Vol.2013-DPS-157 No.4
2013/10/17

ソーシャルコーディングにおける有益な提案の抽出について

檀上 未来[1]　乃村 能成[1]　谷口 秀夫[1]

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は，プロジェクトに対する理解度の低い会員が，過去に議論された内容と重複する無益な提案をすることがある．このような無益な提案はプロジェクトオーナーの手間を増やし，プロジェクトの進行の妨げになる．こうしたプロジェクトオーナーに対する負担を軽減するため，プロジェクトにとって有益な提案を抽出し，提示することは有用であると考えられる．本稿では，ソーシャルコーディングサービスである GitHub を対象として，著名なプロジェクト内で有益な提案が占める割合を調査した．調査の結果，プロジェクト内の有益な提案が占める割合は 50 %を上回っていないことがわかった．また，有益な提案の特徴について考察し，GitHub のシステムから機械的に有益な提案を取得する方式を示した．提案方式に基づいて GitHub 上のプロジェクトの提案数と有益な提案数の推移について考察した．考察の結果，有益な提案が占める割合は，開発初期をのぞけばプロジェクト全体を通してほぼ一定していることがわかった．

キーワード：ソーシャルコーディング, GitHub

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コーディングに適用しようとする試みである．Facebook を代表とするソーシャルネットワークサービス（SNS）は，人と人とのつながりを促進し支援する会員制のサービスと定義される．SNS の会員は，自分が日々感じたことや体験したことを文章や写真で表現して公開する．それに対して他会員からフィードバック（コメント）が付く．こうした相互のやりとりによって，会員同士の交流が深まる．SNS は，ビジネスにも活用されており，社内における部署を越えた交流，人材ネットワークの把握，アイデアの共有に役立っている [2]．NTT 東日本における社内 SNS Sati が事例としてあげられる．

ソーシャルコーディングでは，こうした交流をブログや写真ではなく，ソースコードを使って促進する．SNS で自分の写真アルバムをいくつも公開するのと同じようにソフトウェアプロジェクトをいくつも公開し，それに対して「イイネ」が付けられる．ソースコードに対するレビューや改善案が掲示板のような形で付く．また，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ようなプロジェクトに提供しているのかをつぶさに見られる．互いのコーディングスキルや興味のある分野が一目で分かり，互いの評判を形成したり相互補助の促進につながる [3]．

こうした，ソーシャルコーディングの仕組みを提案し標榜する GitHub[1] は，2013 年 4 月現在で 350 万人の会員と 600 万のソフトウェアプロジェクトを擁するまでになっている [4]．GitHub は，ソースコードレビュー，バージョン管理，チケット管理，ソースコードへのパッチの提示，といったコーディングに関わる作業を SNS の操作になぞらえた簡単な操作として提供し，その過程を可視化し記録するシステムであるともいえる．

ソーシャルコーディングでは，ソーシャルネットワークサービスとよく似た問題が発生する．たとえば，会員間に上下関係がないので，どのプロジェクトに対しても自由に提案やソースコードへのパッチの提供ができる．ソースコードにパッチを取り込むのは，当該プロジェクトのオーナー（開設者）に限られるが，パッチ提供の過程にまつわるやりとりは，掲示板のように衆目に触れるので，プロジェクトオーナーといえども非合理なことはできない．つまりプロジェクトオーナーは，通常のソフトウェアプロジェクトのリーダよりも民主的で合理的な判断を社会的（ソーシャル）に期待される．したがって，プロジェクトオーナーにとって無益と思われる提案であっても無下に否定するわけにはいかず，それらへの対応による手間が増え，プロジェ

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

クトの進行を妨げたり，いわゆる「ソーシャル疲れ」という現象に陥ったりする [5].

また，プロジェクトに対する理解度の低い会員が，過去に議論された内容と重複する提案をすることがよくあり，それを指摘したり，複数の会員から同時期にほぼ同じ内容の提案を受け，それらの議論を調整することもある．

こうしたプロジェクトオーナーに対する負担を軽減するため，コーディングにおける有益な提案とそうでない提案を選別して事前に判定したり，よく似た提案どうしをまとめる作業を補助できれば，有用であると考えられる．

そこで，本稿では，まず 2 章でソーシャルコーディングにおけるソフトウェア開発の手順を解説した上で，そこで発生する上記問題点を明確にし，ユーザが出す「提案」を選別することが重要であることを説明する．3 章では，ソーシャルコーディングのソフトウェア開発プロジェクトにおいて，実際に有益な提案とそうでない提案の割合がどの程度かについて調査する．調査の目的について述べ，有益な提案の特徴を定めた上で調査を行い，その結果を考察する．4 章では，有益な提案の特徴がソーシャルコーディングで取得可能なデータでどう置き換えられるか議論し，実際にその条件で有益な提案を抽出して，有益な提案の特徴を表現できているか確かめる．最後に 5 章で実際のプロジェクトの過去の提案から有益な提案を自動抽出し，その結果について考察する．

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 でのソーシャルコーディングに参加するには，まず GitHub にユーザ登録する．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ーとなってソフトウェア開発プロジェクトを開設できる．プロジェクトは他のユーザにも公開される．また，個人で複数のプロジェクトを開設できる．開設したプロジェクトには，各プロジェクト専用に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，全ユーザが自由にコピー可能で，コピー元とコピー先のバージョンの派生関係がシステムで管理されている．これによって，コピーを改変した別バージョンを元プロジェクトのプロジェクトオーナーに提供しやすくなっている．また，人気のある（よくコピーされる）プロジェクトが一目で分かる．

(2) 全ユーザが自由に書き込める掲示板（issues）
issues によって，ソースコード上の不具合や新しい提案など，ソフトウェアに関する議論ができる．issues は，単なる掲示板ではなく，上記 repository と高度に連携する．具体的には，repository 中の特定バージョンを簡単に引用できたり，別のユーザがコピーして改変したバージョンと，元のバージョンとの差分を簡単に提示して議論できたりする．また，ソースコード取り込みの提案操作が issues に自動的に記録されるので，後からソースコード改変の経緯を追いかけやすい．

GitHub では他のユーザのプロジェクトを閲覧，検索できる．ソースコードを閲覧したり，issues にそのプロジェクトに対する質問や提案を自由に書き込める．

他のユーザのプロジェクトのソースコードに手を加えたい場合は，以下の手順を踏む．

(1) 元プロジェクトの repository をコピーして，元プロジェクトの派生プロジェクトを作成する．この操作を fork と呼ぶ．

(2) 派生プロジェクト（fork 先プロジェクト）は，自身のプロジェクトなので，自由にソースコードを改変できる．

(3) fork 元プロジェクトに改良後のソースコードを取り込んでもらうように要求する．具体的には，fork 元のプロジェクトの issues に fork 先プロジェクトにおける repository のバージョンを提示する．この操作を pull request 操作と呼ぶ．

fork 元のプロジェクトオーナーは，pull request に示されたソースコードを認めれば，受け入れ操作（merge）をするだけで当該ソースコードを簡単に取り込める．こうして，自身が改良したソースコードは fork 元（本家）プロジェクトの一部として取り込まれる．このようにして GitHub 上では開発者同士が掲示板でコミュニケーションを取りながらプロジェクトを進行し，かつその手続きは，システム化されている．

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．例えば，プロジェクトが fork（コピー）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 の回数や提案の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 [6]．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．例えば，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 [7]．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 [8]．また，プロジェクトがもつ掲示板は，質問や提案を書き込むことの他に，改良したソースコードを書き込んだり，バージョン管理システムへのリンクを示したりもできる．この

掲示板がもつ機能により，開発者はプロジェクトのソースコードに簡単にアクセスできる．このように，ツールや機能間の連携不足によって起きていたオープンソフトウェア開発でのコミュニケーション不足という問題点 [9] を解決できる連携されたシステムが，GitHub には備わっている．これらのシステムにより，ソーシャルなソフトウェア開発を促進している．

2.2 GitHub における提案

ソーシャルコーディングを支援するシステムは，プロジェクトへのバグの発見報告や新機能の提案を容易にする．本稿では，このような報告を提案と呼ぶ．GitHub では，提案には issue 機能と pull request 機能を使用する．ソースコードを伴わない提案は issue 機能を使用し，ソースコードを伴う提案は pull request 機能を使用する．issue 機能と pull request 機能によって報告された提案は，どちらも issues に記録される．開発者は，提案に用意される掲示板を利用して，提案の処遇について議論する．プロジェクトオーナーは議論の結果，提案の処遇が決定し議論が終了したと判断した場合，提案を close する．プロジェクトオーナーが close 後にさらに議論が必要だと判断した場合に，提案は reopen されることがある．

2.3 ソーシャルコーディングの問題点

ソーシャルコーディングでは，全ての開発者が自由に提案を作成できる．GitHub においては，全てのユーザが提案を作成可能で，全提案はシステム上で平等に扱われる．これは，ソーシャルコーディングによって得られる利点である．しかし，ソーシャルコーディングには欠点もある．ソーシャルコーディングでは，世界中の不特定多数の開発者がプロジェクトに参加するため，プロジェクトへの理解度の低い者も多い．また，プロジェクトの参加者は時間を経て変化する．このため，以下に示す問題を持った提案が作成されることがある．

(1) 過去の提案と内容が重複した提案
過去に作成された提案と内容が重複した提案が作成される場合がある．プロジェクトオーナーは議論の分散を防ぐため，重複した提案を終了させる．

(2) 提案者のミスが原因の不具合の報告
提案者のプロジェクトに対する理解度が低いため，提案者のミスが原因の不具合をソフトウェアのバグとして報告する場合がある．例えば，計算機にソフトウェアの動作に必要なパッケージが存在しない時に，ソフトウェアが動作しないことをバグとして報告することがある．プロジェクトオーナーは提案者のミスが原因であることを指摘し，解決方法を提示して提案を終了させる．

(3) プロジェクトで扱うべきでない問題や機能の報告
提案者のプロジェクトの方針に対する理解度が低いため，プロジェクトで扱うべきでない事項について提案する場合がある．プロジェクトオーナーはプロジェクトの方針を述べ，提案を終了させる．提案された事項を扱う他のプロジェクトがあれば，そのプロジェクトを提示する場合もある．

これらの提案は，プロジェクトに取り入れられないばかりか，プロジェクトオーナーや他の開発者の手間を増やす．本稿では，このような提案を無益な提案と呼ぶ．対して，プロジェクトに取り込まれる提案や，多くの開発者によって活発に議論が交わされる提案を有益な提案と呼ぶ．

一般的に，大量のタスクの中に必要のないタスクが存在する状態は，全体の進行の遅れの原因となる．このような状態を解決するには，必要のないタスクの量と，それによって起こる遅れの深刻さを自覚することが必要である．このため，プロジェクトオーナーや開発者が，現在プロジェクトが抱えている問題のなかで，無益な提案と有益な提案がどれほどあるか把握しておくことは重要である．

3. GitHub のプロジェクトにおける有益な提案の割合

3.1 調査の目的

GitHub で著名なプロジェクトでの提案を見てみると，過去の提案との重複や，プロジェクトで取り扱うべきでない提案など，無益な提案が多かった．プロジェクトオーナーは無益な提案に対して重複を指摘したり，議論を調整する必要があり手間がかかる．GitHub では，全ユーザ間の掲示板上でのやり取りは誰でも閲覧，参加が可能で，かつシステムのデータベースで管理される．このため，プロジェクトオーナーは無益な提案であっても，手間をかけ，合理的な判断を下す必要がある．しかし，著名なプロジェクトは，プロジェクトに対する提案の数が多い．このため，著名なプロジェクトのプロジェクトオーナーは，提案が有益かどうかを判断する作業に多くの時間を費やす必要がある．こうしたプロジェクトオーナーに対する負担を軽減するため，プロジェクトにとって有益な提案を抽出し，提示することは有用であると考えられる．ここでは，有益な提案の数を自動抽出するために，有益な提案の特徴を明らかにする．このためまず，調査対象のプロジェクトを選定し．調査対象のプロジェクトについて有益な提案と無益な提案の割合を調査する．調査の結果から，2.3 節で挙げた問題が実際に存在していることを確認する．その後，有益な提案の特徴について考察する．

3.2 調査方法

GitHub 上のプロジェクトについて，有益な提案数を調査する．本稿では，この調査により抽出された有益な提案数を有益提案数と呼ぶ．ここでは，fork 数が多いプロジェク

表 1　調査対象のプロジェクト

プロジェクト名	概要	提案データ取得日付	fork 数
bootstrap	Twitter 社が開発するフレームワーク	2013 年 7 月 30 日	19,829
rails	Ruby で開発されたフレームワーク	2013 年 8 月 19 日	6,498
html5-boilerplate	HTML5 のためのフレームワーク	2013 年 9 月 5 日	5,398

表 2　最新 100 提案の調査結果

プロジェクト名	有益提案数/調査提案数	特徴 1 のみ満たす	特徴 2 のみ満たす	どちらも満たす
bootstrap	31/100	29	0	2
rails	47/100	41	1	5
html5-boilerplate	27/100	17	7	3

トのうち上位 3 つを調査対象のプロジェクトとして選定する．表 **1** は，調査対象のプロジェクト名，各プロジェクトの概要，提案に関するデータを取得した日付，および fork 数を示す．fork 数は，2013 年 9 月 16 日時点の値である．調査対象のプロジェクトは，fork 数が多い順に bootstrap，rails，html5-boilerplate である．

調査対象のプロジェクトについて，有益な提案数を調査した．調査は，bootstrap は 2013 年 7 月 30 日，rails は 2013 年 8 月 19 日，html5-boilerplate は 2013 年 9 月 5 日の最新 100 の close された提案について行った．審議が継続中の（open 状態の）提案では，有益かどうかが判断できない場合があるため，close された提案のみを調査対象とした．調査では，著者らがブラウザ上の UI を見て，個々の提案について有益かどうか判断した．ここでは，以下の特徴のうち少なくともひとつを満たすものを有益な提案とした．

特徴 1　提案が示すアイデアやソースコードがプロジェクトに取り込まれている

提案は，プロジェクトにアイデアやソースコードを取り込んでもらうために作成するものである．このため，プロジェクトに取り込まれた提案は有益であるといえる．プロジェクトに取り込まれた提案とは，以下の 2 つの場合を指す．

(1) 提案に対して開発者が提供したソースコードが，プロジェクトに取り込まれる
(2) 提案に対してプロジェクトオーナーがソースコードを作成し，直接プロジェクトに追加する

特徴 2　提案に対して活発に議論が行われた

プロジェクトに取り込まれなかった提案でも，提案を採用すべきか否かについて多くのユーザによって活発に議論される場合がある．その議論はプロジェクトにとって有益であり，その提案も有益であると言える．活発に議論された提案の特徴としては，提案に対するコメント数と参加者数が多いことが挙げられる．しかし，ユーザのミスが原因のバグ報告があった時，提案が有益でないにも関わらず，その解決法について多くの参加者によりコメントが作成される場合がある．このため，コメントが多く参加者が少ないときや，特定のユーザのコメントのみが多数存在するときは，コメントの大半が質問者への解答であることが多いため，有益な提案とならない．また，提案が close した後に活発に議論されることがある．このため，コメント数と参加者数が全体では少ない提案でも，close 後にコメント数が伸びているものは有益な提案となる．これらを考慮して，活発に議論が行われた提案とは，以下の 2 つの場合を指す．

(1) 提案に対するコメント数と参加者数が多く，かつコメントが特定のユーザのものに集中していない
(2) 提案が close した後にコメント数が増加している

3.3　調査結果

表 1 に示す調査対象のプロジェクト内の提案における有益提案数の調査結果を述べる．表 **2** は，調査対象のプロジェクト内の提案における有益な提案が占める割合と，有益な提案のうち，3.2 節で示した有益な提案の各特徴を満たす提案の数を示す．まず，有益な提案が占める割合は，bootstrap，rails，html5-boilerplate において，それぞれ 31 %，47 %，27 %であり，どのプロジェクトにおいても 50 %を下回っている．GitHub のようなソーシャルコーディングでは，提案を作成する前にその提案がプロジェクトにとって有益かどうか話し合う場はなく，提案の作成は個人の判断に委ねられる．プロジェクトに対する理解度の低いユーザが提案を作成することも多いため，無益な提案が増える傾向にあると考えられる．

次に，有益な提案のうち，有益な提案の各特徴を満たす提案の数を見ると，特徴 2 のみを満たす提案数は bootstrap と rails でそれぞれ 0，1 である．これは，特徴 2 によらずともほとんど全ての有益な提案を抽出できることを示している．しかし，html5-boilerplate では特徴 2 のみを満たす提案数は 7 である．これは，特徴 1 のみを満たす提案数が 17 であることに比べても少なくない．プロジェクトの方針によっては特徴 1 だけでは有益な提案が抽出できないことがわかる．このことから，特徴 1 と特徴 2 はどちらも有益な提案の抽出に有用であると言える．

表 3　各条件を満たす有益提案数

有益な条件の特徴	判定有益提案数/有益提案数
条件 A pull request が merge されている	54/53
条件 B commit から参照されている	48/46
条件 C commit message により close されている	21/21
条件 D close 後 reopen されている	7/4
重複をのぞいた総有益提案数の割合	105/98

表 4　有益提案数の判定結果

		調査結果	
		有益	無益
判定結果	有益	93	5（偽陽性）
	無益	12（偽陰性）	190

4. 有益な提案の自動抽出

4.1 自動抽出

3.3 節で述べたように，調査の結果，GitHub で fork 数の多いプロジェクトでは，有益な提案が占める割合が低いことが分かった．3.1 節で述べたように，プロジェクトオーナーや開発者が，プロジェクト内の有益な提案数と無益な提案数について把握しておくことは重要である．有益な提案数と無益な提案数を継続的に把握しておくことで，これらをプロジェクトの健全性の指標のひとつとして活用できる．しかし，個々の提案について有益かどうかを判断し，有益な提案数を把握するのは時間がかかり，手間である．実際，著者らが最新 300 の提案について有益な提案数を把握するには 2 人日を要した．プロジェクト内の有益な提案数の把握にこのような手間がかかる場合，継続的に観測することは難しい．

ここで，プロジェクト内の有益な提案を自動的に抽出する方法を検討する．3 章で述べた調査では GitHub を利用した．GitHub では，プロジェクトがもつ issue，pull request，あるいは fork などの発生履歴がシステム上で管理されている．例えば，issue ではタイトル，コメント数，参照されたコミット，作成日時などが管理されている．有益な提案に共通の特徴があり，その特徴が GitHub で管理されているデータとして存在すれば，GitHub から情報を取得し解析することで，有益な提案を抽出できると考えられる．次節では，有益な提案の特徴を GitHub のシステムから取得可能なデータに置き換える．

4.2 有益な提案の条件

3.2 節で述べた 2 つの特徴を GitHub のシステムから取得可能なデータに置き換えると，以下のような条件になると考えられる．

条件 **A**　pull request が merge されている

プロジェクトオーナーが，他の開発者の pull request によって提供されたソースコードをプロジェクトに取り込むことを merge と呼ぶ．merge 操作は，GitHub のシステム上で管理される．merge 操作は，pull request による提案がプロジェクトに取り込まれていることを示す．このため，特徴 1（1）の「提案に対して開発者が提供したソースコードが，プロジェクトに取り込まれる」に当てはまり，有益な提案と判定できる．

条件 **B**　提案が commit から参照されている

GitHub では，プロジェクト内のソースコードの変更履歴を commit という単位で管理している．プロジェクトオーナーが開発を進めた場合は，pull request 機能を使用せずに，commit をプロジェクトに加える事でプロジェクトのソースコードを変更する場合が多い．提案がどの commit によって対処されたかを示されている場合，提案が当該 commit から参照されている．これは，特徴 1（2）の「提案に対してプロジェクトオーナーがソースコードを作成し，直接プロジェクトに追加する」に当てはまり，有益な提案と判定できる．

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

プロジェクト内のソースコードの変更履歴をまとめた commit を作成する際に，その commit でどのような変更があったのか見分けるために commit message を記述できる．GitHub では，commit message に，その commit が対処した issue について記述し，その commit をプロジェクトに加えると，issue で提案した問題が解決されたと判断され，自動的に issue を終了する仕組みがある．GitHub では，issue がどのような方法で終了されたかも管理されている．commit message によって終了した issue は，有益なソースコードが提供された issue だと言える．プロジェクトオーナー以外がソースコードを提供する場合は pull request 機能を使うため，この方法で提供されたソースコードオーナー自身が提供したものだと言える．このため，特徴 1（2）の「提案に対してプロジェクトオーナーがソースコードを作成し，直接プロジェクトに追加する」に当てはまり，有益な提案と判定できる．

条件 **D**　提案が close 後 reopen されている

GitHub には open 状態の提案の一覧と close した提案の一覧を分けて表示する機能がある．開発者は議論が続いている open 状態の提案の一覧を使用して，議論に加わることが多い．提案が close した後で，更に議論

表 5　調査結果と判定結果の不一致の原因

不一致の原因	詳細	提案数	総提案数
ユーザのミス	(1) 必要のない reopen	2	
	(2) 提案の二重投稿	2	
	(3) プロジェクトオーナーの判断ミス	1	5
条件の設定漏れ	(4) merge された pull request からの参照	1	
	(5) コメントから commit への参照	3	
	(6) 多数のコメントと参加者	7	11
その他	(7) GitHub のバグ	1	1

が必要だとプロジェクトオーナーが判断した場合，議論に加わる開発者を増やすため，もう一度 open 状態に戻す．この操作を reopen と呼ぶ．このため，reopen された提案は活発な議論が行われるものが多い．特徴 2(2) の「提案が close した後にコメント数が増加している」に当てはまり，有益な提案と判定できる．

本稿では，これらの条件のうち少なくともひとつを満たすものを判定有益提案と呼ぶ．調査対象のプロジェクト 3 つにおいて，判定有益提案数を求めた．**表 3** は，3 つのプロジェクトについて，各条件における有益提案数と判定有益提案数を示したものである．有益提案数と判定有益提案数が同等の値であれば，有益な提案を示す条件として，ここで挙げた条件が正しいと言える．表 3 を見ると，条件 A，B，C，および D のそれぞれの判定有益提案数と有益提案数の差ををを計算すると，条件 A から順に 1，2，0，3 であり，ほぼ一致していると言える．これらの結果から，条件 A，B，C，および D は 3.2 節の特徴 1 と 2 をおおむね表現できていると言える．

表 4 は，表 1 で挙げた 3 つの調査対象のプロジェクトにおける調査結果の有益提案数と判定結果の有益提案数の関係を示している．3 つのプロジェクトの結果を統一しているため，全ての値を合計すると，調査対象プロジェクトの提案の総数である 300 を示す．表 4 の左上は判定結果が有益なものの内，調査結果も有益だったものを示しており，右下は判定結果が無益なものの内，調査結果も無益だったものを示している．これらの提案の数は，調査対象の 300 の提案のうち 283 であり，94.3 %を占める．これらの提案は正しく判定できているといえる．一方，正しく判定できていない提案も存在する．表 4 の右上は，判定結果が有益なもののうち，調査結果では無益だったものを示しており，5 個だった．これらの提案は偽陽性をもつ．また，表 4 の左下は，判定結果が無益なものの内，調査結果では有益だったものを示しており，このような提案の数は 12 個だった．これらの提案は偽陰性をもつ．**表 5** は，調査結果と判定結果の不一致の原因を示している．不一致の原因がユーザのミスである提案は偽陽性をもっており，不一致の原因が条件の設定漏れである提案は偽陰性をもっている．また，その他の不一致の原因として，GitHub のバグがあった．このうち，偽陰性をもつ提案は有益な提案の条件がより正確に有益な提案の特徴を表すものであれば，有益な提案として抽出できる．表 5 中の「(4) merge された pull request からの参照」は，参照された側の提案は参照された履歴をデータとして持っていないため，抽出できなかった．また，「(5) コメントから commit への参照」は，コメント内で commit を参照している提案が抽出できていないことに起因する．プロジェクトオーナー自身が追加したソースコードを参照するために，コメント内で commit を参照する場合がある．しかし，この参照は単なるの文字列としてのみ扱われ，システム上で管理される単独の要素として扱われていない．このため，今回の判定では抽出できなかった．「(6) 多数のコメントと参加者」について，提案はコメント数や参加者数をデータとして持っているが，有益かどうかの判定基準がわからなかった．このため，今回の判定では抽出できなかった．(6) に当てはまる提案は，調査対象の 300 の提案のうち 2.3 %を占めている．この問題を解決すれば，判定結果の正しい提案は 94.3 %から 96.7 %まで増加する．

今後，さらに多くのプロジェクトについても調査を行い，他にも抽出に使用可能な条件がないか考察する必要がある．

5. 有益な提案数の推移

4.2 節で述べた有益な提案の条件に基づいて，プロジェクト内の提案数と判定有益提案が占める割合の推移を抽出することを試みる．

プロジェクト内の提案数と判定有益提案が占める割合の推移に関して，以下の 2 つの傾向があると推測できる．

(1) プロジェクトの開始時は，プロジェクトを熟知した特定のメンバが提案を作成するため有益な提案が多い
(2) 新バージョンリリース時は，プロジェクトを熟知していない多数のユーザが提案を作成するため，無益な提案が増加する

プロジェクトが上記の条件を満たしているか，自動抽出したデータから検証してみる．

有益提案数の調査で使用した bootstrap において，プロジェクト内の提案数と判定有益提案数が占める割合の推移を検証した．bootstrap は 2011 年 8 月に GitHub での開発を開始した．3 章の調査では，bootstrap の 100 の提案について調査した．この検証では，2013 年 9 月 5 日までに

close された 10,303 の提案について，4.2 節で定めた条件を使用して判定有益提案を抽出した．bootstrap における提案数内の判定有益提案数が占める割合は，24 %であった．

図 **1** は，bootstrap の提案数と判定有益提案が占める割合の推移を示している．図 1 において，横軸は月を表し，bootstrap が GitHub を使用し始めた 2011 年 8 月 19 日から 2013 年 9 月 5 日までの時系列を表示している．縦軸（左側）は，提案数を示し，縦軸（右側）は，判定有益提案が占める割合を示している．棒グラフは各月における提案数の推移を示し，棒グラフの下部は判定有益提案数，上部は判定無益提案数を示している．折れ線グラフは各月における判定有益提案が占める割合を示している．

図 1 から，プロジェクトの初動時期である 2011 年 8 月から 2011 年 11 月までの 4 ヶ月は判定有益提案が占める割合が 5 割を超えていることがわかる．これは，「プロジェクトの開始時はプロジェクトを熟知した特定のメンバが提案を作成するため有益な提案が多い」に当てはまる．

次に，2012 年 1 月，2012 年 8 月，2013 年 8 月など，提案数が急激に増加している月があるとわかる．これは，新バージョンのリリースが影響していると考えられる．表 **6** は bootstrap の主なバージョンとそのリリース日を示す．図 1 の折れ線グラフの内で丸く塗りつぶされた月は，主なバージョンのリリースがあったことを示す．表 6 と図 1 より，提案数が増加している 2012 年 1 月，2012 年 8 月，2013 年 8 月には，それぞれ version2.0.0，version2.1.0，version3.0.0 がリリースされたことがわかる．このように，提案数が急激に増加している月はバージョンのリリースと重なっていることがわかる．また，これらの月には，判定無益提案数も増加している．これは，「新バージョンリリース時は，プロジェクトを熟知していない不特定多数のユーザが提案を作成するため，無益な提案が増加する」に当てはまる．

以上から，bootstrap は前述した 2 つの予想を満たしており，判定有益提案の判定条件によって，おおよそ正しい判定有益提案数の推移がつかめることがわかった．

また，図 1 から，提案数が急激に増加している月は判定無益提案数も増加していることがわかる．しかし，判定有益提案が占める割合は，減少している傾向にはない．2012 年 4 月以降を見ると，提案数が増加している月は判定有益提案が占める割合も増加している．また，2012 年 1 月に提案数が増加して以降，判定有益提案が占める割合はほぼ 2 割前後で安定していることが分かる．これは，バージョンアップ等により提案数が変動しても，判定有益提案はほぼ一定であることを示している．

6. おわりに

ソーシャルコーディングでは無益な提案が多く存在することを問題に挙げた．ソーシャルコーディングサービスである GitHub を対象として，著名なプロジェクト内で有益な提案が占める割合を調査した．本稿では著名なプロジェクトとして，bootstrap，rails，および html5-boilerplate をとり上げた．有益な提案の特徴として，以下の 2 点を挙げた．

(1) 提案が示すアイデアやソースコードがプロジェクトに取り込まれている
(2) 提案に対して活発に議論が行われた

著名なプロジェクトでは，これらの特徴を満たす有益な提案が占める割合が 50 %を下回っていることを示した．本稿での調査で用いた有益な提案の特徴を GitHub のシステムから取得可能なデータに置き換え，これらの条件が有益な提案の特徴を表現できていることを確かめた．GitHub 上でソフトウェア開発を行うプロジェクトとして bootstrap を取り上げ，有益な提案の条件を使用して，提案数と有益な提案が占める割合の推移について調査した．この結果，bootstrap における提案数内の有益提案が占める割合は 24 %で，開発初期をのぞけば，プロジェクト全体を通してほぼ一定していることがわかった．

残された課題として，有益な提案の更なる条件の発見と，提案作成時に有益かどうか予測可能なシステムの作成の 2 つがある．

謝辞　岡山大学工学部情報工学科の北垣千拡氏には，調査データの集計作業において協力いただいた．また，本研究の一部は，NTT サービスエボリューション研究所の提供する研究設備，回線を活用した．ここに記して謝意を示す．

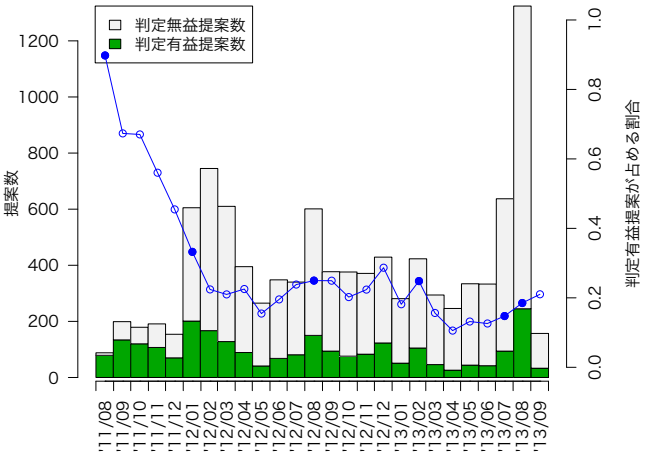

図 **1**　bootstrap の提案数と判定有益提案が占める割合の推移

表 **6**　bootstrap の主なバージョンアップとそのリリース日

リリース日	バージョン名
2011 年 8 月 18 日	version 1.0.0
2012 年 1 月 31 日	version 2.0.0
2012 年 8 月 20 日	version 2.1.0
2013 年 2 月 7 日	version 2.3.0
2013 年 7 月 27 日	version 3.0.0 リリース候補版
2013 年 8 月 19 日	version 3.0.0

参考文献

[1] GitHub, Inc.：GitHub(online)，入手先 ⟨https://github.com/⟩(2013.09.15).
[2] 総務省：ビジネスブログ・ビジネスＳＮＳ 活用事例集，総務省 (2005).
[3] Dabbish, L., Stuart, C., Tsay, J. and Herbsleb, J.: Social coding in GitHub: transparency and collaboration in an open software repository, *Proc. CSCW '12*, pp.1277-1286, ACM(2012).
[4] mojombo：Five years(online)，入手先 ⟨https://github.com/blog/1470-ve-years⟩(2013.09.15).
[5] Yamakami, T.: Towards understanding SNS fatigue: exploration of social experience in the Virtual World, *Proc. 2012 7th International Conference on Computing and Convergence Technology (ICCCT 2012)*, pp.203-207(2012).
[6] Marlow, J. and Dabbish, L.: Activity Traces and Signals in Software Developer Recruitment and Hiring, *Proc. CSCW '13*, pp.145-156, ACM(2013).
[7] 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, ACM(2013).
[8] Datta, S., Majumder, A. and Naidu, K.: Capacitated Team Formation Problem on Social Networks, *Proc. KDD '12*, pp.1005-1013, ACM(2012).
[9] 石川武志，山本哲男，松下誠，井上克郎：ソフトウェア開発時における版管理システムを利用したコミュニケーション支援システムの提案，情報処理学会研究報告，Vol.2001，No.92，pp.23-30（2001）.