Managerial Action in a Stacked Landscape

Bala Iyer	David Dreyfus

Boston University School of Management

595 Commonwealth Ave; Room 641A

Boston, MA 02215

Email:bala@bu.edu

Abstract

Strategic decision makers, Chief Information Officers, Enterprise System Architects, and software project leaders are faced with a similar set of questions: how do complex systems emerge, what are the patterns of emergence, and what are the trade-offs that guide emergence? We argue that complexity in a dynamic environment coupled with incremental innovation and an installed base leads to the formation of a specific pattern of modularity called stacks. Stacks are a type of hierarchy in which upper layers can utilize lower layers, but not vice versa. Stacks emerge as a result of incremental changes guided by a set of trade-offs: flexibility vs. efficiency, domain specificity vs. domain generalizability, system performance vs. user performance; and tight coupling vs. loose coupling. We utilize modularity theory and coordination theory to describe and explore the trade-offs and design moves that lead to stacks by presenting examples at the industry, organizational, architectural, and systems level.

Introduction

Outsourcing, off-shoring, and the development and promotion of web services are the most recent and visible changes to socio-technical systems known as extended enterprises. The enterprise creates a supply chain – better described as a supply web – that connects it to other firms within its own industry and to firms in other industries (Campbell-Kelly 2003; Shapiro et al. 1999). Organizational units are part of a similar value web within the enterprise: organizational units are differentiated and integrated with other units (Thompson 1967). At the systems level the pattern is repeated again. The major components that make up the enterprise architecture are integrated with other components. The collection of such components represents the enterprise's IT infrastructure (Kayworth et al. 2001); the pattern of interconnections between those components represents the enterprise's IT architecture. Within individual software applications, the pattern repeats as software modules are integrated to implement the application.

In order to make decisions regarding such complex, evolving systems, it is helpful to understand how they operate, what drives their evolution, and the trade-offs that are made with different decisions. As firms manage their evolution by competing to access resources within the extended enterprise (Ahuja 2000; Grant et al. 2004; Gulati et al. 1998), or developing and extending their information systems (Iyer et al. 2004), they need to understand the landscape, ecosystem or design space (Simon 1996 [1969]) – that they are operating in. To help managers accomplish this, we present abstractions that make it easier to comprehend the complex reality and a language to express complexity – modularity and stacks – that make it easier to navigate the competitive landscape.

Our abstraction starts with the concept of modularity and its capacity to create value through options (Baldwin et al. 2005). Modularity makes complex systems easier to manage by hiding interdependencies. Modules can specialize in certain functions and communicate with other modules through their interfaces. Taking a simplified view, firms are modules within an economy. Each firm specializes in a select set of products and services, hiding the details of its operations from its customers and partners while providing interfaces to integrate and coordinate them. Within the firm, functional and business units hide the details of their work from other units and integrate their activities through well-designed coordination mechanisms (Malone et al. 1994; Thompson 1967). Information systems within enterprises are similarly designed. The complex code and logic within each application is hidden from other applications, and the applications communicate through well-defined interfaces (Baldwin et al. 2000; Hopkins 2000).

The decision to outsource or off-shore the development of a web-service or the implementation of an extended supply chain depends on the characteristics of the interface between, for example, the outsourced activity and the other activities of the firm. In deciding whether a function should be outsourced or left in a functional business unit, one of the issues from a modularity perspective is interface definition. When a function is outsourced, the interface must generally be better defined and more stable than would otherwise be necessary in order to minimize transaction costs (Markides et al. 1994). The communication with an outsource partner across the interface is generally of lower volume, better structured, and more explicit than if it stayed within the firm (Brusoni et al. 2001). In general, the relationship with the outsourced function is more loosely coupled and the interdependencies between the outsourced function and the rest of the enterprise are lower (Brusoni et al. 2001).

Our abstraction builds on this notion of modularity by adding the concept of stack. Stacks are layered functionality, with each layer elaborating or specializing on the functionality of the layer below. When a company wants to meet a new set of customer demands, they add a new layer on top. A given module can call any layer but each layer can only access layers below it. The idea behind this is that new capabilities are built on top of existing ones and not from scratch. In suggesting that certain complex systems implement (or look like) a stack, we are saying that they show a specific type of modularity in which the logic of stacks overlays the general rules of modularity. The term ‘stack’ is used both descriptively and normatively: stacks emerge as a result of the forces underlying complex system evolution and are attractive to designers due to certain beneficial characteristics.

Stacks share certain characteristics of all modular systems: they help manage complexity, they enable parallel work, and they accommodate future uncertainty (Messerschmitt et al. 2003). Modular systems enable experiments; they provide the designer with valuable options (Baldwin et al. 2000). In a modular system, one module can be replaced without requiring all other modules to also change. But, modular designs do not have an internal hierarchy in which one component is at a higher level than another component. Stacks, however, have a well-defined hierarchy as implied by the metaphor. At the bottom layer is the core capability of the system, firm, or industry. At the top layer is the user, stakeholder, or dependent industry. The upper layers make use of the lower layers and not vice versa.

One of the consequences of stacks is that different layers within the stack can develop at different speeds (Bresnahan et al. 1999). The details of each layer are hidden from the layers above and below a given layer. Another consequence of stacks within an industry is that different firms can supply different layers of the stack, resulting in divided technical leadership (Bresnahan 1998). A third consequence is that customers and firms can experiment with alternative designs at a significantly lower cost than they could in the absence of layered modularity. This has been referred to as combinatorial innovation (Bresnahan et al. 1999). The idea is that every now and then a set of standardized parts or components comes along, triggering a wave of experimentation by innovators who tinker with the many *combinations* of these components. The result: a wealth of new systems built on the newly available components or by recombining existing components. Some of these systems are novel even to the designer of the component!

In the following sections we will explore the emergence and implications of stacks at the software, enterprise, and industry level. It is our contention that similar conditions both result in the formation of stacks and are a consequence of the emergence of stacks. Stacks result from a combination of both purposeful design and emergent response to dynamic environments.

Where do stacks come from?

The systems that we are interested in – information systems, organizations, and industries – share a common set of characteristics that lead to the emergence and desirability of stacks. They are all goal (individual, organizational, or societal) directed, complex systems that mediate the gap between core capabilities and resources (raw materials, skills, knowledge) and the expectations of stakeholders (consumers, users, managers, shareholders, etc.). The challenge is, of course, that the raw materials and core capabilities change, expectations across stakeholders are divergent and changing, and our understanding of both the starting point (the core capabilities and resources) and the end point (the expectations) is poorly specified. Nevertheless, the existence of information systems, successful companies, and a robust economy are a testament to our success at doing a fairly reasonable job at building these complex systems.

There are a limited number of forces that lead to the emergence and desirability of stacks within these complex systems: complexity under the constraint of bounded rationality (Simon 2003), a dynamic environment, incremental innovation, and an installed base. The first two forces lead to modularity, but not necessarily stacks. Given that we can only understand so much, it is easier to create a system with stable components that hide much of the detail instead of dealing with all the detail at once (Simon 2003). If the environment was stable, a monolithic system might emerge in spite of our cognitive limits. However, in the presence of change, complexity leads to modular, stable sub-systems. Environmental change (e.g., a changed user requirement or new government regulation) may require a change to a module but not the entire system.

The second two forces lead to hierarchy – stacks. If we start with the notion of a gap between capability and expectations, and we assume that we can't provide a complete solution as the first step (because we may not even know what the final solution is), we are left with incremental change to a base system. The base system must start with the current resources and capabilities. Incremental innovation builds upon that which exists, creating hierarchy from the bottom of the stack upwards. As innovation takes place and multiple new modules utilize an existing component, the need to continue to support existing modules – the installed base – limits the ability of existing components to change. This creates a stratification and ossification of layers within the stack. The interfaces and functionality of certain layers become, in effect, dominant designs (Utterback et al. 1975) that are infrequently replaced. The emergence of certain dominant designs results in lower production costs for the system because uncertainty is reduced.

The language of stacks

To describe and understand emergent, modular stacks, we adopt and augment the language provided Baldwin and Clark (Baldwin et al. 2000). They define six core modularity operators: splitting, substituting, augmenting, excluding, inverting and porting. *Splitting* takes a single component from the system and converts it into hierarchical design with a core set of independent modules. *Substituting* replaces an existing component with a new component that performs the same operations as the replaced component. *Augmenting* means adding a module and excluding means leaving one out. *Inversion* makes public previously hidden information about a module. This can occur when multiple modules implement common functionality. *Inversion* combines the common functions into a new module that the existing modules can utilize. *Porting* is the property of a system that permits it to be mapped from one context or infrastructure instance (operating systems, processor architectures, programming language, or development environment) to a different context or infrastructure instance.

To the six operators previously defined we add an additional operator, *linking*. The linking operator connects two previously unconnected modules. In developing information systems a key consideration is on using outputs of one system as inputs to another.

In order to explain stacks we add two prefixes to the *inversion* operator -- *Down Layer* and *Up Layer*. These stack operators refer to the movement of modules across stack layers or the creation of new ones. Moving a module *Down Layer* involves the movement of a module into a lower layer of the stack. This is quite commonplace in the IT industry. Examples include the movement of virtual addressing from the operating system to hardware, print functions from applications to operating systems, and record management from applications to database management systems. A recent example within the software industry is the creation of an entire category of software called virtualization software that has been inserted between the hardware and the operating system. In moving a module *Down Layer*, the module becomes more widely available to a greater number of modules at the previous and higher layers.

The *Up Layer* operator refers to the process of moving a module from a lower layer to an upper layer. The need to maintain backward compatibility makes this a less frequent operation. Maintaining backward compatibility comes at a cost, however, so system designers are motivated to remove unnecessary functionality. Examples of *Up Layering* include the elimination of real-addressing modes from Windows. It may still exist, but only through an application-layer emulation package. The presence of middleware is also a manifestation of this operator. When many programs make a set of APIs available for applications in the layer above to use, it becomes very cumbersome for application developers to write applications. As a result, a vendor creates a toolkit to support speedier development. Examples of this include Microsoft's XNA toolkit for developing video game applications for the Xbox, Yahoo!'s software development toolkit (SDK) to allow developers to use their search engine, Webshpere SDK to write e-business applications or Java 2 Platform, Standard Edition (J2SE) that provides a complete environment for applications development on desktops and servers and for deployment in embedded environments.

Businesses, too, move functions into and out of their core (lower stack layer) set of capabilities (see, for example, (Landes 1983)). For example, some firms have moved their logistics logic from within their supervision to third parties such as UPS and FedEx. Similarly, some firms have moved their call centers from within to third party providers.

The language of dependency

So far, we have described an abstraction in which complex systems emerge as a stack of modules and a language that describes how change occurs to this stack. Modules are created, modified, and relocated through a variety of operations. Such operations occur incrementally and, in the presence of a dynamic environment, cognitive limits, and the need to support an installed base, cause complex systems to evolve into stacks. We have described the building blocks without describing the glue that holds them together. The modules that form stacks are connected to each other through communication and interdependency.

Communication can be a bit-stream (Shannon 1948), story-telling (Brown et al. 1991), printed documents, apprenticeships, contracts (Brynjolfsson 1994) or market-based transactions (Williamson 1975). Communication occurs between modules of a software system, within a firm, and between firms within and across industries. Interdependency includes understanding the communication, maintenance of a common interface, availability, timeliness, and performance. We can characterize links between modules in terms of the frequency of communication, the volume of communication, and the stickiness (Teece 1992; von Hippel 1994) of communication. We can characterize the dependency between two modules in terms of their coupling (degree of looseness) (Weick 1976). In a tightly coupled or pooled dependency (industry) system, in which two firms are highly interdependent, we may see vertical integration between the two firms (Brusoni et al. 2001). In a loosely coupled or sequential relationship, the firms may coordinate through the market.

In the next sections we explore the stack concept and its evolution through examples from software systems, organizational architecture, and industry structure.

Software Architecture

Within a software system, the core, base layer is the computer hardware. Between the capabilities of the physical hardware (the computer), and the requirements of the user, is a gap. Of course, the user could use the computer hardware directly, but it is generally too difficult for a user to express her requirements in the form of a machine code, and interpret the resulting zeros and ones in a meaningful way. That is, the user working directly with the machine could fill the gap (as was done in the early days of computing, before the development of operating systems, compilers, and other utilities). Since that time, greatly improved computer hardware economics have enabled software layers to provide flexibility, domain-specific applications, and software developer tools that make it easier to satisfy a growing set of user requirements. Multiple layers of the software stack now hide the computer's physical hardware.

In the case of software, layering (the creation of stacks) reflects a division of the functions implemented in software into units, generally called virtual machines, in which each unit provides a cohesive set of services that software products in other layers can utilize without knowing how these services are implemented. These units, or layers, should interact with each other according to a strict ordering relation. These relations are usually represented as a stack, in which each layer is allowed to use only the functionality in the lower layers. In the case of strict layering, the upper layers can only call the layers immediately below them. The lowest layers are usually built using knowledge of specific computers, communications channels, distribution mechanisms, process dispatchers, etc, and are independent of the applications that may run on them. Higher layers utilize the facilities of lower layers and are more independent of the hardware in which they work, because the existence of lower layers permit them to be so. This means that higher layers don't have to change if there is a change in the computing platform or environment. A change in a lower layer that does not affect the interface used will require no change in higher layers. Also, a change in a higher layer that does not change the facilities required will not affect lower layers.

Developers of software focus on one or a few layers and rely on other developers to provide the other layers. With well-defined and published APIs (Application Programming Interfaces), multiple vendors can supply components that interoperate across layers of the stack. As a result, users can experiment and create a much greater variety of systems than was possible prior to the emergence of the software stack.

Enterprise Architecture

Enterprise architecture is the decomposition of the enterprise information system into manageable parts, the definition of what those parts are, and the orchestration of the interplay among those parts. One can also view it as the value chain for the production of information within the enterprise. At one end of the value chain are sensors that collect the data that is relevant to the enterprise. At the other end are the applications that present the data to the decision maker. In between are layers that help process the data to generate information.

Recent technologies such as middleware, business logic components, and Web Services have extended the middle layers of the stack, providing higher levels of application interoperability than were available previously. Applications and services draw upon layers of more loosely coupled, broadly available technologies that enable wider integration within organizations (e.g., SAP, PeopleSoft, and Oracle) and between organizations (e.g., extranets and XML services).

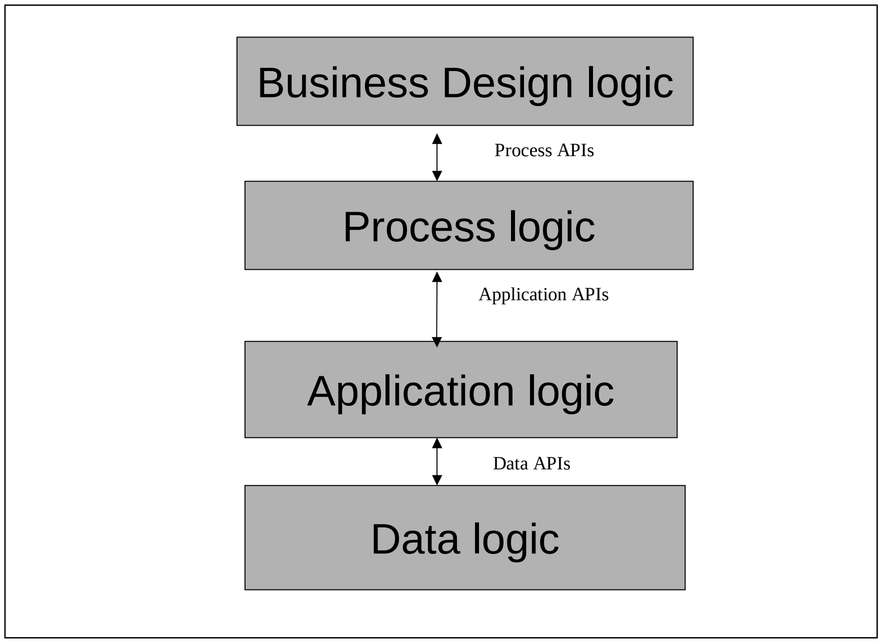

Figure 1. The Enterprise Stack

In addition to the stack including layers that provide greater application interoperability, we have also seen a layering of application logic into what is now called multi-tiered architectures. In this type of design, data, application, process, and business design logic is separated and supported by different software layers (see Figure 1). In the language we described earlier, application modules have *split* to create hierarchy. The overall architecture has been *augmented* as new functions have been added. Functions formerly hidden within applications (e.g. inter-process communication) have been *inverted* to create new, visible modules. These modules have been *down-layered* to create message-oriented computing (e.g., J2EE).

As these layers have been created, dependencies between functions have been reduced. The coupling between layers has become looser. Ordinarily, we would expect the performance to decrease as well. And, if measured as a unit of communication per unit of computing, it would have. Fortunately, units of computing become less expensive more quickly than styles of communication consume them.

One of the often overlooked challenges of moving to the more loosely coupled, inter-organizational styles of computing are assumptions as to what is communicated and how well all parties in the communication understand the message. In general, loose coupling also favors less sticky communication (von Hippel 1994).

If we take an information processing view of the firm (Galbraith 1974), and view the enterprise architecture as a metaphor for organizational architecture, many of the same trends, opportunities, and challenges remain salient.

Organizational Architecture

From a stack-based view, organizations exist to create value for stakeholders from the raw materials, capabilities, and knowledge it has access to. Questions of organizational structure, boundaries of the firm, and social networks are questions of what the modules are, what the stack layers are, and what the communication links between units are. Going back to the early 20th century, we can see the development of Scientific Management (Taylor 1911) as the creation of layers in which task design is separated from task performance. Similarly, Thompson's (Thompson 1967) concept of protecting the core capabilities and Chandler's (Chandler 1962) description of the multi-divisional structure are also about the creation of stacks.

We can recast the debate between centralization and decentralization as one regarding the number of levels in the enterprise stack and the tightness of the coupling between functions both within and across levels of the enterprise. Firms must specialize and integrate; they must coordinate at a high level and yet delegate decisions to where the knowledge is. When a firm centralizes, it is collapsing the stack; it is increasing the tightness of coupling across the functions contained in the collapsing layers. In some cases, layers of the stack are thinned or augmented by IT. For example, the diffusion of computers and intranets within organizations may reduce the number of people in a reporting layer within the organization. When a firm decentralizes, it is increasing the number of layers in the stack or *down layering* functions to a lower level of the stack.

Recently, due to the availability of technologies such as components and Web Services, enterprises are investigating approaches to modeling business processes that match similar approaches in software application development strategies and take advantage of changes in software and continuously decreasing costs of computing. One such approach – using loosely coupled, modular components is compared to "LEGO blocks." Businesses can combine, disassemble and recombine basic business functions to create specific business processes to satisfy different business objectives. This approach has encouraged and resulted from developments in tools that help a business separate its applications into four domains: process logic, application logic, business rules and data (see Figure 1).

A business process is a complete coordinated thread of all the serial and parallel activities needed to deliver value to the enterprise's customers (Davenport 2005). Traditionally, enterprises used software applications to support coordination and execution of business processes. Although these applications were reliable, they were not built to be very flexible, agile, or transparent because they combined and tightly coupled the various assumptions about processes, data, how the business functions and how the applications are designed. This made it very difficult to locate and make changes to any one of the assumptions.

The increased modularity and layering of business processes can be connected to improved understanding of the business processes (e.g., an increase in explicit knowledge), a dynamic environment, and incremental innovation in business processes while supporting the installed base of existing business activity. Increasing the modularity of the business processes requires making the information exchanged between units less sticky and changes to the frequency and quantity of the data exchanged. One of the enabling infrastructures supporting these changes has been corresponding change in the IT systems. Structured, flexible, and high-performance communication has enabled more loosely coupled organizations (Argyres 1995).

Industry Stack

The logic that drives stack creation within information systems and organizational structure also drives stack creation within industries. There are a number of theories that explain why firms exist and what limits there size (Barnard 1938; Daft et al. 1984; Milgrom et al. 1990; Nelson et al. 1982; Taylor 1911; Williamson 1975). Our argument is that one of fundamental reasons firms exist is due to our bounded rationality within a dynamic environment. Firms hide the complexity of building products and delivering services and present a simple interface to potential consumers, suppliers, and partners. The interdependencies within the firm are greater than the interdependencies between elements (people and non-human resources) of the firm and elements outside the firm. Within our framework, firms are modules within an economic system.

As firms enter the economic system, they access raw materials directly or they build upon the outputs of existing firms. Thus, they incrementally add to the existing economic system. Vendors that supply firms must maintain a certain backward compatibility because firms innovate or enter the economy at different rates. This gives rise to the creation of layers – stack.

Within the IT industry (which we will use as our example), the industry stack mirrors the application stacks that are used within the enterprise. At one end are the hardware and infrastructure firms that form the foundation for the services that are provided at the other end of the value chain.

In the early days the industry had a set of vertically integrated companies producing everything that a consumer needed (e,g., DEC, IBM and Wang). As described by Andy Grove (Grove 1996), somewhere around the late 80's a transition from vertical integration to horizontal layers occurred. As a result of this transition, we moved from a single firm offering end-to-end services to modular clusters (Baldwin et al. 2000) or stacks populated by specialist firms (West et al. 2001).

The industry stack divides activities into layers that support each other, as depicted in Figure 2. Today, just as was the case during the era of vertical integration, firms can deliver products that support most (if not all) layers of the stack. For example, consumers can buy chipsets, assembled computers, operating systems (AIX), middleware (Websphere), applications (CRM), and services (Global consulting) from IBM. The main difference in the era of stacks is that IBM provides these products with loose coupling and with open interfaces between them. As a result, consumers of these services have the option to mix and match IBM’s products with those provided by other vendors. In the earlier era, this was not possible – a consumer had to pick a vendor and buy all required services from them. According to Lou Gerstner, former CEO of IBM, most companies specialize in one or a few layers and rely on other companies to offer complementary components. Each of these components is layered above the other, and communicates through more or less standard interfaces, with closer layers being more related to each other than layers that are further apart on the stack.

Lower layers and their components, such as hardware and network services, are often referred to as operating platforms and are fast becoming commodities. They have well defined interfaces with well-defined terms of trade (prices). Firms build competencies on top of these lower layers by carefully selecting middleware and application packages and then launch business services on top of the application layer. The top layer contains all the service firms that provide services such as integration, training and maintenance. This layer includes consulting companies such as Accenture, IBM, Infosys and TCS. Middleware providers include firms such as IBM (with Websphere), BEA systems (with Weblogic) and database vendors such as Oracle. In the application software layer one will find firms such as SAP, Peoplesoft (now part of Oracle), Siebel systems and others. The business services layer includes companies that provide software as a service such as Salesforce.com and Google (e.g., Google Earth).

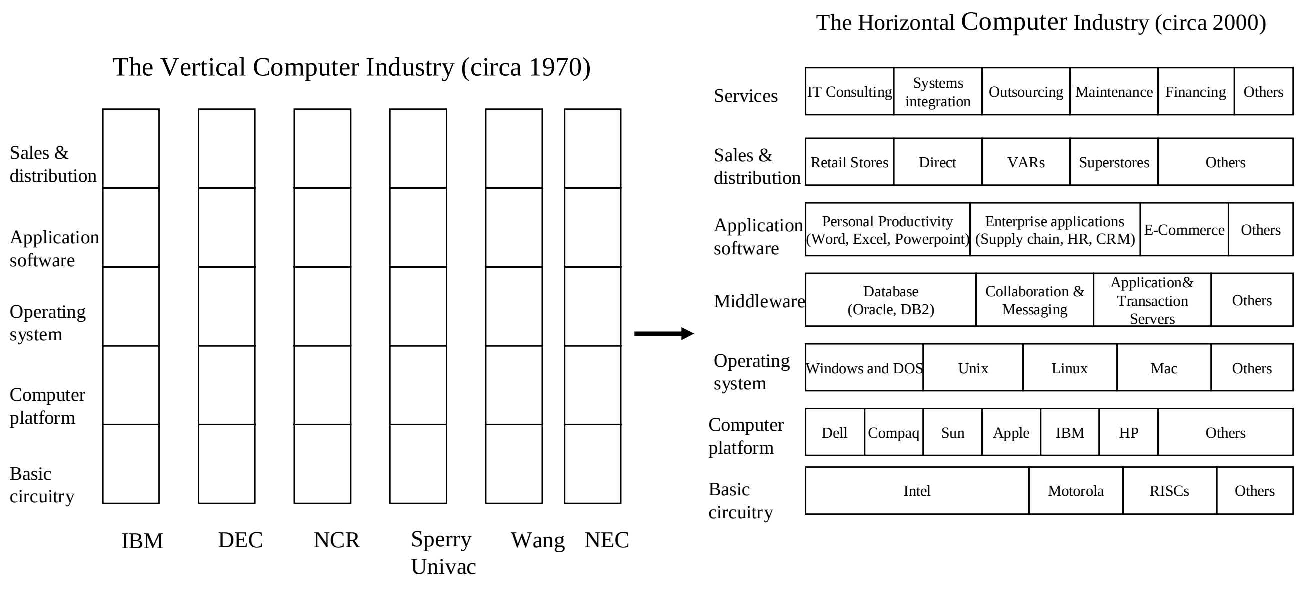

Figure 2: From vertical to horizontal transition

The change from vertical platforms in which the industry had few layers but the underlying software had many layers to one in which the industry matched the software components in terms of layering can be described by the design operators we previously described. New firms (modules) *augmented* the existing economic system. Hierarchies *split* to form new modules in which the new modules *inverted* to become visible (e.g., database vendors emerged as a separate set of companies). Some companies were moved *down layer* as hardware and operating systems acquired vendors of products that were tightly integrated into the core functionality of the respective layers.

One of the consequences of the change in the economic structure of the industry – changes in the stack – is that some companies fail, some limp along, some enter, and some thrive. The existence of stack within the IT industry is both the result of incremental innovation within a dynamic environment and the enabler of innovation. New firms do not need to provide an entire platform in order to add value; they can build on the output (layers) of others.

Another consequence of the change to the industry stack is the nature of communication – the coupling – between modules has changed. In place of the tight coupling that existed within vertically integrated firms, we now have looser coupling through partnerships, alliances, and standard setting committees (Brusoni et al. 2001).

Trade-offs

The granularity of components and the direction and nature of their interdependency is motivated by four sets of interdependent tradeoffs: flexibility vs. efficiency, domain specificity vs. domain generalizability, system performance vs. user performance; and tight coupling vs. loose coupling. Because the nature of the stack is defined by the size of the modules and the nature of their coupling, these trade-offs directly influence the emergence of particular stacks. Implicit in each pair of trade-offs are cost considerations.

The classic trade-off is between efficiency and flexibility (Milgrom et al. 1990). An efficient system is one in which the fewest resources are utilized to produce a given output. In systems with multiple resources, the most efficient system is one that uses the fewest resources in aggregate. If we convert all resources to currency in order to aid comparison, the most efficient system is the one that costs the least.

However, systems in a dynamic environment are subject to change. Thus, the objective function we hope to minimize, cost, changes depending upon our time horizon. We can measure the cost of change as flexibility. Unfortunately, flexibility often comes at a price. There may be upfront investments required to enable long-term flexibility, or there may be operational inefficiencies because certain upfront investments in efficiency have not been made (Utterback et al. 1975). As we build systems or design strategies we must balance short-term efficiency objectives against long-term flexibility objectives.

Investments in efficiency tend to increase asset specificity (Williamson 1975) and, thus, increase potential transaction costs. Investments in efficiency may also increase coordination costs (Malone et al. 1994; Thompson 1967) as slack is eliminated from the system. Such investments may increase the cohesion between modular units to the point where the modular units are effectively merged.

In volatile environments, such as ones with uncertain customer preferences, rapidly changing production functions, or volatile factor markets, flexibility may be more important than efficiency. In this situation, we expect a higher level of modularity with fewer interdependencies between modules. The computer industry is subject to rapid changes in customer preferences (e.g., changes from mainframe computing, to client-server, to N-tiered, to internet), changing production functions (e.g., improvements in price/performance in semiconductors and peripherals have been exponential), and factor markets (e.g., the cost of programmers, software, and hardware). As a result, we have the industry pictured in Figure 2 as described by Bresnahan (Bresnahan 1998).

The second set of trade-offs we identified is domain specificity versus domain generalizability. A domain specific application is one in which knowledge about how the application will be used, or how the customer will utilize the firm’s product or service, is important in the development of the application, product or service. For example, an accounting system is domain specific relative to a spreadsheet application. Although both can be used to sum numbers, the accounting system is built with accounting rules internalized, while the spreadsheet application requires that the user create these rules.

Domain specificity represents an investment by the module’s producer in internalizing physical or intellectual property, and then selling or otherwise provisioning the resulting product or service to clients or customers. To the extent that the resulting product or service represents value to the customer, and such value builds upon the previously existing stack, we can say that domain specificity adds height to the stack (it adds a layer). Domain specificity is subject to the rules of asset-specific investments (Williamson 1975). As a result, we expect such investments to increase when the cost of asset specific investments decreases. An example of this is the explosion in applications developed for database systems such as Informix, Oracle, Foxbase, dBase, etc. These products reduced the cost of developing domain-specific applications by providing general record management and application generators.

Within the software industry, both vendors and customers make asset specific investments. For vendors, the investment is in developing for a specific vertical market. For customers, the investment is in utilizing a specific vendors' product or service. In a volatile environment, we expect that such asset specific investments will be minimized as long as uncertainty regarding production functionality and vendor stability is high. As a result, we expect the coupling between the products and services of new entrants and existing modules on the stack to be looser than that which exists between more established modules. Moreover, we expect greater volatility in terms of firm and product entrance towards the top of the stack (value migrates upwards).

The third set of trade-offs is between system performance and user performance. If our objective is to produce a given output for the lowest cost (efficiency) over time (flexibility), and we conceptualize the cost function as consisting of production costs (i.e., the cost of providing the product or service) and consumption costs (i.e., the costs of utilizing the product or service), and capital costs and labor costs, then we have four variables driving total cost. Within the computer industry the trade-off is between hardware costs and application development time. Managers can choose to purchase hardware to overcome limits in application development, or to invest in application development to make more efficient use of hardware. When hardware was expensive (i.e., pre-1980), considerable investments were made in optimizing the performance of applications. As the price of computing has dropped, the emphasis has been on programmer and end-user productivity. The graphic interface we use today is computationally expensive as measured in instruction cycles, but the computations are extraordinarily inexpensive as measured in monetary terms.

As the cost of computing decreases, we expect greater modularity and looser coupling at lower levels of the stack in order to enable greater flexibility at higher levels where the user is more involved. Moreover, we expect additional layers of stack as the cost of flexibility - measured in computing cycles - falls.

The trade-off between consumption and production costs (where the consumer is the user and the producer is the system, in our original conceptualization) is affected both by the changing economics of each part (production or consumption) and by the interaction between the two. As the system internalizes consumption costs, the stack grows. For example, the development of compilers can be seen as the internalization of the cost of writing machine code, formerly performed by consumers. The internalization of consumption costs depends upon a more explicit understanding of what is required (von Hippel 1994) and small-numbers bargaining (Williamson 1975). Experience with products at existing layers of the stack results in a better understanding of how they can be utilized. Such knowledge can then be encoded in future products, adding layers to the stack. As the number of consumers for a product or product-class increases, threats of opportunism are reduced, enabling the asset-specific investments required for the production of new products.

The fourth trade-off is between tight and loose coupling (Brusoni et al. 2001; Weick 1976). The tightness or looseness of the coupling between two modules is a measure of the level of those modules interdependence. If two modules are tightly coupled, then the failure of one causes failure in the other. In a loosely coupled system, the failure of one module may have no impact on the other. Messaging, clustering, and database products serve to isolate (reduce the coupling) between components of an information system.

We can characterize interdependence along a number of dimensions: availability, data volume, data width, semantics, and coordination mechanism (Malone et al. 1994). Availability is a measure of the dependency of one system on the existence or functioning of another. Data volume is a measure of the dependency of one system on another systems ability to handle a certain volume of data or transactions. Data width is a measure of the complexity of the data exchanged between two systems. Semantic dependency is a measure of the degree to which one system depends upon the other system having the same understanding regarding the meaning of the data. Coordination dependency is a measure of the extent to which one system depends upon the performance of the other in executing its coordination activities.

Tightly coupled systems have higher coordination costs. These costs may be balanced against greater efficiency or the reduction in other costs. Due to the higher coordination costs, we expect tightly-coupled stacks to evolve more slowly, with few modules, and fewer layers. More loosely coupled systems have lower coordination costs and lower efficiency, but greater flexibility.

Implications

Information System, organizational, and industry architectures are key determinants of a firm’s success. Once an architectural decision is made, it has implications on how easy it is to change, the division of development resources within and outside the firm, the ability to achieve certain performance levels and the way the organization is structured. Firms have substantial latitude in choosing their architecture and it is an important managerial decision.

We think it valuable to consider organizational and IT architecture alignment as the alignment of two stacks, and then explore their alignment within the context of the industry stack. While firms cannot predict with certainty the emergence of a particular dominant design within the industry stack, we believe they can influence it to their advantage and design their internal architecture to take advantage of its emergence. We call the process of influencing emergence to their advantage as exercising architectural control.

The process of exercising control is a dynamic process and is not well understood. Therefore, we’d like to offer a framing (stacks) and a set of insights that might guide decision makers (architects) as they consider the various tradeoffs. Even if we can’t guarantee architectural control, we can at least provide some insight into the (emergent) rules of the game.

The first insight is that there are three types of stacks that an architect needs to be aware of, and over which she has varying degrees of control (or influence). Furthermore, these three stacks are motivated by the same internal logic: they are modular designs that emerge as a response to complexity, they fill a gap between core capabilities and stakeholder requirements, and they preserve backward compatibility while supporting innovation and asymmetric rates of change in underlying components. Firms within the industry are organized as a stack in which the lower layers of the stack provide services that are utilized by higher levels. Companies are organized as a stack in which core capabilities are buffered by functions closer to the customer. IT systems are organized as a stack where hardware services are abstracted into virtual machines supporting layers of shared, general purpose, and domain specific services.

The second insight is that these three stacks – industry, enterprise, IT system – are interdependent. At its simplest, we can say that the IT system (stack) supports the enterprise (stack) that is operating within the industry (stack). We can also say that the industry stack influences the emergence of each enterprise stack which, in turn, influences the emergence of a company's IT stack. Neither set of influences is deterministic; managers have choice and influence (though not control) over the emergent behavior.

The third insight is that each stack represents a path-dependent (historically constrained) snapshot of the set of tradeoffs made by multiple, interdependent architects and managers. These trade-offs include concepts such as: efficiency and flexibility, domain-specific solutions and generalized systems, capital productivity and labor productivity, and tight coupling and loose coupling. These tradeoffs are adjusted as underlying technologies, user demands, competitive moves, and knowledge change.

Our fourth insight is that over time value migrates up the stack. Layers within the stack become commoditized as users and vendors are able to understand and articulate their requirements. In other words, products become commodities as uncertainty is reduced and dominant designs emerge. As a consequence of subsequent competition, prices decrease and the value of the design to the vendor drops. Commoditization of lower layers, however, enables incremental innovation in higher layers because of a high degree of understanding and reuse of the lower layer functionality. This type of incremental innovation is typical in stack-oriented industries. Since the incremental innovation addresses previously unmet user needs, users are willing to pay a premium for them – in other words, the value of design to the vendor migrates from the lower layers to the upper ones.

Moreover, there are two competitive moves vendors can make to appropriate value from stacks. First, as described above, vendors can introduce new modules that satisfy unmet user needs. Second, vendors can insert and establish new layers by soliciting other developers to write to that layer. If successful, the new layer becomes a new platform (Gawer et al. 2002). One such example of this is the creation of middleware. The defensive or preemptive move a vendor can make is to incorporate the nascent platform into its own product line. One such example of this is Microsoft's decision to embed the Internet browser into its operating system, to the detriment of other independent vendors of browsers.

One of the implications of the emergence of modules and dominant designs is that stacks (architectures) can (and sometimes must) change in response to them. Once a set of activities or decision making processes is understood well-enough, domain specific languages and applications can be developed, adding to the stack. As the speed of performing some activity is increased (whether due to new insights in physical materials, hardware design, algorithms, requirements, or standards) the various tradeoffs we've been discussing can be adjusted.

Conclusion

We have suggested that stacks are everywhere: they exist in IT systems, businesses, and within industries. Stacks are an inevitable consequence of physical and biological laws: people have limited cognitive ability, modular systems emerge to manage complexity, and the components of systems develop at differential rates requiring some level of backward compatibility (history matters). There are tradeoffs in the development of stacks: flexibility and efficiency, specialization and generalizability, user and system, and tight and loose coupling. Stacks are interdependent: the industry, enterprise, and IT stacks influence each other. Knowing is an emergent process: what we know, how we know it, and the contextual dependency of the knowledge change over time and influence our stack design choices.

We believe that stack-based thinking provides an explanation as to what happens within technology, businesses, and industry. That is, stacks have descriptive value. Moreover, stacks create a design platform in which firms can innovate and experiment at a lower cost that would otherwise be possible.

Future research will empirically examine the value of stack-based thinking for managers and decision makers. Do managers that appreciate stacks and their tradeoffs perform better than those that do not? Does stack-based thinking open up new ways of thinking, seeing, and discussing that result in better decisions?

When considering alignment, do the systems stack and the enterprise-stack need to be isomorphic? Do both stacks need to have the same level of loose and tight coupling? Modularity may make it possible to better manage complexity, but modularity theory may also help us understand the limits of design and the limits of the modularization process.

As managers look for ways to take ideas and create value in the economy, stacks provide a valuable design abstraction that enables them to articulate and enact their vision. Within the computing industry there are many documented stories of firm success and failures that can be explained using the stack analogy. We believe that stacks can also be used as an abstraction to understand change within other industries such as financial services, healthcare, or manufacturing.

References

Ahuja, G. "The duality of collaboration: inducements and opportunities in the formation of interfirm linkages," *Strategic Management Journal* (21:3) 2000, pp 317-343.

Argyres, N.S. "Technology strategy, governance structure and interdivisional coordination," *Journal of Economic Behavior and Organization* (28) 1995, pp 337-358.

Baldwin, C.Y., and Clark, K.B. *Design Rules: The Power of Modularity* MIT Press, Cambridge, Mass., 2000, p. v. <1 >.

Baldwin, C.Y., and Clark, K.B. "Between 'Knowledge' and the 'Economy': Notes on the Scientific Study of Designs," *Working paper*), May 2005.

Barnard, C.I. *The Functions of the Executive*, (30th Anniversary Edition, 1968 ed.) Harvard University Press, Cambridge, MA, 1938, p. 334.

Bresnahan, T. "New Modes of Competition: Implications for the Future Structure of the Computer Industry," 1998.

Bresnahan, T.F., and Greenstein, S. "Technological competition and the structure of the computer industry," *Journal of Industrial Economics* (47:1), Mar 1999, pp 1-40.

Brown, J.S., and Duguid, P. "Organizational Learning and Communities-of-Practice: Toward a Unified View of Working, Learning, and Innovating," INFORMS: Institute for Operations Research, 1991, p. 40.

Brusoni, S., Prencipe, A., and Pavitt, K. "Knowledge Specialization, Organizational Coupling, and the Boundaries of the Frim: Why Do Firms Know More Than They Make?," *Administrative Science Quarterly* (46) 2001, pp 597-621.

Brynjolfsson, E. "Information Assets, Technology, and Organization," *Management Science* (40:12), Dec 1994, pp 1645-1663.

Campbell-Kelly, M. *From airline reservations to Sonic the Hedgehog : a history of the software industry* MIT Press, Cambridge, Mass., 2003, pp. xiv, 372 p.

Chandler, A.D., Jr. *Strategy and Structure: Chapters in the History of the American Industrial Enterprise* MIT Press, Cambridge, MA, 1962, p. 463.

Daft, R.L., and Weick, K.E. "Toward a model of organizations as interpretation systems," *Academy of Management Review* (9:2) 1984.

Davenport, T. "The Coming Commodotization of Process," *Harvard Business Review* (83:6) 2005, pp 100-108.

Galbraith, J.R. "Organization design: An information processing view," *Interfaces* (4:5) 1974.

Gawer, A., and Cusumano, M. *Platform Leadership* Harvard Business School Press, Boston, Massachusetts, 2002, p. 305.

Grant, R., and Baden-Fuller, C. "A Knowledge Accessing Theory of Strategic Alliances," *Journal of Management Studies* (41:1) 2004, pp 61-84.

Grove, A. *Only the Paranoid Survive* Doubleday, New York, NY, 1996.

Gulati, R., and Singh, H. "The architecture of cooperation: Managing coordination costs and appropriation concerns in strategic alliances," *Administrative Science Quarterly* (43:4), Dec 1998, pp 781-814.

Hopkins, J. "Component Primer," *Communications of the ACM* (43:10), Oct 2000, pp 27-30.

Iyer, B., and Gottlieb, R.M. "The Four-Domain Architecture: An approach to support enterprise architecture design.," in: *IBM Systems Journal*, IBM Corporation/IBM Journals, 2004, pp. 587-597.

Kayworth, T.R., Chatterjee, D., and Sambamurthy, V. "Theoretical Justification for IT Infrastructure Investments," *Information Resources Management Journal* (14:3), July-Sept 2001 2001, pp 5--14.

Landes, D.S. *Revolution in time : clocks and the making of the modern world* Belknap Press of Harvard University Press, Cambridge, Mass., 1983, pp. xviii, 482 , [440] of plates.

Malone, T.W., and Crowston, K. "The Interdisciplinary Study of Coordination," *ACM Computing Surveys* (26:1), MAR 1994, pp 87-119.

Markides, C., and Williamson, P. "Related Diversification, Core Competences and Corporate Performance," *Strategic Management Journal* (15:special issue) 1994, pp 149-166.

Messerschmitt, D.G., and Szyperski, C. *Software ecosystem: understanding an indispensable technology and industry* MIT Press, Cambridge, Mass., 2003, pp. xiv, 424 p.

Milgrom, P., and Roberts, J. "The Economics of Modern Manufacturing: Technology, Strategy, and Organization," *American Economic Review* (80:3) 1990, pp 511-528.

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

Shannon, C.E. "A Mathematical Theory of Communication," *The Bell System Technical Journal* (27), Jul/Oct 1948, pp 379-423.

Shapiro, C., and Varian, H.R. *Information Rules: a strategic guide to the network economy*, (1 ed.) Harvard Business School Press, Boston, 1999, p. 352.

Simon, H. "The architecture of complexity," R. Garud, A. Kumaraswamy and R.N. Langlois (eds.), Blackwell, Malden, MA, 2003, pp. viii, 411 p.

Simon, H.A. *The Sciences of the Artificial,* (Third ed.) The MIT Press, Cambridge, MA, 1996 [1969], p. 231.

Taylor, F.W. *The Principles of Scientific Management* Engineering & Management Press, Institute of Industrial Engineers, Norcross, GA, 1911, p. 131.

Teece, D.J. "Competition, cooperation, and innovation: organizational arrangements for regimes of rapid technological progress," *Journal of Economic Behavior and Organization* (18:1) 1992, pp 1-25.

Thompson, J. *Organizations in Action* McGraw-Hill, New York, 1967.

Utterback, J., and Abernathy, W. "A dynamic model of product and process innovation," *Omega* (3) 1975, pp 639-656.

von Hippel, E. ""Sticky Information" and the locus of problem solving: Implications for Innovation," *Management Science* (40:4), April 1994, pp 429-439.

Weick, K.E. "Educational Organizations as Loosely Coupled Systems," *Administrative Science Quarterly* (21) 1976.

West, J., and Dedrick, J. "Open Source Standardization: The Rise of Linux in the Network Era," *Knowledge, Technology & Policy* (14:2) 2001, pp 88-112.

Williamson, O.E. *Markets and Hierarchies: Analysis and Antitrust Implications* The Free Press, New York, 1975, p. 286.