The admirable and usually prescient Ted Lewis may have got it partly wrong about Wintel when he guessed that “Microsoft and Intel are guaranteed a long and prosperous future with little intervention from [the US Department of Justice]” (“Who’s Afraid of Wintel?” Computer, Jan. 1998, pp. 149-152). By now it’s stale news that DOJ, through Joel Klein, head of DOJ’s antitrust division, has moved against Microsoft.

However, a rereading of Lewis’ well-argued column convinces me that DOJ’s case against Microsoft is—at the very least—highly questionable. Either way the action won’t be cheap for the U.S. taxpayer. And Microsoft winning the battle could strengthen their position, which, of course, is not what DOJ’s action is designed to achieve.

Even if DOJ wins, the effect could be bad. Writer Richard Miniter sees a danger that “Klein’s efforts to save Microsoft’s competitors may kill them” (“Busting Microsoft May Give Australia Free Rein,” The Australian Financial Review, May 21, 1998, p. 21). Miniter also claims that “thanks to Klein, American corporations might not be free to compete in the global market; they could be at the feet of bureaucratic regulators who construe every innovation as a step toward monopoly.”

Other commentators have made the same point, though rather less extravagantly. In any case, it looks like war. In war, the main casualties are all too often the bystanders.

MICROSOFT’S STRENGTH

It is Lewis’ view that Microsoft and Intel, left alone, will by 2017 “look as out-of-date and unimportant as Standard Oil is today.” That may be true, but I think the process of obsolescence should be hurried up. As a disgruntled user of Microsoft’s OS, I would like some changes, and I would like them soon. My machine must be rebooted far too often. Furthermore, as a user who finds the keyboard to be quicker than the mouse, I am fed up with the increasingly inconsistent and decreasingly frequent provision of keyboard alternatives to mouse manipulations. I’m also fed up with the uselessness of Microsoft’s various help facilities. Let’s face it, Microsoft’s software quality is poor.

Is the software competing with Microsoft’s any better? I don’t know. I just use what I’m given, and I’m given Microsoft. That’s common today, and that’s what Microsoft seems to have wrapped up. To get better software, Microsoft must be given some competition where it really matters: in the OS itself.

DOJ is trying to get at Microsoft from the applications side of the software structure in focusing on browser software, but that’s the wrong side. Ted Lewis points out that “without its upgrade business from MS Office and Windows 95, Microsoft would be a mere shadow of itself.” And as consulting engineer Bob Weeks makes plain in a letter to the editor (Computer, Feb. 1998, p. 4), Microsoft’s dominance is due to official adoption of its OS. MS Office is strong because MS Windows is strong, rather than the other way around. In other words, Microsoft’s strength is in its foundation, its OS.

The most striking aspect of computing in the 1990s, or at least the late 1990s, is the almost universal acceptance of windows as a way users interact with software. Users get a strong feeling of control, even exhilaration, in having several windows in production at once and being able to choose among them. The second most striking aspect in computing these days is the acceptance of second-rate software to run in those windows.

If having windows is at the heart of what turns users on these days, then what we need is a separation of the windows from the software that runs in them.

VIRTUAL CONSOLES IN HARDWARE

Before windowing took over, the console—the display terminal together with the keyboard—was implemented primarily in hardware. Now that hardware is so much more capable than it used to
be, it could run multiple virtual consoles just as easily as it is used to run single consoles. If we went to a hardware-based multiple-console model, the OS would be much simpler, and its programs would merely request a console from the hardware when needed.

On my desktop PC, the two windows I use most are a DOS window and a Unix window. It’s just like having two machine consoles on my desk, but the windows make it much more convenient. The OS simulates two consoles, but I can place the virtual screens where I want and size them to taste. I can also easily shift between them (with Alt-Tab).

Windowing is, at least from the user’s point of view, a straightforward facility. Each window is basically the visual part of a program’s console, and when control is shifted to a particular window, the keyboard is temporarily the console’s input device. The active program only gets to use its own console—or at least the keyboard or mouse—when the user shifts control to it.

With virtual consoles implemented in hardware, users would find it much easier to shift from OS vendor to OS vendor, and new OSs would be easier to develop without the burden of providing windowing support. Implementing consoles in hardware would let application developers compete with M icsosoft where M icsosoft is strongest: the OS. Furthermore, improvements in windowing could be developed in the hardware itself, probably with little need to change the programming interface.

Virtual terminals under control of hardware would also make it easier to shift to different terminals and promote PC/TV convergence. And special-purpose OSs—such as OSs that support networked domestic appliances, for example—would be easier to develop and easier for people to learn to use if we had a common hardware support system.

With Windows CE poised to expand into a number of markets, M icsosoft could easily be the proverbial 800-pound gorilla and threaten independent application innovation. But virtual consoles could short-circuit M icsosoft’s dominance and could even provide

Continued on page 109

Open Channel

XML: TOWARD BABEL

Continued from page 104

Currently, many designers advocate the use of CSS together with Extensible Markup Language (XML) as a replacement for HTML. Generally, W3C promotes XML as if it were an extension to HTML. In fact, XML is a simplified form of Standard Generalized Markup Language (SGML), in turn, is used to define the syntax of markup languages like HTML.

In other words, XML is basically a dialect of SGML. But by advocating the use of XML on the Web, W3C is essentially suggesting that everyone design a personal language for personal hypertext documents and different languages for different documents.

The XML metalanguage can define the formal syntax of a language, such as nesting rules for elements. The semantics could of course be described in plain English. But this doesn’t seem to be of interest to XML evangelists. They are more interested in just specifying presentation with CSS. Naturally, this means that they do not use CSS as a presentation suggestion only, since (with the XML/CSS model) there is no default or user-defined presentation. Quite probably there will soon be browsers that support XML and CSS to some technically satisfactory degree. The XML/CSS approach to Web publication might well be adequate in special cases. But switching from HTML to XML/CSS as a general solution would be a huge “devolutionary” leap.

As a publishing method, XML/CSS is comparable to using text processing software with styles or macros: XML/CSS structures documents, but the structure has no independent, public semantics. The XML/CSS approach means that instead of developing HTML so that it can adequately express the structure of a wider range of documents, we must create new markup languages.

What this means in practice is illustrated by MathML (http://www.w3.org/TR/REC-MathML/), the mathematical markup language. MathML is, to put it mildly, complex and difficult compared to the old proposals for adding basic mathematical markup into HTML (like the long-expired proposal in the HTML 3.0 draft at http://www.w3.org/MArKUp/html3/).

The Web needs a Renaissance. It must return to its classical roots. One of its classical roots is HTML as a simple, scalable, document format that can be used for information exchange on virtually any platform. Coming back to this model means a return to the original principles of HTML and very carefully extending the language in the spirit of those principles.

It will take time before we all realize that the original HTML proposals are still much stronger than the latest XML/CSS developments.

If you have been wondering whether you should hurry to catch the train and start learning XML and CSS, stop wondering. There is no need to run to the station. The XML/CSS train is leaving, but it’s headed into a land of confusion. Hold tight. There is still a lot of good work to be done with that simple, scalable document format we call HTML.

Jukka Korpela is a systems analyst in the Computing Centre at Helsinki University of Technology, Finland. Contact him at Jukka.Korpela@hut.fi or http://www.hut.fi/u/jkorpela/.
TO RELEASE OR NOT TO RELEASE

It is easy to criticize Microsoft (or any other software company) for not performing enough testing. But software companies cannot always afford to test their products twice as much as they currently do. It costs money and delays product launch. One thing we know for sure: Fast versioning leads to success in the software industry. High-quality products lose out to quick-and-dirty products because Joe Sixpack isn't a discerning consumer. Also, enough brain-dead Larry Lemmings will follow the market leader, so Microsoft could produce a lot of Gonzo Products before its bottom line began to sink. The Hollywood atmosphere of Silicon Valley keeps the Joe Sixpacks and Larry Lemmings buying software because of its popularity rather than its reliability!

The situation begs the question: When should software be released? Should it be held back, like fine wine, or churned out like sausage? The market determines this. For example, Microsoft's Windows NT is aimed at the enterprise, where few Joe Sixpacks remain. Here, software quality and reliability take on an importance lacking in the consumer space. Over 60 percent of the VARS polled by VAR Business Research [VAR-Business, Mar. 16, 1998, p. 8], cited bugs as their main concern in deploying Windows NT. In their view, Microsoft's Windows NT is relatively early in its product development cycle and doesn't measure up to Unix. Perhaps this is why Microsoft has invested seven years and hundreds of millions in development dollars in Windows NT. There are over 500 people working on it and the most recent estimate of code complexity lies between 22 and 27 million LOC. To bring the quality of Windows NT in line with that of Windows NT, Microsoft could attain a higher level of quality by doubling its test effort. Microsoft's software practice appears to require 44 iterations. Jones recommends six to 12, so Microsoft's software practice appears to exceed the industry norm. A total of 42 steps would be required to reduce the defect potential from 2.95 million to 5,000, as Microsoft must have done, requires 18 test iterations. Jones recommends six to 12, so Microsoft's software practice appears to exceed the industry norm. A total of 42 steps would be required to reduce the defect potential from 2.95 million to 5,000, as Microsoft must have done, requires 18 test iterations.

TO RELEASE OR NOT TO RELEASE

It is easy to criticize Microsoft (or any other software company) for not performing enough testing. But software companies cannot always afford to test their products twice as much as they currently do. It costs money and delays product launch. One thing we know for sure: Fast versioning leads to success in the software industry. High-quality products lose out to quick-and-dirty products because Joe Sixpack isn't a discerning consumer. Also, enough brain-dead Larry Lemmings will follow the market leader, so Microsoft could produce a lot of Gonzo Products before its bottom line began to sink. The Hollywood atmosphere of Silicon Valley keeps the Joe Sixpacks and Larry Lemmings buying software because of its popularity rather than its reliability!

The situation begs the question: When should software be released? Should it be held back, like fine wine, or churned out like sausage? The market determines this. For example, Microsoft's Windows NT is aimed at the enterprise, where few Joe Sixpacks remain. Here, software quality and reliability take on an importance lacking in the consumer space. Over 60 percent of the VARS polled by VAR Business Research [VAR-Business, Mar. 16, 1998, p. 8], cited bugs as their main concern in deploying Windows NT. In their view, Windows NT is relatively early in its product development cycle and doesn't measure up to Unix. Perhaps this is why Microsoft has invested seven years and hundreds of millions in development dollars in Windows NT. There are over 500 people working on it and the most recent estimate of code complexity lies between 22 and 27 million LOC. To bring the quality of Windows NT in line with that of Windows NT, Microsoft could attain a higher level of quality by doubling its test effort. Microsoft's software practice appears to exceed the industry norm. A total of 42 steps would be required to reduce the defect potential from 2.95 million to 5,000, as Microsoft must have done, requires 18 test iterations. Jones recommends six to 12, so Microsoft's software practice appears to exceed the industry norm. A total of 42 steps would be required to reduce the defect potential from 2.95 million to 5,000, as Microsoft must have done, requires 18 test iterations.

TO RELEASE OR NOT TO RELEASE

It is easy to criticize Microsoft (or any other software company) for not performing enough testing. But software companies cannot always afford to test their products twice as much as they currently do. It costs money and delays product launch. One thing we know for sure: Fast versioning leads to success in the software industry. High-quality products lose out to quick-and-dirty products because Joe Sixpack isn't a discerning consumer. Also, enough brain-dead Larry Lemmings will follow the market leader, so Microsoft could produce a lot of Gonzo Products before its bottom line began to sink. The Hollywood atmosphere of Silicon Valley keeps the Joe Sixpacks and Larry Lemmings buying software because of its popularity rather than its reliability!

The situation begs the question: When should software be released? Should it be held back, like fine wine, or churned out like sausage? The market determines this. For example, Microsoft's Windows NT is aimed at the enterprise, where few Joe Sixpacks remain. Here, software quality and reliability take on an importance lacking in the consumer space. Over 60 percent of the VARS polled by VAR Business Research [VAR-Business, Mar. 16, 1998, p. 8], cited bugs as their main concern in deploying Windows NT. In their view, Windows NT is relatively early in its product development cycle and doesn't measure up to Unix. Perhaps this is why Microsoft has invested seven years and hundreds of millions in development dollars in Windows NT. There are over 500 people working on it and the most recent estimate of code complexity lies between 22 and 27 million LOC. To bring the quality of Windows NT in line with that of Windows NT, Microsoft could attain a higher level of quality by doubling its test effort. Microsoft's software practice appears to exceed the industry norm. A total of 42 steps would be required to reduce the defect potential from 2.95 million to 5,000, as Microsoft must have done, requires 18 test iterations. Jones recommends six to 12, so Microsoft's software practice appears to exceed the industry norm. A total of 42 steps would be required to reduce the defect potential from 2.95 million to 5,000, as Microsoft must have done, requires 18 test iterations.