IVVES

D5.4 IVVES online experimentation and training platform

ID	Description	Due Date	Status
D5.4	The IVVES online experimentation and training platform	M30	Online

WP Leader: Tanja Vos / Pekka Aho (OUNL)

// WP 5: Content

- Earlier WP5 deliverables related to the platform
  - D5.2 Architecture of the IVVES online experimentation and training platform
- The online platform
  - Guest access
  - Some examples of the content (the full content will be part of D5.6)

// Earlier WP5 deliverables related to the platform

ID	Description	Due Date	Status
D5.1	Requirements for the IVVES online experimentation and training platform	M12	Submitted
D5.2	Architecture of the IVVES online experimentation and training platform	M18	Submitted

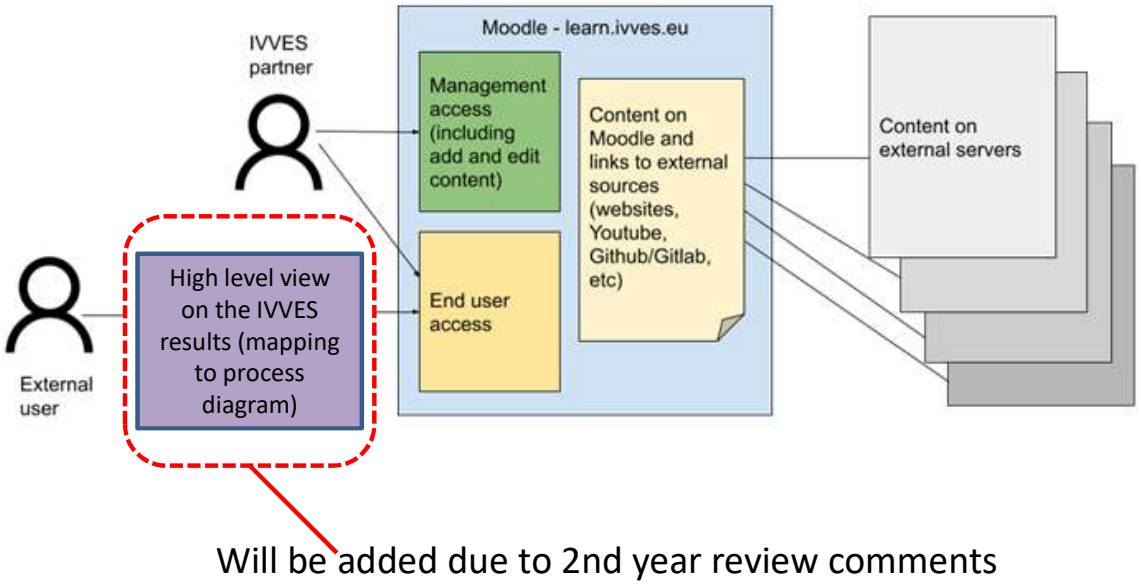
- **Task T5.1 Description**

Defining the requirements and setting up an experimentation framework for integrating the techniques and tools developed in WP2, WP3, and WP4 into the IVVES platform for learning and trying out the tools. Defining the conceptual integration for experimentation and learning and interfaces for generalization and interoperability, concrete integration of the methods and tools, and forming tool chains to combine the methods. This includes an online repository for storing public data, experiences, and tools produced during IVVES through e.g. Github.

- Lead: Open University (NLD)

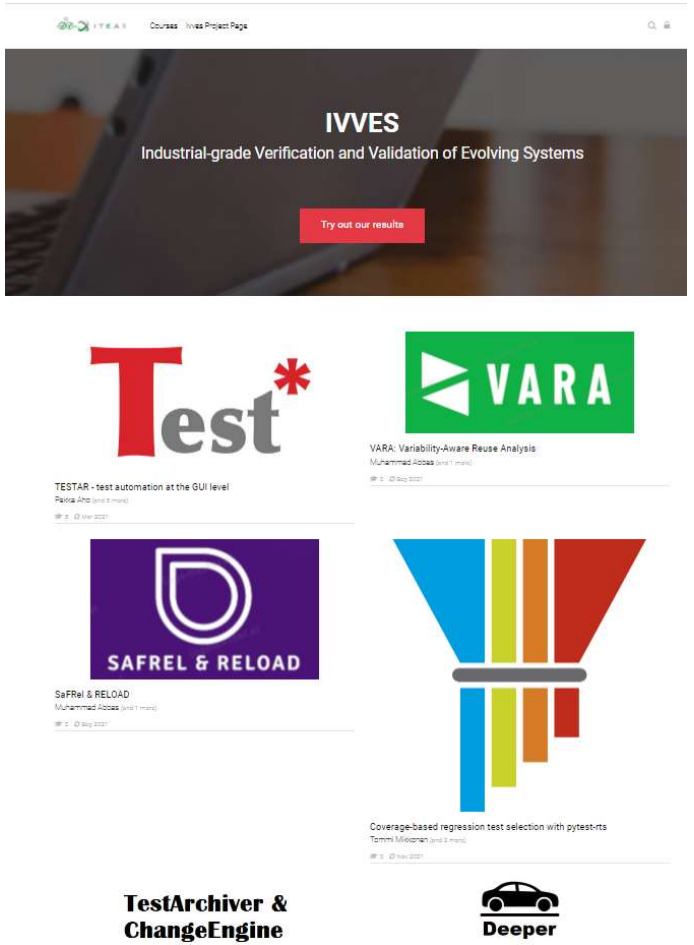
IVVES framework

- an online platform that acts as a front-end for the public IVVES artefacts to the people outside of the consortium
- open source Moodle was selected to be used as a platform



// The online platform

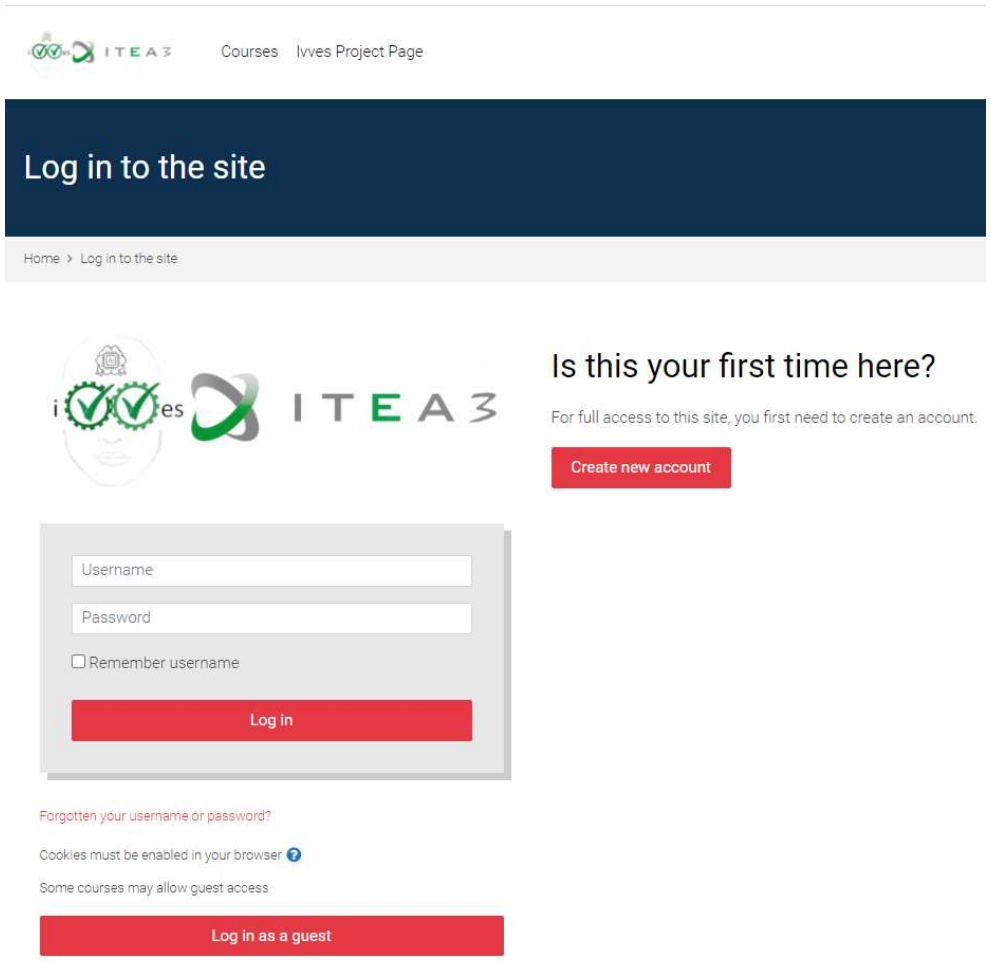
https://learn.ivves.eu/



// The online platform – guest access

Guest access allowed

- *The end users do not necessarily need to register an account*
- *”Log in as a guest”*



// The online platform – some examples of the content

TESTAR

- *The content of the TESTAR online course has been used also for the Software verification and testing course for OUNL students*

TESTAR - test automation at the GUI level

Home > Courses > IVVES tools and techniques > TESTAR

Introduction

GUI Testing is a testing technique where one tests the System Under Test (SUT) solely through its graphical user interface. The way to find errors is to thoroughly observe the status of the GUI after each step of the test. In practice, this type of testing is often carried out manually, a tester following a predefined test case document and verifying whether the application responds to all inputs as expected.

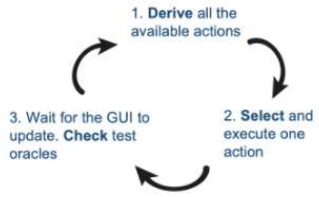
To automate GUI testing approaches there are basically two approaches:

- **script-based** test automation, where test cases or test sequences are defined prior to test execution. That means the test cases are static in the sense that they are not adapted dynamically during the test execution.
- **scriptless** test automation, where the test cases or sequences are generated dynamically, one step at a time, during the test execution, based on the observed state of the system under test.

In this course we concentrate on scriptless test automation in general and on the TESTAR tool specifically.

TESTAR is an open source tool for scriptless test automation through graphical user interface (GUI). TESTAR currently supports the various AI / ML based features for active learning of state models during automated exploration of the GUI of the system under test (SUT) and reinforcement learning based action selection strategies.

TESTAR at the highest level performs 3 steps in a continuous cycle.



In this course we will explain each of these steps in details and show you how you can add more intelligence to each step.

// The online platform – some examples of the content

Pytest-rts

- *Tool for coverage-based regression test selection*

pytest-rts tool

The tool and its source code is available at our git repository https://github.com/F-Secure/pytest-rts.

Guidance how to install it with pip can be found here https://pypi.org/project/pytest-rts/2.1.0.

Pytest-rts currently supports RTS based on changes in the current Git working directory, as well as committed changes. A simple test prioritization algorithm is also implemented by storing the run times of each test case and sorting the queried tests based on this information. The tests with the shortest run time run first to attempt finding a fault as fast as possible.

The tool uses the following steps:

1. After the initial full test suite run, the tool is ready to be used. On the developer's machine, when changes are made to the target project's files, the tool checks for changes in the Git working directory. The tool first constructs a list of changed files according to Git and checks which of those files are either source code files or test code files in our local database.
2. After the tool has determined which files are taken into consideration, it checks the Git diff output for each of those files. From this 'diff', the tool can determine which lines have changed and which lines have shifted from their original position.
3. Then the tool can query all the test functions from our database according to the list of line numbers for the changed lines and run them with Pytest. No database state updating is performed during this. If a user wishes to make these changes final, a Git commit operation is required.
4. When the changes are committed and obtained by the CI server, an agent on the server runs the tool and checks whether the current Git HEAD hash differs from the one that is marked as a last update hash in the mapping database. If so, the tool searches for relevant tests, queries the changes and tests for those changes and checks for newly added test functions by checking what test functions Pytest can find and comparing it to the current state in the database. The tool also calculates how unchanged lines have shifted in the files and performs a database update based on this information.
5. When the tests are run after this, new coverage data is collected and inserted into the database.

Getting started with pytest-rts

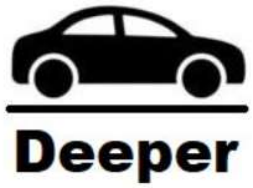
A tutorial is available at https://github.com/F-Secure/pytest-rts/blob/master/docs/tutorial.md

It shows how a developer can use the tool in an open-source project. The steps include (i) initializing the tool, (ii) making changes in the Git working directory, and (iii) committing the changes. The operation that handles committed changes highlights how the tool could work in a CI environment. By initializing the tool in the master branch and then checking out a feature branch results in a scenario where the tool compares the current feature branch Git HEAD to the one in master. The resulting test set from this comparison could be used in pull requests to determine which tests are required to test the actual changes.

// The online platform – some examples of the content

Deeper

- *Tool for simulation-based ADAS testing*



Deeper

Deeper is a simulation-based test generator that uses an evolutionary process for generating critical system-level test cases to test a deep neural network-based lane-keeping system. It is a tailor-made system level testing tool for an automotive use case and has been integrated into a relevant simulator in that domain, i.e., BeamNG simulator. The current version of the tool uses a multi-objective search based on NSGA-II. Deeper augments the NSGA-II with a simple greedy archive to store the candidate solutions that make the car go out of the lane regarding a certain tolerance threshold. This threshold defines at least how much (e.g., 50%) of the car must get out of the lane for the test case to be considered as a failure. Deeper uses a quality population seed to boost the search process with respect to the fixed test budget. It uses Catmull-Rom cubic splines to represent the roads and relies on mutation and re-population evolution operators implemented by DeepJanus testing tool (https://github.com/testingautomated-usi/DeepJanus/tree/master/DeepJanus-BNG).

Input to the tool: The SUT (i.e., ML component) connected to the simulation environment + a set of initial test scenarios

Output of the tool: system-level test cases executable in the simulator (in form of JSON files)

Source or Binary Link: https://github.com/mahshidhelali/Deeper_ADAS_Test_Generator

Instruction manual or tutorial for the tool: https://github.com/mahshidhelali/Deeper_ADAS_Test_Generator

Type: Open source

Contact person: mahshid.helali.moghadam@ri.se