修　士　論　文

題　　目　　構成管理ツールにおける
　　幂等性を持たないモジュールの
　　利用形態解析

主任指導教員　　水野　修　教授

指導教員　　崔　恩瀞　助教

京都工芸繊維大学 大学院工芸科学研究科

情報工学専攻

学生番号　　18622021

氏　　名　　國領　正真

令和２年２月10日提出

様式 2 号

学位論文内容の要旨（和文）

令和 2 年 2 月 10 日

京都工芸繊維大学
大学院工芸科学研究科長　殿

大学院工芸科学研究科 情報工学専攻
平成 30 年入学
学生番号　18622021
氏　名　國領　正真　㊞

（主任指導教員　水野　修　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

構成管理ツールにおける冪等性を持たないモジュールの利用形態解析

2.　論文内容の要旨（400 字程度）

近年，複雑化・巨大化する運用のための基盤環境構築および管理において，属人性の排除や効率化のため，構成管理ツールが採用されている．構成管理ツールでは，再現可能なデプロイのため操作の冪等性が重要視される．一方で，ユーザ定義のスクリプトを対象の計算機上で直接実行する，非冪等な操作をしうる命令的なモジュール（非冪等モジュール）は頻繁に利用されている．そのようなモジュールの多用は開発者らから問題視されているにもかかわらず，それらがなぜ・どのように利用されるかは未だ明らかでない．

本研究では構成管理ツール Ansible に着目し，非冪等モジュールについて，使用される割合や冪等性を保つための努力，それらを代替可能なモジュールの有無について調査した．

非冪等モジュールは主に Ansible によってサポートされていない操作のために使用され，全体の約 45%には代替可能な別のモジュールが存在した．これらの結果は，構成管理のためのスクリプトの品質モデルにおける非冪等モジュールの使用について，改善の余地があることを示唆する．

An Empirical Study of Utilization of Imperative Modules in A Configuration Management Tool

2020 18622021 *KOKURYO Shoma*

Abstract

In recent years, a configuration management tool is adopted to manage a complicated and huge usable systems such as bare-metal servers, cloud computing resources and our personal computers. Such a tool makes the operations to deploy services more efficient and eliminates dependencies on the specific system operators. The operations are required to be idempotent for reproducible deployment. However, the *imperative modules* whose operations may not be idempotent (non-idempotent modules) are used frequently to execute user-defined scripts on the target system; it is unclear why and how they are used, though using them frequently is said to be a bad practice by developers.

We studied the following three questions in a configuration management tool, Ansible: (1) How much non-idempotent modules are used? (2) How do developers keep their operations idempotent? (3) Can we replace non-idempotent modules with other modules?

Non-idempotent modules are mainly used to perform operations that are not supported by Ansible, and about 45% of non-idempotent modules are replaceable by other modules; the replaceable modules might be idempotent. We therefore recommend developers to look at replaceable modules before using non-idempotent modules since replaceable modules might make their operations idempotent.

目　次

1. 緒言

近年，ソフトウェアの迅速な不具合修正と機能追加についての要望は，ますます高まっている．開発者およびそれらの運用者は，そのための様々なプラクティスを考案してきた．ソフトウェアの開発から運用までの時間及び労力，開発者と運用者の間にある技術的・組織的な距離を削減するため，DevOps として知られているソフトウェア開発手法のパラダイムが存在する [1–4]．その一部として，サービス提供のための基盤の構成を機械的に読み取り可能な文書として記述し管理する，Infrastructure as Code（以下 IaC）と呼ばれるプラクティスがある [3]．これは複雑化・巨大化する運用のための基盤環境の構築および管理において，属人性の排除および効率化に寄与し，様々な企業において実際に導入されている [5]．

IaC を実践するために，様々なツールが開発されている．クラウドの仮想マシンなどを管理する Terraform [6] や Pulumi [7]，他にサーバの設定やミドルウェアのデプロイなどを行う Ansible [8]，Puppet [9]，Chef [10] などの構成管理ツールがあげられる．これらによる構成管理の自動化は再現可能であることが期待される．つまり，システムを任意の状態から望ましい状態へと変更できることが期待される．このような再現可能で堅牢な自動化の基礎として，冪等性という概念が知られている [11]．冪等な操作は複数回の実行において常に同じ結果が得られる．例えば，任意の整数と 0 の乗算や，ある整数の絶対値を求める操作などは冪等である．構成管理ツールは冪等な操作を実現するためのいくつかのモジュールを提供しており，開発者らはこれを用いて構成をコードとして記述する．また，非冪等になり得る機能も存在し，利用されている．例えば，定義されたコマンドを対象ホスト上で直接実行する命令的なモジュールがあげられる．これは頻繁に利用されていることが知られている [12]．命令的なモジュールの利用は多くの開発者らからバッドプラクティスであると述べられており，その乱用は問題視されている [13–15]．しかし，それらが実際に，なぜ利用されているのか，どのように利用されているかについては，まだ明らかとなっていない．

本研究では近年利用者の多い Ansible に着目し，実際に利用されているコードを用いて，命令的なモジュールが，なぜ，そしてどのように利用されているのかを明らかにする．ユーザが任意のシェルスクリプトを実行させることができる `command`・

shell・raw・script の 4 つの自明に命令的かつ非冪等な操作をしうるモジュール（以下，非冪等モジュール）を対象とした．データセットとして，一連の機能を提供する単位である Ansible ロールを，ユーザが自由に共有する Web サイトである Ansible Galaxy から各ロールの GitHub リポジトリを取得し，以下の 4 つの研究設問について調査した．

RQ1）Ansible ロールはどの程度の割合で非冪等モジュールを含むか

過去一年間に一度以上更新されている Ansible ロールのうち，ダウンロード数上位 10% 以上のもの 884 個を取得した．合計で 2,656 個の非冪等モジュールを使用するタスクを抽出した．これは全タスクの約 13.6%に相当する．また，非冪等モジュールを 1 つ以上使用するロールは 419 個あり，1 つ以上のタスクを持つロールのうち約 49.3%を占めた．

RQ2）それらのモジュールを利用したタスクで頻繁に行われる操作は何か

RQ1 で取得した 2,656 個のタスクについて，その処理内容を 14 個のカテゴリへと目視で分類した．最も頻繁に見られた操作はアプリケーション固有の操作（約 24.2%）であり，次いでファイルに関連した操作（約 17.5%），OS の設定やサービスに関連する操作（約 15.7%），パッケージに関連する操作（約 11.7%）などが見られた．

RQ3）冪等性を保つためにどの程度努力がなされているか

Ansible のタスクはその実行に条件を設定できる．非冪等モジュールを利用するタスクでは約 51.32%が，それ以外のタスクでも約 46.76%がその実行について何らかの条件を持つことがわかった．また，非冪等モジュールを利用したタスクでは約 50.75%が実行後の出力結果からその操作が変更を引き起こしたかを判定していたが，それ以外のタスクでは約 1.18%しか判定していなかった．

RQ4）代替可能なモジュールは存在するか

処理内容を目視で確認し，約 45.0%のタスクについて代替可能であると判断した．ただし，アプリケーション固有の操作については約 23.1%のみが代替可能であるのに対し，ファイルに関連した操作では約 84.5%が代替可能であった．また，代替不可能なもののうち約 75.3%が Ansible のサポートしていない操作であるためであった．

これらの結果は，IaC スクリプトの品質モデルにおいて課題となっている命令的なモジュールの利用について，改善の可能性を示唆する．また，この成果を流用し，非冪等モジュールを利用して記述しているタスクに対し，代替可能なモジュールを推薦するツールの実装なども考えられる．

本論文では，2 章で必要となる背景知識について言及し，3 章で研究設問およびそれに伴う実験について述べ，4 章で結果を示す．5 章にてそれらの結果から考察し，6 章で先行研究との関連性について述べた後，7 章で妥当性への脅威について言及し，7 章で結言とする．

2. 背景

2.1 DevOps

ソフトウェアを開発し，それを元にサービスなどを提供する企業では，主にソフトウェアの開発チームと運用チームに組織が分けられてきた．それら 2 つの組織は分離されており，その間には大きな障壁があると言われている．DevOps はそういった組織における分離と障壁を解決するための開発手法のパラダイムであり，その目的は迅速かつ頻繁にソフトウェアをリリースすることにある [1–3]．これは文化的な側面と，技術的な側面の両方を持つ．特に技術的な側面では，ソフトウェアのデプロイにおける一部または完全な自動化が特徴である．例えば，ソフトウェアの自動テスト等を行う継続的インテグレーションや，デプロイを自動化する継続的デリバリーなど，複数の概念の組み合わせによって成り立つ．このような自動化のため，開発チームと運用チームは密接に連携する必要がある．

2.1.1 Infrastructure as Code（IaC）

DevOps の実現にあたり，IaC は重要な要素の 1 つである．頻繁なソフトウェアのリリースのため，その開発サイクルもまた高速化する．それに伴い，その実行環境，例えばベアメタルサーバーやクラウド上の仮想マシン，およびその構成やミドルウェアについても継続的な改善が求められる．IaC はこうした要望に応えるための概念であり，ソフトウェア開発におけるプラクティスを用いることで，運用のための操作の高速化および再現性向上を目的とする [3]．

具体的には，ソフトウェアの実行環境整備と関連する様々なツールに，ソースコードの概念を導入する．それらのソースコードは対象の実行環境を表すものであり，その定義に従って環境構築が行われる．ソースコードは単なるテキストであるため，従来ソフトウェアのソースコードに対して適用されてきたバージョン管理機能や，静的解析などのプラクティスを再利用できる．

2.2 冪等性

ある操作を 1 回実行した場合にも，複数回実行した場合にも結果が同じであることを指す概念である [11]．例えば，任意の整数と 0 の乗算や，ある整数の絶対値の計算は冪等な操作である．

2.3 構成管理ツール

IaC を実現するためのツールの一種である．主にサーバ計算機や仮想マシンに対する環境構築を行うツールに対する名称である．ミドルウェアの設定や，OS の設定などの他に，仮想マシン自体の構築などを含む場合もある．代表的なツールとして Ansible，Chef，Puppet などがある．これらのツールはいずれも 2.2 節で述べた冪等性を考慮し，再現可能な自動化を目標としている．

2.4 Ansible

本研究において対象とする構成管理ツールであり，RedHat, Inc. によって開発されている Python 製のオープンソースソフトウェアである．構築する対象の計算機に独自のエージェントソフトウェア (注 1) を必要とせず，UNIX 系 OS の多くが標準でインストールしている Python をエージェントとして使用することが特徴としてあげられる．

2.4.1 用語

Ansible を構成するいくつかの概念について述べる．これは Ansible の概念すべてを網羅する物ではないが，この論文を理解するには十分なものである．

(1) モジュール

Ansible における最小の単位．1 つのモジュールがある操作を司り，特定の種類のデータベースでのユーザ管理から，特定のネットワークデバイスでの VLAN イン

(注 1): 構成管理ツールによる操作を対象の計算機へと仲介するためのソフトウェア

ターフェースの管理まで，様々な用途がある [16]．執筆時点での最新安定版である Ansible 2.9.4 の時点で非推奨のものを含めて 3,387 個存在する [17]．

(2)　タスク

前述のモジュールを用いて，ある操作を実行する単位 [16]．

(3)　ロール

Ansible 1.2 から導入された，一連のタスクを再利用するための単位．例えば管理のためのユーザを追加し，必要な権限を与えるなど．予め規定されたディレクトリ構造へ適切にファイルを配置することで，ロールとして定義できる [18]．

(4)　プレイブック

前述した Ansible のタスクおよびロールを用いて，ある管理対象の計算機に対し，その構成を記述する単位．タスク，またはロールの順序付きリストである．それらのタスクだけでなく適用される変数が同時に定義される場合もある．

2.4.2　記法

YAML というデータ形式を使用する [19]．

YAML 自体はプログラミング言語ではないため，その記述内容はプログラム的な制御構文を持たない．しかし，Ansible では Python のテンプレートエンジンである Jinja2 [20] を使用することで，動的に値を埋め込むこと（以降変数埋め込み）が可能である．変数名を bar とすると，bar の内容を埋め込みたい箇所へ`"{{ bar }}"` のように`{{`と`}}`で変数を囲んで配置することで，実行時に配置した箇所へ変数 bar の内容を埋め込むことができる．ソースコード 2.1 に例を示す．

また，Ansible の文法に沿って要素を記述することで，条件分岐やループ制御なども可能となる．条件分岐を伴うタスクの例をソースコード 2.2 に，ループ制御を伴うタスクの例をソースコード 2.3 に示す．

ソースコード **2.1**　**Jinja2** の文法に則った変数埋め込みの例

```
foo: "{{ bar }}"

baz:
  - "{{ foo }}"
  - "{{ bar }}"
```

ソースコード **2.2**　条件分岐を伴うタスクの例 **[21]**

```
tasks:
  - name: "shut down Debian flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_facts['os_family'] == "Debian"
```

ソースコード **2.3**　ループ制御を伴うタスクの例 **[22]**

```
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  loop:
    - testuser1
    - testuser2
```

2.4.3 宣言的実装と命令的実装

Ansible におけるモジュールは，宣言的な API を持つものと，命令的な API を持つものに分けられる．ただし，Ansible では公式にこれらのモジュールを区別しているわけではないため，本研究では `command`・`shell`・`raw`・`script` の 4 つのモジュール（以下非冪等モジュール）を命令的な API を持つモジュールとして使用する．

宣言的な API とは，対象の状態の宣言のみを行い，その状態に至るまでの手順はツールによって決定されるものを指す．例えば，Hello という行を含むあるファイル`/tmp/hello.txt` を，宣言的 API を持つ `file` モジュールを用いて作成するタスクは，ソースコード 2.4 のように記述できる．このとき，そのファイルが存在しない場合には Hello と書き込まれたファイルが作成される．既にファイルが存在する場合，Hello という行が無ければ新たに書き込みが行われ，既に該当する行があれば，単にアクセス時間が更新される．

これに対し，命令的な API とは，ユーザによって記述された具体的な操作が対象へ直接実行されるものを指す．例えば，指定されたスクリプトを直接実行する `shell` モジュールを用いて，前述と同様のファイル`/tmp/hello.txt` を作成するタスクはソースコード 2.5 のように記述できる．

宣言的なモジュールは，モジュールがその冪等性を担保する場合が多く，ユーザはそれがどのように実現されるかを気にする必要が無い．それに対し，命令的なモジュールはユーザ自身が冪等なスクリプトを記述する必要がある．冪等性を担保することはしばしば容易でなく，変更を伴うスクリプトは望まない状態を導く可能性が指摘されている [12–15]．

2.4.4 各タスクの実行結果

全てのタスクは，その実行結果を対象と結果の組で表す．取り得る結果は表 2.1 に示す 7 通りである．仮に実行されるタスクが全て冪等なモジュールから構成され，ある 1 つの対象 C へそれらが適用された場合を考える．ただし，C における実行ではこれらのタスクは全てエラーを生じず，その実行条件を満たすものとする．このとき，1 回目の実行では `changed` となったタスクが観測されたとしても，2 回目以降の実行では全て `ok` となることが期待される．

ソースコード **2.4**　ファイルの存在を宣言的に定義する例

```
1 - name: /tmp/hello.txt exists
2   lineinfile:
3     path: /tmp/hello.txt
4     line: Hello
5     create: yes
```

ソースコード **2.5**　ファイルの存在を命令的に定義する例

```
1 - name: /tmp/hello.txt exists
2   shell: |
3     if [ ! -f /tmp/hello.txt ]; then
4       echo Hello > /tmp/hello.txt
5     elif [ grep Hello /tmp/hello.txt > /dev/null ]; then
6       echo Hello >> /tmp/hello.txt
7     fi
```

非冪等モジュールはその性質上，複数回実行しても ok とはならない．Ansible はその実行が失敗しないまたはスキップされない限り，changed として報告する．

2.4.5 Ansible Galaxy

Ansible Galaxy (注 2)とは，コミュニティによって開発された Ansible ロールを検索・ダウンロード・共有するための無料の Web サイトである [23]．ユーザは自由にロールをアップロード，またはダウンロードできる．Ansible Galaxy 上の全ての Ansible ロールは GitHub (注 3)上のリポジトリと紐付いており，Web API を通じてそれらを取得できる．

(注 2): https://galaxy.ansible.com/
(注 3): https://github.com/

表 2.1　タスクの取り得る結果

結果	詳細
ok	定義された状態に達しており，変更は生じなかった
changed	定義された状態に達しておらず，変更が生じた
failed	何らかのエラーによって実行に失敗した
rescued	エラーに対する回復措置等のタスクへフォールバックされた
ignored	何らかのエラーが生じたが，無視された
skipped	実行の条件を満たさなかったため，実行されなかった
unreacheable	対象への接続に失敗した

3. 実験準備

3.1 データセットの取得方法

全ての研究設問に対する実験で利用するデータは，galaxy_crawler (注 4)を用いて，Ansible Galaxy の Web API から取得した Ansible ロールについての情報を元に，実際のソースコードを GitHub から入手した．galaxy_crawler は Ansible Galaxy の Web API から取得した Ansible ロールについての情報を PostgreSQL データベースへと格納する．そのように得られた情報から，以下の条件で取得するロールを選択する．

- 過去一年以内に更新がある
- 上記の条件を満たすもののうち，ダウンロード数の上位 10%に相当する

選択された Ansible ロールの GitHub リポジトリの URL を取得し，`git clone` コマンドを用いてリポジトリをダウンロードする．

3.2 取得したデータセットの概要

2019 年 10 月 22 日にデータセットを取得した．Web API からは 21,967 個のロールの情報，及びそのリポジトリの情報が得られた．また，2018 年 10 月 22 日 0 時 0 分 0 秒から，データセットの取得が完了した 2019 年 10 月 22 日 19 時 58 分 00 秒までの間に，1 回以上更新されたロールは 8,832 個であった．その中からダウンロード数上位 10%に当たる 884 個を今回の実験で用いるデータセットとした．

884 個のロールから，865 個のリポジトリが得られた．複数の Ansible ロールを単一のリポジトリで管理する方法が存在するため，リポジトリ数は必ずしも Ansible ロールの数とは一致しない．また，ロールの情報の最終更新時点からリポジトリの取得までの期間に，リポジトリの移転，削除もしくは非公開化により取得できなかったものが 1 つあったため，最終的には 864 個のリポジトリの取得に成功した．

(注 4): `https://github.com/pddg/galaxy_crawler`

4. 実験および結果

4.1 研究設問

本研究の目的を達成するため，4 つの研究設問を設け，それぞれについて実験および回答する．

RQ1 **Ansible ロールはどの程度の割合で非冪等モジュールを含むか**

動機：Chef や Puppet など他の IaC ツールにおいて，非冪等モジュールは頻繁に利用されていることが報告されているが，Ansible でも同様であるか確認する．

RQ2 **非冪等モジュールを利用したタスクで頻繁に行われる操作は何か**

動機：先行研究において，非冪等モジュールが使用されていることは報告されているが，どのような操作が行われるかについては言及されていない．非冪等モジュールに対する需要を調査することで，既存の他のモジュール等で網羅されていない可能性のある操作がどのようなものであるか明らかにする．

RQ3 **冪等性を保つためにどの程度努力がなされているか**

動機：非冪等モジュールによる操作は，その使用者自身が冪等性を担保する必要があるが，実際にそのような努力が行われているかは明らかでない．各タスクの持つ実行条件を調べることで，実際にそのような努力が行われているかを調べる．

RQ4 **代替可能なモジュールは存在するか**

動機：理想的には，非冪等モジュールの利用は全て排除されるべきである．しかし，その利用が避けられない場合などが存在する場合もあり得る．代替可能なものがどの程度含まれるかを明らかにすることで，IaC スクリプトの改善の余地について調査する．

4.2 RQ1）Ansible ロールはどの程度の割合で非冪等モジュールを含むか

4.2.1 実験手法

Ansible ロールは複数のタスクからなり，それらのタスクはロールのリポジトリ直下の `tasks` および `handlers` ディレクトリ内に YAML ファイルとして配置される．各ディレクトリ内の YAML ファイルを再帰的に探索し，全ての YAML ファイルをパースする．このとき，Ansible の変数埋め込みを解決せず，YAML 1.1 の仕様に従ってパースする [19]．例えばソースコード 4.1 では，合計 4 つのタスクが検出される．また，Ansible には `block` という，複数のタスクをまとめる記法が存在する．これは入れ子にできるため，`block` 内を再帰的に探索し，全てのタスクを抽出する．

全てのタスクを抽出した後，使用されているモジュールを識別する．ソースコード 4.1 では，4 つのタスクから，2 つの `command` モジュールが検出される．全てのロールについてこれを行い，非冪等モジュールの使用頻度を取得する．

4.2.2 結果

884 個の Ansible ロールのうち，848 個についてタスクの抽出に成功した．抽出された合計 19,542 個のタスクのうち，非冪等モジュールを使用したタスクは 2,656 個検出された．これはタスク全体のうち約 13.59%に相当する．非冪等モジュールの利用数の各モジュールごとの内訳は表 4.1 の通りである．

また，非冪等モジュールを利用するタスクを 1 つ以上含むロールの数は表 4.2 の通りである．Ansible ロールの約 49.29%は非冪等モジュールを少なくとも 1 つ以上を使用していた．

非冪等モジュールを利用するロールごとに，それらをいくつ利用しているかを調べたところ偏りが見られたため，利用数の上位 5 件について表 4.3 に示す．

ソースコード 4.1　Ansible のコードの例 [24]

```
---
- name: Download daemonize archive.
  get_url:
    url: "https://github.com/bmc/daemonize/archive/release-{{
       daemonize_version }}.tar.gz"
    dest: "{{ workspace }}/daemonize-{{ daemonize_version }}.tar.gz"

- name: Expand daemonize archive.
  unarchive:
    src: "{{ workspace }}/daemonize-{{ daemonize_version }}.tar.gz"
    dest: "{{ workspace }}"
    creates: "{{ workspace }}/daemonize-release-{{ daemonize_version }}/
       INSTALL"
    copy: no

- name: Check if daemonize is installed.
  command: which daemonize
  changed_when: false
  failed_when: false
  register: daemonize_installed

- name: Build daemonize.
  command: >
    {{ item }}
    chdir={{ workspace }}/daemonize-release-{{ daemonize_version }}
  when: daemonize_installed.rc != 0
  with_items:
    - "./configure --prefix={{ daemonize_install_path }}"
    - make
    - make install
  become: yes
```

表 4.1　非冪等モジュールの利用数

モジュール名	使用されている数	割合（%）
command	1,608	60.54
shell	905	34.07
raw	114	4.29
script	29	1.09

表 **4.2**　非冪等モジュールを **1** つ以上使用するロールの数と割合

モジュール名	ロール数	割合（%）
command	267	31.49
shell	171	20.16
raw	15	1.77
script	7	0.83
合計	418	49.29

表 **4.3**　ロールごとの非冪等モジュールの使用数とその割合

ロール名	非冪等モジュール数	割合（%）
MindPointGroup.RHEL7-CIS	471	17.73
florianutz.Ubuntu1604-CIS	99	3.73
oVirt.hosted_engine_setup	93	3.50
anthcourtney.cis-amazon-linux	79	2.97
ceph.ceph_handler	44	1.65

4.3 RQ2）それらのモジュールを利用したタスクで頻繁に行われる操作は何か

4.3.1 実験手法

RQ1 で抽出されたタスクについて，目視で確認し，表 4.4 のカテゴリへ分類する．分類する際にはそのタスク自体の他に周辺のコメントおよび前後のタスクを参照するため，そのタスクを含むファイル全体を目視で確認する．また，複数のコマンドが記述されている場合，そのコードが実現したい操作全体から分類するカテゴリを決定する．

加えて，その処理内容に応じて「読み込みのみ」·「変更を伴う」·「判断できない」の 3 種類に分類する．ただし，処理内容が変数で表され容易には推測できないもの，スクリプトの実行などで実際に実行される処理が不明であるものなどは，全て「判断できない」として分類する．複数のコマンドが記述されている場合，いずれの処理も読み込みのみの場合に限り,「読み込みのみ」として分類する．

この分類は，研究室環境における計算機管理等において Ansible の使用歴 3 年の著者が一人で行う．一度分類した後，その結果を再び目視で確認して検証する．

4.3.2 結果

分類した結果を表 4.5 に示す．最も頻繁に見られた操作はアプリケーション固有の操作（24.26%）であった．次いでファイルに関連する操作（17.51%），OS の設定やサービスに関連する操作（15.67%）がよく見られた．これら 3 種類のカテゴリのみでタスク全体の半数以上（57.44%）を占めていた．

また，そのタスクが変更を伴うかどうかについて分類した結果を表 4.6 に示す．なお表中では「読み込みのみ」を R,「変更を伴う」を W,「判断できない」を U と表記している．以降の図表においてもこの略記を採用する．アプリケーション固有の操作では変更を伴うタスクが多く見られたが，ファイル関連の操作では読み込みのみのタスクの方が多く見られた．

表 4.4　分類するカテゴリの一覧

カテゴリ名	説明	例
app	アプリケーション固有の操作	a2enmod ○○ pyenv install ○○
file	ファイル関連の操作	touch ○○.txt grep ○○ △△.txt
system	OS の設定やサービスに関連する操作	systemctl enable ○○ sysctl -n ○○
package	ソフトウェアパッケージ関連の操作	apt-get install ○○ pip install ○○
script	スクリプトの実行	./○○.sh python ○○.py
version	アプリケーションなどのバージョンの取得	python -V mysql –version
db	データベース関連の操作	mysql -e ”○○” psql -c ”○○”
fs	ファイルシステムに関連した操作	mount ○○ △△ resize2fs
which	あるコマンドが呼び出し可能か調べる操作	which python type mysql
network	ネットワークに関連する操作	curl ○○ wget ○○
build	ソフトウェアのビルドに関連する操作	./configure make
selinux	SELinux に関連する操作	getenforce semodule -i ○○
validation	設定ファイルの書式の検証	nginx -t visudo -cf ○○
other	その他	date, sleep ○○

表 4.5　カテゴリごとのタスク数とその割合

カテゴリ名	タスク数	割合（%）
app	644	24.26
file	465	17.51
system	416	15.67
package	311	11.71
script	151	5.69
version	111	4.18
db	94	3.54
fs	75	2.83
which	56	2.11
network	44	1.66
build	43	1.62
selinux	25	0.94
validation	8	0.30
other	212	7.98

表 **4.6**　カテゴリごとの変更を伴うタスクの数

カテゴリ名	R	W	U
app	128	489	27
file	322	143	0
system	237	178	1
package	108	201	2
other	143	6	63
script	5	18	128
version	109	1	1
db	17	70	7
fs	31	43	1
which	55	1	0
network	17	27	0
build	0	42	1
selinux	9	16	0
validation	8	0	0

4.4 RQ3）冪等性を保つためにどの程度努力がなされているか

4.4.1 実験方法

Ansible のタスクはその実行に関していくつかの条件を設定できる．その条件を定義することで，タスクの実行可否を制御できる．非冪等モジュールのうち command・shell・script の 3 つは when・creates・removes の 3 種類の条件を持ち，raw モジュールは when という条件のみを持つ．when はそのタスクの実行自体を決める Ansible モジュール共通のパラメータである(注 5)．creates および removes は，上記の 3 つの非冪等モジュール特有のパラメータであり，それぞれ「指定されたファイルパスが存在する場合」及び「存在しない場合」にそのタスクを実行しないという条件である．これらを抽出し，タスクがどの程度条件を持つかを検証する．

また，タスクを複数まとめて記述できる block という記法を用いると，含まれるタスク全てに when 属性を設定できる．これにより，その block 内のタスクは，block 自体に設定された when の条件が満たされない限り実行されない．そのため，タスクの持つ条件を抽出する際にはその親となる block 要素を全て辿り，when を持つ block が存在する場合，そのタスクは実行条件を持つとして判定する．例えばソースコード 4.2 では，block 内に command モジュールを使用したタスクが含まれている．このタスク自体には条件が設定されていないが，その block には when が設定されているため，実行条件を持つとして判定する．

加えて，あるタスクの実行結果が他のタスクの実行条件となる場合がある．例えば handlers ディレクトリ以下に置かれたタスクは，その呼び出し側のタスクが changed となった場合のみ実行される．非冪等モジュールでは常に changed となるが，changed_when という条件によりそれを上書きできる(注 6)．真ならば変更が生じた，偽ならば変更は生じなかったことを示す．この条件についても when と共に調べる．

4.4.2 結果

非冪等モジュール以外のモジュールを使用したタスクと比較した結果，表 4.7 のようになった．非冪等モジュールを使用しているタスクが条件を持つ割合は，それ以外

(注 5): https://docs.ansible.com/ansible/2.9/user_guide/playbooks_conditionals.html
(注 6): https://docs.ansible.com/ansible/2.9/user_guide/playbooks_error_handling.html

ソースコード **4.2**　条件を持つ **block** の例 **[25]**

```
---
- name: Install Amplify Agent if not installed
  block:
    - name: Download Amplify Agent script
      get_url:
        url: "{{ nginx_amplify_script_url }}"
        dest: "{{ nginx_amplify_script_path }}"

    - name: Run Amplify Agent install.sh script
      command: "sh /tmp/install-amplify-agent.sh -y"
      environment:
        API_KEY: "{{ nginx_amplify_api_key }}"
      become: true
      become_user: root
      become_method: sudo

    - name: Remove installation script
      file:
        path: "{{ nginx_amplify_script_path }}"
        state: absent

  when: amplify_agent_installed.failed == true
  tags: [configuration, packages]
```

のモジュールを利用しているタスクと比較して，約 4.56%高かった．非冪等モジュールを使用しているタスク全体において条件を持つものは半数程度に止まるが，変更を伴うタスクでは 69.80%となり，条件を持つものが頻繁に見られた．逆に読み込みのみの場合は 30.28%であり，全体と比較して低い結果となった．

changed_when が設定されているかどうかを調べた結果，表 4.8 のようになった．非冪等モジュールでは条件を持つ割合が 50.75%であり，それ以外のもの（1.18%）と比較して非常に高かった．また，表中の「False である割合」とは，条件を持つもののうち，その条件が False であったものの数を指す．つまり，条件を持つものの多くがその実行内容に関わらず,「常に変更は生じていない」として設定されていた．これは非冪等モジュールだけでなく，それ以外のモジュールについても同様の傾向を示した．加えて非冪等モジュールでは，読み込みのみのタスクの約 84%が changed_when を持つのに対し，変更を伴うタスクでは約 24%に留まった．

4.5 RQ4）代替可能なモジュールは存在するか

4.5.1 実験手法

RQ2 と同様に，そのタスク自体の他に周辺のコメントおよび前後のタスクを参照するため，そのタスクを含むファイル全体を目視で確認する．モジュールとは，Ansible の標準モジュール (注 7) を指す．また，代替可能であるとは，以下の条件を満たす場合を指す．

- Ansible 2.9.2 に実装されている機能で同等の操作を実現できる．いくつ機能を組み合わせても良い．
- 置き換える対象のタスクの操作と求める結果が一致すればよい．出力の形式については問わない．
- 読み込みのみの副作用は許容する．
- そのタスク単体を置き換えることができる．

まず，Ansible の実装する機能とは，モジュールだけでなく Ansible の変数（以降

(注 7): https://docs.ansible.com/ansible/2.9/modules/list_of_all_modules.html に列挙されている，Ansible インストール時に使用可能なモジュール．

表 4.7　実行条件を持つタスクの数と割合

	非冪等モジュール				その他	全て
	R	W	U	合計		
条件を持たない	829	373	91	1,293	8,990	10,283
条件を持つ	360	862	141	1,363	7,896	9,259
条件を持つ割合（%）	30.28	69.80	60.78	51.32	46.76	47.38

表 4.8　実行結果を上書きする条件をもつタスクの数と割合

	非冪等モジュール				その他	全て
	R	W	U	合計		
条件を持たない	192	941	175	1,308	16,686	17,994
条件を持つ	997	294	57	1,348	200	1,548
条件を持つ割合（%）	83.85	23.80	24.57	50.75	1.18	7.92
False である割合（%）	97.99	78.91	77.19	92.95	85.50	91.99

Facts)・プラグイン(注 8)等も含む．また，それらを複数組み合わせることで代替できるものについても代替可能であると判定する．例えばソースコード 4.3 は apt モジュール単体で代替できる．代替後の例をソースコード 4.4 に示す．同様に find と file モジュールのような複数のモジュールで 1 つのタスクを置き換え可能な例がソースコード 4.5 である．これもソースコード 4.6 で示すように，1 つのモジュールで実現されていたタスクであっても，複数のモジュールへ分割することができる．また，Facts を用いて代替可能な例がソースコード 4.7 である．Facts は対象の計算機への接続時に Ansible によって収集され，タスク中で自由に参照できる．実際には取得の操作は不要なため，置き換え後にそのタスクは存在しなくなるが，参照方法の例をソースコード 4.8 に示す．

各タスクについて代替可能であるか，および，代替不可能な場合にはその理由を調査する．全てのタスクを実際に置き換え実行するといったことはせず，公式ドキュメントを参照し，代替可能性の検討のみに留める．

4.5.2 結果

処理内容ごとの代替可能タスク数を表 4.9 に示す．約 45%のタスクは置き換え可能と判定された．また，処理内容ごとに見ると R に分類されたタスクでは代替可能なタスクが 689 個（57.95%），W に分類されたタスクでは 505 個（40.89%）と，R に分類されたタスクは W に分類されたものと比べて代替可能なタスクが多かった．

カテゴリごとの代替可能タスク数を表 4.10 に示す．アプリケーション固有の操作については代替可能なものが少なく，ファイル操作や OS およびサービスに関連する操作は代替可能なものが比較的多かった．

代替できるモジュールまたはプラグインについて，それらによって代替できるタスク数上位 10 件を表 4.11 に示す．ただし，関連するモジュールなどはまとめて計算しており，例えば file モジュールと file プラグインはどちらも file として，OpenSSL 関連のモジュールは全て openssl として集計している．debug モジュールによって代替できるものには，不要なタスクであり，単に削除するだけで良い物も含む．facts は Facts によって代替できるものを指す．

また，代替不可能であった理由を図 4.12 に示す．サポートされていない操作であ

(注 8): https://docs.ansible.com/ansible/2.9/plugins/plugins.html

ソースコード **4.3**　**apt** モジュールで代替可能

```
---
- name: Install wget by apt-get
  command: apt-get install wget
```

ソースコード **4.4**　代替後のソースコード **4.3**

```
- name: Install wget by apt-get
  apt:
    name:
      - wget
```

ソースコード **4.5**　**find** および **file** モジュールで代替可能 **[26]**

```
- name: 4.2.4 - Ensure permissions on all logfiles are configured
  shell: "find /var/log -type f -exec chmod g-wx,o-rwx {} +"
```

ソースコード **4.6**　代替後のソースコード **4.5**

```
- block: 4.2.4 - Ensure permissions on all logfiles are configured
    - find:
        paths: /var/log
        file_type: file
        recurse: yes
      register: find_files
    - file:
        path: "{{ item }}"
        mode: g-wx,o-rwx
      loop: "{{ find_files }}"
```

ソースコード **4.7**　**Facts** で代替可能 **[27]**

```
- name: Get Kernel release
  command: uname -r
  register: KERNEL_RELEASE
  changed_when: false
```

ソースコード **4.8**　代替後のソースコード **4.7**

```
# 実際には不要
- debug:
    msg: "{{ ansible_kernel }}"
```

表 4.9　処理内容ごとの代替可能タスク数とその割合

処理内容	代替可能な数	代替不可能な数	代替可能な割合（%）
R	689	500	57.95
W	505	730	40.89
U	2	230	0.86
合計	1,196	1,460	45.03

表 4.10　カテゴリごとの代替可能タスク数

カテゴリ名	代替可能な数	代替不可能な数	代替可能な割合（%）
app	149	495	23.14
file	393	72	84.57
system	234	182	56.25
package	163	148	52.41
script	0	151	0.00
version	6	105	5.405
db	46	48	48.94
fs	31	44	41.33
which	0	56	0.00
network	16	28	36.37
build	20	23	46.51
selinux	1	24	4.00
validation	0	8	0.00
other	137	75	64.62

表 **4.11**　モジュールごとの代替可能タスク数とその割合上位 **10** 件

モジュール名	代替可能数	割合（%）
file	129	10.79
debug	126	10.54
service_facts	108	9.03
package_facts	81	6.77
find	76	6.35
stat	65	5.43
facts	54	4.52
copy	53	4.43
openssl	45	3.76
systemd	34	2.84

るという理由が最も多く，約 75%を占めた．

表 4.3 で示した，非冪等モジュールの使用数上位 5 つのロールについて，代替可能なモジュールの数とその非冪等モジュール数に対する割合を表 4.13 に示す．上位 2 つは代替可能なものが 80%以上を占めた．

表 4.12　代替不可能であった理由

理由	個数	割合（%）
サポートされていない操作である	1099	75.27
判断できない	225	15.27
互換性を保つため	82	5.62
複雑になりすぎるため	33	2.26
不具合を避けるため	23	1.56

表 4.13　非冪等モジュールの利用数上位 5 つのロールにおける代替可能モジュール数とその割合

ロール名	非冪等モジュール数	代替可能数	割合（%）
MindPointGroup.RHEL7-CIS	471	383	81.32
florianutz.Ubuntu1604-CIS	99	89	89.90
oVirt.hosted_engine_setup	93	20	21.51
anthcourtney.cis-amazon-linux	79	35	44.30
ceph.ceph_handler	44	17	38.64

5. 考察

5.1 Ansibleの実装するモジュールの不足

RQ1の結果より，約半数のロールは1つ以上の非冪等モジュールを使用したタスクを持つことが分かった．また，RQ2よりそれらの操作のうち約24%はアプリケーション固有の操作であった．加えて，RQ4の結果より，代替不可能である理由としてサポートされていない操作であることが最多であった．これらの結果から，Ansibleにおいて命令的モジュールは未だサポートされていないアプリケーションの管理のために使用されており，Ansibleモジュールの網羅する分野が狭いために開発者らの需要を満たしていない可能性を示唆する．Ansibleのモジュールはバージョン2.9.4現在で非推奨のものも含めると3,387個存在する[17]．これに対し，Chefにおけるモジュール数は132個である[28]．しかし，Chefではクックブック(注9)の54.7%で命令的なモジュールが使用されており，RQ1の結果と概ね一致していることがわかった[12]．ここから，Ansibleの実装モジュール数の多さは，網羅する機能の豊富さよりも，モジュール1つあたりの責務を小さくし，機能を細分化したことによる可能性が考えられる．

サポートされていない操作に対する非冪等モジュールの使用は避けられない．そのため，IaCスクリプトにおける品質モデルでは，実現される操作がそのツールによってサポートされているかどうかを考慮するべきであると考えられる．

5.2 冪等性に対する懸念

RQ3より，非冪等モジュールを使用するタスクにおいて，変更を伴うタスクでは`when`などの実行自体を制御する構文を使用する割合が約70%を占めた．これは読み込みのみのタスクの約30%と比較して顕著に高い．一方で，読み込みのみのタスクでは，実行結果を上書きする条件である`changed_when`をもつ割合が約84%であり，変更を伴うタスクの約24%と比較して顕著に高い．読み込みのみのタスクは常に実行されても問題なく変更を生じないが，変更を伴うタスクでは実行により状態が変化するため，状況に応じて実行自体の可否を決定する必要がある．開発者らにはそ

(注9): Ansibleにおけるプレイブックに相当するもの．

のような認識があり，それらのタスクの性質によって与えられる条件が変化することを示唆している．

しかし，変更を伴うタスクではまだ約 3 割が実行条件を持たず，それらの実行は制御されていない可能性がある．これらは冪等性を保つ上で，常に実行されたとしても問題無い場合も考えられる．例えば，実行するコマンド自体が冪等性を担保している場合や，引き起こされる変更が実務上問題のない範囲である場合などである．他にも，実務ではその使用者の想定する副作用の範囲内において，冪等性を満たさないケースが許容される可能性も考えられる．構成管理ツールのコードに対する品質モデルでは，このような使用者や実行されるコマンド自体の性質を考慮する必要があると考えられる．

5.3 非冪等モジュールの使用数の偏り

RQ1 の結果より，非冪等モジュールの使用数上位 5 件のロールのみで，今回発見した非冪等モジュールを使用したタスク全体の約 30%を占めている．これは 5.1 節で述べたように，Ansible によってサポートされていないアプリケーションの管理のため，という理由もあるだろう．しかし，それらのうち MindPointGroup.RHEL7-CIS, florianutz.Ubuntu1604-CIS，anthcourtney.cis-amazon-linux は非冪等モジュールの利用数が多いだけでなく，表 4.13 で示したように代替可能数も多い．これらはいずれも Center for Internet Security (注 10) に関連したロールであるとみられ，一部の開発者らの間でモジュールの使用に関する悪例が共有されている可能性がある．

構成管理ツールを採用する理由には様々なものがあると考えられる．例えばその冪等性による再現可能なデプロイを行うツールとして，多数の計算機へ効率よく設定を適用するための自動化ツールとして，など必要な機能は要求によって異なる．そのため，上記のようなロールでは冪等性は重要視されておらず，実務上の問題がない程度の副作用は許容されていると考えられる．しかし，多くの場合 Ansible の標準モジュールの活用によって記法が簡易になる，コミュニティによるサポートを受けやすいなどのメリットもある．目的が冪等性の担保でない場合にも，モジュールの利用を検討する価値はあるだろう．

(注 10): https://www.cisecurity.org/

5.4 頻繁に見られる悪例

本節では，調査中に見られた Ansible コードにおけるいくつかの悪例についてその概要を示す．

5.4.1 外部スクリプトの実行

シェルスクリプトや Python スクリプト，Perl スクリプトなどを実行する例が見られた．これらのスクリプトファイルの利用は，Ansible の YAML ファイル以外のコードのメンテナンスの必要性や，それらのスクリプト内部で冪等性を保持する必要性を生じさせる．冪等性を保つため複雑化したシェルスクリプトは，その保守性に問題が生じる可能性もある．

本来構成管理ツールを利用する上では，そういったスクリプトの挙動をそのツールのモジュールを使った一連のタスクとして切り出し，冪等な操作として再定義すべきである．しかし，既存のソフトウェア資産の流用によるコスト低減や，利用するソフトウェアが公式にそのようなスクリプトの実行を推奨する場合など，いくつかのケースではスクリプトの実行が積極的に採用される場合があると考えられる．そのような場合，事前にスクリプトの冪等性を検証するなど，更なる保証が必要となるだろう．それを補助するためのツールなどの開発も必要とされている可能性があると考えられる．

5.4.2 宣言的 API の命令的利用

本来，Ansible の宣言的なモジュールは，対象の計算機に状態を記述するのみで良い．例えばソースコード 5.1 のように，あるソフトウェアがインストールされているという状態のみを記述する．しかし，調査中にいくつかソースコード 5.2 のような例を確認できた．このように，命令的な記述を利用して状態を宣言する方法を宣言的 API の命令的利用と命名した．

このような使用方法は，常に問題となるわけではない．実際に，以下のようないくつかの例で宣言的 API の命令的利用が見られた．

- 本来利用できるモジュールなどに不具合がある場合

ソースコード **5.1**　宣言的な実装

```
---
- name: wget is installed
  apt:
    name:
      - wget
```

ソースコード **5.2**　宣言的 **API** の命令的利用

```
---
- name: Check that wget is installed
  command: which wget
  register: wget

- name: Install wget
  apt:
    name:
      - wget
  when: wget.rc != 0
```

- 過去のバージョンでは未実装なモジュールを互換性を維持したまま使用したい場合

1つ目の例として，ansible_selinuxというFactsが，SELinuxのPythonライブラリがインストールされていない場合，その実際の状態に関わらず常にFalseを返すという不具合があった(注11)．これにより開発者らは自身でgetenforceコマンドなどにより，SELinuxが有効かどうかを判定する必要があった．他にも，Pythonのバージョン2.7.9未満で見られるServer Name Indication(注12)のサポートの欠如により，一部モジュールでサーバに接続できないなどの不具合もあった(注13)．これを避けるため，wgetコマンドやcurlコマンドを通じてインターネットへとアクセスする必要性が生じたようだ．このように，モジュールの不具合を避けるため，あえて非冪等モジュールを使用する必要性が生じる場合もあり，一概に使用を問題視することは不適である．

2つ目の例としては，対象の計算機での実行中にその計算機を再起動するrebootモジュールや，Fedoraなどで採用されるパッケージマネージャdnfのモジュールなど，Ansibleのリリースに伴い後から実装されたものがあげられる．前者はバージョン2.7から，後者はバージョン1.9から利用可能となっている．構成管理ツールの利用は運用という側面が強く，簡単にAnsibleのバージョンをあげることができないため，後方互換性の維持が重要視されている可能性がある．しかし，Ansibleの古いバージョンには脆弱性の報告もされており，基本的にはより最新のバージョンへ追従することが重要である．これをサポートするため，Ansibleは公式のテストフレームワークとしてMoleculeを開発している[30]．これは仮想環境に対してAnsibleロールを適用し，適用後のそれらの状態を検証するまでの一連のタスクを自動化するツールである．ソフトウェアに対する継続的インテグレーションなどにおける自動テストの有用性はこれまでの研究から明らかになりつつある[31]．特に，異なる環境における自動テストは有用であることが知られている[32]．複数のOSやツールのバージョンを対象とする必要がある構成管理ツールのコードに対しても，それらは有用である可能性がある．

(注11): https://github.com/ansible/ansible/issues/16612
(注12): 1つのグローバルIPアドレスで複数のSSL証明書を使用できるようにしたSSL/TLSの拡張仕様の1つ[29]
(注13): https://github.com/ansible/ansible/issues/12161

5.4.3 Facts の不使用及びモジュールの認知度不足

RQ4 より，service_facts・package_facts・facts の 3 種のみで代替可能数の約 20%を占める．service_facts および package_facts はモジュールとして実装されているが，何らかの変更を伴うものではなく，それぞれサービスの状態，インストール済みパッケージの状態を取得するものである．このように，実際には Ansible の機能で情報を収集できるにも関わらず，Ansible ロールの開発者らは，自分たちが慣れた方法で情報を収集する傾向にある可能性がある．つまり，何らかの情報を出力するコマンドと grep や cut などの出力を加工するコマンドの組み合わせの方が，Facts よりも採用されやすい可能性がある．

その背景には Ansible のドキュメントの不足が考えられる．Facts に関するドキュメント [33] には，具体的な値の例はあるものの，それらが他の OS ではどうなるか，どのようにして値が取得されるのかについては言及されていない．Facts を利用する場合，利用者は自ら実行して実際にどのような Facts を持つのかを調べる必要があり，採用をためらってしまう可能性がある．

また，そのようなドキュメントの不足と関連して，モジュールの認知度および理解度の不足があげられる．代替可能数の上位には file や find など，一般的な名前を持ち，汎用的な UNIX コマンドで同等の操作を実現できるモジュールが見られた．これは，その存在を認知されていない，または詳細な機能や仕様について理解されていないために，広く用いられるコマンドによって置き換えられている可能性がある．5.1 節で述べたように，Ansible には 3,000 を超えるモジュールが存在するため，ドキュメントの総数もまた膨大となる．Ansible はそのモジュール数の多さに対してドキュメントの検索性が低く，どのようにモジュールの存在自体を知るか，また，どのようにその詳細なドキュメントにたどり着くかが課題であると考えられる．

6. 関連研究

6.1 冪等性の検証

通常，冪等性の検証は開発者らが手動で作成したテストによって行われる．ChefSpec [34]，rspec-puppet [35]，Molecule [30] など，仮想マシンを使用し分離されたテスト環境を構築した後，それに対する構成管理の適用と，その構成をテストするフレームワークは広く用いられている．

Hummer らは，Chef のクックブックに対する冪等性検証の自動化フレームワークを提案し，検証した [12]．同様に，Puppet に対しても Hanappi らにより同様のフレームワークが提案されている [36]．いずれも LXC や Docker のようなコンテナ技術を用いた軽量な仮想マシンに対して，様々なケースでそれらの IaC スクリプトを適用する．実行のたびに状態の変化を記録することにより，最終的に対象のシステムの状態が収束するのかを判定する．これらの冪等性の自動検証フレームワークは，開発者らの努力を必要とせずにテストを行える反面，全てのケースを網羅することが難しい点や実行に時間がかかる点などが課題となっている．また，自動テストによって発見された非冪等な 263 個のタスクのうち，90 個がユーザ定義スクリプトを実行するものであった [12]．このように，実際の IaC スクリプトにおいて，命令的なモジュールの使用は問題を引き起こしやすいことが指摘されている．5.2 節で述べたように，開発者らは変更を伴うタスクについてその実行そのものを制御する傾向があるが，これらによって実際に冪等性が保たれるかは不明である．それらの非冪等であると判定されたタスクについて，その処理内容や実行条件を調べることで，命令的モジュールの利用による冪等性に対するリスクを体系づけられる可能性がある．

これに対し，Shambaugh らは Puppet コードに対する静的な解析によって，その収束性を決定するツール *Rehearsal* を提案した [37]．これは Puppet におけるリソース(注 14)を，ファイル操作を行う小さな独自の命令型言語へと置換し，それらを解析することで競合を検出する．ただし，Puppet におけるユーザ定義スクリプトを実行するリソースである `exec` を含むものは，容易に置換できないため対象外となっている．静的解析によるアプローチは実施できれば効果的である反面，実際には `exec` の

(注 14): Ansible におけるモジュールに相当するもの

ようなリソースの存在により，適用できる場面は限られであろうことが指摘されている [36]．実際に，本研究ではアプリケーション固有の操作が最も多いという結果が得られており，それらを同様の手法で解析することは容易でないと考えられる．IaC スクリプトにおいては静的解析よりも実際の実行により判定するフレームワークが現実的な選択肢であると考えられる．

6.2 IaC スクリプトの品質モデル

IaC ツールは，ソースコードを用いることによってバージョン管理やテストなどの，これまでソフトウェアのコードに対して適用されてきたプラクティスを再利用できるというメリットがある．しかし，一方でソースコードが持つ負の側面にもまた直面する．すなわち，コードが大規模化することに伴う複雑さの増加である．ソフトウェアのソースコードと同様に，IaC スクリプトもまた開発および保守していく必要があるため，その品質を維持することは開発者らにとって重要である．

品質の悪化の兆候を捉えるため，コードスメルという概念が提唱されており，これらは既存の汎用プログラミング言語に対して複数の研究が行われている [38, 39]．Sharma らによって，Puppet スクリプトの実装および設計の両方についてコードスメルを測定するモデルが提案されている [14]．ここでは Puppet の `exec` のようなリソースは，Puppet 自体の宣言的であるべきという思想に反するとして，その使用自体をコードスメルとして言及している．Schwarz らは Chef と Puppet の両方に対してコードスメルのメトリクスを提案し，実際に測定した結果を比較している [40]．ここでもユーザ定義スクリプトを実行するリソースは，その実行スクリプトの LOC などがコードスメルのメトリクスとしてあげられている．これらはその処理内容などについては言及していない．しかし，本研究の結果から考えると，それらのリソースはツールによってサポートされていない操作を実現するために用いられる可能性が高い．その使用を避けられない場合にもまとめてコードスメルであると述べることには疑問の余地がある．

また，Scwarz らは Puppet スクリプトに対する品質モデルを定義し，実際に Puppet スクリプトの開発者らによる評価と比較している [13]．ここでは，Puppet スクリプトの開発者らに対し，アンケート調査を通して悪例を収集して品質モデルのための

メトリクスを定義し，Puppet のエキスパートらとその妥当性について議論している．メトリクスの 1 つとして exec リソースの数が上げられており，専門家らもその多用が悪例であることには同意している．しかし，慎重に運用すれば問題が無いという意見もあげられている．本研究の成果は，このような品質モデルに対して改善の可能性を示唆する．ツールによってサポートされていない操作を実現するためには，そのような命令的モジュールの利用を避けることが出来ない．そのため，ある操作を命令的なモジュールで実現しているとき，それを悪例と捉えるかどうかは，宣言的な別のモジュールで置き換えることができるかどうかによって測られるべきである．

7. 妥当性への脅威

7.1 構成概念妥当性

本研究では，冪等性に対する努力の基準として，各タスクに設定できる実行条件の有無を調べた．しかし，実行条件を持たなかったものの中には，実行されるコマンド中で if 文を使用している場合や，常に実行されても問題ないコマンドである場合などが含まれる．そのため，必ずしも実行条件を持たないことが，冪等性を保つための努力の欠如とは言えない．実際に実行し，冪等であるかを検証することによって，その努力は評価されるべきである．このような場合，その Ansible ロールが想定する全ての環境を用意し検証しなければ，一部タスクについて実行されないケースが出るなど，網羅できるタスクが制限される可能性がある．全てのタスクを網羅するためには妥当な基準であったと考える．

他に，代替可能モジュールの探索の際，周辺のコメント等を参考にした点があげられる．これは開発者らの主観および知見に基づいたものであり，実際にその内容が妥当であるかは検証していない．そのため，実際にそのコメント内で言及された不具合などが再現するかどうかには疑問の余地がある．また，コメント等は記述されていないが，実際には何らかの不具合を避けるための手法であるなどの可能性も存在する．これら全てについて検討することは現実的でなく，構成概念妥当性への脅威になり得る．

非冪等モジュールとして `command`・`shell`・`raw`・`script` の 4 つを取り上げた．しかし，Ansible における命令的なモジュールはこれらのみに限らない．Ansible はどれが命令的であるかを明示していないため，著者の経験則に基づき，頻繁に利用されるであろうものを選択した．ただし，これらは全て Ansible の Commands modules に該当するモジュールのうち，後方互換性の保証された安定したモジュールである [41]．これらは Ansible の歴史において古くから利用されていることが期待されるため，今回の目的を達成するためには妥当な選択であると考えられる．

7.2 外的妥当性

本研究では構成管理ツールとして Ansible を用いた．これは通常プログラム的構文を持たない YAML ファイルによって記述するなど，既存研究で多く用いられる Puppet 等との差異が見られる．そのため，全ての結果をその他のツールに対して一般化することは難しい可能性がある．

また，Ansible Galaxy で共有されているオープンソースな Ansible ロールのみを解析対象としている．企業等で実際に使用されている Ansible ロールは異なる性質を持つ可能性がある．構成管理ツールの性質上，パスワードやトークンなどの秘匿情報を含む可能性があるため，それらを調査することは難しいと考えられる．しかし，Ansible Galaxy は Ansible の使用者らによって広く使用されうるものであり，実際に使用される Ansible ロールのデータとして妥当なものであると考えられる．

7.3 内的妥当性

本研究では非冪等モジュールの用途や代替可能なモジュールが存在するかについて，目視で分類することにより調査した．これは著者の知見および経験，そして公式ドキュメントの記述などを参考に手動で行った．その妥当性には疑問の余地があり，実際の開発者らの主張は異なる可能性がある．これは本研究の大きな課題であり，多数の Ansible ロールの開発者らによる多数決等を通して調査することで，より正確な結果を得られる可能性がある．

また，Ansible の変数埋め込みを解決していない点もあげられる．実際にそれらの変数がどのような値を取り得るかは実行時まで確定しないため，7.1 節で述べたように想定されうる全ての環境を用いなければ，網羅できるタスク数を制限される可能性がある．変数に初期値が与えられている場合もあるが，これは必須でない．実行中に定義される動的な変数なども存在する．さらに，単純な文字列・数値だけでなく，連想配列のような複雑な値も取り得るため，値を予測することも難しい．本研究の目的はこれを解決せずともある程度達成できるため，必ずしも必要ではないと考えられる．ただし，目視での分類時，変数の用いられ方によってどのカテゴリへ分類すべきかの判断に迷うことがあった．変数埋め込みを解決した方がより精度の高い結果を得られる可能性がある．

8. 結言

本研究では，構成管理ツールにおける非冪等な操作をしうる命令的なモジュールについて，いくつかの観点からその利用形態を明らかにした．

Ansible において非冪等モジュールは約 45%のロールから少なくとも 1 つ以上利用されており，一般的に広く利用されていると言える．また，それらは主に Ansible によってサポートされていないアプリケーションの管理のために用いられる他，ファイル操作や OS およびサービスの設定のためなどに用いられることを明らかにした．加えて，標準モジュールの不具合を避けるためや，互換性の維持のために用いられる例も見られた．

それらの非冪等モジュールを使用したタスクについて，変更を伴うものでは約 70%について実行条件が設定され，読み込みのみの場合にはその約 84%が常に変更を引き起こさないとして設定されていた．ここから開発者らはそれらの違いを認識し，冪等性担保のために努力していると考えられる．一方，非冪等モジュール全体の約 45%については代替可能な別のモジュールの存在が示唆された．開発者らは非冪等モジュールの使用を検討する際に，同等の操作を実現できるモジュールの存在を確認するべきであると考えられる．

これらの結果から，IaC スクリプトの品質モデルにおける命令的モジュールの取り扱いの改善，代替可能モジュールを提案するツールの作成など，様々な研究への応用が考えられる．

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．また，研究に関する助言を頂きました国立研究開発法人 産業技術総合研究所 主任研究員 山形頼之博士をはじめとする産業技術総合研究所の皆様にも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，本学設計工学専攻　西浦生成先輩，近藤将成先輩，情報工学専攻　塩津拓真君，脇上幸洋君，情報工学課程　杉浦智基君，山本凱君を

はじめとする，ソフトウェア工学研究室の皆さん，研究室生活に癒しを与えてくれたあくあたん，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. Wettinger, U. Breitenbücher, and F. Leymann, “Devopslang – bridging the gap between development and operations,” in Proceedings of the 2014 Service-Oriented and Cloud Computing (ESOCC), eds. M. Villari, W. Zimmermann, and K.-K. Lau, pp.108–122, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[2] J. Smeds, K. Nybom, and I. Porres, “Devops: A definition and perceived adoption impediments,” in Proceedings of the 16th Agile Processes in Software Engineering and Extreme Programming, eds. C. Lassenius, T. Dingsøyr, and M. Paasivaara, pp.166–177, Springer International Publishing, Cham, 2015.

[3] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, and D.A. Tamburri, “DevOps: Introducing infrastructure-as-code,” Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C), pp.497–498, Institute of Electrical and Electronics Engineers Inc., jun 2017.

[4] C.P. Bezemer, S. Eismann, V. Ferme, J. Grohmann, R. Heinrich, P. Jamshidi, W. Shang, A. Van Hoorn, M. Villavicencio, J. Walter, and F. Willnecker, “How is performance addressed in DevOps? A survey on industrial practices,” Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering (ICPE), pp.45–50, Association for Computing Machinery, Inc, apr 2019.

[5] C. Parnin, E. Helms, C. Atlee, H. Boughton, M. Ghattas, A. Glover, J. Holman, J. Micco, B. Murphy, T. Savor, and etal., “The top 10 adages in continuous deployment,” IEEE Softw., vol.34, no.3, p.86–95, May 2017.

[6] HashiCorp, Terraform by HashiCorp, （オンライン）, 入手先〈https://www.terraform.io/〉（参照 2020-1-17）.

[7] Pulumi, Pulumi - Modern Infrastructure as Code, （オンライン）, 入手先〈https://www.pulumi.com/〉（参照 2020-1-17）.

[8] Red Hat, Inc., Ansible is Simple IT Automation, （オンライン）, 入手先〈https://www.ansible.com/〉（参照 2020-1-17）.

[9] Puppet, Powerful infrastructure automation and delivery — Puppet, （オンライン）, 入手先〈https://puppet.com/〉（参照 2020-1-17）.

[10] Chef Software, Inc., Chef: Deploy new code faster and more frequently. Automate infrastructure and applications, （オンライン）, 入手先〈https://www.chef.io/〉（参照 2020-1-17）.

[11] A. Couch and Y. Sun, "On the algebraic structure of convergence," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol.2867, pp.28–40, 2003.

[12] W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam, "Testing idempotence for infrastructure as code," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol.8275 LNCS, pp.368–388, 2013.

[13] E. Van Der Bent, J. Hage, J. Visser, and G. Gousios, "How good is your puppet? An empirically defined and validated quality model for puppet," Proceedings of the 25th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), vol.2018-March, pp.164–174, Institute of Electrical and Electronics Engineers Inc., apr 2018.

[14] T. Sharma, M. Fragkoulis, and D. Spinellis, "Does your configuration code smell?," Proceedings of the 13th Working Conference on Mining Software Repositories (MSR), pp.189–200, Association for Computing Machinery, Inc, may 2016.

[15] M. Guerriero, M. Garriga, D.A. Tamburri, and F. Palomba, "Adoption, support, and challenges of infrastructure-as-code: Insights from industry," Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp.580–589, Sep. 2019.

[16] Red Hat, Inc.，Ansible concepts — Ansible Documentation，（オンライン），入手先〈https://docs.ansible.com/ansible/2.9/user_guide/basic_concepts.html〉（参照 2020-1-31）.

[17] Red Hat, Inc.，All modules — Ansible Documentation，（オンライン），入手

先〈https://docs.ansible.com/ansible/2.9/modules/list_of_all_modules.html〉（参照 2020-1-27）.

[18] Red Hat, Inc.，Roles — Ansible Documentation,（オンライン），入手先〈https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse_roles.html〉（参照 2020-1-31）.

[19] YAML Ain't Markup Language (YAML™) Version 1.1,（オンライン），入手先〈https://yaml.org/spec/1.1/〉（参照 2020-1-28）.

[20] Pallets，Jinja — Jinja Documentation (2.11.x),（オンライン），入手先〈https://jinja.palletsprojects.com/en/2.11.x/〉（参照 2020-1-31）.

[21] Red Hat, Inc.，Conditionals — Ansible Documentation,（オンライン），入手先〈https://docs.ansible.com/ansible/2.9/user_guide/playbooks_conditionals.html〉（参照 2020-2-1）.

[22] Red Hat, Inc.，Loops — Ansible Documentation,（オンライン），入手先〈https://docs.ansible.com/ansible/2.9/user_guide/playbooks_loops.html〉（参照 2020-2-1）.

[23] Red Hat, Inc.，Galaxy User Guide — Ansible Documentation,（オンライン），入手先〈https://docs.ansible.com/ansible/latest/galaxy/user_guide.html〉（参照 2020-2-1）.

[24] J. Geerling, ansible-role-daemonize/main.yml at 1.2.1 · geerlingguy/ansible-role-daemonize,（オンライン）, 入手先〈https://github.com/geerlingguy/ansible-role-daemonize/blob/1.2.1/tasks/main.yml〉（参照 2020-1-24）.

[25] J. Dauphant, ansible-role-nginx/amplify.yml at v2.21.2 · jdauphant/ansible-role-nginx,（オンライン）, 入手先〈https://github.com/jdauphant/ansible-role-nginx/blob/v2.21.2/tasks/amplify.yml#L10-L31〉（参照 2020-1-24）.

[26] anthcourtney, ansible-role-cis-amazon-linux/4.2.4.yml at v1.1.8-beta · anthcourtney/ansible-role-cis-amazon-linux,（オンライン）, 入手先〈https://github.com/anthcourtney/ansible-role-cis-amazon-linux/blob/v1.1.8-beta/tasks/level-1/4.2.4.yml#L6-L12〉（参照 2020-1-24）.

[27] GRyCAP, ansible-role-docker/Debian.yml at 45bfc55c745f3f39f8557de98ddd3d8be9b7ee89 · grycap/ansible-role-docker, （オンライン）, 入手先〈https://github.com/grycap/ansible-role-docker/blob/45bfc55c745f3f39f8557de98ddd3d8be9b7ee89/tasks/Debian.yml〉（参照 2020-1-24）.

[28] Chef Software, Inc.，Resources Reference — Chef Docs,（オンライン），入手先〈https://docs.chef.io/resource_reference.html〉（参照 2020-1-27）.

[29] D.E. 3rd, RFC 6066 - Transport Layer Security (TLS) Extensions: Extension Definitions, IETF（オンライン）, 入手先〈https://tools.ietf.org/html/rfc6066〉（参照 2020-1-27）.

[30] J. Dewey，Molecule — Molecule 2.22 documentation,（オンライン），入手先〈https://molecule.readthedocs.io/en/stable/index.html〉（参照 2020-2-2）.

[31] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, “Quality and productivity outcomes relating to continuous integration in GitHub,” Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), pp.805–816, 2015.

[32] M. Beller, G. Gousios, and A. Zaidman, “Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub,” Proceedings of the 14th International Conference on Mining Software Repositoriesd (MSR), pp.356–367, IEEE Computer Society, jun 2017.

[33] Red Hat, Inc.，Using Variables — Ansible Documentation,（オンライン），入手先〈https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html〉（参照 2020-1-28）.

[34] Chef Software, Inc.，ChefSpec — Chef Docs,（オンライン），入手先〈https://docs.chef.io/chefspec.html〉（参照 2020-2-2）.

[35] T. Sharpe, rspec-puppet,（オンライン）, 入手先〈https://rspec-puppet.com/〉（参照 2020-2-2）.

[36] O. Hanappi, W. Hummer, and S. Dustdar, “Asserting reliable convergence for configuration management scripts,” Proceedings of the 2016 ACM SIGPLAIN International

Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), vol.02-04-Nove, pp.328–343, Association for Computing Machinery, oct 2016.

[37] R. Shambaugh, A. Weiss, and A. Guha, "Rehearsal: A configuration verification tool for puppet," Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), vol.13-17-June, pp.416–430, Association for Computing Machinery, jun 2016.

[38] E. Van Emden and L. Moonen, "Java Quality Assurance by Detecting Code Smells," Proceedings of the Ninth Working Conference on Reverse Engineerin (WCRE), vol.2002-January, pp.97–106, IEEE Computer Society, 2002.

[39] M.I. Azeem, F. Palomba, L. Shi, and Q. Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," Information and Software Technology, vol.108, pp.115–138, Elsevier B.V., apr 2019.

[40] J. Schwarz, A. Steffens, and H. Lichter, "Code smells in infrastructure as code," Proceedings of the 11th International Conference on the Quality of Information and Communications Technology (QUATIC), pp.220–228, 2018.

[41] Red Hat, Inc.，Commands modules — Ansible Documentation,（オンライン），入手先〈https://docs.ansible.com/ansible/2.9/modules/list_of_commands_modules.html〉（参照 2020-1-30）.