A Value-Oriented Theory of Modularity in Design

Yuanfang Cai
Dept. of Computer Science
University of Virginia
Charlottesville, VA, 22904-4740 USA
yc7a@cs.virginia.edu

Kevin J. Sullivan
Dept. of Computer Science
University of Virginia
Charlottesville, VA, 22904-4740 USA
sullivan@cs.virginia.edu

1. INTRODUCTION

We were motivated to undertake the research we describe here by a conversation with two practicing software engineers, who described a dilemma they faced at work. They worked for small company that earned revenues by delivering to a large customer a stream of enhancements to a software tool. The engineers' jobs were to estimate the time to make enhancements and to implement selected enhancements. They were good at estimating, but dissatisfied with the system design, believing that it significantly slowed new feature implementation. They had proposed to management to restructure the tool. However, the management, concerned about disrupting the flow of enhancements thus revenues, and having no clear model of likely benefits, declined. The engineers believed that refactoring would increase the velocity of feature delivery, but they had no sense or ability to analyze the situation quantitatively or to frame it in a way that was compelling to business decision-makers. As a result, the engineers were dissatisfied, and the company incurred a possibly significant opportunity cost.

The problem they faced is one that organizations everywhere grapple with. Should we make a costly investment in design: in this case, in restructuring a system? The problem that we set out to address is that as a discipline, we lack the basic science needed to analyze fundamental design decisions such as this one. Without models and analysis, it's hard to justify costly investments in design, to make major design decisions confidently, or to perhaps even to make them reliably over time.

Our goal is to develop and validate a formal theory of design leading to modeling and analysis methods that support better, value-oriented design decision-making. We are particularly concerned with questions concerning coupling, in general, and of modularity in software design, in particular, which we view as a matter of coupling structure at the level of software design decisions.

EDSER'05, May 15, 2005, St. Louis, Missouri, USA.

Theorists have long recognized the role of coupling in determining such properties as ease of understanding and verification, adaptive capacity, time to market, and quality [1, 3, 2, 12, 14, 11, 4, 6, 15]. Sullivan at al. [17, 16] have suggested that options-based models can provide insights concerning modularity, phased project structures, delaying of decisions and other dynamic strategies. Withey [12] applied a related analysis to reason about the flexibility value of software product line architectures. Favaro [8] developed an options approach to investment analysis for software reuse infrastructures. He used a Black/Scholes valuation model, which, in the absence of complete markets, is not clearly an appropriate model. Baldwin and Clark [3] appear to have been the first to propose that the value of modularity in design (of computer systems) can be modeled as options and valued statistically without assuming complete markets. None of this work has yet produced a precise and generalized model of modularity in design.

We need a theory to bridge the gap between rigorous design modeling and economic design outcome reasoning. Our overall research agenda is to develop such a theory as basis for helping software designers make better design decisions. We see three steps to this goal.

2. OVERALL AGENDA

Our first step was is to establish a design modeling approach based on a view of design as a process of making decisions coupled by a variety of constraints. At this early point in developing our theory, we make at least three simplifying assumptions. First, we consider a point in time at which a set of decisions has been framed, possible choices for each decision are enumerated, and constraints on the set of decisions are understood. Real design involves ongoing decision-framing, not just decision-making in a fixed design space. Second, we assume that each design decision comprises a finite set of options. In reality, decisions cab involve real number and other infinite domains. Third, we assume that constraints on decisions are expressible in a standard logic. It's unclear how constraining this assumption is in practice.

At the same time, the approach has advantages. It focuses on design spaces and their structures rather than on the individual design. It supports the modeling of a wide variety of decisions and constraints. And it operates at a high level of abstraction, promising to enable reasoning about design properties based on early and abstract models.

Having settled on a logical scheme for representing design spaces in this logical form, our next step is to develop a coupling theory for this modeling approach. The question is, if you change a decision in a design, what other decisions might have to change? It turns out that this question is a little more subtle that we had expected.

Our final step is to link economic models with these formalizations to bridge the gap between design and value. For example, we will address the question posed at the beginning of this paper by associating costs with given changes in a system, and comparing the present value of the current and restructured designs net of switching costs and risks.

We present our model in detail in a recent work [5]. We model design spaces as *Finite Domain Constraint Networks (CN's)* augmented with models of asymmetric dominance relations in decision-making (e.g., that a choice of interface can constraint a choice of implementation but not the other way around), and the clustering of design decisions into proto-modules (e.g., all choices of interfaces into an *architecture* module). Our coupling theory rests on what we call the *Design Automaton (DA)*, and a procedure for deriving DA's from CN's. From the concept of DA, we formally defined the concept of pair-wise design decision *dependence*, and developed an algorithm to compute pair-wise coupling structures from declarative design models.

3. CURRENT STATUS

In this section, we briefly introduce the formal concept of CN, DA, and pair-wise coupling structure.

We take finite-domain constraint networks as the core element of our representational framework. Each dimension of a design is represented by a variable. A decision is represented by a binding of a value from a finite set of possible values to such a variable. Dependences among design decisions are expressed as logical constraints. Such a set of variables, domains, and defines a design space, as the set of variable bindings that satisfy the given constraints. We found it necessary to augment CN's with dominance relations on variables and a clustering of variables into proto-modules to model situations where some decisions have precedence over others, and in which the concerns are partitioned into proto-modules.

We use a notion called Design Automaton (DA) to capture variation and the impacts of changes in these spaces. We compute DA's from CN's and dominance relations. In a DA, each state is a valid design under the constraints. Each transition is marked by a change in one decision. The destination of a transition is a minimally perturbed version of the initial state that preserve the given change and that include a minimal set of changes in other variable values. The overall change is minimal in the sense that none of these changes can be undone while maintaining consistency. In general there are multiple ways to restore consistency for a given change, and so the DA is nondeterministic.

We formally defined pair-wise design decision interdependence based on the DA. Pairwise interdependence is an important abstraction of the coupling structure captured in full detail in the DA. A great deal of machinery for reasoning about designs has developed around concepts of pairwise coupling. The design structure matrix (DSM) is a notable, but not the only, example. A DSM is a square matrix with design decisions on the rows and columns, and marks (X's) in the cells that represent dependences. From the structure of the marks in a DSM, one can reason about coupling properties and their implications for design processes, such as available parallelism in design decision-making, modularity or near-modularity, and its economic value [3, 15].

We define two design variables to be pair-wise dependent if, for some design (i.e., some state of the DA), there is some change to the first variable (i.e., a transition trigger) for which the second variable is changed in one of the perturbations (i.e., destination states from the originating state for the given trigger).

We have formalized our theory, including constraint networks, the DA, and the CN-to-DA mapping using Z [13]. Space constraints prevent us from presenting the full theory. The following Z schemas give a flavor of the model. In a standard manner we model a constraint network as a triple (V, D, C). V is a finite set of variables; D, their *domains*; and C, a set of *constraints* on subsets of V. A constraint is modeled as a set of permitted assignments to the variables to which it applies. A valuation of the variables is consistent if and only if its projection onto the variables of each constraint is consistent with that constraint's permitted assignments.

The *cannot_influence* relation models dominance relation among design decisions and filters otherwise valid transitions of the DA. The schema *DA*, below, defines a DA as a nondeterministic finite automaton. The schema *cn2da* formalizes the mapping from a CN to a DA. The solutions of the CN constitute the state set of the DA. Each transition of the DA is triggered by a variable-value binding that represents a change in one decision. A transition has to satisfy the following conditions. First, the triggering change must be present in the destination state. Second, there is a transition between states only if the valuation difference between them is minimal, in that no subset of their differences would produce a consistent state. Third, the *cannot_influence* relation must not be violated. If (x, y) is in *cannot_influence*, then among all the possible ways to restore consistency in the face of a change to x, those involving y are excluded.

We elide certain definitions used in this schema. The function *bindingDiff* compares two assignments and returns the bindings in the second assignment that are different from those first assignment. The function *variablesInBinding* extracts variables involved in an assignment. The function *replace* assigns different values to a set of variables in an assignment, and returns a new assignment. Based on this formal mapping, we have developed an automatic DA derivation procedure.

ConstraintNetwork

$V : \mathbb{F}\ Variable$
Domains
Constraints

---

dom *domain* $= V$
dom *constraints* $= \mathbb{F}\ V$

$$
\begin{array}{|l}
\hline
DA \\
\hline
states : \mathbb{F}\, Assignment \\
alphabet : \mathbb{F}(Variable \times Value) \\
transition : (Assignment \times (Variable \times Value)) \rightarrow \mathbb{F}\, Assignment \\
\hline
\mathrm{dom}(\mathrm{dom}\, transition) = states; \\
alphabet = (\mathrm{ran}(\mathrm{dom}\, transition)); \\
(\mathrm{ran}\, transition) = \mathbb{F}\, states; \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
cn2da : (ConstraintNetwork \times cannot\_influence) \rightarrow DA \\
\hline
\forall\, cn : ConstraintNetwork;\ cif : cannot\_influence;\ da : DA \bullet \\
cn2da(cn, cif) = da \Rightarrow \\
\quad (da.alphabet = cn.domain) \wedge da.states = solutions(cn) \wedge \\
(\forall\, start : valuations(cn);\ binding : cn.domain; \\
\quad endstates : \mathbb{F}\, Assignment \bullet \\
\quad da.transition(start, binding) = endstates \\
\quad \Rightarrow (\forall\, end : endstates \bullet (binding \in end.valueof) \wedge \\
\quad (\forall\, sub : \mathbb{F}(Variable \times Value) \mid sub \subset bindingDiff(start, end) \\
\qquad \bullet\ replace(start, sub) \notin solutions(cn)) \wedge \\
\quad (\forall\, var : variablesInBinding(bindingDiff(start, end)) \bullet \\
\qquad var \notin (cif.cntinf(\!| \{firstbinding\} |\!))))) \\
\hline
\end{array}
$$

We have developed a prototype tool that we call *Simon* that mechanizes the theory. As a test of the potential utility of the theory, we have developed two applications [5]: automatic DSM derivation from augmented CN models and what we call *multi-step non-deterministic design impact analysis.*

Our initial validation experiments are in the form of case studies of Parnas's KWIC, as modeled in our previous work on the options value of modularity [15], and of a model of a web service system developed by Lopes at al. [10] to test the hypothesis that aspect-oriented modularity has economic value.

We showed that we could compute DSM's from logic-based models. We use Alloy [9] to enumerate consistent design states, and our own algorithm to derive DA's, and, from DA's, DSM's. The DSM's that we computed revealed oversights in both our own and Lopes's DSM's, as presented in published papers. We also discovered a potentially serious oversight in Baldwin and Clark's model of the options value of modularity in design: it did not account for the *indirect* ripple effects of change as a component of the cost to exercise a module switching option. A notable feature of the DA is that the destination of any transition captures all "ripple effects," in that all of the changes needed to consistency in a given manner are present.

We believe that our theory has the potential to serve as a foundation for a broader range of applications based on augmented versions of the DA. In this position paper, we sketch more applications of this theory to decision-making problems that software designers care about.

4. APPLICATIONS

We categorize the problems that our framework intends to address into three classes: economic evolution analysis, DSM based analysis, and design comparative analysis. We formalize these problems and their solutions. Combined with economic models, the framework starts to bridge the technical and economic dimensions of a design.

4.1 Economic Evolution Analysis

In our paper [5], we have formally answered the following question: what are all the ways to compensate for a sequence of given decision changes? We model current design as a state in DA, envisioned changes as a sequence of variable-value binding pairs. The answer to this question is to find all the paths in the DA that start from the initial state and go along the edges labeled with specified changes, as shown in the following Z specification.

$$
\begin{array}{|l}
impact : (DA \times Assignment \times (\mathrm{seq}(Variable \times Value))) \\
\quad \nrightarrow (\mathbb{F}(\mathrm{seq}\, Assignment)) \\
\hline
\forall\, da : DA;\ design : Assignment; \\
\quad bindingSeq : \mathrm{seq}(Variable \times Value) \bullet \\
\quad impact(da, design, bindingSeq) = \{solseq : \mathrm{seq}\, Assignment \mid \\
\quad design = solseq(0) \wedge (\forall\, n : 1 \mathinner{..} \#solseq \bullet \\
\qquad solseq(n) \in da.transition(solseq(n-1), bindingSeq(n)))\} \\
\hline
\end{array}
$$

Combining this solution with cost models, we can further formalize and answer the following questions:

- Which is the least expensive of all the possible ways to accommodate a given *sequence of changes* in a design? We will attach a cost model to each change, compute and compare the cost of each evolution path, and output the one with the least cost.
- Given an expected sequence of changes, what is the best way to accommodate the first these changes, what is the best way to make the first change if the goal is to minimize the long-term cost of change? This is a classic decision problem in software engineering, and a problem that vexes practitioners on a daily basis. Based on the answer to the previous question, we just need to select the first step in the optimal path.

4.2 DSM Based Analysis

Since Simon is able to generate DSM's from declarative design models, we should be able to leverage existing DSM tools and methods to answer questions such as the following [18]:

- Are there cyclic dependences among design decisions? This is an important question, because dependence cycles must be broken for design processes to proceed. Where to break them is an interesting questions. Identifying cycles in the DSM is easy.
- What pieces of information are required to start a certain activity? and Where does the information generated by an activity feed into? These two questions are answered directly by the coupling structures represented by DSM's.
- What is the coordination cost implied by a design? Based on a selected cost model, the framework should be able to estimate the coordination cost implied by the coupling structure. This estimation is based on the assumption that the coupling structure of the design task is homomorphic to that of the design space. Another assumption is that tasks can be mapped to agents executing them [7]. Given different clustering methods, the tool should predict which clustering

method will incur the lowest coordination cost This is an ongoing joint work with another colleague.

- What is the net option value of modularity in a design, under the model devised by Baldwin and Clark (or an enhanced version thereof)? To answer this question, we need to associate a parameter called *technical potential* with each module or proto-module in a DSM-based representation. We have modeled proto-modules as sets of variables. To each such set we will have to associate and provide a parameter called *technical potential*. The other parameters of their model can be computed from the DSM.

4.3 Design Comparative Analysis

Our theory appears to have the potential to enable the formalization and analysis of important concepts and decision problems in software engineering. Returning to the refactoring question at the beginning of the paper, for example, we would analyze it as follows. First, we would develop augmented logic models of the current and proposed designs, at a suitably high level of abstraction. Second, we would formulate an expected evolutionary scenario as a sequence of changes, or perhaps as a stochastic process generating change requests. Third, we could measure the cost of change in both cases. Finally, we would account for the switching cost to get from the current to the proposed new design. This cost would involve the direct engineering costs, the opportunity costs incurred (in this case, delay in revenues due to the temporary suspension of feature development), and the value of the option to delay refactoring (in case of uncertainty about any of the key factors).

5. FUTURE WORK

Accordingly, our future work involves three aspects.

The greatest difficulty with our approach, from the perspective of operationalizing it for real modeling and analysis, is that it demands a complete enumeration of feasible design states. We have modeled a large system (that of Lopes), albeit with a small model. For a large, loosely constrained model exhaustive enumeration is infeasible. DSM computation even for small (KWIC-sized) models can take from minutes to several hours.

We have not tried to optimize our approach in any way, preferring to develop the theory, and formalizations of decision problems, aware of performance as a separate concern but one to be addressed later if the modeling approach continues to prove its potential value. There are many techniques for addressing scalability that we could explore, including abstraction, decomposition, and lazy evaluation. For now, our focus is on developing the theory, automating it, performing small scale case studies, and seeking a few more realistic cases to study.

6. REFERENCES

[1] C. W. Alexander. *Notes on the Synthesis of Form.* Harvard University Press, 1970.

[2] W. Ashby. *Design for a Brain.* John Wiley and Sons, 1952.

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

[4] F. Brooks. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19, Apr. 1987.

[5] Y. Cai and K. Sullivan. A value-oriented theory of structure in design, viewed as decision-making activity. In *Submitted for publication to ESEC/FSE 05*, 2005.

[6] E. W. Dijkstra. On the role of scientific thought. In *Selected Writings on Computing: A Personal Perspective*, pages 60–66. Springer-Verlag, 1982.

[7] S. D. Eppinger. Model-based approaches to managing concurrent engineering. *Journal of Engineering Design*, 2(4):283–290, 1991.

[8] J. M. Favor, K. R. Favor, and P. F. Favor. Value based software reuse investment. In *Annals of Software Engineering 5*, pages 5–52, 15May 1998.

[9] D. Jackson. Micromodels of software: Lightweight modeling and analysis with alloy. Feb. 2002.

[10] C. V. Lopes and S. K. Bajracharya. An analysis of modularity in aspect oriented design. 2005.

[11] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–8, Dec. 1972.

[12] H. A. Simon. *The Sciences of the Artificial.* The MIT Press, third edition, 1996.

[13] M. Spivey. The fuzz manual. URL: `http://spivey.oriel.ox.ac.uk/~mike/fuzz/`.

[14] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Systems Journal*, 13(2):115–39, 1974.

[15] K. Sullivan, Y. Cai, B. Hallen, and W. G. Griswold. The structure and value of modularity in software design. *SIGSOFT Software Engineering Notes*, 26(5):99–108, Sept. 2001.

[16] K. J. Sullivan, I. J. Kalet, and D. Notkin. Software design: The options approach. In *2nd International Software Architecture Workshop, Joint Proceedings of the SIGSOFT '96 Workshops, San Francisco, CA, October, 1996.*, pages 15–18, Aug. 1996.

[17] K. J. Sullivan and J. C. Knight. Building programs from massive components. In *Proceedings of the 21st Annual Software Engineering Workshop*, Greenbelt, MD, 4-5 Dec. 1996. IEEE.

[18] A. A. Yassine. An introduction to modeling and analyzing complex product development processes using the design structure matrix (dsm) method. 2004.