

# On Using Net Options Value as a Value Based Design Framework

Sushil Krishna Bajracharya, Trung Chi Ngo, and Cristina Videira Lopes

Department of Informatics  
Donald Bren School of Information and Computer Sciences  
University of California, Irvine  
{sbajrach, trungcn, lopes}@ics.uci.edu

## Abstract

*We present our experiences using Net Options Value (NOV) as a framework for evaluating options in software design. We start with a brief introduction of the NOV model, provide summaries of analyses - where and how NOV has been applied, and list some of the open issues based on the early works in this area. In particular we present our approach to understand an important parameter of the NOV called the 'technical potential' of a module and outline our further research directions in using NOV.*

[10] used NOV in the analysis of designs for the KWIC program proposed by Dave Parnas in [8], and, using results from NOV analysis, validated that an information hiding design is superior to a *protomodular* design. [6] demonstrated how NOV applies to real world applications by using a non-trivial example that showed that aspect-oriented modularization, in certain cases, adds value to an existing modular design. Recently, NOV has been used in the evaluation of various case studies of aspect-oriented designs whose results were consistent with those established in the literature. [5] In other words, NOV results confirmed the established notions of good and bad designs using Aspects.

## 1. Introduction

Net Options Value (NOV) is a model for evaluating modular design structures. It is based on the economic theory of real options and was first introduced by Baldwin and Clark in [1]. The model takes design as an experiment or investment activity whose outcome is unknown. In terms of returns, it might be profitable (better design) or might incur loss (worse design). NOV models such uncertainty by assuming the expected value of a module to be a random variable with a normal distribution. It accounts for the cost involved in making design changes based on the complexity of the modules and the dependencies between the modules. The result from NOV analysis is a real valued number that is taken as an indicator of the value for a given design.

## 2. Applications of NOV

It has been demonstrated that NOV is capable of evaluating modularity in software design at varying degrees of granularity: at the level of a small application [10], moderately sized application using Aspects [6] and at a much grander industrial scale [1].

## 3. Open Issues

Despite these attempts and efforts that support the usability of NOV and its capability to assess design value, irrespective of granularity and implementation strategy, there are many open issues in adopting this model as a valid framework to be used in practice.

Some of the immediate questions that emanate out from these early works done with NOV are as follows:

1. Can we safely generalize the assumptions in NOV to be true for software, across domains, and other dimensions along which software design may vary?
2. How do we associate the numeric value from NOV with meaningful attributes like effort or other related estimates?
3. Is there a valid mapping between conventional software metrics/measures and the parameters in NOV?
4. How do we incorporate the differences between various forms of modular dependencies? In doing a fine grained analysis we come across with questions like "Does inheritance have the same dependency *weight* as a method call?"

5. Many implicit dependencies exist between modules, for example, social dependencies [2]. How do we incorporate such non-technical dependencies in NOV?

We need a deeper understanding of the parameters in NOV, especially the link between economic and software properties, before we are in a position to answer open questions like these. Thus empirical validation of the assumptions we make about NOV and its parameters has a great potential in answering these questions.

We argue that any further research in better understanding NOV will broadly fall into these two categories: (i) Rich representation of dependencies, and (ii) Mapping parameters with more accurately measurable software attributes. A detailed discussion of these is out of scope of this paper. Therefore, we focus on our current approach to understand a more elusive parameter in NOV called the *technical potential* of a module (denoted by  $\sigma$ ).

### 3.1 On Measuring the Technical Potential of a Module

Mathematically  $\sigma$  is the standard deviation coefficient in the following expression for NOV.  $\sigma_i n_i^{1/2} Q(k_i)$  represents the expected value of the module and is a random variable with normal distribution. *Normal* distribution is a convenient assumption, made by Baldwin and Clark in [1], that needs empirical validation. The true nature of this underlying distribution is still an open issue.

$$NOV_i = \max_{k_i} \{ \sigma_i n_i^{1/2} Q(k_i) - C_i(n_i)k_i - Z_i \} \quad (1)$$

Here,  $k$  is the number of experiments,  $n_i$  is the complexity of the  $i_{th}$  module,  $C_i$  is the redesign cost and  $Z_i$  is the visibility cost.  $Q(k)$  is an order statistic that models the value of best  $k$  designs. Further details about the model and the parameters are given in [6, 10, 1].

The NOV model treats design as an investment activity or, precisely, a *value seeking process* [1], and above expression has the following intuitive meaning.

$$\text{Net Options Value} = \text{Expected Benefit} - \text{Redesign Cost} - \text{Visibility Cost}$$

We can directly infer only two mathematical properties of  $\sigma$  from the NOV model: (i) Since  $\sigma$  is a standard deviation coefficient in above equation, it will always be positive, and (ii) Based on the assumption that one experiment in a unimodular design breaks even [1], we get a value of 2.5 for  $\sigma$  of a single module system when the expected benefit and the redesign cost is assumed to be 1. (This is an assumption, made for illustrative purposes.) These two properties are not sufficient enough to make any general arguments about the

exact *relations* (empirical/numerical) that hold for  $\sigma$ . The intuitive meaning of  $\sigma$  further helps in extending our understanding about it.

The intuitive meaning of  $\sigma$  for software was proposed in [10] where they provide the following argument.

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.

[10] associates  $\sigma$  with *the variation in returns* for the investment made in design. This also supports Baldwin and Clark's theory of seeing  $\sigma$  as the indicator of *technical risks* inherent in a module. Furthermore, [10] argued that  $\sigma$  depends on the environmental factors such as *computer, corpus* and *user*. This notion of *environment parameters* (factors) provided valuable insights to us in formulating *external parameters* in [6]. We represented a collection of web-services as external parameters in our example. We further classified the modules into three categories and assigned the maximum weight for  $\sigma$  if the module falls under the category that provided direct service to the end-users. In this way, we captured the essence of *environment parameters* in our analysis.

These observations, both mathematical and intuitive, for  $\sigma$  make us believe that the right way to assess any measurement for  $\sigma$  is to study the evolution of existing modules. If we follow the interpretation of  $\sigma$  in economic terms, we would need to look into the financial history of the modules in capital markets. We believe that the history of software components in capital markets is not the sole indicator of  $\sigma$ , and, it is possible to look in the change patterns of modules throughout their lifetime in the light of several other environment parameters like users, external services, socio-technical implications etc. Thus, we intend to formulate indirect measurements of  $\sigma$  in terms of the environment parameters that are specific to the particular examples we choose to collect historical information. Generalizing these results would be impossible unless a broad spectrum of examples is covered.

We list the following set of tasks as the fundamental problems to be solved in this regard:

- Developing a representational model for measuring  $\sigma$  based on: (i) its mathematical significance in the NOV framework, and (ii) the general understanding and known heuristics about it.
- Developing a measurement function for  $\sigma$  and figuring out the historical data/metrics to collect based on the model.

- Verifying our measurement function for  $\sigma$  based on the data from real world projects.

We believe an attempt to formulate *the* standard model for  $\sigma$  is a challenging task. We intend to start out by formulating a simple model that would suffice for the validation of the current understanding of  $\sigma$  within the scope of examples as seen in [10] and [6].

## 4. Method and Approach

We intend to conduct empirical studies on the evolution of existing software modules to investigate the relationship between  $\sigma$  and environmental parameters. Specifically, we intend to extract information related to: (i) *Change patterns*: in the module for a given period of time, (ii) *Modules*: Depending on the size and implementation of the projects, we'll pick appropriate constructs as *modules*, and (iii) *Environmental Parameters*: We will consider external factors that affect the evolution of a module. Environmental parameters seem to be contextualized, meaning, they will differ according to the design goals and type of a system.

A major challenge in this approach is that it is difficult to find software projects whose evolution is explicitly documented. The history of a software module usually manifests itself as indirect evidences scattered in different development artifacts, including the project's documentation, repositories of source code, defect tracking databases, and mailing list archives. Thus, as part of the project we need to devise techniques and heuristics for collecting meaningful metrics from such historic databases. While we are still in early stages of developing such mechanisms, the findings of previous work, e.g. [4, 9], on mining historic databases show that this research direction is plausible. We plan to look at open source projects as they have been identified as valuable sources for empirical studies of software evolution [7].

### 4.1 Measurement and Validation Techniques

Prerequisites for validating software measurement are well established in literature. [3] However, before proceeding with the validation we need to formulate a correct model for  $\sigma$  considering the fundamental principles of software measurement. At this point very little is known about the nature of  $\sigma$ , for example, the kind of relationships that exist between modules based on the value of  $\sigma$  cannot be established universally. We believe, building on our current understanding about  $\sigma$  as outlined in this paper, we will be able to develop the required prerequisites for validating current heuristics and understanding about  $\sigma$ .

## 5. Conclusions

We have presented our early experience in using NOV in analyzing design options in software and pointed out some of the important research directions we are pursuing in this area. We have mostly focused on issues with the *technical potential* of a module. There are many open issues besides this. In particular, we have not talked about the various *modular operators* [1] in NOV. Some of the issues regarding these operators are pointed out in [6].

As an analytical framework, we find NOV very promising as it ties together modular dependencies, uncertainty, and economic theory in a cohesive model. We foresee further research effort in this field would add significant contributions in the field of value based design and software engineering at large.

## References

- [1] C. Y. Baldwin and K. B. Clark. *Design Rules vol I, The Power of Modularity*. MIT Press, 2000.
- [2] de Souza Cleidson, P. Dourish, D. Redmiles, S. Quirk, and E. Trainer. From technical dependencies to social dependencies. In *CSCW'04 Workshop on Social Networks, Chicago, IL, USA*, November 6-10 2004.
- [3] N. Fenton. *Software metrics (1st ed.): A rigorous approach*. Chapman and Hall, 1991.
- [4] D. M. German and A. Mockus. Automating the measurement of open source projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, May 2003.
- [5] C. V. Lopes. On the nature of aspects: Principles of aspect-oriented design. In *ACM Transactions of Software Engineering*. Under Review.
- [6] C. V. Lopes and S. K. Bajracharya. An analysis of modularity in aspect oriented design. In *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*, pages 15–26, New York, NY, USA, 2005. ACM Press.
- [7] A. Mockus and L. G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of International Conference on Software Maintenance, San Jose, CA USA*, pages 120–130, 2000.
- [8] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.
- [9] G. Succi, J. Paulson, and A. Eberlien. Preliminary results from an experimental study on the growth of open source and commercial software products. In *Third International Workshop on Economics-Driven Software Engineering Research (EDSER 03), Toronto, Canada*, May 2001.
- [10] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen. The structure and value of modularity in software design. In *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 99–108. ACM Press, 2001.