# 卒 業 研 究 報 告 書

題　目　　構成管理ツールにおける命令的操作が及ぼす
ソースコードレビューへの影響調査

指導教員　　水野　修　教授

崔 恩瀞　助教

京都工芸繊維大学 工芸科学部 情報工学課程

学生番号　　17122039

氏　　名　　頭川 剛幸

令和3年2月12日提出

# 構成管理ツールにおける命令的操作が及ぼすソースコードレビューへの影響調査

令和 3 年 2 月 12 日　　17122039　頭川 剛幸

## 概　要

近年増加するソフトウェアの需要に対応するため，企業は DevOps に基づいた開発プロセスを採用している．インフラストラクチャの自動化はそれらの内の一つであり，コードによって計算機の状態を定義可能な構成管理ツールを用いて達成される．構成管理ツールには記述方法として宣言型と命令型の 2 通りが存在し，宣言型では管理対象の計算機に対する状態の記述を行い，命令型ではユーザが自由にスクリプトを組むことで目的の状態に到達する．命令型を用いることは，環境の再現性や堅牢性のために必要とされる冪等性の脅威になるとされている．しかし，それらがソフトウェア開発において，どのような影響を与えているのか未だ明らかになっていない．

本研究では，構成管理ツールである Ansible に着目し，命令型を含む操作（命令的モジュール）の使用によって，コードレビューに必要な時間が増大すると仮説を立てた．命令的モジュールを用いた記述を含むプルリクエスト（PR）について，PR が評価されるまでの時間，PR の持つ各レビュースレッドの所要時間，PR で変更された行数およびファイル数について調査し，命令的モジュールを含む PR とそうでない PR においてどのような差が見られるのか調査を行った．

命令的モジュールの使用は，マージされた PR の評価時間を増大させたが，その差は微々たるものであった．また各レビュースレッドでの議論に必要な時間に対しては時間差が見られなかった．PR 中に行われた変更行数について，マージされた PR に関して約 2 倍の増加が見られ，ファイル数が 2，3 個ほど増加した．

これらの結果から，命令型の使用は変更規模を増大させ，レビューを行う際に確認が必要な項目が増大する可能性があるが，それに伴うレビュー時間の増大が見られなかったため，ソフトウェア開発でのコードレビューにおいて，命令型の使用は問題とならない可能性を示唆している．

# 目　次

# 1. 緒言

近年のソフトウェアに対する需要の高まりから，企業は迅速かつ高品質に，信頼性のある製品を提供することが求められている．需要に対応するため，開発プロセスと運用を一つのサイクルにしたDevOpsモデルが普及した．DevOpsの要素の1つとしてインフラストラクチャの自動化が存在する [1–3]．プロビジョニング，構成，デプロイを自動化することで運用，開発規模の増大に対応することができる．これを達成するツールとして，Puppet [4]，Chef [5]，Ansible [6] 等が開発され現在でも多くの企業で使用されている．インフラにおける最古の構成管理ツールは，1993年に開発されたCFEngineと言われている [7,8]．当時は，サーバの設定に対する自動化が主な目的であった．その後，2005年にCFEngineの影響を受けたツールであるPuppetが登場し，2008年に現ChefのCTOであるAdam Jacob氏によって初めてInfrastructure as Code（以下IaC）という言葉が使われたとされている [9]．2008年頃，AgileやDevOpsの考えが構成管理の概念と結びつき，サーバの設定だけでなくバージョン管理，テスト，CIなどのソフトウェア開発プロセスをシステム管理に応用するものとしてIaCが広く使用されるようになった [10]．

IaCにはシステムの表現方法として大別すると，命令型と宣言型の2種類が存在する．宣言型はマシンのあるべき状態を記述することによってシステムを構築し，命令型では制御対象のマシンの状態毎に必要な操作を明示的に記述する．命令型を使用する場合は制御をユーザに委ねるため，複雑な命令を実行できる一方，正確な手順を慎重に計画する必要があり，大規模なソフトウェア環境をデプロイする場合，拡張性がなく失敗しやすいとされている [11]．

また，IaCが繰り返し実行可能かつ堅牢なシステムであるための基礎として冪等性の概念が存在する [12]．冪等性とは，異なる条件下において複数回の繰り返しを行うことで全て同一な状態に収束する概念のことであり，代数学の基礎として研究されている [13,14]．例として任意の整数のゼロ乗算や，ある整数の絶対値を求める操作などが該当する．IaCで冪等性を担保する方法として，ツールが用意する冪等な操作を行うモジュール(注 1)（以下冪等モジュール）を用いる場合と，ユーザ自らが冪等性の担保を行う場合の2種類が存在する．本研究の調査対象であるAnsibleにお

(注 1): 実行されるコードの最小単位

いて宣言型を用いるモジュールは冪等性を持つ場合が多いが，命令型と一部の宣言型は冪等性を持たないモジュール（以下非冪等モジュール）であり，ユーザがその実行に冪等性をもたせる必要がある．実際，先行研究から Ansible [6] における非冪等モジュールを使用したタスクの内約 51.32%がその実行について条件を持つため [15]，ユーザはコマンドが及ぼすマシン状態への影響を把握し，冪等性を持たせようとしていると考えられる．これらの拡張性が低いことや冪等性の管理をユーザが行う必要がある点から，命令型の使用は問題があるとされているが，実際のソフトウェア開発現場においてどのようなの影響を与えているかは明らかになっていない．

本研究では，冪等性の担保の必要性や記述内容の複雑化が考えられる命令型の使用によって，レビューに掛かる時間が増大すると仮定を立て調査を行った．今回は先行研究と同様に Ansible に着目し，データセットとしてユーザが自由にアップロード可能なサイトである Ansible Galaxy [16] から，GitHub で公開されているリポジトリの内プルリクエスト（以下 PR）に関するデータを抽出する．それらを用いて，命令的モジュールを使用した場合 PR 中のレビュー時間や変更規模に変化が見られるかを明らかにした．ユーザが自由にスクリプトを実行可能である `command`・`shell`・`raw`・`script` の 4 種類の命令型操作を行うモジュール（以下命令的モジュール）を分析対象とし，以下の研究設問（以下 RQ）について調査した．

RQ1) **PR および PR 中のレビュースレッドの所要時間に差が見られるか**

命令的モジュールの使用によって制御構造や実行結果の予測が複雑化し，レビューに必要な時間が増加すると考えた．比較方法として，1 件以上のレビュースレッドを持つ PR に対し，各 PR 中に変更が行われたいくつかのファイルに命令的モジュールを含むものがあるか，またレビュースレッドの指摘箇所に命令的モジュールを含むかどうかを基準として，PR をそれぞれ 2 つのグループに分類した．PR が開始してからマージもしくはクローズされるまでの時間，PR に含まれるレビュースレッドが開始してから最終のコメントが行われるまでの時間のパーセンタイルを求めた．結果として，場合によっては命令的モジュールの使用がレビュー時間を増大させたが，実用上で問題とするほど顕著な差とは言えず時間的影響は少ないと言える．

RQ2) PR 中の変更規模に差が見られるか

命令的モジュールの使用により変更箇所が複雑化する場合，それらに必要な変更規模が大きくなると考えた．RQ1 と同様にレビュースレッドを持つ PR に対し，変更を含むファイルが命令的モジュールを持つか，また，レビュースレッドの指摘箇所に命令的モジュールが含まれるかによってそれぞれの場合で分類を行い，変更された行数とユニークなファイル数を求めた．命令的モジュールを含むマージされた PR では変更行数が約 2 倍ほど増加し，ユニークなファイル数が 2，3 ファイル程度増加したことから，命令的モジュールの使用がこれらの値に影響を与えている可能性がある．

これらの結果は，命令的モジュールの使用はレビューへの負荷になりにくいが，開発者が変更を行う際の負担が増加する可能性を示唆している．

## 2. 研究動機例

命令的モジュールの使用がユーザへ与える負担となる例をソースコード 2.1 および 2.2 に示す．この例はどちらも AWS 上で同様のインスタンスを作成するが，前者は Ansible によって用意された宣言的モジュールを使用し，後者はシェルスクリプトの実行によってインスタンスを作成するものである．前者の場合，システムの堅牢性を保つための要素である冪等性はモジュールが担保しているため，ユーザは必要なパラメータを指定すればよい．具体的には，`community.aws.rds` より以下に示されるパラメーターを定義することで，宣言的にシステムの状態を記述している．後者は，実行する CLI のオプションを調査した上で，それが冪等性を保つように実行制御を行う必要がある．具体的には，`shell` から始まる `aws` コマンドのオプションを指定し，命令的に実行の制御を行っている．また，冪等性を持たせるため，一度実行を行った場合，変数（例では `is_instance_exist`）を用意し，実行結果を Ansible に記録させる．2 度目の実行が行われた場合，`failed_when` によって条件を追加し，結果が記録されている場合には実行させないことで冪等性を担保している．また，この場合実行結果の状態について，宣言的モジュールを用いた場合，最終的な状態が記述されているため視覚的に理解しやすいと言える．本研究では，これらの差異がソースコードレビューにどのような影響を与えているのか明らかにする．

## ソースコード 2.1　宣言的モジュールを用いたデータベース作成例 [17]

```
1  ---
2  - name: Basic mysql provisioning example
3    community.aws.rds:
4      command: create
5      instance_name: new-database
6      db_engine: MySQL
7      size: 10
8      instance_type: db.m1.small
9      user_name: mysql_admin
10     password: 1nsecure
```

## ソースコード 2.2　命令的モジュールを用いたデータベース作成例

```
1  ---
2  - name: Basic mysql provisioning example (imperative)
3    shell: "aws rds create-db-instance\
4                --allocated-storage 10\
5                --db-instance-class db.m1.small\
6                --db-instance-identifier new-database\
7                --engine mysql\
8                --master-username mysql_admin\
9                --master-user-password 1nsecure"
10   register: is_instance_exist
11   failed_when: is_instance_exist not in [0..1]
```

# 3. 背景

## 3.1 Infrastructure as Code

Infrastructure as Code（IaC）は，ソフトウェア開発のプラクティスに基づいたインフラストラクチャ自動化のアプローチである．これは，システムのプロビジョニングと変更，およびその構成の変更を行うための操作を，繰り返し可能で一貫性のあるものにすることに重きをおいている．つまり，システムの状態を全てコードで定義し，変更を加える場合は対象のマシンを直接操作するのではなく管理するコードに変更を加え，自動化を使用してテストを行い，その変更をシステムに適用する運用方法である [18].

## 3.2 構成管理ツール

IaC を達成するために用いられるツールであり，主に対象のコンピューティングリソースに対して環境構築を行うことを目的とする．ミドルウェアの設定やクラウドリソースの構築，ネットワークトポロジの設定等を行う．代表的なツールとして，Ansible，Chef，Puppet がある．

## 3.3 構成管理ツールの冪等性

冪等性は，複数の状態から一意の状態へ収束する性質のことであり，IaC において冪等性はシステムの再現性，堅牢性に対し重要である．Chef や Ansible 等の構成管理ツールでは冪等な操作を行うモジュールを用意しており，ユーザはこれらを用いることでシステムの冪等性を担保している．また，構成管理ツールは冪等でないモジュールも用意しており，これらの使用は冪等性に対する脅威となる．Chef におけるシステムの冪等性を判定するモデルを作成した研究 [12] では冪等性に対する脅威として，(1) 命令的な Chef スクリプトを用いる，(2) スクリプトリソースを用いる，(3) 全体のシステム冪等性を持たない，(4) 外部のリソースに依存している点を挙げている．

## 3.4 Ansible

本研究の調査対象とする構成管理ツールである．Red Hat, Inc. によって開発されているオープンソースソフトウェアである．特徴として，管理対象のマシンに特別なソフトを必要としないことが挙げられる．

### 3.4.1 モジュール

リモートマシンで実行されるコードの最小単位であり，アプリケーションのインストールやクラウド上のインスタンス作成など様々な種類を持つ．多くのモジュールは key=value 引数を指定することで実行内容を制御する．執筆時点での最新安定版である Ansible 2.10.5 において，4,573 個のモジュールが存在する．

### 3.4.2 宣言的モジュールと命令的モジュール

宣言的モジュールとは，引数によって状態を定義するモジュールであり，実際に実行される命令は各モジュールに定義されている．命令的モジュールとは，ユーザが命令を実際に記述し状態を制御する．宣言的モジュールと命令的モジュールについて Ansible は，各モジュールがどちらに属するものか明記していない．本研究では，先行研究 [15] と同様にシェルスクリプトを直接実行可能である `command`，`shell`，`script`，`raw` モジュールを命令的モジュールとした．

### 3.4.3 ロール

Ansible が規定するディレクトリ構造の名称であり，それに従って適切にファイルを配置することで一連の機能の再利用を容易にすることができる．また，Ansible Galaxy [16] ではロールの形式に従ってファイルが共有されている．

## 3.5 プルリクエスト（PR）とレビュースレッド

PR とは，第三者がある変更を行った際にそれらの変更が妥当なものであるか管理者がチェックを行うものであり，GitHub 上の OSS 開発で一般的に用いられている．また，PR の機能として変更箇所に対してレビュースレッドを建てることができ，それらに対して詳細な議論を行うことができる．

### 3.5.1 PR の持つ要素

PR の持つ要素から本研究で取得したものを図 3.1 に示す．

1. PR の状態を表すラベル．すなわちマージ，クローズ，オープン
2. PR の開始時間から終了までの時間
3. 各レビュースレッドの開始時間から終了までの時間
4. 変更が行われたファイル群および変更行数
5. レビュースレッドの対象となるファイルおよび指摘箇所

## 3.6 PR への時間的調査

本研究と同様に，GitHub の PR に対して時間的な調査を行った研究が存在している．Yue らは，PR の評価が終了するまでに掛かる時間に影響を与える因子を調査しており，データセットとして Travis-CI を用いた GitHub リポジトリから PR を取得し，リポジトリの規模や更新頻度，リポジトリ参加者の技量などの側面から調査を行っている [19]．例としてはリポジトリが生成されてからの期間の長さや参加者の数，PR に含まれるコミット数，PR で追加，削除された行数，PR 投稿者の被マージ率等を調査している．PR が終了するまでの時間へ影響を与える因子として，コミット数，及び追加された行数が増えるほど PR の評価に時間を要し，リポジトリのコアメンバー，フォロワーの多い投稿者，過去のマージ率が高い投稿者による PR はより短時間で評価を終えるとされている．

先行研究では，命令的モジュールの利用されている実態を明らかにし，冪等性を担保するための努力や代替可能性を示した．実際，命令的モジュールを使用する場合，宣言的にモジュールを使用する場合に比べ，冪等性の担保のための努力がされ

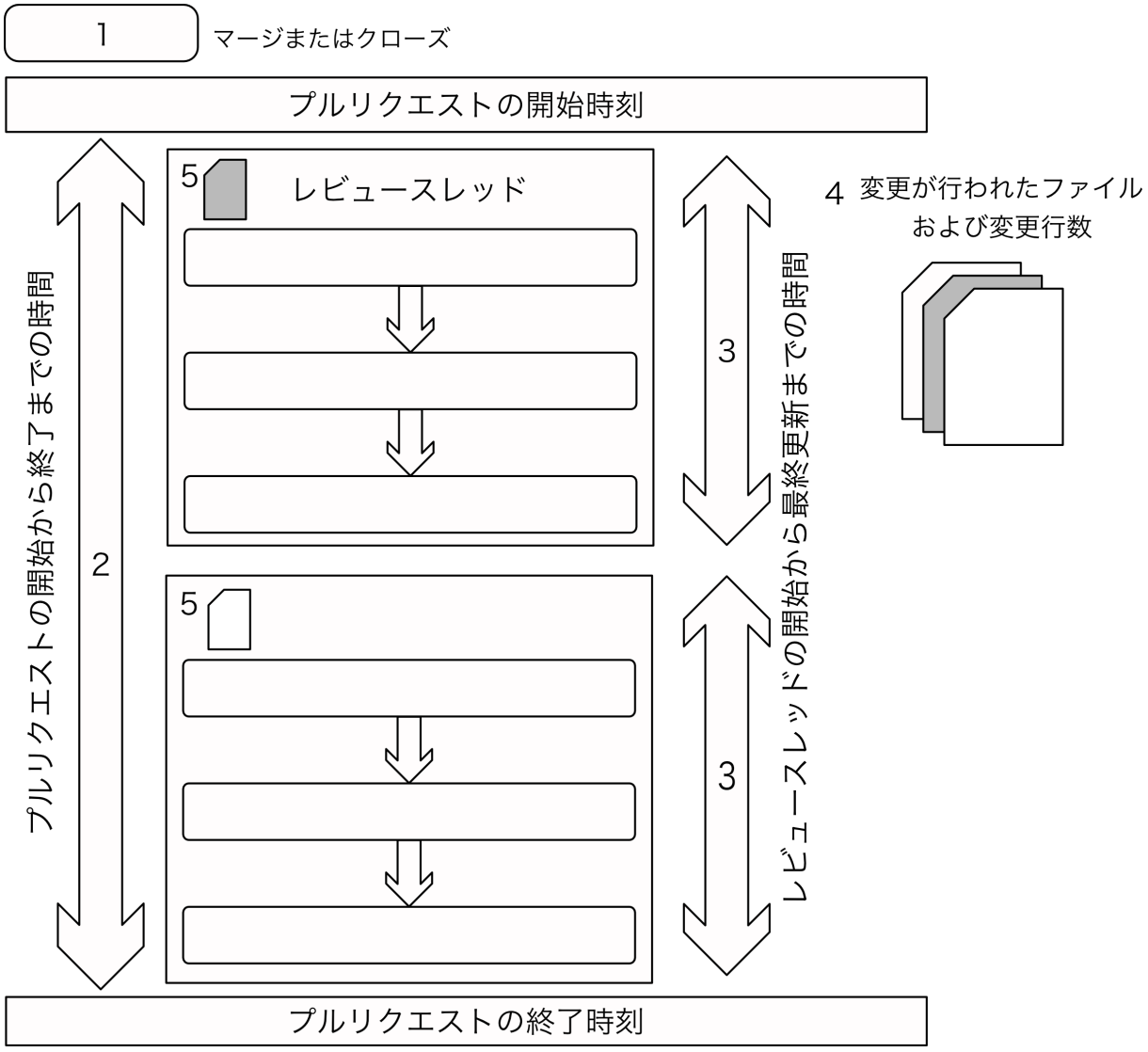
1
マージまたはクローズ
プルリクエストの開始時刻
5
レビュースレッド
4 変更が行われたファイル
および変更行数
プルリクエストの開始から終了までの時間
2
3
レビュースレッドの開始から最終更新までの時間
5
3
プルリクエストの終了時刻

**図 3.1　PR の持つ要素**

ていることが明らかになっているが，時間的にどのような影響を与えているか明らかではない．そのため，本研究の目的であるソースコードレビューへの影響として取り上げたPRの評価時間や，PRでの変更規模についてはこれらの結果を参照するだけでは不十分である．

# 4. 調査概要

先行研究において命令的モジュールの使用には問題があるとされているが，実際には約半数のロールに命令型が使用されている．本研究の目的は，命令型の使用が開発現場において実際にどのような影響を与えているか明らかにすることである．命令型の使用には冪等性を担保するための条件分岐が必要であることや，コマンドの内容が煩雑である場合，実験内容を把握することが難しくなっていくことから，命令的モジュールの使用が記述を煩雑にし，レビューに掛かる時間が増大すると仮説を立てて調査を行った．

調査対象である Ansible では，OSS コミュニティである Ansible Galaxy と呼ばれる Web サービスが存在し GitHub のリポジトリが公開されている．公開されているリポジトリからレビューに関する情報を得るため，GitHub の機能である PR のデータを収集した．PR 中では，プロジェクトへの変更の妥当性を確認するため第三者によるレビューが行われ，レビュー内容によって変更が行われる．命令的モジュールの使用がこれらのレビューや変更時間に影響を与えているか調査した．

評価に用いるデータとして，PR を閉じるまでの所要時間，PR 中のレビュースレッドでの所要時間，変更された総行数を指標とした．また，分類を行うために PR がマージまたはクローズかどうかの状態，変更が行われたファイルおよび変更箇所を収集した．これらの情報から，命令的モジュールの使用状況と PR の状態によって，レビュースレッドを 1 件以上含む PR に対し，命令的モジュールの有無，PR がマージもしくはクローズであるか分類を行い，それぞれのパーセンタイルを求め，値に差が見られるかどうかによって影響の有無を調査した．マージまたはクローズされた PR という場合分けを行った理由として，マージされた PR はリポジトリに変更を与えるため，より活発な議論になる可能性が考えられる．その場合，クローズされた PR との違いを明らかにするため，これらの場合分けを行った．

# 5. 調査手順

## 5.1 データセットの概要

本研究では先行研究 [15] と同様に Ansible に限定して調査を行った．また，先行研究で行われた，ファイル中の命令的モジュールの有無に対する分類結果を用いたため，使用したデータセットは先行研究と同様のものとした．Ansible Galaxy で公開されているロールの内，2019 年 10 月 22 日時点でダウンロード数が上位 10%であり，その時点から過去 1 年以内に更新されたものとしている．それらのロールから GitHub のリポジトリの URL を取得し，GitHub API [20] からマージまたはクローズされた PR に関するデータを収集した．

## 5.2 データセットの取得方法

GitHubAPI から得られるデータのうち以下の要素を取得した．

- PR の状態，すなわちマージまたはクローズ
- PR の開始時間及びマージまたはクローズされた時間
- 各 PR 中に含まれるレビュースレッド開始時間及び最終コメント時刻
- PR 全体及びファイル単体での追加，削除された行数
- PR 中に変更が行われたファイル群
- レビュースレッドの対象となっている変更内容

対象とする Ansible ロールの全てのリポジトリに対してこれらのデータを収集した．対象とした Ansible ロールは先行研究 [15] で使用された 884 個のロールを選択しデータの収集を行った．データ収集が完了した 2021 年 2 月 1 日時点での 26,567 件の PR から，レビュースレッドを含む，マージまたはクローズされた 2,652 件の PR，および PR 中に行われた 7,944 件のレビュースレッドを取得した．

## 5.3 RQ1 への回答方法

命令的モジュールの使用が，制御構造や実行結果の予測を複雑にする場合，レビュー時間が増大すると考えた．PR およびレビュースレッドへの所要時間を調査しそれらのパーセンタイルを比較することによって，命令的モジュールを含む場合と命令的モジュールが含まれない場合に差があるか調査し影響を明らかにする．

## 5.4 RQ2 への回答方法

命令的モジュールの使用によって変更箇所が複雑になる場合，それに伴い変更行数が増大すると考えた．PR で変更されたファイルが命令的モジュールを持つ，またはレビュースレッドでの指摘箇所に命令的モジュールが含まれるかによって PR の分類を行う．PR 中に変更されたファイル数および変更された行数についてそれぞれパーセンタイルを比較し，命令的モジュールの有無によって差が見られるか明らかにする．

# 6. 結果

## 6.1 RQ1) PR および PR 中のレビュースレッドの所要時間に差が見られるか

### 6.1.1 動機

PR やレビュースレッドでは変更内容の妥当性について第三者が確認を行うが，命令的モジュールの使用がその確認に必要な時間に影響を与えているか確認した．

### 6.1.2 手順

PR 内に一件以上レビュースレッドを持つものに関して，PR の開始時刻からマージまたはクローズされるまでの時刻を，PR の解決までの所要時間（以下変更解決時間）として調べた．同様に，レビュースレッドに関して，開始時刻と最終更新時刻の差から所要時間（以下レビュー時間）を求めた．調査の際，以下の場合に分けてそれぞれの時間を調べた．

- 条件 1: 変更されたファイルのうち命令的モジュールを含んだファイルが 1 つ以上存在する
- 条件 2: レビュースレッドの指摘対象となる変更内容に命令的モジュールが 1 つ以上存在する

これらの条件を視覚的に表したものを図 6.1，6.2 および 6.3 に示す．

条件 1 では，レビュースレッドの指摘箇所に関わらず，変更された全てのファイルのうち命令的モジュールを含むものがあるかどうかによって判断を行っているため，図 6.1 では命令的モジュールを含まない PR，図 6.2 では命令的モジュールを含む PR とした．

条件 2 では，レビュースレッドの指摘箇所とされる変更内容に命令的モジュールを含むかどうかによって判断を行い，図 6.2 では命令的モジュールを含んだ変更を指摘対象としていないため，命令的モジュールを含まない PR，図 6.3 では命令的モジュール含んだ変更を指摘対象とするため，命令的モジュールを含む PR とした．また，図中でグレーとなっているものは，ファイル中に命令的モジュールを含むものである．

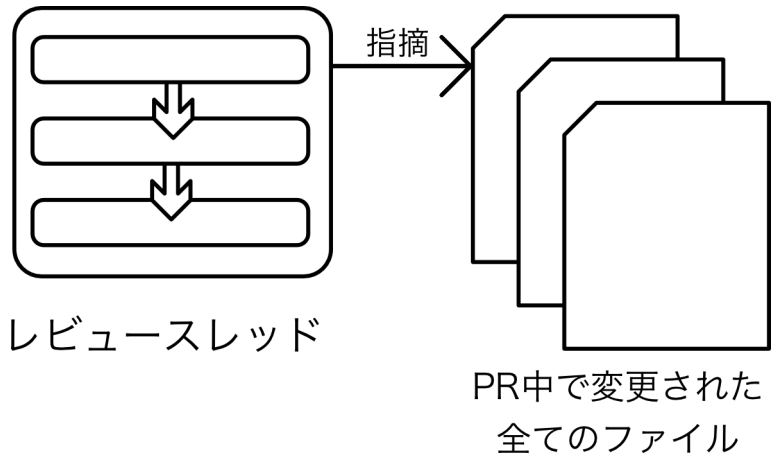


**図 6.1　命令的モジュールを含まない PR のレビュースレッド**

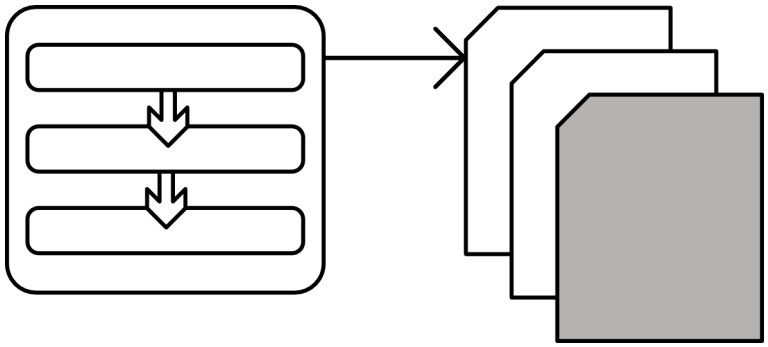

**図 6.2　命令的モジュールを含む PR のレビュースレッド**

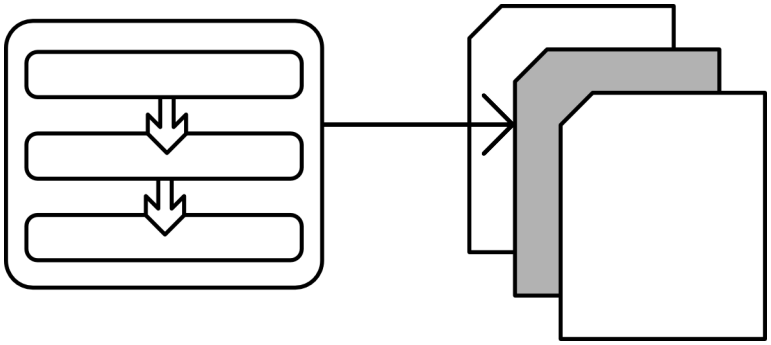

**図 6.3　命令的モジュールを対象とする PR のレビュースレッド**

### 6.1.3 結果

命令的モジュールを含むファイルを 1 つ以上持つか持たないかで分類した場合における，PR 全体および各レビュースレッドの所要時間に関する調査結果を表 6.1 および 6.2 に示す．また，命令的モジュール持つ PR は 573 件であり全体の約 21.6%（573/2,652），命令的モジュールを持つレビュースレッドは 1,364 件であり，全体の約 17.2%（1,364/7,945）であった．

PR での差が最も大きくなったものはクローズ状態での 75 パーセンタイルの値どうしの比較で約 159 日（42,902,418 秒 – 29,155,392 秒）の差が生じた．レビュースレッドでの比較でも同様にクローズ状態での 75 パーセンタイルでの差が最も大きくなり，半日（346,547 秒 – 290,160 秒）程度の差が生じた．また，全体的な傾向としてマージまでの時間はクローズされるまでの時間よりも短くなることがわかった．

次に，レビュースレッドの対象となっている変更内容に命令的モジュールを含むかどうかによって分類し，同様に調査を行った結果を表 6.3 および 6.4 に示す．また，命令的モジュールを持つ PR は 292 件であり全体の約 11%（292/2,652），命令的モジュールを持つレビュースレッドは 999 件であり全体の約 12.5%（999/7,945）であった．ファイル中に含まれるかどうかによって分類した場合とは割合が異なる理由として，レビュースレッドで指摘される必要があり，ファイルに含まれるかどうかに比べ条件が厳しくなったため，割合が減少したと考えられる．PR がマージされた場合，命令的モジュールを使用する場合の各パーセンタイルでの値が，そうでない場合より長くなる傾向にあった．クローズ状態の PR では各パーセンタイルでの値が命令的モジュールを使用しない場合よりも長くなった．レビュースレッドでの比較では，マージ状態の PR では顕著な差が見られなかった．クローズ状態では 25 パーセンタイルおよび 50 パーセンタイルでは命令的モジュールを含む場合が長く，75 パーセンタイルで命令的モジュールを含まない場合が長くなった．
表中の各行における PR の種類について対応は以下の通りである．

- imperative_merged: マージされた状態の PR のうち命令的モジュールを含むもの
- declarative_merged: マージされた状態の PR のうち命令的モジュールを含まないもの

- imperative_closed: クローズされた状態の PR のうち命令的モジュールを含むもの
- declarative_closed: クローズされた状態の PR のうち命令的モジュールを含まないもの

表 6.1　PR の所要時間

| PR の種類 | 最小値 (秒) | 25%(秒) | 50%(秒) | 75%(秒) | 最大値 (秒) |
|---|---|---|---|---|---|
| imperative_merged | 588 | 81,520 | 327,171 | 1,316,762 | 63,204,010 |
| declarative_merged | 10 | 77,977 | 336,912 | 1,222,813 | 72,241,519 |
| imperative_closed | 354 | 1,348,705 | 8,796,006 | 42,902,418 | 148,978,261 |
| declarative_closed | 481 | 1,123,010 | 8,390,101 | 29,155,392 | 156,254,992 |

表 6.2　レビュースレッドの所要時間

| PR の種類 | 最小値 (秒) | 25%(秒) | 50%(秒) | 75%(秒) | 最大値 (秒) |
|---|---|---|---|---|---|
| imperative_merged | 21 | 2,105 | 16,749 | 81,432 | 26,913,820 |
| declarative_merged | 12 | 2,033 | 15,291 | 90,039 | 37,236,864 |
| imperative_closed | 17 | 4,599 | 52,120 | 290,160 | 38,302,858 |
| declarative_closed | 21 | 7,785 | 67,379 | 346,547 | 63,489,097 |

表 6.3　PR の所要時間 (変更内容に命令的モジュールを持つ)

| PR の種類 | 最小値 (秒) | 25%(秒) | 50%(秒) | 75%(秒) | 最大値 (秒) |
|---|---|---|---|---|---|
| imperative_merged | 728 | 150,502 | 515,049 | 1,732,437 | 45,703,527 |
| declarative_merged | 10 | 82,326 | 364,870 | 1,352,164 | 63,204,010 |
| imperative_closed | 46,148 | 1,458,642 | 5,656,538 | 19,279,142 | 115,091,803 |
| declarative_closed | 481 | 1,347,253 | 8,782,023 | 30,198,342 | 152,928,456 |

表 6.4　レビュースレッドの所要時間 (変更内容に命令的モジュールを持つ)

| PR の種類 | 最小値 (秒) | 25%(秒) | 50%(秒) | 75%(秒) | 最大値 (秒) |
|---|---|---|---|---|---|
| imperative_merged | 12 | 1,837 | 16,583 | 80,108 | 5,048,365 |
| declarative_merged | 11 | 2,125 | 14,616 | 89,703 | 37,236,864 |
| imperative_closed | 46 | 8,330 | 99,580 | 349,767 | 38,302,858 |
| declarative_closed | 17 | 4,201 | 58,535 | 405,292 | 63,489,097 |

## 6.2 RQ2) PR 中の変更規模に差が見られるか

### 6.2.1 動機

命令的モジュールはシェルスクリプトを直接実行できるため，自由な記述が可能であり外部スクリプトの実行が可能であるが，それらが及ぼす変更箇所への影響について明らかではない．PR 中に変更された行数を調べることで，命令的モジュールを使用する場合に変更の規模が増加しているのか調査した．

### 6.2.2 手順

PR 中に変更されたそれぞれのファイルでの追加および削除された行数を取得した．新たに 1 行が追加された場合は追加行数 1，ある 1 行が削除された場合は削除行数 1，ある 1 行が一部変更された場合は追加および削除となり 2 行とカウントされる．これらの値を合計したものを変更行数，その PR での各ファイルの変更行数を合計したものを総変更行数とした．変更された全てのファイル数を変更ファイル数とした．RQ1 と同様の条件を用いて命令的モジュールの有無による分類を行い，各パーセンタイルを比較した．

### 6.2.3 結果

ファイル中に命令的モジュールを含むかどうかによって分類した場合における，PR 中の総変更行数，ファイル変更行数および変更ファイル数に関する調査結果を表 6.5，6.6 および 6.7 に示す．また，調査対象の PR のうち命令的モジュールを含むユニークファイルは 874 件であり，全体の約 8.0%（874/10,978）であった．マージ，クローズの場合それぞれで比較した際に，命令的モジュールの有無によって各パーセンタイルで約 1.2 倍の差が見られた．また，ファイル単位での変更数の比較ではそれぞれに違いが見られなかった．変更が行われたファイル数は命令的モジュールを使用した場合に増加が見られた．

変更箇所での命令的モジュールの有無によって分類した場合における，PR 中の総変更行数，ファイル変更行数および変更ファイル数に関する調査結果を表 6.8，6.9 および 6.10 に示す．変更箇所に命令的モジュールを含むユニークファイル数は 287 件であり全体の約 2.6%（287/10,978）であった．PR での変更行数はマージ，クローズ

同様に命令的モジュールを使用した場合に変更行数が約 2 倍増加した．また，ファイル単体での比較では，マージ，クローズのどちらも各パーセンタイルの値の差が大きくなり，命令的モジュールを使用した場合にファイルあたりの変更行数が増加した．変更が行われたファイル数は，ファイル中に命令的モジュールを含むかどうかによって分類した場合と同様な増加が見られた．

**表 6.5　PR 中の総変更行数**

| PR の種類 | 最小値 (行) | 25%(行) | 50%(行) | 75%(行) | 最大値 (行) |
|---|---|---|---|---|---|
| imperative_merged | 1 | 15 | 47 | 156 | 16,825 |
| declarative_merged | 0 | 14 | 36 | 122 | 22,870 |
| imperative_closed | 1 | 11 | 30 | 92 | 3,359 |
| declarative_closed | 2 | 12 | 28 | 80 | 11,904 |

**表 6.6　ファイル単体での変更行数**

| PR の種類 | 最小値 (行) | 25%(行) | 50%(行) | 75%(行) | 最大値 (行) |
|---|---|---|---|---|---|
| imperative_merged | 0 | 4 | 11 | 28 | 663 |
| declarative_merged | 0 | 2 | 7 | 21 | 3,813 |
| imperative_closed | 0 | 4 | 9 | 22 | 279 |
| declarative_closed | 0 | 3 | 7 | 21 | 1,922 |

**表 6.7　変更ファイル数**

| PR の種類 | 最小値 (行) | 25%(行) | 50%(行) | 75%(行) | 最大値 (行) |
|---|---|---|---|---|---|
| imperative_merged | 1 | 3 | 5 | 9 | 115 |
| declarative_merged | 1 | 2 | 3 | 6 | 95 |
| imperative_closed | 1 | 2 | 4 | 6 | 91 |
| declarative_closed | 1 | 1 | 3 | 4 | 138 |

**表 6.8　PR 中の総変更行数 (変更内容に命令的モジュールを持つ)**

| PR の種類 | 最小値 (行) | 25%(行) | 50%(行) | 75%(行) | 最大値 (行) |
|---|---|---|---|---|---|
| imperative_merged | 2 | 22 | 60 | 189 | 7,223 |
| declarative_merged | 1 | 10 | 26 | 94 | 16,825 |
| imperative_closed | 2 | 11 | 25 | 73 | 580 |
| declarative_closed | 1 | 9 | 22 | 61 | 5,952 |

**表 6.9　ファイル単体での変更行数 (変更内容に命令的モジュールを持つ)**

| PR の種類 | 最小値 (行) | 25%(行) | 50%(行) | 75%(行) | 最大値 (行) |
|---|---|---|---|---|---|
| imperative_merged | 1 | 8 | 19 | 40 | 663 |
| declarative_merged | 0 | 2 | 7 | 21 | 3,813 |
| imperative_closed | 1 | 5 | 16 | 27 | 101 |
| declarative_closed | 0 | 3 | 7 | 20 | 1,922 |

**表 6.10　変更ファイル数 (変更内容に命令的モジュールを持つ)**

| PR の種類 | 最小値 (行) | 25%(行) | 50%(行) | 75%(行) | 最大値 (行) |
|---|---|---|---|---|---|
| imperative_merged | 1 | 2 | 4 | 9 | 89 |
| declarative_merged | 1 | 2 | 3 | 7 | 115 |
| imperative_closed | 1 | 1 | 3 | 5 | 23 |
| declarative_closed | 1 | 2 | 3 | 5 | 138 |

# 7. 議論

## 7.1 レビュー，変更時間への影響

RQ1 の結果から，命令的モジュールを使用した場合および使用しなかった場合の変更解決時間には顕著な差が見られなかった．特にマージされた PR はプロジェクトに変更を与えるが，各パーセンタイルでの時間差があまり見られなかった．また，レビュー時間に関しても同様に，マージされる PR において各パーセンタイルの値に顕著な差が見られなかった．これらの結果から，命令的モジュールの有無によるレビューや変更への時間的影響は少ないと考えられる．

クローズされた PR およびレビュースレッドは，命令的モジュールの使用の有無に関わらずマージされた PR に比べ長期化していたが，その差は顕著であり命令的モジュールの使用の有無とは別の要因がある可能性がある．実際に調査を行った PR のうち，長期間放置された PR に対しクローズを行うボットである stale [21] によってクローズされている PR が複数発見された．このような放置されている PR と，実際に議論が長期化した場合との区別を行っていないため，命令的モジュール有無による影響を調査できていない可能性がある．対策として，ボットにクローズされるような長期間放置されている PR を除外する，議論が活発に行われていない PR を除外する，リポジトリにおけるコアメンバーの PR を重視する等が考えられる．

## 7.2 変更規模への影響

RQ2 の結果から，命令的モジュールをファイルに含んでいる場合はそれらが変更箇所に含まれているかどうかに関わらず，PR 全体の変更行数を増大させていると考えられる．命令的モジュールを変更箇所に含む場合，PR 全体およびファイル単体での変更行数が増加しており，命令的モジュールによる変更規模の増加が考えられる．また，PR 中での変更行数はクローズされた PR よりもマージされた PR の方が多く，より活発に議論および変更が行われている可能性がある．

命令的モジュールを含む場合は同時に変更されたユニークなファイル数の増加が見られ，変更が必要なファイルが多くなる可能性が考えられる．ファイル数増加の理由として，命令的モジュールによる外部スクリプトの呼び出しが考えられる．実

際に命令的モジュールを含むいくつかのPR中の変更ファイルにPythonスクリプトの利用が見られた．しかし，本研究ではこれらの呼び出し関係の調査を行っておらず，命令的モジュールを用いてこれらのファイルが呼び出されているかどうか今後の調査が必要である．

## 7.3 研究目的に対する議論

RQ1の結果より命令的モジュールの使用が，第三者のレビューに必要な時間に与える影響は少ないと考えられる．またRQ2の結果より命令的モジュールの使用が，変更に必要な記述量や関連ファイルに影響を与える可能性が考えられる．

また，命令的モジュールを使用した場合にファイル数や記述量が増加したにも関わらず，レビューに必要な時間に差が見られなかったため，より効率的にレビューが可能であった可能性があると考えられる．理由としては，宣言的モジュールを使用した場合，ツールによってどのような命令が行われるか，ユーザが直接コードを確認しても不明であることが多く，実行内容の確認が容易ではないため，レビュー時に変更内容の検証が困難である場合があった可能性がある．

本研究では，命令的モジュールの使用は環境との兼ね合いや冪等性の観点から非推奨とされているが，実際にはレビュー時間には変化が見られず，変更規模に関してのみ影響が見られた．追加された行数，コミット数が増えるほどPRの評価までの時間が長期化するとされているが[19]，本研究の結果では，変更行数の増加が見られたがPRの評価までの時間は増大せず，ソースコードレビューの観点から，命令的モジュールの使用は問題ないが，変更行数は増大しているため，変更を行う開発者への負担は増加していると言える．

# 8. 妥当性への脅威

- PR のみ着目した事に対する妥当性

  本研究ではデータセットとして Ansible Galaxy にて公開されているロールから GitHub 上のリポジトリを取得したため，GitHub で最も一般的に用いられているレビューツールである PR に着目したが，他にも Phabricator [22]，Gerrit [23]，Atlassian Crucible [24] などのソースコードレビューツールが存在するため，それらが同様の結果を示すかどうか今後の調査が必要である．

- PR から抽出した内容に対する妥当性

  PR からそれらの所要時間や変更された行数を抽出したが，それらの値が開発者やレビュワーの技量によって左右される可能性がある [19, 25]．また，レビューの質問に対する回答が放置されている場合や，回答までの期間が異常に長い場合など，プロジェクトの品質にバラつきがあり，これらを同一に扱うべきか検討する必要がある．実際に第三者によって立てられたスレッドの内約半数で議論が行われていなかった．

- 命令型の判断に対する妥当性

  レビュー箇所に命令的モジュールを含むかどうかによって分類を行っていたが，それらが実際にそれらのモジュールを対象としていたか疑問の余地がある．前後のコードや指摘内容を吟味する事によってより正確な分類を行うことができると考える．しかし，それらの判断には 7,000 件を超えるレビュースレッドを確認する必要があり，場合によっては目視で確認するため，その実行には多大な労力が必要である．

- データセットに対する妥当性

  Ansible にのみ絞って調査を行ったため，一般的な結論を得るために他の構成管理ツール (Chef, Pupet, Terraform) でも同様の結果が得られるか確認する必要がある．また，GitHub 上で公開されているリポジトリからのみデータを収集したが，企業での実態が同様の性質を持つかどうか今後の調査が必要である．しかし，これらのデータは機密情報を多く含むため取得は困難である．

# 9. 結言

本研究では構成管理ツールでの命令型の使用が及ぼす PR への影響についてレビュー時間と変更規模の 2 点を調査した.

Ansible のリポジトリにおいて，命令的モジュールの使用によるレビュー時間への影響はそれほど大きなものではなく，変更が行われるファイル数及び行数に変化が見られた．マージされた PR にのみ着目した場合，レビュースレッドおよび PR そのものの所要時間に大きな差は見られず，実際の開発においてそれほど大きな影響ではないと言える．クローズされた PR ではそれまでに放置されていた可能性や，モジュールに関係のないところで時間がかかっていた可能性があるため一概に命令的モジュールが原因であるとは言えない．変更規模に関して，命令的モジュールを使用した場合変更行数，ファイル数ともに増加した．また，変更内容に命令的モジュールを含む場合それらの差がより増大した．これらの結果から，命令的モジュールを使用した場合でも変更規模の増加が生じても，レビューの際に問題となることは少なく，実際のソフトウェア開発現場において命令的モジュールの使用が問題とはならない可能性を示唆している.

# 謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学・人間科学系水野修教授ならびに崔 恩瀞助教に厚く御礼申し上げます．本報告書執筆にあたり貴重な助言を多数頂きました，本学本学設計工学専攻 西浦生成先輩，近藤将成先輩，情報工学専攻 舟山 優先輩，里形 洋道先輩，情報工学課程 中森 陸斗君，寺村英之君をはじめとする，ソフトウェア工学研究室の皆さん，学生生活を通じて著者の 支えとなった家族や友人に深く感謝致します.

# 参考文献


[1] J. Smeds, K. Nybom, and I. Porres, “Devops: a definition and perceived adoption impediments,” In Proceedings of the International conference on agile software development, pp.166–177, Springer, 2015.

[2] R. Jabbari, N. binAli, K. Petersen, and B. Tanveer, “What is devops? a systematic mapping study on definitions and practices,” In Proceedings of the Scientific Workshop Proceedings, pp.1–11, 2016.

[3] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, and D.A. Tamburri, “Devops: introducing infrastructure-as-code,” In Proceedings of the IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pp.497–498, IEEE, 2017.

[4] Puppet, “Powerful infrastructure automation and delivery,” `https://puppet.com`, （参照 2021-01-25）.

[5] ChefSoftware, Inc, “Chef: Enabling the coded enterprise through infrastructure, security and application automation,” `https://www.chef.io`, （参照 2021-01-25）.

[6] RedHat, Inc, “Ansible is simple it automation,” `https://www.ansible.com`, （参照 2021-01-25）.

[7] Northern.tech, Inc., “Cfengine,” `https://cfengine.com`, （参照 2021-01-25）.

[8] K. Torberntsson and Y. Rydin, “A study of configuration management systems: Solutions for deployment and configuration of software in a cloud environment,” 2014.

[9] A. Jacob, “Why startups need automated infrastructures,” `https://www.slideshare.net/adamhjk/why-startups-need-automated-infrastructures`, 2008.

[10] S. Nelson-Smith, Test-Driven Infrastructure with Chef: Bring Behavior-Driven Development to Infrastructure as Code, ” O’Reilly Media, Inc.”, 2013.

[11] “Declarative vs imperative: Devops done right — ubuntu,” https://ubuntu.com/blog/declarative-vs-imperative-devops-done-right.

[12] W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam, “Testing idempotence for infrastructure as code,” In Proceedings of the ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, pp.368–388, Springer, 2013.

[13] J. Gunawardena, “An introduction to idempotency,” Idempotency (Bristol, 1994), vol.11, pp.1–49, 1998.

[14] J.K. Baksalary and O.M. Baksalary, “Idempotency of linear combinations of two idempotent matrices,” Linear Algebra and its Applications, vol.321, no.1-3, pp.3–7, 2000.

[15] S. Kokuryo, M. Kondo, and O. Mizuno, “An empirical study of utilization of imperative modules in ansible,” In Proceedings of the IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), pp.442–449, IEEE, 2020.

[16] RedHat, Inc, “Ansible galaxy,” https://galaxy.ansible.com.

[17] RedHat, Inc, “community.aws.rds – create, delete, or modify amazon rds instances, rds snapshots, and related facts – exapmle,” https://docs.ansible.com/ansible/latest/collections/community/aws/rds_module.html, （参照 2021-01-25）.

[18] K. Morris, Infrastructure as Code, O’Reilly Media, 2020.

[19] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, “Wait for it: Determinants of pull request evaluation latency on github,” In Proceedings of the IEEE/ACM 12th working conference on mining software repositories, pp.367–371, IEEE, 2015.

[20] GitHub, Inc., “Github graphql api - github docs,” https://docs.github.com/en/graphql, （参照 2021-01-25）.

[21] GitHub, “Stale,” https://github.com/marketplace/stale.

[22] Phacility, Inc, “Phabricator,” https://www.phacility.com.

[23] Google, “Gerrit,” https://www.gerritcodereview.com.

[24] Atlassian, "Crucible," https://www.atlassian.com/ja/software/crucible.

[25] Y. Yu, H. Wang, G. Yin, and C.X. Ling, "Reviewer recommender of pull-requests in github," In Proceedings of the IEEE International Conference on Software Maintenance and Evolution, pp.609–612, IEEE, 2014.