミュニティの活性化を図ることでもある．たとえば，bootstrap は，一見ユーザと比べて，コアユーザの有益提案の割合が非常に高く，一見ユーザの発案した提案が軽んじられている可能性がある．このようなプロジェクトで，一見ユーザの発案した有益提案を優先的に吟味できれば，コミュニティの活性化につながる．実際，図 4 から，提案手法を用いることで，bootstrap の平均適合率は向上しており，向上の度合いも最も大きい．また，たとえ平均適合率の向上が少ない jquery のようなプロジェクトであっても，本項で述べた予測結果と，一見ユーザの発案した提案を対象に学習し，

コアユーザの発案した提案を予測した結果を比較することで，一見ユーザとコアユーザが平等に扱われているかどうかの指標となる可能性がある．そこで次節では，一見ユーザの発案した提案を対象に学習した結果から，コアユーザの発案した提案を予測し，一見ユーザとコアユーザの扱われ方の違いについて分析する．

5.2 指標の分析

5.2.1 コアユーザが発案した提案の予測

5.1 節では，一見ユーザの発案した提案を対象に学習し，一見ユーザによって新たに発案された提案について，算出された指標を評価し，指標が有効に機能することを示した．そこで次に，一見ユーザの発案した提案を対象に学習し，コアユーザによって新たに発案された提案について評価する．一見ユーザの発案した提案を対象に学習することによって，コアユーザの発案した提案をユーザの評判のバイアスによらず算出できる．ユーザの評判のバイアスを排除してコアユーザの発案した提案を予測した結果の精度向上が，一見ユーザの発案した提案の予測精度の向上と変わらなければ，そのプロジェクトでは，一見ユーザとコアユーザは平等に扱われている可能性が高い．一方，コアユーザの発案した提案の予測精度の向上が一見ユーザの発案した提案の予測精度の向上よりも低くなれば，コアユーザの発案した提案の採否の判断にユーザの評判によるバイアスが存在する可能性がある．

5.1.3 項で述べた評価方法に従って，表 1 に示したコアユーザの発案した提案を用いて平均適合率を求める．コアユーザが発案した提案を無秩序に並べた場合と提案手法を用いて並べた場合の平均適合率を**図 5** に示す．図 5 から，すべてのプロジェクトについて，提案手法を用いることで提案の平均適合率は向上していることが分かる．しかし，図 4 と比較すると提案手法を用いたことによる平均適合率の向上が少ない．これは，ユーザの評判によるバイアスが提案の採否に影響を与えているためだと考える．そこで，指標を算出した提案のやりとりを分析することで，提案の採否とユーザの評判によるバイアスの関係を考察する．

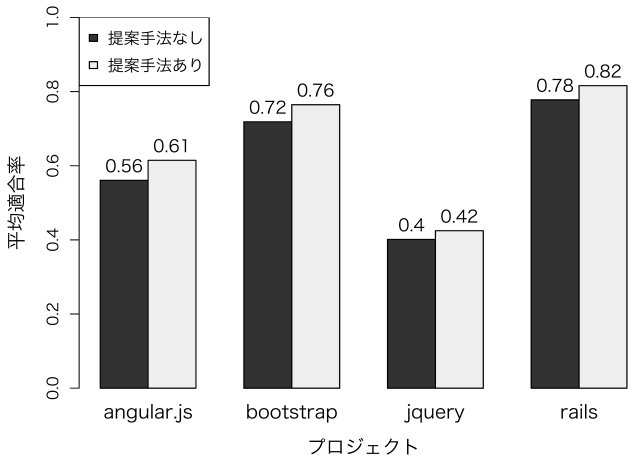

図 5 提案手法を用いた場合と用いない場合の提案の平均適合率（コアユーザ）

Fig. 5 Average precisions for ranked contributions with/without our method (core users).

5.2.2 予測結果の分析

コアユーザの発案した提案の採否の判断に，ユーザの評判によるバイアスが存在しているかを確かめるために，コアユーザの発案した提案について，算出された指標と実際の採否を比較する．たとえば，提案の指標が非常に低い，つまり有益である可能性が低いにもかかわらず，実際には採用されているような提案は，提案を発案したユーザの評判が採否に影響を与えている可能性がある．このような提案について，提案の指標が低く算出された原因と実際に採用された原因を提案にまつわるユーザ間のやりとりから分析する．また，同様に一見ユーザの発案した提案も分析し，提案を発案したユーザによって，提案への対応に違いがあるか考察する．

5.2.3 分析結果

5.2.1 項で指標を算出した提案を分析した結果，提案の良し悪しの指標が低いにもかかわらず，実際には採用されている提案には，以下のケースが存在した．

(1) 内容を十分に吟味されず採用された提案
指標が低く算出された提案の中には，プロジェクトオーナによって十分に内容を吟味されず採用されたものが含まれている．たとえば，angular.js には，提案が採用された後，プロジェクトオーナが提案に関するテストを十分にレビューしなかったことを述べている提案がある．この提案は，その後バグが発見され，プロジェクトオーナによって修正されている．jquery には，内容に不備があることをプロジェクトオーナに指摘されているにもかかわらず，ひとまず採用されている提案もある．プロジェクトオーナは，この提案に関する議論の中で，今後当該部分を改善することを述べている．また，bootstrap には，プロジェクトオーナが議論前に提案を採用したことに対し，採用する前にレビューをするべきだと提案発案者が指摘している提案も存在する．

(2) 議論の中で変更内容が修正された提案
コアユーザの発案した提案には，提案の内容に問題があるが，プロジェクトオーナが提案を即座に拒否するのではなく，提案の変更内容の修正を促し，最終的に提案を採用する場合がある．一方，一見ユーザの発案した提案では，提案の内容が良いか悪いかで即座に採否を判断されるケースが多い．これは，一見ユーザの発案する提案の質が両極端な場合が多いだけでなく，以前プロジェクトに貢献しているコアユーザを無下にできないという心理が働いている可能性がある．

上記のような提案の存在は，誰が提案を発案したかで提案の扱いに違いがあり，提案の採否の判断がユーザの評判によって左右されることを示している．このような提案の良し悪しの指標が低く算出されることで，提案の良し悪しの基準からユーザの評判によるバイアスを排除でき，提案の公平な評価とプロジェクトの健全性の確保につながる．

一方，コアユーザの発案した提案のうち，提案の良し悪しの指標が低いにもかかわらず，実際には採用されている提案には，プロジェクトオーナが発案した提案も多く含まれていた．プロジェクトによっては，プロジェクトオーナ自ら提案を発案し，ユーザ間で議論してから変更を取り込むことで，開発過程の透明化を図る．プロジェクトオーナの発案した提案には，変更行数が非常に多い提案などの一見ユーザの場合だと拒否されやすい提案が含まれる．このような提案は実際に採用されるべき提案であるが，特定のユーザが発案したからこそ採用される提案であると考える．したがって，プロジェクトオーナの発案した提案は，その他の提案と区別をする必要がある．

6. おわりに

本論文では，ソーシャルコーディングにおける提案の良し悪しの指標をユーザの評判によらず算出し，有益な提案を抽出する手法について述べた．具体的には，提案にまつわるユーザの行動を学習し，ユーザの評判によるバイアスを排除した提案の良し悪しの予測手法を述べた．提案手法では，一見ユーザの発案した提案のみを学習データとして用いることで，算出される指標からユーザの評判によるバイアスを排除した．また，評価の結果，提案手法を用いた場合の平均適合率は，提案手法を用いない場合と比べて最高で 0.16 向上した．これは，プロジェクトオーナが実際に抽出できる有益提案数の期待値を 17.3 個から 26.9 個に増加させることを意味する．最後に，コアユーザの発案した提案の採否にユーザの評判によるバイアスがどのように影響しているか分析した．分析の結果，コアユーザの発案した提案の中には，本来ならば採用するべきではない提案にもかかわらず，採用されたと思われる提案が存在した．提案手法がこのような提案の良し悪しの指標を低く算出することを確認した．

ユーザの評判によるバイアスが原因で，本来採用されるべきではない提案が増加すれば，プロジェクトの運営がユーザ間の馴れ合いに左右され，ユーザから反感を買う恐れがある．提案手法により，これらの提案の指標を低く算出することができれば，ユーザ間の馴れ合いを是正し，プロジェクトの健全性を確保できる．

残された課題として，以下の 3 つがある．1 つ目は，提案の指標と実際の採否の関係の詳細な分析である．本論文では，コアユーザの評価結果から，指標と実際の採否が大きく異なる提案に焦点を当て，提案にまつわるやりとりから提案の特徴を分析した．今後は，一見ユーザとコアユーザの両方について，指標と実際の採否の関係を詳細に分析する必要がある．たとえば，一見ユーザか否かと指標の大小，および実際の採否の組合せによって，どのようなケースの提案が，どの程度の割合で存在するかを確かめる．2 つ目は，拒否された提案へのフォローである．本論文では，提案手法を用いることで，どれだけ有益提案を効率的に抽出できるかに着目したが，コミュニティの活性化の観点からは，無益提案に対してどのように振る舞うかも重要である．無益提案に対して，何が原因で拒否するのか，どうすれば採用されるのかを説明すれば，たとえ提案が拒否されても，ユーザの参加意欲は削がれない．しかし，このような説明を提案 1 つ 1 つに行うのは，プロジェクトオーナにとって非常に手間である．そこで，提案手法で算出される提案の指標が低い場合，指標が低く算出された原因を自動で提示すれば，プロジェクトオーナの負担の軽減とユーザの参加意欲の持続につながる．そして，提案を発案する前に，提案の問題点を提示できれば，発案される提案全体の質の向上が期待できる．3 つ目は，学習に用いる新たな素性の検討である．たとえば，commit 数ではなく，採用された提案数に基づく活発度を素性として検討できる．提案は明確に論理的な意味を持つ単位で発案されるため，開発者によって粒度に差が出にくいが，commit は同様の変更であっても開発者によって粒度が異なる．プロジェクトの参加人数やソースコードの規模が大きくなるほど，commit を論理的に最小限度の意味を持つ単位で作成することが推奨されるため，commit の粒度の違いが提案手法の精度に与える影響は小さいと考えるが，提案数を基に活発度を用いることでより影響を小さくできると考える．

謝辞 本研究の一部は，科学研究費補助金・基盤研究 (C)（課題番号：26330224）による研究費を得て実施した．

参考文献

[1] GitHub, Inc.: GitHub, GitHub, Inc. (online), available from ⟨https://github.com/⟩ (accessed 2016-12-17).
[2] Dabbish, L., Stuart, C., Tsay, J. and Herbsleb, J.: Social coding in GitHub: Transparency and collaboration in an open software repository, *Proc. ACM 2012 Conference on Computer Supported Cooperative Work*, pp.1277–1286 (2012).
[3] GitHub, Inc.: About, GitHub, Inc. (online), available from ⟨https://github.com/about/⟩ (accessed 2017-05-1).
[4] 吉川友也，岩田具治，澤田 宏：ソーシャルメディア上の協働：ソーシャルコーディングにおける成功するプロジェクトの要因分析，日本データベース 学会 論文誌= DBSJ Journal，Vol.12, No.3, pp.19–24 (2014).
[5] Jason Tsay, L.D. and Herbsleb, J.: Influence of Social and Technical Factors for Evaluating Contribution in GitHub, *Proc. 36th International Conference on Software Engineering*, pp.356–366 (2014).
[6] Driessen, V.: A successful git branching model, (online), available from ⟨http://nvie.com/posts/a-successful-git-branching-model/⟩ (accessed 2016-12-17).

[7] Marlow, J. and Dabbish, L.: Activity Traces and Signals in Software Developer Recruitment and Hiring, *Proc. CSCW '13*, pp.145–156 (2013).
[8] Marlow, J., Dabbish, L. and Herbsleb, J.: Impression Formation in Online Peer Production: Activity Traces and Personal Profiles in GitHub, *Proc. CSCW '13*, pp.117–128 (2013).
[9] Datta, S., Majumder, A. and Naidu, K.: Capacitated Team Formation Problem on Social Networks, *Proc. KDD '12*, pp.1005–1013 (2012).
[10] Rahman, M.M. and Roy, C.K.: An insight into the pull request of GitHub, *Proc. 11th Working Conference on Mining Software Repositoryes*, pp.372–381 (2014).
[11] Gousios, G.: The GHTorent Dataset and Tool Suite, *Proc. 10th Working Conference on Mining Software Repositories*, *MSR '13*, pp.233–236, IEEE Press (2013).
[12] Georgios Gousios, M.P. and van Deursen, A.: An Exploratory Study of the Pull-based Software Development Model, *Proc. 36th International Conference on Software Engineering*, pp.345–355 (2014).
[13] Weißgerber, P., Neu, D. and Diehl, S.: Small patches get in!, *Proc. 2008 International Working Conference on Mining Software Repositories*, pp.67–76 (2008).

江見 圭祐 （学生会員）

平成 28 年岡山大学工学部情報系学科卒業．同年同大学大学院電子情報システム工学専攻博士前期課程入学．現在，同課程においてグループウェアサービスを研究中．

乃村 能成 （正会員）

平成 5 年九州大学工学部電子工学科卒業．平成 7 年同大学大学院情報工学専攻修士課程修了．同年九州大学工学部助手を経て，現在，岡山大学工学部准教授．博士（情報科学）．本会シニア会員．

谷口 秀夫 （正会員）

昭和 53 年九州大学工学部電子工学科卒業．昭和 55 年同大学大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．NTT データ通信（株），九州大学を経て，現在，岡山大学教授．博士（工学）．電子情報通信学会，ACM 各会員．本会フェロー．