20TH INTERNATIONAL DEPENDENCY AND STRUCTURE MODELING CONFERENCE, DSM 2018

TRIESTE, ITALY, OCTOBER 15 – 17, 2018

Experimenting with the NK and DSM Models

Reem Ali Ahmad, Ali Yassine
Department of Industrial Engineering & Management,
American University of Beirut, Beirut, Lebanon

Abstract: The theory of complex systems, which has been applied successfully in evolutionary biology, is gaining popularity for the modeling and analysis of complex product development (PD) systems. Modeling complex PD systems is essential to understand how system elements and their dependencies impact system properties in several aspects such as performance, convergence, and evolution. In this paper we use the NK and NKC models to simulate and analyze complex PD systems, which are represented by the design structure matrix (DSM). The main objective is to assess whether these models can be useful in analyzing DSMs; particularly, assessing the effect of architecture on performance and evolution.

Keywords: NK Model, Design Structure Matrix (DSM), Product Development, Complex Systems, Performance Evaluation

1 Introduction

The theory of complex systems, which has been applied successfully in evolutionary biology (to study the dynamics and evolution of biological systems), is gaining popularity in product development (PD) to model and analyze man-made systems (e.g., Frenken and Mendritzki, 2012; Oyama et al., 2015). In fact, the biological domain is considered an analogy to complex PD systems where the genes in biological organisms correspond to the components in complex PD systems and genes in biological organisms depend on each other in a similar way to the components in man-made systems. Complexity of biological organisms is reflected by the dependencies between the genes; that is, when one gene is mutated, it may not just affect its own functionality but also affects the functionality of all other interdependent genes (Frenken, 2006). The main difference between the two systems is that man-made systems are designed by designers who are responsible for making the design decisions whereas biological systems depend on natural selection (Beesemyer et al., 2011).

This analogy between biological organisms and man- made complex systems is valid in terms of product evolution as well. Products evolve throughout the generations due to the continuous changes in the interdependent components' design, which increases the systems' performance. It has been argued that the way these interdependencies are distributed between the system's components (i.e. product architecture) affects the product's performance and evolvability (Rivkin and Siggelkow, 2007; Luo, 2015). In this context, modeling complex PD systems is essential to understand how the system elements and their dependencies impact system properties in several aspects such as performance, cost, improvement, convergence, and evolution.

According to the NK model, a product system can be defined as a complex system consisting of a set of N components (or modules), each of which is intended to deliver a specific functionality (Kauffman, 1993). Hence, each component delivers a specific function and, in turn, contributes some value to the overall product system. This value is

referred to by the performance or the fitness value of the component. This component's performance depends on its own (design) decision and the decisions of one or more other components (depending on the system architecture). The decisions made at the component level are binary. That is, each component is available in two variants, which represent two alternative designs. A complete product contains exactly one variant of each component. A vector of length N whose i^{th} element represents a variant of the i^{th} component is called a design configuration. Standard practice in the NK literature denotes the variants by 0 and 1, which allows a configuration to be represented by a binary string (e.g., 0010 for a vector of length $N = 4$). The N-dimensional possibility space is called the design space and a specific component configuration defines a product design. Moreover, a complete product has a corresponding product (system) fitness that depends on the fitness values of its components. The actual resemblance of this product fitness is a measure of the performance of the system as a whole. For example, if the system is a team of employees, then the fitness of the system resembles the problem-solving effectiveness of this team (Solow et al., 2000).

In this paper we introduce an NK-based simulation model to analyze the design structure matrix (DSM) to assess the effect of the product architecture on product performance. In the next section, we introduce the basics of the NK model, and then we test its behavior based on varying N and K values. In Section 3, we introduce the notion of NK model using sub-blocks. In Section 4, we introduce the NKC model and run tests to compare its performance to the standard NK model. We test the various NK models on a set of different systems architectures in Section 5. We conclude the paper in Section 6.

2 NK Model Fundamentals

In the NK model we consider a system of *N* components, where each component depends on K other components (Kauffman, 1993). The NK Model is a mathematical representation of these dependencies, i.e. it assigns to each component a mathematical measure that represents the component's fitness value, taking into account the dependencies between components, as will be explained later in this section. To apply the NK model to the N size system, a N size Design Structure Matrix (DSM) is used to model and represent the system and its components' dependencies, as illustrated in Figure 1.

Suppose we have 3 components in a system, where the performance of each component depends on its own (design) decision and on the decisions of other components. In this case, N=3 and K=1.

	1	2	3
1		X	
2			X
3	X		

Figure 1: DSM Representation of a Complex System

Figure 1 represents the scenario where each off-diagonal mark "X" represents a dependency between two components (Yassine and Braha, 2003). For example, the DSM

(assuming that row i depends on column j) shows that the performance of component 1 depends on its own (design) decision and the decisions of component 2. Similarly, component 2 depends on component 3, and component 3 depends on component 1.
The NK model starts by randomly assigning to each of the *N* components discrete random states (either 0 or 1) and corresponding random fitness values sampled from a uniform distribution ranging between 0 and 1. The fitness of the system, call it F_1, is the average of the fitness values of the N components and can be calculated according to the formula in Equation (1).

$$F_1 = \sum f_i / N \quad (1)$$

Where f_i is the fitness value of component i. In our case, shown in Figure 1, *i* ranges between 1 and 3 ($1 \leq i \leq 3$) since there are 3 components in the system.

Then, one of the *N* components is randomly chosen to change its state and its corresponding fitness value. Furthermore, we change the fitness values of all the components that depend on this chosen component. For example, if we choose to change the state of component *i* ($1 \leq i \leq N$), then if its state is 0 it becomes 1 and vice-versa. Then, we change the fitness value of component *i* as well as the fitness values of all the components *j* ($1 \leq j \leq N$) that depend on component *i.*

Finally, the average fitness is recalculated, to obtain a new average fitness, call it F_2. If F_2 is greater than F_1, then we repeat the above process starting with the new obtained string of states and their corresponding fitness values. If F_2 is less than F_1, then we repeat the above process after choosing a component other than one previously chosen. This simulation process continues until a maximum average fitness is reached. Note that if a string of states is revisited, then their corresponding fitness values should be retained.

2.1 NK Model Simulation

For the DSM in Figure 1, the NK model works as follows. After randomly initializing the states and the fitness values of these components, we obtain initial states 110 and their corresponding fitness values 0.85, 0.57 and 0.63, resulting in an initial average fitness F_1=0.68 (Refer to the 7th row in Table 1). Then, the third component is randomly chosen so its state changes from 0 to 1 and its corresponding fitness value as well as that of component 2 change to 0.02 and 0.55 respectively, resulting in the 8th row in Table 1.

Table 1: Enumeration of the fitness values of the DSM in Figure 1

	States	f_1	f_2	f_3	F
1	**000**	0.31	0.72	0.37	0.47
2	**001**	0.31	0.42	0.51	0.41
3	**010**	0.38	0.57	0.37	0.44
4	**011**	0.38	0.55	0.51	0.48
5	**100**	0.15	0.72	0.63	0.5
6	**101**	0.15	0.42	0.02	0.2
7	**110**	0.85	0.57	0.63	0.68
8	**111**	0.85	0.55	0.02	0.47

This iteration results in the new average fitness $F_2=0.47<0.68=F_1$. For this, we return to the initial '110' string states and randomly choose a new component, i.e. any component other than the 3rd component. This process is repeated until a maximum average fitness F_{max} is reached. Table 1 enumerates the total 8 cases of this DSM. It is worth noting that the fitness values are almost always between 0.5 and 0.7 since we are sampling from a Uniform distribution between 0 and 1.

2.2 Effect of Varying K on the fitness values in the NK Model

To study the effect of K on the evolution of the fitness values, we ran the NK model on three DSMs of size 5, but with different number of dependencies K: a) K=0, b) 0<K<N-1 and c) K=N-1. The variation of the fitness values in these 3 cases is shown in Figure 2.

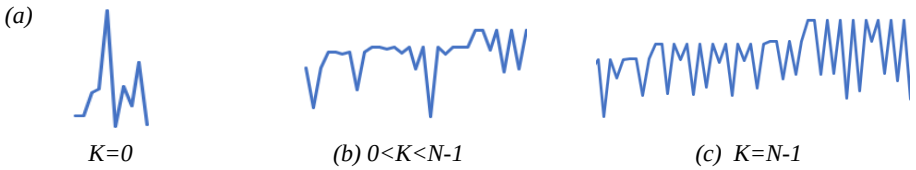


Figure 2: Evolution of the Fitness for Various Values of K (N=5)

Figure 2a represents the case where K=0, i.e. the system has no interactions among its components. In this case, there will only be one state (either 0 or 1) for each element that is responsible for making the highest fitness contribution to the system. This maximum fitness, i.e. the only global optimum, is represented by the highest single peak in Figure 2a. All other sub optimal fitness values will eventually reach the global optimum after having passed through all their neighboring states, which obviously have lower fitness than the global optimum.

We notice from Figure 2b that as the number of dependencies increases to take any value between 0 and N-1 (K=2 in our case), the number of fluctuations increases, and the graph becomes multi-peaked. In this case, each element depends on multiple other elements in the system, causing the number of the local optima to increase significantly and thus making it harder for each element to reach an optimum.

In the third case, as K reaches it maximum value, i.e. K=N-1 (K=4 in our case), the DSM become a completely rugged landscape where each element depends on all other elements in the system. This property causes the search process for the maximum fitness to be very difficult, as represented by the huge increase in the peaks of the graph, in Figure 2c.

2.3 Effect of N and K on the NK Model

To study the effect of the number of elements N and number of dependencies K on the system's behavior, the NK model is applied on several DSMs having different N and K. The corresponding changes in the fitness values and number of iterations are observed and shown in Figure 3.

Observation 1: As shown in Figure 3, for a fixed *N* (the number of components) and as *K* (the number of dependencies) increases, both the fitness and the number of iterations

are not significantly affected. However, both the fitness and the number of iterations increase with *N* for a fixed *K*.

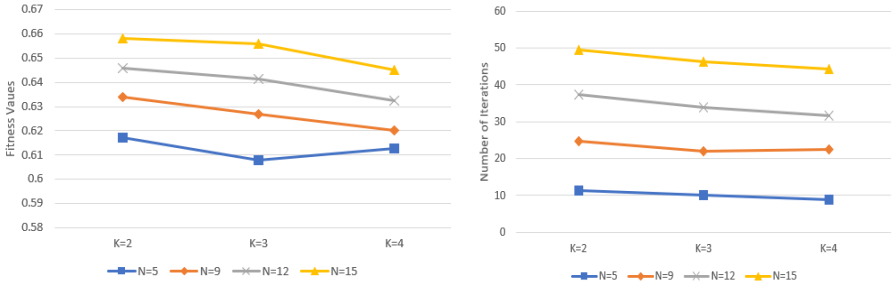
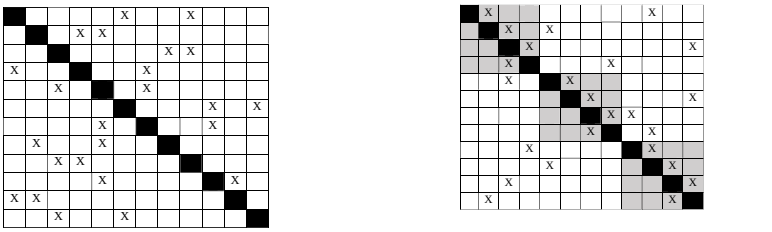


Figure 3: Effect of N and K on the DSM'S behavior

3 NK Model using Sub-blocks

The NK model is also applied in this section; however, the DSM is divided into sub-blocks prior to simulation. In this case, K is divided into two components; K_i and K_o, where $K_i+K_o=K$, K_i is defined as the number of dependencies within the same sub-block, and K_o as the number of dependencies outside the sub-block. For example, consider Figure 4b, where a DSM of size 12 and K=2 (K_i =1 and K_o=1), is divided into three sub-blocks of four components each.

Both Figures 4a and 4b have the same number of components N and dependencies K; however, the main difference is the way these dependencies are distributed. In Figure 4a, interactions between components are randomly distributed, however, in Figure 4b, they are classified according to the number of dependencies within and outside each sub-block, as described above. For example, the first DSM row has 2 marks (i.e. K=2). One of these marks is within the first block in the grey part of the row (since K_i =1) and the other mark is within the white part of the first row (since K_o=1). The rest of the marks are similarly allocated for each row in the DSM.



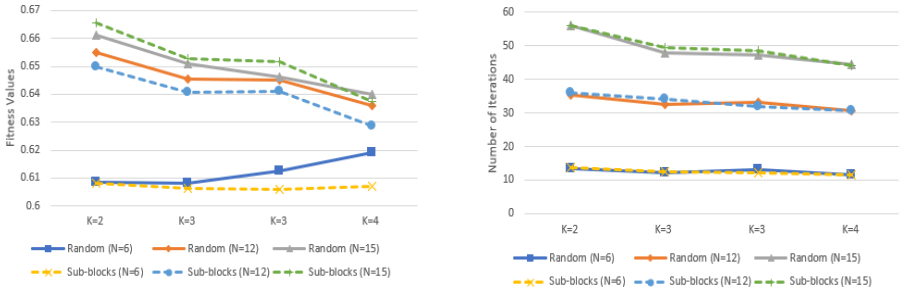
(a) 12 sized Random DSM

(b) 12 sized DSM with sub-blocks

Figure 4: Sample of 12 sized DSMs having different dependencies' distribution

3.1 Effect of N and K on Random NK model and NK model with Sub-blocks

To study the behavior of the DSM with sub-blocks and test how it differs from the random DSM, the NK model is tested for 200 runs on both random and sub-blocks DSMs. This test is applied on DSMs with different number of components (N=6, 12 and 15) and dependencies (K=2, 3 and 4) to compare the maximum average fitness values and the average number of iterations executed by different cases. Also, note that two cases have been considered for K=3; either K_{in} =1 and K_{out} =2 or K_{in}=2 and K_{out}=1.



(a) Variation of the fitness Values

(b) Variation of the number of iterations

Figure 5: Variation of the fitness and number of iterations of the random and sub-blocks DSMs as a function of K

Observation 2: As shown in Figure 5, both random and sub-blocks DSMs behave similarly with an increase in K. We can conclude that the effect of distributing the dependencies between the components using sub-blocks is almost negligible on the system's fitness and number of iterations.

3.2 NK Models with Different Numbers of Sub-blocks

Each DSM, with a certain number of components N, can be divided into different number of sub-blocks. For example, the 12 sized DSM, shown in Figure 4b, is divided into three sub-blocks of 4 components each; however, it can be divided into 2 sub-blocks of 6 components each, or into 4 sub-blocks of 3 components each, etc. Accordingly, we tested the NK model on a 12 sized DSM, with K=4, divided into different number of sub-blocks to study the effect on the fitness values and the number of iterations. We observed that changing the number of sub-blocks did not significantly impact the fitness values nor the number of iterations.

4 NKC Model Fundamentals

In the NKC model, we consider a system of size *N*, but with *S* subsystems (Hordijk and Kauffman, 2005), where:

- N: number of total components that are distributed along S sub systems
- K: number of inter dependencies inside the sub system
- C: number of external dependencies, that is each component in each sub system depends on C other components from other sub systems
- Nj': number of components inside subsystem j. Note that the number of components within a one subsystem may differ from the number of components in another subsystem

We start by randomly assigning discrete random states (either 0 or 1) and random fitness values sampled from a Uniform distribution ranging between 0 and 1 to all components in all subsystems. Then, a random subsystem *j* ($1 \leq j \leq S$) is selected and a component *i* from subsystem *j* is randomly chosen to change its state and its corresponding fitness value. Next, we randomly sample for the fitness of all components in subsystem *j* that depend on component *i*. The average fitness of subsystem *j* is calculated as follows in Equation 2:

$$F_j = \sum fi / N_j' \quad (2)$$

where f_i represent the fitness value of component *i* in subsystem *j*.

If the new average fitness of subsystem *j* is greater than the previous average fitness, then we sample for the fitness values of components, in subsystems other than subsystem *j*, which depend on component *i*. While if the new average fitness of subsystem *j* is lower than the previous average fitness, we chose another component from subsystem *j*.

These steps are repeated until a maximum average fitness of subsystem j is reached. After applying the above scenario for all subsystems S, the maximum average fitness of the whole system is calculated as: $F = \sum F_j / S$ (3)

4.1 Examining the Difference between NK and NKC Models

To notice the difference between the NK and NKC models, both models were run in parallel for 1000 runs on the 12 sized DSM, represented in Figure 4b. The variables of this DSM, represented in both NK and NKC models, are shown in Table 2.

Table 2: Variables of the DSM in Figure 4b in NK and NKC Models

	NK Model	**NKC Model**
N	12	12
K	2	1
C	0	1
S	1	3
N'	-	4

The average fitness values and average number of iterations of the 1000 runs are recorded in Table 3. The simulated 1000 maximum fitness values and number of iterations are displayed in Figure 6.

Table 3: Average maximum fitness and number of iterations in along the 1000 runs

	NK Model	NKC Model
Average Maximum Fitness	0.6505	0.5874
Average Number of Iterations	32.99	24.468

Observation 3: As shown in Table 3 and Figure 6, the NKC model reach a lower maximum fitness, on average, than the NK model and at a lower average number of iterations as well. However, along the 1000 runs, there is not a clear relation between the maximum fitness of the NK and NKC models in each run. As for the number of iterations, the NKC model clearly takes less iterations than the NK model, almost throughout all the 1000 runs, as shown in Figure 6 (right).

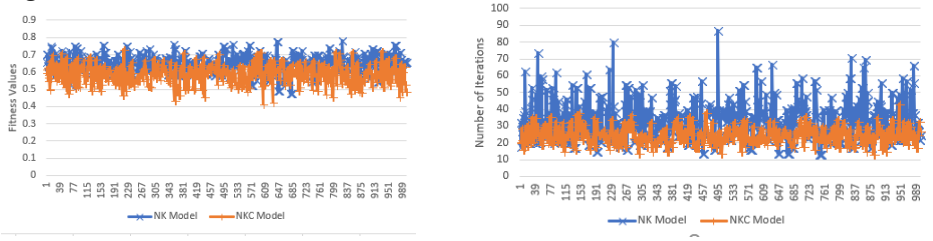
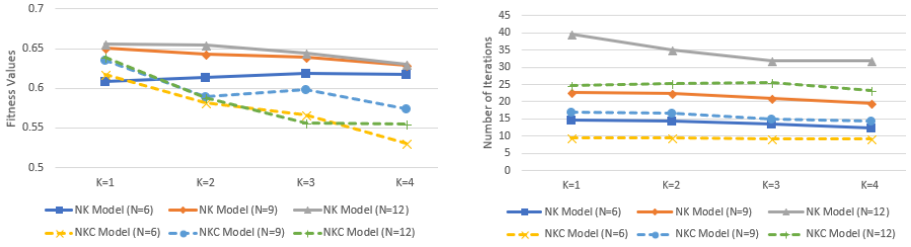


Figure 6: Fitness Values (Left) and Number of Iterations (Right) in NK & NKC Models

4.2 Effect of N and K on the Performance of NK and NKC models

To test the effect of changing N and K on the system's performance in each of the NK and NKC models, both models are applied on DSMs of different sizes (N=6, 9 and 12) and different dependencies (K=1, 2, 3 and 4). Note that changing the number of dependencies in the NKC model is done by either increasing the number of internal dependencies K or the number of external dependencies C.



(a)Variation of the average maximum fitness

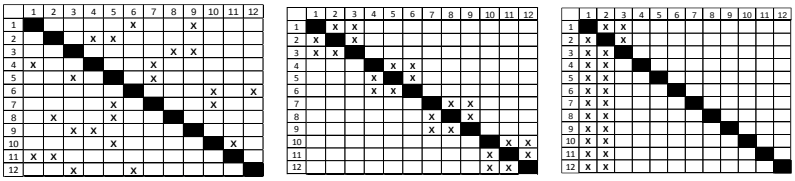
(b)Variation of the avg. no. of iterations

Figure 7: Variation of the average max. fitness as a function of K in the NK & NKC models

Observation 4: As shown in Figure 7, as the number of dependencies K increase, the (average) maximum fitness of both the NK and NKC models generally decrease. In the NK model, the decrease occurs slowly as K increases and the curve somehow remains flat, however the fitness in the NKC model decreases at a faster rate, and this is clear from the slopes that appear to be steeper in the NKC model.

5 Effect of N, K and Architecture on NK and NKC Models

In this section, we perform a comprehensive study in which we test both, the NK and NKC models, on different numbers of elements N (12 and 16), different number of dependencies K (1,2 and 3) and different architectures (Random, Block-Diagonal, and Centralized).

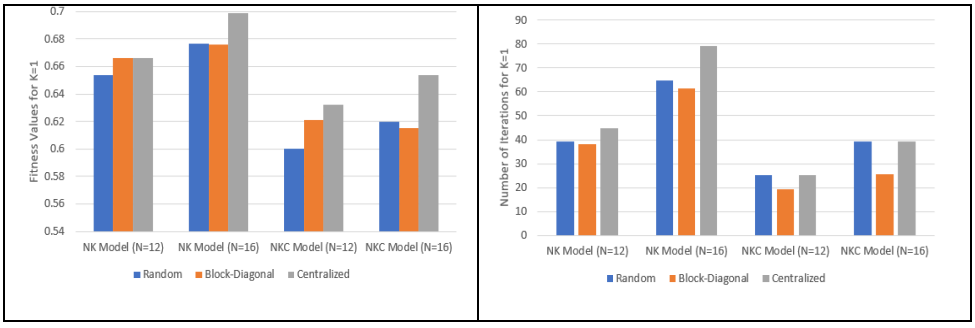


(a) Random *(b) Block Diagonal* *(c) Centralized*

Figure 8: Sample DSM architectures (N=12, K=2)

Sample DSMs of the different architectures is shown in Figure 8. The results of this test (fitness and number of iterations), for each of the three architectures, in the NK and NKC models are presented in Figure 9.

Observation 5: When comparing the fitness values of the DSMs of different architectures in Figure 9 (left-side panels) it is noticed that the Random DSM almost has the lowest fitness values for both values of N=12 and N=16 in the NK and NKC models. On the other hand, we can see that the Centralized DSM always has the highest fitness values. As for the Block-Diagonal DSM, its fitness values vary between those of the Random and Centralized DSMs.



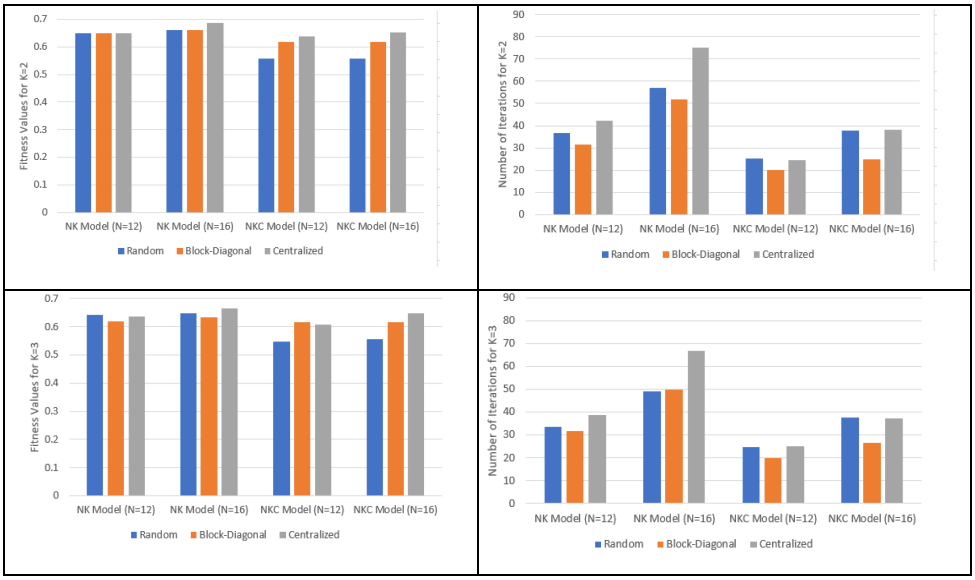


Figure 9: Variation of the fitness values (left-side panels) and the number of iterations (right-side panels) as a function of N in NK and NKC models for different DSM architectures

In Figures 9 (right-side panels), we can see that the Centralized DSM execute the highest number of iterations, whereas the Block-Diagonal takes the lowest number of iterations for both values of N=12 and N=16 in the NK and NKC models. It is noticed that the difference in the number of iterations executed between the three architectures increase in each of the NK and NKC models as N increases, i.e. the difference in the number of iterations between the three architectures is greater when N=16 than when N=12. Also, when comparing the variation of the number of iterations for the different values of K (Figures 9 (b), (d), (f)), it is noticed that the behavior and pattern of variation is the same.

6 Summary and Conclusion

In this paper, we experimented with the NK and NKC models to investigate their utility in the analysis of PD systems represented by DSM models. We tested various parameters (in the NK model) that may impact the system's performance evolution, mainly the number of components in the system, the number of dependencies between these components, and the system's architecture.

We found that as K increases, the fitness is mostly unaffected; however, the process of searching for the maximum fitness becomes harder due to having multiple local optima (when $0<K<N-1$) rather than one global optimum (when $K=0$). Also, as N increases, we noticed that the number of iterations increase, despite the number of dependencies K. As for the fitness, it increases with N, provided that we are comparing for the same value of K. Finally, we concluded that if the components randomly interact with each other, the system's fitness and number of iterations will be smaller than the case when the elements

depend on each other in a structured way (such as Block-Diagonal and Centralized DSMs).

The standard NK and NKC models can be useful for the analysis of PD systems only if some adjustments are made, which relate to the difference between biological systems and man-made (engineered) systems. First, performance in engineered systems is not random and should be proportional to allocated effort. Second, choice of the component to work on is also not random but a deliberate choice is made by the development team. Third, specifying the type of dependencies between the components; that is, when the performance of one component increases, the performance of its dependent components may increase or decrease depending on the nature of this dependency. Also, time and cost implications of the evolution process is not taken is not account. Finally, PD projects have a budget allocated to them and scheduled deadlines to meet. Hence, cannot evolve freely until maximum fitness is reached.

References

Beesemyer, J. Clark, et al., 2011. Developing methods to design for evolvability: research approach and preliminary design principles, 9th Conference on Systems Engineering Research.

Frenken, Koen, and Stefan Mendritzki., 2012. Optimal modularity: a demonstration of the evolutionary advantage of modular architectures. Journal of Evolutionary Economics 22, no. 5: 935-956.

Frenken, Koen., 2006. A fitness landscape approach to technological complexity, modularity, and vertical disintegration. Structural Change and Economic Dynamics17, no. 3: 288-305.

Hordijk, Wim, and Stuart A. Kauffman., 2005. Correlation analysis of coupled fitness landscapes. Complexity 10, no. 6: 41-49.

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

Luo, Jianxi., 2015. A simulation-based method to evaluate the impact of product architecture on product evolvability. Research in Engineering Design 26, no. 4: 355-371.

Oyama, Kyle, Gerard Learmonth, and Raul Chao., 2015. Applying complexity science to new product development: modeling considerations, extensions, and implications. Journal of Engineering and Technology Management 35: 1-24.

Rivkin, Jan W., and Nicolaj Siggelkow., 2007. Patterned interactions in complex systems: Implications for exploration. Management Science 53, no. 7: 1068-1085.

Solow, D., Burnetas, A., Tsai, M. C., & Greenspan, N. S., 2000. On the expected performance of systems with complex interactions among components. *Complex Systems*, *12*(4), 423-456.

Yassine, Ali, and Dan Braha., 2003. Complex concurrent engineering and the design structure matrix method. Concurrent Engineering 11, no. 3: 165-176.

Contact: **Professor Ali A. Yassine,** American University of Beirut, Department of Industrial Engineering & Management, Hamra-Bliss Street, PO Box 11-0236, Beirut, Lebanon, 00-961-1-350-000 Extn. 3439, ali.yassine@aub.edu.lb