Vol.2017-SE-196 No.28
2017/7/21

GitHubにおける言語ごとのビルドファイルの更新理由調査

鐘ヶ江 由佳[1]　玉田 春昭[1]　畑 秀明[2]

概要：今日では，プロジェクトを効率良く進めるためにはビルドツールの導入が不可欠と言われている．問題の早期発見とフィードバックサイクルを短縮する CI (Continuous Integration) ツールにビルドツールが必要となるためである．ビルドツールとはコンパイルやテスト，ソースコードの解析等，ソフトウェア開発で行われるビルド作業を自動化するツールである．それらビルド作業はビルドファイルと呼ばれる定義ファイルに従って行われる．このように開発において，ビルドツールの利用はデファクトスタンダードになっていると言える．しかし，実際どのくらいのプロジェクトでビルドツールが使われているかや，どのようなビルドツールが使われているかの調査は十分ではない．また，ビルドツールが行う処理は定型的な処理であるため，ビルドファイルは頻繁に更新されるものではないと考えられる．そこで本稿では，ビルドファイルに焦点を絞り，多くのプロジェクトを対象にビルドツールの利用の有無，ビルドファイル更新の頻度や更新者を調査する．結果として，Java，Ruby，Python では，特定のビルドツールが使われていることがわかった．また，78.4%のプロジェクトでプロジェクトのコミット回数が一番多い開発者が，同様にビルドファイルへのコミットも一番多いことがわかった．ビルドファイルのコミット時期に着目すると，リリースの前の時期のみにコミットするケースがリリース全体のうち 33.9%であった．

キーワード：ビルドツール

1. はじめに

ビルドツールは，ソフトウェア開発における定型的な作業を自動化するものであり，多くのソフトウェア開発現場で利用されている．特に，開発環境とは異なる環境で，定期的かつ自動的にフルビルドを行うための環境である CI (Continuous Integration) の導入に欠かかせないものである．ここで言う定型的な作業とは，プロジェクト中のソースコードのコンパイルやテストの実施，そして，パッケージングといった作業のことである．これらの一連の作業をまとめて実施することをビルドと呼ぶ．そのビルドの手順は，ビルドファイルと呼ばれるファイルに特定のフォーマットに従って記述されている．

ビルドを手作業で行う場合，非常に手間がかかる．例えば，プロジェクト中のソースコードが大量にある場合，コンパイルコマンドにそれらのソースコードを全て渡す必要がある．ソースコードを一つでも漏らすとコンパイルの失敗や実行時エラーに繋がるため，漏れなく指定する必要がある．一方で，プロジェクトの進行に従って，一部のソースコードが編集されていく．その度に全てのソースコードをコンパイルすると，コンパイルに要する時間が次第に膨大な時間となる．そのため，編集したファイルのみをコンパイルすれば良いものの，やはり必要なファイルを漏れなく指定する必要がある．これらの問題を解決することを目的に make などのビルドツールが作成され，機能を増やしながら今日のビルドツールとなっている．

今日，多くのビルドツールが提供されている．最初期は make[1] のみであったものの，異なる環境の違いを吸収するために autotools[*1] が開発された．続いて，Apache Ant[*2]や Rake[*3]など，特定の言語の特徴に特化したものが提供されている．そして，近年のビルドツールは，これまでのビルドツールの機能を踏まえつつ，より簡便に，かつ多機能になっている．

make が提供され始めてから 30 年近く経っており，その間に新たなビルドツールも数多く提供されている．しかし，少数のプロジェクトに対して，ビルドツールに関する調査は行われているものの [2], [3]，世の中の多くのプロジェクトを対象にしたビルドツールに関する調査は行われていない．

[1] 京都産業大学,
Kyoto Sangyo University
[2] 奈良先端科学技術大学院大学,
Nara Institute of Science and Technology

[*1] http://autotoolset.sourceforge.net/
[*2] https://ant.apache.org
[*3] https://ruby.github.io/rake

具体的には，世の中に多くのビルドツールが存在するが，どのようなツールがどのようなプロジェクトに使われているかは調べられていない．また，ビルドファイルをどのような開発者が更新しているのか，いつ更新しているのかも調査されていない．

そこで，本稿では，ビルドファイルに注目した大規模な分析を行う．分析のゴールとして，以下の 3 つの RQ を挙げた．

RQ 1. どのようなビルドツールが使われているか．
RQ 2. 誰がビルドファイルを更新しているか．
RQ 3. いつビルドファイルが更新されるか．

これらの RQ に答えるため，既存のリポジトリサービスを利用している OSS 1,000 プロジェクトを対象にビルドファイルに関する分析を行った．

本稿の残りの構成は以下の通りである．続く第 2 節は，分析に先立ち必要な事項を説明する．第 3 節で，分析を行い，結果を述べる．そして，第 4 節で，本稿での分析の妥当性について述べ，第 5 節で関連研究との違いを説明する．最後に，第 6 節でこの論文をまとめ，今後の課題を記す．

2. 準備

2.1 リサーチクエスチョン

本稿では，ビルドツールについての以下の 3 つのリサーチクエスチョンについて答えることを目的とする．

RQ 1. どのようなビルドツールが使われているか．
RQ 2. 誰がビルドファイルを更新しているか．
RQ 3. いつビルドファイルが更新されるか．

RQ1 では，世の中には数多くのビルドツールが存在するので，どのビルドツールが実際に使われているのかを調査する．言語の特徴を踏まえたビルドツールが存在するため，言語ごとに傾向を調査する．

ビルドツールはビルドファイルに記されたルールに従って動作する．RQ2 では，ビルドファイルをどのような開発者が更新するかを調査する．一般に，ビルドファイルを更新する開発者は，ソースコードを更新する開発者に比べて少なく，プロジェクトの中の主要な開発者であろうと考えられる．これを確認するため，ビルドファイルの最頻コミット者が，プロジェクトにどのくらい貢献しているかを調査する．

最後の RQ3 では，ビルドファイルがいつ更新されるのか，プロジェクトのリリース日を元に調査を行う．

2.2 プロジェクトのコミット情報から得られるメトリクス

ここでは，プロジェクトのコミットから得られるメトリクスについて定義する．

2.2.1 プロジェクトのコミット情報

プロジェクト全体のコミットは，$C = \{c_1, c_2, ..., c_n\}$ とし，その数を $|C|$ とする．各コミットには，コミット日時，コミットした開発者，コミットコメント，コミットしたファイルの名前，差分情報が含まれている．

各コミットのファイル名は，$F(c_i) = \{f_{i,1}, f_{i,2}, ..., f_{i,m}\}(c_i \in C, 1 \leq i \leq n)$ で得られるものとする．ここで，各ビルドツールのデフォルトのビルドファイル名を $B = \{b_1, b_2, ..., b_l\}$ とする．ビルドファイルに関係したコミットを $C^{b_j} = \{c_i | c_i \in C \wedge b_j \in F(c_i)\}(1 \leq i \leq n, 1 \leq j \leq l, b_j \in B)$ とする（$|C| \geq |C^{b_j}|$）．

2.2.2 ある期間中のコミット回数

コミット日時 d_i を $d_i = \mathrm{date}(c_i)$ で得られるとする．ここで，ある期間 $d_s, d_e (d_s < d_e)$ 間に行われたコミットを $\mathrm{period}(d_s, d_e, C) = \{c_i | c_i \in C \wedge d_s \leq \mathrm{date}(c_i) < d_e\}$ とする．

2.2.3 プロジェクトの開発者情報

各コミットの開発者は $u = \mathrm{user}(c)$ とし，プロジェクト全体の開発者集合 U は，C に属する c_i からユーザ u_i を取り出し，得られた集合から重複を取り除いたものとする．これを $U(C) = \{u_1, u_2, ..., u_k\}$ で表す．各ビルドファイルの開発者集合は，$U(C^{b_j})$ で表せる．どのビルドファイルか区別しない場合，全てのビルドファイルの開発者集合は，$U(C^b)$ で表せる．

2.2.4 プロジェクトの差分情報

各コミットにおけるファイル $f_{i,k}$ の差分情報 $\mathrm{diff}_a(c_i, f_{i,k})$ は，コミット c_i の時に，前のコミット c_{i-1} から $f_{i,k}$ に追加した行数を表すものとする．同様に，$\mathrm{diff}_d(c_i, f_{i,k})$ は，$f_{i,k}$ の c_{i-1} から c_i に更新される時の削除行数を表す．この時，$\mathrm{churn}_a(c_i) = \sum_{k=1}^{k \leq |F(c_i)|} \mathrm{diff}_a(c_i, f_{i,k})$ をコミット c_i の追加行数，$\mathrm{churn}_d(c_i) = \sum_{k=1}^{k \leq |F(c_i)|} \mathrm{diff}_d(c_i, f_{i,k})$ をコミット c_i の削除行数，$\mathrm{churn}(c_i) = \mathrm{churn}_a(c_i) + \mathrm{churn}_d(c_i)$ をコミット c_i の追加削除行数と呼ぶ．

2.2.5 プロジェクトのコミット回数，追加削除行数

また，各ユーザのコミット回数を $\mathrm{commits}(u, C) = |\{c_i | u = \mathrm{user}(c_i) \wedge c_i \in C\}|$ とする．$u \notin U(C)$ の場合は，$\mathrm{commits}(u, C) = 0$ である．同様に，各ユーザの追加削除行数を $\mathrm{churns}(u, C) = \sum\{\mathrm{churn}(c_i) | u = \mathrm{user}(c_i) \wedge c_i \in C\}$ とする．

この時，ユーザをコミット回数で降順に並び替えた時のユーザ u の順位を $\mathrm{rank}_c(u, C)$ で表す．また，追加削除行数でユーザを降順に並び替えた時のユーザ u の順位を $\mathrm{rank}_l(u, C)$ で表す．

2.3 分析対象

本稿では，分析の対象を GitHub でホスティングされているプロジェクトとする．10 言語を対象に，人気順（Star 順）で並べた時の上位 1,000 プロジェクトを対象とした．

表 1　ビルドツールとビルドファイル名の対応表

ビルドツール	ビルドファイル名
Ant	`build.xml`
Bazel	`BUILD`
Cake	`Cakefile`, `build.cake`
Distutils	`setup.py`
Gradle	`build.gradle`
Grunt	`Gruntfile.js`
Gulp	`gulpfile.babel.js`, `gulpfile.js`
Make	`Makefile`
Maven	`pom.xml`
Rake	`Rakefile`
SBT	`build.sbt`
Setuptools	`ez_setup.py`
Webpack	`webpack.config.js`

分析時点での対象言語は，Java, JavaScript, C++, C#, C 言語, CSS, HTML, Ruby, PHP, Python である．プロジェクトの言語とは，そのプロジェクトで主に利用されている言語のことを指している．そのため，主要言語の他にいくつかの言語を利用しているプロジェクトも存在する．

これらの 1,000 プロジェクト× 10 言語の約 10,000 プロジェクトを対象に分析を行う．ただし，GitHub からのクローンに失敗するなど，必ずしも全てのプロジェクトが取得できているわけではない．

また，分析を行うビルドツールは Ant, Bazel, Cake, Distutils, Gradle, Grunt, Gulp, Make, Maven, Rake, SBT, Setuptools, Weppack とし，表 1 に示すビルドファイルが利用されているものとした．つまり，表 1 のビルドファイル名の列が B となる．

3. 分析結果

3.1 RQ1. どのようなビルドツールが使われているか

ここでは，RQ1. 言語ごとに各対象プロジェクトがどのようなビルドツールを利用しているかを調査する．各プロジェクトのディレクトリに表 1 に示すビルドファイルが存在していれば，当該ビルドツールを利用しているものとした．なお，1 つのプロジェクトで複数のビルドツールを利用している場合，特定の一つに限定せず，全てのビルドツールを利用しているものとした．

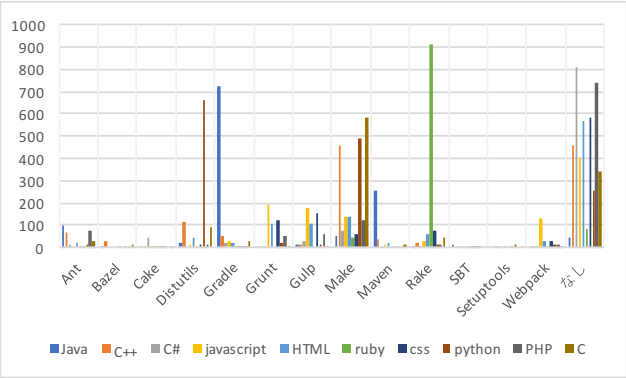

図 1　言語ごとのビルドツールの使用傾向

調査結果を図 1 に示す．横軸はビルドツール，縦軸はその頻度を言語ごとの棒グラフで示している．

図 1 を見ると，Java，Pyhon, Ruby がそれぞれ，言語に特化したビルドツールである Gradle，Distutils, Rake の使用頻度が非常に高いことがわかる．JavaScript では，Grunt, Gulp, Make の使用傾向が 14%〜19%程度で，ビルドツールを利用していないプロジェクトが全体の 40.3%であった．また，C++と C ではそれぞれ，46.2%，58.3%のプロジェクトが Make を利用していた．一方，C++，C#，HTML，CSS，PHP では半数近くのプロジェクトでビルドツールを利用していない．特に C#は 81.7%のプロジェクトがビルドツールを利用していない．ただし，C#は Visual Studio を利用してビルドを行っているプロジェクトが多いことが README やプロジェクトのトップディレクトリに存在するビルドスクリプトから伺える．ただし，このビルドスクリプトは，バッチファイルで書かれており，特定のビルドツールを利用しているわけではないため今回の調査結果からは除外している．

すなわち Java，Ruby，Python の多くのプロジェクトでは，それぞれ Gradle，Rake，Distutils を利用している．一方で，IDE を使うことが前提となっていて，ビルドファイルを利用していない C#のような言語もある．

3.2 RQ2. 誰がビルドファイルを更新しているか

3.2.1 ビルドファイル最頻更新者の調査

Java 言語で書かれた 955 プロジェクトについて，各プロジェクトの開発者のコミット回数を調査した．$\mathrm{rank}_c(u, C) = 1$ となるユーザを u_{ca}（コミット回数が一番多いユーザ），$\mathrm{rank}_c(u, C^{b_i}) = 1$ となるユーザを u_{cb}（ビルドファイルのコミット回数が一番多いユーザ）とした時，u_{ca} と u_{cb} が一致するかを調査した．同様に，$\mathrm{rank}_l(u, C) = 1$ となるユーザを u_{la}（追加削除行数が一番多いユーザ），$\mathrm{rank}_l(u, C^{b_i}) = 1$ となるユーザを u_{lb}（ビルドファイルへの追加削除行数が一番多いユーザ）とした時，u_{la} と u_{lb} が一致するかを調査した．開発者は，コミットに付随する

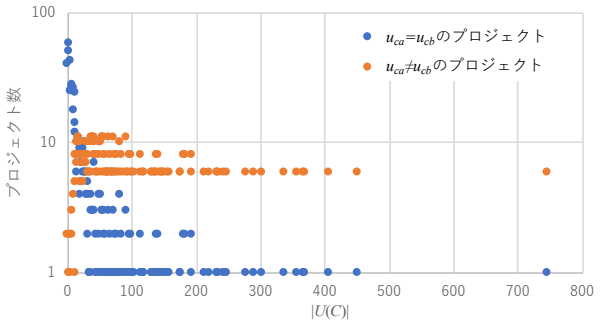

図 2　$|U(C)|$ ごとのプロジェクトの数

表 2　$u_{ca} \neq u_{cb}$ となるプロジェクトの調査

		平均	最大値
All	$\|U(C)\|$	78.68	2,161
	$\mathrm{rank}_c(u_{cb}, C)$	2.46	25
	$\mathrm{rank}_c(u_{cb}, C)/\|U(C)\|$	0.11	0.67
$\|U(C)\| > 5$	$\|U(C)\|$	87.24	2,161
	$\mathrm{rank}_c(u_{cb}, C)$	2.59	25
	$\mathrm{rank}_c(u_{cb}, C)/\|U(C)\|$	0.88	0.57
$\|U(C)\| > 10$	$\|U(C)\|$	101.16	2,161
	$\mathrm{rank}_c(u_{cb}, C)$	2.75	25
	$\mathrm{rank}_c(u_{cb}, C)/\|U(C)\|$	0.66	0.46
$\|U(C)\| > 100$	$\|U(C)\|$	241.52	2,161
	$\mathrm{rank}_c(u_{cb}, C)$	3.65	25
	$\mathrm{rank}_c(u_{cb}, C)/\|U(C)\|$	0.18	0.46

author のメールアドレスで区別した[*4].

ここで，b_i は，最終コミットにおいて，トップディレクトリにあるビルドファイルとする．トップディレクトリに複数のビルドファイルが存在する場合は，使用割合が多いビルドツールを元に利用ビルドツールを判定した．つまり，Gradle，Maven，Ant，Make の順である．

ビルドファイルのコミットがあったプロジェクトの数は 868 であり，そのうち 78.47%のプロジェクトが $u_{ca} = u_{cb}$ であった．また，図 2 にプロジェクトの開発者数 $|U(C)|$ と $u_{ca} = u_{cb}$ となったプロジェクトの数，$u_{ca} \neq u_{cb}$ となったプロジェクトの数の散布図を示す．横軸がプロジェクトの開発者数（$|U(C)|$）であり，縦軸が対応するプロジェクトの数を対数で表している．青のドットが $u_{ca} = u_{cb}$ であったプロジェクトを表しており，赤のドットが $u_{ca} \neq u_{cb}$ であったプロジェクトである．同様に $u_{la} = u_{lb}$ であるプロジェクトの割合は 75.01%であった．

$|U(C)|$ が少ないプロジェクトの多くは $u_{ca} = u_{cb}$ となっている一方，$|U(C)|$ が 10 人を越えると $u_{ca} \neq u_{cb}$ となるプロジェクトが増えてくることがわかる．しかし，$|U(C)|$ が増えたとしても，$u_{ca} = u_{cb}$ のプロジェクトは存在し続けている．

次に，$u_{ca} \neq u_{cb}$ となるプロジェクトを調査した．$\mathrm{rank}_c(u_{cb}, C)$ を調査し，u_{cb} が全体のコミットの中でどれくらい頻繁にコミットしているのかを調査した．対象プロジェクト数は All は 186，$|U| > 5$ では 167，$|U| > 10$ では 142，$|U| > 100$ は 44 であった．

表 2 に結果を示す．$|U(C)|$ の行は，対象となるプロジェクトの開発者数の平均，最大を示している．また，$\mathrm{rank}_c(u_{cb}, C)$ 行は u_{cb} の全体のコミットの中での順位の平均を表している．$\mathrm{rank}_c(u_{cb}, C)/|U(C)|$ 行はラベルの通り，その順位がプロジェクト内でどれくらい上位になるかを表した指標である．

表 2 を見ると，例え $u_{ca} \neq u_{cb}$ であったとしても，u_{cb} のコミット数の順位は，$|U(C)| > 100$ の場合でも平均が 3.65 であり，トップ開発者であることが伺える．すなわち，プロジェクト開発者の中の中心的な役割を担う開発者が，ビルドファイルを編集することが多いことがわかる．

[*4] author の他に commiter 情報も存在するが，今回は committer 情報は利用していない．

3.2.2　ビルドファイルへの貢献度合

ここでは，u_{cb}，u_{lb} のビルドファイルへの貢献度合いを調査した．ビルドファイルへの貢献とは，貢献コミットと貢献行数で判断する．貢献コミットとは，ビルドファイルへのコミット全体のうち，当該ユーザ u のコミット回数の割合（$\frac{\mathrm{commits}(u, C^b)}{\sum_{i=1}^{U(C^b)} \mathrm{commits}(u_i, C^b)}$）とする．ビルドファイルへの貢献行数とは，最新のビルドファイルの各行のうち，当該ユーザの作成した行の割合を指す．

図 3 に u_{cb} の結果を示す．横軸は，割合の 10%間隔であり，一番右端のみ 100%である．縦軸は該当するプロジェクト数を表している．

図 3 を見ると，全体の 29.1%のプロジェクト（255 プロジェクト）がビルドファイルへのコミットの全てを u_{cb} が行っており，37.3%のプロジェクト（316 プロジェクト）で，ビルドファイルの全ての行を u_{cb} が書いたことになる．

図 3 より，貢献コミットは割合が下がるにつれ，該当するプロジェクト数はほぼ単調に減少している一方で，貢献行数は，割合が下がっても該当するプロジェクト数は 25〜55 と，ほぼ横這いである．貢献行数，貢献コミットの低いプロジェクトを調べてみたところ，開発の途中で使用するビルドシステムを変更し，新しいビルドシステムは，別の開発者が主に担当しているようであった．

3.3　RQ3. いつビルドファイルが更新されるか

ここでは，ビルドファイルがいつ更新されるかをプロジェクトのリリース日に着目して分析する．特にリリースからリリースまでを 3 つの区分に分け，各区分で行われるコミット回数の割合に着目する．これにより，いつビルドファイルの修正を行うのかを調査する．

リリースとは，GitHub のリリースページで作成されるものを指す[*5]．プロジェクトは複数のリリース

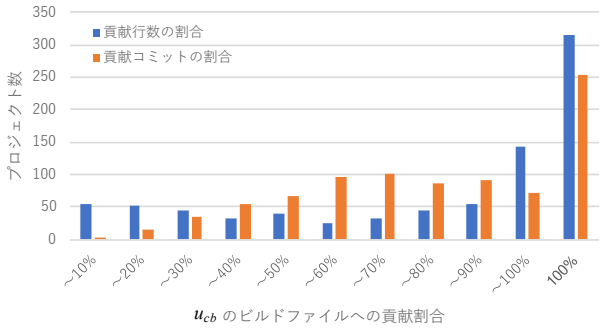

図 3　u_{cb} のビルドファイルへの貢献割合

[*5] https://help.github.com/articles/creating-releases/

$R = \{r_1, r_2, ..., r_x\}$ を持ち，リリース r は，名前，公開日，git の ref id(どのコミットかを指し示す id) を持つ．あるリリース r_{i-1} から別のリリース r_i までの期間を三等分し，各期間にどのくらいのコミットが行われたかを調査した．つまり，リリース r_i の公開日を d_i とした時，$\delta = \frac{d_i - d_{i-1}}{3}$ とする．そして，$\text{period}(d_{i-1}, d_{i-1} + \delta, C), \text{period}(d_{i-1} + \delta, d_i - \delta, C), \text{period}(d_i - \delta, d_i, C)$ のコミット数を調査した．それぞれ T_i^1, T_i^2, T_i^3 とする．ここで，T^k を T_i^k のリストとし，各要素は $T_i^1 + T_i^2 + T_i^3$ で正規化するものとする．

図 4 は全てのコミット C に対して，T^k を調べ，T^k を 0.1 間隔にした時のリリースの頻度を表したグラフである．同様に，図 5 はビルドファイルのコミット C^b に対して，T^k を調べ，T^k を 0.1 間隔にした時のリリースの頻度を表したグラフである．なお，対象プロジェクトの数は図 4 は 4,696 リリース，図 5 は 2,545 リリースである．なお，100%の項目を見ると，リリース間に T^1, T^2, T^3 のいずれかの時期にしかコミットを行なっていないケースである．

図 4 を見ると，10%の T^2 は 1923 と突出して多い．これは，T^2 の時期にコミットを行っているリリースが少ないことを表している．図 5 からは T^2 に加え，T^1 の時期もコミットの少ないプロジェクトが多いと言える．このことから，リリースの中盤にはプロジェクトへのコミットもビルドファイルも更新されることが少ないと言える．

一方で，図 4 の～100%の項目を見ると，$T^1 = 25, T^2 = 5, T^3 = 79$ と非常に低い値である．これは，多くのリリースで特定の時期に集中して開発しているわけではないことが言える．ただし，60%～90%の項目で，T^3 が T^1, T^2 に比べ大きくなっている．どちらかと言えば，後半にコミットが多くなっていることが伺える．

図 5 からは，～10%の項目で T^1, T^2 が突出して高いため，多くのリリースではビルドファイルのコミットを T^1, T^2 の時期に行なっていないように見える．一方で，100%の項目で T^3 が 862 リリースであった．T^3 の時期にビルドファイルを変更するリリースは，リリースに向けてビルドファイルの更新を行なったものと考えられる．

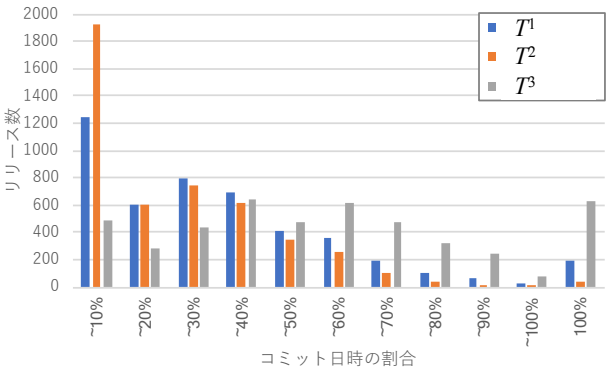

図 4　リリース間のコミット日時の割合

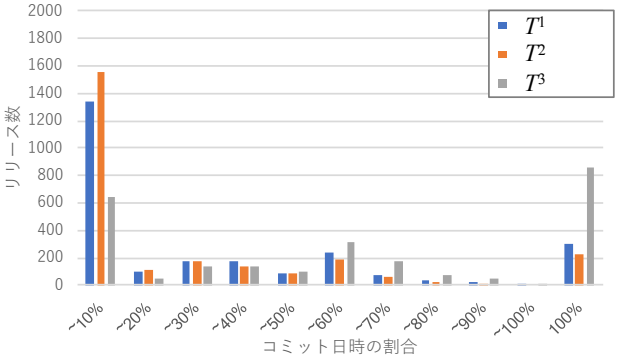

図 5　リリース間のビルドファイルのコミット日時の割合

4. 妥当性への脅威

本稿での調査では，プロジェクトの開発者の数について考慮していない．開発者数の規模が異なればプロジェクトの傾向が異なる可能性もある．開発者の規模で分類することで，よりプロジェクトの特徴に依存しない調査が行えると考えられる．

加えて，コミットの品質や規模についても考慮していない．すなわち，typo の修正によるコミットと機能追加のコミットを同列に扱っている．一方で，これらのコミットが全体のどれくらいを占めているのかが不明である．より信頼性の高い調査のためには，コミットの品質，規模の考慮が必要である．

5. 関連研究

Adams らは，ビルドファイルの複雑性を示し，Linux のビルドシステムの依存関係を図示した [2], [3]．これらの結果は，ビルドシステムのメンテナンスが困難な作業であることを示している．

また，Shane らは 著名な 10 の OSS プロジェクトに対してビルドシステムとソースコードの追加削除行数を調査した [4]．そして，ビルドファイルのメンテナンスがソースコード開発の 27%，テスト開発の 44%のオーバーヘッドをもたらすことを示した．同時にビルド職人への投資が影響範囲を減らすことも示している．

Shridhar らは，18 の OSS プロジェクトについて，ビル

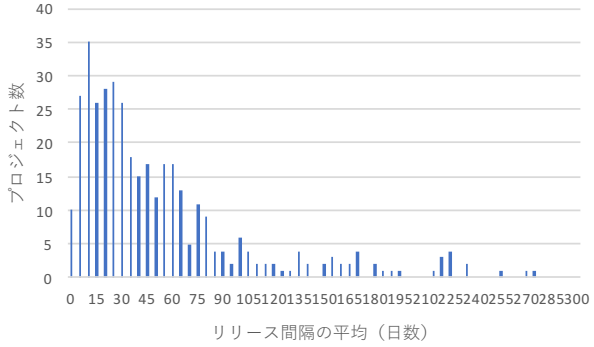

図 6　プログジェクトごとのリリース間隔の平均（日数）

ドファイルがどのように変更されるかを調査した [5]．その結果，修正，適応，機能追加に伴うビルドファイルの変更は，多くの追加削除行数があることを示した．

本研究では，大規模なプロジェクトに対して，調査を行い，プロジェクトで共通する特徴の導出に取り組んだ．

6. おわりに

本稿では，GitHub で公開されている Java プロジェクト約 1,000 個に対して，次の 3 つのリサーチクエスチョンに答えるため分析を行った．

RQ 1. どのようなビルドツールが使われているか．

RQ 2. 誰がビルドファイルを更新しているか．

RQ 3. いつビルドファイルが更新されるか．

RQ1. では，言語ごとに人気順トップ 1,000 プロジェクトを対象に，利用しているビルドツールを調査した．その結果，Java, Ruby, Python はデファクトスタンダードとなるビルドツールが存在することが伺えた．

RQ2. では，誰がビルドファイルを更新しているかをビルドファイルのコミットと全体のコミットを関連づけて調査した．調査対象の 868 プロジェクトの 78.4%ものプロジェクトでプロジェクトのトップ開発者がビルドファイルをもっとも頻繁に更新していることがわかった．

RQ3. では，ビルドファイルの更新時期をリリースに着目して調査した．特定の時期にコミットするリリースが非常に多く，特にリリースの前に更新するケースが 862 件であった．このことから，ビルドファイルは頻繁に更新されないものの，更新される場合は，リリース前であることが多いことが伺える．

第 4 節で述べたように，本稿での調査はプロジェクト固有の事情を考慮していない．すなわち，例外的な事象がどれくらい含まれているかが不明である．そのため，例外的な事象を調査し，それを排除した上での調査を行うことが，より信頼性の高い調査になる．

今後の課題として，次のことが挙げられる．本稿では，コミット回数，貢献行数で降順に整列した時，最上位の開発者しか調査対象にしていない．最上位だけでなく，上位数名の開発者を対象に調査を行う．また，プロジェクトのトップディレクトリ以外に存在するビルドファイルのコミットについても調査が必要である．

謝辞 本研究の一部は JSPS 科研費 17K00196，17K00500，17H00731 の助成を受けたものです．

参考文献

[1] Stuart I. Feldman. Make —a program for maintaining computer programs. *Journal of Software Practice and Experience*, Vol. 9, No. 4, pp. 255–265, April 1979.

[2] Bram Adams, Herman Tromp, Kris de Schutter, and Wolfgang de Meuter. Design recovery and maintenance of build systems. In *Proc. IEEE International Conference on Software Maintenance (ICSM 2007)*, 2007.

[3] Bram Adams, Kris De Schutter, Herman Tromp, and Wolfgang De Meuter. The evolution of the linux build system. In *Proc. 3rd International ERCIM Symposium on Software Evolution*, Vol. 8, 2008.

[4] Shane Mcintosh, Bram Adams, Thanh H.D. Nguyen, Yasutaka Kamei, and Ahmed E. Hassan. An empirical study of build maintenance effort. In *Proc. of the 33rd International Conference on Software Engineering (ICSE 2011)*, pp. 141–150, 2011.

[5] Mini Shridhar, Bram Adams, and Foutse Khomh. A qualitative analysis of software build system changes and build ownership styles, 2014.