A Methodology for Operationalizing Enterprise Architecture and Evaluating Enterprise IT Flexibility

A Methodology for Operationalizing Enterprise Architecture and Evaluating Enterprise IT Flexibility

Alan MacCormack
Robert Lagerstrom
Carliss Y. Baldwin

Working Paper

15-060

January 22, 2015

A Methodology for Operationalizing Enterprise Architecture and Evaluating Enterprise IT Flexibility

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

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

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

A Methodology for Operationalizing Enterprise Architecture and Evaluating Enterprise IT Flexibility

Abstract

We propose a network-based methodology for operationalizing enterprise architecture. Our methodology is based upon using a "Design Structure Matrix" (DSM) to capture the coupling between different components in a firm's architecture, including business and technology-related aspects. We apply our methodology to data gathered in a large pharmaceutical firm. We show that this methodology helps to identify layers in the firm's architecture associated with different technologies (e.g., applications, servers and databases). We also show that it reveals the main "flow of control" within the architecture, as denoted by the classification of components into Core, Peripheral, Shared and Control elements. We analyze the cost of change for a subset of software applications within this architecture. We find that the cost of change is associated with the degree to which applications are highly coupled. We show the best measure of coupling that predicts the cost of change is one that captures all the *direct and indirect* connections between components. We believe our work constitutes an important step in making the concept of enterprise architecture more operational, improving a firm's ability to analyze its architecture, understand its performance implications, and adapt and improve it in the future.

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 accounting functions and provide data to support managerial decision-making. The role of these systems was then 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 and the World Wide Web 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 architectural 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.[1] It provides a way of visualizing, understanding, predicting, and planning for the needs of diverse stakeholders in a seamless and cost-effective way.

Unfortunately, reality often falls short of this ideal. Despite the increasing adoption of EA frameworks such as TOGAF by firms, making changes to a system, adding new system functionality, and integrating different systems (as in a merger) are not straightforward tasks. Changes made to one service or application can create unexpected disruptions in seemingly distant parts of the enterprise architecture (Vakkuri, 2013). In essence, when dealing with

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

complex information system architectures, *changes propagate in unexpected ways*, increasing the costs of adapting the system to future needs. This suggests the need for a more robust methodology to operationalize enterprise architecture in a way that generates a more granular understanding of system design and greater insight into how such systems can be improved.

Several EA frameworks propose using *matrices* to display the relationships among the 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 cycle.[2] These matrices are used to track linkages and dependencies between various parts of a large and complex system. However, the details of how to construct such matrices and, more importantly, how they can be used to improve decision-making are not well specified.

This paper seeks to address this gap, applying a network-based methodology developed to assist in the design of complex products to the design of enterprise architecture. Specifically, previous research 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 a complex system. 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 pharmaceutical company (Dreyfus, 2009). Using this data, we 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 elements of this architecture can be divided into a) "Core" components that tightly-interconnected, b) "Shared" and "Control" components, which have many incoming or outgoing dependencies and, (c) "Peripheral"

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

components that are only loosely-coupled to other elements. The Core, Shared and Control components comprise the main "Flow of Operations" in the enterprise architecture.

For a subset of the components in the firm's enterprise architecture, we analyze data on the cost of change. Specifically, we conduct an analysis that predicts the cost of change for each component, using measures of coupling derived from the DSM. We show that the cost to change highly-coupled components in this architecture is significantly higher than the cost to change components that are only loosely connected to others. Finally, we show that the "best" (i.e., most parsimonious and powerful) measure of coupling that predicts the cost of change is one that captures all of the *direct and indirect* connections between components in the architecture.

The main contribution of our paper lies in developing a robust methodology for understanding enterprise architecture. 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 use data on the information systems of a real firm to highlight the application of our methods. We conclude by relating our findings to the theoretical principles underlying enterprise architecture and discuss the implications for practicing managers.

The paper is organized as follows. Section 2 reviews the literature and positions our work within it. Section 3 describes our data and how it can be used to construct an enterprise architecture DSM. Section 4 shows how a DSM can be used to divide components into "Core" and "Peripheral" elements. Section 5 explores which measures of coupling best predict change cost for a subset of components. Finally, section 6 discusses our results and conclusions.

2. Literature Review and Motivation

In this section, we review the enterprise architecture literature and position our work within it. We follow this by describing recent work on the visualization and measurement of complex software systems, using network-based DSM approaches.

2.1 Enterprise architecture

In many prior studies, EA is defined as a tool for achieving alignment between the perspectives of business and IT. MIT's Center for Information Systems Research defines enterprise architecture as "the organizing logic for business processes and IT infrastructure reflecting the integration and standardization requirements of the company's operating model" (Weill, 2007). Hence prior work on enterprise architecture often emphasizes conceptual models, tools and frameworks that aim to align a firm's business processes with its IT infrastructure (e.g., Aier and Winter, 2009). In sum, most research has focused on the "strategic" level of EA (Tamm, 2011; Aier, 2014, Boh & Yellin, 2007; Ross & Weill, 2006).

This overarching perspective is present in the ISO/IEC/IEEE 42010:2010 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, 2007). As a consequence of this broad scope, EA analysis is typically not limited to IT systems, but encompasses the relationship with and support of business entities. Ultimately, EA targets a holistic and unified scope of organization (Rohloff, 2008; Tyler, 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 EA frameworks have been proposed, which detail the kind of entities that should be part of this effort (e.g., TOGAF, 2009; Lankhorst, 2009; DoDAF, 2007; MODAF 2008). Multiple approaches have been proposed that identify 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). However, these approaches rarely measure the structure or performance of the resulting architecture in a quantitative fashion. Furthermore, the visualizations that result from these models often contain limited information (i.e., are too simple) or too much information (i.e., are too complex) to be a source of insight for managers.

EA has been shown to be a powerful 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). Narrow models can help stakeholders to document and understand how EA affects specific attributes, generating insight into current and future states. They also help to communicate these states to stakeholders in a concrete fashion (i.e., with a focus on specific performance outcomes). Despite this success however, few studies have adopted this narrower, more practical, approach to EA modeling.

In sum, operationalizing enterprise architecture in a robust and reliable way that allows firms to analyze and improve their architectures has proven an elusive goal, due to the wealth of available models and frameworks, and their conceptual nature. Recent work has attempted to bring the topic of enterprise architecture to a more concrete and granular level, applying metrics developed in social network theory to the analysis of a firm's information systems (Dreyfus and Wyner, 2011). However there is as yet, no consensus on which metrics are most relevant to the analysis of enterprise architecture, or how these metrics can be used to visualize and understand a firm's architecture, thereby revealing its structure and properties (Baldwin et al, 2014).

2.1.1 Enterprise Architecture and Layering

While there exist a wide variety of different 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 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) introduce the concept of an IT "ecosystem," highlighting the different roles played by products and applications, component technologies and infrastructure technologies.

While these and other studies differ in the ways that they classify layers in an enterprise architecture, they do share important underlying assumptions. First, layering reflects a division of the functions provided by a system into different units, such that these units can be designed, developed, used and updated in an independent fashion. Second, layering establishes a design "hierarchy" (Clark, 1985) such that each layer tends to interact only with layers immediately above or below it in the hierarchy, reducing complexity. Finally, the direction of the interdependencies between layers is defined such that higher layers "use" lower layers, but not the reverse. This limits the potential for changes to propagate through a system (Gao and Iyer, 2006). For example, consider a software application on a desktop computer. It "uses" functions provided by the operating system, hence the application "depends upon" this lower layer in the system architecture. However, the reverse is not true. The operating system is not (in general) affected by the presence, or absence, of specific applications. 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 layer.

The prevalence of the concept of layering in the enterprise architecture literature suggests an important criterion for any methodology that aims to operationalize this concept. Specifically, the methodology should reveal the layered structure of an enterprise architecture, given a set of directed dependency relationships between elements in this architecture.

2.1.2 Enterprise Architecture and Performance

Much of the literature on EA has been dedicated to conceptual frameworks that help align business needs with IT capabilities (Zachman, 1987; TOGAF, 2009), and managerial processes by which EA planning is accomplished. Surprisingly however, there has been little work to explore the performance benefits of different architectures, especially 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 activities and improved performance outcomes.

Studies that do make claims about the performance benefits of effective enterprise architecture often identify a range of "enablers" that drive important firm outcomes. For example, Tamm et al. (2011) identify four themes discussed in the prior literature as potential mediators of firm outcomes, including better organizational alignment, improved information quality and availability, optimized resource allocation across the business portfolio, and increased complementarities between resources. However, many authors note it is difficult to directly assess the *quality* of a specific enterprise architecture. Hence empirical studies linking enterprise architecture to performance have tended to focus either on assessing the quality of *outputs* from EA planning (e.g., the quality of documentation) or the quality of the EA planning *process* itself (e.g., goal setting, task definition, and governance). Unfortunately, such an approach means it is impossible to differentiate between firms that possess similar EA planning processes and documentation, but have different structures in terms of the architecture in use.

While many diverse benefits are claimed from effective enterprise architecture, a consistent theme that emerges is the role of EA in facilitating *flexibility*. A well-designed enterprise architecture supports a firm's business strategy and achieves an efficient allocation of resources at a point in time. However, the nature of competition means that the context for these decisions shifts over time. Designing enterprise architecture is therefore not a static optimization problem. Rather, the aim is to ensure a *continuous* fit between business and IT needs over time.

Much prior work highlights the role that enterprise architecture plays in facilitating flexibility. For example, in a highly influential article, 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). In essence, EA provides a platform for agility. Duncan (1995) explores the elements that contribute to such flexibility, finding that managers associate this feature with the constructs of compatibility, connectivity, and modularity. Schmidt and Buxmann (2011) measure these attributes, and show that a rigorous and comprehensive enterprise architecture planning process is associated with self-reported improvements on each dimension. Finally, Sambamurthy and Zmud (2000) state that the new organizing logic for enterprise IT is the "platform," which constitutes a "flexible combination of resources, routines and structures" which meets the needs of both current and future IT-enabled functionalities.

The above discussion reveals additional criteria for a methodology that aims to operationalize enterprise architecture. Specifically, the methodology should capture data about the *actual* enterprise architecture in use, not merely the processes by which the architecture was developed, or the documentation produced. Furthermore, the visualizations and measurements output from this methodology should facilitate the analysis of important performance outcomes, and in particular, the extent to which the architecture facilitates flexibility.

2.2 Design Structure Matrices (DSMs)

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] However, while the use of graph theory and network measures to analyze coupling in software systems has a long history (Hall and Preiser, 1984), no clear consensus has emerged on the best coupling metrics to use when predicting system performance. In recent years, a number of studies have adopted measures from social network theory to analyze coupling in software systems (Dreyfus and Wyner, 2011;

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

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 (see Baldwin et al, 2014 for a discussion).[4]

An increasingly popular network-based method 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) have shown that DSMs can be used to visualize the "hidden structure" of software systems, by capturing the level of coupling between components. This method has been used to measure the structure, evolution and performance of *individual* applications, such as Linux and Mozilla (e.g., MacCormack et al., 2006; 2012). In recent work, Lagerström et al. (2013) have shown that this method can be applied to 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.

Metrics that capture the level of coupling for each component can be calculated from a DSM and used to visualize, analyze and 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. And Sturtevant (2013) has shown that software components with high levels of coupling tend to experience more defects, take more

[4] For example, social network measures tend to assume that dependencies are symmetric, and focus mostly on the direct dependencies between elements. In technical systems, many important dependencies are asymmetric, and both direct and indirect dependencies are important to analyze.

time to adapt and are associated with high employee turnover. Network metrics derived from DSMs (e.g., "propagation cost") have also been used to explore the value that might be derived from "re-factoring" designs with poor architectural properties (Ozkaya, 2012).

2.2.1 Design Structure Matrices, Coupling and Change Propagation

A DSM captures all of the dependencies that exist between components in a system. If component A depends upon component B, then any change made to B may affect A. These two components are "coupled." Using a DSM, we can analyze the direct dependencies between components. But we can also analyze the "indirect" dependencies between components, which reflect the possibility that a change may propagate through a system, via a "chain" of direct dependencies. For the example above, 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 direct and indirect coupling in a system provides an indication of the degree to which changes can propagate in a system. Prior work has shown that measures of direct and indirect coupling predict the likelihood of defects and the ease (or difficulty) of adapting a system, among other outcomes (MacCormack, 2010; Sturtevant, 2013).

A DSM is not the only network analysis technique to reveal direct and indirect linkages between components. However, in contrast to techniques such as social network analysis, a DSM also captures information on the *direction* of dependencies. This distinction is important, given dependencies in technical systems are often not symmetric. In the example above, A depends upon B, but that does not imply that B depends upon A. Hence a change to B may propagate to A, whereas A may be changed with no impact on B. A DSM allows the analyst to identify the direction of dependencies, and therefore to assess the "flow of control" in a system (i.e., in which direction chains of dependencies propagate). This, in turn, allows the analyst to discern between systems that are hierarchical in nature (i.e., there exists a strict hierarchical ordering of components) versus structures that are cyclical in nature (i.e., there are groups of components that are mutually interdependent). Hierarchy and cyclicality are important constructs for understanding how changes propagate in complex systems.

3. Using a DSM to Display a Firm's Enterprise Architecture

3.1 The Empirical Context

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

Our study site is the research division of a US biopharmaceutical company "BioPharma" investigated by (Dreyfus, 2009). 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). For a subset of software applications in the firm, data was collected on the cost of change (i.e., the inverse of "flexibility"). Prior work used these data to explore the impact of social network metrics on flexibility (Dreyfus and Wyner, 2011). We extend this work by i) introducing a formal methodology by which to visualize and measure the firm's enterprise architecture, and ii) comparing the impact of different measures of coupling derived from this architecture, on the cost of making changes to the firm's software applications.

Our BioPharma dataset includes information on 407 architectural components and 1,157 dependencies. The architectural components are divided into: eight "business groups;" 191 "software applications;" 92 "schemas;" 49 "application servers;" 47 "database instances;" and 20 "database hosts". "Business groups" are organizational units not technical objects. As noted earlier, the dependence of particular business groups on specific software applications and infrastructure is relevant to studies of enterprise architecture. We consider business groups to be part of the overall enterprise architecture, and include them in our analysis.

The components in the dataset form a layered architecture typical of modern information systems, as we will show later. Furthermore, there are important dependencies between these layers. In this case, 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" imply a directional (i.e., asymmetric) dependency. In contrast, "communicates with" is a bi-directional (i.e., symmetric) dependency.

3.2 Construction of the DSM matrix

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.). But what kinds of linkages between the components should be captured, and how should these be displayed and counted?

In understanding change propagation, 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 of 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 a symmetric relationship. If B uses A, but A does not use B, then B's behavior can change without affecting A. (We ignore indirect usage in this example.) A DSM reveals this asymmetry – the marks in rows denote one direction of the use relationship and the marks in columns denote the other. When usage is symmetric (i.e., B uses A *and* A uses B), the marks are symmetric around the main diagonal of the DSM.

Whether use proceeds from row to column or column to row is a choice to be made by the analyst. There is no standard practice or convention among DSM scholars. However, just as cars should drive on the left or the right to avoid collision, enterprises 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 on) the components in a given column. Specifically, to find the *i*th component of the system, one looks to the *i*th element on the main diagonal. To identify the components it depends on, one looks along its row. To identify the components that depend on it, one looks up and down its column. Consequently, change vulnerability (i.e., "influence") proceeds from column to row.

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 Enterprise Architecture layer diagrams place users on the top. In constructing

DSMs, however, we depart from this practice, and place users below the objects they use. Our reasons for doing so are based upon the concept of design sequence, described below.

When used as a planning tool in design processes, a DSM can indicate 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. Further, 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 at the bottom. This convention ensures that design rules and requirements, which affect subsequent design choices, appear at the top of the DSM.

The next question to answer when constructing a DSM is how should the dependencies between elements be counted? Should the DSM be binary, comprised only of ones and zeros, or should the links have ordinal values? When the components of a system are complex entities in their own right (e.g., like applications, schemas and servers), there can be multiple ways that one component uses or depends upon another. For example, Application B may make different types of requests of Application A on different occasions. 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 might interpret off-diagonal entries in the DSM as probabilities. In this case, a "1" would indicate 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 may 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. Hence we use a *binary* DSM as the baseline for analyzing enterprise architecture.[5]

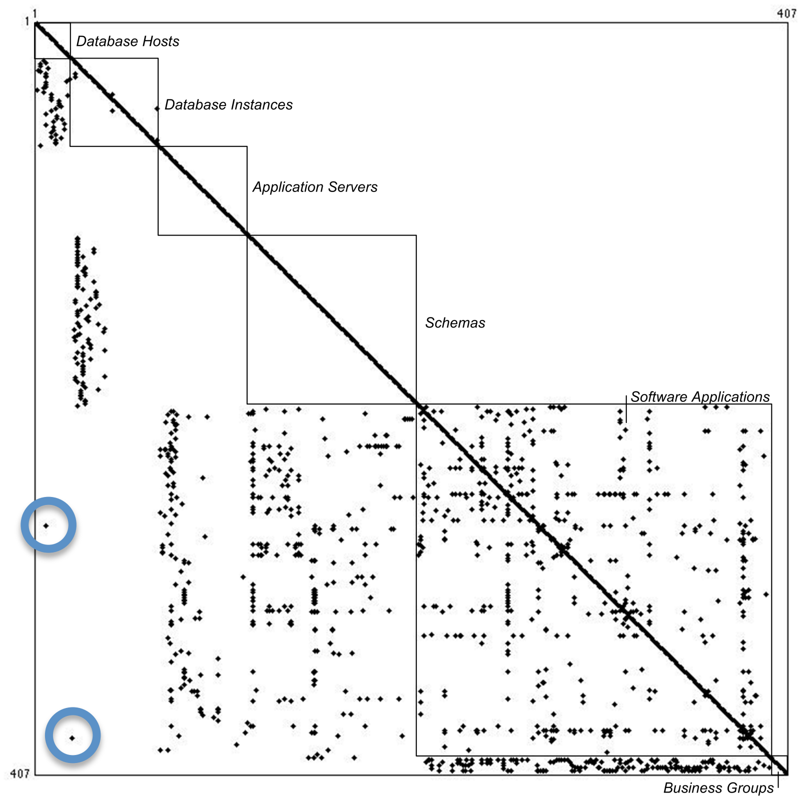
Data on the dependencies in a system can be obtained via automatic or manual methods. As Eppinger and Browning (2012) state: "for most product DSM models, the data collection requires at least some amount of direct discussion with subject matter experts in order to draw out the tacit and system-level 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 the analysis. Software provides an exception, in that past studies rely upon automatic dependency extractors supplied by commercial vendors to help developers understand code (e.g., Cataldo et al., 2006; MacCormack et al., 2006, 2012; Sosa et al., 2013.)

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 (for details, see Dreyfus, 2009). Importantly, many of the 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 deviated substantially from the actual architecture "in use".

[5] Further research may allow the strength of ties or change probabilities to be used to refine this baseline.

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 relationship captured for 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.

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, this DSM confirms that the 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 as recommended in the TOGAF 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.

The Application Interaction Matrix (AIM) is the largest submatrix in the DSM. It shows dependencies caused by interactions between applications in the enterprise's software portfolio. In our 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 on directionality is desirable. In some cases, one application may always ask for a computation, and the other may always supply the result. This hierarchical distinction would be obscured if all dependencies are assumed symmetric. However, if applications switch roles, sometimes requesting and sometimes supplying computational services, a symmetric dependency would be warranted.

4. Analyzing an Enterprise Architecture DSM

Figure 1 displays the layered structure of the enterprise architecture, but does not reveal other architectural characteristics such as indirect coupling, cyclic coupling, hierarchy within groups, modules, or the presence of "core" and "peripheral" components. Matrix operations with a DSM can be used to analyze these additional features. Specifically, the transitive closure of the matrix reveals the 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 mathematical procedure of transitive closure to a matrix 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 similar fashion to a 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. In cases where layers are not known *a priori*, the Visibility DSM can be sorted by VFI and VFO to reveal the hierarchical relationships among components/layers.[6]

VFI and VFO measures can be used to identify cyclic groups of components, each of which is directly or indirectly connected to all the 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 it is easy to identify cyclic groups in a system by sorting on these measures after performing a transitive closure on the direct dependency matrix (Baldwin et al., 2014). Large cyclic groups are problematic for system designers, given a series of changes may propagate via a chain of dependencies to many other components. In such a structure, there is no guarantee that the design process or a change to the design 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 system-level measure of the level of coupling in a design. 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. Dramatic differences in propagation cost have been observed across systems of similar size and function (MacCormack et al., 2012). Yet empirical evidence also suggests that refactoring efforts aimed at increasing modularity can have a major impact on lowering cost (MacCormack et al., 2006; Akaikine, 2009). These findings suggest that at least in software, architecture is not dictated solely by system function, but is under the control of a system's designers.

[6] Note that matrix methods can reveal hierarchical relationships, but will not tease apart discrete groups of components that have equivalent positions in the hierarchy.

In a recent empirical study Baldwin et al. (2014) used this DSM methodology to analyze 1286 software releases from 17 distinct 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 how the components in such systems can be divided into groups – Core, Peripheral, Shared and Control – that share similar properties in terms of coupling. In such systems, dependencies flow from Control components through Core components, to Shared components. Peripheral components lie outside the "Main Flow" of operations in the system.

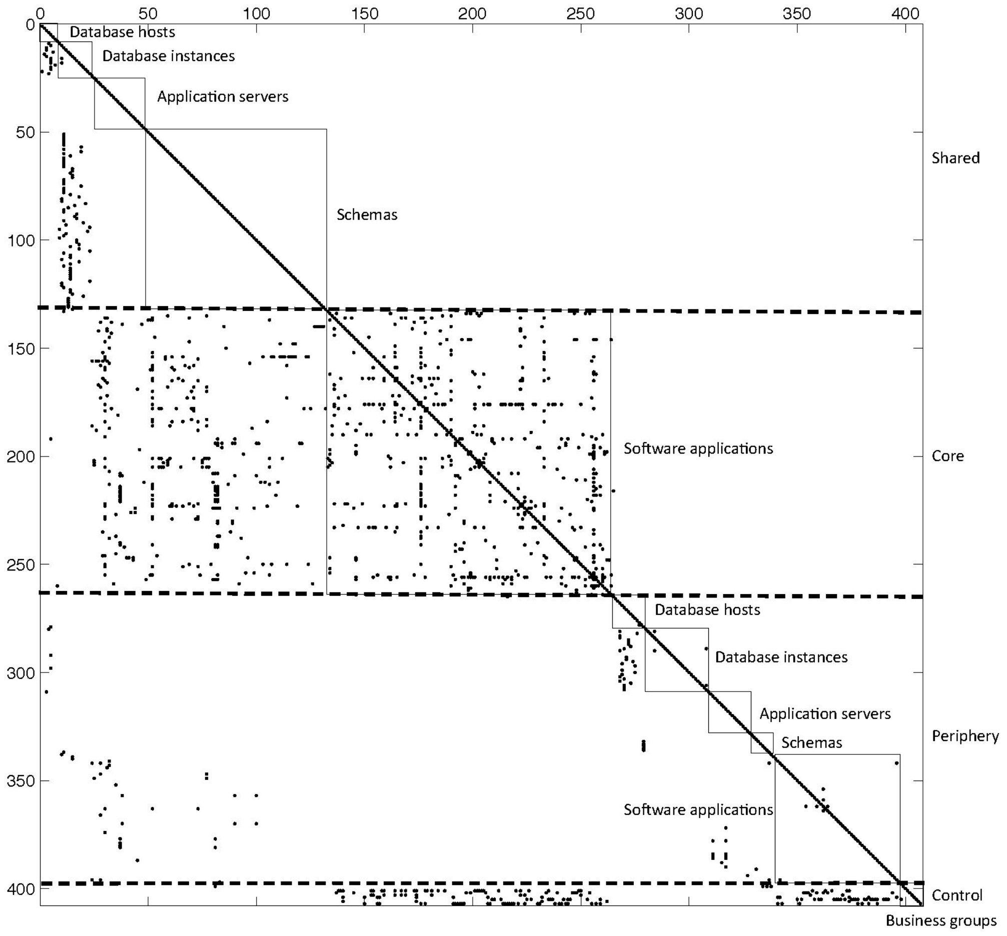
We constructed the Visibility Matrix for BioPharma, and applied the classification methodology described in Baldwin et al. (2014) to the data. We find that the firm's enterprise architecture has a core-periphery structure. Specifically, each layer in Figure 1 has some components that are part of the "Main Flow" and other components that lie in the "Periphery." The distribution of components organized by layer and category is shown in Table 1. It shows that 2/3 of the technical resources in the enterprise architecture are part of the main flow of operations, with 1/3 in the periphery. We believe managers can use this type of classification scheme to set priorities, allocate resources, analyze costs, and understand productivity.

Table 1: Distribution of BioPharma Components 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 DSM in which each layer of the enterprise architecture is divided into components that are part of the main flow versus those that are peripheral. Main-flow components are directly or indirectly connected to all Core components as well as other components. Thus each main-flow component is connected to at least 132 other components. In contrast, the highest level of coupling for any peripheral component is 7. Hence the indirect coupling levels of components in the main flow and the periphery are dramatically different. If coupling is related to the cost of change, then the main-flow components will have higher change costs than the peripheral components. We investigate this empirically in the next section.

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



5. Enterprise Architecture, Component Coupling and IT Flexibility

In this section we show how a DSM generates data that allows us to analyze the flexibility of an enterprise to change its IT systems over time. We focus on flexibility as a performance measure, given the importance of this theme in prior literature. We use a subset of the data from BioPharma, analyzing components for which information on the cost of change is available. Our purpose is to highlight how our methodology provides a generalizable approach to analyzing the relationship between enterprise architecture and measures of performance. But first, we introduce a number of hypotheses about the relationship between different measures of coupling derived from an enterprise architecture DSM and enterprise IT flexibility.

5.1 Hypothesis Formulation

In the previous section, we found that BioPharma's enterprise architecture is comprised of architecturally heterogeneous components. 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 is known about how different types of component coupling affect the costs of changing a system's components. These costs determine the flexibility of the firm to evolve and adapt its IT systems in future.

Coupling theory and design theory both predict that the more coupled or connected a component is, the more difficult, expensive, or risky it will be to change. 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. And components that are cyclically connected may be members of a large or a small cyclic group. Finally, components may be classified as Core, Peripheral, Shared or Control elements.

Measures of these (and other) types of coupling can be calculated directly from an enterprise architecture DSM, and used to evaluate their impact on the cost of change for each component. In this study, we focus in particular on three related coupling measures:

(1) The level of *Direct Coupling* of each component, which is calculated directly from the rows and columns of the Direct Dependency Matrix.

(2) The level of *Indirect Coupling* of each component, which is captured by its category as derived from the Visibility Matrix (i.e., Core, Peripheral, Shared and Control).

(3) The *Closeness Centrality* of each component, which can be calculated for components in the Core (i.e., for components that are in the same network).

In our dataset, data on the cost of change was available only for software applications, whose dependency relationships are symmetric. Hence in this case, it was not possible to explore the impact of differences between the number of incoming and outgoing dependencies for components (i.e., given that these numbers are identical). Furthermore, it was not possible to explore differences in the hierarchical classification of components (i.e., given that a symmetric DSM contains only Core and Peripheral elements, with no Shared or Control elements). In general however, our methodology allows the analyst to examine all of these issues.

Our measures of coupling are likely to be highly correlated. Specifically, components with high levels of direct coupling are more likely to be in the Core. Furthermore, as noted above, closeness "centrality" is only defined for components in the Core (i.e., components in the same network). Finally, Core components with high levels of direct coupling are more likely to have higher "centrality". These relationships mean we must be sensitive to issues of multi-collinearity. To address this issue, we conduct our tests in two stages. First, we explore the impact of Direct and Indirect coupling on flexibility across all the components for which we have data on change cost. Then, for the "Core" components in the system, we test whether the closeness centrality measure provides additional power to explain differences in flexibility.

Stage 1: Direct coupling vs. Core membership. Following Chidamber and Kemerer (1994), we define direct coupling (DC) as the number of direct linkages between a given software application and all others. (Because software linkages are symmetric in the dataset, the number of incoming and outgoing dependencies is the same.) We define Core membership (CORE) as being part of the largest cyclic group, revealed by the transitive closure of the DSM. (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 an enterprise architecture.) As discussed earlier, all members of the Core have *the same number* of direct and indirect dependencies, given they are all directly or indirectly, inter-dependent.

Coupling theory predicts that higher levels of direct coupling will lead to higher change cost. The theory of change propagation predicts higher levels of indirect and cyclic coupling, such as arise in Core components, will also lead to higher change cost. These variables might have additive effects, or they might be substitutes. We thus form 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: Core membership (CORE) adds explanatory power to Direct Coupling (DC) in estimating change cost (CC).

H4: Direct Coupling (DC) adds explanatory power to Core membership (CORE) in estimating change cost.

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.

Stage 2: Closeness Centrality within the Core. For components in the Core, the variable "closeness centrality" (CENT) is calculated for a given component 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).[7] The higher this number, the more "central" is the component. Our final hypothesis explores the theory that closeness centrality may explain variations in change cost for all the components that are inter-connected (i.e., that are members of the Core, and hence are part of the same network):

H5: Within 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 flexibility *only* for the subset of components that are classified as being in the Core.

5.2 Dependent Variable: The Cost to Change a Component

Ironically, in this era of "big data," the lack of appropriately granular data is a large barrier to the systematic investigation of the architecture of information systems. Very few enterprises systematically collect data on change costs, defect rates, or productivity *by component*. To demonstrate the application of our methodology, we use data on the cost to change the software applications at the center of BioPharma's enterprise architecture. Focusing on one layer of the firm's architecture (i.e., as compared to capturing data on all layers) allowed us to i) focus on one type of respondent for data collection, ii) request specific data from respondents, and iii) ensure the data was directly comparable across observations.

[7] In prior work, the closeness centrality for nodes that are not part of a network is sometimes assumed to be zero (i.e., denoting an infinite path between these and other nodes). A separate issue is that closeness centrality 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. Social network measures cannot handle such complexity easily. Instead, they tend to assume all dependencies are bi-directional – which is not the norm in technical systems.

The cost to change each application was assessed by administering a survey to IT Service Owners. Specifically, respondents were asked to estimate the time, in person-years, to perform five operations: deploy, upgrade, replace, decommission, and integrate. These 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).[8]

We received survey responses covering 99 software applications. The change cost estimates reported 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 reported as unknown were removed from the dataset, leaving us with a final set of 77 software applications for analysis.[9] (In the enterprise architecture, 58 of these were classified as Core and 19 were classified as Peripheral.)

For the purposes of our analysis, we combined the change cost estimates for different operations into a single measure of the change cost for each. To do this, we calculated the mean change cost for each application, across all operations for which a response had been provided. The Cronbach's alpha for our aggregate measure of change cost is 0.78.[10]

[8] 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 further details, see Dreyfus, 2009.

[9] In prior work, Dreyfus (2009) and Dreyfus and Wyner (2011) apply different screening criteria to this data, resulting in a set of 62 responses. Later, we discuss the sensitivity of our results to the use of other criteria.

[10] We note that missing values for the change cost associated with individual operations will affect the calculation of Cronbach's alpha. Where the estimate for an operation is missing, we substitute the mean level of change cost estimated for that operation across all respondents to calculate 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).

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 his or her appraisal of change cost in a systematic way. Data on the following variables were collected and included as controls:

(1) *VENDOR* indicates whether an application is developed by a vendor (1) or developed in-house (0). One component missing an entry for this variable was assigned a value of 0.5. (Omitting this data point did not change the results significantly.)

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

(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 also include data on closeness centrality 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, be more likely to perform computations, be less likely to have N-tier architectures, and be 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 for all Variables.

	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 (controls only) show two of the controls are significant: Vendor provided applications tend to have lower change costs and Active applications tend to have higher change costs. Control variables alone explain 18% of the variation in change cost across components.

H2 predicted that direct coupling would be positively correlated with change cost, a relationship that was observed in the raw correlations reported earlier. In model 2 however, a multivariate regression with control variables included, we find direct coupling is only a weak predictor of change cost (p-value = .06), and does not quite reach the threshold for statistical significance. This model explains 21% of the variation in change cost across components. In model 3, we find that Core membership is a highly significant predictor of change cost (p-value

= .005) and has a positive sign. This model explains 26% of the variation in change cost across components, a significant increase relative to the model with only controls. In model 4, when both direct coupling and Core membership are included in the same regression, only Core membership is significant. This model explains 25% of the variation in change cost across components. Hence adding direct coupling to a model that already includes Core membership as a predictor makes the model *worse*. Whatever power direct coupling has as an explanatory variable appears to be fully accounted for by its correlation with Core membership. It does not appear to add additional explanatory power over and above Core membership. In sum, hypotheses *H1*, *H2*, and *H3* appear to be supported by these results, while *H4* is rejected.

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

These initial results are somewhat surprising. Specifically, we find that the measure of component coupling most associated with IT flexibility is not the number of *direct* dependencies that a component has with others, but the set of all direct *and indirect* dependencies it possesses. This suggests that the ability to change a component is profoundly impacted by the potential for changes to propagate from one component to others via chains of dependencies, and hence impact components that may not be obvious from a simple inspection of its "nearest neighbors." The results suggest that the methodology we employ, which focuses on understanding the indirect linkages between components using network-based representations of architecture, adds significant value to prior methods, which tend to be more abstract and conceptual in nature.

In models 5 and 6, we examine only the 59 components in the Core (the largest group of components that are directly and indirectly connected to each other). Model 5 contains only control variables, and produces results very consistent with model 1, as expected. Model 6 includes the measure of closeness centrality for Core components. The measure is not significant in this model. Hence centrality does not provide additional explanatory power in predicting change cost, over and above that provided by Core membership, and hypothesis *H5* is rejected.

5.6 Robustness Checks

We performed a number of robustness checks on our statistical results to assess whether they were sensitive to other assumptions or specifications of variables. First, we note that our basic specification does not control for the size of each component, a control variable that could plausibly affect the cost of making changes. Data on component size (as measured by the number of lines of code and the number of files in each) was available for a subsample of 60 applications in the dataset (Dreyfus, 2009). We ran our models on this smaller sample, including these control variables. We found that the size controls were insignificant and, as expected, the significance level of our control and explanatory variables declined as a result of the decrease in sample size (hence statistical power). However, the results were very consistent with those reported above. This first check therefore indicates that our main results are valid and robust.

We conducted a further test to explore the possibility that transformations of Direct Coupling (DC) might be better predictors of change cost, given this variable has a skewed distribution (many lower values, and fewer higher values) which is truncated at zero. To assess this possibility, we tested the predictive power of the logarithm of Direct Coupling (Ln(DC)). This variable achieved greater statistical significance in a model where only control variables were present (i.e., model 2 in Table 3). However, this variable still explained less of the variation in change cost across the sample than our indicator of Core membership. Hence this second check also indicates that our main results are valid and robust.

Finally, we explored whether Direct Coupling, or transformations of Direct Coupling, would contribute to explaining the remaining variation in change cost among Core components (in the same way we did for the centrality measure). Appendix A reports the results of three models predicting change cost, the first being a model just with controls, the second being a model including controls and Direct Coupling, and the third being a model including controls and the logarithm of Direct Coupling (Ln(DC)). In neither the second or third model is Direct Coupling statistically significant. This suggests that in our dataset, Core membership is the most parsimonious and powerful measure of coupling that explains the cost of change for components. Neither of the other measures of coupling contributes additional explanatory power.

6. Discussion

The main contribution of this paper is in demonstrating a robust and repeatable network-based methodology by which to visualize and measure any firm's enterprise architecture. Our methodology is consistent with prior theoretical work in this area, and addresses several important criteria suggested by a review of this prior work. Specifically, the methodology can i) integrate the consideration of both business and IT related aspects of architecture; ii) identify the distinct layers in the architecture associated with different architectural aspects (e.g., differentiating between application and database technologies); iii) reveal the main "flow of control" within the architecture and its associated layers, as denoted by the classification of components into Shared, Core, Peripheral and Control elements, and: iv) generate measures of component coupling that can be used to predict system reliability, adaptability and performance. Ultimately, this methodology generates insights that cannot be gained from an inspection of documents or processes associated with the enterprise architecture.

A second contribution of this paper lies in exploring the dynamics of how different types of coupling influence the flexibility of enterprise architectures. Specifically, we investigate three related measures of coupling that have been explored in prior work, and assess their relative power in predicting the costs of making a change to a component. We show that the best measure of coupling (i.e., the one most associated with the cost of change) in this dataset is a measure that captures both the *direct and indirect* linkages between components. Once the variation in change cost explained by this measure of coupling is accounted for, other measures of coupling (i.e., a measure of direct coupling, and the "centrality" of each component) add no further explanatory power. While we cannot draw firm conclusions from a single study, the data does indicate that when assessing the flexibility of an enterprise architecture, it is important to capture all possible ways that change might propagate through the system (i.e., via chains of direct dependencies).

For practicing managers, our methodology promises to provide a clearer understanding of the actual *instantiated* architecture that they must manage, as opposed to the "idealistic scenario" often presented in documents and representations of their organization's IT systems. The insights gained from this methodology can be useful in several areas, including i) helping to plan the allocation of resources to different systems, given information about the relative ease/difficulty with which different components can be changed; ii) monitoring the evolution of the enterprise architecture over time, as new components and/or dependencies are introduced (e.g., as the result of an acquisition) and, iii) identifying opportunities for system redesign or refactoring, (e.g., to reduce coupling in the system, and thereby improve system performance).

An important caveat to the benefits listed above however, is the need for a way to generate and access the granular data needed in order to operationalize our methods. Specifically, there is a need to collect data on the dependencies between different elements in the enterprise architecture, the direction of these dependencies, and the way that they evolve over time. In most organizations we are aware of, this type of data does not readily exist. In some, efforts have been made to collect data manually, although there are many challenges associated with this approach, including a lack of incentives for managers to provide accurate and timely information. Furthermore, in our study, we found substantial omissions in the data collected using a manual process, in comparison to the automated tools we used to uncover dependencies. In essence, firms often do not actually know the "real" enterprise architecture that they possess.

The best solution to this problem would be to develop automated ways to detect and capture important dependencies between components in an architecture. This implies the need for a significant level of investment by firms who wish to adopt these methods. However, we believe the potential benefits associated with these investments would more than recoup the cost, given the increase in understanding of enterprise architecture that would result, along with the "option value" generated by identifying possible improvements to it.

For academia, this study builds on prior work in the area of enterprise architecture, and in doing so, satisfies several important criteria by which a methodology for operationalizing EA should be judged. First, it integrates both business and technology-related aspects of architecture, and in doing so, reveals the interdependencies between these two arenas. Second, it makes what has previously been a rather conceptual field of study more concrete, providing a mechanism by which to analyze the actual architecture of a firm's information systems, as opposed to merely judging the quality of the processes or documentation by which the firm's EA is managed. Finally, our methodology outputs metrics that capture the level of coupling between different elements in a firm's architecture, which are a driver of a firm's ability to respond quickly to business challenges. In our setting, we demonstrate that the best measure of coupling is one that captures all of the direct and indirect dependencies between components in the architecture.

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

Of course, the data for demonstrating our methods came from a single division within one firm in the bio-pharmaceutical industry. Hence more studies are needed to provide validation of our methodology across different contexts, and to deepen our understanding of how it should be applied in more complex scenarios. For example, work is needed to validate the different layers/components that should be included in the analysis of enterprise architecture, as well as the different types of dependency that exist between these layers/components. We may find that different types of dependency (e.g., "uses" versus "communicates with") have differing predictive powers depending upon the performance dimensions being analyzed. We may find that different measures of the patterns of coupling generated by dependencies (e.g., direct versus indirect coupling) may predict performance differently in different settings. Ultimately, we believe our research provides a platform that will enable others to confront questions that until now have gone unanswered, given the relative scarcity of empirical methods. We hope future researchers will improve and evolve this platform, in order that we all benefit from the collective knowledge of how it can be applied.

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. 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.

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.

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.

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.

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

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

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.

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.

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.

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., 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., 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.

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.

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.

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.

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