8-2014

Managing Complexity Through Selective Decoupling

C. Jason WOODARD

Singapore Management University, jason.woodard@olin.edu

Eric K. CLEMONS

Managing Complexity Through Selective Decoupling

C. Jason Woodard
School of Information Systems
Singapore Management University
jwoodard@smu.edu.sg

Eric K. Clemons
The Wharton School
University of Pennsylvania
clemons@wharton.upenn.edu

DRAFT of 16 January 2014
(please do not quote or cite)

ABSTRACT

Designers of complex systems are often confounded by the tendency for design changes to increase performance on some dimensions while decreasing it on others. While adopting a more modular architecture may temper these opposing effects, modularization can also deprive designers of opportunities to harness complementarities among system elements. This paper explores this tension using an NK model in which product designers can modify the structure of their fitness landscapes by suppressing or restoring interactions between components. We find that these changes can lead to improved performance by "flattening" harmful interactions that would otherwise cause search efforts to become trapped on local fitness peaks. Despite the fact that this process, which we call *selective decoupling*, tends to lower the maximum performance of a given component, it may increase the maximum attainable product performance. Moreover, selective decoupling can trigger the restoration of previously suppressed interactions, enabling the discovery of integral designs that were inaccessible to local search. We discuss the similarity between this pattern of endogenous architectural change and the stylized product life cycle described by the technology and innovation management literature, and suggest that the forces highlighted by our model might play an underappreciated role in technology and industry evolution.

Keywords:

Technology evolution, product architecture, NK models

Managing Complexity Through Selective Decoupling

> *The original Mac really had no operating system. People keep saying, "Well why didn't we license the operating system?" The simple answer is that there wasn't one. It was all done with lots of tricks with hardware and software. Microprocessors in those days were so weak compared to what we had today. In order to do graphics on a screen you had to consume all of the power of the processor. Then you had to glue chips all around it to enable you to offload other functions. Then you had to put what are called "calls to ROM." There were 400 calls to ROM, which were all the little subroutines that had to be offloaded into the ROM because there was no way you could run these in real time. All these things were neatly held together.*
>
> — John Sculley (Apple CEO, 1983–93)[1]

By the time Apple's Macintosh was released in 1984, the modular architecture of the IBM PC was firmly established. Clean interfaces separated hardware from software, allowing the PC to run multiple operating systems and support a rapidly growing catalog of third-party add-on products. But the PC is an exceptional case, even within the computer industry. Few other category-defining computer products were "born" modular. The first commercially successful mainframes and minicomputers (IBM's 704 and DEC's PDP-8) did not initially ship with operating systems, which led software written for them to be tightly coupled to a specific hardware model or even the particular configuration of a single machine. Modular architectures took time to evolve in other IT-related product categories as well, including disk drives, printers, networking equipment, videogame consoles, handheld devices, and mobile phones.

[1] Interview with Leander Kahney, October 2010. Available at http://www.cultofmac.com/63295/john-sculley-on-steve-jobs-the-full-interview-transcript (accessed 10 January 2014).

Baldwin and Clark (1997: 86) explain why: "It turns out that modular systems are much more difficult to design than comparable interconnected systems. The designers of modular systems must know a great deal about the inner workings of the overall product or process in order to develop the visible design rules necessary to make the modules function as a whole. They have to specify those rules in advance. And while designs at the modular level are proceeding independently, it may seem that all is going well; problems with incomplete or imperfect modularization tend to appear only when the modules come together and work poorly as an integrated whole."

The story of the Macintosh is thus more typical, both of computers and other kinds of complex engineered systems such as automobiles (Jacobides, MacDuffie, & Tae, 2013). The original Mac's hardware was modestly more powerful than the comparable generation of IBM PCs, but its graphical operating system made vastly higher demands on the processor than typical PC software. Constrained by Steve Jobs's goal of making the Mac both small and affordable, the Macintosh team chose to satisfy those demands by adopting an integral design in which the hardware and software components were intricately linked. As these constraints receded under Moore's Law, the Macintosh platform evolved toward a more conventional modular architecture, culminating in a switch to industry-standard Intel processors in 2006. By eliminating interactions that had become unnecessary and adopting a common set of design rules, Apple gained the ability to optimize the design of the Mac's major subsystems independently of each other. For example, removing the cobbled-together circuitry described by John Sculley freed up space for more powerful hardware (e.g., dedicated graphics chips) and made it easier to modify software that had been burned into ROM. At the same time, Apple continued to develop proprietary innovations that created new interactions between product

components, such as high-resolution Retina displays. Thirty years later, the resulting architecture is an emergent hybrid of modularity and integrality that defies easy categorization.

This paper starts from the premise that the "natural state" of a new system architecture is to be highly interdependent, and explores how modularity can arise in a gradual way, without an explicit strategy or goal. We present a computational model in which designers can modify the architecture of their systems by suppressing or restoring interactions between components, similar to the way software engineers "refactor" large computer programs. The designers engage in an adaptive search process that alternates between efforts to improve the performance of a system within its current architecture and efforts to improve the architecture itself. We find that the ability to perform even an extremely limited form of refactoring (decoupling or recoupling a single pair of components at a time) can dramatically improve designers' ability to create high-performance system designs.

The contribution of the paper is to unpack this finding and examine its implications for technology and industry evolution. Our model is a member of the well-studied NK family of fitness landscape models (Kauffman, 1993), which allows us to build on a large body of prior work. To model endogenous architectural change, we employ a technique used by Gavetti and Levinthal (2000) to study differences in the way agents represent a complex fitness landscape as they engage in search behavior. The main source of novelty in our model is that instead of treating these representations as cognitive artifacts—subjective ways of simplifying the world—we treat them as architectural choices that reshape the agents' real fitness landscapes.

The key insight that drives our results is that these choices affect both the ease of navigating a fitness landscape *and* the height of its peaks. As is well known, less coupled landscapes are easier to navigate through "hill-climbing" adaptive search; agents are less likely

to get "stuck" on an inferior local peak while searching for a global optimum (Rivkin & Siggelkow, 2002). But decoupling a pair of components "flattens out" both the peaks and the valleys corresponding to high- and low-performing configurations of those components. The intuition behind this insight is reflected in the familiar complaint that technology standards foster "lowest common denominator" solutions. Standards enable compatibility by ensuring that differences among implementations do not impair their ability to interoperate with each other (i.e., create valleys in their joint fitness landscape). By the same token, standard-compliant implementations are often forbidden to exploit design information that is supposed to be hidden by the standard, even when the use of such information would deliver higher performance or additional functionality (i.e., peaks in the landscape).[2] Refactoring an architecture to decouple one component from another is thus a double-edged sword, leading us to ask when and why it is more effective than simply accepting an existing architecture and trying to optimize the configuration of its components.

The remainder of the paper presents our model and a set of preliminary simulation results that address these questions. Our simulations consider boundedly rational product designers that are limited to incremental hill-climbing behavior; they explore new designs by modifying one product component at a time. Faced with a highly interdependent architecture, they are typically able to take only a few exploratory steps before getting stuck on a local fitness peak. We enable these designers to suppress or restore component interactions through refactoring, which allows them to employ a combination of modular (component-level) and architectural (interaction-level) search. After performing a randomly chosen refactoring step, the current product design is tested

[2] See Garud, Jain, & Kumaraswamy (2002) and Woodard & West (2011) on the conflict between Microsoft and Sun Microsystems over Microsoft's platform-specific extensions to Sun's Java technology.

by executing a sequence of hill-climbing component changes. If the fitness of the resulting design exceeds that of the design prior to refactoring, the modified architecture is kept and the process continues; otherwise it is rejected and another refactoring is tried.

Our main result is that this process tends to yield what we call *selective decoupling*, a progression from tight integration toward more loosely coupled systems (cf. Weick, 1976). Designers tend to make their landscapes flatter over time, yet they are able to discover superior product designs despite the fact that suppressing a component's interactions tends to lower the maximum fitness contribution of that component. The trend is not always monotonic, however. We frequently observed cases in which designers achieved improved performance after restoring a previously suppressed interaction (i.e., recoupling a pair of components through refactoring), especially after repeated episodes of decoupling. This pattern of decoupling followed by recoupling can enable the discovery of integral product designs that are inaccessible to local hill-climbing within a fixed architecture. In other words, a fairly "dumb" search process in the space of product architectures can substitute for "smarter" search processes at the component level.

The endogenous shifts we observed from high to low to intermediate levels of coupling bear a strong resemblance to the stylized dynamics of product architecture described by the literature on technology and industry life cycles (Anderson & Tushman, 1990, Murmann & Frenken, 2006). While the present model is too simple to investigate this connection in a satisfying way, it may be able to serve as a building block for efforts to study technology and industry evolution using computational modeling techniques. We return to this idea in the discussion section of the paper.

BACKGROUND

The relationship between the structure of complex interactions and the performance of boundedly rational search has been a recurring theme in the management literature over the last two decades, spanning the fields of strategy, organization design, and product development. The NK modeling framework has proven to be a versatile and effective tool to study this relationship, yielding a large body of rigorous and insightful research. This work typically takes the structure of interactions as given and explores the consequences of this structure (Rivkin & Siggelkow, 2003, 2007). In contrast, our model is most closely related to work that considers the *antecedents* of interaction structure or allows it to emerge endogenously.

The idea that interaction structures are the product of evolutionary forces has a long history, dating back at least to Herbert Simon's argument that "hierarchic systems will evolve far more quickly than non-hierarchic systems of comparable size" (1962: 468), and thus, "[a]mong possible complex forms, hierarchies are the ones that have time to evolve" (1962: 473). Kauffman and Johnsen (1991: 480) developed the NKC model to investigate "the possibility that a coevolving system of species may collectively tune the parameters governing its own coevolution," where these parameters include K (the degree of interdependence within a species) and C (the degree of coupling between species). More recently, biologists have employed the concept of a neutral network to describe "flat" (i.e., partially decoupled) regions in a fitness landscape; evidence from biological systems such as metabolic networks and regulatory circuits suggests that organisms have repeatedly evolved vast and intricate neutral networks, which have contributed to their robustness, evolvability, and capacity to produce innovations (Wagner, 2005, 2011). A few scholars have studied neutrality in technological and organizational

landscapes (Lobo, Miller, & Fontana, 2004; Jain & Kogut, 2013) but the topic remains largely unexplored in these settings.

A complementary idea, more germane to a managerial context than a biological one, is that interaction structures are the product of conscious human design. Levinthal and Warglien (1999: 343) articulated this idea and formulated it as a design problem:

> One may argue that the structure of interdependencies is given by the "world," and that a designer can at best modify the perception of actors of these inherent interdependencies by manipulating flows of information and patterns of communication—features on which traditional organization design focuses. We contend that while the external environment provides significant constraints, there are usually important degrees of freedom left to the designer. Indeed, a design problem arises *because* there are these degrees of freedom.

They illustrate these degrees of freedom using the example of designing a computer system (similar to our Mac vs. PC story), although their main interest is in organization design. They cite several mechanisms that affect the degree of interdependence in organizations (e.g., group size, communication patterns, and credit assignment schemes), drawing informally on the NK and NKC models for intuition. Gavetti and Levinthal (2000) shifted the design problem from the true structure of interactions to the agents' cognitive representation of those interactions. They "assume that cognitions are grounded on the actual landscape but that they constitute a simplified caricature of the decision context" (2000:121), which they operationalize by assuming that cognitive representations are of lower dimensionality than the actual landscape, and thus that "the organization in choosing a point within its representation is really choosing a region of the landscape" (2000:123). Ethiraj and Levinthal (2004) took the idea of a design problem in yet a different direction, focusing on the design of complex organizations and their task structures. Like Gavetti and Levinthal, they assume that a fitness landscape has a "true, latent structure" which may remain constant or change exogenously. However, their treatment of "second-order

adaptation" endows agents with an explicit set of organizational design operators (splitting, combining, and reallocation) while Gavetti and Levinthal allow only random shifts in an agent's cognitive representation.

Our model is a hybrid of these approaches. The way we implement component decoupling is mathematically identical to the way Gavetti and Levinthal (2000) reduce the dimensionality of a fitness landscape, but we follow Levinthal and Warglien (1999) in treating decoupling as an architectural choice that reshapes an agent's "real" landscape rather than a purely cognitive representation. While our treatment of design operators is not as extensive as Ethiraj and Levinthal (2004), we similarly distinguish between two levels of adaptive search that can be enabled or disabled independently to study their interactions. We were also influenced by Marengo, Dosi, Legrenzi, and Pasquali (2000) and subsequent work (Marengo & Dosi, 2005; Brusoni, Marengo, Prencipe, & Valente, 2007) which formalized the process of problem solving within the NK framework and tied it back to Simon's seminal observations about hierarchy and (near-)decomposability. In addition, we drew inspiration from Frenken (2006a, 2006b), who adapted the generalized NK model of Altenberg (1994) to the domain of complex technological systems. While we confine our attention to the standard NK model in this paper, our model and its software implementation are designed to be extended into the generalized NK setting.

MODEL

The model is general enough to encompass the design of a large class of technological and organizational systems, but for concreteness we focus on the context of product development (Wheelwright & Clark, 1992) and product architecture (Ulrich, 1995). The agents in the model, which we call designers, represent the individuals or teams responsible for both the overall

product architecture and the design or selection of the product's components. To keep our focus on the basic forces operating in the product domain, we set aside for now both the problems of collective action within product development teams and the opportunities for interaction between the designers of different products (cf. Marengo et al., 2000: 758).

Basic Setup

A *product design space* is defined by a set of N components, each of which is associated with a particular function. Each function, in turn, contributes value to the system as a whole. The function of a component can be affected positively or negatively by other components, in which case the components are said to interact. The pattern of interactions, which can be represented by a square binary *influence matrix* (Rivkin & Siggelkow, 2007) defines the product's architecture. Each component is available in two variants, which represent alternative designs (e.g., black-and-white vs. color screen). Both variants as assumed to interact with the same components, and a complete product contains exactly one variant of each component.[3] A vector of length N whose ith element represents a variant of the ith component is called a *component configuration*. As is standard practice in the NK literature, we denote the variants by 0 and 1, which allows a configuration to be represented by a binary string (e.g., 0010 for $N = 4$). Together, an influence matrix and a component configuration define a *product design*.

The fitness of a product design (or equivalently for our purposes, its value or performance) is given by a *fitness landscape*, which is a mapping from the set of 2^N possible component configurations to the real numbers. The structure of our fitness landscapes closely

[3] These simplifying assumptions are pervasive in the NK modeling literature, but they are not innocuous. We have experimented with less restrictive assumptions in other work (e.g., Woodard & Clemons, 2011, 2012).

follows the NK tradition. Let ($\boldsymbol{c}$, $\boldsymbol{D}$, $\mathbf{X}$) denote a product design with configuration $\boldsymbol{c}$, influence matrix $\boldsymbol{D}$, and fitness landscape $\mathbf{X}$. We stipulate that the ith component of $\boldsymbol{c}$ makes a fitness contribution, denoted x_i, that depends on the chosen variant of that component, denoted c_i, and any components that influence it, where the influencing components are indicated by the non-zero elements of the ith row of $\boldsymbol{D}$. Specifically, if a component is influenced by itself and k other components, then we associate each of the 2^{k+1} possible combinations of these variants with a fitness contribution drawn independently at random from a standard uniform distribution. The product's overall fitness, x, is then the average over the N component contributions.

Each simulation run follows the path of a single designer. This path is determined by three parameters: N, the initial coupling level (denoted K), and a random seed. The designer begins with a randomly chosen component configuration ($\boldsymbol{c_0}$), an initial influence matrix ($\boldsymbol{D_0}$) in which each component is influenced by itself and K others chosen at random, and an initial landscape ($\mathbf{X_0}$) with $N \cdot 2^{K+1}$ fitness contributions drawn at random as described above. Over a series of time periods, starting from $t = 1$, the designer attempts to improve the fitness of its design through two interleaved processes. The process of *modular search* operates by changing the design's component configuration, while *architectural search* operates by changing the influence matrix.[4] The remainder of this section describes these processes in more detail.

[4] We use these terms for consistency with the modularity literature (Henderson & Clark, 1990; Baldwin & Clark, 2000; Frenken, 2006), although from the perspective of this literature our products consist of a single module. Recall that our goal is to explore the conditions under which modularity can emerge, so it is important that we do not assume it at the outset. It is nonetheless perfectly natural to distinguish between "modular" changes (which variant of each component is chosen) and "architectural" ones (which components interact with which).

Modular Search

Given a product with configuration ***c*** and fitness x at time t, a modular search is conducted by taking a one-mutant adaptive walk to ***c'***, a local peak in the fitness landscape. In each step, the designer makes a list of neighboring configurations (those with a single variant "flipped" from 0 or 1 or vice versa) whose fitness values exceed x. These are called the adaptive neighbors. If there are no adaptive neighbors, then the walk ends and the simulation clock is advanced to $t + 1$. Otherwise the designer selects an adaptive neighbor at random and attempts another step from that configuration.

Architectural Search

In every time period after the first, modular search is preceded by modifying the current influence matrix, ***D***, to yield a permuted matrix, ***D'***, which is then used to generate a revised fitness landscape, **X'**. These steps are described below. Fitness values for the modular search procedure are computed using **X'**. If the modular search does not yield a fitness improvement (i.e., ***c'*** = ***c***), another iteration of permuting the influence matrix followed by modular search is undertaken until a fitness improvement occurs or there are no more permutations to try. If a fitness improvement occurs, another episode of architectural search is conducted by repeating the above procedure starting from (***c'***, ***D'***, **X'**), otherwise the simulation halts.

For each episode of architectural search, the list of permutations to try is constructed by generating a list of one-mutant neighbors of the current influence matrix, ***D,*** similar to the way neighboring configurations are generated for modular search. Here the neighbors are generated by "flipping" each off-diagonal element in turn, so for N components each influence matrix has $N(N-1)$ neighbors. If x_i is the lowest current fitness contribution, then permutations that affect

row *i* will be tried before those that affect the next lowest fitness contribution (row *j*, with contribution $x_j > x_i$), and so on. Neighbors that affect the same fitness component (i.e., the same row of ***D***) are tried in random order. These rules ensure that architectural search focuses on "bottleneck" components before those that are already performing at a high level.

To generate a revised fitness landscape from a permuted influence matrix (***D'***), we need to consider two cases. If the permutation decouples component *j* from component *i* (i.e., flips column *j* of row *i* from 1 to 0), then the number of distinct fitness contributions drops by half. In this case, each of the remaining fitness contributions is recomputed as the average of two existing values, namely the prior fitness contributions with component *j* set to 0 and 1. Consider a simple example for $N = 3$. If component 1 is influenced by components 2 and 3 but then component 3 is decoupled from it, the contribution of the configuration 00 in the revised landscape is the average of the prior contributions for 000 and 001. The contributions of 01, 10, and 11 are obtained through similar calculations. Decoupling thus "flattens" a fitness landscape by smoothing out each pair of differences between the two variants of the decoupled component.

Conversely, recoupling one component to another requires doubling the number of distinct fitness contributions. In this case, we refer to the initial fitness landscape ($\mathbf{X_0}$) for the needed values. However, if the respective components were uncoupled in the initial landscape, recoupling them has no effect. This rule ensures that architectural search cannot create new peaks or valleys out of thin air; it can only suppress or restore those provided by nature.

RESULTS

We implemented the model in Mathematica and ran a set of preliminary simulations whose results are reported here. We intend to implement several extensions to the model and run

more simulations to study them. Nonetheless, the preliminary results clearly illustrate the key features of the model and appear to be robust to a substantial range of parameter values. We ran simulations for $N = \{6, 8, 10\}$ and two values of K for each N: one yielding fully coupled initial landscapes ($K = N - 1$) and one yielding partly coupled initial landscapes ($K = N / 2 - 1$). To smooth out the effects of randomness in the landscapes and the designers' choices, we executed 10 searches for each of 10 landscapes for each (N, K) parameter pair, for a total of 100 runs per pair (600 overall).

To facilitate analysis, we recorded the complete search trajectory for each designer, as well as the fitness landscape at $t = 0$, $t = 1$, and after each episode of architectural search. The minimum duration of an architectural search was one period, which occurs when the first permutation that is tried results in a fitness-improving adaptive walk. The maximum duration was $N (N - 1)$ periods, which occurs when the last permutation in the list yields an improvement. (If it does not, the simulation halts without recording the episode.)

Architectural Search Yields Selective Decoupling

Before we examine the designers' search performance, let's see how the level of coupling changes over time. We define coupling as the average number of interactions per component (i.e., the number of non-zero off-diagonal elements in the influence matrix divided by N). Recall that higher coupling implies a more "rugged" fitness landscape with more local peaks.

Figure 1 provides two views of the coupling data for $N = 10$ and $K = 9$: the complete set of observations for all 100 runs, and connected trajectories for 10 randomly selected runs (one for each of the 10 random landscapes). Both views show a general downward trend: coupling tends to decrease over time as designers iteratively modify their product architectures and

component configurations. The connected trajectories show that the downward trend is not monotonic. Some successful episodes or architectural search result in decoupling, while others result in recoupling (i.e., restoration of previously suppressed interactions). One of the trajectories is plotted using a thicker stroke to highlight an especially pronounced upswing toward the end of the run. This was uncommon but not rare; 19 out of the 100 trajectories ended with an increase in coupling.

Insert Figure 1 about here

Should this result be surprising? After all, we started with maximal coupling (in this case, 9.0), so the only way to go is down. On the other hand, it is entirely possible for a trajectory not to go anywhere at all. Nothing guarantees that decoupling a single interaction will either directly improve the fitness of a design or enable modular search to reach a higher local peak, yet this happens consistently. (In fact, every designer succeeded in at least 10 consecutive episodes of architectural search; the maximum was 29.) Moreover, all of the (N, K) pairs exhibit a similar downward trend, even the partly coupled ones. So the result may be intuitive and expected, but it nonetheless seems nontrivial.

Selective Decoupling Facilitates the Discovery of Optimal System Designs

We now turn to the relationship between coupling and performance, as measured by product fitness. Since we recorded the fitness landscape after every episode of architectural search, we can compute the optimal fitness at each of these time periods and compare it to the actual fitness of the designer's product. (We do this by exhaustively evaluating all component

configurations and taking the maximum found; this is feasible for $2^{10} = 1024$ configurations under $N = 10$, but rapidly becomes infeasible as N increases.)

Figure 2 plots the actual and optimal fitness values for a typical run. Several features of the plot stand out. The actual fitness trajectory is monotonically increasing, which is necessarily the case since the simulation halts if a fitness-improving permutation cannot be found during architectural search. The optimal fitness trajectory is not monotonic, however. It is mostly—but not exclusively—decreasing toward the beginning of the run, then increasing toward the end. The trajectories converge (in this case, at period 84), then continue to increase in lockstep.

Insert Figure 2 about here

It turns out that all of these features are typical. It is especially notable that all of the trajectories for $N = 10$ and $K = 9$ converged to optimality, at a median period of 66 and a maximum period of 422. The convergence rate declined slightly for smaller design spaces and less initial coupling (to 95% for $N = 6$ and $K = 2$). The curvature of the optimal fitness trajectory is harder to generalize; not all runs exhibited a dip, but the majority did. In this case, the final optimal fitness is slightly higher than the initial optimal fitness (0.783 vs. 0.776). For $N = 10$ and $K = 9$ this is true on average (by 0.032), but 17% of runs ended at a lower optimal fitness than they started with.

Despite the variation in individual trajectories, the pattern illustrated by this figure is straightforward to characterize and interpret. There are typically two phases to the search: pre- and post-convergence. Before convergence, increases in actual fitness are driven mainly by architectural changes that make higher (but still suboptimal) fitness values easier to discover.

These changes tend to reduce optimal fitness by "flattening" peaks generated by certain combinations of highly coupled components, but that is irrelevant to the designer because these combinations were not accessible through the incremental hill-climbing process of modular search. After convergence, the only way to achieve continued improvement is by increasing both the actual and the optimal fitness at the same time. Sometimes this occurs through continued decoupling, which tends to lower the maximum fitness contribution of a given component, yet can increase the maximum attainable product fitness by removing harmful interactions between components. At other times, optimal fitness improvements occur by recoupling components to restore fitness peaks. Either way, the result demonstrates that a simple "one-bit" process of architectural search can substitute for a more sophisticated (or exhaustive) process at the component level.

Architectural Search Dominates Modular Search Within a Fixed Architecture

Figure 3 summarizes the performance data from all of the simulation runs in one graph. Six groups of three bars are shown. Each group aggregates the data from the 100 runs corresponding to a particular combination of N and K, as shown. The three partly coupled settings are on the left, while the fully coupled settings are on the right. Within each group, the bottom bar shows the average initial fitness of a product design (i.e., the fitness at $t = 0$). This is approximately 0.5 for all groups, as we expect since this is the average fitness of a randomly selected product. The middle bar shows the average fitness after one iteration of modular search but before architectural search has taken place (i.e., at $t = 1$). This would be the final fitness if architectural search were disabled, unless we implemented a different way for designers to get "unstuck" from local peaks. The values range from 0.572 (for $N = 10$, $K = 9$) to 0.601 (for $N = 8$,

$K = 3$) with no clear pattern among the groups. The top bar shows the average fitness after iterated architectural and modular search (the process illustrated in Figure 2). Here there is a pattern, although some of the differences are too small to see on the figure: the values increase with both N and K, from 0.756 (for $N = 6$, $K = 2$) to 0.811 (for $N = 10$, $K = 9$).

Insert Figure 3 about here

Again, we should ask whether these results are expected or counterintuitive. In light of Figure 2 and the analysis above, it is not surprising that architectural search (interleaved with modular search) consistently yields higher-performing product designs than modular search alone. It is interesting, however, that not only does the performance of architectural search not suffer as the fitness landscapes get larger and more rugged, but it actually increases. In retrospect this result makes sense because larger and more rugged landscapes have (on average) higher global optima than smaller and less rugged ones, by a basic property of maximum order statistics: the expected maximum over m draws from a random variable increases with m. Nonetheless, it remains a pleasant surprise to us that the ability to locate these optima does not diminish even as N and K increase.

DISCUSSION

The model we presented is almost embarrassingly simple, yet it appears to offer a novel answer to an important question: how can designers simplify a complex system in order to navigate its design space effectively? The validity of our answer ("selective decoupling") depends on at least three points that are not fully resolved in this preliminary version of the

paper. We highlight them here in the spirit of full disclosure, and to serve as input into the next iteration of our research.

First, it must actually be possible to execute the kind of refactoring that we posit. Can a designer always suppress a given dependency and "flatten" the corresponding region of the fitness landscape? Perhaps not. On the other hand, clearly these kinds of design moves are often possible—indeed, they are routine in software engineering—and in many cases they may be even less costly than we assume. In the NK tradition, our modeling choice to compute revised fitness values based on averages over the decoupled component reflects an assumption of ignorance (Kauffman, 1993:41). These values could also be drawn randomly, or made more pessimistic by offsetting them with a penalty; the results should be robust to a range of alternatives. Empirical evidence might also be brought to bear on the extent and cost of refactoring, for example using open-source software projects for which longitudinal dependency data is available (see, e.g., MacCormack, Baldwin, & Rusnak, 2012).

Second, our results need to scale to design spaces of more realistic size. Will we hit a "complexity catastrophe" where single-bit architectural changes no longer facilitate the discovery of optimal designs? Perhaps. On the other hand, the current results already span a substantial range of parameter values. Consider that $N = 6$ and $K = 2$ represents a design space with only 64 unique component configurations and typically only a few local optima, while the design space for $N = 10$ and $K = 9$ contains 1024 configurations (16 times more) and typically hundreds of local optima. Even more strikingly, consider the fact that the number of unique architectures (binary influence matrices) is $2^{N(N-1)}$, or 2^{30} for $N = 6$ (about a billion) vs. 2^{90} for $N = 10$ (more than the number of atoms in the observable universe). Fortunately, while we may not be able to compute global optima for N much larger than 10, we can certainly create

larger landscapes using the usual computational tricks employed by NK modelers (e.g., lazy or deterministic generation of fitness contributions).

Third, we need to embed our designs and designers in a richer environment. How well does refactoring work when multiple design teams are working in parallel at the component level? How do exogenous changes in a fitness landscape affect the efficacy of architectural search? Is architectural search fast enough to compete with designers employing more sophisticated forms of modular search? These questions can all be studied using established methods. Although they would no doubt increase the complexity of the model and its exposition, these extensions could also lead to additional insights.

More broadly, our aim is to study technology and industry evolution using computational modeling techniques. The initial motivation for this model came from the observation that biological systems exhibit vast and intricate neutral networks (Wagner, 2011), but we are not aware of a domain-independent model that explains how such networks could emerge in technological systems. The mechanism of selective decoupling seemed like a promising building block for such a model. Also, we are intrigued by the resemblance between the endogenous shifts we observed from high to low to intermediate levels of coupling (cf. Figure 2) and the stylized product life cycle described by the technology and innovation management literature (Anderson & Tushman, 1990; Murmann & Frenken, 2006). The emergence of a dominant design is surely a powerful driver of decoupling; perhaps these ideas could be linked.

CONCLUSION

In this paper, we presented a model in which product designers can modify the structure of their fitness landscapes by suppressing or restoring interactions between components. We find

that these changes can lead to improved performance by "flattening" harmful interactions that would otherwise cause search efforts to become trapped on local fitness peaks. Despite the fact that this process, which we call *selective coupling*, tends to lower the maximum performance of a given component, it may increase the maximum attainable product performance. Moreover, selective decoupling can trigger the restoration of previously suppressed interactions, enabling the discovery of integral designs that were inaccessible to local search. We hope to strengthen, deepen, and extend these findings in our ongoing work.

REFERENCES

Altenberg, L. 1994. Evolving better representations through selective genome growth. In ***Proceedings of the IEEE World Congress on Computational Intelligence,*** pt. 1, 182–187.

Anderson, P. & Tushman, M. L. 1990. Technological discontinuities and dominant designs: A cyclical model of technological change. ***Administrative Science Quarterly,*** 35(4): 604–633.

Baldwin, C. Y., & Clark, K. B. 1997. Managing in an age of modularity. ***Harvard Business Review,*** 75(5): 84–93.

Baldwin, C. Y., & Clark, K. B. 2000. ***Design Rules: The Power of Modularity,*** vol. 1. MIT Press.

Brusoni, S., Marengo, L., Prencipe, A., & Valente, M. 2007. The value and costs of modularity: A problem-solving perspective. ***European Management Review,*** 4(2): 121–132.

Ethiraj, S. K., & Levinthal, D. A. 2004. Bounded rationality and the search for organizational architecture: An evolutionary perspective on the design of organizations and their evolvability. ***Administrative Science Quarterly,*** 49(3): 404–437.

Frenken, K. 2006a. A fitness landscape approach to technological complexity, modularity, and vertical disintegration. ***Structural Change and Economic Dynamics,*** 17(3): 288–305.

Frenken. 2006b. ***Innovation, Evolution and Complexity Theory.*** Edward Elgar.

Jacobides, M. G., MacDuffie, J. P., & Tae, C. J. 2013. How agency and structure shaped value stasis in the automobile ecosystem. Working paper.

Kauffman, S. A. 1993. ***The Origins of Order: Self-Organization and Selection in Evolution.*** Oxford University Press.

Kauffman, S. A., & Johnson, S. 1991. Coevolution to the edge of chaos: Coupled fitness landscapes, poised states, and coevolutionary avalanches. ***Journal of Theoretical Biology,*** 149(4): 467–505.

Garud, R., Jain, S., & Kumaraswamy, A. 2002. Institutional entrepreneurship in the sponsorship of common technological standards: The case of Sun Microsystems and Java. ***Academy of Management Journal,*** 45(1): 196–214.

Gavetti, G., & Levinthal, D. 2000. Looking forward and looking backward: Cognitive and experiential search. ***Administrative Science Quarterly,*** 45(1): 113–137.

Henderson, R. M., & Clark, K. B. 1990. Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms. ***Administrative Science Quarterly,*** 35(1): 9–20.

Jain, A. & Kogut, B. 2013. Memory and organizational evolvability in a neutral landscape. ***Organization Science,*** published online ahead of print.

Levinthal, D. A. & Warglien, M. 1999. Landscape design: Designing for local action in complex worlds. ***Organization Science,*** 10(3): 342–357.

Lobo, J. Miller, J. H., & Fontana, W. 2004. Neutrality in technological landscapes. Working paper.

MacCormack, A., Baldwin, C., & Rusnak, J. 2012. Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis. ***Research Policy,*** 41(8): 1309–1324.

Marengo, L., & Dosi, G. 2005. Division of labor, organizational coordination and market mechanisms in collective problem-solving. ***Journal of Economic Behavior and Organization,*** 58(2): 303–326.

Marengo, L., Dosi, G., Legrenzi, P., & Pasquali, C., 2000. The structure of problem-solving knowledge and the structure of organizations. ***Industrial and Corporate Change,*** 9(4): 757–788.

Murmann, J. P., & Frenken, K. 2006. Toward a systematic framework for research on dominant designs, technological innovations, and industrial change. ***Research Policy,*** 35(7): 925–952.

Rivkin, J. W. & Siggelkow, N. 2002. Organizational sticking points on NK landscapes. ***Complexity,*** 7(5): 31–43.

Rivkin, J. W. & Siggelkow, N. 2003. Balancing search and stability: Interdependencies among elements of organizational design. ***Management Science,*** 49(3): 290–311.

Rivkin, J. W., & Siggelkow, N. 2007. Patterned interactions in complex systems: Implications for exploration. ***Management Science,*** 53(7): 1068–1085.

Simon, H. A. 1962. The architecture of complexity. ***Proceedings of the American Philosophical Society,*** 106(6): 467–482.

Ulrich, K. 1995. The role of product architecture in the manufacturing firm. ***Research Policy,*** 24(3): 419–440.

Wagner, A. 2005. ***Robustness and Evolvability in Living Systems.*** Princeton University Press.

Wagner. 2011. ***The Origins of Evolutionary Innovations.*** Oxford University Press.

Weick, K. E. 1976. Educational organizations as loosely coupled systems. ***Administrative Science Quarterly,*** 21(1): 1–19.

Wheelwright, S. C., & Clark, K. B. 1992. ***Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency, and Quality.*** Free Press.

Woodard, C. J., & Clemons, E. K. 2011. From primordial soup to platform-based competition: Exploring the emergence of products, systems, and platforms. ***Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS).*** Available at http://dx.doi.org/10.1109/HICSS.2011.208.

Woodard, C. J., & Clemons, E. K. 2012. Modeling technology evolution using generalized genotype-phenotype maps. ***Companion Proceedings of the 14th Annual Genetic and Evolutionary Computation Conference (GECCO 2012),*** 323–330. Available at http://dx.doi.org/10.1145/2330784.2330831.

Woodard, C. J., & West, J. 2011. Strategic responses to standardization: Embrace, extend, or extinguish? In G. Cattani, S. Ferriani, L. Frederiksen, & F. A. Taube (Eds.), ***Advances in Strategic Management,*** 28: 265–288. Available at http://dx.doi.org/10.1108/S0742-3322(2011)0000028014.

FIGURES

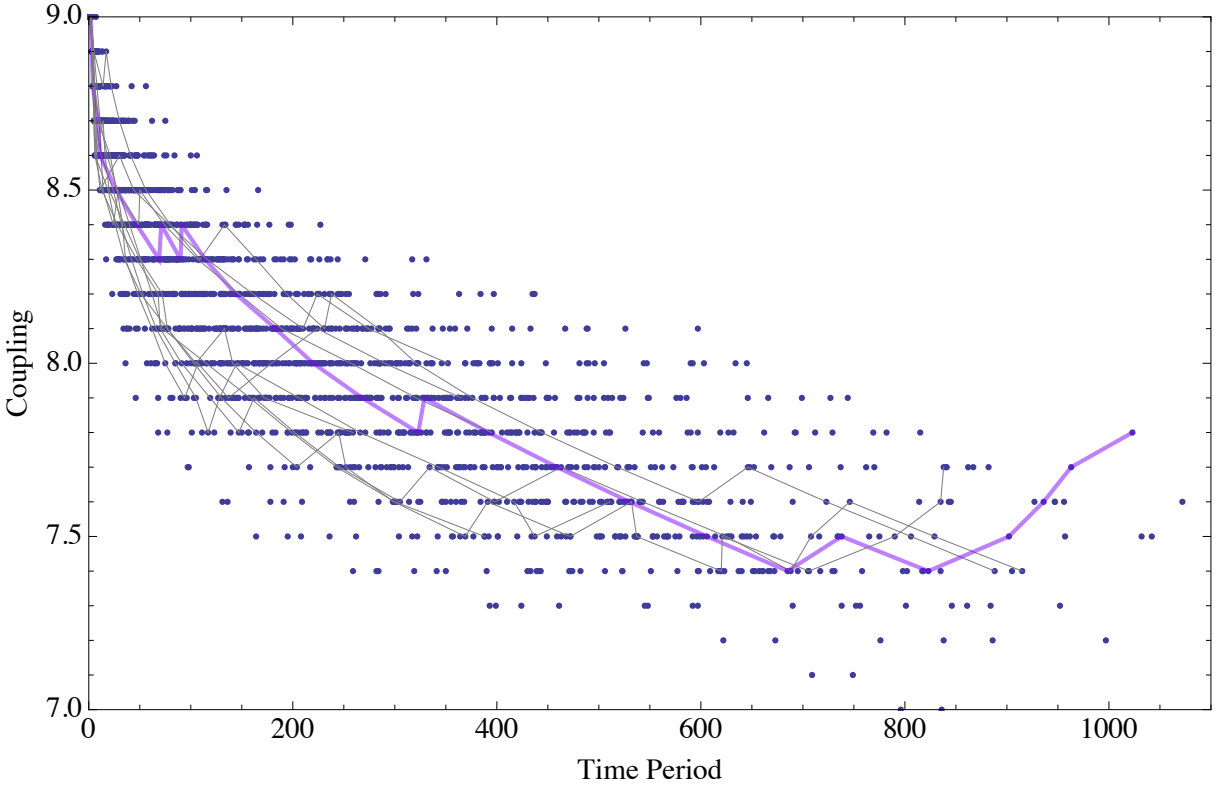

Figure 1: Coupling versus time under iterated architectural and modular search

(N = 10, K = 9; 10 out of 100 trajectories shown)

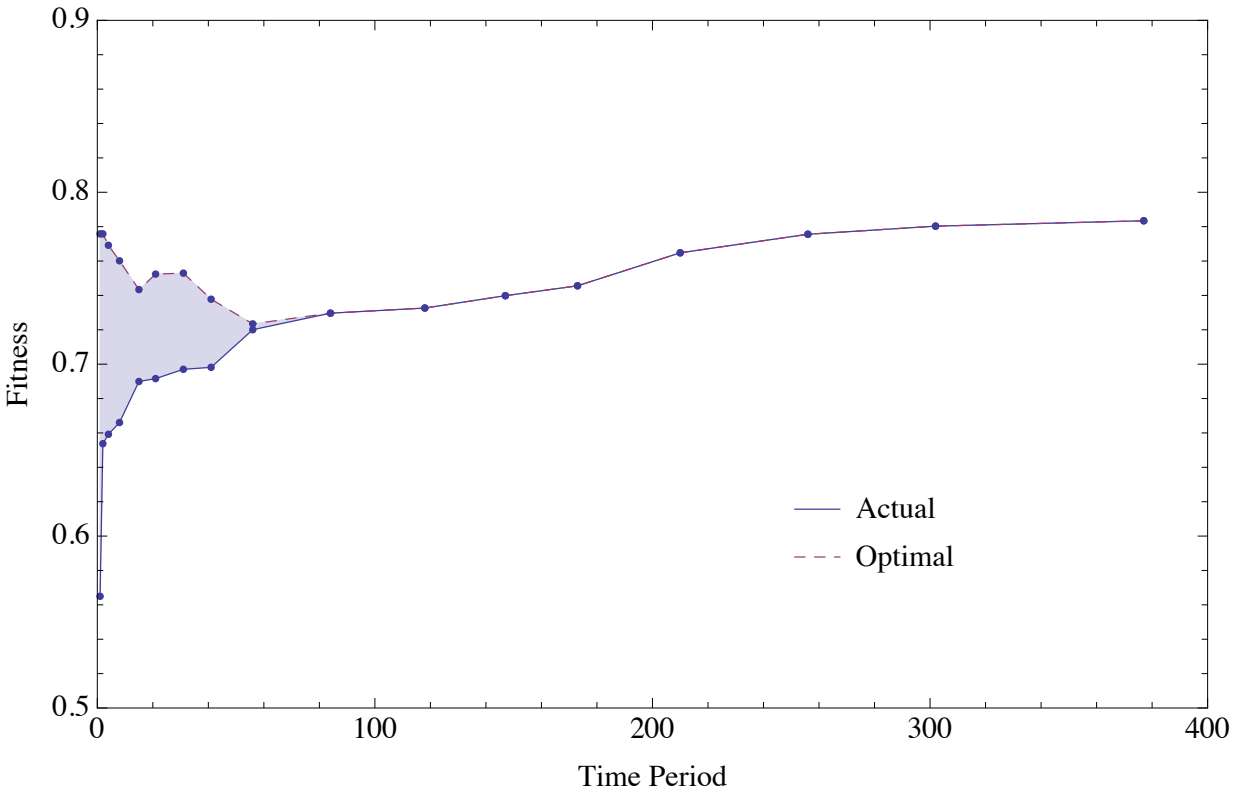

Figure 2: Actual and optimal fitness trajectories for a typical simulation run

(N = 10, K = 9; iterated architectural and modular search, run 26 out of 100)

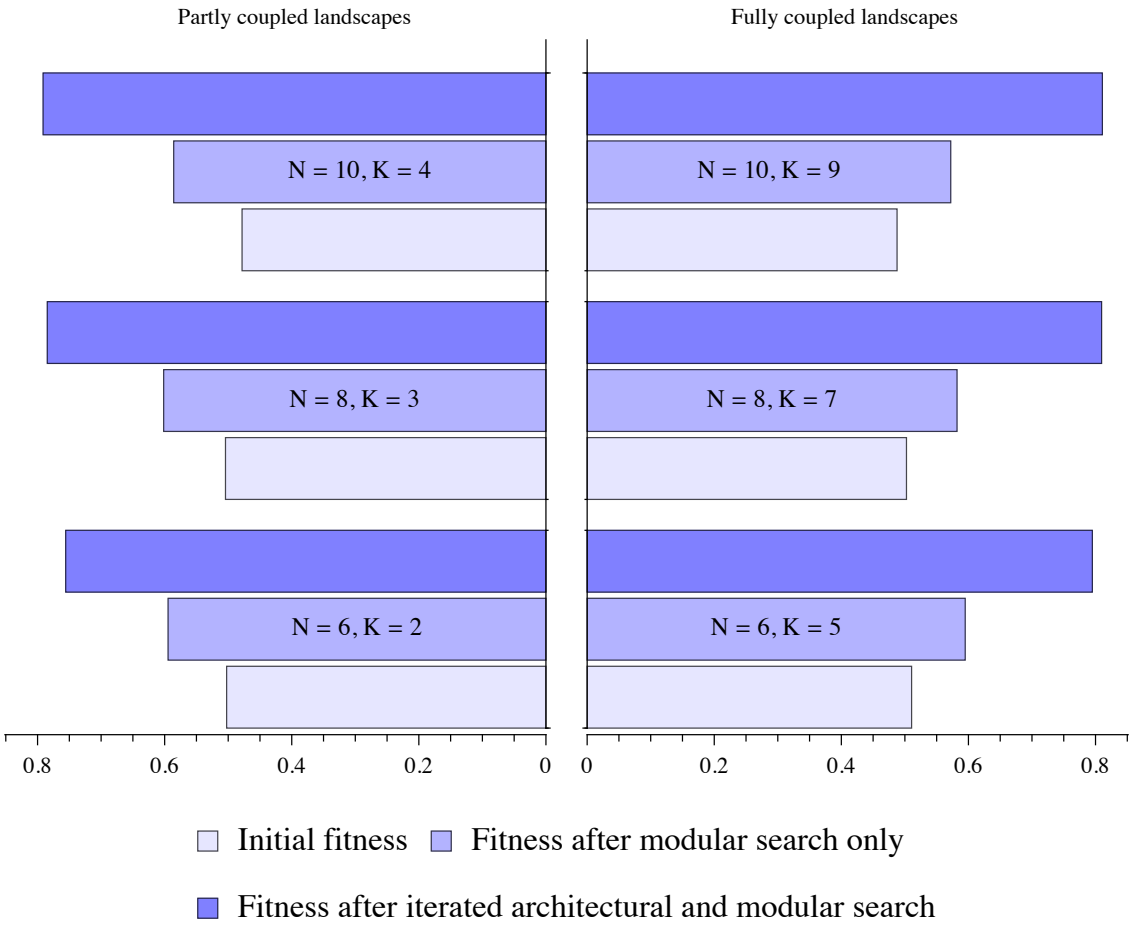

Figure 3: Comparison of average fitness by search procedure

(N = 6, 8, 10; partly and fully coupled landscapes)