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

頭川 剛幸
京都工芸繊維大学 工芸科学部
t-zukawa@se.is.kit.ac.jp

近藤 将成
九州大学 システム情報科学研究院
kondo@ait.kyushu-u.ac.jp

西浦 生成
京都工芸繊維大学 大学院工芸科学研究科
k-nishiura@se.is.kit.ac.jp

崔 恩瀞
京都工芸繊維大学
情報工学・人間科学系
echoi@kit.ac.jp

水野 修
京都工芸繊維大学
情報工学・人間科学系
o-mizuno@kit.ac.jp

要旨

近年複雑化する IT インフラに対応するため，インフラをコードで定義する構成管理ツールが用いられている．表現方法として計算機の状態を定義する宣言型と，スクリプトを記述する命令型がある．命令型は，環境の再現性や堅牢性の脅威になるとされているが，ソフトウェア開発にどのような影響を与えているのか明らかになっていない．本研究では，構成管理ツールの実行の最小単位であるモジュールから，命令的操作を行うモジュール（以下命令的モジュール）によって，コードレビューの時間が増大すると仮説を立てた．命令的モジュール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，レビュースレッドの所要時間，変更行数およびファイル数を調査した．その結果，命令的モジュールの有無によって，マージされた PR およびレビュースレッドの所要時間に差は見られなかった．一方，変更行数およびファイル数に増加が見られた．これらの結果から，命令的モジュールを使用することで変更規模を増大させ，レビュー時の確認項目が増大する可能性があるが，レビュー時間は変化しなかったため，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があることがわかった．

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 Infrastructure as Code（IaC） というプラクティスが存在する [1–3]．IaC は，運用基盤の構成をコードによって記述し．開発プロセスのプラクティスを運用にも利用可能にする．これにより，プロビジョニング，構成，デプロイを自動化することができ，運用，開発規模の増大に対応することができる．

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 [4]．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 [5, 6]．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（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．冪等性を持たないモジュール（以下非冪等モジュール）を用いる場合，ユーザがその実行に冪等性をもたせる必要がある．実際，我々研究グループの先行研究 [7] から Ansible [8] 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，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可能性がある．

IaC にはシステムの表現方法として，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．命令型の使用は，環境再現性や堅牢性の脅威となると考える．環境再現性への影響として，命令型は非冪等な操作となる可能性が高く，開発者は利用するコマンドが与える影響を把握した上で，冪等な操作となるように，実行条件を与える必要がある．また，外部スクリプトの実行が可能であり，その変更による環境への影響を把握する必要がある．堅牢性への影響として，複雑な命令を利用した場合に，可読性への影響や，操作に対する変更，宣言型への変換などが容易で無い可能性がある．これらの問題点から，命令型のレビュー時に，宣言型に比べ実行条件や結果の調査が必要になり，利用するスクリプト等についての保守や，コマンドに対する理解が求められる．

そこで，本研究では命令型の使用によってレビュー時に，変更内容の妥当性の確認に掛かる時間や，変更箇所が増大すると仮説を立てて調査を行った．今回は先行研究 [7] 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[1]からリポジトリのリストを抽出し，GitHub で公開されているリポジトリからプルリクエスト（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仮説を立てた．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，マージされた PR において命令的モジュールを使用する場合，最大値を除き各パーセンタイルの値が増加する傾向にあった．またレビュースレッドの場合では，マージ，クローズのどちらも大きな差が見られず，一貫した結果が得られなかった．

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，変更規模が大きくなると仮説を立てた．命令的モジュールを含むマージされた PR では変更行数，変更ファイル数それぞれに増加が見られ，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．

RQ1 および RQ2 の結果から，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がある．

以降の本稿の構成を紹介する．第 2 章では，簡単な例を示し本研究の必要性を述べる．第 3 章では，研究背景である IaC および PR について説明する．第 4 章では，調査の目的と方法について説明する．第 5 章では，使用したデータセットについて説明する．第 6 章では，それぞれの研究設問に対する動機，調査手順および結果をまとめる．第 7 章では，結果に対し議論を行う．第 8 章では，本研究の結論を述べる．

2. 本研究の着想に至った動機の例

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 1 および 2 に示す．例 1 は Ansible の公式ドキュメントに掲載されている例 [9] であり，例 2 は筆者が例 1 を命令的モジュールを使用するように変更したものである．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，例 1 の `community.aws.rds` より下（4 行目以降）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

[1] `https://galaxy.ansible.com`

ソースコード 1. 宣言的モジュールを用いたデータベース作成例 [9]

```
---
- name: Basic mysql provisioning example
  community.aws.rds:
    command: create
    instance_name: new-database
    db_engine: MySQL
    size: 10
    instance_type: db.m1.small
    user_name: mysql_admin
    password: 1nsecure
```

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．

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 [10].

ソースコード 2. 命令的モジュールを用いたデータベース作成例

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

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 Hummer らの研究 [4] 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（OSS）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 [7] 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[2]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. GitHub におけるプルリクエスト（PR）

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リポジトリのコアメンバー等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 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いる [11]．Yue らが調査した因子の例としては，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．

4. 調査概要

先行研究 [7] 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と報告されている．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．命令型の使用は冪等性のための条件分岐や，実行内容の把握の煩雑化が，レビューに掛かる時間が増大させると仮説を立て調査を行った．

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．

RQ の調査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，ファイル単体での変更行数および変更ファイル数を用いた．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

[2] https://galaxy.ansible.com

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. 調査対象のデータセット

5.1. データセットの概要

本研究では先行研究 [7] 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 [12] 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．また，それらを図 1 に示す．

1. PR の状態，すなわちマージまたはクローズ
2. PR の開始時間及びマージまたはクローズされた時間
3. 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
4. PR 全体及びファイル単体での追加，削除された行数
5. PR 中に変更が行われたファイル群
6. 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 [7] 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，

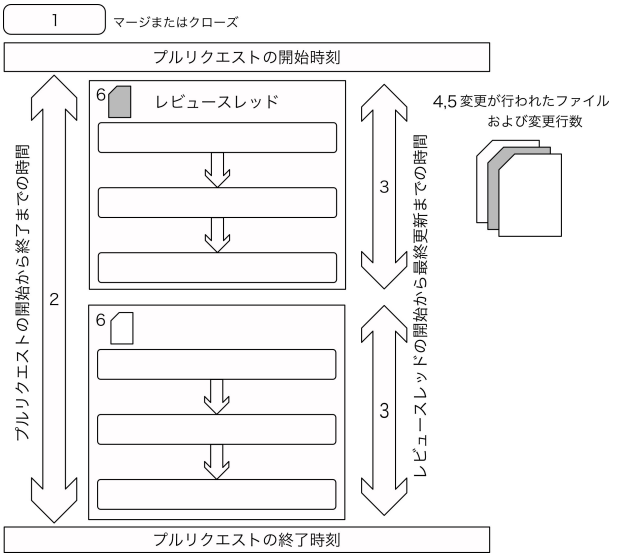


図 1. PR の持つ要素

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．

6. 研究設問の調査手順と結果

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

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．命令的モジュールの使用が実際に影響を与えているか確認する．

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．また，レビュースレッド中に議論が行われていないものに関しては，集計の対象として含めない．調査の際，以下の場合に分けてそれぞれの時間を調べた．

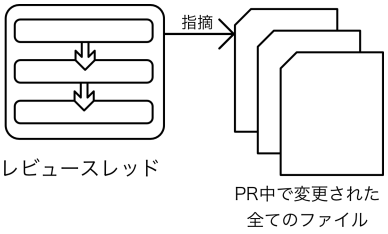


図 2. 命令的モジュールを含まない PR のレビュースレッド

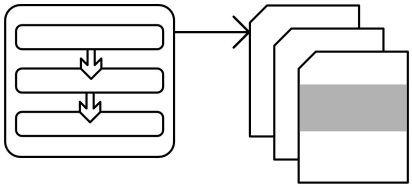


図 3. 命令的モジュールを含む PR のレビュースレッド

- 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 2，3 および 4 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う．よって，図 2 は命令的モジュールを含まない PR，図 3 は命令的モジュールを含む PR となる．

条件 2 では，レビュースレッドの指摘箇所が命令的モジュールを含むかによって判断する．図 3 では指摘箇所に命令的モジュールを含まない PR，図 4 では指摘箇所に命令的モジュールを含む PR とした．また，図中のグレーの箇所はファイルに含まれる命令的モジュールを示している．

6.1.3 結果

条件 1 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図 5 および 6 に示す（以降の図に含まれるラベル IM，DM，IC，DC はそれぞれ，命令型を含むマージ済のもの，命令型を含まないマージ済のもの，命令型を含むクローズされたもの，命令型を含まないクローズされたものを示す）．ここで，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．

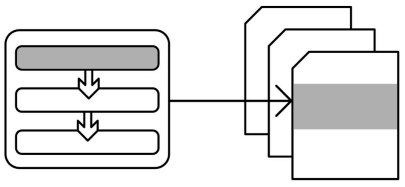


図 4. 命令的モジュールを対象とする PR のレビュースレッド

次に，条件 2 によって分類し，同様に調査を行った結果を図 7 および 8 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．PR の所要時間では，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．レビュースレッドでの比較では，マージ状態および，クローズ状態の PR どちらも一貫した結果が得られなかった．

この RQ1 の調査によって，PR およびレビュースレッドの所要時間は，命令的モジュールの使用により長期化する場合があることがわかったが，それらは実用上で問題とするほどの差では無かった．よって命令的モジュールの使用は，PR およびレビュースレッドの所要時間に影響を与えない可能性がある．

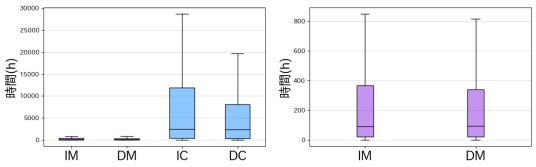


図 5. 条件 1, 変更解決時間

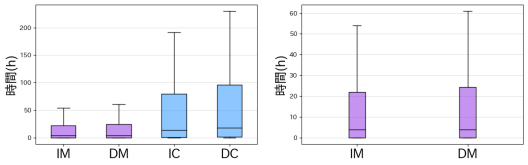


図 6. 条件 1, レビュー時間

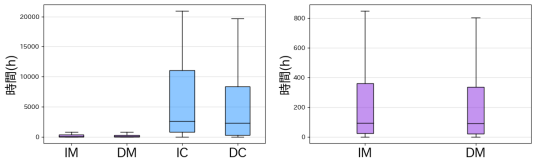


図 7. 条件 2, 変更解決時間

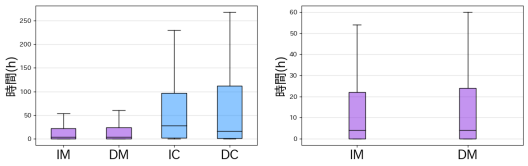


図 8. 条件 2, レビュー時間

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，その分制御が複雑になり，変更が必要な部分が増加すると考えた．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．これらの値をファイルごとに合計したものをファイル変更行数，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図 9a，9b および 9c 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ファイルは 874 件であり，全体の約 8.0%（874/10,978）であった．クローズされた PR 中の総変更行数の 25 パーセンタイル（図 9a）を除き，全ての四分位数で，命令的モジュールを含む場合に値が増加した．

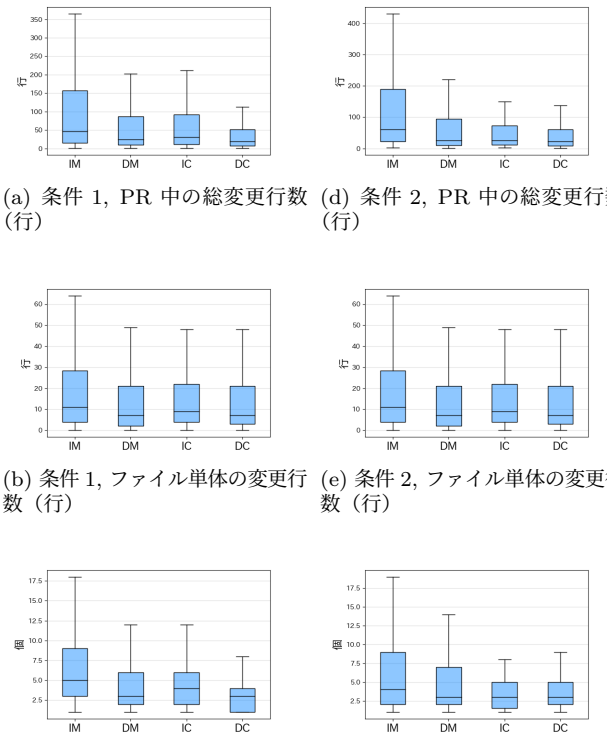
変更箇所での命令的モジュールの有無によって分類した場合における，PR 中の総変更行数，ファイル変更行数および変更ファイル数に関する調査結果を図 9d，9e および 9f に示す．変更箇所に命令的モジュールを含むファイル数は 287 件であり全体の約 2.6%（287/10,978）であった．PR での総変更行数およびファイル単体の変更行数は，最大値を除き，命令的モジュールを使用した場合に変更行数が増加した．変更が行われたファイル数は，クローズされた PR の 25 パーセンタイルを除き，命令的モジュールを含む場合に四分位数の値が増加もしくは同じ値となった．

この RQ2 の調査によって，命令的モジュールの使用による PR の総変更行数，ファイル中の変更行数およびファイル数にそれぞれ増加が見られたため，PR 中の変更規模に差が見られた．よって命令的モジュールの使用は，PR の変更規模を増加させる可能性がある．

7. 議論

7.1. レビュー，変更時間への影響

6.1.3 節で説明した 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



(a) 条件 1, PR 中の総変更行数（行）
(d) 条件 2, PR 中の総変更行数（行）
(b) 条件 1, ファイル単体の変更行数（行）
(e) 条件 2, ファイル単体の変更行数（行）
(c) 条件 1, 変更ファイル数（個）
(f) 条件 2, 変更ファイル数（個）

図 9. 変更規模の四分位範囲

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[3]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．

[3]https://github.com/marketplace/stale

7.2. 変更規模への影響

6.2.3 節で説明した 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ことがわかった．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ファイル数が増加する場合があり，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 やシェルスクリプトの存在が確認された．

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，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．

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できた可能性がある．理由としては，宣言的モジュールを使用した場合，構成管理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．先行研究 [11] では追加された行数，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，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しなかった．そのためソースコードレビューにかかる負担の観点から

は，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．

7.4. 妥当性への脅威

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[4]，Gerrit[5]，Atlassian Crucible[6]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 [11, 13]．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．

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，レビュー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られる．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．

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．

8. 結言

本研究では，IaC に関するソースコード開発現場において，命令的モジュールの使用がどのような影響を与えるか調査した．OSS の開発プロセスである PR に着目し，PR が評価されるまでの時間やレビュースレッドの所要時間，変更行数や変更ファイル数の比較を行い，命令的モジュールの有無によって差が見られるか比較を行った．

調査結果から，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がある．

本研究では，レビュー時間と変更規模についての生産性に関わる調査を行ったが，命令的モジュールの使用による冪等性の実現性や，ヒューマンエラー率などのソフトウェア品質についての影響は明らかになっていない．そのため，今後の課題として故障間隔や修復時間などの品質面についての調査を行った上で，命令的モジュールの使用がもたらす課題についてさらなる研究を行いたい．

謝辞

本研究の一部は，JSPS 科研費 JP19J23477, JP19K20240, JP18H04094 の助成を受けました．

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

[4] https://www.phacility.com
[5] https://www.gerritcodereview.com
[6] https://www.atlassian.com/ja/software/crucible

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] 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.

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

[6] 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.

[7] 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.

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

[9] 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).

[10] K. Morris, Infrastructure as Code, O'Reilly Media, 2020.

[11] 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.

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

[13] 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.