Research Policy 43 (2014) 1381–1397

Contents lists available at ScienceDirect

Research Policy

journal homepage: www.elsevier.com/locate/respol

Hidden structure: Using network methods to map system architecture

Carliss Baldwin, Alan MacCormack*, John Rusnak

Harvard Business School, Soldiers Field, Boston, MA 02163, United States

ARTICLE INFO

Article history:
Received 7 June 2010
Received in revised form 28 April 2014
Accepted 19 May 2014
Available online 21 June 2014

Keywords:
Product design
Architecture
Modularity
Software
Dominant designs

ABSTRACT

In this paper, we describe an operational methodology for characterizing the architecture of complex technical systems and demonstrate its application to a large sample of software releases. Our methodology is based upon directed network graphs, which allows us to identify all of the direct and indirect linkages between the components in a system. We use this approach to define three fundamental architectural patterns, which we label core–periphery, multi-core, and hierarchical. Applying our methodology to a sample of 1286 software releases from 17 applications, we find that the majority of releases possess a "core–periphery" structure. This architecture is 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 in the system. We show that the size of the Core varies widely, even for systems that perform the same function. These differences appear to be associated with different models of development – open, distributed organizations develop systems with smaller Cores, while closed, co-located organizations develop systems with larger Cores. Our findings establish some "stylized facts" about the fine-grained structure of large, real-world technical systems, serving as a point of departure for future empirical work.

1. Introduction

All complex systems can be described in terms of their architecture, that is, as a hierarchy of subsystems that in turn have their own subsystems (Simon, 1962). Critically, however, not all subsystems in an architecture are of equal importance. In particular, some subsystems are "core" to system performance, whereas others are only "peripheral" (Tushman and Rosenkopf, 1992). Core subsystems have been defined as those that are tightly coupled to other subsystems, whereas peripheral subsystems tend to possess only loose connections to other subsystems (Tushman and Murmann, 1998). Studies of technological innovation consistently show that major changes in core subsystems as well as their linkages to other parts of the system can have a significant impact on firm performance as well as industry structure (Henderson and Clark, 1990; Christensen, 1997; Baldwin and Clark, 2000). And yet, despite this wealth of research highlighting the importance of understanding system architecture, there is little empirical evidence on the actual architectural patterns observed across large numbers of real world systems.

In this paper, we propose a method for analyzing the design of complex technical systems and apply it to a large (though non-random) sample of systems in the software industry. Our objective is to understand the extent to which such systems possess a "core–periphery" structure, as well as the degree of heterogeneity within and across system architectures. We also seek to examine how systems evolve over time, since prior work has shown that significant changes in architecture can create major challenges for firms and precipitate changes in industry structure (Henderson and Clark, 1990; Tushman and Rosenkopf, 1992; Tushman and Murmann, 1998; Baldwin and Clark, 2000; Fixson and Park, 2008).

The paper makes a distinct contribution to the literatures of technology management and system design and analysis. In particular, we first describe an operational methodology based on network graphs that can be used to characterize the architecture of large technical systems.[1] Our methodology addresses several weaknesses associated with prior analytical methods that have similar objectives. Specifically, (i) it focuses on *directed* graphs, disentangling differences in structure that stem from dependencies that flow in different directions; (ii) it captures all of the direct and *indirect* dependencies among the components in a system, developing measures of system structure and a classification typology

* Corresponding author. Tel.: +1 6174956856.
E-mail addresses: cbaldwin@hbs.edu (C. Baldwin), amaccormack@hbs.edu (A. MacCormack), jrusnak@alum.mit.edu (J. Rusnak).

[1] We define a large system as one having in excess of 300 interacting elements or components.

http://dx.doi.org/10.1016/j.respol.2014.05.004
0048-7333/

that depend critically on the indirect linkages, and (iii) it provides a heuristic for rearranging the elements in a system, in a way that helps to visualize the system architecture and reveals its "hidden structure" (in contrast, for example, to social network methods, which tend to yield visual representations that are hard to comprehend).

We demonstrate the application of our methodology on a sample of 1286 software releases from 17 distinct systems. We find that the majority of these releases possess a core–periphery architecture using our classification scheme (described below). However, the size of the Core (defined as the percentage of components in the largest cyclic group) varies widely, even for systems that perform the same function. These differences appear to be associated with different models of development – open, distributed organizations develop systems with smaller Cores, whereas closed, co-located organizations tend to develop systems with larger Cores. We find the Core components in a system are often dispersed across different modules rather than being concentrated in one or two, making their detection and management difficult for the system architect. Finally, we demonstrate that technical systems evolve in different ways: some are subject to continuous change, while others display discrete jumps. Our findings establish some early "stylized facts" about the fine-grained structure of large, real-world technical systems.

The paper is organized as follows. Next, we review the relevant literature on dominant designs, core–periphery architectures, and network methods for characterizing architecture. Following that, we describe our methodology for analyzing and classifying architectures based upon the level of direct and indirect coupling between elements. We then describe the results of applying our methodology to a sample of real world software systems. We conclude by describing the limitations of our method, discussing the implications of our findings for scholars and managers, and identifying questions that merit further attention in future.

2. Literature review

In his seminal paper "The Architecture of Complexity," Herbert Simon argued that the architecture of a system, that is, the way the components fit together and interact, is the primary determinant of the system's ability to adapt to environmental shocks and to evolve toward higher levels of functionality (Simon, 1962). However, Simon and others presumed (perhaps implicitly) that the architecture of a complex system would be easily discernible. Unfortunately this is not always the case. Especially in non-physical systems, such as software and services, the structure that appears on the surface and the "hidden" structure that affects adaptation and evolvability may be very different.

2.1. Design decisions, design hierarchies and design cycles

The design of a complex technological system (a product or process) has been shown to comprise a nested hierarchy of design *decisions* (Marple, 1961; Alexander, 1964; Clark, 1985). Decisions made at higher levels of the hierarchy set the agenda (or technical trajectory) for problems that must be solved at lower levels of the hierarchy (Dosi, 1982). These higher-level decisions influence many subsequent design choices, hence are referred to as "core concepts." For example, in developing a new automobile, the choice between an internal combustion engine and electric propulsion represents a core concept that will influence many subsequent decisions about the design. In contrast, the choice of leather versus upholstered seats typically has little bearing on important system-level choices, hence can be viewed as peripheral.

A variety of studies show that a particular set of core concepts can become embedded in an industry, becoming a "dominant design" that sets the agenda for subsequent technical progress (Utterback, 1996; Utterback and Suarez, 1991; Suarez and Utterback, 1995). Dominant designs have been observed in many industries, including typewriters, automobiles and televisions (Utterback and Suarez, 1991). Their emergence is associated with periods of industry consolidation, in which firms pursuing non-dominant designs fail, while those producing superior variants of the dominant design experience increased market share and profits. However, the concept has proved difficult to pin down empirically. Scholars differ on what constitutes a dominant design and whether this phenomenon is an antecedent or a consequence of changing industry structure (Klepper, 1996; Tushman and Murmann, 1998; Murmann and Frenken, 2006).

Murmann and Frenken (2006) suggest that the concept of dominant design can be made more concrete by classifying components (and decisions) according to their "pleiotropy." By definition, high-pleiotropy components cannot be changed without inducing widespread changes throughout the system, some of which may hamper performance or even cause the system to fail. For this reason, the authors argue, the designs of high-pleiotropy components are likely to remain unchanged for long periods of time: such stability is the defining property of a dominant design. The authors proceed to label high-pleiotropy components as the "core" of the system, and other components as the "periphery."

Ultimately, dominant design theory argues that the *hierarchy* of design decisions (and the components that embody those decisions) is a critical dimension for assessing system architecture. At the top of the design hierarchy are components whose properties cannot change without requiring changes in many other parts of the system; at the bottom are components that do not trigger widespread or cascading changes. Thus any methodology for discovering the hidden structure of a complex system must reveal something about the hierarchy of components and related design decisions.

In contrast to dominant design theory, where design decisions are hierarchically ordered, some design decisions may be mutually interdependent. For example, if components *A*, *B*, *C*, and *D* must all fit into a limited space, then any increase in the dimensions of one reduces the space available to the others. The designers of such components are in a state of "reciprocal interdependence" (Thompson, 1967). If they make their initial choices independently, then those decisions must be communicated to the other designers, who may need to change their own original choices. This second-round of decisions, in turn, may trigger a third set of changes, with the process continuing until the designers converge on a set of decisions that satisfies the global constraint. Reciprocal interdependency thus gives rise to feedback and cycling in a design process. Such cycles are a major cause of rework, delay, and cost overruns (Steward, 1981; Eppinger et al., 1994; Sosa et al., 2013). Thus any methodology for discovering the hidden structure of a complex system must reveal not only the *hierarchy* of components and related design decisions but also the presence of *reciprocal interdependence or "cycles"* between them.

2.2. Network methods for characterizing system design

Studies that attempt to characterize the architecture of complex systems often employ network representations and metrics (Holland, 1992; Kauffman, 1993; Rivkin, 2000; Braha et al., 2006; Rivkin and Siggelkow, 2007; Barabasi, 2009). Specifically, they focus on identifying the linkages that exist between the different elements (nodes) in a system (Simon, 1962; Alexander, 1964). A key concept in this work is that of modularity, which refers to the way that a system's architecture is decomposed into different parts or