

OSS 開発プロジェクトにおけるメンバの貢献度調査

プロジェクトマネジメントコース

ソフトウェア開発管理グループ

矢吹研究室

1342011

石川大貴

目次

第 1 章	序論	2
第 2 章	背景	3
2.1	ソフトウェア開発	3
2.2	ウォーターフォール型開発	5
2.3	アジャイル型開発	6
2.4	開発プロセスの比較	6
2.5	テスト駆動開発	7
2.6	オープンソースソフトウェア開発	8
2.7	バージョン管理システム	8
2.8	git	9
2.9	GitHub	13
2.10	GitHub の機能	13
2.11	パレートの法則	17
第 3 章	目的	18
第 4 章	手法	19
4.1	調査対象のプロジェクト	19
4.2	プログラムの作成	21
4.3	プログラムの実行	27
4.4	プログラムの実行	34
第 5 章	結果・考察	47
第 6 章	結論	71
参考文献		72
第 7 章	謝辞	73

第 1 章

序論

ソフトウェア開発の現場では、主にウォーターフォール型開発が採用されていたが、現在ではアジャイル型開発が普及してきている。

アジャイル型開発では、テスト駆動開発がよく採用される。これは、プログラムに必要な各機能について最初にテストを書き、そのテストが動作する必要最低限な実装を行った後、コードを洗練させるという短い工程を繰り返し行う手法である。

開発プロセスに関する情報を誰でも見ることができる、オープンソースソフトウェア (OSS) 開発というものがある。OSS 開発には、OSS ホスティングサービスを利用して開発されることが多い。最もよく利用されている OSS ホスティングサービスの一つが GitHub である。実際、1000 万以上のユーザが GitHub を活用しており、リポジトリ数は 2430 万以上もある。近年の OSS 開発について調査するにあたり、多くのプロジェクトをホストする GitHub が適切だと考える。

本研究で GitHub 上のプロジェクトを調査し、テスト駆動開発において個人がプロジェクトに与える影響を調査したい。今回はメンバが追加したコードの行数を貢献度とし、メインコードとテストコードで違いがあるのか調査する。

パレートの法則が成り立つ可能性が考えられる。パレートの法則とは、全体の数値の大部分は、全体を構成するうちの一部の要素が生み出しているという理論であり、80:20 の法則とも呼ばれる。

第 2 章

背景

2.1 ソフトウェア開発

ソフトウェア開発とは、ユーザのニーズやマーケティング上の目標をソフトウェア製品に変換する作業である。一般的にソフトウェアの設計や製造の一連の工程を行う。ソフトウェア開発において一般的に行われる一連の工程は、要求定義工程、設計工程、実装工程、テスト工程、導入工程である。また、開発後の保守・運用などもある。

2.1.1 要求定義工程

要求定義工程とは、顧客が業務を進める際にどのようなシステムが欲しいのか、どのようなことを実現したいのかといった要求を引き出す、集める工程である。顧客と利用者が一緒でない場合などもあるため、要求が不完全であったり、曖昧であったり、矛盾してしまうことがある。ソフトウェア開発者はそれを聞き出して、一貫性のある要求仕様に纏め上げることが重要である。

2.1.2 設計工程

設計工程とは、可能な限り厳密な方法で、開発すべきソフトウェアを正確に記述する工程である。設計工程には外部設計と内部設計の二つの段階がある。外部設計では、利用者が必要とする要件に基づいて、利用者から見てシステムがどのように動作するかを決める。主に操作画面やインターフェースの構成、書式、データベースの構造などを決める段階である。内部設計では、ハードウェアの構成や調達するソフトウェアパッケージやシステムの処理方式など、要件定義書に記載された機能の実現に必要なシステムの構成を決定する。開発するシステムを大まかな機能ごとにサブシステムに分割し、それらのサブシステム間を繋ぐインターフェースなどを設計する。その後サブシステムをさらに小さいモジュールごとに分割し、各モジュール間のインターフェースなどを定義する。

2.1.3 実装工程

実装工程とは，設計工程にて定義された仕様を元に，機能を組み込んでいく工程である．設計からコードを作成する段階は，ソフトウェア開発において最も明白な工程であるが，必ずしも最大の工程とは限らない．

2.1.4 テスト工程

テスト工程とは，プログラムから仕様でない振舞や欠陥を見つけ出す工程である．テストで見つかったプログラム中の欠陥を修正する作業をデバッグという．プログラムを実際に動かしてみても行うテストを動的テストと呼び，プログラムを実際に動かしてみることなくソースコードやオブジェクトコードを検証する作業を静的テストと呼ぶ．

2.1.5 導入工程

導入工程とは，実際の稼働環境へシステムを移す工程である．ソフトウェア開発はテスト環境にて開発を行うため，利用者が利用できる環境に移す必要がある．ある時点で旧システムから新システムへ一気に切り替える一斉移行や，システムの機能や支店，部門などの単位ごとに新システムへ切り替えて行く順次移行などの方法がある．

2.1.6 保守・運用

保守・運用とは，完成後のシステムにおいて，ハードウェアの故障対応や業務の変化に伴う修正などを行うことである．障害の発生を事前に防ぐために定期的に行われる予防保守と，障害発生後に行われる事後保守がある．ソフトウェアの保守・運用は，初期の開発よりも長期に渡り，手間もかかる場合が多い．

2.2 ウォーターフォール型開発

ソフトウェア開発の手法にウォーターフォール型開発という開発手法がある。ウォーターフォール型開発とは、ソフトウェア開発の一連の工程が主に時系列順で要求定義、外部設計、内部設計、開発、テスト、運用と分割される。原則として前工程が完了しないと次工程に進まず、全ての工程が一度で終わるように計画を立て、進捗を管理する。これにより前工程の成果物の品質を確保し、前工程への手戻りを最小限にしている。利点として工程を明確に区別して、全体の機能設計を済ませてから機能を実装するため、進捗管理がしやすいことが挙げられる。そのため、多人数で作業を行うような大規模なシステム開発に向いている。問題点は前工程に違いがないことを前提にして、仕様を最初にすべて決めてから機能を実装するため、開発着手までに時間がかかる。さらに、テストで不具合が発生すると、後半になるほど手戻りの工数が大きくなってしまうため、途中での仕様変更は困難となる [1]。

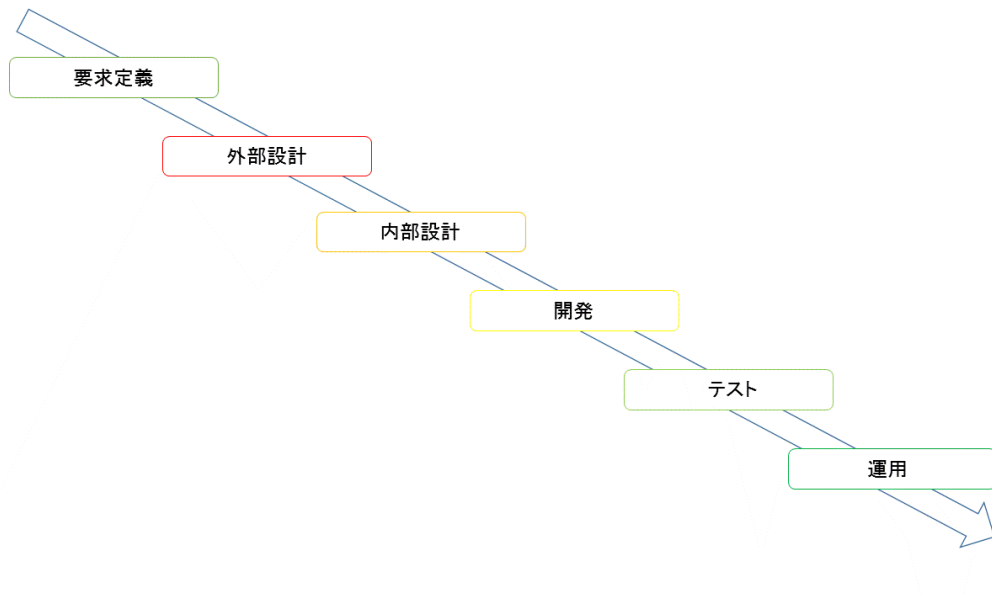


図 2.1 ウォーターフォール型開発

2.3 アジャイル型開発

ソフトウェア開発の手法にアジャイル型開発という開発手法がある。アジャイル型開発は仕様や設計の変更があることを想定し、初めから厳密な仕様や設計は決めない。開発対象を多数の小さな機能に分割し、分割した範囲ごとにおおよその仕様を決める。分割した分、短い期間で実装とテストを繰り返して徐々に開発を進めていくことが出来るので、ウォーターフォール型開発に比べて開発途中の仕様変更が容易でスピードのある開発が可能である。反対に、全体のスケジュールや進捗管理はしにくい [1]。

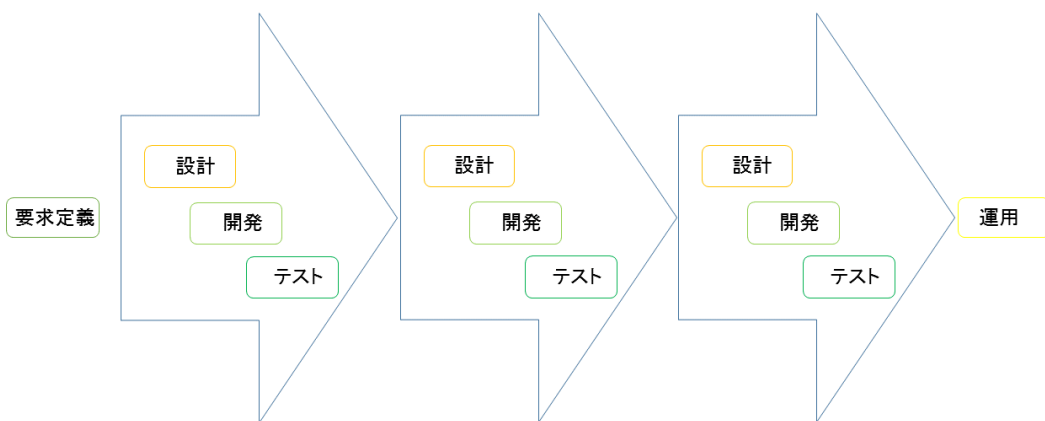


図 2.2 アジャイル型開発

2.4 開発プロセスの比較

ウォーターフォール型開発とアジャイル型開発には、それぞれメリットもあればデメリットの面があるため、プロジェクトごとに使い分ける必要がある。銀行システムなど大きくニーズの変化が起きにくいようなプロジェクトでは計画重視のウォーターフォール型開発を採用し、IT 市場の進化などに対して柔軟に適応して開発を行う必要があるプロジェクトではアジャイルソフトウェア開発を行う。ソフトウェア開発の現場では、主にウォーターフォール型開発が採用されていたが、現在ではアジャイル型開発が普及してきている。ビジネスにスピードが求められるようになり、システムへの要求事項も刻々と変化するため、ウォーターフォール型開発より市場投入までの期間短縮がしやすいアジャイル型開発が普及してきている [1]。

2.5 テスト駆動開発

アジャイル型開発では、テスト駆動開発がよく採用される。テスト駆動開発 (Test Driven Development; TDD) とは、プログラム開発手法の一種でプログラムに必要な各機能について最初にテストを書き、そのテストが動作する必要最低限な開発を行った後、コードを洗練させるという短い工程を繰り返し行う手法である。コードをきれいにすることや早い段階で動くコード、フィードバックを確保することが目的に挙げられる。汚いコードや設計が悪いコードは、バグを生みやすくテストもしづらい。テスト駆動開発では、テストと実装を同時に書くことで、そもそもテストしづらいコードが生まれにくくなり、コードが綺麗になる。また、実装の前にテストを書くことで、実装が正しいのかすぐに確かめられる。実装直後はコードの記憶が明確であるので、修正が容易であり、何週間も後にバグのあるコードを読まずに済む [1]。

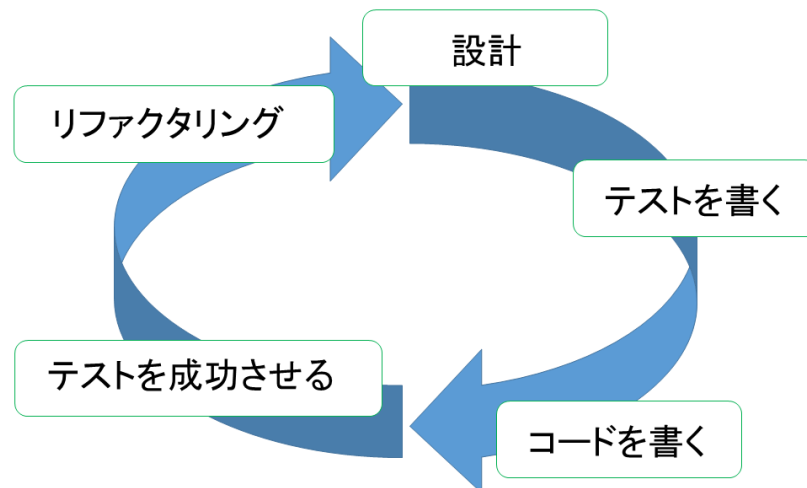


図 2.3 テスト駆動開発

2.6 オープンソースソフトウェア開発

開発プロセスに関する情報を誰でも見ることができるオープンソースソフトウェア（OSS）開発というものがある。企業、個人など参加形態を問わずに誰でもプロジェクトに参加することが可能である。

OSS 開発には、OSS ホスティングサービスを利用して開発されることが多い。オープンソースとは、人間が理解しやすいプログラミング言語で書かれたコンピュータプログラムであるソースコードを広く一般に公開し、誰でも自由に扱ってよいとする考え方である。OSS ホスティングサービスとは、そのオープンソースのソフトウェアを格納するリポジトリを中心に、バージョン管理システムや開発者同士の意思疎通の支援、ソフトウェアの公開機能を持つホスティングサーバをインターネット上で提供するサービスである [2]。

2.7 バージョン管理システム

バージョン管理システムとは、ファイルの作成日時、変更日時、変更点などの履歴を保管することができるシステムである。ソフトウェア開発ではソースコードを作成し、バグの修正や機能の追加ごとにソースコードの状態を記録し、それぞれのバージョンを管理する必要がある。何度も変更を加えたファイルであっても、バージョン管理システムにより、過去の変更内容を確認することや変更前の状態を復元することが容易になる。

多くのバージョン管理システムでは複数の人間がファイルの編集に関わる状況を想定している。複数の人間が複数のファイルをそれぞれ編集すると、ファイルの最新の状態が分からなくなることや同一ファイルに対する変更が競合するなどの問題が生じやすいが、バージョン管理システムではこのような問題を解決できる。

個人のファイル管理に使用することも可能であるし、ソフトウェアのソースコードだけでなく、設定ファイルや原稿の管理などにも使うことも可能である。

バージョン管理システムには、集中型バージョン管理システムと分散型バージョン管理システムがある。

集中型バージョン管理システムは、リポジトリをサーバに集中させて配置するため、1つのリポジトリしか存在しない。データが中央のサーバに集中されるので管理がシンプルであるが、サーバに接続できない状況だと最新のソースコードが取得できないため、開発の進みが遅くなる場合がある。また、サーバが故障等で、最新のソースコードが消えてしまうこともある。

分散型バージョン管理システムは、サーバにある特定のリポジトリを自分のリポジトリに複製し、ローカルで編集できる。サーバにあるリポジトリとは異なるリポジトリとなるため、サーバに接続できなくても開発することができる。分散型は複雑であるが、複数リポジトリがあるため、編集、やり直しが容易にできる。集中型バージョン管理システムと分散型バージョン管理システムのどちらが良いかは、双方にメリット・デメリットがあるのでケースバイケースである [3]。

2.8 git

git は、プログラムのソースコードなどの変更履歴を記録・追跡するための分散型バージョン管理システムである。Git のコマンドは様々なものがあるため一部を次に記す。

```
git init
```

リポジトリを作成する。

```
git init bare
```

ベアリポジトリの作成する。

```
git init shared
```

グループ書き込み権限を有効にする。

```
git clone
```

既存のリポジトリの複製を作成する。

```
git fsck
```

リポジトリの正当性チェックする。

```
git gc
```

リポジトリ内の不要なオブジェクトを削除し、最適化する。

```
git status
```

変更が加えられたファイルを表示する。

```
git diff
```

ファイルに加えられた変更点を diff 形式で表示する。

```
git add
```

作業ディレクトリ内の変更をステージングエリアに追加する .

```
git add a
```

すべての変更を含む作業ディレクトリの内容をステージングエリアに追加する .

```
git add u
```

以前コミットしたファイルだけステージングエリアに追加する .

```
git commit
```

ステージングエリアに追加された変更点をコミットする .

```
git commit -m " <メッセージ> "
```

<メッセージ> をコミットメッセージとして , ステージングエリアに追加された変更点をコミットする .

```
git commit a
```

ステージングエリアに追加されたすべての変更点をコミットする .

```
git commit amend
```

直前のコミットを修正する .

```
git reset
```

直前のコミットを取り消す .

```
git log
```

コミットログを表示する . 後程詳しい解説をする .

```
git revert
```

作業ツリーを指定したコミット時点の状態にまで戻す .

```
git checkout
```

ブランチを切り替える .

```
git branch
```

ブランチ情報の表示およびブランチの作成する。

```
git branch a
```

すべてのブランチを確認する。

```
git branch r
```

リモートブランチを確認する。

```
git branch d
```

ブランチを削除する。

```
git show-branch
```

ブランチの作成，変更，マージ履歴を表示する。

```
git merge
```

ローカルブランチをマージする。

```
git tag
```

コミットにタグを付ける。

```
git stash
```

現在の作業ディレクトリの状態を一時的に保管する。

```
git rebase
```

ブランチの派生元を変更する。

```
git pull
```

リモートリポジトリの変更点をローカルリポジトリにマージする。

```
git push
```

リモートリポジトリに自分のリポジトリの内容を送信する。

```
git remote
```

リモートリポジトリの一覧を表示する。

```
git fetch
```

リモートリポジトリの最新情報を追加する。

```
git config
```

インストールした Git に対してコマンドラインから設定を行う。

```
git clean
```

作業ディレクトリから追跡対象外のファイルを削除する。

2.9 GitHub

よく利用されている OSS ホスティングサービスの一つに GitHub がある。GitHub とは、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り、アメリカのサンフランシスコを拠点とする GitHub 社によって運営されている。GitHub では git バージョン管理システムを使用し、SNS の機能やプロジェクトのバグ管理に使える Issues、コードレビューを効率化する Pull Request などの開発に役立つ機能がいくつもある。

非公開のリポジトリを作成することもできるため、GitHub を使ってバージョン管理を行っている企業も多数ある。さらに、連携が可能な開発ツールやサービスも多くある。実際、1000 万以上のユーザが GitHub を活用しており、リポジトリ数は 2430 万以上もある。このように、多くのプロジェクトをホストする GitHub のプロジェクトを調査することにより、ソフトウェア開発の傾向を調べることができると思われる [4]。

2.10 GitHub の機能

GitHub の機能について、一部の説明を次に記す。

2.10.1 リポジトリ

ファイルやディレクトリの状態を記録する場所である。保存された状態は、内容の変更履歴として格納され、変更履歴を管理したいディレクトリをリポジトリの管理下に置くことで、そのディレクトリ内のファイルやディレクトリの変更履歴を記録することができる。

2.10.2 リモートリポジトリ

手元に置いてあるローカルなリポジトリ以外の、ウェブ上に置かれたリポジトリのことである。

2.10.3 commit・コミット

ファイルやディレクトリの追加や変更を、リポジトリに記録するために行う操作のことである。

2.10.4 clone・クローン

サーバにあるリモートリポジトリをローカルにコピーすることである。

2.10.5 Push・プッシュ

リモートリポジトリに自分の変更履歴がアップロードされて、リモートリポジトリ内の変更履歴がローカルリポジトリの変更履歴と同じ状態にすることである。

2.10.6 branch・ブランチ

履歴の流れを分岐して記録するためのものである。分岐したブランチは他のブランチの影響を受けないため、同じリポジトリ中で複数の変更を同時に進めていくことができる。

2.10.7 pull・プル

リモートリポジトリから最新の変更履歴をダウンロードしてきて、自分のローカルリポジトリにその内容を取り込むことである。

2.10.8 Pull Request・プルリクエスト

相手に対して自分のブランチを pull してもらうように要求する機能のことである。

2.10.9 Origin

clone 元のリモートリポジトリのことである。

2.10.10 ステージング

修正作業によって更新したもののうち、コミット対象の更新を選り分けるための作業のことである。

2.10.11 ステージングエリア

ステージングを行う領域のことである。

2.10.12 Revert

ステージングエリアに追加した変更を戻すことである。

2.10.13 タグ

コミットを参照しやすくするために、わかりやすい名前を付けるものである。

2.10.14 ラベル

自由に作成でき、Issue をフィルタリングできる機能である。

2.10.15 チェックアウト

ブランチ内から作業ディレクトリに展開されたファイル・ディレクトリ群のことである。

2.10.16 トラック

Git 上で変更内容を追跡できるようにすることである。特に、ローカルブランチとリモートブランチとを紐付けすることである。また、ファイルやディレクトリに対して言う場合は、Git オブジェクトに変換してリポジトリに格納することである。

2.10.17 作業ツリー

ブランチ内から作業ディレクトリに展開されたファイル・ディレクトリ群のことである。

2.10.18 Merge・マージ

当該ブランチに対して別のブランチの差分を取り込むことである。

2.10.19 リベース

マージの一種で、当該ブランチの更新差分を無かったことにして、別ブランチの最新を取り込んだ後、当該ブランチの更新差分を適用する処理することである。

2.10.20 チェリーピック

マージの一種で、全ての差分を取り込むのではなく、特定のコミットを指定して取り込むことである。

2.10.21 競合

マージの際に、同一のファイル・ディレクトリに対しての更新が発生していることである。

2.10.22 fetch・フェッチ

リモートリポジトリの最新を取り込むことである。

2.10.23 Fork・フォーク

GitHub のサービスで、相手のリポジトリを自分のリポジトリとしてコピー・保持できる機能のことである。

2.10.24 Issue

一つのタスクを一つの Issue に割り当てて、トラッキングや管理を行えるようにするための機能である。

2.10.25 Wiki

Wiki 機能は、いつでもだれでも文書を書き換えて保存できるため、共同で文章を作成できる機能である。

2.10.26 MileStone・マイルストーン

やるべきタスクの管理に Issue を用いることができるようにする機能である。

2.10.27 Gist

コードの断片など、リポジトリに入れるほどでもないコードを管理、公開するのに重宝する機能である。また、ブログなどを書いている人は Gist にサンプルコードを登録すれば、ブログの記事に埋め込むこともできる。

2.10.28 News Feed

フォローしているユーザや Watch しているプロジェクトの活動情報が流れてくる機能である。最新の動向を確認することである。

2.10.29 スター

GitHub 内のプロジェクトをお気に入りできる機能である。

2.10.30 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。

2.11 パレートの法則

パレートの法則とは、経済において、全体の数値の大部分は全体を構成するうちの一部の要素が生み出しているという理論である。80:20 の法則、ばらつきの法則とも呼ばれる。パレートの法則は経済以外にも、自然現象や社会現象などさまざまな事例に当て嵌められることが多い。ただし、現代で言われるパレートの法則の多くは、法則と言うよりも経験則の類である [5]。

パレートの法則が用いられる事象の例を次に挙げる。

- ビジネスにおいて、売上の 80% は全顧客の 20% が生み出している。よって売上を伸ばすには顧客全員を対象としたサービスを行うよりも、20% の顧客に的を絞ったサービスを行う方が効率的である。
- 商品の売上の 80% は全商品銘柄のうちの 20% で生み出している。
- 売上の 80% は全従業員のうちの 20% で生み出している。
- 仕事の成果の 80% は費やした時間全体のうちの 20% の時間で生み出している。
- プログラムの処理にかかる時間の 80% はコード全体の 20% の部分が占める。

第 3 章

目的

本研究では、GitHub 上のプロジェクトを調査する。テスト駆動開発において、個人の貢献度がメインコードとテストコードで違いがあるのかを調査する。

第 4 章

手法

以下の手順で研究を進める．

1. GitHub のプロジェクトでテストコードがあるプロジェクトを探す．
2. プロジェクトのコミットごとにユーザ名，日時，メインコードの行数，テストコードの行数をそれぞれ取得できるプログラムを作成する．
3. 作成したプログラムを実行する．
4. 取得したデータをメンバごとにソートし，メインコードとテストコードの行数を集計する．
5. メンバごとの行数を比較し，パレートの法則が成り立つか調査をする．

4.1 調査対象のプロジェクト

GitHub のプロジェクトでテストコードがあるプロジェクトを探す．GitHub でホスティングされているプロジェクトを無作為に抽出（ランダムサンプリング）を行い，test フォルダがある場合にそのプロジェクトを調査の対象とする．

本研究で調査するプロジェクトを次に記す．

表 4.1 調査対象プロジェクト

ユーザ名	リポジトリ名
angular	angular.js
caolan	async
jashkenas	backbone
bower	bower
adobe	brackets
jashkenas	coffee-script
Shopify	dashing
plataformatec	devise
discourse	discourse
visionmedia	express
gruntjs	grunt
visionmedia	jade
jekyll	jekyll
jquery	jquery
less	less.js
janl	mustache.js
mozilla	pdf.js
scottjehl	Respond
resque	resque
hakimel	reveal.js
LearnBoost	socket.io
jashkenas	underscore

4.2 プログラムの作成

調査対象のプロジェクトからデータを取得するプログラム作成する．作成するプログラムでは，プロジェクトのコミットごとにユーザ名，日時，メインコードの行数，テストコードの行数をそれぞれ取得できるようにする．

作成下プログラムは次の2つか成り立つ．

lineCounter.sh

```
#!/bin/bash
cd $1
git log --pretty=format:"%H,%cn,%cd" --date=iso --first-parent --no-merges >
  ../$1-commits.csv
cd ..
cat $1-commits.csv | python lineCountScriptCreator.py $1 > $1-count.sh
bash $1-count.sh > $1-count-result.csv 2> $1-error.log
awk -F ',' 'BEGIN{OFS=","} {print $0,$4-$5;}' $1-count-result.csv >
  $1-result.csv
```

lineCountScriptCreator.py

```
from datetime import datetime
from dateutil.parser import parse
from dateutil import tz
import sys
myProject = sys.argv[1].replace("'", "\'")
print("rm %s-error.log" % (myProject))
for line in sys.stdin:
    x = line.split(',')
    myHash = x[0]
    myName = x[1]
    myDate = datetime.strptime(parse(x[2]).astimezone(tz.gettz('UTC')),
'%Y-%m-%d %H:%M:%S')
    print("echo %s >&2" % (myHash))
    print("cd %s" % (myProject))
    print("git checkout -f %s 2>> ../%s-checkout-error.log" %
(myHash, myProject))
    print("cd ..")
    print("if [ -e %s/test ]; then" % (myProject))
    print(" echo %s,%s,%s,$(grep -rI '' '%s' | grep -v '^%s/\.git/' | wc -l),
$(grep -rI '' '%s/test' | wc -l)" % (myHash, myName, myDate, myProject,
myProject, myProject))
    print("else")
    print(" echo %s,%s,%s,$(grep -rI '' '%s' | grep -v '^%s/\.git/' | wc -l),0"
% (myHash, myName, myDate, myProject, myProject))
    print("fi")
```

4.2.1 lineCounter.sh の解説

```
#!bin/bash
```

先頭に指定した「#!」で始まる「シバン」と呼ばれる文字列は、スクリプトを実行するためのインタプリタを指定する。「このシェルスクリプトは bash によって解釈、実行される」と宣言するためのものである。これは決まり文句のようなもので、必ず 1 行目に指定する。

```
cd <パス>
```

「cd」とは、指定したディレクトリに移動することができるコマンドである。<パス>には現在位置から、目的のファイルやフォルダまでの道筋である相対パスと最上位階層から目的のファイルやフォルダまでのすべての道筋である絶対パスのどちらでも使用できる。

次の記号を使うことによって、ディレクトリを移動することもできる。

```
cd /
```

ルート・ディレクトリ (システム上最上層のディレクトリ) に移動する。

```
cd ..
```

親ディレクトリ (一階層上のディレクトリ) に移動する。

```
cd ~/
```

ホームディレクトリ (ユーザ上最上層ディレクトリ) に移動する。

```
git log <オプション>
```

「git log」とはリポジトリの履歴情報を見るコマンドである。デフォルトでは直近のコミットしか見られないため、オプションを使うことによってさまざまな履歴情報を見ることができる。また、最後に「> “ファイル名”」と入れることによって、ファイルに情報を書き出すことができる。

オプションを次に記す。

```
-p -<表示する数>
```

各コミットのパッチを表示する。

```
--word-diff
```

変更点を単語単位で表示する。

```
--stat
```

各コミットで変更されたファイルの統計情報を表示する。

```
--first-parent
```

ブランチを指定する .

--abbrev-commit

SHA-1 チェックサムの全体ではなく最初の数文字のみを表示する .

--name-only

コミット情報の後に変更されたファイルの一覧を表示する .

--relative-date

完全な日付フォーマットではなく , 相対フォーマットで日付を表示する .

--no-merges

マージコミットを除外する .

--grep <単語>

検索をする .

--pretty=format:" <オプション記号 > "

コミットを別のフォーマットで表示する .

表 4.2 オプション記号

記号	説明
%H	コミットのハッシュ
%h	コミットの短縮ハッシュ
%T	ツリーのハッシュ
%t	ツリーの短縮ハッシュ
%P	親のハッシュ
%p	親の短縮ハッシュ
%an	Author の名前
%ae	Author のメールアドレス
%ad	Author の日付
%ar	Author の相対日付
%cn	Committer の名前
%ce	Committer のアドレス
%cd	Committer の日付
%cr	Committer の相対日付
%s	件名


```
cat <ファイル名.>csv | python lineCountScriptCreator.py <リポジトリ名>  
> <ファイル名.>sh
```

「cat」とは、ファイルまたは標準入力の内容をそのまま標準出力に出力するコマンドである。ファイルの中身を確認することができる。また、複数のファイルを指定することで、複数のファイルを連結するのに使うことができる。「|(パイプ)」とは、コマンドの標準出力を次のコマンドに渡す処理ができるコマンドである。「cat」で開いた内容を「lineCountScriptCreator.py」に渡す。lineCountScriptCreator.py は git の履歴情報からソースコードの行数をカウントするスクリプトを作成するプログラムである。lineCountScriptCreator.py については後程説明をする。

```
bash <リポジトリ名.>sh > <リポジトリ名.>csv 2 > <ファイル名-error.>log
```

「bash」とは、ユーザによってキーボードから入力された文字やマウスのクリックなど、与えられた指示を OS の中核部分に伝え、対応した機能を実行するように OS に指示を伝えるコマンドである。lineCountScriptCreator.py にて作成されたスクリプトの実行とその結果を CSV 形式に書き出し、エラーがあればエラーログに書き出す。

```
awk -F ' ',' ' BEGIN{OFS=","} {print $0,$4-$5;} ' <ファイル名_tmp.>csv ><ファイル名  
.>csv
```

「awk」とは、区切られた複数列のテキストデータを処理するときに使うコマンドである。四則演算などもでき、渡されたファイルの一行ずつを処理する。上記の例だと、「-F」は CSV 形式の区切りの文字列を表す。「BEGIN」は読み込み開始とともに実行するという意味である。「OFS」は出力時の区切りを設定する。print は出力する内容を表す。print 内で「\$0」というのは元のデータをそのまま出力するという意味である。また、「\$4-\$5」というのは「4 列目 - 5 列目 (コード合計 - テストコード)」を表す。前者のファイル名には整理したファイル名、後者には整理後の保存したいファイル名を入力する。

4.2.2 lineCountScriptCreator.py の解説

```
from datetime import datetime
from dateutil.parser import parse
from dateutil import tz
import sys
```

「from import」で指定したモジュールをインポートができる。

```
myProject = sys.argv[1].replace("'", "\\'")
```

渡された引数（今回はリポジトリ名）を「'」から「\\'」に置換して myProject 変数に保存する。

```
print("rm %s-error.log" % (myProject))
```

“ リポジトリ名 ” -error.log の削除を行う。

```
for line in sys.stdin:
```

読み込んだファイルの一行一行を処理して繰り返す。

```
x = line.split(',')
```

CSV 形式には「Hash,Name,Date」と別れているので「,」x に配列を使って変数として入れる。

```
myHash = x[0]
```

MyHash 変数に x[0] を代入する。つまり、MyHash 変数には Hash が代入される。

```
myName = x[1]
```

MyName 変数に x[1] を代入する。つまり、MyName 変数には Name が代入される。

```
myDate = datetime.strptime(parse(x[2]).astimezone(tz.gettz('UTC')),
    '%Y-%m-%d %H:%M:%S')
```

myDate 変数に x[2] を代入している代入する。つまり、MyDate 変数には Date が代入される。また、git log で取得したデータの時間は投稿した人の国のタイムゾーンが記載されているので全てを UTC（協定世界時）に変換する。

例) 2017-01-10 00:38:56 -0500 2017-01-10 05:38:56

```
print("echo %s >&2") % (myHash)
print("cd %s") % (myProject)
print("git checkout -f %s 2>> ../%s-checkout-error.log") % (myHash, myProject)
print("cd ..")
```

エラーログを出力する .

```
print("if [ -e %s/test ]; then") % (myProject)
print(" echo %s,%s,%s,$(grep -rI '' '%s' | grep -v '^%s/\.git/' | wc -l),
$(grep -rI '' '%s/test' | wc -l)") % (myHash, myName, myDate,
myProject, myProject, myProject)
print("else")
print(" echo %s,%s,%s,$(grep -rI '' '%s' | grep -v '^%s/\.git/' | wc -l),0")
% (myHash, myName, myDate, myProject, myProject)
print("fi")
```

IF[条件式]...A の処理...else...B の処理という構文である .

条件式はリポジトリに test ディレクトリの中にフォルダがあるかないかを判別する . test ディレクトリの中にファイルがある場合は A の処理を行う . test ディレクトリの中にファイルがない場合は B の処理を行う . A では , Hash , Name , Date , .git フォルダ (履歴情報) を除いた行数と test ディレクトリの行数を出力という処理を行う . A の処理だけでは , test ディレクトリの中にファイルがない場合にエラーが出てしまい , 正常にカウントできないので , B の処理を行う . B の処理では , test ディレクトリの行数を 0 と出力する .

4.3 プログラムの実行

プログラムの実行環境は Linux を想定する．今回は Linux ディストリビューションの一種である CentOS7 を使用する．用意したコンピュータの OS は Windows のため，VirtualBox と Vagrant を使用して仮想マシンを作成し，CentOS7 を導入する．また，必要なソフトウェアを簡単に導入，管理するために Chocolatey をインストールする．

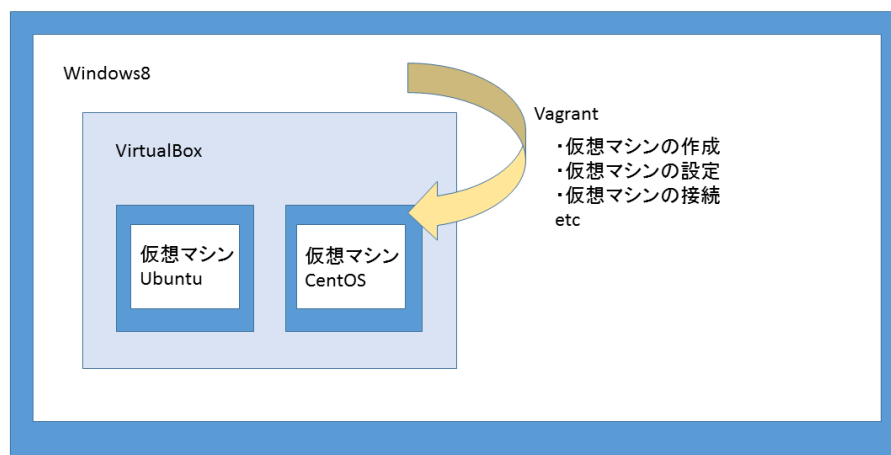


図 4.1 実行環境

4.3.1 Linux とは

Linux とは、狭義では Linux カーネル、広義ではそれをカーネルとして用いたシステムを指す Unix 系オペレーティングシステムである。

Linux は、スーパーコンピュータ、メインフレーム、サーバ、パーソナルコンピュータ、組み込みシステム、幅広い種類のハードウェアと共に使用されている。特にサーバ、メインフレーム、スーパーコンピュータ用の OS として使用される。携帯電話、ネットワークルータ、テレビ、ハードディスクレコーダ、カーナビゲーションシステムといった組み込みシステムでもよく使われている。スマートフォンやタブレット端末用プラットフォーム Android は Linux カーネルの上に構築されている。

Linux の開発は、フリーかつオープンソースなソフトウェアの共同開発として最も傑出した例のひとつである。Linux カーネルのソースコードは無償で入手でき、GNU 一般公衆利用許諾書のもとにおいて、非営利・営利に関わらず誰でも自由に使用、修正、頒布できる。Linux は、世界中の開発者の知識を取り入れるという方法によって、あらゆる方面に利用できる幅広い機能と柔軟性を獲得し、数多くのユーザの協力によって問題を修正していくことで高い信頼性を獲得した。

デスクトップやサーバ用の Linux は、Linux ディストリビューションという形でパッケージ化され配布されている。有名な Linux ディストリビューションとしては、Debian とその派生である Ubuntu、Linux Mint や Red Hat Linux とその派生である Fedora、Red Hat Enterprise Linux、CentOS などがある。各 Linux ディストリビューションは、Linux カーネル、システムソフトウェア、ライブラリ等、巨大なコンパイル済のアプリケーション群を含んでいる。

デスクトップ OS として Linux を使用することは、かつては技術者や上級ユーザだけが行うことというイメージが強かったが、最近では一般ユーザでも容易に使用できる。デスクトップ環境が充実したり、非常に簡単にインストールできるディストリビューションが登場したり、各種ハードウェアへの対応や自動設定機能が大幅に向上するなどしたためである。サーバでの利用を想定したディストリビューションなどでは、標準インストールからグラフィカルインタフェースをすべて排除しているものもある。また、Linux は自由に再頒布できるので、独自のディストリビューションを作ることも自由である [6]。

4.3.2 CentOS とは

CentOS とは、商用 Linux ディストリビューション「Red Hat Enterprise Linux」との完全互換を目指したオープンソース Linux ディストリビューションである。CentOS という名前は、「Community Enterprise Operating System」コミュニティベースで開発されたエンタープライズレベルの OS の略表記と言われている。Linux ディストリビューションの代表格となっており、CentOS プロジェクトによって、積極的にオープンな開発が行われている。

CentOS は、「Red Hat Enterprise Linux」の中のフリーライセンスで使えるパーツを組み合わせた形でリビルドされている。そのため、ライセンス費用がゼロで導入コストを節減できる。商用製品である「Red Hat Enterprise Linux」とほぼ同等の機能を持ち、さまざまな用途に対応できるサーバ OS として、高い品質を備え、長期間運用に耐えうる安定性がある。

CentOS はサーバでの長期間稼働を想定し、安定性を優先した設計思想になっている。最新技術を頻繁に取り込んでいくようなことがないため、他の OS やディストリビューションに比べて、アップデートの回数が少なめになっている。サーバ OS にとって、アップデートのためにサーバを停止させる回数が少ないというのは、重要なポイントである。

オフィススイートやマルチメディアツールなどのインストールをして環境設定をすることによって、デスクトップ OS として使用できる。CentOS は有償の OS を使いつつ、無償での提供を実現した優秀な OS であることやサーバで非常に役に立つことから、Linux の環境を構築するには最適である [7]。

4.3.3 VirtualBox とは

VirtualBox とは、オープンソースの仮想化ソフトウェアの一つである。現在の開発は米国オラクルが行っている。既存のオペレーティングシステム（ホスト OS）上にアプリケーションの一つとしてインストールされ、その中で追加のオペレーティングシステム（ゲスト OS）を実行することができる。例えば、Windows8 がホスト OS として動作しているマシン上で、CentOS をゲスト OS とすることができる。サポートされるホスト OS は Linux , macOS , Microsoft Windows , Solaris である。また後述するように、ソースコードが配布されているため、他の Unix 系の OS でも導入できる。例えば、FreeBSD では ports で導入することができる。ゲスト OS としてサポートされるのは、FreeBSD , Linux , OpenBSD , OS/2 Warp , Windows , Mac OS X Server , Solaris など多岐にわたる。x86/x64 アーキテクチャの OS であれば基本的に動作する [8] 。

VirtualBox 本体により提供される基本機能の一部を次に記す。

- スナップショット
- シームレスモード
- クリップボード
- 共有フォルダ
- シリアルデバイスとシステム間の切替えを支援するユーティリティ
- コマンドラインからの操作
- リモートディスプレイ

4.3.4 Vagrant とは

Vagrant とは、仮想開発環境構築ソフトウェアである。Vagrant を用いると、構成情報を記述した設定ファイルを元に、仮想環境の構築から設定までを自動的に行うことができる。以前は VirtualBox をターゲットとしていたが、1.1 以降のバージョンでは VMware などの他の仮想化ソフトウェアや Amazon EC2 のようなサーバ環境も対象とできるようになった。Vagrant 自身は Ruby で作成されているが、PHP や Python、Java、C#、JavaScript といった他のプログラミング言語の開発においても用いることができる。Vagrant は Chef、puppet などの構成管理ツールと連携して環境構築を自動化できる。開発環境を構築する際、通常は環境構築手順書を見ながら開発者が自分のローカル環境に構築するが、環境構築を自動化してしまえば環境構築する手間と環境構築手順書を書く手間を省ける。ネットワーク上にある Box ファイルを共有することが容易なのでチームで同一の環境を構築することが簡単にできる [9]。

vagrant のコマンドを次に記す。

`vagrant up`

Vagrantfile に記述された設定を読み込み、仮想マシンを起動する。

`vagrant halt`

起動している vagrant を停止する。

`vagrant reload`

起動している仮想マシンを再起動する。

`vagrant destroy`

仮想マシンを消去する。

`vagrant status`

仮想マシンの状態を確認する。

4.3.5 Chocolatey とは

Chocolatey とは、Windows 上で動作するソフトウェアをコマンドラインからインストール、アンインストール、アップデート、検索することができるパッケージ管理ツールである。有名なソフトウェアはだいたい Chocolatey で入れることができる。ブラウザから公式サイトにアクセスし、ダウンロード、インストールという手順を踏まずに、コマンドラインから 1 発でインストールできる手軽さは非常に魅力的である。

4.3.6 実行環境の準備

4.3.7 Chocolatey のインストール

Chocolatey のインストールを行う。Chocolatey のインストールは、Chocolatey のサイト (<https://chocolatey.org/install>) に掲載されているコマンドを実行すればできる。

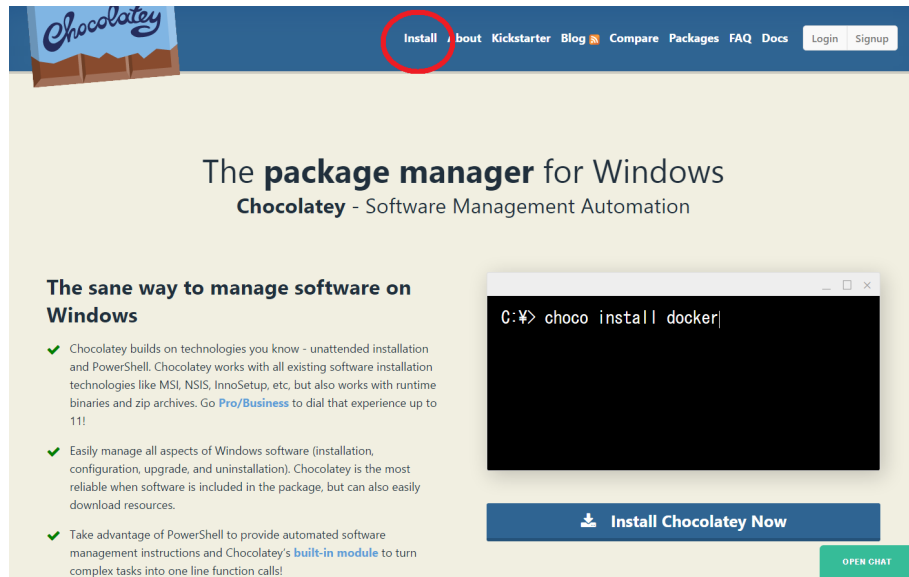


図 4.2 Chocolatey の公式サイト

Chocolatey のサイトに下記の画像ようなページがあるので、赤丸と同じ部分をダブルクリックする。コマンドがコピーされるので、管理者のコマンドプロンプトに貼り付けて Enter キーを押せば実行される。

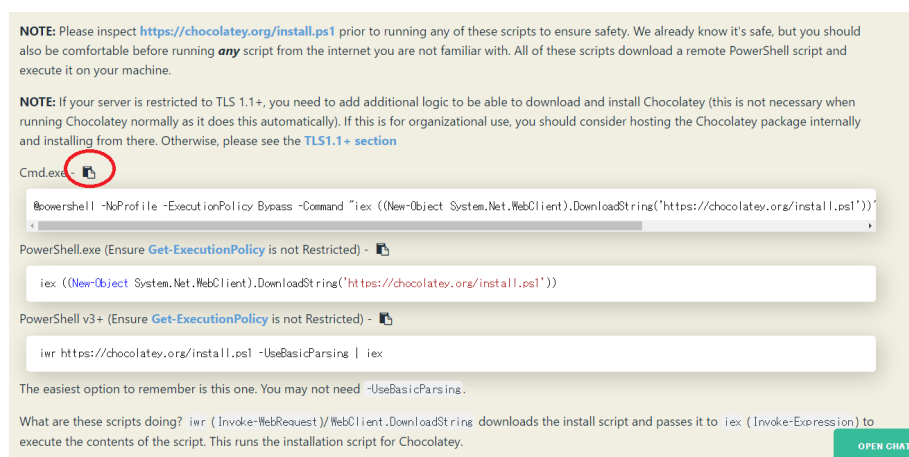


図 4.3 Chocolatey の公式サイト-インストールページ

4.3.8 VirtualBox と Vagrant

導入した Chocolatey で VirtualBox と Vagrant をインストールする．インストール方法は管理者のコマンドプロンプトを開いて chocolatey install コマンドを使用するだけである．このコマンドは cinst と省略できる．バージョンについては，特に気になれば Chocolatey のライブラリにある最新パッケージが適用される．以下のコマンドでインストールする．

```
cinst virtualbox
cinst vagrant
```

4.3.9 仮想マシンの作成

仮想マシンを作成し，CentOS7 を導入する．今回は myVagrant フォルダ作成し，以下で作業を行う．

```
cd myVagrant
mkdir myCentos
cd myCentos
vagrant init puppet/centos7-x64
vagrant up
```

以上で仮想マシンの準備は完了である．

4.3.10 仮想マシンの接続

ゲスト OS に接続する．接続には以下のコマンドを使用する．

```
vagrant ssh
```

ユーザ：vagrant，パスワード：vagrant でログインすることができる．

4.3.11 共有フォルダの作成

ホスト OS とゲスト OS で共有のフォルダを作成する．デバイス 共有フォルダで共有したいホストのフォルダを選択する．ここでは Desktop とする．ゲスト OS のコンソールで mkdir share などとして共有用のフォルダを作成する．

以下のコマンドでフォルダを共有する．

```
sudo mount -t vboxsf Desktop share
```

以上でゲスト OS の share がホスト OS の Desktop と同じになる．

4.3.12 コマンドのインストール

必要なコマンドのインストールを行う。

git のインストール GitHub のプロジェクトからデータを取得するため，git をインストールする．git clone 等のコマンドが使用できるようになる．

```
sudo yum install y git
```

dateutil のインストール作成したプログラムで Python のライブラリの dateutil.parser を使用しているので，dateutil をインストールする．dateutil.parser を使うと，日付を表す文字を datetime オブジェクトに変換できる．

```
sudo yum install -y python-dateutil
```

4.4 プログラムの実行

4.4.1 リポジトリのクローン

調査するプロジェクトのリポジトリをクローンする．git clone < URL > で GitHub 上のプロジェクトをクローンすることができる．「git clone」とは，リポジトリを複製（ローカル環境へコピー）するためのコマンドである．オプション指定をすることによって，次のようなことができる．

```
git clone < URL > <ディレクトリ名>
```

ディレクトリ名の指定をする．

```
git clone depth 1 < URL >
```

新リビジョンのみの取得をする．

git clone はリポジトリの履歴情報までも含めてリポジトリを複製するため，大規模なプロジェクトの場合は多くのファイルを取得・コピーしなければならないことがある．最新のソースファイルを取得する目的のみで使用する場合は，「-depth」オプションで取得するとよい．

プロジェクトをクローンすることができたら，以下のコマンドでプログラムを実行する．

```
bash lineCounter.sh リポジトリ<>
```

リポジトリ名-result.csv のファイルにコミット ID，名前，日時，プロジェクト全体のコード行数，テストコードの行数，メインコードの行数が取得されているはずである．

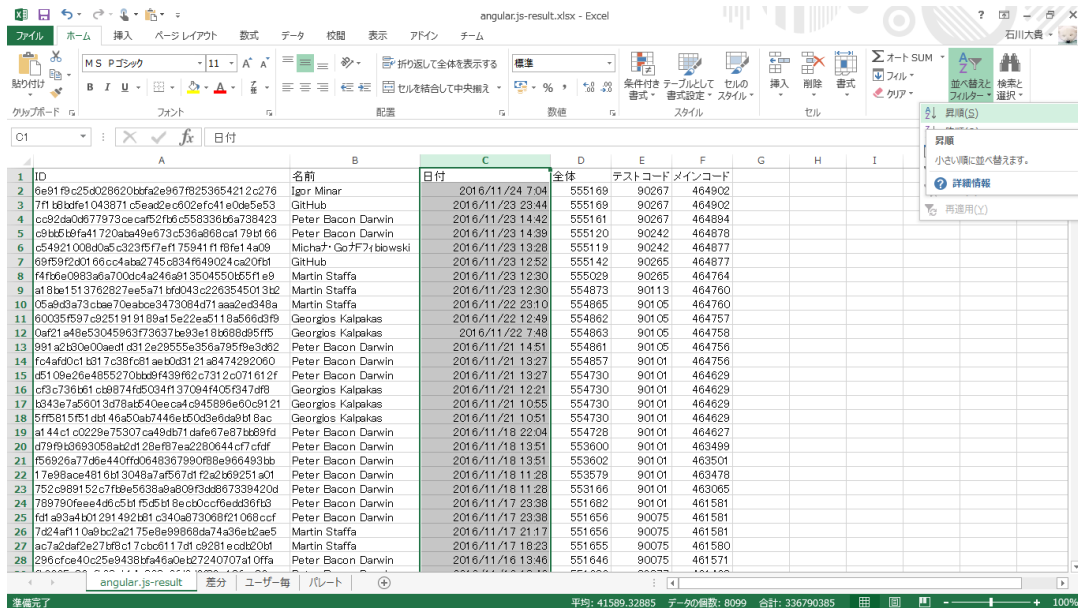
取得したデータをメンバごとにソートし、メインコードとテストコードの行数を集計する。

データの取得ができていれば、ファイル名を<リポジトリ>-result.xlsx にエクスポート (csv 形式からスプレッドシート形式に変換) し、データを処理する。図 1 の 1 行目のように作業が行いやすくなるような見出しを追加する。

	A	B	C	D	E	F	G	H	I	J	K	L
1	ID	名前	日付	全体	テストコード	メインコード						
2	6e91f9c25d028620bfa2e967f8253654212c276	Igor Miner	2016/11/24 1:04	555169	90267	464902						
3	7f1f8bdfef1043871c5eac2ec02efc41e0de5e53	GitHub	2016/11/23 23:44	555169	90267	464902						
4	cc92da0d577973cacaf52fb6c558336b6a738423	Peter Bacon Darwin	2016/11/23 14:42	555161	90267	464894						
5	c9b15bfa41720aba49e673c536a68ca179bf66	Peter Bacon Darwin	2016/11/23 14:39	555120	90242	464878						
6	c54921008d0a5c23f5f7f1f759411f1bfef14a09	Micha? Go?F?biowski	2016/11/23 13:28	555119	90242	464877						
7	69f59f2d0166cc4aba2745c834f649024ca20fbf	GitHub	2016/11/23 12:52	555142	90265	464877						
8	f4fb6e0883a6a700dc4a246a813504550b65f1e9	Martin Staffa	2016/11/23 12:30	555029	90265	464764						
9	af18be1513762827ae5a71bdf043c2263545013b2	Martin Staffa	2016/11/23 12:30	554873	90113	464760						
10	05a9da73cbee70eabc63473004d71aa2e6d949a	Martin Staffa	2016/11/22 23:10	554865	90105	464760						
11	60035f591c025191919915e22ea5118a566d3f9	Georgios Kalpakas	2016/11/22 12:49	554862	90105	464757						
12	0af21a48e53045963f73637be93e18b688d95ff5	Georgios Kalpakas	2016/11/22 7:48	554863	90105	464758						
13	991a280e00aecd1d312e2955e356a795f9e3062	Peter Bacon Darwin	2016/11/21 14:51	554861	90105	464756						
14	fc4afid0c1b317c38fc81aeb0d3121a6474292060	Peter Bacon Darwin	2016/11/21 13:27	554857	90101	464756						
15	d5109e26e4855270bb9f439f62c7312c071612f	Peter Bacon Darwin	2016/11/21 13:27	554730	90101	464629						
16	cf3c736b61cb9874fd5034f137094f405f947d8	Georgios Kalpakas	2016/11/21 12:21	554730	90101	464629						
17	b343e7a56013d78ab640eeac4c945896e60c9121	Georgios Kalpakas	2016/11/21 10:55	554730	90101	464629						
18	5f5b15f51dbf146a09ab744e6b0d3e6da9bf8ac	Georgios Kalpakas	2016/11/21 10:51	554730	90101	464629						
19	af144c1c0229e75307ca49db71dafef7e67b67b69f9	Peter Bacon Darwin	2016/11/18 22:04	554728	90101	464627						
20	d79f9b993056a2d128af97a2280644c77c7df	Peter Bacon Darwin	2016/11/18 13:51	553600	90101	463499						
21	f56926ae77d6e440ff9d04836799f9b9e96493bb	Peter Bacon Darwin	2016/11/18 13:51	553602	90101	463501						
22	17e98ace4816b13048a7af567df12a2b6e251a01	Peter Bacon Darwin	2016/11/18 11:28	553579	90101	463478						
23	752c989152c7fb9e5638a9a809f3d867339420d	Peter Bacon Darwin	2016/11/18 11:28	553166	90101	463005						
24	789790fee4d6c5bf15d5bf8ecb0ccf6edd36fb3	Peter Bacon Darwin	2016/11/17 23:38	551682	90101	461581						
25	fd1a93a4b01291492b61c340a873068f21068ccf	Peter Bacon Darwin	2016/11/17 23:38	551656	90075	461581						
26	7d24af110a9bc2a2175e8e99868da74a36eb2ae5	Martin Staffa	2016/11/17 21:17	551656	90075	461581						
27	ac7a2daf2e27bfb17cbb0117d1c9281e0b2c0bf	Martin Staffa	2016/11/17 18:23	551655	90075	461580						
28	286cfce40c25e943b1fa49a0a2c7240707af0ffa	Peter Bacon Darwin	2016/11/16 13:46	551646	90075	461571						

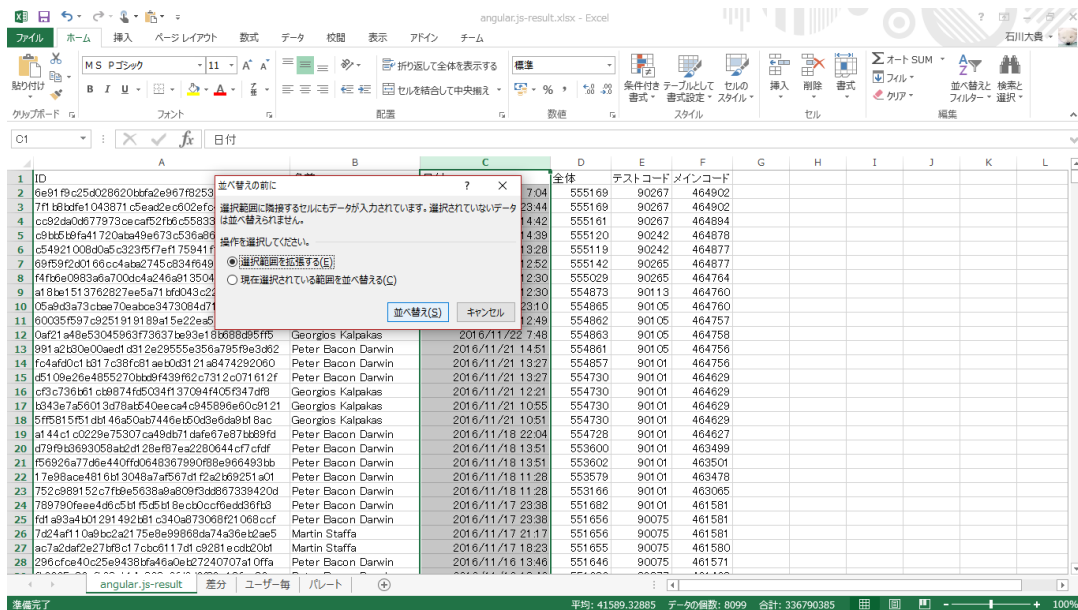
図 4.4 手順 1

新しいコミットから順に並んでいる．コミットごとの追加行数を調べたいので，後のコミットから前のコミットを引いて差分を出せばよい．そのためにまず，古いコミットから順番に並べる．



ID	名前	日付	全体	テストコード	メインコード
6e91f9c25d028620bfa2e967f8253654212c278	Igor Minar	2016/11/24 7:04	555169	90267	464902
7f1b6bdf1043871c5ea22ec02efc41e0a5e553	GitHub	2016/11/23 23:44	555169	90267	464902
cc92da0d577973ceca521b6c558336fa738423	Peter Bacon Darwin	2016/11/23 14:42	555161	90267	464894
c9b65b6fa41720aba49e673c536ae86ca71b166	Peter Bacon Darwin	2016/11/23 14:39	555120	90242	464878
c54921008d0a5c323f57f7e1f175941f1bfef14a09	Micha? Go?F?i?blowski	2016/11/23 13:28	555119	90242	464877
69f59f2d0166cc4aba2745c834f649024ca20f0f	GitHub	2016/11/23 12:52	555142	90265	464877
f4f6e0883a6a700dc4a24a6a13504550b5f1e9	Martin Staffa	2016/11/23 12:30	555029	90265	464764
af18be1513762827ee5a71bf043c2263545013b2	Martin Staffa	2016/11/23 12:30	554873	90113	464760
05a9d3a73cbae70eabce3473064d71aa2ed348a	Martin Staffa	2016/11/22 23:10	554865	90105	464760
60035f597c9251919189af5e22ae5118a566d3f5	Georgios Kalpakas	2016/11/22 12:49	554862	90105	464757
0af21a4be5304596373637be93e18b688d85ff5	Georgios Kalpakas	2016/11/22 7:48	554863	90105	464758
991a2b0e00aed1d312e2955e356a795f9e3d62	Peter Bacon Darwin	2016/11/21 14:51	554861	90105	464756
fc4af9d0c1b17c38f081ae0d3121a474292060	Peter Bacon Darwin	2016/11/21 13:27	554857	90101	464756
c5109e26e4855270bb8f439f62c7312c071612f	Peter Bacon Darwin	2016/11/21 13:27	554730	90101	464629
c3c736b61c8974f5034f137084f406347d0b	Georgios Kalpakas	2016/11/21 12:21	554730	90101	464629
b043e7a56013d78a5f40eacac84589e60c9121	Georgios Kalpakas	2016/11/21 10:55	554730	90101	464629
5ff5f15f51d146a50ab7446ef0d3e6da9f18ac	Georgios Kalpakas	2016/11/21 10:51	554730	90101	464629
af144c1c0229e75307ca49db71dafef7eb7bb89fd	Peter Bacon Darwin	2016/11/18 22:04	554728	90101	464627
d79f9b93059a2d2f28e6f7ea2280644c7cfdf	Peter Bacon Darwin	2016/11/18 13:51	553600	90101	463499
f56926a77d6e440ff0d648367990f9b9e66493bb	Peter Bacon Darwin	2016/11/18 13:51	553602	90101	463501
17e98ace4816b13048a7af567df12a2b69251a01	Peter Bacon Darwin	2016/11/18 11:28	553579	90101	463478
752c889152c7f8e5638a9a09f3d8807339420d	Peter Bacon Darwin	2016/11/18 11:28	553166	90101	463065
78979f0ee446c5b1f5d518ec0cc0f6ed36fb3	Peter Bacon Darwin	2016/11/17 23:38	551682	90101	461581
fd1a93a4b01291492b1c340a87306f921068c0f	Peter Bacon Darwin	2016/11/17 23:38	551656	90075	461581
7d24af110a9bc2a2175e8e99868da74a36e12ae5	Martin Staffa	2016/11/17 21:17	551656	90075	461581
ac7a2daf2e27bf8c17c0cd117f1c9281cd020b1	Martin Staffa	2016/11/17 18:23	551655	90075	461580
286cfce40c25e9438bfa46a0eb27240707a10ffa	Peter Bacon Darwin	2016/11/16 13:46	551646	90075	461571

図 4.5 手順 2



ID	名前	日付	全体	テストコード	メインコード
6e91f9c25d028620bfa2e967f8253654212c278	Igor Minar	2016/11/24 7:04	555169	90267	464902
7f1b6bdf1043871c5ea22ec02efc41e0a5e553	GitHub	2016/11/23 23:44	555169	90267	464902
cc92da0d577973ceca521b6c558336fa738423	Peter Bacon Darwin	2016/11/23 14:42	555161	90267	464894
c9b65b6fa41720aba49e673c536ae86ca71b166	Peter Bacon Darwin	2016/11/23 14:39	555120	90242	464878
c54921008d0a5c323f57f7e1f175941f1bfef14a09	Micha? Go?F?i?blowski	2016/11/23 13:28	555119	90242	464877
69f59f2d0166cc4aba2745c834f649024ca20f0f	GitHub	2016/11/23 12:52	555142	90265	464877
f4f6e0883a6a700dc4a24a6a13504550b5f1e9	Martin Staffa	2016/11/23 12:30	555029	90265	464764
af18be1513762827ee5a71bf043c2263545013b2	Martin Staffa	2016/11/23 12:30	554873	90113	464760
05a9d3a73cbae70eabce3473064d71aa2ed348a	Martin Staffa	2016/11/22 23:10	554865	90105	464760
60035f597c9251919189af5e22ae5118a566d3f5	Georgios Kalpakas	2016/11/22 12:49	554862	90105	464757
0af21a4be5304596373637be93e18b688d85ff5	Georgios Kalpakas	2016/11/22 7:48	554863	90105	464758
991a2b0e00aed1d312e2955e356a795f9e3d62	Peter Bacon Darwin	2016/11/21 14:51	554861	90105	464756
fc4af9d0c1b17c38f081ae0d3121a474292060	Peter Bacon Darwin	2016/11/21 13:27	554857	90101	464756
c5109e26e4855270bb8f439f62c7312c071612f	Peter Bacon Darwin	2016/11/21 13:27	554730	90101	464629
c3c736b61c8974f5034f137084f406347d0b	Georgios Kalpakas	2016/11/21 12:21	554730	90101	464629
b043e7a56013d78a5f40eacac84589e60c9121	Georgios Kalpakas	2016/11/21 10:55	554730	90101	464629
5ff5f15f51d146a50ab7446ef0d3e6da9f18ac	Georgios Kalpakas	2016/11/21 10:51	554730	90101	464629
af144c1c0229e75307ca49db71dafef7eb7bb89fd	Peter Bacon Darwin	2016/11/18 22:04	554728	90101	464627
d79f9b93059a2d2f28e6f7ea2280644c7cfdf	Peter Bacon Darwin	2016/11/18 13:51	553600	90101	463499
f56926a77d6e440ff0d648367990f9b9e66493bb	Peter Bacon Darwin	2016/11/18 13:51	553602	90101	463501
17e98ace4816b13048a7af567df12a2b69251a01	Peter Bacon Darwin	2016/11/18 11:28	553579	90101	463478
752c889152c7f8e5638a9a09f3d8807339420d	Peter Bacon Darwin	2016/11/18 11:28	553166	90101	463065
78979f0ee446c5b1f5d518ec0cc0f6ed36fb3	Peter Bacon Darwin	2016/11/17 23:38	551682	90101	461581
fd1a93a4b01291492b1c340a87306f921068c0f	Peter Bacon Darwin	2016/11/17 23:38	551656	90075	461581
7d24af110a9bc2a2175e8e99868da74a36e12ae5	Martin Staffa	2016/11/17 21:17	551656	90075	461581
ac7a2daf2e27bf8c17c0cd117f1c9281cd020b1	Martin Staffa	2016/11/17 18:23	551655	90075	461580
286cfce40c25e9438bfa46a0eb27240707a10ffa	Peter Bacon Darwin	2016/11/16 13:46	551646	90075	461571

図 4.6 手順 3

A	B	C	D	E	F	G	H	I	J	K	L
ID	名前	日付	全体	テストコード	メインコード	テストコードの差分	メインコードの差分				
c9c176a53bf632ca2bf06ed27382ab72ac21d45d	Adam Abrons	2010/1/6 0:36	14559	3623	10836						
fac0e698a55ade9c8a18f9400c0b0f118de4e0	Adam Abrons	2010/1/6 0:57	14579	3623	10856						
19bbe030ba012b6fca835c1d7e038904a2b84b	Adam Abrons	2010/1/12 17:34	14580	3623	10857						
ba9ee4f0c8f86cd869d566b43f8d129642d01351	Misko Hevery	2010/1/26 4:30	26792	3584	23208						
ac2540fd581f35e8f79240d827d2252da5798c3a2	Misko Hevery	2010/1/26 7:49	26794	3583	23211						
88384854c208f87507c273218fea8500f93801d6	Adam Abrons	2010/1/26 19:27	26806	3583	23223						
f5055c6530ffdb4436f52afe8c52ab6c2f2f14a	Misko Hevery	2010/1/26 23:59	26806	3583	23223						
4ea337f0c83b0b140dd8cf326c134f716875b65e	Misko Hevery	2010/1/27 0:25	26806	3583	23223						
a9c18276416fee42466c4b632757276277fab6d	Misko Hevery	2010/1/29 6:11	26869	3624	23245						
a6f2d562899299fa8f8c4a4b4177df4e49c9c	Misko Hevery	2010/1/29 6:15	26891	3624	23267						
2280411713bf061b0d30b0a1f3531a34272f3b25	Misko Hevery	2010/1/29 18:15	26969	3624	23345						
88b415a3dccefe8052a0590e132c6e4393b4	Misko Hevery	2010/1/29 18:20	26970	3624	23346						
302472f4fa50f95085abf86b890bedf3806973	Misko Hevery	2010/2/4 19:12	26971	3625	23346						
5dd43b65e73ca1708e7f8d5084633b02266a79a	Misko Hevery	2010/2/4 19:45	26981	3633	23348						
1da18e73a4d09b2a1ace92a4094eeba014eb7dc4	Misko Hevery	2010/2/4 21:27	27024	3655	23369						
9f9f19c42f0885e39870195fab8ce2a2621119b7	Misko Hevery	2010/2/4 22:02	27033	3652	23381						
c251fab40291183c8e50ea0fab23c30341cc72d3	Misko Hevery	2010/2/4 23:04	27038	3657	23381						
5eb440c22b67bf069c1419395462b0bf0ca0c23	Misko Hevery	2010/2/4 23:12	27038	3657	23381						
6d75afe6d2eac2bb412becdf0971ca6031eaab4	Misko Hevery	2010/2/5 22:13	27042	3665	23377						
9d566a0cd0225685efbf92c19520b6857628d32	Misko Hevery	2010/2/5 22:41	27068	3667	23401						
799d72931a5a01de74bb68d8a6638cd57ce315	Misko Hevery	2010/2/9 21:13	27071	3666	23405						
9d566fe98c3fa0091efef4ad0178f063da747fc	Misko Hevery	2010/2/9 22:59	27081	3676	23405						
12a2a089b6c31c8ff176c2483f8f95caae47f1afb	Misko Hevery	2010/2/11 17:57	27081	3676	23405						
6cc946413622f1ce97997849e73a06a00f76fd	Misko Hevery	2010/2/12 22:17	27084	3703	23391						
7c8a2ccb8d7b95419dfbe9151928cd2533550e3	Misko Hevery	2010/2/12 22:25	27106	3703	23403						
3f9e2ab8bfcd12cb7df74b0d38cecf2ee4ac84a	Misko Hevery	2010/2/13 3:39	27232	3758	23474						
7e14df90516a41ff1903cc44fa3389710f15556	Misko Hevery	2010/2/18 0:05	27235	3759	23476						

図 4.7 手順 4

テストコードとメインコードの1行目はそのまま隣のセルにコピーする．コピーしたセルの下に後のコミットから前のコミットを引いた結果を出力する．

A	B	C	D	E	F	G	H	I	J
ID	名前	日付	全体	テストコード	メインコード	テストコードの差分	メインコードの差分		
c9c176a53bf632ca2bf06ed27382ab72ac21d45d	Adam Abrons	2010/1/6 0:36	14559	3623	10836				
fac0e698a55ade9c8a18f9400c0b0f118de4e0	Adam Abrons	2010/1/6 0:57	14579	3623	10856				
19bbe030ba012b6fca835c1d7e038904a2b84b	Adam Abrons	2010/1/12 17:34	14580	3623	10857				
ba9ee4f0c8f86cd869d566b43f8d129642d01351	Misko Hevery	2010/1/26 4:30	26792	3584	23208				
ac2540fd581f35e8f79240d827d2252da5798c3a2	Misko Hevery	2010/1/26 7:49	26794	3583	23211				
88384854c208f87507c273218fea8500f93801d6	Adam Abrons	2010/1/26 19:27	26806	3583	23223				
f5055c6530ffdb4436f52afe8c52ab6c2f2f14a	Misko Hevery	2010/1/26 23:59	26806	3583	23223				
4ea337f0c83b0b140dd8cf326c134f716875b65e	Misko Hevery	2010/1/27 0:25	26806	3583	23223				
a9c18276416fee42466c4b632757276277fab6d	Misko Hevery	2010/1/29 6:11	26869	3624	23245				
a6f2d562899299fa8f8c4a4b4177df4e49c9c	Misko Hevery	2010/1/29 6:15	26891	3624	23267				
2280411713bf061b0d30b0a1f3531a34272f3b25	Misko Hevery	2010/1/29 18:15	26969	3624	23345				
88b415a3dccefe8052a0590e132c6e4393b4	Misko Hevery	2010/1/29 18:20	26970	3624	23346				
302472f4fa50f95085abf86b890bedf3806973	Misko Hevery	2010/2/4 19:12	26971	3625	23346				
5dd43b65e73ca1708e7f8d5084633b02266a79a	Misko Hevery	2010/2/4 19:45	26981	3633	23348				
1da18e73a4d09b2a1ace92a4094eeba014eb7dc4	Misko Hevery	2010/2/4 21:27	27024	3655	23369				
9f9f19c42f0885e39870195fab8ce2a2621119b7	Misko Hevery	2010/2/4 22:02	27033	3652	23381				
c251fab40291183c8e50ea0fab23c30341cc72d3	Misko Hevery	2010/2/4 23:04	27038	3657	23381				
5eb440c22b67bf069c1419395462b0bf0ca0c23	Misko Hevery	2010/2/4 23:12	27038	3657	23381				
6d75afe6d2eac2bb412becdf0971ca6031eaab4	Misko Hevery	2010/2/5 22:13	27042	3665	23377				
9d566a0cd0225685efbf92c19520b6857628d32	Misko Hevery	2010/2/5 22:41	27068	3667	23401				
799d72931a5a01de74bb68d8a6638cd57ce315	Misko Hevery	2010/2/9 21:13	27071	3666	23405				
9d566fe98c3fa0091efef4ad0178f063da747fc	Misko Hevery	2010/2/9 22:59	27081	3676	23405				
12a2a089b6c31c8ff176c2483f8f95caae47f1afb	Misko Hevery	2010/2/11 17:57	27081	3676	23405				
6cc946413622f1ce97997849e73a06a00f76fd	Misko Hevery	2010/2/12 22:17	27084	3703	23391				
7c8a2ccb8d7b95419dfbe9151928cd2533550e3	Misko Hevery	2010/2/12 22:25	27106	3703	23403				
3f9e2ab8bfcd12cb7df74b0d38cecf2ee4ac84a	Misko Hevery	2010/2/13 3:39	27232	3758	23474				
7e14df90516a41ff1903cc44fa3389710f15556	Misko Hevery	2010/2/18 0:05	27235	3759	23476				

図 4.8 手順 5

手順 6 の画像のように、赤矢印の方向に引っ張る。青丸で囲んだセルの右下をダブルクリックすることで、数式を書くのは一度で済む。

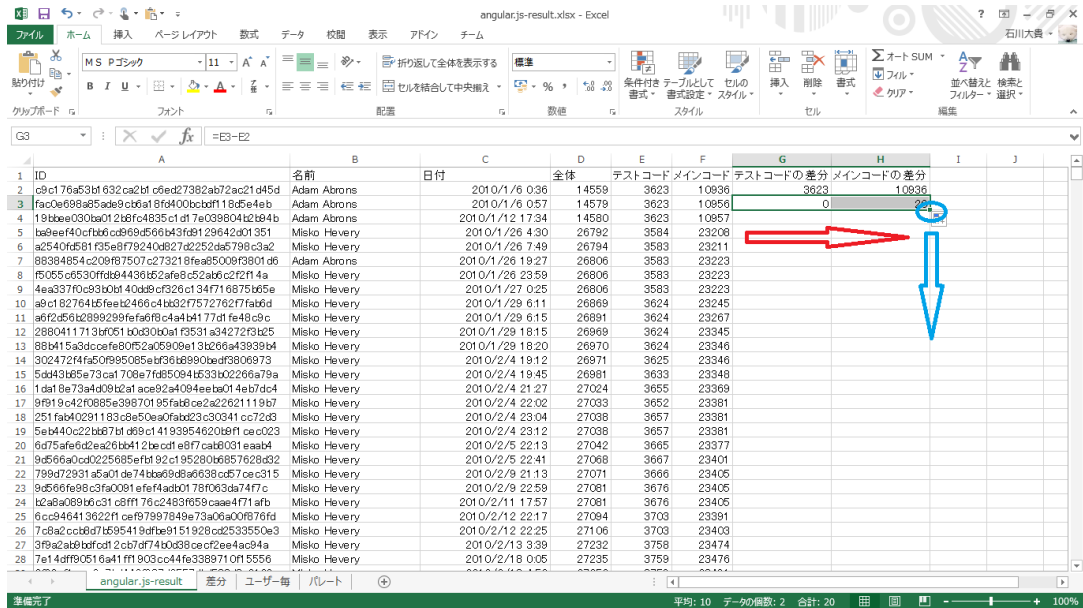


図 4.9 手順 6

手順 7 の画像のように、一番下まで全てのコミットの差分が出せればよい。

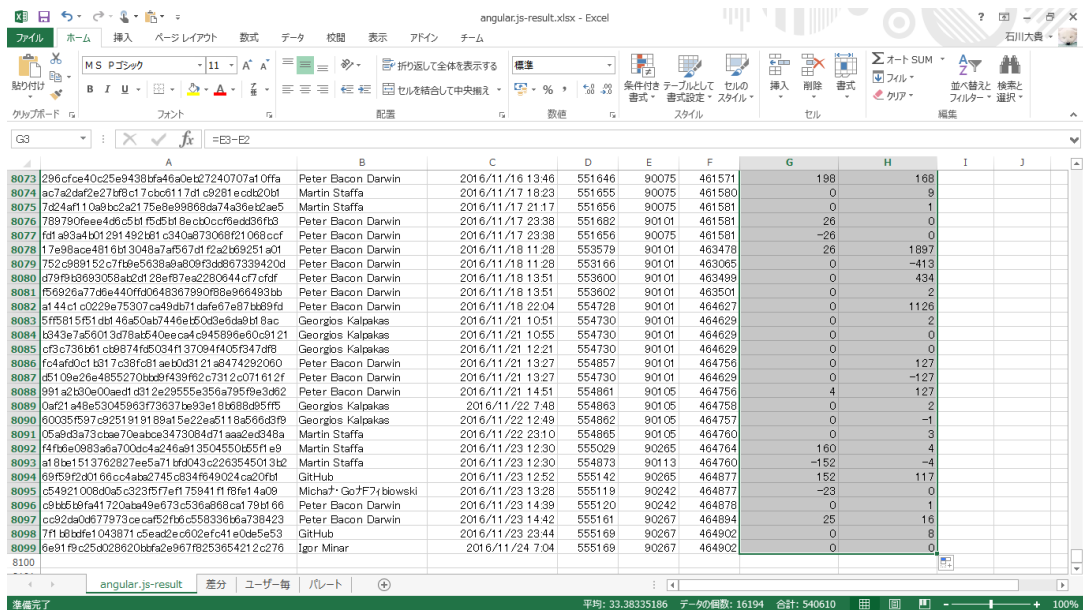


図 4.10 手順 7

作業を行いやすくするため、新しいシートに名前、テストコードとメインコードの差分を移動させる。

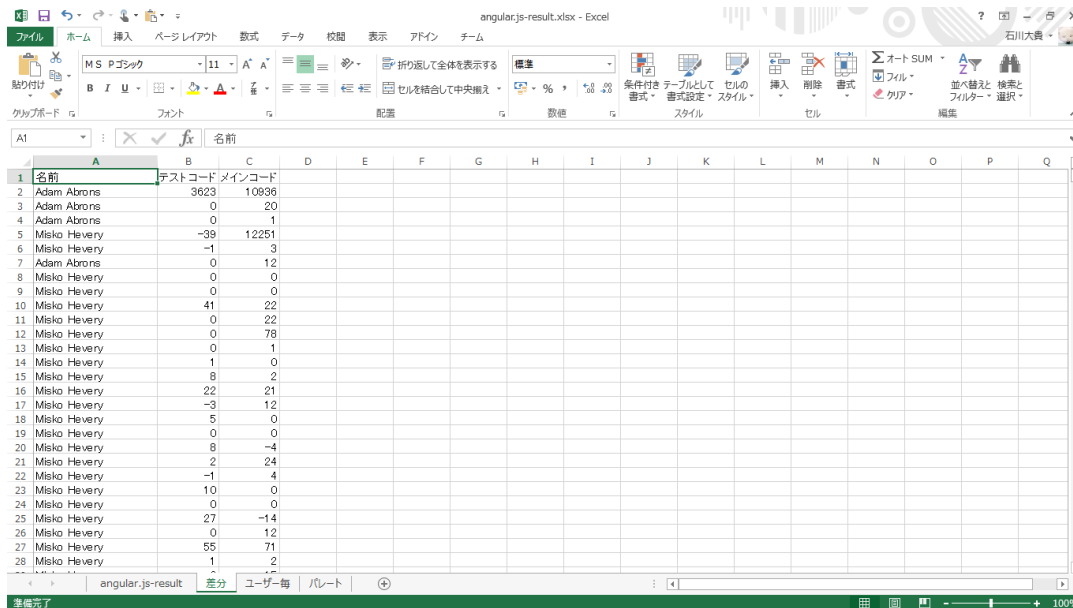


図 4.11 手順 8

1 列目の見出しにフィルターを付ける。

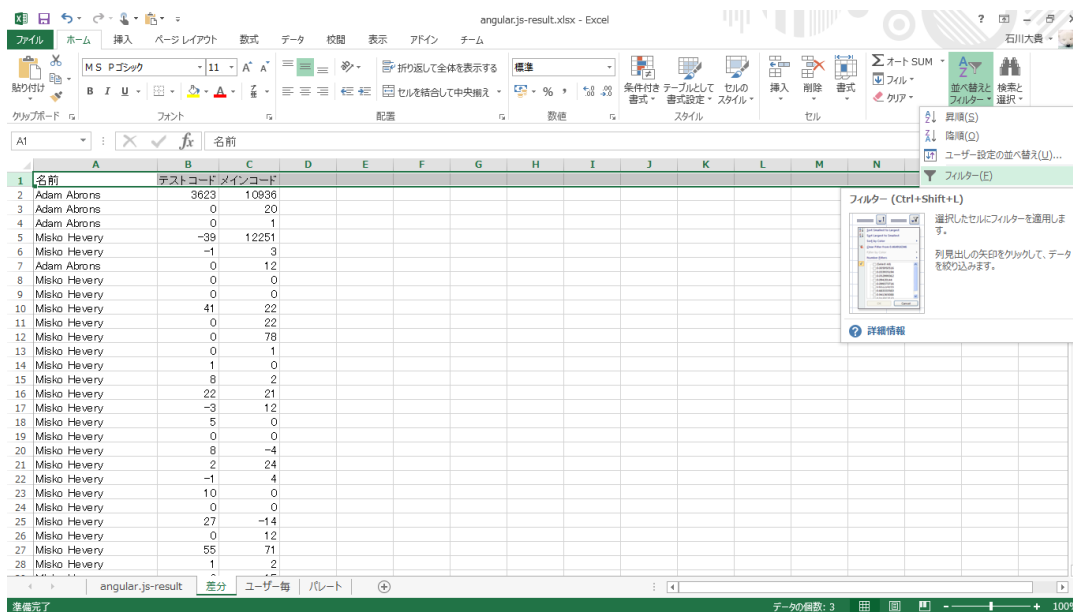


図 4.12 手順 9

1つの名前ごとにフィルターをかける。

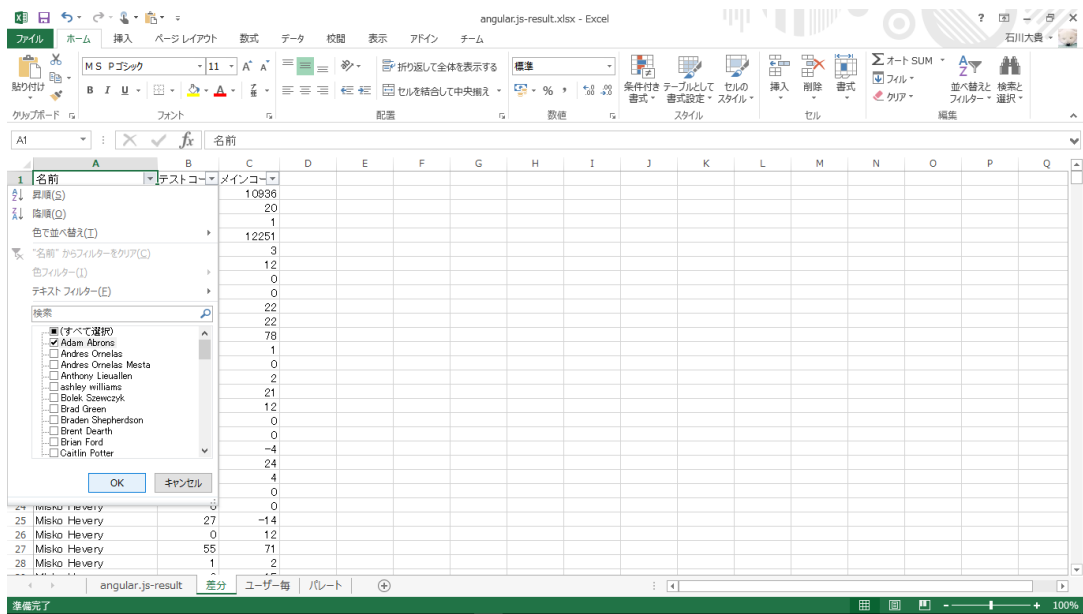


図 4.13 手順 10

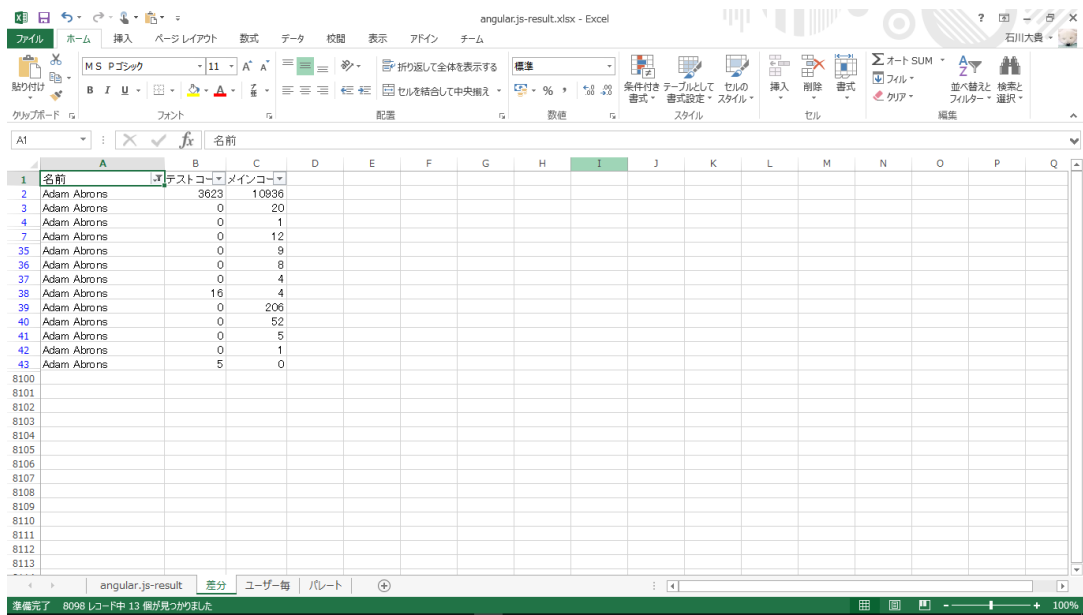


図 4.14 手順 11

テストコードとメインコードの下の空のセルを選択する．オート SUM 機能を使いそれぞれの合計を出力する．

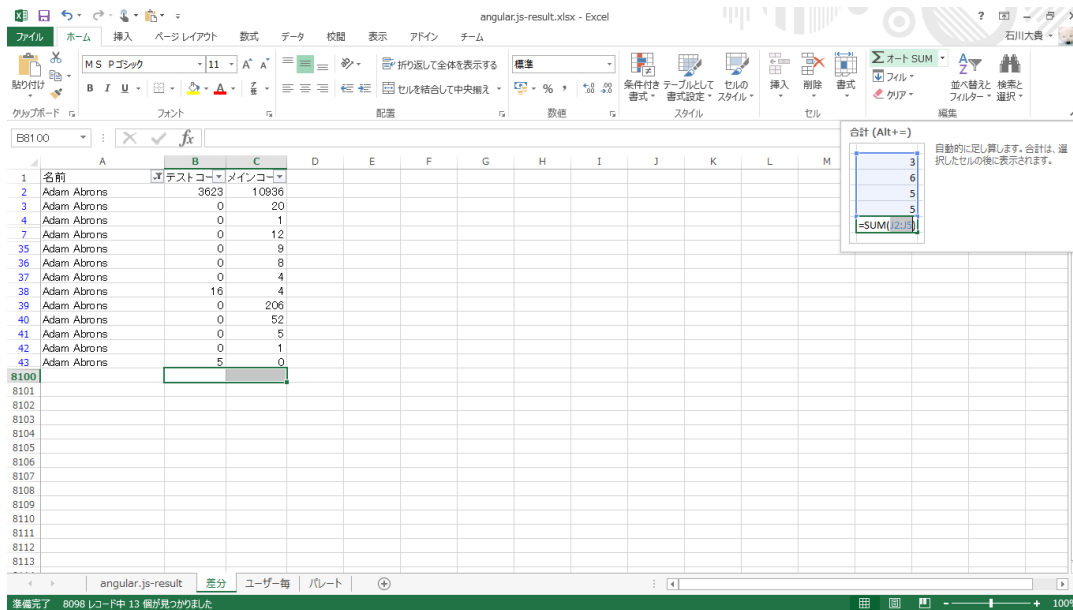


図 4.15 手順 12

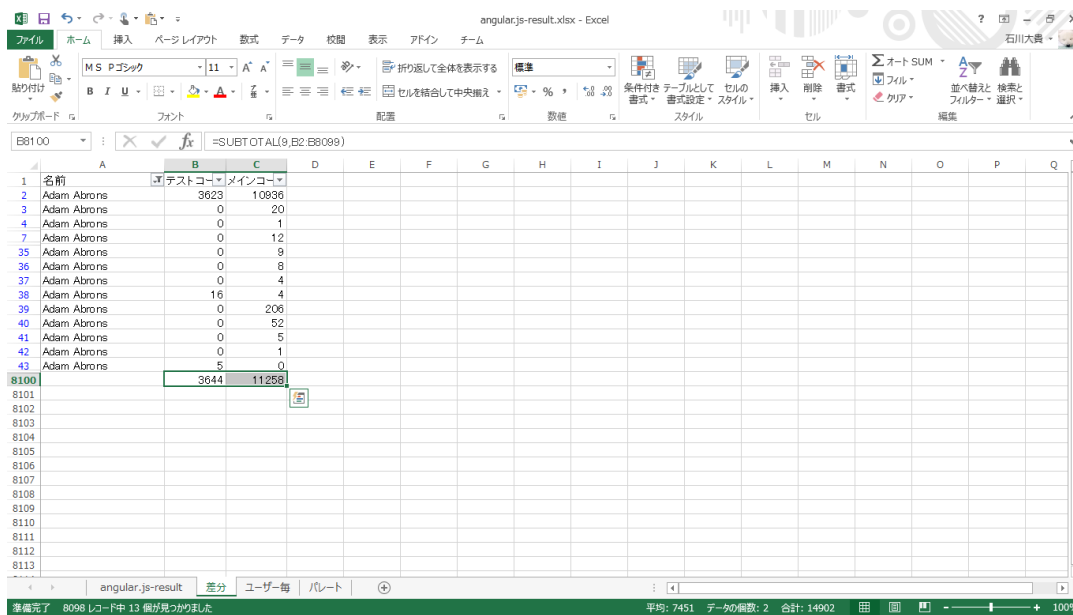


図 4.16 手順 13

新しいシートに名前と出力した合計を記録する．これを全ての名前を記録するまで行う．

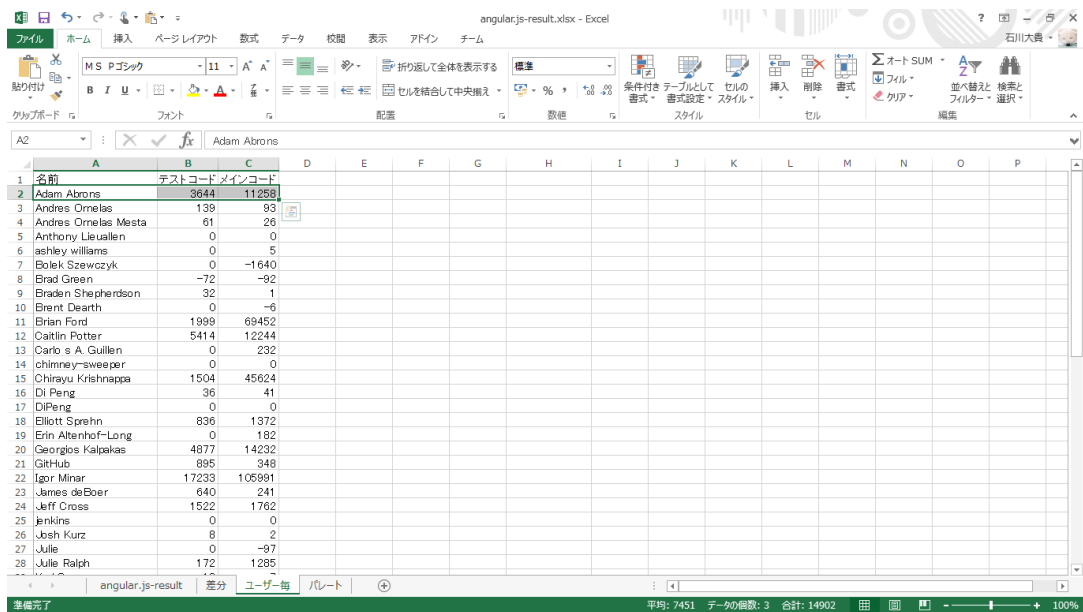


図 4.17 手順 14

全て記録できたら，テストコードの列を選択し，降順で並べ替える．

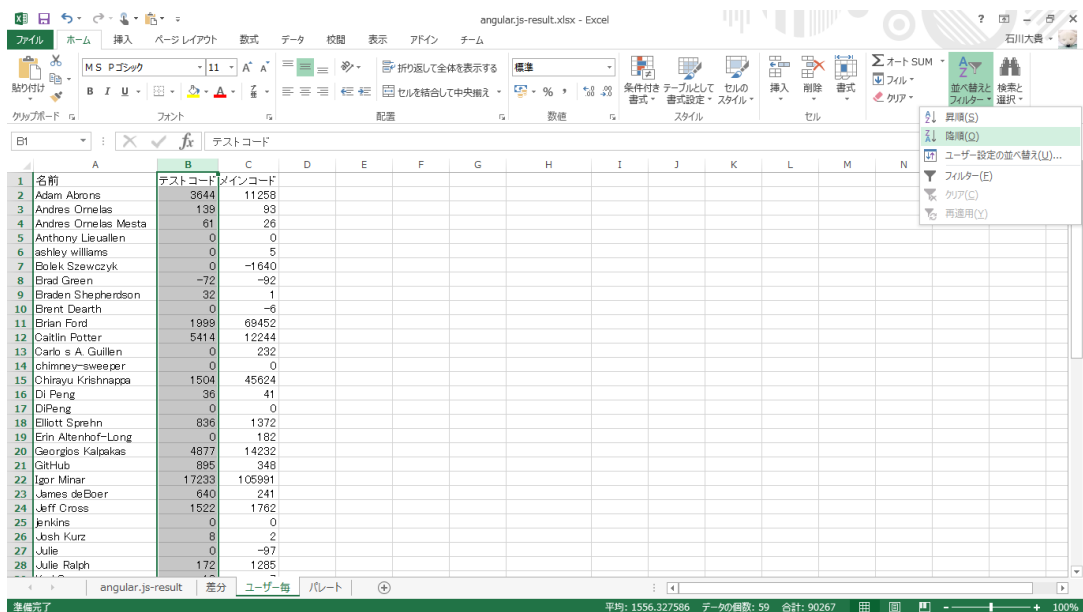


図 4.18 手順 15

テストコードの数字が 1 以上である範囲のセルを選択し，データの個数と合計を新しいシートに記録する．データの個数は参加者の人数となり，合計は追加されたコード行数の合計となる．

名前	テストコード	メインコード
Igor Minar	17233	105991
Peter Bacon Darwin	12717	148054
Matias Niemelä	9658	1659
Misko Hevery	8858	33767
Caitlin Potter	5414	12244
Voja Jina	5044	3557
Georgios Kalpakas	4877	14232
Martin Staffa	4266	1603
Adam Abrons	3644	11259
Lucas Galfaso	2707	856
Pete Bacon Darwin	2639	3000
Tobias Bosch	2426	1990
Brian Ford	1999	69452
Jeff Cross	1522	1762
Chirayu Krishnappa	1504	45624
Lucas Mirelmann	1294	3273
Paweł Kozłowski	1006	477
GitHub	895	348
Elliott Spreh	836	1372
James deBoer	640	241
Michał Goźdź	288	1602
Ken Sheedlo	195	532
Julie Ralph	172	1285
Andres Omeias	139	93
rodyhaddad	123	98
Shyam Seshadri	101	31
Andres Omeias Mesta	61	26

図 4.19 手順 16

名前	テストコード	メインコード
Karl Seamon	13	7
Josh Kurz	8	2
Anthony Lieuallen	0	0
ashley williams	0	5
Bolek Szewczyk	0	-1640
Brent Dearth	0	-6
Carlo s A. Guillen	0	232
chimney-sweeper	0	0
DiPeng	0	0
Erin Altenhof-Long	0	182
Jenkins	0	0
Julie	0	-97
lefos887	0	0
Lefteris Paraskevas	0	0
linclark	0	1
Luther Goh	0	0
Marcy Sutton	0	98
Martin Probst	0	1942
Mitko Hevery	0	24
naomi black	0	-6
naomiblack	0	1
phil	0	0
Rado Kirov	0	5
Rich Snapp	0	0
Richard Littauer	0	0
unknown	0	4
Michał Goźdź	-32	-1891
Brad Green	-72	-92

図 4.20 手順 17

メインコードでも同様に、データの個数と合計を記録する。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	人数		コード行数															
2	テストコード	32	90371															
3	メインコード	42	468634															
4																		
5																		
6																		
7																		
8																		
9																		
10																		
11																		
12																		
13																		
14																		
15																		
16																		
17																		
18																		
19																		
20																		

図 4.21 手順 18

次に、メインコードとテストコードでそれぞれパレートの法則が成り立つのか検証してみる。80% 以上のコード行数を書いているのが 20% 以下のユーザであるときパレートの法則が成り立つとする。

まず、80% のコード行数を計算する。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	人数		コード行数	8割の行数														
2	テストコード	32	90371	72296.8														
3	メインコード	42	468634	374907.2														
4																		
5																		
6																		
7																		
8																		
9																		
10																		
11																		
12																		
13																		
14																		
15																		
16																		
17																		
18																		
19																		
20																		
21																		
22																		
23																		
24																		
25																		
26																		
27																		
28																		

図 4.22 手順 19

前のシートに戻り、複数のセル選択時に表示される合計の数値を確認して、80% のコード行数を超えるところまでセルを選択する。選択した範囲のデータの個数を記録し、これを上位の貢献者の人数とする。また、この時選択した範囲にいた名前のユーザを上位の貢献者たちとし記録しておく。

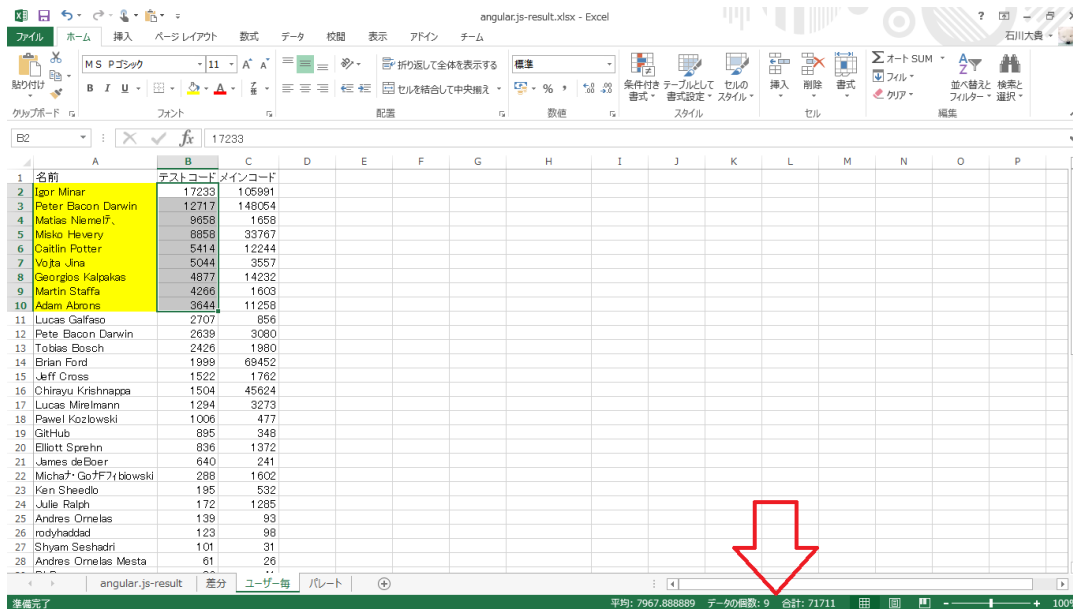


図 4.23 手順 20

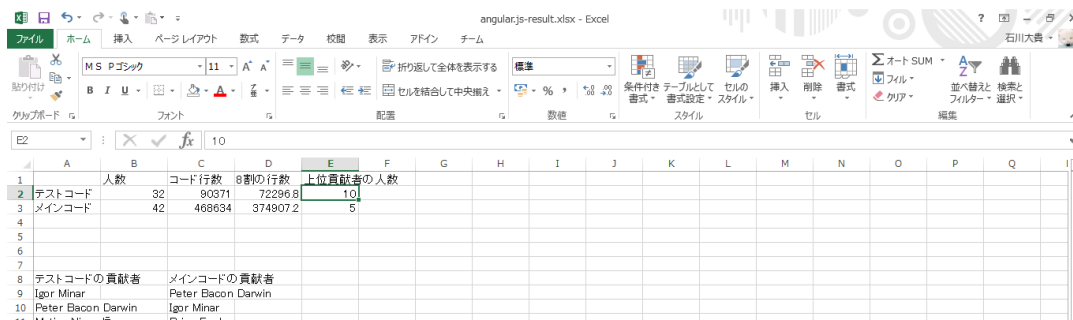


図 4.24 手順 21

上位の貢献者の人数 ÷ 参加者の人数を × 100 を計算して、20% を下回っていればパレートの法則は成り立つものとする。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
		人数	コード行数	割合の行数	上位貢献者の人数												
2	テストコード	32	90371	72296.8	10		31.25										
3	メインコード	42	468634	374907.2	5		11.90476										
8	テストコードの貢献者		メインコードの貢献者														
9	Igor Minar		Peter Bacon Darwin														
10	Peter Bacon Darwin		Igor Minar														
11	Matias Niemelä		Brian Ford														

図 4.25 手順 22

ここまで集計したデータを元に、テストコードとメインコードの上位貢献での同じ人物はいるのか、パレートの法則が成り立つ場合とそうでない場合の違い等を比べてる。

第 5 章

結果・考察

今回 22 件のプロジェクトを調査した．プロジェクトごとにメンバがそれぞれ追加したコード行数を集計した．集計したデータの一覧を次に記す．

表 5.1 angular/angular.js - 1

名前	テスト	本体
Adam Abrons	3644	11258
Andres Ornelas	139	93
Andres Ornelas Mesta	61	26
Anthony Lieuallen	0	0
ashley williams	0	5
Bolek Szewczyk	0	-1640
Brad Green	-72	-92
Braden Shepherdson	32	1
Brent Dearth	0	-6
Brian Ford	1999	69452
Caitlin Potter	5414	12244
Carlo s A. Guillen	0	232
chimney-sweeper	0	0
Chirayu Krishnappa	1504	45624
Di Peng	36	41
DiPeng	0	0
Elliott Sprehn	836	1372
Erin Altenhof-Long	0	182
Georgios Kalpakas	4877	14232
GitHub	895	348
Igor Minar	17233	105991
James deBoer	640	241
Jeff Cross	1522	1762
jenkins	0	0
Josh Kurz	8	2
Julie	0	-97
Julie Ralph	172	1285
Karl Seamon	13	7
Ken Sheedlo	195	532
lefos987	0	0

表 5.2 angular/angular.js - 2

名前	テスト	本体
Lefteris Paraskevas	0	0
linclark	0	1
Lucas Galfaso	2707	856
Lucas Mirelmann	1294	3273
Luther Goh	0	0
Marcy Sutton	0	98
Martin Probst	0	1942
Martin Staffa	4266	1603
Matias Niemel ẽ、	9658	1658
Micha ɶ・Go ɶF ɶi biowski	288	1602
Micha ɶ・Go ɶて冀 iowski	-32	-1891
Misko Hevery	8858	33767
Mi ɶ。ko Hevery	0	24
naomi black	0	-6
naomiblack	0	1
Pawel Kozlowski	1006	477
Pete Bacon Darwin	2639	3080
Peter Bacon Darwin	12717	148054
phil	0	0
Rado Kirov	0	5
Rich Snapp	0	0
Richard Littauer	0	0
rodyhaddad	123	98
Shahar Talmi	24	1593
Shyam Seshadri	101	31
Tobias Bosch	2426	1980
unknown	0	4
Vojta Jina	5044	3557

表 5.3 caolan/async

名前	テスト	本体
Alex Early	-702	3396
Alex Ianus	0	32
Alexander Early	-2397	23834
Allan Carroll	4	0
Beau Gunderson	124	117
Caolan McMahon	3043	4904
GitHub	0	102
Graeme Yeates	-159	3215
Hubert Argasinski	0	334
MaZderMind	24	14
megawac	63	-1
Peter	0	3

表 5.4 jashkenas/backbone - 1

名前	テスト	本体
Adam Krebs	3144	181
Aitor Guevara Escalante	29	22
Andrei Bocan	0	17
Andrew Schaaf	0	5
Brad Dunbar	861	341
Casey Foster	210	39
Dmitry Baranovskiy	0	73
dxgriffiths	0	23
Elijah Insua	0	12
F ㊦ lix	0	0
F ㊦ lix Horro Pita	0	0
GitHub	24140	16144
Graeme	15	11
Graeme Yeates	-1	6
Harry Wolff	0	0
Jamie Rolfs	51	4
Jason Davies	0	-56
Jeremy Ashkenas	18467	15254
Joshua Peek	97	49
Justin Ridgewell	24	65
Kris Jordan	13	3
Maksim Horbachevsky	17	1
Matt	9	-1
michalkot	11	-4
Nick Fitzgerald	90	13
Niels Sandholt Busch	0	-1
Ofer Nave	0	0
Ore Landau	0	2
Paul Uithol	68	68
Raimonds Simanovskis	10	0

表 5.5 jashkenas/backbone - 2

Sam Breed	9	-1
Sam Stephenson	22	8
Samuel Clay	0	0
Ted Han	0	1
Tim Branyen	0	-3
Tim Griesser	992	12
Ville Lautanala	2	1
Will Moffat	0	5

表 5.6 bower/bower

名前	テスト	本体
Adam Stankiewicz	5241	7472
addyosmani	0	0
Alireza Bashiri	72	26
Andre Cruz	267	294
Andre 7・Cruz	3853	5186
Andr 7 Cruz	1858	960
Ben	0	0
Ben Schwarz	0	4
Chris Aniszczyk	234	1868
Dan Heberden	36	16
David DeSandro	0	21
fat	205	224
GitHub	0	0
Jacob Thornton	197	242
Jaime Olmo	209	28
Mat Scales	0	60
Nicolas Gallagher	1593	779
Paul Irish	31	250
Rob Simpson	61	12
Rod Vagg	0	0
Sindre Sorhus	424	161
Vlad Filippov	0	19

表 5.7 adobe/brackets

名前	テスト	本体
名前	テスト	本体
Adam Lehman	0	22
Arno Gourdol	218	291
Arun Bose	1364	7451
Arzhan Kinzhalin	638	3234
Bernhard Sirlinger	3628	1953
Bryan Chin	0	33
Chema	83	412
Dennis Kehrig	1510	7452
ficristo	0	204
GitHub	0	208
Glenn Ruehle	23083	51311
Ian Wehrman	329	60
Ingo Richter	4162	48910
Jason San Jose	5897	2708
Jochen Hagenstroem	0	522
julianasuh	383	659
jzhang	949	2227
Kevin Dangoor	31921	156722
Marcel Gerber	232	3247
Miguel Castillo	297	2533
Narciso Jaramillo	9755	29025
nethip	31	303
njadbe	0	3756
Pete Nyk $\bar{\tau}$, nen	268	-919
Peter Flynn	383	433
Peter Thiess	12955	-2590
Prashanth Nethi	-41	2445
pthiess	-41	422
Randy Edmunds	272	533
RaymondLim	1679	7327
Ryan Stewart	406	1792
Tom $\bar{\tau}$. s Malbr $\bar{\tau}$. n	1578	3158
wALF Utility	0	0
Yoko	98993	144542

表 5.8 jashkenas/coffee-script - 1

名前	テスト	本体
Adriano Bonat	0	1
alunny	0	0
Brody Berg	0	0
Chris Hoffman	0	8
Chris Lloyd	2	20
Colin Ross	0	0
Dan Holmsand	0	0
Daniel J. Pritchett	0	0
Dr Nic Williams	0	-1
Fabian Jakobs	0	4
Federico Builes	0	-1
Geoffrey Booth	242	222
Gerald Lewis	26	0
gfxmonk	2	126
GitHub	191	-4121
James Campos	0	7
Janne Hietam $\bar{\tau}$, ki	0	0
Jason Walton	627	1431
Jeremy Ashkenas	6489	29083
Joshua Peek	18	50
Marc H $\bar{\tau}$, fner	299	-13
matehat	50	144
Matt Lyon	0	4
Maxwell Krohn	46	26
Michael Ficarra	858	11207
Michael Smith	0	8
Nami-Doc	467	181
ngn	14	4
Sam Stephenson	32	5
Samuel Reis	0	0

表 5.9 jashkenas/coffee-script - 2

名前	テスト	本体
Satoshi Murakami	0	0
satyr	114	-429
Simon Lydell	1219	1555
Stan Angeloff	84	321
St 𐄂𐄂 phan Kochen	4	15
Tim Jones	0	151
Timothy Jones	123	206
Tim-Smart	0	17
Trevor Burnham	60	-9

表 5.10 Shopify/dashing

名前	テスト	本体
Daniel Beauchamp	55	16536
David Underwood	57	1381
Dylan Thacker-Smith	0	11
Francois Chagnon	187	39
GitHub	0	-2
John Tajima	0	5
pushmatrix	110	1740
Robert Postill	0	2
Wesley Ellis	0	197

表 5.11 plataformatec/devise

名前	テスト	本体
Carlos A. da Silva	3145	1644
Carlos Antonio da Silva	813	594
Cyril Mougél	0	74
Erich Kist	0	-91
George Guimaraes	0	4
George Guimarães	2	-4
Hugo Baraúna	0	0
Jacques Crocker	70	32
José Valim	3998	5590
Lucas Mazza	866	251
Marcelo Silveira	33	155
Rafael Mendonça Franca	47	62
Rodrigo Flores	189	353
Ulisses Almeida	73	187
Vasiliy Ermolovich	65	25
Vinicius Baggio	94	-35

表 5.12 discourse/discourse

名前	テスト	本体
Arpit Jalan	18	7791
Blake Erickson	0	-1
GitHub	0	19
Guo Xiang Tan	246	3348
Jeff Atwood	3	-16650
Matt Palmer	0	33
Neil Lalonde	1269	97646
Robin Ward	13366	229917
Rogier Hanol	738	53557
Sam	2445	47973
Sam Saffron	44	37787
Shawn Holmes	0	-40

表 5.13 visionmedia/express

名前	テスト	本体
Aaron Heckmann	0	10
Blake Embrey	0	1
ciaranj	0	756
Douglas Christopher Wilson	4694	3222
isaacs	0	0
Jonathan Ong	-629	-894
Linus Unneb $\bar{\tau}$, ck	-46	0
Matt Colyer	0	0
Nick Poulden	0	1
Rand McKinney	0	1
Roman Shtylman	172	458
TJ Holowaychuk	6535	-2132
visionmedia	0	8810

表 5.14 gruntjs/grunt

名前	テスト	本体
Ben Alman	2730	2727
Damon Oehlman	0	14
Feodor Fitsner	0	0
GitHub	0	0
Kyle Robinson Young	96	-6
Mickael Daniel	67	0
Scott Gonz $\bar{\tau}$. lez	0	3
Sindre Sorhus	0	7
Tyler Kellen	-346	-174
Vlad Filippov	5	-145
vladikoff	0	0

表 5.15 visionmedia/jade

名前	テスト	本体
Andreas Lubbe	-375	-11364
Forbes Lindesay	898	-4885
ForbesLindesay	407	10225
GitHub	17	-4252
hemanth.hm	0	0
Joshua Appelman	0	14
Timothy Gu	494	772
Tj Holowaychuk	2559	11802

表 5.16 jekyll/jekyll

名前	テスト	本体
Alfred Xing	52	216
Aman Gupta	0	-77
Antonin Hildebrand	58	48
Arnar Birgisson	0	4
Chris Van Pelt	0	92
Decider UI	0	0
GitHub	0	-1
Jack Danger Canty	0	59
Jeff Hodges	0	1
jekyllbot	2002	1406
Jordon Bedwell	237	538
Josh Nichols	3	61
Kris Brown	284	370
Lee Jarvis	0	2
Luismi Cavalle	8	0
Martin Vilcans	8	4
Matt Hall	0	38
Matt Rogers	610	2192
mreid	8	129
Nick Gerakines	0	82
Nick Quaranto	532	-586
Parker Moore	3481	17212
Pat Hawks	82	64
PJ Hyett	0	1
Robin Dupret	0	10
Ryan Tomayko	0	1
Tim Dysinger	0	2
Toby DiPasquale	0	74
Tom Kirchner	0	0
Tom Preston-Werner	1128	9036
Will Norris	142	43
蝶 蜷帛菅	0	32

表 5.17 jquery/jquery - 1

名前	テスト	本体
0 jeresig	4363	9437
adam j. sontag	0	0
Anton M	-13684	50
Ariel Flesler	14582	154
Brandon Aaron	1077	4545
brandonaaron	42	7
Carl F ゃ rstenberg	3	-2
Chris Talkington	0	28
Colin Snover	251	2768
Corey Frang	745	5529
Corey Jewett	0	29
Dan Heberden	48	0
Dave Methvin	3481	-8302
David Serduke	588	3872
Ed Engelhardt	0	4
Franck Marcia	0	332
Gilles van den Hoven	0	424
GitHub	-1	68
gnarf	8	0
Henri Wiechers	0	1
jaubourg	626	-153
John Resig	4893	-526
Jordan Boesch	-79	-22
Jo 7・n Zaefferer	0	16
Julian Aubourg	23	7
jzaefferer	25	-13
J ゃ rn Zaefferer	-336	1851
Karl Swedberg	0	0
Klaus Hartl	0	6
louisremi	220	158

表 5.18 jquery/jquery - 2

名前	テスト	本体
Michael Geary	0	2
Micha ナ・Go ナ F 74 biowski	218	78
Micha ナ・Go ナ て 冀 iowski	587	1899
Mike Alsup	0	27
Mike Sherov	-2	8
Oleg	9655	18
Oleg Gaidarenko	1929	687
Paul Bakaus	0	321
Paul Irish	10	1
Paul Mclanahan	0	0
Richard Gibson	1648	1129
Richard Worth	0	0
Rick Waldron	773	-7998
Rick Waldron waldron.rick@gmail.com	34	171
rwldrn	0	3
Sam Bisbee	0	3
Scott Gonz 7。 lez	17	111
Sean Catchpole	0	3
Stefan Petre	0	22
Timmy Willison	1574	8693
timmywil	2730	845
Timo Tijhof	0	0
TJ VanToll	16	1
unknown	0	-3
wycats	65	73
Yehuda Katz	514	667

表 5.19 less/less.js

名前	テスト	本体
agatronic	21	96
Alexis Sellier	756	33850
Andrej	179	100
Bass Jobsen	2	7
cloudhead	1680	12627
Luke Page	8268	-16678
Matthew Dean	634	2372
Matthew Smith	49	11
Max Mikhailov	0	4
meri	0	0
Rok Garbas	0	0
seven-phases-max	82	14
Thomas Grainger	20	0

表 5.20 janl/mustache.js

名前	テスト	本体
Abel Martin	0	0
Alexander Lang	21	119
Chad Weider	0	-11
Chris Anderson	50	0
Chris Wanstrath	0	68
Chris Williams	0	40
Damien Mathieu	0	18
David da Silva	268	546
David da Silva Contin	61	73
David da Silva Contin	0	35
GitHub	0	3
Jan Lehnardt	163	1393
Michael Jackson	817	-104
Paul J. Davis	37	6
Phillip Johnsen	63	122
Sebastian Cohnen	0	5

表 5.21 mozilla/pdf.js

名前	テスト	本体
Andreas Gal	218	12044
Artur Adib	1445	13420
Brendan Dahl	3599	9524
Chris Jones	2037	3665
GitHub	618	1241
Jonas	0	16
Jonas Jenwald	1767	14993
Julian Viereck	154	7314
notmasteryet	27	1185
piotrex	41	167
sayrer	0	5
sbarman	0	21
Vivien Nicolas	11	249
vyv03354	273	707
Xavier fung	22	988
Yury Delendik	19182	39044

表 5.22 scottjehl/Respond

名前	テスト	本体
Harry Schmidt	0	27
Jeff Lembeck	0	18
Jeffrey Lembeck	6	716
Paul Irish	0	2
Scott Jehl	0	15
scottjehl	2188	657
Zach Leatherman	176	91

表 5.23 resque/resque

名前	テスト	本体
Chris Wanstrath	1419	4896
defunkt	0	5
Dylan Thacker-Smith	98	121
Gabriel Gilder	96	14
GitHub	173	315
John Barnette	0	7
Ryan Biesemeyer	70	-2739
Ryan Carver	527	161
Steve Klabnik	668	749
steveklabnik	-11	2772
Terence Lee	483	605
Tony Arcieri	10	29

表 5.24 hakimel/reveal.js

名前	テスト	本体
Andy Matthews	0	4
Ben Houston	0	3
Brad Gessler	0	10
Catalin Buzoiu	0	16
Charlie DeTar	0	59
Christian Fehmer	0	76
Damjan Georgievski	0	2
Eric J. Duran	0	4
Eric Weigl	0	2
Gary Murakami	0	4
Hakim El Hattab	4004	15809
hakimel	0	292
Jeremy Mikola	0	5
karimsa	0	20
Mahemoff	0	0
Martin Kurtsson	0	0
Michael Mahemoff	0	0
Raymond Camden	0	3
Razvan Caliman	0	5
Russell Beattie	0	149
VonC	175	54

表 5.25 LearnBoost/socket.io

名前	テスト	本体
Damien Arrachequesne	125	236
einaros	356	538
GitHub	24	126
Guillermo Rauch	175716	2343
Tj Holowaychuk	182	166

表 5.26 jashkenas/underscore

名前	テスト	本体
Adam Krebs	-7887	1017
AJ ONeal	0	0
Brad Dunbar	9957	26
Graeme Yeates	15	93
Jason Davies	5	4
Jed Schmidt	0	0
Jeremy Ashkenas	4266	8435
John Wright	13	5
John-David Dalton	858	119
Jordan Eldredge	155	149
Justin Ridgewell	32	39
Kirill Ishanov	16	16
Kit Cambridge	159	26
Kit Goncharov	80	27
lifesinger	7	3
liuyl	2	-2
matehat	0	1
Michael Ficarra	1147	98
Michael Wolf	0	10
Mike Frawley	0	-1
Nick Stenning	4	0
noah	15	22
Oleg Grenrus	6	-7
opensas	0	-1
Pier Paolo Ramon	9	8
Rick Fletcher	22	0
Ryan W Tenney	31	4
Samuel Clay	11	6
summatix	0	1
Tal Ater	2	0
Tim Caswell	0	5

メンバごとの行数を比較したところ、パレートの法則が成り立つ可能性があるため調査をした。

まとめた結果を次に記す。

1	A	B	C	D	E	F	G	H
	プロジェクト名	追加行数	8割の行数	参加人数	上位貢献者の人数	割合	パレート	
2	angular/angular.js	468634	374907	42	5	12%	○	
3	caolan/async	35951	28760	10	2	20%	○	
4	jashkenas/backbone	32360	25888	26	2	8%	○	
5	bower/bower	17622	14087.6	18	3	17%	○	
6	adobe/brackets	483895	387116	31	4	13%	○	
7	jashkenas/coffee-script	44796	35836.8	24	2	8%	○	
8	Shopify/dashing	19911	15928.8	8	1	13%	○	
9	plataformatec/devise	8971	7176.8	12	2	17%	○	
10	discourse/discourse	478071	382456.8	9	3	33%	×	
11	visionmedia/express	13259	10607.2	8	2	25%	×	
12	gruntjs/grunt	2751	2200.8	4	1	25%	×	
13	visionmedia/jade	22813	18250.4	4	2	50%	×	
14	ekyl/jekyll	31717	25373.6	26	2	8%	○	
15	jquery/jquery	44048	35238.4	41	6	15%	○	
16	less/less.js	49081	39264.8	9	2	22%	×	
17	janl/mustache.js	2428	1942.4	12	3	25%	×	
18	mozilla/pdf.js	104583	83666.4	16	5	31%	×	
19	scott Jehl/Respond	1526	1220.8	7	2	29%	×	
20	resque/resque	9674	7739.2	11	2	18%	○	
21	hakimel/reveal.js	16517	13213.6	18	1	6%	○	
22	LearnBoost/socket.io	3409	2727.2	5	2	40%	×	
23	jashkenas/underscore	10114	8091.2	22	1	5%	○	
24								
25								
26								
27								
28								

図 5.1 メインコードでパレートの法則が成り立つか

1	プロジェクト名	追加行数	8割の行数	参加人数	上位貢献者の人数	割合	パレート
2	angular/angular.js	90371	72296.8	32	10	31%	×
3	caolan/async	3258	2606.4	5	1	20%	○
4	jashkenas/backbone	48281	38624.8	21	2	10%	○
5	bower/bower	14281	11424.8	14	4	29%	×
6	adobe/brackets	201014	160811.2	25	4	16%	○
7	jashkenas/coffee-script	10967	8773.6	21	4	19%	○
8	Shopify/dashing	409	327.2	4	3	75%	×
9	plataformatec/devise	9395	7516	12	3	25%	×
10	discourse/discourse	18129	14503.2	8	2	25%	×
11	visionmedia/express	11401	9120.8	3	2	67%	×
12	gruntjs/grunt	2898	2318.4	4	1	25%	×
13	visionmedia/jade	4375	3500	5	2	40%	×
14	ekyl/jekyll	8635	6908	15	4	27%	×
15	jquery/jquery	50745	40596	30	7	23%	×
16	less/less.js	11691	9352.8	10	2	20%	○
17	janl/mustache.js	1480	1184	8	3	38%	×
18	mozilla/pdf.js	29394	23515.2	13	3	23%	×
19	scott Jehl/Respond	2370	1896	3	1	33%	×
20	resque/resque	3544	2835.2	9	4	44%	×
21	hakimel/reveal.js	4179	3343.2	2	1	50%	×
22	LearnBoost/socket.io	176403	141122.4	5	1	20%	○
23	jashkenas/underscore	16812	13449.6	22	2	9%	○
24							
25							
26							
27							
28							

図 5.2 テストコードでパレートの法則が成り立つか

左の列からプロジェクト名，追加行数，8割の行数，参加人数，上位の貢献者の人数，パレートが入力されている．追加行数は，メインコードもしくはテストコードにコミットで追加された，全メンバーの合計のコード行数である．8割の行数は，追加行数に0.8を掛けた数で上位の貢献者を探す際の基準にした数値である．参加人数は，メインコードもしくはテストコードで1行以上のコードを追加した人数である．上位の貢献者の人数は，8割の行数をより多く書いていたメンバーの人数である．割合は，上位の貢献者の人数÷参加人数をした数値でパレートの法則が成り立つか基準にした数値である．パレートは，パレートの法則が成り立つ場合に○，成り立たない場合に×を入力している．

パレートの法則が成り立つプロジェクトはメインコードでは22件中13件であり，テストコードでは22件中7件であった．

OSS開発でもパレートの法則が成り立つことが考えられるが，同じプロジェクトでもメインコードよりテストコードの方が成り立つ可能性が少ない．これは，メインコードもテストコードも貢献度が高いメンバーの人数はあまり変わらないのに対し，メインコードよりテストコードの方が，参加メンバーが少ないためだと考える．

また，参加人数が少ないプロジェクトより参加人数が多いプロジェクトの方が，パレートの法則が成り立つことが多いように見える．これは，参加人数が増えることで貢献度の高いメンバーと低いメンバーの差が大きく開いてくるからだと考える．少数でプロジェクトを進める方がメンバーの作業量のばらつきはないようだ．

メインコードとテストコードの両方で貢献度の高いメンバに違いがあるのか比べた。メインコードとテストコードの両方で貢献しているメンバがいた場合にその人数を数えた。まとめた結果を次に記す。

	A	B	C
1	プロジェクト名	人数	
2	angular/angular.js	3	
3	caolan/async	1	
4	jashkenas/backbone	2	
5	bower/bower	2	
6	adobe/brackets	3	
7	jashkenas/coffee-script	2	
8	Shopify/dashing	1	
9	plataformatec/devise	2	
10	discourse/discourse	1	
11	visionmedia/express	1	
12	gruntjs/grunt	1	
13	visionmedia/jade	1	
14	jekyll/jekyll	2	
15	jquery/jquery	1	
16	less/less.js	1	
17	janl/mustache.js	2	
18	mozilla/pdf.js	2	
19	scott Jehl/Respond	1	
20	resque/resque	2	
21	hakimel/reveal.js	1	
22	LearnBoost/socket.io	1	
23	jashkenas/underscore	1	
24			
25			
26			

図 5.3 貢献度の高いメンバの人数

メインコードとテストコードの両方で貢献しているメンバは 1 から 3 人ほどであり，その他はどちらかで貢献していることが多かった．少数のメンバがメインコードとテストコードの両方で貢献度が高く，そのメンバが中心的に活動していると考ええる．メインコードとテストコードを書いているメンバは同一人物で違いはあまりないと考えていたが，メインコードとテストコードのどちらかだけで貢献するなど，メインコードとテストコードで違いが出た．

このように，メインコードとテストコードごとでメンバによるプロジェクトへの貢献度の違いを見ることができた．

第 6 章

結論

本研究では，GitHub 上のプロジェクトを調査することにより，テスト駆動開発において，個人の貢献度がメインコードとテストコードで違いがあるのかを調査した．

主にパレートの法則をもとにし，それぞれのプロジェクトを比較した．OSS 開発の現場でもパレートの法則は成り立つ場合がある．メンバによる貢献度の違いが明らかになったため，貢献度の高いメンバの開発手法を参考にすることで，より効率的な開発が可能になると考える．

メインコードとテストコードを書いているメンバは同一人物でない場合が多かった．全体的にテストコードを書くメンバが少ないため，テストコードを書くことでプロジェクトへの貢献度は高まると考える．

参考文献

- [1] 清水竜吾. テストを基準にしたソフトウェア開発プロセスの調査. 卒業論文, 千葉工業大学, 2012.
- [2] オープンソースソフトウェア - wikipedia. <https://ja.wikipedia.org/wiki/%E3%82%AA%E3%83%BC%E3%83%97%E3%83%B3%E3%82%BD%E3%83%BC%E3%82%B9%E3%82%BD%E3%83%95%E3%83%88%E3%82%A6%E3%82%A7%E3%82%A2> (2017.1.10 閲覧).
- [3] バージョン管理システム - wikipedia. <https://ja.wikipedia.org/wiki/%E3%83%90%E3%83%BC%E3%82%B8%E3%83%A7%E3%83%B3%E7%AE%A1%E7%90%86%E3%82%B7%E3%82%B9%E3%83%86%E3%83%A0> (2017.1.10 閲覧).
- [4] ギットハブ・ジャパン合同会社. GitHub. <https://github.co.jp/about/> (2017.1.10 閲覧).
- [5] パレートの法則 - wikipedia. <https://ja.wikipedia.org/wiki/%E3%83%91%E3%83%AC%E3%83%BC%E3%83%88%E3%81%AE%E6%B3%95%E5%89%87> (2017.1.10 閲覧).
- [6] Linux - wikipedia. <https://ja.wikipedia.org/wiki/Linux> (2017.1.10 閲覧).
- [7] CentOS - Wikipedia. <https://ja.wikipedia.org/wiki/CentOS> (2017.1.10 閲覧).
- [8] VirtualBox - Wikipedia. <https://ja.wikipedia.org/wiki/VirtualBox> (2017.1.10 閲覧).
- [9] Vagrant(ソフトウェア) - wikipedia. [https://ja.wikipedia.org/wiki/Vagrant_\(%E3%82%BD%E3%83%95%E3%83%88%E3%82%A6%E3%82%A7%E3%82%A2\)](https://ja.wikipedia.org/wiki/Vagrant_(%E3%82%BD%E3%83%95%E3%83%88%E3%82%A6%E3%82%A7%E3%82%A2)) (2017.1.10 閲覧).

第 7 章

謝辞

本研究を進めるにあたり，矢吹研究室矢吹太郎准教授には，多くの時間をご指導にさいて頂きました．また，矢吹研究室の皆様には，多くの知識や示唆を頂きました．協力していただいた皆様に感謝の気持ちと御礼を申し上げます．