# ソーシャルコーディングにおけるプロジェクト成功の法則

吉川 友也[1,a)]　岩田 具治[2,b)]　澤田 宏[3,c)]

**概要**：ソーシャルコーディングサイト (SCS) は，ソフトウェア開発を Web 上で共有するためのサービスである．代表的な SCS は GitHub や Bitbucket であり，現在多くのオープンソースソフトウェアがこれらの上で開発，提供されている．SCS の特徴は，ソーシャルな環境で開発が行われる点である．すなわち，SCS では開発者をつなぐソーシャルネットワークが構築されており，それぞれの開発者が行ったプロジェクト作成やお気に入り登録等の行動が開発者間で共有される．また，プロジェクト外部の開発者がプロジェクトを改善できる．このようなソーシャルな環境がプロジェクトの人気度や生産性に影響を与える可能性がある．では，どのようなプロジェクトが SCS で成功するのだろうか．本論文では GitHub のデータを使い，お気に入り登録の多いプロジェクトやコードの更新頻度の高いプロジェクトに見られる特徴を分析する．また，この分析を通じて，プロジェクトを成功させる法則について考察する．

## 1. はじめに

ソーシャルコーディングサイト (**S**ocial **C**oding **S**ite; SCS) は，ソフトウェア開発プロジェクトを Web 上で共有するためのサービスである．代表的な SCS は，GitHub[*1] と Bitbucket[*2] であり，これらのサービスは，Web 上での開発やプロジェクト管理を支援する機能を携えている．利用者数は年々増加し，GitHub では 300 万ユーザ[*3]（2013 年 1 月 16 日時点），Bitbucket では 100 万ユーザ[*4]（2013 年 6 月 4 日時点）が利用している．

SCS では，開発者は自らがオーナーとなるプロジェクトを作り，その中でソフトウェア開発を行う．SCS の特徴は，ソーシャルな環境で開発が行われる点である．すなわち，SCS では開発者をつなぐソーシャルネットワークが構築されており，プロジェクト作成やお気に入り登録等の行動が開発者間で共有される．

本論文の目的は，ソーシャルコーディングにおけるプロジェクト成功の法則を見つけ出すことである．成功の指標として，プロジェクトの生産性を表すコードの更新頻度，プロジェクトの人気度を表すお気に入り数，プロジェクトのソーシャル度を表す外部からのコード変更回数を用いる．成功に繋がる要素を発見するために，1) プロジェクトメンバーの構成，2) プロジェクト外部の開発者への対応，3) プログラミング言語の選択に着目する．1) プロジェクトメンバーの構成では，プロジェクトメンバーのネットワーク的性質と成功の指標との相関を見る．また，メンバーの人数と生産性との関係からプロジェクトを進める上で効率的なメンバー数に関して議論する．さらに，仕事量の偏りと成功の関係についても議論する．2) プロジェクト外部の開発者への対応では，外部からのコード変更要求への対応の仕方が成功とどのように関係するのかを分析する．3) プログラミング言語の選択では，プロジェクトが作るプロダクトと成功の関係を分析するために，プロジェクトが使用するプログラミング言語と成功との関係を考察する．

研究プロジェクトやオープンソースソフトウェアプロジェクトなどを対象として，成功につながる要因を分析する研究が既に行われている [2], [8]．しかしながら，ソーシャルコーディングにおけるプロジェクトの成功を扱った論文はこの論文が初めてである．

以下では，2 節でソーシャルコーディングの説明と使用するデータセットの説明を行う．3 節では，ソーシャルコーディングの世界の概観を知るための基本分析を行い，4 節から 6 節で成功に繋がる要因を分析する．7 節では関連研究を紹介し，最後の 8 節で結論を述べる．

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[2] NTT コミュニケーション科学基礎研究所
NTT Communication Science Laboratories
[3] NTT サービスエボリューション研究所
NTT Service Evolution Laboratories
[a)] yuya-y@is.naist.jp
[b)] iwata.tomoharu@lab.ntt.co.jp
[c)] sawada.hiroshi@lab.ntt.co.jp

[*1] https://github.com
[*2] https://bitbucket.org
[*3] https://github.com/blog/1382-three-million-users
[*4] http://blog.bitbucket.org/2013/06/04/atlassian-bitbucket-passes-one-million-users/

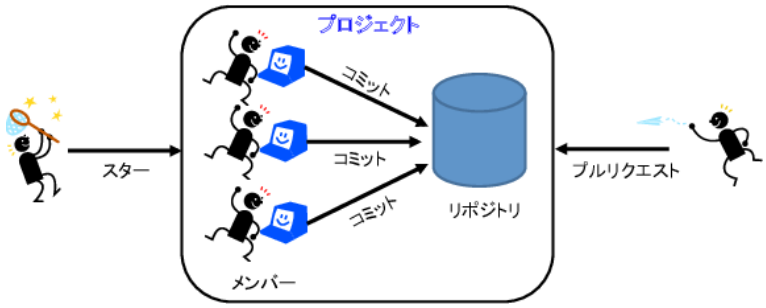



**図 1** ソーシャルコーディングの概略図

## 2. ソーシャルコーディング

### 2.1 ソーシャルコーディングとは

ソーシャルコーディングは Web 上のソーシャルな環境において行われるソフトウェア開発である．図 1 は，ソーシャルコーディングの概略図を示す．ソーシャルコーディングにおける開発は以下のように行われる．開発者（もしくは組織）は自らがオーナーとなるプロジェクトを作成する．そして，開発者はコードの追加・変更を行い，リポジトリと呼ばれるデータベースにコードを追加して，ソフトウェアを作り上げる．このコードを追加する作業を**コミット**と呼ぶ．ソーシャルコーディングにおける特徴の一つは，コミットの履歴などが Web 上で可視化されている点である．これによって，開発者間のやり取りやコードの変更が理解しやすくなっている．ここではオーナーが一人のケースを考えたが，オーナーと同等の権限を持つ開発者を複数人設定することも可能である．さらに，ソーシャルコーディングでは，プロジェクトのメンバーでない人も開発に参加できる．開発者は追加したいコードと共に**プルリクエスト**を送ることによって，自らがオーナーではないプロジェクトのソフトウェアの開発に参加できる．ただし，そのコードがプロジェクトに取り込まれるかどうかは，プロジェクトのメンバーの意思決定によって決まる．

### 2.2 データセット: GitHub

本論文では，代表的なソーシャルコーディングサイトである GitHub のデータを使用する．データは GitHub Archive[*5] より取得した．このデータには，2011 年 2 月から 2013 年 5 月までに行われたコミットやプルリクエストなどの公開されているイベントがすべて含まれている．その中から 30 回以上コミットされたプロジェクトを抽出し，各プロジェクトのメンバー情報やコミット，プルリクエスト，プロジェクトのブックマークを意味するスターなどの必要な情報を収集した．また，GitHub では好きな開発者の行動を追うために，フォローと呼ばれる仕組みが存在する．この仕組みによって作られたネットワークを構築するために，一回以上コミットかフォローした経験のある開発者の情報を収集した．表 1 は，収集した GitHub データセットの統計量を示す．

[*5] http://www.githubarchive.org/

**表 1** 収集した GitHub データセットの統計量

| | 数量 |
|---|---|
| プロジェクト数 | 317,077 |
| 総コミット数 | 41,720,139 |
| 総スター数 | 5,164,934 |
| 総プルリクエスト数 | 1,903,595 |
| 開発者数 | 1,381,121 |
| 総フォロー数 | 1,885,266 |

**表 2** 一日あたりのコミット数，スター数，プルリクエスト数の統計量

| | Commit | Star | PullReq |
|---|---|---|---|
| 平均 | 0.62 | 0.05 | 0.02 |
| 標準偏差 | 3.47 | 0.76 | 0.66 |
| 中央値 | 0.22 | 0.00 | 0.00 |
| 最大値 | 1201.67 | 184.84 | 298.03 |
| 最小値 | 0.04 | 0.00 | 0.00 |

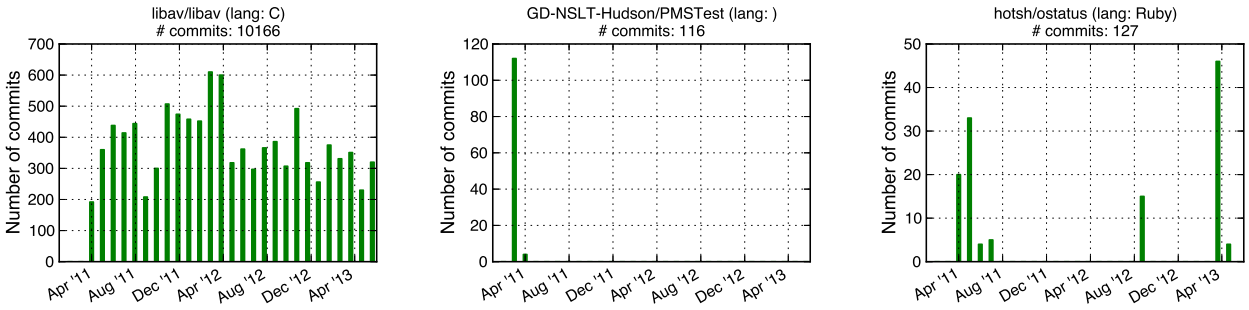
### 2.3 成功を表す指標

プロジェクトの成功を表す指標として以下の 3 種類を使用する．これらは基本的にプロジェクトの経過時間と共に増えるため，プロジェクトの経過日数で割って正規化する．

**コミット数** `Commit`．コミット数はコードやドキュメントなどの追加や変更の回数を表す．従って，プロジェクトの**生産性**を示す指標である．

**スター数** `Star`．スター数はプロジェクトにスターを付けた開発者の数を示す．スターは基本的にブックマークの役割を果たしているため，スター数の多さはプロジェクトの**人気度**を示す．

**プルリクエスト数** `PullReq`．プルリクエスト数はプロジェクト内部のメンバー以外からのコードの変更要求の回数を表す．従って，プロジェクトの**ソーシャル度**を表している．




(a) 定期更新型　(b) 短期集中型　(c) 複数回集中型

**図 2**　プロジェクトの成長パターンの具体例

## 3. 基本分析

まず，ソーシャルコーディングの世界の概観をつかむため，以下の問いに関する分析を行う．

(1) プロジェクトはどれくらいの頻度で更新されるのか？
(2) プロジェクトはどれくらいソーシャルから見られているのか？
(3) プロジェクトはどのように成長するのか？
(4) 開発者間はどのように繋がっているのか？

### 3.1 プロジェクトの統計量

問 (1),(2) に答えるために，プロジェクトの `Commit`, `Star`, `PullReq` の統計量の分布を調査する．表 2 はそれぞれの指標における統計量を示す．一日あたりのコミット数，スター数，プルリクエスト数の平均値はそれぞれ，0.62，0.05，0.02 であった．しかし，標準偏差や最大値，最小値を見て分かるように，これらはプロジェクトによるばらつきが大きい．また，スター数とプルリクエスト数の中央値が 0 となっている．実際，約 53%のプロジェクトはスター数が 0，約 70%はプルリクエスト数が 0 であり，オーナー以外の開発者によって評価されたり変更されるプロジェクトは少ないことが分かる．

### 3.2 プロジェクトの成長パターン

次に，問 (3) に答えるために，プロジェクトがどのように成長していくのかを分析する．分析のために，200 のプロジェクトをランダムに抽出し，3 種類の成長パターンに手作業で分類した．図 2 はそれぞれの成長パターンの具体例を示す。以下では，それぞれの成長パターンの特徴とパターン出現の割合を示す．

- **定期更新型（10%）：** 常にコミットが行われているプロジェクト．
- **一回集中型（45%）：** プロジェクト開始から 1，2 ヶ月でプロジェクトの成長が止まるプロジェクト．
- **複数回集中型（45%）：** 何度か集中してコミットが行われるプロジェクト．

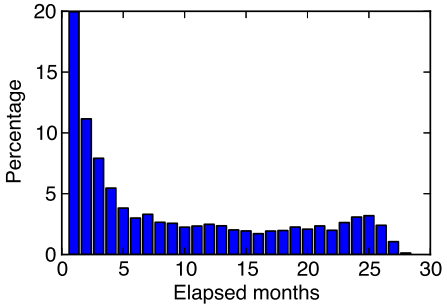



**図 3**　コミットの 90%に到達するまでのプロジェクトの経過月の分布

なお，定期更新型はメンバー数の多いプロジェクトでよく見られ，一回集中型や複数回集中型はメンバーが 1 人のプロジェクトで多く見られる傾向がある．

次に，プロジェクトはどれくらい続くのかを分析する．図 3 は，総コミットの 90%に到達するまでのプロジェクトの経過月の分布を示す．プロジェクトの約 20%は開始から 1 ヶ月以内，約 40%は 3 ヶ月以内でコミットの 90%を終えていることが分かる．一方で，約 20%は 20 ヶ月以上掛かって総コミットの 90%に到達しており，長期間に渡って更新されるプロジェクトも一定数存在することが分かる．

### 3.3 フォローネットワーク

最後に，フォローネットワークを分析することによって開発者の繋がり方の特徴を分析する．

**次数分布．** 図 4 はフォローネットワークの次数分布を示す．横軸は次数，縦軸は正規化した頻度 (PDF) を表す．また，赤の四角点は入次数（フォロワー数）を表し，青の丸点は出次数（フォロー数）を表す．ソーシャルネットワークの次数分布はべき乗則 $f(x) \propto x^{-\alpha}$ に従うとされており，多くのネットワークで $\alpha$ の値が計算されている．フォローネットワークでは，入次数分布は $\alpha = 1.301$，出次数分布は $\alpha = 1.586$ である．先行研究では，Twitter のフォローネットワークは $\alpha = 2.276$ [12]，Mixi の友人ネットワークは $\alpha = 2.4$ 程度 [18] と報告されており，GitHub の

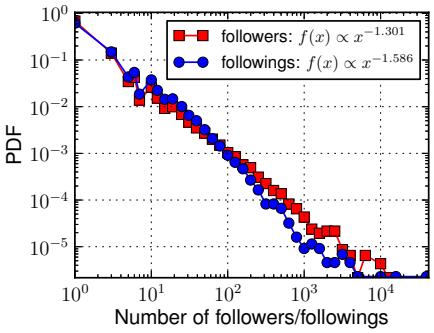



**図 4**　フォローネットワークの次数分布

フォローネットワークの $\alpha$ は比較的小さいことが分かる．これは，芸能人のような誰もが知っているユーザがネットワークに存在しないことが一因であると考えられる．また，$x = 10^2$ 付近から入次数の PDF が出次数の PDF よりも大きくなっている．出次数はユーザがフォローした人数であり，多くのユーザをフォローするためには大きな労力を必要とする．そのため $x = 10^2$ 以降では，出次数の PDF がべき乗則から下に外れている．一方，入次数は所謂 *The rich get richer* 効果により，多数のフォローを獲得するユーザが自然と発生する．このような現象は Twitter のフォローネットワークでも見られる [12].

**相互リンク率．**相互リンク率はネットワーク上の双方向リンクの割合を表す．GitHub のフォローネットワークにおける相互リンク率は 19.1%であり，これは Twitter の 22.1% [12] よりも小さい値である．これはフォローをコミュニケーションというよりは，情報収集のために行っているためだと考えられる．

## 4. メンバー構成と成功の関係

この節では，プロジェクトのメンバー構成や人数がプロジェクトの成功にどのように繋がるかを分析する．ここで，プロジェクトのメンバーはそのプロジェクトの変更権限を持つ開発者とする．これはプロジェクトのオーナーが調整できる要素のみを研究対象とするためである．

開発者間の仕事上の繋がりを表現するために，各プロジェクトの協働ネットワークを以下の手順で構築する．

(1) 各プロジェクト $p \in P$ に対して，プロジェクト内のメンバー間で無向完全ネットワーク $K_p$ を構築する．
(2) メンバー $i$ と $j$ が同じプロジェクトに出現した回数 $c_{i,j}$ を計算する．
(3) 全てのプロジェクトのネットワークを結合して，全体の協働ネットワーク $G = \bigcup_{p \in P} K_p$ を作成する．
(4) 各プロジェクト $p \in P$ に対して，$c_{i,j}$ が 2 以上のリンクから成るプロジェクトの協働ネットワーク $G_p$ を抽出する．

上記の手順で構築したプロジェクトの協働ネットワークは，そのプロジェクト以外での協働関係を反映したものとなっている．この協働ネットワークを使って，プロジェクト内のメンバー構成の特徴量を抽出する．表 3 は抽出した特徴量の一覧を示す．

**表 3**　メンバー構成を表す特徴量

| 記号 | 説明 |
|---|---|
| Member | 変更の権限を持つメンバー数 |
| CoEdgeDense | 協働ネットワークにおけるリンク数をノードペアの組み合わせの数で割った値 |
| CoCluster | 協働ネットワークのクラスター係数 [16] |
| CoFracComp | 協働ネットワークのノード数に対する連結成分の数の比 |
| CoPathLen | 協働ネットワークの平均最短パス長 |

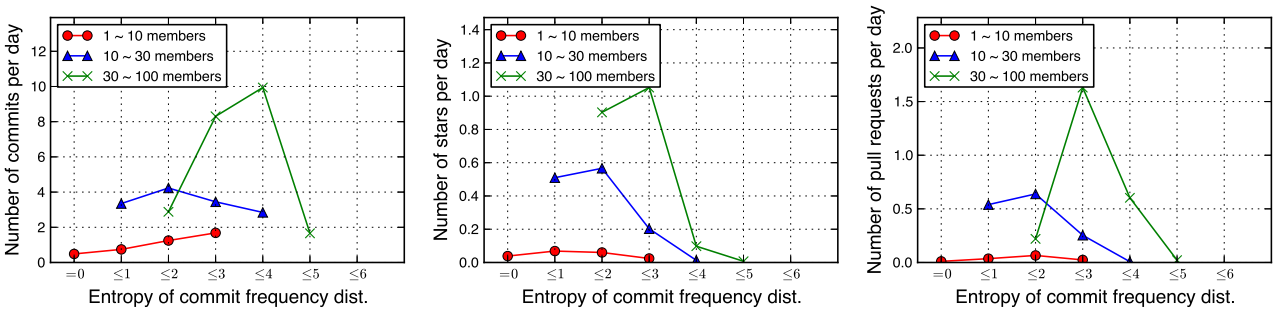
**表 4**　メンバー構成と成功の間のケンドールの順位相関係数

| | Commit | Star | PullReq |
|---|---|---|---|
| Member | 0.16 | 0.11 | 0.18 |
| CoEdgeDense | 0.09 | 0.10 | 0.16 |
| CoCluster | -0.04 | 0.08 | 0.13 |
| CoFracComp | -0.09 | -0.11 | -0.17 |
| CoPathLen | 0.11 | 0.15 | 0.16 |

### 4.1 相関分析

表 4 はメンバー構成の特徴と成功の指標の間のケンドールの順位相関係数を示す．

まず，内部メンバーの数 Member の多さと成功の各指標はある程度相関があることが見て取れる．メンバー数が多いことによってコミット数が増えることは自然ではあるが，外部からの評価であるスターやプルリクエストの獲得にも繋がることが示唆された．次に，CoEdgeDense との相関を見る．CoEdgeDense はプロジェクト内の繋がりの強さを表す指標である．表より，CoEdgeDense は全ての成功の指標において正の相関を持ち，比較的強い相関があったのは PullReq であることが分かる．この結果は，CoFracComp が成功と負の相関を持つことからも説明できる．CoFracComp は協働ネットワークがどれだけ分断されているかを示す．従って，協働ネットワークが分断されていない方が成功するという結果になっており，繋がりの強さが成功にはある程度重要であることが示唆される．その一方で，CoPathLen と成功の各指標は正の相関を持つことは興味深い．CoPathLen は全ての開発者ペア間のネットワークにおける平均距離を示すため，CoEdgeDense が大きいプロジェクトでは CoPathLen は小さくなる傾向がある（CoEdgeDense と CoPathLen の相関係数は-0.34）．CoPathLen が大きいネットワークの特徴の一つは，多くのメンバーと繋がるハブがいないことである．以上の知見をまとめると，プロジェクト内でハブとなるような開発者はできるだけ作らず，ネットワークは分断させず開発者間の繋がりを保つことが重要であると考えられる．




(a) コミット数　(b) スター数　(c) プルリクエスト数

**図 5**　メンバーの貢献分散度と成功の関係

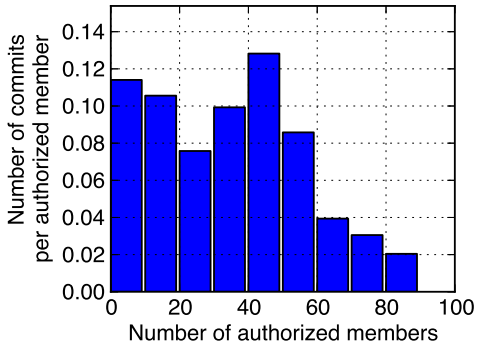



**図 6**　メンバー数とコミット数の関係

### 4.2　効率的なメンバー数

チームによる共同作業における最適なメンバー数の研究が多くなされている [6], [9], [14]．ここでは，プロジェクトの生産性を表すコミット数に焦点を当てて，ソーシャルコーディングのプロジェクトにおける効率的なメンバー数の分析を行う．

図 6 はメンバー数とコミット数の関係を表す．横軸はプロジェクト内のメンバー数で，縦軸はメンバー 1 人あたりのコミット数の中央値である．また，1 つのビンの幅は 10 人である．まず，全体的な傾向として，メンバー数が増えることによって 1 人あたりのコミット数が減少することが分かる．この結果の原因として，1) 仕事が分散してメンバー 1 人 1 人の仕事量が減った，2) 仕事をしないメンバーが現れたことが考えられる．いずれの理由であっても，メンバーの力を効率的に生かせていないことになる．特にメンバー数が 60 人以降で，顕著な減少傾向が見られる．すなわち，プロジェクトの生産性や活発度を上げるために 60 人以上のメンバーを登用することは効率的ではないと考えられる．

### 4.3　メンバーの貢献の偏り

次に，プロジェクト内でのメンバーのコミット数の偏りが成功に対して与える影響を分析する．コミット数の偏りに関して，2 つの極端なケースが想定できる．

(1) メンバーの一部がほとんどのコミットを行う
(2) 全てのメンバーで平等に分担してコミットを行う

では，どちらのケースに当てはまるプロジェクトが成功するのだろうか？各プロジェクトのコミットの偏りを定量化するために，以下のエントロピーを計算する．

$$\texttt{Entropy} = -\sum_{m \in M} \theta_m \log \theta_m \qquad (1)$$

ここで，$M$ はプロジェクトのメンバー集合，$\theta_m$ はそのプロジェクトにおけるメンバー $m$ のコミットの割合を表す．Entropy は一部のメンバーにコミットが集中するとき 0 に近い値となり，コミットが分散するとき大きな値となる．

図 5 はメンバーの貢献分散度 Entropy と成功の各指標の関係を示す．図 5(a) はコミット数との関係を示す．メンバーが 1–10 人と小さいプロジェクトでは，コミットが分散していた方が全体としてのコミット数を増加させることが見て取れる．これは，メンバーが 1 人から 10 人に増加しても一人あたりのコミット数はあまり減少しないため，その規模のプロジェクトではメンバー全員で協力することによってプロジェクトの生産性を上げることができると考えられる．メンバーが 10–30 人の中規模プロジェクトを見ると，メンバーが 10 人以下のプロジェクトよりもコミット数が多くなるものの，Entropy が大きくなってもコミット数が増えないことが分かる．さらに，メンバーが 30–100 人の大規模プロジェクトでは，一部のメンバーにコミットが集中する場合にはその一部のメンバーのコミットの頻度も低く，反対にコミットが分散する場合でも全員のコミット数が少なくなり，中規模プロジェクトのコミット数よりも少なくなる結果が得られた．

興味深いことに，メンバーの貢献分散度は外部評価であるスター数やプルリクエスト数とも関係がある．図 5(b) は獲得したスター数との関係を示す．これを見ると，プロジェクトの規模によらず，Entropy が大きくなると獲得スター数が減ることが分かる．図 5(c) のプルリクエスト数では，スター数で見られた傾向とともに，メンバー数 30 以上で一部のメンバーにコミットが集中したプロジェクトはプルリクエストが得られていないことが分かる．

## 5. プロジェクト外部の開発者への対応と成功の関係

4 節では，プロジェクト内部のメンバーをどのように構成すれば良いかについて言及した．この節では，プロジェクトメンバーがどのような活動をし，プロジェクト外（ソーシャル）からのプルリクエストにどのように対応すると成功するのかについて分析する．表 5 は，本研究で扱うプロジェクト外部の開発者への対応を表す特徴量の一覧を示す．ここで，プルリクエストへの反応とは，プルリクエストの是非に関して議論することやプルリクエストを取り込む（マージする）ことを意味する．

表 6 はプロジェクト外部の開発者への対応と成功の間のケンドールの順位相関係数を示す．まず，プルリクエストへの対応の良さを表す ResponseRate とスター数やプルリクエスト数が相関があることが分かる．すなわち，プロジェクト外の開発者にプルリクエストをしてもらうためには，プルリクエストされたときには無視せず対応することが重要であり，そのことによってプロジェクトの人気度も上昇することが示唆される．では，プルリクエストへの反応の早さは重要なのだろうか．表 6 における ResponseTime の行を見ると，対応の遅さとスター数やプルリクエスト数の間には比較的小さいが正の相関があるとの結果が得られている．プルリクエストに対して遅く対応することが望ましいとは言えないが，意識的に素早く対応する必要はなさそうである．

プルリクエストはプロジェクトメンバーでも行うことができる．このようなことを行うメリットは，コードの変更がオープンな議論の下で行うことができる点である．ソーシャルコーディングにおいては，プロジェクトメンバー以外が開発に参加するケースがあるため，コード変更の透明性は重要だと考えられる．このような開発の仕方は，日本ではチケット駆動開発と呼ばれている．PullReqByInside と成功の関係を見ると，Star や PullReq と正の相関があることが分かる．なお，PullReq はソーシャルからのプルリクエスト数のみを計算している．Commit は，Star や PullReq と相関がないことから，内部でもプルリクエストを利用して開発を行うことによりソーシャルからの人気度を高めたり，多くのバグ報告やコードの修正を受けることができる．

## 6. プログラミング言語と成功の関係

最後に，プロジェクトで使用するプログラミング言語と成功の関係に関する分析を行う．図 7 は 15 種類のプログラミング言語それぞれにおける平均コミット数，平均スター数，平均プルリクエスト数を示す．なお，2012 年 2 月におけるプロジェクト数上位 15 言語を分析対象とした[*6]．

[*6] ”Using Google BigQuery to learn from GitHub data”: http://bit.ly/JFVOht

**表 5** プロジェクト外部の開発者への対応を表す特徴量

| 特徴量 | 説明 |
|---|---|
| ResponseRate | プルリクエストに対してメンバーが 1 ヶ月以内に反応した割合 |
| MergeRate | プルリクエストをプロジェクトに取り入れた割合 |
| ResponseTime | プルリクエストに対してメンバーが最初に反応した経過時間の平均 |
| PullReqByInside | メンバーによるプルリクエストの数 |

**表 6** プロジェクト外部の開発者への対応と成功の間のケンドールの順位相関係数

| | Commit | Star | PullReq |
|---|---|---|---|
| ResponseRate | 0.01 | 0.39 | 0.83 |
| MergeRate | 0.09 | 0.37 | 0.68 |
| ResponseTime | -0.04 | 0.20 | 0.14 |
| PullReqByInside | 0.14 | 0.27 | 0.31 |
| Commit | 1.00 | 0.06 | 0.03 |

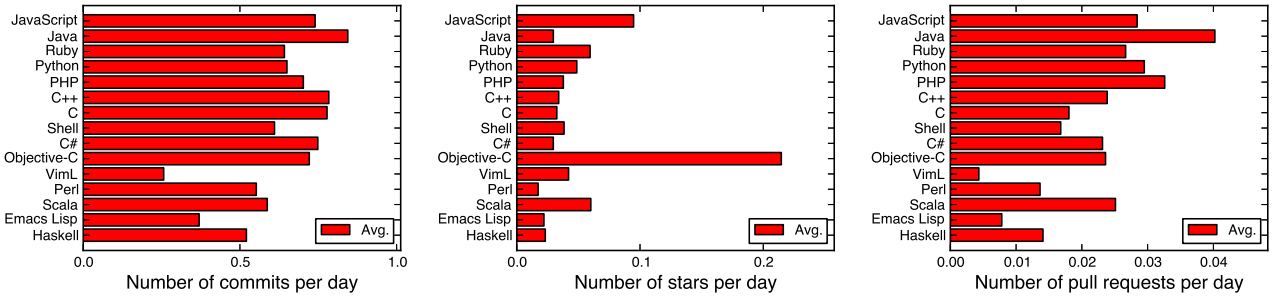
図 7(a) は平均コミット数との関係を示す．Java を使用するプロジェクトは平均コミット数が最も大きい．2 番目が C++，3 番目が C などといった傾向を見ると，デスクトップアプリケーションの開発が比較的活発に行われる傾向が覗える．図 7(b) は平均スター数との関係を示す．平均コミット数の場合とは異なり，Objective-C を使用するプロジェクトが多くのスターを獲得する結果となった．Objective-C は，主にアップルの Mac OSX や iOS 上で動作するアプリケーションの開発で利用される．Net Applications 社によると， 2013 年 8 月時点の Mac OSX のマーケットシェアは 7.26%[*7]であり，Objective-C で書かれたソフトウェアを利用できるユーザは比較的少数である．それでも Objective-C が人気を引き寄せるのは，GitHub のコミュニティが開発者のみから成るためだと考えられる．図 7(c) は平均プルリクエスト数との関係を示す．この図を見ると，最もプルリクエストを集めるのは，コミット数と同様に Java を採用するプロジェクトであった．従って，Java を採用することによって外部からも活発に開発を進められる可能性がある．

## 7. 関連研究

ソフトウェア開発プロジェクトの成功に関する研究は，ソフトウェア工学における重要な問題である．多くの研究はアンケートによる分析やケーススタディを行っている [3], [5], [17]．ここでは，本論文と直接関係するプロジェクトのデータ解析の研究のみを紹介する．

これまでに，ソーシャルコーディングサイトのデータを使用して分析を行った研究はほとんど存在しない．また，本論文はソーシャルコーディングにおける成功の法則を分

[*7] http://www.netmarketshare.com/




(a) コミット数　(b) スター数　(c) プルリクエスト数

**図 7**　プログラミング言語と成功の関係

析した最初の論文である．Thung ら [15] は，GitHub 上の約 10 万プロジェクトから協働関係から開発者ネットワークとプロジェクトネットワークを作成し，これらのネットワーク的性質の調査と PageRank を使ったプロジェクトと開発者のランキングを行った．彼らの構築した協働関係ネットワークの次数分布はベキ則に従わないため，図 4 で示したフォローネットワークと協働関係ネットワークの構造は異なることが分かる．

少し対象範囲を広げ，オープンソースリポジトリの研究についても紹介する．オープンソースリポジトリを対象とした多くの研究が SourceForge.net のデータを使用しており，開発者のネットワークの分析 [1], [10], [13]，協働ネットワークの成長モデルの構築 [7]，コードサイズの成長モデルの構築 [11]，開発者の地理的関係の分析 [4] などが行われている．しかし，我々の知る限り，プロジェクトの成長に繋がる因子を分析した研究は存在しない．

## 8. おわりに

ソーシャルコーディングにおけるプロジェクト成功の法則を見つけ出すために，GitHub のデータ分析を行った．特に，1) プロジェクトメンバーの構成，2) プロジェクト外部の開発者への対応，3) プログラミング言語の選択に着目し，それぞれと成功の関係を調べた．なお，本論文ではプロジェクトの生産性を一日あたりのコミット数，プロジェクトの人気度を一日あたりのスター数，ソーシャル度を一日あたりのプルリクエスト数とし，これらを成功の指標として用いた．

本研究を通じて得られた知見を以下にまとめる．

- プロジェクトメンバーの多いプロジェクトは生産性，人気度，ソーシャル度も大きくなる傾向があるが，メンバー数が 60 人を超えると，一人一人の生産性は下がっていく．
- メンバー間の繋がりの強いプロジェクトは，ソーシャル度が高くなる傾向がある．
- メンバー数が 10 人以下ならば，仕事が分散しているときに生産性が高い．メンバー数が 30 人以上ならば，適度に分散しているときに生産性が高い．
- プルリクエストへの対応の良いプロジェクトは人気度とソーシャル度が高い．
- プロジェクト外部にも開発のやり取りが可視化されることにより，人気度やソーシャル度が高くなる．
- Objective-C を採用するプロジェクトが最も人気度が高い．

今後の研究では，本論文での知見とソーシャルコーディングの特性を生かし，ソーシャルコーディングでの成功するプロジェクトを増やす方法を検討していく予定である．

**謝辞**　本研究は JSPS 特別研究員奨励費の助成を受けたものです．

### 参考文献

[1] Christley, S. and Madey, G.: A Topological Analysis of the Open Souce Software Development Community, *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, IEEE, pp. 198a–198a (2005).

[2] Crowston, K., Howison, J. and Annabi, H.: Information Systems Success in Free and Open Source Software Development: Theory and Measures, *Software Process: Improvement and Practice*, Vol. 11, No. 2, pp. 123–148 (2006).

[3] Dabbish, L., Stuart, C., Tsay, J. and Herbsleb, J.: Social Coding in GitHub: Transparency and Collaboration in An Open Software Repository, *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, New York, New York, USA, ACM Press, p. 1277 (2012).

[4] Dawid, W.: Quantitative Analysis of Open Source Projects on Sourceforge (2005).

[5] Dinh-trong, T. T., Bieman, J. M. and Member, S.: The FreeBSD Project : A Replication Case Study of Open Source Development, Vol. 31, No. 6, pp. 481–494 (2005).

[6] Fan, Z.-P., Feng, B., Jiang, Z.-Z. and Fu, N.: A Method for Member Selection of R&D Teams Using the Individual and Collaborative Information, *Expert Systems with Applications*, Vol. 36, No. 4, pp. 8313–8323 (2009).

[7] Gao, Y. and Madey, G.: Towards Understanding: A Study of The Sourceforge.net Community Using Modeling and Simulation, *Proceedings of the 2007 spring simulation multiconference-Volume 2* (2007).

[8] Guimerà, R., Uzzi, B., Spiro, J. and Amaral, L. a. N.:

Team Assembly Mechanisms Determine Collaboration Network Structure and Team Performance., *Science*, Vol. 308, No. 5722, pp. 697–702 (2005).

[9] Hoegl, M.: Smaller teams–better teamwork: How to keep project teams small, *Business Horizons* (2005).

[10] Hu, D. and Zhao, J.: A Comparison of Evaluation Networks and Collaboration Networks in Open Source Software Communities., *Americas conf. on Information Systems (AMCIS2008)* (2008).

[11] Koch, S.: Evolution of Open Source Software Systems a Large-scale Investigation, *Proceedings of the 1st International Conference on Open Source Systems* (2005).

[12] Kwak, H., Lee, C., Park, H. and Moon, S.: What is Twitter, a social network or a news media?, *Proceedings of The 19th International Conference on World Wide Web*, pp. 591–600 (2010).

[13] Madey, G., Freeh, V. and Tynan, R.: The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory, *Americas conf. on Information Systems (AMCIS2002)*, pp. 1806–1813 (2002).

[14] Mueller, J. S.: Why Individuals in Larger Teams Perform Worse, *Organizational Behavior and Human Decision Processes*, Vol. 117, No. 1, pp. 111–124 (2012).

[15] Thung, F., Bissyande, T. F., Lo, D. and Jiang, L.: Network Structure of Social Coding in GitHub, *17th European Conference on Software Maintenance and Reengineering*, IEEE, pp. 323–326 (2013).

[16] Watts, D. J. and Strogatz, S. H.: Collective dynamics of 'small-world' networks., *Nature*, Vol. 393, No. 6684, pp. 440–2 (online), DOI: 10.1038/30918 (1998).

[17] Yokozawa, M., Shinohara, T. and Ishida, T.: Collaboration with Lean Media : How Open-Source Software Succeeds, *Proceedings of the 2000 ACM conference on Computer supported cooperative work* (2000).

[18] 松尾豊, 安田雪：SNS における関係形成原理：mixi のデータ分析, 人工知能学会論文誌, Vol. 22, pp. 531–541 (2007).