



## Industrial-grade Verification and Validation of Evolving Systems

Labelled in ITEA3, a EUREKA cluster, Call 5

ITEA3 Project Number 18022

### D3.4 Final validation methods and techniques for evolving systems considering feedback from the first evaluation

Due date of deliverable: 31 March 2022 (M30)

Actual date of submission: 28 March 2022

**Start date of project:** 1 October 2019

**Duration:** 39 months

**Organisation name of lead contractor for this deliverable:**

CRIM

**Author(s):** Tatu Aalto, Joonas Oikarinen, Sorin Patrasciu (F-Secure), Pekka Aho (OUNL), Alexandre Petrenko (CRIM), Mahshid Helali Moghadam (RISE), Tijana Nikolic, Almira Pillay (Sogeti), Eva Garcia Martin (Ekkono)

**Status:** completed

**Version number:** V1

**Submission Date:** 28-03-2021

**Doc reference:** IVVES\_Deliverable\_D3.4\_V1.docx

**Work Pack.** WP3

**Task:** T3.1, T3.2, T3.3

**Description:**  
(max 5 lines) The report presents the final validation methods and techniques developed for evolving systems.

<b>Nature:</b>	<input checked="" type="checkbox"/> <b>R</b> =Report, <input type="checkbox"/> <b>P</b> =Prototype, <input type="checkbox"/> <b>D</b> =Demonstrator, <input type="checkbox"/> <b>O</b> =Other		
<b>Dissemination Level:</b>	<b>PU</b>	Public	<b>X</b>
	<b>PP</b>	Restricted to other program participants	
	<b>RE</b>	Restricted to a group specified by the consortium	
	<b>CO</b>	Confidential, only for members of the consortium	

## Table of Contents

1.....	Executive summary	5
2.....	Test generation and test prioritization for fault detection	6
2.1. ...	Scriptless E2E test generation for ES with coverage analysis (OUNL, Innspire, Marviq, F-Secure, ING).....	6
2.1.1. ....	State of the art	6
2.1.2. ....	Contributions	7
2.1.3. ....	Claimed novelty	8
2.2. ....	Machine learning-assisted automated performance testing (RISE)	8
2.2.1. ....	Performance testing: State of the art	8
2.2.2. ....	Contributions	9
2.3. ....	Test generation and prioritization for ESG-investment (SII CONCATTEL)	11
2.3.1. ....	State of the art	11
2.3.2. ....	Contributions	13
2.3.3. ....	Claimed novelty	13
2.4. ....	Anomaly detection for industrial environments (KeyLand)	13
2.4.1. ....	State of the art	13
2.4.2. ....	Contributions	16
2.4.3. ....	Claimed novelty	17
3.....	Flaky tests-based detection (F-Secure, University of Helsinki)	17
3.1. ....	State of the art	18

3.2	Contributions	19
3.3	Claimed novelty	19
4	Test failure root cause analysis (F-Secure, University of Helsinki)	19
4.1	State of the art	20
4.2	Contributions	20
4.3	Claimed novelty	20
5	Oracle mining (CRIM)	20
5.1	Introduction	20
5.2	State of the art	21
5.3	Contributions	22
5.4	Claimed novelty	24
6	Automated test verdict generation via Model Learning (F-Secure, OUNL)	25
6.1	Description (Change-Analyzer)	25
6.2	Example (Change-Analyzer)	25
6.3	State of the art	28
6.4	Anticipated contributions (Change-Analyzer)	28
6.5	Contributions (Testar)	29
7	Conformal prediction for edge applications (Ekkono Solutions)	29
7.1	Introduction	29
7.2	Technical description	29

7.3	..... Example	30
7.4	..... State of the art and anticipated contribution	30
8	..... Code defect risk prediction (Sogeti)	31
8.1	..... Introduction	31
8.2	..... Anticipated contribution	31
8.3	..... Proposed approach	31
8.4	..... DevOps pipeline: integration and traceability	34
8.5	..... Claimed novelty	35
9	..... References	36

## 1. Executive summary

Work package three focuses on solving problems related to complex evolving system, which can be ML based or traditional software projects. The problems address different aspects of evolving systems, starting from automated test case generation, either based on UI of the application or processing different textual sources, to analyzing test result and all the way to different physical aspects of a device.

Scriptless test generation aims to ease the E2E testing by applying both, non-ML and ML techniques to generate test suites with a high coverage level. The project also provides automated change detection and visualization based on comparing inferred state models of consequent SUT versions.

Machine learning-assisted automated performance testing aims to solve the complex requirements in the performance testing area. It uses a reinforcement learning (RL) algorithm to find optimal test cases detecting performance problems.

Test generation and prioritization aims to process different textual data sources with an NPL algorithm and then use RL-based test generation and prioritization.

Research in anomaly detection for industrial environments has analyzed currently existing techniques and tools and is in underway in building a tool to provide more comprehensive view in the current software lifecycle.

Flaky tests-based detection tries to solve a problem related to randomly failing tests by analyzing previous tests data and displaying result in command line.

Test failure root cause analysis is in early stages but tries to provide categories test failures in similarity groups.

Oracle mining uses machine learning to process requirements and tries to provide precise oracles in a two-step process.

Automating test verdict generation via model learning aims to build models from the application UI and display changes between different application versions.

Conformal prediction for edge applications tries to predict with machine learning when hardware might break by collecting data from sensors.

## 2. Test generation and test prioritization for fault detection

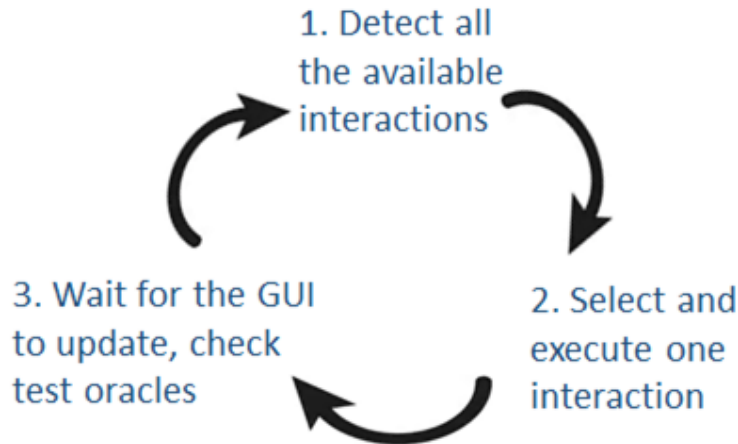
### 2.1. Scriptless E2E test generation for ES with coverage analysis (OUNL, Innspire, Marviq, F-Secure, ING)

Building a test automation pipeline of complex evolving systems (ES) is an elaborate task, especially on the end-to-end (E2E) level. One of the main challenges is the fact that the resulting test suites are difficult to maintain. We plan to address this challenge by applying both, non-ML and ML techniques to generate test suites with a high coverage level. Techniques, methodologies, and tools are needed to be able to:

- generate test cases automatically with a good level of interpretability for
  - different stages of test automation (TA);
  - different types of applications: standalone, web, and mobile applications;
- reuse generated test suites and inferred application models to optimize TA in terms of time and coverage.

#### 2.1.1. State of the art

Traditionally, software test automation is based on scripts (pre-defined test sequences with test oracle checks) that are either automatically generated from models or written by a human. In scriptless test automation, the test sequences are generated dynamically during the execution, usually one step at a time, based on automatically detected available interactions that the end user could perform, or events from the environment. The execution of the action includes waiting for the reaction from the system under test (SUT). Figure 2.1 depicts the process of scriptless testing at a high level.



**Figure 2.1:** Scriptless testing process.

The action selection often involves some level of randomness, and therefore scriptless GUI testing is often called random or monkey testing. One of the research directions for trying to make monkey testing tools smarter has been using AI and machine learning for improving action selection. Usually, some kind of model inference approach has to be used for the learning process. Often, the reward or fitness functions have been connected to increased GUI or code coverage [7], [8]. This kind of strategy usually rewards visiting (i.e., covering) all GUI states at least once. However, visiting all GUI states does not mean that all paths or combinations of paths have been visited.

Inferring state models during automated GUI exploration has been researched with various approaches, for example, GUITAM [9], GUI Driver [10], and Crawljax [11]. However, automated change detection by comparing inferred models of consequent system versions has not been widely researched; Murphy tools [12] seems to be the only existing approach in the literature.

### 2.1.2. Contributions

Open source TESTAR tool is being used for scriptless graphical user interface (GUI) test generation and state model inference in continuous integration (CI) environment for selected software packages in the use cases of F-Secure and ING. TESTAR dynamically generates test sequences during the exploration of SUT, one step at a time, based on detected available actions. TESTAR supports state model inference during the automated exploration of the GUI of an application and uses the model for systematically (but in random order) trying out all the available actions in all the explored states.

During IVVES project, TESTAR has been extended to use the inferred models for reinforcement learning with various reward functions to improve action selection and automated change analysis based on comparing the inferred state model of consequent SUT versions. During the evaluation, we noticed that various SUTs required different levels of abstraction. Therefore, TESTAR was extended with a feature that allows configuring the state abstraction and action abstraction for each SUT. There are also two new open-source tools for automated change detection and visualizing the changes for the end user. One of the tools

is using the database containing the state models inferred by TESTAR. The other one is a stand-alone tool implemented by F-Secure, including also the GUI exploration.

For Windows desktop applications, TESTAR uses Windows accessibility API to access the GUI information for detecting the state of the SUT and available actions. For web applications, TESTAR uses Selenium WebDriver. During IVVES project, TESTAR has been extended to support mobile applications by using Appium to connect to a mobile device emulator.

To increase the speed of state model inference and GUI exploration, TESTAR has been containerized for Docker and extended with support for distributed/parallel GUI exploration with shared state model database.

To increase the effectiveness, TESTAR has been integrated with SonarCube for code smell analysis and directing GUI testing into the risky parts of the code based on previously measured code coverage foot-prints of each GUI action.

### 2.1.3. Claimed novelty

The novel aspects of TESTAR extensions include:

- using inferred state models for machine learning to improve action selection in scriptless GUI testing,
- distributed/parallel model inference and GUI exploration with shared state model,
- using code coverage foot-prints of GUI actions to direct TESTAR action selection to cover specific parts of the code,
- support for testing mobile applications with TESTAR,
- automated change detection and visualization based on comparing inferred state models of consequent SUT versions.

New publications have been written and more are expected from the results.

## 2.2. Machine learning-assisted automated performance testing (RISE)

### 2.2.1. Performance testing: State of the art

Performance, which is also called efficiency in some taxonomies, is a quality characteristic which describes the time and resource bound aspects of a system's behavior and is of great importance for the success of many products. It's measured in terms of some metrics like response time, throughput and resource utilization.

Performance analysis is typically done to measure performance metrics and detect performance-related issues. Performance issues could be referred to as any violation of performance requirements or any functional problems emerged under special performance-related conditions such as heavy workload and limited resource availability. Several methods have been developed to analyze the performance of software products. Performance modelling and performance testing are two main classes of techniques used for performance analysis. Modeling methods [13, 14, 15, 16, 17] are mainly based on the performance models



extracted from the system model or the source code of the target system. While, in performance testing, certain test cases (scenarios) are generated and applied during the execution of the SUT to trigger performance failures, e.g., violations of performance requirements. Many of the common performance testing approaches such as techniques based on source code analysis [18], system model analysis [19, 20], user behavioral models [21], and declarative behavior specifications [22, 23] mostly rely on source code or system models.

*Performance testing challenges.* Software performance testing to find performance issues upon new changes mainly occurred within CI/CD practice is always a challenging task. Therefore, automated performance testing is of importance in this regard and subsequently one of the primary concrete challenges in software performance testing is generating appropriate test cases (test scenarios) in an efficient and cost-effective way. Performance test scenarios are intended to detect performance degradation issues. By detecting performance degradation issues at an early phase, the changes leading to the degradation is easier to localize and could be actively decided upon, and consequently the extra cost at the customer side due to degraded system performance can be avoided.

Currently, the performance test is done manually for many software products. However, due to the increasing complexity and diversity of the products and the services requested by customers, the need to automate the performance testing process is highlighted. In the manual testing process, several test cases have to be executed upon each change in the software. Not only, the manual process is time-consuming and laborious, but also it is error-prone and does not provide any correctness guarantee since it relies on expert knowledge.

On the one hand, in many cases in order to generate test cases, we usually have a large input space to explore which makes manual test case generation a time-consuming and laborious task. On the other hand, we usually do not have any knowledge about the internal structure and dynamics of the SUT, and the interaction with its public interface is the only way to learn about the SUT's performance characteristics. Last but not least, we usually have a limited test budget, which means that we are not free to execute a large set of test cases hoping that some of them will reveal performance defects. Instead, we need to be careful about the test cases that we select for running on the SUT.

Taking advantage of the recent advances in machine learning, machine learning-assisted techniques have been used widely for meeting the need for automated performance testing. In our work, we propose reinforcement learning-assisted approaches which learn from the behavior of the SUT to generate promising/effective test cases automatically and efficiently. The details of the proposed techniques are presented in the following sections.

### 2.2.2. Contributions

Reinforcement learning (RL) [24] is a fundamental machine learning paradigm which is mainly intended to address decision making problems. Inspired by human's learning, RL is used to find the optimal way to make decisions. The learning procedure is quite different from supervised and unsupervised learning algorithms. The learning is based on continuous interaction between a smart RL agent and the problem environment which is system under

test (SUT) in our research case. At each step of interaction, the smart test agent observes the status of the environment and makes a decision. The decision is generating a test scenario, e.g., based on changing the variables involved in forming the test scenario. Then the SUT is executed under the recommended test scenario, and the test agent receives a reward signal indicating the effectiveness of the recommended test scenario. One of the main differences between RL and other learning paradigms is that there is no supervisor in RL, i.e., the agent just receives a reward signal from the environment, and the agent goes through the environment based on a sequential decision-making process.

With regard to the characteristic of RL, we proposed that if the optimal policy (way) for accomplishing the intended performance test objective could be learned by a test agent, the test could be done automatically without need to access to source code or performance/system models. Moreover, once the optimal policy is learned, it can be reused in further testing situations, for example, regression performance testing of a SUT or performance testing of SUTs with similar performance sensitivity to resources [25]. Therefore, the capability of knowledge formation and reusing the gained knowledge in further situations is a key feature leading to test efficiency improvement. Based on this idea, we have proposed an RL-assisted performance testing framework that learns the optimal policy to accomplish the intended test objective without access to system model or source code of SUT. Once it learns, it is able to reuse the learned policy in further testing cases [25, 26, 27]. The proposed framework consists of two performance testing tools: SaFReL [28] and RELOAD [25, 29].

SaFReL, as a self-adaptive fuzzy reinforcement learning test agent which generates performance platform-based test cases, learns how to tune the resource availability to reach an intended performance breaking point for different types of SUTs with different levels of sensitivity to resources. It assumes two phases of learning: initial and transfer learning phases. First, it learns the optimal policy to reach the intended performance breaking point for different types of SUTs, i.e., CPU-intensive, memory-intensive and disk-intensive software. Once learning the optimal policy, it replays the learned policy on further similar SUTs. The conducted experimental evaluation shows that SaFReL can perform efficiently and adaptively on different software programs, i.e., CPU-intensive, memory-intensive and disk-intensive SUTs running on various hardware configurations and with different response time requirements. SaFReL accomplishes the intended test objective, i.e., finding performance breaking point, more efficiently in comparison to a typical stress testing technique which generates performance test cases in an exploratory way. SaFReL leads to reduced cost in terms of computation time by reusing the learned policy upon the SUTs with similar performance sensitivity [28].

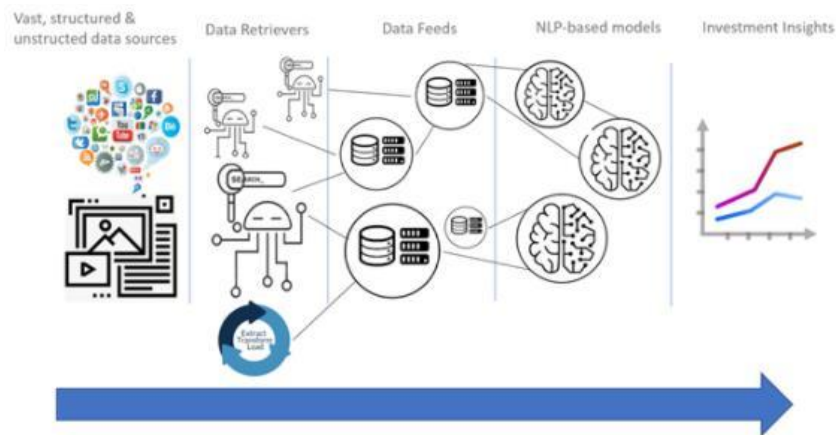
RELOAD is an adaptive RL-driven load testing agent which learns how to tune the load of transactions in the submitted workload to the SUT to accomplish the test objective (e.g., finding an intended performance breaking point). It learns the optimal policy to generate a cost-efficient workload to meet the test objective during an initial learning, then it is able to reuse the learned policy in later tests within the continuous testing context, e.g., for meeting further similar test objectives. RELOAD generates a more accurate and efficient workload to accomplish the test objective compared with baseline and random load testing techniques,

without access to source code or system models. Moreover, once it learns, it is able to reuse the learned policy in further situations, i.e., testing w.r.t. different objectives on the SUT and still keeps the improved efficiency over later test episodes [29].

## 2.3. Test generation and prioritization for ESG-investment (SII CONCATEL)

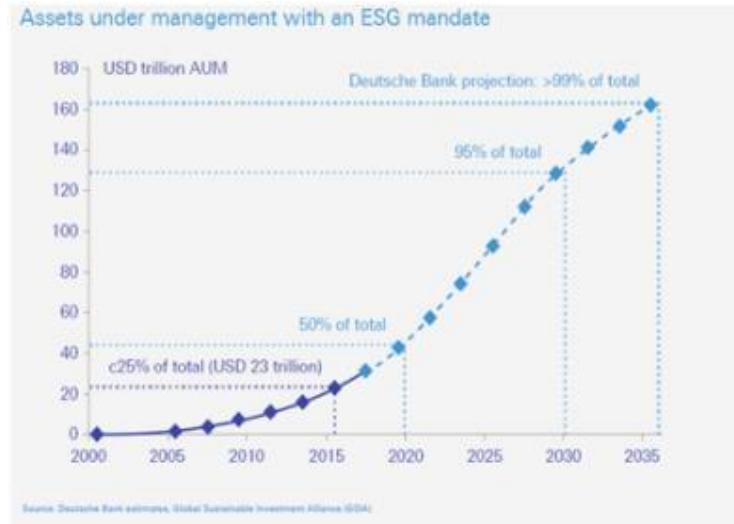
### 2.3.1. State of the art

The irruption of game-changing innovations and open-source technologies in NLP is changing the way that companies work with text. Unstructured text is being used as input data for many industrial domains (i.e., predicting market trends based on sentiment analysis). Data Analytics companies are curating and collating text information from diverse sources to feed AI models (Figure 2.2) and provide trends and insights. Its combination with other AI techniques applied to numerical data is fostering the integration of NLP into regular Data Analysis.



**Figure 2.2:** General workflow for NLP-based models to provide trends and insights in finance sector

ESG (Environmental, Social and Governance) investing refers to a class of investing that is also known as “sustainable investing.” This is an umbrella term for investments that seek positive returns and long-term impact on society, environment and the performance of the business. To assess a company (an asset) based on environmental, social, and governance (ESG) criteria, investors look at a broad range of behaviors to set the ESG score for a given asset.



**Figure 2.3:** Estimates of assets under management with an ESG mandate.  
Source: Deutsche Bank.

The compilation of these scores is based on the analysis of a vast amount of fast changing alternative data sources, including non-structured information (websites, news, corporate reporting...) that can't be processed with traditional keyword searches and manual analysis. Since ESG investment is a solid trend that is increasingly impacting the market (Figure 2.3), the data sources to analyze are growing fast. Given the vast amount of data and information available, that analysis can only be reliably carried out with artificial intelligence. New, powerful AI-based systems are now on the scene that can potentially reduce manual tasks and increase efficiency. However, new V&V techniques are required:

- The growth of AI-based analysis of sources has also impacted the way that companies communicate with the external audience, being savvier with their wording. This is causing the appearance of biased content, that must be taken into account before applying NLP-based techniques -heavily relying on sentiment analysis- to get insights and trends. Hence, a systematic and continuous analysis of source credibility and content credibility must be implemented.
- The AI-based systems must continuously adapt to a great variability in sources and content, that are constantly changing. Information sources evolve, mutate and topics related to ESG change over time. Hence, test maintenance and prioritization are challenging.

Added to this, since the outcomes of the evolving systems are insights that may impact investment decisions, these systems are subject to regtech (regulatory technology) constraints that must be taken into account.

There are different approaches for test case Reinforcement Learning-based test case prioritization. Zhaolin, et al. [83], provides a reference for test case prioritization to save computing resources in Continuous Integration. In [83], a novel reward function is proposed, by using partial historical information of test cases effectively for fast feedback and cost reduction. The approach is focusing on reducing the huge cost in terms of time and resource

availability defining the Average Percentage of Historical Failure with time Window (APHFW), as a novel reinforcement learning reward function, that utilizes a time window to filter recent historical information to calculate reward value.

### 2.3.2. Contributions

The main technical contribution is supporting testing based on Reinforcement Learning for NLP-based ESG evolving systems. Specifically, given an ESG-investment-focused ES and a set of rules defined by an expert for scoring securities with respect to ESG criteria, develop masked language models.



**Figure 2.4:** Templating with masked language models.

The RL-based test generation and prioritization is based in a time window-based reward function that will also take into account the most effective Metamorphic Relations for a given case. For the selection of effective Metamorphic Relations, the model will initially interact with the templates generated in WP2 with masked language models (Figure 2.4), and eventually will control “Plug” operators.

As an outcome, the most effective Metamorphic Relations will be selected to generate the optimal test cases to execute, taking also into account performance.

### 2.3.3. Claimed novelty

- ESG news validation to train the Fintech model
- XAI applied to NLP and keyword recognition
- Automatic detection of fake news.

## 2.4. Anomaly detection for industrial environments (KeyLand)

### 2.4.1. State of the art

KEYLAND has conducted a comprehensive state-of-the-art analysis of the techniques, methodologies and the use of technologies to detect the maintenance needs of AAS, and to reflect the changes that have occurred in the physical system of the machine learning model.

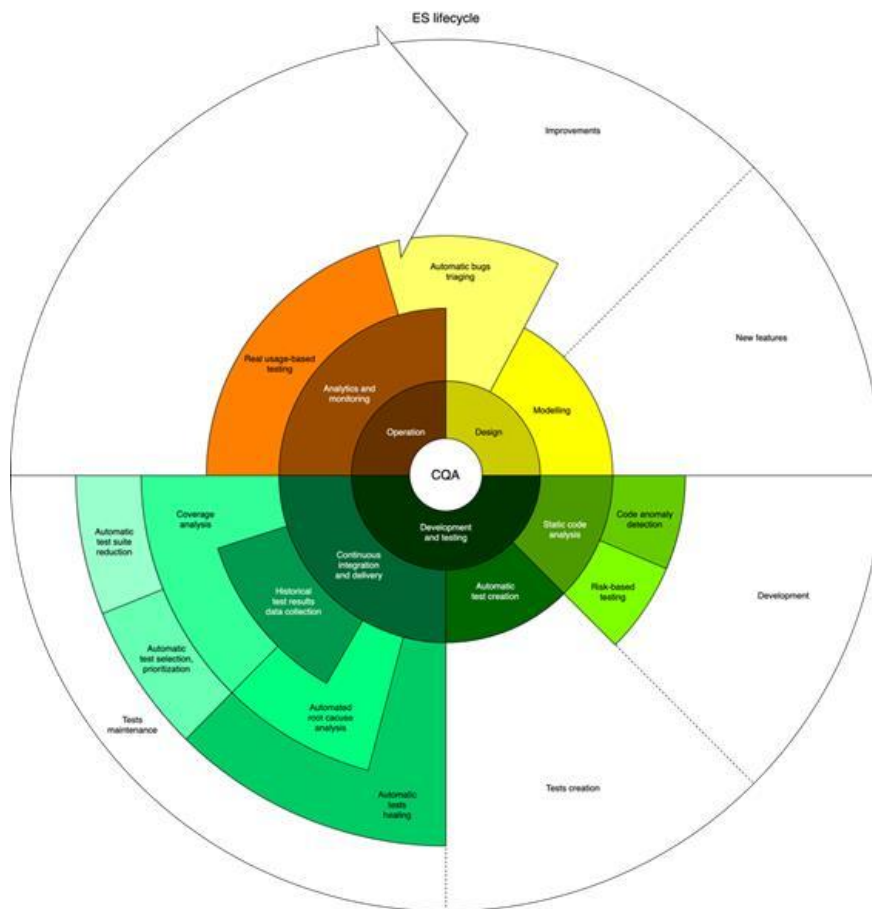
An extensive analysis of the methods and techniques for dynamically assessing and cataloging the quality of the data sets used has also been carried out through an analysis of the different proposals.

A state-of-the-art analysis of the techniques and technologies based on machine learning tests that can be applicable in the context of the industrial sector has been carried out, taking into account all the scenarios and use cases involved in this project.

These analysis tasks have allowed the creation of a common framework for all existing methods, techniques and tools to be used.

This will also allow the development of components capable of working with data collected from different sources.

In order to have a clearer, global picture of the current state of validation techniques for SAAs, at IVVES we have mapped the ES lifecycle in a two-dimensional space where the vertical axis (Y) is used to measure complexity and the horizontal lifecycle stage (X). Figure 2.5 shows that mapping. Since software development is an iterative process, after the production release phase, the design starts again. It is important to understand that the techniques used are based on simpler ones. The "automatic test prioritization" is almost impossible to implement without consolidated components in the context of "CI/CD" and "test result data collection".



**Figure 2.5:** Lifecycle of SAAs

A summary of different existing techniques and tools (which can be integrated in different phases of the project) is shown in the following table:

Technique	Tests level	State	Adoption level	Tools (if available)
Design				
New features				

Modelling					
Threat Modelling	Req.	P	High	<ul style="list-style-type: none"> <li>- STRIDE</li> <li>- P.A.S.T.A.</li> <li>- Trike</li> <li>- VAST</li> </ul>	
TLA	Req.	A	Low	- TLA toolbox	
Improvements					
Automatic bugs triaging	Req., E2E	A	Medium	- CERT Triage tool / Exploitable	
Development and Testing					
Development					
Static code analysis	Unit	P	High	<ul style="list-style-type: none"> <li>- SonarQube</li> <li>- Language specific IDEs, linters and analysis tools</li> </ul>	
Code anomaly detection	Unit	A	Low	- REPD	
Formal Verification	Req.	P	Medium	<ul style="list-style-type: none"> <li>- Uppaal</li> <li>- PRISM</li> <li>- Rebeca (Afra)</li> </ul>	
Risk-based testing	Unit, Int., E2E	P	Medium		
Tests creation					
Automatic tests creation					
Fuzzing	Int., E2E	A	Medium	<ul style="list-style-type: none"> <li>- LibFuzzer etc</li> <li>- American Fuzzy Loop</li> <li>- AddressSanitizer, ThreadSanitizer, MemorySanitizer</li> <li>- OSSFuzz</li> </ul>	
Metamorphic testing	Unit, Int., E2E	A	Low		
Search-based testing	Unit, Int., E2E	A	Medium	<ul style="list-style-type: none"> <li>- EvoSuite</li> <li>- Randoop</li> <li>- Microsoft IntelliTest</li> <li>- DiffBlue Cover</li> </ul>	
Model-based testing	E2E	A	Medium	<ul style="list-style-type: none"> <li>- Test Modeller</li> <li>- APOGEN with Crawljax</li> <li>- ALEX</li> </ul>	
ML-based testing (model free reinforcement learning)	Unit, E2E	A	Low	<ul style="list-style-type: none"> <li>- RELOAD</li> <li>- SaFrEL</li> </ul>	
Tests maintenance					
Automatic test selection and prioritization	Unit, Int., E2E	A	Low	- TestArchiver and ChangeEngine by SALabs	

Automatic root cause analysis	Unit, Int., E2E	A	Low	- Functionize platform - Delta debugging tools
Automatic test suite reduction	Unit, Int., E2E	A	Low	
Automatic healing	Unit, Int., E2E	A	Low	- Functionize platform
<b>Operation</b>				
Analytics and monitoring	E2E	P	High	- AWS CloudWatch - New Relic - Kibana - Google Analytics - Matomo
Real usage-based testing	E2E, Int.	A	Low	

### 2.4.2. Contributions

We are still developing a component whose implement a kind of monitoring and diagnostic module, in which, through the metrics collected by monitoring tools such as Istio or Prometheus, we will try to monitor the system to look for possible operating problems, either due to external factors (machine, network, ...) or internal (malfunctioning of some parts of the system, anomalous input or output values, ...).

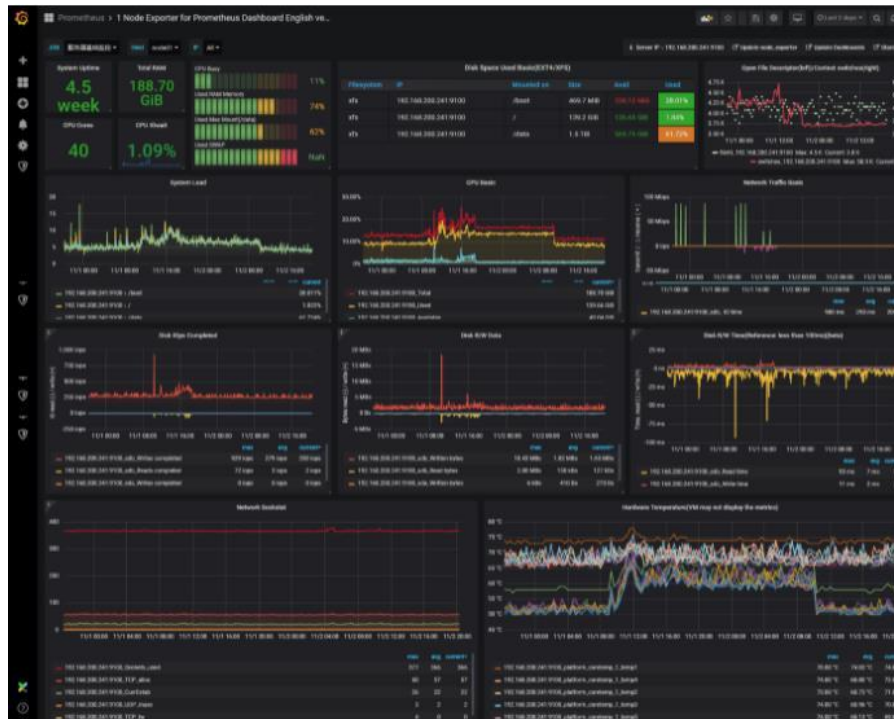
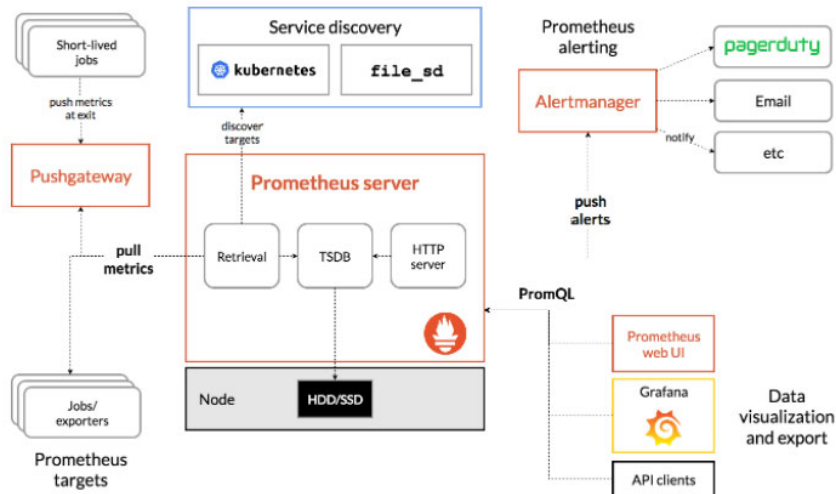


Figure 2.6: Example of Prometheus + Grafana dashboard





**Figure 2.7:** Prometheus Architecture  
[\(https://prometheus.io/docs/introduction/overview/\)](https://prometheus.io/docs/introduction/overview/)

Then, depending on the type of problem detected, we will try to perform a root cause analysis to try to isolate which of the analyzed components are causing some kind of problem, either individually or in combination with each other.

### 2.4.3. Claimed novelty

- Automatic fault detection in industrial environments
- Generator of possible causes of internal or external failures in industrial environments.

## 3. Flaky tests-based detection (F-Secure, University of Helsinki)

In continuous integration and continuous delivery (CI/CD) pipelines, where there usually are lots of tests and test execution is frequent, going through all test results is not possible by humans. Especially in integration and acceptance test level, there are many reasons why a test may randomly fail. For example, the application under test may have integrations to an external system or a system which cannot be controlled by a team developing the application under test. When these external systems suffer a downtime or from bug, it may cause tests to randomly fail. Please note that there are numerous reasons why random failure might happen and it is not limited to the previous example. These random failures, if there are many of them, can reduce the trustworthiness on the CI/CD pipeline.

When people responsible for the CI/CD pipeline review the test results, usually there are not enough resources to review all results or even all the failures from the different executions. Experts usually resort to looking at the latest failures and perhaps looking at the first failure and assume that all failures are originating from the same reason. If there are multiple CI jobs, they might look for failures in a single job and assume that all the jobs have the same reason for failure. Therefore, experts might have different opinions about what test fails randomly and how often random failures happen.

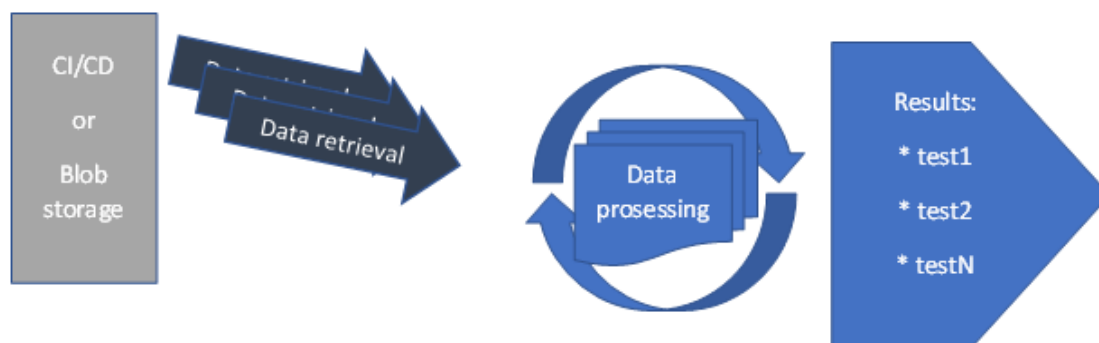
### 3.1. State of the art

Because random failures may reduce the trustworthiness of the CI/CD pipeline, traditionally test results are analyzed manually at looking results from the CI/CD pipeline. When there are random failures in the pipeline, analyzing results and coming to resolution is manual and time-consuming work. That time is away from more important work, for example, new feature development. Currently many CI systems are capable of visualizing the latest execution results or show trends of failing/passing/skipped for a time period. Although many CI/CD systems allow configure how long execution results are kept, many development teams choose to limit the period as narrow as possible, because usually it is expensive to keep results in the CI/CD pipeline.

Also, systems that can provide better visualization services, than usually are available in the CI/CD system, are available for but usually those systems require sending data to external system. Also, these services require effort to set up and maintain.

To get continuous and comprehensive feedback from their CI/CD system, flaky test detection was developed. Flaky test detection has three different parts and all parts are running inside of the CI/CD pipeline.

1. Processing xunit result in xml format
2. Finding the flakiest tests from the results and presenting result easy to understand format.
3. There is also example integration script for GitHub actions, which can be used download xunit result from GitHub actions artifacts.



The flaky test tool process is illustrated in the figure above. First data is retrieved either from CI/CD pipeline or from other storage where xunit results are kept. Tool provides example integration script how xunit result can retrieved from GitHub actions and similar technique can be usually applied in the other CI/CD pipelines or a blob storage. After data retrieval, tool will process the xunit result and calculate based its algorithm which tests are flaky. Tests that change test often and state changes are most resent are more important than tests which are further in history or have less state changes. Tool also provides few options which can be used to adjust the algorithm how test state changes are counted and how many flaky tests are presented in the results. After results are calculated, the tool prints out a simple table of tests which are the flakiest. Table can be viewed from command like STDOUT and optionally tool can be configured to save the result as heatmap image file.

## 3.2. Contributions

Flaky test detection tool is open source and is published under Apache license version 2 in GitHub. It is also available as PyPi package or if used from GitHub actions, it runs as an action in the pipeline.

For the data retrieval part, we provide an example script how to integrate with GitHub actions and how to download artifacts from GitHub actions. For demonstration purposes, there is a separate GitHub project which shows the tool in use, with full integration on GitHub actions.

Development teams can choose the method which is best suitable for them, from zero install in the GitHub Marketplace to fully managed install from PyPi.

## 3.3. Claimed novelty

Jonathan Bell et al. [98] have proposed “a general purpose, lightweight technique for detecting flaky tests by tracking differential coverage”. The technique has an open-source implementation for Java, called “DeFlaker”. Gustavo Pinto et al. [102] have researched how prevalent flaky tests are and have concluded that flaky tests are relatively common in IO projects. They have also noticed that detecting flakiness with test reruns is challenging. Wing Lam et al. [101] have done research on categorizing flaky tests and have come up with two distinct classes of flaky tests; order-dependent and order-independent. They find that most flaky tests are order-dependent. They also introduce a guideline that states that a test should be run only up to five times to reveal flakiness. Alex Grose and Josie Holmes [99] propose “a number of formal definitions of types of nondeterminism (horizontal and vertical) and an implementation, based on these definitions, for detecting and debugging nondeterminism in property-based testing”. Wing Lam et al. [100] propose a technique for enhancing regression test selection by making it dependent-test-aware.

The flaky test detection tool does relay in code coverage calculation or does not require doing analysis in the test or application source code. Example in compiled languages, like C++, calculating code coverage in acceptance test level is complex or is not possible during the runtime of the application. The tool changes the view to the problem and only looks at the test results to determine the flakiness of the tests. As in input data tool uses xunit xml, which is available out of the box in many testing frameworks.

## 4. Test failure root cause analysis (F-Secure, University of Helsinki)

When doing acceptance or integration testing, tests have connections to external systems or depend on a resource which cannot be controlled by team developing. In specially in the acceptance test level, individual test tends to be complex, example test might requires setting up data, driving the application to certain state and many more action before the test reaches the point where it starts the actual testing process. Also, when test setups the application, data or configuration, it is usually test responsibility to clean up the done setups. Because of the complexity in the test, single and same test may fail in multiple different ways. In continuous integration and continuous delivery (CI/CD) pipelines, where there usually is lots of test and test execution interval is frequent, going through all test results is not possible by humans.

When people responsible for the CI/CD pipeline review the test result, usually there are not enough resources to review all results or even all the failures from the different executions. People usually result looking at the latest failures and perhaps looking at the first failure and assume that all failures are originating from the same reason. If there are multiple CI jobs, people might look for failures in a single job and assume that all the jobs have the same reason for failure. Therefore, different people will have different opinions about what is the reason for the failure in the individual test, because test might have been failing in multiple reasons in different test execution runs in the pipeline.

#### 4.1. State of the art

Usually CI/CD systems or other data visualization systems display results based on the single execution in the pipeline and do not take account of the history of execution. Or pipeline can display results as trend, but do not do deeper analysis on the failure reason. Because single test may fail in multiple different ways in different executions in the pipeline or different tests may fail on same underlying reason, just displaying trends or looking at the single execution result is not sufficient to determine the root cause of the test failure.

Instead of counting tests results and doing root cause analysis manually based on the test results, we think that it is possible to parse the test result from the test execution logs and apply machine learning model to group or label similar failures. We are currently in the early stages of the study but based on the initial analysis we believe that it is possible to use ML to find similar failures from test logs. If grouping can be done by ML algorithm, it can be more accurate and can process more data than a human would go through in a manual process. This would shorten the time taken in the analysis of test results and saved time could be more efficiently used to fix the problem.

#### 4.2. Contributions

Currently we do not have any public contributions, because we are in the early stages of research. Initial results look promising, with sample data provided from F-Secure test automation. If the results are good, we aim to publish this tool as an open-source package in GitHub with University of Helsinki.

#### 4.3. Claimed novelty

Currently there are various visualization systems for test results, but the existing systems neither perform analysis on deeper levels nor perform root cause analysis.

### 5. Oracle mining (CRIM)

#### 5.1. Introduction

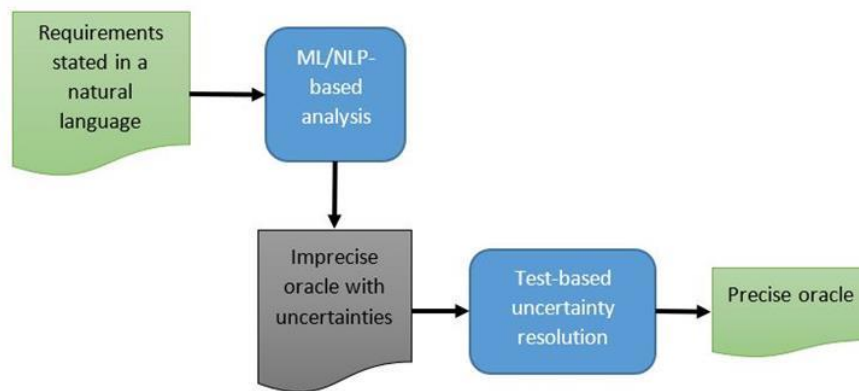
Automating the test generation from requirements expressed in ambiguous natural languages is challenging, even for controlled ones. It can be decomposed into two steps: the automatic generation of formal specifications and test generation from formal specifications. Formal specification plays the role of an oracle in testing, i.e., it specifies the relation between the inputs and the expected outputs. However, constructing formal specifications is a very

challenging task. CS Canada and IVVES industrial partners are facing this challenge in testing (critical) evolving systems. Requirements describe features and functionalities of systems in terms of constraints that must hold on variables that represent concepts (e.g., input, outputs, and states) of the systems. In addition to the variable names, ambiguous words and punctuation marks from the natural languages (e.g., when, if, after, while, where, and, or, do, make, set, etc.) appear in the constraints. The ambiguity of the meaning of some words and marks, the usage of multiple variable names for the same concepts introduce uncertainty in the requirement analysis. For example, a part of a requirement can be enhanced by connecting it to a new part of the requirement via the usage of the word "where"; it is not obvious to determine the connected parts of requirements. The uncertainty leads to various interpretations of each ambiguous part of the requirements and combinations of the interpretations result in a possible vast number of plausible specifications. Approaches are needed to choose proper specifications.

## 5.2. State of the art

Most of the approaches to generate precise oracles or tests from requirement documents are fully automated and aims at producing precise oracles [2, 6, 4, 1, 5]. The direct translation approach makes correspondence between certain patterns in the modelling language for precise oracles and their possible representations in natural languages. The machine translation-based approach uses examples of translated requirements either to infer formal grammar for the requirements or to train a translation model with ML techniques. The approach in [4] automatically generates a precise oracle, which is compared to a manually generated precise oracle for validation of the automatic precise oracle generation procedure. In case the generated oracle is not the expected one, the approach does not propose another version to the expert.

Our contribution is a two-steps approach to generate precise oracles, as illustrated in Figure 6.1. The first step consists in generating imprecise oracles representing a set of plausible precise oracles (specifications). The second step is mining a precise oracle from the imprecise one. Mining a precise oracle corresponds to the resolution of uncertainty in the imprecise oracle in order to choose one of the plausible precise oracles. We suggest involving an expert in realizing this task.



### Figure 5.1: Two-steps approach to generate precise oracles

The next section presents our approach to mining a precise oracle from an imprecise one.

## 5.3. Contributions

In our approach, we represent uncertainty with nondeterministic transitions in a finite state machine. Finite state machine has been used as a formal model for evolving systems [1] and used in developing verification and validation techniques for evolving systems such as model-based testing and model-checking.

We represent a set of plausible precise oracles, called the imprecise oracle, with an input complete and non-deterministic finite state machine (FSM). A plausible precise oracle is then a deterministic and input complete submachine of the imprecise one. Each precise oracle produces a single output sequence in response to an input sequence. A test is nothing else but an input sequence. Precise oracles are distinguishable if they produce different output sequences for the same test; otherwise, they are indistinguishable.

Our approach to assisting an expert in choosing a proper precise oracle from an imprecise one works as follows:

- Randomly generate a test
- **Loop:** Determine the outputs which can be produced by the executions of the plausible precise oracles with the test; this is done by exploring paths of the imprecise oracle
- Ask an expert to choose the expected output; we assume that one of the outputs is expected.
- Remove from the imprecise oracle the precise ones that do not produce the chosen output sequence
- If the imprecise oracle contains only indistinguishable precise oracles, then **return** any one of the precise oracles as expected and **exit**
- Else generate a test that distinguishes two precise oracles in it and **goto loop**

Let us illustrate our contribution. The nondeterministic FSM in Figure 5.2 represents an imprecise oracle for a system. It has 11 transitions  $t_1, t_2 \dots, t_{11}$ . Its inputs  $a$  and  $b$  can represent a Boolean assertion over variables used in requirements. The outputs are 0 and 1. Uncertainty is modelled with nondeterministic transitions in states. For example, in state 3, it is uncertain whether the output is 0 or 1 on input  $a$ . The imprecise oracle defines eight plausible precise oracles. Two of them appear in Figure 5.3.

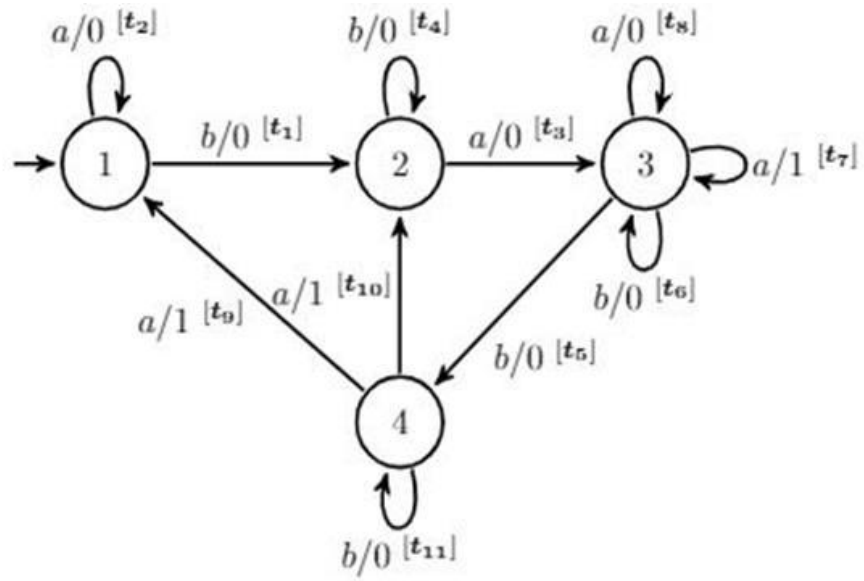
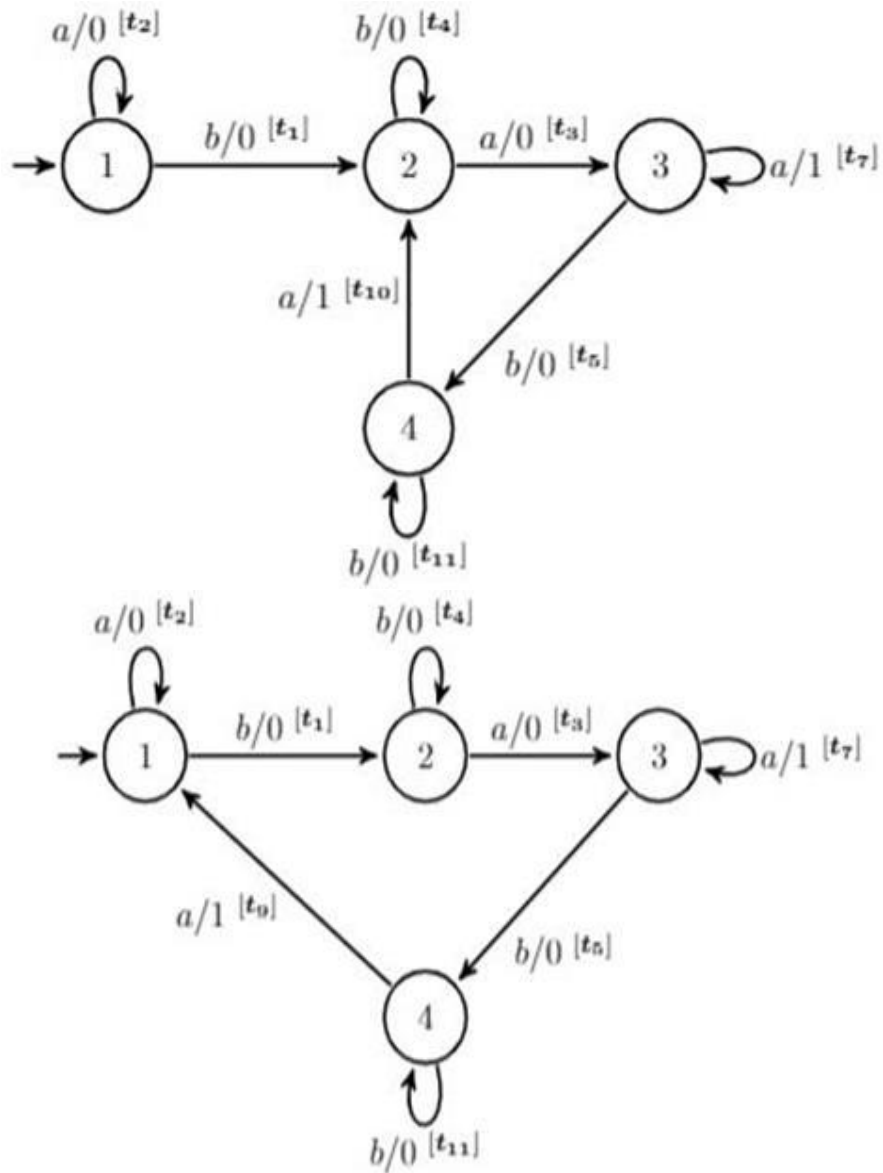


Figure 5.2: An imprecise oracle



**Figure 5.3:** Two plausible oracles

The two plausible precise oracles are indistinguishable with test babaab because both produce output 000100 on the test. For input babaab, the eight precise oracles produce outputs 000100, 000110 and 000000. If the expert judges the output 000100 is expected, the procedure will generate the test babaaa that distinguishes between the two plausible precise oracles in Figure 5.3. The precise oracles produce 000101 and 000100 with the test babaaa. If an expert chooses 000101 as the expected output, then the first precise specification is the expected one. Otherwise, a new test is automatically generated, and the expert is invited to estimate the plausible outputs.

#### 5.4. Claimed novelty

Our two-step approach to generating precise oracle is novel and suitable for evolving systems. We anticipate that it can be easier to automatically generate imprecise oracles without



missing any complex interpretation of the requirements. In addition, the proposed procedure for mining precise is based on the Boolean encoding of the imprecise oracle and constraint resolution. It avoids a one-by-one enumeration of every precise oracle. It proceeds by building partitions of the imprecise oracle, which is also a novelty as compared to our previous work [3].

Ongoing work includes enhancing a prototype tool for the proposed procedure and lifting the procedure to extensions of FSM with variables and complex operations on them. The variables can represent input and output ports of evolving systems on these variables. We are also investigating the ML/NLP based generation of imprecise specifications for requirement documents, especially requirements used by the IVVES industrial partners.

## 6. Automated test verdict generation via Model Learning (F-Secure, OUNL)

In order to achieve automating test verdict generation, first we need to collect data from SUT, which afterwards will be analyzed in order to determine whether the outcome is intended or it is a bug.

The following three main approaches have been used to collect data from SUT:

- Actions done randomly
- Actions done by replaying a prior test sequence
- Actions done using Reinforcement Learning aligned to specific reward strategies.

After the data is collected, it comes down to comparing the results across different executions of the same sequence (model) and determine the differences in a way that will help Software Engineers identify potential issues within the new versions.

For this purpose, F-Secure used two frameworks which were integrated within the existing Test Automation process:

- TESTAR (developed by Open University of Netherlands)
- Change-Analyzer (developed by F-Secure).

### 6.1. Description (Change-Analyzer)

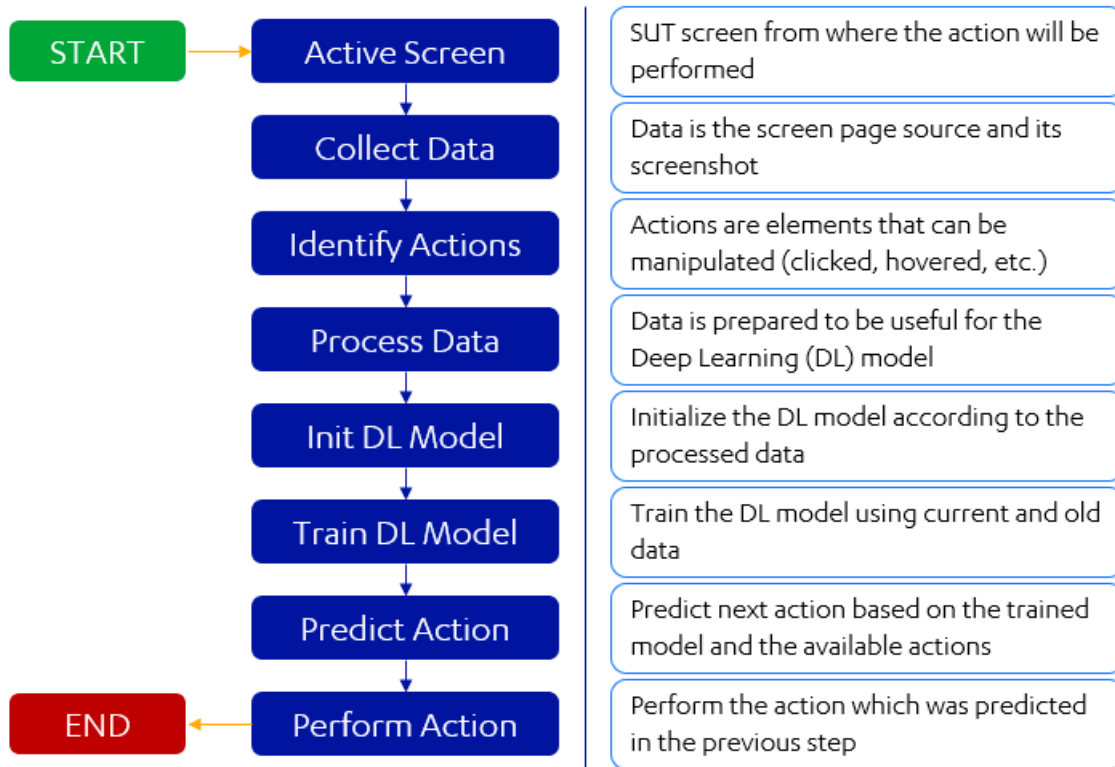
Change-Analyzer (CA) is an open-source framework built utilizing ML techniques, leveraging OpenAI Gym library. CA allows product teams to get feedback regarding their software product, aka SUT (System Under Test) without having any prior knowledge of the SUT.

Essentially, CA is built around the following main Data features:

- Data Collection, done through Automated Exploratory Testing
- Data Reconstruction, done through Automated Regression Testing
- Data Analysis, done through Change Detection
- Data Validation, done through Verdict Generation

### 6.2. Example (Change-Analyzer)

In a simplistic general way, below is the representation of the workflow, using Reinforcement Learning approach to select the next action. The workflow represents a test step from a test sequence. Note that the data will be used later, as basis for test verdict generation.



For every step, the above workflow is repeated until the desired amount of steps is reached, or a blocking action is encountered. When mentioning blocking actions, that can have different meaning, depending on the purpose determined by the reward policy. For instance, we can have test sequences that aim to remain within the designated SUT, and therefore when an action takes the user outside the SUT, we have a blocking action and the test sequence should stop.

Reward policy is a mechanism used to grade the outcome of a performed action. For instance, a naïve approach is to count the amount of changed pixels, using the screenshots from one action to the next. The reward information allow the Reinforcement Learning agent to observe and learn from its actions in order to adapt to the respective reward policy.

When framework is integrated into existing CI/CD pipelines, the generated test sequences can be executed against future versions of SUT. For each execution of the test sequence, the data is recorded in the same way as when the sequence is first discovered.

Once there are two test results of the same test sequence, the output can be compared in order to evaluate the behaviour. The difference is done on two levels:

- On image level: comparing the screenshots
- On source level: comparing the page sources

Based on the sequence differences, a test report is generated. Here each performed step is listed alongside the associated screenshot. Any encountered difference is highlighted to make further evaluation easier.

Here is an example to showcase how the report is currently displayed and how an actual difference is highlighted.

# Change analyzer report

**Expected sequence:** 4bfa83c0-7ddf-11ec-9761-18cc18ca8900 (2022\_01\_25-13\_44\_30)

**Actual sequence:** 2890a6e8-7dd4-11ec-ac3a-18cc18ca8900 (2022\_01\_25-15\_04\_14)

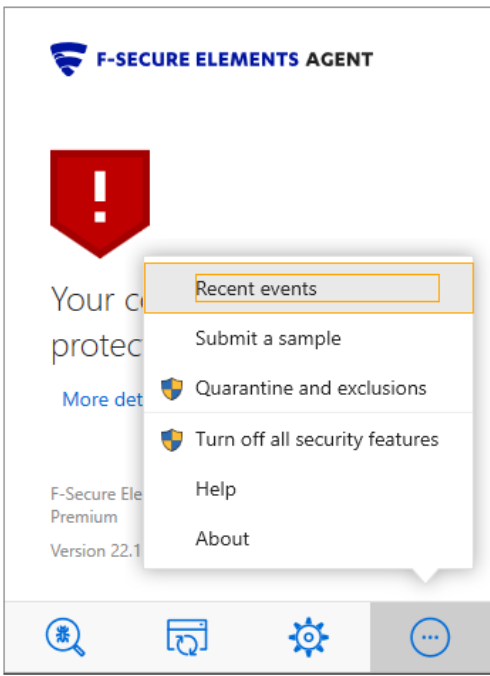
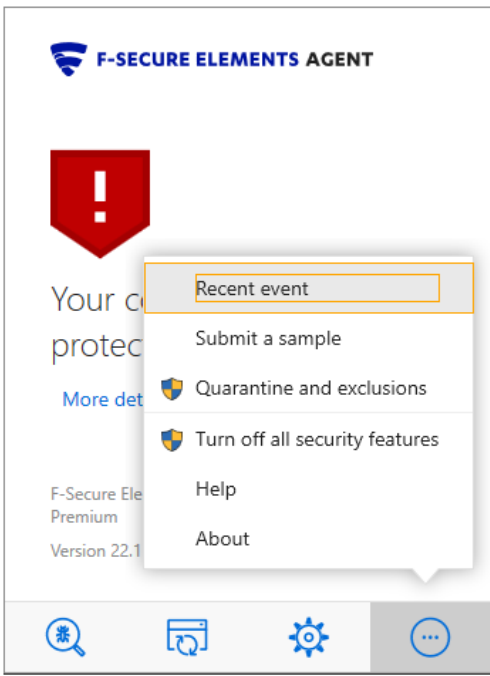
**Date:** 2022\_01\_25-15\_40\_40

Toggle all steps

- Step 1** open the Application
- Step 2** click on Manual scanning
- Step 3** click on Manual scanning
- Step 4** click on More options
- Step 5** click on More options
- Step 6** click on Manual scanning
- Step 7** click on More details
- Step 8** click on Main view

**Step 4** click on More options

**Info** The actual screenshot is not the same as the expected screenshot

Expected screenshot	Actual screenshot
	

As it can be seen, the report presents at first an overview with the performed test steps. Each step can be expanded to see further details considering the test execution. In the above example, the only difference was a missing "s" from "Recent events". The difference is so small that even an experienced

Software Engineer could easily miss it. However, because the difference is visually highlighted, it allows a proper focus on the potential issue.

### 6.3. State of the art

Most of the approaches to test automation and to verdict generation can be summarized as follows:

- Functional requirements are defined (prior knowledge of the SUT)
- Test cases are identified to address the functional requirements
- Test cases are automated and are integrated into a test automation framework
- Test automation framework is executed against different software versions
- Test automation results are compared with a predefined expected output.

However, if an action can be performed by a user, it means that it is a valid action, therefore, it should be tested.

Here is the innovative approach proposed within Change-Analyzer:

- No prior knowledge of the SUT required
- Reinforcement Learning agents explore the SUT
- Data is collected for every performed action
- Performed actions are evaluated based on a given reward policy
- Test sequences are executed against new versions of SUT
- Data collected during test sequences form the model of the SUT
- Data collected during test sequences contains the generated test cases
- Output from different executions is compared to previous results
- Change-Analyzer is integrated into existing test automation framework
- Generated test sequences are executed against different software versions.

### 6.4. Anticipated contributions (Change-Analyzer)

We anticipate several contributions to improve the current state of Change-Analyzer. Below we list main contribution directions:

- Reward mechanism is rather naive
  - Improve reward mechanism to have better address different use cases
- Data used for model training has limited features
  - Create additional features regarding the images used in training the model
- SUT model is based on visual assets such as screenshots and page source
  - Explore expanding SUT model to contain internal state of different components
- Changes are based on UI and page source detection
  - Connect a Reinforcement Learning agent to the source code to map changes from code to changes from UI, and further being able to recommend tests to cover specific changes
- Available actions are limited to clicking
  - Extend the actions to include for example also hovering, right-clicking elements
- The framework is available for Windows and Web applications
  - Extend the framework to address also Mobile applications
- The report is basic
  - Create an actual dashboard for visualization, connecting different analysis, including quality trends

## 6.5. Contributions (Testar)

For functional data collection system data needs to be generated and stored. Testar will be used to generate data against SUT. The best way to achieve this is to integrate Testar into F-Secure's current test automation. This will enable company's standardized delivery of the SUT to target platform and will make Testar execution generic. This means it's possible to extend Testar to be executed also against different software of the company later if needed.

SUT test automation will be executed in the temporary Azure instance which exists only for length of testing. This means that it's not possible to store the data on the same platform. A remote database will be needed to store data. Company's policy declines storing persistent data into the Azure, forcing data to be stored in the AWS. Connection from the Azure to the AWS had to be created and tested. Backups for the AWS database should be enabled in case of data loss or corruption.

Data has to be stored in an organized way so it's available in a reasonable manner. This means planning for different Testar modes and comparing results. Testar doesn't support remote saving for comparing results yet. Therefore, comparing results will be temporarily stored in Jenkins. Later results will be saved in a database when Testar has a feature to support remote saving.

Desired way to execute Testar modes and comparison will be tested and adjusted. Testing in practice will show results and process will be adjusted to correspond with wanted outcome. Testar's comparison tool will be tested and evaluated if it's sufficient in the labeling process. If the comparison tool will turn out to be insufficient, brand new comparing tool should be created. The tool should generate values that are machine readable. Automatable mechanism to detect regression versus improvement in UI should be discovered.

## 7. Conformal prediction for edge applications (Ekkono Solutions)

### 7.1. Introduction

Many machine learning systems involve making predictions, through estimating the value of a dependent variable with a priori unknown ground truth. Verification of such predictive systems, in particular when applied in high-risk applications, is crucial. Traditional machine learning algorithms and means of validation, however, generally lack capabilities for establishing trustworthy verification; e.g., established common-use validation procedures tend to be biased, leading to error frequency on production data not corresponding with error frequency on test data; and, established common-use validation procedures tend to perform verification on a macroscopic level (per-model) rather than a microscopic level (per-prediction), leading to difficulties in verification of individual predictions.

Conformal prediction and in particular the conformal prediction framework presented here offers an alternative method for constructing and evaluating predictive models that is better suited than traditional predictive methods in applications where thorough verification is crucial.

### 7.2. Technical description

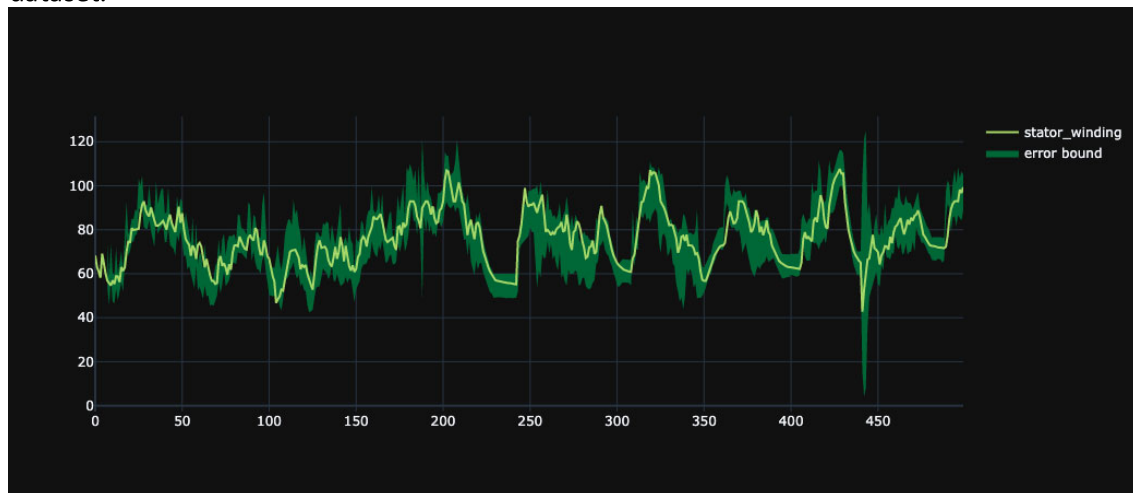
Traditional predictive models output so-called point predictions—a single-valued best-guess prediction for the value of the dependent variable. Conformal predictors, on the other hand, output multi-valued prediction regions that represent a range of likely value assignments for

the dependent variable, constrained by its domain. Any prediction region produced by a conformal predictor comes associated with a very specific, statistically valid, expectation: that the a priori probability of the prediction region containing the ground truth value of the dependent variable is fixed and known. Under these conditions, model and prediction verification becomes straight-forward, as each prediction is guaranteed to contain the correct value of the dependent variable with a user-specified probability.

### 7.3. Example

In this example we create a virtual sensor for the temperature of the stator windings in an electrical motor, from the publicly available electrical motor dataset. The electrical motor data is from a permanent magnet synchronous motor which is typically used in electrical vehicles. Due to the intricate design of the motor, direct measurements of temperature is only possible in a lab setting and not on commercial vehicles. At the same time, precise temperature estimations are getting more and more important in step with the rising relevance of functional safety. The target is best-cost hardware of traction drives in an automotive environment. A machine learning model for this scenario must be small in both terms of memory and CPU. The original dataset<sup>1</sup> had 140 hours (collected at 2Hz) of handcrafted test runs that should simulate typical usage. The data has been down sampled to 1/60 Hz (a sensor reading each minute) since a higher frequency provides little benefit for temperature data. In the resulting dataset we have 8319 instances with eight attributes. The last attribute `stator_winding` is the temperature of the stator winding (see the picture below), as measures in the test bench. This temperature should be predicted using the other variables. Note that `stator_winding` normally shouldn't be used as an input to the model since the sensor would most often be absent at deployment.

The following figure shows an example of the conformal prediction framework applied to such dataset.



### 7.4. State of the art and anticipated contribution

---

<sup>1</sup> Kirchgässner W., Wallscheid O. and Böcker J and described in their publication *Empirical Evaluation of Exponentially Weighted Moving Averages for Simple Linear Thermal Modeling of Permanent Magnet Synchronous Machines* 2019.

In the previous deliverables we have presented an efficient implementation of conformal prediction, able to run on edge applications. The implementation supports online re-calibration of the conformal predictor, to allow for individualized learning, and adapting to change. The provided implementation of conformal prediction is further extended to support fully online training on the edge, without the need to supply any prior training data.

We have improved upon that version by continuing the development and presenting a streamlined execution engine, improved upon the documentation and tutorials, and we have optimized the algorithm for smaller footprint and more energy efficiency.

Finally, we have also been evaluating it in real case scenarios on edge applications.

The novelty of our approach is focused on extending the state-of-the-art research on conformal prediction focusing on incremental learning at the edge. Ekkono's conformal prediction framework is the first commercial application of conformal prediction, in particular for embedded devices and streaming data.

## 8. Code defect risk prediction (Sogeti)

### 8.1. Introduction

In the previous deliverable, we have introduced a method of code analysis and defect prediction using ML. The code defect detection tool aims to assess risk earlier in development and more accurately focus QA activities, making them more efficient. Furthermore, it can be used to improve code maintainability, and make the development team more aware of the code they commit. This will ultimately add value by enhancing the quality of the software while shortening the delivery/release cycle.

For our final approach, we focus on:

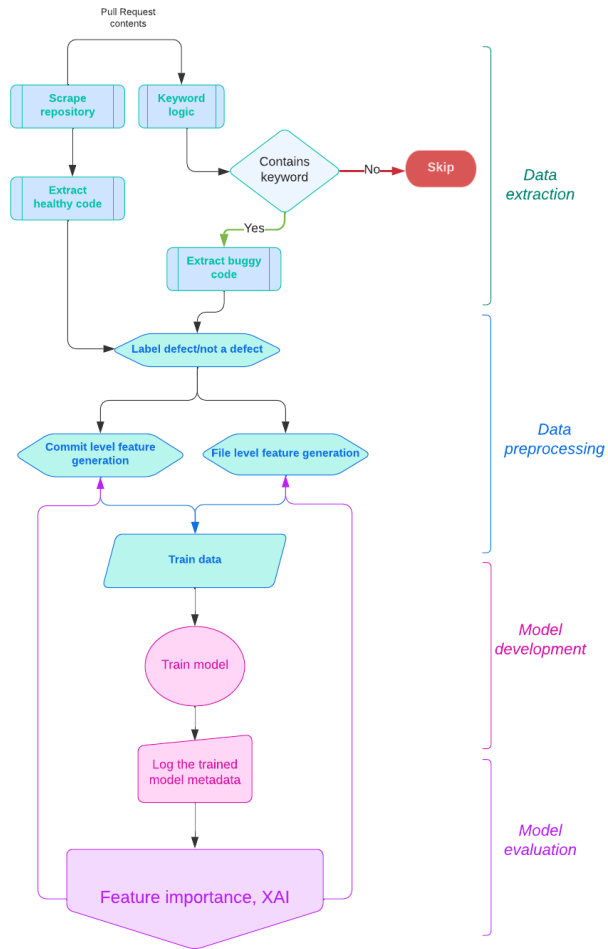
- Providing a list of guidelines for proper version control tracking of buggy code.
- Extending the list of programming languages supported by our tool.
- Calculating feature importance to derive which static code analysis metrics impact the model predictions most.
- Integrating our tool with a WP4 tool, DevAssist.

### 8.2. Anticipated contribution

A final version of the tool with an ML component and XAI layer, with a reporting dashboard. This code quality tool will help to speed up the peer review process by showing developers which parts of their code are risky. The XAI layer will help the developers understand the model output. The model artifacts can be used in DevOps monitoring solutions and improve project management decisions.

### 8.3. Proposed approach

The final version of the code defect risk prediction tool development approach is outlined in Figure 8.1.



**Figure 8.1:** The diagram of the solution approach.

We split the approach per QAIF phase (framework built in WP2 followed to ensure quality of AI solutions).

### Phase 1: Understanding of risks of application

During our work between D3.2. and now, we realized the guidelines for commit messages are extremely important as the data quality is low in case of none being followed. As a final version of this method, we propose a set of guidelines for a defect detection efficient committing of code in a repository, for example:

- The buggy code must be clearly reported on in issues.
- The pull requests must be labelled with adequate labels so the scraping can be optimized.
- Keyword extraction must be specific to use case and repository at hand.

The list of guidelines is evolving and is open to adjustment based on repository at hand.

### Phase 2 & 3: Data Understanding and preparation

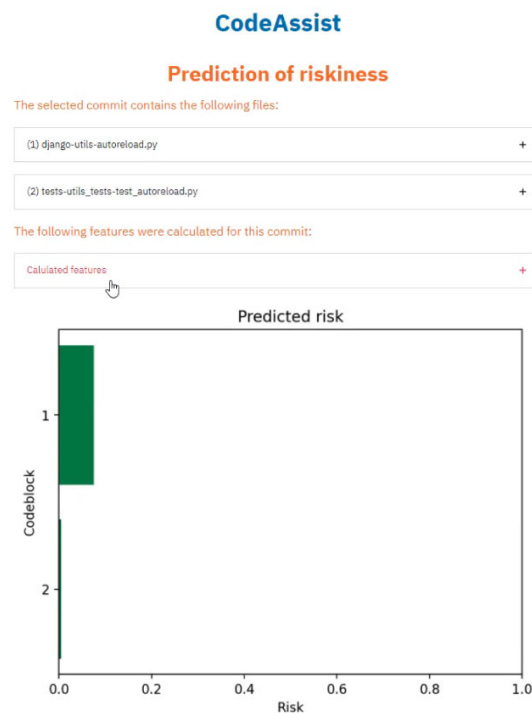


We stand by using static code analysis packages to extract features for model training, therefore, we kept this method in the final version of the tool. Some of the features we derive from scraped code (healthy and buggy) are outlined below:

- Code block level features - complexity, number of distinct operators and operands, single comments, blank lines, etc.
- Metadata features – related to the user and related to the number of files in the commit, added, changed, removed files, etc.

#### Phase 4 & 5: Model development and evaluation

We propose using a Random Forest Classifier as an innovative method to detect the probability of faults and thus determine if the quality of code is risky or not. We use defects as our objective for our machine learning outcome. The inference result can be seen in Figure 8.2.



**Figure 8.2:** The prediction output. The predicted risk is split per committed code block.

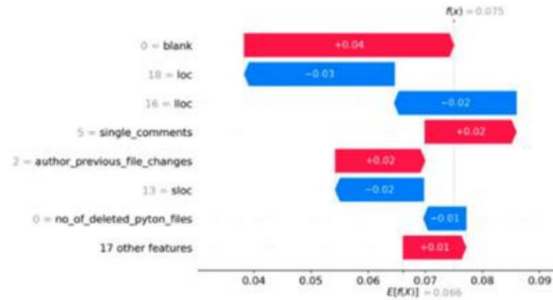
Furthermore, we test the model with XAI methods (SHAP) for:

- Local feature impact.
- Global feature impact (Figure 8.3).

## CodeAssist

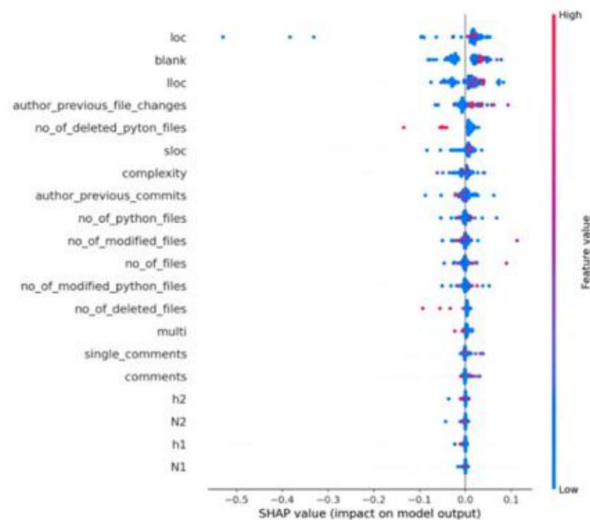
### Prediction explanation

The graph below shows how the different features have impacted the prediction. Starting with the most important features, it shows how each of them contributes to the final score. Blue bars represent a reason the risk is lower than average, while the red bars represent increased risk.



### Feature impacts

The graph below shows the impact that different features have on the predictions that the model makes. For each feature, each dot represents a codechange that has been shown to the model. Different changes can stand out for different reasons. This leads to the spread that can be seen in the graph.



**Figure 8.3:** The top part of the figure is the SHAP plot of local feature impacts, the bottom part shows the global feature impact to the model.

#### 8.4. DevOps pipeline: integration and traceability

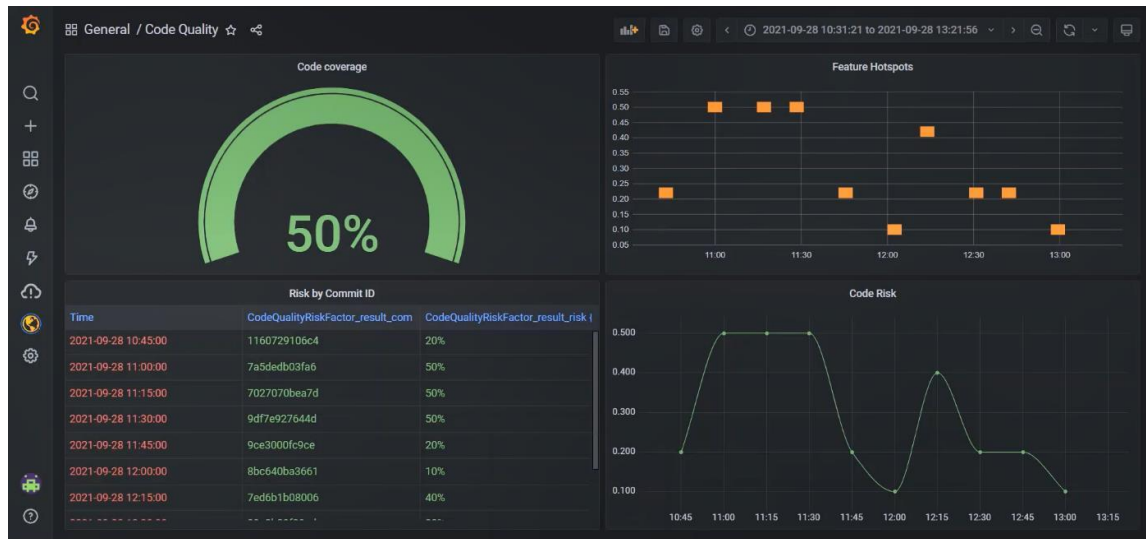
Given the tool Sogeti is working on in WP4, DevAssist – we propose feeding this tool's output into a monitoring dashboard. A representation of this is in Figure 8.4. The widgets in the dashboard include structural and functional metrics like:

- Code coverage.
- Risk by commit ID.
- Feature hotspots.

- Code risk over time.

These widgets provide insight into code commits during a time period, for example an agile sprint. The stakeholders, testers and developers can use it to gain insight into the development process in that time period. This dashboard can answer the following questions related to project management:

- “Which features introduce buggy code?”
- “Did we give enough time for proper development and unit testing during this sprint?”
- “Which commits were flagged as risky?”
- “Has a specific test method reduced code risk over time?”



**Figure 8.4:** A code defect risk prediction report in a monitoring dashboard (Grafana).

Our dashboard of choice is Grafana, a cloud-based time series dashboard commonly used for log analysis and monitoring. Due to Grafana’s extensive plug ins (GitHub and GitLab, for example), there is a myriad of choices regarding data sources and methods for log analysis. Therefore, we propose the usage of this dashboard in the final version to integrate with the relevant data sources of a project to create a unified testing dashboard to provide transparency and monitoring of the development process and drive testing and project management decisions.

### 8.5. Claimed novelty

As industrial systems evolve and grow increasingly complex, we need to find new ways to manage the sustainable growth of development and testing operations. By implementing an automated machine learning code risk model, we can more accurately predict the quality of our code and where the high-risk features are, which will enable the prioritisation of test cases and better organization of projects.

We have developed this tool with ITEA IVVES WP2 methods place to ensure development of validated and quality ML – starting with data and ending with model evaluation. Our tool not

only has a smart component, but also has an explainable layer which helps the users (developers, testers, stakeholders) understand why a certain commit is predicted as risky. This helps align teams and bring transparency to the solution. Finally, the report dashboard is used to drive decisions in testing and project management. Our proposed technical solution is a novelty in the software industry and will shift the way of working towards compliant & quality development.

## 9. References

- [1] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo, "The Oracle Problem in Software Testing: A Survey" *IEEE Transactions on Software Engineering*, 2015.
- [2] I. Buzhinsky, "Formalization of natural language requirements into temporal logics: a survey," 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), Helsinki, Finland, pp. 400-406, 2019.
- [3] O. Nguena Timo, Alexandre Petrenko, S. Ramesh, "Using Imprecise Test Oracles Modelled by FSM". *ICST Workshops*, pp. 32-39, 2019.
- [4] J. Galvani Gregghi, E. Martins, A. Maria Brito, R. Carvalho, "Semi-automatic Generation of Extended Finite State Machines from Natural Language Standard Documents", *DSN Workshops*, pp. 45-50, 2015.
- [5] F. Pudlitz, F. Brokhausen, A. Vogelsang, "Extraction of System States from Natural Language Requirements" *RE* 211-222, 2019
- [6] C. Gustavo, S. Augusto, "Formal Specification Generation from Requirement Documents", *Electronic Notes in Theoretical Computer Science*. 195. 171-188, 2008.
- [7] S. Bauersfeld, T. Vos, "A reinforcement learning approach to automated gui robustness testing", In *Fast Abstracts of the 4th Symposium on Search-Based Software Engineering (SSBSE)*, IEEE, pp. 7–12, 2012.
- [8] A. Esparcia-Alcázar, F. Almenar, T. Vos, U. Rueda, "Using genetic programming to evolve action selection rules in traversal-based automated software testing: results obtained with the TESTAR tool", *Memetic Computing* 10(3): 257-265, 2018.
- [9] Y. Miao and X. Yang, "An FSM based GUI Test Automation Model", the 11th International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 120-126, 2010.
- [10] P. Aho, N. Menz, T. Rätty and I. Schieferdecker, "Automated Java GUI Modeling for Model-Based Testing Purposes," In *Eighth International Conference on Information Technology: New Generations*, pp. 268-273, 2011.
- [11] A. Mesbah, A. van Deursen, and S. Lensenlink, "Crawling Ajax-Based Web Applications through Dynamic Analysis of User Interface State Changes", *ACM Trans. Web* 6, 1, Article 3 (March 2012), 30 pages. DOI:<https://doi.org/10.1145/2109205.2109208>

- [12] P. Aho, M. Suarez, T. Kanstrén and A. M. Memon, "Murphy Tools: Utilizing Extracted GUI Models for Industrial Software Testing," Seventh IEEE International Conference on Software Testing, Verification and Validation Workshops, pp. 343-348, 2014.
- [13] V. Cortellessa, A. Di Marco, P. Inverardi, "Model-based software performance analysis", Springer Science & Business Media, 2011.
- [14] M. Harchol-Balter, "Performance modeling and design of computer systems: queueing theory in action", Cambridge University Press, 2013.
- [15] K. Kant, M. M. Srinivasan, "Introduction to computer system performance evaluation", McGraw-Hill College, 1992.
- [16] S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni, "Model-based performance prediction in software development: A survey", IEEE Transactions on Software Engineering, 295-310, 2004
- [17] H. Koziolok, "Performance evaluation of component-based software systems: A survey. Performance evaluation", pp. 634-658, 2010.
- [18] P. Zhang, S. Elbaum, M. B. Dwyer, "Compositional load test generation for software pipelines", In Proceedings of the International Symposium on Software Testing and Analysis pp. 89-99, 2012.
- [19] V. Garousi, "A genetic algorithm-based stress test requirements generator tool and its empirical evaluation". IEEE Transactions on Software Engineering, 36(6), 778-797, 2010
- [20] M. B. da Silveira, E. D. M. Rodrigues, A. F. Zorzo, L. T. Costa, H. V. Vieira, F. M. de Oliveira, "Generation of Scripts for Performance Testing Based on UML Models", In SEKE, pp. 258-263, 2011.
- [21] C. Lutteroth, G. Weber, "Modeling a realistic workload for performance testing". In 12th International IEEE Enterprise Distributed Object Computing Conference, pp. 149-158, 2008.
- [22] H. Schulz, D. Okanović, A. van Hoorn, V. Ferme, C. Pautasso, "Behavior-driven load testing using contextual knowledge-approach and experiences", In Proceedings of the ACM/SPEC International Conference on Performance Engineering, pp. 265-272, 2019.
- [23] V. Ferme, C. Pautasso, "A declarative approach for performance tests execution in continuous software development environments", In Proceedings of the ACM/SPEC International Conference on Performance Engineering, pp. 261-272, 2018.
- [24] R. S. Sutton, A. G. Barto, "Reinforcement learning: An introduction", MIT press, 2018.
- [25] H. M. Moghadam, "Machine Learning-Assisted Performance Assurance", Licentiate Thesis, Mälardalen University, 2020

[26] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, B. Lisper, "Poster: Performance Testing Driven by Reinforcement Learning", In IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), pp. 402-405, IEEE, 2020

[27] M. H. Moghadam, "Machine learning-assisted performance testing". In Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1187-1189, 2019.

[28] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, B. Lisper, "An Autonomous Performance Testing Framework using Self-Adaptive Fuzzy Reinforcement Learning", Software quality journal, 1-33, 2021

[29] M. H. Moghadam, G. Hamidi, M. Borh, M. Saadatmand, M. Bohlin, B. Lisper, B., & P. Potena, "Performance Testing Using a Smart Reinforcement Learning-Driven Test Agent", In 2021 IEEE Congress on Evolutionary Computation (CEC) (pp. 2385-2394). IEEE. 2021

-----  
[13] S. Pimont and J. Rault, "A software reliability assessment based on a structural and behavioral analysis of programs", In Proc. of the 2nd international conference on Software engineering (ICSE). IEEE, pp. 486-491, 1976.

[14] T. Chow, "Testing Software Design Modeled by Finite-State Machines", IEEE Trans. on sw. eng., vol. SE-4, no. 3, 1978.

[15] V. Cortellessa, A. Di Marco, P. Inverardi, "Model-based software performance analysis", Springer Science & Business Media, 2011.

[16] M. Harchol-Balter, "Performance modeling and design of computer systems: queueing theory in action", Cambridge University Press, 2013.

[17] K. Kant, M. M. Srinivasan, "Introduction to computer system performance evaluation", McGraw-Hill College, 1992.

[18] S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni, "Model-based performance prediction in software development: A survey", IEEE Transactions on Software Engineering, 295-310, 2004.

[19] H. Koziolk, "Performance evaluation of component-based software systems: A survey. Performance evaluation", pp. 634-658, 2010.

[20] P. Zhang, S. Elbaum, M. B. Dwyer, "Compositional load test generation for software pipelines", In Proceedings of the International Symposium on Software Testing and Analysis pp. 89-99, 2012.

[21] V. Garousi, "A genetic algorithm-based stress test requirements generator tool and its empirical evaluation". IEEE Transactions on Software Engineering, 36(6), 778-797, 2010.

- [22] M. B. da Silveira, E. D. M. Rodrigues, A. F. Zorzo, L. T. Costa, H. V. Vieira, F. M. de Oliveira, "Generation of Scripts for Performance Testing Based on UML Models", In SEKE, pp. 258-263, 2011.
- [23] C. Lutteroth, G. Weber, "Modeling a realistic workload for performance testing". In 12th International IEEE Enterprise Distributed Object Computing Conference, pp. 149-158, 2008.
- [24] H. Schulz, D. Okanović, A. van Hoorn, V. Ferme, C. Pautasso, "Behavior-driven load testing using contextual knowledge-approach and experiences", In Proceedings of the ACM/SPEC International Conference on Performance Engineering, pp. 265-272, 2019.
- [25] V. Ferme, C. Pautasso, "A declarative approach for performance tests execution in continuous software development environments", In Proceedings of the ACM/SPEC International Conference on Performance Engineering, pp. 261-272, 2018.
- [26] B. Settles, "Active learning literature survey", University of Wisconsin-Madison Department of Computer Sciences, 2009.
- [27] R. S. Sutton, A. G. Barto, "Reinforcement learning: An introduction", MIT press, 2018.
- [28] H. M. Moghadam, "Machine Learning-Assisted Performance Assurance", Licentiate Thesis, Mälardalen University, 2020.
- [29] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, B. Lisper, "Poster: Performance Testing Driven by Reinforcement Learning", In IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), pp. 402-405, IEEE, 2020,
- [30] M. H. Moghadam, "Machine learning-assisted performance testing". In Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1187-1189, 2019.
- [31] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, B. Lisper, "An Autonomous Performance Testing Framework using Self-Adaptive Fuzzy Reinforcement Learning", arXiv preprint arXiv:1908.06900, 2019.
- [32] G. Hamidi, "Reinforcement Learning Assisted Load Test Generation for E-Commerce Applications", Master thesis, Mälardalen University, 2020.
- [33] J. O'Duinn, The financial cost of a check in (2013). Accessed 2020-08-11.
- [34] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, pp. 426-437, 2016.
- [35] G. Rothermel and M. J. Harrold, "Analyzing regression test selection techniques" IEEE Transactions on software engineering, vol. 22, no. 8, pp. 529-551, 1996.

- [36] B. G. Ryder, F. Tip, "Change impact analysis for object-oriented programs," in Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, pp. 46–53, 2001.
- [37] M. Gligoric, L. Eloussi, D. Marinov, "Ekstazi: Lightweight test selection," in 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 2. IEEE, pp. 713–716, 2015.
- [38] T. A. Budd, D. Angluin, "Two notions of correctness and their relation to testing", *Acta Informatica* 18(1), pp. 31–45, 1982.
- [39] E. J. Weyuker, "Assessing test data adequacy through program inference. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 5(4), pp. 641–655, 1983.
- [40] R.D. King, K.E. Whelan, F.M. Jones, P.G. Reiser, C.H. Bryant, S.H. Muggleton, D.B. Kell, S.G. Oliver, "Functional genomic hypothesis generation and experimentation by a robot scientist", *Nature* 427(6971): pp. 247–252, 2004.
- [41] J. Henkel, A. Diwan, "Discovering algebraic specifications from java classes", In: *European Conference on Object-Oriented Programming*, Springer, pp. 431–456, 2003. <sup>[1]</sup><sub>SEP</sub>
- [42] P. Papadopoulos, N. Walkinshaw, "Black-box test generation from inferred models", In: *Proceedings of the Fourth International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, IEEE Press, pp. 19–24, 2015
- [43] L. C. Briand, Y. Labiche, Z. Bawar, N. T. Spido, "Using machine learning to refine category-partition test specifications and test suites", *Information and Software Technology* 51(11): pp. 1551–1564, 2009.
- [44] V. Vovk, A. Gammerman, G. Shafer, "Algorithmic learning in a random world", Springer Science & Business Media; 2005.
- [45] G. Shafer, V. Vovk, "A tutorial on conformal prediction", *Journal of Machine Learning Research*, vol 9, pp. 371-421, 2008.
- [46] C. Saunders, A. Gammerman, V. Vovk, "Transduction with confidence and credibility", pp. 712-726, 1999.
- [47] V. Balasubramanian, S.S. Ho, V. Vovk, editors. "Conformal prediction for reliable machine learning: theory, adaptations and applications", Newnes, 2014.
- [48] U. Johansson, H. Boström, T. Löfström, H. Linusson, "Regression conformal prediction with random forests", *Machine Learning*, vol 97(1-2), pp. 155-76, 2014.
- [49] H. Linusson, U. Norinder, H. Boström, U. Johansson, T. Löfström, "On the calibration of aggregated conformal predictors", In *Conformal and probabilistic prediction and applications*, pp. 154-173, 2017.



- [50] J. Alvarsson, S.A. McShane, U. Norinder, O. Spjuth, "Predicting with confidence: Using conformal prediction in drug discovery", *Journal of Pharmaceutical Sciences*, 2020.
- [51] N. Bosc, F. Atkinson, E. Felix, A. Gaulton, A. Hersey, A.R. Leach, "Large scale comparison of QSAR and conformal prediction methods and their applications in drug discovery", *Journal of cheminformatics*, vol 11(1), p. 4, 2019.
- [52] M. Eklund, U. Norinder, S. Boyer, L. Carlsson, "The application of conformal prediction to the drug discovery process", *Annals of Mathematics and Artificial Intelligence*, vol 74(1-2), pp. 117-32, 2015.
- [53] M. Pashkovskiy, et. al, "State of the art of validation methods and techniques for complex evolving systems," ITEA, 30-Jun-2020. [Online]. Available: <https://itea3.org/project/ivves.html>. [Accessed: 2020].
- [54] R. Bellairs, "What Is Code Quality? And How to Improve Code Quality," Perforce Software, 2019. [Online]. Available: <https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-code-quality>. [Accessed: 01-Dec-2020].
- [55] A. VanTol. "Python Code Quality: Tools & Best Practices." Real Python, Real Python, 7 Nov. 2020, [realpython.com/python-code-quality/](https://realpython.com/python-code-quality/).
- [56] I. S. Cordasco, flake8 Documentation. (2020) [online] Available at: <<https://flake8.pycqa.org/downloads/en/latest/pdf/>> [Accessed 23 November 2020].
- [57] GitHub. 2020. DmytroLitvinov/Awesome-Flake8-Extensions. [online] Available at: <<https://github.com/DmytroLitvinov/awesome-flake8-extensions>> [Accessed 23 November 2020].
- [58] Radon.readthedocs.io. 2020. Welcome To Radon'S Documentation! — Radon 4.1.0 Documentation. [online] Available at: <<https://radon.readthedocs.io/en/latest/>> [Accessed 23 November 2020].
- [59] Bandit.readthedocs.io. 2020. Welcome To Bandit'S Developer Documentation! — Bandit Documentation. [online] Available at: <<https://bandit.readthedocs.io/en/latest/>> [Accessed 23 November 2020].
- [60] Coverage.readthedocs.io. 2020. Coverage.Py — Coverage.Py 5.3 Documentation. [online] Available at: <<https://coverage.readthedocs.io/en/coverage-5.3/>> [Accessed 23 November 2020].
- [61] M. H. Halstead, "Elements of Software Science", Elsevier, vol 7, 1977.
- [62] K. Pijanowski, "Improve Code Quality Using Test Coverage." CODE Magazine, [www.codemag.com/article/1701081/Improve-Code-Quality-Using-Test-Coverage](http://www.codemag.com/article/1701081/Improve-Code-Quality-Using-Test-Coverage).

- [63] Docs.python.org. 2020. Py\_Compile — Compile Python Source Files — Python 3.9.0 Documentation. [online] Available at: <[https://docs.python.org/3/library/py\\_compile.html](https://docs.python.org/3/library/py_compile.html)> [Accessed 23 November 2020].
- [64] “GitLab CI,” GitLab. [Online]. Available: <https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/>. [Accessed: 07-Dec-2020].
- [65] Jenkins User Documentation. [Online]. Available: <https://www.jenkins.io/doc/>. [Accessed: 07-Dec-2020].
- [66] Wily. [Online]. Available: <https://wily.readthedocs.io/en/latest/>. [Accessed: 07-Dec-2020].
- [67] M. Madera, R. Tomoń, "A case study on machine learning model for code review expert system in software engineering", In Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 1357-1363, 2017.
- [68] P. Deep Singh, A. Chug, "Software defect prediction analysis using machine learning algorithms", In 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence, pp. 775-781, 2017.
- [69] V. Barstad, M. Goodwin, T. Gjørseter, “Predicting Source Code Quality with Static Analysis and Machine Learning”, In Norsk IKT-konferanse for forskning og utdanning. 2014.
- [70] “Interpreting random forests,” Diving into data, 19-Oct-2014. [Online]. Available: <http://blog.datadive.net/interpreting-random-forests/>. [Accessed: 01-Dec-2020].
- [71] GitHub. 2020. Andosa. [online] Available at <<https://github.com/andosa/treeinterpreter>>
- [72] 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, E. Duchesnay, “Scikit-learn: Machine Learning in Python”, JMLR 12, pp. 2825-2830, 2011.
- [73] [E. Klevak](#), [S. Lin](#), [A. Martin](#), [O. Linda](#), [E. Ringger](#), “Out-Of-Bag Anomaly Detection”, arXiv preprint arXiv:2009.09358, 2020.
- [74] K. Zhang, X. Kang, S. Li, “Isolation Forest for Anomaly Detection in Hyperspectral Images” In IEEE International Geoscience and Remote Sensing Symposium \*(IGARSS), IEEE, pp. 437-440, 2019.
- [75] [G. Staerman](#), [P. Mozharovskyi](#), [S. Cléménçon](#), [F. d'Alché-Buc](#), “Functional Isolation Forest”, arXiv preprint arXiv:1904.04573, 2019.
- [76] F. T. Liu, K. M. Ting, Z.-H. Zhou, “Isolation forest”, In Eighth IEEE International Conference on Data Mining, IEEE, pp. 413-422, 2008.
- [77] V. Vercruyssen, “Designing Anomaly Detection Algorithms that Exploit Flexible Supervision”, PhD Thesis, 2020.

- [78] [M. Tulio Ribeiro](#), [T. Wu](#), [C. Guestrin](#), [S. Singh](#), “Beyond Accuracy: Behavioral Testing of NLP Models with CheckList”. arXiv preprint arXiv:2005.04118, 2020.
- [79] [A. Chan](#), [L. Ma](#), [F. Juefei-Xu](#), [X. Xie](#), [Y. Liu](#), [Y. S. Ong](#), “Metamorphic relation based adversarial attacks on differentiable neural computer”. arXiv preprint arXiv:1809.02444, 2018.
- [80] WU, Zhaolin, et al., “A Time Window based Reinforcement Learning Reward for Test Case Prioritization in Continuous Integration”, In Proceedings of the 11th Asia-Pacific Symposium on Internetware, p. 1-6, 2019.
- [81] NASA. [Online]. Available: [https://atmos.nmsu.edu/data\\_and\\_services/atmospheres\\_data/INSIGHT/insight.html](https://atmos.nmsu.edu/data_and_services/atmospheres_data/INSIGHT/insight.html). [Accessed: 15-Dec-2020].
- [82] H. Linusson, U. Johansson, T L fstr m, “Signed-error conformal regression”, In Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 224-236, Springer, 2014.
- [83] [M. Tulio Ribeiro](#), [T. Wu](#), [C. Guestrin](#), [S. Singh](#), “Beyond Accuracy: Behavioral Testing of NLP Models with CheckList”. arXiv preprint arXiv:2005.04118, 2020.
- [84] WU, Zhaolin, et al., “A Time Window based Reinforcement Learning Reward for Test Case Prioritization in Continuous Integration”, In Proceedings of the 11th Asia-Pacific Symposium on Internetware, p. 1-6, 2019.
- [85] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, , and M. Zinkevich, “Data validation for machine learning,” IEEE Transactions on Software Engineering., In SysML, 2019.
- [86] CARLETTI, Mattia, et al. Explainable machine learning in industry 4.0: Evaluating feature importance in anomaly detection to enable root cause analysis. En 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC). IEEE, 2019. p. 21-26.
- [87] Anastasi, S., Madonna, M., & Monica, L. (2021). Implications of embedded artificial intelligence-machine learning on safety of machinery. *Procedia Computer Science*, 180, 338-343.
- [88] Lu, Y. J., & Li, C. T. (2020). GCAN: Graph-aware co-attention networks for explainable fake news detection on social media. arXiv preprint arXiv:2004.11648.
- [89] Pan, J. Z., Pavlova, S., Li, C., Li, N., Li, Y., & Liu, J. (2018, October). Content based fake news detection using knowledge graphs. In *International semantic web conference* (pp. 669-683). Springer, Cham.
- [90] Popat, K., Mukherjee, S., Yates, A., & Weikum, G. (2018). Declare: Debunking fake news and false claims using evidence-aware deep learning. arXiv preprint arXiv:1809.06416.
- [91] Bagherzadeh, M., Kahani, N., & Briand, L. (2021). Reinforcement learning for test case prioritization. *IEEE Transactions on Software Engineering*.

- [93] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016, August). " Why should i trust you?" Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1135-1144).
- [94] Koncel-Kedziorski, R., Bekal, D., Luan, Y., Lapata, M., & Hajishirzi, H. (2019). Text generation from knowledge graphs with graph transformers. arXiv preprint arXiv:1904.02342.
- [95] Liu, Y., Yang, T., You, Z., Fan, W., & Yu, P. S. (2020). Commonsense Evidence Generation and Injection in Reading Comprehension. arXiv preprint arXiv:2005.05240.
- [96] Anastasi, S., Madonna, M., & Monica, L. (2021). Implications of embedded artificial intelligence-machine learning on safety of machinery. *Procedia Computer Science*, 180, 338-343.
- [97] E. Breck, N. Polyzotis, S. Roy, S. E, Whang, , and M. Zinkevich, "Data validation for machine learning," *IEEE Transactions on Software Engineering.*, In SysML, 2019.
- [98] Jonathan Bell, Owolabi Legunsen, Michael Hilton, Lamyaa Eloussi, Tiffany Yung, and Darko Marinov. 2018. DeFlaker: Automatically Detecting Flaky Tests. In Proceedings of ICSE '18: 40th International Conference on Software Engineering, Gothenburg, Sweden, May 27-June 3, 2018 (ICSE '18).
- [99] Alex Groce, Josie Holmes. 2020. Practical Automatic Lightweight Nondeterminism and Flaky Test Detection and Debugging for Python. 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS.)
- [100] Wing Lam, August Shi, Reed Oei, Sai Zhang, Michael D. Ernst, and Tao Xie. 2020. Dependent-Test-Aware Regression Testing Techniques. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '20), July 18–22, 2020
- [101] Wing Lam, Stefan Winter, Angello Astorga, Victoria Stodden, Darko Marinov. Understanding Reproducibility and Characteristics of Flaky Tests Through Test Reruns in Java Projects. 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE).
- [102] Gustavo Pinto, Breno Miranda, Supun Dissanayake, Marcelo d'Amorim, Christoph Treude, and Antonia Bertolino. What is the Vocabulary of Flaky Tests? MSR'20: Proceedings of the 17th International Conference on Mining Software Repositories, June 2020.