The Structure and Value of Modularity in Software Design

Kevin Sullivan, Yuanfang Cai, Ben Hallen
Dept. of Computer Science
University of Virginia P.O.Box 400740
Charlottesville, VA 22904-4740
Tel: +1 (804) 982-2206
{sullivan,yc7a,bh5z}@cs.virginia.edu

William G. Griswold
Dept. of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093-0114
Tel: +1 (858) 534-6898
wgg@cs.ucsd.edu

ABSTRACT
The concept of information hiding modularity is a cornerstone of modern software design thought, but its formulation remains casual and its emphasis on changeability is imperfectly related to the goal of creating value in a given context. We need better models of the structure and value of information hiding, for both their explanatory power and prescriptive utility. We evaluate the potential of a new theory—developed to account for the influence of modularity on the evolution of the computer industry—to inform software design. The theory uses *design structure matrices* to model designs and *real options* techniques to value them. To test the potential utility of the theory for software we represent a model software system in its terms—Parnas's KWIC—and evaluate the results. We contribute an extension to *design structure matrices* and show that the options results are consistent with Parnas's conclusions. Our results suggest that such a theory does have potential to help inform software design.

Categories and Subject Descriptors
D.2.10 [**Software Engineering**]: Design – *methodologies, representation.*

General Terms
Software Engineering, Economics

Keywords
Strategic software design, design structure matrix, design rules.

1. INTRODUCTION
In 1972 Parnas introduced *information hiding* as an approach to devising modular structures for software designs [8]. The approach promised to dramatically improve the adaptability of software. The idea is to decouple design decisions that are likely to change so that they can be changed independently. Parnas supported his claim that information hiding produces better designs with a case study comparing the changeability of two versions of KWIC (*Key Words in Context*), one modularized around processing steps, the other using information hiding.

This idea has been broadly influential, catalyzing the development of abstract data type programming languages, object-oriented design and programming, and the discipline of software architecture. Through all these developments, Parnas's formulation has been adopted largely without change.

We identify an issue for software designers that neither Parnas's formulation nor subsequent developments based on it adequately address. A designer is responsible for producing the greatest benefit for any given investment of time, talent, money, and other resources. Modularization decisions have a tremendous impact in this dimension. However, the information-hiding decision process does not account explicitly for the costs and benefits of modularization, so it does not help designers to design most effectively for *value added.*

In particular, the current formulation does not address two key questions from a value-maximization perspective: (1) which of the available modularizations is best; (2) at what point should one be selected in the face of uncertainty about the first question? Today, designers are encouraged to make modularization decisions at the earliest stages of design. Yet, at these stages they are often unsure of which decisions are likely to change and of the dependencies among them. Moreover, the information-hiding criterion is a qualitative process of modularization against a set of design decisions. The principle way to evaluate such a modularization is to test whether it hides the information it was meant to hide (which is self-referential). We lack precise, quantitative models of the value of proposed modularizations. In particular, we lack models for assessing the relative value of proposed modularizations for the same system.

In this paper we introduce an approach to modeling software designs that addresses these problems, with an emphasis on the question of relative valuation. In particular, the approach allows us to value modular structures relative to inferred non-modular designs, and thus against each other. The approach follows the theory that Baldwin and Clark developed to account for the influence of modularity on the evolution of computer system designs and the structure of the industry that creates them [1][2].

A key idea in this work is that modularity in design creates value in the form of *options* to improve a system by experimenting with new implementations and substituting superior ones. The value of such options is modeled quantitatively. This idea, which is consistent with our work on the options value of flexibility in software [4][11][12], promises to help link structural aspects of design to value as an objective. Parnas's ideas influenced Baldwin and Clark's theory, but our adaptation of the theory to software design appears to be novel.

To test of the potential of the theory to value modularization decisions in software we reformulate Parnas's KWIC analysis in terms of the theory. Two hypotheses underlie this approach.

First, Parnas's example can be mapped in terms of theory. Second, the resulting mapping will produce valuation results consistent with Parnas's widely accepted conclusions.

Mapping the example did reveal one gap in the theory. It lacked the means to represent how the environment in which software is deployed affects the value of its design. We extended the model to represent the *environment.* The extended theory helped in turn to clarify the information hiding idea, not in terms of decisions that are likely to change, but rather in terms of the invariance of visible decisions under changes in the environment.

With the extended theory we were able to value each of the two modularizations against a common ancestral *pre-modular* design of KWIC. Both modularizations are derived from that design by application of the *splitting operator* of the theory's design evolution framework. We found the extended theory to predict that the information hiding KWIC modularization is significantly better than the strawman design. Our results provide an early data point suggesting that the extended theory has the potential to advance the development of precise models that have descriptive and prescriptive power for value-maximizing software design.

The next two sections provide the background material for our analysis and discussion of KWIC, which follows in the three subsequent sections.

2. BACKGROUND

Baldwin and Clark's theory is based on the idea that modularity adds value in the form of *real options* (*options*). An option provides the *right* to make an investment in the future, without a symmetric *obligation* to make that investment. Because an option can have a positive payoff but need never have a negative one, an option has a present value. It is like a lottery ticket.

The important idea in this paper is that a module creates an opportunity to invest in a search for a superior replacement. Such an investment buys the designer an option to replace the current module with the best alternative, or to keep the current one, if it is still the best. Intuitively, the value of such an option is the value added by the optimal experiment-and-replace policy. Knowing this value can help a designer to reason about both the initial investment in modularity, and how much effort to expend searching for alternatives.

The idea that real options ideas can help us to analyze core issues in software design and engineering is not new. Sullivan [12] suggested that such an analysis can provide insights concerning modularity, phased project structures, delaying of decisions and other dynamic design strategies. Sullivan, Chalasani, Jha and Sazawal [11] formalized that options-based analysis, focusing in particular on the value of the flexibility to delay decision making.

Withey [14] applied a related analysis to reasoning about the flexibility value of software product line architectures. He did not appeal to real options concepts explicitly in his analysis.

Favaro [6] developed an options-based approach to investment analysis for software reuse infrastructures. The options approach was used to value the flexibility provided by reuse infrastructures to adapt in the face of uncertain conditions. The approach uses a Black/Scholes (arbitrage-based) valuation model. Using such a model is technically valid only if market-traded assets are identified that track the risks in the option being priced. Favaro assumed this condition to hold, but there are many cases in design where there will be no such tracking assets. Beck [3] is popularizing intuitions emerging from options-based analyses.

Baldwin and Clark appear to have been the first to observe that the value of *modularity* in design (of computer systems) can be modeled as options. They do not to assume that tracking assets exist. Rather, they make a statistical assumption: that the values of independently developed alternatives to an existing module are normally distributed about its value. We discuss these ideas in more detail in the following section.

3. BALDWIN AND CLARK'S MODEL

The parts Baldwin and Clark's theory that are most relevant to this paper are the representation of designs by *design structure matrices* (DSM) [10] [5]; the characteristics of *modular* designs; design evolution under *modular operators*; and the *net options value* (NOV) of modularity.

3.1 Design Structure Matrix

A DSM represents dependencies amongst the *design parameters* of a design. A design parameter represents the range of choices that can be made about an aspect of a design. Typical software design parameters include data structures, algorithms, procedure and type signatures. Others might include graphical interface look-and-feel, interoperability, and performance characteristics.

The rows and columns of a DSM are labeled by the design parameters. A dependence between two parameters is represented by a *mark* (X). A mark in row B, column A means that an efficacious choice for B depends on the choice for A. Parameters that require mutual consistency—algorithm and data structures go hand-in-hand, for example—are *interdependent*, resulting in symmetric (A,B) and (B,A) marks. When one parameter naturally precedes the other—there is no GUI look-and-feel unless there is a GUI—the parameters are called *hierarchically dependent*. Finally, *independent* design parameters may be determined and changed individually.

Choosing the design parameters to model and the values they finally take on is the task of the designer. In exploring the value of a design, the marks in a DSM represent the believed likelihood of deriving a benefit from recognizing and allowing parameter dependencies. DSM's are typically derived from experience with prior versions of the product or similar products. In early versions of a product, there may be relatively few marks in the matrix, capturing little more than the dependencies necessary to deploy a correctly functioning product. A DSM for a subsequent version might include marks for subtle but powerful dependencies that were discovered through use of the product, additions to knowledge, and innovation. Likewise, new design parameters will be recognized and added to the DSM over time. In these respects a DSM represents not just abstract design dependencies, but concrete requirements for communication amongst designers.

3.2 Design Rules and the Evolution of Designs

The dependence and interdependence of design parameters present challenges in design, because the designers managing the parameters have to communicate to achieve an optimal design. If

a parameter such as A in Figure 1 is changed, then the decisions for parameter B may need to be changed, hence propagating to C, and perhaps back to B again. Such dependencies require communication and constrain the ability of designers to work independently. These problems can be addressed by first clustering design parameters into (typically interdependent) *proto-modules* and then applying an operator called *splitting*.

	A	B	C
A	.		
B	X	.	X
C		X	.

Figure 1: An elementary DSM with three design parameters.

Groups of interdependent design parameters are clustered into a *proto-module* to show that the decisions are managed collectively as a single design task (See Figure 2; the dark lines denote the desired proto-module clusters). In essence, such a proto-module is a composite design parameter. To be a true module, in the lexicon of Baldwin and Clark, there can be no marks in the rows or columns outside the bounding box of its cluster connecting it to other modules or proto-modules in the system.

	A-I	B-C D-I	A	B-C D	B	C
A interface	.					
B-C D interface		.				
A	X		.			
B-C data struct.		X		.		
B	X	X			.	
C		X				.

Figure 2: A modular DSM resulting from splitting, adding design rules, and clustering.

Merely clustering cannot convert a monolithic design comprising one large, tightly coupled proto-module into a modular design. Interdependent parameter cycles must be broken to define modules of reasonable size and complexity. Breaking a cycle between two interdependent parameters like B and C requires an additional step called splitting.

The first step in splitting identifies the cause of the cycle—say, a shared data structure definition—and splits it out as its own design parameter (D). B and C no longer cyclically depend on each other, instead taking on a simple hierarchical dependence on D. However, B and C must still wait for the completion of D's design process in order to undertake their own.

To counter this, a design as represented by a DSM can be further modularized during the process of splitting by the introduction of design rules. Design rules are additional design parameters that decouple otherwise linked parameters by asserting "global" rules that the rest of the design parameters must follow. Thus, design rules are de facto hierarchical parameters with respect to the other parameters in a DSM. The most prevalent kind of design rule in software is a module interface. For example, for the DSM in Figure 2, an "A interface" rule could be added that asserts that the implementation of B can only access the implementation of A through an interface defined for A. Thus, A could change details of its implementation freely without affecting B, as long as A's interface did not have to be changed as well. The effects of splitting B and C and adding design rules to break the non-modular dependences is shown in Figure 3.

	A	B	C
A	.		
B	X	.	X
C		X	.

Figure 3: A modular DSM resulting from splitting, adding design rules, and clustering.

In Baldwin and Clark's terminology, a design rule is a (or part of a) *visible module*, and any module that depends only on design rules is a *hidden module*. A hidden module can be adapted or improved without affecting other modules by the application of a second operator called *substitution*.

The splitting and substitution operations are examples of six atomic *modular operators* that Baldwin and Clark introduced to parsimoniously and intuitively describe the operations by which modular designs evolve. The others are *augmentation*, which adds a module to a system, *exclusion*, which removes a module, *inversion*, which standardizes a common design element, and *porting*, which transports a module for use in another system. We do not address these other operators any further in this paper.

3.3 Net Options Value of a Modular Design

Not all modularizations are equally good. Thus, in evolving a design, it is useful to be able to evaluate alternative paths based on quantitative models of value. Such models need not be perfect. What is essential is that they capture the most important terms and that their assumptions and operation be known and understood so that analysts can evaluate their predictions.

3.3.1 Introduction to Real Options Concepts

Baldwin and Clark's theory is based on the idea that modularity in design multiplies and decentralizes *real options* that increase the value of a design. A monolithic system can be replaced only as a whole. There is only one option to replace, and exercising it requires that both the good and the bad parts of the new system be accepted. In a sense, the designer has one option on a portfolio of assets. A system that has two modules, by contrast, can be kept as is, or either or both of the new modules can be accepted, for a total of four options. The designer can accept only the good new modules. By contrast, this designer has a portfolio of options on the modules of the system. A key result in modern finance shows that all else remaining equal, a portfolio of options is worth more than an option on a portfolio.

Baldwin and Clark's theory defines a model for reasoning about the value added to a base system by modularity. They formalize the options value of each modular operator: How much is it worth to be able to substitute modules, augment, etc.

3.3.2 The Net Options Value of a Modular Design

In this paper, we address only substitution options. Splitting a design into n modules increases its base value S_0 by a fraction that is obtained by summing the net option values (NOV_i) of the resulting options:

$$V = S_0 + NOV_1 + NOV_2 + \ldots + NOV_n \qquad (1)$$

NOV is the benefit gained by exercising an option optimally accounting for both payoffs and exercise costs.

Baldwin and Clark present a model for calculating *NOV*. A module creates an opportunity to invest in *k* experiments to (a) create candidate replacements, (b) each at a cost related to the complexity of the module, and, (c) if any of the results are better than the existing choice, to substitute in the best of them, (d) at a cost that related to the visibility of the module to other modules in the system:

$$NOV_i = max_{ki} \{ s_i n_i^{1/2} Q(k_i) - C_i(n_i)k_i - Z_i \} \qquad (2)$$

First, for module *i*, $\sigma_i n_i^{1/2} Q(k_i)$ is the expected *benefit* by the best of k_i independently developed candidate replacements under certain assumptions about the distribution of such values. $C_i(n_i)k_i$ is the cost to run k_i experiments as a function C_i of the module complexity n_i. $Z_i = S_{j\text{-}sees\text{-}i} c_i n_j$ is the cost to replace the module given the number of other modules in the system that directly depend on it, the complexity n_i of each, and the cost to redesign each of its parameters. The *max* picks the number of experiments k_i that maximizes the gain from module *i*.

Figure 4 presents a typical scenario: module value-added increass in the number of experiments (better candidates found) until experiment costs meet diminishing returns. The *max* is the peak. In this case, six experiments maximizes the net gain and is expected to add about 41% value over the existing module.

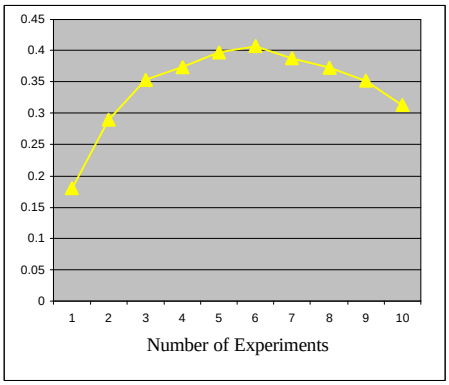

Figure 4. The value added by *k* experiments.

The NOV_i formula assumes that the value added by a candidate replacement is a random variable normally distributed about the value of the existing module choice (normalized to zero), with a variance $\sigma_i^2 n_i$ that reflects the *technical potential* σ_i of the module (the standard deviation on its returns) and the complexity n_i of the module. The assumption of a normal distribution is consistent with the empirical observation that high and low outcomes are rare, with middle outcomes more common. The *Q(k)* represents the expected value of the best of *k* independent draws from a standard normal distribution, assuming they are positive, and is the maximum order statistic of a sample of size *k*.

4. OVERVIEW OF ANALYSIS APPROACH

As an initial test of the potential for DSM's and NOV to improve software design, we apply the ideas to a reformulation of Parnas's comparative analysis of modularizations of *KWIC* (a program to compute permuted indices) [1]. The use of *KWIC* as a benchmark for assessing concepts in software design is well established [7][9].

Parnas presents two modularizations: a traditional "strawman" based on the sequence of abstract steps in converting the input to the output, and a new one based on information hiding. The new design used abstract data type interfaces to decouple key design decisions involving data structure and algorithm choices so that they could be changed without unduly expensive ripple effects.

Parnas then presents a comparative analysis of the *changeability* of the two designs. He postulates changes and assesses how well each modularization can accommodate them, measured by the number of modules that would have to be redesigned for each change. He finds that the information-hiding modularization is better. He concludes that designers should prefer to use an information hiding design process: begin the design by identifying decisions that are likely to change; then define a module to hide each such decision.

Our reformulation of Parnas's example is given in two basic steps. First, we develop DSM's for his two modularizations in order to answer several questions. Do DSM's, as presented by Baldwin and Clark (as well as in the works of Eppinger and Steward [5][10], who invented DSMs), have the expressive capacity to capture the relevant information in the Parnas examples? Do the DSM's reveal key aspects of the designs? Do we learn anything about how to use DSM's to model software?

Second, we apply Baldwin and Clark's substitution NOV model to compute quantitative values of the two modularizations, using parameter values derived from information in the DSM's combined with the judgments of a designer. The results are *back-of-the-envelope* predictions, not precise market valuations; yet they are useful and revealing. We answer two questions. Do the DSM's contain all of the information that we need to justify estimates of values of the NOV parameters? Do the results comport with the accepted conclusions of Parnas?

Our evaluation revealed one shortcoming in the DSM framework relative to our needs. DSM's as used by Baldwin and Clark and in earlier work do not appear to model the environment in which a design is embedded. Consequently, we were unable to model the forces that drove the design changes that Parnas hypothesized for KWIC. Thus, DSM's, as defined, did not permit suffiently rich reasoning about change and did not provide enough information to justify estimates of the environment-dependent *technical potential* parameters of the NOV model.

We thus extended the DSM modeling framework to model what we call *environment parameters* (EP). We call such models *environment and design structure matrices* (EDSM). DP's are under the control of the designer. Even design rules can be changed, albeit possibly at great cost. However, the designer does not control EP's. Our extension to the EDSM framework appears to be both novel and useful. In particular, it captures a number of important issues in software design and, at least in the case of the Parnas modularization, it allows us to infer some of Parnas's tacit assumptions about change drivers.

The next section presents our DSM's for Parnas's KWIC. Next we present our NOV results. Finally, we close with a discussion.

5. DSM-BASED ANALYSIS OF KWIC

For the first modularization, Parnas describes five modules: Input, Circular Shift, Alphabetizing, Output, and Master Control. He concludes, "The defining documents would include a number of pictures showing core format, pointer conventions, calling conventions, etc. All of the interfaces between the four modules must be specified before work could begin….This is a modularization in the sense meant by all proponents of modular programming. The system is divided into a number of modules with well-defined interfaces; each one is small enough and simple enough to be thoroughly understood and well programmed [8]."

5.1 A DSM Model of the "Strawman" Design

We surmise Parnas viewed each module interface as comprising two parts: an exported data structure and a procedure invoked by Master Control. We thus took the choice of data structures, procedure declarations, and algorithms as the DP's of this design. The resulting DSM is presented in **Figure 5.** DP's A, D, G, and J model the procedure interfaces, as design rules, for running the input, shift, sort and output algorithms. B, E, H, and K model the data structure choices as design rules. Parnas states that agreement on them has to occur before independent module implementation can begin. C, F, I, L, and M model the remaining unbound parameters: the choices of algorithms to manipulate the fixed data structures. The DP dependencies are derived directly from Parnas's definitions.

	A	D	G	J	B	E	H	K	C	F	I	L	M
A - Input Type	.												
D - Circ Type		.											
G - Alph Type			.										
J - Out Type				.									
B - In Data					.	X	X						
E - Circ Data					X	.	X						
H - Alph Data					X	X	.						
K - Out Data								.					
C - Input Alg	X				X				.				
F - Circ Alg		X			X	X				.			
I - Alph Alg			X		X	X	X				.		
L - Out Alg				X	X			X	X			.	
M - Master	X	X	X	X									.

Figure 5: DSM for strawman modularization

The DSM immediately reveals key properties of the design. First, the design is a modularization, as Parnas claims: designers develop their parts independently as revealed by the absence of unboxed marks in the lower right quadrant of the DSM. Second, only a small part—the algorithms—is hidden and independently changeable. Third, the algorithms are tightly constrained by the data structure design rules. Moreover, the data structures are an interdependent knot (in the upper left quadrant). The shift data structure points into the line data structure; the alphabetized structure is identical to the shifted structure; etc. Change is thus doubly constrained: Not only are the algorithms constrained by the data structure rules, but these rules themselves would be hard to change because of their tight interdependence.

5.2 A DSM Model of a Pre-Modular Design

By declaring the data structures to be design rules, the designer asserts that there is little to gain by letting them change. Parnas's analysis reflects the costly problems that arise when the designer makes a mistake in prematurely accepting such a conclusion and basing a modularization on it. Furthermore, we can see that the design is also flawed because most of the design parameters are off limits to valuable innovation. The designer has cut off potentially valuable parts of the design space.

One insight emerging from this work is that there can be value in declining to modularize until the topography of the value landscape is understood. This conclusion is consistent with Baldwin and Clark's view: "...designers must know about parameter interdependencies to formulate sensible design rules. If the requisite knowledge isn't there, and designers attempt to modularize anyway, the resulting systems will miss the 'high peaks of value,' and, in the end, may not work at all [p. 260]."

Letting the design rules revert to normal design parameters and clustering the data structures with their respective algorithms (because they are interdependent) produces the DSM of **Figure 6.** This DSM displays the typical diagonal symmetry of outlying marks indicating a non-modular design. We have not necessarily changed any code, but the design (and the design process) is fundamentally different. Rather than a design overconstrained by Draconian design rules, the sense of a potentially complex design process with meetings among many designers is apparent. Innovative or adaptive changes to the circular shifter might have *upstream* impacts on the Line Store, for example—a kind of change that Parnas did not consider in his analysis.

	A	B	C	D	E	F	G	H	I	J	K	L	M
A - In Type	.												X
B - In Data	X	.	X		X	X		X	X			X	
C - In Alg	X	X	.										
D - Circ Type				.									X
E - Circ Data		X		X	.	X		X	X				
F - Circ Alg		X		X	X	.							
G - Alph Type							.						X
H - Alph Data		X			X		X	.	X			X	
I - Alph Alg		X			X		X	X	.				
J - Out Type										.			X
K - Out Data										X	.	X	
L - Out Alg		X						X		X	X	.	
M - Master	X			X			X			X			.

Figure 6: DSM for inferred proto-modular design

5.3 DSM for the Information-Hiding Design

The Line Store that is implicitly bundled with the Input Data is a proto-module that is a prime target for modularization: many other parameters depend on it and vice versa. Splitting the Line Store from the Input and giving each its own interface as a design rule is a typical design step for resolving such a problem. An alternative might be to merely put an interface on the pair and keep them as a single module. However, this DSM does not show that the Line Store is doing double-duty as a buffer for the Input Algorithm as well as serving downstream clients. Thus, it

is more appropriate to split the two. The other proto-modules are modularized by establishing interface design rules for them. The resulting DSM is shown in **Figure 7**. It is notable that this design has more hidden information (parameters O down to L in the figure) than the earlier designs. We will see that under our model, this permits more complex innovation on each of the major system components, increasing the net options value of the design.

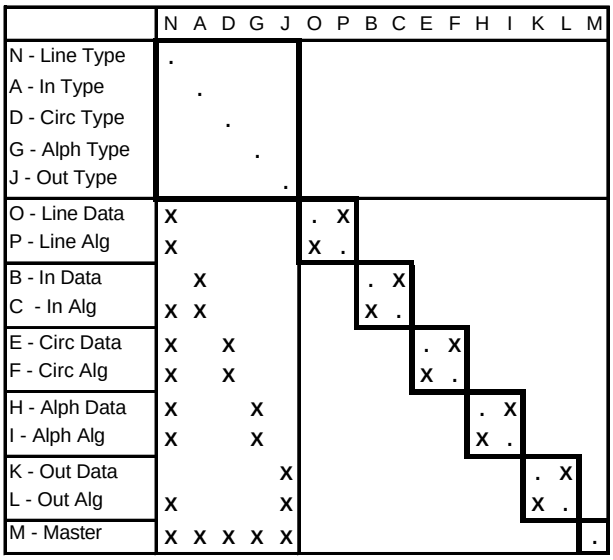

Figure 7: DSM for information hiding modularization

5.4 Introducing Environment Parameters

We can now evaluate the adequacy of DSM's to represent the information needed to reason about modular design in the style of Parnas. We find the DSM to be in part incomplete.

In particular, to make informed decisions about the choice of design rules and clustering of design parameters, we found we needed to know how changes in the environment would affect them. For example, we can perceive the value of splitting apart the Line Store and the Input design parameters by perceiving how they are independently affected by different parameters in the environment. For instance, Input is affected by the operating system, but the line store is affected by the size of the corpus. Indeed, the fitness functions found in evolutionary theories of complex adaptive systems, of which Baldwin and Clark's theory is an instance, are parameterized by the environment.

Not surprisingly perhaps, we were also finding it difficult to estimate Baldwin and Clark's *technical potential* term in the NOV formula, which models the likelihood that changing a module will generate value. This, too, is dependent on environmental conditions (e.g., might a change be required).

In this paper we address this lack with an extension to the DSM framework. We introduce *environment parameters* (EP) to model environments. The key property of an EP as distinct from a DP is that the designer does not control the EP. (Designers might be able *influence* EP's, however.) We call our extended models *environment and design structure matrices* (EDSM's). **Figure 8** presents an EDSM for the strawman KWIC design.

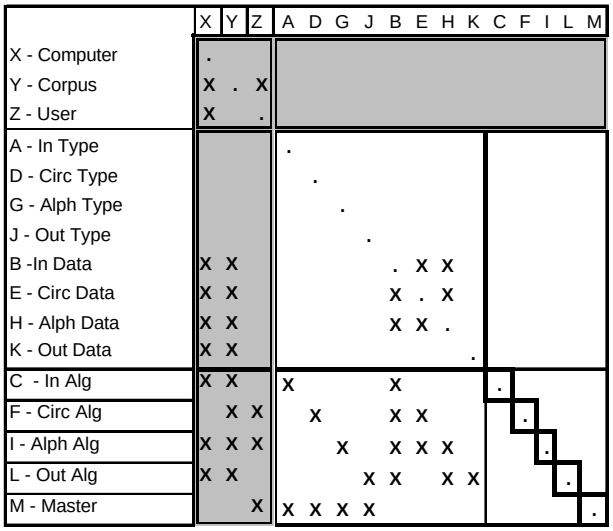

Figure 8: EDSM for strawman modularization

The rows and columns of an EDSM are indexed by both EP's and DP's, with the EP's first by convention. The upper left block of an EDSM thus models interactions among EP's; the middle left block, the impact of EP's on the design rules; the lower left block, their impact on the hidden DPs. The lower right block is the basic DSM, partitioned as before to highlight DR's; and the upper right block models the feedback influence of design decisions (DP's) on the environment (EP's).

Applying the EDSM concept to Parnas's example reveals that the EDSM provides a clear visual representation of genuine information hiding. In particular, the sub-block of an EDSM where the EP's intersect with the DR's should be blank, indicating that the design rules are invariant with respect to changes in the environment: only the decisions hidden within modules have to change when EP's change, not the design rules—the "load bearing walls" of the system. We can now make these ideas more concrete in the context of the KWIC case study.

Parnas *implicitly* valued his KWIC designs in an environment that made it likely that certain design changes would be needed. He noted several decisions "are questionable and likely to change under many circumstances [p. 305]" such as input format, character representation, whether the circular shifter should precompute shifts or compute them on the fly, and similar considerations for alphabetization. Most of these changes are said to depend on a dramatic change in the input size or a dramatic change in the amount of memory. What remains unclear in Parnas's analysis is what forces would lead to such changes in use or the computing infrastructure. We also do not know what other possible changes were ruled out as likely or why. At the time, these programs were written in assembler. Should Parnas have been concerned that a new computer with a new instruction set would render his program inoperable? A dramatic change in input size or memory size could certainly be accompanied by such a change.

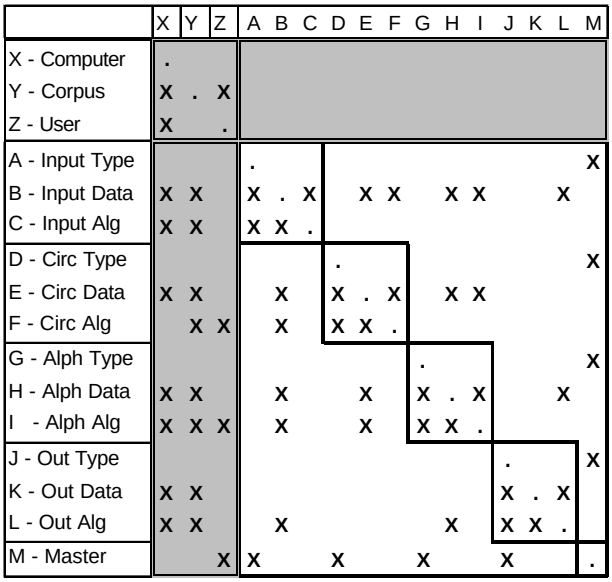

Figure 9: EDSM for inferred proto-modular design

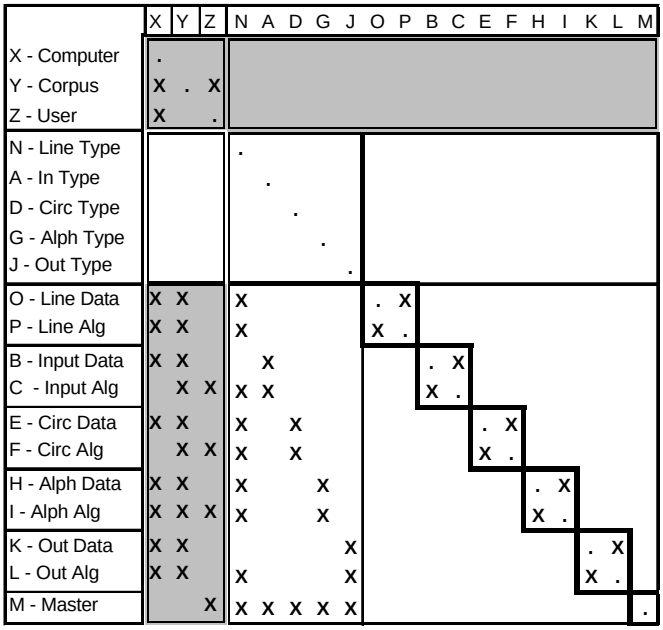

Figure 10: EDSM for information hiding Modularization

By focusing on whether internal *design decisions* are questionable rather than on the external forces that would bring them into question, the scope of considerations is kept artificially narrow. Not long ago, using ASCII for text would be unquestionable. Today internationalization makes that not so. By turning from design decisions to explicit EP's, such issues can perhaps be discovered and accounted for to produce more effective information-hiding designs.

To make this idea concrete, we illustrate it by extending our DSM's for KWIC. We begin by hypothesizing three EP's that Parnas might have selected, and which appear to be implied in his analysis: computer configuration (e.g., device capacity, speed); corpus properties (input size, language—e.g., Japanese); and user profile (e.g., computer savvy or not, interactive or offline). Figure **8**, **9**, and **10** are EDSM's for the strawman, pre-modular, and information hiding designs, respectively.

The key characteristic of the strawman EDSM is that the DR's are not invariant under the EP's. We now make a key observation: The strawman *is* an information-hiding modularization in the sense of Baldwin and Clark: designers can change non-DR DP's (algorithms) independently; but it *is not* an information-hiding design in the sense of Parnas. Basic DSM's alone are insufficient to represent Parnas's idea. We could have annotated the DP's with change probabilities, but we would still miss the essence: the load-bearing walls of an information hiding design (DR's) should be invariant with respect to changes in the environment. Our EDSM notation expresses this idea clearly.

Figure 9 is the EDSM for the pre-modular design in which the data structures are not locked down as DR's. The remaining DR's (the procedure type signatures) are invariant with the EP's, but the extensive dependencies between proto-module DR's suggest that changes in EP's will have costly ripple effects. The design evolution challenge that this EDSM presents is to split the proto-modules in a way that does not create new EP-dependent DR's. Figure 10 models the result: Parnas's information hiding design. The EDSM highlights the invariance of the DRs under the Eps in the sector where the EPs meet the DRs.

6. NOV-BASED ANALYSIS OF KWIC

We can now apply the NOV model to model how much the flexibility is worth in both Parnas designs as a fraction of the value of the base system, taking Parnas's notion of information hiding into account. This analysis is illustrative, of course, and the outputs are a function of the inputs. We justify our estimates of the model parameters using the EDSM's and reasonable back-of-the-envelope assumptions. A benefit of the mathematical model is that it supports rigorous sensitivity analysis. Such an analysis is beyond the scope of this paper; but we will pursue this issue in the future. We make the following assumptions and use the following notations in our analysis:

- N is the number of design parameters in a given design. For the proto-modular and strawman modularizations, $N = 13$. In the information hiding design $N = 16$.
- Given a module of p parameters, its complexity is $n = p/N$.
- The value of one experiment on an unmodularized design, $sN^{1/2}Q(1) = 1$, is the value of the original system.
- The design cost $c=1/N$ of each design parameter is the same, and the cost to redesign the whole system is $cN = 1$.
- The visibility cost of a module i of size n is $Z_i = S_{\text{j sees i}}cn$.
- One experiment on an unmodularized system breaks even: $sN^{1/2}Q(1) - cN = 0$.

Balwin and Clark make the *break-even* assumption for an example in their book [1]. For a given system size, it implies a choice of technical potential for an unmodularized design: in our

case, s = 2.5. We take this as the maximum technical potential of any module in a modularized version. This assumption for the unmodularized KWIC is a modeling assumption, not a precisely justified estimate. In practice, a designer would have to justify the choices of parameter values.

The model of Baldwin and Clark in quite sensitive to technical potential, but they give little guidance in how to estimate it. We have observed that the environment is what determines whether variants on a design are likely to have added value. If there is little added value to be gained by replacing a module in a given environment, no matter how complex it is, that means the module has low technical potential.

We chose to estimate the technical potential of each module as the system technical potential scaled by the fraction of the EP's relevant to the module. We further scaled the technical potential of the modules in the strawman design by 0.5, for two reasons. First, about half of the interactions of the EPs with the strawman design are with the design rules (but as we will see, their visibility makes the cost to change them prohibitive). Second—and more of a judgment call—the hidden modules in this design (algorithms) are tightly constrained by the design rules (data structures that are assumed not to change). There would appear to be little to be gained by varying the algorithm implementations, alone. Figure 11 shows our assumptions about the technical potential of the modules in the strawman and information-hiding designs.

	Strawman		***Info Hiding***	
Module Name	***sigma***	***Z***	***sigma***	***Z***
Design Rules	**2.5**	**1**	**0**	**1**
Line Storage	**NA**	**NA**	**1.6**	**0**
Input	**1.25**	**0**	**2.5**	**0**
Circular Shift	**1.25**	**0**	**2.5**	**0**
Alphabetizing	**1.25**	**0**	**2.5**	**0**
Output	**0.8**	**0**	**1.6**	**0**
Master Control	**0.4**	**0**	**0.8**	**0**

Figure 11. Assumed Technical Potential and Visibility

Figures 12 present the NOV data per module and Figure 13 the corresponding plots for the information hiding design. Figure 13 presents the plots for the strawman. The option value of each module is the value at the peak. We omit this disaggregated data for the strawman design. What matters is the bottom line. Summing the module NOV's gives that the system NOV is 0.26 for the strawman design but 1.56 for the information-hiding design. These numbers are percentages of the value of the non-modularized system, which has base value 1.

Thus the value of the system with the information-hiding design is 2.6 times that of the system with the unmodularized design, and the strawman's is worth only 1.26 times as much. Thus, the information-hiding version of the system is twice as valuable as the strawman. Ignoring the base value and focusing just on modularity, we observe that the information-hiding design provides *6 times more value in the form of modularity* than the strawman's design.

Baldwin and Clark acknowledge that designing modularizations is not free; but, once done, the costs are amortized over future evolution; so the NOV model ignores those costs. Accounting for them is important, but not included in our model. It is doubtful they are anywhere near 150% of the system value. On the other hand, they would come much closer to 26%, which would tend to further reduce the value added by the strawman modularization.

k	0	1	2	3	4	5	6	7	8	9	10	Max
Q(k)	0	0.4	0.68	0.89	1.05	1.17	1.27	1.35	1.42	1.49	1.54	NOV
Design Rules	0	-1.3	-1.6	-1.9	-2.25	-2.56	-2.88	-3.2	-3.5	-3.81	-4.1	0
LineStore	0	0.1	0.14	0.13	0.09	0.04	-0.03	-0.1	-0.19	-0.28	-0.4	0.14
Input	0	0.23	0.35	0.41	0.42	0.41	0.37	0.32	0.26	0.19	0.11	0.42
CirShift	0	0.23	0.35	0.41	0.42	0.41	0.37	0.32	0.26	0.19	0.11	0.42
Alpha	0	0.23	0.35	0.41	0.42	0.41	0.37	0.32	0.26	0.19	0.11	0.42
Output	0	0.1	0.14	0.13	0.09	0.04	-0.03	-0.1	-0.19	-0.28	-0.4	0.14
MsCon.	0	0.02	0.01	-0	-0.04	-0.08	-0.12	-0.2	-0.22	-0.27	-0.3	0.02

Fig 12. Option Values for Information Hiding Design

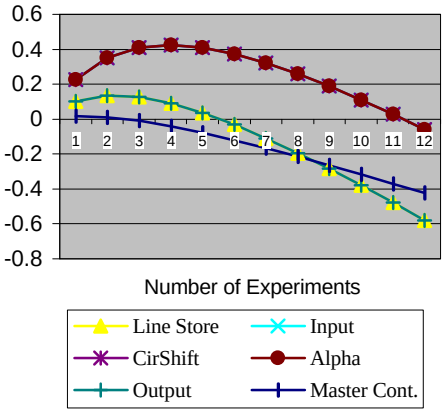

Figure 13: Options Values for Information Hiding Design

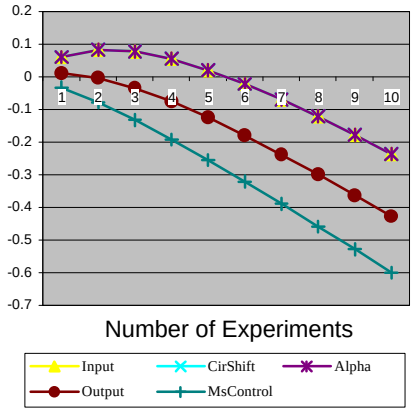

Figure 14: Options Values for Strawman Design

7. DISCUSSION AND CONCLUSION

Parnas's information-hiding criterion for modularity has been enormously influential in computer science. Because it is a qualitative method lacking an independent evaluation criterion, it is not possible to perform a precise comparison of differing designs deriving from the same desiderata.

This paper is a novel application of Baldwin and Clark's options-theoretic method of modular design and valuation to the subject of information-hiding modularity. Our goal was to lend insight into both information-hiding modularity and the ability of options theory to capture Parnas's intent of designing for change. We have provided an early validation of the application of their method to software design by reformulating Parnas's KWIC modularizations in the Baldwin and Clark framework.

Baldwin and Clark's method has two main components, the Design Structure Matrix (DSM) and the Net Option Value formula (NOV). DSM's provide an intuitive, qualitative framework for design. NOV quantifies the consequences of a particular design, thus permitting a precise comparison of differing designs of the same system.

We have shown that these tools provide significant insight into the modularity in the design of software. Yet, precisely modeling Parnas's information-hiding criterion requires explicitly modeling the environment—the context in which the software is intended to be used—in order to capture the notion of design stability in the face of change. We model the environment by extending DSM's to include environment parameters alongside the traditional design parameters. The environment parameters then inform the estimation of the technical potential in the NOV computation. In the process, we learned that Parnas had largely conceived of change in terms of intrinsic properties of the design, rather than in terms of the properties of the environment in which the software is embedded.

With these extensions to the Baldwin and Clark model, we were able to both model the Parnas designs and quantitatively show—under a set of assumptions—that the information-hiding design is indeed superior, consistent with the accepted results of Parnas.

This result has value in at least three dimensions. First, it provides a quantitative account of the benefits of good design. Second, it provides limited but significant evidence that such models have the potential to aid technical decision-making in design with *value added* as an explicit objective function. This paper is thus an early result in the emerging area of *strategic software design* [4], which aims for a descriptive and prescriptive theoretical account of software design as a value-maximizing investment activity. Third, the result supports further investigation of implications that follow from acceptance of such a model. For example, because the value of an option increases with technical potential (risk), modularity creates seemingly paradoxical incentives to seek risks in software design, provided they can be managed by active creation and exploitation of options. The paradox is resolved in large part by the options model, which clarifies that one has the right, but not a requirement, to exercise an option, thus the downside risk (cost) is largely limited to the purchase of the option itself.

In the introduction, we also raised the question of when is the right time to modularize or commit to a software architecture? Parnas's method says to write down the design decisions that are likely to change and then design modules to hide them. This implicitly encourages programmers to modularize at the early stages of design. The NOV calculations of the two KWIC modularizations make the possible consequences clear: without knowledge of the environment parameters, a designer might rush in to implement the strawman design, effectively sacrificing the opportunity to profit from the superior modularization. Yet, designers often do not have the luxury to wait until there is sufficient information to choose the optimal modularization. It may be difficult to precisely estimate how the environment is going to change—innovation and competitive marketplaces are hard to predict. Moreover, many of the best ideas come from the users of the software, so uncertainty is almost certain until the product is released. New design techniques that create options to delay modularizing until sufficient information is available might be explored as a possible solution to this conundrum.

The inclusion of environment parameters in the design process has additional implications. For example, making the most of these parameters requires being able to sense when they are changing and to influence them (slow their change) when possible. Careful design of the coupling between the development process and the environment is critical in strategic software design. For example, for parameters whose values are subject to change, *sensor* technologies—perhaps as simple as being on the mailing list of a standards-setting committee—can help to detect changes and report them to the designers in a timely fashion. Conversely, lobbying a standards-setting organization to, say, deprecate interfaces rather than change them outright can slow environmental change. Thus, accommodating environmental change is not limited to just *anticipating* change, as originally stated by Parnas, but includes more generally both *responsiveness* to change and *manipulation* of change.

This paper represents a first step in the validation of Baldwin and Clark's option-theoretic approach for quantifying the value of modularity in software. Additional studies are required to adequately validate the theory and provide insight into its practical application. Also, in our paper study, we found it difficult to estimate the technical potentials of the modules, despite the added resolution provided by the environment parameters. Validation in an industrial project would not only provide realistic scale, but it would also have considerable historical data to draw upon for the computation of NOV. Such studies would help move the field of software design further down the path to having powerful quantitative models for design.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grants CCR-9804078, CCR-9970985, and ITR-0086003. Our discussions with graduate students at the University of Virginia in CS 851, Spring 2001, at the have been very helpful.

REFERENCES

[1] Baldwin, C. Y. and Clark, K. B. (1999), *Design Rules: The Power of Modularity*

[2] Baldwin, C. and K. Clark, "Modularity and Real Options," Harvard Business School Working Paper 93-???, 1993.

[3] Beck, K. *XP Explained,*

[4] B. Boehm and K.J. Sullivan, “Software Economics: A Roadmap,” in *The Future of Software Engineering*, 22nd International Conference on Software Engineering, June, 2000.

[5] Eppinger, S.D. (1997). “A Planning Method for Integration of Large-Scale Engineering Systems.” Presented at the International Conference on Engineering Design.

[6] Favaro, J.M., K.R. Favaro and P.F. Favaro (1998), “Value Based Software Reuse Investment,” *Annals of Software Engineering 5,* , pp. 5 – 52.

[7] Garlan, D., Kaiser, G.E., and Notkin, D. *Using Tools to Compose Systems. IEEE Computer*, vol. 25, no.6. June 1992.

[8] Parnas, D. L. (1972) *On the Criteria to be Used in Decomposing System into Modules*

[9] Shaw, M., Garlan, D., Allen, R., Klein, D., Ockerbloom, J., Scott, C. and Schumacher, M. (1995) Candidate Model Problems in Software Architecture.

[10] Steward, D.V. (1981). “The Design Structure System: A Method for Managing the Design of Complex Systems.” *IEEE Transactions in Engineering Management* 28(3): 71-84

[11] Sullivan, K.J., P. Chalasani, S. Jha and V. Sazawal, “Software Design as an Investment Activity: A Real Options Perspective,” in *Real Options and Business Srategy: Applications to Decision Making*, L. Trigeorgis, consulting editor, Risk Books, 1999. (Previously Sullivan et al., “Software Design Decisions as Real Options,” Technical Report 97-14, University of Virginia Department of Computer Science, Charlottesville, Virginia, USA, 1997.)

[12] Sullivan, K.J., “Software Design: The Options Approach,” 2nd International Software Architecture Workshop, Joint Proceedings of the SIGSOFT '96 Workshops, San Francisco, CA, October, 1996, pp. 15--18.

[13] E.O. Teisberg, “Methods for evaluating capital investment decisions under uncertainty,” in Real Options in Capital Investment: Models, Strategies, and Applications, L. Trigeorgis, ed., (Westport, Connecticut: Praeger), 1995.

[14] Withey, J., “Investment Analysis of Software Assets for Product Lines,” Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-96-TR-10, 1996.