Wind River AutoTest:
Using GUI System Automation to Facilitate Testing

Education Services
Wind River Systems, Inc.
Ottawa, Ontario

James Mackenzie
V00737373
Work Term 1
Computer Engineering
jimack@uvic.ca

April 22, 2012

In partial fulfillment of the requirements of the
B.Eng. Degree

Supervisor's Approval: To be completed by Co-op Employer

I approve the release of this report to the University of Victoria for evaluation purposes only.
The report is to be considered (select one): □ NOT CONFIDENTIAL  □ CONFIDENTIAL

Signature: __________________________ Position: _____________ Date: ______________

Name (print): __________________________ E-Mail: __________________________ Fax #: ______________

If a report is deemed CONFIDENTIAL, a non-disclosure form signed by an evaluator will be faxed to the employer. The report will be destroyed following evaluation. If the report is NOT CONFIDENTIAL, it will be returned to the student following evaluation.
# Table of Contents

List of Figures......................................................................................................................i

Summary ...............................................................................................................................ii

Glossary of Terms and Symbols ..........................................................................................iii

1.0 - Introduction...................................................................................................................1
   1.1 - Wind River Systems and Wind River Education Services .................................1
   1.2 - The Testing Team and eggPlant Test Automation............................................2
   1.3 - The AutoTest System.........................................................................................3
   1.4 - Scope..................................................................................................................3

2.0 - Design of the AutoTest System..................................................................................4
   2.1 - Lab Indexer.........................................................................................................4
   2.2 - AutoTest Client.................................................................................................4
   2.3 - AutoTest Queue...............................................................................................6
   2.4 - eggPlant Engine...............................................................................................6
   2.4 - AutoTest Web Interface....................................................................................7

3.0 - Implementation of the AutoTest System..................................................................8
   3.1 - Selection of Technologies................................................................................8
   3.1.1 - eggPlant Testing Tool..................................................................................8
   3.1.2 - Python.........................................................................................................8
   3.1.3 - Qt and PySide ............................................................................................9
   3.1.4 - SQLite.........................................................................................................9
   3.1.5 - PHP...........................................................................................................10
   3.2 - Lab Indexer......................................................................................................10
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>AutoTest Client</td>
<td>10</td>
</tr>
<tr>
<td>3.4</td>
<td>AutoTest Queue</td>
<td>12</td>
</tr>
<tr>
<td>3.5</td>
<td>eggPlant Engine</td>
<td>13</td>
</tr>
<tr>
<td>3.6</td>
<td>AutoTest Web Interface</td>
<td>14</td>
</tr>
<tr>
<td>4.0</td>
<td>Conclusions</td>
<td>15</td>
</tr>
<tr>
<td>5.0</td>
<td>Recommendations</td>
<td>16</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>Cited References</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>General References</td>
<td></td>
<td>18</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2a</td>
<td>Example of a function written in SenseTalk that opens Wind River Workbench on an SUT</td>
</tr>
<tr>
<td>2.2a</td>
<td>Screenshot of the AutoTest Client GUI</td>
</tr>
<tr>
<td>2.4a</td>
<td>Screenshot of the AutoTest Web Interface</td>
</tr>
<tr>
<td>3.3a</td>
<td>Example of a function written in Python that uses sockets to determine an internal IP address of an SUT</td>
</tr>
<tr>
<td>3.3b</td>
<td>Example of a test file sent from the AutoTest Client to the AutoTest Queue</td>
</tr>
</tbody>
</table>
Summary

Wind River Systems is one of the world’s most established embedded software manufacturers. Wind River operating systems and development tools are used to power, design, and develop millions of electronic devices worldwide, ranging from microwaves and car stereos to NASA’s Mars rovers. The benefits afforded by training engineers and scientists how to effectively and efficiently design and develop new and existing products with Wind River software can therefore mean significant competitive advantages for a company. Wind River Education Services was created to address this need, and the training courses they provide allow engineers and scientists to gain hands-on experience with Wind River software before their skills are used to create and develop embedded devices of their own. These hands-on training courses are often developed by one engineer, and testing such a course can add days or weeks to the release of a course. In an effort to significantly lessen the time an engineer has to spend testing a course, an automated remote testing system was designed whereby an engineer can request the testing of a course using automated test cases. When a compatible test case has been designed, this automated testing system has the ability to test that test case on any computer in the world, without requiring any further user intervention. Since the only human interaction with the automated testing system comes when a set of tests are initiated, the user is free to continue their work on other projects while their automated test cases are running. It was determined that both the specific method of remote automated testing and the automated testing system, while still in their infancy, warrant further investigation and development. There are specific areas of improvement with regards to the automated testing system that can be addressed with a modicum of effort, the most vital being: the replacement of operating system specific tools and programs, the improvement of presenting test case results to the user, and the removal of insecure and dangerous code.
Glossary of Terms and Symbols

black-box testing
This is a case of software testing where the internal workings of a software program are not, and do not need to be, known by the person or machine testing it.

Comma-separated values (CSV)
A comma-separated values file is a spreadsheet storage format that stores and separates the data in each row by the use of commas inside a plain text file.

Command Line Interface (CLI)
The use of a text-based command line to input and output data to and from a program on a computer.

eggPlant
A proprietary black-box automated GUI testing tool.

File Transfer Protocol (FTP)
A standard protocol designed to transfer files over a TCP-based computer network (i.e., the Internet).

Graphical User Interface (GUI)
The method of controlling input and output on a computer or programs on a computer by interacting with on-screen graphical elements.

image
An exact copy of a computer's hard disk, often containing a fully-operational operating system.

Integrated Development Environment (IDE)
A software application that often provides a developer to not only write, edit, and highlight their code, but also test it within the application itself.

Joint Test Action Group (JTAG)
The common name for the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture. This architecture is the standardized method of testing printed circuit boards.

Optical Character Recognition (OCR)
The recognition of text by a computerized device with the end goal of converting said text into a computer-readable format.

PHP: Hypertext Preprocessor (PHP)
A server-side scripting language, known for its easy-to-learn syntax and wide install base.
PySide
A language binding that enables the direct use of the Qt GUI framework in the Python programming language.

Python
An immensely popular, easy-to-read, object-oriented, strongly-typed programming language.

Qt
A cross-platform GUI development and design framework. Allows a developer to design a GUI and deploy the same GUI on Windows, Mac, and Linux-based operating systems.

ReadyTech
An online training and image hosting website.

Revo
A cloud-based internal image hosting system, internal to Wind River Systems.

sanity test
A simple test or operation performed to ensure the device or program can function at its most basic level.

script
An automated series of instructions carried out in a specific order.

Secure Shell (SSH)
A network protocol designed to allow secure shell access to a machine. This enables a user to use a machine from anywhere in the world as if they were physically operating a shell on the machine itself.

SenseTalk
The scripting language used by the eggPlant GUI automation tool.

SIGINT
Short for signal interrupt, this is a signal sent to a process when the user wants to forcibly end its execution.

SQLite
A lightweight and popular relational database system using a weakly-typed version of the SQL database language.

Subversion
A version control system that allows multiple users to revise different pieces of a program at the same time.
System Under Test (SUT)
The system that the remote test will be executed on.

Universal Serial Bus (USB)
A connection standard used worldwide to transfer data and power to and from consumer electronics.

Virtual Network Computing (VNC)
A method of controlling another computer through a window that presents itself exactly as what a user would see if they were physically sitting in front of it.

Zip
A file compression standard.
1.0 - Introduction

In complex projects, reliable operating systems and detailed debugging tools can provide engineers with vital information about their systems, often preventing major issues before they affect operations in production environments. Familiarity with such advanced debugging tools and operating systems is often the difference between releasing stable products and unstable products.

As one of the world’s largest providers of embedded and mobile operating systems and development tools, Wind River Systems (hereafter referred to as Wind River) provides engineers and scientists with the tools required to develop and test software and hardware products. Wind River can also provide professionals who desire more experience with Wind River software training courses taught by experts in their respective field. These courses feature hands-on training with Wind River software to allow students to gain tangible experience before using Wind River software to develop their own projects. The hands-on training courses are designed to be used on custom-made Linux-based software images; these images are individually tailored to meet a set of specific requirements determined by the creator of the course.

This report will examine the method used to test a course’s software image before it is used in a teaching environment.

1.1 - Wind River Systems and Wind River Education Services

Wind River Systems specializes in embedded and mobile software, operating since 2009 as a wholly-owned subsidiary of Intel Corporation as a part of their Software and Services Group. The main products of Wind River include Simics, a full system simulator that allows developers to test multiple systems and devices on one workstation; Workbench, a program that combines an Eclipse-based integrated development environment with sophisticated JTAG capabilities; and VxWorks, one of the most popular real-time operating systems in the world [1], [2].

Wind River’s products are used worldwide to develop and control a number of diverse products, and therefore the people who use Wind River products must possess the technical knowledge to effectively use products such as VxWorks in an engineering setting.

Addressing the need to teach engineers how to properly use Wind River software, Wind River Education Services is a division of Wind River dedicated to teaching engineers and scientists how to design, develop, and deploy engineering projects using Wind River products. Using a worldwide network of instructors and diverse courses, Education Services can provide training to any number of professionals on every major Wind River product. These training courses make use of hands-on lessons that allow students to gain practical experience under the eye of an experienced instructor. These hands-on lessons require reliable and consistent experiences for each student.
To guarantee such experiences, Education Services uses Fedora 13-based Linux images on bootable USB drives, tailored specifically for each course. These USB drives can be used on any computer that supports the requirements of Fedora 13, any Wind River software required by the course, and the Universal Serial Bus Mass Storage Specification For Bootability, but are most often used on laptops supplied by Education Services [3]. Since these custom images are used to provide in-depth hands-on training to professionals who will invariably be using the knowledge they gain in the classroom in future engineering projects, both reliability and consistency of these USB-based Fedora images are vital.

1.2 - The Testing Team and eggPlant Test Automation

In January 2012, Wind River Education Services created a testing team with the sole purpose of testing USB images destined for course deployment. However, it was determined shortly after the testing team was formed that testing each course-bound image manually would be time-consuming and ultimately worthless, as even minor updates to images can alter both the performance and stability of an image and would therefore require retesting. An automated and maintainable solution was found in the eggPlant testing tool.

eggPlant is a black-box testing tool that allows a host machine to test another machine (the “system under test”, or SUT) via VNC according to a pre-written script. Written in a scripting language called SenseTalk, this script contains instructions to eggPlant for when to click on certain graphical user interface (GUI) elements. These instructions can be as simple as clicking on an image or a string of text, or as complex as starting a video recording of the SUT’s screen. For an example of a function written to open Wind River Workbench, see figure 1.2a. With eggPlant, an engineer only has to write a script that tells eggPlant how to manipulate the graphical user interface elements present on a booted image; if modifications are made to an image, an engineer need only modify the script to reflect the changes to an image and run the script to check the image.

An issue faced with using eggPlant as a testing platform is the preferred method of initiating a test. Since both a host machine and an SUT are required to test an image, there is a not insignificant amount of setup required. For example, the host machine must have both an up-to-
date version of the test script stored locally and the eggPlant testing tool must be installed and properly configured to connect to the SUT. As a single licence for eggPlant can cost more than $4,000 USD, an installation for every instructor or engineer who wants to test an image would not be feasible nor cost-effective [4]. A solution to a two-machine setup for every user who wants to run a test on an image is the AutoTest system.

1.3 - The AutoTest System

The AutoTest system consists of four subsystems: the AutoTest Queue, which accepts, readies, and queues incoming test requests; the eggPlant Engine, which executes individual tests on the SUT; the AutoTest Client, using which a user can request tests to be run on their image; and the Lab Indexer, which searches a Subversion repository for tests that can be run and saves that list into an SQLite database. There is also the AutoTest Web Interface, which is used to display in-depth test results on a web page. Operating together, these five subsystems form a solution to the issue of operating multiple individual two-machine setups, while only using one eggPlant license.

The AutoTest system is not only limited to testing computers on the internal Wind River network: both ReadyTech, an externally-hosted web-based image; and Revo, Wind River’s internally-hosted network of cloud computers, are also able to be tested using the AutoTest system.

1.4 - Scope

This report will examine the design and implementation of the AutoTest system and make specific recommendations on improvements that can be made to the system regarding user interactiveness, long-term maintenance, and compatibility. As the AutoTest system comprises five major parts, this report will examine the design and implementation of each, explaining and justifying the decisions that formed the AutoTest system in its current incarnation.


2.0 - Design of the AutoTest System

The AutoTest system was designed from its inception to be a lightweight and straightforward system that, if required, could be deployed and maintained by a very small group of engineers. To ensure the system is easy to maintain, the AutoTest system is made up of five constituent parts: the Lab Indexer, the AutoTest Client, the AutoTest Queue, the eggPlant Engine, and the AutoTest Web Interface.

2.1 - Lab Indexer

There are over 60 lessons in the Wind River Education Services catalogue, each with between one and ten hands-on labs. Due to the number of labs, not every lab for a lesson has a test case. The Lab Indexer was designed to catalogue what labs have test cases. When a course test script is created for use with the AutoTest system, an AutoTest directory is created in the lab’s folder in the Learning_Modules repository in the Wind River University’s Subversion system (the “WRU Subversion system”). The AutoTest folder stores an eggPlant script that details what and how the lab should be tested. Given a URL to a WRU Subversion code repository, the Lab Indexer recursively searches the Learning_Modules repository for any courses that have an AutoTest directory.

Many labs are stored on the same course image, and occasionally some or all of those labs will have the same set of requirements to test. The Lab Indexer is able to discern which labs have this ability by searching for a text file in each lab’s AutoTest directory called “common_testcase”. This text file contains a single line that points to the common test case in a WRU Subversion code repository.

If any labs are found that contain an AutoTest directory, the Lab Indexer creates an SQLite database and writes the course name, the official name of the lab, the name that the lab is stored under on an SUT, the URL to the course directory in the Learning_Modules Subversion repository, and, if one exists for the test case, the URL to the common test case in the Learning_Modules Subversion repository to the database.

2.2 - AutoTest Client

When a user wants to perform a test of a lab, they must use the AutoTest Client (the “client”) to submit a test request. The main requirements for the client were that the client would present the tests that can be run on the user's machine, allow the user to select the tests they want to run, and then send a request to the AutoTest Queue (see section 2.3) to initiate the tests. The client must be completely opaque with regards to the testing software and protocols used to implement the other subsystems of the AutoTest system. That is, the end user must not be able to discern what software is being used to test their computer, nor should the user be required to have any such knowledge.
Before the client starts, an attempt is made to check the availability of the AutoTest Queue on the local network. If a connection cannot be established, the client warns the user that a connection to the AutoTest Queue could not be established and shuts down. If the AutoTest Queue is available in the network and connectable, the client starts and the user is presented with a set of fields. The user is required to enter an email address (the test results will be sent to this email address when all tests are completed), select which labs to test, and enter any test notes that may be relevant to the test.

The status of tests waiting to be executed by the eggPlant Engine is displayed in the information area. The connection status shows “Connected” when the AutoTest Queue can be pinged, “Disconnected” when it cannot be, and “Completed” when all tests are completed. The estimated wait time is generated from previous runtimes of tests performed by the execution engine.

![Figure 2.2a: The AutoTest Client GUI. Note that a user is unable to submit a test request until at least one lab has been selected and an email address has been entered.](image)

When a user has entered all required information into the client, clicking “Submit Test Request” will contact the computer running the AutoTest Queue. The information sent to the AutoTest Queue included: the IP address of the SUT, the email address of the user, the test cases requested to be run, and system hardware specifications. The remainder of the information required to run the test is added when the information is processed by the AutoTest Queue.
If the user wants to cancel a test request, they can click on the button marked “Cancel Tests” in the client interface. In order to ensure that a user’s SUT cannot be left in an unknown state, it is impossible for a user to cancel a test request if the test is currently running. Once all tests are completed, the client presents the user with a message saying that all tests have been completed and their test results have been sent to the email address they provided.

2.3 - AutoTest Queue

The AutoTest Queue (the “queue”) is used to control the flow of test requests from the AutoTest Client to the eggPlant Engine while preserving the chronological order of the original requests. When a test request is received by the queue, the information is stored and then used to find the corresponding test cases in the Learning_Modules repository in the WRU Subversion system. The queue then downloads the required test cases in a temporary location on the machine hosting the queue. The Subversion revision numbers of the test cases are also collected and stored by the queue for later transmission to the eggPlant Engine for logging purposes.

When the required information for a test request has been located in the WRU Subversion system and downloaded and the test request becomes the oldest chronological test request (i.e., the next test to run in the queue), the queue initiates a request to the eggPlant Engine (see section 2.4). The queue sends a number of parameters to the eggPlant Engine: the test script name and path, the type of SUT (local to the Wind River network, a ReadyTech externally-hosted web-based image, or a Revo internally-hosted web-based image), and the SUT’s IP on the Wind River internal network.

During the running of a test case, the queue polls the eggPlant Engine to check the status of a test case. When all test cases from a given request have been completed, the queue sends an email to the user with a URL to the results of the test and communicates with the AutoTest Client to notify the user that the test request has been completed.

In the case of a serious eggPlant error (i.e., eggPlant not launching properly or not finding a correct license key), the queue exits gracefully, displaying a message on the server hosting the queue notifying the administrator what occurred, and ceases to run any more tests. The queue exits instead of trying to rerun a test because serious eggPlant errors often require significant modifications to eggPlant.

2.4 - eggPlant Engine

eggPlant is primarily used to test GUIs by engineers who are at their own workstations manually interacting with the eggPlant GUI. However, the idea of automated testing using the AutoTest framework is that no-one has to physically interact with the inner workings of the AutoTest system: having an engineer execute each test case request manually defeats the very notion of
automated testing. The eggPlant Engine was designed to automate the running of eggPlant scripts, performing the final steps to successfully initiate an automated test of a lab.

When called by the AutoTest Queue, the eggPlant Engine closes all running instances of eggPlant before attempting to launch the test - this avoids licence errors as running multiple instances of eggPlant on one machine would require a separate licence for each instance. Once all running instances of eggPlant have been closed, eggPlant is reopened using the command-line interface (CLI) and the testing of the requested lab is initiated.

Once the testing of a lab has been completed, the eggPlant Engine profiles the log that eggPlant creates every time a test is completed, analyzing the time between commands in the test case script. This information is useful if an error occurs during testing or during optimization of a test case. Along with the SUT’s system configuration, the eggPlant version, and detailed logging information, this log profile is stored on a remote server for the user to access at a later date. If there are any videos or screenshots that were created during the course of the test, they are included with the logging information. The URL to this information, along with the result of the test (i.e., whether the test case succeeded or failed), is sent back to the AutoTest Queue to be emailed to the user.

2.4 - AutoTest Web Interface

On completion of a test case, the user who initiated the automated testing of their SUT is emailed a URL to their individual test results when their tests are completed. The AutoTest Web Interface (the “web interface”) allows a user to go to a Wind River internal website and view every test stored in the results database generated by the eggPlant Engine. The information shown on the webpage can be seen in figure 2.4a: the test ID, start time, total execution time, test status, revision number, a URL to the test data, and any test notes.

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Start Time</th>
<th>Total Execution Time</th>
<th>Test Status</th>
<th>Revision Number</th>
<th>Test Data</th>
<th>Test Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2012-04-19 19:56:10</td>
<td>0:19:10</td>
<td>Failure</td>
<td>1475</td>
<td>ftp://wruser@128.224.145.151/AutoTestFTP1</td>
<td>Test #1</td>
</tr>
<tr>
<td>2</td>
<td>2012-04-19 22:44:14</td>
<td>0:12:37</td>
<td>Success</td>
<td>1477</td>
<td>ftp://wruser@128.224.145.151/AutoTestFTP2</td>
<td>Test #2</td>
</tr>
<tr>
<td>3</td>
<td>2012-04-20 07:27:48</td>
<td>0:22:26</td>
<td>Success</td>
<td>1488</td>
<td>ftp://wruser@128.224.145.151/AutoTestFTP3</td>
<td>Test #3</td>
</tr>
</tbody>
</table>

*Figure 2.4a: The AutoTest Web Interface displaying three separate test case executions*

As hosting the web interface on the same machine as the eggPlant Engine and the AutoTest Queue could become computationally taxing, the web interface has the ability to use either a locally-stored database or one that is hosted on a remote FTP server. To lessen stress on the network, if a remote database is being used a local copy will be made of the database and the database will only be updated on the request of a user.
3.0 - Implementation of the AutoTest System

The AutoTest system was envisioned as a system that could be used to test any lab in a course, regardless of SUT or host machine link speed to the Wind River internal network, physical location of either the host or SUT, and host or SUT operating system. With that in mind, the implementation of the AutoTest system focused on the use of lightweight and extensible languages that could easily be ported to other systems regardless of operating system.

3.1 - Selection of Technologies

The selection of technologies considered four main factors: licensing, portability between operating systems, cost, and feature set. As the AutoTest system is used in a commercial environment and may be used by other Wind River divisions in the future, the focus was on selecting languages, frameworks, and applications that would be as unrestrictive as possible. Portability between operating systems was also considered to be vital, as development work of the AutoTest would invariably be performed on computers running Windows 7, Fedora 13, Fedora 16, and Ubuntu 10.04 LTS operating systems. To that end, the cross-platform programming language Python was selected to be the programming language of choice for all desktop-based AutoTest subsystems, and PHP was selected for the AutoTest Web Interface.

3.1.1 - eggPlant Testing Tool

The cross-platform eggPlant automated testing tool was chosen over competitors Maveryx and Sikuli for its unparalleled ability to recognize both GUI elements and text using OCR systemwide while running an automated test [5]. Maveryx can only recognize GUI elements in Java-based programs, which would narrow the scope of automated testing severely, whereas Sikuli is in an early beta release and has no OCR functionality [6], [7]. While Maveryx must be run directly on an SUT, Sikuli can, like eggPlant, control an SUT remotely. However, this is only by opening a VNC connection window in a separate program on the host machine and commanding Sikuli to automate actions inside that window as this functionality is not built-in to Sikuli.

Despite Maveryx and Sikuli being licensed under licences that allow them to be freely released (GNU GPL 2.0 and MIT, respectively), the commercial tool eggPlant was chosen not only because of its more complete feature set, but also for its customer support and stability.

3.1.2 - Python

The CPython dialect of the Python programming language was selected over other languages such as C# or Java for the following reasons:
• **Availability of Extensions:** Python has thousands of freely-available extensions (called “modules”). These modules extend the functionality of Python, ranging from modules specifically for GUI development, to modules for cryptography and networking. A number of these modules were eventually selected for use in the AutoTest framework, such as Paramiko, an SSH client, and PySide (see section 3.1.3), used for GUI development of the AutoTest Client.

• **Portability:** As CPython is based on the C programming language, it can run on any operating system capable of running C. While C# can be developed and run on systems other than Windows using the Mono framework, some Windows-specific libraries like System.Management can be unsupported or unstable using Mono [8], [9].

• **Readability:** One of the key features of Python is its readability; indeed, it was designed to be a highly readable language from its conception. Python often uses English keywords where other programming languages would use punctuation - this is exemplified best by the implementation and readability of boolean operators in Python compared to other languages. Readability was considered in the language selection process due to the system being maintained by co-op education university students, as students often only work at co-op placements for four months at a time.

3.1.3 - Qt and PySide

There are a number of different GUI development frameworks available for use in Python, but the decision to use PySide and the Qt framework centred around both licensing and portability considerations. The Qt framework and PySide were released and are developed under licenses that allowed the AutoTest system to make use of them. Since the AutoTest system was never designed to be used outside of Wind River, the LGPL licenses for both Qt and PySide apply, and therefore enable Wind River to use Qt and PySide for the AutoTest system [10], [11].

The PySide GUI development module for Python was chosen over its main rival PyQt for three reasons: licensing, development activity, and future availability. PySide is maintained by Nokia, and is a cornerstone of their cross-platform development, whereas PyQt is developed by a small consulting company with no significant backing [12], [13].

3.1.4 - SQLite

The decision to use the SQLite as the default relational database management system instead of the more feature-complete MySQL database management system was due a number of reasons. One, a lightweight C-based library allows portability between operating systems. Two, SQLite has very strong interoperability with Python and PHP, as both languages have support for SQLite built-in. Three, SQLite is both publicly-released and has a proven record of use with web and desktop applications in thousands of commercial and private projects [14].
3.1.5 - PHP

The PHP server-side scripting language was selected over Java primarily because of the low amount of memory required to execute PHP code on a server and the built-in support for SQLite. The AutoTest system makes use of PHP only to read and display SQLite database information for the AutoTest Web Interface, and therefore there was no need to build a dedicated Java-based web application. While Java allows for much greater separation of layers (i.e., separating the web page that the user sees from the logic behind the Java applet), such separation was not required for the basic tasks required by the AutoTest Web Interface.

3.2 - Lab Indexer

Using a recursive searching algorithm, the Lab Indexer recurses into every folder in the Learning_Modules repository in the WRU Subversion system, using the dump feature of the Lynx CLI web browser to print out a raw text version of the website that is provided. When an AutoTest folder is located, the Lab Indexer uses Lynx to list the AutoTest folder; if a common test case is found, the Linux command wget is used to download a temporary version of the common_testcase file, which contains the URL to the common test case suite in the WRU Subversion system. The downloaded common_testcase file is then read, the URL inside is saved, and the temporary common_testcase file is deleted. When an AutoTest folder is found, the Lab Indexer will not recurse deeper into the directory tree; rather, it will recurse out of the lab folder and continue its search for other AutoTest directories in different lab folders.

After the Learning_Modules repository has been searched, the results of the search are saved in an SQLite database. The information written to the SQLite database is the course name, the official name of the lab, the name that the lab is stored under on an SUT, the URL to the course directory in the Learning_Modules Subversion repository, and, if one exists for the test case, the URL to the common test case in the Learning_Modules Subversion repository. The Lab Indexer uses the SQLite property UNIQUE to ensure that every lab folder name in the database is unique, and the SQLite property AUTOINCREMENT to automatically increment the sort ID of the database. The sort ID of the database is used as the primary key of the database.

3.3 - AutoTest Client

The AutoTest Client (the “client”) is the only AutoTest subsystem to make use of both Python and the PySide wrapper for the Qt GUI framework. The PySide wrapper provides a fully-featured implementation of Qt using the Python programming language. Using PySide, the client provides a user-friendly and intuitive GUI for the AutoTest subsystem.

Two actions are performed before the client becomes visible to the user. First, the client pings the server running the AutoTest Queue and eggPlant Engine three times to check its availability on the network. If the pings are returned within two seconds of their transmission, the client
deems the connection to be satisfactory and continues to load the client. If the pings are not returned, the client will display a warning to the user and exit. Second, and only if the client deems the connection to be satisfactory, the client will scan the /Labs directory of the SUT for the names of the labs stored on the SUT. After these names have been obtained, the client downloads the SQLite database created by the Lab Indexer, and uses the intersection feature found in the Python sets type to create a set of lab names that have test cases created for them in their respective AutoTest course folders in the WRU Subversion system. After this, the client loads the GUI and the lab names that have test cases in their AutoTest course folders are then displayed by the client in the “Available Labs” list in the GUI. See figure 2.2a in section 2.2 for a graphical representation.

When a user clicks the “Submit Test Request” button in the GUI with the appropriate fields filled out (i.e., an email address and at least one test case selected to test) and clicks OK on the message warning them to leave their computer until they receive an email signalling the end of the automated testing of their SUT, the system information of the SUT is collected. Presently, the client is only used to submit test requests on Linux-based systems as WRU only uses Fedora 13 and Fedora 16 based training images. For that reason, the hardware specifications of the SUT are obtained by reading the information stored in the SUT’s Linux-only /proc/meminfo and /proc/cpuinfo files. The code for determining the system information of an SUT in the AutoTest GUI has non-functional placeholders for other operating systems, and when other operating systems are supported by WRU, the client can be easily updated to collect the relevant system information for those operating systems. Taking an operating system agnostic approach, the internal IP address of the SUT is obtained by using the socket library in Python to connect to internal.wrs.com, a website that is accessible only inside the Wind River company network. Establishing the internal IP address with a website that is only accessible inside the Wind River network provides an additional check for Wind River internal network connectivity. Once the system information of the SUT has been collected, a text file that contains both the system information of the SUT and the selected labs is created in the /tmp directory of the SUT, as shown in figure 3.3b.

![Figure 3.3a: Using the socket library in Python 2.6 to obtain an internal IP address](image)

![Figure 3.4b: Example of a text file sent to the AutoTest Queue from the AutoTest Client](image)
On creation of the text file, the title of the file reads the SUT’s date and time, its IP, its status, and the type of SUT. The status of the SUT is a variable modified by the AutoTest Queue, and its use is explained in section 3.4. The type of SUT can be “normal”, “readytech”, or “revo”: “normal” is a machine local to the Wind River network; “readytech” is a ReadyTech externally-hosted web-based image; and “revo” is a Revo internally-hosted web-based image.

Using the FTP transfer protocol, this text file is transferred to a remote system hosting the AutoTest Queue and placed in a folder created by the AutoTest Queue. This folder is where all pending test requests are stored.

While a test case runs on the SUT, the client runs in the background as a minimized window. The client does this so it can check the status of the currently running test case. The client checks the status of the test case by reading a status file on the machine that runs the AutoTest Queue. When the SUT’s IP is found as having completed all its requested test cases, the client presents the user with a dialog box informing them that all test cases are complete. The dialog box appears regardless of how many test cases passed or failed. The client then resets the state of the GUI to the state that it was first opened in, so as to remain ready for any more test case requests any user may have.

3.4 - AutoTest Queue

The AutoTest Queue (the “queue”) stores and chronologically sorts test requests from the AutoTest Client. The queue is able to detect when the eggPlant Engine is running a test case and when it is not, and in the case it is not running a test case the queue will send the oldest chronological test case request to the eggPlant Engine.

Using an infinite loop, the queue scans a folder on the local computer (the “host”) for text files with the same structure as the one seen in figure 3.3b, calling the Python function `os.listdir()` every two seconds with the use of the Python method `sleep()`. This infinite loop can only be interrupted by a SIGINT event. When a new test request is found in the monitored directory, the queue calls the `prep_requests()` method, which checks for any text files submitted by the AutoTest Client that do not have any of their test cases downloaded to the local machine. The queue is able to quickly discern which test request has already had its test cases downloaded by a variable stored in the title of the test request. As an example, the text file seen in figure 3.3b has not had its test cases downloaded as the part of the title of the file after the SUT’s IP still reads “Not_Ready” - a text file that has had its test cases downloaded will read “Ready” instead. If the test case needs to have test cases downloaded from the WRU Subversion system, the queue checks for the location of the test case by reading the database created by the Lab Indexer using the sqlite3 module for Python. Once the test case is found in the database, it is downloaded to a temporary directory on the host, ready for the time the eggPlant Engine can execute the test case on the remote SUT.
When tests are running in the queue, the queue checks the status of the eggPlant Engine using the Python `poll()` method from the Python subprocess library. This method is called on the `subprocess.Popen` object created when the last test case was submitted to the eggPlant Engine by the queue. If the status returned by the `poll()` method is `NULL`, the queue waits in the infinite loop and polls the eggPlant Engine every two seconds, checking for and readying new test cases while it waits for the previous call to the eggPlant Engine to complete. When the call completes, the previously-created `subprocess.Popen` object will return a return code when the `poll()` method is called, signalling that the eggPlant Engine has completed its test case. The queue then calls the next test case, reading this information from the test request’s text file. As there may be multiple test cases to run for any given test request, the queue waits before all the test cases in a test request are completed before it updates its status file with the IP address of the last SUT that has completed its test request.

When a full test request has been completed, the queue removes the contents of the temporary folder that the test cases were downloaded to and sends an email to the user that submitted the test request. This email is sent by the built-in Python module `smtplib` using the `sendmail()` function. Once the email has been sent, the queue finds the oldest test request in the queue folder and runs the process of running each test case contained within its test request text file.

### 3.5 - eggPlant Engine

The eggPlant Engine (the “engine”) initiates eggPlant, running individual test cases as they are sent to it by the AutoTest Queue. The engine is unique in the AutoTest system as it is the only subsystem in the AutoTest system that interacts directly with eggPlant.

When it is called by the AutoTest Queue, the engine ensures that no eggPlant processes are currently running; this check ensures that there are no license issues that are incurred when the engine eventually attempts to execute the requested test case. If there are running eggPlant processes found, the engine will terminate them. The engine then readies eggPlant for the type of SUT (e.g., “Revo” or “ReadyTech”) the test case will be executed on by writing the SUT type to an eggPlant configuration file. The configuration file that must be modified is in reality a library that WRU designed as a part of the automated testing of labs; however, this library will not be examined further in this report as it is outside of the report’s scope. Once the configuration file has been modified, the engine executes the running of the test case on the remote SUT using the `os.system()` command. The engine checks whether or not the test case is still executing by calling the Linux command `ps` and checking the returned list of running processes for the eggPlant process every five seconds.

If the eggPlant process is not found in the list of running processes when the `ps` command is returns, the test case has finished its run. The results of the run are then inserted into the AutoTest run database based on the data in eggPlant’s run history CSV file. The engine reads the eggPlant run history CSV file using the `reader()` method in the built-in Python module `csv` and inserts the start time of the test, the total test time, the Subversion revision number of the test case, the
presumed URL of the test data on the WRU FTP server, and any user test notes into the AutoTest run database using the `execute()` and `commit()` functions in the `sqlite3` module. Once all test data has been written and committed to the AutoTest database, the test case result data listed above and any pictures or video captured by the test case are compressed into a ZIP file and uploaded to the WRU FTP server using the built-in Python functions `zipfile.ZipFile()` and `ftplib.storbinary()`.

### 3.6 - AutoTest Web Interface

The design specification for the AutoTest Web Interface (the “interface”) called for a simple, lightweight, web-based method of reading test results from the AutoTest results database created by the eggPlant Engine. To that end, PHP was used to read the test result data from the interface, with HTML and CSS used to create and style the webpage itself.

Before any data is loaded from the AutoTest results database, the database must be either loaded locally or downloaded from the computer running the eggPlant Engine. The interface checks a boolean variable called `$database_local` which dictates whether to use either a remote or local database. If `$database_local` is true, an SQLite database on the local machine is used. If `$database_local` is false, an FTP connection is made to the computer running the eggPlant Engine and the database is retrieved using the PHP function `ftp_get()`. An important point regarding this operation is that if a database is found locally, a new database will not be downloaded, regardless of the value of `$database_local`. However, if `$database_local` is true, an “Update” button is shown on the interface web page; clicking this button will download a new version of the AutoTest database and reload the web page.
4.0 - Conclusions

With the constant demands of ever-increasing quality standards and strict release deadlines, the AutoTest system fulfils a need to streamline and automate time-consuming sanity tests and general test cases. This automation allows engineers who write labs for Wind River Education Services to continue other work while their test cases are running. The present solution to testing labs is to have an engineer test each lab manually using the lab documents. These lab documents cover every detail of the execution of a lab, as they are meant to be used to teach newcomers and veterans alike how to use Wind River software in an engineering setting. Testing a lab completely to ensure it functions as expected often means that an engineer cannot forgo running even one step, as that step may mean the difference between a successful lab test and a failure.

As these test cases are merely lab documents that were originally designed to be used over a number of days, running these test cases often take hours to execute even by engineers experienced with Wind River software. Therefore, the time savings of automated testing are often significant as are engineers not required to perform the laborious manual testing of an image themselves. When engineers are able to offload their testing work onto a set of automated test cases, they can continue their work on the design and development of new products and new courses for use both inside Wind River and externally through Wind River Education Services.

Automated testing still has a long way to go before it becomes the standard for testing at Wind River Education Services: both error checking and total execution time of test cases are factors in its current state in Education Services as an early beta. These issues, however, are not issues with the AutoTest system itself as the AutoTest system does fulfil its stated goal of automating the execution of remote test cases. The type of automated testing the AutoTest system performs is also inherently difficult to perform, as small fluctuations in GUI layout and design can cause an automated test to fail.

At this point, automated test cases, and indeed computers themselves, are not capable of anything more than extremely basic reasoning while executing commands. This limitation will undoubtedly one day be solved, and when written with the utmost care and attention automated testing scripts can almost match the record of a human tester. It is, however, difficult to argue against the fact that while automated testing is still in its infancy, investigating any possible method to reduce the workload of engineers by even a small percentage in order to enable them to work on new and innovative projects is worth the effort.
5.0 - Recommendations

There are a number of recommendations that can be made to enhance the ability, user-friendliness, and performance of the AutoTest system. This report recommends four specific improvements be made to the AutoTest system.

On the completion of all requested tests, the AutoTest Client should present the user with the results of the requested test cases in full, or at the very least with the number of tests that passed or failed. This would allow an engineer to know at a glance the results of a test, instead of being forced to sift through their email messages or go to a website to find their specific test case results.

The eggPlant Engine makes use of the `os.system()` library to execute eggPlant scripts. To check whether a test case is running, the program `ps` is called using another call to the `os.system()` library. The `os.system()` library is deprecated and slated to be removed in a future version of Python; it has been strongly recommended to use the `subprocess.Popen()` library in its place not only because of its support going forward but also due to its more secure implementation of executing shell commands on a computer. Using the `subprocess.Popen()` library would also allow the eggPlant Engine to use the `poll()` method in place of the `ps` program to check whether or not eggPlant has completed executing a remote test case. Indeed, it is recommended to remove all Linux- and Unix-specific programs from the AutoTest system, so as to enhance its goal of not only making the SUT operating system agnostic, but also the host machine of the AutoTest Queue and eggPlant Engine as well.

Due to the implementation of an infinite loop in the AutoTest Queue, there is no way to end the execution of the AutoTest Queue without issuing a SIGINT via the CTRL-C command on the keyboard. In order to prevent unintended data loss and corruption on both the SUT and the host machine by doing this, it is recommended to implement a more graceful exit procedure such as allowing a user to press a predefined keyboard button such as the return key to quit the AutoTest Queue.

Finally, SUTs located inside the internal Wind River network and not running their images on either Revo or ReadyTech servers have been designated as “normal” SUTs throughout the AutoTest system. The notion of having a “normal” SUT should be removed entirely from the AutoTest system, as the current normal of running SUTs internal to the Wind River network may not be the norm in the future.

While none of these recommendations are of vital importance to the day-to-day operation of the AutoTest system, instituting the recommendations listed above will improve the stability, scalability, and overall user experience of the AutoTest system.
References

Cited References


General References

