ARTICLE

# THE FETTERED LIBERTY TO INTEGRATE: LEGAL IMPLICATIONS OF SOFTWARE ENGINEERING

**BRUCE D. ABRAMSON**[*] **AND DMITRI L. MEHLHORN**[**]

TABLE OF CONTENTS

## I. INTRODUCTION

Microsoft does not monopolize the market for personal computing software. It does, however, monopolize the market for personal computing *platform* software. That position gives Microsoft the ability to leverage its way into adjacent software markets, often by either bundling or integrating its own products into Windows. This strategy can both protect and extend the original platform monopoly, and thus raises a critical question: Under what circumstances should antitrust laws prohibit a software monopolist from integrating new technologies into its monopoly product?

While this query currently focuses predominantly on Microsoft, its scope is likely to grow. Monopolists also dominate a number of niche software markets,[1] and debates remain about the appropriate legal definitions of both

[*] Law Clerk to the Hon. Arthur Gajarsa, United States Court of Appeals for the Federal Circuit. bda9@columbia.edu. Ph.D. (Computer Science), Columbia University; J.D., Georgetown University. Dr. Abramson worked as a consultant to parties involved in the European Commission proceedings against Microsoft before assuming his current position. The views expressed in this article are his alone.

[**] Director, Gerson Lehrman Group, 927 15th Street NW, Suite 900, Washington, DC, 20005. Dmitri@stanfordalumni.org. MPP, Harvard University; JD, Yale University. Mr. Mehlhorn worked as an attorney for one of the parties involved in the European Commission proceedings against Microsoft. The views expressed in this article are his alone.

[1] Leveraging claims against niche software monopolists, though different in many respects from the claims against Microsoft, have arisen in a number of "copyright misuse" cases. *See e.g.*, Lasercomb Am., Inc. v. Reynolds, 911 F.2d 970 (4th Cir. 1990) (regarding CAD/CAM); Alcatel USA, Inc. v. DGI Techs., Inc., 166 F.3d 772 (5th Cir. 1999) (regarding

"platform" and "integration."[2] Microsoft's unique position virtually guarantees that its decisions, actions, and assertions will play a central role in shaping both these debates and their ultimate resolution.

Microsoft has long argued that public authorities are not competent to police software design decisions, and that it should have "unfettered liberty" to integrate new technologies into its operating system so as to deliver exciting new products to eager consumers.[3] Though both Judge Jackson[4] and the D.C. Court of Appeals[5] challenged the extremes to which Microsoft pushed this view, Judge Kollar-Kotelly's November 2002 ruling did little to curtail either Microsoft's abilities or its incentives.[6] That omission was intentional.[7] Judge Kollar-Kotelly emphasized that she was crafting a "specific remedy for the limited ground of [Microsoft's] liability"[8] pending before her, (i.e., Microsoft's illegal maintenance of its Windows platform monopoly[9]), that should not "curtail the ability of a [future] court to determine that Microsoft has illegally tied two products which are separate under the antitrust laws,"[10] and that anti-integration remedies would be "more appropriately addressed as separate claims, in a separate suit . . . ."[11] She thus left open many critical questions about both the general and the specific propriety of leveraging-by-integration.

Some economists have asserted that no further inquiries are necessary—at least for the specific case of Microsoft.[12] In their view, Judge Kollar-Kotelly's ruling should address all outstanding issues that antitrust authorities currently

diagnostic equipment).

[2] Even the D.C. Circuit's announcement that tying claims involving platform software were subject to rule-of-reason analysis implied that the issue was far from resolved: "While our reasoning may at times appear to have broader force, we do not have the confidence to speak to facts outside the record, which contains scant discussion of software integration generally." United States v. Microsoft Corp., 253 F.3d 34, 95 (D.C. Cir. 2001).

[3] *See, e.g.*, United States v. Microsoft Corp., 980 F. Supp. 537, 543 (D.D.C. 1997).

[4] *See id.*

[5] *See Microsoft*, 253 F.3d. at 63 (D.C. Cir. 2001).

[6] *See* New York v. Microsoft Corp., 224 F. Supp. 2d 76 (D.D.C. 2002).

[7] *See id.* at 97.

[8] *See id.* Judge Kollar-Kotelly viewed activities on which no final judgment had been entered—notably the tying of the Internet Explorer (IE) browser to Windows—as outside the scope of a proper remedy.

[9] *See id.*

[10] *United States v. Microsoft Corp.*, 231 F. Supp. 2d 144, 166 (D.D.C. 2002).

[11] *New York v. Microsoft*, 224 F. Supp. 2d. at 134. The government dropped the tying claim before the court reassigned the case to Judge Kollar-Kotelly.

[12] *See, e.g.*, David S. Evans and A. Jorge Padilla, *Tying Machiavelli: The U.S. Microsoft Consent Decree*, NERA, 2002, available at http://www.nera.com/wwt/newsletter_issues/5664.pdf, (last visited December 12, 2002). Evans, Padilla, and NERA have all served as consultants to Microsoft.

investigating Microsoft may consider.[13] Because many of these same economists have also argued that Microsoft's past behavior (including the actions that the courts ruled anticompetitive) did not harm consumers, though, it is hardly surprising that they see little need for future protection.[14] We disagree with both their assessment of the past and their prescriptions for the future. In this brief article, we explain the importance of basic software engineering principles to the legal treatment of software integration, and apply our analysis to assess Microsoft's *technological* argument that its integration of cutting-edge functions into Windows are due the legal deference that courts normally accord to companies' product-design decisions. We establish three key points:

1. Basic software engineering principles indicate that stable, well-understood technology is a practical prerequisite for safe product integration. Premature integration inhibits innovation without improving product quality.
2. Software monopolists can neutralize nascent middleware threats and competitive applications through strategic, and often premature, "integration."
3. Judges and regulators are understandably hesitant about second-guessing a company's product development processes and decisions. However, strategic leveraging falls squarely within the purview of the authorities, and is thus due no special deference.

These three points combine to recommend a remedial rule, well within the competence of adjudicatory bodies, that would protect innovation, competition, and consumers: *Courts should prohibit software monopolists from integrating application technologies sold in competitive markets into their monopoly products at least until those technologies mature*. As long as important new features continue to emerge from the competitive race for product quality, integration of that product into a monopoly platform is likely to squelch innovation and reduce consumer choice without providing any concomitant benefit. In a competitive market, market forces may attenuate such premature integration, but in a monopoly situation only public policy can protect innovation.

## II. ENGINEERING VS. STRATEGY

The fundamental challenge of computer science is that people who speak

[13] *See id.*

[14] *See, e.g.*, David S. Evans et al., *Did Microsoft Harm Consumers? Two Opposing Views*, AEI-Brookings Joint Center for Regulatory Studies, 2000. (Dr. Evans and Dr. Schmalensee argue that Microsoft's behavior *did not* harm consumers, while Dr. Fisher and Dr. Rubinfeld provide the opposing view.)

natural languages such as English need to communicate with hardware that recognizes only the difference between high and low voltage levels.[15] This communication requires a series of translators that successively convert the human input "downward" into increasingly formalized "languages" or logic systems, eventually to voltage levels, and then back "upward" into natural language.[16] All software tackles some part of this translation chain.[17]

These translations define two distinct tasks that must meet at a common language somewhere in the middle.[18] The first is for the human user to learn how to communicate with a user interface.[19] The second is to connect that interface to the underlying hardware.[20] Computer scientists begin this second task by coding high and low voltage levels as 1s and 0s, respectively, to translate from voltage into binary digits (bits).[21] They then group the bits together to generate more complex number sets, numeric codings of the alphabet, and eventually "high level" programming languages.[22] Software engineers can then program in these high-level languages to move the upward translation chain all the way to the user interface.[23]

This user interface thus marks the "translation frontier" between human and computer.[24] Though its language is alien both to natural language speakers and to voltage readers, it can be translated into either language.[25] While people always needed a full translation chain to use computers, the distance between the user interface and English has narrowed considerably in recent years.[26] In the 1960s, virtually all computer users were technically trained professionals who were personally proficient in a specialized computer language.[27] By the 1990s, computer scientists had moved the frontier so far upward that many accomplished computer users today speak no language more technical than

[15] *See* Bruce Abramson, *Promoting Innovation in the Software Industry: A First Principles Approach to Intellectual Property Reform*, 8 B.U. J. SCI. & TECH. L. 75, 114-115 (2001).

[16] *See id.* at 115.

[17] Most of this description of software and computer science is common knowledge, of the sort covered in any good elementary programming text. For a somewhat extended treatment in terms accessible to a legal audience, *see id.* at 113-116 (2001).

[18] *See id.* at 115.

[19] *See id.*

[20] *See id.* at 114.

[21] *See* Abramson, *supra* note 15, at 115 n.152.

[22] *See id.* at 115.

[23] *See id.*

[24] *See id.*

[25] *See id.* at 115.

[26] *See id.*

[27] *See* Abramson, *supra* note 15.

"point-and-click."[28]

As user interfaces targeting the non-specialist became more powerful, software engineers began to marry them to the underlying operating systems to become "platforms" that disguise sophisticated translation chains as simple instructions.[29] These platforms also allow individual applications (such as word processing programs and games) to communicate directly with the underlying hardware. As a result, a software engineer developing a new game, for example, need only design his program to communicate with the platform through its through Application Programming Interfaces (APIs). The APIs continue the series of translations further downward toward voltage levels. A small subset of these applications qualifies as "middleware."[30] While most of the programs visible to human users require inputs directly from those users, middleware programs expose their own APIs and thus can also receive inputs from yet other applications.[31]

Periodically, a new generation of the platform "evolves" the frontier upward by incorporating selected middleware and applications into the pre-existing platform.[32] Each upward evolution affects the worlds of software and computing in at least two significant ways: First, it enables a broader group of people to become software developers and users by reducing the requisite amount of specialized knowledge. Second, it unleashes a wave of competitive innovation on the set of features that will define the next generation of software development. In cases involving monopoly products, this evolution can unleash a third important effect: it can reduce if not eliminate innovation and competition with respect to the subsumed features.[33] When the pre-existing platform is a dominant product, all further innovation on the newly subsumed feature must be compatible with the implementation that the monopolist adopted; the platform monopolist thus controls innovation.[34] In a

[28] *See id.*

[29] *See id.* at 115-16.

[30] *See id.* at 116.

[31] *See id.*

[32] *See id.* at 143-44.

[33] *See, e.g.*, *Giving the Invisible Hand a Helping Hand*, THE ECONOMIST, Nov. 9, 2002 at 14. ("What is striking is how little innovation there has been in the bits of the market that Microsoft dominates, and how much where it has little influence. Operating systems, web browsers and word-processing software all look much as they did five years ago. But not many people are using five-year-old mobile phones, handheld computers or music-sharing software.").

[34] One general feature of a "standard" is that it forces all subsequent developments into conformity. Products incompatible with the standard defining a network cannot work with that network. When a standard defines the only existing network, products incompatible with that standard will not work anywhere. When that dominant standard is proprietary, its owner is a monopolist capable of forcing all potential innovators and product developers to

competitive environment, market forces attenuate the scope of this third effect because developers can still introduce new features that run upon competing platforms. If the evolution integrated features prematurely, the evolved frontier will suffer, as innovation on its competitors will allow them to surpass it in technological sophistication, in product quality, and likely in market performance. In a monopolized platform market, no such constraint exists. Premature integration will deter innovation.

The combination of these effects defines the direction that the software industry will take following each new generation of platform development. It also determines the dividing line between engineering and business strategy—both of which can drive integration decisions. The basic principles of software engineering, however, often make it easy to determine which motivator drove a given decision. Many of these basic principles have changed little since the early days of the computing; elementary programming classes have included them in their curricula for decades. Fred Brooks's *The Mythical Man-Month,*[35] written in 1974 and generally regarded as "the" classic work on software engineering, provides an excellent illustration of this stability.[36] Brooks drew upon practical lessons that he had learned in the 1960s in his roles as a project manager first for the important IBM System/360 family of mainframes, and then for the massive OS/360 operating system (the platform for these machines).[37] Twenty years later, his "Anniversary Edition" affirmed all of his basic messages and most of his subsidiary lessons.[38]

One key principle of software engineering—and the one central to understanding the harm latent in premature integration—is that good software design develops "modules," "components," or "objects" that fit together like pieces of a puzzle.[39] Partitioning complex systems into modules helps tame their complexity.[40] It also minimizes the damage when a developer discards an

choose between conformity to the standard or exiting the market. For a good discussion of strategic behavior in setting or conforming to standards, *see* CARL SHAPIRO & HAL R. VARIAN, INFORMATION RULES, (Harvard Business School Press 1999) at ch. 3.

[35] FREDERICK P. BROOKS, JR., THE MYTHICAL MAN-MONTH (1st ed. 1975).

[36] *See id.*

[37] *See id.*

[38] *Id.* at ch. 19.

[39] *See* ED YOURDON ET AL., MAINSTREAM OBJECTS, (Prentice Hall, 1995) at 79-81 (delineating criteria for determining the quality of a model designed for implementation as software and stating that modularity, in particular, plays a prominent role in determining quality and improving maintainability).

[40] "For every 25 percent increase in problem complexity, there is a 100 percent increase in the software solution." Consequently, partitioning the problem into simpler subtasks reduces the complexity of the solution. ROBERT L. GLASS, FACTS AND FALLACIES OF SOFTWARE ENGINEERING, (Addison-Wesley, 2003) at 58-60.

existing bit of software and rewrites it in light of new discoveries.[41] In other words, "plug-and-play" modular systems are easier to update, modify or fix than those integrated into a monolithic whole. As the software development community gains a better understanding of challenges that it once considered cutting-edge, and as the implementations of code addressing those tasks become robust and stable, developers *may* choose to integrate two previously independent functions in order to enhance some aspect of system performance.

The integration of ill-understood, immature code fragments likely to require further modification and debugging is invariably a bad idea.[42] From the perspective of software engineering, features currently captured by either middleware or applications may only be potentially "ripe" for integration into the platform if they are robust, well understood, and unlikely to undergo further change. In economic terms, these features have converged to a *de facto* standard; few further innovations are likely.[43] Even then, a platform developer who chooses to integrate such features into a next-generation platform might foreclose competition in the market for that feature, but at least such integration would be unlikely to hamper broader software innovation. Decisions of this sort may be consistent with some software engineering practices—though even then, potential gains in the areas of up-front programming time and application running speed will often need to be substantial to compensate for these added costs.

Figure 1 on the following page illustrates the broad superiority of modular programming and demonstrates that every design decision embodies a series of performance tradeoffs. Actions that improve performance along one metric invariably incur costs along some other dimension. When viewed from the perspective of modern technology, Figure 1 illustrates four basic points: (i) There are areas in which commingling, integration, or monolithic design might make sense; (ii) These areas are likely to be quite rare; (iii) Their incidence dwindles with each generation of software technology; and (iv) Modern software economics suggests that the costs of commingling will invariably exceed its benefits. Today, Microsoft's legal team stands virtually alone in favoring complex monolithic programming.[44] Component-based programming

---

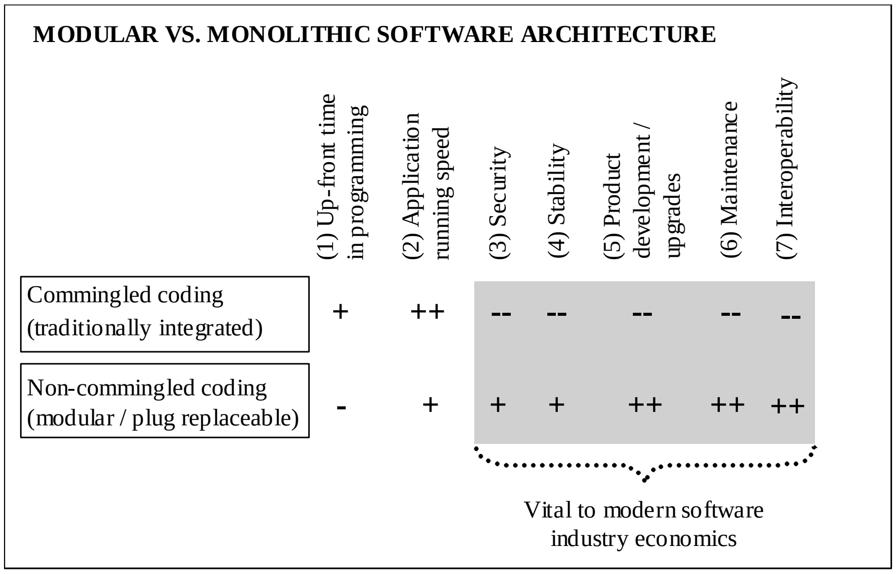
[41] "It is possible to claim that maintenance is a more difficult task than development." Updating software in light of new discoveries is the quintessential maintenance task. It is invariably more difficult to update existing code than it is to develop new plug and play modules. *Id.* at 120.

[42] *See* BROOKS, *supra* note 35 at 142-147 (describing top-down design and componentwise debugging as critical elements in the design procedures long used by the best programmers). *See also* Niklaus Wirth, *Program Development by Stepwise Refinement*, 14 COMMUNICATIONS OF THE ACM 221-227, 1971.

[43] *See generally* Shapiro & Varian, *supra* note 34, at ch. 8.

[44] It is difficult to find even a single contemporary technical reference extolling the virtues of integrated programming. Statements explaining the importance of modularity

is the industry standard; college textbooks,[45] as well as Microsoft's own publications[46] and statements to developers,[47] tout the advantages of plug-and-play software components.



**MODULAR VS. MONOLITHIC SOFTWARE ARCHITECTURE**

| | (1) Up-front time in programming | (2) Application running speed | (3) Security | (4) Stability | (5) Product development / upgrades | (6) Maintenance | (7) Interoperability |
|---|---|---|---|---|---|---|---|
| Commingled coding (traditionally integrated) | + | ++ | -- | -- | -- | -- | -- |
| Non-commingled coding (modular / plug replaceable) | - | + | + | + | ++ | ++ | ++ |

Vital to modern software industry economics

abound. *See, e.g.*, Dwayne Phillips, *Information Hiding in C via Modular Programming*, C/C++ USERS JOURNAL, Jan. 1998, at 57 ("Modular programming requires a little extra work from the programmer, but pays for itself time and again during maintenance . . . "); Alan Radding, *Application Servers Fuel E-business*, INFORMATIONWEEK, June 19, 2000, at 111, *available at* http://www.informationweek.com/791/eai.htm (Building business logic "as reusable components," and thus automating low-level processing functions into components, helps developers eliminate "as much as 70 percent of the coding in an application"); Eric Sanchez and Joe Fenner, *EAI Users Go With The Flow*, INFORMATIONWEEK, Mar. 26, 2001, *available at* http://www.informationweek.com/830/eai.htm (In some areas of software engineering relating to enterprise application integration, "modular pieces of code eliminate 75 percent of the work associated with performing such integration through custom programming.").

[45] *See generally* GEORGE T. HEINEMAN & WILLIAM T. COUNCILL, COMPONENT-BASED SOFTWARE ENGINEERING (Addison-Wesley 2001).

[46] *See generally* STEVE MAGUIRE, WRITING SOLID CODE 87-109 (Microsoft Press 1993) (emphasizing the importance of single-purpose components and interfaces to reduce programming errors).

[47] *See, e.g.*, Michael Vizard and Karen Moser, *Plug and Play is Years Away*, PC WEEK, Apr. 11, 1994, at 45 ("Microsoft Corp. is painting a rosy future for developers, based on plug-and-play software components using object technology. . . .").

**FIGURE 1: A COMPARISON OF THE ENGINEERING STRENGTHS OF MODULAR AND MONOLITHIC DESIGN**

Nevertheless, it is also important to differentiate mature from immature applications. The less mature the application, the greater the risks inherent in integration—and thus the more overwhelming the superiority of modular design.[48] The need for continuous revisions, upgrades, and patches to immature applications implies that even developers are unaware of every aspect of their program's behavior. As a product matures, some of that initial integration risk will attenuate, though some will always remain; even with mature applications, the promised benefits of integration must be significant to overcome the risks inherent in hard-wired cross-module communication. Thus, *any* decision to integrate an immature application is suspicious, and antitrust authorities must learn to differentiate between the circumstances in which engineering design might justify integration and those in which business strategy provides the only plausible explanation.

Under normal circumstances, market discipline will ensure that developers who choose an integration that appears premature possess a good faith belief that they are developing superior products. In a competitive platform market, a developer who integrates an application into its platform prematurely risks destabilizing its platform. A reputation for instability and bizarre side effects could drive potential consumers to alternative platforms. But in the absence of platform competition, the monopolist may see little risk in degrading its platform's performance by integrating premature middleware strategically; market forces cannot provide the discipline needed to elevate engineering concerns over strategy. More importantly, integration by a monopolist effectively adopts the integrated software as the *de facto* standard. Alternative approaches will have a hard time competing on their merits. This premature integration will foreclose new approaches and restrict innovation of improvements in the integrated approach. It will elevate the platform monopolist's proprietary idea to the status of a standard before it has earned that promotion on its merits—thereby serving the monopolist's strategic business objectives at the expense of the consuming public's interest in innovative product improvements.

These basic principles of software engineering have powerful implications for the enforcement of antitrust laws governing technological bundling in software markets. Although authorities are understandably reluctant to second-guess product engineering decisions, they need to ensure that product engineering rather than strategic marketing, truly drives integration decisions

---

[48] *See*, BROOKS, *supra* note 35, at 142-147.

before exercising such deference.[49] Software capabilities embodied in middleware or in applications that are changing rapidly; that are subject to ongoing innovation; that competing developers distinguish by racing to introduce innovative new features; and that these competitive vendors sell as heterogeneous products, invariably deserve to retain their independence and modularity. Antitrust authorities should be skeptical when they see monopolists integrating middleware of this sort, and allow it to stand only if the monopolist can provide a compelling engineering justification. The factual inquiries underlying this approach, while nontrivial, lie well within the realm of questions that public authorities, supported by expert testimony, can adjudicate competently.

## III. THE LEGAL ANALYSIS OF INTEGRATION

Virtually everyone who has considered the unique challenges that software poses to antitrust inquiries into tying, bundling, and integration, has concluded that technological maturity is a central—if not *the* central—issue. Legal and economic scholars have noted that the tests and rules that the U.S. Supreme Court has applied to standard products and markets do not apply easily to software markets: Professor Lawrence Lessig noted that a strict application of those general rules could be over-inclusive in ways that reduced consumers' access to innovative, powerful products.[50] The D.C. Circuit Court of Appeals reached a similar conclusion when it announced that its newly required rule-of-reason analysis for tying cases involving platform software explicitly left open the broader question of general software integration.[51] Antitrust expert J. Gregory Sidak has proposed a decision rule for courts investigating software integration that begins with a preliminary question: "Is the market technologically mature or technologically dynamic?"[52] He reserved his follow-up questions for dynamic software markets; they look at plausible consumer benefits, probable reductions in competition and the consequent consumer harm, and the net effect of the two.[53]

From an economic perspective, Sidak's rule asks the right questions. His

[49] "As a general rule, courts are properly very skeptical about claims that competition has been harmed by a dominant firm's product design changes." United States v. Microsoft Corp., 253 F.3d 34, 95 (D.C. Cir. 2001).

[50] *See* Brief of Amicus Curiae Professor Lawrence Lessig at 24, *United States v. Microsoft Corp.*, 87 F. Supp. 2d 30 (D.D.C. 2000). Professor Lessig is generally viewed as unsympathetic to Microsoft's positions.

[51] *See Microsoft*, 253 F.3d at 95.

[52] *See* J. Gregory Sidak, *An Antitrust Rule for Software Integration*, 18 YALE J. ON REG. 1, 28 (2001). Mr. Sidak has served as a consultant to Microsoft and is generally viewed as sympathetic to Microsoft's positions.

[53] *See id.* at 28-33.

answers, however, invert the lessons of basic software engineering. His analysis implies that the courts should presume that engineering concerns guide all integration decisions unless and until proven otherwise.[54] But the principle of modular design suggests that premature integration, though not always wrong, is always suspicious.[55] As a result, when a court determines that a platform monopolist integrated a technologically dynamic application's functionality into the platform, the court should presume that *strategic* considerations motivated the integration. The court should then allow the defendant to justify its decision—essentially placing the burden on the monopolist. If the platform monopolist can persuade the court that its integration provided consumers with benefits that could not have been achieved via a modular design, and that these benefits were significant enough to overwhelm the consumer harm implicit in reduced competition, the court should allow the integration to stand. In the absence of such a showing, however, the court should view integration in a technologically dynamic market as presumptively strategic, and thus anticompetitive.

All told, the *engineering* case for integration does not depend upon the competitive or monopolistic nature of the platform market. The appropriate engineering approach is generally to maintain modularity at least until the applications are as mature as the underlying platform—and then to integrate if and only if the projected performance enhancements warrant the risk of unintended side effects.[56] Violations of this principle suggest decisions guided by business strategy, not by engineering considerations. Authorities can convert this technical observation into a legal rule by presuming—based on market evidence—that *all integration decisions that appear to be strategic are, in fact, strategic*, unless and until the developer can present a compelling demonstration of the net engineering benefits of integration.

## IV. MICROSOFT'S INTEGRATION STRATEGY

Though most of our discussion to this point has been both theoretical and general, we clearly directed it towards Microsoft and its ever-expanding Windows monopoly. The basic principles of modular design and software maturation explain why only some of Microsoft's historical behavior deserved deference. Microsoft's integration of Windows into DOS appears to have been consistent with sound software engineering principles. Microsoft first launched Windows in 1985 as a middleware product—a graphical interface

[54] *See id.* at 28.

[55] *See supra* pp. 12-13.

[56] Product quality and short-term marketability often point in different directions. The discussions above outlined some basic principles of software engineering that professionals have observed lead to quality software products. Deviations from these principles suggest the elevation of some other set of concerns above those of engineering design.

sitting atop DOS—at a time when the DOS market was still competitive, and Microsoft was but the largest of several competitors.[57] Many industry observers felt that early versions of Windows were buggy and unstable.[58] In response, Microsoft upgraded and updated Windows much more often than its stable DOS operating system. Roughly five years later, Windows 3.0 marked its first powerful release.[59] Five years after that, Windows 95 became the first integrated Windows/DOS platform. Six years later, Microsoft created an entirely new code base for its seamless integration of Windows and DOS capabilities in Windows XP.

Microsoft's bundling of IE into Windows 95 in the early days of browsers, when innovative ideas were still brewing and browser technology was neither stable nor mature, was *inconsistent* with these basic principles. This premature integration had two related effects on technological development. First, by leveraging its platform monopoly into the browser market, Microsoft curbed innovation on browsers other than IE—and thus curtailed competition.[60] Second, by neutralizing the competitive threat from Netscape and Java, Microsoft eliminated the likely innovation focused on an Internet-centered platform—and thus guaranteed that virtually all subsequent browser innovation would have to arise from, or be channeled through, Windows/IE.[61] This

[57] Software magazines' reviews of competing products, for example, often compared then to Microsoft's. As late as 1990, they were able to opine that "[t]he latest incarnation of DR DOS, Digital Research's MS-DOS clone, is an innovative and intriguing operating system that's thoughtfully designed. Version 5.0 is also packed with the extra features that Microsoft's own operating system should have." Caldera v. Microsoft, 72 F. Supp. 2d 1295, 1295 n.1 (D.Utah 1999), *citing* Stan Miastkowski, *A Cure for What Ails DOS*, BYTE, Aug. 1990, at 107.

[58] *See, e.g., id.*

[59] "Microsoft shipped Windows 3.0 on May 22. Compatible with DOS programs, the first successful version of Windows finally offered good enough performance to satisfy PC users. For the new version, Microsoft revamped the interface and created a design that allowed PCs to support large graphical applications for the first time. It also allowed multiple programs to run simultaneously on its Intel 80386 microprocessor." *Computer History Museum Timeline*, available at http://www.computerhistory.org/timeline/timeline.php?timeline_category=sl (last visited July 3, 2004). *See also* STAN J. LIEBOWITZ AND STEPHEN E. MARGOLIS, WINNERS, LOSERS, AND MICROSOFT, (The Independent Institute, 2001) at 143-146 (discussing the speed with which users switched from DOS to Windows following the 1990 release of Windows 3.0).

[60] "Microsoft took actions that could only have been advantageous if they operated to reinforce monopoly power." United States v. Microsoft, Corp., 84 F. Supp. 2d at 28 (D.D.C. 1999).

[61] "Microsoft focused its antipathy on two incarnations of middleware that, working together, had the potential to weaken the applications barrier severely without the assistance of any other middleware. These were Netscape's Web browser and Sun's implementation of the Java technologies. . . ." *Id.*

second effect was particularly harmful. By design, Windows was a translation frontier appropriate for the concerns of business/office users. It thus incorporated design decisions presumably appropriate to those tasks. The Netscape/Java frontier that Microsoft blocked would have embodied decisions appropriate to the Internet. The existence of two (or more) translation frontiers, each optimized to a different set of uses and concerns, would have allowed developers to avail themselves of different feature sets in new and innovative ways that would almost certainly have rendered that alternative frontier both powerful and profitable, though it is obviously impossible to predict precisely what revolutionary new applications might have developed in this environment.

Microsoft's premature integration of IE into Windows was but part of a general trend. At various points in Microsoft's history, it has had to make strategic choices to favor some products over others. Though each critical juncture led to an internal debate, strategic concerns have always trumped engineering design. Microsoft's own developers reportedly often felt that "the company sacrificed innovation for 'strategy,' the complex set of hooks and lock-in techniques that Gates invariably insisted on to steer customers toward Microsoft's end-to-end product line and keep them from being able to competitive products—and which customers hated for the very same reason. . . . The 'strategy tax' could be deeply demoralizing."[62] This elevation of strategy over engineering should come as no surprise: corporations strive to maximize profits, and strategic considerations are central to their calculations. In Microsoft's case, the best way to maximize profits is to preserve and to extend its valuable Windows monopoly.

Microsoft's neutralization of the Netscape/Java threat thus served its own strategic interests while limiting the creative opportunities available to developers at precisely the time that the Internet and the Web were becoming important. Since then, and especially with the 2001 launch of Windows XP, Microsoft has moved to gain control of nascent platforms centered on entertainment and communications.[63] Its chosen methods mirror those used so successfully with IE: the premature integration of both the Windows Media Player (WMP) and Windows Messenger into the platform.[64] These integration decisions played a key role in the European Commission's recent ruling on

[62] DAVID BANK, BREAKING WINDOWS 96 (Free Press, 2001). *See also United States v. Microsoft Corp.*, 84 F. Supp. 2d 9, 51 (D.D.C. 1999) (quoting Microsoft executive James Allchin's explanation of the strategic need to bundle IE into Windows).

[63] *See, e.g.,* John Burgess, EU, Microsoft Cannot Agree on Settlement, THE WASHINGTON POST, March 19, 2004, A1.

[64] Microsoft's Passport and .Net initiatives, both launched concurrent with XP, apply a similar but more complicated strategy to e-commerce. A discussion of these more complicated parallels lies beyond the scope of this article. *See id.*

Microsoft's behavior, currently on appeal.[65]

Regardless of the outcome of the appeal, these integrations reveal a consistent strategy designed to channel all innovation towards Windows. By extending its monopoly into one software market after another, Microsoft ensures that the absorbed market's future remains compatible only with Windows. In so doing, Microsoft simultaneously protects Windows' platform monopoly, precludes the emergence of a nascent threat, and eliminates competition in a previously standalone market. Decreased competition from abuses like technological bundling likely reduces the talent pool and the capital invested in innovation. Monopoly maintenance precludes the innovative exploitation of interactions between human and machine that do not pass through Windows. Because Microsoft's strategic interests are best served by reducing the innovation that it cannot control, unfettered discretion to integrate software into Windows strategically will curb innovation and lead to weaker software products. Antitrust authorities should thus view Microsoft's integration decisions as strategic, and evaluate them as they would other decisions born of business strategy. They are not due the special deference that courts typically accord to engineering design decisions.

## V. THE OPEN QUESTION

Although various courts have commented on Microsoft's claim to the unfettered liberty to integrate,[66] no court has yet announced a workable antitrust rule differentiating permissible from impermissible integration. The

[65] *See id.* The European Commission has been investigating Microsoft for several years. The scope of its investigation covers a number of issues that arose subsequent to the U.S. government's filing of its suit, including integration and leveraging allegations involving the releases of the Windows 2000 and Windows XP operating systems. On August 6, 2003, the Commission issued a press release, announcing: "The European Commission has given Microsoft a final opportunity to comment before it concludes its antitrust probe. The Commission has gathered additional evidence from a wide variety of consumers, suppliers and competitors. This evidence confirms and in many respects bolsters the Commission's earlier finding that Microsoft is leveraging its dominant position from the PC into low-end servers and that Microsoft's tying of Windows Media Player to the Windows PC operating system weakens competition on the merits, stifles product innovation, and ultimately reduces consumer choice. The Commission also invites Microsoft to submit its comments on a series of remedies it intends to impose in order to bring the antitrust infringements it has identified to an end. As this complex investigation draws to a close, the Commission will continue to ensure a meticulous respect of due process. Therefore, the Commission has addressed to Microsoft a final Statement of Objections." Press Release, European Commission, Commission gives Microsoft last opportunity to comment before concluding its antitrust probe (Aug. 6, 2003), *available at* http://europa.eu.int/rapid/start/cgi/guesten.ksh.

[66] *See supra* notes 5-8.

absence of such a rule defines perhaps *the key* challenge facing courts and regulators currently contemplating antitrust issues in the software industry—and in particular, those contemplating Microsoft's actions. This issue is unlikely to disappear until a viable rule emerges.

Judge Kollar-Kotelly chose not to announce such a rule and instead focused entirely on narrow claims concerning software markets that Microsoft had already been adjudicated to have damaged.[67] The behavioral restrictions that she imposed on Microsoft are thus unlikely to provide adequate protection for future software markets. Judge Kollar-Kotelly's analysis of Microsoft's premature integration of IE into Windows, for example, relied on the software markets of late 2002, rather than those of 1995, when the integration began.[68] But even assuming that her cost-benefit analysis was sound with respect to the browser market, it remains inapplicable to either media players or messengers—two software markets whose futures Microsoft's current behavior is shaping.[69] The existence of competitive products with heterogeneous features indicates that media player and messenger technologies are still the focus of innovation driven by competition. These factors suggest that Microsoft's integration of WMP and Messenger into Windows—like its earlier integration of IE—is strategic, not technological.

Beyond Judge Kollar-Kotelly's analysis, though, the behavioral restrictions she placed on Microsoft do little to alter either strategic or technological realities because they do not constrain discretionary integration.[70] Many software companies (including Microsoft) guard their source code as a valuable trade secret. Attempts by anyone lacking access to source code—either computer vendors or end users—to replace a truly integrated function with a competing product could either cripple system performance or strand extraneous bits of conflicting code. In either case, users are likely to conclude that their new application works poorly, and may even notice degraded overall system performance. Most users are likely to blame these problems on the newly installed applications. This problem is endemic to any market in which a software monopolist is able to integrate competitive functionality into the secret source code of a dominant product.

In terms of the direct harm imposed by premature integration, though, the community of developers whose innovation and creativity impel technology forward may suffer even more than consumers. Microsoft's technological bundling achieves two goals. First, by ensuring that all Windows users also possess the integrated products, Microsoft guarantees itself a 100 percent reach

[67] *See supra* notes 10-13.

[68] *See id.*

[69] These markets' futures are among the EU's central concerns. *See e.g.,* Burgess, *supra* note 63.

[70] *See generally Microsoft,* 231 F. Supp. 2d 203 (D.D.C. 2002).

into what had been a competitive market. Even a far superior product not distributed with Windows is unlikely to be resident on all Windows machines. Second, if Microsoft is able to further "game" the system by making its own integrated products the only ones that work well with Windows, most Windows users will bias their selection towards Microsoft for reasons other than the merits of the product. From there, the rest of the "applications barrier to entry"[71] will follow logically from the network nature of the software industry. Most content providers, hardware manufacturers, and independent software developers will follow the user base to Microsoft's products. This favoritism will emerge even if Microsoft takes no overt steps to cripple competing products; it is an outgrowth of the ubiquity of the Microsoft products shipped with the Windows OS. The effect will be exacerbated, however, if Microsoft's products are the only ones capable of running well on Windows. Developers will increasingly favor Microsoft simply because rational suppliers will choose to cater to the largest available market. The applications barrier to entry will thus have a leveraging effect that curbs innovative activity in competing products, ensuring that remaining innovation gravitates towards Windows and its newly integrated products. Microsoft's ongoing leveraging of its Windows monopoly into adjacent software markets via discretionary integration is thus likely to harm consumers in two ways: the short-term frustration of crippled third-party software and the long-term reduction in innovative product development.

## VI. CONCLUSION

Antitrust law should prohibit software monopolists—Microsoft as well as niche providers—from integrating new products into their monopoly products prematurely. The marketplace provides a useful proxy for technological maturity: convergence to a standard. The existence of multiple differentiated products with heterogeneous features is indicative of an immature technology still subject to the ferment of competitive innovation. Thus, the appropriate, workable antitrust rule should prohibit software monopolists from integrating products whose features are still subjects of innovation and competition into their monopoly products—or at the very least, should view such premature integration as *presumptively* anticompetitive.

This remedial rule would level the playing field in software markets adjacent to monopoly products. If antitrust law prohibits strategic integration, software monopolists will integrate only when engineering analyses dictate that integration is appropriate—precisely the same guidance that motivates integration in a competitive environment. Monopolists will thus lose an important bit of leverage and become more likely to compete on their product's merits.

[71] *Id.* at 212.

Antitrust authorities need to remind the software world of Judge Jackson's 1997 admonition that "[the] 'unfettered liberty' to impose [a discretionary] idea of what has been 'integrated' into [a monopoly software product] stops at least at the point at which it would violate established antitrust law."[72] They need to conclude that as a rule, the antitrust laws should prohibit software monopolists from integrating powerful new technologies still subject to robust competition into their monopoly products.

[72] United States v. Microsoft Corp., 980 F. Supp. 537, 543 (D.D.C. 1997).