Chapter 22
User Interface Design in Isolation from Underlying Code and Environment

Izzat Alsmadi
University of Texas A&M, USA

ABSTRACT

The success of any software application heavily depends on the success of its User Interface (UI) design. This is since users communicate with those applications through their UIs and they will build good or bad impressions based on how such UIs help them using the software. UI design evolves through the years to be more platform and even code independent. In addition, the design of an application user interface consumes a significant amount of time and resources. It is expected that not only the same UI design should be relatively easy to transfer from one platform to another, but even from one programming language release to another or even from one programming language to another. In this chapter, we conducted a thorough investigation to describe how UI design evolved through the years to be independent from the code, or any other environment element (e.g. operating system, browser, database, etc.).

INTRODUCTION

Graphical User Interface (GUI) is a recent term for an area that is used to be called Human Computer Interaction (HCI). While both terms are not identical or synonymous, they refer to the same area or subject on how to design the interfaces of software applications through which users interact with those underlying software. On the other side, Application Peripheral Interfaces (APIs) indicate low level interfaces between software applications and other applications, operating systems, databases, hardware components, etc. The commonality between GUI and API (in addition to the last word; Interface) is that both include how the subject software is going to interact with its environment (users; GUI, other applications; API).

The design of the GUI is considered very important and critical to any software. A very successful software, from the inside, may fail by large if its GUI fails to attract users to use and understand features

DOI: 10.4018/978-1-5225-3422-8.ch022

that exist in this software. Alternatively, a very attractive GUI may boost, a shallow software, from the inside to be more successful. You may see many applications or websites in the market that offer the same features or services. Why certain ones are more popular?! Their GUI can be the first thing to think of.

The evaluation of GUIs is also unconventional and we should usually combine some formal verification or testing techniques with some informal techniques. In other words, while the automatic testing of user interfaces is important and continuously growing (Alsmadi & Magel 2007), there are some important parts of the user interface that they should manually be validated by users or testers. In this context validation is the term usually used to indicate those parts of the requirements, unlike verification, that need to be evaluated and tested through the users and not by formal methods, mathematical proofs or test automation tools.

Test automation for user interfaces is very popular and convenient. Testing application interfaces usually consumes a significant amount of project time and resources. The percentage of automating testing activities can vary from one software product to another and from one software module/component to another. In general, it is desirable to achieve 100% or high percentage coverage in user interface testing (Alsmadi, 2014). However, there are many obstacles toward achieving such 100% coverage. There are some user interface aspects that need human or manual validation for approvals. For example, the possible appropriateness of GUI components' layouts or coloring can be very hard for tools to automatically verify without human visual eyes assistance. In addition, the continuous evolution of graphical user interface components creates a challenge on test automation tools. In particular, the majority of those tools use some reverse engineering methods or libraries (e.g. Reflection in Java) to read all GUI components at run time and be able to interact with those components (Amalfitano et al., 2012; Banerjee et al., 2013). Those reverse engineering libraries may not have methods that can extract information from new GUI components that they are not developed to normally handle or serialize.

In this chapter, we will focus on evaluating how the design of user interfaces evolve to be more independent. We will focus on two aspects of this independency: Platform independent, and UI design patterns and principles.

PLATFORM-INDEPENDENT UI DESIGN

The term "platform-independent" or "cross-platforms" have been very important marketing or selling themes for many software applications to show signs of robustness and flexibility. For example, Java, in comparison with C# considers its main distinguished different is being platform-independent where C# can only run within Windows environments whereas Java can run on Windows and many other environments (e.g. MAC, Unix, etc.). User Interface design contributes significantly to making one software application platform-independent or not. In some cases, UI can be platform-independent while optimized to work on one specific platform.

UI interfaces exist in different platform versions or forms. The same application can have a web browser version, Desktop version, console or terminal version and mobile version. There are two main important factors that justify the need to have a unified UI for the same software application on the different platforms:

1. For users, it will be easier to deal with the different versions or forms of the same application on the different platforms if those forms are unified. Their learning curve can be faster and they can