# Managing Architecture Under Emergence: A Conceptual Model and Simulation


| David Dreyfus | Bala Iyer |
|---|---|
| Boston University School of Management | Babson College |
| 595 Commonwealth Ave | TOIM Division |
| Boston, MA 02215 | Babson Hall |
| | Babson Park, MA 02457-0310 |



**Abstract**

*The modern-day enterprise has an information system architecture that changes as a result of distributed decisions made in response to changes in the firm's environment. Architects have a difficult time making decisions about such systems under their control. To help them with this process, we develop a conceptual model of architecture as a network comprising of 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 set of 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.*

*IT management can influence the evolution of the network; 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 and network analysis provides a useful abstraction for understanding architecture. This simulation is validated with a case study at a financial services company. 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.*



**Keywords:** Architecture, networks, control points, simulation, conceptual model, case study.


## 1. Introduction

Complex systems are 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 the interconnections and interdependencies among those elements. Simon [1] continues by suggesting that the natural and efficient approach to managing complexity is to build nearly decomposable systems based on stable subsystems. This approach

minimizes the overall dependency between all elements in the system by isolating elements with high interdependency into modules and then minimizes the interdependency between those modules. If represented as a Design Structure Matrix (DSM) [2], there would be more elements along the diagonal than in other locations. 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.

Extant literature has identified many techniques to deal with complexity. Firms develop hierarchies to both buffer core subsystems [3] and differentiate, specialize and exploit local resources [4]. Firms also consider products and organizational structures as complex systems consisting of both components and an architecture describing their interconnectedness [5]. Under this view, successful firms identify their components and architectures and manage them appropriately.

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, in a dynamic world 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 [8-10]. Such an evolutionary, emergent process modifies the pattern of interdependencies we've previously described.

In this work we conceptualize architectures as social systems and use networks to represent them. In particular, we define architecture as the pattern of connections between components and observe economic, organizational, product, and service architectures. Firms influence the

emergence of these architectures through managerial action; however, no manager (or firm) has enough power to control any but the smallest architecture. 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, decision makers 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 (ACPs). In this paper we develop the concept of ACPs 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. Architecture

There are multiple perspectives on what constitutes IS architecture [11]. Architecture has been viewed strategically [11-16], organizationally [17-21], and technologically [22-26]. 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 [13, 19, 21, 27]. Architecture, as implemented through its IT infrastructure, should be flexible, reliable, robust, scalable, and adaptable [18, 19, 28]. It should support the reuse of business components within a firm, while supporting firm responsiveness, innovativeness, and economies of scope [28].

Architecture implementation involves learning effects. As researchers have explored organizations making changes to architecture, they have identified two strategies: localized

exploitation or enterprise-wide integration [29]. In addition, they have identified that changes do not occur in one step; they occur in stages [11].

Architecture 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 [30], supported by the information system, with the business objectives of the organization.

Zachman expands the concept of information system architecture to include the various perspectives taken during its design and implementation [31]. Iyer and Gottlieb [20] identified three views through which we can examine architecture. The first view - the espoused - is the outcome of designing the planned dependencies between system modules and is the province of the IS Architect (although there may be many stakeholders). The second view - the emergent - is the outcome of implementing individual projects. It is a descriptive view of the actual dependencies that exist amongst system modules. The third view - the in-use - 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.

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 [32]. 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.

## 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 [33, 34]. As the architecture changes – emerges – over time, so do the dependencies between the systems.

The emergent information system reflects the interdependencies between the tasks and groups and serves as an integration mechanism [4] between groups differentiated to manage complexity [35, 36]. The existence of certain information conduits implemented in the IS system influences communication and routines [10] between the groups both syntactically and semantically [37]. The evolution of the information system also reflects different power relationships and conflicts between stakeholders [38].

The information system reflects the debate between organizational elements [39-42]. 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.

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.

IT project cost and success, and system flexibility, robustness, adaptability, and performance are the result of an interaction between system and organization that system architecture influences [34]. Project cost and maintenance is directly related to complexity [43], which is a description of interdependency.

We utilize 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, affect IT project cost, and improve IS characteristics. Network analysis can help us understand the emergent architecture and the shifting influence of components within it, and the implications for architectural control.

## 4. Architecture as emergent

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. The emergent architecture is the actual allocation of computational tasks to computational platforms that

results from the original design (espoused view) realized and modified through IT implementation projects as the system and organization adapt to each other [34, 44, 45].

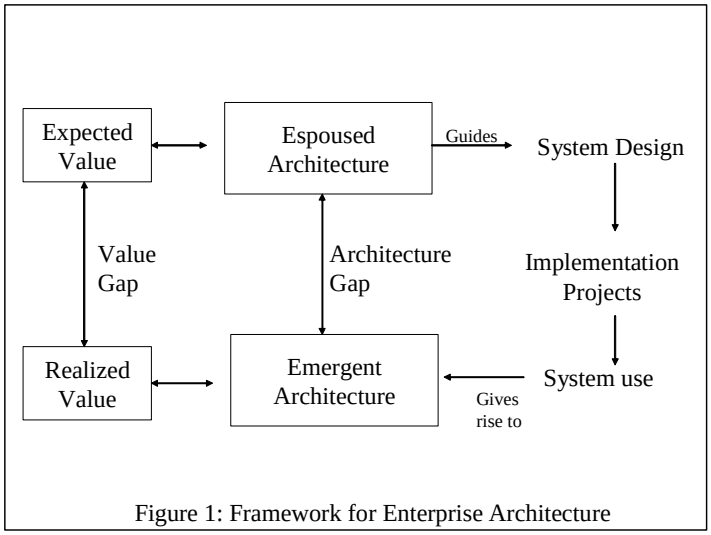


Figure 1: Framework for Enterprise Architecture

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 may be 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. 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 (ACPs)*. 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 [46] 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.

## 5. Network Analysis

Network analysis has been utilized in a great many areas to describe many complex phenomena [47]. Software researchers have used metrics [48] to assess the structural properties of software and have measured complexity using techniques such as DSM [49]. We are introducing the concepts of network analysis into the examination of architecture because network analysis enables us to model the interrelationships between components and treat those relationships as first-class units of observation.

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 by their network position within the espoused architecture.

One of the characteristics of 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 this abstraction, we utilize a single node type and a single edge (link) type. The emphasis is on the relationship between nodes and not on specific business functions, APIs, or tasks.

## 6. Systems

In this paper a component is an executable software application with a well-defined and published interface. As such, it may be used by other programs without the user 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 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. Similarly, multiple copies of the same application are 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 develop it independently within a firm.

## 7. Dependencies

One of the most important tasks in network analysis is defining what constitutes a link. The links between nodes represent conduits through which information, ideas, and knowledge flows. 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. The implication of direct and indirect dependency in the context of nearly decomposable systems [35] 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 separating 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 network literature, can be described through *influence centrality* [50, 51].

System developers implementing changes to the information system must confront these dependencies when modifying the system. 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-1 other components requires a maximum of N*(N-1) negotiations. This includes the negotiations between the target component and its directly connected components, plus the negotiations among those components.

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 component. Adding a new component to the network also shifts the relative importance of existing components within the network.

Integration between two existing component increases the number of dependencies on each component by one. It also increases the number of indirect dependencies between components and may shorten the shortest path between pairs of components. As links are added to the network, the balance of influence among components 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 components or modifying existing components 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 [46, 52]. 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.

To the list of modular operators identified by Baldwin and Clark - splitting, substituting, augmenting, excluding, inverting and porting - we add *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).

## 9. Simulation

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 the emergence of an information architecture. To increase our insight into emergence, we test these propositions:

**Proposition 1**: Emerging architectures result in reduced 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 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 position of the node in the network is ignored when evaluating the probability of an operation being permitted (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 – resource sharing. 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.

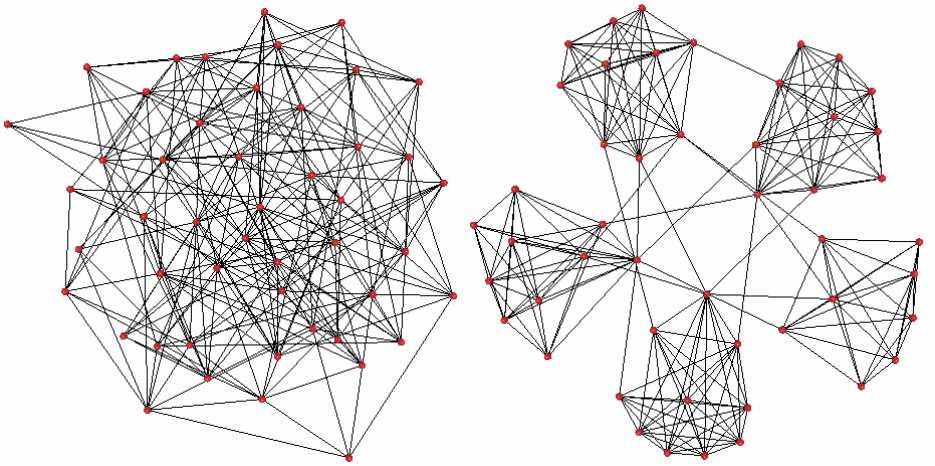

Figure 2: Networks before emergence. The random network is on the left; the Small World network is on the right.

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 to facilitate subsequent analysis.

Network evolution follows the following logic. The simulation is run until 100 operations are performed on the network. Three types of project are possible – Linking, Augmenting, and Splitting – with the probability of project implementation set at 50%, 30%, and 20% respectively. At each clock tick IT projects are proposed until one is accepted. *Linking* projects integrate two randomly selected components (nodes) by *linking* them with a bi-directional edge. *Augmenting* projects create a new component (node) and link it to a randomly selected existing component. *Splitting* projects re-architect a single component into two components. When the existing component is split, the existing integrations (links) are randomly distributed between the old and new components (nodes).

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 by comparing the starting and ending networks' Small World Quotients [53]. For each simulation, a number of standard measures such as network density and node counts are collected. Measures specific to this simulation included the KeyPlayer metric KPP-POS [54], used to measure the fitness of a set of ACPs (described below), and the Small World Quotient. We compute the difference between two networks using the Hamming distance.

In the simulation, ACPs are those components that collectively have the shortest path lengths (geodesics) to all other components. We assume that one component's influence on another component 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 find the ACP set that *maximizes the sum of reciprocal distances*. 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 [54], 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. The fitness measure at the beginning of the run is the starting fitness measure. At the end of the run we calculate the ending fitness of the starting ACP set. The difference between the starting fitness and the ending fitness represents the loss of architectural control. We also calculate the best fitness that could be obtained with a new set of ACP components and the overlap between the ACP set that

represents the best fitness and the original ACP set. All measures are normalized by dividing them by the number of components in the network.

**Summary of results**

The first key findings from this research suggest that, although the starting conditions are important, controlling the evolution of the information system architecture – acting on the emergent architecture rather than the espoused architecture – has the greatest impact on architectural control. 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 the espoused design is not completely lost through the emergence process.

In all cases, the best ACPs after emergence have a lower fitness than the starting fitness because components don't just attach to the ACP components. In addition, the best ACPs after emergence are different from the originals; IS Architects could improve their influence over the network by changing their ACPs.

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, rather than the initial network design, is a significant predictor of ending fitness.

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 of ACP Overlap 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 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 decision makers 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. The continuous arrival of new components 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. The systems that started with a small world structure and emerged through a process guided by architectural thinking remained closest to the service oriented design they started with.

## 10. FinServ Case Study

To demonstrate modeling fidelity and to use the model to construct and evaluate meaningful change management strategies or policies, we present a case study. We use the case study to modify our simulation model and to test if the results are still valid. The case study was used to specify the types of changes that can be made (design moves), the logic used to make architectural decisions, and the number of nodes an architectural group can control. These

factors were then incorporated into the original simulation model to create the new simulation model.

FinServ is an organization that provides processing services to the investment management industry. In addition to our own primary data collection, we used details listed in [55] to inform our case study. FinServ has over a trillion dollars in assets under management. They provide full service transfer agency and accounting services globally.

Over the last couple of decades, FinServ has grown rapidly primarily through mergers and acquisitions, and also by launching new services and entering new regions. As companies were acquired, each line-of-business (LOB) maintained much of the decision rights to control most aspects of their (line-of) business, including design, sales, back-end processing and IT services. The resulting enterprise architecture has duplication of functionality, limited integration, and a wide variety of user interfaces. Each LOB was empowered to focus on, and independently respond to, specific opportunities and needs.

The FinServ architecture had several characteristics that prevented FinServ from quickly responding to changing market conditions. These characteristics included tight coupling between systems, monolithic solutions (often not well documented), and the duplication of functionality (as noted above). This was sustainable during the economic growth phase of the 90’s.

In response to this predicament, FinServ hired a new CIO and created a new corporate initiative to address the problems. The IT organization was changed, creating an enterprise architecture team, a management oversight committee, and an organization to provide company-wide shared application services. A FinServ enterprise architecture strategy was created and a Global Platform, SOA-based, enterprise architecture was defined. Their core

mandate was to reduce IT spending though the elimination of redundant systems, componentization and exposure of core functionality through accessible services, redesign of strategic shared applications and components and the provision of mutually consistent customer access channels (web, voice, wireless, etc.). This organization invests in a strong enterprise architecture organization and actively practices architecture-driven application development with accompanying architecture governance via an Architecture Review Board. This involves investing in technology, processes, people, and organizational structures that can help them deliver on their mandate.

**Type of moves**

Our first sets of interviews were meant to validate the design moves that we used in the old simulation model. The chief architect of FinServ, who has played that role for the last five years, identified the following operations that a project could implement.

1. **Linking.** Adding an integration between two existing systems to support new functionality. An example of this is the creation of a new report that requires access to previously un-accessed data.
2. **Augmentation.** Adding a new system because a line of business (LOB) has a new requirement. A FinServ example is the need to maintain a Sarbanes-Oxley requirement. The new project involves the building or buying of a new application and the creation of integration points with existing systems.
3. **Splitting.** Re-architecting an existing system. As a result of multiple integrations the functionality of the system increases beyond its original scope. It becomes too complex to support the diversity of integrations. At some point, the designers re-architect the system so that its components can evolve semi-independently. In

practice, this situation seems to occur in at least three variations. In some cases a system (e.g., at FinServ, legacy mainframe systems) is enhanced over time to a point where further enhancement becomes increasingly difficult due to the interdependencies between functions. In a second set of cases, the re-architecting occurs because the company acquires another company with similar functionality, and the similar applications need to be rationalized. To merge the distinct functions and eliminate the duplicates, the original system may need to be split into constituent pieces. In a third set of cases, systems are cloned because the original system (e.g., a front-end system) is application specific (e.g., to a particular backend-system), yet its general functionality is desired across multiple applications. Cloning a system and then modifying it to support a new purpose may be quick in the short-run, but it doesn't scale and it does impose long-term maintenance costs. The collection of the set of clones can be considered a single system (until each clone diverges beyond recognition) that can be re-architected into a single, flexible system in which the general functionality is maintained once, and application specific functions are isolated.

4. **Enhancements.** Change an existing application (i.e., component or system) without creating new integrations. These projects generally extend the functionality of the existing application to meet new business needs or modify the application because an underlying system (e.g., Oracle) is upgraded.

Systems and applications are sometimes retired and taken out of service. Generally, this happens in conjunction with the augmentation and re-architecting of existing applications and is properly accounted for in the splitting operation described above. In other cases, the retired

system is replaced by another system but the general pattern of integrations between systems – the sharing of data between systems – is unchanged. In yet other cases, the retired system is eliminated because the supported business function is no longer performed, or the client no longer exists. This function is outside the scope of the simulation.

At FinServ, the rough breakdown of projects by resources used is: 25% for linking, 25% for augmentation, 10% for splitting, and 40% for enhancements. In our simulation, we ignore enhancements because they do not change the enterprise architecture. However, to the extent that the architecture affects the cost and effectiveness of subsequent enhancements, it is significant that 40% of the firm's resources go towards enhancements. Scaling the remaining 60% of the resources that affect the enterprise architecture to 100%, the breakdown of operations is: 42% linking, 42% augmentation, and 16% splitting. We use this calibration for our FinServ-based simulation model.

**Decision logic for architecture**

In our first simulation model, we adopted one of two types of decision logic that influence project acceptance. Firms either execute projects without regard to the long-term impact of the project on system architecture or they evaluate projects on both the short-term business objectives and the long-term architectural impact. The choice of decision logic is influenced both by the costs of engaging architects for evaluation and the history and experience of the business leadership.

At FinServ some of the leadership recognizes the importance of architectural thinking. Other elements of the leadership are focused on short-term objectives and the need to meet the immediate concerns of customers. Finally, the formative experiences of some of the leadership were of a time and place where applications were far less integrated. If the applications are

isolated, thinking of them as such is perfectly appropriate. Problems only seem to exist when the leadership's mental map diverges from practice.

We recognize that firms execute both types of project evaluation logic. However, to illustrate the impact of these two views presented above, we run one simulation under the assumption that architecture is not considered during project evaluation and we run another simulation when architecture is considered during project evaluation.

*Isolated view*. In this logic, each project is evaluated on its own merit without regard to the impact on other systems or the evolution of the overall information system. The integrations between applications either supply valuable data or support important stakeholders. However, because the specific integrations do not affect project authorization, we assume that the integrations required for a project are statistically random.

*Architectural view.* In this logic, each project is evaluated both on its own merit and on its impact on the future architecture of the information system. In particular, projects that link systems that are designed or intended for re-use are given preference. By itself, this leads to systems in which the highly connected nodes become more highly connected. However, some new systems are designated as preferred systems to link to before becoming highly linked. This can occur because either the architects build or buy systems for that purpose (e.g., enterprise system bus components, data warehouses, or data marts) or acquire a new system (e.g., applications that exist in an acquired company that are to be retained) that must be integrated into existing systems.

In our new simulation, we incorporate this logic by making the probability of a link (integration) existing between any to nodes (systems) a function of the degree centrality (number of integrations supported by a system) of each node. To represent the idea that some

systems are to be given a higher preference for integration, even if they currently have few integrations, we randomly assign a few systems (as they enter the enterprise architecture) the property that projects that contain a proposal to link to them are always approved.

**Architectural control**

A key assumption in this paper is that architects cannot actively manage all the systems in an enterprise. They, therefore, select a few and let the decentralized LOB managers handle the rest. Within FinServ there are about 150 IT projects per year, of which the architecture team has some visibility into 20%. The number of projects that the architects get actively involved with is limited by their staffing level to 8.

Architects select the systems they monitor based upon what they think is important. Another key assumption in this paper is that certain systems are valuable not because of their data or functionality, but because of their position in the network of systems within the enterprise.

This assumption is borne out by the following example. FinServ has three mainframe systems that feed data into a data warehouse. As the data are fed into the warehouse they are transformed. This leads to data quality issues due to conversion, completeness, and timeliness problems. The warehouse also feeds data to a data mart to provide efficient reporting to a fourth tier of applications. The transfer from the warehouse to the data mart also involves data transformation and quality issues. Because the warehouse data is suspect, the data mart data is also suspect. Moreover, there aren’t enough data marts. As a result, there is pressure from the LOB managers that use the DSS applications to bypass the data marts and warehouses and go directly to the mainframe data sources. However, this would severely strain the mainframes and would limit subsequent modification of the mainframe applications because of the constraints imposed by dependent systems.

The warehouse has some autarky value due to its data processing and transforming capabilities, but its more significant value is in its position between the mainframe applications and downstream data consumers. By focusing on this positionally important system (i.e., the data warehouse) the architects can better preserve the architecture of the entire information system. It is also by controlling the data definitions in the warehouse that the architects can influence how the data is interpreted in downstream applications.

Based on the above description of the FinServ, we have made changes to our original simulation model to include the new logic for architecture. As shown in Figure 1, FinServ's systems are not isolated: FinServ has 158 interconnected systems (nodes) in their enterprise architecture (network). It has a few highly integrated systems and many sparsely integrated systems. To account for the larger starting size (158 nodes) we increased the number of nodes added during the simulation to 320, thus maintaining the ratio of new nodes to original nodes used in the earlier simulation.

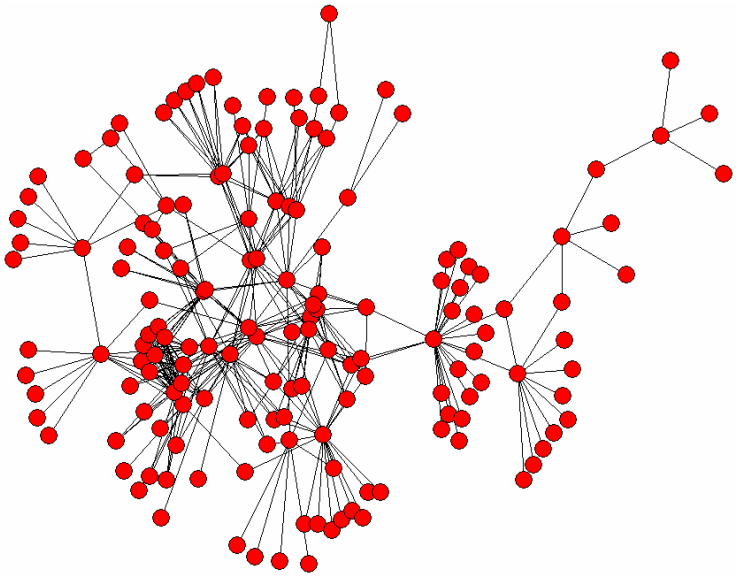

**Figure 1. FinServ enterprise architecture (main component). 158 nodes.**

| 1.<br>Decision logic | 2.<br>Probability new system marked special | 3.<br>Delta Original Fit | 4.<br>Delta New Fit |
|---|---|---|---|
| Isolated view | N/A | -0.30 | -0.24 |
| Architectural view | 0 | -0.34 | -0.23 |
| | 20 | -0.34 | -0.27 |
| | 40 | -0.32 | -0.28 |
| | 60 | -0.33 | -0.29 |
| | 80 | -0.33 | -0.28 |
| | 100 | -0.33 | -0.29 |

**Table 1: Simulation results. N = 10 for each starting condition. 320 projects implemented on 158 node enterprise architecture. Average values displayed.**

The simulation suggests that growth in the enterprise architecture results in a decrease of control for the original control points (column 3). The decrease in control of the original control points is worse if architectural considerations are taken into account in approving projects, but a new set of control points (column 4) can minimize the loss of control when architectural considerations are taken into account. This conundrum is explained by noting that without architectural control, the original positionally important nodes retain their position because no other node become more positionally important; in addition, no alternate set of control points produces particularly good control. By contrast, when architectural considerations are taken into account, the old control points lose their importance because other important control points have emerged. In a set of regression studies we found the difference in control due to the decision logic used to be statistically significant.

Seemingly random incorporation of new important nodes into a system (column 2) reduces the ability of architects to maintain control. The degradation of control applies to both the original set of control points and any alternate set of control points. In a set of regression studies we found that the level of control a set of control points offers is predicted by the

number of control points employed and is inversely predicted by the number of seemingly random incorporated systems designated as important.

In a set of regression studies (not shown) we found that the number of control points is positively correlated (and statistically significant) with both the loss of control for the original control points and the gain of control when a new set of control points is adopted. In effect, architects that pay attention to only a few systems do not experience as much of a loss of control as those that pay attention to more systems because their absolute level of control is lower. Architects that pay attention to more systems also need to be willing to change the systems they monitor to maintain control.

The biggest challenges that FinServ's architects have, however, is not within their technical domain. Instead, the problem is explaining to senior managers why IT is expensive, why failing to invest in architecture will become increasingly expensive, and why outsourcing the support and provision of applications does not address the failure to invest in architecture. Outsourcing may affect where the application resides and who is responsible for its maintenance, but it does not affect the integration of that outsourced system with the other enterprise systems that either feed or require data.

It is only by staying on top of the applications that drive the number of, and complexity of, the integrations between systems that FinServ can manage the cost, flexibility, and risks in their evolving enterprise architecture.

The simulation emphasizes the emergent nature of the enterprise architecture and the need to adjust the firm's resources in response to changes in the architecture. As the case at FinServ highlights, the challenges are not strictly technical. The biggest challenges are communicating the nature of the information system, gaining the necessary resources, and identifying risks.

Systems with high positional value may be important because they influence many other systems. They can also be important because they are outside the firm's control. At FinServ, this point is illustrated with two examples. Some of the systems with high positional value are 3rd party applications. This makes FinServ dependent on another companys' systems. At least one key system was developed twenty-plus years ago by two consultants now in their sixties that are still responsible for its maintenance. The system is undocumented. In both of these cases FinServ has positionally important nodes effectively outside of its control. When these systems were either stand-alone systems, or used by only one other application, the risks associated with these systems may have been low. However, as these systems became more embedded through direct and indirect integrations, their positional importance and associated risks increased.

## 11. Conclusions, limitations and future work

According to the chief architect at FinServ, the conceptual model, network visualization and associated simulation is "a powerful communication tool to gain the support of people to do the right thing." The big issue is being able to visualize what is happening to the architecture and then being able to show others without "wading through reams of spreadsheet data." "Communications with decision makers and stakeholders at a senior level is notoriously difficult. They live in an old world where things were not complicated." The decision makers take an attitude that, "IT is difficult. So, I will buy it from outside. There is still very much the silo model."

From FinServ's perspective, the simulation would serve them better if it were further calibrated with specific project costs and system performance metrics to better illuminate the effect of architectural control on cost, system stability, and system efficiencies. As the chief

architect stated, “Everyone wants to reinvent the wheel. We need metrics for the cost of adding systems, transforming data, and moving data.”

This simulation highlights the effect of system evolution on the changing importance of positionally important systems. Adhering to a preference for system reusability can limit the speed with which list of important systems changes, but it can’t halt change.

Subsequent simulations and empirical examinations can tie system evolution and architectural guidance to specific business and IS outcomes. We anticipate showing that the question asked of the FinServ architect – “When will the architecture be done” – is the wrong question. The architecture is done when the business is closed. Architecting is a process that maintains some level of control over the architecture and produces better outcomes for the organization than would occur in its absence.

In this research we asked the question, can we meaningfully model and represent the complexity of – and changes to - information system architecture in network terms, and then inform decision making through such representation? Our approach was to represent the components of the architecture as nodes, and the dependencies between the components as links or arcs. The answer seems to be, “Yes.”

We focused 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 a system of emergent interdependencies 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 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 to it 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 the ability to align an information system 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 developed the paper based on an 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 used 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 used ACPs identified in the design stage to track changes in the production stage. While we showed 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.

**References**


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

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

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

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

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

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

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

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

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. and C. Szyperski, *Software Ecosystems: Understanding an Indispensable Technology and Industry*. 2003, Cambridge, MA: The MIT Press. 424.

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. Larman, C., *Applying UML And Patterns*. 3 ed. 2005, Upper Saddle River, NJ: Pearson Education, Inc. 703.

27. Kayworth, T.R., D. Chatterjee, and V. Sambamurthy, *Theoretical Justification for IT Infrastructure Investments.*, in *Information Resources Management Journal*. 2001. p. 5.

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

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

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

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

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

33. 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.

34. Leifer, R., *Matching Computer-Based Information Systems with Organizational Structures.* MIS Quarterly, 1988. **12**(1): p. 63-73.

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

36. 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.

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

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

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

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

41. 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.

42. 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.

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

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

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

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

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

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

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

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

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

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

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

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

55. Westerman, G. and R. Walpole (2005) *PFPC: Building an IT Risk Management Competency*. CISR Working Paper **Volume**, DOI: 352