A Methodology for Operationalizing Enterprise Architecture and Evaluating Enterprise IT Flexibility

HARVARD | BUSINESS | SCHOOL

A Methodology for Operationalizing Enterprise Architecture and Evaluating Enterprise IT Flexibility

Alan MacCormack
Robert Lagerstrom
David Dreyfus
Carliss Y. Baldwin

Working Paper

15-060

April 21, 2015

A Methodology for Operationalizing Enterprise Architecture and Evaluating Enterprise IT Flexibility

Alan MacCormack[†]
Robert Lagerstrom*
David Dreyfus**
Carliss Y. Baldwin[†]

[†] Harvard Business School
amacormack@hbs.edu
cbaldwin@hbs.edu

* KTH Royal Institute of Technology, Sweden
robertl@kth.se

** Consulting Data Scientist at
Massachusetts State Auditors Office
https://www.linkedin.com/in/daviddreyfus

A Methodology for Operationalizing Enterprise Architecture and Evaluating Enterprise IT Flexibility

Abstract

We propose a network-based methodology for analyzing a firm's enterprise architecture. Our methodology uses "Design Structure Matrices" (DSMs) to capture the coupling between components in the architecture, including both business and technology-related elements. It addresses the limitations of prior work, in that it i) is based upon the actual architecture "in-use" as opposed to planned or "idealized" versions; ii) identifies discrete layers in a firm's architecture associated with different technologies (e.g., applications, servers and databases); iii) reveals the main "flow of control" within an architecture (i.e., the set of inter-connected components); and iv) generates measures of architecture that can be used to predict performance.

We demonstrate the application of our methodology using a novel dataset developed with the division of a large pharmaceutical firm. The dataset consists of all components in the enterprise architecture, the observed dependencies between them, and estimated costs of change for software applications within this architecture. We show that measures of the architecture derived from a DSM predict the cost of change for software applications. In particular, applications that are tightly coupled to other components in the architecture cost more to change. The analysis also shows that the measure of coupling that best predicts the cost of change is one that captures all direct and *indirect* connections between components (i.e., it captures the potential for changes to propagate via all possible paths between components). Our work represents an important step

in making the concept of enterprise architecture more operational, thereby improving a firm's ability to understand and improve its architecture over time.

1. Introduction

As information becomes more pervasive in the economy, information systems in firms are becoming increasingly more complex. Initially, information systems were designed to automate back-office functions and provide data to support managerial decision-making. The role of these systems was expanded to coordinate the flow of production in factories and supply chains. The invention of the personal computer led to the creation of client-server systems, which enhanced the productivity of office workers and middle managers. Finally, the arrival of the Internet brought a need to support web-based communication, e-commerce, and online communities. Today, even a moderate-size business maintains information systems comprising hundreds of applications and many databases, running on geographically distributed hardware platforms, serving multiple clients. These systems must be secure, reliable, flexible, and capable of evolving when new opportunities arise.

"Enterprise architecture" (EA) is the name given to a set of frameworks, processes and concepts that are used to manage an enterprise's information system infrastructure. For example, TOGAF®, the most-cited framework in this field, was developed by a consortium of firms to provide a standardized approach to the design and management of information systems within and across organizations (TOGAF, 2009).[1] It provides a way of visualizing, understanding and planning for the needs of diverse stakeholders in a seamless and cost-effective way. Unfortunately, despite the increasing adoption of EA frameworks such as TOGAF by firms,

[1] http://pubs.opengroup.org/architecture/togaf9-doc/arch/ (viewed 11/3/14).

making changes to systems, adding new functionality, and/or integrating different systems (e.g., as in a merger) are often not straightforward tasks. The empirical evidence for this, and EA research in general, is mixed at best (Buckl et al., 2009; Seppänen et al., 2009; Dietz & Hoogervorst, 2011). Changes made to one component can create unexpected disruptions in seemingly distant parts of the enterprise (Vakkuri, 2013). In essence, when dealing with complex systems, *changes propagate in unexpected ways*, increasing the costs of adaptation and reducing flexibility. This suggests the need for a method to better operationalize enterprise architecture, in a way that generates a deeper understanding of the actual design "in-use."

Several EA frameworks have proposed using *matrices* to display the relationships among various components of an information system, in an attempt to make EA more operational (e.g., TOGAF, 2009). Unfortunately, it is not clear how these matrices should be combined to generate specific managerial insights (e.g., how to improve the system). Furthermore, the information used to construct them reflects an idealized view of how a system *should* function, rather than data on how it *currently* functions. Finally, these matrices do not yield quantitative measures of architecture that can be used to analyze performance; they are primarily descriptive, not prescriptive, in nature. This paper seeks to address these challenges.

Specifically, prior work has demonstrated the efficacy of using a Design Structure Matrix or DSM—a square matrix that captures the interactions among components—as a tool for visualizing, measuring and characterizing the architecture of complex products (Steward, 1981). We apply this DSM methodology to analyze a firm's information systems architecture, which comprises many interdependent elements, including business groups, applications, databases and hardware. Our data is drawn from work with a large pharmaceutical company (Dreyfus, 2009). We use this data to i) describe how an enterprise architecture DSM is constructed, ii) show that

this DSM reveals the layered structure of a firm's architecture, and iii) highlight how the components in this architecture can be classified into different groups – Core, Peripheral, Shared and Control – based upon the way that they are coupled to the rest of the system. The Core, Shared and Control components comprise the main "flow of control" in the architecture.

We demonstrate the application of our methodology by analyzing the impact of different levels of component coupling on the cost of change for software applications. Specifically, we use multiple measures of coupling derived from the enterprise architecture DSM to predict the cost of change. The results show the cost to change "Core" applications, which are tightly coupled to other system components, are significantly higher than the cost to change "Peripheral" applications, which are only loosely connected to other components. We find that the measure of coupling which best predicts the cost of change is one that captures all of the *direct and indirect* connections between components in the architecture. In sum, it is important to account for all the possible paths by which changes may propagate between two components, even if these components are not directly connected.

The main contribution of this paper lies in developing an operational methodology for analyzing enterprise architecture that addresses the limitations of prior work. Specifically, we show how dependency-matrices, which have previously been applied to the study of product architecture, can be used to gain insight into enterprise architecture. We demonstrate the application of our methodology by analyzing a novel dataset from a real firm, encompassing comprehensive information about all system components and the interdependencies between them. We conclude by relating our findings to prior literature and discussing the implications of our methods for practicing managers.

The paper is organized as follows. Section 2 reviews the literature and motivates our approach. Section 3 introduces our dataset and describes how an enterprise architecture DSM is constructed. Section 4 shows how this DSM is used to classify components into groups based upon their coupling within the system. Section 5 demonstrates the application of our methodology to predicting the cost of software change. Section 6 describes our conclusions.

2. Literature Review and Motivation

This section reviews the enterprise architecture literature, with the aim of understanding the limitations of current approaches, and the criteria by which new methods should be assessed. We then describe recent work on the visualization and measurement of complex software systems using network-based approaches, with a focus on the use of DSMs.

2.1 Enterprise Architecture

Enterprise Architecture is commonly defined as a tool for achieving alignment between a firm's business strategy and its IT infrastructure. For example, MIT's Center for Information Systems Research defines EA as "the organizing logic for business processes and IT infrastructure reflecting the integration and standardization requirements of the company's operating model" (Weill, 2007). Prior work tends to emphasize conceptual models, tools and frameworks that attempt to achieve this alignment (e.g., Aier and Winter, 2009). EA analysis is not limited to IT systems, but encompasses the relationship with and support of business entities.

This overarching perspective is present in the ISO/IEC/IEEE 42010:2011 standard, which defines architecture as "the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution" (ISO/IEC, 2011). Ultimately, EA targets a holistic and unified scope of organization

(Rohloff, 2008; Tyler, 2006). Hence most research has focused on the "strategic" implications of EA efforts (Tamm, 2011; Aier, 2014, Boh & Yellin, 2007; Ross & Weill, 2006).

If the integration of IT and business concerns is one defining aspect of EA, a model-based methodology is another. As the name hints, architectural descriptions are central in EA. These descriptions include entities that cover a broad range of phenomena, such as organizational structure, business processes, software and data, and IT infrastructure (Lankhorst, 2009; Winter & Fischer, 2007; Jonkers, 2006). A large number of frameworks have been proposed, detailing the entities and relationships between them that should be part of this effort (e.g., TOGAF, 2009; Lankhorst, 2009; DoDAF, 2007; MODAF 2008). However, considerable diversity exists, in terms of the primary unit of analysis and terminology adopted by each. For example, various frameworks focus on i) Stakeholders and Aspects to be considered (Zachman, 1987); ii) Viewpoints and Concerns to be analyzed (TOGAF, 2009); and iii) Objects and Attributes to be modeled (Lagerström et al., 2009). This lack of consistency is likely one reason for the limited success reported for EA efforts in studies of practice (Roeleven, 2010).

EA frameworks have been shown to be a useful decision-support tool when focused on the needs of specific decision-makers (Johnson and Ekstedt, 2007). For example, researchers at the KTH Royal Institute of Technology have applied a uniform methodology to model how EA affects the dimensions of security, interoperability, availability, modifiability and data accuracy (Sommestad et al., 2013; Ullberg et al., 2011; Franke et al., 2014; Lagerström et al., 2010; Närman et al., 2011). These narrower models help stakeholders understand how EA affects specific performance attributes, generating insight into current and future states. Nevertheless, these models remain difficult to understand and implement, limiting their practical impact.

Several EA frameworks have proposed using *matrices* to display the relationships among various components of an information system, in an attempt to make EA more operational. For example, TOGAF recommends preparing nine separate matrices at different points in the development of a firm's architecture,[2] which are used to track the linkages and dependencies between various parts of the system. Unfortunately, it is not clear how these matrices should be combined to generate managerial insights (e.g., how to improve). Furthermore, the information used to construct them reflects an idealized view of how a system *should* function, rather than how it actually functions. Finally, these matrices do not yield quantitative measures of architecture that can be used to analyze performance; they are primarily descriptive in nature.

In sum, operationalizing enterprise architecture in a robust and reliable way, that allows firms to analyze and improve their systems, has proven an elusive goal. A diverse range of frameworks exists, each employing different units of analysis and terminology. Furthermore, these frameworks are conceptual in nature, generating few quantitative measures that can be used to analyze performance. Finally, these frameworks focus on idealized versions of a firm's architecture, rather than actual data from the architecture "in-use." While some recent studies attempt to operationalize EA at a granular level (e.g., Mocker, 2009; Dreyfus and Wyner, 2011), there is as yet little consensus on a general *methodology* for how this should be achieved.

2.1.1 Enterprise Architecture and "Layers"

While there exists a diverse range of enterprise architecture frameworks, one of the themes they have in common is the concept of "layers" (Adomavicius, 2008; Yoo et al, 2010; Simon et al, 2013). Simon et al. (2013) describe enterprise architecture management as dealing with different layers, including business, information, application, and technology layers. Yoo et

[2] http://pubs.opengroup.org/architecture/togaf9-doc/arch/chap35.html (viewed 11/3/14)

al. (2010) argue that pervasive digitization has given birth to a "layered-modular" architecture, comprising devices, network technologies, services and content. Finally, Adomavicius et al. (2008) discuss the concept of an IT "ecosystem," highlighting the different roles played by products and applications, component technologies and infrastructure technologies.

While studies differ in the ways that they classify layers in a firm's enterprise architecture, they do share common underlying assumptions. First, layering reflects a division of the functions provided by a system into units, such that these units can be designed, developed, used and updated independently. Second, layering establishes a design "hierarchy" (Clark, 1985) such that each layer tends to interact only with layers immediately above or below it, reducing complexity. Finally, the direction of interdependencies between layers is such that higher layers "use" lower layers, but not the reverse, limiting the potential for changes to propagate (Gao and Iyer, 2006). For example, a software application on a desktop computer "uses" (i.e., depends upon) functions provided by the operating system layer below it. However, the operating system does not (in general) depend upon the applications that use it. This has important implications for the propagation of changes. Changes to the operating system may impact applications, but changes to applications will not, in general, impact the operating system.

The importance of layering in the literature suggests that any methodology for operationalizing EA should be able to identify the layered structure of the enterprise architecture.

2.1.2 Enterprise Architecture and Firm Performance

Much of the literature on EA has focused on frameworks that align business needs with IT capabilities and the processes by which such frameworks are implemented. Surprisingly however, there has been little work to explore the performance benefits of EA, using empirical

data on the actual outcomes achieved by firms. Indeed, Tamm et al. (2011) found that of the top 50 articles on enterprise architecture (as ranked by citation count) only 5 provided any empirical data that sought to explain the link between EA efforts and improved performance outcomes.

Many authors note it is difficult to directly assess the quality of a firm's architecture. Hence empirical studies linking enterprise architecture to performance tend to focus on assessing the quality of *outputs* from EA planning processes (e.g., the quality of the documentation) or the quality of the EA planning *process* itself (e.g., how effectively did the firm set goals, define tasks and govern the effort) (Kluge et al., 2006; Aier et al., 2011; Lagerström et al., 2011; Schmidt and Buxmann, 2011). This approach means that it is difficult to differentiate between firms that follow similar EA planning processes, but which arrive at different outcomes. A more robust method for operationalizing EA should be able to discern between such situations.

Studies that make claims about the performance benefits of EA tend to cite a range of "enablers" that mediate firm outcomes. Recurring themes include better organizational alignment, improved information quality and availability, optimized resource allocation across the business portfolio, and increased complementarities between resources (Tamm et al., 2011). The most consistent theme that emerges in the literature however is the role of EA in facilitating *flexibility*. In an influential paper, Samburmathy at al (2003) argue that the strategic value of information technology investments in firms is defined by their impact on agility, creating "digital options" and "entrepreneurial alertness" (i.e., understanding and exploiting new opportunities). Duncan (1995) explores the factors that contribute to flexibility, showing that managers associate this feature with the attributes of compatibility, connectivity, and modularity. Schmidt and Buxmann (2011) measure these four attributes, and show that a rigorous and comprehensive enterprise architecture planning process is associated with self-reported

improvements on each one. Finally, Sambamurthy and Zmud (2000) suggest the new organizing logic for EA is the "platform," which encompasses a "flexible combination of resources, routines and structures" that meets the needs of both current and future IT-enabled functionalities.

This discussion suggests that any methodology for operationalizing EA should capture data on the architecture in-use by a firm, not merely the processes and documents by which it was developed. Furthermore, the measures output from this methodology should facilitate the analysis of important outcomes, including the extent to which the architecture enables flexibility. For this reason, we chose to demonstrate the application of our methodology by analyzing the cost of change for the software applications within a firm's enterprise architecture.

2.2 Network-based Approaches to System Architecture

Many prior studies have characterized the architecture of complex systems using network representations and metrics (Holland, 1992; Kauffman, 1993; Barabasi, 2009). In particular, they focus on identifying the linkages that exist between different elements (nodes) in a system (Simon, 1962; Alexander, 1964). A key concept that emerges in this literature is that of modularity, which refers to the way that a system's architecture can be decomposed into different parts. Although there are many definitions of modularity, authors agree on its fundamental features: the interdependence of decisions within modules, the independence of decisions between modules, and the hierarchical dependence of modules on components that embody standards and design rules (Mead and Conway, 1980; Baldwin and Clark, 2000).

Studies that use network methods to measure modularity typically focus on analyzing the level of coupling between different elements in a system.[3] The use of graph theory and network

[3] For software systems, this notion is linked with that of *cohesion* (Dhama, 1995). Well-designed software applications have high levels of cohesion (within modules) and low levels of coupling (across modules).

measures to analyze coupling in software systems has a long history (Hall and Preiser, 1984). Many authors show that measures of *direct* component coupling predict important parameters such as defects and productivity (Stevens et al., 1974; Henry and Kafura, 1981; Fenton and Melton, 1990; Chidamber and Kemerer, 1994; Briand et al., 1999; Allen et al., 2007). Despite this, no clear consensus has emerged on exactly how coupling should be defined in software systems, or which are the best coupling metrics to use for predicting performance. In recent years, a number of studies have adopted coupling measures derived from social network theory to analyze software systems (Dreyfus and Wyner, 2011; Wilkie and Kitchenham, 2000; Myers, 2003; Jenkins and Kirk, 2007). However, such measures suffer from well-known limitations that make their application to technical systems difficult to apply and interpret consistently. For example, social network measures tend to assume that dependencies are symmetric. In technical systems, many important dependencies are asymmetric, meaning the direction of coupling is important.

2.1.2 Design Structure Matrices (DSMs)

An increasingly popular network-based method used for analyzing technical systems is the "Design Structure Matrix" or DSM (Steward, 1981; Eppinger et al., 1994; MacCormack et al., 2006; Sosa et al., 2012). A DSM displays the structure of a complex system using a square matrix, in which the rows and columns represent system elements, and the dependencies between elements are captured in off-diagonal cells. Baldwin et al. (2014) show that DSMs can be used to visualize the "hidden structure" of software systems, by analyzing the level of coupling for each component, and classifying them into similar categories based upon the results.

Metrics that capture the level of coupling for each component can be calculated from a DSM and used to understand system structure. For example, MacCormack et al. (2006) and LaMantia et al. (2008) use DSMs and the metric "propagation cost" to compare software system architectures, and to track the evolution of software systems over time. MacCormack et al. (2012) show that the architecture of technical systems tends to "mirror" that of the organizations from which they have evolved. Sturtevant (2013) shows that software components with high levels of coupling tend to experience more defects, take more time to adapt and are associated with high employee turnover. And Ozkaya (2012) shows that metrics derived from DSMs can be used to assess the value released by "re-factoring" designs with poor architectural properties.

In a recent case study, Lagerström et al. (2013) applied DSMs to study a firm's enterprise architecture – in which a large number of interdependent software applications have relationships with other types of components, such as business groups, schemas, servers, databases and other infrastructure elements. In this paper, we formalize and extend this approach, then demonstrate a practical application of the method, by analyzing how measures from a DSM predict flexibility.

2.2.1 Design Structure Matrices and Change Propagation

A DSM captures all of the dependencies that exist between components in a system. If component A depends directly upon component B, then any change made to B may affect A. These two components are "coupled." But using a DSM, we can also analyze the *indirect dependencies* between components, which reflect the potential for changes to propagate in a system via a "chain" of dependencies. For example, if component B, in turn, depends upon component C, then a change to C may affect B, which in turn, might affect A. Therefore, A and C are also "coupled," but indirectly. The level of indirect coupling in a system provides an

indication of the degree to which changes can propagate through a system. Prior work has shown that measures of indirect coupling predict both the level of defects and the ease (or difficulty) with which a system can be adapted (MacCormack, 2010; Sturtevant, 2013).

A DSM is not the only network analysis technique that can reveal both direct and indirect dependencies between components. In contrast to techniques such as social network analysis however a DSM also captures information on the *direction of dependencies*. This distinction is important, given dependencies in technical systems are typically not symmetric. In the example above, A depends upon B, but that does not imply that B also depends upon A. As such, a change to B may propagate to A, whereas component A could be changed with no impact on B. A DSM captures the direction of dependencies, allowing us to determine the "flow of control" in a system (i.e., the direction in which chains of dependencies are likely to propagate). Hence we can discern between systems that are hierarchical in nature (i.e., there exists a strict ordering of components) versus those that are cyclical in nature (i.e., the components are mutually interdependent). Hierarchy and cyclicality are critical constructs for understanding how changes might propagate in complex systems. DSMs can be used to reveal these characteristics.

3. Constructing an Enterprise Architecture DSM

3.1 The Empirical Context

We illustrate our methodology by using a real-world example of a firm's enterprise architecture. The aim is to make these methods concrete and to demonstrate that they provide insight into how real world systems operate. Using real-world data also provides validation that our methods of data collection and analysis are able to scale for practical use in the field.

Our study site is the research division of a US biopharmaceutical company "BioPharma". At this company, "IT Service Owners" are responsible for the divisional information systems, and provide project management, systems analysis, and limited programming services to the organization. Data were collected by examining strategy documents, having IT service owners enter architectural information into a repository, using automated system scanning techniques, and conducting a survey. Details of the data collection protocols are reported in Dreyfus (2009).

Our BioPharma dataset includes information on 407 architectural components and 1,157 dependencies between them. The architectural components are divided into: eight "business groups;" 191 "software applications;" 92 "schemas;" 49 "application servers;" 47 "database instances;" and 20 "database hosts". These components form a layered architecture, typical of modern information systems, as we will show later. Note that "business groups" are organizational units not technical objects. The dependence of particular business groups on specific software applications and infrastructure is integral to studies of enterprise architecture. We consider business groups part of the enterprise architecture, and include them in our analysis.

We capture data on four types of dependency between components – uses, communicates with, runs on, and instantiates. Business units *use* applications; Applications *communicate with* each other, use schemas, and *run on* application servers. Schemas in turn *instantiate* database instances that run on database hosts. Importantly, of these four dependency types, "uses", "instantiates" and "runs on" possess a specific direction (i.e., they are asymmetric dependencies). In contrast, "communicates with" is a bi-directional (i.e., symmetric) dependency.

Dependency data for the BioPharma enterprise architecture was obtained using a combination of manual and automated methods. In particular, interviews were conducted with the IT director and surveys were conducted with IT Service Owners. This information was then

supplemented with the use of open-source and custom tools to monitor the server and network traffic in the system. Data on processes and communication links was then manually aggregated to the level of the individual component (Dreyfus, 2009). Importantly, many links discovered using automatic tools had been overlooked by or were unknown to the IT Service Owners. This indicates that the theoretical (i.e., documented) system architecture can deviate substantially from the actual architecture "in use," validating the broader motivation for our work.

Finally, for a subset of the software applications in the enterprise architecture, data was collected on the cost of making changes (discussed in Section 5). Prior work used these data to explore the predictive power of social network metrics (Dreyfus and Wyner, 2011). We depart from this prior work in that we i) introduce a formal methodology, based upon DSMs, by which to visualize and measure the firm's enterprise architecture, and ii) demonstrate the application of this methodology to analyzing the cost of making changes to the software applications, using measures of the firm's architecture that are derived directly from the DSM.

3.2 Constructing the DSM

A DSM is a way of representing a network. Rows and columns of the matrix denote nodes in the network; off-diagonal entries indicate linkages between the nodes. In the analysis of complex systems, the rows, columns, and main diagonal elements of a DSM correspond to the components of the system—in this case, business groups and technical resources (e.g., software applications, databases, hosts etc.). Hence the first question we must answer is what kinds of linkages between components should be captured, and how should these be counted?

The influential computer scientist David Parnas argued that the most important form of linkage is a directed relationship that he calls "depends on" or "uses" (Parnas, 1972). If B uses A,

then A fulfills a need for B. If the design of A changes, then B's need may go unfulfilled. B's own behavior may then need to change to accommodate the change in A. Thus change propagates in the opposite direction to use. Importantly, Parnas stresses that use is not symmetric. If B uses A, but A does not use B, then B's behavior can change without affecting A. (We ignore the potential for indirect paths between A and B in this example.) As noted earlier, a DSM reveals this asymmetry – the marks in the rows denote one direction of the use relationship and the marks in the columns denote the other. If usage is symmetric (i.e., B uses A *and* A uses B), the marks will be symmetric around the main diagonal of the DSM.

Whether use proceeds from row to column or column to row is a matter of choice. There is no standard approach among DSM scholars. However, just as cars should drive on the left or the right to avoid collision, firms should adopt one or the other convention to avoid confusion. In our methodology, *we define use as proceeding from row to column*. That is, our DSMs show how the components in a given row use (i.e., depend upon) the components in a given column. More generally, for the *i*th component in a system one looks at the *i*th element along the main diagonal. To identify the components that it depends upon, one looks along its row. To identify the components that depend upon it, one looks up and down its column.

In a layered architecture, a second convention determines the ordering of layers from top to bottom. One can place the "users" in higher layers and the objects of use in lower layers or vice versa. Most EA layer diagrams display the users at the top. In constructing DSMs, however, we depart from this practice, and place users below the objects that they use. Our reasons for doing this are based upon the concept of "design sequence" as described below.

When used as a planning tool in a design process, a DSM indicates a possible sequence of design tasks, i.e., which components should be designed before which others. In general, it is

intuitive and desirable to place the first design tasks at the top of a DSM, with later tasks below. In sum, the first components to be designed should be those that other components depend on. For example, suppose that B uses A. A's design should be complete before B's design is begun. Reversing this ordering runs the risk that B will have to be redesigned to comply with changes in A. Reflecting this sequence in a DSM, we place the "most used" layers on the top and the "users" of these layers towards the bottom. This convention ensures that design rules and requirements, which affect subsequent design choices, always appear at the top of the DSM.

The next question to answer is how should dependencies between elements be counted? Should the matrix cells contain only binary information, indicating a linkage, or ordinal values? Consider when the components of a system are complex entities (e.g., like applications, schemas and servers), there can be multiple ways that each component uses or depends upon the others. For example, Application B may make different types of requests of Application A. It is possible to count those different requests and assume a linkage is "stronger" when the number of requests (or request types) is higher. Similarly, following Sharman and Yassine (2007, 2004), one can interpret the off-diagonal entries in a DSM as indicating the probability that a change to one component will cause a change to another. In this scenario, a value of "1" would indicate the certainty of change, while lesser values would indicate merely the possibility of change.

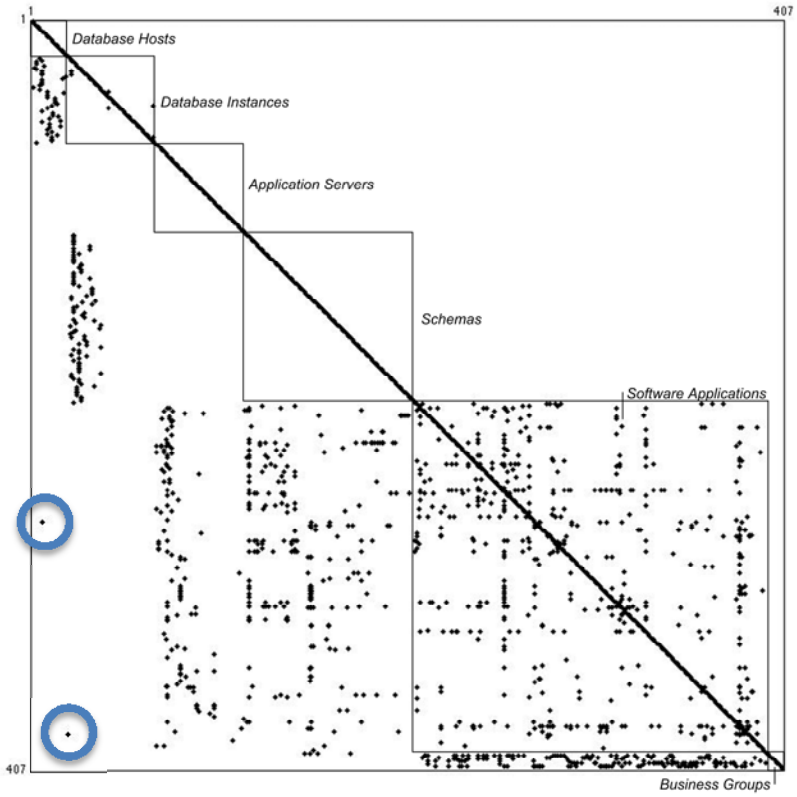
While these are plausible arguments, they are difficult to apply in practice. Establishing the strength of a linkage, or the probability that a change in one component requires a change in the other, requires a deep level of knowledge, which rarely exists in an enterprise setting. Further, allocating different strengths or weights to dependencies can give a false sense of precision in a DSM analysis. The existence of a dependency between two elements, no matter how many ways this dependency is expressed, or how frequently it is observed in operation,

merely signifies the *potential* for changes to propagate between these elements. As a consequence, we use a binary DSM as the baseline for analyzing enterprise architecture.[4]

A layered DSM showing the BioPharma enterprise architecture is presented in Figure 1. The matrix is binary with marks in the off-diagonal cells indicating a direct dependency from row to column (and hence a change vulnerability from column to row). White space indicates there is no direct dependency between elements. To set the order of layers we use knowledge of the logical relationships between components. Usage flows from business groups (at the bottom) to applications, from applications to schemas and application servers, from schemas to database instances and from database instances to database hosts (at the top). Within layers, we order components using the component ID, an arbitrary numbering scheme. Note that "communicates with," the dependency captured between software applications in our data, is bi-directional, hence the marks in the rows and columns of this layer are symmetric around the main diagonal.

[4] We note further research might examine how the strength of linkages or change probabilities could be used to build Enterprise Architecture DSMs that possess greater predictive power than simple binary versions.

Figure 1 A DSM displaying BioPharma's layered Enterprise Architecture



Summing the row entries for a given component in the DSM measures the direct outgoing coupling of that component—the number of other components that it uses. We call this measure the direct "fan-out" dependency of the component. Summing the column entries measures the direct incoming coupling of that component – the number of other components that use it. We call this measure the direct "fan-in" dependency of the component. White space to the right of a given layer indicates that components in the layer do not depend on layers below. White space to the left indicates that components in the layer do not depend on layers above.

On the whole, the DSM confirms that this enterprise architecture displays a good separation of concerns: for the most part, schemas act as an interface between applications and the database instances and hosts. Schemas are also efficiently managed: one schema may serve

several applications and one application may make use of several schemas. There are two exceptions, however, as indicated by the two circles, where specific applications appear to directly use a database instance or a database host. These exceptions may indicate poor encapsulation or non-standard practices, and hence would be worth investigating further.

It is important to note that this DSM combines several diagrams and matrices that are part of TOGAF's approach to enterprise architecture (TOGAF, 2009). The mapping from business groups to applications (at the bottom of the DSM) corresponds to the "Application/Organization Matrix." The square submatrix of applications corresponds to the "Application Interaction Matrix" (AIM). The mapping from applications to schemas and servers (to the left of the AIM) corresponds to the "Application Technology Matrix." Finally, the mapping from schemas to database instances and database instances to database hosts contains the information needed to construct the "Application/Data Matrix," while also showing how the use of data by applications operates through particular schemas and database instances. For these reasons, we believe that our methodology constitutes an important step towards making this framework more operational.

We note that the Application Interaction Matrix (AIM) is the largest submatrix in this DSM. It shows the dependencies caused by interactions between the software applications in the enterprise's portfolio. In this dataset, dependencies between software applications are captured by the term "communicates with," which does not possess directionality (i.e., we do not know which application is requesting a computation and which is performing it). Hence the AIM is symmetric. In general however, capturing information about directionality is always desirable. In particular, one application may always ask for a computation, and another may always supply the result. This distinction would be obscured if all dependencies were merely assumed to be

symmetric. However, if applications switch roles, sometimes requesting and sometimes supplying computational services, a symmetric dependency would in fact be warranted.

4. Analyzing an Enterprise Architecture DSM

Figure 1 displays the layered structure of the enterprise architecture, but does not reveal other important architectural characteristics such as indirect coupling, cyclic coupling, hierarchy, or the presence of "core" and "peripheral" components. Matrix operations can be applied to a DSM to analyze these additional features. Specifically, the transitive closure of the matrix reveals indirect dependencies among components in addition to the direct dependencies (Sharman et al., 2002; Sharman and Yassine, 2004; MacCormack et al., 2006). That is, if C depends on B and B depends on A, transitive closure reveals that C depends on A.

Applying the procedure of transitive closure to a DSM results in what is called the "Visibility" matrix (MacCormack et al., 2006; Baldwin et al., 2014). The visibility matrix captures all of the direct and indirect dependencies between elements. In a similar fashion to the DSM, row sums of the Visibility matrix, called "visibility fan-out" (VFO) measure the direct and indirect outgoing dependencies for a component. Column sums, called "visibility fan-in" (VFI) measure the direct and indirect incoming dependencies for a component. In a layered enterprise architecture, like the one observed in BioPharma, components at the top of the DSM will have high VFI and components at the bottom of the DSM will have high VFO. Critically, in cases where the systems layers are not known *a priori*, the Visibility matrix can be sorted using VFI and VFO to reveal the hierarchical relationships among components/layers.[5]

[5] We note that while matrix methods can reveal the hierarchical relationships among components, they will not be able to tease apart discrete groups of components that have equivalent positions in the hierarchy.

VFI and VFO can be used to identify "cyclic groups" of components, each of which is directly or indirectly connected to all others in the group. Mathematically, members of the same cyclic group all have the same VFI and VFO measures, given they are all connected directly or indirectly to each other. Thus we can identify cyclic groups in a system by sorting on these measures after performing a transitive closure on the DSM (Baldwin et al., 2014). Large cyclic groups are problematic for system designers, given changes to a component may propagate via a chain of dependencies to many other components. In such a structure, the presence of cyclicality means that there is no guarantee that the design process (or a design change) will converge on a globally acceptable solution that satisfies all components (Alexander, 1964; Steward, 1981).

The density of the Visibility matrix, called Propagation Cost, provides a measure of the level of coupling for the system as a whole. Intuitively, the greater the density of the Visibility matrix, the more ways there are for changes to propagate, and thus the higher the cost of change. Large differences in propagation cost are observed across systems of similar size and function (MacCormack et al., 2012). Yet empirical evidence also suggests that refactoring efforts aimed at making a design more modular can lower propagation cost substantially (MacCormack et al., 2006; Akaikine, 2009). These findings suggest that at least for software, architecture is not dictated solely by system function, but varies widely, at the discretion of a system's architects.

Prior work has shown that the components in a system can be classified into different groups according to the levels of coupling they exhibit, as captured by VFI and VFO. Specifically, Baldwin et al. (2014) use DSMs to analyze the structure of 1286 releases from 17 distinct software applications. They find the majority of systems exhibit a "core-periphery" structure, characterized by a single dominant cyclic group of components (the "Core") that is large relative to the system as a whole as well as to other cyclic groups. They show that the

components in such systems can be divided into four groups – Core, Peripheral, Shared and Control – that share similar properties in terms of coupling. In such systems, dependencies (i.e., "usage") flow from Control components, through Core components, to Shared components. This represents the main "flow of control" in the system. Peripheral components, by contrast, lie outside the main flow of control, given they are weakly connected to other system components.

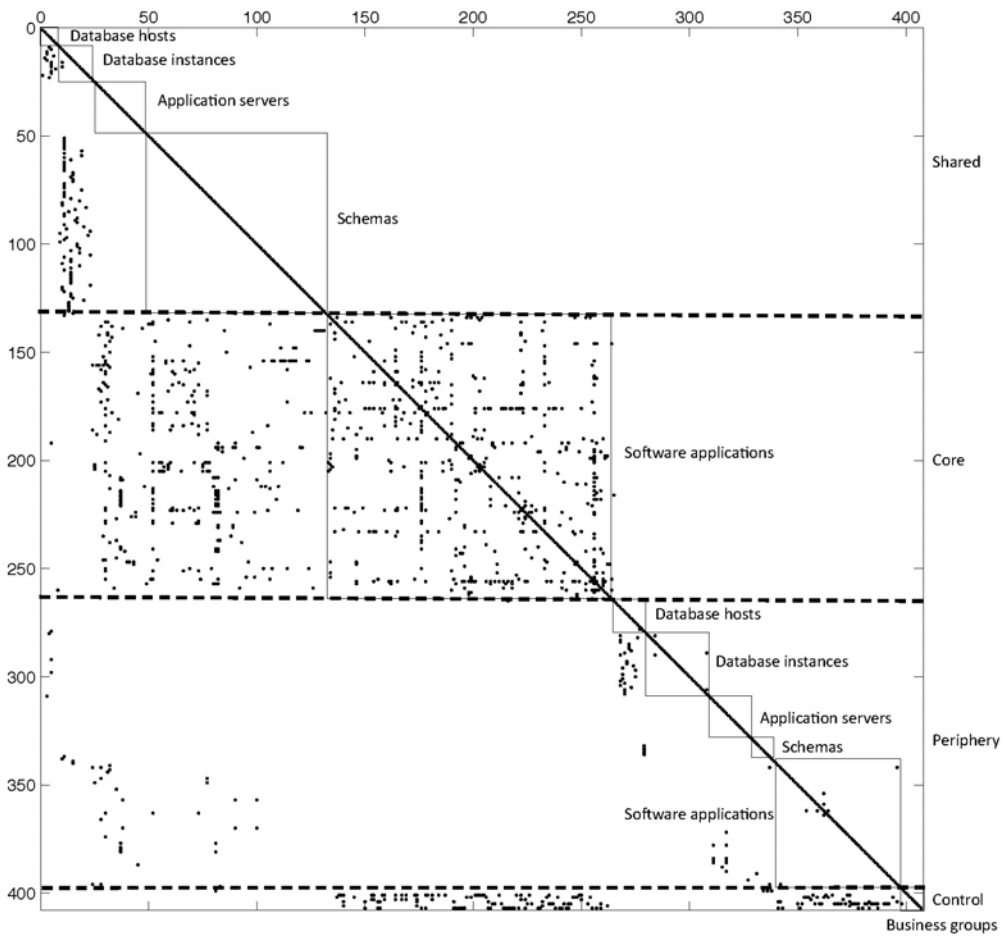
We constructed the Visibility Matrix for BioPharma, and applied the classification methodology described in Baldwin et al. (2014) to the resulting data for VFI and VFO (see Table 1). We find that the firm's enterprise architecture has a core-periphery structure, with 132 "Core" components (i.e., components that are mutually interdependent). Furthermore, all of the Core components in the system are software applications (but note, not all software applications are classified as Core). Each of the layers in the enterprise architecture identified in Figure 1 has some components that are part of the main flow of control, and others in the "Periphery." In total, 2/3 of the components in the architecture are part of the main flow and 1/3 are peripheral. We believe managers will find this type of classification scheme useful to set priorities, allocate resources, analyze costs, and understand potential differences in resource productivity.

Table 1: Distribution of Components in the Architecture by Layer and Category.

	Shared	**Core**	**Control**	**Periphery**
Database hosts	8	0	0	12
Database instances	15	0	0	32
Application servers	27	0	0	22
Schemas	83	0	0	9
Software application	0	132	0	59
Business groups	0	0	7	1
TOTAL	**272**			**135**
Percent of Total	**66%**			**34%**

Figure 2 shows a reorganized view of BioPharma's DSM, organized first, by type of component (i.e., Shared, Core, Periphery, Control) and second, by enterprise architecture layer. We call this the "core-periphery" view of the enterprise architecture DSM. Components in Shared, Core or Control categories are directly or indirectly connected to all Core components (and potentially, other components not in the Core) and hence represent the main flow of control. Thus each main-flow component is connected to at least 132 other components (though the direction of these dependencies will vary by category). In contrast, the highest level of coupling (i.e., visibility fan-in or visibility fan-out) for any peripheral component is only 7. Hence the indirect coupling levels of components in the main flow and the periphery are dramatically different. Assuming the level of component coupling is related to the cost of change, as Parnas (1972) suggests, main-flow components will cost significantly more to change than components in the periphery. We investigate this argument empirically in the following section.

Figure 2: Reorganized DSM showing Main Flow and Peripheral Components.



5. Using the Enterprise Architecture DSM to Predict Performance

In this section, we demonstrate the application of our methodology to the analysis of system performance. In particular, we examine the relationship between measures of component coupling derived from a DSM and the costs of component change. We focus on the cost of change, given that this is a direct measure of flexibility, the most common theme that emerges in the literature on enterprise architecture. Our analysis uses a subset of the data from BioPharma, for which information on the cost of change is available. Specifically, we predict the cost of change for *software applications* in the enterprise architecture. We begin by developing several hypotheses about the relationship between component coupling and the cost of change.

5.1 The Relationship between Coupling and the Cost of Change

In the previous section, we show that BioPharma's enterprise architecture is comprised of components with very different levels of coupling. In complex systems, heterogeneous levels of component coupling are the rule, not the exception (e.g., Lagerström, et al. 2014; Baldwin et al., 2014; Akaikine, 2010; Sturtevant, 2013). However, little empirical evidence exists about how different measures of coupling relate to the costs of change for a system's components. These costs determine the flexibility of a firm to evolve and adapt its IT systems.

Design theory predicts that the more coupled a component is, the more difficult and expensive it will be to change (Simon, 1962). However, the components of a system can be connected in different ways. Specifically, they can be connected directly or indirectly; and they can be connected hierarchically or cyclically. Furthermore, components that are hierarchically connected may be at the top or the bottom of the hierarchy, whereas components that are

cyclically connected may be members of a large or a small cyclic group. Measures of these (and other) types of coupling can be derived from an enterprise architecture DSM.

In this study, we examine the performance impact of three related coupling measures:

(1) The level of *Direct Coupling* for each component, which is calculated by summing the entries in the rows and columns of the enterprise architecture DSM.

(2) The level of *Indirect Coupling* for each component, captured here by its classification as being either a Core or Peripheral component (Baldwin et al, 2014).

(3) The *Closeness Centrality* for a component, a metric from social network theory, which can be calculated for Core components (i.e., those in the same network).[6]

In our dataset, data on the cost of change was available only for software applications, whose dependency relationships are defined to be symmetric. Hence it was not possible to explore the impact of differences between the number of "incoming" and "outgoing" dependencies, nor differences in the hierarchical classification of components (a symmetric DSM contains Core and Peripheral elements, but no Shared or Control elements). In general however, our method allows the exploration of these issues, in cases where dependencies are asymmetric.

Different measures of coupling are likely to be correlated. Specifically, components with high levels of direct coupling are more likely to be members of the Core. Furthermore, closeness centrality is only defined for components in the Core (i.e., those in the same network).[7] Finally, Core components with high levels of direct coupling are more likely to have higher closeness

[6] Closeness centrality captures how "close" a component is to other components in a network. But it can only be calculated for symmetric networks. If A depends upon B, but B does not depend upon A, then the path length from A to B, and from B to A will differ. Centrality cannot capture these subtleties. It assumes dependencies are symmetric, which is not the norm in technical systems, but is true for software applications at BioPharma.

[7] In prior work, the closeness centrality for elements that have no connections to others is sometimes assumed to be zero (i.e., denoting an infinite path between these and other elements).

centrality. These relationships mean that we must be sensitive to issues of multi-collinearity. To address this issue, we conduct our analysis in two stages. First, we explore the differential impact of Direct and Indirect coupling in predicting the cost of change for software applications. Then, for the subset of Core components in the system, we test whether the measure of closeness centrality provides additional explanatory power.

Stage 1: Direct versus Indirect coupling. Following Chidamber and Kemerer (1994), we define direct coupling (DC) as the number of direct dependencies between a software application and all others. Note that because software dependencies are defined as symmetric, the number of incoming and outgoing dependencies is identical. The level of indirect coupling is captured by whether a software application forms part of the largest cyclic group (i.e., the Core) in the system. All members of the Core have the same number of direct and indirect dependencies. Core membership is revealed through transitive closure of the DSM.[8]

Design theory predicts that higher levels of direct coupling will be associated with higher costs to change. The theory of change propagation predicts that higher levels of indirect coupling (as measured by membership of the Core) will also lead to higher costs to change. These effects might be additive, or they might be substitutes. We thus state the following hypotheses:

H1: Direct Coupling (DC) is positively associated with change cost (CC).

H2: Core membership (CORE) is positively associated with change cost (CC).

H3: Direct Coupling (DC) and Core membership (CORE), considered together, explain more of the variation in change cost (CC) than either measure considered alone.

We test these hypotheses by performing OLS regressions for the impact of Direct Coupling and Core membership on the cost of change, both individually and together.

[8] We note there was only one cyclic group in this dataset, thus components not in the Core were not part of *any* cyclic group. In general however, there might be other, smaller, cyclic groups in the enterprise architecture.

Stage 2: Coupling within the Core. For Core components, closeness centrality (CENT) is found by calculating the minimum path length from that component to all other components, summing those path lengths and taking the inverse of this sum (Bounova and de Weck, 2012). The higher this number, the more "central" is the component. Our final hypothesis explores the possibility that that closeness centrality explains variations in change cost for all components that are part of the same cyclic group (i.e., they possess the same high level of indirect coupling):

> *H4: For components that are members of the Core, closeness centrality (CENT) is positively associated with change cost (CC).*

We test this hypothesis by performing an OLS regression for the impact of closeness centrality on the cost of change *only* for Core components.

5.2 Dependent Variable: The Cost of Change for a Component

To demonstrate our methodology, we use data on the cost of change for software applications. Focusing on a single layer of the firm's enterprise architecture (i.e., as opposed to all layers) allowed us to i) identify a specific respondent for data collection, ii) request quantitative data from these respondents, and iii) ensure the data was comparable across units.

The cost to change each application was assessed via a survey sent to IT Service Owners. Respondents were asked to estimate the time, in person-years, to perform five operations: deploy, upgrade, replace, decommission, and integrate. Operations were defined as follows: "A component is *deployed* when it is put into production for the first time; a component is *upgraded* when it is replaced by a new version of the same component; a component is *replaced* when the existing component is removed from the information system and a new component with similar functionality is added to the information system; a component is *decommissioned* when it is

removed from the information system; and a component is *integrated* when modifications are made to it that enable it to 'talk' to another component" (Dreyfus, 2009).[9]

We received survey responses for 99 software applications. The change cost estimates ranged from less than one-person-month to over two-person-years. Respondents could also indicate that the time to perform a given operation was unknown. Applications for which all change costs were unknown were removed from the dataset, resulting in a final sample of 77 applications.[10] For these applications, we combined the change cost estimates for different operations into a single measure, by calculating the mean change cost for operations where a response was provided. The Cronbach's alpha for this aggregate measure was 0.78.[11]

5.3 Control Variables

Change costs may be affected by a number of factors that are unrelated to architecture, including the source of the component, the users of the component, its internal structure, and whether it was the focus of active development at the time of the survey. In addition, the respondent's experience with a given component might affect the appraisal of change cost in a systematic way. Hence data on the following variables were collected and included as controls:

(1) *VENDOR* indicates whether an application is developed by a vendor (1) or in-house (0). One component missing data for this variable was assigned a value of 0.5.[12]

(2) *CLIENT* indicates whether an application is accessed by end-users (1) or not (0).

[9] Specifically, we asked respondents to estimate whether the effort (in person-years) required for each operation fell into the following ranges: <0.10, 0.10-0.249, 0.25-0.49, 0.50-0.99, 1.00-1.99, and > 2.00. The resulting dependent variable was an integer ranging from 1 to 6. For details, see Dreyfus, 2009.

[10] In prior work, Dreyfus (2009) and Dreyfus and Wyner (2011) use different screening criteria, resulting in a sample of 62 responses for analysis. We discuss the sensitivity of results to different screening criteria later.

[11] Where the estimate of change cost for an operation is missing, we substitute the mean level of change cost for that operation from all respondents to calculate Cronbach's alpha. Other ways of treating missing values result in a minimum value for alpha of 0.66 (acceptable) to a maximum value of 0.89 (extremely good).

[12] Omitting the one application with no data provided about vendor did not change the results.

(3) *COMP* indicates whether an application is focused on computation (1) or not (0).

(4) *NTIER* indicates whether an application has an N-tier architecture (1) or some other type of architecture, such as client-server or monolithic (0).

(5) *ACTIVE* indicates whether, at the time of the survey, the component was being actively enhanced (1) or was in maintenance mode (0).

(6) *RES_EXP* measures the respondent's experience with the application in question (less than one year = 1; 1 – 5 years = 2; More than 5 years =3).

5.4 Empirical Data

Table 2 presents the correlation matrix for our variables. Consistent with our hypotheses, both direct coupling (DC) and CORE are positively correlated with change cost. They are also correlated with each other (0.52). In this table, we include data on closeness centrality (CENT) for the entire sample of 77 applications, substituting a value of 0 for the 19 components not in the Core. Hence we observe an extremely high correlation (0.96) between CORE and CENT.

Among the control variables, Active components tend to have higher change costs. Vendor provided components tend to have lower change costs, have lower centrality, are more likely to perform computations, are less likely to have N-tier architectures, and are more likely to be Active. Components with N-tier architectures tend to be more highly coupled by all measures.

Table 2 Descriptive Statistics and Correlation Matrix.

	Mean	*St.Dev*	*#*	CC	DC	CORE	CENT	VENDOR	CLIENT	COMP	NTIER	ACTIVE	RES_EXP
CC	2.46	1.22	77	1	0.33**	0.33*	0.35**	-0.23*	-0.06	-0.11	0.17	0.31**	0.08
DC	3.58	2.83	77	0.33**	1	0.52***	0.68***	-0.19	0.1	0.01	0.39***	0.21	-0.08
CORE	0.75	0.43	77	0.33**	0.52***	1	0.96***	-0.17	0.11	-0.04	0.37***	0.06	-0.19
CENT	1.73	1.03	77	0.35**	0.68***	0.96***	1	-0.26*	0.14	-0.04	0.46***	0.12	-0.15
VENDOR	0.41	0.49	77	-0.23*	-0.19	-0.17	-0.26*	1	-0.06	0.39***	-0.52***	0.35**	0.1
CLIENT	0.71	0.45	77	-0.06	0.1	0.11	0.14	-0.06	1	0.15	0.27*	0.05	-0.11
COMP	0.39	0.49	77	-0.11	0.01	-0.04	-0.04	0.39***	0.15	1	-0.21	-0.05	0.12
NTIER	0.53	0.5	77	0.17	0.39***	0.37***	0.46***	-0.52***	0.27*	-0.21	1	0.08	-0.14

ACTIVE	0.26	0.44	77	0.31**	0.21	0.06	0.12	0.35**	0.05	-0.05	0.08	1	0.18
RES_EXP	2.04	0.84	77	0.08	-0.08	-0.19	-0.15	0.1	-0.11	0.12	-0.14	0.18	1
* p<0.05, ** p<0.01, and ***p<0.001													

5.5 Empirical Results

The results of our regression tests are presented in Table 3. Model 1 contains only controls, showing that two of them are significant: Vendor provided applications tend to have lower change costs and Active applications tend to have higher change costs. The control variables alone explain 18% of the variation in change cost across applications.

Our second hypothesis, H2 predicts that direct coupling is associated with change cost, a relationship suggested by the correlations reported above. In Model 2 however, which includes control variables, we find direct coupling is only a relatively weak predictor of change cost (p-value = .06). This model explains 21% of the variation in change cost across applications. In Model 3, we find CORE is a highly significant predictor of change cost (p-value = .005). This model explains 26% of the variation in change cost across components.

Table 3 Regression Models

Dependent variable: Change Cost (Average)						
Sample:		Full			Core only	
Test #	(1)	(2)	(3)	(4)	(5)	(6)
Hypothesis		H1	H2	H3,H4		H5
DC		0.10†		0.04		
CORE			0.89**	0.77*		
CENT						-0.76
VENDOR	-1.21**	-1.10**	-1.19**	-1.14**	-1.40**	-1.67***
CLIENT	-0.29	-0.26	-0.27	-0.26	-0.69†	-0.71*
COMP	0.28	0.17	0.22	0.18	0.22	0.34
NTIER	-0.17	-0.33	-0.43	-0.47	-0.44	-0.34
ACTIVE	1.37***	1.19**	1.30***	1.23***	1.69***	1.92***
RES_EXP	0.01	0.04	0.09	0.09	-0.01	0.01

Constant	2.77***	2.47***	2.11***	2.06***	3.45***	5.10***
Adj. Rsquare	0.18	0.21	0.26	0.25	0.25	0.27
f	3.75**	3.87**	4.75***	4.21***	4.17**	3.94**
Observations	77	77	77	77	58	58

† p<0.1, * p<0.05, ** p<0.01, and ***p<0.001

In Model 4, we include both direct coupling and CORE in the regression, but only CORE is significant. This model explains 25% of the variation in change cost across applications, a reduction from Model 3. In sum, *H1* and *H2* are supported by our results, but *H3* is rejected. Specifically, adding direct coupling to a model that already includes CORE makes the model *worse*. CORE is the strongest predictor; the power that direct coupling has as an explanatory variable in Model 1 is accounted for by its correlation with CORE.

In models 5 and 6, we analyze only the 59 components in the Core (the largest cyclic group of components). Model 5 contains only control variables, and produces results consistent with model 1. Model 6 includes the measure of closeness centrality, which is not significant. Hence closeness centrality provides no additional explanatory power in predicting change cost, over and above that provided by Core. We therefore reject hypothesis *H4*.

5.6 Robustness Checks

We performed a number of checks to assess whether our results were sensitive to other assumptions or specifications of variables. First, we note that our basic specification did not control for the size of components, a variable that could plausibly affect the cost of changes. Data on component size (measured by the number of lines of code and files in each) was available for a subsample of 60 applications (Dreyfus, 2009). We ran our models on this smaller

sample, including these as controls. The controls were insignificant, while the results for our explanatory variables were consistent with those reported above.[13]

We conducted a test to explore the possibility that transformations of direct coupling might better predict change cost, given this variable has a skewed distribution and is truncated at zero. Specifically, we included the natural log of direct coupling in models, instead of the raw value. We found the transformed variable had more explanatory power than the raw variable (i.e., its use improved the results in Model 2). However, it still explained less of the variation in change cost than CORE, hence was insignificant when included in a model with CORE.

Finally, we explored whether direct coupling, or its natural log, contribute to explaining the variation in change cost among only Core components (as we did for centrality). Appendix A reports the results of three models predicting change cost, the first being a model with controls, the second adding direct coupling, and the third adding the natural log of direct coupling. Direct coupling is not statistically significant in any model. This suggests that in this dataset, CORE is the most parsimonious and powerful measure of coupling that explains the cost of change. Neither direct coupling, nor centrality, contributes additional explanatory power in our models.

6. Discussion and Conclusions

The main contribution of this paper is in developing a robust and repeatable network-based methodology by which to operationalize a firm's enterprise architecture. The methodology is consistent with prior work in this area, and addresses several limitations in this work. Specifically, it i) integrates the consideration of business and IT related attributes; ii) identifies the distinct layers in the architecture associated with different types of entity (e.g., applications

[13] Note, some of the significance levels declined as a result of the decrease in sample size and hence power.

versus databases); iii) reveals the main "flow of control" within the architecture and its associated layers and; iv) generates measures of the architecture that can be used to predict system performance. We demonstrate the application of this methodology using a novel real-world dataset, and show that it generates insights that could not have been gained from the inspection of documents or processes traditionally associated with the EA literature.

A second contribution of this paper lies in the specific results we report using our methodology to analyze enterprise architecture. Specifically, we explore the relationship between different measures of coupling derived from a DSM, and the cost of change for applications. We find the measure of coupling that best predicts change cost is not the number of direct dependencies for a component, but all of its direct *and indirect* dependencies with others. Once the variations in change cost explained by this measure are accounted for, other measures of coupling add no further explanatory power. This suggests a firm's flexibility to adapt its IT infrastructure is driven mainly by the potential for changes to propagate from one component to others via chains of dependencies. This data is not apparent merely by inspection of a component's "nearest neighbors." Rather, our findings lend support to the methods we employ, which focus on understanding all of the indirect paths that exist between components.

For managers, our methodology provides a clear picture of the actual *instantiated* architecture that they must manage, as opposed to the planned or idealized versions often found in documents depicting a firm's enterprise architecture. The insights generated should prove useful in several ways, including i) helping to plan the allocation of resources to different components, based upon information on the relative ease/difficulty of change; ii) monitoring the evolution of the architecture over time, as new components and/or dependencies are introduced

(e.g., when a new firm is acquired) and; iii) identifying opportunities to improve the architecture, for example, by reducing coupling, and hence reducing the cost of change for components.

Ironically, in this era of "big data," the lack of appropriately granular data is the largest barrier to the systematic investigation of enterprise architecture using our methodology. Specifically, at a minimum, firms need to capture data on the dependencies between different components in the enterprise architecture, and the way that these dependencies evolve over time. To put this data to use, they must systematically capture performance data about the cost of change, defect rates, and productivity *by component*. In most organizations we know, this type of data does not exist. In some, efforts have been made to collect data manually. However, there are many challenges associated with this approach, including the lack of incentive to provide accurate and timely information. In our study, we found substantial omissions in the data collected via survey, in comparison to the automated tools used to uncover system dependencies. In essence, many firms do not actually know the "real" enterprise architecture that they possess.

Eppinger and Browning (2012) state: "for most product DSM models, the data collection requires at least some amount of direct discussion with subject matter experts in order to draw out the tacit and system-level knowledge that may not be captured in the documentation." However, manual methods of dependency extraction are labor-intensive, and limit the scale, precision and accuracy of analyses. The ideal solution is to develop more automated ways to detect and capture important dependencies between components in a firm's enterprise architecture.[14] This implies the need for some level of investment by firms who wish to adopt

[14] E.g., in software, automatic dependency extractors supplied by commercial vendors can be used to create DSMs with no manual effort needed (e.g., Cataldo et al., 2006; MacCormack et al., 2006, 2012; Sosa et al., 2013.)

these methods. However, we believe the benefits associated with these investments would more than offset the costs, given the increase in understanding of architecture that would result.

For the academy, this study contributes to the field of enterprise architecture in several ways. First, it makes what has previously been a rather conceptual area more concrete, providing a method to analyze a firm's actual architecture in-use, rather than examining the processes and documents through which it is managed. Second, it provides a way to operationalize frameworks like TOGAF (2009), by defining how the matrices they include can be quantified and analyzed. Finally, our methodology outputs metrics that capture the level of coupling between different components in a firm's architecture, thereby revealing the main "flow of control" in a system.

Our work opens up the potential for further empirical research that could explore the relationship between enterprise architecture and performance in a deeper way. Within organizations, work might focus on the relationship between measures of coupling, and a variety of performance measures relevant to individual components in the architecture (e.g., reliability, defects, productivity, turnover and/or cost). In contrast, studies across organizations might be directed at revealing how measures of enterprise architecture affect *firm-level* performance. The latter area is particularly promising, given prior literature argues there is a strong linkage between certain types of architecture and flexibility or agility. One might ask, for example, whether loosely coupled enterprise architectures, in general, facilitate a more rapid response to business challenges? Or are there subtle nuances to account for, with respect to different layers in the architecture (e.g., is the use of shared databases a best practice)? This methodology allows us to answer such questions, with an approach that can be replicated across studies.

Our analysis is subject to a number of limitations that must be considered when assessing generalizability. In particular, the data to demonstrate our methods came from a single firm.

Hence more work is needed to provide validation of these methods across different contexts. Furthermore, questions remain as to the different layers/components that should be included in the analysis of enterprise architecture, and the types of dependency that exist between them. For example, we may find that different types of dependency (e.g., “uses” versus “communicates with”) predict different dimensions of performance (e.g., cost to change versus defects). Similarly, we may find that different measures of coupling (e.g., direct versus indirect coupling) may predict performance differently in different contexts. Ultimately, our methods provide a platform to enable others to answer questions that until now have gone unanswered. As such, we hope that future researchers will improve and evolve these methods, in order that we benefit from the cumulative nature of academic knowledge.

References

Adomavicius, G., Bockstedt, J. C., Gupta, A., and Kauffman, R. J. 2008. Making sense of technology trends in the information technology landscape: A design science approach. *MIS Quarterly* 32, 4, 779-809.

Aier, S. 2014. The role of organizational culture for grounding, management, guidance and effectiveness of enterprise architecture principles. *Information Systems and E-Business Management 12*, 1, 43-70.

Aier, S., Gleichauf, B., and Winter, R. 2011. Understanding Enterprise Architecture Management Design: An Empirical Analysis. In *Proc. of Wirtschaftsinformatik*, Association for Information Systems.

Aier, S. and Winter, R. 2009. Virtual decoupling for IT/business alignment–conceptual foundations, architecture design and implementation example. *Business & Information Systems Engineering 1*, 2, 150-163.

Akaikine, A. 2010. The Impact of Software Design Structure on Product Maintenance Costs and Measurement of Economic Benefits of Product Redesign. System Design and Management Program Thesis, Massachusetts Institute of Technology.

Alexander, C. 1964. *Notes on the Synthesis of Form*. Harvard University Press.

Allen, E. B., Gottipati, S., and Govindarajan, R. 2007. Measuring size, complexity, and coupling of hypergraph abstractions of software: An information-theory approach. *Software Quality Journal 15*, 2, 179-212.

Baldwin, C. and Clark, K. 2000. *Design Rules, Volume 1: The Power of Modularity*. MIT Press.

Baldwin, C., MacCormack, A., and Rusnack, J. 2014. Hidden structure: Using network methods to map system architecture. *Research Policy*, Article in Press. Accepted May 19 2014.

Barabási, A. 2009. Scale-free networks: A decade and beyond. *Science* 325, 5939, 412-413.

Boh, W. F., and Yellin, D. 2007. Using enterprise architecture standards in managing information technology. *Journal of Management Information Systems* 23, 3, 163-207.

Bounova, G., de Weck, O.L. 2012. Overview of metrics and their correlation patterns for multiple-metric topology analysis on heterogeneous graph ensembles, *Phys. Rev. E* 85.

Briand, L. C., Daly, J. W., and Wust, J. K. 1999. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering 25*, 1, 91-121.

Brown, N., Cai, T., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., and Zazworka N. 2010. Managing technical debt in software-reliant systems. In *Proc. of the FSE/SDP Workshop on the Future of Software Engineering Research (FoSeR'10)*, 47-52.

Buckl, S., Ernst, A. M., Matthes, F., Ramacher, R., and Schweda, C. M. 2009. Using enterprise architecture management patterns to complement TOGAF. In *Proc. of the Enterprise Distributed Object Computing Conference (EDOC'09). IEEE International* 34-41. IEEE.

Cataldo, M., Wagstrom, P., Herbsleb, J., Carley, K. 2006. Identification of coordination requirements: implications for the Design of collaboration and awareness tools. In: *Proc. of the 2006 Conference on Computer Supported Cooperative Work*. pp. 353–362.

Chidamber, S. R., and Kemerer, C. F. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20, 6, 476-493.

Clark, K.B. 1985. The interaction of design hierarchies and market concepts in technological evolution. *Research Policy* 14, 5, 235–251.

Department of Defense Architecture Framework Working Group: *DoD Architecture Framework (DoDAF)*. 2007. Version 1.5, Technical report, Department of Defense, USA.

Dhama, H. 1995. Quantitative models of cohesion and coupling in software. *Journal of Systems and Software* 29, 1, 65-74.

Dietz, J. L., and Hoogervorst, J. A. 2011. A critical investigation of TOGAF: based on the enterprise engineering theory and practice. In *Advances in Enterprise Engineering V, Lecture Notes in Business Information Processing Volume 79, Proc. of the 1st Enterprise Engineering Working Conference (EEWC)*, 76-90, Springer Berlin Heidelberg.

Dreyfus, D. 2009. *Digital Cement: Information System Architecture, Complexity, and Flexibility*. PhD Thesis. Boston University Boston, MA, USA, ISBN: 978-1-109-15107-7.

Dreyfus D. and Wyner, G. 2011. Digital cement: Software portfolio architecture, complexity, and flexibility. In *Proc. of the Americas Conference on Information Systems (AMCIS)*, Association for Information Systems.

Duncan, N. 1995. Capturing Flexibility of Information Technology Infrastructure: A Study of Resource Characteristics and Their Measure. *Journal of Management Information Systems* 12, 2, 37-57.

Eppinger, S. D., Whitney, D.E., Smith, R.P., and Gebala, D. A. 1994. A model-based method for organizing tasks in product development. *Research in Engineering Design* 6, 1, 1-13.

Fenton, N. and Melton, A. 1990. Deriving Structurally Based Software Measures. *Journal of Systems and Software* 12, 3, 177–187.

Franke, U., Johnson, P., and König, J. 2014. An architecture framework for enterprise IT service availability analysis. *Software & Systems Modeling* 13, 4, 1417-1445.

Gao, L. S, and Iyer, B. 2006. Analyzing Complementarities Using Software Stacks for Software Industry Acquisitions. *Journal of Management Information Systems* 23, 2, 119-147.

Hall, N. R., and Preiser, S. 1984. Combined network complexity measures. *IBM journal of research and development* 28, 1, 15-27.

Henry, S., & Kafura, D. 1981. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering* 5, 510-518.

Hinsman C., Snagal, N., and Stafford, J. 2009. Achieving agility through architectural visibility. *Architectures for Adaptive Software Systems*, LNCS 5581, 116-129.

Holland, J. H. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*, seconded. MIT Press, Cambridge, MA.

ISO/IEC/IEEE 2011. *Systems and software engineering - Architecture description.* Technical standard.

Jenkins, S. and Kirk, S. 2007. Software architecture graphs as complex networks: A novel partitioning scheme to measure stability and evolution. *Information Sciences* 177, 12, 2587-2601.

Johnson, P. and Ekstedt, M. 2007. *Enterprise Architecture: Models and analyses for information systems decision making*. Studentlitteratur.

Jonkers, H., Lankhorst, M. M., ter Doest, H. W., Arbab, F., Bosma, H., and Wieringa, R. J. 2006. Enterprise architecture: Management tool and blueprint for the organisation. *Information Systems Frontiers* 8, 2, 63-66.

Kauffman, S.A. 1993. *The Origins of Order*. Oxford University Press, New York.

Kluge, C., Dietzsch, A., and Rosemann, M. 2006. How to realize corporate value from enterprise architecture. In *Proc. of the European Conference on Information Systems* (*ECIS)*, 1572-1581, Association for Information Systems.

Lagerström, R., Franke, U., Johnson, P., and Ullberg, J. 2009. A method for creating enterprise architecture metamodels: applied to systems modifiability analysis. *International Journal of Computer Science and Applications* 6, 5, 89-120.

Lagerström, R., Johnson, P., and Höök, D. 2010. Architecture Analysis of Enterprise Systems Modifiability: Models, Analysis, and Validation. *Journal of Systems and Software* 83, 8, 1387-1403.

Lagerström, R., Sommestad, T., Buschle, M., and Ekstedt, M. 2011. Enterprise architecture management's impact on information technology success. In *Proc. of the 44th Hawaii International Conference on System Sciences (HICSS-44)*, IEEE.

Lagerström, R., Baldwin, C., MacCormack, A., and Dreyfus, D. 2013. Visualizing and Measuring Enterprise Architecture: An Exploratory BioPharma Case. In *Proc. of the 6th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling (PoEM)*. Springer.

Lagerström, R., Baldwin, C., MacCormack, A., and Aier, S. 2014. Visualizing and Measuring Enterprise Application Architecture: An Exploratory Telecom Case. In *Proc. of the Hawaii International Conference on System Sciences (HICSS-47)*, IEEE.

LaMantia, M., Cai, Y., MacCormack, A., and Rusnak, J. 2008. Analyzing the evolution of large-scale software systems using design structure matrices and design rule theory: Two exploratory cases. In *Proc. of the 7th Working IEEE/IFIP Conference on Software Architectures (WICSA7)*.

Lankhorst, M. 2009. *Enterprise Architecture at Work: Modelling, Communication and Analysis*. The Enterprise Engineering Series. Springer.

MacCormack, A., Rusnak, J., and Baldwin, C. 2006. Exploring the structure of complex software designs: an empirical study of open source and proprietary code. *Management Science* 52, 7, 1015–1030.

MacCormack, A. 2010. The Architecture of Complex Systems: Do “Core-Periphery" Structures Dominate?. In *Proc. of Academy of Management*.

MacCormack, A., Baldwin, C., and Rusnak, J. 2012. Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis. *Research Policy* 41, 8, 1309-1324.

Mead, C. and Conway, L. 1980. *Introduction to VLSI Systems*. Addison-Wesley Publishing Co.

Ministry of Defence. 2008. *MOD Architecture Framework (MODAF)*. Version 1.2.003. Technical report, Ministry of Defence, UK.

Mocker, M. 2009. What is complex about 273 applications? Untangling application architecture complexity in a case of European investment banking. In Proc. of the *42nd Hawaii International Conference on System Sciences (HICSS)*, IEEE.

Myers, C. R. 2003. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. *Physical Review* E 68, 4, 046116.

Närman, P., Holm, H., Johnson, P., König, J., Chenine, M., and Ekstedt, M. 2011. Data accuracy assessment using enterprise architecture. *Enterprise Information Systems* 5, 1, 37-58.

Ozkaya, I. 2012. Developing an architecture-focused measurement framework for managing technical debt. In *Software Engineering Institute blog*. http://blog.sei.cmu.edu/post.cfm/developing-an-architecture-focused-measurement-framework-for-managing-technical-debt, accessed Sept. 2013.

Parnas, D. L. 1972. On the criteria to be used in decomposing systems into modules. *Communications of the ACM* 15, 12, 1053-1058.

Roeleven, S. 2010. Why Two Thirds of Enterprise Architecture Projects Fail: An explanation for the limited success of architecture projects. Whitepaper, Software AG.

Rohloff, M. 2008. Framework and Reference for Architecture Design. In *Proc. of the Americas' Conference on Information Systems (AMCIS)*. Paper 118.

Ross, J.W., Weill, P., and Robertson, D. 2006. *Enterprise Architecture As Strategy: Creating a Foundation for Business Execution*. Harvard Business School Press.

Sambamurthy, W. and Zmud, R. 2000. The Organizing Logic for an Enterprise's IT Activities in the Digital Era: A Prognosis of Practice and a Call for Research. *Information Systems Research* 11, 2, 105-114.

Sambamurthy, V., Bharadwaj, A., and Grover, V. 2003. Shaping Agility through Digital Options: Reconceptualizing the Role of Information Technology in Contemporary Firms. *MIS Quarterly* 27, 2, 237-263.

Schmidt, C. and Buxmann, P. 2011. Outcomes and success factors of enterprise IT architecture management: empirical insight from the international financial services industry. *European Journal of Information Systems* 20, 168–185.

Seppänen, V., Heikkila, J., and Liimatainen, K. 2009. Key issues in EA-implementation: case study of two Finnish government agencies. In *Proc. of the IEEE Conference on Commerce and Enterprise Computing (CEC'09)*, 114-120, IEEE.

Sharman, D., Yassine, A., and Carlile, P. 2002. Characterizing modular architectures. In *Proc. of the ASME 14th International Conference on Design Theory & Methodology*, DTM-34024, Montreal, Canada, September.

Sharman, D. and Yassine, A. 2004. Characterizing complex product architectures. *Systems Engineering Journal* 7, 1.

Sharman, D., Yassine, A., 2007. Architectural Valuation using the Design Structure Matrix and Real Options Theory. *Concurrent Engineering* 15, 157–173.

Simon, H. A. 1962. The architecture of complexity. *American Philosophical Society* 106, 6, 467-482.

Simon, D., Fischbach, K., and Schoder, D. 2013. An Exploration of Enterprise Architecture Research. *Communications of the Association for Information Systems* 32, 1, 1-72.

Sommestad, T., Ekstedt, M., and Holm, H. 2013. The Cyber Security Modeling Language: A Tool for Assessing the Vulnerability of Enterprise System Architectures. *IEEE Systems Journal*, Online-first.

Sosa, M. E., Mihm, J., and Browning, T. R. 2013. Linking Cyclicality and Product Quality. *Manufacturing & Service Operations Management 15*, 3, 473-491.

Sosa, M., Eppinger, S., and Rowles, C. 2007. A network approach to define modularity of components in complex products. *Transactions of the ASME* 129, 1118-1129.

Stevens, W. P., Myers, G. J., and Constantine, L. L. 1974. Structured design. *IBM Systems Journal* 13, 2, 115–139.

Steward, D. 1981. The design structure system: A method for managing the design of complex systems. *IEEE Transactions on Engineering Management* 3, 71-74.

Sturtevant, D. J. 2013. *System design and the cost of architectural complexity*. Diss. Massachusetts Institute of Technology.

Tamm, T., Seddon, P. B., Shanks, G., and Reynolds, P. 2011. How does enterprise architecture add value to organisations. *Communications of the Association for Information Systems* 28, 1, 141-168.

The Open Group. 2009. *The Open Group Architecture Framework (TOGAF)*. Version 9.

Tyler, D. F., and Cathcart, T. P. 2006. A structured method for developing agile enterprise architectures. In *Proc. of the International Conference on Agile Manufacturing (ICAM)*, Norfolk, Virginia, USA.

Ullberg, J., Johnson, P., and Buschle, M. 2012. A Language for Interoperability Modeling and Prediction. *Computers in Industry* 63, 8, 766-774.

Vakkuri, E. T. 2013. *Developing Enterprise Architecture with the Design Structure Matrix*. Master Thesis. Tampere University of Technology, Finland.

Weill, P. 2007. Innovating with Information Systems: What do the most agile firms in the world do. In *Proc. of the 6th e-Business Conference*, Barcelona.

Wilkie, F. G., and Kitchenham, B. A. 2000. Coupling measures and change ripples in C++ application software. *Journal of Systems and Software* 52,2, 157-164.

Winter, R. and Fischer, R. 2007. Essential Layers, Artifacts, and Dependencies of Enterprise Architecture. *Journal of Enterprise Architecture* 3, 2, 7-18.

Yoo, Y., Henfridsson, O., and Lyytinen, K. 2010. Research Commentary—The New Organizing Logic of Digital Innovation: An Agenda for Information Systems Research. *Information Systems Research* 21, 4, 724-735.

Zachman, J. A. 1987. A Framework for Information Systems Architecture. *IBM Systems Journal* 26, 3, 276-292.

Appendix A: Models Predicting Change Cost only for Core Components

MODEL	1	2	3
DC		0.03	
DC(ln)			0.09
VENDOR	-1.40**	-1.37**	-1.37**
CLIENT	-0.69†	-0.68†	-0.68†
COMP	0.22	0.20	0.21
NTIER	-0.44	-0.47	-0.46
ACTIVE	1.69***	1.64***	1.65***
RES_EXP	-0.01	0.00	-0.00
Constant	3.45***	3.35***	3.34***
Adj. Rsquare	0.25	0.24	0.24
f	4.17**	3.55	3.52
Observations	58	58	58
† p<0.1, * p<0.05, ** p<0.01, and ***p<0.001			