University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Master of Science in Computer Science	Spring semester, 2016 Open
Writer: Dongjing Liu	.... (Writer's signature)
Faculty supervisor: Morten Mossige	
Thesis title: Fully Automated Graphical User Interface（GUI）Testing With Virtual Machines	
Credits (ECTS): 30	
Key words: Fully automation GUI testing Virtual machines Test case scheduling Test case prioritization	Pages: 93 Enclosure: A zip attachment Stavanger, 15 June 2016

Fully Automated Graphical User Interface(GUI) Testing with Virtual Machines

A DISSERTATION PRESENTED
BY
DONGJING LIU
TO
THE DEPARTMENT OF COMPUTER SCIENCE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF COMPUTER SCIENCE
WITH SUPERVISOR
MORTEN MOSSIGE

UNIVERSITY OF STAVANGER
STAVANGER, NORWAY
JUNE 2016

ACKNOWLEDGMENTS

Here and now, I would like to express my sincere thanks to all who have helped me make this thesis possible and better. Firstly, I am deeply grateful to my honorable supervisor, Morten Mossige, who has checked through my thesis with patience and has given me instructive suggestions.
Then special thanks also goes to the staff in ABB Bryne, especially, Andreas Skaar, who is very kind and generous in offering advice, as well as to the teachers and professors who have taught me over the past two years of study in University of Stavanger.
Finally, I am very grateful to my lovely friends and families who have offered selfless support to me.

Abstract

An approach, entitled fully automated GUI testing with virtual machines is proposed in the thesis. Currently, GUI testing relies a lot on human involvement due to its unique properties. The thesis tries to address the problem by locating GUI testing in virtual machines. Through automating virtual machines, the operations involve in testing are automated. Two virtual machine applications, respectively VMWare workstation and Hyper-V, are studied in the thesis. Based on both theoretical analysis and practical use, VMWare workstation is proven to be more suitable for GUI testing.

The result of the implementation shows obviously that the performance of fully automated GUI testing with virtual machines is much more efficient than manual tests. Especially, when multiple test cases are performed in parallel, the efficiency can be enhanced significantly.

An optimized solution is proposed to further shorten the round trip time, which is named prioritized test case scheduling. The optimized solution aims to detect all underlying faults in the shortest time. With both duration and priority introduced to a test case, the weights of all test cases over final goal is calculated by Analytic Hierarchy Process (AHP) method. Through a simple example study, the result of the optimized solution is proven to be more positive than only test cases duration scheduling and only test cases prioritization.

Contents

Listing of figures

List of Tables

1
Introduction

This chapter will give a brief description of the thesis, starting with essential background information to the motivation for this thesis. Also basic concepts concerning this thesis are explained, The structure of entire thesis is presented at the end of this chapter .

1.1 Background and Motivation

To date, there are numerous Graphical User Interface (GUI) applications emerging in both individuals' daily lives and various industrial fields. Undoubtedly, the existence of GUI applications bring great convenience to users' operations. However, the complexity also increases difficulties to software testing. As for GUI testing, how to guarantee those GUI applications perform properly as expected is critical in software development life cycle. A simple example of GUI testing is shown as figure 1.1.1, the outcome of a click on **Open** under **File** menu should be the *Open* windows as the picture shows, any other outcome indicates that the GUI application does not behave as expected, thus fault exists in the application.

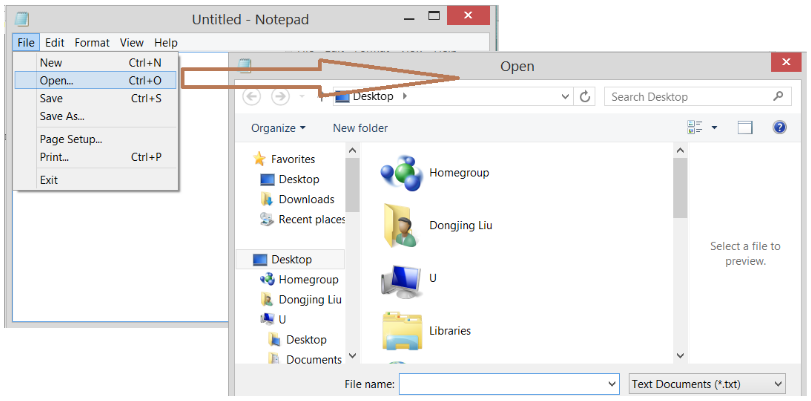


Figure 1.1.1: GUI testing with notepad as example

From the example studied above, GUI testing is critical in guaranteeing the functionalities of GUI applications. As a consequence, it has given rise to academic and industrial interest and concern. Unlike traditional software testing, GUI testing is more complicated and time-consuming to cover all functions provided by the application, which is the reason why current GUI testing relies a lot on manual assistance.

An efficient GUI testing solution can not only improve the performance of application under test but also free human beings from tedious and repeated testing work. Therefore automating GUI testing is important. So far, researchers have proposed some effective methods to realize GUI testing, e.g. Visual GUI testing by Eimil et.all [9], where pattern recognition technology is used in testing process to help find specific GUI elements. While Finite State Machines are implemented in GUI testing by Hu[16] through modeling testing process in mathematic way, similar with Markvove chain adopted in GUI testing process[7] by Yin . Nevertheless, all approaches proposed focus on generating test cases, while some manual assistance is still required in those approaches, e.g. configuring system where GUI tests are located, to certain states before testing, or shutting down system after tests are complete. Thus human beings are still involved in repeated and redundant work. The motivation of the thesis is to provide an automated solution to decrease human intervention as much as possible in the testing process. The solution proposed in this thesis seeks to address the problem with an approach known as fully automated GUI testing.

1.2 Contributions and Outline

The thesis contributes a novel fully automated GUI testing solution. Essentially, the solution takes virtual machines as carrier to execute GUI testing so that all operations involved in testing process are automated. Multiple test cases execution is proposed in this thesis, which optimizes fully automated solution greatly. The evaluation of fully solution is presented in this thesis. Furthermore, a novel test scheduling method with greedy algorithm adopted is illustrated. Moreover, detailed comparison and analysis about two virtual machine applications, VMWare workstation and Hyper-V respectively, are shown so that the better one is selected to implement fully automated GUI testing. The remaining of the thesis is organized as below：

Chapter 2 : studies the basic principle of virtual machine technology. The advantages of locating tests in virtual machines are listed, following with the introduction of VMWare workstation and Hyper-V. And a simple performance comparison is given at the end.

Chapter 3 : describes detailed implementation with regard to automating VMWare workstation and Hyper-V. Besides, comparison of VMWare and Hyper-V in terms of practical use is presented.

Chapter 4 : focuses on implementation of fully GUI testing with a tool provided by Visual studio. Theory concerning GUI testing is studied in chapter 4 as well. The final results of implementation of fully automated solution are analyzed too.

Chapter 5 : proposes a novel test cases scheduling solution for the fully automated approach. Furthermore, priority is introduced to each test case, a combination of test cases execution duration and priority leads to a new method to improve the efficiency with Analytic Hierarchy Process (AHP) used. A simple case is evaluated with the novel solution.

Chapter 6 : draws a conclusion of the entire thesis. Also further work about the solution in industrial evaluation is discussed.

2 Virtual Machine Technology

Since the fully automated GUI testing is implemented in virtual machines, it is essential to study the basic technology of virtual machines. Furthermore, the advantages of performing testing in virtual machine will be listed in this chapter, following with the principle and architecture of the most common virtual machine application, VMWare and Hyper-V respectively. At last a comparison of these two applications are made in a simple manner.

2.1 Virtual Machine Introduction

Virtualization technology is widely used in computing. It refers to creating virtual version (not actual) of a certain item, which can be computer hardware platforms, operating systems, storage devices and so on, In the field of computer science, virtualization, in general sense, is considered as a method of logically dividing the system resources provided by mainframe computers into different applications. For example, in some cases, the memory needed by users may be much larger than the memory size of a physical machine. With virtualization technology, part of the hard disk space can be turned into memory space. Another example of virtualization technology is Virtual Private Network (VPN), a "private" network is virtualized from public network, hence, a secure and stable VPN can be used by users. The virtual machine technology can be traced back to 1960s when IBM divided a powerful machine into several small "pieces" so that system and resources management can be done in a piece level for different purposes.

A virtual machine is, in essence, an application environment or operating system (OS) that is installed on software which emulates dedicated hardware. For users, a virtual machine is identical to a physical machine, for it has identical functions as a physical machine.

Virtual machines are typically created through software, and one or more virtual machines can be generated on the same host machine. Those virtual machines works normally as real physical machines, and users can install application or get access to internet in virtual machines. From the perspective of host machine, they are only processes. However for the application in virtual machines, they are real computers. Before further discussion, the concepts below should be illustrated carefully because they will appear frequently in the thesis.

Virtual machine (VM) is a virtualized machine emulated by software (e.g. VMWare or Hyper-V), logically, it is a physical machine.

Host machine is the existing physical machine with specific hardware details.

Host Operating system (host OS) is the operating system that runs on host machine. For a host machine, there is only one host OS.

Guest Operating system (guest OS) is the operating system that runs on virtual machines. Each virtual machine can be equipped with an unique guest OS.

2.2 Testing in Virtual Machines

Considering the situation that there are several virtual machines existing in a single host machine, these virtual machines work independently without interference to each other. In this case, a physical machine can be deemed as being able to run multiple operating systems simultaneously, which brings out the inspiration of fully automated GUI testing in this thesis.

Locating tests in virtual machine is not a novel concept, With regard to the benefits of testing GUI application in virtual machines, the following items illustrate them well

1. Typically, for software testing, developers tend to test application with different configurations in various environments so that the robustness can be enhanced. However, in this thesis, for a physical machine without an embedded system adopted, the best way to test application in different environments is to install multiple operating systems. Generally, there are two ways to construct multiple OSes in a physical machine. Firstly, installing multiple hard disks,with each storing an operating system. Obviously the approach is not secure enough, because Main Bootable Record (MBR) is vulnerable to get attacked, and in worst case, all operating systems

can be damaged. The second approach is to adopt the virtual machine software. As long as the host machine is powerful enough, multiple virtual machines can work properly. Besides, using virtual machines is also an economic and secure solution.

2. One of the most attractive points of using virtual machines for testing is that the processes in host machine will not be affected by any operations on application under test. Vice versa, the system or applications running on host machine will not influence the testing in virtual machine. Isolation is built between host machine and virtual machines.

3. Under some circumstances in which operating system is attacked via a host machine, damage may require a complete system reinstallation. This is very expensive when considering the files or application inside the host machine. Nevertheless, for a virtual machine, it is cost-free to reinstall , and virtual machine programs generally provide function for users to revert to certain state of guest OS, which makes reinstalling system unnecessary.

4. With snapshot, some repeated and tedious steps involved in testing process (e.g. some configuration to systems) can be avoided through taking snapshot and reverting to the state, which improves the efficiency and avoids manual assistance involved.

The advantages presented above make virtual machine a good approach to execute testing. Virtual machines can provide testing of a diverse environments in an inexpensive and secure way. The isolation from host machine system also make an independent testing environment possible. Last but not least, some practical functions provided by virtual machine software bring convenience for guest OS management, which facilitates testing to some extent.

To install virtual machines in a host computer, a virtual machine software is a prerequisite. However, there exist many different virtual machine applications on the market, e.g. VMWare by VMWare company, Hyper-V provided by Microsoft, or Xen designed by Oracle. They all have their respective pros and cons. Finding a proper one for GUI testing is one of the targets in this thesis. Considering that VMWare and Hyper-V are widely used for Windows operating systems, and Windows OSes are chosen as the platforms for GUI testing in this thesis. Consequently, a detailed study and analysis about VMWare and Hyper-V will be presented in the following section.

2.3 Principle of VMWare workstation

VMWare is a company established in 1998. It provides the world-renowned virtualization infrastructure solutions and cloud infrastructure solutions provided are world famous. Among all the solutions, virtual machine applications it designed are popular both in industrial filed and academic areas. The release of VMWare workstation, VMWare VSphere and Fusion make applying virtual machines more easier. Concerning the features of each product, Fusion is designed mainly for Mac OS X, while VSphere is set for enterprise use, and as a result many complicated function included, e.g. VCenter, database and active directory domain and so on , these functions are unnecessary for this thesis. VMWare workstation therefore is an appropriate application.

VMware Workstation Pro takes advantage of the latest hardware to replicate server, desktop and tablet environments in a virtual machine. Thus Running applications on a breadth of operating systems including Linux, Windows and more at the same time on the same PC without rebooting is possible. VMware Workstation Pro makes it really easy to evaluate new operating systems like Windows 10, and to test software applications and patches. Reference architectures are in an isolated and safe environment. The core technology of virtual machine is hypervisor, also known as virtual machine monitor (VMM). Logically, hypervisor is a platform between host machine hardware and software, it enables one or more operating systems to run on a host machine in parallel. From the definition, it is apparent that hypervisor can get access to all the hardware resources, including hard disk, memory etc. it schedules the resources allocation to each virtual machine, while preventing virtual machines from interfering with each other. There exists two types of hypervisor.Though each of them has its' advantages and disadvantages, it is worthwhile to study and analyze the similarities and differences between them.

Hosted hypervisor (Type 2 hypervisor)

Hosted hypervisor, also called type 2 hypervisor, is a virtual machine monitor that is installed as an application on the host operating system. It is what VMWare workstation is built with.

As figure 2.3.1 shows, hosted hypervisor relies on host operating system for its operations. Hosted hypervisor locates on top of host OS together with other processes in host machine, and virtual machines are built on top of hypervisor. In the background, hosted hypervisor schedules virtual machines by coordinating calls for memory, CPU, hard disk, network and other resources through the host operating system. The working pattern of hosted hypervisor make it easy for an end use to implement virtual machines on a personal computing device. The typical application of type 2 hypervisor is VMWare workstation.

One of the greatest benefits of a type 2 hypervisor is that it can take advantage of any hardware the host OS has driven for. Also monitoring or backing up from the host OS is much easier with a hosted hypervisor due to its architecture. Last but not least, the penalty of this type hypervisor is considerably low, which makes it suitable for development. VMWare workstation adopts hosted hypervisor,

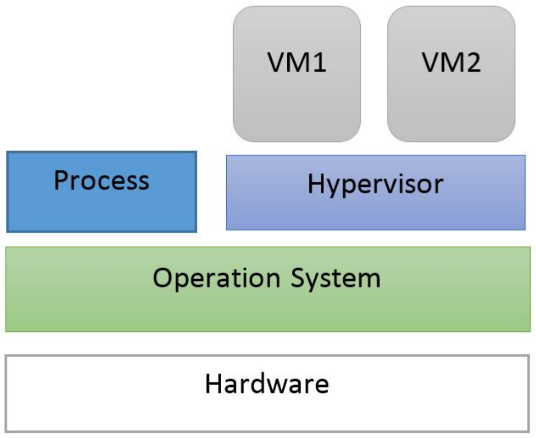


Figure 2.3.1: Architecture of hosted hypervisor.

the advantages mentioned above make it appropriate for implementing GUI testing solution.

2.4 Principle of Hyper-V

Hyper-V is a product designed by Microsoft, which was released firstly in 2008 as a part of Windows Server 2008. It adopts the architecture of bare-meta hypervisor. There are two ways to create virtual machines with Hyper-V, para-virtualization and full-virtualization respectively. The former can be used when guest operation system is the same with host OS, and generally, they are same version of windows OS. Best performance can be gained for virtual machines in para-virtualization. While full-virtualization requires that the CPU supports virtualization (inter-VT or AMD-V) so that different guest operation systems from host one can be created. In order to use hyper-V, some demands towards hardware must be satisfied, i.e CPU architecture must be x86 compatible, and that only ones that meets the requirements are from intel or AMD64. the CPU must have Data Execution Protection (DEP). Minimum 2GB memory is necessary, Windows Server 2008 or later and Windows 8 or later operation system are required too. The hypervisor of Hyper-V is bare-metal hypervisor, also known as type 1 hypervisor.

Bare-metal hypervisor (Type 1 hypervisor)

Bare-metal hypervisor, which has gained more popularity in recent years, runs directly on host machine hardware, not relying on any operation system. It realizes resource allocation, hardware control and virtual machine monitor by taking resources in host machine directly instead of through host OS. Obviously, high-performing can be achieved with this working pattern. Typical products are hyper-V and VMWare vSphere. Compared with a hosted hypervisor, it takes little RAM and is relatively faster to reinstall if needed.

Windows Server 2008 must be run in ”Parent” partitions as shown in figure 2.4.1 . A virtualization stack is included in ”Parent” partition, where management tools and some automation tools are located. Also in each child virtual machine, all operation systems are run in partitions. In addition, VMBus ,which is a high performance architecture included in Hyper-v , is designed to realize communication between parent partition and child partition, which means Server 2008, Windows Vista, Winders Server 2003, and Xen-enabled Linux are able to pass hardware requests along a new memory pipeline.

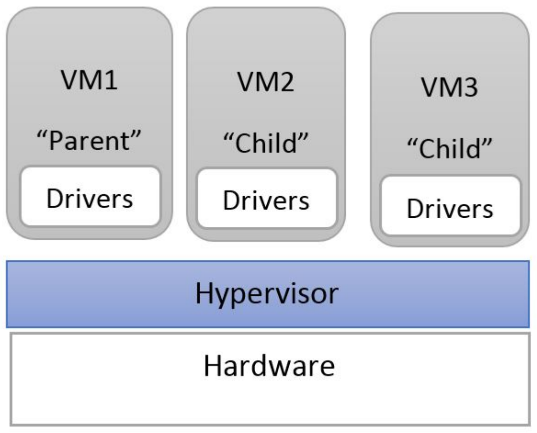


Figure 2.4.1: Architecture of hosted hypervisor.

2.5 Simple Comparison of VMWare and Hyper-V

Basic information concerning various virtual machine programs is presented in the previous section. However, which of these virtual machine applications, is better or more proper for GUI testing is what this thesis tries to find out. Considering that GUI testing in this thesis is set to run on Windows OS. Two excellent virtual machine applications both with superior performance on Windows platform namely VMWare Workstation and Hyper-V , are chosen as candidates for GUI testing in this thesis. The following part aims to compare these two applications in a simple manner.

2.5.1 VMWare performance VS Hyper-V performance

Apparently, the most straightforward criteria of comparing VMWare and Hyper-V is to evaluate the host machine performance when a virtual machine is starting, because in the starting process, both hypervisors will work to schedule the hardware resources to guarantee virtual machines booting successfully. Therefore the host machine performance in the virtual machine starting process can reflect the quality of the software effectively. The performance of a host machine for VMWare workstation and Hyper-v is shown as figure 2.5.1 (a) and (b) separately. The configurations of the host machine and the virtual machines for VMWare workstation and Hyper-v are listed in table 2.5.1 and 2.5.2 separately:

Table 2.5.1: The configuration of host and virtual machine in VMWare workstation

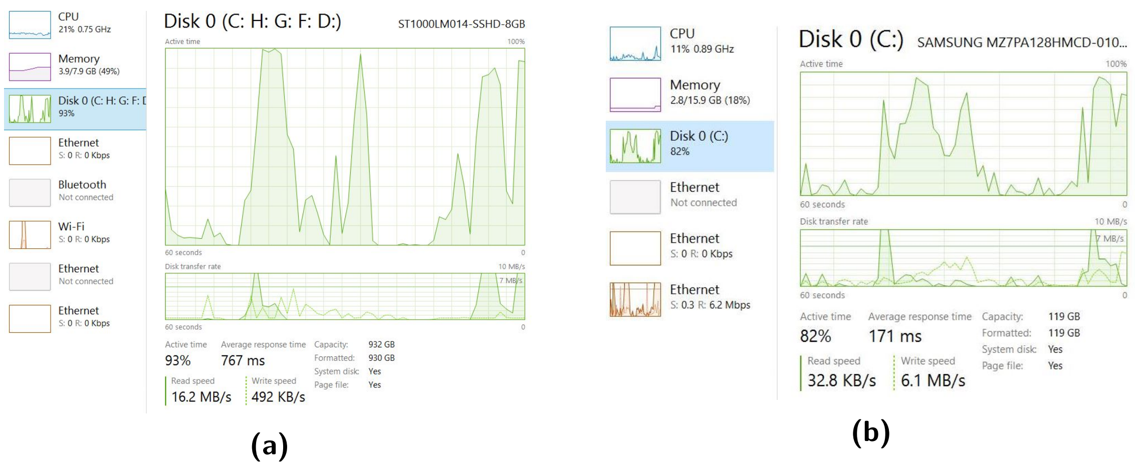
Parameters	Host machine	Virtual machine
Processor	Intel Core i7, CPU @2.00 GHz	1
Memory	8.00 GB Installed	1GB
Hard Disk	932 GB memory	40 GB
Operating System	Windows 8 Pro	Windows 7

Table 2.5.2: The configuration of host and virtual machine in Hyper-V

Parameters	Host machine	Virtual machine
Processor	Intel Core i7, CPU @2.40 GHz	1
Memory	16.00 GB Installed	1GB
Hard Disk	119 GB memory	40 GB
Operating System	Windows 10	Windows 7

The reason why VMWare workstation and Hyper-v are not installed in the same host machine is that they are not compatible to each other. The two different host machines have different hardware settings, which make quantitative analysis impossible, However the performance for two softwares can still be qualitatively reflected. Thus a rough comparison can be made.

Figure 2.5.1 (a) shows the performance of host machine when a virtual machine with guest OS Windows 7 installed in VMWare workstation is powering on. Figure 2.5.1 (b) illustrates the performance of host machine when a virtual machine installed in Hyper-V is starting. Due to the different


```
CPU                     Disk 0 (C: H: G: F: D:)          ST1000LM014-SSHD-8GB
21% 0.75 GHz            Active time                                      100%
Memory
3.9/7.9 GB (49%)
Disk 0 (C: H: G: F: D
93%
Ethernet
S: 0 R: 0 Kbps
Bluetooth
Not connected
Wi-Fi                   60 seconds                                          0
S: 0 R: 0 Kbps          Disk transfer rate                            10 MB/s
Ethernet                                                               7 MB/s
Not connected
Ethernet                60 seconds                                          0
S: 0 R: 0 Kbps          Active time   Average response time   Capacity:     932 GB
                        93%           767 ms                  Formatted:    930 GB
                                                              System disk:  Yes
                        Read speed    Write speed             Page file:    Yes
                        16.2 MB/s     492 KB/s
```
(a)

```
CPU                     Disk 0 (C:)  SAMSUNG MZ7PA128HMCD-010...
11% 0.89 GHz            Active time                                      100%
Memory
2.8/15.9 GB (18%)
Disk 0 (C:)
82%
Ethernet                60 seconds                                          0
Not connected           Disk transfer rate                            10 MB/s
Ethernet                                                               7 MB/s
S: 0 R: 0 Kbps
Ethernet                60 seconds                                          0
S: 0.3 R: 6.2 Mbps      Active time   Average response time   Capacity:     119 GB
                        82%           171 ms                  Formatted:    119 GB
                                                              System disk:  Yes
                        Read speed    Write speed             Page file:    Yes
                        32.8 KB/s     6.1 MB/s
```
(b)

Figure 2.5.1: Performance of host machine for VMWare and Hyper-V

settings with host machines, precise discrepancy is impossible to obtain. However, from figure 2.5.1, it is still easy to tell that Hyper-v consumes less host machine resources, especially in terms of memory and disk usage. It also proves that VMWare workstation occupies more host OS resources due to the principle of a hosted hypervisor, according to which VMWare is considered as a process in the host machine, resources will be scheduled through host OS. While for Hyper-V, it is independent from the host OS, and resources are coordinated directly from host machine hardware.

2.5.2 Hosted Hypervisor VS Bare-metal Hypervisor

Hypervisor, which in essential is a set of code, is the core component of virtual machines. It enables various guest operation systems to share a single host machines' hardware resources, like processor, memory and so on. In other words, Hypervisor controls the host machine resources by allocating processor or memory to virtual machines upon requests, and ensures that different virtual machines do not disturb each other, which is the reason why hypervisor is called Virtual Machine Monitor(VMM). According to the architecture, hypervisors are classified into two types, bare-metal or native hypervisors , also known as type-1 hypervisors, and hosted hypervisors, which is called type-2 hypervisors too.

1. **Native or bare-metal hypervisor**, also called type 1 hypervisor. This hypervisor runs directly on host machine hardware. It realizes resources allocation, hardware control and virtual machine monitor by emulating directly from host machine hardware. A strict requirement on hardware is unavoidable for the usage of type 1 hypervisor. However, an excellent virtual machine performance is possible with type 1 hypervisor.

2. **Hosted hypervisor**, requires a host operation system. This hypervisor emulates hardware resources in the host operation system instead of directly from host machine hardware, In this way, type-2 hypervisor is an application in host OS . Although it can monitor and allocate resources, it is limited by the resources in host OS. Nevertheless, easy use and management is the reason why it is still well received.

From the analysis mentioned above, we know that both hosted and bare-metal hypervisors have their own advantages and disadvantages, e.g. It is easy for hosted hypervisor to manage virtual machines in host OS with a low cost, while bare-metal hypervisor takes little hardware resources from host machine, and therefore a virtual machine runs fast with it. But it is picky concerning hardwares, which means not all machine can implement bare-metal hypervisors. Similarities and differences lies with many other aspects too, the table below shows detailed comparisons in a theoretical way.

Table 2.5.3: Comparison of two type hypervisors.

Parameters	**Bare-metal hypervisor**	**Hosted hypervisor**
Performance	High-performance,low cost	Low-performance, high cost
Compatibility	Specific hardware of OS required	No specific hardware requirement
Easy to use	Easy to install, complicated configuration	Easy to install, configure and use
Availability	Available if host machine is out of order	Not available
Reliability	High reliability due to quality-assurance	No QA
Management	Batch VMs management available	Single VM management required
Cost	Expensive to extend advanced features	Free, or low-cost
Extensibility	High, hundreds of VMs supported	Limited
Products	Microsoft Hyper-V, Oral VM, Linux KVM etc.	VMWare workstation, Microsoft Virtual PC

The differences mentioned in the table are actually the differences lying behind VMWare and Hyper-V. From the table 2.5.1 we can tell that, the bare-metal hypervisor, which Hyper-v relies on, has a better performance and lower cost with regard to occupying host OS resources. However, it a has strict requirement on host machine's hardware, while VMWare workstation is very easy to use and manage. In summary, Hyper-v is suitable for large-scale implementation of virtual machines with a low hardware cost and management achieved. If there are only a small number of virtual machines to be used, VMWare is a better choice due to its convenience to use.

3 Automating Virtual Machines

Basic knowledge about virtual machines is studied in previous chapter, and a simple comparison about VMWare workstation and Hyper-V is presented. In this chapter, the principle and work flow of fully-automated GUI testing with virtual machines will be given. Besides, the corresponding automation implementation involved with virtual machines is demonstrated for both VMWare workstation and Hyper-V. A detailed comparison in terms of practical use is shown finally, and virtual machine software for fully automated GUI testing is chosen according to the comparison.

3.1 Principle of Fully Automated GUI Testing

Currently, for GUI testing, most of research work has put the weight on proposing new approaches to generate GUI test cases, e.g. Marlon et al, proposed a test case generation based on Unified Modeling Language (UML)[37], while Emil et al[9] introduced a new technology using image recognition to identify the GUI objects. While a Finite State Machine is implemented in GUI testing by Hu[16] through modeling testing process in mathematic way. However, all of these work focus on the approaches to generate GUI test cases, considering the entire testing process, the system where GUI testing is executed on has to be turned on manually before performing testing. Therefore, additional manual assistance, which can be turning on systems, installing pre-requested applications or switching off system after all tests are finished, has to be available so that theses test cases can to executed successfully. To some extent, the approaches in currently existing research work are only half-automatic GUI testing in terms of the entire testing process. In this thesis, a fully automated solution is proposed to avoid manual assistance as much as possible.

For the sake of improving the robustness of testing, different testing environments are needed, virtual machines with different language settings will be used in the thesis to achieve the goal. Figure 3.1.1 demonstrates the basic diagram of the project. A pool of guest operation systems with different language options are installed, which are probably located in a remote host or cloud machine considering the remarkable compatibility and outstanding flexibility, or it can be located in a local machine like in this thesis. While in local machine a script is designed to control the virtual machines automatically with Application Programming Interface (API) provided by virtual machine software. Taking the efficiency into account, multiple virtual machines are controlled in parallel.

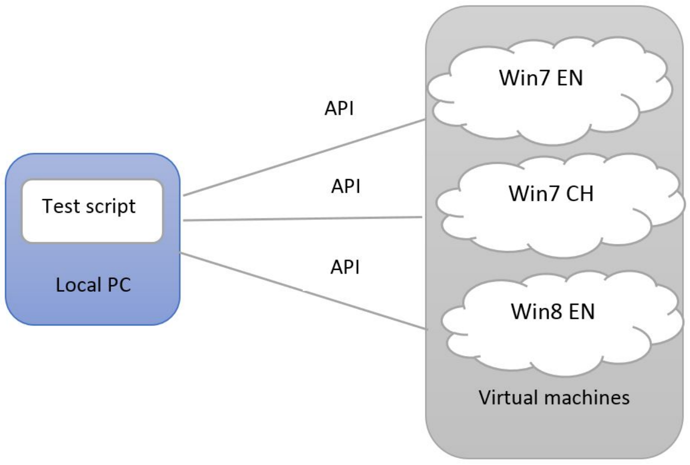


Figure 3.1.1: Diagram of fully automated solution

How a virtual machine can be programmatically controlled by a script to execute a GUI test is critical in this thesis. The typical scenario is shown as figure 3.1.2. Firstly, a guest OS should be powered on automatically by a script, noticeably, guest OS with different language settings are designed to start concurrently, which means that all operations in the workflow are executed simultaneously in multiple guest OSes. Snapshots, which enables the client to get back to any certain state of guest OS, should be created before any operations in guest OS. Thus a clean state of guest OS is available. The program under test is installed automatically via the control of script, following with running the GUI testing, which is the most important part in the project. The outcome of test will be collected by a script on host machine to verify the functionality of program. Last but not least, the program installed should be removed from the guest OS, and the guest OS should be restored to the root snapshot, which means going back to the initial state. The workflow shows basically each operation involved in fully automated solution. It is worthwhile to mention that all the steps involved should be executed automatically and in parallel.

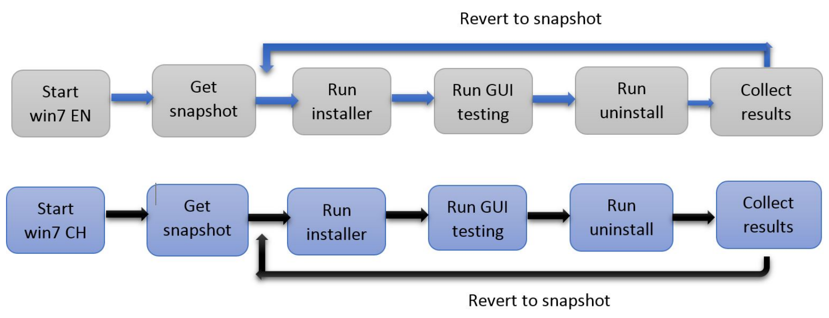


Figure 3.1.2: Workflow of fully automated solution

From the description of the project above, one of the most significant parts of the thesis is to automate virtual machines programmatically. Specifically, the automation includes powering on operating systems, installing programs, running tests, managing snapshots and copying files between host and virtual machines. One possible approach to realize automatic management is to use API provided by virtual machine software, and the other way is to try to control the hardware (e.g. mouse click, keyboard input) in virtual machines. The feasibility and efficiency of these two approaches will be discussed in the following chapter. Furthermore, the implementation of automatic control on both VMWare workstation and Hyper-V will be presented too. A comparison of these two products will be given so that the one works better for fully automated solution will be chosen.

3.2 Automating VMWare Workstation

For automating operating in guest OS, controlling the low-level hardware in virtual machines is always one possible solution. In this way, the specific position of elements under control (e.g. the directory of an executable file) is required. Hardcoding the position of executable file in scripts tends to cause failure if the element moves even slightly. Another possible method is hardcoding the name of the element, however, it is still easy to result in failure if the name of the executable file changes. Obviously the hardware control is possible, but it is neither efficient, nor reliable.

The VMWare company has released virtual infrastructure extension (VIX) API, which enables developers to automate virtual machines programmatically with an asynchronous, job-based programming model.

There are two types of VMWare APIs provided.

- VMWare Virtual Infrastructure SDK: a set of tools and APIs to manage the VMWare Infrastructure environment. A toolkit that contains managed wrappers on top of the SOAP interface provided by a VMWare deployed. The toolkit is mainly applied on VMWare ESX or Virtual Center management , which will not be discussed in this thesis.

- VMWare VIX API. The VIX API allows developers to write programs and scripts aiming to automate virtual machine operations, as well as manipulating guests within virtual machines. It runs on both Windows and Linux and supports management of VMware Server, Workstation, player, fusion and Virtual Infrastructure. Bindings are provided for C, Perl, and COM (Visual Basic, VBscript, C#). Considering the compatibility of windows systems, .Net technology language (C#) is adopted to implement automation in this thesis.

Before automation with Vix API, some concepts concerning implementation should be illustrated, as is shown as follows:

Objects: the Vix API is object oriented. It either creates objects or modifies the properties of existing objects for the majority of functions provided by Vix API.

Handles: Handles are opaque identifiers (actually integers) that can be passed as parameters to functions. Most functions provided by Vix API take a handle as an input parameter.

There are several handle types,and the ones used in this thesis are:

- Virtual Machine handles, it represents a single virtual machine, the virtual machine might or might not be powered on.

- Host handles represent a single host computer, either the local host or a remote host.

- Job handles, through which asynchronous operations are implemented.

- Snapshot handles, which indicate a snapshot of a virtual machine, which can be reverted or deleted.

Considering the compatibility with Windows systems, .Net technology programming language (C#) is adopted to realize remote control through test script. The brief code map is shown in figure 3.2.1, it is noted that only part of the methods in the automation class are shown in the map, the relation of methods and handles are illustrated with arrows, the detailed method description and implementation will be presented in the following part.

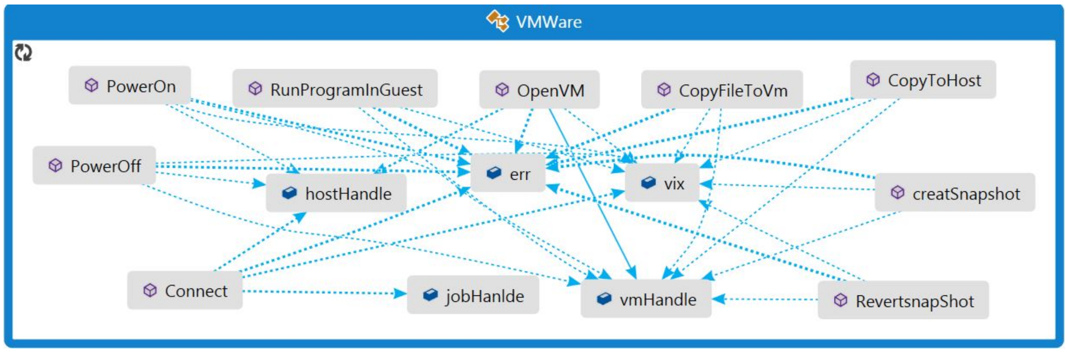


Figure 3.2.1: Code map for VMWare automation class

3.2.1 Connecting to Virtual Machine in VMWare Workstation

To work with virtual machine, a connection between host machine and virtual machine should be established. If the virtual machine is stored on a host running workstation, a local connection is required. And if virtual machine is installed on a remote ESX, a credential should be supplied. With connection between two machines, automation can be realized through various handles. The snippet below shows how a local connection is achieved. Specifically, a **Connect** function provided by Vix API is utilized. The host type ,which describes the VMware software running on the host , is specified by second parameter, no matter where the client is running. It is *VIX_SERVICEPROVIDER_VMWARE_WORKSTATION* in this thesis, which indicates that the software is VMWare workstation.

Listing 3.1: Connection function with Vix API

```
public void Connect()
{
  jobHanlde = vix.Connect(Constants.VIX_API_VERSION,
      Constants.VIX_SERVICEPROVIDER_VMWARE_WORKSTATION, null, 0, null,
      null, 0, null, null);
  int[] propertyIds = new int[1] {
      VixCOM.Constants.VIX_PROPERTY_JOB_RESULT_HANDLE };
  err = jobHanlde.Wait(propertyIds, ref results);
  object[] hostArray = (object[])results;
  hostHandle = (IHost)hostArray[0];
}
```

It is noticeable that the **Connect()** function is an asynchronous function, which means that the function either implements time-consuming operations or interacts with persistent virtual machine state. In this case, the asynchronous function allocates and returns a job handle.The job handle is a Vix object that represents the execution of the asynchronous operation. It can be used to indicate when the asynchronous has finished, and also, it can be utilized to retrieve the results of the completed asynchronous function. Furthermore, the returned job handle may have several result properties that are set when the job has completed. Information returned by a finished job is included in the result properties.

A new job object will be always created by an asynchronous function, and the created job object tracks the status of corresponding asynchronous function, so results of running asynchronous function can be retrieved after the call completes. Hence it is reasonable to wait until the call finishes so as to get results. Currently, there are three approaches to detect when an asynchronous call has finished, namely, pooling the job object for completion, using job object to block calls and using a call back function. Among all the approaches, the second one, which is using job object to block calls, is the most commonly used one. A function called **Wait**() provided by Vix API helps to realize blocking. Moreover, results can be retrieved directly from **Wait**() function, otherwise, an additional **GetProperties**() should be invoked on the job object to get the results back.

In the **Connect**() function above, a result property is retrieved when the **Connect**() has completed namely *Constants.VIX_PROPERTY_JOB_RESULT_HANDLE* , and the **Wait**() function is used to both signal the completion of the asynchronous call and retrieve the final results. It is worth to mention here that the returned result of a **Connect**() function is the host handle, which represents a host machine. Next step after obtaining the host handle in the automation process is to power on certain virtual machines , which will be described in the following section.

3.2.2 Identifying a Virtual Machine

According the principle of Vix API, a handle is used to identify different Vix objects, and there are different types of handles. In the previous work, a host handle is obtained by connecting host machine with virtual machine software. Considering the operation in virtual machines, a similar handle namely virtual machine handle is needed to realize guest operations, which can be exploited to identify or represent a virtual machine.

Mechanism provided by Vix API to obtain a virtual machine handle is converting a virtual machine path to a handle by a **OpenVM** () function, same with **Connect**() function, **OpenVM** () is an asynchronous function called on the host handle. The result of such asynchronous call is therefore a virtual machine handle. **Wait**() is adopted here to obtain the result as in **Connect** function.

3.2.3 Changing State of Virtual Machines

With virtual machine handle obtained, a virtual machine is represented by the handle. In general, automatic management of virtual machines includes booting or shutting down guest OS, In order to start up or shut down virtual machines through script using Vix API, the following steps must be obeyed in the script:

1. Connecting to the host machine on which virtual machine is installed.
2. Getting the handle of host machine.
3. Using host machine handle to convert virtual machine file path to a virtual handle.
4. Utilizing virtual machine handle to call a function **PowerOn**() to start up the virtual machine.

Noticeably, **PowerOn**() function can be used in two ways

- To start up a virtual machine in a previously power-off state.
- To resume execution of a guest operating system in a suspended virtual machine.

For the sake of powering off or suspending a virtual machine, similar with powering on, the first three steps in the above numbered list should be followed, and only difference lies with the last step, using **PowerOff**() or **Suspend**() depends on purpose , instead of using **PowerOn**().

3.2.4 Snapshot Management

Snapshot is one of the most important functions provided by virtual machine software,and plays a critical role in fully automated GUI testing with virtual machines, for snapshot enables guest OS to restore to any certain state. As a consequence, some repeated and tedious testing work can be avoided by reverting to snapshot with system configured directly. Also backing up of testing environment can be obtained easily with snapshot.

With regard to snapshot management, creating snapshot and reverting to certain snapshot is taken into account in this paper. In terms of taking snapshot, a **CreateSnapshot**() function is available to save a copy of virtual machine state as a snapshot object. It is noted that a snapshot with different configurations can be taken with different inputs to the function. Concretely speaking, 0 and *VIX_SNAPSHOT_INCLUDE_MEMORY* can be passed to the function, indicating taking snapshot without memory and with memory respectively.

When considering reverting to a specific snapshot, **RevertToSnapshot**() helps reach the target. It is utilized to restore virtual machine to the state when certain snapshot is created. A snapshot handle is used here to represent a certain snapshot, which can be acquired by **GetRootSnapshot()** function. The parameter in **RevertToSnapshot()** decides how a snapshot will be reverted specifically. If a virtual machine is powered on when the snapshot was created, the parameter will determine how the virtual machine is powered back on.

3.2.5 Automating Guest Operations

The fully automated solution relies a lot on whether the guest operations can be automated. The operations which are necessary for fully automated solution include copying files between host machine and virtual machine, running programs in the virtual machines automatically, creating folder and so on. With the functions provided by Vix API, it is possible to realize all these operations by **CopyFileFromGuestToHost()**,**CopyFileFromHostToGuest()**,**CreateDirectoryInGuest()** and so on. Among all these operations, running programs in guest OS is the most relevant to the GUI testing. Therefore, it is worthwhile to take a further look into the implementation of the operation.

It is noticeable that there are two prerequisite functions before any operations in guest OS. Specifically, users must log in as a VMWare user, which means that user will be granted the permission of guest operations after logging in as a VMWare user. **LoginInGuest()** is the method provided by Vix API to realize log in function, This function establishes a guest operating system authentication context that can be used with guest functions for the given virtual machine handle. In addition to logging in, a collection of Vix services must be ready before all most all guest operations. The waiting time can be specified by **WaitForToolsInGuest()** function.

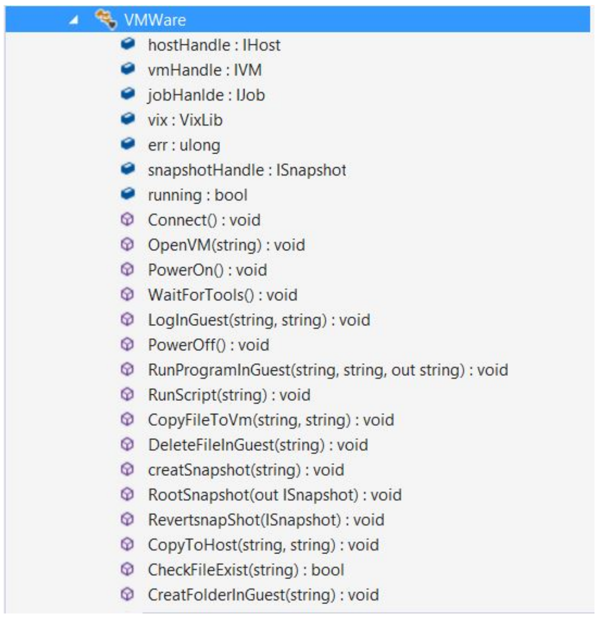
Running programs in guest OS can be achieved through the function of **RunProgramInGuest()**. As shown in the codes below, the first parameter for the method is the absolute path where the program stored in the file system in the absolute directory for the program . The second parameter is command line argument, which provides users to run the program in their desirable way (e.g. passing "VERYSILENT" as the command line arguments can automate installation of an application without manually clicking next or accept terms). For Windows guest operating systems, user must pass *VIX_RUNPROGRAM_ACTIVATE_WINDOW* as the value for the third parameter when running a program with a graphical user interface. This value will ensure that the application's window is visible and not minimized on the guest's screen. The value is very important to GUI testing, because an active desktop is required for GUI testing, if the application under test runs in the background, which means it is invisible to users, and therefore performing GUI testing is impossible.

Listing 3.2: Running program in guest OS with Vix API

```
int[] propertyId = new int[] {
    Constants.VIX_PROPERTY_JOB_RESULT_GUEST_PROGRAM_EXIT_CODE };
IJob jobHandle = vmHandle.RunProgramInGuest(programPath, command,
    Constants.VIX_RUNPROGRAM_ACTIVATE_WINDOW, null, null);
```

Being similar with other asynchronous functions, **RunProgramInGuest**() requires verifying the completion of the call, and the result of calling such function can be obtained by **Wait**() function. The returned results, which are the properties of the created job handle, can be set as following:

- *VIX_PROPERTY_JOB_RESULT_PROCESS_ID*: the process ID of the application which has finish execution.
- *VIX_PROPERTY_JOB_RESULT_GUEST_PROGRAM_ELAPSED_TIME*: the elapsed time of the process in seconds.
- *VIX_PROPERTY_JOB_RESULT_GUEST_PROGRAM_EXIT_CODE*: exit code of the process. it is necessary to obtain value that the application returned.



```
VMWare
  hostHandle : IHost
  vmHandle : IVM
  jobHanlde : IJob
  vix : VixLib
  err : ulong
  snapshotHandle : ISnapshot
  running : bool
  Connect() : void
  OpenVM(string) : void
  PowerOn() : void
  WaitForTools() : void
  LogInGuest(string, string) : void
  PowerOff() : void
  RunProgramInGuest(string, string, out string) : void
  RunScript(string) : void
  CopyFileToVm(string, string) : void
  DeleteFileInGuest(string) : void
  creatSnapshot(string) : void
  RootSnapshot(out ISnapshot) : void
  RevertsnapShot(ISnapshot) : void
  CopyToHost(string, string) : void
  CheckFileExist(string) : bool
  CreatFolderInGuest(string) : void
```

Figure 3.2.2: Variables and methods of VMWare automation class

The **RunProgramInGuest()** function together with other methods shown in the figure 3.2.2 ensure automating virtual machines in VMWare workstation possible, thus the fully automated GUI testing with virtual machines is guaranteed to be applicable .

3.2.6 Manipulating Multiple Virtual Machine Simultaneously

Considering the principle of fully automated GUI testing with virtual machines described in previous chapter, a pool of guest OSes with different settings are installed for different virtual machines. If there exists an approach to perform test cases simultaneously, the performance will be enhanced to a very great extent. Under this circumstance, performing multiple test cases can be equivalent to automating several virtual machines at the same time. However, Vix API does not provide any function to achieve concurrent automation. Fortunately, .net technology does supply method to perform multitasks in parallel, namely multithreading.

A thread is defined as an execution path of a program, also every single thread can be deemed as a unique flow of control. Multithreading indicates that user can have multiple threads of execution inside a single program. When multiple threads are executed, it is likely to have multiple CPUs execution within the same program. As illustrated in figure 3.2.3, the first executed thread is known as main thread, and another three threads can be started simultaneously. Between any pair of threads, thread may switch and exchange data/ results.

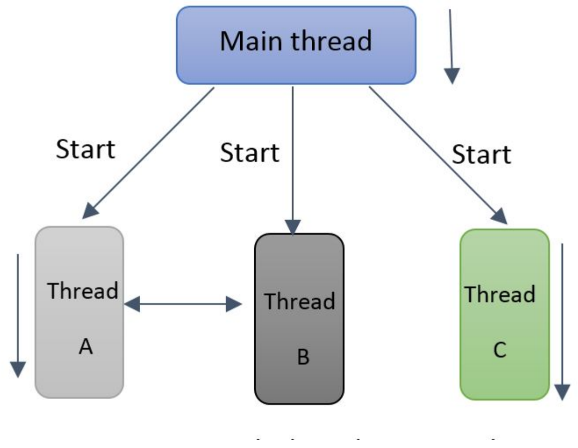


Figure 3.2.3: A brief overview of multithreading

The benefits of using multithreading can be listed as follows:

- Multithreading can be utilized to set isolation between different codes, therefore, the reliability of applications can be improved.

- In most cases, codes can be simplified with multithreading.
- Concurrent execution can be achieved easily. Hence the efficiency of program is increased greatly.

In C#, the **System.Threading.Thread** class is designed for working with threads. Creating and accessing individual threads in a multithreaded application is allowed. When C# program starts execution, the main thread is created automatically, a new thread can be created like normal variable. The following codes realize multithreading in C#.

Listing 3.3: Multithreading implementation

```
for (int j = 0; j < groupNo; j++)
{
  Thread[] testThread = new Thread[groupLength];
  for (int i = 0; i < groupLength; i++)
  {
    testThread[i] = new Thread(tws.TEST);
    testThread[i].Start();
  foreach (Thread thread in testThread)
  {
    thread.Join();
  }
}
```

The snippet above divides all virtual machines into several groups named *groupNo*, and in each group there are certain number of virtual machines namely *groupLength*. By default , virtual machines in a same group will be automated simultaneously, and the next group will start when all virtual machines in current group finish all work. The reason for classifying virtual machine into different groups is that both host and guest OS may be extremely slow-responding if all virtual machines (6 in this thesis) are set to run at the same time. Therefore it will be expensive for the host machine hardware to perform simultaneous automation for all virtual machines. It is reasonable to make compromise between cost and efficiency. Setting different groups for virtual machines is a considerable way to increase the efficiency and decrease the cost at the same time.

The description of various functions in the above section can briefly illustrate how virtual machines installed by VMWare workstation can be automated, and why Vix API is critical to achieve the goal. Moreover, user experience towards Vix API can be generated in the course of automation. Roughly speaking , Vix API is easy to use and the efficiency of various functions provided is quite high. As a consequence, simultaneous automation of virtual machines installed through VMWare workstation is uncomplicated to achieve. the following section will discuss the automation for virtual machines stored through Hyper-V, and a comparison toward automation for VMWare and Hyper-V will be presented at the end.

3.3 Automating Hyper-V

There are plenty of tools users can utilize to manage Hyper-V, e.g. Windows PowerShell acts as a command management tool to automate operation in Hyper-V. However Windows Management Instrumentation (WMI) API provides more classes to manage both hardware and software of host and virtual machine in a programming way, which is the reason why it is adopted in the thesis.

3.3.1 Windows Management Instrumentation (WMI) Technology

Windows Management Instrumentation is a core management technology for Windows. It is based on Common Information Model Object Manager (CIMOM). Visiting, managing and monitoring Windows resources is easy with WMI. WMI allows managing in both local and remote computers. For example, users can start a process in a remote computer and acquire any system information through WMI. Besides, WMI provides a common interface for MMC and scripts to manage different OS component without using different API. Different components of OS are represented by a collection of objects with unique method and properties in WMI. All these objects are stores in a database called CIM repository, user can adopt WMI query language (WQL) to query specific objects and create different classes to represent network switchers, applications and so on. Developer can manage different component of OS through making change to CIM classed by methods and properties provided . The possibility of use WMI in different language is taken into consideration when WMI is designed, programming language like C/C++, Visual Basic, scripting languages (such as VBScript or JScript) .NET family (C# for example) are supported. Therefore, users can choose their preferred language to programmatically manage Hyper-V.

3.3.2 WMI Objects Description for Hyper-V Management by Operations

Similar to operations required for GUI testing in VMWare workstation, guest operations in Hyper-V also consists of some basic operations, which can be shown figure 3.3.1. The code map of Hyper-V automation class illustrates the relation between necessary methods and variables (part of them). It is apparent that the method of **RunProgramInGuest()** has no association with other functions or variable. The reason will be presented in the following section together with the detailed implementation of various methods.

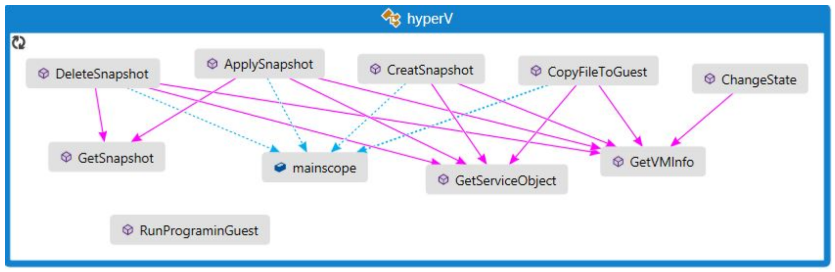


Figure 3.3.1: Code map of Hyper-V automation class

3.3.3 Connecting to Virtual Machine

Unlike VMWare workstation, where connecting virtual machine to host requires obtain the handles of both host machine and virtual machine due to its hosted hypervisor architecture. Connecting to virtual machine in Hyper-V simply means getting the object of certain Virtual Machine. The **Msvm_ComputerSystem** class is designed for developers to get the information of both host machine and virtual machines. The class has some properties and method is used in the thesis, as shown below:

- *Caption* is the properties describing the object, it is set to "Virtual Machine" if the instance represents a VM, if the instances is host machine, it will equal to "Host Computer System"
- *ElementName* is the name user set to virtual machine, for example "win7" in the project. Or it is name for host machine if host machine is represented.
- *EnabledState* is the states of virtual machine representing by an integer, indicates VM is running or turned off and so on, it can be changed by the method of **RequestChangeState**, which is used to power on or power off a virtual machine.
- *Name* is a unique symbol of certain virtual machine. It is recognized by system and is useful when creating instance of system data.

By obtaining the **Msvm_ComputerSystem** class, basic information about both host and virtual machine can be acquired. Noticeably, no matter Powershell or script, administrator right is needed to list all the virtual machine information, otherwise, only host information is obtained. The code below shows how WMI class is used for listing virtual machine information. Before using any WMI class, scope must be set to "\root\virtualization\V2", which is the namespace where Hyper-V locates. Similarly, if users aims at controlling remote machine using WMI, a scope is required to set "IP address\root\cimv2",which is the remote WMI located.

Listing 3.4: Get virtual machine function with WMI API

```
ManagementScope mainscope = new
   ManagementScope(@"\\.\root\virtualization\V2");
mainscope.Connect();
ObjectQuery vmquery = new ObjectQuery(query);
ManagementObjectSearcher vmsearcher = new
   ManagementObjectSearcher(mainscope, vmquery);
ManagementObjectCollection vmCollection = vmsearcher.Get();
foreach (ManagementObject instance in vmCollection)
{
  vm = instance;
  return vm;
}
```

3.3.4 Changing State of Virtual Machines

Since virtual machines information can be acquired, launching/shutting down virtual machines is the function to be realized in the next step. In the previous chapter, we know that *EnabledState* property of **Msvm_ComputerSystem** class indicates the state of virtual machine, however, it is impossible to change the virtual machine state from this property, because of the read-only limitation. Alternatively, **Msvm_ComputerSystem** class has a method of **RequestStateChange**, which enables developers to launch or shut down any virtual machines. The states of a virtual machine, which are represented by different integers, can be changed by modifying an integer to another one. (e.g. 2 in the method represents power on state). The following snippet demonstrates the workflow of change state of virtual machines.

Listing 3.5: Change state of virtual machine with WMI API

```
if (operation.ToLower() == "poweron")
{
 inparam["RequestedState"] = 2;
 }
```

```
    if (operation.ToLower() == "poweroff")
    {
    inparam["RequestedState"] = 3;
    }
   ManagementBaseObject outParams =
       vmObject.InvokeMethod("RequestStateChange",inparam,null);
```

Comparing to VMWare, Hyper-V WMI API provides more efficient function than VMWare because virtual machines' states are represented by an integer, and modification of integer can change the current state of virtual machines. Besides ,the integration of changing states also makes it much easier. while in VIX API, user has to write individual functions for every single state change.

3.3.5 Snapshot Management

Snapshots play an important role in software testing. Essentially, they are the disk, configurations and state of virtual machine in specific time. With snapshots, efficiency can be improved significantly when different computer environments and various conditions in those environment need to be recreated or reproduced many times. Consequently, snapshot management is fundamental in terms of testing in virtual machines, Hyper-V snapshot management therefore is required. Fortunately, WMI provides classes to manage snapshots, namely **Msvm_VirtualSystemSnapshotService** class. It represents the services to create, delete and apply snapshots in virtual machines, and some of its' read-only class can help to gain better understanding of the service. The property and methods are demonstrated as follows:

- *Description* is the very basic property of all class, for **Msvm_VirtualSystemSnapshotService** class, the description is read only and is set to "Service for creating, destroying, and applying virtual machine snapshots"
- *SystemCreationClassName* is a string value read-only property, inferring the name of the class, which is able to hold this servicer, and the value is always set to be **Msvm_ComputerSystem**, which indicates a virtual machine or host machine.
- *InstallDate* indicates the date and time a virtual machine configuration is created, and it is useful to check validity of a snapshot when snapshot tree is adopted.

Generally, the snapshot management follows the procedures below, which is also the basic steps for most of operations using WMI classes:

1. Getting the service object, which means searching the namespace in specific scope.
2. Obtaining method provided by the class using **GetMethodParameters** method provided by the management object type
3. Setting the input parameters, for example, **CreateSnapshot** method has parameters of **AffectedSystem** , and therefore is required to set as the virtual machine path under control.
4. Invoking method with certain parameters and receiving an object representing the output parameters of the method.
5. Reading the value of corresponding properties of output parameters to check status of the method implementation.

In general, virtual machines are required to revert to an initial state after all testing is finished. An initial state can be created by taking snapshot once the guest OS is installed. When it comes to reverting to initial state, **ApplySnapshot** method is provided by the class of **Msvm_VirtualSystemSnapshotService** in WMI . Slightly different with **CreateSnapshot**, a parameter named *Snapshot* representing the snapshot to be applied must be provided. In addition to specifying the snapshot, the virtual machine which the snapshot is created should be provided as well.

GetRelated method is employed to obtain instance of snapshot of certain virtual machine. Specifically, **Msvm_ComputerSystem** represents a virtual system, while **Msvm_VirtualSystemSettingData** stands for a snapshot. The relationship between these two classes is described as Antecedent and Dependent. **GetRelated** method connects these two classes. Therefore, a snapshot for certain virtual machine is returned. Being alike to creating snapshot, applying snapshot requires only setting the parameter of *Snapshot* as snapshot related to the virtual machine, which is the return value of function **GetSnapshot** .

3.3.6 Copying Files Between Host and Virtual Machine

Unlike copying files between host machine and virtual machines installed through VMWare, where files can be copied and pasted directly between two machines, it is more complicated in Hyper-V due to the architecture of type1 hypervisor. The reason behind is that virtual machines constructed through Hyper-v are deemed as independent "physical machines". Therefore it is reasonable that files cannot be copied/pasted directly between two physical machines. Nevertheless, there are existing approaches to share files between host machine and virtual machine. To be specific, sharing files through virtual disk, network and integration service are provided by Hyper-V.

For the first approach, adding a virtual disk to virtual machine is necessary if files are chosen to share through virtual disk. Virtual disk can be considered as a mutual disk between host machine and virtual machine, through copying file to the disk in host machine, files can be found in virtual machine too. The method seems to be proper and efficient. However, it involves in complicated configurations of virtual disks in both host machine and virtual machine. Moreover, virtual machine must be powered off when user is desired to share files.

Second approach, which is more common, is sharing files through network. It is obvious that the basis of this approach is that virtual machine must be able to get access to network, the principle behind is easy to understand. Due to the sharing network between host machine and virtual machine, a sharing directory is needed to realize file communication. The approach is comparatively easier than virtual disk. However, it still requires frequent directory configuration if multiple files are design to be shared.

Evidently, the above two methods presented are not automated solution, which means sharing files is not programmatically controlled but manually set. Therefore a new efficient approach is needed. Fortunately, in Hyper-V WMI V2 version, an integration service is available to copy file automatically. By default, backup option is excluded by integration service, therefore, backup should be selected so as to enable sharing file function in the setting integration service dialog. In Hyper-V WMI provider (V2), an integration service class, which can be used to solve problems that arises from the high level of isolation from virtual machines, is added. **Msvm_GuestFileService** class is designed to enable copies a file to a virtual machine from the Hyper-V host. The necessary properties and methods to copy file is illustrated as below.

- *SystemName* is one of the properties in **Msvm_GuestFileService class** representing the machine name, which can hold the service. In other words, it infers the machine to which files will copy. As a consequence, the value of *SystemName* is set to be the name of virtual machine.

- **Msvm_CopyFileToGuestSettingData** is a class representing parameters for copying file from the host to guest OS

- *SourcePath* is one of the necessary properties in **Msvm_CopyFileToGuestSettingData**, and it indicates the path of source files to be copied, noting that the source path has to be accessible by Hyper-V host OS.

- *DestionationPath* is the corresponding destination path in virtual machine, and the path must be accessible by guest OS. Moreover , the destination file is generated in this path. In this case, the file name from*SourcePath* should be used as destination file name.

- *CreateFullPath* indicates that missing directories in the destination file's path must be generated before copying file.
- *OverWritingExisting* implies that whether overwrite destination file if there is a file already existed.

The method adopted for copying files is **CopyFileToGuest**, an input parameter of this method is *CopyFileToGuestSettings*, which should be set to the instance of **Msvm_CopyFileToGuestSettingData**. The snippet below shows how does the class work to realize copying files to virtual machine.

Listing 3.6: Copy files to virtual machine with WMI API

```
ManagementPath setPath = new
    ManagementPath("Msvm_CopyFileToGuestSettingData");
ManagementClass setDataClass = new ManagementClass(mainscope, setPath, null);
ManagementObject copySetting = setDataClass.CreateInstance();
copySetting["SourcePath"] = SourcePath;
copySetting["DestinationPath"] = DestinationPath;
copySetting["OverWriteExisting"] = false;
settingData[0] = copySetting.GetText(TextFormat.WmiDtd20);
```

3.3.7 Running ProgramS in Virtual Machine

So far, the automated management of Hyper-V is done through the Hyper-V WMI provider (V2), however, in order to run programs automatically in virtual machines. WMI API v2 is not enough. The virtual machines created by Hyper-V are regarded as building on top of hardware directly instead of host operating system, as a result virtual machines are isolated from host machine, which is the reason why copy/paste does not work in Hyper-V. The isolation also makes running program automatically in virtual machine complicated. Besides in Hyper-V WMI provider (V2) classes, there is no class designed to start a process in virtual machines. In order to execute applications in virtual machines automatically, treating virtual machine as remote machine is essential because controlling a machine remotely is possible to achieve.

Connect to a virtual machine's WMI classes

In order to manage guest operating system, connection between host machine and remote machine's (Virtual machine installed) WMI should be established so that management can be achieved by WMI provided in remote machine. Regrading connecting to a remote WMI, the principle is illustrated as Figure 3.3.2. It is clear from the diagram that two connections are needed if an asynchronous call is made, to illustrate. Connection 1 is made by client or script to obtain data from WMI in remote machine, while connection 2 is designed to deliver the result of asynchronous call back. However for synchronous and semi-synchronous call, only connection 1 is required. We can know from figure 3.3.2 that remote connection in WMI is affected by firewall and DCOM, configuration on the firewall and DCOM security must be done properly before remote WMI connection is achieved. The configurations will be presented carefully in the following section.

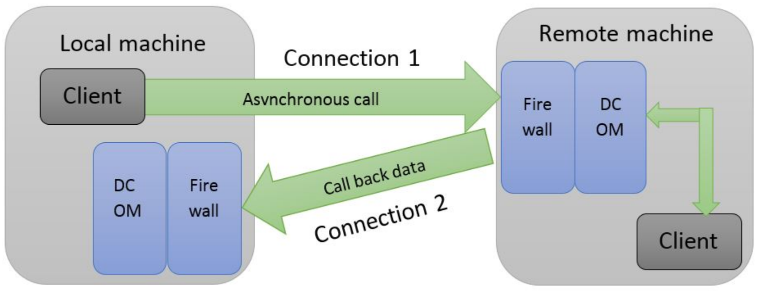


Figure 3.3.2: Remote connection diagram in Windows OS

Firewall configuration

To successfully connect to remote machine, the same username and password credentials identify an account on remote machine. User account used in local machine must ensure that it is a local administrator or domain account in the Administrators group on remote machine. In normal case, firewall plays an important role in filtering untrusted data. By default, data request from remote machine will be blocked by firewall. In order to establish remote WMI connection , either firewall has to be shut down, which is not recommend due to security issue, or firewall is set properly. For connection 1, firewall setting on remote machine must be done locally. In detail, remote administration must be enabled either by firewall user interface or by using NETSH command as below:

netsh firewall set service RemoteAdmin enable

Or if only WMI connection are needed, an exception in the firewall for WMI on the remote machine must be set. Similarly, choice can be made between firewall UI and command prompt using WMI rule group like :

netsh advfirewall firewall set rule group="windows management instrumentation (wmi)" new enable=yes

So far, the firewall setting on remote machine for connection 1 is described above. Some configuration is required for connection 2, which allows delivery the result of asynchronous call back to local host . Firstly, remote administration exception should be enabled same as for connection 1. Specifically , DCOM port TCP 135 must be open. If not , error of 0x800706ba will occur , which indicates that the remote procedure call (PRC) is unavailable. The reason for the error is that the PRC port used by DCOM is closed. DCOM port can be opened by the command prompt as:

netsh firewall add portopening protocol=tcp port=135 name=DCOM_TCP135

Also, unsecapp.exe should be added to the Windows Firewall application exception list so that the result can be delivered back to client. Command prompt is useful to achieve it, which is shown below:

netsh firewall add allowedprogram program=%windir%\system32\wbem\unsecapp.exe name=UNSECAPP

Lastly, for connection 2, it may be an anonymous connection if the remote machine is in a different domain, which will be untrusted by local machine. Therefore, DCOM remote access permission to anonymous connection should be granted on local machine as described in next section.

DCOM security configuration

Distributed Component Object Model (DCOM), a set of concepts and interfaces provided by Microsoft, is an extension of Component Object Model (COM) which enables clients and servers to communicate within the same computer. Traditional COM makes inter-process communication possible, while DCOM enables COM component to communicate across networks. WMI uses DCOM to handle remote calls, which infers that DCOM should be configured correctly to establish remote WMI connection. Typical error of 0x80070005 (Access is denied) will occur if DCOM is not set properly.

In windows, DCOM config utility is designed for configuration DCOM for WMI use, DCOMcnfg.exe can be found with administrative tools through control panel. With the utility, certain user can be granted the permission to connect the computer remotely. By default only administrator or members of administrator group can connect to the remote computer. If the current user is not administrator, activation and launch permissions should be given by adding the user to group or user name lists. Detained setting is available on Microsoft website "Securing a remote WMI connection"

Although Microsoft has provided detailed configuration documentation, sometimes, it still cannot get work done when DCOM has configured correctly, and 0x80070005 still occur. The error obviously shows that problem lies with permission, In this thesis, network service permission is the reason, this probably due to the fact that some network settings are changed by some software silently, the following steps can solve 0x80070005:

1. Right click **My Computer**, and then choose **Management**.
2. In the folder of **Local Users and Groups**, open the folder of **Groups**
3. Double click the **Administrators** listed, **Administrators Properties** dialog box will appear
4. Click **Add** button, in the select **Users** dialog box, click **Advanced**
5. Click **Find** Now button, choose **Network Service**, and click **OK**

The above setting guarantees a administrator have the permission for network service so that WMI can start normally. Configuration of remote WMI connection is trivial. It is error-prone and hard to figure out reasons when theoretic configurations have been applied correctly. For example, in the registry, the value of forceguest under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa should be 0. Also, the user should be granted the permission in the namespace of remote WMI. Generally, if firewall is not set correctly, the error of "PRC servicer is unavailable" will occur, "Access is denied" error is usually caused by DCOM security setting or no permission issue.

Executing application automatically

The configuration mentioned above guarantees that host machine is connected with virtual machine's WMI, however, for controlling virtual machine with WMI classes, either WMIC, which is a command tool provided in windows operating systems, or a script is needed to reach the goal. With regard to the consistency of automation Hyper-V, programs in C# is adopted in the thesis to realize automated management of virtual machine.

For the sake of connecting host machine with virtual machine, except the configuration mentioned above, the IP address or Domain/User name and password of virtual machine is required to establish connection. In order to connect to the WMI classes in virtual machine, the namespace is required set to \ROOT\cimv2, which is default namespace containing the majority useful classes for WMI queries.

With connection between client and the remote WMI (e.g. WMI in virtual machines stored in Hyper-V) established, **Win32_Process** class representing a process in an operating system, is a proper option to execute application automatically in virtual machines. It has various methods concerning a process (e.g. **Create**, which creates a new process, **Terminate**, which terminates a process and all its' threads). The following code demonstrates how an application in virtual machine is executed.

Listing 3.7: Start a process with remote WMI

```
ManagementPath settingpath = new ManagementPath("Win32_ProcessStartup");
ManagementClass startupObject = new ManagementClass(scope, settingpath,null);
ManagementObject startup = startupObject.CreateInstance();
ManagementPath path = new ManagementPath("Win32_Process");
ManagementClass processClass = new ManagementClass(scope,path,null);
ManagementBaseObject inparam = processClass.GetMethodParameters("Create");
inparam["CommandLine"] = prgramPath;
inparam["CurrentDirectory"] = null;
inparam["ProcessStartupInformation"] = startup;
ManagementBaseObject outparam =
    processClass.InvokeMethod("Create",inparam,null);
```

Noticeably, a **Win32_ProcessStartup** object is created to pass information to the **Create** method of **Win32_Process**. To use the **Create** method, some input parameters has to be set. Specifically, *CommandLine* property indicates the command line arguments of the application under execution (e.g. notepad.exe). *CurrentDirectory* is provided primarily for shells that must start a program and specify the program's initial drive and working directory, if the value is set to be NULL, the new process created will have the same path as the calling process. *ProcessStartupInformation* is one of required input parameter, which is usually the instance **of Win32_ProcessStartup** class. By invoking **Create** method with specified parameters, a new process can be created as specified in virtual machines. All variables and methods in Hyper-V automation class are shown as figure 3.3.3

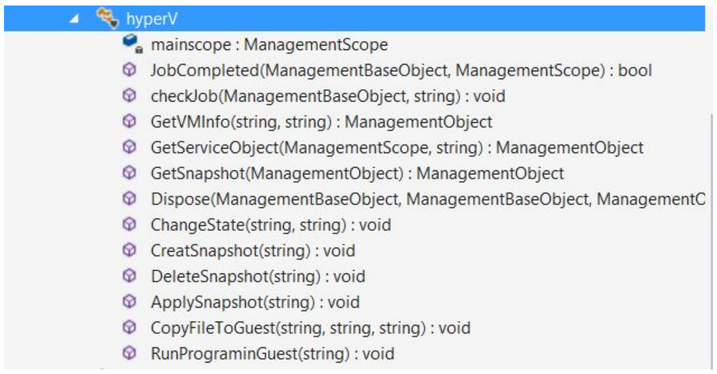

hyperV
mainscope : ManagementScope
JobCompleted(ManagementBaseObject, ManagementScope) : bool
checkJob(ManagementBaseObject, string) : void
GetVMInfo(string, string) : ManagementObject
GetServiceObject(ManagementScope, string) : ManagementObject
GetSnapshot(ManagementObject) : ManagementObject
Dispose(ManagementBaseObject, ManagementBaseObject, ManagementC
ChangeState(string, string) : void
CreatSnapshot(string) : void
DeleteSnapshot(string) : void
ApplySnapshot(string) : void
CopyFileToGuest(string, string, string) : void
RunProgramInGuest(string) : void

Figure 3.3.3: Variables and methods in Hyper-v automation class

From the principle of starting a process in a virtual machine , we know that **RunProgrmInGuest** is independent from other functions (as shown in previous code map) because variables and methods concerning the function is provided by a remote WMI in virtual machine instead of by a local WMI in host machine.

Although WMI class in remote machine (i.e. virtual machine in this thesis) provides valid and efficient approach to start a process from local machine, For security reasons the **Win32_Process**.Create method cannot be used to start an interactive process remotely [2], which means that the process created in virtual machines will run in the background, and will only be visible to task manager. The limitation of **Win32_Process.Create** brings huge disadvantages for GUI testing in virtual machines, undoubtedly, GUI testing requires application runs as a windows on the desktop (i.e. active desktop) . However , without assistance of third-party tool, it is impossible to run application in the foreground. To the best of author's knowledge, with both test controller set to run with test manager and test agent set to run as interactive process , GUI testing in virtual machine is possible. Nevertheless introducing test controller is not expected in this thesis, because it makes automation more complicated, especially compared to the simple solution available in VMWare.

3.4 Comparison of VMWare and Hyper-V

One of the purposes of this thesis is obtaining the detailed differences of hyper-V and VMWare so that best solution for GUI testing with virtual machines can be found. In addition to the theoretical analysis about the architecture and principle in previous chapter, more detailed comparisons in term of practical use in various aspects are listed below.

- Being consistent to theoretical analysis, VMWare is easier to configure than hyper-V. For example, by default host IP address is shared with virtual machines in VMWare, while for Hyper-V, user is required to build new virtual adapter if network access is needed. Also, copy/paste is supported in VMWare but not in hyper-V.

- For virtual machines in Hyper-V, restarting guest OS is required as long as the guest OS is modified, for example in case of Windows updates, or turning off windows firewall and so on. But, changing system setting does not result in the restart of guest OS in VMWare.

- Virtual machines residing in Hyper-V can be run in the background with the start of host OS, while the ones in VMWare must be powered on either manually or by script.

- By contrast with theoretical analysis, a virtual machine in hyper-V runs slower than the one in VMWare with the same configuration of 1GB memory and 1 virtual processor. However the conclusion may be incorrect because the version of host OS is different, specifically, Widows 10 professional for Hyper-V and Windows 8 professional for VMWare.

- For automated test usage, Application Program Interface (API) is important to achieve the goal, both products provide APIs to programmatically controll virtual machines, VIX API for VMWare, and WMI for Hyper-V. As for the resources available online (from official documentation to open source community), WMI API for Hyper-V is more abundant.

- In a view of the functionalities provided by VIX API and WMI, WMI puts more weight on the management of virtual machines, like programmatically resizing virtual hard disk, modifying resources pool settings and so on, while VIX API focuses more on guest operation management. To illustrate, listing file information in guest operation system and running programs are easy to realize with a few lines in Vix API. And automated test in virtual machines demands more operations in guest OS instead of management on virtual machines' configurations, which makes VIX API more preferable.

- Taking accessibility of APIs into consideration, VIX API is easier to use if C# is adopted. But, in order to use WMI, users has to adopt the format of Windows Query Language (WQL), which is similar to Structured Query Language (SQL) and probably is not friendly for users who is accustomed to C#.

- With WMI, automatic mouse click and keyboard input are easy to achieve in virtual machines, however it is not provided by VIX API. With consideration of the way host machine connects to virtual machines, VIX API and WMI works in totally different way. Due to Hyper-V architecture, which is designed directly on hardware, it is necessary to take virtual machines in Hyper-V as remote physical machines. Consequently, connecting to them requires network. As for VMWare, network is unnecessary due to the fact that virtual machines are built as a part of host OS.

- As for guest OS management, it is impossible to create interactive process with remote WMI in guest OS, process can only run silently, and can only be visible to task management. However, for VIX API, interactive process is totally possible.

- Due to the limitation of remote WMI, GUI test with in virtual machine stored in Hyper-V are theoretically impossible with only test agent installed in guest OS, because active desktop is a big challenge for remote WMI. However, with test controller installed , GUI testing can be realized.

The table 3.4.1 demonstrates briefly the differences and similarities in term of practical use for GUI test.

Table 3.4.1: Comparison of VMWare and Hyper-V

Parameters	**Hyper-v**	**VMWare**
Configuration	Complicated	Easy
Automatic power on VM	Possible	Impossible
Guest OS modification	Require restart	Restart not required
VM performance	A little Slow	Fast
API resources	Resourceful	Official documentation
Accessibility of API	WQL required	User-Friendly
API functions	Mainly on VM management	mainly on guest OS management
Low-level control	Possible	Impossible
Remote control	Network required	No specific requirement
Interactive process	Impossible	Possible
GUI test	Test controller and agent	Only test agent
Snapshot management	Easy	Easy
Automatic log in guest OS	Impossible	Impossible

WMI and VIX API have their respective pros and cons. They are similar at some points, e.g snapshots management can be doe easily with both APIs, and automatic logging in guest OS is unachievable for both. However, for GUI testing, VMWare is preferable. The greatest weakness of Hyper-V is that GUI of applications is invisible when the application is executed automatically. Furthermore, the easy use of Vix API and its efficiency makes VMWare suitable.

It is well known that in theory, Hyper-V has a better performance than VMWare. As discussed in previous chapter, Hyper-V has great advantage over VMWare when large amount of virtual machines are installed. Considering the case in this thesis where only limited number of virtual machines are needed, Hyper-v does not have greater advantage than VMWare, especially taking the cost of hardware required by Hyper-V into account. To sum up , VMWare is selected as the virtual machine software due to its high accessibility and resourceful guest OS management methods provided.

4
Automated GUI Testing

In previous chapter, automating virtual machines in both VMWare and Hyper-V is presented, and decision about which suits better for GUI testing is made based on the differences in theory and practical use. In this chapter Necessary knowledge about GUI testing will be introduced , including the basic concepts and various methods and tools available at current stage. A comparison of different tools is given, according to which Coded UI test is chosen for this thesis, following with the implementation of the solution proposed in the thesis namely fully automated GUI testing with machines.In the last section,evaluation of the solution is presented.

4.1 Graphical User Interface (GUI)

Graphical User Interface (GUI), which is the user interface displaying computer operation environment with graphs, plays an important role in modern applications. Compared with the commands adopted by early computer systems (e.g. Disk Operation System), GUI brings significant convenience to users. Nowadays, users interact with program with GUI, and the output of users input (e.g. click button or keyboard input) can be visible immediately in the format of graph.

Nowadays, most of operation systems have provided graphical user interface, like Windows provided by Microsoft or Mac designed by Apple. Programs with GUI is always easy to use by freeings people form complicated command language. For majority of applications at present, they are always installed with GUI applied, which makes GUI testing vital in development cycle.

4.1.1 GUI Objects and Events

GUI is constituting by a collection objects that are placed inside one or more panes/windows, which can be buttons, menus or windows. Generally, a GUI object represents a screen element that is used to display information or enable users to interact with software in a certain way. Each user interaction with the graphical environment, like button clicking or keyboard input causes an event and it is event that manipulates GUI object. Event changes the state of software, which is reflected by the change of appearance of one or more objects. Naturally, GUI is hierarchy, that is to say, a GUI object has its' own parent, siblings and children.

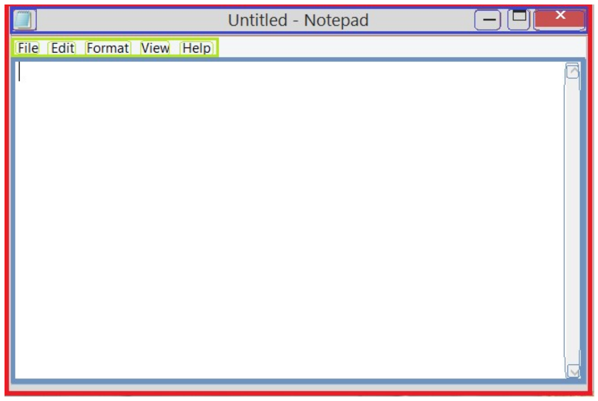


Figure 4.1.1: Graphical User Interface of notepad application

The GUI objects of notepad are shown in figure 4.1.1, they are bordered with different colors. Typically all GUI objects reside in a container, which is known as windows object. As the example indicates. Windows object highlighted with red with the name of Untitled-Notepad holds all GUI components. As a consequence, windows object is the ancestor of all objects in notepad. In detail, windows object has three children, title bar object with name of untitled notepad colored with indigo, green menu bar which has five menu items as children, and lastly document object with gray blue. All these objects have unique set of properties, which usually are identified with discrete value. At any time of execution, the set of value may change, which indicates different states of software. For example, the name of Document object, can be modified from none to anything input form the keyboard, which represents that the text file is no longer empty. Detailed information about untitled notepad GUI hierarchy structure is demonstrated in figure 4.1.2.

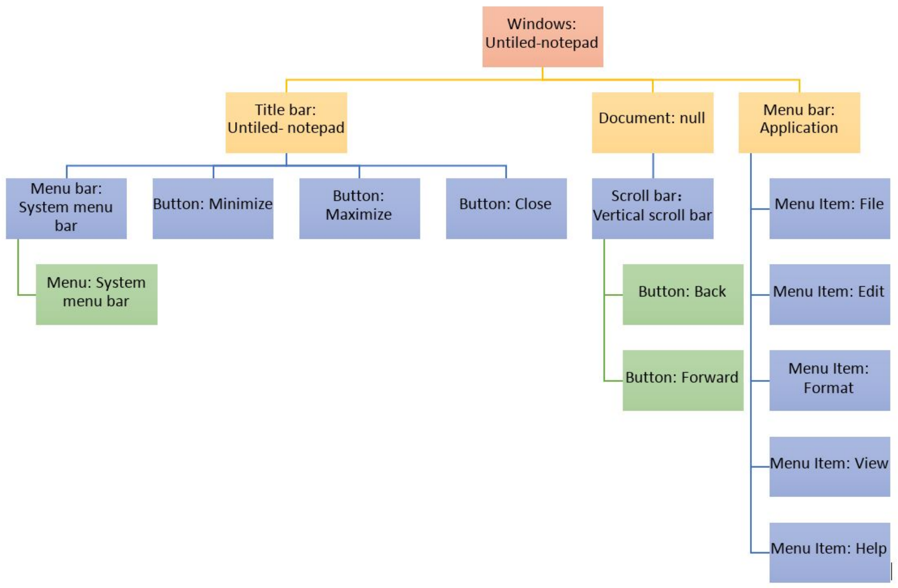


Figure 4.1.2: Hierarchy structure of GUI objects in untiled notepad

As the figure illustrates, the Windows object has three children, title bar, document and menu bar. Each of children has their own properties indicating the state of application, e.g. the name of document is none in the figure which means that the notepad file has no content. The hierarchy structure tells the children of each objects by different colors. It is worth to study the hierarchy structure of GUI application due to the way certain object is searched in testing process. For example, if developers tend to find the menu item file, first step is to find windows object, next object to find is its' child menu bar, and finally menu item file is presented.

For developers, the hierarchy structure of GUI can be exploited to identify GUI events, which is the basis in testing. According to the paper [6] , events are divided into different groups due to the corresponding objects they are related to. They are itemized as :

- **Restricted –focus event**: apparently, user is restricted to operate for this type of event. For example, in the notepad, when the user perform set font event, a window named **Font** appears and users choose font from the lists. Finally, user terminates the interaction by either **click** ok **or** cancel. In this case, set Font is restricted-force event. Termination must be performed in these classifications.

- **Unrestricted-focus events**: is opposite from restricted-force event. For example in notepad, performing event **Find** opens a window named Find ,and user can input anything they want to **Find**

- **Termination event**: typically, click **ok** and **Cancel** appears termination event.
- **Menu-Open event**: is used to open menus. Typically, File and Edit is menu-open event. This type of event does not interact with software.
- **System-interaction event** : interacts with underlying software to execute some action. For example **copy** event.

4.1.2 Characteristics of GUI Application

Being aware of characteristics of GUI application is instrumental to execute test. From the previous introduction, it is reasonable to draw summary for the characteristics shown below:

- Accessing to features and functions of software or systems is provided via various GUI objects, like menu bars, buttons, and keyboard shortcut. GUI applications set the low-level logic of the software apart from users, which makes the software efficient to output expected results, and therefore is convenient to correct errors if any occurs.
- Objects of GUI are diverse and in hierarchic architecture. An objects may contain several other objects, and these objects probably contain various types of other objects too, which forms the hierarchy of GUI.
- GUI applications allow multiple windows to be displayed at the same time.
- The state of GUI applications is event-driven, users interact with applications through certain events like mouse click, keyboard input. Consequently, the value of properties of certain object changes. and therefore the state of application is modified.
- The output of GUI applications execution is not decided only by the event at present, also the initial state and event performance history influence the output.
- GUI applications rely a lot on operating systems. In some cases, operating system functions are called by GUI applications so that the applications behave correctly. As a consequence, the state of external device and operation system have a significant impact on GUI applications, especially, in the circumstance that GUI applications' functions are realized by the interaction between system API and the codes it holds.

4.2 Automated GUI testing

The existence of GUI applications brings huge convenience to users, at the same time the popularity also makes GUI application more and more complicated and hence, difficult to execute GUI test. Literally, GUI testing represents the process of testing an application that adopts graphical user interfaces to make sure that it meets the required specifications. From a technical point of view, GUI testing is mainly about verifying whether GUI objects act as expected and the state of application is changed as expected after certain event is performed. Generally, it is about verifying the function of GUI components, and verification on all states of application is included.

4.2.1 Challenges of automated GUI testing

It is clear that GUI testing differs from traditional testing in various ways. E.g. as for traditional tests, the input of testing is a set of data. However, for GUI application, as mentioned before, is event-driven, the input of GUI application is no longer conventional data set, but a series of event flow. An event may be trigged by another event. Furthermore, there is no distinct output from running GUI applications. Failure of certain event may make the rest of events fail to perform. Therefore, verification of states change should be performance after every single event execution, instead of after completion entire tests. GUI testing brings challenges in this ways. There are also other aspects listed below shown the challenges:

1. As the characteristics of GUI applications indicate, the output of GUI application execution not only rely on the current input but also on the historic performed events and initial state. The feature adds complexity to GUI testing.

2. There exists great amount of state changes in GUI applications, which should be all included in test. This requires that the states and architecture of GUI applications should be monitored at any moment.

3. A test path is essentially a event sequence , it can be generated from permutation and combination of GUI objects and events. Due to the large number of GUI objects and events, the number of test paths, which work as input to the application therefore, are tremendous.

4. The changes of GUI applications state are caused by events, for example a simple click button can change the graphical interface. However, commands or messages from system can also change states of the application. Therefore, uncertainty exists in GUI testing.

The challenges listed above make automated GUI testing difficult, especially when compared to conventional testing methodologies. Currently, GUI testing still relies a lot on manual tests, in which, testers click buttons or input keyboard physically and verification of states is also done in the same way. Apparently, manual test has a great amount of disadvantages, for example, it is very time consuming and testers may easily ignore some test paths. Automated GUI testing is needed especially in industrial area.

4.2.2 GUI Testing Approaches

The importance of GUI application and its' unique challenges has raised concern in both academic and industrial area. A lot of work and researches has been done to propose automatic methodologies. At the current stage , the most common and popular way is to use capture/replay mechanism, in which human actions on GUI application is recorded by certain tools, GUI testing is done by the automatic playback of the previously script recorded. Also , there are some other methods to realize automated GUI testing, e.g. employing the Finite State Machine (FSM) to simulate the interaction made between user and application. FSM model overcomes the defects in capture/replay to some extent. However, it is usually complicated to construct models. There are also other solutions in addition to the two mentioned above. In general, the majority of automated GUI testing methods can be characterized [5] in the figure 4.2.1 below.

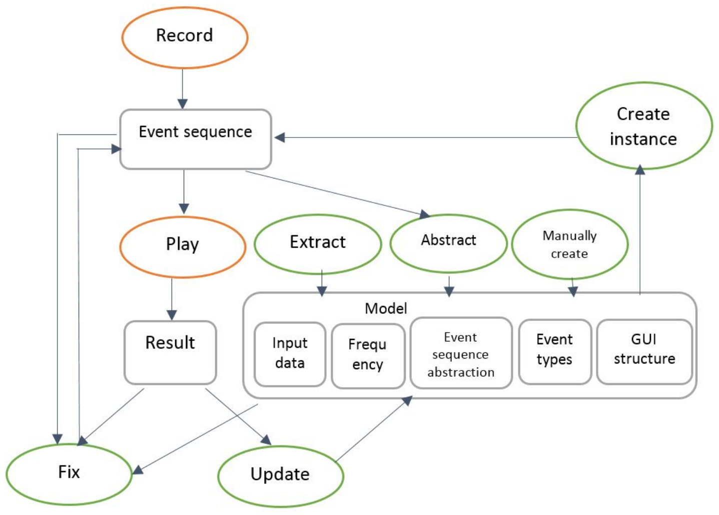


Figure 4.2.1: Brief overview of automated GUI testing methods

The graph in figure 4.2.1 demonstrates briefly the GUI testing process. There are two types of elements in the graph respectively elliptical representing activities, and rectangular denoting the data collections. Each edge connects an activity and a data collection. An important purpose in automated GUI testing is to eliminate the need for a human user during testing , which means running the GUI applications automatically. In other words, running the interactive application (*play* activity) with a certain event sequence (event sequence data collection in the graph).

Taking the most common capture/replay (also known as record/replay) into account, the approach includes a record and replay process (*record* and *paly* activity respectively). The former aims at recording users' behavior when users interact with applications. As a consequence, **event sequence** is generated. Alternatively, there are other automated GUI testing approaches focusing on models, which represents a set of possible event sequences. In these approaches, specific event sequences can be created through a model by generating instances of a model. A model may include various data collections. To illustrate, **event sequence abstraction** can be a part of models, Finite State Machine [16], event flow graph[24], or Markov model [7] can be adopted to build event sequence abstraction. Besides, a model may contain information about **GUI structure**, like the hierarchy structure of notepad mentioned before. In addition, event types is also instrumental in building an event sequence, **frequency** holding the information about which components are most used or most significant in practice. At last, **input data** is included in models too.

In terms of creating a model, there are various options. One possible way is to abstract models form one or more specific event sequences (*abstract* activity). Alternatively, extracting information from GUI applications is another choice (*extract* activity), e.g the GUI hierarchy structure of notepad can be obtained through analysis the graphical interface. Hence, searching for certain objects and generating event sequence is possible. Last but not least, manual creating models is also one of important approaches. It is efficient especially for small or medium sized GUI application, where test paths are limited.

Apart from creating models, maintaining models or event sequences to gain better performance is also necessary in automated GUI testing. Generally, the maintaining strategy is based on the results of previous playback, which can be both pass and failure. As for conventional record/replay approaches, results can be also used to fix the event sequences. Besides, the information from a model can be utilized to fix a specific event sequence, like the GUI structure change information.

Among all automated GUI testing approaches, each method has respective pros and cros. As for model based approaches, although they overcome the defects of record/replay and have high test coverage, most of them require long development cycle and testers are required to be equipped with good programming ability. Moreover, models are typically time consuming to maintain and update. From development's view, testers need to develop different models for different applications, which is inefficient and expensive to maintain, particularly, in industry area, where efficiency and feasibility for implementation is emphasized. Alternatively. Record/replay technique gains popularity due to high efficiency and feasibility.

4.3 Coded User Interface test (CUIT)

Currently, there are many record/replay tools on the market, e.g.Selenium, QTP, and Coded ui test . Coded UI test is a record/replay test tool provided by visual studio for automating GUI tests . Users can verify that the whole application, including its user interface, is functioning correctly. Coded UI Tests are particularly useful if there is validation or other logic in the user interface.

4.3.1 Advantages of Coded UI Test

Similar with other record/replay tools, performing automated GUI testing with coded UI test consists of two steps. First comes the recording , in which user inputs low-level system interaction to the application under test. Automated test is performed in second step, which is automatically playback the recorded script in first step. Among all tools, coded UI test is chosen due to its huge advantages compared to other tools. The strength can be listed below:

- Because CUIT is designed by visual studio, it is easy and efficient to writes script and debugging due to all features of Visual Studio are applicable.
- CUIT can either be executed with Visual Studio or by using Microsoft Test Manager (MTM). With MTM various settings are provided for the test case execution so as to gather a lot of information while executing the test case behind the scene. CUIT provides various test settings to perform test cases in order to capture different data when a bug is created
- For CUIT users can write the script with Visual Studio, making all object programming concepts applicable if required.
- CUIT supports Windows Applications, web applications, WPF applications, SharePoint, Office Client applications and dynamics CRM Web Client applications.
- Coded UI Test supports any data source supported by .NET framework which can be in the forms of a .CSV file, XML file or any other data source like SQL Server table, Access table etc.
- Coded UI Test includes a rich API library to code against and a resilient record and playback tool. It can be extended to support custom controls as well.

- For CUIT and MTM ,users can be provided all the Application Lifecycle Management (ALM) support Team Foundation Server provides. It supports work item tracking, source control or version control and build automation. The support is in-built;

Table 4.3.1: Comparison of different record/replay tools

Category	**Selenium**	**QTP**	**CUIT**
Record/playback	supported	supported	supported
Ease of IDE	Not sufficient	Not sufficient	Completely supported
Ease of execution	Supported	Supported	supported
Language	Supported	Supported	supported
Object supported	No	No	Supported
Application	Only web	Almost all	Almost all
ALM supported	Partially	Partially	Supported

The table above compares CUIT with other popular tools, QTP and Selenium respectively . The comparison is made in various aspects, from ease of IDE and features to the object-oriented language support. It is apparent that CUIT is more powerful and more comprehensive than other GUI tools , In particularly, third-party plug-in and Application Lifecycle Management is supported by CUIT, which makes it suitable for industrial use.

4.3.2 Principle of CUIT

Obviously, the most significant part of record/replay technique is the record and play back engine. Figure 4.3.1 illustrates the high-level architecture of record/replay engine. In general, the architecture is constructed by three levels colored differently in the figure. Specifically, the recorder is designed to listen and capture UI actions.Playback & API component aims at replaying the actions recorded through interpreting codes generated. In order to simplify the way to call detailed UI technologies, technology abstracting layer is set to provide a consistent interface to call into different UI technologies. Hence different plug-ins components used to identify different types UI controls, are designed under technology abstracting layer. In the figure 4.3.1, web plug-in will be picked up for web applications. Rich client plug-ins includes UI Automation (UIA) and Microsoft Active Accessibility (MSAA), the former plug-in is picked up when the GUI application is Windows Presentation Foundation (WPF), while for other applications, e.g. Windows Forms applications, win32 applications, and Microsoft Foundation Class (MFC) applications. MSAA plug-in is used to identify UI controls. Generally speaking, MSAA works for any controls which is not specified in web plug-in or UIA plug-in, which is the reason why MSAA is default plug-in. Third party plug-ins is supported by the engine too.

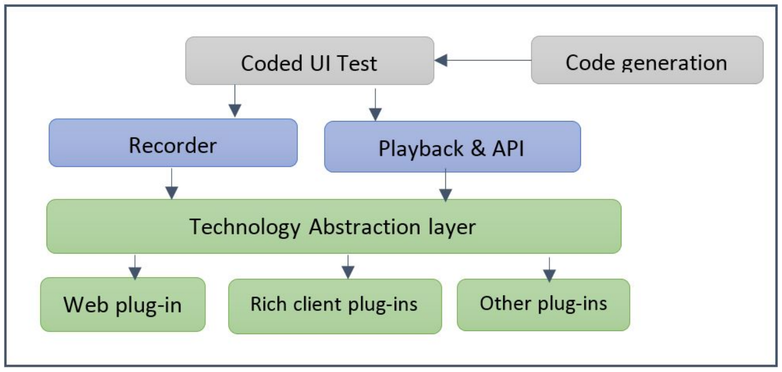


Figure 4.3.1: Architecture of record and replay engine in Coded UI test

Recording UI actions is the basis of CUIT, a simple example is shown to illustrate how the record/replay engine works. A simple mouse click on a WPF applications provokes the following steps at high level for the recorder:

1. Recorder listens to Mouse Event.
2. X and Y coordinates will be acquired when the mouse button is clicked.
3. Recorder will call specific API to get the Control (e.g. button) at location of X, Y (for WPF applica-tion, and MSAA plug-in will be called to identify the button.)
4. Recorder will get the properties and the hierarchy of the button.
5. Information about technology type of plug-in, control hierarchy and control properties will be generated, which is used to search the button in playback process.
6. Capturing the actions, which means mouse click
7. Generating an XML file to represent the recording

Replaying step is one of the most important part in CUIT. Playback engine can be invoked through automation API in CUIT, and playback engine involves the following steps at high level:

1. In order to find the specific control, like a button in previous example, the information generated in step 5 during recording process is adopted to search for the control. Breath first search (BFS) algorithm is used in searching process.

2. When the specific control is found, the playback engine will ensure the control is visible by performing some actions.

3. Before interacting with the control, playback engine needs make sure that the control is ready for specific action, like clicking on a button, some smart algorithms are employed to achieve this goal.

4. Playback tries to ensure that the control that was supposed to have received an action has actually received it.

5. The playback finally performs the UI action on the control.

4.3.3 Workflow of Coded UI test

The playback logic is described clearly in previous section, how CUIT performs GUI testing is important for testers to analyze failures, figure 4.3.2 demonstrates the general workflow of GUI testing with CUIT.

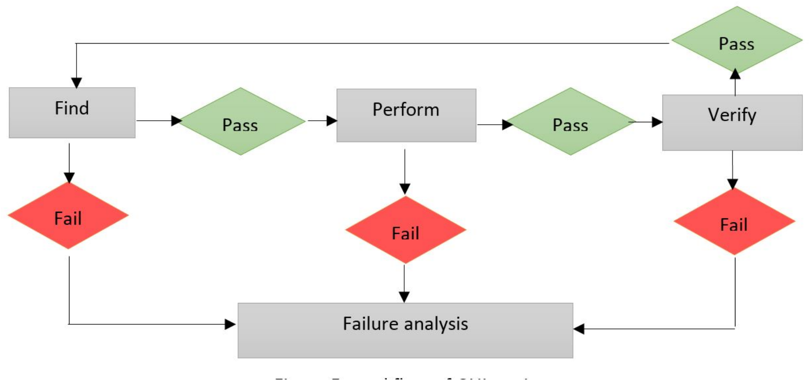


Figure 4.3.2: Workflow of Coded UI test

In figure 4.3.2, rectangulars colored in grey represent activities, while diamond indicates the outcome of GUI testing, either pass or fail. In general, CUIT will perform Find first, which is responsible for finding the specific controls (or finding next control) according to the recorded information (e.g. controls' name, type, hierarchy etc). If the control is not found, then test will end up with "cannot find control with search properties" error, otherwise CUIT will continue to execute Perform activity. If the action cannot be performed, CUIT will fail with "action cannot be executed", the reason behind may varies a lot for different tests. If the action can be performed successfully, CUIT will verify whether the expected state reached , if not, testing will fail too. Apparently, failure information is instrumental in fixing the test script, and therefore failure analysis is also one of the most important step in the entire workflow. Noticeably, the testing will not quit unless there are failures or in the preform activity , some termination events are performed ,like clicking finish or cancel button

4.3.4 Approaches for GUI test in virtual machines

For GUI test using the virtual machines, there are basically two approaches. Firstly, hardcoding the mouse click and keyboard through API, which requires that testers know the absolute position of various GUI components, and failures are prone to appear if the position of program under test moves even a slightly bit. Or in another way, click the name of certain component, similarly, if the name is changed, failure will occur. Another choice is to implement GUI test in virtual machines by dropping test script to guest operation system.

VMWare provides several different software development kit (SDK) products, and each is designed for different community and platforms. To illustrate, VMware HTML Console SDK can be only used for vSphere 5.1 and later, which is a JavaScript library implemented on a basis of WebMKS, provides mouse , keyboard processing and handle as well as cursor changes. While VMWare VIX API is designed for users to control VMWare guest OS programmatically. The advantage of the pattern is that it is easy to download and get what you need for developments clearly. However, when a developer targets at some comprehensive functions, it usually requires extra product. For example, VMWare workstation is only designed for virtual machines which are installed on the exactly same host, and controlling mouse click and keyboard is impossible. This is the reason why dropping test script to guest OS is adopted in this thesis.

Coded UI test provided by visual studio is automated test, which drives application under test through its user interface (UI), and can realize functional test of UI controls and verify the functionality of applications. It is very easy to use in terms of record and playback. Consequently. It is employed in this thesis as automated GUI test tool.

4.4 Evaluation of Coded UI test

Currently, most of currently existing research work has put the weight on proposing new approaches to perform GUI testing, e.g. Marlon et al[37],proposed a test case generation approach based on

Unified Modeling Language (UML), while Emil et al[3] introduced a new technology using image recognition to identify the GUI objects. However, all of these works focus on the approaches to generate GUI testing, the system where GUI testing is executed on has to be turned on manually. Therefore, additional manual assistance, which can be turning on systems, installing pre-requisite applications or switching off system when all tests finished, has to be available for theses test cases to be executed successfully. To some extent, the approaches that current research proposed are only half-automatic GUI testing in terms of the entire testing process. To address this problem to some extent, a fully-automatic GUI testing solution is presented by locating tests in virtual machines. Through automating virtual machines, all the procedures involved in testing process are automated. Besides, multiple virtual machines can be automated concurrently, which means the efficiency of testing can be improved significantly. In addition, evaluation of the on Coded UI Test is given in this paper since no work evaluated the tool especially in industrial cases. The following chapter shows details of evaluation related to CUIT in both host machine and virtual machines

4.4.1 Installation/Uninstallation Test

Installation of an applications is the premise of using the software, at the same time brings changes to the system. In some scenarios, a failure installation may cause severe damage to users' system, leading to a reinstallation of the entire operating system being necessary . From user experience point of view, installation is the first step of using certain software. If installation fails, it will leave a bad impression to the customers. All the circumstances mentioned above indicate that installation test is critical in testing process. Additionally, installation in different systems and configurations should be implemented to guarantee the robustness of the application under test.

For uninstallation, it should be emphasized that successful uninstallation removes the application from a system completely, which means that all files relevant with the application should be deleted, otherwise the installation of new version of application may be influenced significantly. Typically, uninstallation of an application can be done through either uninstall.exe (in some cases uninstall.exe is integrated in the setup file) or uninstall function available in control pane. However considering the circumstance that control pane may have different forms of names in different operating systems, e.g. in Norwegian operating system, control panel is "kontrollpanel" , it is efficient to hide the difference in GUI testing process, which is sensitive to UI properties. Thus uninstallation is done through uninstall.exe in this paper,.

Figure 4.4.1 (a) and (b) illustrates the workflow for installation and uninstallation respectively, notice that both tests are done through executing the setup file. The difference lies with that uninstallation is achieved through Remove function provided by setup file and this function is only available when application is installed successfully.

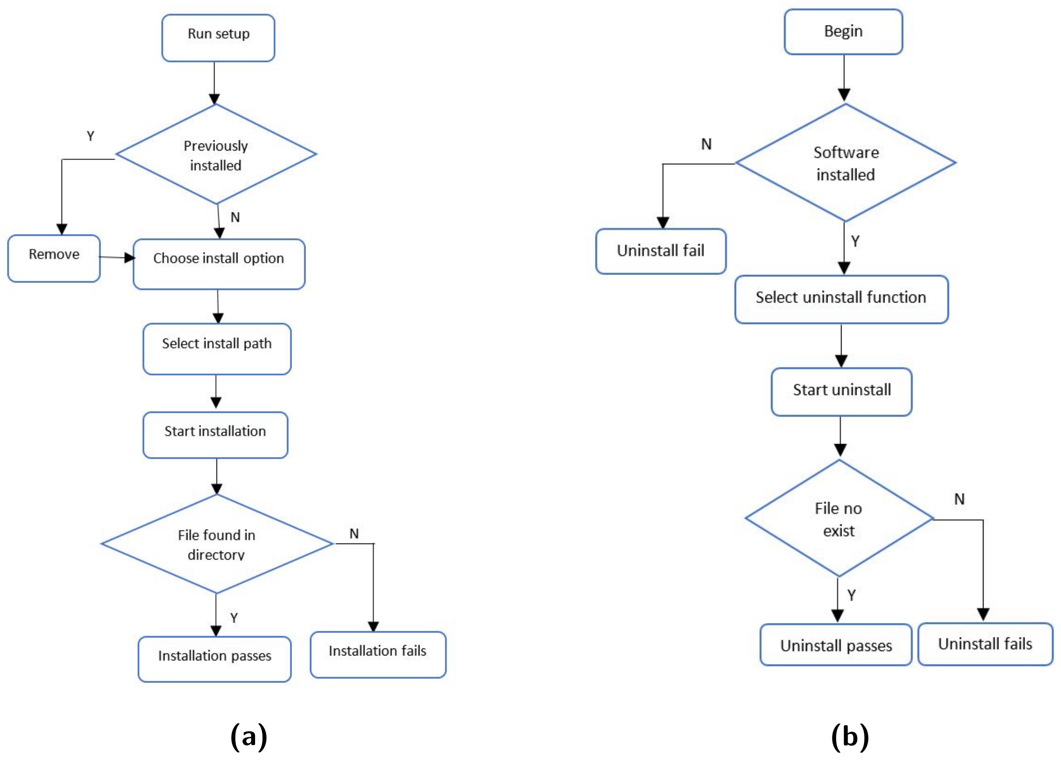


Figure 4.4.1: Installation/Uninstallation GUI testing workflow

As for installation test work flow shown in figure 4.4.1 (a), test starts with running the setup file for the application, and if a previous installation is detected, all the files related to the application from previous installation will be removed. If there is no previous installation found, installation test will start the ordinary install procedure, including choosing installation option or selecting install path. Notice that in this paper, the install path is set to be default directory. The test does not end up with the completion of installation, a followed validation of application installed is necessary. The validation can be about the size and version or any other properties of application installed. The pass or failure of test will be available from the outcome of validation.

Uninstallation test demonstrated in figure 4.4.1 (b) starts with check whether the application is already installed. If not the test will fail with no application installed. Otherwise, uninstallation will start normally from running setup file and choosing Remove function. Similar with installation test, uninstallation test completes with validation whether all files are removed from specified directory. Thus the outcome of uninstallation test can be generated.

In order to illustrate the efficiency of automatic GUI testing for installation and uninstallation, a small sized application called TestApp together with an industrial software named ABBRobView 5, which is designed and released by ABB Robotics AS, is adopted for GUI testing in the thesis. These two software varies in many aspects. One of the most influential sides with regard to completion time taken, is the size of the setup file and application itself. As a consequence, the value of setup file and application is obtained. Furthermore, GUI application is event-driven. The number of events involved in installation/uninstallation testing process also affects the execution time. Table 4.4.1 below shows the time taken to complete testing both manually and by script in host machine.

Table 4.4.1: Execution time of Insatllation&Uninstallation test for TestAPP and RobView 5

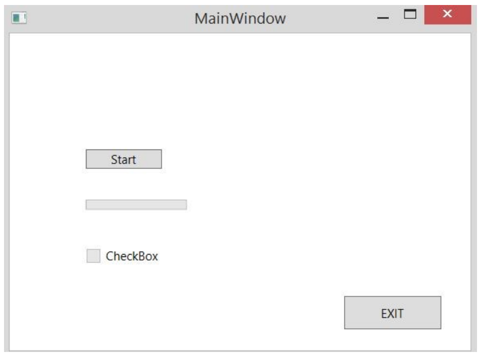
Test	No. events	setup size	App size	Manual test duration	Automatic test duration
Installation TestApp	3	268K	276K	6s	6s
Uninstallation TestApp	4	268K	276K	7s	6s
Installation RobView	3	43.8M	103M	12s	8s
Uninstallation RobView	4	43.8M	103M	20s	17s

Table 4.4.1 shows clearly the completion time for both manual test and automatic test with CUIT for applications with different size. As for the application with small size, like TestApp in the table with size of 276k, automatic test does not have apparent superiority over manual test. While for application with a medium size, CUIT shows great superiority. To illustrate, the time taken to finish manual test for installation of RobView is 12s, while the duration is 8s for automatic test with CUIT. The performance in terms of completion time improved by around 33%, and for uninstallation test, the value is 15%. Remarkably, the number of events is introduced in the table, which is the number of actions involved in testing, e.g. for installation, testing includes **click** accept checkbox, **click** install button and **click** finish button. Number of events for installation and uninstallation test is 3 and 4 respectively, and both are very small. One of significant reasons for automatic test having no obvious advantage over manual test is that too few events are involved in testing process. Nevertheless, automatic GUI test with CUIT adopted is still superior over manual test especially for medium and large scale of programs.

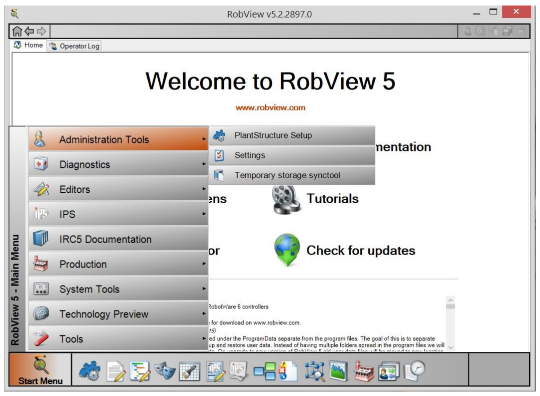
4.4.2 GUI Functional Testing

As the term indicated, functional testing aims at verifying whether an application or system under test behaviors as expected. In traditional testing, functional testing takes data as input, and verifies whether the actual output is consistent with expected outcome, which is usually written in script. However, for GUI testing, there is no data as input since GUI application is event-driven. Sequence of events can be considered as input in some content. The changes of state caused by each event will be compared to expected change. GUI application functional testing is more complicated than conventional ones in terms of input data and verification part.

Similar with installation/uninstallation test, GUI functional testing for both small-sized and medium size applications is shown in this pape. SimpleAPP has a single function as shown in figure 4.4.2 (a) . The application is made of 4 components, and once **start** button is clicked, the progress bar will start working. When progress bar finishes, **check box** will be enabled, (before progress bar finishes, check box is always disable, colored with grey) . **EXIT** button will be clicked after clicking check box ,which closes the application. The whole testing process can be summarized as: click start-wait for progress bar-click check box-click exit. The number of event is 3.



(a)



(b)

Figure 4.4.2: GUI of SimpleAPP and Robview 5 for GUI functional test

With regard to functional testing of RobView 5, it is much more complicated. The number of test cases can be up to 67 if all functions are set to be covered. In order to illustrate the performance of CULT, a test on the function of administration setting is chosen. Specifically, the tests is designed to verify the function of logging in RobView with a certain account. As shown in figur4.4.2 (b), notice that by default, User Account Setting (UAS) is disabled, a restart of RobView is required. Another restart of the application is required after the user name and password is verified so that the application can start with the specified account. To sum up, the number of GUI events involved in UAS functional test is 18. Obviously, functional testing on RobView is much more complicated and time consuming than SimpleAPP

Table 4.4.2 below demonstrates the time taken to execute functional tests manually and by script in host machine(windows 7 OS). Due to the difference of number of events, time taken varies significantly for SimpleAPP and Robview 5. With regard to manual test, duration for Robview 5 is 2 minutes and 20 seconds, while for SimpleAPP , it is much less, only 6 seconds. Similar with installation/uninstallation test, for small-sized GUI applications, CULT does not bring much superiority. For functional test, performance of UAS functional test for Robview 5 improves by 11.4 % . Notice that, for installation/uninstallation test, the value can be up to 33%. The difference results from the fact that in functional testing, it takes longer for CULT to find certain GUI component in a complicated hierarchy structure, which is one of the disadvantages in CUIT. However compared to manual test, CUIT is still faster and more advanced.

Table 4.4.2: Execution time of functional test for SimpleAPP & RobView 5

Test	**No. events**	**Manual test duration**	**Automatic test duration**
SimpleApp	3	7s	6s
RobView 5	18	2min 20 s	2 min 04 s

4.5 Evaluation of Fully automated GUI testing with virtual machines

Virtual machines have brought huge advantages to users. Especially for software testing, with virtual machines, various systems can be built in an inexpensive way, which is instrumental in testing the robustness of application. Moreover, considering the circumstance that "malware" may attack system where application is installed in the testing process, damage may be brought to the system. With executing tests in virtual machines instead of host machine, it is cost free to fix the problem, either by reverting to snapshot or reinstalling the virtual machines. While for fixing the damage in host machine, it is costly to reinstall the entire operating system.

Automating virtual machines is comparatively easy to achieve in an inexpensive cost with the efficient API provided. As a consequence, virtual machines are excellent candidate to solve fully automated testing issue, which means that all the procedures involved in testing can be done automatically in virtual machine and the test results can be copied back to any directory in host machine. Evaluating GUI testing in virtual machines is therefore critical. If the tests executed in virtual machines take extremely long or guest OS responses extremely slow, then virtual machines are not an efficient solution for fully automated testing. Table 4.5.1 shows different execution time of tests mentioned above in different virtual machines. Notice that test in virtual machines is automated by playback coded UI test script, and command prompt and visual studio test agent (MSTest.exe) is employed to automated testing in virtual machines.

Table 4.5.1: Execution time of sets of GUI tests in various virtual machines

Guest OS	**TestApp Installation test**	**TestApp uninstallation test**	**RobView Installation test**	**RobView Uninstallation test**	**SimpleAPP function test**	**Robview function test**
Win7EN	20s	11s	44s	29s	9s	3 min09s
Win7CH	18s	11s	40s	25s	10s	3min02s
Win7NO	22s	13s	42s	28s	7s	3min01s
Win10EN	25s	22s	1min 04s	57s	22s	—
Win10CH	19s	20s	51s	48s	13s	—-
Win10NO	23s	18s	1min13s	1min02s	16s	—-
Average	**21s**	**15s**	**52s**	**41s**	**12s**	**3min04s**

Skimming through table 4.5.1, it can be seen that duration varies a lot in different virtual machines for different tests. Notice that, for windows 10, the duration for RobView UAS setting test is not available because RobView is only designed for windows 7. If it is run in other operating systems, CULT cannot find "start menu" due to accessibility of GUI object is not supported in the OS except for windows 7 . In general, it takes longer to complete tests in windows 10 than windows 7, which is due to the slow response in resources-consuming and high latency Windows 10. Furthermore, the more events involved in test, the longer the duration is. However, there is no linear relation between them.

It is worth mentioning that the duration for a test in a certain virtual machine may correlate significantly with the resource-usage situation in host operating system, if the virtual machine is installed in the same machine with host machine like in this thesis. Taking the principle of hypervisor into account, A virtual machine is regarded as a process in host OS. Consequently, the busier a host machine is, the slower a virtual machine will be. Therefore, to make duration data more representative, an average of duration is computed to provide a more concise description. It is very clear from the average value that installation test takes longer than uninstallation test and applications with smaller size finish testing in shorter time.

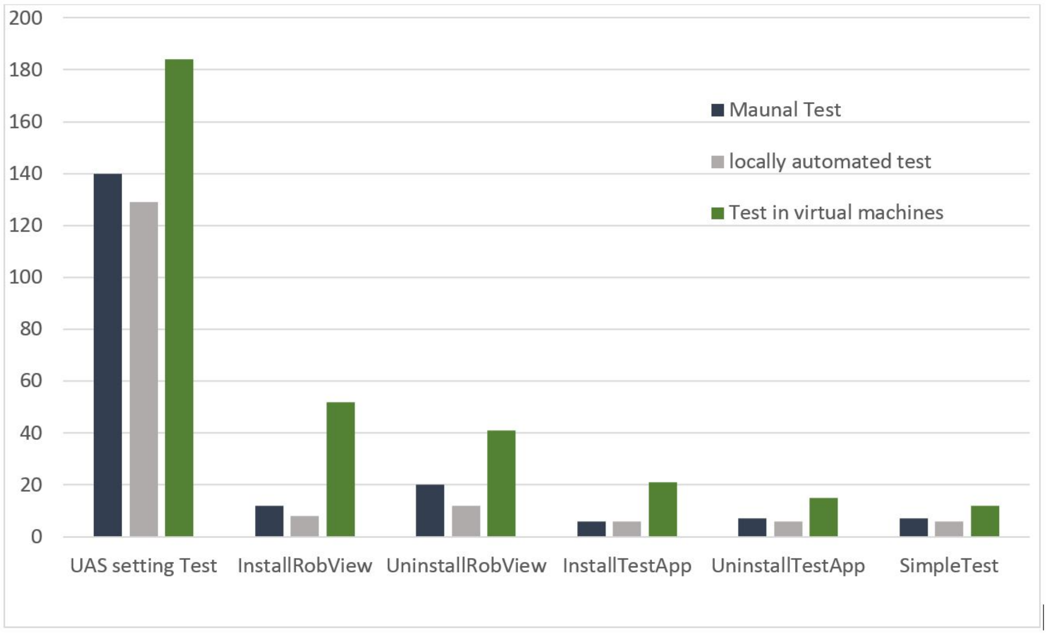


Figure 4.5.1: Test duration for different set of tests in different manner

Figure 4.5.1 demonstrates the differences of test performance in manual test in host machine, automated test in host machine and automated test in virtual machines (with second as time unite). Obviously, test in virtual machines always takes longer time to finish compared to manual test and local automated test. The reason behind the fact is that time taken to start execution (i.e. start test agent MSTest.exe) is by default included in completion time, while there is no starting test agent time taken into account performing automated test from script in host machine. The greater test duration in virtual machines is also partially because of slow response guest operating system in virtual machines, especially operating system like windows 10, which has a high requirement on host machine hardware resources. Considering the reasons described above, although duration in virtual machines is greater than ones in other two ways, it can still be deemed as an efficient approach judging by the benefits it brought.

4.5.1 Results of Executing Multiple Tests Concurrently

So far, no matter manual test or automated test, all test are executed individually, which means at any point of time, only one test is allowed to be performed due to the non-cumulative property of testing[27]. Apparently, if in some way, multiple tests can be executed simultaneously, the efficiency will be enhanced dramatically. No solution is proposed so far to solve it in a host machine. However, with virtual machines, the problem can be addressed properly. Non-cumulative property also applies to virtual machines, which implies that two or more tests execution at the same time in a single virtual machine is not possible. It is conceivable that running multiple virtual machines should be able to solve the problem.

In theory, any number of virtual machines can be manipulated in parallel. Nevertheless, it typically has a strict requirement on the host machine hardware if many virtual machines are running simultaneously. Otherwise guest OS will response extremely slow. As a consequence, compromise between efficiency and cost should be made. In this thesis, a host machine with Intel Core-i7 4510U CPU 2.0GHz processor and 8 G installed memory is used to implementation. if 3 virtual machines with Windows 10 installed, are set to run simultaneously, both host and virtual machines are extremely slow. Therefore, in this thesis , only 2 virtual machines are set to run concurrently, and 3 groups with 2 virtual machines in each are formed.

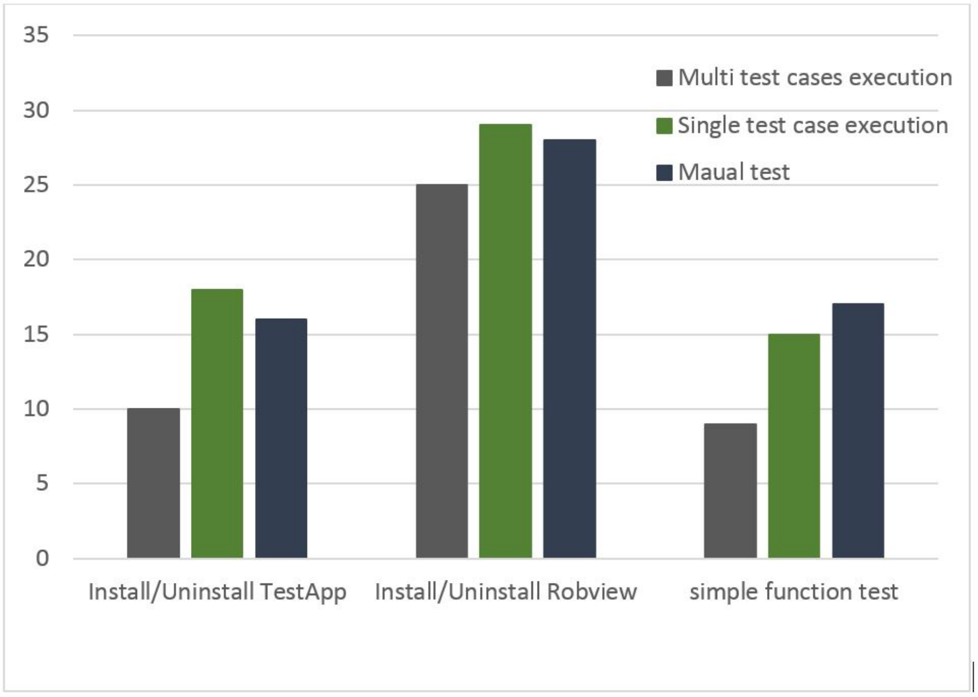


Figure 4.5.2: Round trip time for different sets of GUI testing with different solutions

Figure 4.5.2 illustrates the different round trip time for manual test execution,single test execution and multiple tests execution respectively (with minute as time unit) . Round trip time is the duration between committing a source code change and the reporting test results reported back to developer. Notice that, the round trip time includes time taken to automate virtual machines for both single execution and multiple execution.(i.e. power on a virtual machine –run tests –copy results back to host machine – power off a virtual machine etc). We can tell from the graph that single test execution has a similar performance with the manual test, as explained in above section. The reason is that test agent starting time is included in round trip time in single execution. However, It is clear from the graph that multiple tests execution is always more efficient than single tests execution and manual test. Specifically, for installation and uninstallation test of TestApp, time taken to automate tests in all 6 virtual machines one by one is around 18 minutes, while the round trip time for executing 2 tests simultaneously for 3 times (i.e. run test in all 6 virtual machines) is around 10 min, the performance improved by around 40%. While compared to manual test, performance improved by 11%. Completion time for installation and uninstallation test of RobView in single and multiple execution is 28min and 25min. performance enhances by around 11%. While for simple function test, there is around 40% performance boosted.

Although, the figure above does show some advantages of multiple test case executing over single test case execution and manual test. In theory, the round trip time for multiple test case execution should be twice less than single execution in this case. However, in fact, multiple test case execution is not efficient as expected, the reason behind is that running multiple virtual machines at the same time make both host OS and guest OSes extremely slow-responding, which increases the round trip time greatly. Considering the circumstance that virtual machines are stored in a remote server or cloud machine, there will be no load introduced, and the number of virtual machines that run simultaneously can be up to 10 or even more, the efficiency of testing will improve greatly.

From the analysis of figure 4.5.2, efficiency of simultaneous multiple tests execution is considerable. Apparently, the more tests executed in parallel, the less round trip time is. However, if the host machine is not powerful enough, more virtual machines will make not only virtual machines but also host machine extremely slow, which results in long round trip time. As a result, trade-off between cost and efficiency should be taken into account.

4.6 Failure Analysis

Coded UI test is an impactful tool for GUI testing in various aspects as results shown in previous section. It is easy to use, efficient to maintenance and can fix codes. Nevertheless, failure is prone to appearing in testing process, some of which is due to the disadvantages of CUIT, while part of which is resulted from application under test. It is worth of analyzing the failure not only for fixing the error in application under test, but also for gaining a further understanding of CUIT. Form the work flow of GUI testing described above, failures can be classified into three classes, searching GUI objects failure, performing action failure, and lastly, verification of states failure. For the sake of analyzing various failures in an explicit and concise manner, failures will be demonstrated in category fashion.

4.6.1 Searching GUI Object Failure

Searching failure is the most common one in testing process, however the reasons behind may be various a lot mainly depending on specific circumstance.

- Problem: in installation/uninstallation test, the last step is generally click finish button, however in some cases, tests fail with error "cannot find finish button".

  Reason: Due to the searching mechanism of CUIT, GUI objects should be visible before any action is performed. In this problem, finish button is not visible before time out, because install/uninstall progress bar always takes some time to finish.

  Solution: Add WaitControlExists () method before click finish button in test script, which suspends current thread until target control (GUI object) is visible. Hence existence of GUI control can be guaranteed.

- Problem: test script generated in English version (e.g. windows 7 English), does not perform click Windows system close button, which is built in for every application. Test fails with cannot find close button

  Reason: for different operating system language version, the name for system button may be various in different language system, for example close button in English operating system is "button close", while in Norwegian it is "knapp lukk". Since the GUI control is found according to it's name, it is reasonable that close button cannot be detected in other language version.

  Solution: instead of hardcoding clicking system button, exiting application from function provided by application itself like cancel button.

- Problem: in installation test, by default, when an application is installed, user account control dialog will appear warning that modification on the machine will be done. Generally, clicking OK button will make application installation step into further procedure. However, for CUIT, it is possible to capture the User Account Control dialog.

  Reason: the event of UAC is semantic, which represents that it will not put into event queue of GUI application. Therefore, CUIT will not capture the UAC dialog.

Solution: modifying the setting of User Account Control to ignore the change of modification on machine. As a consequence, the UAC dialog will not appear when new application is about to install.

- Problem: As mentioned before, start menu of RobView is not visible in Windows 8 and Window 10.

  Reason: the problem is probably caused by accessibility support of UIA element (i.e. start menu) is not set for windows 8 and windows 10 considering the application is designed for windows 7 only.

  Solution: modifying the RobView application source code to enable the accessibility support. However, the solution is not verified to be effective yet.

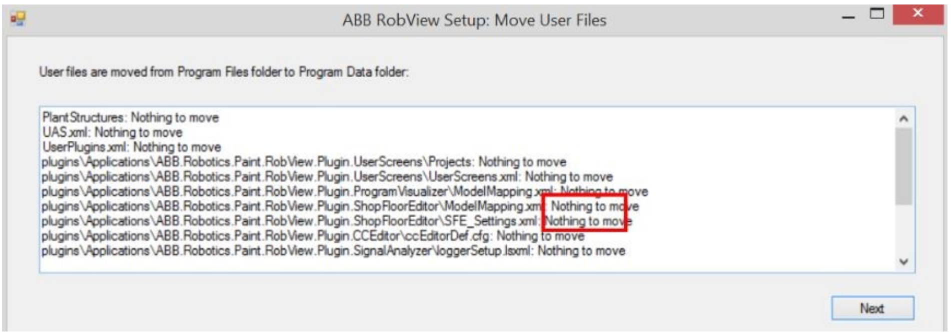
4.6.2 Action failures

Sometimes test cases fail with error like "certain action cannot perform on the control", like the problem shown below.

- Problem : as illustrated in figure 4.6.1, the highlighted rectangle represents the Next button found by the test script, and when click Next button is executed, the test will fail with error "cannot perform click on next button"

  Reason: the problem may caused by the existence of cache control, the cache control may not be deleted completely when the application is developed.

  Solution: change the application source code to remove cache control, or in the case of RobView, removing the Move User files dialog directly.



User files are moved from Program Files folder to Program Data folder:

```
PlantStructures: Nothing to move
UAS.xml: Nothing to move
UserPlugins.xml: Nothing to move
plugins\Applications\ABB.Robotics.Paint.RobView.Plugin.UserScreens\Projects: Nothing to move
plugins\Applications\ABB.Robotics.Paint.RobView.Plugin.UserScreens\UserScreens.xml: Nothing to move
plugins\Applications\ABB.Robotics.Paint.RobView.Plugin.ProgramVisualizer\ModelMapping.xml: Nothing to move
plugins\Applications\ABB.Robotics.Paint.RobView.Plugin.ShopFloorEditor\ModelMapping.xml: Nothing to move
plugins\Applications\ABB.Robotics.Paint.RobView.Plugin.ShopFloorEditor\SFE_Settings.xml: Nothing to move
plugins\Applications\ABB.Robotics.Paint.RobView.Plugin.CCEditor\ccEditorDef.cfg: Nothing to move
plugins\Applications\ABB.Robotics.Paint.RobView.Plugin.SignalAnalyzer\loggerSetup.lsxml: Nothing to move
```

Figure 4.6.1: Action failure in Move User File application

4.6.3 Application Error

- Problem: in the uninstallation test for TestApp, although the test can pass, which means files in certain directory is deleted, it does not actually uninstall the application, and can still find in control panel.

  Reason: probably, the problem is due to the property of setup file, in which duplicated installation is enabled, which means more than two same application installation is allowed. Uninstallation test removes only one of the installations.

  Solution: fixing the setup file to forbid duplicated installation may solve the problem.

4.6.4 System Problem

- Problem: for tests in virtual machines, when previous tests complete, immediate start of next test sometimes lead to fail, especially when the OS responses slowly.

  Reason: the cause of the problem is evidently due to the slow responding system. Previous test probably has not finished in the background, and two or more test executions simultaneously in a single machine is not allowed according to non-cumulative property.

  Solution: Slowing down the system by adding sleep time before executing next test.

5 Test Cases Scheduling in Virtual Machines

The performance of the solution proposed is proven to be considerable in previous chapter. For the purpose of covering all the test paths and verifying every single function of a software, the number of test cases can be enormous amount as mentioned before, as a result completion time will be extremely great. The importance of decreasing the round-trip time, that is, the duration between committing a source code change and the reporting test results reported back to developer, raises concern of testers. As a consequence, scheduling test cases so that the round trip time is minimal is critical to improve the effectiveness in the software testing process. In this thesis, tests can be distributed to different virtual machines instead of performing all test cases in every virtual machine. In this way, the performance of fully automated GUI testing can be optimized.

5.1 Introduction of Test Cases Scheduling

It is apparent that test case scheduling aims to reduce the round trip time as much as possible. The simplest and most straightforward approach is to perform test cases in as many machines as the number of test cases concurrently. However, it is impractical considering the cost. Taking virtual machines in this thesis as an example, running hundreds of virtual machines simultaneously will slow down the host machine and guest operating systems significantly if the virtual machine is stored on a same host machine. Therefore test cases takes longer time to finish. Obviously , the simplest approach is very costly. As a result, test cases scheduling is one proper and inexpensive approach to achieve minimal round trip time.

5.1.1 Description of Test Cases Scheduling

Test cases scheduling in virtual machines simply means ordering the test cases under perform in multiple virtual machines so that the time taken to finish all tests is minimized to the greatest extent. From a mathematical point of view, considering a set of test cases denoted as *T*, which has elements as $\{t_1, t_2 \cdots t_n\}$. These test cases have corresponding execution duration *D*,it can be expressed as $\{d_1, d_2 \cdots d_n\}$. Furthermore, a set of virtual machines is described as *VM*, which has virtual machines as elements shown with $\{vm_1, vm_2 \cdots vm_m\}$. Noticeably , these virtual machines may differ in various aspect, e.g. operation systems , or the allocated hardware resources from host machine. Therefore, the execution duration for each test case may slightly over-estimated to result in a small variation in different virtual machines. Nevertheless, compared to the duration , the variation is too small, so it is reasonable to ignore the changes.

The purpose of test cases scheduling in virtual machines is to find a function *f*, so that $S = f\ (M, T\ )$. *S* is the targeted new sequence of test cases.The overall test execution time T_e can be minimized with the new sequence. The function *f* assigns test cases to different virtual machines. It is clear that the time taken to find the best sequence set *S* is not negligible. Assume that the time required to find best solution is denoted as $T_{s,}$, and the time taken to finish all tests can be represented as T_e. Therefore, the total time taken in testing process T_t is defined as: T_t= T_s+ T_e. Minimizing T_t is the final goal of test cases scheduling.

However, for test cases scheduling in virtual machines, the following constraints should be strictly forced:

- **No-cumulative scheduling** : at any time , a single virtual machine can only executes one test cases. Two or more test cases performing in the same virtual machine at the same time is not allowed.

- **No-preemptive scheduling**: at any time, a test case running on a virtual machine cannot be interrupted in order to execute another test case instead.

- **Machine independent**: it is hypothesized that the execution time of a test case is irrelevant with any specific virtual machines, even though the virtual machines vary in terms of processors allocated, or memory assigned form the host machine.

The optimization problem can be described by a time-discretized table. As table 5.1.1 shown, the table contains a set of test cases ,duration for each test case and its assignment to given virtual machines.

Table 5.1.1: A simple case for test case scheduling

Test	Duration	Executable on
$\mathbf{t}_1$	10	vm_1, vm_2, vm_3
$\mathbf{t}_2$	3	vm_1, vm_2, vm_3
$\mathbf{t}_3$	5	vm_1, vm_2, vm_3
$\mathbf{t}_4$	6	vm_1, vm_2, vm_3
$\mathbf{t}_5$	14	vm_1, vm_2, vm_3
$\mathbf{t}_6$	15	vm_1, vm_2, vm_3
$\mathbf{t}_7$	11	vm_1, vm_2, vm_3
$\mathbf{t}_8$	20	vm_1, vm_2, vm_3
$\mathbf{t}_9$	18	vm_1, vm_2, vm_3

A small test case scheduling problem is presented as table 5.1.1, let T be the set of test cases and set to $\{t_1, t_2 \cdots t_9\}$, and the duration D here is set to $\{10,3,5,6,14,15,11,20, 18\}$. Note that the time unit is not specified here, it can be minute or second. From reality view, the value can be gained by empirical running test cases. The last column in table 5.1.1 indicates the virtual machines these test cases can execute on. In this example, all tests can be performed in all machines. However, in some cases, certain tests can only be executed on certain virtual machine to test the specified function. But, all virtual machines are identical in this example, so test cases can be performed in all virtual machines available, e.g. vm_1, vm_2, $vm_{3,}$ in the example.
From an algorithm point of view, test cases scheduling can be mapped as "multiple-CPU scheduling" [43] problem, the definition is given as:

Def 5.1.1. *A set $S = \{a_1,a_2, ..., a_n\}$ of n proposed activities that compete to use common resources ,which is a set of CPUs denoted as $\{m_1,m_2,\cdots m_m\}$. Each activity a_i has a start time s_i and a finish time f_i, where $s_i \leq f_i < \infty$. If activity a_i is selected, a_i takes place during the half-open time interval $[s_i, f_i)$. Activities a_i and a_j are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap (i.e., a_i and a_j are compatible if $s_i \geq f_j$ or $s_j \geq f_i$). The target is to find the minimal completion time*

It is clear that "multiple-CPU scheduling" is a NP-complete problem, which means that best solution can not be obtained quickly in polynomial time. Obviously, a naive approach is to use brutal-force, where all possible solutions are executed so that best one can be found out. There is no doubt that brutal force can always find an optimal solution, but taking the time taken into consideration ,especially in cases with a great amount of solutions, it is an expensive approach. Some other methods are proposed by researchers in last decades, including heuristic algorithm [2], the solution emulates the behavior in biogenetics way so that a search approach is constructed. Simulate Anneal Arithmetic(SAA) [19] is another approach, SAA interprets slow cooling as a slow decrease in the probability of accepting worse solutions when it searches the solution space. And neural network algorithm 31 is one of the options as well. In those algorithms, randomness is used to get a faster average running time. Furthermore , there is no evidence shown that these are faster and always produce a best solution. Typically, the compromise between time taken and the optimal extent of final solution, should be made. Alternatively, greedy algorithm is proposed to solve the problem in an efficient way. Noticeably, with greedy algorithm adopted, the solution found is not optimal , but "almost" optimal.

One possible best solution for the simple example is shown in figure 5.1.1 with greedy algorithm adopted. The completion time is : 18 + 11 + 6 = 35

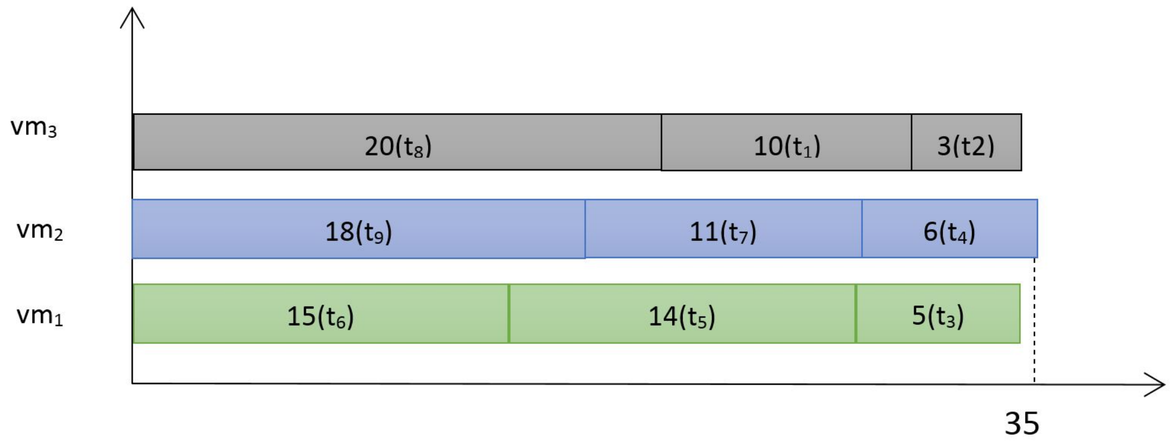


Figure 5.1.1: "Almost" optimal solution for test case scheduling with greedy algorithm

5.1.2 Test Case Scheduling With Greedy Algorithm

From previous analysis, it is clear that greedy algorithm is capable of solving "multiple-CPU scheduling" problem in an effective way. It finds the as optimal as possible solution in a short time. In all the algorithms mentioned, search strategy is always the most critical part to solve problems. Different strategy results in different performance, e.g. search approach in heuristic algorithm is based on the theory in biogenetics. However randomness reduces algorithm's performance to some extent.

While for greedy algorithm, there exists various strategies too. For example searching can start with test cases with shortest time first, or in contrast , always choose tests with longest completion time. It is obvious from figure 5.1.1 that search strategy is based on longest competed time. To make the algorithm clear, mathematical description is employed. For mathematical problems, there is always input and output parameters, in this case, these parameters are illustrated as follows:
Input : As mentioned before, test cases , duration and virtual machines are denoted as T $\{t_1, t_2 \cdots t_n\}$, *VM* $\{vm_1, vm_2 \cdots vm_m\}$, *D* $\{d_1, d_2 \cdots d_n\}$ respectively. For duration , information about the starting execution time s_i and finish execution time f_i can be added to input set . Therefore, each test case can be labeled as stating time , finish time , duration and virtual machines they can be executed on . Among the input parameters, starting time and finishing time are variables to be assigned values. m_i is a variable indicating the virtual machine a certain test case will be assigned to. The value lies in a finite domain $[1,2,\cdots m]$ adn m is the number of virtual machines. For the simple example in table 5.1.1, the value of m_i can be set as [1,2,3]. In general, a test case can be expressed as $\{s_i, f_i, d_i, vm_i\}$
Output : $S\{s_1, s_2 s_n\}$, is the starting time set for corresponding test cases. $M\{m_1,m_2 \cdots m_n\}$ is the numeric value indicating corresponding test case assignment. The greedy algorithm can be illustrated briefly as below:

Algorithm 1 Greedy algorithm for test case scheduling

```
1:  Starting time : $s_i \leftarrow 0$
2:  Finish time : $f_i \leftarrow 0$
3:  Sort :D
4:  for each element $i$ in $D$ do
5:      Find : Maximun value : $d_m$
6:      Find : Virtual Machine : $j$
7:      with shortest finishing time : $f_j$
8:      $s_i \leftarrow m_j$
9:      $m_i \leftarrow j$
10: end for
11: Return S,M
```

With greedy algorithm descripted above, the output of test scheduling for the example in table 5.1.1 is shown as :

S ={20,30,29,29,15,0,18,0,0} , M={3,3,1,2,1,12,3,2}; total completion time is 35.

In general, greedy algorithm in test scheduling aims at assigning test cases with longest- running time to the virtual machines that finishes current job firstly. As mentioned above , greedy algorithm is unable to guarantee the final solution to be optimal. But the algorithm ensures that it is approximate optimal. Therefore we can say that greedy algorithm is still effective, especially considering the cost in both space and time, which is *O(1)* and *O*(n*logn) respectively. For small or medium scale problems, the cost can be ignorable, which makes it suitable to solve the problem of test cases scheduling in virtual machines.

5.1.3 Theoretic Implementation

In this section, applying test case scheduling in multiple virtual machines will be presented. In short , implementing scheduling strategy discussed above in a realistic set of tests, which are designed to verify the function “RobView” in ABB. Round trip time of scheduling will be given.

According to the information from testers in ABB, duration for an individual test to execute varies from 1 minute to 5 minutes (depending on the test steps). In order to cover all functions in “RobView”, there are 67 automated test cases in total. To execute the entire suite, it takes around 1.30 Hours. In this case, for each test cases, the average duration is 1.3 minutes, which means most of the test cases take around 1 minute to complete. However the detailed duration for each test case is not provided, a reasonable assumption is made in this thesis to illustrate the test case scheduling solution. The duration for each test case is assumed as listed in the table.

In table 5.1.2, the corresponding number of test cases is given for each test case duration . The value of duration for each test case are set to be integer so as to simplify the assumption. Being consistent to the information from testers, the value of duration lies between 1 minute to 5 minutes. The overall time is 90 minutes, and the overall number of test cases is 67. The sum of time and number of test cases comfort to the information from tester, therefore, we can say that the assumption is reasonable.

Table 5.1.2: Assumed duration of each test case in GUI testing for RobView

Duration (minute)	Number of test case	Sum of duration (minute)
5	1	5
4	2	8
3	3	9
2	7	14
1	54	54
	67 (Sum)	**90(Sum)**

The assumed duration for each test case is listed in the above table. According to the test scheduling solution (with greedy algorithm adopted) , the round trip time is demonstrated as the following graph. Notice that, in previous work, only 2 virtual machines are manipulated concurrently because the host machine where virtual machine stored is not powerful enough. More than 2 virtual machines manipulation at the same time will make host and guest OS extremely slow-responding . However, considering some situation that host machine is power enough to run 10 or more virtual machines . In order to illustrate the efficiency of test case scheduling, graph 5.1.2 below shows the round trip time with different number of virtual machine executed simultaneously.

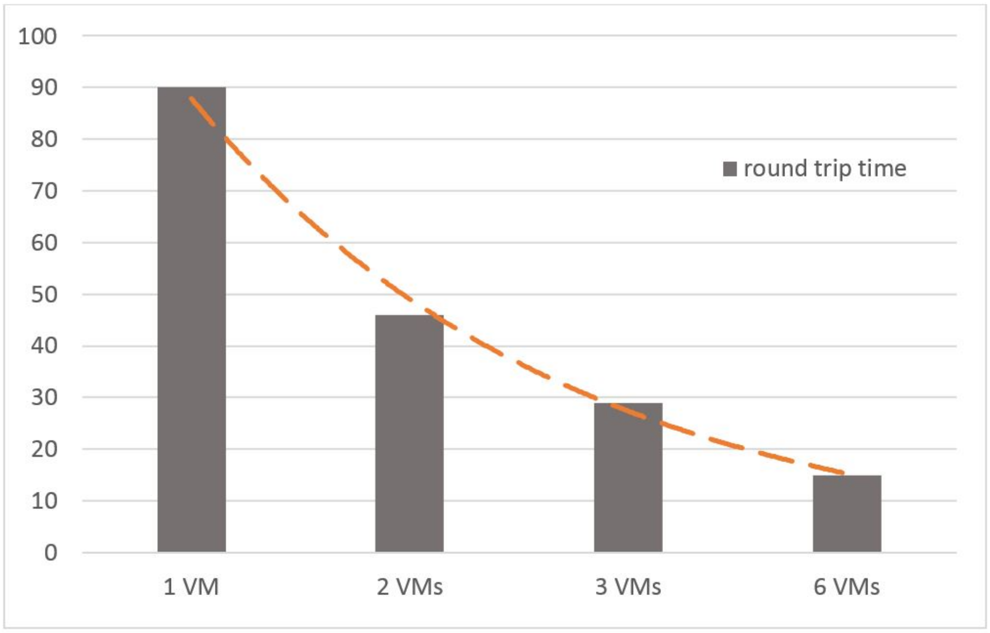


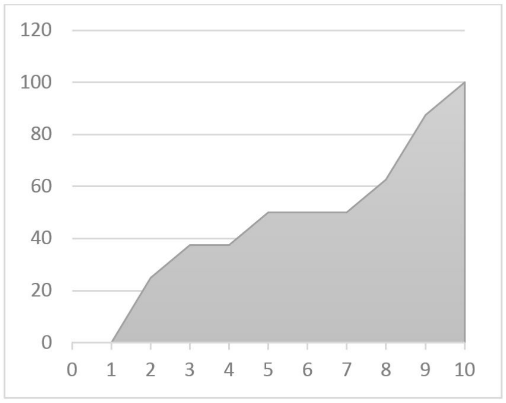
Figure 5.1.2: Round trip time for tests with different number of simultaneous manipulation virtual machines

As show in figure 5.1.2, the more virtual machines are manipulated, the less round trip time is. Specifically, when no test case scheduling involved, the overall time to perform all test cases is 90 minutes as provided from tester. While it takes 46 minutes to complete all testing when 2 virtual machines are executed simultaneously with greedy test case scheduling adopted. Compared with no test case scheduling, the performance improves by around 48 %. The corresponding round trip times for 3 and 4 virtual machines simultaneous manipulation are 29 minutes and 15 minutes respectively. As trend line indicates, the round trip time decreased dramatically. Thus the efficiency is enhanced to a great extent. The industrial case studied above illustrates clearly that test case scheduling can reduce the round trip time and increase testing efficiency significantly, although it is a theoretic evaluation. However, for testing, the round trip time is just one of the parameter which affects the performance.

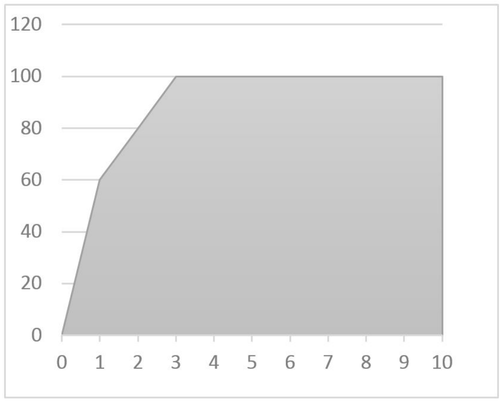
As is known universally, the goal of software testing is to find potential faults, and therefore the detecting faults ability of test cases should be evaluated. The following part discusses how to improve testing performance regarding faults detecting ability of test cases.

5.2 Test cases prioritization

Test cases prioritization is not a novel concept, literally, it means developers may prioritize test cases so that more important ones, according to a certain criteria, will be executed earlier in testing process. One of the potential and common goal is to increase the ability of test cases to detect faults earlier. Rate of fault detection is adopted to indicate the ability, it indicates how fast a test suit detects faults during the testing process. Earlier faults detection bring huge benefits to testing process, an improved rate of detection can provided earlier feedback on the application under test , and therefore, earlier debugging is possible. Moreover, if application under test is suspended due to some reasons, those tests, which has the greatest fault detection ability, will be executed before application is halted. Figure 5.1.2, illustrates briefly the differences of testing with and without prioritization [34].



(a) T_1-T_2-T_3-T_4-T_5-T_6-T_7-T_8-T_9-T_{10}



(b) T_9-T_{10}-T_5-T_2-T_3-T_4-T_6-T_7-T_8-T_1

Figure 5.2.1: Rate of fault detections with two different test cases sequence

Considering 10 test cases T1...T10, and 8 faults contained in the application under test. All faults can be revealed by those 10 test cases. Two different test case execution sequences are proposed to illustrate the effect of prioritization. Namely, test case order 1 : T_1-T_2-T_3-T_4-T_5-T_6-T_7-T_8-T_9-T_{10}, order 2 : T_9-T_{10}-T_5-T_2-T_3-T_4-T_6-T_7-T_8-T_1. In figure 5.2.1 (a) and (b), the area under the curve represents the weighted percentage of faults revealed with the corresponding test cases used.

For the first test order, T1 is executed first, and no faults are detected, following with performing T2 and 2 faults are revealed. Therefore, the corresponding percentage of faults detected is 25% (2 out of 8 in total). After running test case T3, one more fault is detected, thus the detected faults percentage increases up to 37.5 % (3 out of 8). All faults are revealed when all test cases complete. In contrast, figure 2 (b) is apparently faster in detecting faults. Specifically, when first test case T9 is executed, 5 faults is discovered, making the detected faults percentage to be 62.5%. Being different form test order 1, test order 2 reveals all faults when the third test cases T5 is executed . Conclusion can be easily drawn that faults can be detected much faster with test case prioritization , thus round trip time can decrease since not all test cases are necessary for revealing all faults

5.2.1 Description of Test Case Prioritization Problem

In previous work [34], Rothermel et al, defines the test cases prioritization problem, and several solutions are also illustrated. The test case prioritization problem is defined as follow:

Def 5.2.1. *Given : T, a set of test case. PT, the set of permutations of T. f, a function from PT to real number. Problem : Find T' $(T' \in PT)$, such that for all $T''(T'' \in PT, T'! = T'')$, $f(T') > f(T'')$*

Here, PT represents the set of all possible prioritizations, which is essentially the sequence of test case execution. And *f* is a function applied to any such ordering. The selection of *f* may differ according to the desired performance, it can be code coverage, rate of fault detection or fault likelihood. Many previous works have proposed various solutions to target certain goal. To illustrate, Rothermel et al [34] proposed strategy based on both function and statement levels. And Lee et al [23] reduce test suits by adopting test cases that provide coverage of requirements. Offutt et al.[1] employees coverage criteria to decrease prioritization test cases. However, none of the approaches mentioned above can be applied to GUI testing, which is targeting at event-driven application.

5.2.2 Test Case Prioritization For GUI testing

GUI application, as described above, is event-driven, which means that the input is a sequence of event and the change of state caused by each event should be verified in GUI testing. Therefore, a test case prioritization solution to event-driven system is needed. A.M.Memon et al 30 proposed a new approach for GUI test case prioritization. In their work some new concepts are presented to illustrate the prioritization problem as shown in the following:

Parameter: is the GUI application widgets, e.g a checkbox. Parameters can be deemed as the component of GUI application.

Value: is the setting for parameters, e.g. if the checkbox is clicked, then the value for the checkbox parameter is true, otherwise it is false.

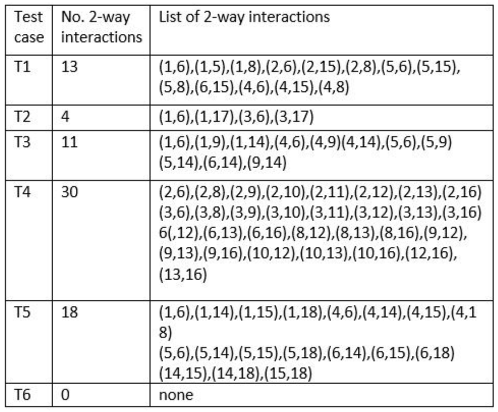
Parameter-value: is the pair of parameter and corresponding value, e.g. in previous case, a parameter-value can be <checkbox, true>.

Action: occurs when users set values to one or more parameters on a windows before visiting next window.

Different test case prioritization solutions are studied in their work, including parameter-value interaction-based criteria, which is based on the interactions between multiple parameter-values. Count-based criteria, solution is built according to the number of actions or windows a test case covers. And frequency-based criteria, top priority is given by the number of most frequently visited windows a test case covers.

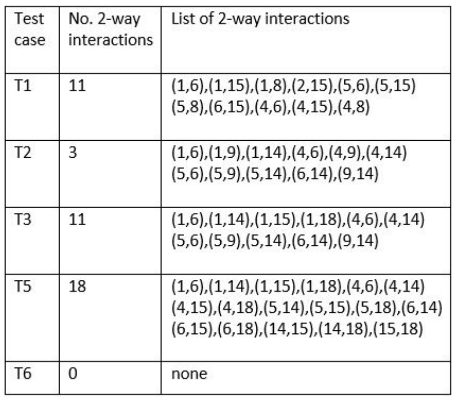
Based on the experiments running on various solutions, the best solution to GUI tests prioritization is parameter-value interaction criteria. Specifically, it is 2-way interaction solution. 2-way criteria is a one of the solution proposed in the category of parameter-value interaction. It will be carefully demonstrated in the following section.

The 2-way criteria is built on the assumption that faults are easily exposed when interactions of parameters set values on different windows. Therefore, a next test case is chosen to maximize the number of 2-way parameter-value interactions between windows. A simple example [30] illustrating 2-way solution is given in the table below:



Test case	No. 2-way interactions	List of 2-way interactions
T1	13	(1,6),(1,5),(1,8),(2,6),(2,15),(2,8),(5,6),(5,15), (5,8),(6,15),(4,6),(4,15),(4,8)
T2	4	(1,6),(1,17),(3,6),(3,17)
T3	11	(1,6),(1,9),(1,14),(4,6),(4,9)(4,14),(5,6),(5,9) (5,14),(6,14),(9,14)
T4	30	(2,6),(2,8),(2,9),(2,10),(2,11),(2,12),(2,13),(2,16) (3,6),(3,8),(3,9),(3,10),(3,11),(3,12),(3,13),(3,16) 6(,12),(6,13),(6,16),(8,12),(8,13),(8,16),(9,12), (9,13),(9,16),(10,12),(10,13),(10,16),(12,16), (13,16)
T5	18	(1,6),(1,14),(1,15),(1,18),(4,6),(4,14),(4,15),(4,1 8) (5,6),(5,14),(5,15),(5,18),(6,14),(6,15),(6,18) (14,15),(14,18),(15,18)
T6	0	none

(a) All 2 way interactions



Test case	No. 2-way interactions	List of 2-way interactions
T1	11	(1,6),(1,15),(1,8),(2,15),(5,6),(5,15) (5,8),(6,15),(4,6),(4,15),(4,8)
T2	3	(1,6),(1,9),(1,14),(4,6),(4,9),(4,14) (5,6),(5,9),(5,14),(6,14),(9,14)
T3	11	(1,6),(1,14),(1,15),(1,18),(4,6),(4,14) (5,6),(5,9),(5,14),(6,14),(9,14)
T5	18	(1,6),(1,14),(1,15),(1,18),(4,6),(4,14) (4,15),(4,18),(5,14),(5,15),(5,18),(6,14) (6,15),(6,18),(14,15),(14,18),(15,18)
T6	0	none

(b) 2-way interactions after T4 is selected

Figure 5.2.2: A simple case to illustrate 2-way interactions criteria

Table 5.2.2 lists 2-way interactions of parameter-values, the numeric value pairs in the interaction indicate the parameter-value on different windows. For example 1-5 represents the possible value of parameters in windows 1, 6-11 indicates the possible value of parameters in windows 2 and so forth. Essentially, the interaction implies the communication of parameters on different windows. According to 2-way criteria, T4 will be firstly selected due to the most interactions it includes. After T4 is selected, the table will change due to the interactions in T4 will be excluded in rest of test cases because they are already excluded. Table 5.2.2 (b) shows the remaining interactions after T4 is selected. Apparently, T5 is the next test case to perform because it contains most interactions untested previously in T4. The final sequence is T4-T5-T3-T1-T2.

Based on the experiments running on all solutions mentioned above, 2-way detects 100% of faults in the fewest test cases. Average proportion of test cases used to reveal all faults is 54%, which means that with only around half of all test cases, all faults can be detected. Therefore, another half of test cases can be cut off. Obviously the 2-way solution improved the performance significantly, it reduces the number of test cases and ensures all faults to be detected. The superior performance of 2-way interaction solution makes it proper for the prioritized test cases scheduling, which will be carefully presented in the following section.

5.3 Prioritized Test Cases Scheduling in Multiple Virtual Machines

As discussed above, test case scheduling ensures round trip time to be minimal, while test case prioritization guarantees that all faults can be revealed as quick as possible. If testers desire to achieve both minimal round trip time and high rate of fault detection, a novel approach should be proposed to combine two properties of a test case together.

The problem of prioritized test cases scheduling in multiple machines can be defined as: given a set of test cases with two properties, completion time and priority number respectively. A function *f* should be found so that the overall round trip time and rate of fault detection is both optimal. It is very obvious that the problem is NP-complete problem. No optimal solution can be find in polynomial time, and there is few researches propose optimal solutions for this problem.

It is reasonable to taking the problem as ”rectangle packing”[22] problem, which is a problem targeting at minimizing the area of rectangle. In detail, there is a bunch of rectangle pieces, and the goal is to arrange them in a rectangle surface so that they don't overlap while keeping the rectangle area as small as possible. In this case, test case with different duration and priority can be deemed as rectangle with different dimensions. For optimal solutions, pre-selecting ”interesting” place to locate next rectangle is the most important part in the problem. Various solutions have been built, including [11] ,it implements the bottom-left heuristic for two-diemnsion bin-packing, the time complexity is $O(N^2)$. And in paper[28], the solution is made based on limitation on the rectangles' coordinates and bounding box dimensions to the set of subset sums of the rectangles' dimensions. However, since the optimization problem is NP-hard, while the problem of deciding whether a set of rectangles which can be packed in a given bounding box is NP-complete. The solution to the problem is quiet complicated, and there is no solution recognized as the best one so far. Considering the problem in this thesis, a complicated algorithm may increase takes time to find best solution, therefore, the round trip time may increase.

Essentially, the basic idea to address the issue is that ordering test cases execution sequence with consideration both it's duration and priority. Hence a criteria of deciding the most "important" test case should be made. The method to weight different test cases is, therefore, critical in this problem.

5.3.1 Analytic Hierarchy Process Solution with Small Example

In order to illustrate the prioritized test cases scheduling solution, a small example is given below. 6 test cases with different time and priority are listed in the table.

Table 5.3.1: Test cases with different priority and duration

Test case	Duration	Priority
T_1	2	6 (highest priority)
T_2	1	2
T_3	5	4
T_4	3	1 (lowest priority)
T_5	6	3
T_6	4	5

In the example in table 5.3.1, 6 test cases with corresponding duration and priority are shown. Note that, discrete value is used here to demonstrate the difference in duration or time, the time unit can be minute or hour. As for priority, the larger a numeric number is, the more important a test case is. Therefore, from the table we can tell that test caste T1 has the highest priority, which is denoted as 6. By recalling the algorithms we discussed before, for test case scheduling with multiple virtual machines, test cases with longest duration will be always selected at first place, while in test case prioritization approach, test cases with highest priority will be chosen to perform firstly. However, for the prioritized test case scheduling with multiple virtual machines, a test case may has a high priority and short duration, like T1 in the example, should it be chosen to perform therefore the final round trip time is minimal? Fortunately, analytic hierarchy process provided a "best decision" approach to solve the issue.

Analytic hierarchy process (AHP) [33] is a structured technique for organizing and analyzing complex decisions based on mathematics and psychology. It was developed by Thomas L. Saaty in the 1970s and has been extensively studied and refined since then. APH has a great range of use in various aspects, where best decision is desired.

Instead of prescribing a "correct" decision, AHP achieves final "optimal decision" that best suiting users' goal and their understanding of the problem. A comprehensive and rational model for structuring a problem is the basis of AHP. Moreover, AHP provides an effective way to represent and quantify the elements, and combine these elements with final goal, also evaluates alternative solutions.

To make a decision in an organized way, solution can be decomposed into the following steps: [33].

1. Define the problem and determine the type of knowledge sought.
2. Structure the decision hierarchy from the top with the goal of the decision, then the objectives from a broad perspective, through the intermediate levels (criteria on which subsequent elements depend) to the lowest level (which usually is a set of the alternatives).
3. Construct a set of pairwise comparison matrices. Each element in an upper level is used to compare the elements in the level immediately below with respect to it.
4. Use the priorities obtained from the comparisons to weigh the priorities in the level immediately below. Do this for every element. Then for each element in the level below add its weighed values and obtain its overall or global priority. Continue this process of weighing and adding until the final priorities of the alternatives in the bottom most level are obtained.

According to the procedures defined above, the following steps are executed in order:

Constructing hierarchy model

One of the most important concept in AHP is hierarchy model, problems can be analyzed in a comprehensive way with the model. Typically, problems can be mapped into 3 layer as shown in 5.3.1, goal layer, criteria layer and alternative layer respectively. As depicted below, the goal layer is the target of scheduling, which is to minimize round trip time and improve rate of fault detection. There are two parameters in the second layer, namely duration and priority, which will affect the final goal. In the third layer, there are 6 alternatives, each has unique duration and priority value.

Constructing judgement matrix

According to AHP ,the weight of two alternatives over a criteria can be analyzed by comparing two elements, e.g. comparing T_1 and T_2 in terms of duration, T_1 is more important. Therefore, judgment matrix is generated by comparison every two alternatives over a criteria. In order to illustrate the weight of two alternatives in a precise way, 1-9 absolute number is introduced to denote how better

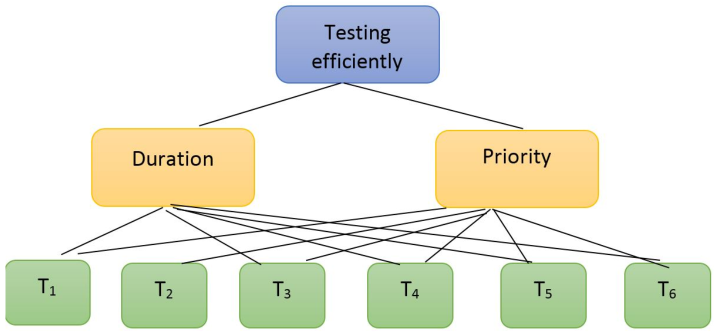


Figure 5.3.1: AHP layers for solving prioritized test case scheduling problem

an alternative is over the others. For example, the duration time of T_5 is 6, while the value for T_2 is 1 , thus, T_5 is extremely better than T_2, so D_{52} will be labeled as 9, while D_{25} is denoted as 1/9. Table 5.3.2 shows the detailed information about how judgment matrix is built, the rule applied for both duration and priority criteria. In addition to constructing the judgment matrix between criteria layer and alternative layer, the weights of criteria over goal layer is critical too. In this case, the weights of duration and priority should be decided according to their importance for achieving the goal . For the simple example, it is assumed that duration and priority is equally important, which means weight put on each criteria is both 0.5, shown as $w_1\{0.5, 0.5\}$

Table 5.3.2: Criteria for building judgement matrix

D_i- D_j or P_i-P_j	D_{ij} or P_{ij}
1 or -1	2 or 1/2 (T_i is a little better/worse than T_j)
2 or -2	3 or 1/3 (T_i is somewhat better/worse than T_j)
3 or -3	5 or 1/5 (T_i is obviously better/worse than T_j)
4 or -4	7 or 1/7 (T_i is strongly better/worse than T_j)
5 or -5	9 or 1/9 (T_i is extremely better/worse than T_j)

Consequently, judgment matrices over duration and priority denoted as D and P respectively are demonstrated as below:

$$D=\begin{bmatrix} 1 & 2 & 1/5 & 1/2 & 1/7 & 1/3 \\ 1/2 & 1 & 1/7 & 1/3 & 1/9 & 1/5 \\ 5 & 7 & 1 & 3 & 1/2 & 2 \\ 2 & 3 & 1/3 & 1 & 1/5 & 1/2 \\ 7 & 9 & 2 & 5 & 1 & 3 \\ 3 & 5 & 1/2 & 2 & 1/3 & 1 \end{bmatrix}, P=\begin{bmatrix} 1 & 7 & 3 & 9 & 5 & 2 \\ 1/7 & 1 & 1/3 & 2 & 1/2 & 1/5 \\ 1/3 & 3 & 1 & 5 & 2 & 1/2 \\ 1/9 & 1/2 & 1/5 & 1 & 1/3 & 1/7 \\ 1/5 & 2 & 1/2 & 3 & 1 & 1/3 \\ 1/2 & 5 & 2 & 7 & 3 & 1 \end{bmatrix}$$

Validation consistency of judgement matrix

With the judgement matrix , it is still rash to make the final decision, because the quality of the matrix should be evaluate so that rationality can be verified. In order to evaluate the consistency of the matrix, consistency ration is utilized As defined as below:

$$CR = \frac{CI}{RI} \tag{5.1}$$

where, CI is the consistency index , which is formulated as :

$$CI = \frac{\lambda_{max} - n}{n - 1} \tag{5.2}$$

λ_{max} is the maximal eigenvalue of matrix .

RI is the random index, which is a constant value for the matrix with a fixed size of n. The value of CI cannot exceed 0.1, otherwise the matrix is considered as inconsistent. In this simple example, it is not difficult to get the value of λ_{max}, which is 6.058 for both matrix, and RI= 1.24, when n=6, therefore:

$$\text{CI}= \tfrac{\lambda_{max}-n}{n-1} = 0.016$$

$$\text{CI}= 0.013 < 0.1;$$

The consistency of both matrix is proven,therefore, they can be used to make final decision. In addition to the judgment matrix, the eigenvector corresponding to the maximal eigenvalue, is utilized to represent the weight for alternatives over certain criteria. In the simple example, eigenvector for duration matrix D can be computed as:

$$W_d = \{0.055, 0.035, 0.26, 0.09, 0.41, 0.15\}$$

while for priority matrix P, corresponding eigenvector is:

$$W_p = \{0.41, 0.055, 0.15, 0.035, 0.09, 0.26\}$$

From the eigenvectors, we can tell that for duration parameter,T1 with weight of 0.055 is less important than T5 with weight of 0.41. Same applies for priority eigenvector.

Computation weight of alternatives over goal

The final step of AHP is to evaluate how important of every single alternative over the goal, i.e. evaluating test cases to find out which one is the most important to achieve minimal round trip time with high rate of faults detection in this thesis. Since alternatives' weights over criteria can be computed with AHP rules, and weights of criteria over final goal can be decided by empirical experience or reference in relevant research. However, in this thesis, it is assumed. W_{ci} is adopted to denote the weights of alternatives over criteria, and $W_{1,}$ represents the weight of criteria over final goal. Therefore, the final weights W of alternatives over final goal can be formulated as :

$$W = \sum_{i=1}^{c.legth} w_1 * w_{ci} \tag{5.3}$$

For the simple example the final test cases' weight is :

$$W = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} * \begin{bmatrix} 0.055 \\ 0.035 \\ 0.26 \\ 0.09 \\ 0.41 \\ 0.15 \end{bmatrix} + \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} * \begin{bmatrix} 0.41 \\ 0.055 \\ 0.15 \\ 0.035 \\ 0.09 \\ 0.26 \end{bmatrix}$$

$$W = \{0.23, 0.04, 0.21, 0.06, 0.25, 0.21\}$$

5.3.2 Evaluation of Prioritized Test Case Scheduling in Virtual Machines

With help of AHP, the weights of each test cases over the targeting goal is achieved. By default, the higher a test case weight is, the more important it is for accomplishing the goal. For the simple example, the final weight can be calculated: $W = \{0.23, 0.04, 0.21, 0.06, 0.25, 0.21\}$ as shown in previous section. Among all, w_5 is maximal, and therefore T5 with duration of 6 and priority of 3 is the most important. T1 with duration of 2 and priority of 6 is followed. T_3 and T_6 have an identical weight, which is 0.21, In this situation, test case with longer duration is more important according the results.

The optimized test case execution sequence, consequently, can be generated. For the small example, the sequence is T_5—T_1—T_3—T_6—T_4—$T_{2,}$ .According to greedy algorithm , selector always chooses the most significant test cases first in testing process, and same with test case scheduling.

Virtual machines with the shortest finishing time is always selected to run test case first. Therefore, the solution for the prioritized test cases scheduling for multiple virtual machines can be shown as figure 5.3.2 (assume that only two virtual machines available for 6 test cases).

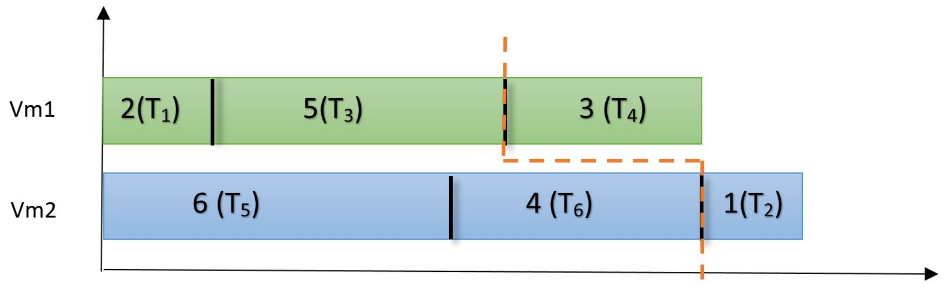


Figure 5.3.2: Solution for prioritized test cases scheduling with AHP

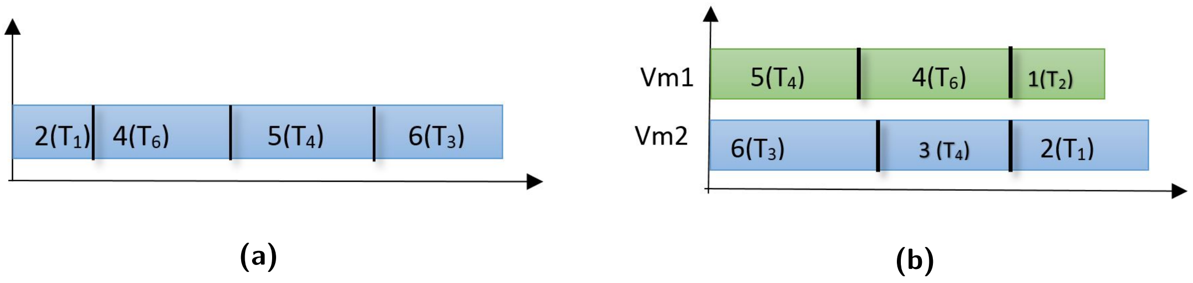


Figure 5.3.3: solution with only test prioritization and only test duration scheduling

The round trip time is 11 time unit for only test case scheduling in two virtual machines as shown in figure 5.3.3(b). Considering test cases to be execute are prioritized, and assuming that 2-way interaction mentioned in previous section is adopted. Thus all faults can be revealed when the 4th test cases is performed. For the simple example, the solution shown in figure 5.3.2 finishes testing with all faults detected in a total round trip time of 10. It is apparent that the prioritized test cases scheduling ensures the result to be better than only duration scheduling and only test case prioritization, which is 17 and 11 time unit as shown in figure 5.3.3(a) and(b).

With the simple example studied, AHP is proven to be efficient in addressing the prioritized test case scheduling problem when 2-way criteria prioritizition is adopted . In the simple example, test case are labeled exactly one priority number using idea 2-way interaction approach . However, in an industrial case, many test case may be at the same priority level, which is a little different from the solution proposed in this thesis. Thus an industrial case with more test cases may be needed to further verify the efficiency of the prioritized test case scheduling solution. In addition, the way of deciding weight of criteria over final goal should be further discussed to make the AHP solution more precise.

6 Conclusion and Further Work

6.1 Conclusion

The thesis proposes an approach, namely fully automated GUI testing with virtual machines. Currently, GUI testing relies a lot on human involvement. The input of GUI application is a sequence of events and the change of state caused by every event should be verified. Thus hardcoding the expected outcome in program as traditional testing is difficult. That is why current GUI testing needs human assistance a lot. This thesis aims to address the problem by locating GUI testing in virtual machines. Through automating virtual machines, the tests performed in guest OS are automated. Two virtual machine programs, respectively VMWare workstation and Hyper-V, are studied in the thesis to make a better decision for GUI testing. Based on both theoretic analysis and practical use, VMWare workstation is proven to be better suited for GUI testing, because its easy to configuration and the efficient programming Vix API provided .

Result of implementation shows obviously that the performance of fully automated GUI testing with virtual machines is much better than manual tests. Especially when multiple test cases are performed in parallel. However with multiple virtual machines automated simultaneously, the compromise of efficiency and cost should be taken into account in this thesis. If all virtual machines are installed in remote server or cloud machine instead of stored in host machine, the performance of fully automated solution will be improved to a great extent.

Moreover, an optimized solution is proposed to further shorten the round trip time, namely prioritized test case scheduling. The optimized solution aims to detect all underlying faults in the shortest time. With both duration and priority introduced to a test case, the weights of all test cases over final goal is calculated by Analytic Hierarchy Process (AHP) method. Through a simple example studied, the result of the optimized solution is proven to be more efficient than only test cases duration scheduling and only test cases prioritization.

6.2 Further Work

Although the solution proposed in the thesis proven to be efficient, some further work can be done to make it more persuasive and precise as listed below:

1. Applying the solution to a bigger scale of test suits. Current work only focuses on a limited set of test cases from ABB. However, with more test cases applied, the solution proposed can be evaluated more accurately.

2. The weight of duration and priority over final goal for the optimized solution is simply set to be equal in this thesis, although the final result shows superior performance. But the weight can be designed in a more scientific way.

3. An industrial case implemented with optimized solution should be studied to further illustrate the efficiency of prioritized test cases scheduling.

4. Virtual machines in this thesis are installed on host machine directly, which will affect the performance of testing. Considering automating virtual machines stored on a remote server, the performance will be enhanced greatly. This should be verified in the future.

References

[1] J. Offutt A. Andrews and R. Alexander. Testing web applications by modeling with fsms. *Software and Systems Modeling*, 4(3):326–345, 2005.

[2] Michel Mittaz Alain Hertz. Heuristic algorithms. *Theory, Solutions and Applications*, pages 327–386, 2000.

[3] Emil Alégroth. Transitioning manual system test suites to automated testing: An industrial case study. *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 56 – 65, 2013.

[4] Paul Ammann and Jeff Offutt. *Introduction to software testing.* Cambridge University Press, 2008.

[5] Milan Jovic Matthias Hauswirth Andrea Adamoli, Dmitrijs Zaparanuks. Automated gui performance testing. *Software Quality Journal*, 19(4):801–839, 2011.

[6] M.L.Soffa Atif M.Memon, M.E.Pollaek. Coverage criteria for gui testing. *8th European software engineering conference*, pages 256–267, 2001.

[7] Bei bei Yin. A case study for invalidating the markovian property of gui software structural profile. *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, 1:447 – 454, 2006.

[8] Boris Beizer. *Software testing techniques.* Dreamtech Press, 2003.

[9] Emil Borjesson. Automated system testing using visual gui testing tools: A comparative study in industry. *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 350–359, 2012.

[10] Renee C Bryce, Sreedevi Sampath, and Atif M Memon. Developing a single model and test prioritization strategies for event-driven software. *Software Engineering, IEEE Transactions on*, 37(1):48–64, 2011.

[11] B. Chazelle. The bottomn-left bin-packing heuristic: An efficient implementation. *IEEE Transactions on Computers*, page 697–707, 1983.

[12] Robert P Goldberg. Survey of virtual machine research. *Computer*, 7(6):34–45, 1974.

[13] Brian R Gruttadauria, Andreas L Bauer, Gregory W Lazar, and Munish T Desai. Common information model (cim) translation to and from windows management interface (wmi) in client server environment, November 29 2005. US Patent 6,971,090.

[14] Wenqi Huang, Duanbing Chen, and Ruchu Xu. A new heuristic algorithm for rectangle packing. *Computers & Operations Research*, 34(11):3270–3280, 2007.

[15] Takayuki Itoh, Yumi Yamaguchi, Yuko Ikehata, and Yasumasa Kajinaga. Hierarchical data visualization using a fast rectangle-packing algorithm. *Visualization and Computer Graphics, IEEE Transactions on*, 10(3):302–313, 2004.

[16] Hu Jin. Finite state machine for automatic gui testing. *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*, pages 1 – 4, 2009.

[17] David S Johnson. The np-completeness column: An ongoing guide. *Journal of Algorithms*, 7 (4):584–601, 1986.

[18] Jung-Min Kim and Adam Porter. A history-based test prioritization technique for regression testing in resource constrained environments, 2002.

[19] C. D. Vecchi M. P Kirkpatrick, S. Gelatt Jr. Optimization by simulated annealing. *Science*, page 671–680, 1983.

[20] D Richard Kuhn, Dolores R Wallace, and Albert M Gallo Jr. Software fault interactions and implications for software testing. *Software Engineering, IEEE Transactions on*, 30(6):418–421, 2004.

[21] IBRAHIMS Kurtulus and EW Davis. Multi-project scheduling: Categorization of heuristic rules performance. *Management Science*, 28(2):161–172, 1982.

[22] Martello S. Monaci M. Lodi, A. Two-dimensional packing problems: A survey. *European Journal of Operational Research (Elsevier)*, 141:241–252, 2002.

[23] M. Song M. Yoon, E. Lee and B. Choi. A test case prioritization through correlation of requirement and risk. journal of software engineering and applications. *Software Engineering and Applications*, 5(10):823–835, 2012.

[24] Atif M. Memon. An event-flow model of gui-based applications for testing. *software testing, verificatin and reliability*, 17:137–157, 2007.

[25] Atif M Memon, Mary Lou Soffa, and Martha E Pollack. Coverage criteria for gui testing, 2001.

[26] Morten Mossige, Arnaud Gotlieb, and Hein Meling. Test generation for robotized paint systems using constraint programming in a continuous integration environment, 2013.

[27] Morten Mossige, Arnaud Gotlieb, and Hein Meling. Testing robotized paint system using constraint programming: an industrial case study. In *Testing Software and Systems*, pages 145–160. Springer, 2014.

[28] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake, and Yoji Kajitani. Rectangle-packing-based module placement, 1995.

[29] Ana CR Paiva, João CP Faria, Nikolai Tillmann, and Raul AM Vidal. A model-to-implementation mapping tool for automated model-based gui testing. In *Formal Methods and Software Engineering*, pages 450–464. Springer, 2005.

[30] Atif M. Memon Renée C. Test suite prioritization by interaction coverage. *Workshop on Domain specific approaches to software test automation*, pages 1–7, 2007.

[31] F Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[32] Thomas L. Saaty. Relative measurement and its generalization in decision making why pairwise comparisons are central in mathematics for the measurement of intangible factors the analytic hierarchy/network process. *Review of the Royal Academy of Exact, Physical and Natural Sciences, Series A: Mathematics (RACSAM)*, 102:251–318, 2008.

[33] Thomas L. Saaty. Decision making with the analytic hierarchy process. *Services Sciences*, 1(1), 2012.

[34] Gregg Rothermel Sebastian Elbaum, Alexey G. Malishevsky. Test case prioritization: A family of empirical studies. *IEEE Transactions on Software Engineering*, 28(2):159–182, 2002.

[35] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor., 2001.

[36] Tommi Takala, Mika Katara, and Julian Harty. Experiences of system-level model-based gui testing of an android application, 2011.

[37] Marlon Vieira. A uml-based approach to system testing. *Innovations in Systems and Software Engineering*, 1:12–24, 2005.

[38] Lee White, Husain Almezen, and Nasser Alzeidi. User-based testing of gui sequences and their interactions, 2001.

[39] Lee J White. Regression testing of gui event interactions, 1996.

[40] Yu-Liang Wu, Wenqi Huang, Siu-chung Lau, CK Wong, and Gilbert H Young. An effective quasi-human based heuristic for solving the rectangle packing problem. *European Journal of Operational Research*, 141(2):341–358, 2002.

[41] Qing Xie. Developing cost-effective model-based techniques for gui testing, 2006.

[42] Xun Yuan and Atif M Memon. Iterative execution-feedback model-directed gui testing. *Information and Software Technology*, 52(5):559–575, 2010.

[43] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. Addressing shared resource contention in multicore processors via scheduling, 2010.