2012

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

A COMPARISON OF AUTOMATED SOFTWARE TESTING TOOLS

Nancy Bordelon

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2012

Approved by
Advisory Committee

Devon Simmonds, Computer Science, Committee Chair

Bryan Reinicke, Information Systems

Laurie Patterson, Computer Science

Jeremy Noble, Manager of Information Systems, ATMC

Abstract

The aim of this research paper is to evaluate and compare three automated software testing tools to determine their usability and effectiveness. Software testing is a crucial part of software development and the process of automating software testing is vital to its success. Currently, complex systems are being built and testing throughout the software development cycle is pertinent to the success of the software. To enable comparison of the automated testing tools, a multi-partitioned metric suite was developed to guide the tool evaluation process. A selected web application was first manually tested; then tested using the three automated testing tools. Automated testing included the development of scripts that not only saves time and resources when applications are updated, but also speeds up the process of testing when regression testing is necessary. An important contribution of this research is the development of the metric suite that facilitates comparison and selection of a desired testing tool for automated testing. Inferences, implications and results are presented and discussed.

List of Tables

List of Figures

Table of Contents

Contents

Chapter 1: Introduction

The American novelist and historian Edward Eggleston once wrote that *"persistent people begin their success where others end in failure"* [29]. Software testing is an area of software development where persistence is essential. Software testing is the process of assessing software quality by using the software with applicable test cases to determine if proposed software requirements are being satisfied [2]. Testing is a fundamental aspect of software engineering, but it is a practice that too often is easily forgotten in today's fast-paced Web application development culture [18]. Testing applications thoroughly and efficiently is necessary for deployment when wanting to retain existing customers and also draw in new customers. Testing is important because software reliability is defined using testing and approximately fifty percent of the software development budget for software projects is spent on testing [7]. Software testing is labor intensive and expensive; therefore, there is a need to reduce human testing [34]. Software testing is necessary because errors are often introduced into software inadvertently as it is designed and constructed [28]. Software has become even more complex today, which means there are more lines of code, and more thorough testing that needs to be done.

When software testing is not completed as thoroughly as needed, the effects can be devastating to a company since there is typically a great financial cost to a company if bugs are found by users after updates are released. Indeed, the reliability of the software is potentially threatened every time bugs are released in software. William Howden, a professor from University of California at San Diego, wrote that "*testing is the unavoidable part of any responsible effort to develop a software system"* [28]. The Systems Development Life Cycle (SDLC) includes: project planning, analysis, design,

implementation, testing and support [30]. Modern approaches to SDLC encourage an iterative approach rather than a linear approach. As such, testing is an ongoing activity that occurs throughout the life of the project. However, systems testing and integration testing are also done just before the software is deployed. It is therefore clear that testing permeates the entire SDLC.

Testing is also important because software is not only pervasive but also often integrated within multiple systems. For example, in the medical field, despite advancements in technology, medication errors continue to cause patient harm and thousands of deaths annually [35]. One hospital successfully integrated two standalone technologies to improve medication administration, thus reducing medication errors, simplifying nursing workflow, and involving pharmacists in checking infusion rates of intravenous medications [35]. While testing of individual systems is important, testing integrated systems is even more important.

Given the repetitive and labor intensive nature of software testing, finding the appropriate tool support is imperative if software testing is to become a mature discipline. Several software testing tools are currently available and it can be assumed that the quality, maintainability, testability and stability of the software are improved by using testing tools [32]. These tools assist software engineers in increasing the quality of the software by automating mechanical aspects of the software-testing task [32]. The problem is that when modifications are made on applications, if there is no process in place to perform automatic testing on applications, then the time spent on testing may be too great. If automatic scripts are written, it will save time and resources when applications are modified in the future and regression testing becomes necessary. In the past, there have been many IT projects that companies invest in. For example, enterprise

resource planning, Year 2000 projects, or online analytical processing that companies have paid high dollar for in expectation of very thorough testing and as such, they expect to get thoroughly tested software [33]. There is, of course, a cost to using automated testing. Developing test scripts that are readable, maintainable, and reliable may be just as challenging as developing the system that is being tested, because the test scripts have to stay in perfect harmony with the application they are testing [33]. Once the test scripts are developed and associated test cases, they may be used repeatedly and save both time and resources. With the proliferation of software testing tools, it may be difficult to determine which automated software testing tool is best to use to achieve the goal of efficiently testing software.

1.1 Proposed Solution

The overall goal of this research is to evaluate and compare several automated software testing tools to determine their usability and effectiveness. To accomplish this goal this research will:

1. Research and select a set of tools to be evaluated
2. Develop a metric suite to be used to evaluate the tools
3. Select target application to be tested
4. Document time spent manually testing the application and record data
5. Perform a feature assessment for the tools with the goal of ranking the tools based on their features and identifying the ideal feature set for an optimal tool
6. Test the target applications using the selected automated testing tools and gather resulting data
7. Evaluate and interpret results and the tools

8. Draw inferences and make recommendations

1.2 Outline of the Paper

In the next chapter the background and related work will be discussed. The methodology including the selected tools and evaluation metrics is presented in chapter 3. The results are presented in chapter 4. Chapter 5 presents the discussion and lessons learned. Finally, conclusion and future work is discussed in chapter 6.

Chapter 2: Background and Related Work

2.1 Importance of Software Testing and Quality Assurance

Many research articles indicate that software testing is important for many reasons. One reason is that software testing detects the defects and minimizes the risk associated with residual defects of software [32]. Quality assurance (QA) has become a critical factor in the successful development of software systems that holds the key to customer satisfaction. QA has a direct impact on the cost and the scheduling of a project [1]. The QA process provides adequate assurance that the software and processes in the product life cycle conform to their specific requirements [9]. QA is usually a separate entity, which functions independently of the development area of a company. QA people on a software development team are typically responsible for defining the process and monitoring the details of execution [9]. QA touches all stages of a software project. It typically requires careful capture and control of many artifacts, as well as strict version management [9].

The process of automating software testing is crucial. Software testing does not appear to be keeping up with the pace of the software code being written. The number of lines of code per day per programmer appears to remain relatively fixed [10]. However, the power of each line of code has increased, which allows complex systems to be built. While Moore's Law doubles the computing power every 18 months, software code size tends to double every seven years [10].

Testing is an essential activity in software engineering [21]. It has been found that a roadmap can be laid out, which can identify destinations as a set of four ultimate and unachievable goals called 'dreams' [21]. One of the 'challenges' of the roadmap is

how much testing should be done. The human factor is a crucial resource for software testing. The testers' skill, commitment, and motivation can make a significant difference between a successful or an ineffective test process [21]. Lastly, the hope of this research is to show more effective ways to test software to reduce the 'challenge' on the roadmap.

2.2 Terminology

It is important to have a basic understanding of terms used in this research. A **test oracle** is a mechanism to evaluate the actual results of a test case as "pass" or "no pass". An oracle produces an expected result for an input and uses a comparator to verify the actual results against the expected results [4]. Another term examined in this paper is **class-based testing**. This is a process of operating a class under specified conditions, observing or recording the results, and making an evaluation of some aspect of the class [7]. **Software testing** is the execution of code using combinations of input and state selected to reveal bugs. The test input normally comes from test cases, which specify the state of the code being tested and its environment, the test inputs or conditions, and the expected results. A **test suite** is a collection of test cases, typically related by a testing goal or implementation dependency. A **test run** is an execution of a test suite with its results. **Coverage** is the percentage of elements required by a test strategy that have been traversed by a given test suite. **Regression testing** occurs when tests are rerun to ensure that the system does not regress after a change. In other words, the system passes all the tests it did before the change [7]. Another important term in software testing is **SCM** (software configuration management), which is an easy way to see changes made to code and also assist in testing. Developers work on a code line and they leave behind a trail of clues that can reveal what parts of the code have been modified, when, how, and by whom [11]. Test plans and automated test suites provide some assurance that the features

that a software product is designed to provide are present and operate as intended. Throughout the life of the product, they are also invaluable in guarding against regression. Another advantage of SCM systems is the ability to group a set of file changes and submit them as a single entity. These groups are referred to as changesets or changelists [11]. Another important term when it comes to web based testing is **XPath** [46], which is used to navigate through elements and attributes in an XML document. The XPath will be used extensively in testing using the Janova tool.

2.3 Software Testing Strategies

There are two main approaches to generating test cases: specification-based and implementation-based (or code-based). Specification-based testing, also known as black box or functional testing focuses on the input/output behavior or functionality of a component [7]. This technique treats the program under test as a 'black box' where no knowledge about the implementation is assumed [42]. Code-based, also known as white box, generates a test suite based on the source code of the program [34]. This research paper is to perform specification-based testing on an application with three different tools and compare the way tests are conducted from all three and also compare the results.

2. 3. 1 Black Box

This type of testing looks at functionality of the application. Software can have different paths to get to a definite endpath. Multiple representations of items also generally mean multiple execution paths. Two issues of complexity to consider are: the number of different execution states software can go through, and the issue of software concurrency. Software concurrency is sometimes used to make software faster or get done with its work sooner and concurrency is sometimes used to make software do more

work over the same interval where speed is secondary to capacity [41]. Concurrency is becoming more prevalent at many levels. Attempting to black box integrations of poor-quality components (the traditional 'big bang' technique) has always been ineffective, but large systems make it exponentially worse. Dependencies among units must be controlled to make integration quality truly feasible [10]. Research has been found to show that there are building blocks for test guidance algorithms, the step evaluation, the state evaluation and the evaluation order [12]. In black box testing, the coverage of the specified behavior of an application is measured. Applications sometimes use third party components like COM (component object model) or CORBA (common object request broker architecture). When applications use these, software developers, testers, and users are not always aware of the complex relationships that are created when functionality comes from an external component. Since external components can have bugs, starting to observe data being passed to third party components and the return values of function calls can reduce the time needed to track down application bugs [26].

2. 3. 2 White Box

Another technique in testing is white box testing, which is testing the code behind the software to ensure the requirements of the software are met. One way of creating an automated software testing tool is to create test cases for specific three-variable functions [2]. Functional testing techniques are limited by the nature of the programming problem and the number of inputs into the system. Suitable programs for functional testing must have a relatively small number of independent inputs. Functional testing techniques allow suitable means for the systematic generation and validation for certain programming problems [2]. White box testing includes creating test cases and then test suites to assist in running multiple test cases at once. If two test cases test the same features of a unit, if

one test case is updated correctly and the other is not, one test may fail while the other passes. This makes the test results ambiguous [3].

2. 3. 3 Testing Web Based Applications

It has been found that in developing testing techniques for web applications, it has become a necessity to control quality attributes and it is much more complicated to test web applications over classical software mainly due to the nature of the web applications [17]. Since there is not a single model to represent an entire web application, there can be a single analysis model which models the three poles of the web application. These three poles are: the client side pages navigated by the user, the server side programs executed at runtime, and the architectural environment hosting the application [17].

Testing is often last in developers' minds when pressured to deliver something, before the competition jumps on it, the market changes, or there are no funds [18]. There is a way to test called 'test-first' programming that means to write a unit test for a new piece of functionality before the new code is written [18]. This can be used when servlets need to be tested, which can be extremely awkward to test. Servlets are awkward to test because they are used as a component in a specific environment – the servlet container [18].

2. 3. 4 Other Types of Software Testing Strategies

Other strategies include object-oriented testing methods such as fault-based testing, and scenario-based testing [28]. There is also system testing and component testing. The phases in system testing are integration, release, and performance testing [31]. The strategies are basically different approaches to testing software depending on the needs of the organization.

2.4 Automated Testing

Automatic testing speeds up the process of testing. One reason automatic testing is important is it ensures that software is reliable, especially when updates are made. To assist in this concept, a software testing workbench is used, which is an integrated set of tools to support the testing process [31]. Another way to ensure software reliability is using test redundancy detection. This reduces test maintenance costs and also ensures the integrity of test suites [3]. It has been shown that once test cases are written, it is a good idea to update the test cases when changes are made to the software. If test suites being updated (test maintenance) are not conducted carefully, the integrity of the test suites being used will be questioned [3]. Steps in a test suite can be almost completely automated using existing tools like JUnit and MuJava and two new tools to generate compatible test cases [5]. Research has found that an experiment was done where the suitability of the tool for generating high-quality test cases was checked [5]. The tool used two groups of students with the same academic level, some of whom used the java tool, while others wrote test cases manually. In the experiment, the students using the tool obtained much better results than the students who did not [5].

2.5 Software Testing Tools

The tools summarized in this section are a select few that are used when performing automatic testing. For each, the following will be reviewed and compared: overview, history, automated progress, features, who uses the tool, and where the tool has been used. Table 17 shows an overview of the tools used in testing software applications.

2. 5. 1 Ranorex

This is a cost-effective and comprehensive tool used for automatic testing. This is a better alternative to conventional testing tools because it tests applications from a user's perspective, using standard programming techniques and common languages such as C# and VB.net. It does not require you to learn a scripting language, because it is written in pure .net code. You can use any one of the three languages, C#, VB.net and Iron Python. It is used by hundreds of enterprise and commercial software companies around the world [37]. The simulation tools such as this can have similar problems to the record and playback methods, as the tests are often tightly coupled to the code, and both methods still rely heavily on expertise to create the correct tests to ensure full coverage [36]. Ranorex is based on XPath, which is a very good way to find certain elements in a web based application. It is a pure .net API, which is very different from other tools which sit on an API [37]. Future plans for this tool involve creating an open and documented interface for the users to write their own plug-ins, which provides the maximum of object recognition for their own applications. Below are some of the features in the tool [37]:

- Image-based recognition
- Contains Record-Replay functionality (Ranorex Recorder)
- Provides seamless integration for 32 and 64 bit operating systems
- Built on the .NET Framework
- Offers a flexible and standard test automation interface
- The test automation modules can be created as simple executable builds, with a standard .NET compiler
- The Ranorex automation library (API) is based on .NET, therefore, allowing you to integrate it into existing test environments and to combine existing automation tasks with Ranorex

- Has smart and easy to read automation code, because of the use of Ranorex repository, which separates GUI identification information from automation code
- Provides the ability to do test automation in your own environment
- Uses standard and modern programming techniques
- Allows testers with less programming knowledge to create professional test modules with Ranorex Recorder
- Ranorex Recorder provides user code actions, which allows developers to provide special validation or automation methods for their testers with less experience in programming
- Targets to get everything automated
- Supports all the technologies through the Ranorex Plug-Ins
- User interface allows for managing test cases and configurations
- Supports use of data variables

2. 5. 2 Rational Functional Tester (RFT)

This product was developed by IBM in 1999. It is an object-oriented automated testing tool. It includes functional and regression testing tools which capture the results of black box tests in a script format. Once captured, these scripts can be executed against future builds of an application to verify that new functionality hasn't disabled previous functionality. With this tool, functional black box tests can be run as well as structural white box tests for memory leaks, code bottlenecks, or measuring code coverage.

In 2006, IBM made a major upgrade to its rational software development platform to better help enterprises build complex applications. The “Baltic”, or IBM Rational

Release 7, was released in 2006. This platform automates much of the software development and delivery process and helps enterprises overcome geographic and organizational silos that hamper development projects. There are 12 products in this new software development platform [6]. The advantages of this tool are [8]:

- Enables regression testing
- Frees up Quality Assurance departments from maintaining and executing basic tests, encouraging the creation of additional, thorough tests
- Automates nontesting activities such as test lab machine preparation and database configuration
- Reduces human error that can occur during activities such as test step execution and test result recording

RFT works with Java, Web based, Microsoft Visual Studio, .NET, terminal-based, SAP, Siebel and Web 2.0 applications [27]. This product uses a patented Object Code Insertion (OCI) technology where no source code is required. The technology looks at the application's executable files. The tools built into the software, including Purify Quantify and PureCoverage, perform white box testing on third party code and controls. Some of the advantages of the white box testing tools are:

- Provides run-time error and memory leak detection
- Records the exact amount of time the application spends in any given block of code for the purpose of finding inefficient code bottlenecks
- Pinpoints areas of application that have and have not been executed

When performing regression tests in this product, if the application changes, for example, images in different locations, tests will not fail because this product uses robust

recognition methods to locate the objects. It does not rely on superficial properties such as screen coordinates to find objects; instead it uses internal recognition properties. This method allows for flexibility in the user interface design, without requiring altering or re-recording the scripts [8].

2. 5. 3 Janova

This tool is similar to others because it enables the user to automate software testing solutions but with this tool it is done in the cloud. The tool does not require scripts to be written; only English-based tools are used that streamlines the task of software implementation with efficient, easy to use tools. Another advantage of this tool is that the pricing starts at $10 per month, which is very reasonable. There is no software to download and no infrastructural investment required [44]. Since it is in the cloud, it is a very quick and easy setup which includes no install. The cloud based software has an easy to navigate home page. This is captured in Figure 5.

2.6 Related Work

Research has been found in many articles where testing tools have been compared. As discussed earlier, one academic research paper presents the design and implementation of an automated software testing tool [2]. The tool described in this paper mechanically created test cases for specific three-variable functions. In each test case that was created, an output is generated and compared to a computed expected test result obtained from an input file [2]. This research provides designers with assurance that the implemented system achieves an extremely high level of correctness [2]. Until now, no research has been done on comparing automated software testing tools to determine, based on metrics, which is the most efficient tool.

Chapter 3: Methodology

The research method includes: (1) identifying a set of tools to be evaluated, (2) developing a metric suite to be used to evaluate the tools, (3) selecting the target application to be tested, (4) manually testing the applications and recording results, (5) performing a feature analysis of each tool and aggregating an ideal feature set, (6) testing the target application using each selected tool and gathering resulting data, (7) interpreting results and (8) drawing appropriate inferences and making recommendations.

3.1 Selected Tools

The testing tools chosen in the comparison for stand-alone based testing were RFT, Ranorex, and Janova. Initially, many other test tools were selected but were rejected because of many reasons. For example, Quicktest and SilkTest were initially chosen but because of the cumbersome setup and initialization they were both rejected. Also, another tool that was initially chosen but rejected was Panorama. This was rejected also because of the cumbersome installation instructions.

The first tool selected is RFT, and the reason it was chosen is because it is the one most widely used. The design of this experiment uses automatic testing tools to go through end user steps in the application and compare features between the tools. The second tool that was evaluated was Janova, and the reason it was chosen was because there is no need to download any software or buy equipment. Since it is cloud based, all that is needed is an Internet connection. The tests, or features, are created, and they are then ready to be 'queued' in the tool. By pressing the queue button, a broker, or sort of middleman, is notified that the feature is ready to run. The broker sends the feature to the next available worker in the cloud where it is then executed against your AUT. The test

is then completed and filtered back down through the broker and back to the browser with results. The last tool that was chosen is Ranorex, and this was chosen mainly due to the fact that it is widely used with web based applications. [45]

3.2 Evaluation Metrics

There are many reasons why we want to compare testing tools. Metrics are important because it shows us a way to compare different tools to efficiently select appropriate tools to use based on what testing needs are at the given time. The criteria for comparison in each tool studied are as follows: features, debugging help, automated progress, support for the testing process, usability, and requirements.

The first metric that will be looked at are the features in each tool. Each feature will be highlighted and compared among the different tools. Table 1 is a description of how the features metric will be defined in this project. The features are listed and then evaluated with each tool.

Table 1:

Definition of Tool Features Metric

	Definition
Install required	This feature will determine whether or not an install is required to use software
Cloud Based	This feature provided by the software, e.g. install required or cloud based
Knowledge of scripts required	This feature provided by the software, e.g. can test scripts be recorded with clicks or is programming knowledge required
Access to code required	This feature provided by the software, e.g. some tests done strictly on lines of code
List of features	The actual features provided by the software, e.g. debugging support

The next metric to be defined is the tool usability metric. In the usability metric, it shows how easily a given tool can be installed, configured and used [7]. This is an important metric in starting to use a tool. Table 2 shows a list of attributes that will be used to collectively define usability. This metric is important because it tells us what the learning curve will be when learning how to use a tool.

Table 2:

Definition of Tool Usability Metric

	Definition
Ease of installation	This feature will document how easy or difficult it is to install the software
User friendly interface	This feature will show how easy or difficult it is to use software and if the features in the tool are easy to understand
Helpfulness of error messages	This feature will show how easy or difficult it is to understand error messages received while using software
Tutorial on 'How-to-Use'	Is a tutorial available on how to use the software and how easy is it to understand
Helpfulness of technical support	How easy or difficult it is to get help when there is a problem with using software
Terminology	This feature will determine whether or not the terminology used in application is easy to understand

The next metric defined is the debugging help metrics. Debugging help is important because if there is a problem with the software there needs to be an easy way to get help. The table below details this metric.

Table 3:

Definition of Debugging Help Metric

	Definition
Ease of getting assistance when an error is encountered	This feature will determine how easy and fast it is to get help from software support to get issue resolved
Helpfulness of where to get help on website	This feature will determine the convenience of where to get help on the company's website
While recording scripts, the ease of using tool	This feature describes whether or not the application is buggy while recording scripts
Documentation of error messages	This feature will determine how well documented and easy to find solutions are to errors received while using software

The next metric outlined is automated progress metric. This is important because when automating scripts, it is important to know how easy the process will be.

Table 4:

Definition of Automated Progress Metric

	Definition
Automated tools availability	This feature will determine whether or not the application has easy to understand automated tools
Automated progress features	This feature describes whether or not automated progress of tests are easily documented
Helpfulness in creating test cases	This feature will determine how easily it is to start developing test cases

The next metric outlined is support for the testing process metric. This tells us how a testing tool actually supports the testing process [7]. This also looks at the support

for comparing test results with an oracle. It looks at whether or not the tool provides a mechanism for automatically comparing the results of the test against the expected results [7]. The table below defines this metric.

Table 5:

Definition of Support for the Testing Process Metric

	Definition
Ability to compare test results with an oracle	This feature will determine if the tool provides a mechanism for automatically comparing results of the test against the expected results
Ability to document test cases	This feature will determine if the tool provides the ability to provide documentation of test cases
Ability to perform regression testing	The feature will determine whether or not the tool provides the ability for automated regression testing

The last metric that will be looked at is requirements. This metric includes things like programming languages, commercial licensing, types of testing environments and system requirements [7]. The last metric outlined is requirements metric. This is important to evaluate because a lot of times it is the easiest thing to look at and quickly review when trying to decide which tool to use.

Table 6:

Definition of Requirements Metric

	Definition
Programming language	This feature will determine what programming languages the tool will work with
Commercial licensing	This will determine what licensing options are available and at what costs

Types of testing environment	This feature will determine if the testing environment a command prompt, Windows GUI, or part of the development environment
System requirements	This feature will determine what operating system the tool will run on, what software requirements does the tool necessitate and what hardware requirements the tool imposes

3.3 General Testing Approach

The tools will be evaluated for their support for web-based testing. The AUT (application under test) was manually tested for a given amount of time. During the stage of manual testing, each feature in the AUT was reviewed to confirm all features were working. In the testing approach, test scripts were written based on what the AUT was written to do. In the automatic testing stage, once each test script below was written, the tests could easily be played over and over depending on how often tests on AUT needed to be run. The hardware specifications of the machines performing tests are below:

Table 7:

Specifications of Testing Machines

	First machine selected to perform tests	**Final machine selected to perform tests**
Manufacturer	HP	HP
Model	Compaq Presario F700 Notebook PC	HP Pavilion g6 Notebook PC
Operating System	Windows 7 – 32bit	Windows 7 - 64bit
Processor	AMD Turion 64 X2 Mobile Technology TL-58 1.90 GHz	intelCorei3 CPU M370@ 2.40 GHz
RAM	2.0 GB	4.0 GB

The table below is the description of what each test script will do.

Table 8:

Test Scripts Definition

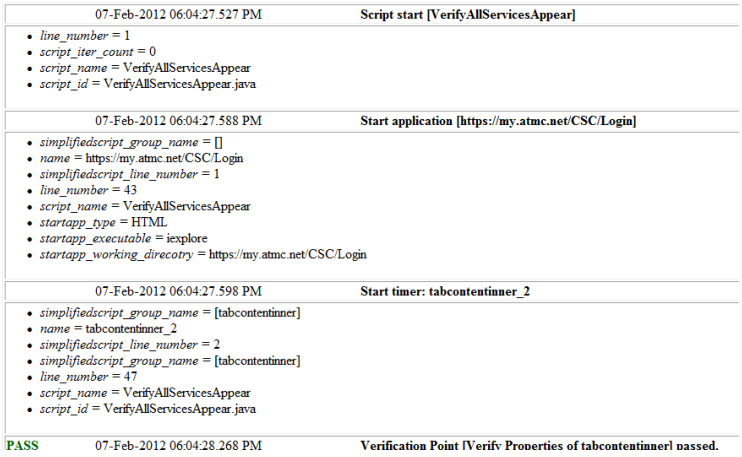
Definition of Test Scripts	
Verify All Services Appear	This script verifies that after logging into application that under the first tab, equipment and services, that all current services appear on page.
Verify Login Works	This script verifies that a user can successfully login.
Verify Tabs Work	The script verifies that after logging into application that each tab on page successfully works.
Verify Statements are Viewable	This script verifies that after logging into application that under the 'statements' tab the current statement appears successfully.
Verify Logout Works	This script verifies that the current user can successfully log out.
Verify Pay Online Works	This script verifies that a payment online can successfully be made.
Verify Printer Friendly PDF Works	This script verifies that when viewing statements, that a PDF can be downloaded and/or viewed.
Verify Toll Calls Are Viewable	This script verifies that after logging into application the tab 'toll calls' can be used to view current or past toll calls either unbilled or previous months toll calls.

3.4 Target Application

In the experiment the application chosen for the comparison is a customer web based application. This particular application was chosen because users need to use the site to access their services to which they currently subscribe to, view current billing statements and history, view toll calls, change account settings and pay their bill online.

Chapter 4: Results

For each test that was completed, the metrics were outlined and results found for each metric. Results were first taken by reviewing the first tool, RFT. Below is an image of one of the logs received after playing back a test script in RFT. The complete logs were captured with screen shots in Appendix B.


07-Feb-2012 06:04:27.527 PM **Script start [VerifyAllServicesAppear]**

- *line_number* = 1
- *script_iter_count* = 0
- *script_name* = VerifyAllServicesAppear
- *script_id* = VerifyAllServicesAppear.java

07-Feb-2012 06:04:27.588 PM **Start application [https://my.atmc.net/CSC/Login]**

- *simplifiedscript_group_name* = []
- *name* = https://my.atmc.net/CSC/Login
- *simplifiedscript_line_number* = 1
- *line_number* = 43
- *script_name* = VerifyAllServicesAppear
- *startapp_type* = HTML
- *startapp_executable* = iexplore
- *startapp_working_direcotry* = https://my.atmc.net/CSC/Login

07-Feb-2012 06:04:27.598 PM **Start timer: tabcontentinner_2**

- *simplifiedscript_group_name* = [tabcontentinner]
- *name* = tabcontentinner_2
- *simplifiedscript_line_number* = 2
- *simplifiedscript_group_name* = [tabcontentinner]
- *line_number* = 47
- *script_name* = VerifyAllServicesAppear
- *script_id* = VerifyAllServicesAppear.java

PASS 07-Feb-2012 06:04:28.268 PM **Verification Point [Verify Properties of tabcontentinner] passed.**

Figure 1: RFT - Log Result of 'Verify All Services Appear' Script

4.1 Results of Three Testing Tools

The results were extensive and outlined in this section. All three tools have been tested, evaluated and compared to each other. The first metric, features of testing tools, has results that are detailed below.

Table 9:

Features of Testing Tools

	Ranorex	Rational Functional Tester	Janova
Install required	Yes	Yes	No
Cloud Based	No	No	Yes

Knowledge of scripts required	No	Yes	No
Access to code required	No	No	No
List of features	Yes	No	No

The next metric results set is for tool usability. This was found to be important when trying to determine which tool would be best to use. The ease of installation is very important if time is a factor. Table 11 shows the results found in looking at usability metric via a ranking system from 1 through 10 [7]. Table 10 shows the results found in looking at the usability metric.

Table 10:

Results of Tool Usability

	Ranorex	Rational Functional Tester	Janova
Ease of installation	Very easy to install	Quite a few steps to install. In addition to the software, it is required to also install IBM Installation Manager.	No install
User friendly interface	Yes	Yes	The terminology in tool hard to understand. Test scripts needed to be 'Features' with 'Scenarios' to then be 'queued' in Test Queue. Each 'feature' had to be configured with a file path.
Helpfulness of error messages	Error messages documented	Error messages documented	Error messages were documented on website but when using links to get additional help on how to resolve, many links were broken and no help existed. Had to email technical support and wait average of 24 hours for response.

Tutorial on 'How-to-Use'	Very easy to follow	From Help / Getting started, there is a 'getting started with the product' link that is very useful	Very easy to follow
Helpfulness of technical support	Easy to find	Not easy to find	Not easy to find
Terminology	Specific to software	Specific to software	Specific to software

Table 11:

Usability of Testing Tools – Range 1 – 10 (1 Lowest to 10 Highest)

	Ranorex	Rational Functional Tester	Janova
Ease of installation	8	7	NA
User friendly interface	6	7	7
Helpfulness of error messages	7	8	3
Tutorial on 'How-to-Use'	6	5	2
Helpfulness of technical support	4	7	8

The next metric result set is for debugging help. This was found to be important when trying to determine which tool is easiest to get help in. One of the error messages received in the Janova tool is documented below. One of the issues with this tool compared to RFT is that the learning curve was much greater. The results are outlined below.

Table 12:

Results of Debugging Help

	Ranorex	Rational Functional Tester	Janova

Ease of getting assistance when an error is encountered	Quick to respond	When playing back first script, the log was viewable. For each verification point, there is a 'view results' link that was broken. When entering this into Help, produced no results. Entered 'view results cannot be seen'	Emailed support and up to 24 hours later gets a response. Seemed that email responses only received 8am – 5pm EST.
Helpfulness of where to get help on website	Very easy to find help	The IBM technical support site is not convenient to use.	Not very helpful. If 'Logout' was not used, received 'the user has already logged in' even if the user was not logged in at that time and prevented the user from logging in for 15 minutes.
While recording scripts, the ease of using tool	No issues	While recording, 2 out of 3 times the Recording toolbar blacked out and was not visible. Also, the toolbar will disappear while recording.	Very difficult to get started based on terminology specific to tool. For example, if there is any step replication in testing a 'Flow' can be used.
Documentation of error messages	Error messages were documented on website	Some of the errors appearing during testing had numbers in error messages. These could easily be found in Help section. Issues like recording toolbar being blacked out (or just becoming not visible) were not documented on Internet anywhere.	When trying to get help with 'missing location string' received the error 'link does not exist'. Also, each time test queue did not succeed, the log to view why would not appear. Error 'IE cannot display the page'. Finally the log would display. See Appendix Figure "Janova Error Log of Failure"

The next metric result set is for automated progress. This was found to be helpful when running batches of tests and knowledge on how tool can perform in regards to automated testing. The results are outlined below.

Table 13:

Results of Automated Progress

	Ranorex	Rational Functional Tester	Janova
Automated tools availability	The logs show details on each pass/fail	The logs show how many verification points pass/fail and why. This can be used to test automatically.	The 'batches' feature is only available with premium membership
Automated progress features	The features are specific to the software	The features are specific to the software	The features are specific to the software
Helpfulness in creating test cases	All test scripts in English	Easy to create in simplified. For example, simplified test scripts can be created (in English) or Java test scripts can be created	Test cases not easily created

The next metric result set is for support for the testing process. This determined whether or not the testing tool fit into the testing process easily [7]. The results are outlined below.

Table 14:

Results of Support for the Testing Process

	Ranorex	Rational Functional Tester	Janova
Ability to compare test results with an oracle	Yes	Yes	No
Ability to document test cases	Yes	Yes	Yes
Ability to perform regression testing	Yes	Yes	Yes

The next metric is Requirements. This is a pertinent metric in determining if the tool is feasible given a hardware system that will be performing the tests. One issue that was encountered in this research was when using the RFT tool. The first machine that was intended on performing tests was a Compaq Presario laptop with 2GB RAM and a 1.9 GHz processor. Each time the tool was opened, the operating system would shut down. Lots of time was spent on determining what the issue was when it was apparent that the hardware requirements were not being met. A new machine was able to be used with the specifications laid out in the general testing approach section. This can be found in table 7. The table below summarizes the requirements metric for all three tools.

Table 15:

Results of Requirements

	Ranorex	Rational Functional Tester	Janova
Programming language	Java, C#, VB.net	Java or VB .NET	Java
Commercial licensing	See website	See website	Basic $10/month
Types of testing environment	Windows GUI	Windows GUI	Windows GUI
System requirements	This depends on the version of the .NET Framework that needs to be installed for the respective Ranorex package. Ranorex requires at least the .NET Framework 2.0 to be installed, Ranorex Studio requires the .NET	Minimum processor: 1.5 GHz Intel(R) Pentium(R) 4 Minimum memory: 1 GB RAM Disk space: Minimum: 750 MB of	Internet access

	Framework 3.5. Consequently:	disk space is required for product package installation. Display: 1024 x 768 display minimum using 256 Colors	

Table 16:

Results of Summary of Testing Tool Overview

Tool	Manuf.	Licensing Cost	Language	Current Version	Type
Ranorex	Ranorex	Premium Machine Bound Licenses + 1st year subscription/maintenance = $1,980 *Annual subscription = $390 Premium Floating Licenses + 1st year subscription/maintenance = $3,980 Annual subscription = $790 Run-Time Machine Bound Licenses + 1st year subscription/maintenance = $489 *Annual Subscription = $99 Run-Time Floating Licenses + 1st year subscription/maintenance = $980 *Annual Subscription = $190	Java	3.2.2	Object Oriented
Janova	Janova	$10/month for Basic	Java	1.4	Cloud
RFT	IBM	See IBM website for pricing	Java or Visual Basic .NET	8. 2. 0	Object Oriented

The table below summarizes the test scripts in all three testing tools with the Pass/Fail results of each tool. When playing back test scripts, the results is either Pass or Fail in all three tools.

Table 17:

Test Script Results

	Ranorex	Rational Functional Tester	Janova
Verify All Services Appear	Pass	Pass	Pass
Verify Login Works	Pass	Pass	Pass
Verify Tabs Work	Pass	Pass	Pass
Verify Statements are Viewable	Pass	Pass	Pass
Verify Logout Works	Pass	Pass	Pass
Verify Pay Online Works	Pass	Pass – This shows a False Negative Result – The last step in script had to be skipped due to cursor not being recognized in payment text box.	Pass
Verify Printer Friendly PDF Works	Pass	Pass	Pass
Verify Toll Calls Are Viewable	Pass	Pass	Pass

Another area of results includes issues that were encountered when using each tool. Some issues were error messages encountered; others were issues with how each tool works or false/positive results when playing back test scripts. In RFT, when recording the script 'Verify Statements are Viewable', the recorder would not recognize changing current month to previous month. Received error 'could not record script'. When recording the script 'Verify Toll Calls are Viewable', the recorder stopped and error

message appeared ‘the script could not be recorded’. Also with this test script another bug was encountered where the software duplicates every step in the script. There is an image of this in Appendix A. Also, a ‘shared’ object map was created, but the same error ‘could not record script’ appeared. Even when restarting the application the object map could not easily be recorded. Once the operating system on machine was rebooted, the object map could then be successfully created.

In Janova, there was a big learning curve. To be able to execute batches of test scripts, the Professional or Enterprise version was required. It was found to be very difficult to easily find out how to create scripts to queue to test. There was a bug found in the application that the support team said would be fixed at some point in the future. Multiple error messages appeared on the page when there was no activity on the ‘test queue’ page for longer than 15 minutes. The page has an auto-refresh every five seconds, so when the session is closed after 15 minutes the page still tries to update but throws error messages. The Internet Explorer browser version 7 was not supported. One system that was being used to perform tests had this version and when the ‘save’ button was used on any page, the browser would unexpectedly close and the user had to wait 15 minutes to log back into the testing tool since the logout button was not used correctly. Also, even though the tool is cloud based, each browser needed additional tools downloaded to ‘inspect’ elements on the web application for testing purposes. The browser needed the IE developer toolbar and Firefox browser needed FireBug installed.

The amount of time spent in each testing tool was documented. In Janova, it had the largest learning curve where it took over 33 hours to get proficient at using the tool to create and run test scripts. In RFT, it only took an average of three hours. In Ranorex, it took less than three hours.

4.2 Scripts in RFT

A systematic evaluation process was followed for this tool. After the tool was installed, the process of creating test scripts was easy to learn. This tool is user friendly and compared to Janova tool the process of recording test scripts was simple and easy to follow using one of the online tutorials. Below the test script was captured in an image to show the steps in the first script. The remaining scripts and log results are found in the appendix.

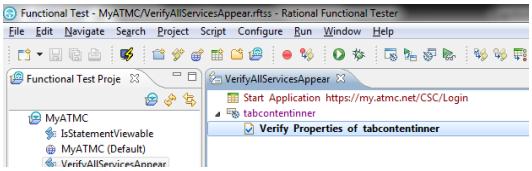


Figure 2: RFT - Test script of 'Verify All Services Appear'

Below is the code that the software generates from creating this first test script.

```
// DO NOT EDIT: This file is automatically generated each time
the script is modified.
// To modify this file either use 'Insert Java Code Snippet'or
'Insert Java Method'
// option from simplified script.

import resources.VerifyAllServicesAppearHelper;
import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.WPF.*;
import com.rational.test.ft.object.interfaces.dojo.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.object.interfaces.flex.*;
import
com.rational.test.ft.object.interfaces.generichtmlsubdomain.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;
import com.ibm.rational.test.ft.object.interfaces.sapwebportal.*;

// BEGIN custom imports
//TODO: Add custom imports here.
// END custom imports
```

```
/**
 * Description   : Functional Test Script
 * @author Nancy
 */
public class VerifyAllServicesAppear extends
VerifyAllServicesAppearHelper
{
	/**
	 * Script Name   : <b>VerifyAllServicesAppear</b>
	 * Generated     : <b>Feb 13, 2012 8:21:08 PM</b>
	 * Description   : Functional Test Script
	 * Original Host : WinNT Version 6.1  Build 7601 (S)
	 *
	 * @since  2012/02/13
	 * @author Nancy
	 */
	public void testMain(Object[] args)
	{


		setSimplifiedScriptLine(1); //Start  Application
https://my.atmc.net/CSC/Login
		startApp("https://my.atmc.net/CSC/Login");

		// Group: tabcontentinner
		setSimplifiedScriptLine(2); //tabcontentinner

		timerStart("tabcontentinner_2");

		setSimplifiedScriptLine(3); //Verify  Properties  of
tabcontentinner

	html_tabcontentinner().performTest(TabContentInnerVP());

		timerStop("tabcontentinner_2");
	}
}
```

Figure 3: RFT - Code from 'Verify All Services Appear' Script

Below is an image of the verification point entitled 'TabContentInner', which verifies the value of the class name on the web page element.

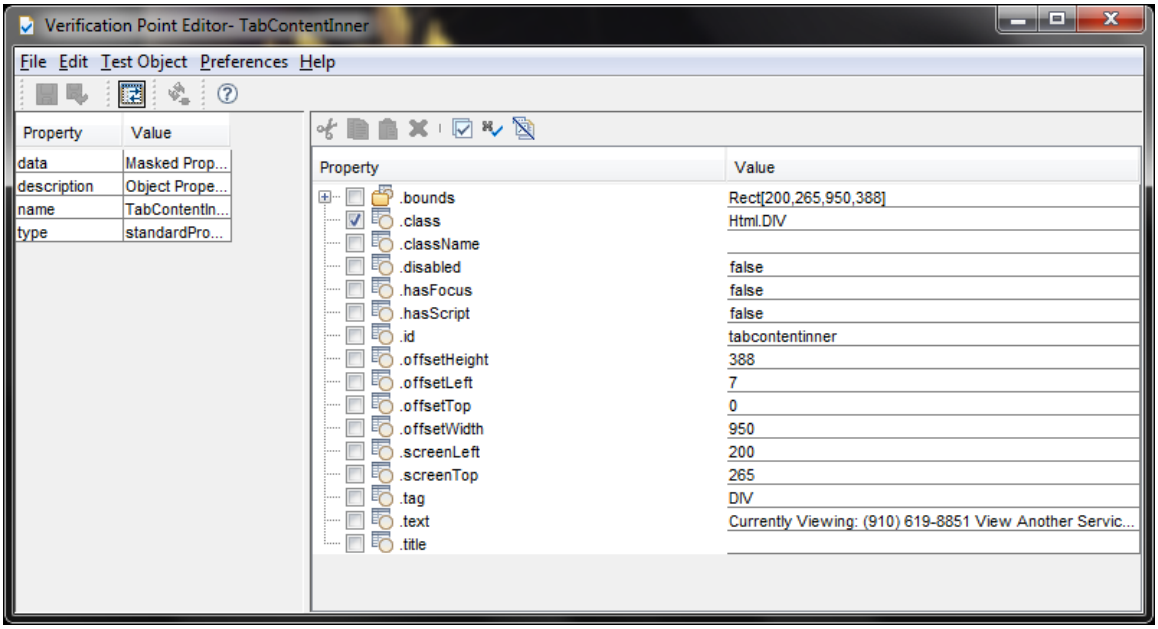


Figure 4: RFT Verification Point

4.3 Scripts in Janova

This tool is advertised as 'write your test in plain English' then 'execute the tests'. After using the tool, it was realized that if the Janova terminology is easily understood and used correctly, then test scripts will execute. Each test can be queued up to be executed. Below is the image of Janova welcome page.

Figure 5: Janova Home Page

For each ‘test’ that needs to be reviewed in an application these are these steps to be followed.

1. Application – Setup the path to application to be tested (AUT)
2. Page – This is where each element to be tested is mapped to a ‘location string’
3. Scenario – A specific behavior necessary to implement a feature
4. Feature – A scenario or group of scenarios where the automated tests are written and can be executed to test an application
5. Step – A line in a Scenario that describes the action to be tested
6. Flow – A series of steps that can be commonly used across multiple Scenarios and Features to save time and increase efficiency
7. Queue – Used to execute a single scenario in a feature

In the beginning stages of testing, none of the test scripts would successfully run. The reason for most of the failed errors was because the web page elements could not be found on the page. A script, or Feature, in the Janova tool included a Scenario. The feature, called “Verify Login Works”, continued to fail until the correct mappings were written to each part of the AUT for this Feature. The image below shows the script details and the three X’s in red font to right of screen show how many times it failed.

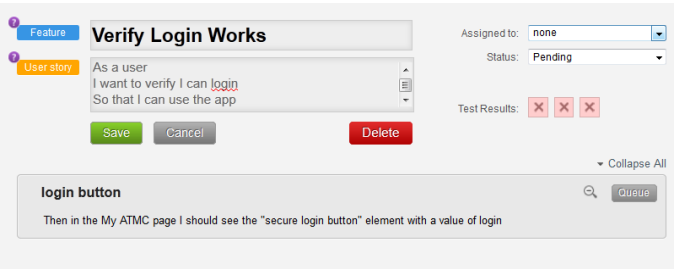


Figure 6: Janova - Test Script 'Verify Login Works' with Multiple Failures

The image below shows the 'Verify Login Works' script once the page elements were captured correctly and the script, or Scenario, passed. Appendix C shows additional scripts and Appendix D shows the logs for the 'passed' scenarios and features test scripts.

Figure 7: Janova - Test Script 'Verify Login Works'

To make the scripts run easier in Janova, a Flow can be created for a series of steps commonly used among Scenarios or Features. The image below shows the Flow that was created and then used in all Features. It was called 'LoginAsNancy' since the login was for a user named Nancy and that user logged into AUT for each Feature that was written.

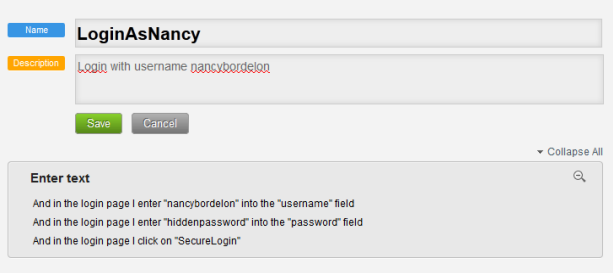


Figure 8: Janova Flow

One of the major reasons why scripts kept failing in this tool was because it is case sensitive based on element names. For example, if a page element is named 'logout'

and the element ‘Logout’ is used in script, the element is not found. The image below shows one of the errors received in the early stages of testing.

Figure 9: Janova - Failed Results of ‘Verify Login Works’ Scenario

When the link ‘click to view’ in red font was clicked on, the link would not work. The image below captures what was received when trying to view why it failed.

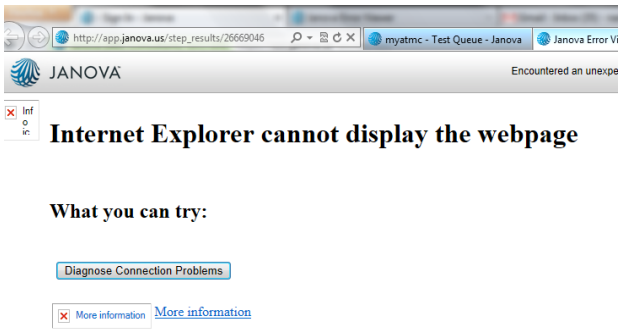


Figure 10: Janova - Error When Trying to Display Reason of Failure

After days of receiving the message above, after browser was updated the ‘click here to view’ link started working. Below is an image of provided failed message details.

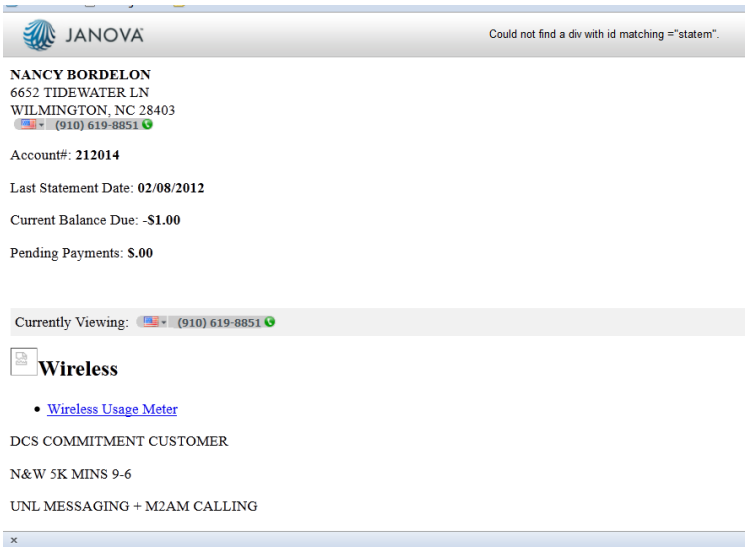


Figure 11: Janova - Error Reason of Failure

Once the learning curve to execute tests was completed, the tool became easy to use. With every feature or scenario that was created, the path of each element on web page had to be mapped. This was difficult since the AUT was a secure web application and elements were hidden using cascading style sheets. For example, each button element could be mapped by using an XPath. The XPath could easily be seen when script was run. Another way to test in this tool was to use an element on the web page using the HTML tag 'href'. If this tag was used as the 'location method' in the page configuration, then the 'location string' could be used with a wildcard character. For example, on the AUT there is an element with a HREF tag of '*.BillStatement'. In Janova there is also a checkbox for the 'REGEX' option, which tells the tool to ignore anything before the text 'BillStatement' and just focus on a link that has that string in the relative path. This part of the testing, where each element needed to be mapped correctly, took the longest. Below is an image of the 'Pages' element in the tool where this mapping was used for the 'Statements' element.

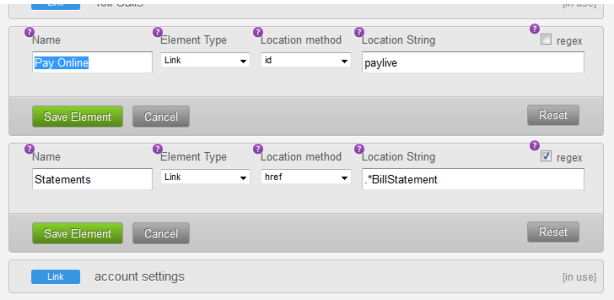


Figure 12: Janova - Mapping of Elements

4.4 Scripts in Ranorex

Ranorex is a very easy to use and extensive automated software testing tool. It is based on XPath, which is used to navigate through elements and attributes in an XML document. Ranorex has great object identification capabilities and supports multiple technologies. The results from Ranorex are very similar to the other two testing tools used. One big difference is that even though it is based on XPath, the XPath of web elements does not need to be mapped individually when creating test scripts. In this tool it is done in the background while clicking on different web elements. The tool was the easiest of the three to learn how to use. It is very easy to follow the 'learn how to use' tutorials in getting started recording test scripts. The image below shows the 'Verify All Services Appear' script. Appendix E shows additional scripts and Appendix F shows the logs for the 'passed' test scripts.

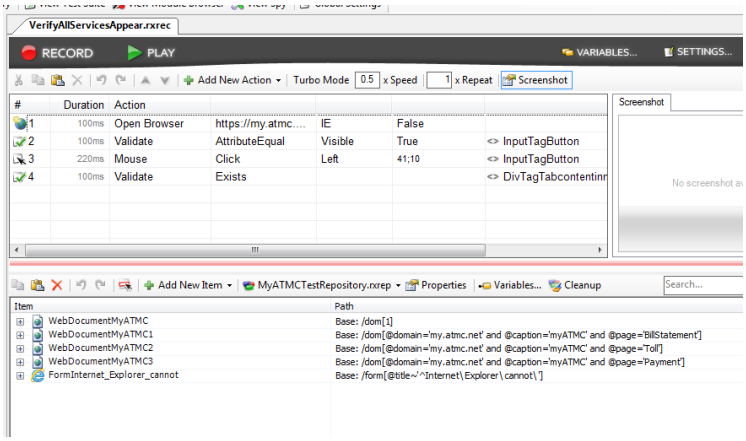


Figure 13: Ranorex - Test Script for 'Verify All Services Appear'

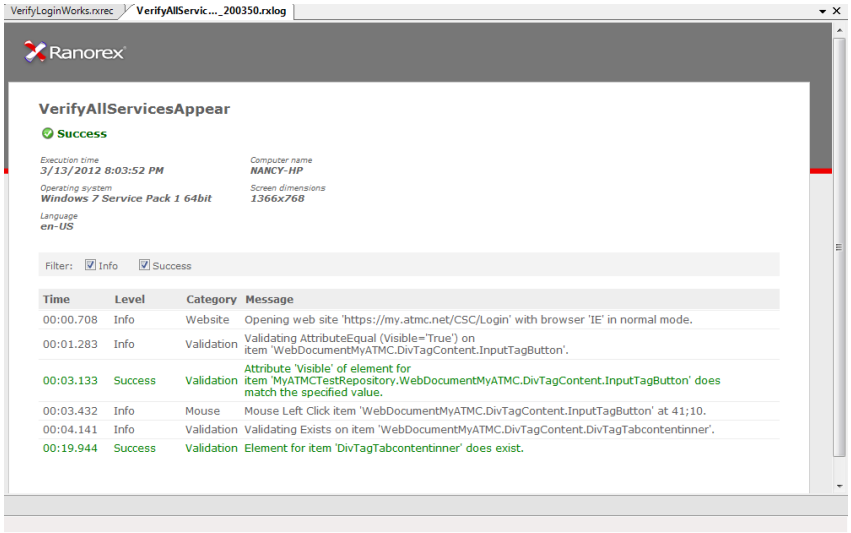


Figure 14: Ranorex – Log Result of 'Verify All Services Appear' Script

Chapter 5: Discussion and Lessons Learned

5.1 Project Rationale

With this research we have tried to show the differences between three software testing tools and why one would work better than another given a need in software testing. I have been driven by many things in inspiration for the research topic. I have been driven by my occupation in the Information Technology field for the last fifteen years. I have been in the computer training field, the desk side and phone support field and currently the software development and software testing field. Information Technology has been an ever changing field and by using so many different software programs throughout my career, I've realized how important bug-free software programs are to the field.

There were many classes that inspired me to further my research with this topic. One was an elective called Component Based Software Engineering. This class really inspired me to learn more about the software testing process and why it is so important. Research was done in that class that intrigued me about how there are different ways to program and how testing is so important in all types of software engineering. Another big reason why this research was chosen is because of the Software Engineering course. This course taught me the essentials of software engineering including how important the testing phase is to the entire software development life cycle. Another class that inspired me was Analysis and Modeling of Information Systems. This class had topics of project estimation and management, implementation issues, and training that really intrigued me to learn more about this field. I have been in the field of Information Technology now for 15 years and learn something new almost every day.

5.2 Lessons Learned

With the first metric result, I learned that the features in each tool are ever so important to determine what each tool can do for testing needs. Debugging help is one of the most important because when bugs or questions arise the testing is at a standstill until the issues are addressed. RFT was found to have the most bugs and/or issues when being compared to the other two testing tools. The error messages that appeared during testing really prolonged the end results. For example, the recording tool bar disappearing while testing really made for a long recording process given that the issue did not clear up until the operating system was restarted. The Janova tool had such a great learning curve. It was quite a task to get to the 'passing' results of each test script. Once the terminology was understood in the tool, the scripts could be written without failing and tests passed as expected. Also, the usability metric was found to be a very important because when selecting a testing tool the learning curve was important in the process of creating necessary test scripts and getting passing results. The requirements metric was also found to be of importance in looking at what tools would be legitimate options given the current hardware of a system performing tests.

5.3 Strengths and Weaknesses

In the Janova tool I would have liked to have seen an easier way to test with a cloud based environment. The time it took to get past the learning curve was too great. Once the tool terminology made sense it was easy to create the test scripts, but it took too much time to navigate through each test script and get past the failed tests. Another weakness of Janova was that if the browser closed unexpectedly, the login session was automatically on hold for 15 minutes. This prevented the testing from progressing, due to

this 15 minute timeout period in the tool. RFT is a very powerful tool and, with regard to regression testing, it is really easy to use for that need. Research has also been done with another AUT, and the regression testing was very easy to perform when updates were made to the AUT.

5.4 Challenges Encountered

The bugs I encountered were time consuming to overcome. The Janova tool really caused the testing time to lengthen from the expected one week to actually six weeks in that tool. The other two testing tools, RFT and Ranorex, did not have the learning curve as in Janova. The bugs encountered in RFT also slowed down testing, since the testing machine would have to be restarted to get past some of the errors encountered.

Chapter 6: Conclusion and Future Work

In concluding this research, I have learned that software testing tools are very different. It takes time and effort and having a software testing goal to know which tool is the best to use given the type of software testing needs. My personal recommendations are the three tools that have been through the metrics in this research. Depending on the needs of a given AUT, the tools recommended would be different given the web based applications or stand alone. RFT is definitely the tool to be used when regression testing is important. Janova is the best tool given it is cloud based and can be accessed from any testing machine with internet access. Ranorex is the best tool for web based applications given the different test automation tools built into the software package.

My vision for the ideal tool is one that is cloud based with no install required and is easy to learn how to use. The ideal testing tool should be easy to navigate and also include many tutorials on how to get started in using the tool. It should also have minimal bugs since the bugs encountered during this research were so time consuming to get past.

Bibliography

1. T. K. Abdel-Hamid, The Economics of Software Quality Assurance: A Simulation-Based Case Study, (Management Information Systems Research Center, University of Minnesota), URL: http://www. jstor. org/pss/249206, 1988.
2. J. Meek, N. Debnath, I. Lee, H. Lee. Algorithmic Design and Implementation of an Automated Testing tool, URL: http://www. computer. org. uncclc. coast. uncwil. edu/portal/web/csdl/abs/proceedings/itng/2011/4367/00/4367a054toc. htm. pp. 54-59, Eighth International Conference on Information Technology: New Generations, 2011.
3. N. Koochakzadeh, V. Garousi, A Tester-Assisted Methodology for Test Redundancy Detection, Advances in Software Engineering, Hindawi Publishing Corporation, V 2010, Article ID 932686, 2009, URL: http://www.hindawi.com/journals/ase/2010/932686/
4. Sara Sprenkle, Holly Esquivel, Barbara Hazelwood, Lori Pollock, WebVizor:A Visualization Tool for Applying Automated Oracles and Analyzing Test Results of Web Applications, IEEE Computer Society, August 2008.
5. Macario Polo, Sergio Tendero, Mario Piattini, Integrating techniques and tools for testing automation, EBSCO host database, ISSN# 09600833, Wiley Publishers, March 2007.
6. Darryl Taft, IBM Readies Rational Revamp, EBSCO host database, ISSN# 15306283, Academic Search Complete, June 2006.
7. David Crowther, Peter Clarke, Examining Software Testing Tools, Dr. Dobb's Journal:Software Tools for the Professional Programmer, ISSN# 1044789X, Academic Search Premier, June 2005, Vol. 30, Issue 6.
8. IBM.com, IBM Rational Functional Tester, IBM Corporation, December 2008, URL: http://public.dhe.ibm.com/common/ssi/ecm/en/rad14072usen/RAD14072USEN.PDF.
9. Stuart Feldman, Quality Assurance: Much More than Testing, ACM Digital Library, Queue – Quality Assurance, February 2005, Vol 3, Issue 1.
10. Keith Stobie, Too Darned Big To Test, ACM Digital Library, Queue – Quality Assurance, February 2005, Vol 3, Issue 1.

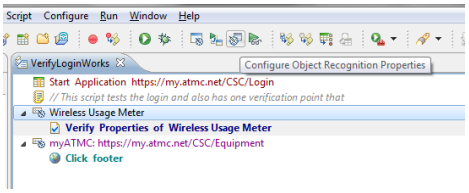
11. William White, Sifting through the Software Sandbox SCM meets QA, ACM Digital Library, Queue – Quality Assurance, February 2005, Vol 3, Issue 1.
12. Antti Kervinen, Pablo Virolainen, Heurisitics for Faster Error Detection with Automated Black Box Testing, Science Direct, Electronic Notes in Theoretical Computer Science, Vol 111, January 2005, URL: http://www. sciencedirect. com/science/article/pii/S1571066104052326
13. Siegfried Goeschl, The JUnit ++ Testing Tool, Dr. Dobb's Journal: Software Tools for the Professional Programmer, February 2001, Academic Search Premier, February 2001, Vol. 26, Issue 2.
14. Sylvia Ilieva, Valentin Pavlov, IlinaManova, A Composable Framework for Test Automation of Service-based applications.
15. Yuetang Deng, Phyllis Frankl, Jiong Wang, Testing Web Based Applications, ACM, September 2004 SIGSOFT Software Engineering Notes, Volume 29 Issue 5.
16. Cem Kaner, Jack Falk, Hung Nguyen, Common Software Errors, Testing Computer Software, Second Edition, 1993.
17. Hamzeh Al Shaar, Ramzi Haraty, Modeling and Automated Blackbox Regression Testing of Web Applications, 2008, Journal of Theoretical & Applied Technology, 4 (12), 1182 – 1198. EBSCO host database, ISSN# 19928645.
18. Edward Hieatt, Robert Mee, Going Faster: Testing the Web Application, IEEE Computer Society, August 2002, Vol 19, Issue 2, Pg (60 – 65), URL: http://www. csse. monash. edu. au/courseware/cse4431/testingWebApps-IEEE-2002. pdf
19. Geoff Dromey, Climbing over the 'No Silver Bullet' Brick Wall, IEEE Computer Society, 2006, Pg (118 – 119).
20. Panagiotis Louridas, JUnit: Unit Testing and Coding in Tandem, IEEE Computer Society, 2006, Pg (12 – 15).
21. Antonia Bertolino, Software Testing Research: Achievement, Challenges, Dreams, IEEE Computer Society, 2007.
22. Bernard Stepien, Liam Peyton, Pulei Xiong, Framework testing of web applications using TTCN-3.
23. Fei Chen, Xiao Hang Zhang, Research on Evolution of Chinese Telecom Industry System Based on Dissipative Structure Theory, EBSCO database.
24. Frank Maurer, Engineering Web Applications with Java, University of Calgary.

25. Fei Chen, Xiao Hang Zhang, Research on Evolution of Chinese Telecom Industry System Based on Dissipative Structure Theory, Journal of computers, April 2011.
26. James Whittaker, Herbert Thompson, Black Box Debugging, ACM Digital Library, December/January 2003-2004,Vol 3, Issue 1.
27. IBM developerWorks, Bridging the Gap Between Black Box and White Box Testing, URL: http://www. ibm. com/developerworks/rational/library/1147. html.
28. Roger Pressman, Software Engineering, A Practitioner's Approach, Sixth Edition, 2005.
29. Chip Davis, Daniel Chirillo, Daniel Gouveia, Fariz Saracevic, Jeffrey Bocarsly, Larry Quesada, Lee Thomas, Marc van Lint, Software Test Engineering with IBM Rational Functional Tester, 2010, International Business Machine Corporation, The Definitive Resource. URL: http://ptgmedia. pearsoncmg. com/images/9780137000661/samplepages/0137000669_sample. pdf
30. John Satzinger, Robert Jackson, Stephen Burd, Object-Oriented Analysis & Design with the Unified Process, Thomson Course Technology, 2005.
31. Ian Sommerville, Software Engineering, Eighth Addition, 2007.
32. A Askarunisa, N Ramraj, Shanmuga Priya, Selecting Effective Coverage Testing Tool Based on Metrics, The Icfai University Journal of Computer Sciences, Vol III, No. 3, 2009.
33. Linda Hayes, The Truth about Automated Test Tools, Datamation, Vol 43, Issue 4, April 1997, EBSCO host database, Academic Search Premier.
34. Pak-Lok Poon, Sau-Fun Tang, GH Tse, TY Chen, Choc'late A Framework for Specification-based Testing, ACM, Vol 53, No. 4, April 2010.
35. Amanda Prusch, Tina Suess, Richard Paoletti, Stephen Olin, Starann Watts, Integrating Technology to Improve Medication Administration, AM J Health-Syst Pharm, Vol 68, ACM May 1, 2011.
36. Judy Bowen, Steve Reeves, UI-Driven Test-First Development of Interactive Systems, Proceedings of the 3rd ACMSIGCHI symposium on Engineering interactive computing systems. ACM, New York, NY, 165-174. DOI = 10.1145/1996461.1996515.
37. Ranorex website, 2012 Ranorex GmbH, URL: www.ranorex.com.

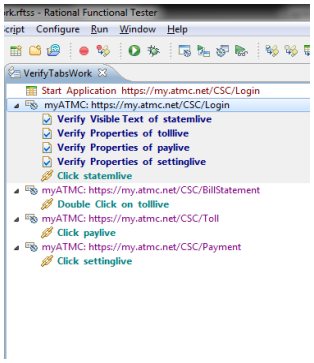
38. Judy Bowen, Steve Reeves, WinRunner Tutorial, URL: http://students.depaul.edu/~slouie/wr_tut.pdf.
39. SilkTest Installation Guide, URL: http://microfus.securehttp.internapcdn.net/secure_MicroFUS/Products/silk/silktest2011/silktest_installguide_en.pdf?token=V1dwQ2UixCApNogQ3oOY.
40. Silktest website, URL: http://www.borland.com/us/products/silk/silktest/.
41. Cameron Hughes, Tracey Hughes, The Joys of Concurrent Programming, 2/27/2004, URL: http://www.informit.com/articles/article.aspx?p=30413.
42. Hong Zhu, Patrick Hall, John May, Software Unit Test Coverage and Adequacy, ACM Digital Library, ACM Computing Surveys, December 1997, Vol. 29, No. 4, page 371. URL: http://laser.cs.umass.edu/courses/cs521-621/papers/ZhuHallMay.pdf.
43. Microfocus website, 2001 – 2012, URL: https://www.microfocus.com/products/silk/index.aspx.
44. Enhanced online News, Janova Launces Simple Yet Powerful Automated Software Testing Solutions Leveraging the Power of the Cloud, 2012 Business Wire, December 1, 2011 URL: http://eon.businesswire.com/news/eon/20110412005635/en/software-testing/software-development/software.
45. Janova website, Janova 2011, URL: http://www.janova.us/.
46. XPath Introduction, 1999-2012, Refsnes Data, URL: http://www.w3schools.com/xpath/xpath_intro.asp.

Appendix A – Test Scripts in RFT

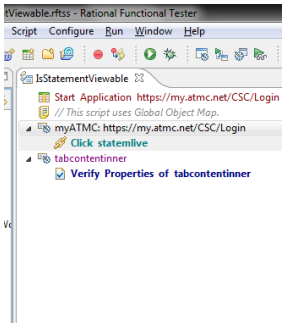
Test script - ‘Verify Login Works’



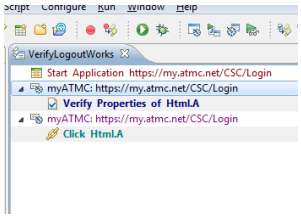
Test script - ‘Verify Tabs Work’



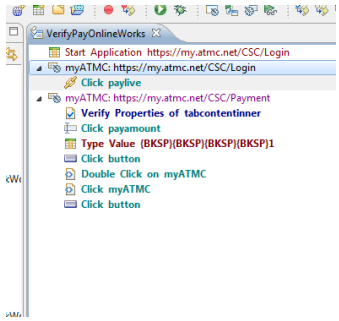
Test script - ‘Is Statement Viewable’



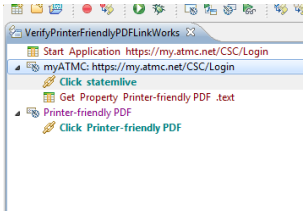
Test script - ‘Verify Logout Works’



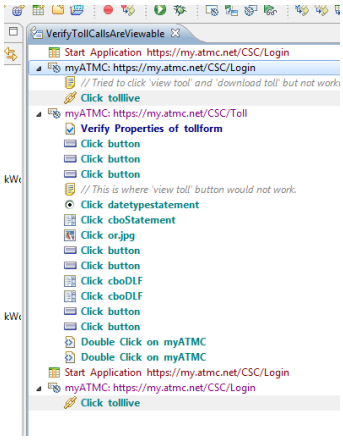
Test script - ‘Verify Pay Online Works’



Test script - ‘Verify Printer Friendly PDF Works’

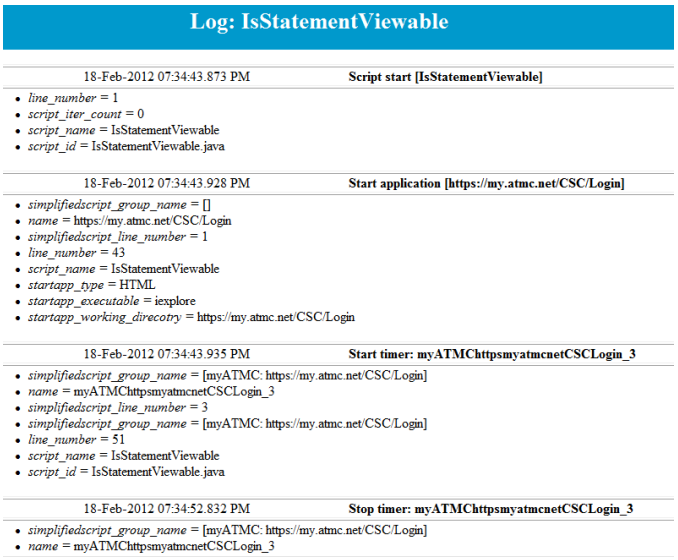


Test script - ‘Verify Toll Calls are Viewable’



Appendix B – Log Results in RFT

Log Results of 'Is Statement Viewable' Script



Log: IsStatementViewable

18-Feb-2012 07:34:43.873 PM **Script start [IsStatementViewable]**

- *line_number* = 1
- *script_iter_count* = 0
- *script_name* = IsStatementViewable
- *script_id* = IsStatementViewable.java

18-Feb-2012 07:34:43.928 PM **Start application [https://my.atmc.net/CSC/Login]**

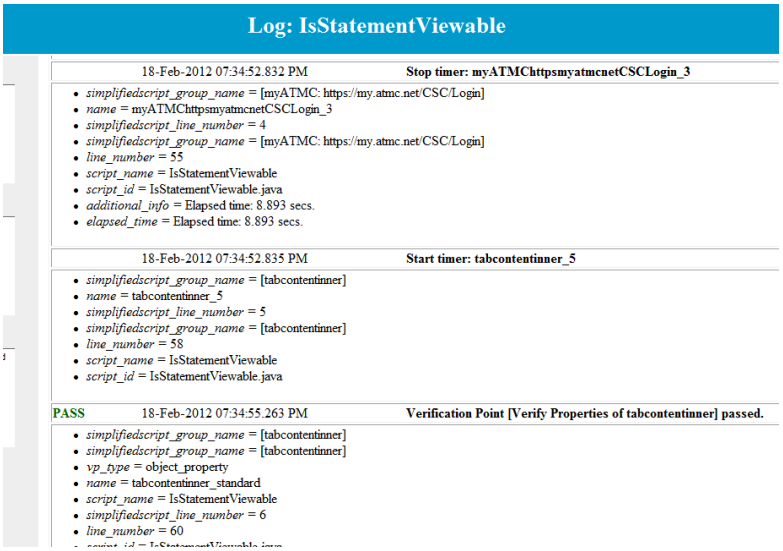
- *simplifiedscript_group_name* = []
- *name* = https://my.atmc.net/CSC/Login
- *simplifiedscript_line_number* = 1
- *line_number* = 43
- *script_name* = IsStatementViewable
- *startapp_type* = HTML
- *startapp_executable* = iexplore
- *startapp_working_directory* = https://my.atmc.net/CSC/Login

18-Feb-2012 07:34:43.935 PM **Start timer: myATMChttpsmyatmcnetCSCLogin_3**

- *simplifiedscript_group_name* = [myATMC: https://my.atmc.net/CSC/Login]
- *name* = myATMChttpsmyatmcnetCSCLogin_3
- *simplifiedscript_line_number* = 3
- *simplifiedscript_group_name* = [myATMC: https://my.atmc.net/CSC/Login]
- *line_number* = 51
- *script_name* = IsStatementViewable
- *script_id* = IsStatementViewable.java

18-Feb-2012 07:34:52.832 PM **Stop timer: myATMChttpsmyatmcnetCSCLogin_3**

- *simplifiedscript_group_name* = [myATMC: https://my.atmc.net/CSC/Login]
- *name* = myATMChttpsmyatmcnetCSCLogin_3



Log: IsStatementViewable

18-Feb-2012 07:34:52.832 PM **Stop timer: myATMChttpsmyatmcnetCSCLogin_3**

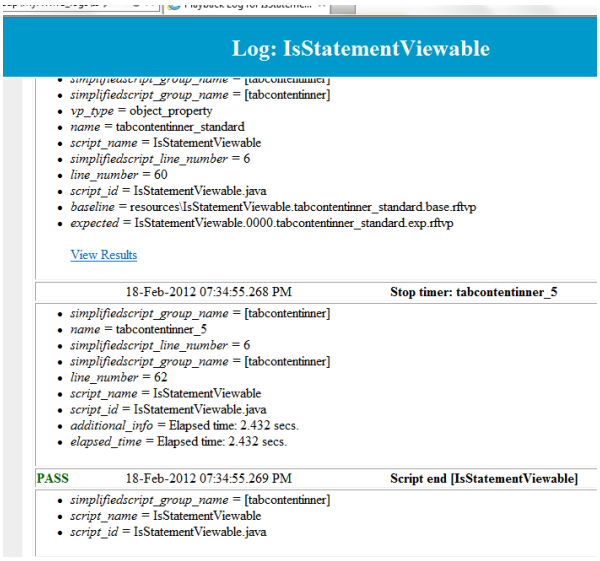
- *simplifiedscript_group_name* = [myATMC: https://my.atmc.net/CSC/Login]
- *name* = myATMChttpsmyatmcnetCSCLogin_3
- *simplifiedscript_line_number* = 4
- *simplifiedscript_group_name* = [myATMC: https://my.atmc.net/CSC/Login]
- *line_number* = 55
- *script_name* = IsStatementViewable
- *script_id* = IsStatementViewable.java
- *additional_info* = Elapsed time: 8.893 secs.
- *elapsed_time* = Elapsed time: 8.893 secs.

18-Feb-2012 07:34:52.835 PM **Start timer: tabcontentinner_5**

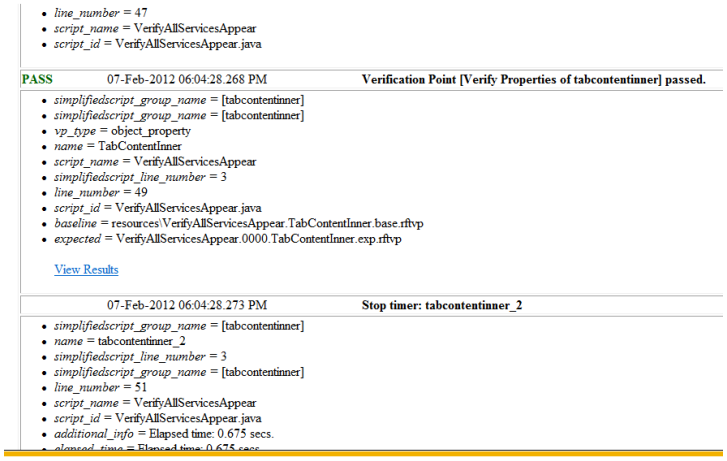
- *simplifiedscript_group_name* = [tabcontentinner]
- *name* = tabcontentinner_5
- *simplifiedscript_line_number* = 5
- *simplifiedscript_group_name* = [tabcontentinner]
- *line_number* = 58
- *script_name* = IsStatementViewable
- *script_id* = IsStatementViewable.java

PASS 18-Feb-2012 07:34:55.263 PM **Verification Point [Verify Properties of tabcontentinner] passed.**

- *simplifiedscript_group_name* = [tabcontentinner]
- *simplifiedscript_group_name* = [tabcontentinner]
- *vp_type* = object_property
- *name* = tabcontentinner_standard
- *script_name* = IsStatementViewable
- *simplifiedscript_line_number* = 6
- *line_number* = 60

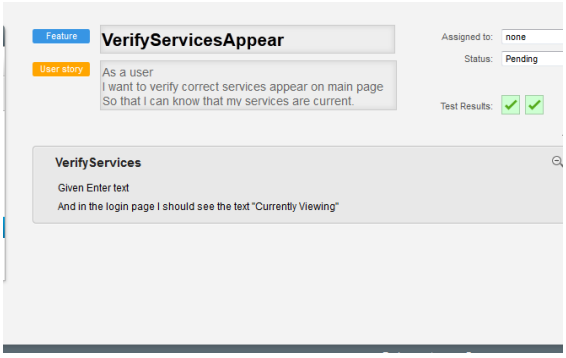


Log Results of 'Verify All Services Appear' Script

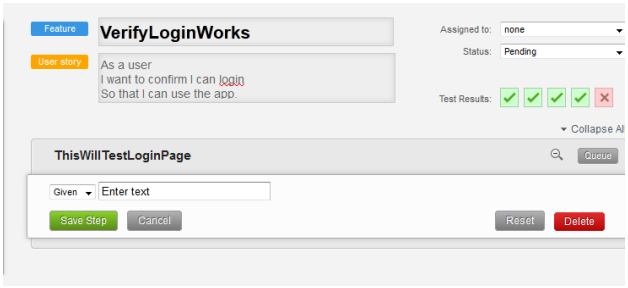


Appendix C – Test Scripts in Janova

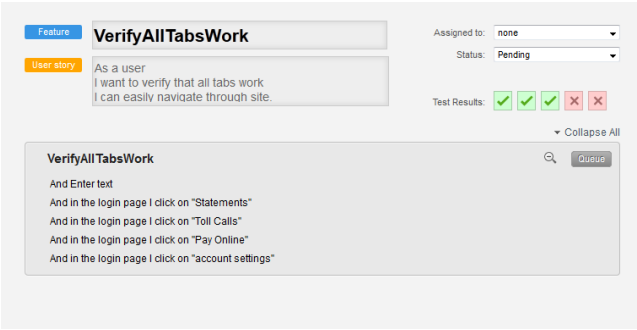
Test script - ‘Verify Services Appear’



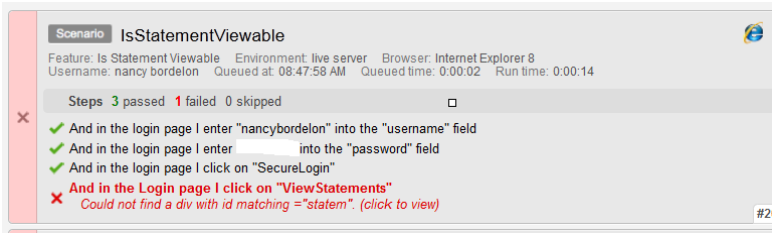
Test script - ‘Verify Login Works’



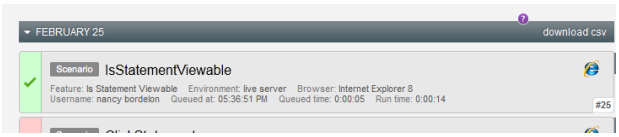
Test script - Verify Tabs Work



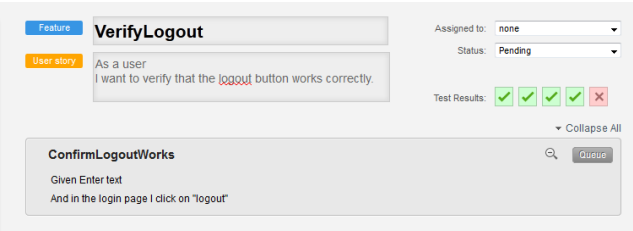
Test script - ‘Is Statement Viewable’ - Image of failed script because of how page element was being mapped



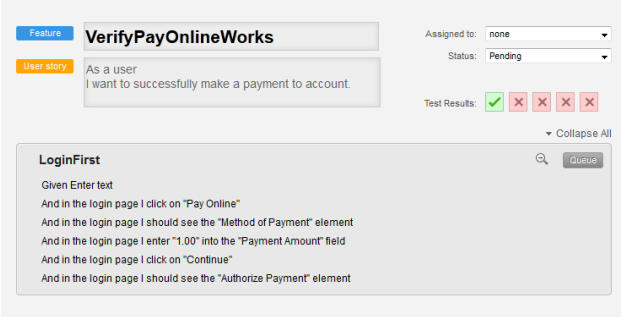
Test script - ‘Is Statement Viewable’



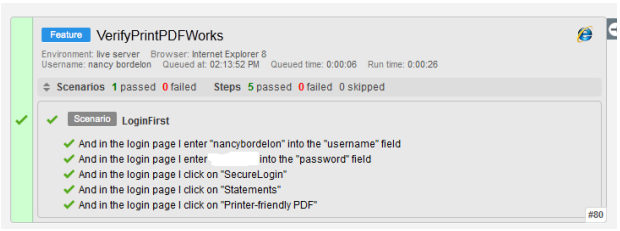
Test script - ‘Verify Logout’



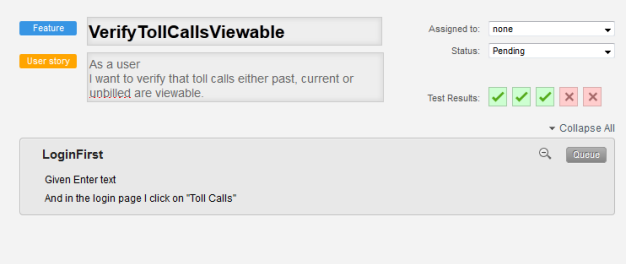
Test script - ‘Verify Pay Online Works’



Test script - ‘Verify Printer PDF Works’

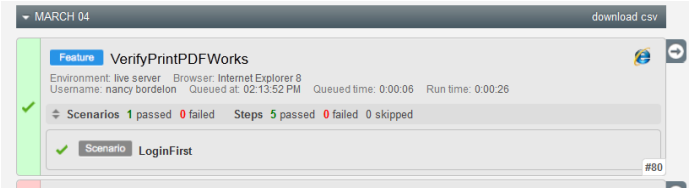


Test script - ‘Verify Toll Calls Viewable’

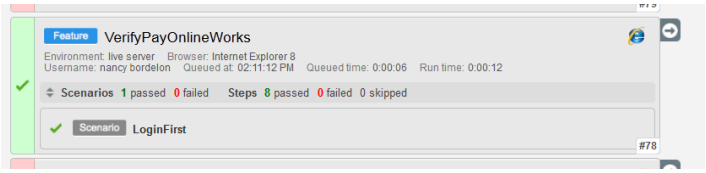

Feature
VerifyTollCallsViewable
User story
As a user
I want to verify that toll calls either past, current or unbilled are viewable.
Assigned to:
none
Status:
Pending
Test Results:
Collapse All
LoginFirst
Queue
Given Enter text
And in the login page I click on "Toll Calls"

Appendix D - Log Results in Janova

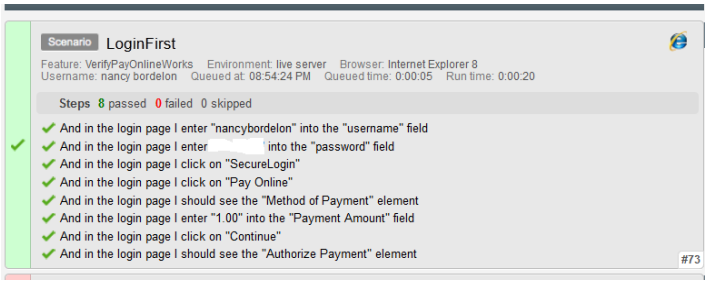
Log Results of ‘Verify Print PDF Works’ Script



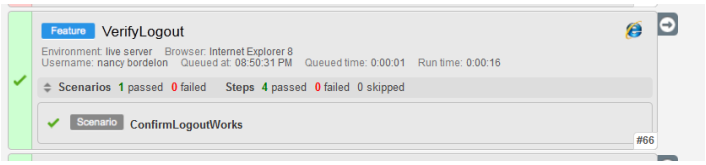
Log Results of ‘Verify Pay Online Works’ Script – This shows Scenario inside as part of ‘Verify Pay Online Works’ Feature



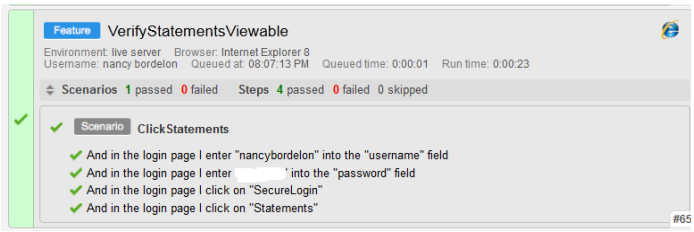
Log Results of ‘Login First’ which has steps for ‘Verify Pay Online Works’ Script



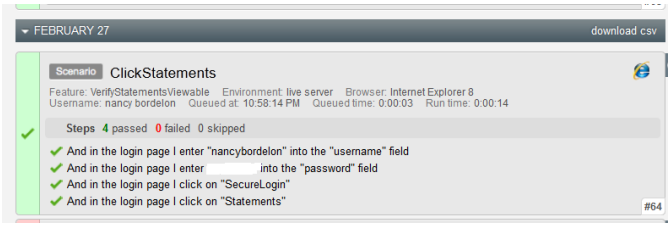
Log Results of ‘Confirm Logout works’ Script



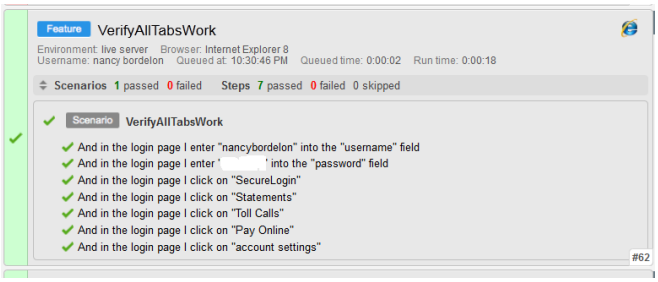
Log Results of ‘Verify Statements are Viewable’ Script



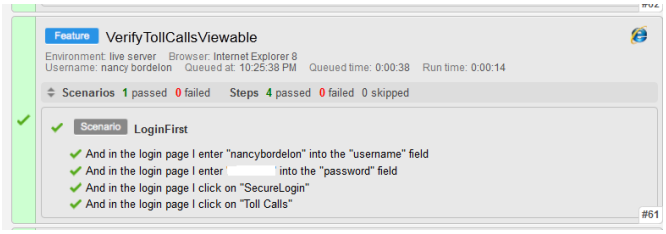
Log Results of ‘Verify Statements Viewable’ Script



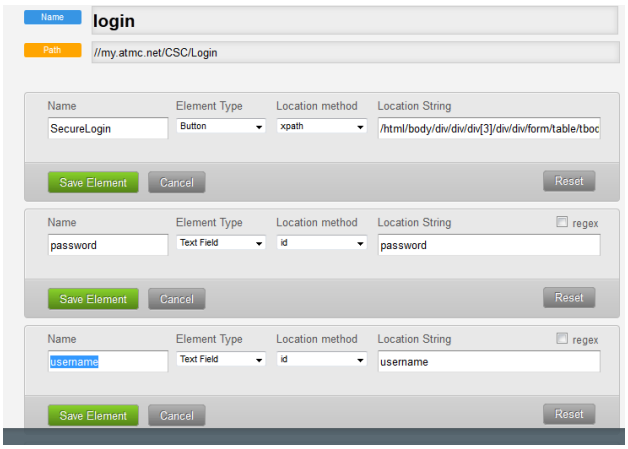
Log Results of ‘Verify All Tabs Work’ Script

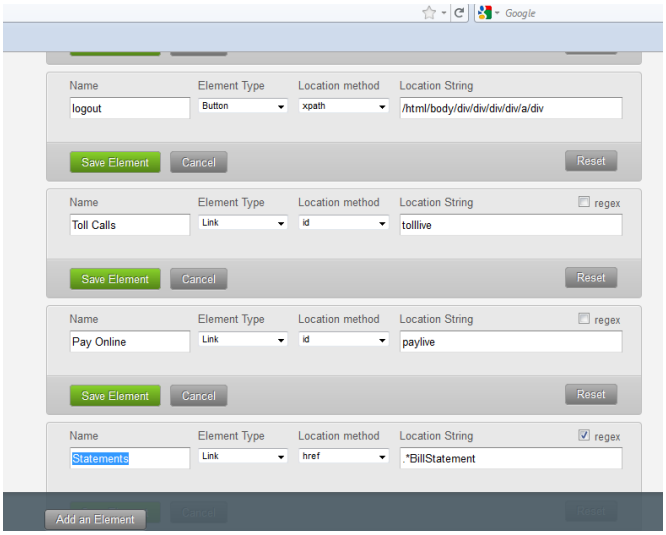


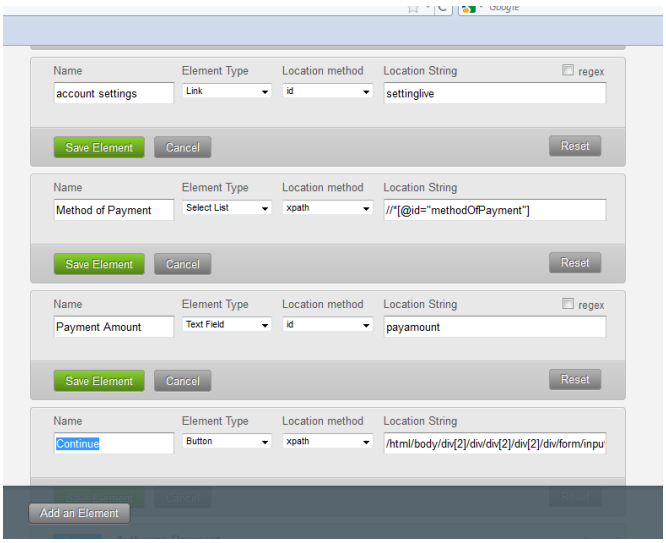
Log Results of ‘Verify Toll Calls Viewable’ Script

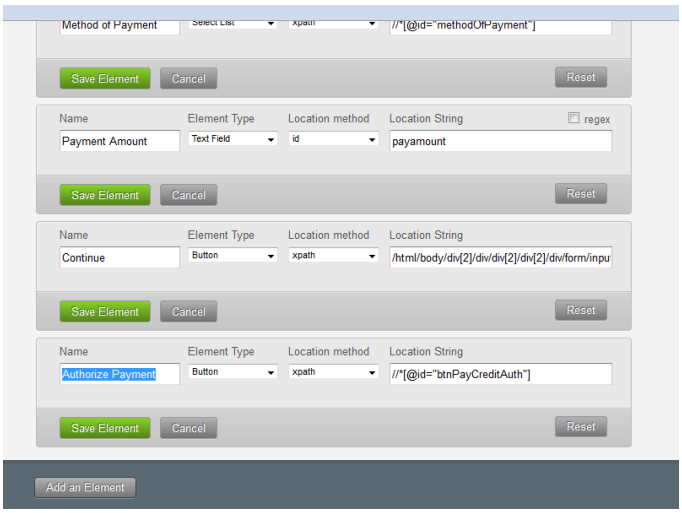


Below shows the Pages view which specifies how AUT web page elements are mapped


Name
login
Path
//my.atmc.net/CSC/Login
Name
Element Type
Location method
Location String
SecureLogin
Button
xpath
/html/body/div/div/div[3]/div/div/form/table/tboc
Save Element
Cancel
Reset
Name
Element Type
Location method
Location String
regex
password
Text Field
id
password
Save Element
Cancel
Reset
Name
Element Type
Location method
Location String
regex
username
Text Field
id
username
Save Element
Cancel
Reset


Google
Name
Element Type
Location method
Location String
logout
Button
xpath
/html/body/div/div/div/div/a/div
Save Element
Cancel
Reset
Name
Element Type
Location method
Location String
regex
Toll Calls
Link
id
tolllive
Save Element
Cancel
Reset
Name
Element Type
Location method
Location String
regex
Pay Online
Link
id
paylive
Save Element
Cancel
Reset
Name
Element Type
Location method
Location String
regex
Statements
Link
href
.*BillStatement
Add an Element


Name
Element Type
Location method
Location String
regex
account settings
Link
id
settinglive
Save Element
Cancel
Reset
Name
Element Type
Location method
Location String
Method of Payment
Select List
xpath
//*[@id="methodOfPayment"]
Save Element
Cancel
Reset
Name
Element Type
Location method
Location String
regex
Payment Amount
Text Field
id
payamount
Save Element
Cancel
Reset
Name
Element Type
Location method
Location String
Continue
Button
xpath
/html/body/div[2]/div/div[2]/div[2]/div/form/inpu
Add an Element


Method of Payment
//*[@id="methodOfPayment"]
Save Element
Cancel
Reset
Name
Element Type
Location method
Location String
regex
Payment Amount
Text Field
id
payamount
Save Element
Cancel
Reset
Name
Element Type
Location method
Location String
Continue
Button
xpath
/html/body/div[2]/div/div[2]/div[2]/div/form/inpu
Save Element
Cancel
Reset
Name
Element Type
Location method
Location String
Authorize Payment
Button
xpath
//*[@id="btnPayCreditAuth"]
Save Element
Cancel
Reset
Add an Element

Appendix E – Test Scripts in Ranorex

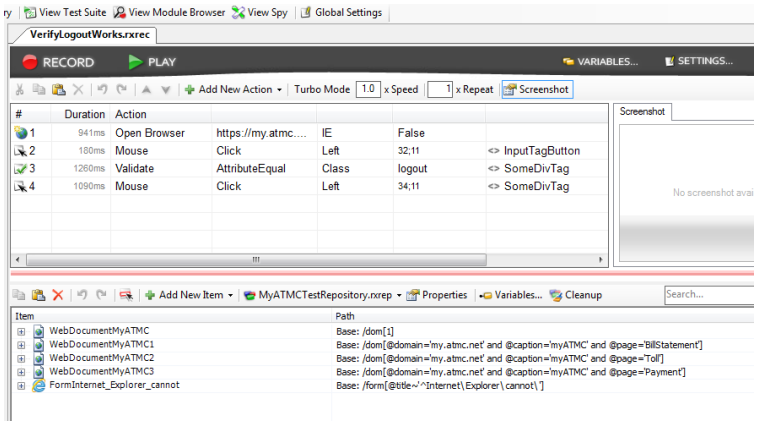
Code automatically generated for the ‘Verify All Services Appear’ test script in the tool

```
'////////////////////////////////////////////////////////////////////////
///////
'
' This file was automatically generated by RANOREX.
' DO NOT MODIFY THIS FILE! It is regenerated by the designer.
' All your modifications will be lost!
' http://www.ranorex.com
'
'////////////////////////////////////////////////////////////////////////
///////

Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.Text.RegularExpressions
Imports System.Drawing
Imports System.Threading
Imports WinForms = System.Windows.Forms

Imports Ranorex
Imports Ranorex.Core
Imports Ranorex.Core.Testing

Namespace MyATMCTest
    ''' <summary>
    ''' The VerifyAllServicesAppear recording.
    ''' </summary>
    <TestModule("c36fdfbe-14c5-4d7f-8962-6869a7b0b85e",
ModuleType.Recording, 1)> _
    Partial Public Class VerifyAllServicesAppear
        Implements ITestModule

        ''' <summary>
        ''' Holds an instance of the MyATMCTestRepository repository.
        ''' </summary>
        Public Shared repo as MyATMCTestRepository =
MyATMCTestRepository.Instance

        Shared _instance as VerifyAllServicesAppear = new
VerifyAllServicesAppear()

        ''' <summary>
        ''' Constructs a new instance.
        ''' </summary>
        Sub New()
        End Sub

        ''' <summary>
```

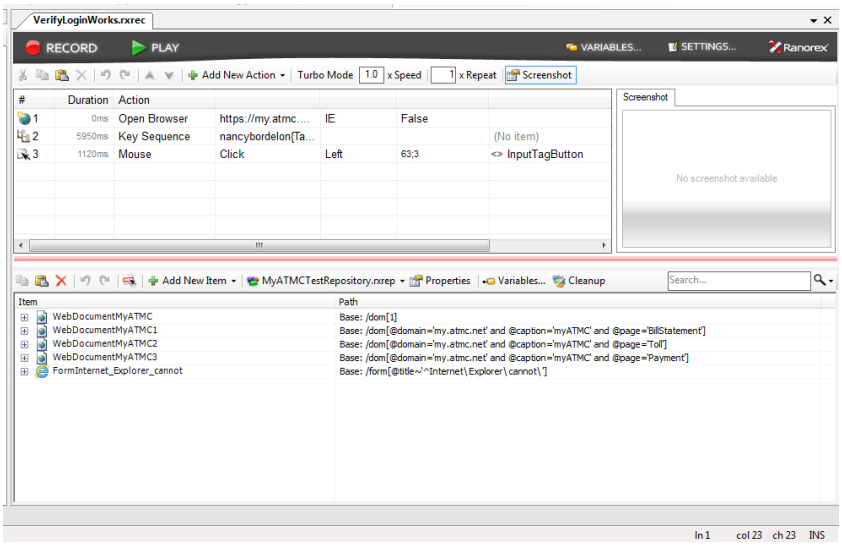
```
        ''' Gets a static instance of this recording.
        ''' </summary>
        Public Shared ReadOnly Property Instance As
VerifyAllServicesAppear
            Get
                Return _instance
            End Get
        End Property

#Region "Variables"

#End Region

        ''' <summary>
        ''' Starts the replay of the static recording <see
cref="Instance"/>.
        ''' </summary>
        <System.CodeDom.Compiler.GeneratedCode("Ranorex", "3.2.2")> _
        Public Shared Sub Start()
            TestModuleRunner.Run(Instance)
        End Sub

        ''' <summary>
        ''' Performs the playback of actions in this recording.
        ''' </summary>
        ''' <remarks>You should not call this method directly, instead
pass the module
        ''' instance to the <see cref="TestModuleRunner.Run(Of
ITestModule)"/> method
        ''' that will in turn invoke this method.</remarks>
        <System.CodeDom.Compiler.GeneratedCode("Ranorex", "3.2.2")> _
        Sub Run() Implements ITestModule.Run
            Mouse.DefaultMoveTime = 600
            Keyboard.DefaultKeyPressTime = 100
            Delay.SpeedFactor = 0.5

            Init()

            Report.Log(ReportLevel.Info, "Website", "Opening web site
'https://my.atmc.net/CSC/Login' with browser 'IE' in normal mode.", new
RecordItemIndex(0))
            Host.Local.OpenBrowser("https://my.atmc.net/CSC/Login",
"IE", "", False, False)
            Delay.Milliseconds(100)

            Report.Log(ReportLevel.Info, "Validation", "Validating
AttributeEqual (Visible='True') on item
'WebDocumentMyATMC.DivTagContent.InputTagButton'.",
repo.WebDocumentMyATMC.DivTagContent.InputTagButtonInfo, new
RecordItemIndex(1))

Validate.Attribute(repo.WebDocumentMyATMC.DivTagContent.InputTagButtonIn
fo, "Visible", "True")
            Delay.Milliseconds(100)
```

```
            Report.Log(ReportLevel.Info, "Mouse", "Mouse Left Click item
'WebDocumentMyATMC.DivTagContent.InputTagButton' at 41;10.",
repo.WebDocumentMyATMC.DivTagContent.InputTagButtonInfo, new
RecordItemIndex(2))

repo.WebDocumentMyATMC.DivTagContent.InputTagButton.Click("41;10", 220)
            Delay.Milliseconds(0)

            Report.Log(ReportLevel.Info, "Validation", "Validating
Exists on item
'WebDocumentMyATMC.DivTagContent.DivTagTabcontentinner'.",
repo.WebDocumentMyATMC.DivTagContent.DivTagTabcontentinnerInfo, new
RecordItemIndex(3))

Validate.Exists(repo.WebDocumentMyATMC.DivTagContent.DivTagTabcontentinn
erInfo)
            Delay.Milliseconds(100)

        End Sub

#Region "Image Feature Data"
#End Region

    End Class

End Namespace
```

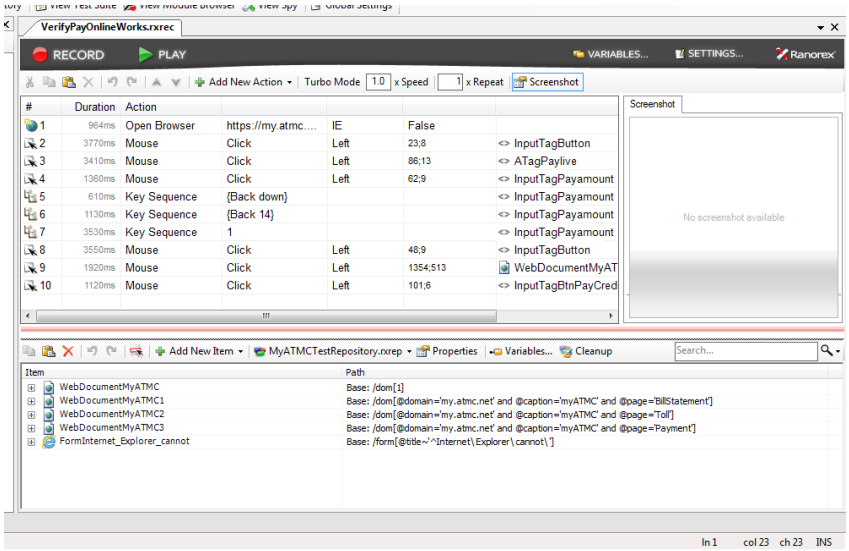
Test script - ‘Verify logout works’



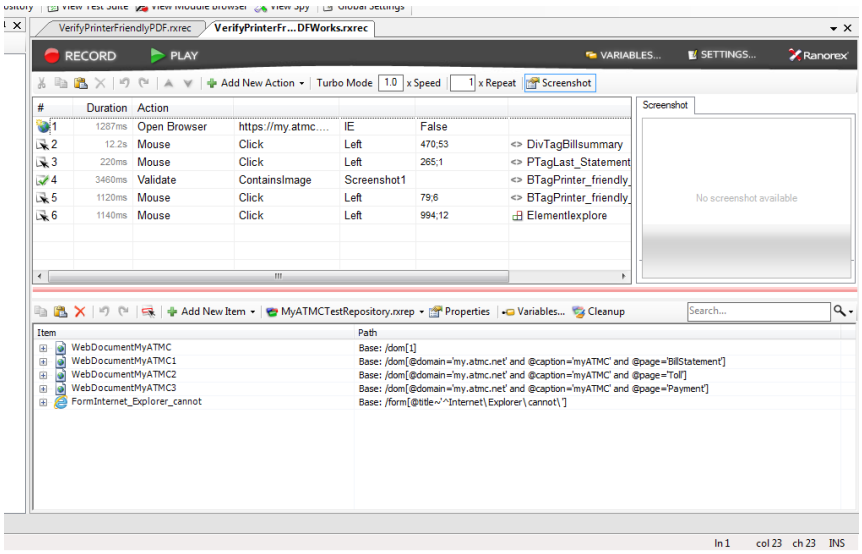
Test script - ‘Verify login works’



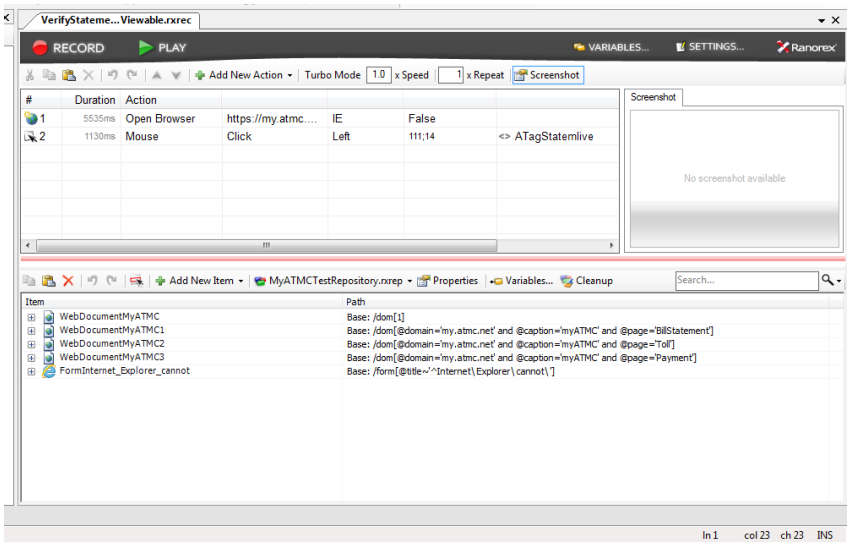
Test script - ‘Verify pay online’ works



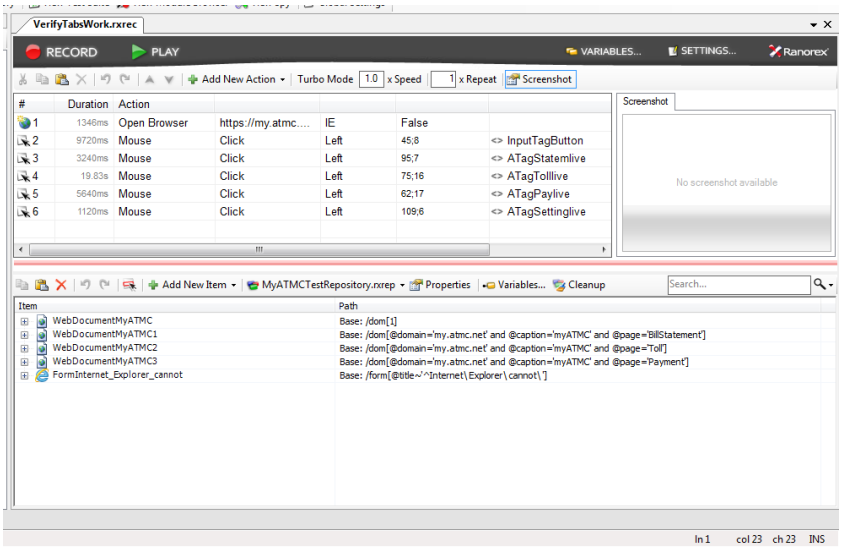
Test script – ‘Verify printer friendly PDF works’



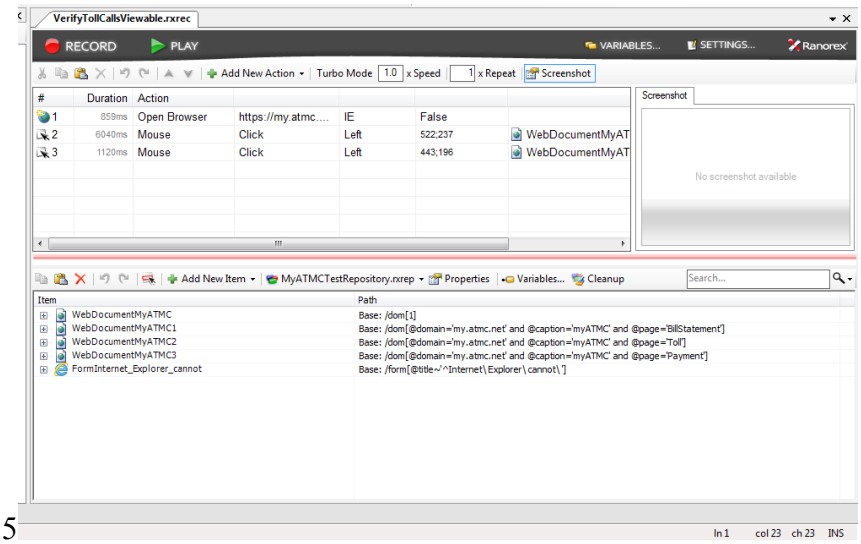
Test script – ‘Verify statements are viewable’



Test script - ‘Verify all tabs work’



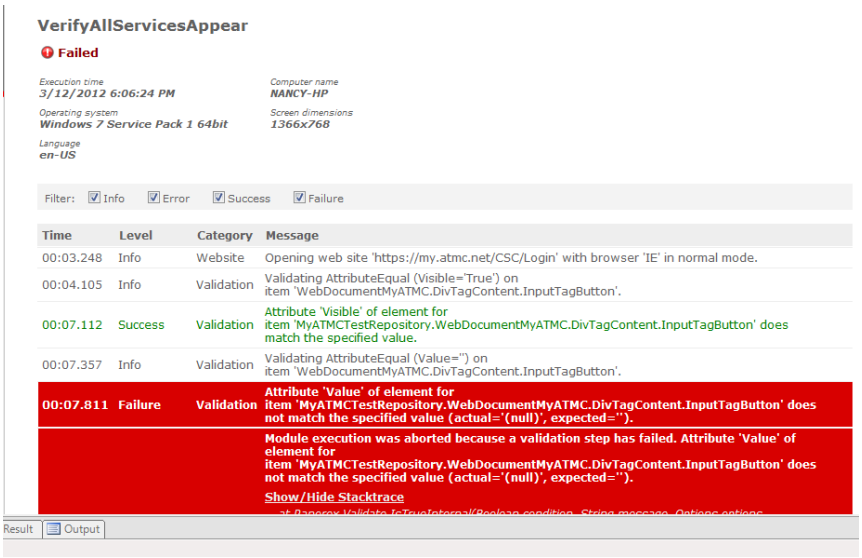
Test script - ‘Verify toll calls are viewable’



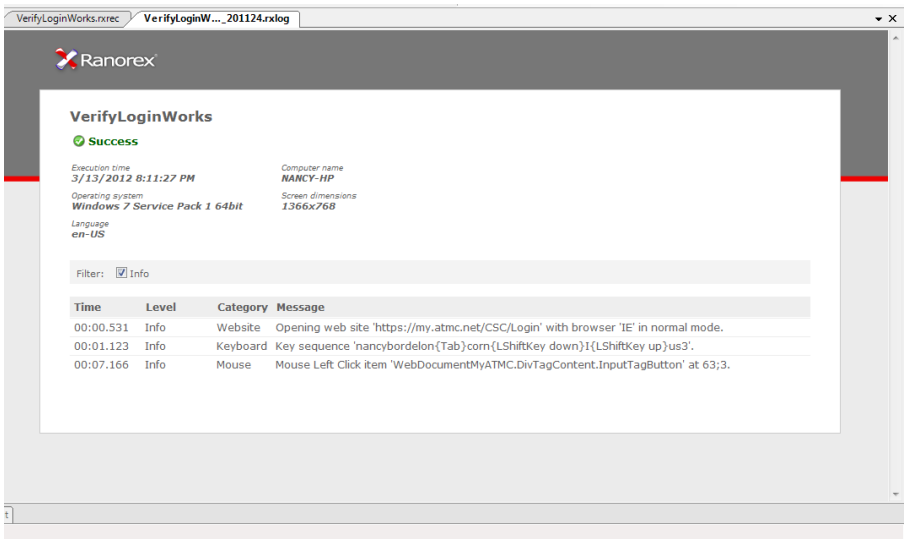
5

Appendix F – Log Results in Ranorex

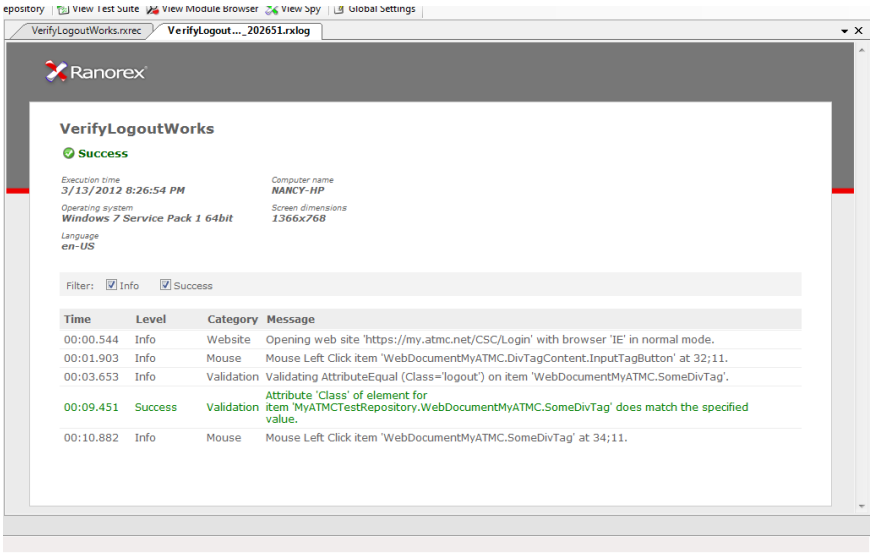
Ranorex - First failed log for 'Verify All Services Appear'. This was due to the script needing to be paused for the website login credentials.



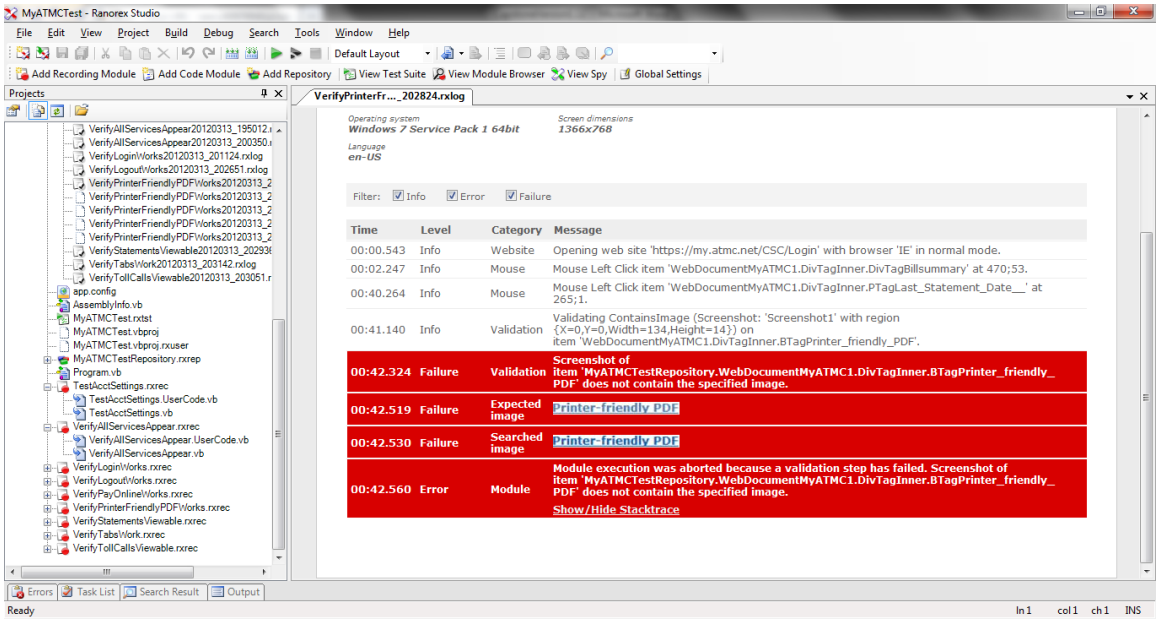
Log Results of 'Verify Login Works' Script



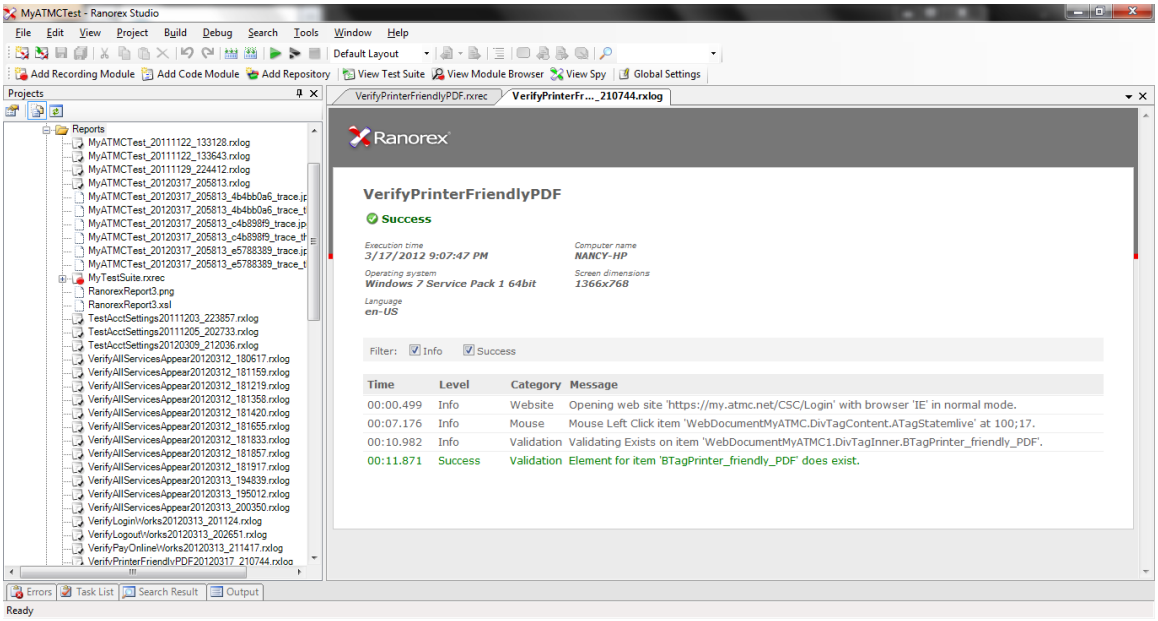
Log Results of 'Verify Logout Works' Script



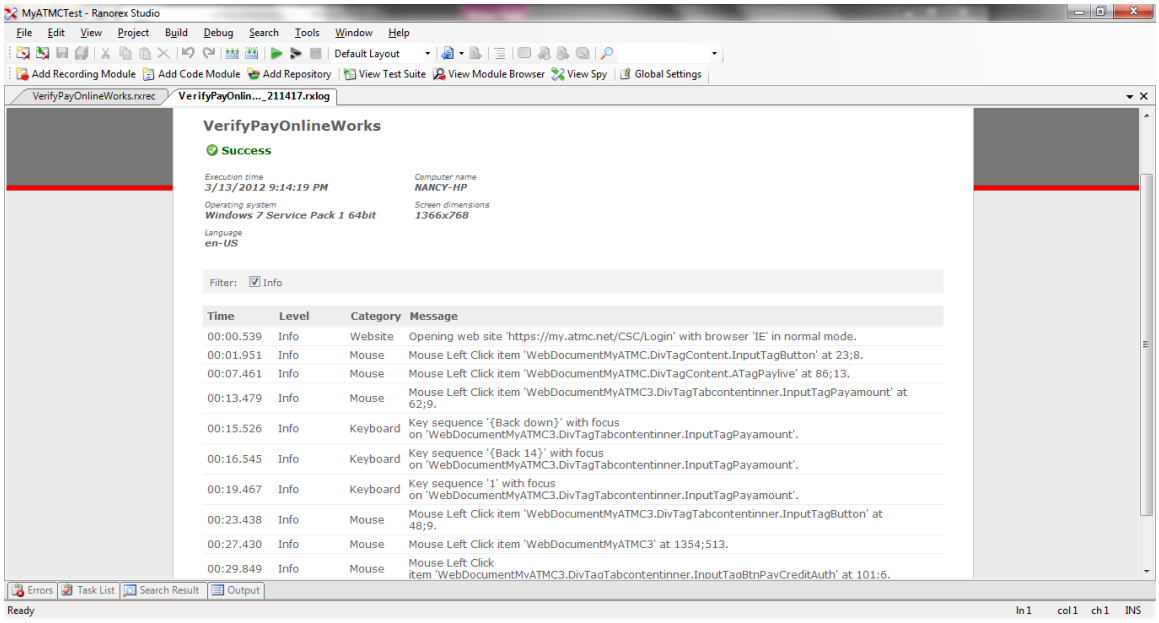
Log Results of failed ‘Verify Printer Friendly PDF Works’ Script



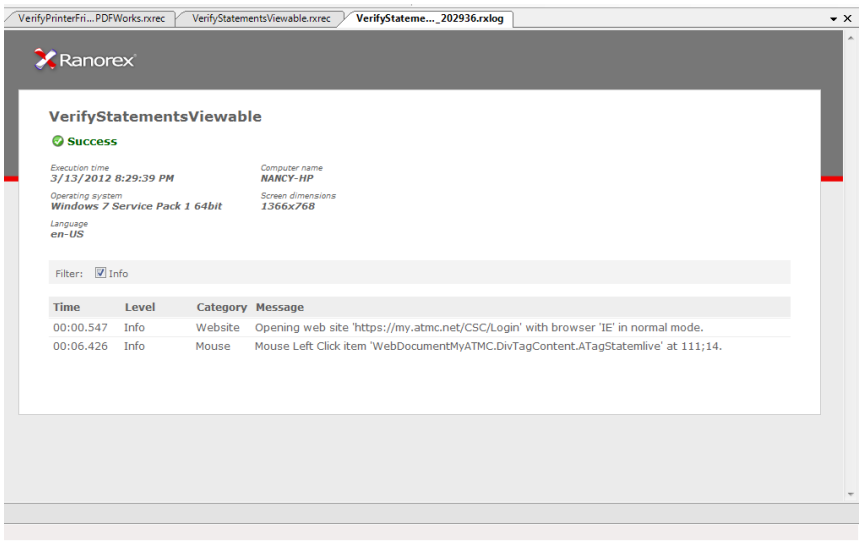
Log Results of ‘Verify Printer Friendly PDF Works’ Script



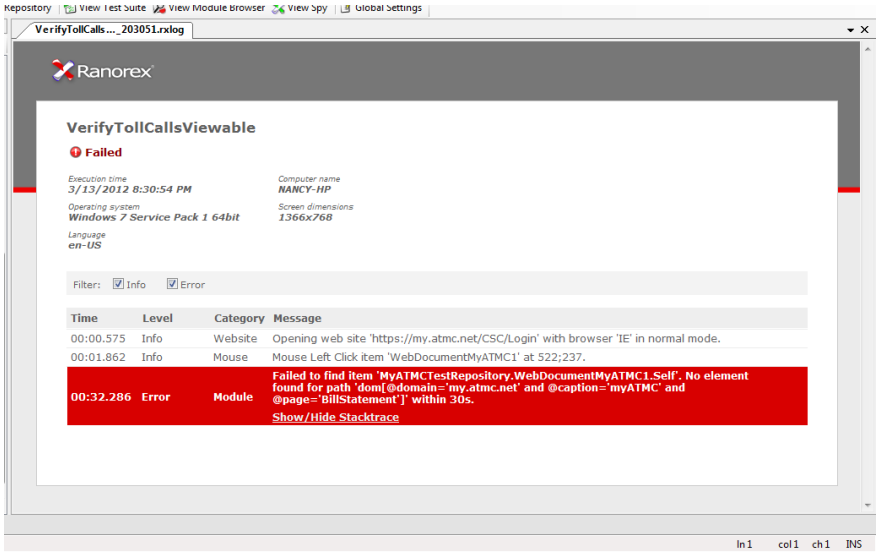
Log Results of 'Verify Pay Online Works' Script



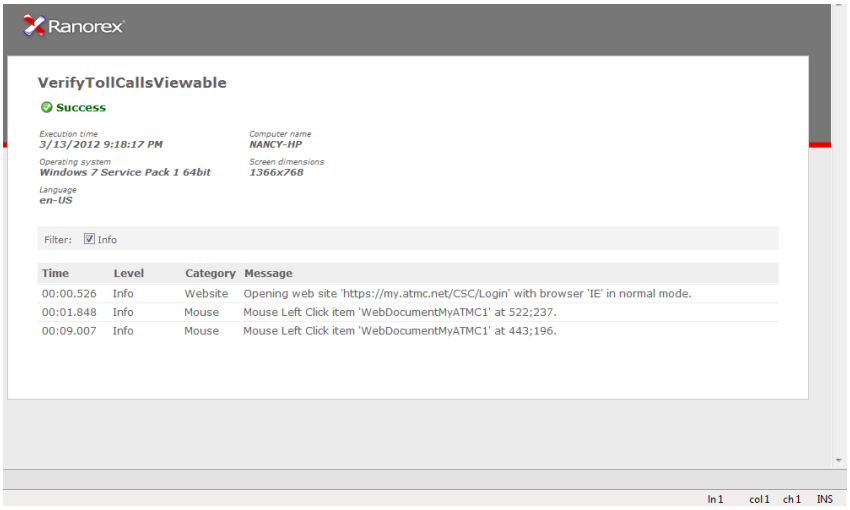
Log Results of 'Verify Statements Viewable' Script



Log Results of failed ‘Verify Toll Calls Viewable’ Script



Log Results of ‘Verify Toll Calls Viewable’ Script



Log Results of 'Verify Tabs Work' Script