# Findings/Lessons Learned

## 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

None of the tools caused an appreciable or even noticeable runtime impact on either the host or the system under test (SUT). However it must also be noted that neither machine was running any time critical processes so an evaluation must be performed on the target machines before a final decision is made.

## 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 VNC Server and VNC Client are used for access to the GUI environment. In summary,

* Automated GUI testing requires Virtual Network Computing 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

The following figure represents the architecture.

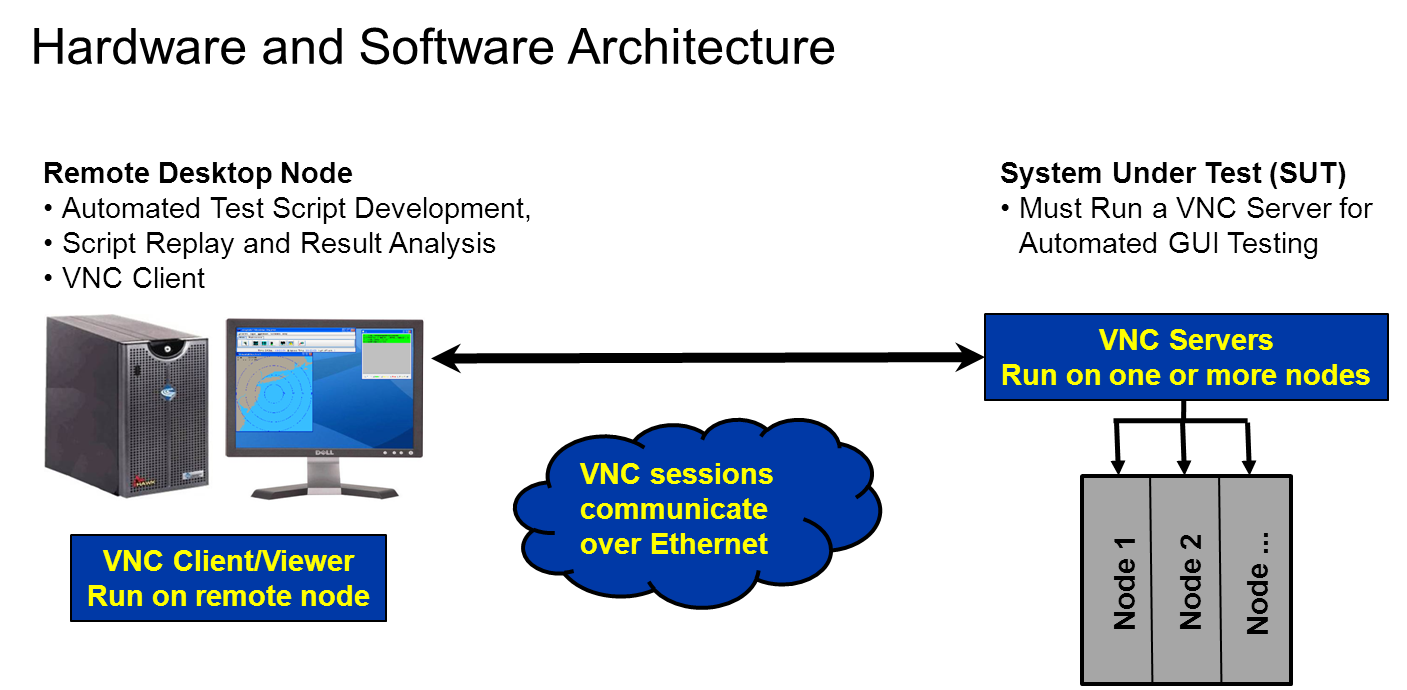


Figure 1 Hardware and Software Architecture Requirements

## 

## Scenario Results Explanation

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 a human, and ATRT performed about as fast as or slightly faster than a human operator. However the scenario execution times are meaningful only for comparison to the execution times for 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" systems, especially because time critical tests were not executed. Please see the section titled, **“Evaluation Scenario Definition”** for a detailed description of the scenario.

The remaining paragraphs in this section provide a high level rollup of all of the evaluated tools, followed by a detailed evaluation of each tool individually.

## Tool Comparison Criteria and Evaluation



Table 1 Comparison of Evaluation Criteria GUI Automated Testing Tools



Table 2 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 | 78 |
| Sikuli | 70 |

Table 3 Tool Comparisons Overall Score

### 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 the 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 that refer 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.

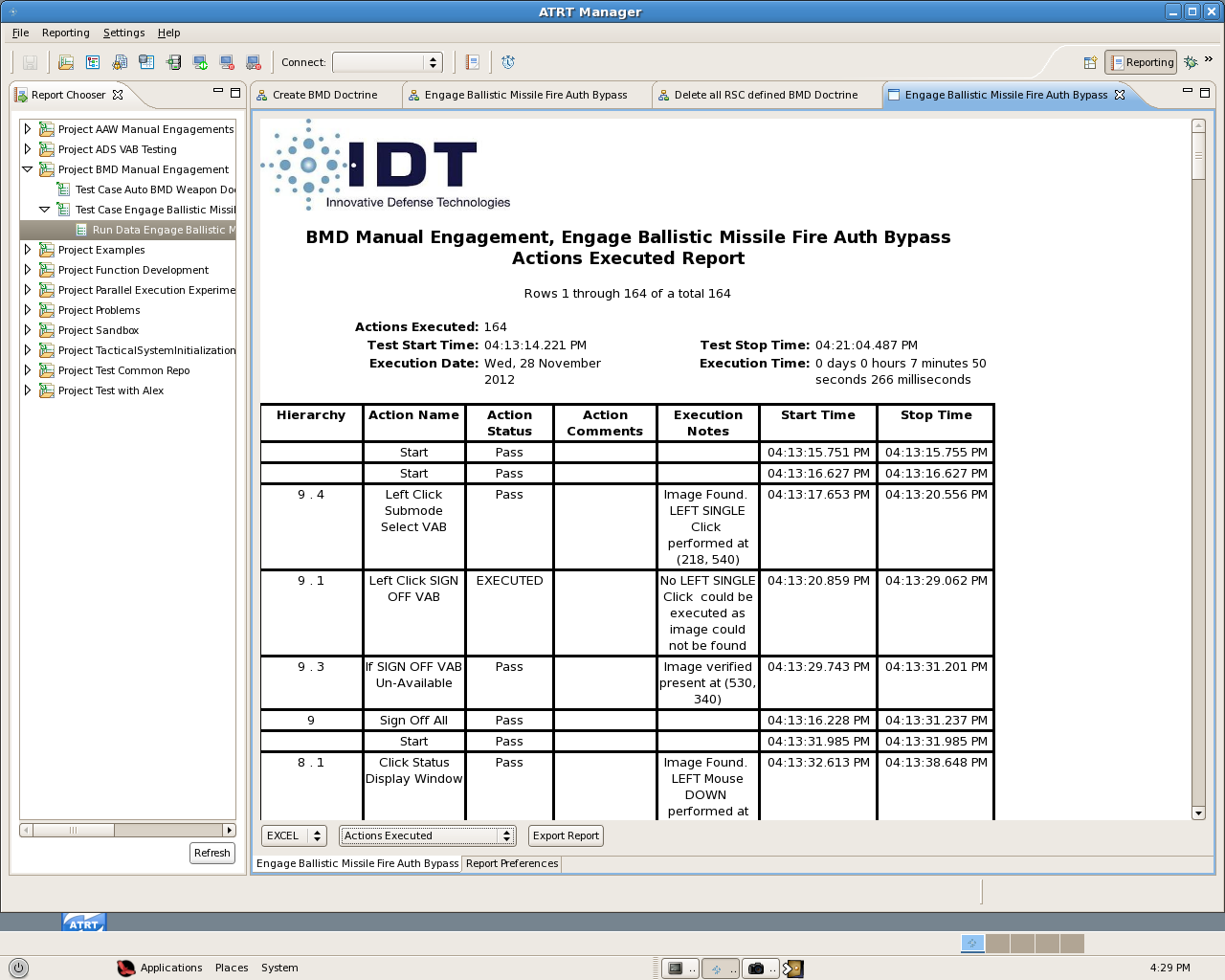


Figure 2 ATRT Example Report of Test Execution

##### 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.