Vol.2017-SE-195 No.23
2017/3/12

# グラフモデルを用いたOSSコミュニティ進化構造分析方法の提案と評価

加藤 聖也[†1] 稲垣 遥太[†1] 青山 幹雄[†1]

**要約**:OSS(Open Source Software)開発コミュニティ内の開発者ネットワークは複雑化している. 本稿では, OSS コミュニティ構造を, グラフモデルを用いた表現方法を提案する. さらに, 提案するグラフモデルの構造をグラフデータベース Neo4j を用いて分析する方法も提案する. 本提案を, GitHub 上の実際の複数の機械学習ライブラリの OSS コミュニティに適用し, コミュニティ進化の構造をモデル化する. このモデルを分析することによって, 非中核, 準中核, 中核の 3 層で構成されるコミュニティ進化モデル, 開発者の行動による3つの成長パターン, 開発段階による開発者間の相互作用の変化の 3 つの特性を発見した. この結果に基づき, 提案方法の妥当性, 有効性を示す.

**キーワード**:オープンソースソフトウェア, グラフデータベース, マイニングリポジトリ

# A Structural Analysis Method of OSS Community Evolution Based on Semantic Graph Models.

SEIYA KATO[†1] YOTA INAGAKI[†1] MIKIO AOYAMA[†1]

## 1. 研究背景と課題

OSS(Open Source Software)開発コミュニティ内の開発者ネットワークは複雑化している. OSS コミュニティを対象としたリポジトリマイニングの解析は既に提案されているが, OSS コミュニティの進化, すなわち動的な構造変化の分析は確立されていない. また, グラフモデルを用いて OSS コミュニティ構造を明らかにする研究も未確立である. 本稿では, 以下を課題とする.

(1) OSS コミュニティ構造のグラフモデル表現

OSS コミュニティのグラフモデル表現が適切であることを, 具体的なグラフモデルをすることで明確にする.

(2) OSS コミュニティ進化構造分析方法の確立

マイニングによる静的な構造解析では, コミュニティの進化を分析できない. そこで, コミュニティ進化構造の分析方法が必要である.

## 2. 関連研究

### 2.1 OSS コミュニティ

OSSコミュニティに参画している開発者の果たす役割を明らかにすることは, OSS コミュニティ進化構造の分析上で重要である. OSS コミュニティに参画している開発者を 8 つの役割に分類する提案や, OSS コミュニティ構造を, 中核, 非中核的開発者に分類するクラスタリングが提案されている[1, 2, 9].

### 2.2 ソフトウェアリポジトリマイニング

ソフトウェアリポジトリマイニング(Mining Software Repositories)は, ソフトウェアリポジトリの分析に関する幅広い研究が行われている. 一方, ソフトウェア進化の研究において, リポジトリから適切な情報の抽出や, 関係や傾向を明らかにする試みが行われている[7, 8, 15]. また, ソフトウェアリポジトリからデータをマイニングし, 事前に定義した開発者の行動に基づいて, 貢献度を評価する提案がある[6].

### 2.3 グラフデータベース[14]

グラフデータベース(グラフ DB)は, グラフデータモデルを基礎とする DB 管理システムである. また, ノードと関係の抽象概念を連接構造にすることで, 問題領域のスキーマを定義することなくモデルを構築できる. 構築されたモデルは, 意味付けされた構造化が可能である.

## 3. アプローチ

研究課題を解決するためのアプローチを以下に示す.

(1) OSS コミュニティ構造分析へのグラフ DB の適用

OSS コミュニティ構造を, プロパティグラフを扱うグラフ DB を用いてモデル化する.

(2) グラフデータベースを用いた分析

グラフ DB が持つクエリを利用して可視化し, 分析を行うことにより, OSS コミュニティの進化の特性を得る.

## 4. 提案方法

### 4.1 提案プロセス

以下に本稿で提案するプロセスを示す(図 1).

†1 南山大学 理工学部 ソフトウェア工学科
Dep. of Software Engineering, Nanzan University

Vol.2017-SE-195 No.23
2017/3/12

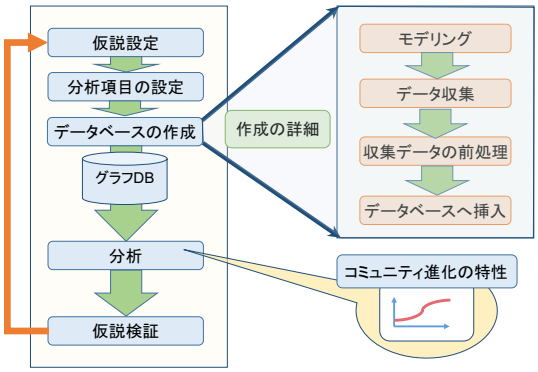



図 1 提案プロセス

Figure 1 Proposed Process

#### 4.1.1 仮説設定

OSS コミュニティが，どのような要素，要因によって進化しているか仮説を立てる．このプロセスは，検証プロセスの結果に基づいて，必要があれば追加，変更する．

#### 4.1.2 分析項目の設定

設定した仮説に基づいて，仮説ごとに明らかにするべき項目を列挙する．列挙した項目を統合して，分析項目を設定する．

#### 4.1.3 グラフ DB の作成

グラフ DB の作成は，モデリング，データ収集，収集データの処理，データベースへの挿入の 4 つのプロセスからなる．

(1) モデリング

新規のノード，関係のデータを追加する必要がある時には，モデル化を行い，分析すべき関係の意味を定義する．

(2) データ収集

データ収集は主に，OSS 開発のプラットフォームが提供している API を使用する．OSS リポジトリから収集するデータとして，GitHub から得られるデータを想定する．

(3) 収集データの前処理

収集するデータは，分析項目で設定した内容に沿う必要がある．取得したデータには，分析に不必要なものが存在する．そこで，DB へデータを挿入する前に，分析に必要なデータをノード及び関係の属性として定義する．

(4) データベースへの挿入

処理が完了した収集データをグラフ DB へ挿入する．挿入には，利用するグラフ DB で使用可能なクエリ言語を使用する．

#### 4.1.4 分析

設定した分析項目に従って分析し，結果を出力する．分析対象の開発コミュニティの性質は同一ではないこと考慮する必要がある．

#### 4.1.5 仮説検証

分析で得られた結果から，立てた仮説を検証する．必要に応じ，得られた結果に基づいて，仮説の追加や変更を行い，はじめからの一連のプロセスを繰り返す．

### 4.2 コミュニティ進化構造分析方法

#### 4.2.1 分析モデル

分析モデルには，空間的な分析と時間的な分析の2つ軸によって分け，マクロな視点とミクロな視点により分析する(図 2)．

コミュニティの進化構造分析はマクロ進化分析，開発者の成長に関する分析はミクロ進化分析であると言える．

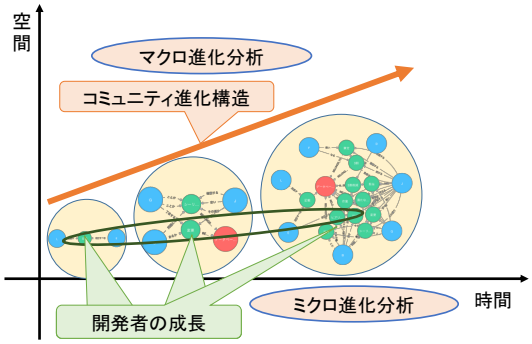



図 2 分析モデル

Figure 2 Analysis Model

#### 4.2.2 マクロ進化分析

(1) コミュニティの代謝構造

OSS コミュニティが進化していくためには，ソフトウェアの成長が重要である．OSS コミュニティが開発するソフトウェアの成長に伴い，新たにコミュニティに対して貢献する開発者が参入してくる特性があると考える．OSS コミュニティが開発を継続するためには，開発しているソフトウェアを利用するユーザの存在は大きい．ユーザからコミュニティに参画する開発者が現れることが推測される．参画した開発者が成長し，中核的な役割を担うようになる特性があると推測する．

また，グラフ上の進化ダイナミクスには循環[11]が定義されている．これに対して本稿では，上記から推測される特性より，コミュニティの代謝構造として，定期的に開発者が参入していること，開発者の役割が変化していることと定義する．

そこで，ある期間ごとに，コミュニティへの貢献度が高い開発者と，新規開発者をグラフとして可視化する．可視化されたグラフを期間ごとに区切ることで，コミュニティへの代謝構造を分析する．

(2) コミュニティの進化ステージモデル

OSSコミュニティ進化には，段階があると考えられる．ソフトウェアの開発は，中核的開発者の数が少ないと，全体としては進化がゆるやかだと考えられる．ソフトウェアの進化に伴い，それを利用するユーザが増加し，機能変更，追加の要求によって新規開発者が参入する．新規開発者は中核的開発者へと成長し，中核的開発者が増加したため，コミュニティの進化が加速する．そこで，中核的開発者による貢献度の変化に着目することで，OSS コミュニティ進化構造に段階があることを明らかにする．

#### 4.2.3 ミクロ進化分析

(1) 開発者の成長要因

開発者の行動の特性に，非中核的または，中核的開発者に成長する主たる要因となると考えられる．そこで，一定期間における，開発者の行動をグラフとしてモデル化することによって，その要因を分析する．

(2) 中核的，新規開発者に共通する更新ファイル数の推移

新規開発者と中核的開発者が更新を行うファイルには，共通するファイルが存在すると考えられる．また，OSS コミュニティ進化の段階によって，更新される対象のファイルには，開発者と更新されるファイルの相互作用の変化が現れると考えられる．そこで，一定期間ごとに，新規開発者と中核的開発者が更新を行ったファイルを抽出し，共通するファイル数を分析する．これより，OSS コミュニティ進化の段階による中核的，新規開発者の関係が分析できると考える．

## 5. プロトタイプの実装

### 5.1 プロトタイプ作成の目的

コミュニティの分析のため，以下の 3 つを目的とする．

(1) リポジトリデータの取得

(2) 挿入データへの前処理

(3) グラフデータベースへのデータ挿入

### 5.2 システムの構成

以下にシステム構成図を示す(図 2).

#### 5.2.1 GitHub

GitHub より提供されている GitHub API v3 を利用し，GitHub 上のリポジトリからデータを取得する．API を利用してデータを取得する処理及び，データの整形はプログラムの一部に組み込んだ．

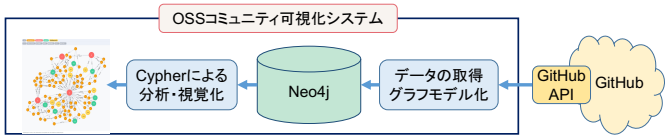



図 3 システム構成図

Figure 3 A System Configuration

#### 5.2.2 Neo4j

取得したデータをグラフデータベース Neo4j[10]へ挿入する．データの挿入には，Cypher Query を利用した．また，Neo4j Bolt ドライバーを利用して，プログラムの一部に組み込んだ．

#### 5.2.3 Neo4j Browser

Neo4j Browser は，Neo4j に組み込まれたクライアントアプリケーションである．Cypher クエリを実行することによって，Neo4j のデータベースに保存されているデータを Browser 上で可視化できる．

## 6. GitHub 上の OSS コミュニティへの適用

### 6.1 分析対象の OSS コミュニティ

機械学習に関する OSS コミュニティ，TensorFlow[5]，Caffe[3]，Chainer[13]，Jubatus[12]を分析対象とした．

分析において，コミュニティ進化を 3 ヶ月ごとに分割し，それぞれを 1 期，2 期などとした．また，3 ヶ月に満たない部分は分析対象外とし，2011 年 10 月 23 日から 2016 年 12 月 1 日までを分析対象とした．

### 6.2 グラフデータベースの作成

以下にグラフ DB の概要を示す(図 3). GitHub 上から取得したデータをノード，関係及びプロパティとして定義した．それぞれの定義を表 3, 4, 5 に示す．

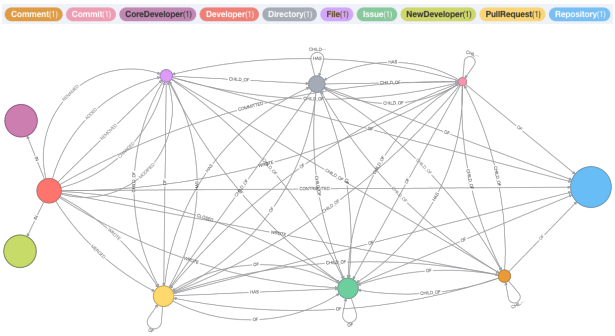



図 4 作成したグラフ DB のモデル

Figure 4 A Graph Database Model

## 7. 発見

### 7.1 コミュニティの代謝構造

コミュニティを期間ごとに区切り，期間ごとに新規参入した開発者と貢献が多い開発者をサブグラフとして可視化した．グラフのノードは，赤は開発者，緑がある期間内に参入した開発者ノードを繋ぐもの，紫はある期間内で貢献が多い開発者ノードを繋ぐもの，となっている．

#### 7.1.1 分析結果

(1) Chainer (図 5)

コミュニティ発足時の開発者の一部が，全期間にわたって中核的開発者として活動している．また，それぞれの期間において，新規開発者も中核的な役割を担っている構造が得られた．その後も，それらの開発者が中核的な役割を担うことは稀であった．

(2) Caffe (図 6)

1 期目，2 期目に参入した開発者が，全期間にわたって中核的開発者として活動している．また，Chainer と同様に，ある期間においてのみ，中核的な役割を果たす新規開発者が存在している．

(3) Jubatus (図 7)

早い段階でコミュニティに参入した開発者が，多くの期間にわ

たって中核的開発者として活動している．全期間にわたり，新規開発者は少ないため，新規開発者が参入した期間の中核的な役割を果たす構造になっている．

**7.1.2 3 層で構成されるコミュニティ進化モデル**

分析方法 1 より，コミュニティ上の開発者の構造を以下の 3 層で構成されるコミュニティ進化モデルとして定義できる．

(1) 非中核：中心的役割を果たさなかった開発者
(2) 準中核：一定期間のみ中心的に貢献する開発者
(3) 中核：長期間，中心になって貢献している開発者

中核的開発者の構成は，コミュニティ発足時の一部の開発者が主体になっている．この構造はどのコミュニティで見られ，コミュニティが進化しても，大きな変化はない．一方，コミュニティ進化に伴い，準中核的開発者は入れ替わっている．

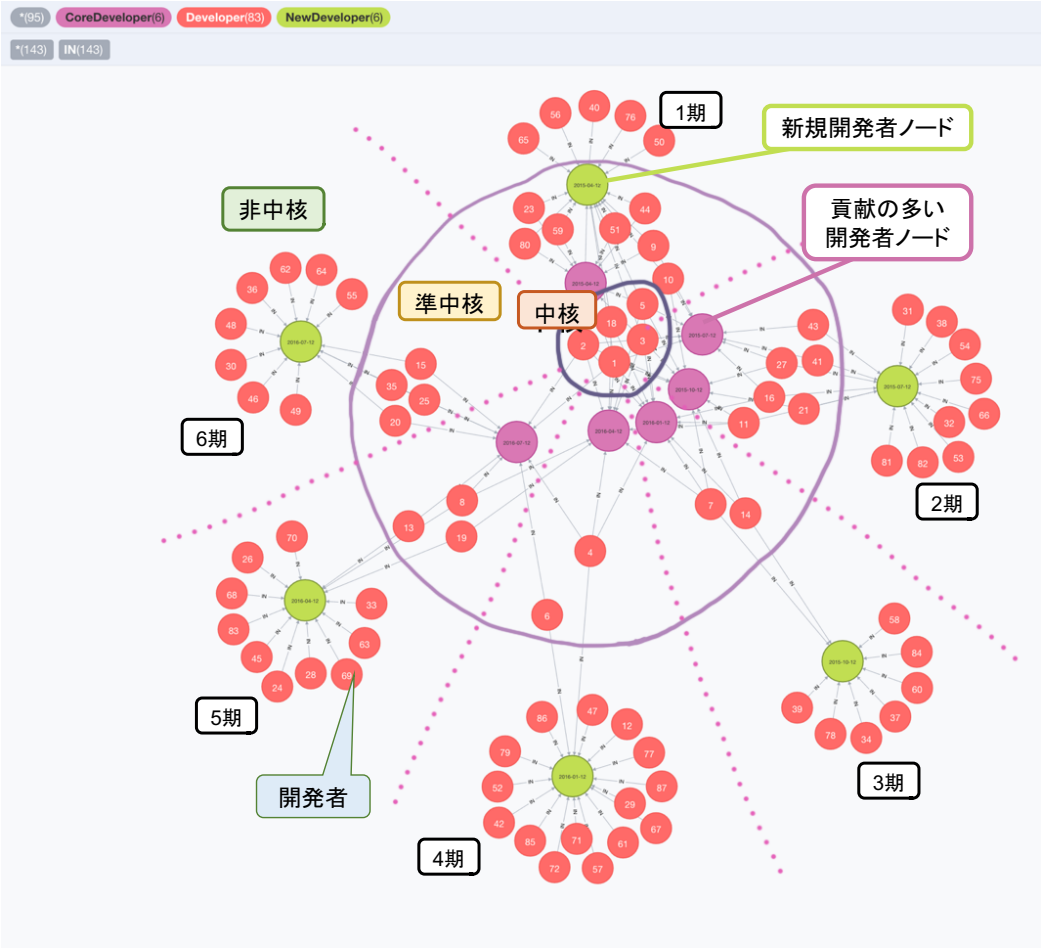



図 5　3 層コミュニティ進化モデル(Chainer)
Figure 5 Three-Layered Community Model of Chainer

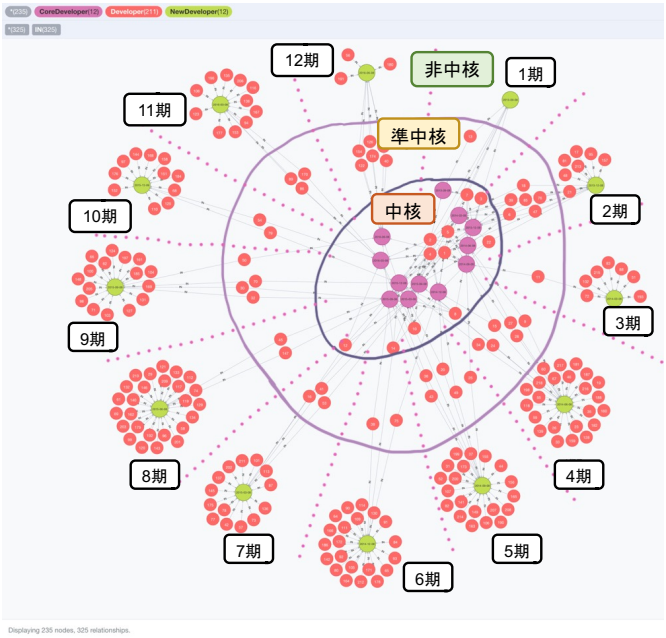



図 6　3 層コミュニティ進化モデル(Caffe)
Figure 6 Three-Layered Community Model

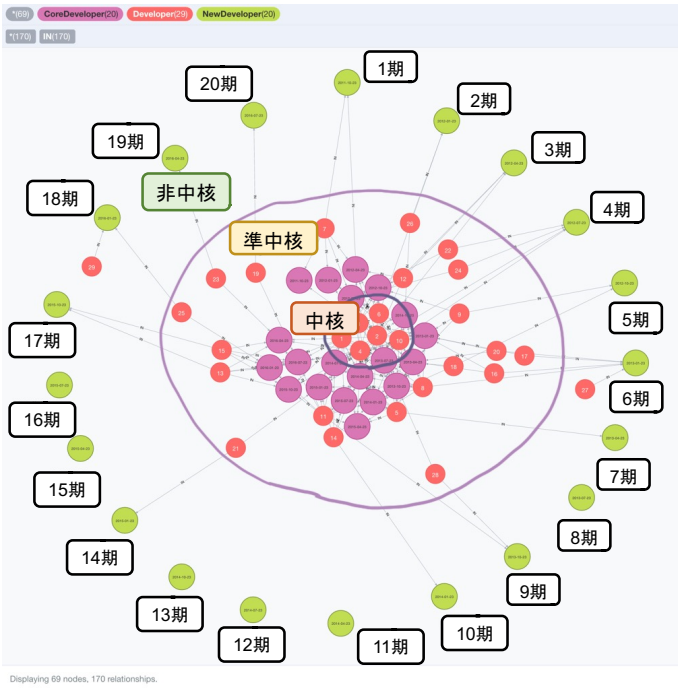



図 7　3 層コミュニティ進化モデル(Jubatus)
Figure 7 Three-Layered Community Model

これは，新規開発者から準中核的開発者に成長する者が現れるが，一定期間のみ中心的な役割を果たして，その後はフェードアウトする傾向が考えられる．

### 7.2 開発者の成長要因

新規参入時からデータ取得時までの，貢献度に差がある開発者 3 人を抽出した．3 人の開発者が，新規参入時から半年間の行動と，その行動に対する他の開発者の行動の関係をサブグラフとして可視化した．ここで，行動は Comment, Issue, Pull Request である．グラフのノードは，赤色が開発者，緑が Issue，黄が Pull Request，橙が Comment を表している．

#### 7.2.1 分析結果

(a) 貢献度が低い開発者 (図 8)

コメントの投稿が多く，他の開発者とのやりとりが見られる．また，Issue に比べ，Pull Request の作成数は少ないため，要望やバグ報告は積極的であるが，機能実装へは消極的な開発者であると考えられる．

(b) 貢献度が中程度の開発者 (図 9)

コメントの投稿は少なく，他の開発者とのやりとりは見えない．しかし，Issue と Pull Request の作成数は同等であり，機能実装へ積極的な開発者であると考えられる．

(c) 貢献度が高い開発者 (図 10)

コメントの投稿は多く，他の開発者とのやりとりも多く見ることができる．また，Issue も Pull Request も同等に作成しており，要望，バグ報告，機能実装のいずれも積極的に行う開発者であると考えられる．

#### 7.2.2 開発者の行動による 3 つの成長パターン

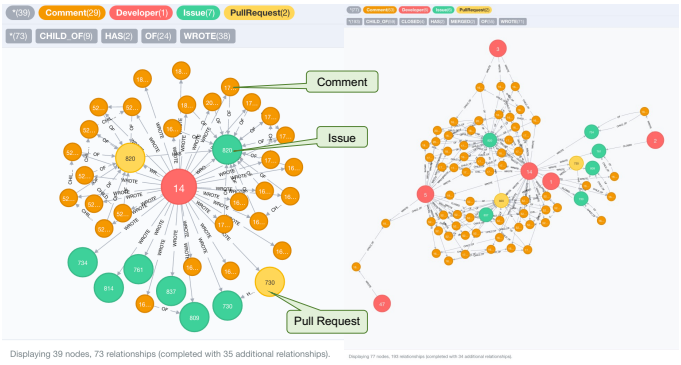
分析結果より，開発者の行動は，以下の 3 層構造のコミュニティ進化モデルに対応尾けることができる．

(1) 非中核的開発者: バグ報告や機能提案を中心に活動

(2) 準中核的開発者: 機能実装を中心に活動

(3) 中核的開発者: バグ報告，機能提案，実装のいずれも積極的に活動

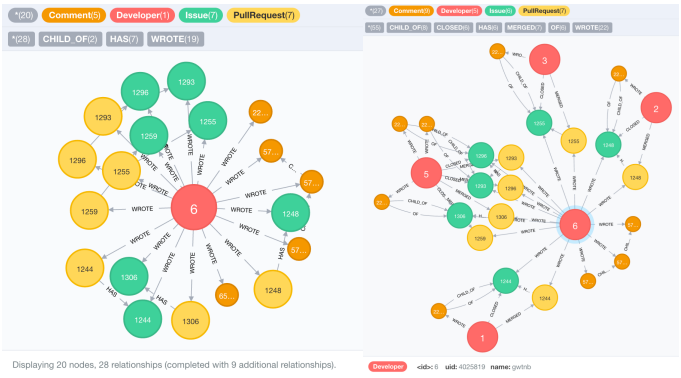
また，3 人の開発者がコミュニティに貢献したコミット数の累計の推移を分析すると，ほぼ貢献数が増加しない者，貢献数は増加しているが増加率が低い者，貢献数が増加し，増加率も高い者に分かれた(図 11). コミット数は，開発者が新規参入してから3期分のデータを使用した．




(a) 開発者の行動　　(a') 行動による相互作用

図 **8** 開発者の行動特性(a)

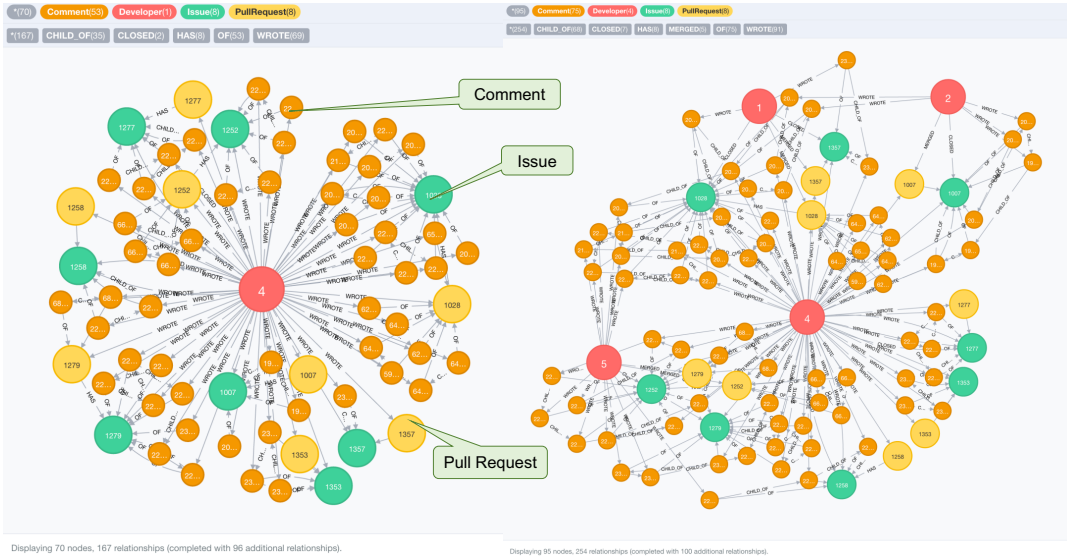
Figure 8 Behavioral Characteristics of Developers (a)



(b) 開発者の行動　　(b') 行動による相互作用

図 **9** 開発者の行動特性(b)

Figure 9 Behavioral Characteristics of Developers (b)




(c) 開発者の行動　　(c') 行動による相互作用

図 **10** 開発者の行動特性(c)

Figure 10 Behavioral Characteristics of Developers (c)

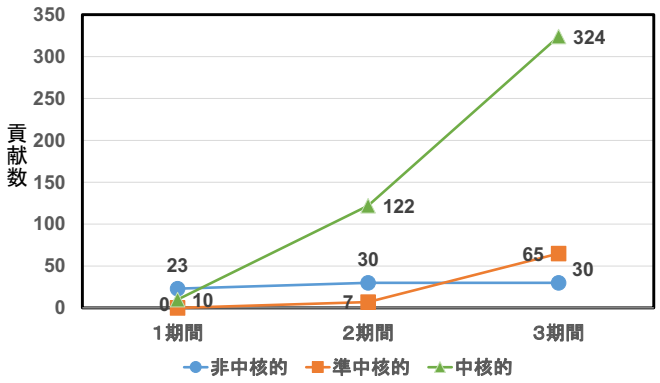



図 **11** 開発者の 3 つ成長パターン
Figure 11 Three Growth Pattern of Developers

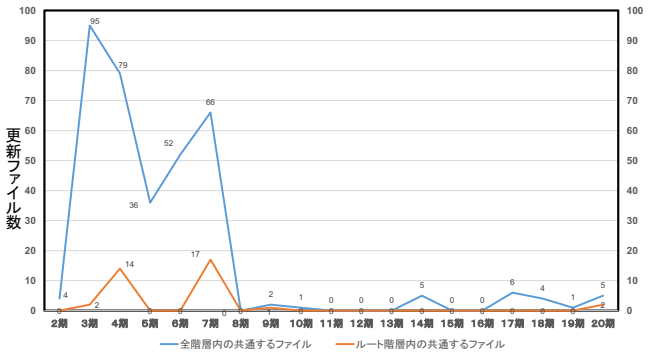



図 **13** 共通更新ファイルの推移(Jubatus)
Figure 13 Number of Common Changed Files of Jubatus

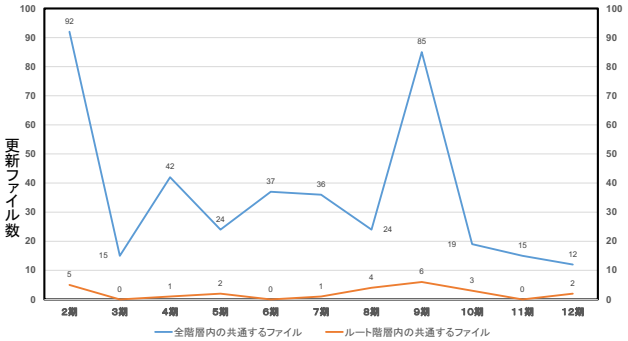



図 **14** 共通更新ファイルの推移(Caffe)
Figure 14 Number of Common Changed Files of Caffe

### 7.3 コミュニティの進化段階

#### 7.3.1 コミュニティの進化ステージモデル

Jubatus での，貢献数の累計の推移を図 12 に示す．4 期あたりから中核的開発者による貢献数の増加が見られ，開発スピードが速くなっている．また，12 期あたりから，貢献数は減少し始めて，開発スピードが遅くなっている．このことから 4 期目あたりから立ち上げ期から成長期へと推移し，12 期あたりから成長期から成熟期へと推移していると考えられる．

成熟期への推移は，PFN が開発を行っている Chainer のコミュニティが 15 期目あたりに発足していることもひとつの要因であると考えられ，Jubatus の開発から Chainer の開発へと推移したため，積極的な開発が行われなくなったというシナリオも推測できる．

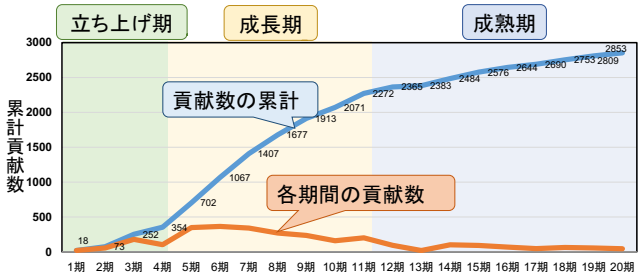



図 **12** 累計貢献数の推移
Figure 12 Trend of Sum of Contributions

#### 7.3.2 中核的，新規開発者に共通する更新ファイル数の推移

7.3.1 より，コミュニティには立ち上げ期，成長期，成熟期の 3 段階進化があることが明らかとなった．そこで，中核的開発者と新規開発者が更新を行うファイルにも，進化段階による特徴が現れると推測できる．

(1) Jubatus (図 13)

2 期から 3 期は更新が全体の共通しているファイルは一時的に増加しており，5 期に向かって半減している．5 期から 8 期にかけては，2 期から 5 期までと同じような山形が見られる．8 期以降は概ね更新が共通したファイルは少なく，全ての期で 10 以下であった．ルート直下に存在する共通したファイルも，概ね同じ形のグラフが見られた．2 期から 5 期は立ち上げ期，5 期から 8 期は成長期，8 期以降は成熟期だと考える．

(2) Caffe (図 14)

2 期では多くのファイルが共通している．2 期から 3 期で一時的に大きく減少した後は，8 期までは振れ幅の小さい増減を繰り返している．この期間を開発の立ち上げ期．その後，8 期から 10 期にかけて山形が見られ，10 期以降は減少傾向にある．2 期から 8 期が立ち上げ期から成長期であり，8 期から 10 期は成長期であり，この後，成熟期に進むと考える．

以上より，リポジトリの成長に伴い，中核的，新規開発者が共通して更新を行うファイルに，ハイプ曲線のような特徴[4]が表れると考えられる．これより，開発の段階について，以下の特徴が考えられる．

(1) 立ち上げ期: 新プロダクトへの期待による積極的な更新
(2) 成長期: 新規開発者による機能提案，実装
(3) 成熟期: プロダクトの完成

また，ファイル更新時は，機能実装による追加，修正が発生するため，ルート階層内に存在するドキュメントやコンフィグファイルなどが同時に更新される可能性が高いと考えられる．

## 8. 考察

### 8.1 グラフモデルによる OSS コミュニティ構造

本稿で提案したプロセスを実際のコミュニティに適用し，設定した仮説に対して，適切に分析項目を設定できた．そのため、

OSS コミュニティの動的な構造変化の分析に必要なデータを十分に収集でき，グラフモデル表現によるコミュニティのモデリングが可能となった．

### 8.2 OSS コミュニティの動的な構造変化の分析方法の確立

コミュニティの進化を時系列に分割して，グラフとして可視化してすることにより，以下の OSS コミュニティの動的な構造を明らかにした．

(1) 非中核，準中核，中核的開発者の 3 層コミュニティ進化モデル
(2) 開発者の行動による成長パターン
(3) 開発段階による開発者間の相互作用の変化

#### 8.2.1 コミュニティ構造と進化モデル

関連研究では，OSS コミュニティの構造を，中核，非中核的開発者として分類していた[2]．本稿では，開発者を中核，非中核に加えて準中核的開発者に分類が可能であることを明らかにした．これより，コミュニティ上の開発者の役割構造をより明確にすることが可能になった．

進化モデルとして，中核的開発者を中心に開発が進み， 新規開発者の参入と，その中から準中核的開発者が現れ，それが入れ替わりながら進化していることを明らかにした．これはコミュニティの運用の支援に活用が期待できる．

#### 8.2.2 開発者の行動による成長パターンの特性

開発者個人の行動に着目して分析を行うと，開発者が中心とする行動から，開発者がコミュニティ内でどのような役割を担う可能性があるかを，パターン化できると考えられる．

行動パターンは，2 つに分類できる．

(1) バグ報告や機能提案を中心とする
(2) 機能の実装を中心とする

非中核的開発者は(1)の行動をとり，準中核，中核的開発者は(2)の行動をとっている可能性が高い，

成長パターンは，3 つに分類できる．

(1) 貢献数がほぼ増加せず，成長率が低い
(2) 貢献数は増加するが，成長率が低い
(3) 貢献数が増加し，成長率も高い

非中核的開発者は(1)，準中核的開発者は(2)，中核的開発者は(3)の成長パターンとなっている．準中核と中核的開発者の違いは，他の開発者とのコミュニケーション量の違いであり，継続的に機能提案，実装を行う開発者は，他の開発者とのコミュニケーションも多いと考えられる．これらの成長パターンから開発者の成長の予測へ活用することが期待できる．

#### 8.2.3 開発段階による開発者間の相互作用の変化

開発段階によって，ファイル更新に中核，新規開発者間の相互作用がみられた．開発は立ち上げ期，成長期，成熟期の 3 つに定義でき，立ち上げ期と成長期で双峰性が見られ，ハイプ曲線[4]の波形と似た意味を持つと考えられる．

#### 8.2.4 計算量の評価

開発者の行動に対する分析にて，可視化のためにサブグラフを出力した．その時の Cypher Query の計算量及び，グラフデータベースに存在するノードと関係の数を示す(表 1, 2)．

表 1 Chainer が持つノードと関係
Table 1 Number of Nodes and Relationships of Chainer

| | ノード | 関係 |
|---|---|---|
| Chainer | 19,099 | 141,061 |

表 2 Cypher Query の計算量
Table 2 Number of Cypher Queries

| | 対象の開発者と他の開発者(ms) | 対象の開発者のみ(ms) |
|---|---|---|
| 非中核的開発者 | 4,118 | 1,330 |
| 準中核的開発者 | 48 | 41 |
| 中核的開発者 | 275 | 46 |

本研究の分析では，実用的な時間でクエリの結果を得ることができた．しかし，グラフデータベースを探索する経路が多いクエリほど，実行時間がかかり，計算量が増加していく．そのため，データベースの規模増大に伴い，グラフの可視化が困難になる可能性が考えられる．

## 9. 今後の課題

### 9.1 分析結果の妥当性検証

本稿では，機械学習ライブラリの開発コミュニティに対して分析を行ったが，結果の妥当性検証は行っていない．そのため，発見した特性を別の OSS コミュニティに適用し，妥当性の検証を行う必要がある．

### 9.2 コミュニティ進化モデルの拡張

本稿では，個々の開発コミュニティに対してのみ分析を行った．しかし，実際には，各コミュニティ間をまたいで活動する開発者が存在すると考えられる．この課題には，コミュニティ群を対象とするモデリングを行う必要がある．

また，本稿の提案が，開発コミュニティに限らない，様々なコミュニティを対象としても有効な分析方法であることを検証する必要がある．

### 9.3 グラフモデルの改善

関係が多いグラフデータベースの探索では，計算量が増加する傾向にあることが明らかであった．しかし，本稿で作成したグラフデータベースのモデルには，不必要なプロパティや関係が存在している可能性がある．これより，モデルの最適化を行うことにより，Cypher Query の実効速度の向上が期待できる．

### 9.4 分析方法の改善

本稿において，分析には Cypher Query を利用したが，可視化できる結果に重点をおいたため，クエリによる探索が最適化されておらず，計算量の増大を招いている可能性が高い．また，対象とするコミュニティによっては，グラフデータベースの大きさも変化するため，同一のクエリによる分析では，グラフの可視化が困難になっていく可能性がある．よって，この問題のスケーラビリティに対応するため，Cypher Query を改善する必要がある．

## 10. まとめ

本研究では，OSS コミュニティ構造にグラフモデル表現が適していることを明らかにし，時間変化に伴う OSS コミュニティの動的な構造変化の分析方法を提案した．OSS コミュニティ構造を分析することにより，以下の 3 つの特性を発見した．

(1) 非中核，準中核，中核的開発者の 3 層コミュニティ進化モデル

(2) 開発者の行動による成長パターン

(3) 開発段階による開発者間の相互作用の変化

これらの特性を用いることで，OSS コミュニティの運用や開発者の成長の支援への活用が期待できる．

## 参考文献

[1] M. Y. Allaho, et al., Analyzing the Social Ties and Structure of Contributors in Open Source Software Community, Proc. of ASONAM 2013, ACM, Aug. 2013, pp. 56-60.

[2] A. Bosu and J. C. Carver, Impact of Developer Reputation on Code Review Outcomes in OSS Projects: An Empirical Investigation, Proc. of ESEM '14, Article No. 33, ACM, Sep. 2014, 10 pages.

[3] BVLC, Caffe, http://caffe.berkeleyvision.org/

[4] J. Fenn, et al., Mastering the Hype Cycle, Harvard Business School Press, 2008

[5] Google Inc., TensorFlow, https://www.tensorflow.org/

[6] G. Gousios, et al., Measuring Developer Contribution from Software Repository Data, Proc. of MSR '08, ACM, May 2008, pp. 129-132.

[7] H Kagdi, et al., A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution, J. of Software Maintenance and Evolution Research and Practice Vol. 19, No. 2, Mar. 2007, pp. 77-131.

[8] W. Leibzon, Social Network of Software Development at GitHub, Proc. of ASONAM 2016, IEEE/ACM, Aug. 2016, pp. 1374-1376.

[9] K. Nakakoji, et al., Evolution Patterns of Open-Source Software Systems and Communities, Proc. of IWPSE '02, ACM, May. 2002, pp. 76-85

[10] Neo Technology, Neo4j, 2016, http://neo4j.com/

[11] M. A. Nowak, Evolutionary Dynamics, Belknap Press, 2006 [竹内康博，ほか (監訳), 進化のダイナミクス，共立出版, 2008].

[12] Preferred Networks Inc., Jubatus, http://jubat.us/ja/

[13] Preferred Networks Inc., Chainer, http://chainer.org/

[14] I. Robinson, et al., Graph Databases, O'Reilly, 2015.

[15] M. Russell, Mining the Social Web, 2nd ed., O'Reilly, 2014.

表 3 ノードの定義

Table 3 Definition of Nodes

| ノード名 | 定義 |
|---|---|
| Comment | 開発者による，Issue，Pull Request 等に対するコメントを表す． |
| Commit | 開発者による，ファイルの追加や変更の履歴を表す． |
| CoreDeveloper | ある期間における中核的開発者を表す． |
| Developer | 開発者を表す． |
| Directory | リポジトリ内のディレクトリを表す． |
| File | リポジトリ内のファイルを表す． |
| Issue | 開発者に作成された Issue を表す． |
| NewDeveloper | ある期間においての新規開発者を表す． |
| Pull Request | 開発者に作成された Pull Request を表す． |
| Repository | コミュニティ内のリポジトリを表す． |

表 4 関係の定義

Table 4 Definition of Relationships

| 関係名 | 定義 |
|---|---|
| ADDED | 開発者によるファイル追加を表す． |
| CHANGED | 開発者によるファイル移動を表す． |
| CHILD_OF | 親ノードと子ノードの関係を表す． |
| CLOSED | 開発者に Issue が閉じられたことを表す． |
| COMMITTED | 開発者による Commit の実行を表す． |
| CONTRIBUTED | リポジトリへの開発者の貢献を表す． |
| HAS | あるノードと他のノードの所属関係を表す． |
| IN | ある開発者が中核的または新規かの内包関係を表す． |
| MERGED | 開発者による Merge の実行を表す． |
| MODIFIED | 開発者によるファイル修正を表す． |
| OF | あるノードと他のノードの所属関係を表す． |
| REMOVED | 開発者によるファイル削除を表す． |
| RENAMED | 開発者によるファイルの名前変更を表す． |
| WROTE | 開発者による Issue, Pull Request, Commit, Comment の作成を表す． |

表 5 プロパティの定義

Table 5 Definition of Properties

| プロパティ名 | 定義 |
|---|---|
| additions | コミットによってコードが追加された行数 |
| comment | コメントの内容 |
| contributions | 貢献の数 |
| date | 日時 |
| deletions | コミットによってコードが削除された行数 |
| from_date | 対象期間の始まりの日時 |
| full_name | リポジトリと，その所有者名を合わせた文字列 |
| message | コミットのメッセージ |
| name | あるノードが持つ名前 |
| number | GitHub 上で付与される番号であり，同一リポジトリ内で一意となっている． |
| owner | リポジトリの所有者名 |
| path | ファイル及びディレクトリのパス |
| repository | リポジトリ名 |
| sha | コミットの識別番号 |
| title | Issue 及び Pull Request のタイトル |
| to_date | 対象期間の終わりの日時 |
| total | additions, deletions の合計 |
| uid | 同じラベルを持つノードを一意に識別するための ID |
| which | Issue 及び Pull Request のどちらの Comment かを識別する |