Performance Impact of Fuzz Testing Windows Embedded Handheld Applications

Nizam Abdallah[1] and Sita Ramakrishnan[2]
[1]*Software Test Automation Research and Consulting, Oakleigh South, Victoria, Australia*
[2]*Faculty of Information Technology, Monash University, Clayton, Victoria, Australia*

Keywords: Fuzz Testing, Random Testing, Automated Testing, Mobile Applications, Windows Embedded, .Net Reflection, Gui Testing, Performance.

Abstract: Mobile systems are increasingly impacting our personal and business lives. It is crucial that we develop mobile software applications that are robust, efficient and deliver value to a wide range of users. As a result, appropriate software testing methodologies need to be employed during the development of these mobile applications to ensure high level of quality and robustness. Software test automation methodologies and fuzz testing techniques have proven to be successful in finding defects during the development lifecycle. However, due to the fact that mobile devices are resource constrained devices with limited memory and CPU, there are performance constraints that need to be considered when developing a test automation framework for mobile devices. This research introduces Torqueo. Torqueo is an automated fuzz testing tool designed specifically to target Windows Embedded Handheld GUI applications. It is capable of interacting with GUI applications using either Win32 API or .NET reflection API, and it is also capable of executing test scenarios from pre-generated test data and randomly generated test data at run time. The experiments described in this paper discuss the performance impact on memory usage of invoking GUI controls using the Win32 API vs. .NET reflection.

1 INTRODUCTION

Today, many software systems are ubiquitous and as a result the complexity of designing, implementing and testing these systems has increased. As people interact with computer systems throughout their daily lives, it is expected that these systems operate safely, correctly and are robust against all types of users and the varying types of user input data these systems are accepting. As the complexity increases, developers need to employ suitable testing techniques that allow them to test these systems effectively and efficiently. Two areas of testing that have proven useful in finding defects in software systems include automated testing (Alsmadi, 2008) and Fuzz testing (Sutton, Green, Amini, 2007). This paper looks at the design and implementation of an automated fuzz testing tool named Torqueo and the performance impact it had on a Windows Embedded handheld device when used to automatically generate and execute tests using Win32 API and .NET Reflection.

The remainder of this paper is divided into five sections. Section 2 discusses the background and motivation for conducting this research and developing Torqueo. Section 3 discusses Torqueo's features. Section 4 describes the experiment conducted to determine the performance impact of executing GUI based tests using two GUI invocation techniques, Win32 API and .NET Reflection. Finally Section 5 details the how Torqueo has been used on real-world applications and outlines future enhancements.

2 BACKGROUND

People interact with computer systems (including embedded systems) many times throughout their daily lives. These interactions include, but not limited to, the alarm clock, radio, mobile phone, GPS navigation system, car, DVD player, television, reading information from an electronic billboard, using ATMs, and personal computers. As these systems continue to impact our daily lives, there is an expectation that these systems be robust, safe and operate correctly. As a result we need to ensure that development and testing methodologies used are also robust, but at the same time, efficient and effective.

Windows Embedded is a set of technologies provided by Microsoft for the development of embedded software systems (Microsoft, 2010). There are several operating systems available to allow developers to develop software including:

- Windows Embedded Compact 7
- Windows Embedded Standard 7
- Windows Embedded Automotive 7
- Windows Embedded POSReady 2011
- Windows Embedded Handheld 6.5
- Windows Embedded Enterprise
- Windows Embedded NavReady
- Windows Embedded Server

As can be seen from the list above; the operating systems available support a large range of devices and device form factors. This research focuses primarily on the Windows Embedded Handheld 6.5 platform.

Important considerations need to be made when testing applications targeting embedded devices, including; connectivity, screen size, different mobile platforms, interruptions and notifications (e.g. phone calls, SMS, Email or system messages such as low battery warnings), power constraints (e.g. battery) and different input types (e.g. stylus, buttons, touch screen, multi-touch, gestures, voice).

Embedded handheld devices are generally resource constrained devices. Examples of these devices include; mobile data terminals, mobile phones, global positioning systems (GPS) and personal digital assistants (PDA). Testing for these devices can be difficult as there is very little test tool support for such devices on the market today. Furthermore testing tools are generally intrusive and do impact the performance of the AUT and therefore may severely impact the performance of an already constrained device.

GUI test automation tools have been out for many years now and in many cases have had a positive impact on finding defects in software systems (Alsmadi, 2008). GUI test automation has also introduced efficiencies in the testing process, by performing the more mundane and boring tasks, while allowing testers to use their skills on more challenging analysis tasks. There are only a limited number of tools available on the market today that allow for GUI automation testing for Windows Embedded Handheld applications.

There are several ways in which a developer can programmatically invoke Windows Embedded GUI applications controls and hardware buttons. Two of these involve the use of calling functions from within the Win32 API and also using .NET reflection (Abdallah, Ramakrishnan, 2009). The advantage of using .NET reflection is that it uses metadata to obtain the application's properties at run time allowing developers to observe these properties' values. However there is an associated performance impact when using .NET reflection. Part of this study involved an experiment in determining the impact of .NET reflection on .NET Compact Framework applications. During the experiment, it was discovered that the performance impact associated with .NET reflection is not always present and is only noticeable when .NET reflection is used to invoke simple non-complex routines (Abdallah, 2010).

Fuzz testing is a testing technique that was introduced in 1990 (Miller, Fredrikson, So, 1990) that involved sending random data input to a program and determining if the program can handle these random inputs without crashing. Since then many new fuzz testing tools and frameworks have been developed and released on the internet as open source. These tools are written in various programming languages and target different areas of an application and different platforms. Some popular open source fuzz testing tools include; Spike, Zuff, Protos and Web Scarab. In addition to these open source tools, companies are also starting to release commercial fuzz testing tools and frameworks. One example of such a tool is Codenomicon's DEFENSICS testing platform (Codenomicon, 2010).

There are many different types of fuzz testing tools available. Some tools are generic fuzzing tools that send random input to an application, while others are developed to test a specific feature or technology used by the software application under test. The list below shows the different types/categories of fuzz testing tools available (Sutton, Green, Amini, 2007).

- Local Fuzzers
  - Command line
  - Environment Variable
  - File Format
- Remote Fuzzers
  - Network Protocol
  - Web Application
  - Web Browser
- In Memory Fuzzers

Fuzz testing tools are either mutation-based or generation based tools. (Sutton, Green, Amini, 2007) describe mutation-based fuzzers as tools that "apply mutations on existing data samples to create test cases" and also describe generation-based fuzzers as tools that "create test cases from scratch that model the target protocol or file format". In addition to this, previous research has been conducted that successfully demonstrated different ways to generate test data for random testing. These methods include the use of adaptive random testing (Chen et. al, 2009) AI planning (Memon, Pollack, Soffa, 2000) and Artificial Neural Networks (Ye, et. al, 2006).

Fuzzing has been successfully used on different types of applications, including UNIX command line applications (Miller et. al, 1995), web applications (Hammersland, Snekkenes, 2008), Windows GUI applications (Forrester, Miller, 2000) and MAC OS GUI applications (Miller, Cooksey, More, 2006). Microsoft released a free tool called Hopper that automatically generates simulated button presses and stylus input for Windows Embedded Handheld Applications (Microsoft, 2010). Motorola have also been successfully using similar random testing techniques to test their Windows Embedded Handheld based mobile phones (Chong, 2006). Google have also released a GUI based fuzz testing tool for the Android platform named monkeyrunner (Google, 2012). The monkeyrunner framework is an extensible framework for the Android platform that can be used to tests the robustness of an Application.

3 TORQUEO

This research involved the development of a fuzz testing tool named Torqueo. Torqueo is specifically designed to target Windows Embedded Handheld applications and is capable of testing both managed (.NET) and unmanaged (Native) GUI applications. Torqueo is a local fuzzer that supports both mutation-based and generation-based fuzzing methodologies as it is capable of generating test input data randomly at run time and execute a pre-generated set of tests.

There are two ways in which Torqueo interacts with GUI applications on a Windows Embedded Handheld device. The first is using the Win32 API to simulate button presses and stylus input. The second method involves using .NET reflection to record the GUI properties of all controls in the AUT. Using simulated button presses and stylus input is advantageous because it accurately simulates the actions the user will be performing and the application will follow the same execution path. However, when using the Win32 API to simulate button presses you need to make sure you first simulate a stylus input at the correct co-ordinates to set focus to the required control before simulating button presses. This requires the developer to have knowledge of pre-set co-ordinates of all GUI controls they wish to interact with.

The alternative method is to use .NET reflection to interact with GUI controls. This will allow the developer to accurately invoke the control required as .NET reflection records all GUI properties at run time. However the potential issue with this is that using .NET reflection to generate GUI events does not accurately simulate user input, because you are directly invoking an event handler rather than the GUI control on screen.

In addition to randomly generating co-ordinates and selecting controls, Torqueo also has additional features as shown in Table 1.

Table 1: Torqueo Features.

Feature	Description
Playback	Play back pre-created actions
Random Input	Generate random stylus and key press events using Win32 or .NET Reflection
Scripted Input	Execute XML based tests and test commands
Phone functionality	Make phone calls and send SMS
Data connections	Initiate a data connection/browse internet
Database IO	Create/Read/Write to SQL Server CE database
File IO	Read/Write to a file
Performance tests	Log CPU and Memory Usage

Torqueo is an application that can run on any Windows Embedded Handheld device that has the .NET Compact Framework 3.5 installed. All tests, results and settings are selected, executed and stored on the device itself. This allows for easy setup and a very quick way to begin testing mobile applications.

Torqueo logs all tests and test data used during any test session. This allows the developer or test analyst to investigate the cause of an issue by looking at the last set of actions that were executed. Furthermore the log file can be used to create XML test scripts or test commands and allow the exact same data to be re-run in future test sessions. While re-running tests from the log file ensures that the same test actions and test data is executed, it does have a limitation where a particular defect will only be found in the same area of the application and not other areas. For example, if interacting with a particular GUI control causes the AUT to crash, the crash will not be found in another area of the application, unless that same control is randomly selected on another form. To overcome this limitation, an algorithm has been implemented to allow Torqueo to record, not just the test action and test data such as co-ordinates used to interact with GUI controls, but also the type of GUI control that was invoked. This algorithm then stores the GUI control invocation sequence and attempts to invoke this same GUI control sequence in other forms in the application. The length of the sequence is determined by a sequence-length variable in a configuration file.

Torqueo allows the user to select the application under test and also allows the user to select if they want to run random tests, scripted tests or both. Figure 1 shows an example of this functionality.

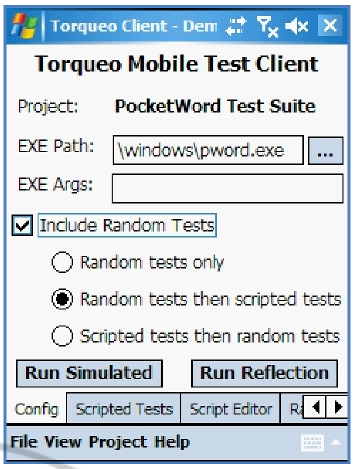
When selecting test scripts to run as part of a test, the user can also select the order in which these tests are executed and also the number of times each script is executed. In addition to selecting a set of scripts to run, XML scripts can be modified and saved on the device using the XML editor. Several settings can also be adjusted for the random test functionality as shown in Figure 2. This includes setting the area of the screen for which simulated stylus input is generated and the also allows the user to exclude certain keys and key combinations from being executed. The user can also add time constraints on the duration of a random test and the interval at which random input is generated.

4 PERFORMANCE IMPACT

As can be seen from the previous section Torqueo has several features, some of which require significant processing and memory when executed on a device. An experiment was conducted to determine the performance impact of Torqueo on a mobile device when being used to test a GUI based application. To ensure that the application under test performs several functions common to mobile applications a custom based GUI application was created for this experiment that consists of multiple GUI forms, SQL Server Compact I/O, XML File I/O and File I/O

This custom application also records statistics to determine how often a particular control was invoked. The application under test was conducted on a single Windows Mobile device. The chosen device was the Motorola MC35.

The experiment involved launching the application under test and randomly invoking the GUI controls and simulating keyboard buttons for 15 minutes. This was repeated 10 times for each GUI form within the custom application. Once a single test was completed three logs were retrieved from the device and the device was then rebooted to



Torqueo Mobile Test Client

Project: PocketWord Test Suite

EXE Path: \windows\pword.exe

EXE Args:

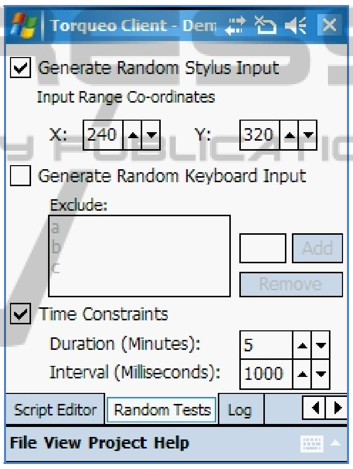
[✓] Include Random Tests

○ Random tests only

● Random tests then scripted tests

○ Scripted tests then random tests

Figure 1: Torqueo mobile client main screen.



[✓] Generate Random Stylus Input

Input Range Co-ordinates

X: 240 Y: 320

[ ] Generate Random Keyboard Input

Exclude:

a
b
c

[✓] Time Constraints

Duration (Minutes): 5

Interval (Milliseconds): 1000

Figure 2: Random test setting.

ensure that memory was cleared so that the device is in a similar state for all tests executed. The logs files retrieved included; the .STAT file, device client log file and the log created by the custom application under test. The .STAT file is a file that is generated by the .NET Compact framework after executing a .NET application on a mobile device. This file is only generated if the following registry value is set to 1.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETCo
mpactFramework\PerfMonitor
```

The .STAT file contains many performance indicators that were used to assess the performance of the test tool running on the device, the primary indicator used during this experiment was the *Peak Bytes Allocated* value. This value is the peak bytes used by the .NET runtime during the execution of a .NET application. The device client log file includes the type of action executed and the data used during each action performed. The log file of the application under test lists the name and type of each control that was invoked and also includes the input type that was invoked by the device client.

Table 2: Average Peak Bytes Allocated.

Standalone Client - Average Peak Bytes Allocated		
Form	**Win32 - Bytes Used**	**Reflection - Bytes Used**
FrmLogin	3062056	2643630.4
FrmMain	2748208	2566116.8
FrmCustomers	2295646.4	3056333.6
FrmProducts	2339048	2509143.2
FrmOrders	2293992	2163578.4
FrmAbout	2293992	2143530.4

The results of the experiment shown in Table 2 and Figure 3, demonstrate that using Win32 API to automatically invoke GUI controls takes up more memory except when executing forms that perform a significant task such as writing to a file or writing to a database, which is exactly what the forms FrmCustomers and FrmProduct do. The other forms (FrmLogin, FrmMain, FrmOrders and FrmAbout) do not perform any computationally expensive processing. The reason that the memory usage of the device client increases when using .NET reflection is because when instantiating a windows form using .NET reflection, the AUT is bound as part of the application that loaded the form. When using Win32 functions to invoke GUI controls, the AUT under test is loaded as an external process, and therefore has no direct performance impact on the device client. This was observed in the *Controls Created* fields in the .STAT files for the Win32 tests and .NET reflection tests.

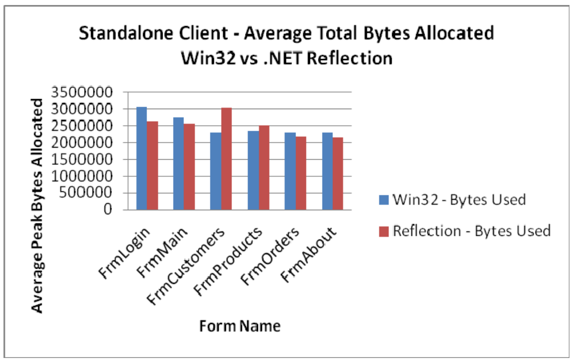


Figure 3: Average Peak Bytes Allocated Graphical Representations.

5 DISCUSSION & FUTURE WORK

Torqueo is an automated fuzzing framework that can generate tests for Windows embedded handheld GUI applications; it can be executed on multiple device configurations and can be used in conjunction with existing testing tools. The research conducted demonstrates that there is a performance impact when automating the invocation of GUI controls using both the Win32 API and .NET reflection. However the results demonstrated that the technique better suited to perform this task depends on the complexity of the operations the GUI is attempting to perform. While there are benefits to having a stand-alone version of Torqueo on a test device, there are also some disadvantages especially when testing using multiple devices as the effort to retrieve logs and configure test scenarios is increased. A client / server version of Torqueo is currently being developed that requires a lightweight agent to be installed on the device and all settings are controlled using a desktop computer as opposed to having all features of the testing installed on the device under test. The initial results have shown a performance improvement on the device.

The stand-alone version of Torqueo has been used to successfully find defects in commercial enterprise mobility applications. The types of defects found were mainly crashes in the application under test. Torqueo was also useful in assisting with reproducing defects that involved running test scenarios over a long period of time mixed with sudden user actions such as powering off the device. This success in finding such defects led to the further development of the tool.

6 CONCLUSIONS

Torqueo was developed to fill a gap in the current mobile development and testing field. An earlier version of Torqueo was used to test several commercial projects and was successful in finding defects in Windows Embedded Handheld applications. The advantage of using Torqueo was that it was highly configurable and allowed for reproduction of defects using the exact same data. The main objective of the tool is to give developers and testers several fuzz testing options to generate random test data from the one framework. However what was unknown was the performance impact on the device while testing applications. This experiment allows testers and developers determine which GUI invocation techniques to apply when conducting tests on Windows Embedded Handheld applications.

REFERENCES

Abdallah, N., 2010. *Performance Impact of Using .NET Reflection in .NET Compact Framework Applications.* Retrieved 12 22, 2010, from Monash University - Clayton School of Information Technology Publications: http://www.csse.monash.edu.au/publications/2010/tr-2010-260-full.pdf

Abdallah, N., & Ramakrishnan, S., 2009. *Automated Stress Testing of Windows Mobile GUI Applications.* International Symposium on Software Reliability Engineering (ISSRE). Mysore, India: IEEE, ISSRE.

Alsmadi, I., 2008. *Building a GUI Test Automation Framework Using the Data Model.* Saarbrucken, Germany: VDM Verlag Dr. Muller Aktiengsellschaft & Co.

Chen, T. Y., Kuo, F.-C., Merkel, R. G., & Tse, T., 2009. *Adaptive Random Testing: the ART of Test Case Diversity.* The University of Hong Kong, Department of Computer Science, Pokfulam, Hong Kong..

Chong, W. H, 2006. *iDEN Phones Automated Stress Testing*. World Academy of Science, Engineering and Technology.

Codenomicon. (n.d.). *Codenomicon Defensics 3.0*. Retrieved 11 16, 2010, from Codenomicon: http://www.codenomicon.com/defensics/

Forrester, J. E., & Miller, B. P., 2000. *An Empirical Study of the Robustness of Windows NT Applications Using Random Testing.* Retrieved 05 16, 2010, from The University of Wisconsin Madison: http://pages.cs.wisc.edu/~bart/fuzz/fuzz-nt.html

Google. (n.d.). *Monkey Runner*. Retrieved 01 13, 2012, from Android Developers Documentation: http://developer.android.com/guide/developing/tools/monkeyrunner_concepts.html

Hammersland, R., & Snekkenes, E., 2008. *Fuzz testing of web applications*. Retrieved 07 16, 2010, from AquaLab Research in Distributed Computing: http://www.aqualab.cs.northwestern.edu/HotWeb08/papers/Hammersland-FTW.pdf

Memon, A. M., Pollack, M. E., & Soffa, M. L., 2000. *A Planning-Based Approach to GUI Testing*. Proceedings of The 13th International Software/Internet Quality Week. San Francisco, California.

Microsoft. (2010, 04 08). *Hopper Test Tool*. Retrieved 07 18, 2010, from MSDN (Microsoft Developer Network): http://msdn.microsoft.com/en-us/library/bb158517.aspx

Microsoft. (n.d.). *Windows Embedded OS*. Retrieved 11 18, 2010, from Microsoft Windows Embedded: http://www.microsoft.com/windowsembedded/en-us/about/what.mspx

Miller, B. P., Cooksey, G., & Moore, F., 2006. *An Empirical Study of the Robustness of MacOS Applications Using Random Testing.* Retrieved 05 16, 2010, from The University of Wisconsin Madison: ftp://ftp.cs.wisc.edu/paradyn/technical_papers/Fuzz-MacOS.pdf

Miller, B. P., Fredrikson, L., & So, B., 1990. *An Empirical Study of the Reliability of UNIX Utilities.* Retrieved 05 16, 2010, from The University of Wisconsin Madison: ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz.pdf

Miller, B. P., Koski, D., Lee, C. P., Maganty, V., Murthy, R., Natarajan, A., et al., 1995. *Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services.* Retrieved 05 16, 2010, from The University of Wisconsin Madison: ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz-revisited.pdf

Sutton, M., Greene, A., & Amini, P., 2007. *Fuzzing Brute Force Vulnerability Discovery.* New Jersey, United States: Pearson Education.

Ye, M., Fneg, B., Lin, Y., & Zhu, L., 2006. *Neural Networks Based Test Case Selection Strategy for GUI Testing*. Proceeding of the 6th World Congress on Intelligent Control and Automation. Dalian, China: IEEE.