Replication project

Source code properties of defective infrastructure as code scripts

Nicolas Legros

Montréal, Canada
nicoals.legros@polytml.ca

Thomas Trépanier

Montréal, Canada
thomas.trépanier@polytml.ca

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Problem statement

Continuous delivery or continuous deployement (CD) is the act of releasing new software versions to the end-users as frequently as possible. This practice has been on the rise in the last decade, and with it, the need for tools to automate the deployment process. Infrastructure as code (IaC) is a practice that aims to automate the deployment of infrastructure by using code. IaC scripts are used to describe the desired state of the infrastructure, and the tools will then deploy the infrastructure to match the desired state.

The IaC scripts are usually stored in a version control system (VCS) such as Git. This allows the developers to collaborate on the scripts and to keep track of the changes made to the scripts. The VCS also allows the developers to review the changes made to the scripts before merging them into the main branch, just as they would do with regular code.

However, few other mechanisms exist to ensure the quality of the scripts. This can lead to faulty configuration being deployed to the infrastructure, which is problematic because it can lead to downtime, security breaches, and other issues. As mentionned in the original paper [1], in 2017, Wikimedia Commons executed a defective IaC script which led to the deletion the home directory of around 270 users.

The original paper aimed at introducing a new gating mechanism for IaC scripts by identifying the source code properties of defective scripts. Furthermore, it compares different defect prediction model which aim at identifying defective scripts before they are executed.

This replication project consists of five sections: *I.* Problem Statement which introduced the problem adressed in the original paper, *II.* Research Questions which presents the research questions that we aim to answer in this replication project, *III.* Approach which describes the approach we used to replicate the paper and answer the research questions, *IV.* Results which presents the results the research questions we adressed, and *V.* Conclusion which concludes the replication project.

II. Research questions

This paper aims at answering two of the three research questions in the original paper, *RQ1* and *RQ3*, which are the following:

A. RQ1: What source code properties characterize defective infrastructure as code scripts?

For RQ1, we identified the code properties that characterize defective IaC scripts using *Mann-Whitney U* test and *Cliff's Delta*. We didn't compute the feature importance using a Random Forest since it was only mentionned that we had to do the methods above in the replication guide. Unfortunately, our computers missed computing power to complete the analysis for the Openstack dataset.

B. RQ3: How can we construct defect prediction models for infrastructure as code scripts using the identified source code properties?

For RQ3, we replicated *3.5.1 Principle component analysis*, *3.5.2 Statistical learners* and *3.5.3 Evaluation methods*. It's good to know that we didn't replicate *3.5.4 Comparison model construction* since we only did the analysis by properties and not by bag-of-words.

These questions will be answered using the approach described in Section III.

III. Approach

A. Repository mining

To answer the research questions, we used the same dataset as the original paper. However, to get a better idea of the actual state of IaC scripts in the reported organizations, we also mined the latest version of the Mirantis, OpenStack and Wikimedia repositories.

To mine the Mirantis and Wikimedia repositories, which are located on Github, we used the Github REST API. We first started by fetching all the repositories of the organization using the following GET request:

```
api/search/repositories?<org_name>
```

Using the list of repository names, we could then fetch the commit history of the every repository which gave us a list of the commits with their creation date using the following GET request:

```
api/<repo_name>/commits
```

This allowed us to filter out the repositories which did not respect **Criteria-3** of the paper, which stated that the repository must still be active. This status was determined to be that a repository must have at least two commits per month.
However, the preceding request did not return the files impacted by the commit, which we needed to validate **Criteria-2** of the paper. Therefore, using the filtered repositories, we fetched the list of files impacted by each commit using the following GET request:

```
api/repos/{owner}/{repos}/commits
```

Using the list of files impacted by each commit, we could validate **Criteria-2** of the paper, which was that at least 11% of the files belonging to the repository must be IaC scripts. It is important to mentionned that we interpreted this criteria as meaning that 11% of every file that have ever been impacted by a commit must be IaC scripts. In that regard, we verified if 11% of the files in every commit of a repository was an IaC script, and if so, we added the repository to the list of valid repositories.
Lastly, we respected **Criteria-1** by only downloading public repositories of the Mirantis and Wikimedia organizations.

To mine the OpenStack repository, which is located on Gerrit, we used the Gitea REST API. We first started by fetching all the repositories of the organization using the following GET request:

```
api/v1/repos/search
```

Using the list of repository names, we could then fetch the commit history of the every repository which gave us a list of the commits with their commit message, creation date, and the files they impacted using the following GET request:

```
api/v1/repos/{owner}/{repos}/commits
```

Finally, using the commit history for each project and the files impacted by each commit, we could filter out repositories that did not respect **Criteria-2** and **Criteria-3** of the paper. The repositories that respect all three critierias will be referred to as the *valid repositories*.

B. Issue mining & Extended Commit Message

Following the mining of the repositories, we mined the issues of the valid Mirantis, OpenStack and Wikimedia repositories to build the Extended Commit Message (XCM) of each commit. The XCM consists of the commit message and the summary of the issue to which the commit relates. To build this XCM, we needed to extract the relevant issue from the commit message.

We first analysed the issue board and commit messages of some mined commits and determined that the issues were refered using their code on the issue tracker.Therefore, we extracted the issue number using the following RegExp pattern on the commit message:

```
RegExp = #\d{1,10}
```

This pattern matches any part of the commit message which consists of a # followed by a number between 1 and 10 digits, which corresponds to an issue number. Using this issue number, we were able to use the Github REST API and Gitea API to fetch the issue description and complete the XCM of the commit.

C. RQ1: What source code properties characterize defective infrastructure as code scripts?

For this research question we used the reported data from the paper[1]. We used the *Mann-Whitney U* test with the Scikit Learn package to evaluate which properties had the biggest influence on defective files. The null hypothesis is that the property is not different between defective and neutral files, and the alternative hypothesis is that the property is larger for defective than neutral files. As in the paper, we consider a significance level of 95% which means we reject the null hypothesis when $p - value < 0.05$.

We also used *Cliff's Delta* by calculating it with Neilernst's package[2] to measure how large the difference between the distribution of each characteristics for defective and neutral files is.

D. RQ3: How can we construct defect prediction models for infrastructure as code scripts using the identified source code properties?

Before using statistical learners, we completed a PCA analysis to determine what properties should be used. We only used the principle components that accounted for at least 95% of the total variance as the input for the statistical learners. We can see in Table I that only one or two principle components account for 95% of the total variance depending on the dataset.

With the component created, we than used it as the input for the different statistical learners. Like the paper, we used Scikit Learn packages to construct the models. The learners that were used are Classification Tree (CART), K Nearest Neighbor (KNN), Logistic Regression (LR), Naive Bayer (NB) and Random Forest (RF).

To evaluate the performance of the different classification models, we used the same metrics as the paper (i.e. precision, recall, AUC, F-measure).

[1]https://figshare.com/s/ad26e370c833e8aa9712
[2]https://github.com/neilernst/cliffsDelta

TABLE I
NUMBER OF PRINCIPLE COMPONENTS FOR THE MODELS

Dataset	Number of components
Mirantis	1
Mozilla	1
Openstack	2
Wikimedia	2

IV. RESULTS AND DISCUSSION

A. RQ1: What source code properties characterize defective infrastructure as code scripts?

After completing the *Mann-Whitney U* test and *Cliff's Delta* test we can identify which properties have a $p-value < 0.05$ for all datasets (i.e. Attribute, Command, Ensure, File, File mode, Hard coded string, Include, Lines of code, Require and SSH Key). After analysis, we can see that we did not obtain exactly the same $p-values$ as the original paper, but we did get the same *Cliff's Delta* values. For example, in our case, the *Comment* property has a $p-value = 0.58$ for Mozilla whereas the original paper has a $p-value = 0.23$. Furthermore, we see that we obtain $p-values \approx 0.01$ for the *URL* property, where the original paper obtains $p-value < 0.001$ for the Mirantis and Wikimedia organizations. We present these results in Table II for the Mirantis dataset, in Table III for the Mozilla dataset and in Table IV for the Wikimedia dataset.

TABLE II
VALIDATION OF IDENTIFIED SOURCE CODE PROPERTIES FOR MIRANTIS

Property	$p-value$	$delta$
Attribute	< 0.001	0.47
Command	0.005	0.24
Comment	< 0.001	0.37
Ensure	< 0.001	0.38
File	< 0.001	0.36
File mode	< 0.001	0.41
Hard coded string	< 0.001	0.55
Include	< 0.001	0.33
Lines of code	< 0.001	0.45
Require	< 0.001	0.36
SSH KEY	< 0.001	0.39
URL	0.009	0.22

TABLE III
VALIDATION OF IDENTIFIED SOURCE CODE PROPERTIES FOR MOZILLA

Property	$p-value$	$delta$
Attribute	< 0.001	0.40
Command	< 0.001	0.18
Comment	0.58	0.03
Ensure	< 0.001	0.09
File	< 0.001	0.18
File mode	< 0.001	0.24
Hard coded string	< 0.001	0.40
Include	< 0.001	0.31
Lines of code	< 0.001	0.50
Require	< 0.001	0.19
SSH KEY	< 0.001	0.24
URL	0.081	0.08

TABLE IV
VALIDATION OF IDENTIFIED SOURCE CODE PROPERTIES FOR WIKIMEDIA

Property	$p-value$	$delta$
Attribute	< 0.001	0.47
Command	0.008	0.18
Comment	< 0.001	0.22
Ensure	< 0.001	0.29
File	< 0.001	0.31
File mode	< 0.001	0.24
Hard coded string	< 0.001	0.55
Include	< 0.001	0.37
Lines of code	< 0.001	0.51
Require	< 0.001	0.32
SSH KEY	< 0.001	0.24
URL	0.011	0.17

B. RQ3: How can we construct defect prediction models for infrastructure as code scripts using the identified source code properties?

As mentionned in the previous section, we only used the principle components that accounted for at least 95% of the total variance. We can see in Table I that only one or two principle components account for 95% of the total variance depending on the dataset. The number of principle components for each dataset corresponds to the ones in the paper.

Since the paper doesn't specify the different parameters for the models, it would be difficult to obtain exactly the same results. Nevertheless, we obtained results that are very similar that bring to the same conclusions. The results from the cross-validation for each model can be found in Table V for the Mirantis dataset, in Table VI for the Mozilla dataset, in Table VII for the Openstack dataset and in Table VIII for the Wikimedia dataset. We haven't included the results from the actual paper, since it's publicly available.

TABLE V
CROSS-VALIDATION RESULTS FOR MIRANTIS

	RF	NB	LR	KNN	CART
AUC	0.701661	0.714252	0.750981	0.693334	0.659597
Recall	0.707425	0.407360	0.649691	0.672546	0.707425
Precision	0.701199	0.846909	0.798322	0.667389	0.701199
F1-measure	0.695896	0.541781	0.708236	0.663964	0.698448

TABLE VI
CROSS-VALIDATION RESULTS FOR MOZILLA

	RF	NB	LR	KNN	CART
AUC	0.731664	0.699599	0.756323	0.713161	0.691230
Recall	0.649017	0.392519	0.565923	0.619417	0.627106
Precision	0.642651	0.831862	0.706600	0.604170	0.642550
F1-measure	0.645764	0.532261	0.626990	0.608864	0.633393

C. Repository Mining & Issue Mining

The result of our replication of the repository mining and XCM building is available in the repository under the

TABLE VII
CROSS-VALIDATION RESULTS FOR OPENSTACK

	RF	NB	LR	KNN	CART
AUC	0.647741	0.694343	0.659972	0.659195	0.574832
Recall	0.667616	0.368902	0.731321	0.687022	0.660176
Precision	0.653112	0.847009	0.643218	0.661449	0.655685
F1-measure	0.660440	0.512676	0.682243	0.673287	0.657360

TABLE VIII
CROSS-VALIDATION RESULTS FOR WIKIMEDIA

	RF	NB	LR	KNN	CART
AUC	0.664721	0.709438	0.736270	0.699140	0.583164
Recall	0.591171	0.366128	0.586200	0.627349	0.587493
Precision	0.664007	0.885945	0.774041	0.732415	0.666620
F1-measure	0.628276	0.515651	0.663515	0.673132	0.623400

`data/processed/<org>` folder. The `data` folder contains the following files:

- `valid-repos.json` which the list of valid repositories for the organization
- `xcms folder` which contains the XCM for the organization

We would like to note that we were unable to fetch the full commit history for some OpenStack and Wikimedia repositories, as we did not have the processing power to fetch the lengthy commmit history for these repositories. We opted to only fetch the commits of the last two years if the full commit history was longer than 2000 commits.
For example, the OpenStack repository contains 252 000 commits, so we only fetch the commits from the past two years.

V. CONCLUSION

5) conclusion

REFERENCES

[1] Rahman, Akond & Williams, Laurie. (2019). "Source Code Properties of Defective Infrastructure as Code Scripts". Information and Software Technology. 112. 10.1016/j.infsof.2019.04.013.