Scaling New Heights

EDSER-5
5th International Workshop on Economic-Driven Software Engineering Research

ICSE'03

International Conference on Software Engineering

Portland, Oregon

May 3-11, 2003

EDSER-5 Proceedings

5th International Workshop on Economic-Driven Software Engineering Research

Table of Contents

Educational issues in economics driven software engineering

CourseForges: Open Source Curriculum Design for Value-Based Software Engineering *1*
Mary Shaw, Shawn Butler, Hakan Erdogmus and Klaus Schmid

Evaluating the value of software

Reasoning About the Value of Dependability: The iDave Model *5*
Barry Boehm, LiGuo Huang, and Apurva Jain

Use of Real Options Theory to Value Software Trade Secrets *9*
Donald J. Reifer

Aiming for values in software

Intentional Software Systems: Engineering Intentionality in Processes of Software Development and Runtime Execution *13*
Kevin Sullivan

Time is Not Money: the case for multi-dimensional accounting in value-based software engineering *17*
Vahe Poladian and Shawn Butler, Mary Shaw and David Garlan

Testing

About the Return on Investment of Test-Driven Development *23*
Matthias M. Müller and Frank Padberg

Selecting a defect prediction model for maintenance resource planning and software insurance *29*
Paul Luo Li, Mary Shaw and Jim Herbsleb

Architectural economics

ArchOptions: A Real Options-Based Model for Predicting the Stability of Software Architectures *35*
Rami Bahsoon and Wolfgang Emmerich

Understanding the Economics of Refactoring *41*
Eleni Stroulia and Rob Leitch

COTS

Composable Process Elements for Developing COTS-Based Applications *47*
Ye Yang, Jesal Bhuta, Barry Boehm, Dan Port and Chris Abts

WinWin Spiral Approach to Developing COTS-Based Applications *54*
Barry Boehm, Dan Port and Ye Yang

Risky business

Economic Risk-Based Management in Software Engineering: The Hermes Initiative 60
Stefan Biffl, Michael Halling and Paul Grünbacher

Software Dependability Risks and the Insurance Process... *66*
Dan Port and LiGuo Huang

Using Risk to Balance Agility and Discipline.. *70*
Barry Boehm

CourseForges

Open Source Curriculum Design for Value-Based Software Engineering

Mary Shaw,
School of Computer Science
Carnegie Mellon University
+1-412-268-2589 <etc>
http://www.cs.cmu.edu/~shaw
mary.shaw@cs.cmu.edu

Shawn Butler
School of Computer Science
Carnegie Mellon University
+1-412-268-[[[?]]]
http://www.cs.cmu.edu[[[?]]]
shawn.butler@cs.cmu.edu

Hakan Erdogmus
Institute for Information Technology
National Research Council
+1-613-991-1018
http://lit-iti.nrc-cnrc.gc.ca
hakan.erdogmus@nrc.cnrc.gc.ca

Klaus Schmid
Fraunhofer IESE
+49-6301-707-158
http://www.iese.fhg.de/Staff/schmid
klaus.schmid@iese.fhg.de

ABSTRACT

As a relatively young discipline within software engineering, value-based software engineering does not yet have an established curriculum. The area draws on models and techniques in so many other disciplines that it is likely to be some time before a single individual is ready to prepare a course or a textbook. Several of the EDSER-4 participants expressed interest and enthusiasm for sharing the effort of developing curriculum and course materials. Inspired by the success of open source software development, especially the distributed collaboration, the free public access to the results, and the lack of administrative overhead; we decided to try to establish a similar community for curriculum development. This report describes progress to date, with emphasis on the community standards for cooperation and sharing.

Keywords

Value-based software engineering education, cooperative curriculum development, open source curriculum development, value-based body of knowledge.

1. BACKGROUND

Course Forges was initiated at the EDSER 4, the 2002 Workshop on Economics Driven Software Engineering Research. Many EDSER participants want to add cost considerations to our software courses, but we don't see near-term prospects for a unified textbook. Further, we all have different expertise in the area. To complicate matters further, most of us don't have the opportunity to add a full course to our institutions' curricula.

We decided that we could help each other by sharing the effort, with different people designing teaching units covering from one lecture to a few weeks' content. Two other communities have reaped the benefits of collaboration, and we would like to build on their success.

- We decided that pride of ownership is much less important than quality, and we should develop the materials in the style of open source software.
- We recognize that consistent presentation helps the reader find information and also helps the author to cover the content consistently. The patterns community has developed useful expositions of software from this intuition, and we would like to do likewise for course content.

The Course Forges community has been established to share effort and benefit of curriculum development in software engineering. In the fullness of time, this may come to serve different curriculum areas. We begin with by focusing on value-based software engineering -- techniques that consider cost as well as benefit in making software design decisions.

Section 2 describes the shared principles -- the Community Values -- that guide this community -- the Course Forges Alliance. Section 3 introduces the web site where the collaborations are taking place. Section 4 describes a potential curriculum for EDSER; Section 5 invites further discussion and active participation.

In the longer term, we hope to find ways to share the presentation of this material to students, though the academic calendars of universities present formidable obstacles to doing this smoothly.

2. COMMUNITY VALUES

Members of a collaborative community expect to share effort and benefits. This page is the current draft of our shared values, principles and standards. It includes a declaration of principle, or shared intent and a discussion of rights and responsibilities of members of the community,

2.1 Overview

We agree in principle to adapt the open source software development model for our purposes. This implies

- Collaborative development
  - Shared development, without an ego stake in authorship
  - Shared documents in a common, relatively public place
- Community standards
  - Shared development effort, with recognition as the chief incentive
  - Intellectual property ground rules encouraging sharing, with public content on this site and the possibility of extending the work for profit in other venues
  - Consistent structure for curriculum units, in the style of the patterns community

2.2 Declaration of Shared Intent

- *Content:* We are jointly interested in developing a curriculum for value-based software decision making, that is, for software design and development in which the significance of cost is on a par with that of functionality.
- *Collaboration:* We would like to share the effort of developing and possibly of offering this material. To this end we need a collaborative environment for developing and distributing the material, so we adapt the open source software model to work for curriculum materials.

- *Adaptability:* Our specific institutional requirements will lead us to courses that differ in detail, so we see the greatest promise in creating a set of short curriculum components that can be combined in different ways. Given the relative youth of this area, it is especially important to have curriculum components that can be incorporated in established courses.
- *Format:* We believe that a lecture-based format is often inadequate for this material. Formats with greater student engagement, such as projects and case studies, are usually more appropriate. Such materials also have greater promise for asynchronous shared offerings.
- *Audience:* The principal target audience is advanced undergraduates and early masters students; some of the components may fit in sophomore-level software engineering courses as well.
- *Resources:* Whenever possible, we would like to rely on external resources, including case studies and open source development tools.

2.3 Rights and Responsibilities

Legal obligations, especially with respect to intellectual property, are expressed in the license terms. This draft of license terms is modeled on the Open Source Initiative's Open Source Definition for code:

- *Free redistribution:* Material may be redistributed, by anyone, including as part of a larger redistribution, in printed or electronic form. Fees may not be charged for this redistribution, other than reasonable reproduction costs.
- *Public originals:* If derived forms (e.g., object code of tools) are distributed, the original form (e.g., source code) must be easily available as well. The original form is the preferred form for further development.
- *Free evolution:* Modification and derivative works are encouraged. They must be redistributed in accordance with this license, without requirement for additional licenses. Redistribution may not discriminate against people, groups, or fields of endeavor.
- *Noncontamination:* If this material is distributed with other material, it may be separated from the package for further redistribution. However it does not "contaminate" the other material. (e.g., if incorporated in a textbook, the rest of the book can be copyrighted by the author, but not the incorporated material -- and the difference must be clear)

This license is very similar to the Creative Commons (http://www.creativecommons.org/) Attribution-Noncommercial License (http://creativecommons.org/licenses/by-nc/1.0)

We also recognize moral obligations. The academic community operates on credit and attribution, especially for promotions and other recognition. Showing influence on other institutions' curricula can be significant at some schools. Therefore we strongly encourage anyone who uses our materials to:

- *Acknowledge:* Record the use of material from a Forge at the obvious place in that Forge. Say what course and institution, what level and how many students, evaluation
- *Contribute:* Consider returning improvements to the Forge

3. THE COURSE FORGE

Just as the SourceForge web site provides a development environment for many independent open source software development projects, we intend the CourseForges site to provide a development environment for many independent cooperative curriculum developments. We begin with one Forge -- for value-based software engineering.

3.1 The Web Site

We began with a conventional web site with discussion areas for public comment and other pages edited by a few core people. This site, http://courseforges.org, contains some discussion of tooling requirements and community values.

We have recently decided that the ability for participants to update the site easily is, at least for now, more important than sophisticated structure or layout. Accordingly, active development, of the organization, of content outlines, and of individual curriculum units, is now taking place on a Wiki at http://seg.iit.nrc.ca/yawc/courseforges/public/wiki.cgi

While we are a small community, simple password protection is sufficient -- anyone who can edit anything can edit everything. We rely on good will, change logs, and the Wiki's built-in version management to keep things under control. This should suffice until the materials are adopted, or even considered for adoption, by people who are not developers. At that time we will need either more sophisticated security or open discussion groups, and we'll move to the open source model in which everyone can review materials, download content, and discuss changes -- but only credentialed people can actually make changes.

3.2 The CourseForges Alliance

The CourseForges Alliance is a group of software engineering researchers committed to collaborative development of curriculum material in the same spirit of sharing that is demonstrated on the Open Source Software development community.

Anybody can browse the pages or download the documents on this site. Members of the Course Forges Alliance may create new pages, upload files and images, and modify existing pages. Password protection enforces this restriction

The friends of the Alliance encourage and support the activity, and they may use the materials, but they are not actively involved.

4. COURSE CONTENTS

While the CourseForges Alliance is open for everyone to contribute, it is fundamentally grounded in the EDSER-community: a group of researchers who address software engineering predominantly from an economic point of view. Consequently, it is planned that the first full course material will be available for a lecture on the EDSER topics. Such a lecture does not yet exist, nor does there exist full agreement on what the topics that should be covered would be.

4.1 Pragmatic Considerations for CourseForges Course Development

When looking at the problem of world-wide course construction, we need to accept that due to differences in local situations, there will never be a single, generic set of course materials that can be arbitrarily used at each university. Rather, the local situation like

the available time for students, lectures the students previously took, the specific focus of the lecturer, and so forth, must be taken into account. This poses a need for adaptability when developing a course. Being software engineers, we recognize of course the similarity between this situation and the development of a reference architecture for a line of software products [1].

Thus, we need to define a customizable approach to developing such a course. The modules of the course must be scalable and, while consistently building on each other, we need to ensure mostly independence of the modules. One approach to achieve this could be to define a basic skeleton of topics that each course must cover (the basics of value-based software development). This base content could be supplemented by other parts that are by themselves not required by other parts. Thus, a module – respectively its content – should be tagged as either *required or optional*.

In order to enable the lecturers to tailor or replace modules according to their needs, and to reorder modules, we need to define what is it each module provides, i.e., the *objectives* of the module, as well as the specific *concepts and methods* it provides, so that other modules can build on this. In order to support the reordering and replacement, it is also important that each module makes known the specific *preconditions* it has on other modules.

4.2 Possible Content of an EDSER-Course

We are far from having a final definition of how an EDSER-course could look like. However, in order to provide focus to such a discussion, we provide here one specific curriculum proposal. In particular, we use this opportunity to point out the range of variability such a curriculum would still support.

A high-level structure of such a course could consist of the following four blocks:

- *Software Business:* This part focuses on providing the students with an understanding of the intrinsic relation between a product, its characteristics, and their relation to the market place. This would also provide the basics for financial and strategic analysis.
- *Value Models:* This part teaches the students the fundamentals of the different forms of value that are relevant in software development and provides them with the key concepts relevant to building value models of software engineering activities.
- *Decision Making:* A key part of value-based software engineering is the need to make decisions based on value tradeoffs. Thus, in this part decision making techniques are taught.
- *Applications:* A collection of examples should illustrate the main technologies taught in the course. These could be discussed either as a fourth part at the end of the course, or scattered throughout the lecture. For this reason, each example description should also describe the required student knowledge for its discussion.

While this schema could provide a common skeleton for all EDSER-courses, the individual instantiation would probably strongly vary in terms of the extent of their treatment of the various topics. We provide here a key list of some topics and point out some parts that could be left out (are treated superficially) in specific courses.

Software Business

The key topics in the software business part are the market-oriented viewpoint (TTM, cost, pricing, customer value, etc.) and the financial concepts (NPV, compound interest, etc.). In addition, this part could address technologies from marketing science in more detail (analysis of customer preferences) and the issue of business strategy (e.g., balanced scorecard approach).

Value Models

This would be the main part and probably also take the main time of such a course. The basis for any kind of model is to make certain aspects measurable, thus this part would start with the topic of metrics and measurement and would further discuss. Then different forms of models would be discussed along with model-building approaches. This point could also link back to the first part by discussing more complex topics from finance like options and decision tree analysis. Finally, aspects important to the development of models like simulation-based approaches and model validation would be discussed. A full list of topics could look like this (<opt> marking typical parts that could be left out):

- Metrics and measurement (types of metrics, GQM, etc.)
- Utility theory <opt>
- Metrics estimation (data elicitation/ gathering techniques) <opt>
- Model types (rule-based, quantitative, etc.) <opt>
- Model building approaches (regression models, CoCoMo, simulation models <opt>, DTA <opt>, QFD <opt>)
- Financial models (options, book-keeping methods) <opt>
- Model validation

Decision Making

Here, typical decision making techniques like the analytical hierarchical process (AHP) or other multi-attribute techniques would be discussed. Further, AI approaches to decision making (e.g., rule-based, case-based-reasoning, and so forth) could be part of this section.

Applications

The applications could of course be distributed across the various sections, wherever appropriate. However, in a specific applications section more voluminous topics could be addressed like the analytical methods for software design: the CBAM-approach, the approach by Sullivan et al. on the value of modularity, or constructive techniques like product line scoping or process optimization (e.g., Agile development). This section would mainly serve a better anchoring of the previous topics and would actually enable the students to understand the software engineering concepts they learned so far better.

5. INVITATION

For CourseForges to succeed, even within the value-based software engineering community, it needs to provide enough useful curriculum material that faculty find it worth the time to look for content there. If it's successful, it will attract other new authors. The challenge for us is to bootstrap the activity so that it has a chance of achieving critical mass.

So this is an invitation to participate in the refinement of community values, the structure of the Wiki, and the development

of curriculum materials. Contact any of the authors or visit the CourseForges Wiki for further information.

6. ACKNOWLEDGMENTS

The CourseForges Wiki is hosted by the National Research Council of Canada at http://seg.iit.nrc.ca/yawc/courseforges/public/wiki.cgi. The static web site http://courseforges.org is hosted by Carnegie Mellon University. This work is supported by the National Science Foundation under Grant ITR-0086003, by the Sloan Software Industry Center at Carnegie Mellon, by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298, by the Software Engineering Institute, under the Networked Systems Survivability Program., and in part by the Eureka 2023 Programme, ITEA project ip00004, Café

7. REFERENCES

[1] Dewayne E. Perry. Generic Architecture Descriptions for Product Lines. In Development and Evolution of Software Architectures for Product Families, Second International ESPRIT ARES Workshop, Las Palmas de Gran Canaria, Spain, February 1998. LNCS 1429, pp.51-56, Springer, 1998.

Reasoning About the Value of Dependability: The iDave Model

Barry Boehm
Department of Computer Science
University of Southern California
Los Angeles, CA 90089, USA
(213)-740-8163
boehm@sunset.usc.edu

LiGuo Huang
Department of Computer Science
University of Southern California
Los Angeles, CA 90089, USA
(213)-740-6505
liguohua@usc.edu

Apurva Jain
Department of Computer Science
University of Southern California
Los Angeles, CA 90089, USA
(213)-740-6505
apurvaja@usc.edu

ABSTRACT

In this paper, we present a framework for reasoning about the value of information processing dependability investments called the Information Dependability Attribute Value Enhancement (iDAVE) model. We describe the overall structure of iDAVE, and illustrate its use in determining the ROI of investments in dependability for a commercial order processing system. We conclude that dynamic and adaptive value-based dependability mechanisms such as iDAVE model will become increasingly important provided evidence that dependability attribute requirement levels tend to be more emergent than pre-specifiable.

General Terms

Measurement, Design, Economics.

Keywords

Value, Cost, Dependability , Return On Investment

1. INTRODUCTION

This paper presents a framework for reasoning about the value of information processing dependability investments called the Information Dependability Attribute Value Enhancement (iDAVE) model. It assumes that a project makes baseline investments in developing information processing capabilities that generate baseline flows of costs, benefits, and returns on investment (ROI). From this baseline, iDAVE provides ways to specify additional investments in enhancing dependability, and to determine the resulting additional costs and the resulting improvements in both dependability attribute levels and their ensuing system benefits. These can then be used to determine the ROI of the dependability investments.

The paper describes the overall structure of iDAVE, and illustrates its use in determining the ROI of investments in dependability for a commercial order processing system. It shows how the results not only provide a useful decision aid for dependability investments, but also provide deeper insights on the nature of dependability investment processes, and on the nature of information processing dependability analysis methods.

2. NATRUE AND STRUCTURE OF THE iDAVE MODEL

The overall form of the iDAVE model is shown in Figure 1. An initial set of cost estimating relationships (CER's) is provided by the COCOMO II model [3]. The COCOMO II CER's enable users to express time-phased information processing capabilities in terms of equivalent size, and to estimate time-phased investment costs in terms of size and the project's product, platform, people, and project attributes. Additional future CER's would include the COCOTS CER's for COTS-related software costs [1], inventory-based CER's for hardware components and COTS licenses, and activity–based CER's for associated investments in training and business process re-engineering.

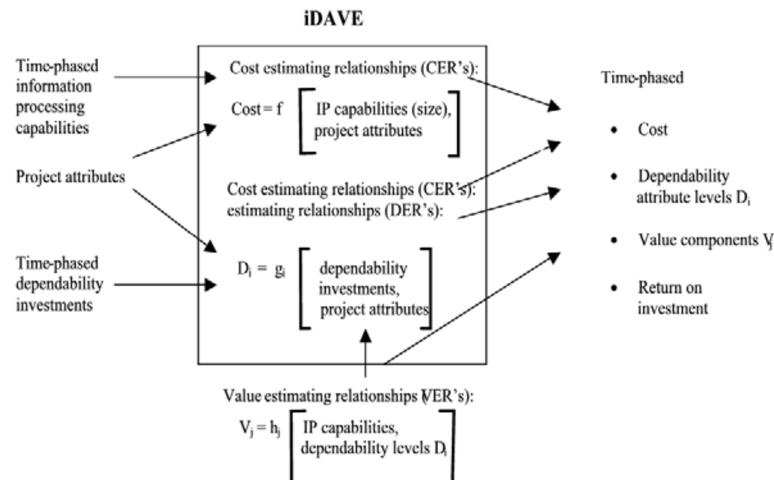


Figure 1. Proposed Information Dependability Attribute Value Enhancement (iDAVE) Model

An initial set of dependability attribute estimating relationships (DER's) is provided by the COQUALMO model [7]. It enables

users to specify time-phased levels of investment in improving dependability attributes, and to estimate the resulting time-phased dependability attribute levels. The current version of COQUALMO estimates delivered defect density in terms of a defect introduction model estimating the rates at which software requirements design, and code defects are introduced, and a subsequent defect removal model. The defect introduction rates are determined as a function of calibrated baseline rates modified by multipliers determined from the project's COCOMO II product, platform, people, and project attribute ratings. The defect removal model estimates the rates of defect removal as a function of the project's levels of investment in automated analysis tools, peer reviews, and execution testing and tools. Initial CER's are available to estimate the costs of these investments. Further COQUALMO extensions will refine its current DER's, and will provide further DER's for estimation of additional dependability attributes such as reliability, availability, and security [5].

The iDAVE model's initial dependability value estimating relationships (VER's) assume that a baseline business case analysis has been performed for various components of value (profit, customer satisfaction, on-time performance) as a function of the time-phased information processing capabilities at nominal dependability attribute levels. These value components are aggregated into an overall time-phased value stream, which is then composed with the time-phased costs (cost of IP capabilities plus dependability investments) and normalized using present-value formulas to produce a time-phased return on investment profile.

The initial iDAVE VER's involved simple relationships such as the operational cost savings per delivered defect avoided, or the loss in sales per pecent of the system downtime. Future extensions are planned to involved more detailed dependability VER's for defect severity distributions and reliability/availability levels, and additional VER's for such dependability attributes as security risk profiles and safety hazard profiles.

3. AN INITIAL iDAVE PROOF-OF-PRINCIPLE ANALYSIS: DEPENDABLE ORDER PROCESSING

The example below illustrates a simple initial use of iDAVE to develop a rough dependability return on investment analysis, using the Sierra Mountainbikes order processing system business case analysis in [2]. It uses this business case analysis as the baseline for assessing future investments in dependability over and above the nominal investments usually made for business data processing systems. Table 1 summarizes the business case for an improved order processing system through its proposed development in 2004-2005 and proposed operation in 2005-2008.

More specifically, the Initial Operational Capability (IOC) for the order processing system will start development on January 1, 2004. It will be installed for beta-testing with the three key distributors on September 30, 2004, and cut over as a replacement for most of the old system on December 31, 2004, at a cumulative investment cost of $4 million. An incremental release of the IOC responding to the most cost-effective fixes and enhancements will occur on March 31, 2005. Concurrently, work will start on the enhancements for the Full Operational Capability (FOC), which will also be beta-tested by the three key distributors, and then cut over as a full replacement for the old system on December 31, 2005, at a cumulative cost of $6 million. Thereafter, six-month increments and annual new releases will be installed at an annual investment level of 500K.

Table 1 shows the corresponding expected benefits and return on investment, ROI = (Benefits – Costs) / Costs, annually for the years 2004-2008. For simplicity in this analysis, the costs and benefits are shown in 2004 dollars to avoid the complications of discounted cash flow calculations, and the 10% annual growth rate in estimated market size is not compounded, both for simplicity and conservatism.

As seen in columns 2-5 of Table 1, Sierra's current market share and profit margins are estimated to stay roughly constant over the 2004-2008 period, with annual profits growing from $7M to $12M, if the new program is not executed. This is a conservative estimate, as the problems with the current system would increase with added sales volume, leading to decreased market share and profitability.

The next columns in Table 1 up through ROI show the expected improvements in market share and profit margins (due both to economies of scale and decreased operational costs) achievable with the new system, and the resulting ROI relative to continuing with the current system. They show that the expected increase in market share (from 20% to 30% by 2008) and profit margins have produced a 45% ROI by the end of the second year of new-system operation (2006):

$$ROI = \frac{Benefits - Costs}{Costs} = \frac{9.4 - 6.5}{6.5} = 0.45$$

The expected ROI by the end of 2008 is 297%.

The final four columns in Table 1 show expected 2004-2008 improvement in overall customer satisfaction and three of its critical components: percentage of late deliveries, ease of use, and in-transit visibility. The latter capability was identified as both important to distributors (if they know what is happening with a delayed shipment, they can improvise workarounds), and one which some of Sierra's competitors were providing. Sierra's expected 2004-2008 improvements with the new system were to improve their 0-5 satisfaction rate on in-transit visibility from a low 1.0 to a high 4.6, and to increase their overall customer satisfaction rate for order processing from 1.7 to 4.6.

4. iDAVE DEPENDABILITY ROI ANALYSIS AND RESULTS

The iDAVE dependability ROI analysis begins by analyzing the effect of increasing dependability investments from the normal business levels to the next higher levels of investment in analysis tool support ($260K), peer review practices ($210K), and test thoroughness ($314K). These correspond to the Nominal and High COQUALMO rating scale levels in Table 2. The resulting total investment of $784K yields COQUALMO estimates of a decrease in delivered defect density from 15 defects per thousand lines of code (D/KSLOC) to 3 D/KSLOC, and an increase in mean time between failures (MTBF) from 300 hours to 10,000 hours.

Table 1. Order Processing System: Expected Benefits and Business Case

		Current System			New System											
Date	Market Size ($M)	Market Share %	Sales	Profits	Market Share %	Sales	Profits	Cost Savings	Change in Profits	Cum. Change in Profits	Cum. Cost	ROI	Late Delivery %	Cust. Statis. 0-5	In-Tran. Visib. 0-5	Ease of Use 0-5
12/31/03	360	20	72	7	20	72	7	0	0	0	0	0	12.4	1.7	1.0	1.8
12/31/04	400	20	80	8	20	80	8	0	0	0	4	-1	11.4	3.0	2.5	3.0
12/31/05	440	20	88	9	22	97	10	2.2	3.2	3.2	6	-.47	7.0	4.0	3.5	4.0
12/31/06	480	20	96	10	25	120	13	3.2	6.2	9.4	6.5	.45	4.0	4.3	4.0	4.3
12/31/07	520	20	104	11	28	146	16	4.0	9.0	18.4	7	1.63	3.0	4.5	4.3	4.5
12/31/08	560	20	112	12	30	168	19	4.4	11.4	29.8	7.5	2.97	2.5	4.6	4.6	4.6

Assuming a mean time to repair of 3 hours yields an improvement in availability = MTBF/(MTBF + MTTR) from 300/303 ~ .99 to 10,000/10,003 ~ .9997.

If we use availability as a proxy for dependability, and assume that a 1% increase in downtime is roughly equivalent to a 1% loss in sales, we can use the Sierra Mountainbikes business case to determine a dependability Value Estimating Relationship (VER). Applying the difference between a .01 loss in sales and a .0003 loss in sales to the 2005-2008 Sierra new system sales total of $531M (adding up the 2005-2008 numbers in column 7 of Table 1) yields a net return on the dependability investment of (.01) ($531M) – (.0003) ($531M)= $5.31M – 0.16M = $5.15M. The COCOMO II Cost Estimating Relationships (CER's) for Tool Support and Process Maturity also generate software rework savings from the investments in early defect prevention and removal of $0.45M, for a total savings of $5.59M. The resulting dependability ROI is (5.59 – 0.784) / 0.784 ~ 6:1. A related interesting result is that added dependability investments have relatively little payoff, as there is only $0.16M left to be saved by decreasing downtime.

This analysis makes a number of assumptions that will require considerable added research to fully justify, but it provides a proof of principle that the COCOMO II CER's, the COQUALMO DER's, and business-case based VER's can be used to produce reasonable estimates of the high–payoff and lower-payoff regions for investments in dependability.

Table 2. Defect Removal Investment Rating Scales

Rating	Automated Analysis	Peer Reviews	Execution Testing and Tools
Very Low	Simple compiler syntax checking.	No peer review.	No testing.
Low	Basic compiler capabilities for static module-level code analysis, syntax, type-checking.	Ad-hoc informal walkthroughs Minimal preparation, no follow-up.	Ad-hoc testing and debugging. Basic text-based debugger
Nom-inal	Some compiler extensions for static module and inter-module level code analysis, syntax, type-checking. Basic requirements and design consistency, traceability checking.	Well-defined sequence of preparation, review, minimal follow-up. Informal review roles and procedures.	Basic unit test, integration test, system test process. Basic test data management, problem tracking support. Test criteria based on checklists.
High	Intermediate-level module and inter-module code syntax and semantic analysis. Simple requirements/design view consistency checking.	Formal review roles with all participants well-trained and procedures applied to all products using basic checklists, follow up.	Well-defined test sequence tailored to organization (acceptance / alpha / beta / flight / *etc.*) test. Basic test coverage tools, test support system. Basic test process management.
Very High	More elaborate requirements/design view consistency checking. Basic distributed-processing and temporal analysis, model checking, symbolic execution.	Formal review roles with all participants well-trained and procedures applied to all product artifacts & changes (formal change control boards). Basic review checklists, root cause analysis. Formal follow-up. Use of historical data on inspection rate, preparation rate, fault density.	More advanced test tools, test data preparation, basic test oracle support, distributed monitoring and analysis, assertion checking. Metrics-based test process management.
Extra High	Formalized* specification and verification. Advanced distributed processing and temporal analysis, model checking, symbolic execution. *Consistency-checkable pre-conditions and post-conditions, but not mathematical theorems.	Formal review roles and procedures for fixes, change control. Extensive review checklists, root cause analysis. Continuous review process improvement. User/Customer involvement, Statistical Process Control.	Highly advanced tools for test oracles, distributed monitoring and analysis, assertion checking Integration of automated analysis and test tools. Model-based test process management.

5. CASE STUDY IMPLICATIONS

More importantly, though, the case study's ability to relate a specific dependability analysis to specific human decisionmaking issues opens the door to an entirely new set of speculations and research directions about the nature of value-based decisionmaking and its implications for dependability-oriented software engineering. In the case study, because added investments in availability have little payoff, does this mean that the Sierra decisionmakers will lose interest in further

dependability investments? Probably not. More likely, they are operating within a Maslow need hierarchy in which satisfied availability needs are no longer motivators, but in which higher-level needs such as reducing security risks may now become more significant motivators.

This casts the analysis of dependability attributes in an entirely new light. Previously, the problem of software attribute analysis has been largely cast as an exercise in static multi-attribute optimizing or satisficing, operating on some pre-weighted combinations of dependability attribute satisfaction levels. The practical decision making issue above indicates that achieving an acceptable or preferred combination of dependability attributes is generally not a pre-specifiable problem but rather a dynamic process in which satisfaction of currently top-priority dependability attributes leads to a new situation in which the attribute priorities are likely to change.

In this situation, dependability attribute requirements become more emergent than pre-specifiable. The process for achieving acceptable dependability becomes no longer a single-pass process, but an evolutionary process, subject to the need to anticipate and develop architectural support for downstream dependability needs. The types of dependability analyzers that become important increasingly involve the types of dynamic and adaptive value–oriented dependability mechanisms being explored in papers such as [4, 6, 8].

6. CONCLUSIONS

The multidimensional nature of "dependability" decisionmaking is compounded by the potentially very high levels of investment required to achieve very high levels of dependability. This creates a demand for methods of reasoning about the cost and value of achieving various levels of dependability attributes in particular project situations.

The iDAVE model presented here provides an overall framework and an initial set of tools for reasoning about the value of dependability. Application of an initial version of iDAVE to an example project decisionmaking situation shows that the model can produce reasonable estimates that distinguish higher-payoff and lower-payoff regions of a project's investment in dependability.

Use of the iDAVE model in this decision situation provided evidence that dependability attribute requirement levels tend to be more emergent than pre-specifiable; that dependability analysis and achievement processes tend to be more evolutionary then single-pass; and that dynamic and adaptive value-based dependability mechanisms will become increasingly important.

7. REFERENCES

[1] Abts, C., Boehm, B., and Clark, E., "COCOTS: A Software COTS-Based System (CBS) Cost Model," Proceedings ESCOM, 2001.

[2] Boehm, B. and L. Huang, "Value-Based Software Engineering: A Case Study," to appear, *IEEE Software*, March 2003.

[3] B.Boehm, C. Abts, A.W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Riefer, and B. Steece, Software Cost Estimation with COCOMO II, Prentice Hall, 2000.

[4] David Garlan and Bradley Schmerl, "Model-based Adaptation for Self-Healing Systems," ACM SIGSOFT Workshop on Self-Healing Systems (WOSS'02), November 18-19, 2002.

[5] Reifer, D., Boehm, B., and Gangadharan, M., "Estimating the Cost of Security for COTS Software," Proceedings Second Intl. Conf. COTS-Based Software Systems, February 2003.

[6] Mary Shaw, "'Self-Healing': Softening Precision to Avoid Brittleness," Proceedings of the First ACM SIGSOFT Workshop on Self-Healing Systems (WOSS '02) Charleston, South Carolina, November 2002, pp. 111-113.

[7] Steece, B., Chulani, S., and Boehm, B., "Determining Software Quality Using COQUALMO," in *Case Studies in Reliability and Maintenance*, W. Blischke and D. Murthy, Eds.: Wiley, 2002

[8] K. Sullivan and J. Knight, "Information Survivability Control Systems," Proceedings of the 21st International Conference on Software Engineering, May, 1999, pp. 184—193.

Use of Real Options Theory to Value Software Trade Secrets

Donald J. Reifer

University of Southern California
Los Angeles, CA 90089
011-310-530-4493
dreifer@earthlink.net

Abstract: In this position paper, we discuss use of real options theory to value software trade secrets. We start by outlining current valuation practices and problems. We next outline a valuation framework that permits software experts to value trade secrets when involved in litigations. The framework takes advantage of real options theory to derive a fair value for use in valuing a trade secret using either the currently accepted cost, income or market approach. We conclude by focusing on the barriers that software experts will have to overcome when presenting their findings within a courtroom environment to non-software participants (judges, attorneys, juries, etc.).

General Terms: Economics, Management

Keywords: Valuation framework, real options theory, software trade secrets

1. Introduction

During the past decades, attorneys become involved in software litigation especially as license, patent, copyright and trade secret terms and conditions have been violated. While license breaches have proved relatively easy to value [1], determining the worth of patents, copyrights and trade secrets has not [3]. Some of the many issues that made valuation difficult include, but are not limited to, the following:

- Traditional approaches to determine value focus on market price and do not include adequate allowances for appreciation of assets, market growth or technology, functional, physical and economic obsolescence [2].
- The “fair value,” “fair market value,” “market value,” “acquisition value,” or “use value” of an intangible asset is difficult to determine especially in light of current economic conditions. Fair value is defined as the amount in terms of dollars that a willing and able buyer would pay for these assets under current market conditions [9].
- Determination of value under the “highest and best use” principle is hard to determine as legal, physical, financial and maximum profitability conditions vary depending on premises of value (e.g., value in place, value in exchange, value in continued use, etc.) [9].
- The range of use and profitability of the use of the intangible assets are difficult to determine in light of future competition and market conditions [6].
- States treat valuation of intangible assets like software trade secrets differently and the case law is non-uniform [5].
- Few cases involving valuing intangible assets like software trade secrets are available to establish precedence in a court of law [4].

In light of these issues, better frameworks are needed to help experts develop a reasonable value estimates for software intangible assets, especially trade secrets, which is the focus of this position paper.

2. Approach

Currently, valuation experts use cost, market and income based approaches, the later of which employs discounted cash flow methods, to value intangible assets.

Valuation is done using cost and/or income projections to develop a fair value estimate to use as a standard for compensation. To augment these approaches for valuing trade secrets, we have enhanced the following valuation framework developed by Pitkethly [8] at Oxford for patents as follows in Figure 1 to include options that address changing risk (e.g., due to market and other conditions):

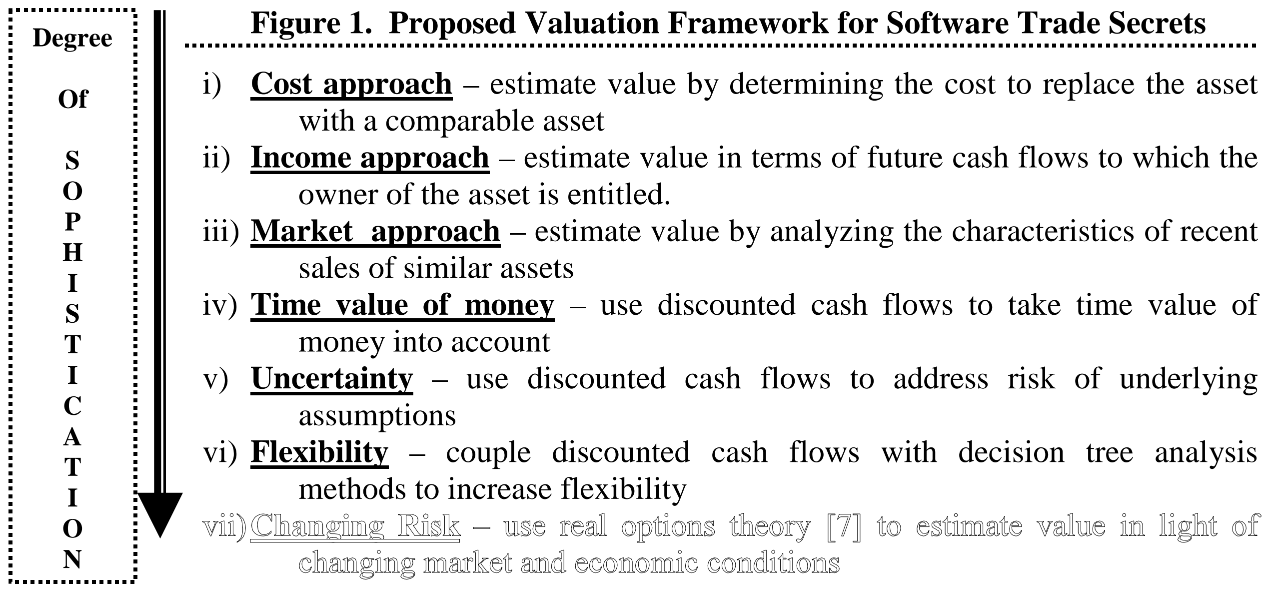


Figure 1. Proposed Valuation Framework for Software Trade Secrets

Degree Of SOPHISTICATION

i) **Cost approach** – estimate value by determining the cost to replace the asset with a comparable asset
ii) **Income approach** – estimate value in terms of future cash flows to which the owner of the asset is entitled.
iii) **Market approach** – estimate value by analyzing the characteristics of recent sales of similar assets
iv) **Time value of money** – use discounted cash flows to take time value of money into account
v) **Uncertainty** – use discounted cash flows to address risk of underlying assumptions
vi) **Flexibility** – couple discounted cash flows with decision tree analysis methods to increase flexibility
vii) Changing Risk – use real options theory [7] to estimate value in light of changing market and economic conditions

Legend: modification made to the original framework in [8] in outline script

For the purpose of this paper, a trade secret is defined as information, including formulas, patterns, compilations, programs, devices, methods, techniques or processes that (1) derives independent economic value, actual, or potential, from not being generally known…and (2) is the subject of efforts that are reasonable under the circumstances to maintain its secrecy [8].

To illustrate the value of the framework, let's take the "wait and see" example shown in Figure 2. This option assumes that instead of plunging into a new market right, we wait one year until economic conditions and chances of success are better. Figure 3 shows the financial advantages of embracing this "wait and see" option.

Figure 2. Real Option Opportunity Tree

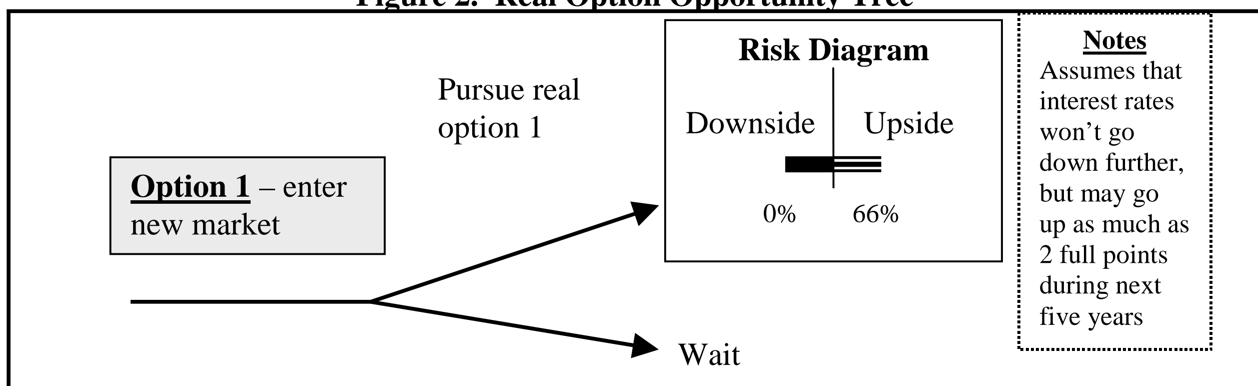
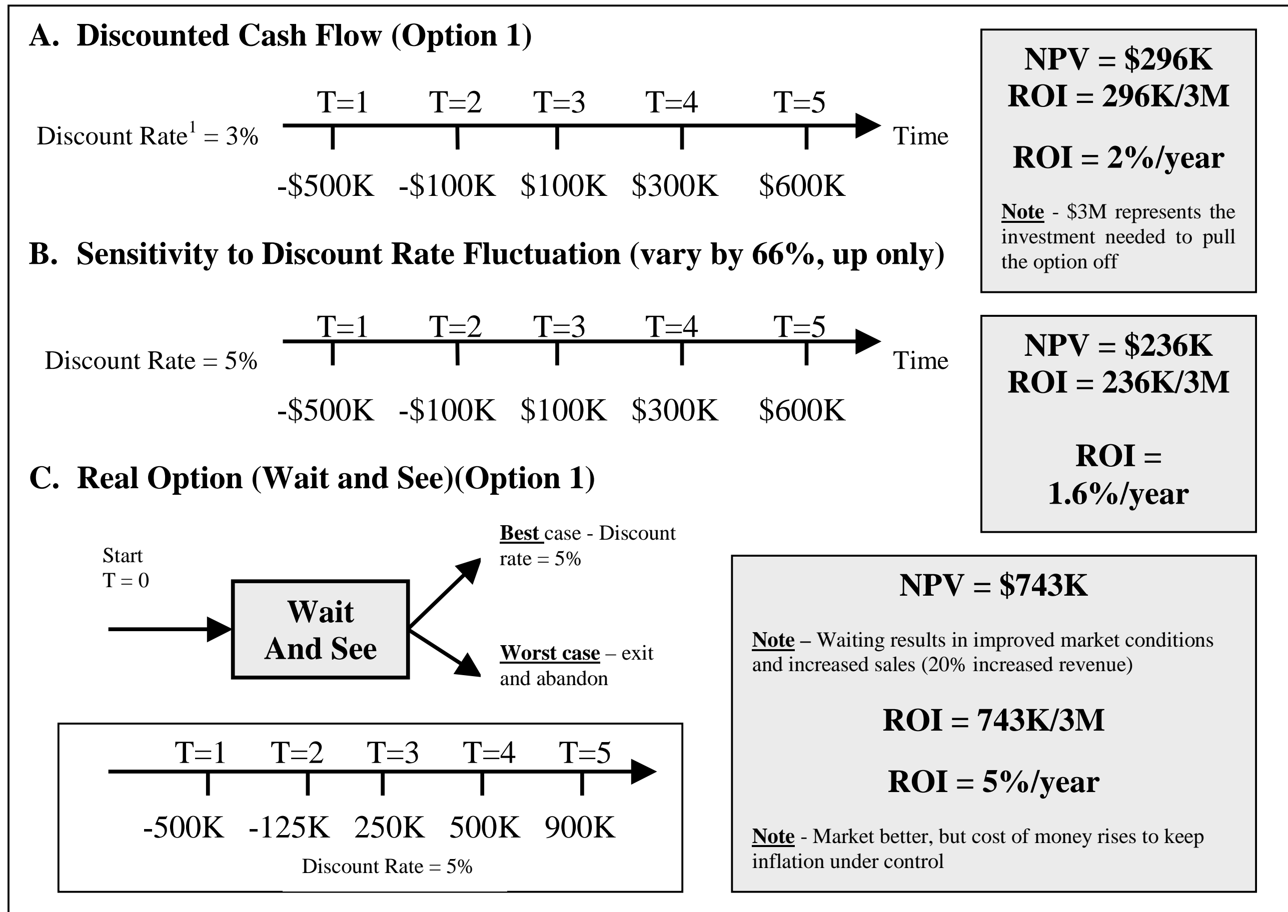


Figure 3. Financial Analysis (Net Present Value and Real Option Analysis)



To assess options completely, you really need to assess the impact of risk. To accomplish this, we propose using the concept of risk vectors. As illustrated in Table 1, risk vectors allow us to weight available options by the forecasted probability (Prob.) of their success. Such probabilities can be determined using a variety of means including statistical analysis, game theory (assess alternatives assuming economics are a game of chance) and Monte Carlo simulations (assess range of potential impacts).

Table 1. Real-Option Risk Vectors

	Option	NPV	Prob.	NPV_W
Vector 1	Pursue option right now	$296K	0.95	$281.2K
Vector 2	Wait and see	$743K	0.86	$639K
….				
Vector n	Do nothing	0	0	0

The table shows using the figures from Figure 3 that even though the "wait and see" option is more risky than other alternatives; it is still the preferred approach. It should be noted that the "do nothing" option is always an alternative. Although it costs nothing, it also yields nothing. However, this option can turn out to be the preferred alternative when all other vectors have a negative Weighted Net Present Value (NPV_W).

3. Summary and Conclusions

The example presented is not fictitious. While we changed the numbers and context

somewhat, we are using the proposed real options framework in a current litigation to value trade secrets that were allegedly divulged. Based upon a search, this is the first case to use such a framework to value software trade secrets. The few cases that we have found rely on discounted case flows almost entirely to come up with a value. Hopefully, the framework we developed will hold up as the case goes through its pre-trial motions and goes to court.

Attorneys specializing in valuing Intangible Assets and Intellectual Property tell us that our approach is innovative. However, time will tell if the approach holds up in a court of law. If it does, the real options approach will be used to create a benchmark that will establish how software trade secrets are valued now and into the future.

3. References

[1] Appraisal Standards Board, Uniform Standards of Professional Appraisal Practice, The Appraisal Foundation, 2002.

[2] Cole, Roland J. and Barnes & Thornburg, "Valuing IP Assets: The Legal Aspects," ICLE Spring 2002, pp. 1-21, 2002.

[3] Damiano, Karen, "Valuing Intangible Assets under SFAS 141," Insights, Winter 2002, Willamette Management Associates, 2002, pp. 17-20.

[4] Goldenberg Norman S. and Tenen, Peter, Legal Briefs: Intellectual Property, Casenotes Publishing Company, 2001.

[5] Loud, Adrian and Reilly, Robert A., "What is a Trade Secret Worth?" Insights, Willamette Management Associates, Special Issue, 2000, pp. 15-18.

[6] Mard, Michael J., "Intellectual Property Valuation Challenges," The Licensing Journal, May 2001, pp. 26-30.

[7] Mun, Johnathan, Real Options Analysis, John Wiley & Sons, 2002.

[8] Pitkethly, Robert, The Valuation of Patents, Judge Institute Working Paper 21/97, The Judge Institute of Management Studies, Cambridge, England, 1997.

[9] Reilly, Robert F. and Schweihs, Robert P., Valuing Intangible Assets, McGraw-Hill, 1998.

Intentional Software Systems:
Engineering Intentionality in Processes of Software Development and Runtime Execution

Position Paper: 5th International Workshop on Economics-Driven Software Engineering Research

Kevin Sullivan
University of Virginia Department of Computer Science
Charlottesville, VA 22904 USA
sullivan@cs.virginia.edu
February 15, 2003

To satisfice their dynamic norms—that is, to be good enough with respect to prevailing *process*-oriented criteria such as survival, ethics, maximizing utility, or creating economic value—intentional systems must be organized to perceive, represent, processes, and respond appropriately, in the environment, to what really *matters*. In this paper, I take the position that software processes—both software design processes and computerized runtime processes—all too frequently behave badly today because they are not conceived and implemented as *intentional systems* governed by appropriately selected *dynamic norms*. The lack of attention to the issue of intentionality, in general, and to the choice of norms, in particular, often leaves such processes subject to wrong or inadequate norms, thus "unaware" of and unresponsive to what matters.

To make the notion of dynamic norms clear, consider the example of a software start-up. The developers want to restructure its code, which, having evolved somewhat messily from a prototype, is in pretty bad shape. Yet, the crucial norm for the organization is business *survival*. If satisficing this norm demands ongoing delivery of functional increments on a tight schedule, code restructuring might have to wait, no matter how attractive it is by other norms, such as the static (entity-oriented) norm, on source code structure, of "being easy to change."

Here is another example. Take the (forthcoming) IEEE Software, March 2003 feature article [1], in which, Boehm and Huang show that the widely used Earned Value Management System for controlling development processes actually has nothing do with *creating stakeholder value*. Rather, the norm governing this software development process is that *spending occur as planned*. A process can be successful by this norm and still be an utter failure by what matters. A norm that arguably would steer the process to perform better with respect to stakeholder outcomes (which, here, is what is deemed to matter) would be "continuous satisficing of all stakeholder value propositions."

More importantly, the prevailing norm fails to cause several crucial things to happen. First and foremost, the norm doesn't demand that the process recognize, monitor, or represent stakeholder value propositions. The wrong norm thus leaves the process ontologically hobbled. Second, as a result of being "unaware" of stakeholder value propositions and of the goal of satisficing them, the process is not driven to be responsive to what matters.

My position is that we need to create systems *engineered for intentionality*.[1] The first step is the choice of governing norms. The second is to confront the technical challenges of crafting perceptual, process-internal representation and processing, and response mechanisms necessary for norm-satisficing, intentional behavior. Mechanisms will generally be needed to perceive and represent material states of both process and environment, and to plan and execute normatively appropriate, adaptive behaviors. The resulting processes will seem aware of their environments, to have goals (desires), and will behave accordingly, acting and adapting appropriately over time.

Consider two examples. First, a development process under a stakeholder-satisfaction norm might represent stakeholder value propositions as evolving business cases, monitor competitors' actions, use a software

[1] Intentionality, in this sense, has a clearly established meaning in the philosophy of mind. The two big problems in that field are consciousness—awareness of self and other—and intentionality—to be *about* something, to have beliefs and desires.

architecture supporting schedule control as an independent variable, and drop features as needed to beat competitors to market. Second, a critical societal information system might "perceive" when it is "under attack" and reconfigure automatically to provide continuity of services *deemed* essential.

I contend that the intentional perspective holds promise for the design of software *processes* of both program development and execution. Of some interest is the observation that the approach says little about software artifacts, the traditional focus of most software engineering attention and technology. The concern is with process, behavior, and what is ultimately sought.

This idea appears to have some merit in at least three dimensions. First, it rests on seminal contemporary work in the philosophy of mind and computation—and intentionality, in particular—namely the work of Brian Cantwell Smith [3]. Second, it has potential not only to rationalize, but to bring into computer science proper—as an intentional science—a consideration of *value and values*. Third, the mechanisms needed for intentional behavior in software development and runtime processes will push the limits of existing technology in the areas of system monitoring of self and environment, internal representation including reference to relevant externals, and appropriate inference and planning techniques, based in part on mappings relating available actions to norm-denominated expected outcomes. The notion of autonomic computing, self-healing systems, and the like, clearly fall under the scope of the notion proposed here.

There is an important homonymic line of work with which the current proposal should not be confused: namely, intentional software [2], and the closely related areas of intentional programming and aspect-oriented software development. These important but quite different ideas all revolve around a specific, *static, technical norm*, pertaining to the relationship between two software representations: design structure and source code. The norm is that the structure of the *source code* should reflect the *design intent* of the developer. "Using Intentional Software, the actual software *source code looks like the design* [1]." The norm is static in that it applies to artifacts, not processes. It is logico-technical, rather than, say ethical or financial, insofar as it concerns structural relationships in code, not *values* (although the motivation is non-technical).

The idea presented here is fundamentally different. First, it centers on dynamic norms: that is, norms that apply to processes rather than to static artifacts. Second, it is a broader idea. To our view, it is the first formulation promising to bring considerations as diverse as ethics (such as the Rawlsian ethics of fairness underlying Boehm's Win-Win model), business value, or *values* (such as having fun), into software engineering in a general and scientific way—rather than as interesting, important, and useful, but ultimately awkward, glue-ons. Third, our idea leads to an entirely different set of issues than those addressed by intentional software. Rather than focusing on modularity, expressiveness, and structural continuity across representations at various levels of abstraction, attention turns to ontology, perception, representation, reasoning, action, and, ultimately, mattering: What material "things" does a process recognize, represent, reference, compute about, and, affect? What does it *care* about? What *matters* to it?

Finally, by making the choice of norms for synthetic processes explicit, the notion presented here provides a link between process design and human values. The question is, under what other norms do we choose the norms to impose on our designed processes? Why make Rawls's ethic of fairness the governing "law" of a development approach such as Win-Win? How about long-term shareholder value maximization? Preservation of freedom? The choice of dynamic norms fundamentally determines many key parameters of a process. Today, these considerations are most important in relation to human processes of software development. However, as our capabilities develop to create more "aware" and "autonomic" software-based systems, we might have to begin to ask about their "intentions," as well—that is, to make sure that we pick "good" ones for them. Ultimately, it's a question of what matters *to us.*

Bibliography

[1] Boehm, B. and L. Huang, "Value-Based Software Engineering: A Case Study," forthcoming, *IEEE Software,* March, 1993.

[2] Intentional Software (corporation). Front page of web site as of this writing. http://intentsoft.com/

[3] Smith, B.C., "God, Approximately," presented at the MIT AI Lab, November 18, 1998. http://www.ageofsig.org/people/bcsmith/print/smith-godapprox4.pdf

Time is Not Money

the case for multi-dimensional accounting in value-based software engineering

Vahe Poladian, Shawn Butler, Mary Shaw, David Garlan
{vahe.poladian, shawn.butler, mary.shaw, david.garlan}@cs.cmu.edu
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

ABSTRACT

"Time is money", or so goes the old saying. Perhaps influenced by this aphorism, some strategies for incorporating costs in the analysis of software design express all costs in currency units for reasons of simplicity and tractability. Indeed, in theoretical economics all costs can, in principle, be expressed in dollars. Software engineering problems, however, often present situations in which converting all costs to a common currency is problematical. In this paper we pinpoint some of these situations and the underlying causes of the problems, and we argue that it is often better to treat costs as a multidimensional value, with dimensions corresponding to distinct types of resources. We go on to highlight the differences among cost dimensions that need to be considered when developing cost-benefit analyses, and we suggest mechanisms for mediating among heterogeneous cost dimensions.

Keywords

Cost analysis, multi-dimensional cost analysis, value-based software engineering.

1. ACCOUNTING FOR COSTS IN SOFTWARE ENGINEERING

Although engineers traditionally focus on the functionality of their designs, they are becoming increasingly away of the need to address total cost of developing and owning the software.. A common approach to cost-benefit analysis is to express all costs and benefits in terms of dollars. To a first approximation, expressing all costs in a single dimension may seem like a reasonable solution. In practice, however, simplistic conversions of costs (or benefits) can be problematical.

Consider a military operations center, which is responsible for managing and directing battlefield assets during times of conflict. A typical operations center uses several applications that will demand different amounts of computer resources (e.g., bandwidth and CPU resources) depending on the military situation. For example, satellite and air reconnaissance assets can provide real-time video coverage of the operational area, but not all the time. Simultaneously, communication channels stream important intelligence and operational information to the center's military commander, but processing the messages is CPU intensive. Unfortunately the amount of information available to the commander can exceed his capacity to receive and process the information and affect his ability to make informed decisions.

The value of each application, and thus the value of the computing resources, will depend on the current military situation. For example, at times the commander will need very detailed videos of the battlefield to make operational decisions, but at other times the commander will need to receive intelligence over communication channels and a less detailed picture of the battle will be adequate. Therefore, dynamic reconfiguration of the computing resources may be essential to making timely military decisions.

The quality of service that these applications provide can be fine-tuned through computer resource adjustments to meet the commander's needs. For example, increasing frame rates and bandwidth allocations can enhance video imagery, but the resulting demand for CPU cycles to process video images can cause delays in message processing.

At any given point in time, finding the optimal allocation of computing resources depends on the value that each application provides to the commander. Finding the optimal resource allocation ultimately requires all the alternatives to be comparable -- typically expressed in a common metric. However, there can be serious drawbacks to making these conversions too early in the analysis process.

First, it is difficult to associate cost or value with a resource without complete information about the resource and the context of its use. In the example above, the value of bandwidth was highly dependent on the battlefield situation and weather. In some cases the value of the resource may be in saving lives, but in other situations the value of the resource may be tied to common economic costs, such as fuel or energy costs, which are more easily translated into dollars[1].

Second, a conversion between metrics may adversely affect the type of analysis that you can do. The value of the resource may be non-linear with respect to the preferences of the user (or commander in the example above). Converting the value of computing resources, such as bandwidth and CPU, to dollars implies that each of the resources is as finely divisible as dollars. Increasing the resolution of video images requires a stepwise increase in bandwidth before the user recognizes a difference in the quality of the video. As a result, calculus-based solutions may appear adequate to solve the problem in the abstract, when in fact discrete algorithms are more appropriate for the problem at hand.

Third, conversions to a common currency can lose information that should affect the types of feasible solutions. Some resources are perishable: they are only valuable for short periods of time, and after that they have no residual value. Converting such a resource to one that is not perishable will cause important information to get lost in the process. In fact, the obtained result may not be feasible according to the original formulation of the

[1] Despite studies that calculate the price of an individual's life, few decision makers are willing to make explicit comparisons.

problem. For example, unused bandwidth is gone forever, and allocating bandwidth to satellite imagery when the satellites are not overhead is not a feasible solution.

These problems can be avoided by using methods that recognize and respect the different properties of different resources. Here we regard cost as a multidimensional quantity, with different dimensions corresponding to different non-commensurable cost metrics. Section 4 presents examples based on two such methods. One of the examples focuses on the automatic run-time configuration of software components based on preferences of the user. The second example tackles the problem of choosing the optimal set of countermeasures to minimize threats to a corporate IT infrastructure.

In this paper, we contribute to understanding the problem on incommensurable multidimensional costs and finding solutions for particular projects by:

- *Characterizing types of costs to show their differences.* In Section 2 we catalog resources that are commonly used in analyzing costs in software development.
- *Proposing a model for treating cost as a multidimensional measure.* In Section 3 we present a model that explains the essential differences among these costs.
- *Analyzing the problems of mapping among cost dimensions*. In Section 4 we discuss analysis techniques that carry through multidimensional costs.
- *Showing how to accommodate methods that require uni-dimensional costs*. In Section 5 we generalize from the examples of Section 4 and discuss ways to balance the information needs that require multiple dimensions with the analysis needs that require a single dimension.

2. SOURCES OF SOFTWARE DEVELOPMENT AND QUALITY COSTS

In software engineering and systems research, cost-benefit analyses have been used to solve cost estimation and optimization problems. SAEM [2], [3] is a cost benefit analysis model that helps security managers choose the best set of countermeasures. Odyssey [9] provides runtime middleware that helps adapt application behavior to resource availability. The Nemesis [10] operating system uses shadow prices and careful accounting to determine optimal allocation of resources among competing applications. Aura [8] aims to reduce user distraction in interactive computing by accounting for human attention as a resource. COCOMO II [1] is a software cost estimation model that calculates the cost of a software project based on various organizational and project parameters. The works cited consider costs and benefits to estimate benefit, determine optimal allocations, and estimate cost.

Table 1 catalogues some of the resources that are considered by the analyses of the research described above. This list is representative rather than complete; it provides the basis for the examples we use in later sections.

Table 1. A Selection of Resources and their sources

Cost dimension	*Examples, citations*
Purchase cost, currency (dollars, for simplicity)	Classical Economics
Staff time	COCOMO [1]
Reputation	SAEM [2]
Lives lost	SAEM
Calendar time, days	COCOMO
Bandwidth	Odyssey [9], Nemesis [10]
Battery Capacity Remaining	Odyssey, Nemesis
User attention	Aura [8]
Software application, e.g. Microsoft Word	Aura

3. PROPERTIES of COSTS/RESOURCES

The introductory example motivates the need to consider separate resource dimensions in cost-benefit analysis. But what are some of the characteristics of different resources that need to be considered during such analyses? Further, how would these characteristics influence the choice of analysis technique? To help answer these questions, we discuss the different properties of some of the resources identified in Table 1, with particular focus on understanding how these differences can be reconciled or mediated. At the end of the discussion, we summarize our findings in Table 2 for a sample of resources.

- *Divisibility/granularity.* This property describes how dense the space of the resource is. Intuitively, this property indicates in what increments the resource can be allocated. Possible values are:
    - *Continuous*: The resource can be allocated at a very fine grain. Bandwidth and battery energy are resources that fit in this group.
    - *Discrete but dense*: The possible allocation points are many, but allocation can not be made continuously. Currency fits this group.
    - *Sparse discrete:* There are very few possible points in the resource space. Editing a document with a particular application, e.g. Microsoft Word, falls in this category.

  The granularity of a resource can influence the choice of the solution method. With continuous resources and in some cases discrete dense resources, calculus-based solutions work well, especially if resource requirements can be described as closed formulas. Sparse discrete resources are best analyzed with discrete methods such as integer programming and knapsack algorithms. Problems with continuous and dense discrete resources can also be tackled using discrete solutions, at the expense an approximate answer. This can be a justified trade-off if no closed-form formulas exist to describe the functions.
- *Fungibility.* This property describes whether a particular resource can be converted to another resource. This property makes sense in the context of a specific problem, and with respect to specific other resources. For example,

- *Complete fungibility:* Common currency is fungible to most other resource.
- *Partial fungibility:* Some interchange is possible between bandwidth and CPU cycles in the software runtime configuration problem. Consider different MPEG decoders using different compression algorithms. One decoder may be relatively bandwidth intensive, while the other may be CPU intensive. Availability of multiple decoders makes it possible to convert between bandwidth and CPU cycles. It is important to realize that the tradeoff is limited a few points.
- *No fungibility:* In software cost estimation problem, it is well known that calendar days and staff months are not interchangeable. Additional staff may even lengthen development time.

- *Measurement Scale.* This property describes the kind of scale that is appropriate for measuring a resource. For example, the set of domestic animals (dog, cat, cow, etc) has nominal scale, as there is no ordering relationship between elements in that set. See the Appendix for a review of measurement scales. Possible choices are:
  - *Nominal.*
  - *Ordinal.*
  - *Integer.*
  - *Ratio.*

  Cost-benefit analysis can sometimes be tackled by converting all resources to the same scale. However, conversion among resources of different scales must be made only when conversions are justified. See Section 4.2 for an example of such conversion.

- *Economies of scale.* This property describes the extent to which a percentage increment in a resource affects the increment of the output of a product that uses the resource as input. Possible values are:
  - *Superlinear Scale:* (also known as positive economies of scale). If a percentage increment in a resource results in proportionately higher increase in output, then we say resource has superlinear scale. Consider the problem of searching for a given record by its unique key in a large database. As a measure of output, consider the size of the database (e.g., total number of records) we are able to search in a fixed amount of time, and as a measure of input, consider the size of hardware we need to have (e.g., CPU speed). Recall that the binary search algorithm runs in time logarithmic with respect to number of items. Thus, CPU size exhibits superlinear scale with respect to the problem size in this case, because incremental increases in the CPU performance dedicated to the search space result in increasingly proportionally larger search space covered.
  - *Linear Scale:* (also known as neutral economies of scale). The benefit of additional quantities of the resource is independent of the problem size.

Table 2. Properties of Sample Resource Dimensions

			Properties of Costs					
		Units	**Measurement Scale**	**Granularity**	**Fungibility**	**Perishability**	**Economies of Scale**	**Rival**
Cost Dimension	Purchase cost	Dollars	Ratio	Dense	Y	N	Linear	Y
	Staff time	Months	Ratio	Sparse	N	Y	Sublinear	Y
	Reputation	Scale	Ordinal	Sparse	N	N	N/A	Y
	Lives lost	Number of Humans	Integer	Sparse	N	N/A	N/A	N/A
	Calendar time	Days	Ratio	Sparse	N	Y	Depends	N
	Bandwidth	Mbps	Ratio	Continuous	N	Y	Depends	Y
	Battery	Joules	Ratio	Continuous	N	N	Depends	Y
	Human attention	Seconds	Ordinal	Sparse	N	N	N/A	Y
	Software application	N/A	Nominal	Sparse	Y	N	N/A	N

- *Sublinear Scale:* (also known as diseconomies of scale). If a percentage increase in resource results in proportionately smaller increase in output, then we say the resource exhibits sublinear scale. Staff size, as input to software projects, exhibits slight diseconomies of scale (COCOMO II).

Notice that this property only applies to resources that are measured on a ratio scale or that can safely be converted to ratio scale.

- *Perishability.* This property describes whether the resource will be forever lost, if not used by certain point in time. Possible values are perishable and non-perishable.
  - *Perishable:* Bandwidth is perishable.
  - *Non-perishable*: Battery energy is not perishable.

  Problems involving perishable and non-perishable resources need to introduce time into the analysis and account for intertemporal possibilities. Utility functions are one possible solution.

- *Rival.* A rival resource is such that the consumption of a unit or amount of a resource by one person or entity precludes the consumption of the same unit by another person.
  - *Rival:* Money, labor, bandwidth, CPU cycles.
  - *Non-rival:* Software application, information goods, calendar days.

  Efficiently allocating rival resources among multiple requestors is the heart of many optimization problems. Aggregate demand for a rival resource can not exceed total supply available. Allocation analysis can be complicated when multiple sources of a resource are available. For example, consider the problem of choosing where to run a particular software application, given a choice of two servers.

4. MULTIDIMENSIONAL ANALYSIS TECHNIQUES

In this section, we present two examples of techniques that consider multiple dimensions of costs in solving cost-benefit problems in practical software systems. The first analysis is performed at run time and helps configure software applications on a mobile computer. The second analysis is performed off-line and optimizes the selection of security technologies to counter threats against corporate IT infrastructure.

4.1 Value-based Software Runtime Configuration

Let's revisit the scenario from the introduction and illustrate some of the problems that can result from early conversions. Tables 3 and 4 show hypothetical runtime operational profiles of the two programs described in Section 1: Messaging and Real-time Video. The quality level information in the first column is provided by the application specification. The second and third columns give resource usage (percentage-of-resource-required/second) to achieve the specified quality level. The resource data depends on the runtime characteristics of the application and the data processed, which can be obtained using profiler tools. The value information in the fourth column is represents the value assessments of the battlefield commander, which can be obtained through elicitation interviews.

The overall objective is to maximize the sum of the values: Value(Messaging) + Value(real-time video). Notice that the quality level is an ordinal scale: it does not make sense to say how much more or by what factor the next level is better than the previous one. A value function, which normalizes the commander's value assessments, converts the quality levels into a ratio scale on the basis of additional information elicited about the application. Bandwidth and CPU are both perishable resources and cannot be stored for future use.

Tables 3. The Operational Profiles of the Applications

Messaging

Quality Level	CPU, %	BW, %	Value
None	0	0	-∞
Very Low	57	17	1
Low	61	23	12
Medium	72	27	55
High	79	29	68
Very High	98	32	75

Real-time Video

Quality Level	CPU, %	BW, %	Value
None	0	0	-∞
Bad	12	43	3
Acceptable	19	52	30
Good	23	69	45
Very Good	27	78	57
Excellent	34	93	89

One approach to solving this problem is to take as given the external prices of CPU and Bandwidth, and convert these to a common currency. Assume the cost of one percent of available CPU is 2 units, and that of one percent of the available Bandwidth is 3 units. A total of 2 * 100 + 3 * 100 = 500 units of total resource are available. Table 4 present the resource requirements in terms of the single currency. Column 2 shows the cost in common currency of providing that level of quality, and column 3 shows the percentage of that cost.

Tables 4. The Operational Profiles Using Common Currency, 500 Units Available

Messaging

Quality Level	Cost	Cost, %
None	0	0
Very Low	165	0.33
Low	191	0.38
Medium	225	0.45
High	245	0.49
Very High	292	0.58

Real-time Video

Quality Level	Cost	Cost, %
None	0	0
Bad	153	0.31
Acceptable	194	0.39
Good	253	0.51
Very Good	288	0.58
Excellent	347	0.69

Notice that according to Table 2, the best combination that can be achieved is High quality of Messaging and Good quality of Real-time Video, which costs 498 units, or just under 100%, and is valued at 113. However, after consulting Table 3, we notice that CPU would be utilized at 101 percent, making that combination unattainable. The problem is that we have allowed conversion of unused Bandwidth into CPU, despite the inappropriateness of this conversion. Indeed, each quality point for either application can be obtained using only a unique CPU and bandwidth vector. The root of the problem is that CPU and bandwidth are not fungible, and our assumption of fungibility leads to an incorrect solution.

Another approach to this problem is to use derivatives, e.g. a calculus method called Lagrange Multipliers. However, since the space of quality points is sparse, any kind of continuous

approximation is likely to yield a solution that is also not in the space of available quality points.

Currently, we are investigating the use of a Multidimensional, Multiple-Choice 0-1 Knapsack algorithm for handling this type of problem [11]. The solution to that problem is similar to the uni-dimensional version, except that it uses a parameterized vector for resource prices, and it iteratively refines the value of the parameters to eventually determine accurate conversion prices.

This technique can be extended to handle perishable resources, such as battery energy. In this case, intertemporal choices must also be considered, and an explicit function must be introduced to measure the value of saving energy for future use.

4.2 Security Attribute Evaluation Method

Traditional security risk management techniques advocate that security managers determine an organization's risk of an attack (a) by calculating the probable cost of the attack, i.e. $risk_a = cost * p(a)$, where p is the probability of the attack. For example if a virus attack results in x hours of lost productivity, then the risk of the attack is typically determined as $\text{Risk}_{\text{virus}} = x *$ average hourly wage rate * p(virus). Converting lost productivity to dollars appears relatively straightforward, but other types of attack consequences such as damaged public reputation or impaired quality of patient care are not as easily converted to dollars.

Unfortunately, simplistic risk calculations such as the one just described do not capture the value that organizations place on different types of costs. First, security managers find it difficult to attach explicit financial value to intangibles, such as public reputation or quality of patient care. Second, even when explicit economic value can be assessed, business executives are often skeptical about the underlying assumptions and lack confidence in the results. For example, organizations are usually less concerned about lost productivity from an attack than direct financial loss. Therefore, techniques that preserve the value of the outcome may produce more convincing results.

The Security Attribute Evaluation Method (SAEM) [2] uses multi-attribute decision analysis techniques to help security managers choose the best set of countermeasures against possible attacks. Although the SAEM risk assessment process reduces costs to a common *threat index*, the organization's value of each type of cost is captured as part of the threat index.

The risk assessment cost dimensions are the *most-likely* types of consequences of a successful attack, e.g., revenue lost, staff hours lost, reputation damage suffered. Security managers determine these cost dimensions. In order to determine the best set of counter-measures, SAEM calculates the relative importance of each consequence. The method introduces value functions to assess the incremental importance and normalize consequences, and uses the SWING-weight method [5] to elicit the importance of each consequence. Finally, SAEM computes the threat index, which is essentially a common, but neutral, cost measure that indicates the relative costs of an attack to other attacks.

5. RECONCILING MULTIDIMENSIONAL ANALYSIS WITH ONE-DIMENSIONAL TECHNIQUES

We have argued that cost-benefit analyses often need to maintain multidimensional representations of costs in order to preserve information about qualitative differences among distinct types of costs. We identified some of the principal characteristics of costs that impede conversion to common units, and we showed the consequences of failing to preserve the distinguishing information.

Eventually, though, we need to make decisions. To do so, we must be able to compare multidimensional costs. Further, some analysis techniques require scalar costs; the value of the analysis, even with loss of information, may be large enough to offset the information loss.

We believe that an appropriate strategy is to preserve the distinctions among different costs as long as practical and to reduce the cost vector to a scalar when circumstances force the conversion.

Consider, then that a system with N cost dimensions is being evaluated in an N-dimensional space, and assume for simplicity that the dimensions are orthogonal. Then each cost point is described by its cost in all the dimensions and corresponds to a point in space. The vector from the origin to that point represents that cost, and the length of that vector is one-dimensional. The problem is, how can we establish a value for the length of the vector? It is clearly inappropriate to treat the indices for the various dimensions as if their units were equivalent. Instead, we believe the proper approach is to preserve the N-dimensional analysis as long as possible, then perform late binding on the conversion by assigning a conversion function from each dimension into some common units. This makes it possible to compute the vector length and reduce the cost vector to a scalar. Vectors in each dimension can be scaled using parameterized weights, and then a common cost can be computed using root-mean-square as if it were Cartesian. This process can be iterated several times in order to achieve more accurate weights. Other approaches may be possible as well.

6. ACKNOWLEDGMENTS

This research is supported by the National Science Foundation under Grants ITR-0086003 and CCR-0113810, by the Sloan Software Industry Center at Carnegie Mellon, by DARPA under contract F30602-00-2-0616, and by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298. Authors would like to thank the members of the CMU Software Research Seminar for their critical feedback.

7. REFERENCES

[1] Barry Boehm. *Software Cost Estimation with COCOMO II*. Prentice Hall PTR, New Jersey: 2000.

[2] Shawn A. Butler. Security Attribute Evaluation Method. A Cost-Benefit Approach. *Proc ICSE 2002 - Int'l Conf on Software Engineering*, 2002

[3] Shawn A. Butler and Paul Fishbeck. Multi-Attribute Risk Assessment. *Symposium on Requirements Engineering for Information Security*, 2002.

[4] Shawn A. Butler and Mary Shaw. Incorporating Nontechnical Attributes in Multi-Attribute Analysis for Security. *Proc EDSER-4: Workshop on Economics-Driven Software Engineering Research*, 2002.

[5] *Proceedings of the Workshops on Economics-Driven Software Engineering Research*, EDSER-1, -2, -3, and -4. Workshops held in conjunction with the 21st through 24th ICSE's: International Conference on Software Engineering, 1999 to 2002.

[6] L. Briand, K. El-Emam, and S. Morasca. On the Application of Measurement Theory in Software Engineering. *Empirical Software Engineering*. 1(1), 1996.

[7] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous & Practical Approach*, International Thomson Computer Press, 1997.

[8] D. Garlan, D.P. Siewiorek, A. Smailagic, P. Steenkiste.. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing* 1(2), April-June, 2002.

[9] J. Flinn, M. Satyanarayanan. Energy-aware Adaptation for Mobile Applications. *Proc 17th SOSP - ACM Symposium on Operating Systems and Principles,* 1999.

[10] R. Neugebauer and D. McAuley. Congestion Prices as Feedback Signals: An Approach to QoS Management. *Proc 9th ACM SIGOPS European Workshop*, 2000.

[11] Vahe Poladian, David Garlan, and Mary Shaw. Software Selection and Configuration in Mobile Environments: A Utility-Based Approach. *Proc EDSER-4 - Workshop on Economics-Driven Software Engineering Research, 2002.*

Appendix: Quick Review of Measurement Theory

Not all measurements are created equal. More precise initial measurements enable more precise analyses and conclusions. Measure theory provides models that explain the differences and limitations.

Most members of this community are already familiar with this material, but many have forgotten the terminology. As a reminder, measure theory recognizes a number of scales for classification or measurement, ordered from less to more powerful [[6],[7]] The table summarizes the characteristics of the major scales.

Some examples of ways these scales can be abused help to show how the character of our data constrains the way we should use it:

"The temperature in Miami is 20 degrees Celsius, the temperature in Pittsburgh is 10 degrees, so it's twice as hot in Miami." *Wrong*. Celsius is an interval scale, and this kind of comparison is only valid in ratio or absolute scales. The Kelvin temperature scale is a ratio scale, so it's ok to convert to Kelvin and compare: "The temperature in Miami is 293 degrees Kelvin, the temperature in Pittsburgh is 283 degrees Kelvin, so it's 7% warmer in Miami."

"We surveyed the population for preferences on a scale of Strong Yes / Yes / OK / No / Strong No and coded the results on a 5-point scale with Strong Yes as 5 and Strong No as 1. Option A averaged 4.0, option B averaged 3.0, and option C averaged 2.0. Therefore option A dominated option B by as much as option B dominated option C." *Wrong*. The preferences are measured on an ordinal scale, and the comparison requires at least an interval scale. This sort of comparison is especially noxious when coupled with comparisons of the costs of the options. This is the kind of problem we're addressing in this paper.

Scale	*Intuition*	*Preserves*	*Example*	*Legitimate transformations*
Nominal	Simple classifica-tion, no order	Differences	Horse, dog, cat	Any one-to-one remapping
Ordinal	Ranking according to criterion	Order	Tiny, small, medium, big, huge	Any monotonic increasing remapping
Interval	Differences are meaningful	Size of difference	Temperature in Celsius or Fahrenheit	Linear remappings with offset (ax+b)
Ratio	Has a zero point	Ratios of values are meaningful	Absolute temperature (Kelvin), values in currency units	Linear remappings without offset (ax)

About the Return on Investment of Test-Driven Development

Matthias M. Müller
Fakultät für Informatik
Universität Karlsruhe, Germany
muellerm@ira.uka.de

Frank Padberg
Fakultät für Informatik
Universität Karlsruhe, Germany
padberg@ira.uka.de

Abstract

Test-driven development is one of the central techniques of Extreme Programming. However, the impact of test-driven development on the business value of a project has not been studied so far. We present an economic model for the return on investment when using test-driven development instead of the conventional development process. Two factors contribute to the return on investment of test-driven development: the productivity difference between test-driven development, and the conventional process and the ability of test-driven development to deliver higher quality code. Furthermore, we can identify when TDD breaks even with conventional development.

1 Introduction

Test-driven development (TDD) is the only way of coding in Extreme Programming (XP). TDD is also known as test-first programming: write down a simple test for each small piece of functionality before you start coding the functionality. TDD guides you through the whole life-cycle of an XP project. There is no design and no explicit testing phase. Both are replaced by automated tests which are executed continuously to ensure high program quality.

Proponents of TDD claim that it leads to faster development and to more reliable code. Both properties would make TDD superior to the conventional development style which is comprised of a detailed design, a coding phase, and test. First empirical evidence shows [2] though that the claim of faster development might not hold in general; even worse, the opposite seems to be true. Therefore, in order to assess TDD we must study the tradeoff between a (possibly) increased development cost for TDD versus a corresponding gain in code quality.

In this paper, we present an economic model for the return on investment of TDD based on the following two assumptions.

- The development with TDD is slower.
- TDD leads to higher quality code.

Other aspects of TDD, e.g. the cost of continuous testing, are not captured explicitly by our model as their impact on the monetary value of the project can not be easily separated. Thus, we consider our model as a first major step towards a full economic assessment of TDD, and it adds to the description of the economic benefit of XP projects [3].

The model compares the development cost for a conventional project with the development cost for a project that uses TDD. The investment cost is the additional effort necessary to complete the TDD project as compared to the conventional project. The life cycle benefit is captured by the difference in quality measured by the number of defects that the TDD team finds and fixes, but the conventional project does not. This defect difference is transformed into a monetary value using the additional developer effort corresponding to finding and fixing these defects in the conventional project. The concepts of the life cycle benefit and the investment cost in our context are depicted in figure 1. The upper horizontal line corresponds to the conventional project with additional quality assurance phase! The lower horizontal line corresponds to the TDD project.

Our model captures the return on investment for an experienced TDD team. Additional cost for training necessary when introducing TDD is not considered.

With this model, we can identify tradeoff lines where TDD becomes beneficial over conventional development. Interestingly, the break-even point is independent of the actual project size, the number of developers per team, and the actual developer salary; the decisive data are productivity difference, quality difference, defect removal time for one defect, working time per developer per month, and the initial defect density.

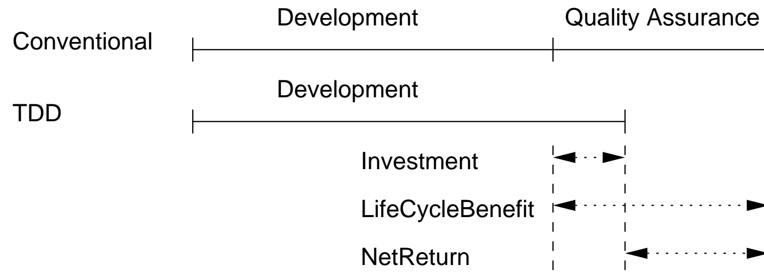


Figure 1. Overview of benefit cost ratio calculation.

2 Model

This section describes those formulas of our model which are necessary to understand the break-even analysis in Section 3. Appendix A contains a comprehensive description of the model formulas.

2.1 Return on Investment

Calculating the return on investment ROI means to add up all the benefits of the investment, subtract the cost, and then compute the ratio of the cost:

$$\mathsf{ROI} = \frac{\mathsf{LifeCycleBenefit} - \mathsf{Investment}}{\mathsf{Investment}}$$

If the investment pays off, the ROI is positive, otherwise negative. In our evaluation of TDD we focus on the benefit cost ratio BCR which is easily derived from the return on investment.

$$\begin{aligned}\mathsf{BCR} &= \frac{\mathsf{LifeCycleBenefit}}{\mathsf{Investment}} \\ &= \mathsf{ROI} + 1\end{aligned}$$

Studying the BCR instead of the ROI makes the break-even analysis much simpler, see below.

2.2 Investment Cost

We first look at the *investment* cost. For the conventional project, the development phase includes design, implementation and test. The development phase of the TDD project is comprised only of test-driven development.

As first empirical evidence suggests, we assume that the TDD project lasts longer than the conventional project. We call the ratio of the project durations the *test-speed-disadvantage* (TSD).

$$\mathsf{TSD} = \frac{\mathsf{Time}_{\mathsf{Conv}}}{\mathsf{Time}_{\mathsf{TDD}}}.$$

Since we assume that the development phase is shorter for the conventional project, the test-speed-disadvantage ranges between 0 and 1:

$$0 < \mathsf{TSD} < 1.$$

Using productivity figures to explain the difference in elapsed development time between the two kinds of project, the TDD development is $(1 - \mathsf{TSD}) \times 100\,\%$ less productive than the conventional project.

Finally, the investment is the difference between the development cost of the TDD project and the conventional project.

2.3 Life Cycle Benefit

Now, we consider the *benefit*. Each development process is characterized by a distinct *defect-removal-efficiency* (DRE). The defect-removal-efficiency denotes the percentage of defects a developer eliminates during development. Initially, a developer inserts a fixed amount of defects per thousands lines of code (*initial-defect-density*, IDD), but he eliminates $\mathsf{DRE} \times 100\,\%$ of the defects during the development process. From the increased reliability assumed for TDD, we have

$$0 < \mathsf{DRE}_{\mathsf{Conv}} < \mathsf{DRE}_{\mathsf{TDD}} < 1.$$

The additional *quality assurance* (QA) phase of the conventional project compensates for the reduced defect-removal-efficiency of the conventional process. The only purpose of the QA phase is to remove all those defects found by TDD but not by the conventional process. The amount of defects to be removed in the QA phase is mainly characterized by

$$\triangle\mathsf{DRE} = \mathsf{DRE}_{\mathsf{TDD}} - \mathsf{DRE}_{\mathsf{Conv}}.$$

The benefit of TDD is equal to the cost of the QA phase for the conventional project. The benefit depends on

the effort (measured in developer months) for repairing one line of code during QA, which is characterized by

$$\mathsf{QAEffort} = \frac{\mathsf{DRT} \times \mathsf{IDD}}{\mathsf{WT}}$$

QAEffort depends on the following:

- The defect removal time DRT. It describes the developer effort in hours for finding and removing one defect.
- The inital defect density IDD. The number of defects per line of code inserted during development.
- The working time WT. The working hours per month of a developer.

The reciprocal of QAEffort is a measure for the productivity during the QA phase.

2.4 Benefit Cost Ratio

The benefit cost ratio is the ratio of the benefit and the investment. Substituting the detailed formulas of our model given in Appendix A, the benefit cost ratio becomes

$$\mathsf{BCR} = \mathsf{QAEffort} \times \mathsf{Prod} \times \frac{\triangle\mathsf{DRE} \times \mathsf{TSD}}{(1 - \mathsf{TSD})}, \qquad (1)$$

where Prod is the productivity of the conventional project during the development phase measured in lines of code per month. Values larger than 1 for the BCR mean a monetary gain from TDD, values smaller than 1 a loss.

2.5 Break Even

Setting the benefit cost ratio equal to 1, we get a relation between the test-speed-disadvantage of TDD and the reliability gain of TDD:

$$\mathsf{TSD} = \frac{1}{\mathsf{c} \times \triangle\mathsf{DRE} + 1}, \text{ or}$$

$$\triangle\mathsf{DRE} = \frac{1 - \mathsf{TSD}}{c \times \mathsf{TSD}}$$

$$c = \mathsf{QAEffort} \times \mathsf{Prod}$$

This relation characterizes the break-even point for TDD. If the difference between the defect-removal-efficiencies is known, a lower bound for the test-speed-disadvantage can be calculated from which on the TDD project starts to be beneficial.

3 Results

3.1 Exploring the Benefit Cost Ratio

As an example, we examine the benefit cost ratio of the following scenario.

Factor	Value
DRT	10 h/defect
IDD	0.1 defects/LOC
WT	135 h/month
Prod	350 LOC/month

Let TSD and △DRE vary. Figure 2 shows the benefit cost ratio plane spanned by the test-speed-disadvantage TSD and the defect-removal-efficiency difference △DRE. Values larger than 4 are cut off.

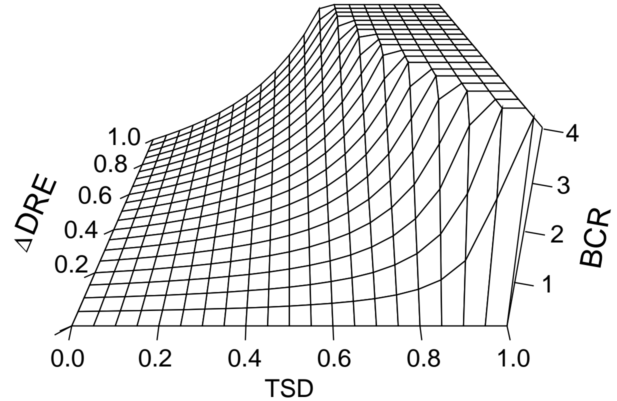


Figure 2. Benefit cost ratio dependent on TSD **and** △Eff

For large values of the test-speed-disadvantage (TSD > 0.9) the TDD project performs almost always better than the conventional project, even for a small defect-removal-efficiency difference. On the other hand, if the test-speed-disadvantage is very small (TSD < 0.2), TDD does not produce any benefit regardless how large the defect-removal-efficiency difference is.
The following table shows some benefit cost ratios for selected values of TSD and △DRE.

TSD = 0.9

△DRE	BCR
0.01	1.0 : 4.3
0.05	1.7 : 1
0.1	2.3 : 1

TSD = 0.3

△DRE	BCR
0.2	1 : 4.5
0.4	1 : 2.3
0.6	1 : 1.5
0.8	1 : 1.1
0.9	1 : 1

If the productivity of TDD is 10 % smaller than the productivity of the conventional project (left table), a 5 % better defect-reduction-efficiency suffices for TDD to break-even with the conventional process (1.7 : 1). If the productivity of TDD is much worse, say, 70 % smaller (right table), even a 80 % better defect-reduction-efficiency does not lead to a gain as compared to the conventional process (BCR is 1 : 1.1).

3.2 Break Even Analysis

With break even analysis, the ranges for TSD and △DRE can be identified where TDD is more beneficial than the conventional process. Figure 3 shows the intersection of the surface in figure 2 with the horizontal plane BCR = 1.

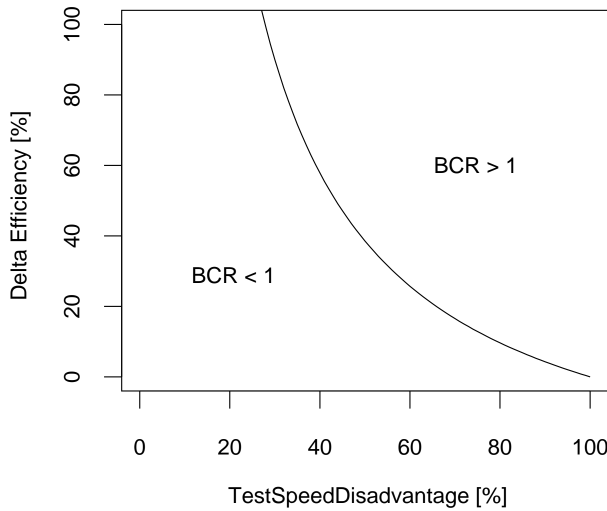


Figure 3. TSD **and** △Eff **plane for** BCR = 1

The right half of the plane (BCR > 1) corresponds to the parameter range of TSD and △DRE where TDD is superior over conventional development. Two observations can be made. First, assuming that practical values for △DRE can not be larger than 20 %, the TSD may not drop below 66 % for TDD, otherwise the TDD cost exceeds its benefit. Second, if the TSD drops below 27 %, TDD does not have a chance to provide any financial return, regardless of how large the improved defect-removal-efficiency may be.

3.3 Varying other project parameters

Figure 4 shows the different cost benefit break-even lines for varying values of the programmer productivity Prod. All other parameters are kept constant.

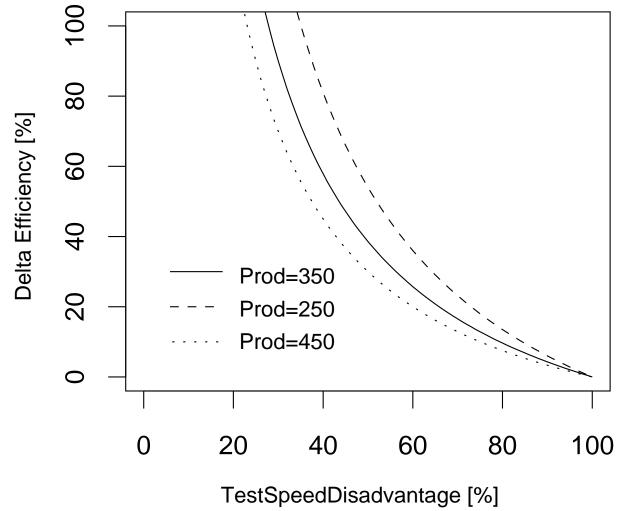


Figure 4. Break even analysis for varying values for Prod

The higher the initial productivity, the higher the chance for TDD to get a financial return over conventional development. This result is not intuitively obvious but, it can easily be derived from (1) and explained with figure 1 as follows. The higher the productivity the shorter the elapsed development time for both TDD and the conventional project. If the elapsed time for the development phase decreases, the investment (difference between both development phases) also decreases, and thus the benefit cost ratio becomes larger.

4 Conclusions

We propose an economic model for the return on investment of test-driven development. Our analysis of the break-even leads, all other parameters are kept constant, to the following conclusions:

- The return on investment of TDD depends to a large extend on the slower development of TDD and the higher quality code of TDD.
- Other factors like the effort for fixing a faulty line of code, or, the productivity of a developer using the conventional development process, have only minor impact on the return on investment of TDD.
- The calculation of the return on investment is independent of the project size, the number of developers, and the developer salary.

Our model assumes an experienced TDD team. The additional cost for training which is necessary when first introducing TDD is ignored so far.

Finally, our model strengthens the need for actual empirical figures (or ranges) for the quality advantage and the loss of productivity of TDD, in order to get a comprehensive evaluation of the cost and benefit of TDD.

References

[1] W. Humphrey. *A discipline for software engineering.* Addison-Wesley, 1997.

[2] M. Müller and O. Hagner. Experiment about test-first programming. *IEE Proceedings Software*, 149(5):131–136, Oct. 2002.

[3] M. Müller and F. Padberg. Extreme programming from an engineering economics point of view. In *International Workshop on Economics-Driven Software Engineering Research (EDSER)*, Orlando, Florida, May 2002.

[4] I. Sommerville. *Software Engineering.* Addison-Wesley, 1995.

A Appendix

A.1 Factors in the Economic Model

The following list explains the factors and their abbreviations used throughout the model.

ProductSize Size of the project in lines of code.

Prod The developer productivity measured in lines of code written per month. This figure includes design, coding, and testing. We assume that the productivity remains constant for all developers during the project. The average productivity ranges between 250 and 550 lines of code per month [4].

Salary Salary for the whole team per year. This factor is not further broken down as it turns out that our model is independent from the actual value for the salary.

NumOfDev The number of developers working in the project. We assume that this number is fixed throughout the whole project.

DRE_{TDD}/ DRE_{Conv} The defect removal efficiency describes the percentage of defects a team removes during development. We assume that TDD has a higher defect removal efficiency than the conventional process.

TSD The test-speed-disadvantage accounts for the additional effort for using TDD during development as compared to the conventional process.

DRT The defect removal time denotes the effort for finding and removing one defect during the QA phase measured in hours per defect.

IDD The number of defects per thousand lines of code inserted during development is described by the initial defect density. A typical number is 100 defects per thousands lines of code [1]. A developer reduces this number of defects according to his defect removal efficiency.

WT The working hours of a developer each month.

A.2 Model Formulas

For the conventional project, the development time is

$$\mathsf{Time}_{\mathsf{Conv}} = \frac{1}{12} \times \frac{\mathsf{ProductSize}}{\mathsf{Prod} \times \mathsf{NumOfDev}}.$$

For the TDD project, the decreased productivity has to be taken into account:

$$\mathsf{Time}_{\mathsf{TDD}} = \frac{\mathsf{Time}_{\mathsf{Conv}}}{\mathsf{TSD}}$$

During the QA phase, the conventional project has to compensate for the lower defect removal efficiency as compared to TDD:

$$\triangle \mathsf{DRE} = \mathsf{DRE}_{\mathsf{TDD}} - \mathsf{DRE}_{\mathsf{Conv}}$$

There have to be

$$\triangle \mathsf{Defect} = \mathsf{ProductSize} \times \mathsf{IDD} \times \triangle \mathsf{DRE}$$

defects removed during QA to get the same defect density as the TDD project. Thus, the time spent in the QA phase is

$$\mathsf{Time}_{\mathsf{QA}} = \frac{1}{12} \times \frac{\mathsf{DRT} \times \triangle \mathsf{Defect}}{\mathsf{WT} \times \mathsf{NumOfDev}}$$

The cost for both the TDD and the conventional project and the QA phase is

$$\mathsf{Cost}_{\mathsf{p}} = \mathsf{Time}_{\mathsf{p}} \times \mathsf{Salary}$$

where $\mathsf{p} \in \{\mathsf{TDD}, \mathsf{Conv}, \mathsf{QA}\}$.

A.3 Calculating the BCR

The benefit cost ratio is defined as the ratio between the life cycle benefit and the investment (cost):

$$\mathsf{BCR} = \frac{\mathsf{LifeCycleBenefit}}{\mathsf{Investment}}$$

Recall that the life cycle benefit equals the cost for the additional QA phase in the conventional process.

$$\mathsf{BCR} = \frac{\mathsf{Cost}_{\mathsf{QA}}}{\mathsf{Cost}_{\mathsf{TDD}} - \mathsf{Cost}_{\mathsf{Conv}}}$$

The factor Salary can be canceled out. Hence,

$$\begin{aligned}\mathsf{BCR} &= \frac{\mathsf{Time}_{\mathsf{QA}}}{\mathsf{Time}_{\mathsf{TDD}} - \mathsf{Time}_{\mathsf{Conv}}} \\ &= \frac{\mathsf{Time}_{\mathsf{QA}}}{\mathsf{Time}_{\mathsf{Conv}} \times (\frac{1}{\mathsf{TSD}} - 1)}.\end{aligned}$$

Further canceling of the factors 12, NumOfDev, and ProductSize leads to

$$\begin{aligned}\mathsf{BCR} &= \frac{\mathsf{DRT} \times \mathsf{IDD} \times \mathsf{Prod} \times \triangle\mathsf{DRE}}{\mathsf{WT} \times (\frac{1}{\mathsf{TSD}} - 1)} \\ &= \frac{\mathsf{DRT} \times \mathsf{IDD} \times \mathsf{Prod} \times \triangle\mathsf{DRE} \times \mathsf{TSD}}{\mathsf{WT} \times (1 - \mathsf{TSD})} \\ &= \frac{\mathsf{DRT} \times \mathsf{IDD}}{\mathsf{WT}} \times \mathsf{Prod} \times \frac{\triangle\mathsf{DRE} \times \mathsf{TSD}}{(1 - \mathsf{TSD})} \\ &= \mathsf{QAEffort} \times \mathsf{Prod} \times \frac{\triangle\mathsf{DRE} \times \mathsf{TSD}}{(1 - \mathsf{TSD})}.\end{aligned}$$

Selecting a defect prediction model for maintenance resource planning and software insurance

Paul Luo Li
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh PA 15213
1-412-268-3043
Paul.Li@cs.cmu.edu

Mary Shaw
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh PA 15213
1-412-268-2589
Mary.Shaw@cs.cmu.edu

Jim Herbsleb
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh PA 15213
1-412-268-8933
jherbsleb@acm.org

ABSTRACT
Better post-release defect prediction models could lead to better maintenance resource allocation and potentially a software insurance system. We examine a special class of software systems and analyze the ability of currently-available defect prediction models to estimate user-reported defects for this class of software, widely-used and multi-release commercial software systems. We survey currently available models and analyze their applicability to an example system. We identify the ways in which current models fall short of addressing the needs for maintenance effort planning and software insurance.

General Terms
Management, measurement, economics, reliability, software maintenance.

Keywords
User-reported defect estimation, empirical maintenance models, software defect models.

1. INTRODUCTION
Hedging the risks associated with owning software is an important economic issue for both producers and consumers of software products. Risk aversion on the part of software consumers has created a market for software maintenance contracts and has opened the opportunity for software insurance. Software producers must manage the uncertainties of providing and marketing support that balance the risks and benefits they present to their customers. A good defect estimation model is an important first step toward pricing maintenance contracts and insurance and toward predicting support costs such as maintenance staffing. We compare models for estimating user-reported defects in the particular setting of widely-used, multi-release commercial software systems. For a defect prediction model to be useful in this setting the model has to account for aspects of the software system and its operating environment that could cause large variances in predictions.

Widely-used and multi-release commercial software systems include, for example, operating systems, servers, web-browsers, and office suites. These software systems are not intended to be mission critical, and it is generally recognized that they contain defects. However, they are used in situations where they are essential to the business interests of the user. The risk aversion characteristics of these users create a market for software maintenance support and software insurance.

Maintenance contracts provide assurance that a reported defect will be resolved within a contracted time and/or in a certain manner, while software insurance would compensate the user for the losses associated with a defect [8]. Although maintenance planners are interested in the costs associated with repairing a defect and insurers are interested in the damages that a defect can cause, both are interested in the number of defects that are likely to occur.

Currently, planning for maintenance activities is mostly ad-hoc, and business insurance does not separately consider software failures. Defect estimation models would improve the quality of information available for these decisions. Widely-used multi release commercial systems are usually developed and tracked with processes that gather historical information that can be used by a defect estimation model. A good defect prediction model should use this information to accurately model defect occurrences in the field and to account for aspects of the system that cause variation in defect rates between releases.

This paper differs from papers that evaluate the effectiveness of reliability models, e.g. [9] in that we focus on user-reported defects, which are defect occurrences in the field, for widely-used multi-release commercial software systems. Papers evaluating reliability models focused on defects found during testing. In addition, they concentrated on custom developed and one-off systems.

While we can only speculate on the value of a software insurance system, the motivation for better maintenance resource planning is clear. A market for software maintenance already exists. Allocating the right amount of resource can provide a competitive advantage for software producing organizations such as making better pricing decisions and lowering staffing costs, while maintaining the level of service.

Section 2 presents an example system and Section 3 describes the complexities associated with estimating defects for such a system. Section 4 surveys the currently-available defect estimation models. Section 5 analyzes how well the models address concerns

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EDSER '03,

raised in Section 3. Section 6 make presents suggestions for further research and conclusions

2. EXAMPLE SYSTEM

We will use a concrete example to help us reason about the defect estimation problem. Consider a software system, SystemX.

SystemX is an operating system with the usual functionalities associated with such a system. Specification of these functionalities does not come from one customer, but rather has evolved through time by adding capability, in response to market demands. Customers of SystemX vary, from application development organizations that develop software for SystemX, to small businesses that depend on SystemX to run applications coordinating suppliers, and private users that use SystemX for leisure.

SystemX is developed by an organization whose business goal is to maximize profit. This organization has a clearly defined, repeatable development process. The organization's profit maximizing strategy is to keep customers happy and loyal by implementing advanced features and by providing customer service, all at minimum costs. In addition to developing SystemX this organization also provides customer service support. This customer service division answers and tracks user/customer questions that arrive through various channels. Incoming inquiries include a combination of questions that reflect defects in the software and questions that are requests for information or symptoms of user confusion. The support organization records a defect when it determines that a customer question is a code related problem. The recorded defects then become the responsibility of the maintenance organization, which could be the same as the development organization, to resolve.

We will use SystemX to compare and evaluate defect estimation models.

3. THE OPERATIONAL SETTING

SystemX is a widely-used multi-release commercial system. This means that successive releases of SystemX implement changes as dictated by the market. Each release will have many instances, installed with different configurations. A new release of SystemX repairs some defects and, as history indicates, introduces new defects. The decisions to adopt a new release and to report defects are made by the customer. This section examines the implications of these conditions. Each condition introduces variation in defect occurrence characteristics, which includes both the number of defects and the occurrence rates. Effects of these variations must be understood in order to predict defect rates accurately.

3.1 User-reported defects

This section addresses user-reported defects, the occurrence of interest. We follow the definition set forth in [6] and [7]: *a user-reported defect is a mistake at the coding level, which manifests in a deviation from the expected behavior that is reported by a user*. We take defects to mean user-reported defects in this paper.

After SystemX is released, the meaningful measure of perceived quality in the commercial setting is defects as reported by the user [6]. The number of users using the system directly causes variations in user-reported defect occurrence characteristics.

We count the number of defects associated with actual code defects. Many users might report the same defect or different symptoms of the same defect. For our purposes, a defect is recorded the first time it is reported, and subsequent reports of the same defect are ignored or attached to the prior defect report. In addition, not all calls to a call center reflect defects; some, for example, reflect misunderstanding by the user. Defects are recorded only when the support organization can associate them with code defects.

3.2 Widely-used systems

This section addresses characteristics of a system that is used in different configurations and for different purposes. Widely-used systems have many instances in operation running with different hardware and software configurations, as is often the case, for example when a system is sold to many different customers.

Differences in configuration for SystemX include other systems, such as databases, middleware, and applications, hardware components, such as processors and storage devices, and/or in purpose of use, such as for business critical infrastructure or for leisure.

Our description of "widely-used" is not to be confused with "widely-distributed", which refers to systems that are implemented with many distributed communicating components. It is also not to be confused with a system that has simply many users. Although System X could have many users, we are concerned here with systems that have multiple instances.

Widely-used characteristic causes variation between defect occurrences characteristics during testing and in the field. The number of different possible configurations of System X makes it infeasible to consider and test all possible configurations during development. Moreover, the large number of different types of uses makes it impractical to discover all usage scenarios.

3.3 Multi-release systems

This section discusses the characteristics of an evolving software system that changes to meet developing trends by incorporating the latest innovations. We characterize multi-release systems as having successive releases that incorporate incremental changes and improvements.

Depending on the features introduced in a release and a customer's usage characteristics, some customers might choose to adopt the latest release of SystemX right away, other might delay adoption, and others might choose to stay with older releases. The development process is also undergoing constant improvement.

Different user adoption strategies cause variation in defect occurrence rates. A development organization might always adopt the latest release of SystemX to assure continued compatibility with its own products. A small business might delay adoption. Leisurely users might never adopt the latest release of SystemX unless they wish to utilize an important feature. Defects associated with particular hardware and software configurations might or might not be discovered for a given release.

Differences in the complexity and extent of the changes introduced by a release can cause variations in defect occurrence characteristics. Releases that introduce major modifications or new technologies, which due to their critical nature and/or complexity might have significant impacts on the development

process, the resulting software system, and user adoption and usage characteristics.

Multi-release systems could also experience variations in defect occurrence characteristics resulting from changes in the development process. Development organizations are constantly trying to improve their processes. The improvements are incremental, but these process changes can affect the number of residual defects in software system.

3.4 Commercial systems

This section presents characteristics of a software system developed by an organization in which the release schedule is driven by market forces.

SystemX's quality is one among many forces guiding development and release decisions. Other considerations include demands of the market, timetables set forth by management, and resource constraints of the organization. Business objectives might make it necessary to make trade offs between time to market and quality. Commercial systems will have variations in defect occurrence characteristics, since the software must sometimes be release with substantial number of defects still in the system.

3.5 Summary

The problem is multi-dimensional. Table 1 presents a summary of the concerns that the defect model must address. This serves as a reference when thinking about possible solutions

Table 1. Summary of concerns.

Characteristics	Sources of variation in user-reported defects
User-reported defects	-Number of users
Widely-used system	-Software configurations -Hardware configurations -Usage characteristics
Multi-release system	-Adoption characteristics -Changes introduced -Development process changes
Commercial system	-Number of residual defects at time of release

4. EXISTING MODELS

For simplicity, we focus only on the defect occurrence rates and treat all defects equally. We recognize that different defects will have different associated costs, such as their damage to the customer associated or cost of their repair to the development organization. We hope to address these concerns in later works. This section examines currently available prediction models focusing on aspects of the models that we will use for our analysis in section 5.

4.1 Parameterized mathematical models

Parameterized mathematical models have a fixed form with parameters that are tuned using data from the current release. Since this class of models was originally developed for customized one-off systems, they assumed that the intended usage environment could be simulated accurately, thus they were able to extend defect occurrence rates found during testing into the field [10]. These models fit their parameters using defect counts and defect occurrence times starting from time zero up to time t. From the tuned model, the defect occurrence rates and in some cases, the numbers of remaining defects after time t are estimated.

β and α are model parameters, which have different meaning and values for each model. The total number of defects, N, is assumed to be given in some models, like the Weibull [6] and variants of the exponential [11]. Since N cannot always be accurately estimated, other models include the total number of defects as a model parameter to be estimated. The model parameters are tuned using data from the current release up to some time t, through mathematical methods like maximum likelihood or least squares. Due to the different model forms, each model will produce a different prediction for the number of defect and the rate of defect occurrences after time t. A summary of commonly applied models is in Table 2.

Table 2. Summary of parameterized mathematical models.

Model Group	Model Form	Input	Output
Finite Exponential	$\mu(t)= \alpha\ (1–\exp(-\beta\ t\ ))$	Defect occurrence counts and times, up to time t	Defect rates and defect counts after time t
Infinite Geometric	$\mu(t)=(1/\beta)\ln((\alpha\beta\exp(\beta))t + 1)$	Defect occurrence counts and times, up to time t	Defects rates and defect counts after time t
Infinite Logarithmic	$\mu(t)=\ln(\alpha\beta t+1)/\beta$	Defect occurrence counts and times, up to time t	Defect rates and defect counts after time t
Finite Weibull	$\mu(t)=N(1–\exp(-\beta t^a))$	Defect occurrence counts and times, up to time t	Defects rates and defect counts after time t
Finite Gamma	$\mu(t)=\alpha(1–(1+\beta t)\exp(-\beta t))$	Defect occurrence counts and times, up to time t	Defect rates and defect counts after time t

Table 3 gives a summary of implied assumption and research and applications. Due to the different model forms and parameterizations, each model in Table 2 has a different implied assumption regarding the hazard rate, the rate at which the defects remaining in the system are uncovered at time t. The hazard rate will be important when we analyze and compare the ability of parameterized models to predict defect for SystemX in Section 5.1. Detailed explanations of the concepts presented in this section can be found in [8].

Table 3. Summary of assumptions

Model Group	Hazard Rate	Defect Total	Research
Finite Exponential	Constant	Finite	Jelinski and Moranda, 1972 Musa, 1979
Infinite Geometric	Decreasing	Infinite	Moranda, 1979
Infinite Logarithmic	Decreasing	Infinite	Musa-Okumoto, 1984
Finite Weibull	Increasing	Finite	Kenny, 1993
Finite Gamma	Increasing	Finite	Yamada, Ohba, and Osaki 1983

4.2 Bayesian methods

In contrast to parameterized mathematical models, which require data from the current release up to time t to estimate defects after time t, Bayesian models can take prior information, such as information from previous releases, to estimate defects before the release of the current release. An important feature of the Bayesian framework is its ability to use information as it become available to improve and adjust estimates. The Bayesian model is the same as its non-Bayesian cousin. The difference is that the parameters in the Bayesian models are treated as variables as well.

Bayesian model need prior distributions for each parameter in the model [10]. There are various ways to construct the prior such as creating a distribution of possible parameter values using fitted model parameters of previous releases or eliciting possible values from experts, and in the worst case where there no prior information is available, it is possible to mathematically construct a non-informative prior that offers no insight into the system but which makes it possible to generate a prediction. Once data becomes available from the field, the model parameters can be adjusted from the prior to better fit the data. At any time t it is possible to have modified predictions as result of re-estimating model parameters using field defect information.

Table 4 gives a summary of the inputs, outputs, and research on Bayesian models. We analyze the ability of Bayesian models to model defects for SystemX in Section 5.2.

Table 4. Summary of Bayesian models.

Model Group	Research	Input	Output
Bayesian Exponential	Littlewood 1987	Historical defect occurrence information and available defect occurrence information	Defects occurrence rates and counts
Bayesian Gamma	Littlewood-Verrall 1980	Historical defect occurrence information and available defect occurrence information	Defect occurrence rates and counts
Bayesian Geometric	Liu 1987	Historical defect occurrence information and available defect occurrence information	Defect occurrence rates and counts

4.3 Product/process models

Process and product models estimate the number of defects in the current release using estimates of the size of the product or deviation from the previous release and known organizational information.

Research in this area traces back to Belady and Lehman's work on system growth and system structure degradation in successive system releases [7]. Their research serves as the basis for several methods, which measure the amount of change in a system and predict the number of defects that result from the changes. Some notable results are summarized in Table 5. We analyze the ability of product/process models to predict defects for SystemX in Section 5.3.

Table 5. Product/process models.

Model Group	Research	Input	Output
Using lines of code with organizational adjustments	Rome Laboratory, 1992 COQUALMO, 1999	Lines of code estimate and process effectiveness estimates	Total number of residual defects
Using software change information	Mockus et al. 2003, Graves et al. 2000	Change management information	Number of residual defects (effort)

5. ANALYSIS OF EXISTING SOLUTIONS

Given the various models available we now examine how well they address the concerns involved with modeling user-reported defect for SystemX.

5.1 Ability of the model to fit the data

The first step is to choose a model that accurately describes defect occurrence data. This will also involve picking a model from Table 3 that fits the conditions of SystemX.

Section 4.1's survey of parameterized mathematical models in practice shows that three categories of models are widely applied: finite exponential, infinite, and finite Weibull and Gamma.

Finite exponential models like those that used by Jelinski and Moranda [11]and Musa [14], (First row in Table 3.) assumes that the number of defects in the system is finite and that the hazard rate, the rate at which defects are uncovered at time t, is constant. These assumptions do not fit SystemX since the repair process can introduce defects, and the rate at which the varying numbers of customers are uncovering defects is non-constant

Infinite models like those used by Musa-Okumoto [15] and Moranda [12], (The third and forth rows in Table 3.) assumes that the number of defects in the system is infinite due to a decreasing hazard rate. Although this class of models correctly assumes that defects are being re-introduced into the system. It also assumes that the rate at which defects are being uncovered is always decreasing, but since the number of users of SystemX increase over time, the rate at which defects in the system is uncovered should be increasing.

The final class of parameterized mathematical models we examine is Weibull and Gamma models like those used by Kenny [6],

Yamada and Ohba [17]. (Row five and six in Table 3) Like exponential models, the number of defects is assumed finite. However, unlike both exponential and infinite models, this class accurately describes the migration characteristic. This class of models can describe the ramping up process due to migration seen in commercial systems, using the expressive power offered by the models' parameters.

The Weibull model has been shown to be able to describe the defect occurrence patterns seen in widely–used commercial systems[6]. We not aware of any other papers that attempts to fit other model to defect occurrence data from a system with our characteristics.

5.2 Ability to incorporate prior information

The main drawback of parameterized mathematical models is the need to wait until defect data from the current release is available. However, if we want to plan for maintenance, we need to make defect predictions before release to the field. We cannot use defects occurrences during testing to tune the model before release, because due to the widely-used condition of SystemX defect rates seen during testing will not reflect occurrence characteristics seen in the field. However, we can use the multi-release characteristic of SystemX to help us to over come this difficulty.

Defect information from previous releases can be used by Bayesian models in Table 4 to construct prior distribution for their model parameters before field defect information from the current release is available. The resulting model will be able to estimate defect occurrence rates and defect occurrence counts for the current release before release to the field. The Bayesian process also uses defect information as it becomes available to update the parameter estimates. At any stage during the Bayesian process, the data collected up to that point can be assumed to be prior information and all the parameters can be re-estimated. Intuitively this means that whenever defects are reported and repaired for SystemX, a Bayesian model can re-computed the number of defects in the system to reflect the fact that repairs could introduce new defect into the system.

The Bayesian Gamma model used by Littlewood should be able to describe the characteristics of a commercial system, but the total number of defects for the current release is a model parameter that is estimated using defect occurrence data from previous release. Intuitively this does not make sense because the number of defects seen in any given release of SystemX is a result of the amount change in the current release and the status of development organization, not a result of the field defect occurrence characteristics of previous releases.

There is not at present a Bayesian Weibull model. However, we feel that the machinery available through Bayesian methods and the ability of the Bayesian framework to update estimates be combined with the Weibull model.

5.3 Ability to account for product/production differences

Up to this point, we have not considered variations due to the product or the organization that developed the product. Since information about the changes a release implements and the development effort is available before release, product/process models use this information to estimate defect occurences.

COQUALMO, the extension into the software quality domain of Bohem's work with COCOMO estimates the number of residual defects in a system using lines of code and process drivers related to the characteristics of the particular organization concerning various defect introduction stages and defect removal methods. These drivers are derived from expert estimates at first, then tuned to match actual defect counts.[3] COQUALMO assumes that the number of defects at release time is the result of two processes during development, defect introduction and defect removal. The drawbacks of COQUALMO are that it estimates only the total number of defects and lines of code. Lines of code alone are not very good estimators of complexity. In addition, maintenance planning and software insurance for SystemX require defect occurrence estimations at any time t, not merely the total number of defect.

A promising alternative is to use change data as captured by change management systems. In addition to lines of change data, this data includes number of changes, feature requests, and developer who made the change. Mockus uses the non-intrusively recorded feature/fix requests, which can be seen as a change at a high level, along with lines of code and number of changes to estimate efforts [13]. The method uses a delay factor to determine when the estimated effort will occur and a ratio to describe the defect repair effort relative to development effort. However, this method does not adoption characteristics, which could vary between releases depending on the features implemented. In addition, this method does not take in to consideration process changes.

Currently, there does not exist a model that incorporates research in parameterized mathematical models, Bayesian models, and process and product models.

6. CONCLUSION AND THE ROAD AHEAD

Widely used and multi-release commercial software systems are an important class of software systems. Reliance on these systems has created a need for maintenance contracts to assure defect resolution and for software insurance to provide compensation in the event of a defect. Both techniques require a defect occurrence model, but no model is currently available that can account for all possible causes of variation. We find existing solutions lacking in their considerations of organizational changes and user adoption characteristics.

Current modeling techniques do not adequately consider organizational changes. Process changes and tool changes can affect defect occurrence characteristics as well as influential personnel changes, such as a lead designer [4]. We think evaluations of the development process from developers who have worked on a release and can better evaluate the process and product variations as shown by postmortems and experience reports. However, currently no model attempts to quickly capture and use this information.

Current approaches to defect modeling do not account for differences in customer adoption characteristics. We feel these differences are important because in the commercial setting the customers are the ones who exercise the code to detect defects. The number of customers and the configurations they use has a direct effect on the number of defects found. Customer support divisions and sales departments could provide the needed information.

Research in customer adoption is not to be confused with customer usage profiles and defect profiles as researched by Wyuker [16] and Bassin and Santhanam [1]. Customer usage profiles describe the frequency with which a piece of code is exercised. Defect profiles describe the type and order in which defects occur. While both are interesting, we are interested in rates of adoption: the number and type of customers who adopt a release and the time when they adopt.

Improved defect estimation model in our setting needs to be validated by showing that it is generally applicable to a wide range of commercial software systems like operating system, servers, office suites, or web browsers. In addition, it has be an improvement over existing solutions such as ones listed in Table 6.

Table 6. Summary of existing solutions

Model Grouping	Best solution	Desirable features	Main Drawbacks
Parameterized mathematical models	Weibull model	-Describes the ramping up pattern	-Assumes no defects are introduced into the system -Descriptive, not predictive thus cannot be used for predict before release.
Bayesian models	Bayesian Gamma model	-Predictive by using priors -Parameters (like defect totals) can change after release	-Does not consider release differences
Process/ Product models	Product change model	-Describes release differences though change data	-Does not consider organizational changes -Does not consider adoption characteristics

Once a defect occurrence estimation model is available, can we move towards structured maintenance planning and begin to evaluate the possibility of software insurance.

7. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under Grand CCR-0086003, by the Carnegie Mellon Sloan Software Center, by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298.

The authors would like to thank Peter Santhanam and Bonnie Ray of IBM Research for their contribution to this work.

The authors would like to thank Audris Mockus for his insights, and Vahe Poladian for his valuable input.

8. REFERENCES

[1] Kathryn Bassin, Peter Santhanam. “Use of Software Triggers to Evaluate Software Process Effectiveness and Capture Customer Usage Profile.” Symposium on Software Reliability Engineering, 1997.

[2] Ram Chillarage, Shriram Biyani, Jeanette Rosenthal. “Measurement of Failure Rate in Widely Distributed Software.”

[3] Sunita Chulani. “COQUALMO9 COnstructive QUAlity MOdel) A Software Defect Density Prediction Model.” Project Control for Software Quality. Editors: Kusters, Cowderoy, Heemstra, and van Veenendaal. Shaker Publishing, 1999.

[4] B, Curtis. H. Krasner, et al. "A field study of the software design process for large systems." Communications of the ACM. 31(11): 1268-1287. 1988.

[5] A.L. Goel, K. Okumoto “Time-Dependent Error –Detection Rate Model for Software and Other Performance Measures.” IEEE Transaction on Reliability,

[6] Garrison Q. Kenny. “Estimating defects in commercial software during operational use.” IEEE 1993.

[7] M.M. Lehman, L.A. Belady. *Program Evolution: Process of Evolution Change*. Academic Press Inc. 1985.

[8] Paul Luo Li, Mary Shaw, Kevin Stolarick, and Kurt Wallnau. “The Potential for Synergy Between Certification and Insurance” Special edition of ACM SIGSOFT from the IWRE in conjunction (ICSR7), April 2002.

[9] Yahswant Malaiya, Nachimuthu Karunanithi, Pradeep Verma. “Predictability of Software-Reliability Models.” IEEE Transaction on Reliability Vol 41. No 4. 1992.

[10] Michael Lyu. *Software Reliability Engineering*. McGraw-Hill, 1996.

[11] P.L Moranda, Z. Jelinski. Final Report on Software Reliability Study. McDonnell Douglas Astronautics Company, MADC Report Number 62921. 1972.

[12] P.L. Moranda. “Event-Altered Rate Models for General Reliability Analysis.” IEEE Transaction on Reliability. Vol R-28,1979 pp376-381.

[13] Audris Mockus, David M. Weiss, Ping Zhang "Understanding and Predicting Effort in Software Projects." ICSE 2003 Proceedings

[14] J.D. Musa. “Validity of Execution-Time Theory of Software Reliability.” IEEE Transactions of Reliability, Vol R-28, 1979. pp181-191.

[15] J.D. Musa, K. Okumoto. “A Logarithmic Poisson Execution Time Model for Software Reliability Measurement.” Proceedings Seventh International Conference on Software Engineering, 1984.pp 230-238.

[16] Elaine Weyuker, S. Rapps. "Data Flow Analysis Techniques For Test Data Selection." Proceedings of the 6th International Conference on Software Engineering (ICSE), Tokyo, Japan, September 1982, pp.272-278.

[17] S. Yamanda, M. Ohba, S. Osaki. “S-Shaped Reliability Growth Modeling for Software Error Detection.” IEEE Transaction on Reliability, Vol R-32. 1983. pp 475-478.

ArchOptions: A Real Options-Based Model for Predicting the Stability of Software Architectures

Rami Bahsoon and Wolfgang Emmerich
Dept. of Computer Science
University College London
Gower Street, WC1E 6BT, London, UK
{r.bahsoon, w.emmerich}@cs.ucl.ac.uk

Abstract

Architectural stability refers to the extent an architecture is flexible to endure evolutionary changes in stakeholders' requirements and the environment, while leaving the architecture intact. We assume that the primary goal of a software architecture is to guide the system's evolution. We contribute to a novel model that exploits options theory to predict architectural stability. The model is predictive: it provides "insights" on the evolution of the software system based on valuing the extent an architecture can endure a set of likely evolutionary changes. The model builds on Black and Scholes financial options theory (Noble Prize winning) to value such extent. We show how we have derived the model: the analogy and assumptions made to reach the model, its formulation, and possible interpretations. We refer to this model as ArchOptions.

Keywords. *Architectural economics; economic-driven software engineering research; relationship between requirements and software architecture; real options theory; requirements evolution.*

1. Introduction

Architectural stability is a concept that bridges the gaps between research in requirements engineering, software architecture, and software economics of complex evolutionary systems. The informal concept of architectural stability refers to the extent an architecture is *flexible* to endure evolutionary changes in stakeholders' requirements and the environment, while leaving the architecture intact.

In an *evolutionary* context, there is a pressing need for stable software architectures. In this context, requirements are generally *volatile*; they are *likely* to change and evolve over time. The change is inevitable as it reflects changes in stakeholders' needs and the environment in which the software system works. The tension between an unstable architecture and the volatile requirements may entail large and disruptive changes for the requirements to be accommodated. The change may "break" the architecture necessitating changes to the architectural structure (e.g. changes to components and interfaces), architectural topology (e.g. architectural *style*, where a style is a generic description of a software architecture), or even changes to the underlying architectural infrastructure (e.g. middleware). It may be expensive and difficult to change the architecture as requirements evolve [11]. Consequently, failing to accommodate the change leads ultimately to the degradation of the usefulness of the system.

From an *economic* perspective, the volatility of requirements may be regarded as a major source of *uncertainty* that confront an architecture during its evolution. It places the investment in a particular architecture at *risk*, where a risk is an event with potentially undesirable outcome whose occurrence has some known probability distribution. To cope with uncertainties, incomplete knowledge in an evolutionary context, and mitigate risks in the investment, there is a critical need for predicting the stability of software architectures. Such prediction is necessary for valuing the long-term investment in a particular architecture; analysing trade-offs between two or more candidate software architectures for stability; analysing the strategic position of the enterprise- if the enterprise is highly centred on the software architecture (as it is the case in web-based service providers companies e.g. amzon.com); and validating the architecture for evolution.

A stable software architecture adds to the software system and to the enterprise owing the architecture a *value*. The added value is attributed to *flexibility* and the *options* that flexibility creates over the evolutionary periods of the software system. An option provides the right to make an investment in the future, without a symmetric obligation to make that investment [6, 19]. The added value under the stability context is strategic in

essence and may not be immediate. It takes the form of (i) accumulated savings through enduring the change without "breaking" the architecture; (ii) supporting reuse; (iii) enhancing the opportunities for strategic "growth" (e.g. regarding an architecture as an asset and instantiating the asset to support new market products); and (iv) giving the enterprise a competitive advantage by banking the stable architecture like any other capitalized asset.

The major idea of this work is that the *flexibility* of an architecture to endure changes in stakeholders' requirements and the environment has a value that can *assist* in predicting the stability of software architectures. More specifically, flexibility adds to the architecture values in the form of *real option* [15, 16]- that give the right but not a symmetric obligation- to evolve the software system and enhance the opportunities for strategic growth by making future follow-on investments (e.g. case of reuse, exploring new markets, expanding the range of services while leaving the architecture intact). As flexibility has a value under uncertainty [1, 8, 9, 17]; the value of these options lies in the enhanced flexibility to cope with uncertainty (i.e. the evolutionary changes). The importance of the idea cannot be overemphasized: it gives the architects/stakeholders an ability to reason about a crucial but previously intangible source of value and to factor it in the prediction of an architecture for stability.

This paper contributes to a novel model for predicting the stability of software architectures using real options theory [15, 16]. We assume that the software architecture's primary goal is to guide the system's evolution. The model is predictive: it provides "insights" on the evolution of the software system based on valuing the *extent* an architecture can endure a set of *likely* evolutionary changes. It uses *value-based* reasoning to prediction and builds on Black and Scholes [5] financial options theory (Noble Prize winning) to value such extent. We refer to this model as ArchOptions.

The paper is further structured as follows. Section 2 briefly discusses why we have taken a real options approach to prediction. Section 3 supplies background on Black & Scholes options pricing technique. Section 4 shows how we have derived the model to predict the stability of software architectures: it presents the analogy, assumptions, approach, and interpretation. Section 5 reviews related work. Section 6 concludes.

2. Why real options?

We view stability as a *strategic* architectural quality that adds to the architecture values in the form of *growth options.* A growth option is a real option to expand with *strategic* importance [16]. Growth options are common in all infrastructure-based (as it is the case of software architectures) or strategic industries, and especially in industries with multiple-product generations or applications [18, 22]. As many early investments can be seen as prerequisites or links in chain of interrelated projects [16], *growth options* set the path for the future opportunities [18, 22]. In the architectural context, future growth opportunities are very much linked to the flexibility of the architecture to endure the likely future changes while leaving the architecture intact, and henceforth to the stability of software architecture. Hence, architectural stability enhances the upside potentials of the architecture, for flexibility sets the path for future follow-on investments and strategic growth (e.g. case of reuse, exploring new markets, expanding the range of services while leaving the architecture intact). The follow-up investments are generally triggered by the inevitable future changes in stakeholders' requirements and the environment. Since the future changes are generally unanticipated, the value of the growth options lies in the enhanced flexibility of the architecture to cope with uncertainty; otherwise, the change may be too expensive to pursue and opportunities may be lost.

Hence, to predict the stability of software architectures taking a value-based reasoning approach, we need a technique that is suitable for strategic and long-term valuation, counts for flexibility, and makes the value of the options created by flexibility tangible (as a way to make the value of stability tangible).

Classical financial techniques, such as Discounted Cash Flow (DCF) analysis and Net Present Value (NPV), *fall short* in dealing with flexibility and uncertainty [18, 22]. The main problem with these techniques is that they are best valid when valuing an ongoing business or an immediate investment. However, in the case of valuing the stability of software architectures in the face of evolutionary changes, the nature of the investment is long-term and strategic. For example, assume that an investment in an architecture appears to be unattractive (e.g. case of negative NPV) at the first instance: unless the enterprise makes the initial investment, subsequent generations or other applications will not even be feasible. The value of the investment, thus, may derive not only from the direct measurable cash flows of the investment, but also from the ability of an architecture to unlock future growth opportunities (e.g. case of reuse, exploring new markets, expanding the range of services while leaving the architecture intact).

Among alternative techniques that are available to make the value of flexibility tangible is real options theory. Real options theory [15, 16] was developed to address the inability of these traditional budgeting techniques to address strategic value. An option is an asset that provides it owner the right without a symmetric obligation to make an investment decision under given terms for a period of time into the future ending with an

expiration date [18, 22]. If conditions favourable to investing arise, the owner can exercise the option by investing the strike price defined by an option. A *call option* gives the right to acquire an asset of uncertain future value for the strike price. A *put option* provides the right to sell an asset at that price. A *European option* can be only exercised on the expiration date of the option. A *real option* is an option on non-financial (real) asset, such as a parcel of land or a new product design.

3. Options pricing using Black & Scholes: background

The best-known financial option pricing method (the seminal work in the field) is that of Black and Scholes [5], which is a solution to a *stochastic* calculus problem. Any variable whose value changes over time in an uncertain way is said to follow a stochastic process.

Under the Black and Scholes model, five parameters are needed to determine the option price. These are the current stock price (S), the strike price (X), the time to expiration (T), the volatility of the stock price (σ), and the free-risk interest rate(r).

The price of the stock option is a function of the stochastic variables underlying stock's price and time. The strike price (X) is the price for which the holder may exercise a contract for the purchase/sale of the underlying stock; also referred to as the *exercise price*. The current stock price (S) if exercised at some time in the future, the payoff from a call option will be the amount by which the stock price exceeds the strike price. Call options, therefore, become more valuable as the stock price increase and less valuable as the strike price increases. The volatility of the stock price (σ) is a statistical measure of the stock price fluctuation over a specific period of time; it is a measure of how uncertain we are about the future of the stock price movements. The value of a call option on an asset depends on the value of the asset itself and the cost of exercising the option.

The expected value of a *European call option* is given by E [max (S_t- X, 0)], where E denotes the expected value of a European call option and S_t denotes the stock price at time t.

The European call option price, C, is the value discounted at the risk-free rate of interest. It calculates to equation (*1*).

$$C = e^{-r\,(T-t)} E\,[\max(S_t - X, 0)] \qquad (1)$$

In a risk-neutral world, ln S_t has the following probability distribution given by *(2)*,

$$\ln S_t \sim \phi\,[\ln S + (r-\sigma^2/2)(T-t),\ \sigma(T-t)^{1/2}] \qquad (2)$$

Where ϕ [m, s] denotes a normal distribution with mean m, and standard deviation S. Evaluating the right-hand side of *(1)*- in application of integral calculus-results in Black and Scholes valuation of a call option.

$$C = S\,N(d_1) - Xe^{-r\,(T-t)}\,N(d_2) \qquad (3)$$

Where,

$$d_1 = \frac{\ln(S/X) + (r + \sigma^2/2)(T-t)}{\sigma(T-t)^{1/2}}$$

$$d_2 = \frac{\ln(S/X) + (r - \sigma^2/2)(T-t)}{\sigma(T-t)^{1/2}} = d_1 - \sigma(T-t)^{1/2}$$

$N(x)$ is the cumulative probability distribution function for a standardized normal variable (i.e., it is the probability that such a variable will be less than x). Interested reader may refer to [12] for a more detailed derivation.

4. Exploiting options theory to predict architectural stability

We derive a model to predict the stability of software architectures from equation *(1)*. We draw the analogy and make assumptions. For every likely evolutionary change, we construct a call option to value the flexibility of the architecture to accommodate the likely change(s)- as a way to make the value of stability tangible. We provide an interpretation of the model in the context of stability.

4.1. Analogy and assumptions

A major insight behind real options theory is that flexibility in real asset is analogous to financial options: investing in flexibility is seen as buying options and exploiting flexibility is seen as exercising them [20]. Having set flexibility as an option problem, the challenge becomes valuing flexibility: we derive a model from *(1)* and exploit [5] to valuation. We map the economic characteristics of the architecture (under development or evolution) onto the parameters of the option model *(1)*- as shown in Table 1. The economic characteristics include the development (evolution) effort, schedule, and budget.

Table 1. Financial/real options/software architecture analogy

Option on stock	Real option on a project	Case of valuing architectural stability
Stock Price	Value of the expected cash flows	Value of the likely change
Exercise Price	Investment cost	Estimate of the likely cost to accommodate the change
Time-to-expiration	Time until opportunity disappears	Time-to-release (and deploy) the software generation
Volatility	Uncertainty of the project value	"Fluctuation" in the value of the requirement as deemed by the stakeholders; or changes in market-value of the requirements over a specified period of time
Risk-free interest rate	Risk-free interest rate	Interest rate relative to budget and schedule

Black and Scholes is an *arbitrage-based* technique. The technique requires knowledge of the value of the asset in question in span of the market. Software architectures, however, are (non-traded) real assets. Real options may be valued similarly to financial options, though they are not traded [18]. Real options valuation based on arbitrage-based pricing techniques determines the value of an asset in question in span of the market value using a correlated *twin asset* [18]. The twin asset is an asset that has the same risks the asset in question will have when the investment has been completed [18, 22].

To facilitate valuation using the principle of a twin asset, we consider the architecture as a portfolio of assets (rather than a single asset). More specifically, we view the architecture as a portfolio of requirements. In this context, we argue that the value of the architecture is in the value of the requirements it supports during the software system operation or tend to support as it evolves. This assumption facilitates calibrating requirements or changes in requirements with their market value.

The application of [5] assumes that the stock option is a function of the *stochastic* variables underlying stock's price and time. We assume that value of an evolvable architecture changes with time. It tends to change in *uncertain* ways and stochastically with the cost/value arising from changes in requirements.

4.2. Constructing call options to make the value of flexibility/stability tangible

Generally speaking, evolutionary changes are unanticipated. We assume that we can elicit a set of representative changes in requirements $\{i_1, i_2, \ldots, i_n\}$ that are *likely* to occur. Let us assume that the value of the architecture is V, where V corresponds to current stock price S_t. As the architecture evolves, the change in i_i is assumed to enhance the architecture value by x_i % with a follow-up investment of I_{ei}, where I_{ei} corresponds to an estimate of the likely cost to accommodate the change. This is similar to a call option to buy (x_i %) of the base project, paying I_{ei} as exercise price. Thus, the investment opportunity in an architecture can be viewed as a base-scale investment in the architecture plus *call options* on the future opportunities, where a future opportunity corresponds to the investment to accommodate the evolving requirement. The value of the constructed call options give an indication of the flexibility of the architecture to endure the likely changes in requirements $\{i_1, i_2, \ldots, i_n\}$. Thus, the value of the architecture materializes to equation *(4)* accounting for V and both the expected value and exercise cost to accommodate i_i for $i \leq n$. We assume that the interest rate is equal to zero for the simplicity of exposition.

$$V + \sum_{i=0}^{n} E\,[\max(x_i V - I_{ei}, 0)] \qquad (4)$$

4.3. Interpretation

We give an interpretation of *(4)* in the context of the evaluation for architectural stability.

For a likely change in requirement i_k,

(a) The option is *in the money*: if $x_k V$ exceeds the exercise cost (i.e. $\max(x_k V - I_{ek}, 0) > 0$), then the architecture is said to be *potentially stable* with respect to i_k. Generally speaking, the higher the value $x_k V$, the better the chances to exceed the exercise price of the option.

(b) The option is *out of money*: if the value of the call option sinks to zero (i.e. $\max(x_k V - I_{ek}, 0) = 0$), then there is no chance that the option will ever worth something in the future. The change is said to exhibit future *threats on the stability of the architecture*; the architecture is unlikely to be stable for *this* change.

Accounting for *all* the *n* likely changes in $\{i_1, i_2, \ldots, i_n\}$,

We interpret the *strategic* value of the investment in an architecture as the acquisition of a base asset that embeds growth opportunities. The values of the call options indicate the ability of an architecture to unlock future growth opportunities and enhance the upside potentials of the architecture (i.e. growth options). If the cumulative expected value of the future investments in all the changes tends to zero, it is very *unlikely* for the architecture to be *stable* with respect to the *likely* changes. In case of trade-offs, we interpret the strategic value relative to other candidate architectures. The more an architecture is able to unlock future opportunities, the more stable and "evolution friendly" it is likely to be.

5. Related work

Economic approaches to software design appeal to the concept of static NPV as a mechanism for estimating value [7, 10]. These techniques, however, are not readily suitable for strategic reasoning of software development as they fail to factor flexibility [6]. *Real options* theory has been adopted to address this problem: Baldwin and Clark [2, 3, 4] studied the flexibility created by modularity in design of components (of computer systems) connected through standard interfaces. They appear to be the first to observe that the value of modularity in design (of computer systems) can be modeled as real options. Sullivan [21] suggested that real options analysis can provide insights concerning modularity, phased projects structures, delaying of decisions and other dynamic software design strategies. Sullivan et al. [20] formalized that option-based analysis, focusing in particular on the flexibility to delay decisions making. Favaro et al. [10] developed an options-based approach to investment analysis for software reuse infrastructures. The options approach was used to value the flexibility provided by reuse to adapt in the face of uncertain conditions. Sullivan et al. [19] extended Baldwin and Clark's theory [2] that is developed to account for the influence of modularity on the evolution of the computer industry. Sullivan et al. [19] argued that the structure and value of modularity in software design creates value in the form of real options. 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. The value of such an option is the value that could be realized by the optimal experiment-and-replace policy. Knowing this value can help a designer to reason about both investment in modularity and how much to spend searching for alternatives.

Our use of real options theory appears to be novel. We use real options to predict the stability of software architectures in the face of the likely evolutionary changes. We value flexibility of the architecture to expand in the face of these changes; henceforth, what we value are the created growth options. For every likely evolutionary change, we construct a call option to value the flexibility of the architecture to accommodate the change(s). Knowing this value can assist in predicting the stability of the architecture for the likely evolutionary change(s). We interpret the strategic value of investment in the architecture as the acquisition of a base asset that embeds growth opportunities. The value(s) of the constructed call options are indicators of the ability of an architecture to unlock future growth opportunities and enhance the upside potentials of the architecture. We exploit [5] to valuation.

6. Conclusions and further work

Real options appears to be well suited to assist in predicting the stability of software architectures: it focuses explicitly on flexibility under uncertainty and makes it feasible to link likely changes to be accommodated by the architecture to value creation. Valuing flexibility- as a way to make the value of architectural stability tangible- appears to be achievable through constructing call option(s). The values of the options become assessing the payoff at exercise time. Our investigation has shown that adopting [5] to valuation seems to be promising. The analogy tends to hold under some assumptions.

The valuation requires the estimation of the behaviour of several parameters of the option model. For financial options, there are several proxies available to predict this behaviour- the most obvious proxy is simply the historical values of the financial asset. In real options such proxies rarely exist and the analyst may need to rely on experience and judgment in his estimations [10]. Our future work entails finding reasonable ways to estimate these parameters.

We will *empirically* evaluate the approach in an industrial setting with SearchSpace, one of UCL industrial partners. SearchSpace is investigating changing one of its products architectural infrastructure from CORBA to EJB. The investment in the change will increasingly be made on the basis of the stability that the architectural infrastructure creates with respect to the forward-looking strategic benefits. Roughly speaking, changing the product infrastructure from CORBA to EJB may (or may not) create growth options. These options

may be exercised at a point in the future to realize certain gains. Evaluating the payoff of these options may give an indication of the stability that such change may create.

The work is expected to form a genuine effort on understating the relation between changes in requirements and the architecture through strategic value-based reasoning. It aims to assist stakeholders' in strategic *"what if"* analysis, analyzing the strategic position of the enterprise- if the enterprise is highly centered on the software architecture (as it is the case in web-based service providers companies) and evaluating trade-offs between two or more candidate software architectures for stability. The intellectual framework is most critical; it demonstrates that with *value-based* reasoning we can improve our ability to evaluate for architectural stability and develop software systems that need to adapt to the inevitable evolving requirements.

7. References

[1] Amram, M., Kulatilaka, N.: Real Options: Managing Strategic Investment in an Uncertain World. Harvard Business School Press, Cambridge, Massachusetts (1999)

[2] Baldwin, C. Y., Clark, K. B.: Design Rules - The Power of Modularity. MIT Press (2001)

[3] Baldwin, C. Y., Clark, K. B.: Managing in the Age of Modularity. Harvard Business Review, Vol. 75(5). (1997) 84-93

[4] Baldwin, C. Y., Clark, K. B.: Modularity and Real options. Working paper, Harvard Business School (1993)

[5] Black, F., Scholes, M.: The Pricing of Options and Corporate Liabilities. Journal of Political Economy (1973)

[6] Boehm, B., Sullivan, K. J.: Software Economics: A Roadmap. In: Finkelstein, A. (ed.): The Future of Software Engineering (2000)

[7] Boehm, B.: Software Engineering Economics. Prentice Hall (1981)

[8] Cox, J., Huang, C. F.: Option Pricing Theory and Its Applications. In: Bhattacarya, S., Constantinides, G. (eds.): Theory of Valuation: Frontiers of Modern Finance, Vol. 1. Totowa, NJ: Rowman and Littlefield (1989) 272-288

[9] Cox, J., Ross, S., Rubinstein, M.: Option Pricing: A Simplified Approach. Journal of Financial Economics. Vol.7 (3). (1979) 229-264

[10] Favaro, J. M., Favaro, K.R., Favaro, P. F.: Value-Based Software Reuse Investment. Annals of Software Engineering. Vol. 5. (1998) 5 – 52

[11] Finkelstein, A.: Architectural Stability, Some Preliminary Comments.http://www.cs.ucl.ac.uk/staff/a.finkelstein/talks.html (2000)

[12] Hull, J. C.: Options, Futures, and Other Derivative Security. Third edition, Prentice-Hall (1997)

[13] McDonald, R., Siegel, D.: The Value of Waiting to Invest. Quarterly Journal of Economics. Vol. 101(4). (1986) 707–727

[14] Merton, R. C.: Continuous-Time Finance. Blackwell, Cambridge, Massachusetts (1990)

[15] Myers, S. C.: Finance Theory and Financial Strategy. Corporate Finance Journal. Vol. 5(1). (1987) 6-13

[16] Myers, S. C.: Determinants of Corporate Borrowing. Journal of Financial Economics. Vol. 5(2). (1977) 147-175

[17] Ross, S. A., Westfield, R.W., Jaffe, J.: Corporate Finance (Fourth). Irwin, Chicago (1996)

[18] Schwartz, S., Trigeorgis, L.: Real options and Investment Under Uncertainty: Classical Readings and Recent Contributions. MIT Press Cambridge, Massachusetts (2000)

[19] Sullivan, K. J., Griswold, W., Cai, Y., Hallen, B.: The Structure and Value of Modularity in Software Design. In: Proc. ESEC/FSE-9, Vienna, Austria (2001) 99-108

[20] Sullivan, K. J.: Chalasani, P., Jha, S., Sazawal, V.: Software Design as an Investment Activity: A Real Options Perspective. In: Real Options and Business Strategy: Applications to Decision-Making. Trigeorgis L.(ed.) Risk Books (1999)

[21] Sullivan, K. J.: Software Design: The Options Approach. In: 2nd International Software Architecture Workshop, Joint Proceedings of the SIGSOFT '96 Workshops. San Francisco, CA (1996) 15–18

[22] Trigeorgis, L.: Real options in Capital Investment: Models, Strategies, and Applications. Praeger Westport, Connecticut London (1995)

Understanding the Economics of Refactoring

Rob Leitch
MacDonald, Dettwiler and Associates, Ltd.
13800 Commerce Parkway
Richmond, BC, V6V 2J3, Canada
1 (604) 231 2184
rleitch@mda.ca

Eleni Stroulia
Department of Computing Science
221 Athabasca Hall, University of Alberta
Edmonton, AB, T6G 2E8, Canada
1 (780) 492 3520
stroulia@cs.ualberta.ca

ABSTRACT

In this paper we discuss a novel method for estimating the expected maintenance savings given a refactoring plan. This work is motivated by the increased adoption of refactoring practices as part of new agile methodologies and the lack of any prescriptive theory on when to refactor.

1. INTRODUCTION AND MOTIVATION

Estimating the cost of future maintenance activities on a working application is an important research question. If such an estimate were possible, the guesswork would be eliminated from the decision of whether to maintain or replace existing software. There is some evidence in the literature [1] [4] that perfective maintenance accounts for the majority of the overall maintenance effort in a project. Perfective maintenance activities aim to improve the quality attributes of the software, such as its performance or its maintainability. Therefore the problem of "maintenance cost prediction" can be recast as "perfective maintenance cost prediction".

A long-standing method in support of perfective maintenance is local source code transformation, more recently re-discovered as "Refactoring" [3]. Although there are many tools developed to support code transformations there is no general agreement on what transformations are beneficial and when these changes should be applied. For example, the refactoring catalog contains "symmetrical" refactorings, i.e., opposite transformations such as "extract method" and "inline method". In addition, there are alternative refactorings applicable to similar low-level designs, such as "extract subclass" and "extract interface". It is up to developers to decide which type of refactoring to apply in anticipation of future development. Furthermore, currently there is only informal advice on when to refactor. Fowler [3] suggests that refactoring may not be beneficial when there is a deadline coming up or when the software is of such poor quality that it would be easier to re-develop it from scratch. This advice implies an estimate of the cost of refactoring vs. the cost of redevelopment. However, no such cost-estimate model exists. In spite of the lack of strong prescribed methodology, most popular agile methods advocate refactoring as a regular practice in the software lifecycle. This practice is becoming widely adopted as the method of choice for improving the extendibility and maintainability of software.

In our recent work, we have been investigating several aspects of refactoring. These aspects include understanding the impact of long-term code transformations on the quality of software design and the nature of developing a cost-benefit model estimating the tradeoff between the up-front cost of refactoring and the expected downstream maintenance savings. Specifically, we are interested in predicting the Return on investment (ROI) for a planned refactoring activity.

ROI = (Maintenance Savings from Proposed Refactoring) / (Development Cost of Planned Refactoring). (1)

If the ROI is greater than or equal to one, then the planned refactoring will be cost effective.

2. ESTIMATING THE REFACTORING ROI

To calculate the Refactoring ROI according to formula (1) above, we need to estimate

1. the development cost of the planned refactoring activity, and
2. the anticipated maintenance cost of each of the two software versions (i.e., before and after refactoring).

We adopt COCOMO [2] to calculate the refactoring-plan development cost, and we propose a novel method for predicting the maintenance effort for the original and restructured designs.

A fairly common approach to this problem has been to try and relate design metrics to observed maintenance costs through regression analysis. However, while metrics can be used to identify outlier design components and to comparatively evaluate alternative designs, there are currently no suitable predictive models of maintenance effort. One reason for this is the nature of software maintenance. Corrective maintenance effort is directly related to latent defects or faults in the system, while perfective and adaptive maintenance are directly related to system enhancement in response to functional evolution or environmental changes. There is evidence that the perfective effort category accounts for the majority of maintenance cost [1] [4]. Because this type of maintenance is influenced by factors external to the system, it is not obvious that such effort can be predicted by design metrics. In addition, there is no general agreement regarding which metrics can predict system fault density.

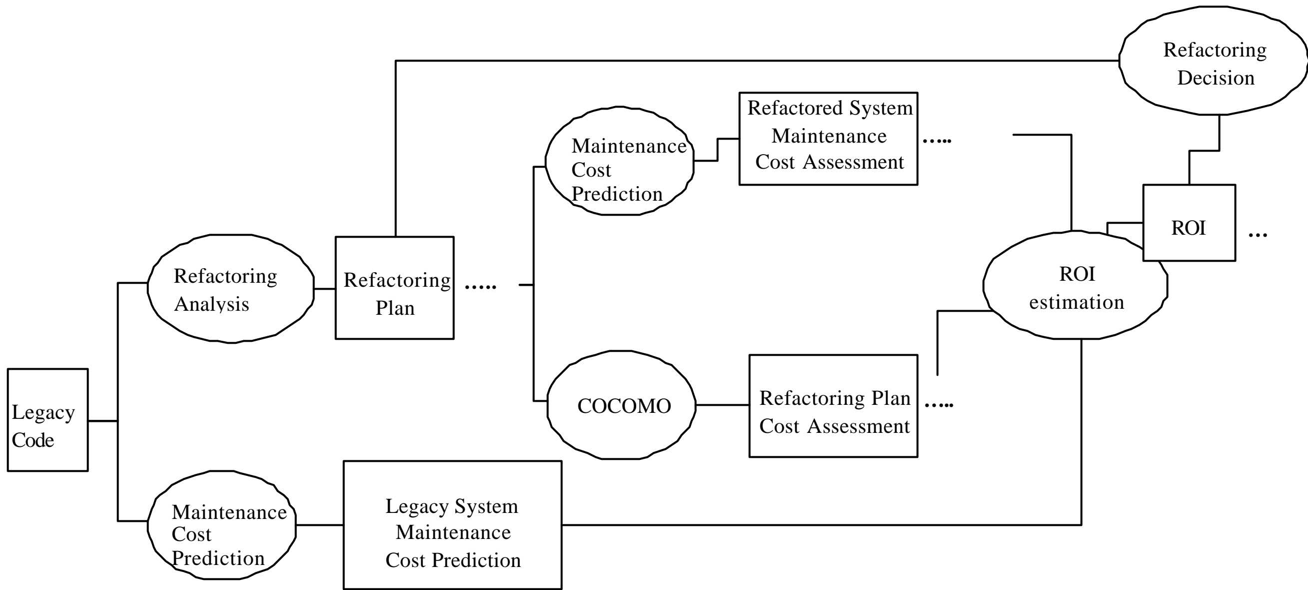


Figure 1: Informed Refactoring Decision Making, using Refactoring ROI estimates.

In our work we have been experimenting with an alternative strategy for predicting maintenance cost. This strategy is based on the following assumptions.

The anticipated future maintenance cost of a given software system is the sum of the costs of each individual future maintenance request.

Maintenance activities occur randomly in the software system, as modifications necessitated by new requirements on the software system. Therefore the probability that a maintenance request will strike an individual module is directly proportional to the size of the module relative to the size of the overall system.

A substantial part of the cost of any single modification is the cost of the regression testing necessitated after the modification is completed. We assume that the regression-testing savings brought about by refactoring can be substantial enough to bring the ROI fraction above 1. With these assumptions we can restate formula (1) as follows:

The regression-testing cost of a particular modification is directly proportional to the amount of code that has to be examined as a result of the change. This in turn can be estimated based on the dependencies of the modified module with the rest of the system.

ROI = (Regression-Testing Savings from Proposed Refactoring) / (Development Cost of Planned Refactoring). (2)

The ROI estimation process implied by these assumptions, as well as the informed refactoring decision-making process it enables, are depicted in Figure 1. Given a legacy system, its expected regression-testing cost is first calculated based on the occurrence of a random maintenance activity. Next, a number of alternative refactoring plans can be formulated and their respective development costs estimated using COCOMO. The predicted regression-testing costs of the proposed new designs are then calculated. At this point, the ROI of each alternative refactoring plan can be computed. These estimates are then used to decide whether refactoring the system is beneficial, and what sort of refactoring plan should be implemented.

3. EXPLORATORY CASE STUDY

Let us now illustrate our ROI estimation method with an exploratory case study using a simple Java system. The trial system was created in a student environment as part of a graduate course in Object-oriented (OO) analysis and design. The code followed a typical OO development cycle, including user requirements definition through use-case analysis, development of a class model, and dynamic state modeling prior to implementation. The application is a real-time traffic light control system for a four-way intersection, including a graphical simulation of the intersection operation. We refer to this system as “TrafficApp”.

For the purposes of counting “Source Lines of Code” (SLOC) in a procedure, we use the definition of a logical source statement as defined in the COCOMOII.2000 model [3]. Using this definition, the trial case study program contains 740 SLOC, broken down into 6 classes and 29 procedures. Table 1 shows the distribution of code and procedures within TrafficApp.

3.1. Refactoring Plan

A source code walkthrough was performed on TrafficApp to identify candidate refactoring opportunities according to the criteria defined in [3]. The result of this walkthrough is a list of suggested refactorings presented in Table 4, along with the projected source code impact for each affected procedure. The data in Table 4

indicates the amount of code to be added or deleted for each procedure. Note that the recommended restructuring will add new procedures to the system. Table 1 shows the predicted impact of the restructuring at the class level, including changes in code size and the number of procedures.

Table 1: Code and procedures in TrafficApp.

	Size (SLOC)		
Class	Before	After	Change
Class 1	411	343	-17%
Class 2	164	81	-51%
Class 3	95	108	14%
Class 4	23	23	0%
Class 5	23	23	0%
Class 6	24	24	0%
TOTAL	740	602	-19%
	No. of Proc.		
Class	Before	After	Change
Class 1	4	11	175%
Class 2	7	8	14%
Class 3	6	7	17%
Class 4	5	5	0%
Class 5	2	2	0%
Class 6	5	5	0%
TOTAL	29	38	31%
	Avg. Proc. Size (SLOC)		
Class	Before	After	Change
Class 1	103	31	-70%
Class 2	23	10	-57%
Class 3	16	15	-3%
Class 4	5	5	0%
Class 5	12	12	0%
Class 6	5	5	0%
TOTAL	26	16	-38%

3.2 Impact on the Dependency Structure

Two sets of data and control dependency graphs were constructed for TrafficApp. One set of graphs represents the system state before refactoring (based on a manual code inspection). The second set of graphs represents the predicted state of TrafficApp after refactoring. The plot of Figure 2 illustrates the difference between these sets of graphs, showing changes in the dependency structure of the system resulting from the proposed restructuring.

3.3. Mean Re-test Impact

Table 5 shows the calculation of the mean re-test impact for TrafficApp before and after restructuring based on the overall dependency graphs and the source code distribution in the system. The mean re-test impact before refactoring is 408 SLOC, while the predicted mean re-test impact after refactoring is 216 SLOC.

3.4. Effort Calculations

Table 2 summarizes the example calculations performed using the COCOMOII.2000 model to predict maintenance costs before and after refactoring as well as the cost of the restructuring. As well, this table presents the COCOMO re-use model parameters assumed for TrafficApp. The net result is a predicted savings of 0.225 person-months per maintenance activity as a result of the proposed restructuring. This compares with a restructuring cost of 1.18 person-months.

Table 2: COCOMOII.2000 cost predictions for TrafficApp.

Parameter	Refact. Cost	Maint. Cost Before	Maint. Cost After	Maint. Savings
Size (KSLOC)	0.740	0.740	0.602	-
EAF	1.000	1.000	1.000	-
Scale Factor	18.970	18.970	18.970	-
Exponent	1.100	1.100	1.100	-
SU	30.000	30.000	30.000	-
AA	4.000	4.000	4.000	-
UNFM	0.400	0.400	0.400	-
DM	13.800	5.000	5.000	-
CM	29.700	5.000	5.000	-
IM	100.000	55.100	35.900	-
Equiv. KSLOC	0.437	0.213	0.131	-
Effort (p-months)	1.180	0.538	0.313	0.225

3.5. ROI Calculation

From Table 2, we can see that the ROI will be greater than one if there are greater than or equal to six maintenance activities after the design restructuring. This is determined by dividing the refactoring cost by the maintenance savings per activity (=1.18/0.225=5.2).

3.6. Results Discussion

Table 3 provides a comparison between the dependency graphs before and after refactoring, measuring the number of dependency paths shown in each graph. This result shows that the density of dependency paths in the restructured graphs is lower than for the original design.

Table 3: Dependency graphs before and after refactoring.

	BEFORE		AFTER		
Graph	No. Dep.	Fill Ratio	No. Dep.	Fill Ratio	Chng.
Data	112	13.3%	147	10.2%	-23.6%
Control	73	8.7%	101	7.0%	-19.4%
Overall	179	21.3%	241	16.7%	-21.6%

In Figure 3, each data point represents the re-test impact of a single procedure versus the probability of that impact occurring for a random maintenance event. Note that the impact data is expressed as a percentage of the total SLOC in the system rather than as an absolute SLOC number. For the combined graph in Figure 3, the code size reference is the original, unchanged version of TrafficApp.

From the above analysis, the proposed refactoring is predicted to decrease the overall code size by 19% and increase the number of procedures in the system by 31%. In addition, the density of dependency paths in the system is predicted to decrease by approximately 22%. This decrease in density appears to result from the introduction of new procedures into the system possessing relatively few external dependencies. These new procedures are created by extracting code from larger original procedures (using the Extract Method and Move Method transformations defined in [3]).

Table 4: Proposed refactoring plan and design impact for TrafficApp.

Proc. No.	Code Problems	Refactoring	Add.	Del.	Proc. No.	Code Problems	Refactoring	Add.	Del.
1	Long Method, Duplicated Code, Feature Envy	Extract Method	24	225	33	N/A (new proc.)	Extract Method	27	0
2	Duplicated Code	Extract Method	4	28	34	N/A (new proc.)	Extract Method	81	0
10	Switch Statement, Duplicated Code, Feature Envy	Move Method	4	49	35	N/A (new proc.)	Extract Method	17	0
11	Long Method, Switch Statement, Duplicated Code	Extract Method	4	56	36	N/A (new proc.)	Extract Method	9	0
30	N/A (new proc.)	Extract Method	4	0	37	N/A (new proc.)	Move Method	13	0
31	N/A (new proc.)	Extract Method	9	0	38	N/A (new proc.)	Extract Method	14	0
32	N/A (new proc.)	Extract Method	10	0	-	-	-	-	-
		SUBTOTAL:	59	358			SUBTOTAL:	161	0
							TOTAL:	220	358

Table 5: Mean re-test impact before and after restructuring.

		BEFORE REFACTORING					AFTER REFACTORING			
Class No.	Proc. No.	Size (SLOC)	Test Impact (SLOC)	Prob.	Mean (SLOC)	Proc. No.	Size (SLOC)	Test Impact (SLOC)	Prob.	Mean (SLOC)
Class 1	1	306	677	41.4%	279.9	1	105	539	17.4%	94.0
	2	80	106	10.8%	11.5	2	56	91	9.3%	8.5
	3	18	84	2.4%	2.0	3	18	32	3.0%	1.0
	4	7	7	0.9%	0.1	4	7	7	1.2%	0.1
	-	-	-	-	-	30	4	116	0.7%	0.8
	-	-	-	-	-	31	9	121	1.5%	1.8
	-	-	-	-	-	32	10	122	1.7%	2.0
	-	-	-	-	-	33	27	139	4.5%	6.2
	-	-	-	-	-	34	81	193	13.5%	26.0
	-	-	-	-	-	35	17	129	2.8%	3.6
	-	-	-	-	-	36	9	91	1.5%	1.4
Class 2	5	27	530	3.6%	19.3	5	27	228	4.5%	10.2
	6	3	203	0.4%	0.8	6	3	141	0.5%	0.7
	7	1	201	0.1%	0.3	7	1	139	0.2%	0.2
	8	3	120	0.4%	0.5	8	3	64	0.5%	0.3
	9	15	99	2.0%	2.0	9	15	61	2.5%	1.5
	10	49	133	6.6%	8.8	10	4	18	0.7%	0.1
	11	66	120	8.9%	10.7	11	14	82	2.3%	1.9
	-	-	-	-	-	38	14	61	2.3%	1.4
Class 3	12	29	677	3.9%	26.5	12	29	539	4.8%	26.0
	13	7	447	0.9%	4.2	13	7	222	1.2%	2.6
	14	18	562	2.4%	13.7	14	18	253	3.0%	7.6
	15	18	99	2.4%	2.4	15	18	61	3.0%	1.8
	16	1	188	0.1%	0.3	16	1	97	0.2%	0.2
	17	22	106	3.0%	3.2	17	22	54	3.7%	2.0
	-	-	-	-	-	37	13	49	2.2%	1.1
Class 4	18	9	357	1.2%	4.3	18	9	156	1.5%	2.3
	19	3	605	0.4%	2.5	19	3	310	0.5%	1.5
	20	3	605	0.4%	2.5	20	3	310	0.5%	1.5
	21	1	1	0.1%	0.0	21	1	1	0.2%	0.0
	22	7	614	0.9%	5.8	22	7	319	1.2%	3.7
Class 5	23	3	336	0.4%	1.4	23	3	135	0.5%	0.7
	24	20	20	2.7%	0.5	24	20	20	3.3%	0.7
Class 6	25	7	321	0.9%	3.0	25	7	120	1.2%	1.4
	26	5	111	0.7%	0.8	26	5	96	0.8%	0.8
	27	1	401	0.1%	0.5	27	1	176	0.2%	0.3
	28	1	1	0.1%	0.0	28	1	1	0.2%	0.0
	29	10	11	1.4%	0.1	29	10	11	1.7%	0.2
		MEAN RE-TEST IMPACT:			408		MEAN RE-TEST IMPACT:			216

Figure 2: The changes in the dependency structure of TrafficApp before and after the proposed restructuring.

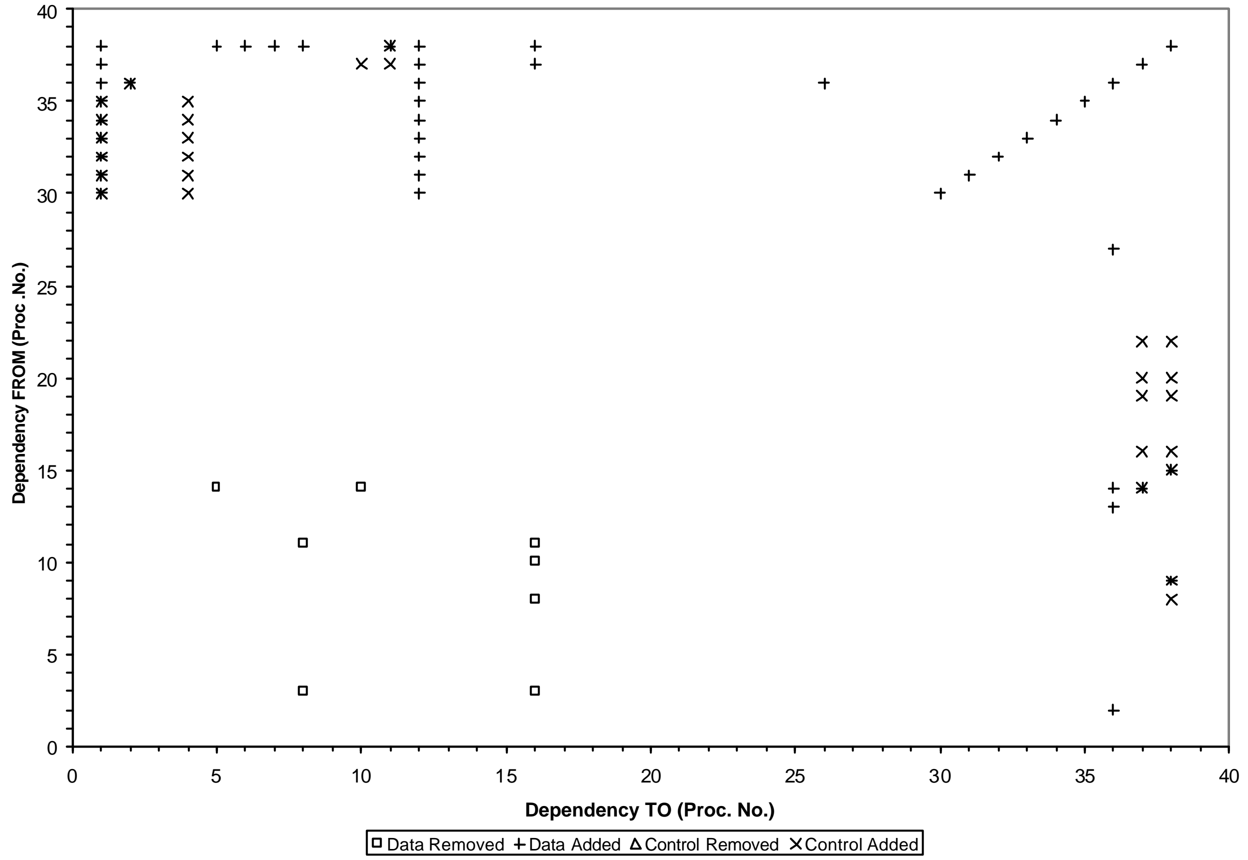
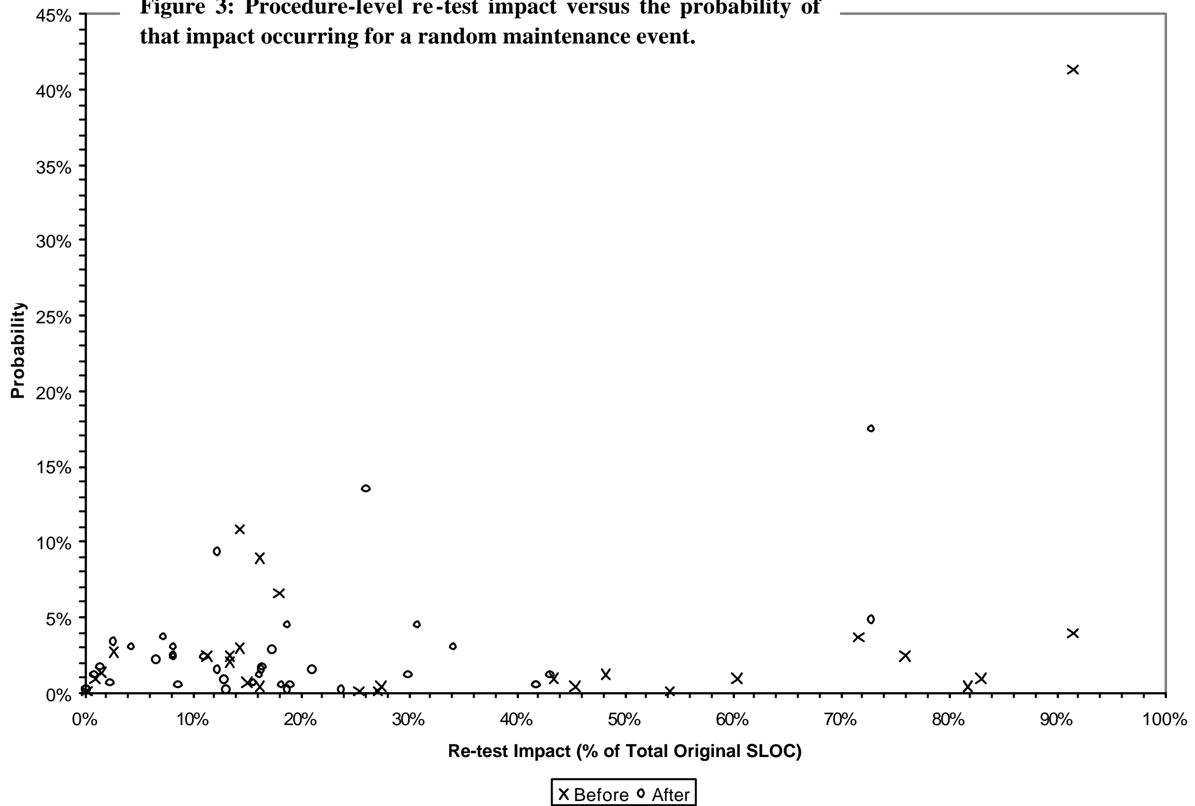


Figure 3: Procedure-level re-test impact versus the probability of that impact occurring for a random maintenance event.



The refactoring appears to reduce the peaks along both axes of the impact probability distribution of Figure 3. Compared to the original distribution, the refactored distribution appears shifted down and to the left.

Procedure-level dependency analysis predicts that the mean regression testing impact (in terms of affected SLOC) of a random maintenance activity will decrease by approximately 47% due to the proposed design restructuring.

Cost estimation modeling using COCOMOII.2000 suggests that the restructuring will be cost effective if six or more maintenance events occur after the refactoring investment.

The results of the procedure-level analysis are not duplicated by a class-level analysis of the same design transformations. In general, the class-level analysis yields more conservative results regarding cost-effectiveness. It appears that the class-level approach is not as sensitive to the proposed design restructuring activities.

4. DISCUSSION

This work is at a very early stage, however we have applied our refactoring ROI method to two exploratory Java case studies: a trial academic system with 740 SLOC and a commercial database application containing 2.5 KSLOC. The case study results provide measurements of the effects of restructuring on parameters such as mean code re-test impact, number of system data and control dependency paths, and system size. In addition, we estimated the break-even point in terms of the number of maintenance activities to achieve ROI > 1 for the proposed design transformations. Our results show that common low-level source code transformations can change the system dependency structure in a beneficial way, allowing recovery of the initial refactoring investment over a number of maintenance activities simply on the basis of regression-testing savings.

This early experience has generated several interesting "research leads" that we would like to pursue. For example, it would be interesting to explore other metrics to estimate maintenance benefits in addition to examining regression-testing costs. Furthermore, we are currently estimating regression-testing cost based on procedure-level dependencies; would other more or less precise metrics be better predictors? Could the refactoring decision-making process be influenced by previous refactorings applied to the system, i.e., can some refactorings preclude other refactorings from occurring in the future?

5. REFERENCES

[1] Basili, V., Briand, L., Condon, S., Kim, Y., Melo, W. y Valett, J.D., Understanding and Predicting the Process of Software Maintenance Releases", Proceedings of the International Conference on Software Engineering, IEEE Computer Society, Los Alamitos, CA (USA), 1996, pp. 464-474.

[2] Boehm, B., Horowitz, E., Madachy, R., Reifer, D., Clark, B.D., Steece, B., Brown, A.W., Chulani, S., and Abts, C., Software Cost Estimation with COCOMO II, Prentice Hall PTR, 2000.

[3] Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D., Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.

[4] Polo, M., Piattini, M., Ruiz, F. Using code metrics to predict maintenance of legacy programs: a case study, IEEE ICSM 2001, Florence, Italy.

Composable Process Elements for Developing COTS-Based Applications

EDSER-5 Position Paper

*Ye Yang, Jesal Bhuta, Barry Boehm, Dan Port, Chris Abts**
*University of Southern California, Texas A&M University**
{ yey, boehm, dport, jesal}@cse.usc.edu, cabts@cgsb.tamu.edu

1. Introduction

In his ICSE 2002 keynote address [3], Robert Balzer issued a challenge to the software engineering community to provide better methods for dealing with COTS-based software systems, and to present them at subsequent ICSE's. This paper provides a partial response to this challenge. It presents some data that we have found useful in understanding COTS-based application (CBA) trends and effort distributions. The COTS effort distributions and sequences also suggest a framework for the primary contributions of the paper: a set of composable process elements and a decision framework for using them in the development of CBA's.

Traditional sequential requirements-design-code-test (waterfall) processes do not work for CBA's [11], simply because the decision to use a COTS product constitutes acceptance of many, if not most, of the requirements that led to the product, and to its design and implementation. In fact, it is most often the case that a COTS product's capabilities will drive the "required" feature set for the new product rather than the other way around, though the choice of COTS products to be used should be driven by the new project's initial set of "most significant requirements." Additionally, the volatility of COTS products [9] introduces a great deal of recursion and concurrency into CBA processes.

Some recent CBA process models have partially addressed these issues by adding CBA extensions to a sequential process framework [8]. These work in some situations, but not in others where the requirements, architecture, and COTS choices evolve concurrently; the example in Section 4 illustrates this point.

Other process frameworks such as the spiral model [5] and the SEI Evolutionary Process for Integrating COTS-Based Systems (EPIC) process [2] provide suitably flexible and concurrent frameworks for CBA processes. However, they have not, to date, provided a specific decision framework for navigating through the option space in developing CBA's. They identify key activities (evaluate alternatives; identify and resolve risks; accumulate specific kinds of knowledge; increase stakeholder buy-in; make incremental decisions that shrink the trade space), but leave their sequencing to the individual CBA developer.

The decision framework presented here is based on our experience in analyzing large CBA's in the course of gathering empirical data for the Constructive CBA cost model (COCOTS), a COTS counterpart to COCOMOII [1,6], and our experience developing and analyzing several dozen e-services CBA's for USC's Information Services Division and its Center for Scholarly Technology [4].

2. Definitions and Context

2.1 Definitions

We adopt the SEI COTS-Based System Initiative's definition [7] of a *COTS product*: A product that is:

- Sold, leased, or licensed to the general public;
- Offered by a vendor trying to profit from it;
- Supported and evolved by the vendor, who retains the intellectual property rights;
- Available in multiple identical copies;
- Used without source code modification.

We also follow the SEI in defining a *COTS-Based System* very generally as "any system, which includes one or more COTS products." This includes most current systems, including many which treat a COTS operating system and other utilities as a relatively stable platform on which to build applications. Such systems can be considered "COTS-based systems," as most of their executing instructions come from COTS products, but COTS considerations do not affect the development process very much.

To provide a focus on the types of applications for which COTS considerations do affect the development process, we define a *COTS-Based Application* as a system for which at least 30% of the end-user functionality (in terms of functional elements: inputs, outputs, queries, external interfaces, internal files) is provided by COTS products, and at least 10 % of the development effort is devoted to COTS considerations. The numbers 30% and 10% are not sacred quantities, but approximate behavioral CBA boundaries observed in the application projects. There was a significant gap observed in COTS-related effort reporting. The projects observed either reported less than 2% or over 10% COTS-related effort, but never between 2-10%.

In our six years of iteratively defining, developing, gathering project data for, and calibrating COCOTS cost estimation model, we identified four primary sources of project effort due to CBA development considerations. These are defined in COCOTS as follows:

- COTS *Assessment* is the activity whereby COTS products are evaluated and selected as viable components for a user application.
- COTS *Tailoring* is the activity whereby COTS software products are configured for use in a specific context. This definition is similar to the SEI definition of "tailoring" [10].
- COTS Glue Code development and integration is the activity whereby code is designed, developed, and used to ensure that COTS products satisfactorily interoperate in support of the user application.

2.3 CBA Activity Distribution

Based on 2000-2002 USC-CSE e-services data and 1996-2001 COCOTS calibration data, we observe a large variation of COTS-related (assessment, tailoring, and glue code) effort distributions. This is clearly illustrated in the e-services and COCOTS COTS effort distributions in Figures 2.3a and 2.3b respectively.

The industry projects in Figure 2.3b were a mix of small-to-large business management, engineering analysis, and command control applications. Assessment effort ranged from 1.25 to 147.5 person-months (PM). Tailoring effort ranged from 3 to 648 PM; glue code effort ranged from 1 to 1411 PM.

Some CBA approaches, including our initial approach to COCOTS, just focus on one CBA activity such as glue code or assessment. As shown in Figure 2.3a, some projects (projects 3, 8, 9, 10) are almost purely tailoring efforts, while other projects (projects 2, 4, 5) spent most of the time on COTS assessment. The industry projects in Figure 2.3b have similar attributes. We also note that all projects had some degree of assessment, and so we never have observed tailoring or glue code only efforts, or a mix of only these two. In addition, the assessment and glue code-only combination is very rare.

In previous work [12] these observations have led us to believe that there are typically three types of CBA projects. These are chiefly-assessment oriented; chiefly assessment and tailoring; or a significant mix of all three COTS-related activities. We found that different CBA types had significantly different project attributes (such as requirements flexibility), risk profiles, and project development characteristics.

Another notable fact found from looking at effort data that was collected on a weekly basis, is that assessment activities (A), tailoring activities (T), and glue code development (G) are not necessarily sequential. Table 1 shows a sampling of the A, T, G sequences for some of the e-services projects (we also denote custom development with the letter C). The sequences of activities are time ordered from left to right and activities undertaken in parallel are indicated by placing the activity letters within parentheses. We note that all sequences begin with assessment. We also note that the small cycle of ATG or A(TG) is very common and often repeating combination.

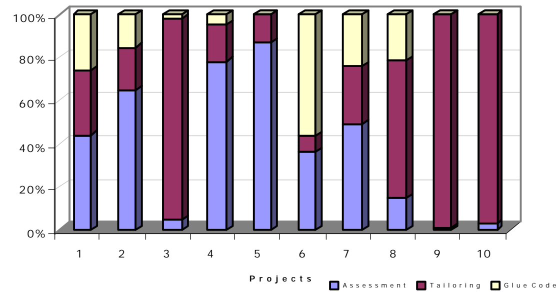


Figure 2.3a. CBA Effort Distribution of USC e-Service Projects

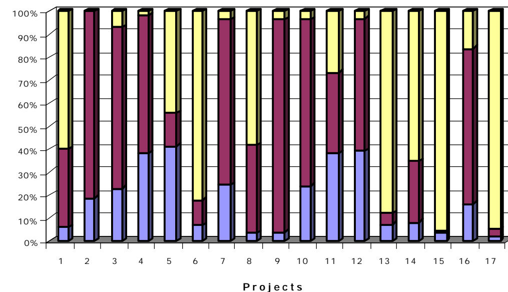


Figure 2.3b. CBAs Effort Distribution of COCOTS Calibration Data

The different combinations of assessment, tailoring, and glue code activities resulted from insufficient earlier assessment (for example, project No. 6), COTS changes (Project No. 5, 7), or requirement changes (project No. 4). Such decision factors are not directly addressed by the current literature on CBS processes. Therefore, we have developed a CBA process decision framework and a set of composable process elements to address and accommodate these critical factors. This decision framework is consistent with our empirical data and the distribution and sequence observations discussed in this section.

No.	Process Map
4	ATGC
5	ATA
6	A(TG)AG
7	A(TG)A(TG)

Table 1. CBA Effort Sequences

3. CBA Process Decision Framework and Process Elements

As evidenced in section 2.3, there are a wide variety of CBA effort distributions, and the particular effort distribution of a CBA significantly reflects its project risks and development characteristics. As such, applying a one-size-fits-all development process is likely to encounter difficulties in addressing the needs and risks of a given CBA.

As the fraction of CBA's has increased among our USC e-services projects, we have encountered increased conflict between CBA process needs and our UML-based MBASE process and documentation guidelines. This has led to a good deal of confusion, frustrating re-work, risky decisions, and a few less than satisfactory products.

A notable example of this re-work occurred within one of the authors' "USC Collaborative Services" project in which the developers scrapped (after much expended effort) their process-mandated UML based design models and substituted extensive and detailed assessments and comparisons of several COTS packages, each of which covered most or all of the desired capabilities.

In analyzing this problem, we found that the ways that the better projects handled their individual assessment, tailoring, and glue code activities exhibited considerable similarity at the process element level. We also found that these process elements fit into a recursive and reentrant decision framework accommodating concurrent CBA activities and frequent go-backs based on new and evolving OC&P's and COTS considerations. We now describe the CBA process decision framework and its respective assessment, tailoring, and glue code elements.

3.1. The CBA Process Decision Framework

Figure 3.1 presents the dominant decisions and activities within CBA development as abstracted from our observations and analysis of USC e-services and CSE- affiliate projects. This represents the overall CBA decision framework that composes the assessment, tailoring, glue code, and custom code development process elements within an overall development lifecycle.

Some explanation of Figure 3.1 is in order. The CBA process is undertaken by "walking" a path from "start" to "Non-CBA Activities" that connects (via arrows) activities as indicated by boxes and decisions that are indicated by ovals. Activities result in information that is passed on as input to either another activity or used to make a decision. Information follows the path that best describes the activity or decision output. Only one labeled path may be taken at any given time for any particular walk; however it is possible to perform multiple activities simultaneously (e.g. developing custom application code and glue code, multiple developers assessing or tailoring).

The small circles with letters A, T, G, C indicate the assessment, tailoring, glue code, and custom code development process elements respectively. With the exception of the latter, each of these areas will be expanded and elaborated in the sections that follow. These areas can generate the development activity sequences indicated in Table 1 by noting the order that these process elements are visited. Each area may enter and exit in numerous ways both from within the area itself or by following the decision framework of Figure 3.1. In addition, this scheme was developed from and is consistent with the CBA activity distributions of Figures 2.3. In particular, only (and in fact all) "legal" distributions are possible (e.g. that all distributions have assessment effort is consistent with all paths in the framework initially passing through the assessment element (or area "A"). We now summarize the less obvious aspects of each process area.

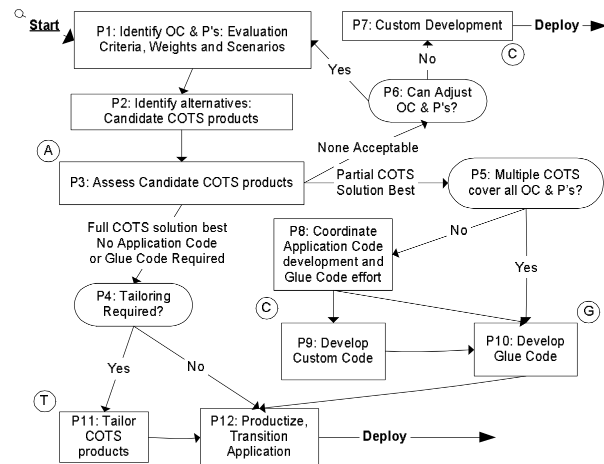


Figure 3.1. CBA Effort Decision Framework

P1: Identify OC&P's: Evaluation Criteria, Weights and Scenarios. This is the entrance to the CBA process where the initial evaluation attributes and desired operational outcomes for the application are established. Risk considerations, stakeholders' priority changes, new COTS releases and other dynamic considerations may significantly alter the objectives, constraints, and priorities (OC&P's). In particular, if no suitable COTS packages are identified, the stakeholders may change the OC&P's and the process is started over with these new considerations.

P2: Identify alternatives: Candidate COTS products. This and activity P1 establish the entry conditions for an Assessment activity.

P5: Multiple COTS cover all OC & P's? If a combination of COTS products can satisfy all the OC&P's, they are integrated via glue-code. Otherwise, COTS packages are combined to cover as much of the OC&P's as feasible and then custom code is developed to cover what remains.

P6: Can Adjust OC & P's?
When no acceptable COTS products can be identified, the OC&P's are re-examined for areas that may allow more options. Are there constraints and priorities that may be relaxed that have eliminated some products from consideration? How firm are the objectives and if adjusted slightly will it enable consideration of more products? Are there analogous areas in which to look for more products and alternatives?

P8: Coordinate Application Code development and Glue Code effort. Custom developed components must eventually be integrated with the chosen COTS products. The interfaces will need to be developed so they are compatible with the COTS products and the particular glue code connectors used. This means that some glue code effort will need to be coordinated with the custom development.

3.2. Assessment Process Element

COTS assessment aims at helping to make buy-or-build choices and helping select the most satisfactory combination of COTS products from various candidates. Current approaches to COTS assessment processes identify key tasks and emphasize the concurrency and high coupling involved in the tasks [14,15]. But they leave open how COTS assessment fits with tailoring, glue code, and the overall process. Here we present a COTS assessment process that provides these linkages as shown in Figure 3.2:

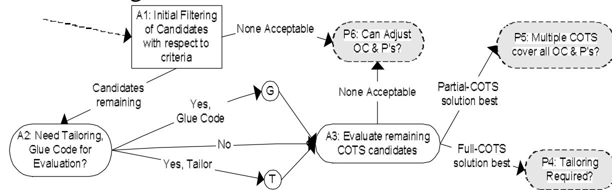


Figure 3.2. The Assessment Process Element

Entry Conditions for Assessment

The entry condition for assessment assumes that suitable COTS evaluation criterion, their corresponding weights, business scenarios, and COTS candidates are present (starting with the results of Fig. 3.1 decision elements P1 and P2).

Evaluation criteria and weights are established based on stakeholder-negotiated OC&P's for the system. Stakeholders also agree on the business scenarios to be used for the assessment. The assessment sub-model of COCOTS has collected an extensive list of attributes used in COTS evaluation [1,6].

A1: Initial Filtering. Initial assessment tries to quickly filter out the unacceptable COTS packages based on the evaluation criteria. The objective of this activity is to reduce the number of COTS candidates needing to be evaluated in detail. If no available COTS products pass this filtering, this assessment element ends up at the "none acceptable" exit.

A2: Tailoring or Glue Code Needed for Evaluation. The remaining COTS candidates from initial filtering will undergo more detailed assessment. To do so, some COTS products need to be tailored (e.g., to assess usability), and some need to be integrated by glue code development (e.g., to assess interoperability).

A3: Detailed Assessment. The focus of detailed assessment is to collect data/information about each COTS candidate against evaluation criteria from pre-designed business scenarios, analyze the data and make decision trade-offs. Some useful techniques are listed here:

1. Use a market watch activity to get the latest COTS information, and collect COTS information from its current users to gain first hand COTS experience from its current user group.
2. Assess vendor supportability to address life cycle issues such as system refresh and maintenance.
3. Develop, instrument, and evaluate prototypes, benchmarks, simulations, or analytic models to analyze key performance parameters and tradeoffs.

A screening matrix or analytic hierarchy process is a useful and common approach to analyze collected evaluation data. The evaluation criteria and COTS candidates work as the columns and rows of the matrix respectively. The final score for a particular COTS candidate is the weighted sum of its points across all of the evaluation criteria. A ranking of all COTS candidates will be produced to help making the COTS decision. However, often a more focused analysis such as a gap analysis [13] or a business case analysis will be needed.

Besides the above major activities taking place during an assessment process element, there are some other management activities that are necessary and even critical to the assessment result. Such management activities are periodic assessment reviews, including the evaluating team, senior management, customers and the key users. The primary tasks for assessment review are to provide feedback to the evaluation process, to negotiate changes of requirements, design and COTS candidates, to adjust and refine the sets of evaluation criteria, weights, and business scenarios, and make final decisions. The final decisions establish different directions for exiting the COTS assessment process. We have identified the following three exit directions:

1. Full COTS solution is the best, which means there is a single COTS product or a combination of COTS products covering desired OC&P's;
2. A partial COTS solution is the best, which means that COTS product(s) only cover part of the OC&P's, and custom development is needed to meet the gap between COTS and OC&P's;
3. No COTS products are acceptable, which means that pure custom development is the optimal

solution, unless the stakeholders are willing to adjust unsatisfied OC&P's.

3.3. Tailoring Process Element

In more cases than not the COTS packages may have to be modified slightly in order to satisfy the OC & P's for the system [12]. If these modifications are directly supported within the COTS packages themselves, then this is considered tailoring activity. The tailoring process element is illustrated in Figure 3.3.

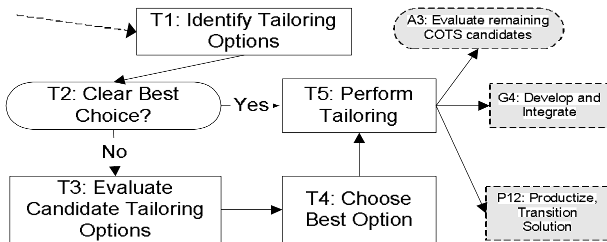


Figure 3.3. The Tailoring Process Element

Entry Conditions for Tailoring
While several COTS products may be tailored simultaneously (often by different people), the tailoring process element focuses on tailoring an individual COTS product. This product may be under consideration by the assessment element; be adapted for use as a glue code component; or be a fully assessed and ready to use product simply needing some specialization. Tailoring may be entered multiple times to accommodate multiple products or refinements to a previously tailored product.

T1: Identify Tailoring Options Identify the candidate options to be used in order to tailor the COTS system. A COTS product may have multiple tailoring options; in such cases the decision must be made as to what capabilities are required to be implemented by which option. As shown in Table 2, tailoring options may include GUI operations, parameter setting, or programming specialized scripts.

T2: Clear Best Choice? If a dominant tailoring COTS tailoring option is found then the developers can proceed to the development of the system. If there are still multiple tailoring options, the developers need to evaluate them in order to select the best option.

T3: Evaluate COTS Tailoring options. When there is no clear choice from T1 on which tailoring options to pursue, some further evaluation may be necessary. The typical evaluation considerations are: need to implement a particular design, the complexity of tailoring needed, need for adaptability and compatibility with other COTS tailoring choices, and available developer resources.

3.4. Glue Code Process Element

The intent of a glue code activity is to integrate COTS products as basic application components. In some fortunate cases, the combination of COTS components and application components being integrated or assessed will easily plug-and-play together. If not, some glue code needs to be defined and developed to integrate the components, and some evaluation may be necessary to converge on the best combination of COTS, glue code, and application code for the solution. A number of architectural approaches for using glue code or connectors to integrate COTS products have been developed [16, 17], but less has been done to work out the process for glue code development and its interactions with other CBA processes. Figure 3.4 illustrates the activities and decisions made when working with glue code.

Entry Conditions for Glue Code
The primary entry conditions are a set of components assessed to require glue code for successful joint operation, and a set of exit conditions. When entered from Assessment, the entry criteria also include the assessment criteria, and the exit conditions may be to develop just enough glue code to determine the most acceptable (if any) combination of components and glue code for a set of evaluation scenarios. When entered from the main decision framework (Fig. 3.1), the exit conditions will be to both determine the best combination of components and glue code, and develop and verify that the combination acceptably satisfies the system OC&P's.

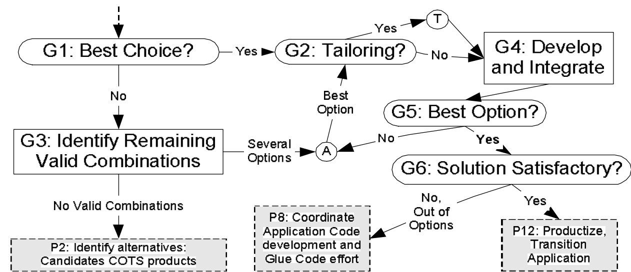


Figure 3.4. The Glue Code Process Element

G1: Best or Only Choice? In this initial decision point determine if there is a clear best choice of viable COTS package (and possible application code) combinations. If so then proceed to tailoring if necessary, otherwise there is a need to evaluate and assess viable combinations.

G3: Identify Valid Combinations. Often COTS packages are attractive from an OC&P standpoint, but cannot be made to feasibly (either technically or economically) interoperate. If none of the candidate packages can feasibly interoperate with respect to the current OC&P's, then more candidates must be generated or the OC&P's must be changed. When there are several valid combinations, the options are to be assessed to identify the best option.

G4: Develop and Integrate. This is the most complex part of the glue code process element. It involves many detailed activities all that must be carefully risk managed. Fortunately there exists a large body of knowledge on the subject of developing and integrating COTS with glue code (such as [16, 17]) and we will not detail them here. The basic tasks are:

1. Determine the interconnection topology options and minimize the complexity of interactions.
2. Evaluate and choose connector options (e.g. events, procedure calls, pipes, shared memory, DB, etc.).
3. Implement the connector infrastructure and develop the appropriate interfaces (simultaneously with application code interfaces if necessary as indicated within the application code process).
4. Integrate the components.

G5: Best Option? It may be that the G4 step produces poorer integrated performance than expected. If so, the G5 step determines whether one of the previously-rejected combinations may be better.

5. Conclusions

The fraction of projects that are COTS-based applications (CBA's with over 30% of end-user functionality provided by COTS and over 10% of development effort devoted to COTS considerations) is rapidly increasing in many application sectors. A 5-year longitudinal analysis of similar small e-services applications showed a growth from 28% CBA's in 1997 to 60% in 2001.

For samples of both small and large CBA's we have analyzed, most COTS-specific effort was devoted to COTS Assessment (A), Tailoring (T), or Glue code (G) activities. There is no one-size-fits-all distribution of A, T, and G effort, although there are some common patterns and significant correlations (e.g., a -.92 negative correlation between amount of Tailoring effort and Glue code effort).

Not only waterfall processes, but also standard object-oriented, UML-based processes have significant difficulties in dealing with the uncontrollable COTS architecture constraints, COTS dynamism, COTS uncertainty, and concurrency of activities involved in developing CBA's.

Our CBA project analysis found that for the most part, "where the effort happens, there the process happens." We also found that the Assessment, Tailoring, and Glue code activities followed similar processes for these elements. These A, T, and G process elements, and a custom application-code construction process element (C), could be composed into an overall process decision framework for CBA's.

However, there was also no one-size-fits-all path through the decision framework. In fact, most CBA processes we have analyzed are dynamic and concurrent, and the process elements need to be reentrant and recursive.

7. References

[1] C. Abts, B. Boehm, and E. Bailey Clark, "COCOTS: A Software COTS-Based System (CBS) Cost Model," *Proceedings, ESCOM 2001*, April 2001, pp. 1-8.
[2] C. Albert and L. Brownsword, "Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview," CMU-SEI-2002-TR-009, July 2002.
[3] R. Balzer, "Living with COTS," *Proceedings, ICSE 24*, May 2002, p. 5.
[4] B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, "Using the WinWin Spiral Model: A Case Study," Computer, July 1998, pp. 33-44.
[5] B. Boehm, "A Spiral Model of Software Development and Enhancement," Computer, May 1988, pp. 61-72.
[6] B. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000.
[7] L. Brownsword, P. Oberndorf, and C. Sledge, "Developing New Processes for COTS-Based Systems," Software, July/August 2000, pp. 48-55.
[8] M. Morisio, C. Seaman, A. Parra, V. Basili, S. Kraft, and S. Condon, "Investigating and Improving a COTS-Based Software Development Process," *Proceedings, ICSE 22*, June 2000, pp. 32-41.
[9] V. Basili and B. Boehm, "COTS Based System Top 10 List," Computer, May 2001, pp 91-93.
[10] B. C. Meyers and P. Oberndorf, *Managing Software Acquisition: Open Systems and COTS Products*, Addision Wesley, 2001.
[11] G. Benguria, A. Garcia, D. Sellier, and S. Tay, "European COTS Working Group: Analysis of the Common Problems and Current Practices of the European COTS Users," *COTS-Based Software Systems (Proceedings, ICCBSS 2002)*, Springer Verlag, 2002, J. Dean and A. Gravel (eds.), pp. 44-53.
[12] D. Port, J. Bhuta, Y. Yang, B. Boehm, "Not All CBS Are Created Equally: COTS Intensive Project Types," *Submitted to ICCBSS 2002.*
[13] C. Ncube and J. Dean, "The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Components," *COTS – Based Software Systems* J. Dean and A. Gravel (eds.), Springer Verlag, 2002, RP-176-187.
[14]. N. Maiden, H.Kim, and C. Ncube, "Rethinking Process Guidance for Selecting Software Components," *COTS-Based Software Systems*, J.Dean and A.Gravel (eds.), Springer Verlag, 2002, pp.151-164.
[15] S.Comella- Dorda, J.Dean, E.Morris, and P.Oberndorf, "A Process for COTS Software Product Evaluation," *COTS -Based Software Systems*, J.Dean and A.Gravel (eds.), Springer Verlag,2002, pg. 86-96
[16]. N. Medvidovic, R. Gamble, and D. Rosenblum, "Towards Software Multioperability: Bridging Heterogeneous Software Interoperability Platforms,"

Proceedings, Fourth International Software Architecture Workshop, 2000
[17] L.Davis and R. Gamble, "Identifying Evolvability for Integration," *COTS-Based Software Systems*, J.Dean and A. Gravel (eds.) Springer Verlag, 2002, pp.65-75

WinWin Spiral Approach to Developing COTS-Based Applications

EDSER-5 Position Paper

Barry Boehm, Dan Port, Ye Yang
*University of Southern California, Texas A&M University**
{boehm, dport, yey}@cse.usc.edu

Abstract

Data collected from five years of developing e-service applications at USC-CSE reveals that an increasing fraction have been commercial-off-the-shelf (COTS)-Based Application (CBA) projects: from 28% in 1997 to 60% in 2001. Data from both small and large CBA projects show that CBA effort is primarily distributed among the three activities of COTS assessment, COTS tailoring, and glue code development and integration, with wide variations in their distribution across projects. We have developed a set of data-motivated composable process elements, in terms of these three activities, for developing CBA's as well an overall decision framework for applying the process elements. We present a real-world example showing how it operates within the WinWin Spiral process model generator to orchestrate, execute, and adapt the process elements to changing project circumstances.

1. Definitions and Context

1.1 Definitions

We adopt the SEI COTS-Based System Initiative's definition [7] of a *COTS product*: A product that is:

- Sold, leased, or licensed to the general public;
- Offered by a vendor trying to profit from it;
- Supported and evolved by the vendor, who retains the intellectual property rights;
- Available in multiple identical copies;
- Used without source code modification.

We also follow the SEI in defining a *COTS-Based System* very generally as "any system, which includes one or more COTS products." This includes most current systems, including many which treat a COTS operating system and other utilities as a relatively stable platform on which to build applications. Such systems can be considered "COTS-based systems," as most of their executing instructions come from COTS products, but COTS considerations do not affect the development process very much.

To provide a focus on the types of applications for which COTS considerations do affect the development process, we define a *COTS-Based Application* as a system for which at least 30% of the end-user functionality (in terms of functional elements: inputs, outputs, queries, external interfaces, internal files) is provided by COTS products, and at least 10 % of the development effort is devoted to COTS considerations. The numbers 30% and 10% are not sacred quantities, but approximate behavioral CBA boundaries observed in the application projects. There was a significant gap observed in COTS-related effort reporting. The projects observed either reported less than 2% or over 10% COTS-related effort, but never between 2-10%.

In our six years of iteratively defining, developing, gathering project data for, and calibrating COCOTS cost estimation model, we identified four primary sources of project effort due to CBA development considerations. These are defined in COCOTS as follows:

- COTS *Assessment* (A) is the activity whereby COTS products are evaluated and selected as viable components for a user application.
- COTS *Tailoring* (T) is the activity whereby COTS software products are configured for use in a specific context. This definition is similar to the SEI definition of "tailoring" [10].
- COTS Glue Code (G) development and integration is the activity whereby code is designed, developed, and used to ensure that COTS products satisfactorily interoperate in support of the user application.

1.2 CBA Growth Trend

An increasing fraction of CBA projects have been observed in over five years' USC-CSE e-services project data. As seen in figure 2.1, the CBA fraction has increased from 28% in 1997 to 60% in 2001.

Major considerations for adopting COTS products in these projects are: 1) the clients' request, 2) the schedule constraint, 3) compliance with organization standards, and 4) the budget constraint. The primary reason for the growth in COTS content has, however, been the large increase in the number of COTS products providing application functions. In 1997, most of the teams were programming their own search engines and Web crawlers, for example; by 2001 these functions were being accomplished by COTS products.

Some of our USC-CSE affiliates have reported similar qualitative trends, but this is the first quantitative data they and we have seen on the rate of increase of CBA projects under any consistent definition and in any application sector (e-services applications probably have higher rates of increase than many other sectors). We have experienced many notable effects of this increase: for example, programming skills are necessary but not sufficient for developing CBA's (see also 8,9,10,11]).

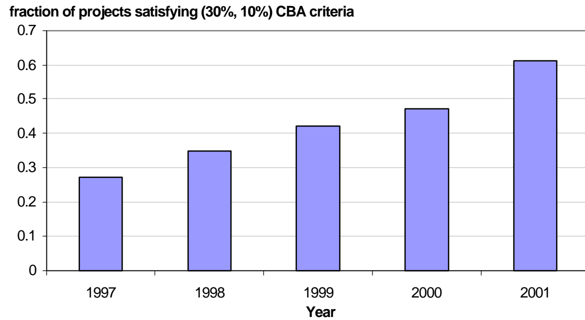


Figure 1.2 CBA Growth in Small E-Service Projects

2. The CBA Process Decision Framework

Figure 2.1 presents the dominant decisions and activities within CBA development as abstracted from our observations and analysis of USC e-services and CSE- affiliate projects. This represents the overall CBA decision framework that composes the assessment, tailoring, glue code, and custom code development process elements within an overall development lifecycle.

Some explanation of Figure 2.1 is in order. The CBA process is undertaken by "walking" a path from "start" to "Non-CBA Activities" that connects (via arrows) activities as indicated by boxes and decisions that are indicated by ovals. Activities result in information that is passed on as input to either another activity or used to make a decision. Information follows the path that best describes the activity or decision output. Only one labeled path may be taken at any given time for any particular walk; however it is possible to perform multiple activities simultaneously (e.g. developing custom application code and glue code, multiple developers assessing or tailoring).

The small circles with letters A, T, G, C indicate the assessment, tailoring, glue code, and custom code development process elements respectively. With the exception of the latter, each of these areas will be expanded and elaborated in the sections that follow. Each area may enter and exit in numerous ways both from within the area itself or by following the decision framework of Figure 2.1. In addition, this scheme was developed from and is consistent with the CBA activity distributions of Figures 2.3. In particular, only (and in fact all) "legal" distributions are possible (e.g. that all distributions have assessment effort is consistent with all paths in the framework initially passing through the assessment element (or area "A"). We now summarize the less obvious aspects of each process area.

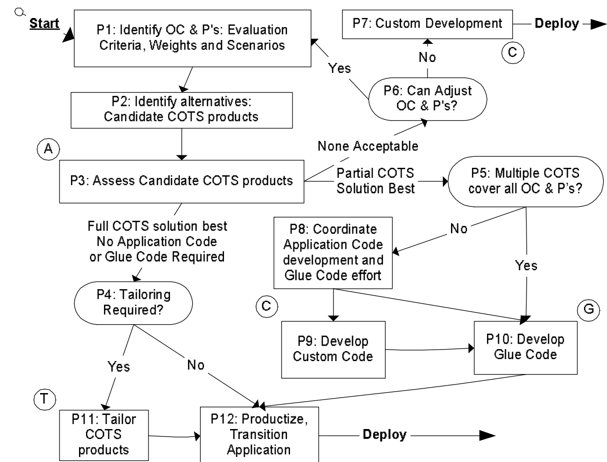


Figure 2.1. CBA Effort Decision Framework

P1: Identify OC&P's: Evaluation Criteria, Weights and Scenarios. This is the entrance to the CBA process where the initial evaluation attributes and desired operational outcomes for the application are established. Risk considerations, stakeholders' priority changes, new COTS releases and other dynamic considerations may significantly alter the objectives, constraints, and priorities (OC&P's). In particular, if no suitable COTS packages are identified, the stakeholders may change the OC&P's and the process is started over with these new considerations.

P2: Identify alternatives: Candidate COTS products. This and activity P1 establish the entry conditions for an Assessment activity.

P5: Multiple COTS cover all OC & P's? If a combination of COTS products can satisfy all the OC&P's, they are integrated via glue-code. Otherwise, COTS packages are combined to cover as much of the OC&P's as feasible and then custom code is developed to cover what remains.

P6: Can Adjust OC & P's?
When no acceptable COTS products can be identified, the OC&P's are re-examined for areas that may allow more options. Are there constraints and priorities that may be relaxed that have eliminated some products from consideration? How firm are the objectives and if adjusted slightly will it enable consideration of more products? Are there analogous areas in which to look for more products and alternatives?

P8: Coordinate Application Code development and Glue Code effort. Custom developed components must eventually be integrated with the chosen COTS products. The interfaces will need to be developed so they are compatible with the COTS products and the particular

glue code connectors used. This means that some glue code effort will need to be coordinated with the custom development.

3. Example WinWin Spiral Approach to CBA Development

3.1. Elaborated WinWin Spiral Model

Figure 3.1 provides a more detailed and concise version of the Win Win Spiral Model than that presented in [5]. It returns to the original four segments of the spiral, and adds stakeholders' win-win elements in appropriate places. It also emphasizes concurrent product and process development, verification and validation; adds priorities to stakeholders' identification of objectives and constraints; and includes the LCO, LCA, and IOC anchor point milestones [19] also adopted by the Rational Unified Process.

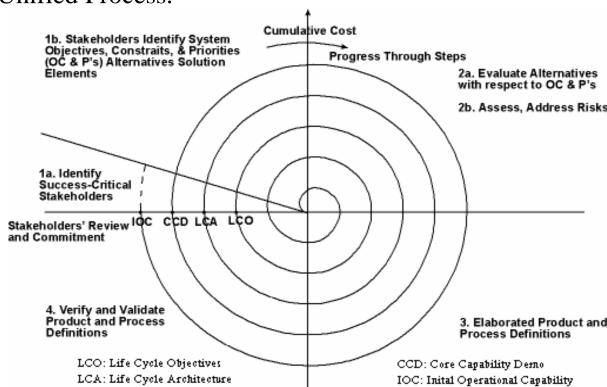


Figure 3.1. Elaborated WinWin Spiral Model

3.2. Example CBA: Oversize Image Viewer

One of the USC e-services COTS-based applications involved the development of a viewing capability for oversized images. The original client needed a system to support viewing of digitized collections of old historical newspapers, but other users became interested in the capability for dealing with maps, art works and other large digitized images. The full system capability included not just image navigation and zoom-in/zoom-out; but image catalog and metadata storage, update, search, and browse; image archive management; and access administration capabilities.

Several COTS products were available for the image processing functions, each with its strengths and weaknesses. None could cover the full system capability, although other COTS capabilities were available for some of these. As the initial operational capability (IOC) was to be developed as a student project, its scope needed to be accomplished by a five–person development team in 24 weeks. The application described in the next section makes some small simplifications of the project for the sake of brevity, but the overall COTS decision sequence and spiral cycles happened largely as described.

3.3. Applying the Decision Framework and the WinWin Spiral Model

The process description provided here for the Oversize Image Viewer (OIV) project covers the project's first three spiral cycles. Each cycle description begins with its use of the WinWin Spiral Model, as the primary sequencing of tasks is driven by the success-critical stakeholders' win conditions and the project's major risk items.

The OIV process description for each cycle then discusses its use of the CBA Process Decision Framework and its process elements. It shows that the framework is not used sequentially, but can be re-entered if the Win Win Spiral risk patterns cause a previous COTS decision to be reconsidered. The resulting CBA decision sequence for the OIV project was a composite process, requiring all four of the Assessment, Tailoring, Glue Code, and Development process elements.

Table 1 provides a spiral model template that is an update of the template used in the original spiral model paper [5]. It shows the major spiral artifacts and activities in the OIV project's first three spiral cycles. The discussion below indicates how these were determined by the major stakeholder OC&P's and project risk items.

3.3.1. Spiral Cycle 1

The original client was a USC librarian whose collections included access to some recently-digitized newspapers covering the early history of Los Angeles. Her main problem was that the newspapers were too large to fit on mainstream computer screens. She was aware that some COTS products were available to do this. She wanted the student developer team to identify the best COTS product to use, and to integrate it into a service for accessing the newspapers' content, covering the full system capability described in section 4.2 above. Lower priorities involved potential additions for text search, usage monitoring, and trend analysis.

Her manager, who served as the customer, had two top-priority system constraints as her primary win conditions. One was to keep the cost of the COTS product below $25K. The other was to get reasonably mature COTS products with at least 5 existing supported customers.

The student developer team's top-priority constraint was to ensure that the system's Initial Operational Capability (IOC) was scoped to be developable within the 24 weeks they had for the project.

The team quickly used these top-priority constraints to filter out two COTS products: system XYZ was too expensive, and system ABC had only one beta-test customer. The other two OIV COTS products, ER Mapper and Mr. SID, had different user interfaces; the major risk was to select one that users would subsequently find unacceptable. This risk was addressed by exercising the two products; this stage of the COTS assessment concluded that ER Mapper had considerably

stronger performance and image navigation characteristics than Mr. SID. Mr SID's main advantage was that it ran on Windows, Unix, and Macintosh platforms, while ER Mapper was only running on Windows. As the client had a Windows-based operation, ER Mapper was identified as the best candidate. Plans were made to tailor it for the overall product solution, and integrate it with other COTS and/or application code, as ER Mapper was not a complete application solution for such functions as cataloguing and search.

When the customer reviewed these plans, however, she felt that the investment in a campus OIV capability should also benefit other campus users, some of whom worked on Unix and Macintosh platforms. She committed to find representatives of these communities to participate in a re-evaluation of ER Mapper and Mr. SID for campus-wide OIV use. The client and developers concurred with this revised plan for spiral cycle 2.

Use of the CBA Decision Framework in Cycle 1

The first three steps of spiral cycle 1 in Table 1 (Stakeholders, OC&P's, Alternatives) include COTS products as alternatives and establish the preconditions (top-level evaluation criteria, weights, and scenarios; candidate COTS products) for entering the CBA Assessment decision framework in Figure 2.1 and 3.2. Spiral step 4 (Evaluation in Table 1) establishes the entry into *Assessment* in Figures 3.1 and 3.2.

	Cycle 1	Cycle 2 (LCO)	Cycle 3 (LCA)
Stakeholders	Developer, customer, library-user client, COTS vendors	Additional user representatives (Unix, Mac communities)	Additional end-users (staff, students) for usability evaluation
OC&P's	Image navigation, cataloguing, search, archive and access administration COTS cost ≤ $25K, ≥ 5 user organizations IOC developed, transitioned in 24 weeks	System usable on Windows, Unix, and Mac platforms	Detailed GUI's satisfy representative users
Alternatives	ER Mapper, Mr SID, Systems ABC, XYZ	ER Mapper, Mr SID	Many GUI alternatives
Evaluation; Risks	XYZ > $25K; ABC < 5 user org's ER Mapper, Mr SID acceptable Risk picking wrong product without exercise	ER Mapper Windows-only; plans to support Unix, Mac; schedule unclear Mr SID supports all 3 platforms Risk of Unix, Mac non-support	Risk of developing wrong GUI without end-user prototyping Mr SID/MY SQL/Java interoperability risks
Risk Addressed	Exercise ER Mapper, Mr SID	Ask ER Mapper for guaranteed Unix, Mac support in 9 months	Prototype full range of system GUI's, Mr SID/My SQL/Java interfaces
Risk Resolution	ER Mapper image navigation, display stronger	ER Mapper: no guaranteed Unix, Mac support even in 18 months	Acceptable GUI's, Mr SID/My SQL/.Java interfaces determined
Product Elaboration	Use ER Mapper for image navigation, display	Use Mr SID for image navigation, MySQL for catalog support, Java for admin/GUI support	Develop production Mr SID/My SQL/Java glue code
Process Elaboration	Tailor ER Mapper for library-user Windows client	Prepare to tailor Mr SID, My SQL to support all 3 platforms	Use Schedule as Independent Variable (SAIV) process to ensure acceptable IOC in 24 weeks
Product Process	Customer: want campus-wide usage, support of Unix, Mac platforms ER Mapper runs only on Windows	Need to address Mr SID/My SQL/Java interoperability, glue code issues; GUI usability issues	Need to prioritize desired capabilities to support SAIV process
Commitment	Customer will find Unix, Mac user community representatives	Customer will buy Mr SID Users will support GUI prototype evaluations	Customer will commit to post-deployment support of software Users will commit to support training, installation, operations

Table 1. Spiral Model Application to

Following the Assessment Framework in Figure 3.2, the initial filtering step eliminated some candidates (XYZ and ABC), but not ER Mapper or Mr. SID. The risk assessment in Table 1 required the two COTS products to be exercised, which involved *Tailoring* to accommodate the newspaper image files, but not glue code at this point. The evaluation identified ER Mapper as the best OIV solution, but only as a partial solution for other needed functions such as cataloguing, search, and archiving.

Thus the Assessment process element (Figure 3.2) exits back to the overall CBA decision Framework (Figure 2.1) in the "Partial COTS solution best" direction. But it cannot proceed further until the Win Win Spiral process determines whether either applications code or added COTS products or both need to be developed for the rest of the application (a lower risk decision deferred to a subsequent spiral cycle).

However, spiral cycle 1 ended with a new decision to revisit *Assessment* with likely new OC&P's emerging from other-OIV-user stakeholders as evaluation criteria. Thus we can see that the CBA decision framework is not sequential, but needs to be recursive and reentrant depending on risk and OC&P decisions made within the Win Win Spiral process.

3.3.2. Spiral Cycle 2

With the new Unix and Mac OIV stakeholders, a new win-win set of OC&P's emerges, including not only Unix and Mac OIV usability but also interoperability with other selected COTS products on all three platforms. The new evaluation/COTS assessment confirmed that Mr. SID was usable on all three platforms, but that ER Mapper had only general plans for Unix and Mac versions.

When ER Mapper declined to guarantee early Unix and Mac versions, Mr. SID became the new choice for the OIV functions. Concurrent assessment of candidate COTS products for the non-OIV functions converged on MySQL for catalog database support and Java for GUI support. Although the initial evaluation indicated that these were interoperable with Mr. SID, a fully interoperable build-upon (vs. throwaway) prototype was scheduled to be developed and interoperability-verified in spiral cycle 3. The other outstanding risk identified was that the system's GUI needed prototyping with additional end-user representatives also planned for spiral cycle 3.

Spiral cycle 2 ended with a WinWin Spiral LCO (Life Cycle Objectives) milestone review. At the LCO review, all of the stakeholders agreed to support the commitments allocated to them in the plans.

Use of the CBA Decision Framework in Cycle 2

The new stakeholders and OC&P's in cycle 2 required the project to backtrack to the beginning of the Assessment process element in Figure 2.1 and 3.2. For the OIV function, ER Mapper was filtered out without further evaluation when it declined to guarantee early Unix and Mac versions. Some tailoring was required to verify that Mr. SID performed satisfactorily on Unix and Mac platforms.

Concurrently, *Assessment* filtering and evaluation tasks were being performed for the cataloguing and GUI functions.

This concurrency is a necessary attribute of most current and future CBA processes. Simple deterministic process representations are simply inadequate to address the dynamism, time-criticality, and varying risk/opportunity patterns of such CBA's. However, the Win Win spiral process provides a workable framework for dealing with risk-driven concurrency, and the composable CBA decision framework and process elements provide workable approaches for handling the associated CBA activities. The dynamism and concurrency makes it clear that the CBA process elements need to be recursive and reentrant, but they provide a much-needed structure for managing the associated complexity.

3.3.2. Spiral Cycle 3

The additional end-user stakeholder communities increased the risk of developing GUI's that were fine for some users and unsatisfactory to others. These risks were resolved by involving representative end users in exercising GUI prototypes for various cataloguing, search, and navigation functions. The major CBA processes involved the *Assessment* of detailed interoperability characteristics of Mr. SID, MySQL, and the GUI software on the Windows, Unix, and Mac platforms. This involved invocation of both the *Tailoring* and *Glue Code* process elements.

The other major risk was the fixed 24-week IOC development schedule. This was handled via the Schedule as Independent Variable (SAIV) process described in [18]. The SAIV process requires customers and users to prioritize their desired capabilities. The priorities are used to define a core capability clearly buildable within the fixed schedule, and to architect the application for ease of adding or dropping borderline-priority features. This approach was satisfactory to the stakeholders, and resulted in a successfully transitioned Initial Operational Capability at the end of the 24 weeks.

Use of the CBA Decision Framework in Cycle 3

The Assessment process for interoperability of Mr SID, My SQL, and the Java GUI components on the Windows, Unix, and Mac platforms did not involve a comparative evaluation of alternative COTS products, although alternatives would have been necessary in case one of the COTS products had proved completely inadequate. The interoperability assessment involved both tailoring of the COTS products for the three platforms and some glue code to (successfully) enable interoperability.

Subsequent spiral cycles to develop the core capability and the IOC did not involve further Assessment, but involved concurrent use of the Tailoring, Glue Code, and custom development processes.

3.4. Summary of CBA Decision Framework Use

The use of the CBA decision framework during the three spiral system definition cycles and the subsequent development activity can be summarized by the sequence A, T; (AA); A, (TG); (TGC). The first spiral cycle involved Assessment supported by Tailoring. The second cycle involved two concurrent pure Assessments for the OIV COTS choice and for the other COTS choices. The third cycle involved an interoperability Assessment supported by concurrent Tailoring and Glue Code processes. The final development activity involved concurrent Tailoring, Glue Code, and custom development processes.

4. Conclusions

Using the WinWin Spiral Model's risk-driven approach coupled with the CBA decision framework as a process model generator, however, enabled projects to generate appropriate combinations of A, T, G, and C process elements that best fit their project situation and dynamics. An extensive discussion of its application to an actual CBA project is provided as an example.

The resulting combinations of A,T,G, and C elements serve as a sort of genetic code for the projects CBA process which can be used to identify and compare it with other projects CBA processes. The analogy can be stretched too far, but it suggests several attractive directions for future research, such as determining how best to represent the concurrency and backtracking aspects; validating and refining effort distributions based on process elements; assessing the validity of the process elements and decision framework in other CBA sectors; and identifying common process element configurations, valid and invalid configurations, or large-grain CBA process patterns.

5. References

[1] C. Abts, B. Boehm, and E. Bailey Clark, "COCOTS: A Software COTS-Based System (CBS) Cost Model," *Proceedings, ESCOM 2001*, April 2001, pp. 1-8.

[2] C. Albert and L. Brownsword, "Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview," CMU-SEI-2002-TR-009, July 2002.

[3] R. Balzer, "Living with COTS," *Proceedings, ICSE 24*, May 2002, p. 5.

[4] B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, "Using the WinWin Spiral Model: A Case Study," Computer, July 1998, pp. 33-44.

[5] B. Boehm, "A Spiral Model of Software Development and Enhancement," Computer, May 1988, pp. 61-72.
[6] B. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000.
[7] L. Brownsword, P. Oberndorf, and C. Sledge, "Developing New Processes for COTS-Based Systems," Software, July/August 2000, pp. 48-55.
[8] M. Morisio, C. Seaman, A. Parra, V. Basili, S. Kraft, and S. Condon, "Investigating and Improving a COTS-Based Software Development Process," *Proceedings, ICSE 22*, June 2000, pp. 32-41.
[9] V. Basili and B. Boehm, "COTS Based System Top 10 List," Computer, May 2001, pp 91-93.
[10] B. C. Meyers and P. Oberndorf, *Managing Software Acquisition: Open Systems and COTS Products*, Addision Wesley, 2001.
[11] G. Benguria, A. Garcia, D. Sellier, and S. Tay, "European COTS Working Group: Analysis of the Common Problems and Current Practices of the European COTS Users," *COTS-Based Software Systems (Proceedings, ICCBSS 2002)*, Springer Verlag, 2002, J. Dean and A. Gravel (eds.), pp. 44-53.
[12] D. Port, J. Bhuta, Y. Yang, B. Boehm, "Not All CBS Are Created Equally: COTS Intensive Project Types," *Submitted to ICCBSS 2002.*
[13] C. Ncube and J. Dean, "The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Components," *COTS – Based Software Systems* J. Dean and A. Gravel (eds.), Springer Verlag, 2002, RP-176-187.
[14]. N. Maiden, H.Kim, and C. Ncube, "Rethinking Process Guidance for Selecting Software Components," *COTS-Based Software Systems*, J.Dean and A.Gravel (eds.), Springer Verlag, 2002, pp.151-164.
[15] S.Comella- Dorda, J.Dean, E.Morris, and P.Oberndorf, "A Process for COTS Software Product Evaluation," *COTS -Based Software Systems_*, J.Dean and A.Gravel (eds.), Springer Verlag,2002, pg. 86-96
[16]. N. Medvidovic, R. Gamble, and D. Rosenblum, "Towards Software Multioperability: Bridging Heterogeneous Software Interoperability Platforms," *Proceedings, Fourth International Software Architecture Workshop*, 2000
[17] L.Davis and R. Gamble, "Identifying Evolvability for Integration," *COTS-Based Software Systems*, J.Dean and A. Gravel (eds.) Springer Verlag, 2002, pp.65-75
[18] B.Boehm, D. Port, L. Huang and W. Brown, "Using the Spiral Model and MBASE to Generate New Acquisition Process Models: SAIV, CAIV, and SCQAIV," *Cross Talk*, January 2002, pp.20-25 (http://www.stsc.hill.af.mil/crosstalk)
[19] B. Boehm, "Anchoring the Software Process," *Software*, July 1996, pp. 73-82

Economic Risk-Based Management in Software Engineering: The HERMES Initiative

Stefan Biffl
Vienna Univ. of Technology
Inst. of Software Technology
A-1040 Vienna, Austria
Stefan.Biffl@tuwien.ac.at

Michael Halling
Johannes Kepler Univ. Linz
Systems Eng. & Automation
A-4040 Linz, Austria
mh@sea.uni-linz.ac.at

Paul Grünbacher
Johannes Kepler Univ. Linz
Systems Eng. & Automation
A-4040 Linz, Austria
gruenbacher@acm.org

Abstract

Developing software of high quality is both socially and economically critical. Nevertheless software projects are often managed badly without considering economic potential and constraints. The decision making process is often performed in an ad-hoc manner and approaches from business administration or operations research are rarely adopted. In Austria, we have recently been developing a research agenda that addresses these issues in an interdisciplinary research plan. This paper introduces and motivates this joint research initiative and identifies important issues needing attention.

1. Introduction

The pervasive impact of software on life in our society makes the capability to develop high-quality software a socially and economically relevant issue. However, approaches to the management of software engineering (SE) projects are often based on surprisingly simplistic assumptions, often just rules of thumb and lessons learned. While there are many projects documented that run well, there are also many reports on late, over-budget, and sometimes spectacularly disastrous projects. While (empirical) SE is good at generating knowledge on a technical level that has a clear use and is less dependent on context assumptions, the improvement on the management level lags behind, possibly due to more complex dependencies of this knowledge on project context. What is missing so far, is a profound exchange of knowledge and collaboration of SE with related research fields, namely business administration and operations research, in order to attack SE management problems in a more comprehensive way.

We have recently been developing a new research initiative in Austria. The proposed Joint Research Project (JRP) “Integrated Economic Risk-based Management in Software Engineering” (IERMSE, further called “Hermes”) aims at addressing some of these issues. Therefore, Hermes combines approaches from the disciplines software engineering, business administration, and operations research to tackle the key challenge of project management (PM) and quality management (QM), i.e., to help develop high-quality software in an economically efficient way. The key issues of the JRP are represented in the title: (a) ‘risk-based’ refers to project uncertainty and variability, which is often connected with mainly negative aspects of defects and loss, and will be completed with the positive side of risk: opportunities in SE projects and an orientation towards the added value of SE projects, processes and products; (b) ‘economic’ refers to the integration of economic points of views with the typical technical focus in SE; (c) ‘management in SE’ includes the full range of management from detail management of project activities to the large-scale management of multiple projects in a business unit. Hermes focuses on strategic proactive management (in contrast to reactive management for local short-term optimization) of multiple software projects in an uncertain and dynamic business environment.

The Hermes JRP is motivated by recent international initiatives, such as the workshop “Economics-Driven SE Research” (EDSER) or the workshop on “SE Decision Support“ (SEDECS) at the International Conference on SE and Knowledge Engineering both stating the need for the systematic integration of scientific economic approaches into SE.

2. Research Areas

Figure 1 presents an overview on the JRP research areas in three groups: (A) general concepts and methodologies for valuation and decision support; (B) management approaches for SE projects and processes; and (C) management support infrastructure.

Research areas A1 und A2 will provide a solid methodological foundation from management science as they develop advanced methods (A1) for the valuation of SE projects and (A2) for making key decisions (regarding

uncertainty, risk, multiple target criteria, and different preferences among stakeholders) in SE management for the framework process model steps of the projects in group B.

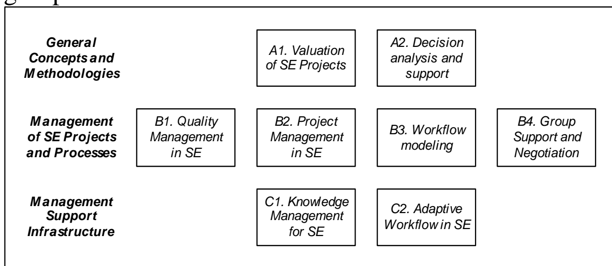


Figure 1. Hermes Research Areas

The research areas in group B form the center of the research initiative as they focus on key application aspects of SE management: (B1) quality management, (B2) project management, (B3) workflow modeling, (B4) group support and negotiation for team building – each with a specific framework process model that can describe the suitability of a range of methods for a particular SE project: from simple methods used in current practice to sophisticated scientific methods to be developed. The four framework process models in projects B1 to B4 allow defining capability levels for all models and methods used for a process step: (a) to assess the capability of models and methods used in current practice, (b) to rank candidates for 'best practice' approaches, and (c) to determine the need for scientific research in these areas. The framework process models also facilitate the empirical evaluation of new scientific methods and the dissemination of suitable methods into practice. Such a framework process model has to be compatible with commonly used SE process models, such as the V-model, the spiral model, or recent agile approaches. The assessment part of the frameworks should be compatible to wide-spread approaches, such as CMM(I) or SPICE: (a) to define capability/maturity levels for key process areas and (b) to allow gap analysis in specific project environments. Experience with existing assessment frameworks documents good results for improvement on technical aspects, but also a need for management support for SE projects, especially for multiple projects. We regard the approach of framework process models with several maturity levels for each step in the model as an excellent opportunity to achieve method development and application that is rooted in practice, supports a strong management vision, and allows stable growth on a clear path to scientific sound methods at a suitable pace for a business partner in practice.

The projects in group C develop advanced management support infrastructure techniques/tools for key areas needed in the processes of the projects in group B: (C1) knowledge management repository and (C2) adaptive workflow tool support.

3. Research Method

In the SE community the importance of empirical research to evaluate technical processes has been growing considerably. This is for example documented in the Empirical Software Engineering journal and an increasing number of empirical papers in top SE journals and conferences. The general research approach in the JRP is empirical validation of hypotheses generated from theory and practice according to the Quality Improvement Paradigm [1] as SE processes with effects that depend on project context cannot be evaluated solely with a theoretical argumentation but must be empirically evaluated.

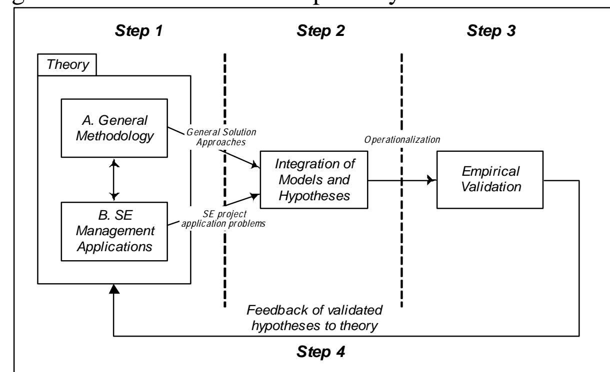


Figure 2. Research Approach

Figure 2 shows the four steps in the research approach: 1. initial theory foundation based on existing research; 2. operationalization of models and hypotheses – application of the initial theory foundation for modeling and simulation studies that prepare the design and project plans for empirical studies (e.g., simulation with prototypes, calibration with empirical data; feasibility studies); 3. conduct of empirical studies (validation of theory and simulation studies with empirical data); and 4. feedback of results from simulation and empirical studies to build an advanced SE theory.

In the initiative we will first focus on developing a detailed theoretical foundation for the applied methods (e.g., stable frameworks for reference processes) and identify solution packages for SE application problems. These solution packages are not necessarily integrated and get evaluated in environments with low risk and a reasonable return on investment (e.g., prototypes and feasibility studies).

Subsequently, we want to validate, refine, and integrate these solution approaches (processes and tools) using empirical and simulation techniques: e.g., evaluation of usability in large field studies. A major focus is also on the dissemination of proven solutions into practice and to spark applied research in industry.

The JRP goals rely on the one hand on advanced knowledge on economic valuation and economic/statistical decision theory and on the other hand on

knowledge on SE processes, negotiation, and SE project management: (a) one cannot take existing approaches in business administration or operations research and simply apply them to SE problems as SE problems differ significantly from traditional situations in Business Administration and Operations Research. Researchers in business administration and economics typically see a software development project as a R&D project, which is untypical and risky, and thus not routinely investigated. (b) SE research usually has a technical and practical focus and is less based on scientifically founded management methods. (c) Project managers usually have a focus on getting things done, rather than on science in general and particularly in SE.

To attain the goals of the proposed JRP there is a need for a critical mass of scientists with deep knowledge in a variety of research areas that would be unlikely to come together in unrelated individual small research projects. Also, to tackle these tasks the project needs researchers with a multi-science background to coordinate the different projects and translate between the research cultures. This joint JRP will enable the collaboration of established research groups, working so far on different aspects of the theory and application of the JRP research areas, in one organized cooperating group. It should also intensify international contacts and collaboration.

Table 1. Summary of Research Projects

Project	*Problem Description*	*Research Goals*
A1: Valuation of SE Projects	Strategic decision-making is often focused on cost and risk instead of value creation.	Apply valuation techniques from corporate finance to SE projects.
A2: Decision Analysis and Support	Characteristics of decisions in SE involve uncertainty, multiple criteria, different incentives of stakeholders, and dynamic environments. Current techniques to support decisions with these characteristics are not adapted to the specific requirements of SE and need to be modified.	Apply and extend multi-criteria, dynamic, and stochastic decision models; apply negotiation and auction techniques to deal with information asymmetries, incentive incompatibility and strategic behavior of stakeholders.
B1: Quality Management in SE	There is little information on the value and risks for different Quality Assurance (QA) techniques and their combination with respect to project context in a company or organization.	Extend and investigate existing QA techniques from a technical perspective with respect to project context; use methods from A1 and A2 to better evaluate and plan QA in different realistic project scenarios.
B2: Project Management in SE	Project management, including several coupled projects in a company, is based on rules of thumb rather than well-studied methods.	Extend project management techniques for a multi-project environment; apply results from A1, A2, and B3 to support key project management decisions.
B3: Workflow Modeling	Project plans for single projects are often unrealistic and expensive to maintain. PERT and GERT techniques provide only limited considerations for cost effectiveness under uncertainty.	Project plans that capture uncertainty in a realistic way for analysis and that are worthwhile to maintain. Activity modeling, resource management with stochastic processes; white-box view on project level based on results from A2.
B4: Group Support and Negotiation in SE	Software is developed in teams; existing research often focuses on individual engineers thus neglecting team issues and collaboration.	Develop methods and tools supporting software development teams; apply negotiation methods as proposed in A2 to support group decision making and allow mutually satisfactory solutions among stakeholders.
C1: Knowledge Management	Different characteristics of SE process models lead often to inefficient knowledge management.	Tools for knowledge management for supporting SE process models in projects B1to B4.
C2: Adaptive Workflow Management in SE	Uncertainty in SE processes demands for flexible work flow management system support, ad-hoc communication and recommendation facilities and a proper balance between pre-modeled workflows on the one hand and incremental planning and ad-hoc reactions on the other hand.	Provide comprehensive WFMS support for SE based on B3, particularly addressing the issues of adaptivity of SE processes, reuse and synthesis of SE process knowledge and ad-hoc collaboration and recommendation facilities.

4. Project Overview

This Section gives an overview of the projects of the proposed JRP Hermes. For each project we present key problems targeted in the JRP and the key research goals of each project. Table 1 gives an overview.

A1. Valuation of SE projects: Current state-of-the-practice and state-of-the-art in software engineering focuses often exclusively on cost issues for decision-making. The main advantage of costs is that they are, at least partly, easier to measure than benefits. However, if we study valuation concepts in business administration, we observe that the goal of all methods is to appropriately quantify the value of a project. Therefore we want to establish a value-oriented valuation approach in software engineering. Based on this value-oriented concept we aim at developing a more complete approach towards project risk management. [4][7][8][12]

A2. Decision analysis and support: Decision problems in software engineering have special characteristics, which distinguish them from traditional decisions problems dealt with in operations research. Therefore an important goal of this research project is to analyze decisions in SE in order to identify feasible optimization methods. As far as different methods are concerned our main focus lies on multi-criteria decision making. Another important dimension of SE decisions is that they usually influence very different stakeholders. Therefore an important part of decision support is to extend existing preference elicitation techniques. Further goals include theoretical support of group decisions and the development of negotiation methods for SE problems. [13][19]

B1. Quality management (QM): QM methods for risk reduction are an integral part of risk management and address mostly product and process risks. Currently there is a large number of quality assurance techniques, but little pragmatic guidance founded on sound theory on when to use which technique. Based on a framework process model, which allows to assess the QM capability of a project organization, we propose to investigate defect reduction techniques – such as formal technical review and testing approaches – as well as tool support options for these techniques in different application contexts to gather data for improved value-oriented QM planning considering not only the technical but also the economic point of view. [2][9][11][16]

B2. Project management: Current project planning in practice suffers from simplistic approaches (a) that lack practical support for modeling uncertainty and project interdependencies to help a project manager decide among several project options and (b) that are easy to maintain over the course of a project. We aim at project plan models that are based on information the project manager can provide, that are maintainable to project change in real project situations, and that support managers in applying methods developed in research areas A1 and A2. From such a value- and risk-oriented approach we expect more realistic plans that can be used for more effective project control and better decision-making. [6][17]

B3. Workflow modeling: This project focuses on modeling activities under variability and uncertainty to investigate the interrelationships of many work packages in a SE project for improved project control under uncertainty in day-to-day activities. Using UML and Petri Nets as modeling frameworks, we will study simulation and optimization techniques. Knowing that there is always a trade-off in modeling between the expressiveness of the model (i.e., the modeling power) and the model complexity (affecting the time to solve the problems) we are looking for efficient evaluation approaches. To achieve this goal, we have to identify problem classes and the appropriate choice of models and parameters in the solution methods. [18]

B4. Group support and negotiation in SE: Software development requires team work and collaboration of different experts belonging to the development team and external project partners. This research project aims at evaluating the dynamics of this team work and at providing tool support using the methods developed in projects A1 andA2. One specific and important aspect of this group support is negotiation because it enables project teams to discuss open issues, develop a shared vision of the project, and create a Win-Win situation for all team members. Therefore we propose to extend existing group support processes and tools for negotiation to other negotiation situations in a SE project, such as project planning, project controlling, risk monitoring, and post-mortem reviews for process improvement. Improving the quality of meetings and teamwork promises to effectively lower the overall project risk. [5][10][14][15]

C1. Knowledge management for SE: Knowledge management (KM) in SE focuses on managing and modeling resources of the software development process to provide useful feedback information or knowledge for the concerned actors (software engineers, end users, and project management). KM can be used for (a) significantly exert a strong influence on decreasing development costs, time to production, and increase software quality as well as (b) helping to deploy knowledge across distributed teams to compress development time frames.

C2. Adaptive workflow management in SE: Workflow management systems (WFMSs) are more and more used to make SE processes explicit and to enable their enactment by workflow engines, thus facilitating standardization and reuse and increasing productivity and efficiency. To cope with the varying degree of uncertainty inherent in every SE project, workflow management systems should not only be able to provide pre-modeled and potentially

automated workflows but should also be adaptive allowing incremental planning and ad-hoc reactions to changing situations which is not fully supported by existing approaches. The emphasize of this project is on a comprehensive WFMS support for SE, particularly addressing the issues of adaptivity of SE processes, reuse and synthesis of SE process knowledge and ad-hoc collaboration and recommendation facilities.

5. Issues

We decided to present this research proposal at the EDSER workshop in order to discuss the following issues with the workshop participants:

1. Is the proposed research agenda complex enough? Or, did we miss any important field that should be represented in the project structure? For example, what about psychology in order to understand and motivate team members of software development teams appropriately.

2. Is the proposed research agenda too complex? Should we remove some fields/projects because they are not required for developing better solutions? Will software engineering maintain its ad-hoc and intuitive characteristic because it simply is rather an art than a craft?

3. Is it worth investing effort into developing better software? Will the market appropriately value high/appropriate quality, or will other approaches towards software engineering like open source projects solve the problem of quality? The proposed projects will result in well-founded methods/process to optimize decision-making. However this will take some time and increase development effort and time. Will their be an incentive for software development companies to use “better” processes?

4. Why are SE management techniques still mainly based on simplistic assumptions and intuition

5. What are the reasons for the lack of methodological foundation in the area of SE, in particular SE economics? For comparable areas like for example corporate valuation (i.e., where the value of entire corporations is modeled) and credit risk estimation (i.e., where the risk of bankruptcy is estimated for a company) a large body of theory exists. However, little effort has so far been invested in well-defined theory in the area of software engineering.

6. How can we best transfer economic methods and tools into SE? What are the pitfalls?

7. Do you know about related research projects that may provide valuable input to this research?

6. Conclusions

In this paper we have briefly reported on a new research initiative that aims at integrating economic theories and approaches into SE to improve decision-making in real-world situations. This initiative is a first step to spark and integrate international research. We invite the EDSER community to share their ideas, suggestions, and concerns.

References

[1] V. Basili, G. Caldiera, H.D. Rombach, "Experience Factory," in J. J. Marciniak, ed., *Encyclopedia of Software Engineering*, John Wiley & Sons: 1994, 469--476.

[2] St. Biffl, “Hierarchical Economic Planning of the Inspection Process”, *Proc. of the 3rd Int. Workshop on Economics-Driven Software Engineering Research (EDSER-3)* IEEE CS Press, May 2001.

[3] B. Boehm “Software Risk Management: Principles and Practices”, *IEEE Software*, Jan. 1991, p.32-41

[4] B. Boehm, *Software Engineering Economics*, Prentice Hall, 1984.

[5] B. Boehm, P. Grünbacher, R.O. Briggs, Developing Groupware for Requirements Negotiation: Lessons Learned, *IEEE Software*, May/June 2001, 46-55.

[6] J. Bosch, Software product lines: Organizational alternatives. In *Proc. of the 23rd International Conference on Software Engineering,* IEEE Computer Society Press, 2001, pp. 91–100..

[7] R. Brealey, S. Myers, “*Principles of Corporate Finance*”, 6th Edition, McGraw-Hill, 2000.

[8] H. Erdogmus, J. Favaro; Keep Your Options Open: Extreme Programming and Economics of Flexibility; in: *XP Perspectives*; eds. Marchesi M.; Succi G.; Addison-Wesley; 2002.

[9] J. Favaro, P. Favaro; When the Pursuit of Quality Destroys Value; *IEEE Software,* May 1996.

[10] J. Favaro; Managing Requirements for Business Value; *IEEE Software* March 2002.

[11] M. Halling, St. Biffl, P. Grünbacher, An Economic Approach for Improving Requirements Negotiation Models with Inspection, to appear: *Requirements Engineering Journal*, Springer, 2003.

[12] W. Harrison, D. Raffo, J. Settle, “Process Improvement as a Capital Investment: Risks and Deferred Paybacks”, *Proc. of the Pacific Northwest Software Quality Conference (PNSQC)*, Portland, Oregon, October, 1999.

[13] R. Keeney, H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. J. Wiley & Sons, New York, 1976.

[14] G.E. Kersten, S.J. Noronha, “WWW-based Negotiation Support: Design, Implementation, and Use”. *Decision Support Systems 25,* , 1999, pp. 135-154.

[15] N. Medvidovic, P. Grünbacher, A: Egyed, “Bridging Models across the Software Lifecycle”, to appear: *Journal of Systems and Software*, 2003.

[16] D. Port, M Halling, R. Kazman, S. Biffl, “Strategic Quality Assurance Planning”, *Proc. of the 4th Int. Workshop on Economics Driven Software Engineering Research (EDSER-4) at the Int. Conf. on Software Engineering*, 2002.

[17] K. Schmid, “An Economic Perspective on Product Line Software Development; First Workshop on Economics-Driven Software Engineering Research”, *Proc. of the 1st Int. Workshop on Economics Driven Software Engineering Research (EDSER-1) at the Int. Conf. on Software Engineering,* May 1999.

[18] K. Sullivan, P. Chalasani, S. Jha, V. Sazawal, "Software Design as an Investment Activity: A Real Options Perspective," in *Real Options and Business Strategy: Applications to Decision Making*, (L. Trigeorgis, ed.), Risk Books, 1999.

[19] R. Vetschera, "Multi-Criteria Agency Theory", *Journal of Multi-Criteria Decision Analysis* 7, 1998, pp. 133-143.

Software Dependability Risks and the Insurance Process

Dan Port
University of Hawaii
Department of Information Management
dport@hawaii.edu

LiGuo Huang
University of Southern California
Department of Computer Science
liguohua@usc.edu

ABSTRACT
The concept of software dependability is intuitively understood but difficult to quantify into a constructive model. In this paper, we present a dependability risk model to convey that dependability is a *relative economic* measure of the dependability attribute risk factors. A system that is "undependable" is "risky" relative to the value (or benefit) of items at risk that is expected from that system. We also propose the questions that our dependability risk model can answer and provide a simple example.

Keywords
dependability risk insurance

1. Introduction
The concept of software dependability is intuitively understood but difficult to quantify into a constructive model. The concept is analogous to hardware dependability in which the goal is to provide a measure of assurance that a system will not fail to perform in an expected manner. Dependability extends beyond system reliability (the focus of which is on how likely a system will cease to function altogether) to an aggregate of *dependability attributes* [1] that include the following:

Robustness: reliability, availability, survivability, recoverability

Protection: security, safety

Quality of Service: accuracy, fidelity, performance assurance, maintainability

Integrity: correctness, verifiability

Availability means the readiness for correct service. Reliability means the continuity of correct service. Survivability means that a software system can repair itself or degrade gracefully to preserve as much critical capabilities as possible in the face of attacks and failures. **Recoverability** means a software system can recover itself from faults. **Security** means absence of unauthorized access to, or handling of, system state. It is the concurrent existence of a) availability for authorized users only, b) confidentiality, and c) integrity with 'improper' taken as meaning 'unauthorized'. **Safety** means absence of unauthorized disclosure of information. **Maintainability** means the ability to undergo repairs and modifications. **Integrity** means absence of improper system state alternations. Integrity is a prerequisite for availability, reliability and safety.

These attributes are mostly compatible and synergetic, but it is not uncommon for there to be some conflicts and tradeoffs. Some examples of this might be within a system that makes use of distributed information to increase the survivability of its data in the event that a particular data location is destroyed. In such a system security is traded off (or complicates the dependability with respect to data security) for survivability as the data must be accessible in multiple locations thereby increasing the number of entry points for possible security breaches. Other examples are building a fail-safe system whereby safety is balanced with quality of service, or employing a graceful degradation policy where survivability now contrasts with quality of service.

The dependability attributes described above only tell part of the story. The degree to which each attribute applies is relative to the expected outcome when the system is subject to negative events (e.g. a component fails, security is violated, data is incorrect).

Our view is that dependability is a *relative economic* measure of the dependability attribute risk factors. A system that is "undependable" is "risky" relative to the value (or benefit) of items at risk that is expected from that system. To illustrate this, consider system faults that result from undependable software (that is, faults with respect to the dependability attributes listed previously). Software faults/defects incur economic losses over time (e.g. IUM's, reputation, reduced sales, opportunities, etc.) with relative to the dependability attributes. For example if a sales system that people depend on to process customer sales is unavailable, there will assuredly be a measurable economic loss. Such was the case when the AT&T business sales

system became bottlenecked and the loss was in thousands of dollars per minute [2].

Ultimately for any system, a return on investment (ROI) is expected due to the continued (dependable) operation of that system. In this light, we invest in dependability as an *insurance policy* in which we make an initial investment (such as fault tolerance) along with continual premiums (e.g. security policies, contingency measures, backup systems) to *insure* against non-achievement of an acceptable ROI. This involves a complex and dynamic interplay of cost, risk, and value with respect to the dependability attributes and operational constraints and priorities. We propose a possible model to help analyze this perspective along with some potentially interesting questions and some initial examples.

2. Dependability Risk Model

Clearly value is created over the time a system operates, and a respectable return on investment (ROI) is expected due to the continued (dependable enough) operation of that system. Risks within the dependability attributes reduce this expected ROI and thus the key to a "dependable" system is to ensure the total investment for the dependable operation of the software system and the expected losses due to dependability risks do not outweigh the expected gains due to flawless operation of the system.

We are looking at risk models rather than the traditional reliability models for the following reasons:

- Risk models may be more empirically accessible than other models
- Overall dependability risk is simply the sum of dependability attribute risks, regardless of dependencies
- There is a well-established theory and practice of risk assessment and management to draw upon
- It matches well with intuitive concepts of dependability

There are many possible value-risk models that may apply to dependability risk. A particularly attractive and straightforward one we have been considering is the "insurance" model [3]. It is summarized as:

$$X(t) = \sum_{i=1}^{N_V(t)} V_i(t) - \sum_{j=1}^{m} I_j - \sum_{i=1}^{N_R(t)} R_i(t)$$

where $V_i(t)$ is a random variable distributed according to the value of item *i* "at risk" at time *t*, $N_V(t)$ is the number of *value events* given totally dependable operation of the system up to time t, I_j is the amount invested to achieve the desired level for each of *m* dependability attributes during the development of the system, $N_R(t)$ is the number of dependability faults up to time t, and $R_i(t)$ is a random variable distributed according to the potential loss (risk) of each type of system failure *i* up to time *t*. The insurance process has parameters that can be estimated with empirical distributions gathered from analogous systems, with the exception of $V_i(t)$, which can be estimated via an *earned-value* [4] model.

3. Dependability Problems To Be Answered by Dependability Risk Model

The dependability risk model can be used to analyze cost-benefit issues such as how dependable is dependable enough. For instance, it helps answer basic dependability questions such as:

- (How much is enough?)
    - Given an investment amount and value earned over time, how <u>low</u> can the dependability risk be before an expected ROI is unachievable?
    - Can we find a minimum *I* (amount of investment) that insures $X(t)$ will never be negative?
- (Risk of Catastrophe) Is there a high risk of an effectively infinite loss (Including low probability, high loss events)?
- (Risk of Recession) Is there a time *t* in which $X(t)$ will <u>ever</u> be negative?
- (Risk of Ruin) Is there a time *t* after which $X(t)$ will <u>always</u> be negative?
- (Risk of Decay) Will cumulative small losses force $X(t)$ to zero over time?

One of its applications is to map various dependability approaches to dependability attributes and benchmark them with respect to risk/value. Therefore we can compare the relative dependability attribute risks and effectiveness of dependability approaches on risk reduction. Defect and fault seeding is often considered for gathering empirical estimates of defect and fault populations.

Our goals are to develop dependability risk models based on development-time and runtime characteristics, evaluate use of dependability risk models as means of determining extent to invest in dependability (e.g. defect removal, dependability mechanisms such as fault handling or a particular architecture style) with respect to system value and risks. We hope that by implementing and continuously monitoring dependability risk models, developers will gain

insight into when and how much to invest in dependability (i.e. fault removal, tolerance mechanisms, prevention, etc.).

The heading of subsections should be in Times New Roman 12-point bold with only the initial letters capitalized. (Note: For subsections and subsubsections, a word like *the* or *a* is not capitalized unless it is the first word of the header.)

4. A Simple Example

Let us consider a painfully oversimplified example to illustrate the analytical considerations described above. For this, say that the value achieved is constant, or $V_i(t) = c$ for some constant *c,* and that the value events occur continuously over time (this is of course grossly oversimplified for any practical software system). We will assume that the number of faults up to time *t* is Poisson with intensity α and that $R_i(t)$ is identically exponentially distributed and independent of *t* with mean μ and variance σ^2. Further, let us ignore initial investment costs for dependability. The dependability risk model will be:

$$X(t) = ct - \sum_{i=1}^{N_R(t)} R_i$$

Let us consider a simplified "how much is enough" question. The expected "profit" for the above model will be $E[X(t)] = (c - \alpha\mu)t$ so clearly to achieve a desirable ROI $c > \alpha\mu$. However, this does not answer the critical question of when a particular ROI will be achieved. Let us deal with the question of how tolerance of dependability risks. That is, how is there a point at which we lose so much value that we should give up on the system? This roughly translates into the probability $\psi(u) = P\{u + X(t) < 0 \text{ for some } t > 0\}$ where *u* is the tolerance value desired. Under these conditions (assuming $c > \alpha\mu$) it can be shown that:

$$\psi(u) = \frac{\alpha\mu}{c} e^{-u\left(\frac{c - \alpha\mu}{c\mu}\right)}$$

As one would expect, larger tolerance of dependability risks means it is less likely that such tolerance will be exceeded at some point.

5. A Proposed Application of the Dependability Risk Model: SCRover

USC is developing the Inspector SCRover (ISCR) as part of the High Dependability Computing Program testbed. We are currently gathering data (e.g. empirical value and dependability fault distributions) in order to apply our dependability model as part of this effort. ISCR is a robot that is developed to assist in public safety situations. These situations might arise after an explosion or earthquake. In these situations, it is desirable to have a robot go into confined spaces and inventory the potential hazardous situations. The rover will be able to navigate autonomously or be maneuvered by the rover operator in a closed environment, up to a certain distance of a target object centered in a webcam's view. Its essential components are:

- ISCR Operator User Interface
- Range Finder
- Stereo Camera
- Battery
- Navigation Guidance & Control (NG&C)
- Rover Hardware

Since ISCR is a mission critical system, any software or hardware failure will cause the mission failure and risk of catastrophe. Dependability becomes a very important level of service requirement which means if operating in autonomous mode the rover will not crash and if operating in non-autonomous mode the rover will follow the operator instructions.

Based on the dependability attributes we defined, the potential dependability risks for Inspector SCRover are as follows:

- **Availability/Reliability Risk**: If the rover runs out of the battery but it fails to detect it or reserve enough power in order to return to its home for recharging, it will stop.
- **Correctness/Accuracy/Fidelity Risks**:

1) The delay or failure of the sensor(s) or the communication between sensors and state variable database, the navigation path could be deviated due to the outdated range finder or position & heading data.
2) In autonomous mode, if the algorithm for position & heading controller has defects, it can also deviate the rover's navigation path.
3) In another case, if the rover fails to transfer from autonomous mode into non-autonomous mode when operator overriding is necessary, it won't be able to follow the operator's command in a fidelity way.

- **Recoverability Risk**:

  It's desirable that the rover can successfully recover itself from some failure. However, it's hard to achieve.

6. REFERENCE

[1] A. Avizienis, J.-C. Laprie and B. Randell, Fundamental Concepts of Dependability, Research Report N01145, LAAS-CNRS, April 2001. http://citeseer.nj.nec.com/avizienis01fundamental.html

[2] A. Jones, “The challenge of building survivable information-intensive systems”, IEEE Computer, Vol. 33, No. 8, August 2000, pp. 39-43.

[3] B. Randell, “System structure for software fault tolerance”, IEEE Transctions on Software Engineering, Vol. SE-1, No. 10, June 1975, pp. 1220-232.

[4] J.C. Laprie, “Dependable computing and fault tolerance: concepts and terminology”, Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-15), Ann Arbor, Michigan, June 1985, pp. 2-11.

Using Risk to Balance Agility and Discipline: A Quantitative Analysis

Barry Boehm
University of Southern California
Department of Computer Science
boehm@sunset.usc.edu

Keywords
agile, discipline, architecting, risk, sweet spot

We have shown several qualitative analyses [2, 3] indicating that one can balance the risks of having too little project discipline with the risks of having too much project discipline, to find a "sweet spot" operating point which minimizes the overall risk exposure for a given project. We have shown qualitatively that as a project's size and criticality increase, the sweet spot moves toward more project discipline, and vice versa.

However, these results would have stronger credibility if shown to be true for a quantitative analysis backed up by a critical mass of data. Here we show the results of such a quantitative analysis, based on the cost estimating relationships in the COCOMO II cost estimation model and its calibration to 161 diverse project data points [1]. The projects in the COCOMO II database include management information systems, electronic services, telecommunications, middleware, engineering and science, command and control, and real time process control software projects. Their sizes range from 2.6 thousand lines of code (KLOC) to 1,300 KLOC, with 13 projects below 10 KLOC and 5 projects above 1000 KLOC.

The risk-balancing analysis is based on one of the calibrated COCOMO II scale factors, "Architecture and Risk Resolution," called RESL in the COCOMO II model. Calibrating the RESL scale factor was a test of the hypothesis that proceeding into software development with inadequate architecture and risk resolution results would cause project effort to increase due to the software rework necessary to overcome the architecture deficiencies and to resolve the risks late in the development cycle – and that the rework cost increase percentage would be larger for larger projects.

The regression analysis to calibrate the RESL factor and the other 22 COCOMO II cost drivers confirmed this hypothesis with a statistically significant result. The calibration results determined that for this sample of projects, the difference between a Very Low RESL rating (corresponding to an architecting investment of 5% of the development time) and an Extra High rating (corresponding to an investment of over 40%, here established at 50%) was an extra 7.07% added to the exponent relating project effort to product size. This translates to an extra 18% effort for a small 10 KSLOC project, and an extra 91% effort for an extra-large 10,000 KSLOC project.

The full set of effects for each of the RESL rating levels and corresponding architecting investment percentages are shown in Table 1 for projects of sizes 10, 100, and 10000 KSLOC. Also shown are the corresponding total-delay-in-delivery percentages, obtained by adding the architecting investment time to the rework time, assuming a constant team size during rework to translate added effort into added schedule. Thus, in the bottom two rows of Table 1, we can see that added investments in architecture definition and risk resolution are more than repaid by savings in rework time for a 10,000 KSLOC project up to an investment of 33%, after which the total delay percentage increases.

This identifies the minimum-delay architecting investment "sweet spot" for a 10,000 KSLOC project to be around 33%. Figure 1 shows the results of Table 1 graphically. It indicates that for a 10,000 KSLOC project, the sweet spot is actually a flat region around a 37% architecting investment. For a 100 KSLOC project, the sweet spot is a flat region around 20%. For a 10 KSLOC project, the sweet spot is at around a 5% investment in architecting. The term "architecting" is taken from Rechtin's System Architecting book [5], in which it includes the overall concurrent effort involved in developing and documenting a system's operational concept, requirements, architecture, and life-cycle strategic plan. It is roughly equivalent to the agilists' term, Big Design Up Front (BDUF) [4]. Thus, the results in Table 1 and Figure 1 confirm that investments in architecting and BDUF are less valuable for small projects, but increasingly necessary as the project size increases.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Conference '00, Month 1-2, 2000, City, State.
Copyright 2000 ACM 1-58113-000-0/00/0000…$5.00.

Table 1. Effect of Architecting Investment Level on Total Project Delay

COCOMO II RESL Rating	Very Low	Low	Nominal	High	Very High	Extra High
% Architecting Investment	5	10	17	25	33	>40 (50)
Scale Factor Exponent for Rework Effort	1.0707	1.0565	1.0424	1.0283	1.0141	1.0000
10 KDSI Project: -Added Rework %	18	14	10	7	3	0
-Project Delay %	23	24	27	32	36	50
100 KDSI Project: -Added Rework %	38	30	21	14	7	0
-Project Delay %	43	40	38	39	40	50
10,000 KDSI Project: -Added Rework %	91	68	48	30	14	0
-Project Delay %	96	78	65	55	47	50

However, the values and sweet spot locations presented in Figure 1 are for nominal values of the other COCOMO II cost drivers and scale factors. Projects in different situations will find that "their mileage may vary." For example, a 10-KSLOC safety-critical (COCOMO II RELY factor rating = Very High) project will find that its sweet spot will be upwards and to the right of the nominal-case 10-KSLOC sweet spot. A 10,000-KSLOC highly-volatile (COCOMO II Requirements Volatility factor = 50%) project will find that its sweet spot will be higher and to the left of the nominal-case 10,000-KSLOC sweet spot, due to the costs of BDUF rework. Also, various other factors can affect the probability (and size) of loss associated with the RESL factor, such as staff capabilities, tool support, and technology uncertainties [1]. And these tradeoffs are only considering project delivery time and productivity and not business value, which would push the sweet spot for safety-critical projects even further to the right. Clearly, there are a number of further issues and situations deserving of additional analysis.

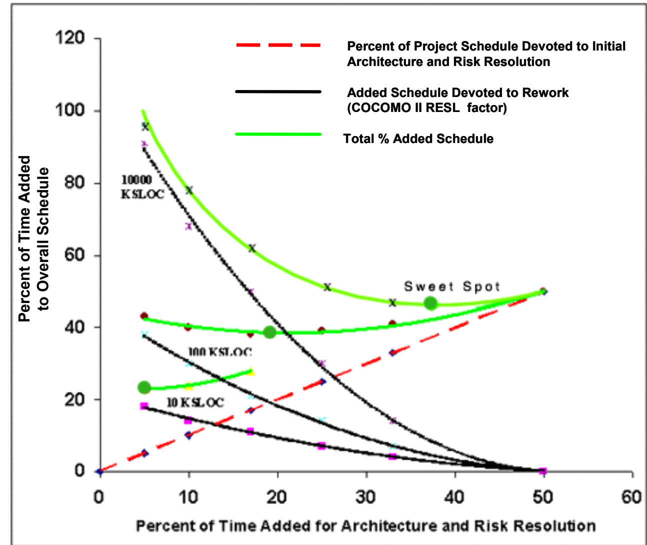


Figure 1. How Much Architecting is Enough?

REFERENCES

[1] B. Boehm, C. Abts, A.W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, Software Cost Estimation with COCOMO II, Prentice Hall, 2000.

[2] B. Boehm and W. Hansen, "The Spiral Model as a Tool for Evolutionary Acquisition," CrossTalk, May 2001, pp. 4-11.

[3] B. Boehm, "Get Ready for Agile Methods, With Care," IEEE Computer, January 2002, pp. 64-69.

[4] P. McBreen, Questioning Extreme Programming, Addison Wesley, 2003.

[5] E. Rechtin, Systems Architecting, Prentice Hall, 1991.