Managing Software Reuse Economics: An Integrated ROI-based Model

A. Mili[1)], **S. Fowler Chmiel, R. Gottumukkala, and L. Zhang**

CSEE Department, West Virginia University

Morgantown, WV 26506-6109, USA

fax: (304) 293 8602

November 6, 2000

[1)]amili@csee.wvu.edu

ABSTRACT

Several cost models have been proposed in the past for estimating, predicting, and analyzing the costs of software reuse. In this paper we propose an integrated ROI-based cost model which attempts to encompass existing models. Our approach is to analyze existing models, identify their dimensions of variance, classify the models along these dimensions, then provide an integrated cost model that makes explicit provisions for these dimensions of variance. In this paper, we also discuss in what sense our model encompasses existing models, present a prototype that supports the cost model, and illustrate the model with a sample example.

Keywords

Software Reuse, Software Cost Estimation, Return on Investment, Domain Engineering, Application Engineering, Component Engineering, COCOMO.

Managing Software Reuse Economics: An Integrated ROI-based Model

A. Mili, S. Fowler Chmiel, R. Gottumukkala, and L. Zhang

CSEE Department, West Virginia University

Morgantown, WV 26506-6109, USA

fax: (304) 293 8602

ABSTRACT

Several cost models have been proposed in the past for estimating, predicting, and analyzing the costs of software reuse. In this paper we propose an integrated ROI-based cost model which attempts to encompass existing models. Our approach is to analyze existing models, identify their dimensions of variance, classify the models along these dimensions, then provide an integrated cost model that makes explicit provisions for these dimensions of variance. In this paper, we also discuss in what sense our model encompasses existing models, present a prototype that supports the cost model, and illustrate the model with a sample example.

Keywords

Software Reuse, Software Cost Estimation, Return on Investment, Domain Engineering, Application Engineering, Component Engineering, COCOMO.

1 A VARIETY OF COST MODELS

The ability to estimate, predict, and monitor lifecycle costs is an integral part of a successful software process, including a software reuse process. We submit the thesis that in software reuse (as in software engineering in general), most decisions can ultimately be rationalized by means of economic considerations. We also submit that software reuse is dependent on the cooperation of many parties, and that software reuse will happen only if all the parties involved in this cooperative process can achieve the goals of the reuse program while they are seeking to achieve their own goals. In this paper we propose an integrated cost model for software reuse, which recognizes the relevant parties in a software reuse process, and quantifies their goals in economic terms. We further find that the economic decisions that arise in the practice of software reuse can all be modeled as investment decisions: at the corporate level, corporate management commits resources to initiate a reuse program, and expects to reap benefits in terms of better product quality, higher

productivity, and shorter time to market; at the (reuse) department level, the reuse department commits resources to initiate a domain engineering initiative and expects to reap benefits by *selling* (be it internally) reusable assets to project teams; project teams commit resources and take risks to adopt a reuse discipline, and expect to reap benefits in terms of productivity, quality, and timeliness of project completion; component developers commit resources to develop a reusable asset, and expect to reap benefits by *selling* the asset to project teams, be it internally. In this paper, we attempt to analyze software reuse investment cycles and highlight their interrelationships.

There has been a great deal of research on the economics of software reuse, and a large number of economic models have been proposed: [Balda and Gustafson 1990; Barnes and Bollinger 1991; Boehm et al 1995; Bollinger and Pfleeger 1990; Bowes et al 1992a; Bowes et al 1992b; COCOTS 1999; Coulange 1998; Favaro 1996; Favaro et al 1998; Frakes and Terry 1994; Frazier 1993; Gaffney and Durek 1989; Gaffney and Cruickshank 1992; Guerrieri et al 1989; Henderson-Sellers 1993; Jones 1994; Kain 1994; Kang and Levy 1989; Leach 1997; Lim 1992; Lim 1994; Lim 1996; Malan 1993; Malan and Wentzel 1993; Margano and Rhoades 1992; Mayobre 1991; Mili 1996; Melo et al 1995; NATO 1991; Pant et al 1996; Poulin and Caruso 1993; Poulin 1997b; Poulin 1997a; Raymond and Hollis 1991; Reifer 1997; Schach 1994; Schach and Yang 1995; Schimsky 1992; Stevens 1993]. Interestingly, even though these models appear to be dealing with the same problem (software reuse cost estimation), they in fact differ significantly from each other. Some of the features that distinguish between these different cost models include:

- *Investment Cycle.* We recognize that most decisions that arise in the practice of software reuse can be modeled as return on investment decisions. Also, we have identified four distinct investment cycles, which we have briefly introduced above: the *corporate* investment cycle, the *domain engineering* investment cycle, the *application engineering* investment cycle, and the *component engineering* investment cycle. Each of these cycles is subject to a specific economic rationale, and can be quantified by means of a variety of economic functions.

- *Economic Function.* Favaro [Favaro 1996] identifies five different functions that an investment assessment model may want to consider: *Net Present Value*, *Payback Value*, *Average Return on Book Value*, *Internal Rate of Return* and *Profitability Index.* Traditional textbook references on engineering economics [Berney and Garstka 1984; Horngren 1981; Van Horne 1983; Viscione 1984] discuss other relevant functions that pertain to investment cycles, which we will present subsequently (in section 2.3).

- *Cost Factors.* For a given investment cycle and a given economic function, the set of cost factors that are taken into account in a cost model is the most important feature of the model: this feature specifies what aspects of the reuse decision we want to consider.

- *Reuse Organization.* Several organizational models are possible in software reuse [Calidiera and Basili 1991; Coulange 1998; Fafchamps 1994]; the organizational structure has some impact on how costs are determined, charged, and accounted for. Many cost models assume specific organizational structures, but virtually none make this assumption explicit.

- *Scope.* Some models consider a punctual decision whereas others consider a long term investment cycle; some models limit the investment cycle on the basis of technical considerations (life expectancy of a reusable asset) whereas others limit the cycle on the basis of strategic considerations (e.g., investment must pay off within three years, investment cycle driven by market considerations).

- *Hypotheses.* Some cost models neglect integration costs, and assume that the cost of building an aggregate of two components is the sum of building each; some cost models assume that software development costs are linear in the size of the product; some cost models fail to take into account the discount rate of resources; some cost models ignore quality gains (because they are unable to quantify them into their equation) and focus on productivity gains; some (virtually all) cost models ignore the inflation of code size that stems from software reuse, when in fact reusable code tends to be larger than code developed for a specific single use (due to requirements of generality); finally, virtually all cost models ignore time-to-market benefits that stem from software reuse.

- *Viewpoint.* Many parties are involved in a software reuse initiative: these include the corporate executives, the producer staff, the consumer staff, the library managers, and component providers. Each of these parties has a specific outlook, a specific objective function to optimize, specific constraints, specific responsibilities —all of which influence the party's interpretation of economic equations.

This variety of attributes accounts for the abundance of economic models, and shows, paradoxically, that there too few models —not too many models. The solution is not, of course, to invent more models to fill the gaps identified above —but rather to investigate generic models that can be specialized along the dimensions discussed above.

2 A GENERIC SOFTWARE REUSE COST MODEL

In [Mili et al 2000], we have outlined a generic cost model, which we discuss in some detail in this paper.

2.1 Variety of Investment Cycles

The main feature of this cost model, as shown in figure 1, is the recognition that there are four distinct investment cycles, and the identification of how these investment cycles feed cost information into each other. For example, a corporate manager would assess the impact of the reuse initiative by taking into consideration the cost of the reuse infrastructure as well as domain engineering costs (cumulated across domains engineering initiatives), and would balance these costs against the benefits reaped from quality and productivity gains achieved in application engineering (cumulated across application development projects). Also, a manager of the producer organization would assess the impact of a domain engineering effort by considering the cost of performing domain analysis and producing reusable assets against the benefits reaped from *selling* these assets to project managers. Appropriate incentive and reward mechanisms must be put in place and adequately fine-tuned to ensure that all four ROI's are positive; this defines the appropriate conditions that are needed to make reuse happen.

2.2 Variety of Cost Factors

At an abstract level, investments can be quantified by six cost factors, which we briefly review in this section. In the sequel, we discuss how these factors can be used to define the various economic functions, then we discuss how these cost factors can be quantified for each investment decision. The six cost factors are:

- *Investment Cycle*, denoted by Y, measured in number of years, typically ranging between 3 and 5, counted from a start date, which we denote by SD.
- *Discount Rate*, denoted by d, is an abstract quantity, that typically ranges between 0.10 and 0.20; it reflects the time value of money (if we consent to spend a dollar today, we expect to get back at least $(1 + d)$ dollars within a year to make the sacrifice worthwhile).
- *Start Date*, denoted by SD, refers to the date at which the investment starts and the initial costs are incurred.
- *Investment Costs*, denoted by IC, and measured in person months. We use the person months (PM) as the unit of measurement because most costs that arise can best be quantified as personnel effort; when, occasionally, we encounter costs that are more naturally quantified in monetary terms, we use a default ratio to convert them to person months.
- *Episodic Benefits*, at year y, for $SD + 1 \leq y \leq SD + Y$, denoted by $B(y)$, and measured in person months.

Corporate Cycle

Investment Costs

Reuse Infrastructure

Episodic Costs

Domain Engineering Cycle

Investment Costs

Domain Analysis Costs

Episodic Costs

Component Engineering Cycle

Investment Costs	*Episodic Costs*	*Episodic Benefits*
Developt for Reuse	Library overhead	Sale of asset

Episodic Benefits

Cost Transfers from Projects

Episodic Benefits

Application Engineering Cycle

Investment Costs	*Episodic Costs*	*Episodic Benefits*
Training +	Reuse Overhead	Productivity gains
Process Impact	+ Reuse Risks	+ Quality gains

Figure 1: Software Reuse Cost Models

- *Episodic Costs*, at year y, for $SD+1 \leq y \leq SD+Y$, denoted by $C(y)$, and measured in person months.

For the sake of uniformity, we take the following notational conventions, which we may occasionally use throughout the paper:

- We extend function C to year SD, and we pose $C(SD) = IC$.
- We extend function B to year SD, and we pose the default value $B(SD) = 0$. There are situations where we want to assign $B(SD)$ a different value than zero; but zero is the implicit value when no other value is specified.

We further extend this convention by considering that for years prior to the start of the cycle (SD), both $B(y)$ and $C(y)$ are zero.

2.3 Variety of Economic Functions

Looking at the literature on engineering economics, we have been able to identify a set of economic functions that can be used to assess the worthiness of an investment decision; we review these briefly below.

- *Net Present Value*, denoted by NPV, measured in person months, and defined by:

$$NPV = -IC + \sum_{z=1}^{Y} \frac{B(SD+z) - C(SD+z)}{(1+d)^z}.$$

  An investment is worthwhile whenever the NPV exceeds zero; it is all the more attractive that NPV is larger. Using the conventions introduced above, we can write the NPV formula as:

$$NPV = \sum_{z=0}^{Y} \frac{B(SD+z) - C(SD+z)}{(1+d)^z}.$$

  For the purposes of this study, we adopt the latter definition of NPV, because it gives us more latitude to decide how to define $B(SD)$ and $C(SD)$.

- *Return on Investment*, denoted by ROI, and defined by (the dimensionless formula):

$$ROI = \frac{NPV}{IC}.$$

  The formula of ROI recognizes that investments involve risks, and that our estimates are not always accurate, hence prorates the net present value (potential payoff) with the investment cost. For the same value of NPV, the investment is all the more worthwhile that IC is relatively smaller.

- *Profitability Index*, denoted by PI, and defined by (the dimensionless formula):

$$PI = \frac{1}{IC} \times \sum_{z=1}^{Y} \frac{B(SD+z) - C(SD+z)}{(1+d)^z}.$$

This quantity prorates the potential profit with respect to the investment cost. An investment is worthwhile whenever PI exceeds 1, and is all the more attractive that PI is greater.

- *Average Rate of Return*, denoted by ARR, and defined by the dimensionless formula:

$$ARR = \frac{PI}{Y}.$$

It prorates the profitability index by the number of years in the investment cycle.

- *Average Return on Book Value*, denoted by $ARBV$, and defined with respect to an *amortization schedule* of the investment cost over the investment cycle. We model the amortization schedule by means of a function $Am(y)$ which satisfies the equation $\sum_{z=1}^{Y} Am(SD+z) = IC$, and we define $ARBV$ by:

$$ARBV = \frac{1}{Y \times IC} \times \sum_{z=1}^{Y} \frac{B(SD+z) - C(SD+z) - Am(SD+z)}{(1+d)^z}.$$

Favaro [Favaro 1996] discourages the use of this function, because it is vulnerable to many accounting distortions and too many subjective factors. Although we agree with Favaro's assessment, we do include it for now, for the sake of completeness, and we usually take $Am(y) = \frac{IC}{Y}$ for all $y : SD+1 \leq y \leq SD+Y$.

- *Internal Rate of Return*, denoted by IRR, and defined as the value of d that makes the net present value zero, i.e. the solution in d of the following equation:

$$\sum_{z=0}^{Y} \frac{B(SD+z) - C(SD+z)}{(1+d)^z} = 0.$$

An investment is worthwhile if IRR is smaller than the corporate discount rate (determined by corporate management as a strategic decision).

- *Payback Value*, denoted by PB and defined as the shortest investment cycle that makes the net present value non negative, i.e. the smallest integer value in Y such that:

$$\sum_{z=0}^{Y} \frac{B(SD+z) - C(SD+z)}{(1+d)^z} \geq 0.$$

An investment is worthwhile if PB is smaller than the amount of time that corporate management is willing to wait for the investment cost to be amortized.

2.4 Variety of Viewpoints

A corporate software reuse initiative involves many stakeholders: corporate managers, domain engineering teams, application engineering teams, individual producers of reusable assets. Each stakeholder has

a distinct return on investment equation, which can be characterized by analyzing the cost factors that we have discussed in section 2.2. In this section, we discuss these cost factors, and investigate how they can be estimated in practice. For the sake of simplicity, we focus our discussion on factors IC, $B(y)$ and $C(y)$, since factors Y and d are uniform within a corporation, hence are the same for all stakeholders; also their derivation (a strategic corporate decision) raises no special issue. This section shows how these costs are cascaded from one investment cycle to the next; this cascade effect is reflected in the design of the automated support tool, which we discuss in section 4.

2.4.1 Notational Conventions

In the foregoing discussion, we had introduced six factors that characterize a *return on investment* decision cycle:

- Y: Investment cycle length.
- d: Discount rate.
- SD: Start date of the investment cycle.
- IC: Initial upfront cost.
- $C(y)$: Cost function C applied at year y, for $SD \leq y \leq SD + Y$. We may occasionally equate IC with $C(SD)$.
- $B(y)$: Benefit function B applied at year y, for $SD \leq y \leq SD + Y$.

In order to distinguish between the different investment cycles that we investigate in the sequel, we resolve to use the following subscripts on each of the relevant cost factors:

γ Component engineering factors.

δ Domain engineering factors.

α Application engineering factors.

ρ Corporate engineering factors.

Hence, for example, $C_\gamma(y)$ represents the episodic cost associated with component γ at year y. In keeping with these conventions, $component sub \in \delta$ means that component γ is developed as part of the domain engineering activity δ; $\gamma \in \alpha$ means that component γ has been used in the development of the application α; $\alpha \in \rho$ means that application α is developed by corporation ρ; $\delta \in \rho$ means that the domain engineering activity δ has been carried out by corporation ρ.

2.4.2 Component Engineering Viewpoint

We review in turn the three cost factors of investment costs, episodic benefits, and episodic costs, as they apply to the component developer. The investment decision we are dealing with here is whether or not to develop a reusable asset to satisfy a tentatively specified set of requirements; the decision will hinge on how much it costs to develop this asset, how much savings project teams will achieve by reusing it, and with what frequency do we expect this asset to be needed in future development projects. Relevant costs are cataloged as follows:

- *Investment Costs.* The upfront investment costs involved in component engineering include the costs of development for reuse, and the costs of reuse certification and library insertion. Specifically, the investment cost of the component engineering investment cycle, which we represent as the episodic cost at year SD, can be quantified by the following equation:

$$C_\gamma(SD) = ER + LI,$$

  where ER is the cost of development for reuse, and LI is the cost of certification and library insertion. In the absence of more accurate information, ER can be estimated by estimating the cost of development from scratch, E (using traditional cost estimation models [Boehm 1981; Boehm et al 1995]) and prorating it with the $RCWR$ factor (*Relative Cost of Writing for Reuse*, for which Poulin [Poulin 1997a] provides default values derived from industrial experience), i.e. $RE = E \times RCWR$.

- *Episodic Costs.* The episodic costs at year y, for $SD + 1 \leq y \leq SD + Y$, can be quantified by the following equation:

$$C_\gamma(y) = OC(y) + MN_\gamma(y),$$

  where $OC(y)$ is the component's share of the yearly cost of operating the library and $MN_\gamma(y)$ is the maintenance cost of component γ at year y. The term $OC(y)$ can trivially be quantified by dividing the labor costs of operating the library by the size of the library, and is realistically independent of γ —although it could conceivably depend on y (e.g. by virtue of economy of scale phenomena, as the size of the library increases). The term $MN_\gamma(y)$ can be quantified by means of COCOMO-like maintenance effort equations [Boehm 1981; Boehm et al 1995]; it is, of course, dependent on the component, and may well depend on the year (e.g. decrease as the component improves, or increase as the component grows larger).

- *Episodic Benefits.* These are the much-touted gains in productivity and quality. Gains in productivity can be quantified by subtracting reuse costs from custom development costs. Custom development

costs can be derived using traditional cost estimation models (e.g. COCOMO [Boehm 1981; Boehm et al 1995]). Reuse costs can be derived using the multitude of empirical results that are available nowadays [Poulin 1997a]. Gains in quality can be quantified in person months by equating them with the gains in maintenance costs integrated over the relevant lifetime of the reusable products [Mili et al 2001; Mili 1996]. Episodic benefits at year y, for $SD + 1 \leq y \leq SD + Y$, can be quantified by the following equation:

$$B_\gamma = freq(y) \times BP_\gamma(y) + freq'(y) \times WP_\gamma(y),$$

where

- $freq(y)$ is the frequency of black box use of the component at hand at year y; depending on whether y is in the past or in the future, $freq(y)$ can be collected from reuse data or estimated on the basis of expert judgment.
- BP_γ is the black box sale price of component γ. This quantity must be low enough to encourage application developers to acquire it, yet high enough to ensure that the component's return on investment is positive. It is subject to the following equation:

  $$RET + INST_\gamma + BP_\gamma < E_\gamma + QG_\gamma,$$

  where RET_γ is the (average) retrieval and assessment cost of a component, $INST_\gamma$ is the (estimated) instantiation cost of component γ, and E_γ is the cost of custom development of component γ, and QG_γ (stands for: quality gains) refers to the value-added to component γ by the fact that this asset is of higher quality than an equivalent custom-developed asset; when we discuss application engineering costs, we will briefly review how this factor can be estimated in practice. Poulin [Poulin 1997a] introduces a factor, called RCR (*Relative Cost of (Black Box) Reuse*), which allows us to formulate $INST_\gamma$ as a linear function of E_γ. Similarly, we introduce a factor called RBP (*Relative Black Box Price*), which allows us to formulate BP_γ as a linear function of E_γ. The equation becomes:

  $$RET + RCR \times E_\gamma + RBP \times E_\gamma < E_\gamma + QG_\gamma.$$

  If we neglect retrieval costs (all the more legitimate that E_γ is large), the equation becomes:

  $$RCR \times E_\gamma + RBP \times E_\gamma < E_\gamma + QG_\gamma.$$

  Assuming $QG_\gamma > 0$ (which is perfectly legitimate), we can simplify this (in)equation to:

  $$RCR \times E_\gamma + RBP \times E_\gamma \leq E_\gamma.$$

Simplifying on both sides by the term E_γ, we find:

$$RCR + RBP \leq 1.$$

Poulin [Poulin 1997a] provides the default value of 0.2 for RCR. We take for RBP the default value 0.6; decreasing RBP puts a strain on the domain engineering balance sheet, whereas increasing RBP enhances the balance sheet of domain engineering, at the expense of application engineering (who then have less incentive to reuse).

- $freq'(y)$ is the frequency of white box use of the component at hand at year y; depending on whether y is in the past or in the future, $freq'(y)$ can be collected from reuse data or estimated on the basis of expert judgment.
- WP_γ is the white box sale price of component γ. This quantity is subject to the following equation:

$$RET + ADP_\gamma + WP_\gamma < E_\gamma + QG'_\gamma,$$

where RET is the (average) retrieval and assessment cost of a component, ADP_γ is the (estimated) adaptation cost (in the context of white box reuse) of component γ, E_γ is the cost of custom development of component γ, and QG'_γ is the value-added quality gains obtained from (white box) reusing asset γ. If we let RCR' be the relative cost of white box reuse, and let RWP be the relative white box price, then we find, by analogy with our earlier discussion,

$$RCR' + RWP \leq 1.$$

Empirical studies [Boehm et al 1995] advocate a default value of 0.67 for RCR'; we let the default value of RWP be 0.20.

2.4.3 Application Engineering Viewpoint

We review in turn the three cost factors of investment costs, episodic benefits, and episodic costs, as they apply to the manager of an application engineering development project. The investment decision we are dealing with here is whether or not to adopt reuse in a given development project; the decision will hinge on how much of a match there is between the project's needs and the available reusable assets, and how much of an overhead reuse adoption involves for the project. The investment cycle of application engineering has some special features, which we review:

- First, we assume that an application development project (especially one that is based on reuse) takes place well within a year, hence all the development costs can be modeled as upfront investment costs, or, equivalently, as episodic costs at year SD.

- Second, the costs of application development with reuse have to be balanced against the cost of custom development of an equivalent application; we model this situation by means of a non-zero value for the benefit function at year SD.

With these two qualifications in mind, we review the costs involved and investigate how to quantify them.

- *Investment Costs.* The costs of reuse adoption include the cost of training, tool acquisition, and the operational impact of reuse processes. The operational risks that stem from reuse adoption include the following contingencies:

  - Components can be retrieved without being relevant, if the library's retrieval procedure does not have good precision.
  - Components can be retrieved and be deemed relevant, but prove subsequently to cost too much to adapt and integrate, if the library insertion procedures do not enforce good quality standards.

  All of these costs can be quantified in person months, specifically the person months wasted due to the distractions caused by poor retrieval precision or poor quality reusable assets.

  At year SD, the application engineering costs for application α are the costs of acquiring reusable assets, which we write as

  $$C_\alpha(SD) = \sum_{\gamma \in \alpha} PR_\gamma,$$

  where PR_γ is the price of component γ. Depending on whether component γ was used *black box* or *white box* in developing application α, we find $PR_\gamma = BP_\gamma$, or $PR_\gamma = WP_\gamma$.

- *Episodic Costs.* For years subsequent to the cycle's start date (SD), episodic costs are zero, i.e.

  $$C_\alpha(y) = 0, SD + 1 \leq y \leq SD + Y.$$

- *Episodic Benefits.* At year SD, the benefit derived from reuse is the cost savings achieved by using reusable assets rather than writing custom code. We quantify this benefit as:

  $$B_\alpha(SD) = E_\alpha - (RA + DEV + IN),$$

  where E_α is the estimated cost of custom development of application α (derived using traditional cost estimation models), RA is the reuse adoption costs (incurred by the project for choosing to reuse), DEV is the cost of developing the code that was custom developed, and IN is the cost of integrating the various components. We assume that we are given a function, say μ, that estimates custom development costs for software products (by traditional methods [Boehm 1981; Boehm et al 1995]), and we use it to estimate terms E_α, DEV, and IN.

- E_α and DEV can be estimated by applying function μ to estimates of the sizes of the whole application and (respectively) the new code.
- IN can be estimated by considering that the integration cost of two components of size A and B is given by:

$$IN(A,B) = \mu(A+B) - (\mu(A) + \mu(B)).$$

We assume that reuse adoption costs (RA) can be estimated by expert judgment.

The episodic benefits of subsequent years, $y > SD$, can be quantified by the savings in maintenance effort that stem from (re)using high quality reusable assets, rather than custom developed assets. We write,

$$B_\alpha(y) = \sum_{\gamma \in \alpha} OCD_\gamma(y), SD+1 \leq y \leq SD+Y,$$

where $OCD_\gamma(y)$, which stands for (*Operating Cost Differential*), is the difference in maintenance/ operating cost at year y between an instance of the component developed for single use and an instance of the same component developed for reuse. Presumably, a component that was developed for reuse is of better quality at delivery, and has undergone more intensive field scrutiny, hence contains fewer bugs than an equivalent component developed from scratch for single use. It is also conceivable that (corrective) maintenance costs decrease from year to year, as the component undergoes more and more testing (hence the dependence on y). The COCOMO model [Boehm 1981] models maintenance costs by means of *Annual Change Traffic*, ACT, which is the ratio of yearly maintenance costs over development costs. We argue that a reusable asset has a lower annual change traffic, since it is of better quality; further, we argue that the value of the annual change traffic depends on whether the asset is black box reused or white box reused. This yields three equations for yearly maintenance effort, which we present in turn below, for (respectively) custom developed components, black box reused components, and white box reuse components.

$$Maint_\gamma = ACT \times E_\gamma.$$
$$Maint'_\gamma = ACT' \times E_\gamma.$$
$$Maint''_\gamma = ACT'' \times E_\gamma.$$

We typically have the inequality:

$$ACT > ACT'' > ACT'.$$

The formula for $OCD_\gamma(y)$ depends on whether component γ is reused under black box reuse or under white box reused. For black box reuse, we have

$$OCD_\gamma(y) = (ACT - ACT') \times E_\gamma.$$

For white box reuse, we have

$$OCD_\gamma(y) = (ACT - ACT'') \times E_\gamma.$$

Boehm [Boehm 1981] uses a default value of 0.15 for ACT; we let the default value of ACT' be 0.08 and the default value of ACT'' be 0.12.

Incidentally, the factor QG_γ introduced above (for quality gains) can now be formulated in terms of $OCD_\gamma(y)$, as follows:

$$QG_\gamma = \sum_{z=1}^{Y} \frac{OCD_\gamma(z+SD)}{(1+d)^z}.$$

2.4.4 Domain Engineering Viewpoint

We review in turn the three cost factors of investment costs, episodic benefits, and episodic costs, as they apply to the manager of a domain engineering initiative. The investment decision we are dealing with here is whether or not to initiate a domain engineering effort in a tentatively specified application domain; the decision hinges on how much development activity is expected in the future within the specified domain, and how much effort is involved in performing domain analysis and design for reuse. Relevant costs are cataloged as follows:

- *Investment Costs.* The upfront investment costs of domain engineering are the (non-trivial) costs of domain analysis, asset development (for reuse) and asset cataloging. We can quantify them by adding the domain analysis costs to the investment costs of all the components that are developed at year SD as part of the domain engineering effort. Quantitatively, we write:

  $$C_\delta(SD) = DA + \sum_{\forall \gamma \in \delta} C_\gamma(SD).$$

  The first term of this equation represents the domain analysis costs; the second term represents the development and cataloging costs of all the domain assets that are developed at year SD. Even though the sum quantifier covers all the domain assets, only assets that are created at year SD are accounted for, since the other assets (that are created in subsequent years) have zero cost for year SD.

- *Episodic Costs.* Episodic costs of the domain engineering cycle at year y, for $SD+1 \leq y \leq SD+Y$, are the sum of the episodic costs of all the components that are developed at year y as part of the domain. Specifically,

  $$C_\delta(y) = \sum_{\forall \gamma \in \delta} C_\gamma(y), y \geq SD+1.$$

The convention that $C_\gamma(SD) = IC$ provides, with respect to the formula of $C_\delta(y)$ that if several components of a domain are developed over several years, they are duly accounted for in the appropriate years. This, in turn, means that we do not have to assume that all the components of a domain have to be developed at once, at the beginning of the domain engineering cycle.

- *Episodic Benefits.* Episodic benefits of the domain engineering cycle at year y, for $SD{+}1 \leq y \leq SD{+}Y$, are the sum of the episodic benefits of all the components that are developed as part of the domain. Specifically,

$$B_\delta(y) = \sum_{\forall \gamma \in \delta} B_\gamma(y), y \geq SD + 1.$$

2.4.5 Corporate Viewpoint

We review in turn the three cost factors of investment costs, episodic benefit, and episodic costs, as they apply to the corporate manager. The investment decision we are dealing with here is whether or not to initiate a corporate software reuse program; the decision will hinge on the expected infrastructure costs, the operational impact of reuse introduction, and the expected volume of development activity. Relevant costs are cataloged as follows:

- *Investment Costs.* The upfront investment costs of the corporate investment cycle include the cost of building a reuse infrastructure and initiating a reuse program, which include: purchasing and installing a repository to hold reusable assets; required hiring, personnel training, and operational modifications within the corporation; eventually, the cost of initially populating the reuse library. All these costs can be computed / estimated at the beginnings of the corporate investment cycle, yielding the value of $C_\rho(SD)$.

$$C_\rho(SD) = INF + OPR,$$

where INF reflects infrastructure costs, and OPR reflects operational costs.

- *Episodic Costs.* The corporate episodic cost at year y, $SD + 1 \leq y \leq SD + Y$, is the cumulative episodic cost of all the domain engineering activities that are active in a given year. It also includes the cumulative upfront cost for all development projects and domain engineering initiatives that started on the given year. This can merely be written as:

$$C_\rho(y) = \sum_{\delta \in \rho} C_\delta(y),$$

$$SD + 1 \leq y \leq SD + Y.$$

- *Episodic Benefits.* The corporate episodic benefit is the cumulative application engineering benefit of all the projects that are in progress on a given year.

$$B_\rho(y) = \sum_{\alpha \in \rho} B_\alpha(y),$$

$$SD + 1 \leq y \leq SD + Y.$$

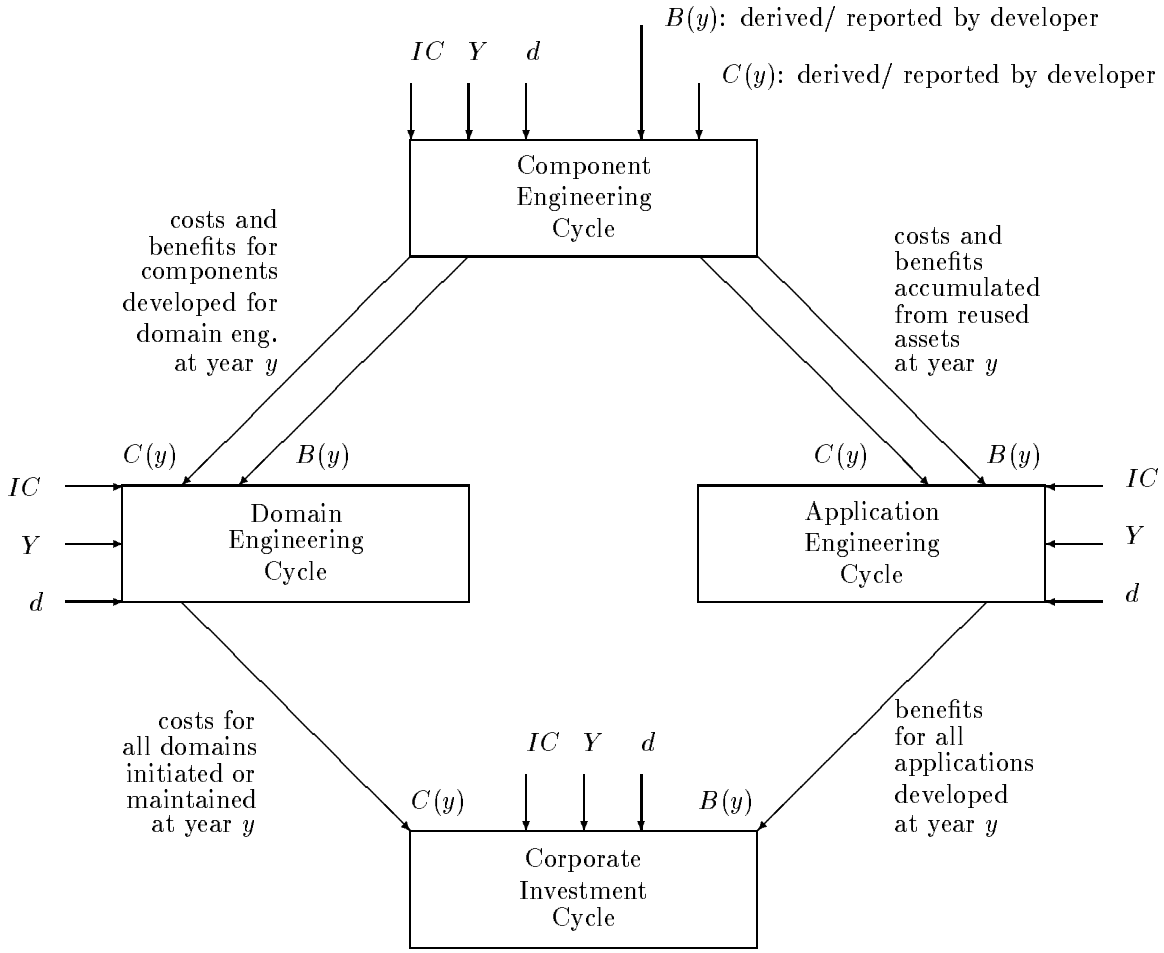
The cascade pattern by which costs are propagated from one decision cycle to the next is highlighted in Figure 2. This figure illustrates in what way the various viewpoints are related to each other, and how the cost parameters can be fine-tuned to ensure that all viewpoints are satisfied. Figure 3 complements Figure 2 by showing the cost table of each investment cycle, for the start date (SD) and for subsequent dates ($y > SD$).

Note that the corporate level benefits are the cumulative benefits reaped from application engineering, while the corporate level costs are the cumulative costs incurred from domain engineering. One may want to ask: what happened to domain engineering benefits and application engineering costs? The answer lies in the way in which costs and benefits are tallied in Figure 3: the benefits of domain engineering, which are *Asset Sales to Projects*, cancel out the costs of application engineering, which are *Purchase of reusable assets.* At the corporate level, this transfer of assets (and reverse transfer of credit) is an internal operation, which does not represent a gain, nor a loss. If the domain engineering activity were selling assets outside the corporation, and/or the application engineering activity were purchasing assets from outside the corporation, then this delicate balance would be broken, and we would have to redefine corporate costs and corporate benefits as the cumulative costs and benefits of both domain engineering and application engineering.

2.5 Variety of Reuse Organizations

As a synthesis of work by Basili et al [Calidiera and Basili 1991], Coulange [Coulange 1998] and Fafchamps [Fafchamps 1994], we have identified five distinct reuse organizations [Mili et al 2001]: *Lone Producer*, *Nested Producer*, *Pool Producer*, *Team Producer*, and *Experience Factory.* These organizations are characterized by a number of features, including: their division of labor, their reporting structure, their reward structure, their incentive structure, and their team cohesion. While we cannot say that our cost model varies significantly according to the specific organization that is adopted, it does rely on the following provisions:

- A clear separation between the producer team (that develops assets for reuse) and the consumer team (that produces applications from reusable assets).



SD	Start Date	*IC*:	upfront investment costs
Y:	Investment cycle (in years)	*B*(*y*):	benefits at year *y*
d:	Discount rate	*C*(*y*):	costs at year *y*

Figure 2: Cascade of Costs Through Investment Cycles

year, y	Cost, $C_\gamma(y)$	Benefit, $B_\gamma(y)$
$y = SD$	Development for reuse	
$y > SD$	residence costs + maintenance	Sales to projects

Component Balance Sheet

year, y	Cost, $C_\delta(y)$	Benefit, $B_\delta(y)$
$y = SD$	Domain Analysis + asset devel.	
$y > SD$	asset dev. + conc. domain analysis	Asset Sales to projects

Domain Balance Sheet

year, y	Cost, $C_\alpha(y)$	Benefit, $B_\alpha(y)$
$y = SD$	Purchase of reus. assets	savings on devel. costs
$y > SD$		Quality gains

Application Balance Sheet

year, y	Cost, $C_\rho(y)$	Benefit, $B_\rho(y)$
$y = SD$	Infrastructure costs	
$y > SD$	Domain costs	Application benefits

Corporate Balance Sheet

Figure 3: Balance Sheets

- A well defined pricing structure between the producer team and the consumer team, whereby the producer team gets credit that is commensurate to the amount of code that consumer teams reuse.
- A well defined pricing structure for assets acquired from external sources, including maintenance clauses.
- A well defined reward structure, whereby members of the producer teams get credit that is commensurate with the volume of reusable assets that they contribute and the frequency with which their products are reused.
- A uniform, coherent policy for data collection, pertaining to all domain engineering, application engineering, and component engineering activities.
- A clearly defined library management function (which manages reusable assets), with carefully monitored library insertion procedures, and carefully tracked costs.

2.6 Variety of Hypotheses

In order to ensure that our model be as general as possible, we have resolved to make the following provisions:

- *Non Linear Cost Effects.* Our model assumes that software development costs are an exponential function of the size; our default equations are those of COCOMO 2.0 [Boehm et al 1995].
- *Integration Costs.* Our model considers that applications are generally derived from reused assets, adapted assets, and custom developed software. It caters to the different equations of these three types, and makes provisions for integration cost: if μ is a function that maps the size of a product into its development cost, then the integration of two components of size A and B is derived as $\mu(A+B) - (\mu(A) + \mu(B))$.
- *Quantifying Quality Gains.* For the sake of uniformity, we quantify quality gains achieved from reuse in terms of person months, by equating quality gains with savings in operating costs of reusable products over their relevant lifetime (i.e. the lifetime covered by the investment cycle).
- *Code Inflation.* When comparing a reuse option against a custom development option (for the sake of estimating productivity gains, for example), it is unfair to consider that the reused product and the alternative custom developed product would have the same size. Typically the latter is significantly smaller than the former; our model makes provisions for this difference, and uses default values when more accurate information is not available.

There is one aspect that we are currently trying to take into account: how to quantify gains in terms of time-to-market; this matter is currently under investigation, and will be incorporated into the model once we have resolved it satisfactorily. Lim [Lim 1994] and Malan [Malan 1993] do make provisions for shortened time to market, but they do not quantify them sufficiently accurately for our purposes.

Once we know how to quantify it, time to market will be be added to the benefits column in the *Application Balance Sheet* (see Figure 3). Shortened time to market carries two benefits for the application developer:

- *Increased Sales volume.* With shortened time to market, the project will start collecting benefits earlier. For a fixed investment cycle (Y), collecting earlier means also collecting more, since the project spends more time in *benefit* (vs. cost) mode.
- *Increased Market share.* Second, by getting to market earlier, the project has the potential of securing a bigger share of the market, by virtue of its exclusive access to users.

While it easy to quantify gains that stem from sales volume, it is much more difficult to quantify those that stem from market share. This matter is currently under investigation.

3 CLASSIFYING EXISTING MODELS

We could argue that by covering the main dimensions discussed in section 1, we have derived a general cost model for software reuse. To test and possibly substantiate our claim, we have resolved to revisit a large number of existing cost models, to characterize them with respect to the dimensions that we have proposed, and to discuss whether they can be seen as an instance of our model. We have analyzed seventeen such models, which we evaluate and classify in this section. In [Lim 1996], Lim analyzes and compares seventeen models of software reuse cost estimation. Whereas we characterize cost models with general features (such as their cycle, their viewpoint, their organization), Lim characterizes them at a more detailed level by their equation, and the cost factors that they involve. Perhaps as a consequence, while Lim concludes with recommendations of how to select an economic model, we conclude by proposing a more comprehensive model.

We use the following abbreviations:

CE	Component Engineering
AE	Application Engineering
DE	Domain Engineering
RE	Corporate Engineering
KLOC	Kilo Lines of Code
KNCSS	Kilo Non Comment Source Statements
ROI	Return On Investment
NPV	Net Present Value
PI	Profitability Index
IRR	Internal Rate of Return
RSI	Reusable Source Instructions
RCR	Relative Cost of Development with Reuse
RCWR	Relative Cost of Writing for Reuse
SIRBO	Source Instructions Reused By Others
IV&V	Independent Verification and Validation
COTS	Commercial Off The Shelf Software

In the sequel, we review in turn the selected models. For each model, we present a brief characterization, followed by a summary assessment, then a comparison to our model.

1. [Bollinger and Pfleeger 1990]: In Bollinger and Pfleeger's cost model the baseline project is defined as an application specific process model that describes overall expectations for how a project is expected to be structured and scheduled. It is also a cost estimation tool that keeps the cost issues attached to the activities from which they are derived (i.e., costs are attached to the components). The cost data comes from actual experience or analysis and project planning for each domain. The integrated cost model's automated tool utilizes the same cost data concepts.

   The cost model of Bollinger and Pfleeger does not always take into account the inclusion effect, where use of one work product explicitly includes reuse of subsequent work products and broad spectrum reuse, where design, documentation, and work experience are reused. The integrated cost model does take into account the inclusion effect and broad spectrum reuse as all components are associated with a domain. Unlike Barnes and Bollinger, Bollinger and Pfleeger suggest using an amortization schedule to distribute the costs as the Integrated Cost Model does. Despite the arbitrary nature of amortization, they felt it was better than basing the cost on the component size or functionality.

2. [Barnes and Bollinger 1991]: Barnes and Bollinger define the reuse investment relation as the comparison of the reuse investment with the reuse benefits. The reuse investment is defined as any cost

Reference	Reuse Cycle	Economic Function	Cost Factors	Reuse Organization	Scope	Hypotheses	Viewpoint	Instantiation
Bollinger and Pfleeger, 1990	CE, RE	reuse benefits	CE costs with and without reuse	producer/ consumer	Across Projects/ Applications	RE with benefits,CE with and without reuse	corporate	CE with amortization, cost sharing domains(DE)
Barnes and Bollinger 1991,	DE, CE, AE	breakeven NPV,ROI Quality Gains	AE costs with and w/o reuse CE,DE	producer/ consumer	Across applications	producer costs = consumer benefits	producer/ consumer	AE quality gains, DE cost, CE costs and benefits
Gaffney and Cruickshank, 1992	DE, AE	return on investment, breakeven function	AE costs, prorated DE costs, productivity	internal procurement, team producer	Long term view across many applications	ignores integration costs	corporate	combining DE cycle w/relevant AE cycles
Margano and Rhoades, 1992	CE	Payback, PV, productivity gains	component costs, overhead, investment	pool producer	AE scope, focus on component	ignores training, process impacts	project manager, component developer	CE,with related DE,AE
D Schimsky 1992,	DE	breakeven AE costs	develop, maintain, reuse code	producer/ consumer	DE costs	cost of component in SLOC, benefits based on breakeven point	project	breakeven point

Reference	Reuse Cycle	Economic Function	Cost Factors	Reuse Organization	Scope	Hypotheses	Viewpoint	Instantiation
Poulin Caruso 1993,	CE, AE, RE	ROI, NPV PI	KLOC, $/KLOC	producer/ consumer	project level, corporate level	ignores time value of money, ROI based on IRR	corporate	ROI for Comp. Eng under black box reuse
Malan and Wentzel 1993,	DE, CE	NPV	overhead lifecycle development	domain centric	Within a Domain	DE costs and benefits	Domain Manager, Asset Developer	DE costs benefits over a discount rate
Frakes and Terry, 1994	AE RE	Reuse level, Reuse Frequency	number of refs to items	Application Centered	Application / Component	Black box only, reuse thresholds	Project / Corporate / Component	Different functions/ metrics
Kain, 1994	DE	Return on Investment	DE costs, AE costs with, w/o reuse	3 teams: producer, consumer, library.	across multiple projects	no quality gains, and no effect of time	project decisions at corporate level	separating investment cycles
Wayne C. Lim, 1994	AE	NPV	Component Costs,Productivity, Reuse with KNCSS	producer/ consumer	project lifecycle	NPV in lifecycle of reuse, no overhead costs	corporate, project-wide	focus on AE cycle,
COCOMO 2.0, Boehm et al., 1995	DE, vs. AE	lifecycle costs	RUSE vs. ESLOC	Any Producer/ Consumer Organization	Arbitrarily Large	Highly calibrated to specific development environments	corporate wide vs. project wide	using component level COCOMO estimates

Reference	Reuse Cycle	Economic Function	Cost Factors	Reuse Organization	Scope	Hypotheses	Viewpoint	Instantiation
Favaro, 1996,	CE	NPV, PI, ARBV, IRR payback	DE, CE	Lone Producer	across projects	ignores specifics of AE, DE	corporate	focus on component DE
Mili, 1996	CE	return on investment	component level factors (eg LOC)	organizations that merge DE with AE	domain wide	equates component ROI with reusability	producer / corporate viewpoint	matches our component level ROI
Devanbu et al., 1996	AE RE	Reuse Benefit	Size, Structure of application	Internal reuse	Project	Rigorous axiomatisation	Project	Abstract metrics vs. cost model
Poulin 1997,	DE, RE, AE	Proj. ROI, DCA, SCA, RCA, CSW, Threshold, PL ROI	LOC, RSI, RCR, SIRBO, error rate, error cost, RCWR	merging producer and consumer teams	Limited term product line engineering	no cognizance for time value of money	product line perspective (one DE, many AE)	separate investment cycles, add new cycles
Favaro J, Favaro K, Favaro P 1998,	RE	Present value, NPV	cash flows, discount rates, risks	producer/ consumer	corporate	NPV take in account risk	corporate	NPV in corporate cycle
COCOTS 1999,	AE	application engineering costs	assessment, tailoring, glue code volatility IV& V	external procurement of COTS components	across domains	COTS acquired as executable code	project/ corporate	AE cycle, additional hypotheses

that does not directly support the completion of an activity's primary development goals but is instead intended to make more work products of that activity easier to reuse (e.g., classifying components). Reuse benefits are the difference between the activity cost with and without reuse. The model of Barnes and Bollinger is primarily project centered.

3. [Gaffney and Cruickshank 1992]: The model proposed by Gaffney and Cruickshank combines domain engineering costs and application engineering costs in a single equation; it does not take into account integration costs, and assumes that the number of applications that make up the domain engineering effort is predetermined. Unlike our model, it does not provide for pricing structure between domain engineering team and application engineering team, and assumes that both activities take place in a single organization. Hence this model makes no provision for COTS-based development and external component acquisition.

4. [Margano and Rhoades 1992]: The model used with this project was the AAS Model which determines the savings by using reuse based on the productivity rate (SLOC/Labor Month) and monthly labor rate ($/Labor Month) of the producer and consumer with the additional costs of management overhead, problem analysis, error correction, and reintegration of existing code added into the total cost. The costs are based on actual data and cost estimates. This model considers the costs at the component/project level rather than the corporate level and does not account for adaptation/modification of reusable components. It stresses how reuse saves in the design phase (where the design phase is 60% of the cost of reuse project) because components can be used or adapted rather than custom developed. This model does not have parameters for system reliability, understandability of the reuse component and maintainability of the component. This model, like ours, uses actual and estimated data in the component engineering cycle. Our model provides for a wider range of investment cycles and a wider range of viewpoints.

5. [Schimsky 1992]: According to Schimsky, the software cost is a function of cost drivers and software size in SLOC. This cost model assumes that the reuse library will have to be completely restocked every 10 years to account for changing technology and that the redesign of components is included in the continuing cost of running the library. The latter is basically the definition of the component engineering white box and black box ROI in the integrated cost model. The cost factors are dependent on the cost to develop the code, maintain and provide code to the users and use reusable code (the investment and periodic costs of the component engineering cycle). The economic benefit to users is the cost avoided by not developing the code from scratch. The benefits of reuse are determined from graphs of the reuse cost ratio (cost to develop with reuse/cost to develop without reuse) versus the

breakeven point. In Schimsky's model, maintenance costs and savings due to higher reliability are ignored. If the breakeven point is greater than 5 years, then reuse is not beneficial because more than 65% of the costs could not be avoided by using reuse.

6. [Poulin and Caruso 1993]: Poulin's model of software reuse is fairly similar, in some respects, to the integrated cost model but focuses on the application engineering cycle of the integrated cost model and does not take into account the domain engineering costs of corporate reuse. The corporate level ROI is based on the IRR and consists of the corporate reuse startup costs, which is the sum of the savings over all the revenue years considered minus the costs divided by $(1 + d)$ (d: discount rate) to the revenue year considered. The NPV is the ROI minus the initial cost and assumes that reuse grows over time and is usually based on historical averages for the relative cost of reuse, the cost of writing reusable code and the amount of vendor purchased reusable code. Poulin and Caruso's model is primarily focused on the corporate viewpoint.

7. [Malan and Wentzel 1993]: Malan and Wetzel's model incorporates the reuse-related cost factors (overhead, development with and without reuse,) of the development and maintenance phases into a "long-term, multi-product net benefit model." Thus, this model subtracts "Reuse-specific overhead and setup costs, such as the cost of installing and managing the library, and conducting domain analysis" from the net savings. Malan and Wetzel includes the time value of money, maintenance costs and savings, and uncertainty (whether the asset will be reused). They validate their model with a hypothetical scenario. Furthermore, though time-to-market gains are discussed, they are not quantified.

8. [Frakes and Terry 1994]: Frakes and Terry introduce reuse level metrics and frequency metrics, and distinguish between internal reuse and external reuse. They do not propose cost models per se, but argue that their metrics reflect the level of benefit achieved from reuse. They also introduce the concept of *threshold levels* (for internal and external reuse), which allow them to quantify the question of when we are dealing with reuse. The model of Frakes and Terry is concerned exclusively with black box reuse, and has a dual application engineering / corporate engineering viewpoint.

9. [Kain 1994]: Kain proposes a return on investment model that is especially geared towards object oriented programming (where reusable assets are objects at various levels of abstraction). The ROI model is fairly simplistic: it assumes that project-level decisions are taken at the corporate level, ignores the time variance of resources (labor months, money, or other assets), fails to distinguish between upfront costs and episodic costs, and fails to quantify quality gains achieved from reuse. The model is applicable within a specific reuse organization (originators, users, and library custodians),

which provides for internal procurement but makes no provisions for acquiring external assets such as COTS products and the like.

10. [Lim 1994]: Lim's cost model was based on net present value (NPV), unlike the models of Gaffney and Durek and Barnes, Bollinger and Pfleeger. Net present value "takes the estimated value of reuse benefits and subtracts it from its associated costs, taking into account the time value of money" by "recognizing the increased profit from shortened time-to-market and accounting for risk" and applied over the entire life cycle of the product, including maintenance. Since the shortened time-to-market is difficult to assess, the overall economic benefit from NPV for these HP reuse projects is conservative.

11. [Boehm et al 1995]: COCOMO 2.0 represents an evolution from the original COCOMO ('81) cost model. It extends the original model in a number of ways, including by encompassing modern programming paradigms such as reuse. Like the original model, COCOMO 2.0 focuses on component level lifecycle costs, expressed in labor months. COCOMO 2.0 takes reuse into account in two ways: it incorporates domain engineering costs by means of the $RUSE$ factor, which reflects the envisaged scope of reuse of the asset at hand; also, it incorporates application engineering costs by means of the $ESLOC$ factor, which prorates the size of reused software as a fraction of newly developed software. Our ROI-based model uses COCOMO 2.0 to estimate many component level and application level cost factors.

12. [Favaro 1996]: Favaro discusses a range of investment analysis functions and argues that NPV is the best function for the purposes of software reuse, by virtue of its additive nature, its immunity to arbitrary factors, and its provision for the time value of money. Although he does not get into specifics, Favaro focuses his investigation to component engineering, specifically to the economics of COTS production and marketing. But his analysis is not carried out from the viewpoint of an asset developer, but rather from the viewpoint of a corporate (marketing) manager.

13. [Mili 1996]: Mili defines the reusability of a component as the return on investment associated with developing that component for reuse, or possibly adapting it from a development project for the purpose of reuse. This model focuses on component level costs, and balances them against potential domain wide benefits. Our component level ROI cycle is similar to this model.

14. [Devanbu et al 1996]: Devanbu et al. propose an axiomatic definition of a *reuse benefit* function, which they use to analyze a number of reuse metrics and cost models. Then they propose a tentative reuse benefit function which is designed to reflect not only *how much* code is being reused, but also *in what manner* it is being reused. They test their function against their set of axioms, validate it against empirical data, and compare it to other reuse metrics and cost models.

15. [Poulin 1997b]: Poulin specializes existing metrics and cost models to the context of product line-based software development. He deliberately chooses a simple model, for the sake of practitioners, in which no provisions are made for the time value of money, and the product line ROI is derived on the basis of a given number of applications developed under the product line effort. He makes provisions for the fact that application development projects may produce reusable assets, and is interested in such functions as: development cost avoidance, service (maintenance) cost avoidance, payoff threshold (the minimal number of reuse instances required to break even), and the costs associated with domain engineering. Poulin uses these cost factors to derive a return on investment formula for product line engineering. Our model separates between the domain investment cycle and the application investment cycle (which Poulin merges), and has two more cycles: the corporate cycle and the component cycle.

16. [Favaro et al 1998]: NPV represents the net totality of all contributions to the value of an investment. It adds the initial investment to the Present Value that is weighted by the compounded discount rate. The discount rate is considered as the penalty for delay of the cash flow similar to interest on a loan. Favaro uses the Capital Pricing Asset Model (CAPM) and other models to determine the discounted cash flow by taking into account the project's sensitivity to market movements, ... in the calculation of NPV. DCF's provide a method for calculating the time value of money over many of the operational benefits and costs. Two other techniques presented for valuation of investments presented are Decision Tree Analysis and Contingent Claims Analysis.

17. [COCOTS 1999]: Following in the COCOMO tradition, the USC's center of software engineering proposes a cost model that estimates manpower costs for software development using COTS products; the model is called *COCOTS*, and comes in two submodels, called *Early Design* and *Post Architecture*. These two models differ in how early in the lifecycle they are deployed, how much information they require, and (consequently) how much precision they provide. COCOTS proceeds by estimating five cost factors: the costs of candidate component assessment, component tailoring, glue code generation, system level programming, and verification-validation. Our model encompasses COCOTS models by making provision for COTS-based application engineering, although our model does not estimate these cost factors but rather relies on the user or infers them from historic/statistical data.

4 AUTOMATED SUPPORT

We are developing a software tool that supports the proposed model. In this section we present a brief description of this tool. A more complete user manual (in progress) for this tool is available on the web [Mili et al 1999a]. Also, it is possible to view a demo of this product on the web at [Mili et al 1999a].

The sample data that is provided for the purposes of the demo is also available for inspection on the same site. The proposed prototype has two main functions:

- *An Archival Function.* The purpose of this function is to keep track of costs and benefits as they arise. To this effect, the prototype runs on top of an Oracle (©Oracle Corporation) database, which contains data about corporate costs and benefits, domain costs and benefits, project costs and benefits, and component costs and benefits.

- *An Analytical Function.* The purpose of this function is to analyze investment cycles by producing any combination of the functions that we have discussed in section 2.3. This analysis can be carried out in post-mortem mode (if the investment cycle fits entirely in the past), in predictive mode (if the investment cycle starts in the present or the future), or in a combined mode (if the investment cycle starts in the past and terminates in the future). Calculations that deal with the past rely on actual data collected by the archival function; calculations that deal with the future rely on predicted/ expected/ elicited data. As time proceeds, predictive data is replaced by actual data. We are currently investigating means whereby actual data can be used to correct predictive data: for example, if domain experts estimate that some component C will be used three times a year, and actual data reports that it has been used five times a year for the last four years, we may want to revise the prediction for next year (in addition to replacing the expected value for years past by actual values).

The screens of this tool are presented in the appendix to this paper. By discussing in turn the various screens, we give the reader some sense of how this product operates. The main menu of this tool (labeled A.1) offers two sets of options, which correspond to the two main functions of the tool; we discuss these functions in turn below.

4.1 Archival Function

The archival function allows the user to record, update and track cost information on all four investment cycles; we review in turn the four input forms of the four cycles.

- *Corporation Cycle.* This form (labeled A.1.1) queries the user for an identification of the corporation, a start date, then information pertaining to the upfront costs of the corporate cycle, which represent infrastructure costs. This form also queries the user about a factor of episodic costs which is determined at the corporate level: library operation costs. The other relevant cost factors will be back-linked to the corporation's record in subsequent data entries: If, subsequently, a user creates a domain cycle

entry and declares it as a domain of this corporation, then its associated costs will be charged as costs to this corporation.

This form is usually filled by a corporate manager at the start of a reuse initiative.

- *Domain Engineering Cycle.* This form (labeled A.1.2) queries the user for an identification of the corporation and an identification of a domain under the corporation. Then the user is prompted to provide a start date, and a domain analysis cost estimate. Components that are subsequently created under this domain cycle will be back-linked to this record, and will have their costs and benefits duly tallied against this domain record.

  This form is usually filled by a domain engineer at the start of a domain engineering initiative.

- *Application Engineering Cycle.* This form (labeled A.1.3) queries the user for an identification of the corporation and an identification of the application under the corporation. Then it prompts the user for information on the start date of the activity, as well as lifecycle costs. In order to derive lifecycle costs, the system queries the user for information on upfront reuse adoption costs, as well as information on the make-up of the application: code developed from scratch; code that is reused verbatim (black box reuse); and code that is adapted (white box reuse). For reused assets (black box, white box), the system merely requests their identification (under the form: (domain name, component name)); it will derive their cost from the component's record in the database, possibly using default values (RBP, RWP).

  This form is usually filled by a development project manager at the end of a development project.

- *Component Engineering Cycle.* This form (labeled A.1.4) queries the user for an identification of the component, which includes three fields: corporation name, domain name, component name. Then it prompts the user for information related to the development and maintenance costs of the component. This information is used subsequently to quantify upfront costs of the component cycle, as well as potential quality gains and productivity gains. Productivity gains are obtained by composing the estimated (possibly by default options) of the sale price with the estimated frequency of reuse. Quality gains are quantified by means of operating cost differential (OCD), which measures the savings of operating/ maintenance costs over the length of the component's investment cycle.

  This form is usually filled by a component developer (asset producer) at the end of a component development step.

4.2 Analytical Function

The analytical function is invoked at a time when the investment cycle of interest is duly documented, and we need to review the main cost factors then compute the investment functions that interest us. This function offers four screens, to parallel the four screens of the archival function. The screens of the analytical function have a common structure, which includes five fields:

- *Identification*, which identifies the cycle of interest, as per the details given in the archival function.
- *Documentation*, which provides information on the main cost drivers of the cycle at hand.
- *Cost Factors*, which displays the upfront investment costs and the episodic costs and benefits per year for the length of the investment cycle.
- *Investment Parameters*, which displays generic investment parameters, such as: cycle length, discount rate, and organizational structure.
- *Economic Functions*, which displays a menu of the investment quantification functions that we may invoke to assess the attractiveness of an investment.

The identification field is discussed above, under the archival function. The fields of cost factors, investment parameters and economic function are the same across investment cycles. In the sequel, we briefly discuss the documentation field for all four screens.

- For the corporation investment cycle (see screen labeled A.2.1), the documentation field displays the list of all the domains and applications that are included under the corporation.
- For the domain engineering cycle (see screen labeled A.2.2), the documentation field displays the list of applications that fit under the domain.
- For the application engineering cycle (see screen labeled A.2.3), the documentation field displays the list of components that are reused in the development of the application, classified into black box reuse and white box reuse.
- For the component engineering cycle (see screen labeled A.2.4), the documentation field displays the expected (or observed) frequency of black box reuse and white box reuse, given by year.

Clicking on the *calculate* key produces the *Result* screen (labeled A.3), which identifies the cycle of interest and returns the results that have been selected in the report screen.

4.3 Default Arguments

The *Default Arguments* screen (labeled A.4) is accessible to privileged users, and enables these users to define default values for constants that are used throughout the system.

5 ILLUSTRATION

In this section, we consider a sample application of our model, and derive the NPV and PI for all investments cycles, using the default values of relevant parameters. We find that not all stakeholders have adequate incentives to collaborate; then we depart from the default values and find that then, all stakeholders are duly motivated to make reuse happen. This example illustrates how a manager can fine-tune relevant parameters to ensure all stakeholders will cooperate to achieve corporate goals.

5.1 Sample Data

To illustrate our cost model, we have taken the following sample data: We consider a corporation, which starts a reuse initiative in 1997 ($SD_\rho = 1997$), and wants to compute its return on investment within three years ($Y = 3$), using the discount rate of 0.15 ($d = 0.15$). The domain engineering team initiates a domain engineering initiative in 1997, with 5 components developed in 1997 and 5 in 1998, for a total of 10 components. The application engineering team develops two projects, one in 1998 and one in 1999. Cost and benefit details are given below.

- *Component engineering cycle.* We consider a reusable asset of size 5 KLOC. Its custom development cost (say, E_γ) can be estimated using traditional COCOMO [Boehm 1981], and its development for reuse cost can be estimated from industry figures ($RCWR$) given in [Poulin 1997a]. We consider that residence costs for this asset are 0.5 person months per year, and maintenance costs are $0.1 \times E_\gamma$ per year. We further assume that the reuse frequency of this asset for years 1997 to 2000 is given by the following table:

Year	1997	1998	1999	2000
Black Box	0	1	2	0
White Box	0	2	1	0

Also, we take the default values for black box price and white box price:

$$BP_\gamma(y) = 0.6 \times E_\gamma, \ \forall y,$$
$$WP_\gamma(y) = 0.2 \times E_\gamma, \ \forall y.$$

- *Domain engineering cycle.* We consider a domain engineering initiative that is initiated in 1997; we assume that in the first year, the domain engineering team performed domain analysis, and developed five assets, and that in 1998 they developed five more reusable assets. For the sake of simplicity, we assume that all the assets have the same characteristics as that which we studied above in component engineering. We use industry figures given by [Poulin 1997a] to estimate the domain analysis costs from the cost of developing the reusable assets.

- *Application Engineering cycle.* We consider two applications, one developed in 1998 and including 10 assets reused in black box, 20 assets reused in white box, and 100 KLOC of custom developed code; and another developed in 1999 and including 20 assets reused in black box, ten assets reused in white box, and 50 KLOC of custom developed code.

- *Corporate cycle.* We consider that the initial corporate investment in reuse infrastructure is 50 person months, and we let corporate costs be the sum of domain costs and corporate benefits be the sum of application benefits.

5.2 Results with Default Values

For each of the four cycles, we give a summary table of costs and benefits by year, then we give the *Net Present Value* and the *Profitability Index.*

- *Component Engineering cycle.* Cost / benefit table.

Year, y	1997	1998	1999	2000
$C(y)$	29.12	2.32	2.32	2.32
$B(y)$	0.00	18.20	25.48	0.00

  Whence we derive,

  $NPV = 0.67$ PM.

  $PI = 1.02$.

  This is very small, and hardly justifies the investment. Better results can be expected for higher reuse frequencies, or a longer investment cycle.

- *Domain engineering cycle.* Cost / benefit table.

Year, y	1997	1998	1999	2000
$C(y)$	245.70	136.50	18.00	18.00
$B(y)$	0.00	182.00	254.80	0.00

Whence we derive,

$NPV = -13.46$ PM.

$PI = 0.96$.

Clearly advocates against an investment; poor showing is due to the small return on investment of component engineering, which is further aggravated by domain analysis costs. Better results can stem from higher reuse frequencies, or a longer investment cycle.

- *Application engineering cycle.* Cost / benefit table, CS98.

Year, y	1998	1999	2000	2001
$C(y)$	182.00	0.00	0.00	0.00
$B(y)$	378.23	87.29	87.29	87.29

Whence we derive,

$NPV = 395.52$ PM.

$PI = 3.17$.

Cost / benefit table, CS99.

Year, y	1999	2000	2001	2001
$C(y)$	254.80	0.00	0.00	0.00
$B(y)$	423.53	67.99	67.99	67.99

Whence we derive,

$NPV = 323.96$ PM.

$PI = 2.27$.

Good showing for both applications; better for the second application because it has less custom developed code and more black box reused (vs. white box reused) code; these applications show a high (higher than average?) ratio of reusable code. This may indicate that factors RBP and RWP have been selected too low.

- *Corporate cycle.* Cost / benefit table.

Year, y	1997	1998	1999	2000
$C(y)$	295.70	136.50	18.00	18.00
$B(y)$	0.00	378.23	510.82	155.28

Whence we derive,

$NPV = 612.13$ PM.

$PI = 1.97$.

Good showing, due to benefits of application engineering. Note that the cost column is identical to the same column for domain engineering; and that the benefit column is the sum, year by year, of the benefits of application engineering.

The following table summarizes the results of this experiment.

Cycle	NPV	PI
Component	0.67	1.02
Domain	-13.46	0.96
Application		
C98	395.52	3.17
C99	323.96	2.27
Corporate	612.13	1.97

5.3 Results with Adjusted Values

Using the default values, we find that the domain engineering function has a negative NPV and a PI which is smaller than 1. Under the current incentive structure, domain engineering has no incentive to initiate domain engineering initiatives. The obvious culprit in this situation are the relative black box price and the relative white box price; they are not sufficiently high to amortize the costs of domain engineering. We change these values as follows, while still satisfying the conditions set forth in section 2.4.2.

$RBP = 0.8$.

$RWP = 0.25$.

For each of the four cycles, we give a summary table of costs and benefits by year, then we give the *Net Present Value* and the *Profitability Index*.

- *Component Engineering cycle*. Cost / benefit table.

Year, y	1997	1998	1999	2000
$C(y)$	29.12	2.32	2.32	2.32
$B(y)$	0.00	23.66	33.67	0.00

Whence we derive,

$NPV = 11.62$ PM.

$PI = 1.34$.

- *Domain engineering cycle.* Cost / benefit table.

Year, y	1997	1998	1999	2000
$C(y)$	245.70	136.50	18.00	18.00
$B(y)$	0.00	236.60	336.70	0.00

Whence we derive,

$NPV = 95.93$ PM.

$PI = 1.26$.

- *Application engineering cycle.* Cost / benefit table, CS98.

Year, y	1998	1999	2000	2001
$C(y)$	236.60	0.00	0.00	0.00
$B(y)$	378.23	87.29	87.29	87.29

Whence we derive,

$NPV = 340.92$ PM.

$PI = 2.44$.

Cost / benefit table, CS99.

Year, y	1999	2000	2001	2001
$C(y)$	336.70	0.00	0.00	0.00
$B(y)$	423.53	67.99	67.99	67.99

Whence we derive,

$NPV = 242.06$ PM.

$PI = 1.72$.

- *Corporate cycle.* Cost / benefit table.

Year, y	1997	1998	1999	2000
$C(y)$	295.70	136.50	18.00	18.00
$B(y)$	0.00	378.23	510.82	155.28

Whence we derive,

$NPV = 612.13$ PM.

$PI = 1.97$.

Note that the new values that we have taken for RBP and RWP have no impact on the costs and benefits at the corporate level, because they affect the flow of credit within the corporation (greater flow from application engineering to domain engineering).

The following table summarizes the results of this experiment.

Cycle	NPV	PI
Component	11.62	1.34
Domain	95.93	1.26
Application		
C98	340.92	2.44
C99	242.06	1.72
Corporate	612.13	1.97

Under the new values of RBP and RWP, all four values of NPV are positive, and all four values of PI are greater than 1. Under these conditions, all stakeholders are expected to cooperate to make reuse happen.

6 CONCLUSION AND PROSPECTS

In this paper, we submit the following premises:

- There are four decisions that arise in the practice of software reuse: a *corporate level decision*, of whether to introduce reuse in the practice of software development; a *domain engineering decision*, of whether to initiate a domain analysis/domain engineering initiative; an *application engineering decision*, of whether to introduce reuse practices for a specific development project; a *component engineering decision*, of whether it is worthwhile to develop a specific component to serve a group of project teams.

- All these decisions can be quantified in economic terms, and justified by means of an economic rationale.

- All four decisions can be modeled as investment decisions; consequently, their economic rationale can be quantified by traditional investment analysis functions (such as ROI, NPV, IRR, ARBV, PI, ARR, PB).

We have derived a cost model that reflects these premises, and have shown how the investment cycles that we have highlighted in these models feed cost/ benefit information into each other in a cascade pattern. We have analyzed how this model can be seen to generalize existing reuse cost models, and have discussed a prototype tool that supports the proposed model by providing an archival function and an analysis function. Furthermore, we have used this model to illustrate what it means to *make reuse happen*, by showing how to define a reward/ incentive structure that encourages/ promotes reuse across the organization.

Among our perspectives for future work, we envisage to enhance the prototype, to incorporate gains in time-to-market into the model, and to further analyze the impact of reuse organizations on our cost models.

ACKNOWLEDGMENTS

This work is supported in part by NASA IV& V Laboratory in Fairmont, WV. The authors are grateful to C. Calhoun and D. McDonough, NASA and to J. Estep, WVHTF, for their interest and support.

REFERENCES

D.M. Balda and D.A. Gustafson. Cost estimation models for the reuse and prototype software development lifecycles. *ACM SIGSOFT Software Engineering News*, 15(3):42–50, July 1990.

B.H. Barnes and T.B. Bollinger. Making reuse cost effective. *IEEE Software*, 8(1):13–24, jan. 1991.

Paul R Berney and Stanley J Garstka. *Accounting: Concepts and Applications.* Richard D Irwin, Inc., Homewood, Illinois, 1984.

B.W. Boehm. *Software Engineering Economics.* Prentice Hall, Englewood Cliffs, NJ, 1981.

B.W. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby. Cost models for future software lifecycle processes: COCOMO 2.0. *Annals of Software Engineering*, 1:57–94, September 1995.

T.B. Bollinger and S.L. Pfleeger. Economics of reuse: Issues and alternatives. *Information and Software Technology*, 32(10):643–652, December 1990.

R.J. Bowes, T.R. Huber, and R.O. Saisi. Informal technical report for the software technology for adaptable, reliable systems (stars) acquisition handbook —final. Technical report, DSD Laboratories Inc., for Airforce Material Command, Hanscom Air Force Base, Ma, October 1992.

R.J. Bowes, T.R. Huber, and R.O. Saisi. Informal technical report for the software technology for adaptable, reliable systems (stars) acquisition handbook —final. Technical report, DSD Laboratories Inc., for Airforce Material Command, Hanscom Air Force Base, Ma, November 1992.

G. Caldiera and V. Basili. Identifying and qualifying reusable software components. *IEEE Computer*, 24(2):61–70, february 1991.

B. Coulange. *Software Reuse*. Springer Verlag, London, UK, 1998.

P. Devanbu, S. Karstu, W. Melo, and W. Thomas. Analytical and empirical evaluation of software reuse metrics. In *Proceedings, International Conference on Software Engineering*, Berlin, Germany, 1996. IEEE Press.

D. Fafchamps. Organizational factors and software reuse. *IEEE Software*, 11(5):31–41, september 1994.

J. Favaro. A comparison of approaches to reuse investment analysis. In *Proceedings, Fourth International Conference on Software Reuse*, pages 136–145, Orlando, FL, April 1996.

J. Favaro, K. Favaro, and P. F. Favaro. Value based software reuse investment. *Annals of Software Engineering*, 5:5–52, 1998.

Center for Software Engineering. COCOTS. Technical report, University of Southern California, Los Angeles, CA, June 1999.

W.B. Frakes and C. Terry. Reuse level metrics. In W.B. Frakes, editor, *Proceedings, Third International Conference on Software Reuse*, pages 139–148, Rio de Janeiro, Brazil, November 1994.

T. Frazier. Economics of software reuse, working group report. In *Second Reuse and Education and Training Workshop*, Morgantown, WV, October 1993.

J.E. Gaffney and R.D. Cruickschank. A general economics model of software reuse. In *Proceedings, International Conference on Software Engineering*, pages 327–337, Melbourne, Australia, May 1992.

J.E. Gaffney and Th. Durek. Software reuse —key to enhanced productivity: Some productivity models. *Information and Software Technology*, 31(5):258–267, June 1989.

E. Guerrieri, L.A. Lashway, and T.B. Ruegsegger. An acquisition strategy for populating a software reuse library. In *National Conference on Software Reusability*, Washington, DC, July 1989.

B. Henderson-Sellers. The economics of reusing library classes. *Journal of Object Oriented Programming*, 6(4):43–50, July-August 1993.

James C. Van Horne. *Financial Management and Policy, Sixth Edition.* Prentice-Hall Inc., Englewood Cliffs, NJ, 1983.

Charles T Horngren. *Introduction to Management Accounting Fifth Edition.* Prentice-Hall Inc, Englewood Cliffs, NJ, 1981.

C. Jones. Economics of software reuse. *IEEE Computer*, 27(7):106–107, July 1994.

B.J. Kain. Measuring the roi of reuse. *Object Magazine*, 4(3):48–54, June 1994.

K.C. Kang and L.S. Levy. Software methodology in the harsh light of economics. *Information and Software Technology*, 31(5), June 1989.

H. Leach. *Software Reuse*. McGraw Hill, New York, NY, 1997.

W.C. Lim. A cost justification model for software reuse. In *Proceedings, Fifth Workshop on Institutionalizing Reuse*, University of Maine, Orono, October 1992.

W.C. Lim. Effects of reuse on quality, productivity and economics. *IEEE Software*, 11(5):23–30, september 1994.

W.C. Lim. Reuse economics: A comparison of seventeen models and directions of future research. In *Proceedings, Fourth International Conference on Software Reuse*, pages 41–50, Orlando, FL, April 1996.

R. Malan. Software reuse: A business perspective. Technical report, Hewlett Packard Laboratories, February 1993.

R. Malan and K. Wentzel. Economics of reuse, revisited. Technical Report HPL-93-31, Hewlett Packard Laboratories, April 1993.

J. Margano and T.E. Rhoads. Software reuse economics: Cost benefit analysis on a large scale ada project. In *Proceedings, International Conference on Software Engineering*, pages 338–348, Melbourne, Australia, May 1992.

G. Mayobre. Using code reusability analysis to identify reusable components from software related to an application domain. In *Proceedings, Fourth International Workshop on Software Reuse*, Reston, Va, November 1991.

W.L. Melo, L.C. Briand, and V.R. Basili. Measuring the impact of reuse on quality and productivity in object oriented systems. Technical Report TR-95-2, University of Maryland, Department of Computer Science, January 1995.

A. Mili, S. Fowler, R. Gottumukkala, and L. Zhang. An Integrated Cost Model for Software Reuse. In *Proceedings, 22nd International Conference on Software Engineering*, Limerick, Ireland, June 4-11, 2000, pp 157-166.

A. Mili, S. Fowler, R. Gottumukkala, and L. Zhang. Software reuse cost estimation. Technical report, CSEE Dept, West Virginia University, `http://www.csee.wvu.edu/reuseroi/`, November 1999.

A. Mili, Sh. Yacoub, E. Addy, and H. Mili. Towards an engineering discipline of software reuse. *IEEE Software*, 16(5):22–31, September/October 1999.

H. Mili, A. Mili, Sh. Yacoub, and E. Addy. Reuse-Based Software Engineering: Techniques, Organization and Measurement. New York, NY: John Wiley and Sons, 2001.

R. Mili. Return on investment of reusable components: Analytical and empirical approaches. Technical report, University of Ottawa, Ottawa, Ontario, Canada, December 1996.

NATO. Standard for the development of reusable software components. Technical Report 18, NATO, August 1991.

Y. Pant, B. Henderson Sellers, and J.N. Verner. Generalization of object oriented components for reuse: Measurement of effort and size change. *Journal of Object Oriented Programming*, 9(2):19–31,41, May 1996.

J. Poulin. *Measuring Software Reuse: Principles, Practices and Economic Models*. Addison Wesley, 1997.

J.S. Poulin. The economics of software product lines. *International Journal of Applied Software Technology*, 3(1):20–34, March 1997.

J.S. Poulin and J.M. Caruso. A reuse metrics and return on investment model. In *Advances in Software Reuse: Proceedings of the Second International Workshop on Software Reusability*, pages 152–166, Lucca, Italy, March 1993.

G.E. Raymond and D.M. Hollis. Software reuse economics model. In *Proceedings of WADAS'91: 7th Washington Ada Symposium Summer SIGAda Meeting*, pages 141–155, McLean, Va, June 1991.

D.J. Reifer. *Practical Software Reuse*. John Wiley and Sons, New York, NY, 1997.

S.R. Schach. The economic impact of software reuse on maintenance. *Journal of Software Maintenance, Research and Practice*, 6(4):185–196, July-August 1994.

S.R. Schach and X.F. Yang. Metrics for targeting candidates for reuse: An experimental approach. In *Proceedings, ACM Symposium on Applied Computing*, pages 379–383, 1995.

D. Schimsky. Software reuse —some realities. *Vitro Tech Journal*, 10(1):47–57, 1992.

B. Stevens. Linking software reengineering and reuse: An economic motivation. *CASE Trends*, pages 24–36, March 1993.

Jerry A. Viscione. *Financial Analysis: Tools and Concepts*. Publications Division, National Association of Credit Management, New York, NY, 1984.

A APPENDIX: SOFTWARE REUSE COST ESTIMATION TOOL

A.1 Input Screens

A.1.1 Corporation Input Screen

A.1.2 Domain Engineering Input Screens

A.1.3 Application Engineering Input Screen

A.1.4 Component Engineering Input Screen

A.2 Report Screens

A.2.1 Corporate Engineering Report

A.2.2 Domain Engineering Report

A.2.3 Application Engineering Report

A.2.4 Component Engineering Report

A.3 Results Screen

A.4 Default Constants Screen