Prateek Jain

Non-functional Test Automation for Windows Phone Apps

Helsinki Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

24 April 2016

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Prateek Jain Non-functional Test Automation for Windows Phone Apps 53 pages + 1 appendices 25 April 2016
Degree	Master's Degree
Degree Programme	Information Technology
Instructor	Peter Hjort, Lecturer

Mobile applications are required to be developed in a short period of time to meet the competitive market's demands. This limitation undermines the product quality and reliability. Therefore, it is necessary to undergo a rigorous testing process not only on functional but also on non-functional requirements.

This study is about automating the non-functional testing areas for the mobile applications. At the beginning manual testing is covered and after that the topic is discussed with examples from previous testing systems. This thesis presents one way to develop an automated testing system. The biggest target for this project was to reduce the test results variation, which makes it more difficult to judge the quality of the app and thus increases the risk of bad quality app being pushed to the market and reduce the test cycle by automating the manual testing process.

The outcome of the study is an NFT automated testing system for the test organization. This tool tests the performance and the memory utilization of the mobile applications. The developed automated testing system is integrated to the testing process.

Keywords	Non-functional testing, automation, Windows Phone, mobile apps

Abbreviations/Acronyms

DUT	Device under Test
GUI	Graphical User Interface
GUID	Globally Unique Identifier
CLR	Common Runtime Language
WinPRT	Windows Phone Runtime
WP	Windows Phone
ETL	Event trace Log
OBA	OEM Background Agent
OEM	Original Equipment Manufacturer
WPA	Windows Performance Analyzer
WPAA	Windows Phone Application Analysis
NFT	Non Functional Testing
MSA	Measure Systems Analysis
ANOVA	Analysis of Variance
Perf	Performance
Apps	Applications
AIAG	Automotive Industry Action Group

Glossary

WINPRT

Windows Phone Runtime is a subset of native API that is built into the operating system.

XAP

It is the file format for Silverlight applications used to distribute and install application software onto Microsoft's Windows Phone 7/8/8.1/10 operating system.

CLR

Common language runtime manages the execution of programs, allowing to share common classes written in any of several supported languages.

Standard deviation

It is a measure that is used to quantify the amount of variation or dispersion of a set of data values.

Variance

It describes how much a random variable differs from its expected value

Total gage R&R

A method to measure the variation due to the measurement system including multiple operators using the same gage.

Gage R&R

This study helps to investigate, the measurement system variability and variation caused by different operators.

MSA

Measurement systems analysis determines the total variation in a process from the measurement system.

Contents

1 Introduction

With the rapid evolution of the wireless market, there has been evolution of countless mobile devices in the recent years. For mobile phones alone, a recent study has estimated that the total number of Mobile subscription approaches total global population in 2013, which is around 6.8 billion. The number of smartphone users worldwide will surpass 2 billion in 2018, according to new figures from Gartner, Inc. A smart phone is essentially one with an embedded operating system or hosting environment that is able to run third-party applications, beyond the standard services of SMS, MMS and voice calling [9]. While the demand for increasingly complex mobile applications is sustained so too are users' expectations for quality. Unlike traditional software, mobile applications should have the characteristics of spontaneous interaction, high reliability, and low power consumption.

By 2017, mobile apps will be downloaded more than 268 billion times, generating revenue of more than $77 billion and making apps one of the most popular computing tools for users across the globe, according to Gartner[21]. Thousands of apps are added to the different app stores on daily basis. The apps market is more consumer driven. In such a competitive situation, one has to be prepared not to miss an opportunity. Mobile applications are required to be developed in a short period of time to meet the competitive market's demand. This urgency undermines product quality and reliability since mobile application developers tend to be more driven by the marketability than meticulous design and testing process requires sufficient time. But aside from that, there is a certain expectation of quality and an application with high quality only gets noticed.

Some other reasons are mobile users expect near real-time resolution of bug. Regular upgrades in mobile platforms are forcing developers to maintain app compatibility. The test cycle grows for every device, and every firmware or software update. Therefore, it is necessary to undergo a rigorous testing process not only on functional but also on non-functional requirements, especially response time limits. In this regard, performance testing of applications is the most important element because of the very restricted operational environment based on real-time functions. As for mobile applications, critical performance factors are related to spontaneous interaction, high reliability, stability and low power consumption.

In practice, manual testing of mobile device applications is time consuming, expensive and very difficult to do effectively. It could also lead to the huge variation in the test results and thus producing inconsistent test results, which then makes it more difficult to judge the quality of the testing and thus increases the risk of bad quality app pushed to the market. Automated testing approaches have proven successful in other areas of software development and, more recently, they have attracted attention in the domain of mobile device applications. Automated testing is attractive essentially because it can reduce the costs and time associated with testing, lead to shorter release cycles, allow developers or testers to focus on constructing effective test cases, and ultimately to improve product quality. Therefore, the research objective related to this topic is to determine a tool/method to speed up the release cycles and remove deviations in the test results (on windows phone platform).

Business Problem

Smartphone users are very critical on the performance of an application especially to spontaneous interaction and app stability. If the apps are not spontaneous users generally would not return to these apps.

Testing of mobile device applications is time consuming and can prove very expensive due to the variation in the results (when done manually). Also the release cycle time of the app would increase when testing on multiple devices (different configurations). Thus it becomes very important find an answer to the below business problem.

> *How to reduce the standard deviations in the non-functional test results and simultaneously reduce the release cycle time?*

Scope

As stated in the above section, mobile applications needs to be delivered to the market in quick time with high quality. And non-functional attributes are one of the key factors in making the application successful in the market. Thus performance testing becomes very important element. Non-functional testing has many areas varying from the spontaneous

interaction, high reliability, stability, low power consumption, etc. The scope of the research is limited to the solution for Generic Response time cases and memory usage of the Windows phone apps for Windows Phone 8.0 platform and optimizing the test cycle time.

Structure

Chapter 1 introduces the actual work and the reason behind this thesis.

Chapter 2 covers the research methodology used to carry out the thesis work. It aims to classify the research characteristics throughout a methodology analysis and the reasons behind such a classification. It also throws the light on the outcome of this research work.

Chapter 3 introduces with the Windows Phone8 architecture and the types of the Windows Phone apps. It also tells about the different non-functional test types and their definitions. Apart from this it explains about the MSFT NFT cases and their pass criteria. It also throws some light on the existing tools in the market.

Chapter 4 tells about the existing system and process, how the testing was done manually in the initial test set-up.

Chapter 5 aims to define the requirements for the automation tool and non-functional test cases. And continue by presenting a tool that tries to fulfil those requirements. It also explains how the needed key components are implemented.

Chapter 6 tells about the test results of the apps, which are picked up for the piloting. The results are then compared between the existing system and newly proposed system. Basically the tool is evaluated based on the pilot experience.

Chapter 7 summarizes the complete project and the future of the presented automation tool is briefly discussed. There it is evaluated how well requirements for this thesis, defined in Section 5.1, are met.

2 Method and Material

This section covers the research methodology and process used to carry out the study. It further explains how the data is been collected and finally throws the light on the outcome of this study.

2.1 Research Method

This chapter covers the research methodology used to carry out the study. It aims to classify the research characteristics throughout a methodology analysis and the reasons behind such a classification.

There are several ways to classify research due to the objectives, approach, procedures and data collection. This research applies the action research methodology with the quantitative analysis (approach) of the data being collected during the research work. Action research is initiated to solve an immediate problem and aims at bringing change into organization. Action research is also cyclic and later cycles are used to challenge and refine the results of the earlier cycles.

Quantitative research is 'Explaining phenomena by collection numerical data that are analysed using mathematically based methods (in particular statistics)' [Aliaga and Gunderson (2000)]

2.2 Research Process

As initially covered in the introduction, this research is focused to find an automated solution as a replacement for the manual method of doing the non-functional testing. So, the first step was to gather the data and processes been deployed in the current system. This serves as the input in designing the new system (requirement gathering).

The data can be collected from the existing source based (more details can be found in the data collection sub-section). Once, the data has been collected there should be ad-

equate evidence to prove existing method is not good enough. Hence, a statistical analysis was performed on the existing data. Minitab16 [17] is a good tool which was used in doing the quantitative analysis.

Since this study is focused on specific areas of the non-functional testing (response time and memory usage by Windows Phone mobile apps), all the relevant test cases are collected from the current system and based on the Microsoft guidelines (detailed in Section 5.1.2) for publishing a mobile app. This again forms the part of requirements.

As a matter of fact, Windows Phone does not have a long history in the market, efforts were made to find out the existing solutions based on the gathered requirements. Eventually a tool developed by MSFT called Windows Phone Application Analysis (WPAA) [4] was identified (as somewhat similar). The tool was then analysed and later used for the benchmarking purposes against the research project. The outcome of this phase is also used in designing the new automated system.

Then, the project design was created based on the gathered requirements. And, based on this project design, the relevant technology were identified to be used in creation of the proposed solution.

Once the design and technology was finalized the creation of the project started. After the completion of the project, a fresh round of execution was done for the selected set of applications. Then the data gathered during the new set of execution was again evaluated using the same methods as used before.

After successful piloting for selected apps and hence proving its validity and reliability, the new automated tool can be deployed across the testing organization.

2.3 Data Collection and Material

The goal for the data collection is to capture quality evidence that then translates to rich data analysis and allows the building of a convincing and credible answer to questions that have been posed. Regardless of the field of study or preference for defining data (quantitative, qualitative), accurate data collection is essential to maintaining the integrity of research. As mentioned above this research is based on the quantitative analysis.

During this research various data samples were collected from the existing data available (from the current test data repository) and also the results were gathered using the new system (outcome of this research).

Existing test assets used in the existing system, outcome from the existing tool and Microsoft guidelines for publishing a mobile app formed the part of the requirements. Apart from this the existing process also played an important part of the requirement gathering.

2.4 Research Outcome

The outcome is the creation of new automated tool/method which can help reduce the variations in the test results and also reduce the cycle time significantly for mobile apps (on Windows Phone platform). But the scope of the research is limited to the solution for optimizing the cycle time and reducing the standard deviation in the response time and memory test results.

3 Background

This chapter introduces the Windows Phone8 architecture and the types of the Windows Phone apps. It also discusses software testing and its approaches. Apart from this it explains non-functional test types and their definitions, and covers MSFT NFT cases and their pass criteria.

3.1 Windows Phone 8 Architecture

Windows Phone 8 is one of the later (released on October 29, 2012) entries in the world of mobiles OS, powered with lot of capabilities. To understand how the Windows Phone apps work and perform, it is very important to understand the underlying windows phone architecture.

Figure 1 gives a peak into the Windows Phone platform architecture.

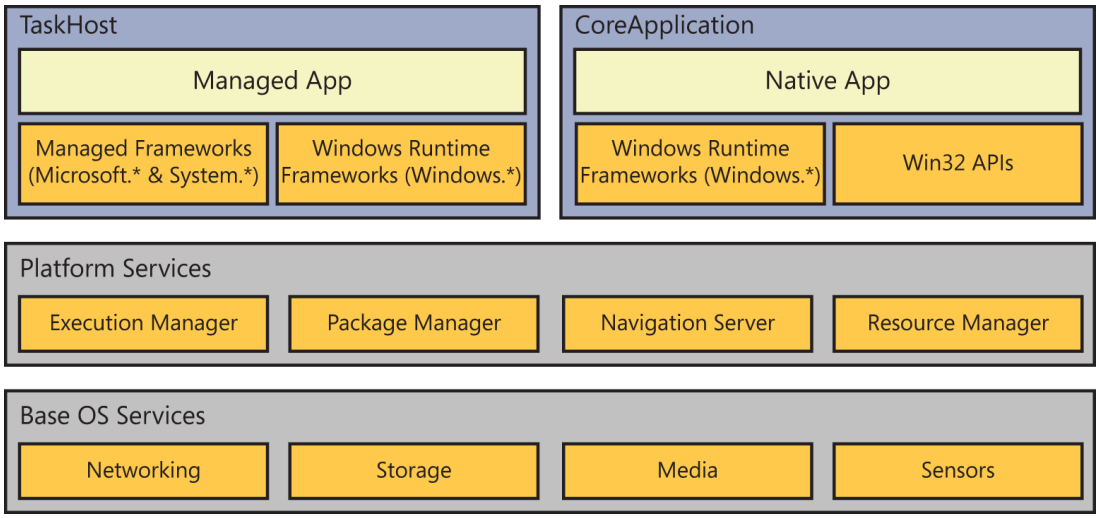

Figure 1 Windows Phone 8 platform for the app models [5]

Windows phone 8 platform architecture comprises of different layers stacked one after another. The bottom block is the shared core which has two parts. Windows core system – contains base OS functionality which is shared across many type of windows devices such as security, networking etc. Mobile core – contains core common language runtime (CLR) which is core .net library, code gen and garbage collector. It also has the trident engine for Internet Explorer, core multimedia and DirectX capabilities.

Above the Windows shared Core block is the platform services which provides different service managers to provide the platform services to the apps been developed and deployed. Below are the different components of this block.

- **Package Manager** is responsible for installing or uninstalling apps and also keeps track of the apps being installed and licensed. It maintains the applications metadata through the app lifecycle and also stores information about the app being tiled on the Home screen. It also tracks of the application's extensibility points which are registered and can be used in appropriate places in the OS [5].
- **Execution Manager** takes care of the events associated with app's execution lifetime like app launch, shutdown and deactivation. A hosting process is also

created by the execution manager for the app so that it can run under it. Execution manager is also responsible to perform similar task for background processes and helps in proper scheduling of those tasks. [5]

- **Navigation Server** manages all of the navigation events between foreground apps on the windows phone. When an app tile is launched from the Home screen, the navigation server passes that information to the execution manager so that the chosen app can be started. Similarly, if the back key is pressed and hold and an app is picked from the list of background apps, the Navigation Server informs the Execution Manager to reactivate that application. [5]
- **Resource Manager** is responsible for monitoring the use of system resources like CPU, memory etc. and ensure the phone is always responsive. It keeps track of the system resources been used by the active processes and also enforces the constraints if needed. For instance, an application or background process can be terminated if it exceeds the allocated resource pool. [5]

Application model lies on the top is the Windows phone platform model. The Windows Phone SDK allows to build apps using a variety of languages and tools. One can build the app using XAML and the choice of managed language, which allows to maintain the investments from existing apps. To provide greater flexibility and performance, Windows Phone 8 introduces the ability to use C++ within the XAML app and in games written using Direct3D. Figure 2 illustrates the set of APIs that make up the Windows Phone API.[1].

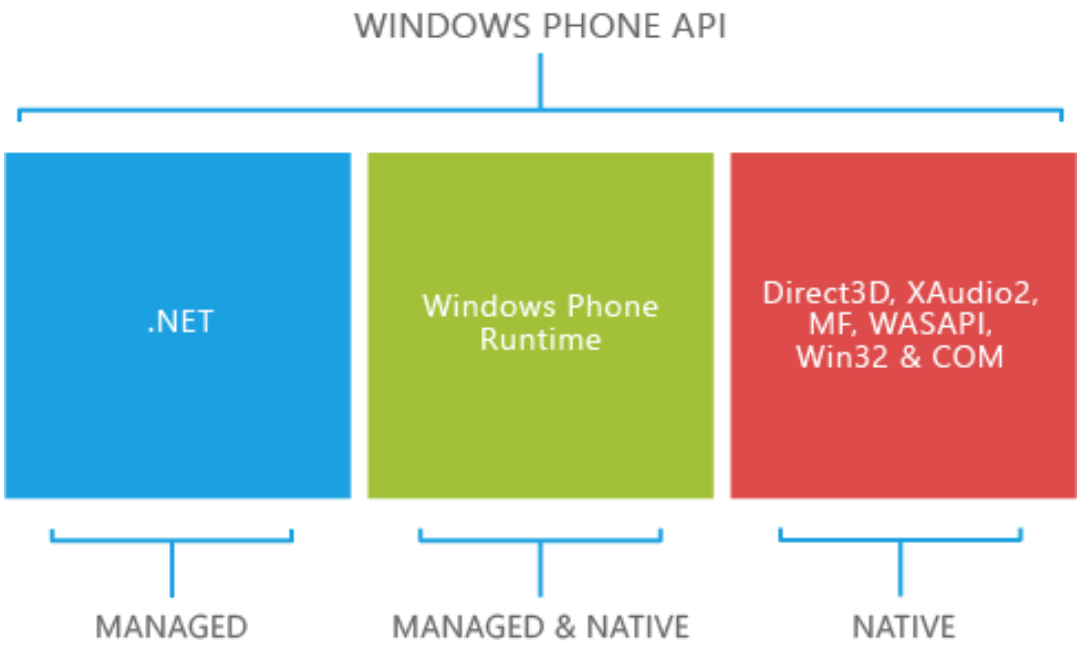

Figure 2 Windows Phone API.[1]

The .NET API represents the managed API on Windows Phone 8 and simplifies the process of accessing user data. An example would be to facilitate the sign-in experience for users. The managed code runs under the control of common language runtime (CLR). Windows Phone Runtime is the subset of native API. They are implemented in C++ and projected into different languages, making it easy to use. On the other hand, Win32 APIs gives the access to low-level features of the platform [1].

3.1.1 Types of Windows Phone Apps

Windows Phone 8 supports several different application flavours, as described in Table 1.

Table 1: Windows Phone 8 app types [5]

App type	Description	Languages Supported	UI Framework	APIs supported
XAML	In this app XAML and managed code is used to implement a UI and not Direct 3D code is used.	C#, Visual Basic	XAML	Microsoft .NET, Windows Phone API, Windows Runtime API
Mixed mode	These apps follow the XAML app structure but allow for the inclusion of native code. These types of apps come into picture when most of the code used is native but also there is a need access to the XAML UI framework and some of the features that are only available to XAML code. The existing native library can be well used in these type of applications.	C#, Visual Basic, C/C++	XAML, Direct3D (via DrawingSurface)	NET Windows Phone API, Windows Runtime API, Win32/COM API (within Windows Runtime components)
Direct3D	These types of app are best suited for games. The apps using Direct3D code offers best to extract the most out of the phone's base hardware. They also offer the code sharing between Windows and Windows Phone.	C/C++	Direct3D	Windows Runtime API Win32/COM API

Windows Phone provides an immersive "hub" experience for its primary content type, and provides a fair amount of extensibility to extend the built-in experience. These extensibility points offer additional ways for users to invoke the app. Apart from the apps, Original Equipment Manufacturer (OEM) can create the background agents and services for the Windows Phone.

3.2 Software Testing

Software testing plays an importance role in delivering the reliable and quality mobile application to the end user, in this dynamic world of continuous and frequent software releases. Software testing approach is broadly categorized as manual and automated testing. Even though exploratory testing (manual) helps to better understand the weakness of the application, it comes with its own set of cost and reliability issues. In this

research the focus is on the test automation, which can help to overcome the reliability and long testing cycle issues of manual testing.

3.2.1 Automated Software Testing

Test automation helps in repeatedly executing the test cases with high consistency on different versions of systems under test. Automation acts as the savior of the test engineers in case of repetitive tasks, thus easing their workload. Test automation leads to more accurate and reliable test results, it also shortens the testing cycle time [19].
Table 2 tells about the common test automation benefits against manual testing [19].

Table 2: Common test automation benefits

Automation testing perform the repetitive operations with consistency and in shorter span of time.
Automating the test cases is very helpful when the test execution is very frequent and the code changes very frequently. By executing the same automated test on the newer version of software can help in finding the regressions.
Automation testing can enable executing the same test set on different machines with different OS platform combinations, concurrently.
Automating repetitive and uninteresting tasks releases test engineers for more rewarding and demanding tasks.
Automation runs test cases significantly faster than human resources and significantly reduces the chance of variation in the test results. Thus helping to maintain the high quality of the software.

It is also worth discussing some issues with the test automation. The initial cost of making an automation system can be very high as compared to the manual testing and might take some time in the beginning, thus needs support from the management. Management can have unrealistic expectations that it could solve all the testing problems. It is not necessary that new issues are found in every round of testing, until there is some code change. Also, it is difficult to find the usability issues with the test automation approach. Maintaining the test automation environment and the test assets in the frequently

changing environment can be very challenging and can cause the breakdown of the test automation system, while the manual testers can accustom themselves easily [19].

3.2.2 Functional Vs Non-functional Testing

Functional Testing of the software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. The main objective of the functional testing is to determine if the output produced by the system matches the pre-defined expected outcome. Apart from this, it is also important to test the non-functional aspect of the applications. With the limited resources available on the mobile devices it even becomes more important to cover the areas like performance, security, usability, power usage, reliability and resource management [7].
In this thesis the focus is on non-functional testing of the mobile apps, which is discussed in detail (Section 3.3).

3.2.3 Types of testing

Traditionally software testing can be sub-divided into three levels such as unit, integration and system testing. The concepts of testing in this thesis revolves around the testing of the mobile applications.

Unit testing

The smallest building block of any system is called a unit. Every unit has an interface and is used for the interaction and also for testing it. Unit testing is handled by programmers who know the code under test before handing over the system to the testing team. The goal of it is to test, if each unit works as intended before being integrated to the main system [20]. There are many approaches for the unit testing and test first development, also called TDD is one of them. Test driven development (TDD) is a software development method that uses short iterations based on the pre-defined test cases. It requires the developers to write the automated test units before writing the actual code.

Then, a test is run and then the code is refactored to the acceptable standards. Such development process induces progressive growth of design and completion of progressive codes and results in optimized unit tests carried out [6]. However, the scope of this study is on system level test automation than unit level testing.

Integration Testing

The testing of the combined units of the application, to determine if they work correctly together is integration testing. Integration testing can expose problems with the interfaces among program components before trouble occurs in real-world program execution [20]. There are different approaches for integration testing like bottom up and top-down approach. In top-down integration testing, the highest-level modules are tested first and progressively lower-level modules are tested after that. While, Bottom-up integration testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules. Continuous integration is one of the most commonly approach these days. It helps to find the regressions and remove them at early phase of daily integration. Thus reducing the integration problems and allows rapid software delivery.

System Testing

This is the next level in the testing and tests the system as a whole. Once all the components are integrated, the application is tested rigorously to verify that it meets the functional and non-functional requirements as specified in the user acceptable document. System Test approach assists in mitigating risks and ensuring a successful project. During system testing the product is tested for the graphical user interface, usability, end-to-end functional testing and non-functional testing aspects etc [8]. This research focuses on the non-functional testing aspects of the mobile applications.

3.3 Non-functional Testing Area

With the increased versatility of the mobile apps, it is becoming necessary to keep in mind not only the mobile functional elements but also the non-functional elements, when

determining the test scope. The term “non-functional testing” refers to testing those aspects of a software application that may not be connected with a defined user action or function (like, interaction, high reliability, stability and low power consumption). The correct specification and adherence of non-functional requirements similarly plays an equal role, in the success of mobile applications.

Therefore, it is important to discuss the different non-functional testing areas. Table 2 provides a brief description about the different test area covered under non-functional testing.

Table 2 Non-functional test area definitions [9, 12]

NFT Testing Areas	Test Type	Description
Performance testing	Response Time	Response time covers sub areas like Completion time, reaction time and latency. Response time is measured as, total time between initial user input and completion of desired action. While reaction time is the elapsed time between initial user input and the subsequent response
	Install Time	Time taken by the application for the installation
	Boot Time	Time taken from pressing the power button till the Home screen appears for first time and subsequent boots.
	Benchmarking	Measuring the similar app on competitor devices and analysing the results in comparison to Windows Phones
Resource Utilization	Memory & CPU Testing	Measure the memory usage for the WP apps and services to check they should not exceed the memory limits set by MSFT Technical guidelines.
Power Management	Current Consumption and sleep mode testing	Measuring average current consumption of the device during the app usage over a period of time. And verifying that device returns to sleep mode after different use cases.
Stress and Reliability	Endurance, Long period testing and Robustness	Endurance: Measures SW reliability with feature specific long lasting user operations.

		Long Period Testing: Measures SW reliability from product- and user profile specific point of views with long term usage. Evaluating and validating a software system's tolerance to faults which occur externally to the system under the test

Even though most of the above discussed non-functional aspects are crucial for a mobile application, this paper focuses on the Generic Response time and memory usage aspect of the windows phone application.

3.4 Windows Phone NFT Test Cases and Certification Criteria

If a developer wants to publish an application to the windows Phone store, the app must comply with the certification requirements specified by MSFT. The certification requirements are divided by type, such as app policies, content policies, and app submission requirements etc. This section gives a brief about it focusing on the non-functional aspect of the application. Below are the Microsoft technical certification criteria for application responsiveness [2, 5].

1. App Launch time: The app must render the first screen or a splash screen within 5 seconds after launch. Also, the app must be responsive to user input within 20 seconds after launch.
2. App responsiveness after being closed: When an app is started after being closed, its launch time must meet the above requirements for App Launch Time (1).
3. App responsiveness after being deactivated: A Windows Phone app is deactivated when the user pushes it to the background. When an app is activated after termination, it must meet the requirements for App Launch Time (1).
4. App responsiveness: App must not appear to be unresponsive for more than three seconds, if it perform some operation. An example would be, downloading data over a network connection or transitioning between different views, the app must display a visual progress or busy indicator.

While designing a mobile application, it is very important to keep in mind that the application can be used on different devices with varying memory. The size of the default memory cap imposed on an app is determined by the app as well as by the memory size of the device [3]. Thus Microsoft has defined certain memory limits for different app types, depending on the device configuration. Table 3 gives an overview on the memory limits for different windows phone 8 applications.

Table 3: MSFT Memory Limits [3]

App type	Lower-memory phones (512 MB)	1-GB phones	2-GB phones
Windows Phone 8.0 (all types)	180 MB	380 MB	780 MB
Silverlight 8.1 and Windows Runtime 8.1	185 MB	390 MB	825 MB
Continuous Background Execution (Windows Phone 8.0 only)	150 MB	150 MB	300 MB

*To use the memory limits described in the preceding table, 2-GB phones must also have Windows Phone 8 Update 3 (that is, a version equal to or greater than 8.0.10492).

The above defined certification criteria should be fulfilled before an application is submitted to Windows Phone marketplace. Thus, it forms an important part of the requirement for this study.

3.5 Existing Tools by Microsoft

Windows Phone Application Analysis [4] includes the option to monitor the app while exercising its features as an ordinary user would use it.

The goal of app monitoring is to help understand the quality of the app, and to give an actionable feedback to improve it. This information helps in improving the app long before it reaches the end user, and to differentiate it from other apps by its responsiveness and its responsible resource usage. The app monitoring feature aims to capture all the key metrics that are relevant from quality perspective, and then to rate the app based on these metrics [4].

App monitoring can help to identify issues such as the following:

- Slow start up time.
- Slow response time to input, such as scrolling or zooming.
- High battery drain.
- Network latency.
- High cost of network data.
- Poor performance as the quality of the network signal changes.
- Out of memory errors caused by high resource usage.

Figure 3 shows an example graph from the application analysis tool, showing the app behaviour and performance.

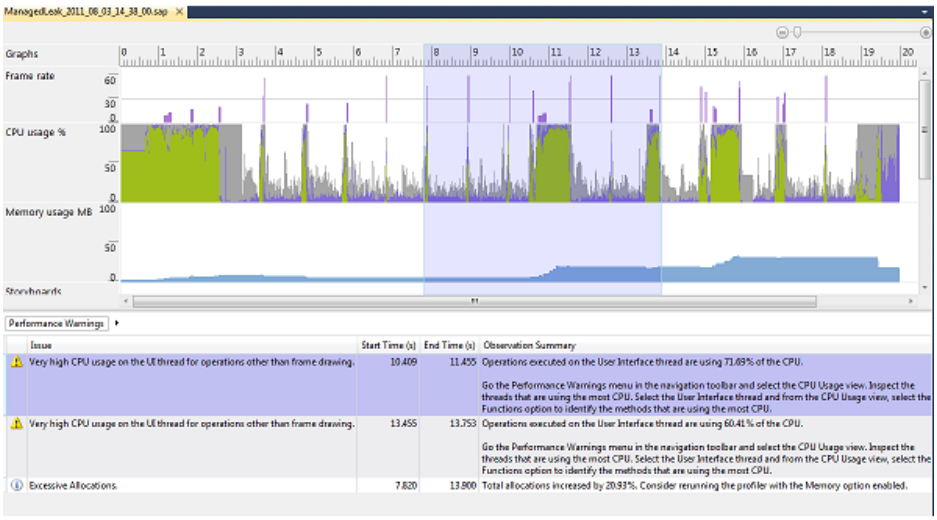

Figure 3: Snapshot of Performance Graph from Windows Phone Performance analysis tool [4]

Each area is color-coded and symbolizes different performance aspects. The test results or the graphs might vary based on the app type. For instance, the frame rate section displays the number of screen redraws, in frames per second. While the frame rate is not shown for XNA framework apps. The memory usage section shows the amount of phone memory being used by the app in megabytes.

3.5.1 Limitations of Existing System

The Windows Phone Application Analysis [4] has some limitations, due to which it is not considered as an alternative to the new proposed automated system. Below are few more limitations of this tool.

1. Need the access to the source code. Thus it becomes difficult for a tester to test the mobile application, if he/she don't have the access to the application source code. More suitable for the unit testing. While this thesis focuses on the system level testing.
2. Confined to limited test set. Not giving the values for all the needed test cases. The performance test cases were pre-defined and cannot be customized.
3. Not expandable. Unable to add more scenarios for the non-functional testing.
4. Unable to trace the Windows Phone Runtime (WINPRT) apps memory.
5. Extensive test reports with the device under test and OS information is not available.
6. Unable to get the continuous logs for trend analysis. Thus unable to use the test results with the existing reporting system.

As discussed in earlier sections, manual testing is very expensive and very difficult to do effectively. The WPAA tool also needs the manual intervention for executing some of the scenarios. Thus based on these limitations a decision was made that Windows Phone Application Analysis tool cannot be used for the complete coverage of the non-functional testing of the Windows Phone apps.

4 Initial State Setup

The aim of this chapter is to explain the test setup and testing process of the existing system, which is time consuming and prone to deliver inconsistent results. Since the focus of this research work is confined to the UI responsiveness and memory utilization of the mobile applications, this section explains only about those NFT areas.

As per the old testing process once the developer commits the code, it is submitted to the continuous integration. During this phase the functional unit test cases are executed to check the regression and then it is moved for the functional system testing. After the

approval from the Quality lead, the release further proceeds to the non-functional testing area. Figure 4 shows the testing process map of the old system.

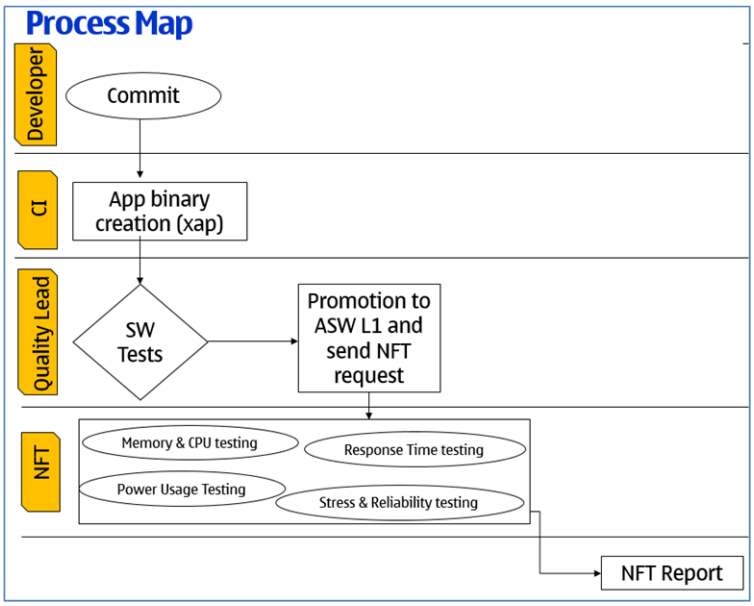

Figure 4: Initial Test process map

As seen in Figure 4, the non-functional testing was done at the very late stage. This could lead to slippage of the non-functional bugs till the very end and sometimes it becomes very expensive to fix them, thus, increasing the overall cost and delay of the project. But aside from that, manual testing also adds to the inconsistent test results and delayed test cycle time.

Performance testing

Performance testing covers overall application responsivness, device boot time, latency and reaction time, as explained in Section 3.3. The initial test setup was based on manual testing. All generic perofmance scenario (defined in Section 3.4) were manually tested using high speed camera. Each scenario was excuted multiple times to get the accurate results. Also, each scenarios were executed on multiple devcies with different configurations. This leads to the longer testing cycle time. These scenarios could be done simultaneously with many testers, but could lead to variation in the results due to the human error. Performance results have to be accurate and understood by the testers. They should provide the same result and leave minimum room for interpretation and thus human error. As this has a direct impact to the costs accumulated during the test rounds on the application development phase.

Resource Utilization

With the limited memeory available on the devices, application memory monitoring becomes one of the major contributor to overall application non-functional testing. Resource utilization covers the memory utilization measuement for the apps, services and background Agents. During this test the memory usage for the WP apps was checked along with the their peak memory. As it should not exceed the memory limits set by MSFT Technical guidelines.

The initial test setup was based on manul testing and WPA tool was used to measure the resource utilization by an app. Since, WPA tool needs the app source code to run the memory profile. The tester needs to setup up the development eviornment on his machine. Then, the tester starts the monitoring via the WPA tool and would simultaneously run the test steps for different scenarios. Each of these scenarios were then repeated on devices with different memory sizes (512 MB, 1GB, 2GB etc). Repeatedly executing the same steps manully, makes the system prone to variations in the test results. The same process was repeated on every release to find the regression. After each round the resuts were noted manully and there was no continous logging available for the results. This process was time consuming, leading to a direct impact to the costs accumulated during testing rounds on the application development phase.

Thus, it was proved that, in practice manual testing of mobile device applications is time consuming, expensive and very difficult to do effectively. It could also lead to the huge variation in the test results, which then makes it more difficult to judge the quality of the app and thus increases the risk of bad quality app pushed to the market. Also the release cycle time of the app would increase when testing on multiple devices (different configurations) and different operating system versions.

5 Test Automation

The aim of this chapter is to define requirements for the automation framework and continue by presenting a tool that tries to fulfill those requirements. It also explains how the needed key components are implemented.

5.1 Requisites for Automated Test Tool

There are multiple requirements such as repeatability, reproducibility, short testing cycle, test monitoring etc which suggest a new automation tool is needed. In high level these requisites are divided into three sections. Each section has its own weightage in the creation of the new automated tool and explained in the below section.

5.1.1 Requisite as General Test Automation Tool

This section covers the basic high level requirements for any test automation framework or tool. Even though there has been many researches and developments done in the field of test automation. However, the high level requirements for the test automation remains same even today. These high level requirements can be categorized as automatic test execution, Ease of Use and tool maintainability.

The first and the foremost requirement for an automated test tool is fully automatic test execution of the test cases. However, executing tests is not enough, the tool must also be capable to analyze the test results, handle the runtime exceptions during test execution and report the test results in a readable format for all the stakeholders [16]. The test automation framework or tool should be easy to use by the engineers or it is very likely to be abandoned. The test engineers should be able to design and edit the tests, run them and monitor the test execution status with ease. If a new person joins the team, he or she should be able to start quickly without much of the programming skills [18]. Maintainability is another very important aspect for any software, be it a test tool or the software under test. The tool must be easy and fast to maintain, when the test system or the environment changes or updates. Apart from this it should be designed in such a way that new features can be added to the tool when the need arise [19].

5.1.2 Requisites from Non-functional Prospective

With the limited memeory available on the devices, application memory monitoring becomes one of the major contributor to overall application non-functional testing. Apps consuming more more can lead to the degraded UI performance of the apps.
Non-functional requirements have been derived from the Microsoft Technical Certification Criteria defined in the Section 3.4 above. The requirements are then broken down in the below test cases.
The NFT test cases are divided into 2 segments and Table 4 covers the performance test case definitions for an application.

Performance test cases definitions

Table 4: Performance Test cases

Requirement	Requirement Text
Measure the launch time of the splash screen	Application's first screen or a splash screen must be visible within defined time after launch.
Measure the first load time of the application	Measure the time elapsed between the initial user input until the app is fully visible and ready for the user input. The measured time must be within the defined time limits.
Measure the app load time after it has been closed	Measure the time elapsed between the initial user input until the app is fully visible and ready for the user input after being closed. The measured time must be within the defined time limits.
Measure the app load time after it has been de-activated (pushed to the background, not active.)	Measure the time elapsed between the initial user input until the app is fully visible and ready for the user input after is been de-activated. The measured time must be within the defined time limits.
Closing time of an application	Measure the closing time of the application and check it is closed within the defined time period.
Is the app closable from main screen?	Check if the application is closable from its main screen using the back button.
Installation time	To measure the application installation time.

The performance test cases defined in Table 4 cover the most generic responsiveness use cases for any mobile application. The main focus of these test cases is cater the initial app launch experience of the end user. However, Table 5 charts down the memory usage test cases for the Windows Phone 8 mobile apps and services.

Memory Test cases

Table 5: Memory test Cases

Measure the memory usage for the WP apps	To measure the memory usage for the windows phone apps and check they should not exceed the memory limits set by MSFT Technical guidelines on devices with different configurations.
Checking Commit limit for the background agents	To measure the commit limit of the windows phone background agents and check it should not exceed the defined limits.
Measuring the memory by the services.	To measure the memory usage for the individual and bundled services. And check that a service should not cross the memory limit.

MSFT has defined separate memory limits each use case defined above, which are discussed above in Table 4, for instance an app has lower memory limit on a phone with less memory and higher limit for mobiles with more memory. The above defined test cases holds fair amount of weightage in this new NFT automated test tool and are considered the base of the new automated tool.

5.1.3 Requisites Gathered from Current System

As mentioned in the above sections, most of the existing non-functional testing was done manually, which lead to the delayed test cycle and inconsistent test results. And it made difficult to judge the quality of the app and thus increases the risk of releasing a good app with bad quality. A root cause analysis was conducted for the existing system using the fish bone diagram. A fish-bone diagram is a quality defect prevention tool to identify potential factors causing an overall effect.

The below fish-bone diagram (Figure 5) depicts the problems in the current system which have severe impact on the current way of working and testing life cycle.

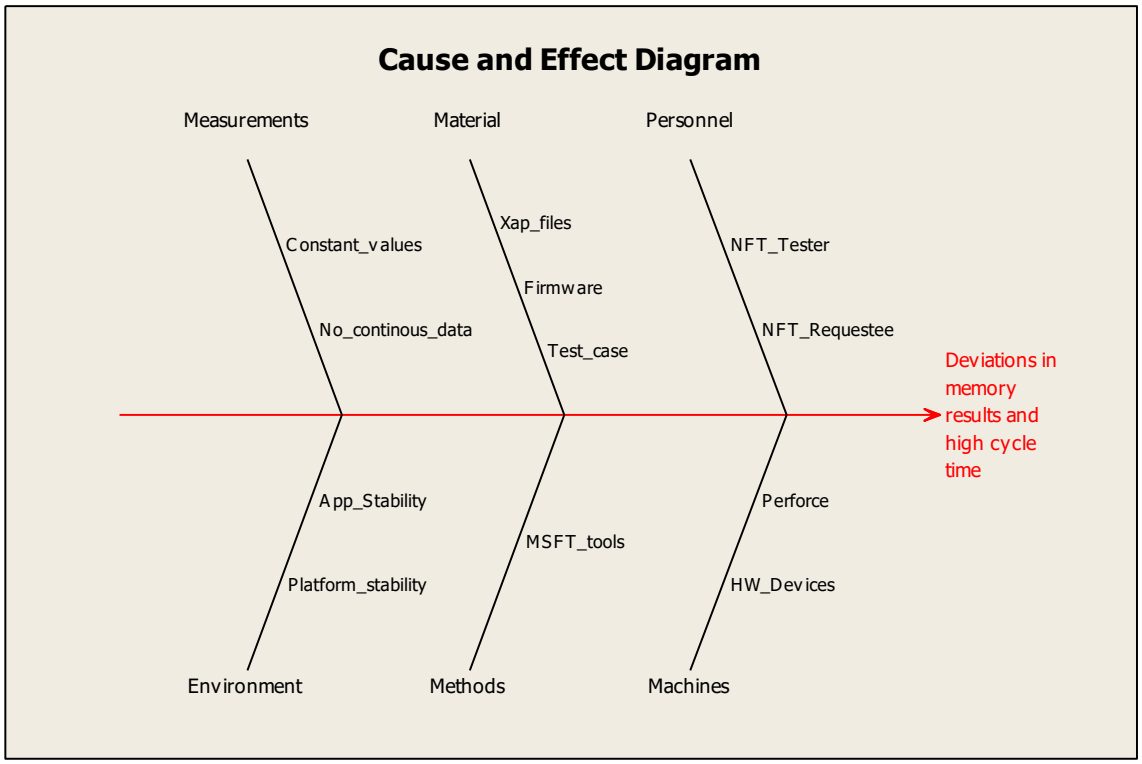

Figure 5 Cause & Effect Diagram

The above identified reasons are turned into the requirements and must be addressed in the new automated tool. Reproducibility and repeatibilty are the major factors which can help in reducing the the variance in the test results. Reproducibility is the ability of a gage, used by multiple operators, to consistently reproduce the same measurement of the same part, under the same conditions [17]. Testing should provide the same result for same use cases (if no changes are made to that area) and leave minimum room for interpretation caused by human error. While repeatibility is the ability of an operator to consistently repeat the same measurement of the same part, using the same gage, under the same conditions[17]. While testing a software it is very important to execute each scenario multiple times to get the accurate results.

As the earlier testing process was manual, execution speed of the test case was another major issue. Thus automating the new testing process can decrease the overall testing cycle time and increase the repeatability and re-useability. As discused in Section 3.5 the exisiting tool (WPAA) lacks the capability of continous results logging and device extensive reports, they become important requiements for the new automated system. Test logs have lot of informaton, but it is good to see the test reports which can provide

the statistical information about the results and cater all the stakehlders. The test report should have a summary about the list of executed test cases with their status, along with the info on below fileds, see below:

- Name and verion of the app
- Device configuration
- OS & Firmware details.
- Total number of passed and executed test cases.
- Etc.

Test reports can either be generated at the end of test execution or later based on the test logs.

This section first defined the high level requirements for the general test automation framework. Then it explained the requirements from the non-functional perspective, which is the core of this tool. And in the later part, the requirements are gathered from the existing system. Though all the requirements go hand-in hand. But due to the time limitations, the focus on each section was defined (in terms of the weightage), based on the business needs. Which in turn drive the development of this automated tool. The next section builds from this foundation and suggests the test tool design fulfilling these requirements.

5.2 Non-functional Test Tool Design

Design is one of the most important phase in the SW development of any tool. Multiple requirements and limitations of the current system suggests that another system is needed for achieving the reduced test life cycle and accurate results. Thus based on the above requirements in Section 5.1, the new automated tool has to be designed.

This section explains about the design (Figure 6) and layout of the newly proposed test tool.

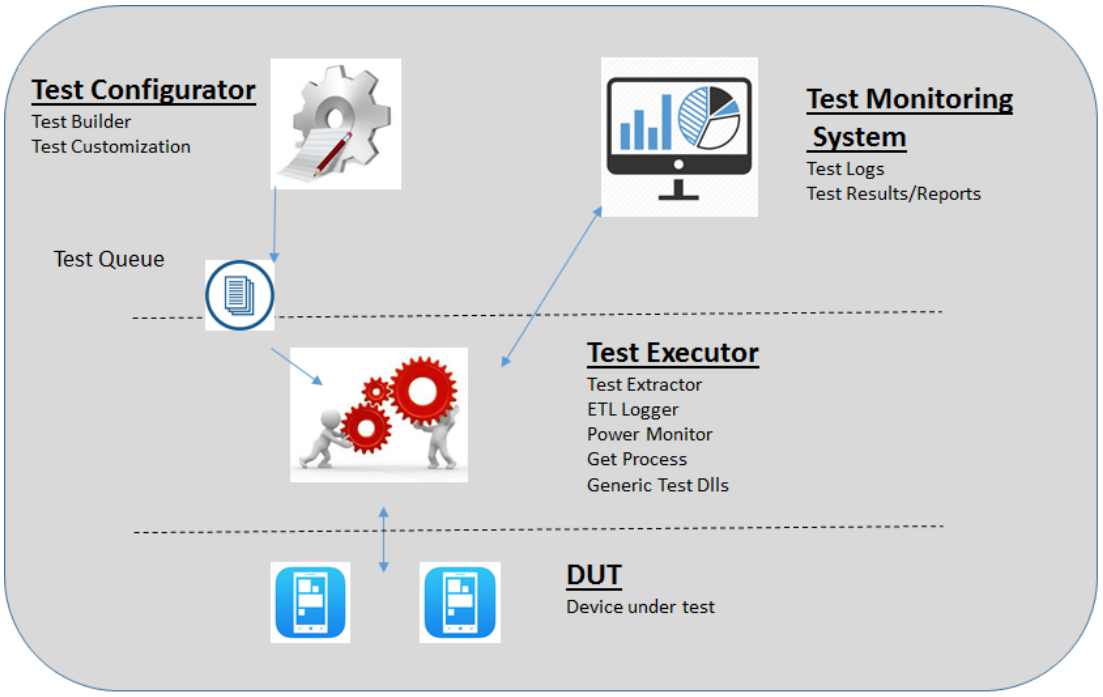

Figure 6: High level - NFT automated tool design

Figure 6 shows the high level design for the new NFT automated test tool. The new test tool is designed in such a way that it is easy to use and maintain. This tool is based on two tier architecture, the top tier is the user interface, which gives test engineers the flexibility for designing the test cases and controlling them. This layer also showcases the test results in user friendly manner. While the underlying tier is the core engine of the new automated tool and performs the real execution task and interacts with the device under test. Figure 7 gives the detailed overview on the design of the new NFT tool.

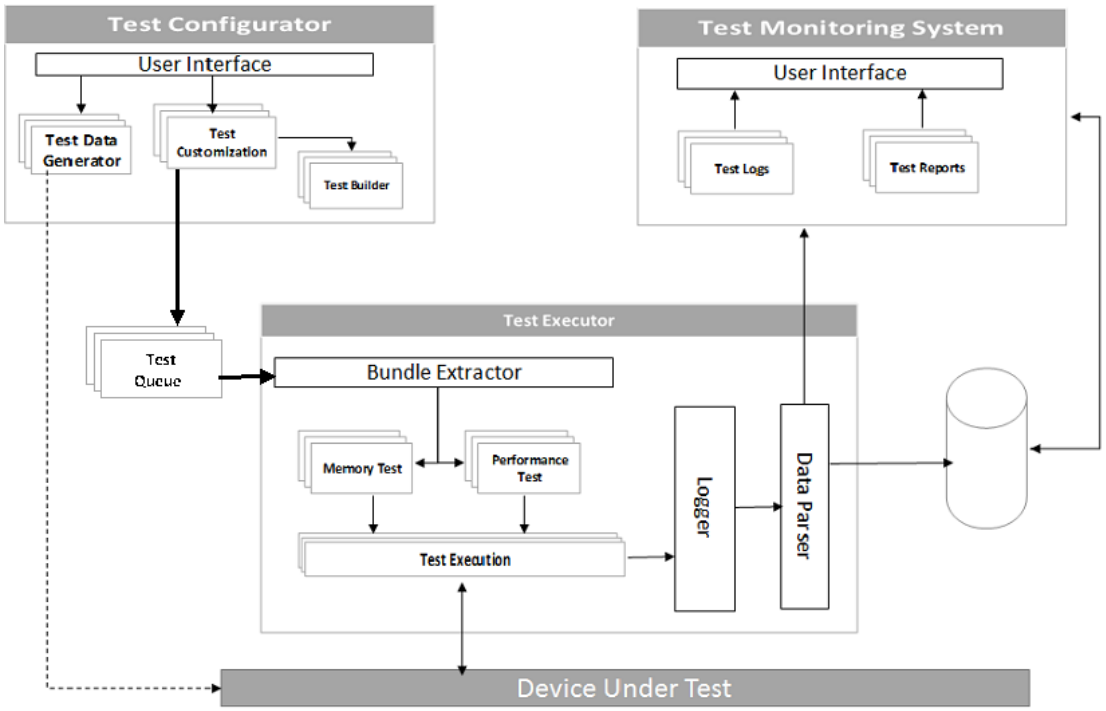

Figure 7: Detailed design diagram - NFT tool

Test Configurator

Test configurator is the non-functional test engineer/expert playground. It has been designed in such a way that a test engineer can use it with minimal training or help.
Test configurator has multiple functionalities, ranging from dummy data generation to test case/set configuration.

Test Data generator is used for generating the dummy test data. The dummy test data is needed to simulate the stress and user scenarios on the device. This data can be photos, music, videos files etc. which is used to fill in the physical memory of the device. Apart from this, the test generator is also capable in creating the process which can eat up the RAM of the device under test (DUT).

Test customization module is used for creating the new test cases/sets and also editing the existing test assets. During the test case creation user can select the process/service against which the test case has to be executed. User also has the flexibility to select the number of test iterations and the device type. And the created test cases can be stored into xml files.

Once the test case is created a test bundle is created using the **test builder**. This test bundle contains all the needed parameters required for the test execution. This bundle is then pushed to the test queue.

Test Monitoring System

Test monitoring system is used for controlling test execution and checking test results. It is expected to have at least the below listed capabilities

- Controlling the test execution.
- Stopping test execution.
- Setting up the logging level.
- Monitoring test execution while tests are running.
- Viewing test logs while tests are running and afterwards.
- Viewing current and old test reports.

The test monitoring system is designed to be a GUI based interface for controlling and stopping test execution, with underlying scripts to control the test execution. The test logs and results should be presented in a readable and graphical format, which is much richer than the plain text reports. The graphical interface also allows the stakeholders to interpret the results in a convenient manner. The exporting of the test results/reports in different formats should also be supported, which can be used by other reporting systems that are already in use.

Test Executor

Test execution system is the core of this automated tool. It accepts the test bundles been prepared using the test configuration system and pushed to the test queue as illustrated in Figure 7. The framework concept relies greatly on reusable components. The most significant component is the test executor which consists of multiple subcomponents. Its main components are bundle extractor, the test library, process monitor, the test data parser and other utilities like logger. This component, is the only one that interacts with the device and executes the test cases on the DUT. How all these and other components work together is illustrated in Figure 9.

Bundle Extractor

Test bundle generated by the test configurator which has all the test details, is pushed to the test queue in the form of a zip file. The bundle extractor, exacts the needed info like app details, number of iteration, test type etc. and passes to the specific test executor.

Executor

Performance Test Preparator

Once the bundle extractor gets the application info like app name, version, GUID etc., the performance test preparator combines this info with other performance parameters. The performance test execution is based on the event based tracing (ETL). An xml is been generated combining all the test cases and their corresponding parameters. A new table node is been created for each test case and things under a particular table is executed together.

Example:

```
<?xml version="1.0"?>
<Data>
  <Table Id="XAML_Appxxx_Setup">
    <Row Description="App_xxx">
      <ParameterName="PackageAumId">
App_xxx__8wekyb3d8bbwe!x36f9fa1cyfdady4cf0y99ecyc03771ed741ax</Parameter>
      <Parameter Name="Prelaunch">back;</Parameter>
    </Row>
  </Table>
  <Table Id=" XAML_Appxxx">
    <Row Description="App_xxx">
      <ParameterName="PackageAumId">
App_xxx__8wekyb3d8bbwe!x36f9fa1cyfdady4cf0y99ecyc03771ed741ax </Parameter>
      <Parameter Name="LaunchApp">tap 200px 300px; flick left; flick left;</Parameter>
    </Row>
  </Table>
</Data>
```

Once this is done, the command is passed on to the test execution module. The ETW tracking happens in parallel. The trace logs are then parsed using a PowerShell script and timestamps of various events are been collected and calculated for the end results.

Memory Test Preparator (Process Monitor)

Similarly once the Memory test preparator gets the details form the bundle extractor, it checks for the process and memory test type. As discussed earlier, the tool could support the memory measurement for the applications, services and the background agents. The new tool also supports both the automated functional test and manual test. Based on these parameters the new package is created and passed on to the test execution module. A supporting service called process monitor (PM) is been designed for the continuous data logging. It is a customizable background memory logger which can trace the memory usage on the specified interval. Default set to .5 seconds. It runs in parallel whenever a memory test case is executed and communicates with the test execution module.

Test Execution Module

This module directly interacts with the device under test (DUT) via IP Over usb. The IP over USB feature allows to connect a PC to a phone's network for a direct connection between the PC and phone. This feature is typically used for transferring files or testing programs. It deploys the app and the test packages on the device and takes care of the test execution for both the performance and memory test cases.

Logger

Logging is one of the core parts of any test automation framework or tool. Apart from the test case results, it should log more detailed information about the test execution and how the system behaved. On top of that the tool must log what it is doing internally, to make it easier to debug problems in the tool itself.

The multiple level logging helps in controlling the information been captured during the execution. Level 1 enables the logging at lower level, while the level 4 captures the test results and reports only. The intermediate levels are capable of capturing the information about the incorrect test environment setup, warnings and debug info.

As already discussed about the need of continuous logging (Section 3.5.1), this logger is been designed to capture the memory usage on continuous basis. This is further been used to generate the memory test reports showing the trends.

Test Data Parser

Its task is processing the output i.e. test result data and forwarding it to the reporting and DB script. The reporting module handles the visualization of the test data and is shown in the form of the test reports. While the DB scripts helps in storing the parsed data into the DB. This stored test results can later be retrieved or can further be used by other reporting systems within the organization. Test data parser is the heart of the reporting system. It further processes the data to convert in different formats.

Reporting

As discussed earlier, test logs have all the information from starting from the test execution but, they are lengthy and not good for seeing test status at a glance. Test reports provides a concise view of the test results. They provide statistical information about the complete test execution. A good test report helps in catering all the stakeholders from test mangers to the developers. Test report should have a summary about the list of executed test cases with their status, along with the info on below fields (Section 5.1.3). Like,

- Name and version of the app
- Device configuration
- OS & Firmware details.
- Total number of passed and executed test cases, etc

Test reports can be either created at the same time when tests are run or they can be constructed based on test logs afterwards. This reporting system is also capable of fetching the reports for the past six months. The reports can be exported in different formats like pdf, xls and can be published by email.

5.3 Implementation and Piloting of NFT Automated System

This section continues from the previous section and describes how the components of the new system were implemented. The tool is then evaluated based on the pilot experiences in the next chapter.

5.3.1 Technical Decisions

It is very important to choose the right technology to develop and maintain any software, as it makes the life of a developer easier. This section details about the technology been chosen for the tool development and the reasoning behind it.

Implementation Language

Since the NFT automation tool focuses on the Non-functional testing of the Windows Phone app, so it was decided to mainly stick with the Microsoft Technologies. This gives the flexibility for the tool to be used and maintained across the company. The UI of the tool is mainly written in C# and it also provides many good libraries for the reporting purposes.

Since scripting languages comes very handy and are mostly commonly used in the development of the automation tools, so Power Shell was chosen for that purposes. PowerShell is “a task-based command-line shell and scripting language… built on the .NET Framework.” PowerShell can help anyone working in the Microsoft ecosystem and can interact with a dizzying number of technologies.

Storing test results and reports

Now a repository needs to be finalized where the all the non-functional test results, reports and logs can be saved. Since, MySQL was used with the older system and holds all the historic data, so it was decided to be used. Also, some of the dashboards were based on the same Database, so it was good idea to continue using it and define the schema for the new system based on it.

5.3.2 Implementation

NFT Tool interface is very simple and easy to use. As discussed earlier, this tool focuses on providing the automated solution for the non-functional testing, covering performance and memory testing.

Performance management involves measurement of the below factors, as defined above in Section 5.1.2

- installation time
- launching time
- initial responsiveness time
- initial responsiveness after deactivation
- initial responsiveness after closing
- uninstallation time
- if the application is closable from the main menu Measurement of these parameters should be the first step during NFT testing process

Memory Management

In this part we are measuring memory snapshots for UI applications, drivers and processes running on the devices. It is a very important to make sure that particular application does not consume too much memory because of big limitation of total memory size on mobile devices. The test cases are based on requirements defined in Section 5.1.2.

User Interface

NFT Tool interface is very simple and easy to use. The user interface is divided into three main sections:

Configuration view

Test configurator is the test engineer/expert playground. It has been designed in such a way that a test engineer can use it with minimal training or help. It has multiple functionalities, ranging from dummy data generation to test case selection. It is further divided into two subpages.

- **Test builder** (Figure 8) - This is the main page of the tool where the test setup takes place by providing Test builder packages and by setting additional parameters.

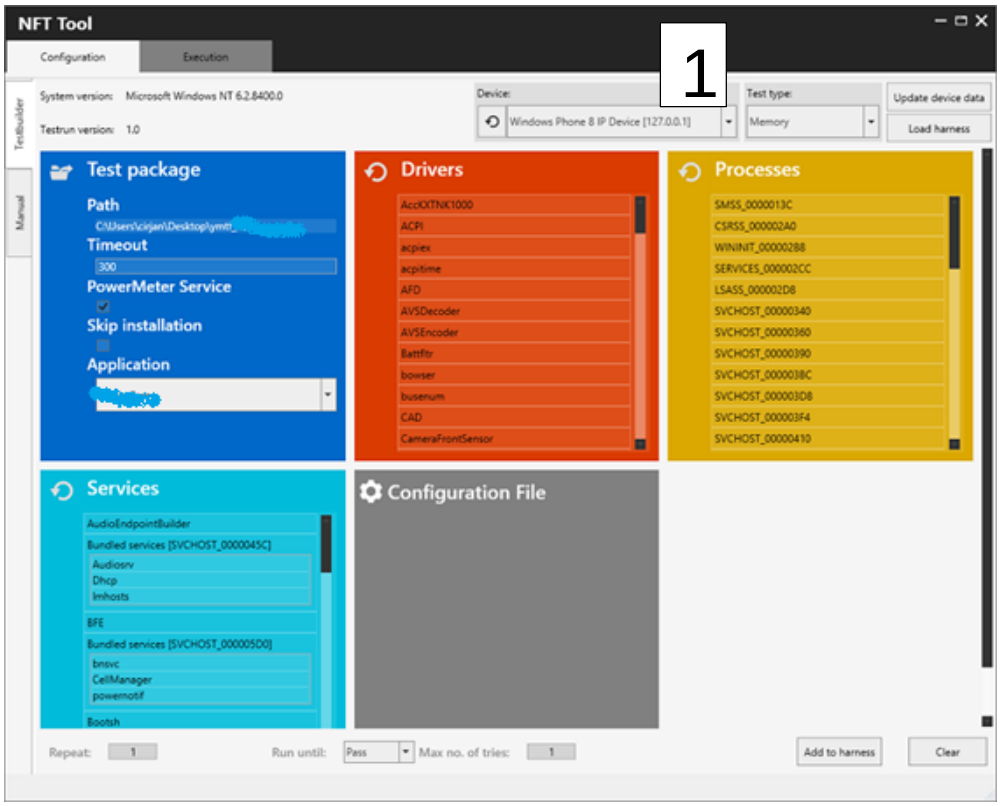

Figure 8: NFT Tool Main Screen GUI

The Test builder view has many different components, which can help the test engineers to design the test case based on their requirements. With the MainScreenDevicesScreenBox [Figure 8-1]**,** test engineers have the liberty to choose the device under test and also select the test type. Here are the different test types:

- **General** - General test type focuses on the performance related use cases. One needs to just provide the test builder package and application XAP file is taken from the Testbuilder package and rest is taken care by the tool.
- **Custom** test - It performs all the tests from the package which is provided and do not make any modifications.
- **Memory** – This test type the NFT Tool is meant for measuring the memory for the applications and the replaces the Tux.Net library by its own specific

version. To be able to complete the tests properly one have to make sure that the tests meet few specific requirements.

- **Manual** - In this test type we are measuring the same parameters as in Memory test. The only difference is that we don't have to provide any automated scenarios. Instead we have to specify the time for the test and perform the actions on the device manually.
- **Composite** - this test type is similar to the Manual test. The only difference is that we are measuring only the memory counters and CPU usage for processes/services/drivers and we don't have to provide any XAP file.

In the test builder view the user has the flexibility to either pass on the existing test cases or create their own test cases. For the existing test cases, a zipped test package is passed onto the tool. The tool then verifies if the package contains all necessary files, for instance, applications XAP file and the configuration. While in case of new test creation, the test engineer can select the process or services listed on the tool UI. The user then selects the number of repetitions for the test and maximum number of times the test case are repeated to get the specified result. Each repetition is displayed separately in the recent results list.

Test Data generator

Test data generator view (Figure 9) can be divided in two sections. The left section allows the user to fill the ram memory by providing the desired percentage of available memory. The right section allows the user to fill the physical memory by providing the desired percentage and the type of files which should be copied on the devices. The filled memory can also be freed by clicking the "Release Memory" button corresponding to each section.

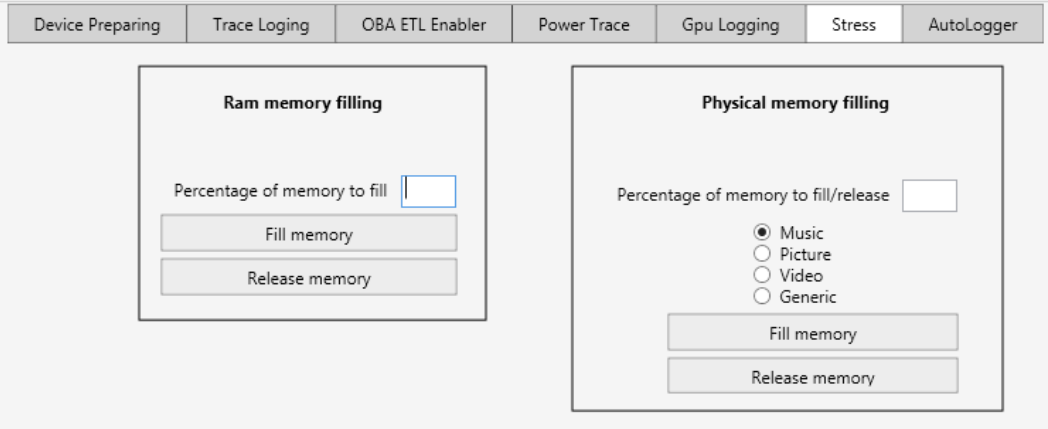

Figure 9: Test data generator

Execution view

In this view test experts can check the test execution progress and list of previous results. Color of the results indicates if the particular test harness is passed or not. Here is the classification for the color codes:

- **green** - all test cases in the test harness passed
- **yellow** - one of few test cases in the test harness failed, some passed
- **red** - the whole test harness failed
- **gray** - NFT Tool was unable to get the results from the device**.**

Execution view is divided into 3 sections.

Execution Queue section contains queue of test harnesses (packages) to be executed. Tests are executed in FIFO order, from bottom to the top of the queue. The test harness can also be removed from the queue.
During the execution all the buttons are disabled, so the test harness cannot be removed from the queue. The executed test harness can exported and later can be used from the Test Builder screen.

Execution Status view helps to check the execution status of the current test harness. The progress bars on this view, show the timeout from the pre-defined value, and also indicates if the execution is still running. While the test information are also shown about the current state of the execution.

Recent results

When the test harness is completed the results are displayed in "Recent results" box. The results are displayed from the newest on the top to the latest on the bottom of the list. The results are grouped by the application.

Figure 10 shows an example of the test results from the new automated tool, where test results are grouped by the app names.

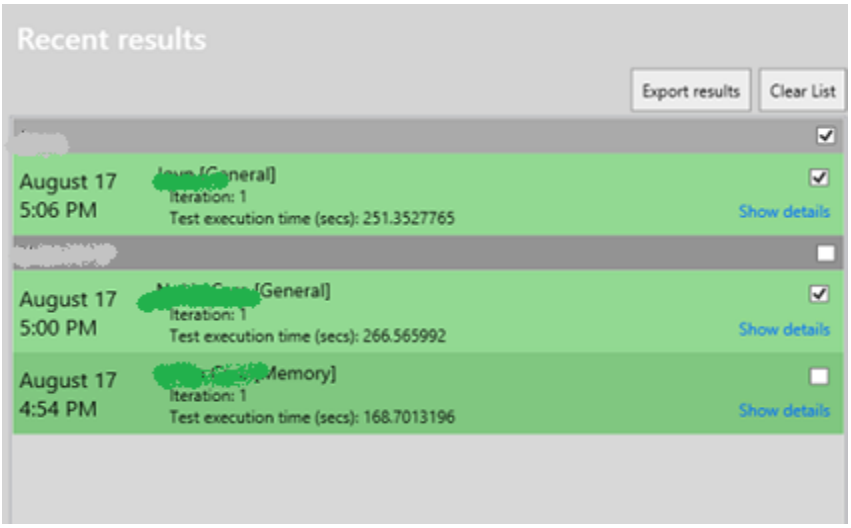

Figure 10: Recent results window

Detailed Test Results window

Detailed test results windows, as all other views in NFT Tool is divided into few main sections. Each of them displays logically divided information about the test cases.
This view changes dynamically depending on the selected test case or on the test harness results.

Test Results Basic Info

This section shows the basic information about the test execution along with the test results, for instance it covers the pass/fail count and the performance test results. The example below (Figure 11) shows the results for the Performance Test Case [General].

Figure 11: Performance test results

Test Case List

This windows lists down all the executed test cases for a test harness and provides the functionality to export the test reports. The report can be exported in four different formats:

- XML with graphs (graphs are stored in separate directory)
- XML without graphs
- PDF
- WRT (Web Reporting Tool) format

This window also helps in customizing the test reports by allowing to choose different memory counters and their units.

Details View

This window has multiple tabs covering the test results in depth, ranging from the summary to detailed scenario graphs. Different tabs in this view are detailed below.

Summary Tab helps in checking the peak values of each memory counter for the process or service under test for each test case. Rows are grouped by the process name.

Logs tab contains XML logs received during the test execution. It is very helpful for debugging, as it contains the test steps of each the test case and list of errors if something went wrong.

Counters tab allows to draw the graphs for drivers and processes counters. One can select many processes and drivers in the same time and many counters. All the graphs will be drawn and grouped by the driver/process name or by the test case name depending on the selection.

Memory tab is displayed only for the memory and manual test cases. On this tab one can check the memory consumption of the UI thread for the Silverlight applications. The memory data can also exported here in CSV data format, to check for the continuous memory consumption.

Figure 12 shows an instance of the detail test results view with the memory graphs.

Figure 12: Memory test results

Once the designing and implementation is completed. The next chapter continues from here by collecting pilot experiences together. Based on the results the overall feasibility of the tool can be evaluated and possible changes can be suggested.

6 Measurement and Analysis

The previous chapter explains how the automation tool works. The concept describes how the different components work and how a request is processed.
In this chapter the results of 2 apps, which were picked up for the piloting are compared between the existing system and newly proposed system, which is been developed as part of this research work. Basically the tool is evaluated based on the pilot experience.

6.1 Measuring UI Responsiveness

A gage R & R study was done for both the older system and the NFT automation tool. This measurement how much variability is caused, when different engineers perform the same test repeatedly. Gage R&R measures the amount of variability induced in measurements by the measurement system itself. Then compares it to the total variability observed in the system, to determine the viability of the measurement system. This study was done using three different testers in 5 iterations, with both manual and automated system.

6.1.1 Results from Existing System

As stated earlier, in the previous system all generic performance scenario were manually tested using high speed camera which leads to human error and was time consuming. There is a total variance of about 30.4% when the same person repeats the same task multiple times. Also the variation in reproducibility (different operator measuring the same item) is recorded as 15.9% which is also not within the acceptance level i.e. less than 10%. According to Automotive Industry Action Group (AIAG) guidelines, if the measurement system's variation is less than 10% of process's variation, then it is acceptable.

Figure 13 shows the summary report of Gage R&R Study for the old testing system.

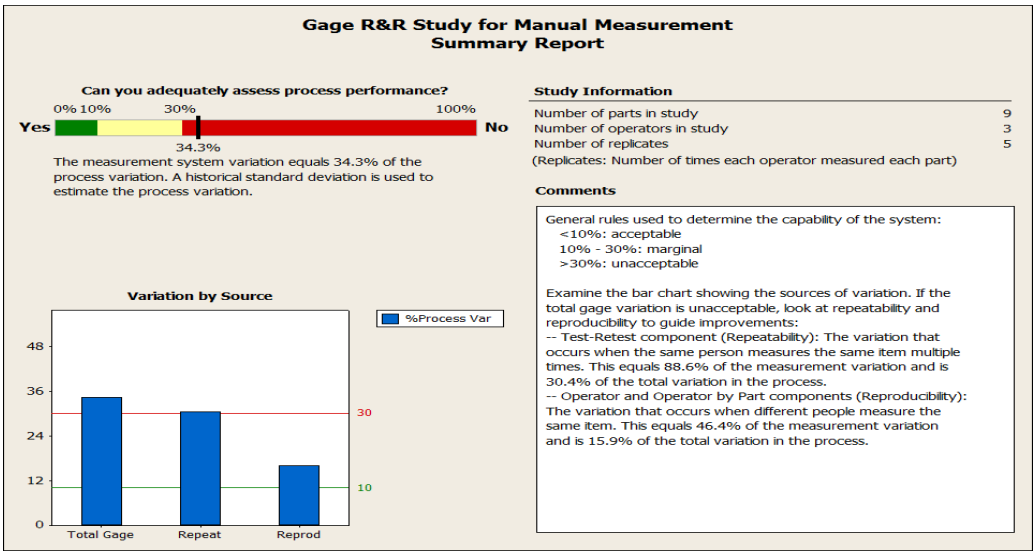

Figure 13: Gage R&R - Existing system

Based on the results in Figure 17 and AIAG guidelines, manual measurement system cannot be considered acceptable with a total GR&R of 27.99 % study variation and 34.5% process variation.

6.1.2 Results from Automated System

The study was conducted with the new automated system, which promises to reduce the test results variation and reduce the test cycle time. There is a total variance of about 0.5% when the same person repeats the same task multiple times with the new automated tool. Also the variation in reproducibility (different operator measuring the same item) is recorded as 0.8% which is well below the acceptance level i.e. less than 10%. According to AIAG guidelines, if the measurement system's variation is less than 10% of process's variation, then it is acceptable.

Figure 14 shows the summary report of Gage R&R Study for the new automated testing system.

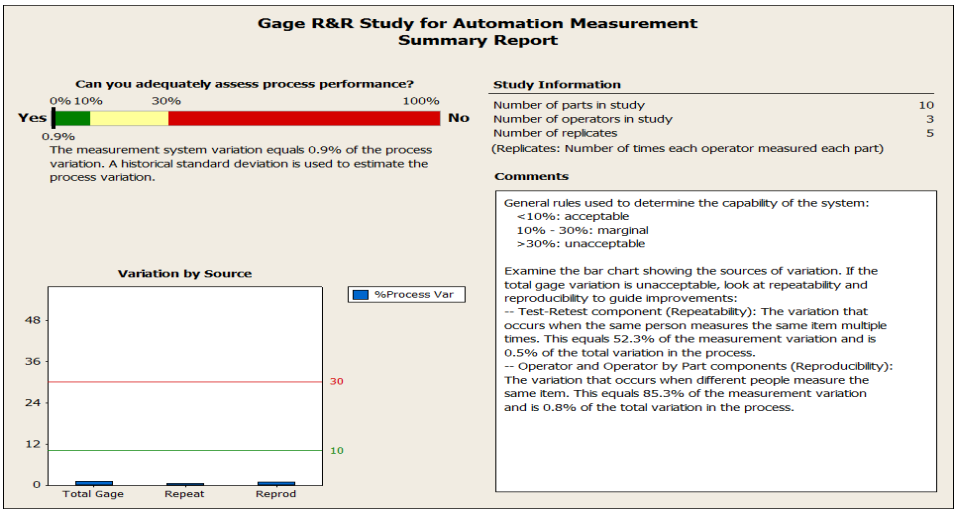

Figure 14: Gage R&R - Automated system

Thus Automated measurement system can be considered acceptable with a total GR&R of 4.7 % study variation and 0.9% process variation, as shown in the test summary above (Figure 16).

6.2 Measuring Memory Consumption

Two studies were done for both the systems to measure if the new automated system scores over the manual system used previously for the memory usage scenarios. Firstly, a normality test was done to check the data is normally distributed. Later a gage R & R study was also done using two operators to check how much variability is caused, when different engineers perform the same test repeatedly.

6.2.1 Results from Existing System

All memory scenarios were manually tested which leads to variation in test results for the same use cases and also leads to high cycle time. In the existing process most of the results are lying between 299 MB to 401 MB. There is a deviation of about 25% (shown in Figure 15), when the same person repeats the same task multiple times, which cannot be considered acceptable for any test system.

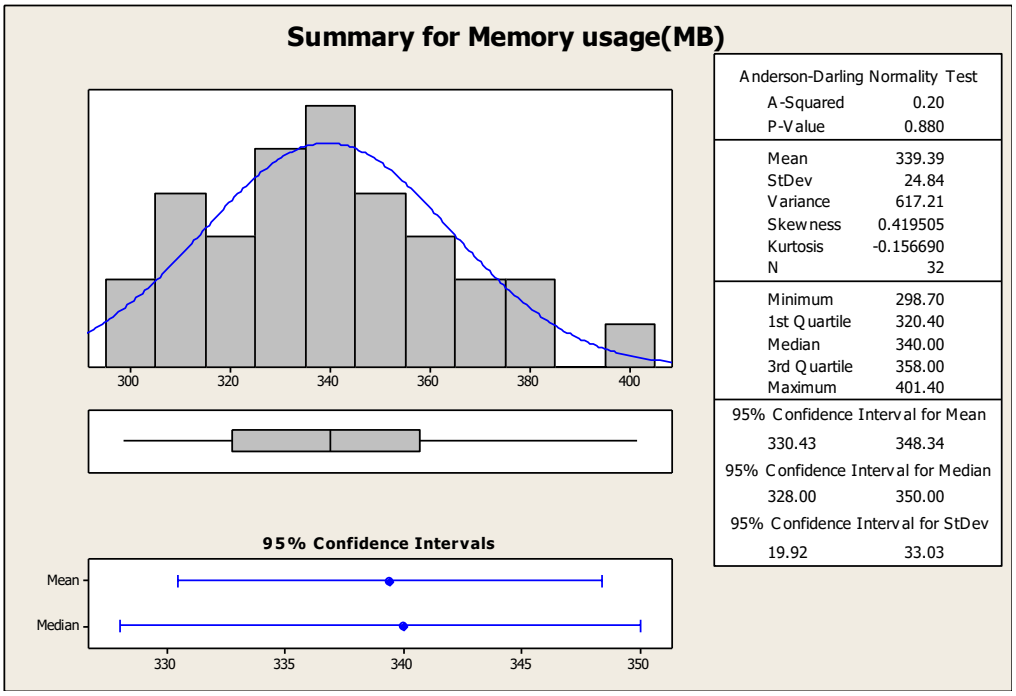

Figure 15: Normality test: existing system

The summary table in Figure 17, shows the minimum and maximum values of the test results along with median. The minimum recorded value is 299 MB while the maximum value is recoded as 401 MB. There is noticeably big amount of variation between the two end points and the standard deviation is recorded as 24.84%. Thus making the test results very unreliable.

Measurement System Analysis-Manual

Another test called MSA (ASTM E2782 Standard Guide for Measurement Systems Analysis), was done to check if the existing system is accurate and stable. By conducting the MSA test, the capacity of the system to produce same results over time can be checked. Measurement Systems Analysis is a key step to any process improvement effort.
Figure 16 gives the overview of the Gage R&R test conducted on the old system for the memory use cases.

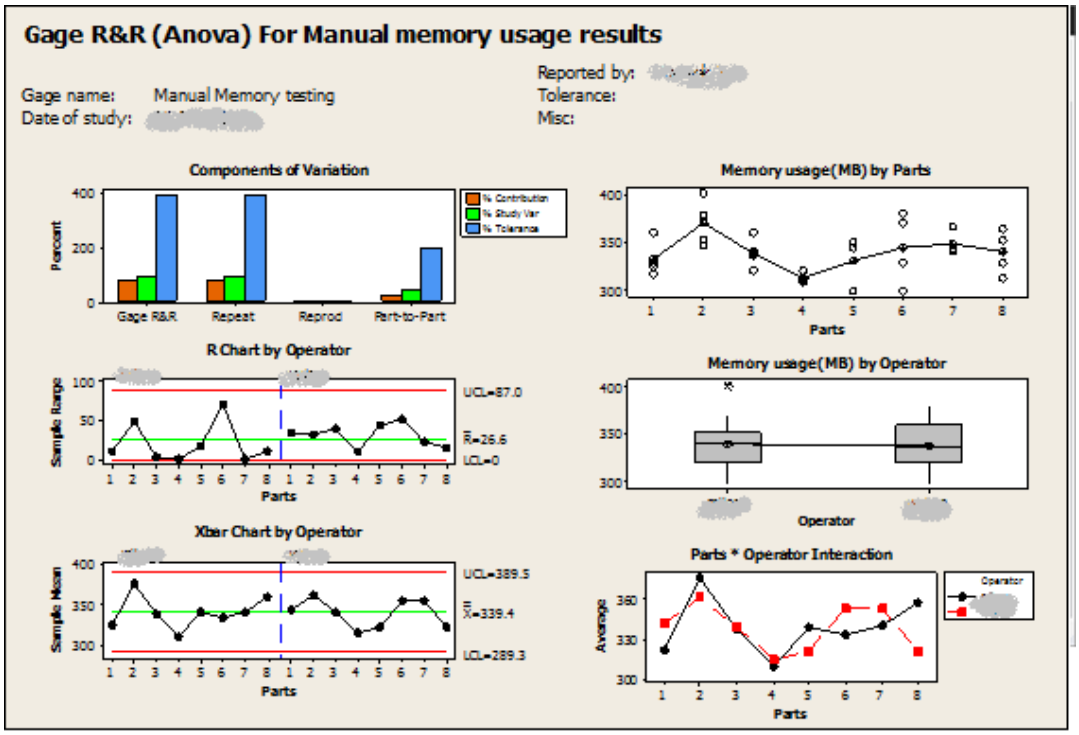

Figure 16: Gage R&R - Manual Process

The measurement results by 2 operators, in multiple repetitions, shows high %Study Variation and %Contribution. So it seems the existing measurement system is not reliable and seems to be the reason for the variance in the test results (as shown in Figure 20 above).

6.2.2 Results from the new automated system

The same study is conducted with the new automated system, which promises to reduce the test results variation and reduce the test cycle time. Figure 17 shows the normality test summary of the new automated test system for memory usage scenarios.

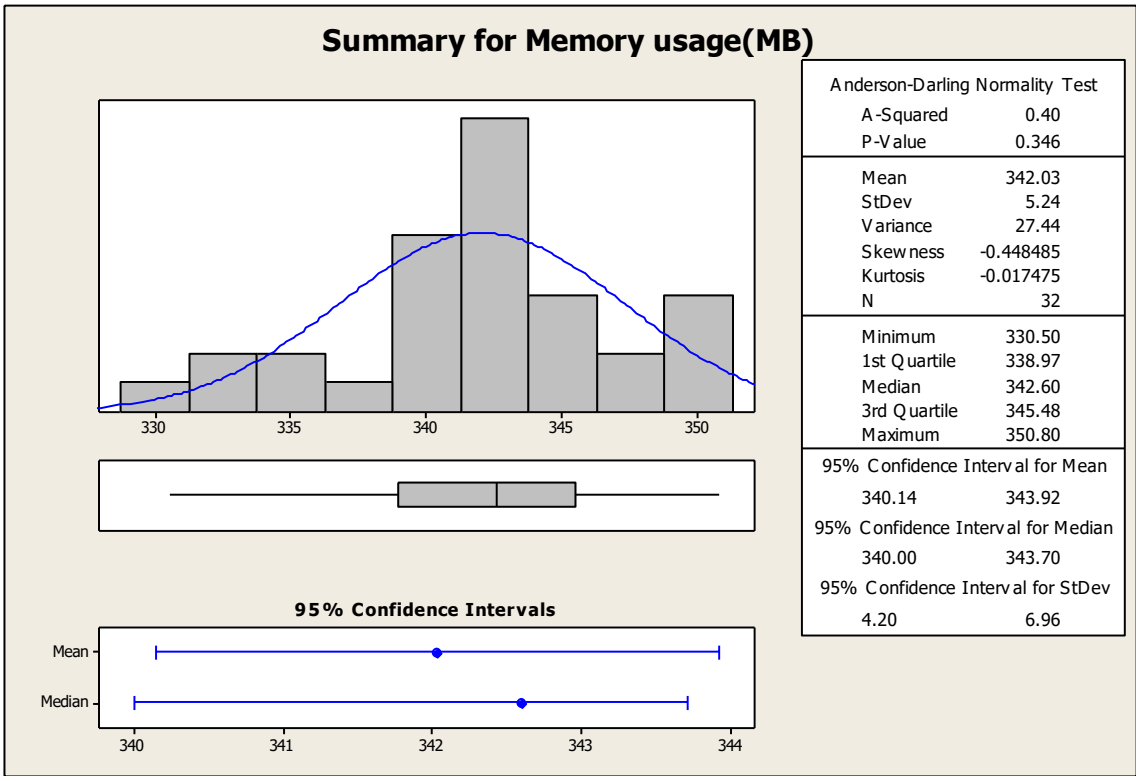

Figure 17: Normality test: New automated system

With the new automated process most of the results are lying between 330MB to 350MB. And the standard deviation is reduced to 5%, which is a significant improvement in the test results variation

Figure 18 gives the overview of the Gage R&R test conducted on the new system for the memory use cases.

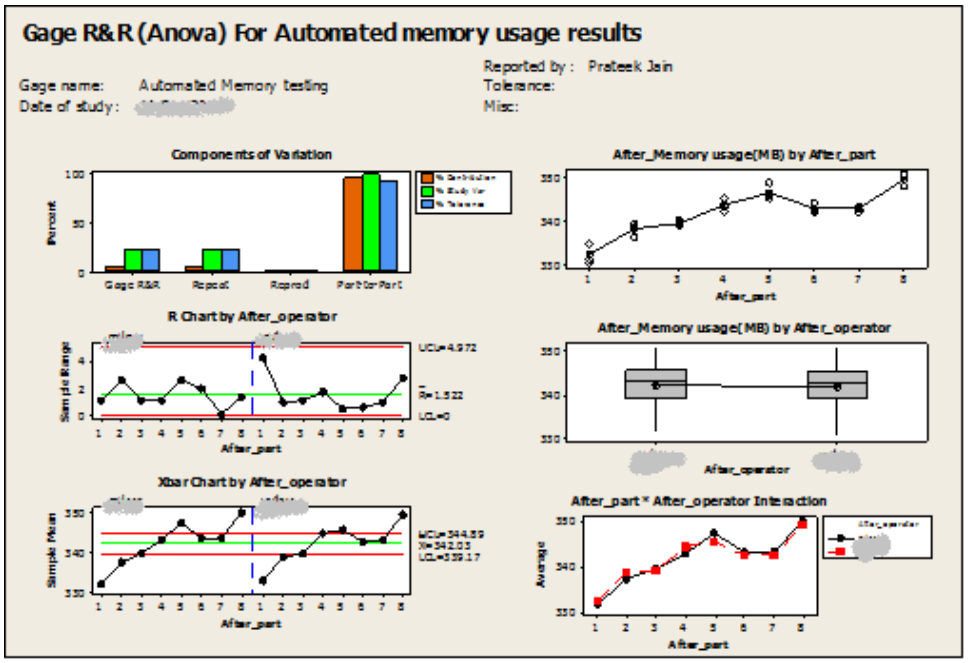

Figure 18: Gage R&R - New automated tool

Also the MSA measurement of the automated system shows the %Study Variation of 22.87% and %Contribution as 5.23%, which can be considered acceptable. If the Total Gage R&R contribution in the %Study Var column is between 10% and 30%, and %Contribution is between 1% and 9%, the measurement system is acceptable depending on the application, the cost of the measuring device, cost of repair, or other factors.

The below comparison (Figure 19) shows there is significant decrease in the variation of the results with the new automated system.

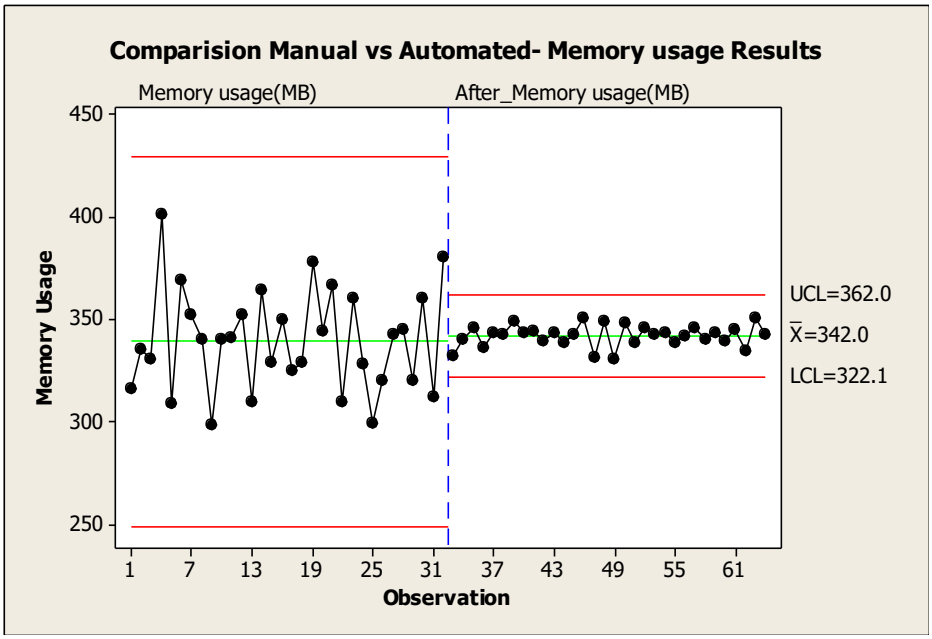

Figure 19: Variance in Test Results - Manual Vs Automated

The standard deviation has been reduced to 5% from 25%. The test results are now in acceptable limits as the delta between upper and lower limit has reduced to 40 MB only.

6.3 Testing Cycle Time

This research started with two main objectives, one to reduce the variation in the results by automated the existing system. Secondly, to reduce the testing cycle time. This section will now focus on measuring and understanding the testing cycle time of the overall process. Figure 20 shows the mean testing cycle time of the existing system for the execution of the test cases was around 80 minutes.

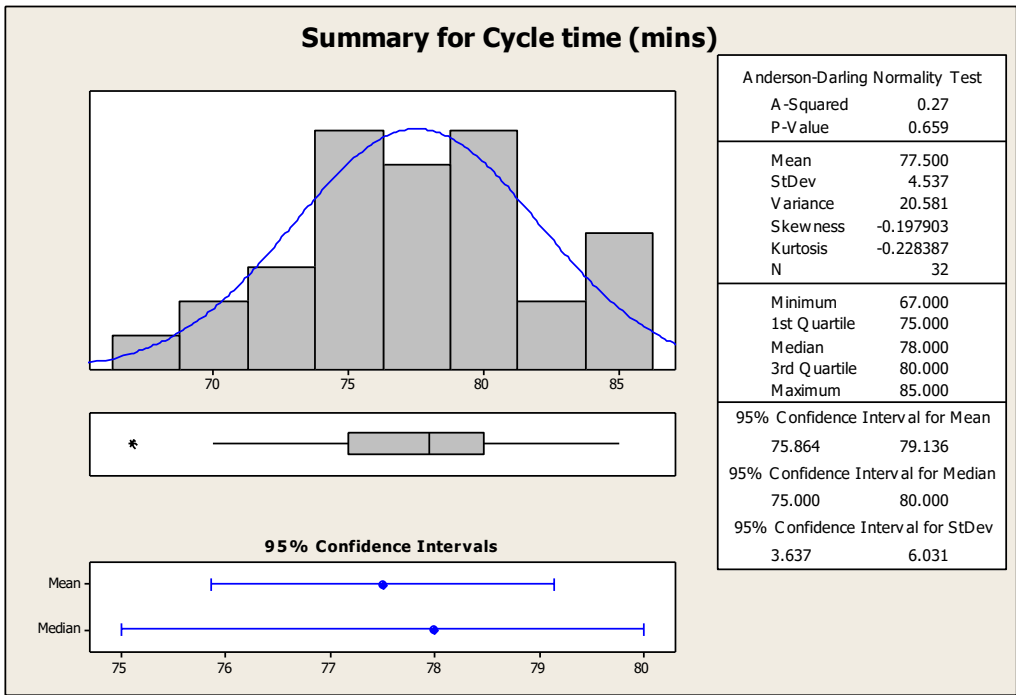

Figure 20: Normality test for test life cycle

But, the new automated system has bring it down significantly. Comparison in Figure 21 shows there is a meaningful shift in the testing cycle time with the new automated system.

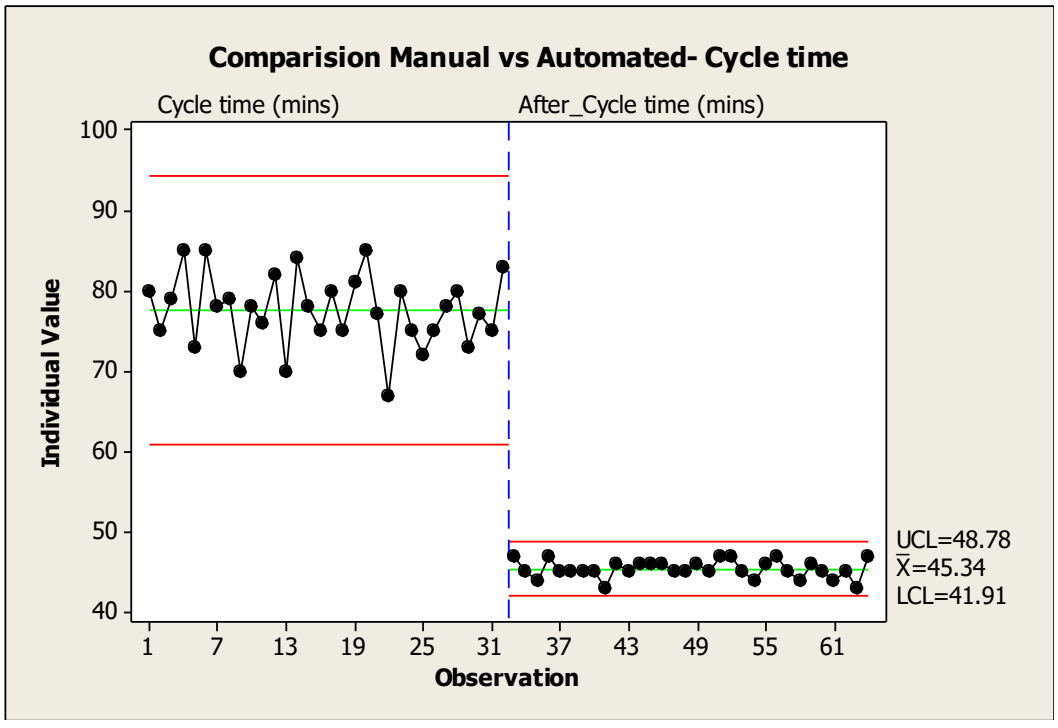

Figure 21: Test cycle time Comparison - Manual vs Automated

Mean testing cycle time has now reduced to 45.34 minutes as compared to 80 minutes from the old testing system. Most of the testing cycle times lie within a range of 41 and 48 minutes. Thus the second goal of this study for reducing the testing cycle time is also achieved.

6.4 Comparative Result Analysis with Similar Test Automation Study

This section presents the comparative study of the test result results for the similar test automation studies. After looking back on this thesis and also at other research works in the same area. It can be summarized that test automation has a clear advantage over manual testing where repetitive tasks are being performed. Thus saving the overall cost of the project.

Similar Study

As an example from the Mater's study "Test automation in Practise" [20] shows the test effort reduction comparison of different test approaches, ranging from no automation to full test automation. It also presents that in many cases 100% automation is not feasible and it does not show significant improvement in cost reductions.

Figure 22 shows the comparison of overall reduction in testing efforts for the same system with no automation, partial automation and full automation. This table is been inherited from a similar study of test automation for the comparison purposes.

	TC_{TEST}	$TC_{TEST\ EFFORT}$	$Reduction_{TEST\ EFFORT}$	TC_{SDP}	Reduction
No test automation	255	7905	-	9825	-
UI test automation	30.6	948	88%	2926	70%
Limited UI test automation	165.2	5122	35%	7100	27%
UI and Unit testing	30.6	336	95%	2794	71%

Table 11 – Reduction by test automation

Figure 22: Reduction in test efforts [20]

In Figure 22, it is clearly visible that there is an overall reduction of 70% in the test efforts between the no test automation and UI test automation.

NFT automation tool

As explained above (in Section 6.3), with the automation of non-functional testing of the mobile application, the mean cycle time is reduced to 45.34 min as compared to 80 min previously. And this is a reduction of almost 44% in the testing cycle time for an application. Apart from this, there is a huge reduction in the deviation of the test results, which was the main objective of the organization, under which this study was conducted. The deviation is reduced by almost 80%, which is a great achievement.

The assessment is positive and the new solution is suggested as acceptable. Even though this tool is been declared successful, but there could still be more improvements done to it. One of those could be adding the support for the Universal apps for Windows Phone 10.

The next section focuses on the summary and some improvement suggestions.

7 Conclusions

The main objective of this section is to look back into the requirements that were set for this study in the beginning and evaluate how well they have been met. Besides that the future of the presented automation tool is also briefly discussed.

Due to the competitive market's demand most of the mobile applications are required to be developed in a short period, which can undermine the application quality. Therefore, it is necessary to undergo a rigorous testing process not only on functional but also on non-functional requirements, especially time limits. Non-functional testing is a large area and its different test areas are discussed in Section 3.3. As for mobile applications, critical performance factors are related to spontaneous interaction, high reliability, stability and low power consumption. While this thesis only concentrates on two aspects of non-functional testing that is responsive testing and resource utilization testing.

In practice, manual testing of mobile device applications is time consuming, expensive and very difficult to do effectively. It could also lead to the huge variation in the test results, which then makes it more difficult to judge the quality of the app and thus increases the risk of bad quality app pushed to the market. Automated testing is attractive essentially because it can reduce the costs and time associated with testing, lead to shorter release cycles and allows developers and testers to focus on constructing effective test cases.

The starting point for this project was a need to automatize most parts of the non-functional testing to remove deviations in the test results and reduce the testing life cycle.

The study of the previous testing system with the basic principles of test automation provided a good base for this project. The initial setup was based on the manual testing which is time consuming and expensive, so a new test automation system needs to be developed. The requirements for the new system were broadly classified into three areas. Firstly, the generic automation requisites covering repeatability, ease of use, maintainability etc. Secondly, requisites from the existing system reduce variance, continuous logging, reporting and so on. Thirdly, requisites from the non-functional perspective covering the performance and memory test definitions. These requirements are explained thoroughly in Section 5.1. Multiple requirements and limitations of the current

system suggests that another system is needed for achieving the reduced test life cycle and accurate results.

The NFT test automation tool is mainly constructed of three parts: test configurator, monitoring system and test executor. Test Configurator is the tester's playground, providing the functionality ranging from test case configuration to dummy data generation. Previously, the tester has to manually copy the test data. During the test case creation user can select the process/service against which the test case has to be executed and schedule it for later execution. The performance test cases are pre-stored and the test engineer can customize the memory related test cases (requirement in Section 5.1.2). The created test cases can be stored and used later, enabling the re-usability and applicability (requirement in Section 5.1.3). The tool UI is simple and easy to use (requirement in Section 5.1.1), one of the basic of test automation.

The test executor is the centrum of the whole tool and it controls testing devices and communicates with the client and the service. The user can also run multiple iterations of the stored test set in the same test environment, eliminating human error and reducing the variance in test results (requirement in Section 5.1.3). It enables the reproducibility and repeatability (requirement in Section 5.1.3). It also takes care of continuous logging (requirement in Section 5.1.3) and collects results at the end of each test cycle. After each execution the results were stored separately. While in the previous system everything was done manually. The tool software is coded mostly with C# and PowerShell. For test cases there are multiple helper functions created to make writing even easier, enabling maintainability (requirement in Section 5.1.1) and adding new features to the framework. The tool software is coded mostly with C# and PowerShell, which is used commonly within the organization.

The test monitoring system is used for controlling test execution and checking test results. The test logs and results are now presented in a readable and graphical format, which is much richer than the plain text reports. These test reports covers details of application under test, DUT, OS details and test summary(requirement in Section 5.1.3) Test results can stored in different formats and can be exported/retrieved later on. This enables the re-usability and ease to use (requirement in Section 5.1.1, 5.1.3).

Now we have automated test system which takes the application's XAP as an input and produces the detailed test report for the desired area. The new NFT test automation tool was piloted on 2 apps and the results discussed (in Section 6), explains what is been achieved by the new automated system. The results are similar for automated and manual testing with reduction in the variation of test results. The variation in the test results have been reduced by 80% for memory and 85% for performance test cases. Also, the testing cycle time has come down to 45 minutes from 80 minutes. With this the system was verified to fit for use for non-functional testing of the windows phone 8 applications. Thus meeting all the requirements set during this project.

Even the NFT test automation tool is already in active use there are still some development on going. There are lots of improvements to be done. At the moment it does not support the windows phone 10 applications. However, in future it would be the next step to take this work forward and enable it for windows phone 10 applications. It can be further extended in other important non-functional testing areas like stress and reliability, network communication delays affecting the overall application responsiveness etc. Since the tool already had the test data generator, it can also be used to stress the device under test (DUT) by increasing the memory and CPU pressure and thus checking how the apps behave under such situation. Application performance under the realistic network conditions is also of the major areas. The application may behave differently on different networks as network protocols impact throughput and delays. The above mentioned (two) areas are the perfect candidates to be added to this tool. The hope is that in the near future it is expanded and used in other NFT areas.

References

[1] Windows Phone API reference [online]. http://msdn.microsoft.com/enus/library/windowsphone/develop/ff626516(v=vs.105).aspx ; 2014. Accessed on April 2014

[2] Technical certification requirements for Windows Phone [online].

http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh184840(v=vs.105).aspx; 2014. Accessed on April 2014

[3] App memory limits for Windows Phone 8 [online].

http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj681682(v=vs.105).aspx; 2014. Accessed on April 2014

[4] Windows Phone Application Analysis for Windows Phone 8 [online].

http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202934(v=vs.105).aspx; 2014. Accessed on April 2014

[5] Andrew Whitechapel, Sean McKenna; Windows Phone 8 Development Internals. Chapter 1. Vision and architecture; 2013. Accessed on April 2014

[6] Heejin Kim, Byoungju Choi, Seokjin Yoon. Performance Testing based on Test-Driven Development for Mobile Applications [online]. Suwon, S. Korea: International Conference on Ubiquitous Information Management and Communication; January 15-16, 2009. p.612-617.

[7] Heejin Kim, Byoungju Choi, W. Eric Wong. Performance Testing of Mobile Applications at the Unit Test Level [online]. Shanghai, China: Third IEEE International Conference on Secure Software Integration and Reliability Improvement; July 8-10 2009. p.171-179. URL: http://ieeexplore.ieee.org.ezproxy.metropolia.fi/stamp/stamp.jsp?tp=&arnumber=5325380. Accessed on May 2014

[8] Jiang Bo, Long Xiang, Gao Xiaopeng. MobileTest: A Tool Supporting Automatic Black Box Test for Software on Smart Mobile Devices [online]. IEEE:Second International Workshop on Automation of Software Test; 2007.

[9] Sakura She, Sasindran Sivapalan, Ian Warren, Hermes. A Tool for Testing Mobile Device Applications [online]. Gold Cost, Australia: Australian Software Engineering Conference; 2009. p.121-129.

[10] Kirubakaran, B., Karthikeyani, V. Mobile application testing — Challenges and solution approach through automation [online]. Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference; 21-22 Feb 2013. p 79-84. URL: http://ieeexplore.ieee.org.ezproxy.metropolia.fi/xpl/articleDetails.jsp?arnumber=6496451.

[11] M. E. Delamaro, A. M. R. Vincenzi, J. C. Maldonado. A Strategy to Perform Coverage Testing of Mobile Applications [online]. Shanghai, China: AST; May 23, 2006. p 118-124.

[12] The company's intranet. Non-functional testing - Wiki. Internal Wiki. [Online] Accessed on Apr 2014.

[13] Kristoffer Dyrkorn, Frank Wathne. Automated Testing of Non-functional Requirements [online]. Nashville, Tennessee, USA : OOPSLA'08; October 19–23, 2008. p 719-720. Accessed on

[14] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, A GUI Crawling-based technique for Android Mobile Application Testing, Software Testing [online]. Naples, Italy: Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference; 21-25 March 2011. p 252 – 261. URL: http://ieeexplore.ieee.org.ezproxy.metropolia.fi/xpl/articleDetails.jsp?arnumber=5954416.

[15] E. Kit. Integrated, effective test design and automation. Software Development, pages 27–41, February 1999

[16] Nagle. Test automation frameworks, 2000. URL http://safsdev.sourceforge.net/DataDrivenTestAutomationFrameworks.htm. February 20, 2005

[17] https://www.minitab.com/en-us/products/minitab/

[18] E. Dustin, J. Rashka, and J. Paul. Automated Software Testing. Addison-Wesley, 1999.

[19] M. Fewster and D. Graham. Software Test Automation. Addison-Wesley, 1999.

[20] Test Automation in Practice. Master's Thesis, Delft University of Technology Delft, the Netherlands. Kishenkumar Bhaggan BSc. 2009 p 47-54

[21] http://www.gartner.com/newsroom/id/2654115