修 士 論 文

題　目　　巨大ソフトウェアリポジトリ群への
マイニングに基づくソフトウェア開発者間の
ネットワーク分析

主任指導教員　　辻野　嘉宏　教授

指導教員　　水野　修　准教授

京都工芸繊維大学大学院 工芸科学研究科

情報工学専攻

学生番号　　11622004

氏　　名　　出原　真人

平成25年2月8日提出

様式 2 号

学位論文の要旨（和文）

平成 25 年 2 月 8 日

京都工芸繊維大学大学院
工芸科学研究科長　殿

工芸科学研究科　　情報工学専攻
平成 23 年入学
学生番号　　11622004
氏　　名　　出原　真人　㊞

（主任指導教員　　辻野　嘉宏　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

巨大ソフトウェアリポジトリ群へのマイニングに基づくソフトウェア開発者間のネットワーク分析

2.　論文内容の要旨（400 字程度）

本論文では，複数のソフトウェアプロジェクトのデータから開発者ネットワークを構築し，開発者ネットワーク内で活動的な開発者の特徴を分析した．分析のために，一般に公開されている 8391 個のオープンソースソフトウェアのリポジトリを収集し，データベース化を行う．

我々は巨大なリポジトリ群に存在する開発者の間には大きな 1 つの開発者ネットワークが構築されるという仮説を立てた．この仮説のために，複数プロジェクトに関わる開発者を媒介にして多数の開発者が 1 つのネットワークに存在するかどうかを確認した．分析の結果，96%の開発者が属する巨大なネットワークが確認された．また，そのネットワークにおける開発者間の距離を開発者ホップとして定義し，開発者ネットワークに一定のスモールワールド性が存在することを示す．また，開発者ネットワーク上に存在する中心人物を推定し，実際に誰が中心となっているかを調べる．また，求めた開発者ホップを用いて任意の 2 人の開発者間の開発者ホップを計算する Web サービスを構築する．同時に，構築した開発者ネットワークを分析することで，直接共同開発者の数が開発者のアクティビティの高低を示す一定の指標となることを確認する．

Analyzing Developer Networks Through Mining Huge Software Repositories

2013 11622004 *Mahito IDEHARA*

Abstract

In this thesis, we analyse the characteristics of active developers in the open source software developments by constructing developer-relation networks from a large number of software repositories. To do so, we mined 8391 open repositories in public and made a database for the developer relations.

We first assumed that there is a dominating network among the developers of open source software. We found this hypothesis is true from the analysis of database since there is a large network including 96% of developers. We also defined the developer hop between two developers, and calculated it for any pairs of developers. Using this hops, we investigate who is in the center of this large network. At the same time, we analyzed active developers in this network who joined in more than two projects. The results show that the number of co-developers correlates the activeness of the developers.

目　次

1. 緒言

ソフトウェアプロジェクトの特徴を分析する手段としてリポジトリマイニングがある．リポジトリとはバージョン管理システムを用いてソフトウェアを開発する際に，過去から現在までの履歴を残しておく記録庫のようなものである．この記録を分析し，ソフトウェアの開発に関連する有効な知見を発見することがリポジトリマイニングと呼ばれる研究分野である．

これらは主にリポジトリに含まれるソフトウェアのメタデータから得られる情報に着目したものであるが，それとは別に，ソースコードの開発者に着目した研究も存在する．例として，そのソースコードに関わった開発者が多いほどそのコードは欠陥を含みやすくなるか否か，といったものが挙げられる．

開発者に着目する研究の一環として，そのプロジェクトに関わる開発者たちが構成する構造に着目するものが存在する．例えば，単一のオープンソースソフトウェアプロジェクトを分析したとき，その内部にはコミュニケーション構造などの形でソーシャルネットワーク構造が存在することが判明している．

オープンソースソフトウェア開発者の中には，プロジェクトを渡り歩いて発展させていくアクティビティの高い開発者が存在していることが分かっている．彼らの存在を追跡し，その特徴を分析するのを最終目標に定め，本研究ではその第一歩として複数のプロジェクトから開発者ネットワークを構築し，その中でアクティビティの高い開発者が持つ特徴を分析することを目的としている．

また，前提としてリポジトリマイニングを行うためには分析対象となるソフトウェアプロジェクトのリポジトリを用意する必要がある．その都度目的にあったプロジェクトを探してきてそれぞれに実験を行うのが主流であるが，最近ではリポジトリを集めた大規模なデータベースを作成し，その群に対して実験を行う研究も出てきている．

上記のような事情を踏まえ，本研究では以下に示す課題を達成するために実験を行った．

- ソフトウェアプロジェクトを収集しマイニング用の巨大リポジトリ群を作成する．
- 複数のプロジェクトから一つの開発者ネットワークを構築することは可能か調

査する．

- 可能であった場合，その中でアクティビティの高い開発者の特徴を調査する．

最後に、本論文の以降の構成を示す．第 2 章では，開発者ネットワークの分析手法と関連研究について詳しく説明する．第 3 章では，本研究の研究設問について説明する．第 4 章では，実験の概要について詳しく説明する．第 5 章では，実験の結果について述べる．第 6 章では，本実験の妥当性，及び結果に対する考察，および，今後の課題について述べる．第 7 章では，本研究のまとめについて述べる．

2. 開発者ネットワークの分析

この章では，リポジトリ群の必要性や開発者ネットワークについて説明する．

2.1 ソフトウェアメトリクス

ソフトウェアメトリクスは，1970 年代後半頃からソフトウェアの特性を測るための指標として提唱され始めた概念である．具体的には，ソースコードの規模，複雑さ，保守性などの「属性」の情報を定量的に示すことができる指標である．具体的な例としては，プログラムのステップ数やバグの数がこれに該当する．ソフトウェアメトリクスの多くはソースコードやプロジェクトの開発履歴から測定できる．そのため，バージョン管理システムを導入して開発されたオープンソースソフトウェアはメトリクスを用いた適用実験が行いやすいという利点がある．ソフトウェアメトリクスの用途としては，ソフトウェアの品質の測定や，誤り傾向の強いモジュールを判別するなど様々な方法が研究されている．

本論文では，ソフトウェアの開発者に関わるメトリクスを中心に収集し，開発者ネットワークの分析を行う．

2.2 リポジトリ群

提案したソフトウェアメトリクスやモデル式を評価し，一般化するためにはできるだけ多くのプロジェクトに対して提案手法の適用実験を行うのが望ましい．実験対象のプロジェクトの入手先としては，SourceForge や GitHub など，オープンソースソフトウェアのリポジトリが集合的に公開されているポータル（Forge サイトと呼ぶ）が多数存在しており，それらの Forge サイトを利用するのが一般的である．

最近では，実験対象に適したプロジェクトを集めてデータベースを作成する動きも出てきており，本研究も適用実験に適していると考えられるプロジェクトを集めてデータベース化する事を目的の一つとしている．

2.3 Git と GitHub リポジトリ

Git は分散型のバージョン管理システムの一つである．コマンド一つでサーバ上から対象のプロジェクトのリポジトリをローカルにコピーすることができ，過去のソースコードを確認するのにサーバを参照する必要が無いという利点がある．また，ファイルの移動や名前変更に強く，それらの変更が行われてもある程度はファイルの更新履歴を遡って追跡することができる．これらの利便性から本研究では Git でバージョン管理されたオープンソースソフトウェアのプロジェクトを分析対象とする．

GitHub は Git のリポジトリを中心に扱っている有名な Forge サイトの一つであり，個人的なプロジェクトから有名なプロジェクトまで数多くのプロジェクトを保有している利用者の多い開発コミュニティである．

2.4 開発者ネットワーク

本研究では集めたデータからプロジェクトの垣根を越えた開発者間の繋がりを分析する．分析する手段として開発者ネットワークを用いる．開発者ネットワークとは，開発者をノードとし，同一のファイルに対して更新を行った他の開発者を共同開発者としてその関係をエッジで表すネットワークである．

例として，図 2.1 のようなファイル更新関係を持つプロジェクト A とプロジェクト B を考える．このようなファイル更新関係を持つ場合，Alice と Bob は FileC について共同開発者であり，Alice と Charlie は FileD について共同開発者であるが，Bob と Charlie は共同開発者ではない．従って，これらのプロジェクトから生成される開発者ネットワークは図 2.2 のようになる．開発者ネットワーク上において，Bob と Charlie の間には直接の共同開発関係は存在していないが，Alice を通して間接的に繋がっていることが確認できる．

スモールワールド現象

スモールワールド現象とは，知り合いの知り合いを辿っていけば比較的少ない数で世界中の誰とでも繋がることができる，という仮説である．1962 年に Stanley Milgram が行った検証実験では，アメリカ合衆国内から無作為に選ばれた 2 人は平均

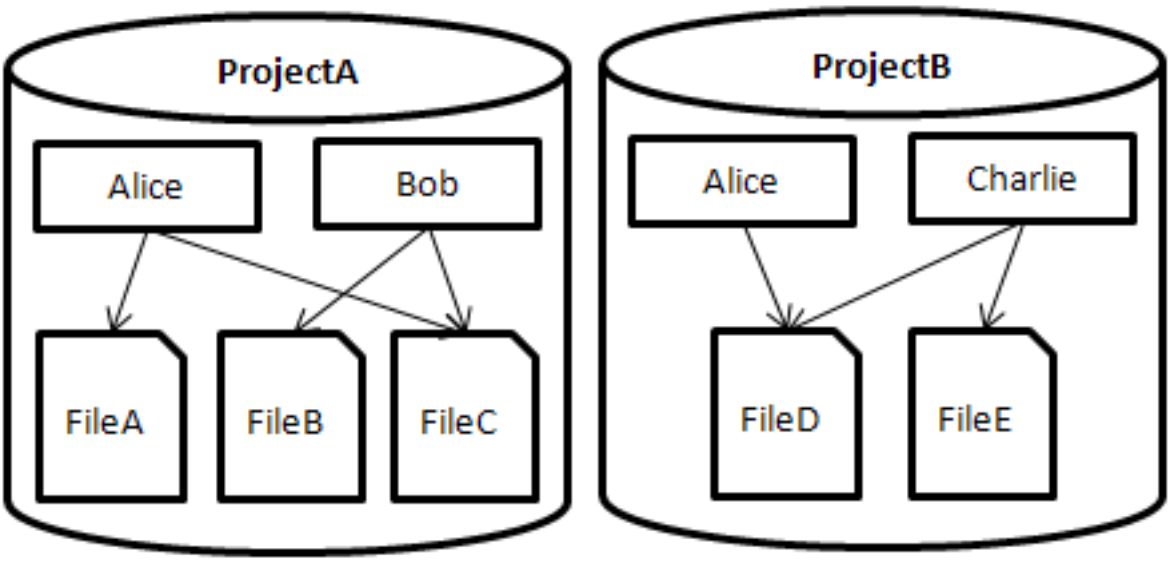

図 2.1　ファイルの更新関係の例

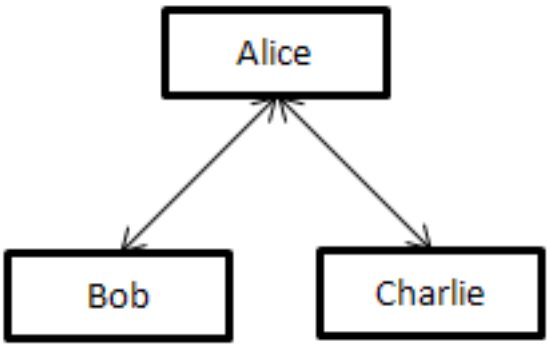

図 2.2　図 2.1 から構築される開発者ネットワーク

して6人の知り合いを通して繋がっているという結果が得られ，このことから「六次の隔たり」という言葉が生まれた [1, 2].

本研究では，収集したプロジェクトのリポジトリを分析してプロジェクトの垣根を越えた開発者ネットワークを構築し，それが独立した小さな世界の集まりになるのか，どこかで繋がった1つの世界となるのかを調査する．

Erdős Number

エルデシュ数（Erdős Number）とは，ハンガリーの数学者 Paul Erdős との近さを論文の共著関係によって表す指標である．論文の著者をノード，共著関係をエッジとした協力グラフにおいて，著者間を結ぶ最短パスの長さを二人の間の距離と定義する．このとき，協力グラフにおけるエルデシュとの距離をエルデシュ数と定義する．エルデシュ自身のエルデシュ数は 0 とする．科学者たちの間では小さいエルデシュ数を持つ事が名誉であると冗談のように言われている．

Developer Hop と Torvalds Number

本研究では，構築した開発者ネットワーク上における開発者間の距離を開発者ホップ（Developer Hop）と定義する．開発車間の距離は最短パスの長さで定義し，起点となる開発者の開発者ホップは 0 とする．

例えば，図 2.2 の開発者ネットワークにおいて Bob を起点とした開発者ホップを考える場合，Bob の開発者ホップは 0，Alice の開発者ホップは 1 となり，Charlie の開発者ホップは 2 となる．

また，エルデシュ数に習い，開発者ネットワークにおいて Linux の生みの親であり Git の開発者でもある Linus Torvalds を起点として各開発者に与えられる開発者ホップをトーバルズ数（Torvalds Number）と定義する．

2.5 関連研究

適用実験の対象とする為に，大量のプロジェクトのデータを収集する研究が行われている．Mockus は多くの Forge サイトからプロジェクトのリポジトリを収集し，そのデータを分析して様々なインデックス付けを行った [3]．Gousios らは GitHub の

タイムライン上に現れる全てのプロジェクトを分散した拠点から可能な限り収集し，Torrent で共有する GHTorrrent というシステムを提案した [4].

単一のリポジトリから何らかのネットワークを構築して調査を行う研究には次のようなものがある．Menely らは「充分な人数がいれば全てのバグはすぐさま発見され修正される」というリーナスの法則が正しいのかどうか調査する目的で実証研究を行った [5]．その後，開発者の活動性のメトリクスを提案し，ファイルに関わる人数による欠陥の出現率と除去される割合を調べ,「多すぎる人数が製作物を駄目にする」という説と「充分な人数がいれば全てのバグはすぐさま発見され修正される」という説に対して調査を行った [6]．他に，一ヶ月以内に同じファイルに更新した開発者を共同開発者と定義し，各開発者にアンケートを取って，実際に開発者が共同作業をしていると感じている開発者とリポジトリを解析して共同開発者と判断された開発者はどの程度共通しているかを調査も行っている．これによりソフトウェア工学にソーシャルネットワーク解析メトリクスを用いることの有効性を評価した [7]．Bird らはメーリングリストの内容やプロジェクトのリポジトリから分析を行い，オープンソースプロジェクトが無秩序なバザーのような構造をしているのかを調査した．社会参加者と開発振る舞いの関係を特徴づけて，大規模なオープンソースプロジェクトに存在する社会構造を検査した結果，存在する構造は典型的なバザー構造ではなく有機的にチームへ組織していく傾向にある事を確かめた [8]．その後，大規模で複雑且つ成功したオープンソースプロジェクトでは開発チーム内に小さな社会が自然発生していると考え，オープンソースプロジェクトが自己組織化していく過程を調査した [9]．また，e-mail アドレスをもとにソーシャルネットワークを作る試みも行っている [10]．この論文の提案をもとに，Navarro は同一の開発者を判定するための名寄せのアルゴリズムを実装した [11]．Kagdi らはソフトウェアの依存関係をネットワークとして解析し，現在のバージョンから予測される未来とプロジェクトの履歴から予測される未来を組み合わせてソフトウェアの変更を予測する方法の調査と開発の必要性を主張した [12]．Rostal らはソフトウェア開発プロジェクトから構築できるネットワークに存在する弱いリンクの役割について調査し，単純なシステムと複雑なシステムの持つリンクの違いについて調査した [13].

開発者ネットワークを用いてソフトウェアの欠陥を予測する方法が研究されている．Meneely らは Code Churn メトリクス [14] を情報に持った開発者ネットワークを

構築し，ネットワークの繋がりとソフトウェアメトリクスからソフトウェアの欠陥予測を行った．最近行われた変更による変化量が大きいほど，ソフトウェアに欠陥が混入されやすくなるが，その変更が多くの他の開発者と一緒に共同作業をしたことがある有名な開発者によるものであればどうなるかを調査した [15].

3. 研究設問

ここでは, 本実験で調査する研究設問（RQ:Research Questions）について説明する.

本研究は将来的に，プロジェクトを渡り歩き発展させていく開発者達を検出し軌跡を追う事や，開発者ネットワークに対して様々なソーシャルネットワーク解析的アプローチを行う事を目的としている.

これらの目的の為に本研究では，多数のプロジェクトを統合して開発者ネットワークを構築する事は出来るのか，構築されたネットワークはソーシャルネットワーク的性質を持っているのか，活動的な開発者の特徴を抽出する事は出来るのか等について調査を行う.

従って，本研究では以下に設定する研究設問について調査を行う.

RQ1 どれぐらいの割合の開発者が大きな一つの開発者ネットワークに所属しているか？

RQ2 開発者は最大でいくつのプロジェクトに関わっているか？平均, 分布は？

RQ3 開発者ホップ, トーバルズ数はいくらぐらいか？

RQ4 高いアクティビティを持つ開発者にはどのような特徴があるか？

4. 実験概要

4.1 リポジトリの収集

本研究ではネットワークを構築するという目的から，収集するプロジェクトのリポジトリは多くの人が関わっているのが望ましく，有名なプロジェクトである方が良いと判断した．また，分析対象であるリポジトリはファイルの移動や名前変更に強い Git で管理されたものが良いと考えた．従って，本研究ではリポジトリの大部分を Git の有名な Forge サイトである GitHub から収集した．

収集手順

最初に，多くの人が関わっていそうなプロジェクトを判別する為に Google BigQuery を利用した．Google BigQuery は Google の提供する企業向けのビッグデータ解析サービスである．サンプルのデータセットとして GitHub のアーカイブが含まれているので，本研究ではそのデータセットを利用した．GitHub 上でウォッチャーが多いプロジェクトほど多くの人が注目している，つまり有名で多くの人が関わっていると考え，次のような Query を実行した．

```
SELECT repository_name, watchers, repository_language, repository_description,
repository_url FROM (SELECT repository_name, count(repository_name) as watchers,
repository_language, repository_description, repository_url
FROM [githubarchive:github.timeline] WHERE type="WatchEvent"
GROUP BY repository_name, repository_language, repository_description, repository_url
ORDER BY watchers DESC) as watched WHERE watchers >=100
```

これにより，ウォッチャーが 100 人以上のプロジェクトの，リポジトリ名，ウォッチャーの数，主な開発言語，説明，リポジトリの URL が分かる．ここで，次のコマンドを実行する事でプロジェクトのリポジトリを収集した．

```
git clone リポジトリの URL
```

これにより，既に存在しないいくつかのプロジェクトを除いた 8000 件強のプロジェクトのリポジトリを収集する事が出来た．

他に，git.eclipse.org からもリポジトリを収集した．

4.2 開発者ネットワークの構築

収集したプロジェクトのリポジトリから必要な情報をデータベースに登録し，開発者ネットワークを構築する手順を説明する．

1. リポジトリに含まれる全てのコミットのデータを取得する
   最初に，リポジトリに含まれる開発者や変更されたファイル等，コミットに関連した情報を抜き出す必要がある．

   ```
   git --git-dir=$INPUT/.git log -p --pretty=full --all
   ```

   $INPUT はリポジトリの存在するパスとする．このコマンドによってリポジトリに記録された全てのコミットのデータを抜き出す事が出来る．
2. 開発者，ファイル情報を登録する
   抜き出されたデータに存在する開発者に開発者 ID を振って登録する．この際，名前と e-mail アドレスで同一人物判定を行い，同一人物であれば既に登録されている開発者 ID を使用する．
   さらに，ファイル ID，同一ファイル ID，コミットした開発者の ID，プロジェクト ID，ファイル名などのファイル情報を登録する．Git は名前の変更をある程度追い掛けることが出来るため，ファイル名のほかに同一ファイル ID を付与して，後の処理に使用している．
3. 上記の手順を取得した全てのプロジェクトのリポジトリに対して実行し，全ての開発者とファイルを登録する．
4. 共同開発者情報を登録する
   登録されたファイル情報を元に，同一ファイル ID が等しいファイルに対してコミットを行っている開発者を共同開発者とし，開発者 ID を組にして登録する．

以上の手順を実行することで，開発者をノード，共同開発者情報をエッジとした開発者ネットワークが構築出来る．

データベースのスキーマ

上記の手順で抜き出された各種データは図 4.1 に示されるようなスキーマを持つデータベースに保存されている．テーブルと属性はそれぞれ次のような意味を持っている．

- authors

  開発者の名前や e-mail アドレス等の情報が保存されているテーブル．

  - id　開発者 ID．開発者ごとに一意の値が振られている．
  - name　開発者の名前．
  - email　開発者の e-mail アドレス．

- projects

  プロジェクト名等のプロジェクトの情報が保存されているテーブル．

  - id　プロジェクト ID．プロジェクトごとに一意の値が振られている．
  - name　プロジェクト名．

- files

  ファイル名やその変更者等，ファイルの情報が保存されているテーブル．

  - id　ファイル ID．ファイルの各コミットごとに一意の値が振られている．今のところ使用しない．
  - project　プロジェクト ID．ファイルがどのプロジェクトに属するかを示す．
  - author　開発者 ID．ファイルがだれによって更新されたかを示す．
  - name　ファイル名．パスを含んだファイル名の情報が保存されている．
  - identical　同一ファイル ID．files テーブルのなかで同じファイルを見分けるのに使用する．

- coauthors

  共同開発者の情報が保存されているテーブル．

  - author　開発者 ID．
  - coauthor　共同開発者の開発者 ID．
  - identical　同一ファイル ID．この ID のファイルを通して author と coauthor が繋がっている．

- sameauthors

  同一開発者のリストを保存したテーブル．大本になる開発者のデータは登録されない．登録しようとした開発者が既に authors テーブルに登録されていた場合，その開発者の情報はこのテーブルに登録される．authors テーブルと組み

合わせることで，ある開発者の持つ全ての名前, 又はメールアドレスを調べることができる．

- id　同一開発者 ID.
- authorid　開発者 ID.
- name　名前.
- email　e-mail アドレス.
- project　プロジェクト ID.

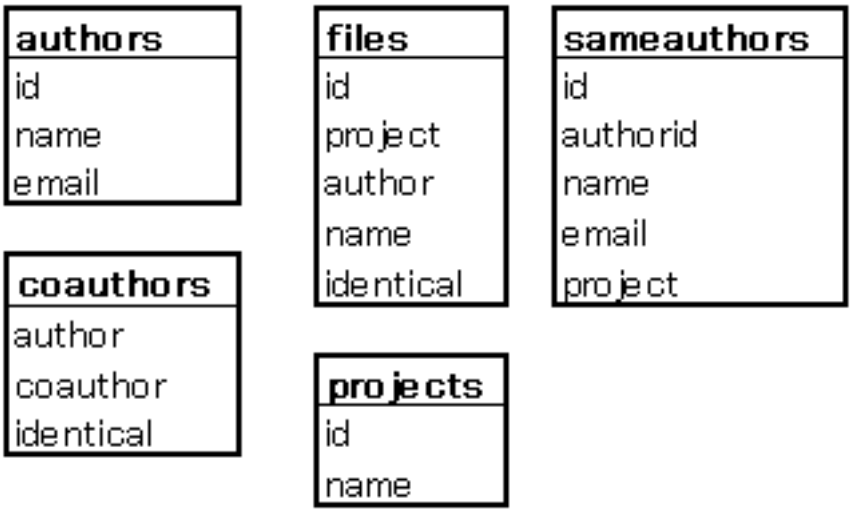

図 4.1　開発者ネットワークデータベースのスキーマ

5. 実験結果

5.1 リポジトリの収集とデータベースの作成

プロジェクトのリポジトリを収集した結果，表 5.1 に示すように GitHub から 8,140 件，git.eclipse.org から 251 件の計 8,391 のリポジトリを得ることが出来た．

これらのプロジェクトのリポジトリから，4.1 節で示した方法を用いてデータベースを作成した．

同一人物判定は名前が等しいか e-mail が等しければ同一人物であると判定する（名前が優先）単純な方法を採用しているが，何人かの開発者がデフォルトネーム，又はデフォルトメールアドレスを使用しているため，誤って同一人物と判定されてしまう問題が発見された．この問題に対処するために，名前として'root'，'unknown' を使用している，又は e-mail として'none@none' を使用している人物は例外として同一人物と判定しない事とした．

構築されたデータベースを簡単に調査した結果，表 5.2 に示すように，総開発者数 94,786 人，総ファイル数は約 1,000 万，総更新数は約 1 億回にも昇った．また，各開発者が参加していたプロジェクト数の分布は図 5.1 のようになった．単一のプロジェクトにしか参加していなかった開発者は，58,191 人存在した．これは今回調査した開発者の 61.39 % にあたる．この事は，オープンソースソフトウェア開発者のおよそ 4 割が 2 つ以上のプロジェクトに関わっている事を示している．

さらに，表 5.3 にもう少し詳しいデータベースの統計値を示しておく．

5.2 開発者ホップの計算

作成したデータベースの coauthors テーブルには図 5.2 に示すような開発者関係グラフのエッジに対応するデータが格納されている．それらのデータから開発者ネットワークを構築し, ダイクストラのアルゴリズムを用いて最短経路を計算することで図 5.3 に示すような開発者ホップグラフが作成できる．

エッジの情報を表 5.4 に示す．これらのエッジを基に開発者ネットワークを構築して解析したところ，いくつかの独立した開発者ネットワークが確認できた．その情報を表 5.5 に示す．開発者ネットワークの規模はその開発者ネットワークが含むノー

表 5.1　データの収集元

	件数	時期
GitHub	8,140	November 2012
git.eclipse.org	251	June 2012

表 5.2　リポジトリ概要

総プロジェクト数	8,391
総開発者数	94,786
総ファイル数	10,765,798
総更新数	95,855,501

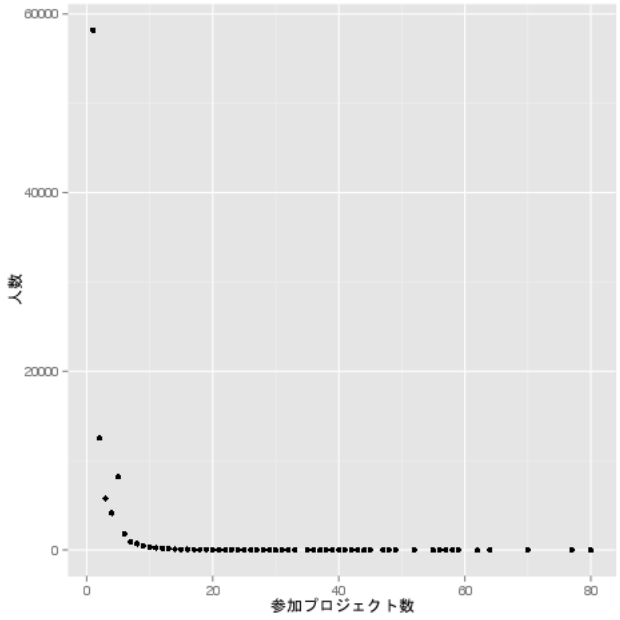

図 5.1　開発者の参加プロジェクト数

表 5.3　データセット詳細

	最小値	中央値	平均値	最大値
プロジェクト毎の開発者数	1	6	24.79	9,174
プロジェクト毎のファイル数	1	95	1,286.10	380,164
プロジェクト毎の更新数	1	659	11,451.00	8,604,750
ファイル毎の更新数	1	2	8.90	111,232
ファイル毎の開発者数	1	1	2.00	704
開発者毎のプロジェクト数	1	1	2.21	80
開発者，プロジェクト毎のファイル数	1	3	103.70	167,057
開発者，ファイル毎の更新数	1	2	4.46	17,322

ドの数とする．これを見ると，一つの大きな開発者ネットワークが確認できる．規模が 91,298 ということは，今回調査した開発者の 96.32 %が一つの開発者ネットワーク上で繋がっているという事になる．ここで，この一番大きい開発者ネットワークを対象にして，開発者ホップを求めることにする．

開発者ホップは，前述の通りにダイクストラのアルゴリズムを用いて求める．このとき，エッジの重みは考慮せずそれぞれのエッジが等しい重さを持つこととする．各ノードを起点にしてそれぞれ開発者ホップを求めたところ，結果は表 5.6 のようになった．最大開発者ホップとは，その人物を起点として開発者ホップを計算したとき，ネットワーク上に存在する最大ホップ，即ち起点から最も遠いところに存在する開発者が持っている開発者ホップと定義する．最大開発者ホップの最小は 7，最大でも 13 で，平均して 9 人を辿れば同一の開発者ネットワーク上に存在するどの開発者とでも繋がることができるのが確認できる．

また，自分と直接繋がっている開発者（つまり，開発者ホップが 1 の開発者）を直接共同開発者と定義したとき，各開発者が持つ直接共同開発者数の分布を図 5.4 に示す．多くの開発者が 100 人以下の開発者と直接つながっており，一部の開発者がハブとなって多くの他の開発者と繋がっているのが確認できる．

本研究では，得られた開発者ホップのデータを元に，Web 上で任意の 2 人の開発者間のホップ数を求めることが出来るサービスを公開している (注 1)（図 5.5）．

5.3 ネットワークの中心人物とトーバルズ数

ここで，この開発者ネットワークの中心にいる人物と最も外れにいる人物を調べることにする．最大開発者ホップが最も小さく直接共同開発者の人数が最も多い人物をネットワークの中心人物，最大開発者ホップが最も大きく直接共同開発者の人数が最も少ない人物をネットワークの最も外れにいる人物と考える．調査を行った結果，ネットワークの最も外れにいた人物は最大開発者ホップ 13，直接共同開発者を 1 人持つ polyethene という人物であり，最も中心にいた人物は最大開発者ホップ 7，直接共同開発者を 6,677 人持つ Linus Torvalds であるという結果が得られた．Linus は Linux や Git の開発者である有名な人物であり，技術者なら誰もが知っている開発

(注 1)：http://se.is.kit.ac.jp/devhop/

表 5.4　ネットワークノードとエッジ

ノード数	94,786
開発者関係グラフエッジ数	74,243,222
開発者ネットワークグラフエッジ数	3,844,055

表 5.5　開発者ネットワークの規模

ネットワーク規模	個数	ネットワーク規模	個数
91298	1	12	1
229	1	10	6
74	1	9	3
44	1	8	4
34	1	7	8
32	1	6	4
26	1	5	18
21	1	4	39
18	2	3	79
17	1	2	205
15	2	1	1,815
13	2		

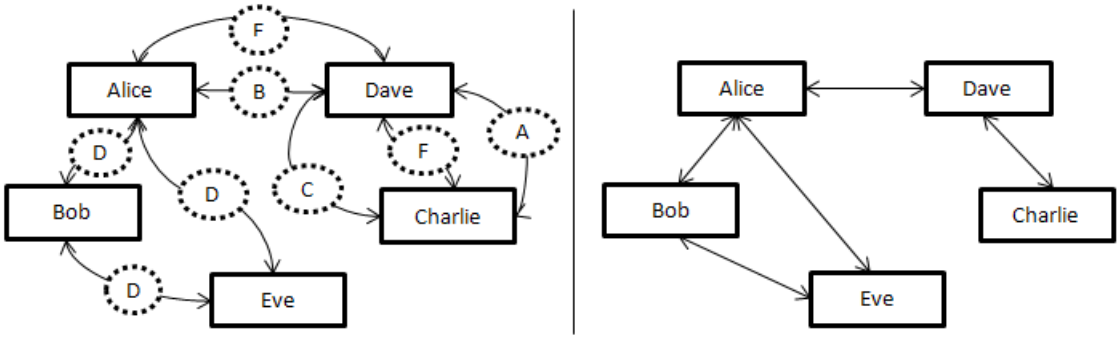

図 5.2　開発者関係グラフ (左) と開発者ネットワークグラフ (右)

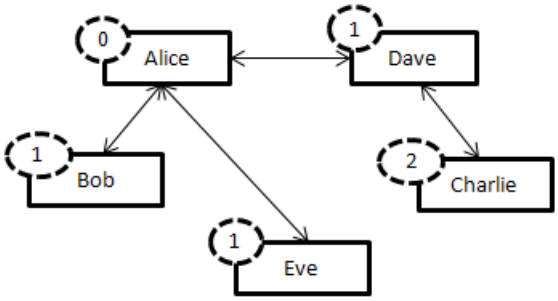

図 5.3　Alice を中心にした開発者ホップグラフ

表 5.6　開発者ホップグラフの詳細

	最小値	中央値	平均値	最大値
最大開発者ホップ	7	9	8.72	13
トーバルズ数	0	3	2.61	7
直接共同開発者数	1	20	83.79	6,677

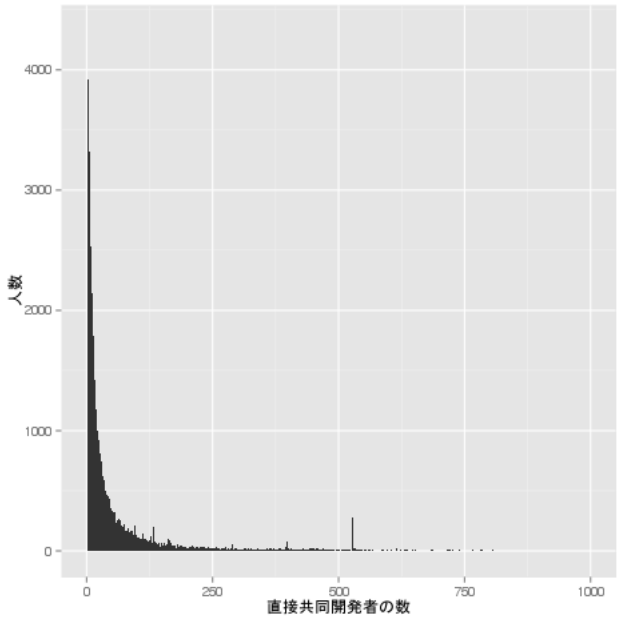

図 5.4　直接共同開発者数の分布

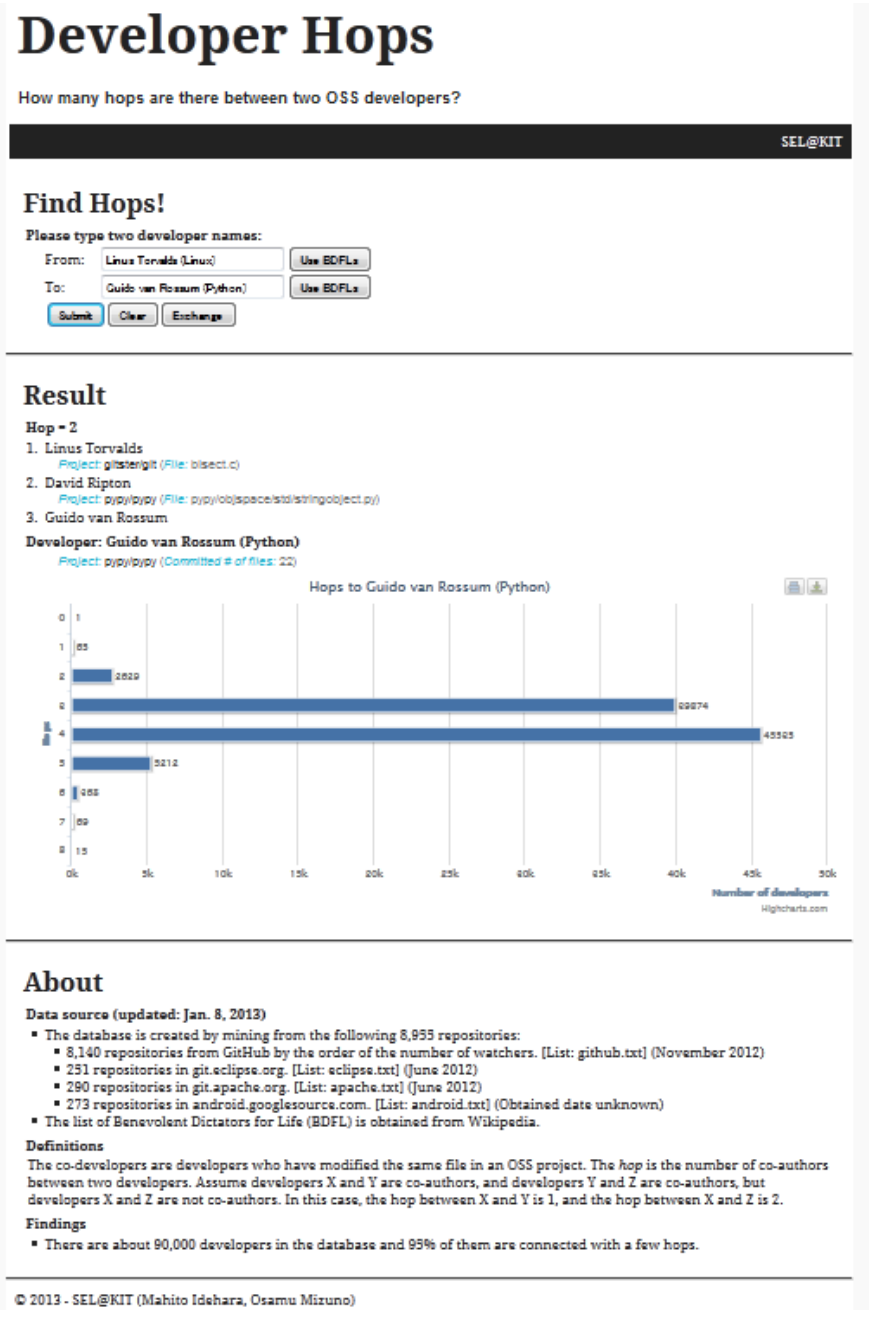

図 5.5　開発者ホップ計算サービス

者が実際に開発者ネットワークの中心に存在するという興味深い結果が得られたことになる．

これら 2 人の人物を起点として得られた開発者ホップグラフを，グラフ描画ツールの UbiGraph (注 2)を用いて描画したものが図 5.6 と図 5.7 である．エッジの色は開発者ホップ数で決められている．どちらの図も中心に本人がいるものであるが，polyethene のグラフは外に向かうつれてどんどん広がっていき，Linus のグラフは中心に大部分が集まっているのが見て取れる．また，どちらのグラフもところどころにクラスタが形成されているのが確認できる．

エルデシュ数に倣い，Linus を起点とした開発者ホップを考えた時，各開発者に割り当てられた開発者ホップ数のことをトーバルズ数と定義する．その統計は表 5.6 のようになる．これを見るとどの開発者も平均 3 人の協力関係を辿れば Linus に辿りつくことが出来ることが分かる．

5.4 開発者のアクティビティとその特徴

ここでは開発者の活動性を調査する．そのために，開発者の活動性をそのプロジェクトでの関わったファイル数や更新を行った数として参加プロジェクト数や開発者ホップの値との関連性を確かめる．

実験対象のプロジェクトとして，収集したプロジェクトから次の 5 つのプロジェクトを選んだ．

- mozilla/mozzila-central：
  Mozilla の提供する Firefox などの製品のプロジェクト
- torvalds/linux：
  Linus Torvalds が主導する Linux のメインのプロジェクト
- gokhanmoral/siyahkernel3：
  GalaxyS2 などに適応することができる Android カーネルのプロジェクト
- rails/rails：
  Ruby on Rails のプロジェクト
- mxcl/homebrew：

(注 2)：http://ubietylab.net/ubigraph/

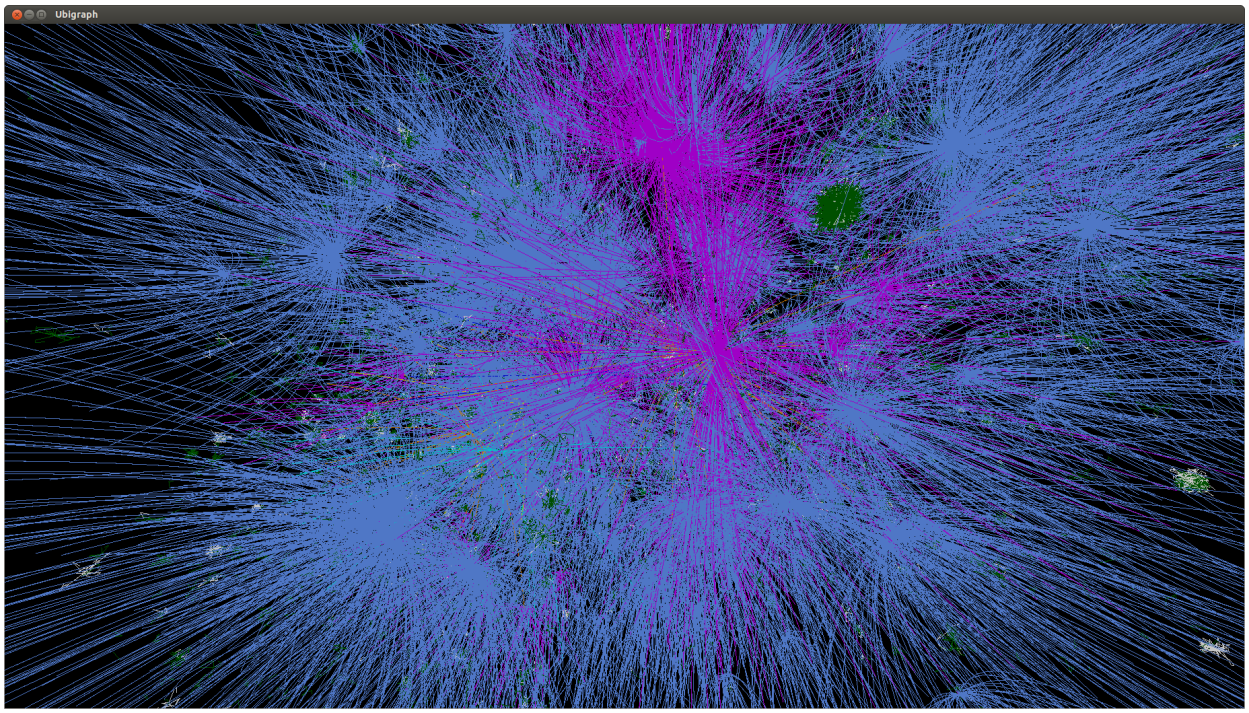

図 5.6　polyethene の開発者ホップグラフ

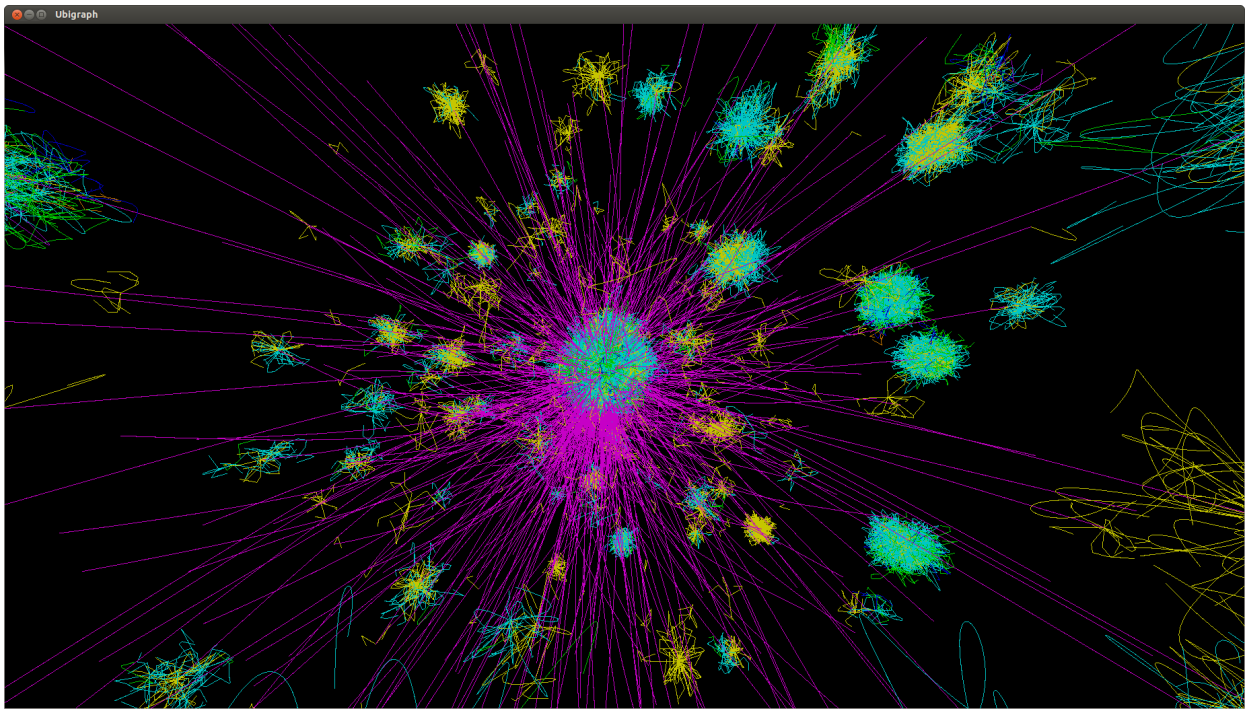

図 5.7　Linus の開発者ホップグラフ

OS X 用のパッケージマネージャである homebrew のプロジェクト

これらのプロジェクトを選んだ理由としては，関わっている開発者が多い，それなりのファイル数と更新数を持っている，規模が適度に分散しているなどがあげられる．

各プロジェクトの参加人数などの情報を表 5.7 に示す．これら 5 つのプロジェクトについて，参加している開発者の参加プロジェクト数，直接共同開発者数と開発者がそのプロジェクトで編集したファイルの総数，更新の総数，ファイル 1 個当たりの平均更新数との間のスピアマンの順位相関係数を統計計算ツールの R (注 3) を使用して求めた．その結果を表 5.8 に示す．これらの結果から，開発者の参加プロジェクト数は関わるファイルや更新回数と相関が無いが，直接共同開発者が多い，すなわち全体の開発者ネットワークにおいて中心部に近い開発者ほど一つのプロジェクトで関わるファイル数や更新数が多い傾向にあることが分かる．

さらに，ノイズを軽減する目的で 5 つのプロジェクトのうち 2 つ以上のプロジェクトに参加している開発者のみを残して再度相関関係を調査した．その結果を表 5.9 に示す．一部のプロジェクトを除いて相関係数が上昇しているのが確認できる．相関係数が上昇したところを太字で示す．

(注 3)：http://cran.r-project.org/

表 5.7　実験対象のプロジェクト

プロジェクト名	参加人数	総ファイル数	総更新数
mozilla/mozilla-central	2,389	198,057	2,809,593
torvalds/linux	9,059	70,327	1,616,614
gokhanmoral/siyahkernel3	7,676	61,855	1,269,508
rails/rails	1,943	9,373	187,806
mxcl/homebrew	2,619	2,802	44,248

表 5.8　各プロジェクトの相関係数 (1)

プロジェクト名		更新ファイル数	総更新数	平均更新数
mozilla/mozilla-central	参加プロジェクト数	0.014	0.027	0.070
	直接共同開発者数	0.428	0.568	0.228
torvalds/linux	参加プロジェクト数	0.072	0.085	0.015
	直接共同開発者数	0.613	0.686	0.201
gokhanmoral/siyahkernel3	参加プロジェクト数	0.055	0.069	-0.001
	直接共同開発者数	0.576	0.678	0.200
rails/rails	参加プロジェクト数	0.262	0.223	0.063
	直接共同開発者数	0.605	0.515	0.174
mxcl/homebrew	参加プロジェクト数	0.057	0.059	0.062
	直接共同開発者数	0.391	0.364	0.089

表 5.9　各プロジェクトの相関係数 (2)

プロジェクト名		更新ファイル数	総更新数	平均更新数
mozilla/mozilla-central	参加プロジェクト数	**0.170**	**0.236**	**0.339**
	直接共同開発者数	**0.660**	**0.659**	0.103
torvalds/linux	参加プロジェクト数	0.064	0.072	-0.006
	直接共同開発者数	**0.616**	**0.689**	0.197
gokhanmoral/siyahkernel3	参加プロジェクト数	**0.065**	**0.077**	**-0.003**
	直接共同開発者数	**0.579**	**0.679**	**0.201**
rails/rails	参加プロジェクト数	**0.536**	**0.485**	**0.198**
	直接共同開発者数	**0.684**	**0.675**	**0.395**
mxcl/homebrew	参加プロジェクト数	**0.227**	**0.239**	**0.161**
	直接共同開発者数	0.162	0.171	**0.134**

6. 考察

6.1 妥当性の検証

最初に，本実験で得られた結果の妥当性について検証を行う．

データセットの不備：本実験はオープンソースソフトウェアを分析対象にしているが，オープンソースソフトウェアのプロジェクトでは商業のプロジェクトよりも完璧な入力を期待することができない．例として，名前や e-mail アドレスの誤入力や初期値の使用，コミットメッセージの不完全等の問題が発生することなどが挙げられる．特に，同一の開発者を判定する必要がある今回の実験では名前と e-mail アドレスに関係するミスが分析結果に影響を及ぼしている可能性がある．

プログラムの不備：本実験では，リポジトリの収集，データの抽出，開発者ネットワークの構築，開発者ホップの計測，結果の集計などの処理を作成したプログラムによって行っているが，これらのプログラムに不備があった場合，実験結果に影響を与えている可能性がある．

名寄せの不備：本研究では，開発者の同一性を判断するのに単純なアルゴリズムを使用しているが，開発者の誤入力により同一と判断されない場合が存在し，さらに同姓同名によって別々の開発者が同一人物と判断されている場合がある可能性がある．これらの誤差が開発者ネットワークの繋がりに影響を与えている可能性がある．

6.2 研究設問の考察

本研究で設定していた研究設問について考察を行う．

RQ1 どれぐらいの割合の開発者が大きな一つの開発者ネットワークに所属しているか？

単一のプロジェクトからネットワークを構築した研究が過去に存在するが，複数のプロジェクトの開発者情報を頼りにネットワークの構築を試みた場合，大きな塊になるのか多数の小さな塊になってしまうのかを調査する目的でこの設問を設定した．

結果を考察するために表 5.3，及びに表 5.5 を参考にする．5.2 節で述べたとおり，今回調査したリポジトリ群に含まれる開発者 94,786 人のうち 91,298 人（96.32 %）が

一固まりの開発者ネットワークとして繋がっていることが確認できている．さらに，1 プロジェクトに存在する開発者の人数は最大でも 9174 人，平均 24 人，中央値で高々6 人であるのでこれだけの人が繋がるためにはプロジェクトを渡り歩きハブとしての役割を人物が存在することが考えられる．

RQ2 開発者は最大でいくつのプロジェクトに関わっているか？平均, 分布は？

多くのプロジェクトに参加しているほど活発な開発者であると考え，そのような開発者がどの程度存在するのかを調査する目的でこの設問を設定した．

結果を考察するために表 5.3，及びに図 5.1 を参考にする．5.1 節で述べたとおり，単一のプロジェクトにしか参加していなかった開発者は 58,191 人存在した．これは今回調査した開発者の 61.39 %にあたる．このことから，オープンソースソフトウェアに参加している開発者のうち 2 個以上のプロジェクトに参加しているのは全体の 4 割程度しかいないことがわかった．平均すると 2.21 プロジェクトになってしまうが，中には 40 以上のプロジェクトに参加しているようなベテランの開発者も一定数存在しているのが確認できている．また，今後対象となるプロジェクトを増やしていく過程で 2 個以上のプロジェクトに参加している開発者の値は大きく変動する可能性がある．

RQ3 開発者ホップ, トーバルズ数はいくらぐらいか？

将来的にソーシャルネットワーク解析技術からのアプローチを適用できるかどうかを判断するため，収集したデータから構築した開発者ネットワークがスモールワールド性をもつかどうかを調査する目的でこの設問を設定した．

結果を考察するために表 5.6 を参考にする．5.3 節で述べとおり，どの開発者も平均して 9 人を辿れば他の開発者と繋がることができ，相手をリーナスに限定した場合，平均 3 人の開発者を辿ればリーナスに辿りつけるという結果が確認できた．これは，今回構築した開発者ネットワークにスモールワールド性があることを示していると考えられる．

これらの値についてさらに考察するために，各数値間の相関係数 R を表 6.1 に示す．トーバルズ数と開発者ホップ間の相関が高く，リーナスからの距離が離れるほど開発者ネットワークの端のほうに近づいていくということが確認できる．また，

参加プロジェクト数と他の数値にはある程度の相関関係があるもののそれほど高くはないという結果が確認でき，これは適当にプロジェクトの参加数だけを増加させたとしても必ずしもネットワークの中心に近づくわけではないということを示していると考えられる．

RQ4 高いアクティビティを持つ開発者にはどのような特徴があるか？

将来的にプロジェクトを渡るアクティブな開発者の分析を行うことを目的として，アクティブ開発者の特徴を調査するためにこの設問を設定した．

結果を考察するために表 5.7，表 5.8 と表 5.9 を参考にする．各プロジェクトにおいて関わったファイルの数や総更新数が多いほどアクティブであると定義するとき，参加プロジェクト数はほとんど無相関であることが多く，相関があっても直接共同開発者数ほどの相関は無いことが読み取れる．homebrew のみ直接共同開発者数が低いのは，総ファイル数，総更新数が他のプロジェクトに対して少なく，十分なサンプルが得られていないためだと考えられる．ノイズ軽減策をとった表 5.9 ではさらにデータ数の軽減の影響を受けている可能性がある．他に，使用されているプログラミング言語の違いが影響しているのではないかと考えられる．

結論としては，高い直接共同開発者数を持つ開発者ほど多くのファイルに関わり多く更新する傾向が確認され，高いアクティビティを持っていると考えられる．

6.3 今後の課題

以上の考察を踏まえ，本節では今後の課題について考察する．

リポジトリの収集：開発者ネットワークを現実に即したものに近づけるために，さらなるリポジトリの収集が必要だと考えられる．

データセットの拡充：files テーブルにコミット日時を追加する．同じファイルに対して更新を行った開発者というのを共同開発者とした場合，長く続いたプロジェクトにおいては何年も前にコミットした人と共同開発者になるが，実際の繋がりはほとんどないということが起こりえる．コミット日時を追加することで，一カ月以内に同じファイルに更新を行った開発者，のように条件付けを行うことが可能になる．

言語別のネットワーク構築：使用言語によって相関度合いに違いがある可能性が確認されたため，使用言語別に開発者ネットワークを構築し，同様の実験を行って

表 6.1　測定値間の相関係数

	参加 プロジェクト数	直接共同 開発者数	トーバルズ数	最大開発者ホップ
参加プロジェクト数	-	0.308	-0.373	-0.364
直接共同開発者数	0.308	-	-0.388	-0.354
トーバルズ数	-0.373	-0.388	-	0.735
最大開発者ホップ	-0.364	-0.354	0.735	-

調査する必要があると考えられる．

名寄せ問題の改善：新たなアルゴリズムの採用を考えるとともに，手動で同一人物とされるような開発者をチェックし，少しずつ誤判別を減らしていく工夫が必要である．

特徴量の抽出：今回の結果をもとに，直接共同開発者数以外にも開発者のアクティビティと相関がありそうなメトリクスを提案し，調査を続けていく必要がある．

7. 結言

本研究では，複数のソフトウェアプロジェクトのデータから開発者ネットワークを構築し，その中で活動的な開発者の特徴を分析を行った．その過程において 8391 個のソフトウェアプロジェクトのリポジトリを収集し，データベース化を行った．

巨大なリポジトリ群から開発者ネットワークを構築した場合，大きな開発者ネットワークが存在しており，複数プロジェクトに存在する開発者を媒介にして多数のプロジェクトをつなぐことが出来ることが確認できた．

開発者ホップを計算することにより，開発者ネットワークに一定のスモールワールド性が存在することを示し，開発者ネットワーク上の中心人物を突き止めることができた．また，求めた開発者ホップを用いて任意の 2 人の開発者間の開発者ホップを提供する Web サービスを構築した．

構築した開発者ネットワークを分析し，直接共同開発者の数が開発者のアクティビティの一定の指標となることを示した．

今後の課題としては，名寄せ問題の解決や，さらなる開発者のアクティビティに関係した特徴量を分析することなどが考えられる．また，収集したリポジトリ群に対してトピック分析等の更なる分析を行っていくことを予定している．

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学部門水野修准教授に厚く御礼申し上げます．本報告書執筆にあたり貴重な助言を多数頂きました，本学情報工学専攻梁軍偉君，川本公章君，中井道君，椋代凛君，大西哲朗君，学生生活を通じて筆者の支えとなった家族や友人に深く感謝致します．

参考文献

[1] S. Milgram, “The small world problem,” Psychology today, vol.2, no.1, pp.60–67, 1967.

[2] J. Travers and S. Milgram, “An experimental study of the small world problem,” Sociometry, pp.425–443, 1969.

[3] A. Mockus, “Amassing and indexing a large sample of version control systems: Towards the census of public source code history,” Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, pp.11–20, MSR ’09, IEEE Computer Society, Washington, DC, USA, 2009.

[4] G. Gousios and D. Spinellis, “Ghtorrent: Github’s data from a firehose,” Mining Software Repositories (MSR), 2012 9th IEEE Working Conference onIEEE, pp.12–21 2012.

[5] A. Meneely and L. Williams, “Secure open source collaboration: an empirical study of linus’ law,” Proceedings of the 16th ACM conference on Computer and communications security, pp.453–462, CCS ’09, ACM, New York, NY, USA, 2009.

[6] A. Meneely and L. Williams, “Strengthening the empirical analysis of the relationship between linus’ law and software security,” Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp.9:1–9:10, ESEM ’10, ACM, New York, NY, USA, 2010.

[7] A. Meneely and L. Williams, “Socio-technical developer networks: should we trust our measurements?,” Proceedings of the 33rd International Conference on Software Engineering, pp.281–290, ICSE ’11, ACM, New York, NY, USA, 2011.

[8] C. Bird, “Sociotechnical coordination and collaboration in open source software,” Proceedings of the 2011 27th IEEE International Conference on Software Maintenance, pp.568–573, ICSM ’11, IEEE Computer Society, Washington, DC, USA, 2011.

[9] C. Bird, D. Pattison, R. D’Souza, V. Filkov, and P. Devanbu, “Latent social structure in open source projects,” Proceedings of the 16th ACM SIGSOFT International

Symposium on Foundations of software engineering, pp.24–35, SIGSOFT '08/FSE-16, ACM, New York, NY, USA, 2008.

[10] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," Proceedings of the 2006 international workshop on Mining software repositories, pp.137–143, MSR '06, ACM, New York, NY, USA, 2006.

[11] G. Navarro, "A guided tour to approximate string matching," ACM Comput. Surv., vol.33, no.1, pp.31–88, March 2001.

[12] H. Kagdi and J.I. Maletic, "Combining single-version and evolutionary dependencies for software-change prediction," Proceedings of the Fourth International Workshop on Mining Software Repositories, pp.17–, MSR '07, IEEE Computer Society, Washington, DC, USA, 2007.

[13] P. Rostal and S. Consulting, "Towards an understanding of the role of weak links in software development,".

[14] J.C. Munson and S.G. Elbaum, "Code churn: A measure for estimating the impact of code change," Proceedings of the International Conference on Software Maintenance, pp.24–, ICSM '98, IEEE Computer Society, Washington, DC, USA, 1998.

[15] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, pp.13–23, SIGSOFT '08/FSE-16, ACM, New York, NY, USA, 2008.