Architectural control and emergent architecture: a network perspective

David Dreyfus	Bala Iyer
Boston University School of Management 595 Commonwealth Ave; Room 641A Boston, MA 02215 Email:bala@acs.bu.edu	

Abstract

Understanding the evolution of a complex system is a fundamental problem tackled within the management science discipline. In this work, we view complex systems from a network perspective as comprising a set of nodes linked by dependencies. Furthermore, we classify changes to it via addition or subtraction of nodes and links. As changes are made to the nodes and links, the underlying architecture of the system evolves. A key question is -- can we control the emergence of this complex network by closely managing and monitoring a subset of nodes? We define this set of nodes as architectural control points.

Information systems are used as a setting to define our concepts and test three propositions. IS architecture emerges as a result of a sequence of IS project implementations. The architecture that emerges can be viewed as a network of software components linked by their interdependencies. The network influences, and is influenced by, the intra-organizational interdependencies in which it is embedded. IT management can influence the evolution of the network, and, by extension, the evolution of the organization. However, given time and cost constraints, IT management can most directly influence only a few of the components in the network, the architectural control points. In this research, we use a simulation model to show how a network perspective, using research methods from social network analysis, provides a useful abstraction for understanding architecture. We apply modular operators from design theory to enact changes to architecture. Finally, we show that by following a few simple rules, enterprises can improve the fitness of their architecture as the network emerges and the control points shift over time.

1. Introduction

Complex systems have been defined as ones that are composed of interrelated subsystems, each of the latter being in turn hierarchic in structure until we reach some lowest level of elementary subsystem [1]. Complexity arises as a result of both the number of distinct elements in the system and the nature of interconnections and interdependencies among those elements. Simon [1] continues by suggesting that hierarchies are built on stable subsystems and are nearly decomposable. The core idea of decomposability is the minimization of dependency between all the components within the complex system. The dependencies within a stable

subsystem are greater than the dependencies across subsystems. If represented as a network, such a system would contain clusters in which nodes within a cluster would have more links to other nodes within the cluster than to nodes in other clusters. If represented as a Design Structure Matrix (DSM) [2], there would be more elements along the diagonal than in other locations.

Extant literature has identified many techniques to deal with complexity. One technique that firms use in order to cope with complexity is by developing a hierarchy to both buffer core subsystems [3] and differentiate, specialize and exploit local resources [4]. Another technique used to understand a complex system is to consider them as products that consist of both components and an architecture for their interconnectedness [5]. Under this view, firms could either focus on a component or the interface. Also, one firm's architecture could be another firm's component.

In a static world, firms could ultimately discover their entire range of organizational structure and product architecture options, and then select their optimal, equilibrium solution set. However, we live in a dynamic world in which the important design constraints and critical subsystems change over time, resulting in patterns of disruptive change in economies, firms, and products [5-7]. As a result, firms go through a process of sequential search in product design, organization, and routines adjusting their internal processes to a changing external environment [1, 8, 9]. Such an evolutionary, emergent process modifies the pattern of interdependencies we've previously described.

If we characterize the pattern of connections between components as architecture, then we observe a world of economic architectures, organizational architectures, and product (including services) architectures. Clearly, firms (through managerial action) influence the emergence of

these architectures. However, no manager (or firm) has enough power to control any but the smallest architectures. As more stakeholders get involved in a system, the influence of any single actor is reduced. Moreover, within any of these architectures, some components are more important to the evolution of the rest of the architecture than others. Therefore, managers faced with a limited ability to control need to be aware of the important components within architectures and the stakeholders that control them.

We call these important components the architectural control points (ACP). In this paper we develop the concept of ACP using the context of information systems, provide a method to identify them, and show how one can control architectural evolution by using architectural principles.

2. Competing using information systems

Information systems are a significant investment for the modern day enterprise [10]. In the face of turbulent, competitive environments, firms must continuously innovate and engage in multiple IT implementation projects. IT managers must decide whether to make or buy systems and furthermore, whether to in-source or out-source the projects. To provide guidance for project-level decision-making, companies develop a designed architecture. However, subsequent projects' impact on existing applications, data, and technology can, and, often does create a gap between the emergent architecture and the designed architecture. We propose that the ability to define and manage this gap will improve firms' ability to realize the expected returns on their IT investments. We argue that understanding this emergent architecture allows managers to recognize the impact of potential projects, and make decisions that maximize the chances of individual-project and IT-strategy success.

Although IS architecture has no universally accepted definition in either the research arena or in the practitioner world, many perspectives abound [11]. Architecture has been viewed strategically [11-16], organizationally [17-21], and technologically [22-25].

Previous studies have summarized that an IS architecture includes a group of shared, tangible IT resources (i.e., hardware, software, data, training, management, etc.) that provide a foundation to enable present and future business applications [15, 19, 21, 26]. Architecture, as implemented through its IT infrastructure, should be flexible, reliable, robust, scalable, and adaptable [18, 19, 26]. It should support the reuse of business components within a firm, while supporting firm responsiveness, innovativeness, and economies of scope [26]. A review of these and other articles clearly illustrates that when the given definition for architecture is translated into action, the concept becomes very complicated.

Architecture implementation also involves learning effects. As researchers have explored organizations making changes to architecture, they have identified two strategies: localized exploitation or enterprise-wide integration [27]. In addition, they have identified that changes do not occur in one step; they occur in stages [11].

Architecture also reflects and supports business strategy. Architecture is not just concerned with the allocation of resources at the physical level, but also the support of strategic business goals. The architectural challenge is not just cost minimization in the allocation of task to computational device, but the alignment of the task structure [28], supported by the information system, with the business objectives of the organization.

Zachman provides a useful framework that identifies the components of an IT architecture as well as the various perspectives taken during the design and implementation of an architecture [29]. In this view, there is no such thing as a single information architecture; there are many.

Separate architectures exist for scope/objectives, business model, information system description, technology model, detailed description, and machine language description. For these six categories, there are also the descriptions for *who, what, how, where, when,* and *why.* Altogether, there are thirty-six possible architectures.

Iyer and Gottlieb [30] identified three views through which we can examine architecture. The first view is the espoused (Architecture-in-Design). It is the outcome of the process of defining and modeling the architecture and describes the planned dependencies between system modules. They argue that although there may be multiple participants in the design process, the espoused view is the province of the IS Architect. As a result of implementing individual projects, an enterprise architecture emerges that may differ from the espoused view. This emergent architecture is the second view. It is a descriptive view of the actual dependencies that exist amongst system modules. Finally, Iyer and Gottlieb define the third view, architecture in-use (Architecture-in-Operation). The in-use view highlights the dependencies between and among system components and organizational groups that arise from the business of doing the work of the enterprise – selling products, buying supplies, managing employees, etc - as employees, suppliers, and other stakeholders interact with the system.

The emergent architecture can be a result of conscious or unconscious action [31]. That is, organizations with information systems can evaluate projects with the intent to understand how the project will impact the emergent architecture, or they can evaluate projects without this architectural perspective. An organization can develop an architecture, implement it according to plan, and then have a series of subsequent IT projects that, in response to specific or general business requirements, modify it. Iyer and Gottlieb (2004) argue that the way architecture

emerges may subsequently impact how well the firm can achieve IT/business strategy alignment as well as the goals of flexibility, reliability, robustness, scalability, adaptability, and reusability.

In the quest to align IT and business strategy, many approaches are relevant. Take for example, the technical perspective taken by architecture modeling [32]. UML modelers develop use cases from which models are developed, structured analysis highlights the relationships between data elements and processes [33], and Petri-Nets have been used to analyze coordination constraints [34].

Taking a different approach to alignment, researchers have proposed general modular systems theory and applied it to study inter-firm product modularity [35]. In their book, Design Rules [36], Baldwin and Clark propose a general theory for modularity to explain how industries have accomplished hitherto unimaginable levels of innovation and growth. Their approach to understanding architecture is focused on identifying the components of architecture and the design rules that govern their conceptualization, creation and delivery.

The dependencies that exist between components, while acknowledged, are rarely analyzed. These dependencies have more than first-order effects. Cumulative effects of dyadic dependencies between systems create higher-order dependencies that can impact a firm's ability to align its IT architecture with its business strategy. In this work we focus on the dependencies and, in particular, how the dependencies between the components and organizational interdependencies co-evolve.

At its core, architecture influences the design decisions and the investment behavior of an organization. When viewed from this perspective, dependencies within an architecture act as conduits through which ideas, norms, and meaning flow. IS architects influence both the

emergent architecture and the host organization through IT project guidelines. Due to the limited resources available to the architects, the selection of which parts of the architecture they pay attention to may be of significant importance to the eventual shape of the network, and their ability to influence the systems within it and the groups that use it. We utilize social network analysis concepts to examine dependencies between software components and illustrate how changes in emergent architecture lead to changes in IS architects' ability to influence organizational behavior. Social network analysis can help us understand the emergent architecture, the shifting influence of components within it, and the implications on IS architects' ability to influence organizational behavior.

3. Architecture as social system

The architectures developed by the organization reflects its composite IS strategy. The espoused architecture represents the codified understanding of what the organization requires, subject to technological and organizational limits. The implementation of the architecture enables certain organizational capabilities, and constrains other capabilities. The tasks that are allocated to specific computers represent the actions and responsibilities of specific users. The allocation of tasks to systems, thus, mirrors existing dependencies between groups, and creates new dependencies between them [37, 38]. As the architecture changes – emerges – over time, so do the dependencies between the systems.

The components of the information system are imbued with symbolic meaning that shapes and is shaped by the using organization [39]. Organizations can utilize IT systems to enhance their ability to seek out new information, but these same systems can also limit the information the firm sees [40]. Information not expected by the system can too easily be filtered or otherwise ignored [41]. Different groups may view their environment and the other groups

within the organization differently. This differentiation process is a natural response to complexity [42], but also requires that the firm integrate the different components [4]. Achieving an effective balance between expected and new information is another objective for a successful architecture. [43]

The challenge of integration is not automated or eliminated by information systems. Conflicts between groups are reflected in conflicts within the information system [44]. The conflict is not the sharing of bits, or even in the scheduled exchange of information. The conflict is due to different semantics, different understandings of the world. The organization does not share a single cognition; rather, cognition is distributed [45] throughout the organization and is realized in the varying routines and tasks that bind its members [9].

The emergent information system reflects the interdependencies between the tasks and groups and serves as an integration mechanism between differentiated groups within the organization. The existence of certain information conduits implemented in the IS system influences communication between the groups both syntactically and semantically [46]. The evolution of the information system also reflects different power relationships and conflicts [47].

The information system reflects the debate between organizational elements [48-50]. The system both shapes the debate by shaping information conduits and manipulating dependencies, and is shaped by the debate as groups implement IT projects. IS architects enter the debate initially by influencing the espoused architecture, and subsequently through the evaluation of, and influence over, subsequent IT projects.

4. Emergent Architectures

As described earlier, architectures have three views - espoused, emergent, and in-use [30]. The espoused architecture is the view held by the system architects. It is the architecture described in UML diagrams, E-R diagrams, and design specifications. The emergent architecture is the actual allocation of tasks to platforms that results from the original implementation subsequently modified by changes made to the system as the system and organization adapt to each other [38, 51, 52]. The in-use architecture represents the patterns of interaction and dependency between components of the IS as they are utilized. It incorporates human actors into the information system [53].

Our focus in this research is the emergent architecture because we are interested in how ongoing changes within the system affect the scope of managerial choice. Because in-use architecture can be construed as an extension of the emergent, we will focus on emergent architecture first.

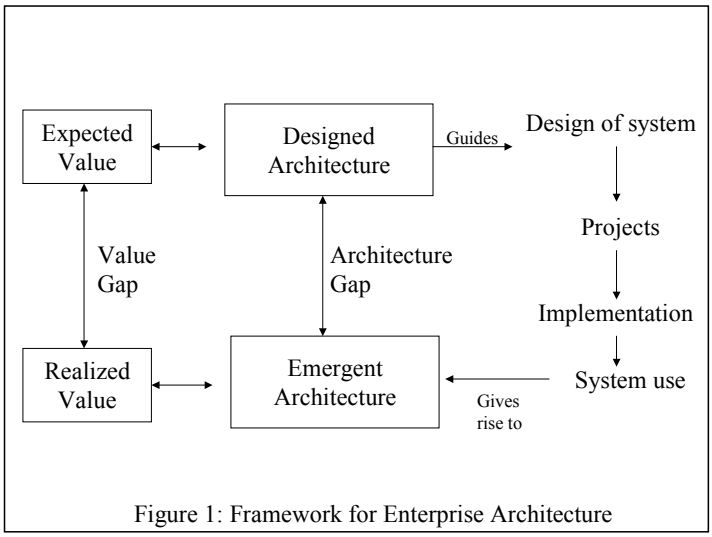

Figure 1: Framework for Enterprise Architecture

The gap between the espoused and emergent architectures is created the moment developers, in an attempt to implement the espoused architecture, make decisions that were not specified in

the espoused architectural documents. As users adjust the system through one-off integrations, modifications, and additions, and as subsequent projects modify the information system, the gap between the espoused and emergent architecture may grow. The gap can be narrowed through proactive enforcement or modification of the espoused architecture. However, in large-scale systems, this is a time consuming job that often lags behind the "grounded reality" of the organization and, hence, is hardly ever done.

Given this reality and limited organizational resources, it is important to identify a subset of systems that the organization can choose to control. A system can be deemed important for two reasons. It can be intrinsically important because of its attributes or the stand-alone value it provides to the firm. We call this autarky value. For example, an accounting system maybe valuable because it may hold specific data, perform critical tasks, represent large investments, or have large numbers of users. A system can also be deemed important because of its position in a larger network of interdependent systems. CIOs often focus on the intrinsically important systems and allow business units to flexibly create other as necessary. Positionally important systems derive their status because they influence the emergence of the architecture, the ideas that flow through it, and the dependencies it creates and reflects. Having control of these systems allows decision makers to influence the evolution of the architecture to support the business goals of the enterprise. We call these systems *architectural control points (ACP)*. It is important to note that the ACPs and intrinsically important systems are not necessarily the same.

To describe and understand emergent architecture, we adopt modularity theory [23, 24]. Modularity theory enables us to view an architecture as an interconnected set of modules. Baldwin and Clark [36] have also provided a set of operators to describe changes that can be

made to an architecture (described later). ACPs are modules that can be used by a stakeholder to manage the flow of information between connected components of a modular system. An ACP can, in effect, become a gateway or bottleneck through which information flows.

Emergent architectures can be viewed at different levels of abstraction (varying amounts of detail), depending upon the purpose and perspective of the observer. An architectural rendering of a system is, by definition, an abstraction of the real thing. No matter which tools, methods, and techniques are utilized to capture the architecture, no architectural rendering will be as complete as the actual system. Thus, different stakeholders decide what to include and what to exclude in describing the architecture. By extension, the architecture that emerges as a result of change over time is similarly an abstraction. In the next section, we present a network-oriented architectural abstraction.

5. Network Analysis

Network analysis has been utilized in a great many areas to describe many complex phenomena [54]. Software researchers have proposed metrics [55] to assess the structural properties of software and measure complexity using techniques such as DSM [56]. We are introducing the concepts of social network analysis into the examination of architecture because information systems are both the product of social construction [49] and first-class actors in the network that includes the stakeholders of the information system [57]. The computer-based information system is the manifestation of an ongoing dialog between the different stakeholders of the system. It is both shaped by the organization and will shape the organization in a cyclic, recursive process [38, 52].

Examining the emergent architecture provides additional insight into the evolution and interactions within the organization. However, as we have argued, the information system is

not a passive reflector of this evolution. It also contributes to the evolution of the organization [38]. Therefore, the information system architect plays a role in shaping the subsequent organization.

The architect that is interested in such variables as IT project cost and success, and system flexibility, robustness, adaptability, and performance should be interested in the ongoing evolution of the information system architecture. These outcomes are the result of an interaction between system and organization that system architecture influences [38]. Project cost and maintenance is directly related to complexity [58], which is a description of interdependency. If the goal of architecture is to provide flexibility, robustness, and adaptability, then any change to the architecture will affect those characteristics.

Managers looking to address IT project performance and overall system performance must balance the needs of various internal and external organizations, and even engage in the negotiations that decide whether a system is a success or failure [59]. Dependencies between components of the IS architecture reflect dependencies between the organizations that utilize the components. The architect's ability to modify the network is constrained by the interdependencies of the groups utilizing the existing components. Moreover, the interdependencies evolve as the information system is adapted by the organization, and as the organization adapts to the information system. Through frequent intervention, the architect can influence the evolution of the information system, but may not be able to control it.

If we assume that the architects cannot control all the components of the information system, then the question the architect wants answered is which components should be controlled, or more actively managed. That is, what are the architectural control points? Similarly, how does the espoused architecture influence subsequent emergence and control points? Are there

architectures and simple rules that minimize the change in control points? The control points may be identified according to a specific set of criteria, or they may be identified due to their network position within the espoused architecture. In a system of emergent interdependencies, however, the systems identified as core by one set of managers, at one point in time, may not be identified as the most important by either the original set of managers or other groups of stakeholders at another point in time. That is, the loci of interdependence may shift over time. In this research, we identify importance by network position, as described later.

One of the characteristics of social network analysis is that complex phenomena are abstracted to a simple representation of nodes and links. In our abstraction, nodes represent software components and links represent dependencies. Within a network, we utilize a single node type and a single edge (link) type. The emphasis in this architectural rendition of the network is the relationship between nodes and not on specific business functions, APIs, or tasks. For any actual information system, many different network representation can be created, each representing a different abstraction of the components and the links between them.

6. Systems

In our abstraction, we are interested in representing computing applications and their interdependencies. As we will discuss later, interdependencies have many forms. Similarly, computing applications have multiple definitions. For the purpose of this paper, we will be fairly precise in our definition of a computing application. It is an executable software component with a well-defined and published interface. As such, it may be used by other programs without the user of the component being aware of its implementation details. It is separately buildable, installable, and manageable. That is, it must be possible to compile, link,

and ship the computing application independently of the other applications in the information system.

An application that consists of multiple modules that must all be present to have a functioning system (e.g., a database management system) is considered a single component. For an application module to be considered a separate component, it must be possible to procure that module from different vendors or it may have been developed independently within a firm.

7. Dependencies

One of the tasks in social network analysis is defining what constitutes a link. The links between nodes represent conduits through which information, ideas, and knowledge flows. In individual and organizational research, links are defined as either formal, contractual or non-formal, non-contractual. At the individual level, links include friendship (i.e., A is a friend of B), advice (i.e., A seeks advice from B), trust (i.e., A trusts B), reporting (i.e., A reports to B), and association (i.e., A and B are linked by virtue of being members of the same organization). At the organizational level, links are frequently defined through formal alliances. Firm A is linked to firm B if they have announced an alliance. Links also exist if firm A is a supplier to firm B or if the firms collaborate.

Within information systems, the most obvious links between systems are those in which two systems exchange data. In this model, two nodes are linked if they share data. As a dependency, we would say that system A depends on system B for data. However, by drawing on the coordination literature [22] and organizational research on interdependence [3], we understand that there are multiple types of dependencies by which we could characterize the tasks represented by systems: resource sharing, producer/consumer, and simultaneity.

A resource dependency exists when system A depends on data managed by system B. For example, applications that share a common Oracle database have a dependency on the shared data resource managed by Oracle. The database management system (DBMS) has many facilities to manage the dependency on this shared resource. A producer/consumer dependency exists when system A depends upon the results of system B. For example, a general ledger application depends upon the data processed in a sales order application. A simultaneity dependency exists when two or more tasks must be completed before a third task can begin. For example, a reporting application may need data from multiple systems before compiling the report.

Dependencies are both direct and indirect. A direct dependency exists when two systems directly exchange data or that have their tasks coordinated. An indirect dependency exists when application A depends upon application B, which depends upon application C. For example, a reporting application directly dependent upon a data warehouse may be indirectly dependent upon a transaction-supporting database system. The warehouse may minimize the indirect dependency of A on C by hiding the complexity and details of the underlying transaction system, but it is ultimately still dependent upon the timeliness and accuracy of the underlying data.

The implication of direct and indirect dependency in the context of nearly decomposable systems [42] is that the influence of one component of the information system on another decreases as the distance between the two components increases. In network terms, the influence of one component upon another component decreases as the number of links between them increases.

The influence a component exerts on the rest of the system is a function of the number of direct links the component has to the other components, and the indirect links between the component and all others. This measure, in the social network literature, can be described through *influence centrality* [60, 61]. In our study, we approximate this measure by assuming that influence travels along geodesics and then utilizing Key Player [62] to compute. As we shall see later, the implication is that components that may be important at one point in time may decline in importance (influence) relative to the other nodes of the network depending upon how the network evolves.

System developers implementing changes to the information system must confront these dependencies when modifying the system. A formal abstraction of the ways in which the system can be modified is presented later. However, to better understand the nature of dependency, we need to consider three types of changes to the system: modifying an existing component, adding and linking a new component into the system, and linking two existing components.

In order to modify an existing component, the dependencies between the component and its direct dependents must be negotiated. If we make the simplifying assumption that the maximum number of dependencies between any two components is two (bidirectional), and we ignore indirect dependencies, then modifying a component connected to N other components requires a maximum of (N+1)*N negotiations. This includes the negotiations between the target component and its alters, plus the negotiations between the interdependent alters. (In a maximally connected ego network, the number of arcs is N*(N-1)).

Adding a new component to the network involves the negotiation of two dependencies: the dependency of the new component on its point of connection to the network, and the point of

connection's dependency on the new component. By accepting the connection, the point of connection's freedom for subsequent change has been restricted to the extent it will need to negotiate change with the new node. Adding a new node to the network also shifts the relative importance of existing nodes within the network.

Integration between two existing nodes increases the number of dependencies on each node by one. It also increases the number of indirect dependencies between components and may shorten the shortest path between pairs of nodes. As links are added to the network, the balance of influence among nodes may change.

An architect interested in managing the evolution of the network, its impact on the organization, and the organization's impact on the system may be interested in how these modifications to the network affect certain network characteristics. In our research, we are interested in the shape of the network (the pattern of nodes and links) predicting overall system characteristics and the success of adding nodes or modifying existing nodes within the network. Before we can examine the implications of network change on network characteristics, we need a more formal mechanism to describe changes to the network.

8. Design Moves

To describe and understand emergent architecture, we adopt advances in modularity theory [35, 36]. Modularity is a state descriptor of a firm's architecture, which influences the potential moves or changes that can be made to it. These moves can be described by a set of modular operators.

The list of modular operators identified by Baldwin and Clark is comprised of the six core operators: splitting, substituting, augmenting, excluding, inverting and porting. 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 using the outputs of one system as inputs into another. This linking can be between two systems that run on the same operating system or between systems that run on two different operating systems. For example, a data mining application that draws on a new data source can be seen as linking to that data source.

Within our network, we can further abstract the operator set to: modifying a node and its adjacent links, adding a node and a link, and adding a link. We are uninterested (at this point) in why a node is modified (thus collapsing substitution, splitting, and inversion); we are uninterested in disconnected networks (thus, we ignore augmentation without a link); and, we do not envision systems decreasing in complexity (thus, we ignore exclusion).

So far, we have described reasons for taking a network perspective on architecture, what constitutes nodes and dependencies, and design moves to transform architecture. In the next section we will describe our simulation.

9. Simulation

In order to understand the relationship between network emergence and the change in the architectural control points in systems in which no single authority controls all the projects, we simulate emergence of architecture. In order to increase our insight into emergence, we test these propositions:

Proposition 1: Emerging architectures result in lesser control of key nodes.

Proposition 2: Architectural thinking reduces the decline in control of key nodes.

Proposition 3: Preferred network topologies such as small worlds are preserved with architectural thinking.

As in any simulation, our models are abstractions. We make the following simplifications and assumptions: all dependencies between nodes are of a single type, organizations react to changes in the competitive environment by making changes to the architecture, and only three types of changes – splitting, augmenting and linking – are allowed. Finally, we assume that all nodes have the same intrinsic or autarky value.

The simulations are run under two sets of rules with two different starting conditions. In the first set of rules, the probability of an operation being permitted on a node of the network is uniformly distributed across the nodes (i.e., no architectural thinking). In the second set of rules, the probability of an operation being permitted on a node is proportional to the normalized degree centrality of the node (the probability is d/N, where d is the number of links adjacent to the node, and N is the total number of nodes in the network). We characterize the use of degree centrality by an organization as evidence for the presence of architectural thinking. In fact, sharing resources is considered a key dimension for evaluating the presence of architecture within firms [25].

The two starting conditions are the shape of the espoused network (see Figure 2): in one set of simulations, the simulation starts with a random network, in the other it starts with a small world network. These two starting conditions are created to represent unplanned and planned architectural starting conditions, respectively. When we have a small world starting condition, we assume that the firm has created domains of services or systems (say customer related or product related) that are highly connected with one another and loosely connected with other domains of systems. This style of architectural thinking is prevalent today in service oriented architectures (SOA) in general and Web Services in particular.

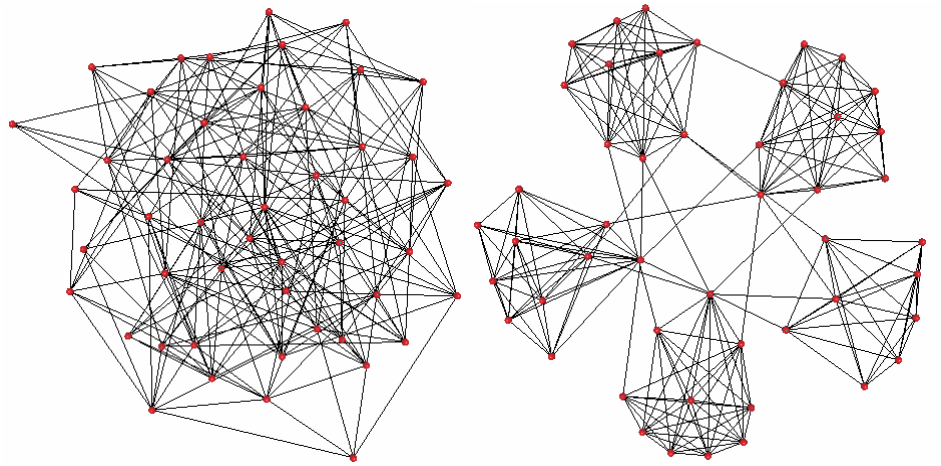

Figure 2: Networks before emergence. The random network is on the left; the Small World network is on the right.

For each of the four cases, the simulation was run 10 times. The results are based upon averages. Both networks start with 48 nodes and a density of 0.20 (20% of all possible ties among nodes exist in the networks). The networks are constructed to have the same node count in order to facilitate subsequent analysis. To create the random network, we generate 48 nodes and then randomly selected pairs of nodes to link until the desired density is achieved. Both networks represent a starting condition for analyzing the impact of architectural change on the importance of specific architectural control points (i.e., specific nodes in the network).

The purpose of this simulation is to understand the impact of change on the importance of systems, and not to understand how different types of change can impact specific systems. As a result, only a limited simulation is performed.

Network evolution follows the following logic. The simulation is run until 100 operations are performed on the network. At each clock tick up to three IT projects are proposed. The first proposal is to integrate two randomly selected components (nodes) by *linking* them with a bi-directional edge. The second proposal is to *augment* the network with a new component (add a node) and connect the new component to a randomly selected existing component (creating an

edge to an existing node). The third proposal is to *split* an existing component into two components (because the existing component has become too complex). When the existing component (randomly selected node) is split, the existing integrations (links) are randomly distributed between the old and new components (nodes).

For each simulation run, we simulate the arrival of project proposals (design move). In our model, there is a 50% probability that a *linking* project will be proposed; there is a 30% probability that an *augmenting* project will be proposed; and there is a 20% probability that a *splitting* project will be proposed. When a *linking* project is proposed, and the probability of approval is architecturally informed, the probability of approval is (d1+d2)/N, where d1 and d2 are the cardinality of edges adjacent to randomly selected nodes 1 and 2 respectively, and N is the total number of nodes in the network. When an *augmentation* project is proposed and the probability of approval is architecturally informed, the probability of approval is d/N, where d is the cardinality of edges adjacent to a randomly selected target node. When the *splitting* project is proposed, it is rejected out-of-hand if there isn't more than two links adjacent to the randomly selected node. If this test is passed, the probability of approval is d/N.

Dependent variables

Broadly speaking, we are interested in two characteristics of the emergent architecture. First, how well the initially selected set of ACP's influence is preserved. Second, how well the initial design pattern (network topology) is preserved.

We measure preservation of influence by comparing the fitness of the initially selected ACPs before and after the simulation run. In this case, fitness is a measure of how efficiently the ACPs reach the other nodes in the network. The measure of design preservation is captured through the networks' Small World Quotient [62]. A high value represents a high ratio of

clustering to short path lengths relative to a random network with the same node count and arc density. For each simulation, a number of standard measures such as network density and node counts are collected. In addition, we compute the following:

Hamming Distance. This is a measure of the difference between the starting and ending networks. To calculate this distance, we first enlarge the smaller network by adding isolated nodes to it until the two networks have an equal number of nodes. Then we count the number of edges in each network that are not in the other, and normalize by dividing by the total number of edges ($\sum | Xij - Yij | / (\sum Xij + \sum Yij)$).

Architectural control point measure. Architectural control points (ACPs) are those components that collectively have the shortest path lengths (geodesics) to all other components. Within this simulation, we assume that one node's influence on another node is proportional to the reciprocal distance to that other node. We assume that IT managers will want to minimize the distance between the nodes they manage intensively and all other nodes. In terms of this simulation, we want to *maximize the reciprocal distance*. To select the ACPs, and measure the influence of the selection on the other nodes in the network, we make the simplifying assumption that influence in the network travels along shortest paths and then utilize KeyPlayer metric KPP-POS [63], which provides a fitness measure that reflects cumulative lengths of the short paths between the ACPs and the rest of the network.

At the beginning of each simulation run, we determine the ACPs of the network assuming we could select 3, 4, 5, or 6 components from the network for the ACP set. We store the list of nodes and calculated the following measures.

Small World Quotient. The essence of a small world network is a pattern of connections between nodes that results in multiple clusters and short path lengths (see Figure 2). The small

world quotient is a ratio of two ratios. The numerator is the ratio of the average cluster coefficient to average geodesic in the given network. The denominator is the same calculation performed on a random network with the same number of nodes and arcs. A random network has a low small world quotient. As it turns out, networks that either have the idealized small world shape (see Figure 2) or a scale-free structure (the degree centrality of the nodes follows a power law distribution) have high small world quotients.

Starting Fitness. This is the weighted percentage of all nodes reached by nodes in the best ACP set. This number represents the sum of the reciprocal shortest paths from one node in the ACP set to each of the non ACP nodes in the graph. We normalize the fitness measure by dividing it by the number of nodes in the graph.

After evolving the network, we then calculate the following measures.

Fitness Difference. This is the difference between the fitness measure for the ACPs at the beginning of the simulation and the fitness measure at the end of the simulation for the same set of ACPs. The difference shows the change in the fitness of the starting ACP set. In effect, this is the change in the importance of the components that the IT manager has identified as important.

Overlap. This is the number of nodes that are in common between the optimal ACP set at the beginning of the simulation and at the end of the simulation. The size of the ACP set is held constant, but the nodes in the set are changed if it will improve fit.

Best Fit. This is the fitness measure for the new ACP set determined at the end of the simulation. This represents what the IT manager could have if he/she changed the ACP set definition as the network evolved.

Small World Difference. This is the difference between the Small World Quotient of the network at the beginning of the simulation and at the end. A change the Small World Quotient represents a change in the ratio of clustering to path lengths and indicates a change in the basic pattern of links within the network.

The results are displayed in Table 1. As measured by the Hamming distance, the greatest change in the network structure occurred in the random network. The small world network, which appears more modular, is less different after 100 operations than the random network. This suggests that espoused design is not completely lost through the emergence process.

In all cases, the original architectural control points that achieve the best reach to all the other nodes in the network are no longer the best after 100 operations. Although there is some overlap, the IS Architects could improve their influence over the network by changing their ACPs. Looking at Table 1, we see that the ending fitness of the original control points drops. Finding a new best set of control points does not restore reach completely – the best fitness is always less than the original – but the best fitness is always higher than the ending fitness.

It does appear that the imposition of architectural rules has a significant impact on the fitness of the architectural control points. Architectural evolution that decreases network density decreases the fitness of the control points. A subsequent regression used to perform a sensitivity analysis on the results shows that the architectural rules' impact on density, and not the initial network design, is a significant predictor of ending fitness.

Simulation Condition	ACP Count	Node Count	Density	Hamming Distance	ACP Overlap	Starting Fitness	Ending Fitness	Best Fitness	Starting Small World Quot	Delta Small World Quot
Random network (a)	3	101.8	0.062	0.250	1.1	0.888	0.585	0.647	0.905	0.443
	4				1.2	0.946	0.636	0.701		
	5				1.1	0.979	0.648	0.743		
	6				2.2	0.995	0.694	0.773		
Random network (b)	3	86.9	0.086	0.231	0.9	0.874	0.665	0.729	0.875	0.450
	4				0.5	0.936	0.703	0.785		
	5				1.2	0.973	0.745	0.827		
	6				1.4	0.995	0.776	0.860		
Small world network (a)	3	98.3	0.060	0.201	0.7	0.880	0.572	0.647	3.310	0.433
	4				0.8	0.950	0.641	0.719		
	5				0.5	1.000	0.667	0.771		
	6				1.0	1.000	0.701	0.805		
Small world network (b)	3	85.6	0.083	0.207	1.1	0.880	0.632	0.729	3.310	-0.150
	4				1.0	0.950	0.709	0.799		
	5				0.9	1.000	0.716	0.843		
	6				0.7	1.000	0.753	0.874		

Table 1: Simulation results. N = 10 for each starting condition. Average values displayed.
a. Uniform acceptance probability. b. Degree adjusted acceptance probability.

In order to better evaluate the causes of the deterioration in fitness after evolution, we ran two simple regressions. In the first one, we regressed ACP Overlap on two categoricals - whether or not the initial network was a Small World and whether or not the probability of project acceptance was weighted by a node's degree centrality – plus the initial size of the ACP set. The results (not shown) show that the only significant predictor is the size of the ACP set, which explains only 4.5% of the variance.

The second regression regressed the fitness difference on the same two categoricals plus the size of the ACP set. The results show that the weighting of project acceptance explains 32% of the variance in ACP fitness. Systems that grow primarily in the number of links (see Table 1) seem to have a smaller impact on the importance of the ACPs than systems that grow through the addition of nodes. The higher linking preference can also be interpreted as a reuse

preference, which then suggests that systems that give a preference to reuse do a better job of preserving the initial ACPs.

A second important finding is that the small world network (the service oriented design) is only robust if architectural thinking guides its emergence. In all other simulations the Small World Quotient increased. Ordinarily, one would consider such an increase a positive development. However, when we look at the Q values in Table 1, we see that the emergent networks tend towards centralization (a scale-free pattern) and not a service oriented design. Thus, we conclude that a decrease in the Small World Quotient is a positive development.

In all the simulations, network growth tends to create a core-periphery structure in which a dense center of interconnected systems is surrounded by one-off systems. Although these emerged systems all have a high small world quotient, none of them have a clear design pattern. Consistent with the descriptive data and sensitivity analysis, the system that started with a small world structure and emerged through a process guided by architectural thinking is closest to the service oriented design it started with.

10. Conclusions

In this research we ask the question, can we meaningfully represent the complexity of information system architecture in social network terms, and then capture value from such representation? Our approach is to represent the components of the architecture as nodes, and the dependencies between the components as links or arcs.

We focus our attention on the emergent view of architecture. The emergent view is appealing because it reflects past inter-organizational interdependencies and influences future ones. Certain nodes occupy key positions in a network and, more importantly, confer unique decision rights to owners of those nodes. We call such nodes architectural control points

(ACPs). These ACPs, however, are moving targets and are not necessarily known a priori or under the control of the CIO or architect.

As the architecture evolves, the ACPs change. In organizations in which power and influence is multi-polar and decision making is distributed, the influential systems may also change. In addition, a failure to control the control points may result in a socio-technological system that evolves in a way inconsistent with the business strategy. Moreover, the system's evolution may make subsequent changes more costly to implement and result in a system that is more error prone.

These changes to the ACPs are important to understand. Architecture impacts the flexibility, stability, and performance of information systems and ability to align the system goals with a firm's business strategy. Emergence causes the underlying architecture to change and impacts the cost, timeliness, and success of subsequent software projects.

Our simulations confirm three propositions regarding the emergent nature of information systems. First, network growth results in a deterioration of the network influence offered by an initial set of control points. Second, architectural thinking in the form of rules that guides emergence can reduce the degradation of influence. Third, preferred designs are best maintained through architecturally informed rules that guide emergence.

While we develop the paper based on the information systems context, we can generalize the thinking to the development of physical products, information products, inter-organizational networks, social networks or even networks of services. A critical step before we can claim generalizability is to demonstrate that network abstractions utilizing nodes and links of information systems, product architectures, organizational architectures, and inter-market architectures have predictive value.

For the behavioral part of the simulation we use design operators. These design operators are at a high level of abstraction and can be applied to both physical and informational goods. Their arrival rates and semantics can be customized to suit particular phenomenon under study.

Similarly, within the literature researchers have identified architecture in design, production and use. In this paper, we use ACPs identified in the design stage to track changes in the production stage. While we show that architecture control points change when we go from design to production, we can also test to see if the same result holds true when we move from design to use or from production to use.

11. References

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

2. Steward, D., *The Desing Structure System: A Method for Managing the Design of Complex Systems.* IEEE Transactions of Engineering Management, 1981. **28**(3): p. 71-84.

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

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

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

6. Christensen, C.M. and J.L. Bower, *Customer power, strategic investment, and the failure of leading firms.* Strategic Management Journal, 1996. **17**(3): p. 197-218.

7. Landes, D.S., *Revolution in time : clocks and the making of the modern world*. 1983, Cambridge, Mass.: Belknap Press of Harvard University Press. xviii, 482 , [40] of plates.

8. Cyert, R.M. and J.G. March, *A behavioral theory of the firm*. 2nd ed. 1963, Cambridge, Mass., USA: Blackwell Business. 332.

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

10. Brynjolfsson, E., et al., *Intangible assets: Computers and organizational capital / Comments and discussion.* Brookings Papers on Economic Activity, 2002(1): p. 137.

11. Ross, J.W., *Creating a Strategic IT Architecture Competency: Learning in Stages.* MIS Quarterly Executive, 2003. **2**(1): p. 31-43.

12. Sauer, C. and L.P. Willcocks, *The Evolution of the Organizational Architect.* Sloan Management Review, 2002(Spring): p. 41-49.

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

14. Morris, C. and C. Ferguson, *How Architecture Wins Technology Wars.* Harvard Business Review, 1993. **71**(2): p. 86-97.

15. McKay, D. and D. Brockway, *Building IT Infrastructure for the 1990s*, Nolan, Editor. 1989, Norton & Company: New York. p. 1--11.

16. West, J. and J. Dedrick, *Innovation and Control in Standards Architectures: The Rise and Fall of Japan's PC-98.* Information Systems Research, 2000. **11**(2): p. 197-216.

17. Richardson, G., B. Jackson, and G. Dickson, *A Principle-Based Enterprise Architecture: Lessons from Texaco and Star Enterprise.* MIS Quarterly, 1990. **14**(4): p. 385-403.

18. Byrd, T.A. and D.E. Turner, *Measuring the flexibility of information technology infrastructure: Exploratory analysis of a construct.* Journal of Management Information Systems, 2000. **17**(1): p. 167-208.

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

20. Iyer, B., G. Shankaranarayanan., and M. Lenard, *Model Management Decision Environment: A Web Service Prototype for Spreadsheet Models.* Decision Support System (forthcoming), 2004.

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

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

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

24. Nezlek, G.S., H.K. Jain, and D.L. Nazareth, *An integrated approach to enterprise computing architectures.* Communications of the ACM, 1999. **42**(11): p. 82-90.

25. Parnas, D.L., *On the Criteria To Be Used in Decomposing Systems Into Modules.* Communications of the ACM, 1972. **15**(12): p. 1053-1058.

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

27. Allen, B.R. and A.C. Boynton, *Information architecture: In search of efficient flexibility.* MIS Quarterly, 1991. **15**(4): p. 435-450.

28. Gasser, L., *The Integration of Computing and Routine Work.* ACM Transactions on Office Information Systems, 1986. **4**(3): p. 205-225.

29. Zachman, J.A., *A framework for information systems architecture.*, in *IBM Systems Journal*. 1987, IBM Corporation/IBM Journals. p. 454.

30. Venkatraman, V.N., C.-H. Lee, and B. Iyer, *Is Ambidexterity a Valuable Organizational Capability? An Empirical Test in Software Product Introductions, 1991-2002*. 2005, Boston University.

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

32. Zhao, L. and K. Siau, *Component-based Development using UML.* Communications of the Association for Information Systems, 2002. **9**(12): p. 207-222.

33. Yourdon, E., *Modern structured analysis*. 1989, Englewood Cliffs, N.J.: Yourdon Press. x, 672.

34. Gunter, C.A., *Abstracting dependencies between software configuration items.* ACM Trans. Softw. Eng. Methodol., 2000. **9**(1): p. 94-131.

35. Schilling, M., *Toward a General Systems Theory and its Application to Interfirm Product Modularity.* Academy of Management Review, 2000. **25**(2): p. 312-334.

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

37. Tillquist, J., J.L. King, and C. Woo, *A representational scheme for analyzing information technology and organizational dependency.* MIS Quarterly, 2002. **26**(2): p. 91.

38. Glaeser, E., et al., *Growth in Cities.* Journal of Political Economy, 1992. **100**(61): p. 1126-1152.

39. Prasad, P., *Symbolic Processes in the Implementation of Technological Change: A Symbolic Interactionist Study of Work Computerization.* Academy of Management Journal, 1993. **36**(6): p. 1400-1429.

40. Feldman, M.S. and J.G. March, *Information in Organizations as Signal and Symbol.* Administrative Science Quarterly, 1981. **26**(2): p. 171-186.

41. Ackoff, R.L., *Management Misinformation Systems.* Management Science, 1967. **14**(4): p. 147-156.

42. Simon, H., *The architecture of complexity*, R. Garud, A. Kumaraswamy, and R.N. Langlois, Editors. 2003, Blackwell: Malden, MA. p. viii, 411 p.

43. March, J.G., *Exploration and Exploitation in Organizational Learning.* Organization Science, 1991. **2**(1): p. 71-87.

44. Carlile, P.R., *A Pragmatic View of Knowledge and Boundaries: Boundary Objects in New Product Development.* Organization Science, 2002. **13**(4): p. 442-455.

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

46. Argyres, N.S., *Technology strategy, governance structure and interdivisional coordination.* Journal of Economic Behavior and Organization, 1995. **28**: p. 337-358.

47. Kling, R., *Social Analyses of Computing: Theoretical Perspectives in Recent Empirical Research.* ACM Computing Surveys, 1980. **12**(1): p. 61-110.

48. DeSanctis, G. and M.S. Poole, *Capturing the Complexity in Advanced Technology Use: Adaptive Structuration Theory.* Organization Science, 1994. **5**(2): p. 121-147.

49. Pinch, T.J. and W.E. Bijker, *The Social Construction of Facts and Artifacts*, in *The Social Construction of Technological Systems*, T. Pinch, Editor. 1987, The MIT Press: Cambridge, MA. p. 17-50.

50. Barley, S.R., *Technology as an Occasion for Structuring: Evidence from Observations of CT Scanners and the Social Order of Radiology Departments.* Administrative Science Quarterly, 1986. **31**(1): p. 78.

51. Orlikowski, W.J., *The Duality of Technology: Rethinking the Concept of Technology in Organizations.* Organization Science, 1992. **3**(3): p. 398-427.

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

53. Kling, R., *Cooperation, coordination and control in computer-supported work.* Communications of the ACM, 1991. **34**(12): p. 83-88.

54. Barabasi, A.-L., *Linked: The New Science of Networks*. 2002, Cambridge, MA: Perseus Publishing. 280.

55. Chidamber, S. and C. Kemerer, *A metrics suite for object oriented desing.* IEEE Transactions on Software Engineering, 1998. **20**(6): p. 476-493.

56. MacCormack, A., J. Rusnak, and C. Baldwin, *Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code*. 2005.

57. Akrich, M., *The De-scription of Technical Artifacts*, W.E. Bijker and J. Law, Editors. 1992, MIT Press: Cambridge, MA. p. 205–224.

58. Banker, R.D., et al., *Software complexity and maintenance costs.* Association for Computing Machinery. Communications of the ACM, 1993. **36**(11): p. 81.

59. Wilson, M. and D. Howcroft, *Re-conceptualising failure: Social shaping meets IS research.* European Journal of Information Systems, 2002. **11**(4): p. 236.

60. Hubbell, C.H., *An input-output approach to clique identification.* Sociometry, 1965. **28**: p. 377-399.

61. Katz, L., *A new status index derived from sociometric analysis.* Psychometrika, 1953. **18**(1): p. 39-43.

62. Uzzi, B. and J. Spiro, *Do Small Worlds Make Big Differences? Artist Networks and the Success of Broadway Musicals, 1945-1989.* Forthcoming.

63. Borgatti, S.P. *The Key Player Problem.* in *Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers*. 2003: National Academy of Sciences Press