

Complexity

“*Easy to use is easy to say, but hard to do*”

Jim Allchin – 9/4/97

While I think everyone acknowledges that our software and PCs in general are hard to use, I don’t think the magnitude of the problem is understood at all. No other consumer product in history has been so hard to use and so fragile. Yet computers and software have continued selling despite this. It is an amazing testament to all the benefits that personal computers bring that consumers are willing to put up with the vast array of associated problems.

All of us have our horror stories about spouses, neighbors, family members, *ourselves* where we wasted a weekend (or longer!) trying to get something to work. We are supposed to be experts, but even we can’t get things to work properly. (Any one who gets ISDN working without getting help from someone else deserves an award.) It is a total embarrassment. Craig Mundie said to me recently that he never touches his machines in his home except at the beginning of a weekend because it knows it might take the entire weekend to recover from whatever goes wrong.

What is fascinating is that when you talk to average users, they assume that *they* aren’t smart enough or that *they* are doing something wrong when the computer doesn’t work properly. However, if those same people were dealing with other consumer appliances, they would more than likely say that the appliance was designed wrong. There is even a culture growing dealing with “idiots” and “dummies” that have to work with computers. This is an amazing social phenomenon. Eventually consumers will wake up with the equivalent of Ralph Nader taking the lead.

So, is it just that software has gotten so complicated that we can’t make the system resilient and easy to use any more? Is it that the demands for features mean that we can’t focus on naïve users? Is performance or memory size more important than consistent operation? What are we going to do about the hardware companies that continue building unreliable PC products?

Our software is approaching massive size (and internal complexity). NT Workstation 5.0 will have around 27 million lines of code at Beta 1 – and more will be added before it finally ships some time next year. For perspective, the first version of NT only had around 6 million lines of code. In my view however, the size of the system has nothing to do with whether the system can be made resilient or easy to use. In fact, more software is required to make systems consistent and more reliable; it just has to be the *right* software.

Our products are becoming more and more feature rich. We have ways to “tweak” virtually everything. The amount of information known publicly about flags/fields stored in the registry and how to tailor aspects of the software is amazing. We present user interface dialog box tabs covering technical features that only a miniscule fraction of all users will ever need (or understand). You could argue that this is great --- it’s a Turing machine, you know – you can customize it to do anything. Unfortunately, we seem to look at these things as a computer geek does and forget about “the rest of us”. On the other hand, we have done some good work in areas where we have added features (e.g., spelling correction in Word) which do not add additional complexity. I believe you can simplify the experience for naïve users and still maintain the flexibility for power users.

What about performance? Is it ok to trade off consistency or reliability for performance? How about trading off consistency or reliability for smaller memory size? Is it ok to make the system super fast *most* of the time, if it doesn’t operate correctly (or crashes) in some rare cases? Microsoft has a disease dealing with these questions. I have been in many meetings where tradeoffs like this are actively discussed from billg on down. “We could win the benchmark.” “It saves 80K of memory.” This is a travesty. It would be better to not have a feature than have one that doesn’t work consistently 100% of the time.

So the question is: Is it possible to have a set of rules that guide the development of software that is more resilient, consistent, and easier to use? Unfortunately, software is still art and I do not know of any perfect rules, but I do believe there is a philosophy that can be adopted to start addressing our complexity morass. The rest of this paper covers this philosophy.

The thoughts below may seem very obvious. You may disagree with some of them. Nevertheless, I claim that following these rules would result in software order of magnitude less complex and friendlier to humans. The question is why don’t we follow them.

# Quality

From rec.humor.funny:

|  |  |
| --- | --- |
| **Multitasking**  You can crash several programs all at once. No waiting!  **Built-in Networking**  You can crash several PC’s all at once. No need to buy Novell Personal Netware or LANtastic to crash.  **Microsoft Network**  Connect with other Windows 95 users and talk about your crash experiences. | **PnP**  Plug and Pray (that it works)  **Multimedia**  Experience the immense sign and sound of crashing.  **Compatible with existing software**  It will also crash your existing software |

From KISS Software, Rescue Me! Product advertisement: “There are over 9,345 ways to trash your system.” “Windows in your TV? Who wants to boot your TV every few hours?” I could go on and on. Our products have a poor quality reputation.

Quality is about ensuring the system works consistently: every time, all the time. A key metric I think about is crashing. Our software should never crash – period. We ship software today that we *know* has bugs in it that can crash the system or that has architectural holes that would allow an errant application to crash the system. We are under “market pressure” and somehow are able to convince ourselves that the likelihood of the condition (usually very complicated) occurring is very small and so it is ok to ship the product and fix it in the next release.

The first building block for making a system easy to use is quality. Quality doesn’t make a system easy to use; on the other hand, you can’t have a good system without it. You can have a very hard to use system that has high quality. MVS or VM/CP represent such systems. Once you get one of these systems running they are amazing resilient to hardware and software problems. I spent time with IBM engineers adding code for some of the error recovery paths for VM/CP. Tons of code exists to handle faults in these types of systems. It’s not that these systems were designed correctly. If you changed a line of code you were required to add your initials to the comment field on that line. I would look at pages and pages of BAL assembler code where the comment field of *every* line had been updated. And not by a single person. In more cases than not, there would be 5 or more initials on the same BAL line. Code (and quality?) by brute force.

So everyone says they support quality (Mom and apple pie, right?). What should we change?

* We should tradeoff compatibility for quality if necessary. At a minimum if we know that being compatible with some feature (from the past) could cause the system to crash or get in some inconsistent state, then we should have an option that a user can select that prevents the compatibility, in favor of stability. Users then have a choice in how the system operates.
* No algorithms should be used which we know a priori could cause the system to fail or enter an inconsistent state. No benchmark, size, or schedule crunch should be allowed to come between quality and us.
* We must ensure that PC hardware doesn’t impede our quality efforts. The classic example here is ISA slots allowing non-PnP cards to be supported. But, it goes beyond this. We need to drive the industry to address areas that are likely causes of consumer issues and faults in the system. An example here would be the fact that PC-Card (or PCMCIA) devices can be removed without warning from the system. That’s like being able to rip a CD out of your audio system while it is still moving and playing. If we want to improve the experience for end-users, we must focus on improving the fragile hardware we have today in PCs.

# Resiliency

Ok, suppose the system doesn’t crash. But, does it recover gracefully from problems? Resiliency is about having the system recover automatically from environmental issues (e.g., server going away, DHCP server not found, etc.) or common user mistakes.

This takes a very specific designer mindset. You have to assume that if something could go wrong, it will and then program defensively for these problems. Too often I find designers programming for the case when everything works perfectly. When an error occurs, they simply give up, spew an error message to the screen, and return an error message to their caller (who of course does nothing with the error except return it to their caller, etc.).

Networking is perhaps the area where this is most visible. Our networking code is extremely fragile today. Virtually any problem (lost packet, server not responding after we have a connection to it, configuration error, etc.) either makes the system totally unresponsive (with no indication to the end-user what is going on) or worse yet it actually stops working correctly. This lack of resiliency is a problem for the operating system (e.g., networking errors during authentication, etc.), but it’s also a problem for applications such as mail, calendar, chat, browsing, etc. No other peripheral is more error prone today than the network. It takes special designs to deal with this.