|  |
| --- |
| Asrc Federal Mission solutions engineering |
| Automated GUI Test Tool Evaluation |
| Automated Test Initiative |
|  |
| **Automated GUI Test Tool Evaluation** |
| **March, 2014** |

|  |
| --- |
|  |

# Table of Contents

[Table of Contents 1](#_Toc383001444)

[Introduction 5](#_Toc383001445)

[Problem Space 5](#_Toc383001446)

[Mission 5](#_Toc383001447)

[Automated Graphical User Interface Test Tools 6](#_Toc383001448)

[Test Levels 6](#_Toc383001449)

[Tool Capabilities 6](#_Toc383001450)

[Tools Evaluated 7](#_Toc383001451)

[Automated Test and Retest Test Manager (ATRT TM) Version 5.6.8 8](#_Toc383001452)

[eggPlant Functional Version 14.0.1 8](#_Toc383001453)

[Sikuli Version 1.0.1 8](#_Toc383001454)

[Evaluation Scenario Definition 8](#_Toc383001455)

[Scenario Overview 9](#_Toc383001456)

[Scenario Design 9](#_Toc383001457)

[Environmental Requirements 9](#_Toc383001458)

[Detailed Tool Evaluation 10](#_Toc383001459)

[ATRT 10](#_Toc383001460)

[Test Modularity and Reuse 10](#_Toc383001461)

[Test Creation Language or Mechanism 10](#_Toc383001462)

[Function/Sub-Test Definition 10](#_Toc383001463)

[Object Oriented Capability 10](#_Toc383001464)

[Tool Usability 11](#_Toc383001465)

[IDE Ease of Use 11](#_Toc383001466)

[Time to Create Common Scenario 11](#_Toc383001467)

[Time to Execute Common Scenario 11](#_Toc383001468)

[SUT Interaction and Performance 11](#_Toc383001469)

[Image Capture and Scan 11](#_Toc383001470)

[Collaboration 11](#_Toc383001471)

[Other 12](#_Toc383001472)

[Linking Requirements to Test Steps 12](#_Toc383001473)

[Test Execution Reporting Capabilities 12](#_Toc383001474)

[Customer Support and Licensing 13](#_Toc383001475)

[eggPlant Functional 13](#_Toc383001476)

[Test Modularity and Reuse 13](#_Toc383001477)

[Test Creation Language or Mechanism 13](#_Toc383001478)

[Function/Sub-Test Definition 14](#_Toc383001479)

[Object Oriented Capability 14](#_Toc383001480)

[Tool Usability 14](#_Toc383001481)

[IDE Ease of Use 14](#_Toc383001482)

[Time to Create Common Scenario 14](#_Toc383001483)

[Time to Execute Common Scenario 15](#_Toc383001484)

[SUT Interaction and Performance 15](#_Toc383001485)

[Image Capture and Scan 15](#_Toc383001486)

[Optical Character Recognition 15](#_Toc383001487)

[Collaboration 15](#_Toc383001488)

[Other 15](#_Toc383001489)

[Linking Requirements to Test Steps 15](#_Toc383001490)

[Test Execution Reporting Capabilities 15](#_Toc383001491)

[Customer Support and Licensing 16](#_Toc383001492)

[Sikuli 16](#_Toc383001493)

[Test Modularity and Reuse 16](#_Toc383001494)

[Test Creation Language or Mechanism 16](#_Toc383001495)

[Function/Sub-Test Definition 16](#_Toc383001496)

[Object Oriented Capability 16](#_Toc383001497)

[Tool Usability 16](#_Toc383001498)

[IDE Ease of Use 16](#_Toc383001499)

[Time to Create Common Scenario 17](#_Toc383001500)

[Time to Execute Common Scenario 17](#_Toc383001501)

[SUT Interaction and Performance 17](#_Toc383001502)

[Image Capture and Scan 17](#_Toc383001503)

[Collaboration 18](#_Toc383001504)

[Other 18](#_Toc383001505)

[Linking Requirements to Test Steps 18](#_Toc383001506)

[Test Execution Reporting Capabilities 18](#_Toc383001507)

[Debugging 18](#_Toc383001508)

[Customer Support and Licensing 18](#_Toc383001509)

[Testing Environment and Design Considerations 18](#_Toc383001510)

[Findings and Lessons Learned 19](#_Toc383001511)

[Tool Performance Summary 19](#_Toc383001512)

[Testing Environment 19](#_Toc383001513)

[Host Machine: 19](#_Toc383001514)

[System Under Test: 19](#_Toc383001515)

[Tool Comparison Criteria and Evaluation 19](#_Toc383001516)

[Overall Score 20](#_Toc383001517)

[Automated Test Applicability 20](#_Toc383001518)

[Caveats 21](#_Toc383001519)

[Conclusion 21](#_Toc383001520)

[Appendix A 22](#_Toc383001521)

[License Quotes For ATRT and eggPlant 22](#_Toc383001522)

[Appendix B 26](#_Toc383001535)

[Scenario Design 26](#_Toc383001536)

[Scenario Steps 26](#_Toc383001537)

[Appendix C 29](#_Toc383001538)

[Implementation Notes 29](#_Toc383001539)

# Introduction

Defense budgets are growing tighter and system/software developers are being pressured to create higher-quality products for less money. As a result, it is becoming increasingly important to find ways to reduce costs and improve the efficiency of testing.

Testing has historically been a manual effort requiring extensive human interaction with the system under test (SUT). The high cost of human labor can drive programs to reduce the overall amount of testing performed. Although this reduces costs, it also increases the risk that problems will not be discovered until late in the development cycle, if at all. The sooner defects are found in the life cycle of a program, the cheaper and easier they are to fix.

Test automation can provide a solution by reducing the manpower necessary to perform exhaustive product testing at multiple levels.

# Problem Space

There are two general approaches to test automation:

* Code-driven testing - The interfaces to classes, modules or libraries are tested with a variety of input arguments to validate that the results returned are correct.
* Graphical user interface (GUI) testing - A testing framework generates user interface events such as keystrokes and mouse clicks, and observes the changes that result in the user interface, to validate that the observable behavior of the program is correct.

## Mission

The purpose of this effort is to evaluate commonly used Automated GUI Testing tools in order to gain experience with them, compare and contrast their features, and assess their strengths and weaknesses as a whole.

A variety of tools exist for this task, both proprietary and open source. The tools chosen for evaluation are **Automated Test and Retest, Test manager** (ATRT TM) by Innovative Defense Technologies, **eggPlant** Functional by TestPlant, and **Sikuli** which is Open Source. Detailed evaluations of each were conducted and results are included below.

# Automated Graphical User Interface Test Tools

Automated GUI Test Tools use image matching technology to manipulate GUI applications on the “System Under Test” (SUT). The tools "see" the screen of the SUT, and can recognize, for example, when a particular window, icon or text appears onscreen. Because the tool only looks at the "screen" of the SUT, the technology used in the application to be tested does not matter. The tool requires no direct ties to the SUT source code. This is advantageous because it allows the SUT to be tested as a black box without the need for special instrumentation.

Many of the available tools utilize a two-computer system consisting of a host controller machine, where scripts are authored and executed, and a SUT machine which runs a “Virtual Network Computing” (VNC) server. The test tool connects to the VNC server via a built-in viewer connection. Thus, the SUT could be any system that has a VNC server for it.

## Test Levels

Automated GUI Test tools can support all levels of testing that require the ability to “record and playback” GUI interaction. However, they are most useful for system and component level tests where the SUT utilizes a GUI to execute system actions. Applicability to software test levels is summarized in the following table.

|  |  |  |
| --- | --- | --- |
| Test Level | Test Level  Description | Applicability to  Automated GUI Test Tools |
| 1 | Unit Testing – tests that verify the functionality of a specific section of source code at the function/class level. | **Low** – generally performed via code driven testing methods. May be utilized to automate GUI based unit test drivers. |
| 2 | Integration Testing – tests that seek to verify the interfaces between components against a software design. | **Low** – generally performed via code driven testing methods. May be utilized to automate GUI based component test drivers. |
| 3 | Component interface Testing – tests that validate the handling of data passed between various units, or subsystem components. | **Moderate** – useful for testing GUI driven components themselves or driving the system GUI to test other components. |
| 4 | System Testing – tests which verify that a fully integrated system meets its requirements. | **High** – very useful to automate GUI driven user tasks in order to exhaustively and regressively test system requirements. |
| 5 | Acceptance Testing – tests performed for or by the customer as part of the delivery hand off process. | **Moderate** – can be useful if the customer is involved in validating the automated tests, but generally this level of testing is performed manually |

Table 1 Applicability of Operator Action Based Automated Testing

## Tool Capabilities

In order to adequately test graphically oriented software, a tool must capture the actions and reactions of the operator and the user interface. This requires capabilities to capture screen images and operator actions, play them back on an automated or ad hoc basis, and validate the results.

Because the tool will be part of the project’s development system, it must support team collaboration and provide the ability to manage and version control tool artifacts (model files, script files, image files, etc.).

The following tool capabilities are needed:

* GUI Manipulation – ability to interact with the SUT GUI by generating mouse and keyboard events as necessary to manipulate the interface elements. Provide the means to control the timing and playback of sequential interactions.
* Image Capture/Comparison – ability to capture the state of the SUT GUI and scan it for the existence of desired images (buttons, window titles, toolbar icons, etc.). Image content is subject to underlying system scaling and resolution differences. Image comparisons need to be tolerant of these differences.
* Optical Character Recognition – ability to scan captured SUT GUI images and extract textual content for comparison to expected values, or for the purpose of obtaining displayed calculated result values.
* Integrated Development Environment (IDE) – ability to create tests; start, pause, debug, and stop tests; and manipulate tool artifacts in a coordinated and developer friendly manner.
* Team Collaboration – support multiple user roles for test development, test execution, and test result management for issue tracking.
* Artifact Management – provide features to manage and version control the tool artifacts or support the use of external source management tools.

# Tools Evaluated

As already stated the tools chosen for evaluation are Automated Test and Retest, Test manager (ATRT TM) eggPlant Functional, and Sikuli. The following table and subsequent sections describe several characteristics of each.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tool** | **Developer** | **Host** | **SUT** | **Test Creation Mechanism** | **License/Cost** |
| ATRT 5.6.8 | Innovative Defense Technologies idtus.com | Linux, Mac OS X, Windows | Any that support VNC | Graphical Flowchart | Commercial (see Appendix A) |
| Eggplant Functional 14.0.1 | TestPlant testplant.com | Linux, Mac OS X, Windows | Any that support VNC | Scripting (SenseTalk) | Commercial (see Appendix A) |
| Sikuli  1.01 | Sikuli Script sikuli.org | Linux, Mac OS X, Windows | Linux, Mac OS X, Windows | Scripting (Python) | Open Source MIT License Free |

Table Evaluated Automated GUI Test Tools

## Automated Test and Retest Test Manager (ATRT TM) Version 5.6.8

ATRT TM is a commercial product provided by Innovative Defense Technologies, LLC.

* ATRT TM provides a model-based graphical user interface allowing test writers to drag and drop elements onto a canvas to build tests. Test steps and graphical images are stored in the model.
* There is a capability to create reusable functions that may be shared among models.
* Written tests are completely graphical in nature; there is no scripting language or lines of code associated with the developed tests.
* ATRT requires a commercial license. The base cost is $7500 for a single user license for the first year, and $2500 for each subsequent year which includes support and updates. Multiple developers may use ATRT but only one at a time. Please see Appendix A for additional options.
* For more information see <http://idtus.com/products/atrt-test-manager/>.

## eggPlant Functional Version 14.0.1

eggPlant is a commercial tool suite produced by TestPlant. eggPlant Functional is the automated GUI test development and execution tool in the suite. Other eggPlant tools provide additional capabilities to manage test suites, do performance testing, do continuous integration testing, etc. eggPlant Functional was the only tool in the eggPlant Suite that was evaluated. eggPlant Functional is hereinafter referred to as eggPlant.

* eggPlant’s default scripting language is based on the test definition language “SenseTalk”, although the documentation states that other languages may be used.
* eggPlant’s IDE generates an organized directory structure with user provided file names, and all lend themselves well to a source control system.
* eggPlant requires a commercial license with many options. Please see Appendix A for details.
* eggPlant runs on a host computer running Linux, Mac OS X, or Windows. It controls the SUT via VNC.
* For more information, see <http://www.testplant.com/eggplant/testing-tools/eggplant-developer/>.

## Sikuli Version 1.0.1

Sikuli is open source software released under the MIT license.

* Sikuli’s scripting language is Python and is well documented with an abundance of books, examples and tutorials.
* It provides specific extensions to Python libraries, which are in turn extensible by the test writer.
* Sikuli’s default IDE generates an organized directory structure with user provided file names, and all lend themselves well to a source control system.
* For more information, see <http://www.sikuli.org>.

# Evaluation Scenario Definition

A well-defined GUI test scenario was implemented for each tool with the goal of evaluating tool features, capabilities, and performance under common conditions. In order to facilitate evaluation, the continuous integration tool, **Jenkins** <http://jenkins-ci.org/> is used to automate running the common scenario. This provides the benefits of being able to coordinate starting the scenario with one click along with the capture of scenario execution times. The evaluation scenario is described in the following paragraphs.

## Scenario Overview

The SUT chosen for this evaluation scenario is an ASRC Federal MSE developed web application called CIWi (Component Installation Wizard). CIWi provides capabilities for packaging and deploying software components over a network to distributed remote hosts. CIWi’s GUI provides features for selection of a packaged software component and automates deployment of the component’s files onto all hosts where the software was defined to be installed.

The CIWi GUI provides an adequately complex system for the purpose of evaluating the Automated GUI Test tools. The scenario automates use of the CIWi GUI to install and then uninstall a packaged software component on one network host.

## Scenario Design

With respect to the details of the scenario, please see **Appendix C** for the scenario design. The time to both implement and execute the scenario with each tool is recorded as a metric and included in the “**Detailed Tool Evaluation, Tool Usability”** sections of the respective tools, as well as in the summary table in the **“Findings and Lessons Learned”** section.

## Environmental Requirements

The tools evaluated in this study rely on visual technology to automate and test graphical user interfaces using screenshot images. As such the host and SUT must provide a capability for screens to be displayed and manipulated during testing. To this end a Virtual Network Computing (VNC) Server and a VNC Client are used for access to the GUI environment. In summary,

* Automated GUI testing requires Virtual Network Computing (VNC) sessions
* VNC sessions communicating over the network enable script development and execution
* VNC sessions are only required to support graphical interactions
* Executable programs not relying on graphical interaction do not require VNC sessions
* Tests requiring sub-second responses are not candidates for tools that rely solely on image recognition

The following figure represents the architecture.

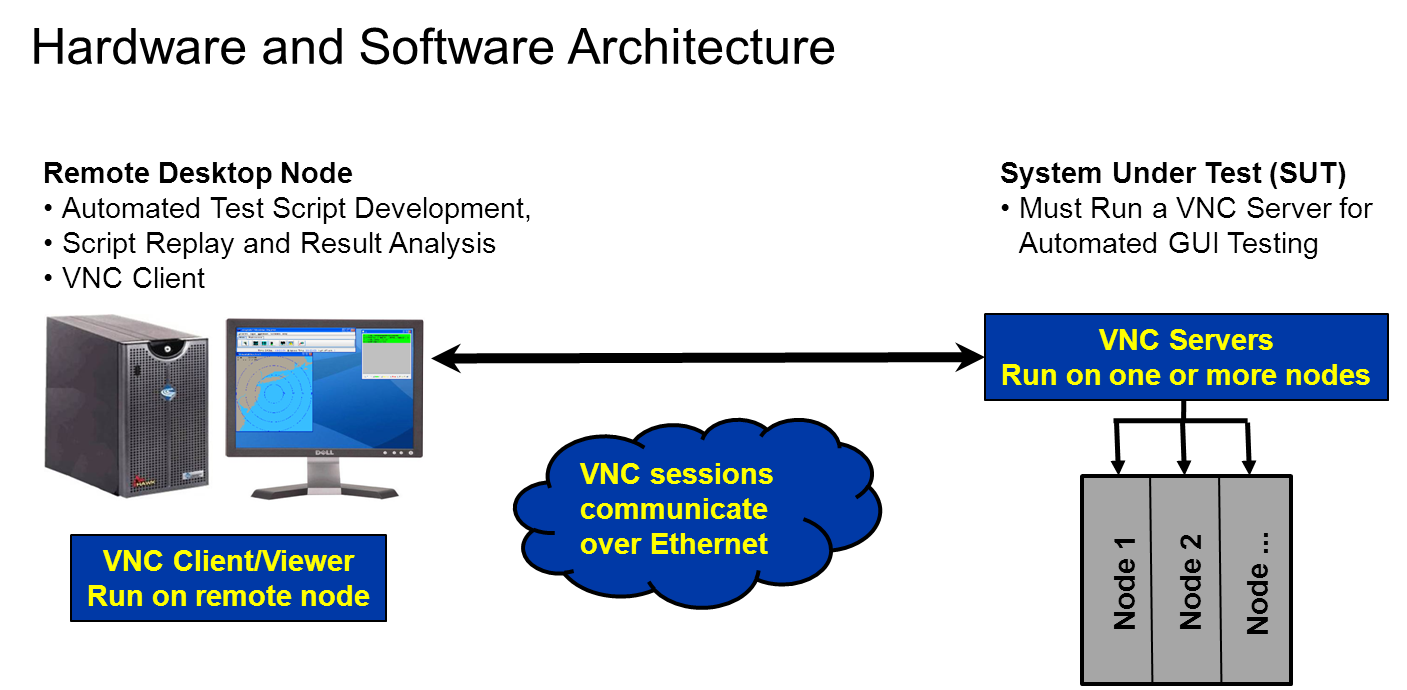


Figure Hardware and Software Architecture Requirements

## Detailed Tool Evaluation

The following sections dissect each tool in detail using a common set of criteria. The section titled, **“Findings and Lessons Learned”** includes a comparison summarizing the criteria and performance of each.

### ATRT

#### Test Modularity and Reuse

##### Test Creation Language or Mechanism

ATRT TM provides a model-based graphical user interface allowing test writers to drag and drop elements onto a canvas to build tests. This could potentially allow for non-developers to come up to speed faster than if they were required to learn a new scripting language before creating tests. Most of the usual flow control constructs are available in some form using the provided graphical elements.

##### Function/Sub-Test Definition

There is a capability to create reusable functions that may be shared among models. Functions are handled by a Function Manager and may be inserted as needed into test cases. When using ATRT it’s important to note that both the name and functionality of the inserted instance can change and changing the name has no bearing on the original function. However any changes made to the function logic will change the original, and more importantly the original’s use in other existing test steps. This isn’t necessarily considered a drawback but it must be understood when using functions.

##### Object Oriented Capability

ATRT leans toward generation of procedurally based scripts rather than object or object oriented based. Test case, test step, and function reuse is certainly possible and straightforward but creating classes and objects with state and behavior does not appear to be supported.

#### Tool Usability

##### IDE Ease of Use

With respect to creating tests, it was found that using ATRT’s IDE is somewhat cumbersome. For instance, in order to capture a graphical image, the test writer must open a project, a test case, a test step, and then the test step must be the “active” canvas in the IDE before any images on the system under test (SUT) are available for capture. This was one of the most frustrating idiosyncrasies of the IDE because being in the wrong “canvas” happened quite often.

Passing parameters in ATRT is possibly the most cumbersome aspect evaluated in the ATRT IDE. In order to pass parameters they must first be defined and assigned values using multiple IDE graphical elements. The receiving test step must then be created, employing and configuring more IDE elements. Finally, the enclosing test case containing these new test steps must chain the steps together using additional IDE capabilities. Also, there is no compilation or syntax checking so if one or more of the above is omitted or done wrong, there appears to be no feedback to aid in the debug of the parameter passing. The test fails during execution and the user is left to reviewing the graphical test steps to try and find the omission or error.

##### Time to Create Common Scenario

It required 6 hours 30 minutes to code the passing case only; the failure case was not completed due to management decision. Note that even though only one of the two scenarios were coded, more time was spent developing only one scenario in ATRT than required to code both scenarios in the other tools.

##### Time to Execute Common Scenario

* 2 minutes 45 seconds for passing case only.
* Failure case was not completed due to management decision.

ATRT was the slowest to execute the scenario.

#### SUT Interaction and Performance

##### Image Capture and Scan

As with the other tools, images are captured from the system under test using the mouse to click and drag a “rubber band” around the desired image. Images are saved in the model as portable network graphics (PNG) files. There is a capability within the IDE to zoom to the pixel level, providing a “Limited Pixel Image Search” for refining the image match. This allows for tuning of the match sensitivity, making the algorithm more or less stringent. As with the other tools, there are capabilities to limit the search to a specific region on the screen, and to choose a “click point” or “hotspot” so that the center of the matched image is not the only place for the mouse to click. This is useful for instance when trying to find the correct “File” menu when there may be several visible on a screen at one time.

ATRT assigns names to the PNG file using strings of numbers and these names may not be overridden. Images may be shared among test cases by opening multiple windows and copying the image between them but there is no easy way to know if an image is shared among other test cases. This may be important because changing the pixel set for one image changes it for all test cases referring to that image, and it is difficult to know if it is shared. By comparison, the other evaluated tools allow for user defined file names, where these names can be searched throughout the files in the file system (using operating system tools) to find any additional references. This operating system search method is potentially possible in ATRT but not as straight forward due to the ATRT file naming scheme.

##### Collaboration

This area is perhaps ATRT TM’s biggest weakness. As previously stated, all artifacts created by ATRT are stored in a model. The file names for tests and image captures are generated automatically and are not user configurable. These names are typically 36 hexadecimal characters long, not including the extension, and are not meant to be edited (or even effectively viewed) outside of the model. This means that single files are not good candidates for configuration management or source revision control tools. Exporting of a model is possible but the output is one compressed file. The only useful means of source control is to provide a descriptive name to the exported compressed model file and then to add it as a lump sum to a revision control system. In effect this creates a restore point of a known good model and set of tests in case future changes or model corruptions render the current model undesirable.

Also, ATRT does not provide a useful facility to compare models. Test writing and collaboration among developers does not scale to more than a few test authors per model. In order to avoid overwriting another’s work, diligent manual communication among developers is a requirement.

#### Other

##### Linking Requirements to Test Steps

According to documentation, ATRT TM provides the ability to both document and import requirements, to associate requirements to test cases, and provides full traceability of all test artifacts back to the requirements. ATRT is the only tool that advertises this capability but it was not evaluated in this study.

##### Test Execution Reporting Capabilities

ATRT is capable of providing very good reports with varying levels of detail and is the best of the tools tested. This capability is one of the strongest assets of ATRT. An example is provided below.

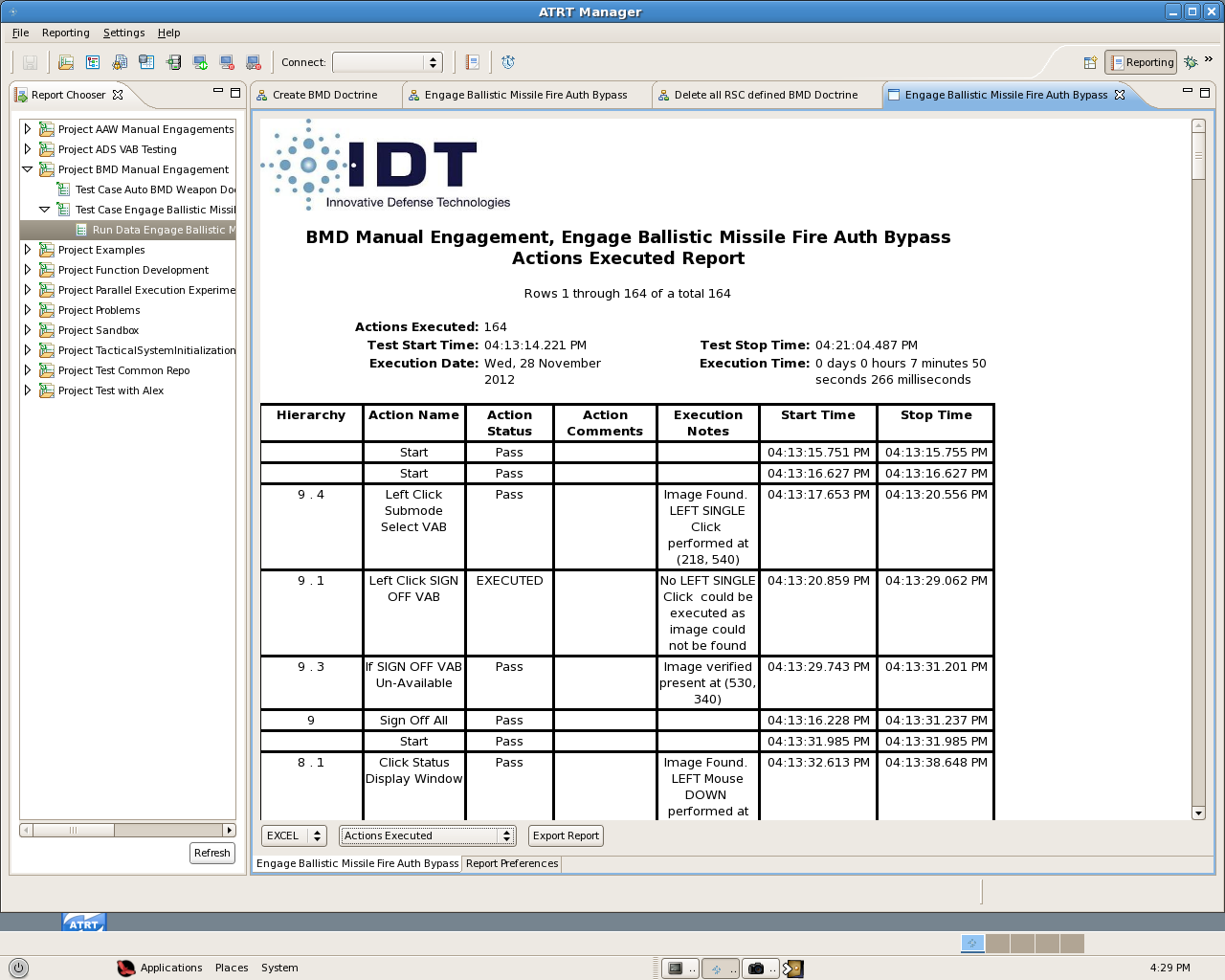


Figure ATRT Example Report of Test Execution

##### Customer Support and Licensing

ATRT customer support was very responsive but this was sales support. Technical support for this effort was not evaluated. Finally, there are multiple options for licensing of ATRT. Please see Appendix A for details.

### eggPlant Functional

#### Test Modularity and Reuse

##### Test Creation Language or Mechanism

eggPlant scripts are written in a proprietary language called “SenseTalk.” It is a powerful high-level language but it can be very verbose. It has syntactical elements similar to other common, modern programming languages. However, its verbosity can be frustrating for developers used to more concise languages.

The fact that SenseTalk is proprietary limits its portability, and also limits the number of libraries available for use. The language reference documentation available at the product website was informative and seemed complete, but the layout was a little confusing making it sometimes difficult to find specific information. No third party SenseTalk reference books were located during this effort.

SenseTalk scripting is used in conjunction with a "guided record" mode, meaning the scripter uses the eggPlant IDE to remotely manipulate the SUT GUI, record the actions, and verify expected SUT GUI reactions with SenseTalk script commands. The user directly views the SUT screen in a VNC viewer window. Available actions are selectable from a tool bar displayed at the top of the window. With the IDE in “capture mode” an action taken on the SUT GUI is recorded as a SenseTalk command in the script file currently active in the IDE, and a corresponding captured screen image is stored as a PNG (Portable Network Graphics) file. The SUT immediately responds to the action allowing the scripter to continue the process and generate another script command to verify the expected SUT GUI reaction. The script file can later be manually modified within the IDE to incorporate additional logic if necessary.

##### Function/Sub-Test Definition

SenseTalk provides handler and function constructs which appear to be equivalent to subroutines and functions in other modern languages. Argument values can be passed and returned by value and by reference.

##### Object Oriented Capability

SenseTalk is an object oriented (OO) language; however, its syntax and concepts are somewhat different than other modern OO languages. Most other languages define classes of objects and implement behavior at the class level, for all objects in that class. In SenseTalk, each individual object has its own script, so it can have its own unique behavior. SenseTalk has no classes, but its helpers provide a similar set of capabilities by allowing objects to use functionality provided by any number of other objects (multiple inheritance). SenseTalk refers to this as an “all-object (classless) approach”.

Experimentation was done with some SenseTalk OO capabilities (scripts as objects) and found them to be very useful for abstracting SUT elements, but the syntax and concepts are not straight forward and do incur penalty for learning curve.

#### Tool Usability

##### IDE Ease of Use

The eggPlant IDE is extremely easy to use. The work flow of capturing SUT GUI actions and reactions within the IDE as script commands and image files was intuitive and easy to understand. An image editing tab in the IDE provides an easy mechanism to name and manipulate the captured image files and define “click points” for mouse clicks. The IDE includes a debugger allowing scripts to be easily started, stopped at breakpoint, single stepped, etc. Source editing capabilities were good but not on par with other common IDE’s like Eclipse or NetBeans. Source syntax highlighting and automatic indent level features were available, but sometimes responded strangely at times.

##### Time to Create Common Scenario

eggPlant was the first tool to be used to implement the common scenario. It took approximately 4 hours to complete. However, this included 30 to 45 minutes of debugging the scenario because CIWi was left in a strange state when an intentional failure was induced. This was a scenario logic issue, not a tool problem. The workaround for this was twofold:

* Add a step to the scenario to detect the error condition and close the error dialog box (click OK).
* Configure Firefox to always run in private browsing mode so that the CIWi session is reset when the VNC server is stopped and the browser closes.

This solution was carried forward and implemented for the other tools so development times were reduced for the others. Even with that, scenario creation took longer in each of the other tools than it did with eggPlant.

##### Time to Execute Common Scenario

Jenkins recorded the time to execute the scenarios with eggPlant to be:

* 55 seconds for successful completion of the entire scenario.
* 40 seconds when failure was induced.

#### SUT Interaction and Performance

##### Image Capture and Scan

Images are captured from the SUT by using the mouse to click and drag a “rubber band” around the desired image in a connected VNC window. After capture, the IDE displays a dialog box showing the captured image and allowing and an image file name to be assigned. eggPlant automatically does an Optical Character Recognition (OCR) scan of the image and suggests a file name based on the image’s textual content (if any) – a nice feature. Images are saved in the test suite file system “Images” directory as PNG files, but this can be user overridden. Test scripts reference the images by file name without the “.png” extension, so file naming is important and useful for code readability. For example, the script command **Click “FileMenu”** refers to the FileMenu.png image file and clearly indicates the intent of the code. The dialog box also provides the capability to select the means by which the image is scanned when the script runs (tolerant, precise, OCR text, etc.).

A list of all captured images is displayed by the IDE. Selecting an image file in the IDE opens a tab for the image permitting edit of all metadata. The entire process is intuitive and flow between scripting and image capture mode is seamless.

##### Optical Character Recognition

As described above, image capture and scan work seamlessly with eggPlant OCR capabilities.

#### Collaboration

Artifacts created by EggPlant are stored as individual files within a user defined file system location. Test scripts, result logs, and project properties are stored as simple text files, and screen images are stored as PNG binary files. This layout can be easily supported by various third party source control systems (svn, git, ClearCase, etc.) to manage project files across multiple team members.

The cost associated with eggPlant licensing is a limiting factor to team collaboration, especially when compared to free, open source alternatives such as Sikuli. Costs will limit the number hosts where the tool can be installed, and impact availability across the team.

#### Other

##### Linking Requirements to Test Steps

eggPlant does not provide this capability.

##### Test Execution Reporting Capabilities

Test result data (logs, screen images, etc.) are stored under the test suite Results directory for each run of the test suite. Clicking “Show Results” from within the IDE’s Run Window opens a tab that lists the date/time of each test execution, and success/failure metrics. Selecting a specific test execution date/time displays more details for that test run including a timestamp and status for the each script command and a full screen image for the state of the SUT screen for any failures. The information seemed complete and accurate.

##### Customer Support and Licensing

TestPlant customer support was difficult to contact. They did not respond to an email request on their website for a trial license. Calls to various contact numbers provided on their website went unanswered for several attempts over the span of an afternoon. Eventually we were able to leave a message with an operator. We were contacted soon after by sales personnel, and a 2 week trial license was promptly emailed for evaluation. Technical support for this effort was not evaluated. Also, the 2 week trial was extended for another 2 weeks to allow for further testing. There are multiple options for licensing of eggPlant. Please see Appendix A for details.

### Sikuli

#### Test Modularity and Reuse

##### Test Creation Language or Mechanism

Sikuli’s scripting language is Python, a high level, general purpose language which is well documented with an abundance of books, examples and tutorials. It must be noted that the version of Python used in Sikuli is not based on the standard Python interpreter, C-Python, but instead is based on the Jython interpreter. Jython is an implementation of the Python language for the Java platform. This means that user written or Open Source supplied Python/Jython modules will run if they are supported by Jython 2.5.1 (as of this writing). If any code references the C-type interface (inline C code and/or direct access to C/C++ libraries) then that code will not run in Sikuli because it is not supported by Jython. For more information see “Import user defined Python Modules” (<https://answers.launchpad.net/sikuli/+faq/1114> ) and “Language and Syntax” (<http://www.jython.org/jythonbook/en/1.0/LangSyntax.html>) for more information.

##### Function/Sub-Test Definition

Sikuli handles functions and sub-tests like any typical high level Object Oriented language. See the next section for more details.

##### Object Oriented Capability

Because the language for Sikuli automated test scripts is Python, Sikuli scripts are fully Object Oriented. Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. A wealth of information may be found at <http://docs.python.org/2/tutorial/classes.html>.

#### Tool Usability

##### IDE Ease of Use

Sikuli’s IDE is rather sparse. It provides a basic text editing pane for writing and saving scripts and a chooser for selecting operator GUI actions for image capture, text input, etc. The documentation states there is a capability to use other IDE’s but this was not tested.

In order to break a project up into multiple source files, there are a few rules that must be followed to make the IDE aware of the environment. These were not difficult to find in online documentation but if the rules are not adhered to and saved in the proper places the IDE will not be able to execute the scripts.

As of this writing the IDE contains a few bugs that range from annoying to quite troublesome. The two most prevalent are:

1. Sikuli IDE Preferences - The first problem relates to directory preferences where captured images and scripts are stored. If this directory is not set **prior** to attempting to save the scripts and images, the Sikuli IDE will not properly save the data.
2. Sikuli/Java Environment - The second problem relates to the IDE not running. If a simple Xterm can be displayed from the command line, then the problem could be due to the settings for the Java environment automatically created by the Sikuli IDE. When the Sikuli IDE runs, it creates and updates a preferences file in ~userhome/.java/.userPrefs/org/sikuli/ide/pref.xml (pay attention to the “.” in .java and .userPrefs). If the Sikuli IDE is not displaying or running, then deleting that pref.xml file may be necessary before rerunning the Sikuli IDE. If that doesn’t fix the problem then deletion of the entire Sikuli directory from that path is the next step. That should fix the environment problem but doing this will remove preferences (Issue 1) and they must be set again before saving any Sikuli scripts.

Also, during this effort there was one occurrence of an anomaly where the IDE threw an internal exception and changes made since the last save were lost. Fortunately saves were done very often so there was no impact in this situation; things could have been much worse.

##### Time to Create Common Scenario

It took approximately 5 hours and 45 minutes to implement the common scenario with Sikuli - approximately 1.5 times longer than it took with eggPlant but less time than ATRT.

##### Time to Execute Common Scenario

Jenkins recorded the time to execute the scenarios with Sikuli to be:

* 2 minutes 20 seconds for successful completion of the entire scenario.
* 1 minute 20 seconds when failure was induced.

These times are about 50% slower than eggPlant execution times, and slightly faster than ATRT execution times.

#### SUT Interaction and Performance

##### Image Capture and Scan

As with the other tools, images are captured from the system under test using the mouse to click and drag a “rubber band” around the desired image. As with the other tools images are saved in the user defined Sikuli directory structure as portable network graphics (PNG) files and the IDE allows for these files to be saved using descriptive names.

The Sikuli IDE also has a “Matching Preview” view. When activated this provides a snapshot of the currently displayed screen and allows the script writer, during script development, to determine if the captured image is found once, more than once, or not at all. It also allows for tuning of the match sensitivity, making the algorithm more or less stringent.

As with the other tools, there are capabilities to limit the search to a specific region on the screen, and to choose a “click point” or “hotspot” so that the center of the matched image is not the only place where the mouse click will take place. This is useful for instance when trying to find the correct “File” menu when there may be several visible on a screen at one time. However one of the limitations with Sikuli is in its implementation of image “hotspots”. Hotspots are limited to the view of the *captured* image. In contrast, eggPlant allows the hotspot to be set *prior* to saving the image, enabling the image to be a small rectangle while still providing the capability of setting the hotspot a significant distance (in number of pixels) away from the center of the captured image.

#### Collaboration

Sikuli’s IDE generates an organized directory structure with user provided file names, and all lend themselves well to a source control system.

#### Other

##### Linking Requirements to Test Steps

This capability is not available in Sikuli.

##### Test Execution Reporting Capabilities

Sikuli’s test execution and reporting capabilities are the weakest among the evaluated tools. When running a test script, the Sikuli IDE closes its window before the script executes. After several seconds of inactivity, and what appears to be initialization time, scripted actions begin taking place. The execution time of the script is on par with the other tools, as long as the images are found. When an image is not found, there is another period of inactivity and the IDE is redisplayed, printing out where the script stopped and why. The problem is there appears to be no capability to pause or interrupt a running script, or to debug it. The script simply stops and reports either the line or function that failed.

##### Debugging

Sikuli’s default IDE is the only one evaluated that does not have a debugger. Debugging in the default IDE is possible but it relies on the creativity of the developer. In most cases debugging likely requires “debug” code to be placed at strategic places within the source to allow the executing script to stop as desired, and then deleted when the bug is fixed. At best it is cumbersome.

##### Customer Support and Licensing

Sikuli is Open Source and uses the MIT license. There is an active community of developers and users online.

##### Testing Environment and Design Considerations

Sikuli does not automatically manage VNC connections like the other tools. In eggPlant and ATRT the IDEs and test scripts provide the functionality of connecting to and executing via a VNC running on the SUT. In contrast, Sikuli does not provide a means of connecting to a remote SUT but instead simply captures what it “sees”. The obvious and easiest solution would be to install Sikuli on the SUT and run the tests from there. However the goal was to make all tools execute in the same manner, which is to run on a local host and test a remote SUT. The added benefit to this approach is it more accurately emulates the current “real world” tactical environment. After careful deliberation the following methodology was used:

For test development:

* Manually open a vncviewer on the test host (rhel1) connected to the vncserver on the SUT (rhel10)
* Start Sikuli on the test host (rhel1)
* Use Sikuli to manipulate the SUT (rhel10) via the vncviewer window.

For test execution:

* Ensure that xhost + is performed on the SUT vncserver (added step to Jenkins project that starts vncserver: ssh nagios@sut “export DISPLAY=:3; xhost +” )
* Run Sikuli on test host with DISPLAY exported to SUT.
* export DISPLAY=sut:3
* /home/autotest/SikuliX/runIDE -r /home/autotest/AutomatedTest/Scenario/sikuli/Scenario.sikuli

# Findings and Lessons Learned

## Tool Performance Summary

### Testing Environment

All tools were evaluated on MSE’s unclassified personal computers.

#### Host Machine:

* OS: RedHat Linux 6.4
* CPU: Intel Core 2
* RAM: 4 GBytes

#### System Under Test:

* OS: RedHat Linux 6.4
* CPU: Intel Core I7
* RAM: 4 GBytes

With respect to execution performance, eggPlant completed the scenarios at least 50% faster than the other tools. Sikuli was next and performed the tasks faster than ATRT, and ATRT performed about as fast as or slightly faster than a knowledgeable human operator. However the scenario execution times are meaningful only for comparison to the execution times of the other tools evaluated. The results are not meant to imply that ATRT and Sikuli are not sufficiently responsive, or that eggPlant is right for "time critical" testing. Please see the section titled, **“Evaluation Scenario Definition”** and especially **“Automated Test Applicability”** for details.

### Tool Comparison Criteria and Evaluation



Table Comparison of Evaluation Criteria GUI Automated Testing Tools



Table Comparison of Evaluation Criteria GUI Automated Testing Tools (continued)

### Overall Score

To provide a means of showing “at a glance” how the tools stack up against one another, subjective ratings were used in several of the columns in the above comparison matrices. The ratings are based on a scale of 1 to 10 (10 being the best) with a total possible score of 100. The scores are summarized below.

|  |  |
| --- | --- |
| Tool | Overall Score (out of 100) |
| ATRT Test Manager | 57 |
| eggPlant | 79 |
| Sikuli | 70 |

Table Tool Comparisons Overall Score

## Automated Test Applicability

Obviously, the creation of an automated test requires more time than manually executing that test one time, so the highest return on investment is realized on tests intended to be run repeatedly or frequently. Types of tests in this category can include Load and Cycle, Performance Measurement, and Regression tests where common, unchanging functionality may be verified with each software release. Additionally, the best cases for automation are tests where operator intervention is not needed. If a user is required to perform manual steps in order to fulfill successful execution of an automated test, such as loading of an external system for example, then the investment in and benefit of the automation is reduced.

Moreover, the tools evaluated in this study rely heavily on image recognition and repeatable results. If the SUT’s graphical displays are still in flux, or the system behavior is non-deterministic (i.e. “it works sometimes”), then the test complexity, rework, and maintenance of the automated tests will be very high in terms of both time and expense. Because of this automated tests in general are best suited for systems and software that have reached a more mature, stable state.

Lastly, because graphically oriented automated testing uses image capture and subsequent recognition, test steps using screen searches may take many seconds to complete. Restricting search regions can improve response times however tests requiring sub-second responses are typically not candidates for graphically oriented automated tests. In these cases, message based test drivers and post analysis of recorded data are better choices.

## Caveats

None of the tools caused an appreciable or even noticeable runtime impact on either the host or the system under test (SUT) machines. There are two caveats to this.

1. No tests were run with time critical (sub-second) response requirements. In general, tests requiring sub-second responses are not candidates for tools that rely solely on image recognition (see **Automated Test Applicability** section above).
2. Neither the host nor the SUT machines were running any time critical processes so an evaluation must be performed on the target machines before a final decision is made.

# Conclusion

All of the tools evaluated were capable of performing the automated test scenario adequately with each one having their respective strengths and weaknesses. The highest performer in terms of usability and script execution times is eggPlant but it has a recurring annual cost and is the most expensive. Sikuli is the runner up. Its IDE is less mature and less “user friendly”, and it has slower script execution times than those of eggPlant but it is free. ATRT TM is neither free nor as capable or user friendly as the others. Its strengths are that it includes requirements linkage to test steps and a flexible reporting capability. However its greatest weakness, and its ultimate downfall, is that the collaboration capabilities are very lacking and as such it would not make a viable selection for large software projects with many developers.

# Appendix A

## License Quotes For ATRT and eggPlant

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **quotE**  Date: March 19, 2014  Quote # 3203-175  Expiration Date: 4/14/2014 | | | |  | | |
|  | | | | | | |
| **Innovative Defense Technologies**  **4401 Wilson Blvd., Suite 810**  **Arlington, VA 22203**  **703-522-5561**  **Fax 703-807-0072** | | **To** | Antonio Ricco  Mission Solutions Engineering  121 Whittendale Drive  Moorestown, NJ 08057 | | | |
| **Salesperson** | **atrt License terms** | | | | **Payment terms** | **quote EXPIRATION date** |
| Durwin Bullock dbullock@idtus.com | Perpetual license, software new releases, and customer support via email.  After 12 months, additional software releases and support via email may be extended under a maintenance agreement at 25% of the license cost for each additional year. | | | | Due on receipt | 4/14/14 |
|  | | | | | | |
| **Item** | **description** | | | | **Quantity** | **line total** |
|  | **Software:**   * ATRT License | | | | 1  5  10 | $7,500.00  $37,500.00  $67,500.00 |
|  |  | | | |  |  |
|  |  | | | |  |  |
| This is a quotation on the item description named above. IDT is not required to collect sales tax in the state of Massachusetts.  The buyer agrees to file any required use tax returns and assumes responsibility for any sales tax and other charges assessed on IDT for buyer’s failure to do so.  To accept this quotation, sign here and return: | | | | | **Subtotal** |  |
|  |  |
| **Total** |  |
|  | | | | | |  |

Ordering Information

How to Complete your Purchase

**Purchase order**

Fax your purchase order printed on Company Letterhead to +1-703-807-0072, or mail it to:

Attn: Accounting

Innovative Defense Technologies, LLC

4401 Wilson Blvd., Suite 810

Arlington, VA 22203

**The following information is required with your order:**

Authorized approval for purchase (authorized signature, company letterhead, company stamp, purchase order number or any other references required by your purchase department)

IDT quote number

Total amount of order

Delivery and invoice address

Contact information, if required. See the section below.

IDT payment terms: NET 30 Days

**Thank you for your business!**

Above is a quote for 1, 5 and 10 licenses (seats). When purchasing 10 you get a 10% price discount. When you purchase a license you own the product. During the first 12 months you are entitled to all new releases and phone support. To continue getting this support after the first year cost 25% of the original cost. If you don’t need a maintenance agreement after the first year you still own the product and can continue using. If you have any questions don’t hesitate to contact me.

Regards,

Durwin Bullock

Director of Operations

Innovative Defense Technologies

4401 Wilson Blvd, Ste 810

Arlington, VA 22203

703-522-5561 (ph)

703-807-0072 (fax)

703-405-6690 (mobile)



[www.testplant.com](http://richard%2Eward@testplant.com/)

eggPlant Licensing

eggPlant is sold on a 12 month term basis which includes…

•   Full use to the latest version of eggPlant

•   Full support and maintenance

•   Free access to all new product releases

•   Full access to our online guides and training material

Terms of 24 months or greater are also available and include extended term discounts.

eggPlant can be purchased with different levels of functionality. This functionality allows licenses to be fixed or floating [eggPlant or eggPlant Team], develop or execute [Development or Execution] and finally the addition of our API [eggPlant Drive].

So to establish the licenses you require please choose…

1)     eggPlant or eggPlant Team License?

An eggPlant license can only be used on one computer (sometimes called a "node-locked license"), i.e. It can have multiple users however they all have to operate through the same machine. You may change the computer that a specific license is associated with, but this is a manual process through your eggPlant license portal, greenHouse.

An eggPlant Team license provides more flexibility by allowing a team of testers to share a license. The license can be used by any computer on your network, but only one tester may be using it at any given time. If you need multiple testers to be using eggPlant at the same time then you will need to purchase multiple eggPlant Team licenses.

2)     Development or Execution?

A "development" license allows a tester to use the full functionality of eggPlant. Most importantly this includes both creating and executing test scripts. An "execution" license allows a tester to execute existing eggPlant scripts, but they cannot develop scripts.

3)     eggPlant Drive?

eggPlant Drive provides you with an API that gives you full access to all the functionality of eggPlant. Everything from running an existing test script, to interactively executing specific commands on the system-under-test (e.g. screen touch). This allows you to fully integrate eggPlant into your wider testing, continuous integration or development environment. Please note that eggPlant Team licenses include eggPlant Drive.

List Price:

**eggPlant Team Development: $11,999**

**eggPlant Development: $6,999**

**eggPlant Execution: $3,499**

Kind regards,  
Richard Ward  
Chief Sales Officer

TestPlant Inc                                  T     [+1 415 426 3560](http://www.testplant.com)  
201 Spear Street                           M    [+1 415 481 1146](http://www.testplant.com/)

San Francisco, CA 94105             [richard.ward@testplant.com](mailto:richard.ward@testplant.com)         USA                        

# Appendix B

## Scenario Design

The high level scenario design is as follows:

* Launch a web browser
* Login to the CIWi web application
* Use CIWi to install a software package onto a remote host
* Verify install was successful
* Use CIWi to uninstall the package from the same host
* Verify uninstall was successful
* Logout of CIWi

The scenario is run twice for each tool in order to evaluate tool performance under success and failure conditions. The CIWi Java service running on the remote node is stopped in order to create a repeatable situation where the scenario reliably fails. This causes the CIWi web application to fail when it attempts to submit an install transaction, thus permitting the evaluation of tool failure detection, handling, and reporting capabilities.

A VNC server is started on the SUT host machine. Each tool performs the test using a VNC viewer to access the SUT desktop “screen”.

As already mentioned, **Jenkins**, is used to automate running the common scenario for each tool. This provides the benefits of being able to coordinate starting the scenario with one click along with the capture of scenario execution times. To this end the following discrete steps are created and used within Jenkins for each of the tool:

* Start VNC server on the SUT
* Start CIWi service
* Run scenario expecting success
* Stop CIWi service (to induce failure during next scenario run)
* Run scenario expecting failure
* Stop the VNC server on the SUT (to reset the SUT desktop)

## Scenario Steps

The following are the detailed steps of the common scenario used for evaluation of all tools.

* Launch Firefox
* Navigate to rhelserv1/ciwi/ciwi/
* On "CIWI Login" screen,
  + Type Username: ciwi
  + Type Password: asd123
  + Click "Login" or "type return"
* On the "Domain" window,
  + Select "Innovation Lab" from dropdown
  + Click "Set Domain"
* At "Main Menu" screen,
  + Click "Components
* On the "Components" screen,
  + Click "Modify" for the ADS component
* On the "Component Status" screen,
  + Click the tab "By Host"
  + Click the "Install" button for the rhel1 host
* On "Remote Login Credentials" screen,
  + For "Username", type "autotest"
  + For "Password", type "autotest"
  + For "Install as User", delete all contents of the text entry box
  + Click "Next"
* On "Installation Path" screen,
  + Type "/home/autotest/ciwi"
  + Click "Next"
* On "Site Specific Questions" screen
  + Click "Next"
* On "Physical Hosts" screen
  + Click "Next"
* On "Summary" screen
  + Click "Install"
* On the screen that pops up,
  + Verify "submitted"
  + Click "OK"
* On the "Components" screen,
  + Click the "Modify" button for the ADS component
* On the "Component Status" screen,
  + Click the tab "By Host"
  + Wait for up to 1 minute for the rhel1 "Status" column to display "Installed"
  + Click "Uninstall"
* On "Remote Login Credentials" screen,
  + For "Username", type "autotest"
  + For "Password", type "autotest"
  + For "Uninstall as User", delete all contents of the text entry box
  + Click "Next"
* On "Site Specific Questions" screen
  + Click "Next"
* On "Hosts" screen
  + Click "Next"
* On "Summary" screen
  + Click "Uninstall"
* On the screen that pops up,
  + Verify "submitted"
  + Click "OK"
* On the "Components" screen,
  + Click the "Modify" button for the ADS component
* On the "Component Status" screen,
  + Click the tab "By Host"
  + Wait for up to 1 minute for the rhel1 "Status" column to display "Uninstalled"
* Click "Logout"
  + Exit Firefox

From the above, each step is broken down into pre-defined subroutines to be implemented using each tool in order to evaluate test modularity mechanisms of the tools.

# Appendix C

## Implementation Notes

2.10.2014

\* created autotest account on rhel1

// add user

system-config-users

// add autotest to labroot group

usermod --gid 16777278 autotest

=====================

eggPlant

\* downloaded eggPlant functional (tgz containing rpm)

provided contact info for free trial license

sudo rpm --verbose --hash --install Downloads/eggPlant\_redhat/eggPlant14.01.rpm

Preparing... ########################################### [100%]

    1:eggPlant ########################################### [100%]

rpm -q --list eggPlant > NOTES/rpm.list.txt

This command gave a 3600+ line of files.

Got evaluation license from eggPlant, good for 2 weeks. This came in an email.

eggPlant email from richard.ward@testplant.com

Ran eggplant from the command line (it resides in /usr/local/bin/eggplant)

Running this for the first time brought up a license manager. Entered license key and left the username empty.

License: 5n5v-bg6f-ezhn-ylyy-hsft-qm

Subsequent runs of eggplant simply brings up eggplant and skips the license step.

A requirement of eggPlant is a vncserver.

Machine rhel1 happens to have tigervnc installed but it doesn't matter.

vncserver :3 -geometry 1900x1020

vnc\* and Xvnc were sftp'ed from rhel1 to rhel10, and placed in ~autotest/vnc.

Ran a vncserver :3 -geometry 1900x1020 on rhel10 and ran eggplant from rhel1.

=================

SIKULI

\* download sikuli jar

https://launchpad.net/sikuli/sikulix/1.0.1/+download/sikuli-setup.jar

\* download OpenCV

http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.8/opencv-2.4.8.zip/download

\* download tesseract (used version tesseract-3.02.02-5.mga4.i586.rpm)

http://rpmfind.net/linux/rpm2html/search.php?query=tesseractNeeded cmake for OpenCV for Sikuli (at a minimum)

Install docs: http://docs.opencv.org/doc/tutorials/introduction/linux\_install/linux\_install.html

Source tarball from: http://sourceforge.net/projects/opencvlibrary/files/latest/download (opencv-2.4.8.zip)

yum search all cmake

That command yielded cmake28.i686

yum install cmake28.i686

This created a cmake28 executable in /usr/bin

In ~autotest, extracted opencv-2.4.8.zip into ~autotest/opencv-2.4.8

cd ~autotest/opencv-2.4.8

mkdir release

cd release

cmake28 -D CMAKE\_BUILD\_TYPE=RELEASE \_D CMAKE\_INSTALL\_PREFIX=/usr/local ..

From release directory (above),

make

This took about 10+ minutes.

sudo make install

This added many files, and executables to /usr/local/bin/

Need tesseract-ocr for Sikuli

tesseract wiki: https://code.google.com/p/tesseract-ocr/

yum search failed - need to build from source

downloaded tarball from https://code.google.com/p/tesseract-ocr/downloads/detail?name=tesseract-ocr-3.02.02.tar.gz

compile docs: https://code.google.com/p/tesseract-ocr/wiki/Compiling

Docs say Leptonica is required to build tesseract

Tesseract 3.02 requires at least v1.69 of Leptonica.

Download leptonica source from: http://www.leptonica.org/source/leptonica-1.70.tar.gz

Leptonica build docs: http://tpgit.github.io/UnOfficialLeptDocs/leptonica/README.html#overview

Untar to ~/leptonica-1.70 and cd there

./configure

make

sudo make install [as root; this puts liblept.a into /usr/local/lib/

                and all the progs into /usr/local/bin/ ]

Build tesseract from source

cd ~/tesseract-ocr

./autogen.sh

./configure

sudo make install

Download tesseract data from: http://tesseract-ocr.googlecode.com/files/tesseract-ocr-3.02.eng.tar.gz

cd ~

tar xvf Downloads/tesseract-ocr-3.02.eng.tar.gz

cd /usr/local/share/tessdata

sudo cp ~/tesseract-ocr/tessdata/\* .

Start sikuli first time: // appears to finalize installation

cd ~/SikuliX

java -jar sikuli-setup.jar

click OK

Select 1 and 5 (Pack1 and tesseract OCR)

click Setup Now

click Yes to download prompt

exits to shell after downloads complete

Start sikuli:

cd ~/SikuliX

./runIDE <= automatically created by previous install step

IDE RUNS BUT CRASHES

SikuliX-1.0.1-SetupLog.txt says:

[debug (2/10/14 1:32:14 PM)] ResourceLoaderBasic: loadLib: Found: VisionProxy

[error (2/10/14 1:32:14 PM)] ResourceLoaderBasic: loadLib: Fatal Error 110: loading: libVisionProxy.so

[error (2/10/14 1:32:14 PM)] ResourceLoaderBasic: loadLib: Since native library was found, it might be a problem with needed dependent libraries

/home/autotest/SikuliX/libs/libVisionProxy.so: libopencv\_core.so.2.4: cannot open shared object file: No such file or directory

[error (2/10/14 1:32:14 PM)] Terminating SikuliX after a fatal error(110)! Sorry, but it makes no sense to continue!

If you do not have any idea about the error cause or solution, run again

with a Debug level of 3. You might paste the output to the Q&A board.

[debug (2/10/14 1:32:14 PM)] SikuliXFinal: cleanUp: 0

DEBUG:

ldd libs/libVisionProxy.so

ldd: warning: you do not have execution permission for `./libVisionProxy.so'

linux-gate.so.1 => (0x003a9000)

libopencv\_core.so.2.4 => not found

libopencv\_highgui.so.2.4 => not found

libopencv\_imgproc.so.2.4 => not found

CORRECTION: re-install sikuli with LD\_LIBRARY\_PATH set

export LD\_LIBRARY\_PATH=/usr/local/lib

mv ~/SikuliX/ ~/SikuliX.1

mkdir ~/SikuliX

cp ~/Downloads/sikuli-setup.jar ~/SikuliX

java -jar sikuli-setup.jar

success

added export of LD\_LIBRARY\_PATH to ~/SikuliX/runIDE script

Attempting to run from the local machine (rhel1):

./runIDE

First thing done was to go to File->Preferences

Set "Where to store images" to /home/autotest/Testing/Sikuli (created that directory manually)

In TextSearch and OCR section of window, clicked "allow searching for text" and "allow OCR"

Clicked "Save" and closed window using "X"

The IDE suggested that the currently running application should be restarted so restarted using

./runIDE

SUCCESS! on both hosts, not sure why previous attempt crashed - maybe because preferences were

not set??

############################

Setting up two nodes to run a common scenario from all tools.

Using rhel1 as host machine from which tests are run

Using rhel10 as system under test (SUT)

Using autotest user on host

Using nagios user on SUT

On host machine, for the autotest user, used ssh-keygen to create ~/.ssh/id\_rsa and ~/.ssh/id\_rsa.pub

On SUT machine, for the nagios user, added the contents of id\_rsa.pub to ~/.ssh/authorized\_keys

To execute a command remotely, from the host to the SUT,

ssh nagios@rhel10 'ls -l'

However we received the error:

reverse mapping checking getaddrinfo for rhel10.uird.local [192.168.195.110] failed - POSSIBLE BREAK-IN ATTEMPT!

Last login: Mon Feb 24 13:56:30 2014 from localhost.localdomain

Fixed this by adding the ip address of rhel10 to the /etc/hosts file in rhel1

Then Bill had the brilliant idea (not being facetious) to add 'sut' to the name in the /etc/hosts file

instead of using rhel10. This way it can change in the future.

### Started a vncserver on the SUT as follows

ssh nagios@rhel10 '/home/nagios/vnc/vncserver -geometry 1800x750 :3'

Needed to set the password for the vnc once.

Ran eggplant and edited the connection to use

user nagios

password nagios

### Running Jenkins

From the ~autotest/Downloads directory:

java -jar jenkins.war --httpPort=8080

8080 is an example, not a requirement. Then to use it, run Firefox and use the port you gave.

Note you can also use https --httpsPort=8080

Created ssh keys

################### Design the common scenario and configure the hosts as needed

February, 25.

The goal of the scenario will be:

\* login to the CIWi web application on rhelserv1

\* use CIWi to install an ADS demo package onto rhel1

\* verify install was successful

\* use CIWi to uninstall the ADS demo package

\* verify uninstall was successful

\* logout of CIWi

Then we will use Jenkins automate running each tool in a passing situation, and

a failure situation and gather metrics for the report from Jenkis data.

To create a situation where the scenario should fail, we will stop

the CIWi Java service. This will cause the CIWi web application to

fail when it attempts to submit an install transaction.

In summary, Jenkins will run:

\* start ciwi service

\* run eggplant scenario expecting success

\* stop ciwi service

\* run eggplant scenario expecting failure

\* start ciwi service

\* run sikuli scenario expecting success

\* stop ciwi service

\* run sikuli scenario expecting failure

\* start ciwi service

\* run ATRT scenario expecting success

\* stop ciwi service

\* run ATRT scenario expecting failure

Created list of generic steps to be performed for the scenario for

each tool (see NOTES/Scenario\_Steps.txt)

From that, we broke down the steps into subroutines by name (see NOTES/Scenario\_Design.txt)

The intent is to implement the scenario with each tool following this design

noting the development time for each tool for comparison in the paper.

################### Host configuration for eggPlant (and other tools)

\* CIWi web application runs on rhelserv1 (url = http://rhelserv1/ciwi/ciwi)

\* defined sut as rhel10 address in /etc/hosts on rhel1 and rhel10

\* using nagois account on sut

\* using autotest account on test host (rhel1)

\* setup ssh keys to allow remote commands from autotest@rhel1 to nagios@rhel10

  to allow starting vncserver on rhel10 from jenkins on rhel1 (and other stuff)

\* setup ssh keys to allow remote commands from wkraemer@rhel1 to wkraemer@rhelserv1

  to allow starting CIWi service from jenkins on rhel1

\* created projects in Jenkins to:

  \* start and stop the CIWi java service on rhelserv1

  \* start and stop vncserver on the sut (rhel10)

  \* run the eggplant scenario for success

  \* run the eggplant scenario for failure

################### Creating Scenarios with eggplant

eggPlant: Started at 1:30 PM February, 25.

eggPlant: Completed at 5:50 PM

Total: 4 hours

However, this included 30 to 45 minutes

of debugging the scenario because CIWi was left in a strange state

when we induced failure. The workaround for this was two fold:

\* detect the error in the eggplant scenario and close

  the error dialog box (click ok)

\* configure firefox to always run in private browing mode so

  that the CIWi session is reset when the browser closes.

Our intent is now to implement these features in the scenarios for

the other tools

NOTE/QUESTION: we made extensive use of OCR in the eggplant scenario.

eggPlant seems to often default to this type of image comparison. eggPlant

also captures a PNG file even though using OCR...??? is it searching

for a similar image and THEN doing OCR???

################### Host configuration for Sikuli

Sikuli does not automatically manage VNC connections like the other tools.

Sikuli is not currently installed on the sut (rhel10)

After much thought about how to manage this to simulate a real world

environment like this (sut and test host are different machines)

we decided on the following:

For test development:

\* manually open a vncviewer on the testhost (rhel1) connected to the sut

\* start sikuli on the test host (rhel1)

\* use Sikuli to manipulate the sut (rhel10) via the vncviewer window

\* NOTE: sikuli IDE displays outside of vnc - we found that this makes

  it easier to capture screen images for menus this way because

  the menus stay open in the vncviewer when we activate the sikuli

  IDE for capture.

For test execution:

\* ensure that xhost + is performed on the sut vncserver

  (added step to Jenkins project that starts vncserver: ssh )

\* run sikuli on test host with DISPLAY exported to sut.

  export DISPLAY=sut:3

  /home/autotest/SikuliX/runIDE -r /home/autotest/AutomatedTest/Scenario/sikuli/Scenario.sikuli

##### later, found this info to help with capuring menu images with sikuli:

http://baydin.com/blog/2010/06/5-sikuli-pitfalls-and-how-to-avoid-them/comment-page-1/

4. Having trouble with context-sensitive/popup menus

If you tried to follow along with the previous example, you likely ran into a problem when you tried to capture the “Control Panel” option in the start menu. After opening the start menu, switching focus back to the Sikuli IDE will cause the start menu to close, thwarting your effort to capture an image. Now, you could use PrintScreen while the start menu is open, paste the image into an image processor, and then use Sikuli to capture the image from the image processor, but thankfully, there’s a better way.

Sikuli installs hotkeys for common tasks like capturing an image (CTRL + SHIFT + 2 by default), and they don’t cause the current program to lose focus. So you can simply open the start menu/context-sensitive menu of your choice and use the hotkey to capture the screen. That way the menu won’t disappear in the process.

BUT - found this also didn't work well when running IDE in vnc (menus close)!!!!!!

################### Creating Scenarios with Sikuli

Sikuli: Started at 3:00 PM February, 26.

Sikuli: Stop at 5:00 PM February, 26

(not complete yet...not really even started!)

Subtotal: 2hrs

Note: about an hour of this was debugging strangeness we were

seeing with sikuli.

Sikuli: Continuing at 9:00 AM February, 27.

10:35 Sikilu IDE has issue - will not allow file save:

[error] SikuliIDE: Problem when trying to invoke menu action doSave

Error: null

Quit and restarted IDE, no apparent issues

Sikuli: Stop at 12:00 PM February, 27

Sikuli: continue at 1:00 PM February, 27

Sikuli: Completed at 1:45 PM February, 27

Subtotal: 3hrs 45 mins

Total time: 5hrs 45mins