Empirical Software Engineering (2020) 25:3430–3467
https://doi.org/10.1007/s10664-020-09841-8

The 'as code' activities: development anti-patterns for infrastructure as code

Akond Rahman[1] · **Effat Farhana**[2] · **Laurie Williams**[2]

Published online: 17 August 2020

Abstract

Context: The 'as code' suffix in infrastructure as code (IaC) refers to applying software engineering activities, such as version control, to maintain IaC scripts. Without the application of these activities, defects that can have serious consequences may be introduced in IaC scripts. A systematic investigation of the development anti-patterns for IaC scripts can guide practitioners in identifying activities to avoid defects in IaC scripts. Development anti-patterns are recurring development activities that relate with defective IaC scripts.

Goal: *The goal of this paper is to help practitioners improve the quality of infrastructure as code (IaC) scripts by identifying development activities that relate with defective IaC scripts.*

Methodology: We identify development anti-patterns by adopting a mixed-methods approach, where we apply quantitative analysis with 2,138 open source IaC scripts and conduct a survey with 51 practitioners.

Findings: We observe five development activities to be related with defective IaC scripts from our quantitative analysis. We identify five development anti-patterns namely, 'boss is not around', 'many cooks spoil', 'minors are spoiler', 'silos', and 'unfocused contribution'.

Conclusion: Our identified development anti-patterns suggest the importance of 'as code' activities in IaC because these activities are related to quality of IaC scripts.

Keywords Anti-pattern · Bugs · Configuration script · Continuous deployment · Defect · Devops · Infrastructure as code · Practice · Puppet · Quality

Communicated by: Daniel Méndez

✉ Akond Rahman
arahman@tntech.edu

Effat Farhana
efarhan@ncsu.edu

Laurie Williams
lawilli3@ncsu.edu

[1] Department of Computer Science, Tennessee Technological University, Cookeville, TN, USA

[2] Department of Computer Science, North Carolina State University, Raleigh, NC, USA

1 Introduction

Infrastructure as code (IaC) is the practice of automatically defining and managing deployment environments, system configurations, and infrastructure through source code (Humble and Farley D 2010). Before the inception of IaC, system operators used to create custom deployment scripts, which were not developed and maintained using a systematic software development process (Turnbull 2007). The 'as code' suffix refers to applying development activities considered to be good practices in software development, such as keeping scripts in version control, testing, and submitting code changes in small units (Morris 2016). With the availability of cloud computing resources such as Amazon Web Services, [1] the development and maintenance of deployment scripts became complex, which motivated information technology (IT) organizations to treat their deployment scripts as regular software source code.

IaC scripts are also referred to as configuration scripts (Sharma et al. 2016; Humble and Farley D 2010) or configuration as code scripts (Rahman et al. 2018). IT organizations widely use commercial tools such as Puppet, to implement the practice of IaC (Humble and Farley D 2010; Jiang and Adams 2015; Shambaugh et al. 2016). These IaC tools provide programming syntax and libraries so that programmers can specify configuration and dependency information as scripts. For example, Puppet provides the 'user resource' library (Labs 2018) to add, update, and delete users in local/remote servers and cloud instances. Instead of manually logging into servers and running commands, users can be configured on multiple servers by automatically running a single IaC script. The use of IaC scripts has resulted in benefits for IT organizations. For example, the use of IaC scripts helped the National Aeronautics and Space Administration (NASA) to reduce its multi-day patching process to 45 minutes (Ansible 2019). The Enterprise Strategy Group surveyed practitioners and reported the use of IaC scripts to help IT organizations gain 210% in time savings and 97% in cost savings on average (Leone M 2016).

However, IaC scripts are susceptible to defects, similar to software source code (Jiang and Adams 2015). Defects in IaC scripts propagate at scale, causing serious consequences. For example, the execution of a defective IaC script erased home directories of ∼270 users in cloud instances maintained by Wikimedia Commons (2017). As another example, a defective IaC script resulted in an outage worth of 150 million USD for Amazon Web Services (Hersher 2017).

One strategy to prevent defects in IaC scripts is to identify development anti-patterns. A development anti-pattern is a recurring practice, which relates to a negative consequence in software development (Brown et al. 1998). In our paper, a development anti-pattern is a recurring development activity that relates with defective scripts, as determined by empirical analysis. We can identify development activities that relate with defective IaC scripts by mining open source software (OSS) repositories (Adams and McIntosh 2016). Let us consider Fig. 1, which provides a code snippet from an actual commit ('825a073'). [2] This 460-line long commit includes a defect. Reviewing large-sized commits, such as '825a073', is challenging (MacLeod et al. 2018). As a result, defects existent in large commits may not be detected during inspection. This anecdotal example suggests a relationship that may exist between the activity of submitting large commits and defects that appear in IaC scripts. By mining metrics from OSS repositories, we can identify recurring development activities,

[1] https://aws.amazon.com/

[2] https://github.com/Mirantis/puppet-manifests

```
{
     require => [                                    Defect: wrong package
       Package['libmysql-java'],
       File['/var/lib/gerrit/review_site/lib'],
       Package['gerrit'],
     ],
     notify => Service['gerrit'],
}
```

Fig. 1 Example of a defect included in the '825a073' commit, which is 460 lines long

such as submitting large-sized commits, and investigate their relationship with defective IaC scripts.

In IaC development, submitting small sized commits is considered as an 'as code' activity (Morris 2016). The activity of submitting small-sized commits is also applicable for software engineering in general (MacLeod et al. 2018; Rigby et al. 2008). Through systematic analysis, we can determine if non-adoption of the 'as code' activities actually relate with defective IaC scripts. For example, if submitting large-sized commits is related with defective IaC scripts, then the identified relationship will underline the importance of the 'as code' activity of submitting small sized commits. If we observe large-sized commits to recur across datasets, and observe large-sized commits to be related with defective IaC scripts, then we can identify submitting large-sized commits as a development anti-pattern. We quantify development activities using development activity metrics i.e. metrics that quantify an activity used to develop IaC scripts. For example, 'commit size' is a development activity metric that quantifies how large or small are submitted commits are for IaC development.

With development activity metrics we can quantify a set of activities used in IaC script development. We use each of these metrics as each one of them correspond to a development activity. Not all development activity metrics are actionable or can be mined from our repositories, and that is why we identify a specific set of development activity metrics.

The metrics help us to quantitatively determine if a development activity is related with IaC script quality. But, to obtain relevance of our quantitative findings amongst practitioners we only cannot rely on quantitative findings. Hence, we survey practitioners involved in IaC script development. Survey results could reveal if our findings have relevance and also explain the reasons why practitioners may or may not find our findings to be relevant.

Identification of development activity metrics can also be useful to prioritize inspection and testing efforts. We can construct prediction models with development activity metrics. Constructed prediction models can help practitioners automatically identify scripts that are likely to be defective. Instead of inspecting and testing all IaC scripts used by a team, practitioners from the team can prioritize their inspection and testing efforts for a subset of scripts. For general purpose programming languages defect prediction models are used to prioritize inspection and testing efforts in industry (Tosun et al. 2010; Turhan et al. 2009), and similar efforts can also be pursued for IaC scripts as well.

The goal of this paper is to help practitioners improve the quality of infrastructure as code (IaC) scripts by identifying development activities that relate with defective IaC scripts.

We answer the following research questions:

– **RQ1**: How are development activity metrics quantitatively related with defective infrastructure as code scripts?

- **RQ2**: What are practitioner perceptions on the relationship between development activity metrics and defective infrastructure as code scripts?
- **RQ3**: How can we construct defect prediction models for IaC scripts using development activity metrics?

We conduct quantitative analysis with 2,138 IaC scripts collected from 94 OSS repositories to investigate what development activity metrics relate with defective scripts. We hypothesize seven development activity metrics to show relationship with defective scripts. We consider these metrics for two reasons: (i) the metrics can provide actionable advice; and (ii) the metrics can be mined from OSS repositories. We investigate the relationship between the seven hypothesized metrics and defective IaC scripts using (i) statistical analysis, (ii) developer surveys and interviews, and (iii) prediction models.

Contributions: We list our contributions as following:

- A list of development anti-patterns for IaC scripts; and
- A set of defect prediction models constructed using development activity metrics.

We organize the rest of the paper as following: in Section 2 we describe relevant related work. We describe our dataset construction process in Section 3. We answer RQ1, RQ2, and RQ3 respectively, in Sections 4, 5, and 6. We discuss our findings in Section 7. We discuss our limitations in Section 8 and conclusions in Section 9.

2 Related Work

Our paper is related to research studies that have focused on IaC technologies. Sharma et al. (2016) identified 13 code smells that may cause maintenance problems for Puppet scripts, such as missing default case and improper alignment. Jiang and Adams (2015) investigated the co-evolution of IaC scripts and other software artifacts and reported IaC scripts to experience frequent churn, similar to software source code. Weiss et al. (2017) proposed and evaluated 'Tortoise', a tool that automatically corrects erroneous configurations in IaC scripts. Bent et al. (2018) proposed and validated nine metrics to detect maintainability issues in IaC scripts. Rahman et al. (2018) investigated the questions that developers ask on Stack Overflow to identify the potential challenges developers face while working with Puppet. Rahman and Williams in separate studies characterized defective IaC scripts using text mining (Rahman and Williams 2018), and by identifying source code properties (Rahman and Williams 2019). In another work, Rahman et al. (2019) identified 21,201 occurrences of security smells for IaC scripts that included 1,326 occurrences of hard-coded passwords. Rahman et al. (2020) also constructed a defect taxonomy for IaC scripts that included eight defect categories.

Our paper is related with prior research that have studied the relationships between development activity metrics and software quality. Meneely and Williams (2009) observed that development activity metrics, such as developer count, show relationship with vulnerable files. Rahman and Devanbu (2013a) observed that prediction models constructed using development activity metrics have higher prediction performance than models built with source code metrics. Pingzer et al. (2008) observed that development activity metrics are related with software failures. Tufano et al. (2017) used development metrics, to predict fix-inducing code changes.

The above-mentioned publications have not focused on empirical analysis that investigates what development anti-patterns may exist for IaC scripts. We address this research gap by taking inspiration from prior work that has used development activity metrics and apply it to the domain of IaC. IaC scripts use domain specific languages (Shambaugh et al. 2016). Domain specific languages are different from general purpose programming languages with respect to syntax and semantics. The syntax and semantics of DSLs are different from GPLs Voelter (2013) Hudak (1998) Van Wyk et al. (2007). Furthermore, IaC scripts are often developed by system administrators who may not be well-versed on recommended software engineering practices, such as use of frequent commits. Differences in syntax, semantics, and development background necessitates to investigate to what extent existing development activity metrics are related with IaC script quality.

3 Methodology Overview, Datasets, and Metrics

In this section, first we provide an overview of methodology. Next, we discuss necessary datasets and metrics.

3.1 Methodology Overview

The practice of IaC includes the application of recommended development activities for traditional software engineering, such as submitting small-sized commits, in IaC script development. In our empirical study, we systematically investigate if development activity metrics that correlate with software quality as reported in prior work (Meneely and Williams 2009; Bird et al. 2011), also show correlation with defective IaC scripts. We conduct our empirical study using a mixed-methods approach (Easterbrook et al. 2008), where we investigate the relationship between development activity metrics and defective scripts. *First*, to answer **RQ1**, we derive development activity metrics using a scoping review (Anderson et al. 2008; Arksey H and O'Malley L 2005) i.e., a literature review in limited scope (Section 3.4), and determine the relationship of the identified metrics using (i) Mann Whitney U Test (Mann and Whitney 1947) and (ii) One way Multivariate Analysis of Variance (OMANOVA) (Huberty and Olejnik 2006). *Second*, we answer **RQ2** to obtain practitioner perspective on our quantitative results from RQ1. We conduct an online survey and semi-structured interviews to synthesize practitioner perceptions. Our synthesis reveals the reasons why practitioners agree and disagree with our quantitative findings from RQ1. *Third*, we answer **RQ3** by investigating if defect prediction models can be constructed using the development activity metrics which relate with defective scripts. The constructed prediction models can help practitioners to prioritize defective IaC scripts for further inspection and testing. A summary of our methodology is presented in Fig. 2.

To answer our research questions, we construct datasets in which there is a mapping between a defect and an IaC script. In this section, we describe the methodology to construct the datasets, and the metrics that we used to conduct our empirical study. First, we describe the methodology to construct our datasets. In our dataset, a defect is an imperfection that needs to be replaced or repaired, based upon the IEEE definition of defects (IEEE 2010). A defect-related commit is a commit whose message indicates an action was taken to address a defect. We refer to an IaC script that is listed in a defect-related commit as a defective script. An IaC script that is listed in a commit, which is not defect-related is referred to as a neutral script.

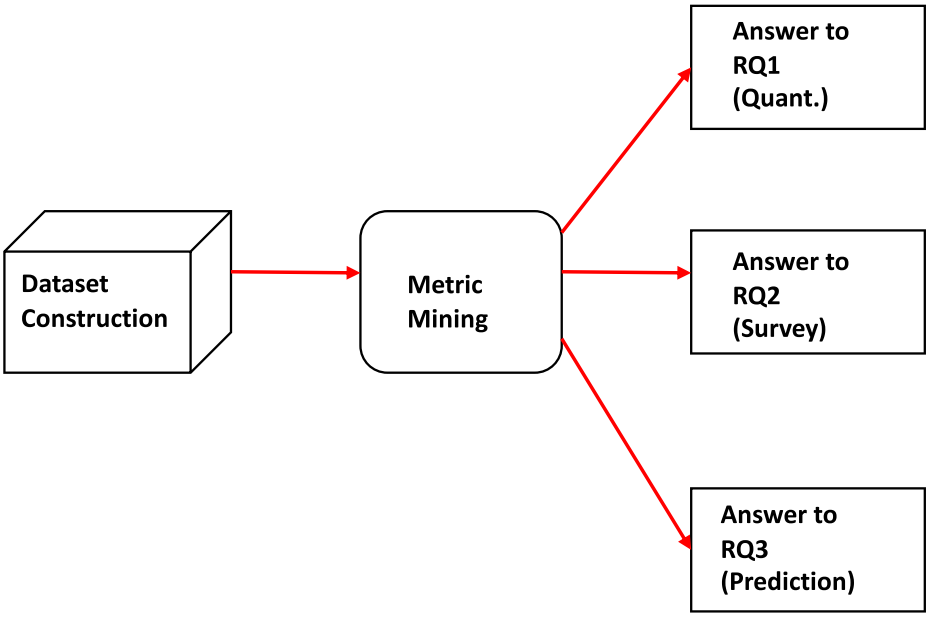

Fig. 2 A summary of our methodology to answer RQ1, RQ2, and RQ3

3.2 Dataset Construction

We use Puppet scripts to construct our dataset because Puppet is considered one of the most popular tools to implement IaC (Jiang and Adams 2015) (Shambaugh et al. 2016) and has been used by companies since 2005 (McCune JT and Jeffrey 2011). We construct our datasets using OSS repositories collected from four organizations: Mirantis, Mozilla, Openstack, and Wikimedia Commons. We select repositories from these four organizations because these organizations create or use cloud-based infrastructure services, and our assumption is that an analysis of IaC scripts collected from these organizations could give us sufficient data to conduct our research study. Our assumption is that repositories collected from these four organizations will contain sufficient IaC scripts for analysis. We describe our steps to construct datasets below:

3.2.1 Repository Selection

We select repositories needed for analysis by applying the following criteria:

Criterion-1 The repository must be available for download.

Criterion-2 At least 11% of the files belonging to the repository must be IaC scripts. Jiang and Adams (2015) reported that in OSS repositories IaC scripts co-exist with other types of files, such as Makefiles and source code files. They observed a median of 11% of the files to be IaC scripts. Our hypothesis is that we will be able to obtain repositories that contain sufficient amount of IaC scripts by using the cutoff of 11%.

Criterion-3 The repository must have at least two commits per month. Munaiah et al. (2017) used the threshold of at least two commits per month to determine which repositories have enough development activity for software organizations. We use this threshold to filter repositories with short development activity.

3.2.2 Commit Message Processing

For commit message processing we use: (i) commits that indicate modification of IaC scripts; and (ii) issue reports that are linked with the commits. We use commits because commits contain information on how and why a file was changed. We also use issue report summaries which can provide more insights on the issue. *First*, we extract commits that were used to modify at least one IaC script. A commit lists the changes made on one or multiple files (Alali et al. 2008). *Second*, we extract commit message i.e. the message of the commit identified from the previous step. The commit messages indicate why the changes were made to the corresponding files (Alali et al. 2008). *Third*, if the commit message included a unique identifier that maps the commit to an issue in the issue tracking system, we extract the identifier and use that identifier to extract the summary of the issue. We use regular expression and the corresponding issue tracking API to extract the issue identifier. *Fourth*, we combine the commit message with any existing issue summary to construct the message for analysis. We refer to the combined message as 'extended commit message (ECM)' throughout the rest of the paper. We use the extracted ECMs to separate the defect-related commits from the non defect-related commits, as described in Section 3.2.3.

3.2.3 Determining Defect-Related Commits

We use defect-related commits to identify the defective IaC scripts. We apply qualitative analysis to determine which commits are defect-related commits using the following three steps:

Categorization Phase At least two raters with software engineering experience determine which of the collected commits are defect-related. We adopt this approach to mitigate the subjectivity introduced by a single rater. Each rater determine an ECM as defect-related if the ECM suggests a defect-related action is taken. For reference, we provide raters with an electronic handbook on IaC (Labs 2018) and the IEEE publication on anomaly classification (IEEE 2010). The number of ECMs to which we observe agreements amongst the raters should be recorded and the Cohen's Kappa (Cohen 1960) score should be computed.

Resolution Phase Raters can disagree if a commit is defect-related. To resolve such disagreements, we use an additional rater who we refer to as the 'resolver'. Upon completion of this step, we can classify which commits and ECMs are defect-related. We determine a script to be defective if the script is modified in a defect-related commit.

Member Checking To evaluate the ratings of the raters in the categorization and the resolution phase, we randomly select 50 ECMs for each dataset. We contact practitioners and ask if they agree to our categorization of ECMs. High agreement between the raters' categorization and practitioners' feedback is an indication of how well the raters performed. The percentage of ECMs to which practitioners agreed upon and the Cohen's Kappa score should be computed.

3.3 Summary of Dataset Construction

We apply the three selection criteria presented in Section 3.2.1 to identify repositories for our analysis. We describe how many of the repositories satisfied each of the three criterion in Table 1. Each row corresponds to the count of repositories that satisfy each criterion. For example for Mirantis 26 repositories satisfy Criterion-1. Altogether, we obtain 94 repositories to extract Puppet scripts from.

We report summary statistics on the collected repositories in Table 2. For example, for Mirantis we collect 180 Puppet scripts that map to 1,021 commits. Of these 1,021 commits, 82 commits include identifiers for bug reports. We identify 33.7% of the 1,021 commits as defect-related commits, and 53.3% of the 180 scripts as defective.

We categorize ECMs to classify which collected commits are defect-related, using the following three phases.

Categorization Phase: We describe the categorization phase for the four datasets:

Mirantis: We recruit students in a graduate course related to software engineering titled 'Software Security' via e-mail. The course was conducted in Fall 2017 at North Carolina State University. The number of students in the class was 58, and 32 students agreed to participate. The average experience of the 32 students in software engineering is two years. On average, each student took 2.1 hours to categorize the 200 ECMs. We randomly distribute the 1,021 ECMs amongst the students such that each ECM is rated by at least two students.

Mozilla: Two graduate students separately apply qualitative analysis on 3,074 ECMs. The first and second rater, respectively, have experience in software engineering of three and two years. The first and second rater, respectively, took 37.0 and 51.2 hours to complete the categorization.

Openstack: Two graduate students, separately, apply qualitative analysis on 7,808 ECMs from Openstack repositories. The first and second rater, respectively, have software engineering experience of two and one years. The first and second rater completed the categorization of the 7,808 ECMs, respectively, in 80 and 130 hours.

Wikimedia: We recruit students in a graduate course related to software engineering titled 'Software Security' via e-mail. The course was conducted in Fall 2016 at North Carolina State University. The number of students in the class was 74, and 54 students agreed to participate. We randomly distribute the 972 ECMs amongst the students such that each ECM is rated by at least two students. The average experience of the 54 students in software engineering is 2.3 years. On average, each student took 2.1 hours to categorize the 140 ECMs.

Table 1 Selection Criteria to Construct Defect Datasets

Criteria	Mirantis	Mozilla	Openstack	Wikimedia
Criterion-1	26	1,594	1,253	1,638
Criterion-2	20	2	61	11
Criterion-3	20	2	61	11
Final	20	2	61	11

Table 2 Statistics of Four Datasets

Attributes	Mirantis	Mozilla	Openstack	Wikimedia
Puppet Scripts	180	299	1,363	296
Commits with Puppet Scripts	1,021	3,074	7,808	972
Commits with Report IDs	82 of 1021, 8.0%	2764 of 3074, 89.9%	2252 of 7808, 28.8%	210 of 972, 21.6%
Defect-related Commits	344 of 1021, 33.7%	558 of 3074, 18.1%	1987 of 7808, 25.4%	298 of 972, 30.6%
Defective Puppet Scripts	96 of 180, 53.3%	137 of 299, 45.8%	793 of 1363, 58.2%	160 of 296, 54.0%

Table 3 Member Checking Phase

Property	Mirantis	Mozilla	Openstack	Wikimedia
Contacts	5	6	10	7
Agreement	91.0%	94.0%	92.0%	98.0%
Cohen's κ	0.8	0.9	0.8	0.9
Interpretation	Substantial	Almost Perfect	Substantial	Almost Perfect

Resolution Phase The first author of the paper is the resolver who resolved disagreements for all four datasets. The Cohen's Kappa is 0.5, 0.6, 0.5, and 0.7, respectively, for Mirantis, Mozilla, Openstack, and Wikimedia. Based on Landis and Koch's interpretation (Landis and Koch 1977), we observe 'Fair', 'Moderate', 'Fair' and 'Substantial' agreement amongst raters for the datasets Mirantis, Mozilla, Openstack, and Wikimedia.

Member Checking Following our methodology in Section 3.2.3, we report the agreement level between the raters' and the practitioners' categorization for 50 randomly-selected ECMs in Table 3. The 'Contacts' row presents how many developers we contacted. All contacted practitioners responded. We report the agreement level and Cohen's Kappa score for 50 ECMs respectively, in 'Agreement' and 'Cohen's κ' rows.

We observe that the agreement between ours and the practitioners' categorization varies from 0.8 to 0.9, which is higher than that of the agreement between the raters in the Categorization Phase. One possible explanation can be related to how the resolver resolved the disagreements. The first author of the paper has industry experience in writing IaC scripts, which may help to determine categorizations that are consistent with practitioners, potentially leading to higher agreement. Another possible explanation can be related to the sample provided to the practitioners: the provided sample, though randomly selected, may include commit messages whose categorization are relatively easy to agree upon.

Finally upon applying qualitative analysis, we identify defect-related commits. These defect-related commits list the changed Puppet scripts which we use to identify the defective Puppet scripts. We present the count of defect-related commits and defective Puppet scripts in Table 2.

Dataset Constructed datasets are available online.[3]

3.4 Metrics

We consider seven development activity metrics: developer count, disjointness in developer groups, highest contributor's code, minor contributors, normalized commit size, scatteredness, and unfocused contribution. We consider these metrics for two reasons: (i) the metrics can provide actionable advice; and (ii) the metrics can be mined from the 94 repositories collected in Section 3. The criterion to determine metric actionability is "*a metric has actionability if it allows a software manager to make an empirically informed decision*" (Meneely et al. 2013).

[3] https://figshare.com/s/c88ece116f803c09beb6

We obtain these metrics by performing a scoping review (Munn et al. 2018; Arksey H and O'Malley L 2005; Anderson et al. 2008) of International Conference on Software Engineering (ICSE) to identify papers published from 2008 to 2018 that are related to defect prediction, and identify papers published in the Conference on Computer and Communications Security (CCS) that are related to vulnerability prediction. We select these two conferences as ICSE and CCS are considered the premier venues, respectively, for software engineering and computer security. We apply scoping review, which is a reduced form of systematic literature review. In our paper, we investigate what development activities might be mentioned in prior literature, and scoping reviews can provide guidance on what development activity metrics could be useful for defect and vulnerability prediction. The first author conducted the scoping review. Upon application of scoping review, we identify seven metrics, each of which are described below with appropriate references:

Developer count Similar to prior research (Meneely and Williams 2009), we hypothesize defective scripts will be modified by more developers compared to that of neutral scripts. Developer count is actionable because the metric can provide actionable advice for software managers on how many developers could be allocated to an IaC script. Furthermore, using this metric practitioners can perform additional review and testing on scripts that are modified by many developers.

Disjointness in developer groups Disjointness measures how separate one developer group is to another. A developer group is a set of developers who work on the same set of scripts (Meneely and Williams 2009). Disjointness in developer groups is actionable because the metric can provide actionable advice for software managers on whether developers should work disjointly or with collaboration. We hypothesize that defective IaC scripts will be modified by developer groups that exhibit more disjointness compared to that of neutral scripts. Similar to prior research (Meneely and Williams 2009), with (1) we use the maximum of edge betweenness metric (MAX_EDG_BTW) in the developer network to measure the amount of disjointness between developer groups.

We construct the graph with nodes, where each node is a developer. We only add an edge between two nodes if the same script is modified by two developers. Figure 3 shows a developer network. As an example, in Fig. 3, script $S1$ is modified by $Dev4$ and $Dev5$. Our constructed developer network is an undirected, unweighted graph where each node corresponds to a developer. Each edge in the developer network connects two nodes if the same script is modified by developers that correspond to the two nodes of interest. In the developer network, a MAX_EDG_BTW metric can tell empirically the groups of developers who are working together on the same script. We hypothesise that a defective script will have more developer groups hwo are disjoint compared to that of neutral scripts. For example, a script modified by two developer groups with a MAX_EDG_BTW of 0.6 is likely to be more defective compared to that of a script, with MAX_EDG_BTW of 0.2.

$$MAX\_EDG\_BTW(e) = max \sum_{(a,b) \neq e} \frac{\text{\# shortest paths between a and b that pass through e}}{\text{\# shortest paths between a and b}} \tag{1}$$

Highest contributor's code The highest contributor is the developer who authored the highest number of lines of code (LOC) for an IaC script. Highest contributor's code is actionable because the metric can provide actionable advice for software managers

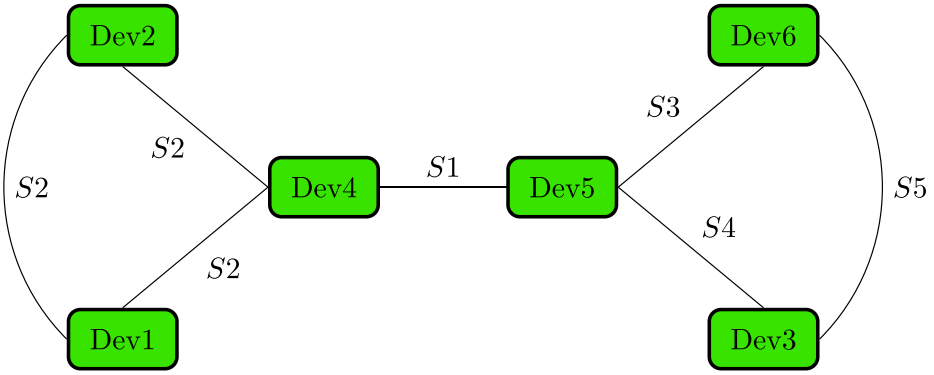

Fig. 3 A hypothetical example to demonstrate our calculation of disjointness between developers

on whether or not contribution amount matters to the quality of IaC scripts. Similar to prior research (Rahman and Devanbu 2013b), we hypothesize that the highest contributor's code is significantly smaller in defective scripts compared to that of neutral scripts, which implies that developers who do not contribute majority of the scripts could introduce defects in IaC scripts. We calculate the contribution by the highest contributor ($HIGHEST\_CONTRIB\_CODE$) using (2).

$$HIGHEST\_CONTRIB\_CODE = \frac{\text{number of lines authored by the highest contributor}}{\text{lines of code}} \quad (2)$$

Minor contributors Minor contributors is a subset of the developers who modify $\leq$ 5% of total code for a script. Minor contributors is actionable because the metric can provide actionable advice for software managers on whether or not contribution amount matters to the quality of IaC scripts. Prior research (Rahman and Devanbu 2013b) reported that scripts that are modified by minor contributors are more susceptible to defects. We hypothesize that the number of minor contributors are significantly larger for defective scripts compared to that of neutral scripts.

Normalized commit size Rahman and Devanbu (2013b) used normalized commit size to construct defect prediction models. Normalized commit size is actionable because the metric can provide actionable advice for software managers on whether IaC scripts should be developed by making small-sized commits or large-sized commits. We hypothesize that large-sized commits are more likely to appear in defective scripts than neutral scripts. We measure commit size using (3), where we compute total lines added and deleted in all commits, and normalize by total commit count for a script.

$$Norm\_commit\_size = \frac{\sum_{i=1}^{C} \text{Total lines added/deleted in commit } i}{\text{Total number of commits}} \quad (3)$$

Scatteredness Scatteredness is actionable because the metric can provide actionable advice for software managers on whether or not submitted code changes should be spread out across a script or should be grouped together in a specific location of a script. Based upon findings from Hassan (2009), we hypothesize defective scripts will include more changes that are spread throughout the script when compared to neutral scripts. In (4), we calculate x_i, which we use in (5) to quantify scatteredness of a script.

As an example, let us assume *Script*#1 has 10 LOC and six commits. *Script*#2 has seven LOC with four commits. For *Script*#1, three modifications are made to line#6 and 7 each. For *Script*#2, line#1, 2, 6, and 7 are modified once. According to (5), the scatteredness score for *Script*#1 and *Script*#2 are respectively, 0.8 and 2.0.

$$x_i = \frac{\text{number of times line } i \text{ is modified}}{\text{number of commits in the script}} \tag{4}$$

$$Scatteredness = -\sum_{i=1}^{N} (x_i log_2 x_i) \tag{5}$$

Unfocused contribution Unfocused contribution occurs when a script is changed by many developers who are also making many changes to other scripts (Pinzger et al. 2008). Prior research has reported unfocused contribution to be related with software vulnerabilities (Meneely and Williams 2009). Unfocused contribution is actionable because the metric can provide actionable advice for software managers on whether or not multiple developers should modify multiple scripts. Unfocused contribution can also help practitioners identify scripts that require further inspection and testing. We hypothesize that defective scripts will be modified more through unfocused contributions compared to that of neutral scripts.

Similar to prior work (Meneely and Williams 2009), we use a graph called a contribution network, which is an un-directed, weighted graph. As shown in Fig. 4, two types of nodes exist in a contribution network: script (circle) and developer (rectangle). When a developer modifies a script, an edge between that developer and that script will exist. No edges are allowed between developers or between scripts. The number of commits the developer makes to a script is the weight for that edge. While determining unfocused contribution we use shortest paths that pass through a script. While determining shortest paths edge weights are accounted.

We use the betweenness centrality metric ($BETW\_CENT$) of the contribution network to measure unfocused contribution using (6). Equation 6 presents the formula to compute betweenness centrality for script x, where a and b are developer nodes. A script with high betweenness indicates that the script has been modified by many developers who have made changes to other scripts.

We use Fig. 4 to provide an example for our betweenness centrality metric using S5. According to Fig. 4, $S5$ is modified by two developers, $Dev3$ and $Dev6$. There are two paths that connect $Dev3$ and $Dev6$: (i) $Dev3-S5-Dev6$, and (ii) $Dev3-S4-Dev5-S3-Dev6$ with a path length of respectively, 3 and 6. Only one shortest path exists between $Dev3$ and $Dev6$, which is $Dev3-S5-Dev6$. Only one shortest path exists between $Dev3$ and $Dev6$ that also involves $S5$, which is $Dev3-S5-Dev6$. According to (6), $BETW\_CENT$ for script $S5$ is 1/1 = 1.0.

$$BETW\_CENT\ (\text{x}) = \sum_{a \neq x \neq b} \frac{\text{no. of shortest paths between a and b that pass through x}}{\text{no. of shortest paths between a and b}} \tag{6}$$

4 RQ1: How are Development Activity Metrics Quantitatively Related with Defective Infrastructure as Code Scripts?

In Sections 4.1 and 4.2 we respectively, provide the methodology and results to answer RQ1.

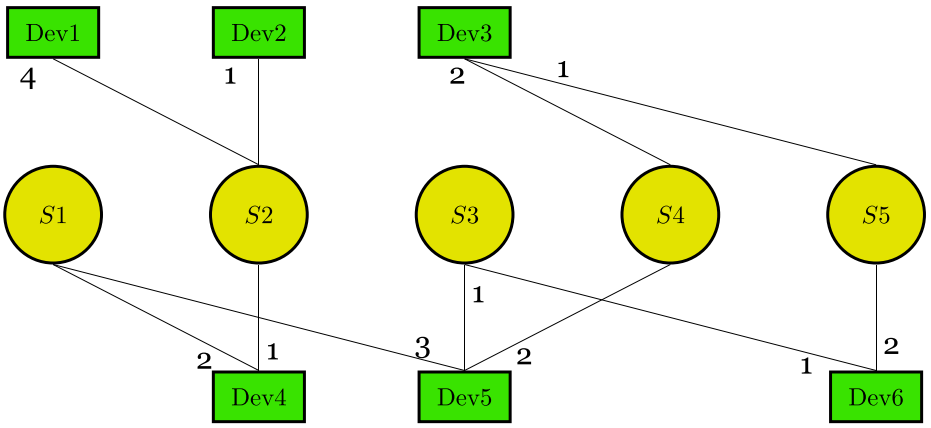

Fig. 4 A hypothetical example to demonstrate our calculation of unfocused contribution

4.1 Methodology to Answer RQ1

We answer RQ1 by quantifying the relationship between each metric and defective IaC scripts. To determine the relationship we compare distributions instead of correlation analysis. Correlation analysis involves specifying cutoffs to specify if two features are strongly correlated or not (Tan et al. 2005). Application of correlation would require us, the authors, to decide if a development activity metric is strongly or weakly correlated with defects in IaC scripts. Using our judgement to determine correlations is susceptible to bias, and hence, we selected distribution comparison to determine the relationship between each metric and defective scripts.

We use the seven development activity metrics identified from Section 3.4. For each metric, we compare the distribution for that metric between defective and neutral scripts.

For metrics developer count, disjointness in developer groups, minor contributors, normalized commit size, scatteredness, and unfocused contribution, we state the following null and alternate hypothesis:

- *Null*: the metric is not larger for defective scripts than neutral scripts.
- *Alternate*: the metric is larger for defective scripts than neutral scripts.

For highest contributor's code, we state the following null and alternate hypothesis:

- *Null*: highest contributor's code is not larger for neutral scripts than defective scripts.
- *Alternate*: highest contributor's code is larger for neutral scripts than defective scripts.

We reject the null hypothesis for the seven metrics using two approaches:

- Mann Whitney U Test: We use the Mann-Whitney U test (Mann and Whitney 1947) to compare the metric values for defective and neutral scripts. The Mann-Whitney U test is non-parametric compares two distributions and states if one distribution is significantly larger or smaller than the other.
- One way multivariate analysis of variance (OMANOVA): One Way Multivariate analysis of variance (OMANOVA) is a statistical approach to compare multivariate means (Huberty and Olejnik 2006). OMANOVA is used to measure the impact of one or more independent variables on two or more dependent variables or outcomes. To

measure the impact the OMANOVA test compares if the means for each independent variable is significantly different for the multiple outcomes. If the mean is significantly different, then we can conclude that a correlation exists between a certain independent variable and the outcome. The significance of difference in means is quantified using p-values. In OMANOVA analysis the impact of multiple independent variables can be tested simultaneously. While measuring the impact for one independent variable, the impact of other independent variables is accounted by computing how significantly different the mean values are.

We apply OMANOVA to sanity check the results from Mann Whitney U test as size and age of scripts can also impact our two possible outcomes: defective script and non-defective script. Using OMANOVA we can measure the impact for each of the seven metrics on our dependent variable. We can apply OMANOVA as it works for dependent variables with two or more outcomes.

OMANOVA expects independent variables to be normally distributed. If an independent variable is not normal then we apply $log_e(x + 1)$ to transform the independent variables. We repeat the OMANOVA analysis for the seven metrics, where we also control for two additional independent variables: size and age of a script. We measure size and age of a script, respectively, in terms of lines of code and months between the first and last commit for the script. In our OMANOVA tests, if the mean for one of the seven metrics is significantly different even after including size and age, then we can conclude that our metric has a correlation with the outcomes even after considering the impact of size and age.

Following Cramer and Howitt's observations (Cramer and Howitt 2004), for both Mann-Whitney U test and MANOVA we determine a metric to be significantly different if $p < 0.01$.

We compute effect size using Cliff's Delta (1993) for metrics that are significantly different between defective and non-defective scripts. Cliff's Delta is a non-parametric test (Cliff 1993) to compare the distribution of each metric between defective and neutral scripts. The Mann-Whitney U test shows if a relationship exists, but does not reveal the magnitude of differences (Sullivan and Feinn 2012). Effect size shows the magnitude of differences between two distributions. We use Romano et al.'s (2006) recommendations to interpret the observed Cliff's Delta values: the difference between two groups is 'large' if Cliff's Delta is greater than 0.47. A Cliff's Delta value between 0.33 and 0.47 indicates a 'medium' difference. A Cliff's Delta value between 0.14 and 0.33 indicates a 'small' difference. Finally, a Cliff's Delta value less than 0.14 indicates a 'negligible' difference.

4.2 Statistical Analysis Results

We organize the section in the following two subsections: first, we provide summary of metrics for all datasets in Section 4.2.1. Next, we list the development activity metrics that relate with defective IaC scripts in Section 4.2.2.

4.2.1 Metric Summary for all Datasets

We first provide minimum, maximum, and standard deviation of each metric for Mirantis, Mozilla, Openstack, and Wikimedia, respectively in Tables 4, 5, 6, and 7.

Table 4 Distribution of metrics for neutral and defective scripts in the Mirantis Dataset

Property	Defective	Neutral
Developer count	(1.0, 9.0, 1.2)	(1.0, 3.0, 0.6)
Disjointness in dev. groups	(0.0, 0.5, 0.1)	(0.0, 0.4, 0.2)
Highest contrib. code	(0.3, 1.0, 0.2)	(0.0, 1.0, 0.2)
Minor contributors	(0.0, 6.0, 0.8)	(0.0, 1.0, 0.2)
Norm_commit_size	(4.3, 191.7, 23.0)	(0.0, 102.5, 21.0)
Scatteredness	(0.0, 5.9, 2.1)	(0.0, 4.8, 1.6)
Unfocused contribution	(1.0, 9.0, 1.1)	(0.0, 3.0, 0.5)

Each tuple expresses the minimum, maximum, and standard deviation for each metric for both neutral and defective scripts

Table 5 Distribution of metrics for neutral and defective scripts in the Mozilla Dataset

Property	Defective	Neutral
Developer count	(1.0, 25.0, 4.3)	(1.0, 8.0, 1.3)
Disjointness in dev. groups	(0.0, 0.6, 0.2)	(0.0, 0.4, 0.2)
Highest contrib. code	(0.2, 1.0, 0.2)	(0.0, 1.0, 0.2)
Minor contributors	(0.0, 12.0, 2.2)	(0.0, 6.0, 0.6)
Norm_commit_size	(0.0, 65.0, 9.7)	(2.5, 62.3, 10.5)
Scatteredness	(0.0, 4.4, 1.0)	(0.2, 1.0, 0.2)
Unfocused contribution	(0.0, 5.0, 1.0)	(1.0, 8.0, 3.1)

Each tuple expresses the minimum, maximum, and standard deviation for each metric for both neutral and defective scripts

Table 6 Distribution of metrics for neutral and defective scripts in the Openstack Dataset

Property	Defective	Neutral
Developer count	(1.0, 43.0, 4.0)	(1, 11, 1.4)
Disjointness in dev. groups	(0.0, 0.5, 0.1)	(0.0, 0.5, 0.2)
Highest contrib. code	(0.1, 1.0, 0.2)	(0.2, 1.0, 0.2)
Minor contributors	(0.0, 36.0, 3.1)	(0.0, 7.0, 0.8)
Norm_commit_size	(0.4, 277.0, 31.3)	(0.3, 207.0, 24.4)
Scatteredness	(0.0, 6.7, 1.7)	(0.0, 6.2, 1.2)
Unfocused contribution	(1.0, 42.9, 4.0)	(1, 11, 1.3)

Each tuple expresses the minimum, maximum, and standard deviation for each metric for both neutral and defective scripts

Table 7 Distribution of metrics for neutral and defective scripts in the Wikimedia Dataset

Property	Defective	Neutral
Developer count	(1.0, 11.0, 1.7)	(1.0, 5.0, 0.8)
Disjointness in dev. groups	(0.0, 0.5, 0.2)	(0.0, 0.4, 0.2)
Highest contrib. code	(0.2, 1.0, 0.2)	(0.4, 1.0, 0.1)
Minor contributors	(0.0, 7.0, 1.1)	(0.0, 2.0, 0.4)
Norm_commit_size	(3.0, 170.5, 18.2)	(3.0, 135.5, 15.5)
Scatteredness	(0.0, 5.9, 1.6)	(0.0, 5.5, 1.4)
Unfocused contribution	(1.0, 11.0, 1.5)	(1.0, 5.0, 0.7)

Each tuple expresses the minimum, maximum, and standard deviation for each metric for both neutral and defective scripts

4.2.2 Metrics that Rzelate with Defective Scripts

In this section we provide the statistical test results for the seven development activity metrics.

Mann-Whitney U Test Results We use Mann-Whitney U Test to compare the distribution of each metric between defective and neutral scripts, and report the relationship between metrics and defective scripts. According to Table 8, we observe a relationship to exist for five of the seven metrics: developer count, minor contributors, highest contributor's code, disjointness in developer groups, and unfocused contribution. The mean values of defective and neutral scripts are presented in the '(DE, NE)' column. The metrics for which $p < 0.01$, as determined by the Mann Whitney U test is indicated in grey cells. The metrics for which $p < 0.01$ for all datasets are followed by a star symbol (✪). We report the p-values in Table 9.

In Table 10, we present the Cliff's Delta values. Along with Cliff's Delta values we report Romano et al.'s (2006) interpretation of Cliff's Delta values: 'N', 'S', 'M', and 'L' respectively indicates 'negligible', 'small', 'medium', and 'large' difference. As indicated in bold, the metrics for which we observe 'medium' or 'large' differences between defective and neutral scripts across all datasets are: developer count and unfocused contribution. In short,

Table 8 Mean values for development activity metrics. The metrics for which $p < 0.01$ for all datasets are followed by a star symbol (✪)

Metric	MIR	MOZ	OST	WIK
	DE, NE	DE, NE	DE, NE	DE, NE
Developer count ✪	2.5, 1.5	4.1, 2.1	4.3, 2.2	2.6, 1.6
Disjointness in dev. groups ✪	0.4, 0.2	0.4, 0.3	0.4, 0.3	0.3, 0.2
Highest contrib. code ✪	0.7, 0.8	0.7, 0.8	0.6, 0.8	0.8, 0.9
Minor contributors ✪	0.4, 0.0	1.1, 0.2	1.6, 0.4	0.6, 0.1
Norm_commit_size	26.5, 24.1	14.0, 12.4	27.8, 28.3	18.8, 15.6
Scatteredness	2.6, 2.6	2.9, 2.2	3.4, 3.3	3.0, 2.4
Unfocused contribution ✪	2.5, 1.5	3.4, 1.8	4.3, 2.2	2.6, 1.6

Table 9 $p - value$ for development activity metrics. The metrics for which $p < 0.01$ for all datasets are followed by a star symbol (✪)

Metric	MIR	MOZ	OST	WIK
Developer count ✪	2.1×10^{-12}	3.8×10^{-8}	1.5×10^{-38}	6.4×10^{-9}
Disjointness in dev. groups ✪	4.9×10^{-10}	5.6×10^{-5}	1.2×10^{-21}	2.3×10^{-5}
Highest contrib. code ✪	8.7×10^{-7}	0.0001	2.3×10^{-31}	6.9×10^{-5}
Minor contributors ✪	4.7×10^{-6}	2.4×10^{-7}	4.2×10^{-26}	3.9×10^{-8}
Norm_commit_size	0.08	0.02	0.96	0.003
Scatteredness	0.2	8.0×10^{-9}	7.2×10^{-7}	8.5×10^{-7}
Unfocused contribution ✪	2.0×10^{-11}	1.6×10^{-8}	1.4×10^{-37}	6.3×10^{-8}

relationship exists for five metrics, but the difference in metrics is large or medium for two metrics: developers and unfocused contribution, which suggests that for these two metrics the differences are more observable than the other three metrics. Practitioners can use the reported effect size measures as a strategy to prioritize which development anti-patterns they may act upon. For example, as the difference is 'large' or 'medium' for developer count, practitioners may thoroughly inspect IaC scripts modified by multiple developers (Table 9).

From Table 8 we observe, on average, developer count is two times higher for defective scripts than neutral scripts. Figure 5 presents the minimum and maximum number of developers who modify defective and neutral scripts. According to Fig. 5, a neutral script may be modified by at most 11 developers. From Table 8 we also observe minor contributors to relate with defective scripts.

Figure 6 presents the minimum and maximum number of minor contributors who modify defective and neutral scripts. We observe a neutral script may be modified by at most seven minor contributors. Also from Table 11, we observe the minimum and maximum number of minor contributors for defective scripts are modified by ≥ 8 minor contributors, and can be modified as many as 36 minor contributors. A complete breakdown of minimum and maximum values for developers and minor contributors is presented in Table 11.

OMANOVA Results We provide results of our OMANOVA analysis in Tables 12, 13, 14, 15, 16, 17, and 18. In each of these tables we report the p-values, which show if the mean values are significantly different between defective and non-defective scripts. For example,

Table 10 Effect size values for development activity metrics. The metrics for which $p < 0.01$ for all datasets are followed by a star symbol (✪)

Metric	MIR	MOZ	OST	WIK
Developer count ✪	0.55 (L)	0.35 (M)	0.40 (M)	0.36 (M)
Disjointness in dev. groups ✪	0.42 (M)	0.21 (S)	0.21 (S)	0.22 (S)
Highest contrib. code ✪	0.41 (M)	0.24 (S)	0.37 (M)	0.25 (S)
Minor contributors ✪	0.26 (S)	0.28 (S)	0.30 (S)	0.27 (S)
Norm_commit_size	0.12 (N)	0.13 (N)	0.06 (N)	0.18 (S)
Scatteredness	0.06 (N)	0.38 (M)	0.15 (S)	0.32 (S)
Unfocused contribution ✪	0.56 (L)	0.35 (M)	0.40 (M)	0.36 (M)

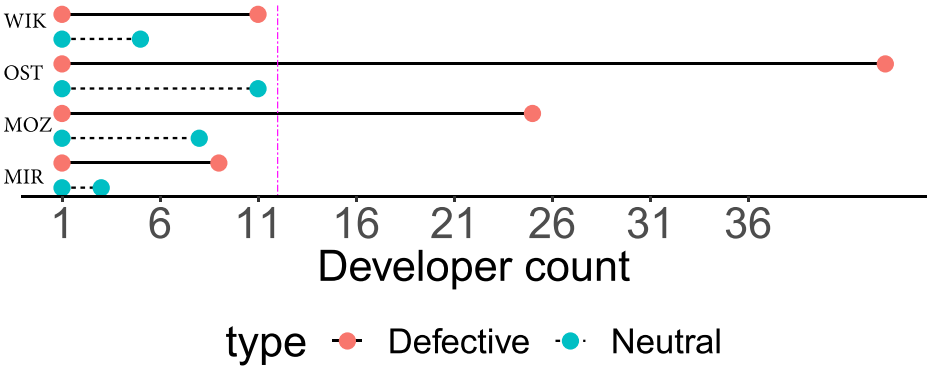

Fig. 5 Count of developers modifying scripts. Neutral scripts are modified by no more than 11 developers

in Table 12 p-value is < 0.01 for size, age, and developer count, indicating size and age to correlate with defective scripts along with developer count.

For all four datasets, the development activity metrics that correlate with defective scripts even when the effect of size and age is considered are: developer count, disjointness in developer groups, highest contributor code, minor contributors, and unfocused contributions. Our OMANOVA analysis is consistent with our Mann Whitney U Test results.

Answer to RQ1: Based on quantitative analysis, metrics related with defective scripts are: developer count, disjointness in developer groups, highest contributor's code, minor contributors, and unfocused contribution.

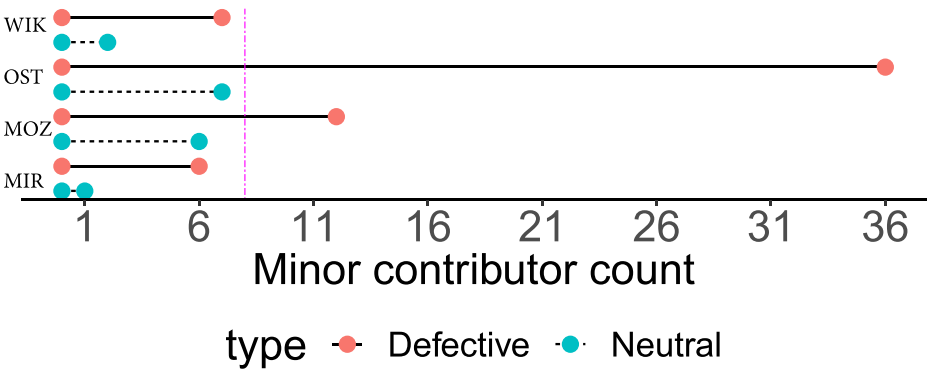

Fig. 6 No. of minor contributors modifying scripts. Neutral scripts are modified by no more than seven minor contributors

Table 11 Minimum and maximum values for developer count and minor contributor count for defective and neutral IaC scripts

Dataset	Developers		Minor contributors	
	Neutral	Defective	Neutral	Defective
Mirantis	(1, 3)	(1, 9)	(0, 6)	(0, 1)
Mozilla	(1, 8)	(1, 25)	(0, 6)	(0, 12)
Openstack	(1, 11)	(1, 43)	(0, 7)	(0, 36)
Wikimedia	(1, 5)	(1, 11)	(0, 2)	(0, 7)

Table 12 OMANOVA results for all datasets when effect of size and age is considered with developer count

Metric	MIR	MOZ	OST	WIK
Size	3.4×10^{-8}	$< 2.2 \times 10^{-16}$	9.5×10^{-11}	8.5×10^{-15}
Age	3.2×10^{-13}	3.5×10^{-7}	$< 2.2 \times 10^{-16}$	7.3×10^{-7}
Developer count	9.3×10^{-13}	1.6×10^{-9}	$< 2.2 \times 10^{-16}$	2.2×10^{-9}

Developer count is significantly larger for defective scripts compared to neutral scripts for all datasets even when the effect of size and age is considered. Each cell represents a p-value for the metric

Table 13 OMANOVA results for all datasets when effect of size and age is considered with disjointness in developer groups

Metric	MIR	MOZ	OST	WIK
Size	3.4×10^{-8}	$< 2.2 \times 10^{-16}$	9.5×10^{-11}	8.5×10^{-15}
Age	3.2×10^{-13}	3.5×10^{-7}	$< 2.2 \times 10^{-16}$	7.3×10^{-7}
Disjointness in dev. groups	1.1×10^{-10}	9.4×10^{-5}	$< 2.2 \times 10^{-16}$	3.6×10^{-5}

Disjointness in developer groups is significantly larger for defective scripts compared to neutral scripts for all datasets even when the effect of size and age is considered. Each cell represents a p-value for the metric

Table 14 OMANOVA results for all datasets when effect of size and age is considered with highest contributor's code

Metric	MIR	MOZ	OST	WIK
Size	3.4×10^{-8}	$< 2.2 \times 10^{-16}$	9.5×10^{-11}	8.5×10^{-15}
Age	3.3×10^{-13}	3.5×10^{-7}	$< 2.2 \times 10^{-16}$	7.3×10^{-7}
Highest contrib. code	0.008	0.004	$< 2.2 \times 10^{-16}$	0.0007

Highest contributor's code is significantly larger for neutral scripts compared to defective scripts for all datasets even when the effect of size and age is considered. Each cell represents a p-value for the metric

Table 15 OMANOVA results for all datasets when effect of size and age is considered with minor contributors

Metric	MIR	MOZ	OST	WIK
Size	3.4×10^{-8}	$< 2.2 \times 10^{-16}$	9.4×10^{-11}	8.6×10^{-15}
Age	3.3×10^{-13}	3.5×10^{-7}	$< 2.2 \times 10^{-16}$	7.3×10^{-7}
Minor contributors	9.2×10^{-6}	4.6×10^{-8}	$< 2.2 \times 10^{-16}$	4.2×10^{-8}

Minor contributors is significantly larger for defective scripts compared to neutral scripts for all datasets even when the effect of size and age is considered. Each cell represents a p-value for the metric

Table 16 OMANOVA results for all datasets when effect of size and age is considered with normalized commit size

Metric	MIR	MOZ	OST	WIK
Size	3.4×10^{-8}	$< 2.2 \times 10^{-16}$	9.5×10^{-11}	8.5×10^{-15}
Age	3.2×10^{-13}	3.5×10^{-7}	$< 2.2 \times 10^{-16}$	7.3×10^{-7}
Norm_commit_size	0.03	0.06	0.26	0.01

Normalized commit size is significantly larger for defective scripts compared to neutral scripts for none of the datasets when the effect of size and age is considered. Each cell represents a p-value for the metric

Table 17 OMANOVA results for all datasets when effect of size and age is considered with scatteredness

Metric	MIR	MOZ	OST	WIK
Size	3.4×10^{-8}	$< 2.2 \times 10^{-16}$	9.5×10^{-11}	8.5×10^{-15}
Age	3.2×10^{-13}	3.5×10^{-7}	$< 2.2 \times 10^{-16}$	7.3×10^{-7}
Scatteredness	0.25	0.0001	0.04	0.02

Scatteredness is significantly larger for defective scripts compared to neutral scripts for one of the four datasets when the effect of size and age is considered. Each cell represents a p-value for the metric

Table 18 OMANOVA results for all datasets when effect of size and age is considered with unfocused contribution

Metric	MIR	MOZ	OST	WIK
Size	3.4×10^{-8}	$< 2.2 \times 10^{-16}$	9.5×10^{-11}	8.5×10^{-15}
Age	3.2×10^{-13}	3.5×10^{-7}	$< 2.2 \times 10^{-16}$	7.3×10^{-7}
Unfocused contribution	9.3×10^{-13}	4.3×10^{-5}	$< 2.2 \times 10^{-16}$	2.3×10^{-9}

Unfocused contribution is significantly larger for defective scripts compared to neutral scripts for all datasets even when the effect of size and age is considered. Each cell represents a p-value for the metric

5 RQ2: What are Practitioner Perceptions on the Relationship Between Development Activity Metrics and Defective Infrastructure as Code Scripts?

In Sections 5.1 and 5.2 we respectively report the methodology and results for RQ2

5.1 Methodology

We answer RQ2 by using the metrics listed in Section 3.4. We perform two tasks: (i) deploy a survey to practitioners; and (ii) conduct semi-structured interviews. In the survey, we asked the practitioners to what extent they agreed with a relationship between each of the seven development activity metric and defective IaC scripts. We constructed the survey following Kitchenham and Pfleeger's guidelines (Kitchenham and Pfleeger 2008) as follows: (i) use Likert scale to measure agreement levels: strongly disagree, disagree, neutral, agree, and strongly agree; (ii) add explanations related to the purpose of the study, how to conduct the survey, preservation of confidentiality, relevance to participants, and an estimate of the time to complete the questionnaire; (iii) administer a pilot survey to get initial feedback on the survey. Following Smith et al.'s (2013) observations on software engineering survey incentives, we offer a drawing of five Amazon gift cards as an incentive for participation.

From the pilot study feedback with five graduate students, we add an open-ended question to the survey, where we ask participants to provide any other development activity that is related with quality but not represented in our set of seven metrics. We deploy the survey to 250 practitioners from July 12, 2018 to Jan 15, 2019. We collect developer e-mail addresses from the 94 repositories. Following IRB#12598 protocol,[4] we distribute the survey to practitioners via e-mail.

Semi-structured interview The results from the survey indicates the level of agreement from practitioners, but not the reasons that attribute to that perception. We conducted semi-structured interviews, i.e. an interview where practitioners are asked open-ended and closed questions to identify these reasons. We conduct all interviews over Skype or Google Hangouts. In each interview, we asked the interviewee to what extent they agree on the relationship between each metric and defective script using the Likert scale: strongly disagree, disagree, neutral, agree and strongly agree. Then, we asked about reasons that attribute to their perception. In the end, we asked interviewees if they would like to provide additional information about development activities related to defective scripts, but not included in the survey. We recruited the interviewees from the set of 250 practitioners to which we deployed the survey. We followed the guidelines provided by Hove and Anda (2005) while conducting the interviews. We executed pilot interviews with a voluntary graduate student. During the interviews, we explained the purpose of the interview and explicitly mention that the interviewee's confidentiality will be respected.

Qualitative analysis We transcribe the audio of each interview into textual content. From the textual content of semi-structured interviews, we apply a qualitative analysis technique called descriptive coding (Saldana 2015) to determine reasons that can be attributed to practitioner's perceptions about the relationship between the development activity metric and defective scripts. As shown in Fig. 7, we first identify 'raw text' from the interview. We

[4] https://research.ncsu.edu/sparcs/compliance/irb/using-the-eirb-system/

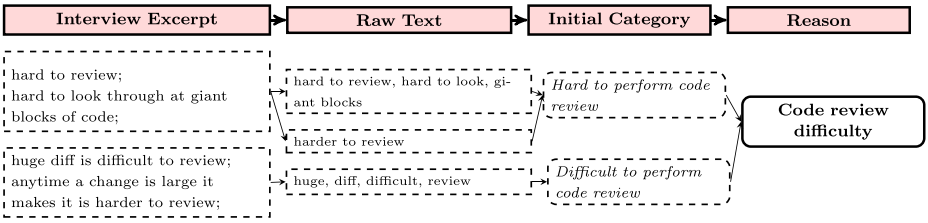

Fig. 7 Example of how we use qualitative analysis to determine reasons that attribute to practitioner perception

extract raw text if any portion of the content provided a reason related to practitioner agreement or disagreement for a specific metric. For agreement, we consider agree and strongly agree. For disagreement, we consider disagree and strongly disagree. Next we generate categories from the codes. Finally, we derive the reasons from the identified categories. We report the reasons along with the corresponding metric.

5.2 Answer to RQ2

We answer RQ2 by providing survey results and results from qualitative analysis in this section.

5.2.1 Survey Results

We obtained a 20.4% response rate for our survey, which is typical for software engineering-based surveys (Smith et al. 2013). The mean experience in Puppet of the survey respondents is 3.9 years. We present the findings in Fig. 8. The percentage values on the right hand side correspond to the percentage of practitioners which responded agree or strongly agree. For example, we observe 84% of practitioners to agree or strongly agree with scatteredness. At least 80% of the practitioners who responded, agreed or strongly agreed that scatteredness and normalized commit size are related with quality of IaC scripts. The least agreed upon metric is unfocused contribution with 45% agreement.

Of the 51 survey respondents, 11 agreed to participate for semi-structured interviews. Of the 11 interviewees, eight were developers, two were DevOps consultants, and one was

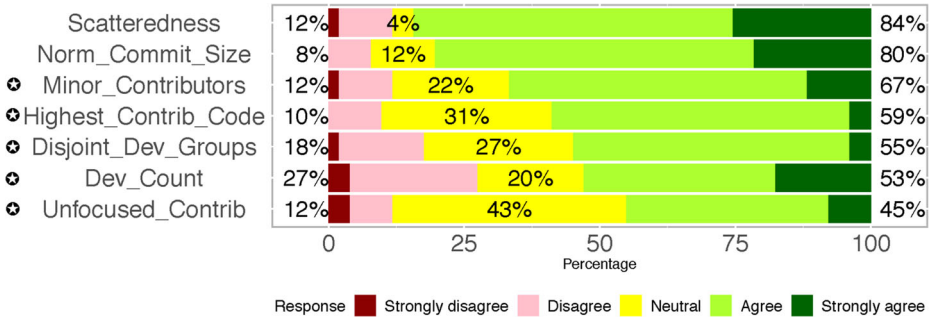

Fig. 8 Survey findings that illustrate to what extent practitioners agree with the seven metrics and their relationships with defective scripts

a project manager. The mean experience of the 11 interviewees is 5.1 years. The duration of interviews varied from 13 to 29 minutes. Through our qualitative analysis of interview content, we identified a set of reasons that attribute to the agreements or disagreements. For the seven metrics, the first and second author, respectively, obtained 8 and 10 reasons that attribute to practitioner perception. The Cohen's Kappa is 0.87, and we resolve the disagreements for the identified reasons by discussing if additional reasons identified by the second author are similar to that of the first author. Upon resolving disagreements, we obtain eight reasons that are identified both by the first author and second author.

5.2.2 Why do Practitioners Agree and Disagree?

The reasons for agreement and disagreement are respectively underlined using a solid and dashed line. The number of interviewees who agree and disagree are enclosed in a parenthesis. A summary of the reasons on why practitioners agree and disagree is listed in Table 19.

Agreement and disagreement reasons reported by practitioners for the seven identified metrics are stated below:

Developer count (8, 3): Developer count and defective scripts are related because of communication problems, as the complexity of maintaining proper communication may increase. One interviewee stated"*obviously the more people have their hand on the code base, and if good communication isn't happening, then they're gonna have different opinions about how things should end up*". Brooks (1995) provided a rule of thumb stating"*if there are n workers working on a project...there are potentially almost* 2^n *teams within which coordination must occur*", indicating adding developers to software development will result in increased communication complexity. Disagreeing interviewees stated when developers work on the same script knowledge sharing happens, which can increase the quality of IaC scripts. One interviewee also reminded about reality:"*I don't think its reasonable to expect that each script is maintained by only a low amount of people. Generally, they [teams] strive to have many people to write as many scripts as possible because that one guy who understands that code can leave*".

Disjointness between developer groups (7, 3): Disjointness between developer groups and defective scripts can be related because of communication problems between the groups. According to agreeing interviewees developers have their own opinions, and without coordination scripts are likely to be defective:"*if I need to hop in a Puppet module for my needs, there should be clear communications around the purpose of the module and how updates should occur, I am gonna make updates to serve my purpose but they*

Table 19 Summary of Agreement and Disagreement Reasons for the Seven Development Activity Metrics

Metric	Agreement	Disagreement
Developer count	communication problem	knowledge sharing
Disjointness between developer groups	communication problem	knowledge sharing
Highest contributor's code	lack of context	knowledge sharing
Minor contributors	lack of context	knowledge sharing
Normalized commit size	code review and debugging difficulty	developer experience
Scatteredness	code review difficulty	lack of context
Unfocused contribution	distraction	bubble problem

may have effects outside the module.". According to disagreeing practitioners, due to knowledge sharing, collaboration between developers, even if in disjoint groups, will lead to increased quality of IaC scripts because collaboration leads to knowledge sharing.
Highest contributor's code (8, 3): Interviewees mentioned lack of context: other developers do not have the full the context as the highest contributor, and if the other developers contribute more than the highest contributor, the script suffers from quality issues. On the contrary, due to knowledge sharing, the highest contributor's code amount may not relate with defective scripts. The highest contributor can share knowledge with other developers on how to modify the script without introducing defects. Such reasoning is consistent with Martin's 'collective ownership' strategy (Martin 2011), which advocates for any team member to make changes to any software artifact.
Minor contributors (8, 3): The relationship between minor contributors and defective scripts can exist because of lack of context: developers who contribute less do not have the full context compared with the developers who contribute the majority of the code. Interviewees disagreed for knowledge sharing, as working on the same script can facilitate sharing of common knowledge needed to modify defects.
Normalized commit size (8, 2): Interviewees stated commit size and defective scripts is related for two reasons: code review difficulty and debugging difficulty. Researchers (MacLeod et al. 2018) have reported large-sized commits are harder to review, and susceptible of introducing defects. Their observation are also supported by anecdotal evidence by industry experts in IaC (Morris 2016), who advise in committing small changes at a time. Furthermore, with large-sized commits, bug localization becomes challenging. One interviewee mentioned"*you should always split your changes ... this [large-sized] commit is not helping you to find when and where the bug was introduced; it is harder to fix and revert*". Reviewing Puppet code can be harder according to one interviewee"*reviewing Puppet code is more challenging than regular code ... if I look for a JavaScript diff I can look side by side and see what the new functions do, whereas with Puppet you have to think it through your head*". Interviewees also mentioned how large-sized commits are introduced. One interviewee mentioned IaC scripts can be authored by system operators who do not have a software development background:"*Ansible and Puppet modules are written by sysadmins and not developers...we have been in this shift where sysadmins become infrastructure developers and the setup of being an admin to a developer is different because there is a difference between a developer and a sysadmin.*". Disagreeing interviewees mentioned developer experience. A developer who is well-versed with the script may be able to submit a large-sized commit without making the script defective.
Scatteredness (7, 3): Code review difficulty is why interviewees perceive scatteredness is related with defective IaC scripts. If the changes are spread throughout the script, developers may miss a defect while reviewing the code changes. On the contrary, three interviewees disagreed stating lack of context as the reason.
Unfocused contribution (7, 4): Distraction is why unfocused contribution is related to defective scripts. Working on multiple scripts can lead to distraction, which could influence the quality of IaC script development. Their reasoning is congruent with Weinberg (1992), who proposed a rule of thumb suggesting a 10% decrease in software quality whenever developers switch between projects. Interviewees who disagreed stated the bubble problem–developers may get stuck in a 'bubble' if they don't work on multiple scripts. When developers work on multiple scripts, they gather knowledge collected from one script and apply it to another script. One interviewee said "*working on one script*

means you are in a bubble, you cannot see what solutions were conceived to solve certain problems".

Additional Development Activities Mentioned by Practitioners During the interview process the practitioners mentioned additional development activities not included in our set of seven metrics. These activities are: not using version control for IaC scripts, development of IaC scripts without design, undocumented code in IaC scripts, not using feature flags, not using gradual rollouts, inadequate logging, and inadequate testing.

Nuanced perspectives At least 80% of the survey respondents agree that large-sized commits and scatteredness are related to defective IaC scripts. Our quantitative analysis reveals that the relationship of these two metrics with defective scripts are not prevalent, and we refrain ourselves from declaring these two activities as anti-patterns. Practitioner perceptions are often formed by their own experiences, which may not always be supported by actual evidence (Devanbu et al. 2016). According to Srikanth and Menzies (C and Menzies 2019), "*documenting developer beliefs should be the start, not the end, of software engineering research. Once prevalent beliefs are found, they should be checked against real-world data*", suggesting researchers to complement developer perception analysis with software repository data. Our findings suggest a nuanced perspective on developers' perception on anti-patterns, as the relationship between normalized commit size and defective scripts is not as prevalent as practitioners perceive (Morris 2016; Oktaba 2015). While considering the use of these development anti-patterns we advise practitioners to consider the nuanced context of perception and quantitative findings.

Answer to RQ2: Practitioners show varying agreement on the relationship between seven development activity metrics and defective IaC scripts. They mostly agree with scatteredness and least agree with unfocused contribution.

6 RQ3: How can we Construct Defect Prediction Models for IaC Scripts using Development Activity Metrics?

Defect prediction models can help practitioners automatically identify scripts that are likely to be defective. Instead of inspecting and testing all IaC scripts used by a team, practitioners from the team can prioritize their inspection and testing efforts for a smaller set of scripts that are likely to be defective. In Sections 6.1 and 6.2 respectively, we provide the methodology and results for RQ3.

6.1 Methodology for RQ3

Our procedure to construct models to predict defective scripts can be summarized as following:

Metric exclusion: From statistical analysis we exclude any metric, which shows no relationship with defective IaC scripts for any of the four datasets. Rest of the metrics are used to construct defect prediction models. Our hypothesis that if a metric shows no quantitative relationship with defective scripts for any of the datasets, then metric the may not be useful in constructing defect prediction models.

Log transformation: We first apply log-transformation on the extracted counts for each source code property. Application of log transformation on numerical features help in prediction (Menzies et al. 2007).
Principal Component Analysis (PCA): We use principal component analysis (PCA) (Tan et al. 2005) to account for multi-collinearity amongst features (Tan et al. 2005). Principal components that account for at least 95% of the total variance are used as input to statistical learners.
Statistical learners: We use four statistical learners to construct prediction models. These learners are classification and regression trees (CART), logistic regression (LR), Naive Bayes (NB), and Random Forest (RF). CART generates a tree based on the impurity measure, and uses that tree to provide decisions based on input features (Breiman and et al 1984). We select CART because this learner does not make any assumption on the distribution of features, and is robust to model overfitting (Tan et al. 2005) (Breiman and et al 1984). LR estimates the probability that a data point belongs to a certain class, given the values of features (Freedman 2005). LR provides good performance for classification if the features are roughly linear (Freedman 2005). We select LR because this learner performs well for classification problems (Freedman 2005) such as defect prediction (Rahman and Devanbu 2013a) and fault prediction (Hall et al. 2012). The NB classification technique computes the posterior probability of each class to make prediction decisions. We select NB because prior research has reported that defect prediction models that use NB perform well (Hall et al. 2012). RF is an ensemble technique that creates multiple classification trees, each of which are generated by taking random subsets of the training data (Breiman 2001; Tan et al. 2005). Unlike LR, RF does not expect features to be linear for good classification performance. Researchers (Ghotra et al. 2015) recommended the use of statistical learners that uses ensemble techniques to build defect prediction models.
10×10-fold evaluation: We use 10×10-fold cross validation to evaluate our prediction models. We use the 10×10-fold cross validation evaluation approach by randomly partitioning the dataset into 10 equal sized subsamples or folds (Tan et al. 2005). The performance of the constructed prediction models are tested by using nine of the 10 folds as training data and the remaining fold as test data. We repeat the process of creating training and test data using 10 folds 10 times.
Parameter tuning of learners: Following (Fu et al. 2016) and Tantithamthavorn wt al. (Tantithamthavorn et al. 2016)'s recommendations we tune the parameters of statistical learners using differential evolution (DE). We select the parameters needed to tune form prior work (Fu et al. 2016).
Comparison with prior approaches: We compare the development activity-based constructed prediction models with three prior approaches that address quality issues in IaC scripts: (i) the bag-of-word (BOW) approach (Rahman and Williams 2018), (ii) implementation smells (Sharma et al. 2016) listed in Table 20, and (iii) script-level quality metrics (van der Bent et al. 2018) listed in Table 21.

We use a variant of the *S*cott *K*nott (SK) test (Tantithamthavorn et al. 2017) to compare prediction performance. This variant of SK does not assume input to be normal, and accounts for negligible effect size (Tantithamthavorn et al. 2017). SK uses hierarchical clustering analysis to partition the input data into significantly ($\alpha = 0.05$) distinct ranks (Tantithamthavorn et al. 2017).
Performance measures: We use three performance metrics to evaluate the constructed prediction models: precision, recall, and F-measure. Precision measures the proportion

Table 20 Implementation Smells (Sharma et al. 2016)

Smell Name	Description
Missing Default Case	The default case is missing
Inconsistent Naming	Names deviates from convention recommended by configuration tool vendors
Complex Expression	The configuration script contains one or many difficult-to-understand complex expressions
Duplicate Entity	The configuration script contains duplicate hash keys or parameters
Misplaced Attribute	Placement of attributes within a resource or a class does not follow a recommended order
Improper Alignment	The code is not properly aligned
Invalid Property Value	The configuration script contains invalid value of a property or an attribute
Incomplete Tasks	The configuration scripts include comments that has 'fixme' and 'todo' as keywords
Deprecated Statement	The configuration script uses one of the deprecated statements
Improper Quote	Single and double quotes are misused in the configuration scripts
Long Statement	The configuration script contains long statements
Incomplete Conditional	A terminating 'else' clause in an if-else block
Unguarded Variable	A variable is not enclosed in braces when being interpolated in a string

of IaC scripts that are actually defective given that the model predicts as defective. Recall measures the proportion of defective IaC scripts that are correctly predicted by the prediction model. F-measure is the harmonic mean of precision and recall.

6.2 Answer to RQ3

As shown in Table 8, we observe all metrics to show a relation with defective IaC scripts for at least one dataset. We do not remove any metrics from our set while constructing prediction models. We report the number of principal components used to construct our defect prediction models in Table 22.

We report the performance of our constructed prediction models to predict defective scripts as following: we compare median precision achieved for 10×10-fold using development activity metrics to that of BOW, code quality, and implementation smells respectively, in Tables 23, 24, and 25. Tables 26, 27, and 28, respectively, compare the median recall

Table 21 Code Quality Metrics (van der Bent et al. 2018)

Quality Metric	Description
Filelength	Lines in a script
Complexity	Total control statements and alternatives in case statements per script
Parameters	Total parameters per script
Execs	Total 'exec' per script
Lint warnings	Lint warnings per script
Fan-in	Incoming dependencies per script

Table 22 Number of Principal Components used for Prediction Models

Dataset	Activity	Code	BOW
Mirantis	4	3	50
Mozilla	4	4	140
Openstack	4	5	400
Wikimedia	3	4	150

Table 23 Comparing Median Precision Between Development Activity Metrics and BOW

Dataset	Development				BOW			
	CART	LR	NB	RF	CART	LR	NB	RF
MIR	0.84	0.69	0.71	0.81	0.62	0.63	0.75	0.72
MOZ	0.73	0.65	0.67	0.82	0.51	0.64	0.63	0.65
OST	0.71	0.67	0.71	0.69	0.63	0.58	0.74	0.57
WIK	0.71	0.68	0.65	0.67	0.60	0.66	0.75	0.76

Table 24 Comparing Median Precision Between Development Activity Metrics and Code Quality Metrics

Dataset	Development				Code Quality			
	CART	LR	NB	RF	CART	LR	NB	RF
MIR	0.84	0.69	0.71	0.81	0.75	0.52	0.51	0.50
MOZ	0.73	0.65	0.67	0.82	0.74	0.53	0.51	0.51
OST	0.71	0.67	0.71	0.69	0.75	0.58	0.74	0.57
WIK	0.71	0.68	0.65	0.67	0.76	0.52	0.61	0.52

Table 25 Comparing Median Precision Between Development Activity Metrics and Implementation Smells

Dataset	Development				Implementation Smell			
	CART	LR	NB	RF	CART	LR	NB	RF
MIR	0.84	0.69	0.71	0.81	0.52	0.66	0.69	0.79
MOZ	0.73	0.65	0.67	0.82	0.60	0.72	0.75	0.71
OST	0.71	0.67	0.71	0.69	0.57	0.65	0.77	0.66
WIK	0.71	0.68	0.65	0.67	0.52	0.65	0.77	0.65

Table 26 Comparing Median Recall between Development Activity Metrics and BOW

Dataset	Development				BOW			
	CART	LR	NB	RF	CART	LR	NB	RF
MIR	0.89	0.86	0.86	0.88	0.69	0.53	0.67	0.84
MOZ	0.70	0.55	0.42	0.70	0.25	0.41	0.51	0.65
OST	0.88	0.76	0.60	0.90	0.62	0.61	0.71	0.70
WIK	0.68	0.59	0.56	0.76	0.65	0.47	0.75	0.80

of models using development activity metrics to that with BOW, code quality metrics, and implementation smells. Finally, Tables 29, 30, and 31 respectively compare the median F-measure of models using development activity metrics to that with BOW, code quality metrics, and implementation smells. For all of above the above-mentioned Tables, the shaded cells indicate the highest median precision as determined by SK test.

With respect to F-measure, development activity metrics are better than the three approaches for three datasets. Using development activity metrics, we construct prediction models that have the highest median F-measure for three datasets. Considering median recall, prediction models with implementation smells provide the best prediction performance for three datasets. With respect to median precision, development activity-based models provide the highest prediction performance for two datasets. Our findings show that prediction models created using development activity metrics are better for predicting which scripts are defective when compared with the BOW, code quality metrics, and implementation smells approaches.

Implications of Prediction Models With respect to money, personnel, and time, inspection and testing of software source files is an expensive procedure, which should be allocated efficiently (Ostrand et al. 2004) (Shihab et al. 2011) (Elberzhager et al. 2012). Practitioners who maintain and develop IaC scripts need to prioritize what IaC scripts need more inspection and testing. Our constructed prediction models lay the groundwork for (i) providing practitioners the opportunity to automatically identify which IaC scripts are likely to have a defect, and (ii) future research that can build on top of our work to quantify usability of IaC defect prediction models. Furthermore, the model is constructed with development activity

Table 27 Comparing Median Recall between Development Activity Metrics and Code Quality Metrics

Dataset	Development				Code Quality			
	CART	LR	NB	RF	CART	LR	NB	RF
MIR	0.89	0.86	0.86	0.88	0.78	0.71	0.86	0.77
MOZ	0.70	0.55	0.42	0.70	0.69	0.59	0.42	0.67
OST	0.88	0.76	0.60	0.90	0.84	0.73	0.60	0.96
WIK	0.68	0.59	0.56	0.76	0.75	0.74	0.56	0.75

Table 28 Comparing Median Recall between Development Activity Metrics and Implementation Smells

Dataset	Development				Implementation Smell			
	CART	LR	NB	RF	CART	LR	NB	RF
MIR	0.89	0.86	0.86	0.88	0.90	0.96	0.23	0.86
MOZ	0.70	0.55	0.42	0.70	0.47	0.41	0.32	0.49
OST	0.88	0.76	0.60	0.90	0.84	0.99	0.09	0.96
WIK	0.68	0.59	0.56	0.76	0.93	0.90	0.05	0.81

metrics, which indicates that regardless of the language the model can be used to predict which scripts are likely to be defective.

Answer to RQ3: With respect to F-measure, defect prediction models constructed with development activity metrics outperform three approaches: bag of words (BOW), implementation smells, and code quality metrics.

7 Discussion

In this section, we discuss our findings by first synthesizing the development activities that relate with defective scripts into development anti-patterns. We also discuss future work in this section.

Development Anti-patterns We identify a set of development anti-patterns based on two criteria: each activity must be (i) related to software quality as reported in prior literature and (ii) supported by quantitative evidence across all four datasets (Section 4.2). We alphabetically list the identified five development anti-patterns with definitions, prior literature that supports an activity, and possible solutions. The presented possible solutions are recommendations from authors that are subject to empirical validation.

Table 29 Comparing Median F-measure between Development Activity Metrics and BOW

Dataset	Development				BOW			
	CART	LR	NB	RF	CART	LR	NB	RF
MIR	0.80	0.77	0.78	0.80	0.66	0.59	0.67	0.77
MOZ	0.68	0.60	0.52	0.71	0.34	0.49	0.49	0.64
OST	0.74	0.71	0.66	0.74	0.62	0.65	0.69	0.67
WIK	0.67	0.63	0.61	0.68	0.63	0.56	0.71	0.72

Anti-pattern#1: Boss is Not Around
Definition: The highest contributor does not author all lines of an IaC script.
Observation: The highest contributor may have the full context of the script, which other developers may not have. If the other developers contribute more than the highest contributor, the corresponding script may be defective. On average, the highest contributor contributes 80%∼90% of the code for neutral scripts.
Prior literature: Bird et al. (2011), Rahman and Devanbu (2013a)
Metric: Highest contributor's code ($HIGHEST\_CONTRIB\_CODE$)
Solution: Perform team planning so that the highest contributor can modify and/or verify all the content added to a script.

Anti-pattern#2: Many Cooks Spoil
Definition: Having multiple developers working on the same script.
Observation: Defective scripts are modified by 12∼43 developers, whereas, neutral scripts modified by no more than 11 developers.
Prior literature: Meneely and Williams (2009), Businge et al. (2017)
Metric: Developer count
Solution: Thoroughly inspect scripts modified by multiple developers.

Anti-pattern#3: Minors are Spoilers
Definition: The activity of a script being modified by developer(s) who writes no more than 5% code of the script.
Observation: A neutral script may be modified by at most seven minor contributors, whereas, defective scripts can be modified by 8∼36 minor contributors.
Prior literature: Bird et al. (2011), Rahman and Devanbu (2013a)
Metric: Minor contributors
Solution: Thoroughly inspect scripts modified by multiple minor contributors.

Anti-pattern#4: Silos
Definition: The activity of developers working in disjoint groups.
Observation: Defective scripts are modified by developer groups, which are 1.3∼2.0 times more disjoint, on average, compared to that of neutral scripts. Even though practitioners perceive IaC as a tool to break silos (Bright 2017), our quantitative findings suggest that silos exist in IaC development.
Prior literature: Meneely and Williams (2009)
Metric: Disjointness in developer groups
Solution: Collaborate more instead of working in disjoint groups in IaC development.

Anti-pattern#5: Unfocused Contribution
Definition: The activity of developers working on an IaC script, who also modify other scripts.
Observation: On average, for defective scripts unfocused contribution is 62.5%∼95.4% higher than neutral scripts.
Prior literature: Meneely and Williams (2009)
Metric: Unfocused contribution
Solution: Perform team planning so that developers can focus on developing one script within a certain time period, instead of multiple developers working simultaneously.

Table 30 Comparing median F-measure between Development Activity Metrics and Code Quality Metrics

Dataset	Development				Code quality			
	CART	LR	NB	RF	CART	LR	NB	RF
MIR	0.80	0.77	0.78	0.80	0.73	0.68	0.71	0.75
MOZ	0.68	0.60	0.52	0.71	0.68	0.65	0.63	0.66
OST	0.74	0.71	0.66	0.74	0.70	0.69	0.59	0.72
WIK	0.67	0.63	0.61	0.68	0.72	0.72	0.63	0.73

Future Work We have identified a list of reasons that attribute to practitioner perception in Section 5.2. Future researchers can investigate to what extent these findings generalize by conducting large-scale quantitative studies. Furthermore, researchers can investigate the synergies between the development activities within themselves.

We have provided a list of possible solutions that are based on author judgement, and not validated by empirical research. Researchers can investigate if our suggested solutions can be validated empirically.

Practitioners have listed a set of development anti-patterns that are not included in our set of five: not using version control for IaC scripts, development of IaC scripts without design, undocumented code in IaC scripts, not using feature flags, not using gradual rollouts, inadequate logging, and inadequate testing. We urge researchers to investigate if practitioner-mentioned anti-patterns are substantiated by empirical evidence. Such investigation can also yield empirical studies that can characterize other development aspects of IaC, such as usage frequency of logging, testing, and version control. For example, researchers can investigate if testing anti-patterns (Garousi and Küçük 2018) and logging anti-patterns (Chen and Jiang 2017) that are applicable for general purpose programming languages, also apply for IaC.

Our prediction performance results from Section 6.2 suggest that one approach i.e., development activity metrics or BOW or code quality metrics is not comprehensive because depending on prediction performance metric, one single approach does not provide the best results. Future researchers can investigate what techniques are needed to obtain prediction models that perform better. Furthermore, researchers can investigate how to build and evaluate prediction models that also includes cost-effective measures as used in defect prediction research (Arisholm et al. 2010).

Table 31 Comparing Median F-measure between Development Activity Metrics and Implementation Smells

Dataset	Development				Implementation Smell			
	CART	LR	NB	RF	CART	LR	NB	RF
MIR	0.80	0.77	0.78	0.80	0.66	0.68	0.31	0.63
MOZ	0.68	0.60	0.52	0.71	0.53	0.53	0.45	0.53
OST	0.74	0.71	0.66	0.74	0.71	0.73	0.16	0.72
WIK	0.67	0.63	0.61	0.68	0.67	0.66	0.09	0.63

8 Threats to Validity

We discuss the limitations of our paper as following:

Conclusion Validity Our findings are dependent on the four datasets, which are constructed by raters. The data construction process is susceptible to human judgment, and the raters' experience can bias the identified defective IaC scripts. The process of determining the reasons for practitioner perception are susceptible to rater bias. In both cases, we mitigate these threats by assigning at least two raters.

Our selection thresholds can be limiting. For example, a repository may contain sufficient amount of Ansible or Chef scripts, but maintained by one practitioner. Such repositories even though active, will be excluded in our analysis based on criteria mention in Section 3.

When developing IaC scripts, other socio-technical factors may contribute to the quality of IaC scripts that are not captured by our statistical analysis and metrics. We mitigate this limitation by conducting a univariate and a multivariate analysis to establish the correlation between the seven development activity metrics and defective scripts.

Construct validity Use of raters to identify defect-related commits is susceptible to mono-method bias, where subjective judgment of raters can influence the findings. We mitigated this threat by using at least two raters and a resolver.

For two datasets, Mirantis and Wikimedia, we used raters who are graduate students who determined defect-related commits as part of their class work. Students who participated in the process can be subject to evaluation apprehension, i.e. consciously or sub-consciously relating their performance with the grades they would achieve for the course. We mitigated this threat by clearly explaining to the students that their rating would not affect their grades.

The raters involved in the categorization process had professional experience in software engineering for at two years on average. Their experience in software engineering may make the raters curious about the expected outcomes of the categorization process, which may effect the distribution of the categorization process. Furthermore, the resolver also has professional experience in software engineering and IaC script development, which could influence the outcome of the defect category distribution.

External Validity Our findings are subject to external validity, as our findings may not be generalizable for the proprietary domain. The identified reasons and development anti-patterns also may not be comprehensive. Furthermore, our datasets include Puppet scripts, and our derived anti-patterns may not generalize to Ansible or Chef scripts. Also, our sample size of survey respondents is not comprehensive and may not be representative of all IaC practitioners.

Internal Validity We have used a combination of commit messages and issue report descriptions to determine if an IaC script is defective. We acknowledge that these messages might not have given the full context for the raters. Our set of metrics is also not comprehensive.

9 Conclusion

Defects in IaC scripts can cause serious consequences e.g. creating wide-scale outages. Through systematic investigation, we can identify development anti-patterns for IaC scripts

that can be used to advise practitioners on how to improve the quality of IaC scripts. We apply quantitative analysis on 2,138 IaC scripts to determine development anti-patterns. We identify five development anti-patterns, namely, 'boss is not around', 'many cooks spoil', 'minors are spoilers', 'silos', and 'unfocused contribution'. Our findings show defective scripts are related to development activities considered as software engineering best practices, such as the number of developers working on a script. Our identified development anti-patterns suggest the importance of 'as code' activities i.e., application of recommended software development activities for IaC script, as inadequate application of recommended development activities are empirically related to defective IaC scripts. We hope our paper will facilitate further research in the domain of IaC script quality.

Acknowledgements The NSA Science of Security Lablet (award H98230-17-D-0080) at the North Carolina State University supported this research study. We thank the Realsearch research group members for their useful feedback. We also thank the practitioners who answered our questions.

References

Adams B, McIntosh S (2016) Modern release engineering in a nutshell – why researchers should care. In: 2016 IEEE 23Rd international conference on software analysis, evolution, and reengineering (SANER), vol 5, pp 78–90. https://doi.org/10.1109/SANER.2016.108

Alali A, Kagdi H, Maletic JI (2008) What's a typical commit? a characterization of open source software repositories. In: 2008 16th IEEE international conference on program comprehension, pp 182–191. https://doi.org/10.1109/ICPC.2008.24

Anderson S, Allen P, Peckham S, Goodwin N (2008) Asking the right questions: scoping studies in the commissioning of research on the organisation and delivery of health services. Health Res Policy Syst 6(1):7

Ansible (2019) Nasa: Increasing cloud efficiency with ansible and ansible tower. Tech. Rep., Ansible, https://www.ansible.com/hubfs/pdf/Ansible-Case-Study-NASA.pdf?hsLang=en-us

Arisholm E, Briand LC, Johannessen EB (2010) A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. J Syst Software 83(1):2–17. https://doi.org/10.1016/j.jss.2009.06.055. sI: Top Scholars

Arksey H, O'Malley L (2005) Scoping studies: towards a methodological framework. Int J Soc Res Methodol 8(1):19–32. https://doi.org/10.1080/1364557032000119616

Bird C, Nagappan N, Murphy B, Gall H, Devanbu P (2011) Don't touch my code!: examining the effects of ownership on software quality. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th european conference on foundations of software engineering, ACM, New York, NY, USA, ESEC/FSE '11, pp 4–14. https://doi.org/10.1145/2025113.2025119

Breiman L (2001) Random forests. Mach Learn 45(1):5–32. https://doi.org/10.1023/A:1010933404324

Breiman L et al (1984) Classification and Regression Trees, 1st. Chapman & Hall, New York. http://www.crcpress.com/catalog/C4841.htm

Bright J (2017) Slalom's approach to breaking down silos between devops and security teams. https://blog.chef.io/2017/08/16/slaloms-approach-to-breaking-down-silos-between-devops-and-security/. [Online; Accessed 18-Feb-2019]

Brooks FP Jr (1995) The Mythical Man-month (Anniversary Ed.) Addison-Wesley Longman Publishing Co., Inc, Boston

Brown WH, Malveau RC, McCormick HWS, Mowbray TJ (1998) Antipatterns: Refactoring Software, Architectures, and Projects in Crisis, 1st. John Wiley & Sons, Inc., New York

Businge J, Kawuma S, Bainomugisha E, Khomh F, Nabaasa E (2017) Code authorship and fault-proneness of open-source android applications: an empirical study. In: Proceedings of the 13th international conference on predictive models and data analytics in software engineering, ACM, New York, NY, USA, PROMISE, pp 33–42. https://doi.org/10.1145/3127005.3127009

C SN, Menzies T (2019) Assessing developer beliefs: a reply to "perceptions, expectations, and challenges in defect prediction". arXiv:1904.05794

Chen B, Jiang ZMJ (2017) Characterizing and detecting anti-patterns in the logging code. In: Proceedings of the 39th international conference on software engineering, IEEE Press, ICSE '17, pp 71–81. https://doi.org/10.1109/ICSE.2017.15

Cliff N (1993) Dominance statistics: ordinal analyses to answer ordinal questions. Psychol Bull 114(3):494–509

Cohen J (1960) A coefficient of agreement for nominal scales. Educ Psychol Meas 20(1):37–46. https://doi.org/10.1177/001316446002000104

Commons W (2017) Incident documentation/20170118-Labs. https://wikitech.wikimedia.org/wiki/Incident_documentation/20170118-Labs [Online; accessed 27-Jan-2019]

Cramer D, Howitt DL (2004) The Sage dictionary of statistics: a practical resource for students in the social sciences. Sage

Devanbu P, Zimmermann T, Bird C (2016) Belief and evidence in empirical software engineering. In: Proceedings of the 38th international conference on software engineering, ACM, New York, NY, USA, ICSE '16, pp 108–119. https://doi.org/10.1145/2884781.2884812

Easterbrook S, Singer J, Storey MA, Damian D (2008) Selecting empirical methods for software engineering research. Springer London, London, pp 285–311

Elberzhager F, Kremer S, Münch J, Assmann D (2012) Guiding testing activities by predicting defect-prone parts using product and inspection metrics. In: 2012 38th Euromicro conference on software engineering and advanced applications, pp 406–413. https://doi.org/10.1109/SEAA.2012.30

Freedman D (2005) Statistical models : theory and practice. Cambridge University Press, Cambridge

Fu W, Menzies T, Shen X (2016) Tuning for software analytics: is it really necessary? Inf Softw Technol 76:135–146. http://www.sciencedirect.com/science/article/pii/S0950584916300738

Garousi V, Küçük B (2018) Smells in software test code: a survey of knowledge in industry and academia. J Syst Software 138:52–81. http://www.sciencedirect.com/science/article/pii/S0164121217303060

Ghotra B, McIntosh S, Hassan AE (2015) Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proceedings of the 37th international conference on software engineering - volume 1. IEEE Press, Piscataway, pp 789–800. http://dl.acm.org/citation.cfm?id=2818754.2818850

Hall T, Beecham S, Bowes D, Gray D, Counsell S (2012) A systematic literature review on fault prediction performance in software engineering. IEEE T Software Eng 38(6):1276–1304. https://doi.org/10.1109/TSE.2011.103

Hassan AE (2009) Predicting faults using the complexity of code changes. In: Proceedings of the 31st international conference on software engineering, IEEE computer society, Washington, DC, USA, ICSE '09, pp 78–88. https://doi.org/10.1109/ICSE.2009.5070510

Hersher R (2017) Incident documentation/20170118-Labs. https://www.npr.org/sections/thetwo-way/2017/03/03/518322734/amazon-and-the-150-million-typo, [Online; accessed 27-Jan-2019]

Hove SE, Anda B (2005) Experiences from conducting semi-structured interviews in empirical software engineering research. In: 11th IEEE International software metrics symposium (METRICS'05), pp 10 pp.–23 https://doi.org/10.1109/METRICS.2005.24

Huberty CJ, Olejnik S (2006) Applied MANOVA and discriminant analysis, vol 498. John Wiley & Sons, New York

Hudak P (1998) Modular domain specific languages and tools. In: Proceedings. Fifth international conference on software reuse (Cat. No.98TB100203), pp 134–142. https://doi.org/10.1109/ICSR.1998.685738

Humble J, Farley D (2010) Continuous delivery: reliable software releases through build, test, and deployment automation, 1st. Addison-Wesley Professional, Boston

IEEE (2010) Ieee standard classification for software anomalies. IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993) pp 1–23. https://doi.org/10.1109/IEEESTD.2010.5399061

Jiang Y, Adams B (2015) Co-evolution of infrastructure and source code: an empirical study. In: Proceedings of the 12th working conference on mining software repositories, ieee press, Piscataway, NJ, USA, MSR '15, pp 45–55. http://dl.acm.org/citation.cfm?id=2820518.2820527

Kitchenham BA, Pfleeger SL (2008) Personal opinion surveys. Springer London, London, pp 63–92. https://doi.org/10.1007/978-1-84800-044-5_3

Labs P (2018) Puppet documentation. https://docs.puppet.com/, [Online; accessed 08-Aug-2018]

Landis JR, Koch GG (1977) The measurement of observer agreement for categorical data. Biometrics 33(1):159–174. http://www.jstor.org/stable/2529310

Leone M (2016) The economic benefits of puppet enterprise. Tech. rep., ESG. https://puppet.com/resources/analyst-report/the-economic-benefits-puppet-enterprise

MacLeod L, Greiler M, Storey M, Bird C, Czerwonka J (2018) Code reviewing in the trenches: challenges and best practices. IEEE Softw 35(4):34–42. https://doi.org/10.1109/MS.2017.265100500

Mann HB, Whitney DR (1947) On a test of whether one of two random variables is stochastically larger than the other. Ann Math Statist 18(1):50–60. http://www.jstor.org/stable/2236101
Martin RC (2011) The clean coder: a code of conduct for professional programmers. Pearson Education
McCune JT, Jeffrey (2011) Pro Puppet, 1st edn. Apress. https://doi.org/10.1007/978-1-4302-3058-8. https://www.springer.com/gp/book/9781430230571
Meneely A, Williams L (2009) Secure open source collaboration: an empirical study of linus' law. In: Proceedings of the 16th ACM conference on computer and communications security, ACM, New York, NY, USA, CCS '09, pp 453–462. https://doi.org/10.1145/1653662.1653717
Meneely A, Smith B, Williams L (2013) Validating software metrics: a spectrum of philosophies. ACM Trans Softw Eng Methodol 21(4):24:1–24:28. https://doi.org/10.1145/2377656.2377661
Menzies T, Greenwald J, Frank A (2007) Data mining static code attributes to learn defect predictors. IEEE T Software Eng 33(1):2–13. https://doi.org/10.1109/TSE.2007.256941
Morris K (2016) Infrastructure as code: managing servers in the cloud. " O'Reilly Media, Inc."
Munaiah N, Kroh S, Cabrey C, Nagappan M (2017) Curating github for engineered software projects. Empirical Software Engineering pp 1–35. https://doi.org/10.1007/s10664-017-9512-6
Munn Z, Peters MD, Stern C, Tufanaru C, McArthur A, Aromataris E (2018) Systematic review or scoping review? guidance for authors when choosing between a systematic or scoping review approach. BMC Med Res Methodol 18(1):143
Oktaba P (2015) Keep your commits small. https://dzone.com/articles/keep-your-commits-small, [Online; accessed 08-Feb-2019]
Ostrand TJ, Weyuker EJ, Bell RM (2004) Where the bugs are. In: Proceedings of the 2004 ACM SIGSOFT international symposium on software testing and analysis, ACM, New York, NY, USA, ISSTA '04, pp 86–96. https://doi.org/10.1145/1007512.1007524
Pinzger M, Nagappan N, Murphy B (2008) Can developer-module networks predict failures? In: Proceedings of the 16th ACM SIGSOFT international symposium on foundations of software engineering, ACM, New York, NY, USA, SIGSOFT '08/FSE-16, pp 2–12. https://doi.org/10.1145/1453101.1453105
Rahman A, Williams L (2018) Characterizing defective configuration scripts used for continuous deployment. In: 2018 IEEE 11th International conference on software testing, verification and validation (ICST), pp 34–45. https://doi.org/10.1109/ICST.2018.00014
Rahman A, Williams L (2019) Source code properties of defective infrastructure as code scripts. Information and Software Technology. https://doi.org/10.1016/j.infsof.2019.04.013, http://www.sciencedirect.com/science/article/pii/S0950584919300965
Rahman A, Partho A, Morrison P, Williams L (2018) What questions do programmers ask about configuration as code? In: Proceedings of the 4th international workshop on rapid continuous software engineering, ACM, New York, NY, USA, RCoSE '18, pp 16–22. https://doi.org/10.1145/3194760.3194769
Rahman A, Parnin C, Williams L (2019) The seven sins: Security smells in infrastructure as code scripts. In: Proceedings of the 41st international conference on software engineering, IEEE Press, Piscataway, NJ, USA, ICSE '19, pp 164–175. https://doi.org/10.1109/ICSE.2019.00033
Rahman A, Farhana E, Parnin C, Williams L (2020) Gang of eight: a defect taxonomy for infrastructure as code scripts. In: Proceedings of the 42nd international conference on software engineering, ICSE '20, to appear
Rahman F, Devanbu P (2013a) How, and why, process metrics are better. In: Proceedings of the 2013 international conference on software engineering, IEEE Press, Piscataway, NJ, USA, ICSE '13, pp 432–441. http://dl.acm.org/citation.cfm?id=2486788.2486846
Rahman F, Devanbu P (2013b) How, and why, process metrics are better. In: Proceedings of the 2013 international conference on software engineering, IEEE press, Piscataway, NJ, USA, ICSE '13, pp 432–441. http://dl.acm.org/citation.cfm?id=2486788.2486846
Rigby PC, German DM, Storey MA (2008) Open source software peer review practices: a case study of the apache server. In: Proceedings of the 30th international conference on software engineering, ACM, New York, NY, USA, ICSE '08, pp 541–550. https://doi.org/10.1145/1368088.1368162
Romano J, Kromrey J, Coraggio J, Skowronek J (2006) Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys? In: Annual meeting of the florida association of institutional research, pp 1–3
Saldana J (2015) The coding manual for qualitative researchers. Sage
Shambaugh R, Weiss A, Guha A (2016) Rehearsal: a configuration verification tool for puppet. SIGPLAN Not 51(6):416–430. https://doi.org/10.1145/2980983.2908083
Sharma T, Fragkoulis M, Spinellis D (2016) Does your configuration code smell? In: Proceedings of the 13th international conference on mining software repositories, ACM, New York, NY, USA, MSR '16, pp 189–200. https://doi.org/10.1145/2901739.2901761

Shihab E, Jiang ZM, Adams B, Hassan AE, Bowerman R (2011) Prioritizing the creation of unit tests in legacy software systems. Software Pract Exper 41(10):1027–1048. https://doi.org/10.1002/spe.1053

Smith E, Loftin R, Murphy-Hill E, Bird C, Zimmermann T (2013) Improving developer participation rates in surveys. In: 2013 6th International workshop on cooperative and human aspects of software engineering (CHASE), pp 89–92. https://doi.org/10.1109/CHASE.2013.6614738

Sullivan GM, Feinn R (2012) Using effect size-or why the p value is not enough. J Grad Med Educ 4(3):279–282. https://doi.org/10.4300/JGME-D-12-00156.1

Tan PN, Steinbach M, Kumar V (2005) Introduction to data mining, 1st. Addison-Wesley Longman Publishing Co., Inc., Boston

Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2016) Automated parameter optimization of classification techniques for defect prediction models. In: Proceedings of the 38th international conference on software engineering, ACM, New York, NY, USA, ICSE '16, pp 321–332 https://doi.org/10.1145/2884781.2884857

Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2017) An empirical comparison of model validation techniques for defect prediction models. IEEE Trans Softw Eng 43(1):1–18. https://doi.org/10.1109/TSE.2016.2584050

Tosun A, Bener A, Turhan B, Menzies T (2010) Practical considerations in deploying statistical methods for defect prediction: a case study within the turkish telecommunications industry. Inf Softw Technol 52(11):1242–1257. https://doi.org/10.1016/j.infsof.2010.06.006

Tufano M, Bavota G, Poshyvanyk D, Di Penta M, Oliveto R, De Lucia A (2017) An empirical study on developer-related factors characterizing fix-inducing commits. J Softw Evol Proc 29(1):e1797. https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1797

Turhan B, Kocak G, Bener A (2009) Data mining source code for locating software bugs: a case study in telecommunication industry. Expert Syst Appl 36(6):9986–9990. https://doi.org/10.1016/j.eswa.2008.12.028. http://www.sciencedirect.com/science/article/pii/S0957417408009275

Turnbull J (2007) Pulling strings with puppet: automated system administration done right. Apress

van der Bent E, Hage J, Visser J, Gousios G (2018) How good is your puppet? an empirically defined and validated quality model for puppet. In: 2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER), pp 164–174. https://doi.org/10.1109/SANER.2018.8330206

Van Wyk E, Krishnan L, Bodin D, Schwerdfeger A (2007) Attribute grammar-based language extensions for java. In: Proceedings of the 21st european conference on object-oriented programming, springer-verlag, Berlin, Heidelberg, ECOOP'07, pp 575–599. http://dl.acm.org/citation.cfm?id=2394758.2394796

Voelter M (2013) DSL engineering: designing implementing and using domain-specific languages. CreateSpace Independent Publishing Platform, USA

Weinberg GM (1992) Quality software management (vol. 1): systems thinking. Dorset House Publishing Co., Inc., New York

Weiss A, Guha A, Brun Y (2017) Tortoise: Interactive system configuration repair. In: Proceedings of the 32Nd IEEE/ACM international conference on automated software engineering, IEEE press, Piscataway, NJ, USA, ASE 2017, pp 625–636. http://dl.acm.org/citation.cfm?id=3155562.3155641