Measuring the *Technical Potential* of a Software Module
ICS 225 Project Proposal

Winter 2005

Sushil Bajracharya and Trung Ngo

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

Abstract

This document serves as our proposal for ICS 225 (Software Proceses) project . We provide some background information, define the problem we intend to solve and the methodology we plan to follow for the project. We also provide an annotated bibliography at the end of this document.

1. Introduction

Net Options Value (NOV) is a model for evaluating modular design strucutres based on the economic theory of real options. The basic idea behind the model is to take design as an experiment or investment activity. The outcome of this investment is unknown. In terms of returns, it might be profitable (better design) or might incur loss (worse design). NOV takes into account such uncertainity (modeled by a probability distribution of the design value) and the existing structure of design to simulate possible experiments along the distribution curve. The results boil down to real numbers to be taken as indicator of the values for designs.

It has been demonstrated that this model is capable of evaluating modularity in software design at varying degrees of granularity: at the level of small application [19], moderately sized application using Aspects [13] and at a much grander industrial scale [2].

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

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

1. How do the assumptions or hypothesis from the economic theory carries over to Software. For example NOV assumes the outcome of design/experiments to have normal distrubtion. Can we safely generalize this assumption to be true for software across several domains and dimensions along which software design may vary.

2. How do we interpret the term *experiment* in software design?

3. How do we interpret the result from NOV ? It gives us a numeric value for each design we are comparing, but, how do we associate such numbers with meaningful attributes lke effort or other related estimates ?

4. The structure of the software (such as the dependency structure of modules and levels of abstractions) and conventional metrics needs or can be mapped into the parameters that NOV defines. What is a valid mapping of such properties?

These all are the questions that we believe needs considerable research effort. We see empirical validation of some of the assumptions having a great potential in answering the kinds of the questions we have stated above.

In our project (for ICS 225), we intend to focus on validating the understanding about a parameter called the *Technical Potential* (denoted by σ) of a module that is an important parameter to be calculated for using NOV in evaluating design options.

2. Problem Statement

We list the following set of tasks as the fundamenal problems to be solved in this project:

- Developing a model for σ based on: (i) its mathematical significance in the NOV framework, and (ii) the general understanding and known heurisitcs for measuring σ.
- Developing a measurement function for σ and figuring out the metrics to collect based on the model.
- Collecting required statistics for verifying our measurement function for σ based on the data from a real world project.

2.1 Model for σ (Technical Potential) of a module

We believe an attempt to formulate *the* standard model for σ is a challenging task and not feasible to be accomplished in its entirety in a small project like this. However, we intend to formulate a simple model for σ that would suffice the validation of the current understanding of σ within the scope of [19] and [13]. We will look into its mathematical significance in the NOV model and its intuitive significance as established in the related literature.

Mathematically σ is the standard deviation coefficient in the following expression for NOV.

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

The NOV model, treats design as an investment activity or precisely a *value seeking process* [2], and above expression basically has the following intuitive meaning.

> Net Options Value of the i_{th} module = Benefit - Redesign Cost - Visibility Cost

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. Further details about the model and the parameters are given in [13], [19] and [2].

The intuitive meaning for σ for software was proposed in [19] where they provide the following argument for σ.

> 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 EPs [1] relevant to the module.

[1]EP meaning Environment Parameters

Finally [19] associates σ with *risks, in the sense of variation in returns*.

These observations, both mathematical and intuitive, for σ makes us believe that the only way to assess any measurement for σ is to study the evolution of existing modules. If we follow the interpretation of σ in economic terms, we would need to look into the financial history of the module in capital markets. We believe that the history of software components in capital markets are not the sole indicators of σ, 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 etc.

Thus we intend to formulate the indirect measurement of σ in terms of those environment parameters that seem feasible to be studied within the scope of our project. We believe it will also depend on the particular example we choose to collect statistical information from.

3. Method and Approach

We will look into the CVS repository of a moderately sized open source project that has been under development for a considerable period of time (at least few years) for the following:

- Categorize modules and select some key modules (at least five) from the system.
- Determine the environmental parameters pertaining to those modules.
- Extract change information about those modules and data about how the environmental parameters affected the modules during their evolution.
- Validate our hypothesis about σ on the information we extract.
- We plan to use a tool like [15] to collect the information we require or use existing collected data if we could find any.

Here are some early intuition about the parameters we are going to collect and analyse:

1. *Change Information:* Look into the change pattern in the module for a given period of time. We intend to look into the frequency of commits that were made on a module during a certain period of time. (A possible graph might be number of changes made per module each week for one year)
2. *Modules:* Depending on the size and implemented language of the project we choose, we'll pick an appropriate construct as a *module*. For Java programs, we would consider individual classes as modules. Another alternative is considering packages instead of classes.

3. *Environmental Parameters* were introduced in [19] to model the environmental conditions. σ is assumed to be dependent on the environment. Factors like *computer*, *corpus* and *user* were taken as the representative of environmental conditions.

   [13] presented the notion of *external parameters*, a collection of web-services, as an instance of environmental parameters in their example. They further classified the modules into three categories and assigned the maximum weight for σ if the module falled under the category that provided direct service to the end-users. In this way they captured the essence of *environement parameters* in their analysis. We intend to investigate these assumptions about σ.

   For this, we will categorize the modules based on, the services they provide to the end users, and, how they are dependent on the environment. We will model the environment with external libraries and APIs the module (under study) depends on.

3.1 Validation Techniques

Fenton talks about the necessary prerequisites for vaidating software measurement in [7]. Chapter 6 in [7] specially lists some of the data collection and analysis methods. Before proceeding with the validation we need to focus on formulating the model for σ, considering the fundamental aspects of measurement as suggested in [7], Chapter 2 and 4. At this point we are finding this task more challenging given the subtlety inherent in σ. However, we believe, building on our current understanding about σ as oulined in section 2.1 we will be able to develop the required prerequisites for the validation.

4. Conclusions

We do not claim to develop a solid understanding about the nature of σ in this project. Our main objective will be to refine our intuition about σ based on its mathematical significance and properties that have been attributed to it in earlier works. In particular, we seek to validate the heuristics given for calcualting σ in [13].

The annotated bibliography contains summaries for additional related materials we have referred to as of this date. We feel that the information we need to complete this project is more of fundamenal in nature. In other words, most of the papers we referred to discuss a particular model and the metrics they use for the model. What is important to us is the technique and the reasons the authors had in formulating those models and most of the referenes lack a thorough treatment of such issues. One of the reasons might be that the techniques we are seeking for are fairly standard and well understood. Comprehensive treatments of such statistical and empirical techniques seem to be readily available (For eg; [17]). It seems, our primary resource for this project for such techniques will be [7], as it deals with the requirements for valid measurement and assessment of software properties.

We are still assimilating the materials from the various references and still looking for other relevant works. However, at this point we already have some insights about what do we need to look for to complete this project.

References

[1] John Rusnak Alan MacCormack and Carliss Baldwin. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *published online, http://www.people.hbs.edu/cbaldwin/DR2/ MRBDesignStructure17thSep1.pdf.*

> This paper uses Design Structure Matrices (a tool which is also used in NOV analysis) and metrics based on these structures to develop metrics to evaluate the modularity of design of publicly available projects, the Linux Kernel and Mozilla. It shows that evaluating design by extracting information from source repositories is possible. However the work that has been done here is slightly different as we are seeking to validate our measurement function whereas this paper uses their metric to evaluate the design.

[2] Carliss Y. Baldwin and Kim B. Clark. *Design Rules vol I, The Power of Modularity*. MIT Press, 2000.

> This is the definite reference for the Net Options Value (NOV) model for analyzing designs. Baldwin and Clark build the theory and the analytical framework for NOV based on: (i) the economic theory of Real Options, (ii) seminal philosophical foundations for design, and (iii) historical results from computer industry, mainly IBM/360.

[3] Barry Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

> In this book, Barry Boehm introduced the Constructive Cost Model (COCOMO) for software project and plannning. The comprehensive treatment of COCOMO and usage of conventional metrics and economic theory in constructing the model seems to be relevant to our project. However, we

need to delve deeper in the book for better understanding the mechanics behind COCOMO.

[4] Barry W. Boehm, Bradford Clark, Ellis Horowitz, J. Christopher Westland, Raymond J. Madachy, and Richard W. Selby. Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering*, 1:57–94, 1995.

This paper introduces COCOMO 2.0 as a successor to COCOMO 81. COCOMO 2.0 is a new software cost estimation model aimed at software development practices of the 1990's and 2000's. It provides a tailorable set of different size-related metrics and new adjustment factors for dealing with software reuse, re-engineering, conversion, and maintenance. COCOMO 2.0 is a quantitative framework that takes into account several metrics such as; Object Points, Function Points, and Source Lines of Code; for making predictions about the effort required. A deeper understanding of how such metrics fits in COCOMO can give some insights on how metrics can be used in quantitative models.

[5] Lionel Briand, Sandro Morasca, and Victor R. Basili. Defining and validating high-level design metrics. Technical report, 1994.

Most software measurement approaches focus on capturing characteristics of software code. However, source code is usually avaiable at later phases of the software development life cycle. This paper focuses on exploring high-level design metrics for software systems. The hypothesis of this approach is that early availability of software metrics enable 1) early detection of problems, 2) better software quality monitoring, and 3) more accurate planning. Two ratio-based design metrics are proposed in the paper for measuring the degree of cohesion and coupling of software components. These metrics are experimented and empirically validated on three NASA projects.

[6] Serge Demeyer, Stephane Ducasse, and Oscar Nierstrasz. Finding refactorings via change metrics. In *OOPSLA '00: Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 166–177. ACM Press, 2000.

Refactoring is a common programming technique for enhancing the internal structure of a program without affecting its external behavior. Refactoring involves making incremental changes to the structure of code to achieve particular design goals. The main key contribution that this paper made is to propose techniques for identifying refactorings in the development history of a software system. Specifically, applying software metrics to successive versions of a software system and measuring changes helps program understanding and design recovery. Techniques proposed in this paper can help in detecting refactorings in the development history of a software module. Our early intuition makes us believe that there is a strong relationship between environmental parameters and this type of design change.

[7] Norman Fenton. *Software metrics (1st ed.): A rigorous approach*. Chapman and Hall, 1991.

This book is one of the major references for our project. It provides a critical analysis of many proposed metrics in software that span wide range of measurements. These measurements are categorized into metrics that measure internal attrubutes, external attribures and resources like productivity. More important for our project are the earlier six chapters that details fundamentals of software measurement, data collection/analysis and software measurement validation. (Chapters 2 -6)

[8] Norman Fenton and Shari Lawrence Pfleeger. *Software metrics (2nd ed.): a rigorous and practical approach*. PWS Publishing Co., 1997.

This is the second edition of Fenton's popular book on metrics.

[9] Norman E. Fenton and Martin Neil. Software metrics: roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 357–370. ACM Press, 2000.

This paper presents a prediction model based on Bayesian Networks. Fenton argues why conventional metrics are not good enough in making reliable predictions about software properties and how a causal model based on bayesian net (BBN) can help make more accurate and predictions predictions. It claims that metrics are required but should be used with care in a more reliable model like BBN.

[10] Harald Gall, Mehdi Jazayeri, and Jacek Krajewski. Detection of logical coupling based on product release history. In *Proceedings. International Conference on Software Maintenance, 1998*, pages 190–198. IEEE, 1998.

This paper presents the CEASAR approach for identifying change patterns among modules based on their development history. This approach consists of two processes *Change Sequence Analysis* (CSA) for identifying common patterns of changes and *Change Report Analysis* (CRA) for validating the patterns found in the CSA process. This paper is very relevant to our project as it demonstrates the use of information on changes from CVS and to reveal the hidden logical coupling among modules of large software projects.

[11] Harald Gall, Mehdi Jazayeri, and Jacek Krajewski. Cvs release history data for detecting logical couplings. In *Proceedings. Sixth International Workshop on Principles of Software Evolution, 2003*, pages 13–23. IEEE, 2003.

QCR is a software evolution analysis technique consisting of three complementary techniques: 1) *Quantitative Analysis* for observing change and growth rates, 2) *Change Sequence Analysis* (CSA) for revealing common change patterns, and *Relation Analysis* for recovering dependencies by change-related attributes. This paper focuses the discussion of the RA technique, which was introduced to complement the other two (published in ’97 and ’98). The RA technique deals with extracting neccessary information from CVS about changes made to single modules during their history of development. These evidences of evolution are used to compare and reveal logical dependencies between modules (modules were most frequently changed together).

[12] Chris F. Kemerer and Sandra Slaughter. An empirical approach to studying software evolution. *Software, IEEE*, volume: 25, Issue: 4:493–509, 1999.

In this paper, the authors provide a set of methods and techniques for designing empirical study of software evolution. As we plan to look at the development history of real projects for verifying the correctness and usefulness of our new set of metrics, this paper is definitely a valuable source for us.

[13] Cristina Lopes and Sushil Krishna Bajracharya. An analysis of modularity in aspect oriented design. In *AOSD 05*. ACM Press, to appear, 2005.

This paper applies NOV to evaluate several design changes in a web-based system from its creation (by resusing a freely available system) to its completion where two of its features: *logging* and *authentication* are modularized using Aspects. This paper demonstrates that NOV is applicable in the design of of non-trivial real world application and is able to capture modularization using novel techniques like Aspects. A key highlight of this paper is it presents an early intuition about evaluating technical potential of a module and also refines the notion of *environment parameter* introduced by Sullivan et.al

[14] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.

Dave Parnas introduced the concept of Information Hiding in this paper and presented his idea using the Key Words In Context (KWIC) example. He argued that software systems should be decomposed into modules considering many changeability factors while facilitating independent development of modules and comprehensibility of the software produced. His introduced *Information Hiding* as the criteria to decompose system into modules unlike the then conventional way of decomposition using program flow or flowcharts.

[15] Gregorio Robles, Stefan Koch, and Jesus M. Gonzalez-Barahona. Remote analysis and measurement of libre software systems by means of the cvsanaly tool. In *2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS ’04). 26th International Conference on Software Engineering (Edinburgh, Scotland)*, May 2004.

This paper explores ways to remotely measure and analyze software projects whose source is publicly available. CVSAnalY, the tool presented in this paper, has the capability of accessing and extracting metrics from CVS (Concurrent Versions System) source code repository. The tool mentioned in this paper seems to generate change information about each CVS module. For our project we need a more fine-grained information, especially change information per progaram units.

[16] H. Dieter Rombach. Design measurement: some lessons learned. *Software, IEEE*, volume: 7, Issue: 2:17–25, 1990.

In this paper Rombach presented a design-measurement framework based on the experience he gained from measuring design aspects of some large-scaled software projects. He suggested that designers should choose and tailor an effective measurement approach such as the Goal/Question/Metrics (GQM) for measuring certain design aspects of interest. He argued a successful design measurement method should consider a variety of design measures, including both *abstract* and *specific* measures, *process* and *product* measures, *direct* and *indirect* measures, and *objective* and *subjective* measures.

[17] Richard J. Shavelson. *Statistical Reasoning for Behavioral Sciences*. Allyn and Bacon, inc., 1988.

This book is an extensive treatment of statistical methods for doing empirical research. It particularly focus on dseigning and drawing statistical inferences from experiments in behavioral sciences.

[18] Mary Shaw and David Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., 1996.

This book is one of the earliest and comprehensive treatment of Software Architecture. In the context of our project, this book presents several architectural designs for the KWIC program that was first proposed by Parnas and further used by Sullivan et.al as an example to demonstrate the use of NOV in evaluating design. However, we believe that we cannot use KWIC example and the criteria to evaluate design for KWIC implementations given in this book since the data we need to evaluate the measures of technical potential has to deal with the evolution of design or modules and we donot have any such data available for KWIC.

[19] Kevin J. Sullivan, William G. Griswold, Yuanfang Cai, and Ben 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.

This seems to be the first paper that treats NOV in the context of software. It uses the KWIC program proposed by Dave Parnas as an example to demonstrate the usage of NOV in evaluating design. A key contribution this paper makes is the notion of Environment Parameters as the indicators of Technical Potential of a module in a given design.

[20] Rini van Soligen and Egon Berghout. *The Goal/Question/Metric method*. McGraw-Hill, 1999.

The Goal/Question/Metric (GQM) model was first introduced by Victor Basili and David Weiss in the 80's. This approach suggested that software measurement must be defined in a top-down, goal-based fashion. A GQM-based measurement program is a hierarchy structure starting with some particular goals (objectives of measurement). The goals are refined into set of questions, and then each question is refined to a set of metrics that need to be collected. Our method of devising the the set of metrics for measuring components' technical values also fall under the GQM spectrum. In particular the goal driven approach is consistent with the notion of associating design values based on design goals that was demonstrated in one of the papers on using NOV by Lopes and Bajracharya.