

目次

第1章 序論.....	5
1.1. 本章の構成.....	1
1.2. 研究背景.....	1
1.3. 研究目的.....	2
1.4. 研究方法.....	2
1.5. プロジェクトマネジメントの関連.....	2
1.6. 本論文の構成.....	3
第2章 GitHub について.....	5
2.1. 本章の構成.....	5
2.2. バージョン管理システムについて.....	5
2.2. 分散型バージョン管理システム.....	6
2.3. Git について.....	7
2.3.1. Git の特徴.....	7
2.4. GitHub とは.....	8
2.4.1. GitHub の基本用語.....	8
2.4.2. GitHub の機能.....	10
第3章 Big Data について.....	13
3.1. 本章の構成.....	13
3.2. Big Data とは.....	13
3.2.1. 4V による Big Data の定義.....	14
3.2.2. 3V でのビッグデータの定義.....	15
3.3. ビッグデータ処理のパターン.....	16
3.3.1. バッチ処理.....	16
3.3.2. インタラクティブクエリー処理.....	17
3.3.3. ストリームデータ処理.....	19
3.4. Google BigQuery.....	20
3.4.1 基本的なシステム構成.....	20
3.4.2 BigQuery を使用する.....	21
第4章 開発・調査.....	31
4.1. 本章の構成.....	31
4.2. 調査対象.....	31
4.2.1. 調査対象データ.....	31
4.3. 調査方法.....	31
4.3.1. 調査環境構築.....	31

4.3.2. Windows での導入.....	32
4.3.3. おおもとになるデータの抽出.....	33
4.4. MySQL.....	35
4.4.1. MySQL への接続.....	35
4.4.2. データベースの作成と削除.....	36
4.4.3. データのインポート.....	37
4.4.4. インデックス.....	37
4.4.5. SELECT 文の詳細.....	37
4.4.6. SQL.....	39
第 5 章 調査結果・考察.....	31
5.1. 本章の構成.....	43
5.2. 調査結果.....	43
5.2.1. 全体の比較.....	43
5.3. 考察.....	60

図目次

図 1	分散型バージョン管理システム 参考文献[4]	6
図 2	ビッグデータの 4V	15
図 3	BigQuery の入出力 参考文献[1]	18
図 4	ストリームデータ処理 参考文献[1]	19
図 5	BigQuery 処理パターン 参考文献[1]	20
図 6	Google BigQuery 公式画面 (https://cloud.google.com/bigquery/what-is-bigquery)	21
図 7	サインアップの画面	21
図 8	Projects 画面	22
図 9	プロジェクト作成サブ画面	23
図 10	プロジェクト画面	24
図 11	BigQuery 選択	24
図 12	BigQuery 画面から dataset 作成	25
図 13	Display project	26
図 14	Project ID 指定	26
図 15	COMPOSE QUERY の選択	27
図 16	コードの入力	27
図 17	コードの実行	28
図 18	実行結果	29
図 19	Windows 版の XAMPP をダウンロード	32
図 20	XAMPP Control Panel	33
図 21	BigQuery で処理	34
図 22	MySQL の起動	35
図 23	プロジェクト数, 平均 Fork 数	46
図 24	プロジェクト数, 標準偏差	46
図 25	URL 別 Fork 数	47
図 26	JavaScript	48
図 27	Ruby	48
図 28	Java	49
図 29	Python	49
図 30	C	50
図 31	PHP	51
図 32	C++	51
図 33	Objective-C	52
図 34	VimL	53
図 35	C#	53

図 36	Perl.....	54
図 37	Powershell.....	54
図 38	Haxe	55
図 39	FORTTRAN	56
図 40	F#.....	56
図 41	Visual Basic.....	57
図 42	Delphi.....	57
図 43	ColdFusion.....	58
図 44	Scheme	58
図 45	Assembly	59
図 46	Arduino.....	60

表目次

表 1	単位接頭辞と十進表記 引用文献[1]	13
表 2	ビッグデータ 3 つの処理パターン 引用文献[1].....	16
表 3	テーブル定義.....	39
表 4	プロジェクト数, 平均 fork 数, 標準偏差.....	43

第 1 章

序論

1.1. 本章の構成

第1章では、本論文の序論を述べる。研究背景、研究目的、研究方法、プロジェクトマネジメントの関連、本論文について記述する。

1.2. 研究背景

ソフトウェア開発プロジェクトのための共有ウェブサービスである、GitHub のプロジェクトについて調べれば、オープンソフトウェア開発プロジェクトの実態がつかめるはずである。

実際に GitHub を調べて分かったことの例として、怒りの表現を含むコミットメッセージの割合、地域によるオープンソースプロジェクトへの貢献者などの分布図があげられる。これらの結果は、GitHub Data challenge というイベントで上位に入賞している分析結果である。

GitHub のデータ解析は難しい。なぜなら、データが膨大なため、その収集と処理が難しいからである。データの収集が難しいという問題は、一つのプロジェクトにより簡単になった。大量のデータを集めるために、GitHub のプロジェクトのタイムラインを記録し、アーカイブ化させ、簡単にアクセスできるようにするためのプロジェクトで GitHub Archive である。

データの処理が難しいという問題は、データ量が多すぎるために膨大な量のデータを処理するソフトウェアが少ない点である。GitHub Archive と連動させデータ処理ができるソフトウェアに Google BigQuery がある。BigQuery は、簡単にビッグデータを処理するためのソフトウェアであり、SQL に似たクエリを従来のやり方よりも短時間で簡単に実行できる。このソフトウェアの登場により手軽に大量のデータを処理することができるようになった。

これまでの調査で、オープンソフトウェア開発でどのようなプログラミング言語がよく使われているかを調べることに成功したが、プロジェクトが Fork される確率の、プログラミング言語による違いが分かれば、オープンソフトウェア開発プロジェクトについての理解が深まると思われる。Fork とは、GitHub 上で公開されている成果物に独自の変更を加える際に行う複製のことである。

1.3. 研究目的

GitHub 上で公開されているオープンソフトウェア開発プロジェクトを Google BigQuery を利用し調査する。オープンソースソフトウェアの開発プロジェクトにおいて、使用するプログラミング言語が異なると、Fork される確率、つまりプロジェクトに貢献する人が現れる確率が異なるということがわかっている。

しかし、この結果は、Fork された回数が多いものについてのみ調査して得られたものであった。

そこで本研究では、Fork された回数が非常に少ないものも対象にして、プログラミング言語による貢献者の出現確率を調査する。

1.4. 研究方法

大量のデータを処理することが予想されるので Google BigQuery を利用する。Google BigQuery を使って、GitHub 上のプロジェクトが採用しているプログラミング言語と Fork されている数を収集・統計処理し、Fork される確率のプログラミング言語による違いを明らかにする。

1.5. プロジェクトマネジメントの関連

この研究では、Fork される確率のプログラミング言語による貢献者の出現確立を調査する。

ステークホルダーとは、プロジェクトの意思決定、アクティビティ、成果に影響したり、影響されたり、あるいは自ら影響されると感じる個人、グループ、または組織である。ステークホルダーは、プロジェクトに積極的に関与したり、プロジェクトのパフォーマンスや完了によって自らの利害がプラスまたはマイナスの影響を受けたりする。さまざまなステークホルダーには競合する期待があり、プロジェクト内でコンフリクトを生じることがある。ステークホルダーはまた、戦略的なビジネス目標やその他のニーズを満たす成果を得るため、プロジェクト、プロジェクトの成果物、およびプロジェクト・チームに影響を及ぼすことがある。(PMBOK 見てる) [1]

以上のことから貢献者をステークホルダーと捉えることができるので本研究は PMBOK が提唱する、プロジェクト・ステークホルダー・マネジメントに最も関連があると考えられる。

1.6. 本論文の構成

第1章では序論，第2章ではバージョン管理システムについて，Gitを中心として記述し，本研究での調査対象である，GitHubについて記述する．第3章ではBig Dataについての説明を記述する．第4章ではプロジェクトマネジメントとの関係を記述する．第5章では具体的な調査方法，調査経過，データを示し，データ解析を行い，考察を行う．

参考文献

- [1] The GitHub Data Challenge. <https://github.com/blog/1118-the-github-data-challenge>(参照 2015-1-28).
- [2] The GitHub Data Challenge II. <https://github.com/blog/1450-the-github-data-challenge-ii>(参照 2015-1-28).
- [3] Data Challenge II Results .<https://github.com/blog/1544-data-challenge-ii-results> (参照 2015-1-28)

第 2 章

GitHub について

2.1. 本章の構成

本章では本研究で調査する対象であるバージョン管理システム、GitHub についての基本知識、GitHub で行われている GitHub Data challenge について記述していく。

2.2. バージョン管理システムについて

バージョン管理システムは、ファイルの履歴を管理するシステムであり、修正や追加などの作業によって生成されたファイルについての複数の履歴を記録し、後から古い履歴の取り出しや、差分の参照が可能になるシステムである。これらのファイルの更新履歴をリポジトリと呼び、自分が作業したファイルの更新をリポジトリに反映させることをコミットするという。またバージョン管理システムソフトウェアによっては、ファイルの DELETE や移動の履歴を確認する機能や、特定の利用者がファイルの管理する権限を獲得するロック機能、複数の変更を統合するマージ機能がある。

開発プロジェクトにおいて複数人で同一のファイルを編集する必要があるとき、バージョン管理システムの機能が役立つのである。バージョン管理システムを利用すると、更新者や変更点、変更日時が確認できるため、誰がいつこの編集を行ったのかすぐに理解することが出来、混乱や時間の無駄遣いを避けることが出来、とても効率的に作業が進められるのである。

ソフトウェア開発においては、バージョン管理システムは特に有効的である。ソースコードなど長い文を編集した際など変更点を容易に把握でき管理できるので、障害がいつ発生したのか、どの時点から問題となっていたのか、いつ修正されたのかななどを容易に調べることが出来、早期の門ファイ解決が可能になる。

また、バージョン管理システムは、管理方法の違いにより 3 つに分類される。ファイル単位で個別バージョン管理を行う「個別バージョン管理システム」、リポジトリをサーバーで一元管理し、コミットなどの操作はメンバーが行う「集中型バージョン管理システム」、リポジトリをメンバーで管理でき、そのメンバー間でリポジトリの連携が可能である「分別型バージョン管理システム」である。[4]

2.2. 分散型バージョン管理システム

3つの種類のバージョン管理システムが存在すると前述したが、ここでは本論文の調査環境である GitHub に用いられるバージョン管理システムの分散型バージョン管理システムについて述べる。

分散型のバージョン管理システムでは、リポジトリをサーバー上ではなく、クライアントであるメンバーのコンピュータ上にも作成し、コミットの参照や差分を所得する場合に手元にあるリポジトリにアクセスするため、常にネットワークに接続している必要がないのである。またローカルファイルにリポジトリが存在するため、高速に動作することが可能であり、変更点を送受信する仕組みがあるため、複数のリポジトリ間で連携することが出来る。この分散型バージョン管理システムを行えるのが Git である。以下に分散型バージョン管理システムを図で表す。[4]

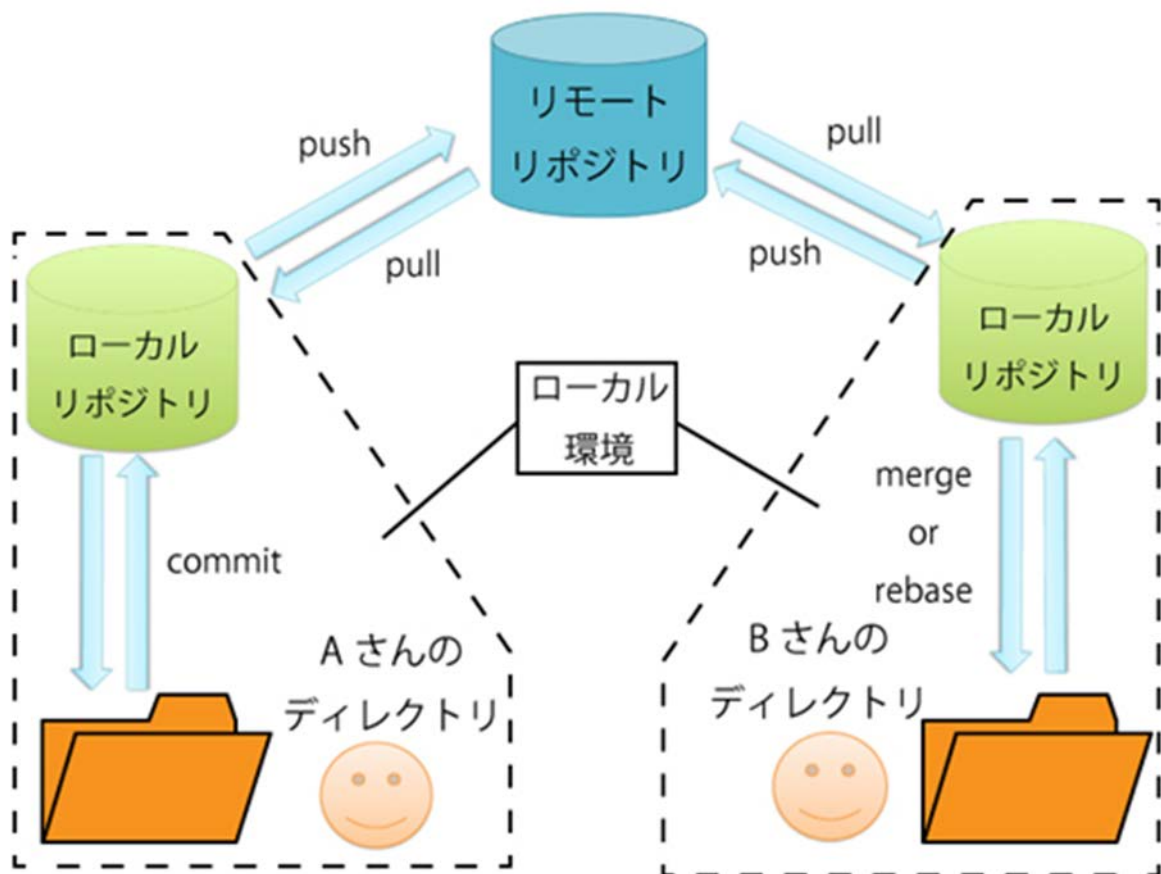


図 1 分散型バージョン管理システム 引用文献[4]

2.3. Git について

分散型バージョン管理システムである Git は 2005 年、Linux カーネル開発現場での必要性から開発が始まった。それまで、Linux カーネル開発のソース管理には BitKeeper というバージョン管理システムが用いられていたのである。これは BitMover 社製の商用のバージョン管理システムである。Bitkeeper は、先進の分散型バージョン管理システムで、カーネルプロジェクトが採用した当時、オープンソースの世界にはこれに匹敵する分散型バージョン管理システムの使用が不可欠であった。そのため、Linux は Git を開発したのである。

Git は分散型バージョン管理システムのため、サーバーを必要としない。またユーザーそれぞれのコンピュータ上にリポジトリを持ち、それぞれが互いに連携しあうことができる。さらに、基本的なそれぞれのリポジトリにすべての履歴が保存されるため、差分やログの表示などを高速に行えるのである。

リポジトリ間連携はネットワーク通信やメールを経由して行う。他にもリポジトリを共有リポジトリとして公開する仕組みや、ユーザー管理と組み合わせる方法があり、集中型バージョン管理システムのような利用形態もとれる。

また、他のバージョン管理システムとのデータ交換も可能であり、既存の他のバージョン管理システムのリポジトリを Git リポジトリへ変更することや、中央リポジトリに他のバージョン管理システムのリポジトリを利用し、手元では Git を利用する、といった形態をとることができるのである。

オープンソースのバージョン管理システムとしては、CVS や Subversion が有名で、今でもこれらの集中型システムはよく使われている。しかし、近年になって、Linux カーネル、X.org, Ruby on Rails, Perl といった有名なプロジェクトが Git に乗りかえて成功裡に使用しているのをみて、Git を使用し始めとする分散型バージョン管理システムを使用するプロジェクトは、飛躍的に増加してきている。[4]

2.3.1. Git の特徴

Git は主にファイル自身、ファイルの集合としてのツリー、そしてコミット情報という 3 つの情報を管理している。それぞれの情報はハッシュ値をもとに管理され、このハッシュ値はファイルが同一かどうかの判断にも用いられる。Git はコミット情報を差分管理ではなく、ファイルそのままを保持しているという特徴を持っている。さらに時間的な変遷を管理する仕組みや、コミットをメールで受信する仕組みなどがある。また、ローカルコンピュータ上にリポジトリを持つため、場所や時間、あるいはネットワーク接続の状態に関わらずコミットすることが出来る。[4]

2.4. GitHub とは

GitHub とは、GitHub.com により運営されている Git ホスティングサイトであり、Git リポジトリを利用してプロジェクトやソースコードの管理を行うバージョン管理システムを提供しているサービスである。GitHub は Git ホスティングサイトとしては最も多く利用されているサービスで、170 万を超える人が利用している。

また GitHub はソーシャルな機能が特徴で、プログラマー同士がコードの共有を行ったり、コードの公開をし合ったりしている。そして、GitHub ではソースコードだけでなく、画像やドキュメントなど、どんなファイルでもアップロードすることが出来、管理することが出来る。[4]

2.4.1. GitHub の基本用語

GitHub の基本用語について以下に記述する。

① Git リポジトリ

GitHub で提供するデータベースのようなものであり、論文に記述してあるリポジトリは Git リポジトリのことである。リポジトリの管理ユーザーが公開するユーザーの範囲を選択することができ、公開の場合は、誰でも閲覧することができる。だが、非公開に選択すると特定のユーザーのみしか閲覧することはできない。

② コミット (commit)

ファイルの変更履歴情報を閲覧したり、ファイルの変更を保存したりすることである。

③ 共有リポジトリ

チームメンバー間で共有できるリポジトリのことであり、ソースコードのメインバージョンが保存されている。

④ ローカルリポジトリ

作業者のコンピュータ上にあるリポジトリである。

⑤ インデックス

ローカルリポジトリへ反映する変更を一時的に保存しておく場所である。インデックスの内容は、commit によりローカルリポジトリへ反映される。

⑥ アドオン (add)

ソフトウェアに追加される拡張機能のことである。

⑦ 作業ツリー

ローカルリポジトリ上にある現在の作業ファイルである。作業ツリーの変更点は `add` によるインデックスに追加される。

⑧ ディレクトリ

フォルダのことであり、ファイルを分類・整理するための保管場所である。

⑨ フォロー

特定のユーザーをフォローすることが出来る。

⑩ フォロワー

ユーザーをフォローしているユーザー。

⑪ スター

Face book, mixi などの SNS で利用されている「いいね!」のようなもの。つけられたスターはカウントされ、スターのカウントが多いほど、他のユーザーから注目されていると認識できる。

⑫ リビジョン

ある期間内までの過去のプロジェクトデータやある程度まとまったプロジェクトデータを記録したものである。

⑬ ウィキ (Wiki)

その場にページが表示され、ドキュメントやコードがかかる場所である。

⑭ Organization

個人的に `GitHut` を使用している人が仕事などで `GitHub` を仕事用に使用したい場合にアカウントをもう 1 つ作成するのではなく、同じアカウントで会社用に使用するアカウントとして使用するものである。

2.4.2. GitHub の機能

GitHub は Git をサポートするために様々な機能を備えている。以下には機能について記述する。

- フォーク (Fork)

GitHub 上で公開されているリポジトリを自分のリポジトリとして複製する動作。既存のリポジトリに対して独自の変更を加えたい場合に利用する。

- ライト (write)

フォークしたときにコピーしたオリジナルのリポジトリのデータに書き込みをすること。

- プルリクエスト (Pull Request)

フォークしたリポジトリに対して変更を加へ、それをフォーク元のリポジトリに取り込んで貰う様にリクエストを送ること。

- マージ (merge)

プルリクエストした人がその人のリポジトリに対して行った変更を自分のリポジトリにも取り入れること。

- イシュー (Issue)

1つのタスクを1つの Issue に割り当てて、データの監視や管理を行えるようにするための機能。1つの機能変更や修正などに対して1つの Issue が割り当てられるため、Issue を見れば、そのタスクの変更や修正に関することがすべてわかるよう管理できるのである。また、Issue にタグやマイルストーンをつけることも可能である。タグ機能は初期の設定の場合では、「バグ」、「重複」、「強化」、「無効」、「質問」が設定されている。タグの種類は増やすことも可能である。また、Issue には、「Open」と「Close」機能があり、Issue を受信した人は受信した Issue を拝見したら Open をクリックし、拝見し終わったら、Close をクリックすることで、Issue を発行したユーザーに拝見し終わったことを通知することができる。

- ウォッチ機能

他の人のデータを見ることが出来る。他の人の進捗状況やプロジェクトの内容を閲覧できる。

- グループ機能

特定のアカウントユーザーで構成し、特定のユーザー同士でリポジトリを共有したりすることができる。グループ機能を公開の状態にすることで、グループ以外のユーザーも閲覧できるようになる。非公開の状態にするとグループで決められたユーザー以外は閲覧やリポジトリを操作することができないように設定することができる。

- 検索機能

検索機能は「検索ウィンドウ」に検索ワード（ユーザー名やプロジェクト名，コードなど）を入力するとそれに関連した情報を表示することができる．

参考文献

- [1] 濱野純.入門 Git, 秀和システム. (2009-9-25)
- [2] 片岡巖. WEB+DB PRESS Vol.69, 技術評論社. (2012-7-25)
- [3] GitHub. GitHub Developer. <http://developer.GitHub.com/v3/issues/>.(参照 2014-8-10).
- [4] 久保孝樹.「チケットを活用するオープンソフトウェア開発の実態調査」 (2013-1-30).

第 3 章

Big Data について

3.1. 本章の構成

本章では本研究で GitHub にあげられているデータを収集するにあたり, 利用する Big Data 処理技術についての基本知識, 本研究で使用する Google BigQuery について記述していく.

3.2. Big Data とは

パソコン, スマートフォンが普及し「ビッグデータ」という言葉が流行することからもわかるように, 私たちは膨大な情報を生み出しながら生活している. Google や Yahoo! に寄せられる大量の検索クエリや, Twitter, Facebook などの SNS に投稿される文章や画像, 動画, スマートフォンを利用するサービス等で収集される位置情報データ, 防犯カメラで記録される人間の表情や動きのデータなどの膨大な量のデータを指す.

ビッグデータとは一般的にペタ (1,000 兆) バイト級 (表 1) のデータ量と言われている. このような数値的定義もあるが, ペタバイト以下であればビッグデータでは無いという訳でもなく, 本質的には, 「従来の手段では管理しきれない規模のデータ」を指す.

表 1 単位接頭辞と十進表記 引用文献[1]

10^n	接頭辞	記号	漢数字表記 (命数法)	十進数表記	分類
10^{21}	ゼタ (zetta)	Z	十垓	1 000 000 000 000 000 000 000	ビッグデータ
10^{18}	エクサ (exa)	E	百京	1 000 000 000 000 000 000	ビッグデータ
10^{15}	ペタ (peta)	P	千兆	1 000 000 000 000 000	ビッグデータ
10^{12}	テラ (tera)	T	一兆	1 000 000 000 000	
10^9	ギガ (giga)	G	十億	1 000 000 000	
10^6	メガ (mega)	M	百万	1 000 000	
10^3	キロ (kilo)	K	千	1 000	

3.2.1. 4V による Big Data の定義

IBM 社による Big Data の定義で 4V というものがある。4V とは、容量 (Volume)、種類 (Variety)、頻度・スピード (Velocity)、正確さ (Veracity) から構成されている。

◆容量 (Volume)

ビッグデータの特徴である容量の巨大さを指す。企業内外にはデータが溢れており、数テラバイトから数ペタバイトにもおよぶ。またデータが増大することによる計算量も非常に膨大となる。

◆種類 (Variety)

ビッグデータは企業システムで通常扱っているような顧客情報や販売データ、経理データ、在庫データなどの構造化データであるとは限らない。テキスト、音声、ビデオ、ページ遷移、ログファイルなどのさまざまな種類の非構造化データも存在する。

◆頻度・スピード (Velocity)

今この瞬間にも、ものすごい頻度で RFID などの IC タグやセンサーなどからデータが生成されている。昨今の変化の著しい市場環境では、これらのデータによりリアルタイムに対応したものを求められている。

◆正確さ (Veracity)

データの矛盾、曖昧さによる不確実性、近似値を積み重ねた不正確さなどを排除して、本当に信頼できるデータが意思決定には重要である。

以上が IBM による 4V の定義であるが、容量 (Volume) については最初に記述したように、必ずしもペタバイト以上でなければならないとは考えていない。

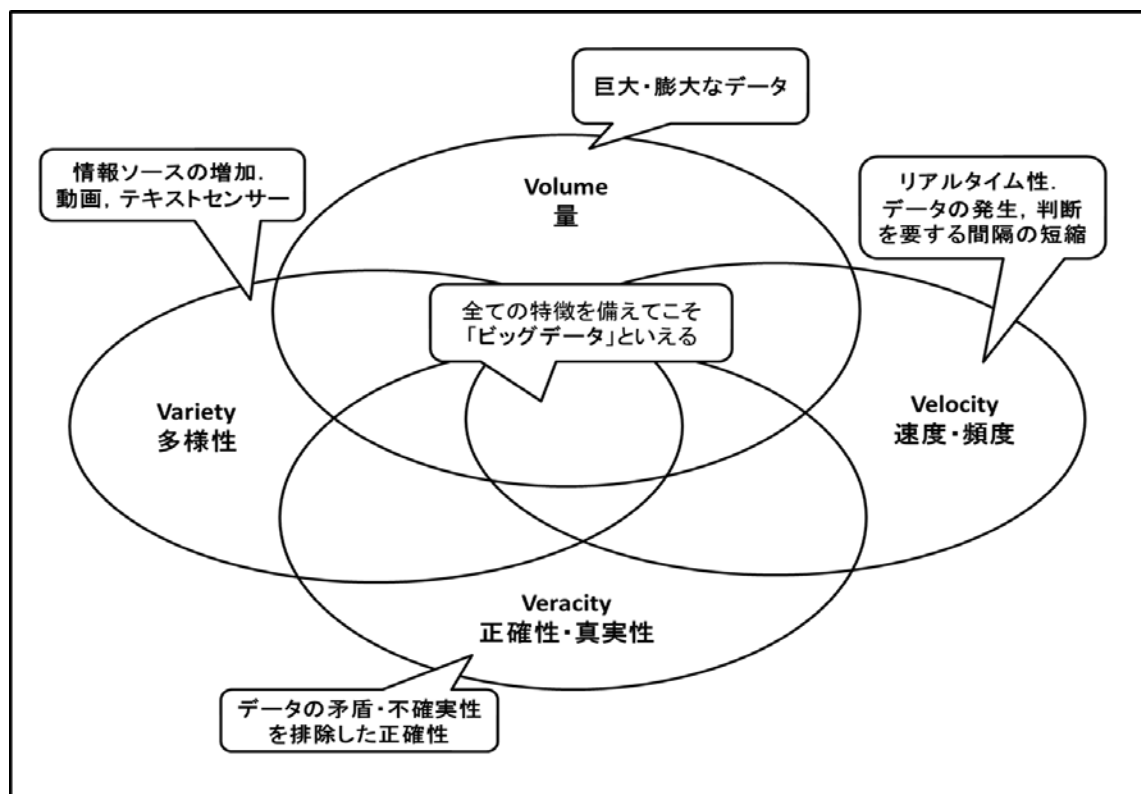


図 2 ビッグデータの 4V

3.2.2. 3V でのビッグデータの定義

3V で表す場合は、容量 (Volume)、種類 (Variety)、頻度・スピード (Velocity) の 3 つになり、正確さ (Veracity) は含まれない。確かにビッグデータには正確でないデータが混在することもあり、例えば Twitter などの SNS データには、冗談やデマ情報の書き込みなども混じっており、センサーなどでも故障によるノイズが混じることもあります。データ量が少ない場合には、外れ値として手作業で除去することも可能だがいわゆるビッグデータと言われている大量のデータの場合は、手作業によるデマ情報やノイズなどの除去はほとんど不可能である。

しかし、ビッグデータで収集するデータは殆どが生データであるという特徴がある。正確さをどう定義するかにもよるが、センサーからの入力データなどは生データそのものだが、SNS からの入力データなども生データであり、その意味では正確なデータといえる。つまり、編集などで手が加わっていないデータであり、またそこで発せられるメッセージはその人が、その人の環境により制約などを感じるデータでは無いからだ。これは、勤務する企業・組織内で作成する報告書など対比して考えるとわかりやすいだろう。

3.3 ビッグデータ処理のパターン

下記の表に、3つの処理パターンの特性を簡潔にまとめたものを記述する。

表 2 ビッグデータ 3つの処理パターン 引用文献[1]

	バッチ処理	インタラクティブクエリー処理	ストリームデータ処理
実行タイミング	ユーザー指定と定期的実行	ユーザー指定と定期的実行	常時連続実行
処理単位	蓄積データをバッチで一括処理	蓄積データをバッチで一括処理	少数のフローデータ処理
実行時間	分～時間	秒～分	ミリ秒～秒
処理モデル	MapReduce	クエリ・OLTP	ストリーム処理

表 2 のようにビッグデータの処理パターンには、「バッチ処理」、「インタラクティブクエリー処理」、「ストリームデータ処理」の 3 種類のパターンがある。

バッチ処理では蓄積データをバッチで一括処理だが、これは Google 検索用に開発された MapReduce 処理を利用した Hadoop が代表的である。しかし、Hadoop はビッグデータ処理用として開発されたものではないので、処理結果作成に時間がかかるという欠点である。

インタラクティブクエリー処理は、蓄積された大容量データをオンラインクエリなど使用して一括解析処理するものである。インタラクティブクエリー処理では蓄積されたビッグデータを数秒から数分で実行する。本研究で使用する BigQuery の処理エンジン Dremel はその処理スピードの速さに特徴がある。

ストリームデータ処理は大量発生する実世界データを逐次に時系列処理する技術である。データ発生時にあらかじめ登録したシナリオに従って集計・分析に必要なデータを抽出し、データ処理を行う。このように逐次時系列でデータ処理できることから、最新の情報、その中での特異な値の発生などに対してリアルタイムに対応するシステムを構築できることが特徴で IoT への応用に最適な処理方法といえる。[1]

3.3.1 バッチ処理

最初に MapReduce で代表される、バッチ処理の特性を見る。

数十年前のメインフレームは、主記憶が数百キロバイト、価格は数千万円程度だった。これを現在の PC（主記憶数ギガバイト、価格は 10 万円程度）と比べた場合、価格性能比では約 100 万倍にもなる。また、CPU 処理スピードと、ネットワークの帯域幅についてはそれぞれ、主記憶と類似の性能向上を遂げてきている。[1]

このようなことは、コンピューター関係以外の業種では全く例を見ない群を抜く性能向上であり、この急激なプラットフォームの真価がクラウドコンピューティングやそのうえで実行されるビッグデータ処理などを可能にしている。

ただしこれはクラウドなどに限ったことではなく、ITの世界ではこれまでも短いタイムスパンで新しいテクノロジー・ブレイクスルーやビジネスモデルが出現してきており、これはプラットフォームやネットワークの進歩に依存している部分が多くある。言葉を変えれば、これらの新しい発想はその時点でのプラットフォーム性能で初めて成り立つものであり、これをわずかも前の世代に思いついたとしても、実現不可能である場合が多い。

このようにクラウドなどの先端ITシステムは、現在のそしてこれからも進化を続けるはずのプラットフォームやネットワークに依存したものである。[1]

3.3.2. インタラクティブクエリー処理

インタラクティブ処理は、本研究で使う処理方法である。BigQueryによる説明を記述す。

Google がリリースするソフトウェアツールには、もともと Google が社内使用の目的で開発していたものも少なくなく、BigQuery もそれに当てはまる。Google も当初は社内使用でも MapReduce を使用してビッグデータ処理を行っていたが、バッチ処理による結果生成の遅延や処理を行うための準備の煩雑さなどから、それに代わるツールとして開発されたのが BigQuery である。

BigQuery はデータの入力または JSON フォーマットのファイルから直接行うことができ、Cloud Storage からのデータロードも可能である。ビッグデータの解析や絞込みは RDB (Relational Database) の SQL に類似したクエリ言語を使用し、UI 画面や PC のコマンドラインから容易にデータ検索を行うことができ、また Excel を使用し検索・表示を行うことが出来る。[1]

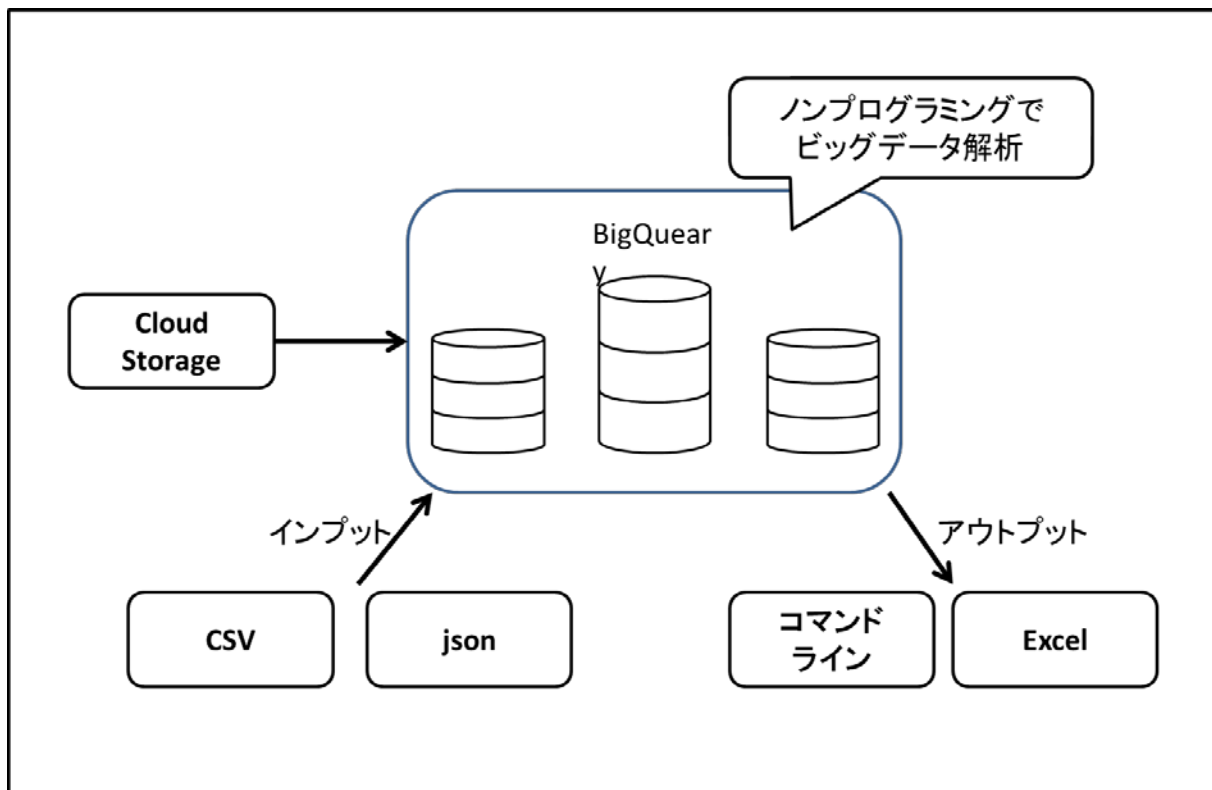


図 3 BigQuery の入出力 引用文献[1]

BigQuery を使用したインタラクティブクエリ処理では、マウス操作と簡単なキー入力によってすべての操作を行うことができ、また結果出力も数秒から数分以内で得ることができる。したがって、何かこのデータを解析したいと思った時、その場で気軽に行えるという点が一番の特徴として挙げられる。

また、BigQuery でのビッグデータ解析は大量のデータを超高速で行えるのも大きな特徴で、例として 15 億行のデータに対する比較的複雑な集計問い合わせが 20 秒から 25 秒で返ってきたというユーザの実行結果もある。BigQuery ではインデックスを作成する必要がなくデータをロードするだけでこのような高速クエリが実行でき、またキャッシュはトグルボタンから有効無効を切り替えられるが、キャッシュを使っていなくても同様の結果が得られる。[1]

3.3.3 ストリームデータ処理

ストリームデータ処理は、大量発生する時系列のデータ（ストリームデータ）をリアルタイムに逐次処理する技術。

ストリームデータ処理は、データ発生時に、あらかじめ登録したシナリオにしたがって集計・分析に必要なデータを抽出し、データ処理を行う。その際、分析対象データをメモリー上で処理する「インメモリデータ処理技術」により、高速なデータ処理を実現している。これらの技術によって、大量データを高速に、かつリアルタイムに処理でき、たとえば、株価のテクニカル指標やランキング情報から売買をリアルタイムに自動判定する,といったシステムに大変有効だ.ほかにも,リアルタイムの在庫管理や,不正操作の監視を行うシステムなど,多くの利用目的が考えられる.[1]

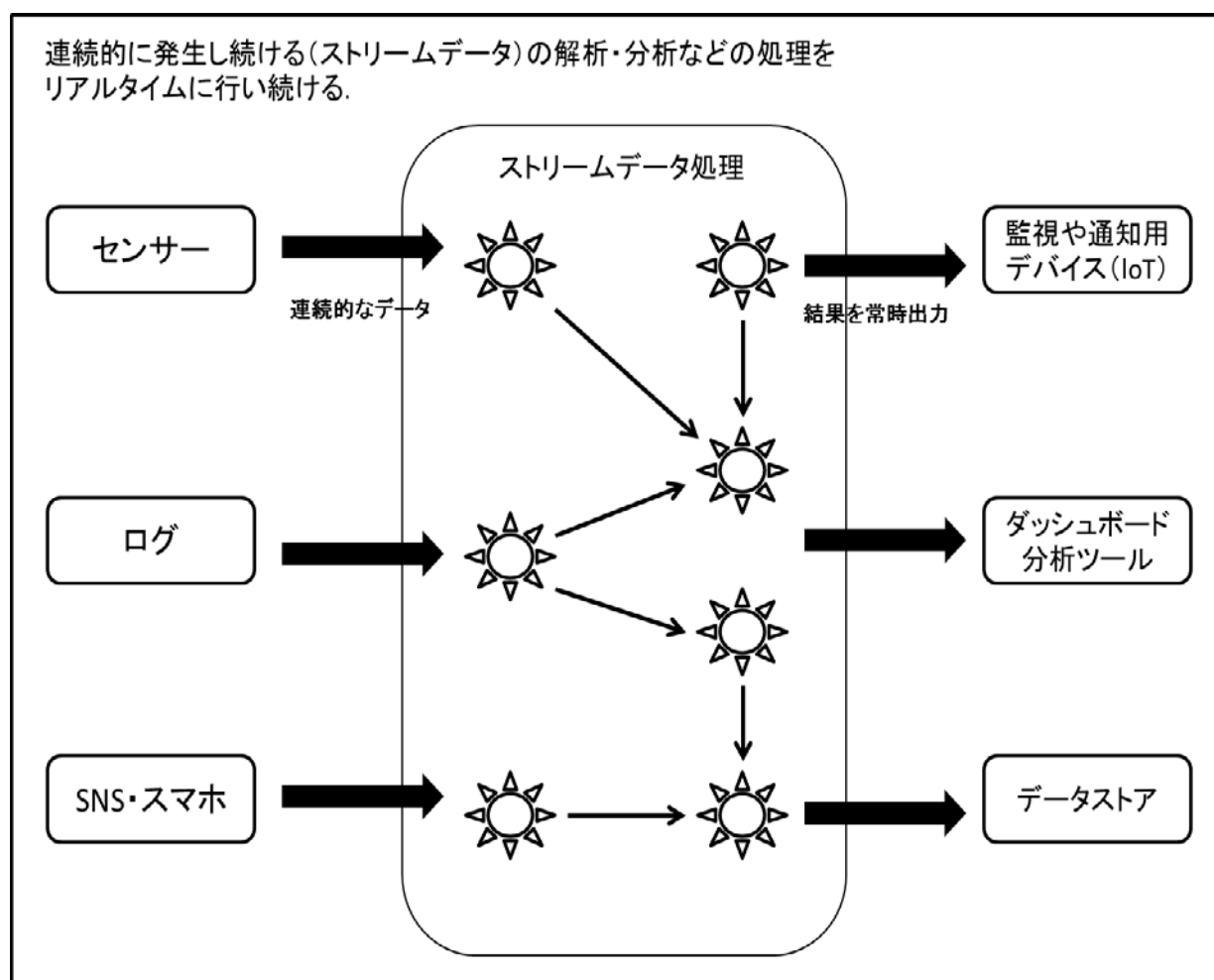


図 4 ストリームデータ処理 引用文献[1]

3.4. Google BigQuery

Google BigQuery の基本構成と特徴について記述する.

3.4.1 基本的なシステム構成

BigQuery を使用する場合の基本的なシステム構成はシンプルである. Web UI からの操作以外はすべて Excel と CSV ファイルを入出力に使用する図を以下に記述する.

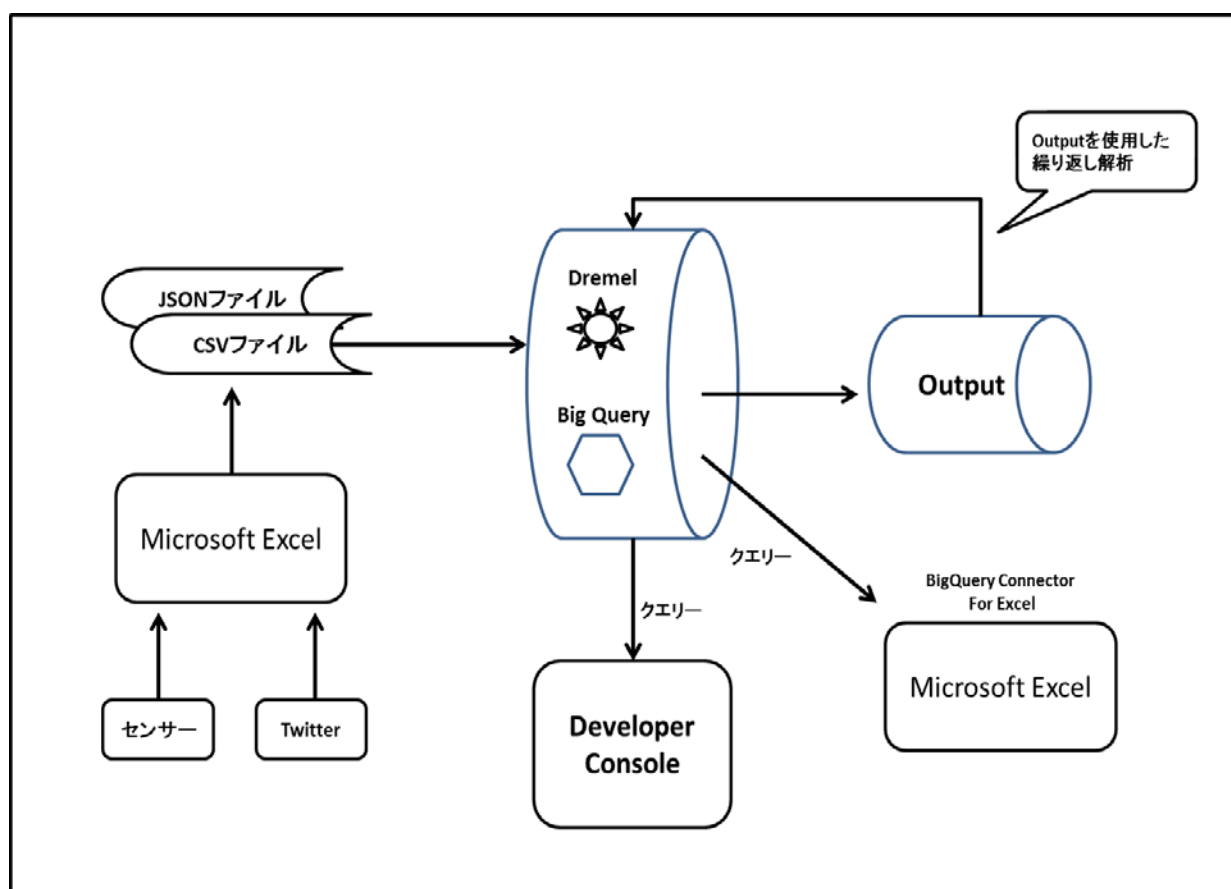


図 5 BigQuery 処理パターン 引用文献[1]

3.4.2 BigQuery を使用する

BigQuery にサインアップする。
最初にサインアップを行う。

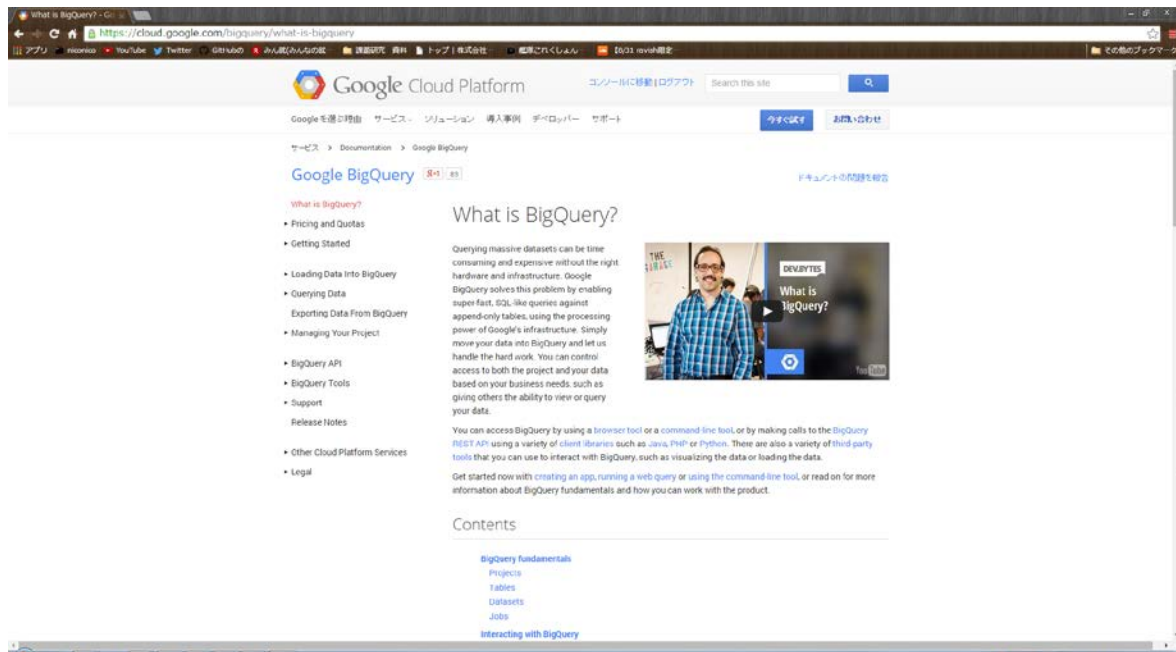


図 6 Google BigQuery 公式画面 (<https://cloud.google.com/bigquery/what-is-bigquery>)

サインアップでは、図 6 の BigQuery 公式画面を表示する。画面上部の「今すぐ試す」からサインアップを行う。「今すぐ試す」を押すと下記（図 7）のような画面が表示される



図 7 サインアップの画面

サインアップすると Google Developers Console の「Projects」画面（図 8）が表示される。

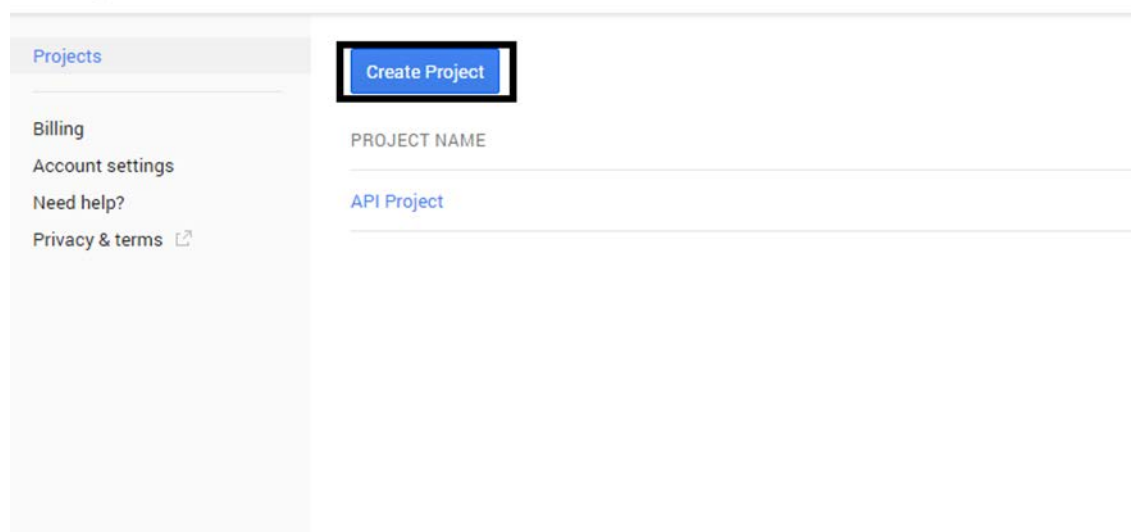
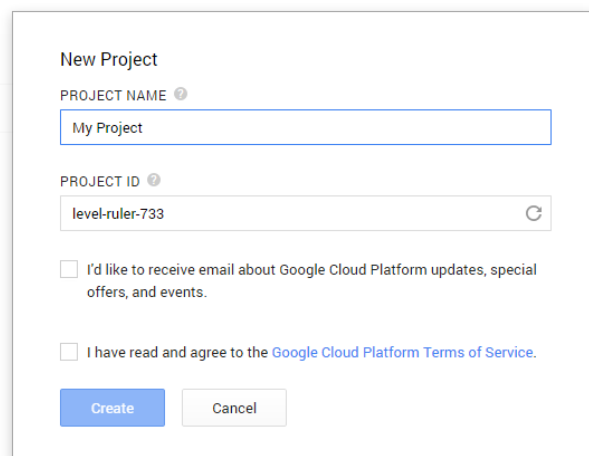


図 8 Projects 画面

私の場合,すでにプロジェクトが登録されていますが,初めての場合は「PROJECT NAME」以下に何も表示されていないはずである.画面上部の「Create Project」をクリックして表示される「New Project」サブ画面から,プロジェクト名 (Project name) とプロジェクト ID (Project ID) を入力して,新規のプロジェクトを作成する.プロジェクト名とプロジェクト ID は英字または英数字でなんでも構わないが,プロジェクト名はプロジェクト内容に関連した名前にし,プロジェクト ID はその短縮形にするのが良い.



New Project

PROJECT NAME ?

My Project

PROJECT ID ?

level-ruler-733

☐ I'd like to receive email about Google Cloud Platform updates, special offers, and events.

☐ I have read and agree to the [Google Cloud Platform Terms of Service](#).

Create Cancel

図 9 プロジェクト作成サブ画面

次に、作成された「PROJECT NAME」のリンクをクリックして表示される画面の左ペインで APIs&auth 下の APIs をクリックすると図 10 の画面表示になるので、BigQuery API の status を画面右端のボタンをクリックして「ON」にする。ただし、この設定は私の研究の場合なのでノン・プログラミングで BigQuery を使用することに徹するのであれば、必ずしもやる必要はない。



図 10 プロジェクト画面

GitHub Archive と連動させるために左ペインの Big Data から BigQuery を選択する。

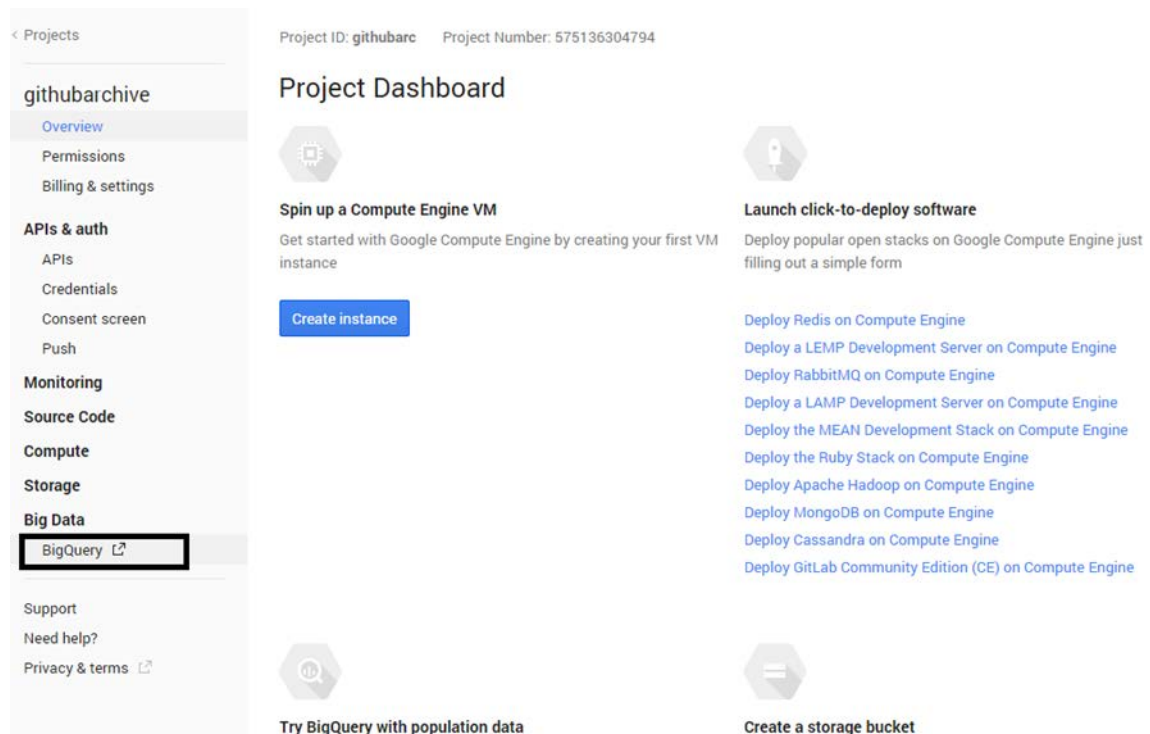


図 11 BigQuery 選択

図 12 の画面から BigQuery の操作を行っていく。BigQuery では、データの入れ物として「Dataset」を作成し、その中に「Table」を作成する。Table は 1 つの Dataset に複数作成することができる。

図 12 の左ペインでプロジェクト名の右にある▽のマークをクリックすると、プルダウンメニューが表示される。プルダウンメニューで「Switch to project」を選択する。

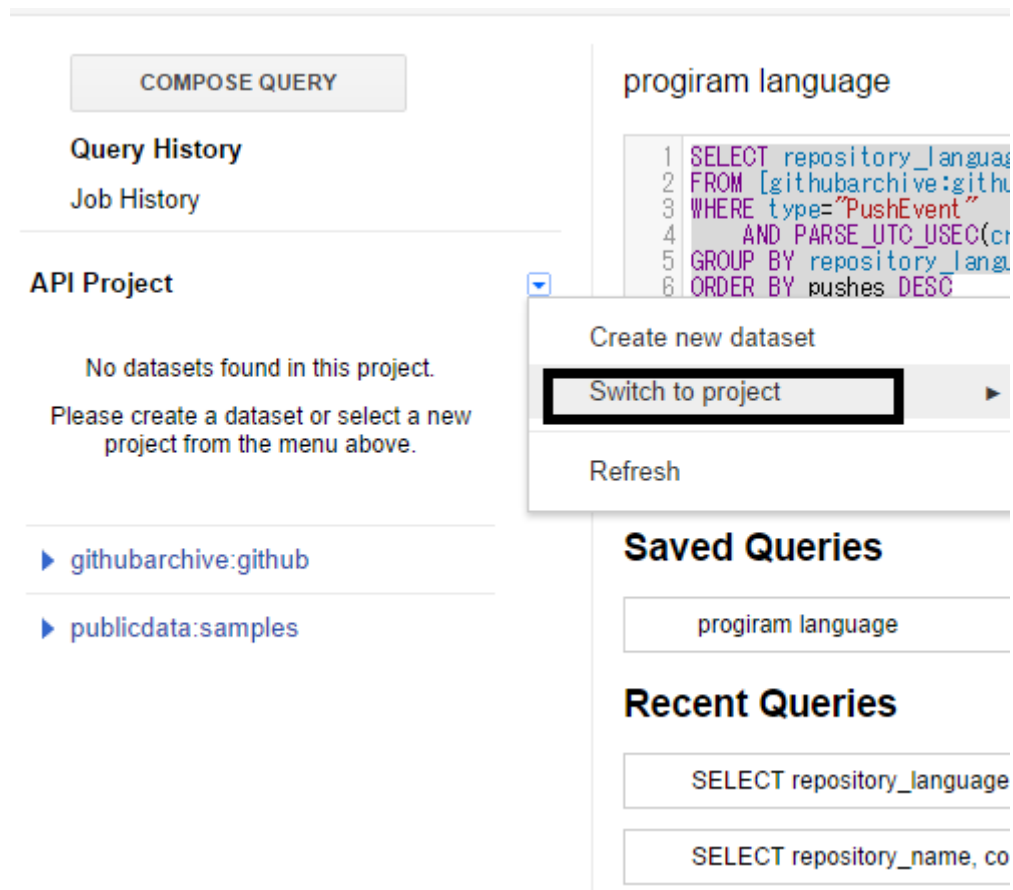


図 12 BigQuery 画面から dataset 作成

Switch to project 選択後図 13 のような選択肢が表示されるので「Display project」を選択する。

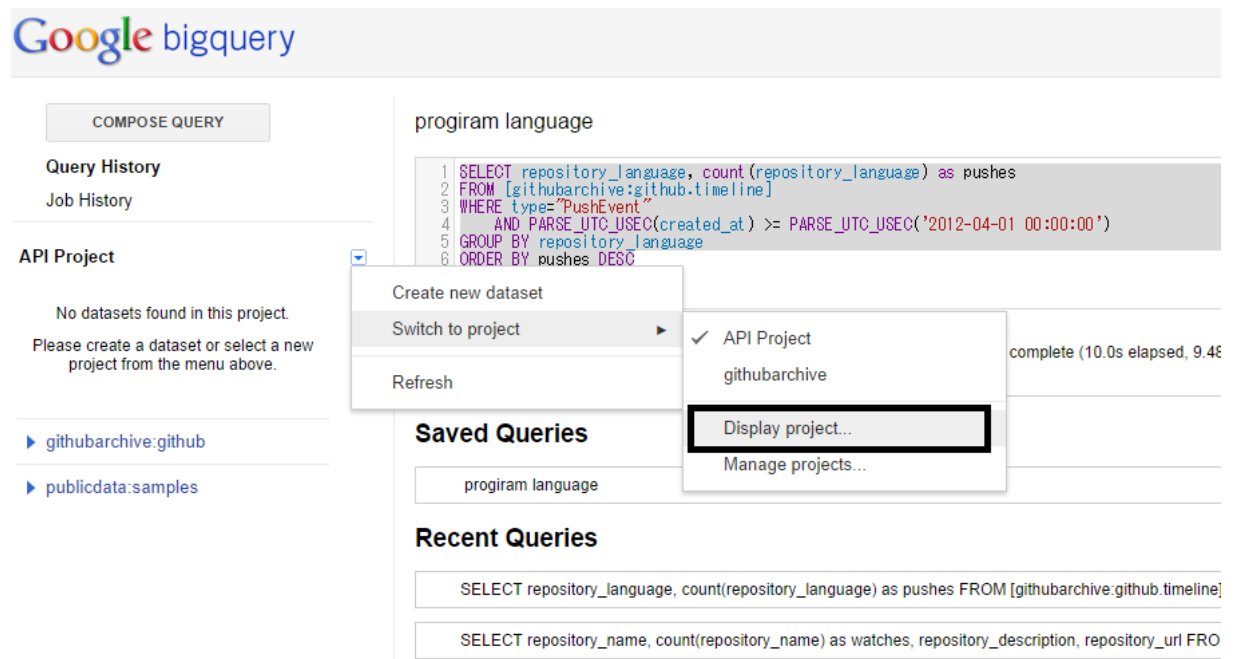


図 13 Display project

図 14 のようなダイアログが表示されたら Project ID（ここでは githubarchive）を入力する。入力後「OK」ボタンをクリックする。

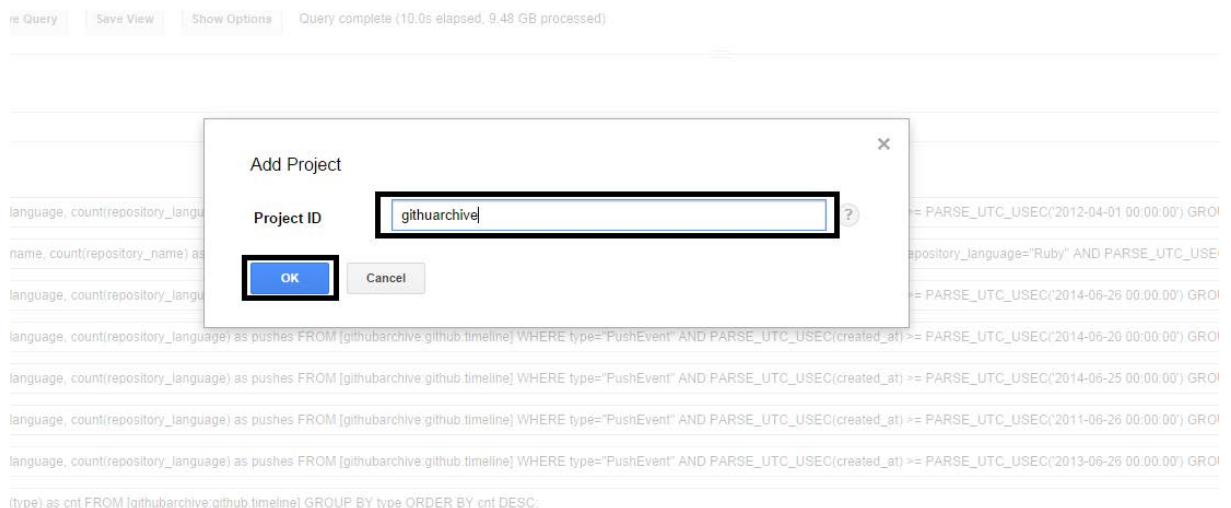


図 14 Project ID 指定

完了後、左ペイン上部にある「COMPOSE QUERY」を選択する。

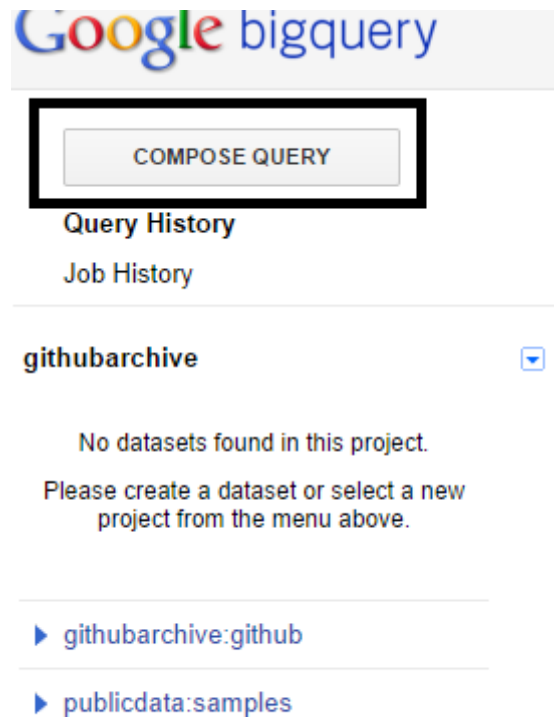


図 15 COMPOSE QUERY の選択

選択後図 16 のような画面になるので黒枠の部分にコードを入力する。



図 16 コードの入力

コードの例として、GitHub で 2012-04-01 から使われているプログラミング言語を解析するコードを以下に記述する。

```
SELECT repository_language, count(repository_language) as pushes
FROM [githubarchive:github.timeline]
WHERE type="PushEvent"
      AND PARSE_UTC_USEC(created_at) >= PARSE_UTC_USEC('2012-04-01 00:00:00')
GROUP BY repository_language
ORDER BY pushes DESC
```

コード入力後、New Query の下にある「RUN QUERY」をクリックする。



図 17 コードの実行

New Query

```

1 SELECT repository_language, count(repository_language) as pushes
2 FROM [githubarchive:github.timeline]
3 WHERE type= 'PushEvent'
4 AND PARSE_UTC_USEC(created_at) >= PARSE_UTC_USEC('2012-04-01 00:00:00')
5 GROUP BY repository_language
6 ORDER BY pushes DESC

```

RUN QUERY Save Query Save View Show Options Query complete (5.3s elapsed, cached)

Query Results 9:35pm, 14 Oct 2014

Row	repository_language	pushes
1	JavaScript	21864210
2	Java	14148511
3	Python	11045096
4	Ruby	10506238
5	PHP	9769443
6	C++	6574131
7	C	5915965
8	CSS	5770702
9	Shell	3653936
10	C#	2774953
11	Objective-C	1978772
12	Perl	1514504
13	VimL	1234261
14	Scala	798215
15	CoffeeScript	774351
16	Go	772313
17	R	612397

参考文献

- [1]Google BigQuery ではじめる自前ビッグデータ処理入門 2014-10-10
- [2]IMB ビッグデータ. <http://www-03.ibm.com/software/products/ja/category/bigdata>(参照 2014-8-28)
- [3]オラクルデータベースインサイダー. https://blogs.oracle.com/dbjp/entry/bigdata_000244(参照 2014-8-28)
- [4]IT ジャーナリスト星暁雄の"情報論"ノート 2012-06-01
<http://hoshi.air-nifty.com/diary/2012/06/githubgoogle-bi.html> (参照 2014-8-28)
- [5]WORKSIGHT 2013-07-22 <http://www.worksight.jp/issues/289.html> (参照 2014-8-28)
- [6]ニッセイ基礎研究所 ビッグデータで何がわかるか 2013-11-08
<http://www.nli-research.co.jp/report/report/2013/11/repo1311-c3.html> (参照 2014-8-28)

第 4 章

開発・調査

4.1. 本章の構成

本章では調査対象の記述，GitHub からデータを収集するためのプロジェクトである GitHub Archive の説明，Google BigQuery で使うコードの設計の解説を記述する．

4.2. 調査対象

ここでは，本研究において調査対象となるデータの記述，およびそれらのデータを抽出する対象の期間を記述する．

4.2.1. 調査対象データ

本研究では，2012 年 2 月 12 日から 2014 年 11 月 18 日までの GitHub であげられているオープンソースソフトウェア開発の使用されているプログラミング言語，Fork 数，プロジェクト数を Google BigQuery を利用し GitHub Archive から抽出する．

4.3. 調査方法

本研究に必要である調査環境の構築を記述し，本調査で目的とするプログラミング言語，Fork 数，プロジェクト数を抽出するためのプログラムの解説，抽出したデータからグラフを書くために必要なプログラムの解説を行う．

4.3.1. 調査環境構築

本研究では，データを集めるのには，ビッグデータの処理技術を使うが集めたデータから調査のために必要なデータのみを抽出するのに MySQL を利用する．MySQL を選択した理由は，自由に無料で使うことが出来ること，広く普及されているのでウェブ上や書籍で情報が集めやすいためである．

4.3.2. Windows での導入

Windows では、XAMPP を導入する。

XAMPP のウェブサイト (<https://www.apachefriends.org/jp/index.html>) から、Windows 版のインストーラをダウンロードし、実行する (図 19)。



図 19 Windows 版の XAMPP をダウンロード

XAMPP Control Panel (図 20)で Apache を「Running」の状態にし、ウェブブラウザから <http://localhost/> にアクセスし、ページが表示されればインストール完了。

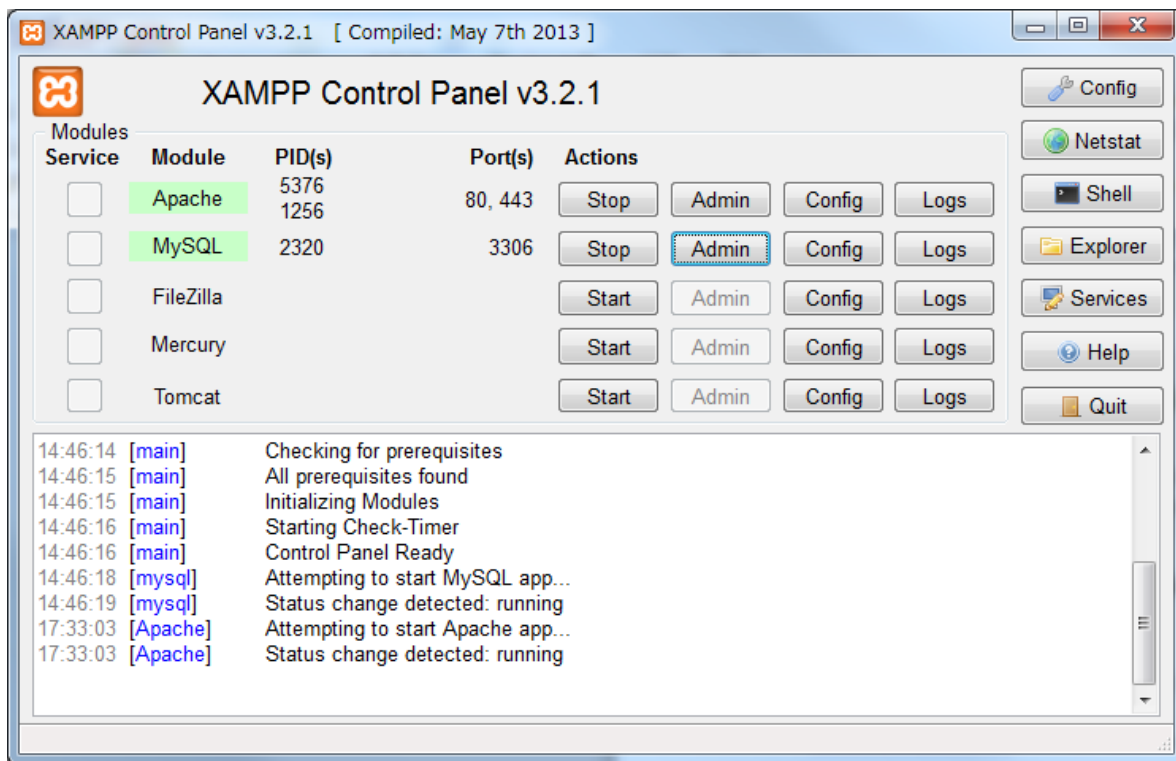


図 20 XAMPP Control Panel

4.3.3. おおもとになるデータの抽出

GitHub からデータを抽出する方法を記述する。

①以下の SQL を BigQuery で処理する。

```
SELECT repository.url, repository.forks, repository.language
FROM [publicdata:samples.github_nested]
```

処理した結果を「Save as Table」で自分のプロジェクトに保存（図 21）

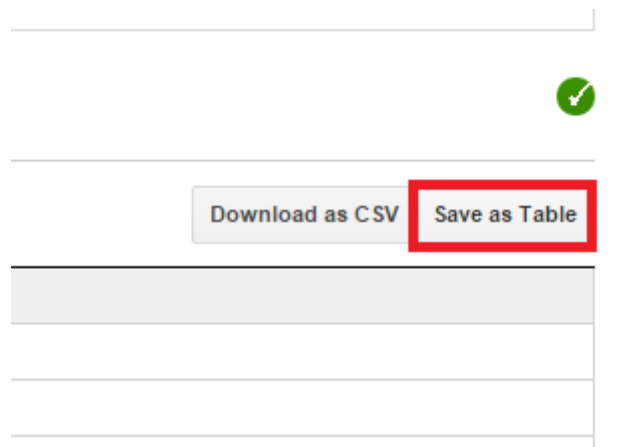


図 21 BigQueary での処理

②保存されたテーブルを Export する.

エクスポートしようとする量が多すぎるために課金をしなくてはならなくなったため矢吹先生にお願いして Export していただいた.

以上で BigQueary で GitHub にあげられているプログラミング言語, Fork 数, プロジェクトの URL の抽出は完了である.

4.4. MySQL

MySQL の操作, そこで使用した SQL 文の説明を行う.

4.4.1. MySQL への接続

XAMPP Control Panel から MySQL を Start させ, Windows のホームから「コマンドプロンプト」を起動する. コマンドプロンプトから MySQL に接続するためのコマンドは,

「c:/xampp/mysql/bin/mysql --local-infile -uroot」最初に利用するときは, ユーザー名は“root”となる.

MySQL に接続すると, プロンプトが「mysql>」となる. このプロンプトのもとでは, SQL というプログラミング言語を用いて RDBMS を操作する.

図 22 MySQL の起動

4.4.2. データベースの作成と削除

データベースの作成は、CREATE DATABASE 分で行う。

CREATE DATABASE データベース名 DEFAULT CHARACTER SET utf8;

この研究では、mydb という名前で作業を進める。まず、次のような SQL 文でデータベースを作成する。

```
mysql> create database mydb default charset=utf8;
```

SHOW DATABASESE 文で、MySQL が管理しているデータベースを確認できる。

データベース mydb の利用を開始するためには USE 文を使用する。

```
mydb> USE mydb
```

データベースの削除には DROP DATABASE 文で行う。先に作ったデータベース mydb を削除する SQL 文は以下の文を用いる。

```
mysql> DROP DATABASE mydb;
```

データベース作成後、その中にテーブルを作成する。テーブルの作成には、CREATE TABLE 文を使用する。SQL 文は以下の文を用いることで本研究に必要なテーブルの作成が可能。

```
mysql>create table projects (  
  id int auto_increment primary key,  
  url varchar(200),  
  forks int,  
  lang varchar(100),  
  key (lang)  
);
```

失敗した場合のテーブルの削除には、「DROP TABLE テーブル名」で削除が可能。テーブルの作成は一回でよい。

4.4.3. データのインポート

上記で作成したテーブルに GoogleBigQuery で抽出した GitHub のデータをインポートする方法について記述する。MySQL への接続時に、「mysql-local-infile ...」としないと LOAD 文が使えない場合もあるので注意が必要。

```
mysql>load data local infile "forks.csv" into table projects
fields terminated by ','
(url,forks,lang);
```

正しく読み込めたかの確認する際、安易に「SELECT * FROM projects;」などとしないように注意が必要。大量のデータを扱っているので表示に時間がかかってしまうので LIMIT 節を使って、取得する行数を制限して確認すると良い。

```
mysql>select * from projects limit 10;
```

4.4.4. インデックス

インデックスをすることにより検索を早めることが出来る。本実験ではデータ量が非常に多いのでインデックスを作り必要なデータを集める速度を上げる必要がある。本実験では一般的なインデックスを作成する。CREATE TABLE 文中で、「KEY (カラム名)」あるいは「INDEX (カラム名)」のように記述して作成する。対象が文字列の際には「カラム名 (長さ)」として、インデックスに利用する長さを指定する。

4.4.5. SELECT 文の詳細

本研究で使用する SELECT 文の詳細を説明する。

- SELECT 取り出したいデータ
- FROM 対象テーブル
- WHERE 対象を限定する条件
- AVG AVG(expr)は expr の平均を返す
- COUNT COUNT(*)は該当レコード数を返す。COUNT(expr)は expr のうちで、NULL でないものの数を返す。COUNT(DISTINCT expr, [expr...])は expr のうちで、NULL でないものの数を返す。
- MAX MAX(expr)は expr の最大値を返す。

- MIN MIN(expr)は expr の最小値を返す.
- 標準偏差 STDDEV_POP(expr)=STDDEV(expr)は母集団標準偏差を, STDDEV_SAMP(expr)は標本標準偏差を返す. 本研究では母集団標準偏差を利用する.
- SUM SUM(expr)は expr の和を返す.
- 分散 VAR_POP(expr)は母集団標準偏差, VAR_SAMP(expr)は標本偏差を返す.
- LIMIT LIMIT は, 取得する行数を制限する.
- DISTINCT DISTINCT は, 結果から重複を取り除く.
- GROUP BY GROUP BY はグループごとに関数値を求める.
- ORDER BY ORDER BY 節は結果を並び替える.
- DESC ORDER BY 節の最後に DESC を入力することで降順にすることが可能
- NOT(expr="...") 「“”」に囲まれているものを除いた結果を求める.
- OUTFILE 求めた結果をエクスポートする際に使用する.
mysql>SELECT * INTO OUTFILE 'ファイル名' FROM テーブル名;
上記のような形で使用する. ファイル名をフルパスで指定しない場合 MySQL のデータディレクトリにファイルが作成される.
- KEY CREATE TABLE 文中で KEY(カラム名)あるいは「INDEX (カラム名)」のように記述して使用する.
- DESC ORDER BY 節の最後に DESC を入力することで降順にすることが可能
- DELETE データを削除する際に使用する. WHERE 節を省略するとすべてのデータが削除される.

4.4.6. SQL

実際に使用した SQL を記述する.

使用したテーブルは以下 (表 3) のようなカラムで構成されている.

表 3 テーブル定義

Field	Type	Null	Default	Extra
id	int(11)	NO	0	
url	varchar(200)	YES	NULL	
forks	int(11)	YES	NULL	
lang	varchar(200)	YES	NULL	

●データベースの作成

```
mysql>create database mydb default charset=utf8;  
load
```

コマンドプロンプトで MySQL 起動後にデータベースを作成する. この作業は最初の 1 回のみ.

●データベースの使用

```
mysql> use mydb
```

データベース作成後にこのコマンドを入力しデータベースの使用を許可する. この作業は MySQL 起動時に毎回行う.

●テーブル作成

```
mysql>create table project3 (  
  id int auto_increment primary key,  
  url varchar(200),  
  forks int,  
  lang varchar(100),  
  key (lang)  
);
```

このコマンドを入力してデータベース内にテーブルを作成する. テーブルの構成要素は, カラム名, データ型, 性質の順に列挙する.

最後に「Key」を置くことで同時にインデックスの作成も行う.

- データのインポート

```
mysql>load data local infile "forks.csv" into table project3  
fields terminated by ','  
(url,forks,lang);
```

作成したテーブルに BigQyeary で抽出したデータをインポートする.

- データのインポートに成功したかの確認作業

```
mysql>select * from project3 limit 10;
```

実際にデータのインポートに成功したかの確認作業. リミットを 10 に設定することで大量のデータを検索するのを止める.

- 1 行目削除

```
mysql>delete from project3 where id=1;
```

先ほどの確認作業で 1 行目に不備があったため 1 行目を削除する SQL を使用した.

- プロジェクト数

```
mysql>select lang, count(*) from projects where not(lang = "")group by lang;
```

使用されているプログラミング言語が不明なものを除く言語ごとのプロジェクト数を表示させる.

- 平均

```
mysql> select lang, avg(forks) as avgfork from project3 where not(lang = "") group by lang order by  
avgfork;
```

使用されているプログラミング言語が不明なものを除く言語ごとの平均.

- 分散

```
mysql>select lang, VAR_POP(forks) as VAR_POPfork from projects where not(lang = "") group by lang  
order by VAR_POPfork;
```

使用されているプログラミング言語が不明なものを除く言語ごとの分散.

- エクスポート

```
mysql>select distinct url,forks into outfile 'c:/work/test.csv' from project3;
```

異なる URL 別の Fork 数のデータを CSV 形式でエクスポートする.

```
mysql>select distinct url,forks into outfile 'c:/work/JavaScript.csv' from project3 where lang='JavaScript';
```

言語別の Fork 数のデータを CSV 形式でエクスポートする. SQL 文の lang='JavaScript'を他の言語名を書くことで他の言語の Fork 数を取ることもできる.

- プロジェクト数, 平均, 分散

```
mysql> select lang, count(distinct url) as n, avg(forks),stddev_pop(forks) from project3 group by lang order by n desc;
```

言語ごとのプロジェクト数, 平均, 分散を一回で表示する. Group 以下の文を取り除くことで全体のプロジェクト数, 平均, 分散を出すことが出来る.

以上の方法を用いて GitHub のデータを引きだし平均, 分散, 標準偏差, ヒストグラムで比べ, 調査した.

参考文献

[1]佐久田博司, 矢吹太朗, Web アプリケーション構築入門, 第 2 版, 森北出版株式会社, 2014-2-20.

第 5 章

調査結果・考察

5.1. 本章の構成

本章では前章で記述した調査方法を用いて、GitHub 内のプロジェクトデータを集め調査した結果を記述し、その結果に対する考察を記述していく。

5.2. 調査結果

実際に調査を行い得られた平均、標準偏差、分散、ヒストグラムと解説を以下に記述していく。

5.2.1. 全体の比較

表 4 プロジェクト数, 平均 fork 数, 標準偏差

言語	プロジェクト数	平均 Fork 数	標準偏差
全体	209405	62.7892	353.0447
JavaScript	37795	127.6464	571.9613
Ruby	35802	152.9987	591.7913
Java	24371	15.8745	54.6264
Python	23760	21.0954	63.4868
PHP	21377	36.4115	137.6646
C	12340	15.5236	64.0051
C++	10216	35.4933	140.845
Shell	7060	31.1657	181.6001
Objective-C	6857	48.5569	111.374
VimL	5388	19.2055	68.1633
C#	5309	19.0451	46.5216
Perl	3533	8.5386	23.3256
Emacs Lisp	2016	16.1578	72.6258
Scala	1835	25.2202	55.605
CoffeeScript	1490	37.5742	99.0243
Haskell	1482	6.4859	13.5758
Clojure	1370	16.4743	38.2034
Erlang	1053	17.5717	45.6909
ActionScript	1009	9.0415	22.7379
Lua	889	3.6927	11.5027
Go	850	3.9362	8.8663
Groovy	802	5.081	9.7447

Puppet	594	4.3499	10.6146
R	536	4.1059	8.3893
Common Lisp	468	2.3275	2.8325
Matlab	408	1.496	1.6514
D	301	16.0559	33.5596
OCaml	295	7.9683	12.9141
Arduino	216	3.0204	5.0904
Assembly	215	2.3669	3.8967
Scheme	207	1.4642	1.9433
ColdFusion	183	9.6693	13.6654
Delphi	160	4.5636	5.711
Visual Basic	159	1.8107	3.0611
F#	133	8.685	10.0879
FORTRAN	132	4.8138	8.8442
HaXe	125	1.7893	1.6118
Powershell	107	5.2426	11.6247
Prolog	99	1.1399	1.1613
Rust	97	81.5072	52.0713
Racket	90	11.8613	17.836
Verilog	85	1.5453	2.3341
ASP	80	1.204	0.9147
VHDL	80	1.2041	0.6523
Objective-J	69	53.6024	94.1253
Vala	66	2.41	3.237
Tcl	50	1.8263	2.2258
Modelica	43	11.4	21.9358
AutoHotkey	42	2.861	5.1543
Standard ML	39	1.455	0.6757
SuperCollider	37	1.1255	0.4103
Eiffel	32	1.6532	1.8343
Pure Data	30	1.2516	0.9246
XQuery	30	1.7518	1.5897
Smalltalk	28	1.8775	2.4098
Elixir	26	31.124	20.9278
Coq	23	1.0542	1.0428
OpenEdge ABL	22	1.4818	1.0849

Io	21	1.4138	1.051
Opa	19	2.4026	2.9729
Nemerle	18	13.0585	14.6881
Ada	17	0.9438	0.301
Boo	12	1.0519	1.3665
Arc	12	12.1356	11.1514
Factor	10	34.661	29.0491
Gosu	10	3.1111	3.5495
Scilab	9	1.0265	0.1607
Dylan	9	5.1795	5.8698
Apex	8	1.0606	0.6937
Turing	7	6	2.0237
Rebol	7	9.4128	5.6165
Parrot	7	8.2581	4.9901
Ioke	5	1	0
Nimrod	5	10	5.1364
Mirah	4	2.8	1.4697
Nu	4	2	1.7321
ooc	3	2.25	1.0897
Logtalk	2	1.6667	0.4714
Self	2	6	3
Bro	2	1	0
Fantom	1	1	0
Kotlin	1	1	0
Max/MSP	1	1	0
repository_language	1	0	0
Fancy	1	9	0

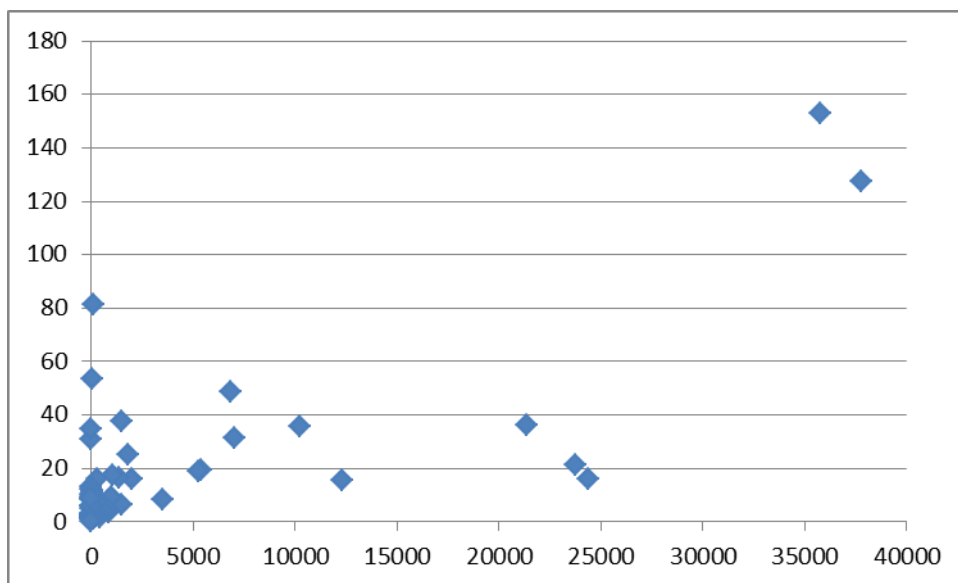


図 23 プロジェクト数, 平均 Fork 数

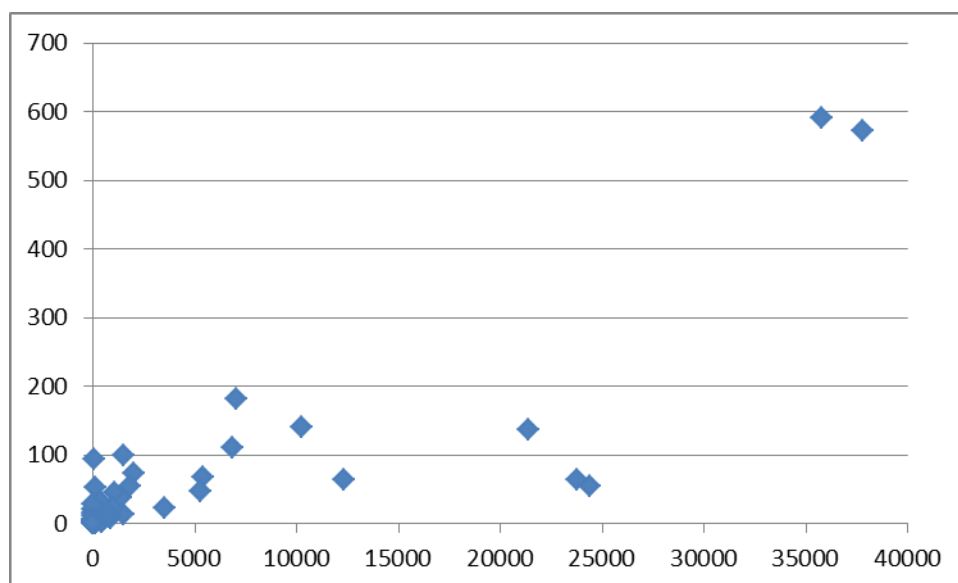


図 24 プロジェクト数, 標準偏差

図 23 は横軸がプロジェクト数, 縦軸が平均 Fork 数の分布図. 図 24 は, 横軸がプロジェクト数, 縦軸が標準偏差の分布図.

4.4.6.項で記述した, エクスポートの異なる URL 別 Fork 数のデータで作ったヒストグラムである.

GitHub にあげられているプロジェクトで使われている, プログラミング言語の種類とプロジェクト数, 平均 Fork 数, 標準偏差の表である. プロジェクトの数では, JavaScript のほうが Ruby より数が多いのだが平均や標準偏差で見ると Ruby の方が JavaScript より多いことが分かる.

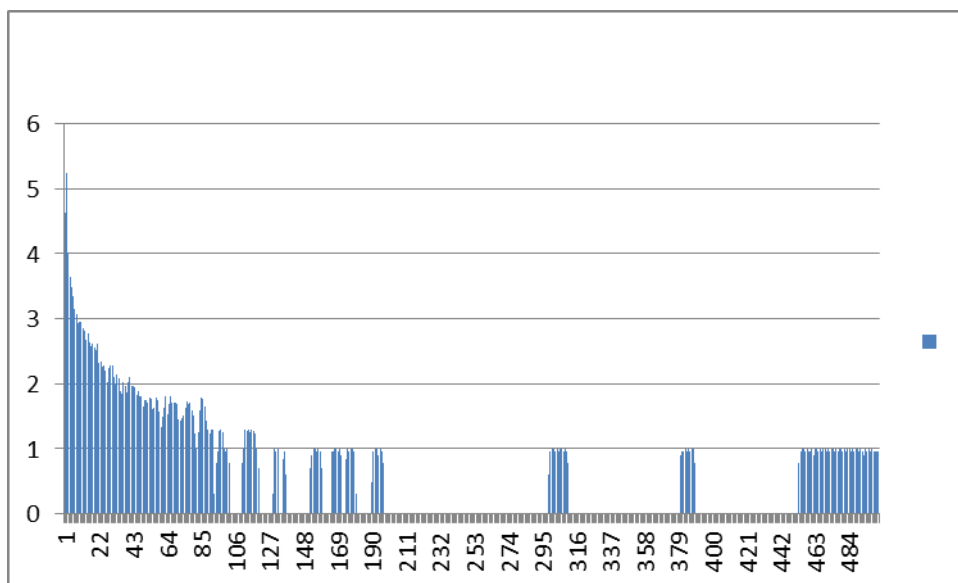


図 25 URL 別 Fork 数

図 25 は, 4.4.6.項で記述したエクスポートの異なる URL 別 Fork 数のデータで作った度数表の頻度を \log_{10} したヒストグラムである.

基の頻度でヒストグラムを作成すると, 桁が大きすぎてヒストグラムがとても見辛いので \log を使用した.

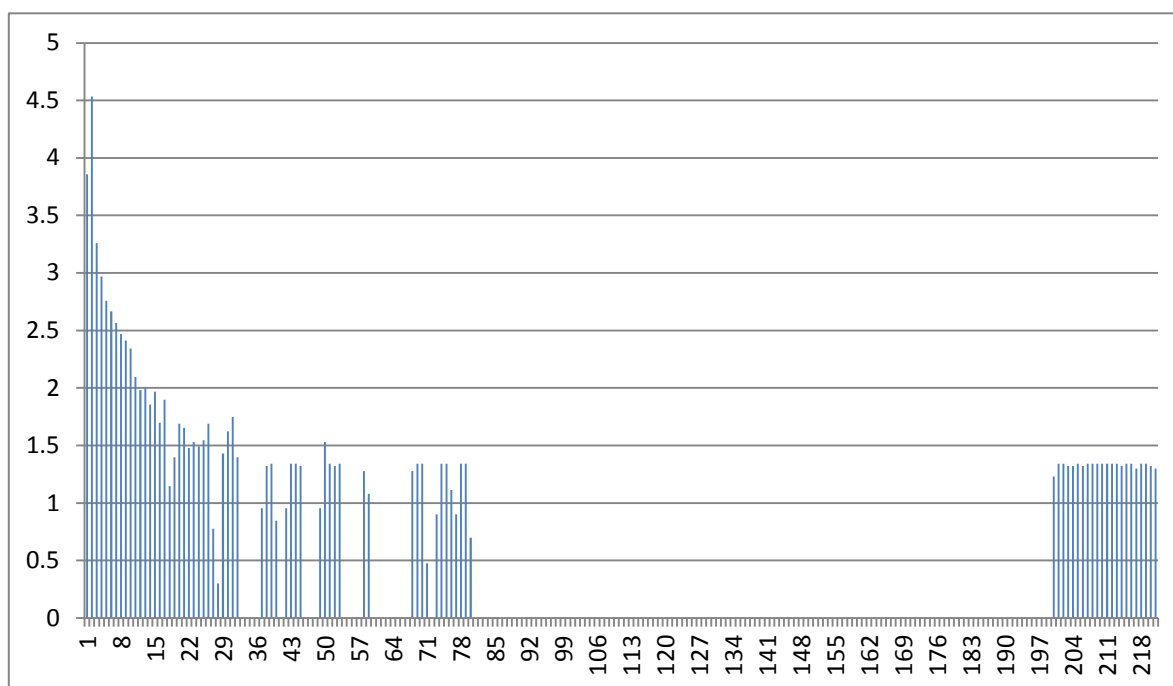


図 26 JavaScript

図 26 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした JavaScript のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

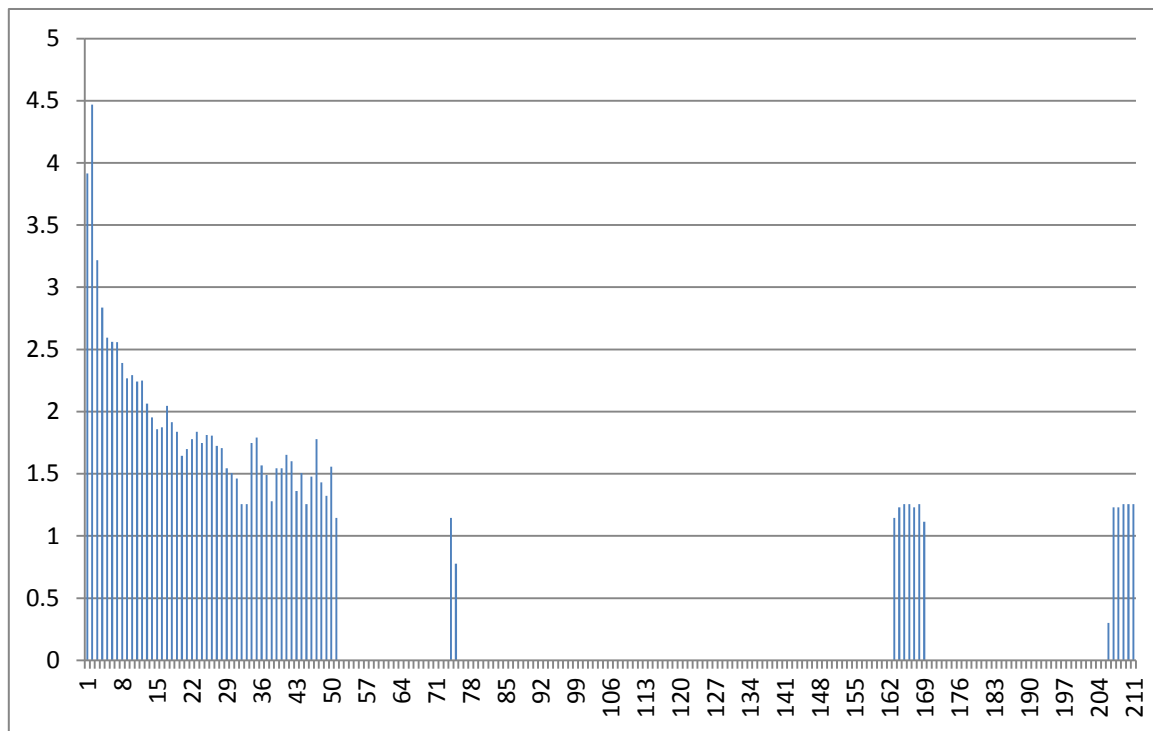


図 27 Ruby

図 27 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Ruby のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

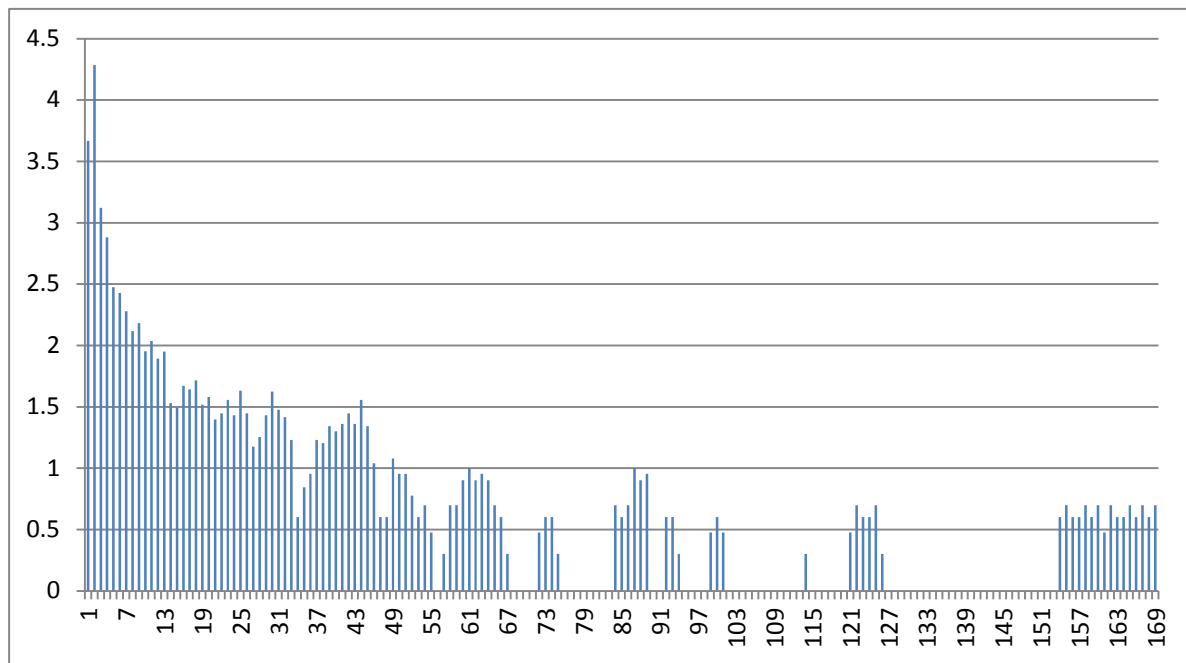


図 28 Java

図 28 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Java のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

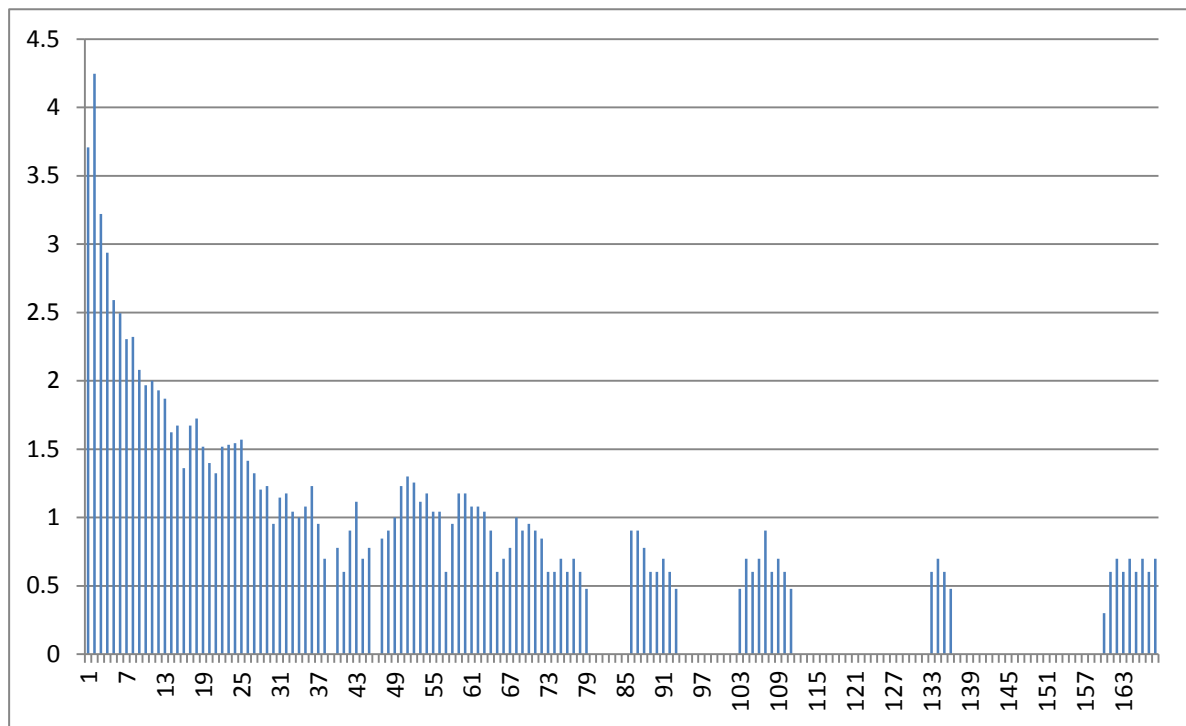


図 29 Python

図 29 は, 4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Python のデータで作った度数表の頻度を \log_{10} したヒストグラムである.

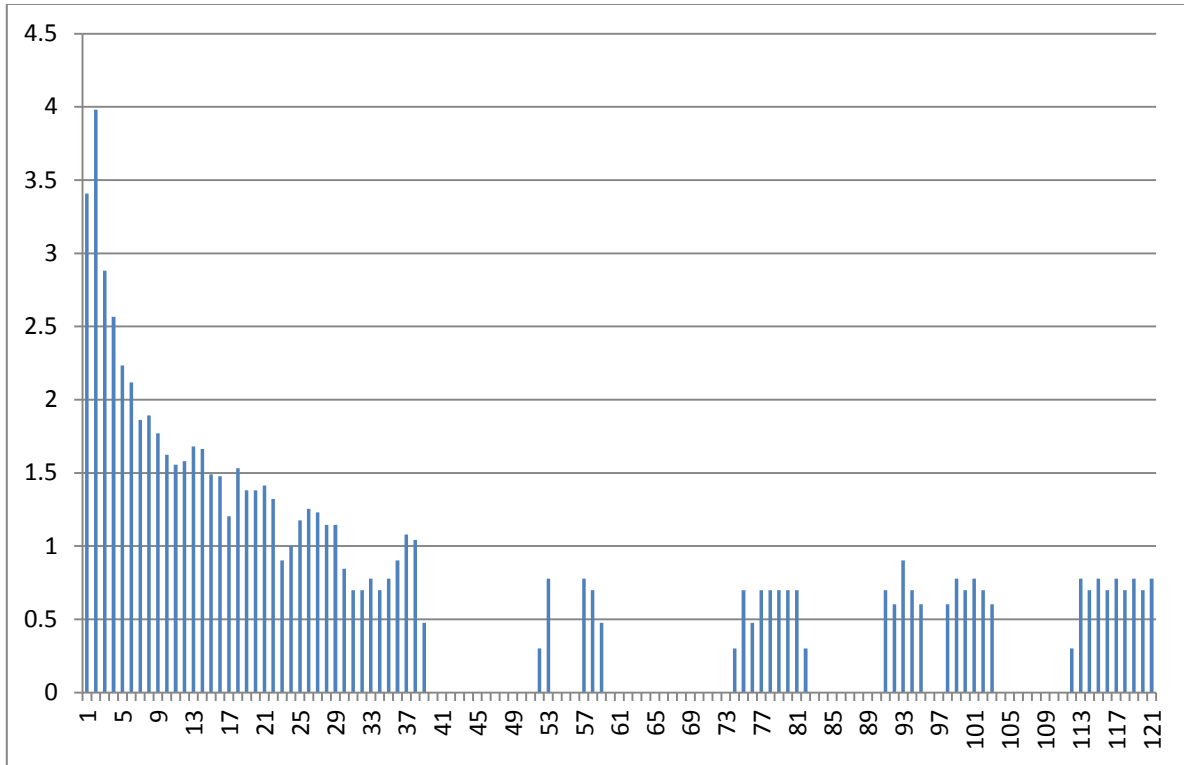


図 30 C

図 30 は, 4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした C のデータで作った度数表の頻度を \log_{10} したヒストグラムである.

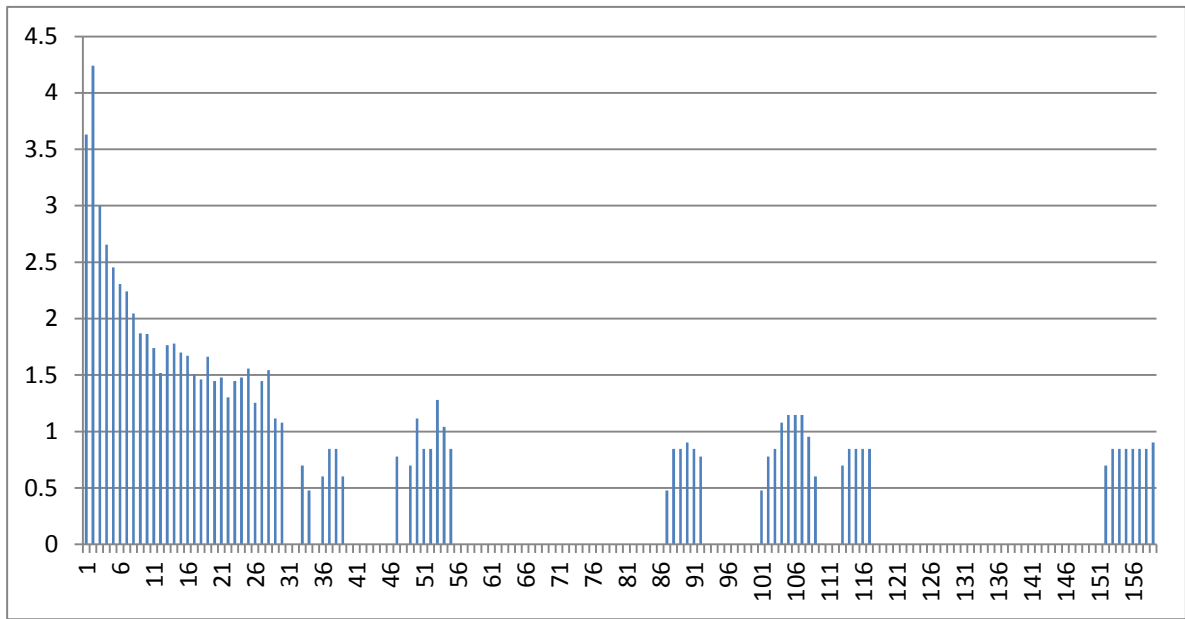


図 31 PHP

図 31 は, 4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした PHP のデータで作った度数表の頻度を \log_{10} したヒストグラムである.

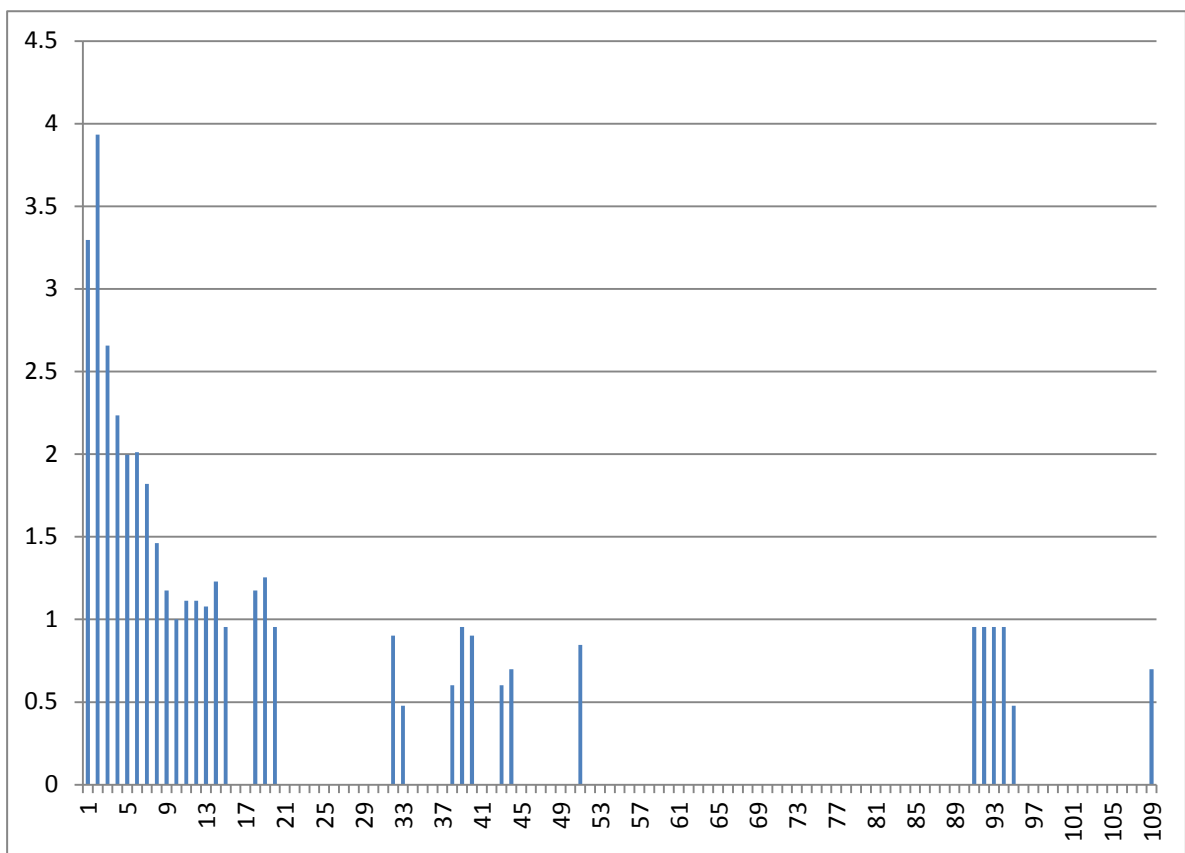


図 32 C++

図 32 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした C++ のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

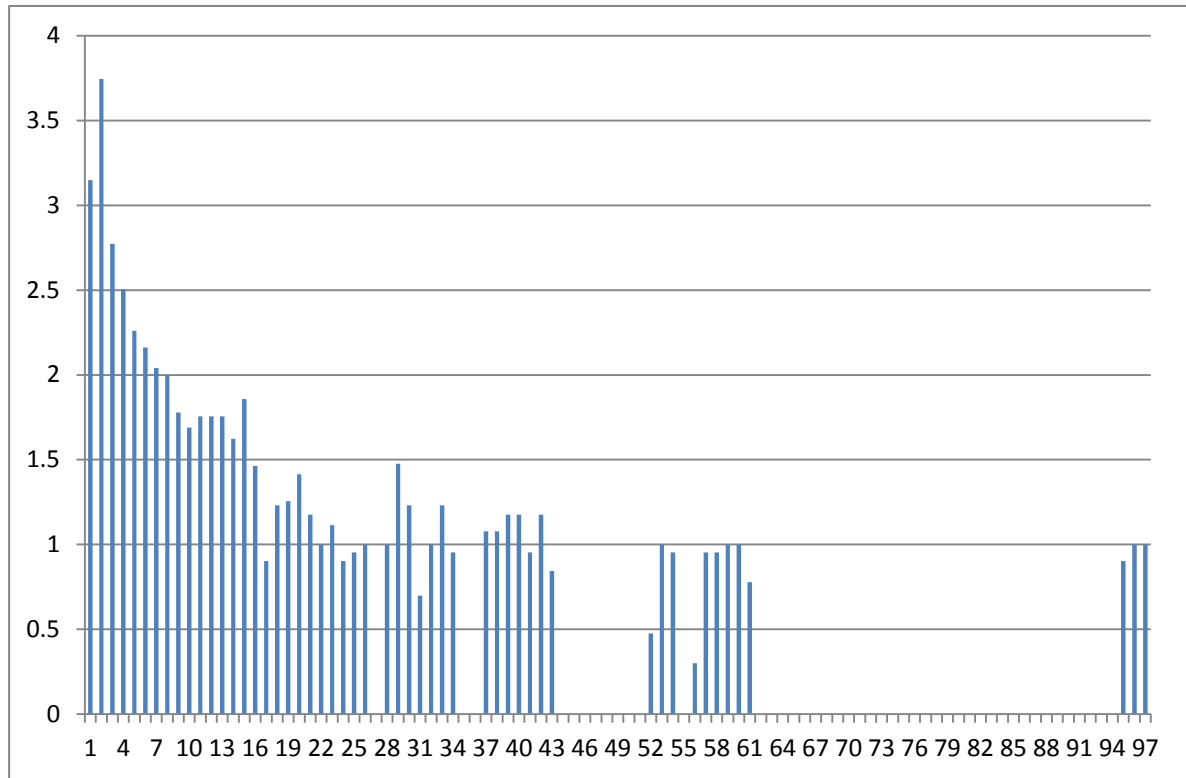


図 33 Objective-C

図 33 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Objective-C のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

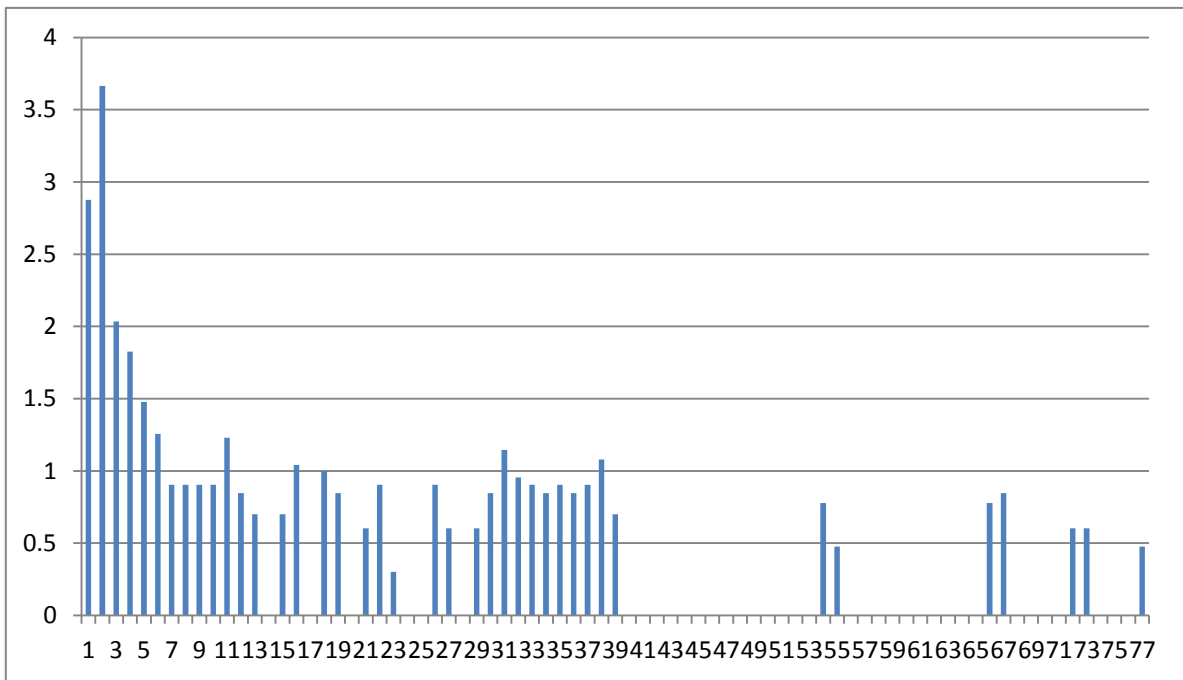


図 34 VimL

図 34 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした VimL のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

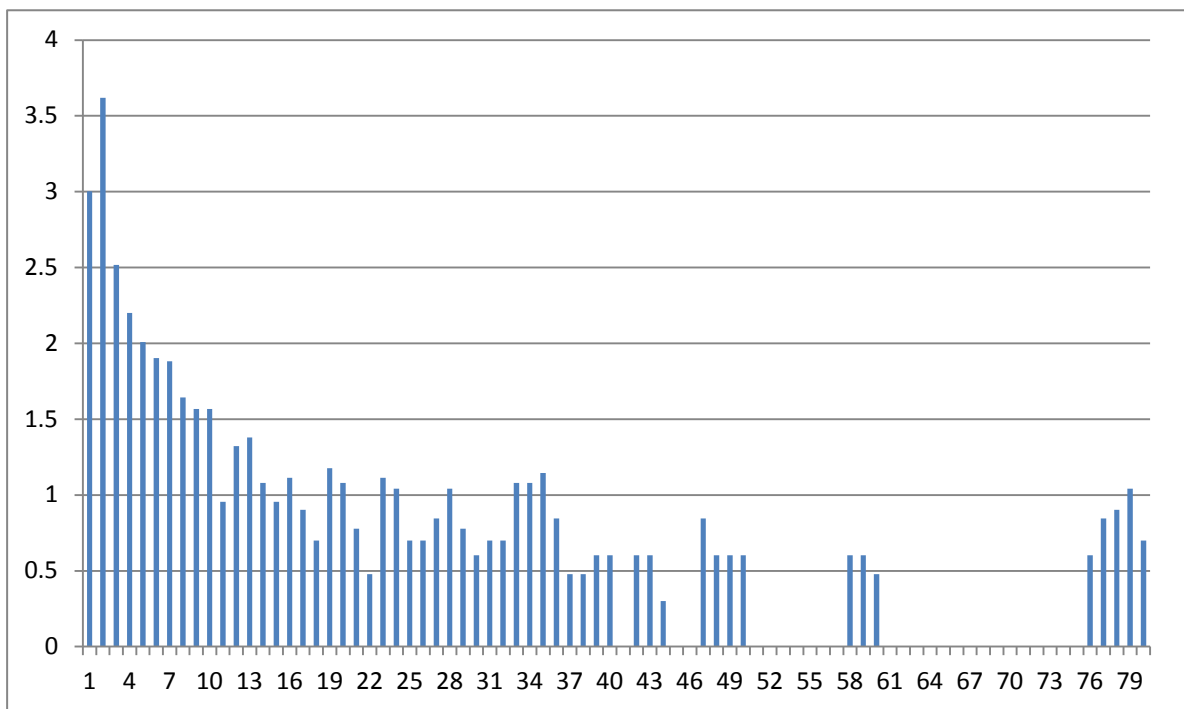


図 35 C#

図 35 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした C# のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

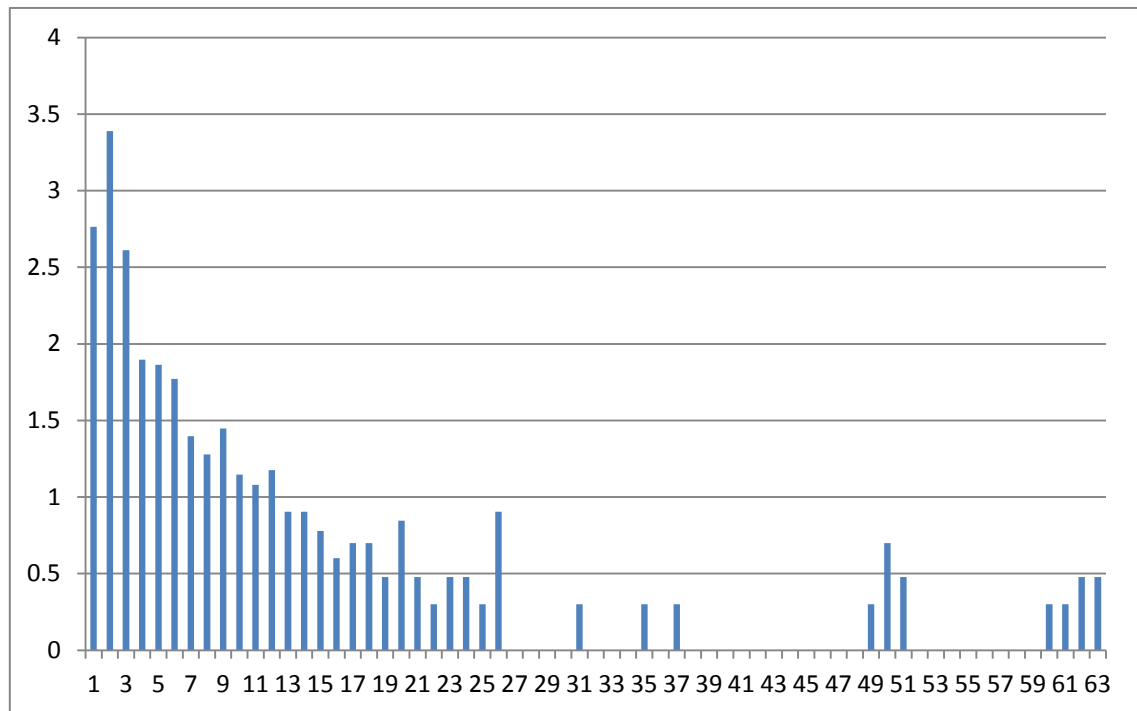


図 36 Perl

図 36 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Powershell のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

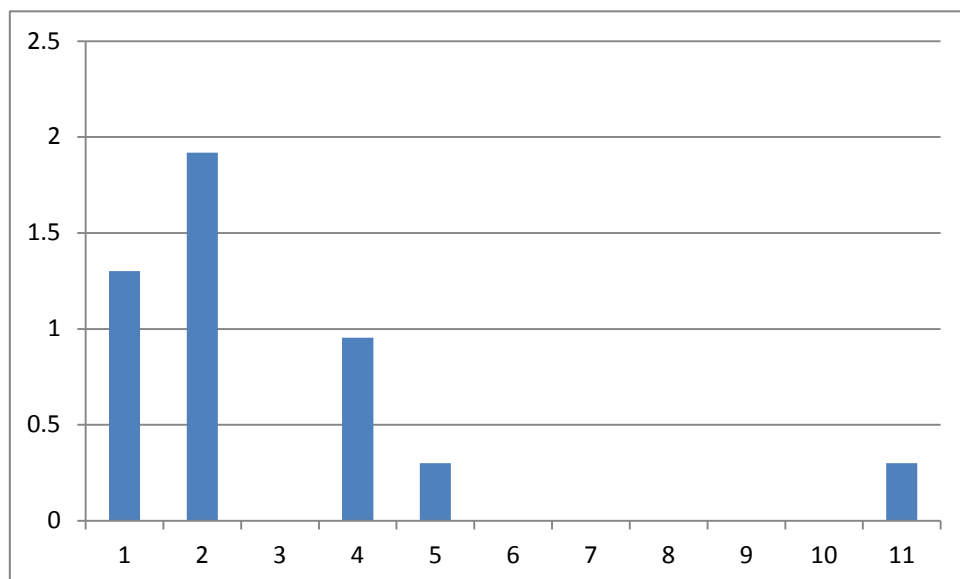


図 37 Powershell

図 37 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Powershell のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

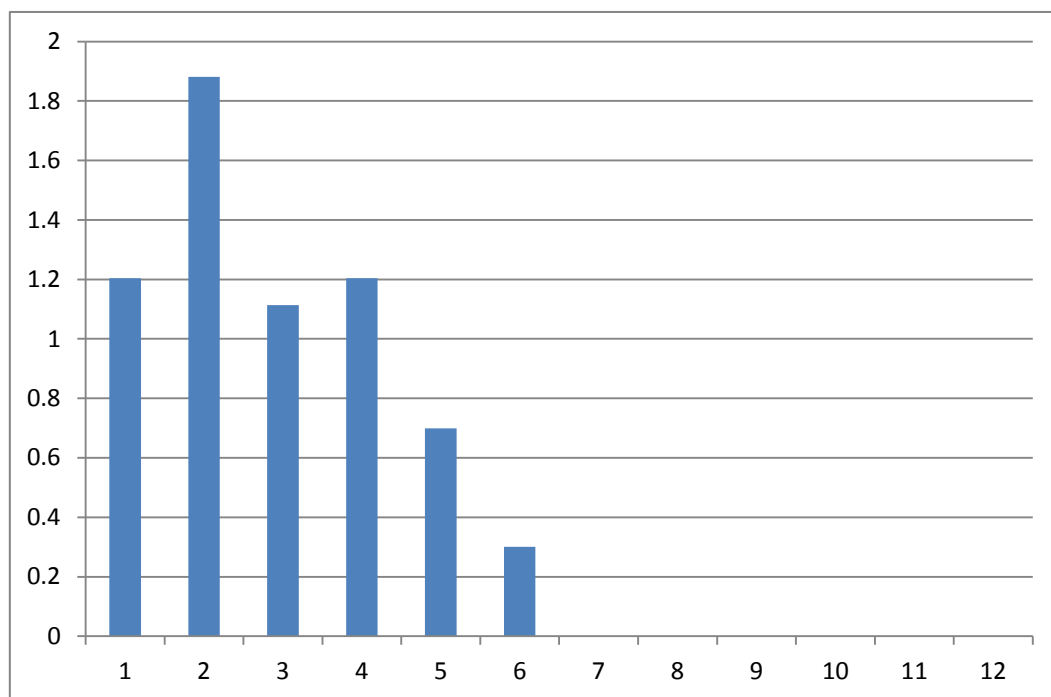


図 38 Haxe

図 38 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした HaXe のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

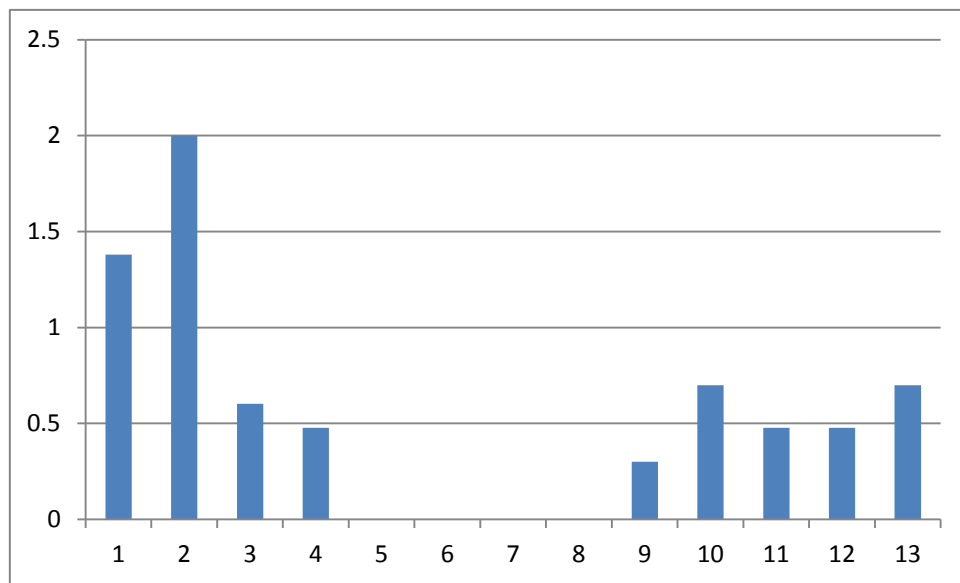


図 39 FORTRAN

図 39 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした FORTRAN のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

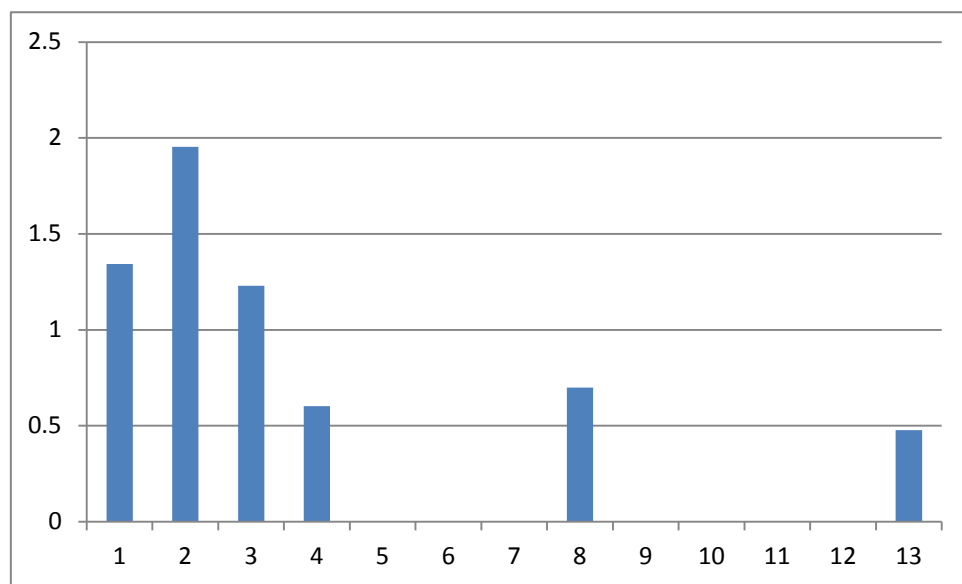


図 40 F#

図 40 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした FORTRAN のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

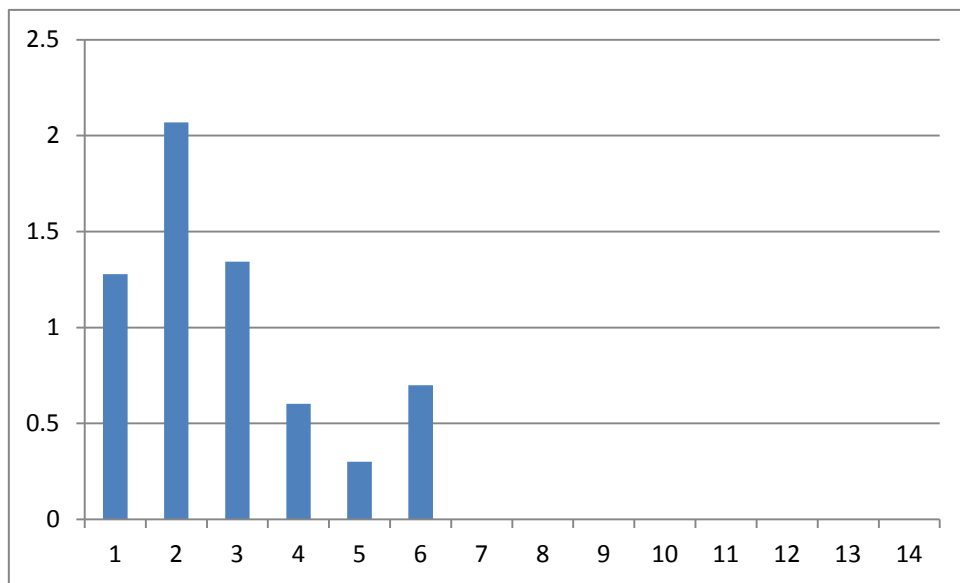


図 41 Visual Basic

図 41 は, 4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Visual Basic のデータで作った度数表の頻度を \log_{10} したヒストグラムである.

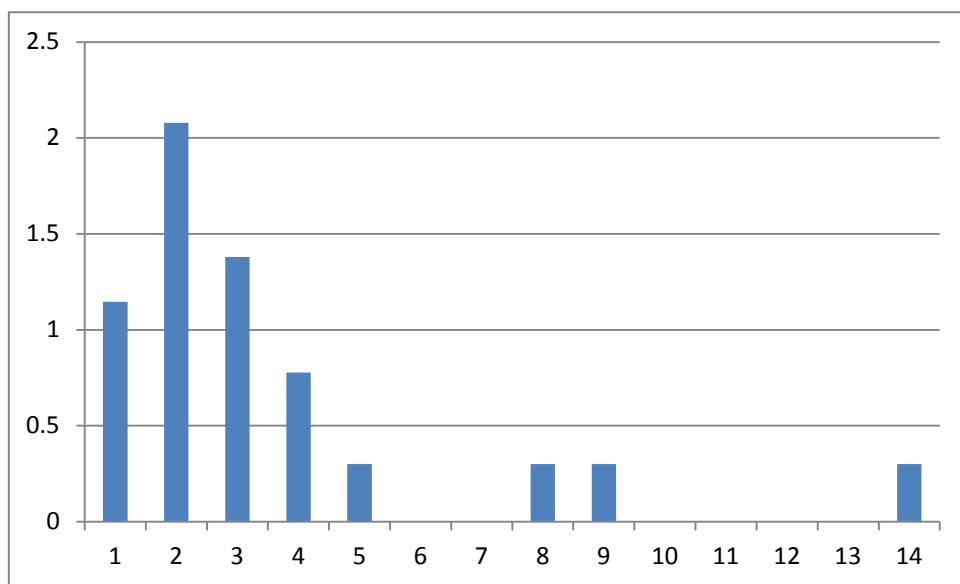


図 42 Delphi

図 42 は, 4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Delphi のデータで作った度数表の頻度を \log_{10} したヒストグラムである.

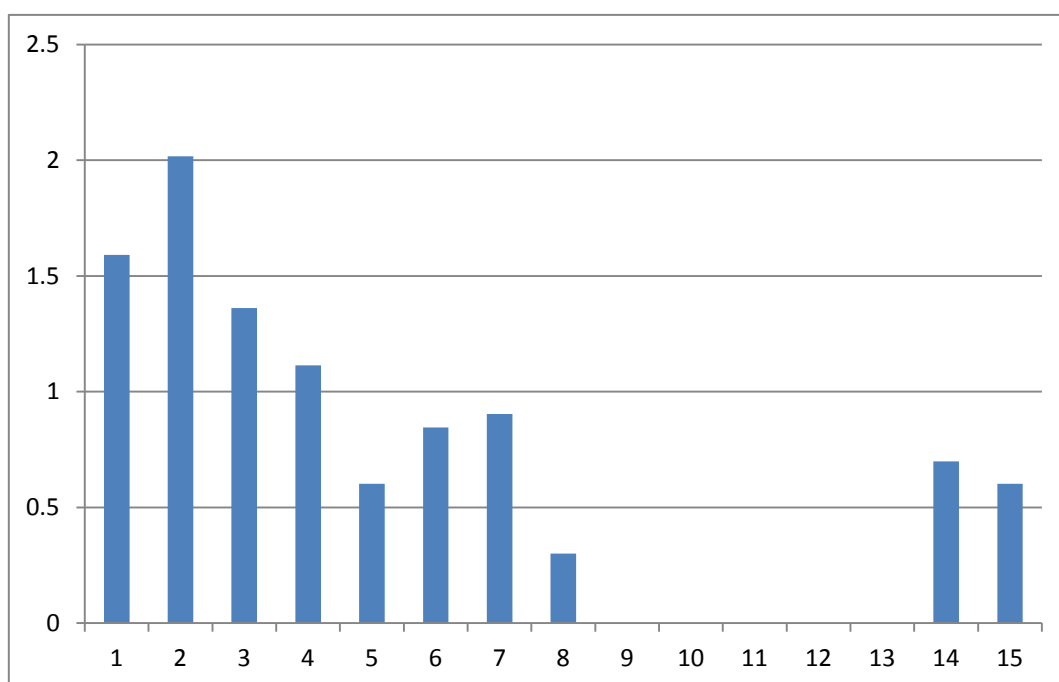


図 43 ColdFusion

図 43 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Delphi のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

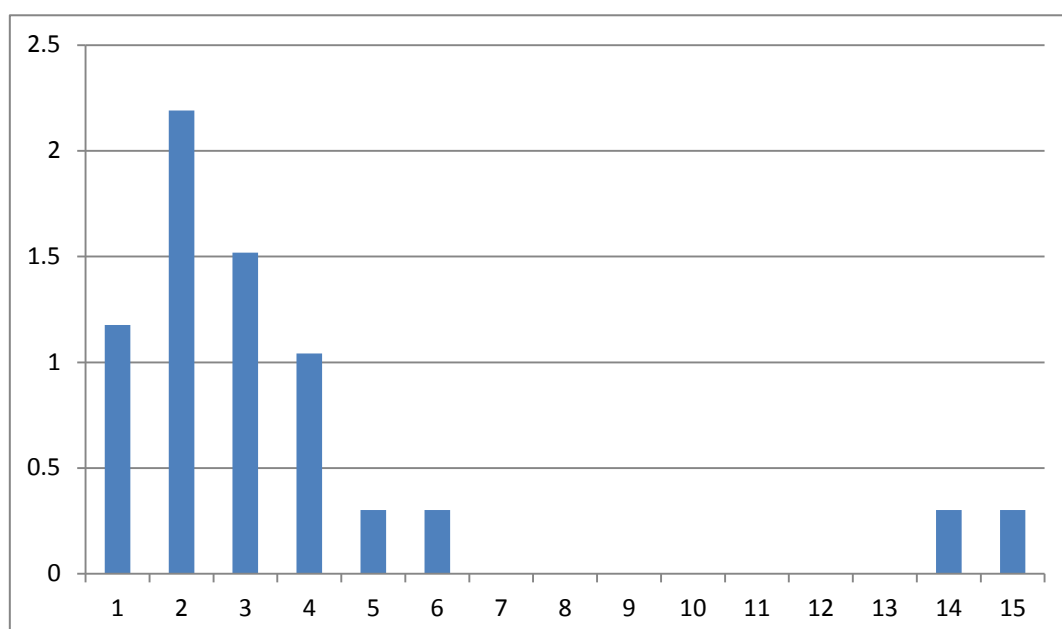


図 44 Scheme

図 44 は, 4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Scheme のデータで作った度数表の頻度を \log_{10} したヒストグラムである.

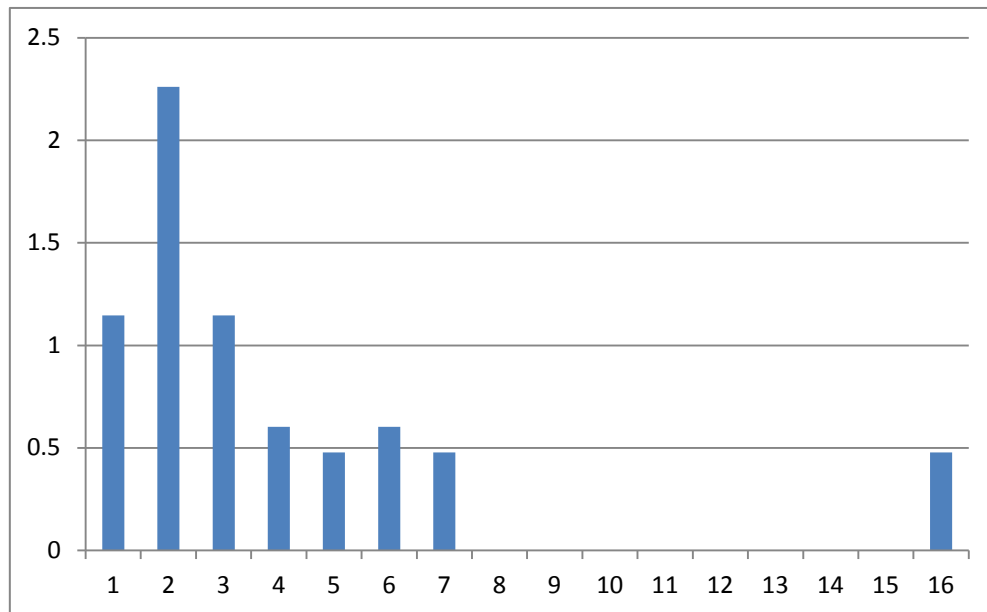


図 45 Assembly

図 45 は, 4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Assembly のデータで作った度数表の頻度を \log_{10} したヒストグラムである.

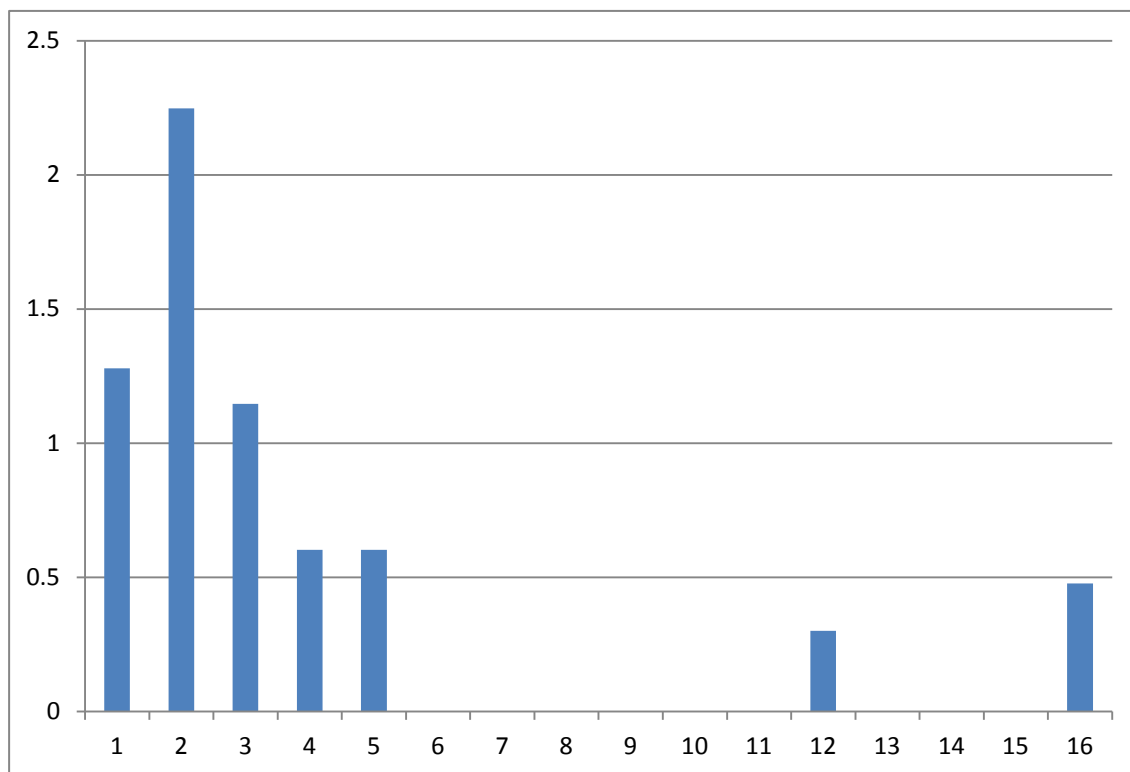


図 46 Arduino

図 46 は、4.4.6.項で記述した異なる言語別の Fork 数でエクスポートした Arduino のデータで作った度数表の頻度を \log_{10} したヒストグラムである。

5.3. 考察

以上の結果から考察を行った。

各プログラミング言語ごとの Fork 数を平均、標準偏差、ヒストグラムの複数の視点から見ることによってプロジェクト数が多い方が Fork されているわけではないことが分かった。プロジェクト数が少ないのに平均 Fork 数が多いものもあるのでプロジェクトを立ち上げる側で人気が高い言語が必ずしも参加者側からも人気が高いとは言えないことが分かる。

この結果から近年のオープンソフトウェアの開発現場で使われているプログラミング言語のトレンドも把握することが出来た。これにより今後プログラミングを学ぶ際の指標として使うこともできると考えられる。

謝辞

本研究を進めるにあたり、厳しくも優しくご指導して頂いた矢吹太朗准教授に感謝いたします。そして、多くのご指摘を下さいました矢吹研究室の同期・後輩の皆様に感謝いたします。