An Integrated Cost Model for Software Reuse

Senta Fowler Chmiel

Dissertation submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Science

Ali Mili, Ph.D., Chair
Jim Mooney, Ph.D.
George Trapp, Ph.D.
Michael Palmer, Ph.D.
Tim Pearson, Ph.D.

Morgantown, West Virginia
2000

Keywords: Software Reuse, Economics, Cost Models

Copyright 2000 Senta F. Chmiel
Abstract

An Integrated Cost Model for Software Reuse

Senta Fowler Chmiel

Several cost models have been proposed and assessed in the past for predicting and analyzing the costs associated with software reuse. The integrated cost model, which is presented in this thesis, is based on the premise that there are four decisions in the software reuse lifecycle: the decision to adopt a reuse-based software paradigm; the decision to initiate a domain engineering initiative; the decision to use reusable assets in the development of a given application; and the decision to develop reusable assets. The integrated cost model further considers that all these decisions must be justified by an economic rationale, and all can be modeled as investment decisions. This model, a generalization of existing reuse cost models, illustrates that these investment cycles feed information into each other in a cascade pattern.
Acknowledgements

From the start of the Ph.D. program in 1993, the journey to finish this dissertation has been long and hard. I have changed topics and chairs at least three times, got a full-time job as a database administrator, got married in 1997 and had a daughter in 1999. Without the help of the many individuals I will mention below this goal would have never been accomplished.

First and foremost I would like to thank Dr. Ali Mili for taking me on as a student and supporting me the whole time in the last two years that I have been under him. His support enabled me to finish my thesis and have opportunities to present my research at conferences like ICSE 2000 in Limerick, Ireland. He also provided me with graduate students (Lisa Zhang and Ravi Gottumukkala) to work on the prototype demo designed to support/validate this integrated cost model and with financial support from the WVU CSEE department and NASA IV&V Laboratory in Fairmont, WV.

I would also like to thank my committee (Dr. Pearson, Dr. Palmer, Dr. Mooney, and Dr. Trapp). Dr. Pearson joined my committee when Dr. Spangler left WVU. Dr. Trapp kindly agreed to join my committee when I was short a committee member. Dr. Palmer and Dr. Mooney sacrificed their valuable time to help me be able to accomplish this tremendous task.

Furthermore, I would like to thank my family without whose support I would not be able to accomplish this goal. My father edited multiple copies of my thesis for grammar and spelling. My husband, Walter, helped me translate the economic functions into programmable formulas. My mother and mother-in-law served as babysitters to Jennifer, my daughter, this past year so that I had the time to write and defend my thesis.

Lastly, I would like to thank Sister Doris Faber for her emotional support and help. Her constant reminders at the swimming pool to get this thesis done gave me that “push” to finally finish.
# Table of Contents

Abstract ............................................................................................................................... ii

Acknowledgements ............................................................................................................ iii

Table of Figures ................................................................................................................. ix

Table of Tables .................................................................................................................... x

Part 1 Software Reuse Economics ..................................................................................... 1

Chapter 1 Software Reuse ................................................................................................... 2
  1.1 Software Reuse and Software Design ....................................................................... 3
    1.1.1 Software Reuse versus Software Design ............................................................ 3
    1.1.2 Software Reuse ................................................................................................... 4
    1.1.3 Software Reuse Advantages ............................................................................... 5
      1.1.3.1 Earlier Time-to-Market ............................................................................... 5
      1.1.3.2 Quality Gains ............................................................................................... 6
      1.1.3.3 Productivity Gains ....................................................................................... 6
  1.2 Software Reuse Products and Processes ................................................................. 6
    1.2.1 Software Reuse Products ................................................................................. 6
    1.2.2 Software Reuse Agents ................................................................................... 7
    1.2.3 Software Reuse Processes ............................................................................... 9
  1.3 State of the Art and State of Practice ...................................................................... 13
    1.3.1 Software Reuse Initiative ................................................................................. 13
    1.3.2 Software Reuse Barriers ................................................................................... 13
  1.4 Conclusion ............................................................................................................... 15

Chapter 2 Software Engineering Economics .................................................................... 16
  2.1 Software Engineering Lifecycle .............................................................................. 17
  2.2 Software Cost Models ............................................................................................. 20
    2.2.1 Algorithmic Models ......................................................................................... 21
      2.2.1.1 Wolverton .................................................................................................. 21
      2.2.1.2 Walson-Felix ............................................................................................. 21
      2.2.1.3 COCOMO ................................................................................................. 22
      2.2.1.4 Putnam/Norden .......................................................................................... 24
    2.2.2 Function Point Analysis ................................................................................... 24
  2.3 Development Time .................................................................................................. 26
  2.4 Conclusion ............................................................................................................... 27

Chapter 3. Investment Analysis ........................................................................................ 28
  3.1 An Investment Cycle ............................................................................................... 29
  3.2 Present Value ........................................................................................................... 29
  3.3 Net Present Value .................................................................................................... 30
    3.3.1 Definition ......................................................................................................... 30
    3.3.2 Advantages ....................................................................................................... 30
5.1.4 Reuse Organization ................................................................. 49
5.1.5 Scope .................................................................................... 50
5.1.6 Hypothesis ........................................................................... 50
5.1.7 Viewpoint ............................................................................ 51
5.1.8 Instantiation ....................................................................... 51
5.2 Various Economic Models ...................................................... 51
  5.2.1 Barnes and Bollinger .......................................................... 54
  5.2.2 Bollinger and Pfleeger ......................................................... 54
  5.2.3 Schimsky ........................................................................... 56
  5.2.4 Margono and Rhoades ....................................................... 56
  5.2.5 Poulin/Carouso ................................................................. 59
  5.2.6 Lim ................................................................................... 61
  5.2.7 Favaro/Favaro/Favaro ....................................................... 61
  5.2.8 Kain .................................................................................. 63
  5.2.9 Malan and Wetzel .............................................................. 64
  5.2.10 Gaffney and Durek .......................................................... 65
  5.2.11 Kang and Levy ................................................................. 67
  5.2.12 Balda .............................................................................. 68
  5.2.13 COCOMO 2.0 ................................................................. 69
  5.2.14 COCOTS .......................................................................... 71
    5.2.14.1 COTS ....................................................................... 71
    5.2.14.2 COCOTS .................................................................... 72
  5.2.15 Stevens ........................................................................... 74
  5.2.16 Raymond and Hollis ....................................................... 75
  5.2.17 Gaffney and Cruickshank .............................................. 76
  5.2.18 Poulin ............................................................................. 77
  5.2.19 Lim ................................................................................. 78
  5.2.20 Frakes and Terry ............................................................. 79
  5.2.21 Devanbu ........................................................................ 80
  5.2.22 Mili ............................................................................... 81
  5.2.23 Hancock ........................................................................ 82
  5.2.24 Favaro ........................................................................... 82
5.3 Conclusion ............................................................................. 84

Chapter 6. Quantifying Software Reuse Costs and Benefits .................. 85
  6.1 Productivity Gains ................................................................. 86
  6.2 Quality Gains ....................................................................... 87
  6.3 Shortened Time-to-Market .................................................... 88
  6.4 Conclusion ........................................................................... 90

Chapter 7. Component Engineering Cycle ........................................ 91
  7.1 Investment Costs ................................................................. 92
  7.2 Episodic Costs ..................................................................... 93
  7.3 Episodic Benefits ............................................................... 94
  7.4 Summary ............................................................................ 96
  7.5 Example of Component Engineering Costs and Benefits ......... 96
Chapter 8. Application Engineering Cycle ................................................................. 99
  8.1 Up-front Costs .................................................................................................. 100
  8.2 Periodic Costs ................................................................................................. 101
  8.3 Periodic Benefits ............................................................................................ 101
  8.4 An Illustrative Example of Application Engineering Benefits and Costs ........ 102
    8.4.1 Application cs98 ..................................................................................... 103
    8.4.2 Application cs99 ..................................................................................... 104

Chapter 9. Domain Engineering Cycle ...................................................................... 106
  9.1 Up-front costs ................................................................................................. 107
  9.2 Episodic Costs ............................................................................................... 108
  9.3 Episodic Benefits ........................................................................................... 108
  9.4 An Illustrative Example of Domain Engineering Costs and Benefits ............ 108

Chapter 10 Corporate Engineering Cycle .................................................................. 110
  10.1 Investment Costs .......................................................................................... 111
  10.2 Episodic Costs ............................................................................................. 111
  10.3 Episodic Benefits ........................................................................................ 112
  10.4 An Illustrative Example of Corporate Costs and Benefits ......................... 112

Chapter 11 An Example of Application of the Integrated Cost Model ....................... 115
  11.1 Component Engineering Cycle ................................................................... 116
  11.2 Domain Engineering Cycle ......................................................................... 119
  11.3 Application Engineering Cycle ................................................................... 121
    11.3.1 Application cs98 .................................................................................. 121
    11.3.2 Application cs99 .................................................................................. 123
  11.4 Corporate Engineering Cycle ...................................................................... 125
  11.5 Summary of the Calculation of the Economic Functions for this Example ...... 127

Chapter 12. An Automated Tool ................................................................................. 130
  12.1 Prototype Design ......................................................................................... 131
  12.2 Archival Function ....................................................................................... 134
    12.2.1 Corporation Cycle Input ...................................................................... 136
    12.2.2 Domain Cycle Input ............................................................................ 137
    12.2.3 Application Cycle Input ...................................................................... 137
    12.2.4 Component Cycle Input ...................................................................... 138
  12.3 Analytical Function ..................................................................................... 139

Chapter 13. Conclusion .............................................................................................. 142
  13.1 Motivation .................................................................................................... 143
  13.2 Summary ...................................................................................................... 143
  13.3 Assessment ................................................................................................... 143
  13.4 Comparison .................................................................................................. 143
  13.5 Prospects ...................................................................................................... 144

Bibliography .......................................................................................................... 145
Table of Figures

Figure 1 Software Reuse Lifecycle ................................................................. 12
Figure 2 Integrated Software Reuse Cost Model .............................................. 42
Figure 3 Cascade of Costs through Investment Cycles .................................... 45
Figure 4 Investment Cycle Balance Sheets ...................................................... 46
Figure 5 Archival and Analytical Functions of the Automated Tool ............... 135
Table of Tables

Table 1 Reuse Organizations........................................................................................................ 9
Table 2 Activities and Phases of the Software Lifecycle.......................................................... 19
Table 3 Function Point to Lines of Code Equivalences ....................................................... 26
Table 4 Software Reuse Economics Models ......................................................................... 53
Table 5 Economic Functions.................................................................................................. 83
Table 6 RCWR Constants ................................................................................................... 93
Table 7 Summary of Engineering Cycle’s Economic Functions ....................................... 127
Part I. Software Reuse Economics
Chapter 1. Software Reuse
1.1 Software Reuse and Software Design

1.1.1 Software Reuse versus Software Design

In other engineering disciplines, reuse is part of good engineering design. With software reuse, hardware reuse is unheard of. However, in software engineering, software reuse is an issue that makes it a topic of much research as well as the subject of specialized conferences and university courses.

The key to software reuse is the development and cataloging of reusable assets. These reusable assets can be programs, subprograms, designs, requirement documents, etc. In chemical, mechanical, and civil engineering, the components that make up these products are specific components, mechanical parts, and building materials with well defined properties and usage protocols. However in software engineering, software reuse products are composed of reusable assets that are information rich and difficult to characterize, match, and capture the properties of. In software design, the product is dynamic with many different procedures, abstractions, specifications, and parameters. In hardware design, the product is concrete and fixed in its definition. Also hardware components can be classified and cataloged, which is generally very difficult for software components.

For example, consider the building of a car. A car consists of a standard architecture. It consists of a front end, passenger compartment, and back end. The front end contains the engine and wheels. The passenger compartment contains seats, a steering wheel, and dash. The back end consists of wheels, gas tank, and trunk. Each of these components of the car each is comprised of a standard architecture also. For example, the dash typically has a fuel gauge, odometer, mileage indicator, oil indicator, lights, and glove compartment. Likewise the engine consists of a radiator, spark plugs, carburetor, oil filter, air filter, brake fluid, etc. If any of these components were missing, the car would not run. The design of these components may be different i.e., the mileage indicator may be electronic or it may be mechanical but the standard components are there. Furthermore, it may be modified to contain a tape deck or CD player. These components can be classified on the basis of factors like vendor, size, etc. For example, a wheel can be classified by vendor (Firestone, Michelin, GoodYear) and size R13, R14, R15 and type AT, XT, Radial, etc. As a result of this standard architecture, vendors of the car components know exactly what they are building for. Thus, one would not build the engine of a car and not include the spark plugs.

Software engineering, unlike mechanical engineering, does not have a standard architecture for software design. It also does not have standard components or procedures for classifying/characterizing its components. Software engineering does not have a “automobile design standard.” Thus, cataloging the parts of a software product is not as easy as cataloging car parts. Furthermore, designing software components is dynamic and difficult because missing parts of the software may not be obvious. Since there is no software design standard, how would a software engineer know his program is...
missing its “spark plugs”? In software engineering, components are not standardized like a wheel, steering wheel, or dash because components can be functions, procedures, design patterns, experiences, and other knowledge. Since there is no standard for developing such components, the software engineer is faced with the dilemma of how to characterize and develop a software component. Furthermore, since there is no software architectural standard or software design standard, reuse is not an integral part of the design as in the other engineering disciplines.

Software reuse differs from software design, unlike “hardware reuse” that is an integral part of engineering design. Good software design advocates design software from reusable assets and produce software systems with the goal that they also may be reused [1]. In an attempt to make software reuse an integral part of software design, software reuse deals with producing reusable assets (domain engineering) and with exploiting reusable assets (application engineering). However, unlike other engineering disciplines, the cost to make software reuse an integral part of software design may outweigh the benefits.

In mechanical engineering, there is no such thing as “autopart reuse.” However, to build this car, specific car parts must be integrated. These parts (tires, steering wheel, drive shaft) not only exist but are classified, easy to retrieve, and possess well defined properties. If a software engineer were to build a car, the software components to build this simulation would likely not exist. If these components did exist, the engineer would need to address the issues of software reuse to integrate them into the final product. To run the specific simulation of the tires or steering wheel requires the knowledge/understanding of the parameters that make drive these programs. The mechanical engineer does not have to deal with the issues of such parameters. Unlike the mechanical engineering who has “autopart reuse” built into the design process, software engineers must address software reuse specifically in the design process.

Thus, in other engineering disciplines, reuse is an integral part of good engineering design [1]. However in software engineering, most modules are not engineered for reuse because they do not solve a generic problem. That is because they are difficult to understand and too expensive to modify, and because they are poorly documented. Furthermore, building for reuse is technically more difficult and costs more than traditional development, and developers and project managers are not motivated to invest [2].

1.1.2 Software Reuse

What is software reuse? “Software reuse is the process whereby an institution defines a set of systematic operating procedures to produce, classify and retrieve software artifacts for the purpose of using them in its development activities” [3]. Software reuse is systematic activity whereby the production and consumption of the reusable artifact are independent, with an emphasis on form, rather than function, and on language compatibility.
Software reuse systems are developed, not manufactured, and developed in the “creative, intellectual sense, rather than produced in the manufacturing sense” [4]. For this reason, unlike in other engineering disciplines, software reuse in some situations has become somewhat necessary. Software reuse is a consequence of good software design. If software is designed well, then it is produced from reusable assets.

Reuse artifacts come in many forms:
- Executable Code
- Source Code
- Requirements Specifications
- Designs
- Test Data
- Documentation
- Architectures

For a component (executable/source code) to be reusable it must be useful, usable/understandable, adaptable, portable and of high quality. Reuse tends to be successful when the domain is narrow and stable with a small number of abstractions and a well-understood vocabulary. When designing for reuse the key is to separate context from content and concept and to factor out commonality, i.e., isolate change [5]. According to Poulin [6] the general attributes of reuse are ease of understanding, functional completeness, reliability, good error and exception handling, information hiding, high cohesion and low coupling, and portability.

1.1.3 Software Reuse Advantages

Software reuse has the following advantages:
1. Earlier time-to-market
   - Reduction in the cost of development
2. Quality Gains
   - Increased reliability through use of reviewed and tested software [7]
   - Embodiment of organization standards in reusable components [8]
3. Productivity Gains
   - Effective use of specialists
   - Reduction in overall risk

1.1.3.1 Earlier Time-to-Market

One of the hidden benefits of reuse is that development must follow good software engineering practices and that prototyped projects tend to be designed and developed sooner due to the wide usage. This earlier development allows the application to enter the market sooner and reap the additional benefits that result from this earlier marketing of the application. If the application was built without reuse it would run the risk of
having another application take over the market and losing the additional benefits gained with early marketing.

1.1.3.2 Quality Gains

By using reusable components, development costs may be reduced by 1-12%, defect rates by 10%, and maintenance costs by 20-50%. Productivity gains from 20-40% have been published for reuse [9]. A decrease in maintenance costs (up to 90%), has been reported when reusable code, code templates and application generators have been used to develop new systems [5]. The extra cost of designing and building reusable components and thoroughly testing them reduces the chance of failure and the need for further modification/upgrade. Moreover, the more a reusable component is used, the fewer defects it tends to have [10].

1.1.3.3 Productivity Gains

Reuse improves productivity because the lifecycle now requires less effort to obtain the same outcome and reduces the amount of time and labor needed to develop and maintain software products. Furthermore, because work products are used multiple times, the defect fixes from each reuse accumulate, resulting in higher quality. The defining characteristic of "good reuse is not the reuse of software per se but the reuse of human problem solving [11]." Reuse allows for this expertise to be extended to projects/companies that cannot afford such expertise, thereby improving productivity.

The producer and consumer of reusable assets can avoid duplication of effort by centrally maintaining the reuse components, managing their evolution, and propagating upgrades using reuse technology. Furthermore, using reusable black box components allows further enhancement and correction quicker and at a lower cost [10].

1.2 Software Reuse Products and Processes

1.2.1 Software Reuse Products

Software reuse products must be sufficiently generic to be reusable and sufficiently specific so that they are easy to instantiate. These products can be in the form of
1.2.2 Software Reuse Agents

In order to develop and/or use reusable software products, the organization needs to adapt a reuse organization schema. Fafchamps [31] and Basili [4] introduce five possible producer-consumer relationship models. In these models, the consumer designs and develops products with reusable components while the producer designs and develops these reusable components. Reuse design and development is dependent on the organization size, talent pool and structure, functional structure, i.e., whether the group is diverse with technical expertise, and divisional structure, i.e., specialists grouped into teams. The five models presented by Fafchamps [31] and Basili [4] are

- Lone Producer
- Nested Producer
- Pool Producer
- Team Producer
- Experience Factory

In the Lone Producer team, reuse services exist in two consumer teams that design, develop, and maintain reusable components. In the Nested Producer team, a product team with reuse services and expertise may have a secondary reporting line. In the Pool Producer team, two or more teams collaborate to produce and share components. In the Team Producer team, the reuse team handles the organizational structure. The experience factory is based on the quality improvement paradigm: characterize the environment, set the goals, choose the process for improvement, execute the processes of constructing the products, analyze the data and practices to evaluate, and package the experiences for future projects [4]. The advantages and disadvantages of each organization are summarized in Table 3 [4,31].
Within each of these organizations the following reuse-specific personnel are necessary:

- Reuse Manager
- Librarian
- Domain Analyst
- Application Engineer

The reuse manager is a software manager. This individual must have a knowledge of software management issues including software lifecycle, software project planning, software cost estimation, and programmer productivity. Furthermore, this manager must be fluent in software reuse organizations, software reuse economics, software reuse metrics and software reuse products. This manager must also be able to handle management issues such as team cohesion, communications overhead, turnover impact, loyalty conflict, project competition, library cohesion, and programmer incentives.

The librarian is responsible for providing a useful, usable, and efficient library to the producers and consumers. This individual must find the appropriate mechanism for representing and cataloging reusable assets. Hence, this individual needs skills in not only information retrieval but also in programming languages, logic, formal specifications, etc. The librarian is also responsible for the generality, correctness, reliability and clarity of the components of the library. The latter necessitates that the librarian is fluent in software quality and software metrics.

The domain analyst is critical to the success and failure of the reuse initiative. This individual investigates application domains to determine its worthiness of packaging the development experiences for future reuse. The domain analyst identifies commonly recurring components and/or problem solving patterns. Furthermore, he decides whether the compositional development (source code assets) or generative development (design patterns) paradigm is appropriate for the specific domain.

The application engineer decides whether to use the software assets as black box or white box components. Black box components are used verbatim while white box components are adapted or modified. If the components are adapted or modified, they lose their verification and must be re-tested. It is the job of the application engineer to decide if it is worth the risk of losing program quality and programmer productivity to adapt a component, to develop a component from scratch or to use the component as is in the application.
<table>
<thead>
<tr>
<th>Model</th>
<th>Advantage(s)</th>
<th>Disadvantage(s)</th>
</tr>
</thead>
</table>
| Lone Producer| Junior programmers that maintain components or experienced programmers that design and develop large components | Communication overhead
Work overload
Isolation
Multiple supporting lines
Priority selling |
| Nested Producer| Home team                                                                   | Double reporting
Resource diversion due to other tasks
Isolation from reuse peers
No specialization |
| Pool Producer | No organizational changes
Limited scope reuse program
Maximum of 3 teams
Sharing of common components reduces work redundancy | High communication overhead
Unresolved conflicts
Conflicting priorities with time-to-market, R&D, etc.
Not suited for long-term design with reuse strategy |
| Team Producer | One manager
Flexibility
Specialization
Ownership
Knowledge of other projects
Good for long-term reuse | Alienation of producers from consumers
Work with many projects
Communication and maintenance overhead |
| Experience Factory | Separated product development focus from learning and reuse focus
Consolidated and integrated Activities management and development processes formalized | Cost of instituting the program
Continual accumulation of evaluated experiences required |

Table 1 Reuse Organizations

1.2.3 Software Reuse Processes

Software reuse goes through the following processes:

- Component Engineering
- Domain Engineering
- Application Engineering
- Corporate Engineering

In component engineering, assets are developed for reuse. A reusable component, unlike a single use component, must satisfy a wide range of functional requirements and be as generic as possible.

In domain engineering, domain analysis occurs where a given application domain is analyzed to identify commonly recurring components or patterns. In this producer phase, the domain analyst must define reusable assets that are sufficiently generic to have good
reuse potential but sufficiently specific to be easy to instantiate and retrieve and he must
determine which functions, control structures, and data requirements are common
throughout the domain and if they exist as reusable components [12].

Application engineering uses black box components and white box components
developed in the component engineering phase to build applications. Black box
components are used verbatim upon retrieval from the reuse library, whereas white box
components are used after modification or adaptation. With white box component reuse
in an application, the component must be verified and tested to ensure that there is no loss
of productivity or loss of quality.

The integration of all of these engineering phases constitute corporate engineering. In
corporate engineering, the reuse infrastructure is set up to enable the component
development, component classification, domain analysis, and application development.
Corporate engineering includes management restructuring, reuse training, operational
procedural changes, etc. In corporate engineering, the producer (e.g., domain analyst)
develops reusable assets while a consumer (e.g., application engineer) exploits the
reusable assets developed by the producer.

The lifecycle of a reusable asset proceeds through the following phases [3]:
- Requirements Specification
- Development for Reuse/Reengineering for Reuse
- Verification for Reuse
- Storage
- Requirements Elicitation
- Retrieval
- Assessment
- Selection
- Integration and Verification

In the requirements phase, the producer attempts to get a complete description of the
problem/product and associated requirements and environment characteristics from the
consumer. In this phase, the producer learns from the consumer the number of users,
type of hardware/software to be used, and type of functionality this software product is
supposed to encompass. This phase typically includes a feasibility study to determine if
the solution is economically and technically feasible. From the requirements acquired in
the requirements phase, the producer must develop assets generically to allow for future
reuse in the development for reuse/reengineering for the reuse phase. Furthermore, the
producer may need to reengineer the existing process to allow for such reuse. After the
assets are developed for reuse, they are tested (verified and validated). In the storage
phase, the resulting asset is classified, stored, maintained in a repository for consumer
retrieval. This phase of the software asset lifecycle also includes the design and building
of the asset retrieval mechanism.
In the requirements elicitation phase, consumers define what they need, i.e., the functionality they require. In the retrieval phase, the consumers try to find their required functionality in a reuse repository. After retrieving the necessary assets from the repository, the consumer determines (1) what additional assets are needed, (2) if it is feasible to build the application from these reusable assets, (3) what code is needed to integrate these assets, and (4) what additional code is needed to fulfill the necessary requirements. Also during this phase, the consumer determines if it is more cost effective to modify the existing components or to build the component from scratch. After assessing the feasibility of the new code development and use of reusable components, the consumer (1) selects the needed reusable components, (2) codes the additional needed components, (3) incorporates the located and retrieved assets as is (black box reuse) or modifies these assets (white box reuse), and (4) codes the “glue code”. After the coding of the complete product and during the integration of the various black and white box components, the code is assessed to determine if it will adhere to the required functionality and verified to determine that it does not interfere with the functionality of the original retrieved components.

This software reuse lifecycle is illustrated in Figure 1. The link between the producer and the consumer is the software reuse library. The producer performs domain analysis and component specification, which he applies to the development of the reusable assets. This domain analysis determines the feasibility of producing a reusable asset. In the component specification phase, the specification of the required functions, interfaces, and performance for the particular asset are defined, resulting in a requirements specification and a detailed design document. The detailed design document is then used to code the reusable assets during the development phase. These reusable assets are then verified and validated and finally certified before they are stored in the software reuse library. During verification, the software component is analyzed to determine if it was built according to specification. During the validation phase, the component is analyzed to see if the component accomplishes the intended goal.

The consumer uses the requirements he has defined to retrieve the appropriate component(s) from the repository. These retrieved reusable components are then assessed to determine if they can be used as is (instantiated) or modified to fit the specifications (adapted). The black box components, those that were instantiated, and the white box components, those that were adapted, are integrated into a single system.

Component engineering accomplishes the component specification, development for reuse, verification and validation, reuse certification, and storage phases of the software reuse lifecycle. Domain engineering represents the domain analysis and component specification phases of the producer. Application engineering, on the other hand, represents the requirements specification, retrieval, assessment, instantiation, adaptation, and integration phases of the software lifecycle of the consumer. Corporate engineering results from the production of reusable assets in domain engineering and exploitation of reusable assets in application engineering, which in turn embodies all of the phases of the software lifecycle.
Figure 1 Software Reuse Lifecycle
1.3 State of the Art and State of Practice

1.3.1 Software Reuse Initiative

Software engineering with software reuse is only possible if the appropriate software reuse program is set up. According to Schmisky [13], some of the technical considerations required to establish a software reuse program are identifying the domain of interest, performing domain analysis, acquiring the components identified in the domain analysis, and establishing a parts library. In domain analysis, common system characteristics are generalized, common components and processes are identified, and their interrelationships are defined in a generic domain. After this domain analysis the identified components need to be acquired from other established libraries or suppliers or developed from scratch. One problem with this acquisition, as well the establishment of a component library, is providing accurate, thorough cataloging with descriptive entries for each component.

According to the DoD Executive Steering Committee, the steps to achieve reuse success are to [14]:

- Establish domains
- Define reusable products
- Establish criteria for deciding ownership
- Integrate reuse into the development and maintenance process
- Define a model for business decisions
- Define metrics to evaluate reuse success
- Define component guidelines
- Identify technology base
- Provide education and training
- Provide near-term products and services

Successful reuse programs have been implemented at IBM, AT&T, Hewlett-Packard, Toshiba, Motorola, GTE, Fujitsu, Hitachi, Boeing, Ford Aerospace, General Dynamics, McDonnell Douglas, NASA, and Kodak [15].

1.3.2 Software Reuse Barriers

Even though software reuse has been successful in many big corporations, software reuse has many barriers. Margono and Rhoades [16] point out some of the common reuse barriers including the following problems: the part does not exist, is not available, is not found, is not valid, is not understood, cannot be integrated, or there is no attempt at reuse. The latter barriers can be caused by poor management, lack of education, lack of economic incentive, poor search tools, poor testing, poor documentation, etc. Furthermore, reuse can have hidden costs. Some of these hidden costs consist of maintenance and frustration in satisfying multiple customers.
Software engineering experts routinely recognize that software reuse is driven primarily by economic considerations. “Software managers feel the pressure to simultaneously improve time-to-market (TTM), software quality and staff productivity. They need to reduce costs of both development and maintenance. At the same time, they need to maintain or increase the ability to respond effectively and rapidly to the changes in markets, requirements, or business cycles [15].”

Software designed for reuse requires 20-25% more time to develop and to learn its use on initial start-up. Tracz [5] claims the cost of making software reusable is 60% more than developing it for single use due to the additional generalization, documentation, testing and library support and maintenance. Furthermore, the cost to reuse software is about five lines of reused code to one line of new code due to the cost of finding it, understanding how to use it, etc. [5] Favaro notes that the cost of creating a reusable code component is about twice that of creating a non-reusable version, and costs to integrate reused components into new products ranged from 10-20% of the cost of creating non-reusable versions. He also comments that the relative cost of producing a reusable component ranged from 120-140% of the cost of creating a non-reusable version, and integration costs ranged from 10-63% of the cost of creating a non-reusable version [17].

Other factors that are usually cited as obstacles to software reuse include [5]:
- Lack of components to reuse
- Cost to make reusable components
- Lack of quality of components
- Inability to locate components to reuse
- Lack of understanding on how to use the component
- Difficulty in modifying the component
- Inefficient generalization of components
- Difficulty in integrating/composing components
- Violation of property rights

Some of these inhibitors can be overcome. For example, the lack of components to reuse can be overcome by domain analysis and domain-specific components, the basis of the domain engineering cycle. Domain analysis identifies reusable resources, provides not only search and classification context, but also provides a design rationale and a context for understanding the software.

Other inhibitors of reuse are people, processes, management, economics, scope, domain, standards, and technology. A management that is unwilling to underwrite the up-front costs for reuse or is unwilling to encourage reuse and provide incentives to developers creates a barrier for a successful reuse program. Furthermore, if the developers/programmers have a bias against reuse, the problem is compounded. Other barriers to reuse are economic (system costing, component pricing, support costs, metrics), legal (copyright/liability), and technical (components/library repositories, classification, query/search policies, tool for generating, integrating, building components). Some work products are just not reusable. However, those products that
are reusable will lead to standardization of documentation, testing, specifications, programming, and processes which require more time and money [15].

1.4 Conclusion

Despite the many disadvantages associated with reuse, it has proven to be not only a successful but also a beneficial financial venture in the long term. In other engineering disciplines, reuse is an integral part of the design and not a subject that requires special consideration. Thus, software engineering needs to make software reuse an integral part of design rather than a subject of study and debate.
Chapter 2. Software Engineering Economics
Software reuse needs to be an integral part of software engineering. However, as noted in the previous chapter, software reuse has its costs and risks. To understand the economics of software reuse, one must first understand the economics associated with the software engineering process, which this chapter discusses.

2.1 Software Engineering Lifecycle

A software lifecycle describes the phases and activities of a software process. The most common lifecycle model in software is the waterfall model, which models the process as a set of chronological phases. Orthogonally, each phase is divided into a set of concurrent activities. The most common waterfall model is Boehm’s model, which provides eight phases and eight activities.

The eight phases of Boehm’s model [18] are:
1. System Feasibility
2. Software Plans and Requirements
3. Product Design
4. Detailed Design
5. Coding
6. Integration
7. Implementation
8. Operations and Maintenance

In the system feasibility phase the economic and technical feasibility of building the product is assessed and alternatives are analyzed. In the requirements phase, the functional, interface and performance specifications are elicited and captured. In the product design phase, these requirements are drafted into user manuals and test plans and are used to specify the hardware and software architecture as well as the control and data structures of the product. In the detailed design, the details of the control structures and software and hardware architecture including the algorithms, assumptions, data structures, and interface relations are detailed. During the coding phase the code is then generated based on this detailed design. All developed code components are then integrated during the integration phase. The final product and its associated installation, training, and conversion utilities are delivered to the users during the implementation phase. During the operations and maintenance phase updates to the system are tasked and completed. In all of the eight phases there is ongoing validation and verification. In verification the “truth of the correspondence between the software product and its specification” (building the product right) is established while in validation the “fitness or worth of the software product for its operational mission” (building the right product) is determined [18].
The eight activities of Boehm’s model [18] are:

1. **Requirements Analysis**
   “Determination, specification, review, and update of the software’s functional, performance, interface, and verification requirements”

2. **Software Design**
   “Determination, specification, review and update of hardware-software architecture, program design, and database design”

3. **Programming**
   “Detailed design, code, unit test, and integration of individual computer program components. Includes personnel planning, tool acquisition, database development, component level documentation, and intermediate level programming management.”

4. **Test Planning**
   “Specification, review, and update of product test and acceptance test plans. Acquisition of associated test drivers, test tools, and test data.”

5. **Verification and Validation**
   “Performance and independent requirements validation, design V&V, product test, and acceptance test. Acquisition of requirements and design V&V tools.”

6. **Project Office Functions**
   “Project level management functions. Includes project level planning and control, contract and subcontract management, and customer interface.”

7. **Configuration Management and Quality Assurance (CM/QA)**
   “Configuration management includes product identification, change control, status accounting, operation of support library, development and monitoring of end item acceptance plan. Quality assurance includes development and monitoring of project standards, and technical audits of software products and processes.”

8. **Manuals**
   “Development and update of users’ manuals, operators’ manuals, and maintenance manuals.”

These activities and phases are organized into a two dimensional structure shown in Table 2 [18].

The cost model that is discussed in the sequel applies to software projects that follow the lifecycle model of this section. Also, all the effort equations of this cost model cover a specific set of lifecycle phases, namely, system feasibility, product design, detailed design, coding, integration, implementation, and operations and maintenance.
<table>
<thead>
<tr>
<th>Phase</th>
<th>Activity</th>
<th>Plans and Requirements</th>
<th>Product Design</th>
<th>Programming</th>
<th>Integration and Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Requirements Analysis</td>
<td>Analyze existing system, determine user needs, integrate, document, and iterate requirements</td>
<td>Update Requirements</td>
<td>Update Requirements</td>
<td>Update Requirements</td>
</tr>
<tr>
<td></td>
<td>Product Design</td>
<td>Develop basic architecture, models, prototypes; risk analysis</td>
<td>Develop product design; models prototypes, risk analysis</td>
<td>Update design</td>
<td>Update design</td>
</tr>
<tr>
<td></td>
<td>Programming</td>
<td>Top-level personnel and tools planning</td>
<td>Personnel planning; tools, utilities acquisition</td>
<td>Detailed design, code and unit test, component documentation, integration planning</td>
<td>Integrate software, update components</td>
</tr>
<tr>
<td></td>
<td>Test Planning</td>
<td>Acceptance test requirements, top-level test plans</td>
<td>Draft test plans; acquire test tools</td>
<td>Detailed test plans, acquire test tools</td>
<td>Detailed test plans, install test tools</td>
</tr>
<tr>
<td></td>
<td>Verification and Validation</td>
<td>Validate requirements, acquire requirements, design V&amp;V tools</td>
<td>V&amp;V product design, acquire design V&amp;V tools</td>
<td>V&amp;V top portions of code, V&amp;V design changes</td>
<td>Perform product test, acceptance test, V&amp;V design changes</td>
</tr>
<tr>
<td></td>
<td>Project Office Functions</td>
<td>Project level management, project MIS planning, contracts, liaison, etc.</td>
<td>Project level management, project MIS planning, contracts, liaison, etc.</td>
<td>Project level management, project MIS planning, contracts, liaison, etc.</td>
<td>Project level management, project MIS planning, contracts, liaison, etc.</td>
</tr>
<tr>
<td></td>
<td>CM/QA</td>
<td>CM/QA plans, procedures, acceptance plan, identify CM/QA tools</td>
<td>CM/QA of requirements design; project standards, acquire CM/QA tools</td>
<td>CM/QA of requirements design; code, operate library</td>
<td>CM/QA of requirements design; code, operate library; monitor acceptance plan</td>
</tr>
<tr>
<td></td>
<td>Manuals</td>
<td>Outline portions of users’ manuals</td>
<td>Draft users’, operators’ manuals, outline maintenance manual</td>
<td>Full draft users’ and operators’ manuals</td>
<td>Final users’, operators’, and maintenance manuals</td>
</tr>
</tbody>
</table>

Table 2 Activities and Phases of the Software Lifecycle
2.2 Software Cost Models

Several models have been proposed in the past to estimate the development cost and development schedule of a software product. Furthermore, many of these cost models provide not only overall project estimates but also effort and schedule distribution by phase and activity. Typically, development effort is measured in person-months (E) and the development schedule is measured in months. However, some models depart from these conventions. Software costs are driven by a wide variety of cost factors; the most commonly used cost function is product size, which is typically measured in lines of code (LOC).

Cost estimation models define effort/cost of software engineering in terms of effort (E= man-months) and/or lines of code (KLOC= Lines of Code/1000). A line of code can be defined as a source or machine instruction, comment, and/or blank. Effort can be defined as the amount of time/money it takes to produce a line of code. Poulin finds lines of code are used to quantify the effort in software development despite its deficiencies as a unit of measure because they are simple to understand, easy to collect and compare and difficult to distort [9]. Other authors [18-21] object to the use of LOC as a measure of size/functionality/service because LOC is programming-language-dependent, it penalizes well-designed shorter programs, cannot accommodate nonprocedural languages, and is difficult to estimate.

Software costs estimation is difficult due to conflicting project goals, lack of detailed requirements, difficulty in estimating effort and continuous emergence of new development processes, methods, and tools. It is difficult to identify and quantify the factors needed to develop, locate, understand, modify, integrate and test code needed in a project.

Barry Boehm [18] discusses several software engineering cost techniques for estimating software costs including:
1. Algorithmic Models
2. Rules of Thumb
3. Expert Judgment
4. Estimation by Analogy
5. Design to Cost
6. Price-to-win Estimating
7. Top-down Estimating
8. Bottom-up Estimating

Discussed in this chapter will be algorithmic techniques and function point analysis techniques to quantify the software engineering lifecycle in terms of lines of code and/or functionality provided by the software product.
\subsection*{2.2.1 Algorithmic Models}

Algorithmic models suffer from unreliable software cost estimates due to the following reasons:
1. Lack of historical data from past projects
2. Lack of past history on similar projects (First-of-a-kind system)
3. Lack of expertise by the estimator
4. Premature estimation
5. Failure to update estimates with project and environment changes
6. Late discovery of missing pieces in the estimation process [22].

Furthermore, due to the subjective definition of the units of output/input (effort/size of the asset) and due to use of averages to validate them since real project data does not exist, the algorithmic models to be discussed in the subsequent sections have limited validity.

\subsubsection*{2.2.1.1 Wolverton}

Wolverton’s model [23] is a linear model in the size of the proposed product whereby:

\[ M_k = S_i C_{ij} \]

where
- \( M_k \)=Cost to develop module k
- \( C \)=Cost matrix
- \( i \)=Module type
- \( j \)=Complexity
- \( S_k \)=Size

The problem with this model is that the assessment of complexity is subjective, programming experience and hardware characteristics are not taken into account, and it is difficult for the user to assess the costs, especially integration costs.

\subsubsection*{2.2.1.2 Walson-Felix}

In the Walson-Felix model [23],

\[ E = 5.2\text{KLOC}^{0.91} \]

where
- \( E \)=Programming effort
- KLOC=Size of the product in thousands of lines of code
In this software model the primary cost driver is software size that is corrected in the productivity index by 29 cost drivers. This equation was based on 60 IBM projects where the average productivity for a low complexity project is 500 lines of code per man-month.

### 2.2.1.3 COCOMO

COCOMO, the Constructive Cost Model [3,18,21-23], calculates the cost of developing software using the waterfall model. It takes the estimated size of the software project in lines of code and computes the software development effort and time. The equation used for the three classes of projects (organic, embedded, and semi-detached) in basic COCOMO to calculate effort is:

\[ E = bKLOC^e \]

where

- \( E \) = Effort
- \( KLOC \) = Number of thousands of delivered source instructions.

<table>
<thead>
<tr>
<th>Class</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>2.4</td>
<td>1.05</td>
</tr>
<tr>
<td>Embedded</td>
<td>3.6</td>
<td>1.20</td>
</tr>
<tr>
<td>Semi-Detached</td>
<td>3.0</td>
<td>1.12</td>
</tr>
</tbody>
</table>

A delivered source instruction, the primary cost driver of this model, is any line of source text irrespective of the actual number of instructions/operations on that line. However comments are excluded. The resulting effort is in person-months which Boehm defines as 152 hours (19 days) of working time (taking into account training, vacation and sick time).

This model also makes many assumptions:

1. The development period starts at the software design phase and ends at the end of the integration and test phase. Activities like feasibility studies, installation plans, conversion plans, training, … are not included in this model. [22]
2. Only directly charged labor is accounted for; labor for operators, secretaries, and others are not accounted for. [22]
3. The requirement’s specifications do not change significantly after the requirements phase. This model applies to projects over 2000 delivered source instructions otherwise the personnel will dominate the calculation. [22]

In the organic class, a small team with much experience develops software in a known environment. This class incurs little initial overhead. In the embedded class, the
environment is inflexible with many constraints. The semi-detached class has both experienced and inexperienced programmers working on a large project.

Boehm’s basic COCOMO model is expanded to yield the Intermediate and Detailed COCOMO which takes into account fifteen cost drivers that affect productivity from a product, computer, personnel, and project perspective. They are:

- **Product**
  - Required software reliability
  - Database size
  - Software complexity
- **Computer**
  - Execution time constraint
  - Main storage constraint
  - Virtual machine volatility
  - Computer turnaround time
- **Personnel**
  - Analyst capability
  - Applications experience
  - Programmer capability
  - Virtual machine experience
  - Programming language experience
- **Project**
  - Use of modern programming practices
  - Use of software tools
  - Schedule constraints

Intermediate COCOMO multiplies the nominal effort equation by an effort adjustment factor, which is the product of the fifteen adjustment factors (AAF’s). Detailed COCOMO unlike Intermediate COCOMO is sensitive to the software engineering phase (analysis, design, etc.) and is based on expert judgment, analysis of available project data, models, etc., but is seldom used.

All models of COCOMO consider the software to be one product and not the integration/combination of many heterogeneous systems. Thus, the above multipliers are applied to the system as a whole, instead of to the individual subsystems.

COCOMO estimates the cost of developing software using the waterfall lifecycle model. The problem with this model is that it assumes no reuse and few changes in the requirements after they are identified. Thus, this model does not reflect the assumptions of reuse. [12]
2.2.1.4 Putnam/Norden

Putnam and Norden [23] subscribe to the premise that the distribution of manpower approximates the Rayleigh distribution:

\[
MR(t) = 2Kae^{-at^2}
\]

where
- MR = Manpower at time t
- a = Speed up factor (slope of the curve)
- K = Total manpower including maintenance phase
- t = Time

Results of this equation are close to the rule of thumb for the time spent in the different software engineering cycles [23]:
- 40% development
- 60% maintenance

However this model is suitable only for very large development projects (15 man-years or longer) and does not take into account past experience of the personnel.

2.2.2 Function Point Analysis

Function Point Analysis was developed to estimate software costs by analyzing software requirements and is based on the number of functional primitives (FP) rather than expected lines of code (which are not available at requirements specification time). DeMarco’s cost estimation model [23] is based on the number of these functional primitives and dataflow diagrams that model the system to be developed. These functional primitives are the primitives at the lowest level of the data flow diagrams of the requirements/specification stage. Function point analysis, which was developed by Albrecht [23], takes DeMarco’s concept of functional primitives one step further and defines the unadjusted function points (UFP):

\[
UFP=4I+5O+4E+10L+7F
\]

where the entities are defined as
- I = Input Types
- O = Output Types
- E = Inquiry Types
- L = Logical Internal Files (indexes)
- F = Interfaces

and the multipliers (4,5,4,7,10) are the medium complexity weights.
The technical complexity factor (TCF), the number of characteristics that affect the development effort, is defined as:

\[
TCF = 0.65 + 0.01\text{DI}
\]

where

\[
\text{DI} = \text{Total Degree of Influence}
\]

Factors that can impact the Total Degree of Influence include:

- Data communications
- Performance
- Distributed functions
- Transactional rate
- Online data entry
- Online update
- Complex processing
- Ease of installation
- Multiple sites
- Ease of change
- Reusability of the system code
- Design for end-user efficiency
- Heavily used operation configuration [23]

and are rated on a scale of zero to five where zero is no influence and five is a strong influence. The unadjusted function points and technical complexity factor equations are combined to define a function point as a measure of “software functionality based on the counted or estimated number of ‘externals’ (inputs, outputs, inquiries, and interfaces) of a system, plus the estimated number of internal files [23]:

\[
FP = (UFP)(TCF)
\]

This technique to evaluate the size of a system encompasses the following problems:

- Difficulty in distinguishing between the different interface types
- Difficulty in determining the difference between input, inquiry, and interface types
- Recognition that only externally visible I/O is counted
- Calibration to the environment of the entities
- Complexity of counting twice: UFP and TCF
- Lack of uniform way to count function points

Casper Jones [24] of the Software Productivity Consortium estimates that one function point is equivalent to 100 logical C source code statements. Examples of function point to source code statement equivalences are illustrated in Table 3.
<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Lines of Code per Function Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler</td>
<td>300</td>
</tr>
<tr>
<td>COBOL</td>
<td>100</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>100</td>
</tr>
<tr>
<td>PASCAL</td>
<td>90</td>
</tr>
<tr>
<td>Ada</td>
<td>70</td>
</tr>
<tr>
<td>Object Oriented Languages</td>
<td>30</td>
</tr>
<tr>
<td>Fourth Generation Languages</td>
<td>20</td>
</tr>
<tr>
<td>Program Generators</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 3 Function Point to Lines of Code Equivalences [24a]

2.3 Development Time

A discussion of software cost estimation would not be complete without discussing development time. Development time (T), otherwise known as the development schedule, has been approximated by Walston-Felix, Boehm, and Putnam. All three models are similar:

\[
T = 2.5E^{0.35} \\
T = 2.5E^{0.38} \\
T = 2.4E^{0.33}
\]

All three models reflect the fact that if the development time is shortened, the speed up factor of the curve is increased. However, to accommodate this shortened development time, manpower increases, and productivity decreases due to communication overhead and training of new team members. Brook’s Law summarizes this concept well: Adding manpower to a late project will only make it later:

\[
L = 777p^{-0.5}
\]

where

- \( L \) = Lines of Code
- \( p \) = Team size

Furthermore, the shorter the development, the higher the costs, typically [23].
2.4 Conclusion

Both the algorithmic approach and function point analysis approach to estimating software engineering costs suffer from similar problems: lack of data, differences in the definition of entities/lines of code, differences in environment, subjective assignment of values to cost factors, and basis on old projects and technology. Furthermore, these approaches account mostly for development and do not account for maintenance, documentation and other related costs, and do not reflect the dynamic nature of cost estimation, i.e., early time-to-market and change in market value over time. Despite the inadequacies of the algorithmic approach, the algorithmic approach possesses considerably fewer problems than function point analysis. Furthermore, COCOMO is considered to be the most comprehensive algorithmic model for determining development costs without reuse. For this reason, COCOMO is used throughout this thesis to determine development costs from scratch.
Chapter 3. Investment Analysis
3.1 An Investment Cycle

An investment cycle is a process whereby one consents to invest some up-front resources in the hope of recovering within some predefined amount of time, the initial investment, as well as a substantial proportion of the benefits. The main factors that characterize an investment are

- Investment Costs (IC) measured in person months
- Investment Cycle (Y) measured in number of years and ranging typically 3 to 5 and incremented from a start date (SD)
- Discount Rate (d) is the time value of money and typically ranges from .10 to .20
- Episodic Benefits (B(y)) measured in person-months at year y where SD+1≤y≤SD+Y
- Episodic Costs (C(y)) measured in person-months at year y where SD+1≤y≤SD+Y

The following assumptions are also made
1. C(SD)=IC
2. B(SD)=0
3. ∀y<SD, B(y)=0 and C(y)=0

In order to determine whether an investment is worthwhile, one must analyze all these factors. At the time when one must make a go/no-go decision, some of these factors are known and some (B(y), C(y), y, Y) are not known. Hence, one must choose methods that allow one to estimate the unknown quantities. Furthermore, one must make provisions for risks involved in the decision.

The investment decisions can be quantified by means of an investment analysis function. The following functions have been identified in literature [25-29] in the past as capital budgeting techniques: Net Present Value (NPV), Payback Period (PB), Average Return on Book Value (ARBV), Internal Rate of Return (IRR), Return on Investment (ROI), and Profitability Index (PI). These capital budgeting techniques reflect the economic worth of a project.

3.2 Present Value

The concept of present value is the key to investment analysis decisions. Present value is based on the notion that a dollar today is worth more than a dollar tomorrow (time value of money). Present value is based on the discounted cash flow formula (DCF) which weights the relative contributions of cash flows in the future by a discount rate d according to the period (year y) in which the cash flow occurs CF;

\[
P V = \frac{CF_1}{(1 + d)} + \frac{CF_2}{(1 + d)^2} + \ldots + \frac{CF_y}{(1 + d)^y} = \sum_{y=1}^{Y} \frac{CF_y}{(1 + d)^y} = \sum_{y=1}^{Y} \frac{B(y) - C(y)}{(1 + d)^y}
\]
DCF is the best method to use for long-range decisions because it weighs the time value of money and is based on cash inflows/outflows instead of net income. The discount rate (also known as opportunity cost and cost of capital) represents the opportunity cost of the project or the rate of return expected from an equivalent investment on the capital markets. It is normally different in different periods (short versus long term rates) but is typically a single rate. DCF is a technique to calculate NPV, the net totality of all contributions to the value of the investment, payback, and IRR. It assumes that the predicted cash flows will occur at specified times (certainty) and that the original amount of the investment can be considered borrowed or loaned at some specified rate of return (interest).

3.3 Net Present Value

3.3.1 Definition

Net present value represents the expected economic contribution of a project [25]. It is defined as the present value of the cash inflows minus the present value of the cash outflows (typically the initial investment). NPV is the expected economic contribution of a project that is additive and allows evaluation of combinations of projects while taking into account differences of scale [25]. These cash flows are the present value discounted by the required rate of return [27] where present value is what the value of $x$ now is to be in $n$ periods at an interest rate of $d$.

\[
\begin{align*}
\text{NPV} &= CF_0 + PV \\
&= -IC + \sum_{y=1}^{Y} \frac{B(SD + y) + C(SD + y)}{(1 + d)^y} \\
&= \sum_{y=0}^{Y} \frac{B(SD + y) + C(SD + y)}{(1 + d)^y}
\end{align*}
\]

3.3.2 Advantages

NPV takes into account the differences of scale, time value of money, magnitudes of the projected cash flows, and the riskiness of the project. Furthermore, it is additive allowing the evaluation of combinations of projects. “The Net Present Value (NPV) method is a very useful tool for evaluating capital investments. We believe it is the best method because it gives clear information on the size of the impact of an investment on firm value and it is the easiest to use.”[27]
3.3.3 Disadvantages

NPV does not really measure the benefits relative to the initial investment due to project risks and estimation errors. NPV represents a hypothetical value not a certified value. A hypothetical value is worth the risk only if it is large relative to the investment.

3.3.4 Evaluation Criteria

An investment is beneficial if the NPV is greater than or equal to zero because earnings will be greater than or equal to the required return.

3.4 Average Return on Book Value/Average Rate of Return

3.4.1 Definition

Average return on book value (ARBV) is the projected accounting net income over the life of the project, divided by the average book value of the investment [29] or the ratio of the average annual profits to the average investment in the project [25]. It is also known as return on assets (ROA) and accounting rate of return or average rate of return (ARR). Typically its calculation is based on accounting data (profits after taxes) [26].

\[
\text{ARBV} = \frac{O - D}{\text{ARBVI}}
\]

\[
\text{NPV} = \frac{Y}{\frac{IC}{2}}
\]

where  
O = the average net income  
D = the incremental average annual depreciation  
ARBVI = the average initial incremental amount invested.  
NPV=Net present value  
IC=Initial cost  
Y=Duration of the investment

This equation assumes that in the determination of the average projected book value of the investment over the life of the project, Y, the asset value declines (depreciates) at a uniform rate from its initial investment to 0. Hence, the average projected book value is the average investment which can be found by dividing the initial investment by 2. For example, assume the initial investment was $40K and the average book value for years 1-
4 of this asset was $35K, $25K, $15K and $5K. The average book value would be $80K/4 or $40K/2 or $20K.

### 3.4.2 Advantages

ARBV is a simple measure that takes advantage of available accounting information [26,28].

### 3.4.3 Disadvantages

The target rate of return is arbitrary and dependent on the amortization schedule that is only applied to the capital investment and not to the operating expenses. Furthermore, this function relies on the net income, not the cash flows, ignores the project riskiness, and ignores the time value of money. [27,29]. According to Garstka, “It is an arbitrary rule without economic basis and can lead to incorrect capital investment decisions” [29].

### 3.4.4 Evaluation Criteria

Investment occurs if the value meets the predefined required, or cutoff, rate of return, e.g., 20%.

### 3.5 Payback Period

#### 3.5.1 Definition

Payback Value (also know as the breakeven point) is the number of years required to recover the initial cash investment and measures how quickly investment dollars may be recouped. This ratio does not measure profitability. It allows investors to compare the profit consequences of alternative selling prices of an asset. It is defined as the ratio of the initial fixed investment over annual cash inflows for a recovery period [25] or the point where the total revenue equals the total cost [30]. It is defined as

\[ PB = \frac{I}{O} \]

where

- \( I \) = the initial investment
- \( O \) = cumulative projected cash inflows [28-29]

or as fixed expenses/(Selling price per unit – Variable expenses per unit) [27]. In terms of costs and benefits, it is the smallest value of Y that satisfies the following:

\[ PB = \sum_{y=0}^{Y} B(SD + y) - C(SD + y) \geq 0 \]
Note the latter does not take in account the time value of money. To take into account the time value of money, discounted payback is defined as:

\[
P_B = \sum_{y=0}^{\infty} \frac{B(SD + y) - C(SD + y)}{(1 + d)^y} \geq 0
\]

3.5.2 Advantages

The payback period considers cash flows and has some consideration for the time value of money. Furthermore, it is a measure of the risk and liquidity of a project. The shorter the payback period, the less risky the project. It is also a better measure than average return on book value because it considers cash flows rather than accounting profits [26].

3.5.3 Disadvantages

This ratio is arbitrary and subjective and not sensitive to scale, ignoring depreciation. It also makes a linear assumption assuming the selling price is the same no matter how many units are sold. It also does not take into account the magnitude and timing of the cash flows. Furthermore, this approach only implicitly considers the time value of money and it fails to recognize cash flows after the payback period and therefore cannot be considered a measure of profitability.

3.5.4 Evaluation Criteria

Investment is recommended if this value is smaller than the proposed investment cycle period. “When the payback method is used, it is more appropriately treated as a constraint to be satisfied than as a profitability measure to be maximized” [25].

3.6 Profitability Index

3.6.1 Definition

Profitability Index judges how well a company did in controlling expenses and earning a return on resources. This benefit-cost ratio is the ratio of the present value of the cash inflows to the present value of the cash outflows [27] or the present value of the future net cash flows over the initial cost [25].
3.6.2 Advantages

Since PI is derived from NPV, if NPV > 0 then PI > 1, if NPV < 0 then PI < 1 and if NPV=0 then PI=1.

3.6.3 Disadvantages

This ratio is not additive and not sensitive to projects of different scale. Furthermore, PI is not as reliable as NPV since a higher PI will not necessarily increase the value of one investment over another.

3.6.4 Evaluation Criteria

Investment is beneficial if PI is greater than 1, in which case the rate of return is greater than the required rate and the net present value is greater than or equal to 1.

3.7 Internal Rate of Return

3.7.1 Definition

Internal Rate of Return or yield criterion is “the maximum rate of return that can be paid for the capital employed over the life of the investment without loss on the project” [28]. It is also defined as “the rate of discount that equates the present value of the cash flows with the initial investment associated with the project” [26], in other words, the discount rate that makes NPV equal to zero.

IRR is the value of d that solves the following equation:

$$NPV = \sum_{y=0}^{\infty} \frac{B(y) - C(y)}{(1 + d)^y} = 0$$
3.7.2 Advantages

When NPV is zero, IRR is equal to the required rate. Furthermore, when NPV is greater than zero, then IRR is greater than the required rate and an increase in the market price should result. It can be used to judge whether an investment is expected to increase in value [27]. Many financial managers feel that the internal rate of return is easier to visualize and to interpret than is the net present value because business managers relate better to rates of return than dollar returns [26,27]

3.7.3 Disadvantages

IRR is computed by trial-and-error interpolation of the present value tables and is more difficult to calculate than NPV and PI. This function cannot model multiple discount rates, but is modeled from forecasted cash flows, and has a problem with scale of projects. Furthermore, this approach assumes reinvestment at the IRR, unlike NPV, that assumes that all immediate cash flows are reinvested. Lastly, if the cash flows change signs over the life of the investment (ignoring year 0), the equation may not have a unique solution.

3.7.4 Evaluation Criteria

Investment is beneficial if the IRR is greater than or equal to the opportunity cost of the project (minimum desired rate of return, cutoff rate, hurdle rate, target rate, discount rate, or cost of capital).

3.8 Return on Investment

3.8.1 Definition

The Rate of Return on Investment (ROI) is a measure of profitability (how to maximize income given the same risks and given the same amount of resources). This rate of return on assets is calculated as the rate of profit or income percentage of revenue (net profit/sales) times the capital turnover rate (sales/investment), which is also known as the DuPont formula [26,30]. The DuPont formula breaks down the ROI into a profit-on-sales (net profit/sales) and asset efficiency (sales/investment) components. If there is a low net profit margin, then there is typically a high total asset turnover. This ratio is defined as the profits after taxes divided by the total assets:
3.8.2 Advantages

Return on Investment is the ratio of the savings to the investments and is a measure of profitability [28].

3.8.3 Disadvantages

Historical ROI measurements can result in management adopting a short-term ROI by not replacing older assets with newer, more productive assets that increase ROI in the future but penalize short-term ROI [29].

3.8.4 Evaluation Criteria

ROI is “often called the firm’s return on total assets measures, the overall effectiveness of management in generating profits with the available assets. The higher the firm’s return on investment, the better” [26].

3.9 Conclusion

ARBV and Payback value are considered unsophisticated techniques to determine the acceptability of a capital expenditure because unlike NPV, IRR and PI, they do not consider the time value of money [26]. Furthermore, NPV is theoretically a better technique than IRR and ROI, but IRR is more popular because managers can relate better to rates of return than to dollar returns. Nevertheless, for the sake of completeness, all of the functions will be included in the integrated cost model.
Part II. An Integrated Approach
Chapter 4. An Integrated Cost Model
4.1 Software Engineering Reuse Decisions

Software engineering reuse investment decisions need to be made on different levels of the company from the corporate level to the programming level. At the corporate level, managers determine if a reuse initiative will be worthwhile to the corporation and if there will be any benefit to such an initiative. At the managerial level, domain engineering managers create reuse development procedures, policies, and incentives while application engineering managers analyze the worthiness of incorporating reuse into a project. At the programmer level, programmers must be trained to take advantage of reuse and use these skills to develop “reusable” code. All of these decisions enable a good reuse program to be implemented but this policy has its costs. Traditional software engineering economics do not take into account the long-term benefits of reuse, the overhead costs associated with reuse, and the maintenance savings due to quality and productivity gains. Furthermore, traditional investment analysis is focussed on the cash inflows and outflows of a business. Since software development and product development is so unlike traditional manufacturing, these investment analysis functions need to be adapted to accommodate this type of product. Presently, over twenty different cost models exist that can be used to evaluate these decisions. The details of these existing models will be presented in Chapter 5. In this chapter, a generalized cost model is presented that encompasses the characteristics of these existing models and incorporates enhancements such as the quantification of quality and productivity gains and shorted time-to-market.

4.2 Investment Decisions

These different decision levels are reflected in the different investment cycles [1]: the corporate investment cycle, domain engineering investment cycle, application engineering investment cycle, and component engineering investment cycle (Figure 2). Each of these investment cycles is characterized by the following cost factors:

- Investment costs, IC
- Periodic costs at year \( y \), \( C(y) \)
- Periodic benefits at year \( y \), \( B(y) \)
- Discount rate, \( d \)
- Duration, \( Y \)

The investment cycle, \( Y \), is measured in years and ranges from 3 to 5 years. The discount rate (the interest rate), \( d \), ranges from .10 to .20, is dimensionless, and reflects the time value of money. The investment costs, IC, are measured in person-months and indicate the startup costs. The periodic benefits, \( B(y) \), are the benefits gained by creating, using, and classifying a reusable asset and are measured in person-months. The periodic costs, \( C(y) \), are the costs associated with creating, retrieving and using a reusable asset and measured in person-months.
The latter cost factors that define the investment cycle are used to quantify the various investment decisions. Based on these cost factors, the economic functions (NPV, PB, PI, ARBV, ROI, and IRR) described in Chapter 3 are computed to enable reuse investment decisions to be made.

4.2.1 Corporate Engineering Cycle

In the corporate engineering investment cycle, the corporate management must decide whether to incorporate a reuse initiative into the corporation. At this level, management must make a commitment to up-front costs such as the development of a library of reusable assets (infrastructure), training, the change in operating procedures to accommodate, for example, the use of the reuse library (operational impact), and the creation of a reuse department (management restructuring). The latter includes hiring a reuse manager who is a software manager fluent in software reuse organizations, software reuse economics, software reuse metrics and software reuse products. With the commitment of these initial costs plus the periodic costs of domain engineering (domain analysis costs, development of assets for reuse, and library overhead), corporate management hopes to reap the benefits of application engineering, i.e., better quality products, higher productivity, and shorter time-to-market.

4.2.2 Domain Engineering Cycle

In the domain engineering cycle, the departmental manager decides whether to commit to a domain analysis initiative. By committing to this initiative, the manager must absorb the initial cost of the domain analysis and the period costs of the development of the reusable assets (and associated overhead of library storage of these reusable assets). This manager commits these resources in hopes of being able to sell these assets to project teams. The domain analysis is done by a domain engineer who decides whether a generative development paradigm (design patterns) or a compositional development paradigm (source code functions/modules) is appropriate for the domain being analyzed and whether the domain is worthy of future reuse. This person also analyzes an application domain to identify commonly recurring components or patterns by synthesizing, compiling, packaging and archiving past software experiences/knowledge in this particular domain. From this compiled knowledge, reusable assets can be developed, compiled, and stored in the reuse library by component developers in the component engineering cycle for future retrieval by application engineers during the application engineering cycle.

4.2.3 Component Engineering Cycle

In the component engineering cycle, after a decision was made by the domain engineer about whether the asset is worthwhile to develop, (whether it contains a lot of commonality), developers commit resources to develop this reusable asset. Thus, the initial cost of this cycle consists of the development for reuse and development of storage procedures (i.e., cataloging of these assets). In addition to these initial costs, the producer
(developer) absorbs the periodic costs associated with storing and retrieving the asset in the hope of the benefit of selling the assets to project teams of the application engineering cycle. The latter includes the cost of the librarian who is responsible for the cataloging and retrieval mechanisms associated with the storage of the reusable assets as well as ensuring the software generality, correctness, reliability and clarity.

4.2.4 Application Engineering Cycle

The application engineering cycle consists of applying software assets to specific projects. In this cycle the application engineer decides whether to use the retrieved reusable asset verbatim (black box selection) or to use the retrieved reusable asset after modification or adaptation (white box selection). If the modification of the asset involves operations like changing a constant value or adding values to initialize the procedure/function, it is considered to be instantiation instead of adaptation. Furthermore, if the component used is modified, it must be verified to ensure its correctness since previous verification is voided by the modification. This modification or reengineering of assets poses the problems of losing program quality and programmer productivity. To modify the component, the component must be analyzed and understood, an activity which consumes additional programming resources. Thus, project teams must commit resources to accommodate the up-front costs of training, the change in processes/procedures and the associated impact, the introduction of reuse, and the risk (relevance of the asset to the project). In addition to these initial costs, the periodic costs of poor precision (retrieving non-relevant assets) and poor recall (failing to retrieve relevant assets) contribute to the total costs of this cycle. These costs are committed to by this consumer in hope of periodic benefits of gains in quality, productivity and time-to-market.
**Corporate Cycle**

*Investment Costs*

Reuse Infrastructure

*Episodic Costs*

---

**Domain Engineering Cycle**

*Investment Costs*

Domain Analysis Cost

*Episodic Costs*

<table>
<thead>
<tr>
<th>Component Engineering Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Investment Costs</em></td>
</tr>
<tr>
<td>Develop For Reuse</td>
</tr>
</tbody>
</table>

*Episodic Benefits*

Cost Transfers from Projects

---

*Episodic Benefits*

---

**Application Engineering Cycle**

<table>
<thead>
<tr>
<th>Investment Costs</th>
<th>Episodic Costs</th>
<th>Episodic Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>Reuse Overhead</td>
<td>Productivity Gains</td>
</tr>
<tr>
<td>Process Impact</td>
<td>Reuse Risks</td>
<td>Quality Gains</td>
</tr>
</tbody>
</table>

---

**Figure 2 Integrated Software Reuse Cost Model**
4.3 Cost Structure

The integrated cost model is a generalized cost model that is based on these investment cycles and the associated cascading costs and benefits. In this model, the component engineering cycle is the domain engineering cycle costs and the domain engineering cycle corporate engineering cycle costs, while the application engineering cycle is the corporate engineering cycle benefits.

4.3.1 Cascading Costs and Benefits

These benefits and costs are propagated in a cascading pattern from one decision cycle to another (Figure 3). The basis of the costs and benefits of all the cycles is the component engineering cycle. The costs and benefits of components developed for domain engineering comprise the periodic costs of the domain engineering cycle. These costs plus the initial costs of domain analysis during the domain engineering cycle propagate to the corporate engineering cycle. Thus, the component engineering cycle is a subset of the domain engineering cycle which is a subset of the corporate engineering cycle. By minimizing the costs of component engineering and domain engineering, the costs of the corporate engineering cycle can therefore be minimized. The benefits of application engineering follow a similar pattern.

The costs and benefits of component flow to the application engineering cycle through the sale of the reusable assets to project teams. The benefit from the sale of these assets to the project team, plus the quality, productivity, and shortened time-to-market gains of the application engineering cycle from all applications developed, comprise the benefits of the corporate engineering phase. Thus, the component engineering cycle is a subset of the application engineering cycle which is a subset of the corporate engineering cycle. By maximizing the benefit of the component engineering cycle (the sale of the assets to the application project teams), the corporate benefit can be increased. Furthermore, by maximizing the productivity, quality, and shortened time-to-market gains of the application engineering cycle, this benefit can be further increased.

As a result of this flow of costs and benefits from cycle to cycle, investment decisions can be made based on the economic functions (NPV, IRR, PI, PB, ARBV, and ROI) described in Chapter 3, at each level of the investment cycle, to determine if a reuse initiative is worthwhile. To maximize the ROI at the corporate level, one must maximize the ROI at each of the intermediate levels (domain, application, and component). For example, if the ROI at the component level is negative, then the ROI’s at the domain and application level need to be non-negative for the investment to be worthwhile at the corporate level. The latter is not likely unless the component level ROI is made non-negative through the sale of additional assets. Furthermore, if the ROI at the domain or application cycle level are non-negative, the corporate level ROI is likely to be non-negative as a result of this propagation of the costs of the domain engineering cycle and benefits of the application engineering cycle to the corporate cycle.
4.3.2 Integrated Cost Model Assumptions

The integrated cost model makes assumptions in the following areas:

- Buying and Selling of Assets
  The corporate engineering cycle benefits result from the benefits of the application engineering cycle, and the corporate engineering cycle costs result from the costs of the domain engineering cycle. However, this cycle does not include the benefits from the domain engineering cycle and the costs from the application engineering cycle due to the assumption that the selling and purchasing of assets is internal to the corporation. Thus, these benefits from domain engineering are the costs of application engineering (Figure 4). Without this assumption, these costs and benefits from the application and domain engineering cycle would need to be included in the calculation. For this reason, the equations of this model assume internal acquisition and transfer of components. In situations where external assets are acquired, the equations will have to be modified slightly.

- Reuse Organizations

Although this model does not specifically account for the differences/costs associated with various reuse organizations (Lone Producer, Nested Producer, Pool Producer, Team Producer, and Experience Factory), it does make the following assumptions that are related to reuse organizations:
1. A clear separation between the producer team (that develops assets for reuse) and the consumer team (that produces applications from reusable assets).
2. A well-defined pricing structure between the producer team and the consumer team, whereby the producer team gets credit for the amount of code the consumer team reuses.
3. A well-defined pricing structure for assets acquired from external sources, including maintenance fixes.
4. A well-defined reward structure where producer teams get compensated for the volume of reusable assets they contribute and the frequency with which they are reused.
5. A uniform, coherent policy for data collection for all domain, component, and application engineering activities.
6. A clearly defined library management function to manage reusable assets with carefully monitored library insertion procedures and carefully tracked costs.
Figure 3 Cascade of Costs through Investment Cycles
### Component Balance Sheet

<table>
<thead>
<tr>
<th>Year, y</th>
<th>Cost, C(y)</th>
<th>Benefit, B(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>y=SD</td>
<td>Development for Reuse</td>
<td></td>
</tr>
<tr>
<td>y&gt;SD</td>
<td>Residence Costs + Maintenance</td>
<td>Sales to Projects</td>
</tr>
</tbody>
</table>

### Domain Balance Sheet

<table>
<thead>
<tr>
<th>Year, y</th>
<th>Cost, C(y)</th>
<th>Benefit, B(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>y=SD</td>
<td>Domain Analysis + Asset Development</td>
<td></td>
</tr>
<tr>
<td>y&gt;SD</td>
<td>Asset Development + Continued Domain Analysis</td>
<td>Asset Sales to Projects</td>
</tr>
</tbody>
</table>

### Application Balance Sheet

<table>
<thead>
<tr>
<th>Year, y</th>
<th>Cost, C(y)</th>
<th>Benefit, B(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>y=SD</td>
<td>Purchase of Reusable Assets</td>
<td>Savings of Development Costs</td>
</tr>
<tr>
<td>y&gt;SD</td>
<td></td>
<td>Quality Gains</td>
</tr>
</tbody>
</table>

### Corporate Balance Sheet

<table>
<thead>
<tr>
<th>Year, y</th>
<th>Cost, C(y)</th>
<th>Benefit, B(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>y=SD</td>
<td>Infrastructure</td>
<td></td>
</tr>
<tr>
<td>y&gt;SD</td>
<td>Domain Costs</td>
<td>Application Benefits</td>
</tr>
</tbody>
</table>

**Figure 4 Investment Cycle Balance Sheets**
• Non-Linear Cost Effects
This model uses COCOMO 2.0 equations to estimate the development costs as a function of the size of an asset. The equations derive development costs as an exponential function of the size and they recognize that costs are not linear in size [18].

• Integration Costs
It is assumed that the applications are developed from reused, adapted, and customized assets such that the integration of two components of size A and B is:

\[ CI = C(A+B) - C(A) - C(B) \]

where
\[ CI = \text{Integration Cost} \]
\[ C(X) = \text{Cost to develop an asset of size } X \]

• Quality Gains
Quality gains are quantified in person-months and are to be considered the as savings in operating costs of reusable products over the lifetime of the four investment cycles.

• Code Inflation
This cost model recognizes that code developed for reuse could be significantly larger than the code developed from single use. It makes provisions for adjusting the size of a reusable asset to the size of an asset with equivalent functionality.

• Time-to-Market
Time-to-Market is a benefit of the application engineering phase with the following advantages:
1. Increased sales volume
2. Increased market share
Thus, earlier time to market enables the consumer to secure a bigger market share and an increased sales/use of the product. This model only quantifies the increased sales volume and not increased market share.

4.4 Conclusion

The integrated cost model enables managers to make decisions on the four levels of the investment cycle: component, application, domain and corporate engineering cycle. As a result of these investment cycles cascading information into one another, one can calculate the benefits and costs of reuse and determine if a project is worth the risk. Although this model makes a number of assumptions, it encompasses the features of many of the existing cost models. The next chapter will detail the existing cost models and show how the integrated cost model incorporates these models.
Chapter 5. A Classification of Software Reuse Cost Models
The integrated cost model stems from an analysis of existing cost models and attempts to generalize these existing models under a common generic framework. This chapter presents a survey of existing cost models and discusses to what extent the integrated cost model generalizes them.

5.1 A Classification Scheme

Each cost model is characterized by the following features: Investment Cycle, Economic Function, Cost Factors, Reuse Organization, Scope, Hypothesis, and Viewpoint.

5.1.1 Investment Cycles

Most decisions that arise in the software reuse can be modeled as a return on investment decisions. Four distinct investment cycles have been identified: corporate investment cycle, domain engineering investment cycle, application engineering investment cycle and component engineering investment cycle. Each of these cycles reflects an economic decision and can be quantified by means of economic functions. Software reuse cost models can be characterized first and foremost by these decision cycles.

5.1.2 Economic Functions

The four investment decisions can be quantified in terms of the following economic functions: Net Present Value, Payback Value, Average Return on Book Value, Internal Rate of Return, and Profitability Index. These functions are based on the five different functions that were presented by Favaro [17] and are described in detail in Chapter 3 for each investment cycle. Existing cost models can be characterized by these functions that are used to quantify the reuse decisions.

5.1.3 Cost Factors

Cost factors are any identifiable feature of a new project that is likely to significantly change one or more nominal values of the baseline project. These cost factors specify what aspects of the reuse decision are to be considered. Examples of cost factors are error rate, overhead, KLOC, assessment, size, risks, etc.

5.1.4 Reuse Organization

Since the organizational structure has some impact on how costs are determined, charged, and accounted for, this feature indicates whether a specific organizational structure is assumed. Many models assume a specific organizational structure but few explicitly
account for it. Fafchamps [31] and Basili [4] introduces five possible producer-consumer relationship models. In these models, the consumer designs and develops products with reusable components while the producer designs and develops these reusable components. Reuse design and development is dependent on the organization size, talent pool and structure, functional structure, i.e., whether the group is diverse with technical expertise, and divisional structure, i.e., specialists grouped into teams. The five models presented by Fafchamps [31] and Basili [4] are

- Lone Producer
- Nested Producer
- Pool Producer
- Team Producer
- Experience Factory

In the Lone Producer team, reuse services exist in two consumer teams that design, develop, and maintain reusable components. In the Nested Producer team, a product team with reuse services and expertise may have a secondary reporting line. In the Pool Producer team, two or more teams collaborate to produce and share components. In the Team Producer team, the reuse team handles the organizational structure. The experience factory is based on the quality improvement paradigm: characterize the environment, set the goals, choose the process for improvement, execute the processes of constructing the products, analyze the data and practices to evaluate, and package the experiences for future projects [4].

The advantages and disadvantages of each organization are summarized in Table 1 [4,31].

5.1.5 Scope

Scope indicates whether the decision is short-term, long-term, limited, or based on technical/managerial considerations. Limited decisions based on technical/managerial considerations typically depend on factors like the life expectancy of the reusable asset. Decisions with a limited scope are based on strategic considerations; i.e., investment must break even in three years.

5.1.6 Hypothesis

Hypothesis summarizes what the model assumes or neglects to take into account with its model, i.e., failure to account for the discount rate of the resources and/or the assumption that the software development costs are linear. Other hypotheses include failing to amortize resources over time, ignoring quality gains, neglecting integration costs, ignoring of quality gains, focusing on productivity gains, ignoring the inflation of code size due to reuse, and ignoring time-to-market benefits that result from software reuse.
5.1.7 Viewpoint

Viewpoint indicates what part of the software reuse initiative this model takes the standpoint of: corporate executives, producer staff, consumer staff, library managers and/or component providers. The viewpoint is different for the decision cycles, i.e., someone may wish to optimize one function while taking the viewpoint of another. For example, a corporate manager may wish to optimize the NPV of domain engineering. In this case the cycle is domain engineering but the viewpoint is corporate manager.

5.1.8 Instantiation

The term instantiation indicates how the integrated cost model integrates/substantiates this model in its model.

5.2 Various Economic Models

A large number of economic models have been proposed that differ significantly in their solution to the software reuse cost estimation problem (Table 4) [32]. The variety of attributes described above account for the differences among these models as illustrated in the table.
<table>
<thead>
<tr>
<th>Reference</th>
<th>Reuse Cycle</th>
<th>Economic Function</th>
<th>Cost Factors</th>
<th>Reuse Organization</th>
<th>Scope</th>
<th>Hypotheses</th>
<th>Viewpoint</th>
<th>Instantiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bollinger and Pfleeger, 1990</td>
<td>CE, RE</td>
<td>Reuse benefits</td>
<td>CE costs with and without reuse</td>
<td>Producer/ consumer</td>
<td>Across Projects/ Application s</td>
<td>RE with benefits, CE with and without reuse</td>
<td>Corporate</td>
<td>CE with amortization cost sharing domains (DE)</td>
</tr>
<tr>
<td>Barnes and Bollinger 1991</td>
<td>DE, CE, AE</td>
<td>Breakeven NPV, ROI Quality Gains</td>
<td>AE costs with and w/o reuse CE, DE</td>
<td>Producer/ consumer</td>
<td>Across applications</td>
<td>Producer costs = consumer benefits</td>
<td>Producer/ consumer</td>
<td>AE quality gains, DE cost, CE costs and benefits</td>
</tr>
<tr>
<td>Gaffney and Cruickshak 1992</td>
<td>DE, AE</td>
<td>Return on investment breakeven function</td>
<td>AE costs prorated DE costs productivit y</td>
<td>Internal procurement, team producer</td>
<td>Long term view across many applications</td>
<td>Ignores integration costs</td>
<td>Corporate</td>
<td>Combining DE cycle w/relevant AE cycles</td>
</tr>
<tr>
<td>Margano and Rhoades 1992</td>
<td>CE</td>
<td>Payback, PV, productivity gains</td>
<td>Component costs, overhead, investment</td>
<td>Pool producer</td>
<td>AE scope, focus on component</td>
<td>Ignores training, process impacts</td>
<td>Project manager, component developer</td>
<td>CE, with related DE, AE</td>
</tr>
<tr>
<td>Schimsky, D.1992</td>
<td>DE</td>
<td>Breakeven AE costs</td>
<td>Develop, maintain, reuse code</td>
<td>Producer/ consumer</td>
<td>DE costs</td>
<td>Cost of component in SLOC, benefits based on breakeven point</td>
<td>Project</td>
<td>Breakeven point</td>
</tr>
<tr>
<td>Malan and Wenzel 1993</td>
<td>DE, CE</td>
<td>NPV</td>
<td>Overhead lifecycle developme nt</td>
<td>Domain centric</td>
<td>Within a domain</td>
<td>DE costs and benefits</td>
<td>Domain Manager, Asset Developer</td>
<td>DE costs benefits over a discount rate</td>
</tr>
<tr>
<td>Poulin Caruso 1993</td>
<td>CE, AE, RE</td>
<td>ROI, NPV, PI</td>
<td>KLOC /KLOC</td>
<td>Producer/ consumer</td>
<td>Project level, corporate level</td>
<td>Ignores time value of money, ROI based on IRR</td>
<td>Corporate</td>
<td>ROI for RE,AE under black box</td>
</tr>
<tr>
<td>Frakes and Terry 1994</td>
<td>AE, RE</td>
<td>Reuse Level, Reuse Frequency</td>
<td>Number of references to items</td>
<td>Application Centered</td>
<td>Application / Component</td>
<td>Black box only, reuse thresholds</td>
<td>Project/Cor porate/Com ponent</td>
<td>Different functions / metrics</td>
</tr>
<tr>
<td>Kain, 1994</td>
<td>DE</td>
<td>Return on Investment</td>
<td>DE costs, AE costs with, w/o reuse</td>
<td>Three teams producer, consumer, library.</td>
<td>Across multiple projects</td>
<td>No quality gains and no effect on time</td>
<td>Project decisions at corporate level</td>
<td>Separating investment cycles</td>
</tr>
<tr>
<td>Wayne, C. Lim, 1994</td>
<td>AE</td>
<td>NPV</td>
<td>Component Costs, Productivity, Reuse with KNCSS</td>
<td>Producer/ consumer</td>
<td>Project lifecycles</td>
<td>NPV in lifecycle of reuse, on overhead costs</td>
<td>Corporate project-wide</td>
<td>Focus on AE cycle,</td>
</tr>
<tr>
<td>COCOMO 2.0, Boehm et al., 1995</td>
<td>DE, VS AE</td>
<td>Lifecycle cost</td>
<td>RUSE vs ESLOC</td>
<td>Any Producer/Consumer Organization</td>
<td>Arbitrarily Large</td>
<td>Highly calibrated to specific developments environments</td>
<td>Corporate-wide vs. project-wide</td>
<td>Using components level COCOMO estimates</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------</td>
<td>----------------</td>
<td>----------------</td>
<td>-----------------------------------</td>
<td>-------------------</td>
<td>----------------------------------------------------------</td>
<td>-----------------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>Devanbu et al. 1996</td>
<td>AE, RE</td>
<td>Reuse Benefit</td>
<td>Size, structure of application</td>
<td>Internal Reuse</td>
<td>Project</td>
<td>Rigorous axiomatisation</td>
<td>Project</td>
<td>Abstract metrics vs. cost model</td>
</tr>
<tr>
<td>Favaro 1996</td>
<td>CE</td>
<td>NPV, PI, ARBV, IRR, payback</td>
<td>DE, CE</td>
<td>Lone Producer</td>
<td>Across projects</td>
<td>Ignores specifics of AE, DE</td>
<td>Corporate</td>
<td>Focus on component DE</td>
</tr>
<tr>
<td>Mili 1996</td>
<td>CE</td>
<td>Return on investment</td>
<td>Component levels factors (e.g. LOC)</td>
<td>Organization that merge DE with AE</td>
<td>Domain wide</td>
<td>Equates component ROI with reusability</td>
<td>Producer/corporate viewpoint</td>
<td>Matches component level ROI</td>
</tr>
<tr>
<td>Poulin 1997</td>
<td>DE, RE, AE</td>
<td>Proj. ROI, DCA, SCA, RCA, CSW, Threshold, PL ROI</td>
<td>LOC, RSI, RCR, SIRBO, error rate, error cost, RCWR</td>
<td>Merging producer and consumer teams</td>
<td>Limited term product line engineering</td>
<td>Low cognizance for time value of money</td>
<td>Product line perspective (one DE, many AE)</td>
<td>Separate investment cycles, add new cycles</td>
</tr>
<tr>
<td>Favaro J., Favaro K., Favaro P., 1998</td>
<td>RE</td>
<td>Present value, NPV</td>
<td>Cash flow, discount rates, risks</td>
<td>Producer/consumer</td>
<td>Corporate</td>
<td>NPV take in account risks</td>
<td>Corporate</td>
<td>NPV in corporate cycle</td>
</tr>
<tr>
<td>COCOTS 1999</td>
<td>AE</td>
<td>Application engineering costs</td>
<td>Assessment, tailoring, glue code volatility IV &amp; V</td>
<td>External procurement of COTS components</td>
<td>Across domains</td>
<td>COTS acquired as executable code</td>
<td>Project/corporate</td>
<td>AE cycle, additional hypotheses</td>
</tr>
</tbody>
</table>

CE  Component Engineering  
AE  Application Engineering  
DE  Domain Engineering  
RE  Corporate Engineering  
KLOC  Thousand Lines of Code  
KNCSS  Thousand 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  

**Table 4 Software Reuse Economics Models**
5.2.1 Barnes and Bollinger

Barnes and Bollinger [10] define their reuse investment relation as the comparison of the reuse investment with the reuse benefits. The reuse investment is defined as any costs that do not directly support the completion of an activity’s primary development goals but are instead intended to make more work products of that activity easier to reuse (i.e., classifying components). Reuse benefits are the difference between the activity cost with and without reuse. The difference between Barnes and Bollinger and the integrated cost model is that the integrated cost model prorates the investment.

Barnes and Bollinger also define the concept of quality of investment (the Benefits/Total Reuse Investment). When the value of this function is greater than 1, good returns can be expected. The latter is achieved by increasing reuse with components of a high level of unique expertise and with high quality libraries with good documentation and little adaptation required; by reducing cost by making components easy to find, adapt, and integrate; and by reducing investment costs by not overextending components or making them too general. The integrated cost model accounts for this concept in the application engineering periodic benefits. The quality of investment is a subset of the NPV calculation in the integrated cost model.

5.2.2 Bollinger and Pfleeger

In Bollinger and Pfleeger’s cost model [7], 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. Furthermore, although producers incur initial cost and receive little or no benefit from reuse efforts initially as in this model, the integrated cost model tries to eliminate the effort spent by consumers to understand, retrieve and adapt components through the component engineering cycle and associated domain engineering cycle. In these cycles, components developed for reuse are associated with domains in the domain analysis (domain engineering investment phase), to lessen this extra cost by the consumers.

These reuse investments are both excess costs and optimal costs, which are not needed to complete a project. The reuse benefits are defined as
B = \sum_{i=1}^{n} (\text{Development}_i - \text{Adaptation}_i) - \text{Investment}

where

- n = Number of projects or development activities
- Development = Costs of building the product without reusable products
- Adaptation = Cost of finding, customizing, and integrating the reusable part versus developing it
- B = Benefit
- Investment = Total cost of all resources applied specifically to making the product

Although initially the investment by the consumer is high, the integrated cost model tries to demonstrate to the producer the future benefits/risks associated with making this reuse investment. Through the calculations of the various economic functions, a producer can determine if these excess costs will be beneficial to the corporation in the future.

The cost model of Bollinger and Pfleeger does not always take into account both the inclusion effect, where use of one work product explicitly includes reuse of subsequent work products and also 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, and all domains are associated with an application. Furthermore, components or domains can be associated with an application. The reuse of design, documentation, and work experience are taken into account in the investment costs of the component and application and the corporation.

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.

Lastly, the integrated cost model takes advantage of Bollinger and Pfleeger’s idea of a baseline project, (1) the concepts of cost-sharing domains (grouping present and future projects for which costs can be added together and treated as a unit value, i.e., reuse of a given set of components), (2) the reuse benefits inequality, and (3) the delta cost estimator (measurement of the impact of using previously defined baseline projects). The integrated cost model uses these concepts, the different engineering cycles, and different economic functions as part of the groundwork for a cost estimation tool that enables management to make informed decisions about the economics for reuse.
5.2.3 Schimsky

According to Schimsky [13], the software cost is a function of cost drivers and software size in SLOC. Consequently, the integrated cost model allows users to determine the reuse economics via SLOC and person-months. In Schimsky’s model,

\[
\text{Cost} = (\text{cost to develop a reusable SLOC})(\text{library size in SLOC}) + \\
(\text{average yearly cost to operate library})(\text{years}) + \\
(\text{cost to use a reusable SLOC})(\text{average yearly SLOC use})(\text{years}) - \\
(\text{cost to use a normal SLOC})(\text{average yearly SLOC use})(\text{years})
\]

This equation 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. This breakeven point is determined by these charts and by the reuse ratio, library size, and the cost of normal code development and assumes that the cost to develop with reuse is .05-.65 times the cost to develop new [13].

5.2.4 Margono and Rhoades

Margono and Rhoades [16] discuss many other models including the SPC Model, GTE Model, and the JIAWG Model (combination of Ada-COCOMO and SPC reuse cost models). These models reflect the amortized cost of producing a reusable asset, cost to reuse N times, and the cost to develop an asset if there were not reuse.

In the SPC model, the cost of a software product is the cost to develop new plus the cost of reusing existing software:

\[
C = (1 - R)(1) + (R)(b + \frac{E}{n})
\]
where

\[ R = \text{Proportion of reusable code in a software product} \]
\[ b = \text{Cost of integrating reusable code relative to the creation and} \]
\[ \text{integration of a new code} \]
\[ E = \text{Cost of developing reusable code relative to the creation of all new} \]
\[ \text{code} \]
\[ N = \text{Number of uses over which the cost of the reusable code is amortized.} \]

In the GTE-Contel model, centralized cost-sharing domains (CSD) are used across multiple projects and the benefit is the net cost to reuse (cost without reuse minus the cost with reuse) minus the reuse investment (cost to identify, classify, develop/purchase).

\[ B = \left( \sum_{i=1}^{n} (D_i - A_i) \right) - I \]

where

- \( B \) = Net reuse benefit
- \( D \) = Development cost without reuse
- \( A \) = Development cost with reuse (finding, customizing, and integrating reusable components)
- \( I \) = Total reuse investment
- \( n \) = Number of development activities

The JIAWG model is based on a reusable software object (RSO). An RSO is a lifecycle product that includes requirements, designs, algorithms, code, test cases, etc., which are developed to be reused. Thus the cost of software development with reuse in staff months is

\[ E = c(l + \frac{OAR \cdot b_{18}}{m} - OR \cdot b_{19})E_R \]

where

- \( c \) = Adjustment factor for application domain
- \( b_{18} \) = Reuse cost factor
- \( m \) = Number of RSOs used across deliveries in a project
- \( b_{19} \) = Reuse benefit factor
- \( E_R \) = Cost to develop without reuse using Ada-COCOMO model
- \( OAR \) = Object acquisition ratio
- \( OR \) = Effective object reuse ratio

The latter models were used to access the reuse effort on the Federal Aviation Administration’s Advanced Automation System project. The AAS model was 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 amortized cost of producing \( X \) is:
\[
C_p = \frac{U_p L}{n}
\]

where
- \(L\) = Estimated SLOC
- \(n\) = Number of potential consumers
- \(U_p\) = Estimated costs for developing X in $/SLOC
- \(C_p\) = Amortized cost of producing x

while the cost of reusing or consuming product X is:

\[
C_c = nU_c L
\]

where
- \(U_c\) = Estimated costs of reusing X in $/SLOC
- \(L\) = Estimated SLOC
- \(n\) = Number of potential consumers
- \(C_c\) = Cost of reusing

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 the way 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 uniquely developed. Additional overhead cost due to reuse testing is estimated to be 25%. This testing overhead is assumed in the application engineering cycle of the integrated cost model. An additional 10% overhead exists in the architecture phases (modularizing for reuse and reviewing requirements to identified common requirements). The latter additional overhead is assumed in the domain engineering cycle of the integrated cost model. All models indicate that the cost to develop for reuse is twice that of non-reuse, but reuse is good for rapid prototyping and performance improvement. This model does not have parameters for system reliability, understandability of the reuse component, and maintainability of the component. Furthermore, it does not allow for an estimation of the cost tradeoffs of using an existing component and rewriting some of the code or writing new code. Also reintegration (the integration of software that has been developed and integrated in an earlier and similar system and which must be reintegrated with some additional functionality) was considered to be 10% of the total development cost. In the integrated cost model this cost is incorporated in the application development cycle.

This model considers the costs at the component/project level rather than at the corporate level and did not account for adaptation/modification of reusable components. It stressed 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 uses actual and estimated data in the component engineering cycle.
5.2.5 Poulin/Carouso

Poulin and Carouso’s model [9] of software reuse is very close 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 reuse percent of this model is the portion of product lines in KLOC that are saved by reusing software from other products or product releases and indicate the amount of reuse in a product or an organization’s practice. The Project Level ROI consists of the costs avoided by building a project from reusable code less the cost to write for reuse:

$$\text{ROI} = \text{RCA} + \text{ORCA} - \text{ADC}$$

where

- RCA = Reuse Cost Avoidance (the reduction of total product costs as a result of reuse)
- ORCA = Other RCA benefits of reuse in initial program and is defined as

$$\text{ORCA} = \sum_{i=1}^{n} \text{SIRBO}_i (1 - \text{relative cost of reuse}_i)(\text{New Code Cost}_i) + \sum_{i=1}^{n} \text{SIRBO}_i (\text{TVUA Rate}_i)(\text{Cost per TVUA}_i)$$

where

- SIRBO = Shipped source instructions (non-commented instructions)
- TVUA Rate = Software development error rate
- TVUA = Cost to maintain components
- Relative cost of reuse = Cost to integrate versus the cost to create new
- ADC = Additional Development Costs

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+discount rate) to the revenue year considered:

$$C_0 = \frac{(R_1 - C_1)}{(1 + k)} + \frac{(R_2 - C_2)}{(1 + k)^2} + ... + \frac{R_n - C_n}{(1 + k)^n}$$

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:

$$\text{NPV} = -C_0 + \frac{(R_1 - C_1)}{(1 + k)} + \frac{(R_2 - C_2)}{(1 + k)^2} + ... + \frac{(R_n - C_n)}{(1 + k)^n}$$
where

\[ C_0 = \text{Corporate reuse start-up costs} \]
\[ R_i = \text{Revenue (savings) in year } i \]
\[ C_i = \text{Costs in year } i \]
\[ n = \text{Number of years revenues are to be considered} \]
\[ k = \text{Discount rate} \]

The Productivity Index is the productivity with reuse relative to the productivity without reuse. All of these ROI cost models account for the savings in maintenance by means of the costs avoided from not having to fix errors in newly developed code. Poulin and Caruso do not indicate how to obtain the corporate costs and revenues in order to calculate these economic functions as the integrated cost model does. This model assumes the user will be able to supply the following parameters: SSI/CSI, RSI, SSI/CSI written for reuse, Cost/LOC, RCR, RCWR, APAR rate, cost/TVUA, and SIRBO, Cost/LOC, RCR, APAR/Rate, Cost/TVUA from other projects using the code. This model does not take into account the domain engineering costs/benefits in the calculation of the corporate ROI.

Poulin and Caruso also show the NATO model as related work. They claim that this model adjusts for time-value of money but there is little guidance on how to collect data required for this model. The NATO model consists of “listing the major benefits and costs of reuse, then applying the time-value of money formulas to adjust for the future values” [46] but has no definitive means of collecting the data. In this model the total savings due to reuse, less the accession and maintenance costs, are:

\[ \text{NSP} = [\text{NSR}(N)] - (C_a + C_m) \]

where

\[ \text{NSR} = S_r - C_r \]

where

\( S_r = \text{Savings due to avoided cost (sum of the costs avoided each time the component is used) } \)
\( C_r = \text{Relative cost of reuse or the costs incurred each time the component is reused including identification, retrieval, familiarization, modification, and installation} \)
\( C_a = \text{Cost to add the component to the library including obtaining raw material, developing the complete component, and installing it in the library} \)
\( C_m = \text{Cost to maintain the component in the library} \)
\( N = \text{Number of uses} \)
Like Poulin and Caruso’s model, this model focuses on application engineering and does not take into account domain engineering as the integrated cost model does.

### 5.2.6 Lim

For Lim [33], productivity is defined in terms of the reuse ratio

\[
\text{Reuse Ratio} = \frac{(\text{Reuse KNCSS} + \text{Modified KNCSS})}{\text{Product Total KNCSS}} (100)
\]

The cost of reuse includes creating or purchasing reuse work products, libraries, tools and implementing reuse-related processes, all of which are taken into account for in the investment costs of the corporate engineering cycle of the integrated cost model. Net present value takes the estimated value of reuse benefits and subtracts from it associated costs, taking into account the time-value of money. According to Lim, NPV recognizes the potential increased profit from shortened time-to-market and by accounting for risk. The figures in this paper do not include overhead costs including manager's time but include only the time spent by the producer to create the reusable work product. The integrated cost model defines productivity as the gains in development costs by using reusable components versus writing the code from scratch. Both models recognize that productivity decreases with increased modification of the original code.

### 5.2.7 Favaro/Favaro/Favaro

Favaro [34] emphasizes the economics of software reuse in the cost model. The premise of his model is present value, net present value, and the effect of the discount rate \( r \) on these economic functions. He considers the discount rate to be a penalty for delay of cash flow (like interest on a loan). Favaro argues that NPV undervalues many reuse-related investments because DCF does not use a risk-adjusted discount rate (RADR) the return expected by investors which is proportional to the stock’s/project’s sensitivity to market movements (Capital Asset Pricing Model (CAPM)). Thus, systematic risk is not accounted for in typical reuse economic models. In the integrated cost model it is up to the user to input the discount rate that he feels is appropriate. Using Decision Tree Analysis, the failings of DCF can be overcome by modeling different kinds of outcomes and management decisions, but this type analysis can become unwieldy, and it does not account for the treatment of risk since it uses a single discount rate. The alternative: Contingent Claims Analysis (CCA) based on option pricing theory where an option is a derivative whose value depends on the value of the underlying asset. The risk of the option varies with time and price and is based on the Black-Scholes formula:

\[
C = [N(d_1)P] - [N(d_2)PV(EX)]
\]
where

\[
d_1 = \frac{\log\left(\frac{P}{PV(EX)}\right) - \frac{\sigma \sqrt{t}}{2}}{\sigma \sqrt{t}}
\]

\[
d_2 = d_1 - \sigma \sqrt{t}
\]

N(d)=Cumulative normal probability density function
EX=Exercise price
PV = Number obtained by discounting back by the continuously compounded risk-free interest rate
t=Time to exercise date of option
P=Price of security
\sigma = Standard deviation per period of continuously compounded rate of return on security

The problem with this formula is that options/assets are traded on financial markets which is not the case with software reuse and a single standard deviation is assumed, which may not be realistic with software reuse. Of these techniques CCA is recommended.

Using the financial principles of the latter economic investment strategies, Favaro proposes a value-based reuse investment (VBRI) framework based on value-based management:

1. Economic value maximization drives reuse investment strategies for business (where the key to this principle is reduced time-to-market and value creation of the asset)
2. Strategy drives selection of reuse investments
   A. The strategic context elevates us to the level of business
   B. The two primary determinants of business value creation are market economics and competitive position
3. Investments are actively structured to maximize embedded strategic option (Use of DCF to maximize value and strategic options is key to this principle)
4. Both discounted cash flow and options-based approaches are used to capture the full value of reuse investments
   A. Discounted cash flow techniques are used to capture the operational benefits of reuse
   B. Reuse investment is all about strategic options (DCF augmented with additional mechanisms)
   C. Option-based approaches are used to capture the strategic benefits of reuse.

To this end the integrated cost model incorporates the principles of VBRI by using DCF’s to calculate NPV and ROI. Furthermore, the archival/analytical functions of the integrated cost model’s automated tool captures the strategic options.

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.

5.2.8 Kain

Like the integrated cost model, Kain’s model [14] is based on the costs and benefits associated with reuse. Reuse costs include the tasks of defining and building reusable components, managing the storage, recording, and maintaining the components. “Reuse benefits include the savings from not having to recreate similar functionality in other applications and the use of proven components.” Kain’s economic model is based on Henderson-Sellers’ definition of ROI:

$$\text{ROI} = \frac{S - (C_r + C_m + C_d + C_g)}{C_g}$$

where

$S$=Cost of the project without reuse

$C_r$=Cost of finding reusable components

$C_m$=Cost of modifying the components

$C_g$=Cost of generalization

However, Henderson-Sellers only accounted for object-oriented class components. Kain explained that reuse is a long-term investment and that without reuse, maintenance can be extremely high (80% of the total budget). This maintenance cost can be reduced with reusable components because reusable components are cohesive (based on a problem domain), encapsulated (separation of the component interface and its implementation), and loosely coupled (not dependent on other components).

Kain proposes a return on the 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 (1) taken at the corporate level, (2) ignores the time variance of resources (labor months, money, or other assets), (3) fails to distinguish between up front costs and periodic costs, and (4) 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.
5.2.9 Malan and Wetzel

The Malan and Wetzel model [35] addresses a number of deficiencies in many of the latter models (Barnes and Bollinger, Bollinger and Pfleeger, Gaffney and Cruickshank and Lim). Malan and Wetzel’s model incorporates the reuse-related cost factors (overhead, development with and without reuse, risk, time value of money, profit from shortened time-to-market, etc.) 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:

\[
S = \left( \sum_{i=1}^{n} \left( \sum_{j=1}^{m} \frac{C_{Nj}}{(1 - i)^j} \right) - \sum_{j=1}^{r} \frac{C_{CRj}}{(1 - i)^j} \right) p_i + \sum_{j=1}^{r} \left( \frac{C_{Nuj}}{(1 - i)^j} - \frac{C_{CRuj}}{(1 - i)^j} \right) - \frac{C_{PRj}}{(1 - i)^j} + \frac{C_{PRUj}}{(1 - i)^j} + \frac{A_j}{(1 - i)^j} \right]
\]

where:
- \(n\) = Number of products sharing the reusable components
- \(C_{ni}\) = Cost to develop product \(i\) without reuse
- \(C_{cri}\) = Cost of creating product \(i\) with reuse
- \(C_{ni} - C_{cri}\) = Expected consumer cost saved for product \(i\)
- \(C_{nu} - C_{crui}\) = Cost to produce upgrade \(j\) to product \(i\) without reuse less the consumers cost to integrate the upgraded component into product \(i\)
- \(C_{prui}\) = Discounted expected producer cost to create upgrade \(j\)
- \(C_{pr}\) = Expected cost that the producer incurs in producing the reusable components
- \(A\) = Reuse-specific overhead and setup costs incurred by the family of products
- \(C_{ni} - C_{cri}\) = Consumer savings for product \(i\)
- \(m\) = Expected number of periods taken to produce the resuable asset
- \(f\) = Expected period to finish creating the product with reuse
- \(nf\) = Expected period to finish creating the product without reuse
- \(r\) = Number of upgrades
- \(Y\) = Number of periods under consideration
- \(j\) = Period
- \(p_i\) = Probability that the reuse instance \(i\) will be actualized

Malan and Wetzel included the time value of money, maintenance costs and savings, and uncertainty (whether the asset will be reused). They validated their model with a hypothetical scenario. Furthermore, though time-to-market gains are discussed, they are not quantified. Thus, the integrated model attempts to quantify time-to-market gains unlike the Malan and Wetzel model.
5.2.10 Gaffney and Durek

Gaffney and Durek [36,37] propose three models: simple, cost-of-development, and general models. All three models assume that the software system is composed of only new and reused code. Furthermore, each of the models deals with relative costs. These models reflect the cost to develop software with reuse relative to developing without reuse, the effect of reuse on productivity and the number of reuses to breakeven.

In the simple model

\[ C = (b - 1)R + 1 \]

where
- \( C \) = Cost to develop the product (1=no reuse)
- \( R \) = Proportion of reused code in the product
- \( b \) = Cost to incorporate reuse relative to the cost of building components from scratch. It is dependent on the type of reusable component (code, design, requirements, and testing).

Productivity is then defined as \( 1/C \).

In the cost-to-development model,

\[ C = (1 - R) + R\left(b + \frac{E}{n}\right) \]

where
- \( E \) = Cost of producing reusable code relative to the cost of producing non-reusable code (usually 1.25-2.0) and is defined as
  - Cost per unit to incorporate a reusable component
  - Cost per unit to develop a non-reusable component

where
- Cost per unit to incorporate a reusable component = Requirements + design + code and debug + testing + archive and repository costs of the component.
- \( n \) = Number of uses over which the code cost is amortized
- \( R \) = Proportion of reused code in the product

The payoff threshold is defined as \( E/1-b \). As code becomes more “reusable” the number of uses for payoff decreases.
In the general model [36]

\[
C = (1 - R_c - R) + \left( R_c \frac{E}{m} \right) + R \left( b + \frac{E}{n} \right)
\]

where

- \(E\) = Relative cost of creating reusable code
- \(R\) = Proportion of reused code in this product
- \(n\) = Number of uses over which the code cost is amortized
- \(m\) = Number of uses over which the reusable new code is to be amortized
- \(R_c\) = Proportion of new code created to be reusable on other projects
- \(b\) = Cost relative to that for all new code of incorporating the reused code into the new product

The general model is an extension of the cost-to-development model where code is reused among multiple repositories. \(b\) was estimated as the cost to reuse a component/cost to build a component from scratch where the requirements phase constitutes 27\%, design 50\%, implementation 15\%, and testing 8\%. Thus, if reuse is used in all but the testing phase, \(b\) is .08

These authors also discuss quantifying quality. Unlike the integrated cost model that considers quality gains to be the savings in maintenance, Gaffney and Durek considered quality to be the number of errors discovered, which should be positive because reuse provides for error discovery. Thus,

\[
LR = (1 + R) + R \left( p^{(n-l)} \right)
\]

where

- \(p\) = Latent error content/(errors discovered and removed + latent error content + integration and system tests)
- \(LR\) = Latent error content of the software
- \(R\) = Proportion of reused code in the product

Lastly, these authors considered the effect of reuse on time (schedule) required to develop a product:

\[
S = Bk^vT^q
\]

where

- \(S\) = New lines of code
- \(B\) = Development constant representing the development environment and complexity of the software
- \(K\) = Development labor (labor-months)
- \(T\) = Development schedule (years)
- \(v, q\) = Empirically fitted numbers
The latter equation indicates that reuse may impact development schedules. These authors’ model amortizes the cost of the reuse program including the additional cost to build reusable components across all components using the component [Poulin and Caruso].

5.2.11 Kang and Levy

Kang’s economic model [2] focusses on object-oriented programming. In object-oriented programming, programs are based on objects, data abstraction and encapsulation. Kang explores the different reuse organizations and their respective costs. For example, to hire an expert would be based on the following:

\[ D = C \frac{N}{P} + KNE \]

where
- \( C \) = Yearly salary
- \( P \) = Number of lines of code produced yearly
- \( E \) = Error rate
- \( K \) = Cost for fixing the error
- \( N \) = Number of lines of code in the system

The problem with an expert is that he may not be economical for the organization and experts may not want to do only specialized work. In a centrally supported organization, planning for reuse becomes part of the business strategy. However, in this organization the operational cost would need to be less than the total development cost:

\[ \text{operational cost} < \sum_{i=1}^{n} N_i C_i \]

where
- \( N_i \) = Number of users of module \( i \)
- \( C_i \) = Cost of developing module \( i \) without reuse
- \( n \) = Number of modules.

The problem with this organization is that the return on investment takes a long time, especially due to the overhead of training, domain analysis, and reusable design/development which enables a trade relationship among producers and consumers. In this organization, developers are free to sell/buy components. Economically, developing reusable components will cost more, but it lets developers profit from their work and encourages competition. This approach is worthwhile only if
\[ \sum_{i=1}^{n} N_i C_i \geq \sum_{i=1}^{n} C_i K_i + \sum_{i=1}^{n} (N_i - 1) L_i \]

where
- \( n \) = Number of modules
- \( C_i \) = The cost of developing module \( i \)
- \( N_i \) = The number of users of the module \( i \)
- \( K \) = The extra cost of making the module \( i \) reusable
- \( L_i \) = The costs associated with learning and incorporating the module \( i \) in the system

Although the integrated model does not make special provisions for object-oriented programming, a component can be an object, class, etc.

### 5.2.12 Balda

Balda [12] bases his model on the COCOMO equation and the reuse lifecycle model with additional terms for code developed for future reuse, black box reuse, and white box reuse. He also made the following assumptions:
1. Cost of reuse at the design and code levels is the same.
2. Modifications made to the reusable components do not affect the costs of reusing the component.
3. All reuse implementation methods cost the same.
4. Cataloging and retrieving affect the costs but assume that the components can be easily located and retrieved.
5. Projects are developed using the waterfall lifecycle model.

Thus, the effort to develop for reuse in person-months is

\[ PM = \alpha N_1^\beta + 20\delta\alpha N_2^\beta \]

where
- \( N_1 \) = Unique KDSI
- \( N_2 \) = KDSI developed for reuse
- \( \alpha, \beta \) = COCOMO
- \( \delta = .0909 - .1739(180 - 348\% \text{ of effort to develop for reuse}) \)
- \( 20\delta \) = Increase in reuse effort for development from reuse versus reusing a component
- \( PM \) = Effort

Like Balda the integrated cost model uses COCOMO to determine the development costs and also takes into account black box and white box reuse costs in the component and application engineering cycles.
5.2.13 COCOMO 2.0

COCOMO 2.0 [38] represents an evolution from the original COCOMO ('81) cost model that was based on the waterfall model. It extends the original model in a number of ways, including encompassing modern programming paradigms such as reuse, COTS, re-engineering, object oriented development, and non-sequential, rapid development models. 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:
1. It incorporates domain engineering costs by means of the RUSE factor, which reflects the envisaged scope of reuse of the asset at hand.
2. 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.

COCOMO 2.0 no longer outputs point estimates. Instead, COCOMO 2.0 outputs range estimates of software cost and effort that is tied to the degree of the definition of the estimation inputs. Unlike COCOMO that was based on lines of code, COCOMO 2.0 is based on function points and language or lines of code. To adjust for reuse, COCOMO 2.0 introduces the NOP (New Object Points) factor to adjust the object point count for reuse and %reuse which is the percentage pro-rated by degree of reuse of screens, reports, and 3GL modules reused from previous applications. Details on this object-point estimation can be found in Boehm, 1995. Lines of code is based on the logical source statement and is directly calculated using the automated metrics tool of Amadeus.

COCOMO 2.0 uses unadjusted function points for sizing. Function points indicate the amount of functionality in a project and are based on the following function types:
1. External Input (inputs)
2. External Output (outputs)
3. Internal Logical File (files)
4. External Interface Files (interfaces)
5. External Inquiry (queries)
These function points are adjusted by reuse factors, cost driver effort multipliers, and exponent scale factors.

The original COCOMO is based the cost of reusing software on a linear function of the extent that the reused software needs to be modified. In this model, cost of reuse is determined from estimating the amount of software to be adapted (ASLOC), percentage of design modification (DM), the percentage of code modification (CM), and the percentage of integration effort (IM). However, reuse costs are not linear but non-linear as there is a 5% cost for assessing, selecting, and assimilating the reusable component. Furthermore, small modifications result in large costs because of the cost to understand the software to be modified and the cost of interface validation.
The COCOMO 2.0 reuse equation is defined as

\[
ESLOC = ASLOC \left[ \frac{AA + SU + 0.4(DM) + 0.3(CM) + 0.3(IM)}{100} \right]
\]

where

ESLOC = Effort in standard lines of code
ASLOC = Adapted source lines of code
AA = Assessment and assimilation (0 for none and 8 for extensive test and evaluation and documentation)
SU = Software understanding (10% if very high on structure, application clarity, and self-descriptiveness and 50% if it is very low on the latter)
DM = Percentage of design modification
CM = Percentage of code modification
IM = Percentage of integration effort

COCOMO 2.0 like intermediate and detailed COCOMO applies effort-multiplier cost drivers based on product, platform, personnel, and project factors.

1. Product Factors
   A. Required Software Reliability (RELY)
   B. Database Size (DATA)
   C. Product Complexity (CPLX)
   D. Required Reusability (RUSE)-Additional effort to construct components that are intended to be reused (1.75)
   E. Documentation (DOCU)

2. Platform Factors
   A. Execution Time Constraint (TIME) and Main Storage Constraint (STOR)
   B. Platform Volatility (VOL)
   C. Computer Turnaround Time (TURN)-Dropped in COCOMO 2.0

3. Personnel Factors
   A. Analyst Capability (ACAP) and Programmer Capability (PCAP)
   B. Application Experience (AEXP), Platform Experience (PEXP), and Language and Tool Experience (LTEX)
   C. Personnel Continuity (PCON) (3-48%)

4. Project Factors
   A. Use of Modern Programming Practices (MODP)
   B. Use of Software Tools (TOOL)
   C. Multisite Development (SITE)- New with COCOMO 2.0
   D. Required Development Schedule (SCED)
   E. Classified Security Application (SECU)-Dropped in COCOMO 2.0
Lastly, like other cost models, COCOMO 2.0 estimates development schedule as

\[
TDEV = \frac{[3.0(\text{PM})^{0.33+0.2(\text{B}-1.01)}] \text{SCED}}{100}
\]

where

- \(TDEV\) = Time in months from requirements definition to acceptance
- \(\text{PM}\) = Estimated person-months
- \(\text{SCED}\)% = Schedule compression/expansion percentage

COCOMO 2.0 gives a comprehensive model for estimating reuse development costs. However, like COCOMO this model does not account for the domain and application engineering costs associated with reusing components with black box and white box reuse as the integrated cost model does.

5.2.14 COCOTS

5.2.14.1 COTS

COTS, Commercial-Off-The-Shelf software [39], is like reuse in the following ways:

1. The same components are used in many systems
2. Components are maintained not built by developers
3. Many components are available and users are able to select what is needed
4. Cost to find, select, maintain, and integrate is less than the costs of building and maintaining the same component from scratch.

However, though COTS assets look like reusable assets and are considered reusable assets, they typically have the following drawbacks/problems:

- Are not of high quality
- Do not meet performance parameters
- Are a problem to evaluate
- Are dependent on vendor
- Possess license issues
- Are difficult to integrate
- Possess few guidelines to estimate schedule and resource requirements.

Furthermore, COTS software has the following advantages [wysiwyg]:

- Predictable license costs
- Broadly used, mature technology
- Available now
- Dedicated support organization
- Hardware/software independence
- Rich in functionality
- Frequent upgrades
The cost benefits of COTS are quantified by the savings in cost, operational quality, functionality, time-to-market, and maintenance overhead. Savings occur as a result of using a product that is produced once and used multiple times thereby allowing the user to buy the product at a much lower price than would be the case if it were developed from scratch. The multiple use allows users to share in the maintenance costs while increasing the quality of the product through the additional testing and debugging. The use of COTS products also enables the user to have additional functionality and an earlier time-to-market than custom products because products developed from scratch tend to have a lengthy development schedule. All of the latter are quantified in the benefits of the application engineering phase of the integrated cost model. The cost to acquire and integrate the COTS products is quantified in the integrated costs model’s application engineering costs. Lastly, the additional maintenance associated with COTS products is incorporated in the component engineering costs. Although the integrated model does not take COTS’ assets specifically into consideration, these types of assets can be evaluated assuming the COTS product is an application and inferring information from the vendor/programmer for the other engineering cycles.

5.2.14.2 COCOTS

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 [40], and comes in two sub-models, called Early Design and Post Architecture. These two models differ in just 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:

1. Candidate component assessment
2. Component tailoring
3. Integration/"glue code" generation
4. System level programming
5. Verification-validation.

Assessment is selecting components for use in the system to be developed. The initial filtering effort is defined as:

\[ \text{Total Effort} = (\text{# COTS candidates})(\text{Average Filtering Effort/Candidate}) \]

and the final selection effort is

\[ \text{Total Effort} = \sum (\text{# COTS Candidates})(\text{Average Assessment Effort/Candidate}) \]

The latter is summed over the seventeen attributes:

1. Correctness
2. Availability/Robustness
3. Security
4. Product Performance
5. Understandability
6. Ease of Use
7. Version Compatibility
8. Inter-Component Capability
9. Flexibility
10. Installation/Upgrade
11. Portability
12. Functionality
13. Price
14. Maturity
15. Vendor Support
16. User Training
17. Vendor Concessions

Tailoring configures the COTS program for use in a specific context.

\[
\text{Total Effort} = \sum \left( \frac{\# \text{COTS Candidates Tailored at Complexity Level}}{\text{Average Effort at Tailoring Complexity Level in Domain}} \right)
\]

The sum is over the complexity levels that are rated based on the following tailoring activities:

1. Parameter specification
2. Script Writing
3. I/O Report and GUI Screen Specification and Layout
4. Security/Access Protocol Initialization and Setup
5. Availability of COTS Tailoring Tools

The “glue code” is the code external to the COTS code that is needed to integrate the COTS code into the system being developed where

\[
\text{Total Effort} = A[(\text{size})(1+ \text{breakage})]^{p} \times (\text{Effort Multiplier})
\]

The effort multipliers and nonlinear scale factor are
- ACIEP-COTS Integrator Experience with Product
- ACIPC-COTS Integrator Personnel Capability
- AXCIP-Integrator Experience with COTS Integration Processes
- APCON-Integrator Personnel Continuity
- ACPMT-COTS Product Maturity
- ACSEW-COTS Supplier Product Extension Willingness
- ACPCPX-COTS Product Interface Complexity
- ACPPS-COTS Supplier Product Support
- ACPTD-COTS Supplier Provided Training and Documentation
- ACREL-Constraints on Application System/Subsystem Reliability
- AACPX-Application Interface Complexity
• ACPER-Constraints on COTS Technical Performance
• ASPRT-Application System Portability
• AAREN-Application Architectural Engineering

The system level programming is dependent on the number of updates/new versions of the COTS code. The latter is needed due to the volatility of incorporating COTS components.

The integrated cost model encompasses COCOTS models by making provision for COTS-based application engineering, although the integrated cost model does not estimate these cost factors but rather relies on the user or infers them from historic/statistical data.

5.2.15 Stevens

Barry Stevens [41] applied the Kang/Levy model and Gaffney/Durek Model to the Restructured Naval Tactical Data Systems (RTNDS) architecture to determine the answers to the following questions presented by Pfleeger:
1. How do we decide when the benefits of reuse outweigh the costs?
2. How do we estimate the cost of making a component reusable?
3. How do we estimate the benefits of reusing a component?

In RNTDS Kang/Levy’s central support organization is implemented with a Common Reuse Library (CRL). The only departure from this model was that the determination of the new components is done by the development project team rather than by a central control group. Applying the Gaffney and Durek model to RNTDS, b is calculated to be .0233 and E is approximately 1. Although reuse brought an increase architecture/repository cost there were equal savings in the testing phase. The n was normalized to be 11 and C was calculated to be .114. Using the Gaffney model the average cost per unit of reused software is $477.68. Stevens notes that although there are general guidelines suggested with the models, using these models is difficult without a good knowledge of the parameters and/or examples from similar projects. In addition, these models are not specifically geared for re-engineering. Based on this economic evaluation, 89-99% of the programs used in this project is reused code.
5.2.16 Raymond and Hollis

Raymond and Hollis [42] refine Gaffney’s model by
- Changing the variable conventions and documentation to improve understanding and consistency
- Changing the labor rate for the lines of code within the library for investment costs only and excluding component maintenance ($/KSLOC)
- Changing return on investment to be return on investment and library return on investment.

This model is designed for government developers because it assumes that the fixed costs for the creation of new programs is zero since the fixed cost cannot be saved because the personnel, equipment, and facilities cannot be discarded. This model assumes that the cost per KSLOC of reuse code includes the cost to modify the component as well as to integrate the component into the application and library system. To spread the cost of the library across the spectrum of users, the fixed cost of the reuse lines of code is the development cost multiplied by the size of the library and divided by the number of times the library has been accessed. Thus the total cost of the application system is

\[ C_t = C_{vn}(S_n) + C_{vr}(S_r) + C_{fr} \]

where
- \( S_n \) = Size of total new code
- \( C_{vn} \) = Variable cost of writing new code
- \( C_{vr} \) = Variable cost of modifying reused code
- \( S_r \) = Size of the total modified code
- \( C_{fr} \) = Fixed costs of developing reuse code in a library

Raymond and Hollis go on to define the breakeven point, the productivity, and the return on investment. The breakeven point is defined as the point at which the relative cost reduction is 1 and productivity is the inverse of the relative cost reduction. Relative Cost Reduction (Co) is the cost of an application divided by the cost of all new tailored code:

\[ N_o = \frac{C_d}{C_{vn} - C_{vr}} \]

where
- \( N_o \) = Breakeven point
- \( C_d \) = Cost of developing a reuse library
- \( E \) = Library efficiency
- \( C_{vn} \) = Variable cost of writing new code
- \( C_{vr} \) = Variable cost of modifying reused code
Return on Investment (ROI) is defined as the total cost of the application without reuse divided by the total cost of the application with reuse:

\[ I_r = \text{ROI} = \frac{S_t}{C_{vn}(S_n) + C_{vr}(S_r) + C_{fr}} \]

where

- \( C_{vn} \) = Variable cost of writing new code
- \( C_{vr} \) = Variable cost of modifying reused code
- \( S_t \) = Total size of the application
- \( C_{fr} \) = Fixed cost of developing reuse assets in a library which is defined as

\[ C_{fr} = \frac{C_d S_l}{N} \]

where

- \( n \) = Number of uses
- \( S_l \) = Size of the library.
- \( C_d \) = Cost of reusing code

If ROI < 0, then the manager is spending more money on reuse than on new code.

The efficiency of the library is defined as quotient of \( S_r \) and \( S_l \). Raymond and Hollis then proceed to define the breakeven point and the return on investment on the library. The integrated cost model, like Raymond and Hollis’s, accounts for the application costs from building from reusable components and new code and calculates the ROI and breakeven point like this model does.

5.2.17 Gaffney and Cruickshank

The model proposed by Gaffney and Cruickshank [43,44] 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.

\[ C_s = C_{us} S_s = \frac{C_{de} S_l}{N} + C_{vn} S_n + C_{vr} S_r \]

where

- \( C_{us} \) = Unit cost of the application system
- \( S_s \) = Total Size of the application system in source statements
- \( C_{de} \) = Unit cost of domain engineering
- \( C_{vn} \) = Unit cost of new code developed for this application system
- \( C_{vr} \) = Unit cost of reusing code from the reuse library in this application system with black box reuse.
- \( S_l \) = Expected value of the unduplicated size of the reuse library
- \( S_n \) = Amount of new code in source statements developed for this application
- \( S_r \) = Amount of reuse incorporated in this application in source statements
Unlike the integrated cost model, it does not provide for pricing structure between the domain engineering team and the 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. These authors also discuss breakeven point and return on investment for up-front domain engineering and incremental engineering. In the case of up-front domain engineering, the breakeven point is defined as

\[ N_0 = \frac{C_{de}}{(C_{vn} - C_{vr})E} \]

where
- \( E \) = Efficiency factor
- \( C_{de} \) = Unit cost of domain engineering
- \( C_{vn} \) = Unit cost of new code developed for this application system
- \( C_{vr} \) = Unit cost of reusing code from the reuse library in this application

and the return on investment is defined as

\[ \text{ROI} = \left[ \frac{N(E)(C_{vn} - C_{vr})}{C_{de}} - 1 \right] \times 100 \]

where
- \( N \) = Number of applications
- \( E \) = Efficiency factor
- \( C_{de} \) = Unit cost of domain engineering
- \( C_{vn} \) = Unit cost of new code developed for this application system
- \( C_{vr} \) = Unit cost of reusing code from the reuse library in this application

5.2.18 Poulin

Poulin [45] in *The Economics of Software Product Lines* compensates for the incompleteness of cost-benefit methods’ inability to quantify and gather every cost-benefit factor. Using Relative Cost of Reuse (RCR) and Relative Cost of Writing for Reuse (RCWR), ROI can be defined. RCR is the ratio of the effort it takes to reuse software without modification to developing from scratch, and RCWR is the ratio of the effort it takes to develop reusable software to developing it from scratch. RCR is typically .2 and RCWR is typically 1.5 and is based on environmental factors like experience, reuse organization, and software complexity. Project ROI is then defined as

\[ \text{ROI} = \sum_{i=1}^{n} \text{RCA}_i - \text{ADC} \]

where
- \( \text{RCA} \) = Reuse Cost Avoidance = DCA + SCA
where

\[
\begin{align*}
\text{DCA} &= \text{Development Cost Avoidance} = \text{RSI}(1-\text{RCR})(\text{New Code Cost}) \\
\text{SCA} &= \text{Service Cost Avoidance} = \text{RSI}(\text{Error Rate})(\text{Error Cost}) \\
\text{ADC} &= \text{Additional Development Costs} \\
&= (\text{RCWR}-1) (\text{Code written for reuse by others}) (\text{New Code Cost})
\end{align*}
\]

Poulin [44] also defines the payoff threshold, the number of times a component must be reused to recover the investment made to develop the component or breakeven point:

\[
n_0 = \frac{\text{RCWR}}{1 - \text{RCR}}
\]

and using the defaults of .2 for RCR and 1.5 for RCWR, \(n_0\) is 1.88. The above applies to domain-specific software architectures.

Poulin’s model is designed for 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 (1) development cost avoidance, (2) service (maintenance) cost avoidance, (3) payoff threshold (the minimal number of reuse instances required to break even), and (4) the costs associated with domain engineering. Poulin uses these cost factors to derive a return on investment formula for product line engineering.

The integrated cost model, like Poulin’s, recognizes that the cost associated with writing for reuse must be accounted for. The integrated cost model accounts for writing for reuse in the application engineering cycle when the reuse adoption costs and reuse development costs are accounted for in the cost to build the application. The integrated cost model extends Poulin’s model by demonstrating the benefits using black box and white box components in the component engineering cycle. Unlike Poulin’s, the integrated model also accounts for the domain analysis costs that are incurred in writing and using reusable components.

### 5.2.19 Lim

Lim’s paper [46], *Reuse Economics: A Comparison of Seventeen Models and Directions for Future Reuse*, characterizes the above models based on a lexicon of common data elements (operands in the models) and presents a matrix of identifying these data elements. Lim describes each of the above models, translates the model using a common lexicon, and maps the data elements of the lexicon to each model. In this paper, risk is defined as the likelihood that cash-flows in a given year will occur, and overhead is defined as the costs associated with the reuse library, archiving, training, and management. Based on the data elements, Lim found most of the models incorporated (1) the cost to the producer to create the asset for reuse, (2) the number of times the
reusable asset is reused, (3) the cost to create the asset without reuse, (4) the cost to consumer to reuse the asset, and (5) the cost to consumer to create a non-reusable version of the asset. Of the 17 models, 41% outputted savings from reuse, 23% outputted ROI, 18% showed a relative comparison, and 9% indicated the number of reuses to break even. Most of these models (65%) do not indicate whether the producer or the consumer bears the cost of producing the reusable asset and (71%) do not account for the time value of money, the increased profit from shortened time-to-market (88%), the maintenance phase (71%), and the overhead costs of reuse (59%) (reuse library, architecture, training, management, etc.). Lim recommended that future research needs to develop better data collection methods to identify cost drivers and cost savings and enable application of the economic models and return on investment models.

Lim presents a survey of the existing cost models and aims to compare these models emphasizing the pros and cons of each model. The integrated cost model on the other hand aims to be a comprehensive model that encompasses existing models. The integrated cost model other goal is to identify the major costs drivers: investment length, investment rate, benefits, and costs of the four engineering cycles (component, domain, application, and corporation) and also to enable the user to make sound economic decisions based on the application of one economic model. This economic model in turn presents the user with five economic functions from which the user can base his/her decision: NPV, ARBV, Payback, ROI, and IRR.

### 5.2.20 Frakes and Terry

Frakes and Terry [47] define the reuse threshold level both internal and external, the reuse frequency, and size weighting. The internal threshold level is the maximum number of uses of an internal item that can occur before reuse occurs, and the external threshold level is the maximum number of uses of an external item that can occur before reuse occurs. The total reuse level is defined as

\[ TRL = \frac{IU}{T} + \frac{EU}{T} \]

where

- \( IU \) = Number of internal lower order items which are used more than ITL
- \( EU \) = Number of external lower order items which are used more than ETL
- \( ITL \) = Internal threshold level (maximum internal uses before it is considered reused)
- \( ETL \) = External threshold level (maximum external users before it is considered reused)

Reuse frequency is referencing components, and size weighting is the complexity of the reused component. Using multi-call graph abstraction, these authors found that internal reuse increases as programs increase in size while external reuse decreases. Furthermore, increases in the threshold levels, both internal and external, decrease the reuse level. 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 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.

5.2.21 Devanbu

Devanbu, et al. [48], 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 that 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.

These authors’ cost model evaluates reuse benefits both analytically and empirically. Analytically they measure the time, money and quality benefits due to reuse. Empirically, they use a GEN++ toolset that is based on C++ code and student data from student projects from the University of Maryland to validate their analytical measures.

Reuse increases productivity eliminating re-implementation of the same asset/product, increased quality due to testing, and avoids maintenance due to the increase use and testing. The reuse benefit resulting from the latter is the financial gain due to reuse

Reuse benefit is defined as

\[
R(S) = \frac{\text{Cost of developing } S \text{ without reuse} - \text{Cost of developing } S \text{ with reuse}}{\text{Cost of } S \text{ without reuse}}
\]

Reuse benefit differs with different implementations because the different implementations can increase or decrease reuse while maintaining functionality. This model is also sensitive to the cost of the components being reused. Furthermore this model recognizes that reusing external components is better than reusing internal components and that incorporating new code does not increase the benefit. Lastly, verbatim reuse is better than adapted reuse. The integrated model also recognizes that the reuse benefit is the savings of not having to produce the product from scratch.
These authors quantify productivity as the size of the system divided by the cost spent to develop it:

$$II = a(1+R)^b$$

where
- $R$ = Reuse benefit
- $a, b$ = Coefficients estimated with standard least squares regression
- $II$ = Productivity

When there is no reuse, productivity is the coefficient $a$. They quantify quality as the measure of the fault and error density. The integrated cost model quantifies productivity and quality based on the savings in development from a decreased error rate.

### 5.2.22 Mili

Mili [49] 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. Mili’s definition of return on investment is dependent on whether the reuse lifecycle is synchronous or asynchronous. In the synchronous reuse lifecycle, the components are evaluated to determine if they should be stored in a reuse repository. Return on investment is defined as

$$\text{ROI} = \frac{1}{IC} \left( \sum_{y=1}^{Y} \frac{OC(y)}{(1+d)^y} - IC \right)$$

where
- $IC$ = Investment cost differential
- $OC$ = Operating cost differential
- $d$ = Discount rate
- $Y$ = Investment Period

In the asynchronous reuse process models’ return on investment, the manager determines whether to develop a component for reuse or not. The return on investment is defined as

$$\text{ROI} = \frac{1}{IC} \left( \sum_{y=1}^{Y} \frac{OC(y)}{(1+d)^y} - DR \right)$$

where
- $DR$ = Cost of developing component $c$ for reuse
- $OC$ = Operating cost differential = Store+Retrieve+Maintain
- $d$ = Discount rate
- $Y$ = Investment Period
This model focuses on component level costs and balances them against potential domain-wide benefits. This paper’s component level ROI cycle is similar to this model.

5.2.23 Hancock

In Hancock’s model [9], the present value is calculated as follows:

\[
PV = \sum_{t=0}^{n} \frac{B_t - C_t}{(1 + d)^t}
\]

where
- \(B_t\) = Benefits in year \(t\)
- \(C_t\) = Costs in year \(t\)
- \(d\) = Discount rate

The integrated cost model is also based on PV.

5.2.24 Favaro

Favaro [17] does not present an economics model but discusses the following investment functions:

- Net Present Value (NPV)
  \[
  NPV = C_0 + PV
  \]
  where
  - \(C_0\) = Initial Investment
  - \(PV\) = Present Value defined as
    \[
    \sum \frac{C_t}{(1 + k_t)^t}
    \]
    where
    - \(C_t\) = Future cash flow in period \(t\)
    - \(k_t\) = Discount rate in period \(t\)

- Payback
  \[
  N_0 = \frac{E}{(1 - b)}
  \]
  where
  - \(N_0\) = Number of times a component must be used before its cost is recovered
  - \(b\) = Relative costs of integrating the component
  - \(E\) = Relative costs of developing a component for reuse
- **Average Return on Book Value (ARBV)**
  Divide the average profits from predicted cash flows by the average net book value of the investment
- **Internal Rate of Return (IRR)**
  Discount rate that makes NPV = 0
- **Profitability Index (PI)**
  \[
  \text{PI} = \frac{\text{PV}}{C_0}
  \]

Favaro does not get into specifics but focuses on component engineering from the viewpoint of a corporate manager. He summarizes his findings as follows (Table 5):

<table>
<thead>
<tr>
<th>Approach/Rule</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net Present Value (NPV)</td>
<td>Most acceptable, realistic approach</td>
</tr>
<tr>
<td>Accept if NPV &gt; 0</td>
<td>Values are additive allowing project combinations to be evaluated</td>
</tr>
<tr>
<td></td>
<td>Takes differences of scale into consideration</td>
</tr>
<tr>
<td>Payback</td>
<td>Usually does not discount cash flows</td>
</tr>
<tr>
<td>Accept if payback within some specified target time period</td>
<td>Arbitrary cutoff dates for payback</td>
</tr>
<tr>
<td></td>
<td>Ignores cash flows after cutoff date</td>
</tr>
<tr>
<td></td>
<td>Not sensitive to scale</td>
</tr>
<tr>
<td>Average Return on Book Value</td>
<td>Insensitive to cash flow pattern</td>
</tr>
<tr>
<td>Accept if predicted rate of return greater than a predetermined target</td>
<td>Dependent on accounting practices</td>
</tr>
<tr>
<td></td>
<td>Arbitrary targets</td>
</tr>
<tr>
<td>Internal Rate of Return (IRR)</td>
<td>Based upon discounted cash flows</td>
</tr>
<tr>
<td>Accept if IRR is greater than the opportunity cost of the project</td>
<td>No direct economic significance</td>
</tr>
<tr>
<td></td>
<td>Subject to math anomalies</td>
</tr>
<tr>
<td></td>
<td>Not sensitive to scale</td>
</tr>
<tr>
<td>Profitability Index (PI)</td>
<td>Conceptually closest to NPV</td>
</tr>
<tr>
<td>Accept if PI&gt;1</td>
<td>Values not additive</td>
</tr>
<tr>
<td></td>
<td>Not sensitive to scale</td>
</tr>
</tbody>
</table>

**Table 5 Economic Functions**

As a consequence of this research, the integrated cost model evaluates these economic functions for each of the engineering cycles: component, domain, application and corporate. Like Favaro, the integrated cost model focuses on NPV due to its advantage over the other economic functions.
5.3 Conclusion

This chapter attempts to classify existing reuse models by means of the six-dimensional structure that includes: investment cycles, economic functions, costs factors, reuse organization, scope, and hypothesis. This six-dimensional structure seems to adequately characterize all of the cost model that have been analyzed. Also, the integrated cost model appears to encompass most of the aspects covered by these 24 existing models.
Chapter 6. Quantifying Software Reuse Costs and Benefits
The benefits of software reuse do not come without its associated costs. Reuse results in
the additional costs incurred by the producer to make a generalized, adaptable, quality
component that is well documented and tested. Even re-engineered components must be
understood before adapted and tested. Consumers incur the cost of selecting, adapting,
integrating and testing this reusable component but save by not having to redevelop each
component. Thus, a producer assumes a higher cost to design and implement a reusable
asset than a custom developed asset, but the consumer saves time and money by not
having to develop the component. [35]

The three key benefits of software reuse are higher productivity, better quality, and
shortened time-to-market. These benefits are the periodic benefits of the application
engineering cycle which in turn are the benefits (savings) of the corporate engineering
cycle. In the subsequent sections, the quantification of these key benefits will be
detailed.

6.1 Productivity Gains

Productivity gains are the gains in development costs that result from reuse. Productivity
gains are accounted for in the application engineering cycle and propagated from there to
the corporate engineering cycle. These gains are a function of the volume of code
developed from scratch, white box reuse, and black box reuse.

If a component were to be developed from scratch it would cost \( E \) (person-months) to
develop. To produce this same component using black box reuse, \( E \) would be multiplied
by the Relative Cost of Reuse (RCR). Studies [5,16-17,45-46] have found that RCR
between 0.03 and 0.4; as a synthesis of these studies, the default value of this factor is
taken as 0.2 [45]. Thus, the productivity gains for black box reuse would be

\[
PG_{BB} = E - (E(RCR))
\]

To produce this same component using white box reuse, \( E \) would be multiplied by the
Relative Cost of Adaptation, \( RCA \), typically 0.67. Thus the productivity gains for white
box reuse would be

\[
PG_{WB} = E - (E(RCA))
\]

To determine the overall productivity gains of the application, the sum of the productivity
gains of the components comprising the application would have to be calculated and
multiplied by the average salary of a developer for a month (Pay):

\[
PG = \left( \sum_{i=1}^{n} (PG_{WB_i} + PG_{BB_i}) \right)(Pay)
\]
6.2 Quality Gains

Quality gains achieved from reuse results from the savings in operating costs of reusable assets over the investment cycle lifetime. The latter is quantified with the Operating Cost Differential (OCD). If a component is developed for reuse, especially black box reuse, then it is of better quality than a single use component since it is tested more frequently, documented clearer and designed with reuse in mind. OCD is the difference between the cost of operating a high quality component versus a low quality component. The latter is estimated using the COCOMO and COCOMO 2.0 formula for maintenance costs, Annual Change Traffic (ACT).

Typically the annual maintenance cost (ACT) is 15% of development without reuse for a single use component. For the sake of argument, a good quality component that is well documented, structured and possessing few bugs is assumed to cost 40% less to operate/maintain than that of a poor quality component with the same functionality. Thus OCD for a black box component is defined as

\[
OCD = E(ACT)(0.6) = E(0.15)(0.6) = 0.9E
\]

where
- \(E\) = Effort in person-months (PM)
- \(ACT\) = Annual Change Traffic

For white box reuse, the cost of maintaining and operating a modified reusable component versus the cost of maintaining a component built from scratch is much higher (~65%). Thus OCD, is much smaller. For example, assuming 65% is the cost of maintaining a white box component then

\[
OCD = E(ACT)(0.35) = E(0.15)(0.35) = 0.42E
\]

Thus, the quality of a black box component is much better than that of a white box component because the white box component requires modification and adaptation that invalidates the quality assurance testing/validation of the original unmodified component.
Using the OCD’s from the black and white box components that comprise the application, the overall quality gains (in person-months) of the application/project that is built with reuse can be obtained at year \( y \) as

\[
QG(y) = \sum_{i=1}^{n} nb(y) \frac{(OCD_i)}{(1 + d)^y}(Pay)
\]

where
- \( QG \) = Quality Gains of an application
- \( nb(y) \) = Number of instances of an application in a given year \( y \)
- \( OCD \) = Operating Cost Differential as defined above
- \( d \) = Discount rate
- \( y \) = Year
- \( n \) = Number of components
- \( Pay \) = Average monthly salary of a developer

Thus the total quality gains resulting from using reusable assets is

\[
TQG = \sum_{y=1}^{Y} QG(y)
\]

where
- \( TQG \) = Total quality gains
- \( QG \) = Quality Gains at year \( y \)
- \( Y \) = Length of the investment

### 6.3 Shortened Time-to-Market

One goal of a producer is to have a shortened time-to-market of an application. Getting a reusable application to market early generates additional revenue but not without the additional production costs (both fixed and variable). Thus, the benefit of this early release of the application is not just the additional revenue generated in that earlier time period but the present value of the difference of the sales in this time period.

In the past, authors like Lim [33] and Malan [35] recognize the need to incorporate shortened time-to-market into the quantification of the costs and benefits of reuse but never formally quantified this benefit. Consequently, this shortcoming is addressed in the integrated cost model by the introduction of the concept of the Reuse Cost Index (RCI).

RCI is a ratio that indicates the benefit of a shortened time-to-market that results from building an application from reusable assets. This index indicates not only the reduction in time to build the application but also the increase cost benefit that results from this shortened application development cycle. This index assumes that the production and purchasing of the reusable assets is internal to the corporation.
For a long term project (greater than 1 year), the Net Present Value (NPV) function at the application engineering level is used to calculate RCI:

\[
RCL = \frac{\text{NPV}_s}{\text{NPV}_r}
\]

where
- \( RCL = \text{Reuse Cost Index} \)
- \( \text{NPV}_s = \text{Net Present Value of the application developed from Scratch} \)
- \( \text{NPV}_r = \text{Net Present Value of the application developed with Reuse} \)

Net Present Value takes into account the additional resources needed to produce the application in the initial costs while also accounting for the additional revenue generated by the additional uses of the application. Furthermore, these additional uses of the application increase the quality of the application by means of additional use and testing of the application.

For a short-term project (less than 1 year), Net Profitability at the application engineering level is used to calculate RCI:

\[
RCL = \frac{\sum_{t=0}^{T_s} (B_s(t) - C_s(t))}{\sum_{t=0}^{T_r} (B_r(t) - C_r(t))}
\]

where
- \( RCL = \text{Reuse Cost Index} \)
- \( B_s(t) = \text{Benefits from building from Scratch} \)
- \( C_s(t) = \text{Costs to build from Scratch} \)
- \( B_r(t) = \text{Benefits from building with Reuse} \)
- \( C_r(t) = \text{Costs from building with Reuse} \)
- \( T_s = \text{Time to complete the project (T \leq 1 year) from scratch} \)
- \( T_r = \text{Time to complete the project (T \leq 1 year) with reuse} \)
- \( t = \text{Interval of total time T (months, weeks, days)} \)

Building a application from scratch will result in higher costs and lower revenue while building from reuse results in greater benefit due to earlier time-to-market and less cost due to the need for fewer programmers.

For example, in a long term project, if the NPV is 1 for an application developed from scratch and if the NPV is 2 for an application developed with reusable components, then it will take half the time to develop an application using reuse than it is to develop it from scratch. Furthermore, the benefit from building the application with reusable assets is two times the benefit of developing it from scratch. In a short term project, if it takes...
four programmers six weeks (5.6PM) at a salary of $4K/month to develop an application from scratch ($22400) and it takes two programmers three weeks (1.4PM) at a salary of $4K/month to build this same application using reusable assets ($5600), then the application is on the market eighteen weeks earlier. Between 7-24 weeks, the application is used five times and from 24-36 weeks, the application is used another five times. If we assume the cost of the asset is .5PM with an average salary of $4K/month then the benefit from resale of the application built from reusable components is 10(.5PM)($4K)=$20000 and the benefit from the application built from scratch is 5(.5PM)($4K)=$10000. Thus,

$$\text{RCI} = \frac{|$10000 - $22400|}{|$20000 - $5600|} = .86$$

Thus, the benefit from the earlier marketing of the application built with reuse is 86%. Note that this does not reflect gains in terms of market share.

### 6.4 Conclusion

In the Integrated Cost Model, the benefits of application engineering are quality gains and productivity gains. As shown above, these gains are incorporated in the calculation of B(y) of the application and corporation. B(y) is then applied to the economic functions (NPV, PI, ROI, ARBV, and PB) to enable managers to see the benefits of reuse. Shortened time-to-market although not explicitly denoted in the integrated cost model, is implicitly incorporated in the calculation of NPV.
Chapter 7. Component Engineering Cycle
In the component engineering cycle the investment decision is to determine if a reusable component is worthy of being developed and then stored in a software library for the purpose of subsequent reuse. This decision to use reusable components is based on the requirements of the component, the development costs, the savings gained by using the reusable component, and the need for the component in future projects.

### 7.1 Investment Costs

The up-front costs of component engineering include the costs of developing the asset for reuse, the costs of reuse certification, and the library insertion costs:

\[ IC = ER + LI \]

where

- \( ER = \) Cost of development for reuse
- \( LI = \) Cost of certification and Library Insertion

The library insertion costs can range from trivial (the cost of adding an entry to the library) to complex (incorporating library-specific quality standards, pre-insertion domain analysis, etc.).

\( ER \), if not available, can be estimated by prorating the costs of development from scratch by the Relative Cost of Writing for Reuse (RCWR):

\[ ER = E(\text{RCWR})(\text{Pay}) \]

where

- \( E = \) Cost of developing an equivalent system from scratch in person-months
- \( \text{RCWR} = \) Relative Cost of Writing for Reuse (Table 6) [3]
- \( \text{Pay} = \) Average monthly salary of the developer

For the sake of simplicity, the Integrated Cost Model assumes a default of 1.50 for \( \text{RCWR} \).

If \( E \) is not available, it can be derived from Boehme’s basic COCOMO model in semi-detached development mode:

\[ E = 3S^{1.12} \]

where

- \( E = \) Cost of development of the component from scratch without reuse
- \( S = \) Estimated product size in thousands of lines of code
Table 6 RCWR Constants

<table>
<thead>
<tr>
<th>Source</th>
<th>RCWR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bardo</td>
<td>1.15-1.25</td>
</tr>
<tr>
<td>Reifer</td>
<td>1.10-1.36</td>
</tr>
<tr>
<td>Cadwell</td>
<td>1.25-1.30</td>
</tr>
<tr>
<td>Lim</td>
<td>1.11-1.80</td>
</tr>
<tr>
<td>Gaffney and Cruickshank</td>
<td>1.50</td>
</tr>
<tr>
<td>Poulin</td>
<td>1.50</td>
</tr>
<tr>
<td>Jones</td>
<td>1.50</td>
</tr>
<tr>
<td>Pant</td>
<td>1.55</td>
</tr>
<tr>
<td>Favaro</td>
<td>1.00-2.20</td>
</tr>
<tr>
<td>Tracz</td>
<td>1.60</td>
</tr>
<tr>
<td>IBM</td>
<td>1.25-2.00</td>
</tr>
<tr>
<td>Lockheed Martin</td>
<td>1.86</td>
</tr>
<tr>
<td>Margano</td>
<td>2.00</td>
</tr>
</tbody>
</table>

7.2 Episodic Costs

The episodic costs are based on the cost of operating the reuse library and the maintenance costs of the component. At year $y$ where $SD+1 \leq y \leq SD+Y$,

\[ C_c(y) = OC(y)(Pay_d) + MN(y)(Pay_d) \]

where
- $C_c(y)$ = Periodic Cost at year $y$ of a component
- $OC(y)$ = Operating Cost at year $y$
- $MN(y)$ = Maintenance Costs
- $Pay_d$ = Average monthly salary of the developer
- $Pay_l$ = Average monthly salary of the developer

The operating cost is the cost of operating the reuse library prorated to a single component. The operating costs can be derived from dividing the labor costs of operating the library by the size of the library. For example, if it takes one full-time person to operate the library and there are 240 components in the library, then

\[ OC(y) = \frac{12}{240} = .05 \text{ PM} \]
This assumes that there will be no increase in the operating costs as the library grows and is a simplistic way of deriving the operating costs.

The maintenance costs can be derived from the COCOMO maintenance-effort equations. This maintenance activity is applied to the costs via the Annual Change Traffic (ACT) factor. ACT is the ratio of the yearly maintenance cost to the development cost and is typically around 0.15 [18] since annual maintenance is 15% of the cost of development from scratch (single use).

\[ MN(y) = E(ACT) \]

where
\[ MN(y) = \text{Maintenance Costs} \]
\[ E = \text{Development Cost (PM)} \]

### 7.3 Episodic Benefits

The episodic benefits of component level reuse are gains in productivity and quality. Gains in productivity are derived from subtracting reuse costs from development of a component from scratch. As discussed above in the investment costs, the costs of development of a component from scratch can be derived from the COCOMO model based on the size of the component. Reuse costs can be derived from the many models present in literature that were outlined in Chapter 5. Gains in quality are derived from the maintenance savings over the years, i.e., the better the quality of the product the less maintenance and more use of the product. Thus, the benefits at SD+1 ≤ y ≤ SD+Y

\[ B_c(y) = \text{freq}_{BB}(y)(BP(y)) + \text{freq}_{WB}(y)(WP(y)) \]

where
\[ B_c(y) = \text{Episodic benefits of component engineering} \]
\[ \text{freq}_{BB}(y) = \text{Frequency of black box reuse of the component at year y} \]
\[ \text{freq}_{WB}(y) = \text{Frequency of the white box reuse of the component at year y} \]
\[ BP(y) = \text{Black box price of the component} \]
\[ WP(y) = \text{White box price of the component} \]

The black box price and the white box selling price can be derived from the Relative Black Box (RBP) and Relative White Box (RWP) price ratios. These component costs must take into account the cost to retrieve the asset (RET), and the cost of instantiation (INST) (the cost to determine if the component is a good “fit”) or the cost of adaptation (ADP).

For black box component level reuse to be worthwhile, the selling price of the component (BP), retrieval costs (RET), and instantiation costs (INST) must be less than the cost to develop from scratch (E):
\[ \text{RET} + \text{INST} + \text{BP} < E \]
where
- \text{RET} = \text{Cost to retrieve the asset}
- \text{INST} = \text{Cost to instantiate the component}
- \text{BP} = \text{Black box component sale price}
- \text{E} = \text{Cost to develop from scratch without reuse}

Since instantiation costs are approximately 20\% of custom development costs

\[
\begin{align*}
\text{RET} + \text{BP} + \text{RCR}(E) &< E \\
\text{RET} + \text{BP} + 0.2E &< E \\
\text{RET} + \text{BP} &< 0.8E \\
\end{align*}
\]

where
- \text{RET} = \text{Cost to retrieve the asset}
- \text{RCR} = \text{Relative Cost of Reuse}
- \text{BP} = \text{Black box component sale price}
- \text{E} = \text{Cost to develop from scratch without reuse}

Furthermore, if retrieval costs are neglected which are low for good, well documented repositories and are usually much smaller than the other terms of the equation,

\[ \text{BP} < 0.8E \]

The relative black box price (RBP) is the ratio of the black box price to the cost of developing the component from scratch or

\[ RBP = \frac{\text{BP}}{E} < 0.8 \]

Since the domain engineer must fix the price of the asset so as to maximize his benefit per sale, while making sure the acquisition of the asset is still attractive to the application engineer (otherwise the application engineering will develop his own). Thus, the savings can be divided in half between the application engineer and the domain engineer and RBP is taken as 0.4 or if the domain engineer expects to make more than one sale, the domain engineer can be given less than half or \( RBP = 0.3 \). In the integrated cost model, a default of 0.4 is taken for RBP.

For white box component level reuse to be worthwhile, the selling price of the component (WP), retrieval costs (RET), and adaptation costs (ADAPT) must be less than the cost to develop from scratch (E):

\[ \text{RET} + \text{ADP} + \text{WP} < E \]
where
- \( \text{RET} \) = Cost to retrieve the asset
- \( \text{ADP} \) = Cost to adapt the component
- \( \text{WP} \) = White box component sale price
- \( \text{E} \) = Cost to develop from scratch without reuse

Since typical adaptation costs (RWP) are 67% of custom development costs

\[
\text{RET} + \text{WP} + \text{RWP}(\text{E}) < \text{E} \\
\text{RET} + \text{WP} + 0.67\text{E} < \text{E} \\
\text{RET} + \text{WP} < 0.33\text{E}
\]

As in black box reuse, the retrieval costs in white box reuse are neglected so the white box price selling price (RWP) is

\[
\text{RWP} = \frac{\text{WP}}{\text{E}} < 0.33\text{E}
\]

A default value of 0.15 is taken for RWP which corresponds to a fair decision of the benefits of white box reuse between the producer and the consumer.

### 7.4 Summary

<table>
<thead>
<tr>
<th>Year ( y )</th>
<th>Cost ( C(y) )</th>
<th>Benefit ( B(y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>Development for Reuse</td>
<td></td>
</tr>
<tr>
<td>&gt;SD</td>
<td>Residence Costs</td>
<td>Sales to Projects</td>
</tr>
<tr>
<td></td>
<td>Maintenance Costs</td>
<td></td>
</tr>
</tbody>
</table>

Thus the initial costs of components is the development for reuse. In subsequent years the costs consist of maintenance and repository residence costs. Furthermore, in the initial year little to no benefits are seen but in subsequent years, the producer gains the benefits of the sales of these components to other projects.

### 7.5 Example of Component Engineering Costs and Benefits

Consider a component of size 5 KLOC that is developed by a programmer earning $60K/year and maintained in a library by a librarian earning $48K/year to maintain 240 components. In this library, library maintenance is 10% of the development costs of this component from scratch and library insertion is 10% of the development costs from scratch.
The Development Cost from Scratch using the Basic COCOMO in organic mode:

\[ E = 3S^{1.12} = 3(5^{1.12}) = 18.196 = 18.2PM \]

The Development Cost for Reuse

\[ ER = (RCWR)(E)(Pay) = 1.5(18.2)(\frac{60K}{12}) = \$136,500 \]

The Library Insertion Cost

\[ LI = 0.1E(Pay) = (1.2)(18.2)(\frac{48K}{12}) = \$7,280 \]

The Initial Cost or Cost at \( y=SD \)

\[ IC = ER + LI = \$136,500 + \$7,280 = \$143,780 \]

The Maintenance Cost of this component at year \( y>SD \)

\[ MN(y) = 0.1E = .1(18.2) = 1.82PM \]

The Operating Cost of Maintaining this Component in the Library at year \( y>SD \)

\[ OC(y) = \frac{12}{240} = 0.05PM \]

Thus, the cost associated with this component each year after the start date, \( y>SD \), is

\[ C_c(y) = OC(y)(Pay) + MN(y)(Pay) \]

\[ C_c(y) = (0.05PM)(\frac{48K}{12}) + (1.82PM)(\frac{60K}{12}) \]

\[ C_c(y) = \$200 + \$9,100 = \$9,300 \]

Finally the benefits of this reusable component after the start date, \( y>SD \), is

\[ B_c(y) = (freqb(y))(BP(y)) + (freqw(y))(WP(y)) \]

\[ BP(y) = .4E(Pay) = .4(18.2)(\frac{60K}{12}) = \$36,400 \]

\[ WP(y) = .15E(Pay) = .15(18.2)(\frac{60K}{12}) = \$13,650 \]
To be fair to the application and domain engineer, RBP is 0.4 and RWP is 0.15.

<table>
<thead>
<tr>
<th>Year</th>
<th>Freq_{bb}</th>
<th>Freq_{wb}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1999</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Using the above frequencies and white box and black box prices, the benefits from component reuse are calculated as follows:

\[ B_c(y) = (\text{freq}_{bb}(y))(\text{BP}(y)) + (\text{freq}_{wb}(y))(\text{WP}(y)) \]

\[ B_c(1997) = 0 \]
\[ B_c(1998) = 1(36,400) + 2(13,650) = $63,700 \]
\[ B_c(1999) = 2(36,400) + 1(13,650) = $86,450 \]
\[ B_c(2000) = 0 \]

Thus the benefits and costs for this reusable component for each year are:

<table>
<thead>
<tr>
<th>Year</th>
<th>C(Year)</th>
<th>B(Year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>$143,780.00</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>$9,300.00</td>
<td>$63,700.00</td>
</tr>
<tr>
<td>1999</td>
<td>$9,300.00</td>
<td>$86,450.00</td>
</tr>
<tr>
<td>2000</td>
<td>$9,300.00</td>
<td>0</td>
</tr>
</tbody>
</table>

If this component was not developed for reuse it would have cost the company $91,000. However, although the component was developed for reuse and cost $143,780 the first year, the benefits in the subsequent years from reuse would pay for the development of the component.
Chapter 8. Application Engineering Cycle
In the application engineering cycle, the project manager determines the worthiness of applying reuse technology to a project. To this end, the manager must determine if there is a match between the project’s requirements and the available reusable assets and how much reuse adaptation/instantiation overhead is involved in using these reusable assets. The overall cost of developing an application is:

\[
AP = E + BP + WP + CI + O
\]

where

- \( E \) = Cost to develop the components/“glue code” of the application from scratch
- \( BP \) = Cost of acquiring black box assets for the application
- \( WP \) = Cost of acquiring white box assets for the application
- \( CI \) = Cost of integrating the components
- \( O \) = Overhead of using reuse
- \( AP \) = Application development cost

At this decision level the following assumptions are made:

1. The application development project takes place within a year. Thus, all development costs are modeled as up-front costs (episodic costs at year SD) and do not need to take into account the time value of money.

2. For the costs of application development with reuse to balance against the costs of developing the same application from scratch, the costs savings achieved by using reusable assets rather than writing customized code from scratch is derived as a benefit at year SD.

3. Since all purchasing and selling of the assets is within the corporation, the benefit of selling the asset (in domain engineering) is synonymous to the purchasing of the asset (in application engineering). Thus, \( C(y) = 0 \) for application engineering and \( B(y) = 0 \) for domain engineering for this balancing of purchasing and selling the reusable assets to be possible at \( y > SD \).

### 8.1 Up-front Costs

The up-front costs at the application engineering level include the costs of training, of tool acquisition, and of the following operational considerations:

- Components can be retrieved without being relevant, especially if the library’s retrieval procedure is inadequate.
- Relevant components can be retrieved but prove to be too costly to adapt and integrate.

As a result of the assumption that application development takes place within the year, the up-front costs are the costs of acquiring the reusable assets. The cost of acquiring the reusable assets is modeled as the sum of the prices of the individual components in the application. Thus,
IC = \sum_{i=1}^{NC} PR_i

where
IC=Initial cost  
NC=Number of components  
PR=Price of the component used as black box (BP) or white box (WP)

8.2 Periodic Costs

Since the purchasing and selling of the components is within the corporation, the episodic costs of application engineering is zero for years subsequent to the start date (SD).

\[ C_a (y) = 0, SD + 1 \leq y \leq SD + Y \]

8.3 Periodic Benefits

The benefit of application engineering is the savings accrued by using reusable assets instead of developing the code from scratch (at year SD) and the savings in maintenance that result from using high quality reusable components rather than components developed from scratch.

At year SD, the benefits

\[ B_a (SD) = (E - (RA + DEV + IN))(Pay) \]

where

E=Cost of developing the application from scratch without reuse  
RA=Reuse Adoption Costs  
DEV=Cost of developing custom glue code  
IN=Cost of integrating the various components  
Pay=Average monthly salary of a developer

E and DEV are calculated by applying COCOMO to the size (number of lines of code in thousands). IN is estimated based on the formula in Chapter 4.
For $y > SD$, the benefit of application engineering is

$$B_a(y) = OCD(y)(Pay)$$

where

- $B_a(y)$ = Benefits of application engineering at year $y$
- $OCD(y)$ = Operating Cost Differential at year $y$
- $Pay$ = Average monthly salary of the developer(s) maintaining the component

$OCD$ is the difference in maintenance/operating costs between a component developed for reuse and a component developed from scratch. This assumes that a component developed with reuse (especially if it is a black box component) is of better quality than a component developed from scratch for single use since it has been designed more carefully, tested more thoroughly, and documented more precisely. Furthermore, as a result of the latter, less maintenance of the component is necessary over subsequent years. The maintenance costs are calculated using the maintenance cost formula for COCOMO/COCOMO 2.0:

$$OCD = (ACT - ACT')E(Pay) = .06E(Pay)$$

where

- $OCD$ = Operating Cost Differential
- $ACT$ = Annual Change Traffic
- $ACT'$ = Annual Change Traffic for reusable assets
- $E$ = Cost to develop the component from scratch
- $Pay$ = Average monthly salary of the developer

$ACT$ is the ratio of the yearly maintenance cost to the development cost and typically around 0.15 [Boehm] and $ACT'$ around 0.09 since the operating cost differential due to software reuse is 9% of the development effort.

### 8.4 An Illustrative Example of Application Engineering Benefits and Costs

<table>
<thead>
<tr>
<th>Year $y$</th>
<th>Cost $C_a(y)$</th>
<th>Benefit $B_a(y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>Purchase of Reusable Assets</td>
<td>Savings in Development Costs</td>
</tr>
<tr>
<td>$&gt;SD$</td>
<td></td>
<td>Quality Gains</td>
</tr>
</tbody>
</table>

For example, consider two applications. One application, cs98, consists of 10 black box assets of size 5K, 20 white box assets of size 5K, and 100K of custom code. The other application, cs99, consists of 20 black box of size 5K, 10 white box of size 5K, and 50K custom code. These applications assume the price for white box reuse (WP) is 0.15E and
black box (BP) reuse is 0.40E and developed by a programmer earning $60K/year. Thus, as illustrated in Chapter 7,

\[ E_{5k} = 3S_{1.12}^{1.12} = 3(5^{1.12}) = 18.2PM \]

\[ BP(y) = 0.4E(Pay) = 0.4(18.2PM)(\frac{\$60K}{12}) = \$36,400 \]

\[ WP(y) = 0.15E(Pay) = 0.15(18.2PM)(\frac{\$60K}{12}) = \$13,650 \]

### 8.4.1 Application cs98

10 Black Box Assets  
20 White Box Assets  
100K of Custom Code  
Total Lines of Code=10(5K)+20(5K)+100K=250K

The costs associated with the application are

\[ IC = \sum_{i=1}^{NC} PR_i \]

\[ IC = C_a (1998) \]
\[ C_a (1998) = 10BP(y) + 20WP(y) = 10(\$36,400) + 20(\$13,650) = \$637,000 \]
\[ C_a (1999) = 0 \]
\[ C_a (2000) = 0 \]
\[ C_a (2001) = 0 \]

The benefits associated with the application are

\[ B_a (SD) = (E - (RA + DEV + INT))(Pay) \]

where

\[ \text{DEV} = \text{E}_{250K} = 3(100)^{1.12} \]
\[ \text{RA} = 10(0.20(\text{E}_{5k})) + 20(0.67(\text{E}_{5k})) \]
\[ \text{INT} = \text{E}_d - (\sum_{i=1}^{n} E_i) \]

where

\[ n = \text{Total number of components} \]
\[ tl = \text{Total lines of code of all the components} \]
\[ B_a(1998) = (E^{250K} - 10(0.20(E^{5K})) - 20(0.67(E^{5K})) - E^{100K} - (E^{150K} - 30(E^{5K}))(Pay) \]
\[ = (3(250)^{1.12} - 10(0.20(3(5)^{1.12})) - 20(0.67(3(5)^{1.12})) - 3(100)^{1.12}) - (3(150)^{1.12}) + 30((5)^{1.12})\left(\frac{S60K}{12}\right) \]
\[ = (1,454.84 - 36.39 - 243.88 - 521.34 - (821.00 - 546.00))(\$5000) \]
\[ = \$1,891,150.00 \]

\[ B_a(1999) = (ACT - ACT')E(Pay) = 0.06(E^{250K})(Pay) = 0.06(3(250)^{1.12})\left(\frac{S60K}{12}\right) = \$436,450.83 \]

\[ B_a(2000) = (ACT - ACT')E(Pay) = 0.06(E^{250K})(Pay) = 0.06(3(250)^{1.12})\left(\frac{S60K}{12}\right) = \$436,450.83 \]

\[ B_a(2001) = (ACT - ACT')E(Pay) = 0.06(E^{250K})(Pay) = 0.06(3(250)^{1.12})\left(\frac{S60K}{12}\right) = \$436,450.83 \]

<table>
<thead>
<tr>
<th>Year</th>
<th>C(Year)</th>
<th>B(Year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998</td>
<td>$637,000.00</td>
<td>$1,891,150.00</td>
</tr>
<tr>
<td>1999</td>
<td>0</td>
<td>$436,450.83</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>$436,450.83</td>
</tr>
<tr>
<td>2001</td>
<td>0</td>
<td>$436,450.83</td>
</tr>
</tbody>
</table>

**8.4.2 Application cs99**

20 Black Box
10 White Box
50 K Custom
Total Lines of Code=20(5K)+10(5K)+50K=200K

The costs associated with application cs99 are

\[ IC = \sum_{i=1}^{NC} PR_i \]

\[ IC = C_a(1999) \]

\[ C_a(1999) = 20BP(y) + 10WP(y) = 20(36,400) + 10(13,650) = 864,500 \]

\[ C_a(2000) = 0 \]

\[ C_a(2001) = 0 \]

\[ C_a(2002) = 0 \]

\[ B_a(SD) = (E - (RA + DEV + IN))(Pay) \]
\[ B_a(1998) = (E_{200K} - 20(E_{5k}) - 10(0.67(E_{5k})) - E_{50K} - (E_{150K} - 30(E_{5k}))(Pay) \]
\[ = 3(200)^{1.12} - 20(0.20(3(5)^{1.12}) - 10(0.67(3)(5)^{1.12}) - 3(50)^{1.12} - (3(150)^{1.12} + 30(5)^{1.12})\left(\frac{60K}{12}\right) \]
\[ = (1133.12 - 72.80 - 121.94 - 239.87 - (821.00 - 546.00))(-5000) \]
\[ = 2,117,550 \]

\[ B_a(1999) = (ACT - ACT')E(Pay) = 0.06(E_{200K})(Pay) = 0.06(3(200)^{1.12})\left(\frac{60K}{12}\right) = 339,935.17 \]

\[ B_a(2000) = (ACT - ACT')E(Pay) = 0.06(E_{200K})(Pay) = 0.06(3(200)^{1.12})\left(\frac{60K}{12}\right) = 339,935.17 \]

\[ B_a(2001) = (ACT - ACT')E(Pay) = 0.06(E_{200K})(Pay) = 0.06(3(200)^{1.12})\left(\frac{60K}{12}\right) = 339,935.17 \]

<table>
<thead>
<tr>
<th>Year</th>
<th>C(Year)</th>
<th>B(Year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>$864,500.00</td>
<td>$2,117,500.00</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>$339,935.17</td>
</tr>
<tr>
<td>2001</td>
<td>0</td>
<td>$339,935.17</td>
</tr>
<tr>
<td>2002</td>
<td>0</td>
<td>$339,935.17</td>
</tr>
</tbody>
</table>
Chapter 9. Domain Engineering Cycle
In the domain engineering cycle, managers must decide if it is worthwhile to initiate a domain engineering effort in a pre-specified application domain. Factors that affect this decision are

- Whether the application domain has well defined functions that are recurrent across applications
- Whether these functions vary from application to application and how frequently they are used
- What the cost of developing these components with and without reuse is
- How much effort is necessary to perform domain analysis and design for reuse

9.1 Up-front costs

The up-front costs of the domain engineering cycle are the costs associated with domain analysis, component development for reuse, and component cataloging.

\[ IC = \sum_{c \in d} C_c(SD)(Pay) + DA \]

where

- \( C_c(SD) \) = Development and cataloging costs of all the domain components
- \( DA \) = Domain analysis costs
- \( Pay \) = Average monthly salary of the developer

The domain analysis costs result from the analysis of the application domain and investigation of the common features that applications within the domain may have that may warrant developing reusable components. This domain analysis also includes

- Identifying the common and variable features
- Specifying an architecture for domain applications
- Specifying reusable assets with appropriate generality

Developing reusable assets develops components for reuse with generality of specification and design and quality. These costs including domain engineering costs are

\[ C_d(y) = RCDE(\sum_{i=1}^{N} E_i)(Pay) \]

where

- \( C_d(y) \) = Total costs of developing reusable components
- \( RCDE \) = Relative Costs of Domain Engineering (default is 2.0)
- \( E \) = Cost to develop the component from scratch
- \( Pay \) = Average monthly salary of the developer
9.2 Episodic Costs

The episodic costs of the domain engineering cycle are the sum of the costs of all components developed at year $y$ as part of the domain. If $SD+1 \leq y \leq SD+Y$, then the episodic costs at year $y$ are the sum of the costs of the components at year $y$:

$$C_d(y) = \sum_{c \in d} C_c(y)(\text{Pay})$$

where

- $C_d(y) =$ Episodic costs of domain engineering, for domain $d$
- $C_c(y) =$ Cost of development of the component, $c$, at year $y$. $C_c$ is calculated using the formulas in Chapter 7
- Pay =$\text{Average monthly salary of the developer}$

Thus, all of the components associated with a particular domain do not need to be developed at the beginning of the domain engineering cycle but can be developed over time.

9.3 Episodic Benefits

The episodic benefits of the domain engineering cycle are the sum of the benefits of all components that are part of the domain. Thus, for $SD+1 \leq y \leq SD+Y$,

$$B_d(y) = \sum_{c \in d} B_c(y)$$

where

- $B_d(y) =$ Episodic benefits of the domain engineering cycle
- $B_c(y) =$ Benefit of component, $c$, that results from the sale of $c$ to project teams. $B_c$ is calculated using the formulas in Chapter 7.

9.4 An Illustrative Example of Domain Engineering Costs and Benefits

<table>
<thead>
<tr>
<th>Year, y</th>
<th>Cost, C(y)</th>
<th>Benefit, B(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>Domain Analysis, Asset Development</td>
<td></td>
</tr>
<tr>
<td>&gt;SD</td>
<td>Asset Development, Concentrated Domain Analysis</td>
<td>Asset Sales to Projects</td>
</tr>
</tbody>
</table>

Consider a domain that consists of 10 components of size 5K of which 5 components were developed in 1997 and 5 components in 1998. The domain analysis costs were 60%
of the cost of development of a component developed from scratch. This domain analyst earns $108K/yr and programmer earns $60K/yr. The development cost for reuse (ER) for a 5K component is:

$$ER = RCWR(E)(Pay) = 1.5(18.2PM)(\frac{108K}{12}) = $136,500$$

$$DA = (10).6E(Pay) = 6(18.2PM)(\frac{100K}{12}) = $982,800$$

The costs associated with domain engineering are:

$$C_d(1997) = IC = DA + \sum_{y \in d} C_d(y) = DA + 5(RCWR(E_{1997})) = DA + 5(ER_{1997})$$

$$= 982,800 + 5(136,500) = $1,665,300$$

$$C_d(1998) = \sum_{y \in d} C_d(1998) = 5(ER_{1998}) = 5(136,500) = $682,500$$

$$C_d(1999) = 0$$

$$C_d(2000) = 0$$

The benefits associated with domain engineering are:

$$B_d(1997) = 5(B_c(1997)) = 5(0) = 0$$

$$B_d(1998) = 10(B_c(1998)) = 10(63,700) = $637,000$$

$$B_d(1999) = 10(B_c(1999)) = 10(86,450) = $864,500$$

$$B_d(2000) = 10(B_c(2000)) = 10(0) = 0 PM$$

where $B_c(y)$ is the benefit of the component at year, $y$.

<table>
<thead>
<tr>
<th>Year</th>
<th>C(Year)</th>
<th>B(Year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>$1,665,300.00</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>$682,500.00</td>
<td>$637,000.00</td>
</tr>
<tr>
<td>1999</td>
<td>0</td>
<td>$864,500.00</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Chapter 10. Corporate Engineering Cycle
The corporate engineering cycle decision is the overall decision of whether to launch a software reuse program within the corporation. This cycle includes all the other investment cycles (Figure 2) such that the periodic costs of this cycle are the domain engineering costs, and the periodic benefits of this cycle are the application engineering cycle benefits. The domain engineering cycle includes the costs from the component engineering cycle. As a result of this cascade structure, the benefit of selling reusable components in the domain engineering cycle is cancelled by the cost of purchasing these components in the application engineering cycle, resulting in a transfer of resources within the corporation that has no impact on the corporate balance sheet.

10.1 Investment Costs

The up-front costs of the corporate engineering cycle include the costs of building a reuse infrastructure and implementing a reuse program (IS):
- Purchase and installation of a repository to store the components
- Personnel hiring and training
- Operational modifications
- Cost of populating the reuse library initially

Thus for the corporate engineering cycle

$$IC = IS$$

where

IC=Investment Cost  
IS=Infrastructure Costs

10.2 Episodic Costs

The periodic costs of the corporate investment cycle are the periodic costs of the domain engineering cycle, which are the costs of developing reusable components, otherwise known as the component engineering cycle. The latter is possible due to the assumption that the benefit of selling these reusable components of the domain engineering phase is the same as the cost of purchasing these components in the application engineering phase. Thus, these costs include the up-front costs of the domain engineering initiatives and the up-front costs of component development.

$$C(y) = \sum_{d \in corp} C_d(y)$$

where

SD+1\leq y \leq SD+Y  
C(y)=Episodic costs of the corporate engineering lifecycle at year y  
C_d(y)=Costs of the domain engineering cycle at year y
10.3 Episodic Benefits

The episodic benefits of the corporate engineering cycle are the benefits of the application engineering cycle projects. The latter assumes that the cost of purchasing the reusable components for these projects is the same as the selling price of these components in the domain engineering phase, all of which are within the corporation. These benefits include gains in productivity (savings in development effort) and gains in quality (savings in maintenance costs).

\[ B(y) = \sum_{a \in \text{Corp}} B_a(y) \]

where

\[ SD+1 \leq y \leq SD+Y \]

\[ B(y) = \text{Episodic benefits of the corporate engineering phase} \]

\[ B_a(y) = \text{Benefits of application engineering cycle, a} \]

10.4 An Illustrative Example of Corporate Costs and Benefits

<table>
<thead>
<tr>
<th>Year, y</th>
<th>Costs, C(y)</th>
<th>Benefits, B(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>Infrastructure</td>
<td></td>
</tr>
<tr>
<td>&gt;SD</td>
<td>Domain Costs</td>
<td>Application Benefits</td>
</tr>
</tbody>
</table>

In this example, the corporation started in 1997 and consists of the cs98 and cs99 application and the domain described in Chapter 9. It is assumed the infrastructure costs were $250,000.

The episodic costs associated with this corporation are:

\[ C(1997) = IC = 250,000 + C_d(1997) = 250,000 + 1,665,300 = 1,915,300 \]


\[ C(1999) = C_d(1999) = 0 \]


The episodic benefits associated with this corporation are:

\[ B(1997) = 0 \]

\[ B(1998) = B_a(1998_{cs98}) = 1,891,150.00 \]

\[ B(1999) = B_a(1999_{cs98}) + B_a(1999_{cs99}) = 436,450.83 + 2,117,550.00 = 2,554,000.83 \]

\[ B(2000) = B_a(2000_{cs98}) + B_a(2000_{cs99}) = 436,450.83 + 339,935.17 = 776,386.00 \]
The following is the summary of the costs and benefits of the various engineering cycles:

**Component Engineering Balance Sheet**

<table>
<thead>
<tr>
<th>Year, y</th>
<th>Cost, ( C_c(y) )</th>
<th>Benefit, ( B_c(y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>$143,780.00</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>$9,300.00</td>
<td>$63,700.00</td>
</tr>
<tr>
<td>1999</td>
<td>$9,300.00</td>
<td>$86,450.00</td>
</tr>
<tr>
<td>2000</td>
<td>$9,300.00</td>
<td>0</td>
</tr>
</tbody>
</table>

**Application Engineering Balance Sheet**

**CS 98**

<table>
<thead>
<tr>
<th>Year, y</th>
<th>Cost, ( C_a(y) )</th>
<th>Benefit, ( B_a(y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998</td>
<td>$637,000.00</td>
<td>$1,891,150.00</td>
</tr>
<tr>
<td>1999</td>
<td>0</td>
<td>$436,450.83</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>$436,450.83</td>
</tr>
<tr>
<td>2001</td>
<td>0</td>
<td>$436,450.83</td>
</tr>
</tbody>
</table>

**CS 99**

<table>
<thead>
<tr>
<th>Year, y</th>
<th>Cost, ( C_a(y) )</th>
<th>Benefit, ( B_a(y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>$864,500.00</td>
<td>$2,117,500.00</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>$339,935.17</td>
</tr>
<tr>
<td>2001</td>
<td>0</td>
<td>$339,935.17</td>
</tr>
<tr>
<td>2002</td>
<td>0</td>
<td>$339,935.17</td>
</tr>
</tbody>
</table>
Domain Engineering Balance

<table>
<thead>
<tr>
<th>Year, y</th>
<th>Cost, C_d(y)</th>
<th>Benefit, B_d(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>$1,665,300.00</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>$682,500.00</td>
<td>$637,000.00</td>
</tr>
<tr>
<td>1999</td>
<td>0</td>
<td>$864,500.00</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Corporate Engineering Balance Sheet

<table>
<thead>
<tr>
<th>Year, y</th>
<th>Costs, C(y)</th>
<th>Benefits, B(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>$1,915,300.00</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>$682,500.00</td>
<td>$1,891,150.00</td>
</tr>
<tr>
<td>1999</td>
<td>0</td>
<td>$2,554,000.83</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>$776,386.00</td>
</tr>
</tbody>
</table>

The corporate engineering costs and benefits stem from the costs and benefits of the component engineering phase. The costs associated with the corporate engineering cycle are the costs of all domains initiated or maintained during the lifecycle of the corporation, while the benefits of the corporate investment cycle are the benefits from all applications developed during the lifecycle of the corporation. The costs of all the domains result from the costs and benefits of the components developed for domain engineering, and the benefits of application engineering cycle result from the costs and benefits accumulated from the reused assets that comprise the application. Thus, the costs and benefits of the corporation are the sum of the costs and benefits from the application, domain and component engineering cycles.

The component engineering cycles feed the costs and benefits to the domain engineering cycle and application engineering cycle (Figure 3). These cycles in turn feed the corporate engineering cycle in such a way that the costs from the domain engineering cycle comprise the costs of the corporate engineering cycle, and the benefits from building applications from reusable assets comprise the benefits of the corporate engineering cycle.

From the costs and benefits of each of these reuse engineering cycles and from some additional cost factors such as investment rate(d), initial costs (IC), and length of investment cycle(Y), the economic worthiness of reuse in the corporation can be derived using the economic functions presented in Chapter 3. In the next chapter, this derivation of the economic functions from the costs and benefits of the component, domain, application and corporate engineering cycles will be illustrated for each cycle.
Chapter 11. An Example of Application of the Integrated Cost Model
The example illustrated in this chapter consists of a corporation that starts its reuse initiative in 1997 consisting of 5 components of size 5 KLOC that were developed in 1997 and 5 components of size 5 KLOC that were developed in 1998. In 1997, a domain engineering initiative is started which produces 10 components. Furthermore, two application engineering initiatives were initiated:
- cs98 in 1998 consisting of 10 black box components and 20 white box components and 100K of custom code
- cs99 in 1999 consisting of 20 black box components and 10 white box components and 50K of custom code

The following assumptions were made with respect to this corporation:
- The average yearly salary of a developer is $60K
- The average yearly salary of a librarian is $48K
- The average yearly salary of a domain analyst is $108K
- The infrastructure costs associated with the corporation’s reuse initiative is $250,000

For each engineering cycle (component, domain, application and corporate) the following economic functions are calculated, based on a duration of 3 years and a investment rate of 15%:
- Net Present Value (NPV)
- Profitability Index (PI)
- Average Return on Book Value (ARBV)
- Payback Value
- Return on Investment (ROI)

11.1 Component Engineering Cycle
Consider a component of size 5 KLOC that is developed by a programmer earning $60K/year and maintained in a library by a librarian earning $48K/year to maintain 240 components. Consider, library maintenance is 10% of the development costs of this component from scratch and library insertion is 10% of the development costs from scratch.

The Development Cost from Scratch using the Basic COCOMO in organic mode:

\[ E = 3S^{1.12} = 3(5^{1.12}) = 18.196 = 18.2PM \]

The Development Cost for Reuse

\[ ER = (RCWR)(E)(Pay) = 1.5(18.2)\left(\frac{60K}{12}\right) = 136,500 \]

The Library Insertion Cost

\[ LI = 0.1E(Pay) = (0.1)(18.92)\left(\frac{48K}{12}\right) = 7,280 \]
The Initial Cost or Cost at $y=SD$

\[ IC = ER + LI = $136,500 + $7,280 = $143,780 \]

The Maintenance Cost of this component at year $y>SD$

\[ MN(y) = 0.1E = .1(18.2) = 1.82PM \]

The Operating Cost of Maintaining this Component in the Library at year $y>SD$

\[ OC(y) = \frac{12}{240} = 0.05PM \]

Thus, the cost associated with this component each year after the start date, $y>SD$, is

\[ C_c(y) = OC(y)(Pay) + MN(y)(Pay) \]

\[ C_c(y) = (0.05PM)(\frac{$48K}{12}) + (1.82PM)(\frac{$60K}{12}) \]

\[ C_c(y) = $200 + $9,100 = $9,300 \]

Finally the benefits of this reusable component after the start date, $y>SD$, is

\[ B_c(y) = (freqb(y))(BP(y)) + (freqw(y))(WP(y)) \]

\[ BP(y) = .4E(Pay) = .4(18.2)(\frac{$60K}{12}) = $36,400 \]

\[ WP(y) = .15E(Pay) = .15(18.2)(\frac{$60K}{12}) = $13,650 \]

To be fair to the application and domain engineer, RBP is 0.4 and RWP is 0.15.

<table>
<thead>
<tr>
<th>Year</th>
<th>Freq bb</th>
<th>Freq wb</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1999</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Using the above frequencies and white box and black box prices, the benefits from component reuse are calculated as follows:
Thus the benefits and costs for this reusable component for each year are:

<table>
<thead>
<tr>
<th>Year</th>
<th>C(Year)</th>
<th>B(Year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>$143,780.00</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>$9,300.00</td>
<td>$63,700.00</td>
</tr>
<tr>
<td>1999</td>
<td>$9,300.00</td>
<td>$86,450.00</td>
</tr>
<tr>
<td>2000</td>
<td>$9,300.00</td>
<td>0</td>
</tr>
</tbody>
</table>

From these costs and benefits the following economic functions as described in Chapter 3 can be calculated:

NPV
\[
NPV = \frac{B(1997) - C(1997)}{(1 + .15)^0} + \frac{B(1998) - C(1998)}{(1 + .15)^1} + \frac{B(1999) - C(1999)}{(1 + .15)^2} + \frac{B(2000) - C(2000)}{(1 + .15)^3} \\
= \frac{0 - 143,780}{1.15} + \frac{63,700 - 9,300}{1.15^2} + \frac{86,450 - 9,300}{1.15^3} + \frac{0 - 9,300}{1.15^3} \\
= -143,780 + 47,304.35 + 58,336.48 - 6,114.90 \\
= -$44,254.07
\]

ROI = \frac{NPV}{IC} = \frac{-44,254.07}{143,780} = -.31

\[
PI = \frac{NPVb}{NPVe} = \frac{\left(0 + \frac{63,700}{1.15} + \frac{86,450}{1.3225} + 0\right)}{\left(143,780 + \frac{9,300}{1.15} + \frac{9,300}{1.3225} + \frac{9,300}{1.520875}\right)} \\
= \frac{120,759.92}{165,014.00} = .73
\]

\[
ARBV = \frac{NPV}{IC} = \frac{-44,254.07}{3 \times \frac{143,780}{2}} = \frac{-14751.36}{71,890.00} = -.21
\]
Payback Period = >3 years

<table>
<thead>
<tr>
<th>Y</th>
<th>Sum(C(Y))</th>
<th>Sum(B(Y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>$143,780.00</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>$153,080.00</td>
<td>$63,700.00</td>
</tr>
<tr>
<td>1999</td>
<td>$162,380.00</td>
<td>$150,150.00</td>
</tr>
</tbody>
</table>

11.2 Domain Engineering Cycle

Consider a domain initiative that consists of 10 components of size 5K of which 5 components were developed in 1997 and 5 components in 1998. The domain analysis costs were 60% of the cost of development of a component developed from scratch. This domain analyst earns $108K/yr and programmer earns $60K/yr.

The development cost for reuse (ER) for a 5K component is:

$$ ER = RCWR(E)(Pay) = 1.5(18.2PM)(\frac{\$60K}{12}) = \$136,500 $$

$$ DA = (10).6E(Pay) = 6(18.2PM)(\frac{\$108K}{12}) = \$982,800 $$

The costs associated with domain engineering are:

$$ C_d(1997) = IC = DA + \sum_{c \in d} C_c(y) = DA + 5(RCWR(E_{1997})) = DA + 5(ER_{1997}) $$

$$ = \$982,800 + 5(\$136,500) = \$1,665,300 $$

$$ C_d(1998) = \sum_{c \in d} C_c(1998) = 5(ER_{1998}) = 5(\$136,500) = \$682,500 $$

$$ C_d(1999) = 0 $$

$$ C_d(2000) = 0 $$

The benefits associated with domain engineering are:

$$ B_d(1997) = 5(B_c(1997)) = 5(0) = \$0 $$

$$ B_d(1998) = 10(B_c(1998)) = 10(\$63,700) = \$637,000 $$

$$ B_d(1999) = 10(B_c(1999)) = 10(\$86,450) = \$864,500 $$

$$ B_d(2000) = 10(B_c(2000)) = 10(0) = 0 \text{ PM} $$
where $B_c(y)$ is the benefit of the component at year, $y$.

<table>
<thead>
<tr>
<th>Year</th>
<th>C(Year)</th>
<th>B(Year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>$1,665,300.00</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>$682,500.00</td>
<td>$637,000.00</td>
</tr>
<tr>
<td>1999</td>
<td>0</td>
<td>$864,500.00</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

These latter costs and benefits are used to determine the following functions:

NPV

$$\text{NPV} = \frac{B(1997) - C(1997)}{(1+.15)^0} + \frac{B(1998) - C(1998)}{(1+.15)^1} + \frac{B(1999) - C(1999)}{(1+.15)^2} + \frac{B(2000) - C(2000)}{(1+.15)^3}$$

$$= \frac{(-$0 - $1,665,300)}{1.15} + \frac{($637,000 - $682,500)}{1.15^2} + \frac{($864,500 - $0)}{1.15^3} + \frac{($0 - $0)}{1.15^4}$$

$$= -$1,665,300 - 39,565.22 + 653,686.20$$

$$= -$1,051,179.02$$

ROI = $\frac{\text{NPV}}{\text{IC}} = \frac{-$1,051,179.02}{$1,665,300.00} = -.63$

PI = $\frac{\text{NPV}_b}{\text{NPV}_c} = \frac{($0 + $637,000)}{1.15} + \frac{($864,500 + 0)}{1.3225} + \frac{(0 + 0)}{(1,665,300 + $682,500)} + \frac{(0 + 0)}{1.15}$

$$= \frac{$1,207,599.24}{$2,258,778.26} = .53$$

ARBV

$$\frac{\text{NPV}}{\text{IC}} = \frac{-$1,051,179.02}{\frac{3}{2}} = \frac{3}{1,665,300} = \frac{-$832,650.00}{-$.350,393.01} = -.42$$
Payback Period does not occur after 3 years

<table>
<thead>
<tr>
<th>Y</th>
<th>Sum(C(Y))</th>
<th>Sum(B(Y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>$1,665,300.00</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>$2,347,800.00</td>
<td>$637,000.00</td>
</tr>
<tr>
<td>1999</td>
<td>$2,347,800.00</td>
<td>$1,501,500.00</td>
</tr>
<tr>
<td>2000</td>
<td>$2,347,800.00</td>
<td>$1,501,500.00</td>
</tr>
</tbody>
</table>

11.3 Application Engineering Cycle

11.3.1 Application cs98

10 Black Box Assets
20 White Box Assets
100K of Custom Code
Total Lines of Code=10(5K)+20(5K)+100K=250K

The costs associated with the application are:

\[ IC = \sum_{i=1}^{NC} PR_i \]

\[ IC = C_a (1998) \]
\[ C_a (1998) = 10BP(y) + 20WP(y) = 10(36,400) + 20(13,650) = 637,000 \]
\[ C_a (1999) = 0 \]
\[ C_a (2000) = 0 \]
\[ C_a (2001) = 0 \]

The benefits associated with the application are

\[ B_a (SD) = (E - (RA + DEV + IN))(Pay) \]

where

\[ DEV = E_{250K} = 3(100)^{1.12} \]
\[ RA = 10(0.20(E_{5K}) + 20(0.67(E_{5K})) \]
\[ INT = E_d - \left( \sum_{i=1}^{n} E_i \right) \]
where

\[ n = \text{Total number of components} \]
\[ \text{tl} = \text{Total lines of code of all the components} \]

\[
B_a(1998) = (E_{250K} - 10(.20(E_{5k})) - 20(.67(E_{5k})) - E_{100K} - (E_{150K} - 30(E_{5k}))(Pay)
= (3(250)^{1.12} - 10(.20(3(5)^{1.12}) - 20(.67(3(5)^{1.12}) - 3(100)^{1.12} - (3(150)^{1.12}) + 30((5)^{1.12}))(\frac{\$60K}{12})
= (1,454.84 + 36.39 + 243.88 - 521.34 - (821.00 - 546.00))($5000)
= $1,891,150.00
B_a(1999) = (ACT - ACT')E(Pay) = .06 (E_{250k})(Pay) = .06(3(250)^{1.12})(\frac{\$60K}{12}) = $436,450.83
B_a(2000) = (ACT - ACT')E(Pay) = .06 (E_{250k})(Pay) = .06(3(250)^{1.12})(\frac{\$60K}{12}) = $436,450.83
B_a(2001) = (ACT - ACT')E(Pay) = .06 (E_{250k})(Pay) = .06(3(250)^{1.12})(\frac{\$60K}{12}) = $436,450.83

\begin{array}{|c|c|c|}
\hline
\text{Year} & \text{C(Year)} & \text{B(Year)} \\
\hline
1998 & $637,000.00 & $1,891,150.00 \\
1999 & 0 & $436,450.83 \\
2000 & 0 & $436,450.83 \\
2001 & 0 & $436,450.83 \\
\hline
\end{array}

From the latter costs and benefits the following economic functions are determined:

NPV
\[
\text{NPV} = \frac{B(1997) - C(1997)}{(1 + .15)^0} + \frac{B(1998) - C(1998)}{(1 + .15)^1} + \frac{B(1999) - C(1999)}{(1 + .15)^2} + \frac{B(2000) - C(2000)}{(1 + .15)^3}
= ($1,891,150 - $637,000) + \frac{($436,450.83 - $0)}{1.15} + \frac{($436,450.83 - $0)}{1.15^2} + \frac{($436,450.83 - $0)}{1.15^3}
= $1,254,150 + 379,522.46 + 330,019.53 + 286,973.51
= $2,250,665.50
\]
ROI = \frac{NPV}{IC} = \frac{\$2,250,665.50}{\$637,000} = 3.53

\[ PI = \frac{NPV_b}{NPV_c} = \frac{\$1,891,150 + \frac{\$436,450.83}{1.15} + \frac{\$436,450.83}{1.3225} + \frac{\$436,450.83}{1.520875}}{\$637,000 + \$0 + \$0 + \$0} = 4.53 \]

\[ ARBV = \frac{NPV}{Y} = \frac{\$2,250,665.50}{3} = \frac{\$750,221.83}{\$318,500} = 2.36 \]

Payback Period = 1 year

<table>
<thead>
<tr>
<th>Year</th>
<th>Sum(C(Y))</th>
<th>Sum(B(Y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998</td>
<td>$637,000.00</td>
<td>$1,891,150.00</td>
</tr>
</tbody>
</table>

11.3.2 Application cs99

20 Black Box
10 White Box
50 K Custom
Total Lines of Code=20(5K)+10(5K)+50K=200K

The costs associated with application cs99 are

\[ IC = \sum_{i=1}^{NC} PR_i \]

\[ IC = C_a (1999) \]
\[ C_a (1999) = 20BP(y) + 10WP(y) = 20(\$36,400) + 10(\$13,650) = \$864,500 \]
\[ C_a (2000) = 0 \]
\[ C_a (2001) = 0 \]
\[ C_a (2002) = 0 \]
From these costs and benefits the following economic functions are determined:

NPV

\[
NPV = \frac{B(1997) - C(1997)}{(1 + .15)^0} + \frac{B(1998) - C(1998)}{(1 + .15)^1} + \frac{B(1999) - C(1999)}{(1 + .15)^2} + \frac{B(2000) - C(2000)}{(1 + .15)^3}
\]

\[
\]

\[
= ($2,117,500 - $864,500) + \frac{($339,935.17 - $0)}{1.15} + \frac{($339,935.17 - $0)}{1.15^2} + \frac{($339,935.17 - 0)}{1.15^3}
\]

\[
= $1,253,000.00 + $295,595.80 + $257,039.83 + $223,512.89
\]

\[
= $2,029,148.52
\]

ROI = \frac{NPV}{IC} = \frac{$2,029,148.52}{$864,500} = 2.35

In this example, the corporation started its reuse initiative in 1997 consisting of the cs98 and cs99 application and the domain described in Section 11.2. It is assumed the infrastructure costs were $250,000.

The episodic costs associated with this corporation are:

\[
C(1997) = IC = \$250,000 + C_d (1997) = \$250,000 + \$1,665,300 = $1,915,300 \\
\]

The episodic benefits associated with this corporation are:

\[
B(1997) = 0 \\
B(1998) = B_d (1998_{cs98}) = $1,891,150 \\
B(1999) = B_d (1999_{cs98}) + B_a (1999_{cs99}) = $436,450.83 + $2,117,550.00 \\
\quad = $2,554,000.83 \\
\quad = $776,386.00
\]
<table>
<thead>
<tr>
<th>Year</th>
<th>C(Year)</th>
<th>B(Year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>$1,915,300.00</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>$682,500.00</td>
<td>$1,891,150.00</td>
</tr>
<tr>
<td>1999</td>
<td>0</td>
<td>$2,554,000.83</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>$776,386.00</td>
</tr>
</tbody>
</table>

From the benefits and costs determined above the following economic functions are calculated:

NPV

\[
\text{NPV} = \frac{B(1997) - C(1997)}{(1 + .15)^0} + \frac{B(1998) - C(1998)}{(1 + .15)^1} + \frac{B(1999) - C(1999)}{(1 + .15)^2} + \frac{B(2000) - C(2000)}{(1 + .15)^3}
\]

\[
= \frac{(-$0 - $1,915,300)}{1} + \frac{($1,891,150 - $682,500)}{1.15} + \frac{($2,554,000 - $0)}{1.15^2} + \frac{($776,386 - 0)}{1.15^3}
\]

\[
= -$1,915,300 + $1,051,000 + $1,931,191.55 + $510,486.40
\]

\[
= $1,571,377.95
\]

ROI = \frac{\text{NPV}}{\text{IC}} = \frac{$1,571,377.95}{$1,915,300} = .82

\[
\text{PI} = \frac{\text{NPV}_b}{\text{NPV}_c} = \frac{($0 + \frac{$1,891,150}{1.15} + \frac{$2,554,000}{1.3225} + \frac{$776,386}{1.520875})}{($1,915,300 + \frac{$682,500}{1.15} + $0 + $0)}
\]

\[
= \frac{$4,086,155.59}{$2,508,778.26} = 1.63
\]

ARBV

\[
\frac{\text{NPV} \times \frac{\text{Y}}{\text{IC}}}{2} = \frac{\text{NPV} \times \frac{3}{2}}{2} = \frac{\$1,571,377.95 \times \frac{3}{2}}{2} = \frac{\$525,792.65}{957,650} = .55
\]
Payback Period = 3 years

<table>
<thead>
<tr>
<th>Year</th>
<th>Sum(C(Y))</th>
<th>Sum(B(Y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>$1,915,300.00</td>
<td>0</td>
</tr>
<tr>
<td>1998</td>
<td>$2,597,800.00</td>
<td>$1,891,150.00</td>
</tr>
<tr>
<td>1999</td>
<td>$2,597,800.00</td>
<td>4,445,150.83</td>
</tr>
</tbody>
</table>

11.5 Summary of the Calculation of the Economic Functions for this Example

The economic functions results are summarized in Table 7 for each of the engineering cycles for a corporation consisting of 10 components of size 5K, five of which were developed in 1997 and five of which were developed in 1998. These components were developed for the domain that was developed in 1997 and used in two applications in 1998, CS98, and 1999, CS99.

<table>
<thead>
<tr>
<th>Engineering Cycle</th>
<th>NPV</th>
<th>PI</th>
<th>ARBV</th>
<th>ROI</th>
<th>Payback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>-$44,254.07</td>
<td>0.73</td>
<td>-0.21</td>
<td>-0.31</td>
<td>&gt;3 years</td>
</tr>
<tr>
<td>Domain</td>
<td>-$1,051,179.02</td>
<td>0.53</td>
<td>-0.42</td>
<td>-0.63</td>
<td>&gt;3 years</td>
</tr>
<tr>
<td>Application, CS98</td>
<td>$2,250,665.50</td>
<td>4.53</td>
<td>2.36</td>
<td>3.53</td>
<td>1 year</td>
</tr>
<tr>
<td>Application, CS99</td>
<td>$2,029,148.52</td>
<td>3.35</td>
<td>1.56</td>
<td>2.35</td>
<td>1 year</td>
</tr>
<tr>
<td>Corporate</td>
<td>$1,577,377.95</td>
<td>1.63</td>
<td>0.55</td>
<td>0.82</td>
<td>3 years</td>
</tr>
</tbody>
</table>

Table 7 Summary of Engineering Cycle’s Economic Functions

From a manager’s point of view this corporate reuse investment would be worthwhile. On average, the manager would recover the initial reuse investment within 3 years and profit $1,577,377.95. Furthermore, the ROI is 82% with an ARBV of 55% and PI =1.63.

Both application investments (cs98 and cs99) would be recouped within a year. Both NPV, ROI, and PI indicate building these applications with reusable components would be a good risk and investment. These investments would return $2,250,665.50 and $2,029,148.52 with a ARBV of 236% and 156% and ROI of 353% and 235%.
If this decision were made on the component or domain engineering cycle level, this investment may not have been made. At these decision levels, NPV, ARBV, and ROI are negative, PI < 1, and, as noted before, investment is not recouped in 3 years. Thus, the domain engineering and component engineering initiatives do not appear to be a good risk. The latter is due to the cost of developing the components for reuse and the domain analysis costs.

In order to maximize the investment, the NPV’s at the component and domain engineering level need to be positive. Thus if the relative black box price became 0.8 and the relative white box price became 0.25, the following would result:

<table>
<thead>
<tr>
<th>Engineering Cycle</th>
<th>NPV</th>
<th>PI</th>
<th>ARBV</th>
<th>ROI</th>
<th>Payback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>$77,382.16</td>
<td>1.39</td>
<td>0.36</td>
<td>0.54</td>
<td>2 years</td>
</tr>
<tr>
<td>Domain</td>
<td>$42,885.25</td>
<td>1.02</td>
<td>0.02</td>
<td>0.03</td>
<td>2 years</td>
</tr>
<tr>
<td>Application, CS98</td>
<td>$1,704,665.50</td>
<td>2.44</td>
<td>0.96</td>
<td>1.44</td>
<td>1 year</td>
</tr>
<tr>
<td>Application, CS99</td>
<td>$1,210,148.52</td>
<td>1.72</td>
<td>0.48</td>
<td>0.72</td>
<td>1 year</td>
</tr>
<tr>
<td>Corporate</td>
<td>$1,577,377.95</td>
<td>1.63</td>
<td>0.55</td>
<td>0.82</td>
<td>3 years</td>
</tr>
</tbody>
</table>

Since the costs of domain engineering and the benefits of application engineering comprise the corporate engineering calculations, the costs of domain engineering are recovered in application engineering through the reuse of the components developed in component engineering. This cascading of costs and benefits allows managers to realize the potential of a reuse initiative at both the corporate level and intermediate decision levels (component, domain, and application). Furthermore, by maximizing the ROI, NPV, and ARBV and minimizing the payback period at the component, domain, and application engineering cycle levels, the maximum benefit can be achieved at the corporate level.

As indicated throughout this thesis, NPV is the best indicator if an investment is worthwhile. ROI is a derivative of NPV and takes into account the initial investment. PI, another derivative of NPV, is the ratio of the benefits to the costs, which can return deceiving results, as illustrated above, with the component engineering cycle. ARBV is a poor economic indicator for software since software does not depreciate; it only becomes obsolete. Hence ARBV assumes that maintenance prevents the software from becoming obsolete. Lastly, payback period can be determined based on the costs and benefits derived in Chapters 7-10. If the application is predominantly built from reusable components developed in the corporation, this payback can occur in a short period of time, 1-2 years.

Thus, for this corporation and the applications developed within this corporation with reuse, this reuse initiative investment appears to be worthwhile. Using these economic
functions a manager can determine if using reuse in the building of an application would be worthwhile. In the next chapter, an automated tool will be discussed that enables managers to input the associated costs, frequency of use of the components, and investment costs, and calculate these economic functions illustrated by the example in this chapter.
Chapter 12. An Automated Tool
Many authors have emphasized the "need for effective techniques for evaluating reuse potential and for determining likely investment costs" as well as the need for a "mechanism for quantifying reuse benefits" [7]. Furthermore, as Margano and Rhoades [16] state: “the key to a successful measurement program is that the data collection process itself be as transparent and effortless as possible to all parties concerned. Further, knowing what types of data to collect is essential because a project can be easily tempted to gather too little, too much, or irrelevant information.”

A tool [51-54] that supports the proposed model has been designed and prototyped that runs on top of an Oracle database (©Oracle Corporation) which contains data about corporate costs and benefits, domain costs and benefits, project costs and benefits, and component costs and benefits. The proposed prototype has two main functions:

- Archival: This function is to keep track of the costs and benefits as they arise.
- Analytical: This function is to analyze investment cycles by predicting any combination of the economic functions as presented in Chapter 3.

This analysis can be carried out in post-mortem mode, in predictive mode or in a combination of these modes (Figure 5). Calculations that deal in the past rely on actual data collected by the archival function while calculations that deal in the future rely on predicted/expected data. As time proceeds, predictive data is to be replaced by actual data. Thus, on December 31 of each year, a PL/SQL program will make the predictive data become actual data. A means whereby (1) actual data can be used to correct predictive data and (2) predictive data is adjusted in light of past data is currently being investigated.

### 12.1 Prototype Design

The original prototype design proposed by Senta Fowler Chmiel is in Appendix B. This design was given to two graduate students under the direction of Dr. Ali Mili for implementation. This original design went through a rapid prototyping development phase for two years under the direction of Dr. Ali Mili and Senta Fowler Chmiel. The resulting prototype contained the database schema in Appendix C and the PL/SQL routines to calculate the costs and benefits at the component and domain level in Appendix D. The code still needs to be developed for the application and corporate cycles.

The tables in the database schema in Appendix C store information related to application engineering projects, domain engineering initiatives, and component engineering component descriptions. Users start by populating the corporation table, then the domain table, and then the component tables via the input screens described in Section 12.2.

The corporation table contains the required field, corporation name. It also contains optional fields for infrastructure costs, training costs, operation impact costs (op_impact), management restructuring costs (mgmt_restr), and miscellaneous costs. The
The total_up_cost field is the sum of the latter fields. The activity_date is automatically populated on creation of a corporation and indicates the start of the reuse initiative for this particular corporation. Lastly, the cost of building and maintaining the software repository (li) is contained in this table.

The domain table contains the required fields of domain name and corporation name. This table depends on the corporation being created first (and thus having an entry in the corporation table). It also has fields for activity_date, the start date of the domain initiative and ic, the domain analysis costs. The include field is not used at this time but is intended for the purposes of determining if this domain should be included in the calculations of the economic functions.

The component table contains the component name, domain name, corporation name, start date of component development (activity_date), development costs calculated using COCOMO (E), library insertion cost (li), the cost of development with reuse based on RCWR(E)(dev_cost), black box price (bbp), and white box price (wbp). The fields OCD, operating cost differential, and include, are not used at this time. The include field is intended for the purposes of determining if this domain should be included in the calculations of the economic functions.

Related and dependent on this table is the compfreq table. This table contains the same key fields of component name, domain name, and corporation name as the component table. This table contains information related to the frequency of use of the component for a particular year. The freq_bb and freq_wb fields contain the frequencies of use of the component as a black box and white box component. The currfreq_bb and currfreq_wb are for future use and indicate the actual frequency of use and are not used at this time.

The application table contains the required fields of application name and corporation name. This table also contains a field for size of the entire application (S) and start date of the application (activity_date). The overhead, prior costs (pc), and include fields are not used at this time. The include field is intended for the purposes of determining if this domain should be included in the calculations of the economic functions. The overhead and pc fields are to be used to indicate any additional overhead and prior costs associated with the application that may have been overlooked in the current calculations.

The appcompb and appcompw tables indicate the cost of use of reusable components in a particular application. These tables have the same structure except one is related to white box reuse (appcompw) and one is related to black box reuse (appcompb). Each table has the key fields for application name, corporation name, and domain name. The activity_date field indicates the date of entry of this information. The B and W fields indicate the number of lines of code in this black box or white box component and the b_cost and w_cost fields indicate the cost of development of these black box and white box components based on the number of lines of code inputted in the B and W fields.
The default_constants table contains information on the constants that are used in the calculations of the costs and benefits of the different components, domains, applications, and corporations. The key field of this table is the corporation name. Thus, the values of these constants will be used for all cycles within a particular corporation. RCWR is the relative cost of writing for reuse and has a default of 0.2. RBP is the relative black box price and is a default of 0.40. RWP is the relative white box price and is a default of 0.20. RATE is the interest rate used in the calculation of the economic functions and is a default of 0.15. ACT is the annual change traffic without reuse and is a default of 0.15. ACTP is the annual change traffic for reuse and is a default of 0.10. RAC is not used at this time and will be deleted from this table in future prototypes. These default values are populated via the default constants screen described in section 12.3 but can be modified by the user.

The ecoresult is a table to store the results of the economic function calculations for display on the Results screen of Appendix A.3. The table has fields for corporation name, application name, domain name, component name, economic function (ecofcn) and value of the economic function (value).

The results for the ecoresult screen are based on the costs and benefits stored in the temporary tables, costfactors, costfactora, and costfactord. Costfactors contains the costs and benefits the components of a corporation. This table contains the required fields component name, domain name, and corporation name. For each year of the investment cycle (year), it contains the costs (cyc) and benefits (byc) of the component(s) comprising the domain, application, and/or corporation. The fields cyc and byc for the component are calculated using the PL/SQL routines, ic, ccomp, and bcomp. The costfactora table contains the required key fields of corporation name and application name. Like costfactors it also contains fields for each year of the investment cycle of the application (year), and the associated costs (cyc) and benefits (byc). The fields byc and cyc for the application are calculated using the PL/SQL routine ica and cacomp and bacomp routines to be developed in the future. The costfactord table contains the corporation name and domain name as the key required fields as well as fields for each year of the investment cycle (year) and the associated benefits (byc) and costs (cyc). The fields byc and cyc for the domain are calculated with the PL/SQL routines, bdcomp and cdcomp.

The initial costs of the component cycle are calculated with the PL/SQL routine, ic. This routine takes the corporation, domain, and component name as inputs and returns the initial cost of the component engineering cycle in PM. This calculation is based on the library insertion cost (li) and development cost (E) from the component table and RCWR from the default_constants table and is calculated to be E(RCWR)+li.

The initial costs of the application cycle are calculated with the PL/SQL routine, ica. This routine calculated the cost of all the components associated with a particular application. The inputs to this routine are the corporation name and application name and returns the initial cost in PM. This routine selects the respective black box and white box component names from the appcompb and appcompw tables and uses these to obtain the black box and white box prices from the component table. The resulting initial cost is the
sum of the black box and white box prices of all the components associated with the particular application.

The episodic costs and episodic benefits of the component and domain engineering cycles are calculated with the routines ccomp, bcomp, cdcomp, and bdcomp. The inputs to the component functions, ccomp and bcomp, are the number of years of the investment cycle (i) and the corporation, domain, and component names. The inputs to the domain functions, cdcomp and bdcomp, are the number of years of the investment and the corporation and domain names.

The component episodic costs are determined using the PL/SQL function, ccomp. This function retrieves the start date of the component (activity_date) and number of components (numcomp) for a particular domain and corporation from the component table. It then calculates the maintenance costs based on the value of the operating cost differential from the component table (ocd) and the number of components: oc/numcomp. The domain episodic costs are determined using the PL/SQL function, cdcomp. This function does the same calculation as ccomp but does it for all of the components of the domain using the cursor, comp_cursor. These functions return the episodic costs in PM.

The component episodic benefits are calculated using the PL/SQL function, bcomp. This function retrieves the year of the start date of the component and converts this to a number. This year with the component, domain, and corporation names are used to retrieve the frequency of use of the component from the compfreq table. The sum of these frequencies for black box and white box reuse are multiplied by the black box and white box prices retrieved from the component table. The result of this calculation is returned as the episodic benefit of the component. The domain episodic benefits are calculated the same way but for all components of the domain through the use of the comp_cursor and the variable totbyc which represents the total episodic benefits of all components of the domain.

In the following two sections, 12.2 and 12.3, the archival and analytical functionality of this prototype will be described in more detail.

This prototype design is currently being submitted for funding by Dr. Ali Mili and Senta Fowler Chmiel for development of a fully functioning prototype.

12.2 Archival Function

The archival function of this automated tool keeps track of the costs and benefits of the various investment cycles. The key to this function is to track the costs and benefits of the component engineering phase.

The input screens (Appendix A1) and the associated database schema (Appendix B1) of this prototype automated tool allow for the input of the costs and benefits of the various investment cycles as well as for the display of the associated information for each
investment cycle. Each input screen allows the user to add new components, domains, applications and corporations as new entities or as clones of existing data. It also allows the user to modify or delete existing components, domains, applications, and corporations. In the following sections the input screens will be discussed in more detail for each investment cycle.

Figure 5 Archival and Analytical Functions of the Automated Tool
12.2.1 Corporation Cycle Input

The corporation input screen (Appendix A1.1) allows the user to input up-front costs and library operation costs that are associated with the overall corporation, typically by the corporate manager at the start of the reuse initiative. The following represent up-front costs:

- Infrastructure
- Training
- Operational Impact
- Management Restructuring
- Other

The sum of the latter fields is the Total field that represents the initial cost, IC, or C(SD) where SD, the start date, is the value of the Activity Date field on this form.

Infrastructure costs are those costs associated with the software library of the corporation. Some examples of reuse infrastructure costs are those such as repository, catalog and storage library mechanisms, search mechanisms, etc., that need to be in place to initiate a good reuse program.

Management restructuring costs are those costs associated with changing the structure of the organization in the initiation of a reuse program. Examples of reuse restructuring are implementing the division of labor, reporting structure, reward structure, incentive structure and team cohesion of one of the various reuse organizations: lone producer, experience factory, nested producer, pool producer, and team producer.

Training costs are the costs associated with training the producers and consumers on the keys of reuse programming and use. Producers will incur up-front costs with training personnel on the techniques of reuse programming, while consumers will be faced with the costs associated with training on how to retrieve, use (black box reuse), integrate, and possible modify (white box reuse) reusable components.

The operational impact costs are those associated with the changes in procedures that are necessary to have a reuse program. One major operational impact cost is the development of domain analysis procedures and standards. Without this change, reuse will not be as successful because the common characteristics of components will not be generalized in a consistent, uniform manner.

Other costs are any other up-front costs, not identified above, that a corporation may or may not incur.

The library operation costs are the costs to operate the reuse library. This costs are basically the costs of one or more librarians in person-months who are tasked with the cataloging, insertion, retrieval, and storage of the reusable components. This field is used in the calculation of the initial costs in the component engineering cycle.
The episodic benefits of this cycle are derived from the episodic benefits of the application cycle, \( B_a(y) \) (Section 12.1.3). The episodic costs are derived from the episodic costs of the domain engineering cycle, \( C_d(y) \) (Section 12.1.2).

### 12.2.2 Domain Cycle Input

This screen (Appendix A1.2) is the domain engineer’s mechanism to delineate the domain analysis costs at the start of the domain engineering reuse initiative. For each domain in a corporation, the initial costs, \( IC \), are the domain analysis costs plus the cost of development and cataloging of the domain component \((C_c(SD) \text{ where } SD \text{ is the Activity Date field.})\). The episodic costs, \( C_d(y) \), after the initial start date, \( SD \), is the sum of \( C_c(SD+y) \) of the component engineering phase. The episodic benefits, \( B_d(y) \), are derived from the component engineering benefits, \( B_c(y) \). The latter calculations are detailed in Chapter 9.

The domain analysis costs, typically indicated in person-months, include all the costs associated with analyzing requirements, setting up domains, and classifying components into domains.

### 12.2.3 Application Cycle Input

In this input form (Appendix A1.3) the development project manager at the end of the development project inputs the costs associated with the development of an application using reusable assets. The following are up-front costs are collected but not utilized by the existing model:

- Prior Costs
- Overhead Costs

and the following elements are collected to determine the periodic costs, \( C_a(y) \), and benefits, \( B_a(y) \), of this cycle:

- Size of Product
- Size of New Code

The cost of the components comprising the application represent the initial costs, \( IC \), at start date, \( SD \), represented by Activity Date field. The episodic benefits, \( B_a(y) \) are the costs of developing the application from scratch prorated by \( ACT-ACT' \), which are obtained from the Default Constants screen and represent the savings in maintenance due to using reuse.
The prior costs of the application engineering cycle are any costs associated with the training of personnel on the specifics of the reuse library and associated implementation language, as well as the costs to acquire tools needed to support this reuse infrastructure.

The overhead costs are those costs associated with
- Retrieval of components that do not “fit” the project
- Adaptation of a component (white box reuse) that cannot be used because it costs too much or is of poor quality
- Any costs that result in the retrieval, use, or adaptation of a component that are not adequate for the particular project.

This cost typically occurs when the library has few components, poor quality components, and/or poorly documented components.

The size of new code is the amount of code in KLOC that “glues” the black and white box components specified in the next section of the form together.

The size of the project is used to determine the costs associated with building the application from scratch versus using reuse. This value is derived from the value of the Size of New Code field and the Size and Types of components used in the application. The benefits are calculated using the equations of Section 8.3 where Dev is derived from the Size of New Code field, and RA, E and IN are derived from the Size field of the black box and white box sections of this form. The costs and size of the white box and black box components used in the calculation of the reuse costs are listed in the White box Reuse and Black box Reuse sections. In these sections the cost is derived using the defaults for RBP and RWP taken from the Default Arguments form with the COCOMO basic semi-organic equation and based on the data entered in the component cycle input form.

12.2.4 Component Cycle Input

This input form (Appendix A1.5) collects the cost and benefit information of each component. For each component entered for a particular domain and corporation, the up-front costs, periodic costs, and periodic benefits are inputted by the developer of the component. Productivity gains are quantified using the price of the component inputted by the user, used as is (black box reuse) or adapted/modified (white box reuse). In the absence of this sale price of the component, RBP and RWP and COCOMO can be used to determine these quantities based on the size of the component. In a similar fashion the quality gains are determined. However, in this case, the quality gains are quantified by using the operating costs differential that measures the savings of operating/maintenance costs. Hence the following data is collected or calculated based on the single use cost:
• Single use cost
• Development cost
• Library insertion cost
• Operating cost differential
• Black Box Price
• White Box Price

The single use cost is the “size” of the component in KLOC. This single use cost is used to determine the development cost (Development Cost field) using basic COCOMO semi-organic model and the value of RCWR from the Default Constants screen. This development cost is prorated with RBP to get the black box price and RWP to get the white box price if they are not available. The latter is used in the calculation of the benefits, \( B_c(y) \). Lastly, this single use cost is used to determine the maintenance costs (Operating Cost Differential field) by applying ACT from the Default Constants screen to its value if the operating cost differential is not available. This field, plus the Library Operating Cost field value from the Corporate Engineering form prorated per month, represents the periodic costs, \( C_c(y) \).

The library insertion cost is the cost in person-months to insert and possibly catalog the reusable component in the corporate reuse library. This field, plus the Development Cost field, represents the initial costs, IC or \( C_c(SD) \) where SD is the value of the Activity Date field, of this cycle.

The final section of this form collects information that is pertinent to the quantification of the benefits, \( B_c(y) \), of this phase and the others. For each integer year, the frequency of use as a black box component (“as is”) and white box component (adapted/modified) are inputted.

### 12.3 Analytical Function

The analytical function serves as a mechanism to report the archived data as well as to analyze this data. When the investment cycle data is fully documented for a particular phase, the investment factors of interest can be computed. The screens of the analytical function (Appendix A.2.1-A.2.4) follow the same format for each of the four investment cycles:

• Identification
• Documentation
• Cost Factors
• Investment Parameters
• Economic Functions
The identification section identifies the cycle being displayed as follows:

<table>
<thead>
<tr>
<th>Screen</th>
<th>Identification Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corporation Engineering Cycle</td>
<td>Corporation Name</td>
</tr>
<tr>
<td>Application Engineering Cycle</td>
<td>Application and Corporation Name</td>
</tr>
<tr>
<td>Domain Engineering Cycle</td>
<td>Domain and Corporation Name</td>
</tr>
<tr>
<td>Component Engineering Cycle</td>
<td>Component, Domain, and Corporation Name</td>
</tr>
</tbody>
</table>

The documentation section provides information of the main costs drivers associated with the investment cycle being reported as follows:

<table>
<thead>
<tr>
<th>Investment Cycle</th>
<th>Information in Documentation Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corporation</td>
<td>All the domains and applications associated with that corporation</td>
</tr>
<tr>
<td>Domain</td>
<td>All components associated with that domain</td>
</tr>
<tr>
<td>Application</td>
<td>All components associated with that application are classified based on whether they were reused with white box or black box reuse</td>
</tr>
<tr>
<td>Component</td>
<td>Expected/Observed frequency of reuse by year classified as white box or black box reuse</td>
</tr>
</tbody>
</table>

The cost factors section displays the investment costs, the episodic costs, and episodic benefits for each year that result from the inputs explained in Section 12.1.

The investment parameters section allows the user to input the following investment parameters:

- Duration
- Discount Rate
- Organizational Structure

The duration is the length of the investment cycle. The discount rate is the interest rate that will be applied to the economic calculations. The organizational structure is a drop-down list that allows the user to choose from the following organizations: Lone Producer, Nested Producer, Pool Producer, Team Producer, and Experience Factory.

The economic functions section displays a list of economic functions that can be checked to determine the worthiness of a reuse initiative. Presently, the following functions are available:
• Return on Investment
• Net Present Value
• Internal Rate of Return
• Average Return on Book Value
• Profitability Index
• Payback Value

After choosing an economic function and entering the investment parameters, the user can press the calculate button that calculates the results of the above functions based on the formulas in Chapter 3 and Chapters 7-10 and takes the user to the results screen (A.3). This screen identifies the corporation, domain, application, and/or component associated with the calculation and then displays each economic function chosen and its associated result.

The default arguments screen displays the defaults for a particular corporation for the following parameters that are used in the various calculations described in this thesis:

• Relative Cost of Writing for Reuse (RCWR)
• Annual Maintenance Cost (ACT)
• Annual Maintenance Cost for Reusable Assets (ACT’)
• Relative Black Box Price (RBP)
• Relative White Box Price (RWP)
• Discount Rate (Rate)

RBP and RWP are used in the calculation of the costs of black box and white box reuse in the domain and component engineering benefits and application engineering costs.

ACT-ACT’ represents the percentage of reuse maintenance costs.

RCWR is used to derive the domain engineering development costs and the component engineering development costs.

RATE is the investment rate used in the calculation of the economic functions.
Chapter 13. Conclusion
13.1 Motivation

Many different reuse economic models have been proposed to evaluate the worthiness of a reuse initiative (Chapter 5). However, no one model encompasses all of the economic decisions a stakeholder must make concerning incorporating reuse in a project. Furthermore, many of these existing models do not take into account the benefits of shortened time-to-market, quality gains, and productivity gains of reuse. The integrated cost model encompasses many of the features of existing models and expands on the existing models to quantify time-to-market.

13.2 Summary

The integrated cost model reflects the thesis that
1. There are four key decisions (corporate, component, domain, and application engineering cycles) that arise in software reuse life cycles
2. These four key decisions can be justified by an economic rationale and modeled as investments and quantified by investment analysis functions.
3. This model shows how these investment cycles feed information into each other in a cascade pattern.

Furthermore, it has been demonstrated how this model is a generalization of existing reuse cost models. Lastly, a proposed tool to support/validate this model which provides archival and analysis functions is described and illustrated.

13.3 Assessment

The integrated cost model encapsulates the existing economic models while enhancing these models with the quantification of shortened time-to-market. Managers can use this model to determine the economic worthiness of reuse projects. Furthermore, managers can determine whether reuse is worthwhile at not only the corporate level but also at the component, domain, and application levels. They can also use the model not only in archival/analytical fashion but also in a predictive manner.

13.4 Comparison

This model generalizes the existing economic models in literature by incorporating the economic evaluation of reuse at not just one decision level but all four decision levels (corporate, component, domain, and application engineering cycles). This model also accounts for maintenance costs, shortened time-to-market, quality gains, and productivity gains. However, this model assumes that all buying and selling is done within a corporation. Furthermore, it assumes that the development costs are determined by COCOMO and the average salary of the librarians, programmers, managers, etc., relative to the number of lines of code of the components comprising the applications. A further limitation of this model, as well as existing models, is the quantification of the economic
functions used to determine if the reuse initiative is worthwhile. Since code
development/application development is not a tangible product like a car, it is difficult to
quantify the benefits and costs. As a consequence, costs are associated with the
component development, while benefits are associated with the reuse of these
components to build applications. However, by assessing these costs and benefits
(though generalized in nature due to the characteristics of the product), managers can
assess the economic worthiness of a reuse initiative in a more simplistic manner than
existing models propose.

13.5 Prospects

This model addresses the economics of a reuse initiative within a company. Future work
could:

• Determine the economics of buying and selling components outside the corporation.
• Enhance the proposed prototype to automate the analytical and archival functions to
  encompass the buying and selling of assets outside the corporation.
• Enhance the proposed prototype to incorporate gains in time-to-market.
• Analyze the impact of reuse organizations on this cost model.
• Incorporate COTS into the model.
• Analyze the effect code inflation.
• Analyze the effect of earlier time-to-market with respect to market shares
Bibliography


Appendix A  Automated Tool Screens
Appendix A.1 Input Screens
Appendix A.1.1 Corporation Input Screen
Appendix A.1.2 Domain Engineering Input Screen
Appendix A.1.3 Application Engineering Input Screen
Appendix A.1.4 Component Engineering Input Screen
Appendix A.2 Report Screens
Appendix A.2.1 Corporate Engineering Report

CORPORATION CYCLE REPORT

Identification
Select corporation:

Documentation
Application
Domain

Cost Factors
IC
Costs
Benefits

Investment Parameters
Duration
Discount Rate
Organization

Economic Functions
- Return on Investment
- Net Present Value
- Average Rate of Return
- Internal Rate of Return
- Average Return on Book Value
- Profitability Index
- Payback Value

Calculate
Exit
Appendix A.2.2 Domain Engineering Report
Appendix A.2.3 Application Engineering Report

**APPLICATION CYCLE REPORT**

**Identification**
- Select corporation: [Blank]
- Select Application: [Blank]

**Documentation**
- **Black Box**
- **White Box**

**Cost Factors**
<table>
<thead>
<tr>
<th>Year</th>
<th>Costs</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Investment Parameters**
- Duration: [Blank]
- Discount Rate: [Blank]

**Economic Functions:**
- Return on Investment
- Net Present Value
- Average Rate of Return
- Internal Rate of Return
- Average Return on Book Value
- Profitability Index
- Pay Back Value

[Buttons: Calculate, Exit]
Appendix A.3 Results Screen

| Corporation: | RSI |
| Domain:      | Queues Simulation |
| Application: |                |
| Component:   | Simulation Controller |

<table>
<thead>
<tr>
<th>Economic Function</th>
<th>Result</th>
<th>Economic Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exit
Appendix A.4 Default Constants Screen
Appendix B  Reuse ROI Prototype Design

Component Engineering Cycle Parameters

Tables

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Columns</th>
<th>Constraints</th>
<th>DB Type</th>
<th>Default</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Component_Name</td>
<td>Not Null</td>
<td>varchar2(60)</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Domain_Name</td>
<td>Not Null</td>
<td>varchar2(60)</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Corporation_Name</td>
<td>Not Null</td>
<td>varchar2(60)</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Activity_Date</td>
<td>Not Null</td>
<td>Date</td>
<td>dd-mon-yyyy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RCWR</td>
<td>Number</td>
<td>1.50</td>
<td>5,2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>Number</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LI</td>
<td>Number</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCD</td>
<td>Number</td>
<td>0.72</td>
<td>5,2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Include</td>
<td>varchar2(2)</td>
<td>Hidden</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Columns</th>
<th>Constraints</th>
<th>DB Type</th>
<th>Default</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompFreq</td>
<td>Component_Name</td>
<td>Not Null</td>
<td>varchar2(60)</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Domain_Name</td>
<td>Not Null</td>
<td>varchar2(60)</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Corporation_Name</td>
<td>Not Null</td>
<td>varchar2(60)</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Year</td>
<td>Number</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Freq_bb</td>
<td>Number</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Freq_wb</td>
<td>Number</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Component PK(Component_Name,Domain_Name,Corporation_Name)
CompFreq.Component_Name(Component,Component_Name)
CompFreq.Domain_Name(Component,Domain_Name)
CompFreq.Corporation_Name(Component,Corporation_Name)
Component.Domain_Name(Domain,Domain_Name)
Component.Corporation_Name(Corporation,Corporation_Name)

Form Design
Block Name: CompQ_Blk   Type: Control  Component

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>DB Type</th>
<th>Item Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp_Name</td>
<td>Component</td>
<td>Text Box</td>
<td>Text Box</td>
</tr>
<tr>
<td>Dom_Name</td>
<td>Domain</td>
<td>Text Box</td>
<td>Text Box</td>
</tr>
<tr>
<td>Corp_Name</td>
<td>Corporation</td>
<td></td>
<td>Push Button</td>
</tr>
<tr>
<td>Add</td>
<td>Add</td>
<td></td>
<td>Push Button</td>
</tr>
<tr>
<td>Modify</td>
<td>Modify</td>
<td></td>
<td>Push Button</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete</td>
<td></td>
<td>Push Button</td>
</tr>
</tbody>
</table>

Block Name: Comp_Blk   Type: Database  Table: Component

162
Where: component.component_name =:compq_blk.component_name and component.domain_name = :compq_blk.domain_name and component.corporation_name = :compq_blk.corporation_name

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>DB Type</th>
<th>Item Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component_Name</td>
<td>Component</td>
<td>Component.Component_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Domain_Name</td>
<td>Domain</td>
<td>Component.Domain_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Corporation</td>
<td>Corporation</td>
<td>Component.Corporation_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Activity_Date</td>
<td>Date</td>
<td>Component.Activity_Date</td>
<td>Text Box</td>
</tr>
<tr>
<td>RCWR</td>
<td>Relative Cost of Writing for Reuse</td>
<td>Component.RCWR</td>
<td>Text Box</td>
</tr>
<tr>
<td>RWCR_Units</td>
<td></td>
<td>Drop Down</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Cost to Develop from Scratch</td>
<td>Component.E</td>
<td>Text Box</td>
</tr>
<tr>
<td>E_Units</td>
<td>Drop Down</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LI</td>
<td>Cost to Insert into Reuse Library</td>
<td>Component.LI</td>
<td>Text Box</td>
</tr>
<tr>
<td>LI_Units</td>
<td>Drop Down</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCD</td>
<td>Operating Cost Differential</td>
<td>Component.OCD</td>
<td>Text Box</td>
</tr>
<tr>
<td>OCD_Units</td>
<td>Drop Down</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Block Name: CompFreq_Blk     Type: Database     Component
Multi-Record Block: 5 displayed  Scroll Bar:True
Where: compfreq.component_name =:compq_blk.component_name and compfreq.domain_name = :compq_blk.domain_name and compfreq.corporation_name = :compq_blk.corporation_name

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>DB Type</th>
<th>Item Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component_Name</td>
<td>Component</td>
<td>CompFreq.Component_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Domain_Name</td>
<td>Domain</td>
<td>CompFreq.Domain_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Corporation</td>
<td>Corporation</td>
<td>CompFreq.Corporation_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Year</td>
<td>Year</td>
<td>CompFreq.year</td>
<td>Text Box</td>
</tr>
<tr>
<td>Freq_bb</td>
<td>Black Box Frequency</td>
<td>CompFreq.freq_bb</td>
<td>Text Box</td>
</tr>
<tr>
<td>Freq_wb</td>
<td>White Box Frequency</td>
<td>CompFreq.freq_wb</td>
<td>Text Box</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancel</td>
<td></td>
<td>Push Button</td>
</tr>
<tr>
<td>Save</td>
<td>Save</td>
<td></td>
<td>Push Button</td>
</tr>
<tr>
<td>Exit</td>
<td>Exit</td>
<td></td>
<td>Push Button</td>
</tr>
</tbody>
</table>

Drop Down List Definition (RWCR_Units,E_Units,LI_Units,OCD_Units):
The drop down lists for these items consist of the following:
PM
PD
PH
Alerts
Name: Add_Alert
Type: Stop
Alert Button 1: Yes
Alert Button 2: No
Alert Button 3: Cancel
Message: Do You Want to Add from Existing Data?

LOVs
Record Group: Corp_Name
Query: select distinct corporation_name from component;

Record Group: Dom_Name
Query: select distinct domain_name from component where corporation_name = :corpnm;

Record Group: Comp_Name
Query: select component_name from component where corporation_name = :corpnm and domain_name = :domnm;

LOV: Corp_LOV
Title: Corporation
Record Group: Corp_Name
Returns: comp_blk.corporation_name

LOV: Dom_LOV
Title: Domain
Record Group: Dom_Name
Returns: comp_blk.domain_name

LOV: Comp_LOV
Title: Component
Record Group: Comp_Name
Returns: comp_blk.component_name

Triggers Definitions
Delete Button

When-Button-Pressed

next_block;

Update component set include='NO' where
component_name = :compq_blk.component_name and
domain_name = :compq_blk.domain_name and
corporation_name = :compq_blk.corporation_name;

Commit;
Message(‘Component Deleted.’);
exit_form;

Add Button When-Button-Pressed
Declare
alert_button Number;
result Boolean;
begin
alert_button := Show_Alert(‘Add_Alert’);
if alert_button = ALERT_BUTTON1 then
begin
boolean := Show_Lov(‘Corp_LOV’);
boolean := Show_Lov(‘Dom_LOV’);
boolean := Show_Lov(‘Comp_LOV’);
next_block;
execute_query;
next_block;
execute_query;
previous_block;
go_item(comp_blk.activity_date);
end
elseif alert_button = ALERT_BUTTON2 then
next_block;
else
go_item(compq_blk.component_name);
message(‘Operation Cancelled’);
end if;

Modify Button When-Button-Pressed
If system.cursor_block != ‘Comp_Blk’ then
    go_block(‘comp_blk’);
    :comp_blk.rcwr_units="PM”;
    :comp_blk.e_units="PM”;
    :comp_blk.li_units="PM”;
    :comp_blk.ocd_units="PM”;
Execute_Query;
End If;

Next_Block;
Execute_Query;
Previous_Block;
GoItem(comp_blk.activity_date);
Exit
exit_form;

Save
declare
    rcwr_pm   Number;
    e_pm     Number;
    li_pm    Number;
    ocd_pm   Number;
begin
    rcwr_pm := :comp_blk.rcwr;
    if :rcwr_units = 'PD' then
        rcwr_pm := :comp_blk.rcwr/30;
    if :rcwr_units = 'PH' then
        rcwr_pm := :comp_blk.rcwr/720;
    :comp_blk.rcwr := rcwr_pm;

    e_pm := :comp_blk.e;
    if :e_units = 'PD' then
        e_pm := :comp_blk.e/30;
    if :e_units = 'PH' then
        e_pm := :comp_blk.e/720;
    :comp_blk.e := e_pm;

    li_pm := :comp_blk.li;
    if :li_units = 'PD' then
        li_pm := :comp_blk.li/30;
    if :li_units = 'PH' then
        li_pm := :comp_blk.li/720;
    :comp_blk.li := li_pm;

    ocd_pm := :comp_blk.ocd;
    if :ocd_units = 'PD' then
        ocd_pm := :comp_blk.ocd/30;
    if :ocd_units = 'PH' then
        ocd_pm := :comp_blk.ocd/720;
    :comp_blk.ocd := ocd_pm;

    commit_form;

Cancel
rollback;
exit_form;

Domain Engineering Cycle Parameters
### Tables

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Columns</th>
<th>Constraints</th>
<th>DB Type</th>
<th>Default</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Domain_Name</td>
<td>Not Null</td>
<td>varchar2(60)</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Corporation_Name</td>
<td>Not Null</td>
<td>varchar2(60)</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Activity_Date</td>
<td>Date</td>
<td>Date</td>
<td>dd-mon-yyyy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IC</td>
<td>Number</td>
<td>8,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>Number</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OC</td>
<td>Number</td>
<td>8,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RCDE</td>
<td>Number</td>
<td>2.0</td>
<td>5,2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RBP</td>
<td>Number</td>
<td>0.40</td>
<td>5,2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RWP</td>
<td>Number</td>
<td>0.20</td>
<td>5,2</td>
<td></td>
</tr>
</tbody>
</table>

Domain PK(Domain_Name,Corporation_Name)
Domain.Corporation_Name(Corporation.Corporation_Name)

### Form Design

**Block Name: DomQ_Blk**  
**Type:** Control  
**Component**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>DB Type</th>
<th>Item Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dom_Name</td>
<td>Domain</td>
<td>Domain.Domain_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Corp_Name</td>
<td>Corporation</td>
<td>Domain.Corporation_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Add</td>
<td>Add</td>
<td></td>
<td>Push Button</td>
</tr>
<tr>
<td>Modify</td>
<td>Modify</td>
<td></td>
<td>Push Button</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete</td>
<td></td>
<td>Push Button</td>
</tr>
</tbody>
</table>

**Block Name: Dom_Blk**  
**Type:** Database  
**Table:** Domain  
**Where:** Domain.domain_name = :domq_blk.domain_name and domain.corporation_name = :domq_blk.corporation_name

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>DB Type</th>
<th>Item Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain_Name</td>
<td>Domain</td>
<td>Domain.Domain_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Corporation</td>
<td>Corporation</td>
<td>Domain.Corporation_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Activity_Date</td>
<td>Date</td>
<td>Domain.Activity_Date</td>
<td>Text Box</td>
</tr>
<tr>
<td>IC</td>
<td>Domain Analysis Cost</td>
<td>Domain.IC</td>
<td>Text Box</td>
</tr>
<tr>
<td>IC_Units</td>
<td>Number of Components</td>
<td>Domain.A</td>
<td>Text Box</td>
</tr>
<tr>
<td>A</td>
<td>Operating Cost of Library</td>
<td>Domain.OC</td>
<td>Text Box</td>
</tr>
<tr>
<td>OC</td>
<td>OC_Units</td>
<td>Rcde</td>
<td>Drop Down</td>
</tr>
<tr>
<td>RCDE</td>
<td>RCDE_Units</td>
<td>RBP</td>
<td>Drop Down</td>
</tr>
<tr>
<td>RBP</td>
<td>RBP_Units</td>
<td></td>
<td>Drop Down</td>
</tr>
</tbody>
</table>
Drop Down List Definition (IC_Units,OC_Units,RCDE_Units,RBP_Units,RWP_Units):
The drop down lists for these items consist of the following:
- PM
- PD
- PH

Alerts
Name: Add_Alert
Type: Stop
Alert Button 1: Yes
Alert Button 2: No
Alert Button 3: Cancel
Message: Do You Want to Add from Existing Data?

LOVs
Record Group: Corp_Name
Query: select distinct corporation_name from domain;

Record Group: Dom_Name
Query: select domain_name from domain where corporation_name = :corpnm;

LOV: Corp_LOV
Title: Corporation
Record Group: Corp_Name
Returns: dom_blk.corporation_name

LOV: Dom_LOV
Title: Domain
Record Group: Dom_Name
Returns: dom_blk.domain_name

Triggers Definitions
Delete Button When-Button-Pressed
next_block;

Update domain set include='NO' where
domain_name = :domq_blk.domain_name and
corporation_name = :domq_blk.corporation_name;
Commit;

Message(‘Domain Deleted.’);

exit_form;

Add Button When-Button-Pressed
Declare
alert_button Number;
result Boolean;
begin
alert_button := Show_Alert(‘Add_Alert’);
if alert_button = ALERT_BUTTON1 then
begin
    boolean := Show_Lov(‘Corp_LOV’);
    boolean := Show_Lov(‘Dom_LOV’);
    next_block;
    execute_query;
    go_item(dom_blk.activity_date);
end
elseif alert_button = ALERT_BUTTON2 then
    next_block;
else
    go_item(domq_blk.domain_name);
    message(‘Operation Cancelled’);
end if;

Modify Button When-Button-Pressed
If system.cursor_block != ‘Dom_Blk’ then
    go_block(‘dom_blk’);
    :dom_blk.ic_units=”PM”;
    :dom_blk.oc_units=”PM”;
    :dom_blk.rcde_units=”PM”;
    :dom_blk.rbp_units=”PM”;
    :dom_blk.rwp_units=”PM”;
End If;

Go_Item(dom_blk.activity_date);

Exit When-Button-Pressed
exit_form;
Save When-Button-Pressed
declare
begin
  ic_pm := :dom_blk.ic;
  if :ic_units = 'PD' then
    ic_pm := :dom_blk.ic/30;
  if :ic_units = 'PH' then
    ic_pm = :dom_blk.ic/720;
  :dom_blk.ic := ic_pm;

  oc_pm := :com_blk.oc;
  if :oc_units = 'PD' then
    oc_pm := :dom_blk.oc/30;
  if :oc_units = 'PH' then
    oc_pm = :dom_blk.oc/720;
  :dom_blk.oc := oc_pm;

  rcde_pm := :dom_blk.rcde;
  if :rcde_units = 'PD' then
    rcde_pm := :dom_blk.rcde/30;
  if :rcde_units = 'PH' then
    rcde_pm = :dom_blk.rcde/720;
  :dom_blk.rcde := rcde_pm;

  rbp_pm := :dom_blk.rbp;
  if :rbp_units = 'PD' then
    rbp_pm := :dom_blk.rbp/30;
  if :rbp_units = 'PH' then
    rbp_pm = :dom_blk.rbp/720;
  :dom_blk.rbp := rbp_pm;

  rwp_pm := :dom_blk.rwp;
  if :rwp_units = 'PD' then
    rwp_pm := :dom_blk.rwp/30;
  if :rwp_units = 'PH' then
    rwp_pm = :dom_blk.rwp/720;
  :dom_blk.rwp := rwp_pm;

  commit_form;

  Cancel         When-Button-Pressed
  rollback;
exit_form;

Application Engineering Cycle Parameters

Tables

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Columns</th>
<th>Constraints</th>
<th>DB Type</th>
<th>Default</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Application_Name</td>
<td>Not Null</td>
<td>varchar2(60)</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Corporation_Name</td>
<td>Not Null</td>
<td>varchar2(60)</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Activity_Date</td>
<td></td>
<td>Date</td>
<td>dd-mon-yyyy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>Number</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>Number</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>Number</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>Number</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B_Cost</td>
<td>Number</td>
<td>8,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>W_Cost</td>
<td>Number</td>
<td>8,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S_Cost</td>
<td>Number</td>
<td>8,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Overhead</td>
<td>Number</td>
<td>8,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PC</td>
<td>Number</td>
<td>8,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Include</td>
<td></td>
<td>varchar2(2)</td>
<td>Hidden</td>
<td></td>
</tr>
</tbody>
</table>

Domain PK(Domain_Name, Corporation_Name)
Domain.Corporation_Name(Corporation.Corporation_Name)

Form Design

Block Name: AppQ_Blk   Type: Control   Component

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>DB Type</th>
<th>Item Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>App_Name</td>
<td>Application</td>
<td>Text Box</td>
<td></td>
</tr>
<tr>
<td>Corp_Name</td>
<td>Corporation</td>
<td>Text Box</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>Add</td>
<td>Push Button</td>
<td></td>
</tr>
<tr>
<td>Modify</td>
<td>Modify</td>
<td>Push Button</td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td>Delete</td>
<td>Push Button</td>
<td></td>
</tr>
</tbody>
</table>

Block Name: App_Blk   Type: Database   Table: Application
Where: Application.application_name = :appq_blk.application_name and application.corporation_name = :appq_blk.corporation_name

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>DB Type</th>
<th>Item Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application_Name</td>
<td>Application</td>
<td>Application.Application_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Corporation</td>
<td>Corporation</td>
<td>Domain.Corporation_Name</td>
<td>Text Box</td>
</tr>
<tr>
<td>Activity_Date</td>
<td>Date</td>
<td>Domain.Activity_Date</td>
<td>Text Box</td>
</tr>
<tr>
<td>P</td>
<td>Size of Software Product</td>
<td>Application.P</td>
<td>Text Box</td>
</tr>
</tbody>
</table>
P_Units
S
S_Units
B
B_Units
W
W_Units
B_Cost
BCost_Units
W_Cost
WCost_Units
S_Cost
SCost_Units
Overhead
Overhead_Units
PC
PC_Units

Cancel
Save
Exit

Drop Down
Text Box
Drop Down
Text Box
Drop Down
Text Box
Drop Down
Text Box
Drop Down
Text Box
Drop Down
Text Box
Drop Down
Text Box
Drop Down
Text Box
Drop Down
Push Button
Push Button
Push Button

Drop Down List Definition (BCost_Units,WCost_Units,SCost_Units,Overhead_Units,PC_Units):
The drop down lists for these items consist of the following:
PM
PD
PH

Drop Down List Definition (P_Units,S_Units,B_Units,W_Units):
The drop down lists for these items consist of the following:
KLOC
LOC

Alerts
Name: Add_Alert
Type: Stop
Alert Button 1: Yes
Alert Button 2: No
Alert Button 3: Cancel
Message: Do You Want to Add from Existing Data?
LOVs
Record Group: Corp_Name
Query: select distinct corporation_name from domain;

Record Group: App_Name
Query: select application_name from application where corporation_name = :corpnm;

LOV: Corp_LOV
Title: Corporation
Record Group: Corp_Name
Returns: dom_blk.corporation_name

LOV: App_LOV
Title: Application
Record Group: App_Name
Returns: app_blk.application_name

Triggers Definitions
Delete Button       When-Button-Pressed


next_block;

Update application set include='NO' where
application_name = :appq_blk.application_name and
corporation_name = :appq_blk.corporation_name;
Commit;
Message('Application Deleted.');
exit_form;

Add Button       When-Button-Pressed
Declare
alert_button Number;
result Boolean;
begin
alert_button := Show_Alert ('Add_Alert');
if alert_button = ALERT_BUTTON1 then
begin
boolean := Show_Lov('Corp_LOV');
boolean := Show_Lov('App_LOV');
next_block;
execute_query;
go_item(doapp_blk.activity_date);
end
elseif alert_button = ALERT_BUTTON2 then
    next_block;
else
    go_item(appq_blk.application_name);
    message (‘Operation Cancelled’);
end if;

Modify Button       When-Button-Pressed

  If system.cursor_block != ‘App_Blk’ then
    go_block(‘app_blk’);
    :app_blk.bcost_units="PM’;
    :app_blk.wcost_units="PM’;
    :app_blk.scost_units="PM’;
    :app_blk.overhead_units="PM’;
    :app_blk.pc_units="PM’;
  Execute_Query;
  End If;

Go_Item(app_blk.activity_date);

Exit         When-Button-Pressed
  exit_form;
Save        When-Button-Pressed

declare
  bcost_pm   Number;
  wcost_pm   Number;
  scost_pm   Number;
  overhead_pm Number;
  pc_pm    Number;
  p_kloc    Number;
  s_kloc    Number;
  b_kloc    Number;
  w_kloc    Number;

begin
  if :p_units = ‘LOC’ then
    p_kloc := :app_blk.p/1000;
    :app_blk.p := p_kloc;
  end if;
  if :s_units = ‘LOC’ then
    s_kloc := :app_blk.s/1000;
    :app_blk.s := s_kloc;
  end if;
  if :b_units = ‘LOC’ then
    b_kloc := :app_blk.b/1000;
:app_blk.b := b_kloc;
end if;
if :w_units = ‘LOC’ then
  w_kloc := :app_blk.w/1000;
  :app_blk.w := w_kloc;
end if;

bcost_pm := :app_blk.bcost;
if :bcost_units = ‘PD’ then
  bcost_pm := :app_blk.bcost/30;
if :bcost_units = ‘PH’ then
  bcost_pm = :app_blk.bcost/720;
:app_blk.bcost := bcost_pm;

wcost_pm := :com_blk.wcost;
if :wcost_units = ‘PD’ then
  wcost_pm := :app_blk.wcost/30;
if :wcost_units = ‘PH’ then
  wcost_pm = :app_blk.wcost/720;
:app_blk.wcost := wcost_pm;

scost_pm := :app_blk.scost;
if :scost_units = ‘PD’ then
  scost_pm := :app_blk.scost/30;
if :scost_units = ‘PH’ then
  scost_pm = :app_blk.scost/720;
:app_blk.scost := scost_pm;

overhead_pm := :app_blk.overhead;
if :overhead_units = ‘PD’ then
  overhead_pm := :app_blk.overhead/30;
if :overhead_units = ‘PH’ then
  overhead_pm = :app_blk.overhead/720;
:app_blk.overhead := overhead_pm;

pc_pm := :app_blk.pc;
if :pc_units = ‘PD’ then
  pc_pm := :app_blk.pc/30;
if :pc_units = ‘PH’ then
  pc_pm = :app_blk.pc/720;
:app_blk.pc := pc_pm;

commit_form;

Cancel When-Button-Pressed
rollback;
exit_form;
Appendix C  Reuse ROI Prototype Database Schema

### component

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPONENT_NAME</td>
<td></td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>DOMAIN_NAME</td>
<td></td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>CORPORATION_NAME</td>
<td></td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>ACTIVITY_DATE</td>
<td></td>
<td>DATE</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>LI</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>OCD</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>INCLUDE</td>
<td></td>
<td>VARCHAR2(2)</td>
</tr>
<tr>
<td>DEV_COST</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>BBP</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>WBP</td>
<td></td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

### compfreq

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPONENT_NAME</td>
<td></td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>DOMAIN_NAME</td>
<td></td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>CORPORATION_NAME</td>
<td></td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>YEAR</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>FREQ_BB</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>FREQ_WB</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>CURRFREQ_BB</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>CURRFREQ_WB</td>
<td></td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

### application

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_NAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>CORPORATION_NAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>ACTIVITY_DATE</td>
<td></td>
<td>DATE</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>OVERHEAD</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>PC</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>INCLUDE</td>
<td></td>
<td>VARCHAR2(2)</td>
</tr>
</tbody>
</table>

### appcompw

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_NAME</td>
<td></td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>CORPORATION_NAME</td>
<td></td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>DOMAIN_NAME</td>
<td></td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>COMPONENT_NAME</td>
<td></td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>W</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>W_COST</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>ACTIVITY_DATE</td>
<td></td>
<td>DATE</td>
</tr>
</tbody>
</table>
appcompb
Name Null? Type
------------------------------- -------- ----
APPLICATION_NAME VARCHAR2(60)
CORPORATION_NAME VARCHAR2(60)
DOMAIN_NAME VARCHAR2(60)
COMPONENT_NAME VARCHAR2(60)
B NUMBER
B_COST NUMBER
ACTIVITY_DATE DATE

domain
Name Null? Type
------------------------------- -------- ----
DOMAIN_NAME NOT NULL VARCHAR2(60)
CORPORATION_NAME NOT NULL VARCHAR2(60)
ACTIVITY_DATE DATE
INCLUDE VARCHAR2(2)
IC NUMBER(8,2)

corporation
Name Null? Type
------------------------------- -------- ----
CORPORATION_NAME NOT NULL VARCHAR2(60)
INFRASTRUCTURE NUMBER
TRAINING NUMBER
OP_IMPACT NUMBER
MGMT_RESTR NUMBER
MISC NUMBER
TOTAL_UP_COST NUMBER
ACTIVITY_DATE DATE
LI NUMBER(8)

default_constants
Name Null? Type
------------------------------- -------- ----
CORPORATION_NAME NOT NULL VARCHAR2(60)
RCWR NUMBER(5,2)
RAC NUMBER(5,2)
RBP NUMBER(5,2)
RWP NUMBER(5,2)
RATE NUMBER(5,2)
ACT NUMBER(5,2)
ACTP NUMBER(5,2)

ecoreresult
Name Null? Type
------------------------------- -------- ----
CORPORATION_NAME VARCHAR2(60)
APPLICATION_NAME VARCHAR2(60)
DOMAIN_NAME VARCHAR2(60)
COMPONENT_NAME VARCHAR2(60)
ECOFCN VARCHAR2(60)
VALUE NUMBER
<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPONENT_NAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>DOMAIN_NAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>CORPORATION_NAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(60)</td>
</tr>
<tr>
<td>YEAR</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BYC</td>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>CYC</td>
<td>NUMBER</td>
<td></td>
</tr>
</tbody>
</table>

This prototype also had the following primary keys:
- pk_factor on costfactors(component_name, domain_name, corporation, year)
- pk_factora on costfactora(corporation_name, application_name, year)
- pk_factord on costfactord(corporation_name, domain_name, year)
To calculate the initial costs for the component cycle:

```sql
function ic(corp in component.corporation_name%type,
            dom in component.domain_name%type, comp in
            component.component_name%type)
return number is
  ic   number;
  e    number;
  l    number;
  rc   number;
begin
  select li,e into l,e from component
  where component.corporation_name=corp
        and component.domain_name=dom and
        component.component_name=comp;
  select rcwr into rc from default_constants
  where default_constants.corporation_name=corp;
  ic:=e*rc +l;
return ic;
end;
```

To calculate the initial cost of the components in a particular application:

```sql
FUNCTION ica(
    corp in application.corporation_name%type,
    app in application.application_name%type) RETURN Number IS
  bp number;
  wp number;
  comp component.component_name%type;
  ic  number;
  cursor compb_cursor is
    select component_name from appcompb
    where corporation_name = corp and
    application_name = app;
  cursor compw_cursor is
    select component_name from appcompw
    where corporation_name = corp and
    application_name = app;
BEGIN
  ic :=0;
  open compb_cursor;
  loop
    fetch compb_cursor into comp;
    exit when compb_cursor%NOTFOUND;
    if compb_cursor%FOUND then
      select bbp  into bp from component
      where component.corporation_name=corp and
      component.component_name=comp;
      ic := ic + bp;
    end if;
```
end loop;
close compb_cursor;

open compw_cursor;
loop
fetch compw_cursor into comp;
exit when compw_cursor%NOTFOUND;
if compw_cursor%FOUND then
select wbp into wp from component where
component.corporation_name=corp and
component.component_name=comp;
ic := ic + wp;
end if;
end loop;
close compw_cursor;
return ic;
end;

To take a number to a power of i:

function power2(cost in number, i in number) return number is
begin
return (cost**i);
end;

To calculate the episodic costs of the domain cycle:

FUNCTION cdcomp(i in number,
corp in component.corporation_name%type,
dom in component.domain_name%type ) RETURN Number IS
J varchar(4);
D NUMBER;
cyc number;
oc number;
umcomp number;
comp component.component_name%type;
totocd number;
totcomp number;
totc number;
cursor comp_cursor is
select component_name from component
where corporation_name = corp and
domain_name = dom;
BEGIN

totocd :=0;
totcomp :=0;
open comp_cursor;
loop
fetch comp_cursor into comp;
exit when comp_cursor%NOTFOUND;
if comp_cursor%FOUND then
select to_char(activity_date,'YYYY')
into j from component
where
component.corporation_name=corp
    and component.domain_name=dom
    and component.component_name=comp;
d:=to_number(j));
d:=d+(i-1);
select count(*) into numcomp from component
where
    component.corporation_name=corp
    and component.domain_name=dom
    and component.component_name=comp;
sellect ocd into oc from component
where
    component.corporation_name=corp
    and component.domain_name=dom
    and component.component_name=comp;
totcomp :=totcomp +numcomp;
totocd :=totocd + oc;
end if;
end loop;
close comp_cursor;
totc := totocd/totcomp;
return totc;
END;

To calculate the episodic costs of the component cycle:

FUNCTION ccomp(i in number,
corp in component.corporation_name%type,
dom in component.domain_name%type,
comp in component.component_name%type) RETURN Number IS
    J varchar(4);
    D NUMBER;
    cyc number;
    oc number;
    numcomp number;
BEGIN
    select to_char(activity_date,'YYYY') into j from component where
        component.corporation_name=corp
        and component.domain_name=dom and
        component.component_name=comp;
d:=to_number(j));
d:=d+(i-1);
select count(*) into numcomp from component
where
    component.corporation_name=corp
    and component.domain_name=dom
    and component.component_name=comp;
select ocd into oc from component
where
    component.corporation_name=corp
    and component.domain_name=dom
    and component.component_name=comp;
return oc/numcomp;
END;
To calculate the episodic benefits of the domain cycle:

FUNCTION bdcomp(i in number,
corp in component.corporation_name%type,
dom in component.domain_name%type) RETURN Number IS
bp number;
wp number;
comp component.component_name%type;
freqb number;
freqw number;
J varchar(4);
D NUMBER;
byc  number;
cursor comp_cursor is
select component_name from component
where corporation_name = corp and
domain_name = dom;
totbyc number;
BEGIN
  totbyc :=0;
  open comp_cursor;
  loop
    fetch comp_cursor into comp;
    exit when comp_cursor%NOTFOUND;
    if comp_cursor%FOUND then
      select to_char(activity_date,'YYYY')
      into j
      from component where
        component.corporation_name=corp
        and component.domain_name=dom and
        component.component_name=comp;
      d:=to_number(j);
      d:=d+(i-1);
      select bbp, wbp into bp,wp from component where
        component.corporation_name=corp
        and component.domain_name=dom and
        component.component_name=comp;
      select sum(freq_bb), sum(freq_wb)
      into freqb, freqw from compfreq where
        compfreq.component_name=comp
        and compfreq.domain_name=dom
        and compfreq.corporation_name=corp
        and compfreq.year = to_char(D);
      byc := freqb*bp + freqw*wp;
      totbyc := totbyc + byc;
    end if;
  end loop;
  close comp_cursor;
  return totbyc;
END;

To calculate the episodic benefits of the component cycle:

FUNCTION bcomp(i in number,
corp in component.corporation_name%type,
dom in component.domain_name%type,
comp in component.component_name&type) RETURN Number IS
bp number;
wp number;
freqb number;
freqw number;
J varchar(4);
D NUMBER;
byc number;
BEGIN
  select to_char(activity_date,’YYYY’) into j from component where
    component.corporation_name=corp
    and component.domain_name=dom and
    component.component_name=comp;
d:=to_number(j);
d:=d+(i-1);
select bbp, wbp into bp,wp from component where
  component.corporation_name=corp
  and component.domain_name=dom and
  component.component_name=comp;
select sum(freq_bb), sum(freq_wb) into freqb, freqw from compfreq where
  compfreq.component_name=comp and
  compfreq.domain_name=dom and
  compfreq.corporation_name=corp and
  compfreq.year = to_char(D);
byc := freqb*bp + freqw*wp;
return byc;
END;
Vita
Senta Fowler Chmiel  
1085 Canyon Road  
Morgantown, WV 26508  
(304) 296-0547

OBJECTIVE:  
A job that utilizes the knowledge and skills of computer science and database administration.

EDUCATION:  
West Virginia University, College of Arts and Sciences, Morgantown, WV  
Ph.D. Computer Science, December 2000  
THESIS: An Integrated Cost Model for Software Reuse  
Upsilon Pi Epsilon  

West Virginia University, College of Arts and Sciences, Morgantown, WV  
M.S. Computer Science, August 1993  
THESIS: Access to CAD Prosthetic Parts Database Using an Information Sharing System  

Alfred University, New York State College of Ceramics, Alfred, NY  
M.S. Ceramic Engineering, May 1991  
THESIS: Development of an Electroceramic Actuator for Control of Radial Saw Blade Vibrations  

Rutgers University, College of Engineering, New Brunswick, NJ  
B.S. in Ceramic Engineering, May 1989  
B.A. in Computer Science, December 1989  
Garden State Scholar  

CERTIFICATIONS:  
ORACLE Certified Professional Program: Oracle 7.3 Database Administrator  
ORACLE Certified Professional Program: Oracle 8 Database Administrator  

INDUSTRIAL WORK EXPERIENCE:  
DATABASE COORDINATOR  
September 1998 - Present  
West Virginia University, Morgantown, WV  
Database administrator and system administrator for the WVU BANNER student information system.  
Responsibilities include creation and defragmentation of ORACLE 7.3.4 and ORACLE 8 databases on an AIX RS6000 SP platform, resizing of tables, system and database backups using ADSM, database tuning, client ORACLE installs (PC, MAC, NT), database and operating system security, and implementation of BANNER upgrades. Manage database administrators with BANNER upgrades, SQL*PLUS, Pro*C and Pro*COBOL programming, ORACLE Application Web Server, Developer 2000, SQL*NET, and database administration questions/issues. Responsible for the WVU interactive voice response system including backups, software upgrades and code modifications.  

DATABASE ADMINISTRATOR-SR  
April 1996 – September 1998  
West Virginia Network, Morgantown, WV  
Database administrator for the WVU BANNER student information system including defragmentation of ORACLE 7.3.4 database on an AIX RS6000 platform, resizing of tables, backups, database tuning, client ORACLE installs (PC, MAC, NT), and implementation of BANNER upgrades. Assist users from WVU and the other 13 state institutions with BANNER upgrades, SQL*PLUS, Pro*C and Pro*COBOL programming. SQL*FORMS 3.0, ORACLE WebServer, Developer 2000, SQL*NET, and database administration questions. Responsible for the WVU interactive voice response system including backups, software upgrades and code modifications. Successfully implemented WVU’s first client/server GUI student system utilizing Windows 95 and Macintosh clients, Windows NT file server, and AIX RS6000 Oracle Server.
SOFTWARE ENGINEER

Science Applications International Corporation/ASSET, Morgantown, WV
May 1995 - April 1996

Database Administrator of an ORACLE 7.0 database on an AIX RS6000 platform. Duties included: documenting the SRL library mechanism, maintaining the database, providing validation to the database, designing/implementing database schemas, enhancing the database (with stored procedures and functions), maintaining an electronic commerce database, modifying, designing, and implementing HTML pages on the WWW, writing UNIX scripts (including sed and awk scripts), writing SQL*PLUS, SQL*FORMS, Pro*C, and PL/SQL scripts/programs, and writing Web CGI's for applications like Web Forms.

PROGRAMMER ANALYST

Consolidated Natural Gas Transmission, Clarksburg, WV
April 1994 - May 1995

Evaluated and selected a time and attendance (computer and telephone) software vendor which included a computer site equipment analysis for over 1900 employees and a functional requirements and detailed design definitions. Rewrote existing sickness, vacation, and other benefit modules and enhanced selected time and attendance software using PowerBuilder and ORACLE 7.0.

SOFTWARE ENGINEER

D.N. American, Fairmont, WV
February 1994 - April 1994

Evaluation and analysis of data repositories and data repository software and development of a data dictionary for the IWSDB under the CALS contract.

SOFTWARE ENGINEER

New Bold Enterprises, Fairmont, WV
January 1994

Developed software in MOTIF for a Project Action Items Form under the COBRA DANE contract.

APPLICATION PROGRAMMER

Boeing Computer Support Services, Reston, VA
Summer 1992

Developed, coded, and tested MVS ORACLE forms for TOMRS, a database application of Space Station Freedom. Also updated the design book and acted as an intercessor for help, documentation, and training.

APPLICATION PROGRAMMER

Griffiss Air Force Base, Rome, NY
Summer 1991

Designed a database of reliability, maintainability/supportability, and testability tools using Paradox 3.0 and evaluated other database packages.

PROGRAMMER

Corning Inc., Corning, NY
Summer 1989

Automated FORTRAN 2D flow model programs of a glass melting tank, added code to produce graphs using PLOT79, and set up input files for tank modeling.

TECHNICIAN

IBM, East Fishkill, NY
Summer 1988

Studied hydration of precursors used in laser deposition by x-ray diffraction. Installed hardware and software and wrote application programs for the coprocessor board used to run the x-ray diffractometer by an IBM PC.

EDUCATIONAL WORK EXPERIENCE:

RESEARCH ASSISTANT

Concurrent Engineering and Resource Center, Morgantown, WV
May 1993 - December 1993

Developed an ISS gateway to medical information and tested a database for a multimedia patient record system.

TEACHING ASSISTANT

West Virginia University, Morgantown, WV

Taught a microcomputer application course lecture and lab in DOS, DBASE III PLUS, and QUATTRO.

TEACHING ASSISTANT

Alfred University, College of Ceramic Engineering, Alfred, NY
January 1990 - May 1991

Taught PASCAL, FORTRAN and application packages like MICROSOFT WORKS to freshmen.
COMPUTER SUPERVISOR
January 1988 - December 1989
Rutgers University Center for Computer Information Services, Piscataway, NJ
Debugged programs, provided technical computer support, and supervised computer facility of 8 SUN workstations, 45 Macintoshes, and 15 VAX/VMS systems.

STUDENT TECHNICIAN
July 1986 - May 1989
Rutgers University Department of Ceramics, Piscataway, NJ
Analyzed molecular dynamic simulations of alkali silicate glasses, prepared alumina coating samples, and prepared cordierite-mullite-glass samples.

SKILLS:

LANGUAGES
BASIC
FORTRAN
C
C++
JAVA

ASSEMBLY
PASCAL
APL
EXPRESS
VRML

PROLOG
LISP
SMALLTALK
HTML

SYSTEMS
MACHINE
VAX
ASSEMBLY
MassComp
UNIX

VAX
AS900
MACINTOSH
VM/CMS

SUN
IBM PC
SPARC

AIX
CMS

C APL SMALLTALK IBM PC SUN 2,4, SPARC
C++ EXPRESS HTML AIX RS6000
JAVA

FORTRAN
PASCAL
LISP

LANGUAGES
FORTRAN
PASCAL
LISP

SYSTEMS
UNIX
APPLE Ile

SUN
IBM PC
SPARC

C APL SMALLTALK IBM PC SUN 2,4, SPARC
C++ EXPRESS HTML AIX RS6000
JAVA

LANGUAGES
FORTRAN
PASCAL
LISP

SYSTEMS
UNIX
APPLE Ile

SUN
IBM PC
SPARC

C APL SMALLTALK IBM PC SUN 2,4, SPARC
C++ EXPRESS HTML AIX RS6000
JAVA

FEM PACKAGES/MODELING PACKAGES
SCADA
GPSS
FRACRTALS

MACPOISSON
IDEAS

DBMS
MVS ORACLE
VAX ORACLE PARADOX

VAX
ORACLE 7,8

EXCEL
LOTUS QUATTRO

ORACLE APPLICATIONS

MISC
MOTIF
LAN WORKPLACE
LOTUS NOTES
POWERBUILDER

JCL
POWERPOINT
SED and AWK

SPREADSHEETS
EXCEL
LOTUS QUATTRO

ORACLE WEBSERVER

WORD PROCESSING
MACWRITE II
AMNIPRO
EMACS

WORD 5.0
MICROSOFT WORKS
MICROSOFT WORD

VI
CRICKETGRAPH

MACPAINT

GRAPHICS
MACDRAFT

MACDRAFT

POWERBUILDER

PUBLICATIONS:


Fowler S., Karinthi R. Access to a CAD Repository of Prosthetic Parts Using an Information Sharing System, Presented at the International Conference on Computer Applications in Engineering and Medicine, Indianapolis, IN, March 1995.


COLLEGE/PROFESSIONAL ACTIVITIES:

TECHNICAL
Pittsburgh ORACLE Users Group: 1995-Present
International ORACLE Users Group: 1995-Present
American Ceramic Society: 1987-Present
Association for Computing Machinery: 1991-Present
IEEE Computer Society: 1991-Present
IEEE: 1990-Present
Society of Women Engineers: 1985-1996 (Treasurer 1987-88)
Materials Research Society: 1990-94

RECREATIONAL
WVU Student Grotto: 1991-1996
Fayette County Modified Softball League
Statistician: 1993-1997
Finger Lakes Association Field Hockey: 1990
North Jersey Field Hockey Club: 1987-89
Forest People: 1990-91
Outdoors Club: Caving leader (Treasurer 1988-89)

SERVICE
Harrison County YMCA Swim Instructor Volunteer: 1994-199

REFERENCES: Available upon request from WVU Career Services.