03-020

Intellectual Property, Architecture, and the Management of Technological Transitions: Evidence from Microsoft Corporation

Alan MacCormack

Marco Iansiti

Abstract

Many studies highlight the challenges facing incumbent firms in responding effectively to major technological transitions. While some authors argue that these challenges can be overcome by firms possessing what have been called "dynamic capabilities," little work has described in detail the critical resources that these capabilities leverage, or the processes through which these resources accumulate and evolve. This paper explores these issues through an in-depth exploratory case study of one firm that has demonstrated consistently strong performance in an industry that is highly dynamic and uncertain.

The focus for our study is Microsoft, the leading firm in the software industry. We motivate our focus on Microsoft by providing evidence that the firm's product performance has been consistently strong over a period of time in which there have been several major technological transitions – one indicator that a firm possesses dynamic capabilities. We support our argument by showing that Microsoft's performance developing new products in response to one of these transitions – the rise of the Internet – was superior to a sample of both incumbents and new entrants.

We present qualitative data that describes the roots of Microsoft's dynamic capabilities, highlighting the way that the firm develops, stores and evolves its intellectual property to address new market challenges. Specifically, Microsoft codifies knowledge in the form of software "components," which can be leveraged across multiple product lines over time, and accessed by firms developing complementary products. We argue that the process of componentization, the component "libraries" that result, the architectural frameworks that define how these components interact, and the process through which these components are evolved to address environmental changes represent critical resources enabling the firm to respond to major technological transitions. We illustrate our arguments by describing Microsoft's response to two recent transitions.

I. Introduction

What explains the success of incumbent firms in one of the world's most turbulent industries? Despite a strong tradition of research in the field of technological evolution across a variety of environments, (Clark and Fujimoto, 1991; Henderson and Cockburn, 1994; Iansiti and West, 1997; Christensen, 1997; Pisano, 1996) little research has analyzed the management of technological transitions in the software industry (Iansiti and MacCormack, 1997; Cusumano and Selby, 1995). Yet this industry provides fertile ground for such a study; since the late 1970's, it has been characterized by a number of significant changes, including architectural innovations, disruptive technologies and competence destroying developments (Tushman and Anderson, 1986; Henderson and Clark, 1990; Baldwin and Clark, 1999). Concurrent with these changes, record amounts of venture capital have been poured into new ventures, with the explicit aim of tipping the balance of power away from industry incumbents (Gompers and Lerner, 1999; 2001).

In such an environment, one might have predicted that industry leadership would be a fleeting, transient experience for firms. Yet across several segments of the software industry, leadership has remained remarkably stable, as incumbents have adapted to a variety of deep-rooted changes in technology, market and business models. How have these leaders, companies such as Microsoft, IBM and Oracle, been able to hold on to their positions in the industry? With regard to the former at least, much of the popular press has focused on market power and Microsoft's dominant position in PC operating systems. Recently however, academics have begun to offer alternatives to such economic explanations (see, for example, Christensen 2001). In this paper, we contribute to the set of alternative explanations, focusing on the role played by "dynamic capabilities" and in particular, the critical resources that these capabilities build and leverage (Teece, Pisano and Shuen, 1997; Eisenhardt and Martin, 2000).

Our study focuses on Microsoft Corporation, arguably the most successful software firm to have ever existed. We show that Microsoft has exhibited consistently strong product performance relative to competitors over the last 15 years, despite significant technological and market changes during this period. We argue that this consistency in performance, given the rapidly changing environment, is one indicator that the firm possesses dynamic capability (Iansiti and Clark, 1994). We support this argument with quantitative data on Microsoft's response to one recent technological transition – the rise of the World Wide Web. Specifically, we show that Microsoft's first efforts to develop a new product in this segment – the Internet Explorer browser – were more productive than a sample of comparable projects completed by other firms, many of which were newly founded to exploit this major technological shift. These data serve to motivate the main focus of our paper: An inductive study of the critical resources within Microsoft that provide the source of its dynamic capability, using qualitative data collected via an extensive interview program conducted at the firm.

We find that Microsoft's success in maintaining a position of industry leadership can be traced, in part, to the manner in which it manages its intellectual property ("IP") foundations. This approach is based upon a combination of tools, architectural frameworks and components whose roots are over 20 years old. At its heart lies Microsoft's process of software componentization and the resulting set of code libraries, which in combination, make up a critical and evolving set of firm-specific resources (Penrose, 1959; Wernerfelt, 1984; Eisenhardt and Martin, 2000). We describe how

Microsoft has developed, deployed and evolved these resources over time, illustrating our arguments with insights gained from field-based interviews.

Microsoft's "component model" helps it sustain industry leadership in two ways. The direct impact is in the area of product development, where the use of software components enables the organization to leverage resources across multiple product lines, and over time. When new technologies emerge, Microsoft has a framework in place that allows them to capture the benefits of these technologies in a way that is compatible with its existing component base. But this software component model also has impact *outside* Microsoft's organizational boundaries, by virtue of its distribution as part of the firm's development environment and programming tools. The component model is central to the company's interaction with a vast community of customers and third-party development partners (Iansiti and Levien, 2002). Microsoft adds value to this community by providing access to valuable intellectual property, along with productivity-enhancing tools. In return, the community enhances Microsoft's own position by developing complementary products that extend the impact of Microsoft's own product lines.

Our observations extend the literature examining the effective management of technological transitions (e.g., Tushman and O'Reilly, 1997; Brown and Eisenhardt 1997; Gawer and Cusumano 2002). In particular, we highlight some of the critical resources and capabilities required to create, leverage and evolve a company's intellectual property base in an uncertain environment. Importantly, our approach to analyzing intellectual property goes beyond patent analysis (e.g., Griliches, 1990; Jaffe and Trajtenberg, 2000) to directly examine the evolution of Microsoft's software code libraries and the intellectual property that these capture. These observations have implications for managers in a variety of IP-oriented industries.

The paper is divided into five main sections. In section II, we provide the conceptual motivation for our work. In section III, we describe the context for the study and motivate our focus on Microsoft using historical data on the firm's performance at both the product and project level. We describe our research methods in section IV, which were based on an extensive interview program conducted at the company. In section V, we report the findings from our work, following this with a discussion of how these findings contribute to the literature in section VI.

II. Conceptual Foundations

Organizations often respond inappropriately to technological challenges. While a variety of academics have analyzed these difficulties, the reasons reported for firms' failures tend to vary. Tushman and Anderson (1986) for example, describe how firms in a variety of industries stumbled when faced with technological transitions that rendered their existing competencies obsolete. Henderson and Clark (1990) by contrast, find that even when firms possess the required competencies to overcome a transition, they may not *recognize* the ways in which these need to be recombined in a new product architecture. Finally, Christensen and Bower (1997) show that firms often fail to *invest* in new technologies that eventually come to dominate a market, because their resource allocation processes are overly focused on the needs of *current* customers.

While the studies described above shed light on different aspects of the organizational challenges facing firms, they share a common theme: Established firms are prone to failure when faced with major technological transitions. Yet these dynamics are by no means inevitable. Several studies provide evidence that incumbents can prosper from periods of industry upheaval, given they have valuable resources and/or complementary assets that entrants do not possess (Tripsas, 1997; Westerman, Iansiti and McFarlan, 2002). And even in the research contexts mentioned previously, there are examples of incumbent success. For example, in Tushman and Anderson's study, four of eleven "competence destroying" innovations were introduced by existing firms. What then differentiates those firms that succeed in overcoming major technological transitions from those that don't?

One possible answer lies in a line of research that emphasizes the ability of firms to cope with uncertainty and change. This work asserts that the challenges raised by discontinuous technological change can be overcome by firms possessing what have been called "dynamic capabilities" (Levinthal and March, 1981; Nelson and Winter, 1982; Iansiti and Clark, 1994; Teece, Pisano and Shuen, 1997; Eisenhardt and Martin, 2000). This emphasis on capabilities has strong foundations. In recent years, a variety of empirical studies have documented large and persistent variations in operational performance between firms (e.g., Garvin, 1987; Clark and Fujimoto, 1991; Henderson and Cockburn, 1992; Iansiti, 1997; Cusumano and Selby, 1995). Furthermore, recent work has emphasized the *dynamic* nature of the capabilities that are required when firms must compete in uncertain and rapidly changing environments (e.g., Nelson and Winter, 1982; Wernerfelt, 1984; Hayes, Wheelwright, and Clark, 1988; Hamel and Prahalad, 1994; Leonard-Barton, 1992; Brown and Eisenhardt, 1997; Eisenhardt and Martin, 2000). In such environments, firms do not have the luxury of competing through the development of traditional “static” competences. Rather they need to develop the ability to *sense* changes in their environments, and thereafter mount an effective *response* to these changes. In this respect, dynamic capabilities can be defined as a “firm’s ability to integrate, build and reconfigure internal and external competencies to address rapidly changing environments” (Teece, Pisano and Shuen, 1997).

Despite the appeal of a capability-based view, this line of research has come under fire. Some authors criticize it for not capturing with enough detail and persuasiveness, exactly what dynamic capabilities consist of. Others criticize it for not describing the processes through which these capabilities evolve or the manner in which they interact with a firm’s resource base (Eisenhardt and Martin, 2000). In response to such concerns, researchers have begun to focus more on the *operational* foundations of a firm’s capability base and on the processes that underlie its evolution. For example, Iansiti and Clark (1994) describe how dynamic capabilities can rely on the effective execution of product development and problem solving processes within firms. Eisenhardt and Martin (2000) assert that dynamic capabilities can be identified in a variety of business processes, including product development, strategic decision making, and resource allocation. Importantly, these authors point to the importance of the critical resources that such capabilities build and leverage (Wernerfelt, 1984; Barney, 1991). It is this evolving resource base that differentiates the firm, enabling it to respond effectively to major technological change. As Eisenhardt and Martin (2000) argue, “long-term advantage lies in the resource configurations that managers build using dynamic capabilities, not in the capabilities themselves.”

The aim of this paper is to build a more detailed understanding of the dynamic capabilities through which firms overcome major technological transitions and thereby sustain positions of industry leadership. Specifically, we aim to characterize the critical resources that dynamic capabilities build and leverage as well as the processes through which these resources accumulate and evolve. In the next section, we describe the context for this work and motivate our focus on a single firm.

III. Research Context

Given the objective of our study, we needed to identify a dynamic and uncertain industry that had experienced a number of major technological transitions that would be expected to challenge industry incumbents. We then needed to identify a firm that had performed at a high level over the period in which these transitions occurred, assessing the weight of evidence as to whether this performance was attributable, at least in part, to its possessing dynamic capabilities. With respect to the first of these tasks, we focused our attention on evaluating various sectors of the computer industry, given the deep contextual knowledge we had acquired during several prior studies conducted in this industry (e.g., see Iansiti, 1997; MacCormack 2001). The final choice of sector was determined by assessing which had experienced the greatest number of *recent* technological transitions, given this would allow us to gather qualitative data from interview participants on events they were more likely to recollect accurately. This led us to focus on the software industry.

The software industry is an environment characterized by frequent major technological transitions. Table 1 lists a number of transitions affecting the software industry over the period from 1980 onwards.[1] Each has brought a significant technological and/or market dislocation, affecting the capabilities, market relationships and business models of existing firms. For example, the transition to graphical computing greatly expanded the community of potential computer users, promoting a variety of new software applications focused on less savvy consumers as opposed to sophisticated business users. This had a consequent impact on the capabilities required to develop products for these users. Alternatively, consider the rise of the World Wide Web, which brought the need to incorporate new communication protocols, standards and languages (e.g., TCP/IP, HTTP, HTML) into existing products, while generating whole new categories of applications (e.g., Web browsers and Web servers).

Given the frequency with which major technological transitions occur in the software industry, one might imagine that a position of sustained industry leadership would be difficult to achieve. Some firms however, have succeeded in overcoming these transitions, and indeed, appear to have prospered from the associated turbulence. In this respect, Microsoft's performance over this time period is both remarkable and unique. Founded in 1975, Microsoft grew to become the world's most valuable software firm by 1987. At that time, it accounted for 14.8% of the total market capitalization of all firms listed under the software Standard Industry Classification (SIC) code. By February 2002,

[1] We established this list from interviews with 8 experts from a variety of leading firms in the software industry. We report here a selection of the most important transitions mentioned by these experts.

that figure had grown to 52.6%[2]. Additional evidence of Microsoft's sustained high performance comes from the fact that since it went public in 1986, it has reported increased revenues in every year and increased profits in every year but two (figure 1).

Consistently high performance over a period of time in which several major technological transitions occur is one indicator that a firm might possess dynamic capabilities (Iansiti and Clark (1994). However, there are competing explanations for Microsoft's strong *financial* performance, many of which center on the firm's market power and dominant position in PC operating systems[3] (although these arguments do not explain how Microsoft attained such power in the first place). We therefore explored some *operational* measures of Microsoft's performance at the level of individual products, these being less affected by the impact of industry market structure. We argue that consistent excellence across a firm's product line, over a period of time in which several major technological transitions occur, is a likely indicator of dynamic capability.

We first report figures from a database of independent product reviews appearing in the industry trade press over the period from 1986 to 2001 (Appendix A describes how this database was assembled). These reviews evaluate the performance of products in a specific market segment (e.g., word processors) and award a "win" to the product (or products) found to be "best-in-class." Table 2 shows the performance of each Microsoft product in the database in terms of product wins. The percentage of wins is typically high, exceeding 70% for 6 of the 10 products analyzed. Only three of the products have won less than 50% of the reviews for their respective market segments over the period

[2] The 2002 figure was exceeded only once, in 1998, when Microsoft accounted for 54.3% of the total market capitalization of firms in the software SIC code.
[3] Industry analysts typically quote Microsoft as having over 90% market share of PC operating systems.

examined[4]. Figure 2 shows the aggregate performance of Microsoft's products by year. The average annual win rate is around 67%, indicating that Microsoft products won two-thirds of the independent product reviews over the period examined. Only once in 16 years does the figure drop below 50%. Given the number and diversity of competitors faced in each different product category, we can conclude that Microsoft's product performance has been consistently high, both across multiple products and over time[5].

While the performance of Microsoft's products is striking, questions still remain with respect to whether dynamic capabilities provide the most likely explanation. Most importantly, it is possible that Microsoft's scale advantages and deep pockets explain the results, given product reviews take no account of the resources involved in creating a product. To put it bluntly, Microsoft may throw more people at projects than competitors do. Further concerns surround the question of Microsoft's performance specifically at the time of technological transitions. The strong performance observed across its existing product lines over time might serve to obscure the poor performance of a few more critical projects aimed at overcoming each technological transition.

In response to these concerns, we report data on Microsoft's performance developing a new product in responding to one recent technological transition – the rise of the World Wide Web. This transition began with the adoption of a set of open communication standards (e.g., TCP/IP, HTTP and HTML) which provided a foundation

---

[4] The number of reviews is not constant over time. For example, more reviews were available for earlier versions of Internet Explorer that were inferior to competing browsers than for later versions that outperformed competitors in all reviews after 1997. This, in part, explains Explorer's low overall win rate.
[5] Further evidence comes from comparing each Microsoft product against the strongest competitor in each segment. In 7 out of 10 segments, the difference in win rate between Microsoft's product and the nearest competitor is statistically significant ($p<0.1$). Note that we use the strongest competitor at any point in time over the sample period. Some competitive products (e.g., OS/2) were not offered for the entire period.

for creating "sites" on the Internet[6] that could be accessed using a new graphical interface application – the "browser." Our analysis involved comparing Microsoft's performance in developing the first versions of its browser – Internet Explorer – to a sample of Internet software projects completed within other firms at the same time. Many of these other firms were new entrants formed specifically to take advantage of the rise of the World Wide Web. We use data from a prior study of development practices in the Internet Software industry (Appendix B describes how the data for this study was collected; see MacCormack et al, 2001 for details). We focus on the projects that developed versions 3.0 and 4.0 of Explorer, given previous versions of the product were based on code licensed from another firm and involved little effort by Microsoft staff.[7]

Figure 3 shows the productivity of Microsoft's first two projects (in terms of lines of new code developed per person-day) against the sample of comparable projects. We might have expected Microsoft's first efforts developing a browser to be less productive than others, given the challenges that face incumbents in transitioning to new technologies. Similarly, if Microsoft had used its deep pockets to throw resources at this new product, its productivity would be significantly lower than the sample. By contrast, we find that Microsoft achieved higher productivity than the top quartile of the sample. Furthermore, the data also suggests that this higher productivity did not come at the

[6] The Internet was a pre-existing network connecting academics, scientists and government organizations.
[7] This allowed Microsoft to quickly provide a product in a space where it had no offering. The Internet Explorer 3.0 project was the first major internal project to develop a Web browser at Microsoft.

expense of product quality. Both projects resulted in products that were rated as equal to or higher in quality than comparable products.[8]

When one considers that we are focusing on a single company, in an industry characterized by a distributed talent pool and significant investments in R&D by a wide variety of organizations, Microsoft's consistently strong performance is surprising. Furthermore, the data we present suggests that this superior performance is not explained solely by the impact of Microsoft's market power. Microsoft's products typically performed better than the competition, and did so consistently across different product categories at different points in time. Furthermore, its efforts to develop new products in response to one recent technological transition were more productive than many comparable projects conducted in other firms. Given the uncertain nature of the industry and the presence of several major technological transitions over the period examined, the evidence suggests Microsoft possesses what have been called dynamic capabilities. Our work sought to explore the underlying nature of these capabilities in greater detail.

IV. Research Methodology

The objective for our study was to gain a more detailed understanding of the dynamic capabilities through which firms overcome major technological transitions. Specifically, we aimed to both characterize the critical resources that these capabilities build and leverage, as well as the processes through which these resources accumulate and evolve. To achieve these goals, we adopted an inductive research methodology,

[8] A panel of experts evaluated the quality of products in the sample on a 1-7 scale, with a score of 4 being comparable to competitive products. Explorer 3.0 scored 4.0 and Explorer 4.0 scored 5.3.

based upon an exploratory case study at a single firm, using qualitative data collected through an extensive interview program. This approach was appropriate given the descriptive, theory-building nature of our study.

Our work was carried out in a number of unstructured interviews conducted over several visits, which engaged senior engineers, architects, and managers with deep experience of how Microsoft has evolved as an organization over the past 15 years. Our aim in each interview was to probe the reasons for Microsoft's long-term success, and in particular, its ability to successfully manage major technological transitions such as those listed in table 1. While respondents often pointed to a variety of factors as playing a role in Microsoft's success, we focused on explanations linked to the firm's resource base, and the processes through which these resources were created and augmented. For example, we did not consider Microsoft's new product development process a source of dynamic capability *per se*, given it has been admired and studied for over a decade (see Iansiti, 1991; Cusumano and Selby, 1995; MacCormack and Herman, 2000). Instead, we sought to explore the critical resources created *within* this process, and to understand the value of these resources with respect to managing *future* technological transitions.

Given the path dependent process through which dynamic capabilities develop, we first conducted interviews in the company's developer tools and platforms group, one of the oldest groups in the company. This group is often the first to perceive threats from potential technological transitions, given its role in understanding the community of outside developers that leverage Microsoft technologies and assessing the next-generation solutions that these developers require. We then interviewed experienced individuals in key positions of responsibility from other important groups, such as the

Windows and Productivity Applications groups. Finally, we sought examples of how Microsoft managed two recent technological transitions; the first being the development of the Internet Explorer browser in response to the rise of the World Wide Web; the second being the integration of XML into Microsoft's products in response to the rise of distributed architectures that rely upon machine to machine interactions over a network.

As our interview program progressed, we synthesized the insights gained into a concise historical account of Microsoft's development, focusing on the resources that appeared to be central in explaining Microsoft's success. We crosschecked the qualitative data provided by interviewees against each other, to ensure the insights gained were representative of the broader organization's view. We circulated early drafts of our evolving analysis among respondents to gain feedback on whether the account that was emerging made sense to interviewees. Where necessary, we also confirmed facts and figures provided by interviewees using separate (and where possible, public) data sources. In total, our work involved interviews with over 24 different respondents during a 12-month period, with each interview lasting, on average, for 1.5 hours. These efforts built on prior fieldwork at Microsoft performed by the authors at numerous times over the past 12 years (e.g., Iansiti, 1991; Iansiti, 1997; MacCormack and Herman, 2000; 2002).

IV. Research Findings

Developer tools and platforms are the heart and soul of Microsoft. The roots of this influence reach back to the first days of the company. In 1975, Bill Gates and Paul Allen, inspired by an article on the microcomputer in Popular Electronics, decided to write a version of the high-level programming language BASIC for use on the MITS

Altair computer.[9] In developing their BASIC compiler[10] they adopted a vision of computing ubiquity, “a computer on every desktop.” At the heart of this vision was the idea that developer tools, such as BASIC, would be critical for the broad diffusion of micro computing. Microsoft BASIC was widely licensed, hence was available on every major microcomputer of the time (including the TRS 80, Commodore 64, Apple II, MITS Altair, and others). Other programming tools quickly followed: Microsoft added a COBOL compiler in 1976, a Fortran compiler in 1977, a Pascal compiler in 1980 and a 16-bit C compiler in 1983. The impact on the community of application programmers was fundamental. Microcomputers now had tools that allowed programmers to build applications more easily, hence spurring increased growth and adoption.

It was during these early years that the PC architecture as we know it today was born with the adoption of the ISA (Industry Standard Architecture), the design of the PC Bus and the development of the Disc Operating System (DOS). This was a fundamental transition that, for the first time, truly decoupled hardware from software. DOS, developed by Microsoft in response to a request from IBM, became a software “hub” that connected to hardware through a set of “Drivers” and to applications through a set of “Application Programming Interfaces” (APIs). This concept was to change the computing climate, enabling a wide range of different types of hardware and software applications to work together. It spawned the birth of the PC “ecosystem,” comprising component and system vendors, application developers and consultants.

---

[9] “Microsoft Timeline,” available at http://www.microsoft.com/mscorp/museum/musTimeline.asp.
[10] A compiler converts high level programming instructions into a format that the computer can understand.

It was at this time that Microsoft first began to invest in creating libraries of programming "components" – building blocks of intellectual property that could be used in the creation of different software applications. The impetus for creating these components was initially related to the need to provide developers with pre-defined interfaces through which they could access commonly used functions and features. Why re-invent the wheel if someone had worked out what it should look like? Microsoft's tools therefore began to evolve to include not only a programming environment for developers, but also an increasing amount of *pre-defined* functionality that encapsulated best-in-class implementations and algorithms. In essence, Microsoft was codifying knowledge and embedding it in a form that could be leveraged, both by itself and others.

The benefits of software componentization also became apparent in the operating system business, which faced the problem of having many different third-party applications work seamlessly with DOS. To solve this problem, Microsoft began to "expose" a number of software components in the product to third-party developers, resulting in a set of API's for accessing the functionality that DOS provided. "Architecting" the product into components allowed Microsoft to expose only the parts of the software necessary for outside developers to access functionality required by their applications. The firm could therefore differentiate between "boundary" components and "internal" components. This provided a mechanism through which the core intellectual property that the firm had developed could be protected, while leveraging connections to this core in the form of interfaces that could be accessed by outside developers.

The process of software componentization began to become much more widespread and formalized at Microsoft during the late 1980s. At the core of these efforts was the COM ("component object model") architecture or programming model. In essence, COM defined the way that new applications should interact with or "call" pre-existing software components as well as how the components themselves should interact. Significantly, the COM programming model defined an interface through which components could query each other to dynamically discover the other interfaces and components that were available at any point in time. This resulted in a highly flexible architecture, providing the foundation for integrating an unforeseen variety of programming components that would follow thereafter.

The further evolution of Microsoft's component model occurred in the early 1990s, driven to a large extent by work in the applications division, where the increasing penetration of different productivity applications (e.g., Word, Excel, etc.) began creating a need for greater consistency between them. Among these efforts, the Applications Architecture group developed a new framework called "Object Linking and Embedding" (OLE) to better integrate data and programming components across different applications. While OLE was originally developed to help Microsoft's own applications work together more effectively, the framework and its associated software components were soon released to the external development community and embedded in Microsoft's developer tools. With COM providing the overall architecture for development efforts, and OLE defining a set of high-level interfaces between application components, Microsoft's architectural frameworks and associated intellectual property were becoming a powerful combination. Consequently, they began to penetrate the development approaches and associated code libraries of many independent software vendors (ISVs) at an increasingly rapid rate.

To maximize impact across the industry, Microsoft continued to embed its architectural frameworks and associated software components in its development tools. For example, in 1991 Microsoft introduced Visual Basic 1.0, its first development tool to feature an easy to use, graphical environment. Despite the radical appearance of the product, Visual Basic leveraged many of Microsoft's existing software components, layering these connections below a new user-interface. It also introduced many new components to the community of ISVs that used Microsoft's platform. For example, one set of components, known as "custom controls," included a variety of new building blocks, such as support for multimedia files and handwriting recognition.[11] A programmer had to know very little about the topic to use the custom controls – she would simply leverage the code provided in the Microsoft component, the complexity of the content being hidden by the interface provided to her.

As the 1990s progressed, Microsoft's architectural foundations began to evolve to reflect the changing technological context. For example, as networks became increasingly more important in the early 1990s, changes were made to the COM programming framework to facilitate networked relationships between clients and servers in an innovation that became known as the "Distributed Component Object Model" (DCOM). Similarly, during the mid 1990s, OLE was extended to cope with distributed computing environments, becoming widely deployed as a technology known as ActiveX. Simultaneously with these changes, Microsoft made moves to ensure that these innovations were integrated into its tools product line. Visual Basic, Visual C++ and

[11] Support for multimedia and handwriting recognition were released with the Professional Toolkit for Visual Basic 1.0 in March 1992. S. J. Johnson, "Visual Basic Toolkit Ships," InfoWorld, March 16, 2002.

other Microsoft development tools therefore became a vehicle for leveraging the firm’s architectural frameworks and associated software components. As a consequence, these technologies were often adopted and deployed in a rapid fashion by the community of third party firms that developed on Microsoft’s platforms.

By now, it was evident that Microsoft had been building a set of rather unique resources (even if it did not fully recognize this fact early in its development). First of all, its process of software componentization allowed the firm to capture and embed intellectual property in a flexible and easily accessible form. The component libraries resulting from this process represented, in essence, a vast repository of critical knowledge resources that could be leveraged across multiple product lines, and over time. The architectural frameworks like COM and OLE defined how developers, both inside and outside the firm, could take advantage of these components, through the use of well-defined interfaces. And finally, when changes occurred in the external environment, efforts were made to evolve the firm’s component base and architectural frameworks to ensure these remained relevant, as illustrated by the changes to DCOM and ActiveX.

The benefits that came from adopting this software “component model” were apparent on several dimensions. First, with respect to the *internal* organization, Microsoft’s development efforts were more productive than they otherwise would have been, given developers could draw upon a wide variety of pre-existing software components that represented prior investments in intellectual property. Second, with respect to the *external* community of third-party developers, Microsoft benefited from the deployment of additional products that were complementary to its own proprietary solutions. In return, Microsoft provided this community with increased development productivity through tools and platforms that leverage its programming models and associated components. These dynamics made it increasingly difficult for Microsoft's competitors over time. Not only were they competing head-to-head with Microsoft's products – they were also competing against the repository of knowledge resources accumulating in Microsoft's component libraries.

So how did this component model allow Microsoft to respond effectively when faced with major technological transitions? Certainly, the firm had a well-established base of knowledge available to use, in a form that could quickly be brought to bear on emerging opportunities. But it is during periods of technological change when a firm's existing knowledge base is most likely to be found inadequate. In this respect, two processes appeared to be critical. The first was the process of componentization itself, which provides a way for *new* intellectual property to be codified and integrated into Microsoft's knowledge base in a way that ensures compatibility with existing components. The second was the process by which Microsoft evolved its component model to reflect changes in the broader technological context. Below, we illustrate these processes in action, by discussing Microsoft's response to two recent transitions.

Transition 1: *Leveraging* the Component Model to Develop Internet Explorer

Microsoft appeared to be somewhat slow to recognize the potential of the Internet. The early leaders in this space were small but rapidly growing start-ups, such as Netscape and Yahoo!, which were funded specifically to exploit this new space, and attracted a wealth of talented developers as a result. As a consequence of these dynamics, by mid-1995, many analysts thought that Microsoft was yet another incumbent about to stumble when faced with a major technological transition in its core business. Views such as those espoused in *The Economist* were commonplace: "The Internet changes everything. It will turn Microsoft's operating system advantages into so much surplus baggage, slowing it as nimble newcomers sprint ahead."[12] On December 7th 1995 however, Bill Gates announced to the industry, "The sleeping giant has awakened," and proceeded to outline the details of a sweeping campaign to integrate the web into Microsoft's strategy and products.[13] As part of this campaign, a new Internet Platform and Tools division was created with responsibility for developing the new products required to respond effectively to this transition. Central to its efforts was the development of an Internet browser that could compete with that of Netscape, which was considered to be a generation ahead in browser technology.

Microsoft's early efforts at providing a browser had been based on code licensed from a firm called Spyglass. When the firm began developing its next generation browser in November 1995, it faced a critical decision: Should it simply build on top of the code it had licensed from Spyglass, or should the project team re-architect the product from scratch to leverage the firm's existing component model? The decision was not an easy one. Starting from scratch would slow the project down considerably, at a time when most observers suggested speed was of the essence (see Figure 4 for the project timeline). Furthermore, it might involve trade-offs in performance, given competitors would likely optimize the design for only this one specific application. Using a component-based approach, however, would have the advantage of ensuring that the

[12] *The Economist*, May 25th 1996.
[13] This speech is often referred to as the Pearl Harbor speech. See MacCormack and Herman, 2002.

application was compatible with the way Microsoft's other products were built. And perhaps more importantly, it would provide an opportunity to turn Internet Explorer into an integral part of Microsoft's platform strategy. That is, the resulting components would be valuable not only as part of Explorer, but also in providing a foundation for integrating Internet technologies into other products, both those developed by Microsoft and those built by its development partners. Ultimately, proponents of a component-based approach won the day. As one of the project architects recalled:

> *Componentizing Explorer 3.0 was very challenging and added a lot of time to the project. Part of the reasoning [for doing this] was internal – we looked at the code base for Explorer and it was really fragile. Adding a lot of features on top of this would have been really hard. So we made the decision to stop and componentize. … But part of the reasoning was also external: a lot of people wanted [to leverage the technology], and we felt it had the potential to be a new user interface service…So we made a specific investment to look at the browser as part of a framework enabling others to build programs….*[14]

One of the first implications of adopting a component-based approach was that major efforts had to be put into developing a robust architectural framework that defined how the various components of the product should fit together. In this respect, it was particularly important to identify how the new technologies that had emerged during this technological transition (i.e., new communication protocols such as HTML and HTTP) would interact with Microsoft's pre-existing components and frameworks, such as COM and OLE. These architectural decisions were all the more critical given subsequent work to develop the components themselves was performed by more than one development group. While the main Explorer team handled the majority of the work, some components that dealt with what were regarded as "core" functions (e.g., security) were

[14] Interview with Andrew Layman, March 4th, 2002.

developed by separate groups that were not tied to this specific product.[15] This structure allowed Microsoft to leverage domain experts to develop certain core functions, ensuring that the resulting software components were designed to reflect Microsoft's *overall* platform strategy, rather than being optimized for only a single product.

The results of these moves paid off in several ways. First of all, while Internet Explorer became a compelling end-user application, the project also provided the firm with a source of valuable intellectual property that could be leveraged across other Microsoft products that were increasingly affected by the emerging technological transition. Perhaps more significantly however, with a complementary set of development tools, APIs, and a library of reusable components, Explorer became an integral part of a platform that could be leveraged by third-party firms. These firms grew to include both important strategic partners like AOL and Dell, as well as firms who were better known as competitors to Microsoft in other product segments, such as Intuit.[16]

By contrast, Microsoft's main competitor Netscape initially viewed the browser as a stand-alone monolithic product, limiting the potential impact from its innovation.[17] While over time, it began to realize the power of adopting a component-based approach, by this time it was well behind Microsoft on several fronts: First, in understanding how to re-architect the browser into a number of more modular components; second, in understanding the implications of a component-based approach for how the development

[15] A program manager on Explorer 3.0 and 4.0 indicated that between 25-35% of the new code developed in these two projects was developed in groups outside the main development team.
[16] All these firms subsequently built customized browsers using components of Internet Explorer.
[17] Cusumano and Yoffie (1998) describe Netscape's struggles with regard to this issue. They note that "...a serious weakness in Netscape's product set in 1997 and 1998 was that client [i.e., browser] feature teams shared a lot of code that developers had not sufficiently modularized. The resulting spaghetti code made it nearly impossible to build and test components in independent teams, or release the components as discrete chunks of functionality." Cusumano and Yoffie (1998), p202.

organization should be structured; and third, in assembling a set of complementary components and architectural frameworks (e.g., such as COM) that would work seamlessly with the new components, once developed.

Transition 2: *Evolving* the Component Model to make use of XML[18]

One of the greatest problems in building information technology systems is the integration of data and programs that use different formats and languages and operate on a variety of different, distributed, computers. The Extensible Markup Language (XML) promises to remedy many of these integration challenges, providing a standard way to represent information that will allow distributed computers to communicate with each other programmatically (i.e., without a user being required).

Microsoft's work on XML began in June 1996, when Adam Bosworth (an architect who had worked on both Windows and Internet Explorer) approached Jean Paoli looking for a new approach to the presentation of data. At that time, Paoli and another Microsoft engineer were members of a World Wide Web Consortium (W3C) working group that was helping to write the first specification for XML (eventually published in draft form in November, 1996).[19] It was during these discussions that Bosworth, Paoli and Andrew Layman, an architect who had worked on the componentization of Internet Explorer, came to a critical realization. XML's true potential was not in document management and data presentation – the original vision of

[18] Details of how Microsoft moved to integrate the use of XML across its product lines can be found in MacCormack and Herman (2002).
[19] The first draft of the XML specification was co-authored by Tim Bray (from Textuality) and C. M. Sperberg (from the University of Illinois).

the W3C group – but lay in the field of machine-to-machine interaction. Specifically, XML could provide the framework for connecting a diverse set of computing platforms. In doing so, it could link Microsoft's existing programming model and components to an even greater set of systems, applications, and services. Layman recalled:

> *Our contribution was not in creating XML, but in seeing its potential at an early stage. It was about connecting computers… XML manages the facts necessary for different systems to talk to each other. But it is designed to include only the minimum number of necessary facts. So XML is the thinnest possible thing to standardize on that will ensure interoperability across a diverse, decentralized, set of computers.*[20]

At the same time as Bosworth, Paoli and Layman were coming to this realization, several individuals at Microsoft had began to sense that a major technological transition was underway in the broader computing environment. Nowhere was this clearer than in the platform and tools group, where it was evident that Microsoft's platform dominance was under threat. Developers were beginning to choose a new platform – the Internet – on which to write their applications. This meant that code was increasingly being written to run on the server, and not on the client where Windows was positioned. Vic Gundotra, who ran platform marketing at the time, explained:

> *The applications that were exciting in 1995, 1996, 1997, were Amazon, eBay, Yahoo! – applications that had nothing to do with our platform. They didn't write those on Windows. They wrote those on a new platform based on TCP/IP, HTTP, HTML… The operating system battle had been lost by then. The operating system – while a piece of the puzzle – is not the overall platform. You need a platform that manages the servers, the communication between the servers and the client, between client and clients – a new kind of platform that developers will target… We started brainstorming about what kind of platform would we need, and we came up with a few principles. Those principles wrapped around interoperability*

[20] Interview with Microsoft employee Andrew Layman, March 4th, 2002

between all systems, they wrapped around open standards that were as simple as the Internet protocols .[21]

Gundotra and one of his colleagues, Charles Fitzgerald began "evangelizing" the need for Microsoft to *evolve* their platform vision, lobbying senior executives at every opportunity. One of these executives, Paul Maritz, the VP of platform strategy and the developers group, was particularly receptive to their message. He provided valuable "air cover," helping Gundotra and Fitzgerald network with other people inside the firm that were developing technologies and frameworks that could help in their efforts.

One of these people was Yuval Neeman, the VP of Developer Tools. Neeman too, had seen the trend of developers targeting the Internet for new applications. He also realized that the Internet was as yet a very immature platform with poor development tools, and therein was an opportunity. But evolving Microsoft's existing components and frameworks to support a world where distributed systems and applications were linked over the Internet posed many challenges. First of all, the Internet worked in a different way to the way software was developed at companies like Microsoft. Neeman explained:

In the early 1990's, people thought – in Microsoft and other places – the way you build things with code is you have 'objects' talking with other 'objects' in what we call 'tightly-coupled applications'… if I change something on my side, you have to change something on yours…What the web gave you, in contrast, was a more loosely-coupled model…When you use a browser, the web sites you talk to change constantly, but you don't have to change at all…this was a radical difference. Our developer tools however, did not reflect this difference.[22]

[21] Vic Gundotra, as quoted in Microsoft.NET, MacCormack and Herman, 2002.
[22] Yuval Neeman, as quoted in Microsoft.NET, MacCormack and Herman, 2002.

The other major hurdle in moving to a platform based on the Internet was that it was to all intents and purposes “dumb” – you could only interface with it through a user interaction, as opposed to doing it programmatically. Neeman explained:

> *One of the things we thought was very important…was to move away from just a dumb client talking to a server, and instead have a way for a client to talk to a server to make a method call, to invoke code in the server programmatically from the client. And we decided that this has to be a loosely-coupled environment for this to work… We were struggling with how to encode it though… Our teams began talking with Adam [Bosworth] and caught on to his ideas about XML.*[23]

By 1988, Bosworth and several other experienced architects had made substantial progress in their experiments with XML. They had defined how the technology should operate, and how Microsoft could take advantage of XML by extending its existing component model and architectural frameworks. These efforts, on top of the work by Gundotra, Fitzgerald and Neeman all pointed in the same direction – towards extending the reach of Microsoft’s platform so that *any* machine or device, anywhere in the world, could connect together. And XML would provide the mechanism for this to happen.

Thereafter, XML quickly began to have an impact on multiple groups at Microsoft, significantly influencing the evolution of the firm’s component model. For example, to solve the “dumb” Internet problem, Neeman and Bosworth focused resources on developing a way of using XML to express Remote Procedure Calls. These calls would allow applications to invoke other applications over the Internet. With such a feature, applications running on a user’s desktop would be able to call a myriad of different “web

[23] Ibid…

services" over the Internet, and have these services run seamlessly as if they were an integral part of the application.[24]

Other groups needed to work out how to re-architect Microsoft's existing components and frameworks to operate in the newly emerging context. This proved particularly challenging in the tools group, where in early 1997, a project had begun to develop a common, multi-language, integrated development environment.[25] As the various development teams thought through how to incorporate the use of XML and the necessary "plumbing" to deliver web services however, it became clear just how extensive the changes would need to be. Neeman explained:

> *Visual Basic [one of the tools products] was a tightly-coupled environment, and building it out to accommodate the loosely-coupled environment of the web was a much more painful project than we'd anticipated… A key question began to emerge: Could we break compatibility? This was a highly alarming prospect*[26]*.*

Breaking compatibility was regarded as a cardinal sin at Microsoft. Doing so would mean that developers who had used older versions of Visual Basic to develop applications would not be able to recompile their source code and port it to the newer version. Eventually however, Neeman had to make the decision to do it. The changes required by the use of XML and the move to a loosely-coupled programming model were too extensive. But critically, the logic behind this decision and the others like it were informed by a clear sense of direction. Specifically, how could Microsoft take maximum advantage of the technological transition that was underway?

[24] For example, a language translation button in an email application might call a "Web service" over the Internet to provide the translation service without the user having any knowledge of this interaction.
[25] The idea was to integrate Microsoft's various development tool products – for example, Visual Basic and Visual C++ – into a single suite, as had been done with Office and its component applications years earlier.
[26] Neeman as quoted in MacCormack and Herman, 2002.

By the fall of 1999, the efforts of Gundotra, Fitzgerald, Neeman, Bosworth and others had been integrated to create the vision for a comprehensive programming framework and component library that would allow developers to build and deploy XML-based web services – an effort that became known as "Microsoft.NET." While the initiative to this point had mainly involved Microsoft's platform and tools group, its execution would depend upon changes being made to many of Microsoft's other product lines. Existing programming components embedded in products like Office and Windows would have to be made available as services that could be called remotely using XML. This in turn required a variety of other technologies and standards be developed.[27] While many challenges associated with delivering on the promise of XML lay ahead of the firm, the critical work had been done. Specifically, Microsoft had established a framework for how these new technologies should be integrated into its component model, and the changes that would have to be made to this model as a result.

To conclude, our interviews suggest that the initiative to develop and deploy XML across Microsoft's product-line did not result from a clear top-down strategic direction, but was the result of experienced middle managers identifying the need for change, developing a vision for how the emerging technological transition should impact the firm's existing component model and "selling" this plan to senior management. In this respect, the role of the platform and tools group was central, given this group leads the organization in terms of sensing changing patterns of use by developers. Furthermore, the ongoing work to navigate the transition was carried out by a number of the firm's most experienced architects who had the skills to understand both how the new

[27] E.g., the Simple Object Access Protocol (SOAP) and the Web Service Description Language (WSDL).

technologies implied by the use of XML could be integrated into Microsoft's existing component model, and the changes that would have to be made to this model as a result.

V. Discussion

Our findings have significant implications for the debate on the management of technological evolution. Many papers have underlined the difficulties faced by incumbent firms in responding to major technological and market changes (Tushman and Anderson, 1986; Henderson and Clark, 1990; Christensen and Bower, 1997). This stream of work can sometimes leave the impression that such firms face insurmountable challenges in dynamic environments. Other authors however, offer hope to incumbents by arguing that organizations can survive and indeed prosper from technological transitions providing they possess what have been called dynamic capabilities (Teece, Pisano, and Shuen, 1997; Iansiti and Clark, 1994; Tushman and O'Reilly, 1997; Eisenhardt and Martin, 2000). Our research lends support to this latter perspective.

In particular, our work sheds light on the nature of the critical resources that dynamic capabilities leverage (Wernerfelt, 1984; Barney, 1991; Eisenhardt and Martin, 2000). We describe how Microsoft's process of software componentization has allowed the firm to develop and codify its intellectual property, making it available for use across multiple products in its current line, as well as providing a foundation upon which future products can be built. As an output of this process, the organization has built a critical base of resources in the form of a vast array of component "libraries." Our observations suggest that this set of libraries may be one of the most critical resources in explaining the consistency in performance of Microsoft’s products and overall business over time.

Microsoft’s component libraries can be stored at length, augmented and recombined at will, unlike tacit or experiential bases of knowledge, which are typically tied to individuals and whose interactions are difficult to manage (e.g., Leonard-Barton, 1995; Von Hippel, 1990). They therefore provide a more flexible and enduring source of advantage than other sources of knowledge. Furthermore, we have observed that over time, older components have been combined with new components, ported to new hardware platforms, included in new products, applied to new business models, and leveraged by different parts of the organization. They have therefore had a critical *internal* impact in terms of making Microsoft’s development efforts more productive than they otherwise would be in their absence. But our work also highlights the *external* impact of Microsoft’s component model, which takes effect in two ways: First, in allowing outside developers to leverage components provided as part of Microsoft’s development tools; and second, by allowing outside developers to access a number of the software components embedded in Microsoft’s products, through the use of clearly-defined interfaces or APIs. The result is a platform for development that reaches far beyond Microsoft’s organizational boundaries; Microsoft estimates the number of developers that belong to the Microsoft Development Network (“MSDN”) at more than 5 million, and the number of companies that leverage its tools at over 10,000. This is a major source of competitive advantage. It is not enough for a competitor to offer a more attractive product. To be a viable threat, they must offer a similarly comprehensive component model, with tools and components that offer substantial advantages over Microsoft's already widely deployed technologies.

We should note that the argument we make regarding the power of Microsoft's component model does not hinge on whether the algorithms embedded in these components, at any point in time, represent the "best" solutions to the functionality that each provides. The critical resource that Microsoft possesses does not rely on superior components per se, but rather is founded upon having a broad set of components that operate seamlessly within a coherent overall architecture. To clarify, it is entirely possible that a group of software experts, given enough time and money, would be able to come up with better individual components or architectural frameworks in terms of pure *technical* performance. The problem is they would be many years behind Microsoft in terms of populating their particular component model with useful intellectual property. In this respect, our findings support the arguments of Teece, Pisano and Shuen (1997) who assert the importance of path dependence is magnified in the presence of increasing returns to adoption (e.g., network externalities). Whether by chance or by foresight, Microsoft's early decision to invest in a component model and distribute parts of the IP in this model to third-party firms has, many years later, resulted in a set of resources that provide a critical source of advantage when facing major technological transitions.

We believe our work sheds new light on the critical role of architectural design choices in shaping the resources developed within a firm. Specifically, Microsoft's architectural frameworks (e.g., COM and OLE) appear to be as important a resource as the components themselves, given these frameworks specify the way these components interact, as well as the way in which new intellectual property should be codified so that it is compatible with them. These frameworks provide a structure within which the *process* of componentization can occur. In this respect, they provide a unified way with which to coordinate the firm's responses to technological transitions, which are necessarily broadly distributed among its community of managers and engineers.

These arguments suggest an important extension to the growing literature on the topic of product modularity (Langlois and Robertson, 1992; Sanderson and Uzumeri, 1995; Ulrich, 1995; Sanchez and Mahoney, 1996; Schilling, 2000; Baldwin and Clark, 2000). In particular, we show that the conscious choices a firm makes to "modularize" a product may form the basis for developing critical resources that can be applied across a product line and endure for multiple product generations. More specifically, our work illustrates the power that results from ensuring these architectural decisions are coordinated across a firm's entire product line (both present and future) as opposed to being considered on a product-by-product basis. This points to the critical role played by a firm's most experienced "architects," in terms of the need to develop a framework that ties together the many individual decisions that are made with respect to the day-to-day evolution of a firm's product line (Henderson and Clark, 1990; Baldwin and Clark, 2000). Our work suggests that these architectural choices have a major impact on the likelihood of success in negotiating technological transitions.

Our findings have major implications for how firms should structure their new product development activities. While most research on this topic has focused on the design of effective processes and team structures for managing development (e.g., Wheelwright and Clark, 1992; Iansiti, 1997) our work suggests that development projects fulfill multiple objectives in firms that possess dynamic capabilities. Specifically, each project represents not only an effort to stake out a near-term product market position, but also an effort to develop intellectual property that will last for considerably longer and apply to a broader set of products (Helfat and Raubitschek, 2000). It is the intellectual property, not the product it first appears in, that has the more enduring value. This, in turn, highlights the critical importance of the specific processes that create this IP. In particular, it is Microsoft's componentization process, not its overall product development process, which provides the key to its dynamic capability.

Our work suggests that determining the optimal structure for managing development is a complex problem in a firm that possesses dynamic capabilities. In some circumstances, the development of critical resources may best be performed by a separate team, given this allows a greater focus on how these resources should be designed for leverage across a firm's entire product line (both present and future). This choice however, implies a trade-off, in that the design of these resources is likely to be less "optimal" with respect to the initial products being developed (as compared to the designs that would result if these resources were tightly integrated into each product). Altrenatively, it might be beneficial for a project to consume additional resources with the purpose of ensuring that the IP developed is componentized in a manner that is compatible with a firm's existing architectural frameworks. Significantly, either choice implies a negative impact on some dimensions of *project-level* performance (e.g., in terms of team productivity or product performance). That is, *investing in the development of critical resources that underpin dynamic capabilities will sometime require trade-offs to be made with respect to the performance of individual projects*. This insight highlights the potential problems in adopting the project as unit of analysis when trying to understand the development of dynamic capabilities. Our results suggest that this endeavor requires a broader perspective that is inherently longitudinal in nature.

Our arguments to this point have focused on the critical resources that Microsoft possesses, as embodied in its component libraries and architectural frameworks. For these resources to be a source of *dynamic* capability however, we must identify the processes through which these resources are evolved, given the potential for technological transitions to render them obsolete. In this respect, our work highlights Microsoft's ability to both recognize emerging new technologies and thereafter to assess how these technologies should impact its existing resource base. With respect to the former, we observe the important role played by groups that lead the broader organization in terms of "sensing" changes in the external context. For example, the developer tools group at Microsoft helps the firm identify threats from newly emerging platforms before these platforms have matured. And the firm's involvement with standard setting bodies plays a critical role in helping understand the impact of new technologies at an early stage, even influencing their development in a way that plays to Microsoft strengths. Importantly however, our work suggests that these sensing mechanisms are not top-down strategic processes, but are "emergent" in nature (see Burgelman, 1994; Mintzberg, 1987) being triggered by the tensions generated when trends in the external environment threaten the firm's existing resource base. Hence the process of resolving such tensions is best conducted by a firm's most experienced architects, given these individuals have both a deep knowledge of the firm's resources as well as the skills to understand how changes in the external context can be integrated into and shape the evolution of these resources.

VI. Conclusion

Our work has explored the critical resources that dynamic capabilities leverage. In particular, we have described how Microsoft's process of software componentization allows it to capture and embed intellectual property in a flexible and easily accessible form. The component "libraries" that result from this process and the architectural frameworks that specify how these components interact represent critical resources that can provide a source of competitive advantage in dynamic environments. The benefits from these resources manifest themselves internally, in terms of enhanced development productivity, and externally, through their role in attracting third-party developers to build upon Microsoft's platforms. Our findings contribute to the literature on the resource-based view of the firm, and in particular, help to build a more operational understanding of how certain types of resources can provide a source of dynamic capability (Wernerfelt, 1984; Barney, 1991; Eisenhardt and Martin, 2000).

Our study focused on a single firm and adopted an inductive research methodology based upon qualitative data collected via interviews. While this was appropriate given the nature of our investigation, we note that with such a research design, we cannot rule out competing explanations for Microsoft's sustained high performance in the software industry. We also have no data on whether other firms in the software industry have adopted software component models similar to Microsoft's, or on how such firms have performed. In this respect, we have yet to identify whether some aspects of this model are more critical than others. This is a topic for ongoing study.

Our findings suggest a number of avenues for future work. In particular, we have shown that the processes through which critical resources accumulate as well as the actual resources themselves can often be identified and measured. The study of software component libraries, for example, is amenable to quantitative analysis, given these libraries can be tracked over time, and the process of creation, evolution and recombination examined in detail. The same is true of other contexts where intellectual property can be codified and stored in libraries, as is the case in the biotechnology and semiconductor industries. We believe that these types of analyses are likely to provide complementary insights to more traditional research approaches for studying the role of intellectual property, such as those that focus on the analysis of patent counts. Indeed, such research should become increasingly possible in many industries, as the proportion of information embedded in computer-aided design and simulation systems rises. With this agenda, we feel there is a significant opportunity to deepen our understanding of the roots of dynamic capability, and the critical resources that these capabilities leverage.

Bibliography

Baldwin, C. Y., and K. B. Clark. (2000). *Design Rules: The Power of Modularity*. Cambridge, MA: MIT Press.

Barney, J.B. Firm Resources and Sustained Competitive Advantage. *Journal of Management* 17(1): 99-120.

Brown, S. L. and K. M. Eisenhardt (1997). The Art of Continuous Change. *Administrative Science Quarterly 42(1):* 1 - 34.

Burgelman, R.A. (1994). Fading Memories: A Process Theory of Strategic Business Exit in Dynamic Environments. *Administrative Science Quarterly,* March Issue: 25-55.

Christensen, C.M. (1997). *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. Boston: Harvard Business School Press.

Christensen, C. (2001). Skate to Where the Money Will Be. *Harvard Business Review* (November).

Christensen, C.M. and J.L. Bower (1996). Customer Power, Strategic Investment and the Failure of Leading Firms. *Strategic Management Review, 17, 197-218*.

Clark, K. B., and T. Fujimoto (1991). *Product Development Performance*. Boston: Harvard Business School Press.

Cusumano, M., and R. Selby (1995). *Microsoft Secrets.* New York: Free Press, New York.

Cusumano, M. and D. Yoffie (1998). *Competing on Internet Time*. Free Press, New York.

Dalkey, N. and O. Helmer (1963). An Experimental Application of the Delphi Method to the Use of Experts. *Management Science, 9(3),* 458-67.

Eisenhardt, K. M. and J. A. Martin (2000). Dynamic Capabilities: What are they? *Strategic Management Journal, 21*, 1105-1121.

Garvin, D (1987). Competing on the Eight Dimensions of Quality. *Harvard Business Review* (Nov-Dec).

Gawer, A. and M. Cusumano (2002). *Platform Leadership: How Intel, Microsoft, and Cisco Drive Industry Innovation*. Boston: Harvard Business School Press.

Gompers, P. and J. Lerner (2001). *The Money of Invention: How Venture Capital Creates New Wealth*. Boston: Harvard Business School Press.

Gompers, P. and J. Lerner (1999). *The Venture Capital Cycle*. Cambridge: MIT Press.

Griliches, Zvi (1990). Patent Statistics as Economic Indicators: A Survey. *Journal of Economic Literature, 28:* 1661-1707.

Hamel, G. and C. K. Prahalad (1994). *Competing for the Future: Breakthrough Strategies for Seizing Control of Your Industry and Creating the Markets of Tomorrow*. Boston: Harvard Business School Press.

Hayes, R., S. Wheelwright, and K. Clark (1988). *Dynamic Manufacturing: Creating the Learning Organization*. New York: The Free Press.

Helfat, C.E. and R.S. Raubitschek (2000). Product Sequencing: Co-Evolution of Knowledge, Capabilities, and Products. *Strategic Management Journal, 21*: 961-989.

Henderson, R., and Clark, K.B. (1990). Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms. *Administrative Sciences Quarterly, 35(1)*: 9-30.

Henderson, R.H. and Cockburn, I.C (1994). Measuring Core Competence? Evidence from the Pharmaceutical Industry. *MIT Sloan School of Management Working Paper*.

Henderson, R. and Cockburn, I (1992). Scale, Scope and Spillovers. *MIT Sloan School of Management Working Paper*.

Iansiti, M. (1991). Microsoft Office Business Unit. *Harvard Business School Case 691-033*.

Iansiti, M. 1997. *Technology Integration: Making Critical Choices in a Dynamic World*. HBS Press, Boston, MA.

Iansiti, M (1997). From Technological Potential to Product Performance: An Empirical Analysis. *Research Policy, 26(3)*: 345-365.

Iansiti, M., and K. B. Clark (1994). Integration and Dynamic Capability: Evidence from Development in Automobiles and Mainframe Computers. *Industrial and Corporate Change,3(3):* 557-605.

Iansiti, M. and Levien (2002). The New Operational Dynamics of Business Ecosystems: Implications for Policy, Operations and Technology Strategy. *Harvard Business School Working Paper*, 03-030.

Iansiti, M. and A. MacCormack (1997). Developing Products on Internet Time. *Harvard Business Review, (Sep-Oct).*

Jaffe, Adam B. and Manuel Trajtenberg (2002). *Patents, Citations and Innovations: A Window on the Knowledge Economy*. Cambridge: MIT Press.

Langlois, R.N. and P.L. Robertson. (1992). Networks and Innovation in a Modular System: Lessons from the Microcomputer and Stereo Component Industries. *Research Policy,* 21: 132-157.

Leonard-Barton, D (1992). Core Capabilities and Core Rigidities: A Paradox in Managing Product Development. *Strategic Management Journal, 13*: 111-125.

Leonard-Baron, D (1995). *Wellsprings of Knowledge: Building and Sustaining the Sources of Innovation*. Boston: Harvard Business School Press.

Levinthal, D. and J.G. March (1981). A Model of Adaptive Organizational Search. *Journal of Economic Behavior & Organization, 2.*

MacCormack, A. D (2001). Product-Development Practices That Work: How Internet Companies Build Software. *Sloan Management Review 42(2):* 75-84.

MacCormack, A., and K. Herman (2000). Microsoft Office 2000: Multimedia. *Harvard Business School Case 600-023*.

MacCormack, A. and K. Herman (2002). Microsoft.NET. *Harvard Business School Case 602-086*.

MacCormack, A. D., R. Verganti, and M. Iansiti (2001). Developing Products on Internet Time: The Anatomy of a Flexible Development Process. *Management Science, 47(1).*

Mintzberg, H. (1987). Crafting Strategy. *Harvard Business Review (Jul/Aug)*

Nelson, R., and S. Winter (1982). *An Evolutionary Theory of Economic Change*. Cambridge, MA: Harvard University Press.

Penrose, E. T (1959). *The Theory and Growth of the Firm*. New York: Wiley.

Pisano, G. P (1996). *The Development Factory: Unlocking the Potential of Process Innovation*. Boston: Harvard Business School Press.

Schilling, M.A. (2000). Toward a General Modular Systems Theory and its Application to Interfirm Product Modularity. *Academy of Management Review*. Vol.25 No.2 312-334.

Sanchez, R., J.T. Mahoney. (1996). Modularity, Flexibility, and Knowledge Management in Product and Organization Design. *Strategic Management Journal* 17: 63-76.

Sanderson, S. and Uzumeri, M. (1995). Managing Product Families: The Case of the Sony Walkman. *Research Policy*, Vol 24, No. 5, 761-782.

Shapiro, C. and H. Varian (1999). *Information Rules: A Strategic Guide to the Network Economy*. Boston: Harvard Business School Press.

Teece, D. G.P. Pisano and A. Shuen (1997). Dynamic Capabilities and Strategic Management. *Strategic Management Journal, 18(7)*: 509-533.

Teece, D. J (2000). *Managing Intellectual Capital: Organizing, Strategic, and Policy Dimensions.* Oxford: Oxford University Press.

Tripsas, M (1997). Unraveling the Process of Creative Destruction: Complementary assets and incumbent survival in the typesetter industry. *Strategic Management Journal*, *18*.

Tushman, M.L. and P. Anderson (1986). Technological Discontinuities and Organizational Environments. *Administrative Science Quarterly*, *31(3)*: 439-465.

Tushman, M. L. and C.A. O'Reilly III (1997). *Winning Through Innovation*. Boston: Harvard Business School Press.

Ulrich, K. T. (1995). The role of Product Architecture in the Manufacturing Firm. *Research Policy* 24: 419-440.

Von Hippel, E. (1990). The Impact of 'Sticky Data' on Innovation and Problem Solving. *MIT Sloan School of Management Working Paper* #3147-90-BPS.

Wernerfelt, B. (1984). A Resource-Based View of the Firm. *Strategic Management Journal*, *5*: 171-180.

West, J. and Iansiti, M (1997). Experience, Experimentation, and the Accumulation of Organizational Capabilities: An Empirical Study of R&D in the Semiconductor Industry. *Harvard Business School Working Paper*, 96-032.

Westerman, G., M. Iansiti, and F. W. McFarlan (2002). The Tortoise and the Hare: An Empirical Investigation of the Dynamics of Integration, Differentiation and Incumbent Adaptation. *Harvard Business School Working Paper*, 03-002.

Wheelwright, S. C., K. B. Clark. 1992. *Revolutionizing Product Development*, Free Press, NY.

Table 1: Selected Technological Transitions in the Software Industry

Transition	**Timing**	**Description**
Personal Computer platform	1981 on	Evolution of a personal computing platform with application programming interfaces ("APIs") that effectively buffer hardware from software .
Graphical Computing	1983-1992	Intuitive, graphical user interfaces, opening up computer usage to less sophisticated users. First introduced in the Apple computer, later adopted in the design of PC operating systems
Object Oriented Programming	1987 - 1992	New programming model based on programming components or objects, which can be reused in different applications. Associated with a change in programming language from C to C++
World Wide Web	1993-1998	The rapid adoption of weak coupling standards such as HTTP and HTML, as well as tools for accessing remote information, such as the Web browser
Multi-tiered architectures	1995 - 1998	Adoption of multi-tiered platforms, including Web and application servers to deploy distributed applications
XML Integration	1998 - future	Machine to machine integration achieve using Extensible Markup Language (XML)
Web Services	2000- future	Infrastructure that enables the rapid deployment and interconnection of a broad base of computing services available for access over the Web

Source: Interviews with 8 industry experts from a number of leading software firms.

Table 2: Number of "Wins" by Microsoft Products.

Microsoft Product	Reviewed Time Period	Total Number of Reviews	Total Number of "Wins" by Microsoft Product	Percentage of "Wins" by Microsoft Product	Number of Shared "Wins"*
Access	1992-1998	14	10	71%	2
Excel	1988-1998	25	21	84%	5
Money	1992-2001	24	5	21%	1
PowerPoint	1990-1998	22	8	36%	2
Word	1986-1998	30	27	90%	10
Office	1994-2001	14	14	100%	0
Internet Explorer	1995-2001	16	6	37.5%	0
Visual Basic/Visual Studio	1992-2002	10	8	80%	0
Windows server	1993-2000	12	9	75%	2
Windows	1987-2001	36	19	53%	1

* Reviews when a Microsoft software product shared "win" with another software product.

Figure 1: Microsoft Financial Performance

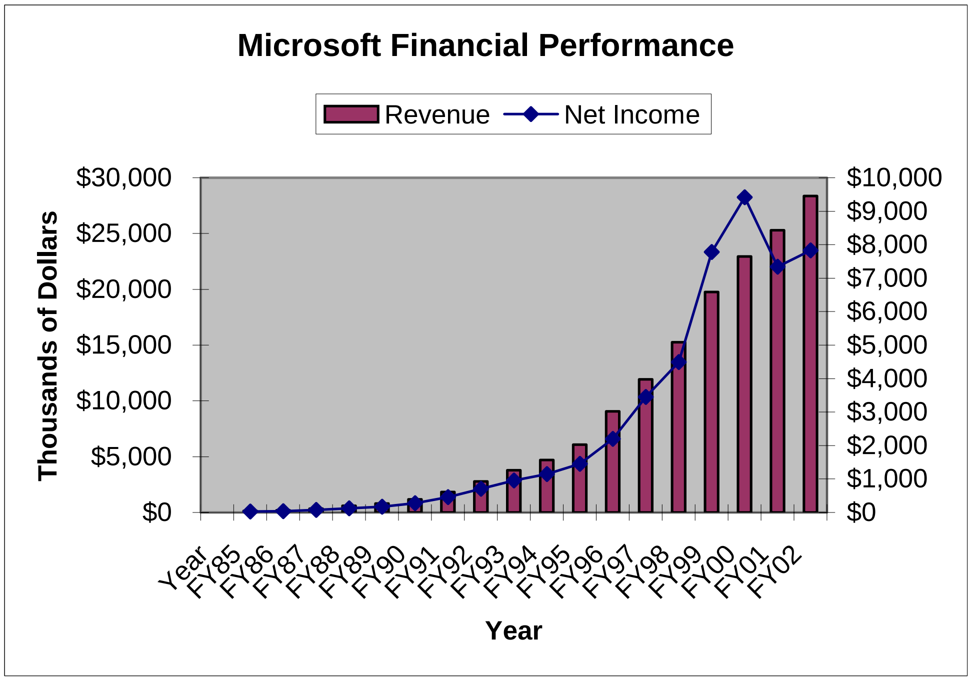
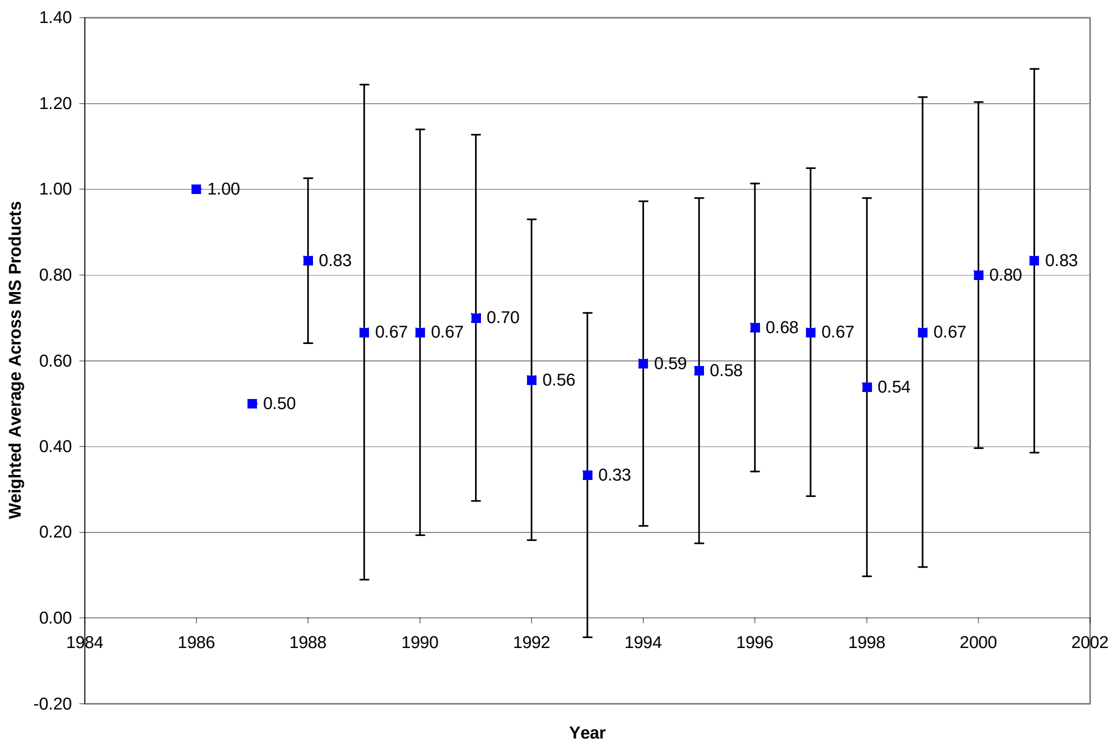


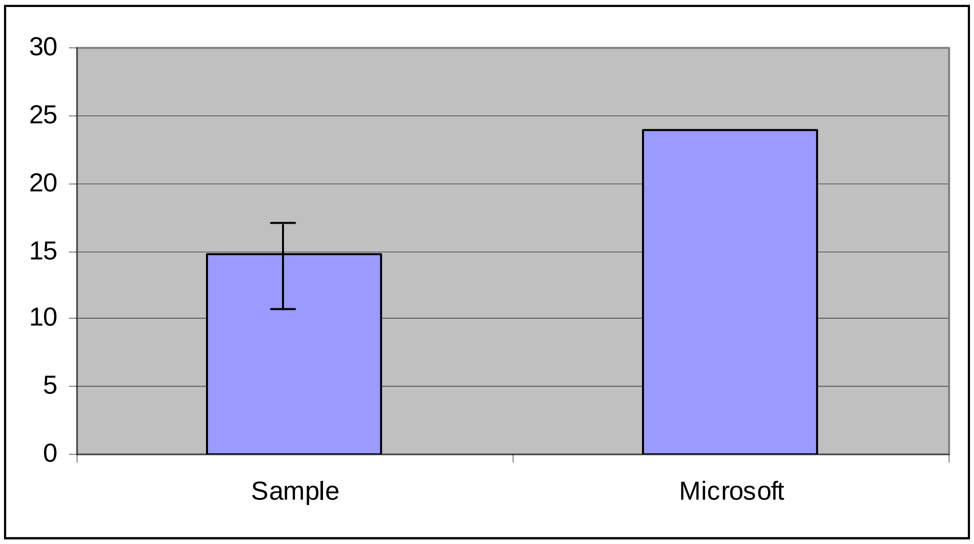
Figure 2: Plot of Microsoft Product Wins over Time



Note: The error bars note the standard deviation in the average win percentage, and are displayed only if the number of observations is greater than 2.

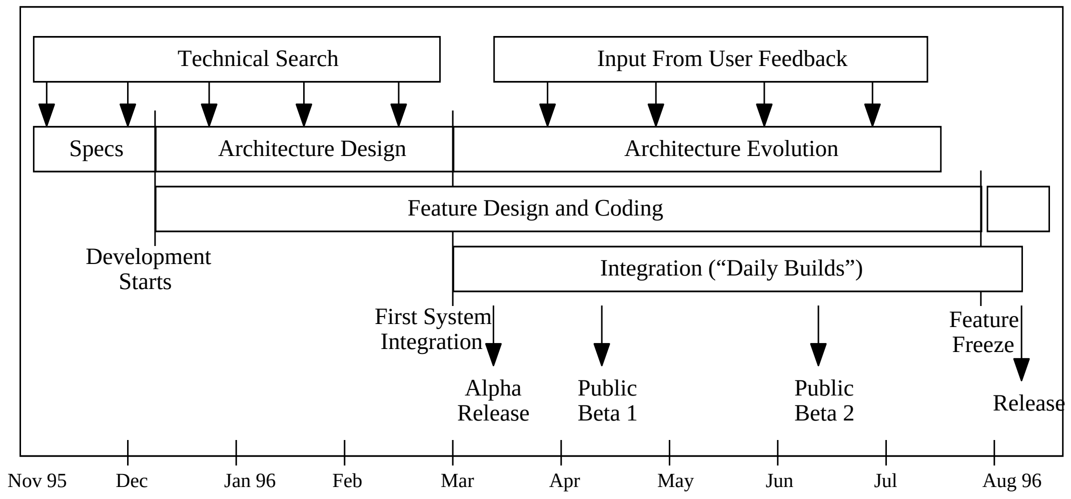
Figure 3: Comparison of Development Productivity

This figure compares the average productivity in new lines of code per person-day of two Microsoft projects (Internet Explorer 3.0 and Internet Explorer 4.0) versus the average for a sample of Internet software projects completed at the same time. The sample mean is 14.8, and the top quartile is 17.5 (shown on the chart). Microsoft achieved 23.9.



Source: Sample of 29 Internet software projects (MaCormack et al, 2001).

Figure 4: Timeline for the Development of Microsoft Explorer 3.0.



Source: MacCormack, 2001.

Appendix A: Methodology for Assembling Database of Product Reviews

To build the database of reviews, we selected five leading computer journals based on paid circulation in 2000 – PC Magazine, PC World, Computer Shopper, InfoWorld, and Computerworld – and used *Computer Select* CDs and the *Nexis* database to locate journal issues for the period starting from the first release of a given Microsoft product until the present time. Given our aim of looking at performance over time, we excluded products for which only a few reviews could be found. Our final sample therefore includes reviews for six major types of end-user software (Access, Excel, Internet Explorer, Money, PowerPoint, Word), one suite of applications (Microsoft Office), Microsoft's server and client operating systems and Microsoft's developer tool suite (Visual Basic, which has now become part of a broader tool called Visual Studio). Note that since some of the journals mentioned above do not review servers and developer tools we selected a slightly different set of magazines for these products. We looked at developer tools' reviews in PC Magazine, PC World, InfoWorld and Dr. Dobb's Journal, and at server operating system reviews in PC Magazine, PC Week, Network Computing, Network World, InfoWorld, Internet Week and ENT.

We measured performance in two ways. First, we calculated product "wins", where a "win" is defined as the situation when a review identifies a product as the best without assigning any numeric scores. "Win" categories included: "Editor's Choice," "Best of the Year," "Best Buy," "Award of Technical Excellence," and from server reviews, categories such as "PC Week Labs Analyst's Choice Award" and "Network World Blue Ribbon Award." Second, we obtained a more exact measure of a product's quality by reporting the exact ratings given by reviewers. When a product was rated by specific characteristics, we calculated an overall quality rating by taking the average value of these characteristics. When a product was rated by qualitative categories such as "good" we converted ratings into numeric values, normalized to a scale of 0-10[28] [29].

While collecting data on reviews we encountered several problems. First, we used electronic versions of journals in which for some cases, tables were omitted preventing us from reporting numeric scores for some product versions. Second, since the *Computer Select* CDs covered the period from 1991 until 2001, we had to use the *Nexis* database for the period before 1991. Unfortunately, *Nexis* does not include PC World and Computer Shopper issues before 1991. Also, issues of ENT and Internet Week were not available before 1997, and issues of Network Computing were not available before 1994. Finally, most reviews stopped reviewing individual applications and development tools separately around 1997, and began reviewing them as a part of the suites in which they were included. To compensate for this omission we looked at reviews for Microsoft Office from 1995 until 2001 and for Visual Studio from 1997 until 2002.

[28] The following schemes were used: "excellent" = 10, "good" = 7.5, "fair"= 5, and "poor" = 2.5; or "excellent" = 10, "very good" = 7.5, "good" = 6.25, "fair" = 5, and "poor" = 2.5.

[29] In some cases there were several "winning" software products or several software products with the same highest score. In such cases we reported all "wins" and the highest numeric scores earned by Microsoft software products and indicated how many wins were shared with other software products.

Appendix B: Methodology for Capturing Data on Development Performance.

Our analysis of Microsoft's performance developing its Internet Explorer browser is based on data captured during a two-year study of product development practices in the internet software industry (MacCormack et al, 2001).[30] Data were collected through the use of a survey instrument distributed to project managers at a sample of firms developing software products for the Internet identified through reviewing industry journals. In most cases, responding firms were contacted to collect additional descriptive data on projects. The final sample consisted of 29 completed projects from 17 firms, covering a broad range of internet applications, including products, services and development tools targeted at both commercial and consumer users[31].

In order to compare the performance of projects, two outcome measures were captured. The first measured the productivity of the development team, in terms of lines of new code developed per person-day of development effort (defined as the effort involved in developing and testing the code base). This measure was adjusted to reflect the fact that different types of products used different programming languages[32]. Note the resulting metric is only an approximate measure of software productivity (i.e., "good" programmers often implement a given set of functionality with fewer lines of code). It is however, a measure that can be captured in a consistent manner across projects.

[30] Note that the data was collected for the purposes of examining effective development process design. Participating firms were assured that their data would be kept confidential unless they provided permission for us to identify specific observations. This permission has been obtained from Microsoft.
[31] We approached 39 firms to participate in the survey phase. We followed up with phone calls and emails to solicit responses, eventually collecting survey data on 29 projects from 17 firms (a response rate of 43%). We collected additional descriptive data for 22 of the 29 individual projects through interviews.
[32] Specifically, projects to develop services such as web sites were found to have higher lines of code per person-day figures, due to their use of languages such as HTML, as opposed to C or C++ or Java.

The second outcome measure captured the quality of the product, relative to "competitive products that targeted similar customer needs at the time the product was launched." Quality was defined as a combination of breadth of features, performance (e.g., speed of operation) and reliability. Products were evaluated by a panel of 14 independent reviewers from leading industry journals using a two-round Delphi process (Dalkey and Helmer, 1963). This ensured that reviewers compared each product against a similar competitive set, a necessary task given many product segments were new to the industry at the time. Experts scored each product on a 7-point Likert scale, a score of 4 indicating a product was of comparable quality to competitors. Experts also reported how familiar they were with each product. We used the mean quality score of experts that were familiar with each product as the second performance measure.