卒 業 研 究 報 告 書

題　目　　開発者ネットワークに基づく
　　開発者評価サービスの試作

指導教員　　水野　修　准教授

京都工芸繊維大学 工芸科学部 情報工学課程

学生番号　　10122013

氏　　名　　河端　駿也

平成26年2月14日提出

開発者ネットワークに基づく
開発者評価サービスの試作

平成 26 年 2 月 14 日　　10122013　河端　駿也

概　要

ソフトウェアを分析する研究には，プロダクト，プロセス，リソースの 3 つの観点がある．プロダクトに着目した研究の例としては，ソースコードから抽出したメタデータや，ソースコードに対して何らかの解析を実行した上でその出力を利用するものなどがある．これらは，プロダクトとしてのソースコードに着目している．これに対し，ソフトウェア開発におけるリソース，すなわち，ソースコードの書き手側，開発者自身に着目した研究も存在する．

本研究では，ソフトウェア開発におけるリソースに着目した研究の一環として，オープンソースソフトウェア（OSS）における開発者の評価サービス提供を目的とする．そのために，OSS プロジェクトのリポジトリから開発者評価に役立つ情報を取得し，開発者のデータベース作成，共同開発者同士を結んだ開発者ネットワークの構築，評価サービスの試作をする．具体的には，本研究室で用いているソフトウェア開発者間のネットワーク分析ツールを拡張し，ネットワークにおける開発者間の距離を計算するのに加え，OSS における開発者個人の能力を計算するサービスとした．また，試作したツールを用いて開発者の能力評価が可能かどうかの実験を行った．その結果，ある程度開発者の能力を視覚化することに成功した．

目　次

1. 緒言

ソフトウェアを分析する研究には，プロダクト，プロセス，リソースの 3 つの観点がある．プロダクトに着目した研究の例としては，ソースコードから抽出したメタデータや，ソースコードに対して何らかの解析を実行した上でその出力を利用するものなどがある．これらは，プロダクトとしてのソースコードに着目している．これに対し，ソフトウェア開発におけるリソース，すなわち，ソースコードの書き手側，開発者自身に着目した研究も存在する．

開発者間のネットワーク分析ツール [1] 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もとに，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確認する過程で作成された Web サービスである．結果，96%の開発者が属する巨大なネットワークが存在することがわかっている．

また，オープンソースソフトウェア（OSS）ではプロジェクト内で小さなコミニティが形成されていると仮定し調べたものもある [2]．この論文によると，プロジェクト内で小さなコミニティが自然発生しており，またそのコミニティは技術議論も活発で，お互い協力性もあったことがわかっている．

更に，バグ混入者が行う会話の性質を調べたものがある [3]．この論文によると，バグ修正者の割合はバグ混入者自身とバグ混入者の共同開発者以外の開発者が多く，また，バグトラッキングシステム（BTS）内でバグ混入者はどのような会話を行っていたのかを調べた結果，バグ混入者は会話で重要な役割を果たしているが，バグ混入者同士の会話は乏しいことがわかっている．

本論文では，本研究室で用いているソフトウェア開発者間のネットワーク分析ツールを拡張し，OSS における開発者個人の能力を計算するサービスとしていく．既存のツールでは，開発者ホップ数の計算と，ホップとして仲介した開発者を出力するのみであった．そこで，開発者が OSS においてどの程度の能力か知るために，開発者ホップ数に加え，開発者の能力評価結果を出力する．ただし，今回は試作のため，リポジトリ数は既存のツールで使われている 8,140 より少ない 100 リポジトリのみで行う．本論文のリポジトリ数が少ない状態での試作結果を踏まえ，リポジトリを多くしても実用的な時間で評価結果を計算し出力できるツールとなることを目標と

する．

最後に2章以降の構成を説明する．2章では本研究の研究設問を説明する．3章では開発者ネットワークの分析手法を説明する．4章では実験概要を説明する．5章では実験結果を述べる．6章では結果の考察をする．7章では結言を述べる．

2. 研究設問

研究目的を明確にするため，次の研究設問を設ける．

RQ1: 開発者の評価をリポジトリから得られる情報で行うことはできるか．

RQ2: 開発者評価指標の算出をツールを用いてできるか．

RQ3: 実用的な時間で任意の開発者評価を行うことができるか．

RQ1 は，リポジトリの情報のみで開発者の評価を行えるかを考察する．RQ2 は，開発者の評価に役立つ指標を実験により算出できるかを考察する．そして RQ3 は，ツールが結果を出すのにかかる計算時間が実用に耐えうるものかを考察する．

3. 分析の準備

3.1 ソフトウェアメトリクス

ソフトウェアメトリクスとは，ソフトウェアの規模，複雑さ，保守性などの属性情報を定量的に示す指標である．メトリクスに用いる情報は，プロジェクトにバージョン管理システムを利用していれば入手しやすい．そのため，ソースコードが公開されていてバージョン管理システムを利用しているオープンソースプロジェクトが実験対象になりやすい．

本論文では，OSS のリポジトリから得られたメトリクスデータを基に開発者評価を行う．

3.2 Git バージョン管理システム

Git とは，フリーでオープンソースな分散バージョン管理システムであり，Linux 開発者達が Linux カーネルを管理するために開発されたものである．分散型のため，リポジトリをリモートからローカルにコピーし，ローカルで作業を行うことができる．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できる．Git を用いた Web ベースのホスティングサービスである GitHub が普及したこともあり，多くのプロジェクトで使われている．バージョン管理システムを用いることで開発履歴をたどることができるので，リポジトリマイニングの対象になりやすい．

本論文では，git-log コマンドを用いてリポジトリから情報を得る [4]．本実験で使用したオプションを次に示す．

- –git-dir=
  ’=’ 以降に.git ディレクトリのパスを指定する．
- -p
  パッチ形式で出力する．
- –pretty=format:
  ’:’ 以降に以下の書式でプリティプリントする．今回，書式は%an（開発者名），

%ae（開発者 e-mail アドレス），%n（改行），%ai（開発者のコミット時間），%s（コミットメッセージ）を用いた．

- –all
  参照できるすべてのコミットログを出力する．
- –numstat
  1 フィールド目に追加行，2 フィールド目に削除行，3 フィールド目にファイル名を出力する．

3.3 Google BigQuery

Google BigQuery は，Google が提供する企業向けのビッグデータ解析サービスである．本研究室で用いているソフトウェア開発者間のネットワーク分析ツールでは，サンプルのデータとして含まれている GitHub リポジトリが用いられている．本実験では，そのデータの一部を利用する．

3.4 開発者ネットワーク

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したネットワークである．例として図 3.1 を見る．プロジェクト A では，Alice と Bob がファイル B を通して共同開発者になる．また，プロジェクト B では，Alice と Charlie がファイル D を通して共同開発者になる．これより，図 3.2 の開発者ネットワークが構築できる．ここで，Bob と Charlie の間には直接な共同開発者関係はないが，Alice を通すことで間接な共同開発者関係となる．

3.5 開発者ホップ

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と定義する．開発者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．また，ノード，開発者の重み付けはしない．図 3.2 より，Bob を起点とした開発者ホップを考えると，Bob の開発者ホップは 0，Alice の開発者ホップは 1，

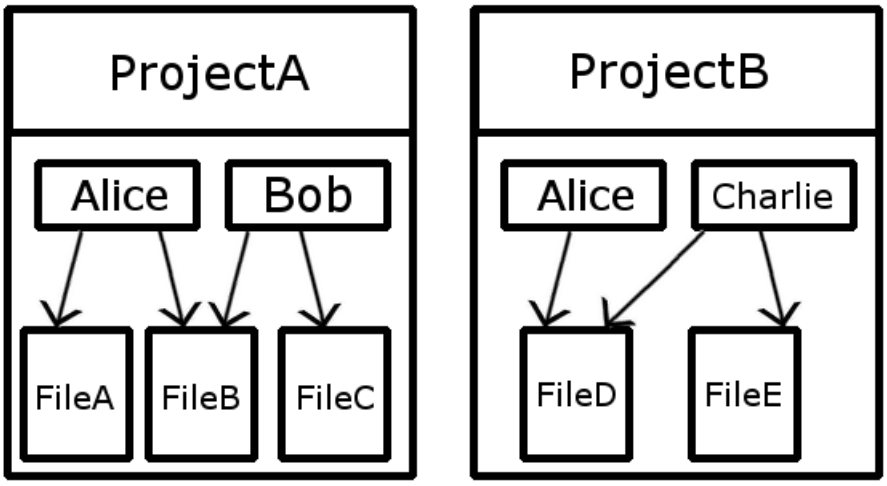

図 3.1　ファイル更新関係の例

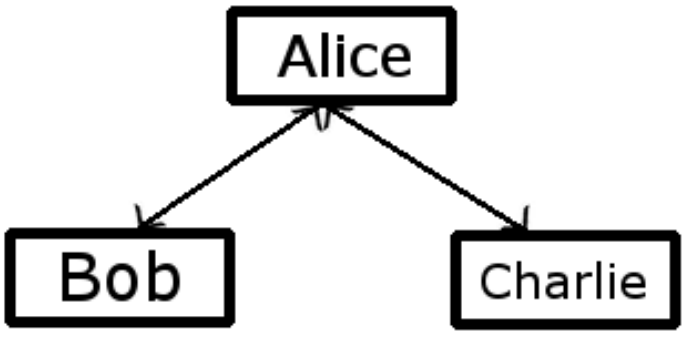

図 3.2　構築した開発者ネットワーク

そして Charlie の開発者ホップは 2 となる．構築した開発者ネットワークから任意の 2 開発者間の開発者ホップを求めるときは，ダイクストラのアルゴリズムを用いる．

4. 開発者ネットワークの分析と開発者評価

4.1 リポジトリの収集

本実験では，本研究室で用いられている，Google BigQuery から取得した GitHub アーカイブを用いる．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，GitHub アーカイブの取得には次の Query を実行している．

```
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 がわかる．ここで，次のコマンドを実行することで，プロジェクトのリポジトリを収集する．

```
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 8,140 件のプロジェクトのリポジトリを収集することができる．

4.2 分析の手順

本研究において，開発者の評価を行うためには，先行研究で収集されたデータだけでは足りないものがある．そのため，全データをリポジトリから収集し直し，データベースを作成し直す必要がある．足りないデータは次の通りである．

- コミット日時

  1 日あたりのコミット，修正回数を求めるのに必要となる．
- コミットメッセージ

  修正コミットの判断に必要となる．
- コミット時におけるファイルの追加，削除行数

  総コード行数を求めるのに必要となる．

手順は次の通りとする．

1. git-log コマンドのオプションを変更

   先行研究時の git-log コマンドは次の通りであった．

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

   ここで，$INPUT はリポジトリの存在するパスとする．このコマンドでは，上で述べた足りないデータが取り出せないため，オプションを変更し，代わりに次の git-log コマンドを実行する．

   ```
   git --git-dir=$INPUT/.git log -p \
     --pretty=format:'Author: %an <%ae>%nDate: %ai%nSubject: %s%n' \
     --all --numstat
   ```

   これを実行することで，先行研究で用いたデータと足りないデータのすべてが出力される．
2. git-log コマンドの出力からデータを抜き出す

   オプションを変更した git-log コマンドの出力から，先行研究で用いるデータと足りないデータのすべてを抜き出す．

4.3 データベースの作成

収集したリポジトリからデータベースを作成する手順を説明する．

1. 任意のリポジトリからコミットデータの取得

   4.2 章で示した git-log コマンドを実行し，開発者情報を取り出す．

2. データベースの作成

   取得したコミットデータを，後述のデータベースに登録していく．

   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．しかし，デフォルトネーム，デフォルトメールアドレスを使用している開発者もいるため，誤って同一人物と判定されてしまう問題がある．そこで，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，または 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しないとする．

3. 上記の手順をすべてのリポジトリに適用
4. 共同開発者の登録

   共同開発者は，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とする．

4.4 データベースのスキーマ

今回，実験で用いるテーブルと属性は次の通りである．

- authors

  カラムに開発者名，e-mail アドレスを持つテーブル．

  - id: 開発者 ID．開発者毎に一意の値を持つ．
  - name: 開発者名．
  - email: 開発者の e-mail アドレス．

- projects

  カラムにプロジェクト名を持つテーブル．

  - id: プロジェクト ID．プロジェクト毎に一意の値を持つ．
  - name: プロジェクト名．

- files

  カラムにファイル関連の情報を持つテーブル．

  - id: ファイル ID．各コミット時のファイル毎に一意の値を持つ．
  - project: プロジェクト ID．ファイルが属すプロジェクトを示す．

- author: 開発者 ID．ファイルを更新した開発者を示す．
- 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．
- hash: ファイルハッシュ．ファイルがコミット時に付けられたハッシュ情報を持つ．
- identical: 同一ファイル ID．files テーブルの中で同じファイルを見分けるために使用する．
- author_date: コミット日時．UTC，協定世界時で保存されている．
- subject: コミットメッセージ．
- insertion: 追加行数．前回から今回のコミット間でのファイル追加行数を持つ．
- deletion: 削除行数．前回から今回のコミット間でのファイル削除行数を持つ．
- after_or_before: ファイルに付けられたハッシュが変更前か変更後かの情報を持つ．

- coauthors

  カラムに共同開発者の情報を持つテーブル．

  - id: 共同開発者 ID．
  - author: 開発者 ID．
  - coauthor: 共同開発者の開発者 ID．
  - identical: 同一共同開発者 ID．この ID を通して共同開発者同士が繋がっている．

- sameauthors

  カラムに同一開発者の情報を持つテーブル．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により，ある開発者の持つ情報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.

4.5 開発者ネットワークの構築

作成したデータベースから，開発者ネットワークを以下の手順で構築する.

1. データベースから共同開発者の組をすべて抜き出す
2. あるノードから辿れる，すべてのユニークなノード数を開発者ネットワークの規模とする
3. 2. の処理をすべてのノードで行う

4.6 開発者評価の内容

開発者を評価する内容とその評価値を次のように設定する.

- OSS への貢献度
  - コミット回数
  - 1 日あたりのコミット回数
  - 修正回数
  - 1 日あたりの修正回数
  - 参加プロジェクト数
  - 総コード行数
- 活発度
  - 1 日あたりのコミット回数
  - 1 日あたりの修正回数
- 修正力
  - 修正回数
  - 1 日あたりの修正回数
- 扱えるプログラミング言語数
  - 拡張子毎のファイル数

4.7 各開発者評価値の求め方

4.6 章で設定した各開発者評価値の求め方を次に示す.

- コミット回数

  ユニークなコミット日時の個数とする.

- 1 日あたりのコミット回数

  計算式は次とする.

  コミット回数 / （最新のコミット時間 - 最古のコミット時間）

  ここで，コミット回数が 1 回の場合は零，また，最新と最古のコミット時間の差分が 1 日未満であれば，コミット回数を 1 日あたりのコミット回数とする.

- 修正回数

  修正コミットは，コミットメッセージに

  'correct', 'fix', 'modify', 'resolve', ' 修正'

  のいずれかを含むものとする．これらの単語は，大文字小文字関係なく，時制変化していても修正コミットとみなす.

- 1 日あたりの修正回数

  1 日あたりのコミット回数と同じ求め方をする.

- 参加プロジェクト数

  開発者がファイルを編集した，ユニークなプロジェクトの個数とする.

- 総コード行数

  計算式は次とする.

  追加行数の合計 - 削除行数の合計

  値は物理 LOC になる.

- 拡張子毎のファイル数

  編集した拡張子をファイル名から抜き出し，それぞれでファイル数をカウントし，降順で出力する．拡張子が無いファイル，ファイル名の最後が ~ のバックアップファイル，または .DS_Store のプラットフォーム依存ファイルは含めない.

5. 分析結果

5.1 開発者データベースの作成

今回，Google BigQuery から取得した GitHub リポジトリ 8,140 個のうち，無作為に 100 個のリポジトリを選んだ．そのリポジトリから，4 章に示した方法を用いて開発者データベースを作成した．表 5.1 にリポジトリの概要を示す．また，図 5.1 に開発者の参加プロジェクト数のグラフを示す．今回，参加プロジェクト数が 1，つまり一つのプロジェクトにしか参加していない開発者は 1,617 人であった．これは総開発者数の 94%にあたる．表 5.2 にデータセットの詳細を示す．

5.2 開発者ネットワークの構築

作成したデータベースから，4.5 章の通りに開発者ネットワークを構築した．表 5.3 にネットワークノードとエッジの詳細を示す．ここで，ノード数は共同開発者の関係に含まれるユニークな開発者の数，開発者関係グラフエッジ数はユニークな同一共同開発者 ID の数，開発者ネットワークグラフエッジ数はユニークな共同開発者関係のエッジ数である．また，表 5.4 に構築した開発者ネットワークの規模を示す．ここで，規模が 453 と最も大きいネットワークは，単一のプロジェクトのみで形成されたネットワークであり，このプロジェクトは最大の開発者数を持つプロジェクトだった．対して，規模が 326 と 2 番目に大きいネットワークは，20 プロジェクトから形成されたネットワークであった．これは，総開発者数の 19.04%，総プロジェクト数の 15.71%にまたがっている．なお，その次に規模が 74 と 3 番目に大きいネットワークは，単一のプロジェクトのみで形成されたネットワークであった．

5.3 開発者評価サービスの作成

本研究室で用いられているツールに，4.7 章の値を追加で表示するようにした．図 5.2 に作成した開発者評価サービスのスクリーンショットを示す．このツールは，まず二人の開発者の名前を，From と To に入力する．以降ではこの 2 人の開発者を From 開発者，To 開発者と呼ぶ．その後 submit ボタンを押すと，計算結果が図 5.2

表 5.1　リポジトリ概要

総プロジェクト数	136
総開発者数	1,712
総ファイル数	116,879
総更新数	533,144

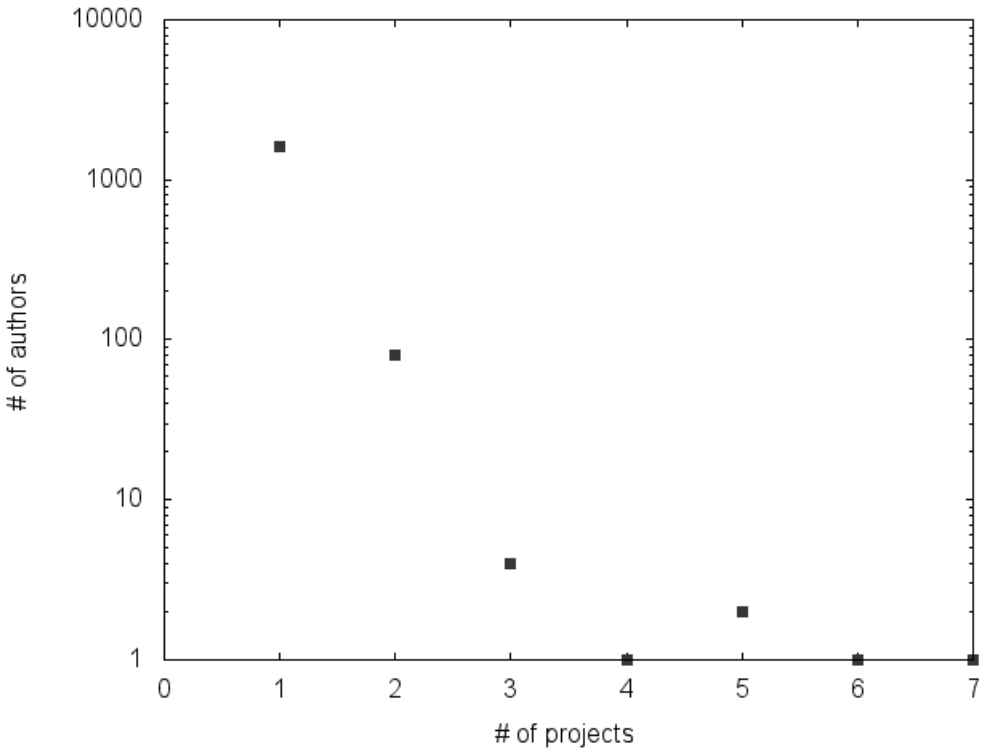

図 5.1　開発者の参加プロジェクト数

表 5.2　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	5	13.35	454
プロジェクト毎のファイル数	1	72.5	859.40	27,256
プロジェクト毎の更新数	1	244.5	1,625.17	37,943
ファイル毎の更新数	1	1	2.24	729
ファイル毎の開発者数	1	1	1.45	41
開発者毎のプロジェクト数	1	1	1.06	7

表 5.3　ネットワークノードとエッジ

ノード数	1,671
開発者関係グラフエッジ数	34,789
開発者ネットワークグラフエッジ数	12,737

表 5.4　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
453	1	17	1
326	1	16	1
74	1	12	1
67	1	11	2
55	1	10	2
53	2	9	1
39	1	8	2
36	1	7	5
34	1	6	3
32	1	5	8
29	1	4	3
27	2	3	14
25	1	2	10
22	2	1	41
18	1		

の枠部に表示される．出力内容は，ホップ数，ホップ経路（開発者，経路となったプロジェクトとファイル），To 開発者の評価情報およびホップ数の分布である．

ここで，To 開発者の評価情報のみ出力し，From 開発者の評価情報を出力しなかった理由を述べる．本実験の開発者評価サービスは開発者評価値のみの出力を目的としたツールではなく，先行研究のツールに追加する機能だった．そして先行研究のツールでは，図 5.2 の枠部以外の機能，ホップ数，ホップ経路，To 開発者のホップ数分布が実装されていた．そのため，今回の評価サービスは To 開発者のホップ数分布に合わせて，To 開発者のみの開発者評価値を出力する機能とした．

表 5.5 と表 5.6 に実行例を示す．それぞれ，Damien Elmes 氏は最もコミット，修正回数が多い開発者，Mark Engel 氏は最も 1 日あたりのコミット回数が多い開発者，Martin Provencher 氏は最も 1 日あたりの修正回数が多い開発者，Sam Vermette 氏は最も参加プロジェクト数が多い開発者，Patrick Meehan 氏は最も総コード行数が多い開発者，Roel van Dijk 氏は最もファイルのユニークな拡張子数が多い開発者であった．

5.4 開発者評価の時間測定

実際に開発者評価サービスを利用し，結果出力までどの程度の時間がかかるのか調べた．本実験では，Google Chrome に標準搭載されているデベロッパーツールの Network パネルを用いた [5]．表 5.7 にその結果を示す．ここで，かかった時間とはリクエストの始めから結果レスポンスの取得完了までである．ホップ数に着目し，同じホップ数だが別の開発者の組で時間測定した結果も，表のかかった時間付近の結果であった．

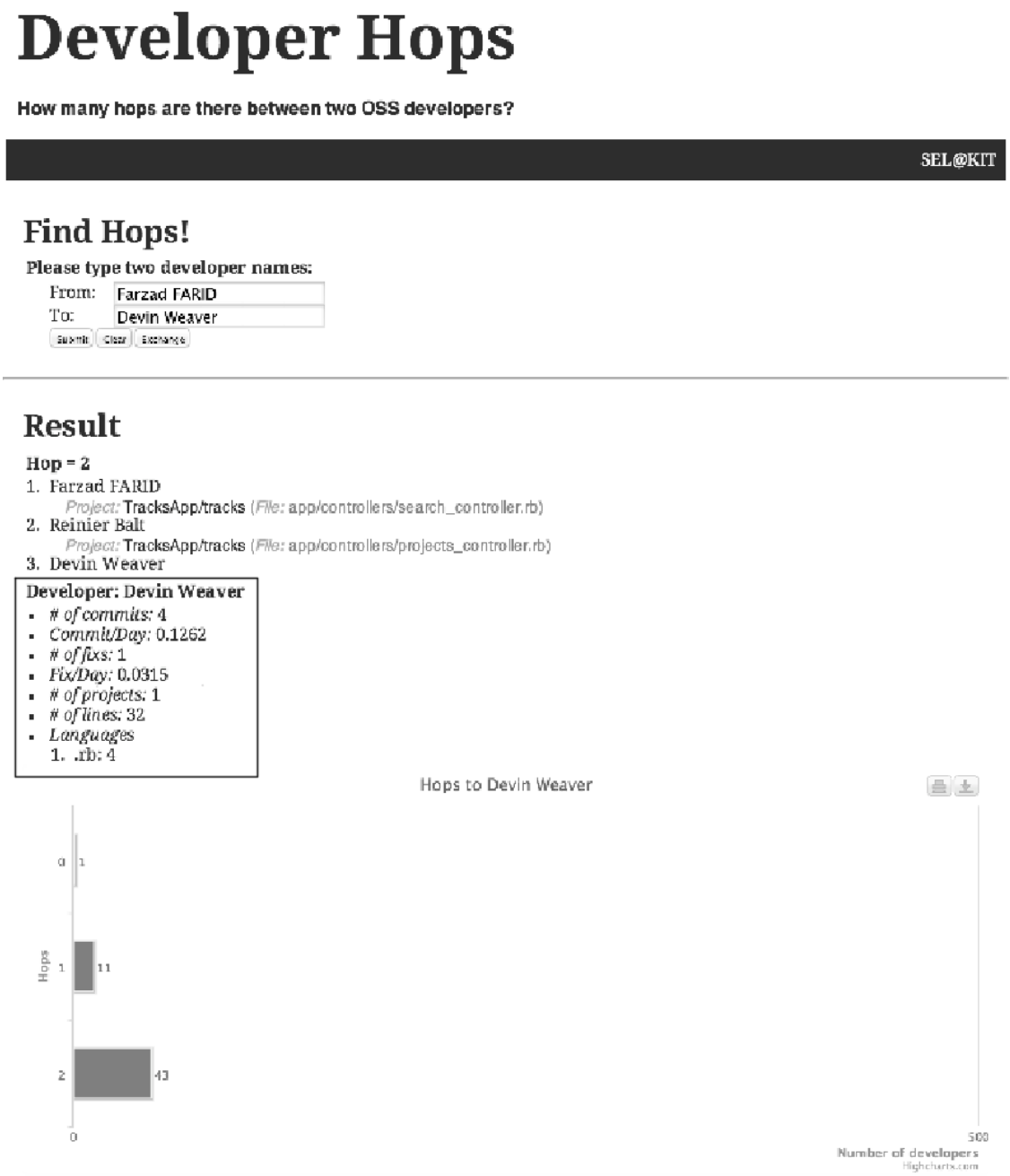

図 5.2　開発者評価サービス

表 5.5　コミット，修正関連の実行結果例

開発者名	コミット回数	1 日あたりのコミット回数	修正回数	1 日あたりの修正回数
Damien Elmes	5,002	3.3286	736	0.4898
Mark Engel	44	44	2	2
Martin Provencher	16	16	5	5
Sam Vermette	418	0.6568	85	0.1336
Patrick Meehan	1,145	1.9206	360	0.6039
Roel van Dijk	70	1.0895	13	0.2023

表 5.6　表 5.5 以外の実行結果例

開発者名	参加プロジェクト数	総コード行数	ファイルのユニークな拡張子数
Damien Elmes	2	164,990	1
Mark Engel	1	-90	3
Martin Provencher	1	4,196	1
Sam Vermette	7	49,439	1
Patrick Meehan	2	7,447,862	3
Roel van Dijk	1	3,528	349

表 5.7　開発者評価サービスの時間測定結果

開発者 (From/To)	ホップ数	かかった時間 [s]
Aaron Lerch / Zach Wise	0	0.21
Chris O'Hara / Susmitha Girumala	1	0.33
Aaron Lerch / Adam Roben	2	0.95
Brad Phelan / Vadim M. Bryshev	3	0.72
RFelix / U-chrisoChris	4	0.70
Jeff Pickhardt / Vadim M. Bryshev	5	0.94
BJTerry / Brad Phelan	6	2.44
Rick DeNatale / Vadim M. Bryshev	7	2.37

6. 考察

6.1 妥当性の検証

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

本実験では，オープンソースのプロジェクトを対象としている．そのため，開発者情報の登録が厳密でないので，名前が 'root' や 'unknown'，e-mail アドレスが 'none@none' といった個人の特定ができないデータがあった．よって，個人判定が厳密にできていないのでデータセットに不備がある.

プログラムの不備

本実験では，リポジトリ収集，データベース作成，開発者ネットワーク構築，開発者評価，結果の集計などでプログラムを用いているが，これらのプログラムに不備がある可能性がある.

修正コミット判断の不備

本実験では，修正コミットの判断にて，コミットメッセージに修正関連の単語を含んでいれば修正コミットとみなした．そのため，その単語が含まれていないコミットは修正コミットとみなせておらず，また逆に，その単語を含んでいるが修正コミットでないコミットを修正コミットとみなしている可能性がある.

対象リポジトリの不足

今回利用したリポジトリは，本研究室が利用しているものであり，そこでは 96.32%の開発者が 1 つの開発者ネットワーク上で繋がっている．しかし，今回は 8,140 リポジトリのうち 100 リポジトリのみを利用したため，規模が 453 と最も大きいネットワークでも総開発者数の 26.46%であった（表 5.4）．また，これは単一のプロジェクトのみから形成されていた.

6.2 研究設問の考察

本実験で定めた研究設問に対して考察を行う.

RQ1: 開発者の評価をリポジトリから得られる情報で行うことはできるか.

本実験では，既存のツールに 4.7 章の開発者情報を追加し，テキスト形式で出力するようにした．図 5.2 を見る．To に入力した開発者の評価が，枠部に出力できた．また，表 5.5 および表 5.6 の開発者や他の開発者で何度か実行したときも，正常に評価値が出力できていた．よって，今回設定した開発者情報は出力できることがわかる.

RQ2: 開発者評価指標の算出をツールを用いてできるか.

表 6.1 に，今回利用した 100 リポジトリ，総開発者 1,712 人の開発者評価値集計を示す．今回の評価値の計算方法では，1 日あたりのコミット回数，1 日あたりの修正回数どちらの中央値も 0 となり，開発者の評価としてわかりにくいものとなった．0 ではなく，比較できる数値として出力する必要がある．また，表 5.5 および表 5.6 において，1 日あたりのコミット回数が最も多い Mark Engel 氏と，1 日あたりの修正回数が最も多い Martin Provencher 氏を見る．この 2 人は，1 日あたりのコミット回数とコミット回数，1 日あたりの修正回数と修正回数が，それぞれ同じ値となっている．これは，4.7 章の求め方より，最新と最古のコミット時間間隔が 1 日未満だったためである．1 日未満で活発な活動をした開発者ではあるが，その値は最新と最古のコミット時間間隔が 1 日以上の開発者と正確な比較ができていない．そこで修正案を次に示す.

- “回数 / 日数の差分” のまま表示
  (最新のコミット時間 - 最古のコミット時間) のみ計算し，除算は行わないようにする.
- “回数 / (ある定めた時間 - 最古のコミット時間)”
  最新のコミット時間ではなく，最古のコミット時間から定めた時間までの差分とする．定める時間の案としては，リポジトリを収集した時刻，プログラムの実行開始時間などが考えられる.
- 差分時間の単位を変更
  1 日あたりではなく，1 週間，または 1ヶ月などの期間に変更する.

また，全体の開発者と構築されたネットワークに属す開発者を比べていく．表 5.4 において，規模が 326 と 2 番目に大きいネットワークに属す開発者のみ

表 6.1　100 リポジトリの全開発者評価値の集計

	最小値	中央値	平均値	最大値
コミット回数	0	1	24.00	5,002
1 日あたりのコミット回数	0	0	0.5640	44
修正回数	0	0	4.66	736
1 日あたりの修正回数	0	0	0.1073	5
参加プロジェクト数	0	1	1.06	7
総コード行数	-2,110	14	16,593.01	7,447,862
ファイルのユニークな拡張子数	0	1	3.28	349

表 6.2　ネットワーク規模 326 の開発者評価値の集計

	最小値	中央値	平均値	最大値
コミット回数	1	1	17.87	2,857
1 日あたりのコミット回数	0	0	0.8178	44
修正回数	0	0	2.68	253
1 日あたりの修正回数	0	0	0.1224	5
参加プロジェクト数	1	1	1.12	6
総コード行数	-142	2	8,097.29	1,451,050
ファイルのユニークな拡張子数	1	1	1.82	68

で開発者評価値を集計した結果を表 6.2 に示す．このネットワークを選んだ理由は，規模が 453 と最も大きいネットワークでは共同開発者の関係がプロジェクト間を超えず，単一のプロジェクトでしか構築されていなかったため，規模が 326 と 2 番目に大きく共同開発者の関係が 20 プロジェクトにまたがっていたこのネットワークを選択した．表 6.1 と表 6.2 を見比べる．規模が 326 のネットワークに属す開発者は，全体の開発者よりコミット回数と修正回数の平均値は低いが，1 日あたりのコミット，修正回数の平均値は高いことがわかる．また，全体で 1 日あたりのコミット，修正回数が最も高い開発者，Mark Engel 氏と Martin Provencher 氏は，326 のネットワークに属していることもわかる．また，参加プロジェクト数の平均値も全体より高いが，総コード行数やファイルのユニークな拡張子数の平均値は全体より低くなっている．このことから，326 のネットワークに属す開発者は全体と比べ，活発度は高いが物理 LOC 値は高くなく，またコミット時に用いていた言語の拡張子数は少ないことがわかる．

RQ3: 実用的な時間で任意の開発者評価を行うことができるか．

表 5.7 について，ホップ数を固定した場合とホップ数別の場合とで考察していく．

まずホップ数を固定して考察する．同じホップ数だが別の開発者の組で時間測定した結果も，表に示した時間とほぼ同じであった．このことから，今回追加した開発者評価値は，開発者に依存する計算時間ではないことがわかる．

次にホップ数別に見ていく．ホップ数が 5 以下の入力を見ると，1 秒以内に結果を返していることがわかる．これより，ホップ数が 5 以下なら，実用的な時間で評価を行うことができる．対して，ホップ数が 5 より大きい 6 や 7 のときは 2 秒程度の時間がかかっている．また，今回の実験では 100 リポジトリのみを用いているが，このツールを正確な開発者評価に役立てるには，更に多くのリポジトリが必要となり，リポジトリが多くなると最大のホップ数も大きくなってくる．そのため，実用的な時間で評価するには，ホップ数が大きいときの計算を改善する必要がある．

6.3 今後の課題

以上の考察を踏まえ，今後の課題を考察する．

対象リポジトリの増加

今回のリポジトリ数 100 で構築した開発者ネットワークのうち最も大きいものは総開発者数の 26.46%であり，これは全リポジトリを用いたときの 96.32%より小さなものとなった．また，このときのネットワークは単一プロジェクトのみで構築されていた．今後はリポジトリ数を増やして一般化する必要がある．

計算時間の短縮

今回のリポジトリ数が少ない場合でも，ホップ数が 6, 7 と大きいときには結果取得まで 2 秒かかった．今後，リポジトリ数を多くすれば，開発者ネットワークの規模も大きくなり，大きいホップ数で繋がる開発者も増えてくる．そのため，実用的な時間で結果を出せるようにする必要がある．

修正コミット判断の強化

今回の修正コミット判断は，コミットメッセージに修正関連の単語を含んでいれば修正コミット，という単純なものであった．結果の精度を上げるためにも正確な修正コミット判断が必要である．

1 日あたりのコミット，修正回数計算の見直し

RQ2 で考察した内容や他の方法で計算する必要がある．

拡張子毎のファイル数結果の強化

今回の実験で評価値として追加した “拡張子毎のファイル数” は，開発者がどんなプログラミング言語を使用しているのか知るために設定した．今回のツールでは，言語毎に集計せず，拡張子のみを使用したファイル数で降順に出力している．今後は，言語ごとの物理 LOC，開発者が得意としている言語，開発者がどの分野でプロジェクトに貢献しているかなどを出力する強化が必要となる．

また，拡張子がないファイルは考慮していないため，README や ChangeLog といった拡張子がないファイルは集計できていない．今後は拡張子がないファイルも考慮していく必要がある．

開発者評価値の追加

今回設定した開発者評価値以外にも，リポジトリから有用な開発者情報を抜け出せる可能性がある．

評価結果（OSS への貢献度，活発度，修正力，扱えるプログラミング言語数）の関係考察

修正力が高い人は扱う言語数も多いのか、それとも 1 つの言語だけを扱っているのか、といった各評価結果との関係を考察する必要がある．

結果表示方法の見直し

今回のツールは，既存の出力の下にテキストとして出力しただけであり，見やすさは考慮していなかった．フォントサイズや色，グラフなどを用いて見やすいものとする必要がある．

7. 結言

本研究では，既存のツールに開発者評価情報を追加した．

8,140 リポジトリ中 100 のリポジトリを無作為抽出した．開発者評価のために，リポジトリから収集する情報を増やし，データベースのスキーマを再構築した．データベースの共同開発者関係から開発者ネットワークを構築した．構築した開発者ネットワークでは，リポジトリ数が足りないため大規模なネットワークは構築できていないことがわかった．また，データベースから開発者情報を計算し Web サービスに追加出力させるようにした．

今後の課題として，対象リポジトリの増加，計算時間の短縮，開発者評価値を求める計算方法の見直し，結果表示方法の見直しがある．

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，本学情報工学専攻大西哲朗先輩，椋代凜先輩，岡嶋秀記先輩，研究生胡軼凡先輩，情報工学課程川島尚己君，山田晃久君，学生生活を通じて著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] 出原真人，“巨大ソフトウェアリポジトリ郡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,” 修士論文，pp.1–34，Feb. 2013.

[2] C. Bird，D. Pattison，RaissaD’Souza，V. Filkov，P. Devanbu，“Latent social structure in open source projects,” Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, no.SIGSOFT ’08/FSE-16, pp.24–35, New York, NY, USA, 2008.

[3] M.L. Bernardi, G. Canfora, G.A.D. Lucca, M.D. Penta, and D. Distante, “Do developers introduce bugs when they do not communicate? the case of eclipse and mozilla,” 2012 16th European Conference on Software Maintenance and Reengineering, pp.139–148, Szeged, 27-30 March 2012.

[4] Git, git-log Documentation, git（オンライン）, 入手先 ⟨http://git-scm.com/docs/git-log⟩（参照 2014-02-10）.

[5] Google Developers, Evaluating network performance, Google（オンライン）, 入手先 ⟨https://developers.google.com/chrome-developer-tools/docs/network⟩（参照 2014-02-10）.