

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

プロジェクトマネジメントをサポートする

WEBアプリケーションに関する研究

Research on WEB applications that support

Project management

プロジェクトマネジメントコース

矢吹研究室

0842062　工藤亮/Akira KUDO

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

内容

[1 序論 2](#_Toc369194572)

[1.1 本章の構成 2](#_Toc369194573)

[1.2 研究の背景 2](#_Toc369194574)

[1.3 研究の目的 3](#_Toc369194575)

[1.4 研究の方法 3](#_Toc369194576)

[1.5 プロジェクトマネジメントとの関連 3](#_Toc369194577)

[1.6 本論文の構成 3](#_Toc369194578)

[1.7 参考文献 4](#_Toc369194579)

[2.ソフトウェア開発プロジェクトについて 6](#_Toc369194580)

[2.1　本章の構成 6](#_Toc369194581)

[2.2　ソフトウェア開発プロジェクトについて 6](#_Toc369194582)

[2.3ソフトウェア開発の歴史 7](#_Toc369194583)

[2.4　ソフトウェア開発プロジェクトの成功事例 7](#_Toc369194584)

[2.4.1　成功事例 7](#_Toc369194585)

[2.4.2　成功事例の考察 7](#_Toc369194586)

[2.5　ソフトウェア開発プロジェクトの失敗事例 8](#_Toc369194587)

[2.5.1　失敗事例 8](#_Toc369194588)

[2.5.2　失敗原因 8](#_Toc369194589)

[2.6　事例から得られたこと 9](#_Toc369194590)

[2.7　参考文献 10](#_Toc369194591)

[3.　バージョン管理システム 12](#_Toc369194592)

[3.1　本章の構成 12](#_Toc369194593)

[3.2　バージョン管理システムについて 12](#_Toc369194594)

[3.3　バージョン管理システムの種類 12](#_Toc369194595)

[3.3.1　個別バージョン管理システム 13](#_Toc369194596)

[3.3.2　集中型バージョン管理システム 14](#_Toc369194597)

[3.3.3　分散型バージョン管理システム 14](#_Toc369194598)

[3.4　Gitについて 15](#_Toc369194599)

[3.4.1　Gitの歴史 16](#_Toc369194600)

[3.4.2　Gitの特徴 16](#_Toc369194601)

[3.4.3　Gitの利用状況 16](#_Toc369194602)

[3.4.4　GitとGitHubの違い 17](#_Toc369194603)

[3.5　GitHubについて 17](#_Toc369194604)

[3.6　GitHubの利用状況 17](#_Toc369194605)

[3.7　GitHubの基本用語 18](#_Toc369194606)

[3.8　GitHubの機能 19](#_Toc369194607)

[3.9　参考文献 21](#_Toc369194608)

[4.　システム構築について 23](#_Toc369194609)

[4.1　本章の構成 23](#_Toc369194610)

[4.2　APIについて 23](#_Toc369194611)

[4.2.1　APIを公開する・公開されるメリット 23](#_Toc369194612)

[4.3　GitHubのAPIの種類 23](#_Toc369194613)

[4.4　EVMの概要 34](#_Toc369194614)

[4.5　EVMの関連するプロジェクトマネジメント項目について 35](#_Toc369194615)

[4.6　参考文献 36](#_Toc369194616)

[5.プロジェクトマネジメントについて 38](#_Toc369194617)

[5.1　本章の構成 38](#_Toc369194618)

[5.2　プロジェクト作業の監視・コントロール・プロセス群について 38](#_Toc369194619)

[5.2.1　インプット 39](#_Toc369194620)

[5.2.2　ツールと技法 41](#_Toc369194621)

[5.2.3　アウトプット 41](#_Toc369194622)

[5.3　参考文献 43](#_Toc369194623)

**第1章**

**序論**

1 序論

* 1. 本章の構成

第1章では，本論文の序論について述べる．研究背景，目的，方法，プロジェクト

ネジメントとの関係，本論文の構成について述べる．

* 1. 研究の背景

ソフトウェア開発はバージョン管理システムを用いて行われるのが一般的である．バージョン管理システムが一般的になった理由は， プロジェクトを運営していく上でいろいろな場面で役立つということである． 開発者どうしのコミュニケーション，リリース管理，バグ管理，コードの安定性の確保，安心して新機能を実験できる環境，各開発者の権限の管理など，あらゆる場面でバージョン管理が利用することができるのである．バージョン管理システムは，これらの内容をひとまとめにして管理する．その中心となる機能が，変更管理である．これは，プロジェクト内のファイルが変更されるたびに，その変更についてのメタデータ（更新日や更新者など）を収集する仕組みである．そして，あとからその変更の内容を再現できるようにするのである．つまり，変更が発生した単位で情報を管理する仕組みといえるのである．

バージョン管理システムはソースコード管理システムなどとも呼ばれ，大規模な開発を行う際には必須と言っても過言ではないのである．また，大規模な開発だけでなく小規模な開発や個人による開発においても，ファイルの変更履歴の記録やバックアップといった用途で活用されており，プロジェクト内のさまざまなファイルの変更履歴を管理するためのテクノロジーや習慣を組み合わせたものである．バージョン管理システムを提供するサービス（以下，バージョン管理サービス）もSubversionやCVS（Concurrent Versions System），Git，Mercurial，Visual SourceSafeなどと数多くある．

ソフトウェア開発プロジェクトの中心にバージョン管理システムをおくことができる．そのような開発体制においても，一般的なプロジェクトと同様に，PMBOKで提示されているようなWBS（Work Breakdown Structure）やEVM（Earned Value Management）のようなマネジメントの活動がバージョン管理サービスでサポートされることが望ましいのである．

プロジェクトマネジメントにおいて，プロジェクトの進捗状況や進捗に係わるリスクを把握し，コントロールするためにEVMが用いられる．EVMは，プロジェクトを計画通りに進行させ，進捗状況に対応した判断を行うのに必要であり重要である．GitHub（バージョン管理システム，Gitを提供するサービス．ギットハブ）のような既存のバージョン管理サービス上では，EVMを見ることはできないである．

そのため，バージョン管理システムを用いるプロジェクトマネジャーは，計画時のコストのデータと計画時の時間のデータを進行中の進捗時のコストのデータと進捗時の時間のデータを取り出して，別々に処理することによってEVMを描かなければならないのである．

* 1. 研究の目的

バージョン管理サービスのGitHubのデータから自動的にEVMを描くシステムを開発する．このシステムは，GitHubのIssueを利用する．Issueとは，タスクを管理するためのシステムおよびそこで管理されるタスクのことである．1つのタスクに1つのIssueが作成され，管理される．

このシステムにより，プロジェクトマネジャーを含むメンバー全員がプロジェクトの進捗状況を手軽に把握できるようになる．その結果，作業の遅延などへの対応（リスク・マネジメント）やスケジュール管理（タイム・マネジメント）が効率よく行えるようになることも期待できる．

* 1. 研究の方法

本研究では，バージョン管理サービスのGitHubを調査し，EVMに必要なデータとして，計画時のコストのデータと計画時の時間のデータ，進行中の進捗時のコストのデータ，進捗時の時間のデータをIssueに記述する仕様を決定する．そして，Issueに記述されたデータを抽出するプログラムを作成し，Issueから抽出されたデータをもとにEVMを描画するシステムを構築する．その後，過去のプロジェクトのデータを構築したEVMを描画するシステムに適用し，システムの運用とシステムの検証によるシステムテストを行う．

* 1. プロジェクトマネジメントとの関連

　本研究では，バージョン管理サービスのGitHubのデータの中からEVMに必要なデータを抽出し，そのデータをもとにEVMを自動的に作成できるシステムを構築する．このシステムにより，プロジェクトマネジャーを含むプロジェクトメンバー全員がプロジェクトの進捗状況を手軽に把握できるようになる．その結果，作業の遅延などへの対応が可能になり，リスク・マネジメントに貢献することができる．また，スケジュールの管理などへの対応も可能であり，タイム・マネジメントにも貢献することができる．

* 1. 本論文の構成

　第1章は，序論，第2章から第4章に関しては，各項目の基礎知識，問題点，必要性などをまとめていく．第5章から第6章までは実験の方法，考察，結果を記し結論をまとめていく．

　第1章では，研究の背景，目的，方法，プロジェクトマネジメントとの関連を記述していく．第2章では，ソフトウェア開発について調査する．ソフトウェア開発の歴史，成功事例，失敗事例，失敗事例の原因などを調査する．第3章ではGitHubの基礎知識を述べる．バージョン管理システムについての調査，バージョン管理システムの種類，成功事例などを述べる．第4章では，マネジメントについて述べる．本研究ではリスク・マネジメントとタイム・マネジメントについて述べる．

第5章ではEVMについて調査する．第6章では，システム構築に必要な情報をまとめる．第7章では検証結果から，考察を行う．

* 1. 参考文献

[1] Project Managemant Institute, PMBOK Guide, 第4版, PMI, p5-6. 2008．

[2] 松島造道, CVS-Subversionを使ったバージョン管理, SOURCEFORGE.JP, http://sourceforge.jp/magazine/08/09/09/1038233, 2013-09-26.

[3] Fogel Karl, バージョン管理 第3章 技術的な問題, オープンソースソフトウェアの育て方, http://producingoss.com/ja/vc.html, 2013-09-27.

第2章

ソフトウェア開発プロジェクトについて

# 2.ソフトウェア開発プロジェクトについて

## 2.1　本章の構成

　本章では，本研究はバージョン管理システムを中心におく開発体制になる前のソフトウェア開発プロジェクトについて調査する．ソフトウェア開発プロジェクトは以前ではどのように行われていたのかを調査し，ソフトウェア開発プロジェクトの歴史を知る．また成功事例，失敗事例，失敗事例の原因などを調査することによって，バージョン管理システムの必要性を明確にする．

## 2.2　ソフトウェア開発プロジェクトについて

　ソフトウェア開発プロジェクトはプロジェクトの開始時から多くの失敗の可能性があると予想される．ソフトウェア開発プロジェクトにはいくつかの特徴的な傾向がある．最近のソフトウェア開発プロジェクトの特徴は以下のようである．

・従来の開発に比べて期待される開発工期が相対的に短い．

・要求仕様が固まっていない状態で開発を始めねばならないケースが多い．

・さらに要求の追加・変更が開発中にも継続的に発生にする．

・利用すべき技術や製品が次々に登場・改訂される中で開発しなければならない．

・アプリケーション固有画面，Web画面，i-モード，そのほか携帯端末などのさまざまなインタフェイスに対応しなければならない．

経済産業省が2005年に公開した仕組みソフトウェア産業実態調査[5]によれば，80%以上

のプロジェクトの開発期間が1年未満であることがグラフから読み取れる．規模が増大しているにもかかわらず，期間は短いままである．結果として，プロジェクトに投入される技術者が増え，最近では数百人規模の開発体制も増加傾向にある．ソフトウェア開発プロジェクトでは開発期間が最近では6ヶ月未満で納品する傾向にある．

このグラフから開発期間が短いことがわかる．そのため，期間内に終わらせるためには，ソフトウェア開発プロジェクトのプロジェクトマネジャーはプロジェクトの進捗管理には細心の注意が必要になる．

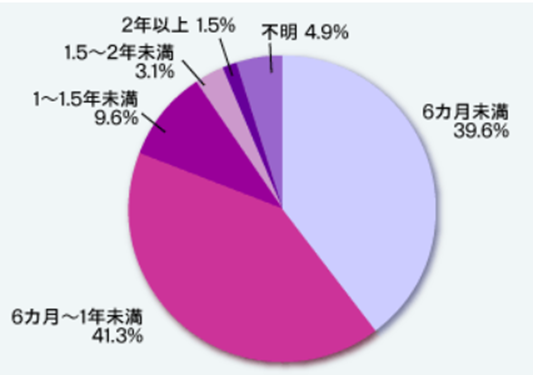


図1-1　ソフトウェア開発プロジェクトの平均的な開発期間[4]

## 2.3ソフトウェア開発の歴史

　1970年代の後半から1980年代にかけてITの大規模システム構築が急激に発生したが，プロジェクトマネジメント技術や方法論が未熟だったために，勘と度胸で実行せざるを得なかったのである．その結果，多くのプロジェクトに問題が発生したのである．こうした中で数少ない大規模プロジェクトを成功に導いたプロジェクト・リーダーがスーパーSEとして話題になったのである．このためにスーパーSEの育成方法が話題となり，各方面で色々試行したが，スーパーSEの育成は極めて困難なことが分かった．このことを境にスーパーSEを頼る以外の成功手法として「プロジェクトマネジメントの方法論」や「プロジェクト組織の仕組み」などを研究したのである．また，企業経営手法やスポーツチームの成功事例から，人間関係や心理的側面もプロジェクトマネジメントの重要な要素の一部だということも分かったのである．更に，プロジェクト・チームの組織能力向上という視点から，CMM（組織の能力を評価する方法論）が米国で発案され，CMMを組織能力評価のみならず組織能力向上に利用することも始まったのである.[5]

## 2.4　ソフトウェア開発プロジェクトの成功事例

　ここでは，過去に行われたプロジェクトで成功した事例を考察する．事例から，なぜこのソフトウェア開発プロジェクトは成功したのかを考察する．

### 2.4.1　成功事例

　このプロジェクトは日本の企業と中国の企業が協力して行うプロジェクトである．プロジェクト全体のプロジェクトマネジャーは10年以上も開発経験をもつ人物である．内容は要件定義，外部設計は日本の企業が担当し，内部設計，プログラム開発以降は中国，システムテストは日本で行うということになったのである．中国の企業は品質，進捗の管理を独立して行われ，日本の企業へは毎週報告することに決められていたのである．使用についての質疑応答，変更の管理は中国の企業と日本の企業とで決められたやり方でやり取りしていたのである．中国の企業でプロジェクト進行中の問題解決として，日本の企業の技術者にレビューを頼むことになったのである．プロジェクトは当初，決められた計画内のスケジュール中に成果物を納品することができ，品質の面でもバグ数は許量範囲内であり，日本の企業に中国の企業から納品した後のシステムテストは，無事に成功したのである．その後，ユーザーテストを終えて稼働することができたのである．この開発体形は，オフショア開発である．

### 2.4.2　成功事例の考察

　このソフトウェア開発プロジェクトの成功事例から成功した要因は，開発プロジェクトを担当したプロジェクトマネジャーの経験が豊富だという点にある．10年以上も開発の経験があったことで必要作業項目も詳細に指示をすることができる．また，進捗管理に関しても細かい双方の企業での情報のやり取りも毎週やることによりプロジェクトマネジャーが進捗状況を把握することもできる．その結果，中国の企業での進行中の問題を解決するために日本の企業の技術者にレビューを任せることにより，問題解決ができたのである．プロジェクトはスケジュール中に成果物を納品することができ，その後のシステムテストとユーザーテストも完了することができたのである．このソフトウェア開発プロジェクトの開発体形はオフショア開発というプロジェクトの参加メンバーが離れたところにいながら，ネットワークを通じてプロジェクトを進行していく開発体形である．プロジェクトに重要なコミュニケーションをとりにくい環境である中，進捗管理のために毎週の報告をすることでコミュニケーションを可能にし，進捗状況を把握することができたのである．

## 2.5　ソフトウェア開発プロジェクトの失敗事例

　ここでは，過去に行われたソフトウェア開発プロジェクトの事例を考察し，事例の原因からソフトウェア開発プロジェクトにおけるリスク分析を行う．

### 2.5.1　失敗事例

　このソフトウェア開発プロジェクトはデンバー空港の自動手荷物運搬システムを開発するプロジェクトである．1993年の10月29日に，その当時の空港としては非常に巨大で最新の設備を備えたデンバー国際空港がオープンする予定でした．しかし，なぜかオープンの日になっても自動手荷物運搬システムだけはうまく起動しなかったのである．その結果として，空港のオープンが16ヶ月ほど遅延して1995年の2月28日にオープンしたのである．手荷物運搬システムがないのは空港として常識的に考えられないという一般的な考えから空港はオープンの遅延をよぎなくされたのである．その遅れによって，問題の自動手荷物運搬システムを構築していた会社は，デンバー市から空港のオープンの遅延の代償として1日ごとに12,000ドル（1994年の為替レート換算で約120万円）を請求されることになってしまったのである．さらに，起動しなかった自動手荷物運搬システムのバックアップシステムを構築するために，5,000万ドル（同じく1995年で約45億円）を追加請求されたのである．さらに，構築したバックアップシステムが2000年問題に対応していなかったため，さらに巨額のコスト10億円が費やされたのである．

### 2.5.2　失敗原因

　問題の原因が明確にはなってないが，問題定義としてリスク分析や進捗状況の管理が曖昧だと考えられる．また，組み込みテストを行う場合はCPUが熱で暴走させないために，CPUを休み休みテストを行わなければならないのである．そうした遅延やあるいは手戻りがあり進捗計画時に遅延などのリスク計画とそのリスクによる作業の遅延を予想したスケジュールの管理が必要だったのである．さらに，このプロジェクトでは要求が変更されることがたびたびあったのである．このプロジェクトはソフトウェア開発会社だけでなく，デンバー市や航空会社も係わっているプロジェクトであったために，開発チーム以外からも仕様変更などが多数突きつけられてしまったのである．

　つまりこのプロジェクトは，変更要求の多い「外的要因」を抱えていたのにもかかわらず，トラブル耐性の低い体質であったうえに，それらへの対策を行うはずのプロジェクトマネジャーを失うという「内的要因」が重なった結果が原因である．

## 2.6　事例から得られたこと

　成功事例と失敗事例から得られたことは，成功事例では，進捗状況を把握するためにコミュニケーションを必要とするが，開発体形はプロジェクト関係メンバーが離れて活動するというオフショア開発であったが，毎週の報告をすることによりプロジェクトマネジャーが進捗状況を把握することができる．そして，問題が発生した場合でも対処できたということである．失敗事例では，失敗要因となる問題項目が多々ある．まず，内的要因はプロジェクトの進行中にプロジェクトマネジャーが変更になることである．次に，組み込みテスト時にはCPUが熱暴走させないために，CPUを休み休みテストを行うことにより，作業の遅延を招いてしまったのである．外的要因は，プロジェクト規模が大規模であることである．次に，顧客からの要件定義の変更の回数が多く，そのためにシステム開発の不具合も起きやすくなったのである．この場合の原因は，進捗管理を行うプロジェクトマネジャーがプロジェクトの進行中に変更になってしまったことにより，進捗状況に応じて対応できないことである．

## 2.7　参考文献

[1] 伊藤誠一, ソフトウェアの成功と失敗, 初版, 1997,

[2] 株式会社アイティネット, ITプロジェクト失敗の原因と対策, ITプロジェクト管理考,http://www.itnetinc.co.jp/ITPMopinion/projectfailurereason.htm, 2013-09-30.

[3] 国際大学法人東京農工大学, なぜソフトウェア開発にプロジェクト管理が必要か, ソフトウェア開発環境の概要, http://www.tuat.ac.jp/~asiaprog/courses/project/lesson03/, 2013-09-30.

[4] ITmedia Inc, ETSSは人材&スキル不足を解決できるか, MONOist, http://monoist.atmarkit.co.jp/mn/articles/0601/27/news122.html, 2013-09-15.

[5] システム管理者の会, IT業界におけるプロジェクトマネジメントの大まかな歴史, システム管理者の会ポータルサイト, http://www.sysadmingroup.jp/kh/rpt/001/34.html, 2013-10-01.

[6] 勝呂暖生, ソフトウェアプロジェクト管理, 株式会社翔英, 2012, p16-19.

[7] 北京英普思科技発展有限公司, 成功事例, INPROSYSTEM LED, http://www.inprosystem.com/Templates/cgsl\_jp.html, 2013-10-08.

第3章

バージョン管理システムについて



# 3.　バージョン管理システム

## 3.1　本章の構成

　本章では，最終的な本研究の目的で使用するバージョン管理システムについての基本知識や種類などを述べる．その後，Gitの基本情報やGitHubの現状について述べ，その後，GitとGitHubの違いについて述べた後，専門用語の説明を解説しながら，機能について述べる．

## 3.2　バージョン管理システムについて

　バージョン管理システムは，ファイルの履歴を管理するシステムである．何らかの作業によって生成されたファイルについての複数の履歴を記録し，後から古い履歴の取り出しや，差分の参照ができる仕組みである．このファイルの履歴をリポジトリ，履歴の1つ1つをコミットと呼び，履歴をリポジトリに追加や登録することをコミットすると言うのである．またバージョン管理システムソフトウェアによっては，ファイルの削除や移動の履歴を確認する機能や，特定の利用者がファイルの管理する権限を獲得するロック機能，複数の変更を統合するマージ機能がある．

　バージョン管理システムは，複数人で同一のファイルを編集する必要があるときに，バージョン管理システムの機能が役立つのである．例えば，テキストファイルをファイルサーバに置いて共有していたとする．この状態で編集やコピーを続けていくと，誰が最新版を持っているか，また，誰がいつ，どこを変更したなどの履歴がわからなくなってしまうのである．ここでこのファイルをバージョン管理システムで管理すれば，変更点と変更者，変更日時を管理できるようになるのである．

　またソフトウェア開発においては，バージョン管理システムは特に有効に稼働するのである．変更点を管理できるので，障害がいつ発生したのか，どの時点から問題となっていたのか，いつ修正されたのかなどを簡単に調査できるからである．

## 3.3　バージョン管理システムの種類

　このバージョン管理システム（Version Control System，以下VCS）は，管理の方法から3つの分けることができる．個別バージョン管理システム，プロジェクトメンバーのためのサーバ型の集中型バージョン管理システム，そしてGitに代表される分散型バージョン管理システムである．以下にVCSの種類と代表とするVCSソフトウェアを表3-1に示す．これら3つのVCSについて説明する．

表3-1　バージョン管理システムの種類と概要[2]

|  |  |
| --- | --- |
| VCSの種類 | 概要 |
| 個別VCS | ファイル単位で個別システムのバージョン管理を行う．（SCCS，RCS） |
| 集中型VCS | リポジトリをサーバで一元管理し，コミットなどの操作はプロジェクトメンバーから行う（CVS，Subversion，Perforce） |
| 分散型VCS | リポジトリをプロジェクトメンバーでも管理することができる．このプロジェクトメンバー間でリポジトリの連携ができる．（Git，Bazaar，Mercurial，BitKeeper） |

### 3.3.1　個別バージョン管理システム

　初期のオープンソースソフトウェアとしてのバージョン管理システムといえばRCSが挙げられる．RCSはファイル単位での管理しかできないため，バージョン管理については限定的なことしかできなかったのである．個人での利用であれば問題ないかもしれないが，複数人が利用するプロジェクトでは何らかの運用上の工夫が必要になることが多くなるため，集中型バージョン管理システムであるCVSなどへと発展していくことになったのである．

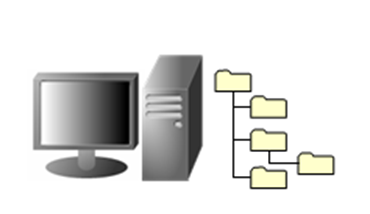


図3-1　個別バージョン管理システム（個人開発）[6]



図3-2　個別バージョン管理システム（少数でのチーム開発）[6]

### 3.3.2　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，複数人で使う場合は通常プロジェクトメンバー・サーバ型の構成をとり，リポジトリはサーバ上に格納されているのである．リポジトリに対する操作はプロジェクトメンバーから行うが，ネットワーク的に接続していなければならないという制約がある．集中型バージョン管理システムとしては，CVSのほかにSubversionや商用ソフトウェアであるPerforceなどがある．これらは現在でも多く利用されているが，次第に分散型バージョン管理システムに移行しつつある．

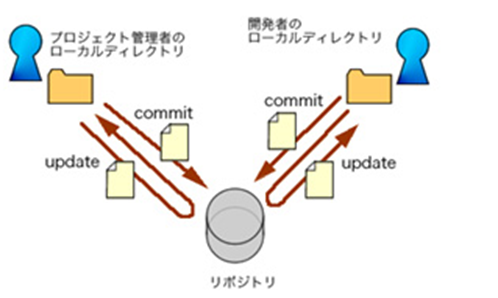


図3-3　集中型バージョン管理[7]

### 3.3.3　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では，リポジトリをサーバ上ではなく，プロジェクトメンバーであるコンピュータ上にも作成するのである．コミットの参照や差分を取得する場合に，手元にあるリポジトリにアクセスするため，集中型にあったネットワーク接続が必要であるという欠点がないのである．また，ローカルファイルシステムにリポジトリがあるので高速に動作することも，利点のひとつである．さらに変更点を送受信する仕組みがあるため，複数のリポジトリ間で連携できるのである．

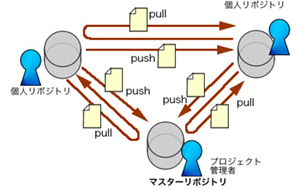


図3-4　分散型バージョン管理システム[7]

以下の表3-2でオープンソースプロジェクトと概要，採用されている分散型バージョン管理システムを示す．

表3-2　オープンソースプロジェクトで採用されている分散型バージョン管理システム[3]

|  |  |  |
| --- | --- | --- |
| プロジェクト | 概要 | VCS |
| Linux | カーネル | Git |
| Android | OS | Git |
| Perl | 言語 | Git |
| Ruby | 言語 | Subversion |
| Ruby on Rails | Webアプリケーションフレームワーク | Git |
| X.Org | グラフィカルウィンドウシステム | Git |
| Samba | Windowsとのファイル共有 | Git |
| Emacs | エディタ | Bazaar |
| Vim | エディタ | Mercurial |
| Mozilla Firefox | ブラウザ | Mercurial |
| Chromium | ブラウザ | Subversion |
| FreeBSD | OS | Subversion |

## 3.4　Gitについて

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

　リポジトリ間連携はネットワーク通信（ssh，http，git，プロトコルなど）やメールを経由して行うのである．ほかにもリポジトリを共有リポジトリとして公開する仕組みや，ユーザー管理と組み合わせる方法があり，集中型バージョン管理システムのような利用形態もとれるのである．

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

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

### 3.4.1　Gitの歴史

　Linuxの創始者Linus TorvaldsがGitの開発を始めたのは2005年の春頃のことである．それまで，Linuxカーネル開発のソース管理にはBitKeeperというバージョン管理システムが用いられていたのである．これはBitMover社製の商用のバージョン管理システムである．Bitkeeperは，先進の分散型バージョン管理システムで，カーネルプロジェクトが採用した当時，オープンソースの世界にはこれに匹敵する分散型バージョン管理システムの使用が不可欠であったのである．

　同社はオープンソースプロジェクトに対して無料使用のライセンスを与えていたのですが，どれも機能・性能面で満足できるものではありませんでした．そこでLinuxは，「カーネルプロジェクトから2週間休み」を宣言して現在のGitの原形となるプログラム郡を書き，これを公開したのである．Linus Torvalds氏自身が作成したという背景があり，さらに機能や性能に好評だったためGitはプログラマーに受け入れられる理由の1つである．公開当初は，ハッカーにしか使えないぐらい荒削りのプログラムだったらしいが，多くの開発者の協力を経て，今では世界中のプログラマーに利用されるようになったのである．

### 3.4.2　Gitの特徴

　Gitは，ファイル自身，ファイルの集合としてのツリー，そしてコミット情報という3つの情報を管理しているのである．それぞれの情報はハッシュ値をもとに管理されており，このハッシュ値とはファイルが同一かどうかの判定にも利用されているのである．また，Gitではコミット情報を，SubversionやCVSのような差分管理ではなく，ファイルをそのまま保持しているというのも特徴である．

　さらに時間的な変遷を管理する仕組みreflogや，コミットをメールで送受信する仕組みもある．reflogは時系列でのコミットやブランチ切り替えなど，リポジトリに関する変更を管理する機能である．例えば先週の変更はどのようなものだったのかを後で確認するときなどに効果を発揮するのである．コミットをメールで送受信する仕組みの機能は特にオープンソースプロジェクトで活躍する機能である．コミット情報をパッチとしてメールでやりとりできるため，そのパッチについてプロジェクトのメーリングリストなどで議論することができる．また，ローカルコンピュータ上にリポジトリを持つため，場所や時間，あるいはネットワーク接続の状態に関わらずコミットを扱えるという利点がある．

### 3.4.3　Gitの利用状況

　GitHubの登場で広がるGitの利用率は上昇する傾向である．2008年に登場したGitHubは，Gitリポジトリのホスティングサービスである．ソースコードをもとにソーシャルネットワーキングサービスでもあり，気軽に自分のリポジトリを公開できるのである．近年では多くのオープンソースプロジェクトがGitHubで公開されているのである．GitHubには，他のユーザーのリポジトリの複製を簡単に作成できる「Fork」という機能があり，変更の送信や複製元の変更への追従もできるのである．これらはすべてGitの基本機能であり，実際にどのように動作するかを理解すれば，GitHubを活用しやすいのである．

### 3.4.4　GitとGitHubの違い

　Gitでは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しているのである．GitHubは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サービスのことである．つまり，GitHubで公開しているソフトウェアのソースコードは，すべてGitで管理されているのである．

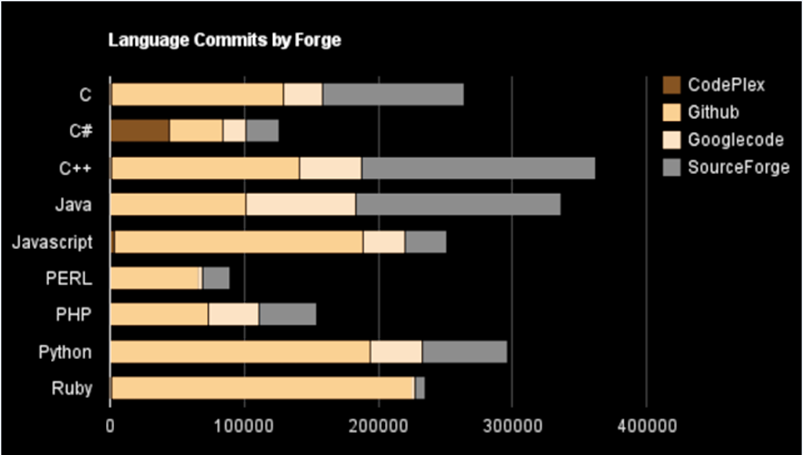
## 3.5　GitHubについて

　GitHubはコードを共有したり，公開したりするためプログラマーのソーシャルネットワーキングサービスとされるのである．GitHubはGitをサポートするためのサービスである．

Gitはプロジェクトの改訂履歴を管理し保存するものである．そして，プログラムのコードに利用されることが多いが，WordのドキュメントでもFinal Cutのプロジェクトでも，どんなタイプのファイルでも管理できるのである．つまり，コンピュータのプログラムに限らず，どんなドキュメントでも，すべてのアップデート履歴を保存し管理できるファイルシステムである．また，GitHubは一般に公開しているコードのリポジトリ集を企業に売り込んだり，就職活動にも役立てている．就職活動においてSEやプログラマーを募集するIT関連業界ではGitHubのリポジトリを履歴書として選考する企業もでてきている．

## 3.6　GitHubの利用状況

　GitHubは，2012年月で約560万リポジトリをホスティングしており，年間稼働率99.69%となっている．参加しているプログラマーは世界中に存在しており，170万人を超えている．そして，現在の2013年は約530万人[8]に達している．



GitHub利用状況[7]

## 3.7　GitHubの基本用語

　GitやGitHubについて論述するためには，GitHubの基本用語を知っておく必要がある．ここでは，論述する内容に記載されている基本用語について以下に記述する．

・Gitリポジトリ：

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

・コミット（commit）：

　ファイルの変更履歴情報を閲覧したり，ファイルの変更を保存することである．

・共有リポジトリ：

　共有リポジトリとは，チームメンバーで共有するリポジトリで，ソースコードのメインバージョンが保存されている．

・ローカルリポジトリ：

　作業者のコンピュータ上にあるリポジトリである．

・インデックス：

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

・アドオン（add）：

　ソフトウェアに追加される拡張機能のことである．

・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．

・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．

・フォロー：フォローができる．

・フォロワー：ユーザーが他のユーザーをフォローすることができる．

・スター：「お気に入り」，「いいね！」のようなもの．つけられたスターはカウントされ，スターのカウントが多いとほかのユーザーからの注目をされていると認識できる．また，そのデータは活発に活動（更新）されているとみなされる．

・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．

・ウィキ（Wiki）：

　その場にページが表示され，ドキュメントやコードがかける場所である．

・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．

## 3.8　GitHubの機能

GitHubはGitをサポートするためにさまざまな機能を備えている．以下には機能について記述する．

・フォーク（forking）：

1つのプロジェクトが複数に分岐していくことである．だれかのリポジトリをほかの人がコピーし改変していくことである．

・ライト（write）：

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

・プルリクエスト：

　フォークを行い，コピーしたリポジトリのデータにライトしたことをオリジナルのリポジトリのユーザーに通知を送信することである．

・マージ（merge）:

プルリクエストした人がその人のリポジトリに対して行われた変更を自分のリポジトリにも取り入れることである．

・イシュー（Issue）:

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

・ウォッチ機能：

　他の人のデータを見ることができる．他の人の進捗状況やプロジェクト内容を閲覧できる．

・グループ機能：

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

・検索機能：

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

・ブランチ：

ブランチとは，ある時点のソースコードを分岐させて開発するもので，ソフトウェアのメインの開発をもとに，そこから派生したコードの部分のことである．

・リモートブランチ：

　リモートブランチとは，共有リポジトリの中にあるブランチのことである．

## 3.9　参考文献

[1] 岩松信洋, Gitによるバージョン管理, 初版, オーム社, 2011, p1, p4-5.

[2] 濱野純, 入門Git, 株式会社秀和システム, 初版, 2010, p1.

[3] 片岡巌, WEB+DB PRESS, vol.69, 初版, 2012, p20.

[4] 国立大学法人東京農工大学, 開発環境の概要とバージョン管理ツール，ドキュメントテーションツールを知る, ソフトウェア開発環境の概要, http://www.tuat.ac.jp/~asiaprog/courses/oss/lesson03/, 2013-09-29.

[5] ITmedia Inc, 分散バージョン管理システムGit入門, ITmediaエンタープライズ, http://www.itmedia.co.jp/enterprise/articles/0902/09/news019.html, 2013-09-29.

[6] GitHub,Inc, 5.4M developers, https://github.com/features/community, 2013-09-29.

[7] fusion, GitHubエピローグ, Yahoo Japan Corporation, http://blogs.yahoo.co.jp/igproj\_fusion/12541553.html, 2013-09-29.

[8] Iwatani, そもそもGitHubとは一体何か, TechCrunch Japan, http://jp.techcrunch.com/2012/07/15/20120714what-exactly-is-github-anyway/, 2013-09-29.

[9] 片岡巌, 開発ツール徹底攻略, 第2版, 2013, p49-53.

[10] 岡本隆, 2012, Gitポケットリファレンス, p1, p19-p20

第4章

システム構築について

# 4.　システム構築について

## 4.1　本章の構成

　この章では，最終的な本研究の目的で開発するシステムを構築する際に使用するGitHubのAPI と目的に必要なデータを抽出する基準としてEVMについて記述をする．

## 4.2　APIについて

　API（Application Programming Interface，アプリケーション・プログラミング・インターフェース）とは，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，ウィンドウ制御，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のである．

　つまり，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に，非常に簡潔にプログラムができるように設定されたインタフェースのことである．

### 4.2.1　APIを公開する・公開されるメリット

　Web製作者にとっては，他社の膨大なデータベースや機能を無料で利用できるため，Webサイトの開発のコストを大幅に削除できるので，効率的に制作できる．このため，個人や小資本の会社でも，人気のWebサイトを低コストで効率的に作成されることが可能になったのである．

　また，API提供会社にとっては，自社のみでは考え付かないようなWebサービスや自社のみでは実現できないようなWebサービスを外部の人間が作成してくれるため，API利用サイトから自社のサイトへユーザーが流入してくるのを期待できる．さらに，自社のデータベースや機能を間接的に自社のサイトを利用してくれるユーザーが増加し，それが結果としてデータを集積するスピードが速まり，競争優位に立つことを期待できるのである．

## 4.3　GitHubのAPIの種類

　GitHubのAPIとはGitHubの機能を抽出する時に必要なコードである．GitHubのAPIは機能ごとに設定されている．IssueにはIssueのAPIがあり，マイルストーンには，マイルストーンのAPIがある．以下に各機能ごとのAPIを記述する．

・Issue（イシュー）のAPIは以下に記述する．

表4-1　IssueのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 取得 | /repos/{userName}/{repoName}/issues/{id} | 選択されたIssueを表示する． |
| 取得 | /repos/{userName}/{repoName}/issues | リポジトリのIssueの一覧を表示する． |
| 取得 | /issues | ユーザーのIssueの一覧を表示する． |
| 送信 | /repos/{userName}/{repoName}/issues/{id} | 指定されたリポジトリのIssueを編集する． |
| 送信 | /repos/{userName}/{repoName}/issues | 指定されたリポジトリにIssueを作成する． |

・Issue Events（イシューイベント）のAPIは以下に記述する．

表4-2　Issue EventsのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 取得 | /repos/{userName}/{repoName}/issues/events/{id} | リポジトリの指定したIssueの変化を表示する． |
| 取得 | /repos/{userName}/{repoName}/issues/events | リポジトリのIssueの変化を表示する． |
| 取得 | /repos/{userName}/{repoName}/issues/{issueId}/events | リポジトリの指定したIssueの変化を表示する． |

・Issue Comments（イシューコメント）のAPIは以下に記述する．

表4-3　 Issue CommentsのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /repos/{userName}/{repoName}/issues/comments/{id} | リポジトリのIssueのコメントを削除する． |
| 取得 | /repos/{userName}/{repoName}/issues/comments/{id} | リポジトリのIssueのコメントを表示する． |
| 取得 | /repos/{userName}/{repoName}/issues/{id}/comments | リポジトリのIssueの返信コメントを表示する． |
| 送信 | /repos/{userName}/{repoName}/issues/{id}/comments | リポジトリのIssueにコメントを作成する． |
| 送信 | /repos/{userName}/{repoName}/issues/comments/{id} | リポジトリのIssueにコメントを編集する． |

・Events（イベント）のAPIは以下に記述する．

表4-4　 EventsのAPIと意味

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

・Label（ラベル）のAPIは以下に記述する．

表4-5　 LabelのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /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． |
| 削除 | /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． |
| 削除 | /repos/{userName}/{repoName}/issues/{id}/labels/{id} | 指定したリポジトリのIssueからラベルを削除する． |
| 取得 | /repos/{userName}/{repoName}/milestones/{id}/labels | すべてのマイルストーンを表示する． |
| 取得 | /repos/{userName}/{repoName}/labels | リポジトリ内のラベルがついたものの一覧を表示する． |
| 取得 | /repos/{userName}/{repoName}/labels/{id} | リポジトリ内の選択されたラベルを表示する． |
| 送信 | /repos/{userName}/{repoName}/issues/{id}/labels | リポジトリ内のIssueにラベルを追加する． |
| 送信 | /repos/{userName}/{repoName}/issues/{id}/labels | リポジトリ内のラベルを更新する． |
| 送信 | /repos/{userName}/{repoName}/labels/{id} | リポジトリ内にラベルを作成する． |
| 交換 | /repos/{userName}/{repoName}/issues/{id}/labels | Issueのすべてのラベルを交換する． |

・Gists（ギスト）のAPIは以下に記述する．

表4-6　GistsのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /gists/{id}/star | ギストのスターを解除する． |
| 削除 | /gists/{id} | ギストのスターを削除する． |
| 取得 | /gists | 認証されたユーザーのギストを一覧表示するか，匿名で呼び出された場合はすべての共有ギストを表示する． |
| 取得 | /users/{userName}/gists | ユーザー一覧のギストを表示する． |
| 取得 | /gists/public | 共有ギストの一覧を表示する． |
| 取得 | /gists/{id}/star | 指定したギストにスターがついているかを表示する． |
| 取得 | /gists/starred | ユーザーがスターをつけたギストの一覧を表示する． |
| 取得 | /gists/{id} | 指定したギストの情報を表示する． |
| 送信 | /users/{userName}/gists | 新しいギストを作成する． |
| 送信 | /gists/{id}/fork | ギストをフォークする． |
| 送信 | /gists/{id} | ギストを編集する． |
| 更新 | /gists/{id}/star | ギストのスターを更新する． |

・Gists Comments（ギストコメント）のAPIは以下に記述する．

表4-7　Gists CommentsのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /gists/comments/{id} | ギストコメントを削除する． |
| 取得 | /gists/{gistId}/comments | ギストコメントの一覧を表示する． |
| 取得 | /gists/comments/{id} | ギストコメントの返信を表示する． |
| 送信 | /gists/comments/{id} | ギストコメントを編集する |
| 送信 | /gists/{gistId}/comments | ギストコメントを作成する． |

・GitData（ギットデータ）のAPIは以下に記述する．

表4-8　GitDataのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 取得 | /repos/{userName}/{repoName}/git/commits/{sha} | 使用ユーザーはGitリポジトリからコミットを表示する． |
| 取得 | /repos/{userName}/{repoName}/git/trees/{sha} | 使用ユーザーはGitリポジトリからツリー形式を表示する． |
| 取得 | /repos/{userName}/{repoName}/git/tags/{sha} | 使用ユーザーはGitリポジトリのタグを表示する． |
| 取得 | /repos/{userName}/{repoName}/git/refs/tags | 使用ユーザーのGitリポジトリのすべての証明書タグを表示する． |
| 取得 | /repos/{userName}/{repoName}/git/refs | 使用ユーザーのGitリポジトリのすべての証明書を表示する． |
| 取得 | /repos/{userName}/{repoName}/git/refs/{ref} | 使用ユーザーはGitリポジトリの証明書を表示する． |
| 取得 | /repos/{userName}/{repoName}/git/blobs/{sha} | 使用ユーザーはGitリポジトリのブロブを表示する． |
| 取得 | /repos/{userName}/{repoName}/git/trees/{sha}?recursive=1 | 使用ユーザーはGitリポジトリの1つ前のツリーを入手する． |
| 送信 | /repos/{userName}/{repoName}/git/blobs | 使用ユーザーのGitリポジトリにブロブを作成する． |
| 送信 | /repos/{userName}/{repoName}/git/trees | 使用ユーザーのGitリポジトリにツリーを作成する． |
| 送信 | /repos/{userName}/{repoName}/git/refs/{ref} | 使用ユーザーのGitリポジトリ内の証明書を更新する． |
| 送信 | /repos/{userName}/{repoName}/git/tags | 使用ユーザーのGitリポジトリ内にタグオブジェクトを作成する． |
| 送信 | /repos/{userName}/{repoName}/git/commits | 使用ユーザーはGitリポジトリ内にコミットを作成する． |
| 送信 | /repos/{userName}/{repoName}/git/refs/{ref} | 使用ユーザーのGitリポジトリの証明書を作成する． |

・Repo Download（ダウンロード・リポジトリ）のAPIは以下に記述する．

表4-9　Repo DownloadのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /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． |
| 取得 | /repos/{userName}/{repoName}/ downloads/{id} | リポジトリのダウンロードデータを表示する． |
| 取得 | /repos/{userName}/{repoName}/downloads | リポジトリのダウンロードデータの一覧を表示する． |
| 送信 | /repos/{userName}/{repoName}/downloads | リポジトリから新しくダウンロードする． |

・Repo Deploy Key（デプロイ・リポジトリ）のAPIは以下に記述する．

表4-10　Repo Deploy KeyのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /repos/{userName}/{repoName}/keys/{id} | リポジトリの暗号鍵を削除する． |
| 取得 | /repos/{userName}/{repoName}/keys/{id} | リポジトリの暗号鍵を表示する． |
| 取得 | /repos/{userName}/{repoName}/keys | リポジトリの暗号鍵の一覧を表示する． |
| 送信 | /repos/{userName}/{repoName}/keys | リポジトリの暗号鍵を作成する． |
| 送信 | /repos/{userName}/{repoName}/keys/{id} | リポジトリの暗号鍵を編集する． |

・Repo Fork（フォーク・リポジトリ）のAPIは以下に記述する．

表4-11　Repo ForkのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 取得 | /repos/{userName}/{repoName}/forks | フォークデータの一覧を表示する． |
| 送信 | /repos/{userName}/{repoName}/forks | フォークを作成する． |

・Repo Hook（フック・リポジトリ）のAPIは以下に記述する．

表4-12 Repo HookのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /repos/{userName}/{repoName}/hooks/{id} | フックを削除する． |
| 取得 | /repos/{userName}/{repoName}/hooks | フックの一覧を表示する． |
| 取得 | /repos/{userName}/{repoName}/hooks/{id} | 指定したフックを表示する． |
| 送信 | /repos/{userName}/{repoName}/hooks | フックを作成する． |
| 送信 | /repos/{userName}/{repoName}/hooks/{id} | フックを編集する． |
| 送信 | /repos/{userName}/{repoName}/hooks/{id}/test | フックのテスト（検証）を行う． |

・PullRequest（プルリクエスト）のAPIは以下に記述する．

表4-13　PullRequestのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 取得 | /repos/{userName}/{repoName}/pulls | リポジトリ内のプルリクエストの一覧を表示する． |
| 取得 | /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表示する． |
| 取得 | /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表示する． |
| 取得 | /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． |
| 取得 | /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表示する． |
| 送信 | /repos/{userName}/{repoName}/pulls/{id} | プルリクエストを更新する． |
| 送信 | /repos/{userName}/{repoName}/pulls | プルリクエストを作成する． |
| 更新 | /repos/{userName}/{repoName}/pulls/{id}/merge | マージをプルリクエストする． |

・Pull Request Review Comment（プルリクエスト・レビューコメント）のAPIは以下に記述する．

表4-14　Pull Request Review CommentのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /repos/{userName}/{repoName}/pulls/comments/{id} | プルリクエストのコメントを削除する． |
| 取得 | /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． |
| 取得 | /repos/{userName}/{repoName}/pulls/{id} | プルリクエストへの指定されたコメントを表示する． |
| 送信 | /repos/{userName}/{repoName}/pulls/comments/{id} | コメントを編集する． |
| 送信 | /repos/{userName}/{repoName}/pulls/{id}/comments | コメントを作成する． |

・Milestone（マイルストーン）のAPIは以下に記述する．

表4-15　MilestoneのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /repos/{userName}/{repoName}/milestones/{id} | マイルストーンを削除する． |
| 取得 | /repos/{userName}/{repoName}/milestones | マイルストーンの一覧を表示する． |
| 取得 | /repos/{userName}/{repoName}/milestones/{id} | 指定したマイルストーンの表示する． |
| 送信 | /repos/{userName}/{repoName}/milestones | マイルストーンを作成する． |
| 送信 | /repos/{userName}/{repoName}/milestones/{id} | マイルストーンを更新する． |

・User（ユーザー）のAPIは以下に記述する．

表4-16　UserのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 取得 | /user | 認証されたユーザーを表示する． |
| 取得 | /users/{userName} | 指定したユーザーを表示する． |
| 送信 | /user | 認証されたユーザーを更新する． |

・User Email（ユーザー・メール）のAPIは以下に記述する．

表4-17　User EmailのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /user/emails | 電子メールアドレスを削除する． |
| 取得 | /user/emails | ユーザーの電子メールアドレスの一覧を表示する． |
| 送信 | /user/emails | 電子メールアドレスを追加する． |

・User Follower（ユーザー・フォロワー）のAPIは以下に記述する．

表4-18　User FollowerのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /user/following/{userName} | フォローを削除する． |
| 取得 | /user/following | 認証しているユーザーのフォローしている一覧を表示する． |
| 取得 | /users/{userName}/followers | フォロワーの一覧を表示する． |
| 取得 | /user/followers | 認証されたユーザーのフォロワーの一覧を表示する． |
| 取得 | /users/{userName}/following | フォローしているユーザーの一覧を表示する． |
| 取得 | /user/following/{userName} | フォローしているかどうかを表示する。 |
| 送信 | /user/following/{userName} | ユーザーをフォローする。 |

・User Public Key（ユーザー暗証キー）のAPIは以下に記述する．

表4-19　User Public KeyのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /user/keys/{id} | 公開鍵を削除する． |
| 取得 | /user/keys | 公開鍵の一覧を表示する． |
| 取得 | /user/keys/{id} | 指定した公開鍵を表示する． |
| 送信 | /user/keys/{id} | 公開鍵を更新する． |
| 送信 | /user/keys | 公開鍵を作成する． |

・Repos（リポジトリ）のAPIは以下に記述する．

表4-20　ReposのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 取得 | /repos/{userName}/{repoName}/branches | ブランチの一覧を表示する． |
| 取得 | /user/repos | 認証済みユーザーのリポジトリの一覧を表示する． |
| 取得 | /users/{userName}/repos | 指定したユーザーの共有リポジトリの一覧を表示する． |
| 取得 | /repos/{userName}/{repoName} | ユーザーはリポジトリを表示する． |
| 取得 | /orgs/{org}/repos | 指定した組織の共有リポジトリの一覧を表示する． |
| 取得 | /repos/{userName}/{repoName}/ contributors | 貢献者の一覧を表示する． |
| 取得 | /repos/{userName}/{repoName}/languages | 言語の一覧を表示する． |
| 取得 | /repos/{userName}/{repoName}/teams | チームの一覧を表示する． |
| 取得 | /repos/{userName}/{repoName}/tags | タグの一覧を表示する． |
| 送信 | /user/repos | 認証されたユーザーのためにリポジトリを作成する． |
| 送信 | /repos/{userName}/{repoName} | ユーザーのリポジトリを編集する． |
| 送信 | /orgs/{org}/repos | 組織のリポジトリを作成する． |

・Repo Commit（リポジトリ・コミット）のAPIは以下に記述する．

表4-21　Repo CommitのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /repos/{userName}/{repoName}/comments/{id} | コミットコメントを削除する． |
| 取得 | /repos/{userName}/{repoName}/commits | コミットの一覧を表示する． |
| 取得 | /repos/{userName}/{repoName}/commits/{sha} | 指定したコミットを表示する． |
| 取得 | /repos/{userName}/{repoName}/comments | コミットへのコメントの一覧を表示する． |
| 取得 | /repos/{userName}/{repoName}/commits/{sha}/comments | 指定したコミットへのコメントを表示する． |
| 取得 | /repos/{userName}/{repoName}/comments/{id} | 指定したコメントを表示する． |
| 取得 | /repos/{userName}/{repoName}/compare/{base}...{head} | 2つのコミットを比較する． |
| 送信 | /repos/{userName}/{repoName}/comments/{id} | コミットのコメントを更新する． |
| 送信 | /repos/{userName}/{repoName}/commits/{sha}/comments | コミットのコメントを作成する． |

・Repos Watching（鑑賞用リポジトリ）のAPIは以下に記述する．

表4-22　Repos WatchingのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /user/watched/{userName}/{repoName} | リポジトリのウォッチをやめる． |
| 取得 | /user/watched | ユーザーによってウォッチされているリポジトリの一覧を表示する． |
| 取得 | /repos/{userName}/{repoName}/watchers | ユーザーがウォッチしているリポジトリの一覧を表示する． |
| 取得 | /users/{userName}/watched | ユーザーがウォッチしたリポジトリの一覧を表示する． |
| 取得 | /user/watched/{userName}/{repoName} | 自分のリポジトリウォッチしているユーザーの一覧を表示する． |
| 更新 | /user/watched/{userName}/{repoName} | リポジトリをウォッチする． |

・Repo Collaborator（共有リポジトリ）のAPIは以下に記述する．

表4-23　Repo CollaboratorのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /repos/{userName}/{repoName}/collaborators/{user} | 共同リポジトリを削除する． |
| 取得 | /repos/{userName}/{repoName}/collaborators | 共同リポジトリの一覧を表示する． |
| 取得 | /repos/{userName}/{repoName}/collaborators/{user} | 共同リポジトリの協力者を表示する． |
| 更新 | /repos/{userName}/{repoName}/ collaborators/{user} | 共同リポジトリのユーザーを追加する． |

・Organization（組織）のAPIは以下に記述する．

表4-24　OrganizationのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 取得 | /user/orgs | 組織に認証されたユーザーの一覧を表示する． |
| 取得 | /orgs/{org} | 組織を表示する． |
| 取得 | /users/{userName}/orgs | 組織のユーザーの一覧を表示する． |
| 送信 | /orgs/{org} | 組織を編集する． |

・Organization Team（組織チーム）のAPIは以下に記述する．

表4-25　Organization TeamのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /teams/{id}/members/{userName} | チームメンバーを削除する． |
| 削除 | /teams/{id}/repos/{userName}/{repoName} | チームからリポジトリを削除する． |
| 削除 | /teams/{id} | チームを削除する． |
| 取得 | /teams/{id}/members/{userName} | チームメンバーを表示する。 |
| 取得 | /teams/{id}/members | チームのメンバーの一覧を表示する． |
| 取得 | /teams/{id} | チーム詳細を表示する． |
| 取得 | /orgs/{org}/teams | 組織内のチームの一覧を表示する． |
| 取得 | /teams/{id}/repos/{userName}/{repoName} | チーム内のリポジトリを表示する． |
| 取得 | /teams/{id}/repos | チームのリポジトリの一覧を表示する． |
| 送信 | /orgs/{org}/teams | チームを作成する． |
| 送信 | /teams/{id} | チームを編集する． |
| 更新 | /teams/{id}/members/{userName} | チームメンバーを追加する． |
| 更新 | /teams/{id}/repos/{userName}/{repoName} | チームリポジトリを追加する． |

・Organization Members（組織メンバー）のAPIは以下に記述する．

表4-26　Organization MembersのAPIと意味

|  |  |  |
| --- | --- | --- |
| APIの種類 | API | 意味 |
| 削除 | /orgs/{org}/public\_members/{userName} | ユーザーのメンバーシップを非公開にする． |
| 削除 | /orgs/{org}/members/{userName} | メンバーを組織から削除する． |
| 取得 | /orgs/{org}/members/{userName} | メンバーを表示する． |
| 取得 | /orgs/{org}/members | 組織のメンバーであるすべてのユーザー一覧を表示する． |
| 取得 | /orgs/{org}/public\_members/{userName} | パブリックメンバーを表示する。 |
| 取得 | /orgs/{org}/public\_members | パブリックメンバーの一覧を表示する． |
| 更新 | /teams/{id}/members/{user} | メンバーを追加する． |
| 更新 | /orgs/{org}/public\_members/{userName} | ユーザーの役割を変更する． |

## 4.4　EVMの概要

　EVMはプロジェクトの進捗や作業のパフォーマンスを，出来高の価値によって定量化し，プロジェクトの現在および今後の状況を評価する手法である．EVMでは計画および実績計上においてWBSを利用するのである．WBSは，プロジェクトで実施する作業を成果物作成を目的にして作業工程を段階的に表したものである．EVMを利用するためには，計画段階での各作業に対して予定している作業にかかるコスト，および作業の開始から終了予定を決定する．

　プロジェクトの監視の時には，各作業が終わり次第に作業を行ったメンバーはプロジェクトマネジャーに報告する必要がある．この時に作業の開始時や終了時などの作業の状態に従ってその時のEV（Earned Value）の実際に作業した時間とAC（Actual Cost）の実際にかかったコストを報告する．作業が終了時にはPV（Planned Value）の値と等しいのが理想的である．

　EVMでは，EV，AC，PVのそれぞれの値をもとに現在のプロジェクトの状況および今後のプロジェクトの方針を決める．

## 4.5　EVMの関連するプロジェクトマネジメント項目について

　プロジェクトマネジメントの知識エリアでは，コミュニケーション・マネジメントの領域における1つツールとして位置付けられるが，ほかにも，タイム・マネジメント，コスト・マネジメント，調達マネジメントとの関係性がある．また，EVMは，スコープ，スケジュール，コストなどプロジェクトの計画をベースラインとして設定し，そのベースラインを基礎としてプロジェクトの状況を監視し，監視の内容から分析と評価し，コントロールすることから，プロジェクトマネジメントの基本原則を取り扱う総合的な関係を持ち合わせている．

以下の表4-1はEVMの関連用語と説明を記述したものである．

表4-1　EVM関連用語と説明

|  |  |  |
| --- | --- | --- |
| 用語 | 正式名称 | 説明 |
| WBS | Work Breakdown Structure | プロジェクトに必要な作業項目を具体的に表示．作業スケジュールと進捗状況の把握が可能までに詳細化し，断層構造で表示したもの． |
| EVM | Earned Value Management | プロジェクトの進捗管理や今後の予測などをEV，AC，PVの各値を読み取り作業時間や実際にかかったコストを把握，管理をする方法 |
| EV | Earned Value | 実際にかかった時間 |
| AC | Actual Cost | 実際にかかったコスト |
| PV | Planned Value | 計画時に予想された時間とコスト |

## 4.6　参考文献

[1] 情報処理振興事業協会, 2002, EVM活用型プロジェクトマネジメント導入ガイドライン, 情報技術・市場評価基盤構築事業, p2-4

[2] 木村良一, EVMの基礎, http://www.mss.co.jp/technology/report/pdf/17-06.pdf#search='EVM%E5%88%86%E6%9E% 90%E3%81%AE%E5%9F%BA%E7%A4%8E', 2013-10-01.

[3] 田村啓, APIとは, プログラミング実習室, http://www.program-study.com/report/api, 2013-10-06.

[4] APIS.IO, APIS.IO, http://apis.io/GitHub, 2013-08-30.

[5] GitHub, GitHub Developer, http://developer.github.com/v3/issues/, 2013-08-30.

第5章

プロジェクトマネジメントについて

# 5.プロジェクトマネジメントについて

## 5.1　本章の構成

　この章では，本研究と関係のあるマネジメントの定義について記述する．本研究の目的に最も関係のあるプロジェクトマネジメント・プロセス群と知識エリアの分類の表を参照にプロジェクトマネジメント・プロセス群の監視・コントロール・プロセス群について記述する．

## 5.2　プロジェクト作業の監視・コントロール・プロセス群について

　プロジェクト作業の監視・コントロール・プロセス群の目的は，プロジェクトが計画通りに進んでいるかを確認するものである．計画通りでない場合は，その対処の内容と対処してよいかの伺い書を作成することである．プロジェクトマネジャーは，作業者から報告される実績報告を元に，計画通り（プロジェクトマネジメント計画通り）にプロジェクトが進んでいるか確認するのである．計画通りにプロジェクトが進行していない場合は，処置に必要なことをまとめて提言を行うのである．

　PMBOKによれば，プロジェクト作業の監視・コントロールは，プロジェクトマネジメント計画書に定義されたパフォーマンス目標を達成するため，進捗を追跡し，レビューし，統制するプロセスと定義されているのである．このプロセス郡が役立つ主な点は，プロジェクトマネジメント計画書との差異を識別するために，プロジェクトのパフォーマンスを定期的かつ一貫して監視し，測定することである．

プロジェクト作業において監視とコントロールは以下のように定義する．

（1）監視

　プロジェクトを通して実行されるプロジェクトマネジメントの一側面である．監視にはパフォーマンス情報の収集，測定，配布，およびプロセスの改善を推進するために行う測定結果とその傾向の評価が含まれるのである．この継続的な監視を行うことによって，プロジェクトマネジメント・チームはプロジェクトの健全度を洞察し，特に注意が必要な箇所を識別にできるのである．

（2）コントロール

是正処置や予防措置を行うことおよび再計画を立てること，さらに実施した処置によりパフォーマンスの課題が解決されたかどうかを判断するために行動計画をフォローアップすることが含まれるのである．

プロジェクト作業の監視・コントロール・プロセスで行うことは，以下に記述する．

・プロジェクトマネジメント計画書とプロジェクトの実施との比較

・是正処置や予防処置が必要かどうかを決定するためのパフォーマンスの評価，および必

　要に応じた処置の提言を行う．

・新たなリスクの特定および既存のプロジェクト・リスク分析，追跡，監視．そのために

は，リスクを特定し，状況を報告し，適切なリスク対応計画を実行することを確実に行

う．

・プロジェクトを通して，プロジェクトの成果物と関連文書について，正確でタイムリー

な情報データベースの維持を行う．

・状況報告，進捗測定，予測等の基礎となる情報の提供を行う．

・承認済み変更が発生した際に，それらの実施状況の監視を行う．

この継続的な監視により，プロジェクト・チームは，プロジェクトの状況を把握し，引き続き注意が必要な領域を特定する．監視・コントロール・プロセス郡は，あるプロセス郡内で実施される作業の監視・コントロールだけでなく，プロジェクト全体の作業の監視とコントロールも行う．複数のフェーズをもつプロジェクトにおいては，監視・コントロール・プロセス郡は，プロジェクトマネジメント計画書に沿ってプロジェクトに対し是正処置や予防処置を講じるように，プロジェクト・フェーズ間の調整をする．このレビューの結果，プロジェクトマネジメント計画書の更新が提言され，承認される場合がある．例えば，あるアクティビティの終了日を守れないことが明らかな場合は，現在の要員計画の調整，残業時間の増加，予算とスケジュール目標間のトレードオフなどが必要になる．

このプロセスのインプットは，「プロジェクトマネジメント計画書」，「実績報告書」，「組織体の環境要因」，「組織のプロセス資産」である．ツールと技法は，「専門家の判断」である．アウトプットは「変更要求」，「プロジェクトマネジメント計画書更新版」，「プロジェクト文書更新版」であり，以下で各項目を記述する．

### 5.2.1　インプット

・プロジェクトマネジメント計画書

　プロジェクトマネジメント計画書には，計画プロセスで作成したすべての補助のマネジメント計画書およびベースラインを統合し，集約する．プロジェクトマネジメント計画書には以下のものが含まれるが，これらに限定されるものではない．

・プロジェクトに対応して選択したライフサイクル，および各フェーズに適用するプロセス

・プロジェクトマネジメント・チームがテーラリングした結果，得られた次のような項目

　・プロジェクトマネジメント・チームが選択したプロジェクトマネジメント・プロセス

　・選定した各プロセスの実行のレベル

　・プロセスを実行するために使用するツールと技法の記述

　・当該プロジェクトをマネジメントするために選定したプロセスを適用する方法．これ

は選定したプロセス間における依存関係と相互作用，およびもっとも重要なインプットとアウトプットも含まれる．

・プロジェクト目標を達成するための作業の実行方法．

・変更を監視し，コントロールする方法を文書化した変更マネジメント計画書

・コンフィギュレーション・マネジメントの実施方法を文書化しコンフィギュレーション・

マネジメント計画書

・パフォーマンス測定ベースラインの一貫性を維持する方法

・ステークホルダー間のコミュニケーションのためのニーズと技法

・未解決の課題や未決定事項の処理を促進するために，内容，範囲，時期などをレビューする重要なマネジメント・レビュー．

プロジェクトメネジメント計画書の記述レベルは概要・詳細いずれの場合もある．また，

いくつかの補助計画書で構成することもできる．それぞれの補助計画書は，当該プロジェ

クトが必要とする程度まで詳しく記述する．いったんプロジェクトマネジメント計画書が

ベースラインとして確立されたあとは，変更要求が提起され，統合変更管理を通して承認

された場合に限りプロジェクトマネジメント計画書を変更することができる．

プロジェクト・ベースラインには以下のものがあるが，これらに限定されるものではない．

・スケジュール・ベースライン

・コスト・パフォーマンス・ベースライン

・スコープ・ベースライン

補助計画書には以下のものがあるが，これらに限定されるものではない．

・スコープ・マネジメント計画書

・要求事項マネジメント計画書

・スケジュール・マネジメント計画書

・コスト・マネジメント計画書

・品質マネジメント計画書

・プロセス改善計画書

・人的資源計画書

・コミュニケーション・マネジメント計画書

・リスク・マネジメント計画書

・調達マネジメント計画書

　スコープ・ベースライン，スケジュール・ベースライン，およびコスト・ベースライン

がまとまってパフォーマンス測定ベースラインとなり，統合したパフォーマンスを測定す

るための統括的プロジェクト・ベースラインとして使用する．パフォーマンス測定ベース

ラインは，アーンド・バリューの測定に使用される．

・実績報告書

　プロジェクト・チームは，アクティビティ，業績，マイルストーン，特定した課題，問題などについて詳しく記述した報告書を作成する．重要な情報を報告するために用いる実績報告書には以下のものがあるが，これらに限定されるものではない．

・現状の状況

・期間中の重要な実績

・予定されたアクティビティ

・予測

・課題

・組織体の環境要因

　プロジェクト作業の監視・コントロール・プロセスに影響を与える組織体の環境要因には以下のものがあるが，これらに限定されるものではない．

・国家標準や業界標準など

・企業の作業認可システム

・ステークホルダーのリスク許容度

・プロジェクトマネジメント情報システム（例：スケジュール・ソフトのような自動化ルール一式，コンフィギュレーション・マネジメント・システム，情報収集・配布システム，他のオンライン自動化システムとWebインターフェース）

・組織のプロセス資産

　プロジェクト作業の監視・コントロールのプロセスに影響を与える組織のプロセス資産には以下のものがあるが，これらに限定されるものではない．

・組織のコミュニケーション要求事項

・財務管理手順（例：作業時間報告，勘定コード，必要な支出や支払いのレビュー，標準契約条件）

・課題と欠陥のマネジメントの手順

・リスク・コントロールの手順．これには，リスク区分，発生確率の定義と影響度，発生確率・影響度マトリックスなどが含まれる．

・プロセス測定データベース．これは，プロセスやプロダクトの測定データを利用可能にするために使用される．

・教訓データベース

### 5.2.2　ツールと技法

専門家の判断

　プロジェクトマネジメント・チームは，プロジェクト作業の監視・コントロール・プロセスから得られる情報を評価するために，専門家の判断を利用する．プロジェクトマネジャーは，プロジェクト・チームと協力し，プロジェクトに対する期待とプロジェクトのパフォーマンスを一致されるように，必要な処置を決定する．

### 5.2.3　アウトプット

・変更要求

　計画値と実績値を比較した結果，変更要求が発行される．それにより，プロジェクトや成果物のスコープは拡大され，調整され，あるいは縮小される．変更により，プロジェクトマネジメント計画書，プロジェクト文書，またはプロダクトの要素成果物が影響を受ける．変更には，以下の種別のものがあるが，これらには限定されるものではない．

・是正処置

　予期される将来のパフォーマンスをプロジェクトマネジメント計画書に沿ったものとするために，プロジェクト作業に出される文書による指示．

・予防処置

　プロジェクト・リスクによって好ましくない結果が発生する確率を低減するために取られる行動の文書による指示．

・欠陥修正

　プロジェクト構成要素の欠陥の公式な文書による特定．それには欠陥の修正あるいは構成要素の全面的な取り換えのいずれかの勧告を含める．

プロジェクトマネジメント計画書更新版

　プロジェクトマネジメント計画書を構成するもので，このプロセスにおいて更新されるものには以下のものがあるが，これらに限定されるものではない．

・スケジュール・マネジメント計画書

・コスト・マネジメント計画書

・品質マネジメント計画書

・スコープ・ベースライン

・スケジュール・ベースライン

・コスト・パフォーマンス・ベースライン

・プロジェクト文書更新版

更新されるプロジェクト文書には以下のものがあるが，これらに限定されるものではない．

・予測

・実績報告書

・課題ログ

## 5.3　参考文献

[1] Project Managemant Institute, PMBOK Guide, 第4版, PMI, ,2008, p39, p59-61, p81-82.

[2] プロジェクト作業の監視コントロール, PMへの道, http://sayaji.seesaa.net/article/96501685.html, 2013-09-15.