Designing an Agile Software Portfolio Architecture: The Impact of Coupling on Performance

Citation	MacCormack, Alan, Robert Lagerstrom, Martin Mocker, and Carliss Y. Baldwin. "Designing an Agile Software Portfolio Architecture: The Impact of Coupling on Performance." Harvard Business School Working Paper, No. 17-105, May 2017.
Citable link	http://nrs.harvard.edu/urn-3:HUL.InstRepos:33110111
Terms of Use	This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Open Access Policy Articles, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#OAP

Designing an Agile Software Portfolio Architecture: The Impact of Coupling on Performance

Alan MacCormack
Martin Mocker
Robert Lagerstrom
Carliss Y. Baldwin

Working Paper 17-105

Designing an Agile Software Portfolio Architecture: The Impact of Coupling on Performance

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

Designing an Agile Software Portfolio Architecture: The Impact of Coupling on Performance

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 corporation encompasses a myriad of different software applications, each of which must work in concert to deliver functionality to end-users. However, the increasingly complex and dynamic nature of competition in today's product-markets dictates that this software *portfolio* be continually evolved and adapted, in order 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, little work has examined the antecedents of IT agility, with respect to the choices a firm makes when designing its "Software Portfolio Architecture."

We address this gap in the literature by exploring the relationship between software portfolio architecture and IT agility at the level of the individual applications in the architecture. In particular, we draw from modular systems theory to develop a series of hypotheses about how different types of *coupling* impact the ability to update, remove or replace the software applications in a firm's portfolio. We test our hypotheses using longitudinal data from a large financial services firm, comprising over 1,000 applications and over 3,000 dependencies between them. Our methods allow us to disentangle the effects of different types and levels of coupling.

Our analysis reveals that applications with higher levels of coupling cost more to update, are harder to remove, and are harder to replace, than those with lower coupling. The measures of coupling that best explain differences in IT agility include all *indirect* dependencies between software applications (i.e., they include coupling and dependency relationships that are not easily visible to the system architect). Our results reveal the critical importance of software portfolio design decisions, in developing a portfolio of applications that can evolve and adapt over time.

1. Introduction

As information has become more pervasive in the economy, information systems within firms have become increasingly 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 the needs of today's business challenges. However, they must also be flexible and adaptable, capable of evolving to meet new and emerging challenges that will undoubtedly arrive *tomorrow*. How can a firm design its portfolio of software applications in order to simultaneously confront these challenges?

Early work on the design of information systems focused on the first of these challenges: to design an architecture optimized for a given set of business conditions and strategic choices. The resulting field of study, comprising conceptual frameworks, processes and tools that seek to achieve this alignment, is known as Enterprise Architecture (EA) (Weill, 2007). The process of developing EA is "top-down" in nature. For a given firm and strategy, the goal is to design the optimal information systems architecture. Early work in this area therefore, paid little attention to understanding how the rate of change in the environment, or in a firm's own strategy, should impact this architecture. As competitive landscapes have become more complex and dynamic however, it is increasingly clear that this view of EA is no longer sufficient.

In response to calls for a greater focus on the "IT Artifact" a distinct stream of research began to explore how a firm's information systems could facilitate the development of *new* capabilities (Orlikowski and Iacono, 2001; Tiwana and Konsynski, 2010). Hence modern theories emphasize the need for IT architectures 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 can quickly reconfigure resources to respond to new challenges, ensuring a continuous alignment of IT assets with changing business needs. In contrast to EA, this work implies a "bottom-up" approach to system design. The aim is to design an architecture that can sense and respond to new challenges, the nature of which cannot be predicted in advance (Sambumurthy et al, 2003; Hanseth and Lyytinen, 2010).

Robust empirical work exploring the impact of information systems architecture on IT agility has been scarce and yields mixed results (Schmidt and Buxmann, 2011; Kim et al, 2011; Liu et al, 2013). Most studies adopt the firm as unit of analysis, capture broad holistic measures of IT infrastructure flexibility and assess the impact of these constructs on measures of overall firm performance (Duncan, 1995). For example, Tiwana and Konsynski (2010) show that firms with more modular IT architectures (i.e., that make greater use of loosely-coupled, standardized components) have higher perceived levels of IT agility. However, firms are not monolithic. They comprise different organizational units that use different applications and require different levels of flexibility (Lawrence and Lorsch, 1967). Research to better understand the roots of IT agility must recognize this heterogeneity and adopt methods to assess its impact.

We address this gap in the literature by exploring the relationship between software portfolio architecture and IT agility at the level of the individual applications in the architecture. In particular, we draw from modular systems theory to develop and test a series of hypotheses about how different types of *coupling* impact three specific dimensions of agility: the ability to update, remove or replace software applications in the firm's portfolio. Our methods, which are based upon network analysis, allow us to disentangle the impact of different types of coupling.

We test our hypotheses with data from a financial services firm. In contrast to prior work, which adopts holistic measures of IT architecture, we capture fine-grained data on the *dependencies between all software applications* in the firm. The data encompasses over 1,000 different software applications and 3,000 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, and to develop measures of IT agility. We supplement this data with figures on the annual cost of maintenance for all applications in the portfolio at the start of the period.

We find that 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 decommissioned, and are less likely to be added to the portfolio. The measures of coupling that best predict IT agility capture all *indirect* connections between applications. In sum, it is critical to account for all possible paths by which changes may propagate, when assessing the ability to update, remove or replace applications. Our work deepens our understanding of how firms can design software portfolio architectures to improve their agility.

The paper is organized as follows. In section 2, we review the literature that motivates our work. In section 3, we develop theory and derive our research hypotheses. In section 4, we describe our methods, which make use of a network-based methodology to identify and measure the coupling between software applications. In section 5, we introduce our empirical setting and describe our data. In section 6, we provide the results of our statistical tests. Finally, in section 7, we discuss the implications of our results for research and for practice.

2. Literature Review

2.1 IT Architecture Research

Research on IT Architecture was motivated by critiques of EA research which noted the emphasis on processes and governance structures through 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, the World Wide Web and the use of digital technologies 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). As a consequence, IT architecture research has focused more sharply on the "IT Artifact," and in particular, features of architecture that facilitate the development of new firm capabilities (Tiwana and Konsynski, 2010; Sambamurthy and Zmud (2000).

Early work in the field focused on understanding desirable features of IT technologies, and in particular, the antecedents of more *flexible IT infrastructure*. Duncan (1995) and Byrd and Turner (2000) established constructs thought to underpin a more flexible IT infrastructure. They emphasized the need for IT systems with greater compatibility and connectivity, through the use of system-wide standards and interfaces. They also emphasized the need for greater levels of modularity (i.e., loose coupling between applications, data and infrastructure) so that IT components could be deployed, modified and updated with minimal impact on other elements.

Subsequent work sought to deepen our understanding of how these concepts could be operationalized in firms competing in a dynamic, connected digital world. Sambamurthy and Zmud (2000) suggest the new organizing logic for IT architecture is the platform, which encompasses 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) introduce the concept of an IT "ecosystem," highlighting the different roles played by products, applications, component technologies and infrastructure technologies, including those *external* to the firm. Finally, Yoo et al. (2010) describe how pervasive digitization has given rise to a new "layered-modular" architecture, comprising devices, network technologies, services and content.

In contrast to EA models, IT Architecture research implies a "bottom-up" approach to design; the aim is to create an architecture that can sense and respond to new challenges, the nature of which cannot be predicted ex-ante (Sambumurthy et al, 2003). Firms with layered, modular IT architectures 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, and not the norm in firms with complex infrastructures (Baldwin et al 2014). Firms more often grapple with a mixture of systems of different vintages designed using different frameworks, to meet different demands for different decision makers (Ross 2003). To move from this status quo, towards a layered, modular architecture, firms must embrace new frameworks for the role of IT, and design new structures by which the included technologies will work together (Ross, 2003; 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). For example, Kim et al (2011) show 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) find IT infrastructure flexibility is *not* associated with agility, but contributes to performance only via its association with increased levels of absorptive capacity (Cohen and Levinthal, 1990). Schmidt and Buxmann (2011) show that higher quality enterprise architecture planning processes are associated with more flexible IT infrastructures. In this work however, IT infrastructure is conceived of as an *output*, whereas in studies of IT Architecture, it is typically considered an *input*.

The most important study in this stream of work comes from Tiwana and Konsynski (2010) who characterize 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, 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, the firm is conceived of as a monolithic entity; hence measures of IT function and architecture are homogenous. But firms are not monolithic; they comprise differentiated organizational units, with different objectives and different 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 (Orlikowski and Iacono, 2001; Sambumurthy and Zmud, 2000; Yoo et al, 2010). Research to better understand the roots of agility must recognize this heterogeneity, and adopt methods to assess its impact.

The second challenge relates to the fact that prior studies lack a consistent definition of agility, and fail to consider that this ability comprises *multiple* dimensions of performance, associated with differing types and levels of IT change. For example, 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 skills and resources, and may be impacted differently by properties of the software portfolio architecture. Research to better understand the roots of agility must explore the impact of IT architecture decisions across *multiple* dimensions of IT change.

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 this architecture. We define measures for the different types of coupling between applications, and develop hypotheses for how these measures impact the ability to make changes to applications. Our approach captures the heterogeneity that exists across the portfolio of applications in a firm's IT architecture, and explores the impact of this heterogeneity on multiple dimensions of change: specifically, the ability to update, remove, and replace applications.

3. Theory Development

3.1 Modular Systems Theory

The scheme by which a system's functions are allocated to components and the way that these components interact is called its "architecture" (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.

In the field of IT Architecture, several studies support this theory, showing that firm-level measures of IT modularity are associated with an IT function that is perceived as agile, adaptive and flexible (Tiwana and Konsynski, 2010). However, studies that explore this dynamic at the firm level do not provide sufficient granularity to understand the precise mechanisms through which architecture impacts agility. In order to tackle such questions, we must examine the relationship between architecture and agility at the level of the *individual applications* that form the heart of a firm's IT infrastructure. In the next section, we develop theory about the different types of coupling that exist between applications in a firm's software portfolio. We then define three distinct measures of IT agility that might be impacted by these different types of coupling.

3.2 Different Types of Coupling that Impact 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. To capture the insight that a dependency may be asymmetric, we define coupling as the property of being used (i.e., depended upon) by another component. Component A is coupled with component B if B uses A. Hence if A changes, B may have to change as well.

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 design 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 (labeled "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 exist between applications in the architecture (see **Figure 1**).

Figure 1: Coupling Relationships between Applications

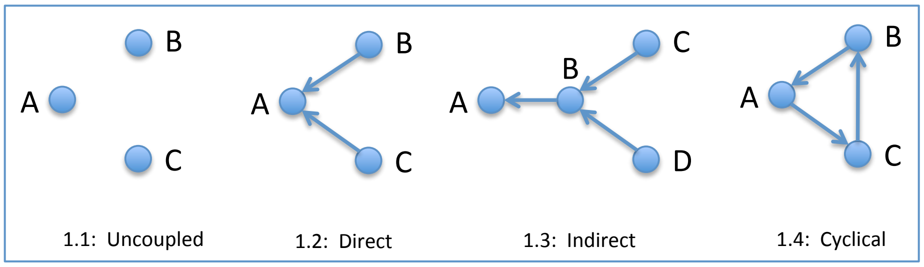


Figure 1.1 represents the base case, in which component A is not coupled to any other. In Figure 1.2, component A is *directly* coupled with components B and C. Modular systems theory predicts that components with higher levels of direct coupling are more costly to change, given the need to consider 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 likely to be weaker than direct coupling relationships, the former are not as visible to a system architect, hence more likely to produce unintended system behaviors. Empirical work has shown that changes to one component in an IT system often create unexpected disruptions in distant parts of the system (Vakkuri, 2013). Measures of indirect coupling have been shown to predict the number of defects and the ease with which software can be adapted (MacCormack, 2010; MacCormack and Sturtevant, 2016).

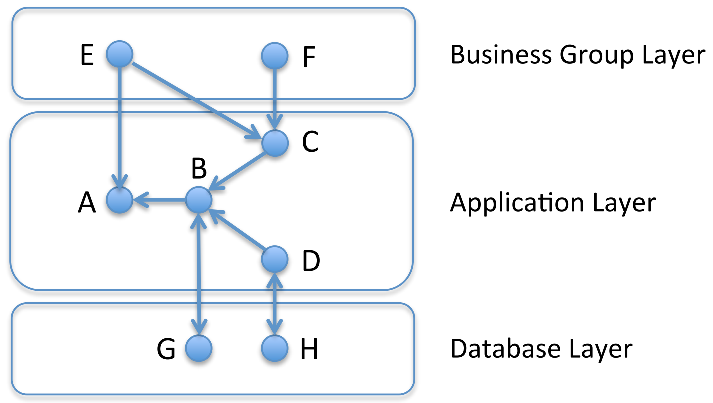
Figure 1.4 illustrates a third pattern of coupling between applications, called *cyclic* coupling (Whitney, et al, 2004; Sosa et al, 2013). In this system, A is coupled with B, B is coupled with C, and C is coupled with A. These components form a cyclic group – a group of components that are mutually interdependent. In contrast to figure 1.3, there is no "hierarchy" or ordering of these components, such that one can be designed (or changed) before the others. Rather, components in cyclic groups must often be designed (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).[1]

Importantly, the portfolio of software applications within a firm represents only one layer in a firm's IT architecture. Applications are the chief mechanism through which a firm's IT infrastructure supports the delivery of core business capabilities (MacCormack et al, 2016). But other 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*

[1] Measures of different types of coupling are likely to be correlated. It will be important to be sensitive to this in empirical tests.

applications and other layers. In this study, we consider the layers immediately above and below the application layer. In particular, business groups use applications and applications may read from/write to databases[2] (see **Figure 2**). Modular systems theory suggests that applications with higher levels of coupling to these layers will be more costly and difficult to change.

Figure 2: Coupling Relationships between Layers



3.3 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. Subsequent studies often assumed these constructs to be *proxies* for IT agility and sought to identify their antecedents. For example, Schmidt and Buxmann (2011) find that higher-quality EA planning processes are associated with higher perceived levels of connectivity, compatibility and modularity. Similarly, Joachim et al (2013) find that governance mechanisms used in Service Oriented Architectures (SOA) are associated with IT infrastructure flexibility, as measured by connectivity, compatibility and modularity.

In contrast to studies that view the constructs of connectivity, compatibility and modularity as proxies for IT agility (i.e., measures of *output*) other work considers these to be features of IT architecture (i.e., measures of *input*) and explores their impact on performance.

[2] Applications may read from or write to a database; hence we denote this dependency as bi-directional in this figure.

For example, Kim et al (2011) show that measures of compatibility, connectivity and modularity are correlated with perceptions of a firm's ability to change existing business processes. Similarly, Tiwana and Konsynski (2010) show that the use of loose coupling and standardization are correlated with perceptions of the IT function as being agile, adaptive, flexible and responsive. We share the view that concepts such as modularity and loose coupling capture features of IT architecture, and are not proxies for IT agility. In our work therefore, we define explicit measures of IT agility, to test the assertion that these constructs impact 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 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 capture 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 represents (in theory) the lowest level of IT agility. Second, *we capture whether an application is decommissioned* (i.e., retired) over a 4-year period. Decommissioning applications that are obsolete or no longer needed is a critical task in the modern firm, and growing in importance given a dynamic and competitive marketplace (IBM, 2009). Indeed, a recent study of firms' software portfolios by Aier et al (2009) found that over 40% of applications were retired over a 4-year time period. Decommissioning is a major undertaking, requiring 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. As a result, applications with high levels of coupling are likely to be more difficult and costly to retire. Finally, *we capture the addition of new applications to a firm's portfolio*. As firms replace obsolete technologies, build new capabilities, and enter new markets, adding new applications is an increasingly critical capability. This requires the development of new functionality, and the integration of this functionality with existing applications and infrastructure. Integrating new applications into a firm's software portfolio is likely to be more costly and difficult to the extent that they require high levels of coupling to existing applications and infrastructure.

3.4 The Relationship between Application Coupling and IT Agility

We have identified the different types of coupling that exist between 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 measures of coupling predict the degree to which an application can be changed. Hence we state our hypotheses, as follows:

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 decommissioned*, on average, than applications with lower levels of coupling.

Hypothesis 3: New applications added to the software portfolio *will have lower levels of coupling*, on average, than the legacy applications that comprise this portfolio

For each of the hypotheses above, we conduct empirical tests for the impact of all types of coupling defined in our theory. We have no ex-ante view as to the relative strength of effects for the different types of coupling we define. Rather, we rely on our empirical tests to identify which of them have more explanatory power in predicting different dimensions of IT agility.

4. Research Methods

To develop measures of the different types of 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. Importantly, DSMs allow us to capture the *direction* of dependencies between elements, and hence to discriminate between incoming and outgoing dependency relationships. Using a matrix to capture dependency relationships also facilitates the discovery of *indirect* dependencies 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" of 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 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, showing how measures derived from the DSM of a firm's *software portfolio architecture* predict measures of IT agility.

4.1 Measuring *Direct* Coupling and Dependency 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 (i.e., dependency) proceeds from row to column in our DSMs, hence coupling proceeds from column to row. That is, reading down the column of an application reveals all applications that are coupled with it. As a matter of definition, main diagonal elements are set to one (i.e., applications "depend on" themselves).

The levels of direct coupling and dependency for an application can be read directly from the DSM. Specifically, for the *i*th application in a portfolio, the level of direct coupling is found by summing entries in the *i*th column. The level of direct dependency is found by summing entries in the *i*th row.[3] In general, these measures will be different, unless all 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 and Dependency in a DSM

Using a DSM, we can also find the *indirect* dependencies between applications, which reflect the potential for changes to propagate. To identify indirect relationships in a system, 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 (MacCormack et al., 2006; Baldwin et al., 2014). The visibility matrix captures all of the *indirect* dependencies between applications.[4] In a similar fashion to the direct dependency DSM, the level of indirect coupling for an application is therefore captured in the column sums of the visibility matrix. The level of indirect dependency for an application is captured in the row sums of the visibility matrix.[5]

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 have been observed across

[3] In prior work, the term Direct Fan-In (DFI) has been used to denote the direct coupling of a component, and Direct Fan-Out (DFO) has been used to denote the direct dependency of a component (Baldwin et al, 2014).
[4] In our work, we define *indirect* coupling and dependency as also encompassing all *direct* relationships between elements.
[5] In prior work, the term Visibility Fan-In (VFI) has been used to denote the indirect coupling for a component, and Visibility Fan-Out (VFO) to denote the indirect dependency for a component (e.g., Baldwin et al, 2014).

software systems of similar size and function (Baldwin et al, 2014). These differences are predicted, in part, by the structure of the developing organization (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). In combination, these findings suggest that in complex systems, design decisions are impacted significantly by organizational constraints, as well as explicit design choices.

4.3 Measuring *Cyclic* Coupling in a Design Structure Matrix

The visibility matrix can be used to identify "cyclic groups" of applications, each of which is directly or indirectly connected to all others. Mathematically, members of a cyclic group all have the same indirect coupling and indirect dependency measures, given that they are all connected directly or indirectly to each other. Thus we can identify cyclic groups in a system by sorting applications by these two measures (Baldwin et al., 2014).

Prior work has shown that 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 and dependency that they exhibit, as compared to members of this cyclic group. We apply the same classification process to applications in a firm's software portfolio.

Cyclically coupled applications are members of the largest cyclic group, and have high levels of both indirect coupling and dependency. *Indirectly coupled* applications have high levels of indirect coupling (i.e., they are "used," directly or indirectly, by many other applications). *Indirectly dependent* applications have high levels of indirect dependency, (i.e., they "use," directly or indirectly, many other applications). *Peripheral* applications have low levels of both indirect coupling and dependency. In a software portfolio, indirectly coupled, cyclically coupled and indirectly dependent applications are called “main flow” applications. Peripheral applications lie outside the main flow, being loosely coupled to other applications.

4.4 Revealing *Hierarchy* using a Design Structure Matrix

When used as a planning tool in a design process, a DSM indicates a possible sequence of design tasks, i.e., which components should be designed before which others (Steward, 1981; Eppinger et al, 1994). In general, it is intuitive and desirable to place the first design tasks at the top of a DSM, with later tasks below. In sum, the first components to be designed should be those that other components depend upon. Reflecting this discussion, we place the “most used” applications at the top of the DSM and “users” of other applications towards the bottom of the DSM. Hence applications at the top have high levels of indirect coupling whereas applications towards the bottom have high levels of indirect dependency. The resulting DSM possesses a “lower diagonal form,” in which most dependencies are below the diagonal, with above diagonal entries indicating the presence of cyclical coupling (Sosa et al, 2013). Critically, in cases where the hierarchy of applications is not known a priori, the visibility matrix can be sorted using measures of indirect coupling and dependency to *reveal* these relationships. In our work, we carry out this procedure, to reveal the implicit hierarchy among applications.

5. Empirical Setting

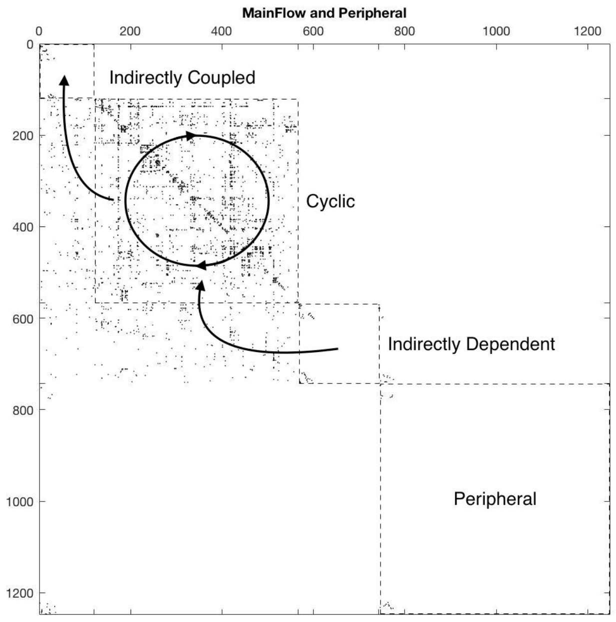
We test our hypotheses using data on the software portfolio of a large European bank. The data was a part of an initiative taken to develop a better understanding of the linkages between software applications, and the performance of the portfolio. Each quarter, the bank asks application owners to enter information in a database. For each application, data is collected on the go-live-date of the application, if it is in-house or externally developed, if it is under development or in production, the departments that use the application, the operating systems that it supports, the databases that it uses, and the dependencies it has with other applications.

In order to test our hypotheses about IT agility, we captured data on active software applications and their dependencies in both 2008 and 2012. We were also given data on operating and maintenance costs for 2008. In 2008, the collection of data on the software portfolio was fairly new and consequently, dependency data was not provided for all applications. Further, data on operating and maintenance cost was only reported for a subset of applications. Hence our sample for analysis does not consist of all active applications. However, we were told, in general, that the data included the most important of them. Missing data was more likely for applications that were smaller and less important.

Sample Data for 2008

The 2008 software portfolio consists of 1,558 active applications. Of these, 1,247 contained reliable data on application dependencies. Thus, our sample consists of 1,247 applications and 3,482 dependencies. Using the Design Structure Matrix methodology described earlier, we identified all of the direct and indirect coupling and dependency relationships between applications in the portfolio. We then classified applications using the methods described earlier (Baldwin et al, 2014). We find the 2008 software portfolio architecture has a large *cyclically coupled* group of 447 applications representing 36% of the system. These applications are all mutually interdependent. We find 120 applications (10%) are *indirectly coupled* (i.e., they have high indirect coupling but low indirect dependency). We find 175 applications (14%) are *indirectly dependent* (i.e., they have high indirect dependency, but low indirect coupling). Finally, we find 505 applications (41%) are *peripheral* (i.e., they have low levels of both indirect coupling and dependency). Figure 3 shows the firm's software portfolio architecture in DSM form, with applications grouped by category. Arrows indicate the "main flow" of dependencies between groups. (Peripheral applications are not in the main flow).

Figure 3: Coupling and Dependency Relationships for the 2008 Software Portfolio



Sample Data for 2012

The 2012 software portfolio contains 1,251 applications and 3,969 dependencies. All applications contained sufficient data for analysis in this time period, hence our sample represents the entire population. The analysis of the 2012 portfolio reveals a large group of 441 *cyclically coupled* applications, representing 35% of the system. We find 80 applications (6%) are indirectly coupled, 298 applications (28%) are indirectly dependent and 432 applications (35%) are peripheral. Table 1 shows a comparison of the firm's applications grouped by indirect coupling category, in 2008 and 2012. While the number of applications in the two time periods is consistent, and the number of applications in each category broadly similar, this analysis hides a significant movement of applications into and out of the portfolio, as discussed below.

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

	2008		2012	
Category	**Number**	**%**	**Number**	**%**
Indirectly Coupled	120	9.6%	80	6.4%
Cyclically Coupled	447	35.9%	441	35.3%
Indirectly Dependent	174	14.0%	298	23.8%
Peripheral	505	40.5%	432	34.5%
TOTAL	1247	100.0%	1251	100.0%

5.1 Application changes between 2008 and 2012

Between 2008 and 2012 there was substantial change in the software portfolio at the bank. In particular, the bank went through a merger with another bank, and as a result, underwent 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 decommissioned, new applications were added, existing applications were updated (and may have switched categories), and data was collected for applications where none existed in 2008. Table 2 shows the movement of software applications out of (retired), into (added), and across (moved category) the software portfolio between 2008 and 2012. Furthermore, we show active applications with missing data in 2008, for which data was available in 2012.

Table 2: Movement of Applications in the Software Portfolio between 2008-2012

	2008	Application Retired	Application Added	Net Moved Category	New Data Available	2012
Indirectly Coupled	120	-53	14	-19	18	80
Cyclically Coupled	447	-121	72	-8	51	441
Indirectly Dependent	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 cost data were aggregated across multiple applications). The applications for which maintenance cost data were available is not randomly distributed, but is biased towards applications that are more important. Hence we must control for this bias.

We control for the non-random exclusion of maintenance cost data by rebalancing our sample for hypothesis one, to ensure that the sample has the same characteristics as the population. Table 4 presents data on how we achieve this. Consider, our sample of active applications in 2008 for which we have data is 1,247, of which 505 (40.5%) are peripheral applications. However, 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 the 2008 population. In order to create a sample for testing hypothesis one, we therefore oversample the observations in underrepresented categories, to match the proportions of the 2008 population.[6] For example, we replicate the 19 observations from peripheral applications, to produce a total of 266 applications in this category. After this procedure, our final sample contains 660 observations with which to test hypothesis one, distributed 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
Indirectly Coupled	120	9.6%	37	9.8%	74	11.2%
Cyclically Coupled	447	35.9%	249	66.2%	249	37.7%
Indirectly Dependent	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%

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

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 inter-layer and inter-application coupling.

Table 5: Measures used in the Study

Dependent Variables: Measures of IT Agility

MaintCost – 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).

Decomm – We capture whether an application was decommissioned (1) or suvived (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).

Inter-Layer Coupling – Inter-Layer

# 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.

Inter-Application Coupling

DirCop – measures the number of *directly coupled* applications for an application

DirDep – measures the number of *directly dependent* applications for an application.

IndCop – indicates whether an application is in the *indirectly coupled* category (1) or not (0).

CycCop – indicates whether an application is in the *cyclically coupled* category (1) or not (0).

IndDep – indicates if an application is in the *indirectly dependent* category (1) or not (0).

MainFlow – Indirectly coupled, cyclically coupled, or indirectly dependent applications are main-flow applications.

5.4 Descriptive statistics

In Table 6, we provide descriptive statistics for the samples used to test hypotheses 1-2. The samples are different for each test, given maintenance cost is only available for a subset of 2008 applications, whereas decommissioning is an observable outcome for all applications that exist in 2008. (The test for hypothesis three is a comparison of coupling and dependency levels for new applications versus all applications that exist in 2008 – the sample for the latter is the same as Hypothesis 2). Correlation tables for the 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
MaintCost	0	5,776.50	314.66	569.60	-	-	-	-
Decomm	-	-	-	-	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
DirCup	0	85	3.43	6.46	0	85	3.17	6.97
DirDep	0	35	3.62	5.92	0	79	3.25	6.36
MainFlow	0	1	0.60	0.49	0	1	0.63	0.48
IndCup	0	1	0.11	0.32	0	1	0.10	0.30
Cyclic	0	1	0.38	0.48	0	1	0.40	0.49
IndDep	0	1	0.11	0.31	0	1	0.14	0.34
	n=660				*n=957*[7]			

First, we note maintenance cost is skewed; hence we use a log transformation for this variable in statistical tests. Second, the rate at which applications are decommissioned between 2008 and 2012 is 56%. This mirrors other empirical work in this area that demonstrates high turnover in software portfolios (Aier et al, 2009). The average age of applications is 8.7 years. Age is also skewed; hence we use a log transformation for this variable in statistical tests. Almost all applications (98%) are in production and only 6% support more than one operating system. 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.

6. Empirical Results

6.1 Hypothesis 1: The Relationship between Coupling and Maintenance Cost

[7] Control variable data was not available for all 1257 applications. Hence our statistical models are based upon n=957.

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

Table 7: Models Predicting the Cost of Application Maintenance

Ln (MaintCost)	**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 (DirCop)			0.212†			
Ln (DirDep)			0.289*			
MainFlow (MF)				1.409***		1.307***
IndCop					1.524***	
Cyclic					1.417***	
IndDep					1.265***	
MF x Ln DirCop						0.162
PER x Ln DirCop						-0.233
PER x Ln DirDep						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 inter-layer coupling variables, both of which are significant. Applications that are used by more business departments and that are connected to a database management system cost more to maintain. In total, these inter-layer coupling variables increase the variation explained to 13.5%. Models 3-6 explore the predictive power of various measures of inter-application coupling. Model 3 includes measures of direct coupling and dependency. We use a log transformation for these variables given they are highly skewed. Only one of the variables is significant hence the increase in R-squared for this model is limited as compared to model 2 (from 13.5% to 15.3%).[8] In model 4, we remove direct coupling and dependency variables, and instead include main-flow – a variable that indicates whether an application is indirectly coupled, cyclically coupled or indirectly dependent. 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 three component categories – indirectly coupled, cyclically coupled and indirectly dependent. 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 and 5, given the high correlations between direct and indirect coupling and dependency measures (see the Appendices). We note however, that measures of indirect coupling and dependency have a higher correlation with maintenance cost than measures of direct coupling and dependency, and a greater level of statistical significance in our models. *We conclude that indirect coupling and dependency measures are more important than direct coupling and dependency measures in explaining maintenance cost – the first dimension of IT agility*.

In model 6, we use interaction terms to evaluate the impact of direct coupling and dependency measures *within* different categories associated with indirect coupling and dependency. In particular, we interact the main-flow and peripheral variables, with the level of direct coupling and dependency for an application. For peripheral applications, we find the measure of direct dependency adds significant explanatory power to our model. In total, our

[8] We note that direct coupling and direct dependency are strongly correlated (see the Appendices); hence this model should be considered a joint test of significance for these variables.

final model explains 19.5% of the variation in maintenance costs. Control variables explain 8.5% of this variation, and coupling variables explain 11% of this variation. Among the coupling variables, inter-layer coupling variables explain 5% of the variation, and inter-application coupling variables explain 6% of the variation.

6.1.1 Exploring the Impact of Indirect Coupling on Application Maintenance Cost

To understand the dynamics of how *indirect* coupling and dependency impact maintenance cost we further analyzed the relationship between the outcome and these categories. 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 applications have the highest average maintenance cost, followed by indirectly coupled, indirectly dependent, and lastly peripheral applications. They also have a higher variation in maintenance costs than applications in other categories. The implication is that cyclically coupled applications are harder to predict (and hence to budget) with respect to maintenance costs and also more likely to be outliers on this dimension of agility.

Table 8: Differences in Maintenance Cost by Category

	Maintenance Cost		
	Mean	**St. Dev**	**Skewness**
Indirectly Coupled	344.86	452.48	2.21
Cyclically Coupled	497.54	753.80	3.55
Indirectly Dependent	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 greater amounts of coupling cost more to maintain. We find support for the predictive power of inter-layer coupling variables (Users and Database Management systems) as well as inter-application coupling variables. We find that indirect coupling and dependency are better predictors of maintenance costs than direct coupling and dependency. While our statistical models cannot differentiate between the impact of different types of indirect coupling and dependency, cyclically coupled applications possess the highest maintenance costs, and experience the highest variation in costs.

6.2 Hypothesis 2: The Relationship between Coupling and Decommissioning

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

Table 9: Logistic Regression Models Predicting Application Decommissioning

Decomm (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 (DirCop)			-0.638***			
Ln (DirDep)			-0.359***			
MainFlow				-2.664***		-1.491***
IndCop					-2.536***	
Cyclic					-3.063***	
IndDep					-2.157***	
MF x DirCop						-0.437***
PER x DirDep						-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 decommissioned. Applications that support more operating systems are less likely to be decommissioned. In total, these variables explain 24.1% more variation than the null model (i.e., a model with no predictors).[9] In model 2, we add inter-layer coupling variables; one is strongly significant ($p<0.1\%$), the other marginally significant ($p<10\%$). Applications that make use of more database management systems are less likely to be decommissioned. Applications with more users may be less likely to be decommissioned. In total, these variables increase the variation explained to 35.8%. Models 3-6 explore the predictive power of various measures of inter-application coupling. Model 3 includes measures of direct coupling and dependency. We use a log transformation for these variables given they are highly skewed. 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 and dependency variables, and instead include main-flow. This variable is strongly significant, and increases the model fit to 50.0%.

In model 5, we split main-flow applications into three component categories – indirectly coupled, cyclically coupled and indirectly dependent. All three are significant. In addition, the model fit improves over model 4, from 50.0% to 51.4%. We note that the coefficients on the three variables are statistically different from each other. Specifically, cyclically coupled applications are the least likely to be decommissioned, and indirectly dependent applications are the most likely to be decommissioned (but still far less likely than peripheral applications). We cannot include measures of direct coupling in models 4 and 5, given the high correlations between direct and indirect coupling and dependency measures (see the Appendices). We note however, that measures of indirect coupling and dependency have a higher correlation with decommissioning than measures of direct coupling and dependency, and a greater level of statistical significance in our models. *We conclude that indirect coupling and dependency*

[9] 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%).

measures are more important than direct coupling and dependency measures in explaining decommissioning – the second dimension of IT agility.

In model 6, we use interaction terms to evaluate the impact of direct coupling and dependency measures *within* different categories associated with indirect coupling and dependency. In particular, we interact the main-flow and peripheral variables, with the level of direct coupling and dependency for applications. For main-flow applications, we find the measure of direct coupling adds significant explanatory power to our model. For peripheral applications, we find the measure of direct dependency adds significant explanatory power to our model. In both cases, higher levels of direct coupling or dependency within a category are associated with a lower likelihood of being decommissioned. In total, our final model explains 52.8% more of the variation in application decommissioning than the null model. Control variables account for 24.1% of the improvement in model fit, and coupling variables account for 28.7%. Among the coupling variables, inter-layer coupling variables account for 11.7% of the improvement in model fit, whereas inter-application coupling variables account for 17%.

6.2.1 Exploring the Impact of Coupling on Application Decommissioning

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

Table 10: Differences in Application Decommissioning by Category

	Applications in 2008	Decommissioned by 2012	Percentage Decommissioned
Indirectly Coupled	120	53	44.2%
Cyclically Coupled	447	121	27.1%
Indirectly Dependent	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 greater coupling are less likely to be decommissioned. We find support for the predictive power of inter-layer coupling variables (Users and Database Management systems) and inter-application coupling variables. We find that indirect coupling and dependency are better predictors of decommissioning than direct coupling and dependency. And our statistical models show that cyclically coupled applications are much less likely to be decommissioned than other categories. The descriptive analysis highlights this result, and shows, by contrast, the huge rate at which peripheral applications are decommissioned 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 have lower levels of coupling and dependency compared to the legacy applications in the portfolio at the start of this period. To test this assertion, we compare the distribution of applications by category for 2008, to that of new applications added to the portfolio. All else being equal, one would predict that new applications should mirror the distribution of legacy applications, in terms of coupling and dependency (the “null” hypothesis).

Table 11 shows the distribution of applications by indirect coupling category for the 2008 portfolio, as well as for all new applications added between 2008 and 2012.[10] For example, in 2008, there were 120 applications that were indirectly coupled, representing 9.6% of the portfolio. Of the 423 new applications added between 2008 and 2012 however, only 14 (i.e., 3.3% of new additions) were indirectly coupled. If new applications were added in a way that mirrors the coupling of legacy applications, we would expect 9.6% of the 423 new applications (i.e., 40 applications) to be indirectly coupled. The actual outcome is significantly below what is expected. The ratio of the actual to the expected outcome is 0.34. This reflects the degree to which the number of new applications either falls short of (i.e., is less than one) or exceeds (i.e., is more than one) the expected number of applications in a category for the null hypothesis.

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**
Indirectly Coupled	120	9.62%	14	3.30%	0.34
Cyclically Coupled	447	35.85%	72	17.02%	0.47
Indirectly Dependent	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 indirectly coupled and cyclically coupled categories. By contrast, new applications occur at a greater rate than expected in the indirectly dependent and peripheral categories. These results make intuitive

[10] 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.

sense. Adding new applications that have little or no coupling or dependency relationships with other applications (i.e., peripheral applications) should be relatively easy to do. Furthermore, adding new applications that use or "depend upon" existing applications should be easier than adding new applications that are used by or "depended upon" by existing applications. In essence, new applications can take advantage of the existing functionality provided by legacy applications (but not the reverse). Our results show this dynamic – adding new applications that depend on existing applications – happens 85% more than expected (i.e., the ratio is 1.85).

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 new applications added between 2008 and 2012. The test statistic shows a strong relationship between the four categories and the addition of new applications, as compared to the 2008 distribution (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 different level of coupling and dependency than existing applications. In particular, new applications are more likely to be peripheral or indirectly dependent, and less likely to be indirectly coupled or cyclically coupled.

7. Discussion

The main contribution of this paper is in developing theory about the relationship between a firm's software portfolio architecture and IT agility. Specifically, we find a strong link between the level of coupling and dependency for individual applications in the software portfolio, and the degree to which applications can be changed. Applications that possess greater levels of coupling and dependency are more costly to maintain, and less likely to be decommissioned. Furthermore, new applications added to the portfolio differ significantly from the applications in the legacy portfolio. In particular, they are less likely to possess high levels of coupling, and more likely to be peripheral, or depend only on applications that exist.

Our work explores coupling and dependency relationships both within the application layer, and between this layer and others in the IT architecture. With respect to the former, we show that *indirect* coupling and dependency relationships have a stronger association with IT agility than *direct* coupling and dependency relationships. With respect to the latter, we show that the number of users (i.e., business groups) for an application, and the number of database management systems to which it is connected, are also strong predictors of IT agility. The best models predicting maintenance costs and application decommissioning include measures of *all* aspects of coupling and dependency: direct, indirect, inter-layer and inter-application.

In order to highlight the power of the various measures of coupling employed in this study, we conduct a decomposition of variance for hypotheses one and two. Table 12 shows the amount of explained variance in each outcome that is attributable to control variables, inter-layer coupling variables, and inter-application coupling and dependency variables. We break the latter into three; first showing the variation explained by *direct* coupling and dependency measures, next showing the variation explained by *indirect* coupling and dependency measures, and finally showing the variation explained by all of these measures combined (i.e., our best fit models).

First, while control variables explain a sizeable amount of the explained variance in our models, measures of coupling explain more of the variance for each outcome. Second, measures of *indirect* coupling and dependency have a significantly stronger impact than measures of *direct* coupling and dependency in both cases. In models predicting maintenance cost, indirect measures explain over twice the variance of direct measures. In models predicting decommissioning, indirect measures explain 32.7% more of the variance than direct measures.

Third, despite the statistical dominance of indirect measures in our models, and the strong correlation between direct and indirect measures, *the best model in each case combines direct and indirect measures*. In essence, measures of direct coupling and dependency help to explain variations in each outcome that remain *within* the indirect coupling and dependency categories.

Table 12: Decomposition of Variance Explained by Hypothesis

	Maintenance Cost H1			Decommissioning H2		
Control Variables	8.5%	8.5%	8.5%	24.1%	24.1%	24.1%
External Coupling	5.0%	5.0%	5.0%	11.7%	11.7%	11.7%
Internal-Direct	1.8%			11.0%		
Internal-Indirect		4.2%[11]			14.6%[12]	
Internal-Combined			6.0%			16.0%
TOTAL EXPLAINED	15.3%	17.7%	19.5%	46.8%	50.0%	52.8%

Furthermore, while both inter-layer and inter-application measures of coupling and dependency are significant in our models, we find the latter have more power in predicting IT agility. With respect to maintenance costs, inter-application measures account for 20% more of the variation than inter-layer coupling measures (i.e., 6.0% versus 5.0%). With respect to decommissioning, inter-application measures account for 36.8% more of the variation than inter-layer coupling measures (16.0% versus 11.7%). These results reveal 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 and dependency that exist *between* the components of this portfolio.

[11] 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.
[12] 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.

The finding that *indirect* measures of coupling and dependency have more power in predicting IT agility than *direct* measures 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 inspection of an application's nearest neighbors in the portfolio, but represent "hidden structure" that can only be revealed by an analysis of indirect pathways in a system (Baldwin et al, 2014).

Our work suggests that *cyclically coupled* applications present the toughest challenges to a system architect. These applications have the highest average cost, and the highest variations in cost. They are also far less likely to be decommissioned than other applications. Finally, new applications of this type are added to the portfolio at a rate 50% lower than would be expected. In the firm we studied, 35% of the applications were cyclically coupled in both 2008 and 2012. Hence 440+ applications were mutually interdependent over this timeframe. Making changes to a "core" set of applications of this size, would likely be a hugely complicated endeavor, given each change to a single application could propagate to affect all others.

Our study has distinct implications for managers. In particular, our methodology provides a way to measure the *real* software portfolio architecture that firm's possess, as opposed to the high level conceptual representations often found in documents depicting a firm's IT systems. The insights generated 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; ii) monitoring the evolution of the software portfolio over time, as new applications and/or dependencies are introduced, and; iii) identifying opportunities to improve the portfolio, for example, by reducing coupling, and hence the cost of change for an application.

Ironically, in this era of big data, the lack of granular data may be the largest barrier to the systematic investigation of software portfolio architecture. Firms need to capture data on the coupling and dependency between applications in the portfolio, and the way that these evolve over time. To use this data for prediction, they must also systematically capture data on the cost of change. *In most organizations with which we have worked, this type of data does not exist.* In some, efforts have been made to collect this data manually. However, there are many challenges associated with this approach, including a lack of incentives to provide accurate and timely information. In essence, many firms do not know their "real" software portfolio architecture.

Our work opens up the potential for further research to explore the relationship between software portfolio architecture and IT agility. In this study, features of our dataset made it difficult to disentangle the effects of different categories of indirect coupling and dependency. 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 they have with IT change. While we focused only on software applications in our analysis of IT agility, our methods could be extended further, to look at the cost of change for other IT system components.

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 an application, the data to test our theoretical propositions comes from a single firm. Hence additional work is needed to validate that our results hold for other firms. We may find that different types of firm, or different managerial processes within firms, influence the results. Indeed, studies *across* different organizations might reveal how measures of IT architecture impact firm-level performance. This area is promising, given prior literature argues there is a strong link between IT architecture and firm-level agility. We hope that this paper and the methods it describes, will allow us to answer these questions, with a robust 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.

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. 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.

MacCormack, A., and Sturtevant, D. 2016. Technical Debt and System Architecture: The Impact of Coupling on Defect-Related Activity. *Journal of Systems and Software*, accepted for publication.

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.

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

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.

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.

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,

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*	MainF	Cyclic	IndCup	IndDep	*DirCup*	*DirDep*
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						
MainF	0.35*	0.23**	-0.07	0.08*	-0.29*	0.49*	0.02	1					
Cyclic	0.27*	0.33*	-0.04	0.04	-0.29*	0.40*	0.09*	**0.64***	1				
IndCup	0.09*	-0.06	0.03	0.03	0.06	0.09*	-0.01	0.29*	-0.28*	1			
IndDep	0.05	-0.09*	-0.14*	0.12*	-0.08*	0.05	-0.10*	0.29*	-0.27*	-0.12*	1		
DirCup	0.27*	0.47*	0.05	-0.07	-0.18*	0.37*	0.17*	0.58*	0.70*	0.09*	-0.27*	1	
DirDep	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)

	Decomm.	*Age*	State	OS	Vendor	DBMS	*Users*	MFlow	Cyclic	IndCup	IndDep	*DirCup*	*DirDep*
Decomm.	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						
MainF	-0.58*	-0.06	0.02	0.09*	-0.53*	0.51*	0.09*	1					
Cyclic	-0.48*	0.14*	0.06	0.01	-0.45*	0.49*	0.12*	**0.62***	1				
IndCup	-0.08*	-0.06	0.03	0.01	-0.03	-0.09	0.09	0.25*	-0.26*	1			
IndDep	-0.06	-0.15*	-0.08*	0.01*	-0.08*	0.03	-0.06*	0.30*	-0.32*	-0.13*	1		
DirCup	-0.51*	0.19*	0.09*	0.07	-0.46*	0.51*	0.20*	**0.62***	**0.76***	0.09*	-0.28*	1	
DirDep	-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()