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, in 2017, Wikimedia Commons executed a defective IaC script which led to the deletion the home directory of around 270 users.

The replicated 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.

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?

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

These questions will be answered using the approach described in section *III*

III. Approach

3) Methodologies for answering each RQ (How to mine the repositories, the tools employed, and ML models they've used).

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:

```
https://api.github.com/search/repositories?<or
```

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:

```
https://api.github.com/repos/<repo_name>/commi
```

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:

```
https://api.github.com/repos/{owner}/{repo}/co
```

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 Gerrit REST API. We first started by fetching all the repositories of the organization using the following GET request:

B. 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*[2] to measure how large the difference between the distribution of each characteristics for defective and neutral files is.

C. 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 principal component 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

IV. Results and discussion

4) Results and discussion, including a comparison between the results of the replication study and the original study and a list of the limitations encountered during the replication.

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

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

TABLE I
Number of principle components for the models

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

TABLE II
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 III
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

TABLE IV
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

V. Conclusion

5) conclusion

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[4] K. Elissa, "Title of paper if known," unpublished.

TABLE V
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

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.