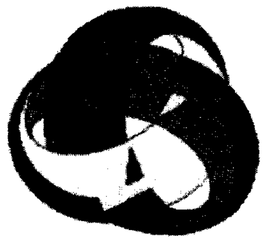
Too Much of a Good Thing?

Identifying and Resolving Bloat in the User Interface: A CHI 98 Workshop

Leah Kaufman and Brad Weed



Introduction

Computer industry criticisms of 'bloat' are not a new occurrence. Typically, users and pundits alike complain about how slowly applications load and run or the amount of RAM required to run a single, let alone multiple, programs. In recent years, however, we've seen additional criticism of bloat in the software interface design and the sheer number of features built into products. A 1992 critique [1] stated that:

"Maybe we've been blitzed by too many kitchen-sink demos. But we do notice a troublesome trend: A lot of new GUI products have become really, really hard to figure out. We keep seeing screens buried in layer upon layer of dialog boxes and windows, cluttered with randomly placed icons, file directories, pick lists, radio buttons, data entry fields, and other graphical furniture."

Blame for this bloat in the interface layout and controls is usually attributed to a steady increase in the functions and features built into each new version of software products. As MS-NBC columnist Joan Connell explicitly stated:

"In 1992, there were 311 commands in the word-processing program I use to write this column. In 1995 it took 647 commands to arrange sentences and paragraphs, check spelling and do all the other mysterious things necessary to transmute human thought to a computer and deliver it to whoever wishes to read. And today, Microsoft Word has become so adept a human helper that there now are 1,016 commands required to accomplish roughly the same task..."

What impact does this sheer increase in number of functions have on users trying to do their work? How does it affect the visual design and layout of the interface? Is the problem perceptual and due to poor visual design? Or is it conceptual and rooted in an overabundance of features?

We've reached a point in the software industry where its important to understand the source of this problem, it's effect on users, and figure out how it might be resolved.

This workshop brought together usability researchers and interface designers to jointly examine the problem of feature bloat in software. Through discussion, presentations, shared insights and experience with interface design, testing, and use, we worked towards a clear understanding of this issue and recommendations for addressing and studying it.

Opening Exercise

The workshop began with an affinity exercise – we posted three pairs of pictures each showing an early and later model of given device: a telephone, a watch, and a computer game. The workshop participants considered each pair of pictures, noted the differences, put a phrase or description of each difference on a post-it note, and placed the note next to the picture. Then, after removing the pictures the participants organized all of the post-it notes into categories. The labels they gave to each category became our springboard for discussing how the changes in product design become a source of bloat.

Aspects of Bloat

Each aspect represents one of the categories from the affinity exercise. These descriptions were built through group discussion and sharing our experiences with observing users and using and designing software.

Feature richness

A device with many features makes it possible to do many things with that one object or tool. Feature richness turns into bloat when there are more features than you want to use.

Not obvious how to accomplish a task

When using a feature-rich device, it can be difficult to match the features to specific tasks, especially if the features are poorly organized or displayed.

Unnecessary information

The tasks we do and the devices we make for doing them both vary in complexity. Bloat arises when the user expects the task to be simple and the device for doing the task is unnecessarily complicated.

Visual Clutter

When the user interface contains a lot to look at, you get a sense of crowding and clutter, especially when the organization isn't recognizable

Misuse of Color and Design Elements

If color is overused then too many parts of the interface compete for the user's attention.

Physical constraints of the context

Some interface designs end up feeling bloated because of physical constraints such as not enough space or limits on how graphical elements can convey organization.

Too many widgets?

Some bloated interface designs contain too many indistinguishable buttons and controls. The controls may also be poorly designed for their task.

System fragility

Too many unknown features can cause users to worry about whether their actions are correct and if the system is liable to break. They become afraid that one wrong move will bring the system down.

Excessive learning time

There should be a comparable payoff for putting the time into learning how to use a feature. Bloat happens when the amount of effort needed to learn a feature isn't commensurate with its utility.

We decided that these categories can act as a diagnostic tool for explaining why an interface might feel bloated. They also imply the characteristics of an unbloated UI: visually engaging, recognizable organization, appropriate use of color, all of which support efficient use of the features. We also noted that bloat may differ for novice and experienced users. Novices may be overwhelmed by the objects, by the number of unfamiliar visual elements. Experts, because they more readily recognize the interface elements, may be more attuned to actions and how quickly they can do their tasks.

We concluded that even with these criteria for identifying bloat in the interface, software with a lot of features is *not* inherently bad, hard to use, poorly designed, or bloated. Instead, we believe it's the combination of the user's software experience, goals and tasks, and how well the software matches these that leads to the experience of bloat in the UI.

Why does Bloat happen?

We next considered why software bloat occurs and came up with a range of answers. First, consumers frequently believe that more is better, hence, users buy products that have more features. This is buoyed by the desire to be efficient in spending and 'get the most for your money'. Secondly, users like to feel smart; buying the more advanced, comprehensive version of the software may make them feel good about themselves.

From the software company's perspective new features help distinguish their software from competitors and keeping the old features means that current users can still do tasks the way that they're used to doing them. Within the company programmers like to be creative, to design and implement new features. Some new features are so easy to add it seems that there's no good reason not to include it.

Finally, in our concern for making the software customizable and usable for a wide range of users we inadvertently contribute to bloat when we add on multiple methods for doing a task and include even more explanations on how to use the software.

Presentations

Each participant gave a presentation on an aspect of bloat, either problems, solutions, or information about users and their reactions to bloat. Andrea Mankoski (JavaSoft) looked at legacy features and challenged us about whether features used by 3% of users could be cut from a product. Avi Parush (HIT, Tel Aviv University) examined the usability and design issues in five different scheduling products. Joyce Westerink (Phillips Research Labs) pointed out that the drive to incorporate new technologies is usually considered more important than making products more usable.

Joanna McGrenere (University of Toronto) characterized unbloated software as software that makes a good match between the set of skills needed to operate the system and the set of skills the user brings to the system. From this perspective, users are more successful with a system when there's less to learn. Similarly, Sean Draine (Microsoft) reported that users like feature growth but hate UI growth because it means problems of discoverability, learning, and command confusion.

Donna Wallace (Microsoft) described a UI that gave a simplified initial interface for new users and allowed experienced users the option of building a more complex UI by adding advanced features to their screen. This complemented Erica Seidel's (Sony Research Labs) presentation on using different interface modes for distinguishing between classes of tasks.

Finally, Brian K. Smith (MIT Media Lab) noted that software works best when its tools match users expectations. His research showed that removing already working tools just because another technology is available, isn't a viable solution.

In addition, the authors presentated two topics: The difficulty of creating simple designs for complex tasks; and results from usability tests that demonstrating that performance-wise, bloat is a function of the proportion of unfamiliar features in the interface.

Projected Research

To wrap up the workshop we discussed the research questions provoked by the day's work. We want to know how quickly do users skill sets change. What percentage of people are satisfied doing intermediate-level work and don't need more sophisticated features in their software? How much are users willing to customize their software? Would making this easier create interfaces that better fit each user and their tasks? Can we design general purpose software that is also appropriate for particular tasks?

References

1. Interfaces: a complaint about clutter. *Soft Letter*, Feb 22, 1992 v9 n7 p5(2)

About the Authors

Leah Kaufman is a Usability Engineer for Microsoft Office.

Brad Weed is the Product Design Group Manager for Microsoft Office at Microsoft.

Addresses

Leah Kaufman: leahk@microsoft.com;

Brad Weed
1 Microsoft Way
Redmond, WA 98052-6399, USA

Bradwe@microsoft.com