

The Value, Costs and Organizational Consequences of Modularity

Carliss Y. Baldwin
Kim B. Clark

May, 2003

Our thanks to Datta Kulkarni and Robin Stevenson for many conversations that have informed our work over the past several years, as well as comments and suggestions that greatly improved this presentation. Thanks also to Barbara Feinberg, who over many years and countless discussions helped us to develop and refine our ideas.

An earlier version of this talk was prepared as the keynote address for the Opening Conference of the Research Institute of Economy, Trade and Industry (RIETI), "Modularity—Impacts to Japan's Industry," Tokyo, Japan, July 12-13, 2001. Our thanks to the sponsors and participants of the conference, especially Masahiko Aoki and Sozaburo Okamatsu, Nobuo Ikeda, Takahiro Fujimoto, Hiroyuki Chuma, Jiro Kokuryo, Noriyuki Yanagawa, Nobuo Okubo, Shuzo Fujimura, Hiroshi Hashimoto, Hiroshi Kuwahara, and Keiichi Enoki for extremely stimulating discussion and pertinent critiques.

We alone are responsible for errors, oversights and faulty reasoning.

Copyright © Carliss Y. Baldwin and Kim B. Clark, May 2003.

Working papers are in draft form. This working paper is distributed for purposes of comment and discussion only. It may not be reproduced without permission of the copyright holder. Copies of working papers are available from the author.

Introduction

In the last decade, the concept of modularity has caught the attention of engineers, management researchers and corporate strategists in a number of industries. When a product or process is “modularized,” the elements of its design are split up and assigned to modules according to a formal architecture or plan. From an engineering perspective, a modularization generally has three purposes:

- To make complexity manageable;
- To enable parallel work; and
- To accommodate future uncertainty.

Modularity accommodates uncertainty because particular elements of a modular design may be changed after the fact and in unforeseen ways as long as the design rules are obeyed. Thus, within a modular architecture, new module designs may be substituted for older ones easily and at low cost.

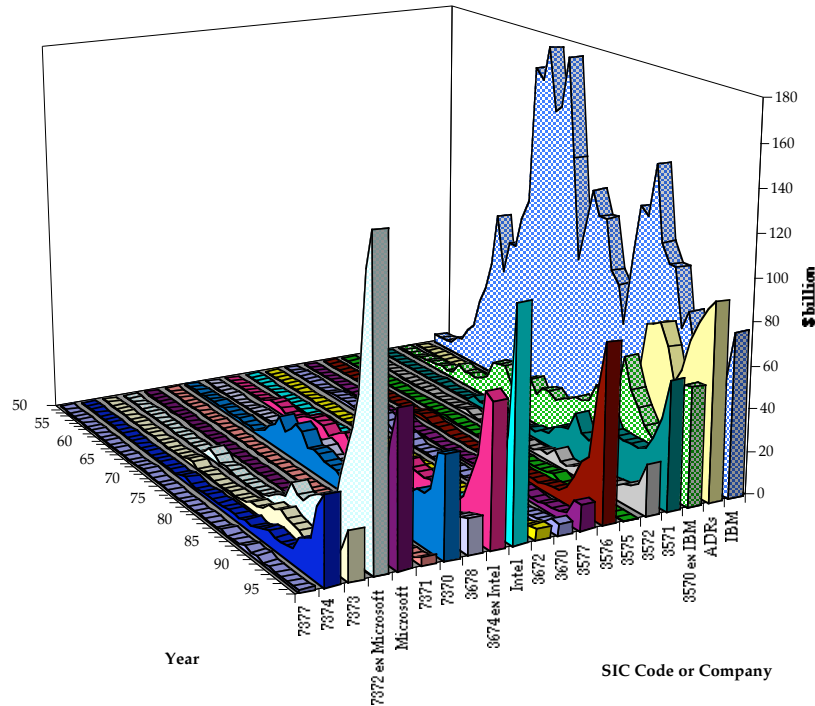
In this talk, I will cover three main points. First, we will see that *modularity is a financial force* that can change the structure of an industry. Then, we will explore the *value* and *costs* that are associated with constructing and exploiting a modular design. In particular, we shall see that modularity is not “free.” Finally we will examine the ways in which modularity shapes organizations and the risks that it poses for particular firms.

Modularity as a Financial Force

To indicate the financial power of modularity, let us begin by looking at some data from the computer industry. Figure 1 is a graph of the market values (in 1996 constant US dollars) of substantially all the public corporations in the computer industry from 1950 to 1996, broken out into sixteen subsectors.¹ The chart tells a story of industry evolution that runs counter to conventional wisdom. The dominant theories of industry evolution—backed up by a great deal of supporting empirical evidence—describe a process of pre-emptive investment by large, well-capitalized firms, leading to stable market structures and high levels of concentration over long periods of time.² The data in Figure 1, by contrast, show that there was indeed a period in which the computer industry was highly concentrated, with IBM playing the role of dominant firm. (IBM’s market value is the blue “mountain range” that forms the backdrop of the chart.) But we can also see that in the 1980s, the computer industry “got away” from

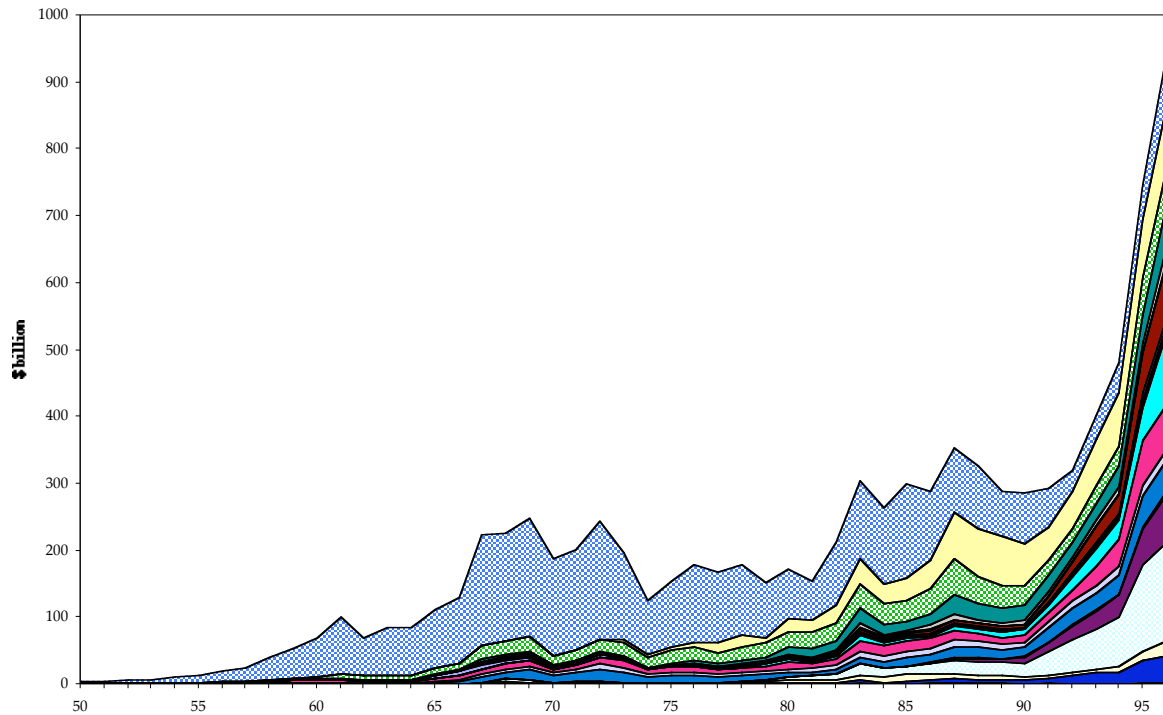
IBM. In 1969, 71% of the market value of the computer industry was tied up in IBM stock. But by 1996, no firm accounted for more than 15% of the total value of the industry.

Figure 1
The Market Value of the Computer Industry
By sector, 1950-1996 in constant 1996 US dollars



The pattern in Figure 1 reveals births of firms and industries as well as deaths of firms. The database consists of about 1,500 firms. Of the 1,500 firms that entered, about 500 no longer exist. The figure also shows that whereas value initially was very closely concentrated in a few firms, it has migrated over time and is now spread out over a very large number of firms across sixteen different industry categories. The third thing this figure shows can be viewed more easily if you stack the values one on top of the other, as in Figure 2. There you can see that this group of firms has experienced significant value increases over time. That is, as value was migrating across many, many firms, the aggregate value rose dramatically.

Figure 2
The Market Value of the Computer Industry
 Aggregated, 1950-1996, in constant 1996 US dollars



So we have seen here a pattern revealing more firms, more products, and more value created over time. Indeed, the computer industry today consists of a large *cluster* of over 1000 firms, no one of which is very large relative to the whole. In addition, the total market value of the industry is spread quite uniformly across the sixteen sub-industries. Modularity is what allowed this *creation* of value as well as the *dispersion* of value across so many firms to take place. Let me explain what I mean.

I'll begin by noting that my purpose here is not to be an advocate of modularity per se. Rather, I advocate the *rigorous mapping, measurement, and analysis* of modularity. Modularity is not appropriate to all products and all businesses at all times. It is, however, a potentially valuable characteristic of artifacts and of businesses that can be mapped, measured, tracked, and studied. And through mapping the modularity of structures and analyzing the value and costs of modularity in specific contexts, we can learn a great deal about what aspects of modularity contribute to good or bad economic performance in products, firms and industries.

For example, there is a fundamental modularity in computer designs that has caused the computer industry to evolve to its present cluster structure. The connections among products and companies are quite complicated in this cluster. Most firms do not design and make whole computer systems; instead, they design or make *modules* that are parts of larger systems.

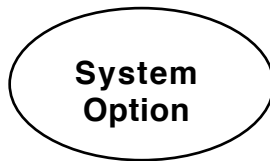
Modules, in fact, are always distinct parts of a larger system. They are designed and produced independently of one another, but function together as a whole. Modularity allows tasks—both design tasks and production tasks—to be divided among groups, which can work independently and do not have to be part of the same *firm*. (We'll see shortly how a system can be modularized.) Put briefly, compatibility among modules is ensured by “design rules” that govern the architecture, the interfaces, and the standardized tests of the system. Thus “modularizing” a system involves specifying its *architecture*, that is, what its modules are; specifying its *interfaces*, i.e., how the modules interact; and specifying a set of *tests* that establish that the modules will work together and how well each module performs its job.

Modularity in a system does many things. First, a modular structure makes complexity manageable by providing a *well-designed* “division of cognitive labor,” (to use Williamson’s and Aoki’s phrase³). Second, modularity organizes and enables parallel work: work on modules can go on simultaneously without ongoing coordination among them. Finally, a modular architecture allows module designs to be changed and improved over time without undercutting the functionality of the system as a whole. In this sense a modular architecture is “tolerant of uncertainty” and “welcomes experimentation” in the design of modules.

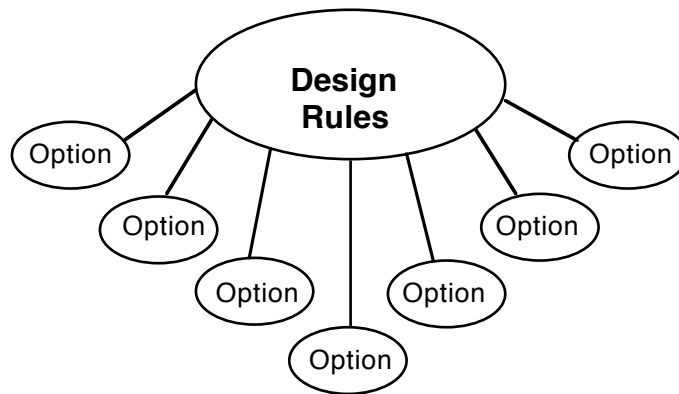
New designs in turn have “option-like” properties. According to modern finance theory, an option is “the right but not the obligation” to choose a course of action and obtain an associated payoff. In the case of designs, the “optional” course of action is the opportunity to implement a new design. *But the new way does not have to be adopted*, and indeed, it should only be adopted if it is better than the alternatives. This in turn means that the economic value of a new design is properly modeled as an option. The field of finance has established that options are valuable. Therefore, modularity creates valuable options, as can be seen in Figure 3.

Figure 3
Modularity Creates Design Options

System Before Modularization



System after Modularization



The figure shows a system that makes a transition from being one interdependent whole to being a modular system. In this transition, the system goes from having one large option (i.e., to take it or leave it) to having many smaller options (to take or leave each module). There is at least one and sometimes more than one option per module in the system. As such, a modularization multiplies the valuable options in the system. At the same time, a modularization moves decisions from a central point of control to points of control in the individual modules. That in turn allows the now-decentralized system to evolve in particular ways.

Notice however that by modularizing, one barrier to entry by competitors, the high costs associated with developing an entire system (like an automobile), are reduced to the costs of developing individual modules. Thus a modularization potentially sows the seeds of increased competition in the future.

What is Modularity?

Let us now look at modularity more carefully. To help us do so, I will represent the design of an artifact using a technique called Design Structure Matrix (DSM) Mapping. In this mapping technique, an artifact is described as a set of design parameters. The design parameters are listed on the rows and

columns of the DSM matrix. Then the matrix is filled in by checking—for each parameter—which other parameters affect it and are affected by it. For example, if Parameter A affects the choice of Parameter B, then we will put a mark “X” in the cell where the *column of A* and the *row of B* intersect. We then repeat this process until we have recorded all parameter interactions. The result is a map of the “dependencies” that affect the detailed structure of the artifact. For example, Figure 4 is a DSM map of the dependencies in the design (and design process) for a laptop computer system circa 1993.⁴

These maps can be constructed for any artifact, whether it is tangible or intangible. Thus there are DSM maps of products, like computers and automobiles, and DSM maps of both production processes and design processes. Many such maps have been constructed by Professor Steven Eppinger and his colleagues at MIT.⁵ DSM mapping techniques are also being utilized today by a number of consulting firms.

The DSM map in Figure 4 indicates that the laptop computer design has four blocks of very tightly interrelated design parameters corresponding to the (Disk) Drive System, the Main Board, the LCD Screen, and the Packaging of the machine. There is also a scattering of dependencies (“X’s”) outside the blocks. The dependencies arise both above and below the main diagonal blocks, thus the blocks are *interdependent*. In other words, the DSM map in Figure 4 represents an *interdependent system*, not a modular system.

An important feature of interdependent systems is *cycling* and *iteration* are needed to resolve interdependencies. For example, as shown in the figure, the location of the computer’s graphics controller creates dependencies between the Main Board and the LCD Screen and vice versa. Because of these dependencies, there will be consequences and ramifications of any choice made at this particular point: these are indicated by the arrows in the diagram. If two people or two teams were working on the different components, for instance, they would have to confer about the location of the graphics controller at least once in order to coordinate their design choices. But unforeseen consequences might arise later, and cause the initial choice to be revisited. There would then be further consequences: new arrows would arise, which, through the chain of dependencies, might wander all over the map. That is cycling in the context of an interdependent design or task structure.

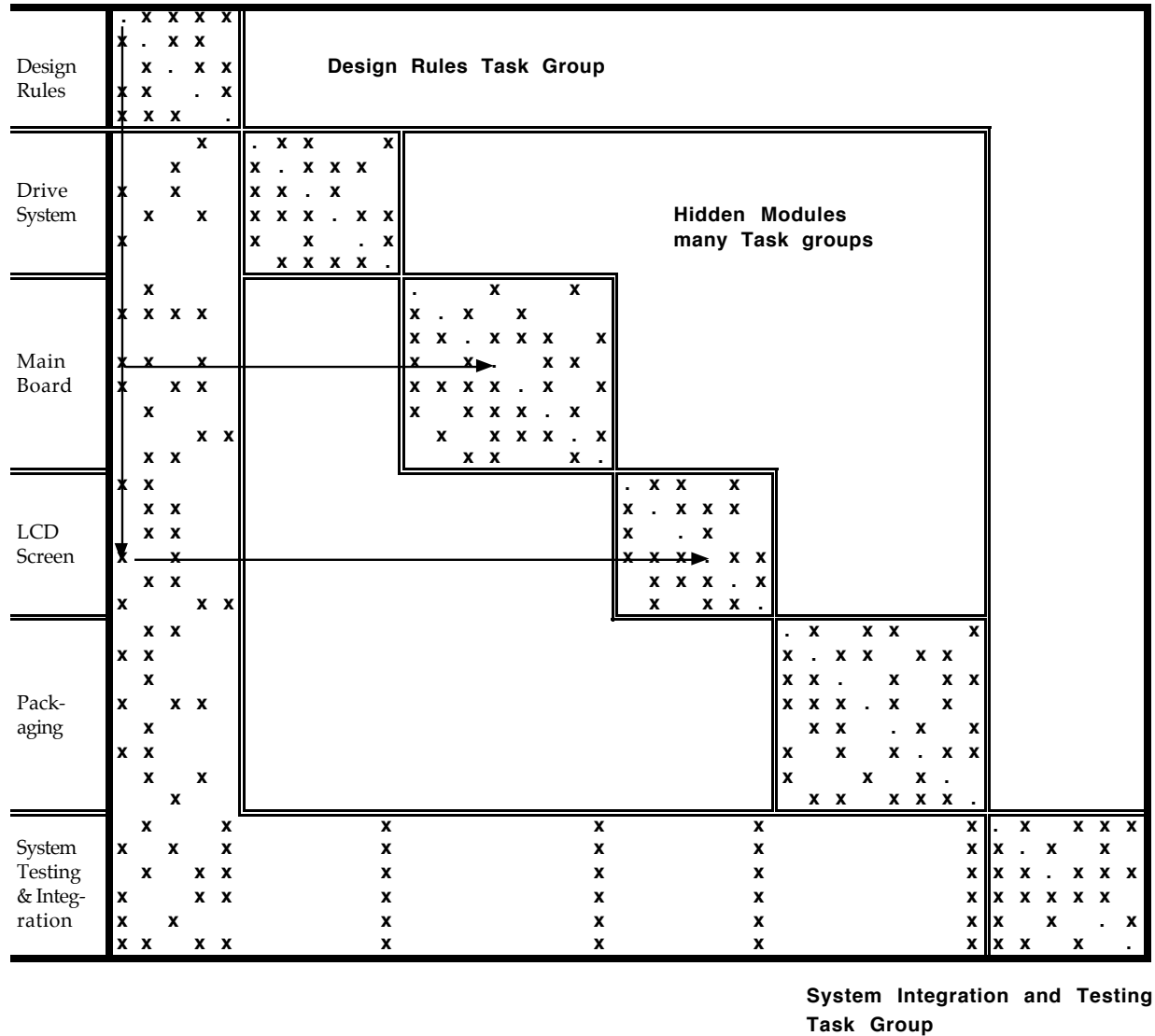
However, it is important to know that the DSM map for a product or process need not be and arguably should not be set hard and fast forever. Dependencies and interdependencies can be modified by a process of design rationalization, which works in the following way. Suppose that the designers of the laptop computer system wished to eliminate the interdependencies between the Main Board and the Screen that were due to the graphics controller location. They could do so by setting a *design rule* that located the graphics controller on or off the Board. By this action the two design teams would have restricted their alternatives, *but* they would also have eliminated a source of cycling between two of the blocks of the design. Figure 5 shows a new DSM map. Two dependencies, one above the diagonal and one below, which were present before, now do not exist: they are absent from the circled areas in the map. Instead there is a design rule that is known (visible) to both sets of designers, and which they must obey.

Figure 5
Eliminating Interdependencies by Creating a Design Rule

Design Rules	Graphics Controller - Yes/no										
Drive System	. x x x	x	x	x	x	x	x	x	x	x	x
	x . x x x	x	x	x	x	x	x	x	x	x	x
	x x . x	x	x	x	x	x	x	x	x	x	x
	x x x . x x	x	x	x	x	x	x	x	x	x	x
	x x . x	x	x	x	x	x	x	x	x	x	x
Main Board	x x x x .	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x
LCD Screen	x x x x	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x
Packaging	x x x x	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x
	x x x x	x	x	x	x	x	x	x	x	x	x

Carrying this process through to its logical conclusion results in a radically different structure: a *modular structure* as shown in Figure 6. Here we have the same highly interdependent blocks as before: the Drive System, the Main Board, the LCD Screen, and Packaging. And within those blocks essentially nothing has changed, the pattern of interdependency is the same. *But the out-of-block dependencies both above and below the main diagonal have all disappeared.*

Figure 6
Modularization of a Laptop Computer Design



How does that happen? First, in the new structure, each of the former out-of-block dependencies has been addressed by a design rule. Thus, there is now a new “Design Rules” block (not drawn to scale), whose parameters affect many of the parameters in the component blocks. Those dependencies are indicated by the “x”s in the vertical column below the Design Rules block. (In the computer industry, design rule parameters are often called “standards.”) By obeying the design rules, teams working in each of the component blocks—which are now modules—can maintain conformity with the other parts of the

system. (Note that there has been another, earlier stage in the process: a stage in which design rules are established.)

The new process, in Figure 6, delivers four separate items, which must still be integrated into a functioning whole. In addition, no set of design rules is perfect, and unforeseen compatibility problems may be revealed in the latter stages of the modular process. For these reasons, a “System Integration and Testing” block appears in the lower right corner of the modular DSM. This block is affected by the design rules, as well as by some external parameters of the so-called hidden modules. But decisions taken in this block do not affect choices in the prior blocks (if they do, the structure is no longer modular).

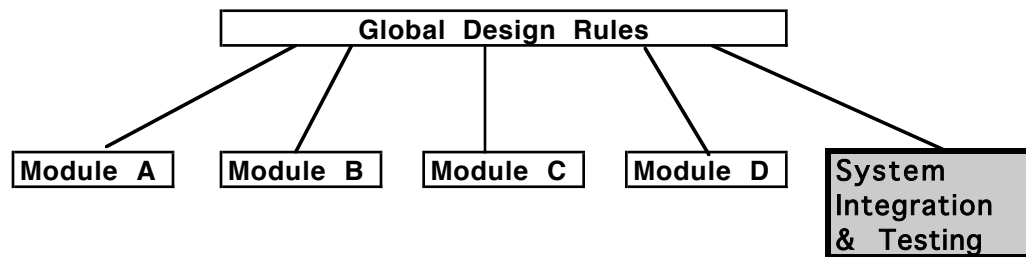
Therefore, a modular structure has three characteristic parts: design rules, which are known and obeyed by teams responsible for individual modules; so-called hidden modules that “look to” the design rules, but are *independent of one another* as work is proceeding; and a systems integration and testing module in which the hidden modules are assembled into a system, and any remaining, minor problems of incompatibility are resolved.

In summary, using DSM mapping techniques, I have shown you two generic structures for products and processes, as well as how it is possible to go from one to the other. A complex product or process may go from being *interdependent* to being *modular* in the following way. The “architects” of the system must first identify the dependencies between the components and address them via a set of design rules. Second, they must create encapsulated or “hidden” modules corresponding to the components of the system. And third, they must establish a separate system integration and testing module to assemble the components and resolve unforeseen incompatibilities.

A DSM map is one way to represent a modular system: a design hierarchy is another.⁶ A design hierarchy shows which modules are affected by which other modules. (See Figure 7.) At the very top of the design hierarchy are the system-wide design rules: these must be obeyed (hence “are visible to”) all modules in the system. Below the system-wide design rules, there may be “architectural modules,” which establish design rules for certain subsystems. There are no architectural modules in Figure 7, but in complex modular systems there may be several layers of architectural modules. For example, operating systems like Microsoft Windows and Unix are architectural modules in a computer system. Finally, at the bottom of the design hierarchy are the hidden modules of the system: these must obey the design rules, hence “look to” them. But the hidden modules’ own parameters are “encapsulated”: they do not affect,

and hence do not need to be known to, those working on other modules. Hidden modules are thus the primary source of option value in a modular system.

Figure 7
A Two-level Modular Design Hierarchy



The system comprises four "hidden" modules and a "System Integration and Testing" stage. From the standpoint of the designers of A, B, C, and D (the hidden modules), system integration and testing is simply another module: they do not need to know the details of what goes on in that phase as long as they adhere to the global design rules. Obviously, the converse is not true: the integrators and testers have to know something about the hidden modules in order to do their work. How much they need to know depends on the completeness of the design rules. Over time, as knowledge accumulates, more testing will occur within the modules, and the amount of information passed to the integrators will decrease. The special, time-dependent role of integration and testing is noted by the heavy black border around and gray shading within the "module."

Modularity in Design, Production and Use

Humans interact with artifacts in three basic ways: they design them; produce them; and use them. There are, as a result, three basic types of modularity. The laptop computer was an example of modularity in the *design* of a complex product. However, one can observe modularity in *production* processes and in the *use* of products as well.

Manufacturers have used *modularity in production* to simplify complex processes for a century or more. Indeed, process modularity is fundamental to mass production. Car makers, for example, routinely arrange to manufacture the components of an automobile at different sites and bring them together for final assembly. They can do so because they have completely and precisely specified how the parts will interact with the vehicle. The engineering specifications of a component (its dimensions, tolerances, functionality, etc.) constitutes a set of design rules for the factories that supply the parts.

Modularity in use allows consumers to mix and match elements to come up with a final product that suits their taste and needs. For example, consumers often buy bed frames, mattresses, pillows, linens,

and covers made by different manufacturers and distributed through different retailers. The parts all fit together because different manufacturers make the goods in standard sizes. These standard dimensions constitute design rules that are binding on manufacturers, wholesalers, retailers, and users. Modularity in use can spur innovation: the manufacturers can independently experiment with new products and concepts, such as futon mattresses or fabric blends, and find ready consumer acceptance as long as their products conform to the standard dimensions.

In some cases, it happens that the “natural modules” of design, production and use line up with one another. This is true of many parts of a computer system, for example, disk drives, chips, software, etc. In these instances, locating the “breakpoints” of a modular architecture is relatively easy. In other cases, however, the interdependencies that are most salient in the design process are different from those that matter for production (or use). For example, when the components of a particular subsystem of an automobile, like climate control, are spread throughout the vehicle, the “natural modules” of production will be very different from the “natural modules” of design. Coming up with a single, modular architecture for the whole system then becomes very difficult, if not impossible. Companies must then assess the costs and benefits of different types and degrees of modularity and decide which if any are worth pursuing. As a matter of corporate strategy, companies can and do favor different types of modularity.

Modular Organizations

Note that the ‘modularized’ architecture of Figures 6 and 7 leads naturally to a “modularized” organization. In a recent paper which addressed the nature of transactions we postulated that the activities represented by the x 's in Figures 5 and 6 naturally map onto organizations since each interaction captured on the DSM represents a transfer of material or information or both. In this way it is natural to look at Figure 5 and see a ‘traditional’ organization structured around specific technologies or disciplines. The on-diagonal interactions naturally represent interactions that are internal to each organizational unit while the off-diagonal interactions might be viewed as interactions that require coordinators or liaison personnel whose function is to assure that activities in each unit of the overall endeavor remain synchronized and coordinated.

In the modularized design of Figure 6 however, many of the cross-unit liaison and coordination functions have been eliminated. This is done in two ways: through design rules or standards and through encapsulation. Design rules or standards ensure that decisions that affect multiple units are fixed and communicated ahead of time, and not changed along the way. Encapsulation means that all activities associated with specific aspects of the product are conducted within one organizational unit – even if that organizational unit embodies multiple skillsets, disciplines or activities. This may lead to duplication of skillsets – for example the Main Board and Packaging groups may both need people who are skilled in understanding interconnections – but, if they are to be encapsulated, these groups must be scaled and staffed to their own needs without calling on shared resources, and without requiring extensive cross-unit coordination activities. Similarly note that the transactions have now been simplified to a simple handoff from one group to the succeeding group.

An educational analogy might be useful here. Instead of a laptop computer, Figure 5 might as easily represent the traditional departmental or discipline-based organizational structure of a university (like Harvard). A discipline-based organizational structure is well-suited to teaching courses, but it is ill-suited to carrying out broadly-based research initiatives that cut across many disciplines. Indeed, in order to conduct interdisciplinary investigations, the traditional departmental structure requires cross-unit interactions as shown in Figure 5. These wide-ranging interactions are generally time-consuming and prone to cycling. In addition, many of the most task-relevant interactions may get lost in the shuffle and not take place at all.

By contrast, a center-based or project-based collaborative structure gathers participants from multiple disciplines and organizes them into self-contained teams as in Figure 6. Even though some interactions are lost, and there may be duplication of resources across centers, the new organizational structure imposes a much smaller coordination burden on the overall research endeavor. The reduced coordination costs in turn can offset the opportunity losses and costs of implementing the more modular, team-based structure. If the coordination cost savings are large, then a “more modular” organization is a good idea. But there is always a tradeoff—some things are lost while other things are gained, and there is a cost of making the transition.

Modular Operators

Arguably the key benefit of modular systems is that, especially at the lower, hidden-module levels of the design hierarchy, they can evolve. The information that is encapsulated in the hidden modules can change, as long as the design rules are obeyed. Therefore, modular systems, as I have said, are “tolerant of uncertainty” and “welcome experimentation” *in their subsystems*. (In contrast, design rules, once established, tend to be both rigid and long lasting.)

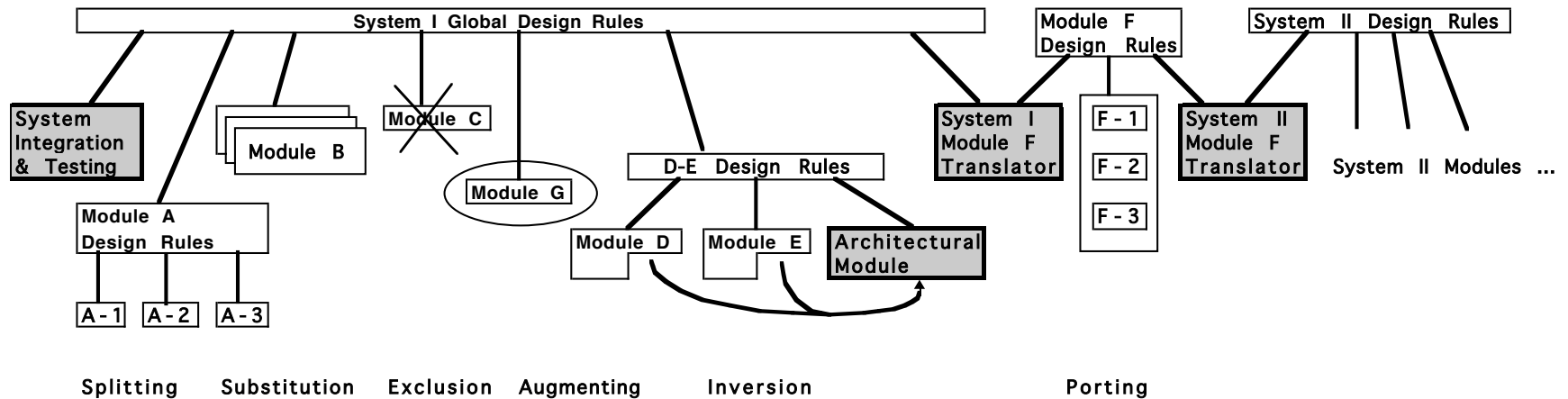
There are certain generic actions one can apply to a modular system. Following the lead of John Holland of the University of Michigan,⁷ Kim Clark and I have labeled such actions “operators.” In our prior work, we identified six operators, and analyzed the sources of their economic value. Our list of six is not exhaustive, but it is the beginning of a useful taxonomy. In particular, given a modular structure, one can:

- *split* any module;
- *substitute* a newer module design for an older one;
- *exclude* a module;
- *augment* the system by adding a module that was not there before;
- collect common elements across several modules and organize them as a new level in the hierarchy (modular *inversion*); and
- create a “shell” around a module so that it works in systems other than one for which it was initially designed (modular *porting*).

Figure 8 shows how each operator affects the structure of a modular system. The modular operators come in pairs. Splitting and substituting go together, augmenting and excluding go together, and what we call inversion and porting are also related to one another. Let me reiterate that this is by no means an exhaustive list of operators. I know of three additional operators that I would offer as candidates for the list and other suggestions are welcome too.

The important thing to understand is that *operators correspond to options* in the modular system. As options, they can be valued using standard analytic techniques from finance. They can be valued *generically* in terms of mathematical expressions of value and costs. They can also be valued *specifically* for individual designs if one is willing to gather the requisite data.

Figure 8
The Effect of the Six Operators on a Modular System



We started with a generic two-level modular design structure, as shown in Figure 5-3, but with six modules (A, B, C, D, E, F) instead of four. (To display the porting operator, we moved the "System Integration & Testing Module" to the left-hand side of the figure.) We then applied each operator to a different set of modules.

Module A was **Split** into three sub-modules.

Three different **Substitutes** were developed for Module B.

Module C was **Excluded**.

A new Module G was created to **Augment** the system.

Common elements of Modules D and E were **Inverted**. Subsystem design rules and an architectural module were developed to allow the inversion.

Module F was **Ported**. First it was split; then its "interior" modules were grouped within a shell; then translator modules were developed.

The ending system is a three-level system, with two **modular subsystems** performing the functions of Modules A, D and E in the old system. In addition to the standard hidden modules, there are three kinds of special modules, which are indicated by heavy black borders and shaded interiors:

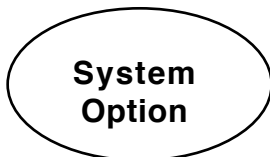
- System Integration & Testing Module
- Architectural Module
- Translator Module(s)

An Illustration of Option Value: The Value of Splitting and Substitution

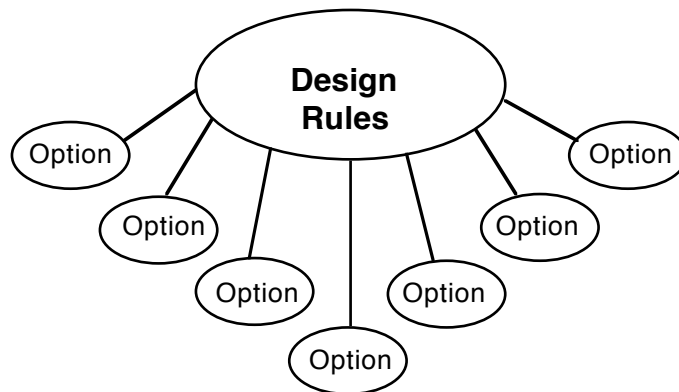
Please recall that the essential fact about modularity is that tasks within different modules can proceed independently and the results can be combined in a variety of ways. Thus in a one-module design process, only two outcomes are possible: either the new design will be superior to the old and will replace it, or the new design will fall short, and the old design will continue to be used. However, if the design has been split into two modules, then, when the new module designs are finished, the designers will have four choices. They can introduce an entirely new system, keep the old system, or combine one old module with one new one. Similarly, a three-module system generates eight alternatives, a four-module structure generates sixteen, and so on. As Figure 3 (reproduced below) displayed, this expansion in the number of options is the direct result of changing the *point of selection* from the system level to the module level.

Figure 3 (reproduced)
Modularity Creates Design Options

System Before Modularization



System after Modularization

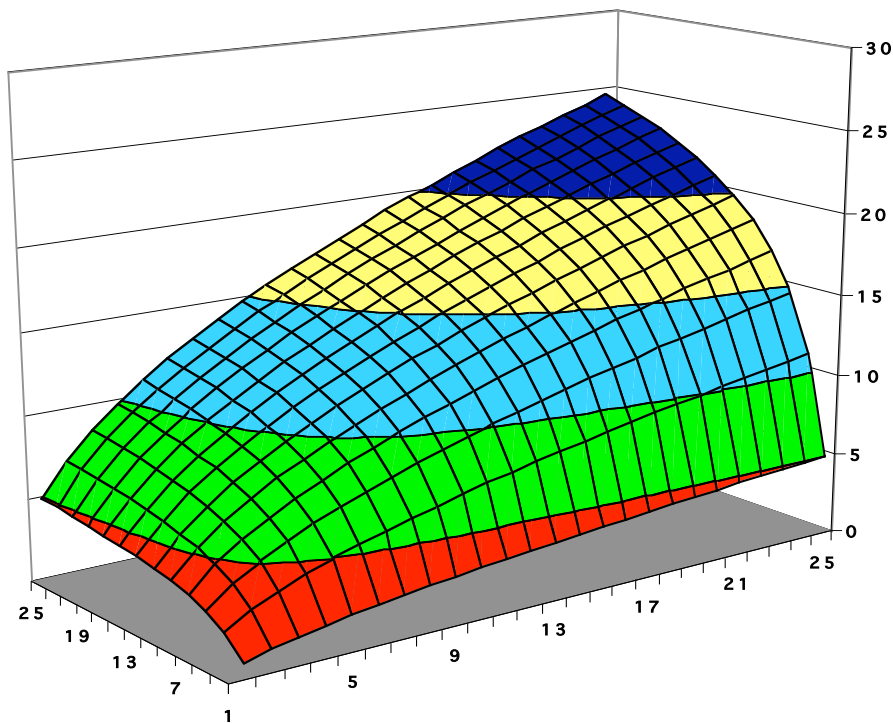


Breaking up a large system into some number, j , modules gives us an old version and a new version of each module, and 2^j alternative configurations. But we don't need to stop with one new design for each module! In fact, for each module we could think of embarking on k parallel design experiments.

We would then wait until the results were in, and *pick the experimental design that delivered us the highest value after the fact.* (That is known as the “best of breed” approach to a modular design.)

In fact, the option to select the maximum of k risky outcomes is a valuation problem in finance that was solved by Rene Stulz in 1982.⁸ It is also amenable to the technique of “order statistics,” which is commonly used in statistics.⁹ After going through the mathematics, which I intend to spare you, we arrive at the results shown in Figure 9. The way to read this chart is as follows. The vertical axis shows “value” as a function of two variables. The first variable, shown on the right-hand dimension, is the number of modules in a system. In the figure, this variable ranges from 1 to 25. The second variable, on the left, is the number of design experiments, i.e. R&D projects, per module. This variable also ranges from 1 to 25.

Figure 9
The Value of Splitting and Substitution



The surface in the figure indicates the value of different combinations of modules and experiments. As we go out along the middle of this surface, we see the value of running one experiment on one module, two experiments on each of two modules, and so on, until at the far corner, we have 25

experiments on each of 25 individual modules in the system. The figure shows is that there is strong complementarity between modules and experiments. More modules make more experiments more valuable, and vice versa. The two things go together.

This result is compelling, because the amount of economic value being created here is really quite large. The values are calculated relative to the value of a single experiment in an un-modular system; we see, in effect, that a complex project organized as 25 modules with 25 experiments per module can obtain approximately 25 times the value of the same project organized as a single, interdependent whole. Now, my values are hypothetical, and I have not yet counted any costs. But a value multiple of 25 will pay for a lot of architecting and reorganization! In other words, the incentives afforded by the combination of modularity and experimentation are so great that in a free-market economy, if it is possible to modularize, someone will do so in order to capture this value.

The result is also surprising! Modules and experiments work together, and therefore, as you increase modularity, and you must increase experimentation, that is, R&D, in order to get the best outcome. This is costly, as we shall see. But there is another kind of "cost" involved: the cost of potential innovation ignored. Very frequently engineers with the encouragement of managers will modularize a system in order to manage its complexity. Yet it is possible that neither group will understand what the modularization implies for the value of innovation and experimentation within the system. And if the top managers in the modularizing firm ignore that innovative value, the result will be strong incentives for other firms to enter the marketplace, offering their own innovative modules.

Something like that happened at IBM after the company introduced the first modular computer, System/360, in the mid-1960s. System/360 was a powerful and popular modular system, a *tour de force* in terms of product design, marketing, and manufacturing. However, IBM did not understand the value of the options created by its own modular design, hence it did not increase its inhouse design efforts to reflect the modularity of its own system. As a result, it left profitable module design opportunities "on the table," and as economics predicts, other firms moved in and seized these opportunities.

Many of the new firms that entered the computer industry during the 1970s in the wake of System/360 were founded by former IBM engineers. The engineers who had worked on System/360 and its successors could see the module options very well, and knew the design rules of the system. Thus when IBM's top managers did not fund their projects, they took those projects elsewhere. Beginning in

the early 1970s, scores and then hundreds of engineers left IBM and joined others in founding companies that supplied “plug-compatible” modules for IBM’s System/360 and 370. As it happens, one of IBM’s main R&D labs was located in San Jose, California, and the exodus of engineers from the San Jose labs was one of the key factors that contributed to the emergence of what we now call Silicon Valley.

As I said at the beginning of this talk, modular options are inherently decentralized. The promulgators of design rules—the architects of a modular system—do not have to give permission for the module options to be exercised. That is very efficient, but it is also quite dangerous for firms that may want create modular systems in order to own, control, and profit from them. Modularity creates value, but that value can be very difficult to capture. So, modularity in its interaction with experimentation is compelling, surprising, and dangerous.

The Costs of Modularity

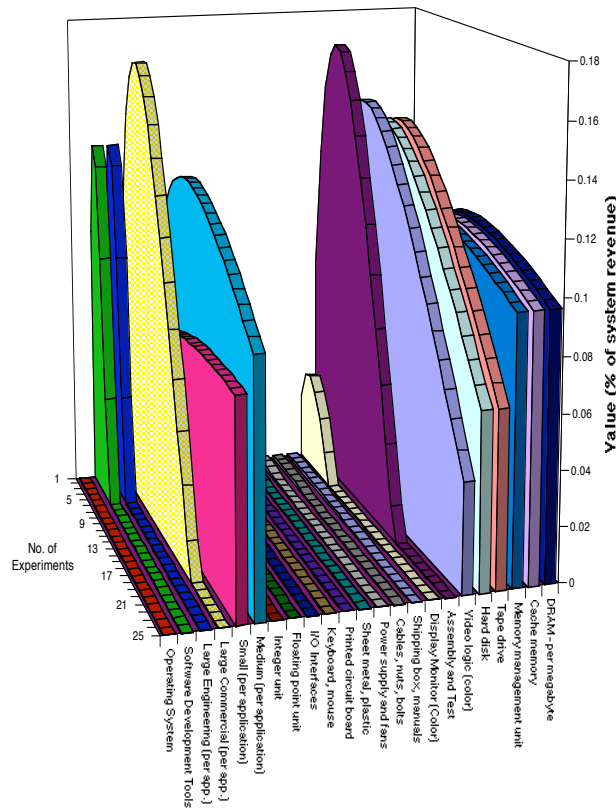
I do not want to close without addressing the costs of modularity. I might then leave you with the impression that modularity is free. It is not. In fact, the costs of creating and exploiting a modular system are as numerous as the value is large.

There are, first of all, the costs of making an interdependent system modular: the cost of creating and disseminating design rules. In the early part of this talk, you could see how painstaking the process of modularization must be, if it is to succeed in creating truly independent modules. *Every important cross-module dependency must be understood and addressed via a design rule.*

Obviously the density of the dependencies matters here: some systems are naturally more “loosely-coupled” than others. Circuits, the physical system on which computers are based, are one-dimensional; whereas mechanical solids are three-dimensional. Clearly it is harder to split up complex, curved, 3-dimensional designs, and to create flexible interfaces for them: there are more dependencies to manage, and the tolerances are much tighter. Thus modularizing an automobile’s design is a tougher engineering problem than modularizing a circuit design: the cost of creating a modular architecture and related interfaces will be higher. This has led some scholars, like Daniel Whitney at MIT, to predict that autos and airplanes will achieve only limited modularity in practice.¹⁰ If that is so, the option values of such systems will be limited in relation to systems that can more easily be modularized.

It is also costly to run the experiments needed to realize the potential value of a modular system. Finally, it is costly to design the tests needed to determine if particular modules are compatible with the system, and which one performs best. The costs of architecture, experiments, and tests are all inherent in the modular design process itself. The interaction of option value and these costs causes each module in a large system to have a unique value profile. For example, Figure 10 shows the value profiles (relative to the number of experiments) for the modules of a computer workstation circa 1990.

Figure 10
Value Profiles for the Modules of a Computer Workstation



Conclusion

Let me conclude by commenting on the perils of modularity. Modularity requires that you operate all aspects of your business more efficiently than your competitors. Returning to the computer

industry, let us now focus on the introduction of the IBM PC in the 1980s.¹¹ By this time, IBM had learned the basic lessons of modularity. In particular, IBM's managers understood how modularity encourages both innovation and entry on modules. In fact, the PC was designed with highly modular architecture, and IBM leveraged this design by outsourcing most hardware and software components. IBM's managers also understood that they needed to protect the company's privileged position within the architecture. Thus IBM retained control of what were thought to be the key design and process modules—a chip called the BIOS (Basic Input Output System) and the manufacturing process of the PC itself. By exercising control of these critical and essential "architectural modules," IBM's managers believed that they could manage the rate of innovation in PC's. The idea was to obtain maximum return from each generation of PC, before going on to the next.

However, the founders of Compaq had a different idea. First, they independently and legally replicated IBM's technical control element, the BIOS. Then they designed a machine that was fully compatible with IBM's published and non-proprietary specifications. They bought the key modules of the PC — the chip and the operating system — and all the other parts they needed from IBM's own suppliers. And they went to market with an IBM-compatible PC built around the newer, faster 386 chip while IBM was still marketing 286 machines. Within a year Compaq had sales of \$100 million; by 1990, its revenues were \$3 billion and climbing, and IBM was looking to exit from its unprofitable PC business.

However, as everyone knows, the leading player in the PC market today is not Compaq but Dell.¹² In a nutshell, Dell did to Compaq what Compaq did to IBM: it took advantage of the benefits of modularity and designed a technologically competitive, Wintel-compatible PC. But Dell also used the tools of *process modularity* more effectively than Compaq in order to arrive at a more efficient, less-asset-intensive business model. Thus Dell outsourced even more of its manufacturing activities than Compaq. Dell builds machines to order and sells directly to consumers, thereby cutting out both dealers' margins and inventory. Compaq simply could not compete. As it saw the handwriting on the wall in PCs in early 1998, it tried to move into higher-margin "enterprise computing" through its acquisition of Digital Equipment. But it did not succeed in making this transition. Today, Compaq no longer exists as a separate company.

This story makes a fitting end to the formal part of this presentation. To summarize, the widespread adoption of modularity across an industry can set in motion an uncontrollable process of

industry evolution. It opens the door to a very Darwinian world in which neither the developer of a modular architecture nor the leading practitioner and exponent of that modular architecture may ultimately prevail. It is a world of change and opportunity, but, as the recent stock market bubble and the related telecomm failures have taught us, it can also be a world of chaos and inefficiency. In short I hope I have convinced you that modularity is not good or bad. Rather, it is important, and it is costly. And it is dangerous to ignore.

¹ The argument and all figures in this paper are taken from C.Y. Baldwin and K.B. Clark, *Design Rules, Volume 1: The Power of Modularity*, © MIT Press, 2000, reprinted by permission.

² The original theory of pre-emptive investment leading to industry concentration, with supporting historical evidence, was put forward by Alfred Chandler (1962, 1977). A complementary theory of concentration following the emergence of a “dominant design” was put forward by William Abernathy and James Utterback (1978). Modern formulations of these theories and some large-scale empirical tests have been developed by John Sutton (1992) and Steven Klepper (1996). Oliver Williamson (1985, Ch. 11) has interpreted the structures of modern corporations (unified and multi-divisional) as responses to potential opportunism (the hazards of market contracting). It is our position that the basic “task structures” and the economic incentives of modular design (and production) systems are different from the task structures and incentives of classic large-volume, high-flow-through production and distribution systems. Therefore the organizational forms that arise to coordinate modular design (and production) may not resemble the classic structures of the modern corporation.

³ O.E. Williamson, 1999, “Human Action and Economic Organization,” mimeo, University of California, Berkeley; quoted in M. Aoki, *Towards a Comparative Institutional Analysis*, 2001, MIT Press, Chapter 4.

⁴ The DSM methodology was invented by Donald Steward. The DSM map shown in Figure 4 was prepared by Kent McCord and Steven Eppinger, and appeared in K.R. McCord and S.D. Eppinger, 1993, “Managing the Integration Problem in Concurrent Engineering,” MIT Sloan School of Management Working Paper, no. 3594, August.

⁵ See, for example, S.D. Eppinger, D.E. Whitney, R.P. Smith, and D.A. Gebala, 1994, “A Model-Based Method for Organizing Tasks in Product Development,” *Research in Engineering Design* 6(1):1-13; S.D. Eppinger, 1991, “Model-Based Approaches to Managing Concurrent Engineering,” *Journal of Engineering Design* 2(4): 283-290. Also, the publications listed at <http://web.mit.edu/dsm/>.

⁶ D.L. Marples, 1961, “The Decisions of Engineering Design,” *IEEE Transactions in Engineering Management*, 2: 55-81. See also, K.B. Clark, 1985, “The Interaction of Design Hierarchies and Market Concepts in Technological Evolution,” *Research Policy* 14(5): 235-251.

⁷ J.H. Holland, 1992, *Adaptation in Natural and Artificial Systems 2nd Edition*, University of Michigan Press, Ann Arbor, MI.

⁸ R.M. Stulz, 1982, “Options on the Minimum or Maximum of Two Risky Assets,” *Journal of Financial Economics*, 10: 161-185.

⁹ See, for example, B.W. Lindgren, 1968, *Statistical Theory*, MacMillan, New York, Chapter 8.

¹⁰ D.E. Whitney, 1996, “Why Mechanical Design Cannot Be Like VLSI Design,” <http://web.mit.edu/ctpid/www/Whitney/morepapers/design.pdf>, viewed April 9, 2001.

¹¹ This history is recounted in Charles H. Ferguson and Charles R. Morris, *Computer Wars: The Fall of IBM and the Future of Global Technology*, Times Books, NY, 1994.

¹² On Compaq vs. Dell, see Steven C. Wheelwright and Matt Verlinden, 1998 "Compaq Computer Corporation," 9-698-094, Harvard Business School Publishing Company, Boston, MA; and Carliss Y. Baldwin and Barbara Feinberg, 1980, "Compaq: The DEC Acquisition," 9-800-199, Harvard Business School Publishing Company, Boston, MA.