International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 04 | Apr 2020 www.irjet.net p-ISSN: 2395-0072

Framework for Automation Testing to Audit System Performance in Windows Operating System

Mayank Agrawal[1], Chethana R Murthy[2]

[1]*Dept. of Information Science and Engineering, RV College of Engineering, Bangalore*
[2]*Dept. of Information Science and Engineering, R.V. College of Engineering, Bangalore, India*

Abstract - *Increasing application demands on processors can result in high operating temperatures, a risk of CPU throttling and poor performance. Balancing operating speed with battery life is essential to delivering the optimal experience users demand today. To optimize overall performance of a system, an efficient strategy to measure system performance is required. This paper presents a novel method to develop a framework which describes and implements a way to monitor system performance remotely and autonomously. The application will provide autonomous control to the user to control the target system to initiate actions related with measurements. This tool will allow the user to launch test cases to ensure that the application meets its specification and performs the intended functions via GUI automation, which are necessary to monitor the system. Although due to inherent automation limitations experiment results show that, with automation testing reduces almost four times manual effort.*

Key Words: **Software development, Automation, WPA, PyWinAuto, Tkinter, Software metrics.**

1. INTRODUCTION

Technology is growing at a rapid rate with constantly being built and improved upon. More and more new features are added to serve user satisfaction. Today a semiconductor chip device can support multiple features ranging from low power consumption services such as messaging, phone, appointment, alarm, entertainment, Email messaging to high power consumption services such as gaming, high speed internet surfing, etc. Models are released to the market on a daily basis with improved or brand-new features. Most of these features are largely implemented using software development [1], [2]. These applications rely heavily on underlying hardware components to harness extreme processor functionality. Every feature of each new model must be tested prior to its release. But due to rapid feature changes and development most of the developer oversee the underlying hardware. They often miscalculate the small power leakage which can eventually affect the system performance [1]. The interaction of features with the hardware sensors must be checked, so as to ensure their proper integration. User-level functional tests are crucial to both developers and testers to reduce consumer dissatisfaction and technical assistance cost.

Functionality testing uses many use-case scenarios. These use cases are mostly user centric and tend to imitate an average user frequent action. Each test case (TC) is a sequence of steps that performs a specific task or a group of tasks. TC also specifies the expected results. Users normally manually perform TCs. To estimate the correct measurements a TC may be repeated several times over the successive life cycle of software development. As most of these test case executions are repeated tasks, manual testing becomes both time consuming and error prone. An automated test case can automatically produce the steps that would be performed manually by the users.

This paper presents a novel object-oriented framework tailored to support Automated Test case execution for user-level functional testing of windows-based systems. The test automation framework allows automation of functional TCs necessary to measure system performance. The framework aims to produce a product family to achieve productivity gain to both the system users and their dependents. Once a TC is automated, it can be executed as many times as needed, through all the different build versions of the system and thus reducing the time to analyse correct performance measurement of the system. The main contribution of this paper consists in the analysis of system performance via various windows systems by automating the appropriate TCs remotely and autonomously.

The remainder of this paper is organized as follows: Section 2 reviews related work; the structure of the automation framework is described in Section 3; Section 4 summarizes implementation details; Section 5 analyses results; Section 6 describes limitations and feature enhancements and finally our conclusions are drawn in Section 7.

2. RELATED WORK

2.1 Common Practices

While manual execution of Test Case execution is still current practice, adaptation of test automation for unit and regression testing is featured. Although many automation tools are already available in market, full automation is still a dream. For e.g., more than 40 per cent of TCs cannot be designed for automation in traditional user-level test suites. While the strategies vary from one user to the other, test automation has been incrementally introduced at the user-levels, resulting in the effort required to carry out tests. The main approaches employ either inhouse developed test suites, such as PTF [3], or third-party test case systems, such as TestQuest Pro (R) [4]. We have devised a novel test case automation infrastructure based on average user activities on

the system. These test case infrastructures are the key stone of the framework developed.

2.1 Related research topics

Related approaches on test automation address two basic goals: test case execution and test case analysis.

The automation of test implementation and outcome interpretation aims at generating software objects capable of executing test suites and contrasting the results obtained with those obtained by running the same using manual methods [8]. Different works seem to indicate that there is not so far, a reliable and complete solution for test automation challenges [9]. On the contrary, distinct successful approaches are reported [8], [10], [11], [12], [13]. A quantitative trade-off analysis [13] will be done to determine the economic feasibility of the project automation. Since high iteration frequency is a requirement for automating a TC, specific drawbacks should be resisted, such as misjudging the effort needed for manual implementation or underestimating the percentage of tasks potentially tailored for automation [17].

Given that test automation frequently includes creating software for testing applications, an alternate solution to obtaining a better trade-off is to facilitate the reuse of applications while developing test ware. Object-oriented systems are modular artefacts in software that will help the creation in test ware [17]. JUnit is a common illustration of a system extended to the software application domain [14].

Because there is a trade-off between tautology and reuse efficiency, domain-specific systems (such as JUnit) are projected to result in a lower reuse rate than application-specific models. This was the driving force behind the introduction of a modern application-specific architecture geared to cell phones. The review needs data to measure the effect of the program on test automation. That is why this paper's key objective is to evaluate the quantitative effect of an application-specific architecture on the implementation of the state-of-the-art technologies in real life.

3. DESIGN DESCRIPTION

The defined system is an object-oriented architecture designed to automate practical test case implementation for windows-based systems at different consumer-level. To automate a test case, it requires the necessary resources, but this method designed is ultimately procedural. In other terms, it deals with the automation of test execution, not automated test production. The framework exploits the GUI elements present in the windows system to automate the test case execution, which works by locating and searching the required GUI element and issuing commands to launch the desired actions.

This method has its drawback where the model must rely on the specific version of the system to find the same set of GUI elements. Thus, making the framework highly dependent. The framework works as a higher-level abstraction of similar user actions.

3.1 Low- Level Implementation Infrastructure

In order to interface with the Windows PC, the framework uses Python Library. Python's Tkinter Library is used to render an application programming interface (API) that allows the user to simulate desired events remotely on target system under test [6]. WiFi connections are used to establish remote connection to target system from the Host device. PyWinAuto Library is used to automate GUI actions from the host device to the target [5]. The rich library of PyWinAuto and use of WiFi to connect host and target device allows hosts to simulate several events like key pressing, mouse action, selection/deselection, etc. remotely.

However, complete automation cannot be achieved by Python Libraries due to inherent limitations posed by the system. Command Language Interface (CLI) is used to modify register level entries [7]. CLI provides privileged accesses of the system to launch scripts to measure and monitor performance measurements. Python combined with CLI allows the framework to achieve full automation with benefits like scheduling scripts, threading process, terminating process.

3.2 High Level Implementation Infrastructure

The Tool is a system of modules and each module contains the use cases for the system. The actor at the host side interacts with the rendered API by creating a port for connection, selecting Operating System version, selecting testcase variations, etc.

The port at the host side listens for the incoming connection from the target system. Once the connection is established and hello packet is send to ensure handshake protocol, the command is transferred to target system containing the test case name, number of iterations for which the test case is to be run. The command is then parsed at the server side.

A concurrent thread is also created to establish connection to DAQ. A Data Acquisition System (DAQ) used in this framework is Windows Performance Analyzer (WPA) tool provided by Windows Operating System. WPA tool displays performance of system in form of graphical format. It helps in monitoring the system performance and allows the user to generate trace to determine issues related with performance of system [18].

3.3 Detailed design of Automation Framework

In this section, further details, algorithmic design and an in-depth explanation of each of the modules is provided. Low-level components and subcomponents of various parts are also described.

3.3.1 Client end functionalities and actions

The client works by rendering a GUI for test case execution. After receiving all the information viz. IP address of target system, initial check of whether the WPA is in stable state to collect data or not. The GUI runs a thread to create a target socket client and waits indefinitely for the connection. At the

same time a second thread is launched which is specific for the WPA.

This thread starts with configuring the WPA and checks if the WPA is in a stable state to collect data or not. If the WPA is found in improper state, it is reconfigured. If the WPA is found in stable state, a start acquisition command is sent to the WPA.

3.3.2 Target end functionalities and actions

The Target side application waits for the connection from the host, before connection establishment the system performs some predefined settings. These predefined settings are necessary to be performed every time the test case is executed. These are the settings necessary to set, in order to collect correct measured values. For example, before launching any test case it is necessary to switch on airplane mode. This is necessary because we do not want any unwanted applications (not associated with test case) to be active. Similarly, for any test case execution we want only those applications to be activated which are essential for the current test case. Any unnecessary application is closed to avoid any unwanted power leakage.

Once the current test case execution is finished. The scheduled script is launched automatically, this script first looks for any open application and tries to close it. Now, when the system is ready to launch the next test case and for that purpose it switches off the airplane mode and waits for the socket server connection. The same steps of execution are repeated till the list of test cases is exhausted.

4. IMPLEMENTATION DETAILS

The programming language that is chosen to model the tool is Python. Python is the most essential and required component of the project. Python is one of the most useful languages which is currently being widely used because of its extensive coverage in support tools and structures for fast development. In addition to the Python programming language, batch scripting commands are also used.

The Command line language support is necessary because there are many functionalities which can be done only at root level. This root level access cannot be done using high level language like python.

PyWinAuto is used for GUI automation, this library contains tons of application interfaces to launch windows-based tasks. The module provides almost every functionality which can allow GUI based automation with minimal lines of code. At its simplest it allows the user to send mouse and keyboard actions to windows dialogs and controls. It emulates user actions like mouse clicks/movements and keyboard key presses. Pywinauto provides an important benefit of accessing the GUI control elements with the help of attributes. The knowledge of PyWinAuto is necessary to build the tool.

Python provides the Tkinter framework which is the only framework that's built into the Python standard library. Visual elements are rendered using native operating system elements (like windows), hence the applications built with Tkinter look like they belong on the platform where they're run. Although Tkinter is considered the de-facto Python GUI framework, it's not without criticism.

WPA is used to monitor system performance. WPA provides a nice GUI to study trace of system performance. It also projects detailed information about system platform states, devices states, software activity, CPU utilization, memory utilization, and other system events.

5. EXPERIMENTAL RESULTS

5.1 Results

The outcome of this tool would be performance measurement values which shall match with the values obtained from manual measurement. The outcome of this project could be achieved by following the methodology and achieving the objectives in the described order.

The major outcomes of this project would be:

1. All the prerequisite settings are performed before launching any test cases.
2. To ensure correct test cases are executed on target system.
3. To generate correct performance measurement values using WPA in a simple and readable manner.

The outcomes and results of the objectives set:

1. A successful build of the Automation Tool on to the Windows machine is the outcome of the first objective. The build instructions of the Automation Tool must be understood in order to execute them on the machine.
2. Importing a successfully built PywinAuto Library for further automation and development of the GUI using Tkinter is the outcome of the second objective. This objective is time consuming and is achieved by executing several modules of PywinAuto and Tkinter.
3. Generating Graphs from the WPA profile to observe system behavior to identify problems.

Outcome from the project is to build an efficient and reliable system which can automate the defined tasks effectively. The results obtained after the analysis should be satisfactory enough to make the system more reliable.

5.2 Comparative Analysis

The current application which is developed could be compared to the existing methods of generating the performance measurements values using the WPA tool. Existing techniques rely on manual intervention of task performance and would make the whole process more tedious and exhausting.

The major existing limitation solved using the current application are:

1. Requires a lot of manual effort to carry out a small set of tasks.

2. Enough time was lost in the process of creating and studying results. Repeated actions were performed if some actions led to absurd results.
3. Overcoming these limitations was one of the major concerns for the product and this would enable users using this tool to:
    a. Repeated task execution was automated saving a lot of user time.
    b. Analysis of data generated by WPA was made easier by launching the event using a single click.

This tool would pose a great advantage to the users of system as it saves user's time to obtain the same set of results. This would help the users to analyze the system performance by measuring the real time data while the system apps are running. Now users can achieve more accurate results within a short span of time.

6. Conclusion

The tool aims to achieve certain tasks completion by allowing GUI automation on the target side. It will allow the QA team to assess the rapid changes in the system by releasing the system within a short interval of time. The tool allows remote execution of use-case in target system which makes the overall system more autonomous and independent. It provides a single platform to control all the activities needed to obtain performance measured values. It also allows the user to monitor the task performance through the interface. The major aim is to obtain productivity by performing the repeated actions within a short interval of time. It increases the quality of results obtained and helps in robustness of the system.

REFERENCES

[1] Chunlei Shi, Brett Walker, Eric Zeisel, Brian Hu and Gene McAllister. D.: A Highly Integrated Power Management IC for Advanced Mobile Applications. In: Proc. In IEEE Custom Integrated Circuits Conference (CICC), pp 1-3 (2006)

[2] Emmannuel Ofori-Attah, Xiaohang Wang, Michael Opoku Agyeman. D.: A Survey of System Level Power Management Schemes in the Dark-Silicon Era for Many-Core Architectures. In: Proc.: EAI Endorsed Transactions on Industrial Networks and Intelligent Systems. pp (2-3)

[3] Esipchuk, I., Validov, D.: PTF-based Test Automation for JAVA Applications on Mobile Phones. In: Proc. IEEE 10th International Symposium on Consumer Electronics (ISCE), pp. 1–3 (2006)

[4] Test Quest, Test Quest Pro (2006) available at http://www.testquest.com.

[5] PywinAuto, for UI automation. Author notes, available at https://pywinauto.readthedocs.io/en/latest/getting_started.html.

[6] Python Tkinter Library, for user interface rendering. Documentation, available at https://docs.python.org/3/library/tk.html.

[7] Windows register manual, for modifying register level entry. Available at, https://help.comodo.com/topic-159-1-290-3248-.html.

[8] Pretschner, A., et al.: One Evaluation of Model-Based Testing and its Automation. In: Proc. International Conference on Software Engineering, pp. 398–401 (2005).

[9] Dalal, S.R., et al.: Model-Based Testing in Practice. In: Proc. International Conference on Software Engineering, pp. 1–6 (1999).

[10] Bredereke, J., Schlingloff, B.: An automated, Flexible Testing Environment for UMTS. In: Proc. 14th IFIP TC6/WG 6.1 International Conference on Testing of Communicating Systems, pp. 90–94 (2002).

[11] Heikkil¨a, T., Tenno, P., V¨a¨an¨anen, J.: Testing Automation with Computer Aided Test Case Generation. In: Proc. 14th IFIP TC6/WG 6.1 International Conference on Testing of Communicating Systems, pp. 215–216 (2002).

[12] Chi, C., Hao, R.: Test Generation for Interaction Detection in Feature-Rich Communication Systems. In: Proc. 17th IFIP TC6/WG 6.1 International Conference, TestCom, pp. 242–257 (2005).

[13] Gamma, E., Beck, K.: JUnit specification (2006) available at http://www.junit.org.

[14] Berner, S., et al.: Observations and Lessons Learned from Automated Testing. In: Proc. International Conference on Software Engineering, pp. 575–579 (2005).

[15] Xia, S., et al.: Automated Test Generation for Engineering Applications. In: Proc. International Conference on Automated Software Engineering, pp. 285–286 (2005).

[16] Ramler, R., Wolfmaier, K.: Economic Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost. In: Proc. International Workshop on Automation of Software Test, pp. 88–91 (2006).

[17] Window Performance Analyzer Tool to analyze System Performance.