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 D5.2. Proof-of-concept μDevOps platform

May 2025

Abstract

This is the accompanying document of Deliverable D5.2 of the μDevOps project, entitled "Proof-of-concept μDevOps platform", with reference to the development of the testing techniques for quality improvement and assessment designed and implemented during the project. 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 testing and assessment with respect to several quality attributes of interest, ranging from functional testing to reliability, performance, robustness and energy consumption, described in Deliverable 5.1. The artifacts are made available on the project website `www.udevops.eu`, as well as on the following GitHub uDevOps Repository:

https://github.com/uDEVOPS2020/Integrated-Quality-Assessment-and-Improvement-Framework/tree/main

and at the linked Zenodo repository:

https://doi.org/10.5281/zenodo.15563864, indexed by OpenAIRE.

CONTENTS

1 INTRODUCTION

This document describes the artifacts implemented for the μDevOps proof-of-concept. The testing process that the proof-of-concept supports is described in D5.1. It foresees the implementation of a set of "services" for testing, categorized as *testing for assessment* and *testing for improvement*, and further distinguished as means to support *fault avoidance*, *fault prediction*, *fault removal*, and *fault tolerance* – which are all the dimensions of the macro-attribute known as *Dependability*.

This deliverable consolidates and builds upon outcomes from previous work packages (WP2, WP3, and WP4), including new techniques developed in WP5, each contributing specific techniques and tools for context-aware, in vivo, and risk-based testing. The result is a framework that seamlessly integrates diverse techniques exploiting a variety of gathered data – from code repositories and system logs to runtime performance metrics - into a coherent process for both quality assessment and quality improvement, and for both the *Dev* and *Ops* stage.

The techniques target a wide array of quality attributes, including but not limited to reliability, performance, energy consumption, and robustness.

Deliverable 5.2 contains all the artifacts (code and documentation) developed for a total of 14 techniques or assessment studies (the "services" described in D5.1). Therefore, this accompanying document does not describe the techniques, but focuses on the structure of the repository to help navigate the artifacts.

It is important to note that, as a proof of concept, *not all techniques are at the same maturity level*. Specifically, we distinguish, from top to bottom maturity level:

- **Level 1**. Fully implemented tools with user-friendly instructions, requiring minimal effort to apply to any system under test.
- **Level 2**. Collections of scripts executable with provided prototypes; moderate effort needed to adapt to other systems.
- **Level 3**. Artifacts demonstrating usage through project-specific experiments; significant effort required to generalize.

Finally, most of the services are experimented on open and widely-used benchmarking system, such as `Train Ticket`, `Sock Shop`, Android apps, ML benchmarks such as MNIST, CIFAR. Others, especially energ-related services, are also experimented on the industrial applications provided by the partners.

2 PROOF-OF-CONCEPT OF THE INTEGRATED QUALITY ASSESSMENT AND IMPROVEMENT FRAMEWORK

2.1 OVERVIEW

The uDevOps Repository located at:

https://github.com/uDEVOPS2020/Integrated-Quality-Assessment-and-Improvement-Framework/tree/main

serves as a proof of concept for the project, whos aim was to share and enhance knowledge about software quality in microservice systems through a unified view. The PoC integrates diverse testing techniques, facilitating both quality assessment and improvement. It leverages data from code repositories, system logs, and runtime performance metrics to support Fault Avoidance, Fault Prediction, Fault Removal, Fault Tolerance Targeted quality attributes include Reliability, Robustness, Performance, Energy Consumption.

All developed artifacts, including code and documentation, are housed within this repository and its sub-repositories.

2.2 REPOSITORY STRUCTURE

The main repository encompasses several sub-repositories, each dedicated to specific testing techniques. Each sub-repository contains its own `README.md` detailing specific functionalities, setup instructions, and usage guidelines.

Here, we list the services with the associated sub-repository. These are categorized by what they support:

1. **Dev or Ops** stage;
2. **Improvement or Assessment**;
3. **Fault Avoidance, Prediction, Removal or Tolerance**;
4. **Quality Attribute**

Figure 2.1 depicts all the techniques, described in the following:

- **Functional and Robustness Testing**
  Description: Technique for defects detection to be applied at unit, integration or system testing stage before release.
  Stage: Dev

2.2. REPOSITORY STRUCTURE

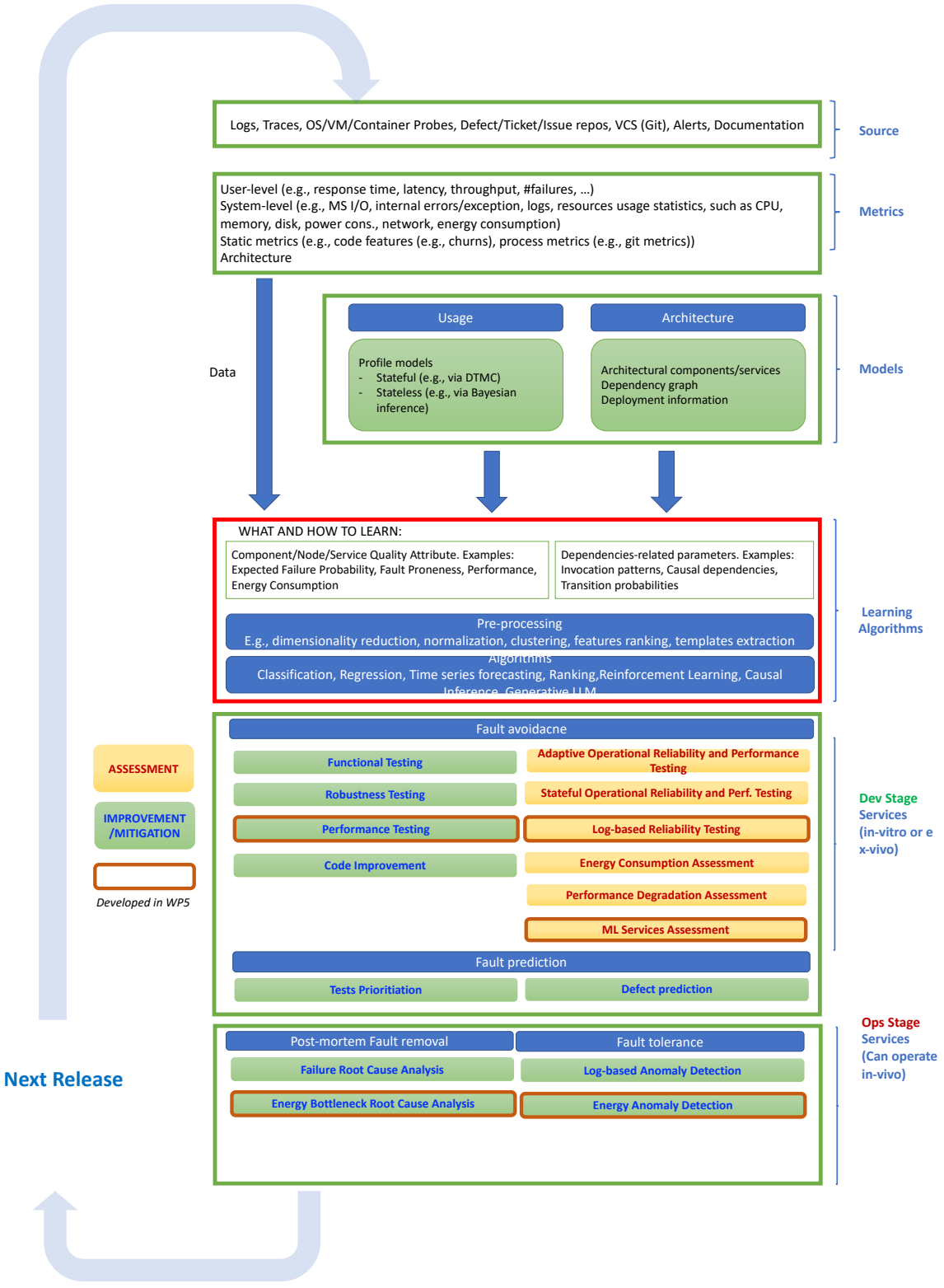

Figure 2.1. Testing and Assessment Techniques

Goal: Improvement

Scope: Fault Avoidance

Quality attribute: functional correctness, robustness

Internal name: `MacroHive`

Level of Maturity: 1

Link: MacroHive

- **Performance Testing**

  Description: Technique for generating critical performance configurations, to be applied at non-functional system testing stage.

  Stage: Dev

  Goal: Improvement

  Scope: Fault Avoidance

  Quality attribute: performance

  Internal name: `CallMIT`

  Level of Maturity: 1

  Link: CALLMIT

- **Code Improvement**

  Description: Technique for dead code elimination, to be applied at development stage as static code analysis.

  Stage: Dev

  Goal: Improvement

Scope: Fault Avoidance

Quality attribute: functional correctness

Internal name: `Lacuna`

Level of Maturity: 1

Link: Lacuna

- **Adaptive Operational Reliability Testing**

  Description: Ex-vivo technique using field data to derive tests. It is a *stateless* testing technique, where microservices are tested individually by generating invocations to the endpoints API by harnessing *adaptive sampling*.

  Stage: Dev

  Goal: Assessment

  Scope: Fault Avoidance

  Quality attribute: Reliability

  Internal name: `EMART`

  Level of Maturity: 1

  Link: EMART

- **Stateful Operational Reliability and Performance Testing**

  Description: Ex-vivo technique using field data to derive tests. It is a *stateful* testing technique, with a focus on continuous testing and monitoring. It involves replicating the observed runtime workload, in terms of type and intensity of requests, defining a profile for testing

that enables assessment of both reliability and performance.

Stage: Dev

Goal: Assessment

Scope: Fault Avoidance

Quality attribute: Reliability and Performance

Internal name: `MIPaRT`

Level of Maturity: 1

Link: MIPaRT

- **Log-based Reliability Testing**

  Description: Ex-vivo technique that exploits logs data for operational reliability assessment, as well as for fault detection and coverage driven by the observed behaviour in operation. Useful when operational data about specific inputs are expensive to collect (e.g., require instrumentation) and we want to minimize the manual intervention.

  Stage: Dev

  Goal: Assessment and Improvement

  Scope: Fault Avoidance

  Quality attribute: Reliability and Functional Correctness

  Internal name: `LoMiT`

  Level of Maturity: 2

  Link: LoMiT

- **Energy Consumption Assessment**

  Description: Analysis of energy consumption and performance metrics of web apps, IoT, and of monitoring tools in microservices.

  Stage: Dev

  Goal: Assessment

  Scope: Fault Avoidance

  Quality attribute: Energy, Performance

  Internal name: `N/A`

  Level of Maturity: 2, 3

  Link: Energy Consumption Assessment, Energy-Monitoring

- **ML Services Assessment**

  Description: Algorithms to assess accuracy of ML services based on sampling, for both image classifcation and LLMs. Useful for microservices wrapping ML-based functionalities.

  Stage: Dev

  Goal: Assessment

  Scope: Fault Avoidance

  Quality attribute: Reliability

  Internal name: `DeepSample`

  `DeepSample4LLM`

  Level of Maturity: 2

  Link: DeepSample, DeepSample4LLM

- **Test prioritization**

  Description: Given a list of tests to run, the goal of this service is to run first the ones more likely to expose failures. Prioritization is done by applying machine learning (learning-to-rank) algorithms to features of request/response and/or of the invoked Microservice that correlate more with quality metrics.

  Stage: Dev

  Goal: Improvement

  Scope: Fault Prediction

  Quality attribute: Functional Correctness

  Internal name: `Learning-To-Rank`

  Level of Maturity: 2

  Link: Learning-To-Rank

- **Defect Prediction**

  Description: The objective of this technique is to support the early identification of commits more likely to introduce defects, namely: given an application developed in a continuous integration/DevOps setting (hence with frequent commits), the goal is to alert on those commits more likely to introduce a defect in the deployed code. This is done by applying just-in-time (JIT) prediction enriched with the feature stability score computation.

  Stage: Dev

Goal: Improvement

Scope: Fault Prediction

Quality attribute: Functional Correctness

Internal name: `N/A`

Level of Maturity: 2

Link: Defect-Prediction

- **Failure Root Cause Analysis**

  Description: A service called `uKnows`, developed in the context of `MacroHive` service – see above - which, based on collected data, exploits causal reasoning to determine which microservices are responsible for failures and, in general, for erroneous behaviour

  Stage: Ops

  Goal: Improvement

  Scope: Fault Removal

  Quality attribute:

  Internal name: `uKnows`

  Level of Maturity: 1

  Link: uKnows

- **Log-based Anomlay Detection**

  Description: A log mining service for microservice-based systems that converts diverse log data into numeric representations without requiring prior format or application knowledge, that enables

effective anomaly detection by identifying patterns across multiple log sources.

Stage: Ops

Goal: Improvement

Scope: Fault Tolerance

Quality attribute: Reliability

Internal name: `Micro2vec`

Level of Maturity: 2

Link: Micro2vec

- **Energy Anomaly Detection**

  Description: Service encapsulating algorithms for anomaly detection to spot energy bottlenecks in a microservice system.

  Stage: Ops

  Goal: Improvement

  Scope: Fault Tolerance

  Quality attribute: Energy

  Internal name: `N/A`

  Level of Maturity: 3

  Link:

  Multivariate Energy AD & RCA

- **Energy Root Cause Analysis**

2.2. *REPOSITORY STRUCTURE*

Description: Service encapsulating algorithms for Root Cause Analysis to spot services causing energy bottlenecks in a microservice system.

Stage: Ops

Goal: Improvement

Scope: Fault Tolerance

Quality attribute: Energy

Internal name: `N/A`

Level of Maturity: 3

Link:

Multivariate Energy AD & RCA