INTERNSHIP REPORT
ON
POLARIS AND ALTAIR

A Report submitted to the Rajiv Gandhi University of Knowledge Technologies in fulfillment of the degree of Bachelor of Technology in Computer Science.

By

G Manjunath

R170244

Under the supervision of

E SUSMITHA

Assistant Professor

Computer Science Engineering

Idupulapaya, Vempalli,

Kadapa - 516 330, Andhra Pradesh, India

CERTIFICATE

This is to certify that the project work titled “Long Term Internship on POLARIS and ALTAIR” is a long internship submitted by G Manjunath (R170244) in the department of Computer Science and Engineering in partial fulfillment of requirements for the award of degree of Bachelor of Technology for the year 2022-2023 carried out the work under the supervision

E Susmitha,	N Satyanandaram,
Project Internal Guide,	Head of the Department,
CSE,	CSE,
RGUKT, R.K Valley.	RGUKT, R.K Valley.

DECLARATION

I G Manjunath hereby declare that this report entitled "Long Term Internship on Polaris and Altair" submitted by me under the guidance and supervision of E Susmitha is a bonafide work. I also declare that it has not been submitted previously in part or in full to this University or other University or Institution for the award of any degree or diploma.

Date: 30-04-2022 G Manjunath
Place: RK Valley (R170244)

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts success.

I am extremely grateful to our Director.Prof. K. SANDHYA RANI for fostering an excellent academic climate in our institution.

I also express my sincere gratitude to our respected Head of the Department Mr.N.SATYANANDARAM for his encouragement, overall guidance in viewing this project a good asset and effort in bringing out this project.

I would like to convey thanks to our guide at college Ms. E Susmitha for her guidance, encouragement, co-operation and kindness during the entire duration of the course and academics.

My sincere thanks to all the members who helped me directly and indirectly in the completion of project work. I express my profound gratitude to all our friends and family members for their encouragement.

CONTENTS

ABSTRACT

POLARIS

The Polaris is a cloud-based test management tool that assesses applications for security vulnerabilities

The Polaris allows users to manage tests and test schedules. Such test management tasks include updating test targets, scheduling automated tests, and modifying test schedules.

Altair

The Altair is also a cloud-based test management tool that assesses applications for security vulnerabilities

The Altair also allows users to manage tests and test schedules. Such test management tasks include updating test targets, scheduling automated tests, and modifying test schedules. Users can also produce reports that filter test results by various products or issues. Altair contains the combined features of Polaris and MSP(Managed Services Platform) supporting both CLI and GUI.

OVERVIEW

Polaris

Polaris provides a Web interface that enables the setup and scheduling of various types of tests. Once a test has begun, it can proceed through various states before it finally completes.

Altair

Altair provides a Web interface as well as Command Line Interface that enables the setup and scheduling of various types of tests. Once a test has begun, it can proceed through various states before it finally completes. Finally assessment results will be displayed on web page.

WORKING PROCESS OF MANUAL TESTING

Polaris and Altair

First of all we should create a target in the Polaris and Altair Platforms. A target represents the URL of a Web application when the system under test is a Web application. It represents the type of operating system when the system under the test is a mobile application. It represents the programming language type when the system under test is source code.

Targeted test performed by a security consultant on the customer's application/network to find potential security issues. So we schedule a test on that created target and it goes through a various steps to complete the test

And in between the various steps before completing the test, assessors actually go through the test scheduled i.e., a scheduled project and try to add the vulnerabilities/findings and based on that vulnerabilities/findings for a project, we will generate a PDF and docm reports and give them to the customers.

And we also show the vulnerabilities/findings in a project in their dashboard page of Polaris and Altair Portals. And we will document the required and necessary test cases for the new release features

WORKING PROCESS OF AUTOMATION TESTING

Polaris

As part of Automation Testing, we have 1 repo for Polaris Project

- test-ngp

test-ngp

test-ngp repo is used to automate the Polaris website/portal (the website we use to schedule and test the projects)

Altair

As part of Automation Testing, we have 1 repo for Altair Project

- test-altair

test-altair

test-altair repo is used to automate the Altair website/portal (the website we use to schedule and test the projects)

AUTOMATION STRATEGY

The test automation strategy consists of three major parts:

(i) Framework

(ii) Continuous Deployment

(iii) STS Integration.

Framework

The Test Harness for CIM Test Automation includes the Testing Framework (engine for executing and supporting tests), and our Test Library.

The Testing Framework includes the following components:

TestNG – the test execution runner Selenium

WebDriver – the core web browser automation library

SikuliX – the core desktop GUI automation library

Shelob – the Page/Element meta-framework for reusable test objects

Agent Framework – application-specific behavior abstraction framework between wiring and test cases

Together, these form the core of the GUI automation framework. The web application test automation support simplifies creating and maintaining UI tests against CIM, or any future web apps developed at Coverity. The GUI test automation support allows us to test any GUI application. Each product team will be responsible for wiring their products with the framework. Ownership of the framework will belong to the QA Automation Team.

The Test Library contains all automated tests. All tests will be written in Java and be executable within a developer's IDE. The Library will live in the same repository as the Framework. Ownership of this component will be assumed by the QA Automation Team.

Continuous Deployment

The Continuous Deployment (CD) requirements are:

- Tests to be executed automatically against the latest deployment of products when changes are made to the application (preferably via Hudson /Jenkins using the Maven deploy lifecycle). A Selenium Grid will be used for the browser clients. Other machines running applicable GUI desktop platforms will also be needed to quickly execute GUI tests. The STS farm will be used for this purpose.
- Due to the duration of automated tests, these will run after a build completes (rather than gating the build itself). These will be triggered on a schedule, likely daily. The tests will run on STS.
- CIM test deployment should include a reference DB that is built with the latest schema changes. This ensures that the testing database is always in a known good state that is reproducible.

STS Integration

The Test Harness can be used independently from STS/scenario. The core Test Harness code does not live in the scenario repository, and does not require any component of STS to execute tests. However, STS provides the backbone to the Coverity R&D organization's test infrastructure, so it is important for the Test Harness to integrate into that infrastructure. Basic requirements are:

- Test suites are automatically executed via STS mechanisms. A mechanism in the scenario repo will launch cimautomation test suites.
- Test results are provided via standard STSweb mechanisms. A STS reporter mechanism will push data to STSweb.

Automation Layers

There are several separate layers which comprise the automation environment (arranged from inner to outer layers):

(i) Test Driver (Selenium WebDriver, SikuliX, etc.)
(ii) Driver Wrapper (Shelob, ScreenRegion et al, etc.)
(iii) Wiring
(iv) Agent Framework
(v) Test Cases
(vi) TestNG
(vii) Maven Project
(viii) Continuous Deployment
(ix) SauceLabs Integration
(x) STS Integration

Test Driver: WebDriver

WebDriver (also know as Selenium 2), is the latest iteration of the popular open source browser automation framework. Previous versions of Selenium supplied a number of legacy tools - Selenium IDE to construct tests via a FireFox plugin, and Selenium RC which used a standalone server to perform proxy injection to send automation commands to the browser. While supported in Selenium 2, neither of these components are used in our environment. Our current implementation relies solely on the WebDriver and Grid components of Selenium 2.x. While not supported on as many browsers as Selenium 1, the main advantage of using Selenium 2.x is that it provides native access to the browser API. This allows finer control of the browser and better accommodates the latencies of testing modern AJAX web applications. In addition, Selenium 2.x has a new implementation of Grid which is directly compatible with the WebDriver component. This allows tests to be executed in parallel.

SauceLabs has support for Selenium 2 natively, but is generally behind the latest stable release. (Today, we host our own Selenium Grid as part of our STS Farm, rather than relying on SauceLabs.) At the time of writing this, the latest version of Selenium was 2.11. The rapid pace of Selenium development requires a means of decoupling direct dependencies against WebDriver. This is the role occupied by Shelob.

Driver Wrapper: Shelob

The function of Shelob is to provide test harness bindings to the AUT while abstracting away direct dependencies to the underlying WebDriver automation layer. This serves to isolate the rest of the test harness from implementation changes at the automation layer (WebDriver). This meta-framework allows construction of Page and Element objects via inheritance.

Wiring

The current CC Davis wiring layer is structured as follows:

```
src/main/java/com/coverity/automation/
|
|-- PRODUCT
|   `-- wiring
|       |-- core
|       |   |-- elements
|       |   `-- ...
|       `-- ...
`-- ...
```

The organization of the wiring directory largely reflects the organization of the target application itself and our Test Plan.

Agent Framework

The Agent framework also lives in cimautomation's common package. The structure of this area of cimautomation is as follows:

```
src/main/java/com/coverity/automation/common/agents/
|-- exception
|-- models
`-- ...
```

The fundamental building blocks of the Agent framework are found in the above subpackages. Agents for specific products will be in automation/PRODUCT /agents directory.

Test Cases

The cimautomation Davis test case packages are organized in this way:

```
src/main/java/com/coverity/ces/webdriver/cases/
|-- common  (utilities, base classes, etc.)
|-- coordination
|   `-- remoteconfig
|-- crud
|   |-- configuration
|   `-- projectsissues
|-- e2e
|   |-- configuration
|   `-- projectsstreams
|-- functional
|   |-- configuration
|   `-- projectsissues
|-- infrastructure
|-- performance  (tests used to help optimize application usage)
|-- regressions
|-- scalability  (tests used to gauge behavior with large numbers of objects)
|-- security
|-- smoke
|   |-- configuration
|   |-- helpmenu
|   |-- projectsissues
|   `-- usermenu
`-- validation
    |-- configuration
    |-- helpmenu
    |-- projectsissues
    `-- usermenu
```

In order to minimize the feedback loop for CIM developers, test cases will be divided into several test categories as can be seen in the structure above. These categories will be run as test suites via integration with STS.

The smoke tests may also be executed separately via integration with the Jenkins build system. The goal is provide a "fail fast" mechanism to inform developers of application breakages. While developers will have the ability to run tests directly through their IDE (Eclipse/Netbeans), it is expected that back-end changes not directly associated with the UI can be caught within a reasonably short period of time from when a change was committed to version control, to when these smoke tests report their results.

TestNG

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use, such as:

- Annotations.
- Run your tests in arbitrarily big thread pools with various policies available (all methods in their own thread, one thread per test class, etc...).
- Test that your code is multithread safe.
- Flexible test configuration.
- Support for data-driven testing (with @DataProvider).
- Support for parameters.
- Powerful execution model (no more TestSuite).
- Supported by a variety of tools and plug-ins (Eclipse, IDEA, Maven, etc...).
- Embeds BeanShell for further flexibility.
- Default JDK functions for runtime and logging (no dependencies).
- Dependent methods for application server testing.

TestNG is designed to cover all categories of tests: unit, functional, end-to-end, integration, etc...

Maven Project

To enable developers to get up-and-running as quickly as possible, the cimautomation project has been fully Mavenized and should allow developers to import directly into Eclipse

Continuous Deployment

This component has the following dependencies :

- design and implementation of Test Parameterization
- a reference database schema and seed data required for initial smoke testing
- a CD staging environment for CIM
- possible changes to CIM to support automation of elements currently unresolvable via Xpath
- a client-side wait mechanism for AJAX calls

SauceLabs Integration

We were using SauceLabs for running our suites on STS since late 2012. But as of April, 2013, we have decided to begin moving much of our suite runs into our local Selenium Grid housed in our STS farm. This will benefit us in multiple ways:

- We have full control over the infrastructure.
- We save a pile of money, since SauceLabs charges by the minute for test runs.
- Our tests run far faster, since they have far less network latency to deal with. (Earlier experiments showed that test runs over SauceLabs were very sensitive to network latency, on the order of about 20-25% slower for every millisecond of latency.)

STS Integration

STS integration of cimautomation involves various pieces.

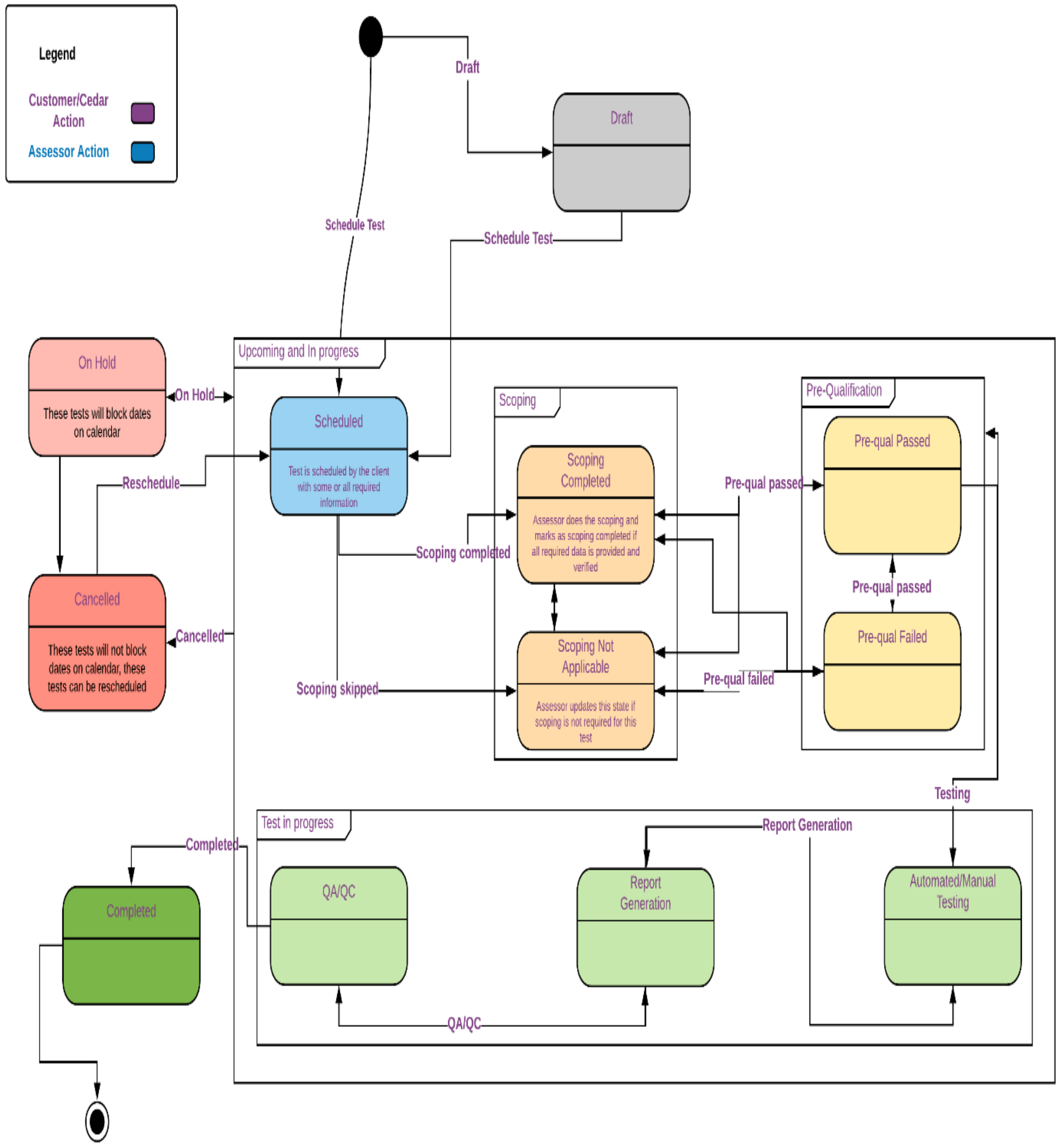
STS Reporter – The stsreporter module used with cimautomation is a Java component that is used to report test results to STSweb. (This is not to be confused with the Ruby/Perl STS Reporter used by other STS components.) This stsreporter repository can be found here: **git.coverity.com:/home/git /stsreporter.git**

Scenario -- The scenario repository contains traditional STS tests. We use directories inside, such as scenario/tests/cim/smoke to kick off cimautomation test suite runs.

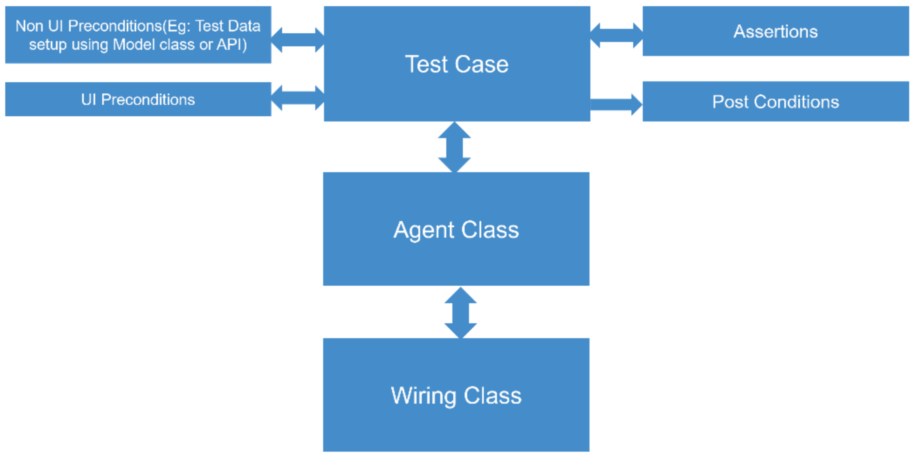
SauceLabs -- The browsers used are all living in SauceLabs. A proxy is enabled to allow the communication between the internal machine running cimautomation tests, and SauceLabs. Ideally, this will be improved to use a less manual mechanism for launching at some future time. This proxy may need to be restarted at times:

BLOCK DIAGRAM

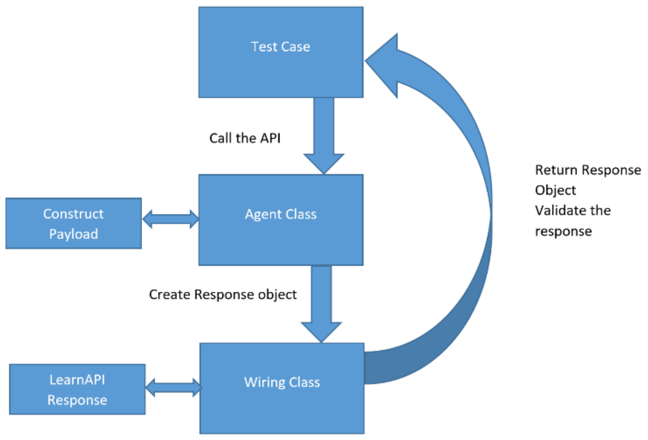
Polair and Altair



UI TEST WORKFLOW



API TEST WORKFLOW



CONCLUSION

Polaris

- Polaris is very useful project to know the vulnerabilities of a particular project.
- It shows the vulnerabilities in project.
- It shows the severity of particular issue in the project.

Altair

- Altair is also very useful project to know the vulnerabilities of a particular project.
- It shows more vulnerabilities in the testing project compared to Polaris.
- It shows various details about issues in the project like Issue Type, Issue Severity and possible fixes etc.

REFERENCES

- https://www.selenium.dev/documentation/webdriver/
- http://sikulix.com/
- https://testng.org/doc/
- https://saucelabs.com/
- https://maven.apache.org/guides/