NBER WORKING PAPER SERIES

# ECONOMIC PERSPECTIVES ON SOFTWARE DESIGN: PC OPERATING SYSTEMS AND PLATFORMS

Steven J. Davis
Jack MacCrisken
Kevin M. Murphy

Working Paper 8411
http://www.nber.org/papers/w8411

NATIONAL BUREAU OF ECONOMIC RESEARCH
1050 Massachusetts Avenue
Cambridge, MA 02138
August 2001

Available for download at http://gsbwww.uchicago.edu/fac/steven.davis/research. We have benefited greatly from the assistance of many persons in preparing this study. We thank James Allchin, Hillel Cooperman, Paul Maritz and Todd Nielsen of Microsoft Corporation for taking the time to speak with us about software design issues generally and the evolution of the PC operating system in particular. We thank David Evans, Albert Nichols, Peter Passell, Bernard Reddy and Lynn Shisido-Topel for many helpful comments. Passell also provided valuable editorial assistance, and Shishido-Topel and Grant Govertsen provided much able research assistance. We gratefully acknowledge research support from Microsoft Corporation. The views expressed herein are those of the authors and not necessarily those of the National Bureau of Economic Research.

Economic Perspectives on Software Design:
PC Operating Systems and Platforms
Steven J. Davis, Jack MacCrisken and Kevin M. Murphy
NBER Working Paper No. 8411
August 2001
JEL No. L86, L13, L41

## ABSTRACT

Improvements in the software that provides hardware management, user interface and platform functions have played a central role in the growth and transformation of the personal computer (PC) industry. Several forces shape the design of these "operating system" products and propel their evolution over time, including:

A. The need to efficiently manage the interacting components of PC systems so as to keep pace with rapid advances in computer technologies, simplify computer use and facilitate the development of applications software.
B. The need to maintain compatibility with existing applications while preserving the flexibility to incorporate additional functions that support new applications.
C. The desire to economize on customer support costs and assign clear responsibility for making the interacting components of the PC work together.
D. The desire to bundle multiple software features into a single package so as to more effectively meet the demand for complementary applications or reduce the diversity in product valuations among consumers.

We analyze these forces and the factors that determine whether and when new features and functions are included in commercial operating system products. We also explain how this integration and bundling spurs growth in the PC industry and fosters innovation.

Steven J. Davis
University of Chicago Graduate School of Business
1101 East 58th Street
Chicago, IL 60637,
Chicago Partners, LLC
and NBER
sjd@gsbsjd.uchicago.edu

Jack MacCrisken
Chicago Partners, LLC
516 Greer Road
Palo Alto, CA 94303
MacCrisken@aol.com

Kevin M. Murphy
University of Chicago Graduate School of Business
1101 East 58th Street
Chicago, IL 60637,
Chicago Partners, LLC
and NBER
Murphy@gsbkma.uchicago.edu

# 1. Introduction

This paper considers several economic forces that shape the design of software. We focus on "operating system" software that provides an applications platform and user interface for personal computers, and which is often combined with basic functions that manage the computer hardware. Prominent examples include the Apple Mac OS, IBM OS/2 and Microsoft Windows. Much of our analysis also applies to the design of software applications and other software that coordinate many interacting components of a larger system.

One reason to study the economic forces that shape the design and evolution of operating system products is to better understand the spectacular growth and productivity performance of the personal computer industry. Over the last two decades, the PC has evolved from an expensive and clunky device with a narrow range of applications into an inexpensive technological marvel used by hundreds of millions. The PC has become ubiquitous in data storage, information processing, communication and entertainment activities at the work place and in the home. It is now a major business and consumer product and a key complement to many types of creative activity. Improvements in operating system software, broadly defined, played a major role in this transformation.

A second reason is to build a sounder analytic basis for the treatment of product design issues under antitrust law. Design features of Microsoft Windows were key issues in *United States v. Microsoft Corporation,* the most prominent antitrust case in a generation.[1] According to the government, Microsoft engaged in various anticompetitive

[1] Civil Action 98-1232 (D.D.C. 1998).

actions, including the tying of its web browser to Windows. The presiding trial judge concurred, ordering that Microsoft be split in two and placing tight restrictions on the design of their respective software products.[2] We do not explicitly assess legal tying doctrine, but our analysis suggests that product integration and bundling are often in consumers' interests.[3]

Third, software is inherently malleable in ways that bring product design issues to the fore. Software code can be expanded, modified and combined to add functionality, bundle features and redraw the boundaries between product categories. Moreover, major and minor design changes in software products often have minimal impact on marginal costs. Extra features do not alter the (extremely low) cost of stamping a CD-ROM that contains the code for a software product. In addition, consumers can easily dispose of many unwanted features in software: the relevant code can reside unused on a computer disk indefinitely without significant cost. All told, these characteristics lead to an extraordinary degree of design flexibility in software.

Fourth, and not coincidentally, competition in many software products is exhibited through innovation rather than price.[4] Thus the consumer benefits from

---

[2] As of this writing, the remedy is stayed, pending a review of the case by an appellate court. The District Court's Findings of Fact (11/5/1999), Conclusions of Law and Final Order (4/13/2000) and Final Judgment (6/7/2000) in *U.S. v. Microsoft* are available at http://www.usdoj.gov/atr/cases/ms_index.htm. Economic analyses of the case include Economides (2000), Evans, Nichols and Schmalensee (2001), Fisher and Rubinfeld (2000), Gilbert and Katz (2001), Klein (2001) and Whinston (2001).

[3] *U.S. v. Microsoft* involves allegations of an illegal tie of Microsoft's Windows 9x software, which provides hardware management functions, a user interface and an applications platform, to its Internet Explorer web browsing software. Another recent antitrust case, *Caldera, Inc. v. Microsoft Corp.* (72 F. Supp.2d 1295 (D. Utah 1999)) involves an allegedly illegal tie of the MS-DOS operating system to Microsoft Windows 3.1, which provides a user interface and applications platform. See Hylton and Salinger (2001) for a detailed assessment of tying law and theory that treats both of these cases.

[4] On dynamic competition in software markets, in particular, see Evans, Nichols and Reddy (1999), Evans and Schmalensee (2001) and Liebowitz and Margolis (1999). Other studies of dynamic competition include Gans, Hsu and Stern (2000), Reinganum (1985) and Vickers (1986).

competition largely take the form of product improvements, new product introductions and, occasionally, the creation of whole new product categories.

Finally, the PC operating system is the prime example of a product that derives most of its value from its capacity to function as a platform for other products. And as a platform, the demand for PC operating systems is influenced by network effects. Easy file sharing, widespread familiarity with the same user interface and the compatibility of software applications across computers and computer users are examples of direct network benefits from the use of a common software platform. The greater incentive for software developers to invest in new applications as the number of platform users grows is an example of an indirect network benefit. Thus design characteristics – in particular, the ability to create a common standard and to elicit the development of complementary applications – are critical determinants of market outcomes for platform products.

Section 2 of the paper provides important factual background and highlights the declining cost and growing power of personal computing. Section 3 identifies three basic forces that propel the evolution of commercial operating system products: the need to keep pace with advances in computer-related technology, the need to simplify computer use, and the desire to stimulate new applications for the operating system-as-platform. Section 4 elaborates on design flexibility in software products and discusses alternative concepts of software integration. Section 5 describes "componentized" design architectures for complex software products, analyzes the costs and benefits of componentization, and explains its role in managing the evolution of a software platform. Section 6 explains how the need to manage the interacting components of a PC system drives the continual integration of new features into operating systems. Section 6 also

sketches a theory of how and when operating systems evolve in order to simplify end-user experience with the computer system, facilitate applications development and reduce customer support costs for software vendors. Section 7 discusses demand-based motives for the bundling of software applications and utilities with operating system products, and the implications of bundling for economic efficiency and consumer welfare.

Our study draws on a variety of sources for factual background and analysis. The discovery and trial record in *U.S. v. Microsoft* brought forth a wealth of testimony related to the economic and technological forces that shape the design of operating systems and other software. Several senior executives and software developers at Microsoft kindly answered our questions about software design issues and the evolution of the PC operating system.[5] We also draw on previous research in economics and related fields, especially in our analysis of demand-based motives for software bundling.

# 2. Factual Background

## *2.1 Operating Systems and Platforms*

Every computer requires a central processing unit (CPU) and an operating system (OS). The CPU is hardware, typically one or more microprocessors, that performs basic operations. The OS is software that manages the CPU and other hardware such as the keyboard, monitor, storage media and communication devices. Hardware management functions are often combined with an interface between the user and the computer. The

[5] We interviewed James Allchin, Hillel Cooperman, Paul Maritz and Tod Nielsen at the Microsoft corporate campus in Redmond, Washington on August 24, 1999. At the time of our interviews, Allchin was Senior Vice President for Personal and Business Systems, Cooperman was a project manager for a future version of Windows under development, Maritz was Group Vice President for Platforms and Applications

interface allows the user to access and manipulate files, run programs and operate the hardware, either directly or through instructions generated by applications software. In turn, the hardware management and user interface functions are often combined with a software platform into a single "operating system" product.

A software platform contains Application Programming Interfaces (APIs) that specify how a software developer can access useful modules of code built into the platform. The APIs, and the underlying code modules, enable a software developer to economize on writing new code for applications software. Essentially, the applications software calls on the processing functions built into the platform product, which reduces the need for applications developers to write code that performs routine functions. Microsoft Windows, for example, contains thousands of APIs that can be accessed by software applications and that are relied upon by software developers. In this way, a software platform supports the development and operation of software applications such as word processors, spreadsheets and games.[6]

## *2.2 Expanding Functionality of OS Products*

A striking aspect of the evolution in commercial OS products is the continual integration of new features, many of which began as stand-alone applications. Examples include the graphical user interface, disk management and data compression utilities, memory management utilities, fax and e-mail utilities, support for local area networks,

---

and Nielsen was Vice President, Developer Marketing. Allchin and Maritz were also major witnesses in *U.S. v. Microsoft.*

[6] An OS product can function as a software platform, but another platform can also be layered on top of an OS. For example, Microsoft Windows 3.x consisted of a graphical user interface and applications platform layered on top of the DOS operating system. Later, this graphical user interface was integrated with other operating system functions to become the Windows 9x and Windows NT line of products. A software application can also serve as a platform for other add-on software products, and the hardware in a computer system can be viewed as a platform for running software. See Section II in Evans, Nichols and Schmalensee (2001) for a more detailed discussion of alternative types of software platforms.

integrated audio support and web browsing functions. Software that cost hundreds or even thousands of dollars in the early 1990s is now routinely included with operating system products, and at a small fraction of the original cost.

Why this process has been so relentless – and what it means for competition – is controversial and important. Indeed, the questions received a great deal of attention in *U.S. v. Microsoft*.[7] Undeniably, though, the continual integration of new features into OS products pre-dates the coming of age of Microsoft Windows, circa 1992. All commercially successful OS products aimed at the general computer user in recent decades have expanded functionality over time. This suggests that the impulse to include ever more functions in operating systems reflects fundamental economic and technological forces.

[7] David Farber, Professor of Telecommunication Systems at the University of Pennsylvania's Moore School of Engineering, remarks in his 10/5/98 deposition (page 91) that memory management systems were add-ons in early versions of DOS, but it became efficient over time to "include them integrally" in the OS. John Soyring, Director of Network Computing Software at IBM, discusses IBM's development of Internet browsing software and its inclusion in OS/2 on pages 35, 37 and 38 of his 11/18/98 a.m. live testimony. Avadis Tevanian, Senior Vice President of Software Engineering at Apple Computer Corporation, discusses the integration or bundling of Internet functionality with the Mac OS on pages 37-38, 43, 47 and 65-66 of his 11/5/98 p.m. live testimony. James Gosling, Vice President and Sun Fellow at Sun Microsystems and Chief Scientist of the Java Software Division, discusses "built-in web support" for a web-enhanced version of the Solaris OS on pages 34 and 36 of his 12/9/98 p.m. live testimony. Steven McGeady, Intel Vice President and participant in Intel's early Internet and Java development efforts, expresses the view in his 10/8/98 deposition (pages 59-61) that multimedia software should be built into standard PCs. Glenn Weadock, President of Independent Software, Inc., states in his 1/8/98 deposition (pages 103-104) that the definition of basic OS functionality has evolved over time. Edward Felten, Assistant Professor of Computer Science at Princeton University notes with approval in his 12/14/98 a.m. live testimony (page 44) that new versions of software products often have more functionality and give users more choice. William Harris, President and CEO of Intuit, Inc., also remarks upon the expanding functionality of the OS over time in his 1/4/99 a.m. live testimony (pages 50-51). According to Harris, the benefits to Intuit of expanded OS functionality include additional code for modem support and the management of printer drivers. Harris also notes (pages 51-52) that Intuit itself has expanded the functionality of its software products over time by integrating new Web functionality and by bundling different application products together. Here and below, all citations to live, deposition and filed testimony refer to *U.S. v. Microsoft*, unless otherwise noted. Written testimony and transcripts of oral testimony by Microsoft witnesses, along with Microsoft's legal filings, are available at http://www.microsoft.com/trial/mswitness/default.asp. Court and government filings and the testimony of government witnesses are available at http://www.usdoj.gov/atr/cases/ms_index.htm. The Court of Appeals website at http://ecfp.cadc.uscourts.gov provides links to the filings in the appeals portion of the case.

We identify and discuss three such forces in the next section. To set the stage, we first review the breathtaking pace of technological advance and cost reduction during the PC era.

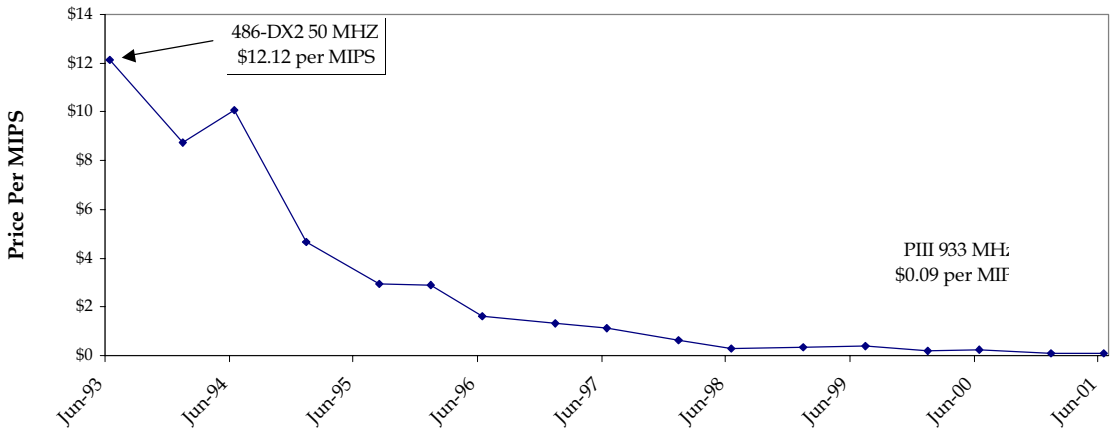
## *2.3 The Growing Power and Declining Cost of Personal Computing*

It is hard to exaggerate the pace of technological change – and the resulting collapse in the cost of computing power – that have transformed the structure of the computer industry several times in two decades.

Start with microprocessors. When IBM introduced the first PC in 1981, its Intel processor contained 29,000 transistors; Pentium III processors, now standard on low-end PCs, contain 9.5 million transistors. Processing power rose commensurately: The Intel Pentium II 450 MHz chip (released in late 1998) performs roughly 4,000 times as many instructions per second as the Intel 8086-5 chip that did the number-crunching for the IBM PC-XT, a popular PC in the mid 1980s. The price for mid-range Intel processors fell from $12.12 per MIPS (millions of instructions per second) in 1993 to $0.09 per MIPS in 2001 (Figure 1).[8]

[8] We compiled the data for Figures 1 through 5 and Table 1 in this section from advertisements in back issues of trade and advertising publications for the PC industry. To construct Figures 1, 2 and 3, we first identified the characteristics of new mid-range PCs at each date (Intel processor, memory amount and hard disk capacity). We then priced each of these components as separate, stand-alone items. The data underlying Figures 1 through 5 and Table 1, and additional information about its construction and sources, are available from the authors upon request.

**Figure 1: The Price of Processing Power in Personal Computers, 1993-2001, Intel Processor MIPS**




Or consider random access memory (RAM), the memory on a chip that can be accessed in billionths of a second. PCs using DOS, the leading OS until the early 1990s, could not utilize more than 640,000 bytes of memory at a time. Today, 64 million bytes of RAM is viewed as a bare minimum in a desktop PC. The price per megabyte of RAM fell from $880 in January 1984 to just 38 cents in June 2001 – an average rate of decline of 44 percent annually (Figure 2).

Much the same has happened with "hard drive" storage. In the late 1980s, 20-megabyte drives were widely viewed as adequate for the typical home or office desktop. By mid 1999, even cheap PCs came with drives that held at least 4,300 megabytes and that offered much faster data access than a decade earlier. The price per megabyte of storage fell from $199 in January 1983 to a mere half penny in June 2001 – an average annual rate of decline of 57 percent (Figure 3).

**Figure 2. The Price of Random Access Memory (RAM) in Personal Computers, 1984 -2001, Log Scale**

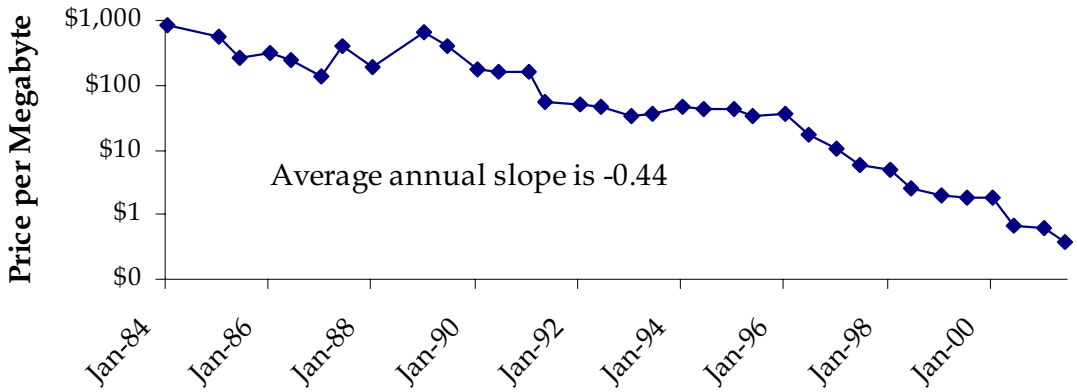
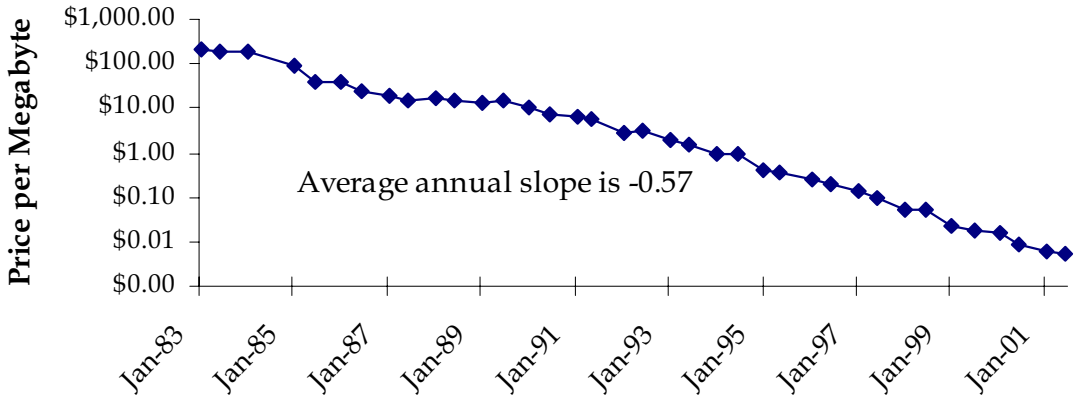


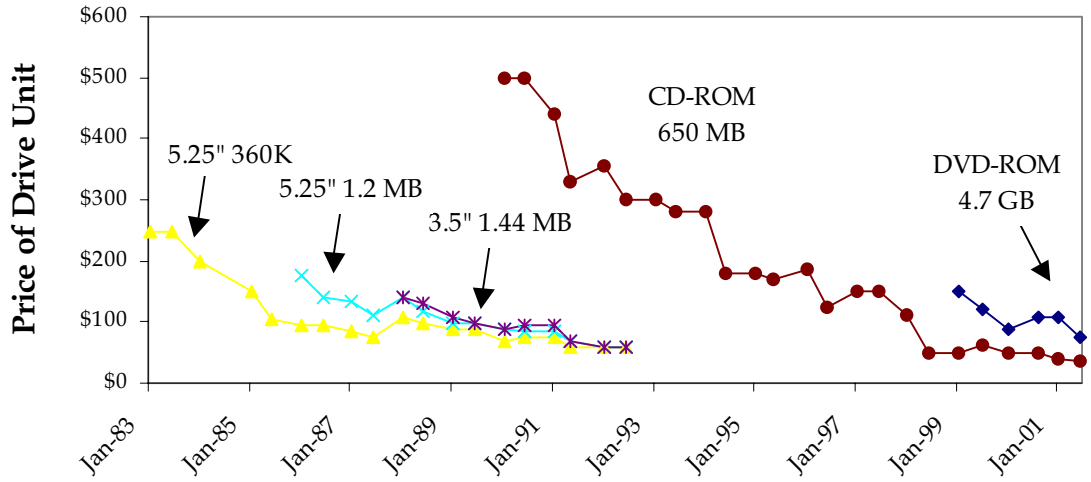

**Figure 3. The Price of Hard Disk Storage in Personal Computers, 1983-2001, Log Scale**




Alternative forms of storage have proliferated, even as prices plummeted (Figure 4). "Floppy" disk drives, once the only form of storage for PCs, used 5-1/4 inch disks that

held a mere 160 kilobytes in 1981.[9] By 1983, floppy capacity had increased to 360 kilobytes at a cost of $708 per megabyte for the drive unit. The next big advance in secondary storage was a 1.2-megabyte disk at a cost of $145 per megabyte for the drive unit. CD-ROM drives, introduced in 1990, hold over 600 megabytes, and the "read only" storage capacity costs a mere 76 cents a megabyte. Storage capacity for DVD-ROM devices, introduced on PCs in 1998, cost less than a penny per megabyte. Over the 1983-1999 period, the price per megabyte in secondary storage devices fell at an average rate of 62 percent annually.

**Figure 4. The Price of Secondary Storage Devices for Personal Computers, 1983 - 2001**




The ongoing revolution in printer technology has been equally dramatic, if harder to quantify. In 1984 a slow, noisy, low-resolution "dot-matrix" printer cost about $600 and, as a practical matter, printed only text. The black and white laser printers of the early

[9] In 1981, DOS Version 1.0 supported an 8-sector 160 KB floppy (single side). Version 1.1, released in 1982, supported an 8-sector 320 KB floppy (double sided). Version 2.0 (1983) supported a 9-sector 360 KB floppy, and Version 3.0 (1984) supported a 1.2 MB floppy. See "DOS Versions" in Computer Language Company (1999).

1990s were much sharper and printed graphics – but were an order of magnitude more expensive. But in recent years color printers using "inkjet" technology have overwhelmed other technologies in low-end printers. They produce fine color at high resolution – and slower models cost around $100 as of mid 1999. (Table 1)

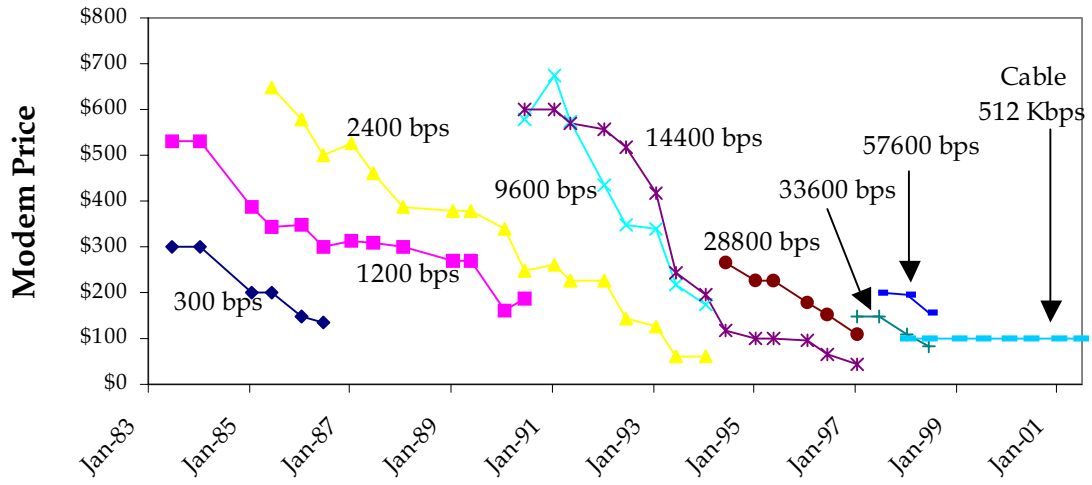
Modems, too, have come of age. In the early 1980s, the typical home or office modem could transfer data at 300 bps (bits per second) and was effectively limited to transmitting text. Today, the standard modem on PCs is capable of 57,600 bps. Cable modems and DSL service can transfer data at several million bps. Prices, no surprise, have fallen sharply. In 1985 the price for a modem device amounted to roughly 27 cents per bps transfer rate, more than 1,000 times the corresponding price for cable modem and DSL devices 15 years later (Figure 5).

## Table 1. Printers for Personal Computers, Prices and Characteristics, 1984-2001

| Debut Year | Category | Product | Specifications | Price | Date |
|---|---|---|---|---|---|
| | B&W Dot Matrix | Epson FX-80 | 160 characters/sec., 9 pin dot matrix | $589 | Jan-84 |
| | | Epson FX-80 | | $429 | Jan-85 |
| | | Epson LX 80 | 100 characters/second, letter quality | $232 | Jun-86 |
| 1984 | B&W Ink Jet | ThinkJet | 300x300 dpi, 1-2 minutes per page | $495 | Oct-84 |
| | B&W Ink Jet | ThinkJet | | $445 | Oct-84 |
| | B&W Ink Jet | ThinkJet | | $435 | Jan-85 |
| | B&W Ink Jet | ThinkJet | | $388 | Jan-86 |
| 1984 | B&W Laser | LaserJet | 300x300 dpi, 8 pages per minute | | |
| | B&W Laser | LaserJet | | $2,995 | Jan-85 |
| | B&W Laser | LaserJet | | $2,376 | Jan-86 |
| | B&W Laser | LaserJet | | $2,289 | Jan-87 |
| | B&W Laser | LaserJet IIIsi | 300 dpi, 17 pages per minute | $3,525 | Jan-92 |
| | B&W Laser | LaserJet 4 | 600 dpi, 8 pages per minute | $1,435 | Jan-93 |
| | B&W Laser | LaserJet 4L | 300 dpi, 4 pages per minute | $669 | Jan-95 |
| | B&W Laser | LaserJet 5L | 600 dpi, 4 pages per minute | $527 | Jan-96 |
| | B&W Laser | LaserJet 5si | 600 dpi, 24 pages per minute | $2,449 | Jan-98 |
| | B&W Laser | LaserJet 1100se | 600 dpi, 8 pages per minute | $399 | Jun-99 |
| | B&W Laser | LaserJet 2100M | 1200 dpi, 10 pages per minute | $629 | Jun-00 |
| | B&W Laser | LaserJet 1100xi | 600 dpi, 8 pages per minute | $349 | Dec-00 |

| | | | | | |
|---|---|---|---|---|---|
| | B&W Laser | LaserJet 1200 | 1200 dpi, 15 pages per minute | $377 | Jul-01 |
| 1991 | Color Ink Jet | Color DeskJet 500C | 300 dpi, 4 minutes per page color | $1,059 | Oct-91 |
| | Color Ink Jet | Color DeskJet 500C | | $719 | Jan-92 |
| | Color Ink Jet | Color DeskJet 500C | | $699 | Jun-92 |
| | Color Ink Jet | Color DeskJet 500C | | $619 | Jan-93 |
| | Color Ink Jet | Color DeskJet 500C | | $396 | Jan-94 |
| | Color Ink Jet | Color DeskJet 500C | | $299 | Jan-95 |
| | Color Ink Jet | Color DeskJet 560C | 300 dpi, 3 minutes per page color | $479 | Jan-95 |
| | Color Ink Jet | Color DeskJet 660C | 300 dpi, 3 minutes per page color | $388 | Jan-96 |
| | Color Ink Jet | Color DeskJet 680C | 300 dpi, 1.5 pages per minute color | $292 | Jan-97 |
| | Color Ink Jet | Color DeskJet 672C | 300 dpi, 1.5 pages per minute color | $199 | Jun-98 |
| | Color Ink Jet | Color DeskJet 420C | 300 dpi, 4 minutes per page color | $119 | Feb-99 |
| | Color Ink Jet | Color DeskJet 420C | 300 dpi, 4 minutes per page color | $99 | Jun-99 |
| | Color Ink Jet | Color DeskJet 812C | 600 dpi, 3 pages per minute | $149 | Jan-00 |
| | Color Ink Jet | Color DeskJet 930C | 1200 dpi, 2 pages per minute | $199 | Jun-00 |
| | Color Ink Jet | Color DeskJet 930C | 1200 dpi, 2 pages per minute | $177 | Jul-01 |
| 1994 | Color Laser | HP Color LaserJet | 300 dpi, 1-2 pages per minute | $6,000 | Sep-94 |
| | Color Laser | HP Color LaserJet | | $5,929 | Jan-96 |
| | Color Laser | HP Color LaserJet 5 | 300 dpi, 2-3 pages per minute | $5,829 | Jan-97 |
| | Color Laser | HP Color LaserJet 5 | | $3,939 | Jun-98 |
| | Color Laser | HP Color LaserJet 4500 | 600 dpi, 4 pages per minute | $2,499 | Feb-99 |
| | Color Laser | Tektronix Phaser 740/N | 600 dpi, 5 pages per minutes | $1,850 | Jun-99 |
| | Color Laser | QMS Magicolor 330 | 600 dpi, 4 pages per minute | $1,849 | Jan-00 |
| | Color Laser | HP Color LaserJet 4550 | 600 dpi, 4 pages per minute | $1,888 | Jul-01 |

**Figure 5. The Price of Modem Devices for Personal Computers, 1983-2001**




All told, this onslaught of technology has put incredible power in the hands of anyone with a thousand dollars to spend.[10] In early 1986, a "high end" PC with a 286 processor running at 8 megahertz (MHz), half a megabyte of RAM and 20 megabytes of hard drive storage cost about $2,800. As of early October 1999, $947 bought a Compaq Presario 5360 PC system with a 450 MHz processor, 64 megabytes of RAM and 4 megabytes of video RAM, 10,000 megabytes of disk storage, a 15-inch high-resolution color monitor, a 56K modem, a 32x CD-ROM, Microsoft Windows 98 and a color ink-jet printer with a resolution of 1200 x 1200 dpi.[11] For an extra 250 dollars, the color printer could be replaced by a multi-function printer, fax, copier and scanner.[12]

[10] Berndt and Rappaport (2001, Table 1) estimate that the quality-adjusted price of desktop personal computers fell at an average rate of 27 percent per year from 1976 to 1999 and more than 35 percent per year in the 1990s.

[11] Based on advertisements in volume 21Q of the PC Mall mail-order catalog. The PC system described in the text reflects PC Mall order numbers 47090, 45775 and 48269 plus the application of a $150 3-piece bundle rebate.

[12] Replacing order number 48269 with number 44476.

The incredible pace of technological change and cost reduction on the hardware side of PC systems has driven many of the developments on the software side. Market leadership in commercial OS and other software categories has undergone several shifts in recent decades, even though the Microsoft Windows family of operating systems has held the leading position since 1993. As emphasized by Evans, Nichols and Reddy (1999), major shifts in the leadership of software categories have often been closely linked to technological developments in hardware. This fact has not been lost on suppliers of OS software, and largely explains why they have integrated new features that keep OS products abreast of technological developments in hardware.

# 3. Three Key Forces that Propel the Evolution of OS Products

## *3.1 Keeping Pace with Technological Advances*

As suggested above, rapid innovation in computers is one of the great technological wonders of our age. And the resulting changes in what users expect from computers compel frequent upgrades in complementary products – especially ones as central to computer performance as operating systems.

By the early 1990s, exponential advances in speed and functionality had made PCs a staple on virtually every desk in the corporate enterprise, while advances in the efficiency and miniaturization of chips, batteries, hard drives and other components made computers easily portable. New innovations in data sharing and interconnectivity, led by Novell and 3Com, gave rise to an entire new industry in PC networks and servers.

In the mid-1990s the face of computing was again changed by technology – this time by the Internet. The Internet powered a surge in the demand for both PCs and

servers and created an entirely new computing environment in the form of the web browser. Yet more recently, hand-held computing devices such as the Palm Pilot have come of age, providing still another hardware platform for data manipulation and communication.

This explosive growth in memory, processing muscle and versatility has created powerful incentives to add new capabilities and features to OS products that keep pace with hardware developments. For example, Microsoft made PC-DOS 1.0 to specifications set for the original IBM PC in 1981. When IBM added a hard drive to the PC a year later, version 2.0 was enhanced to support the additional storage medium. DOS 3.1 adapted the PC for use on local area networks. DOS 3.2 and DOS 3.3 supported the new 3-1/2 inch, 720-kilobyte and 1.44-megabyte floppy disks respectively. And when the 386 microprocessor arrived, DOS version 3.3 was there to support it.[13]

By no coincidence, leadership in the market for OS products has often been up for grabs since the mainframe era of the 1960s. Microsoft emerged as the pre-eminent producer of PC operating system products in the early 1990s. As late as 1992, though, some knowledgeable observers touted IBM's OS/2 product as likely to become the dominant general-purpose OS for PCs.[14]

Consider the ongoing impact of printer technology on operating systems. In the mid-1980s the only high-quality printers for home and office use were very expensive black-and-white laser printers. But color ink-jet printers, introduced in 1991, are now nearly as ubiquitous as microwave ovens. They have opened desktop printing of cards,

[13] Cusumano and Selby (1995, page 148) provide a more detailed narrative of how the first several generations of DOS evolved in response to hardware advances.

brochures and the like to a mass market. And, in tandem with the advent of low-cost digital still cameras, they are making the PC an integral part of photography. To accommodate the plethora of sophisticated new printers from a half-dozen major manufacturers, operating systems have incorporated ever-larger numbers of ever-more-complex printer "drivers." It seems likely that future OS products will incorporate basic software for manipulating and printing photo images.

More generally, so long as the cost of processing power, random access memory and data storage fall at 40-60 percent a year, the practical uses of personal computers will continue to expand rapidly. The ability to manage huge amounts of data and display complex graphics has decentralized publishing from the factory to the desktop. Rapid number crunching and super-cheap memory have transformed the spreadsheet into an all-purpose tool for business, finance and science. The Internet gives PCs access to avalanches of data. And with the expansion of functionality, of course, comes new demands on OS products to manage more hardware and more software with more sophisticated user interfaces.

## *3.2 Simplifying Computer Use*

A second key force behind OS integration is the need to make computers easier to use. Just as standard features on an automobile evolved from bare necessities to include gas gauges, heaters, defrosters and other "extras," computer operating systems have evolved to accommodate consumer tastes and to facilitate computer use. For example,

[14] See Gookin (1992). As Evans and Schmalensee (2001) note, analysts also disagreed over whether OS/2 Warp 3.0 (introduced in November 1994) would prevail over Windows 95 (introduced in August 1995) to become the leading OS product for PCs.

the UNIX operating system, used on a substantial share of engineering and scientific workstations, gradually added a full-screen editor, virtual memory, terminal independence, job control capability and a networking API called "sockets".[15]

And because of the tremendous popularity of Internet applications among today's computer users,[16] web browsers are now routinely integrated into (or bundled with) OS products. Internet Explorer, the web browsing technology developed by Microsoft, is integrated into Windows 98 and later versions of Windows. But Microsoft is hardly alone in packaging a browser with an OS. IBM led this trend, developing its own browser for OS/2. Sun Microsystems' Solaris and Java OS, SCO's UnixWare and Open Server all provide browser functionality by bundling Netscape Navigator with the OS. In addition to bundling their OS products with Netscape Navigator, Sun also supplies the HotJava web browser.

PCs of the past were notoriously frustrating to end-users. Walter Mossberg's *Wall Street Journal* column on personal technology debuted in 1991 "with the contention that personal computers are too hard to use, and that the blame lies not with the people trying to use them but with the supposed geniuses who design the machines and the software that runs on them." Mossberg later lamented, "even rocket scientists are baffled by personal computers. I once got e-mail from a scientist at NASA who works on giant supercomputers all day with aplomb but goes home at night and finds to his great

[15] See Quarterman and Wilhelm (1993), chapter 2.
[16] According to a recent survey by Compaq, a major PC manufacturer and vendor, on-line services are now the number one reason why consumers buy PCs (Ramstad, 1998). Reid (1997) describes the explosive growth of the Internet.

frustration that he can't get the family IBM-compatible PC to run multimedia software correctly."[17]

As the market for PCs has widened, the market pressure to make them easier to use has intensified. Weekend tinkers may revel in the most intricate workings of the PC. For their part, science labs, graduate schools and businesses with sophisticated information management departments may tolerate – even take pride in their ability to control – the PC's eccentricities. But those early markets for computers are saturated. Much of the growth in demand now comes from untutored users – small businesses and households – who expect to plug and play (or work).

One much-welcomed response to this changing market reality has been the expansion of the PC operating system to include elements that previously stood alone, such as software "drivers" to run peripherals ranging from scanners to DVD players. Likewise, the addition of TCP/IP communications software has simplified the once absurdly complicated task of linking computers to network servers, effectively opening the Internet to the untutored.[18]

### *3.3 Attracting Software Developers*

A third key force behind integration is the desire to encourage software developers to create new applications for the operating system-as-platform, thereby enhancing the operating system's value to consumers and giving it a leg up in the market.

---

[17] The quotation is drawn from a 1994 column explaining Mossberg's reason for initiating the column three years earlier.

[18] TCP/IP stands for Transmission Control Protocol/Internet Protocol. Because of its position in the software hierarchy between the Internet application and the communication I/O system (e.g., a modem or LAN), the configuration of a non-integrated TCP/IP package is extremely difficult for the non-technical user.

Platform vendors compete by making it cheap and easy for independent software vendors to develop applications. They accomplish this goal, in part, by providing software building blocks (APIs) that lower the cost of developing complementary software applications.[19] [20] These building blocks include support for soundcards, integrated audio, extended memory, object linking and embedding, scalable fonts, and the like.

The desire to support software developers also explains why commercial OS products incorporate web-browsing support and other Internet functionality.[21] The integration of IE into Windows ensures software developers that a browser with known functionality and specifications is available with Windows. And this, in turn, facilitates the development of simpler, yet more powerful, software applications.

Microsoft's own word processor, Word 97 (and later versions), uses Internet Explorer (or other default browser) to automatically convert any typed-in web address (URL) into a live link. Clicking on the address brings up the web page in the browser, if

---

[19] In our interview with Todd Nielsen, Microsoft Vice President of Developer Marketing, he stressed that independent software vendors want a rich infrastructure, but they don't want to "do the plumbing." He also stressed that Microsoft undertakes extensive efforts to cultivate relationships with independent software vendors, to educate them about new and existing APIs in Microsoft platform products, to encourage their participation in beta testing, and to respond to their concerns.

[20] See Cusumano and Yoffie (1998), pages 76-78, for a discussion of Netscape's use of APIs to promote its Navigator and Communicator products as applications platforms. Typically, platform vendors also devote considerable additional resources to stimulating the development of complementary applications by independent software firms. As of 1995, Microsoft spent about 65 million dollars annually supporting independent software developers and had about 400 technical support engineers who exclusively served independent developers. See Microsoft (1995). According to the same document, Microsoft hosts about 200 developer conferences and seminars per year that are attended by more than 30,000 developers worldwide. Microsoft also engages in an extensive consultation and testing program with developers prior to the release of new OS software. As an example, approximately 7,500 copies of an early version of Windows 95 were distributed to independent hardware and software vendors by December of 1993. About 12,000 beta-1 kits for Windows 95 were distributed by June of 1994. See page 20 of Microsoft (1995).

[21] The integration of Internet Explorer into the Windows 98 OS is one of Microsoft's allegedly anticompetitive actions in *U.S. v. Microsoft,* but other leading suppliers of OS products have also made the integration of web-support features and Internet functionality an important focus of OS development efforts. IBM has "Internet-enabled" OS/2 and other software products and perceives "tremendous value" in doing so. (Soyring, 11/18/98 a.m. live testimony, pages 36-37.) Some Internet protocols and functionality are "built into" the Mac OS, and other Internet functionality is bundled with the Mac OS. (Tevanian, 11/5/98 p.m. live testimony, pages 43, 63, 66 and 67.) Similarly, "built-in web support for the Web-

the user is connected to the Internet. A stand-alone browser can be used to achieve comparable functionality – indeed, many operating system products work this way. But integration ensures the availability of specific browser services for any software application running on the OS. And this, in turn, encourages the development of applications that rely on browser functionality. The designers of Quicken financial planning software, for example, can be confident that every PC user who runs Windows 98 (or later versions) has access to built-in code that enables rapid, automatic information retrieval from the web.

### *3.4 Summary of Key Forces*

To summarize, three important forces propel the evolution of the PC operating system and lead to the continual integration of new features: (a) the need to keep pace with new products and advances in computer technologies, (b) the need to simplify PC use, including the desire to incorporate features that have become part of the basic functionality of PC systems, and (c) the desire to stimulate the development of new and improved software applications that complement the OS and enhance its value as a platform.

These key forces behind the evolution of OS products came through loud and clear in our interviews with Microsoft personnel, as did other forces that we discuss below. Paul Maritz, Microsoft Group Vice President for Platforms and Applications (at the time of our interview) summarized it this way. When thinking about which features to integrate into its platform products, Microsoft asks the following questions:

---

enhanced Solaris" OS includes TCP/IP, NFS protocols and limited URL support. (Gosling, 12/9/98 p.m. live testimony, pages 34 and 36.)

- Does the feature help the user?
- Does it help developers?
- Is a competitor doing it?
- Does it break backward compatibility?

In addition, according to Maritz, widely used, non-differentiated utilities and widely used features with high interoperability requirements tend to get integrated.

# 4. Software Design Flexibility and Integration Concepts

## *4.1 The Inherent Flexibility of Software Design*

Several witnesses in *U.S. v. Microsoft* emphasized the remarkable flexibility of software. Professor David Farber neatly describes this flexibility in his written direct testimony:

> [S]oftware modules are then 'knitted together' into unified programs. That is, each software product is built up from simple low-level routines that are then called by routines at a higher level of composition. Routines at each level are called by yet higher level routines until the desired functionality of the end product is achieved. In this manner, all software is built up layer by layer though the use of often-large numbers of routines, but each with limited complexity. As a result of this layering, software has an inherently malleable and modular structure that gives software developers broad freedom in combining (i.e., bundling) different functions into software products. This malleability also gives a software developer two related types of design freedom: (1) to integrate two separate CD-ROM's because the functions on one particular CD-ROM can be integrated by an OEM or retail end user with functions on another CD-ROM and (2) to determine which functions to include within software sold

> as one product and which to separate and sell as a different product, whether produced by the same or different software developer, for installation and use together by the retail end user. (page 7)

In short, the same functionality can be achieved in many ways – and, in particular, with more or less tightly integrated code.[22]

Flexibility and choice in software design work against efforts to define the boundary between OS and applications software, or between the operating system, narrowly conceived, and the broader notion of an operating environment. Indeed, the inherent flexibility of software design underscores the arbitrary nature of definitions of what is part of the operating system and what isn't.[23]

Several other factors confound attempts to fix these boundaries.[24] First, there is no consensus as to which functions belong within the domain of the operating system.[25]

---

[22] Soyring also remarks upon the great flexibility in software design on pages 66-68 of his 11/18/98 a.m. live trial testimony and on page 11 of his filed testimony. The same theme is implicit in much of the live testimony of Felten and Tevanian, much of the live and deposition testimony of Weadock, and much of the deposition testimony of William Harris (President and CEO of Intuit, Inc.). Weadock explicitly discusses flexibility in software design on pages 48-49 of his 1/8/98 deposition testimony. Harris discusses the contrast between the componentized design of Internet Explorer and the design of Netscape Navigator on pages 9-10, 45 and 73 of his 1/4/99 p.m. live testimony and pages 34-35 of his 9/29/98 deposition testimony. Tevanian discusses how Apple weighed the choice between full technological integration and bundling of Internet protocols and functionality into the Macintosh OS on pages 37-39, 43, 47, 63, 65 and 66 of his 11/5/98 p.m. live testimony and page 47 of his 7/17/98 deposition testimony. A central theme of Felten's testimony is that Microsoft chose to design Windows 98 in a tightly integrated manner, even though technical efficiency does not require such a design.

[23] Weadock stresses that shared program libraries create indeterminacy in delineating a specific code set that defines a software product. See pages 110, 115-116 and 119-120 of his 1/8/98 deposition testimony. He also states (pages 56-57) that "The delineation of what is system software versus what is OS software is a difficult and, except perhaps in legal proceedings, irrelevant distinction." Farber (10/5/98 deposition, pages 103-104) states that it is something of a norm as to what file systems are simple or primitive enough to be considered part of the OS.

[24] Weadock's deposition testimony quite effectively highlights the difficulties. He returns several times to the issue of what constitutes integration with the OS, what properly qualifies as part of the OS, and what functionality is properly deemed within the domain of the OS. However, it is difficult or impossible to distill any general principles regarding these matters from Weadock's testimony, because he advances often-conflicting criteria and so many exceptions to particular rules. See pages 13-14, 46, 48-58, 62, 65, 101-104 and 115-121 of his 1/8/98 deposition testimony. Weadock is aware of these tensions and ambiguities. For example, he remarks that a disk defragmentation utility falls into a gray area as regards OS integration. This type of utility performs a basic housekeeping function, which tends to confer OS status in Weadock's view, but it can also be acquired from third-party vendors, which argues against OS status in his view.

Second, most knowledgeable observers agree that the functions of the PC operating system, however narrowly defined, have grown and will continue to grow. Third, while experts have proposed a variety of tests for whether a particular set of files is actually integrated into the OS, these tests are ambiguous, inconsistent – and often in conflict with common sense views about the functional boundaries of the operating system.[26]

The flexibility of software design matters to our analysis for two reasons. First, we devote only modest attention to the "engineering" aspects of OS design that often loom large in textbook discussions of computer programming techniques.[27] Principles matter in software development, as does the issue of meeting design objectives at reasonable cost. But the inherent flexibility of software code, coupled with the very large size of the market for OS products, implies that the engineering costs of software

---

[25] Some witnesses make this point explicitly. See, for example, page 55 of Weadock's 1/8/98 deposition testimony.

[26] A partial review of expert testimony in *U.S. v. Microsoft* turned up ten distinct criteria for determining whether particular software components (files or groups of files) are properly deemed integrated, "built into" or part of the PC operating system:

1. Whether there exist multiple cross-dependencies among the components. (Soyring, 11/18/98 a.m. live testimony, pages 48 and 67, and 10/15/98 deposition testimony, pages 167 and 189.)
2. Whether the component in question exposes APIs to third-party users. (Soyring, 11/18/98 a.m. live testimony, page 48, and 10/15/98 deposition testimony, page 167.)
3. Whether the component in question is called upon by a wide variety of applications software. (Weadock, 1/8/98 deposition, pages 48-49.)
4. Whether the component performs services that are not directly available to end-users. (Weadock, 1/8/98 deposition, pages 49-53.)
5. Whether the component performs "low-level" services; i.e., interacts in a fairly detailed and specific way with specific devices. (See pages 49-53 of Weadock's 1/8/98 deposition testimony, where he mentions network card drivers and Winsock as examples of software features that lie within the boundaries of the OS because, in part, they perform low-level services. Farber (10/5/98 deposition, pages 91-92) defines a computer OS "as something that provides low level services, [but an] operating system platform or product or release, as commercially understood is a lot more.")
6. Whether the component performs housekeeping functions. (Weadock, 1/8/98 deposition, page 53.)
7. Whether the component provides security and protection for other essential operating system functions. (Farber, filed testimony, page 10.)
8. Whether the component performs a function also performed by a separate software product available on the market from another vendor. (Soyring, 10/15/98 deposition, pages 207-208; Weadock, 1/8/98 deposition, pages 46, 48, 56-58, 62 and 65.)
9. The extent to which the component is essential to basic OS functionality. (Weadock, 1/8/98 deposition, pages101-102.)

programming are unlikely to outweigh strong commercial incentives for achieving a particular design objective.

Second, design flexibility implies that the boundary between OS and applications software is fuzzy and mutable – and will in all likelihood remain so. Hence, we think that efforts to draw a bright line between operating systems and applications software on the basis of technical design criteria are misguided. Instead, we treat OS software as a commercial product, not simply a technological object, and we recognize that the functionality of commercial OS products must evolve in response to technological and market forces.

### *4.2 Functional and Whole-Cloth Integration of New Features*

To most PC users, "integration" means that the software features and hardware components work together smoothly and with modest effort. Computer buyers value the perception and experience of integrated performance – whether or not there is integration in a technical sense.[28] Hence, in marketing their software, firms refer to integration in the casual sense of integrated look and function, rather than in terms of internal design architecture.[29] PC users also value integration in the sense of easy, automated installation

---

10. The extent to which the component can be traced back to earlier versions of the OS. (Weadock, 1/8/98 deposition, page 46.)

[27] Farber discusses some principles of software engineering and design in his filed testimony.

[28] Several witnesses in *U.S. v. Microsoft* testify to this effect. See pages 35-37, 52, 54 of the 11/18/98 a.m. and page 86 of the 11/19/98 p.m. live testimony and pages 165-166 of the 10/15/98 deposition testimony offered by John Soyring (IBM), pages 59 and 62-63 of the 11/5/98 p.m. live testimony by Tevanian (Apple), and page 37 of the 10/28/98 p.m. live testimony by David Colburn, Senior Vice President of Business Affairs for America Online, Inc.

[29] See, for example, pages 37-38 of the 12/9/98 p.m. live testimony by James Gosling. Gosling is shown a Sun marketing document that reads in part, "In addition to its HotJava Browser, the web-enhanced Solaris operating environment also comes standard with Java Virtual Machine, JIT compiler and integrated Java API's. … Through the web-enhanced Solaris operating environment we offer web-based client server computing." Gosling responds as follows: "But when it talks about the Java software being built in as an integral part of the operating environment, it doesn't mean anything stronger than the Java Virtual Machine is another application that runs on top of the OS. … The CD-ROM is nothing much different than sort of the digital equivalent of a paper bag into which you throw, you know, whatever you think is appropriate."

of distinct software products, even when the products are simply bundled together on the same CD-ROM.[30] In a parallel manner, much of our analysis flows from a functionality-based perspective on integration and product design. From this perspective, it is the result that matters, not the means by which that functionality is achieved.

Nonetheless, the technical distinction between integration and bundling does bear on an economic analysis of OS design. Whole-cloth or "tight" integration, in the sense of multiple cross-dependencies among major components of the OS, involves costs and benefits. On the cost side, whole-cloth integration makes it harder for a PC user to economize on disk usage by deleting (or never acquiring) unneeded files or fragments of code. OS design can also affect memory usage on the PC, because an entire file must be loaded into memory even if only a portion of the code in the file is required for the execution of a particular task.[31] The bottom line: OS integration (in the sense of cross-dependencies) can place greater demands on the storage and memory resources needed to operate a PC. Hence, the huge size of a modern integrated operating system like Windows 98 or the MacOS imposes a cost – literal and figurative – on system resources.

But this cost is modest and falling rapidly. By 1999, disk storage costs per megabyte were less than one-half of one percent of 1990 levels.[32] And the capacity of the hard drive on a typical new PC system was 9,100 megabytes – up from 65 megabytes in

---

See page 35 of Soyring's 11/18/98 a.m. live testimony for a very similar exchange regarding an IBM marketing document on the "integration" of Internet functionality into OS/2.

[30] As a case in point, IBM decided to install multiple different software products within its OS/2 offering in response to customer complaints about difficult, time-consuming installation procedures (Soyring, deposition testimony, pages 165-166).

[31] See pages 55-57 of the 12/14/98 p.m. live testimony by Professor Edward Felten, a specialist in operating systems, Internet software and Web browsing programs.

[32] Based on advertisements in back issues of PC magazine, disk storage costs per megabyte fell from $5.92 in May 1990 (Windows 3.0 release date) to $2.95 in April 1992 (Windows 3.1 release date) to 33 cents in August 1995 (Windows 95 release date) to 3 cents in June 1998 (Windows 98 release date) to 2 cents per megabyte in May 1999.

1990.[33] Thus as of May 1999, even a 50 percent reduction in the size of Windows 98 would free only 1.2 percent of the disk space on a typical new PC system at an implied savings in hardware cost of just $2.25.[34]

As we noted earlier, random access memory installed on new computers has also increased dramatically in recent years, and the cost of RAM has declined more than proportionately. Over the course of the 1990s, the memory capacity of the typical new PC system rose from 2 megabytes to 64 megabytes, while the buyer's outlay per megabyte fell to less than 2 percent of 1990 levels.[35] What's more, effective memory usage is no longer severely constrained by the technical parameters of PC operating systems, as it was with PC-DOS.

In analyzing the efficiency of software integration it's also worth noting that, while "small" files dedicated to narrowly defined tasks may economize on memory requirements, the organization of the OS into fewer, larger files also has technical advantages.[36] First, it takes longer to load many small files into random access memory than one large file comprised of the same code. Second, the "calling file," which also resides in memory, must contain more code to load many small files than one large file. Third, breaking software programs into small files increases disk storage requirements, because each file, no matter how small, uses a minimum amount of disk space.[37]

---

[33] The 1990 figure is from PC magazine's annual review of best products, and the 1999 figure is based on advertisements in the 5/25/99 issue of PC magazine.

[34] Based on a PC system with a 9.1-gigabyte hard drive and a Windows 98 system requiring 225 megabytes of disk storage. At 2 cents per megabyte, the approximate cost of disk storage as of May 1999, the disk space freed up by a 50 percent reduction in the size of Windows 98 amounts to about $2.25.

[35] Based on advertisements in back issues of PC magazine, memory costs per megabyte fell from $99 in May 1990 to $37.25 in April 1992 to $36.13 in August 1995 to $2.95 in June 1998 to $1.55 as of May 1999.

[36] Farber discusses several criteria that govern how software files are organized into files on pages 7-8 of his filed testimony. He observes that "The most technically efficient size for a file is generally larger than a single routine and smaller than an entire application."

[37] Four kilobytes under Windows 98, 32 kilobytes under many earlier PC operating systems.

That said, there is little doubt that, on balance, whole-cloth OS design increases the demands on disk storage and, in some circumstances, on the memory resources required to operate a PC. However, these resource demands are a minor factor in light of the development of larger, cheaper storage disks and memory chips. Thus our economic analysis of OS design places little weight on considerations related to storage and memory requirements, even though they loom large in some technically oriented discussions of OS design.[38]

More important, in our view, whole-cloth integration can serve both end-users and applications developers by promulgating a common standard provided by an OS product. The shared files and other cross-dependencies make it less attractive to tinker with the OS, because such tinkering is likely to degrade overall performance. And by preserving a standardized environment, a highly integrated OS like Windows 98 assures hardware and software vendors that the full set of capabilities and APIs is available in every installation of the product. By reducing the number of configuration possibilities for the code modules, a standardized environment also lowers the cost and difficulty of testing OS performance and its interaction with complementary software and hardware products.

Thus whole-cloth integration protects a platform vendor and other software and hardware firms from poor performance caused by unpredictable differences across installations of the platform. The bottom line is lower testing and customer support costs

[38] See, for example, pages 11-12 of Professor Farber's filed testimony and page 44 of the 12/14/98 a.m. live testimony and pages 55-59 of the 12/15/98 p.m. live testimony offered by Professor Felten.

for the platform vendor, lower development and customer support costs for the suppliers of complementary products – and greater customer satisfaction.[39]

# 5. Componentized Design Architectures

Computer users value software for performance, ease of use and compatibility with other elements of a computer system. The internal design of software – *how* it accomplishes tasks or achieves ease of use and compatibility – is of little intrinsic interest to end-users. If two software products offer comparable capability, ease of use and compatibility, end-users value them equally, even if they rely on radically different internal designs.

Nevertheless, internal design can greatly influence the market response to software products and their capacity to successfully evolve. Software platforms require useful applications programming interfaces (APIs) to encourage the development of complementary applications. Better APIs lead to more and better software applications, which benefit consumers, the platform vendor and independent software developers. Indeed, good APIs are essential for the success of a software platform, even though few users of, say, Windows or the Mac OS could identify specific APIs or explain how they smooth the way for applications software.

We turn now to another aspect of internal software design, often referred to as "componentization". Like APIs, a componentized design provides indirect benefits that are not readily apparent to end-users. Componentization can facilitate product

[39] Of course, the standardized feature set promulgated by whole-cloth integration may not suit all users of OS software. Weadock, Farber and other witnesses for the government in the Microsoft antitrust action stress this point in connection with the whole-cloth integration of Internet Explorer into Windows 98.

development, design and testing and thereby reduce the cost of supplying software. It can also facilitate continual improvements in a large, complex system like one comprised of a software platform and thousands of complementary applications. But there's a downside, too: Componentization can diminish performance or add to product development costs.[40]

To develop these points, we shall describe the concept of componentization and identify its costs and benefits.[41] We also explain why, other things equal, a componentized approach to software design is more attractive and efficient for a firm that offers a broad line of software or one that sells large-scale system software. Microsoft fits this description on both counts.

## *5.1 What Is Componentization?*

Componentization refers to a modular design architecture that structures and constrains the interactions among elements of a software system. This design architecture prescribes the pathways along which components communicate, and the precise manner in which one component "requests" information or processing services from another.

A few analogies can make this abstract concept more tangible and clarify some of the tradeoffs involved in a componentized design.

- Consider two alternative "design architectures" for an integrated TV-VCR system. One is a closed system housed in a one-piece construction. This architecture makes the machine compact, easy to use and cheaper to manufacture. A second is a modular construction with separate units for the TV and the VCR. The units are only

[40] Cusumano and Yoffie (1998), pages 180-198 and 201-221, provide an insightful discussion of Netscape's struggle to achieve a componentized design architecture for its Navigator and Communicator products, the obstacles it encountered in pursuit of that goal, and the difficulties it faced because of its limited success in achieving a componentized architecture. Baldwin and Clark (2000) provide an extensive description and analysis of modular design strategies.

"integrated" in the sense that they easily connect and work together. This design makes it easy to upgrade individual pieces of the system. If the modular system is "open", it can also accommodate new components, like a DVD player, later on. Thus the modular design makes for a more flexible system, even though it may be less efficient or more costly in the short run.

- Consider the distinction between open-stack and closed-stack policies for book retrieval in a library. An open-stack policy permits patrons to take books directly from the stacks, and by whatever route seems desirable or convenient. In computer parlance, the user can "make calls" on the library services in an unconstrained manner. In contrast, a closed-stack policy processes all requests for book retrieval through a designated checkpoint. Here, there is a single pathway (or a limited number) by which users "make calls" upon library services. Like a closed-stack library, componentized software prescribes and limits the pathways by which a user (or other software) can call upon the processing services produced inside the component.
- Consider the provision of french fries as a metaphor for the provision of computing resources or processing services. Because patrons have different appetites, one size portion does not fit all, and a dispensary might be designed to distribute one fry at a time in order to meet every customer's wishes on the button. However, because most customers want many fries, this design solution involves a very large number of individual "calls" on the kitchen, slowing deliveries and reducing the dispensary's total capacity to deliver the goods. A more efficient design would dispense many fries

[41] Our understanding of componentization benefited greatly from interviews with Paul Maritz and James Allchin of Microsoft.

at a time – but not too many, because a lot would be wasted if they came in orders of 500 when the typical customer wanted just 100. Thus, designing an efficient dispensary requires forethought about the optimal batch size.

- Consider the development of a new fighter jet that incorporates several distinct, but interacting, technological advances. A fighter is a complicated technological "system" with many subsystems and interacting components. Each must properly work and interact in order for the overall system to perform at maximum capacity. An intelligent design architecture will make it possible to organize development and testing around many small teams, each of which focuses on a subsystem or component. This approach allows development to proceed along many fronts, tackling many relatively small problems simultaneously. Of course, it is essential that the components work together when reassembled into an overall system. This re-assembly may take place many times in the course of developing, testing and refining a new fighter jet or other system product. So clearly, the "componentization" of the fighter jet system cannot be carried out in a haphazard way. It requires forethought about how the components will interact once put back together.

Each of these analogies captures an important aspect of software componentization. The TV-VCR combo highlights the concept of modularity, which often involves a tradeoff between long run "dynamic" benefits and short run "static" costs. The library example illuminates the constraints on the pathways by which users retrieve information or make calls on system services. As explained below, this type of constraint also involves a tradeoff between dynamic benefits and static costs. The french fry example, while contrived, shows the importance of forethought about the manner in

which components are linked and the nature of requests for processing services. The fighter jet example highlights the virtues of a componentized approach to the development and testing of a new system-like product.

## *5.2 The Costs and Benefits of Componentization*

Designing intelligently componentized software is difficult, time consuming – and thus expensive. According to Microsoft's Paul Maritz, componentized design "requires a great deal of abstract thinking of a sort that human beings aren't naturally good at".[42] It is especially difficult to componentize an existing large-scale software product that was not originally designed that way.

An ill-conceived decomposition can generate tremendous demands on the microprocessor by increasing the number of calls between interfacing components. Even with the powerful processing capability of today's computers, it is very easy to slow execution dramatically. Thus it is not enough that software components work together; they must do so in a way that avoids excessive demands on the overall system.

Even a well-designed decomposition can degrade performance. Maritz offers an example involving the HTML "renderer," which is essential technology for web browsing.[43] The HTML renderer functions as a distinct component within Microsoft's Internet Explorer, but in Netscape Navigator the renderer is closely mingled with other functionality and code. According to Maritz, this mingling of the HTML renderer with other functionality in Navigator allows for faster processing than a componentized design. As a consequence, Microsoft had to put more effort into streamlining operations

[42] Interview with the authors. See Chapter 4 in Cusumano and Yoffie (1998) for an account of the difficulties that Netscape faced in the pursuit of a componentized design architecture for its Navigator and Communicator software.
[43] Interview with the authors.

within the components of Internet Explorer in order to achieve processing speeds in web browsing activities comparable to Navigator's.

On the other side of the ledger, a componentized design delivers many benefits. Some of these are obvious; others are subtle.

First, componentization facilitates code sharing across same-generation programs and code reuse in new products. Code sharing has several benefits:

- It reduces the need to reinvent the same wheel for each program, thereby economizing on development costs.
- Since the code has a longer useful life, developers have greater incentives to invest in optimizing a component's technical performance.
- By simplifying the interactions among blocks of software code, componentization reduces product testing and debugging costs. This benefit is in addition to cost savings afforded by the reuse of code that is already tested and debugged.
- Code sharing across products (e.g., Microsoft's Word, Excel and PowerPoint) helps to harmonize the user interface and other aspects of the user experience.[44]

Second, componentization makes it easier to integrate new functionality into existing software by restricting and simplifying interactions within the program.[45] And when the introduction of new functionality does cause problems, a componentized

[44] Cusumano and Selby (1995) sound similar notes in their discussion of Microsoft's approach to software design architecture. For example, they write on page 235 that "Sharing helps to harmonize the "look and feel" of different products; it also facilitates user tasks that require more than one application, reduces redundant writing of code, and cuts down the size of individual applications."

[45] Drawing on their interview with Microsoft developer Jon De Vaan, Cusumano and Selby (1995, page 245) provide a nice example of how software code that is not properly compartmentalized makes it difficult to add new features. In their discussion of the "revert to save" feature in Excel 5.0, they write that "The original procedure was very complicated, affecting as many as twenty different parts of the program. Since people did not commonly use the feature, developers would often forget it existed and "break" the feature when making an unrelated change. As a result, it has been historically riddled with bugs. De Vaan replaced this with a much simpler design that centralizes the function in one place in the code, so that developers working on other part of the system do not have to worry about it."

design makes it easier to identify the source of the problem and fix it. This advantage becomes more important as software products and systems become larger, because more complicated systems increase the potential for unforeseen interactions that create bugs. Hence, by simplifying and compartmentalizing design, componentization makes is easier to handle modern systems with hundreds of thousands or even millions of lines of code.

Third, componentization makes it easier to maintain the "backward compatibility" of platforms as they evolve.[46] Recall the closed-stack versus open-stack book retrieval analogy. The rearrangement of book stacks disrupts the pathways by which library users retrieve books under an open-stack policy. Hence, users must "reprogram" their "calls" on library services when the stacks are rearranged. Under a closed-stack policy, however, library patrons continue to present book-retrieval requests in the same manner as before. Likewise, computer users (or other software components) can continue to present their requests to a redesigned software component, so long as the component's interface remains unchanged.

This advantage is especially valuable in a product like Windows that serves as a platform for thousands of software applications. To attract users, after all, a new Windows release must continue to serve as a platform for the existing stock of

[46] The absence of a proper design strategy for maintaining backward compatibility can be disastrous for a software firm. Shapiro and Varian (1997, page 194) make this point with a concrete example. "In some cases, the desire to maintain compatibility with previous generations has been the undoing of market leaders. The dBase programming language was hobbled because each new version of dBase had to be able to run programs written for all earlier versions. Over time, layers of dBase programming code accumulated on top of each other. Ashton-Tate, the maker of dBase, recognized that this resulted in awkward "bloatware", which degraded the performance of dBase. Unable to improve dBase in a timely fashion, and facing competition from Borland's more elegant, object-oriented relational database program, Paradox, dBase's fortunes fell sharply."

Windows applications.[47] Componentization makes it easier to sustain the legacy even as the system expands its functionality.

Fourth, componentization facilitates a small-teams approach to software development by making it easier to break a project into discrete tasks. This benefit is obviously more valuable in the development of large-scale products like software platforms or integrated collections of software applications like the Microsoft Office Suite. In fact, Microsoft places great emphasis on a small-teams approach to the development of even the largest, most complicated software.[48]

The preceding discussion identifies some interesting tradeoffs in software design related to componentization. First, there is a tradeoff between static efficiency and dynamic efficiency – between short-run and long run advantages.[49] Intelligently componentized software is more flexible in that it eases the integration of new functionality into an evolving software system. However, a one-piece design may be less costly to achieve and deliver faster processing.

Second, there is a tradeoff between scope and specialization. When the same basic functionality is used in many related (but not identical) products and circumstances, there is a greater payoff to the careful design of a single component that delivers the functionality widely. Alternatively, when the functionality is required in essentially the

[47] Brad Silverberg, then Senior Vice President at Microsoft, emphasizes this point in an August 1993 interview with Cusumano and Selby (1995, pages 167-168): "[Windows] 3.0 was pretty big and pretty slow; 3.1 made a lot of improvements … [But] at some point you can't break compatibility, either. It's the interfaces. Some of them define the APIs through the applications. In some ways, if we could do them over again, we know how we could do it so we could write the system faster. But once you have those interfaces, you're pretty much locked. You can't just change them and break applications. A system like we have, we don't own it; the ISVs [independent software vendors] own it. We [the Windows/MS-DOS group] exist for one purpose, which is to run applications. And [if] you break an application, you don't have a reason for being any more."

[48] Cusumano and Selby (1995, especially chapter 2) develop this theme in rich detail.

[49] See Chapter 4 in Cusumano and Yoffie (1998), especially pages 194-196, for a discussion of the dynamic tradeoffs that Netscape faced in the creation of modular design architectures for its software products.

same product and circumstances repeatedly, it becomes more attractive to embed it within software dedicated to a narrower range of activities.

### *5.3 An Example*

Consider the different internal designs of Microsoft's Internet Explorer and Netscape's Navigator and Communicator software. Early on, Microsoft decided to pursue a highly componentized design strategy for its web-browsing technologies.[50] Netscape's browser software lacked the same degree of design modularity.[51]

As explained by the CEO of Intuit, the maker of Quicken financial software, a componentized browser provides the ability to show "an HTML frame within the context of the user interface of our products,"[52] permitting a seamless experience for the users of Quicken products.[53] Basically, the user remains within the Quicken environment, even when calling upon browser technology to retrieve information from the web. Early browsing software did not have this capability.

Intuit faced this issue squarely in 1996 and early 1997.[54] Netscape Navigator was not componentized as of early 1997, which led to discussions between Intuit and Netscape about the development of a componentized version of the then-leading browser. The componentized nature of Microsoft's Internet Explorer technologies in Windows 95 became an important factor in Intuit's decision to switch from Navigator to IE as its

[50] See the Declaration of David Cole (November 8, 1997) in *U.S. v. Microsoft* for an informative description of Internet Explorer's componentized design architecture.
[51] See pages 183-185 in Cusumano and Yoffie (1998) on the contrast between the highly componentized design of Microsoft's Internet Explorer software and the design of Netscape's Navigator and Communicator software.
[52] Harris, 9/29/98 deposition testimony, pages 34-35.
[53] Harris, 1/4/99 p.m. live testimony, page 45.
[54] Harris, 1/4/99 p.m. live testimony, pages 9-12, and 9/29/98 deposition testimony, pages 34-36.

primary browser for Quicken products.[55] Intuit subsequently distributed some five million copies of IE with 1997 versions of Quicken, Turbotax and Quickbooks.[56]

Also in this period, America Online (AOL) sought to provide a "seamless consumer interaction … when [going] from one environment to another", such as from AOL's proprietary network to the web.[57] Netscape's browser technology required a visible leap from one environment to another, whereas the componentized nature of IE (which AOL adopted) allowed for a seamless experience.[58]

It's clear, then, that the componentized design of IE made it more attractive to some providers of complementary software applications and Internet services – and, in the process, enhanced the value of Windows as a software platform. The examples also illustrate how design flexibility allowed for the same basic functionality – browsing technology – to be implemented in different ways that, in turn, had an important influence on market outcomes.

## *5.4 Componentized Software: Why, Especially, at Microsoft?*

Microsoft platform products like Windows and Windows NT and business applications like the Office Suite are among the largest software systems offered by any mass-market software vendor. Because a componentized design architecture facilitates a small-team approach, it is especially valuable in the development and improvement of these large software systems. Componentization also helps maintain backward

[55] Harris, 1/4/99 p.m. live testimony, pages 9-10.
[56] Harris, 1/4/99 p.m. live testimony, page 75.
[57] Colburn, 10/28/98 p.m. live testimony, page 37, and 10/29/98 a.m. live testimony, page 34.
[58] Colburn, 12/9/98 p.m. live testimony, pages 37-40. Colburn speaks about an "integrated browser" as providing the seamless user experience that AOL sought, but the context of his remarks make clear that he is using this terminology to refer to the componentization of browser technology.

compatibility in the evolution of platform products.[59] Microsoft owns the most successful commercial software platforms. It also has a larger, more diverse set of independent software vendors writing to its platform than any other software firm.

In addition, Microsoft has the broadest line of software and the largest revenues from software of any firm in the world. The company spends more on software development than any other firm. Hence, it has the most to gain from code reuse, from optimizing the inner workings of a software component and from harmonizing features and performance across software products. The emphasis on componentization at Microsoft has intensified over the past decade in line with the increase in its software development efforts, the expanding breadth of its software product line and the increasing scale and complexity of products like Windows and Office.[60]

This interpretation of Microsoft's approach to software development and design also fits with the observation that Microsoft has become a leading developer of object-oriented software design tools. These tools facilitate the use of a componentized design strategy.

By the early 1990s, and perhaps earlier, Microsoft's approach to software development emphasized several of the virtues associated with componentized design architecture. In an August 1993 interview, Microsoft CEO Bill Gates stated that two key principles for managing software development are “A development process that allows

[59] The costs of maintaining backward compatibility come through loud and clear in some remarks by Lou Perazzoli, Software Engineering Manager for Windows NT in 1993. In the course of discussing Microsoft efforts to incorporate Windows 3.1 features into Windows NT 3.0, Perazzoli states that in a typical week, about 1,000 new bugs get opened. "And the question is, how the hell can anybody developing software have so many bugs? It turns out that it is called “compatibility”. If we didn't have to be compatible with Windows [3.1], we wouldn't have so many bugs." [As quoted on page 319 of Cusumano and Selby (1995).]

[60] Microsoft's moves toward greater emphasis on modularity and componentization over time is a clear theme in the discussions of Cusumano and Selby (1995), although they do not use the term

large teams to work like small teams" and "Product architectures that reduce interdependencies among teams."[61]

Since the late 1980s, Microsoft has made a conscious effort to harmonize the user interface, feature set and performance characteristics across its major software products. Initially, this effort focused on external user-oriented aspects of software such as the user interface and the content of pull-down menus. With time, though, code sharing and functional integration received greater emphasis. This process is especially evident in the MS Office Suite. Originally, Word, Excel and PowerPoint were independent products with little or no shared code and limited integration. As discussed at length in Cusumano and Selby (1995), much had changed by the mid-1990s. The Office products are now closely integrated and share much of their code. What's more, their ongoing development is now closely coordinated. Seen in this light, the move toward componentized design at Microsoft is one step in a long-term effort toward greater harmonization of features and greater functional integration across software.

# 6. Efficient Management of Interacting Features and Components

## *6.1 Simplifying the End-User Experience*

To perform properly, technologically advanced products often require complex coordination of many interacting components. Achieving the necessary degree of coordination presents challenges in design, installation and operation. Most consumers,

---

"componentization". Paul Maritz and James Allchin confirmed this move in the course of our interviews with them.

[61] Cusumano and Selby (1995), page 25 and, also, page 237.

however, place a high premium on ease of use, even for products that depend on arcane technology. They want no-fuss installation, push-button performance and automated management of interacting components. When problems arise, they expect quick solutions.

It should be no surprise, then, that, many product improvements are responses to customer demand for ease of use. Examples include cable-ready televisions, integrated home stereo systems, factory-installed automobile air conditioners, fax-copier machines, internal PC modems and PC distributors who pre-install the OS and other software. As these examples suggest, greater ease of use often involves the integration of distinct components that could be sold separately.

Consider the example of automobile air conditioners. When they first became available, air conditioners were installed by the auto dealer or by a specialized firm hired by the dealer. Problems were common and often involved the interaction between the air conditioner and components of the automobile engine. For example, cooling systems designed for cars without air conditioning often boiled over when subjected to the added demands of an air conditioner.

Other potential interaction problems were more subtle or specific to certain engine designs. A/C compressors have very high peak power requirements, which can stall the engine. A slippable connection between the engine and the A/C compressor such as a belt-drive can solve the problem. But with a rigid coupling – such as a gear system – serious problems arise if the A/C compressor clicks on when the engine is already under high load. By the same token, diesel engines have a different power curve from gasoline engines and different acceleration characteristics. So it may be necessary

to shut the A/C compressor off in diesel-powered vehicles during warm-up and acceleration. Gasoline engines, by contrast, rarely require this type of regulation of the air conditioning system.

Because of such complications, the availability of "factory air" became a big selling point for automobile manufacturers. While "factory air" was often still installed by the dealer, the system was designed to the automaker's specifications and carried the automaker's warranty. Automobile cooling systems were beefed up to account for the added load, and boil over problems – common in the 1960s – became a thing of the past. Today, car buyers take all this for granted. They buy a car, push the "A/C ON" button, and it works.

The preceding remarks can be distilled into three assumptions that inform a theory of product integration:

(i) Customers and dealers have a limited capacity or desire to manage the complexities generated by interacting components.

(ii) Greater ease of operating the product often involves the integration of distinct components that can be sold separately.

(iii) Integration requires up-front design costs beyond initial product development costs.

Two other factors influence the timing of integration:

(iv) When a new product or feature is introduced, performance, cost and demand are initially uncertain. The uncertainty may involve the ultimate level of consumer acceptance, which version of the product will ultimately achieve the greatest commercial success and technological performance, and the size of up-front design costs required for seamless integration.

(v) The demand for a successful new product grows over time as information about its availability, characteristics and performance diffuses among the base of potential users.

Assumption (i) implies a latent demand for integration, but not every new feature or product will eventually be integrated into a larger system. Other things equal, integration is more attractive to the manufacturer of the system product – and likely to generate greater social gains – when more buyers use the stand-alone feature or product,[62] when the design and production costs of integration are lower, and when the stand-alone feature interacts with other features of the system in a complicated way.

Uncertainty about the technical performance of a new product and its interaction with system components creates an incentive to delay integration. As technical problems are identified and resolved this concern diminishes, so that integration becomes more attractive. Both from the perspective of economic efficiency and the system manufacturer's profitability, technological uncertainty adds value to the option of waiting to integrate. This "option value" arises because a manufacturer incurs irreversible costs

[62] In some cases, the stand-alone product may be unpopular because it is difficult to use (e.g., TCP/IP), but a smart developer can foresee that it would be popular if integrated.

when it commits to one approach to design and integration. The costs may involve direct expenditures, or they may take the form of harm to the system manufacturer's reputation for quality and reliability if the newly integrated feature performs poorly or interferes with other system components. By retaining the option to select its approach to integration, the manufacturer leaves room to act on information not yet available.

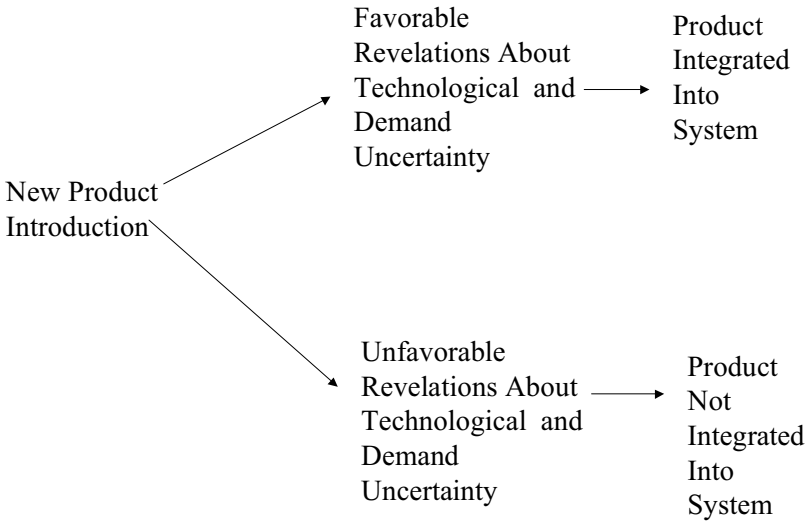
In short, the option value of waiting induced by technological uncertainty reflects two issues: (a) whether integration involves acceptable performance gains relative to the cost of achieving integration and, (b) when more than one technical option is available, how best to integrate the new product. Of course, this option value is not the only consideration that governs the timing of integration. If the benefits of integration are large or the up-front costs of integration are small, early integration becomes more attractive.

Uncertainty about demand for the new product also induces an option value of waiting to integrate. This option value of waiting reflects two issues: (a) whether demand is sufficiently high to justify the fixed integration costs, and (b) which version of the new product or feature to integrate. Of course, as demand-side sources of uncertainty diminish, integration becomes more attractive.

Even when eventual integration is assured, delay may be rational. Both technical uncertainty about how to integrate and demand uncertainty about which version to integrate encourage delay. Even when there is no technical or demand uncertainty, a pure "discounting" effect – the time-value of money invested – encourages delay if the up-front costs are high relative to the initial rewards from integration.

Hence there are strong economic incentives to postpone integration until (a) kinks and performance problems in the new product are identified and resolved, and (b) demand for the new product is revealed to be sufficiently high to justify the costs. Large benefits and small sunk costs encourage earlier integration.

This type of theory delivers the following stylized introduction-innovation path for new products:




The theory also points to a potentially important complementarity between the integration of existing features and products into a system and the development of new stand-alone products that interact with the system. The integration of existing products into a computer OS, for example, economizes on the limited capacity of users and dealers to manage interacting components. Integration thereby opens the door to the development and introduction of additional non-integrated products. For example, the integration of better support for video display hardware and software into the OS

simplifies the use of video-intensive applications and encourages the customer to make greater use of non-integrated products like scanners and digital cameras.[63]

In the extreme scenario in which customers and dealers have a fixed tolerance for managing interacting components, halting integration eventually stifles the development and introduction of new products. In this regard, it is striking that despite the automation of everything from the clutch to the choke to radio tuning, the modern automobile has about as many user-operated controls as the earliest autos. Drivers, it seems, will put up with just so much distraction when they are in motion. Moreover, even if there is no absolute upper bound on the capacity of consumers and dealers to manage interacting components, assumption (i) implies that integration of existing products into a system increases the demand for new non-integrated products that make use of the system.

### *6.2 Facilitating Applications Development*

We can summarize the foregoing theory: Product integration simplifies end-user experience by helping to manage the interacting components in a system product. We now sketch a parallel and complementary theory that focuses on how integration facilitates the development of new applications for the system product. In the case of computers, the integration of key software building blocks (APIs) into the PC operating system promotes innovation and product variety by reducing the cost of innovation.[64]

The key assumption in this theory of how OS integration facilitates applications development is analogous to assumption (i) above: Software applications developers have a limited capacity or desire to manage the complexities generated by interacting software

[63] See Boyce (1998) for an evaluation of the integration of new features into Windows 98.
[64] This sort of integration also facilitates the on-line distribution of software products, a point we take up below.

and hardware components. Other assumptions in this theory parallel assumptions (ii)-(v) above.

Most developers of software applications focus on "high-level" design and functionality while leaving critical but routine tasks such as file management, memory management, graphical displays, and video and audio management to the operating system. This division of labor relieves developers of the need to re-invent the wheel in each application and allows them to focus on their areas of expertise and commercial interest.

Specialization at this broad level is reflected in the often-drawn distinction between "systems programmers" and "applications programmers." The distinction is usefully applied to firms as well as individuals. Norton, for example, is a successful, well-respected software firm that mostly employs system programmers and specializes in operating system functions. Corel, another respected software firm, mostly employs applications programmers and focuses almost entirely on applications-oriented software.

Many software developers are highly specialized in a particular application – e.g., financial analysis, architecture, electric power distribution or laboratory automation – and they focus on specialized aspects of their field. Their knowledge, when embodied in software applications, is often the main source of added value and commercial viability for their products. For example, the local power company buys a computer-aided design package from AutoDesk primarily because of the expert knowledge embodied in the software, not because the CAD package offers better standard features (e.g., toolbars) than other software products.

Likewise, the incorporation of TCP/IP and other popular networking protocols into Windows provides independent software vendors a larger standard set of system services to leverage into specialized applications. Because the Hypertext Markup Language (HTML) has become a standard for web applications, its use by the Windows Help facility in place of a proprietary language makes it easy for developers to code sound and video into Help files and to provide Help links to the developer's web site.

This efficient division of labor among software developers implies that the integration of richer support tools into the OS leads to better, less costly software applications. Software developers know they can design applications with Internet Explorer functionality in mind because the IE technologies are integrated into the Windows OS. In addition, as explained in Section 5, the "componentized" nature of IE technologies affords greater flexibility in program design.

Thus, an applications program can call on the HTML display facility of the browser interface – one component of the IE technologies – whether or not the computer user is on the Internet. Alternatively, by calling on other components of the IE technologies, an applications program can execute Internet transactions without displaying web pages or making any activity visible to the user. Intuit's Quicken is now designed so that a Windows user can access the latest interest rates and stock prices from the Web without leaving the Quicken application. This unobtrusive use of components of the IE technologies is more convenient than retrieving the same information by initiating a browser shell program with a separate user interface.

For another indication of how the integration of software into the PC operating system facilitates specialization and product variety, consider some responses by

software developers to improvements in IE technology that were integrated into Windows 98. According to *Business Week* "at least four new browsers" that ride on top of IE were introduced shortly after the release of Windows 98.[65] Because Microsoft has built the complex software into Windows needed to run Java applets, display graphics, play audio and video and perform other tasks expected of browsers, even the simplest browser built around IE technology can do these things.

In this way, OS integration gives product developers stronger incentives to create specialized browsers. IBM's Lotus Development, for example, has developed a browser built on IE technology that "works well and has a distinctive Notes look". Similarly, MediaLive has developed Surf Monkey, a "kid-safe" browser using IE technology.[66]

Note, too, that the integration of software building blocks into the PC operating system makes on-line distribution of new software faster. *Business Week* offers the example of the stand-alone Netscape Communications' Navigator browser, whose 8 MB of code take at least 45 minutes to download on a 33.6 kilobit-per-second connection. By contrast, Bigfoot's NeoPlanet Browser, which makes use of IE components that are already part of Windows, occupies just 791 KB and can be downloaded in five minutes. Consequently, the wide dissemination of IE technology through OS integration increases competition and innovation in browser shell programs.

Basically, OS integration reduces the cost of innovation in the applications market. This positive effect on the supply side of innovation adds to the demand-side

[65] Wildstrom (1998).

effects discussed above, whereby OS integration opens the door to new PC applications by simplifying the use of interacting components in a computer system.

OS integration is also a useful way to standardize the computing and software development environment. Furthermore, by facilitating on-line distribution, it reduces the cost of distributing applications products. On-line distribution is especially important for specialized software products that are unlikely to be carried by traditional retail outlets.

Regarding which software building blocks become integrated into the OS and when, it should be clear that an analysis parallel to the one developed in the preceding section applies here as well. For example, once a platform supplier publishes a set of APIs that developers come to rely on, it becomes costly to remove the APIs or to alter them in ways that damage backward compatibility. Hence, a platform vendor has strong incentives to postpone the introduction of new APIs until he is confident about both their usefulness and his ability to offer continued support for the APIs as the platform evolves.

## *6.3 Reducing Customer Support Costs*

Another motive for integrating stand-alone elements into the OS is the desire to hold down customer support costs. For many software firms, customer support is a major cost of business and, in a world of dirt-cheap CD-ROMs, the main incremental cost of selling software.

---

[66] The creation of custom browsers is explicitly taught in the Microsoft documentation on IE4. Chapter 11, pages 135-145, in Microsoft (1998) describes the WebBrowser Control, from which custom browsers can easily be built.

This point comes through loud and clear in Microsoft's experience with customer support.[67] As of the early 1990s, Microsoft was fielding about 60,000 customer support inquiries per day, including 20,000 phone calls that had to be handled by product support engineers. Microsoft personnel in the customer support division actually outnumbered the firm's software developers.[68]

Microsoft answered one customer support call for every three software units sold, at an average cost of twelve dollars each. At that rate, just a few calls from a customer can wipe out the profit from selling most software products, even before factoring in development costs. As of 1993, in fact, customer support costs equaled 20 percent of the gross revenues from Windows and 25 percent from Windows NT. Most calls came from customers who had purchased the software within the previous 90 days, and about half of these "relate to setting up or installing the software…printing, usage of new or changed features, the operating environment, and interoperability with other products."[69]

This stark reality encouraged Microsoft to focus on design advances that would reduce customer support calls and costs. These innovations drew heavily on usability studies for new software products along with detailed statistical summaries of the problems that prompted support inquiries.[70]

The careful integration of software functionality into the OS can dramatically reduce the costs of customer support. Consider, for example, what it took for a PC user

[67] This paragraph draws on pages 224 and 362-370 of Cusumano and Selby (1995).
[68] According to remarks by Mike Maples, former Microsoft Vice President, as quoted in Cusumano and Selby (1995) on page 367.
[69] This quotation is from page 365 in Cusumano and Selby (1995), who cite a document titled "Case Study: The New Microsoft Support Network."
[70] Cusumano and Selby (1995), pages 375-384. Our interviews with Hillel Cooperman and other Microsoft personnel indicate that intensive usability studies have become an even more important input to software design decisions at Microsoft in recent years.

to get on-line in, say, 1993. The user had to acquire and install an operating system, a modem, a browser or other Internet software, an Internet provider, an Internet account and a TCP/IP "stack." He also had to enter the appropriate parameters and data for each piece of software and hardware. Getting on-line required that all the information be entered correctly and that all the pieces worked together properly. As many who undertook this task can ruefully recall, getting on-line also required innumerable calls to customer support centers. By contrast, the integration of these elements into modern operating systems like Windows and MacOS 9 allows today's PC user to stroke a few keys to get on-line the first time. At Microsoft, and no doubt other firms, this type of OS integration has dramatically reduced the resources devoted to helping customers connect to the Web and other networks.[71]

The integration of the HTML-based Help system into Windows 98 was also motivated, in part, by the desire to reduce customer support costs. Frequent web references point users to documents that provide background, examples, technical information and links to other web-based information sources. This system can be used at all levels of the software development hierarchy – OS tools, applications and company-based Intranet software.

Not all stand-alone elements will eventually be integrated into OS products. As discussed earlier in the chapter, integration becomes more attractive as the use of the stand-alone element increases, when integration leads to lower design and production costs, and when the stand-alone element interacts with the other elements of the OS in a complicated way. This section simply adds another potential benefit that must be

[71] Personal communication with James Allchin.

weighed against the costs. In parallel with our earlier discussion, the timing of integration is influenced by uncertainty regarding the magnitude of customer support costs and how integration might most effectively reduce such costs.

The demand for OS integration can also arise from customer preferences for dealing with a single vendor – especially in cases where there are many interacting components in a complex system. Computer software systems can be extremely complicated, sometimes involving several sub-systems with a million lines of code each.[72] When problems arise, customers want to know where to turn for solutions. And when a single seller supplies all the components, the answer is clear.

Early users of microcomputers had to assemble the systems, piece by piece. No one, except for an occasional "techie", attempted the task without assistance from customer support centers. If something went wrong with the computer after assembly, the user had to determine which parts vendor to contact for advice before he could even hope to return the computer to working order. Altair was the first to build a microcomputer system that included a power supply, motherboard, CPU, memory, I/O and OS (BASIC interpreter). The present-day predominance of this form of integration suggests, among other things, that customers highly value the opportunity to buy all the pieces assembled and supported by a single vendor.

Or consider a more personal example. One of us (MacCrisken) worked as the Data Base Administrator for Intel in the early 1980s. During that period, Intel experienced a variety of network performance problems, which often involved the

[72] The complexity of microcomputer OS software has increased dramatically over time. MS-DOS 1.0 was designed for the original IBM PC in 1981 and had about 4,000 lines of code, MS-DOS 3.0 was designed

interaction of the mainframe OS, the communications software and the network hardware. Fortunately for Intel, all of these products came from IBM. So when problems arose, Intel called on IBM – and the buck stopped there.

However, if a customer buys routers from Cisco, computers from Dell, modems from 3Com and the components don't interact properly, what options are open to the customer? Too often the choice is between bouncing from finger-pointing vendor to figure-pointer vendor, or hiring a third party troubleshooter to solve the problem. [73]

Sometimes, the customer's demand for a single vendor can be met by the integration of additional features into a computer's operating system. This permits the seller to test the integration exhaustively before putting the components on the market, *and* it assigns clear responsibility for the failure of interacting components to work together. Less-informed and less-sophisticated computer users are especially likely to value the acquisition of software products from a single firm with a well-established reputation for quality and customer support.

## 7. Demand-Based Motives for Software Bundling

Software is frequently bundled in the sense that multiple features or applications are packaged together or distributed jointly. Often, one or more items in the bundle are (or were) previously sold as separate products.

---

for the IBM PC/AT in 1984 and had about 40,000 lines of code (Ichibiah and Knepper, 1991, pp. 252-253), and Windows 95 has about 11 million lines of code (Reid 1997, p. 149).

[73] See, e.g. McCartney (1986). In discussing the pros and cons of using more than one vendor for a computer system, the writer reports that "when a problem develops on a multi-vendor network, suppliers tend to pass the buck." And, "no one takes responsibility."

A few examples highlight the ubiquity of software bundling. PCs often come bundled with a large collection of software applications at no extra charge. Adobe, Corel, Lotus, Microsoft, Norton and other software firms often bundle stand-alone applications together in "suites" or multi-feature packages. PCTools, for example, offered a large collection of distinct software features on a single CD-ROM. And software vendors routinely bundle large collections of utilities and features with OS and platform products. In 1993, for example, Microsoft Windows for the first time included disk compression features similar to Stac's Stacker and fax functions similar to Delrina's WinFax product (Markoff, 1996).

As some of these examples suggest, the joint distribution of software features and applications often takes place even when there are no technical benefits or cost savings from code-level integration. In line with this observation, we show that demand-based effects alone can provide powerful motives for software bundling. We also explain why software bundling motivated by demand considerations leads to economic efficiency.

Before developing the economic logic of bundling, some preliminary remarks help place this section's analysis in proper perspective.[74] First, demand-based motives for software bundling are distinct from, but fully consistent with, the motivations for OS integration discussed in earlier sections. Second, the benefits of bundling can be achieved by any method of joint distribution, including integration. As a practical matter, a mix of technological, timesaving and demand-based factors may motivate the integration of particular software functionality. But for analytic clarity, we emphasize purely demand-based motives in this section. Third, demand considerations can lead a profit-maximizing

[74] See Shapiro and Varian (1999, pages 73-79) for a broad, introductory perspective on bundling.

firm to offer certain software applications at a zero or negative price. When this occurs, even a tiny technological benefit or cost saving makes integration the preferred strategy for achieving joint distribution.

## *7.1 Complementary Demand with the Operating System*

The zero-price bundling and integration of certain software features with Microsoft's Windows OS has been criticized as predatory and anti-competitive.[75] It is important to recognize, however, that the most basic theory of pricing by a multi-product firm with downward sloping demand points to a very different explanation for this behavior – an explanation which implies that zero-price bundling of software applications with OS products is a socially beneficial form of competition.

Start with the standard theory of pricing for a firm that sells multiple products with complementary demands. Two products are said to be complements when greater sales of one stimulate demand for the other. As an example familiar to many parents, sales of Barbie dolls stimulate the demand for Barbie clothes. After reviewing the pricing implications of complementary demand, we apply the logic to the bundling of OS and applications software. We then consider some implications of complementary demand for market structure and consumer welfare.

Consider a multi-product firm with downward sloping demand for each product. To focus on demand-based explanations for bundling, assume that production costs are unrelated across products. Also, set aside the motivations for product integration treated in earlier sections. To further simplify the exposition, assume that all of the firm's

[75] For example, see the Complaint filed by the U.S. Department of Justice on May 18, 1998 in *U.S. v. Microsoft* and the Plaintiffs' Revised Proposed Findings of Facts filed on August 10, 1999.

products are complements, so that greater sales of any one product increase demand for the others.

Under these conditions, complementary demand encourages the multi-product firm to set lower prices than would a collection of independent firms, each selling a single product. The logic is straightforward: a lower price on any one product generates additional sales of that product and all products with complementary demands. A multi-product firm internalizes this demand spillover onto the complementary products, while independent single-product firms do not.[76] In fact, complementary demand can lead the firm to price some products below marginal cost.

This point can be demonstrated by means of a simple model. Assume that the firm sells two complementary products with linear demand curves,

$$q_1 = a_1 - b_1 p_1 - d p_2$$

$$q_2 = a_2 - b_2 p_2 - d p_1,$$

where $q$ and $p$ denote quantities and prices, and the $a$, $b$ and $d$ parameters describe demand. Positive values for $a$ and $b$ imply positive, downward sloping demand curves, and a positive value for $d$ corresponds to the case of complementary demands. Assume that $d$ is less than $b_1$ and $b_2$, so that own-price effects dominate. In addition, assume that the firm produces and sells each product at constant marginal costs denoted by $c_1$ and $c_2$.

Consider a numerical example in which the first product has bigger demand. The following parameter values fit this situation: $a_1 = 100$, $a_2 = 50$, $b_1 = b_2 = 1$, $c_1 = c_2 = 10$,

[76] Tirole (1988) treats the pricing behavior of multiproduct firms with interdependent demands at length. He also provides extensive references to the relevant literature. Portions of his treatment in chapters 1 and 3 are especially pertinent to the discussion at hand.

and $d = .5$. With these values, profit maximization yields the prices, $p_1$=55 and $p_2$=5.[77] The firm prices good 2 below marginal cost in order to generate more profits by stimulating additional sales of good 1.

Now apply this logic to Microsoft's bundling of OS and applications software. The complementarity requirement certainly holds in this case, because Microsoft's PC software applications typically run on computer platforms that make use of its Windows OS products. So sales of PC applications software stimulate demand for Windows, and vice-versa. To understand why complementary demand easily leads to zero-price bundling of applications software, it is important to recognize that the marginal cost of software sales may be quite low.[78] If we modify the previous example so that marginal cost is zero for each product, the profit-maximizing prices become $p_1$=33.3 and $p_2$=0.

Zero marginal costs are not necessary for a zero price outcome, although low marginal costs make such an outcome more likely. If we modify the numerical example so that $a_2$=20 and retain the assumption that marginal cost equals 10 for each good, the firm's profit-maximizing prices become $p_1$=65 and $p_2$=-15. The relatively low demand for good 2, coupled with complementary demand and low (but positive) marginal costs, leads the firm to set the price of good 2 below zero in order to stimulate sales of good 1.

In practice, a negative price may or may not exploit complementary demand any more effectively than a zero price. If a negative price can be conditioned on actual use, then the firm can earn more profits in the preceding example by paying others to use

[77] The mathematical supplement to Davis and Murphy (2001) derives expressions for the profit-maximizing prices.

[78] The marginal costs of software production (i.e., replication) and distribution are often quite low, but customer support costs are high for many software products. So the full marginal cost is low for some, but certainly not all, software products.

product 2. Alternatively, if paying someone to take possession of a product provides no guarantee or incentive that he will actually use it, there is no point in offering the product at a negative price. Instead, by distributing the product at no charge, the firm maximizes product usage (and any effects of complementary demand) without incurring the additional expense of paying customers to take possession. Taking this observation into account and setting $p_2$=0 in the example with $a_2$=20 leads to a profit-maximizing price for the first good of $p_1$=57.5.

Adobe has pursued this pricing strategy for complementary software products with great success. As Shapiro and Varian (1999, page 254) observe, Adobe allowed its portable document format (PDF) to "become an open standard but cleverly exploited the complementarities between creating and viewing a document. Adobe charged for the PDF creation software, while giving away the viewing software." Adobe successfully pursued a similar strategy with its Postscript page-description language and related software products.

Although extremely simple, the theoretical examples capture three salient aspects of the pricing and bundling of OS and applications software. First, OS and applications software are complementary in use. Second, marginal costs (of production, distribution and customer support) are quite low for many types of software. Third, the demand for OS software is typically greater than the demand for a particular applications product. The third point, in particular, indicates that OS products play the role of good 1 in the numerical examples, and applications play the role of good 2.

We have now developed a simple explanation for the zero-price bundling of software features or applications with the OS that does not involve any dynamic or strategic considerations. Nor does it involve any direct technological or timesaving benefits of integration or reductions in customer support costs of the sort highlighted in earlier sections. Instead, the key elements in this explanation are low marginal costs for the bundled feature and complementary demand with the OS. Other things equal, zero-price bundling is more likely for complementary goods that have relatively low demand.[79]

Two additional cost factors reinforce this complementarity motive for zero-cost bundling of software applications and features with the OS. First, this method of bundling is more convenient for the consumer than any other distribution method because it eliminates the time and effort associated with acquiring and installing the zero-price item.[80] Second, this form of bundling is also less costly for the software firm than other forms of distribution. There are no separate distribution costs and no customer support costs related to installation.

Other aspects of Microsoft's pricing behavior lend support to this interpretation of zero-price bundling with the OS. In particular, Microsoft has been a price-cutter in many software application categories such as CD encyclopedias, web browsers, personal financial planning and core business applications.[81] This behavior is a natural consequence of Microsoft's broad software product line, given that distinct categories of

[79] In fact, with zero marginal costs, $(a_1 / a_2) > (b_1 / d)$ is a necessary and sufficient condition for $p_2 < 0$ in the two-good model.

[80] The relevance of this point is borne out in testimony by IBM's Director of Network Computing in *U.S. v. Microsoft.* See the deposition testimony of John Soyring, pages 165-166.

software applications are complements in use and, especially, the strong complementarity between Microsoft's OS products and its software applications that run on the OS. Complementary demand across product lines gives Microsoft a stronger incentive than its competitors to set low prices, even when it has the same development and production costs and the same degree of market power for particular software products.[82]

## 7.2 Implications for Market Structure and Consumer Welfare

Whether it involves zero-price bundling of features with the OS or lower prices on stand-alone products, this type of behavior improves economic efficiency and helps consumers. The incentive for a multi-product firm to set lower prices in the face of complementary demand leads to higher output, more consumer benefits and greater economic efficiency.[83]

Indeed, in the simple two-good model, consumers are unambiguously better off when a single, integrated firm sells both products than when a different firm sells each product. What's more, total profits are higher with a single, integrated firm. Since profits are higher and consumers are better off when a single firm sells both goods, economic efficiency must also be greater.[84]

Complementary demand ($d>0$) is the critical assumption that underlies the favorable effects of the single-firm market structure on consumer welfare and economic efficiency. While our mathematical model is highly stylized, the economic logic of

---

[81] See Liebowitz and Margolis (1999), especially pages 154-157, for some systematic evidence. Khanna and Yoffie (1996) discuss how deep discounting on Microsoft's Office Suite helped bring about a steep decline in the price of business applications software during the 1990s.

[82] We develop this point more fully in Section IX of Davis, MacCrisken and Murphy (1999). Shapiro and Varian (1999) also emphasize this point. See their chapter 6, especially page 162.

[83] This point regarding complementary demand is closely related to standard arguments about the social benefits of vertical integration when both upstream and downstream firms exercise market power.

complementary demand is general, and the model carries important implications for thinking about market structure, efficiency and consumer welfare in software markets.

To appreciate some of these implications, consider the evolution of the software market during the 1990s. Many commentators remark with a tone of concern, even alarm, that Microsoft has become the leading vendor for many business and consumer software products during the 1990s.[85] One source of this concern is a perception that Microsoft enjoys an unfair competitive advantage over other software firms because of the tremendous success of the Windows platform.

Without pretending to address this issue in full, we can garner some important insights by applying our analysis to the question of market structure and consumer welfare. To that end, suppose that two complementary products – let us call them OS and WB – have been developed by separate firms. The demand structure and marginal costs of production are the same as above. Initially, the two firms independently price and sell their products. Recognizing that the action of each affects the other, the two firms behave strategically in the manner of Bertrand (prices as strategies) or Cournot (quantities as strategies).[86]

We have already established that total profits are higher when a single, integrated firm sells both products. Hence, the two firms have a strong incentive to merge. Under the assumptions set forth here, a merger would be profitable for the firms *and* beneficial

[84] The mathematical supplement in Davis and Murphy (2001) contains a precise statement of these claims and a proof.
[85] See, for example, the remarks on pages 37-38 by Katz and Shapiro (1999), two prominent and highly respected economists with extensive backgrounds in industrial organization and antitrust matters. Their tone is one of concern, not alarm.
[86] The mathematical supplement in Davis and Murphy (2001) spells this out explicitly and derives explicit solutions for prices and quantities in the Bertrand and Cournot cases.

for consumers. Nonetheless, suppose that the two firms do not merge, because other aspects of their businesses do not mesh well, or perhaps because of opposition by the antitrust authorities.[87]

In the absence of a merger or acquisition, each firm might subsidize the price charged by the other. In this manner, the two firms could try to internalize the demand complementarity without actually merging. Firms sometimes do enter into mutual promotion agreements that contain cross subsidies of this sort. Of course, if this option were a perfect substitute for merging, there would be no incentive to merge. In practice, though, inter-firm subsidies often run into serious practical problems. If the OS firm subsidizes the sale of WB, the WB firm may respond by expanding into market segments that are not especially helpful to the OS firm. Practical problems with this type of subsidy arrangement are apt to be especially severe when the net-of-subsidy price for WB is zero or negative, as in some of our numerical examples. An even more basic problem with the cross-subsidy solution is that it does not confer common ownership and control. The two firms may have imperfectly aligned incentives over how to market or design their respective products. Thus, even if a successful cross-subsidy arrangement is feasible, there will remain incentives to bring the two complementary products under common ownership and control.

[87] Opposition by the antitrust authorities in the presence of important demand complementarities is quite plausible. In 1994, the U.S. Department of Justice denied a proposed acquisition of Intuit by Microsoft. Intuit sells the popular Quicken line of software, which was and is the leading personal finance software for PCs. Intuit's line of software products is highly complementary to Microsoft's operating system software. Nonetheless, the DOJ denied the proposed acquisition, because Microsoft Money was the leading competitor to Quicken. See Katz and Shapiro (1999). To take another example, it is highly unlikely that the DOJ would have acquiesced to a Microsoft acquisition of Netscape in 1994 when Microsoft had no browser, or in 1995 when Microsoft's Internet Explorer was clearly inferior to Netscape's Navigator.

This brings us squarely to the issue of market entry. We tailor our assumptions so that the analysis speaks to government claims in *U.S. v. Microsoft.* Let us rule out cross-firm subsidies and consider four specific assumptions. First, the two goods are initially owned and sold by separate firms. Second, it is costly for the OS firm to develop its own version of WB but much more costly for the WB firm to develop its own version of OS. Third, the demand structure and marginal production costs are such that it is profit maximizing for an integrated firm to sell WB at a zero or negative price. Fourth, demand is great enough so that entry into the WB market is profitable for the OS firm. These four assumptions reflect key aspects of the market situation circa 1994 as it pertains to Microsoft and its Windows OS, on one hand, and Netscape and its Navigator web browser, on the other hand.

Given these assumptions, the OS firm finds it profitable to develop its own version of WB, although the development costs are quite high. After incurring these large development costs, it then proceeds to give away its version of WB at no charge. It may even pay others to distribute WB or pay consumers to use it. This course of action obviously harms the firm that originally developed the WB product. Indeed, in our stylized model, the OS firm drives the original WB firm out of the market.

On first encounter, the decision by the OS firm to spend large sums to develop a "zero-revenue" product might appear anti-competitive or predatory. The government has drawn just such inferences in *U.S. v. Microsoft* from facts that parallel the assumptions in our stylized analysis. Indeed, the government's chief economic expert in the case draws the conclusion that "Microsoft had monopoly power, and its bundling and related actions 'made no business sense' save for the protection of that power." (Fisher, 2000, page 183)

But it is clear from our analysis that even the most basic two-good model of complementary demand delivers a simple, pro-consumer interpretation of Microsoft's behavior regarding the development and pricing of Internet Explorer. There is nothing esoteric about this explanation – no strategic or dynamic considerations are at play. Just basic economics.

If this complementary demand interpretation of Microsoft's pricing behavior is correct, Microsoft's large and broad presence in software markets has highly beneficial effects for consumers and overall economic efficiency. By the same token, breaking Microsoft into an OS company and a separate software applications company, as the trial judge ordered in *U.S. v. Microsoft*, would lead to lower output and higher consumer prices in both market segments.

## *7.3 Complementarities with other Sources of Profits*

Another simple demand-based explanation for zero-price bundling emphasizes complementarities with profit opportunities in other markets, rather than complementarity in use with the OS. To borrow Klein's (1999) terminology, product A has "negative marginal costs" when its adoption and use by customers generate additional profits for the firm in other markets, say B and C, that exceed the marginal costs of producing, distributing and supporting product A. In principle, this form of complementarity can arise even when there is no direct complementarity in use between product A and products B or C. In practice, "negative marginal costs" may reinforce the bundling motive that stems from direct complementarity in use.

Web browsers like the IE browser shell and Netscape Navigator are examples of software with this potential. Greater use of Navigator, for instance, increased traffic flow

on Netscape's web site, which enabled Netscape to earn more from web advertisements and from commissions on Internet commerce.[88] The commissions arise in connection with revenue-sharing agreements that Netscape made with firms that sell products and services over the Internet. Netscape earned a commission on the sales that resulted when Navigator directed web traffic to another firm's web site.

This type of revenue-sharing arrangement is an important aspect of Internet commerce and is by no means limited to web browsers. For example, AOL and Yahoo! generate revenue through such agreements. PC manufacturers Compaq and Gateway have also taken steps to generate revenues from Internet commerce in connection with their OEM sales. These PC makers redesigned their keyboards to give prominent placement to the web sites and services of their on-line partners. When a customer uses a Compaq PC to access AOL, for example, Compaq gets a share of the customer's monthly AOL fees (Ramstad, 1998).

Klein (1999) stresses negative marginal costs of this sort in his explanation for Microsoft's vigorous efforts to distribute its browser software through zero-price bundling and integration with its OS products.[89] This negative marginal cost feature of browsers reinforces the complementarity motive for zero-price bundling that we developed above. In other words, negative marginal costs and complementarity in use with OS products are distinct forces that favor zero-price bundling. Either force alone can lead to zero-price bundling. When both forces are in play, the demand-based motive for zero-price bundling or integration with the OS becomes stronger.

[88] See Cusumano and Yoffie (1998), pages 36-38, 149-151, 200-201 and 325-328.
[89] Klein uses the term "packaging" to refer to bundling motivated by demand complementarities in use with the OS.

We think Klein overstates the role of "negative marginal costs" in Microsoft's decision to distribute Internet Explorer at no charge. Microsoft, unlike Netscape, generated little revenue from advertisements and commissions earned in connection with customer flow through its web portal. Furthermore, Microsoft's decision to componentize the design of Internet Explorer involved a sacrifice of potential portal-related revenues, because the componentized design made it easier for other companies to use IE technologies to build their own browsers and thereby direct customers to non-Microsoft web portals.[90]

Nonetheless, we agree with Klein's basic point that web browsers are highly complementary with other sources of profits. Prominent examples include Netscape itself, AOL and other online service providers, Internet aggregators likeYahoo!, and Internet retailers like Amazon.com. For this reason, it seems likely that the price of web browsers would have gravitated toward zero regardless of whether Microsoft had pursued a zero-price policy for Internet Explorer. Microsoft's actions simply accelerated the process.

Two additional remarks clarify the relationship between the negative marginal cost and complementarity-in-use explanations for zero-price bundling with OS products. First, browsers exhibit negative marginal costs because they are complementary to other activities with profit-making potential for the browser firm -- namely, Internet sales and web advertisements. In this respect, negative marginal costs involve a complementarity, but not a direct complementarity in use with the OS. Second, even the negative marginal

[90] We confirmed this point in our interview with Paul Maritz. According to Maritiz, Microsoft recognized that its decision to componentize IE, and especially its decision to license the componentized IE technology to AOL, would detract from the growth and revenue potential of MSN, Microsoft's web portal. However,

cost explanation for the zero-price bundling of browsers requires some form of complementarity with the OS. Otherwise, the browser could just as well be freely distributed on street corners rather than bundled with the OS. In addition to the complementarity in use at the heart of our first demand-based explanation for bundling, three complementarities in distribution play a role in the zero-price bundling of browsers with the OS: (a) It is cheaper to distribute browser and OS software together; (b) it is more convenient for customers to acquire them together; and (c) customers who place a high value on the OS are more likely to also place a high value on a browser.

## 7.4 Reducing the Diversity of Buyer Valuations

A third demand-based explanation for product bundling begins with the observation that buyer valuations of a bundle are often much less dispersed than valuations of the individual items in the bundle. So, by combining many items into a single package, the high and low valuations a customer attaches to particular items tend to average out. Consequently, a seller can much more confidently predict a customer's valuation of the bundle than of any one item in the bundle.

A newspaper is a good example of this type of bundled product. Customer valuations of the entire newspaper are much less dispersed than their valuations of individual sections devoted to sports, weather, international news, and so on. The same idea applies to individual articles. Customer valuations of particular articles about baseball's home run leaders, the likelihood that Michael Jordan will resume his NBA career and predictions for the upcoming football season are much more dispersed than the valuations attached to the sports section as a whole.

---

the full integration of Web-support functionality into Windows, including the componentization of IE, was

Uncertainty regarding the value that consumers place on individual products undermines effective pricing from the firm's standpoint. If individual valuations for a product are highly dispersed, the firm must choose between higher prices that exclude many consumers with low valuations and lower prices that forego substantial surplus to many consumers with high valuations. By bundling items together, the firm can reduce the diversity in customer valuations for the bundled product relative to the individual items -- or, at a minimum, reduce the diversity in customer valuations as a percentage of bundle value as its size grows.[91] The firm may then be able to set a price for the bundle that generates more revenue than it could obtain by separately pricing the individual items. Hence, provided that the costs of including multiple goods are not large, bundling can lead to higher profits.[92]

A simple numerical example shows how bundling can lead to higher profits and greater economic efficiency by reducing the diversity of buyer valuations. Consider a firm that owns the rights to 100 "information goods" such as software products. Assume the firm can replicate these items at zero cost. The firm sells the items in a market with anonymous buyers who differ in the value that they attach to individual items. In

---

viewed as essential to preserve the viability of the Windows platform.

[91] To be more precise, suppose that customer valuations have the same mean and standard deviation for each item. With a negative correlation across customers, even a weak one, in the valuations attached to individual items, the variance in valuations for the bundled product can shrink with bundle size. The example we develop below has this property. Even with a zero or positive correlation (but less than one) across customers in the valuations on individual items, the average valuation per item in the bundle converges to a constant by the law of large numbers, while the standard deviation of the valuation attached to the entire bundle grows less than proportionately to bundle size. Hence, the ratio of the standard deviation of the bundle value to the total bundle value declines with bundle size.

[92] This idea dates to Stigler (1963), who showed how bundling can increase profits when consumer valuations for two goods are negatively correlated. Schmalensee (1984) showed that bundling can increase profits even when consumer valuations of the two goods are uncorrelated or positively correlated. Bakos and Brynjolfsson (1999) develop this type of bundling theory in a direction that is especially applicable to information goods. McAfee, McMillan and Whinston (1989) and Bakos and Brynjolfsson consider motives for mixed bundling, in which the firm sells the same product separately and as part of a bundle. Adams and

particular, suppose that there are also 100 consumer types, indexed by i=1,2,...100. The ith consumer places a value of 101 on the ith item and a value of 1 on the other 99 items.[93] Consumers purchase one or zero units of each item, and there are equal numbers of each consumer type.

Given these assumptions, Table 2 displays the firm's profit-maximizing outcomes for bundles of various sizes. If the firm sells items individually, then it faces the following demand curve for each item: at a price less than or equal to 1, everyone buys; at a price greater than 1 but less than or equal to 101, only the high-valuation type buys; and at a price greater than 101, no one buys. Clearly, the firm will never find it advantageous to sell individual items for a price less than 1 or between 1 and 101. If the firm prices individual items at 1, it sells them to every type, and its profits on all items amount to 10,000 per hundred consumers. If it prices individual items at 101, it sells only to high types, and its profits amount to 10,100 per hundred consumers. So, given that the firm sells items individually, the profit-maximizing price is 101 and the total profit is 10,100 per hundred consumers. In this way, we determine the profit-maximizing outcomes for the first row in the table. The other rows are filled in using parallel logic.

---

Yellen (1976) provide an insightful early treatment of how bundling affects consumer surplus and economic efficiency.

[93] To be more precise, we assume that the ith consumer places a value of 101+z on the ith good, where z is a very small positive number that we ignore in the calculations below.

## Table 2. Profit-Maximizing Outcomes for Bundles of Various Sizes

| (1) | (2) | (3) | (4) | (5) | (6) | (7) |
|---|---|---|---|---|---|---|
| Bundle Size (# Of Types Who Buy Each Bundle) | Number of Product Bundles | Bundle Price | Price Per Item in Bundle (3)/(1) | Profits Per 100 Consumers (1)*(2)*(3) | Coefficient Of Variation of Bundle Valuations * | Fraction of Items Used by Each Consumer |
| 1 | 100 | 101 | 101 | 10,100 | 497.5 | 0.01 |
| 2 | 50 | 102 | 51 | 10,200 | 350.0 | 0.02 |
| 4 | 25 | 104 | 26 | 10,400 | 244.9 | 0.04 |
| 5 | 20 | 105 | 21 | 10,500 | 217.9 | 0.05 |
| 10 | 10 | 110 | 11 | 11,000 | 150.0 | 0.10 |
| 20 | 5 | 120 | 6 | 12,000 | 100.0 | 0.20 |
| 25 | 4 | 125 | 5 | 12,500 | 86.6 | 0.25 |
| 50 | 2 | 150 | 3 | 15,000 | 50.0 | 0.50 |
| 100 | 1 | 200 | 2 | 20,000 | 0.0 | 1.00 |

* Column (6) reports the coefficient of variation of buyer valuations for a bundle of the indicated size. The coefficient of variation is calculated as 100 times the standard deviation of buyer valuations divided by the mean of buyer valuations. The mean and standard deviations are calculated over all buyers, whether or not they actually purchase the bundle at the profit-maximizing price. The resulting coefficient of variation measures the diversity of buyer valuations placed on the bundle, expressed as a percentage of the mean valuation for the bundle. In this example, the coefficient of variation is the same for all bundles of a given size.

As the table shows, the firm maximizes profits in this example by selecting the largest bundle. This result reflects the effect of bundling on the diversity of buyer valuations for the bundled product relative to the individual items in the bundle. Indeed, in this simple example, *every* consumer attaches a value of *exactly* 200 to the bundle that contains all 100 items. More generally, column (6) shows that the diversity among

potential customers in the valuations attached to a bundle declines with bundle size. This diversity reduction aspect or "predictive value" of bundling leads a profit-maximizing firm to choose a lower price per item in a larger bundle. The firm more than makes up for the lower price per item by selling the bundled items to a larger number of customers. Total profits rise with bundle size, as indicated by column (5).[94]

The predictive value of bundling improves economic efficiency by promoting the widespread distribution and use of the firm's information goods. To understand why widespread distribution is efficient, recall that the firm can replicate its information goods at zero cost. Because replication is costless, and because every consumer type places a positive value on each information good in this example, economic efficiency calls for the widest possible distribution. Bundling leads to wider distribution and use, as shown in column (7) of the table. The largest bundle maximizes the distribution and use of the individual information goods.

A major barrier to this type of bundling in many cases is the cost of including additional items in the bundle. High marginal costs for individual items greatly reduce the attractiveness of bundling when customers place little or no value on many individual items. Even modest marginal costs can undermine bundling. If we modify the example in Table 1 so that the marginal cost of each item is 2 rather than 0, the profit-maximizing bundle size is one item, instead of 100 items. For this reason, bundling to reduce buyer

[94] In this example, bundling raises profits by facilitating consumer acquisition of low-valuation items that they would not otherwise buy. Bundling can also raise profits by enabling the firm to more effectively extract consumer surplus on high-valuation items that consumers would buy in any event. To see this point, modify the example in the text by introducing 100 new consumer types as follows. Assume that the ith new type places a value of 200 on the ith item and a value of zero on the other 99 items, for i=1,2,…100. Provided that the number of new-type consumers is sufficiently small, the profit-maximizing price at each bundle size is the same as in Table 2. Furthermore, efficiency and profits still rise with bundle size.

diversity is much more attractive for information goods than for physical goods, which typically involve non-negligible marginal costs. For information goods stored in electronic form, the marginal costs of replication and transmission are near zero.

These observations help explain important aspects of pricing behavior by firms that sell digital products. For example, many Internet aggregators provide access to enormous information libraries and a wide range of services for a single flat fee. For $23.95 a month (as of July 2001), AOL provides unlimited access to a wide range of services including Internet access, stock quotes, foreign exchange and commodity market information, e-mail and instant messaging to other online users. All of these services are accessible from local phone numbers in most localities in the United States.

E-library (http://www.elibrary.com) is another example. Bakos and Brynjolfsson (1999) write: "As of 1997 and 1998, E-library provides access to a bundle of 150 newspapers, 800 magazines, 2,000 works of literature, 18,000 photos, and thousands of additional information goods for a fixed price of $59.95 per year for individual users."

The same economic rationale that explains the bundling approach adopted by AOL, E-library and many other Internet firms also helps explain bundling by software firms. The key elements in this theory of software bundling are low marginal costs and a desire to reduce the diversity of buyer valuations. Hence, software features or applications that create large customer support burdens are unlikely to be bundled if the only motive is a reduction in the diversity of buyer valuations.

---

However, consumer surplus now declines with bundle size. For a more general treatment, see Bakos and Brynjolfsson (1999, 2000).

Unlike the complementarity-based explanation for bundling, the buyer diversity explanation requires no particular relationship among the bundled items. Thus, according to this theory, it is not surprising that a grab-bag collection of software utilities and applications are often bundled with OS products. For example, an OS upgrade might contain some Internet utilities, Plug-and-Play functions, Multimedia support, WebTV, and 1394/USB support.[95]

Product bundling motivated by a desire to reduce buyer diversity promotes economic efficiency for much the same reason that it allows firms to increase profits. The predictive value of bundling leads firms to price and market low marginal cost goods in such a way that they become more widely distributed. Consequently, consumers acquire and use many products that they would not purchase separately in the absence of bundling. For goods with a zero marginal cost of replication and transmission, the widespread distribution and use promoted by bundling is an efficiency-enhancing outcome.

The consumer welfare implications of bundling motivated by a desire to reduce buyer diversity are less clear. Salinger (1995) considers a case where bundling leads to higher consumer welfare. Bakos and Brynjolfsson (1999) make clear that this result is not general. However, both analyses approach the consumer welfare effects of bundling from a static perspective. In a dynamic setting, the higher profits generated by bundling provide incentives to develop the items or features in the bundle.[96] This point is

[95] 1394 and USB refer to standards for high-speed serial communication.
[96] Even in a static setting, bundling may enable a firm to earn enough revenues to cover fixed costs and continue serving customers, when it would otherwise exit the market. See Bakos and Brynjolfsson (2000).

especially pertinent in the context of software and other information goods, because their costs of development are often high, even if their costs of replication are low.

### *7.5 Summary of Demand-Based Motives for Bundling*

In summary, this section develops three purely demand-based explanations for the zero-price bundling or integration of software utilities and applications with the OS. First, the complementarity in use between the OS and applications software will in some circumstances make it optimal to price the application at zero; i.e., to bundle it with the OS at no extra charge. Second, when the application involves a "negative marginal cost" to the supplier, in the sense that greater customer use sufficiently enhances profit for the seller in other markets, zero-price bundling with the OS is again optimal. Third, bundling of software features and products can increase profits and lead to more widespread product distribution by reducing the diversity of buyer valuations.

Low marginal costs facilitate bundling under the complementarity explanation and are especially important in the buyer-diversity explanation. The complementarity and buyer-diversity explanations both predict that software features with high customer support costs are less likely to be bundled with the OS.[97] Demand complementarities are an essential element in the first explanation for bundling that we developed in this section, but they are inessential to the other two. All three demand-based explanations are quite distinct from, but compatible with, the explanations that we developed in Section 6 under the heading of "Efficient Management of Interacting Features and Components."

[97] Section 6.3 points out that some software features are integrated into the OS as a method to *reduce* customer support costs. That argument is distinct from the explanations for bundling developed in this

Our analysis of demand-based motives for bundling carries important implications for thinking about market structure, consumer welfare and economic efficiency in the software industry. For example, the complementary nature of demand for OS software and web browsing software implies a simple and plausible interpretation of Microsoft's decision to spend hundreds of millions of dollars developing and promoting Internet Explorer, a so-called "zero-revenue product". This complementary demand explanation involves no complicated dynamic or strategic behavior. And there is nothing anti-competitive or predatory about Microsoft's pricing and product development decisions under this simple interpretation. In fact, consumers are better off, and economic efficiency is greater.

# 8. Concluding Remarks

Improvements in the software that provides hardware management, user interface and platform functions have played a central role in the growth and transformation of the personal computer (PC) industry. Several forces shape the design of these operating system products and propel their evolution, including:

- The need to efficiently manage the interacting components of PC systems so as to keep pace with rapid advances in computer technologies, simplify computer use and facilitate the development of applications software.
- The need to maintain compatibility with existing applications while preserving the flexibility to incorporate additional functions that support new applications.

---

section. In particular, the complementarity and buyer-diversity explanations for bundling apply with greater force for software applications that *have low* marginal costs.

- The desire to economize on customer support costs and assign clear responsibility for making the interacting components of the PC work together.
- The desire to bundle multiple software features into a single package so as to more effectively meet the demand for complementary applications or reduce the diversity in product valuations among consumers.

The integration and bundling of new features and functions into operating system products spurs growth in the PC industry and fosters innovation through several channels. By making PC systems easier to set up and use, integration opens the door to new, non-integrated hardware and software products. It also expands the number of PC users and the range of PC uses. In addition, the integration of APIs (software building blocks) into operating system products enables applications developers to concentrate on their areas of expertise. This specialization leads to an increase in the quality, number and variety of software applications. Note, too, that the integration of widely used features and software development tools into operating system products promotes a standard computing environment. As a separate point, demand-based motives for bundling applications and utilities with operating system products lead to wider and cheaper distribution of software among PC users. Bundling also adds to consumer welfare by stimulating the development of software applications that would otherwise be unprofitable.

These beneficial effects of operating system integration (and bundling) enlarge the market for both software and hardware. Because scale economies are important in the computer industry, the market-enlarging effects of integration mean greater enjoyment of network benefits (as PC usage grows) and lower average costs (as up-front product development expenditures are spread more widely). Hence, the full benefits derived from

adding features and functions to operating system products are greater, perhaps much greater, than the immediate benefits. In short, our analysis indicates that the integration and bundling of new features and functions into PC operating system products have been highly beneficial for consumers and a major stimulus to growth and innovation in the computer industry.

We thus conclude that judicial or regulatory restrictions on software design would likely retard innovation in the computer industry and hurt consumers. To be sure, there are circumstances under which it is feasible and profitable for a firm with market power to design products for anti-competitive purposes, and to harm consumers in the process. Whinston (1990) shows that product design can be used to exclude a rival from the market for a tied good while raising the tying firm's profit *and* harming consumers.[98] Farrell and Katz (2000) identify conditions under which product integration by the monopoly supplier of one component in a system can reduce the incentives for innovation by other firms. And a firm might intentionally design a platform product to raise costs for rivals who compete in the sale of complementary applications.

A full assessment of the issues raised by anti-competitive product designs is beyond the scope of this study. However, a few points should help to place the matter in perspective.[99] First, theoretical demonstrations of profitable, but anti-competitive, product design are fragile in the sense that they do not survive natural modifications to the underlying assumptions. Second, the circumstances that give rise to the possibility of

[98] See Carlton and Waldman (2000) for a related analysis that is motivated by allegations in *U.S. v. Microsoft.*

[99] See Hylton and Salinger (2001) and Evans and Schmalensee (2001) for an extended treatment of these issues. Also, see Easterbrook (2000) and Posner (2000), who emphasize the limited capacity of the judicial process and antitrust enforcement machinery to effectively respond to allegedly anti-competitive conduct in information and technology-intensive sectors. Easterbrook and Posner are federal judges who sit on the

anti-competitive product designs also give rise to the possibility of other harmful, anti-competitive strategies. Hence, it is unclear whether product design restrictions can prevent or ameliorate anti-competitive conduct. Third, as a practical matter, it can be extremely difficult to distinguish anti-competitive product designs from pro-competitive designs, or to determine whether the harm caused by an allegedly anti-competitive design outweighs the beneficial effects.[100] Fourth, even if it were possible to discern, say, anti-competitive forms of integration and bundling and respond with legal restrictions on design that address the underlying problem, there would remain the danger that the design restrictions would impede beneficial forms of integration and bundling. As this study shows, highly beneficial forms of product integration and bundling are ubiquitous for software platforms and software products generally.[101] These points reinforce our view that legal restrictions on the design of software products are likely to slow innovation and harm consumers.

As we stressed at the outset, software is inherently malleable, and competition in many software product categories revolves around innovation. This reality partly motivated our focus on design issues rather than the pricing of software. Of course, pricing also matters because it directly affects consumer welfare, firms' profits and the incentives to innovate. Pricing and design also interact in many ways in addition to the connection between pricing and bundling that we discussed – see Shapiro and Varian (1999).

---

U.S. Court of Appeals. Davis, MacCrisken and Murphy (1999, Section IX) consider whether a platform supplier can profit by adopting a design that raises costs for rival suppliers of complementary applications.

[100] Harm to rivals and the exclusion of rivals from a significant share of sales are not good indicators of anti-competitive behavior or consumer harm. For example, Davis, Murphy and Topel (2001) show that a product design that excludes rival sales can be highly beneficial for consumers.

[101] Tying can also have other benefits not emphasized in our study such as lower production costs.

Davis, Murphy and Topel (2001) stress that product design changes can either intensify or soften price competition. Rival firms with secure positions in the same market have strong incentives to differentiate their products in ways that relax price competition.[102] To bring this about, a firm can focus on design improvements that appeal more strongly to its existing customers than to its rival's customers. In contrast, when a firm seeks to displace a rival, it becomes attractive to intensify price competition by designing product improvements that appeal strongly to the rival's customers.

The durability of software raises dynamic pricing issues of the sort identified by Coase (1972). The seller of any durable product competes against its own past and future sales, a fact that can strongly affect the profitability of alternative design strategies. See the analyses by Fudenberg and Tirole (1998) and Ellison and Fudenberg (2000) and our discussion in Davis, MacCrisken and Murphy (1999). Product design choices also play an important role in the evolution of standards for system products and affect compatibility with products supplied by other firms. Besen and Farrell (1994), Katz and Shapiro (1994) and Shapiro and Varian (1999) contain useful discussions on these topics and references to related work.

[102] See, also, Shaked and Sutton (1982), Economides (1986) and Chapter 7 in Tirole (1988).

## References


Adams, William James and Janet I. Yellen, 1976, "Commodity Bundling and the Burden of Monopoly," *Quarterly Journal of Economics, 90,* no. 3, 475-498.

Bakos, Yannis and Erik Brynjolfsson, 1999, "Bundling Information Goods: Pricing, Profits and Efficiency," *Management Science, 45,* no. 12 (December).

Bakos, Yannis and Erik Brynjolfsson, 2000, "Bundling and Competition on the Internet: Aggregation Strategies for Information Goods" *Marketing Science* (January).

Baldwin, Carliss Y. and Kim B. Clark, 2000, Design Rules: Volume 1, The Power of Modularity. Cambridge, Massachusetts: The MIT Press.

Berndt, Ernst R. and Neal J. Rappaport, 2001, "Price and Quality of Desktop and Mobile Personal Computers: A Quarter-Century Historical Overview" *American Economic Review, 91,* no. 2 (May), 268-273.

Besen, Stanley and Joseph Farrell, 1994, "Choosing How to Compete: Strategies and Tactics in Standardization," *Journal of Economic Perspectives, 3,* no. 2.

Boyce, Jim, 1998, Inside Windows 98. New Riders Publishing.

Carlton, Dennis W. and Michael Waldman, 2000, "The Strategic Use of Tying to Preserve and Create Market Power in Evolving Industries," George J. Stigler Center for the Study of the Economy and the State, Working Paper No. 145 (March). Available at http://gsbwww.uchicago.edu/research/cses/WorkingPapersPDF's/145.pdf.

Coase, R.H., 1972, "Durability and Monopoly," *Journal of Law and Economics*, 15, 149-149.

Computer Language Company, 1999, *Computer Desktop Encyclopedia.* Point Pleasant, Pennsylvania. Searchable at http://www.computerlanguage.com/sitemain/content.html.

Cusumano, Michael A., and Richard W. Selby, 1995, Microsoft Secrets. New York: Simon & Schuster.

Cusumano, Michael A. and David B. Yoffie, 1998, Competing on Internet Time: Lessons from Netscape and its Battle with Microsoft. New York: The Free Press.

Davis, Steven J., Jack MacCrisken and Kevin M. Murphy, 1999, "The Evolution of the PC Operating System: An Economic Analysis of Software Design," June 29. Available at http://gsbwww.uchicago.edu/fac/steven.davis/research.

Davis, Steven J. and Kevin M. Murphy, 2000, "A Competitive Perspective on Internet Explorer," *American Economic Review, 90,* no. 2, 184-187. Republished in expanded form with mathematical supplement in this volume. Available at http://gsbwww.uchicago.edu/fac/steven.davis/research.

Davis, Steven J., Kevin M. Murphy and Robert H.Topel, 2001, "Entry, Pricing and Product Design in an Initially Monopolized Market." Available at http://gsbwww.uchicago.edu/fac/steven.davis/research.

Easterbrook, Frank H., 2000, "Information and Antitrust". Keynote address delivered at a symposium on "Antitrust in the Information Age" at the University of Chicago on October 29, 1999. Revised version is dated October 4, 2000.

Economides, Nicholas, 1986, "Minimal and Maximal Product Differentiation in Hotelling's Duopoly," *Economic Letters, 21,* 67-71.

Economides, Nicholas, 2000, "The Microsoft Antitrust Case," Working paper 2000-09, Stern School of Business, New York University. Available at http://www.stern.nyu.edu/networks.

Ellison, Glenn and Drew Fudenberg, 2000, "The Neo-Luddite's Lament: Excessive Upgrades in the Software Industry," *Rand Journal of Economics, 31,* no. 2 (Summer), 253-272.

Evans, David S., Albert Nichols and Bernard Reddy, 1999, "The Rise and Fall of Leaders in Personal Computer Software," National Economic Research Associates. Available at http://www.neramicrosoft.com/NeraDocuments/Analyses.

Evans, David S., Albert Nichols and Richard Schmalensee, 2001, "An Analysis of the Government's Case in *U.S. v. Microsoft,*" *Antitrust Bulletin, 46,* no. 2 *(*Summer).

Evans, David S. and Richard Schmalensee, 2001, "Some Economic Aspects of Antitrust Analysis in Dynamically Competitive Industries," NBER Working Paper 8268 (May).

Farrell, Joseph and Michael L. Katz, 2000, "Innovation, Rent Extraction, and Integration in Systems Markets," *Journal of Industrial Economics, 48,* no. 4 (December), 413-432

Fisher, Franklin M., 2000, "The IBM and Microsoft Cases: What's the Difference?" *American Economic Review, 90,* no. 2 (May), 180-183.

Fisher, Franklin M. and Daniel L. Rubinfeld 2000, "United States v. Microsoft: An Economic Analysis," in Did Microsoft Harm Consumers? Two Opposing Views. Washington, DC: AEI-Brookings Joint Center for Regulatory Studies.

Fudenberg, Drew and Jean Tirole, 1998, "Upgrades, Trade-Ins and Buybacks" *Rand Journal of Economics, 29,* no. 2 (Summer).

Gans, Joshua S., David H. Hsu and Scott Stern, 2000, "When Does Start-Up Innovation Spur the Gale of Creative Destruction?" NBER Working Paper 7851.

Gilbert, Richard J. and Michael L. Katz, 2001, "An Economist's Guide to *U.S. v. Microsoft*," *Journal of Economic Perspectives, 15,* no. 2 (Spring), 25-44.

Gookin, Dan, 1992, "Windows 3.1 vs. OS/2 2.0," *InfoWorld.*

Hylton, Keith N. and Michael Salinger, 2001, "Tying Law and Policy: A Decision Theoretic Approach," Boston University School of Law Working Paper Series, Law and Economics Working Paper No. 01-04. Available at http://www.bu.edu/law/faculty/papers.

Ichbiah, Daniel and Susan L. Knepper, 1991, The Making of Microsoft. Prima Publishing.

Katz, Michael and Carl Shapiro, 1994, "Systems Competition and Network Effects," *Journal of Economic Perspectives, 3,* no. 2.

Khanna, Tarun and David Yoffie, 1996, "Microsoft, 1995," Harvard Business School Case Study 9-795-147.

Klein, Benjamin, 1999, "Microsoft's Use of Zero Price Bundling to Fight the Browser Wars" in Jeffrey A. Eisenach and Thomas M. Lenard, editors, Competition, Convergence and the Microsoft Monopoly. Boston, MA: Kluwer Academic Publishers for the Progress & Freedom Foundation.

Klein, Benjamin, 2001, "The Microsoft Case: What Can a Dominant Firm Do to Defend Its Market Position?" *Journal of Economic Perspectives, 15,* no. 2 (Spring), 45-62.

Liebowitz, Stanley J. and Stephen E. Margolis, 1999, Winners, Losers and Microsoft: Competition and Antitrust in High Technology. Oakland, California: The Independent Institute.

Markoff, A. 1996, "Tomorrow, the World Wide Web," *New York Times*, July 16, page D1.

McAfee, R. Preston, J. McMillan and Michael D. Whinston, 1989, "Multiproduct Monopoly, Commodity Bundling, and Correlation of Values," *Quarterly Journal of Economics*, 104 (May), 371-384.

McCartney, Laton, 1986, "The Pros and Cons of Going Multi-Vendor (Practicality of Companies Utilizing More Than One Vendor For Computer Systems)," *Dun's Business Month*, 128 (November), 93.

Microsoft, 1995, "Microsoft Developer Relations: Proving Microsoft's Commitment to Third Parties," Microsoft Corporation White Paper.

Microsoft, 1998, Microsoft Internet Explorer Resource Kit. Microsoft Press.

Ramstad, Evan, 1998, "PC Makers Hunt for Gold in Internet Hookups," *Wall Street Journal*, August 12, Page B1.

Reid, Robert H., 1997, Architects of the Web: 1,000 Days that Built the Future of Business. New York: Wiley.

Reinganum, Jennifer F., 1985, "Innovation and Industry Evolution," *Quarterly Journal of Economics, 99,* no. 1, 81-99.

Posner, Richard, 2000, "Antitrust in the New Economy." Address delivered to a conference on antitrust sponsored by the American Law Institute-American Bar Association Committee on Continuing Professional Education, September 14, 2000, New York and published in *Tech Law Journal.* Available at http://www.techlawjournal.com/atr/-2000914posner.asp.

Salinger, MA, 1995, "A Graphical Analysis of Bundling," *Journal of Business*, 68, no. 1, 85-98.

Schmalensee, Richard L., 1984, "Gaussian Demand and Commodity Bundling," *Journal of Business*, 57 (January), S211-S230.

Shaked, Avner and John Sutton, 1982, "Relaxing Price Competition through Product Differentiation," *Review of Economic Studies, 49,* 3-17.

Shapiro, Carl and Michael L. Katz, 1999, "Antitrust in Software Markets," in Jeffrey a. Eisenach and Thomas M. Lenards, editors, Competition, Innovation and the Microsoft Monopoly: Antitrust in the Digital Marketplace. Boston, MA: Kluwer Academic Publishers for the Progress & Freedom Foundation.

Shapiro, Carl and Hal R. Varian, 1999, Information Rules: A Strategic Guide to the Network Economy. Boston, MA: Harvard Business School Press.

Stigler, George J., 1963, "United States v. Loew's, Inc.: A Note on Block Booking," *Supreme Court Review*, pp. 152-157.

Tirole, Jean, 1988, The *Theory of Industrial Organization*. Cambridge, MA: MIT Press.

Vickers, John, 1986, "The Evolution of Market Structure When There Is a Sequence of Innovations," *Journal of Industrial Economics, 35,* no. 1, 1-12.

Whinston, Michael, D., 1990, "Tying, Foreclosure, and Exclusion," *American Economic Review, 80,* no. 4 (September), 837-859.

Whinston, Michael, D., 2001, "Exclusivity and Tying in *U.S. v. Microsoft:* What We Know, and Don't Know," *Journal of Economic Perspectives, 15,* no. 2 (Spring), 63-80.

Wildstrom, Stephen H., 1998, "Build Your Own Browser," *Business Week*, July 20, Page 17.