HARVARD | BUSINESS | SCHOOL

Digital Agility: The Impact of Software Portfolio Architecture on IT System Evolution

Alan MacCormack
Martin Mocker
Robert Lagerstrom
Carliss Y. Baldwin

Working Paper 17-105

Digital Agility: The Impact of Software Portfolio Architecture on IT System Evolution

Alan MacCormack
Harvard Business School

Robert Lagerstrom
KTH Royal Institute of Technology

Martin Mocker
Reutlingen University

Carliss Y. Baldwin
Harvard Business School

Working Paper 17-105

Digital Agility: The Impact of Software Portfolio Architecture on IT System Evolution

Alan MacCormack[1]
Robert Lagerstrom[2]
Martin Mocker[3]
Carliss Y. Baldwin[1]

[1] Harvard Business School, Harvard University, Boston, USA
[2] KTH Royal Institute of Technology, Stockholm, Sweden
[3] ESB Business School, Reutlingen University, Reutlingen, Germany

Keywords: Information Systems, Software, Architecture, Modularity, Agility, Coupling.

Abstract

The modern industrial firm increasingly relies on software to support its competitive position. However, the uncertain and dynamic nature of today's global marketplace dictates that this software be continually evolved and adapted, to meet new business challenges. This ability – to rapidly update, improve, remove, replace, and reimagine the software applications that underpin a firm's competitive position – is at the heart of what has been called IT agility. Unfortunately, we have little understanding of the antecedents of IT agility, specifically with respect to the choices that a firm makes when designing its portfolio of software applications.

In this paper, we explore the relationship between software portfolio architecture and IT agility. In particular, we use modular systems theory to examine how different types of coupling impact the ability to maintain, retire and commission new software applications. We test our hypotheses with a unique longitudinal dataset from a large financial services firm. Our sample comprises information on over 2,000 software applications observed over a 4-year period.

We find that applications with higher levels of coupling cost more to maintain, are less likely to be retired, and are less likely to be commissioned. However, we show specific types of coupling present greater challenges than others, in terms of their impact. In particular, applications that are *cyclically coupled* (i.e., mutually interdependent) are the most difficult to manage, in terms of maintaining and updating the software portfolio. Our results suggest that IT managers have a critical design role to play, in firms that seek enhanced digital agility.

1. Introduction

Software is eating the world. With this dramatic headline, Mark Andreessen, the influential venture capitalist and co-founder of Netscape, began a 2011 Wall Street Journal article on the impact that software was having on the global economy (Andreessen, 2011). Andreessen's argument was compelling. Software is not only increasingly embedded in products and services that are part of our daily lives; behind the scenes, it also plays a growing role in delivering the organizational capabilities needed to compete in a dynamic and uncertain world. Software-focused companies are creating massive amounts of wealth. But even traditional firms competing in the "physical world" now rely on software systems and services to power the critical operating capabilities needed to survive in a world of constant change.

As this process of digital transformation has accelerated across the economy, software systems within firms have become correspondingly more complex. Today, even a moderately sized business maintains information systems comprising hundreds of applications and databases, running on geographically distributed hardware platforms, and serving multiple clients. These systems must be reliable, efficient and secure enough to meet today's business challenges. Yet they must also be flexible and adaptable, capable of evolving to meet the new and emerging challenges that will undoubtedly arrive tomorrow. How should a firm design its portfolio of software applications to meet these potentially conflicting objectives?

Early academic work in this area focused on the first of these challenges: the design of an architecture optimized for a given set of business conditions and strategic choices. The resulting field of study, Enterprise Architecture (EA), yielded conceptual frameworks, processes and tools that helped to achieve alignment between a firm's business strategy and its IT system (Zachman, 1987; Weill, 2007). More recently, a distinct stream of research has begun to explore

how a firm's IT architecture can facilitate the development of *new* business capabilities (Orlikowski and Iacono, 2001; Sambamurthy, W. and Zmud, R. 2000). Those theories emphasize the need for systems that facilitate *agility*, through the use of layered, modular technologies (Yoo et al, 2010; Tanriverdi et al, 2010; Tiwana et al, 2010). Firms with modular architectures quickly reconfigure software resources to respond to new challenges, ensuring the continuous alignment of IT assets with changing business needs (Sambumurthy et al, 2003; Hanseth and Lyytinen, 2010). The overarching goal for these firms is optionality, and not optimality.

The studies cited above, and a host of others, have made strong conceptual and theoretical contributions to our understanding of the role of IT architecture in the modern firm. Unfortunately, robust empirical work exploring the link between IT architecture and agility has been rare and yields mixed results (Tiwana and Konsynski, 2010; Schmidt and Buxmann, 2011; Kim et al, 2011; Liu et al, 2013). Several common challenges are observed in these studies. First, they rarely measure architecture directly, but instead capture broad-based *perceptions* of architectural characteristics (e.g., the extent of loosely-coupling). Second, they tend to adopt the firm as unit of analysis, hence measures of IT function and architecture are assumed to be homogenous across the organization, ignoring the inherent heterogeneity in a portfolio of IT assets (e.g., the mix of legacy and new technologies). Finally, they lack a consistent definition of agility, and seldom consider the fact that this ability is likely to encompass *multiple* dimensions of performance, associated with different types and levels of IT change.

We address these gaps in the literature by exploring the relationship between software portfolio architecture and IT agility at the level of the individual applications in the portfolio. In particular, we draw from modular systems theory to develop and test a series of hypotheses about how different types of coupling impact three dimensions of agility: the ability to maintain, to

retire and to commission new software applications. Our methods, which are based upon network analysis, allow us to measure the precise level of coupling for all applications in the portfolio. This contrasts with the broad perceptual measures often used in prior research.

We test our hypotheses with data from a large financial services firm. Our dataset comprises over 2,000 software applications and the dependencies between them. Critically, we capture data at two distinct points in time, four years apart. This approach allows us to identify changes in the portfolio, hence to develop measures of IT agility. We supplement this data with figures on the cost of maintenance for all applications in the portfolio at the start of this period.

We find differences in the level of coupling for applications explain large variations in IT agility. Specifically, applications with high levels of coupling cost more to update, are less likely to be retired, and are less likely to be commissioned (i.e., added to the portfolio). We show that the coupling between different applications has more power in explaining agility than the coupling between different layers in the IT architecture (e.g., between applications and databases). Finally, we show the measure of coupling that best predicts agility captures whether applications are *cyclically coupled* (i.e., are part of a group of mutually interdependent applications). These results deepen our understanding of how to design effective software portfolio architectures as well as improve existing architectures in order to enhance their agility.

The paper is organized as follows. In section 2, we review the literature that motivates our work. In section 3, we develop our theory and derive a number of hypotheses. In section 4, we describe our methods, which make use of network analysis to measure the different types of coupling between software applications. In section 5, we describe our empirical setting and our data. In section 6, we provide the results of our statistical tests. In section 7, we discuss the implications of our findings for academia and the world of practice.

2. Literature Review

2.1 IT Architecture Research

Early studies of IT Architecture were motivated by critiques of Enterprise Architecture research which noted the overwhelming emphasis on processes and governance structures by which IT is managed, as opposed to features of the technology itself (Orlikowski and Iacono, 2001; Tilson et al, 2010). Furthermore, the rapid rise of the Internet, World Wide Web and use of digital technologies had brought a need to revisit prior conceptions for the role of IT, to reflect the new dynamics of a digital age with its rapidly shifting competitive landscapes (Hansen and Lyytinen, 2010, Woodard et al, 2013). As a consequence, IT Architecture research has tended to focus more sharply on the "IT Artifact", and in particular, features of architecture that facilitate the development of new capabilities (Sambamurthy and Zmud, 2000).

Early work in the field focused on understanding desirable features of IT technologies, and in particular, the antecedents of a more flexible IT infrastructure. Duncan (1995) and Byrd and Turner (2000) defined several constructs that underpin such an infrastructure, emphasizing the need for systems with greater levels of *compatibility, connectivity and modularity* (i.e., loose coupling between applications, data and infrastructure). Subsequent work sought to deepen our understanding of how these concepts should be operationalized in firms competing in a dynamic, digitally-connected world. Sambamurthy and Zmud (2000) argued the organizing logic for IT architecture is the platform, encompassing a "flexible combination of resources, routines and structures" that facilitate agility by creating digital options and enhancing entrepreneurial alertness (Samburmathy at al, 2003). Adomavicius et al. (2008) defined the concept of an IT "ecosystem," highlighting the roles played by products, applications, component technologies and infrastructure technologies. Finally, Yoo et al. (2010) described how pervasive digitization

has given rise to a new "layered-modular" architecture, comprising devices, network technologies, services and content. Firms with layered, modular IT architectures can quickly reconfigure resources to respond to new challenges, creating a continuous stream of new capabilities (Tanriverdi et al, 2010; Tiwana et al, 2010). Yet layered, modular IT architectures are not easy to build, nor the norm in firms with complex infrastructures (Parker et al, 2016). Firms more often grapple with a mixture of systems of different vintages, designed using different frameworks, to meet the demands of different decision makers (Ross 2003). Moving to a layered, modular architecture requires firms to embrace new frameworks for the role of IT, and new structures by which the technologies will work together (Ross and Westerman, 2004).

2.2.1 Empirical Studies linking IT Architecture with IT agility

Empirical studies linking IT Architecture to agility have been scarce and limited in scope. Most work to date has been case-based, or used firm-level survey measures of IT infrastructure to demonstrate correlation with performance (Salmela, 2015). Schmidt and Buxmann (2011) showed that higher quality enterprise architecture planning processes are associated with more flexible IT infrastructures, as captured by the constructs of connectivity, compatibility and modularity. Kim et al (2011) showed measures of IT infrastructure flexibility, as captured by the constructs of compatibility, connectivity and modularity, are correlated with a firm's ability to change existing business processes. Conversely, Liu et al (2013) found IT infrastructure flexibility is *not* associated with agility, but contributes to performance only via its association with increased absorptive capacity (Cohen and Levinthal, 1990).

An important study in this stream of work is by Tiwana and Konsynski (2010) who characterized IT Architecture on two dimensions: loose coupling and standardization. They show that these measures are associated with an IT function that is perceived as agile, adaptive,

flexible and responsive. While this work informs our knowledge of the features of IT architecture that contribute to IT agility however, we still lack insight on the precise mechanisms through which these effects are manifested. The first challenge relates to the fact that in this and other studies, architecture is not measured directly, but is captured by the *perceptions* of organizational participants. Yet prior studies have shown that perceptions can diverge significantly from the "real" (i.e., instantiated) architecture that a firm possesses. There is often "hidden structure" that can be revealed only via more granular analysis (Baldwin et al, 2014).

The second challenge relates to the fact that in many studies, the firm is the unit of analysis; hence measures of IT function and architecture are assumed to be homogenous. But firms are not monolithic; they comprise differentiated organizational units, with different objectives and levels of flexibility (Lawrence and Lorsch, 1967). Similarly, the components of a firm's IT architecture are diverse, play different roles, are connected in different ways, and vary in the cost of adaptation (Sambumurthy and Zmud, 2000; Yoo et al, 2010).

The final challenge relates to the fact that prior studies lack a consistent definition of agility, and seldom consider that this ability encompasses *multiple* dimensions of performance. A firm's software applications must be maintained, improved, upgraded, ported to different platforms, decommissioned and/or replaced on a periodic basis. These activities place different demands on a firm's infrastructure, require different resources, and may be impacted differently by properties of the software portfolio.

This study aims to address the limitations described above. In particular, we examine the impact of a firm's software portfolio architecture at the level of the individual applications in the portfolio. Our approach captures the heterogeneity that exists across applications, and explores the impact of this heterogeneity on multiple dimensions of agility.

3. Theory Development

3.1 Modular Systems Theory

Architecture is defined as the scheme by which a system's functions are allocated to components and the way that these components interact (Ulrich, 1995; Whitney et al, 2004). Modularity is a concept that helps us to characterize different architectures (Sanchez and Mahoney, 1996; Schilling, 2000). It refers to the way that a system is "decomposed" into parts or modules (Simon, 1962). Although there are different definitions of modularity, authors agree on its fundamental features: the interdependence of decisions *within* modules, and the independence of decisions *between* modules (Mead and Conway, 1980; Baldwin and Clark, 2000). The latter is referred to as "loose-coupling." Modular systems are loosely coupled in that changes to one module have little or no impact on others (MacCormack et al, 2012).

Modular systems theory is the name given to a body of theory that explores the design of systems and the costs and benefits that arise from modular designs (Sanchez and Mahoney, 1996; Schilling, 2000; Baldwin and Clark, 2000). This theory has been broadly applied, to the study of biological systems, technical systems and organizational systems (Kauffman, 1993; Weick, 2001; Langlois 2002; Berente and Yoo, 2012). A central tenet of the theory is that modular systems (i.e., systems with loosely-coupled modules or components) can be adapted with lower costs and with greater speed, given changes are isolated within modules. Conversely, in a system with tight coupling, adaptation is costly and takes longer, given the potential for changes to propagate between different parts of the system.

Below, we develop theory about the different *types* of coupling that exist between applications in a firm's software portfolio architecture. We then define three distinct measures of IT agility that might be impacted by these different types of coupling.

3.2 Types of Coupling between Software Applications

The computer scientist David Parnas argued that, in technical systems, the most important form of linkage between components is a directed relationship 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 behavior may then need to change to accommodate the change in A. Hence change propagates in the opposite direction to use. Parnas stressed that use is not symmetric. If B uses A, but A does not use B, then B might change with no impact on A. Our first step in theory building is to relate Parnas' concept of dependency to component coupling. We define coupling as the property of being used (i.e., depended upon) by another component or using (i.e., depending upon) another component. Component A is coupled with component B if B uses A ("Fan-in" coupling) or if A uses B ("Fan-out" coupling).

Modular systems theory predicts the more coupled a component is, the more costly and time consuming it will be to change (Simon, 1962; Sanchez and Mahoney, 1996). However, the components of a system can be coupled in different ways (Baldwin and Clark, 2000). They can be coupled *directly or indirectly*; and they can be coupled *hierarchically or cyclically*. Furthermore, components that are hierarchically coupled may be located at the top or the bottom of the hierarchy (Clark, 1985); and components that are cyclically coupled may be members of a large or small cyclic group of components (Sosa et al, 2013). For the purposes of theory development, we consider a single application (called "A") within a firm's IT architecture. The question we explore is how does the presence of coupling (or the lack thereof) affect the cost of change for this application? To answer this question, we consider four different patterns of coupling that can exist between applications in a software portfolio (see **Figure 1**).

Figure 1: Types of Coupling between Software Applications

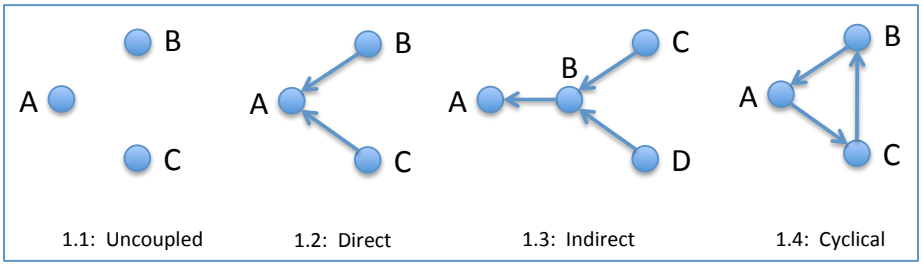


Figure 1.1 represents the base case, in which component A is not coupled to any other. In Figure 1.2, component A is *directly coupled* with components B and C. Modular systems theory predicts that components with higher levels of direct coupling are more costly to change, given the need to consider the potential impact of changing the coupled component on the dependent components (Simon, 1962). Hence we predict that component A would be more costly to change than a similar component with no coupling (e.g., as in Figure 1.1). Support for such a relationship is found in empirical studies of software, in which the components are source files or classes, and dependencies denote relationships between them (Chidamber and Kemerer, 1994).

Figure 1.3 depicts a more complex set of relationships between system components. Component A is directly coupled to B but *indirectly coupled* to C and D. In this system, changes may propagate between components that are not directly connected, via a "chain" of dependencies. While indirect coupling relationships are weaker than direct coupling relationships, they are not as visible to the system architect, hence more likely to produce unintended system behaviors. Indeed, empirical work has shown that changes to one component in an IT system can often create unexpected disruptions or defects in operation in distant parts of the system (Vakkuri, 2013; MacCormack and Sturtevant, 2016).

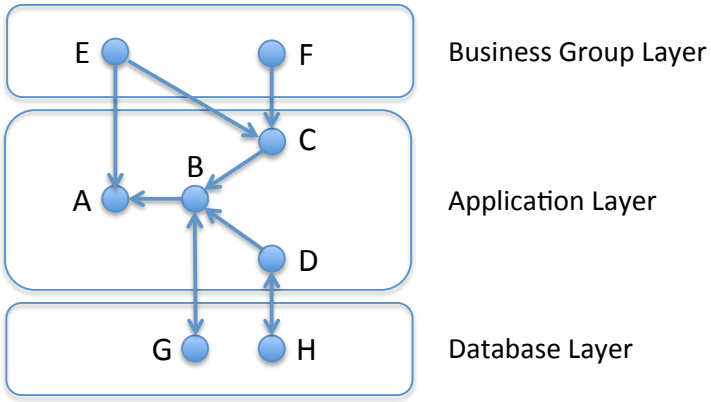
Figure 1.4 illustrates a third pattern of coupling between applications, called *cyclic coupling* (Whitney, et al, 2004; Sosa et al, 2013). In this system, A is coupled with B, B is coupled with C, and C is coupled with A. These components form a cyclic group – a group of

components that are mutually interdependent. In contrast to figure 1.3, there is no "hierarchy" or natural ordering of these components, such that one can be developed (or changed) before the others. Rather, components in cyclic groups must often be developed (or changed) concurrently, to ensure that they work together effectively. When cyclic groups are large, this presents a significant challenge, increasing the cost of change for components (Baldwin et al, 2014).

3.3 Types of Coupling between Software Applications and other Layers

Applications are the chief mechanism through which a firm's IT infrastructure supports the delivery of core business capabilities (Ramasubbu and Kemerer, 2015). However, the portfolio of software applications represents only one layer in a firm's IT architecture. Other important layers exist, including databases, database hosts, application servers, and business groups that use the capabilities this infrastructure provides. In addition to coupling relationships between applications, a firm's agility may be impacted by coupling relationships between applications and these other layers. In this study, we consider the layers immediately above and below the application layer. In particular, business groups use applications and applications read from/write to databases (see **Figure 2**). Modular systems theory suggests that applications with higher levels of coupling to these adjacent layers will be more costly and difficult to change.

Figure 2: Types of Coupling between Software Applications and other Layers



3.4 Measures of IT Agility in Technical Systems

Early work by Duncan (1995) and Byrd and Turner (2000) identified important constructs believed to underpin IT infrastructure flexibility, focusing on connectivity, compatibility and modularity. Many studies assume these constructs to be *proxies* for IT agility (i.e., measures of *output*) and seek to identify their antecedents (Schmidt and Buxmann, 2011; Joachim et al, 2013). In contrast, other work considers these constructs to be *features* of an IT architecture (i.e., measures of *input*) and explores their impact on performance (Kim et al, 2011; Tiwana and Konsynski, 2010). We share the view that constructs like modularity represent features of IT architecture, and are not direct proxies for IT agility. In our work therefore, we define explicit measures of IT agility, to test the assertion that architecture impacts this ability.

Our theory explores the impact of the coupling on individual applications; hence we define measures of IT agility at this level (i.e., and not for the firm as a whole). In particular, we focus on three types of change that applications experience, as a business evolves over time. First, we measure *the cost to maintain an application*, encompassing changes to fix errors in operation, and incremental updates to its functionality. Maintenance involves the smallest degree of change to an application, hence requires the lowest amount of IT agility. Second, we capture *the likelihood of an application being retired* (i.e., decommissioned) over a 4-year period. Retiring applications that are obsolete or no longer needed is a critical task in the modern firm, given a dynamic and changing marketplace (IBM, 2009). Indeed, a recent study of firm software portfolios by Aier et al (2009) found that > 40% of applications were retired over a 4-year period. This task involves not only the removal of functionality from the software portfolio, but also the removal of linkages between the application to be retired and those that remain. Hence retiring an application requires a higher degree of change (hence agility) than maintaining it.

For our third measure of agility, *we capture the commissioning of new applications for the firm's portfolio*. As firms replace obsolete technologies, build new capabilities, and enter new markets, adding new applications is a critical capability. This involves the development of new functionality, and the integration of this new functionality with existing applications and infrastructure. We argue that adding new applications represents the largest degree of change to a portfolio, requiring the highest level of IT agility. This is likely to be more difficult to the extent that new applications require high levels of coupling to other applications in the portfolio.

3.5 The Relationship between Application Coupling and IT Agility

We have identified different types of coupling that exist between the applications in a firm's software portfolio, and defined three measures of IT agility, which can be captured for these applications. Modular systems theory suggests that applications with higher levels of coupling will be more difficult and costly to change, hence measures of coupling will be negatively associated with these measures of IT agility. Hence we state our hypotheses below:

Hypothesis 1: Applications with higher levels of coupling will be *more costly to maintain,* on average, than applications with lower levels of coupling.

Hypothesis 2: Applications with higher levels of coupling will be *less likely to be retired*, on average, than applications with lower levels of coupling.

Hypothesis 3: New applications added to the portfolio will have *lower levels of coupling*, on average, than the applications in the portfolio at the start of the period.

We conduct empirical tests for the impact of all types of coupling defined above (i.e., direct, indirect, cyclical and between-layer coupling). We have no ex-ante predictions as to the relative strengths of these different types of coupling. Rather, we rely on our empirical tests to identify which of them have more explanatory power for each dimension of agility.

4. Research Methods

To develop measures of the coupling between applications, we use Design Structure Matrices (DSMs) a popular network-based method for analyzing technical systems (Steward, 1981; Eppinger et al., 1994; MacCormack et al., 2006; 2012; Sosa et al., 2007). A DSM highlights the structure of a system using a square matrix, in which rows and columns represent system elements, and dependencies between elements are captured in off-diagonal cells. DSMs capture the direction of dependencies between elements, and hence can discriminate between incoming and outgoing dependency relationships. Using a matrix to capture dependency relationships also facilitates the discovery of *indirect* and *cyclical* relationships between elements, which can be identified via well-known matrix operations.

Baldwin et al. (2014) show that DSMs can be used to understand the "hidden structure" in software systems, by capturing the level of direct, indirect and cyclic coupling between source files, and classifying files into categories based upon the results. Lagerström et al. (2013) and MacCormack et al (2016) show that this approach can be extended to study a firm's enterprise IT 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 infrastructure. In this paper, we build upon and extend this approach, by exploring how measures derived from the DSM of a firm's software portfolio architecture predict IT agility.

4.1 Measuring *Direct Coupling* in a DSM

A DSM is a way of representing a network. Rows and columns of the matrix denote nodes in the network; and off-diagonal entries indicate dependencies between nodes. In the analysis of software portfolio architecture, the rows, columns, and main diagonal elements of the DSM correspond to software applications. Linkages *between* applications are represented by off-

diagonal entries in the DSM (set to one) and indicate that a coupling relationship exists between two applications. As a matter of convention, usage proceeds from row to column in our DSMs. Hence reading down the column of an application reveals all applications that depend upon it, and reading across the row reveals all the applications it depends upon. As a matter of definition, all main diagonal elements are set to one (i.e., applications "depend upon" themselves).

The levels of "fan-in" and "fan-out" coupling for an application can be read directly from a DSM. Specifically, for the *i*th application in a portfolio, the level of direct fan-in coupling is found by summing entries in the *i*th column. The level of direct fan-out coupling is found by summing entries in the *i*th row. In general, these measures will be different, unless all the dependencies for a focal application are symmetric. If usage is symmetric (i.e., A uses B and B uses A), then off-diagonal entries in the DSM will be symmetric around the main diagonal.

4.2 Measuring *Indirect Coupling* in a DSM

Using a DSM, we can find the *indirect* dependencies between applications, which reflect the potential for changes to propagate in a system. To identify indirect relationships, we apply the procedure of transitive closure to the direct dependency DSM and set all positive entries equal to one. The result is the "visibility" matrix (Sharman and Yassine 2004; MacCormack et al., 2006). The visibility matrix captures all *indirect* dependencies between applications.[1] In a similar fashion to the direct dependency DSM, the level of indirect fan-in and fan-out coupling for an application is captured in the row and column sums of the visibility matrix.

The density of the visibility matrix, called *propagation cost*, measures the level of indirect coupling for the software portfolio as a whole. Intuitively, the greater the density of this matrix, the more ways there are for changes to propagate across applications, and thus the higher the potential cost of change. Large differences in propagation cost are observed across software

[1] Note by definition, the set of *indirect* coupling relationships also includes the *direct* relationships between elements.

systems of similar size and function (Baldwin et al, 2014). These differences are predicted, in part, by the way that development activities are organized (MacCormack et al, 2012). However, empirical evidence also suggests that refactoring efforts aimed at making software more modular can lower propagation cost substantially (MacCormack et al., 2006; Akakine, 2009).

4.3 Measuring *Cyclic Coupling* in a DSM

The visibility matrix can be used to identify "cyclic groups" of applications, each of which is directly or indirectly connected to the others. Mathematically, members of a cyclic group all have the same levels of indirect fan-in and fan-out coupling, given they are all connected directly or indirectly to each other. Hence we can identify cyclic groups in a system by sorting applications by these measures and grouping components with matching values (Baldwin et al., 2014). Prior work has shown the majority of software 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 (Baldwin et al, 2014). The components in such systems can be classified into four categories according to the levels of indirect coupling they exhibit, as compared to members of this cyclic group. We apply the same classification process to the applications in a firm's software portfolio.

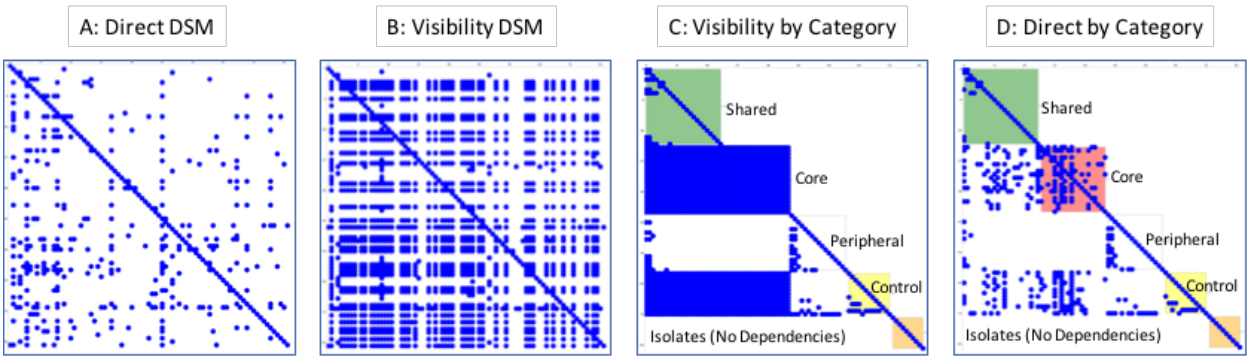
Core applications are members of the largest cyclic group, and have high levels of both fan-in and fan-out coupling. *Shared* applications have high levels of fan-in coupling (i.e., they are "used," directly or indirectly, by Core and other applications). *Control* applications have high levels of fan-out coupling, (i.e., they "use," directly or indirectly, Core and other applications). *Peripheral* applications have low levels of both fan-in and fan-out coupling. In a software portfolio, Shared, Core and Control applications are called "Main Flow" applications. Peripheral applications, being only loosely coupled to other applications, lie outside this main flow.

4.4 Capturing *Hierarchy* in a DSM

When used as a planning tool in a design process, a DSM indicates a possible sequence of design tasks, i.e., which components to design before which others (Steward, 1981; Eppinger et al, 1994). In general, it is desirable to place early design tasks at the top of a DSM, with later tasks below. That is, the first components to be designed should be those that other components depend upon. Reflecting this discussion, we place the most used applications (i.e., Shared applications) at the top of the DSM and the largest users of other applications (i.e., Control applications) at the bottom. The resulting DSM has a "lower diagonal form," with most dependencies below the diagonal, above diagonal entries indicating the presence of cyclic coupling (Sosa et al, 2013). This reveals the *hierarchy of relationships* between applications.

Figure 3 illustrates the steps in our analysis method for a simple system: (A) shows the direct dependency DSM; (B) shows the visibility matrix, which reveals all indirect dependencies; (C) shows the visibility matrix, sorted into the four indirect coupling categories; and (D) shows the direct dependency DSM, with components sorted by these four categories.[2]

Figure 3: The DSM Analysis Method applied to an Example System (Linux v0.01).



[2] In this example, we show a fifth category called "Isolates." These are components with *no* dependencies to any other components in the system. As such, they are not a part of the network graph. In prior work, these have been analyzed separately, or included in the Peripheral category, given that they have zero coupling with other components (MacCormack and Sturtevant, 2016).

5. Empirical Setting

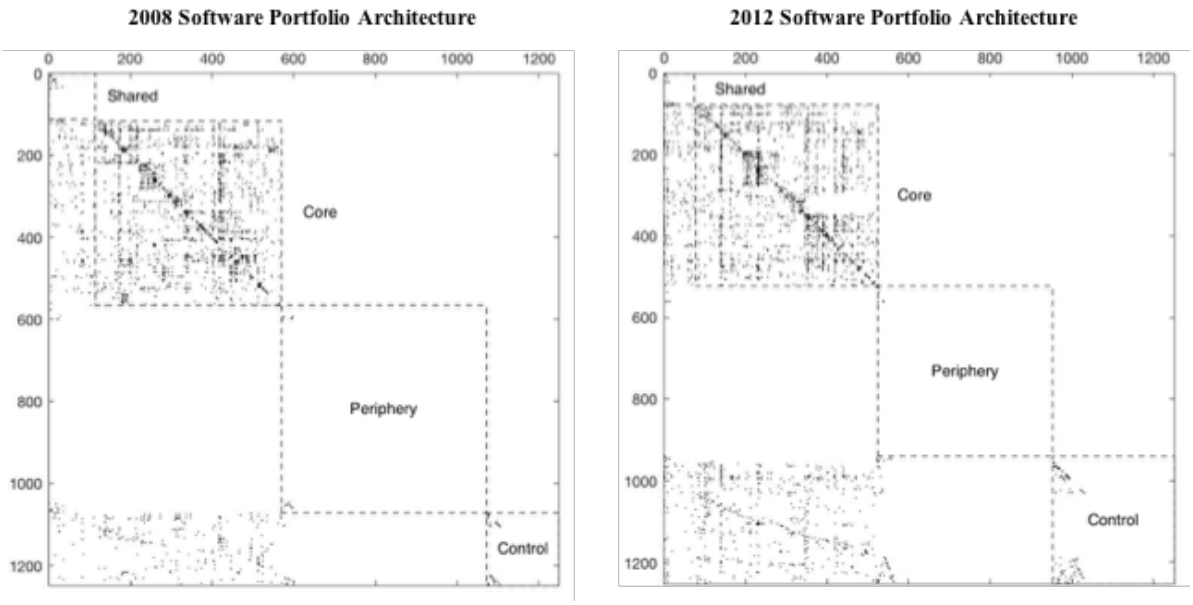
We test our hypotheses using a unique dataset from the software portfolio of a large European bank. The data was gathered as part of an initiative to develop a better understanding of how linkages between software applications affect application performance. Each quarter, the bank asks application owners to enter information in a database. Data is collected for each application on the departments that use it, the operating systems that it supports, the databases that it uses, and the dependencies that it has with other applications. In order to test our hypotheses, we captured data on active software applications for two time periods: 2008 and 2012. We were also provided with data on application maintenance costs for 2008.

Sample Data for 2008

The 2008 software portfolio consists of 1,558 active applications. Of these, 1,247 contained reliable data on application dependencies.[3] Thus, our sample consists of 1,247 applications and 3,482 dependencies. Using the methods described earlier, we identified all direct and indirect coupling relationships between applications in the portfolio, and classified applications into the four indirect coupling categories. We find the 2008 software portfolio has a large Core of 447 applications (36% of the portfolio) which are mutually interdependent. 120 applications (10%) are classified as Shared (i.e., they are used, directly or indirectly, by many applications). 175 applications (14%) are classified as Control (i.e., they use many other applications, directly or indirectly). Finally, 505 applications (41%) are classified as Peripheral (i.e., they have low levels of indirect fan-in and fan-out coupling to other applications). **Figure 4** shows the firm's software portfolio architecture in DSM form for 2008 and 2012.

[3] In 2008, the collection of data on the software portfolio was fairly new and consequently, data was not available for all applications. We were told, in general, that missing data was more likely for applications that were smaller and less important.

Figure 4: The Firm's Software Portfolio Architecture in 2008 and 2012



Sample Data for 2012

The 2012 software portfolio contains 1,251 applications and 3,969 dependencies. All applications contain sufficient data for analysis in this period, hence our sample represents the entire population. The analysis of the 2012 portfolio reveals a large Core of 441 applications, representing 35% of the system. 80 applications (6%) are classified as Shared, 298 applications (28%) are classified as Control and 432 applications (35%) are classified as Peripheral. Table 1 shows a comparison of the firm's applications grouped by category, in 2008 and 2012. The total number of applications is similar, and the number in each category is also consistent. However, this analysis hides significant changes in the application portfolio, as discussed below.

Table 1: Comparison of Applications by Category for 2008 and 2012

	2008		2012	
Category	**Number**	**%**	**Number**	**%**
Shared	120	9.6%	80	6.4%
Core	447	35.9%	441	35.3%
Control	174	14.0%	298	23.8%
Peripheral	505	40.5%	432	34.5%
TOTAL	1247	100.0%	1251	100.0%

5.1 Portfolio changes between 2008 and 2012

Between 2008 and 2012 there were substantial changes in the software portfolio. In particular, the firm went through a merger with another bank, and as a result, there was a substantial rationalization of the application portfolio. As one manager remarked:

> "*There where massive changes in the IT landscape, resulting from the decommissioning of redundant or outdated applications. Furthermore, a number of applications from [the acquired bank] were taken over. Finally, data had to be migrated between the two.*" – Senior Enterprise Architect.

Thus, during the years between 2008 and 2012, many software applications were retired, new applications were commissioned, existing applications were updated (and hence may have moved category), and data was collected for applications where none existed in 2008. Table 2 shows the movement of software applications across the portfolio from 2008 to 2012. It includes active applications with missing data in 2008, for which data became available in 2012.

Table 2: Movement of Applications across the Software Portfolio from 2008 to 2012

Category	2008	Application Retired	Application Added	Moved Category (Net)	New Data Available	2012 (Row Sum)
Shared	120	-53	14	-19	18	80
Core	447	-121	72	-8	51	441
Control	175	-87	110	+28	72	298
Peripheral	505	-469	227	-1	170	432
Missing Data	311	N/A	N/A	N/A	-311	N/A
TOTAL	1,558	-730	423	N/A	N/A	1251

5.2 Data on Maintenance Cost for Applications

Data on annual maintenance costs was available for 376 of the 1,247 applications for which we have data in 2008. For other applications, application owners either did not provide the cost data, did not possess the cost data, or could not identify the unique costs attributable to

an application (e.g., because data were aggregated across multiple applications). The applications for which maintenance cost data is available are not randomly distributed, but are biased towards more important applications. Hence we must control for this bias in our tests.

We control for the non-random nature of missing maintenance cost data by rebalancing our sample for testing hypothesis one, to ensure it has the same characteristics as the population. Table 4 presents data on how we do this. Consider, the sample of applications in 2008 for which we have data is 1,247, of which 505 (40.5%) are classified as Peripheral. But we only have cost data for 19 of these applications. Of the 376 applications for which we have cost data, only 5.1% are Peripheral, a far lower proportion than in the 2008 population overall. In order to create a sample for testing, we therefore oversample observations in underrepresented categories, to match the proportions of the 2008 population.[4] For example, we replicate the 19 observations of Peripheral applications, to produce a total of 266 applications in this category. Our final sample for testing hypothesis one contains 660 observations, as shown below.

Table 4: Constructing a Representative Sample for Testing Hypothesis One

Category	2008 Applications	% by Category	Apps with Cost Data	% by Category	With Re-Sampling	% by Category
Shared	120	9.6%	37	9.8%	74	11.2%
Core	447	35.9%	249	66.2%	249	37.7%
Control	174	14.0%	71	18.9%	71	10.8%
Peripheral	505	40.5%	19	5.1%	266	40.3%
TOTAL	1247	100.0%	376	100%	660	100%

5.3 Empirical Measures

Table 5 below describes our measures of IT agility, control measures that may impact measures of agility, and measures of both between-layer and between-application coupling.

[4] See http://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/.

Table 5: Measures used in the Study

Dependent Variables: Measures of IT Agility
Cost – The maintenance cost for an application, defined as the "cost that an application produces for maintaining it, i.e. fixing errors and making minor changes needed to keep the current state of requirement-fulfillment." (Mocker, 2009).
Retired – We capture whether an application was Retired (1) or Survived (0) between 2008 and 2012.
New – We capture data on new applications added to the portfolio between 2008 and 2012.
Control Variables
Age – measures the age of an application (the number of years since the first go-live date).
State – indicates if an application is in production (1) or still in development (0).
OS – indicates the number of operating systems supported by an application; two or more (1) or one (0).
Vendor – indicates whether an application is from an external vendor (1) or was developed in-house (0).
Independent Variables: Coupling Between Applications and other Layers
DBMS – number of database management systems application is linked to; one or more (1) or none (0).
Users – indicates the number of business departments that use an application.
Independent Variables: Coupling Between Applications
Direct FI – measures the number of applications that directly depend upon the focal application (fan-in coupling)
Direct FO – measures the number of applications that a focal application directly depends upon (fan-out coupling).
Shared – indicates whether an application is in the *Shared* category based upon its level of indirect coupling (1) or not (0).
Core – indicates whether an application is in the *Core* category based upon its level of indirect coupling (1) or not (0).
Control – indicates whether an application is in the *Control* category based upon its level of indirect coupling (1) or not (0).
Peripheral – indicates whether an application is in the *Peripheral* category based upon its level of indirect coupling (1) or not (0).
Main Flow – indicates whether an application is in the *Shared, Core* or *Control* categories (1) or in the Peripheral category (0).

5.4 Descriptive statistics

In Table 6, we provide descriptive statistics for the samples used to test hypotheses one and two. The samples are different for each test, given the need to rebalance the sample to reflect biases in the availability of maintenance cost data. (Note that the test for hypothesis three involves a comparison of the coupling characteristics for new applications in 2012 versus the set of all applications that exist in 2008 – the sample for the latter is the same as Hypothesis 2). Correlation tables for these two samples are provided in the Appendices.

Table 6: Descriptive Statistics for Hypotheses 1 and 2

	A: Sample for Hypothesis 1				B: Sample for Hypothesis 2			
	Min	Max	Mean	St Dev	Min	Max	Mean	St.Dev
Cost	0	5,776.50	314.66	569.60	-	-	-	-
Retired	-	-	-	-	0	1	0.56	0.50
Age	1	29	7.13	4.64	0	31	8.73	5.03
State	0	1	0.99	0.09	0	1	0.98	0.15
# OS	0	1	0.08	0.27	0	1	0.06	0.23
Vendor	0	1	0.40	0.49	0	1	0.52	0.50
# DBMS	0	1	0.60	0.49	0	1	0.45	0.50
# Users	1	26	4.33	6.17	1	30	4.68	6.69
Direct FI	0	85	3.43	6.46	0	85	3.17	6.97
Direct FO	0	35	3.62	5.92	0	79	3.25	6.36
Main Flow	0	1	0.60	0.49	0	1	0.63	0.48
Shared	0	1	0.11	0.32	0	1	0.10	0.30
Core	0	1	0.38	0.48	0	1	0.40	0.49
Control	0	1	0.11	0.31	0	1	0.14	0.34
	n=660				*n=957*[5]			

First, we note the maintenance cost data is skewed; hence we use a log transformation for this variable in statistical tests. Second, the rate at which applications are retired between 2008 and 2012 is 56%. This mirrors other empirical work in this area that demonstrates a high turnover in software portfolios (Aier et al, 2009). The average age of applications is 8.7 years.[6] Age is also skewed; hence we use a log transformation for this variable in statistical tests. Almost all applications are in production (98%) and support only one operating system (94%). Vendor provided applications constitute 52% of the population and 45% of applications are linked to at least one database. Finally, there are 4.7 users (i.e., departments) on average per application. This variable is also skewed; hence we use a log transformation in statistical tests.

[5] Control variable data was not available for all 1257 applications. Hence our sample for H2 is n=957.
[6] We report data here for the sample used to test hypothesis 2, given this represents the full set of applications (as opposed to the sample for hypothesis 1, which contains oversampled data, as noted earlier).

6. Empirical Results

6.1 Hypothesis 1: The Relationship between Coupling and Maintenance Cost

Table 7 presents a series of models predicting the maintenance cost for each application, using control variables and the predictor variables described above. Note that we use a log transformation for the dependent variable given maintenance cost is skewed.

Table 7: Models Predicting the Cost of Application Maintenance

Ln (Cost)	Model1	Model2	Model 3	Model 4	Model 5	Model 6
Constant	0.341	-0.992	-0.765	-1.664	-1.57	-1.621
Ln (Age)	0.643***	0.427*	0.142	0.186	0.188	0.386†
State	2.777*	3.132**	3.32**	3.704**	3.624**	3.277**
# OS	1.059**	0.702†	0.685†	0.57	0.588	0.755†
Vendor	-1.348***	-0.656**	-0.497*	-0.537*	-0.553*	-0.212
# DBMS		1.452***	1.122***	0.87***	0.854***	0.707**
Ln (# Users)		0.266*	0.212*	0.268*	0.264*	0.219*
Ln (Direct FI)			0.212†			
Ln (Direct FO)			0.289*			
Main Flow (MF)				1.409***		1.307***
Shared					1.524***	
Core					1.417***	
Control					1.265***	
MF x Ln DFI						0.162
Periph x Ln DFI						-0.233
Periph x Ln DFO						1.691***
Adj. R-square	0.085	0.135	0.153	0.177	0.175	0.195
F-statistic	16.34***	18.15***	15.93***	21.25***	16.53***	17.01***
n=660; † p<0.1, * p<0.05, ** p<0.01, and ***p<0.001						

Model 1 includes only control variables, all of which are significant. Older applications, applications in production, applications that support multiple operating systems, and applications developed in-house cost more to maintain. In total, these variables explain 8.5% of the variance in maintenance cost. In model 2, we add between-layer coupling variables, both of which are significant. Applications used by more business departments and connected to a database

management system cost more to maintain. In total, these coupling variables increase the variation explained to 13.5%. Models 3-6 explore the predictive power of various measures of between-application coupling. Model 3 includes measures of direct fan-in and fan-out coupling. We use a log transformation for these variables given they are skewed. Only one of the variables is significant. The R-squared for this model increases from 13.5% to 15.3%.[7] In model 4, we remove direct coupling variables, and instead include the main-flow variable – which indicates whether an application is in the Shared, Core, or Control category. This variable is significant, and increases the model R-squared from 13.5% to 17.7%, as compared to model 2.

In model 5, we split main-flow applications into its three component categories – Shared, Core and Control. All three are significant. However, the model fit does not improve over model 4, and the coefficients are not statistically different from each other. We cannot include measures of direct coupling in models 4 or 5, given the high correlations between direct and indirect coupling (see the Appendices). We note however, that measures of indirect coupling have a higher correlation with cost than measures of direct coupling, and a greater level of significance in our models. *We conclude that indirect coupling measures have more power than direct coupling measures in explaining cost – our first dimension of IT agility*.

In model 6, we use interaction terms to evaluate the impact of direct coupling measures *within* different indirect coupling categories. In particular, we interact main-flow and peripheral variables, with the level of direct coupling. For peripheral applications, we find the measure of direct fan-out coupling adds significant explanatory power to our model. In total, our final model explains 19.5% of the variation in cost. Control variables explain 8.5%, between-layer coupling variables explain 5%, and between-application coupling variables explain 6%.

[7] We note that direct fan-in and direct fan-out coupling are strongly correlated (see the Appendices).

6.1.1 Exploring the Impact of Indirect Coupling on Application Maintenance Cost

To explore the dynamics of how indirect coupling categories impact cost we further analyzed their relationship with the outcome. Table 8 presents data on the mean, standard deviation, and skewness of maintenance cost by category. (We present here the raw data, not the transformed data used in our statistical models). We observe that cyclically coupled Core applications have the highest cost, followed by Shared, Control, and then Peripheral applications. Core applications also experience higher variations in cost than applications in other categories. The implication is that cyclically coupled applications are harder to predict (and hence to budget for) with respect to maintenance cost and more likely to be outliers on this dimension of agility.

Table 8: Differences in Maintenance Cost by Indirect Coupling Category

	Maintenance Cost		
	Mean	**St. Dev**	**Skewness**
Shared	344.86	452.48	2.21
Core	497.54	753.80	3.55
Control	255.37	468.93	3.86
Peripheral	143.41	293.11	3.11

In sum, the evidence we present suggests that hypothesis one is supported. Applications with higher coupling cost more to maintain. We find support for the predictive power of both between-layer coupling variables (i.e., the number of departments and database connections) as well as between-application coupling variables. We show that indirect coupling categories are better predictors of maintenance costs than measures of direct coupling. While our statistical models cannot differentiate between the impact of different types of indirect coupling (i.e., Shared, Core and Control), we show that cyclically coupled Core applications have the highest maintenance costs, and experience the highest variation in these costs.

6.2 Hypothesis 2: The Relationship between Coupling and Application Retirement

Table 9 presents a series of logistic regression models predicting the probability of an application being retired between 2008 and 2012.

Table 9: Logit Models Predicting Application Retirement

Retire (1-0)	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6
Constant	-0.423	0.930†	0.141	2.481***	2.187***	0.968
Ln (Age)	0.395***	0.481***	0.776***	0.569***	0.664***	0.652***
State	-0.973†	-1.237*	-0.893	-0.968	-0.866	-0.631
# OS	-0.938**	-0.524	-0.563	-0.454	-0.533	-0.519
Vendor	1.812***	1.137***	0.625***	0.497**	0.438*	0.322†
# DBMS		-1.679***	-0.990***	-1.178***	-1.025***	-0.913***
Ln (# Users)		-0.150†	0.029	-0.080	-0.043	0.021
Ln (Direct FI)			-0.638***			
Ln (Direct FO)			-0.359***			
Main Flow (MF)				-2.664***		-1.491***
Shared					-2.536***	
Core					-3.063***	
Control					-2.157***	
MF x Ln DFI						-0.437***
Periph. x Ln DFO						-1.94**
Chi-square	189.56***	297.07***	411.23***	446.88***	462.31***	479.71***
Cox&Snell R^2	0.180	0.267	0.349	0.373	0.383	0.394
Nagelkerke R^2	0.241	0.358	0.468	0.500	0.514	0.528
n=975; † p<0.1, * p<0.05, ** p<0.01, and ***p<0.001						

Model 1 includes only control variables, three of which are significant. Older applications and applications from vendors are more likely to be retired. Applications that support more operating systems are less likely to be retired. In total, these variables explain 24.1% more variation than the null model (i.e., a model with no predictors).[8] In model 2, we add between-layer coupling variables; one is strongly significant (p<0.1%), the other only marginally so (p<10%). Applications that make use of database management systems are less likely to be

[8] We use Nagelkerke's pseudo R-squared statistic to compare models. This mirrors the Cox & Snell statistic, but is adjusted so that a perfectly fitted model would yield a 100% value (the Cox & Snell statistic cannot take a value of 100%).

retired. Applications with more users *may* be less likely to be retired. In total, these variables increase the variation explained to 35.8%. Model 3 includes measures of direct coupling. (We use a log transformation for these variables as before.) Both variables are strongly significant, with the model showing an increase in the variation explained over the null model to 46.8%. In model 4, we remove direct coupling measures, and include the variable for main-flow applications. This variable is strongly significant, and increases the model fit to 50.0%.

In model 5, we split main-flow applications into its three component categories – Shared, Core and Control. All three are significant. In addition, the model fit improves over model 4, and the coefficients on the variables are statistically different from each other. Specifically, Core applications are the least likely to be retired, and Control applications are the most likely to be retired (but still significantly less likely than Peripheral applications – the baseline). We cannot include measures of direct coupling in models 4 or 5, given the high correlations between direct and indirect coupling (see the Appendices). We note however, that measures of indirect coupling have a higher correlation with retirement than measures of direct coupling, and a greater level of significance in our models. *We conclude that indirect coupling measures have more power than direct coupling measures in explaining retirement – the second dimension of IT agility*.

In model 6, we use interaction terms to evaluate the impact of direct coupling measures *within* different indirect coupling categories. In particular, we interact main-flow and peripheral variables, with the level of direct coupling. For main-flow applications, we find the measure of direct fan-in coupling adds significant power to our model. For peripheral applications, we find the measure of direct fan-out coupling adds significant power to our model. In both cases, higher levels of coupling within a category are associated with a lower likelihood of being retired. In total, our final model explains 52.8% more of the variation in retirement than the null model.

Control variables explain 24.1% of the improvement in model fit, between-layer coupling variables explain 11.7% and between-application coupling variables explain 17%.

6.2.1 Exploring the Impact of Coupling on Application Retirement

To explore the dynamics of how indirect coupling categories impact retirement we further analyzed their relationship with the outcome. Table 10 presents data on the number of applications by category in 2008 and the number retired between 2008 and 2012, expressed as an absolute number, and as a percentage. We observe first, that Peripheral applications have a large probability of being retired. Of the 505 peripheral applications in 2008, almost 93% are retired by 2012 (compared to 58.5% for the sample overall). This highlights the huge turnover for applications with little or no indirect coupling in the software portfolio. Second, by contrast, the percentage of cyclically coupled Core applications that are retired between 2008 and 2012 is only 27%. In sum, Peripheral applications are retired at 3X the rate of Core applications. Finally, we note the rate at which Shared and Control applications are retired over this time period is similar, at 44.2% and 49.7% respectively.

Table 10: Differences in Application Retirement by Category

	Applications in 2008	Retired by 2012	Percentage Retired
Shared	120	53	44.2%
Core	447	121	27.1%
Control	175	87	49.7%
Peripheral	505	469	92.9%
TOTAL	1247	730	58.5%

In sum, the evidence we present suggests that hypothesis two is supported. Applications with higher coupling are less likely to be retired. We find support for the predictive power of both between-layer coupling variables (i.e., the number of departments and database

connections) as well as between-application coupling variables. We show that indirect coupling categories are better predictors of retirement than measures of direct coupling. And our statistical models show that Core applications are much less likely to be retired than other categories. The descriptive analysis highlights this result, while showing the extremely high rate at which Peripheral applications are retired over the same period.

6.3 Hypothesis 3: The Relationship between Coupling and New Applications

Our third hypothesis asserts that new applications added to the software portfolio between 2008 and 2012 will have lower levels of coupling compared to the set of all applications in the portfolio at the start of the period. To test this assertion, we compare the distribution of applications by category for 2008, to that of new applications added to the portfolio thereafter. All else being equal, one would predict that new applications should mirror the distribution of all 2008 applications, in terms of their coupling characteristics (the "null" hypothesis).

Table 11 shows the distribution of applications by indirect coupling category for the 2008 portfolio, as well as for new applications added between 2008 and 2012.[9] For example, in 2008, there were 120 Shared applications, representing 9.6% of the portfolio. Of the 423 new applications added between 2008 and 2012 however, only 14 (i.e., 3.3% of them) were Shared. If new applications were added in a way that mirrors the characteristics of all 2008 applications, we would expect 9.6% of the 423 new applications (i.e., 40 applications) to be Shared. The actual outcome is significantly below what is expected. The ratio of the actual to expected outcome is 0.34. This ratio reflects the degree to which the number of new applications either falls short of (<1) or exceeds (>1) the expected number in each category.

[9] Note in this analysis, we count only *new* applications added to the software portfolio between 2008 and 2012. We do not include the 311 active applications that existed in 2008, but which were missing data and hence were not assigned a coupling category.

Table 11: Number of Applications by Category for 2008 and for New Applications

	Applications in the 2008 Portfolio		New Applications added 2008-2012		Ratio of Actual to Expected
Shared	120	9.62%	14	3.30%	0.34
Core	447	35.85%	72	17.02%	0.47
Control	175	14.03%	110	26.00%	1.85
Peripheral	505	40.50%	227	53.66%	1.32
TOTAL	1247	100%	423	100.00%	1.00
Chi-Square (df=3)	90.723***				
*** p < 0.001					

We find that new applications occur less frequently than expected in the Shared and Core categories. By contrast, new applications occur at a greater rate than expected in the Control and Peripheral categories. The results make intuitive sense. Adding new applications that have little or no coupling relationships with other applications (i.e., Peripheral applications) should be relatively easy. Furthermore, adding new applications that use or "depend upon" legacy applications should be easier than adding new applications that are used by or "depended upon" by legacy applications. In essence, new applications can take advantage of existing functionality provided by legacy applications, but the reverse is more difficult to achieve.

To test whether the differences reported above are significant, we run a Chi-Square test of independence between the distribution of applications across categories for 2008 and for all new applications. The test statistic demonstrates that the distribution across categories is statistically different (i.e., the Chi-Square statistic exceeds a threshold value of 33.94). We conclude that hypothesis three is supported. New applications have a significantly lower number of applications in high indirect coupling categories. In particular, they are more likely to be Peripheral or Control applications, and less likely to be Shared or Core applications.

7. Discussion

The distinctive contribution of this paper is in developing better theory about the relationship between a firm's software portfolio architecture and its level of IT agility. Specifically, we find a strong link between the level of coupling of applications in the portfolio, and the degree to which these applications can be changed. Applications with higher levels of coupling are more costly to maintain, less likely to be retired and less likely to be commissioned.

A unique feature of our work is that it explores *different types of coupling* that impact applications, and hence can potentially affect IT agility. Specifically, we examine coupling relationships within the application layer, as well as between this layer and others in the IT architecture. With respect to the former, we find that indirect coupling relationships have a stronger association with IT agility than direct coupling relationships. With respect to the latter, we find that the number of users (i.e., business groups) for an application, and the number of databases to which it is connected, are also strong predictors of the cost and likelihood of change. We note that the best models predicting maintenance costs and application retirement include measures of *all* types of coupling: direct, indirect, cyclic and between-layer coupling.

In order to highlight the power of the different measures of coupling, we conduct a decomposition of variance analysis for hypotheses one and two. Table 12 shows the amount of explained variance for each outcome that is attributable to control variables, between-layer coupling variables, and between-application coupling variables. We break the latter into three; first showing the variation explained by direct coupling measures, next showing the variation explained by indirect coupling measures, and finally showing the variation explained by all measures combined (i.e., models which include interaction terms).

Table 12: Decomposition of the Variance Explained by Type of Variable

	Maintenance Cost H1			Application Retirement H2		
Control Variables	8.5%	8.5%	8.5%	24.1%	24.1%	24.1%
Between Layer Coupling	5.0%	5.0%	5.0%	11.7%	11.7%	11.7%
Direct Coupling	1.8%			11.0%		
Indirect Coupling		4.2%[10]			14.6%[11]	
Direct and Indirect			6.0%			16.0%
TOTAL VARIANCE	15.3%	17.7%	19.5%	46.8%	50.0%	52.8%

Several observations are apparent from this analysis. First, while control variables explain a sizeable amount of the variance in our models, measures of coupling explain more. Second, while both between-layer and between-application measures of coupling are significant, the latter have more power in predicting IT agility. This result reveals a paradox confronting IT managers as they direct efforts to enhance IT agility. While their focus is often on better structuring the relationship between applications and other layers in the IT architecture (e.g., databases and infrastructure), our results suggest their attention is better directed elsewhere. In particular, they must pay greater attention to the application portfolio itself, and specifically, the patterns of coupling that exist *between* the components of this portfolio.

Returning to the analysis of variance, we find measures of indirect coupling have more power than measures of direct coupling. This mirrors the results of similar studies looking at the impact of design decisions within software systems (MacCormack and Sturtevant, 2016). Direct relationships between components are more easily visible to a system architect, hence can be explicitly managed. They may not be problematic if constrained to a small group of components. Indirect relationships however, bring the potential for changes to propagate from one component to another via chains of dependencies. These chains are not easily visible by

[10] The baseline for this data is model 4 in Table 7, which includes main-flow as the predictor variable. Breaking main-flow into its three constituent components, as is done in model 5, yielded a *decrease* in the variance explained.
[11] The baseline for this data is model 5 in Table 9, which includes the variables indirectly coupled, cyclically coupled and indirectly dependent as predictors. This model explains more variance than a model that just includes main-flow as a predictor.

inspection of an application's nearest neighbors. They represent "hidden structure" that can only be revealed by an analysis of indirect pathways in a system (Baldwin et al, 2014).[12]

Of the different types of indirect coupling present in the portfolio, our work suggests that *cyclically coupled Core applications present the toughest challenges to managers*. These applications have the highest average cost, and the highest variation in cost. They are also less likely to be retired and less likely to be commissioned, suggesting it is both costly and difficult to perform these tasks. In the firm we studied, 35% of applications were cyclically coupled in 2008 and 2012, meaning 440 applications were mutually interdependent in both periods. Making changes to a set of Core applications of this size would be a complicated endeavor, given any change to a single application could potentially propagate to affect many others.

For the academy, our work contributes to the stream of research exploring the role of IT Architecture in a dynamic and digitally-connected world (Sambamurthy and Zmud, 2000; Samburmathy at al, 2003; Yoo et al., 2010; Tanriverdi et al, 2010; Tiwana et al, 2010). In particular, we build upon and extend existing empirical contributions, which have examined the role of modularity and loose-coupling in IT systems (e.g., Tiwana and Konsynski, 2010; Kim et al, 2011), by showing the precise mechanisms through which coupling impacts performance. Our distinct contribution is to conceptualize and operationalize different forms of coupling that impact the evolution of a firm's IT systems, and to examine their association with multiple dimensions of IT agility. Importantly, we achieve these objectives using *absolute* measures of a firm's software portfolio architecture, in contrast to the perceptual measures used in prior studies. And we do this at a more granular level of analysis than is the norm in prior studies (i.e., we focus on individual software applications instead of a firm's entire infrastructure).

[12] Note, the best model in each case includes both types of measures. In essence, measures of direct coupling help to explain the variations in each outcome that remain *within* each of the indirect coupling categories.

Our study has distinct implications for managers. First of all, our methodology provides a road map for measuring the *real* software portfolio architecture that a firm possesses, as opposed to the high level conceptual representations often found in documents depicting a firm's IT systems. The insights that this approach generates should prove useful in several ways, including i) helping to plan the allocation of resources to different applications, based upon predictions of the relative ease/difficulty of change; and ii) monitoring the evolution of the software portfolio over time, as new applications and/or dependencies are introduced. However, we also believe that the specific results we find in this study can help managers better direct their efforts to improve software portfolio architecture, and hence to enhance their digital agility.

In particular, we have shown that dependencies between applications are costly, while at the same time, they decrease a firm's ability to make changes to these applications. In developing new systems, managers must therefore closely monitor the creation of dependencies, to ensure that, i) they are necessary from the perspective of system performance, and ii) the patterns of indirect coupling they create are understood in terms of the potential to propagate. Moreover, our results suggest that managers should limit the number of cyclically-coupled applications in a software portfolio, to achieve a "Core" that is as small as possible. Of course, there are limits to what can be achieved in this respect, given the well-understood trade-offs between a system's performance, and the degree to which its components are loosely-coupled (Ulrich, 1995). We note however, that prior work exploring this phenomenon in software systems has found major differences in Core size, even for systems of similar size and function (MacCormack et al, 2012). This implies that the designers of a firm's software portfolio have considerable freedom to "manage" the level of coupling that exists between applications.

For a potent example of the power of decoupling in a software portfolio, it is informative to look at Amazon. In 2001, the firm had a monolithic architecture, with many complex pieces of software combined in a single system. As CTO Werner Vogels noted, "It couldn't evolve anymore. The parts that needed to scale independently were tied into sharing resources with other unknown code paths. There was no isolation" (Vogels, 2006). Subsequently, the firm undertook an extensive restructuring of the software portfolio, transforming it into a distributed, decentralized, Service Oriented Architecture. This facilitated a level of isolation that allowed the firm to build software components more rapidly and independently. By 2006, Vogels stated, this transformation had "become one of our main strategic advantages" (Vogels, 2006).

Ironically however, in this era of big data, the lack of granular data may be the largest barrier to the systematic investigation of software portfolio architecture. In order to analyze architecture, organizations must capture rigorous data on the coupling between applications in the portfolio, and the way that this evolves over time. To use this data for improvement, they must also capture data on the cost of application change in a systematic fashion. In most organizations with which we have worked, this type of data does not exist. In some, like the firm we worked with, efforts have been made to collect this data manually. However, there are many challenges associated with this approach, including the lack of proper incentives to provide accurate and/or timely information. As a consequence, we believe it is likely that many firms do not understand their "real" software portfolio architecture. Instead, they place their faith in simplistic, idealized representations of IT architecture that likely obscure its most important characteristics, with respect to enhancing their digital agility.

Our work opens up the potential for further research to explore the relationship between software portfolio architecture and IT agility. For example, in this study, features of our dataset

made it difficult to disentangle the effects of different categories of coupling on some of our measures of IT agility. However, in other settings, this will not always be true. We believe it important to study these mechanisms more deeply, to fully understand the relationships that different types of coupling have, with each other and with different types of change. Specifically, given the heterogeneity across firms' IT systems, and the mix of new and legacy software that firms possess, we believe there is promise in longitudinal studies that examine these issues. If firms discover that they possess a software portfolio architecture with undesirable characteristics, is there evidence that effective remedial action can be taken? Research that focuses on the impact of managerial interventions, like that taken by Amazon, would be invaluable, to assess whether the projected benefits from such actions can be realized in practice. And if not, what might be the mediating factors that determine the value released?

Finally, we note our study is subject to a number of limitations that must be considered when assessing the generalizability of results. In particular, while our unit of analysis is the software application, for which we have over 2,000 observations, the data used to test our propositions comes from a single firm. Additional work is needed to validate that our results hold for other firms and industries. We may find that different types of firm, or different managerial processes within firms, influence these results. Furthermore, studies across different organizations might reveal how measures of IT architecture impact *firm-level* performance. This topic is promising, given the prior literature that argues for a link between IT architecture and firm-level agility. Our hope is that this paper, and the methods that it describes, will allow us to answer such questions, with a rigorous approach that can be replicated across studies.

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. Buckl, S. Franke, U. Gleichauf, B. Johnson, P. Narman, P. Schweda, C. Ullberg, J. A Survival Analysis of Application Life Spans based on Enterprise Architecture Models, *Proc. of the 3rd Intl. Wkshp on Enterprise Modeling and Information Systems Architecture* (EMISA).

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.

Andreessen, M. 2011. Why Software is Eating the World, *Wall Street Journal*, August 20 2011.

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.

Berente, N. and Yoo, Y., 2012. Institutional contradictions and loose coupling: Postimplementation of NASA's enterprise information system. *Information Systems Research*, *23*(2), pp.376-396.

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

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.

Cohen, W. M., and D. A. Levinthal. "Absorptive Capacity: A New Perspective on Learning and Innovation." *Administrative Science Quarterly*, vol. 35, no. 1, 1990, pp. 128–152.

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.

Hanseth, O. and Lyytinen, K., 2010. Design theory for dynamic complexity in information infrastructures: the case of building internet. *Journal of Information Technology*, *25*(1), pp.1-19.

IBM. 2009. Application Ccnsolidation and retirement projects: Strategies that deliver ROI. IBM Software, White Paper.

Joachim, N. Beimborn, D. and Weitzel, T. 2013. The influence of SOA governance mechanisms on IT flexibility and service reuse, *The Journal of Strategic Information Systems*, 22 (1), 86-101.

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

Kim, G., Shin, B., Kim, K.K., and Lee, H.G. 2011. IT capabilities, process-oriented dynamic capabilities, and firm financial performance. *Journal of the Association for Information Systems* 12, 7.

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.

Langlois, R.N. 2002. Modularity in technology and organization. *Journal of economic behavior & organization* 49, 1, 19-37.

Lawrence, P.R., and Lorsch, J.W. 1967. Differentiation and integration in complex organizations. *Administrative science quarterly*.

Liu, H.,Ke, W., Wei, K.K., and Hua, Z. 2013. The impact of IT capabilities on firm performance: The mediating roles of absorptive capacity and supply chain agility. *Decision Support Systems* 54, 3, 1452-1462.

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., 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.

MacCormack, A., and Sturtevant, D. Technical Debt and System Architecture: The Impact of Coupling on Defect-Related Activity. *Journal of Systems and Software*. Vol. 120, p170-182, 2016.

MacCormack, A. Lagerstrom, R. Baldwin, C., and Dreyfus, D 2016. Building the Agile Enterprise: IT Architecture, Modularity and the Cost of IT Change. *Harvard Business School Working Paper 15:060.*

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

Mocker, M. 2009. What is Complex about 273 Applications? Untangling Application Architecture Complexity in a case of European Investment Banking, *Proc. of the 42nd Hawaii Intl. Conf. on System Sciences*.

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

Parker, G. Van Alstyne, M. and Choudary, S.P. *Platform Revolution*, W.W. Norton & Co., NY 2016.

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

Ramasubbu, N., Kemerer, C.K., Technical Debt and the Reliability of Enterprise Software Systems: A Competing Risks Analysis, *Management Science* 62 (5), 1487-1510 2015

Ross, J.W. 2003. Creating a strategic IT architecture competency: Learning in stages. MIT Sloan School of Management Working Paper No. 4314-03.

Ross, J.W., and Westerman, G. 2004. Preparing for utility computing: The role of IT architecture and relationship management. *IBM systems journal*, 43, 1, 5-19.

Salmela, H., Tapanainen, T., Baiyere, A., Hallanoro, M., and Galliers, R. 2015. IS Agility Research: An Assessment and Future Directions. *ECIS 2015 Completed Research Papers*. Paper 155.

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.

Sanchez, R.A., Mahoney, J.T. 1996. Modularity, flexibility and knowledge management in product and organizational design. *Strategic Management Journal* 17, 63–76.

Schilling, M.A. 2000. Toward a general systems theory and its application to interfirm product modularity. Academy of Management Review 25 (2), 312–334.

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. and A. Yassine 2004 Characterizing Complex Product Architectures, *Sys. Engineering Journal*, 7(1).

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

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.

Tanriverdi, H. A. Rai and N Venkatraman. 2010. Reframing the dominant quests of information systems strategy research for complex adaptive business systems, *Information Systems Research*, Vol 21 No 4, 2010.

Tilson, D., Lyytinen, K. and Sørensen, C., 2010. Research commentary-digital infrastructures: the missing IS research agenda. *Information systems research*, *21*(4), pp.748-759.

Tiwana, A. and B. Konsynski. 2010. Complementarities between organizational IT architecture and governance structure. Information Systems Research, Vol 21, No 2, 2010.

Ulrich, K. 1995. The role of product architecture in the manufacturing firm. *Research Policy* 24, 419–440.

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

Vogels, W. A conversation with Werner Vogels. Interview by J. Gray. *ACM Queue*, May 2006.

Weick, K. E. 2001. *Making sense of the organization*. Malden, MA: Blackwell Publishers.

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.

Whitney, D.E. (Chair) and the ESD Architecture Committee. 2004. The Influence of Architecture in engineering Systems. Engineering Systems Monograph,

Woodard, J.C. Ramasubbu, N and Tschang, F.T. and Sambamurthy, V. Design Capital and Design Moves: The Logic of Digital Business Strategy, *MISQ*. 37, (2), 537-564. 2013

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: Correlation Table for Data used to Predict Maintenance Cost (Hypothesis 1)

	Mcost	*Age*	State	OS	Vendor	DBMS	*Users*	MainFlow	Core	Shared	Control	*Dir FI*	*Dir FO*
Mcost	1												
Age	0.13*	1											
State	0.09*	0.17*	1										
OS	0.08*	-0.26*	-0.10*	1									
Vendor	-0.24*	0.03	0.04	-0.09*	1								
DBMS	0.32*	0.09*	-0.07	0.17*	-0.49*	1							
Users	0.10*	0.18*	0.08*	-0.09*	-0.01	-0.03	1						
MainFlow	0.35*	0.23**	-0.07	0.08*	-0.29*	0.49*	0.02	1					
Core	0.27*	0.33*	-0.04	0.04	-0.29*	0.40*	0.09*	**0.64***	1				
Shared	0.09*	-0.06	0.03	0.03	0.06	0.09*	-0.01	0.29*	-0.28*	1			
Control	0.05	-0.09*	-0.14*	0.12*	-0.08*	0.05	-0.10*	0.29*	-0.27*	-0.12*	1		
Dir FI	0.27*	0.47*	0.05	-0.07	-0.18*	0.37*	0.17*	0.58*	0.70*	0.09*	-0.27*	1	
Dir FO	0.29*	0.26*	-0.02	0.02	-0.36*	0.40*	0.12*	0.52*	0.71*	-0.33*	0.04	**0.61***	1

n=660, * p<0.05, *italic*=ln()

Appendix B: Correlation Table for Data used to Predict Application Decommissioning (Hypothesis 2)

	Retire	*Age*	State	OS	Vendor	DBMS	*Users*	MainFlow	Core	Shared	Control	*Dir FI*	*Dir FO*
Retire	1												
Age	0.09*	1											
State	-0.01	0.34*	1										
OS	-0.12*	-0.18*	-0.08*	1									
Vendor	0.41*	-0.01	0.04	-0.04	1								
DBMS	-0.48*	-0.02	-0.06	0.14*	-0.51*	1							
Users	-0.04	0.08*	0.04	-0.06	-0.03	-0.02	1						
MainFlow	-0.58*	-0.06	0.02	0.09*	-0.53*	0.51*	0.09*	1					
Core	-0.48*	0.14*	0.06	0.01	-0.45*	0.49*	0.12*	**0.62***	1				
Shared	-0.08*	-0.06	0.03	0.01	-0.03	-0.09	0.09	0.25*	-0.26*	1			
Control	-0.06	-0.15*	-0.08*	0.01*	-0.08*	0.03	-0.06*	0.30*	-0.32*	-0.13*	1		
Dir FI	-0.51*	0.19*	0.09*	0.07	-0.46*	0.51*	0.20*	**0.62***	**0.76***	0.09*	-0.28*	1	
Dir FO	-0.50*	0.10*	0.04	0.06	-0.49*	0.53*	0.14*	**0.66***	**0.74***	-0.23*	0.06	**0.74***	1

n=957, * p<0.05, italic=ln()