Foreword

By Dave Winer, CEO, UserLand Software

I remember as if it were yesterday my first experience with a user. I had been developing a software product for three years, all the while thinking it was easy to use. A friend who had been listening to me gush about how great it was asked if he could try it. Hesitantly, I said yes.

I launched the program and we switched seats. I tried to say nothing as he wondered what to do. The software didn't have anything to say. "What should I do?" he asked. I thought to myself, "I have some work to do."

This is the moment of truth for any software developer, and one we avoid. In *The Soul of a New Machine*, Tracy Kidder tells about the first problem report from "the field" about a computer system developed at Data General in the late 1970s. The lead developer was surprised. In his mind the computer was a development project; that real people would try to *use* it attacked his perception of his own product.

We all go through this; at a superficial level we think we're designing for users, but no matter how hard we try, we're designing for who we think the user is, and that means, sadly, that we're designing for ourselves. Until you prove that it's usable by other people, your software is certainly *not* designed for them.

Until you make the shift and let the users tell you how your software works, it simply can't be usable. Every successful software product is proof of this, as is every failure. How many times have you installed some software or visited a Web site and wondered, "What does this do?" Now, understand that other people are asking the same question about your software. It's a puzzle, to solve it you

must figure out how to get your software into a user's mind, and to learn how to do that, you must learn how that mind works.

Joel's book is a milestone built on a strong foundation of practical experience. He's absolutely right that user testing is easy. You don't need a lab to do it, although many people think you do. You just need a computer and a person who doesn't know your software. It's an iterative process. Do it once, it'll change your whole perspective. Do some engineering. Do it again with a new person. Repeat the process until the first-time user knows what to do and can actually use the software to do what it was designed to do.

Joel's book is about more than software design and user-centricity. Once you learn how to communicate with users through software, it's inevitable that all your communication will improve. The central "aha" is to realize that other people use your software, and they don't know what you know, and they don't think like you think they do.

There are some very simple truths in this book, and sometimes the simplest truths can be most difficult. But Joel makes it so easy! His stories are clear and human and fun. And that may be the biggest lesson, if you haven't been designing for users, you're not having as much fun doing software as you could.

I can tell you from personal experience that there's nothing more satisfying as a professional software developer than to have a product resonate with the market, to have thousands of people tell you that they couldn't work without your software. To get there, you have to learn from them as you teach. Yes, your software is great, I believe you, but if no one uses it, it can't make the world a better place.

Dave Winer
http://www.scripting.com/

Introduction

Most of the hard core C++ programmers I know *hate* user interface programming. This surprises me because I find UI programming to be quintessentially easy, straightforward, and fun.

It's *easy* because you usually don't need algorithms more sophisticated than how to center one rectangle in another. It's *straightforward* because when you make a mistake, you can see it right away and correct it. It's *fun* because the results of your work are immediately visible. You feel like you are sculpting the program directly.

I think most programmers' fear of UI programming comes from their fear of doing UI *design*. They think that UI design is like graphic design: that mysterious process by which creative, latte-drinking, all-dressed-in-black people with interesting piercings produce cool-looking artistic stuff. Programmers see themselves as analytic, logical thinkers: strong at reasoning, weak on artistic judgment. So they think they can't do UI design.

Actually, I've found UI design to be quite easy and quite rational. It's not a mysterious matter that requires an art school degree and a penchant for neon-purple hair. There is a rational way to think about user interfaces with some simple, logical rules that you can apply anywhere to improve the interfaces of the programs you work on.

This book is not *Zen and the Art of UI Design*. It's not art, it's not Buddhism, it's just a set of rules. A way of thinking rationally and methodically. This book is designed for programmers. I assume you don't need instructions for *how* to make a menu bar; rather, you need to think about what to put in your menu bar (or whether to have one at all). You'll learn the primary axiom which guides all good UI design and some of the corollaries. We'll look at some examples from real life, modern GUI programs. When you're done, you'll know enough to be a significantly better UI designer.