8-5-2011

Digital Cement: Software Portfolio Architecture, Complexity, and Flexibility

David Dreyfus
Boston University, ddreyfus@bu.edu

George Wyner
Boston University, gwyner@bu.edu

Digital Cement: Software Portfolio Architecture, Complexity, and Flexibility

David Dreyfus
Boston University
ddreyfus@bu.edu

George M. Wyner
Boston University
gwyner@bu.edu

ABSTRACT

This paper is about the relationship between an organization's software portfolio architecture and its ability to make changes to it. Responding to business and technology changes often involves modifying the software portfolio and the speed and cost of making changes to the software portfolio is a measure of the system's flexibility. The specific research question is: "How does software portfolio architecture affect software portfolio flexibility?"

This research develops measures of architectural and component complexity and hypothesizes that these constructs affect one dimension of software portfolio flexibility: architectural flexibility. The hypotheses are tested by (1) collecting component and dependency data from a biopharmaceutical company's software portfolio, (2) combining this data with survey, system instrumentation, and archive data, and then (3) estimating and interpreting multiple regression models. The general conclusions from the research are that both complexity at the component level and complexity at the architectural level affect software portfolio flexibility.

Keywords

Information system architecture, complexity, information system flexibility, software measurement, software portfolio.

INTRODUCTION

Organizations build or acquire collections of software assets (e.g., applications, databases, scripts, servers, etc.) with the expectation that these assets will enable the organization to effectively respond to its environment. These assets are frequently integrated with each other so that data and processing can be seamlessly shared. These integrations, however, create dependencies between these assets, which raises the question of whether such dependencies could in any way restrict an organization's ability to respond to changes in its business and technology environment. The purpose of this paper is to show that these dependencies do affect an organization's ability to change.

We call the collection of software assets assembled to satisfy the needs of an organization a **software portfolio**. A software portfolio is not a random collection of assets; it is a specific set of assets assembled into a system to fulfill a purpose (Mason and Mitroff, 1973). Our analysis focuses on how decisions made *by the organization* while it constructs its software portfolio affect the organization's subsequent ability to change.

Using the definition provided by Brooks (1975), we define the pattern of dependencies between components within the software portfolio as the **software portfolio architecture**. Architectures describe the underlying structure of a system (Simon, 1996). While we often think of architecture as normative, it needn't be exclusively so. In this research we take a descriptive approach to software portfolio architecture, discovering the pattern of dependencies as they exist in the organization's software portfolio.

Software portfolios are not static, nor are they monolithic; people within the organization make choices to add, modify, remove, and combine components as they deem appropriate (Huber, 1990) in response to changes in their environment, changes in available technologies, and competitive pressures (Lawrence and Lorsch, 1967). Organizations interested in retaining or increasing their ability to make changes need to know those properties of their software portfolios that affect their flexibility (Byrd and Turner, 2000; Duncan, 1995; Kayworth, Chatterjee and Sambamurthy, 2001). This paper addresses this need by exploring the relationship between characteristics of a software portfolio's architecture and its resultant flexibility. The specific research question examined in this paper is: "How does software portfolio architecture affect software portfolio flexibility?" As many of the constructs and theory are new to this research context, this investigation is exploratory.

The ability to change the software portfolio architecture (flexibility) has important consequences. An organization's software portfolio implementation can either facilitate change by operating as innovation infrastructure (Weill and Broadbent, 1998), or it can retard change by acting as a form of *digital cement* in which the software portfolio hardens in place, leaving the organization unable to change the software portfolio and thus in some respects unable to change itself.

In this paper we will develop a set of hypotheses concerning the relationship between architecture and flexibility which we then test in the context of the software portfolio of a biopharmaceutical company.

LITERATURE REVIEW

Prior research has not adequately explored the characteristics of software portfolios within their organizational contexts (Orlikowski and Iacono, 2001). The research that does exist has focused on individual components (Darcy, Kemerer, Slaughter and Tomayko, 2005; MacCormack, Rusnak and Baldwin, 2006), not systems of components, the focus of this paper.

Researchers have described architecture in a variety of ways (Allen and Boynton, 1991; Broadbent and Weill, 1997). Most discussions of architecture are normative: what architecture *should* be and how it should be created "from scratch" (Zachman, 1987). Software portfolio architecture is a description of the components that constitute the system; how those components interact, connect, and communicate with each other; and how the components operate to realize the software portfolio designers' intent (Baldwin and Clark, 2000; Perry and Wolf, 1992; Ulrich, 1995).

Relatively unexplored is the value of descriptive architecture: architecture used to describe a system after it has been built (Henderson and Clark, 1990). This paper focuses on descriptive architecture.

Researchers have focused on the implications of architecture in a variety of domains (Banker, Davis and Slaughter, 1998; Duncan, 1995; Henderson and Venkatraman, 1993; Ross, Weill and Robertson, 2006; Sambamurthy, Bharadwaj and Grover, 2003) using a number of different methods (Bass, Clements and Kazman, 1998; Nezlek, Jain and Nazareth, 1999; Shibata, Yano and Kodama, 2005). Software infrastructure is close to the construct of a software portfolio in that both constructs consist of multiple, interdependent software assets. The infrastructure flexibility literature has built a conceptual and measurement model of flexibility as comprising architectural and organizational characteristics measured through survey instruments (Byrd and Turner, 2000; Nelson and Ghods, 1999). Infrastructure architectures, however, have not been measured structurally, and their effects on infrastructure flexibility have not been evaluated. In this paper, we explore methods and theory to address this gap within the broader context of software portfolios rather than infrastructure only.

THEORY

Software Portfolio Architecture

A system is a combination of components that function as a complete whole (Baldwin and Clark, 2000; Parnas, 1972; Simon, 1996). One way to describe a system is through its architecture. Brooks (1975) divides the design of software systems into components and architecture. The components are the parts of a system that the software developers implement in computer code; architecture describes the overall form of the system.

In this paper, software portfolio architecture is a description of *components* and *dependencies*. In the most general sense, a **dependency** exists if one component has knowledge of another such that a change in one could affect the other (Object Management Group Inc., 2007). This abstract representation of software portfolio architecture hides many specific details including the different types of components used and the different mechanisms by which they are dependent on each other.

In this paper we will represent the architecture of a software portfolio using a network diagram in which nodes represent components and lines between pairs of nodes represent dependencies. Such structural views of architecture are common in research linking architecture to performance in other contexts (Baldwin and Clark, 2000; MacCormack et al., 2006).

Software portfolio flexibility

Based on Upton (1994), this paper defines software portfolio flexibility *as the property of a software portfolio that enables it to change or react with little penalty in time, effort, cost, or performance.* This definition retains the abstract generality of Upton's definition with an important clarification: flexibility is a property of a system. Such a definition bounds the unit of analysis; in this research, the system is a software portfolio. Additionally, this definition excludes concepts such as organizational slack, skills, or other resources because such concepts refer to the ability of the organization to change a system, not the system itself (Galbraith, 1973).

Complexity

To explore the relationship between software portfolio architecture and software portfolio flexibility this research adopts the perspective taken in the modularity and software maintenance literatures in which the structural qualities of an architecture are measured and the relationships between these measures and flexibility are tested (Darcy et al., 2005; MacCormack et al., 2006). This paper focuses on one structural quality of architecture: complexity.

Simon (1996) identifies a complex system as one whose behavior cannot be fully known based upon an understanding of its components. Baldwin and Clark (2000) provide a basis for a mathematical representation of complexity. They define each choice made by a designer as specifying a design parameter, the smallest unit of a design. This approach suggests that an incomplete understanding of interactions among a system's components results from an incomplete understanding of interactions among the design parameters. The bigger the software portfolio, the greater the number of choices made within each component, and the greater the potential number of interactions. These interactions complicate designers' task performance (Darcy et al., 2005; Wood, 1986) and affect the difficulty in finding high value designs (Alexander, 1964; Levinthal, 1997), which may collectively make modifying a software portfolio difficult.

We can measure complexity at both the level of the architecture as a whole and for an individual component:

Complexity in software is generally defined in terms of two measurement categories: cohesion and coupling (Stevens, Myers and Constantine, 1974). *Cohesion* is a measure of the relatedness of the design parameters within a component. *Coupling* is a measure of the extent that one component depends on another. This paper develops *architectural complexity* measures based on cohesion and coupling among components. Our conjecture is that these measures, traditionally employed within an application, can yield insight when applied at the software portfolio level.

The software literature has defined additional complexity measures: data, structural, and procedural complexity (Tegarden, Sheetz and Monarchi, 1995; Zuse, 1991). This paper develops *component complexity* measures based on these three measures as defined in Tegarden et al. (1995): **Structural complexity** is a measure of the intra-component coupling of the modules that comprise the component (Henry and Kafura, 1981). **Procedural complexity** is a measure of the number of design parameters in the component and the interactions among them (Tegarden et al., 1995; Zuse, 1991). **Data complexity** is a measure of complexity of the component's data model (Banker and Slaughter, 2000; Card and Agresti, 1988).

The relationship between software portfolio architecture and software portfolio flexibility

Changing a software portfolio involves identifying a gap between current and desired performance, identifying potential choices, making decisions, and evaluating outcomes. Complexity impinges on the designers who change a software portolio in two ways: first, complexity increases the cognitive demands on the designer (Anderson, Reder and Lebiere, 1996; Banker and Slaughter, 2000; Miller, 1956; Wood, 1986). Second, complexity increases the difficulty of finding a fit between design and context (Alexander, 1964; Baldwin and Clark, 2000; Levinthal, 1997; Simon, 1996).

The general argument in this paper is that components vary in their dependency on the other components in the software portfolio and that these differences in dependency result in differences in both types of complexity, which cause differences in architectural flexibility. Complexity increases the cognitive efforts required of those making changes to the components and increases the difficulty of finding high-value modifications.

HYPOTHESES

Based upon the preceding theory, we propose five hypotheses that relate complexity to flexibility.

Component complexity

Modifying the software portfolio by deploying, upgrading, replacing, or decommissioning a component requires the designer to have some understanding of the component. This understanding is required to select the right component, configure or change it so that it fits the organization's requirements, or find suitable alternatives. Component complexity complicates the effort of achieving the necessary understanding and either reduces the probability that the designer's understanding will lead to successfully meeting organizational expectations or increases the probability that the designer will need to make multiple attempts in order to meet organizational expectations (Darcy et al., 2005; Fleming and Sorenson, 2001; Levinthal, 1997; Wood, 1986). There are three hypotheses associated with this argument based on the dimensions of component complexity identified above:

H1: Increased structural complexity is associated with decreased flexibility.

H2: Increased procedural complexity is associated with decreased flexibility.

H3: Increased data complexity is associated with decreased flexibility.

Architectural complexity

Two hypotheses, based on component coupling and cohesion, reflect the effect of increased architectural complexity on software portfolio flexibility. Component coupling is a measure of the degree of dependency between a particular component and all other components in the software portfolio. As coupling increases, the number of other components the designer must consider and the number of potential interactions among those components increases. As a result, increases in component coupling impinge on the designers' cognitive limits and increase the difficulty of finding a fit between design and context. Both of these factors combine to reduce the probability of a good outcome.

Component cohesion is a measure of the degree of dependency among the components that a particular component (the "focal" component) is *directly* dependent on. It is a measure of the degree of dependency between a focal component, the components it is connected to (its neighbors), and the dependencies among the neighbors. Cohesion is a measure of the direct and indirect dependencies among the immediate neighbors of the focal component.

The more dependencies in the neighborhood of a focal component, the more interactions among the design parameters of the interconnected components and the more understanding required by a designer to make a change to the focal component. The effort of achieving the necessary understanding either reduces the probability that the designer's understanding will lead to successfully meeting organizational expectations or increases the probability that the designer will need to make multiple attempts in order to meet organizational expectations (Darcy et al., 2005; Fleming and Sorenson, 2001; Levinthal, 1997; Wood, 1986).

Thus,

H4: Increased component coupling is associated with decreased flexibility.

H5: Increased component cohesion is associated with decreased flexibility.

METHODS

Data collection site

Data to test the theory describing the relationship between software portfolio architecture and software portfolio flexibility were collected from a research division of BPC (a pseudonym), a biopharmaceutical company. The individuals responsible for the components are called IT Service Owners; they helped collect data about the components. The IT Service Owners provide project management, systems analysis, and some limited programming services to the organization.

BPC is particularly well-suited to the investigation of a software portfolio architecture that has grown, in the words of the IT director, "organically." Modifications to the software portfolio have followed the needs of the users, rather than prescriptions set forth in a designed, normative architecture. This is representative of organizations in which incremental changes are made to an existing software portfolio as business requirements and technologies change.

Data collection methods

Data were collected by examining strategy documents, having IT Service Owners enter architectural information into a repository, using automated system scanning techniques, and executing a survey instrument.

Site's software portfolio architecture

The site's software portfolio consists of 147 components and 388 dependencies. Components are software assets that BPC can treat independently of other such components and over which BPC can exercise architectural control (Jensen and Meckling, 1992). Of the 147 components, six were removed from analysis because they were deemed by BPC as non-material. The remaining 141 components have 258 component-to-component dependencies.

Of the 141 components, 119 were assigned to IT Service Owners. There are sufficient attribute data for 78 of the 119 "owned" components to construct the independent variables. Of these 78 components, 62 have a completed survey (described next) in which the IT Service Owner claimed enough knowledge of the component to give a meaningful response.

Operationalization of the dependent variable

We operationalize architectural flexibility by measuring the cost of making an architectural change: the lower the cost the greater the flexibility.

Cost data were collected through a survey administered to the IT Service Owners. Drawing upon the work of Baldwin and Clark (2000), and in consultation with BPC, we define the architectural operations that could be performed on a software portfolio component as *deploy, upgrade, replace, decommission,* and *integrate*. A component is *integrated* when modifications are made to it that enable it to 'talk' to another component, which creates a dependency.

All IT Service Owners with component knowledge (nine individuals) provided estimates of the cost of each architectural operation listed above for the 62 components that ultimately enter the regression model. Each owner supplied a response for two to 17 components. In order to minimize bias, respondents did not provide responses for all the components they were responsible for in a single survey session, and a respondent control variable is added to the regression model.

A single flexibility variable, COST, was created from the five flexibility survey measures by summing their values for each component. Cronbach's alpha for COST is 0.83, summarized with the other variables in Table 1.

Operationalization of independent variables

The component and architectural complexity constructs were operationalized as follows:

Structural, procedural, and data complexity

In the software measurement literature, structural and procedural complexity are traditionally measured by analyzing source code. Source code is not available for the components at BPC. Therefore, structural and procedural complexity are measured through a factor analysis of the file types and sizes distributed with each component (Kim and Mueller, 1978; Tegarden et al., 1995; Zuse, 1991). Since the number of lines of source code is highly correlated with the sum of the sizes of the resulting compiled files, we measured the sum (in bytes) and number of files of each file type (java, html, object, executable). These measures created a profile of each application and factored into the complexity measures used in this paper.

Data complexity is measured by counting the database tables associated with each component under the assumption that the more tables accessed by a component, the more data, and relationships among the data, there are.

Coupling

Coupling is operationalized through the network analysis measure **closeness centrality** (Borgatti and Everett, 2006; Wasserman and Faust, 1994) because it measures the effect of all components on any given component. We calculate closeness centrality for a component by summing the value of the reciprocal path values between it and each other component in the network. The path value is the sum of the log(size) of the components encountered along the path. Because we sum reciprocal paths, unreachable components are taken as contributing zero.

The use of the reciprocal results in very big components contributing less to the closeness measure than small components. The intuition for this is that an integration uses only a small part of a larger component's capabilities; the larger component provides interfaces or other technologies that make integration easier; and the larger component is more mature and, thus, doesn't need tight integrations.

Cohesion

Cohesion is operationalized through the network analysis measure **embeddedness** (Borgatti and Everett, 2006; Wasserman and Faust, 1994) because it measures the effect of a component's neighbors on each other. The *neighbors* of a component are those components connected to it with path of length one. Embeddedness is estimated by calculating the path density among the component's neighbors (Wasserman and Faust, 1994). Density is computed by dividing the total number of edges by the total number of pairs of components. The density is bound between 0 and 1. In the case of an isolated component or a component connected to only one other component the value is set to 0.

Controls

The statistical model controls for two sources of bias. Response bias is controlled for by including the amount of time, COMPTIME, the respondent has spent working with the component. The amount of time respondents have spent with their components potentially affects their component knowledge and the accuracy of their cost estimates.

The second source of bias is the components themselves. The component characteristics identified in Table 2 were controlled for to address the possibility that different types of components have different levels of flexibility unrelated to the complexity measures identified in this paper.

Variable	Description	N[a]	Range	Mean	Var.
COST	Compound flexibility measure	62	2.20-6.02	3.60	1.11
STRUCT_COMP	Structural complexity	78	-1.88-2.31	0.00	0.94
PROC_COMP	Procedural complexity	78	-1.59-1.25	0.00	0.96
DATA_COMP[b]	Data complexity	78	0-5.99	2.20	4.99
CLOSE_W	Closeness centrality. Arc value =log(size)	78	0-10.30	2.90	10.20
EMBED[c]	Embeddedness	78	0-0.69	0.14	0.05
COMPTIME[d]	Respondent's Time with component	62	1-3	2.00	0.689

[a] The number of components (rows) with valid data.

[b] The range, mean, and variance summary statistics include the 0 values.

[c] Logarithm transformed

[d] A value of 1 indicates less than one year, a value of 2 indicates more than one but less than five years, and a value of 3 indicates more than five years.

Table 1. Summary statistics for non-categorical variables

Name	Description
COMM	Component developed by vendor.
CLIENT	Component accessed by end-users.
COMP	Component oriented towards computation.
NTIER	Component has N-Tier architecture.
STATIC	Component is no longer being enhanced.

Table 2. Component categorical variables

OLS regression

COST is estimated with ordinary least squares (OLS) regression.

RESULTS

DV=COST	Model 1	Model 2	Model 3
COMPTIME	0.12	0.05	0.43*
COMM		-0.43	-0.41
CLIENT		-0.33	-0.40
COMP		-0.15	-0.15
NTIER		-0.05	-0.07
STATIC		-0.94*	-0.19
STRUCT_COMP			0.10
PROC_COMP			0.20
DATA_COMP			0.21**
CLOSE_W			0.10*
EMBED			0.49
Constant	3.36**	4.71**	2.54**
Observations	62	62	62
F-test	0.55	1.83	3.69***
R-squared	0.01	0.17	0.45
Adjusted-R2	-0.01	0.08	0.33
DF	1	6	11
* p<0.05, ** p<0.01, and *** p<0.001			

Table 3. OLS results for dependent variable COST

Table 3 shows model significance and coefficient estimates for the COST regression equation. The regression equation is built in stages in order to evaluate the additional explanatory power of the independent variables over base models that only include control variables. Model 1 consists of respondent controls only; Model 2 adds the component controls; and Model 3 adds the complexity measures used for hypothesis testing.

The difference in adjusted R-squared values and increases in F-test scores between Model 3 and the prior models suggests that the independent variables collectively have statistically significant explanatory power and support the general hypothesis that component and software portfolio architectural complexity predict software portfolio flexibility.

The respondent's time with the component, COMPTIME, is significant ($p<0.05$) and positively associated with COST; this suggests that as the IT Service Owners spend time with their components they more fully appreciate the difficulty in making changes to them. None of the component controls are significant.

STRUCT_COMP is used to test H1.Although the coefficient is in the expected direction, it is not significant; therefore, H1 is not supported.

PROC_COMP is used to test H2. Although the coefficient is in the expected direction, it is not significant; therefore, H2 is not supported.

DATA_COMP is used to test H3 It is significant and in the expected direction ($p<0.01$); therefore, H3 is supported. Data complexity, possibly because it is not hidden from the components that access the data, has a significant effect on flexibility.

CLOSE_W is used to test H4. It is significant and in the expected direction ($p<0.01$); therefore, H4 is supported. Closeness has a significant effect on flexibility. Flexibility is a function of how coupled a component is to all other components, not just to those that integrate with it directly.

EMBED is used to test H5. It is in the expected direction, but it is not significant; therefore, H5 is not supported.

CONCLUSION

In this paper we have made the case that there is insight to be gained by treating an organization's software portfolio as an architecture – documenting the components and dependencies as observed in that portfolio as it has come to exist in the organization. We have argued that the complexity of this architecture has an impact on the ease with which designers are able to modify that architecture.

Our analysis of the software portfolio architecture at BPC supports this conjecture. We saw statistical support for the hypothesis that increases in the data complexity of components leads to decreased architectural flexibility. We also found evidence that it is not just the properties of individual components but also the pattern of dependencies among those components that make a difference. Specifically we found that increased coupling among components correlates with decreased flexibility.

As noted in the introduction, this is an exploratory study and the results must be treated as such. In particular, as a single-site study, the results may be an artifact of the specific data collected at this site at this time. Clearly additional study across sites and industries will be necessary to determine whether the preliminary findings from BPC represent a generalizable relationship between architecture and flexibility. The contribution of this study is, first, to develop the theoretical and methodological basis for carrying out such investigations, and second to provide preliminary evidence that there may well be an important connection between the architecture of a software portfolio and the ability of the organization to adapt that portfolio in response to changing business conditions.

REFERENCES

1. Alexander, C. (1964) Notes on the synthesis of form, Harvard University Press, Cambridge, MA.
2. Allen, B. R. and Boynton, A. C. (1991) Information Architecture: In Search of Efficient Flexibility, *MIS Quarterly*, 15, 4, 435-445.
3. Anderson, J. R., Reder, L. M. and Lebiere, C. (1996) Working memory: Activation limitations on retrieval, *Cognitive Psychology*, 30, 3, 221-256.
4. Baldwin, C. Y. and Clark, K. B. (2000) Design Rules: The Power of Modularity, MIT Press, Cambridge, MA.
5. Banker, R. D., Davis, G. B. and Slaughter, S. A. (1998) Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study, *Management Science*, 44, 4, 433-450.
6. Banker, R. D. and Slaughter, S. A. (2000) The moderating effects of structure on volatility and complexity in software enhancement, *Information Systems Research*, 11, 3, 219-240.
7. Bass, L., Clements, P. and Kazman, R. (1998) Software Architecture in Practice, Addison-Wesley, New York.
8. Borgatti, S. P. and Everett, M. G. (2006) A graph-theoretic perspective on centrality, *Social Networks*, 28, 4, 466-484.
9. Broadbent, M. and Weill, P. (1997) Management by maxim: How business and IT managers can create IT infrastructures, *Sloan Management Review*, 38, 3, 77-92.

10. Brooks, F. P. (1975) The mythical man-month: essays on software engineering, Addison-Wesley Pub. Co., Reading, Mass.
11. 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.
12. Card, D. N. and Agresti, W. W. (1988) Measuring Software-Design Complexity, *Journal of Systems and Software*, 8, 3, 185-197.
13. Darcy, D. P., Kemerer, C. F., Slaughter, S. A. and Tomayko, J. E. (2005) The structural complexity of software: An experimental test, *IEEE Transactions on Software Engineering*, 31, 11, 982-995.
14. Duncan, N. B. (1995) Capturing flexibility of information technology infrastructure: A study of resource characteristics and their measure, *Journal of Management Information Systems*, 12, 2, 37-58.
15. Fleming, L. and Sorenson, O. (2001) The Dangers of Modularity, *Harvard Business Review*, 79, 8, 20-21.
16. Galbraith, J. R. (1973) Designing complex organizations, Addison-Wesley, Reading, Mass.
17. Henderson, J. C. and Venkatraman, N. (1993) Strategic alignment: Leveraging information technology for transforming organizations, *IBM Systems Journal*, 32, 1, 4-16.
18. Henderson, R. M. and Clark, K. B. (1990) Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms, *Administrative Science Quarterly*, 35, 1, 9-30.
19. Henry, S. and Kafura, D. (1981) Software Structure Metrics Based on Information Flow, *IEEE Transactions on Software Engineering*, SE-7, 5, 510-518.
20. Huber, G. P. (1990) A Theory of the Effects of Advanced Information Technologies on Organizational Design, Intelligence, and Decision Making, *Academy of Management Review*, 15, 1, 47-71.
21. Jensen, M. and Meckling, W. (1992) Knowledge, Control and Organizational Structure Parts I and II, in Lars Werin and Hans Hijkander (Eds.) *Contract Economics,* Basil Blackwell, Cambridge, MA, 251-274.
22. Kayworth, T. R., Chatterjee, D. and Sambamurthy, V. (2001) Theoretical Justification for IT Infrastructure Investments, *Information Resources Management Journal*, 14, 3, 5-14.
23. Kim, J.-O. and Mueller, C. W. (1978) Factor analysis: statistical methods and practical issues, Sage Publications, Beverly Hills, Calif.
24. Lawrence, P. R. and Lorsch, J. W. (1967) Organization and Environment: Managing Differentiation and Integration, Harvard Business School Press, Boston.
25. Levinthal, D. A. (1997) Adaptation on rugged landscapes, *Management Science*, 43, 7, 934-950.
26. MacCormack, A., Rusnak, J. and Baldwin, C. Y. (2006) Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code, *Management Science*, 52, 7, 1015-1030.
27. Mason, R. and Mitroff, I. (1973) A program for research on management information systems, *Management Science*, 19, 5, 475-487.
28. Miller, G. A. (1956) The Magical Number Seven, Plus or Minus Two: Some limits on our capacity for processing information, *The Psychological Review*, 63, 2, 81-97.
29. Nelson, K. M. and Ghods, M. (1999) Measuring technology flexibility, *European Journal of Information Systems*, 7, 4, 232-240.
30. Nezlek, G. S., Jain, H. K. and Nazareth, D. L. (1999) An integrated approach to enterprise computing architectures, *Communications of the ACM*, 42, 11, 82-90.
31. Object Management Group Inc. (2007) UML Superstructure Specification, v2.1.1.
32. 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.
33. Parnas, D. L. (1972) On the criteria to be used in decomposing systems into modules, *Communications of the ACM*, 15, 12, 1053-1058.
34. Perry, D. E. and Wolf, A. L. (1992) Foundations for the Study of Software Architecture, *ACM SIGSOFT Software Engineering Notes*, 17, 4, 50-52.
35. Ross, J. W., Weill, P. and Robertson, D. (2006) Enterprise architecture as strategy: creating a foundation for business execution, Harvard Business School Press, Boston.
36. 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.
37. Shibata, T., Yano, M. and Kodama, F. (2005) Empirical analysis of evolution of product architecture: Fanuc numerical controllers from 1962 to 1997, *Research Policy*, 34, 1, 13-31.
38. Simon, H. A. (1996) The Sciences of the Artificial (Edition 3), The MIT Press, Cambridge, MA.
39. Stevens, W. P., Myers, G. J. and Constantine, L. L. (1974) Structured Design, *IBM Systems Journal*, 13, 2, 115-139.
40. Tegarden, D. P., Sheetz, S. D. and Monarchi, D. E. (1995) A Software Complexity Model of Object-Oriented Systems, *Decision Support Systems*, 13, 3-4, 241-262.

41. Ulrich, K. (1995) The role of product architecture in the manufacturing firm, *Research Policy*, 24, 3, 419-440.
42. Upton, D. M. (1994) The management of manufacturing flexibility, *California Management Review*, 36, 2, 72-89.
43. Wasserman, S. and Faust, K. (1994) Social Network Analysis: Methods and Applications, Cambridge University Press, Cambridge.
44. Weill, P. and Broadbent, M. (1998) Leveraging the New Infrastructure: How Market Leaders Capitalize on Information Technology, Harvard Business School Press, Boston.
45. Wood, R. E. (1986) Task Complexity: Definition of the Construct, *Organizational Behavior and Human Decision Processes*, 37, 1, 60-82.
46. Zachman, J. A. (1987) A framework for information systems architecture, *IBM Systems Journal*, 26, 3, 276-293.
47. Zuse, H. (1991) Software complexity: measures and methods, Walter de Gruyter & Co., Hawthorne, NJ.