Architecture in Design Science: The Case of Stacks

David Dreyfus
Bala Iyer
Department of Management Information Systems
Boston University
{bala, ddreyfus}@bu.edu

Abstract

Design creates value by turning resources into things that people use. Within the field of information systems, design has been studied by researchers practicing under the design science paradigm. One component of design -- architecture -- is the focus of our study. Research on architecture is often prescriptive. Although architects and planners have an espoused view of architecture (often based on prescriptive research), implementation and modification projects result in an emerged architecture that may differ from the espoused view. Research on emergent architecture is descriptive.

In this paper, we elaborate on an architectural abstraction called stacks. Stacks can emerge at the systems, enterprise, organizational, and industry level. In our model, stacks, and the operators used to modify them, are represented identically. The model and operators form a language that can be used to describe and understand architectural emergence. To accomplish this, we follow the norms of design science and compose our problem statement by basing it on the technological, organizational, and strategic literatures. Our contribution is to both the behavioral and design science literature through the creation and analysis of our theories and model.

Introduction

The literature contains a number of potentially confusing research streams that overlap in terms, concepts, and phenomena – design (Dryer 1999; Galbraith 1974; Ware 2000; Watson et al. 2004), science of design (Alexander 1964; Baldwin et al. 2000;

DESRIST 2006. February 24-25, 2006, Claremont, CA.

Baldwin et al. 2004; Baldwin et al. 2005; Boland 2002; Churchman 1971; Simon 1996 [1969]), and design science (Hevner et al. 2004) – and the distinction between design theories (Markus et al. 2002), designs as architecture, and finished artifacts (Iyer et al. 2004b). In the first category, we have designs and the research in how to create better designs. This research is largely prescriptive in that it seeks better ways of designing artifacts. The science of design is often concerned with design as descriptive and with the affordances of design. From a science of design perspective, design is both a natural and social phenomena.

Design science is a research paradigm in which designers (builders of systems) and social scientists develop insights and knowledge through interaction with artifacts. The artifact is an occasion for both independent and joint learning. Social scientists and scientists of design provide descriptive knowledge of the nature of social and social-technical interactions. Designers of systems incorporate this knowledge as they build new systems and develop new insights into the process of designing and building systems. During the process of evaluating the resulting artifact, new insights and knowledge are developed by both social scientists and designers of systems.

Design science has its philosophical roots in the work of Emanuel Kant in which the physical world cannot be observed directly because it is mediated by our minds. Thus, we cannot make claims about an objective reality; we can only make claims regarding the system of theories we develop regarding the objective reality. Design science takes this one step further and argues that design science is pursued with a purpose: utility. The search for utility replaces the search for Truth in positivist research.

Design science can more immediately trace its roots back to engineering disciplines and *The Sciences of the Artificial* (Simon 1996 [1969]). Because design science seeks greater utility, research is oriented around the construction of new artifacts. However, design science is not just about creating and evaluating new instances of systems; it is about building on, and contributing to, social, technical, and design theories in order to produce better designs. Following the work of Simon, design science is concerned with goal-directed search in which we seek better ways of organizing internal resources in order to achieve greater utility. Design occurs within social and technological systems, and at multiple levels of analysis (Boland 2002).

The process of developing and building new artifacts can result in progress in a number of critical research areas. Certainly, better artifacts can result in greater utility. However, we can also develop theories of design in which we develop new approaches to new types of problems (Churchman 1971; Markus et al. 2002). We can develop specific methodologies, methods, and techniques. We can create and evaluate new algorithms, models, and data representations. Not only can we develop and evaluate new physical artifacts, the designs that lead to artifacts can also be evaluated.

Design has been described as the set of instructions based on knowledge that turns resources into things, products and services, that people value and use (Baldwin et al. 2005). Typically, a design process is purposeful: it starts with a problem and ends with an artifact that solves the problem. When the problem being solved is complex, the process used to create products and services itself must be organized. As a result, design activities fall into two categories: architectures, which are used to organize design, and complete designs, which are the end result of such processes (Baldwin et al. 2005).

The field of information systems deals with artificial (man made) objects and phenomena. In particular, we design systems that meet certain goals. Before building these systems (as in other design activities) designers create an architecture. The architecture describes the components of the system and the pattern characteristics of their interconnection. It includes the interface between the components and specific ways to test the properties and performance of the system (Baldwin et al. 2000). Because the architecture of the system and its task decomposition is isomorphic, the architecture describes a sensible subdivision of the tasks involved in designing the system. The architecture is then instantiated (built) in the form of a working system.

Using the design science paradigm, researchers create and evaluate IT artifacts that are intended to solve identified organizational problems (Hevner et al. 2004). Design science research deals with two design processes and four design artifacts. The two processes are: build and evaluate. The artifacts are constructs, models, methods, and instantiations. Typically, while evaluating the system, researchers treat the system as a black box. However, architecture mediates the application of knowledge to the construction of artifacts and its subsequent affordances (Orlikowski 2000). Thus, it must be evaluated in accordance with the same design science principals that its physical instantiations are.

Architecture can be evaluated by the strategic value - responsiveness, innovativeness, and economies of scope - it creates for the firm (Kayworth et al. 2001). Architectures – like designs - exist whether designed consciously or unconsciously (Alexander 1964). Thus, architectures can be evaluated both *prescriptively and descriptively*. Our contributions, as prescribed by (Hevner et al. 2004) in this research are three fold. First,

we recognize architecture as an artifact, in addition to its instantiation as a working system. As we described earlier, the generic process of design creates architecture. Furthermore, several IS researchers recognize its importance (Duncan 1995; Iyer et al. 2004a; Kayworth et al. 2001; Weill et al. 1998). Second, we present a formal way to describe architecture and changes that can be made to it. Finally, we outline a methodology to capture, measure, and evaluate architecture.

Recent trends such as outsourcing, off-shoring, integrated and flexible supply chains, and the development and promotion of web services are visible manifestations of socio-technical systems. These systems create extended enterprises that are complex and evolving. In order to make decisions regarding such systems, it is helpful to understand how they operate, what's driving 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, or developing and extending their information systems, they need to understand the landscape – ecosystem or design space – that they are operating in (Alexander 1964). In order to help managers accomplish this, we present abstractions that make it easier to comprehend the complex reality and a language – modularity and stacks – that make it easier to navigate the competitive landscape.

In building the abstraction, we are building an IS artifact (March et al. 1995) that enables organizational and information systems scholars and practitioners to understand and address the problems in information system, organizational, and strategic design. We are also getting closer to some of the core theories and features of information systems and representational design (Orlikowski et al. 2001; Weber 2003) that should focus our attention on the artifacts. Following the Design Science paradigm (Hevner et al. 2004),

we take our problem statement from the technology, organizational, and strategy literature that seeks to understand the emergence of systems (Pinch et al. 1987), organizations (Nelson et al. 1982), and industries (White 2002). We then contribute to both the behavioral literature and the design science literature through the creation and analysis of our theories and model.

In the following sections we develop the model and language of stacks, explain how they emerge, test the model against industry observations, and then conclude with discussions, limitations, and plans for future research.

Stacks

Modularity and stacks have been part of the computer science literature for decades. Research into these concepts, however, has been largely prescriptive. The research explains how and why system builders seek to create modular systems and stacks and the affordances such designs provide. Stacks are described as a type of modular system in which a specific hierarchy is designed. Modularity enables combinatorial innovation and a separation of considerations.

More recent research on modularity is descriptive. It seeks to understand all designs as incorporating some qualities of modularity and the affordances of designs with different modularity qualities. It seeks to understand how designs emerge in situations in which there is no single authority responsible for the design.

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 (Simon 2003). Modules can specialize in certain functions and communicate with other modules through their interfaces. Taking a

simplified view of industry, 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 with them in a market (Granovetter 1985; White 1981). 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 (Lawrence et al. 1967; 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 (Messerschmitt et al. 2003).

The decision to outsource or off-shore, the development of a web-service, or the implementation of extended supply chains is fundamentally a question of the type of interface between, inter alia, the outsourced activity and the other activities of the firm. In deciding whether a function should be outsourced or left in a functional unit, the issue from a modularity perspective is only one of interface definition. When a function is outsourced, the interface must generally be better defined and more stable. The communication across the interface is generally of lower volume, better structured, and more explicit. In general, the relationship with the outsourced function is more loosely coupled and the interdependencies are minimized (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 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 application 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 it shows 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 (Baldwin et al. 2000). Modular systems enable experiments; they provide the designer with valuable options (Balasubramanian et al. 1999). 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 but not vice versa.

One of the consequences of stacks is that different layers within the stack can develop at different speeds. 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 (Varian 2003). 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. Although we find stacks at all these levels, it is interesting to note that the representation used to capture them and operators required to describe changes that can be made to them remain the same. 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. More emphatically, stacks emerge independently of conscious design.

Stack emergence

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.) (Simon 1996 [1969]). 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 1997 [1945]), a dynamic environment (Schumpeter 1942; Scott 1995), incremental innovation (Brown et al. 1997), and an installed base. The first two forces lead to modularity, but not necessarily hierarchy. 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. 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. Certain

layers become, in effect, dominant designs (Utterback et al. 1975) that are infrequently replaced.

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, thereby linking them.

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. 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. As the module becomes more widely available it is also subject to greater network effects (Shapiro et al. 1999).

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 (for examples, see (Landes 1983)).

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 are connected through communication and dependency.

Communication can be a bit-stream, story-telling (Boland et al. 1995), printed documents, apprenticeships, contracts (Brynjolfsson 1994), or market-based transactions (Williamson 1981). Communication occurs between modules of a software system, within a firm, and between firms within and across industries. Dependency includes understanding the communication, maintenance of a common interface, availability, timeliness, and performance. We can characterize this link between modules in terms of the frequency of communication, the volume of communication, and the stickiness (Teece 1998) of communication. We can characterize the dependency between two modules in terms of their coupling (degree of looseness) and directionality (which module can initiate communication). In a tightly coupled (industry) system, in which two firms are highly interdependent, we may see a vertical integration between the two firms (Brusoni et al. 2001). In a loosely coupled relationship, the firms may coordinate through

the market. In the normative view of stacks, modules in an upper layer can initiate communication with a module in a lower layer, but not vice versa.

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. This is the gap that software designs attempt to fill (Simon 1996 [1969]). Of course, the user could use the computer hardware directly, but it is often 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 gap could be filled by the user working with the machine. In the early days of computing, however, this is what was done (Campbell-Kelly 2003). Since that time, greatly improving 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. The computer hardware is now hidden by multiple layers of the software stack.

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 (Shaw et al. 1996). 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 the computers, communications channels, distribution mechanisms, process dispatchers, etc, and are independent of the applications that may run on them. Higher layers use 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 on 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.

This view of architecture is particularly useful to developers. As we have moved from client server to n-tier architectures, developers of software have tried to focus on one or a few layers and relied 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 (Boland 1979). 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 levels 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).

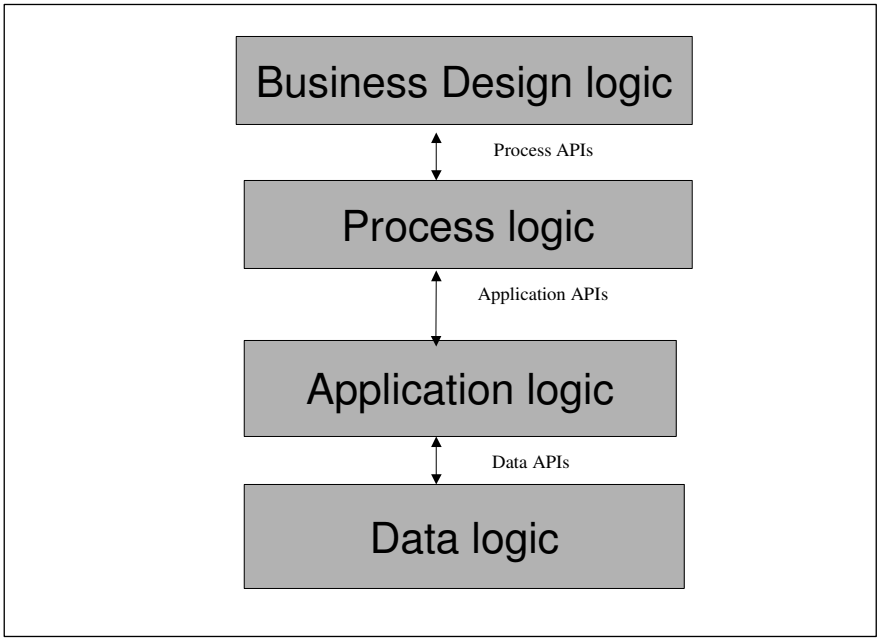

Figure 1. The Enterprise Stack

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. 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, the dependencies 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. As we described earlier, loose coupling also favors less sticky communication.

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. This type of architecture is very valuable to the CIO of an enterprise. They can use this to determine what each business unit within an enterprise is doing and how the various projects that are being implemented impact the overall architecture of the firm.

Organizational stack

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 several years, 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 similarly about the creation of stacks.

We can recast the debate between centralization and decentralization (George et al. 1991; Huber 1990) 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 (Pinsonneault et al. 1997). 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. This is a direct result of the changing economics and tradeoffs described earlier: as computers get faster, systems can opt for flexibility without

sacrificing too much performance. 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 (Alavi et al. 2001).

A business process is a complete coordinated thread of all the serial and parallel activities needed to deliver value to the enterprise's customers (Georgakopoulos et al. 1995). 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 changes in the IT systems. Structured, flexible, and high-performance communication has enabled more loosely coupled organizations (Argyres 1995).

This view of architecture is relevant to the CEO trying to get the organization to respond to changes in the environment. Architecture in this case could be use to support an acquisition or to interact with other organizations within the ecosystem.

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 their size (Barnard 1938; Conner 1991; Conner et al. 1996; Jensen et al. 1992; Milgrom et al. 1988; Nelson et al. 1982; Weick et al. 1999; Williamson 1981). 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.

The industry stack divides activities into layers that are complementary to 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) or 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 or below 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 in 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 application packages and middleware 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 Webshpere), 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. In the business services layer companies that provide software as a service like Salesforce.com or Google's mapping program Google Earth.

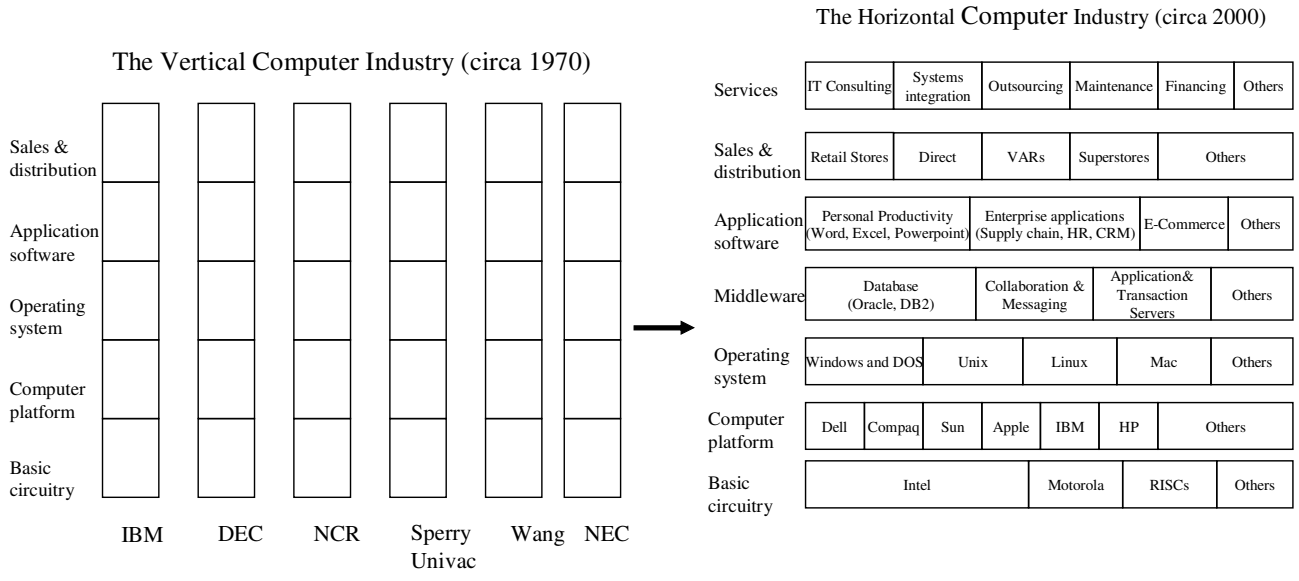

Figure 2: From vertical to horizontal transition

The change from vertical platforms in which the industry had few layers but the underlying software had 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.

Implications

Information System, organizational, and industry architectures are key determinants of a firm's success (Baldwin et al. 2005; Schilling 2000). Once an architectural decision is made, it has implications for 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 (Henderson et al. 1993; Henderson et al. 1990).

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 dominant design within the industry stack, they can influence it to their advantage and design the internal architecture to take advantage of the emergence. We call the process of influencing emergence to their advantage as exercising architectural control (reference omitted).

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) which 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 tradeoffs include the 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 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.

Evaluation

Abstractions and models can have a significant impact on the ways tasks, problems, and the design landscape are conceived (Boland 2002; Hevner et al. 2004). The solution to a problem reflects its representation. Architecture as a design science formalism, the

stack abstraction, and the language used to describe them adds to the growing body of work on representational design, modularity, and coupling. It adds to the language we have regarding coordination (Malone et al. 1994; Thompson 1967) and the management of complexity (Simon 2003).

Architecture is a design artifact that is then instantiated and evaluated. The problems we address through the development of architectures are relevant to a broad array of technological, organizational, and strategic designs. One of the foremost questions for IS developers is the creation of systems that aid organizations as measured by application-specific utility functions (Boland 1979; Boland et al. 1994). The challenge in developing such systems is our cognitive limits, the changing nature of work, and the dynamic interaction between organization and systems (Orlikowski 1996). Stack-based thinking enables architects to create systems that reflect and support this changing environment. It encourages architects to consider the creation of layers, coupling between modules, and the directionality of that coupling. It is an approach to the management of complexity.

We have attempted to evaluate the stack metaphor by testing it against three levels of analysis. It is an espoused metaphor in computer science and it explains behavior when we examine the IT industry. While it has an intuitive appeal in explaining enterprise and organizational architecture, we haven't examined firm details to make conclusive statements. However, it does seem consistent with the large body of work provided by organizational and strategic researchers.

We suggest that many of the systems instantiated and evaluated according to the principals of design science can be further evaluated by exposing and exploring their underlying architectures. Such analysis would enable us to say not only that a particular

system provides a certain utility, but why. What is it about the specific bundles of features (modules) and the way they are connected (architecture) that gives rise to the observed utility?

Design theories (Markus et al. 2002), methodologies, and ontologies are foundations for design science artifacts (Hevner et al. 2004). Stacks, coupling, modularity, and design operators – the language of architecture - are readily accessible representations and seem to accurately reflect the business and technology environments designers work in. This language is both descriptive and prescriptive. It provides a language for describing both what has occurred and what can be.

Architecture is at the intersection of behavioral and design science concerns. The systems we build reflect our assumptions regarding the nature of the problem. To paraphrase Simon (Simon 1996 [1969]), solving a problem means getting its representation right. Reversing this equation suggests that examining an artifact tells us about the designers underlying assumptions regarding the problem.

Architectures enable us to describe and discuss our assumptions regarding the solution space. If the problem of design is search, then architecture describes the search. But, architecture is both descriptive and prescriptive. It tells us where we've looked, and where we can look. It is through architecture that we incorporate the theories provided by behavioral researchers. We can then test the theories through instantiations of the design. However, we can go further than this. We can examine the architecture of instantiations and ask what is it about the architecture of the design that leads to the utility of the instantiation. Thus, we further suggest that by examining the architecture of instantiated

systems, design science can both test and develop insights and theories commonly associated with behavioral scientists.

Although design instantiations may be perishable (Hevner et al. 2004), we suggest that many of the principals that underlie their architecture are not. Object-oriented databases may not be in common use, but surely there are lessons to be learned about how they were built and how organizations experimented with them that can be applied in other settings. The challenge in design science is to evaluate both the instantiation and the design.

Conclusion

We have suggested that stacks are everywhere: they exist in IT systems, businesses, and within industries. Although well understood prescriptively, 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.

As part of future research work, we have identified a general set of questions that can be posed at the systems, enterprise, organizational, or industry level. Given that we are dealing with dependencies between artifacts, one obvious question concerns the underlying topology/structure of the system -- does the emerged architecture differ from the espoused one? If so, what network-based metrics can be used to characterize that deviation? From a performance perspective, how does the topology of the underlying artifacts affect the flexibility of the firm or system or organization or the industry itself? Since we describe a particular type of architecture -- stacks, does the incorporation of this abstraction improve the explanatory power of models used m measure performance? As a result, does it open up new ways of thinking, seeing, and discussing that result in better sense making and decisions?

When considering alignment, does 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 be used as an abstraction to compete even within other industries such as financial services, healthcare, or manufacturing.

References

Alavi, M., and Leidner, D.E. "Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues," *MIS Quarterly* (25:1), Mar 2001, pp 107-136.

Alexander, C. *Notes on the synthesis of form* Harvard University Press, Cambridge,, 1964, p. 216 p.

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

Balasubramanian, P., Kulatilaka, N., and Storck, J. "Managing information technology investments using a real-options approach," *Journal of Strategic Information Systems* (9:1), Mar 1999, pp 39-62.

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. "Modularity in the Design of Complex Engineering Systems," *Working paper*), January, 2004 2004.

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.

Boland, J.R.J., and Tenkasi, R.V. "Perspective Making and Perspective-Taking in Communities of Knowing," *Organization Science* (6:4), Jul-Aug 1995, pp 350-372.

Boland, R.J. "Design in the Punctuation of Management Action," in: *Managing as Designing: Creating a Vocabulary for Management Education and Research,* R. Boland (ed.), Frontiers of Management Workshop, Weatherhead School of Management, June 14-15, (http://design.cwru.edu), 2002.

Boland, R.J.J. "Control, Causality, and Information System Requirements," *Accounting, Organizations and Society* (4:4) 1979, pp 259-272.

Boland, R.J.J., Tenkasi, R.V., and Te eni, D. "Designing information technology to support distributed cognition," *Organization Science* (5:3), Aug 1994 1994, p 456.

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

Brown, S.L., and Eisenhardt, K.M. "The Art of Continuous Change: Linking Complexity Theory and Time-paced Evolution in Relentlessly Shifting Organizations," *Administrative Science Quarterly* (42:1), March 1997 1997, pp 1-34.

Brusoni, S., Prencipe, A., and Pavitt, K. "Knowledge specialization, organizational coupling, and the boundaries of the firm: Why do firms know more than they make?," *Administrative Science Quarterly* (46:4), Dec 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.

Churchman, C.W. *The design of inquiring systems: basic concepts of systems and organization* Basic Books, New York,, 1971, pp. ix, 288.

Conner, K.R. "A Historical Comparison of Resource-Based Theory and Five Schools of Thought Within Industrial Organization Economics: Do We Have a New Theory of the Firm?," *Journal of Management* (17:1) 1991, p 121.

Conner, K.R., and Prahalad, C.K. "A Resource-based Theory of the Firm: Knowledge Versus Opportunism," *Organization Science* (7:5), September-October 1996 1996, pp 477-501.

Dryer, D.C. "Getting personal with computers: How to design personalities for agents," *Applied Artificial Intelligence* (13:3), Apr-May 1999, pp 273-295.

Duncan, N.B. "Capturing flexibility of information technology infrastructure: A study of resource characteristics and their measure," in: *Journal of Management Information Systems*, M.E. Sharpe Inc., 1995, p. 37.

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

Gawer, A., and Cusumano, M.A. *Platform leadership: how Intel, Microsoft, and Cisco drive industry innovation* Harvard Business School Press, Boston, Mass., 2002, pp. xiv, 305 p.

Georgakopoulos, D., Hornick, M., and Sheth, A. "An Overview of Workflow Management - from Process Modeling to Workflow Automation Infrastructure," *Distributed and Parallel Databases* (3:2), APR 1995, pp 119-153.

George, J.F., and King, J.L. "Examining the Computing and Centralization Debate," *Communications of the ACM* (34:7), July 1991 1991, pp 63-72.

Granovetter, M.S. "Economic action and social structure: The problem of embeddedness," *American Journal of Sociology* (91) 1985, pp 481-510.

Grove, A.S. *Only the paranoid survive : how to exploit the crisis points that challenge every company and career*, (1st ed.) Currency Doubleday, New York, 1996, pp. xii, 210 p.

Henderson, J.C., and Venkatraman, N. "Strategic alignment: Leveraging information technology for transforming organizations," *IBM Systems Journal* (32:1) 1993, pp 4-16.

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

Hevner, A.R., March, S.T., Park, J., and Ram, S. "Design science in Information Systems Research," *MIS Quarterly* (28:1), March 2004, pp 75-105.

Huber, G.P. "A Theory of the Effects of Advanced Information Technologies on Organizational Design, Intelligence, and Decision Making," *Academy of Management Review* (15:1), January 1990, pp 47-71.

Iyer, B., and Gottlieb, R. "The Four-Domain Architecture: An approach to support enterprise architecture design," *IBM Systems Journal* (43:3) 2004a, pp 587-597.

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, 2004b, pp. 587-597.

Jensen, M., and Meckling, W. "Knowledge, Control and Organizational Structure," L. Werin and H. Hijkander (eds.), Basil Blackwell, Cambridge, MA, 1992, pp. 251-274.

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.

Lawrence, P.R., and Lorsch, J.W. *Organization and Environment; Managing Differentiation and Integration* Division of Research Graduate School of Business Administration Harvard University, Boston,, 1967, pp. xv, 279.

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

March, S.T., and Smith, G.F. "Design and natural science research on information technology," *Decision Support Systems* (15:4), Dec 1995 1995, p 251.

Markus, M.L., Majchrzak, A., and Gasser, L. "A design theory for systems that support emergent knowledge processes," *MIS Quarterly* (26:3), SEP 2002, pp 179-212.

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. "Economic-Theories of the Firm - Past, Present, and Future," (21:3) 1988, pp 444-458.

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

Orlikowski, W.J. "Improvising Organizational Transformation Over Time: A Situated Change Perspective," *Information Systems Research* (7:1), March 1996 1996, pp 63-92.

Orlikowski, W.J. "Using technology and constituting structures: A practice lens for studying technology in organizations," *Organization Science* (11:4), July-August 2000 2000, pp 404-428.

Orlikowski, W.J., and Iacono, C.S. "Research commentary: Desperately seeking the "IT" in IT research - A call to theorizing the IT artifact," *Information Systems Research* (12:2), June 2001 2001, pp 121-134.

Pinch, T.J., and Bijker, W.E. "The Social Construction of Facts and Artifacts," W.E. Bijker, T. Hughes and T. Pinch (eds.), The MIT Press, Cambridge, MA, 1987, pp. 17-50.

Pinsonneault, A., and Kraemer, K.L. "Middle management downsizing: An empirical investigation of the impact of information technology," *Management Science* (43:5), MAY 1997, pp 659-679.

Schilling, M.A. "Toward a general modular systems theory and its application to interfirm product modularity," *Academy of Management Review* (25:2), April 2000, pp 312-334.

Schumpeter, J. *Capitalism, Socialism and Democracy, 2nd ed.* George Allen & Unwin, Ltd, London, 1942.

Scott, W.R. *Institutions and organizations* SAGE, Thousand Oaks, 1995, pp. xv, 178 p.

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.

Shaw, M., and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline* Prentice Hall, 1996.

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.

Simon, H.A. *Administrative behavior: a study of decision-making processes in administrative organizations*, (4th ed.) Free Press, New York, 1997 [1945], pp. xv, 368.

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

Teece, D.J. "Capturing value from knowledge assets: The new economy, markets for know-how, and intangible assets," *California Management Review* (40:3), Spr 1998, pp 55-+.

Thompson, J.D. *Organizations in action: social science bases of administrative theory* McGraw-Hill, New York, 1967, pp. xi, 192.

Utterback, J.M., and Abernathy, W.J. "A dynamic model of process and product innovation," *Omega* (33) 1975, pp 639-656.

Varian, H.R. "Economics of Information Technology," in: *Working paper*, University of California, Berkeley, 2003.

Ware, C. *Information visualization : perception for design* Morgan Kaufman, San Francisco, 2000, p. xxiii 438 p.

Watson, R.T., Zinkhan, G.M., and Pitt, L.F. "Object-Orientation: A Tool for Enterprise Design," *California Management Review* (46:4), Summer 2004, p 89.

Weber, R. "Editor's Comments: Still Desperately Seeking the IT Artifact," *MIS Quarterly* (27:2), June 2003, pp iii-xi.

Weick, K.E., and Quinn, R.E. "Organizational change and development," *Annual Review of Psychology* (50) 1999, pp 361-386.

Weill, P., and Broadbent, M. *Leveraging the New Infrastructure: How Market Leaders Capitalize on Information Technology* Harvard Business School Press, Boston, 1998.

White, H.C. "Where Do Markets Come From?," *American Journal of Sociology* (87:3), Nov. 1981, pp 517-547.

White, H.C. *Markets from networks : socioeconomic models of production* Princeton University Press, Princeton, N.J., 2002, pp. xvii, 389 p.

Williamson, O.E. "The Economics of Organization: The Transaction Cost Approach," *American Journal of Sociology* (87:3), Nov. 1981, pp 548-577.