WHITE PAPER

AUTONOMOUS USER INTERFACE (UI) TESTING USING SELF-HEALING ARTIFICIAL INTELLIGENCE
This white paper explains how User Interface (UI) Testing based on Deep Learning/Artificial Intelligence surpasses conventional automated testing for UI Applications. Employing script-less automation provides maximum benefits with minimal efforts. Furthermore, the ever-evolving Machine Learning and Artificial Intelligence (ML/AI) capabilities, combined with statistical and analytical insights, makes the solution far superior to others, thereby overcoming the limitations of conventional testing.

Opportunities with Autonomous UI Testing solution

Transformation progression to autonomous UI testing
The Autonomous UI Testing solution is a script-less, Framework-agnostic solution; no coding experience is needed prior to executing and leveraging the solution. Very minimal user input is needed—only once—and then the solution will find all the flows automatically and execute them.

Inputs to the solution include the URL of the website that needs to be tested, and a set of positive test data details to be used by the solution for validating the various interactions on the UI screen and validating the flows created by the solution.

The concept of Self-Healing included in the solution makes the Autonomous UI Testing solution robust to handling changes. The identification of added or updated functionality for a given UI compared to its previous version is the first step towards making the system autonomous. That is followed by a Deep Learning AI model which automatically updates the test flows. Statistical insights and the analysis of the self-healing module and automated execution provides the user with deeper understanding, and user feedback further fine-tunes the Deep Learning AI model.

CHALLENGES WITH CONVENTIONAL UI TESTING

It is important that UIs are tested thoroughly, but the process is often difficult. Challenges include:

- **Manual efforts are required**
  Current conventional models require testers to understand the underlying Automation Framework and have a programming background. It is time-consuming for users to extract all the Web Elements, create an object repository, and write the corresponding scripts to automate the flows. The process requires an intensive amount of work to create a suitable test flow and execute it.

- **Constant changes in UI**
  Change is inevitable, but quickly becomes a nightmare for testers. With each change in business requirements and user demands, there is a constant change in the UI also. This requires significant manual efforts to identify the changes in the UI and subsequently make updates in the automation test suites to all the scripts impacted. This situation essentially leads to continuous maintenance efforts to avoid the test suite from failing.

- **Limitations of conventional test automation tools**
  Current conventional UI Testing tools are too tightly coupled to a designated Test Automation Framework. Such tools come with pre-requisites requiring users to be aware of the Automation Tool’s nuances, with coding skills needed and an ability to write the test scripts according to the specific tool’s coding quality and standards.

The AI/ML model is complex enough to understand deeper functionalities of the UI and is simple enough to be used with incredible amount of data without fail. This model is also easier to use in continuous testing scenarios.
Benefits of Autonomous UI Testing

- **Decreased costs**
  With Script-less Automation, the solution provides savings on product development costs by eliminating the manual efforts involved in test script creation, test script maintenance and test execution.

  Being Framework agnostic the solution is not tied to any Automation Testing Framework (and in fact is designed to work without any Automation Framework). This eliminates the cost involved in purchasing licensed test automation tools.

- **Increased quality and test coverage**
  With the Automation UI Testing solution 100% test scenario/flow coverage is achieved to ensure the UI product suite under test is delivered with higher quality and far fewer defects. Valid and invalid flows are identified automatically, and saved for future reference.

- **Increased velocity and predictability**
  The solution provides excellent adaptability and throughput: It is capable of dynamically adapting to UI updates by implementing a comprehensive mapping of business flows and their tests. This enables quicker creation/selection and a more focused test flow designed to validate changes to the product, thus improving the test velocity and risk evaluation of any planned change/fix.

- **Reduced maintenance efforts**
  Being Self-healing, the Autonomous UI Testing solution heals itself as the UI changes, greatly reducing the manual effort needed compared to maintaining and updating the test cases for each change with conventional testing.

Architecture overview

The architecture of the Autonomous UI Testing solution can be divided into two parts, as depicted below:

**Architecture and building blocks**

<table>
<thead>
<tr>
<th>Initial setup</th>
<th>Regular execution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data for training</strong> &lt;br&gt; Deep Learning Model</td>
<td>1) Driver UI</td>
</tr>
<tr>
<td>2) UI Data</td>
<td>2) UI Data</td>
</tr>
<tr>
<td>3) Database</td>
<td>3) Database</td>
</tr>
<tr>
<td>4) Analysis</td>
<td>4) Analysis</td>
</tr>
<tr>
<td>5) Result Dashboard</td>
<td>5) Result Dashboard</td>
</tr>
</tbody>
</table>

- **Identifying changes**
  - UI Changes
  - Flow Changes

- **Training AI model**
  - Ref from database

- **Update/Add flows**

- **Testing with updated flows**
Initial setup:

For the first run, the UI-driver module will be executed. This is a Python-based program that runs and identifies all the flows automatically for all screens in a given URL and stores them as a reference in a database. The result of the execution is analyzed and displayed on a dashboard containing information about the flows the program identified and executed and their corresponding results. In parallel, the Deep Learning model is trained and engaged in self-healing for the next run.

Regular execution:

For each iteration the UI-driver module references the database for base flows and identifies the changes in the new UI. The changes are then matched against the original UI using the Deep Learning AI model trained earlier. The model then heals the flows as needed for the updated interface. Newly added features are also identified, which are sent along with updated flows to the UI-driver module for execution to analyze and store the results.

The comparisons and inference related to the previous and current execution, along with the details of self-healing and updated features, are shown on a user friendly dashboard. The dashboard enables the user to provide feedback to the solution, a process which makes the tool even more robust and accurate over time.