

Title / Titel

## Towards an Open and Dynamic Test Automation

---

Speaker(s) / Referent(s)

Cuomo, Vincenzo / ST Incard, Italy (IT)

---

To whom is the presentation addressed? / An wen richtet sich der Beitrag?

This presentation is for developer, testers and test managers dealing with test automation issues.

---

Keywords / Stichwörter

Test Automation, GUI Testing, Open Source, Java

---

Abstract / Zusammenfassung

As Test Automators, when testing GUI-based applications, using one of the off-the-shelf Test Automation tools currently available, we are 'prisoner' of the so-called "GUI Maps": static, a-priori, representations of the User Interface to test. These maps are required both to build and run the automated test scripts: they contain the references to the GUI objects to test and are used by the runtime environment of the Test Automation tool during the scripts execution. To get the GUI Maps the software application must be available for testing. This make impossible to write, even partly, the automated test scripts before the application is released for testing. In this way the Test Automation process will always start later in the software lifecycle.

Moreover, as any other software artefact, the GUI Maps need to be maintained: in case of GUI changes, you need to modify or (in most cases!) re-capture the maps. And GUIs change very often! Changes, even trivial, can affect dramatically the automated test scripts requiring continuous reworking. GUI maps, as static and a-priori tools, need a huge effort in building and maintaining the automated scripts. And why test automators have to care about these maps?

To address this problem, we started a new project, OpenTest, to design and build an innovative test automation framework trying to solve the above limitations. OpenTest is an open-source platform powered by an engine that, transparently to the user, gets the GUI Maps directly at runtime during the script execution. No static, a-priori, maps are required to launch the test scripts and test automators don't have to care about these objects.

When needed, the runtime environment captures automatically, in a standard XML representation called 'snapshot', the structure of the current user interface, searching for the GUI element to test. For example, if the script contains the instruction of pushing a button with caption 'OK', the system catches the snapshot of the current application's GUI structure, searching for an element of type='button' with a caption='OK'. Once found the button is pushed by the test automation robot. Testers don't have to provide any a-priori GUI Map containing the 'OK' button.

Decoupling the scripts from the GUI Maps allows writing the automated tests early in the software lifecycle, for example, just after the specification phase, running them immediately after the application is released for testing.

Removing the dependency from the GUI Maps, hence from the particular application under testing, allows porting, with less effort, the same testware in different testing projects against different applications.

---

As the dynamic management of the GUI Maps was the first challenge, finding the GUI object to test was also a critical task.

When the test scripts are derived from the application's specifications, not always the knowledge of the GUI objects to test is well stated and completely defined (this is the case of early draft or poorly written specifications). In many cases testers have partial information on the objects to test.

The search engine of the OpenTest platform implements several searching algorithms to find the GUI objects to test even in case of poor information. The provided searching criteria are configurable by users through an XML file.

Moreover, such searching flexibility allows managing many cases of GUI changes without modifying the test scripts, hence reducing the total maintenance effort.

But what happens if the GUI object to test is not captured or not found by the platform?

This could happen when the applications are built by not using standard GUI objects.

To solve this problem, and to increase the quality of the platform, OpenTest has been conceived with a plug-in based architecture to support extensions. All components have a plug-in engine that allows extending their functionalities.

So, if a proprietary GUI object type is not automatically captured by the platform, users may extend the snapshot manager component, by deploying a plug-in supporting the recognition of such object.

Testers can also extend the search engine by developing and deploying a plug-in with their own searching criteria that best fit their applications domain.

Currently OpenTest works with Java applications, but testers can extend the platform to work with applications written in different programming languages.

OpenTest has been tested with many applications from different contexts, to verify the functionality and performance of the snapshot manager and the searching engine, including plug-ins.

The platform has been also used in several test projects, and the main benefits held were: 1) early implementation of the test automation scripts and possibility of executing such tests more often; 2) more automated tests and less manual testing; 3) reduced maintenance effort (up to 60%); 4) higher reuse (40%) of test scripts through similar applications...

The work on OpenTest is far from being completed. The platform is under development to enhance its capabilities for managing the so-called 'unexpected GUI state'. An unexpected GUI state is a collection of one or more GUI objects, not expected in the test script. Unexpected states typically are due to bugs in the application under test or, sometimes, to changes. What should be happening if an unexpected GUI object, for example an unexpected error message window is encountered? Currently the majority of the test automation tools fails, and the test cases execution is blocked. The great innovation of OpenTest will be the capacity of managing such states. OpenTest should be able to try to 'resolve' the unexpected state and to continue the test execution. This could be possible by an 'intelligent' algorithm based on a set of 'known states' recorded from past test projects and 'behaviours' (i.e. sequence of actions) to resolve such states. Once an unexpected state is encountered the system should search in a data base of 'known states' for a similar state, and once found the associated behaviour must be executed. For example, in case of an unexpected error message the system will search in a data-base for a similar error message window and once found the associated behaviour will be executed (e.g. press the button 'Cancel' to discard the message). In case of success, OpenTest should be able to learn and store, for further use, both state and actions to resolve the faulty state.

### **Biography / Biografie**

Vincenzo Cuomo, since seven years, heads up a Software Testing and Quality Assurance group at ST Incard. He has more than ten years experience in software quality, testing and process improvement. He combines his experience and passion for test automation with a research approach to enhance the state-of-the-art of this fundamental matter. Vincenzo is frequent speaker at various conferences and writer of articles on software testing.

---

### **Contact information / Kontaktinformationen**

Cuomo, Vincenzo  
ST Incard  
Software Testing and Quality Assurance  
ZI Marcianise SUD

81025 Marcianise  
Italy

---