ARCHITECTURE, INVESTMENT AND PERFORMANCE OF COMPLEX PRODUCT DEVELOPMENT

Ali A. Yassine[*]
Department of Industrial Engineering & Management
American University of Beirut
Beirut, Lebanon
Joe Naoum-Sawaya
Ivey Business School
Western University
London, Ontario, Canada

ABSTRACT

Firms engaging in product development (PD) face the imperative problem of allocating scarce development resources to a multitude of opportunities. In this paper, we propose a mathematical formulation to optimize PD investment decisions. The model maximizes the performance of a product under development, based on its architecture and the firm's available resource, by choosing the optimal resource allocation across product modules and design rules that govern the relationships between these modules. Results based on a comprehensive experiment (with various architectural patterns, escalating number of dependencies, and different problem sizes) shed light on three important hypotheses. First, product architecture affects resource allocation decisions and ultimately product performance. The second hypothesis tests whether modular or integral architectures can attain higher performance levels based on our formulation. A third hypothesis states that there is a shift in the temporal allocation of resources from design rules to individual modules; thus, supporting the move from integral to modular architectures as the product evolves across multiple generations. Finally, the model and the experimental results provide design and managerial insights to both development engineers and managers. Specifically, for development engineers, the model and its analysis provide guidance for selecting the product architecture which leads to maximum performance. For development managers, the model and its analysis assist in deciding the optimal budget proportions to be allocated to modules and to design rules, given a fixed architecture and budget.

Keywords: Product development, product architecture, design rules, resource allocation.

[*] Corresponding author. ay11@aub.edu.lb.

1. INTRODUCTION

The role of architecture in defining systems and describing their behavior has been thoroughly documented in the systems engineering literature (Crawley et al., 2004). As a matter of fact, complexity theory is built upon the notion that a complex system is defined not only by its constituent parts (i.e., components and modules) but also by how these parts interact together (i.e. dependency structure or simply architecture) to provide utility to customers (Simon, 1965; Bar-Yam, 1997). The complexity of the system stems primarily from the fact that some of these interactions are unknown in nature and magnitude and their implications on system performance (e.g., laptop weight or storage capacity) and development process performance (development time and cost). To further explore systems architecture, researchers have distinguished between modular and integral architectures as the two polar extremes of an architectural continuum (Ulrich, 1995). More recently, network theory has added several interesting contributions that relate to the dependency structure of complex systems, which include architectural descriptions such as hierarchy, scale free, small world, etc. (Barabasi, 2002; Newman, 2003).

In addition to being a way to understand, manage, and characterize the architecture of a complex product system, in product development (PD) modularity is a strategy to design complex systems. Modularization proceeds in two steps: establishing a modular architecture and developing design rules (Baldwin and Clark, 2000). Establishing a modular architecture is concerned with the division of a product system into smaller building blocks, called modules, such that interactions within module boundaries are maximized and interactions between modules are eliminated or minimized.[1] Using this strategy, developing complex systems is based upon designing individual modules that are part of a larger system according to a formal architecture.[2]

[1] There are various clustering algorithms in the literature (e.g., Yu et al. 2007; Borjesson and Hölttä-Otto, 2014); however, in this paper, we treat the grouping of components into modules as exogenous and focus on between-module dependencies.

[2] Product or system architecture is the mapping of functional elements into physical modules (Ulrich, 1995).

One main benefit of modularization is that it facilitates system performance improvement by independently improving individual modules within the system. However, in complex systems, we cannot simply design each module separately (to improve its performance) and ignore the potential impact on the performance of other modules.[3] As opposed to ideal modularity, integral systems involve a strong dependency between individual modules where changes made to any module (to improve its performance) may deteriorate or improve the performance of others. Consequently, in an integral architecture an optimal performance for each individual module may not necessarily lead to a global optimal performance for the whole product or system due to the complex interactions between the various modules (Mihm et al., 2003). Thus, the second step of modularization is concerned with the dependencies outside module boundaries to investigate the possibility of eliminating them using design rules (Baldwin and Clark, 2000). An improved understanding of the dependency or relationship between modules (i.e., its nature, magnitude, and implication on involved modules) reduces the strength of dependence through either (a) the creation of a higher level design rule in the DSM, or (b) internalizing the rule within the design of each module (Baldwin and Clark, 2000). This process facilitates reaching a mutual upfront agreement (which is the design rule) regarding some design decisions.[4] That is, design rules function as standards mediating interdependencies between modules (Martin and Ishii, 2002; Frenken, 2006).

Although modular systems have many benefits related to development time and cost, a perfectly modular design may not always be achievable due to business and technical constraints (Hölttä-Otto and de Weck, 2007). Cutherell (1996) argues that integral architecture is often driven by product performance and cost while modular architecture is driven by variety, product change, engineering standards, and service requirements. Along similar lines, Whitney (2004)

[3] Performance can be defined as the return to development effort (or budget) along a technical (or economical) dimension of interest. Examples include the speed of a microprocessor or the number of features implemented in a new software release. Economical examples of performance can include market share and revenues.

[4] This process may reduce the design freedom (alternatives) for one or both modules, but provides the dependent module a head start and may save costly iterations, negotiations, and perhaps integration problems, which may surface later during development.

argues that modularity is not always a desirable property; a modular product is likely to be larger, heavier, and less energy efficient; especially in case of high power mechanical products (e.g., electro-mechanical-optical products), as opposed to low power signal processor type products (e.g., VSLI design, which can be considered fully modular).

This paper aims to develop a mathematical optimization model for resource allocation in PD to maximize product performance (by investing in product modules and in design rules) where the topology of the product architecture is taken into account. The model serves as a decision support tool for making resource allocation decisions early during the development process. That is, armed with an early assessment of product architecture, managers can compare and select from alternative architectures and gain insights into the appropriate level of effort that must be invested in various modules and their dependencies in order to maximize product performance.

The model extends prior PD models in general, and resource allocation models in particular, in two ways. First, we address the problem of improving product performance subject to an imposed architecture. By explicitly accounting for architecture, the model facilitates a better representation of PD practices where the team is not only challenged with the task of component engineering (i.e., improving module performance), but also systems engineering (i.e., explicitly addressing dependencies with other modules). A second contribution of the paper is the explicit segregation of PD investment effort into module and design rules. While investment in modules improves module performance, investment in design rules increases modularity of the product architecture.

The model yields three main results. First, we present a novel optimization model of how product architecture can be impacted by development efforts to explore performance evolution. The model and its analysis provide managers with a guide to the conscience assessment and comparison of alternative product architectures for architecture selection and redesign; and additionally, a tool to allocate scarce development resources optimally. Second, we explicitly address investments in dependencies between modules to reduce them (i.e., establishment of

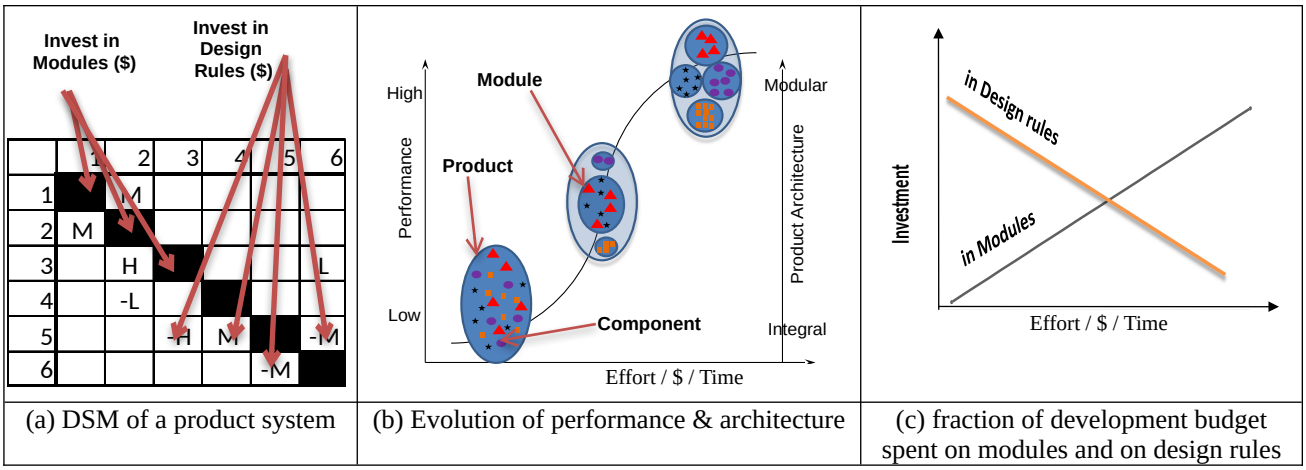
design rules), improving modularity; highlighting the influence of architectural dependencies on product performance. Finally, the model allowed us to test three hypotheses relating system architecture to investment (i.e., resource allocation) decisions and product performance.

In the next section, we introduce and discuss these three hypotheses. In order to investigate these hypotheses, we first develop in Section 4 a mathematical optimization model that uses product architecture to optimize the allocation of development budget to various modules and design rules. Then, in Section 5, we build a full factorial experiment using various architectural patterns that characterize product architectures. We apply the optimization model to these various product architectures and perform statistical analysis on the results. Section 5 also includes a discussion of these results and their managerial implications. In Section 6 we describe a subjective assessment of the various input parameters for the proposed optimization model, and a validation case study. Finally, the summary and conclusion are presented in Section 7.

2. SETTING OF HYPOTHESES

The ultimate objective of this paper is to build a foundation for a formal theory that maps product architecture to PD investment decisions and to product performance, as well, as envisioned in Figure 1. Figure 1(a) shows a product system represented by its Design Structure Matrix (DSM).[5] The PD investments can be placed on individual modules (represented by the diagonal DSM elements) and on design rules (represented by the off-diagonal DSM elements) that govern the relationships between these modules. In line with the literature (e.g., Cohen et al., 1996; Joglekar et al., 200; Kamrad et al., 2013), investments in individual modules result in performance improvement for the concerned modules. On the other hand, investments in design rules (i.e. between two modules) result in an improved understanding of the relationship between these modules which lead to the elimination of these dependencies or a reduction in their strength (Baldwin and Clark, 2000; Martin and Ishii, 2002).

[5] A DSM model displays the architecture of a complex system (or project) as a square matrix with *n* rows and columns (representing components of a system or tasks of a project), and *m* off-diagonal elements (representing dependencies between components or predecessor relationships between tasks) (Eppinger and Browning, 2012). It enables the user to model, visualize, and analyze the dependencies among the entities of any system and derive suggestions for its improvement or synthesis (Yassine and Braha, 2003; Sharman and Yassine, 2004).



(a) DSM of a product system | (b) Evolution of performance & architecture | (c) fraction of development budget spent on modules and on design rules

Figure 1: Investment decisions and its relationship to integral-modular dynamics

Many researchers have linked system performance to architecture (e.g., Henderson and Clark, 1990; Rivkin and Siggelkow, 2007; Braha and Bar-Yam, 2007; Gokpinar et al., 2010; Sosa et al., 2011; Cataldo and Ehrlich, 2012), but only few (e.g., Ethiraj, 2007; Ethiraj and Posen, 2013) have done so through R&D investments. Ethiraj (2007) argued that the interactions between components in a product system create constraint components that condition the R&D investments. He defined a constraint component as the one that holds other components from achieving their full performance potential. These constraint components act as performance bottlenecks and dictate the allocation of inventive efforts. Dong and Sarkar (2015) claimed that the endogenous progress potential of a product system is based on the complexity of its knowledge structure, which is represented by the knowledge associated with the product's modules and their dependency structure. Thus, the extent to which a product responds to investment and improves is a direct function of its architecture. Grounded in these observations, we postulate the following hypothesis regarding the relationship between the product architecture and performance through allocation decisions.

> Hypothesis 1: The product architecture (varied by holding number of dependencies fixed but changing their locations within a DSM model) affects resource allocation decisions and ultimately product (or system) performance.

A substantial amount of recent literature suggests that many products are becoming more modular over time (Schilling, 2000; Christensen et al., 2002; Fixson and Park, 2008). Fine and

Whitney (1999) introduced an intriguing theory that explains the integral-modular dynamics in a system. They explained that with an industry exhibiting a vertical structure and with an integrated product system, a number of forces[6] push toward a loss of the established position and possible disintegration of the product architecture and industry structure. On the other hand, with a modular product (and horizontal industry structure), numerous other forces[7] push toward the integration of product architecture and industry structure.

The development of a complex system starts with an integrated architecture to accomplish a specific functionality (Langlois and Robertson, 1992; Grove, 1996); however, demand for variability might induce the organization to offer the system with a few different product configurations (Schilling, 2000). As this process unravels, the organization's understanding of the complex system matures with time (specifically how the various pieces and modules interact and affect each other) and the architecture of the system evolves in the direction of modularity facilitating the isolation of some modules that can be developed faster and/or better independent of the rest of the system; thus, improving the overall functionality or performance of the system.[8] Eventually the system becomes extremely modular and performance improvement returns to expended efforts exhibit dimensioning returns. At this point architectural innovation kicks in favoring a return to a different integral architecture and the cycle repeats (Fine and Whitney, 1999). More recently, Luo, (2015) and Frenken and Mendritzki (2012) presented NK-models supporting the hypothesis of modularity as a mechanism to increase the speed of performance evolution; claiming that modular product architecture promotes performance evolution, whereas integral product architecture limits performance evolution.

These arguments lead to the hypothesized evolution of product performance and architecture as investments are made multiple times and over several generations, as shown in

[6] Competition, task complexity, and organizational rigidities that set in once a firm has an established market position.
[7] Technical advances, market power in one or more module suppliers, and potential profitability from integrating into a proprietary system offering.
[8] This is exactly what Baldwin and Clark (2000) call the "Splitting" operator of modular design to increase the option value of a system.

Figure 1(b).[9] In the beginning, the product exhibits an integral architecture (represented by a solid ellipse in Figure 1(b)). As PD investments are made, the architecture of this system evolves in the direction of modularity to allow for the isolation of some modules. This process continues until a highly modular architecture emerges (represented by an ellipse with small circles inside it). Beyond this point organizations cannot attain further performance improvement without going to a new S-curve (i.e., architectural innovation as discussed in Henderson and Clark (1990)). Therefore, this discussion leads us to the second hypothesis regarding the speed of improvement between modular versus integral product systems.

> Hypothesis 2: Holding number of dependencies fixed but changing their locations within DSM, modular architectures can attain higher performance levels compared to integral architectures.

Although the discussion for Hypothesis 2, above, argues for a shift from integral to more modular architecture as the product evolves, existing literature is scarce regarding how does this shift occurs. The DSM literature treats the dependency values as time invariant; for example, the time (and cost) of iterative activities are reduced using impact and learning multipliers and not through reducing the dependency values as iterations unravel (Eppinger and Browning, 2012). As an exception, Martin and Ishii (2002) described a method to decouple (i.e., modify) a product architecture so that future generations of the product would require less design effort. The method is based on calculating a generational variety index (GVI) and a coupling index (CI). While the GVI indicated the amount of redesign effort needed for each module to meet future changes, the CI indicated the amount of redesign effort needed for the corresponding dependent modules. Based on these indices, they were able to prioritize interface standardization and modularization efforts. However, this process occurs only once at the outset of the PD process and cannot be repeated. It would be interesting to allow this process to evolve dynamically over multiple generations of the product and observe the sequence of module design and modularization effort.

[9] The S-curve shown in Figure 1(b) is a widely used tool for thinking about technological innovation and evolution (see, for example, Foster (1986) and Anderson and Tushman (1990)).

Along similar lines, Fixson and Park (2008) developed a product architecture measurement based on interface irreversibility which measures the effort required to disconnect a module's interface, and interface standardization measuring the degree of compatibility between modules. Based on these measures, they conducted a longitudinal study of the bicycle industry from 1980 to 1990 and showed that these metrics have changed over time as the bicycle architecture evolved. Finally, Ethiraj and Posen (2013) have traced the constraint components in Personal Computers from 1981 to 1998 and have reported shifts in the allocation of inventive effort in various modules over time caused by temporal dependency changes in the product architecture.

This discussion allows us to further hypothesize that if we were to plot the corresponding budget fractions spent on modules and on design rules along a generational timeline, we expect to see the trends shown in Figure 1(c). That is, the proportion of the budget spent on design rules decreases (as our understanding of the system matures with time), and this is accompanied with an increase in the fraction invested in modules as time (and budget) progresses. This scenario is in line with the typical systems engineering practice of attending to module dependencies first before proceeding to detailed design (i.e., working on modules). Hence, we formulate the final hypothesis as follows.

> Hypothesis 3: For the same product architecture, there is a shift in the temporal allocation of resources from design rules to individual modules; thus, supporting the move from integral to modular architectures as the product evolves (across multiple generations).

3. LITERATURE REVIEW

The topic of this paper is at the intersection of several research streams from product development and modularity literature, to network analysis techniques and complexity theory. This section discusses these streams of literature as they relate to the proposed model.

2.1 Product Development (PD) Literature

Mathematical optimization models in the PD literature mostly focus on the timing, cost, and execution sequence of the various development activities (Krishnan and Ulrich, 2001; Browning

and Ramasesh, 2007). Other models discuss product architecture from the viewpoint of family planning, platforming decisions, design for variety, customization, change propagation, and evolvability (e.g., Martin and Ishii, 2002; Simpson, 2004; Clarkson, 2004; Zacharias and Yassine, 2008; Luo, 2015). Few of these models discuss resource allocation decisions within a PD context (Cooper and Kleinschmidt, 1988; Kavadias and Chao, 2008); however, they do not consider explicitly the impact of architecture on resource allocation decisions and ultimately performance.

Few exceptions in the PD literature relate product architecture to some measure of product performance. For example, Cohen et al. (1996) examine the tradeoff between product performance and profit as a function of a fixed sales window where more time spent on improving product development performance results in lost sales due to fixed sales window. Alternatively, if the product is released prematurely, then profit is lost due to unsatisfied customers. The model yields an optimal development time that maximizes profit.

Joglekar et al. (2001) investigate a PD process consisting of two design activities and also assume a simple production function for generating performance: the more time spent working on a task, the higher the level of performance that can be achieved. Their model tracks the degree to which each task adds to the overall performance in response to the effort devoted to it. They assume that each task contributes to the overall system performance at a different rate and at the same time deteriorates the performance of the other coupled tasks by generating rework, and thus requiring extra effort for coordination. Based on the degree of coupling or dependency between the two activities, they determine the optimal execution strategy (i.e., sequential, parallel, or overlapped) that will maximize the overall product performance.

Mihm et al. (2003) address a complex product development project composed of many inter-related modules where each engineer is responsible for designing a certain module by taking into consideration the status of other modules that are present in the PD project. Accordingly, an overall system performance was defined as the sum of the performances of the individual modules. The performance of any module depends on its own design decision and on

the design decisions of all other modules that influence it. As a result, they have found that the design project becomes more difficult to converge to a solution as the number of interacting modules increases, and thus suggested modularization to remedy this situation.

Ulku and Schmidt (2011) developed a mathematical model to optimize the product performance as a function of its architecture and the quality (i.e. performance) of its modules. The functional form that they use for performance is increasing in module quality and decreasing in the level of product modularity. Then, the product performance was mapped to market share. They found that modular architectures are more likely in supply chain network when adversarial relationships exist. On the contrary, long term trust-based relationships incubate integral product architectures. Despite their contribution, they failed to provide a comprehensive analysis for practical situations that involve more than two subsystems or modules; a situation which we address in this paper.

2.2 Network Analysis Literature

In the network (or graph) theory literature, we can also trace many efforts for characterizing and linking the network structure to system behavior; asserting the advantages and disadvantages of possessing a specific structure (e.g., Watts, 1999; Strogatz, 2001). Issues of robustness to random failure of nodes and even quality and development speed as a function of network architecture were also studied (Braha and Bar-Yam, 2007; Gokpinar et al., 2010; Sosa et al., 2011; Cataldo and Ehrlich, 2012).

Several researchers have applied network analysis to analyze the statistical properties of large engineering projects that are represented as networks (e.g., Fuge et al., 2014; Braha and Bar-Yam, 2004). An important aspect of using network analysis is defining the nodes and the connecting links. The nodes may represent people, tasks, or function calls, which communicate via links representing engineering change orders, parameters, specifications, or signals. Once the nodes and links have been defined, several graph properties may be calculated including both global (i.e., whole network) and local (i.e., node-centric) measures. These measures are ultimately used to explain and perhaps predict the behavior of these network systems. A full

summary of these measures is beyond the scope of the paper, but interested readers are directed to Wasserman and Faust (1994) and Newman (2010).

The study of such engineering networks has led to many interesting results. Most notably, Braha and Bar-Yam (2007) have shown that the underlying network structural properties provide key information about the characteristics of error and defect propagation, both whether and how rapidly it occurs. These architectural properties have implications for the functional utility of engineering systems including sensitivity and robustness (i.e., error tolerance) properties. In particular, in the context of resource allocation in complex engineering networks, remarkable improvement in the performance of engineering systems can be achieved by focusing engineering and management efforts and investments on central information-consuming and information generating nodes and their couplings with neighboring nodes.

Metrics that capture architectural properties can be derived from network analysis. One such metric relates to the position of a module within the overall product architecture. Network theory refers to this measure as the ‘degree’ of a node in a network (Wasserman and Faust, 1994). Sosa et al. (2007) adopted this approach in defining modularity of a complex product and defined modularity, from the perspective of a component and not the system, by looking at the lack of interfaces or connectivity among the components. These connections represent design dependencies and the degree measure represents how components share direct interfaces with adjacent components. Hence, a component is considered more modular if it is more independent as indicated by its number of connections or its disconnectedness. Conversely, the more connected or dependent a component is with other components, the less modular it is.

As a follow up, Sosa et al. (2011) examined how the presence of highly connected nodes (called hubs) relates to a system’s quality. They provide empirical evidence that the presence of hubs in a system’s architecture is associated with a low number of defects. Furthermore, they showed that complex engineered systems may have an optimal fraction of hub components with respect to system quality. A recent paper by Sosa et al. (2013) looked at the relationship between architecture and quality in a product network by defining a “cyclicality” measure, which relates

to the amount of coupling between modules. They defined cyclicality as the extent to which a component depends on itself via other product components. They found that components involved in cycles were significantly more likely to contain bugs (i.e. lower quality).

2.3 Complex Systems Literature

Studies of complex adaptive systems, set initially in the physical and biological sciences, have made the transition to management science. The NK model (Kauffman, 1993) has become a popular platform for studying organizations as complex adaptive systems (e.g., Ethiraj and Levinthal 2004). Modeling efforts using the NK model assumed that interactions among decisions are random and they have overlooked the pattern of interaction among these decisions.

Rivkin and Siggelkow (2007) investigated the impact of organizational structure, as reflected in the interactions among decisions, on the level of exploration necessary to discover a good configuration of decisions. The results showed that fixing the total number of interactions among decisions, a shift in the pattern of interaction can significantly alter the shape of the performance landscape of the organization. Ten different DSM patterns were simulated using an NK model and the results showed that some DSM patterns can have different performance landscapes although the total number of interactions was kept fixed.

Luo (2015) proposes a simulation-based method to assess the effect of product architecture on product evolvability by analyzing a design structure matrix using an NK model. He defined product evolvability as the ability of the product's design to successively generate performance-improving variants. Through his simulation results for the NK model on various DSM architectures, he showed that there is negative impact component interaction density on product evolvability, supporting the claim that modular product architecture promotes product evolution, whereas integral product architecture limits product evolution. Similar approaches and observations were noted in simulations of the NK model by other authors (e.g., McNerney et al., 2011; Frenken and Mendritzki, 2012).

All of the above NK-based models have used a random fitness function to capture component performance contribution. Alternatively, in this paper, we use a more sensible

performance contribution function based on optimized effort-driven production functions, which clearly present an improvement existing NK-based models of module and system performance; particularly used for product development.

4. THE PROPOSED MODEL

In this section we present a mathematical programing formulation for the optimal investment decisions in PD. As discussed in Mihm et al. (2003) and Rivkin and Siggelkow (2007), for example, the overall product (or system) performance can be described as the total performance of the individual modules. Furthermore, the performance of each individual module is dependent on the amount of resources invested in that particular module (Dutton and Thomas, 1984). However, as discussed in Allada and Lan (2002) and Martin and Ishii (2002), the interdependencies among the different modules require that any investment made in one particular module to be accompanied by additional investments in dependent modules in order to ensure their compatibility with each other. We assume that the interdependency can be reduced by investing in the design rules (defining the connections or relationships between these modules); thus, reducing the level of interdependency among the modules.[10] Hence, the problem is stated as finding the optimal investment amounts that should be made in each of the individual modules and in the design rules in order to maximize the overall product performance. The following decision variables are then used to model the problem: α_i is the amount invested in module *i*, and $\theta_{ij}$is the amount invested in the design rules for the connection between module *i* and module *j*. The overall system performance P is then expressed as the sum of the individual performance of the *N* modules such that:

$$P_{\square} = \sum_{i=1}^{N} p_i(\alpha) \quad (1)$$

where p_i ($1 \le i \le N$) is the performance of module *i* that is an increasing function of the total investment in module *i*. That is, the more investment is spent on a module, the higher the level of performance that can be achieved (Cohen et al., 1996; Joglekar et al., 2001; Kamrad et al., 2013).

[10] For example, communication reduces the negative effect of rework at the expense of communication time (Loch and Terwiesch, 1998).

Furthermore, since in an integrated system modules impact each other, we assume that investing in a module also affects (i.e. either improves or deteriorates) the performance of other dependent modules (McNerney et al., 2011; Luo, 2015). However, the performance, p_i, cannot be negative because we can always choose to keep the current design of module *i* and only incur a compatibility cost as discussed next.[11] Hence, the performance of each module is an increasing function of investments in the module itself and in other dependent modules; thus, performance of module *i*, $p_i(\alpha)$, is a function of α which is the investment vector ($\alpha_1, \ldots, \alpha_N$) in all modules.

In addition to the amount invested to improve the performance, additional investments are required to insure compatibility among the modules. The investment that is due to compatibility, l_{ij}, is a function $L_{ij}(\alpha_i, \theta_{ij})$ that is increasing in α_i and decreasing in θ_{ij}. Thus, for two dependent modules (say module *j* depends on *i*), investing α_i in a module *i* necessitates an additional investment in module *j* that is proportional to α_i. Furthermore, the additional investment $L_{ij}(\alpha_i, \theta_{ij})$ can be reduced by investing θ_{ij} in the design rules between modules *i* and *j*. It is noteworthy that in our model we only consider direct compatibility costs and ignore indirect ones.[12] Finally, a maximum investment budget *B* is considered to be available throughout the development process which is normalized to 1. The investment decision problem has the following general form:

$$max\, P_{\square} = ¿ \sum_{i=1}^{N} p_i(\alpha), (2) ¿ s.t. \sum_{i=1}^{N} \alpha_i + \sum_{i=1}^{N} \sum_{j=1; j>i}^{N} \theta_{ij} + \sum_{i=1}^{N} \sum_{j=1; j\neq i}^{N} l_{ij} \leq B, (3)$$

$$l_{ij} = L_{ij}(\alpha_j, \theta_{ij}), (4)$$

$$\alpha_i, \theta_{ij}, l_{ij} \geq 0 \; \forall (i, j). (5)$$

The auxiliary variable l_{ij} denotes the investment needed by module *i* to insure compatibility with module *j* after $\alpha_j$is invested in module *j*. It is worth noting that a set of performance ceiling constraints (of the form $p_i(\alpha) \leq p_i^{ceiling}$) can be added to this base optimization model.

[11] This is a reasonable assumption which is similar to Baldwin and Clark's (2000), where a truncated Normal distribution is used for module performance.

[12] For instance, in an architecture where module A impacts B and module B impacts C, investing in module A alone will result in compatibility cost incurred for module B only, and not C.

4.1 **The Stylized Model**

In this section, we consider functional forms for $p_i(\alpha)$ and $L_{ij}(\alpha_j, \theta_{ij})$ and present a stylized model based on (2)-(5). Particularly, the performance function $p_i(\alpha)$ of each module *i* follows a production function that is increasing in terms of the resource α_i (Cohen et al., 1996; Joglekar et al., 2001; Kamrad et al., 2013). The slope of the production function is also a function of the resources α_j that are invested in other modules. $p_i(\alpha)$ is then of the following form:

$$p_i(\alpha) = \max\left\{0, \left(C_i + \sum_{\substack{j=1 \\ j \neq i}}^{N} C_j f_{ji} \alpha_j\right)\alpha_i\right\}. \quad (6)$$

The module performance is zero when we have zero investment in the module and C_i is a measure of complexity for module *i*. C_i is hence a proxy for the design complexity (or module size as in Baldwin and Clark (2000)) of each of the modules where a simple module (i.e. small C_i value) will have a small impact on the overall system performance while a complex module (i.e. large C_i value) has a larger impact. It is assumed that each module has its specific C_i which is known in advance.

The factor f_{ij} represents the strength of dependency among modules *i* and *j*, where $-1 \leq f_{ij} \leq 1$.[13] We note that if $f_{ij} = 0$, then module *i* is independent from module *j* and hence investments in modules *j* ($j \neq i$) will have no impact on module *i*. In the cases where $f_{ij} > 0$, investments in modules *j* increase the slope of the performance function of module *i* thus increasing the performance of *i* (i.e., synergistic or performance-enhancing dependency). Finally, when $f_{ij} < 0$, investments in module *j* decrease the slope of the performance function of module *i* (i.e., damaging or performance-reducing dependency). Note that in order to capture the synergistic effect of integrality between modules *i* and *j*, concurrent investment must be made in the modules. Thus, the term $C_j f_{ij} \alpha_j$ is multiplied by α_i in Equation (6).

In order to ensure compatibility after upgrades among the interconnected modules, an investment in one module requires additional investments in the other dependent modules (that

[13] We assume a symmetrical reciprocal dependency structure; that is, $f_{ij} = f_{ji}$.

is, directly connected to it). The development manager can elect to invest in the design rules in order to reduce or even eliminate the dependencies among the modules (Baldwin and Clark, 2000). Investing in the design rules reduces the amount that needs to be spent on dependent modules. For instance in practice, an investment may be made in the connections between the modules in order to make them more generic or standardized. The compatibility among the interconnected modules is modeled as follows. An investment α_i in module *i* requires an equivalent investment $L_{ij}(\alpha_i, \theta_{ij})$ to be made in each module *j* to insure *j*'s compatibility with changes in *i*.[14] The following decreasing function $L_{ij}(\alpha_i, \theta_{ij})$ is considered:

$$L_{ij}(\alpha_i, \theta_{ij}) = \left(\frac{C_j}{C_i}\right)\left(|f_{ij}|e^{-(k_{ij})\theta_{ij}}\right)\alpha_i \forall (i, j); (7)$$

where $\theta_{ij}$is the amount invested in the design rules for the connection between modules *i* and *j*, and $k_{ij} \geq 0$, reflects a design knowledge parameter that alters the sensitivity of the function $L_{ij}(\alpha_i, \theta_{ij})$ depending on the designer's prior knowledge about the connection between modules *i* and *j* (Dong and Sarkar, 2015). Note that the compatibility cost is incurred regardless of the sign of f_{ij}; thus, the absolute value used in equation (7).

Following equation (7), an investment α_i in module *i*, requires an additional proportional investment $\left(\frac{C_j}{C_i}\right)\left(|f_{ij}|e^{-(k_{ij})\theta_{ij}}\right)$ in module *j* to ensure compatibility of module *j* with the updated module *i*. We note the scaling factor $\frac{C_j}{C_i}$ is included to account for the fact that the investment in each module should be proportional to its complexity. Therefore, the total investment in module *i* includes the amount α_i for updating module *i* in addition to an amount $\left(\frac{C_i}{C_j}\right)\left(|f_{ij}|e^{-(k_{ij})\theta_{ij}}\right)$ for every α_j invested in the other modules (impacting module *i*) to ensure the compatibility of the system. The investment decision problem is then formulated as follows:

[14] A similar assumption was made in Smith and Eppinger (1997) and Yassine et al. (2003). In their DSM analysis, they assumed that all tasks in the DSM proceed in parallel and iterate multiple times. At any iteration stage, one unit of work by any task results in a fraction of rework for the other dependent tasks during the next iteration stage.

$$max\ P_{\square} = \sum_{i=1}^{N} p_i(\alpha) = \sum_{i=1}^{N} max_{\square} \left\{ 0, \left(C_i + \sum_{\substack{j=1 \\ j \neq i}}^{N} C_j f_{ji} \alpha_j \right) \alpha_i \right\} (8)$$

$$s.t. \sum_{i=1}^{N} \alpha_i + \sum_{i=1}^{N} \sum_{j=1; j>i}^{N} \theta_{ij} + \sum_{i=1}^{N} \sum_{j=1; j \neq i}^{N} l_{ij} \leq B(9)$$

$$l_{ij} = \left(\frac{C_j}{C_i} \right) \left(|f_{ij}| e^{-(k_{ij})\theta_{ij}} \right) \alpha_i \forall (i,j) (10)$$

$$\alpha_i, \theta_{ij}, l_{ij} \geq 0 \forall (i,j) (11)$$

The objective function (8) maximizes the total weighted performance of all the modules.[15] Constraint (9) is a budget constraint which limits the total investments in the individual modules, the total investment in the design rules, in addition to the investment that is required to ensure compatibility to a maximum budget B.[16] Constraint (10) insures the compatibility of the system by forcing an investment module *i* following an investment in module *j*. Finally, note that since the objective function is increasing in α and all the coefficients of α in the budget constraint are non-negative, it can be easily verified that the budget constraint is binding at optimality.

4.2 **The Dynamic Model**

The dynamic model simulates the product development decisions over several investment runs in order to investigate if there is any trend (i.e., shift) in investment allocations between the design rules and the modules. Thus, the dynamic model, which is used to study the second hypothesis, is obtained by applying the static model (Equations 8 through 11) iteratively multiple times over several product generations. Starting from an initial system containing modules and dependencies, we solve the static problem *n* times where at each iteration $1 \leq t \leq n$ a new budget B_t is available for investment. Furthermore, in order to account for the updates that occurred in the previous investment cycle, the system state at the start of each iteration *t* is updated to include the values of the design rules that resulted from the investments in period *t*-1. In other words, the values of f_{ij} at period *t* are updated to include the investment θ_{ij} that occurred at iteration (*t*-1). Therefore, the new values of f_{ij} in period *t* are set as $f_{ij}^{t} = \left(f_{ij}^{t-1} e^{-(k_{ij})\theta_{ij}^{t-1}} \right)$. We also assume that the

[15] Note that the objective function is a convex function. However, since we are maximizing, then the problem is a non-convex optimization problem.

[16] Constraint (9) is neither convex nor concave.

performance from one investment cycle to the next is additive: $P_t = \sum_{i=1}^{t} P_i$. For example, assume that budget B_1 (in iteration 1) resulted in a total product performance of P_1 when the optimization problem is solved the first time. Then, in iteration 2, we use a new budget B_2 and solve the optimization problem again (using updated f_{ij} values). The new total performance P_2 in the second iteration is achieved on top of P_1 and thus the cumulative performance by the end of iteration 2 is $P_1 + P_2$.

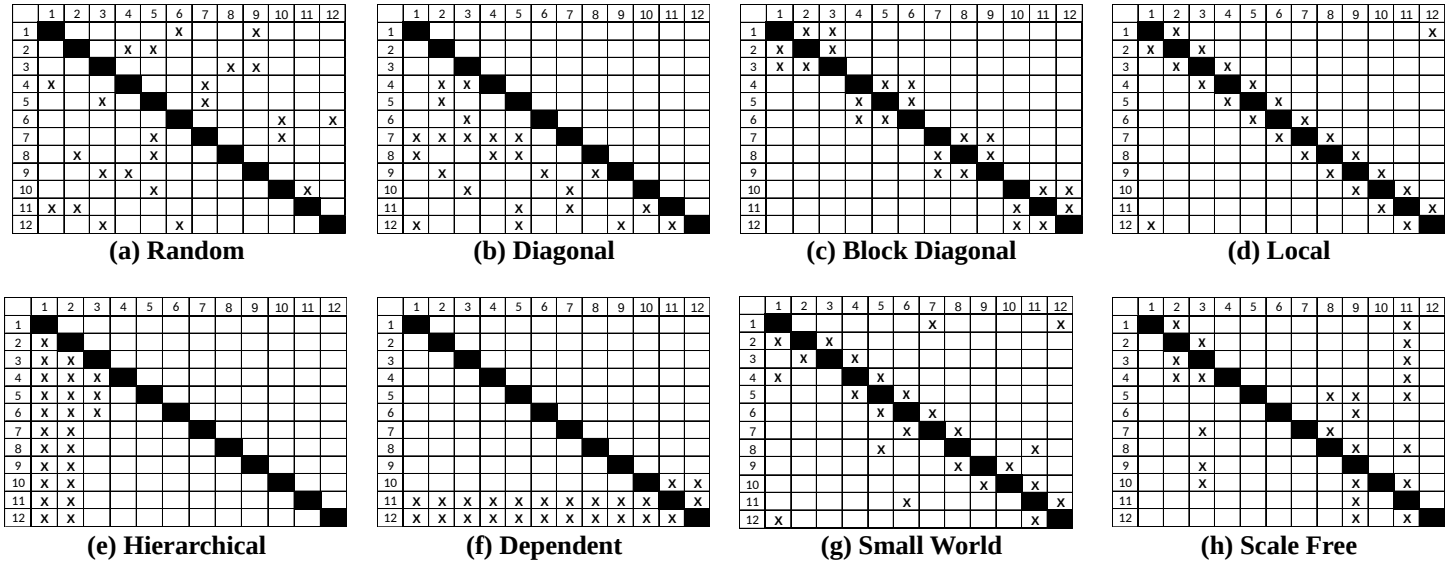
5. EXPERIMENTAL SETUP, ANALYSIS AND RESULTS

To test the three hypotheses that were outlined in Section 2, we applied the proposed optimization model on a controlled set of product architectures. In this set, we have produced product DSMs of sizes 12 and 16 elements (i.e. modules) in order to detect any statistical differences due to problem size. Furthermore, we controlled the total number of dependencies within each DSM by holding fixed the number of allowed interactions per DSM element. According to the NK model, the number of dependencies is equal to N × K. We have used N = 12 and 16 as two levels for factor N, and K = 1, 2 and 3 as three levels for factor K. Our choice for the three levels for K is supported by ample evidence from the literature.[17]

In many real-world settings DSM interdependencies are patterned (i.e. have a particular architecture) rather than random (Rivkin and Siggelkow, 2007). So, we have embedded different architectural patterns into the various DSM experiments; mostly the relevant DSM configurations presented in Rivkin and Siggelkow (2007). Thus, a third factor in our experimental design is called architecture and is set to eight levels as follows: Random (Benchmark), Diagonal (Sequential Engineering), Block Diagonal (Modular), Local (Sequential Engineering with Limited Concurrency), Hierarchical (Concurrent Engineering with Global Design Rules), and Dependent (Platform), Small World (Integrated), and Scale Free (Integrated). Next, we provide details on the reasons behind our choice of each of these eight DSM

[17] Several researchers have investigated the average number of links per node in a product DSM (i.e., a DSM where the nodes are physical components or product modules) and they all reported an average nodal degree between two and three (Van Wie et al., 2001; Greer, 2002; Whitney, 2005; Rivkin and Siggelkow 2007; Whitney, 2008).

architectures and their suitability in representing PD networks in our experimental design. Figure 2 shows the eight product architectures for N = 12 and K = 2.



(a) Random **(b) Diagonal** **(c) Block Diagonal** **(d) Local**

(e) Hierarchical **(f) Dependent** **(g) Small World** **(h) Scale Free**

Figure 2: The eight different DSM architectures. All with the same number of total interactions: N = 12 and K = 2; NK = 24 total module interactions (Adapted from Rivkin and Siggelkow (2007))

We define the random architecture by placing exactly K dependencies (i.e. marks in the DSM) randomly along the off-diagonal positions for each DSM row (see Figure 2a). Although this DSM configuration is a popular setup in the organization literature (Rivkin, 2000), we merely used it in our analysis as a benchmark for comparison to other more realistic product architectures.

We start discussing product architectures that are grounded in the product development and DSM literature (Yassine and Braha, 2003). For instance, both the diagonal and the block diagonal architectures are a direct result of two popular DSM analysis techniques: partitioning and clustering (Yassine et al, 1999; Yassine et al., 2007). Partitioning algorithms convert any DSM into a lower triangular form, where design decisions are ordered in such a way that highly influential modules are designed first followed by the less influential modules (Meier et al., 2007). In this hierarchy, a module cannot affect other modules above it in the sorting hierarchy. Diagonal DSMs represents a sequential engineering approach which is a popular development strategy in many organizations. To generate a diagonal DSM, marks are added randomly below

the diagonal until the total number of required or set dependencies is reached (see Figure 2b). While the objective of DSM partitioning is to obtain a lower triangular DSM, clustering algorithms strive to convert a DSM into a block diagonal form. Block diagonal DSMs (Figure 2c) represent a modular architecture where interactions are confined to groups of modules and no interactions are allowed between these groups (Yu et al., 2007). This architecture represents a practical modular design for many product systems (Baldwin and Clark; 2000).

The local architecture is formed by allowing each DSM element to be affected by exactly K neighboring nodes on either side of it (see Figure 2d). The local architecture is a reasonable choice for many product systems that follow a sequential engineering approach with limited concurrency, where design decisions made for one module constrain the design decisions for the next module and design iterations are confined to a small set of neighboring modules.

The hierarchical architecture is a special case of the diagonal architecture, which assumes that design decisions are ordered in such a way that highly influential modules affect all other modules below them in the hierarchy. Baldwin and Clark (2000) refer to this type of DSM architecture when discussing concurrent development using system-wide or global design rules. These are the modules that impose rules on all other modules in the system and are placed at the top of the design hierarchy. Hierarchical DSMs are created by adding dependency marks below the diagonal, starting with the first column, then in the second column, and so on until the required number of dependencies is reached (see Figure 2e).

The dependent architecture captures the platform strategy adopted by many firms to create product variety and derivatives (Simpson, 2004; Zacharias and Yassine, 2008). This architecture is characterized by the existence of platform modules; sometimes called “bus” modules (Sharman and Yassine, 2003). Bus modules have relations with all other peripheral modules and constitute the heart of the product architecture. Product variety and derivatives are created by replacing or altering any of the peripheral modules. We construct such a matrix by adding dependency marks starting with the last row, then in the second-to-last row, and so on until the required number of dependencies is reached (see Figure 2f).

So far we have utilized various product architectures discussed in the DSM literature. However, the next two architectures are adopted from the complex network literature which we think have theoretical support to be representative of product architectures. The small world architecture is formed by starting with a local architecture and then each off-diagonal dependency mark is exchanged with a randomly chosen location in the same row with a probability p, where $0 \leq p \leq 1$ (see Figure 2g). The small world architecture is a reasonable architecture for product systems as most module dependencies are local but few dependencies exist between distant modules. Many real world complex systems have been shown to be small world networks (see, for example, Braha and Bar-Yam (2007)).

Dynamic growth and preferential attachment supports the formation of scale free networks in PD environments (Valverde et al., 2002; Barabasi et al., 2000). Preferential attachment assumes that a network is evolving by the constant addition of new nodes over time. Given that a new node appears, and assuming that it has one link to attach to another node, it will link to older nodes with higher probability than to recently added nodes. Therefore, as the network grows, hubs will be created and will dominate the network's connectivity over time. Preferential attachment may explain the scale free nature of PD projects if we envision a project starting initially with a handful of modules (e.g. core modules) and as the product design evolves, more modules are added to the product. However, the core modules are likely to affect new modules since they may set or constrain the product requirements. To generate the scale free DSM architecture (see Figure 2h), we follow the steps suggested by Barabasi and Albert (1999). Their algorithm simply captures a "rich-get-richer" dynamic where modules that already have many dependencies are more likely to attract new dependencies than modules with fewer dependencies. We refer to both the small world and the scale free architectures as integrated architectures as no modular patterns can be readily apparent in their dependency structure.

Additionally, for each of the 48 different combinations of architecture, N, and K, we have generated 50 sample problems using randomly generated f_{ij} values (resulting in a total of 2400 problems). The f_{ij} values were sampled from a Uniform distribution between -1 and 1. In all the

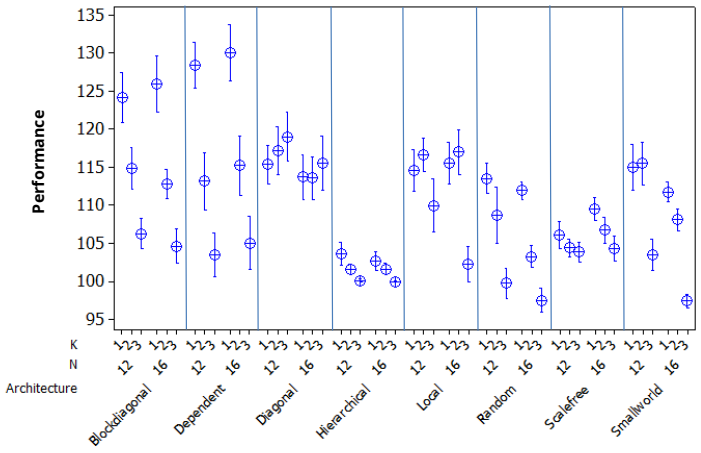
2400 problems, the budget was set to 1, k_{ij} =50 and c_{ij} = 100. Finally, we have solved all the 2400 problems optimally (using the model of Section 3) and noted four outputs: total performance, total investment in modules, total investment in design rules, and total compatibility investment.[18] Various statistical analyses on the obtained optimization results were performed and are discussed next.

5.1 Analysis and Discussion of the Static Model

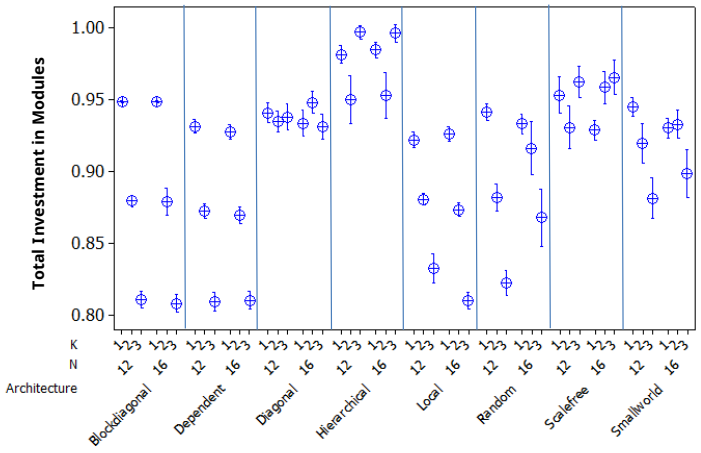
We start first by performing an ANOVA (analysis of variance) on the three main factors: N, K, and Architecture and their 2-way interactions. We have used three different response variables: total performance (or simply performance), total investment in modules (referred to as Alpha_tot), and total investment in design rules (referred to as Theta_tot). The results show no statistical difference (at 99% confidence) between the different N levels, so our results are insensitive to the choice of N. However, the analysis shows a statistical difference between the various levels of K and Architecture. Additionally, the analysis also shows statistically significant 2-way interactions between N and Architecture and between K and Architecture.

The results for the solutions of the 2400 DSM problems sorted by N, K and Architecture are shown in Figure 3. The figure provides an overview of the results using ANOM (one-way analysis of means) for all experiments (averaging over all factors and assuming a 99% confidence level). It is evident from this figure that there are significant variations (between experimental configurations) in mean and variance of performance and budget breakdown

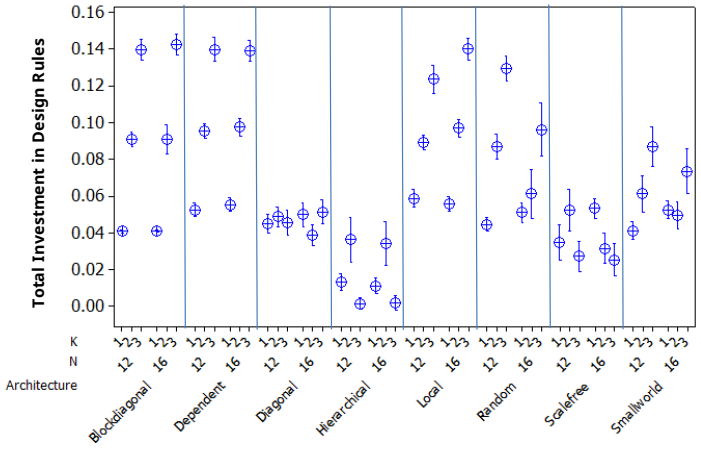
[18] YALMIP is used to formulate the non-linear optimization problem (http://users.isy.liu.se/johanl/yalmip/). To solve the non-linear and non-convex problem, bmibnb with fmincon solver which is part of the Matlab optimization toolbox is used. The bmibnb solver is a global solver for non-convex problems that implements a branch-and-bound approach and uses linear programming relaxations and convex envelope approximations to ensure global optimality.



(a) Performance



(b) Total Investment in Modules



(c) Total Investment in Design Rules

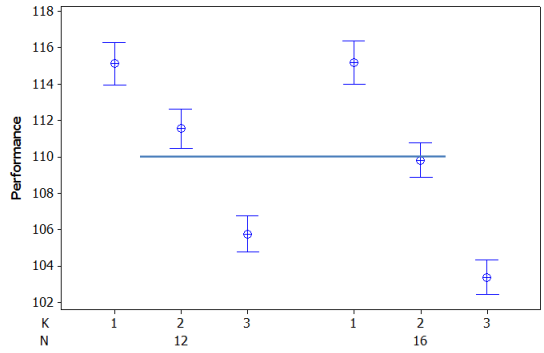
Figure 3: ANOM (99% confidence intervals) grouped N, K and Architecture

between modules and design rules. We also note an overall decreasing trend in performance as K increases, in agreement with Luo's (2015) simulations. This is accompanied with a decrease in module investments and an increase in design rules investments. To hone down on these observations, we lumped the data into different groups (N and K) and performed ANOM on them. The results are shown in Figure 4.

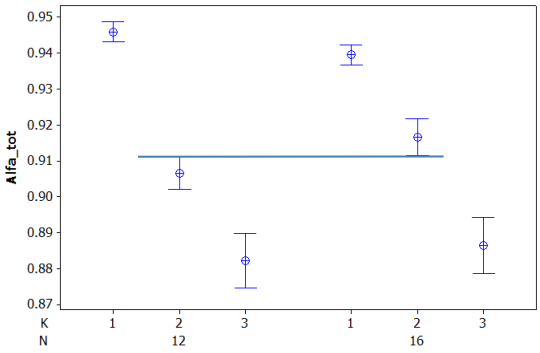
The ANOM plots in Figure 4 also reiterate the ANOVA results regarding the insignificant interaction between N and K considering all three response variables of performance, module, and design rules investments. Also, this figure shows that an increase in the total number of DSM interactions results in a statistically significant reduction in total product performance (Figure 4a) and in total investment in modules (Figure 4b), while it results in an increase in investment in design rules (Figure 4c) and in compatibility cost (Figure 4d).

Similar ANOM plots can be obtained for Architecture and N taken together and for Architecture and K together, as shown in Figure 5. Figure 5a shows a change in behavior for small world networks between N = 12 and 16. Figure 5b shows a more significant change in performance amongst the various architectures as K is increased from 1 to 3, which is indicative of the statistically significant interaction between Architecture and K, as indicated in the ANOVA analysis above. The block-diagonal and dependent architectures perform exceptionally well at low K but are outperformed by the diagonal architecture at high K.

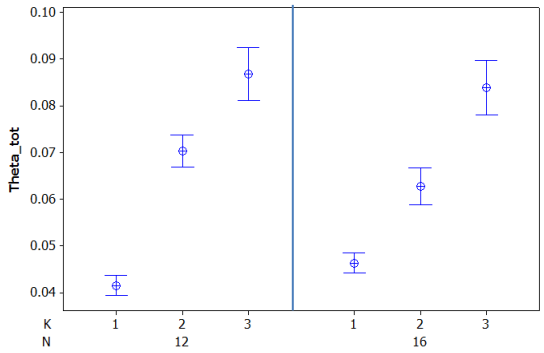
These results show that the block diagonal and dependent architectures are best for products with small number of dependencies among modules (i.e., low K values). As for the block diagonal, small blocks around the diagonal are better than large blocks. As for the dependent architecture, platforms made up of few modules are also preferred to platforms composed of large number of modules. As the number of dependencies among modules increase, it is better to organize the modules around a dependent architecture, where design decisions are made sequentially.



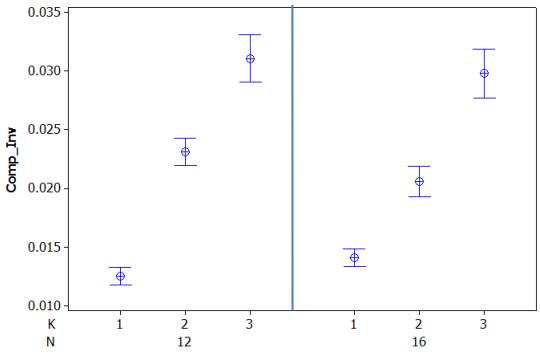
(a) Total Performance (Performance)



(b) Total Investment in Modules (Alpha_tot)

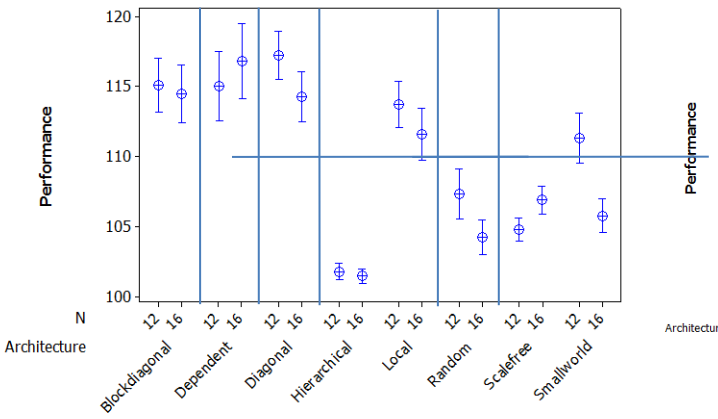


(c) Total Investment in Design Rules (Theta_tot)

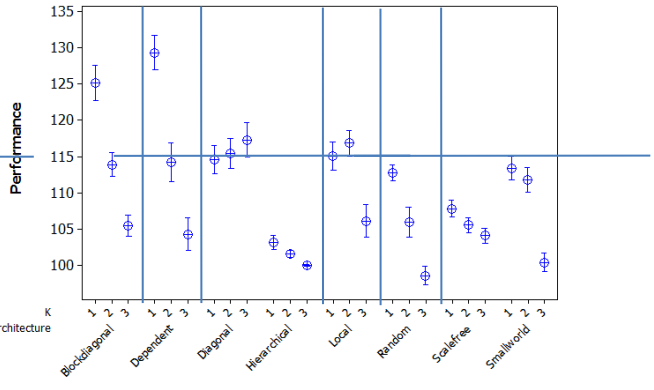


(d) Total Investment in Compatibility (Comp_Inv)

Figure 4: ANOM (99% confidence intervals) for N and K



(a) Architecture & N Interaction



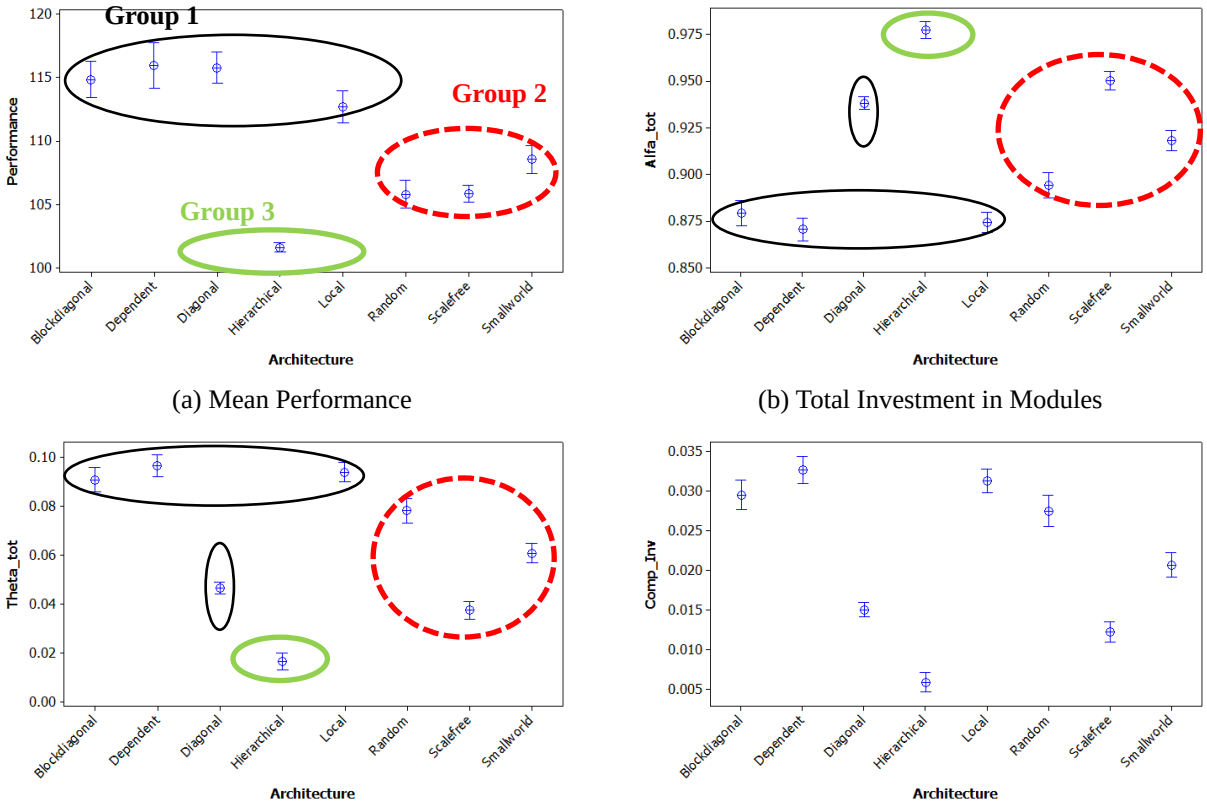
(b) Architecture & K Interaction

Figure 5: ANOM (99% confidence intervals) for Architecture & K and Architecture & N

5.2 Investigation of the Proposed Hypotheses 1 and 2

In this section we investigate Hypotheses 1 and 2 posed earlier in the paper. To do so, we group the data around architecture only and perform an ANOM as shown in Figure 6. The figure shows

the results for the solutions of the 2400 DSM problems sorted by product architecture (averaging over all factors and assuming a 99% confidence level). It clearly confirms that architecture affects resource allocation which in turn affects performance (Hypothesis 1). It also shows significant differences in system performance between the different architectures, confirming our hypothesis that system performance is indeed affected by architectural choice. Additionally this figure shows that some product architectures provide higher performance although exhibit larger variability in performance too. Alternatively, we note that although the performance of Group 3 architectures is lowest, its performance variance is the smallest amongst all groups.



(a) Mean Performance

(b) Total Investment in Modules

(c) Total Investment in Design Rules

(d) Total Investment in Compatibility

Figure 6: ANOM (99% confidence intervals) for 8 different DSM architectures

Inspecting the constituent architectures of Group 1, in Figure 6a, shows that modular architectures (represented by block-diagonal, dependent, and local) achieved higher performance than integrated architectures as represented by Group 2 architectures (composed of random, small world and scale free). This confirms Hypothesis 2 that modular architectures facilitate a

faster increase in performance as compared to integrated ones, in agreement with the literature (Frenken and Mendritzki (2012).

It is also interesting to compare the performance level of the eight architectures (i.e., Figure 6a) with the investment profiles in modules (and in design rules) as shown in Figure 6b (and in Figure 6c). This comparison shows that although Group 1 architectures have a statistically higher performance than Group 2, their total investments in modules are statistically lower than Group 2 (diagonal architecture is an exception); however, larger investments are made in design rules (compared to Group 2). Alternatively, Group 3 which has the lowest performance also has the highest investment in modules (Figure 6b) and the lowest investment in design rules (Figure 6c).

The results in Figure 6c highlight the importance of investing in design rules regardless of the type of product architecture. Five to ten percent of the budget must always be assigned to design rules in order to understand the dynamics that govern relationships between modules. Managers are normally aware of investments in various product modules but for the first time we show that an equally important decision regarding investment in design rules is to be made.

5.3 Analysis and Discussion of the Dynamic Model (Hypothesis 3)

In the dynamic model, we perform the iterative procedure described earlier in Section 3.2. We consider a product network of six modules where all the modules are connected to each other. Then, we ran the static model for $n = 6$ iterations, where $B_t = 1$ in very iteration. We assume that all modules have identical complexity values (i.e., $C_i = 100$ and $k_{ij} = 50$) and the f_{ij} values were sampled from a Uniform distribution between -1 and 1, which resulted in 300 different problems. The results are summarized in Figure 7. As expected, the plots in Figure 7 show that the investment in the modules increases while the investment in the design rules decreases to zero (hypothesis 3). Note that we also performed the same procedure on product networks of size eight and ten modules and noted the same behavior depicted in Figure 7.

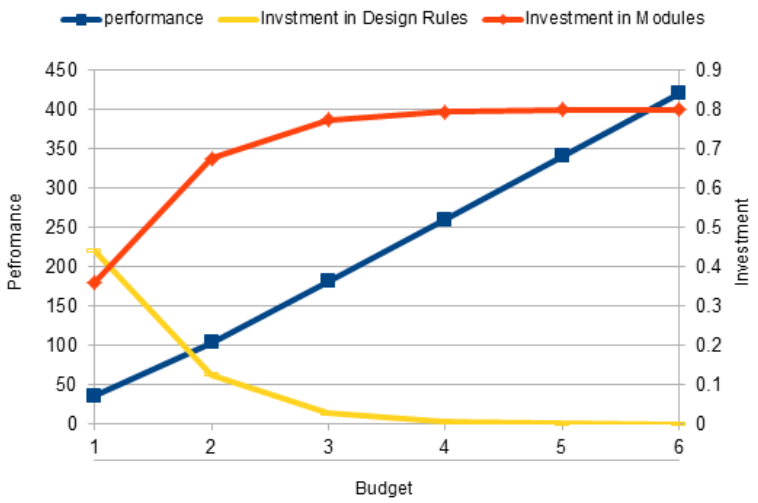


Figure 7: Performance evolution and proportion of budget spent on modules and design rules (using average values for the output of the 100 problems)

As seen in Figure 7, at some point in future time (6 generations in this particular example), organizations cannot attain further performance improvement without going to a new architecture. This represents an intriguing approach to model technological and product evolution and particularly the need for and impact of architectural innovation using our proposed model (Henderson and Clark, 1990).

Finally, when inspecting the optimal solution structure of the dynamic investment model, we observed that the budget was spent on a particular module and all dependencies emanating from it. As the budget was incremented form one generation to the next, this process continued in the same manner until all modules and their associated dependencies were addressed. It is worth noting that this investment strategy is akin of increasing the value of the system through the splitting modularity operator proposed by Baldwin and Clark (2000), where the optimization process is also splitting one module at a time (from the rest of the system) to speed-up performance evolution.

6. ASSESSMENT OF MODEL PARAMETERS AND APPLICATION EXAMPLE

In this section we describe a subjective assessment procedure for the various input parameters of the proposed model, and a validation case study. The case study involves a software product upgrade project in a company that develops Enterprise Resource Planning (ERP) systems. An

ERP system typically consists of multiple software modules that may be individually developed, customized, and purchased to fit the specific needs of a business. The software upgrade is supposed to improve various functionalities that belong to different software modules (but cover all the modules). The planned improvements were selected from a long list of prioritized functionalities, which are primarily based on internal stakeholders' assessments (i.e., marketing and customer service departments) and customer feedback.

Similar to the Wideband Delphi estimation method, our assessment proceeds in four phases: Kick-off, individual estimation, paired estimation, and review / wrap-up (Stellman and Greene, 2005). In the first phase, a one-hour kick-off meeting is conducted with the project team leader to identify the various developers involved in the project and to get acquainted with the technical details of the various modules. In the second phase, 30-minutes individual interviews are conducted with the software developers in charge of the various modules to estimate module complexity (c_i) and dependency values (f_{ij}).[19] In the third, phase, one-hour paired interviews are conducted for each non-zero f_{ij} identified in phase 2 in order to estimate the k_{ij} values. In the fourth phase, a two-hour review and wrap-up meeting with the team leader and all developers (already interviewed in phases 2 and 3) is conducted to go over the collected estimates and get concurrence. During this phase discrepancies and disagreements may arise regarding any of the estimates. To resolve such occurrences, a consensus based approach is utilized, where each estimation round for the disputed parameter is followed by discussion (of assumptions and rationale) until a consensus on its value by the developers is reached.

In the first phase, we met with the team leader who clarified that the software product under investigation is composed of eight main modules each focusing one area of the business process. The eight modules that are used in this study are:

- Human Resources Management System (HRMS): provides several human resources capabilities that include payroll, benefits management, performance management, recruitment, etc.

[19] One for each module in case more than one developer per module was identified.

- Customer Relationship Management (CRM): manages all the information that is related to engagements with customers.
- Sales are Receivables (SR): provides accounting capabilities to manage the sales and receivables such as quotations, sales order, invoicing, etc.
- Purchase and Payables (PP): provides purchasing and vendor management capabilities such as purchase transactions, compliance, approval systems, etc.
- Inventory and Warehousing (IW): provides capabilities for managing and tracking finished products inventory, parts, tools, etc.
- Financials (F): manages all information related to financial data and generates reports such as financial statements and balance sheets.
- Administration and User Configuration (AUC): used to manage and configure the ERP system.
- Reporting and Analytics (RA): Generates out-of-the-box as well as customized reports to and provides assessments of the key performance indicators.

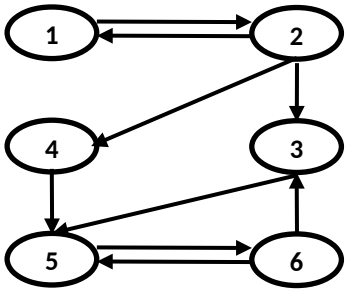
After defining the main modules of the ERP system, in the second phase, we start building the DSM. During the individual interviews, we first asked each developer about the complexity of the module they are in charge of. There a number of metrics that are typically used to assess the complexity of a software. Particularly, the developers of the ERP that we are studying use the cyclomatic complexity metric (McCabe 1976) which measures the number of linearly independent paths in the program's source code. Each module is given a normalized complexity score ranging from 0 to 100. The complexities that are used for the eight modules are 20, 50, 30, 30, 30, 50, 70, and 100 respectively. We also asked each developer about whether their work is impacted or dependent on other modules and in what ways: positively as synergistic to their work, or negatively as deteriorating for their work. Once we determined the sign for $f_{ij,}$ the magnitude was also assessed as either Low (L) with a score of 0.2, Medium (M) with a score of 0.5; or High (H) with a score of 0.7. The magnitude of each f_{ij} was decided by asking the

developers of modules *i* and *j* about how likely they have to meet and coordinate when they are updating one of the modules.

In the third phase, we conducted paired interviews between developers *i* and *j* to assess the value of k_{ij}. During these paired interviews, we first shared the f_{ij} value obtained from phase 2 interviews and inquired about the nature of this dependency. Both developers commented on the nature of the dependency and provided an estimate of k_{ij}. We used a consensus based approach, where each estimation round for k_{ij} was followed by discussion (of assumptions and rationale) until a consensus (on the value of a particular k_{ij} value) between the two developers was reached.

Recall that the k_{ij} values represent the state or amount of prior knowledge that exists about the relationship or dependency between two modules *i* and *j*. So, if there is already a substantial amount of knowledge within the organization about the nature and magnitude (i.e., physics) of this dependency, then a high value for k_{ij} is assigned. A high k_{ij} value means that a small investment in the dependency between modules *i* and *j* can eliminate this dependency or substantially reduce it. That is, if it is relatively simple to come up with a proper design rule between modules *i* and *j*, then a high value for k_{ij} is assigned. We have found empirically (through the previous simulation experiments) that a high (H) value for k_{ij} of 50 works well. Similarly, when knowledge about the nature of dependency is scarce, a low (L) of 10 for k_{ij} is assigned. A medium (M) value of 25 can also be assigned to k_{ij}. From our interviews, the developers indicated that a substantial knowledge is typically developed between the modules that have high dependency because the teams that are working on developing these modules would often coordinate. On the other hand, a lower knowledge is reached between the modules that have lower dependencies due to the lower frequency of coordination and knowledge sharing. This is rather expected particularly for a software project has matured through a number of development cycles. Therefore a $k_{ij} = 10$ is used between the modules with low dependency while $k_{ij} = 25$ and $k_{ij} = 50$ are used between the modules with medium and high dependencies respectively.

In phase 4 review and wrap-up meeting, the gathered assessments were shared with the whole development team using the schematic representation of the problem (modules and dependency structure among them) as shown in Figure 8. In our case, we did not have to conduct any additional assessments due to any disagreements between developers after reviewing all the assessments presented to them.



(a) Network representation

	1	2	3	4	5	6
1		M				
2	M					
3		H				L
4		-L				
5			-H	M		-M
6					-M	

(b) DSM representation showing f_{ij} values

	1	2	3	4	5	6
1	100	M				
2	M	100				
3		M	100			M
4		M		100		
5			M	M	100	M
6					M	100

(c) DSM representation showing c_i values (along diagonal) and k_{ij} values (off-diagonal elements)

Figure 8: Software upgrade project data

The problem (in Figure 8) was solved using our proposed optimization model and the results were to invest in the reporting and analytics module as well as in the design rules between the reporting and analytics module and each of the CRM, Sales and Receivables, Purchase & Payables, Inventory and Warehousing, and the Financials modules (α_8 = 0.55, $\theta_{1;2}$= 0.045, $\theta_{1;3,4,5}$= 0.035, and $\theta_{1;6}$= 0.045,). This recommendation was communicated to the project manager in charge of the software development process and was found to largely agree with the budget allocated by management. Particularly, the project manager revealed that the majority of the recent ERP software updates have been driven by updates to the reporting and analytics module. Investments in the other modules have been only made to accommodate the requirements of the reporting and analytics module and are thus mainly investment for compatibility. Furthermore, the project manager has been intrigued by the idea of investment in the design rules to reduce compatibility investments and revealed that over the past, the team has put little effort on upgrading the interfaces that integrate the various modules which resulted in relatively high effort being put for managing the compatibility after the various module updates.

Upon further discussion of the results with the project manager, we discovered that many of these ‘design rules’ investments were made indirectly though formal coordination meetings that were held periodically throughout the development process by the various module teams to work out solutions to avert future design decisions conflicts. In addition, there were numerous informal discussions among developers almost on daily basis. So, it is not very unusual to think about these coordination meetings and discussions as design rules’ investments. The project manager finally indicated that based on our analysis, they will explore formally investing in the components that integrate the various modules to potentially reduce development time and improve performance.

7. SUMMARY AND CONCLUSION

Senior managers, development managers, and project managers are sometimes forced to make resource allocation decisions based primarily on intuition or heuristic rules; however, the model in this paper and the resultant insights, provide a framework to follow when resource allocation decisions in product development are required. For some time, the PD literature has recognized the importance of architecture conceptually (for example, Simon (1965) and Ulrich (1995) among others). Recently, others have empirically observed this too (for example, Gopinkar et al. (2010) and Sosa et al. (2013) among others). However, few have built analytical and mathematical optimization models to relate architecture to important managerial decisions such as resource allocation. This paper is the first to suggest such a model.

This paper provides managers with a tool to allocate scarce development resources optimally by introducing a mathematical model which maximizes total product performance based on the product architecture. Such allocations can target both the modules themselves and the design rules that dictate the dependency amongst these modules. Model analyses show that development engineers and managers must understand the architecture of the product when making design and resource allocation decisions. More importantly, design and managerial guidelines that inform product development engineers and managers can result from the analysis

of this model. Specifically, for development engineers, the model and its analysis provide the ability to select the product architecture which leads to maximum performance. When development managers have no control over architectural choice, the model help these managers in deciding on the optimal budget proportions to be allocated to modules and to design rules, given a fixed architecture and budget. Finally, our analyses highlight the importance of investing in design rules regardless of the type of product architecture.

Several limitations of the proposed model can present fruitful avenues for future research. For example, our model can benefit from empirical, real world design structures and applications to validate the resultant investment profiles. One approach is to use the DSMs that have been described in the literature (e.g., Eppinger and Browning, 2012) and solve their optimization problems using the model proposed in this paper. By using one of the 'modularity' indices in the literature one can then compare the performance for various DSMs that exhibit different levels of modularity and the integral-modular dynamics as discussed earlier in this paper.

The shape of the production function can also be improved. In the current model, we assume a linear production function (for module performance) whose slope is determined by the level of investment in a particular module and also investments in all other modules that impact it. If the performance return to module investment is nonlinear (e.g., S-curve type), then the resulting optimization problem is not easily solved (Ağralı and Geunes, 2009). However, this may present a refinement to our proposed model.

Another limitation of the basic model proposed in this paper is its deterministic nature; so, dynamic investment policies need to be considered in future work; especially, if the performance is modeled stochastically. Then, the investment decisions are updated throughout the development process. That is, in each period total performance should be optimized and product development plan should be revised based on investments committed in the previous period.

REFERENCES

Ağralı, S., Geunes, J. (2009). Solving knapsack problems with S-curve return functions. *European Journal of Operational Research*, 193(2), 605-615.

Allada, V., Lan, J. (2002). New Modules Launch Planning For Evolving Modular Product Families. *Proceedings of the DECT'02: ASME 2002 Design Engineering Technical Conferences*", Montreal, Canada, Sep. 29–Oct. 2, pp. 349–358.

Anderson, P., Tushman, M. (1990). Technological Discontinuities and Dominant Designs: A cyclical model of technological change. *Administrative Science Quarterly*, 35, 604-633.

Baldwin, C., Clark, K. (2000). *Design Rules: The Power of Modularity*. Cambridge, MA, USA: MIT Press.

Barabasi, A.L. (2002). *Linked: The Science of Networks*. Cambridge, MA., USA: Perseus Publishing.

Barabasi, A. L., Albert, R., Jeong, H. (2000). Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A: Statistical Mechanics and its Applications*, *281*(1), 69-77.

Barabasi, A. L., Albert, R. (1999). Emergence of scaling in random networks. S*cience*, 286(5439), 509-512.

Bar-Yam, Y. (1997). *Dynamics of complex systems* (Vol. 213). Reading, MA: Addison-Wesley.

Braha, D., Bar-Yam, Y. (2004). The Topology of Large-Scale Engineering Problem-Solving Networks. *Physical Review E.*, 69 (1).

Braha, D., Bar-Yam, Y. (2007). The Statistical Mechanics of Complex Product Development: Empirical and Analytical Results. *Management Science*, 53 (7), 1127-1145.

Borjesson, F., Hölttä-Otto, K. (2014). A module generation algorithm for product architecture based on component interactions and strategic drivers. *Research in Engineering Design*, 25(1), 31-51.

Browning, T. R., Ramasesh, R.V. (2007). A Survey of Activity Network-Based Process Models for Managing Product Development Projects. *Production and Ops. Mgmt.* 16(2) 217-240.

Cataldo, M., Ehrlich, K. (2012). The impact of communication structure on new product development outcomes. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 3081-3090). ACM.

Christensen, C.M., Verlinden, M., Westerman, G., (2002). Disruption, disintegration and the dissipation of differentiability. Ind. Corporate Change, vol. 11, no. 5, pp. 955–993.

Clarkson, P. J., Simons, C., Eckert, C. (2004). Predicting change propagation in complex design. Journal of Mechanical Design, 126(5), 788-797.

Cohen, M. A., Eliasberg, J., Ho, T. H. (1996). New product development: The performance and time-to-market tradeoff. *Management Science*, 42(2), 173-186.

Cooper, R.G., Kleinschmidt, E.J. (1988). Resource Allocation in the New Product Process. *Industrial Marketing Management* 17, 249-262.

Crawley, E., de Weck, O., Eppinger, S., Magee, C., Moses, J., Seering, W., Schindall, J., Wallace, D., Whitney, D. (2004). The influence of architecture in engineering systems. MIT *Engineering Systems Monograph, 2006*.

Cutherell, D. (1996). Product Architecture. *The PDMA Handbook of New Product Development*, Rosenau M., Griffin A., Castellion G., and Anschuetz N. (eds), John Wiley & sons.

Dong, A., Sarkar, S. (2015). Forecasting technological progress potential based on the complexity of product knowledge. *Technological Forecasting and Social Change*, 90, 599-610.

Dutton, J. M., Thomas, A. (1984). Treating progress functions as a managerial opportunity. *Academy of Management Review*, 9(2), 235-247.

Eppinger, S. D. and Browning, T. R. (2012). *Design Structure Matrix Methods and Applications*, MIT Press, Cambridge, MA.

Ethiraj, S. K., Levinthal, D. A. (2004). Modularity and innovation in complex systems. *Management Science* 50(2), 159–173.

Ethiraj, S. K., Posen, H. E. (2013). Do product architectures affect innovation productivity in complex product ecosystems. *Advances in Strategic Management*, 30, 127-166.

Ethiraj, S. K. (2007). Allocation of Inventive Effort in Complex Product Systems. Strategic Management Journal, 28(6):563-584.

Fine, C.H., Whitney, D.E. (1999). Is the Make-Buy Decision Process a Core Competence? in Moreno Muffatto and Kulwant Pawar (eds.), *Logistics in the Information Age*, Servizi Grafici Editoriali, Padova, Italy, 1999, pp. 31-63.

Fixson, S. K., Park, J. K. (2008). The Power of Integrality: Linkages between Product Architecture, Innovation, and Industry Structure. *Research Policy*, 37(8), 1296-1316

Foster, R. (1986). The S curve: A New Forecasting Tool. Chapter 4 in Innovation, *The Attacker's Advantage*, Summit Books, Simon and Schuster, New York, 88-111.

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

Frenken, K., Mendritzki, S. (2012). Optimal modularity: a demonstration of the evolutionary advantage of modular architectures. *Journal of Evolutionary Economics*, 22(5), 935-956.

Fuge, M., Tee, K., Agogino, A., Maton, N. (2014). Analysis of collaborative design networks: A case study of openideo. *Journal of Computing and Information Science in Engineering*, 14(2), 021009.

Gokpinar, B., Hopp, W. J., Iravani, S. M. (2010). The impact of misalignment of organizational structure and product architecture on quality in complex product development. *Management Science*, 56(3), 468-484.

Greer, J., (2002). *Effort Flow Analysis: A Methodology for Directed Product Evolution Using Rigid Body and Compliant Mechanisms*. PhD Dissertation, University of Texas at Austin.

Grove, Andrew S. (1996). *Only the Paranoid Survive*. New York: Doubleday.

Henderson, R. M., Clark, K. B. (1990). Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quarterly*, 9-30.

Hölttä-Otto, K., de Weck, O. (2007). Degree of modularity in engineering systems and products with technical and business constraints. *Concurrent Engineering*, 15(2), 113-126.

Joglekar, N., Yassine, A., Eppinger, S., Whitney, D. (2001). Performance of Coupled Product Development Activities with a Deadline. *Management Science*, 47(12), 1605-1620.

Kamrad, B., Schmidt, G. M., Ulku, S. (2013). Analyzing product architecture under technological change: Modular upgradeability tradeoffs. *IEEE Transactions on Engineering Management*, 60(2), 289-300.

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

Kavadias, S., Chao, R. (2008). Resource allocation and new product development portfolio management. *Handbook of New Product Development Management*, 135.

Kreimer, A., Borenstein, E., Gophna, U., Ruppin, E. (2008). The evolution of modularity in bacterial metabolic networks. *Proceedings of the National Academy of Sciences*, 105(19), 6976-6981.

Krishnan, V., Ulrich, K. (2001). Product Development Decisions: A Review of the Literature. *Management Science*, 47(1) 1-21.

Langlois, R. and P. Robertson (1992). Networks and Innovation in a Modular System: Lessons from the Microcomputer and Stereo Component Industries. *Research Policy*, 21(4), 297-313.

Loch, C. H., Terwiesch, C. (1998). Communication and Uncertainty in Concurrent Engineering. *Management Science*, 44(8), 1032-1048.

Luo, J. (2015). A simulation-based method to evaluate the impact of product architecture on product evolvability. *Research in Engineering Design*, 26(4), 355-371.

Martin, M. V., Ishii, K. (2002). Design for variety: developing standardized and modularized product platform architectures. Research in Engineering Design, 13(4), 213-235.

McCabe, T. J. (1976). A complexity measure. Software Engineering, IEEE Transactions on, (4), 308-320.

Meier, C., Yassine, A. A., Browning, T. R. (2007). Design process sequencing with competent genetic algorithms. *Journal of Mechanical Design*, *129*(6), 566-585.

McNerney, J., Farmer, J. D., Redner, S., Trancik, J. E. (2011). Role of design complexity in technology improvement. *Proceedings of the National Academy of Sciences*, 108(22), 9008-9013.

Mihm, J., Lock, C., Huchzermeier, A. (2003). Problem-Solving Oscillations in Complex Engineering Projects. *Management Science*, 49 (6), 733-750.

Newman, M. (2010). *Networks: An Introduction*, Oxford University Press, Oxford, NY.

Newman, M. E. (2003). The structure and function of complex networks. *SIAM review*, *45*(2), 167-256.

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

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

Sharman, D., Yassine A., (2004). Characterizing Complex Product Architectures. *Systems Engineering*. 7(1), 35-60.

Simon, H. A. (1965). The architecture of complexity. *General systems*,*10*(1965), 63-76.

Simpson, T. (2004). Product platform design and customization: Status and promise. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* (AI EDAM) 18(1), 3-20.

Smith, R. P., Eppinger, S.D. (1997). Identifying Controlling Features of Engineering Design Iteration, *Manage. Science*, 43(3), 276-293.

Sosa, M., Mihm, J., Browning, T. (2013). Linking Cyclicality and Product Quality. *Manufacturing & Service Operations Management*, 15(3), 473-491.

Sosa, M., Mihm, J., Browning, T. (2011). Degree distribution and quality in complex engineered systems. *Journal of Mechanical Design*, 133(10), 101008.

Sosa, M. E., Eppinger, S. D., Rowles, C. M. (2007). A network approach to define modularity of components in complex products. *Journal of Mechanical Design*, 129(11), 1118-1129.

Stellman, A., Greene, J. (2005). *Applied Software Project Management*. O'Reilly Media, Inc.

Strogatz, S. (2001). Exploring Complex Networks. *Nature* 410(8 March):268-276.

Anderson, P., Tushman, M. (1990). Technological Discontinuities and Dominant Designs: A cyclical model of technological change. *Administrative Science Quarterly*, 35, 604-633.

Ülkü, S., Schmidt, G. (2011). Matching product architecture and supply chain configuration. *Production and Operations Management*, 20(1), 16-31.

Ulrich, K., (1995). The Role of Product Architecture in the Manufacturing Firm. *Journal of Research Policy*, 24, 419-440.

Ulrich, K., Eppinger, S.D. (2008). *Product Design and Development*. New York, NY: McGraw-Hill/Irwin.

Van Wie, M. J., Greer, J. L., Campbell, M. I., Stone, R. B., & Wood, K. L. (2001, September). Interfaces and product architecture. In *Proceedings of DETC* (Vol. 1, pp. 9-12).

Valverde, S., Cancho, R. F., & Sole, R. V. (2002). Scale-free networks from optimal design. *Europhysics Letters*, 60(4), 512.

Wasserman, S., Faust, K. (1994). *Social Network Analysis*. Cambridge University Press, New York.

Watts, D. J. (1999). *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton, N.J.: Princeton University Press.

Whitney, D. E. (2008). Network Models of Mechanical Assemblies. In *Unifying Themes in Complex Systems* (pp. 331-338). Springer Berlin Heidelberg.

Whitney, D. E. (2005). *Degree correlations and motifs in technological networks*. Working paper, ESD-WP-2005-10. Massachusetts Institute of Technology, Engineering Systems Division.

Whitney D. E. (2004). *Physical Limits to Modularity*. Working paper, ESD-WP-2003-01.03-ESD, Massachusetts Institute of Technology, Engineering Systems Division.

Yassine A., Joglekar N., Braha, D., Eppinger S.D., Whitney D. (2003). Information Hiding in Product Development: The Design Churn Effect. *Research in Engineering Design*. 14(3), 131-144.

Yassine, A., Braha, D. (2003). Complex Concurrent Engineering and the Design Structure Matrix Method. *Concurrent Engineering: Research & Applications* 11(3) 165-176.

Yassine, A., Falkenburg, D., Chelst, K. (1999). Engineering design management: an information structure approach. *International Journal of production research*, 37(13), 2957-2975.

Yu, T. L., Yassine, A. A., Goldberg, D. E. (2007). An information theoretic method for developing modular architectures using genetic algorithms.*Research in Engineering Design*, *18*(2), 91-109.

Zacharias, N., Yassine, A. (2008). Optimal platform investment for product family design. *Journal of Intelligent Manufacturing*, 19(2), 131-148.