The Architecture of Platforms: A Unified View

Carliss Y. Baldwin*

C. Jason Woodard†

*corresponding author:
Harvard Business School
Boston, MA, 02163
cbaldwin@hbs.edu

† Singapore Management University
jwoodard@smu.edu.sg

The Architecture of Platforms: A Unified View

Carliss Y. Baldwin and C. Jason Woodard

Abstract

The central role of "platform" products and services in mediating the activities of disaggregated "clusters" or "ecosystems" of firms has been widely recognized. But platforms and the systems in which they are embedded are very diverse. In particular, platforms may exist within firms as product lines, across firms as multi-product systems, and in the form of multi-sided markets. In this paper we argue that there is a fundamental unity in the architecture of platforms. Platform architectures are modularizations of complex systems in which certain components (the platform itself) remain stable, while others (the complements) are encouraged to vary in cross-section or over time. Among the most stable elements in a platform architecture are the modular interfaces that mediate between the platform and its complements. These interfaces are even more stable than the interior core of the platform, thus control over the interfaces amounts to control over the platform and its evolution. We describe three ways of representing platform architectures: network graphs, design structure matrices and layer maps. We conclude by addressing a number of fundamental strategic questions suggested by a unified view of platforms.

Introduction

Product and system designers have long exploited opportunities to create families of complex artifacts by developing and recombining modular components. An especially common design pattern has come to be associated with the concept of a *platform*, which we define as a set of stable components that supports variety and evolvability in a system by constraining the linkages among the other components. Our goal in this chapter is to shed light on the relationships between platforms and the systems in which they are embedded, in order to better understand firms and industries where platforms play an important role.

We begin by reviewing the use of the term "platform" in three distinct but related fields: product development, technology strategy and industrial economics. Although the term is used in diverse ways that seem difficult to reconcile, we find a number of common threads—most importantly, the conservation or reuse of a core component to achieve economies of scale while reducing the cost of creating a wide variety of complementary components.

This combination of conservation and variety has been harnessed both in product lines within the boundaries of a single firm (e.g., the Sony Walkman) and in large clusters or ecosystems of interdependent firms (as in the computer industry), as well as in the more general setting of multi-sided markets such as credit cards, shopping malls and search engines. We argue that the fundamental architecture behind all platforms is essentially the same: namely, the system is partitioned into a set of "core" components with low variety and a complementary set of "peripheral" components with high variety (Tushman and Murmann, 1998). The low-variety components constitute the platform. They are the long-lived elements of the system and thus implicitly or explicitly establish the system's interfaces, the rules governing interactions among the different parts.

This underlying architectural unity motivates our effort to develop a common set of analytical tools for studying platforms and the industries that produce them. Among the most important tools are representation schemes—in other words, ways to draw pictures of platform architectures. We discuss approaches from three different literatures: network graphs from organization theory, design structure matrices from engineering design, and layer maps from technology strategy. Each of these representations has both strengths and limitations, which we describe.

The chapter concludes by addressing what we feel are the most salient questions suggested by a unified view of platform architectures. First, when is a platform architecture preferable to allowing all components to vary arbitrarily? Second, when can a platform and its peripheral components (complements) remain within the control of a single firm? Third, when should a firm allow or encourage outsiders to develop complements to a platform it controls? And fourth, if a firm does allow—or is forced to accept—external complementors, which components of the system should it strive to retain control over?

What is a platform, really?

According to the Oxford English Dictionary, the word "platform" has been used since the 16th century to denote "a raised level surface on which people or things can stand, usually a discrete structure intended for a particular activity or operation." Somewhat surprisingly, the word has been used in an abstract sense for nearly as long. The OED cites examples from as early as 1574 in which "platform" refers to "a design, a concept, an idea; (something serving as) a pattern or model."

More recently, the concept of a platform has been developed by management scholars in three overlapping waves of research, respectively focused on products, technological systems and transactions. We briefly sketch these waves in the remainder of this section, then examine their common underlying logic in the next. Our citations to this vast literature are merely illustrative; a thorough review would be a substantial and worthwhile project in its own right.

Product development researchers first used the term "platform" to describe projects that created a new generation or family of products for a particular firm. In their seminal work on product development planning and execution, Wheelwright and Clark (1992a) introduced the term "platform product" to describe new products that "meet the needs of a core group of customers but [are designed] *for easy modification* into derivatives through the addition, substitution, or removal of features" (p. 73; emphasis added). This work was followed by research on "platform investments" (Kogut and Kulatilaka, 1994), "platform technologies" (Kim and Kogut, 1996), and more generally, "platform thinking" (Sawhney, 1998), along with rich field studies (e.g., Sanderson and Uzumeri,

1995) and managerial advice on platform-oriented product planning (e.g., Meyer and Lehnerd, 1997; Robertson and Ulrich, 1998).

In the second wave, *technology strategists* identified platforms as valuable points of control (and rent extraction) in an industry. Competition between platforms thus came to be seen as an important force at the industry level, with the ability to determine both the success and failure of firms and the evolution of product designs. Bresnahan and Greenstein (1999) developed a theory to explain the evolving structure of the computer industry, which remained concentrated around a small number of dominant platforms even as competition intensified within certain market segments. Influential studies of Microsoft and Netscape illustrated contrasting approaches to market leadership, with Microsoft forming a "Platforms Group" to consolidate its efforts around the Windows operating system (Cusumano and Selby, 1995), and Netscape adopting a "cross-platform" strategy—in part by creating a new internal platform, the Netscape Portable Runtime, to permit its browser to work with any operating system (Cusumano and Yoffie, 1998). Using Intel, Microsoft and Cisco as major case studies, Gawer and Cusumano (2004) built on this work to articulate a more general framework for "platform leadership" in rapidly evolving product systems.

Industrial economists subsequently adopted the term "platform" to characterize products, services, firms, or institutions that mediate transactions between two or more groups of agents (Rochet and Tirole, 2003). This literature emphasizes situations in which network externalities across these groups create a "chicken-and-egg problem," which must be solved by platform owners, typically by cross-subsidizing between groups or even giving away products or services for free (Parker and Van Alstyne, 2005). While this research has built on the previous waves to explain the competitive dynamics of system industries (see, e.g., Evans, Hagiu and Schmalensee, 2006), its empirical scope is more general, including phenomena as diverse as credit card payment networks, shopping malls and dating services (Eisenmann, 2008; Hagiu, 2008).

In the literatures of product development, technology strategy and industrial economics, many seemingly disparate things have been labeled platforms, including auto body frames, video game consoles, software programs, websites, back-office processing systems, shopping malls and credit cards. However, across these literatures, the things called platforms have common roots in engineering design. This shared heritage motivates our effort to identify common structural features, which we call "the architecture of platforms," and to develop tools for representing and reasoning about platforms in a general way.

Platform architecture and design rules

Most platform definitions focus on the reuse or sharing of common elements across complex products or systems of production. For example, according to Meyer and Lehnerd (1997), "A product platform is a set of common components, modules, or parts from which a stream of derivative products can be efficiently created and launched" (p. 7). Robertson and Ulrich (1998) defined a platform more broadly as "the collection of assets that are shared by a set of products" (p. 20), where assets may include components, processes, knowledge and people. Bresnahan and Greenstein (1999) extended the concept to the industry level: "We define a platform as a bundle of standard components around which buyers and sellers coordinate efforts" (p. 4).

The economic logic of component reuse is simple but powerful. For components that are fixed, it is often possible to realize economies of scale through increased production volume, amortization of fixed costs across product families or generations, and more efficient use of complementary assets (e.g., distribution channels and technical support services). At the same time, economies of scope are created at the system level by reducing the cost of developing product variants that are targeted at different markets or incorporate new technologies. Moreover, decisions to develop these variants frequently take the form of real options, allowing firms to run multiple design experiments and select only the best outcomes without compromising the functioning of the whole system (Baldwin and Clark, 2000).

Although it is a recurring theme across diverse literatures, the reuse and sharing of the core components of a platform is only half the story. Wheelwright and Clark observed that "a platform project creates products (and processes) that *embed an architecture* for [a] system … . In fact, *it is the architecture of the system that enables other features to be added or existing features to be removed* in tailoring derivative products to special market niches" (1992b, p. 96; emphasis added). The relationship between platforms and architecture has received little attention up to now, but we think it is crucial to understanding the nature of platforms and hence the dynamics of platform-based innovation and competition.

What is an architecture?

Product architecture was defined by Ulrich (1995) as "the scheme by which the function of a product is allocated to physical components" (p. 419). This definition, also used by Baldwin and Clark (2000), draws a distinction between the architecture of a system and its interfaces, which determine how the different components of the system will coordinate their behavior to work together. More recently, however, scholars in the fields of engineering and product design have begun to collapse this distinction by including interface specifications within the definition of architecture (Whitney et al., 2004; Fixson and Park, 2008). For example, in a definitive white paper, the Engineering Systems Division at MIT defined the architecture (of any complex system) as "an abstract description of the entities of a system *and how they are related*" (Whitney et al., 2004, p. 2; emphasis added). Thus the term has come to encompass not only a description of the system's overall structure and function, but also the constraints that govern the relationships among components and allow them to interoperate.

The fundamental feature of a *platform architecture*, in our view, is that certain components remain fixed over the life of the platform, while others are allowed to vary in cross-section or change over time. Thus either by design or simply because it is the longest-lived component in the system, a platform embodies a set of stable constraints, or *design rules*, that govern the relationships among components. This view is consistent with the emphasis on reuse and sharing expressed by other authors, but suggests additional implications for the evolution of platform systems over time.

In particular, fixing the interfaces between components creates specific *thin crossing points* in the network of relationships between the elements of the system (Baldwin, 2008). At these points, the dependencies between the respective components are constrained to obey the interface specification, while other forms of dependency are ruled out. (In contrast, when there is no pre-specified interface governing interactions, designers may introduce any form of dependency that seems beneficial.)

Interfaces in turn establish the boundaries of *modules*—components of a system whose "elements are powerfully connected among themselves and relatively weakly connected to elements in other [components]" (Baldwin and Clark, 2000, p. 63). Because they define points of weak linkage (thin crossing points) in a network of relationships, modular interfaces reduce both coordination costs and transaction costs across the module boundary (Baldwin and Clark, 2000; Baldwin, 2008). It follows that the existence of modules and modular interfaces in a large system reduces the costs of splitting design and production across multiple firms (Langlois and Robertson, 1992; Sanchez and Mahoney, 1996). This kind of disaggregation gives rise to *modular clusters* or *business ecosystems* of complementary and competing firms (Baldwin and Clark, 1997; Iansiti and Levien, 2004; Baldwin and Woodard, 2007). The modular cluster form of industrial organization tends to be good for consumers and for component and system producers collectively, though it is potentially dangerous for the platform architect, as IBM discovered when it lost control of its personal computer architecture to two of its component suppliers, Intel and Microsoft (Ferguson and Morris, 1993).

Platforms and evolvability

An important property of platform systems is that they are evolvable, in the sense that they can adapt to unanticipated changes in the external environment. In both biological and economic systems, evolution proceeds via the mechanisms of variation and selective retention of advantageous forms (Campbell, 1965). Darwin himself went to great lengths to establish variation as an empirical fact in biological systems, but he did not consider how such variability arose.

In recent years, biologists have turned their attention to this question. They have theorized that in complex multi-cellular organisms, the huge variety in outward forms is in fact accomplished through the *conservation* of core metabolic processes at the cellular level. These core processes are conserved precisely because they deconstrain other, complementary processes that support variation, and thus facilitate evolutionary adaptation (Kirschner and Gerhart, 1998). Thus it is the *combination* of stable core processes and variable complementary processes that generates useful (i.e., non-lethal) variation in complex organisms. Moreover, the conserved processes "seem designed to minimize the interdependence of processes, [and thus] are flexible and robust mechanisms that support change and variability in other processes." (p. 8426).

The same arguments can be applied to man-made systems. A platform architecture partitions a system into stable core components and variable peripheral components. By promoting the reuse of core components, such partitioning can reduce the cost of variety and innovation *at the* *system level*. The whole system does not have to be invented or rebuilt from scratch to generate a new product, accommodate heterogeneous tastes, or respond to changes in the external environment. In this fashion, the platform system as a whole becomes *evolvable*: it can be adapted at low cost without losing its identity or continuity of design.

From systems biology we learn when platform architectures are advantageous. Simple systems (like bacteria) can and do vary their core processes (Kirschner and Gerhart, 1998). But there is a limit to the complexity that can be achieved when every subsystem is subject to change. Thus a platform architecture is advantageous when the system as a whole is complex—comprised of many interacting parts—but its requirements are heterogeneous, future technological developments are uncertain, and/or the system must adapt to unanticipated environmental changes. The benefits of a platform architecture are variety in the present and evolvability through time. The catch is that the platform architect must know which components should remain stable and which should vary. The architect must also be able to create stable yet versatile interfaces, which can accommodate linkages that are unforeseen at the time the architecture is created.

Different parts, same elephant?

The diversity of the literature on platforms raises the question of whether the various "platform" concepts are synonymous or simply evocative uses of the same word to mean different things. We subscribe to the former view, arguing that at the level of architecture all platform systems are fundamentally the same. On this view, a platform architecture displays a special type of modularity, in which a product or system is split into a set of components with low variety and high reusability, and another set with high variety and low reusability. The first set is called "the platform." The second has no generic name but might be called "the complements" of the platform. The platform and its complements are distinct modules in the system architecture. Their interoperability is made possible via design rules, that is, interface specifications binding on both.

To see the fundamental similarity of platform architectures, consider first the distinction between a product platform within a single firm (e.g., the Sony Walkman, described by Sanderson and Uzumeri, 1995), and a platform whose complements are supplied by many different firms (e.g., Microsoft Windows). While platforms that span firm boundaries raise numerous, wide-ranging *strategic* concerns—explored by many of the contributors to this volume—at the level of *architecture*, there is no inherent difference between these "internal" and "external" platforms. Both modularize the system in ways that facilitate component reuse and variety in product offerings. Both implicitly or explicitly specify interfaces that mediate interactions among components. Both allocate decision rights (again, at least implicitly) that determine who can interact with or modify which components in what ways.

At first glance, there *does* seem to be a difference between these technology platforms (both within firms and across firms) and the platforms featured in the literature on multi-sided markets. Although some examples are frequently discussed in both literatures (e.g., video games and computer operating systems), others appear to have little in common (e.g., credit card payment systems, shopping malls and search engines). However, the driving force behind multi-sided markets is the need to induce coordination among two or more groups of agents, and what they "coordinate on" is precisely a fixed point in the architecture of transactions in which they collectively participate. That fixed point may be a particular component or system, for example the Visa payment-processing system, which both issues cards to consumers and approves transactions on behalf of merchants. Or it may be a physical location, as in the case of a shopping mall or singles bar. Or it may be a convention, such as the use of currency as a medium of exchange (Grewal, 2008) or technical standards for compatibility between systems, such as the TCP/IP protocols of the Internet (Clark, 1988).

Just as technology platforms provide economies of scale and scope within and across firms, multi-sided market platforms exhibit economies of scale for the platform and economies of scope for the various groups that transact with each other. All platform systems also have the capacity to create low-cost, decentralized options, such as the ability to download new applications onto a mobile device, or to wander into a store in almost any city in the world with credit card in hand.

Finally, all platform systems exhibit tensions between platform owners and complementors. These tensions are variously expressed. For internal platforms, tension arises through the threat of entry by third-party component makers who can hook into the platform at its visible interfaces and create compatible substitutes for the firm's own components. For external platforms and multi-sided markets, the main threat is disintermediation: by replicating or reverse engineering the platform side of these interfaces, rivals may be able to “clone” the platform itself and compete with it directly.

In summary, platform architectures are united in that they partition a system into low- and high-variety components. The two different types of components can be combined into a working system because pre-specified interfaces regulate both sides. The interfaces must be stable relative to the components that depend on them, hence are, by definition, part of the platform. But the interfaces must also be versatile: they cannot overly constrain the complements or they will reduce the variety and flexibility of the system as a whole. Because interfaces are the main junction points in a platform system, they are a source of strategic tension between platform owners and actual or potential complementors.

In pointing out these similarities we do not intend to minimize the differences between different types of platforms. Our goal is simply to be able to talk about platform architectures across different fields in a general way. Having a common language in turn makes it easier to visualize platform architectures using a common set of representation techniques.

Picturing platform relationships

Herbert Simon famously said, “Every problem-solving effort must begin with creating a representation for the problem” and “solving a problem simply means representing it so as to make the solution transparent” (Simon, 1996, pp. 108 and 132). In other words, good representations can help illuminate important dimensions of a problem—in our case, to understand platform behavior and strategy.

However, any representation is an abstraction from reality, highlighting some features while obscuring others. In this section, we describe three ways of representing platforms and their architecture: network graphs, design structure matrices and layer maps. These representations originate in different literatures and draw attention to different aspects of the phenomenon. Below we describe each type, give examples, and discuss the strengths and limitations of each approach.

Network graphs

Network graphs, which have been used extensively to study social networks and other complex patterns of relationships, are probably the most common way of representing platforms and their complements. In these representations, products or firms are designated as nodes and relationships between them as links. In a simple platform system, the platform would be the central node (or hub) in the network, with complements (or complementors) radiating outward from that point. For example, Figure 1 is a network graph depicting alliances between software firms in 1990. As the central node in this network, one can surmise that IBM supplies platforms (both hardware and software) that support a wide variety of complementors.

=== INSERT FIGURE 1 ABOUT HERE ===

Network graphs like this are useful when the platform in question has a simple hub-and-spoke structure. In such cases, the graph can show (1) the existence of one or more central elements; (2) the size of the "cloud" of complements (or complementors); and (3) the evolution of the platform and complements over time. The visualizations are augmented by a host of metrics derived from social network theory and the study of complex systems.

The first limitation of this representation scheme is evident in the encircled subset in the upper right corner. Within the circle, the simple hub-and-spoke structure disappears: many of the complementors have formed alliances with one another. As the number of lateral connections grows, it is more difficult to see what is going on via the network graph.

The second limitation of network graphs is that they are best suited to depicting *reciprocal* relationships. What is actually shown in Figure 1 is the existence of an alliance *between* two firms. One cannot see whether one firm's software *depends on* the other firm's code. Directional dependencies can be represented in a network graph by drawing the links as one- or two-headed arrows. However, such dependencies are hard for the eye to distinguish unless they are very homogeneous (all arrows radiating outward, for example). Design structure matrices, discussed next, are a more powerful way to visualize directional dependencies within a complex system.

Design structure matrices

In the design of any artifact, a key relationship between design elements is "uses" or "depends on" (Parnas, 1972). Design element B *depends on* A if a change in A *may require* a subsequent change in B. For example, if the contents of a particular software file changes, the files that call it may need to change as well. The calling files *depend on* the original file. In a platform architecture, if the platform incorporates interfaces that complementors must use to access the system, the complements depend on the platform.

Directional relationships like "depends on" can be visualized using design structure matrices or DSMs (Eppinger, 1991; Baldwin and Clark, 2000). To construct a DSM, one first assigns the design elements—for example, software files—to the rows and columns of a square matrix. Then if element B depends on A, one puts a mark in the row of B and the column of A. When all dependencies have been accounted for, the result is a *square but not symmetric* matrix. For example Figure 2 shows a DSM for OpenSolaris, a software project maintained by Sun Microsystems. In this instance, the rows and columns represent the 12,080 files in the codebase, and the marks represent function calls from the row file to the column file.

=== INSERT FIGURE 2 ABOUT HERE ===

The main advantages of the matrix view are twofold. First, the matrix easily depicts asymmetric relationships. Second, the matrix can structure a very large amount of information without overwhelming the eye. (Imagine a network graph with 12,000 nodes and 10 million arrows: that is what is being represented by the matrix in Figure 2.)

Can we see a platform in this architecture? First, note that Solaris is an operating system, on which complementors can build applications. Thus Solaris *is* a platform, and we are looking inside it. (Indeed, we are focusing on the *kernel* of Solaris, which is a subset of the whole operating system.) However, by showing this interior view, we can see "a platform within the platform." In fact, there are two sets of files on which many others depend. First there are files which are called by many files in the system but do not themselves call others. These appear as a vertical stripe, called a "bus," at the far left of the matrix. Second, there is a set of approximately 3,000 files that are densely interconnected with each other. These appear as a densely shaded square block on the main diagonal. The vast majority of files in the system call into this "core," as can be seen from the densely populated columns directly above it. Recent work has shown that a densely connected core such as this tends to be conserved as the codebase evolves. One reason for this is that touching any part of the core may have wide-ranging, unpredictable ramifications, thus such code is "hard to kill" (MacCormack et al., 2007).

The DSM view reveals two things about a system. First, it shows whether there are any "thin crossing points," i.e., places where the system might be split apart into modules. In the case of Solaris, the vast majority of the code is either in the core or calls into the core, hence there are no natural breakpoints. In contrast, the system shown in Figure 3 has two modules separated by a thin crossing point (LaMantia et al., 2008). In this case, the large upper block of the DSM represents the company's proprietary platform, while the smaller lower block represents licensed-in code. The codebase was designed in this modular fashion to protect the platform from opportunism on the part of the licensor. With this architecture in place, the company could respond to the threat of hold-up by simply splitting the codebase at the thin crossing point, and substituting a new module for the licensed code.

Second, the DSM shows the degree to which the system is evolvable. Again, the kernel of Solaris does not appear very evolvable: the interior of the platform is quite monolithic. In contrast, the system shown in Figure 3 *is* evolvable: the platform can be conserved and the complementary, licensed-in code replaced at low cost. (Indeed, shortly after creating this architecture, the company began to offer products that used open-source code in place of the licensed code.)

=== INSERT FIGURE 3 ABOUT HERE ===

When one wants to look in detail at the structure of a system, especially the size and boundaries of modules, DSMs are a good alternative to network graphs. However, they too suppress features that are of interest for some types of analysis. For example, it is easier to depict the existence of variants—competing modules that do similar things—in a graphical representation than in matrix form. (One can illustrate substitutes by grouping sets of nodes together, cf. Woodard, 2007.) DSMs also indicate the *existence* of modular interfaces at thin crossing points, but do not indicate anything about those interfaces. Finally, DSMs usually require some prior knowledge of the system to generate informative clusters.[1]

Layer maps

The representations discussed thus far can be used to compare and contrast different platform architectures, but they cannot easily comprehend cases where components operate across platforms or different platforms compete. How platform architecture affects competition and complementarity in an industry or ecosystem is hard to see via network graphs or DSMs. In contrast, layer maps are a useful way to visualize “industry architectures” (Jacobides et al., 2006).

To the best of our knowledge, the first layer map was constructed by Andy Grove, then CEO of Intel Corporation, to depict the changing structure of the computer industry (Grove, 1996). As he explained:

> The computer industry used to be vertically aligned. … [A] company developed its own chips, its own hardware and its own software, sold and serviced by its own people.
>
> Over time, … a new horizontal industry emerged. … A consumer could pick a chip from [one vendor], choose an operating system [from another], grab one of several ready-to-use applications off the shelf … (pp. 39–42).

Figure 4 is an adaptation of Grove’s original maps. Although Grove did not use these terms, we have labeled the vertical industry architecture “vertical silos” and the horizontal architecture a “modular cluster.”

=== INSERT FIGURE 4 ABOUT HERE ===

To construct a layer map for a given industry, one first determines all the complementary components in a particular system. This list can be obtained from technical descriptions of the system. Components are then arranged vertically in separate layers, forming what engineers often call a “stack.” Then one determines who competes in the product markets defined by the layers. Vertically integrated firms span several layers; specialized component suppliers appear in only one.

[1] In the absence of such knowledge, automatic clustering algorithms can be used, but for matrices of a realistic size, such algorithms have proven to be both computationally intensive and unreliable in the sense that they yield different results from different starting points (Rusnak, 2005).

Layer maps can easily show changes in industry architecture over time. Usually, such changes occur after and as a consequence of changes in underlying product and process designs. Thus Baldwin and Clark (2000) found that the computer industry became more vertically disintegrated following the introduction of System/360's modular architecture. Vertical disintegration continued after IBM deployed a modular architecture and then encouraged external suppliers to provide both hardware and software for its PC. The change in industry architecture following the introduction of the PC can be seen in Figure 5, which shows layer maps of the greater computer industry in 1984 and 2005.[2] Conversely, Fixson and Park (2008) used layer maps and supporting analysis to show how the bicycle drive train industry became more concentrated and vertically integrated following Shimano's successful introduction of a highly integral (non-modular) product, the Shimano Index System (SIS).

=== INSERT FIGURE 5 ABOUT HERE ===

Simplified layer maps can be used to model "co-opetition," that is, the simultaneous presence of competitors and complementors in an industry (Brandenburger and Nalebuff, 1996). Most such models consider one or two firms in each of two layers (e.g., Casadesus-Masanell et al., 2008), or a two-layer platform structure with one firm (the platform) in one layer and *n* firms (the complementors) in the other (e.g., Hagiu, 2008). In a departure from these precedents, Baldwin and Woodard (2007) modeled price competition and complementarity in an industry with an arbitrary number of layers. For simple platform architectures, they showed that (1) platform architectures based on open, public interfaces may give rise to industry clusters that are as profitable, in aggregate, as a system-wide monopoly; and conversely, (2) competing platforms based on closed, proprietary interfaces tend to destroy value through ferocious price competition.

As with all representations, layer maps have their drawbacks. First and foremost, they assume that the components of a system can be neatly categorized into substitutes and complements,

[2] The "greater" computer industry is defined as all firms in the 17 NAICS industries supplying computer hardware and software. Areas in the layer map are assigned to firms on the basis of market capitalization. When a firm participates in several layers, market capitalization is allocated on the basis of revenue.

while in reality there are many shades of gray. For example, is flash memory a substitute or complement to a disk drive? Is an action game a substitute or complement to a multi-player game? The neat block structure of a layer map often hides a more messy underlying reality.

But hiding messy reality is exactly what a good representation is supposed to do. Details and ambiguities are suppressed so that patterns can be revealed. Each of the representations discussed in this section—network graphs, design structure matrices and layer maps—highlights some aspects of a platform architecture while obscuring others. Each can illuminate the underlying phenomenon; each can inform strategy by suggesting different risks, threats, or opportunities. But no single representation should be used to the exclusion of others.

Conclusion

Our unified view of platform architecture raises questions of interest to both researchers and strategists. We briefly discuss four of these questions below, then conclude with a summary of our basic argument.

1. When is a platform architecture useful?

Platform architectures are useful when the underlying system is complex, but needs to adapt to changing tastes and technologies. Complex systems, by definition, have many parts that must work together to achieve a functioning whole. But tight integration can lead to rigidity. Platform architectures, in contrast, make the system evolvable. Even the core components can evolve—only the interfaces need to be stable.

2. When will a platform system "spill over" the boundaries of a single firm? When can it be contained?

A platform system consists of a core, its complements, and the interfaces between them. Whether such a system can be contained within the bounds of a single firm and its supply chain depends on both the system design and the appropriability regime of the firm's industry (Teece, 1986). Systems are harder to contain when they have a modular structure with clean interfaces (i.e., very thin crossing points) and the interfaces are weakly appropriable, in the sense that they cannot be protected against duplication or reverse engineering.

The following account from a popular book on PC repair describes how Apple Computer managed to contain the Macintosh platform within its boundaries:

> [T]here are no clones or compatibles of the Apple Macintosh system. It is not that Mac [hardware] can't be duplicated …. The real problem is that *Apple owns the Mac OS* as well as the BIOS, and because Apple has seen fit not to license them, no other company can sell an Apple-compatible system. Also, note that *the Mac BIOS and OS are very tightly integrated*; the Mac BIOS is very large and complex and is essentially part of the OS, unlike the much simpler and more easily duplicated BIOS found on PCs. The greater complexity and integration has allowed both the Mac BIOS and OS to escape any clean-room duplication efforts. This means that without Apple's blessing (in the form of licensing), no Mac clones are likely ever to exist. (Mueller, 2003, p. 28; emphasis added.)

In other words, there are two lines of defense for a firm seeking to prevent others from supplying complements to its platform. The first is the ownership of critical interface components (Apple owns the Mac OS and BIOS and can choose to license them or not). The second line is "complexity and integration," in other words lack of modularity: the fact that the interface components do not create "sufficiently thin" crossing points between the different parts of the system. Complexity and integration of the firm's *internal platform design* protect the interfaces from legal duplication via clean-room reverse engineering techniques. But, at the same time, complexity and integration make the overall system less flexible, hence less evolvable. Recalling the Solaris DSM pictured in Figure 2, there are few places in this architecture for an outside party to offer a competing subsystem. Hence the Solaris platform would be relatively easy to contain, if Sun wished to do so.[3] By the same token, the *interior* of Solaris does not appear very evolvable, although it may support evolution in the surrounding system.

The flip side of this question is, when will outsiders try to "invade" a platform? This is closely related to the next question. The short answer is, whenever they can and there are sufficient financial incentives to do so!

3. When should the platform owner encourage outsiders to develop complements to the platform? In other words, when should the owner adopt a "platform strategy"?

Outside complementors can be of great value to the system when there is a lot of "option

[3] Sun kept Solaris proprietary for many years, but in 2005 released much of its code under an open-source license. See http://www.sun.com/smi/Press/sunflash/2005-01/sunflash.20050125.1.xml (viewed 26 August 2008).

potential" in the complementary modules. An option is "the right but not the obligation to take a specific action," in this case, choose one complement over another. Option value is low when consumer tastes are homogeneous and predictable, and designs are on a tightly determined technological trajectory (Dosi, 1982). In such cases, it is usually obvious what will succeed in the market, hence the value of multiple experiments and diverse approaches is low.

Option value is high when consumer tastes are heterogeneous or unpredictable, and technological trajectories are uncertain. In these cases, it is not obvious what will succeed, hence the value of multiple experiments and diverse approaches is high. Outside complementors will be attracted to the platform if there is option value in the complements, *provided the platform owner does not expropriate all the value they create*. Iansiti and Levien (2004) call the excessive expropriation of value a "dominator strategy," and argue that such strategies tend to yield unhealthy business ecosystems. Woodard (2008) finds using computational experiments that even selfish and fairly myopic platform owners can learn to avoid "overtaxing" their ecosystem members and find a balance that yields more investment in platforms with higher option value.

Before adopting a platform strategy, however, the architect must ask, what can outside complementors do that the architect's own employees cannot? This question deserves careful consideration. In the first place, outsiders may have skills, capabilities, or an understanding of user needs that those inside the firm do not. Commenting on the possibilities of innovation outside the boundaries of his firm, Bill Joy, a founder of Sun Microsystems, famously said, "Not all smart people work for you."[4] The corollary of this observation is, "Smart people are hard to find, but may find you if you open up your platform."

Second, because they have property rights in their own output, outside complementors experience high-powered incentives that can induce them to greater levels of effort than insiders (Baker, Gibbons and Murphy, 2002). Furthermore, in a world of imperfect capital markets, complementors may be able to provide investment capital that would not be available to the platform owner directly. Accessing this capital in turn makes it possible for a successful platform system to grow faster than its competitors (Baldwin and Clark, 2006). For example, Boudreau (2008)

[4] Source: http://www.pbs.org/cringely/nerdtv/transcripts/003.html (viewed 14 August 2008).

found that, in handheld computing devices, opening up the platform to external developers increased the rate of new device releases by a factor of five. When network externalities are present, such growth can propel the platform to competitive success (see, e.g., Shapiro and Varian, 1999).

Finally, platform complementarity can be a transient relationship that does not warrant employment or any other type of formal contract. This is often the case in multi-sided markets. The buyers and sellers on eBay want to transact with one another, and are willing to pay a fee to the platform if it reduces their transaction costs. The same holds for the merchant and customer in a credit card transaction, and the searchers, searchees and advertisers on Google. For these agents, participation in the platform is a small part of a much larger set of activities. Bringing these complementors "inside the walls" of the platform owner is simply not a sensible option.

4. In a platform strategy with external complementors, which components of the overall system should be retained and developed by the platform architect?

In any platform system, there are three types of components: (1) the complements, which exhibit high variety and high rates of change over time; (2) the core components, which remain stable as the complements change; and (3) the interfaces, which are the design rules that allow the core and the complements to operate as one system. Both the core components and the interfaces are relatively long-lived, hence part of "the platform."

At first glance, it might appear that the architect should retain control of the core of the system, but this conclusion is not always correct. In man-made systems, the core components of the platform can evolve over time, hence may be subject to competitive pressures. For example, the core of the first IBM PC was the system board, which contained (among other things) an Intel 8088 CPU. By design, all manner of hardware and software could be connected via the system board and would interoperate. This gave users a wide range of configuration options, and gave complementors rich incentives to develop better hardware devices and new software applications. Not by accident, the core was what IBM chose to sell.

However, in contrast to biological systems in which core processes are conserved through evolutionary change (Kirschner and Gerhart, 1998), the core processes of the original IBM PC were not conserved. The system board of a Lenovo or Dell machine today is very different from that of a 1981 PC. A Pentium processor running at 2.2 GHz is worlds apart from the 8088 processor in the original machine.

What *has* been conserved are the interfaces that sit between the system board, the hardware, and the software. There were three such interfaces: (1) the Basic Input/Output System (BIOS), which insulated software developers from the hardware so that new devices could be added without rewriting programs; (2) the instruction set of the Intel 8088 chip, the basic commands into which all software had to be translated; and (3) the operating system, a set of "higher-level" services that human programmers could use instead of accessing the BIOS or CPU directly. All of these interfaces have survived to the present day, and thus many programs written and compiled for a 1981 IBM PC will run correctly on a Lenovo or Dell PC in 2008.

This example teaches us that in a man-made platform, the interior of the core—the part hidden behind the interfaces—is not an essential part of the platform. In contrast, the interfaces *are* essential: the PC interfaces have lasted much longer than their hardware implementations, and their strategic significance has been profound.

What was problematic for IBM in the PC platform architecture was that IBM controlled only one of three critical interfaces, namely the BIOS. This interface, moreover, was simple and not tightly integrated with other parts of the system (cf. Mueller, 2005, quoted above). As is well known, the BIOS was reverse engineered by Compaq and then Phoenix Technologies in the early 1980s (Ferguson and Morris, 1993). At that point, ownership of the true PC platform—those components that would be conserved over the lifetime of the system—devolved onto Intel and Microsoft, where it remains to this day.

Summary of the argument

We have argued that there is a fundamental unity in the architecture of platforms. In essence, a "platform architecture" is a modularization which partitions the system into (1) a set of components whose design is stable and (2) a complementary set of components which are allowed—indeed encouraged—to vary. The combination of stability and variety is accomplished via "stable, yet versatile" interfaces, which govern the interactions of components. The interface specifications are part of the platform; indeed they may be the only components that remain truly stable over long periods of time. The combination of stability and variety in the architecture makes it possible to create novelty without developing a whole new system from scratch. Thus platform systems are evolvable.

Although they display a fundamental unity at the level of architecture, platform systems vary a great deal in construction and appearance. Some are physical (a singles bar), others are virtual (a social networking website). Some platform systems are contained within a single firm or a supply chain, while others are spread over ecosystems consisting of thousands or tens of thousands of firms. Many of these platforms support multi-sided markets.

A benefit of viewing platform architectures in a unified way is that theories and observations of seemingly disparate phenomena in diverse fields can be brought into focus as part of a coherent whole. We hope that this view will in turn serve as a focusing device for new efforts in theory-building and empirical analysis.

Acknowledgements

We are grateful to Annabelle Gawer for conceiving of this project and to the participants of the Platforms, Markets and Innovation Conference at Imperial College, London on 2 June 2008 for many thoughtful and productive comments. We are greatly indebted to Alan MacCormack, John Rusnak, Matt LaMantia, Yuanfang Cai, Michael Jacobides, Reza Dizaji, Bala Iyer, Chi-Hyon Lee, and Venkat Venkatraman for allowing us to use their graphs, matrices and layer maps. Finally we thank the Harvard Business School Division of Research and the Singapore Management University School of Information Systems for generous financial support.

References

Baker, George, Robert Gibbons and Kevin J. Murphy (2002), 'Relational contracts and the theory of the firm,' *Quarterly Journal of Economics*, **117** (1), 39–84.

Baldwin, Carliss Y. (2008), 'Where do transactions come from? Modularity, transactions, and the boundaries of firms,' *Industrial and Corporate Change*, **17** (1), 155–195.

Baldwin, Carliss Y. and Kim B. Clark (1997), 'Managing in an age of modularity,' *Harvard Business Review*, **75** (5), 84–93.

Baldwin, Carliss Y. and Kim B. Clark (2000), *Design Rules, Volume 1: The Power of Modularity*, Cambridge, MA, US: MIT Press.

Baldwin, Carliss Y. and Kim B. Clark (2006), 'Architectural innovation and dynamic competition: The smaller "footprint" strategy,' Harvard Business School Working Paper 07-014, August.

Baldwin, Carliss Y. and C. Jason Woodard (2007), 'Competition in modular clusters,' Harvard Business School Working Paper 08-042, December.

Boudreau, Kevin (2008), 'Opening the platform vs. opening the complementary good? The effect on product innovation in handheld computing,' working paper, August.

Brandenburger, Adam M. and Barry J. Nalebuff (1996), *Co-opetition*, New York: Doubleday.

Bresnahan, Timothy F. and Shane Greenstein (1999), 'Technological competition and the structure of the computer industry,' *Journal of Industrial Economics*, **47** (1), 1–40.

Campbell, Donald T. (1965), 'Variation and selective retention in socio-cultural evolution,' In Herbert R. Barringer, George I. Blanksten and Raymond W. Mack (eds), *Social Change in Developing Areas: A Reinterpretation of Evolutionary Theory*, Cambridge, MA, US: Schenkman; pp. 19–49.

Casadesus-Masanell, Ramon, Barry Nalebuff and David B. Yoffie (2008), 'Competing complements,' Harvard Business School Working Paper 09-009, 21 November.

Clark, David D. (1988), 'The design philosophy of the DARPA Internet protocols,' *Computer Communications Review*, **18** (4), 106–114.

Cusumano, Michael A. and Richard W. Selby (1995), *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets and Manages People*, New York: Free Press.

Cusumano, Michael A. and David B. Yoffie (1998), *Competing on Internet Time: Lessons From Netscape and Its Battle with Microsoft*, New York: Free Press.

Dosi, Giovanni (1982), 'Technological paradigms and technological trajectories: A suggested interpretation of the determinants and directions of technical change,' *Research Policy*, **11** (3), 147–162.

Eisenmann, Thomas R. (2008) "Managing Shared and Proprietary Platforms," *California Management Review*, 50(4): 31-53.

Eppinger, Steven D. (1991), 'Model-based approaches to managing concurrent engineering,' *Journal* *of Engineering Design*, **2** (4), 283–290.

Evans, David S., Andrei Hagiu and Richard Schmalensee (2006), *Invisible Engines: How Software Platforms Drive Innovation and Transform Industries*, Cambridge, MA, US: MIT Press.

Ferguson, Charles H. and Charles R. Morris (1993), *Computer Wars: How the West Can Win in a Post-IBM World*, New York: Times Books.

Fixson, Sebastian K. and Jin-Kyu Park (2008), 'The power of integrality: Linkages between product architecture, innovation, and industry structure,' *Research Policy*, **37** (8), 1296–1316.

Gawer, Annabelle and Michael A. Cusumano (2004), *Platform Leadership: How Intel, Microsoft, and Cisco Drive Industry Innovation*, Boston: Harvard Business School Press.

Grewal, David Singh (2008), *Network Power: The Social Dynamics of Globalization*, New Haven: Yale University Press.

Grove, Andrew S. (1996), *Only the Paranoid Survive*, New York: Doubleday.

Hagiu, Andrei (2008), 'Two-sided platforms: Product variety and pricing structures,' working paper, 29 July.

Iansiti, Marco and Roy Levien (2004), *The Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability*, Boston: Harvard Business School Press.

Iyer, Bala, Chi-Hyon Lee and N. Venkatraman (2006), 'Managing in a "small world ecosystem": Lessons from the software sector,' *California Management Review*, **48** (3), 28–47.

Jacobides, Michael G., Thorbjørn Knudsen and Mie Augier (2006), 'Benefiting from innovation: Value creation, value appropriation and the role of industry architectures,' *Research Policy*, **35** (8), 1200–1221.

Kim, Dong-Jae and Bruce Kogut (1996), 'Technological platforms and diversification,' *Organization Science*, **7** (3), 283–301.

Kirschner, Marc and John Gerhart (1998), 'Evolvability,' *Proceedings of the National Academy of Sciences*, **95**, 8420–8427.

Kogut, Bruce and Nalin Kulatilaka (1994), 'Options thinking and platform investments: Investing in opportunity,' *California Management Review*, **36** (2), 52–71.

LaMantia, Matthew J., Yuanfang Cai, Alan D. MacCormack and John Rusnak (2008), 'Analyzing the evolution of large-scale software systems using design structure matrices and design rule theory:

Two exploratory cases,' *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, Washington, DC, US: IEEE Computer Society, 83–92.

Langlois, Richard N. and Paul L. Robertson (1992), 'Networks and innovation in a modular system: Lessons from the microcomputer and stereo component industries,' *Research Policy*, **21** (4), 297–313.

MacCormack, Alan, John Rusnak and Carliss Baldwin (2007), 'The impact of component modularity on design evolution: Evidence from the software industry,' Harvard Business School Working Paper 08-038.

Meyer, Marc H. and Alvin P. Lehnerd (1997), *The Power of Product Platforms: Building Value and Cost Leadership*, New York: Free Press.

Mueller, Scott (2003), *Upgrading and Repairing PCs*, 15th ed., Indianapolis: Que Publishing.

Parker, Geoffrey G. and Marshall W. Van Alstyne (2005), 'Two-sided network effects: A theory of information product design,' *Management Science*, **51** (10), 1494–1504.

Parnas, David L. (1972), 'On the criteria for decomposing systems into modules,' *Communications of the ACM*, **15** (12), 1053–1058.

Robertson, David and Karl Ulrich (1998), 'Planning for product platforms,' *Sloan Management Review*, **39** (4), 19–31.

Rochet, Jean-Charles and Jean Tirole (2003), 'Platform competition in two-sided markets,' *Journal of the European Economic Association*, **1** (4), 990–1029.

Rusnak, John (2005), *The Design Structure Analysis System: A Tool to Analyze Software Architecture*, Ph.D. thesis, Harvard University.

Sanchez, Ronald A. and Joseph T. Mahoney (1996), 'Modularity, flexibility and knowledge management in product and organizational design,' *Strategic Management Journal*, **17** (winter special issue), 63–76.

Sanderson, Susan and Mustafa Uzumeri (1995), 'Managing product families: The case of the Sony Walkman,' *Research Policy*, **24** (5), 761–782.

Sawhney, Mohanbir S. (1998), 'Leveraged high-variety strategies: From portfolio thinking to platform thinking,' *Journal of the Academy of Marketing Science*, **26** (1), 54–61.

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

Teece, David J. (1986), 'Profiting from technological innovation: Implications for integration, collaboration, licensing and public policy,' *Research Policy*, **15** (6), 285–305.

Tushman, Michael L. and Johann Peter Murmann (1998), 'Dominant designs, technology cycles and organizational outcomes,' *Research in Organizational Behavior*, **20**, 231–266.

Ulrich, Karl (1995), 'The role of product architecture in the manufacturing firm,' *Research Policy*, **24** (3), 419–440.

Wheelwright, Steven C. and Kim B. Clark (1992a), 'Creating project plans to focus product development,' *Harvard Business Review*, **70** (2), 67–83.

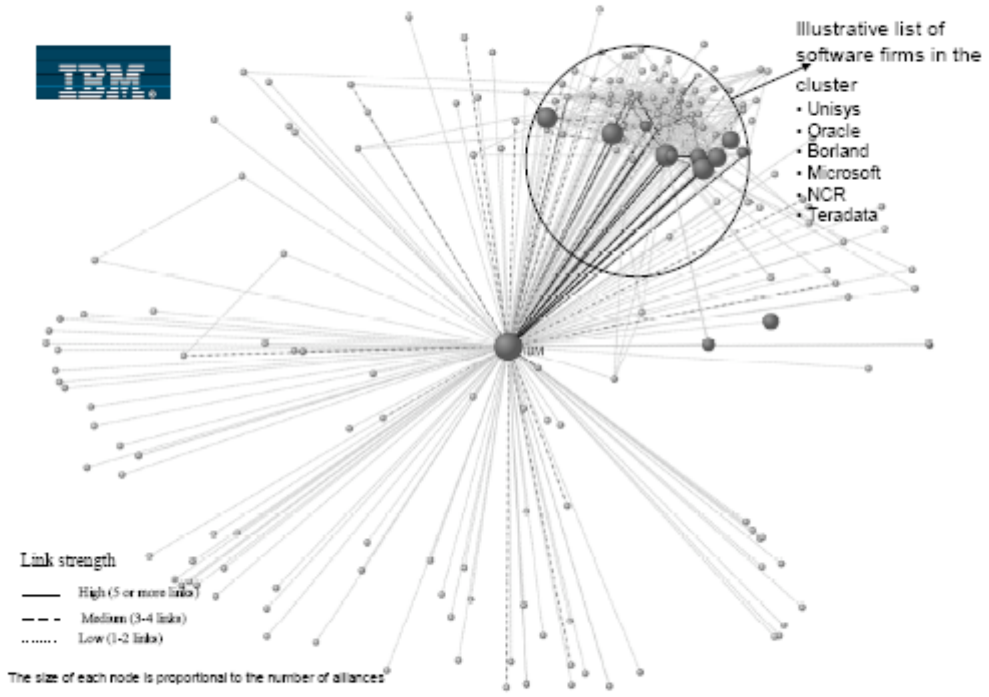
Wheelwright, Steven C. and Kim B. Clark (1992b), *Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency, and Quality*, New York: Free Press.

Whitney, Daniel (chair) and the ESD Architecture Committee (2004), 'The influence of architecture in engineering systems,' Engineering Systems Monograph, MIT Engineering Systems Division, March.

Woodard, C. Jason (2007), 'Modeling architectural strategy using design structure networks,' paper presented at the Second International Conference on Design Science Research in Information Systems and Technology (DESRIST 2007), Pasadena, US, 13–15 May.

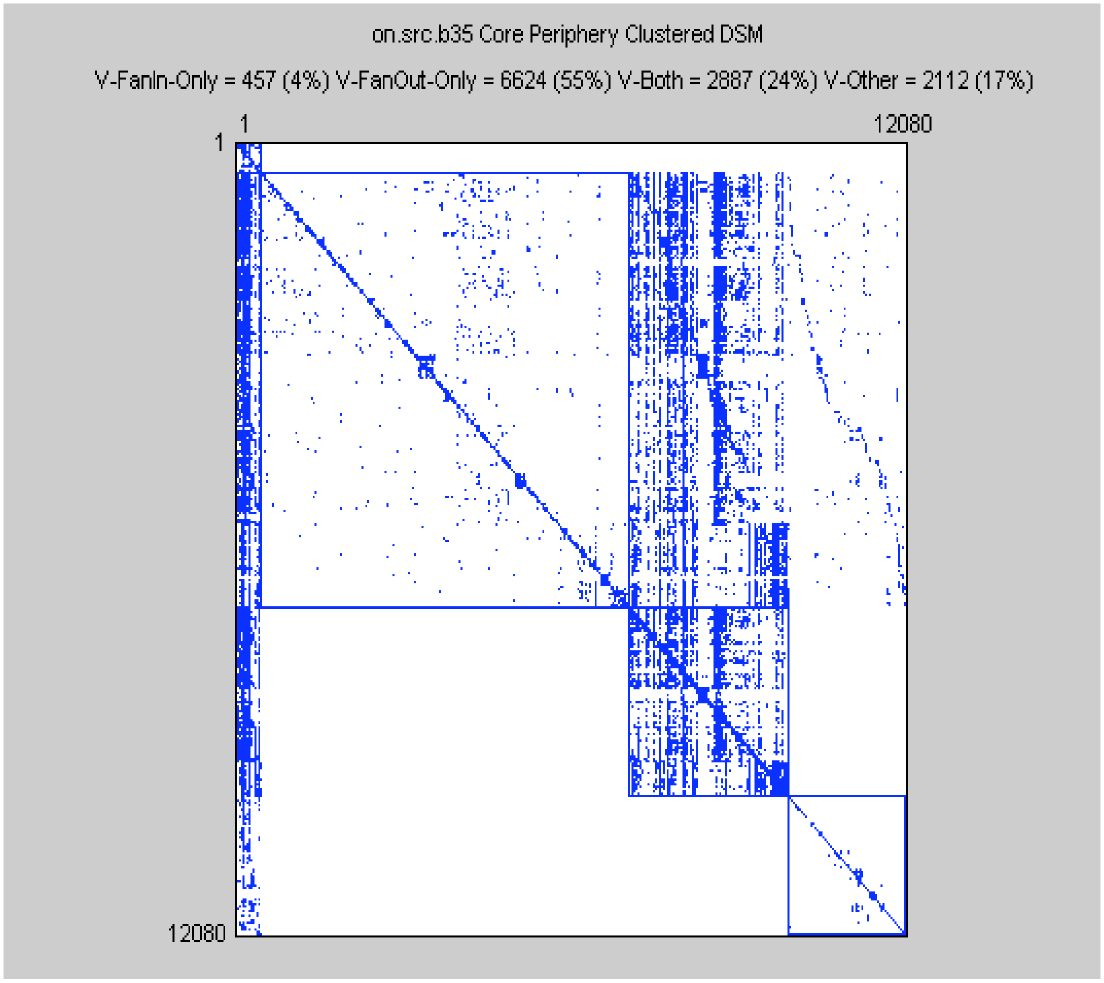
Woodard, C. Jason (2008), 'Platform competition in digital systems: Architectural control and value migration,' working paper, 30 May.

Figure 1
IBM's network in 1990



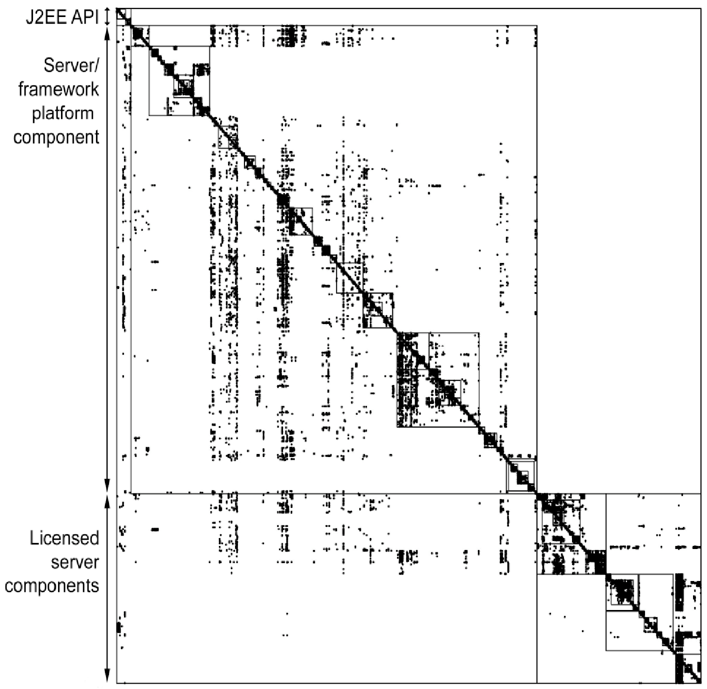
Source: Iyer, Lee and Venkatraman (2006). Reproduced by permission.

Figure 2
Design structure of OpenSolaris



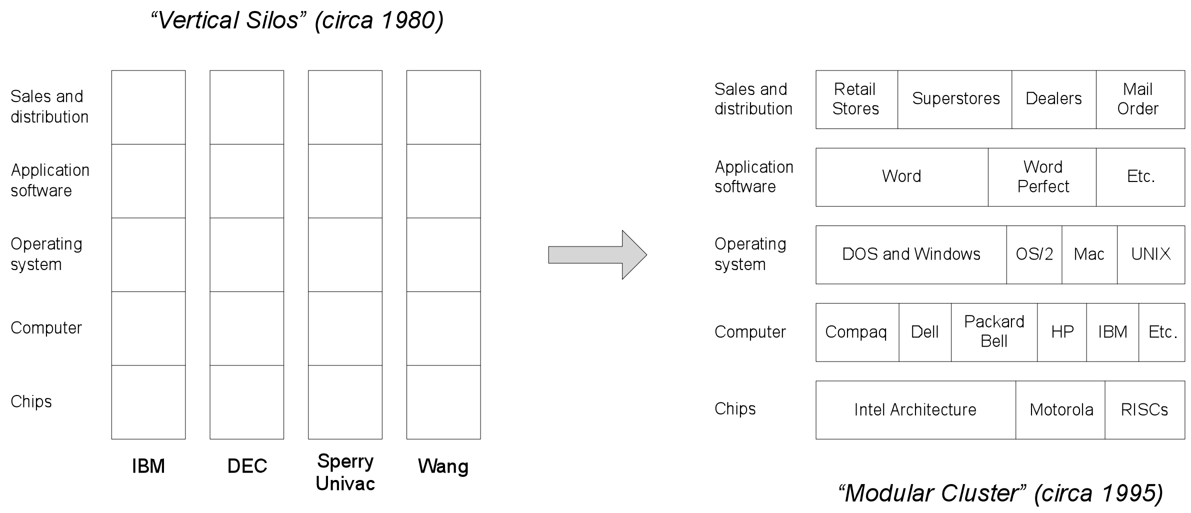
Source: A. MacCormack, J. Rusnak and C. Y. Baldwin, private communication. Reproduced by permission.

Figure 3
A system with two modules


J2EE API
Server/ framework platform component
Licensed server components

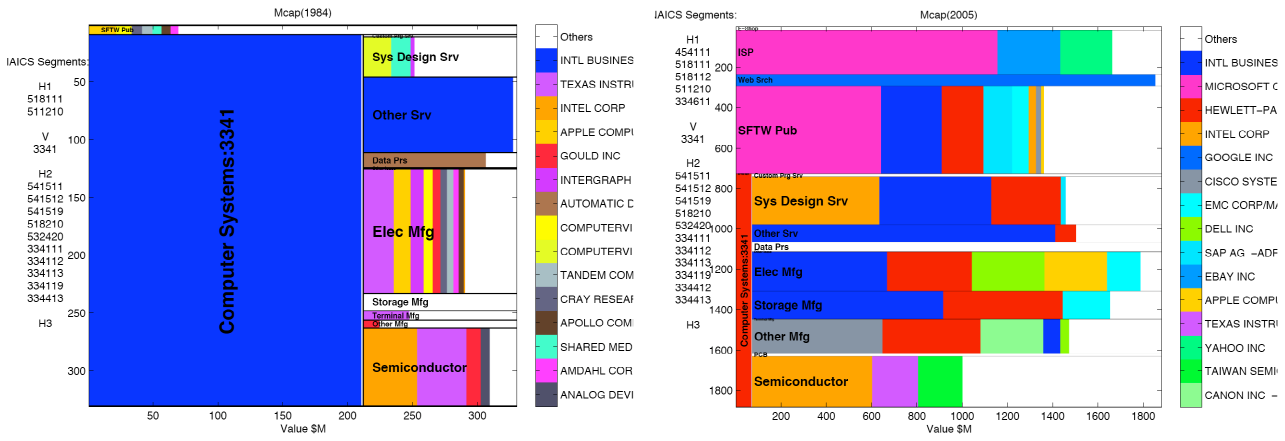
Source: LaMantia et al. (2008). Reproduced by permission.

Figure 4
The vertical-to-horizontal transition in the computer industry



Source: Adapted from Grove (1996, p. 44).

Figure 5
Layer maps of the computer industry in 1984 and 2005



Source: C. Y. Baldwin, M. G. Jacobides and R. Dizaji, private communication. Reproduced by permission.