Anomaly Detection in DevOps Toolchain

Antonio Capizzi
Università degli Studi di Messina
8 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Salvatore Distefano
Università degli Studi di Messina
220 PUBLICATIONS **2,220** CITATIONS
SEE PROFILE

Luiz Jonatã Pires de Araújo
Innopolis University
32 PUBLICATIONS **65** CITATIONS
SEE PROFILE

Manuel Mazzara
Innopolis University
280 PUBLICATIONS **2,151** CITATIONS
SEE PROFILE

Anomaly Detection in DevOps Toolchain

Antonio Capizzi[1], Salvatore Distefano[1], Luiz J.P. Araújo[2]
Manuel Mazzara[2], Muhammad Ahmad[1,3], Evgeny Bobrov[2]

[1] University of Messina, Italy
[2] Innopolis University, Innopolis, Respublika Tatarstan, Russian Federation
[3] Department of Computer Engineering, Khwaja Fareed University of Engineering and Information Technology, Pakistan.

Abstract. The tools employed in the DevOps Toolchain generates a large quantity of data that is typically ignored or inspected only on particular occasions, at most. However, the analysis of such data could enable the extraction of useful information about the status and evolution of the project. For example, metrics like the "lines of code added since the last release" or "failures detected in the staging environment" are good indicators for predicting potential risks in the incoming release. In order to prevent problems appearing in later stages of production, an anomaly detection system can operate in the staging environment to compare the current incoming release with previous ones according to predefined metrics. The analysis is conducted before going into production to identify anomalies which should be addressed by human operators that address false-positive and negatives that can appear. In this paper, we describe a prototypical implementation of the aforementioned idea in the form of a "proof of concept". The current study effectively demonstrates the feasibility of the approach for a set of implemented functionalities.

1 Introduction

Evolution of software engineering spans over more than fifty years where different problems have been presented, and solutions explored [1]. From *"structured programming"* to *"life cycle models"* and *"software development methodologies*, researchers and developers have better understood the software development process and its complexity. Meanwhile, a fast-speed growing technological progress has transformed the usage of computers from devices for numerical and scientific computation into everyday ubiquitous devices. This progress has not stopped, and an increasing number of companies are moving to Agile methodologies, also including in the software development process feedback from operational stages in a DevOps [2, 3] fashion.

Continuous delivery (CD) is an important concept part of the DevOps philosophy and practice as it enables organizations to deliver new features quickly as well as to create a repeatable and reliable process incrementally improving to bring software from concept to customer. The goal of CD is to enable a constant flow of changes into the production via an automated software production line - the *continuous delivery pipeline.*

The CD pipeline has a variable complexity and can be constituted by several phases supported by different tools. However, the core idea is always the same: when a developer integrates or fixes a functionality into the software, a set of software tools automatically builds the application, starts the automatic tests and, finally, delivers the new feature.

CD is made possible via automation to eliminate several manual routines during software production, testing, and delivery. CD pipeline automation involves in the toolchain different tools, each generating messages, data and logs. However, the amount of recorded data can prevent its manual inspection when one searches for a specific issue or traces back abnormal behavior. Inside a DevOps toolchain, data is generated and stored in different formats. The analysis of such data is a daunting task even for an experienced professional as well as its processing, recognition, mining and, consequently, addressing of critical aspects.

In this paper, we discuss how to automatically analyze the data generated during a DevOps toolchain integrated to anomaly detection (AD) methods for identifying potentially harmful software releases. As a result, software releases that can lead to potential malfunctioning during the normal system life could be identified. The implemented approach could still lead to false-positives and false-negatives since no approach can overcome this theoretical limitation [4]; however, developers are provided with an instrument to validate and maintain the code. This investigation focuses on an ongoing project structured according to the DevOps philosophy, and we will apply analytical techniques to gain insights for professionals involved in the software development process.

In Section 2, background is provided, with specific regard to DevOps toolchains and AD techniques and tools. In Section 3, we presented an approach for integrating AD into a project structured with DevOps. After that, section 4 describes the case study, in details: the SpaceViewer application, the corresponding DevOps process and toolchain and the developed AD module, the SpaceViewer AD system - SVADS. Section 5 then reports on the experiments and obtained results, also compared against those obtained by offline tools on the full SpaceViewer dataset, demonstrating the effectiveness of the proposed approach. Section 6 summarises the key aspects of the proposed approach and future work.

2 Background

This section introduce some technical background for this work project and its implementation. This research is bringing together two different communities with research literature and vocabulary sporadically overlapping. Thus, we first discuss the details of DevOps toolchains, and then we report on data science techniques adopted in the software development process.

2.1 The DevOps toolchain

DevOps [2] consists of a set of practices to promote collaboration between the developers, IT professionals (in particular sysadmin, i.e. who works on IT operations) and quality assurance personnel. DevOps is implemented via a set of software tools [5] that enable the management of an environment in which software can be built, tested and released quickly, frequently, and a more reliable manner. In addition to CD, *continuous integration* (CI) stands as a key concept in DevOps approaches. A typical example of CI consists of continuously integrating changes made by developers into a repository, then a project build is automatically executed, if the build works well automatic tests are started. IF also automatic tests passes, the change is integrated into the code through CD and published in production environment.

One of the main objectives of DevOps is to mitigate problems in production, which is done by reducing the gap between development and testing environments with the production environment. Collaborations between "Dev" and "Ops" aiming to reduce this gap make use of a complex toolchain including, at least, some version control tool (e.g. Git), CI/CD automation tools (e.g. Jenkins), package managers (e.g. NPM) and test tools (e.g. JUnit). Other additional tools used in DevOps are configuration management tools (e.g. Ansible), monitoring tools (e.g. Nagios), security tools (e.g. SonarCube), team collaboration tools (e.g. Jira) and database management tools (e.g. Flyway). DevOps infrastructures are typically either fully implemented on cloud platforms. It is a good practice in DevOps to build the entire infrastructure using containers; therefore, tools for containerization (e.g. Docker) are employed, sometimes coupled by tools for containers orchestration (e.g. Kubernetes).

An outcome from the complex pipeline involved in a DevOps project is the generation of a large amount of data, in particular, log files and metrics generated in each stage. Examples of activities that generate considerable data on the project cycle include changes made by developers; the application building and its corresponding entries on the compilation and dependencies of the project; the execution of automatic tests; and software usage by end-users after release into the production.

A large amount of the data generated in a DevOps toolchain requires some form of automation and possibly dimensionality reduction and feature selection [6]. However, collecting, storing, and analysing such a high dimensional data could enable insights into how to improve the DevOps pipeline [7]. For example, historical data can be analyzed to estimate a probabilistic measure of the success of a new release.

2.2 Anomaly detection in software development

The application of data science techniques to software development processes has become increasingly popular in the last decades, in part due to the availability of a growing amount of data generated during the development process. Methods like data preprocessing and machine learning have been used for tasks including estimating programming effort, predicting risks to the project and identifying defects in the produced artefacts [8].

In recent years, the term "AIOps" has been coined to refer to a set of techniques which employ machine learning and artificial intelligence to enable the analysis of data from IT operation tools [9]. As a result, there has been a noticeable improvement in service delivery, IT efficiency and superior user experience [10]. Applications of AIOps to DevOps processes, mainly to analyze data produced by the toolchain, specifically in operation, have been proposed in literature [11]. In another example, AIops has been used to support software development processes within an organization during the migration from waterfall processes to Agile/DevOps [12].

AD has been an increasingly popular approach for identifying observations that deviate from the expected pattern in the data. In data science, AD refers to a set of techniques used for identifying observations which occur with low frequency in the dataset, i.e. entries that do not conform to the expected distribution or pattern. Such data entries raise suspicion and represent potential risk depending on the context in which data has been collected. Examples of applications of AD in different problem domains include detection of bank frauds [13], structural defects in building construction [14], system health monitoring and errors in a text [15]. It is trustworthy mentioning that there has been limited literature demonstrating the application of AD methods in the context of DevOps. An example of AD applied to DevOps operations in a Cloud platform was reported in [16].

3 Integrating anomaly detection into DevOps

As mentioned previously, the vast amount of data generated by the DevOps toolchain enables the use of AD techniques to reduce the probability of software errors released in production. An AD system can compare the multivariate features of the prospective release with the collected data from previous versions. The DevOps study analyzed in this work is following the development, staging, and production model. In this model, the activities are sorted in three deployment environments, detailed as follows:

- **Development**: environment in which the developers work and can quickly test new features.
- **Staging**: testing environment to experiment and test the new features that have to be merged to the system.
- **Production**: environment in which the software is released and utilised by end-users.

The development and staging environments offer an opportunity for assessing the correctness of the prospective release. Moreover, the data collected during these stages enable the application of data science techniques such as AD for preventing software errors. The most suitable approach depends on the characteristics of the data. For example, if a considerable amount of labeled data is available, supervised learning techniques (e.g. support vector machine) can lead to satisfactory predictive accuracy. In case there is no information whether each observation in the training dataset is an anomaly, an unsupervised learning technique is the most suitable approach.

This study employs the local outlier factor (LOF) algorithm, which is an unsupervised AD technique which computes the local density deviation of a multivariate data point compared to its neighbors. This method enables the identification and plotting of anomalies in the data and supports better decision-making [17]. The LOF algorithm is used before a new version of the software is moved from the staging phase to the release phase. In other words, it identifies whether the prospective release significantly deviates from exiting distributions in the following set of metrics: the number of pushes, builds and errors, lines of code that have been changed and the number of failed tests. Fig. 1 shows the operational flow, which consists of three macro-phases distinguishing development, AD and recovery activities.

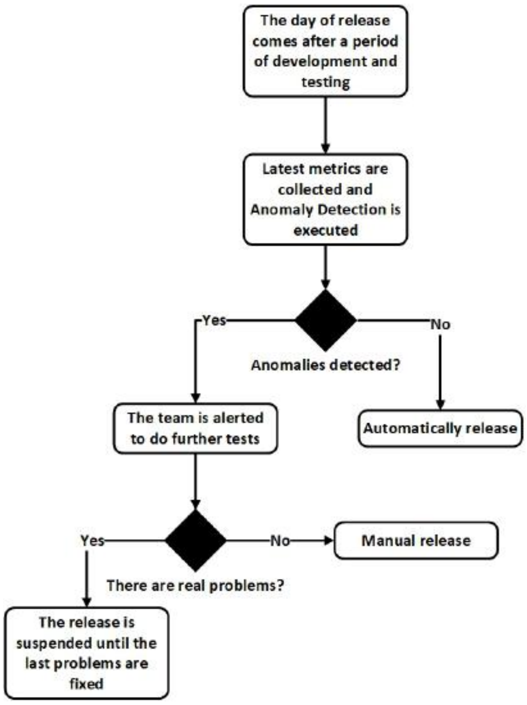
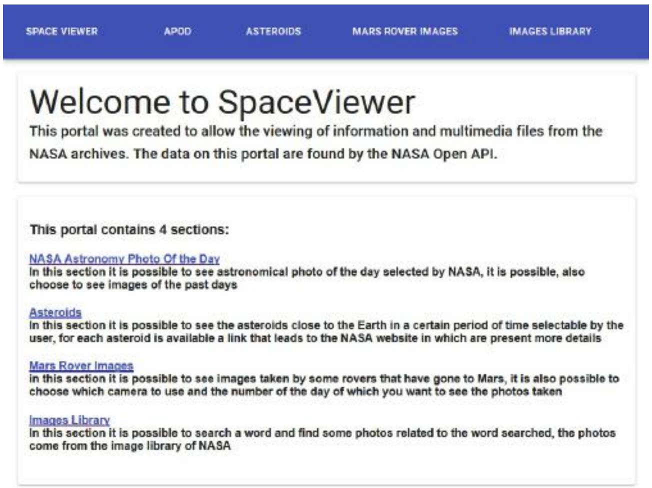

The day of release comes after a period of development and testing
Latest metrics are collected and Anomaly Detection is executed
Yes
No
Anomalies detected?
The team is alerted to do further tests
Automatically release
Yes
No
Manual release
There are real problems?
The release is suspended until the last problems are fixed

Fig. 1. The anomaly detection task in the proposed DevOps workflow.

In the development stage, software development and testing are implemented in the development and staging environments as described previously. These activities are performed between the current release and the next version. The activities in this stage are mostly executed by the development team. In the detection stage, AD using the LOF algorithm is employed and possibly coupled with advanced computational techniques like artificial intelligence and machine learning. Moreover, the comparison of distinct AD methods can provide more a well-informed decision in the recovery phase, when a human actor assesses the identified anomalies.

4 A case study: SpaceViewer

This section describes a proof-of-concept application developed by exploiting a DevOps approach and toolchain proposed in this work. It consists of a Web application developed by adopting a DevOps process: *SpaceViewer* [18]. SpaceViewer is a ReactJS [19] project enabling queries for interacting and interfacing the NASA space archive exploiting their Open APIs [20]. A client-server app has been implemented where the server-side small back-end interface [21] (developed in Python 3.7 [22] using Flask 1.0.2 [23]) sends a token to the client app necessary to query the NASA DB. Fig. 2 reports the SpaceViewer homepage with the main features implemented.



Welcome to SpaceViewer

This portal was created to allow the viewing of information and multimedia files from the NASA archives. The data on this portal are found by the NASA Open API.

This portal contains 4 sections:

NASA Astronomy Photo Of the Day
In this section it is possible to see astronomical photo of the day selected by NASA, it is possible, also choose to see images of the past days

Asteroids
In this section it is possible to see the asteroids close to the Earth in a certain period of time selectable by the user, for each asteroid is available a link that leads to the NASA website in which are present more details

Mars Rover Images
in this section it is possible to see images taken by some rovers that have gone to Mars, it is also possible to choose which camera to use and the number of the day of which you want to see the photos taken

Images Library
In this section it is possible to search a word and find some photos related to the word searched, the photos come from the image library of NASA

Fig. 2. SpaceViewer ReactJS web-application

4.1 DevOps toolchain

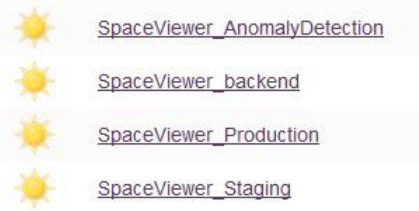
The DevOps toolchain adopted in the SpaceViewer app development is composed of the following tools

- **Jenkins** [24]: CI/CD and automation
- **GitHub** [25]: version control
- **CodeClimate** [26]: assessment of the quality of the source code
- **Docker** [27]: deployment tool
- **Slack** [28]: team collaboration and management of automatic alerts from Jenkins Jobs
- **Node Package Manager - NPM** [29]: run build, deploy, and automatic test of the ReactJS application

- **SpaceViewer Anomaly Detection System - SVADS** [30]: this tool was created specifically for this experimentation, it will be described in the section 4.2

As discussed in Section 3, the deployment environments have been implemented as follows:

- **Development environment**: local in developer machines.
- **Staging environment**: remote server deployed in a Docker container and triggered by Jenkins. Whenever a new version of the software is pushed on the GitHub repository, the staging environment is automatically rebuilt.
- **Production environment**: remote server in a Docker container triggered by Jenkins. Before a build in production, SVADS is triggered.



SpaceViewer_AnomalyDetection
SpaceViewer_backend
SpaceViewer_Production
SpaceViewer_Staging

Fig. 3. SpaceViewer Jenkins Jobs (Pipelines).

The Jenkins tool has been set up to manage such deployment environments. Fig. 3 depicts the Jenkins Jobs created for the SpaceViewer case study, thus establishing a Jenkins pipelines [31]. Jenkins jobs are mainly instantiated for deploying in staging (SpaceViewer_Staging) and production (SpaceViewer_Production), while additional jobs are created to run the back-end process (SpaceViewer_Backend) and perform AD before launching the production job (SpaceViewer_AnomalyDetection).

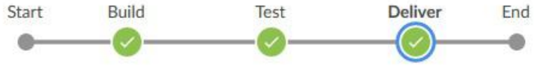


Fig. 4. Staging/Production Pipelines stages

The pipelines for both the Staging and the Production deployments consist of the stages shown in Fig. 4. An automatic system in Jenkins triggering the rebuild in Staging at every Development push on the GitHub repository has been deployed. As stated above, before deploying in Production, the AD job has to be performed to detect any possible anomaly or issue in the DevOps development process. Then, if no anomalies are detected, the Production job is automatically triggered and the SpaceViewer software version is released in Production. In the SpaceViewer DevOps pipeline, Jenkins is also connected through a specific plugin [32] to the messaging software Slack [28]. This way, the team can receive real-time automatic alerts regarding Jenkins jobs outcomes (e.g. failure and success).

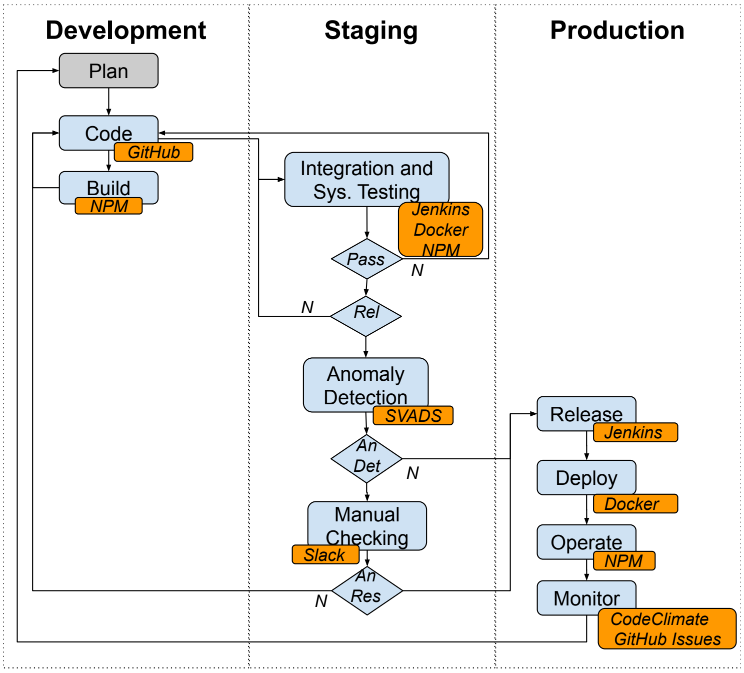


Fig. 5. SpaceViewer DevOps process and toolchain.

The overall SpaceViewer DevOps process and toolchain are shown in Fig. 5, highlighting the different stages of the process and the main tools involved. The swim-lanes identify the three environments taken into account, correlating their activities with the different stages of the process. As stated above, the latter two environments are deployed into two independent containers, while the Development one runs locally into the development/developer machines. The only step that is not directly involved in the SpaceViewer automated DevOps process is the initial Plan one. After planning, coding activities (Code) trigger the DevOps pipeline with specific metrics from the development environment and tools (ReactJS and GitHub), as discussed in the following section. Once implemented, SpaceViewer modules are ready for unit testing and building loop (Build exploiting the NPM tool) and, after that, they are automatically released to the Staging Environment for Integration and System Testing by the Jenkins SpaceViewer_Staging job, triggered by GitHub pushes into the repository. This stage loops until related activities, mainly testing ones, are performed and successfully passed, then triggering the release if ready for that, always orchestrated by the SpaceViewer_Staging job (see Fig. 3). If so, the AD job (SpaceViewer_AnomalyDetection) is launched and run the SVADS tool. In the case of anomaly the control is demanded to the people involved in project for further Manual Checking, automatically informing the team about the anomaly through a Slack chat, the procedure of release is suspended and in production remains the latest version of application. On the other hand, if there are no anomalies, the SpaceViewer_Production job (see Fig. 3) is triggered by SVADS, the production environment is rebuilt and the latest features are integrated (Release, Deploy, Operate, Monitor) through the corresponding tools in the pipeline.

4.2 Space Viewer Anomaly Detection System

The tool for AD - Space Viewer Anomaly Detection System, SVADS in short - has been developed in Python and, in the SpaceViewer case study [30], consists of a script launched by the Jenkins before the delivery in Production of a new version of the software. SVADS retrieves data relating to the last development period (i.e. since the day after the last release, to the day the new release is being executed), generated by the DevOps toolchain and collected by the system meanwhile, to perform AD. The SVADS algorithm is mainly tasked at detecting outliers in the SpaceViewer software release to Production, to avoid potential issues for the software in Production. It implements the Local Outlier Factor (LOF) algorithm [33] by exploiting the *scikit-learn* Python Library [34]. After executing the SVADS algorithm, the system fills the FLAG attribute indicating the presence/absence of an anomaly, and stores latest data in the dataset for future release AD.
Specifically, such a dataset is comprised of performance metrics collected via Rest APIs provided by the DevOps toolchain shown in Fig. 5. The parameters taken into account by the SVADS dataset are reported below and, as discussed above, are related to the modifications done exclusively in the last DevOps cycle:

- Number of lines of code ($NLoC$) added, modified or deleted divided by the number of commits ($NCom$) from GitHub in the Code stage - $P1 = NLoC/NComm$
- Number of builds that failed when executing the Jenkins pipeline to deploy in staging from the Integration and System Testing phase - $P2$
- Number of automatic tests that failed when executing the Jenkins pipeline to deploy in staging from the Integration and System Testing phase - $P3$
- Number of deliveries that failed when executing the Jenkins pipeline to deploy in staging from the Integration and System Testing phase - $P4$
- Number of issues reported by CodeClimate from the Code and Monitor phases - $P5$

- Number of issues reported in GitHub from Operation and Monitor phases - $P6$

Each entry in the dataset corresponds to a software release and the parameters $P1 - P6$ are the number of occurrences of related events since the last release. They are therefore reset by any new release. The values of such attributes are normalized according to the number of working days elapsed since the last release to mitigate the effects of longer periods of maintenance. It also reflects the good practice of performing regular "small" commits in contrast to doing few but substantial commits. The following attributes capturing meta-data of each entry are also added to the dataset:

- A unique identifier - ID
- The date of the release, i.e. when the parameter values are collected and written into the dataset - $DATE$

Some of the above DevOps toolchain metrics are often used to also support better decision-making regarding potential risks in a software release. For example, a high number of failed builds, automated tests and deliveries in Staging might be an indicator that a specific release requires additional management effort. It is trustworthy mentioning that such a dataset can also enable the observation of complex patterns involving different parameters related to the occurrence of software defects, errors or faults.

It is important to point out that the SVADS tool was created for this case study, but it can be used for any project that has a DevOps Toolchain like the one used in this study.

5 Experiments, results and discussion

The experimentation of the proposed approach for the DevOps toolchain in the SpaceViewer case study started in early July 2019 and took approximately one month. In this experimentation, data entries conforming the format defined in Section 4.2 were added to the SpaceViewer dataset at the moment of every software release in production by the SVADS tool. Table 1 reports the full dataset describing 25 subsequent releases between 4th of July and 8th of August, uniquely identified by the attribute ID.

Firstly, an initial dataset was generated to attend the requirement of a considerable quantity of observations to perform an unsupervised AD method. In this study, data concerning software releases were collected for ten days without being processed by the SVADS module. After this initial period, the AD system was then activated, thus starting operating on the SpaceViewer DevOps process, as shown in Fig. 5. For each new release, the LOF algorithm was trained with the dataset comprising previous releases and the current candidate release. Finally, the data describing the last release is appended to the dataset and available for future use. Fig. 6 illustrates the output from the LOF model after the 25th release, i.e. the outlier scores for each observation.

Fig. 6 enables the observation of several insights into the integration of AD into DevOps. First, SVADS supports the identification of data

$P1$	$P2$	$P3$	$P4$	$P5$	$P6$	ID	$DATE$
22.57	0.04	0.06	0.08	0	0	1	7/4/2019
59	2	3	5	0	1	2	7/5/2019
87	1	4	6	0	1	3	7/6/2019
13	1	3	6	0	0	4	7/7/2019
130	3	4	5	1	0	5	7/8/2019
135	3	6	8	3	0	6	7/9/2019
27	2	4	7	6	0	7	7/10/2019
10	2	4	6	4	0	8	7/11/2019
40	0	1	3	6	0	9	7/12/2019
21	3	5	6	6	0	10	7/13/2019
33	3	5	6	6	0	11	7/14/2019
65	6	8	10	8	0	12	7/15/2019
90	3	4	6	8	0	13	7/16/2019
114	6	7	10	13	0	14	7/17/2019
255	5	9	9	12	0	15	7/18/2019
44	3	4	5	13	0	16	7/19/2019
123	4	6	8	17	0	17	7/22/2019
171	5	7	8	23	0	18	7/24/2019
100	3	4	5	23	0	19	7/25/2019
42	1	5	6	23	0	20	7/26/2019
94	1	3	4	8	0	21	7/29/2019
243	29	30	31	13	0	22	7/30/2019
28	5	6	8	15	0	23	7/31/2019
244	45	48	50	0	0	24	8/1/2019
35	6	7	8	0	0	25	8/8/2019

Table 1. The SpaceViewer dataset.

entries, i.e. software releases, that clearly fails to conform expected patterns in data. For example, IDs 15, 22 and 24 have higher outlier scores and easily distinguished from their peers. Second, SVADS requires some degree of human interference for labelling data with edging feature values. For example, the release with ID in Fig. 6 is closer to most of the releases than to the clearly identified anomalies. In larger projects in the real-world, SVADS would flag such releases as requiring further assessment by the project manager. Finally, the collection and analysis of such data enable the observation of patterns between features such as lines of codes, stages of development and occurrences of anomalies. In the implemented case study, for example, anomaly releases have been mostly identified by higher code volumes or Staging failures.

An interesting matter that deserves further consideration is whether an unsupervised AD (outlier detection) method should be employed instead of supervised AD (novelty detection). For the first case, at the moment of a new release, the AD model is trained with the entire dataset and outlier scores above a specified threshold indicate anomalies. In the second method, it is assumed that there is the availability of a significant

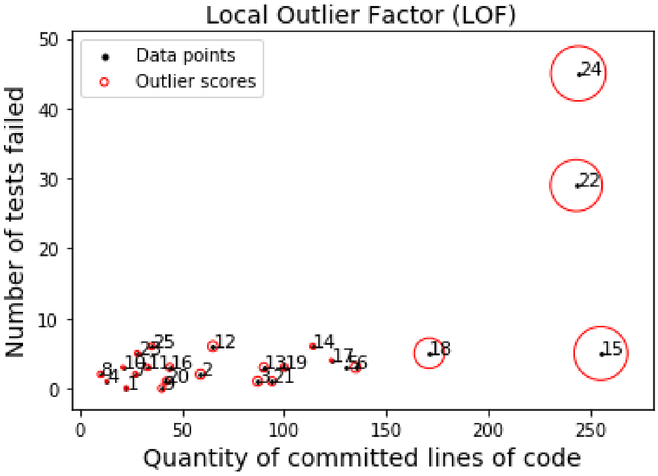


Fig. 6. Outlier scores for the dataset using LOF for anomaly detection on the full SpaceViewer dataset.

number of software releases. Moreover, it is also necessary that each release has been labeled by a specialist (e.g. the project manager) whether it is an anomaly. Hence, the latter method can be noticed as closer to a policy-based approach for AD.
The implemented method was validated against other offline statistical and machine learning techniques. Several statistical methods can be utilised for identifying outliers, including the popular k-nearest neighbors and LOF. Moreover, some AD models outperform others depending on the characteristics of the data and the problem domain. Fig. 6 illustrates four different AD models trained using the generated dataset.
These outcomes from the models in Fig. 7 reinforce the usefulness of the proposed SVADS approach. In fact, an ensemble of AD models enables a more precise and undisputed decision regarding software releases that are likely to result in an error in the production environment. Finally, some AD models can provide decision boundaries for classifying anomalies which enable one to gain insights regarding which features that are more likely yo result in a risk to the ongoing project.

6 Conclusions

DevOps is becoming an increasingly adopted approach in software development, gaining attention from both industry and academia as per the rising number of projects, conferences, and training programs in this field [3, 35]. A DevOps toolchain typically generates a large amount of data that enables the extraction of information regarding the status and progress of the addressing project. In this paper, we described a prototypical implementation of a system for detecting anomalies in software release adopting DevOps development process.

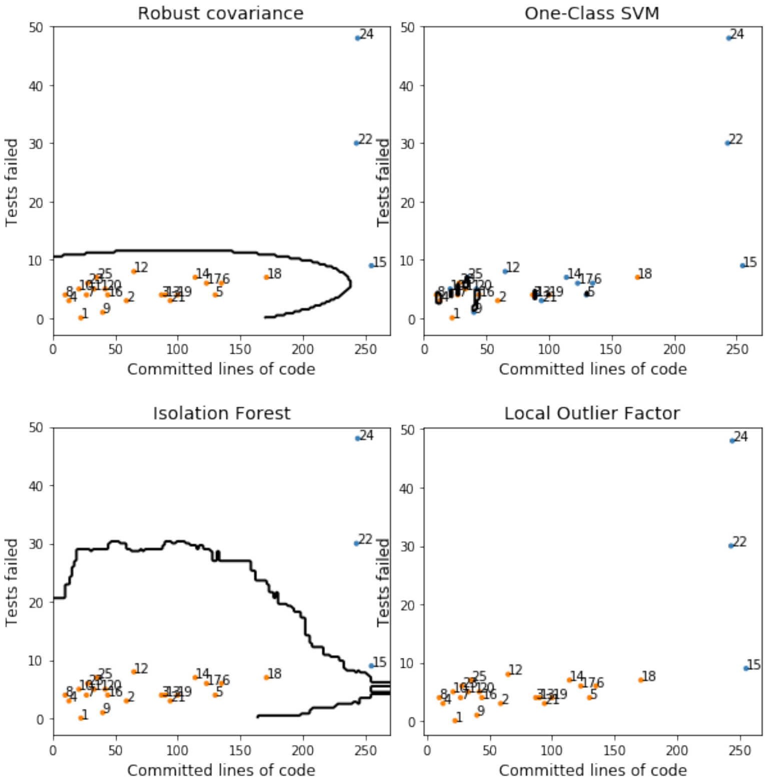


Fig. 7. Comparing different AD methods and decision boundaries on the SpaceViewer Dataset.

Despite the small number of functionalities implemented in our SpaceViewer case study, this paper demonstrates the feasibility of the proposed workflow. Obtained results and their comparison against powerful solution integrating several AD models proves the validity of the proposed approach and its effectiveness as a tool for supporting decision-making and precise identification of potentially harmful candidate releases in the production. Furthermore, a dataset on AD for software release in the DevOps toolchain has been generated and made publicly available for the community.

Future work will approach the stabilization of the current implementation and broader experimentation in real-world production environments and an more extensive number of features, which has been scarcely reported in the literature. Moreover, future research will approach a broader discussion on how to consider the fluctuation of feature values can indicate anomalies through the project life-cycle.

References

1. A. I. Wasserman, "Modern software development methodologies and their environments," *Computer Physics Communications*, vol. 38, no. 2, pp. 119 – 134, 1985.
2. L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective.* Addison-Wesley Professional, 1st ed., 2015.
3. J. Bruel, M. Mazzara, and B. Meyer, eds., *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment - First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers*, vol. 11350 of *Lecture Notes in Computer Science*, Springer, 2019.
4. J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to automata theory, languages, and computation, 3rd Edition.* Pearson international edition, Addison-Wesley, 2007.
5. M. Kersten, "A cambrian explosion of devops tools," *IEEE Software*, vol. 35, pp. 14–17, mar 2018.
6. S. Protasov, A. M. Khan, K. Sozykin, and M. Ahmad, "Using deep features for video scene detection and annotation," *Signal, Image and Video Processing*, vol. 12, pp. 991–999, Jul 2018.
7. K. Kontogiannis, C. Brealey, A. Giammaria, B. Countryman, M. Grigoriou, M. Jimenez, M. Fokaefs, F. Kassam, and F. Bordeleau, "2nd workshop on devops and software analytics for continuous engineering and improvement," in *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, CASCON '18, (Riverton, NJ, USA), pp. 369–370, IBM Corp., 2018.
8. B. Akinsanya, L. J. Araujo, M. Charikova, S. Gimaeva, A. Grichshenko, A. Khan, M. Mazzara, O. Okonicha, and S. Daniil, "Machine learning and value generation in software development: a survey," in *Software Testing, Machine Learning and Complex Process Analysis (TMPA-2019)*, pp. 1–10, Springer International Publishing, (in press, forthcoming).
9. Z. Li and Y. Dang, "Aiops: Challenges and experiences in azure," (Santa Clara, CA), USENIX Association, May 2019.
10. Y. Yang, D. Falessi, T. Menzies, and J. Hihn, "Actionable analytics for software engineering," *IEEE Software*, vol. 35, pp. 51–53, January 2018.
11. J. Hoffman, "How AIOps Supports a DevOps World." `https://thenewstack.io/how-aiops-supports-a-devops-world/`.
12. B. Snyder and B. Curtis, "Using analytics to guide improvement during an agiledevops transformation," *IEEE Software*, vol. 35, pp. 78–83, January 2018.
13. C. Guo, H. Wang, H. Dai, S. Cheng, and T. Wang, "Fraud risk monitoring system for e-banking transactions," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pp. 100–105, Aug 2018.

14. P. Chen, S. Yang, and J. A. McCann, "Distributed real-time anomaly detection in networked industrial sensing systems," *IEEE Transactions on Industrial Electronics*, vol. 62, pp. 3832–3842, June 2015.
15. V. Chandola, A. Banerjee, and V. Kumar, *Anomaly Detection*, pp. 1–15. Boston, MA: Springer US, 2016.
16. D. Sun, M. Fu, L. Zhu, G. Li, and Q. Lu, "Non-intrusive anomaly detection with streaming performance metrics and logs for devops in public clouds: A case study in aws," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, pp. 278–289, April 2016.
17. V. J. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, pp. 85–126, Oct 2004.
18. A. Capizzi, "SpaceViewer - a ReactJS portal for NASA Open API consultation." `https://github.com/antoniocapizzi95/SpaceViewer/`.
19. Facebook, "ReactJS - A JavaScript library for building user interfaces." `https://reactjs.org/`.
20. NASA, "NASA Open API." `https://api.nasa.gov/`.
21. A. Capizzi, "SpaceViewer - little back end." `https://github.com/antoniocapizzi95/SpaceViewer_BE/`.
22. P. S. Foundation, "Python - Programming Language." `https://www.python.org/`.
23. T. P. Projects, "Flask is a lightweight WSGI web application framework.." `https://palletsprojects.com/p/flask/`.
24. K. Kawaguchi, "Jenkins - an open source automation server which enables developers around the world to reliably build, test, and deploy their software." `https://jenkins.io/`.
25. S. C. P. J. H. Tom Preston-Werner, Chris Wanstrath, "GitHub - The world's leading software development platform." `https://github.com/`.
26. CodeClimate, "CodeClimate Quality." `https://codeclimate.com/quality/`.
27. I. Docker, "Docker - Build, Share, and Run Any App, Anywhere." `https://www.docker.com/`.
28. S. Technologies, "Slack is where work flows. It's where the people you need, the information you share, and the tools you use come together to get things done." `https://slack.com/`.
29. K. M. Isaac Z. Schlueter, Rebecca Turner, "Node Package Manager." `https://www.npmjs.com/`.
30. A. Capizzi, "Anomaly Detection System used for SpaceViewer DevOps Toolchain." `https://github.com/antoniocapizzi95/SpaceViewer_ADS/`.
31. K. Kawaguchi, "Jenkins Pipeline Documentation." `https://jenkins.io/doc/book/pipeline/`.
32. T. Jacomb, "Slack Notification Plugin for Jenkins." `https://plugins.jenkins.io/slack`.
33. scikit learn, "Novelty detection with Local Outlier Factor (LOF)." `https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_novelty_detection.html/`.
34. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and

E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

35. M. Mazzara, A. Naumchev, L. Safina, A. Sillitti, and K. Urysov, "Teaching devops in corporate environments - an experience report," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment - First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers*, pp. 100–111, 2018.