Use Interface Testing

Mike Kelly

First Published on SQAtester.com

In this article we will take a look at user interface testing. I will attempt to offer you insight into what choices are involved in creating a UI as well as why those insights are important. I will then discuss some simple yet powerful strategies for finding errors. This article is intended mainly for the novice tester, and or testers new to graphical applications.

In their book *Testing Computer Software*; Cem Kaner, Jack Falk, and Hung Quoc Nguyen describe the compromise that a programmer must make when designing a user interface. The programmer must reconcile the competing priorities of a well-designed user interface. Kaner, Falk, and Nguyen list the following factors of UI design:

- Functionality
- Time to learn how to use the program
- How well the user remembers how to use the program
- Speed of performance
- Rate of user errors
- The user's satisfaction with the program

To this list I would add the real world properties:

- Technologies available to the designer
- Timeline for completion

As a programmer designs their user interface they will be faced with tradeoffs between these various factors. Which technologies available offer the best performance (C/C++) but are also easy for the user to use and have a consistent look and feel (JAVA)? How much of the functionality can one implement and still meet the deadline for this release? And how does one design the system in such a way that it is easy for beginners to learn (Windows) and yet flexible and lightweight enough for power users to leverage (Unix)? There is no one answer to any of these problems. Further, with every release each of these factors may take a different priority making the user interface take on different qualities for each release.

So why do I, the tester, care about all of this? I have my set of development standards, industry standards, and I know what is acceptable as far as look and feel for my application. I can spot a user interface error just a quick, if not quicker then the next tester! Well it's really not all that simple. It's important to understand what goes into the design of the interface, because it offers a deeper understanding of what types of error can present themselves, as well as making you aware of when the designer has introduced an error, and when the designer

has simply made a tradeoff. When a tester is not aware of these factors defect tickets get returned as "Functions as Designed" even though the tester "knows" that this is a bug.

Testing a user interface can be the most frustrating testing that can take place. The designer knows what he/she wants to see, and that vision quite often differs from what the tester is expecting. This results in frustration by both parties as the designer is getting what they deem to be premature defects submitted against them, and the tester thinking that the designer can't even read a simple standards document. That's not to say the designer is always right, but having filled both shoes it is easy to appreciate that both parties can be right.

So what should I look for when I'm testing a user interface? First and foremost ask questions, more so then on any other part of the application. Get to understand why certain decisions were made. This will not only reduce tensions between those developing and those testing, but it will better equip you to assess where the risks are in the system. Knowing that the developers are using a technology for the first time (a new reporting tool, a new language, etc…) will not only make you a kinder, gentler tester, it will encourage you to focus on simple behaviors that you might otherwise assume work. Just because you have tested a JAVA tree does not mean your developer has built one before. Simple things like expanding and collapsing nodes could be a potential problem.

Next step, grab someone off of the street and have him or her use your system. Sounds silly right? Wrong. As you develop test cases, see prototypes, look at the database, and submit tickets you loose the ability to look at the system from the perspective of a new user. Look at the questions a tester new to the project asks compared to a tester who's been there for two years. Most questions are simply an attempt by the new person to understand why things were done the way they where, not to determine what the system is doing. They can readily see the tradeoffs that took place, because they were not privy to the problems that forced the compromise. By stealing a resource from another team for a few days you can quickly determine the some of the "softer" requirements:

- Time to learn how to use the program
- How well the user remembers how to use the program
- Rate of user errors
- The user's satisfaction with the program

If your user interface has any performance requirements, automate that performance testing. Requirements like refresh rates, load and save times, use of system resources by specific controls, etc, can all fall under user interface testing. Instead of sitting there with a stopwatch (something I have actually done to prove a point to a developer) get a tool that will do it for you. If you do no other automation for you application this, at the very least, *needs* to be automated.

Finally, trace your requirements to where they become manifest within the application. Automating your testing will make this easier, but even without automation this is a must do. Build a physical checklist for each requirement and simply see if you can find it in the GUI. Once you find it, half the battle is over. The next step is to validate it as nessesary (either manual or automated, both methods are good). This simple process alone will help you find numerous gaps in functionality that you would have otherwise missed. Further, it will also offer insight into system navigation. If you can't find something quickly, there is a good chance that there is a problem with the design.

The user interface is the most ambiguous part of an application. Remember to have patience, be methodical in hunting down your requirements, to automate your performance tests, and to get a fresh set of eyes on the interface for each release of the project. These simple steps will get you on the right track to better user interfaces. There are hundreds of other things you can and should be doing, but these simple steps will get you on the right track. Recommended reading for an in depth look at user interface error can be found in *Testing Computer Software*.