How many times does your computer just sit there doing nothing after you entered a command. There is no disk activity; no screen activity; nothing. But, you can’t do anything since the application or the operating system is “locked up”. This is because the software isn’t handling the network (flakiness) correctly. The network will continue to be very error prone (e.g., different delays, outages, errors, node failures, etc.) for quite some time in the future. Moreover, the fact that we want to be mobile moving between wireless cells or connecting between very different speed networks means that handling networking resiliency must be a top priority for us. Architecturally, we need to learn about network errors in a standard way and then avoid multiple timeouts, nested retries, etc.

We do have parts of the operating system that are more resilient today than other parts. One example is scandisk that automatically “repairs” the system if there are disk inconsistencies detected. The issue today is that scandisk may not correct the problem totally. In fact, it could delete a critical file. Still it is better than not doing anything. Assuming that we must have a utility like scandisk (because of the file system design) we need to ensure that upper levels of the software can recover from perhaps poor deletion choices that scandisk might make. (Furthermore, as noted later, if we have to have scandisk utilities, they should run silently without introducing the user to any bizarre terminology or operational choices.)

Resiliency is about keeping the system running at all costs. An example of this is that if the NT 4 shell (or IE 4 shell on NT 5) dies, the desktop disappears for a second. But, the operating system restarts the shell quietly in the background. The system is resilient to shell faults. This is not the case with win95 or win98.

Ways to improve resiliency:

* Auto-configure. Do not let the user configure things. Networking is one of the most complicated areas of our system. Asking a novice (or even an “expert”) to configure a protocol stack is like asking a layperson to do surgery – it takes training and even then they sometimes make a mistake. If you can't auto-configure, then the system should probably be redesigned so that it can be auto-configured.
* Assume a machine is transiently connected to the network. Never allow the user interface to be stopped by a network outage. Switch gracefully (but visibly) from network mode to offline mode.
* Once an error is detected (e.g., network timeout), it should be sensed only once. That is, at most one timeout should happen until there is good communications established again.
* Design for error cases even if it means the regular operation of the system is impacted slightly. A good example here is out of disk space. Assume you run out of disk space at the worse possible time. Will the system still function correctly? Can the user still remove some space? If it takes allocating some free disk space as a buffer during normal operation (to ensure that the system is still totally functional) then do it.

# Terminology

For some strange reason we insist on trying to teach people a foreign language: *geektalk*. My Mom doesn’t want to learn another language right now. She just wants to track her church work, write letters, exchange email, track her budget, look for things on the Internet, etc.

Geektalk is actually one of our worse sins in interacting with a user. Our systems tend to produce more internal information than a user needs to know and in doing so we use terms that only a computer literate person (and in some cases programmer) would understand.

Turn on (note a naïve user thinks a “boot” is a shoe) any PC and just watch the words/characters fly by.

**24576 KB OK**

**SystemSoft BIOS for Opt Viper 557/508N 1.00 (2450-35) (Version 01.11)**

**SystemSoft Plug-n-Play BIOS**

**<F10> to enter Computer Setup**

Oh, I see…

Now of all of the above, what do you think a first time user *has* to know? What terms would they know? Why do we insist on telling a user all these things? I believe it is because the system is basically in debug mode all the time. We never know when a user might have to know the internals of the system to deal with a problem.

When our software starts running we are even worse. I can’t even begin to represent the insane number of terms we use. We use terms such as “FAT32”, “clusters”, “defragging”, “DHCP”, “DNS”, “WBEM”, “WAN Mini-port”, “ODBC”, “enable IP forwarding”, “bindings”, “NDIS Proxy TAPI Service Provider”, “Video Compression Codecs”, “Device Tree”, “devmgmt”, IRQ, DMA, “Launch Internet Explorer Browser” (launch?), “Insert ActiveX Monitor Control”, etc.

And our error messages are nothing short of amazing. Consider these error messages from NT 5:

* No IPSEC Policy DN.
* {A79C9582-19D1-11D1-91C7-C4252BDEA3A4} : Could not find an adapter.
* The Net Logon service terminated with service-specific error 3095.
* No Windows NT Domain Controller is available for domain NTWKSTA. (This event is expected and can be ignored when booting with the 'No Net' Hardware Profile.) The following error occurred: The RPC server is unavailable.
* Illegal instruction
* And on and on.

When a TV is sold the manufacturer doesn’t have to explain how to read the color bands (detailing resistance) on some resistor that is used. Why do we insist on trying to make computer people out of everyone? We are still making software for the hobbyist. We need to make software for the consumer. Do we want the user to debug the system? Is our software so unreliable that we have to be in pseudo debugging mode all the time?

Of all the issues raised in this paper, reducing information dumping and simplifying/removing terminology is the easiest to change. Every product should have a glossary of terms that is used throughout the product. We can introduce new (computer) concepts to end-users, but they should be done with great care. Developers and user education (e.g., help, documentation) should only communicate using layperson’s language and the approved glossary.

One other key point is that some areas of a product might require more advanced knowledge and therefore a richer set of terms. In this case there should be an advanced button (or tool) that permits access to these terms (otherwise they are hidden). In general though these advanced dialogs should be avoided. Why? Because having the concepts exposed at all makes it hard to avoid having some error/information message pop out at a naïve user (e.g., “can’t renew IP address because DHCP server is not responding”). It would be far better in my view to have a command line utility that adjusts certain parameters (as in debugging mode), then have this information exposed in a general UI. Then we would never have to even explain what something like a DHCP server was to an end-user.

# Consistency

Lack of consistency is a key problem in our user interface and our underlying conceptual model. Humans are amazingly adept at dealing with ambiguity and inconsistency – but only up to a point. Much of learning is about understanding rules. If A, then B. Anyone who programs knows that deep nesting of **if**/**else** statements is very difficult to debug. In every day life it is no different. If you have to memorize one exception after exception, it is very difficult to not make mistakes unless you are living and breathing the environment constantly. Teaching someone about the system deals with teaching him or her about the rules. You want the rules to be very simple: if you do this, then that happens.

Our systems have tons of footnotes on how it works. Let’s look at some examples.

**Inconsistent drag-drop**. When you drag a file from one place to another, it would seem reasonable that the file will be moved, not copied. That rule is not correct however. If you drag a file on the same disk, then the file is moved. However, if you drag between disks, the files are copied – not moved. If you drag an EXE, then a shortcut is created, and the file is neither copied nor moved – that is, unless the destination is a removable drive.

**Tray**. This area on the screen doesn’t seem to follow *any* rules. It holds mostly icons except for the clock that is. Some icons get double-clicks, some get single-clicks, some get right-clicked, and some don’t get clicked at all. Some icons have tool-tips, others don’t (it depends on whether the application window is selected or not). With IE 4, there is yet another area on the tray that operates totally differently than the other tray components. It is magic.

**Shortcuts**. Shortcuts are very inconsistent since they are not supported through win32 APIs. Shortcuts are kludges done on top of the file system. The result? Terrible inconsistency. Whether a shortcut is linked to a file or a folder, it is always treated as a file. Specifically, you can’t specify a shortcut to a folder in a path. So, shortcuts are “sort of” transparent, but not always.

We should not be rigid about consistency. There is a higher level principle of familiarity (with real –world objects/tasks) that I think is even more important than consistency across all aspects of the system. Nevertheless, we are not consistent enough today in our systems. We use different terms for the same task, object, etc. in different parts of our system (e.g., Emergency Recovery Disk). We need fewer concepts defined and few ways to do things. Concepts that we introduce should be consistent unless there is a overwhelming reason not to be.

# Familiarity

A key to allowing people to learn something quickly is to build on prior knowledge. We need to maximize the concepts and techniques in our products that users are familiar with from their real world experiences. The first time someone uses an automated washbowl, they look curiously around the sink for handles. Once someone explains how it works or they discover how it works independently, then from then on they will intuitively try putting their hands under the nozzle if they don’t see handles. Of course, in some countries the water flow is controlled through foot petals. And someone who is used to automatic washbowls will be surprised when the water doesn’t come on. (I know because this happened to me.) The point is that once someone learns something they become familiar with it and expect it to be the same in a variety of situations.

Our user interface and our conceptual model should attempt to model within reason tasks or objects that people already know. If there is an analogy that exists outside the computer realm that a culture understands we should use it in our systems. A good example of this is highlighting in Word. This is something that virtually all people are familiar with. Using yellow as the default and calling the feature highlighting is easy to understand for most everyone.