Dealing with Complexity in UI Design

Guest post by **Julian Fietkau** 2011-03-11 Graphics 50 Comments

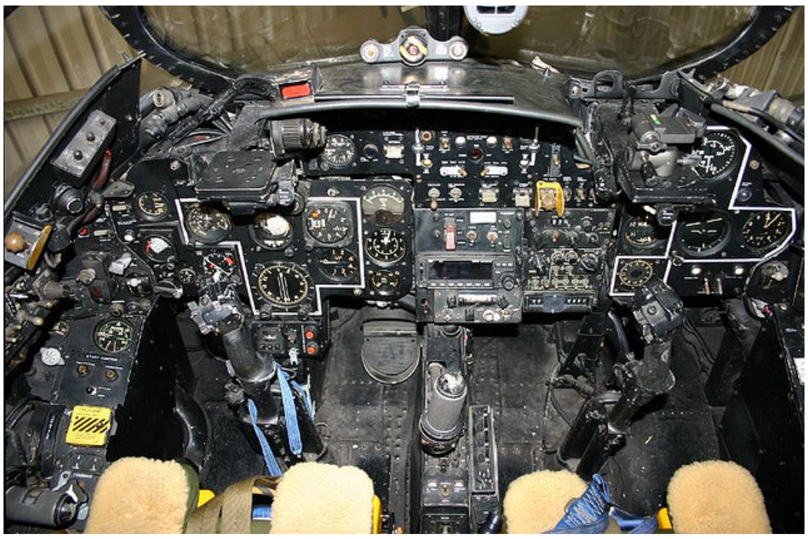
Over the past few decades, the software that enables us to be productive with our computers has become increasingly sophisticated and complex. Today's UI designers are faced with the challenge of devising graphical user interfaces that are easy to grasp and use, yet still provide access to a wide range of features. Here are some ideas about the nature of GUI complexity, followed by a couple of thoughts on simplicity that might just surprise you.

ON THE NATURE OF COMPLEXITY

Everybody wants simple software. As a programmer, I often read about "bloat" and "featuritis" and how they are problematic both for software development and usage. So what compels software developers worldwide to keep creating ever more complex software?

In the interest of clarity, let's start with some definitions. The meaning of "complexity" that I'll be referring to in this article derives pretty much directly from Don Norman's most recent book, *Living with Complexity* (amazon) and can roughly be summarized like this: **complexity** is a *measure for the structural sophistication of a system* (be it a software application, an airplane cockpit or a toaster). That is, the more parts and properties, the more nooks and crannies something has, the more complex it is. I prefer this definition to other common ones because it decouples the complexity from its widespread negative connotation. It's purely descriptive and non-judgmental. In contrast, a system may also be **complicated**, which I (in accordance with Don Norman) use as a *measure for the extent in which it is confusing and unpredictable to a human user.* This acknowledges that something being "complicated" is, at its core, a subjective feeling or, if you prefer, a psychological phenomenon. It has to be considered within the context of an individual human's experience to be meaningful.

Don't believe me? Take a look at the following user interface.



("Hawker Hunter Cockpit" by Greg Weir on flickr, used under CC-BY)

To any non-pilot, this is an interface nightmare. The sheer number of buttons and levers is both complex and complicated, unless you know how to fly a plane. In that case, the complexity remains obvious, but instead of a paralyzing, complicated mess of buttons you see an overview of all possible actions, nicely ordered in meaningful groups, with everything important right in your reach. The difference lies in the education about the tools, and experience with the problems they enable you to solve.

So “complex” and “complicated” are not the same thing, but they’re not completely independent either. Add complexity without care, and you might soon end up with a very complicated system. Likewise, reducing complexity is a viable way to make something less complicated. But as we will see, it is not the only way at all (or even necessarily a good one).

COMPLEXITY ON THE RISE

How do you actually measure how complex or complicated something is? I don’t have a definitive answer for either of them, but there are approximations we can make. I’ll use this opportunity to corroborate the claim that software is becoming more complex with some quantitative data courtesy of Jensen Harris, who wrote a four part article series about the history of the Microsoft Office UI (recommended reading for his musings on complexity). Among other insightful information, he also presents the *number of toolbars* across major versions of MS Office up until Office 2007, which is when they were abolished in favor of the Ribbon. These numbers are reproduced in the following table:

Version of MS Word	1.0	2.0	6.0	95	97	2000	2002	2003	2007
Number of toolbars	2	2	8	9	18	23	30	31	n/a

Of course the number of toolbars is not a perfect measure of the available functionality, but it works for seeing the trend, doesn’t it? If you wish to delve deeper, Harris’s blog offers a lot of detailed information about the interface complexity of Microsoft Office.

“But,” you say, “not all software is getting more complex! What about my simple and elegant Web 2.0 apps?” Good point! Let’s take a look at Google Docs, late 2007 and Google Docs today. It started out as a lightweight online alternative to traditional word processing software, but soon had to face the reality that, for people to actually switch over on a large scale, it has to provide the features that users need and want (which apparently even includes those awful rulers). Just skim the Google Docs Blog and check for yourself how many of those blog entries deal solely with the introduction of new features.

In summary: The increase of complexity in modern software is a very real thing. But *why*? The answer is surprisingly simple: expansion. A lot of software is created with monetary return in mind, which obviously depends on the number of people who are willing to fork over the necessary amount. Even free software is often fighting for both mind- and market share. So, how do you get your software to appeal to more people? Let’s see what Joel Spolsky has to say about feature bloat:

> I think it is a misattribution to say, for example, that the iPod is successful *because it lacks features*. If you start to believe that, you’ll believe, among other things, that you should *take out* features to increase your product’s success. With six years of experience running my own software company I can tell you that *nothing* we have *ever* done at Fog Creek has increased our revenue more than releasing a new version with more features. Nothing. The flow to our bottom line from new versions with new features is absolutely undeniable. It’s like gravity. When we tried Google ads, when we implemented various affiliate schemes, or when an article about FogBugz appears in the press, we could barely see the effect on the bottom line. When a new version comes out with new features, we see a sudden, undeniable, substantial, and permanent increase in revenue.

I rest my case.

CAPABLE, YET USABLE

The other side of the coin is increased usability, another thing that everybody wants.

Software today is consumed quicker than ever. Remember when you bought software in a store after careful consideration, the (printed and included!) manual was several hundred pages long and the installation could consume whole evenings?

Today you get software on an ad hoc basis through the internet. With software repositories and app stores becoming more widespread, download, installation and system integration are reduced to a single click. The software world is experiencing a paradigm shift from "software as a long-term investment" to "software as a commodity", and to gain traction, software has to be quickly learnable and discoverable. Steve Krug, author of the iconically titled *Don't Make Me Think* (amazon), has the mentality figured out:

> One of the things that becomes obvious as soon as you do any usability testing — whether you're testing Web sites, software, or household appliances — is the extent to which people use things all the time without understanding how they work, or with completely wrong-headed ideas about how they work.
>
> Faced with any sort of technology, very few people take the time to read instructions. Instead, we forge ahead and muddle through, making up our own vaguely plausible stories about what we're doing and why it works.

Besides broad functionality, good usability is a second cornerstone for the potential to expand the user base. With all this background information in mind, we can rephrase the opening question: How do we deliver good usability without compromising feature breadth? Should we be striving for simplicity, as it is so often demanded?

WHAT TO STRIVE FOR

When I ask myself how I should create a good user experience, the trusty ol' ISO 9241-110 always works as a reminder of what my conceptual goals should be. *Simplicity* is conspicuously absent. Instead, I read about properties like *suitability for the task*, *suitability for learning* or *self descriptiveness*. These are the things that are actually helpful to my users, rather than being a matter of taste or preference.

If we interpret *simplicity* as the reduction of the number of available actions at a given point, or (perhaps less academically) *taking out features*, then it is certainly a way to reduce the dreaded "aura of complicated".

But, and if you take away anything from this article I hope it is this one thought, **simplicity is a means, not an end.** There are many other ways to create interfaces with great usability without compromising complexity or limiting features.

In fact, if anyone's interested, I will be writing a follow-up article describing and showcasing several such techniques. I'd be glad to hear your thoughts on the topic.

About the author:
Julian Fietkau is a student of computer science and human computer interaction at the University of Hamburg. He is hoping for a career in interaction design and usability engineering.