The Structure and Value of Modularity in Software Design

Kevin J. Sullivan
University of Virginia
Department of Computer Science
Charlottesville, VA 22904-4740 USA
Tel: +1 804 982 2206
sullivan@virginia.edu

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

Yuanfang Cai, Ben Hallen
University of Virginia
Department of Computer Science
Charlottesville, VA 22904-4740 USA
Tel: +1 804 982 2200
{yc7a,bh5z}@virginia.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 added value in a given context. We need better explanatory and prescriptive models of the nature and value of information hiding. 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 apply it to Parnas's KWIC designs. We contribute an extension to design structure matrices, and we 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
Design, Economics

Keywords
Software, design structure matrix, real options, modularity.

1. INTRODUCTION
In 1972 Parnas introduced *information hiding* as an approach to devising modular structures for software designs [7]. 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.

ESEC/FSE 2001, Vienna, Austria

This idea has been broadly influential, contributing to 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 most effectively to *design 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 this stage they are often unsure 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 principal way to evaluate such a modularization is to test whether it hides the information it was meant to hide, which is circular. We lack precise, quantitative models of the value of proposed modularizations. In particular, we lack models for assessing the relative values of alternative 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 proto-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].

A key idea in this work is that modularity in design creates value in the form of *real options* to improve a system by experimenting with new implementations and substituting any superior ones that are found. The value of such options is modeled quantitatively. This idea, which is consistent with our work on the options value of flexibility in software [3][10][11], promises to help link structural aspects of design to value objectives. Parnas's ideas did influence 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 the theory. Second, the resulting mapping will produce valuation results consistent with Parnas's widely accepted conclusions.

Mapping the example did reveal a 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 certain decisions under changes in the environment.

With the extended theory we were able to value each of the two modularizations against an inferred, ancestral, *proto-modular* design. Both modularizations are seen to derive from that design by application of the *splitting operator* of the theory's design evolution framework. We found the extended theory predicts that Parnas's *information hiding design* is significantly better than his *strawman design*, given our modeling assumptions. Our results provide a data point suggesting that the extended theory has the potential to advance the development of precise models having descriptive and prescriptive power for value-based software design.

The next two sections provide the background material for our analysis and discussion of KWIC, which follow 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*. 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 positive present value. It is like a lottery ticket.

The important idea in this paper is that a module creates an option to invest in a search for a superior replacement and to replace the currently selected module with the best alternative discovered, or to keep the current one if it is still the best choice. Intuitively, the value of such an option is the value that would be realized by the optimal experiment-and-replace policy. Knowing this value can help a designer to reason about both the investment in modularity, and how much to spend 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 [11] 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 [10] formalized that options-based analysis, focusing in particular on the value of the flexibility to delay decision making. Withey [12] 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 [5] 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 raises theoretical concerns. It is valid only if market-priced assets can be identified that track the risks in the option being priced. Favaro assumed this condition to hold. There are many cases in design where there are no tracking assets. Beck [2] is popularizing intuitions arising 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 real options. They do not assume that tracking assets exist. Rather, their option valuation formula is statistical. It assumes that the added values of independently developed alternatives to existing modules are normally distributed. We discuss the details in the following section.

3. BALDWIN AND CLARK'S MODEL

The parts of Baldwin and Clark's theory that are most relevant to this paper are the representation of modularity in design by *design structure matrices* (DSM) [4][9]; the evolution of modular designs as adaptive systems under the action of *modular operators*; and the *net options value* (NOV) of modularity.

3.1 Design Structure Matrix

A DSM represents a design space to be searched for valuable designs. The DSM represents the design space in terms of its *design parameters* (DP); the *values* of these parameters, if bound; and *dependencies* amongst parameter values. A design parameter, in turn, represents the need for a choice to be made about some aspect of a design. Design parameters for software could represent choices of data structures, algorithms, type signatures, user interface look-and-feel, real time response characteristics, security aspects, power usage, etc.

The rows and columns of a DSM are labeled by design parameter names. The design parameters are represented by the diagonal entries. A dependence between two design parameters is represented by an off-diagonal mark (X) in the relevant DSM matrix position.

DSM's can be derived from experience with prior versions of a product or similar products. In early product versions, there may be few marks in the matrix, capturing little more than the dependencies necessary to deploy a functioning product. A DSM for a subsequent version might include marks for subtle but powerful dependencies 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.

Figure 1 presents an example. The parameters are A, B, and C. The X in row B, column A means that an efficacious choice for B depends on the choice made for A. Parameters requiring mutual consistency are *interdependent*, resulting in symmetric marks: (B,C) and (C,B). Algorithm and data structure choices go hand-in-hand, for example. When one parameter choice must precede another the parameters are said to be *hierarchically dependent*: (B,A). The choice of a component standard typically precedes choices about component design, for example. *Independent* parameters, such as truly hidden data representation choices, can be changed without coordination.

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

Figure 1: DSM for a design of three parameters.

Choosing design parameters and their values are two key design activities. First a design space is established, then it is searched for high-value designs. Marks in a DSM represent the belief that benefits can be gained by recognizing and permitting dependencies among parameter values and by respecting them during the search process.

DSMs thus represent not only abstract parameter dependencies, but concrete requirements for designers to *communicate* in order to search the design space efficiently and effectively. Consider Figure 1 again. If the value selected for parameter A is changed, the need to revisit the choice for B has to be communicated. A change to B could affect C, then perhaps B again. Parameter dependencies constrain the ability of designers to make decisions independently, complicating design development and evolution.

A third key design activity is thus to structure the design space and decision-making process by structuring the design parameters and their dependencies. Two key activities in this dimension are the clustering of related design parameters into *proto-modules* and corresponding *design tasks*, and applying a modular operator, *splitting*, to decouple design parameters and the corresponding design tasks.

3.2 Design Rules and Design Evolution

A group of interdependent design parameters is clustered into a *proto-module* to show that the decisions are managed collectively as a single design task. See Figure 2. Dark lines denote proto-module clusters. A proto-module is essentially a composite design parameter. To be a true module there should be no marks in the rows or columns outside the bounding box of its cluster that connect it to other modules or proto-modules in the system.

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

Figure 2: DSM for a proto-modular design.

Merely clustering a monolithic design into proto-modules does not produce a modular design. In Figure 2, for example, B depends on A, and so the B-C proto-module depends on the A proto-module. To obtain a modular design, dependencies across proto-modules must be broken. Breaking them requires the use of the modular operator called splitting.

The first step in splitting is to identify the cause of the dependence and to reify the issue as a separate design parameter. The value of the new parameter will constrain the values of the original parameters. They will therefore come to depend on it. The second step is to fix the value of the new parameter in a way that permits the original parameters to be changed independently as long as they are only changed in ways consistent with the new constraint.

Consider again the dependence of B on A in Figure 2. This dependence can be eliminated, permitting choices for A and B–C to be made independently, by introducing a new parameter, I, as illustrated in Figure 3. B no longer depends on A. Instead both take on a hierarchical dependence on I. A key instance of this operation in software is the introduction of an abstract data type module interface. I would represent an *A-interface* parameter. The B implementation parameter would be constrained to access A only through the interface represented by the value of I. Thus, A could change its implementation freely without affecting B, as long as A's interface did not have to be changed as well.

The A and B tasks must wait for the I task to complete—for a particular interface definition to be assigned as the value of I. If that value is changed, A and B might have to change, as well. A design parameter, such as I, that decouples otherwise dependent proto-modules, is a *design rule* when it has a value and an assertion is made that that value is intended not to change. Design rules impose constraints that other parameters must respect and that they can assume are stable. Design rules constrain and structure the design space and search process. Figure 3 uses a darker box around the I column to signify that I is a design rule for this design.

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

Figure 3: DSM for a modular design obtained by splitting.

Modularization through the imposition of design rules is the key to structuring the design space and search. The modularization that introducing I achieves is reflected in the block-diagonal structure of the rest of the DSM, with A and B–C becoming truly independent modules. In Baldwin and Clark's terminology, a design rule, such as I, is a (or is part of) a *visible module*; and a 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 modular operator called *substitution*, which we address in detail in the next subsection. Splitting and substitution are two of six *modular operators* that Baldwin and Clark introduced to model design evolution in a complex adaptive systems framework. Splitting serves to modularize proto-modular designs. The other operations modify already modular designs. They 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. In evolving a design by exploring and possibly moving to adjacent points in design space, it is useful to be able to evaluate designs using quantitative models of value. Such models need not be perfect. However, they must capture the most important terms and their assumptions and operation must be known and understood so that analysts understand and 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 both multiplies and decentralizes *real options* that increase the value of a design. At one extreme, non-modular systems can only be replaced whole—even if only by an incrementally better version—and the authority to accept changes is centralized. The designer has one option: to accept a whole "portfolio" of assets—a candidate replacement system with good and bad parts intertwined.

On the other hand, a system of independent modules can be kept as is, or any or all modules can be replaced independently. Improvements can be accepted while regressions can be rejected. The replacement decisions are decentralized in the sense that the designers responsible for modules can make substitution decisions without coordination. Modularity provides a portfolio of options. A result in modern finance shows that, all else being equal, a portfolio of options is worth more than an option on a portfolio. It is in this sense that modularity, per se, adds value.

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 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 expected payoff of exercising a search and substitute option optimally, accounting for both benefits and exercise costs.

Baldwin and Clark present a model for calculating *NOV*. A module creates an opportunity (a) to invest in k experiments to 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 to be gained by accepting the best positive-valued candidate generated by k_i independent experiments, under certain assumptions about the distribution of the values of such candidates. $C_i(n_i)k_i$ 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_j 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 illustrates the *NOV* computation. The value of the expected maximum value alternative found by exploration increases with k, the number of experiments. However, experimentation costs meet diminishing returns on this search. The *max*, *NOV*, is the peak as a function of k. In this illustrative case, generating six module alternatives is expected to maximize the gain, adding about 40% to the value of the current system.

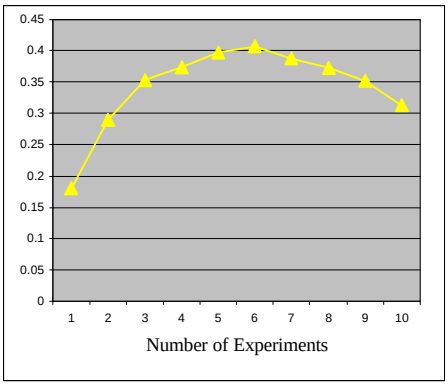

Figure 4. Expected value added by *k* experiments.

The *NOV* 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) [7]. The use of *KWIC* as a benchmark for assessing concepts in software design is well established [6][8].

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 uses abstract data type interfaces to decouple key design decisions involving data structure and algorithm choices so that they can 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 (and in the earlier works of Eppinger and Steward [4][9]), 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 based on very rough parameter values, not precise market valuations; yet they are revealing. We answer two questions. Do the DSM's contain key information needed 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 sufficiently 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 endogenous: 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. They are exogenous. 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. Then 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 [7]."

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, namely the algorithm choices, constituting the space to be searched by the design process. We derived the DP dependencies 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, it is a modularization, as Parnas claims: designers develop 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, and the data structures are an interdependent knot (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 data structure rules, but these rules are hard to change because of their interdependence.

5.2 A DSM Model of a Proto-Modular Design

By declaring the data structures to be design rules, the designer asserts there is little to gain by letting them change. Parnas's analysis reflects the costly problems that arise when the designer prematurely accepts such a view and bases a modularization on it. The design is also problematical because most 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 value landscape is sufficiently well 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 [1]."

Letting the design rules revert to normal design parameters and clustering data structures with algorithms yields the inferred proto-modular DSM of Figure 6. This DSM displays the typical diagonal symmetry of outlying marks indicating a proto-modular design. We have not changed any code, but the design and the design process are fundamentally different. Rather than a design constrained by Draconian design rules, the sense of a 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.

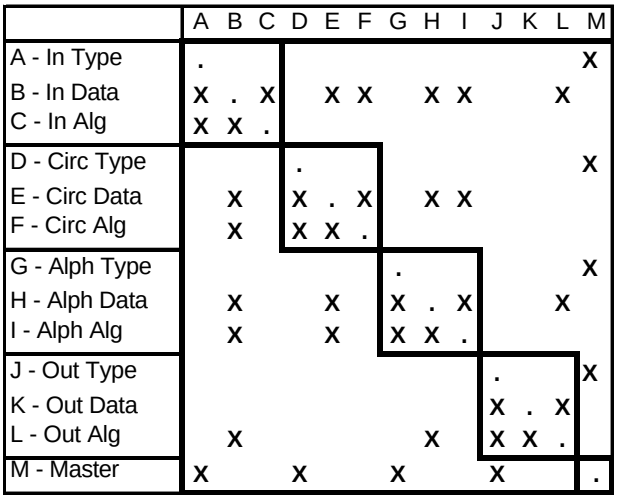

Figure 6: DSM for inferred proto-modular design

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.

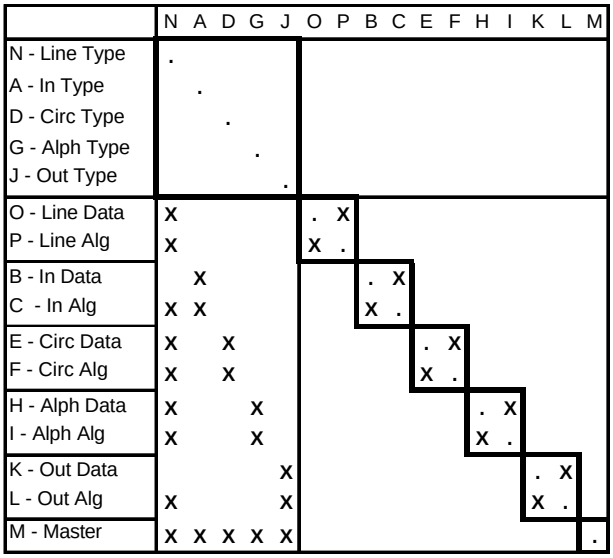

Figure 7: DSM for information hiding modularization

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. 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 also found it hard to justify estimates of the *technical potential* term in the NOV formula, which models the likelihood that changing a module will generate value. This, too, depends on environmental conditions (e.g., might customers be willing to pay for a change).

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.

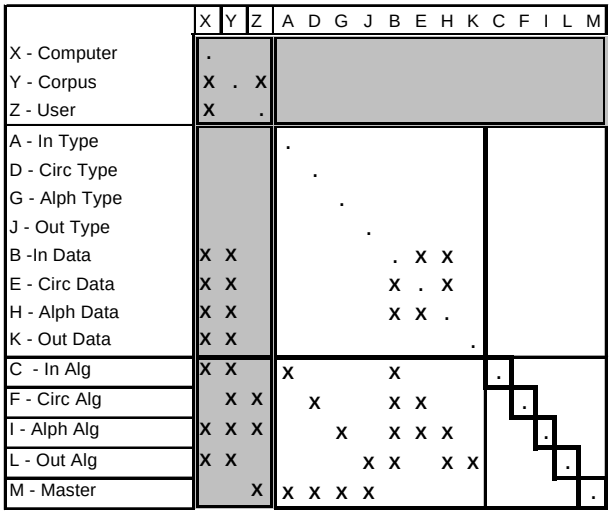

Figure 8: EDSM for strawman modularization

The rows and columns of an EDSM are indexed by both EP's and DP's, with the EP's first by convention. The shaded upper left block of an EDSM models interactions among EP's; the shaded middle left block, the impact of EP's on the design rules; the shaded lower left block, their impact on the hidden DPs; the unshaded lower right block is the basic DSM, partitioned as before to highlight DR's; and the shaded upper right models any feedback influence of design decisions (DP's) on the environment (EP's). Such influence is possible without control.

We now argue that EDSMs provide a clear visual representation of genuine information hiding. Information hiding is indicated

when the sub-block of an EDSM where the EP's intersect with the DR's is blank, signaling that the design rules are invariant with respect to changes in the environment. In this case, 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 make these ideas concrete in terms of KWIC.

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.

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). Figures 8, 9, and 10 are EDSM's for the strawman, proto-modular, and information hiding designs, respectively.

The key characteristic of the strawman EDSM is that the DR's (A,...,K) are not invariant under the EP's (X,Y,Z). 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. However, it is *not* an information-hiding design in the sense of Parnas. Basic DSM's alone are thus 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 captures this idea.

Figure 9 is the EDSM for the proto-modular design. The procedure type signature parameters are invariant with the EP's, but the extensive dependencies between DR's suggest that changes in EP's could 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.

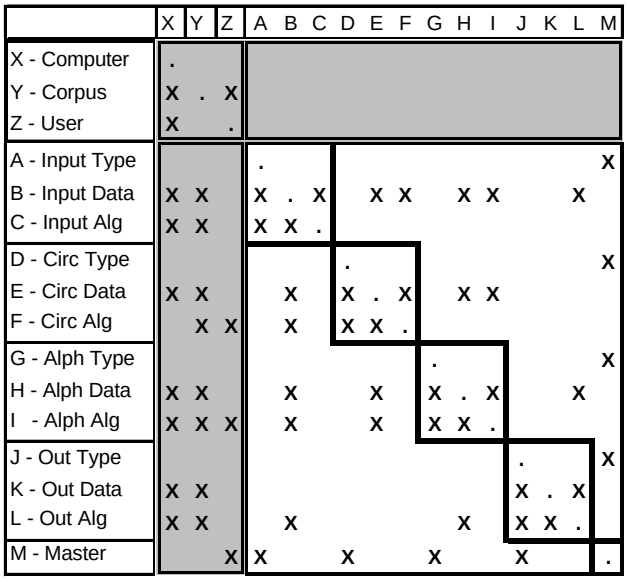

Figure 9: EDSM for inferred proto-modular design

Figure 10 models a genuine information hiding modularization: Parnas's information hiding design. The EDSM highlights the invariance of the DRs (N,...,J) under the EPs (X,Y,Z) in the sector where the EPs meet the DRs.

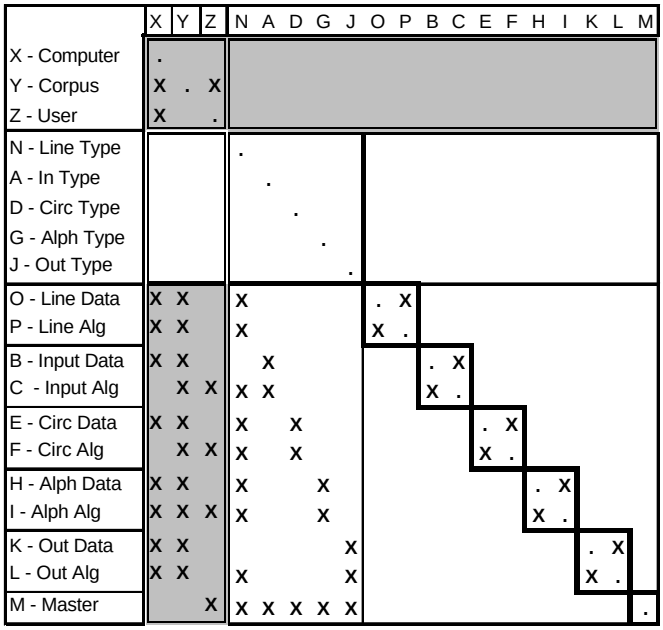

Figure 10: EDSM for information hiding Modularization

6. NOV-BASED ANALYSIS OF KWIC

We can now apply the NOV model to estimate how much the flexibility is worth in Parnas's designs as a fraction of the value of the base system. Our analysis is mostly illustrative. They key parameter value is sigma. Finding proxies for estimating sigma is a major challenge in all real options applications, and we lack the kind of data that would be needed to do so convincingly for the Parnas case study. It remains an open challenge to strongly justify precise estimates for real options in software design. We use back-of-the-envelope assumptions refined using EDSMs. A

benefit of the mathematical model is that it does support rigorous sensitivity analysis. Predictions based on good judgment that are also robust to minor variations in parameter values can provide valuable insights. Sensitivity analysis is beyond the scope of this paper; but we will pursue it 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 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 generally have to justify the choices of parameter values more convincingly.

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 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 and visibility 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

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 14 presents the plots for the strawman. The option value of each module is the value at the peak. We omit the 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 proto-modularized system, which has base value 1.

Thus the value of the system with the information-hiding design is predicted to be 2.6 times that of the system with the unmodularized design; and the strawman's, worth but 1.26 times as much. Our model suggests that the information-hiding version of the system is twice as valuable as the strawman. Ignoring the base value and focusing just on modularity, the model predicts that the information-hiding design provides *6 times more value in the form of modularity* than the strawman 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

Figure 12. Option Values for Information Hiding Design

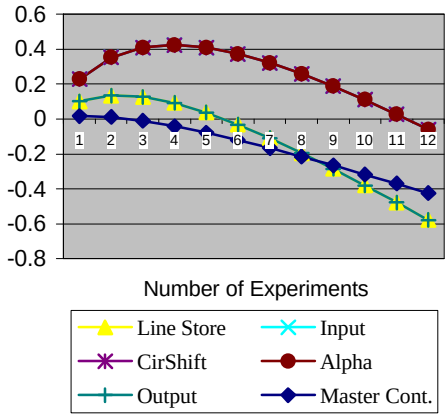

Figure 13: Options Values for Information Hiding Design

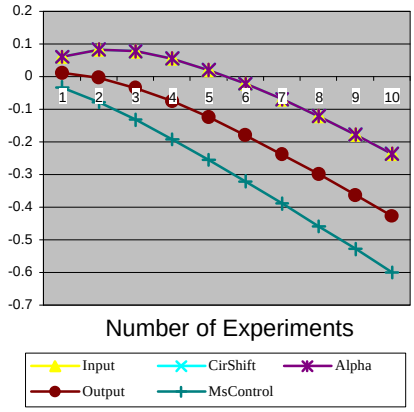

Figure 14: Options Values for Strawman Design

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 presents a novel adaptation and application of Baldwin and Clark's options-theoretic model to the subject of information-hiding modularity in software. Our goal was to gain insight into both information-hiding modularity and the ability of the theory to capture Parnas's notion of design for change. We provide an early test of the application of the theory 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 representing design spaces and the structures of the activities by which they are searched. NOV quantifies the value added to a design beyond the value of its basic function by modularity. In principle, the theory provides at least an outline of a quantitative value-based framework for comparing designs.

We found that capturing Parnas's information-hiding criterion in terms of DSMs requires modeling the environment in which the software is intended to be used. 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 model both Parnas designs effectively, and to justify *relative* parameter estimates for the NOV model. The DSM modeling framework appears to have significant potential value as a representational framework for software design. Scalability of the diagrams is an issue. We are investigating hierarchical DSMs as a solution.

Our models also predict, based on our modeling assumptions, that the information-hiding design is indeed superior to the strawman design. Our results are consistent with the accepted results of Parnas. This result has value in at least three dimensions. First, it provides some evidence that it might be possible to develop a quantitative account of the value of good design. Second, it provides limited but significant evidence that such models have the potential to support a discipline of *design for value added*. This paper is thus an early result in the emerging area of *strategic software design* [3], 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, in the sense of variation in returns), modularity creates seemingly paradoxical incentives to seek risks in software design, provided they can be managed by creating and managing options. The paradox is resolved in part by the options model, which clarifies that one has the right, but not a requirement, to exercise an option, thus the downside risk is largely limited to the purchase of the option itself.

Some qualifications are needed. First, we did not provide compelling *absolute* estimates of the technical potential. To do so appears to be impossible for KWIC because it is divorced from real markets. A key question is how to justify estimates, especially of sigma, for real projects. The *relative* valuations that we used in this paper are conservative, in that the value to be gained by varying algorithms but not data structures is low. Second, the NOV model builds on simplifying assumptions. For example, it assumes that substitute modules add value to a system independently. Richer models might be needed to have generalized prescriptive utility for software design. Our results show that models of this sort can provide insight and that they have some potential to support design, but not that designers should try to use them in practice at this time.

In the introduction, we asked when is the right time to modularize or commit to an architecture? Parnas's method is to identify design decisions that are likely to change and to design modules to hide them. It encourages programmers to modularize early. The NOV calculations of the KWIC modularizations make the dangers clear: without knowledge of the environment parameters and design parameter dependencies, a designer might rush in to implement the strawman design, sacrificing the opportunity to profit from better designs. Yet, designers often cannot wait for complete information to choose a perfect 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 software users, so uncertainty is certain until the product is released. New design techniques that create options to delay modularizing until enough is known 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. Careful design of the coupling between the development process and environment is critical in strategic software design. 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.

We plan several directions for additional future work. First, we need to understand what kind of proxies can support estimates of technical potential. Historical databases and markets are likely data sources. Second, evaluation in real projects is needed to provide realistic scale and access to data for estimating NOV. Third, we will develop and evaluate tool support for large, hierarchical EDSMs and use it to evaluate how well EDSMs can aid in designing large systems. Fourth, EDSMs seem to lead to a generalized concept of interface that we will explore: comprising all of the dependencies that cross module boundaries. Finally, we will explore the notion that software designs evolve on fitness landscapes as adaptive systems. Such studies would help move the field of software design toward having powerful quantitative models of design value and nearer to a genuine science of design.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grants CCR-9804078, CCR-9970985, and ITR-0086003. Carliss Baldwin provided valuable comments on option valuation. Discussions with graduate students in the Software Design Seminar, CS 851, at the University of Virginia, Spring 2001, were inspiring and helpful.

REFERENCES

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

[2] Beck, K. *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[3] B. Boehm and K.J. Sullivan, "Software economics: a roadmap," in *The Future of Software Engineering*, 22nd International Conference on Software Engineering, June, 2000. pp. 319–344.

[4] Eppinger, S.D.. "A planning method for integration of large-scale engineering systems," International Conference on Engineering Design, 1997.

[5] Favaro, J.M., K.R. Favaro and P.F. Favaro (1998), "Value-based software reuse investment," *Annals of Software Engineering 5*, , pp. 5 – 52.

[6] Garlan, D., Kaiser, G.E., and Notkin, D., "Using tools to compose systems," *IEEE Computer*, vol. 25, no.6. June 1992.

[7] Parnas, D. L., "On the criteria to be used in decomposing system into modules," *Communications of the ACM*, Vol. 15, No. 12, December 1972 pp. 1053–1058.

[8] Shaw, M., Garlan, D., Allen, R., Klein, D., Ockerbloom, J., Scott, C. and Schumacher, M. "Candidate model problems in software architecture," Computer Science Department, Carnegie-Mellon University, Technical Report (Shaw 95-2), January, 1995.

[9] Steward, D.V., "The Design Structure System: A Method for Managing the Design of Complex Systems." *IEEE Transactions in Engineering Management*, vol. 28, no. 3, 1981, pp. 71-84.

[10] 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.)

[11] 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.

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