コミュニティ型チーム活動の進化 ～ COVID-19 対策サイト開発から学ぶ ～

松尾谷 徹
有限会社 デバッグ工学研究所
matsuodani@debugeng.com

増田 礼子
フェリカネットワークス株式会社
Ayako.Masuda@FeliCaNetworks.co.jp

要旨

この *Future Presentation (FP)* は，*IT* 技術者の働き方やモチベーションについての提案と，その根拠となった事例の説明です．事例は，従来の働き方とは異なる形態で驚異的な成果を示した「*COVID-19* 対策サイト開発」です．従来の働き方とは，企業や職場に閉じたクローズド型です．異なる形態とは，開発を受注した企業が *Open Souce Software (OSS)* として *GitHub* のオープン・コミュニティを活用したことです．ここでは，クローズド開発とオープン・コミュニティ開発を比較しました．

成果の大きな差は，開発や修正のスピードに現れており，二桁ほどの差がありました．一方，生産性やコストについては，オープン・コミュニティが参加者に対価を支払うことがないので，計測すらありません．職場などのリアルな環境での活動ではないこともあり，階層型の管理や予実管理は存在せず，自律管理で運用されていました．一番の差は，働く技術者のモチベーションです，クロースド型の職場は，ロイヤリティやチーム結束力がモチベーションです．オープン・コミュニティでは，存在承認に近いものがモチベーションになっていると推測されます． そのモチベーション源は雇用者や仲間からではなく，外部から与えられていることに注目して，この提案を行います．

1. はじめに

働き方改革は「労働時間を短縮して生産性を上げる」ための知恵を求めています．そのケーススタディとして「東京都 新型コロナウィルス感染症対策サイト開発」[1,2] (以降,「COVID-19 対策サイト開発」と呼ぶ) を取り上げて，技術者の観点から提案をまとめます．

事例分析の結果では，開発や修正のスピードにおいて，二桁ほどの差がありました．この差を生む仕組みや原因について分析を行いました．その背景に，管理監督によって技術者の労働を制御する方式と，技術者の自律性から労働する方式の差があります．ここでの主張は，自律的な行動の源となるモチベーションに関する分析がメインテーマです．この提案は次に示す節構成です．

2. 事例の背景： Open Souce Software (OSS) とオープン・コミュニティ開発
3. 事例「COVID-19 対策サイト開発」の概要
4. 分析結果：桁違いの顕著な特性
5. オープン・コミュニティの原動力
6. 提案のまとめ

2. OSS とオープン・コミュニティ開発

ソースコードの共有は，コンピュータの利用が始まって以来，学術機関や研究機関の間で行われています．1970 年代に入り，ソフトウェアを商用に利用するため，ソフトウェアの配布に制約を付与するプロプライエタリ・ソフトウェアと，ソースコードを非公開とするクローズドソースの商習慣が出来上がりました．国内の大手ソフトウェア企業は，現在でもこのビジネスモデルが中心であり，プロダクトの生産性や品質を高める努力を 40 年に渡り続けています [3].

一方，OSS は，商用とは別に発展を続け，1982 年には GNU プロジェクトが始まり，ボランティアによるコミュニティだけではなく，非営利団体として技術者を雇用し開発を拡大しました [4]．1991 年には，Unix カーネルと結合してシステム一式が完成し，PC などへの提供が行われました [5]．さらなる大きな進展はリーナス・トーバルズによる「Linux」開発です．それまでの OSS 開発もコミュニティで行われていましたが，リーナスはオープン・コミュニティで開発を始めました [6]．この仕組みにより，21 世紀に入ってから，OSS は爆発的な発展を始めます．この頃から，OSS 開発には，ホビイストやハッカーだけでなく，団体や企業からも技術者が参加するようになりました．日本では，現在でも OSS をホビイストやハッカーによる趣味の作品と思い込んでる人も少なくありませんが，実態は IT 系サービス・ビジネスの中核になっています．

リーマンショック後，プロプライエタリ・ビジネスは急激に衰退し，サービス系のビジネスが躍進しました．システムの複雑化から依存関係が増大し，OSS を含まないシステム提供ができなくなました．2010 年頃から，欧米の多くの企業がプロダクト型ビジネスをやめ，サービス型ビジネスへとシフトしました．その過程で，社内のローズド開発を OSS 開発へ移行しました．その受け口として，GitHub [7] の Organization (以降，Org と略す) が使われ，OSS のビジネス利用が始まりました [8]．GitHub には，日本の人口を超えるリポジトリ (プロジェクトに相当) が登録されていますが，Star の数などによる評価が高いものは，企業や団体が Org を運用するリポジトリであり，有償のサービスを併設している場合もあります．

今回のケーススタディも，商用サービスを提供する企業が開発に OSS を活用したものです．その特徴は，企業に閉じた開発チームではなく，オープン・コミュニティとして開発チームを運用した点です．そこから生まれたクローズド開発とは違った大きな効果に着目して，この提案を行います．

ここで用いたデータは，GitHub 社が提供する REST API v3 を使いました [9]．API インターフェースで得られないファイル別の修正などの詳細な記録は Git の log から分析しました [10]．いずれも公開されているデータです．

3. 「COVID-19 対策サイト開発」の概要

この提案の事例として，東京都が開設した新型コロナウイルス感染症 (COVID-19) に関する最新情報を提供するサイトの開発をとりあげました．東京都がある企業に情報サイト開発を発注し，その企業が GitHub のリポジトリを使って OSS 開発を行いました．このサイトの機能は，さまざまな情報をネット経由で提供する表示型のサイトです．処理の記述は，メイン言語として JavaScript フレームワークである Vue.js が使われています [11]．他にも，TypeScript，JavaScript，Python が使われ，データは，json，md (MarkDown)，yml で記述されています．

時系列で開発を観ると，2020 年 2 月 29 日に GitHub に登録され，ここから開発が始まり，3 月 5 日に初版がリリースされ，サイトが公開されました．その後，オープン・コミュニティの特性から，多くの支援者が参加し，その数は 250 名を超えています．並行して，札幌，名古屋，福岡，大阪，長野，浜松，など 40 を超える都市で流用され，それぞれの情報サイトが開設されました．

開発記録から直接確認できる特徴として，次のものが挙げられます．

- 開発の速さ
- 新たに参加する技術者 (貢献者) の立上りの速さ
- 問題処理 (Issue) の回答率と対応の速さ
- 他サイトへの流用容易性と流用の速さ

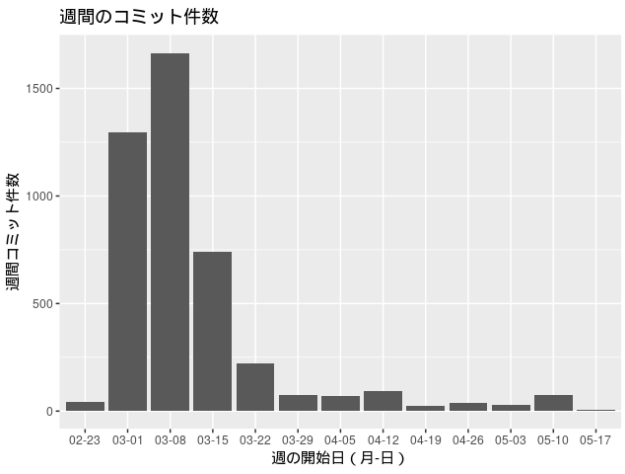

図 1: 週単位のコミット件数

開発の速さは，期間あたりのリポジトリへのコミット数から推測することができます．図 1 は，週単位で区切った期間でコミット件数の変化を表したものです．コミット件数は，3 月の第 1 週は 1,300 件，第 2 週には 1,700 件に達しています．クローズド開発における最速のアジャイル方式でも，マスタ・リポジトリに対して 1 週間あたり二桁のコミットを行うのは非常に稀です．事例の開発スピードは四桁なので，少なくとも二桁以上の差があると思われます．

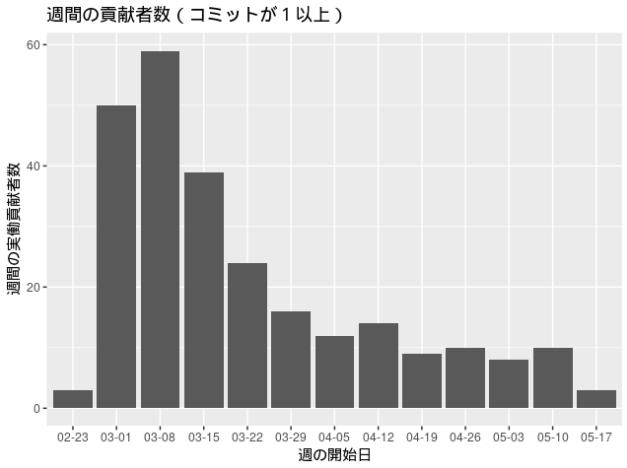

図 2: 週単位の実働貢献者数

初めて参加した技術者 (GitHub では Contributer：貢献者と呼ぶ) の立上りの速さとは，たとえばクローズド開発において，何らかの理由から急遽，工数投入が必要となった時に，技術者を確保して投入し，投入による成果が現れるまでの期間に相当します．図 2 は，週単位で区切った期間における技術者の人数を表しています．3 月の第 1 週で 50 名を超え，第 2 週では 60 名に達しています．図 1 は，開発の成果を表すもので，参加と同時に成果を出しています．週単位で技術者の増減があっても，引き継ぎなど成果に直結しない間接的な活動で開発が停滞する現象は見られません．技術者の人的資源調達や，投入時の立上りなど，クローズド開発と比べ，そのスピードは桁のオーダで差が見られます．この背景に，どのような管理システム，あるいはソフトウェア構造があるのかなど，興味深いテーマです．

GitHub における問題処理は Issue と呼ばれるシステムで処理されます，Issue には，バグの処理だけでなく，新規開発機能の担当募集や，仕様変更などさまざまな案件を対象にします．Issue の状態は，open と closed の 2 つです．クローズド開発のように，管理者が案件の担当者を決め，作業指示を行うシステムはありません．Issue が open になると，対応できる技術者が自律的に対策を作り，プルリクエスト (対策の確認依頼) を返します．

2 つの点で，クローズド開発とのスピード差が見られます．一つは，バグの発見が早いこと，もう一つは修正が速いことです．クローズド開発では，バグを見つけ出すためのレビューやテストに大きなコストを投入します．統合試験工程で発見されるバグもあり，後戻りコストが課題になっています．一方，この事例におけるバグ検出は，技術者が担っています．後述する GitHub workflow により，1 行の修正であっても，関連する機能について最新ビルドを用いてシステムレベルで動作確認を行います．その結果，多くの眼で，最新のシステム挙動を確認することになり，バグや GUI の不整合を早く発見することができています．

記録から，5 月 24 日現在のデータを観ると，3,814 件の Issue が提案されいて，当日発生分を含め closed は，3,698 件であり，対応率は 96.9 % です．一般的に，バグ票や問題処理票の回答時間分布は，正規分布とは異なっており，平均や分散で特性を示すことができませんが，四分位数で表すことで評価することができます．この事例では，Issue の 50 % の回答時間は 0.5 時間 (30 分) 以内で，75 % は 18 時間以内でした．

GitHub における他サイトへの流用は，Fork です．Fork は，GitHub に登録した人が，自分のリポジトリに Clone (Git のチェックアウト機能) する操作なので，どのように流用したのかを確認するには，Fork 先を調べる必要があります．この事例の Fork 件数は 1,900 件と増え続けており，その中で Fork 後に何らかの修正を行ったリポジトリは 70 件ありました．明示的に，都市名を示して Fork しているものもあり，40 件ほどは，流用が行われていることを確認しました．

4. 桁違いの特性原因を探る

ここでは，従来型の開発文化，すなわちクローズド開発の観点から，事例で観測された特性について考察します．

まず，ここでのクローズド開発について簡単に定義すると，プロジェクト開発のように有期限でリソース制約 (予算や工数) が存在し，目的を持った開発活動を想定しています．目的や制約は，顧客との契約が強く影響し，

変更するには経営的な稟議が必要です．開発体制は，雇用や契約により技術者を確保し，階層的な職位によって活動の指示と進捗管理を行います．技術者の役割は，たとえばサブシステム開発や品質保証やテストなどのように，機能的に分割されています．

事例とした COVID-19 対策サイト開発は，クローズド開発とは全く異なる異文化での開発のように見えます．その違いは，管理監督の役割や規範がない，テストフェーズや第三者テストがないなど真逆に近いものです．しかし，東京都から受注した企業は，顧客との間で何らかの契約や開発計画があったはずです．実際に行われた開発は，GitHub によるオープン・コミュニティ開発であり，OSS 開発であり，GitHub workflow でした． その結果，企業側が雇用した技術者とは別に，支援者として多くの技術者がオープン・コミュニティに加わり，開発を加速したのです．雇用側の技術者も，オープン・コミュニティ型の開発を取り入れ，開発スピードを加速しました．

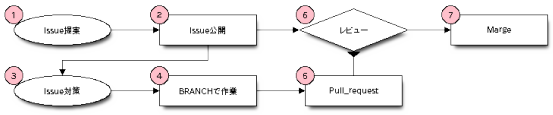

図 3: GitHub workflow の概要

オープン・コミュニティ型の開発とは，具体的には，GitHub workflow (または GitHub flow) のことです．このワークフローは，多人数で同時開発や保守を実行する時，混乱なく速く行うためのものです [12]．ワークフローの概略を図 3 に示し，それぞれの work について概略を示します．

1. Issue 提案
   議論が必要な機能要求やバグ連絡など課題を提起する

2. Issue Open (公開)
   課題が公開され，意見 (サポート) を待つ

3. Issue 対策
   対策提案者 (Open Issue に対する対策や回答を行う貢献者) が対策を提案する

4. BRANCH で作業
   対策を Topic ブランチで開発し，テストは Master から対策用のブランチを作成して評価する

5. Pull request
   Issue に対する解が得られれば，対策内容のレビュー要求を出す
   この時点で最新の統合状態で動作テストを実行する

6. レビュー
   関係者がレビューする
   関係者はそれぞれの状況で異なる

7. Marge (Issue Closed)
   レビュー後，問題がなければ Master にマージし，Issue を Closed とする

この GitHub workflow は，活動規範として共有されているため，見ず知らずの多くの技術者がオープン・コミュニティに参加しても，競合を起こすことなく，独立して開発を進めることができています．ソフトウェアの構造も，この規範に則って極力独立に作られ，相互に影響することを少なくしています．

GitHub workflow と，クローズド開発でも用いられている Git flow の差に注目する必要があります．ライブラリの入れ物としては同じ Git ですが，作業フローは異なります．GitHub workflow では，マスタに本番リポジトリを用います．Git flow では，本番用とは別に開発用のリポジトリや保守用のブランチを用います．GitHub workflow における Issue に対する修正活動は，修正を行う直前に本番リポジトリから Clone し，修正を確かめてプルリクエストを出すことで，対策活動中に本番リポジトリが他の修正対応等のマージが生じないよう，短い期間で修正する方式です． これによりマージされていない修正間の相互影響を減らすことができます．

5. オープン・コミュニティの原動力

事例で紹介した COVID-19 対策サイト開発は，オープン・コミュニティによる開発が特徴です．OSS は，もともとコミュニティ活動 (目的や理念を共有する社会集団の活動) です．OSS のコミュニティは，決してクローズドではないのですが，GNU など高度なコミュニティへの参加には障壁がありました．GitHub のオープン・コミュニティは，参加障壁を低くしたことが大きな特徴と考えられます．その結果，非常に多くの人が開発に参

加しています．GitHub 社も，ドキュメントの誤字脱字の修正から貢献することを推奨しています．

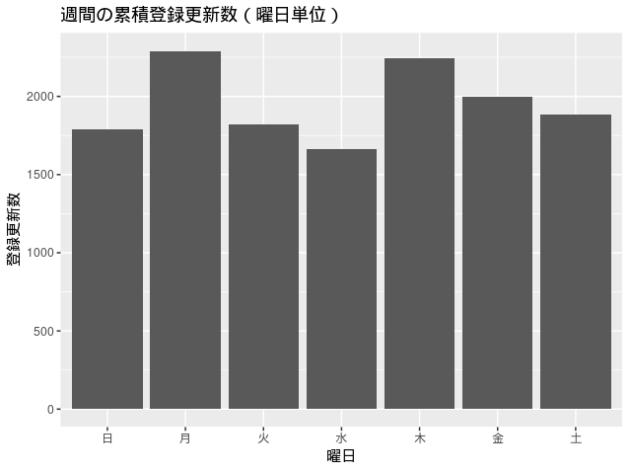

図 4: 曜日ごとの累積コミット分布

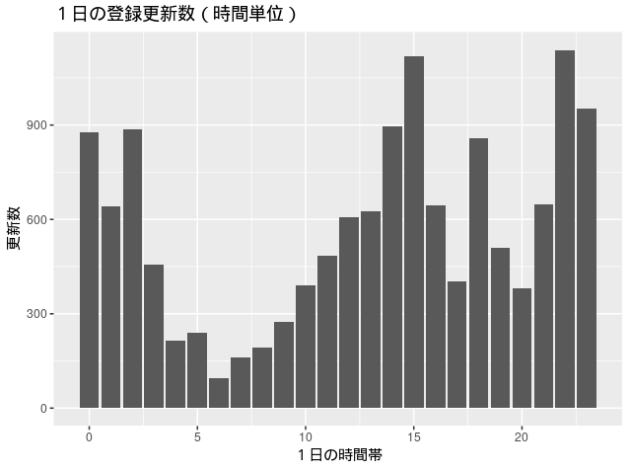

図 5: 24 時間の累積コミット分布

クローズド開発においては，契約や雇用など，ロイヤリティとの交換取引として，量的あるいは質的な貢献が求められます．厄介なことに，貢献の評価者や評価の価値観が主観的で技術的な価値とは異なる場合もあります．同じ企業内でも，部門によってその価値観は異なり，技術的な貢献が評価されないこともあり，技術は停滞する傾向にあります．

どちらが技術者にとって働きやすく，どちらが成果を出せるのでしょうか．机上で議論するとさまざまな意見が出てくるでしょう．ここでは，データから，オープン・コミュニティの働き方を見てみます．図 4 は，一週間のどの曜日で，どのくらい働いたかをコミット数の分布で示したグラフです．図 5 は，1 日 (24 時間) の何時にリポジトリを更新したかを表しています．図 4 と図 5 は，ある個人の働き方ではなく，全メンバの累計から求めたものです．クローズド開発 (労働指示の場合) であれば，かなりひどい労働基準法違反ですが，オープン・コミュニティでは，空いた時間で貢献する強いモチベーションによる自律的な行動として読み取ることができます．この指示型ではあり得ない働き方の特徴は，今回の事例だけでなく，地域を問わず多くの OSS 開発 / オープン・コミュニティで観測されています．

では，何がモチベーションとして働き，このような自律的な活動を生み出しているのでしょうか．従来のコミュニティ活動であれば，コミュニティとしての集団 = チームとしての団結から生じていると思われていました．しかしこの事例でも，そのようなメンバ間でのコミュニケーションや結束は見られません．目的共有は考えられますが，仲間としての団結 = 同一化については確認できません．

活動が活発な人に見られる特徴として，フォロワーの多さが挙げられます．GitHub に限らず，ネット上で個人が活動するさまざまなサイトでは，フォロワーの仕組みを取り入れています．

「フォロワーが多いと，モチベーションが維持され，広く活躍する」と考えられます．フォロワーの数と活動の関係は，GitHub の場合，一つのリポジトリ (プロジェクト) に閉じて論じることができません．オープン・コミュニティとは，包括プロジェクトのような垣根を超えた活動です．図 6 は，フォロワーの数と，その技術者が主催するリポジトリの数を示した散布図です．関係があることを回帰線で示しています．

フォロワー数に対数を用いたのは，その分布が対数正規分布であるためで，回帰分析を有効に使う分析上の理由です．リポジトリとは，今回の事例「COVID-19 対策サイト開発」のような開発プロジェクトで，技術者個人でも開設することができます．活発な技術者は，多くのリポジトリを自ら開設し，かつ，他のリポジトリも援助しています．

ここまで，事例「COVID-19 対策サイト開発」の活動だけを示して，驚異的と表現しましたが，支援した技術者の多くが，図 6 に示すように，自身でもリポジトリを主催して活動しています．また，ここでは分析していませんが，事例以外のリポジトリへの支援もあります．

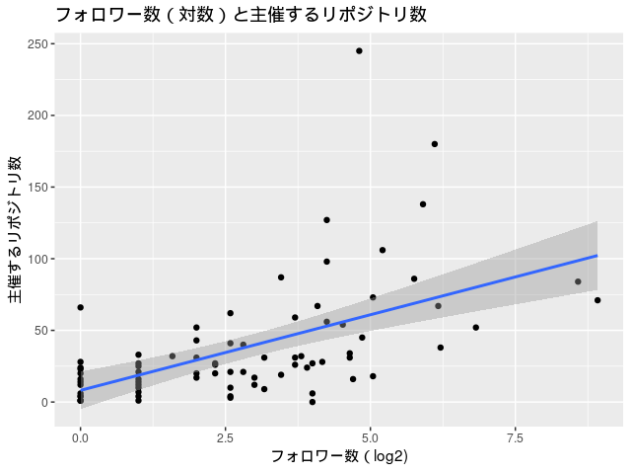

図 6: 技術者のフォロワー数 (対数) と保有リポジトリ数

6. 提案のまとめ

事例として取り上げた「COVID-19 対策サイト開発」は，クローズド開発と比べ，二桁ほど速い開発を実現していました．しかも，前章の最後に示しように，その活躍は一つのリポジトリに閉じていないのです．歩く速さでたとえると，少なく見積もっても，時速 4 km と，時速 400 km 程度の差です．差が大きすぎて比較対象外と考え，多くの人は，ひとまず先送りを選ぶかもしれません．

ここでの提案は，OSS 開発やオープン・コミュニティへの改宗を推奨するものではありません．技術者にとって，働きやすいとはどういうことでしょうか？なぜ，寝食を忘れて没頭するのでしょうか？これらについて，事例がヒントを与えていると考え，事例を紹介し，その推進力の一部を調べ，さらに調べ学ぶ価値があることを提案します．

7. おわりに

働き方改革「労働時間を短縮して生産性を上げる」という課題は，意義は解るけれども誰も達成できなかった課題です．しかし，「COVID-19 対策サイト開発」の事例を観ると想定を上回る成果を出しています．この事例に対しては，自分たちとは事情が異なるのでできないとする「言い訳論」と，過剰に成果を称賛する「称賛」の 2 極の論戦が考えられますが，ここで提案するのは，技術者の社会行動としてモチベーションの影響です．現段階では証明できていませんが，おそらく「社会的な承認欲求」だと考えています．

クローズド開発では，さまざまなストレスに耐えるモチベーション源として，仲間，チームが有効であり，我々は，「職場 = チーム」と考えチームの活性化について研究を進めてきました．数年前から OSS 開発における状況の調査を始めて，全く異なる働き方に遭遇しました．ここで示した「COVID-19 対策サイト開発」の状況と同種の傾向が，多くの OSS 開発で観測されています [13]．この傾向は，10 年前の OSS 開発データでも現れています．働き方のターニングポイントが近い予感がしています．

参考文献

[1] 東京都 新型コロナウィルス感染症対策サイト, https://stopcovid19.metro.tokyo.lg.jp/ (accessed:2020/03/16)

[2] 東京都 新型コロナウイルス感染症対策サイト (GitHub) (Tokyo COVID-19 Task Force website), https://github.com/tokyo-metropolitan-gov/covid19 (accessed:2020/03/16)

[3] オープンソースソフトウェアの歴史 (Wikipedia), https://ja.wikipedia.org/w/index.php?title=オープンソースソフトウェアの歴史&oldid=72952775 (accessed:2020/05/28)

[4] GNU (Wikipedia), https://ja.wikipedia.org/w/index.php?title=GNU&oldid=76163596 (accessed:2020/05/28)

[5] UNIX (Wikipedia), https://ja.wikipedia.org/w/index.php?title=UNIX&oldid=77135146 (accessed:2020/05/28)

[6] Linux (Wikipedia), https://ja.wikipedia.org/w/index.php?title=Linux&oldid=77752398 (accessed:2020/05/28)

[7] GitHub, https://github.com/ (accessed:2020/03/16)

[8] Organization の作成 - GitHub ヘルプ, https://help.github.com/ja/enterprise/2.19/admin/user-management/creating-organizations (accessed:2020/05/28)

[9] RES API v3, https://developer.github.com/v3/ (accessed:2020/03/16)

[10] Git - git-log Documentation, https://git-scm.com/docs/git-log (accessed:2020/03/16)

[11] Vue.js (Wikipedia), https://ja.wikipedia.org/w/index.php?title=Vue.js&oldid=75855212 (accessed:2020/03/16)

[12] ワークフローを設定する - GitHub ヘルプ, urlhttps://help.github.com/ja/actions/configuring-and-managing-workflows/configuring-a-workflow (accessed:2020/05/28)

[13] 増田礼子, 森本千佳子, 松尾谷徹, 津田和彦, 「大規模オープンソース・ソフトウェアプロジェクトにおける開発効率の計測」, 電気学会論文誌 C (電子・情報・システム部門誌), Vol.138, No.8, pp.1011–1019, 一般社団法人 電気学会, 2018