2413

自発的ソフトウェア進化を促すプロジェクト状態の推定

Investigating Project States Boosting Spontaneous Software Evolution

○中村 匡秀[*1*5]，戸田 航史[*2]，玉田 春昭[*3]，松本 健一[*4]
Masahide NAKAMURA[*1]，Koji TODA[*2], Haruaki TAMADA[*3], Kenichi MATSUMOTO[*4]

[*1] 神戸大学　Kobe University
[*2] 福岡工業大学　Fukuoka Institute of Technology
[*3] 京都産業大学　Kyoto Sangyo University
[*4] 奈良先端科学技術大学院大学　Nara Institute of Science and Technology
[*5] 理化学研究所・革新知能統合研究センター　RIKEN AIP

Social coding platforms (SCPs) have realized spontaneous software evolution, where new source code and ideas are spontaneously proposed by altruistic developers. Although there are many projects operated by active communities performing spontaneous evolution, it is yet unclear that how such successful projects and communities have been formed and governed. In this paper, we propose a method that can investigate the history of every project in the SCP. Introducing the concept of project as a city, we consider every project in the SCP as a city, where a government and citizens develop a city through collaborative activities. We then identify essential attributes that characterize a state of a city. For each attribute, we develop metrics that quantify the state S(p, t) of a project p at time t. An experimental evaluation investigating GitHub projects of famous code editors shows that the proposed metrics well visualize the history of the projects from essential perspectives of a city.

Key Words : social coding, software evolution, project management, software metrics, governance, smart city

1. 緒　　言

近年ソーシャルコーディング(1)という新しいソフトウェア開発方式が注目を集めている．ソーシャルコーディングは，多数の開発者が協調し，集合知によってソフトウェアを開発する手法である．その代表的なプラットフォーム GitHub(2)や Bitbucket(3)では，フォーク・ブランチ機能により，開発者は，他の開発者の作業に影響を与えることなく，機能追加や不具合修正のためのソースコード変更を独自のアイデアや判断で行うことができる．更に，プルリクエスト機能により，その変更内容を当該ソースコードに反映することを，ソースコードの管理者（プロダクトオーナー）に提案することができる．プルリクエストを受け取ったプロダクトオーナーは変更内容をレビューし，ソースコードに反映するかどうかその採否を判断する．

ソーシャルコーディングは，オープンソースソフトウェア開発ですでに広く普及しており，一部の先進的な企業でも採用され始めている．従来のソフトウェア開発との大きな違いは，ソフトウェアの変更，すなわち，ソフトウェア進化(4)のアイデアが，上意下達ではなく開発者自身から自発的に提案されることである．このような新たなソフトウェア進化の形態を，我々は「自発的ソフトウェア進化」と呼ぶ(5)．

自発的ソフトウェア進化は，開発者自身のアイデアや解決法に基づいてソフトウェアが変更されるため，プロダクトオーナーが想像もしなかった斬新な機能が提案されたり，把握しきれていなかった不具合の修正が行われたりする．そのため，従来の上意下達のやり方に比べて，より創造的で価値の高いプロダクトが生まれる可能性が高い．例えば， Microsoft の VSCode(6)は，GitHub 上で開発が進められている著名なコードエディタである．プロジェクトには多くのボランティアがコードを提供し，革新的な機能が日々追加さえている．まさに，自発的ソフトウェア進化がうまく回っているプロジェクトである．

しかしながら，こうした成功プロジェクトがどのように管理され，あるいは，活発なコミュニティがどの

ように形成されているのかは自明ではない．これらを明らかにするには，プロジェクトが開始されてから現在まで，どのような軌跡をたどったかの歴史を知る必要がある．幸運なことに，GitHub には様々なイベントログが記録・蓄積されているので，プロジェクトにおいて過去どのようなことが起こったかを振り返って見ることができる．そこで本研究では，GitHub の成功プロジェクトが，これまでにどのような**状態**をたどってきたかを分析することで，自発的ソフトウェア進化に寄与する要因を解明することを試みる．より具体的には，プロジェクト p と任意の時点 t について，S(p,t)を p の t における状態とするとき，t を変化させながら S(p,t)を観測することで，p の進化を観測しようというアプローチである．

本研究の目的は，以下の 2 つのリサーチクエスチョンに答えることである：

RQ1: 状態 S(p,t)を何でもってどのように定義するか？

RQ2: 成功プロジェクトの状態はどのように推移しているか？

RQ1 に答えるため，我々は「都市としてのプロジェクト」 (Project as a City)の考え方を提案する．ソーシャルコーディングのプロジェクトを都市とみなし，自治体と住民が共同して都市を発展させるスマートシティ[7]のアナロジーを取り入れる．初めに，都市の構成要素や取り組みを，プロジェクト内のエンティティやアクションに写像するマッピングを考える．次に，都市の状態を性質づける基本的な属性を洗い出す：人口，規模，都市の課題，課題解決能力，自治体の能力．これら都市の属性に対応するプロジェクトの属性に対して，その度合いを評価する 8 種類の尺度（メトリクス）を定義する．各メトリクスの値により，プロジェクト p の時点 t における状態 S(p,t)を性質づける．

RQ2 に答えるため， GitHub における成功プロジェクトの進化を分析する実験を行う．具体的には，3 種類の著名なコードエディタ Atom[8]，Brackets[9]，VSCode[6]について，提案メトリクスを活用し，状態の推移を字形れるで観測する．実験の結果，提案メトリクスによって，3 つのプロジェクトの進化を都市に見立てた進化の観点から観測できることを確認した．

2. 準　　備

2・1 ソーシャルコーディング基盤 GitHub

GitHub は，クラウドベースのソフトウェア開発プラットフォームである．コードや文書バージョン管理には Git を採用するとともに，ソーシャルコーディングのための様々な機能を提供している．GitHub は現在，2,800 万人のユーザを抱える世界最大規模のソーシャルコーディング基盤である．

GitHub において，プロジェクトの文書やコードは，レポジトリとよばれる貯蔵庫に管理される．レポジトリはプロジェクトの 1 人または複数人のコアメンバーによって管理される．公開されたレポジトリは，コアメンバー以外でも参照可能であるが，コアメンバー以外の改変・更新は許されない．フォーク(fork)とは，公開レポジトリを自分のプロジェクトに複製することである．これにより，コアメンバー以外の開発者が，オリジナルに影響を与えることなく，独自のアイデアで自由にコードを改造できるようになる．開発者が自ら作成したコードの改造をシェアしたい場合，オリジナルのレポジトリに対して，プルリクエスト(pull request)を発行する．もし，コアメンバーがこの改造を有益だと判断した場合，プルリクエストを受理して改造をマージ(merge)することができる．もし，コアメンバーがシェアに同意しない場合，プルリクエストは棄却(reject)される．

もし，GitHub のユーザがある特定のプロジェクトに興味を持った場合，星(star)をつけることができる．星の数はプロジェクトの人気を表す指標となる．もし，ユーザがバグを見つけたり，機能の追加や改訂を要望したりする場合には，リポジトリに対してイシュー(issue)を提起する．イシューはプロジェクトにおける課題として扱われ，ほかのユーザが課題解決のコメント・ヒントを送ったり，あるいは開発者がコードを改造してプルリクエストを送ったりする．課題が解決すると，そのイシューはクローズされる．

2・2 自発的ソフトウェア進化

ソフトウェア進化とは，一旦出荷されたソフトウェアに対する変更を受け入れる仕組みや活動を指す[1]．これまで多くのソフトウェア開発では，プロジェクト・マネージャ(PM)を責任者とする階層型組織に基づく開発手

法が採られている．すなわち，PM が成果を定義し，仕事を設計し，プロジェクトを管理する役割を果たし，開発者は PM の指示に従って開発するという，いわば上意下達によって，ソフトウェアの開発が進められている．

GitHub をはじめとするソーシャルコーディング基盤の登場によって，新たなプロジェクトのあり方が可能になった．つまり，開発者がプルリクエストによって新たなコードを提案し，PM は提案をレビューしてマージするか否かの選択をする．ソフトウェア進化が PM から開発者への上意下達ではなく，開発者から PM の提案によって駆動する新たな進化体系が可能になったのである．我々はこの新たなソフトウェア進化を「自発的ソフトウェア進化」と呼び，科学研究費・基盤研究(A)「自発的ソフトウェア進化の加速に向けた基礎技術の開発」に取り組んでいる[5]．主な課題として，自発的進化をどのように統制（ガバナンス）するのか，開発者の協調によって生まれる新たな作業（ソーシャル・オーバーヘッド）をいかに低減するのか，コミュニティの自発性をどのように維持していくのか（持続可能性）というテーマが存在する．本研究は，特にガバナンスの課題に取り組むものである．

3. Project as a City: 都市としてのプロジェクト

3・1 キーアイデア

リサーチクエスチョン RQ1 に答えるため，Project as a City の考え方を導入する．2.1 で見たように，ソーシャルコーディング基盤の任意のプロジェクトは，コアメンバーによって統制されている．そのプロジェクトに興味を持つ任意の開発者やユーザはコミュニティ，すなわちある種の社会を形成する．コミュニティの誰もがコードやイベントログを見ることができ，問題を提起したり，要望を提案したり，コードの改造を提案したりできる．そこには静的な上下関係は存在しない．

こうした観測は，スマートシティ[6]にも見ることができる．スマートシティでは，都市の様々なデータが集められ市民に開示される．これらのデータを活用して自治体と市民が協働し，都市機能の効率化や市民の暮らし向きを向上させる取り組みを行う．まさに，ソーシャルコーディングの取り組みに類似する．

ある研究[6]では，スマートシティの状態をベクトル［s1:v1, s2:v2, ..., sm:vm］で表現している．ここで，si は都市のある属性，vi はその値を表す．都市を状態機械(State Machine)と捉えて，スマートシティにおける活動を，現在の都市状態から理想（受理）状態へ遷移させる状態遷移と定義する．本研究では，上記のアナロジーを採用し，ソーシャルコーディング基盤の構成要素を都市の構成要素に対応付けることで，プロジェクトの状態 S(p, t)を定義することを狙う．

3・2 要素の対応付け

ソーシャルコーディング基盤の基本的な要素を都市の構成要素に対応付けるマッピングを定義する．理解しやすいように，GitHub を代表的なソーシャル基盤として取り入れる．2.1 の用語も併せて参照されたい．

- 1 つのソフトウェアプロジェクトは，1 つの都市に関連付ける．
- コードは都市の機能に関連付ける．新しいコードをマージすることは，都市に新しい機能を追加することに対応する．
- プロジェクトに興味を持つ開発者およびユーザは，市民に関連付ける．便宜上，プロジェクトに星を付けたユーザを市民とみなすことにする．
- コアメンバーは，都市の政府に関連付ける．原則，政府が都市の変更権限を持っているためである．
- イシューは都市の課題あるいは都市への要望に関連付ける．イシューの作成・クローズは，都市の課題提起，課題解決に相当する．
- プルリクエストは，課題に対する解決の提案に関連付ける．あるイシューに対するプルリクエストの作成は，市民から政府に対して，ある都市課題を解決する提案を行うことに相当する．

3・3 進化を特徴づける本質的な属性

プロジェクトを都市に対応付けることで，状態を定義するための属性を考えやすくなる．都市の歴史や進化を考える時，以下の属性は非常に重要である．もちろんこれらは都市の全ての面をカバーするものではない．

A) 人口： その都市が市民にとってどれだけ魅力的かを表す指標である.
B) 規模： その都市がより便利により高機能にどれだけ開発されているかを表す指標である.
C) 課題： その都市において市民が暮らしにどれだけ問題を抱えているかを表す指標である.
D) 課題解決能力： 都市の課題がどこまで解決できているかを表す指標である.
E) 政府の対応能力： 政府がどれだけ積極的に市民の提案を聞き入れ課題解決しているかを表す指標である.

プロジェクトpと時点tが与えられたとき，上記のi番目の属性をpのt時点で計測するメトリクスmi(p,t)を考える．この時，プロジェクトの状態を以下のように定義する.

$$S(p,t) = [m1(p,t),\ m2(p,t),\ \ldots,\ mn(p,t)]$$

各メトリクスmiの具体的な定義は次節で述べる.

4. 都市としてのプロジェクトを性質づけるメトリクスの提案

4・1 人口

人口は都市にとって最も基本かつ重要な要素である．人口の増加は，都市における活動，新たな市民からの意見，課題解決のための労力を向上させる．人口は，都市の規模や，課題の数・多様性，課題解決能力，政府の寛大さにも影響する.

ソフトウェアプロジェクトを都市に見立てる時，人口は開発に関与するすべての参加者（コアメンバー，非コアメンバー，興味があるユーザ）とすべきである．しかし，すべての参加者がフィードバックやコード改造を提案するわけではないので，それらの正確な数を把握することは難しい.

そこで我々は，プロジェクトにつけられた星の数によって，プロジェクトの人口を表すことにする．なぜなら，ユーザがプロジェクトに対して顕著な貢献がなくても，最低限興味があれば，星をつけることはできるからである．よって，星の数はプロジェクトの人口を性質づけるメトリクスとして妥当だと考えた.

N-STAR(p,t)をプロジェクトpが時点tで稼いでいる「星の数」と定義する.

4・2 規模

都市がどれだけ発展しているかを表す規模は，人口同様重要な要素である．ソフトウェアプロジェクトを都市に見立てる時，規模はリポジトリに新たに追加・マージされたソースコードの量に関連付けることができる.

ソースコードの量は単純にコード行数(LoC, Lines of Code)で計測可能である．GitHubでは，プルリクエストがマージされるとき，変更があったコードはdiff形式で記録され，それより追加・削除されたコードの行数を数えることができる．すべてのプルリクエストについて，行数を累積することで，そのプロジェクトがどれだけ発展しているかを規模として捉えられる.

ここで，新たなメトリクス「マージコード行数」を定義する．今，pをプロジェクト，tを時点，rをプルリクエストとする．lmod(r)をrにおいて変更された行数とする．また，cpr(p,t)をpにおいてt以前にクローズされたすべてのプルリクエストの集合，mpr(p,t) (⊆cpr(p,t))をpにおいてt以前にマージされたプルリクエストの集合とする．この時，マージコード行数N_MGCL(p,t)は次のように定義される.

$$N\_MGCL(p,t) = \sum_{r\ =mpr(p,t)} lmod(r) \tag{1}$$

4・3 課題

都市における課題はその進化への障害となる.都市の人口や規模が増えるにつれ,様々な課題が出てくる.何の対策もされなければ，単純に課題の数は増加するばかりである．理想的には，課題はその発生からなる

べく早く解決されるべきである.長期間放置されれば,住民は都市および政府に失望し,去っていくだろう.政府は現在どんな課題が残っているかを認識し，しかるべきアクションを取らなければならない.

都市としてのプロジェクトにおいては，課題はイシューに対応付けられる.解決されたイシューはクローズされる.よって，残りのイシューの数を数えることで，都市が現在抱えている課題を性質づけられる.いま，pをプロジェクト，tを時点とするとき，N_AIS(p,t)をpにおいてt以前に存在したすべてのイシューのとする.同様に,N_CIS(p,t)をt以前にクローズされたイシューの数とする.この時，メトリクス「残イシュー数」 N_RIS(p,t),「残イシュー率」 R_RIS(p,t)を次のように定義する：

$$N\_RIS(p,t) = \boldsymbol{N\_AIS(p,t) - N\_CIS(p,t)} \tag{2}$$

$$R\_RIS(p,t) = \frac{\boldsymbol{N\_RIS(p,t)}}{\boldsymbol{N\_AIS(p,t)}} \tag{3}$$

これらの値が大きくなれば，放置されている課題の数が多いことを意味する.

4・4 問題解決能力

問題解決能力は，都市の未来に向けた発展に重要な要素である.もし政府や市民が目の前の課題を即座に解決できなければ，新たな問題がその上に山積されてしまう.よって，都市は課題をなるべく早く解決する能力を有するべきである.

この解決能力を定量化するために，我々はイシューの解決に要する時間を利用する.つまり,「イシューの生存時間」によって解決能力を性質づける.今，イシューxに対して，since(x)，until(x)をそれぞれxが作成された，クローズされた時刻とする.この時，xの生存時間lt(x)は，以下の時間差で定義される：lt(x)=until(x)−since(x).今，CIS(p,t)をpにおいてt以前にクローズされたイシューの集合とする.この時，メトリクス「イシュー生存時間」 LT_CIS(p,t)を次のように定義する：

$$LT\_CIS(p,t) = \frac{\sum_{x \in \boldsymbol{CIS(p,t)}} \boldsymbol{lt(x)}}{\boldsymbol{N\_CIS(p,t)}} \tag{4}$$

LT_CIS(p,t)が大きくなれば,課題が解決されずに長時間放置されていることを意味する.便宜上,LT_CIS(p,t)の単位を日とする.

4・5 政府の対応能力

自発的進化をサポートする政府の重要な役割は，市民から寄せられる提案（すなわち，プルリクエスト）を精査し，その提案を採択するか棄却するかを決定することである.このプロセスを，受容性，迅速性，健全性の観点から評価する.

受容性は，政府が市民からの提案をどの程度採択するか，すなわち，その寛容性や民主制の度合いに関係する.もし市民からの提案がほとんど受理されなければ，彼らは失望し都市を去るかもしれない.

市民の提案はプルリクエストなので，全プルリクエストのうち，何割が受理されたかを計測すれば，受容性を性質づけられる.メトリクス「マージされたプルリクエスト率」R_MGPR(p,t)を次のように定義する：

$$R\_MGPR(p,t) = \frac{|\boldsymbol{mpr(p,t)}|}{|\boldsymbol{cpr(p,t)}|} \tag{5}$$

迅速性は，政府が採択・棄却のプロセスをいかに早く完了するかを表す指標である.進化の速度に影響を与える.迅速性を評価するために,「プルリクエストの生存時間」を利用する.プルリクエストrに対して，since(r)，until(r) をそれぞれrが作成，クローズした時刻とする.この時，rの生存時間は，

$$\mathrm{lt(r)} = until(r) - since(r)$$

で定義される．すべてのプルリクエストの生存時間を平均することで，メトリクス「プルリクエスト生存時間」LT-PR(p,t)を次のように定義する：

$$LT\_PR(p,t) = \frac{\sum_{r \in cpr(p,t)} \boldsymbol{lt(r)}}{|\boldsymbol{cpr(p,t)}|} \tag{6}$$

健全性は，政府がいかに提案の質を検査しているかの度合いを表す．GitHub をはじめとするソーシャルコーディング基盤では「任意のプルリクエストは最低でも 1-2 人の第三者にレビューされるべきである」というルールを推奨している．第三者によるレビューは一般に時間がかかるので，このルールはしばしば無視されることがある．こうしたケースが続けば，提案の品質が精査されなくなり，プロダクトの品質が下がる．健全性を定量化するために，2 名以上のユーザが議論化していないプルリクエストの割合を計測する．プルリクエスト r に対して，nop(r)を r に関わったユーザの数とする．この時，レビュールールを違反しているプルリクエストの集合を，

$$\mathrm{mprlp(p,t)} = \{r \mid r \in mpr(p,t) \wedge nop(r) < 2\}$$

と定義する．健全性のメトリクス「参加者を欠くプルリクエスト率」R_PRLP(p,t)を次のように定義する：

$$R\_PRLP(p,t) = \frac{|\boldsymbol{mpr\_lp(p,t)}|}{|\boldsymbol{mpr(p,t)}|} \tag{7}$$

5. 実験的評価

5・1 実験の目的と設定

1 節で述べた RQ2 に答えるため，提案メトリクスを GitHub の著名なプロジェクトに適用し，進化を観測する．各プロジェクトの進化を，3.3 節に挙げた都市の属性それぞれの観点で，時系列で可視化する．対象プロジェクトとして，vscode (Microsoft/vscode)，atom (atom/atom)，brackets (adobe/brackets)を選択した．これらはいずれも著名なコードエディタであり，自発的進化がうまく回っているプロジェクトである．

表 1 に 3 つのプロジェクトのサマリデータを挙げる．

表 1: 対象プロジェクトのサマリ (2019-05-10 現在のデータ)

Project	Star	Commit	Issue	Pull Request	Created at
vscode	73,554	48,840	67,825	5,148	2015-09-03
atom	48,654	36,581	14,648	4,393	2012-01-20
brackets	29,756	17,739	9,276	5,474	2011-12-07

5・2 データの収集

提案メトリクスを計算するために，それぞれのプロジェクトから GitHub GraphQL API[10][11]を用いて，様々なデータを抽出した．収集したデータ項目は次の通り：星がつけられた日時，各イシューの作成・クローズ日時，クローズしたかどうか，各プルリクエストの作成・クローズ日時，追加された行数，削除された行数，参加した人数，コメントの数，クローズしたか，マージしたか．

5・3 進化の可視化

収集したデータに基づいて，各プロジェクト p に対して t を p の作成時点から現在までを移動させながら，提案メトリクス m(p,t)を計算した．図 1 にプロットした結果を示す．各グラフは 3 つの線を含み，赤い点線が atom，緑色の鎖線が bracket，青の実線が vscode を表している．横軸は時刻を，縦軸はメトリクスの値を示す．

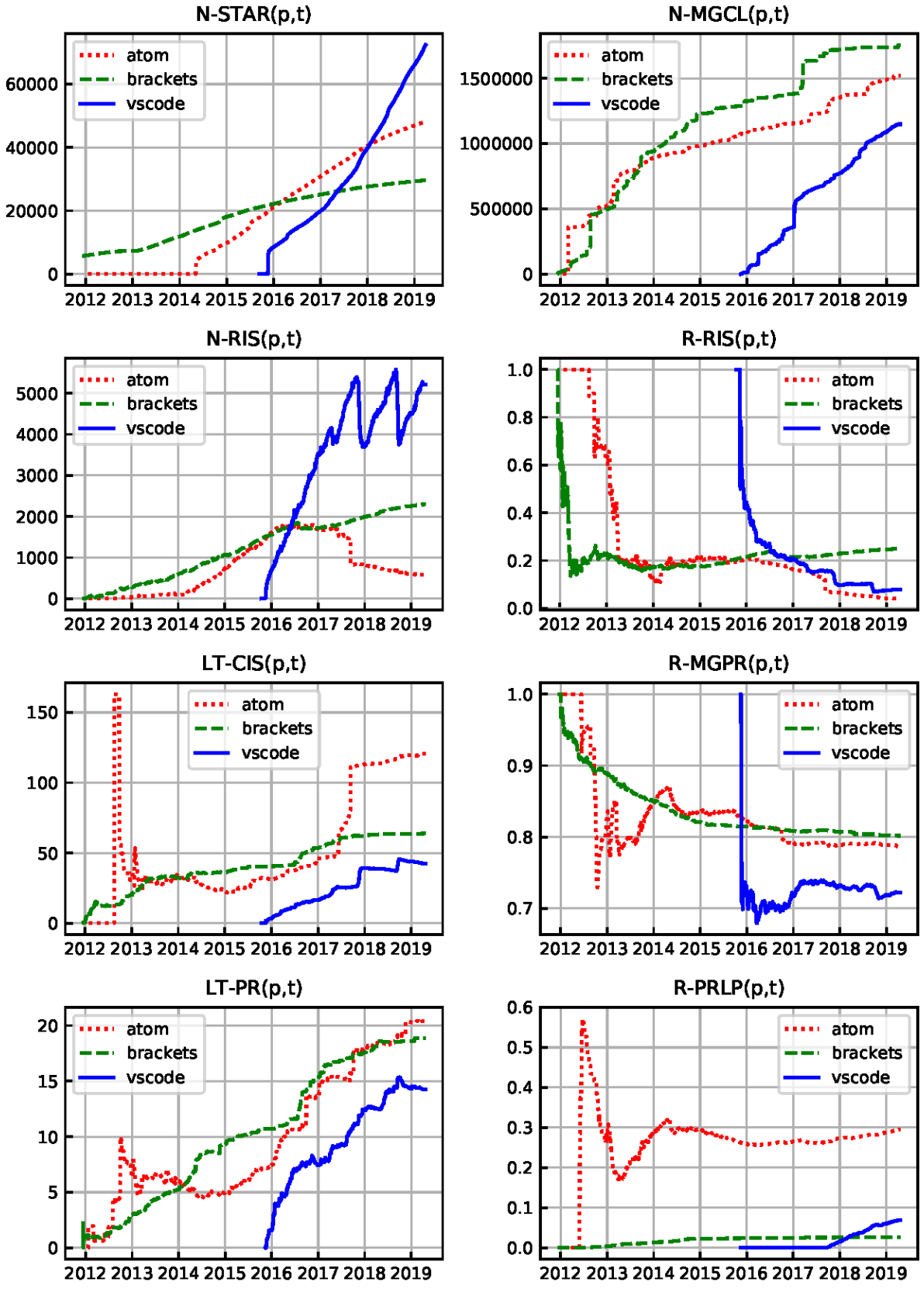

図 1： 提案メトリクスによる GitHub のプロジェクトの進化

5・4 進化の可視化

収集したデータに基づいて,各プロジェクトpに対してtをpの作成時点から現在までを移動させながら,提案メトリクスm(p,t)を計算した．図1にプロットした結果を示す．各グラフは3つの線を含み，赤い点線がatom，緑色の鎖線がbracket，青の実線がvscodeを表している．横軸は時刻を，縦軸はメトリクスの値を示す．以降，各属性について概観する．

人口：図1のN_STAR(p,t)のグラフから，3つの全プロジェクトについて人口が右上がりに増加していることがわかる．特に，vscodeの伸びは顕著である．残り2つのプロジェクトは伸びが若干鈍化しているが，プロジェクトが成熟してきている証であると取れる．いくつかの点で急激な増加がみられるが，これはプロジェクト内で起こった特別なイベントに対応している．例えば，2015年末のvscodeのジャンプは，プロジェクトの初リリースがあった11月18日に相当する．

規模：図1のN_MGCL(p,t)のグラフから，3つの全プロジェクトについて規模が右上がりに増えていることがわかる．vscodeは他の2つより後発にもかかわらず，急激に追いつきつつある．急激なN_MGCL(p,t)の増加は，大規模なコード改訂に相当し，大抵は新機能の追加である．グラフの横ばいの区間は，プロジェクトがあまり活発でない期間である．変更する必要がないか，統制がうまくいっていない可能性がある．

課題：図1のN_RIS(p,t)から，それぞれのプロジェクトの課題の推移に特徴があることが見て取れる．atomは2016年にかけて徐々に課題が増え続け，2017年半ばまで横ばい，以降減少をたどっている．プロジェクトが枯れてきたことを示している．bracketsは，課題が少しずつ積み上がり，2016年に一旦フラットになりつつ，その後また徐々に増えている．vscodeは，2017年以降のこぎり型の推移を示している．開発者からの課題解決の提案を，コアメンバーがあるタイミング（おそらく新しいリリース）で定期的に棚卸していることがわかる．課題の絶対数は各プロジェクト固有の推移を見せているが，R_RIS(p,t)のグラフに示す課題の割合でみるとどのプロジェクトも右下がりに課題数をバーンダウンさせていることがわかる．課題は0になることが理想的だが，人口が増えるにつれイシューの質も玉石混交となる．本質的でない課題が残るのはある程度仕方がないことかもしれない．

課題解決能力：図1のLT_CIS(p,t)から，イシューがクローズされる平均日数が見て取れる．理想的には，ある値以下で推移することが望ましいが，どのプロジェクトにおいても値が徐々に増えていることがわかる．おそらく，人口の増加に伴い非本質的なイシューが乱立し，解決が新しいイシューに追い付いていないと思われる．イシューの質的な分析が必要となるが，これは将来課題としたい．

政府の対応能力：図3のR_MGPR(p,t)から，コアメンバーのプルリクエストの受容性が見られる．vscodeの受容性は2016年以降他の2つのプロジェクトより厳しい値を取っている．受容性の理想値を決めるのは難しい．プルリクエストによって提案されるコード改訂の質によるからである．すべてのコード改訂が非常に重要なら，受容性は1に近づくが，一般に人口が増えると質は下がる傾向にある．その意味では各プロジェクトが現実的な採択率に落ち着いているとみられる．次にLT_PR(p,t)から，コアメンバーの迅速性が見られる．どのプロジェクトでも右上がりを続け，2019年には15日～20日のプルリクエスト承認期間に達している．これは，プルリクエストの数が増えても，決定プロセスを行うコアメンバーのキャパシティはそれほど増えないということを意味している．ただし，イシューの生存期間に比べれば相当短い．これは，イシュー解決するかわからない状態で作成されるのに対し,プルリクエストは解決の道筋が立ってから提案されるからである．最後にR_PPLP(p,t)から，健全性の指標として，きちんと精査されていないプルリクエストの割合が見て取れる．atomは他の2つに比べて高い値となっていて，コアメンバーが第三者の承認なしに変更をコミットしているようである．おそらくatomのプロジェクト運用ポリシーに関連することである．理想的には0に近いことが望ましい．bracketsとvscodeはおおむねルールを順守できている．

5・5 考察

図1において，自発的ソフトウェア進化がうまく回っている3つのプロジェクトの進化を可視化できた．都市の重要な属性の観点からその歴史を眺めた時，それぞれのプロジェクトに特有の性質が見られる．その一方で，3つに共通する観測も見られる：

1) 人口と規模は長い時間をかけて現在まで増え続けている．

2) 人口と規模が増加するにつれ，課題と課題解決にかかる時間も増える．政府の迅速性も下がる．
3) 課題はコミュニティによって活発に対応され，残イシュー数は低い割合に保たれている．
4) 受容性および健全性は，プロジェクトごとに固有の値に収束している．

上記 1)の観測は，人口の増加が自発的ソフトウェア進化の成功のカギを握るということである．より多くの人がプロジェクトに参加することで，より多くの課題やその解決策が生まれ，プロジェクトの新陳代謝が起こる．2)は，ソーシャル・オーバーヘッドの存在を示唆している．より多くの人が参加するにつれ，合議に要するコミュニケーションの負担が増えるのである．3)は，これらの成功プロジェクトでは，重要な課題は一定の割合でバーンダウン[12]されるということである．4)はプロジェクトごとに固有のポリシーが存在することを示唆している．これについてはより深い分析が必要である．

提案手法の長所は，プロジェクトpの任意の時点tにおける状態を定量化できることである．これにより，コミュニティのメンバーは都市としてのプロジェクトの現在状態をいつでも評価できる．これによって，現在何が問題なのか，何をすべきかを想起させることができる．市民自身らによる状態モニタリングは，プロジェクトの統制に不可欠である．また，これらのメトリクスは，CI(継続的インテグレーション)や CD(継続的デリバリー)のツールに統合可能である．一つの応用例は，現在の状態が許容値から外れた場合，ボットが何らかの介入アクションを自動実行するというものである．

提案手法の限界点は，メトリクスが質的な面まで踏み込んでいないことである．例えば，イシューの重要性については考慮しておらず，重要なバグフィクスも新しい機能の要求も区別せずにカウントしている．GitHub では，開発者がイシューに重要度・緊急度を表すラベルを付けることができる．しかしながら，ラベル付けの作法はプロジェクトごとにばらばらのため，一貫した定量化が難しい状態である．このような質的分析は，今後の課題としたい．

6. 関 連 研 究

GitHub リポジトリをマイニングする研究は，これまでにも多く存在する．Gousious ら[13]は，GitHub リポジトリを 2012 年 2 月～2013 年 8 月の間分析し，そのサマリを作成している．Steinmacher ら[14]は，コアメンバー以外のプルリクエストの量的・質的評価を行っている．Trockman ら[1]は，バッジを分析しており，GitHub の透明性に影響することを示している．Yang ら[15]は，4 つのオープンソースプロジェクトにおける貢献とレビューの関係を調べている．Constantiou ら[16]は，Ruby プロジェクトのコミット回数，行数，プロジェクト数をタイムライン上に要約している．Pinto ら[17]は，GitHub における開発言語を因果とする貢献者のふるまいを分析している．Borges ら[18]は，GitHub の星に着目し，多くの星を稼いだプロジェクトの性質を分析している．

我々の研究は，GitHub のプロジェクトを分析する上記の研究に関連している．しかしながら，対象を自発的ソフトウェア進化に寄与する要因に着目していること，プロジェクトの任意の時点 t の状態に着目していること，都市としてのプロジェクトという新しい概念を取り入れたことに独自性を有している．

7. 結 語

自発的ソフトウェア進化を理解するために，本研究ではソーシャルコーディング基盤におけるソフトウェアプロジェクトの歴史を分析する方法を提案した．都市としてのプロジェクトという概念を取り入れ，自発的進化にとって重要な属性を洗い出した．次に，プロジェクト p の任意の時点 t における状態 S(p,t)を可視化するために，8 種類のメトリクスを提案した．その結果，プロジェクトの状態を都市の人口，規模，課題，課題解決能力，政府の対応能力の観点から評価することが可能になった．さらに，提案メトリクスを GitHub の著名なコードエディタのプロジェクトに適用し，それらの進化を可視化した．自発的進化がうまく回っているこれらのプロジェクトに共通するいくつかの観測を得た．

現在，より多くのプロジェクトに対してメトリクスを適用し，自発的進化の良いパターン・悪いパターンを発見する実験を継続中である．このようなパターンを見つけることで，継続的で健全な自発的進化を統制できる．質的な評価も重要な将来課題である．

謝　　辞

本研究の一部は JSPS 科研費 JP19H01138，JP17H00731，JP18H03242，JP18H03342，JP19H04154，JP19K02973 の助成を受けている．

文　　献

(1) A. Trockman, S. Zhou, C. K¨astner, and B. Vasilescu, “Adding sparkle to social coding: An empirical study of repository badges in the np ecosystem,” in Proceedings of the 40th International Conference of Software Engineering, ICSE ’18, (New York, NY, USA), pp. 511–522 ACM, 2018.
(2) “GitHub: The world’s leading software development platform.” https //github.com.
(3) “Bitbucket: The Git solution for professional teams.” https://bitbucket.org.
(4) M. M. Lehman, “Programs, life cycles, and laws of software evolution, Proceedings of the IEEE, vol. 68, pp. 1060–1076, Sep. 1980.
(5) K. Matsumoto, H. Hata, M. Nakamura, H. Tamada, A. Ihara S. Morisaki, M. Tsunoda, K. Toda, M. Ohira, and A. Monden, “Developmen of fundamental technologies to accelerate spontaneous software evolution.” JSPS Kakenhi Grant-in-Aid for Scientific Research (A) JP17H00731, 2017.
(6) Microsoft, “Visual Studio Code.” https://github.com/microsoft/vscode.
(7) M. Nakamura and L. du Bousquet, “Constructing execution and lifecycle models for smart city services with self-aware IoT” in IEE 12th International Conference on Autonomic Computing (ICAC2015) pp. 289–294, July 2015.
(8) “Atom: The hackable text editor.” https://github.com/atom/atom.
(9) Adobe, Inc., “An open source code editor for the web, written in JavaScript, HTML and CSS.” https://github.com/adobe/brackets.
(10) GitHub, “GraphQL API v4.” https://developer.github.com/v4/.
(11) “GraphQL June 2018 edition.” https://graphql.github.io/graphql-spec June2018/.
(12) G. Dinwiddie, “Feel the burn, getting the most out of burn charts,” Bette Software, vol. 11, no. 9, pp. 26–31, 2009.
(13) G. Gousios, M. Pinzger, and A. v. Deursen, “An exploratory study of the pull-based software development model,” in Proceedings of the 36t International Conference on Software Engineering, ICSE 2014, pp. 345–355, ACM, 2014.
(14) I. Steinmacher, G. Pinto, I. S. Wiese, and M. A. Gerosa, “Almost there A study on quasi-contributors in open-source software projects,” in 201 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 256–266, May 2018.
(15) X. Yang, N. Yoshida, R. G. Kula, and H. Iida, “Peer review social network (peRSoN) in open source projects,” IEICE Transactions of Information and Systems, vol. E99-D, pp. 661–670, 3 2016.
(16) E. Constantinou and T. Mens, “Socio-technical evolution of the rub ecosystem in GitHub,” in 2017 IEEE 24th International Conference of Software Analysis, Evolution and Reengineering (SANER), pp. 34–44 Feb 2017.
(17) G. Pinto, I. Steinmacher, and M. A. Gerosa, “More common that you think: An in-depth study of casual contributors,” in 2016 IEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol. 1, pp. 112–123, March 2016.
(18) H. Borges, A. Hora, and M. T. Valente, “Understanding the factors that impact the popularity of Github repositories,” in 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 334 344, Oct 2016.