### 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 expressive 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. We were unable to locate any third party SenseTalk reference books.

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”.

We experimented 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 we induced failure. 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 OCR scan of the image and suggests a file name based on the image textual content (if any) – a nice feature. Images are saved in the test suite file system “Images” directory as PNG files. Test scripts reference the images by file name without the “.png” extension, so file naming is important 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 scan 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. INSERT SOME PERFORMANCE INFO HERE? TBD

#### Collaboration

Artifacts created by EggPlant are stored as multiple 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, including 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.

There are multiple options for licensing of Eggplant. Please see Appendix A for details.