Project funded by the EU Horizon 2020 programme under the Marie

Skldowska-Curie grant agreement No 871342

uDevoOps

Software Quality Assurance for Microservice Development Operations Engineering

Deliverable D4.2. Workflow development

November 2023

Abstract

This is the accompanying document of Deliverable D4.2 of the μDevOps project, entitled “Workflow development”, with reference to the development of risk different assessment techniques. The type of the deliverable is marked as *Other*, and is made up of software artifacts, along with this accompanying document. The implemented artifacts support the integrated risk assessment and mitigation strategies described in the deliverable D4.1. The artifacts are made available on the project website www.udevops.eu, as well as on the following GitHub repository:

https://github.com/uDEVOPS2020/Risk_Assessment_Mitigation.git

and at the linked Zenodo repository:

https://doi.org/10.5281/zenodo.10422810, indexed by OpenAIRE

1 INTRODUCTION

This document describes the artifacts implemented for risk assessment and mitigation techniques under development for the Microservice quality assessment, work package 4.

Before presenting the single artifacts' repositories, here we first briefly describe how one could combine estimates obtained with different methods among the implemented ones. The strategies presented in D4.1 (”Business risk assessment techniques ”) adopt different techniques able to provide an estimate of probability of failure (PFD). In almost all the cases, this is derived by a frequentist strategy, where the number of observed failures is used by an estimator depending on how the sample was selected. In one case, we have also explored the use of a Bayesian strategy, with the advantage of having a simple way to update the estimates as more data becomes available.

The choice of the method to adopt depend on many aspects, including the information available (e.g., auxiliary variable for sampling), the uncertainty on the operational profile, how often the profile changes, the proneness of the system to fail (e.g., with few failures, a Bayesian approach may be better in avoiding overestimation), etc.

Regardless the estimators chosen, a simple yet effective policy to combine the estimates is by combining them. Every estimate is associated with a variance of the estimator, and this variance is directly related to the confidence we can put in the estimation (higher variance, less stable estimate, little confidence). Therefore, given a quality attribute (for instance Reliability $R = 1 - PFD$), a simple way to combine two estimates R_1 and R_2 obtained with two distinct methods is to take the weighted average $\hat{R}_1 \cdot w_1 + \hat{R}_2 \cdot w_2$, where the weights w_1 and w_2 are the variances of the estimate $w_1 = V(\hat{R}_1), w_2 = V(\hat{R}_2)$.

The result from these techniques can be combined with the failure impact qualitative scale reported in D4.1 to give the risk estimate. Moreover, the risk mitigation techniques defined in the last part of D4.1 serve to reduce the failure probability, hence the risk, with reference to several quality attributes of interest, primarily reliability, performance and energy.

The focus of this document is to accompany the artifacts implemented and made available for all such techniques. Each of the following sections reports the instructions for using the technique and reproduce the results reported in the published papers and recalled in Deliverable D4.1. The artifacts will be extended/integrated in the rest of the project, as the Consortium will advance with WP5.

All the artifacts are available on the project website `www.udevops.eu`, as well as on the following GitHub repository:

https://github.com/uDEVOPS2020/Risk_Assessment_Mitigation.git

and at the linked Zenodo repository:

https://doi.org/10.5281/zenodo.10422810, indexed by OpenAIRE

2 RISK ASSESSMENT

2.1 RELIABILITY-RELATED ASSESSMENT

2.1.1 DNN ASSESSMENT AND IMPROVEMENT CYCLE (DAIC)

This artifact is the replication package related to Guerriero et al. (2023b) discussed in section 3.2.1 of Deliverable 4.1. In this repository, the code to repeat the experiments is available. We reported the datasets to completely reproduce the experiments on MNIST.

The file "results.csv" contains all the raw results discussed in the paper.

Requirements To run the experiments it is required to install Python 3 and all the requirements reported in the requirement.txt file. DNN-OS requires the installation of KNIME (https://www.knime.com/downloads).

Execution To execute all the experiments run the command "sh run_complete.sh". All the pre-trained models are available. The script will stop the execution at each cycle to allow the execution of DNN-OS on KNIME. To execute DNN-OS it is required to set the path to the desired training, validation, and test sets automatically generated by the Python script. The operational accuracy estimate can be read as the output of the last "column rename" block.

The experiments with SelfChecher can be executed by running the code available in the replication package available at https://github.com/self-checker/SelfChecker.

2.1.2 Image Classification Oracle Surrogate (ICOS)

This artifact corresponds to the implementation of ICOS according to Guerriero et al. (2023a) as described in section 3.2.1 of Deliverable 4.1. In this repository, the code to repeat the experiments on MNIST, CIFAR10, and CIFAR100 is available.

The Python code in MNISTCNNs.py, CIFAR10CNNs.py, and CIFAR100CNNs.py can be executed to generate the .csv files containing the training, validation, and test sets ready to be submitted to ICOS. All the requirements needed to execute the Python code are listed in the requirements.txt file.

The implementation of ICOS provided needs an installation of KNIME (v4.0.0). After the import of the .knar file into the KNIME workspace, the .csv file generated can be inserted in input in the "CSV Reader" blocks according to their tags. With the “Double Configuration” blocks the minimum confidence and support can be set. The “CSV Writer” blocks can be configured with the path where the CSV containing the output of the testing session can be saved. The “Java Snippet” block, named “IDI”, inside the “ICOS” metanode can be used to switch the partitioning criterion by uncommenting the corresponding code. The “Partitioning” blocks, “Node 15” and “Node 77” (inside the “CRO” metanode), can be used to change the test set size for ICOS and CRO respectively.

2.2 PERFORMANCE- AND ENERGY-RELATED ASSESSMENT

2.2.1 Computer Vision

This repository contains the replication package of the paper by Hampau et al. (2022) discussed in section 3.3.2 of Deliverable 4.1.

The full dataset including raw data, data analysis Python scripts, and automatizing scripts produced during the study are available.

Running the experiment:

- On local machine run: python3 main.py
- One can customise the number or type of treatments by editing the rows.csv file

Post-requisites:

- For new raw data, manually copy all logs to raw_logs folder, following the directory tree format from the replication package

2.2.2 Android Apps

This artifact implements the replication package and provides the dataset of the paper by Horn et al. (2023) for MOBILESoft 2023 discussed in section 3.3.2 of Deliverable 4.1.

Data analysis From the data generated by the experiment or provided in the archive 'raw_results.tar.gz' the dataset is compiled using the script 'experiment/utils/run_to_csv.py'.

Replicating the experiment The original readme of Android Runner can be found in "./ANDROID-RUNNER.md". Follow the setup instructions for Android Runner before you continue. The experiment itself and specific instructions can be found under "./experiment/README.md". To perform the replication, three main steps are required: 1. Obtain the versions of the APKs of the apps listed "./experiment/apks/summary.txt" (due to legal reasons, we cannot provide them directly) 2. Install the prerequisites relating to Android Runner, app subjects and R 3. Run the experiment using Android Runner 4. Compile the dataset from the raw results and perform the data analysis

Overview of the experiment setup Figure 2.1 shows the components of the experiment setup.

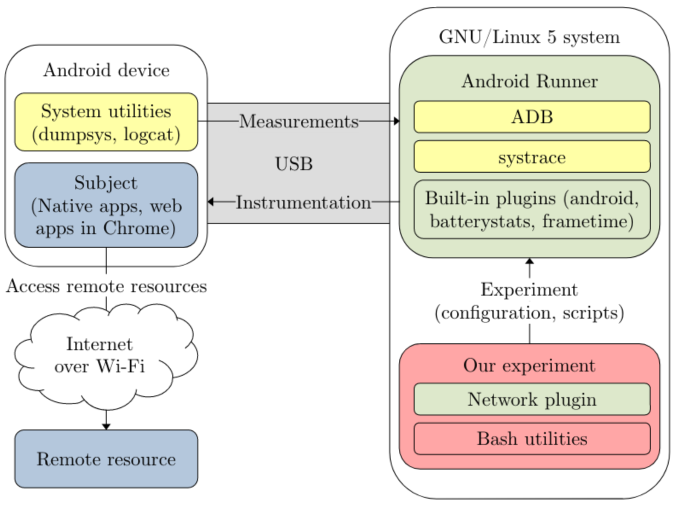

Figure 2.1. Experiment setup

2.2.3 IoT

This artifact contains the replication package and dataset of the paper published at EASE 2023 (Research track) by Wagner et al. (2023) presented in section 3.3.2 of Deliverable 4.1.

This study has been designed, developed, and reported by the following investigators:

- Linus Wagner
- Maximilian Mayer

- Andrea Marino
- Alireza Soldani Nezhad
- Hugo Zwaan
- Ivano Malavolta

For any information, interested researchers can contact us by sending an email to any of the investigators listed above. A full replication package, including the software setup for conducting our experiments, benchmarks, an Ansible and Docker setup package, the final dataset generated, and scripts to analyse and visualize the resulting data are described below.

The 'experiment-runner @ 1613d3d' is based on *Robot Runner* introduced by Swanborn and Malavolta (2021) in "Robot Runner: A Tool for Automatically Executing Experiments on Robotics Software" (https://doi.org/10.1109/ICSE-Companion52605.2021.00029), and was forked from https://github.com/S2-group/experiment-runner.

Each of the folders listed above is described in detail in the remainder of this readme. Crucially, the USAGE.md describes how to use this replication package in detail.

Analysis This folder contains the data that has been generated through our experiments, scripts to process, analyse, and visualize this data, and the final visualization results.

2.2. PERFORMANCE- AND ENERGY-RELATED ASSESSMENT

The data in the TSV file has been generated from several results and different experimental runs. The data was merged by hand, however, this is the only manual interaction within this process.

Applications This folder contains the benchmarks used in our experiments. Four different computational problems, implemented in four different programming languages. All benchmarks are taken from "The Computer Language Benchmarks Game" (`https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html`) and, if needed, adapter for our purposes. They do not use multi-threading, as WASM does not support that out-of-the-box.

Environment This folder contains an Ansible setup used to automatically deploy the experiment setup on multiple network devices at once, and a Docker setup to compile our benchmarks to WASM executables.

The following tools are installed and used through this environment setup:

- Python (`https://www.python.org/`)
- Docker (`https://www.docker.com/`)
- Time (`https://man7.org/linux/man-pages/man1/time.1.html`)

- PowerJoular (`https://github.com/joular/powerjoular`)
- experiment-runner (`https://github.com/marinoandrea/experiment-runner`)
- Wasmer (`https://wasmer.io/`)
- Wasmtime (`https://wasmtime.dev/`)

Experiment Runner Our experiment setup uses the *experiment-runner*(`https://github.com/S2-group/experiment-runner`) to execute experiments in a controlled environment. This experiment runner comes with its own documentation, which can be found in the corresponding repository. However, we modified it to fit our needs.

2.2.4 Monitoring tools

This artifact represents the replication package for the paper by Dinga et al. (2023) discussed in section 3.3.2 of Deliverable 4.1.

This repo contains the raw data and analysis scripts related to all the activities carried out for the experimentation. It also contains the TrainTicket benchmark system with the integration of four monitoring tools.

This study has been designed, developed, and reported by the following investigators:

2.2. *PERFORMANCE- AND ENERGY-RELATED ASSESSMENT*

- Madalina Dinga (VU Amsterdam)
- Ivano Malavolta (VU Amsterdam)
- Luca Giamattei (University of Naples Federico II)
- Antonio Guerriero (University of Naples Federico II)
- Roberto Pietrantuono (University of Naples Federico II)

For any information, interested researchers can contact us by sending an email to any of the investigators listed above.

Instructions for replicating the experiment In the following, the required steps for replicating the experiments are reported:

1. **Experiment execution** Experiment-runner is used for automating the execution of the experiments. *Infrastructure setup*. The experiment uses the version of TTS available in the [TrainTicket](./TrainTicket) folder. It includes the integration of the TTS with a selection of four monitoring tools (ELK stack, Netdata, Prometheus and Zipkin) and the set of a set of 34 load test scripts generated with [K6](https://k6.io/). Please check [this readme](./TrainTicket/readme.md) file for the detailed instructions about how to setup and deploy the various versions of the TTS used in this experiment.

The pipeline can be triggered using Experiment-Runner with the

‘RunnerConfig-monitoring.py‘ specification. Please follow the instructions in the project README.

Required software:

- **K6** is required to be installed on the machine for running the K6 load test scripts.
- Python3, required by Experiment-Runner
- Experiment-Runner (https://github.com/S2-group/experiment-runner) - Note that the framework is not supported on Windows.

2. **Data analysis** Data processing and data analysis scripts for data obtained during the experiment performed on the Train Ticket system (https://github.com/FudanSELab/train-ticket) are available in the *data* folder.

3 RISK MITIGATION

3.1 LACUNA-EVALUATION

This repository contains all the material needed to replicate an experiment on Lacuna V2 (https://github.com/S2-group/Lacuna) evaluation. This tool allows the removal of JavaScript dead code with 4 different optimization levels. The goal of the experiment is to evaluate the impact of this smell on various run-time metrics, measured in mobile web applications on Android smartphones.

The collected metrics and the tools used to measure them, are the following:

- *Energy consumption*
  - Energy (J) $\rightarrow$ Trepn plugin
- *Performance*
  - CPU usage (%) $\rightarrow$ Trepn (https://github.com/S2-group/

Lacuna-evaluation/tree/main/android-runner/AndroidRunner/Plugins/trepn) plugin

- Memory usage (MB) → Trepn plugin
- Loading time (ms) → JS snippet in each web page
- First Paint (ms) → JS snippet in each web page using the perfumeJS (https://zizzamia.github.io/perfume/) library
- First Contentful Paint (ms) → JS snippet in each web page using the perfumeJS (https://zizzamia.github.io/perfume/) library

- *Network usage*
  - Number of packets → mitmproxy (https://mitmproxy.org/)
  - Bytes transferred (KB) → mitmproxy (https://mitmproxy.org/)

Content All the data and tools required for the replication of the experiment are provided in the following folders:

- LacunaV2-master (https://github.com/S2-group/Lacuna-evaluation/tree/main/LacunaV2-master) - Tool used to remove JavaScript dead code from the subjects

- android-runner (https://github.com/S2-group/Lacuna-evaluation/tree/main/android-runner) - Tool adopted to automate the execution of the experiment on the Android device
- data (https://github.com/S2-group/Lacuna-evaluation/tree/main/data) - Raw and aggregated data obtained from the execution of the experiment, including also the raw data about the count of the number of functions in each subject
- data_analysis (https://github.com/S2-group/Lacuna-evaluation/tree/main/data_analysis) - Script adopted for data processing and analysis
- scripts (https://github.com/S2-group/Lacuna-evaluation/tree/main/scripts) - Aggregation scripts adopted on raw data and scripts used on the subjects
- subjects (https://github.com/S2-group/Lacuna-evaluation/tree/main/subjects) - Web Applications executed on the mobile device during the experiment. For each subject there are 4 different versions, one for every optimization level in Lacuna V2.
- TodoMVC (https://github.com/S2-group/Lacuna-evaluation/tree/main/subjects/TodoMVC) - 20 web apps from the [TodoMVC

project](https://todomvc.com/)

- lacunaWebPages (https://github.com/S2-group/Lacuna-evaluation/tree/main/subjects/lacunaWebPages) - 16 popular web pages which are part of the [Tranco](https://tranco-list.eu/) list. This folder contains also the set of all 150 potentially-usable subjects from the Tranco list

Required software

- Http-server (https://www.npmjs.com/package/http-server) (‘npm install http-server‘)
- Pluginbase (https://pypi.org/project/pluginbase/) (‘pip install pluginbase‘)
- BeautifulSoup (https://pypi.org/project/beautifulsoup4/) (‘pip install beautifulsoup4‘)

For [AndroidRunner](https://github.com/S2-group/android-runner):

- Python 3 (https://www.python.org/downloads/)
- Android Debug Bridge (https://developer.android.com/studio/command-line/adb) (‘sudo apt install android-tools-adb‘)
- monkeyrunner (https://developer.android.com/studio/test/monkeyrunner) (‘sudo apt install monkeyrunner‘)

- JDK 8 (https://openjdk.java.net/install/) (‘sudo apt-get install openjdk-8-jre‘)
- lxml (https://lxml.de/installation.html) (‘sudo apt install python-lxml‘)

For Mitmproxy, check https://docs.mitmproxy.org/stable/overview-installation/.

Setup The following commands have already been executed on the subjects in this repository.

Create the additional variants for each subject Apply JavaScript dead code removal to the angularjs_require subject, using the analyzers tajs and dynamic and optimization level 2

node LacunaV2-master/lacuna ./TodoMVC/lvl0/angularjs_require -a tajs dynamic -o 2 -d ./TodoMVC/lvl2/angularjs_require -f

Add JavaScript snippet to each subject This command is executed on all variants of the subjects to add a JavaScript snippet that allows the measurement of loading time, fp, and fcp metrics using the perfumeJS library. The file AddJS.py is in the scripts (https://github.com/S2-group/Lacuna-evaluation/tree/main/scripts) folder.

python3 AddJS.py

Run the experiment To host the subjects on the machine, execute the following command in the directory where you have ‘subjects‘:

Hosting the current directory on port 2020

http-server -p 2020

To be able to access the localhost websites on the mobile device:

Enter the portnumber that the websites are hosted on

adb reverse tcp:2020 tcp:2020

To collect details of the packets transferred for each run of each subject, execute the command below and add a proxy to the network that the mobile device is connected to. Make sure that the port used for the proxy on the mobile device is the same as the one used in the command below.

Run mitmproxy on port 5050 and save the flow file in a text file such as flowFileName

mitmproxy -p 5050 –set save$_{s}tream_{f}ile = flowFileName$

To execute AndroidRunner a configuration file is needed. Make sure that the path to monkeyrunner in the config file used, points to the respective location on your machine.

The config file we used is in `/android-runner/examples/trepn/config_webfinal.json`

```
python3 android-runner path/to/config
```

Aggregate results To be able to analyze the results of the metrics collected, we need to aggregate the results for each variant of each metric into a separate csv file.

The command below aggregates the results for the loading time, fp and fcp metrics placed within the ‘results_perfumeJS‘ directory.

Aggregate loading time, fp, fcp

```
python3 aggregate_perfumeJS.py path/to/results_perfumeJS
```

To aggregate the mitmproxy results, first we need to convert the flow text file(s) generated into csv file(s). If the experiment produced multiple text flow files, execute the following command for each file.

Convert the text flow file into a csv file

```
python3 logfileToCSV.py flowFileName flowFileName.csv
```

Now, to aggregate the mitmproxy results, make sure that the perfumeJS metrics are aggregated first. The reason for this is that aggregating mitmproxy results uses the starting time of each run, which is aggregated in the directory ‘aggregated_perfumeJS‘.

The command below aggregates the mitmproxy results that are in a folder with only the csv flow files, such as ‘results_mitmproxy/‘, given the directory ‘aggregated_perfumeJS‘ with the aggregated starting time.

Aggregate packets transferred and bytes transferred

```
python3 aggregate_mitmproxy.py path/to/results_mitmproxy path/to/aggregated_perfumeJS
```

3.2 MICRO2VEC

This repository accompanies the submission of the manuscript: Micro2vec: Anomaly Detection in Microservices Systems by Mining Numeric Representations of Computer Logs

co-authored by

Marcello Cinque*, Raffaele Della Corte*, and Antonio Pecchia** *Università degli Studi di Napoli Federico II **Università degli Studi del Sannio

to JOURNAL OF NETWORK AND COMPUTER APPLICATIONS.

CONTENT:

This repo contains the following folders/sub-folders:

- Training
- Test
  - AUTH
  - DEL

3.2. MICRO2VEC

- DOS
- KILL
- NORMATIVE
- REG

Each folder/sub-folder contains the following set of 13 logs (*):

- astaire.txt
- bono.txt
- cassandra.txt
- chronos.txt
- ellis.txt
- homer.txt
- homestead.txt
- homesteadaccess.txt
- homesteadprov.txt
- ralf.txt
- ralfaccess.txt

- sprout.txt
- sproutaccess.txt

NOTE:

1) ’Training’ logs are used to tune the detection approach presented in the paper.

2) ’Test’ logs are collected within normative/anomalous events. They are used to test the proposed approach, and they are collected by means of controlled experiments.

3) Training-Test logs are collected with independent runs of the system in hand.

REGULAR EXPRESSIONS This repository also provides the regex.txt file, which contains regular expressions used in our study as well as for replication purposes.

(*) Some files are missing due to GitHub space limitations. Contact us in case. CONTACT:

For any further information please contact the Authors: macinque@unina.it, raffaele.dellacorte2@unina.it, antonio.pecchia@unisannio.it

REFERENCES

Dinga, M., Malavolta, I., Giamattei, L., Guerriero, A., and Pietrantuono, R. (2023). "An empirical evaluation of the energy and performance overhead of monitoring tools on docker-based systems." *Service-Oriented Computing*, F. Monti, S. Rinderle-Ma, A. Ruiz Cortés, Z. Zheng, and M. Mecella, eds., Cham, Springer Nature Switzerland, 181–196.

Guerriero, A., Lyu, M. R., Pietrantuono, R., and Russo, S. (2023a). "Assessing operational accuracy of cnn-based image classifiers using an oracle surrogate." *Intelligent Systems with Applications*, 17, 200172.

Guerriero, A., Pietrantuono, R., and Russo, S. (2023b). "Iterative assessment and improvement of dnn operational accuracy." *2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, Los Alamitos, CA, USA, IEEE Computer Society, 43–48, <https://doi.ieeecomputersociety.org/10.1109/ICSE-NIER58687.2023.00014> (may).

Hampau, R., Kaptein, M., Emden, R., Rost, T., and Malavolta, I. (2022). "An empirical study on the performance and energy consumption of ai containerization strategies for computer-vision tasks on the edge." 50–59 (06).

Horn, R., Lahnaoui, A., Reinoso, E., Peng, S., Isakov, V., Islam, T., and Malavolta, I. (2023). "Native vs web apps: Comparing the energy consumption and performance of android apps and their web counterparts." *2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 44–54.

Swanborn, S. and Malavolta, I. (2021). "Robot runner: A tool for automatically executing experiments on robotics software." *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 33–36.

Wagner, L., Mayer, M., Marino, A., Soldani Nezhad, A., Zwaan, H., and Malavolta, I. (2023). "On the energy consumption and performance of webassembly binaries across programming languages and runtimes in iot." *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, EASE '23, New York, NY, USA, Association for Computing Machinery, 72–82, <https://doi.org/10.1145/3593434.3593454>.