千葉工業大学　社会システム科学部

プロジェクトマネジメント学科

平成25年度　卒業論文

テストを基準にした

ソフトウェア開発プロセスの調査

|  |  |
| --- | --- |
| 指導教員印 | 学科受付印 |
|  |  |

ソフトウェア開発管理コース

矢吹研究室

1042060　清水 竜吾／Ryugo SHIMIZU

**目次**

[**第1章** **序論** 1](#_Toc369194695)

[**1.1.** **研究背景** 1](#_Toc369194696)

[**1.2.** **研究目的** 2](#_Toc369194697)

[**1.3.** **プロジェクトマネジメントとの関連性** 2](#_Toc369194698)

[**第2章** **ソフトウェア開発について** 3](#_Toc369194699)

[**2.1.** **ソフトウェア開発とは** 3](#_Toc369194700)

[**2.2.** **ソフトウェア開発プロセスについて** 3](#_Toc369194701)

[**2.2.1.** **要求（要件）定義工程とは** 3](#_Toc369194702)

[**2.2.2.** **設計工程とは** 3](#_Toc369194703)

[**2.2.3.** **実装（コーティング）工程とは** 3](#_Toc369194704)

[**2.2.4.** **テスト工程とは** 4](#_Toc369194705)

[**2.2.4.1.** **主なテスト技法** 4](#_Toc369194706)

[**2.2.5.** **導入（運用）工程とは** 5](#_Toc369194707)

[**2.2.6.** **上流・下流工程とは** 5](#_Toc369194708)

[**2.3.** **開発プロセスの種類** 6](#_Toc369194709)

[**2.4.** **開発プロセスの比較** 8](#_Toc369194710)

[**第3章** **GitHubについて** 9](#_Toc369194711)

[**3.1.** **バージョン管理システム** 10](#_Toc369194712)

[**3.2.** **バージョン管理システム** 11](#_Toc369194713)

[**3.3.** **用語説明** 13](#_Toc369194714)

[**3.4.** **GitHub APIについて** 17](#_Toc369194715)

[**3.4.1.** **APIとは** 17](#_Toc369194716)

[**3.4.2.** **GitHub APIとは** 18](#_Toc369194717)

[**3.4.2.1.** **GitHub APIリスト** 19](#_Toc369194718)

[**第4章** **本論** 29](#_Toc369194719)

[**4.1.** **研究方法** 29](#_Toc369194720)

[**4.2.** **研究手順** 29](#_Toc369194721)

[**4.3.** **進捗状況** 30](#_Toc369194722)

[**参考文献** 31](#_Toc369194723)

**図目次**

[図 1‑1.GitHub登録者・リポジトリ数の推移 1](#_Toc369194724)

[図 2‑1.ウォーターフォール・モデル図 6](#_Toc369194725)

[図 2‑2.アジャイルソフトウェア開発図 7](#_Toc369194726)

[図 2‑3.開発プロセスの比較図 8](#_Toc369194727)

[図 3‑1.2012年1月1日時点での言語別コミット数 10](#_Toc369194728)

[図 3‑2.集中型バージョン管理イメージ図 11](#_Toc369194729)

[図 3‑3.分散型バージョン管理図イメージ図 12](#_Toc369194730)

[図 3‑4.APIイメージ図 17](#_Toc369194731)

[図 3‑5.GitHub API使用例 18](#_Toc369194732)

**表目次**

[表 3‑1. GitHubで開発が進められているソフトウェア 9](#_Toc369194232)

[表 3‑2. GitHub APIリスト 19](#_Toc369194233)

# **序論**

本章では，本研究の背景，目的，プロジェクトマネジメントとの関連性を記す．

* 1. **研究背景**

これまでの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手法が採用されていた．それに代わってアジャイル型開発手法が普及してきている．ウォーターフォール型開発手法は仕様を最初にすべて決めてしまうため，スケジュール管理は容易だが，開発途中の仕様変更は難しい．その一方で，アジャイル型開発手法は簡単なウォーターフォール型開発手法をサイクルして開発を行うため，開発途中の仕様変更が容易である．

アジャイル型開発手法では，テスト駆動開発手法（Test Driven Development）がよく採用される．これは，先にプログラムに必要な機能について初めにテストを考案し，そのテストが動作する必要最低限な実装を行い，コードを洗練させ，短い工程で計画・開発を繰り返し行う手法である．

実際の製品として販売・配布されている多くのソフトウェアはテストを含めて，開発プロセスに関する情報は一般には知ることができなかったが，近年では開発プロセスに関する情報を一般でも見ることができるオープンソースソフトウェア（OSS）の開発が盛んである．OSS開発には，OSSホスティングサービスを利用しての開発されることが多い．ホスティングサービスとはソースコードやドキュメントのバージョン管理するシステムやバグ・テスト情報を記録できるシステムなどの複合システムである．

最もよく利用されているホスティングサービスの一つがGitHubである．実際，GitHubの**図 1 1.GitHub登録者・リポジトリ数の推移**をみるとリポジトリ数は，サービスを開始した2008年には38,423件だったのに対して，2012年には4,614,306件と大幅に増加した．ユーザ数も同期間に41,157人から2,763,437人へと大幅に増加した．

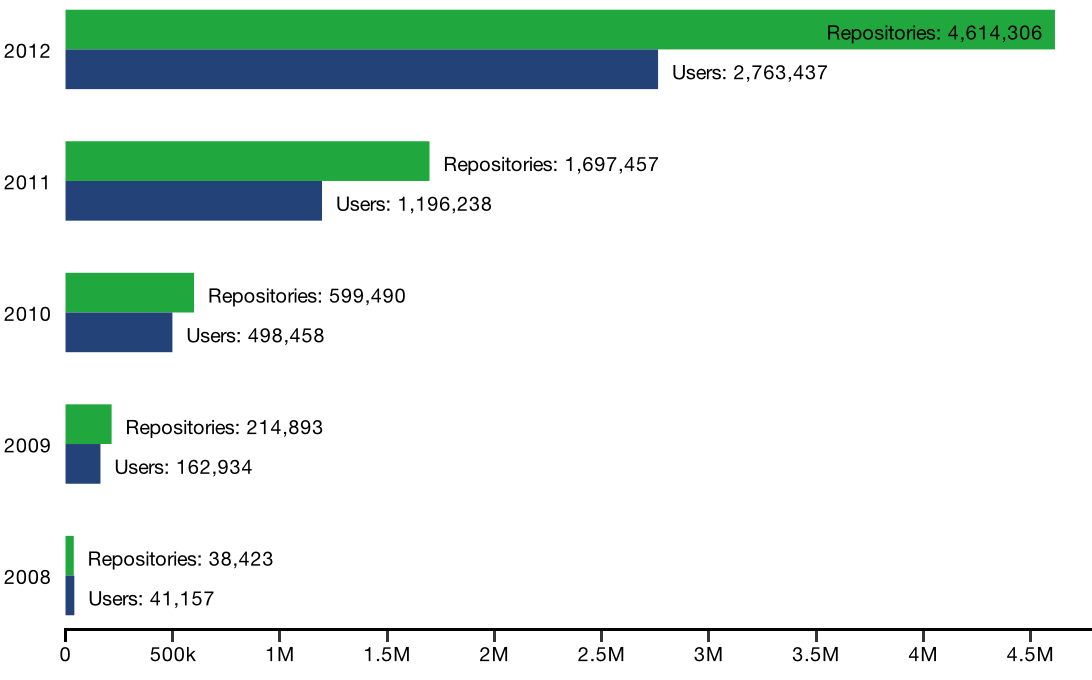


図 1‑1.GitHub登録者・リポジトリ数の推移

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・解析することによって，近年のソフトウェア開発傾向を調べることができると思われる．

* 1. **研究目的**

本研究では，現実のソフトウェア開発プロセスであるテスト工程に着目して調査し，ソフトウェア開発プロセスを明らかにすることを目的とする．

* 1. **プロジェクトマネジメントとの関連性**

本研究は，現実のソフトウェア開発がどのようなプロセスで行われているかを調べる研究である．ソフトウェア開発におけるプロジェクトマネジャーは，開発プロセスについて深く理解しておくことが望ましく，本研究の成果はそれに貢献することが期待される．

# **ソフトウェア開発について**

本章では，本研究の目的である『テスト工程』に着目する前にソフトウェア開発の基本知識や種類などについて記す．

* 1. **ソフトウェア開発とは**

ソフトウェア開発とは一般的にコンピュータ・ソフトウェアの設計や製造の一連の工程のことである．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（要件）定義工程・設計工程・実装（コーティング）工程・テスト工程・導入（運用）工程』である．

* 1. **ソフトウェア開発プロセスについて**

ソフトウェア開発プロセスの際に取られる各工程について記す．

* + 1. **要求（要件）定義工程とは**

要求定義工程では，利用者が業務を進める際に，そのシステムなどを使ってどのようなことがしたいのか，どのようなことができなければ困るのかといった内容を，開発者による聞き取りなどを通じて明らかにし，要求定義書として文書化する工程である．また，開発者は利用者がそのソフトウェアにてどのようなことを行いたいのかを元に，その行いたいことを実現するのに実装しなければならない機能や，満たさなければならない性能などを開発者が検討して明確にし，要件を定義する工程である．

* + 1. **設計工程とは**

設計工程には以下の二つの段階がある．以下はそれを説明する．

* **外部設計**

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ていく．主に操作画面やユーザ・インターフェースの構成や，書式や，データベースの構造などを決める段階である．

* **内部設計**

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し，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使用などを設計する．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段階である．

* + 1. **実装（コーティング）工程とは**

実装（コーティング）工程では，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．

* + 1. **テスト工程とは**

1. **モジュールテスト（ユニットテスト）**

テスト工程では，モジュールごとにテストを行い，バグ（論理上の誤り）を検出して修正を行う．

1. **ソフトウェア結合テスト**

モジュールごとにテストを行った後にモジュールを組み合わせてモジュール間のインターフェースに不具合がないかを検証する．また，検証するには様々な技法がある．それらの技法は後述の節に記す．

1. **システム結合テスト（システムテスト）**

モジュール間のインターフェースに不具合がなかったらシステム全体を稼働させて行うテストを行う．そのテストでは，必要な機能・性能が満たされているかを検証する．テスト内容としては，必要な機能が正常に動くか・要求された性能か・エラー処理が正常に動作するか・負荷をかけても動作するか・ユーザ・インターフェースが使いやすいかを検証するテストを行う．

* + - 1. **主なテスト技法**

主なテスト技法は以下のものである．

* **ボトムアップテスト**

複数のモジュールを結合させたサブシステムを，下位モジュールから順に結合しながら検証していくテスト方法．複数のモジュールを使用したプログラムを開発する場合は，バグの発生個所を特定しやすくするために，全てのモジュールを組み合わせて一気にテストするのではなく少しずつ組み合わせながら順次テストしていく方式が一般的である．

* **トップダウンテスト**

ボトムアップテストとは逆に，上位モジュールから順に結合しながらテストする方法．

* **ビッグバンテスト**

複数のモジュールを結合させたサブシステムを，全てのモジュールを組み合わせてから一気に動作検証するテスト方法．それぞれのモジュールは単体テストを完了している必要がある．大規模なシステムでビッグバンテストを行うと問題点の特定が難しいため，小規模なシステムや構造の単純なプログラムのテストに用いられる場合がほとんどである．

* **ホワイトボックス法とブラックボックス法**

チェックを行うために必要な値を含めたデータのことを，テストデータと言う．テストデータの作成技法として，プログラムを作成する開発者の視点で作るホワイトボックス法，ユーザ側の視点で作るブラックボックス法がある．

ホワイトボックス法では，プログラムの処理手順や処理内容などの内部構造を意識して，バグが発見できるようにテストデータを作り込む．プログラムの作成者であるエンジニアが，モジュールテスト用のテストデータを作成するときに用いる技法である．

それに対して，ブラックボックス法は，プログラムの内部構造を意識せず，「入力データAを処理すると，Bが結果の値として返ってこなければならない」という要件をクリアするかを検証するテストデータを作るときに用いる技法である．ソフトウェア結合テスト以降のテストで，要件定義書に定められた機能を満たすことを検証するためのテストデータ作成などに使われる技法である．

* + 1. **導入（運用）工程とは**

ソフトウェア開発はテスト環境にて開発し，実際の稼働環境へシステムを移す．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．

一度完成したシステムも，ハードウェアの故障への対応や，業務の変化に伴う修正などの保守が，随時行われる．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．

* + 1. **上流・下流工程とは**

ソフトウェア開発において行われる一連の工程は，『要求定義（要件定義）工程・設計工程』を上流工程といい，『実装（コーティング）工程・テスト工程・導入（運用）工程』を下流工程と言う．

上流工程とは，コンピュータ・ソフトウェアの設計や製造の初期工程のことである．上流工程にて，システム全体の規模や，開発にかかる期間やコストなども上流工程にて決定される．

下流工程とは，実装や導入など具体的なソフトウェアの構築・配備に関する工程のことである．

* 1. **開発プロセスの種類**
     1. **ウォーターフォール・モデル**

ウォーターフォール・モデルはソフトウェア開発において，古典的なプロセスモデルである．先進的な情報システム開発の世界や小規模開発においては必ずしも最適なモデルとは言えませんが，今も大きな規模のソフトウェア・システムを開発する場合の基本となるモデルです．

ウォーターフォール・モデルは，図 2-1のように『要件定義工程・設計工程・実装（コーティング）工程・テスト工程・導入（運用）工程』などの各段階に分けあたかも水が落ちるかのように工程を進め，前の工程へ戻らないような進め方をする．

要求（要件）

定義工程

設計工程

実装（コーティング）工程

テスト工程

導入（運用）工程

図 2‑1.ウォーターフォール・モデル図

ウォーターフォール・モデルでは，工程を明確に区別しているため，開発の進捗管理がしやすく多人数での共同作業なる大規模システムの開発においても，信頼性の高いシステムを構築できるというメリットがある．

一方では．様々な欠点がある．コーティングを始めるまでにすべての要求と設計を確定しなくてはならない．しかし，要求事項や設計の詳細をすべて明確にするのは簡単ではない．ビジネスにスピードを求められる時代ではシステムへの要求事項も時々刻々と変化しているのが普通である．現実の世界はウォーターフォール・モデルが求めるようには，厳密には工程後に区切られて動かないのである．また，ウォーターフォール・モデルでは大量のドキュメントの作成に必要とし，開発時間がかかるため，工程の流れが最後まで終了しない限りユーザはたとえ部分的にでもシステムを入手して使い始めることができない．

* + 1. **アジャイルソフトウェア開発**

アジャイルソフトウェア開発は，ソフトウェア開発において，ソフトウェア工学において迅速かつ適応的にソフトウェア開発を行う軽量な開発手法群の総称である．

近年，アジャイルソフトウェア開発手法が数多く考案されている．ソフトウェア開発で実際に採用される事例も少しずつではあるが増えつつある． アジャイルソフトウェア開発手法の例としては，エクストリーム・プログラミング (XP) などがある．

アジャイルソフトウェア開発は，図 2-1のように『要件定義工程・設計工程・実装（コーティング）工程・テスト工程・導入（運用）工程』などの各段階に分けあたかも水が落ちるかのように工程を進め，前の工程へ戻らないような進め方をする．

**1期間中に**

**複数回繰り返し**

レビュー

テスト工程

実装（コーティング）工程

設計工程

仕様変更

要求（要件）

定義工程

設計工程

実装（コーティング）工程

テスト工程

レビュー

図 2‑2.アジャイルソフトウェア開発図

アジャイルソフトウェア開発では，工程を明確に区別していなく，開発の進捗管理がしにくい．一方では，様々な利点がある．ウォーターフォール・モデルとは違い，すべての要求と設計をしなくても，ある程度の要求と設計を立ててしまえばプロセスを進めることができる．そのため，IT市場の進化への適応することができる．また，ウォーターフォール・モデルでは，ドキュメントを詳しく作らないことが多いかわりに，ユーザは部分的なシステムを使うことができる．

* + 1. **アジャイルソフトウェア開発**

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実装をとりあえず行った後，コードを洗練させるという短い工程を繰り返すスタイルである．また，テストを最初に書くことをテストファストという．

* 1. **開発プロセスの比較**

開発手法にはこれらのような開発プロセスがある．開発手法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，ソフトウェア開発規模によって適宜変更する必要がある．

一般的には以下の図のように，IT市場の進化などに対して適応的開発を行う必要がある場合は，アジャイルソフトウェア開発を行う．また，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モデルを採用する．

適応的開発　　　　　　　　　　　計画重視

アジャイルソフトウェア

開発

ウォーターフォール

モデル

反復型開発

図 2‑3.開発プロセスの比較図

# **GitHubについて**

GitHubとは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，Gitバージョン管理システムである．

GitHubはGistと呼ばれるpastebinスタイルのサイトも運営している．それらのコンテンツは具体的には個々のリポジトリのためのウィキおよびウェブページでありGitリポジトリを通して編集される．

尚現在では，GitHubを就職活動の選考書類として，GitHubアカウントを使用する会社も出てきている．

また，GitHubとは別に和製GitHubと言われる，株式会社ビズリーチが運営するIT・ウェブエンジニアの協業を支援するサイト「コードブレイク」を正式オープンし，プログラムのソースコードを共有できるリポジトリを提供するGitホスティングサービス「gitBREAK」（ギットブレイク）が開始されたが，本稿では近年のソフトウェア開発傾向を調べるために，世界中で使われ，ユーザ数が多いGitHubを用いることとする．

GitHubで開発が進められている有名なソフトウェアの一部は以下の表のとおりである．

表 3‑1. GitHubで開発が進められているソフトウェア

|  |  |
| --- | --- |
| Ruby on Rail | Rubyにて使われる代表的なWEBフレームワーク |
| node | JavaScriptにて人気のあるプラットフォーム |
| jQuery | JavaScriptライブラリ |
| Symfony2 | PHPにてつくられたフルスタックWEBフレームワーク |
| Bootstrap | ついったーのようなインターフェースをつくれるコンポーネント集 |

また，GitHubで公開されているプロジェクトには様々なプログラミング言語があり，また，背景で述べたように，GitHubは最もよく利用されているホスティングサービスの一つのため，GitHubで使われている言語は世界的に使われている言語ともとらえられるため，以下の図から，近頃はJavaScriptやJAVAが人気である．

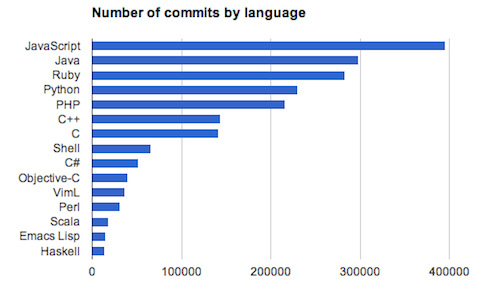


図 3‑1.2012年1月1日時点での言語別コミット数

* 1. **バージョン管理システム**

本章で出てくるGitとGitHubは別物である．

GitとはGitリポジトリというデータの貯蔵庫にソースコードなどを入れて利用する．このGitリポジトリを置く場所をインターネット上に提供しているのが　GitHubである．つまり，GitHubで公開されているソフトウェアのコードはすべてGitでされている．Gitについて理解しておくことはGitHubを使う上で重要である．

* 1. **バージョン管理システム**

バージョン管理システムとはソフトウェア開発ではソースコードを作成しながらソフトウェアを作り上げていきますが，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．そのようなソースコードの管理をするソフトウェアが「バージョン管理システム」である．ソフトウェア開発を複数人で開発を行う場合に必要不可欠なソフトウェアとなっている．

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（SVN（Apache Subversion））や，分散型バージョン管理システム（Git）などがある．

本稿ではGitバージョン管理システムを取り扱う．Gitバージョン管理システムは分散型バージョン管理システムに分類される．Gitバージョン管理システムはLinux開発のために2005年ごろにGitの原型となるプログラムが開発された．当初，この管理システムはLINUXのバージョン管理システムを置き換える目的で作成された．LINUXカーネルは高度な変更が加えられるソフトウェアのため，既存のバージョン管理システム（SVNなど）十分な機能や性能が確保できないため，新しく開発された．

* **集中型バージョン管理システム**

集中型バージョン管理システムは図3-1のように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がほとんど行えない．また，サーバが故障してデータが消えてしまった場合は，最新のソースコードをロストしてしまいます．

リポジトリ

**開発者Ａ**

**開発者Ｂ**

**Commit**

**Checkout**

**Commit**

**Checkout**

**図 3‑2.集中型バージョン管理イメージ図**

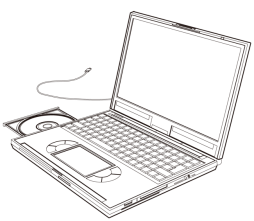
* **分散型バージョン管理システム**

分散型バージョン管理システムは，図3-2のようにサーバにある特定のリポジトリを自分のアカウント以下のリポジトリに複製し，複製したリポジトリは自分の好きなように編集でき，サーバにあるリポジトリとは異なるリポジトリとなる．

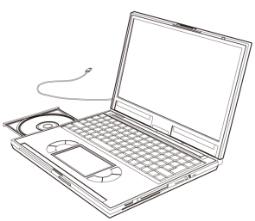
分散型は複雑であるが，図3-2のように複数リポジトリがあるのが特徴である．手元の開発環境にもリポジトリがあるため，リモートのリポジトリに接続できなくても開発することができる．

**開発者Ａ**

**開発者B**



Git



Git

**GitHub**

Git

Git

**Fork**

**Pull Request**

**Pull**

**Push**

**Pull**

**Push**

**ローカル環境**

**ローカル環境**

図 3‑3.分散型バージョン管理図イメージ図

* + 1. **バージョン管理システム比較**

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．

しかし，今日ではGitとGitHubの普及により，今後は分散型が圧倒的に普及するのは明らかである．ルールによっては，分散型バージョン管理システムで集中型バージョン管理システムのような管理もできる．そのため，今後，集中型バージョン管理システムを使うケースが少なくなると考えられている．同様に，今後集中型バージョン管理システムから分散型バージョン管理システムへ移行していくと思われる．

* 1. **用語説明**

以下に本章にて出てくる専門用語やGitHubの提供する機能の説明をする．

* + 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のである．

* + 1. **コミット**

ファイルやディレクトリの追加・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．

* + 1. **圧縮コミット**

複数のコミットを一つのコミットにまとめることである．

* + 1. **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．

* + 1. **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なるようにすることである．

* + 1. **ブランチ**

履歴の流れを分岐して記録していくためのものである．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のである．

* + 1. **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．

* + 1. **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．

* + 1. **Origin**

clone元のリモートリポジトリのことである．

* + 1. **Upstream**

当該リポジトリよりも上流に位置するリモートリポジトリのことである．

* + 1. **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．

* + 1. **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．

* + 1. **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．

* + 1. **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のことである．

* + 1. **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．

* + 1. **HEAD**

現在使用しているブランチの先頭を表す名前のことである．

* + 1. **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を指す．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を指すのである．

* + 1. **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・ディレクトリ郡のことである．

* + 1. **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．

* + 1. **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．

* + 1. **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．

* + 1. **競合**

マージ（リベースを含む）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．

* + 1. **フェッチ**

リモートリポジトリの最新を取り込むこと

* + 1. **二等分**

二分探索の要領で，バグが入り込んだコミットを突き止める作業

* + 1. **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．

* + 1. **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．

* + 1. **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．

* + 1. **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．

* + 1. **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ときに使われる．

* + 1. **GitHub API**

GitHubでは開発者向けにAPIを公開している．また，GitHub APIについては後述に記す．

* + 1. **アドオン**

ソフトウェアに追加される拡張機能のことである．

* + 1. **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ことである．

* + 1. **ディレクトリ**

　フォルダのことである．ファイルを分類・整理するための保管場所である．

* + 1. **フォロー**

手助けすることである．

* + 1. **フォロワー**

ユーザがフォローしている相手から視点のことである．

* + 1. **スター**

お気に入りで後で再確認など後で見るときなどに使う

* + 1. **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．

* + 1. **ウィキ（Wiki）**

　Webサーバ上のハイパーテキスト文書を書き換えるシステムの一種である．それらシステムに使われるソフトウェア自体や，システムを利用して作成された文書群全体を指してウィキと呼ぶこともある

* + 1. **Oragnization**

　Oragnizationとは普段，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のがOragnizationである．

* 1. **GitHub APIについて**

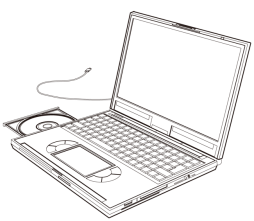
本節では本研究においてツールを作成する際に，GitHub APIを使用する．そのため本節ではAPI・GitHub APIについて調査する．

* + 1. **APIとは**

アプリケーションプログラミングインタフェースのことである．ソフトウェアコンポーネントが互いにやりとりするのに使用するインターフェースの仕様である．APIには，サブルーチン，データ構造，オブジェクトクラス，変数などの仕様が含まれる．APIには様々な形態があり，POSIXのような国際規格，マイクロソフトの Windows API のようなベンダーによる文書，プログラミング言語のライブラリがある．

APIは，アプリケーションから利用できる，オペレーティングシステムやプログラミング言語で用意されたライブラリなどの機能の入り口となるものである．主に，ファイル制御，ウインドウ制御，画像処理，文字制御などのための関数として提供されることが多い．

つまり，簡単にいえば，アプリケーションをプログラムするにあたって，プログラムの手間を省くため，もっと簡潔にプログラムできるように設定されたインターフェースの事である．



**API提供サーバ**

**API利用サーバ**

**一般ユーザ**

1. **WEBサイトにアクセス**

**②APIと通じてデータ取得**

**③XML等のデータを返送**

**④データを表示**

図 3‑4.APIイメージ図

* + 1. **GitHub APIとは**

GitHub APIとはGitHubにて使用するために用意されたAPIである。

GitHub APIを使用した例は図 3-5のようになる。



**拡大**

図 3‑5.GitHub API使用例

図3-5のように，GitHub内のデータをXML形式で呼び出すことができる．

このようにXML形式でデータを取得することで，プログラムに読み込んだりできる．また，API種類『DELETE・POST・PUT』などを使えば，GitHub外からのGitHubの操作することができる．

今回ソフトウェア開発プロジェクトの情報を読み込むときに使用する．

* + - 1. **GitHub APIリスト**

本項では，GitHubにあるAPIをリスト形式で説明をする．尚現在APIを調査中につき一部不完全である．

表 3‑2. GitHub APIリスト

|  |  |  |
| --- | --- | --- |
| **API種類** | **APIコード** | **意味** |
| **Events** | | |
| GET | /users/{userName}/events | ユーザが実行したイベントの一覧を取得する． |
| GET | /networks/{userName}/{repoName}/events | ネットワーク内でユーザが実行したリポジトリの共有イベントの一覧を取得する． |
| GET | /orgs/{org}/events | 組織の共有イベントの一覧を取得する． |
| GET | /repos/{userName}/{repoName}/events | ユーザが実行したリポジトリのイベントを取得する． |
| GET | /events | 共有イベント名を取得する． |
| GET | /users/{userName}/events/orgs/{org} | 組織でユーザが実行したイベントの一覧を取得する． |
| GET | /repos/{userName}/{repoName}/issues/events | リポジトリでユーザが実行したイベントの一覧を取得する． |
| GET | /users/{userName}/received\_events | ユーザが受信したイベントの一覧を取得する． |
| GET | /users/{userName}/events/public | ユーザの共有イベント名の一覧を取得する． |
| GET | /users/{userName}/received\_events/public | ユーザが受信した共有イベントの一覧を取得する． |

|  |  |  |
| --- | --- | --- |
| **Gists Comments** | | |
| DELETE | /gists/comments/{id} | ギストコメントを削除する． |
| GET | /gists/{gistId}/comments | ギストコメントの一覧を取得する． |
| GET | /gists/comments/{id} | ギストコメントの返信を取得する． |
| POST | /gists/comments/{id} | ギストコメントを編集する． |
| POST | /gists/{gistId}/comments | ギストコメントを作成する． |

|  |  |  |
| --- | --- | --- |
| **Gists** | | |
| DELETE | /gists/{id}/star | ギストのスターを解除する． |
| DELETE | /gists/{id} | ギストのスターを削除する． |
| GET | /gists | 認証されたユーザのギストを一覧取得するか，匿名で呼び出された場合はすべての共有ギストを取得します． |
| GET | /users/{userName}/gists | ユーザ一覧のギストを取得する． |
| GET | /gists/public | 共有ギストの一覧を取得する． |
| GET | /gists/{id}/star | 指定したギストにスターがついているかを取得します． |
| GET | /gists/starred | ユーザがスターをつけたギストの一覧を取得します． |
| GET | /gists/{id} | 指定したギストの情報を取得する． |
| POST | /users/{userName}/gists | 新しいギストを作成する． |
| POST | /gists/{id}/fork | ギストをフォークする |
| POST | /gists/{id} | ギストを編集する． |
| PUT | /gists/{id}/star | ギストのスターを更新する． |

|  |  |  |
| --- | --- | --- |
| **Issue Comments** | | |
| DELETE | **/repos/{userName}/{repoName}/issues/comments/{id}** | リポジトリのIssueのコメントを削除する． |
| GET | **/repos/{userName}/{repoName}/issues/comments/{id}** | リポジトリのIssueのコメントを取得する． |
| GET | **/repos/{userName}/{repoName}/issues/{id}/comments** | リポジトリのIssueの返信コメントを取得する． |
| POST | **/repos/{userName}/{repoName}/issues/{id}/comments** | リポジトリのIssueにコメントを作成する． |
| POST | **/repos/{userName}/{repoName}/issues/comments/{id}** | リポジトリのIssueにコメントを編集する． |

|  |  |  |
| --- | --- | --- |
| **Git Date** | | |
| GET | /repos/{userName}/{repoName}/git/commits/{sha} | 使用ユーザはGitリポジトリからコミットを取得する． |
| GET | /repos/{userName}/{repoName}/git/trees/{sha} | 使用ユーザはGitリポジトリからツリー形式に取得する． |
| GET | /repos/{userName}/{repoName}/git/tags/{sha} | 使用ユーザはGitリポジトリのタグを取得する． |
| GET | /repos/{userName}/{repoName}/git/refs/tags | 使用ユーザのGitリポジトリの全ての証明書タグを取得する． |
| GET | /repos/{userName}/{repoName}/git/refs | 使用ユーザのGitリポジトリのすべて証明書を取得する． |
| GET | **/repos/{userName}/{repoName}/git/refs/{ref}** | 使用ユーザのGitリポジトリの証明書を取得する． |
| GET | **/repos/{userName}/{repoName}/git/blobs/{sha}** | 使用ユーザのGitリポジトリのブロブを取得する． |
| GET | **/repos/{userName}/{repoName}/git/trees/{sha}?recursive=1** | 使用ユーザのGitリポジトリの１つ前のツリーを入手する． |
| POST | **/repos/{userName}/{repoName}/git/blobs** | 使用ユーザのGitリポジトリにブロブを作成する． |
| POST | **/repos/{userName}/{repoName}/git/trees** | 使用ユーザのGitリポジトリにツリーを作成する． |
| POST | **/repos/{userName}/{repoName}/git/refs/{ref}** | 使用ユーザのGitリポジトリの証明書を更新する． |
| POST | **/repos/{userName}/{repoName}/git/tags** | 使用ユーザのGitリポジトリ内にタグオブジェクトを作成する． |
| POST | **/repos/{userName}/{repoName}/git/commits** | 使用ユーザのGitリポジトリ内にコミットを作成する． |
| POST | **/repos/{userName}/{repoName}/git/refs** | 使用ユーザのGitリポジトリの証明書を作成する． |

|  |  |  |
| --- | --- | --- |
| **Issue Events** | | |
| GET | **/repos/{userName}/{repoName}/issues/{id}** | リポジトリの指定したIssueの変化を取得する． |
| GET | **/repos/{userName}/{repoName}/issues** | リポジトリのIssueの変化を取得する． |
| GET | **/repos/{userName}/{repoName}/issues/{issueId}/events** | リポジトリの指定したIssueの変化を取得する． |

|  |  |  |
| --- | --- | --- |
| **Issue** | | |
| GET | **/repos/{userName}/{repoName}/issues/{id}** | 選択されたIssueを取得する |
| GET | **/repos/{userName}/{repoName}/issues** | リポジトリのIssueの一覧を取得する． |
| GET | **/issues** | ユーザのIssueの一覧を取得する． |
| POST | **/repos/{userName}/{repoName}/issues/{id}** | 指定されたリポジトリのIssueを編集する． |
| POST | **/repos/{userName}/{repoName}/issues** | 指定されたリポジトリにIssueを作成する． |
| **Mile Stones** | | |
| DELETE | **/repos/{userName}/{repoName}/milestones/{id}** | マイルストーンを削除する． |
| GET | **/repos/{userName}/{repoName}/milestones** | マイルストーンの一覧を表示する． |
| GET | **/repos/{userName}/{repoName}/milestones/{id}** | 指定したマイルストーンの表示する． |
| POST | **/repos/{userName}/{repoName}/milestones** | マイルストーンを作成する． |
| POST | **/repos/{userName}/{repoName}/milestones/{id}** | マイルストーンを更新する． |

|  |  |  |
| --- | --- | --- |
| **Labels** | | |
| DELETE | **/repos/{userName}/{repoName}/issues/{id}/labels** | 指定したリポジトリのIssueから全てのラベルを削除する． |
| DELETE | **/repos/{userName}/{repoName}/labels/{id}** | 指定したリポジトリから指定ラベルを削除する． |
| DELETE | **/repos/{userName}/{repoName}/issues/{id}/labels/{id}** | 指定したリポジトリのIssueから指定したラベルを削除する． |
| GET | **/repos/{userName}/{repoName}/milestones/{id}/labels** | すべてのマイルストーンを取得する． |
| GET | **/repos/{userName}/{repoName}/labels** | リポジトリ内のラベルがついたもの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/labels/{id}** | リポジトリ内の選択されたラベルを取得する． |
| GET | **/repos/{userName}/{repoName}/issues/{id}/labels** | リポジトリ内のIssueにつけられラベルを取得する． |
| POST | **/repos/{userName}/{repoName}/issues/{id}/labels** | リポジトリ内のIssueにラベルを追加する． |
| POST | **/repos/{userName}/{repoName}/labels/{id}** | リポジトリ内のラベルを更新する． |
| POST | **/repos/{userName}/{repoName}/labels** | リポジトリ内にラベルを作成する． |
| PUT | **/repos/{userName}/{repoName}/issues/{id}/labels** | Issueのすべてのラベルを交換する． |

|  |  |  |
| --- | --- | --- |
| **Organization Members** | | |
| DELETE | **/orgs/{org}/public\_members/{userName}** | ユーザのメンバシップを非公開にする． |
| DELETE | **/orgs/{org}/members/{userName}** | メンバを組織から削除する． |
| GET | **/orgs/{org}/members/{userName}** | メンバを取得する． |
| GET | **/orgs/{org}/members** | 組織のメンバのユーザ一覧を取得する． |
| GET | **/orgs/{org}/public\_members/{userName}** | パブリックメンバを取得する． |
| GET | **/orgs/{org}/public\_members** | パブリックメンバの一覧を取得する． |
| PUT | **/teams/{id}/members/{user}** | メンバを追加する． |
| PUT | **/orgs/{org}/public\_members/{userName}** | ユーザの役割を変更する． |

|  |  |  |
| --- | --- | --- |
| **Organization** | | |
| GET | **/user/orgs** | 組織に認証されたユーザの一覧を取得する． |
| GET | **/orgs/{org}** | 組織を取得する． |
| GET | **/users/{userName}/orgs** | 組織のユーザの一覧を取得する． |
| POST | **/orgs/{org}** | 組織を編集する． |

|  |  |  |
| --- | --- | --- |
| **Organization Teams** | | |
| DELETE | **/teams/{id}/members/{userName}** | チームメンバを削除する． |
| DELETE | **/teams/{id}/repos/{userName}/{repoName}** | チームからリポジトリを削除する． |
| DELETE | **/teams/{id}** | チームを削除する． |
| GET | **/teams/{id}/members/{userName}** | チームメンバを取得する． |
| GET | **/teams/{id}/members** | チームのメンバの一覧を取得する． |
| GET | **/teams/{id}** | チーム詳細を取得する． |
| GET | **/orgs/{org}/teams** | 組織内のチームの一覧を取得する． |
| GET | **/teams/{id}/repos/{userName}/{repoName}** | チーム内のリポジトリを取得する． |
| GET | **/teams/{id}/repos** | チームのリポジトリの一覧を取得する． |
| POST | **/orgs/{org}/teams** | チームを作成する． |
| POST | **/teams/{id}** | チームを編集する． |
| PUT | **/teams/{id}/members/{userName}** | チームメンバを追加する． |
| PUT | **/teams/{id}/repos/{userName}/{repoName}** | チームリポジトリを追加する． |

|  |  |  |
| --- | --- | --- |
| **Pull Requests** | | |
| GET | **/repos/{userName}/{repoName}/pulls** | リポジトリ内のプルリクエストの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/pulls/{id}/commits** | リポジトリ内のコミットへのプルリクエストの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/pulls/{id}/files** | リポジトリからファイルへのプルリクエストの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/pulls/{id}** | リポジトリから指定したプルリクエストを表示する． |
| GET | **/repos/{userName}/{repoName}/pulls/{id}/merge** | リポジトリからマージへのプルリクエストを取得する． |
| POST | **/repos/{userName}/{repoName}/pulls/{id}** | プルリクエストを更新する． |
| POST | **/repos/{userName}/{repoName}/pulls** | プルリクエストを作成する． |
| PUT | **/repos/{userName}/{repoName}/pulls/{id}/merge** | マージをプルリクエストする． |

|  |  |  |
| --- | --- | --- |
| **Pull Requests Review Comments** | | |
| DELETE | **/repos/{userName}/{repoName}/pulls/comments/{id}** | プルリクエストのコメントを削除する． |
| GET | **/repos/{userName}/{repoName}/pulls/{id}/comments** | プルリクエストへのコメントの一覧をする． |
| GET | **/repos/{userName}/{repoName}/pulls/{id}** | プルリクエストへの指定されたコメントを取得する． |
| POST | **/repos/{userName}/{repoName}/pulls/comments/{id}** | コメントを編集する． |
| POST | **/repos/{userName}/{repoName}/pulls/{id}/comments** | コメントを作成する． |

|  |  |  |
| --- | --- | --- |
| **Repo Collaborators** | | |
| DELETE | **/repos/{userName}/{repoName}/collaborators/{user}** | 共同リポジトリを削除する． |
| GET | **/repos/{userName}/{repoName}/collaborators** | 共同リポジトリの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/collaborators/{user}** | 共同リポジトリの協力者を取得する． |
| PUT | **/repos/{userName}/{repoName}/collaborators/{user}** | 共同リポジトリのユーザを追加する． |
| **Repo Commits** | | |
| DELETE | **/repos/{userName}/{repoName}/comments/{id}** | コミットコメントを削除する． |
| GET | **/repos/{userName}/{repoName}/commits** | コミットの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/commits/{sha}** | 指定したコミットを取得する． |
| GET | **/repos/{userName}/{repoName}/comments** | コミットへのコメントの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/commits/{sha}/comments** | 指定したコミットへのコメントを取得する． |
| GET | **/repos/{userName}/{repoName}/comments/{id}** | 指定したコメントを取得する |
| GET | **/repos/{userName}/{repoName}/compare/{base}...{head}** | 2つのコミットを比較する． |
| POST | **/repos/{userName}/{repoName}/comments/{id}** | コミットのコメントを更新する． |
| POST | **/repos/{userName}/{repoName}/commits/{sha}/comments** | コミットのコメントを作成する． |

|  |  |  |
| --- | --- | --- |
| **Repo Deploy Keys** | | |
| DELETE | **/repos/{userName}/{repoName}/keys/{id}** | リポジトリの暗号鍵を削除する． |
| GET | **/repos/{userName}/{repoName}/keys/{id}** | リポジトリの暗号鍵を取得する． |
| GET | **/repos/{userName}/{repoName}/keys** | リポジトリの暗号鍵の一覧を取得する． |
| POST | **/repos/{userName}/{repoName}/keys** | リポジトリの暗号鍵を作成する． |
| POST | **/repos/{userName}/{repoName}/keys/{id}** | リポジトリの暗号鍵を編集する． |

|  |  |  |
| --- | --- | --- |
| **Repo Downloads** | | |
| DELETE | **/repos/{userName}/{repoName}/downloads/{id}** | リポジトリのダウンロードデータを削除する． |
| GET | **/repos/{userName}/{repoName}/downloads/{id}** | リポジトリのダウンロードデータを取得する． |
| GET | **/repos/{userName}/{repoName}/downloads** | リポジトリのダウンロードデータの一覧を取得する． |
| POST | **/repos/{userName}/{repoName}/downloads** | リポジトリから新しくダウンロードする． |

|  |  |  |
| --- | --- | --- |
| **Repo Forks** | | |
| GET | **/repos/{userName}/{repoName}/forks** | フォークデータの一覧を取得する． |
| POST | **/repos/{userName}/{repoName}/forks** | フォークを作成する． |

|  |  |  |
| --- | --- | --- |
| **Repo Hooks** | | |
| DELETE | **/repos/{userName}/{repoName}/hooks/{id}** | フックを削除する． |
| GET | **/repos/{userName}/{repoName}/hooks** | フックの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/hooks/{id}** | 指定したフックを取得する． |
| POST | **/repos/{userName}/{repoName}/hooks** | フックを作成する． |
| POST | **/repos/{userName}/{repoName}/hooks/{id}** | フックを編集する． |
| POST | **/repos/{userName}/{repoName}/hooks/{id}/test** | フックのテスト（検証）を行う． |

|  |  |  |
| --- | --- | --- |
| **Repos** | | |
| GET | **/repos/{userName}/{repoName}/branches** | ブランチの一覧を取得する． |
| GET | **/user/repos** | 認証済みユーザのリポジトリの一覧を取得する． |
| GET | **/users/{userName}/repos** | 指定したユーザの共有リポジトリの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}** | ユーザはリポジトリを取得する． |
| GET | **/orgs/{org}/repos** | 指定した組織の共有リポジトリの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/contributors** | 貢献者の一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/languages** | 言語の一覧を取得する． |

|  |  |  |
| --- | --- | --- |
| GET | **/repos/{userName}/{repoName}/teams** | チームの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/tags** | タグの一覧を取得する． |
| POST | **/user/repos** | 認証されたユーザのためにリポジトリを作成する． |
| POST | **/repos/{userName}/{repoName}** | ユーザのリポジトリを編集する． |
| POST | **/orgs/{org}/repos** | 組織のリポジトリを作成する． |

|  |  |  |
| --- | --- | --- |
| **Repo Watching** | | |
| DELETE | **/user/watched/{userName}/{repoName}** | リポジトリのウォッチをやめる． |
| GET | **/user/watched** | ユーザによってウォッチされているリポジトリの一覧を取得する． |
| GET | **/repos/{userName}/{repoName}/watchers** | ユーザがウォッチしているリポジトリの一覧を取得する． |
| GET | **/users/{userName}/watched** | ユーザがウォッチしたリポジトリの一覧を取得する． |
| GET | **/user/watched/{userName}/{repoName}** | 自分のリポジトリウォッチしているユーザの一覧を取得する． |
| PUT | **/user/watched/{userName}/{repoName}** | リポジトリをウォッチする |

|  |  |  |
| --- | --- | --- |
| **User** | | |
| GET | **/user** | 認証されたユーザを取得する． |
| GET | **/users/{userName}** | 指定したユーザを取得する． |
| POST | **/user** | 認証されたユーザを更新する． |

|  |  |  |
| --- | --- | --- |
| **User Emails** | | |
| DELETE | **/user/emails** | 電子メールアドレスを削除する． |
| GET | **/user/emails** | ユーザの電子メールアドレスの一覧を取得する． |
| POST | **/user/emails** | 電子メールアドレスを追加する． |

|  |  |  |
| --- | --- | --- |
| **User Followers** | | |
| DELETE | **/user/following/{userName}** | フォローを削除する． |
| GET | **/user/following** | 認証しているユーザのフォローしている一覧を取得する． |
| GET | **/users/{userName}/followers** | フォロワーの一覧を取得する． |
| GET | **/user/followers** | 認証されたユーザのフォロワーの一覧を取得する． |
| GET | **/users/{userName}/following** | フォローしているユーザの一覧を取得する． |
| GET | **/user/following/{userName}** | フォローしているかどうかを取得する． |
| PUT | **/user/following/{userName}** | ユーザをフォローする． |
| **API種類** | **APIコード** | **意味** |

|  |  |  |
| --- | --- | --- |
| **User Public Keys** | | |
| DELETE | **/user/keys/{id}** | 公開鍵を削除する． |
| GET | **/user/keys** | 公開鍵の一覧を取得する． |
| GET | **/user/keys/{id}** | 指定した公開鍵を取得する． |
| POST | **/user/keys/{id}** | 公開鍵を更新する． |
| POST | **/user/keys** | 公開鍵を作成する． |

# **本論**

本章では，これまでの調査を含めた研究方法と，現状の進捗における研究状況を報告する．

* 1. **研究方法**

本研究はGitHubにホスティングされているソフトウェアのテストコードの構成を調べ，それを解析するためのツールを作成する．そのツールを用いて，GitHubでホスティングされているプロジェクトを網羅的に調査し，現在のソフトウェア開発の実情を明らかにする．現在のソフトウェア開発の実情を調べる方法の例として，ソースコードを製品とテストに分け，それぞれの成長の様子を調べるということが考えられる．

* 1. **研究手順**

本研究の流れを以下に記す．

ツールを作成するための調査

解析するためのツールを作成

作成したツールをGitHubでホスティングされているオープンソースソフトウェアに使用

結果の考察

1. **ツールを作成するための調査**

ツールを作成するには様々な調査が必要である．GitHubでホスティングされているソフトウェアのソースコードを製品とテストに分けそれぞれの成長の様子を調べる方法の仮説を立て検証する．一方で成長の様子を取得するためのGitHub APIを使用するためにGitHub APIを調査する．

1. **解析するためのツールを作成**

本研究では解析するためのツールを作る際にGitHubのデータを用いるためにGitHub APIを採用する．GitHub APIにて必要なデータをGitHubでホスティングされているソフトウェアから取得させるプログラムを作成する．

1. **作成したツールを使用**

GitHubでホスティングされているソフトウェアを網羅的に無作為抽出（ランダムサンプリング）を行い，標本を抜く．無作為抽出で取得した標本に対して②で作成したツールを使用し解析する．

1. **結果の考察**

上記の手順で研究を行い，調査内容・解析が齎すデータの考察を行うことによって，近年のソフトウェア開発プロセスを明らかにし，テスト駆動開発の実情が明らかになることが期待される．

* 1. **進捗状況**

現在の進捗状況は以下の段階まで実験・調査中である．

1. ツールを作成するための調査
2. 解析するためのツールを作成

以降に各段階についての詳細を記す．

1. **ツールを作成するための調査**

現段階での成長の様子を調べる方法の仮説として，製品コードやテストコードの行数，または製品コードやテストコードの関数の数の計測にて製品とテストのそれぞれの成長の様子を図れるのではないかと仮説を立てた．なお，現在実際に調査して検証中．

1. **解析するためのツールを作成**

解析するためのツールを作成に使うAPIを調査終了．次は実際に使うGitHub API について調査中．①仮説を検証できたら実際にGitHub APIを使ってツールの作成を行う．

# **参考文献**

[1] Mint（経営情報研究会). 図解でわかる ソフトウェア開発のすべて. 日本実業出版社, 2000, 327p.

[2] 佐藤聖規, 和田貴久, 河村雅人, 米沢弘樹, 山岸啓, 川口耕介. Jenkins実践入門. 技術評論社, 2011, 336p.

[3] The GitHub Blog. The Octoverse in 2012. 2012-12-19. https://github.com/blog/1359-the-octoverse-in-2012

[4] パスサポ | めざせ！ 情報処理技術者試験 パスサポ … 技術評論社. グルーピングで楽に覚える情報処理試験用語. 2009-10-08. http://gihyo.jp/lifestyle/serial/01/ipa-terminology/

[5] Internet.Watch. 和製GitHub「ギットブレイク」公開、プライベートリポジトリが無料・無制限. 2013-05-29. http://internet.watch.impress.co.jp/docs/news/20130529\_601351.html

[6] apis.io. APIS.io :: The open-source API directory. 2013-05-29. http://apis.io/github

[7] ITエキスパートのための問題解決メディア - ＠IT. ガチで5分で分かる分散型バージョン管理システムGit. 2013-07-05. http://www.atmarkit.co.jp/ait/articles/1307/05/news028.html

[8] 飯山教史, 町田欣史, 高橋和也, 小堀一雄. 現場で使えるソフトウェアテスト. 翔泳社, 2008, 344p.

[9] ジェームズ・ウィテカー, ジェーソン・アーボン, ジェフ・キャローロ, 長尾高弘. テストから見えてくるグーグルのソフトウェア開発. 日経BP社, 2013, 432p.

|  |
| --- |
| git clone https://github.com/$1/$2  cd $2  git log --pretty=format:"%H,%cd" --date=iso --first-parent --no-merges > $2-commits.csv  cd ..  cat $2-commits.csv | python **lineCountScriptCreator.py** $2 > $2-count.sh  bash $2-count.sh > $2-count-result.csv 2> $2-error.log  rm -f count-result.csv  cp $2-count-result.csv count-result\_tmp.csv  awk -F ',' 'BEGIN{OFS=","} {print $0,$3-$4;}' count-result\_tmp.csv > count-result.csv  rm -f count-result\_tmp.csv  gnuplot <<EOF  set datafile separator ","  set xdata time  set timefmt "%Y-%m-%d %H:%M:%S"  set format x "%y-%m"  set xl "Date Time" font 'Times New Roman,25'  set yl "Lines" font 'Times New Roman,25'  set terminal png size 1680,1050  set lmargin 14  set rmargin 15  set tmargin 5  set border 15 lw 2  set tics font "Times New Roman,20"  set title "$2"  set title font 'Times New Roman,50'  set key outside font 'Times New Roman,20' spacing 1.5  set out "lines.png"  plot 'count-result.csv' using 2:3 with lines lw 2 title 'Total', 'count-result.csv' using 2:5 with lines lw 2 title 'Code','count-result.csv' using 2:4 with lines lw 2 title 'Test'  EOF  rm -f $2-lines.png  mv lines.png $2-lines.png |

|  |
| --- |
| git clone https://github.com/”ユーザー名”/”リポジトリ名” |

|  |
| --- |
| git log --pretty=format:"%H,%cd" --date=iso --first-parent --no-merges > “リポジトリ名“-commits.csv |

|  |
| --- |
| cat “リポジトリ名“-commits.csv | python lineCountScriptCreator.py “リポジトリ名“ > “リポジトリ名”-count.sh |

|  |
| --- |
| bash “リポジトリ名”-count.sh > “リポジトリ名”-count-result.csv 2 > “リポジトリ名”-error.log |

|  |
| --- |
| cp “リポジトリ名”-count-result.csv count-result\_tmp.csv |

|  |
| --- |
| awk -F ',' 'BEGIN{OFS=","} {print $0,$3-$4;}' count-result\_tmp.csv > count-result.csv |

|  |
| --- |
| rm -f count-result\_tmp.csv |

|  |
| --- |
| gnuplot <<EOF  set datafile separator ","  set xdata time  set timefmt "%Y-%m-%d %H:%M:%S"  set format x "%y-%m"  set xl "Date Time" font 'Times New Roman,25'  set yl "Lines" font 'Times New Roman,25'  set terminal png size 1680,1050  set lmargin 14  set rmargin 15  set tmargin 5  set border 15 lw 2  set tics font "Times New Roman,20"  set title "$2"  set title font 'Times New Roman,50'  set key outside font 'Times New Roman,20' spacing 1.5  set out "lines.png"  plot 'count-result.csv' using 2:3 with lines lw 2 title 'Total', 'count-result.csv' using 2:5 with lines lw 2 title 'Code','count-result.csv' using 2:4 with lines lw 2 title 'Test'  EOF |

|  |
| --- |
| # -\*- coding: utf-8 -\*-  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]  #git logの結果はタイムゾーンがばらばら。すべてUTCに統一する。  myDate = datetime.strftime(parse(x[1]).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,$(grep -rI '' '%s' | grep -v '^%s/\.git/' | wc -l),$(grep -rI '' '%s/test' | wc -l)") % (myHash, myDate, myProject, myProject, myProject)  print("else")  print(" echo %s,%s,$(grep -rI '' '%s' | grep -v '^%s/\.git/' | wc -l),0") % (myHash, myDate, myProject, myProject)  print("fi") |