Pamela T. Geriner · Thomas R. Gulledge
William P. Hutzler (Eds.)

Software Engineering Economics and Declining Budgets

With 63 Figures

Springer-Verlag

Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Dr. Pamela T. Geriner
Economic Analysis Center
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102-3481, USA

Professor Dr. Thomas R. Gulledge
The Institute of Public Policy
George Mason University
4400 University Drive
Fairfax, VA 22030-4444, USA

Dr. William P. Hutzler
Economic Analysis Center
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102-3481, USA

ISBN-13: 978-3-642-78880-2 e-ISBN-13: 978-3-642-78878-9
DOI: 10.1007/978-3-642-78878-9

43/2202-5 4 3 2 1 0 - Printed on acid-free paper

PREFACE

This volume presents a selection of the presentations from the second annual conference on Analytical Methods in Software Engineering Economics held at The MITRE Corporation in McLean, Virginia. The papers are representative of the issues that are of interest to researchers in the economics of information systems and software engineering economics.

The MITRE Software Engineering Economics Conference was designed to address some of the new and difficult challenges that face our profession. The primary objective of these annual conferences is to address new theoretical and applications directions in Software Engineering Economics, a relatively new discipline that deals with the management and control of all segments of the software life-cycle. This collection of papers places additional emphasis on the Federal Government's Information Resource Management (IRM) initiative. The issues that relate to the economics of IRM, such as Business Re-engineering, Functional Economic Analysis, Organizational Process Modeling and the Economics of Reuse, are addressed.

We thank William R. Flury from the MITRE Economic Analysis Center for serving as a conference co-chair and everyone else who helped to make this conference a success, especially those who graciously allowed us to include their work in this volume.

Pamela T. Geriner and William P. Hutzler
Economic Analysis Center
The MITRE Corporation
McLean, Virginia 22102 USA

Thomas R. Gulledge
The Institute of Public Policy
George Mason University
Fairfax, Virginia 22030 USA

TABLE OF CONTENTS

MEASUREMENT ACTIVITY AND SEI PROCESS MATURITY LEVELS

by

Richard Werling
Software Productivity Consortium
2214 Rock Hill Road
Herndon, Virginia 22070

I. OVERVIEW

This paper suggests that improving an organization's software project measurement function is both necessary and economically effective in raising that organization's maturity level.

Measurement and measurement-related activities can provide a foundation on which organizations achieve higher process maturity levels, as defined by the Software Engineering Institute (SEI). No software development organization can progress to higher levels of process maturity until its measurement program is institutionalized. Many requirements for higher maturity levels implicitly rely on functioning measurement systems to measure properties of the software products and the software development process, derive metrics from those measurements, and support effective action based on the results (Humphrey and Sweet 1987).

The paper describes highlights of the SEI capability maturity model, demonstrates that effective measurement is essential in successful implementation of a maturity growth program, and that software measurement helps produce higher quality, more useful software products and processes, while improving the level of both process and capability maturity.

II. RAISING MATURITY LEVELS IS NECESSARY

Suppliers of systems containing software must attain higher process maturity levels to remain competitive. Recent trends in U.S. Government procurements of systems containing software make this essential. For example, prospective vendors are considered high-risk suppliers if their software process maturity is below level 2. To be responsive to procurements now in process, developers must demonstrate that their software development process meets requirements of SEI levels 2 or 3. It has been

suggested that U.S. Government acquisition organizations require aggressive action to encourage suppliers who now have level 1 software processes to improve to level 2, and require level 2 organizations to dedicate resources to improve their process to reach level 3.

An organization's "software process" is considered here to be that set of activities, methods, tools, and practices that guide its people in the production of software. It is useful to think of "process" in terms of its interaction with people, methods, and technology.

Senior management's legitimate concerns about costs are met by observing that overhead cost for a measurement program—two to four percent of cost for software development—is minor compared to the improvements in project performance.

III. SEI AND THE CONCEPT OF PROCESS MATURITY LEVEL

The Software Engineering Institute has developed two models of how organizations develop software. The first "process maturity model" (Humphrey and Sweet, 1987) gave the preliminary version of a process maturity questionnaire. This preliminary version was intended to provide "a simple tool for identifying areas where an organization's software process needed improvement. Unfortunately, the questionnaire was too often regarded as the 'model' rather than as a vehicle for exploring process maturity issues" (Paulk, 1991, vii).

In the next four years SEI, working with industry and government, evolved the software process maturity framework into a fully defined product, the capability maturity model for software (CMM). The CMM emphasizes the key practices that evidence an organization's commitment and ability to perform software development, and "... provides organizations with more effective guidance for establishing process improvement programs than was offered by the [preliminary process] maturity questionnaire." (ibid.)

The CMM model serves three operational needs. It provides: (1) an *underlying structure* for consistent, reliable assessments of software processes; (2) a vehicle for applying *process management* and *quality improvement* concepts to software development and maintenance; and (3) a *guide* for organizations to use in planning their evolution toward a culture of engineering excellence. In this latter role, the CMM is designed to help software organizations:

- characterize the *state* of their current software practice (the state of *their* art) in terms of "process"

- set *goals* for improving their process
- set *priorities* for implementing changes in their process.

A. Characteristics and definitions of software process maturity levels

Major characteristics of the five levels of SEI software process maturity are briefly summarized in Table 1, which also indicates the actions required to reach the next higher level. At the present time, more than 80 percent of software development organizations are at SEI level 1, and about 10 percent at level 2. The combined total of organizations at levels 3–5 is under ten percent, although a higher proportion of individual projects may be found at these levels.

B. Evolution of Measurement-related activities

Can this complex model be partitioned in some way to make it more tractable? The string of measurement activities can help make the model more practical to work with. Figure 1 illustrates graphically the emphasis of measurement, and how *measurement* functions evolve, for each maturity level. It shows the specific SEI maturity levels at which each function is required. For example, the function "Project estimating and Tracking", shows that estimating, measuring and tracking of *project* size, schedule, risk, and cost is required at level *2 and at all higher* maturity levels. By level three, emphasis of measurement functions expands from project to **process**, which is continued through level 5. Systematic process change can begin at level 3.

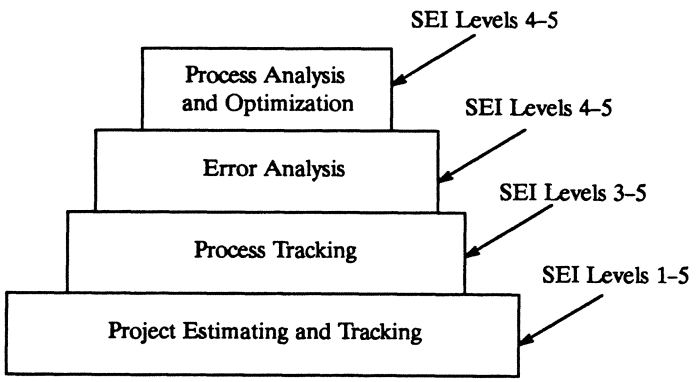


Figure 1. Measurement foundation for maturity levels 2–5.

At level 4, a second major change occurs: a managed and controlled metrics database has been put in place for process measurement, analysis, and improvement across all

Table 1. Characteristics and Definitions of Maturity Levels

Process Level	Typical Characteristics	Implications
5. Optimizing	With its process under statistical control, the organization now has a quantitative basis for continuous process improvement and optimization. Quantitative measurements are fed back into the process.	Measurements support the continuing emphasis on optimizing methods: to control; to improve activities that inject errors; and to reduce process variability. Organization's major focus is now on improving and optimizing its operations. More sophisticated analyses of error and cost data; routine error prevention and process improvement activities.
4. Managed	Use detailed measures to quantitatively understand and control software process and products. Reasonable statistical control in place over process, thus over costs, schedules, and quality of product and process.	This is when significant quality improvements begin. The organization bases its operating decisions on quantitative process data and conducts extensive analyses of data gathered during software reviews and tests. Process database in place, with data from projects throughout organization.
3. Defined	For both management and engineering activities, process for each project is now documented and standardized. All projects use a documented and approved version of the organization's standard software process.	Organization has made both organizational and methodological improvements, including design and code reviews and training programs, and has achieved an increased organizational focus on software engineering. Next need to learn to measure and analyze process performance, and to manage quality.
2. Repeatable	Basic management processes are in place to track project requirements, cost, schedule, and functionality. Process discipline helps developers repeat earlier successes with similar applications. Quality is variable.	Organization has a stable, repeatable process by improving management of: requirements; commitments; quality assurance; configuration; and subcontracts. Next needs defined standard process, peer reviews, and improved training; integrated software management and intergroup coordination.
1. Initial	Has ill-defined procedures and controls. Developers driven from crisis to crisis by unplanned priorities and unmanaged change. Surprises cause unpredictable cost, schedule, and quality. Success depends on individual heroic efforts.	Developers' process must become stabilized and repeatable before orderly process improvement is possible. Start with estimates of size, cost, risks, and schedules; performance tracking; change control; and improve management of: requirements; commitments; baseline configurations; quality assurance; and subcontracts.

projects. The emphasis is on routine measurement and analysis of software process activity, and on management of software quality. Level 5 represents a process that is well enough understood to be continually optimized. Major activities then shift to prevention of defects, management of process change and of technology innovation.

The CMM is structured to reflect the evolution of software development activities at higher maturity levels, both in depth and in detail. Table 2 shows how measurement activities evolve from levels 2 through 5. At level 1, functions typically collect insufficient data to control and manage a software project. For level 2, a minimum data set is collected and used for management. Level 3 functions add to level 2's by defining the organization's software development process and by estimating for a project's defined software process (which is obtained by tailoring the organization's standard process). Levels 4 and 5 maintain a managed and controlled database containing process metrics across all projects.

Table 2. Measurement-Related Activities Evolve Across Maturity Levels

Level 2, Repeatable Process	Level 3, Defined	Levels 4 and 5
Estimate, plan, and measure software size, resources, staffing levels, costs, risks, and schedules. Collect data on cost, size, schedule. Maintain profiles over time for: units designed; build/ release content; units completing test; units integrated; test progress, requirements changes, and staffing. Maintain profiles over time for use of target system memory, throughput, and I/O channels. Collect statistics on design *errors* and on code and test *errors* found in reviews and inspections.	Level 2 data, plus: Maintain formal records for progress of unit development. Develop standards for software measurement. Maintain formal records for test coverage. Develop experience-based metrics for estimating cost, size, and schedule. Coordinate software process asset metrics database at organization level	Levels 2 and 3 data, plus: Routinely estimate, project, compare to actuals, and analyze errors found in reviews and inspections of requirements, designs, and code and test. Routinely analyze software productivity for major process steps. Maintain managed, controlled process database for process metrics across all projects. Use metrics in systematic efforts to prevent defects, assess beneficial process innovations, and manage process change. Collect process and product data, and analyze, according to documented procedures

Level 4 then focuses on: further identifying and quantifying the organization's software development processes; defining quantitative goals for product and process quality; selecting process and product data to be collected and analyses to be performed on the data; and process and product metrics to be used in managing a project. The software product quality goals are flowed down to subcontractors.

At level 5, the organization begins the continuing task of optimizing the process. It begins by using metrics in systematic efforts to prevent defects, identify and assess beneficial process innovations, and manage process change. It institutionalizes many systematic techniques to incorporate lessons learned from process measurements.

C. Evolution of Management-related activities

A sound measurement program is necessary but not sufficient to attain higher process maturity levels. Senior management must act to standardize an organization's basic software development process. To reach level 2, authority from senior management is needed to act in three other process maturity level 2 areas:

- Provide *training*, with standard required courses for software managers, technical leaders, and especially for those who may be unfamiliar with the differences in managing system, hardware and software projects.

- Promote more *rigorous management* of software development projects by formalizing the organization's basic software development methods in standards and requiring their use, and by controlling the changes that occur as projects evolve.

- Require an effective organization for software development and ensure that independent functions are in place, for each project involving software development, for configuration control and for software quality assurance.

Table 3 illustrates the evolution for the functions of training, rigorous management, and definition and documentation of methods and standards. Training for Level 3 provides standard courses for: quality management, managing software professionals, basic software methods, inspections, and specific courses required for each job function. Level 4 and 5 organizations build on the organization's standard process, with courses on quality management, quantitative process management, advanced development technology, prototyping, and planning and development of technical careers.

Table 3. Evolution of Management Activities by Capability Maturity Level

Level 2—"Repeatable"	Level 3—"Defined"	Levels 4 and 5
Standard Courses Provided by Training Function:		
Standard **required** courses for new software managers, technical leaders; and for reviews, and inspections. Planning, estimating, and tracking product size, resource requirements, staffing levels, schedules, risks, and costs. Management of requirements, configuration,quality assurance; selection and management of subcontracts.	Quality management Managing software professionals Basic software methods Inspections Courses required for each job function which are defined in training plans Project-specific training needs	Quality management Quantitative process management Advanced development methods Prototyping Planning and development of technical careers
Organization Rigorously Manages:		
Commitments Allocated requirements, changes to requirements, and risks of software products Software Size Schedules Cost SQA planning and execution CM planning and execution	Project performance, tracked by key activity in the defined standard process Software product engineering using appropriate state-of-practice tools and methods Developing and documenting process standards and methods Development tools, methods, process definitions, and standards which are under CM for each project	Project performance against quality plan for each project Product defect levels, inspection and test coverage and efficiency, error distribution, and effectiveness of tools and methods Track and review quality performance of subcontractors Process metric definitions maintained under CM control
Basic Methods and Standards are Defined, Documented, and Followed for:		
Software development planning: estimating software size; projecting, planning, and scheduling resources Making changes to allocated requirements, designs, and code Conducting reviews, audits, and inspections	Estimating resources for each key activity in defined software process Managing risks: plans identify technical and business exposures and define process means to address them Staffing plans for special skills and application domains	Inspections, tools, methods, quality plans, and quality tracking by process task Customization of process and environment Prototyping and quantitative measures of goodness of design

IV. SEI-BASED ASSESSMENTS

Why is this structure so important to software developers? As noted before, large federal government procurements may specify that developers meet minimum process maturity levels, and may require that a "software capability evaluation" be performed by a government team to validate the process level. The basis for the team's evaluation is the same SEI process maturity model used for an internal SEI-based assessment. By performing their own SEI-based assessments, developers can guide their own process improvement work, concentrating on areas that provide them with competitive advantages.

SEI assessments begin with four to six representative project leaders completing an assessment questionnaire. The assessment team later follows through with probing, "open-ended" questions to verify that the process characteristics indicated by the project leaders' questionnaires are typical of the organization's standard processes. For an organization to be certified as having attained a particular level of software process maturity, investigation of responses to SEI assessment questions must show the presence of from 80 to 90 percent of the characteristics required for that level.

A. Currently used questionnaire

The questionnaire version used since 1987, published in (Humphrey and Sweet, 1987), has 85 process-related questions plus 16 (ungraded) questions on tools and technology. Only the 85 process-related questions are used to determine an organization's process maturity level. Table 4, Process Maturity Levels and Metrics, shows the distribution of metrics-related questions across the process maturity levels. The questionnaire's focus on measurement is important; 38 percent of the 85 items in the current (1987) version questionnaire are related to measurement.

Table 4. Process Maturity Levels and Metrics

Maturity Level	Total Number of Questions	Related To Metrics	Metrics Questions as Percent of Total
5	4	2	50
4	16	11	69
3	32	7	22
2	33	12	36
Total	85	32	38

It must be emphasized that an assessment questionnaire serves only as the starting point for process assessments. It is only a tool used by the assessment team to identify areas for more detailed on-site investigation in assessments and evaluations.

B. Highlights of the Capability Maturity Model

Currently SEI is working to provide questionnaires based on the CMM by about the end of 1992. New questionnaires, based on the CMM shown in Tables 5 and 6, are expected to be used in much the same way as the current one. To understand future, CMM-based, questionnaires requires a brief look at the CMM (Paulk et al. 1991 28,40). Table 5, "Key process areas of the SEI capability maturity model", shows a total of 18 "key process areas" that organizations must have in place to qualify for various levels of process maturity. Of the Six key process areas for level 2, seven for level 3, two for level 4, and 3 for level 5. Areas listed in *italic* type have significant measurement-related components. Acronyms, shown at the right edge of the table, are used in this section as shorthand for the full name of a key process area. For example, "CM" is used to represent "software configuration management."

Table 5. Key Process Areas of the SEI Capability Maturity Model.

Process Level	Key Process Areas*	Acronym
5. Optimizing	Prevent defects	(DP)
	Manage process change	*(PC)*
	Manage technology innovation	*(TI)*
4. Managed	*Process measurement and analysis*	*(PA)*
	Management of quality	*(QM)*
3. Defined	*Focus on organization process*	*(PF)*
	Define organization process	*(PD)*
	Training programs	*(TP)*
	Integrated software management	*(IM)*
	Software product engineering	*(PE)*
	Intergroup coordination	*(IC)*
	Peer reviews	(PR)
2. Repeatable	*Manage requirements*	*(RM)*
	Plan software projects	*(PP)*
	Track and oversee software projects	*(PT)*
	Manage software subcontracts	*(SM)*
	Software quality assurance (SQA)	*(QA)*
	Software configuration management	*(CM)*

* Key Process Areas in *italic* type are measurement-related.

C. CMM-based questionnaires

Table 5 showed that the CMM identifies 18 key process areas (KPA), such as project planning, quality assurance, peer reviews, and defect discovery and prevention. For each key process area the CMM identifies goals and a number of key practices that help realize the goals. Each key process area has five categories of key practices:

1. Practices that show a *commitment to perform* (e.g., establishing policies and procedures)

2. Practices that show an organization's *ability to perform* (e.g., training and tools),

3. *Activities performed* to guarantee that the key process is realized,

4. Practices that *monitor the implementation* of the activities (e.g., measurements),

5. Practices that *verify the implementation* of the activities (e.g., reviews and audits).

The extent of the CMM is indicated in Table 6, which shows a total of 344 practices, 91 are primarily metrics-related activities. In five key process areas (process measurement and analysis, quality management, software project tracking and oversight, integrated software management, and software project planning) the majority of practices are metrics-related.

Table 6. Extent of the Capability Maturity Model

			CMM Practices					
Maturity Level	Key Process Areas	Goals	Commit	Ability to Perform	Activities	Monitor	Verify	Sub-total
5	3	7	7	13	26	8	6	60
4	2	7	2	7	24	2	9	44
3	7	22	8	26	56	12	16	118
2	6	19	9	25	63	6	19	122
TOTAL	18	55	26	71	169	28	50	344

Table 6 shows that the CMM has a total of 18 separate key process areas. Those KPAs contain 55 separate goals. Thus, an assessment team will consult

- 26 practices that show a commitment to perform (e.g., establishing policies and procedures)
- 71 practices that show an organization's ability to perform (e.g., training and tools)
- 169 activities performed to guarantee that the key process is realized
- 28 practices that monitor the implementation of the activities (e.g., measurements)
- 50 practices that verify the implementation of the activities (e.g., reviews and audits

The level 2 KPA, "project tracking and oversight" (PT), requires tracking of: software size, costs, schedule, computer resources, risks, and software engineering technical activities. Tracking these activities helps to realize two goals explicitly stated in the CMM:

Goal 1. Actual results and performance of the software project are tracked against documented and approved plans,

Goal 2. Corrective actions are taken when actual results and performance of the software project deviate significantly from the plans.

Some of the 344 practices shown in Table 6 will become questions in the successor to the original questionnaire. SEI plans to have the CMM serve as a "map that guides the on-site investigation" for both internal process assessments and for government-sponsored software capability evaluation. Several different questionnaire versions are expected. When published, perhaps in late 1992, each new questionnaire version is expected to be about 120 questions in length.

V. MANAGEMENT AND ORGANIZATIONAL REQUIREMENTS

The four of every five software developing organizations which are at the initial level level of software process maturity today, are intensely interested in understanding the nature of the changes that must be made for them to survive the step to level 2. A single-page summary might help to understand the the magnitude of the challenge, and a strategy for making the step relatively quickly. Table 7, Summary of activities needed at SEI maturity level two, was designed to meet this need. It combines both the measurement-related and management-related activities for the functions of training, rigorous management, and basic methods and standards for the organization's software process.

Table 7. Summary of activities needed at SEI maturity level two.

Requirement	Evidence of compliance
Measurement	Formal procedures are followed to: Estimate, plan, and measure: software size, resource usage, staffing levels, development cost, schedules, and risks [software technical risks and risks for resources, schedule, and costs] from proposal throughout project life. Maintain profiles over time, compared to plan for: (a) status of each requirement allocated to software, and staffing; (b) units designed, build/release content, units completing test, units integrated, test progress, and trouble reports; (c) achievement of schedule milestones [e.g., units designed, build/release content, units completing test, units integrated, and test progress]; (d) CSCI size, work completed, effort and funds expended per CSC and CSCI; (e) critical target computer resources [utilization of target system memory, I/O channels, and throughput]; (f) cost and schedule status of software subcontracts; and (g) numbers of product reviews, process reviews, and audits. Collect statistics on design errors and on code and test errors found in reviews and inspections.
Basic Methods and Standards are Defined, Documented, and Followed for:	Software design, code, and test; estimating software size; projecting, planning, and scheduling resources Making changes to requirements, designs, and code Conducting reviews, inspections, and audits
Organization Rigorously Manages:	Commitments and risks Requirements, changes in requirements, configuration management [i.e., who can make changes to products], and size of software products Schedules and cost Quality assurance and Configuration Management
Standard Courses Provided by Training Function:	Standard, required courses for software development managers and technical leaders; and courses in conducting reviews, inspections, and audits. Planning, estimating, and tracking for: product size, checkpoint performance, resource requirements, and staffing levels Change control and configuration management Organization's process of commitment/approval/accountability Subcontract management

The CMM represents and codifies good software engineering practices. Senior management might object to the cost, staff size, and training needed to conform to the CMM. Individual line engineers might object to the CMM's emphases on measuring and controlling their activities. Nevertheless, there is no doubt that any organization which wants to win Government contracts for software development must have a software process with explicitly defined, performed, monitored, measured and verified activities. This paper has shown why a good measurement and metrics program is the correct place to begin.

VI. REFERENCES

Humphrey, W.S., and W.L. Sweet 1987 — *A Method for Assessing the Software Engineering Capability of Contractors*, CMU/SEI-87-TR-23. Pittsburgh, Pennsylvania: Software Engineering Institute.

Humphrey, W.S., D.H. Kitson, and T.C. Kasse 1989 — *The State of Software Engineering Practice: A Preliminary Report*, CMU/SEI-89-TR-1. Pittsburgh, Pennsylvania: Software Engineering Institute.

Humphrey, W.S. 1989 — *Managing the Software Process*. Reading, Massachusetts: Addison-Wesley.

Paulk, M. et al. 1991 — Capability Maturity Model for Software, *CMU/SEI-91-TR-24. Pittsburgh, Pennsylvania: Software Engineering Institute.*

Werling, R., C. McGowan, and R.D. Cruickshank 1992 — *Using Metrics to Raise Your Process Maturity Level*. Herndon, VA. Software Productivity Consortium. Tutorial given at the Fourth Annual Oregon Workshop on Software Metrics, Silver Falls, OR, March 23, 1992.

Economical Software Starts With Business

Experiencing CASE Tool Projects with Business, Industry and Government

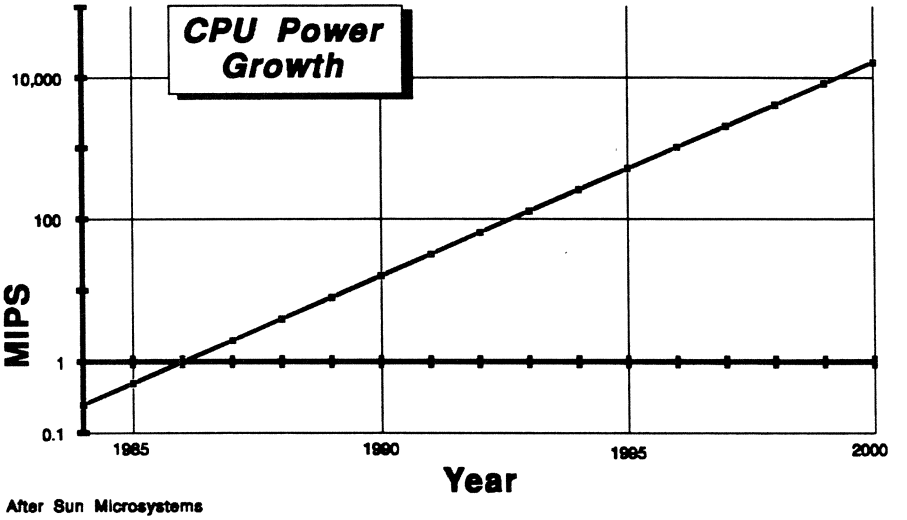
William J. Hobler Jr.
JAMES MARTIN & Co.
2100 Reston Parkway
Reston, VA 22091

1. Introduction

Productivity, quality, and flexibility are critical software engineering issues for the 1990s and beyond. Total Quality Management shows that productivity and quality are directly equivalent to speed. Speed of delivery is first in the top ten systems development issues in the minds of system development executives[1]. The late 1980s saw the world reconfigured by the fall of communism, the unification of Europe, and the emergence of the Pacific Rim nations as world class manufacturers. Much of this change can be attributed to the nature of global communications. Not only can nations see and hear conditions in other nations via television but whole industries are coordinated via telecommunications. The Boeing 747 aircraft uses parts or assemblies, whose production is coordinated using information networks, from 22 different nations from Europe west to Japan. The world wide political change has increased the complexity of business and government environments to an extent not yet fully realized. The majority of business and government software systems no longer support either efficient execution of international policy or successful competition in world markets. The most successful software in this environment is that which simplifies the human process, what is successful is software that automates complexity. The world wide electronic funds transfer system simplifies transfer of

financial resources and enabled the phenomenal increase in third world manufacturing of high technology products.

American computer hardware developers have responded to this dynamic environment by doubling the power of computing chips every year since 1984.[2]



Projected CPU power in MIPS. Note that in 1992 delivered CPUs exceed the predicted 81 MIPS.

As illustrated this expansion of power is expected to continue through the year 2000.

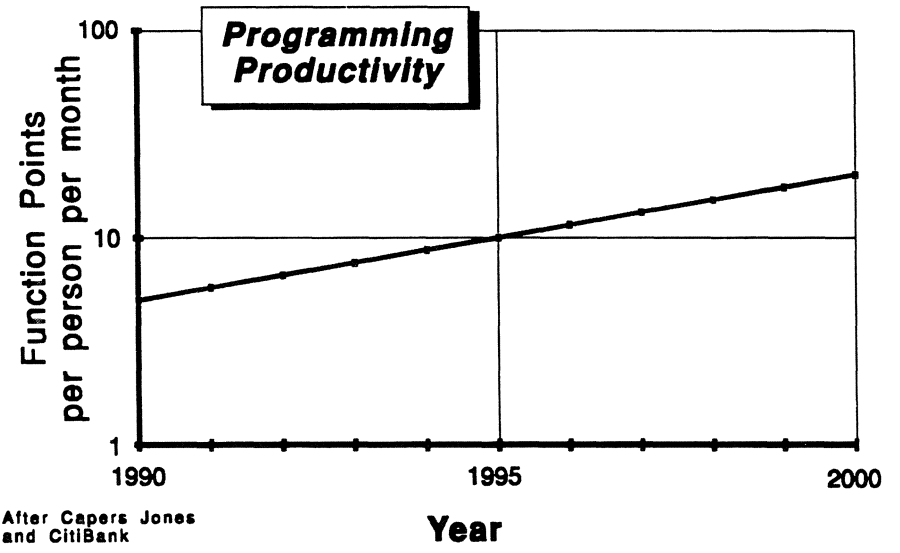
Concurrently Capers Jones[3] estimated the productivity of programmers in the United States to be 5 Function Points per person-month in 1990 and CitiBank estimates the growth in software productivity at 15% per year. The curves are dramatically different.

In effect there is more computing power available and more need to apply that power than the software industry can fill.

Software projects are required to:

- produce more complex functionality,
- in less time,
- reduce maintenance,
- increase reliability and
- increase security

with fewer resources. It is imperative that software engineers address ways of accomplishing these objectives.



Programmer productivity growth

1.1 Assertion

This paper asserts that the elements of computer systems that can radically improve business and system development productivity are in use and that concepts for the integration and use of the development systems are being applied. These systems can support enterprise engineering, business re–engineering, information systems forward engineering, and information

systems re–engineering. Further, the most economical use of software engineering is to apply it and total systems engineering practice toward integrating the existing tools and concepts into comprehensive system development platforms and methodologies to be used by non–information system trained business and government people.

1.2 Approach

The software engineering discipline must first broaden its scope into engineering software and hardware systems that support enterprise people achieve their daily objectives. Discussions concerning the viability of one language or another for specific applications pale in importance when measured against the critical need to enable global competition to improve the quality of life for both the industrialized and third world. The definition conforms to the precepts of total systems engineering in which all elements of a system are considered parts of the whole. The argument support this paper's assertion is experiential in nature. It draws from software engineering, hardware engineering, human engineering and management research and experience as documented in publications specific to those disciplines. It also draws on experiencing government and industry enterprises who are or have re-engineered their business and exploited information technology in the process.

2. Engineering Disciplines

All engineering disciplines exhibit features in common. Computer systems and software engineering, though to young to be considered mature, exhibit some of the same characteristics. Generally the characteristics are:

Graphic Guidelines. The end product concept is drawn as a guide to all productive effort. Even in building a house an architectect's drawing guides the remaining work. The architect's drawing illustrate the form that serve the buyer's function. In most disciplines this drawing is maintained as a computer file, a model that can be easily modified.

Small Teams. The overall problem is broken into small separable projects. Home construction is accomplished by small teams of plumbers, carpenters, and electricians among others. The efficiency of small teams in software development is thoroughly discussed by Dr. Frederick Brooks in the classic *Mythical Man Month*[4].

Control of Interfaces. The efforts of the various projects are coordinated at their interface. Interface control can be illustrated by the electrical industry's agreement concerning the color coding of home electrical wiring as documented in the national building code. Electrical engineers are much further advanced in setting and using standards than software engineers which makes control of electrical interfaces much easier than computer interfaces.

Standard Techniques. Codification of standard analysis techniques is required to assure a safe and high quality product. In civil engineering many of these techniques are implemented in CADD tools. Stress analysis can be accomplished on assemblies without having to fabricate and test the assembly. Thereby, less experienced engineers can accomplish complex routine work freeing the more experienced engineers for application of more advanced technology.

Intelligent Graphics. The final products are described graphically and, in some cases, these graphics serve as instructions to machines that produce the product. Perhaps the most advanced application of this concept is in production of silicon computer devices.

Standard Parts. Assemblies and sub-assemblies are constructed from standard parts. The standardization of parts allows many manufacturers compete for business and releases the engineer to assemble most of not all of the product without having to complete a detailed part by part engineering analysis. In the housing industry, the great majority of new homes are built from standard parts. It is no coincidence that building material's dimensions come in multiples of four feet.

Comprehensive Automation. Tools developed to assist the engineer provide facilities to involve the product user directly in the design and development process. Architects can walk users through the interior of a building via computer simulation. Virtual reality systems are allowing Japanese homeowners to design their own kitchens and bathrooms.

2.1 Software engineering follows these more mature disciplines

Although software engineering is still very young, the engineering features cited are all present in information systems technology.

Context Diagram. Almost all of the upper CASE (Computer Aided Systems Engineering) tools provide for drawing some form of a 'context' diagram. This representation may require more than one diagram but the information system including data, software and hardware systems describing the long term end product is illustrated. These drawings represent models of computer systems that mimic the data and activities of the business. They are, normally, not models of the business, rather models of business information systems.

Small Development Teams. Several development methodologies describe techniques or algorithms for sub-setting large computer system designs into small separable business area or business system projects. The theory of these techniques are 10 to 15 years old. These techniques are supported by several CASE tools[5].

Procedure Calls. The software industry is just adopting a philosophy that sets a standard calling procedure within a project, program, or organization. In the Object Oriented segment of the industry there is an effort to standardize the calling procedure across all Object Oriented software. This and the defacto SQL standard are examples of beginning attempts to control information system interfaces. Much more needs to be done.

Standard Techniques. Information systems development literature is replete with techniques for producing safe high quality software. CASE tools variously well implement some of these techniques. In this area the contrast between most software engineers and engineers from other disciplines is striking. Software engineers know techniques that consistently improve the quality of their products but make little use of the techniques. Capers Jones reports that although simple peer review desk checks of source code can reduce errors dramatically, a great majority of information system organizations do not use this technique. By contrast automotive engineers are adopting any measure to assure that a high quality product is delivered in as little elapsed time as possible. One possible and unfortunate conclusion could be that software engineers are not as interested in customer satisfaction as are automotive engineers.

Intelligent Graphics. The CASE tool industry has many graphic descriptions of information systems. The most mature representations are of the data base design. The CASE tool can generate all of the Data Description Language to establish the data structure directly from the diagram and its underlying documentation. Generating program logic in source code is not as mature. In most cases some sort of Process Description Language is needed to generate source code. Some CASE tools actually require that source code be written for any but the most simple data manipulation.

Standard Parts. Perhaps most encouraging information industry trend is Object Oriented concepts. One must applaud the Object Management Group in its success toward standardizing the interface to object libraries. What is needed are tools that permit designers to assemble library objects into business systems without having to employ source code. A graphical interface tool into the object manager is needed.

Comprehensive Automation. Information system developers have prototyping tools that permit working with users designing business

systems. These tools are of varying depth with the more complete prototype being fully functional systems which are ready for implementation as soon as the user accepts them.

2.2 Maturing software engineering

Relatively isolated efforts are maturing the discipline of software engineering in the direction of automation. To fully mature the software engineering discipline should emulate manufacturing engineering. Manufacturing engineering is integrating the analysis, design, and manufacturing process. Computer Integrated Manufacturing (CIM) development is being pursued on an international scale. Their goals are to;

- flexibly produce a great variety of parts and assemblies with minimum human intervention in the manufacturing process,
- transfer specifications and machine instructions across the world electronically, and
- have all the resulting assemblies uniformly work together as the intended product.

The CIM computer extends the engineer's influence into the manufacturing floor. What is needed to bring this economy to software engineering?

3. Extended Software Engineering

The concept of a Software Integrated Manufacturing (SIM) system is software engineering, expanded to include conceptualization, analysis, and manufacturing. It can realize the economy needed to meet the challenges of the 1990's because it can leverage the technical skill and knowledge of software engineers for use by business people. This goal is within reach, most of the needed concepts and tools are commercially available. The unifying concept has been discussed but the body of software engineers has not coalesced toward realizing the concept. Software engineering does not the same drive for SIM as is focussed on CIM.

3.1 Total Software Engineering Concept

The great majority of software in existence and being produced is the software supporting commerce and government. When compared to operating system software; or real time control systems; or military command, control and communications software, the software that supports the complexity of commerce and government is pedestrian. Most commercial calculations, data structures, and procedures are simple and massively redundant. Yet, in this rapidly changing world economic and business environment we are building pedestrian and redundant software at 5 function points per person per month. Software engineering must rationalize the process of producing the everyday software.

The concept is a comprehensive system that models the way business systems are conceived, analyzed, designed, produced, and maintained. The SIM must address the total system; hardware, system software, application software, and manual processes. This is a model that can be exercised by business or government people with the result that the business or government experts develop and maintain total business systems that match their exact needs over the total life cycle.

While most current Computer Aided System Engineering (CASE) tools build models of the business information system what is needed is a model of the process of engineering business systems. This viewpoint is to model the process business uses to conceive of and implement business systems. Since business models are relatively simple collections of related business data, procedures, and rules that each assume only a finite number of forms, software engineering should be able to model how these elements are assembled into useful business systems. Once this process model is developed implementing it for use by business people should be of comparable difficulty as assembling a Computer Integrated Manufacturing system for use by manufacturers.

3.2 Characteristics of software SIM

Most of the characteristics of an integrated system are present in commercially available software or existing prototypes. Many of them were developed to support businesses people using system development methodologies. These isolated concepts if aggregated, would free software engineers to pursue some of the more technical challenges of information technology.

3.2.1 Concept to Grave

Software SIM must support the business from its concept until it is replaced. The business person who develops a new business concept should be able to (1) analyze the concept to determine the potential return on investment, (2) develop the rules, procedures and data that apply, (3) separate them into human and computer based processes, (4) generate the information system (5) mount the system on any platform needed, (6) integrate the total system into the organization's operations, (7) change the system as needed by the business, and (7) replace the system as needed without interrupting business operations. These capabilities recognize information systems as one part of the business process but completely integrates them into the business.

No CASE tool provides this breadth of capability. The Integrated CASE tools start with building models of data and procedural processes and finish with generating application software on the selected platform[6]. They support modification of the software system during its lifetime. Other tools[7] permit embedding forward chaining and backward chaining rules in the procedural portions of the logic, but this must be accomplished by experienced programmers. Logic processing is much needed by commerce and government but the software industry isolates logic and procedural processing from each other. Modification of the hardware platform is difficult in that it may require substantial changes

to the application software. Yet, facilities to accommodate hardware changes are available. The most familiar of these are the numerous cross compiler, less available are facilities that support adding or deleting hardware capabilities. These facilities are present in some operating system software and in isolated application development and maintenance systems.[8]

3.2.2 No Training Required

The facilities provided the business person should be so familiar within the business context that the business user should not require training or education, or minimize training, above that of their business discipline. In the United States business people, managers and workers alike, are computer literate. Daily, they use software systems that support the theory and operation of their chosen discipline. Accountants have powerful financial accounting, controls, and analysis tools. In addition the recent emergence of Business Process Improvement and Total Quality Management techniques have introduced powerful business process analysis techniques to business and government people. The software SIM tool should extend the look and feel of this familiar software and these new techniques into expressing the data, processes and rules needed to support the business person. While comparing the Purchase Order Amount with the Invoice Amount to authorize payment the Purchasing Officer should be able to ask, why is this process needed or why does a trained accountant have to do this? Moreover, if the Purchasing Officer decides that if the Purchase Order Amount and Invoice Amount are equal then the invoice should be paid without human intervention, then the software SIM should generate the processes to pay the matched invoices. The ability to use business expressed rules as the basis for executable code was demonstrated by the RUBRIC Project[9].

3.2.3 Total System Support

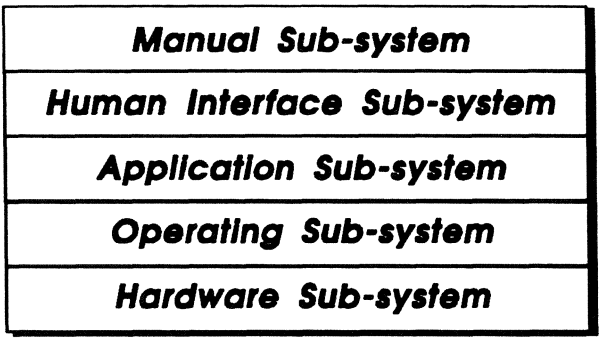
Just as manufacturing engineers recognize the manufacturing system as comprising the material handling system, the machines, the manual processes, and the people involved, software engineering must expend its view. Computer systems are only one part of the business system, a tool to assist business people in developing and operating their business systems must account for all of the elements in the business system. One decomposition of a business system into its sub-systems yields five systems.

The manual sub-system. All human enterprise has a component that can only be accomplished by human processes. In the Computer Integrated Manufacturing process, component design, raw material specification, material logistics, and many other processes are accomplished by people. Software SIM must have facilities for designating certain processes as manual and specify the system interfaces to them.

No current I-CASE tool addresses this interface from the manual sub-system view. Nor do they assist in developing the manual procedures required.

The man-machine interface sub-system. It is estimated that 80 percent of CPU cycles are devoted to graphical user interface processing. The interface sub-system is so complex a challenge that several system development tools are devoted to just this function. These tools are an indication of the capability of the software engineering discipline to leverage their skills across many less experienced developers.

The application software sub-system. This sub-system contains the business data structures, rules, and procedures supporting the business user. As far as the user is concerned this is the only sub-system visible to them.



Total Business System sub-systems

The operating software sub-system. The operating software consists of all of the system software needed to administer the application software on the hardware sub-system. In terms of Peter G. W. Keen[10] this and the hardware sub-system constitute the Information Technology Platform.

The hardware sub-system. This may be as simple as a single workstation or as complex as a world wide network. What is needed is a tool that assists in deployment of the man-machine interface sub-system and the application software sub-system into the operating software sub-system and the hardware sub-system. One of the client-server CASE tools has just such services available for a limited selection of hardware and software sub-systems.[11] The facilities of the Object Request Broker, developed by the Object Management Group, and object oriented technology hold promise of being able to deliver this type of flexibility to business.

4. A SIM CONCEPT

This concept is a collection of features currently implemented in a number of CASE tools and methodologies. There are several techniques drawn from either Business Process Improvement or Total Quality Management disciplines. The objective of bringing all of these features together into one system is to provide business people with the ability to develop and manage their information resources as integral elements of their business systems

with minimum effort and minimum recourse to specialized technical resources. The concept allows many programming staff people to migrate to business analysis and operational responsibilities.

4.1 SIM Functionality

A Software Integrated Manufacturing system should manufacture 95% of business software systems for 80% of the existing information technology platforms. This choice of numbers recognizes the commonality of procedures and data structures in business, industry, and government and the volatility of hardware and system software. A SIM capability should easily maintain pace with 95% of the business procedures used world wide, the proliferation of new technical capabilities and the lack of standardization of languages, operating systems, telecommunications protocols, and hardware provides SIM developers with a more rapidly moving technical environment.

4.1.1 DEVELOPMENT FUNCTIONS

The development environment must be graphical in nature using the interface most familiar to business users. In the graphics environment the user should be able to manipulate graphical models of the Business Process, Value Stream or Business System needed. The SIM should be able to check the validity of the models and interpret them as they would operate on the implementation platform. The business user should be able to draw a picture of what is needed and then operate an interpretation of the results. The prototyping capability is available, to some extent, in many current CASE tools. The ability to interpret graphical representations into operating prototypes is less available because most prototypes, if they are to emulate fully operational systems, require writing significant amounts of code. Early development versions of graphical interpreters existing academia and isolated commercial software developers. SIM must process graphical representations of the following Business System features into executable modules;

Graphical User Interfaces. Development software that assists X-windows™, Windows™, Presentation Manager™, and MacIntosh™ developers already exists.

Data Structures. The more traditional CASE tools use Entity Relationship Diagrams as the data modeling graphic. This diagram can directly translate into relation data base data definition language. For business people to effectively use Entity Relationship Diagram techniques requires significant training. Even though the formal training may require only a day or two it takes several months of practice before real proficiency is obtained. Object Oriented CASE tools define data structures as attributes in a Class and relationships that exist among classes. While the Object Oriented definitions are currently textual vice graphical at least one graphic based Object Oriented tool is being developed[12].

Procedural Logic. Procedural logic is currently derived from dependency diagrams and action diagrams[13]. CASE tool development efforts are expanding the dependency diagram functionality to include actions that trigger following procedures and actions that block the start of procedures until all preconditions are met. Procedural logic should be derived from business process diagramming described in Business Process Improvement literature or Process Flow Diagrams from continuous process improvement in Total Quality Management. With the exception of Action Diagrams these diagrams are familiar to most business people.

Current CASE tools either ignore procedural logic and present the business with option to process menus, edit or enter data, and produce reports or they require the business person write procedural code. The state-of-the-art tools construct program description language in an action diagram through presenting the operator with menu options specific to the process description logic being input. The SIM must move beyond this dependence on pseudo code to intelligent graphics.

Rule Based Logic. Most business and government activities are governed by rules known to and easily expressed by business and government people. The SIM should accept these rules, attach them to the appropriate procedural logic and develop operating modules that implement the rules. This capability was demonstrated in Europe in 1987[14]. Rule based programming embedded in procedural logic is a feature of the ProKappa™ tool by Intellicorp, Inc. Intellicorp is testing a graphics based version of their rules system.

Project Management. Business software development projects quickly outgrow the original team's capacity. The SIM must actively assist in defining separable small projects within a larger project. Either the data and processes must be clustered into groups that can be developed independent of other development projects or Object Classes must be so grouped. The more traditional CASE tools use affinity analysis and clustering algorithms[15] to form these independent projects.

The SIM should keep statistics concerning the completion of objects within the project so that the project can gage its status. Object completion criteria will change from project type to project type. For instance a data object such as an Entity Type need not have Attributes defined in a Strategic Planning project. This same Entity Type is required to have all attributes fully defined during the Business Area Analysis project. Some effort in this statistics area has started but most project status is maintained manually.

Process Flow Diagrams. One the most powerful business analysis techniques traces the flow of work through an organization. The SIM should automate the diagrams and information gathered from this analysis to make it easier for the business to obtain process improvements and process automation.

Security, access, and approval. Security and access are two business functions that are standard throughout business and industry. These

functions should be provided with the development environment. The concept of electronic approvals as opposed to hard copy signature approval is a fairly new concept. Electronic approval of engineering drawings are legally binding, this type of security should be provided by the development and runtime environments.

Data warehouse. Facilities for aggregating data as it is received into a corporate EIS or strategic planning data base are being individually programmed. Many of the functions required could be provided as DBMS resident procedure triggered as transactions are executed.

Image processing. Today only the most technologically advanced enterprises are benefiting from image technology. The technology is not complex and should be a facility available in the SIM.

Report generation. The formatting and production of reports should not be included in the development environment. Reports are an operating environment issue.

Information technology platform independence. The development environment products should be independent of the target technical platform. Within the development environment business people should be able to develop and prototype their business processes. The prototype must be interactively available to the team so that changes are immediately tested and approved.

This platform independence is partially delivered by some I-CASE vendors whose encyclopedias can be transported to a number of production environment and processed into executable systems on each environment. The SIM should include 80 percent or more of the commercially available production environments.

4.1.2 Operating functions

Most CASE tools provide some management of the models in their repository and some management of the generated source code.

Production system version control, deployment, and installation of production versions are largely left to the information systems operations organization. The functions are accounting and clerical in nature, amenable to automation.

Logical and physical resources. Information systems are designed around logical resources that must be related to physical resources at implementation time. The data in the logical data base must be contained on disks in a disk drive. One CASE tool in use today provides a deployment tool in which the logical resource is displayed in a column and the operator provides the identification (Names) if the physical resources in a corresponding column. The ability to deploy business systems into different physical resources is essential.

Version control. Large enterprises require significant time to deploy versions of production software to all of their people. SIM must supply administrative software that controls versions and deployment of them.

System deployment. With the proliferation of intelligent workstations the deployment of software is a logistics challenge. Several systems by which software can be deployed over the enterprise network are being used today. One vendor[16] deploys client software to the server. A server procedure updates clients as they use the server.

Report generation and management. There are three functions needed in the operating system. User design and specification of the reports should be interactive. Users should be able to see the results of their design on a graphical user interface so that there are no surprises.

SIM software should optimize report generation to reduce the time required to process long reports.

Management of report generation functions should include access to the report generation facility, approval of reports before production, and production scheduling with respect to periodicity and physical

resources. The scheduling function should include resource contention procedures.

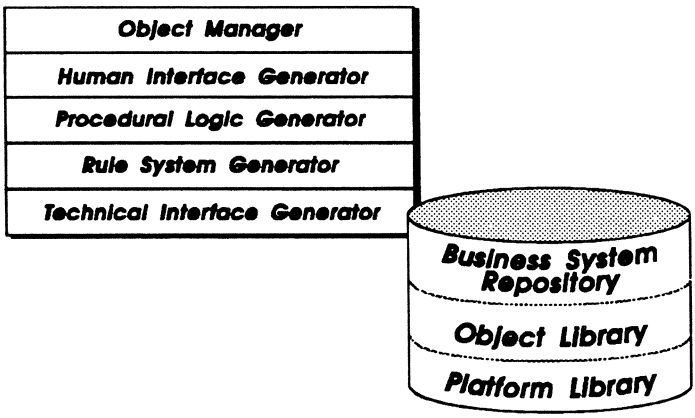
Ad-hoc queries. The production system should include a powerful graphical based ad-hoc query facility.

4.2 SIM architectures

The detailed architectures for deploying a SIM are already in place in the cited CASE tools and others becoming available to commerce and industry. Unifying these into an overall structure and rationalizing the generated systems to the operating and telecommunications systems in use is the major undertaking. The rationalization can be made using objects that interface generated systems to the target systems.

4.2.1 DEVELOPMENT ARCHITECTURES

Object Manager. Assembling the best of current CASE and software engineering technology calls for using Objects as building blocks. These must be intelligently and graphically managed. The Object Manager provides the business user–developer with two primary functions. The first is to manage the library of objects to make them easily available to the development team with the goal of assisting rapid, efficient development. The second is to permit easy addition and replacement of objects, that is to keep the library full of



Macro structure of development environment

objects that relate to current business operations. These facilities extend the concept of the Object Management Group's Object Request Broker to include management in the development environment and library management.

Human Interface Generator. All of the objects in the CUA (Common User Access) specification and all of the activities associated with them constitute a formidable design and programming challenge. The business–developer should have all of these elements at their command in a graphical design and implementation tool.

Procedural Logic Generator. This set of graphical tools makes maximum use of graphical representation for generation of code. Where the logic cannot be inferred from the graphics the business–developer should not have to type in third or fourth generation language statements.

Rule System Generator. This set of tools should also be based on using intelligent graphics for code generation. Decision trees, logic matrices and formalized if–then–else statements can all be reduced to code.

Technical Interface Generator. In the developers architecture this function is to emulate the production system technical platform so that the business–developer is in an interactive prototype environment throughout the development process. This facility should ensure that at each step in the development the developers can test functions that are complete. If they test a function that has not been completed the Technical Interface Generator must return the developer to the application ready for the next operation. The whole development environment must respond such that development can proceed without interruption.

Business System Repository. All development should be based on constructing a model of the business processes, rules, and data in a repository. The repository should have advanced facility to coordinate

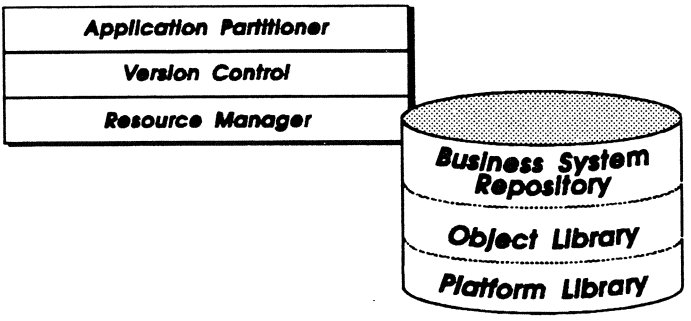
models of several development teams and to manage different versions of each model. Repository based development builds into the generated systems the ability to easily change the system by modifying the repository model.

Object Library. The objects used in development are data structures in a file. The development environment keeps the objects for ease of developer use,

Platform Library. This is a library of objects that emulate the target platform in the development environment. They support prototyping in a virtual production environment.

4.2.2 DEPLOYMENT ARCHITECTURES

SIM generated systems should be deployable to any number of technical environments. The deployment system provides this facility and the objects that make it possible. The mechanism for tailoring a SIM application to a particular platform is one of associating all of the application resource requirements to the platform resources. If the platform user interface is X-Windows, then the application is given the X-Windows object library.



Macro structure of deployment environment

Application partitioner. Platform transparency is attained by the application partitioner. During development the logical resources need by the Business System are logically named. On installation these logical names are associated with physical resources using the Resources Manager. The Application Partitioner then partitions the

Business System among the physical resources. For instance, if the objective of partitioning were to minimize the traffic on a network the partition may place all data base access software on data base servers and all application procedural logic on client machines.

Version control. The version control software will have to provide version control of the production system, modules in test, the production model, models in test, and reusable objects. This is a sophisticated set of services that must allow for merging models and management of shared objects. This facility is present in many Integrated CASE tools today[17].

Resource manager. The resource manager maintains an inventory of the hardware and system software resources available to the Business Systems. This inventory is associated with the logical resources needed by the Business Systems. The Resource System manages the Platform Library which contains all of the objects needed to interface the Business Systems to the platform.

---

1 Information Week, June 22, 1992, page 16

2 SUN MicroSystems Inc. quoted in JAMES MARTIN Insight, Inc. 'Object Oriented Methodology Concepts 1992

3 Capers Jones, *Applied Software Measurement,* McGraw-Hill 1991

4 Frederick P. Brooks, Jr, *Mythical Man Month,* Addison-Wesley Publishing Co. 1975

5 For an example of a CASE tool that supports intelligent subsetting see the ADW™ by Knowledgeware, Inc.

6 For an example of an Integrated CASE tool of this capability see the IEF™ byTexas Instruments, Inc.

7 See the ProKappa™ tool by Intellicorp, Inc.

8 See the Ellipse™, from Cooperative Solutions, Inc., system management facilities.

9 ESPRIT Project 928:RUBRIC-Six Monthly Report February 1987

10 Peter G. W. Keen, *Shaping the Future,* Harvard Buiness Review Press 1992

[11] See Ellipse™ by Cooperative Solutions, Inc.

[12] See Intellicorp, Inc. Object Management Tool development.

[13] For definitions of these diagrams see James Martin and Carma McClure, *Diagramming Techniques for Analysts and Programmers,* Prentice-Hall, Inc. 1985

[14] See ESPRIT Project 928:RUBRIC-Six Monthly Report February 1987

[15] See IEF™ by Texas Instruments, Inc. or ADW™ by Knowledgeware, Inc.

[16] See Ellipse™ by Cooperative Solutions, Inc.

[17] See Texas Instruments Inc., Knowledgeware Inc., or the Bachman tool sets.

The Challenge Of Managing And Developing A Very Large And Dynamic Management Information System

Dr. Palmer W. Smith[1]
BDM International, Inc.
1900 Founders Drive
Kettering, Ohio 45420 USA

I. INTRODUCTION

General Bernard Randolph once said, according to Alton Marsh writing in *Government Executive* [1], that he retired as Commander of the Air Force Systems Command without ever completing a software project on time. His reference was to embedded systems. But frankly, there is little evidence to suggest that anyone, or a very fortunate few, can say they have completed a software development project of any real size or kind on schedule and within cost. In her presentation at the National CASECON in 1989 [2], Lois Zells reported that she found estimates for Management Information Systems (MIS) development are commonly off by 400 to 1000 percent before a detailed analysis is completed. In addition, her extensive research showed an average 50 percent error in cost estimates, even after a detailed design is completed. No wonder quality, costs, schedule, usability, meeting user requirements, etc., are of continuing constant concern when developing a MIS.

BDM International, Inc. (BDM), in its eighth year of a ten year, $240 million Logistics MIS development program for the Air Force, has experienced the full range of challenges associated with these concerns and responded with implementing new technology into a "back to basics" solid and disciplined management approach. This paper discusses the problems, challenges, lessons learned and actions taken to turn a very large logistics system MIS development into a highly successful program with proven

---

[1] Many people have lived through the challenges and "excitement" of building and continuing to build the Air Force Requirements Data Bank (RDB). Many people contributed ideas, worked at ironing out problems, tuning concepts, and integrating the approaches discussed here. This author has simply the privilege of documenting others' ideas, hard work and efforts. Special acknowledgment is given to Donald E. Harter, BDM RDB Program Manager and Mark Filteau, formerly of BDM, for the vision to plan and implement the concepts in this paper. Even now, with two years to go, the challenges are ever present and the management intense.

results, high quality software, on schedule and on or under cost in a firm fixed price incentive fee environment.

II. BACKGROUND

In 1984, BDM began to build one of the major components of the Air Force (AF) Logistics Modernization System (LMS), the Requirements Data Bank (RDB). For development purposes, the AF had broken its total logistics process into three major components and several minor components in order to completely modernize its ability to support and sustain AF weapon systems and their associated infrastructure into the 21st Century. The LMS was a well thought out and planned $1.2 billion effort, the largest and most comprehensive military logistics management information system development project ever undertaken. The reasons for developing the LMS by components which were functionally and data integrated, but not physically integrated, is obvious to anyone who has ever had to tackle even a moderate LMS development project.

The purpose of the RDB is to automate and integrate the Air Force logistics process to forecast, budget, execute, and manage procurement and repair requirements for material to support all AF weapon systems, equipment, and other major end items. The RDB replaced 18 existing major logistics systems, is expected to have a 680+ gigabyte

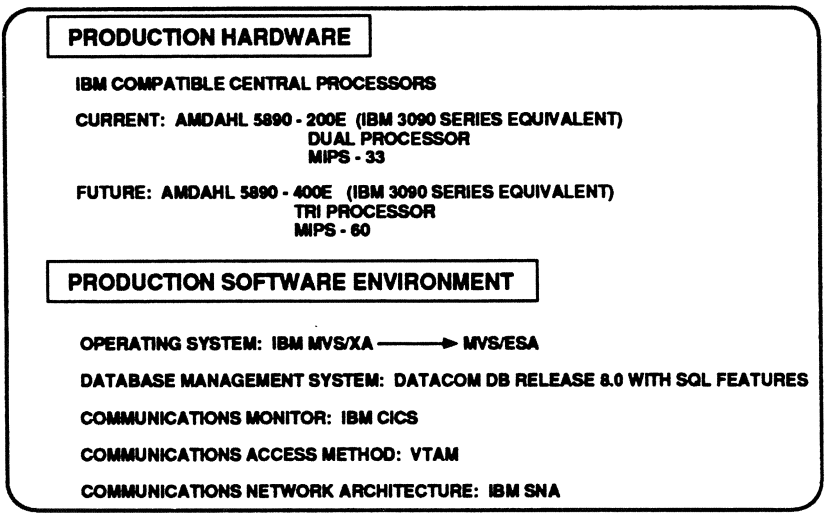


Figure 1. RDB Production Hardware & Software Environment

relational data base with over 26,000 data elements, over 4.2 million lines of code, over 1500 screens and reports, and about 16,000 on line users spread across the nation, when fully implemented in October 1994. It is being developed with the most comprehensive and complete DOD-STD-7935A documentation ever. It currently has over 500,000 pages, and growing, of documentation under configuration management. Figure 1 gives the RDB production hardware and software environment.

With only half of the system implemented in production, the RDB already has proven its value during the recent Desert Shield and Desert Storm. And it has been chosen as the basic platform and data base for the Requirements functionality of the Department of Defense (DOD) Logistics Corporate Information Management (CIM) initiative by the Joint Logistics Systems Center (JLSC).

III. DYNAMIC CHANGE AND GROWTH

During the first three years of the RDB program, a dynamic environment negated the ability to truly baseline system requirements. The initial requirements documentation consisted of over 3,700 pages with an estimated system size of 800,000 lines of code in 1984. A distributed architecture was planned for both processing and data storage. Three years later, the requirements documentation consisted of over 13,700 pages with an estimated system size of about 4,200,000 lines of code. During this time over 1,257 Baseline Change Requests (BCRs) were received and the architecture redesigned for centralized processing against a large centralized relational data base.

The BDM management team and the client were not ready for this dynamic environment which produced many initial management and development concerns as shown in Figure 2. Most, if not all, of the normal problems experienced on any large MIS project were present, along with cost increases and schedule slippage. In addition, the AF requested that the Cost-Plus-Fixed-Fee contract be converted to a Firm Fixed Price contract for the remaining seven years since is was agreed that requirements were much better defined. It was difficult to envision providing a customer a firm estimate for the remaining 3,200,000 lines of code seven years in advance. Yet, it was a requirement.

During this three year period many lessons were learned and relearned and concrete actions taken to gain and maintain control of requirements, design, quality, cost and schedule, and customer expectations. Part of the solution involved the introduction of new

Computer-Aided Software Engineering (CASE) technology, but this was not the real key. The center and focus of the solution was the implementation of a strict management disciplined environment within which the CASE technology was embedded and which was coordinated with the customer, i.e., going back to basics.

IV. LESSONS LEARNED AND ACTIONS TAKEN

The lessons learned and the management and development actions taken to transform this effort into a success are given in Figure 3. These are discussed in more detail in the following paragraphs in the form of the BDM's strategy and tactics evolved over time to face and solve the problems. In addition, because of the anticipated size and consideration of the thousands of AF staff who would be impacted by the RDB, an evolutionary design process and phased implementation were developed (Figure 4). This was done with a better understanding of the total impact on the customer's culture, the complex user/machine system which would require significant "change" management, and the critical need to manage customer expectations over the life of system development and implementation.

A. Major Considerations in Developing The BDM Strategy

In developing a strategy it is important to realize that the quality and cost management of an information system is determined by four factors. The first is user perceptions of requirements. This major difficulty encountered in costing and developing large systems is the "I'll know it when I see it" phenomenon. Another way to state this phenomenon is: "I don't know for sure what I want, but I want you to develop something. As you do I'll tell you whether or not it is what I want and whether I want you to change it." A major research project of 116 organizations found the major cause of poor estimates of cost and schedule was just this problem, changing user requirements [3] Figure 5 demonstrates the balance required if reliable estimates are to be achievable. A static engineering environment may provide a high probability of meeting cost and schedule, but guarantees the user will not be satisfied with the result. On the other hand, a dynamic environment which someday may satisfy the user will guarantee a cost overrun and schedule slippage. A balance is to establish a product baseline upon which reasonable estimates can be made with allowances for limited change proposals. ***The battle has to be fought here.*** Software quality and costs can only be defined in terms of an established baseline.

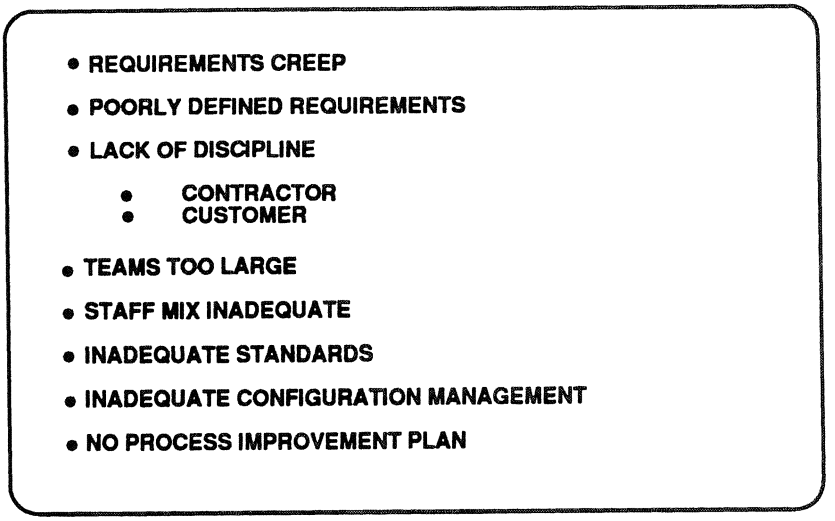


Figure 2. Initial Management and Development Concerns

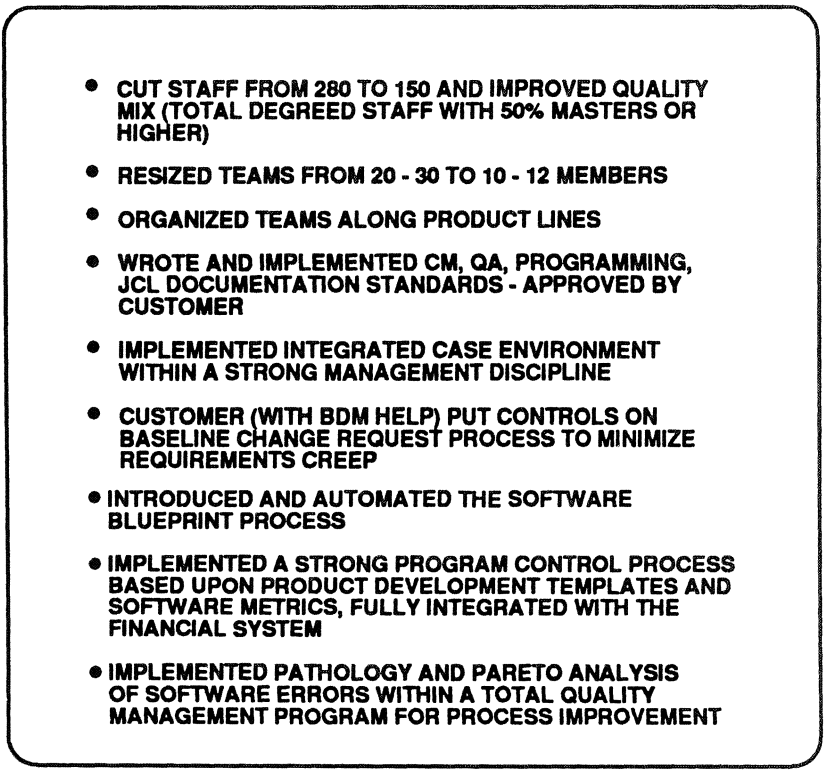


Figure 3. Lessons Learned and Concrete Actions to Improve Management and Total Development Process

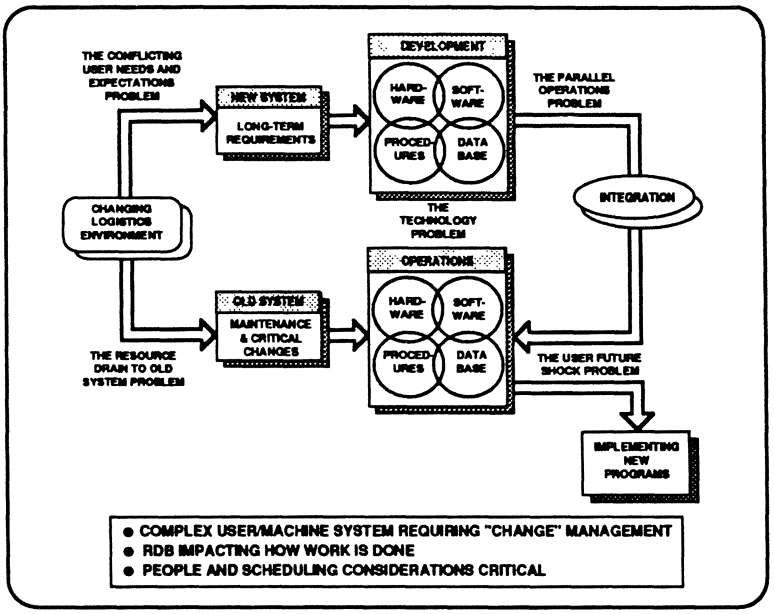


Figure 4. RDB Requires An Evolutionary Design Process And Implementation

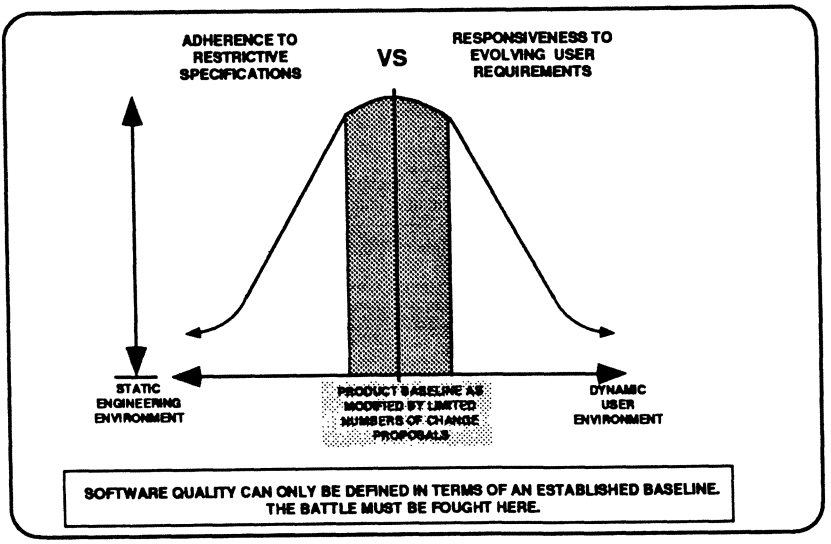


Figure 5. Users Have Great Difficulty Articulating Requirements

Complexity, the second major factor affecting cost and quality is an error present in all estimates (Figure 6). The system always seems simpler than it is in reality. Just throw some lines of code together and that is it! Information systems are among the most complex artifacts built by man. A failure to recognize this often results in poor resource allocation due to the serious error of misjudging complexity. An F-15 Aircraft is 75,000 parts flying in close formation with one pilot, a complex machine and environment. But a very large MIS like the RDB, with millions of lines of code, a 680 gigabyte relational data base, and hundreds of interfaces working in close formation and the potential of 16,000 pilots on-line all over the country. A complexity which, if ignored, guarantees many future long term problems, difficulty in accomplishing the purpose of the MIS, the wrong kind of visibility, and an unnecessary expenditure of scarce resources.

The third factor, the economics of error correction, dominates software development quality and costing (Figure 7). The rush to meet schedules and come in on or under budget is prevalent today. Its symptoms are poor requirements documentation and analysis, sketchy or missing design documents, and plenty of code with plenty of errors that need fixing after the fact. A planned strategy aimed at reducing high cost errors in analysis and design is mandatory in order to provide a fighting chance of meeting some cost and schedule estimate. The key is a disciplined management and technical design environment with embedded automated CASE tools. Here the team player and conformist are valuable resources.

Politics is the fourth factor (Figure 8). Information systems have the potential to alter the balance of power in organizations. Building an MIS must be regarded as more of a political act than a technology act and managed in a flexible, evolutionary manner. The failure to recognize this fact has resulted in an MIS which is not used or is canceled during development.

Another major consideration in developing a strategy is a software engineering process based upon the client's Concept of Operations. The BDM strategy focuses on providing the client visibility that senior management's objectives are being met, how these objectives will be supported, what the user will see, and how BDM's approach will limit risk (Figure 9).

The product-oriented environment allows development teams to stay with a subsystem from analysis until independent system test began. Some continue through the

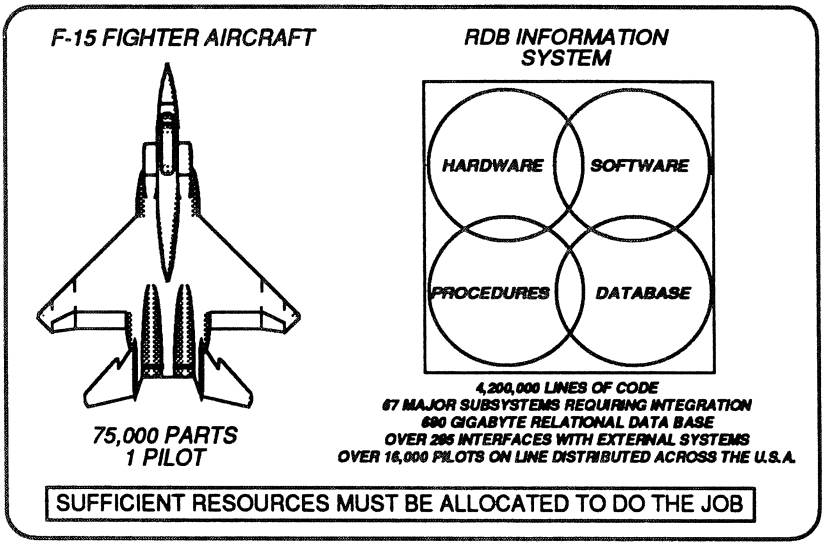


Figure 6. Information Systems Are Among
The Most Complex Artifacts Built By Man

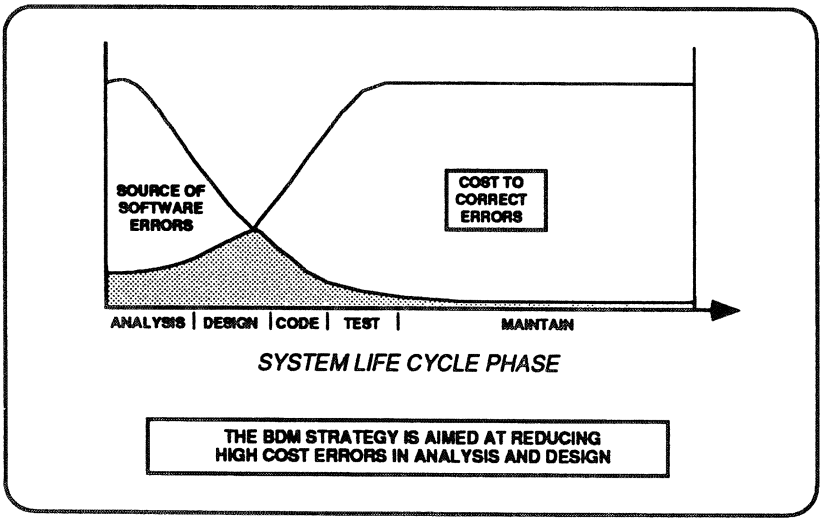


Figure 7. Software Development Is Dominated
By The Economics Of Error Correction

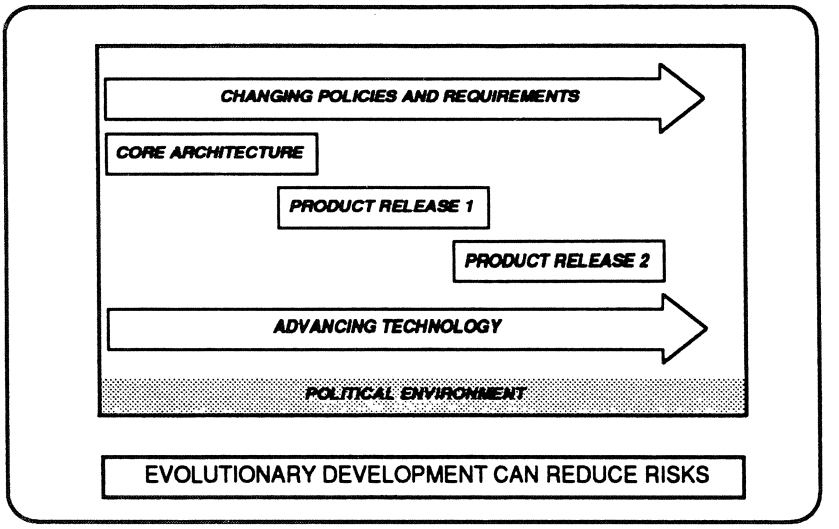


Figure 8. Building An MS Is A Political Act And
Should Be Managed Accordingly

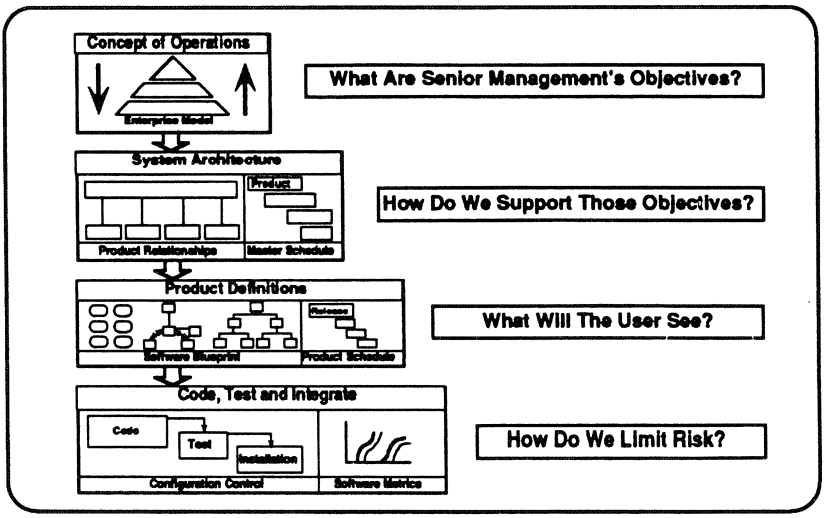


Figure 9. The Software Engineering Process
Is Based On The Client's
Concept Of Operations

client's Independent Verification and Validation (IV&V) testing to fix any errors found in the design or coding. This allows a team to retain functional, design, and system expertise in addition to the satisfaction of seeing their part of the product through to completion. The overall result is better morale, higher quality and productivity, improved processes, and reduced life cycle cost.

In addition, under the product-oriented management concept, a separate architecture team has responsibility for system integration issues, system testing, and data base design, configuration management, and documentation control This forces communications between the product teams since each team has to work with all three functional areas of the architecture team. The director of the architecture team also chairs the Configuration Control Board (CCB).

A separate program control support team provides monitoring of schedules and costs, develops all PERT/CPM networks for each product with the product manager, develops and maintains software metrics based upon data provided by the product teams, and conducts independent cost and variance analysis. Additional communication and oversight is provided by the conduct of a weekly Program Management Status Review (PMSR), when each product manager presents the status of his/her product in the presence of other peers and representatives from all other organizational components.

It is important to emphasize that the implementation of CASE tools does not provide higher productivity or even better quality unless embedded within a strict management discipline. Productivity tools are not a substitute for good engineering practice or management discipline.

B. The Tactics

BDM's tactics are used to implement the BDM strategy and are focused on software product quality. Every facet of the software development process such as user training, software engineering, quality assurance, program planning and control, product testing, configuration management, project organization, and BDM's Software Productivity Enhancement Center (SPEC)^{TM} are focused on the concept of providing the Requisite Quality and Quantity on Time for Controlled Costs (RQ^2TC^2) for a product. Success is dependent upon clear articulation of tactics and strong management support. Each team player, regardless of their area of concern, must understand the tactics and

know that management will be involved. The following paragraphs discuss components of the tactics.

C. The Strategy

Considering the above, BDM's strategy, developed out of the lessons learned, has three key drivers: staffing policy, productivity tools, and management discipline (Figure 10). As shown in Figure 3, BDM had about 280 staff on the RDB program very early in its life cycle. This staff was reduced to about 150 near the end of the first three years. This was the result of changing the staffing policy to move towards the BDM normal professional staff mix of 50 percent with advanced degrees and providing the best tools available. The result - higher productivity than with the 280 staff members and higher quality software products. Immersion of the quality staff and new technology within a strict management discipline and having a product-oriented management structure completed the strategy.

The project organization is based on the concept of "checks and balances" (Figure 11). The organization concept involves responsibilities with a system focus, such as systems integration; responsibilities with a user focus, such as system test; and responsibilities with a product focus, such as each Computer Program Configuration Item (CPCI) team director focused on product development.

The Quality Assurance (QA) assessment approach is proactive. QA influences all aspects of development through supporting the planning and building in of quality and the planning and performing of quality evaluation (Figure 12). Evaluation of products and requirements, as well as methodologies, provides key feedback for consideration in process improvement.

Program Control, which is responsible for the scheduling and cost driver portion of the strategy, is also based on a concept of checks and balances (Figure 13). The development of a Work Breakdown Structure (WBS) based upon contract requirements and mapped to process descriptions within the requirements documentation supports the development of a detailed product plan for each team. This effort is conducted by the Program Control team. The product team uses a Function Point costing model to evaluate and check cost estimates, the RDB master schedule to determine product beginning and end development time, and a standard product schedule template for developing

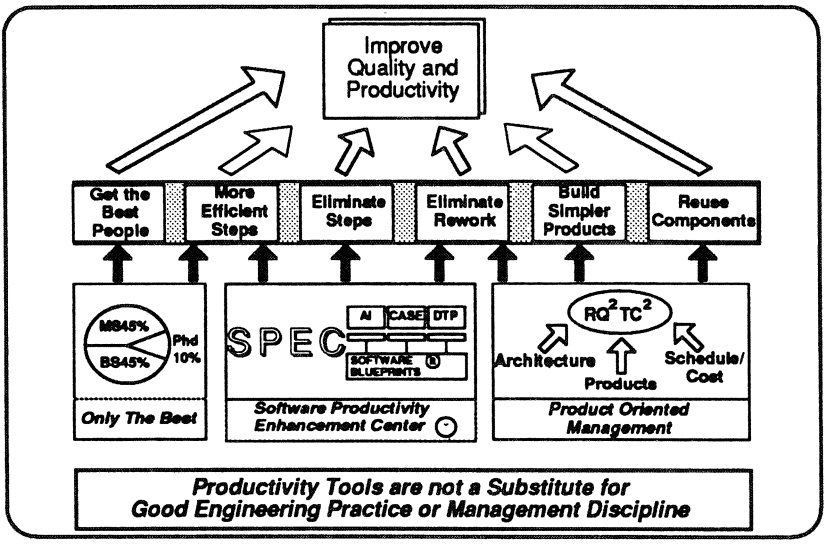


Figure 10. Staffing Policy, Productivity Tools And Management Discipline Are The Key Drivers Of The EDM Strategy

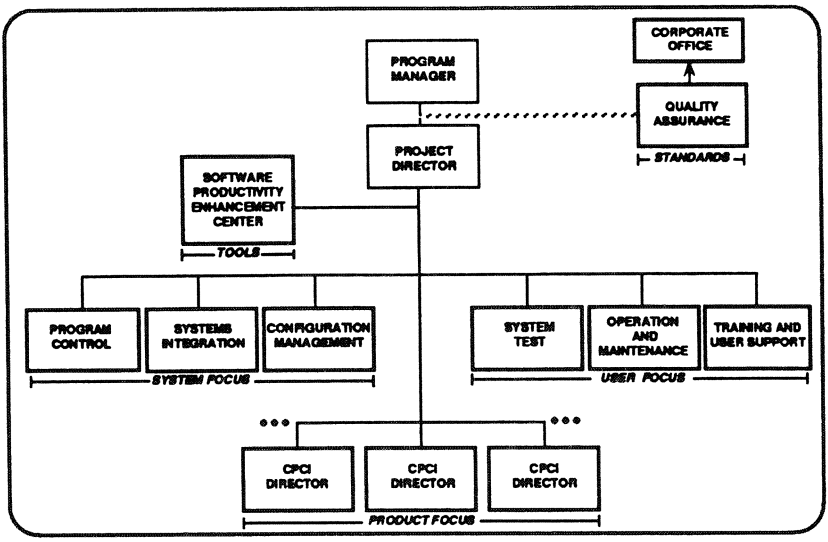


Figure 11. The Project Organization is Based on the Concept of "Checks and Balances"

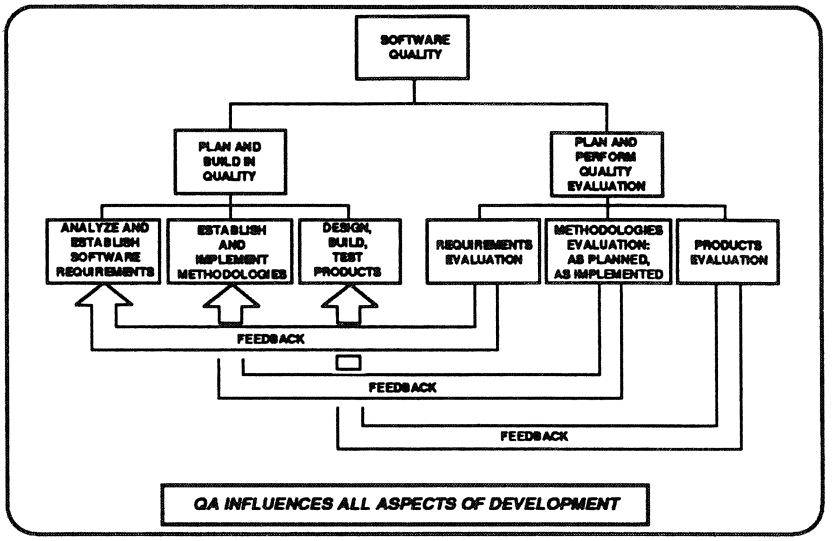


Figure 12. The QA Assessment
Approach Is Proactive

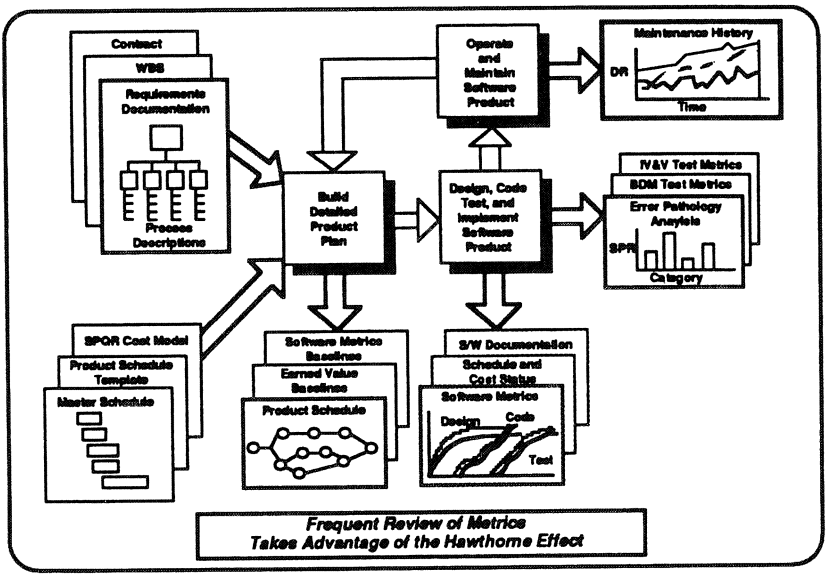


Figure 13. Program Control Is Based
On The Concept Of "Checks And Balances"

PERT/CPM activities and entering durations of activities to contribute to the detailed product plan. This plan consists of baselined software metrics controlled by the program control team and updated by data supplied weekly by the product team and baselined earned value charts controlled by the program control team and updated by a combination of product team progress reports and hourly costs data from integration with the corporate cost accounting system.

During product development, the product teams are responsible for all documentation deliverables to configuration management and schedule and performance status reporting to the program control group. A joint weekly review of all metrics focuses the attention of each product team on the total project. After system tests and IV&V tests, error pathology analyses supported by Fishbone Analysis identifies areas within the development process where significant errors are occurring, providing input for process improvement. Tracking and analysis of software errors found after the software is in production (implemented) further supports process improvement related to both the software testing and development process.

BDM's cost/schedule estimating approach was developed to support the firm fixed price bid for the final seven years of the RDB program development (Figure 14). Because of the history of cost estimating failures, considerable time was spent in developing a system based upon multiple checks for consistency and reason to minimize risk for estimating the over $150 million dollars remaining effort. A function point model, the use of empirically derived productivity factors, and estimates from experienced product managers and teams used in an interactive process and then mapped to the activities of PERT/CPM activities has provided BDM with the information required for accurate planning and reasonable costing. The results have been very promising, but experience has shown that the management intensity and discipline must be continuous.

BDM's SPEC™ is built upon commercially available tools (Figure 15). The integration of key components provides connectivity to the mainframe development environment and access to the configured and standardized Data Element Dictionary (DED). Customized Excelerator™ supports the system analysis, design and BDM's Software Blueprint® development and production through full integration with Ventura Desktop Publisher. BDM's approach to software documentation saved $5 million in two years.

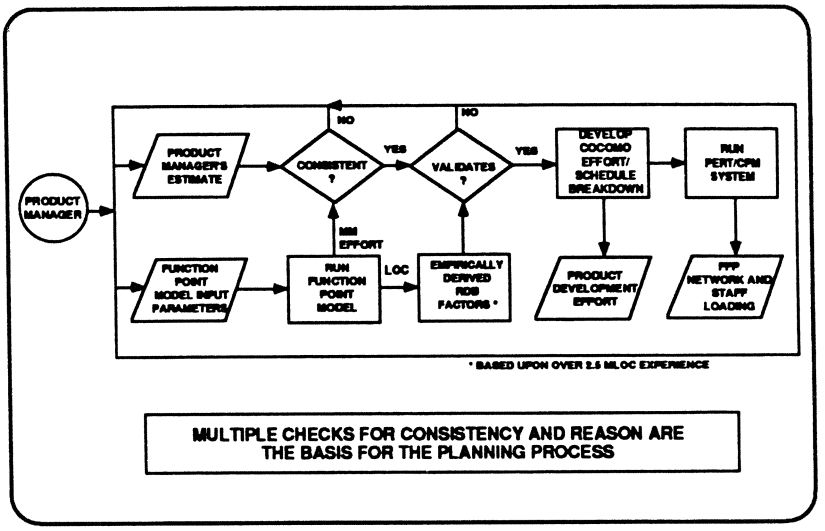


Figure 14. BDM Cost/Schedule Estimating Approach is Designed to Minimize Risk

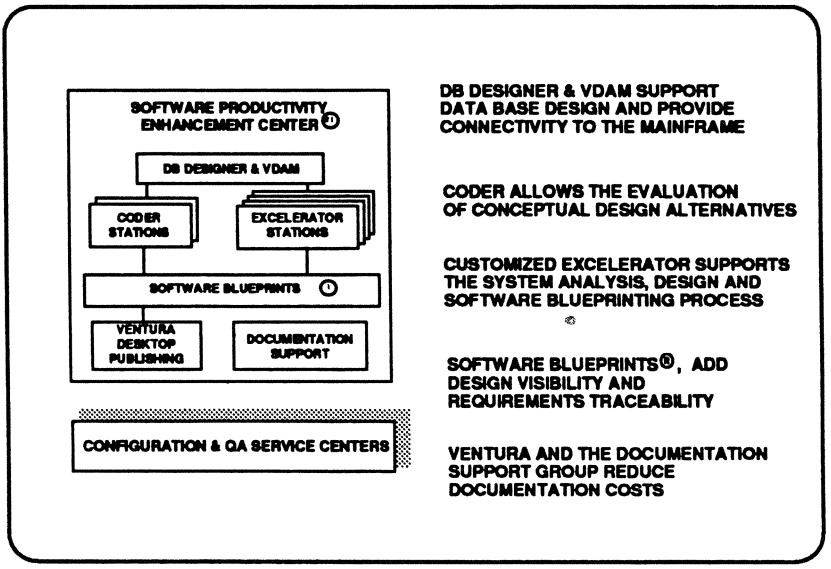


Figure 15. The Software Productivity Enhancement Center (SPEC)™ is Built Upon Commercially Available Tools

The BDM Software Blueprint® is the most important output of the SPEC™. It consists of a complete picture of the design (Figure 16). It maps from what is visible to the user to the underlying engineering detail required by the programmer. It also provides a rapid paper prototyping method for design walkthroughs by the product team and for walkthroughs with the client. Each screen or report segment is mapped completely to the lowest detail of the requirements documentation, ensuring the client user complete visibility of design to requirements. Another key component of the tactics is the development of high quality Unit Development Folders (UDFs), the building blocks of the detailed design. High quality UDFs mean low risk coding and maintenance code (Figure 17).

A major discipline of the tactics is Configuration Management (CM). CM is the key to project discipline and is used to maintain product baseline control. The CM staff is the channel through which all correspondence and documentation move between the product teams and all other components, including the Client (Figure 18).

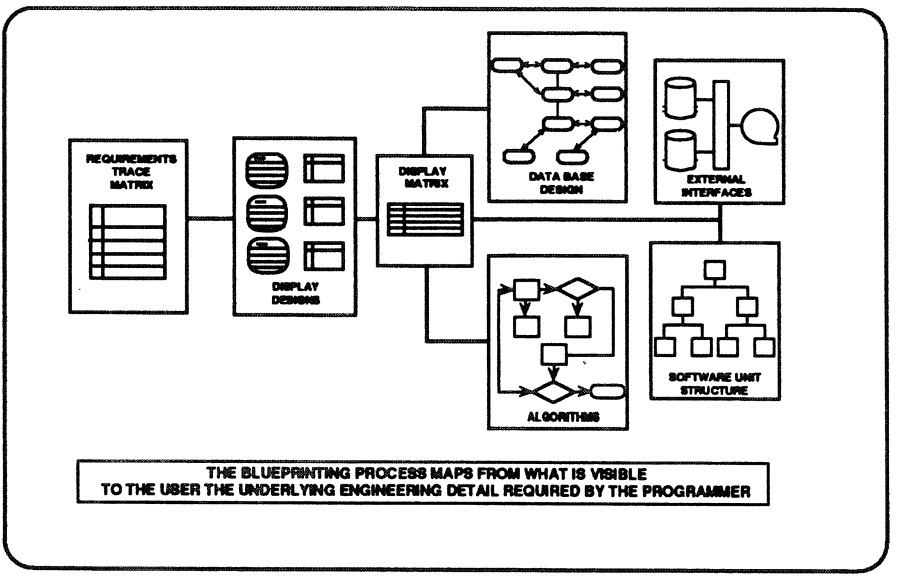


Figure 16. The Software Blueprint is the Most Important Output of the Software Productivity Enhancement Center

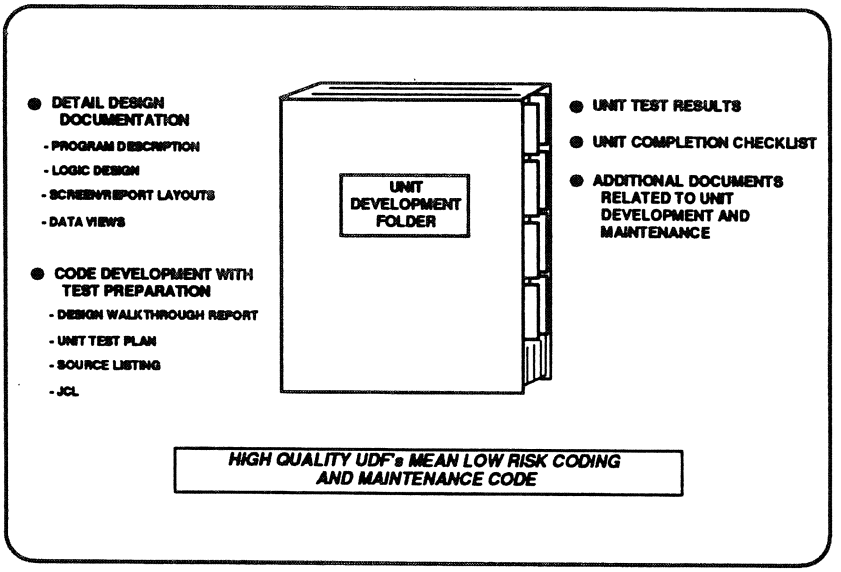


Figure 17. Unit Development Folders Are The Building Blocks Of The Detailed Design

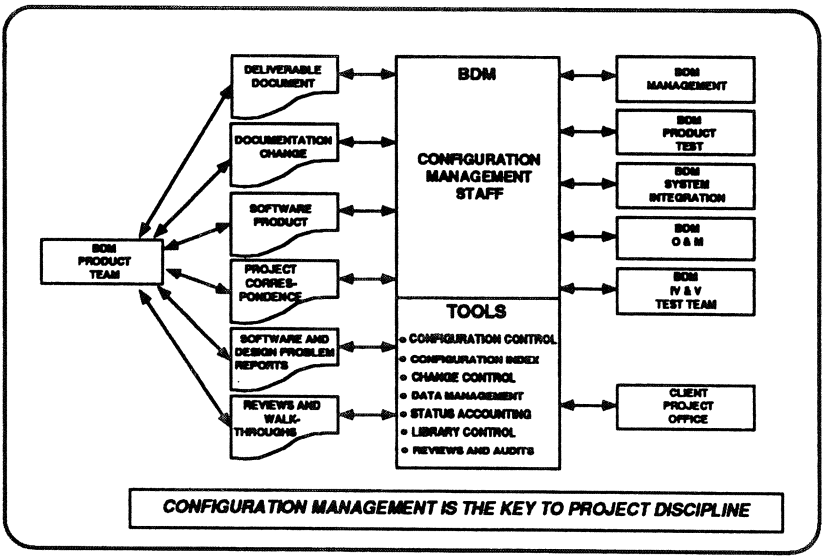


Figure 18. Product Baseline Control Is Maintained Through Configuration Management

V. RESULTS

The results from the BDM strategy and tactics to implement the strategy is very encouraging. The RDB program has received the Logistics Management Systems Center Commander's Award for Quality for 1991. Cost performance for the last four Option Years has been excellent (Figure 19). Validated error per thousand lines of code at user acceptance (results of the client's IV&V testing team) compare very favorably with industry norms (Figure 20). Measured maintenance actions for code that has been delivered to production and in use by the Client also compared very favorably with industry norms (Figure 21). Code quality is also supported by the use of only eight maintenance programmers with assistance from three data base staff to maintain over 2.5 million lines of code in production, with over 1,000 screens and reports. The current maintenance staff has maintained about the same backlog level and conducted system enhancements even in the face of increasing users and screens and reports in the field

	% OF TARGET	% OF ESTIMATED
OPTION YEAR 4		
TOTAL (INCLUDES SUSTAINING ENGINEERING)	111%	99.5%
OPTION YEAR 5		
TOTAL (INCLUDES SUSTAINING ENGINEERING)	99.1%	89.4%
OPTION YEAR 6		
TOTAL (INCLUDES SUSTAINING ENGINEERING)	97.6%	93.2%
OPTION YEAR 7*		
Estimated - Currently 85% Complete	100%	94%

Figure 19. Cost Performance Results
For The Last Four Option Years
Under Firm Fixed Price Environment

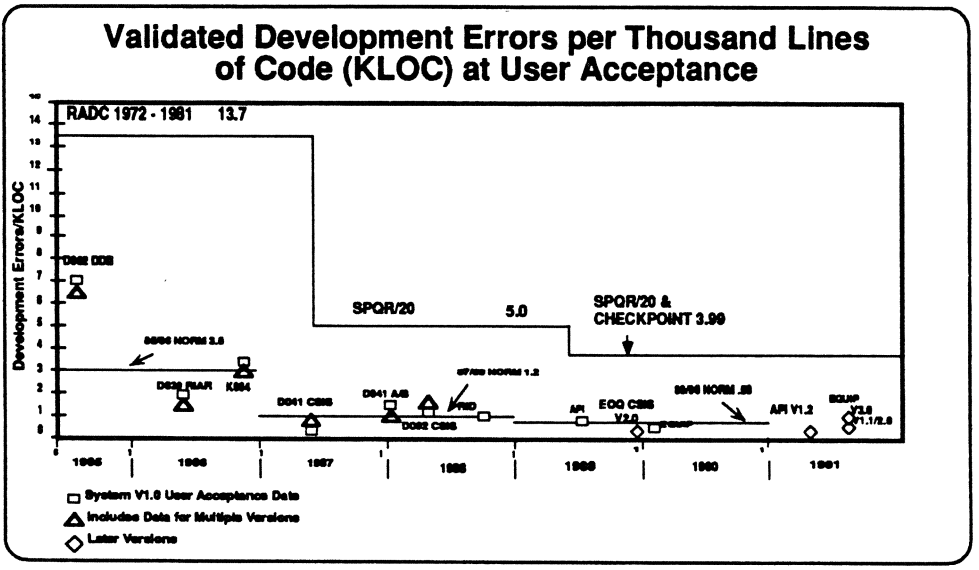


Figure 20. BDM Error Rates Compare Favorably With Industry Norms

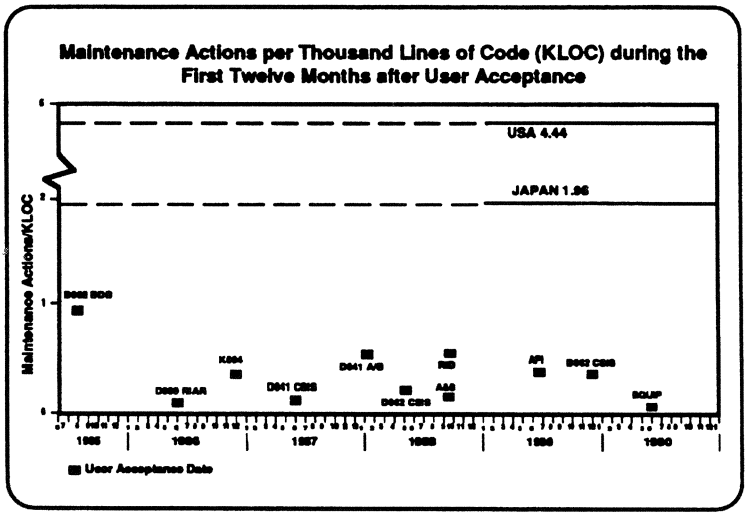


Figure 21. The In-service Reliability Of BDM Software Compares Favorably With Industry Norms

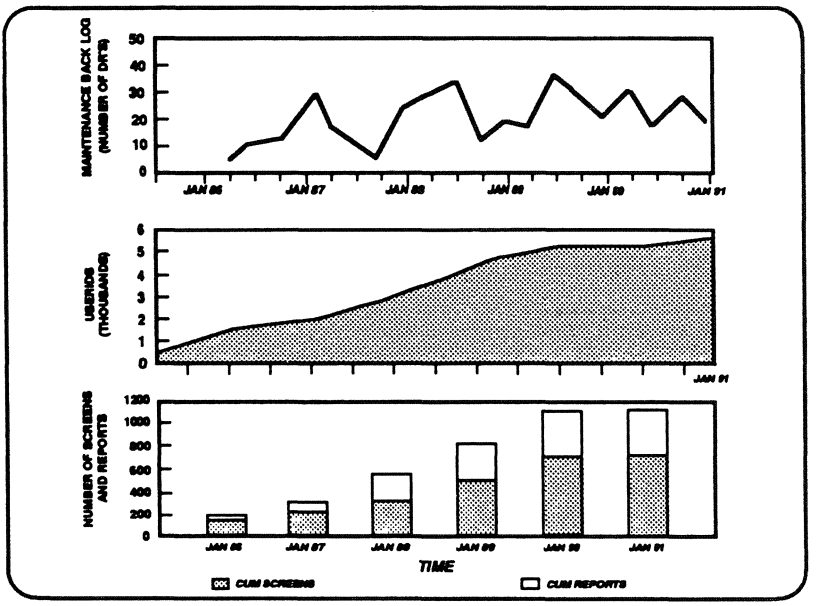


Figure 22. Maintenance Back Log Is Stable With Increasing Product Complexity And Light Maintenance Staff With Assistance From 3 Database Staff For 2.5 Million Lines Of Code In Production

(Figure 22). Figure 23 demonstrates the value of the Air Force RDB effort which is scheduled to be fully implemented in October 1994. Tremendous productivity gains have been derived from the use of the software components currently implemented.

VI. SUMMARY

Lessons learned and actions taken, as shown previously in Figure 3, as a result of BDM's strategy and implementing tactics have resulted in a successful solution to the early problems and challenges faced on this very large logistics management information system. The AF team's support and contributions to the total team effort have greatly enhanced the BDM actions.

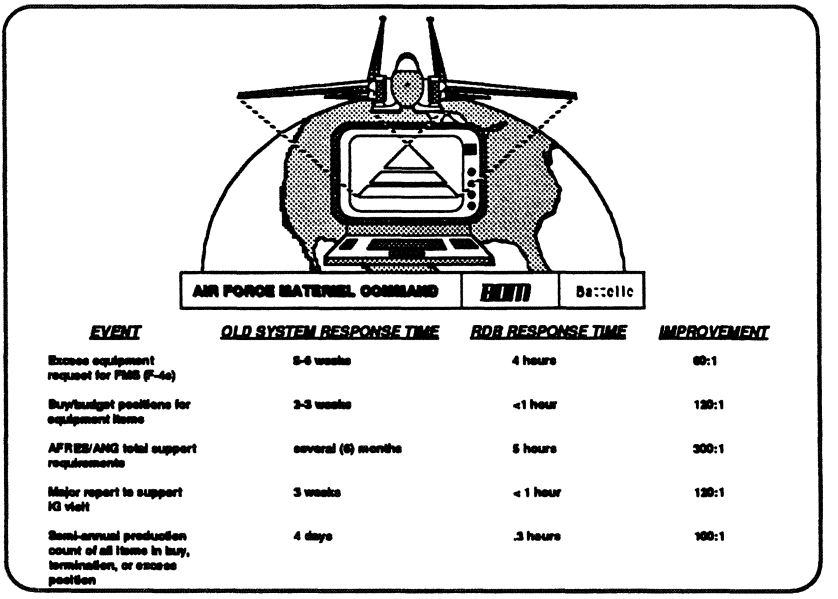


Figure 23. RDB Met Or Exceeded
User Expectations

VII. REFERENCES

[1] Marsh, Alton, "Pentagon Up Against a Software Wall," *Government Executive*, May, 1990, 62 - 63.

[2] Zells, Lois, "Project Management and CASE," Presentations of the National CASEC ON, New York City, New York, June 20 - 22, 1989.

[3] Lederer, Albert L. and Jayesh Prasad, "The Information Systems Development Cost Estimating Conundrum," A Presentation at the National OSRA/TIMS meeting, October, 1989.

The MERMAID Project

AJC Cowderoy, J O Jenkins and A Poulymenakou
School of Informatics, City University
London, EC1V 0HB
United Kingdom

1. INTRODUCTION

The tendency for software development projects to be completed over schedule and over budget has been documented extensively [1,2]. Additionally many projects are completed within budgetary and schedule target only as a result of the customer agreeing to accept reduced functionality.

A particular area of research of relevance to this phenomenon is software cost modelling. Many researchers have attempted to model the interrelationships of project cost parameters for instance Putnam [3]. These parameters are the Total Project Effort (in person time units). Elapsed time or schedule, T and the average staffing level M throughout the project. In his classic book, The Mythical Man Month, Fred Brooks [4] exposes the fallacy that effort and schedule are freely interchangeable. All current cost models are produced on the assumption that there is very limited scope for schedule compression unless there is a corresponding reduction in delivered functionality.

2. MERMAID

The Metrication and Resource Modelling Aid (MERMAID) project, partially financed by the Commission of the European Communities (CEC) as Project 2046 began in October 1988 and its goals are as folows:

- Improvement of understanding of the relationships between software development productivity and product and process metrics.

- To facilitate the widespread technology transfer from the Consortium to the European Software industry.

- To facilitate the widespread uptake of cost estimation techniques by the provision of prototype cost estimation tools.

The applicability of the tools developed by the MERMAID consortium is considered to encompass both embedded systems and Management Information Systems.

MERMAID has developed a family of methods for cost estimation, many of which have had tools implemented in the first two prototypes. These prototypes are best considered as toolkits or workbenches. Figure 1 gives an architectural overview of these prototypes.

The first prototype was demonstrated in november 1990. It was developed on a SUN 3/60 Workstation using an objective oriented extension of the C language, Objective C. Two versions exist, one running under the Portable Common Tool Environment (PCTE) and the other is a UNIX implementation. The second prototype was demonstrated in November 1991, versions of which were developed on an IBM PS/2 running either WINDOWS [3] or OS/2 and Presentation Manager. A first commercial version was demonstrated at the 14th International Conference on Software Engineering in May, 1992.

3. MERMAID ESTIMATION METHODS

At the time of the start of the MERMAID project, October 1988, the commonly used approaches to cost estimation were as follows:

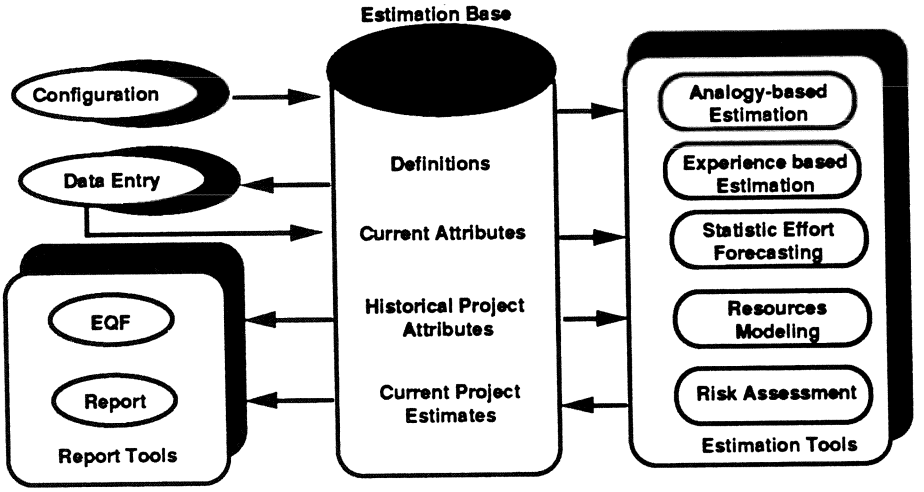


Figure 1. Mermaid Architecture

- Expert Judgement, ie informal guestimate of the resources required.

- Analogy, similar to the above but influenced by the identification of a similar completed project, similar to the one being planned.

- Parametric Model, use of a cost estimation tool based on a number of existing models of the relationships between project cost parameters and cost drivers. Models on which tools were based included SLIM [3], COCOMO [5], and COMPMO [6].

Tools based on these models can be calibrated for a particular environment. However, research has shown that despite calibration, the accuracy of estimates produced by cost estimation tools is poor [7]. There are several contributing factors to this inaccuracy. these include the difficulty, if not impossibility, of estimating the size of the product to be developed as early in the lifecycle as the Requirement Analysis Phase. Additionally calibration depends on the existence of moderate quantities of past project data collected in a consistent manner.

The MERMAID approach is based on the use of locally-based and user-defined metrics. Furthermore wherever possible actual measures as against estimates are used as input to the statistical estimating facility. This has the advantage of ensuring consistency and accuracy. Today's tools based on parametric models normally require the project manager or estimator to input an estimate of the size of the software product to be developed. This is either expressed in Lines of Code (LOC) or in the form of a function-based metrics, Function Point Count. This latter metric was developed within IBM by Alan Albrecht [8] and purports to measure the size of an application in terms of the functionality to be delivered to the user. It has the advantage over LOC in that it can be measured as soon as an outline logical design of the application is available. It is the view of the MERMAID project that wherever possible estimates must be based on measures not other estimates. This is coupled with the development lifecycle model assumed by MERMAID, ie a project is then regarded as a series of milestones separated by phases. This view enables the estimator to model any organizations lifecycle. Care must be taken not to confuse this use of the term phase with its use in a particular lifecycle model.

4. DEVELOPMENT PRODUCTIVITY ANALYSIS

Bailey and Basili [9] suggested that it should be possible to develop a satisfactory estimation model for a given environment using a small number of independent product and process attributes as the dependent variables in a multivariable linear regression model. Such an approach differs from that found in the COCOMO [5] model where the nominal estimate derived from the standard estimation model is subjected to a multiplicative adjustment factor. This measures the combined influence of a number of cost or productivity drivers. Recent research [10] has suggested that many these cost drivers are not independent.

An analysis by MERMAID [11] of a dataset from various commercial MIS environments using principal component analysis of 21 supposed productivity factors indicated that 7 principal components accounted for over 75% of the variability of the

data and that no other component accounted for more than 5% of the variability. The main source of the belief that staff and environment characteristics are significant factors in determining development productivity is the importance placed on them in most of the published cost models. A note of caution against reading too much into these analyses must be sounded. Any retrospective validation using data from completed projects is fraught with methodological difficulties, not the least being the uncertainty over the accuracy of the measurements. Classical experimental design involving the monitoring of on-going projects is rarely possible in this domain.

5. MERMAID AND ESPRIT

The Commission of the European Communities (CEC) launched ESPRIT in 1983 largely as a consequence of the realization in Europe that its Information Technology industry was becoming increasingly uncomparative particularly viz-a-viz Japan. ESPRIT projects cover both hardware and software technologies and address both factory and office automation. Both software engineering and artificial intelligence are primary focuses. MERMAID is one of a number of projects, some current and others which have been completed, which address aspects of software development, management and metrics. Two immediate predecessors of MERMAID in which some of ideas were developed were IMPW [12] and REQUEST [13]. The former developed an integrated project management toolkit which included support for cost estimation and latter was concerned with modelling aspects of software reliability.

ESPRIT has begun its third phase and today's projects have a clearer user requirement orientation than those in the first phase which were largely technology push projects. The change in emphasis followed early evaluations of the programma which showed a disappointing take-up of the research output by industry even in cases where industry itself was the prime mover behind the project.

6. THE IMMEDIATE FUTURE

The MERMAID consortium is putting the finishing touches to the specification of the tool functionality to be included in the Mark [2] prototype due for release late in 1992. Considerable attention is been paid to developing risk assessment capability which conforms to the MERMAID philosophy. Such a risk assessment capability will require access to a knowledge base of pervious projects and measures of risk to project budget and schedule will be estimated using similar statistical and analogical techniques to those used for effort estimation. In addition, the inclusion of a facility to examine effort and schedule trade-off is planned. This will be based on the Kunomaa Resource (KURE) Model [14] which takes a thermodynamic view of the software development process. An alternative to this involves the use of a System Dynamics Model [15] Superior sensitivity superior sensitivity analysis capability can be provided in this way provided an adequate understanding of the relationships between schedule effort and manpower level is established for the development environment. A further pair of estimation methods for use when there is a shortage of data describing past projects will be provided; these are the Analogy-Based Estimation Method, suitable when there is between 4 and 10 projects and the Experience-Based Estimation Method for fewer than 4 . The final prototype will be available in late 1992.

LIST OF REFERENCES

1. De Marco T (1982), Controlling Software Projects, Yourdon Press, New York

2. US Congress (1989), Bugs in the Program: Problems in Federal Government Computer Software Development and Regulation. Staff Study for the House of Representatives Committee on Science, Space and Technology.

3. Putnam L H (1987), A general empirical solution to the macro software sizing and estimation problem IEEE Trans. Software Engineering SE-4 (4).

4. Brooks F (1975), The Mythical Man Month, Addison-Wesley, London.

5. Boehm, B W (1981), Software Engineering Economics, Prentice Hall, Englefield. Cliffs, N J.

6. Conte S, Dunsmore H and Shen V Y, (1986), Software Engineering Metrics and Models, Benjamin-Cumimns, Menlo Park CA.

7. Kemerer C F (1987), An Empirical Validation of Software Cost Estimation Models, Comm, ACM 30(5).

8. Albrecht A J Measuring Application Development Productivity Proceedings Joint SHARE/GUIDE Symposium, October 1979, pp83-92.

9. Bailey J W and Basili V (1981) A Meta model for software development resource expenditure. Proceedings of the 5th International Conference on Software Engineering, pp107-116.

10. Subramian G H and Breslawski S (1989) A case for dimensionality reduction in software development of effort estimates. TR 89-02 Computer and Information Science Department, Temple University, Philadelphia PA.

11. Kitchenham B and Kirakowski J (1991), 2nd Analysis of Mermaid Data. Mermaid Project Deliverable D 3.3.B.

12. Bosco M, Jenkins J and Verbruggen R (1987), Integrated Management Process Workbench (IMPW) Advance Papers, 1st International Workshop on CASE, Cambridge, Mass.

13. Linkman S J (1990) Quantitative monitoring of software development by time based and intercheckpoint monitoring. Software Engineering Journal Vol. 5(1).

14. Harry, C M and Jenkins J (1991) Mermaid Working Paper TCU-5-0-D-500/1.

15. Bell G and Jenkins J (1991), Proceedings of the 7th International COCOMO Users Group Meeting, SEI, Pittsburgh.

SOFTWARE REUSE AND PRODUCTIVITY: AN EMPIRICAL VIEW

Thomas P. Frazier
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, Virginia 22311 USA

A. INTRODUCTION

Software reuse, by which we mean the reuse of existing software code or design, has gained a great deal of attention because it seems to offer a way to attain substantial increases in software development productivity. Under the reuse paradigm the software developer would select from libraries or repositories of software components and build his program "component-by-component" instead of via the traditional method of writing code "line-by-line." A component could be an existing standard mathematical or statistical routine, utilities related to operating systems, or even an entire program. Also, there are different levels or degrees of reuse. Some components can be reused verbatim while others must be modified before being incorporated into the new program.

Reuse would seem to enhance labor productivity, by which we mean the amount of output (usually measured in lines of code) per unit of labor effort (usually measured in hours or months) in both the development and maintenance phases of the software program life cycle. In the development phase, the practice of reuse would increase labor productivity because less new code needs to be written. Boehm and Papaccio [1] cite data from a firm that suggest for typical new business applications, reusable code accounts for as much as 60 percent of the total code count. In the maintenance phase reuse would increase software quality (measured in terms of defects) due to multiple testing and error removal opportunities. The smaller the number of errors, the lower the number of corrective maintenance actions incurred as a result of reuse [2].

There are costs to practicing reuse. The ability to capture the benefits of reuse depend to some degree on investment in the infrastructure needed to facilitate reuse. Kang and Levy [3] point out that investments are required to make organizational changes that promote reuse. Examples of these organizational changes are extensive training projects that emphasize programming conventions geared to reuse, and central support staffs that monitor development activities and make reusable components available. The authors

argue that developing potentially reusable software is technically more difficult and should cost more than software not developed for reuse. The additional costs are associated with incorporating flexibility to handle a variety of uses. The additional cost to develop reusable code have been estimated to range from 10 percent for software to be reused across an individual project to 50 percent for software to be reused across multiple projects [4].

There are also technical barriers to reuse that impose costs. The engineering problems of defining and establishing a reuse domain (a family of software projects having similar descriptions) [5], and building and populating reuse repositories [6] are but two of these technical barriers to reuse.

While the theory of reuse and its impact on labor productivity is rather straightforward, there is little empirical evidence to either support or refute the idea that reuse should result in higher labor productivity. There is a great deal of literature on the technical aspects of software reuse (e.g., see [7]) and on software development labor productivity. (A good survey is found in [1]; more recent examples are [8] and [9].) Unfortunately, there has been little empirical study of the direct impact of the practice of reuse on software development labor productivity.

A recent study of reuse savings used data collected by the National Aeronautics and Space Administration (NASA)/University of Maryland Software Engineering Laboratory (SEL) on twenty-five flight dynamics development software projects [10]. The study included a sample of 2,954 software modules (subroutines) written in FORTRAN. The modules were stratified according to their origin: complete reuse without revision, reuse with slight revisions (< 25% changes), reuse with major revision (≥ 25% changes), and complete new development. The average productivity per module (measured in source line of code per tenth of development effort hour) were computed for each of the classes of modules. The complete reuse modules averaged 23 lines of code per tenth of hour. The slight revision modules averaged 1.8 lines of code per tenth of hour. The major revision modules averaged 1.4 lines of code per tenth of hour. Complete new development modules averaged 1.1 lines of code per tenth of hour.

One limitation (and strength) of the SEL data is that the data are focused on a rather small and well-defined area, ground support software for unmanned spacecraft

control, for a single customer. This narrow focus makes extrapolation to other application areas difficult. However, it makes an excellent domain environment in which to observe reuse, and it ensures some consistency in data collection and analysis. Selby's work focuses on the module level versus the more aggregate project level. Would we observe the same results at the project level from other application areas, other languages, and with projects for a wide variety of customers?

This paper attempts to answer this question by empirically testing the hypothesis that projects that made use of existing software components would take less labor effort per line of delivered code than projects not built from existing code. In addition, this study seeks to extend Selby's work by modeling additional variables that influence software development productivity. Specifically, we test to see if verbatim reuse and modified reuse take less labor effort per line of delivered code than code developed from scratch.

The next section describes the model used to test the hypothesis. The third section presents the data used in the model. The data are observations on large management information systems applications taken from a sample of 90 Department of Defense (DoD) projects.

The fourth section presents the results. Verbatim reuse is found to have a significant impact on software development productivity. In our sample of projects, modified reuse does not have a significant impact on software development productivity. The fifth section discusses the cost savings implications of the research results.

B. MODEL

A model was constructed and its parameters estimated by regression analysis in order to test the hypothesis programs that made use of existing software components would take less labor effort per line of delivered source code than programs not built from existing code.

The model measures software development productivity as the total source lines of code produced (Q) divided by the total labor effort (L) expended to produce the code. Each program is generally a mix of new code (N), code reused from other sources after

being modified (*M*), and code reused without modification or verbatim (*V*).[1] We would expect verbatim reuse and modified reuse require less labor effort per line of delivered code than new code developed from scratch. In addition, we would expect verbatim reuse to require less labor effort per line of code than modified reuse. A well specified model of software development productivity would also include factors that account for the complexity of the software product, the software development tools employed in the development environment and the vintage of the software [1]. Therefore, the labor productivity of a given program depends on six arguments:

$$\frac{Q}{L} = F\,(R_N,\ R_M,\ R_V,\ C,\ T,\ A), \tag{1}$$

where R_N, R_M, and R_V represent the percentage of the total code that was new, reused with modification, and reused verbatim respectively and *C*, *T*, and *A* represent those complexity, tool environment, and age factors cited above.

We cannot estimate Equation (1) directly, since the percentages of the code that are verbatim, modified, and new are collinear with each other. However, we can make use of the fact that these percentages sum to one and rewrite Equation (1) in terms of just the percentage of total code that is modified and the percentage of total code that is verbatim. This is shown in Equation (2)

$$\frac{Q}{L} = F\,[(1 - R_M - R_V),\ R_M,\ R_V,\ C,\ T,\ A], \tag{2}$$

Ordinary least square (OLS) can be used to estimate Equation (2). The OLS model is shown in Equation (3)

$$\frac{Q}{L} = \alpha_0 + \alpha_1 R_M + \alpha_2 R_V + \alpha_3 C + \alpha_4 T + \alpha_5 A + \varepsilon, \tag{3}$$

where α_0, α_1, α_2, α_3, α_4, and α_5 are the parameters to be estimated and ε is the disturbance term that is assumed to be distributed independent normal, $N(\sigma^2)$,). All other symbols retain their previous meanings.

---

1 Selby's complete reuse without revision is equivalent to our verbatim reuse. Complete new development is our new development category. Reuse with slight revision (<25% changes) and reuse with major revision (≥25% changes) are equivalent to our modified reuse category since we make no distinction as to the degree of reuse.

The productivity associated with the modified code is interpreted as $\alpha_0 + \alpha_1$, the productivity associated with the verbatim code is interpreted as $\alpha_0 + \alpha_2$, and the productivity associated with the new code is interpreted as α_0.

If the hypothesis is correct that verbatim reuse and modified reuse take less labor effort per line of delivered code than new code developed from scratch, we would expect the productivity associated with the verbatim reused code and the productivity of the modified reused code to be significantly greater than the productivity associated with the new code. In addition we would expect the verbatim reuse productivity to be greater than the modified reuse. The hypotheses are summarized in Table 1.

Table 1. Productivity Hypotheses

Hypothesis	Regression Coefficient
Modified reuse productivity is greater than new code productivity	$\alpha_1 > \alpha_0$
Verbatim reuse productivity is greater than new code productivity	$\alpha_2 > \alpha_0$
Verbatim reuse productivity is greater than modified code productivity	$\alpha_2 > \alpha_1$

Because the OLS model includes the explanatory variables *C*, *T*, *A*, in addition to the reuse variables, the α_0 term picks up influences from those additional explanatory variables. To account for these effects, the means of the additional variables [i.e., $(\alpha_3\overline{C} + \alpha_4\overline{T} + \alpha_5\overline{A})$] must be subtracted out of α_0 before the productivities can be computed.

C. THE DATA

The data used to estimate the model were taken from a stratified survey sample of 90 Department of Defense (DoD) automated information system (AIS) programs. These AIS programs perform general-purpose automatic data processing functions. They were developed by both in-house and contractor personnel, and they cover a time period from the early 1970s to the late 1980s [11].

Because there is no completely documented inventory of DoD AISs, the sampling strategy involved collecting data on software development projects that are consistent in age, function, and DoD component within the known DoD hardware

inventory. The DoD computer hardware inventory is contained in the Automated Resources Management System maintained by the Defense Logistics Agency [12].

The survey contained information that could be used to construct the variables represented in Equation (3). Specifically, the survey data included the number of lines of source code, the number of lines of code reused, the effort required to develop these lines of code, the software engineering tools used in the development environment, the age of code, and the complexity of the code. Each of the variables are discussed below.

Respondents to the survey were asked three questions about the number of delivered source lines of code (Q). The respondents were asked to estimate the total number of source lines of code (all code counts were measured in thousands of delivered source lines) for the program, the number of source lines of code reused from other sources after being modified, and the number of source lines of code reused without modification. New lines of code were defined as the total number of source lines of code minus the sum of the number of source lines of code reused from other sources after being modified and the number of source lines of code reused verbatim.

Because of the difficulty in comparing lines of code across computer languages, only those COBOL projects were included in the sample for this study. The respondents were instructed to follow the code counting convention used in [13]. Lines of code are defined to include all program instructions created by program personnel and processed into machine code by some combination of preprocessors, compilers, and assemblers. It excludes comment cards and unmodified utility software, but it includes job control language, format statements, and data declarations. Instructions are defined as lines of code or card images. It excludes non-delivered support software such as test drivers. However, if the support software were developed with the same care as delivered software, with their own reviews, test plans, documentation, etc., then they were counted.

The labor variable measures the person-months of effort required to complete the main software build. This is defined to include effort for detailed design, coding, integration, quality assurance, configuration management, publications, and management.

In order to account for exogenous trends in software development productivity growth [11] over time, a trend variable was constructed. The variable measures the age of

the software since it was certified as fully operational. The variable is measured in terms of calendar months.

Each respondent in the survey indicated the availability and usage of the software productivity tools and procedures (as described in Table 27-7 of Reference [13]), plus several additional tools that were developed and employed subsequently. In order to ensure definition consistency across all respondents, a glossary of terms concerning each of the tools was provided with the survey instrument.

Boehm [13] developed a five-level classification scheme: very low, low, nominal, high, and very high. We used a modification of Boehm's classification scheme. Those tools that were described as being low and very low were aggregated into a single category. The same grouping strategy was executed for those tools that Boehm labeled high and very high. Hence, we have three tool categories: low, nominal, and high. The tools and the groupings are summarized in Table 2.[2]

Table 2. Software Tools Rating Scale

Ratings	Tools
Low	Assembler, Chief Programmer Team, Configuration Management, Database Aids, Batch Debuggers, Programming Support, Time-sharing Operating System, Performance Measurement and Analysis Tools
Nominal	HOL Compiler, Project and Data Entry Control Systems, Data Dictionaries, Interactive Debuggers, Source Code Formatters, Report Generators, Screen Generators, Reusable Source and Object Code Library System, Virtual Memory Operating System, Macro Assembler, Text Editor and Manager
High	Cross Compiler, Conversion Aids, Database Design Aids, Data Base Management System (DBMS), Distributed Processing, Active Documentation Tools, Failure Analyses Tools, Formal Verification, Display Formatters, Code Generators, Application Generators, Integrated Computer-Assisted Software Engineering Environments, Local Area Network, Program Design Language and Tools, Requirements Specification Language and Analyzer, Interactive Source Editor, Automated Verification System, Expert Systems Applications to Software Engineering, Instruction Set Simulators, Regression Testing, Restructuring Tools and Test Coverage Analyzers

2 This listing contains all the tools presented in Bohem's original work [13] plus additional tools that were developed later.

In order to account for the impact that these tools might have on software productivity, a dummy variable was constructed. If the program employed at least one of the tools the "high" category the dummy variable was coded as a 1; otherwise it was coded as a 0. This variable is represented as T in regression Equation (3).

The complexity of the software program was measured by the respondents' answers to a question concerning the algorithms and logic design of the software. The question asked the respondents to check one of six descriptions of the software. The descriptions were presented in descending order of complexity. If the respondents checked the description (deemed the most complicated) "Algorithms and logic design were created from scratch. Many complicated hardware/software interfaces had to be defined as design matured," a dummy variable was coded as 1. If the respondents checked any of the other five descriptions, the dummy variable was coded as 0. This variable became the measure of the complexity of the software code.

Forty-nine of the original 90 projects in the survey met the requirement of being written in COBOL and contained data on all the variables detailed above. Twenty-nine of the projects employed no reused code. Twenty projects reported reusing (either modifying or verbatim) existing code. Table 3 presents the size and proportion of new, modified reuse, and verbatim reuse of code for the 49 projects. The means and standard deviations of the projects in the sample are also presented.

The average size of the 49 projects is 318 thousand source lines of code (KSLOC). The average size of the 29 projects not employing reuse is 239 KSLOC. The average size of the 20 projects employing reuse is 431 KSLOC. However, project size was found not to be significantly related to the incidence of reuse.

The sample means and standard deviations for all the variables used in the model are presented in Table 4.

D. RESULTS

OLS was used to estimate Equation (3). The results are presented in Table 5. All the variables carry the expected signs. The modified code variable is clearly not statistically significant. A one-percent increase in the fraction of code that is reused

Table 3. Percentage of Code by Origin by Project

Project Identification	Size (KSLOC)	Percentage of Code		
		New	Modified	Verbatim
A001	130	1.00	0.00	0.00
A002	400	0.40	0.50	0.10
A004	1	0.00	0.50	0.50
A007	58	0.00	0.62	0.38
A008	68	1.00	0.00	0.00
A010	12	0.96	0.00	0.00
A011	1,200	1.00	0.04	0.00
A012	500	1.00	0.00	0.00
A014	20	1.00	0.00	0.00
A018	45	0.33	0.00	0.00
A020	3	1.00	0.67	0.00
A021	75	1.00	0.00	0.00
A027	500	1.00	0.00	0.00
A027	500	1.00	0.00	0.00
A028	250	1.00	0.00	0.00
D001	147	1.00	0.00	0.00
D002	1,098	1.00	0.00	0.00
D004	1,500	1.00	0.00	0.00
F002	22	1.00	0.00	0.00
F004	60	1.00	0.00	0.00
F005	50	1.00	0.00	0.00
F006	1,784	0.44	0.56	0.00
F007	100	0.00	0.10	0.90
F008	115	1.00	0.00	0.00
F011	102	1.00	0.00	0.00
F012	10	1.00	0.00	0.00
F014	110	1.00	0.00	0.00
F015	80	0.38	0.31	0.31
F018	212	1.00	0.00	0.00
F021	50	0.70	0.30	0.00
F023	41	1.00	0.00	0.00
F025	45	0.00	0.24	0.76
F026	38	1.00	0.00	0.00
F027	200	1.00	0.00	0.00
F029	5	1.00	0.00	0.00
F030	90	0.76	0.19	0.06
F032	750	0.00	0.33	0.67
F033	50	1.00	0.00	0.00
F034	387	0.18	0.19	0.63
F035	20	0.25	0.45	0.30
F036	4	1.00	0.00	0.00
N004	900	0.17	0.08	0.75
N008	250	1.00	0.00	0.00
N009	15	0.47	0.00	0.53
N010	750	1.00	0.00	0.00
N011	700	0.56	0.00	0.44
N012	25	0.84	0.16	0.00
N013	2,000	0.98	0.01	0.00
N015	108	1.00	0.00	0.00
N016	500	1.00	0.00	0.00
Mean	318	0.76	0.11	0.13
Std. Deviation	478	0.36	0.19	0.25

verbatim increases developer productivity by 1,075 source lines of code per month. An additional calendar month of age decreases developer productivity by four source lines of code per month. If the project employed at least one of the tools the "high" category, developer productivity increases by 539 sources lines of code per month. More complicated code (i.e., algorithms and logic design were created from scratch and complicated hardware/software interfaces had to be defined as design matured) decreases developer productivity by 455 sources lines of code per month.

Table 4. Sample Characteristics: Means and Standard Deviations

Variable Name	Symbol	Units	Mean	Standard Deviation
Productivity	$\frac{Q}{L}$	Thousands of source lines of code per person-mont	.991	.977
Modified reused code	R_M	Fraction of total project code	.107	.191
Verbatim reused code	R_V	Fraction of total project code	.129	.251
Age	A	Calendar months	89	69
Tools	T	1 or 0	.877	.331
Complexity	C	1 or 0	.367	.487

Table 5. OLS Estimates of Software Development Productivity

Variable	Coefficient	Standard Error
Intercept	0.923	0.426
Modified Code	0.038	0.776
Verbatim Code	1.075	0.539
Age	-0.004	0.001
Tools	0.539	0.378
Complexity	-0.455	0.267
n = 49	R^2 = .27	

Adjusting the intercept term as described and using the regression results, we can compute the new, modified, and verbatim code productivities can be computed. The

computed productivity and the productivities normalized to new code development are shown in Table 6.

Table 6. Estimates of Software Development Productivity by Code Origin

Origin of Code	Productivity (KSLOC per person-month)	Normalized
New Code	0.98	1.00
Modified Code	1.02	1.04
Verbatim Code	2.08	2.12

Modified reuse is estimated to generate productivity improvements of about 4 percent over new development. However, the regression coefficient of the modified reuse variable was not statistically significant. We concluded, at least for the sampled projects in this study, that there were no significant productivity improvements from partial reuse.

This result contrasts with findings reported by Selby et al. in [10] that productivity improvements of 45 percent for "slight revision" and 30 percent for "major revisions". It should be noted that Selby's unit of observation was the software module, whereas in this study the unit of observation is the entire software project. One would expect the project data to include the effort needed to access and integrate the reused modules into the project. This integration effort is probably not captured in the module data.

Verbatim reuse, on the other hand, is estimated to produce productivity improvements 2.12 times greater than code developed from scratch. This finding is consistent with experience that some effort is needed to integrate even completely reused software into the project [14]. While our sample indicates that verbatim reuse generates relatively large increases in development productivity, the improvements are not as large as the factor of 24 improvements that "complete reuse" had over new development reported by Selby [10]. The reasons for this difference are most likely the same as those offered for the modified reuse results.

The regression results also confirm that verbatim reuse produces larger increases in productivity than modified reuse.

Weighting the mix of new, modified, and verbatim code of projects in the sample by their respective computed productivity allows a comparison of the effort that was expended to develop the code versus the effort that would have been expended to develop the same code from scratch. The average productivity improvement from all sources of reuse when applied to the sample is 22 percent.

E. SUMMARY

This paper empirically tested the hypothesis that software development projects that made use of existing software components would take less labor effort per line of delivered code than projects not built from existing code. Data taken from a survey of 49 DoD projects were used in an OLS model to test the hypothesis. All of the projects were written in COBOL and all performed general-purpose automatic data processing functions.

The results indicate verbatim reuse produced productivity improvements 2.12 times greater than code developed from scratch. Modified reuse was estimated to improve productivity by 4 percent. However, there was no statistically significant difference in the impact of modified code and new code on development productivity. Weighting the sample observations by the computed productivities allowed us to compare the effort that was expended to develop the code versus the effort that would have been expended to develop the same code from scratch. The average productivity improvement from all sources of reuse when applied to the sample was computed to be 22 percent.

This paper also presented results of additional explanatory variables that characterized the development environment and the age and complexity of the code. All of these explanatory variables carried the expected signs and were statistically significant.

REFERENCES

[1] Boehm, B. W., and P.N. Papaccio. "Understanding and Controlling Software Costs," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, October 1988.

[2] Cruickshank, R. D., and J. E. Gaffney. "An Economics Model of Software Reuse." Proceedings, Conference on Analytical Methods in Software Engineering Economics, McLean, VA, April 1991.

[3] Kang, K. C., and L. S. Levy. "Software Methodology in the Harsh Light of Economics." *Information and Software Technology*, vol. 31, no. 5, June 1989.

[4] Boehm, B. W., and W. E. Royce. "Ada, COCOMO and the Ada Process Model." Proceedings, Fifth International COCOMO User's Group Meeting, SEI, Pittsburgh, PA, October 1989.

[5] Parnas, D. "Design of Program Families," *IEEE Transactions on Software Engineering*, vol. 2, no. 1, June 1976.

[6] Garnett, E. S., and J. A. Mariani. "Software Reclamation," *Software Engineering Journal*, vol. 5, no. 3, May 1990.

[7] Tracz, W. (ed.). *Software Reuse Emerging Technology*. IEEE Computer Society Press, 1988.

[8] Humphrey, W. S., and N. D. Singpurwalla. "Predicting (Individual) Software Productivity," *IEEE Transactions on Software Engineering*, vol. 17, no. 2, February 1991.

[9] Banker, R. D., and C. F. Kemerer. "Scale Economies in New Software Development," *IEEE Transactions on Software Engineering*, vol. 15, no. 10, October 1989.

[10] Selby, R. W. "Empirically Analyzing Software Reuse in a Production Environment" in Tracz, W. (ed), *Software Reuse Emerging Technology*, IEEE Computer Society Press, 1988.

[11] Levitan, K. B., J. Salasin, T. P. Frazier, and B. N. Angier. "Final Report on the Status of Software Obsolescence in the DoD." Paper P-2136, Institute for Defense Analyses, August 1988.

[12] Defense Logistics Agency. "DoD Automated Resources Management System (ARMS) Users Guide." April 1985.

[13] Boehm, B. W., *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[14] Boehm, B. W., and W. L. Scherlis. "Megaprogramming." Mimeograph, undated.

A SOFTWARE COST MODEL OF REUSE WITHIN A SINGLE SYSTEM

R. D. Cruickshank and J. E. Gaffney, Jr.
Software Productivity Consortium
2214 Rock Hill Road
Herndon, Virginia 22070

Overview

This paper presents a software economics model that can be used to demonstrate the cost effect of the the multiple use (i.e., reuse) of software components within a single software system. The basic idea is that one unit of code may be employed in more than one functional unit of a given software system, thereby reducing the overall development cost from what it would have been if each copy of that unit of code had had to be developed separately in each instance in which it was employed. Other economics effects that may occur, such as enhanced quality and reduced schedule, are not covered here. This model is based on earlier work (Gaffney and Durek 1991; Gaffney 1989; Cruickshank and Gaffney 1991) on the economics of software reuse. The earlier focus was on the economics benefits of reuse over multiple software systems. The new model presents the cost benefits that can result from the multiple use of one software component developed for a given software system within that software system.

Reuse economics models can be used to demonstrate the benefits that can be derived from software. These models aid the financial analyst, manager, or software engineer to better understand reuse in terms of the potential impact of decisions such as how much software to reuse. The paper presents a model that can be used to determine the economics impact of reuse within a single software system or product. An example application of this model is provided that uses data from an actual project.

Modes of Software Reuse

There are two modes of software reuse; *within* a single software system or *across* several related software systems. The former mode may be called *intra-project* reuse; the latter may be called *inter-project* reuse.The first mode includes the case in which a software project produces one product or computer software configuration item (CSCI) (Department of Defense 1988a). The first mode also includes the in which software components are reused over a set of functionally related but separately specified and implemented products (CSCIs) which are parts of a single large software system. The second mode, reuse over several related software products, includes the case in which software components are reused over a succession of related software products. In this situation, a series of software projects over time produces a succession of versions of a software

product, a version being a major revision and update. The second mode also covers the case in which several new systems are developed concurrently which use some software components in common. It is important to note that both modes can occur in one system. That is, there can be reuse within a given system as well across that system and others.

Software *reuse* can be viewed as *multiple use;* that term is used in this paper. Software can be *reused* or *multiply used* in a number of ways. For example, a single software development project could use a specific software function or module in several places in the product being produced. This is an example of multiple use within a single system. Several software functions could be multiply reused in the software product being developed. Alternatively, software reuse can occur across a family (Parnas 1976) of software systems. In this mode of multiple use, a function or set of functions could be multiply used across successive versions of a software system. A family (Parnas 1976) is a set of software systems having similar descriptions. These systems have similar requirements that can be (or are) satisfied by a common architecture and represent a set of closely related design choices at the detailed level.

To generalize, software can be used singly or multiply within a family of related software systems. As shown by Parnas, large scale reuse can be sequential (from one software system to another as with a succession of versions of a software product) or parallel (a number of software products developed simultaneously as part of a single software system).

The term 'reusable' means simply that a software unit or component can be employed in more than one instance. These components may be used once or many times in a given software system, or they may be used many times in a family of software systems. In the latter case, a common set of components may be used by several systems which could be developed in parallel or sequentially.

Reuse Within a Single System

Reuse within a single software system is the multiple use of one or more new units or components developed for that system. For example, an analysis of the requirements for a new system might indicate that each of several major functions in that system includes a very similar or identical capability. One software unit is then developed which can be employed in each of those major functions. Figure 1 represents this concept of the multiple use of n new software components in N major functions of a single system. A given software system has two categories of code, *new* and *reused* (IEEE 1992). There are two categories of *new code*, multiply used, as just described, and singly used. The *reused* code is that which is employed across several systems. The relationships of the

costs of new and reused code is covered in the reuse cost model equation presented later in this paper.

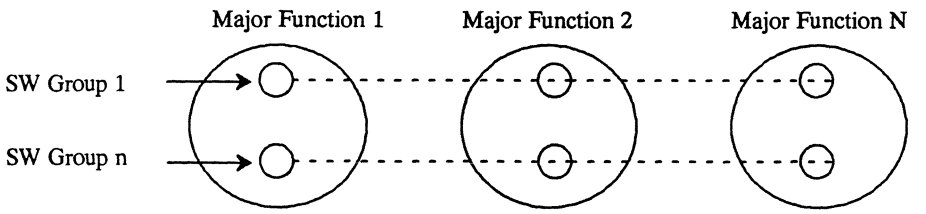


Figure 1. Reuse Within a Single System

Activity-Based Cost Models

Activity-based cost models (Cruickshank and Lesser 1982; Gaffney 1983; Cruickshank and Gaffney 1991) represent the cost of a software development project in terms of the costs of the activities which comprise the development cycle for that project. Typical development activities are: requirements analysis, design, implementation, and testing. The cost model presented here for new code which includes reuse within a project is an activity-based cost model.The cost of developing the new code for a software system, C_N, is equal to the product of the total unit cost (in labor months per thousand source statements, LM/KSLOC) and the amount of new code (in KSLOC) developed. The total unit cost is equal to the sum of the unit costs for the activities that are employed in the creation of the new code. The total cost of all n activities in the creation of the new code in a software system can be represented by the equation:

$$C_N = \sum_{i=1}^{n} (LM/KSLOC)_{i,new} \cdot KSLOC_{new}$$

A similar equation can written for the cost of reusing code, C_R, for reuse across systems.

The set of activities actually used on a given project may be selected developed from a 'menu' of process activities. The values used in estimating the unit costs of the activities may be based on the cost data from .previous experience maintained by the development organization. The software development activities used in this paper correspond to the activities in DoD Standard 2167A (Department of Defense 1988b). These activities are listed in Table 1.

Table 1. Symbols for Unit Costs of Development Activities

Activity	Symbol of Unit Cost
Requirements Analysis	C_{RA}
Preliminary Design	C_{PD}
Detailed Design	C_{DD}
Code and Unit Test	C_{CUT}
CSC Integration Test	C_{CIT}
CSCI Test	C_{CT}

Reuse Cost Models

Each mode of reuse (intra-system and inter-system) used on a project should pay for itself. This requirement should be reflected in the economics model used to evaluate the potential for each type of reuse. Payoff analyses can be done separately for each mode of reuse, but the payoff of one mode may be affected by the presence of the other mode of reuse. Reuse over a succession of versions (mode two) should amortize the capital investment made (including that for domain analysis) to provide for reuse across the systems. The total cost of multiple use of a particular software component within a single system (mode one), including the cost of domain analysis, must not exceed the cost of uniquely developing each of the replications.

The cost of developing the new code in a software system in which there is intra-system reuse, reuse within a single system, (mode one) is given by:

$$C_N = C_{SU} + C_U + C_C$$

- C_N = Cost of developing the new code for a system (in labor months, LM, or labor hours, LH).
- C_{SU} = Cost of development activities for single use (unique) code.
- C_U = Cost of unique development activities for multiply used code created for this system.
- C_C = Cost of common development activities for multiply used code created for this system.

The cost model for reuse over several software systems or over several versions of a software system (mode two) is given by:

$$C_S = \frac{C_D}{N} + C_N + C_R$$

- C_D = Cost of domain engineering which includes the creation of code that is reused across N systems. C_D is prorated across N systems (LM).
- C_N = Cost of the new code developed for an application in a family of systems (LM).
- C_R = Cost of reusing code developed for reuse across N systems (LM).

This model can include mode 1 reuse, multiple use of new code, by substituting the equation for C_N given above (for mode 1 reuse) for C_N in the equation for C_S.

Definition of the Cost Model of Reuse Within a Single System

Let:

- S_N = the total size of the new code in the delivered system (in KSLOC).
- S_U = the total size of the non-replicated new code (in KSLOC) in the software product uniquely developed for the software system.
- S_{Si} = the amount of new code (in KSLOC) in component i developed for multiple use in the system. There are n such components. The i^{th} component is replicated N_i times.
- S_{RD} = the total amount of new code developed for multiple use in the system. This is the sum of the sizes of the n new code components developed for multiple use in the software system.

$$S_{RD} = \sum_{i=1}^{n} S_{Si}$$

- S_{RE} = the total amount (all replications) of multiply used new code (KSLOC), where:

$$S_{RE} = \sum_{i=1}^{n} N_i \cdot S_{Si}$$

Thus, the total size of a the new code in a software product, S_N KSLOC, which includes the multiple use of n software components, is given by:

$$S_N = S_U + S_{RE}$$

Thus intra-system multiple use enables S_{RE}–S_{RD} KSLOC less code to be developed. This reduction in the amount of new code that has to be developed is the source of any potential savings in the cost of new code development in a software system.

Model Equations

The principal cost spreading assumptions of the model for multiply-used or shared code are:

1. The requirements analysis, preliminary design, detailed design, and code/unit test activities are done once for each unit to be replicated. The costs of these activities for a unit are amortized over all of the replications of that unit.

2. The unit costs of requirements analysis, preliminary design, detailed design, and code/unit test are 'm' times those for a unit of code which is intended for single use.Some experience The parameter 'm' is the ratio of the total cost of implementing a software unit suitable for multiple use to the cost of implementing a software unit for single use. The parameter m is assumed to be a number greater than or equal to 1.0 since the costs of implementing a unit suitable for multiple use are assumed to include the extra costs of domain analysis and reengineering in addition to the costs of development for single use. The model assumes the same value of m for each activity. Some experience suggests that 'm' may be on the order of 1.5 to 2.0.

3. The cost of system testing is allocated to all of the copies of the multiply used code. This cost is composed of the activities of CSC integration test and CSCI test as shown in Table 1.

4. Intra-system multiple use consists of making N copies of each multiply used component and distributing then through the system as required. Note that the cost model of reuse within a single system also covers the case in which just one copy of each desired components is made and those single copies are shared by the rest of the system.

Let:

- C_{NB} = the baseline cost of the new code in a system. This is the value of C_N for the system if none of the new code were done for multiple use ($S_{RE}=0$).
- C_{NR} = the cost of the of the new code in a system in which there is multiple use of some of the new code ($S_{RE}>0$).

Note that $C_{NR} < C_{NB}$ for reuse or multiple use within a system to pay off.

The cost of new code for a software product without multiple use, C_{NB}, is given by:

$$C_{NB} = (C_{RA} + C_{PD} + C_{DD} + C_{CUT} + C_{CIT} + C_{CT}) \cdot S_N$$

The cost of new code for a software system with multiple use of code, C_{NR}, is the sum of three elements including the cost of developing the unique code or:

$$C_N = C_{SU} + C_U + C_C$$

1. The cost of developing the unique (single use) code is $C_{SU} = C_{VNB} \cdot S_U$, or:

$$C_{SU} = (C_{RA} + C_{PD} + C_{DD} + C_{CUT} + C_{CIT} + C_{CT}) \cdot S_U$$

2. The cost of creating one copy of each of the N groups of multiply used code is $C_U = m \cdot C_{VNB} \cdot S_{RD}$, or:

$$C_U = m \cdot (C_{RA} + C_{PD} + C_{DD} + C_{CUT}) \cdot S_{RD}$$

The parameter m is the ratio of the cost of developing a unit suitable for multiple use within this software system relative to the cost of developing a unit not especially designed for multiple use.

3. The cost of system testing all copies of the multiply used code is $C_C = C_{VNBT} \cdot S_{RE}$, or:

$$C_C = (C_{CIT} + C_{CT}) \cdot S_{RE}$$

That is, C_C is the cost of doing CSC integration test and CSCI test.

Thus, the overall cost of the new code in an application system in which there is both unique and multiply used code is given by the expression:

$$C_N = C_{VN} .. S_N = C_{VNB} \cdot S_U + m \cdot C_{VNBD} \cdot S_{RD} + C_{VNBT} \cdot S_{RE}$$

Observe that $C_{VNB} = C_{VND} + C_{VNT}$. That is, the baseline new code development unit cost is equal to the sum of the development and the testing unit costs.

The savings, T_S, achieved by reuse or multiple use in the creation of a program is:

$$T_S = C_N - C_{NR}$$

Intra-system reuse pays off if $T_S > 0$. Since $T_S = C_{NB} - C_{NR}$ and $S_N = S_U + S_{RE}$:

$$T_S = (C_{RA} + C_{PD} + C_{DD} + C_{CUT}) \cdot S_{RE} - m \cdot (C_{RA} + C_{PD} + C_{DD} + C_{CUT}) \cdot S_{RD}$$

Thus intra-system reuse pays off if:

$$T_S = S_{RE} - m \cdot S_{RD} > 0$$

Intra-System Reuse Break-Even Cost Threshold

Intra-system reuse (multiple use) pays off whenever:

$$\frac{S_{RE}}{S_{RD}} > m$$

The intra-system reuse (multiple use) break-even cost threshold occurs when:

$$\frac{S_{RE}}{S_{RD}} = m$$

We then define the break-even multiple use expansion ratio, X, to be:

$$\frac{S_{RE}}{S_{RD}} = \frac{\sum_{i=1}^{n} N_i \cdot S_{Si}}{\sum_{i=1}^{n} S_{Si}} = X$$

Multiple use or reuse within a program is desirable if $T_S > 0$, i.e. if:

$$\frac{S_{RE}}{S_{RD}} = \frac{\sum_{i=1}^{n} N_i \cdot S_{si}}{\sum_{i=1}^{n} S_{si}} > m$$

As long as the value of the parameter m is less than the value of the multiple use expansion ratio, it is possible to realize savings through multiple use of software components to compose the new software in an application system.

Using the definition of X provided above, we can rewrite the expression for C_N given earlier, and then develop the expression for C_{VN}, the unit cost of the new code (including the multiply used code) for a software system. This expression can be employed to show the effect of multiply use of software components on the value of C_{VN} (to reduce it from the value it would have were there no multiply use).

Recall the expression for C_N developed above:

$$C_N = C_{VN} \cdot S_N = C_{VNB} \cdot S_U + m \cdot C_{VNBD} \cdot S_{RD} + C_{VNBT} \cdot S_{RE}$$

Dividing through by S_N and taking the definition of X into account, we have:

$$C_{VN} = C_{VNB} \cdot (S_U / \cdot S_N) + [\,(m/X) \cdot C_{VNBD} + C_{VNBT}] \cdot (S_{RE}/S_N)$$

Note that if there is no intra-system reuse, $S_{RE} = 0$, $S_U = S_N$, and $C_{VN} = C_{VNB}$. Whereas, when there is intra-system reuse, C_{VN} is reduced from this value to one which we can term C_{VNR}.

Model Application

The model is applied to the unit costs in Table 4 and to the software multiple use data in Tables 2 and 3. The code referred to in this example (Sutton 1991) was not actually coded in Ada. However, that is irrelevant to the use of its code counts in the example application of the model equations.

Table 2. Multiple Use Sizes

Reuse Parameter	Size in KSLOC
S_N	93.393
S_U	23.483
S_{RD}	11.414

In the present case there are n = 3 groups of multiply used code, and the values for the the three S_{Si} and N_i are given in Table 3.

Table 3. Application Data

Group (i)	Number of Copies (N_i)	Amount of Code in KSLOC (S_{Si})	N_iS_{Si}
1	5	7.070	35.35
2	2	1.110	2.22
3	10	3.234	32.34
	Total	11.414	69.91

Thus, $S_{RE} = 69.91$ KSLOC.

The multiple use expansion ratio for this example then is:

$$X = \frac{S_{RE}}{S_{RD}} = \frac{69.91}{11.414} = 6.05$$

Thus 6.05 is the maximum value of m that could exist for break-even to occur with intra-system reuse in the present case. If m is less than this figure, a return will be realized on the investment in creating software units for multiple use in this system. This 6.05 figure is much larger than the value of m of 1.5 cited earlier. Hence, we would expect a good return on the investment in developing units suitable for multiple use and using them in this system.

Table 4 shows some unit costs by activity that are characteristic of embedded Ada software.

Table 4. Ada Software Development Unit Costs

Activity	Symbol	Unit Costs, LM/KSLOC	% Of Dev. Process
Requirements Analysis	C_{RA}	0.74	7.4
Preliminary Design	C_{PD}	1.67	16.7
Detailed Design	C_{DD}	2.22	22.2
Code and Unit Test	C_{CUT}	2.22	22.2
CSC Integration Test	C_{CIT}	1.60	16.0
CSCI Test	C_{CT}	1.55	15.5
Totals	—	10.00	100.0

Using the equations developed above, and letting $m = 1.5$, we find that $C_N = 933.93$ LM, the cost for development without reuse., and $C_{NR} = 572.23$ LM, the cost of development with reuse. That is, $C_{NR} = 0.613 \cdot C_N$. Also, note that the savings achievable

through reuse,T_S, in this case is 361.70 LM, or 38.7 percent of C_N. This result could be computed using the equation developed above for C_{VN} as a function of m, X, etc.

Observations About Domain Engineering

Both large scale reuse across systems as well as multiple use within a system can be difficult to achieve unless the software components to be reused and/or multiply used have been developed based on a domain analysis. Domain analysis is a major activity of domain engineering which is devoted to the creation of software components that can be used in more than one instance organized in a domain.

Scope of the Term "Multiple Use"

It is important to note that as stated earlier, multiple use can consist of making N copies of each multiply used component and distributing them through the system as required. However, the cost model of reuse within a single system also covers the case where just one copy of each of the desired components is made and these single copies are shared by the rest of the system. Subroutines that are callable from any point in the computer program are such an example. Multiply used code is actually shared code.

References

Cruickshank, R.D., and J.E. Gaffney Jr. 1991 — An Economics Model of Software Reuse. *Conference on Analytical Methods in Software Engineering Economics.* McLean, Virginia: MITRE Corp.

Cruickshank, R.D., and M. Lesser 1982 — An Approach to Estimating and Controlling Software Development Costs. *The Economics of Data Processing.* New York, New York: Wiley.

Department of Defense 1988a — *Military Standard – Configuration Control, Engineering Changes, and Waivers,* DoD-STD-480B. Washington, D.C.

Department of Defense 1988b — *Military Standard – Defense System Software Development,* DoD-STD-2167A. Washington, D.C.

Gaffney, J.E. Jr. 1983 — *Approaches To Estimating And Controlling Software Costs,* Computer Measurement Group International Conference on Computer Performance Evaluation, 1983, Washington, D.C.

Gaffney, J.E. Jr. 1989 — *An Economics Foundation for Software Reuse*. Herndon, Virginia: Software Productivity Consortium and AIAA *Conference on Computers,* Monterey, California.

Gaffney, J.E. Jr., and T. Durek 1991 — Software Reuse—Key to Enhanced Productivity: Some Quantitative Models. *The Economics of Information Systems and Software.* pp 204-219. R. Veryard, ed. Butterworth-Heinemann, Oxford, England.

Institute of Electrical and Electronic Engineers 1992 — IEEE Standard for Software Productivity Metrics (P1045). Draft 5.0.

Parnas, D.L. 1976 — On the Design and Development of Program Families. *IEEE Transactions on Software Engineering* SE-2:1.

FORESTRY AS AN ALTERNATIVE METAPHOR FOR SOFTWARE DEVELOPMENT: APPLYING MULTIOBJECTIVE IMPACT ANALYSIS

Gregory K. SHEA
The Software Productivity Consortium
2214 Rock Hill Road
Herndon, Virginia 22070 USA

Clement L. MCGOWAN
The MITRE Corporation
7525 Colshire Drive
McLean, Virginia 22102 USA

I. USING METAPHORS TO UNDERSTAND

Metaphors draw parallels between dissimilar situations. Metaphors help us understand and "see" one situation in terms of another. For example, the "evening of life" from the metaphor "a life is like a day" suggests a life, mostly completed, moving inexorably to its end. Metaphors sometimes have the power of poetry to change how we feel about and see something familiar.

Indeed, as George Lakoff observes [11], "the way we use imaginative mechanisms [is] central to how we construct categories to make sense of experience." Metaphors are one of our principal imaginative mechanisms. Metaphors provide us with (i) a distinctive but partial perspective, (ii) a way to think, see, and discuss, (iii) new insights, and even (iv) "transferred" feelings because we focus on the parallels. Analogies help us to understand.

The language we use to describe a situation often contains hidden, implicit metaphors. For example, Lakoff [11] presents several metaphors for anger including the heat of a fluid in a container and an opponent in a struggle. Phrases about anger such as "lost his cool," "reached the boiling point," "simmer down," "all steamed up," "blew up, exploded" invoke the comparison with heating a fluid in a covered container. Similarly anger as an opponent is implied by verb phrases such as battle, overcome by, subdue, wrestle with, seized by, yielded to, and fought back.

Metaphors for processes can give us insight, but when we adopt a single (implicit) metaphor for a process it can dramatically influence our subsequent actions, decisions, success measures, feelings, and even how we see people participating in the process. For example, Gareth Morgan [14] examines some images of business organizations. The organization as a machine is one pervasive metaphor that sees a business organization as rationally divided into distinct units each with defined responsibilities. This machine view promotes good communication paths for directives and status reports in order to control the system. This metaphor leads naturally to efficiency studies to "tune" the machine, but not to a fundamental restructuring. The "organization as a machine" view also downplays human aspects (e.g., filling a vacancy becomes "fitting a resource into a slot").

There are strengths and weaknesses, insights and blind spots associated with any compelling metaphor for a large process. When we view a business organization as a political system, we analyze actions and recommendations in terms of stakeholders. The political metaphor helps us to deal with power, interest groups, and conflict resolution. However, this particular metaphor, when used exclusively, creates a cynical, politicized attitude (e.g., always looking for hidden agendas).

II. Software Development as Manufacturing

The dominant metaphor for software development is the manufacturing assembly line. The phases of development (e.g., define the user requirements, design the overall system as interacting components, detail the workings of the components, code the components, test, and integrate) are seen as stations in an assembly line that produce intermediate products (or sub-assemblies). Over time this software assembly line transforms user needs into a working system.

This prevailing manufacturing metaphor directly influences the models and methods used to improve the engineering and economics of software. Most importantly, the software development work flow was seen as a process to be defined, institutionalized, monitored, automated, and improved. This was a fundamental advance. In addition, scheduling techniques from manufacturing dealt with parallel

development tasks and helped to identify critical paths. The basic measures of productivity, costs, and quality from manufacturing were adapted to software development.

Today, manufacturing quality assurance techniques (such as statistical process control) from Deming and Juran [4, 8] are applied analogously to the software development process. Improving the quality (in terms of defect density) of intermediate products (e.g., design documents or code) on the critical path should improve, it is believed, the overall quality of the final product and the productivity of the process.

A. Manufacturing Metaphor Weaknesses

The difficulty with the manufacturing metaphor is that focusing improvement efforts, as in manufacturing, on keeping outputs within tolerance doesn't address the principal risks of software development [3]. Manufacturing assumes that function follows form. That is, if the parts (e.g., design or code modules) are close enough to their specifications, then the parts will function correctly. This manufacturing view largely ignores software system design, which is the allocation of computational responsibilities and interfaces to components. A deign must satisfy clearly the identified requirements and constraints and subsequently absorb changes to the requirements. Redesign and reallocation of components and interfaces is an important – and risky – part of software development. Besides ignoring design, the manufacturing metaphor for software development gives little insight into how to go about making tradeoffs among objectives (e.g., cost, functionality, schedule, reliability, quality) [2].

B. Seeking an Alternative Metaphor

We seek an alternative metaphor for software development. This alternative should stress that form follows function by having planning and designing as principal activities. The new metaphor should deal with multiple conflicting

objectives, tradeoffs and satisficing. It should also handle constraints, mid-development changes, and system evolution over time (i.e., continued development after initial "delivery"). Our new metaphor should guide us in how we monitor software development, detect problems (or breakdowns) and then intervene and correct the situation (in an engineering sense). That is, our metaphor should correspond better to the realities of software development than does the assembly line metaphor.

In addition, a well-chosen metaphor for software development will suggest mathematical and analytic techniques to apply. Such techniques would help us to evaluate the effects and the tradeoffs of different courses of action. Further, the alternative metaphor should suggest how we accommodate new software development methods and changes to (or greater quantitative knowledge of) existing methods.

With the above goals and criteria for a new metaphor for software development, we have identified three natural candidates. These candidates are (i) urban planning dealing with zoning, infrastructures, and building codes, (ii) civil engineering with the design, plan, and construction of bridges, dams, etc., and (iii) public sector forestry. These three candidate metaphors have many parallels in terms of the issues dealt with and the mathematical techniques employed. Partially for its novelty, and principally for its parallels to essential software development concerns, we use the planning and developing of a public forest as our alternative metaphor.

III. Software Development as Forestry

Planning a large, public forest involves three main steps. First, the objectives are listed in order of importance with a particular emphasis on time scale (i.e., system properties that take longer to develop or must be maintained over an extended time span). Second, the proposed forest is specified and decomposed down to the stand level (i.e., analogous to the component of detailed design). This decomposition is an iterative process with much rework to deal with system constraints, to discover any implicit conflicts in the objectives, and to mitigate risks

(including possible changes in or extensions to the original objectives over time). Third, forest planners determine ways to monitor the forest-under-development for predicted progress towards the objectives. Monitoring a forest during development involves multiple actions (e.g., samples, tests, observations) with different time and space granularities.

A. Forestry Objectives and Constraints

To plan a public forest, one must deal with multiple objectives and constraints [16]. Typically the public forest must: be economically self-supporting, have a specified quality and variety (species) of trees, conform to stringent water quality standards, support natural wildlife as well as recreational activities, yield sufficient timber harvests over time, and be aesthetically pleasing. Planners must view the forest (i.e., the software-to-be-developed) as a system. This means they must project the system and its environment over time and plan for possible changes. Forest planners need to optimize for both long-term and short-term performance. During development they must evaluate corrective actions based on overall impact, not just as localized improvements.

B. Forestry Design

The design of a public forest offers many interesting insights into, and parallels with, the realities of software design. Some of these parallels are:

- Rank system requirements (objectives) in order based on the time they take to achieve.
- Partition the system (forest) into subsystems (zones) that trace to requirements (objectives).
- Evaluate the design against a sufficiency level for each objective.
- Adjust the interfaces (zone boundaries) as needed.
- Partition the subsystems (zones) into components (stands) to assign work and to monitor.

- Project (estimate) over time the properties and characteristics of the components (stands).

Building in quality is fundamental to the forestry design philosophy. You get quality by good design, good (silvicultural) techniques, and good execution. You cannot get adequate quality by inspection (of stands). From the design itself you can derive the metrics to monitor the system under development. Your should determine your choice of state variables (i.e., what to monitor and measure) from the system requirements (objectives). Then you aim to express the consequences of development actions (and of uncertain behaviors) in terms of the state variables.

C. Forestry Monitoring

The monitoring and measuring of a forest provides a new perspective on software metrics. The goal of monitoring is to uncover problems early both to minimize their effects and to optimize the development and monitoring activities with respect to the system objectives. To monitor we inspect and measure components (stands) and compare the measures against projections (such as rate of growth, health, amount). There are standard remedial actions to correct problems (such as fire or insect damage). Measuring and monitoring activities continue during maintenance. Techniques are used to optimize the frequency and the timing of component (stand) inspections.

D. Operations Research Techniques Applied to Forestry

We contend that forestry planning and engineering is a better metaphor for software development than is the manufacturing assembly line. Forestry is an application area where operations research techniques have been used extensively. Lembersky and Johnson [12] give a thorough description of a model, variables, solution method, and resources. Kennedy [10] covers dynamic programming applications in forestry management. Harrison and Rosenthal [6] use a nonlinear utility function with eight objectives tailored individually to each landowner. And

Anderson et al. [1] deal with the multiobjective nature of managing public forests. They use non-interior set estimation instead of preference-based goal programming.

In what follows, we will exploit the forestry metaphor by showing how multiobjective, multistage operations research techniques previously applied to forestry may be applied to software development. In particular, all objectives and criteria are not mapped to a single common unit (e.g., dollar cost); in fact, objectives are often incommensurable, such as system reliability and software component size. Instead, alternative actions and interventions at decision points are evaluated against their impact on the multiple end objectives (i.e., optimize for end goals rather than for a particular local property of an intermediate product, such as the quality of a design document).

IV. Applying (Forestry-Inspired) Techniques to Software Development

As stated earlier, our metaphor requires optimizing for end goals while accommodating development constraints. This suggests a decision framework which takes the form of a *constrained nonlinear program* [13] like that shown below:

$$\begin{aligned} &\text{minimize } \mathbf{f}(\mathbf{x}) \\ &\text{subject to } \mathbf{h}(\mathbf{x}) = \mathbf{0}, \\ &\qquad \mathbf{g}(\mathbf{x}) \leq \mathbf{0}, \\ &\qquad \mathbf{x} \in \Omega \end{aligned} \tag{1}$$

where $\mathbf{x}$ is the state vector (e.g., cost or schedule); $\mathbf{f}(\mathbf{x})$ represents a vector objective function (e.g., dollars or labor-hours) to be optimized; $\mathbf{h}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ are vector constraints (e.g., budget or work-week), and Ω is the feasible solution space (defined in Section IV.B). Although Equation (1) captures the essence of constrained optimization, it does not give much insight into handling multiple, often conflicting and incommensurate, objectives. It also poorly captures the dynamic nature of forest development.

A. MULTIOBJECTIVE MULTISTAGE OPTIMIZATION

One adaptation which does capture the optimization of multiple objectives over time is *multiobjective multistage optimization* [5] where the software system is represented as a state vector, and the stage-wise evolution of the system is described by the following two equations:

$$\begin{aligned} x(k+1) &= g(x(k), u(k), k) \quad , \qquad k = 0,1,\cdots \\ x(0) &= x_0 \end{aligned} \tag{2}$$

given

$$g(\cdot,\cdot,k) : \Re^n \times \Re^r \rightarrow \Re^n \tag{3}$$

where $x(k) \in \Re^n$ denotes the state of the system at (development) stage k, $u(k) \in \Re^r$ denotes the decision (policy or action) at stage k, $g(\cdot,\cdot,k)$ is a function representing the transition of the system from state $x(k)$ under decision $u(k)$ at stage k, and x_0 is the initial state.

Assume that the (software development) process has T stages, and that all system objectives are elements of a set $\mathcal{F}$, defined by:

$$\mathcal{F} = \left\{ J_i : J_i = f_i^k(x(k), u(k), k), \quad i = 1,2,\ldots,N_k ; \quad k = 0,1,\ldots,T-1 \right\} \tag{4}$$

where $f_i^k(x(k),u(k),k)$ is a function representing the i objectives of interest at stage k. Based on the previous four equations, the optimization problem can be expressed as:

$$\min_u \begin{bmatrix} f_1^0(x,u;0) \\ \vdots \\ f_{N_0}^0(x,u;0) \end{bmatrix}, \ldots, \min_u \begin{bmatrix} f_1^1(x,u;1) \\ \vdots \\ f_{N_1}^1(x,u;1) \end{bmatrix}, \ldots, \min_u \begin{bmatrix} f_1^{T-1}(x,u;T-1) \\ \vdots \\ f_{N_{T-1}}^{T-1}(x,u;T-1) \end{bmatrix} \tag{5}$$

subject to:

$$x(k+1) = g(x(k),u(k),k), \quad k = 0,\cdots,T-1$$
$$x(0) = x_0, \tag{6}$$

given

$$(x(k),u(k)) \in \Omega_k \subseteq \Re^n \times \Re^r, \quad k = 0,\cdots,T-1$$
$$x(T) \in \Omega_T \subseteq \Re^n \tag{7}$$

where $f_i^k(x,u,k) \equiv f_i^k(x(k),u(k),k)$; N_k indexes the objectives of interest at stage k; $(x(k),u(k)) = [x_1(k),\cdots,x_n(k), u_1(k),\cdots,u_r(k)]'$ (where ' denotes transpose); and the feasible solution space sets ($\Omega_k, k = 0,\cdots,T$ and Ω_T) are each specified by a system of inequality constraints.

From a software development perspective, $\mathcal{F}$ represents the set of process goals. Furthermore, Equation (4) permits sub-goals to be defined for each phase of the development cycle. The optimization problem outlined in Equation (5) involves solving a sequence of single-stage, multiobjective optimization problems where the decisions made at stage k affect stages $k+1,k+2,\ldots,T-1$. This is because the objectives in Equation (4) are themselves a function of the states which evolve according to Equation (2).

B. Definitions

To demonstrate how this model accommodates multiple objectives that often conflict, we must first define the following concepts:

> ***Policy Decision.*** A policy decision is any pair (X,U) where $U = [u(0),\cdots,u(T-1)]$ is the decision sequence and $X = [x(0),\cdots,x(T)]$ is the corresponding state trajectory determined by Equation (2).

Feasible Decision. A policy decision (X,U) is feasible if it satisfies both the decision and state space constraints, shown in Equation (7).

Noninferior Decision. A feasible policy decision (X^*,U^*) is said to be noninferior if no other feasible policy decision (X,U) exists such that $f_i^k(x,u,k) \leq f_i^k(x^*,u^*,k)\ \forall i \in N_k \equiv 1,2,\cdots,N_k, k=0,\cdots,T-1$ with strict inequality holding for at least one i and k. Otherwise it is inferior, and is said to be dominated by some other noninferior decision.

Preferred Decision. A noninferior policy decision (X^*,U^*) is a preferred policy decision if it is preferred by the decision maker over all other noninferior policy decisions.

The definition for a noninferior decision shows that it is not sufficient simply to obtain noninferior solutions at each stage (strict inequality holding for at least one i) but rather to obtain noninferior solutions for the entire process horizon (strict inequality holding for at lest one k). The examples shown later will demonstrate that for any given noninferior policy decision, the only way to increase the value of one objective function is to decrease the value of some other objective function in that same stage or decrease the value of any objective function in another stage.

C. MARKOV DECISION PROCESS

Another approach involves modeling Equation (1) as a *Markov Decision Process* (MDP). In its most general form, the MDP, shown in Equation (8), is developed and solved as follows [7]:

1. The problem is divided into states which completely describe all of the possible conditions that the system could be in at any point in time.
2. Possible actions to be taken, or alternatives, are determined for each state of the system.

3. The probability of going from one state to another, given that a specific action is chosen, is assessed for all states.
4. A reward (be it financial, preferential, or some combination form) is associated with the transition from one state to another, under each alternative in each state.
5. The process is examined over an appropriate period of time, and the expected reward of the various actions upon the states of the system is maximized.

This is expressed mathematically as:

$$\underline{v}_n' = \max_a \left[\underline{Q}_m^a + \beta \sum_{n=1}^{N} p_{mn}^a \underline{v}_n \right] \quad \text{for each } m,\ m = 1,\ldots,N \tag{8}$$

where:

$\underline{v}_n \equiv$ The present value of the previous policy (i-dimensional vector)

$m,n \equiv$ States of the system

$a \equiv$ Alternative

$\underline{Q}_m^a \equiv$ Expected reward from a single transition from state m under alternative a (i-dimensional vector)

$\beta \equiv$ Discount factor $= \dfrac{1}{1 + \text{interest rate}}$

$p_{mn}^a \equiv$ Probability of going from state m to state n given that alternative a is chosen

$\underline{v}_n' \equiv$ The present value of the current policy (i-dimensional vector)

Equation (8) is solved by iteration until

$$\underline{v}_n' = \underline{v}_n \quad \forall n,\ n = 1,\ldots,N \tag{9}$$

The MDP representation is a good one for software development because:

- Formal process models frequently make use of a state representation [19,15].
- Probabilistic models capture the intuitive notion that movement from a given state to a more desirable state is not necessarily a deterministic function; however, it may be influenced by the actions taken.
- Costs are usually incurred in moving from a given state to any other state.

A collection of best alternatives for each state of the system forms an optimal policy and can be likened to the noninferior policy decision defined earlier. In systems where the state transition is probabilistic, Equation (8) can be substituted for Equation (2).

V. EXAMPLES

The following four examples focus on the verification aspects of software development. Imagine a three-stage ($k = 3$) verification process consisting of design, code, and integration stages. At each stage, one may choose any combination of verification methods available for use at that stage, or no method at all. This is represented in Figure 1.

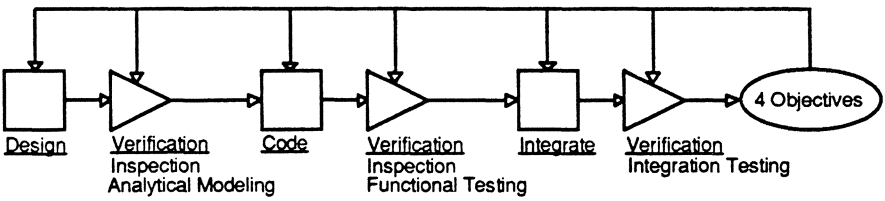


Figure 1. Three-stage Verification Process Representation

For this sample verification process, four objectives are important:

- Control errors per KSLOC ($i = 1$)

- Interface errors per KSLOC ($i = 2$)
- Labor cost (in dollars) per KSLOC ($i = 3$)
- Machine cost (in dollars) per KSLOC ($i = 4$)

The measured effects of the verification methods upon the four objectives, by stage, are shown in Table 1. The entries are defined as follows:

g_1^k: Proportion of control errors detected

g_2^k: Proportion of interface errors detected

g_3^k: Labor costs ($K/KSLOC)

g_4^k: Machine costs ($K/KSLOC)

Remember that choosing neither method (in effect, performing no verification) at a given stage is always a choice.

Table 1. Verification Method Data by Stage[1]

a	k	Stage	Method=u(k)	g_1^k	g_2^k	g_3^k	g_4^k
1	1	Design	Inspection	0.60	0.80	300	0
2	1	Design	An. Model.	0.90	0.90	3000	300
3	1	Design	Both	0.94	0.97	3300	300
4	1	Design	Neither	0.00	0.00	0	0
5	2	Code	Inspection	0.60	0.80	1200	0
6	2	Code	Func. Test.	0.80	0.60	1800	200
7	2	Code	Both	0.90	0.90	2600	200
8	2	Code	Neither	0.00	0.00	0	0

[1]Data derived from Basili [2] and the Software Productivity Consortium.

9	3	Integrate	Int. Test.	0.50	0.90	900	100
10	3	Integrate	Neither	0.00	0.00	0	0

The state of the system (state vector) at any given stage is represented by a four-tuple consisting of: control error density, interface error density, labor cost/KSLOC, and machine cost/KSLOC. For purposes of illustration, suppose that state transitions are defined by:

$$\begin{aligned} x_i^{(k+1)} &= \left[1 - g_i^k u(k)\right] x_i^k \quad \text{for} \quad i = 1,2 \\ x_i^{(k+1)} &= g_i^k u(k) + x_i^k \quad \text{for} \quad i = 3,4 \end{aligned} \tag{10}$$

If the initial state of the system is defined as:

$$x_0 = (9.000 \text{ control errors/ KSLOC, } 9.000 \text{ interface errors/ KSLOC,}$$
$$\$0 \text{ labor / KSLOC, } \$0 \text{ machine / KSLOC})$$

Equation (10) can be applied to the data in Table 1 to generate the policy space after each stage. Table 2 shows the thirty-two policies (composed of a sequence of actions) and their effects upon the four-objective state of the system after the integration stage. Similar tables can be generated for the policy space after the design and code stages.

Table 2. Policy Space after Integration Stage

Policy	Design	Code	Integrate	Control Errors per KSLOC	Interface Errors per KSLOC	Labor Costs (K$) per KSLOC	Machine Costs (K$) per KSLOC
1	Inspection	Inspection	Int. Test.	0.720	0.072	2400	100
2	Inspection	Inspection	Neither	1.440	0.720	1500	0
3	Inspection	Func. Test.	Int. Test.	0.360	0.036	3000	300
4	Inspection	Func. Test.	Neither	0.720	0.360	2100	200
5	Inspection	Both	Int. Test.	0.180	0.018	3800	300
6	Inspection	Both	Neither	0.360	0.180	2900	200
7	Inspection	Neither	Int. Test.	1.800	0.180	1200	100
8	Inspection	Neither	Neither	3.600	1.800	300	0
9	An. Model.	Inspection	Int. Test.	0.180	0.036	5100	400
10	An. Model.	Inspection	Neither	0.360	0.360	4200	300
11	An. Model.	Func. Test.	Int. Test.	0.090	0.018	5700	600
12	An. Model.	Func. Test.	Neither	0.180	0.180	4800	500
13	An. Model.	Both	Int. Test.	0.045	0.009	6500	600
14	An. Model.	Both	Neither	0.090	0.090	5600	500
15	An. Model.	Neither	Int. Test.	0.450	0.090	3900	400
16	An. Model.	Neither	Neither	0.900	0.900	3000	300
17	Both	Inspection	Int. Test.	0.108	0.011	5400	400

18	Both	Inspection	Neither	0.216	0.108	4500	300
19	Both	Func.Test.	Int. Test.	0.054	0.005	6000	600
20	Both	Func. Test.	Neither	0.108	0.003	5100	500
21	Both	Both	Int. Test.	0.027	0.027	6800	600
22	Both	Both	Neither	0.054	0.027	5900	500
23	Both	Neither	Int. Test.	0.270	0.027	4200	400
24	Both	Neither	Neither	0.540	0.270	3300	300
25	Neither	Inspection	Int. Test.	1.800	0.360	2100	100
26	Neither	Inspection	Neither	3.600	3.600	1200	0
27	Neither	Func. Test.	Int. Test.	0.900	0.180	2700	300
28	Neither	Func. Test.	Neither	1.800	1.800	1800	200
29	Neither	Both	Int. Test.	0.450	0.090	3500	300
30	Neither	Both	Neither	0.900	0.900	2600	200
31	Neither	Neither	Int. Test.	4.500	0.900	900	100
32	Neither	Neither	Neither	9.000	9.000	0	0

A. Example One

Suppose one wanted to minimize total labor costs while constraining the final control error density to no more than 0.300 errors/KSLOC, final interface error density to no more than 0.100 errors/KSLOC, and total machine costs to no more than $400K/KSLOC. From Table 2, the feasible solution space consists of policies 9, 17, and 23; the noninferior policies are also 9, 17, and 23. If a hypothetical decision maker wanted to minimize control errors among the noninferior choices, policy 17

would be the preferred decision, as shown below (remember that $U^*(k)$ refers to the recommended action at stage k):

$U^*(1)$ = use both verification methods
$U^*(2)$ = perform inspection
$U^*(3)$ = perform integration testing

with: labor cost = \$5400K/KSLOC
control error density = 0.108 errors/KSLOC
interface error density = 0.011 errors/KSLOC
machine cost = \$400K/KSLOC.

B. Example Two

Suppose one instead wanted to minimize the final control error density while constraining the final interface error density to no more than 0.100 errors/KSLOC, total labor costs to no more than \$4000K/KSLOC, and total machine costs to no more than \$400K/KSLOC. From Table 2, the feasible solution space consists of policies 1, 3, 5, 15, and 29 (note that policies 15 and 29 have the same control error densities). Since policy 29 dominates policy 15 and policy 3 dominates policy 29, the noninferior policies are 1, 3, and 5. If a hypothetical decision maker wanted to minimize interface errors among the noninferior choices, policy 5 would be the preferred decision, as shown below:

$U^*(1)$ = perform inspection
$U^*(2)$ = use both verification methods
$U^*(3)$ = perform integration testing

with: interface error density = 0.018 errors/KSLOC
control error density = 0.180 errors/KSLOC
labor cost = \$3800K/KSLOC
machine cost = \$300K/KSLOC.

C. EXAMPLE THREE

Now suppose that one wanted to minimize the final control error density while constraining total labor costs to no more than \$5500K/KSLOC, and total machine costs to no more than \$500K/KSLOC. In addition, interface error density limits, or "goals," have been specified for each stage. Specifically, interface errors may not exceed:

- 1.000 errors/KSLOC at design
- 0.500 errors/KSLOC at code
- 0.100 errors/KSLOC at integration

This is an example of making tradeoffs across all three stages. From Table 2, the feasible solution space consists of policies 1, 3, 5, 6, 9, 15, 17, 20, 23, and 29. Policies 1, 15, and 29 violate the goals at code, and policies 3, 5, and 6 violate the goals at design (these tables are not shown). Therefore, the remaining feasible policies are 9, 17, 20, and 23; they are all noninferior. If a hypothetical decision maker wanted to minimize labor costs among the noninferior choices, policy 23 would be the preferred decision:

$U^*(1)$ = use both verification methods
$U^*(2)$ = no action
$U^*(3)$ = perform integration testing

with: labor cost = \$4200K/KSLOC
control error density = 0.270 errors/KSLOC
interface error density = 0.027 errors/KSLOC
machine cost = \$400K/KSLOC.

D. EXAMPLE FOUR

Finally, suppose that one wished to keep the final control error density at or below 1.500 errors/KSLOC and the final interface error density at or below 0.500 errors/KSLOC. Furthermore, suppose that the verification process has a predetermined machine budget: spend not more than \$200K/KSLOC on machine verification by the code stage, and not more than \$500K/KSLOC on machine verification by the integration stage. Verification labor costs are unrestricted at design, but should be kept to a minimum at the code and integration stages. This is an example of tradeoffs being made at given stages (e.g., labor costs at code and integration stages) and tradeoffs being made across stages (e.g., constrained machine verification costs at code and integration stages). At the integration stage, policies 7, 8, 25, 26, 28, 31, and 32 violated the control error constraint; policies 2, 16, 26, 28, 30, 31, and 32 violate the interface error constraint, and policies 11, 13, 19, and 21 violate the machine cost constraint. At the code stage (not shown), policies 9, 10, 12, 14, 15, 17, 18, 20, 22, 23, and 24 violate the machine cost constraint; in addition, policy 1 dominates policy 27 and policy 3 dominates policy 29. As a result, the feasible policies are 1, 3, 4, 5, and 6; they are all noninferior. If a hypothetical decision maker wanted to minimize control and interface errors among the noninferior choices, policy 5 would be the preferred decision, as shown below:

$U^*(1)$ = perform inspection
$U^*(2)$ = use both verification methods
$U^*(3)$ = perform integration testing

with: control error density = 0.180 errors/KSLOC
interface error density = 0.018 errors/KSLOC
labor cost = \$3800K/KSLOC
machine cost = \$300K/KSLOC

While these four examples do not constitute a definitive evaluation, they serve to illustrate the potential value of taking quantitative analysis techniques from the forestry sector and applying them to software development.

VI. Conclusions

We find that metaphors help us see and understand; however, they can also blind us to other perspectives. The use of forestry as metaphor for software development has some advantages over the manufacturing metaphor. In particular, forestry emphasizes designing and monitoring to deal with many objectives, constraints, and changes. Inspired by applications to forestry, we propose to apply multistage, multiobjective optimization to software development. The examples illustrate a promising evaluative framework to: decide on alternative development actions based on their overall system impact (with respect to user-specified goals); direct future data gathering; and quantify the cost/benefits of individual and combined software development actions.

REFERENCES

[1] Anderson, A. et al. "Systems Analysis for the Forest Sector," *TIMS Studies in Management Sciences* **21** (1986):1-23

[2] Boehm, B. and P. Papaccio, "Understanding and Controlling Costs," *IEEE Transactions on Software Engineering*, **SE-14** (1988):1462-1477.

[3] Bollinger, T. and C. McGowan, "A Critical Look at Software Capability Evaluations," *IEEE Software* **8**, 4 (1991): 25-41.

[4] Deming, W.E., *Out of the Crisis*, Cambridge, Massachusetts: MIT Press, 1982.

[5] Gomide, F. and Y. Haimes, "The Multiobjective Multistage Impact Analysis Method: Theoretical Basis," *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-14** (1984):88-98.

[6] Harrison, T. and R. Rosenthal, "A Multiobjective Optimization Approach for Scheduling Timber Harvests on Nonindustrial Private Forest Lands," *TIMS Studies in Management Sciences* **21** (1986),:269-283.

[7] Howard, R., *Dynamic Programming and Markov Processes*, Cambridge, Massachusetts: MIT Press, 1964.

[8] Juran, J.M., *Juran on Planning for Quality*, New York: Macmillan, 1988.

[9] Kellner, M., "Representation Formalisms for Software Process Modeling," *Proceedings of the 4th International Software Process Workshop*, Devon, UK, 1989.

[10] Kennedy, J., *Dynamic Programming: Applications to Agriculture and Natural Resources*, Essex, England: Elsevier Publishers, 1986.

[11] Lakoff, G., *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*, Chicago: Univ. of Chicago Press, 1987.

[12] Lembersky, M. and K. Johnson, "Optimal Polices for Managed Strands: An Infinite Horizon Markov Decision Process Approach," *Forest Science* **21**, 2 (1975):107-122.

[13] Luenberger, D., *Linear and Nonlinear Programming*, Reading, Massachusetts: Addison Wesley, 1984.

[14] Morgan, G., *Images of Organization*, London: Sage Publications, 1986.

[15] Rombach, H., "A Specification Framework for Software Process: Formal Specification and Derivation of Information Base Requirements," *Proceedings of the 4th International Software Process Workshop*, Devon, UK, 1989.

[16] Virginia Department of Forestry. *Management Plan for Appomattox-Buckingham State Forest* (Revised, 1986). By S. F. Warner. Charlottesville, Virginia, 1983.

TOOLS FOR MANAGING REPOSITORY OBJECTS

Rajiv D. Banker
Carlson School of Management
University of Minnesota

Tomas Isakowitz
and
Robert J. Kauffman
Stern School of Business
New York University

Rachna Kumar
College of Business Administration
University of Texas at Austin

Dani Zweig
Department of Administrative Sciences
Naval Postgraduate School

1. AUTOMATING REPOSITORY EVALUATION

The past few years have seen the introduction of repository-based computer aided software engineering (CASE) tools which may finally enable us to develop software which is reliable and affordable. With the new tools come new challenges for management: Repository-based CASE changes software development to such an extent that traditional approaches to estimation, performance, and productivity assessment may no longer suffice — if they ever did. Fortunately, the same tools enable us to carry out better, more cost-effective and more timely measurement and control than was previously possible.

Automated Metrics and the Management of an Object Repository

From the perspective of senior managers of software development, there are three characteristics of the new technologies that stand out:

(1) *Productivity enhancement.* Development tasks that used to require great effort and expense may be largely automated. This changes the basis for software cost estimation and control.

(2) *Software reuse.* The repository acts as a long-term storehouse for the firm's entire application systems inventory. It stores it in a manner which makes reuse more practical. Firms that hope to achieve high levels of reuse (on the order of 50%) must move from generally encouraging reuse to explicitly managing it.

(3) *Access to measurement.* The repository holds the intermediate lifecycle outputs — of analysis and design, and not just the final software product. As a result, it becomes practical to automate the computation of the metrics which managers need in order to take full advantage of the new technologies.

Over the last several years, we have been conducting a research program to shed light on how integrated CASE supports improved software productivity and software reliability through the reuse of repository software objects. We have found that successful management of this effort depends upon a number of factors:

(1) the reliability of cost estimation for CASE projects, in an environment in which source lines of code are almost meaningless, and in which costs can vary by a factor of two depending on the degree of reuse achieved;

(2) the extent to which software developers effectively search a repository to identify software objects that are candidates for reuse;

(3) how software reuse is promoted and monitored; and,

(4) the extent to which various kinds of software objects (especially those which

are the most expensive to build) are actually reused.

Managers can only hope to control these factors if they can measure them, and measure them in a cost-effective manner. In practice, this means automating as much of the analysis as possible. Fortunately, our research has shown that it is feasible to do so -- and to a far greater extent than we initially envisioned. By automating a number of useful repository evaluation procedures, we can provide senior managers with new perspectives on the performance of their software development operations.

STRESS: Seer Technologies Repository Evaluation Software Suite

Our long-term study of CASE-based software development continues at several sites that deployed the same integrated CASE tools. Among them are The First Boston Corporation, a New York City-based investment bank, and Carter Hawley Hale Information Services, the data processing arm of a large Los Angeles-based retailing firm. These firms allowed us to examine extensively and report on their evolving software object repositories. (For a more detailed discussion of these studies, see [1] and [2].) Their repositories were created with an integrated CASE tool called *High Productivity Systems* (**HPS**). HPS promotes modular design, object reuse and object naming conventions. It also enables the programming of applications that can be run cooperatively on multiple operating platforms, without requiring a developer to write code in the programming language that is native to each of the platforms. Instead, HPS simplifies development, by enabling the developer to create software functions using a single fourth generation "rules language", which is then processed by a code generator and translated into whatever 3GL source languages best suit the target platforms.

The metrics which we, as researchers, needed in order to analyze software development were the same ones that the managers needed in order to control it. The primary insight which made the measurement practical was that all the

information that was needed could be derived from information that was already stored within the repository. In cooperation with The First Boston Corporation and Seer Technologies (the original developers of HPS), we began to develope the conceptual basis of *STRESS, Seer Technologies' Repository Evaluation Software Suite*, a set of automated software repository evaluation tools. At present, STRESS consists of several automated analysis tools:

(1) **FPA**, the automated *Function Point Analyzer*,
(2) **OPAL**, the *Object Points Analyzer*, a new software cost estimation capability
(3) **SRA**, the automated *Software Reuse Analyzer*,
(4) **ORCA**, the *Object Reuse Classification Analyzer*.

The remainder of this paper describes the STRESS tool set in greater detail, and discusses how it can make repository object management possible.

2. FUNCTION POINT ANALYSIS

The most commonly-used bases for estimating and controlling software costs, schedules, and productivity are *source lines of code* and *function points*. The function point methodology, which computes a point score based on the functionality provided by the system, is illustrated in Figure 1. A standard weight is assigned to each system function, based on its type and complexity (e.g., 5 points for an output of average complexity), and the total count is multiplied by an environmental complexity modifier which reflects the impact of task-specific factors.

Function point analysis, which measures the amount of data processing actually being performed by a system, has a number of advantages over counting source lines of code. Function points are language-independent, they allow for differences in task complexity between systems of similar size, and they can be estimated much earlier in the life cycle. For example, we can estimate function points

during design, when we know what the system will do, but source lines of code can't physically be counted until the end of the coding phase.

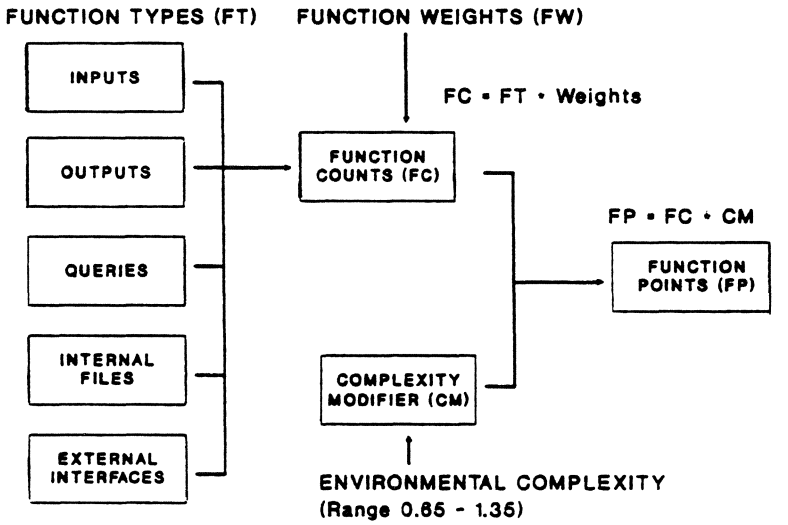


Figure 1: Function Point Analysis

Despite these benefits and others, source lines of code remain the more commonly used measure. Function point analysis requires considerable and expensive manual effort to compute, whereas the counting of source lines is easily automated. For integrated CASE environments such as HPS, however, counting source lines of code is of relatively little use: much of the functionality of the system is represented in the CASE tool's internal representation, rather than in traditional source code. Our solution was to use that internal data in automating the function point analysis.

The Function Point Analyzer (FPA)

Function point analysis has been difficult to automate in traditional software engineering environments, because it requires detailed knowledge of the system being analyzed. For example, the analyst must know whether the module which will receive a data flow is considered to be part of the application system or external to it. This

information may not be readily available, but it will determine whether the data flow counts towards the system's function point total. Or, the analyst must know how many data elements are being passed within a given data flow, as this will determine the complexity, and hence the value, of that function point contribution. Or, the analyst may have to examine the code of a data input module to make sure that the designers didn't use the same module to also perform some output function (e.g., display prompts), which might count towards the function point total. Even if such information is available in the system documentation (as in principle it ought to be, and in practice it often is not), the number of such decisions which have to be made add up to a formidable amount of paper-chasing for the analyst.

In an integrated CASE environment, most or all of this information will already be contained within the repository. The information which an integrated CASE tool must store about the system whose development it is supporting includes much or all of the information needed for the function point analysis. Different CASE tools will store the information in different ways. Figure 2 illustrates the mapping from the HPS repository representation of a software application to its equivalents in user functions.

The objects inside the application boundary on the figure are those which belong to the system being analyzed, and the lines connecting them represent calling relationships. In traditional systems, the analyst must rely upon naming conventions to determine which modules belong to a system and which don't. The analyst may also have to examine the actual code so as not to be misled by, for example, software reuse or obsolete documentation. In the integrated CASE environment, each calling relationship between a pair of objects is stored in the repository as part of the tool's knowledge about the system. The *Function Point Analyzer* (FPA) can identify the objects which are part of the application system by searching the repository.

Similarly, the repository has to know precisely what data elements are being passed to or from each and every object, in order to maintain the control and

consistency needed by an integrated environment.

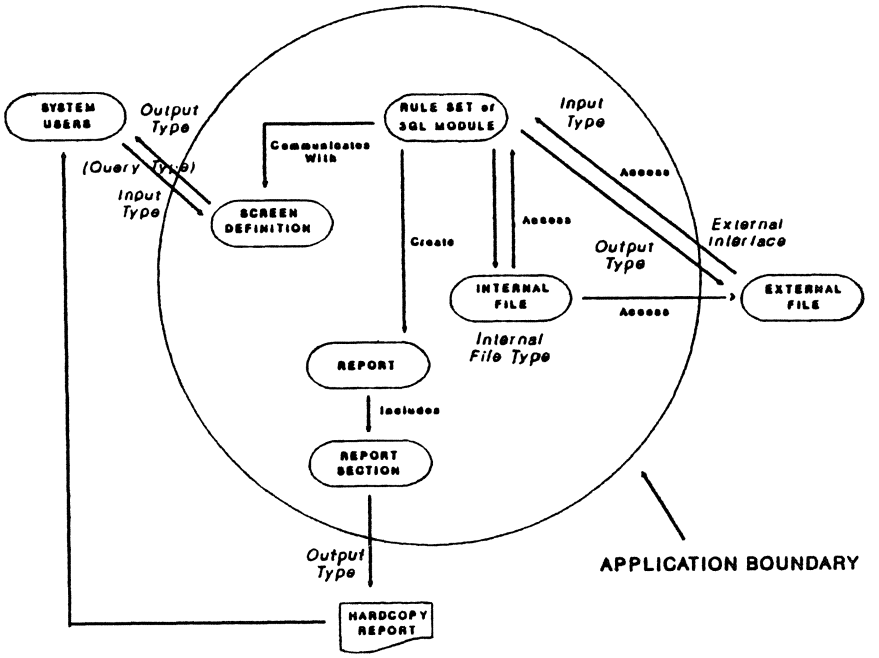


Figure 2: Mapping HPS Objects to Function Points

Determining the actual functionality of each object is the most implementation-dependent step, and the one that will vary the most from CASE tool to CASE tool. In HPS, the semantics of the 4GL *Rules Language* (a meta-language representing the objects and calling relationships that define the functionality of an application) constrain each object to a well-defined purpose (e.g., controlling one window, or generating one report segment). Since all interactions between HPS objects are mediated by database 'views', and since all database views are in the repository, the Function Point Analyzer can read the type and complexity of each data flow directly from the repository.

What all these capabilities of the Function Point Analyzer have in common is that they only depend on the information which HPS maintains, internally, *about* the system. At no point does it become necessary to examine the code itself.

3. OBJECT POINT ANALYSIS

Automating function point analysis gave us a good basis for tracking productivity improvements, both against the firms' old baselines, and against industry standards and industry leaders. Interviews with project managers, however, revealed that there were disadvantages to using Function Points as a basis for controlling individual HPS projects:

(1) Function points collapse the benefits of enhanced productivity through CASE-based automation and the benefits of software reuse.

(2) The shift to CASE was accompanied by a growing emphasis on early-life-cycle activities, particularly enterprise modelling and business analysis, and function points are more oriented towards design and post-design activities.

(3) HPS developers and managers were used to working directly with HPS objects, and the mapping from objects to function points wasn't intuitive to the managers. For the first time, the mapping was close enough that managers could think of asking for better. What they wanted was a way to use the repository objects directly, as a basis for planning and control.

In order to satisfy this demand, we had to first develop an estimation mechanism that was based on repository objects, and then demonstrate that it could equal function point analysis in predictive power and automatability.

The Object Point Analyzer (OPAL)

In an integrated CASE environment, the repository objects created in early phases of the software development life cycle will be high-level abstractions of those to be created during the coding/construction phase. The more information the repository contains about those early objects, the better our ability to make early and

reliable predictions of project costs. As was the case with the function point analyzer, the specifics of the mapping will depend upon the implementation of the CASE environment.

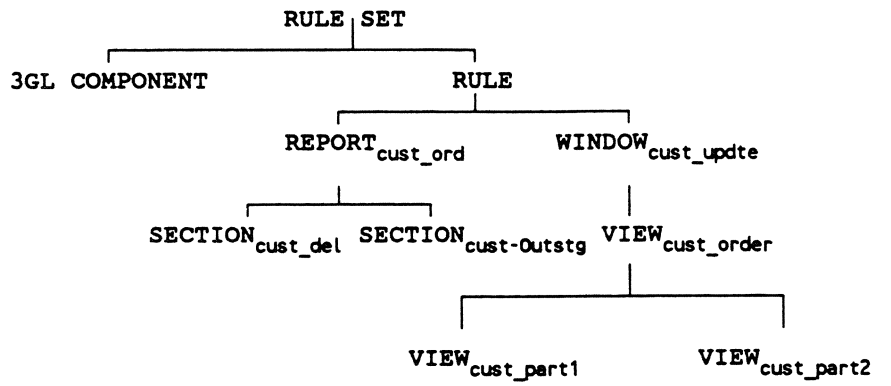
OPAL, the *Object Point AnaLyzer*, was developed as a cost estimation facility for the HPS environment. It differs from the Function Point Analyzer primarily in providing a direct mapping from HPS objects to cost estimates.

Our interviews with project managers revealed that they were already using object-based cost estimation informally, assigning so many days of development effort for each type of object. Using those informal heuristics as a starting point, we used regression analysis first to give us more precise estimation weights and later to validate these results against actual projects.

OPAL computes *object points*, a metric inspired by function points, but better suited to the ICASE environment. Object points are based directly upon the objects stored within the repository, rather than upon the interactions between those object. In HPS terms, object points are assigned for each WINDOW, for each REPORT, for each 3GL MODULE, etc. Instances of each object type can be simple, average, or complex, with the more complex objects receiving higher object point scores. The computation of object points is illustrated in Figure 3. The objects depicted are part of a much larger application. Each object is assigned a complexity rating, based on empirically derived factors such as the number of objects it calls in turn, and then an object point score.

Because the CASE environment limits the functionality allowed to each object type, this is a true measure of application system functionality as well as of programmer effort. It was practical to automate the classification of objects because of the information the repository maintains about each object. Like FPA, OPAL uses the repository's internal representation of the application system to determine which objects should be considered in the analysis, and what complexity ratings to assign

each object. The corresponding effort estimates are taken from OPAL's object-effort-weight tables. These store standard cost estimates, derived through prior empirical analysis, for simple, average, and complex instances of each object type.



OBJECT TYPE	COMPLEXITY DEFINING CHARACTERISTICS	COMPLEXITY CLASS	OBJECT POINTS
WINDOW	3 VIEWS; 1 RULE	average	2
REPORT	2 SECTIONS; 1 RULE	simple	2
3GL COMPNT		complex	8
...			

Figure 3: Illustration of Object Point Computation

We used nineteen medium-to-large software development projects to test OPAL's cost estimates against those based on function point analysis. The two estimators were found to be equally good predictors, but managers found object points easier to use and to interpret.

The results of the object point analysis can be presented in various ways, according to the requirements of the manager. Figure 4, for example, gives an object point breakdown of a subsystem, by object type.

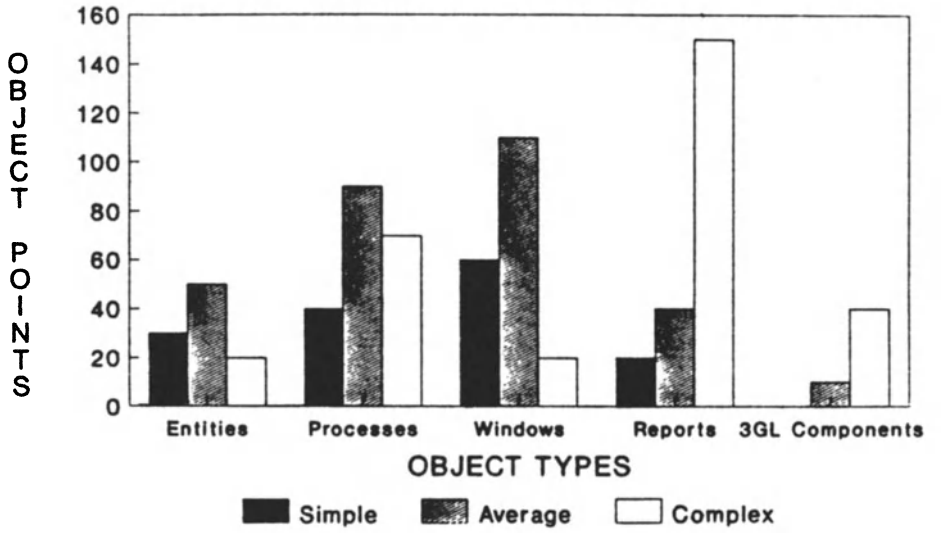


Figure 4: Object Point Breakdown by Object Type for Subsystem "Reorder"

4. SOFTWARE REUSE ANALYSIS

Software reuse is known to be a major source of productivity gains in software development. Based on claims that are often seen in the popular press, some organizations routinely expect reuse levels of 30 to 50%. But such high levels of reuse require an environment in which software reuse is supported from both a technical and a managerial standpoint; appropriate incentives for developers to reuse software; and a measurement program that provides a feedback mechanism to tell developers how much their efforts are paying off.

Object-based integrated CASE tools such as HPS provide the requisite technical support: they store software objects at a level of granularity which is far more conducive to reuse than traditional procedure-based software. They may also automate the mechanics of implementing reuse. HPS, for example, allows developers to reuse an object by simply adding a calling relationship to the repository. Measurement of reuse is also possible with CASE, especially when there is a

business value of software reuse that is occurring. This is captured by SRA in a metric called *reuse value*. Reuse value is computed by translating the standard cost of the effort that would have been required, had the software objects that were reused been built from scratch. This is a highly useful metric because it helps managers to determine whether reuse pays off in development cost reductions.

Management of Software Reuse

SRA may be used to track software reuse within a given project, but such analysis generally comes after the fact. The main power of the tool is guiding the organization's long-term software reuse efforts. Figure 5, shown below, tracks our two sites' software reuse efforts over a comparable 20-month period.

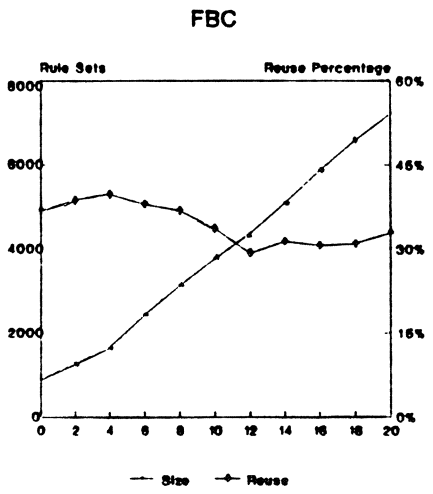


Figure 5a: Reuse and Repository Growth

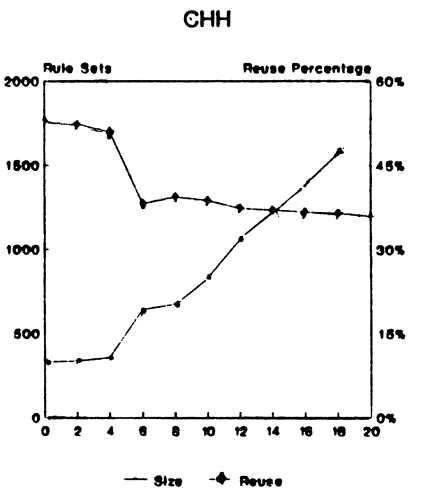


Figure 5b: Reuse and Repository Growth

The striking result is that while repository sizes grew steadily throughout the observation period, reuse levels almost immediately stabilized around the 30% level. Further use of SRA enabled us to analyze these results. Since HPS maintains a repository history of each object, it was possible to determine who created and who

repository that stores objects and their calling relationships. Such measurement is a prerequisite for accurate cost estimation. After all, software project cost estimation isn't going to be very reliable if we don't know whether to expect 30% reuse or 70% reuse of pre-existing software in a new system!

The Software Reuse Analyzer (SRA)

The repository-based architecture of HPS makes it practical, as we saw in the description of the function point analyzer, to query the repository to determine the extent of software reuse. This is accomplished through SRA, the *Software Reuse Analyzer*, which begins its analysis by creating a list of objects belonging to a given system. (This part of the analyzer's software was first developed for FPA and then, appropriately enough, reused in SRA.) We can also query the repository to determine how many times each object has been reused. Finally, the CASE tool maintains an object history which allows us to distinguish between internal reuse and external reuse. *Internal reuse* occurs when an object is created for a given application system and then used multiple times within that system. *External reuse*, on the other hand, occurs when the object being reused was initially created for a different application. The latter is more difficult to achieve, but is also more profitable.

SRA was built to deliver a number of useful managerial metrics. For example, it reports on two related metrics that offer an at-a-glance picture of the extent of reuse in an application: *new object percent*, the percentage of an application that had to be custom-programmed, and *reuse percent*, the percentage of the application constructed from reused objects. As we pointed out above, managers will further wish to distinguish between internal and external reuse percentages, to gauge how effectively developers are leveraging the existing repository. SRA can decompose reuse percent into *internal reuse percent* and *external reuse percent*.

A second important piece of information that managers will want is the

reused each object, and for which applications. The results were enlightening, as suggested by Figures 6 and 7 below.

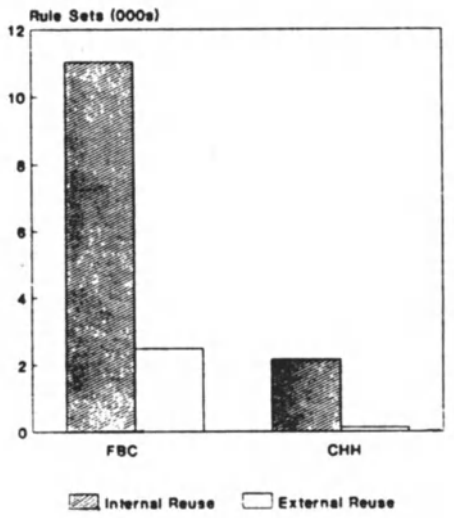


Figure 6: Internal and External Reuse

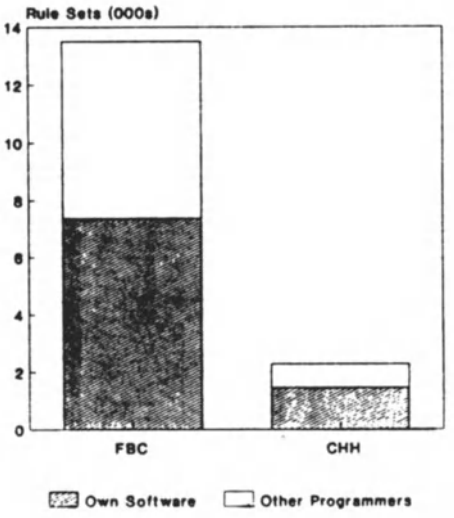


Figure 7: Reuse of Own Software

There was a strong and expected bias towards internal reuse. Developers preferred to get as much leverage as possible from the objects of the system under construction, rather than search the other systems in the repository for reuse candidates. What was not expected, however, was that most of the instances of external reuse consisted of programmers reusing objects that they themselves had previously created. In other words, little effort was being made to search for reusable objects. If developers personally knew of a reuse candidate, they used it; if not, it was simpler to write a new object than to search the repository for a reusable one. This went a long way towards explaining why the growth in repository size, and hence in reuse opportunities, was not resulting in growth in the reuse rates.

Figure 8 graphs the reuse levels of individual programmers against their overall output.

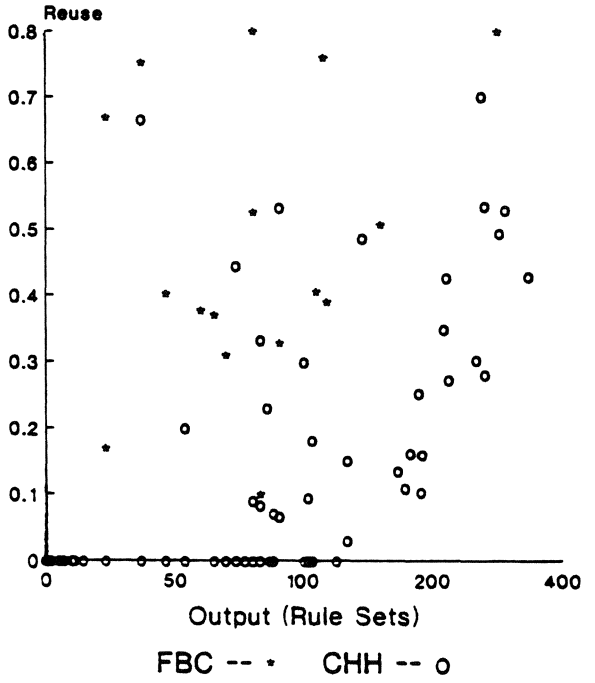


Figure 8: Reuse and Programmer Output

What the wide variation in programmer performance tends to obscure is the impact of the extremes. Software reuse analysis revealed that over 50% of the programmers at the research sites contributed no reuse whatsoever. On the other hand, the top 5% were responsible for over 20% of the objects in the repository and over 50% of the reuse. Only a few programmers were taking advantage of the substantial productivity gains that software reuse offered.

5. OBJECT REUSE CLASSIFICATION

We conclude our overview of the STRESS toolset with a discussion of a tool which is still in the research and development phase: the *Object Reuse Classification Analyzer*. Whereas software reuse analysis measures the level of reuse achieved, object reuse classification enables us to determine the repository's reuse potential, and

supports developers in achieving that potential.

Repository Search for Reuse

We observed that one of the striking results of the software reuse analysis was the propensity of developers to reuse familiar objects, rather than to search extensively for unfamiliar, but possibly superior, reuse candidates. A mature repository may easily contain tens of thousands of software objects, only a fraction of which will be familiar to any one programmer or analyst. A developer who focuses on familiar objects (and most of the reuse we observed involved developers reusing software they themselves had created) will miss many software reuse opportunities.

Our interviews with HPS programmers confirmed what others have already discovered: search is difficult. The high productivity of an integrated CASE environment such as HPS makes it faster to write a new object from scratch than to search an enormous repository for an existing object which is a close enough fit. (This is as true for the analyst trying to design a system which will take advantage of software reuse as it is for the programmer trying to find an object to perform a specific task.) A more extended search may pay long-run dividends, in the form of reduced maintenance costs, but this is an argument which programmers and project managers have rarely found convincing in the face of immediate schedule pressures. So, if we want developers to take advantage of the untapped reuse potential, we have to provide automated search support.

Figure 9 illustrates the conceptual foundation of object classification analysis. We can think of the repository as consisting of a large number of objects within a "search space", with similar objects being closer together and dissimilar objects being further apart. The classification scheme is used to produce a similarity metric that determines the "distance" between repository objects. We can then give the system a description of the object we need, and ask for a short list of repository objects which are 'close' enough to the described object to be reuse candidates.

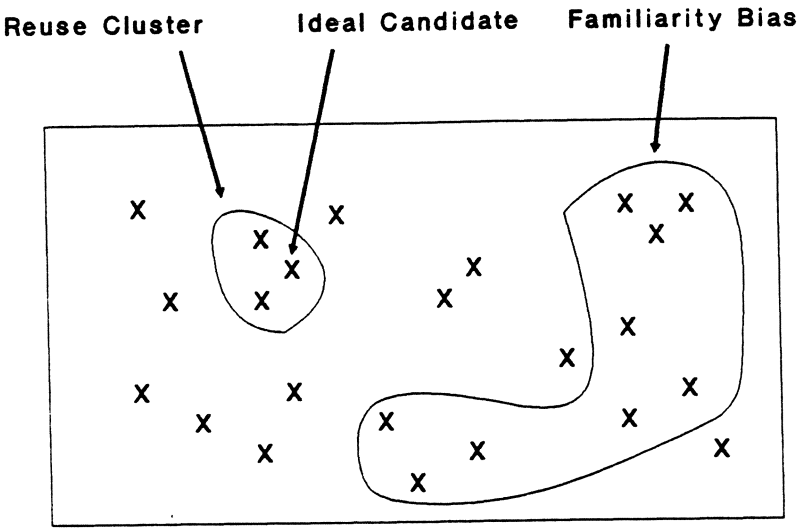


Figure 9: Reuse Clustering

Object Reuse Classification Analyzer (ORCA)

ORCA, the Object Reuse Classification Analyzer, has three functions: classification support, development support, and repository evaluation support.

1) **Classification support.** The classification scheme used by ORCA is an extension of Prieto-Diaz's faceted classification schema [4]. In such a schema, an object is classified along a number of dimensions — the facets -- and two objects may be 'close' to each other with respect to one or more facets. Figure 10, for example, illustrates a four-facet classification of a needed software module, and of two candidates for reuse. In this example, the functional similarities between the first component and the target object make it a better candidate than the second component, even though the second component was written for the target setting.

	NEEDED	COMPONENT 1	COMPONENT 2
Function	cross-validate	cross-validate	purge
Application	personnel	inventory	payroll
Objects	dates	dates	records
Setting	bank branch	dept store	bank branch

Figure 10: A Four-Facet Classification of Software Entities

ORCA supports *multiple classification criteria*. Multiple sets of facets may be defined, instead of a single criterion, or a single set of facets, with different classifications applying to different object types and to different stages of software development. Each set presents a criterion by which to analyze the repository. This allows, for example, for the case of two objects which would be judged to be far apart during business design, but might be closely related during technical design. The technical functionality may be similar, even when the business application functionality appears to be unrelated. Based on this multi-faceted classification schema, we can compute a quantitative metric to determine functional similarity between objects.

As the classification example suggested, an object classification scheme will use a combination of technical characteristics (e.g., object type, application system) and functional characteristics (e.g., purpose of module). The technical characteristics can be determined automatically, from information in the repository. For other facets, the developer can be prompted to choose from a list of options. The specific functionality-related classes and options may differ from one site to another, in which case the schema must be customized on the basis of interviews with software developers.

2) **Development Support.** The key design principle is to reduce the developer's involvement in the screening stage to a minimum — to let the analyzer worry about finding the potential needles in the haystack — and to

provide a short enough list of candidates that the developer will be able to give serious consideration to each. The search for reuse candidates takes place in two stages:

* Stage 1, *screening*, involves the purposeful evaluation of a large set of object reuse candidates from the entire repository to produce a short list of near matches for further investigation.

* Stage 2, *identification*, enables the developer to examine individual objects more closely to determine whether there is a match in terms of the required functionality.

When systems design is done well, it is very likely that a by-product of the effort will be a repository representation which can be matched to other existing repository objects at the time that technical design is completed. What remains is to ensure that there is a mechanism in place that enables a designer to test his design against the existing repository to determine what functionality might be reused as is, what might be adapted from very similar objects, and what needs to be built from scratch.

3) **Repository Evaluation Support.** Besides helping developers to find and inspect candidates for reuse, ORCA may also be used to classify objects and evaluate the repository as a whole. On the one hand, it can be used to identify redundancy -- unexploited reuse opportunities. A mapping of the repository will identify "reuse clusters", sets of objects which are similar in functionality, and can probably be consolidated into a smaller number of objects. Figure 11 illustrates the results of such consolidation.

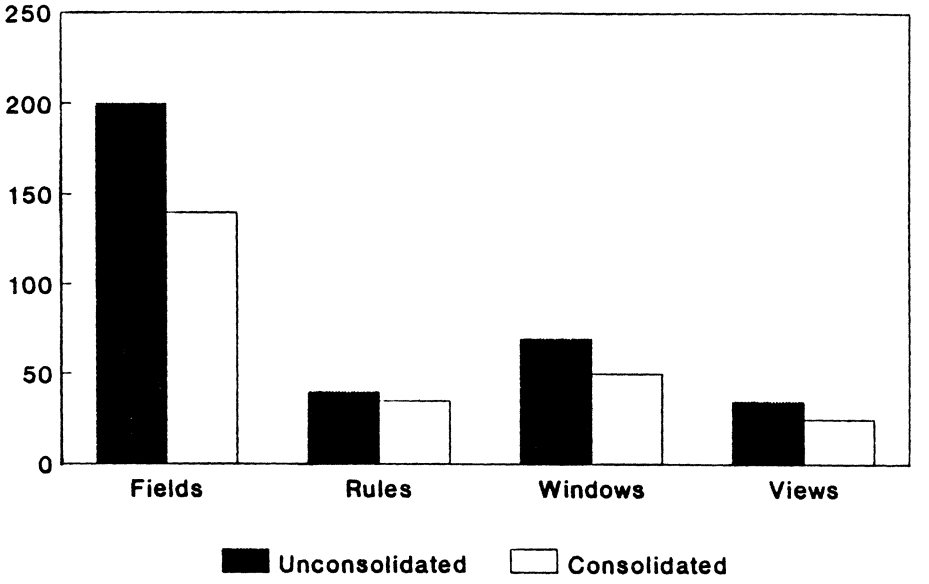


Figure 11: Consolidating a Repository by Detecting Reuse Clusters

Note that different types of repository objects are likely to benefit differently from such consolidation. On the other hand, such a mapping may identify gaps in the repository -- application areas in which developers will be less able to rely upon reuse support from the rest of the repository.

6. TOOLS TO MANAGE THE REPOSITORY: A RESEARCH AGENDA

Our current research efforts on repository object management software tools are focused on four primary tasks:

(1) *Implementation of the tools to support measurement will support longitudinal analysis of productivity and reuse.* With the help of Seer Technologies, we are working to install the Function Point Analyzer and Software Reuse Analyzer at a number of firms, in the U.S., Europe and Asia. This will enable us to carry out a large-scale longitudinal study of development productivity and software reuse, that expands upon our pilot studies in these

domains. As Kemerer [3] has pointed out, one of the most challenging problems facing software development managers is how to speed the move down the CASE development learning curve. In the absence of empirical results that estimate the learning curves that different firms have actually experienced, it will be difficult to provide much guidance as to the factors that enhance or inhibit firms to achieve better performance more rapidly.

(2) *We plan to further examine opportunities to extend the capabilities of the repository evaluation software tools to support other kinds of analysis.* We have already done a significant amount of this work on an informal basis, through specially developed repository queries. These queries have enabled us to investigate aspects of the repository that help to explain the 30% technical cap on reuse that we observed in the early days of software development at The First Boston Corporation and at Carter Hawley Hale Information Services. They also allowed us to determine which developers reuse software objects the most, and what kinds of software objects are involved. The results of such analysis has provided senior management at the firms whose data we analyzed with a fresh perspective on their software development operations.

(3) *The object points concept requires further empirical research to validate it for use in multiple settings.* Additional field study work, with Seer Technologies and its clients, and with other CASE vendors and their clients, will enable us to apply and validate the object point metrics we have proposed for software cost estimation in repository object-based integrated CASE environments. This process will only be possible through the deployment and application of the Object Point Analyzer, OPAL. We expect that additional field study research will enable us to uncover the extent to which the object complexity weights may vary with different software development environments.

(4) *Additional conceptual and empirical research is required to support the completion of a full design document for object reuse classification.* There are two research challenges related to this portion of our agenda. We are currently performing a set of structured interviews with software developers who use HPS to identify unique classificatory facets. Meanwhile, we are working to construct the elements of the analysis method that, given a workable classification scheme, will enable software developers to identify potentially reusable objects.

In this article, we have attempted to give the reader an appreciation of the kinds of measures which it is practical to derive from an automated analysis of an integrated CASE system. STRESS, the Seer Technologies Repository Evaluation Software Suite, enhances the ability of managers to control repository-based software development. It also makes it practical for us, as researchers, to perform data-intensive empirical analyses of software development processes. Software reuse, as this paper suggests, is of particular interest in this environment.

REFERENCES

1. Banker, R. D., Kauffman, R. J., Wright, C., and Zweig, D. "Automating Reuse and Output Metrics in an Object-based Computer Aided Software Engineering Environment." *IEEE Transactions on Software Engineering*, forthcoming.

2. Banker, R. D., Kauffman, R. J., and Zweig, D. "Repository Evaluation of Software Reuse." *IEEE Transactions on Software Engineering*, forthcoming.

3. Kemerer, C. F. "How the Learning Curve Affects CASE Tool Adoption", *IEEE Software*, volume 9, number 3, May 1992, pp. 23-28.

4. Prieto-Diaz, R., and Freeman, P. "Classifying Software for Reusability." *IEEE Software*, January 1987, pp. 6-16.

We wish to thank Mark Baric, Gene Bedell, Vivek Wadhwa, Tom Lewis and Gig Graham, for access to managers, developers and data related to software development activities at Seer Technologies and its clients. We are also indebted to Len Ehrlich, Michael Oara and Tom Robben for assistance with various aspects of this research program. We offer special thanks to Charles Wright, currently on assignment with Seer Technologies in Switzerland; his contributions to the automated function point analysis and software reuse analysis facilities that are described in this paper helped to make the work possible.

METHODOLOGICAL ISSUES IN FUNCTIONAL ECONOMIC ANALYSIS

Thomas R. GULLEDGE
Edgar H. SIBLEY
George Mason University
Fairfax, VA 22030-4444 USA

Ted K. YAMASHITA
Battelle Memorial Institute
2101 Wilson Boulevard, Suite 800
Arlington, VA 22201-3008 USA

I. PRELIMINARY REMARKS

Functional Economic Analysis (FEA) is a methodology for modeling the costs, benefits, and risks associated with alternative investment and management practices. FEA is the primary decision support methodology for Department of Defense (DoD) Business Reengineering (BRE) analyses performed under the Corporate Information Management (CIM) initiative. Since July 1991, functional managers have been required to prepare FEAs to support all investment decisions for automated information systems. This paper draws on several published FEA documents [5,12,35], and presents a discussion of a logical and consistent framework for performing FEAs.

The main subjects discussed here relate to process flow modeling, economic analysis, resource modeling, and risk analysis. We see physical process flow modeling as the central component of FEA. The process flow model is used to describe a business process, and it is impossible to measure improvements associated with alternative ways of doing business if the costs of the current way of doing business are unknown. We assert that costs, benefits, and risks must be related to the activities of the process flow model. The implication is that cost and resource data of any process flow activity due to be changed must be collected and modeled.

Since the DoD does not currently accumulate data by process activities, a complete change in cost analysis culture is being advocated. The DoD Comptroller's *Unit Cost* program is not the answer to this problem. While cost may be aggregated by

output measure, more detailed data are needed to support FEA. A proper analysis requires that the variable cost of the activities that go into the production of output be measured. We believe that most FEA researchers and practitioners agree that process flow modeling, combined with activity-based costing, is the proper way to approach FEA.

A theoretically correct FEA methodology should extend process flow modeling to include cost/resource modeling and risk analysis. While there are still many unresolved issues, we believe that the extension addresses some of the difficult problems associated with BRE and FEA. The cost estimator *assigns* costs, benefits, and risks to process flow activities, while the economic analyst *projects* these same factors over the planning horizon. However, since functional process flows are dynamic feedback control systems, modeling is a necessity. A theoretically correct approach must calculate resource[1] consumption as a function of workload, and as a second step, assign costs to the resources. Since resource cost is a time-dependent function of the quantity of resources consumed, the approach must permit the costing of the workload as the process flow and resource prices change over time. In addition, the approach must consider uncertainty in the resources consumed, the workload, cost, or any other model variable or parameter.

II. INTRODUCTION

A. Overall Strategy

We focus on FEA details that we consider important and in need of further elucidation. These details relate to cost definitions, cost accounting, cost modeling, and risk analysis. We believe that the basic issues discussed here must be resolved if activity accounting methodologies are to be successfully merged with policy analysis.[2]

---

[1] Resources are personnel, computers, floor-space, etc. The model is used to project the changes in resource consumption as workload varies. This is a crucial point, especially since the DoD workload is projected to decrease dramatically in the 1990s.

[2] The basic framework for policy analysis is presented in [29].

A methodology is an open set of procedures which provides the means for solving problems. It is not a *cook book* of procedures that can be applied as an algorithm to every situation. Every FEA is different, hence the methodology can only describe the general problem solving framework. A methodology should provide a strategy for problem solving in the absence of political or other restrictions; i.e., policy restrictions should be viewed as constraints on the methodology. As these constraints arise they should be analyzed and included, using the unconstrained methodology as a reference for benchmarking suboptimality.

There has been some pressure to mandate "tools" to help the functional manager perform FEAs. While tools force standardization of FEA output, we believe that such discussions are premature. We have concentrated our efforts on defining the methodology, and we will consider automation once the methodology is understood. However, we note that the described methodology is consistent with the required IDEF[3] process modeling presentation format.

B. What Is Functional Economic Analysis?

Functional Economic Analysis is a methodology for modeling the costs, benefits, and risks associated with alternative investment and management practices. The approach is *zero-based* in that it considers new ways of doing business, managing organizations, and investing in information technology. That is, the tasks and activities of each organizational unit as well as all supporting technology and constraining regulations are potential decision variables. While savings can be realized from investments in information technology, it is generally agreed that most efficiency gains and cost savings accrue from optimizing the organization's business processes.

Functional Economic Analysis is the primary decision support methodology for evaluating Business Reengineering (BRE) alternatives. The purpose of BRE is to change

---

[3] Since February, 1992, all DoD process models in support of Corporate Information Management initiatives have been required to be constructed using $IDEF_0$. IDEF was developed for the U.S. Air Force's Integrated Computer Integrated Manufacturing Program (ICAM) [39]. IDEF, the nested acronym, stands for the ICAM Definition approach to system study.

business processes and culture radically. This is accomplished by identifying and implementing new ways of doing business. The primary focus is on restructuring organizations and improving business methods and practices in accordance with customer demands. Thus technology, including information technology, plays a supporting role in the analysis process.

III. POLICY ANALYSIS ISSUES

Many of the FEA concepts are as old as the policy analysis discipline. The procedure has many of the characteristics of the seven step procedure described by Sage [32]:

1. problem definition,
2. value-system design,
3. system synthesis,
4. systems analysis and modeling,
5. optimization and alternative ranking,
6. decision making,
7. planning for action.

This seven step process is modified and extended for FEA by considering relevant issues in Cost Analysis (see, for example, [16]), Cost-Benefit Analysis (see, for example, [33]), Industrial Engineering, Activity-Based Costing (see, for example, [4]), Activity-Based Management (see, for example, [20]), Risk Analysis (see, for example, [13]), and nonlinear dynamic modeling (see, for example, [6]).

IV. PROCESS FLOW ISSUES

As previously noted, FEA is process oriented. A basic tool for analyzing business processes is the industrial engineering process flow chart, as described in the *Industrial Engineering Handbook* [21]. Process flow charts are an essential component of FEA. This statement may seem somewhat extreme, since economic analyses are routinely executed in the absence of process flow charts. However, FEA has a process

orientation; hence, the interactions among functional activities in the production of organizational outputs must be understood. The functional manager must see how the activities interact in order to determine cost changes, and the process flow chart is a proven method for identifying activities and pictorially describing how they interact.

There are process flow aggregation issues that need additional study, but the rule-of-thumb is that the process flow is presented at an aggregation level that provides cost visibility and managerial cost control. We believe that the process flow disaggregation decision is difficult and not well understood in the context of FEA. Additional research through additional applications is needed.

V. COST ISSUES

A. Basic Cost Analysis Framework

The assigning of cost to activities is the essence of FEA. Hence, an understanding of fundamental cost measurement and tracing issues is imperative for a proper understanding of FEA.[4] However, those who understand these issues know why economic analysis on large scale systems is so difficult, especially within the DoD environment.

We agree with the activity-based approach to FEA; however, we do not believe that activity-based costs are necessarily the only costs that should be considered within the context of FEA. The cost analysis methodology is dependent on the system under study as well as the decision time horizon. An analogous situation has been noted in manufacturing [1]. For example, activity-based methods have been used successfully in private industry to identify and monitor profitable product lines [10]. In many DoD applications, especially mission-critical applications, short-term capacity expansion may be much more important than profitability. For these scenarios, capacity and bottleneck costs may play a more important role in alternative selection than activity-based costs.

---

[4] The cost issues are well understood in Europe, especially in Germany. This history can be traced to the early work of Schmalenbach [34], with many of the relevant issues summarized in [38]. Other significant works are by Riebel [31], and a more mathematical treatment provided by Stöppler and Mattfeld [36].

However, for most DoD business process applications of FEA, activity-based cost modeling will be central to the effort.

Activities are "what" an organization does [28]. In more detail, activities are the things that organizational entities (e.g., managers, employees, etc.) do on a day-to-day basis. "Processing a report" or "inspecting a product" are examples of activities. "Tasks" are simply the steps taken in performing an activity. We prefer to think of an activity as a process that transforms inputs into outputs. The tasks are the essence of the transformation process.

As already noted, process flow modeling may be used to represent an organizational process pictorially. At one aggregation level, the process flow nodes could represent activities. An additional disaggregation would result in more detail, and the process flow nodes could be process tasks. Our current position is that it is probably not necessary to present cost at the task level, but an understanding of the tasks and their interactions is necessary in order to model cost at the activity level.

If activities are modified or eliminated, as in the generation of FEA alternatives, then an obvious question is: How does cost change? This is a difficult question to answer. It involves activity-based cost modeling, a concept that is understood in manufacturing settings [27] but has not received much attention in the organizational reengineering literature. Later we add emphasis to additional important modeling issues, but prior to the modeling discussion it is necessary to define cost.

B. What Is Cost?

This important question has no obvious answer. It is therefore the source of much confusion in the organizational reengineering literature. At the heart of the confusion are issues that describe the relationships between budgeted dollars and actual costs, and between financial and managerial accounting. These issues require careful consideration.

We adopt the fixed and variable cost definitions proposed by Kaplan and Atkinson [22]:

> *Fixed Costs* are those costs unaffected by variations in activity level in a given period. They are expected to be constant throughout the period, independent of the level of outputs produced or inputs consumed. Of course, whether a cost is fixed or variable will depend both on what factor is selected to predict cost variation and on the time period over which cost is expected to vary. Fixed costs can be thought of as short-run, perhaps annual, capacity costs; they represent the cost of resources required to perform a planned level of activities.
>
> *Variable Costs* fluctuate in response to changes in the underlying activity level of the firm or in response to change in the underlying scale of activity. Materials, some components of labor (both direct and supervisory), overhead, and marketing costs are typical examples of short-term variable costs. Some short-run variable costs, such as direct materials and direct labor costs, will increase proportionately with the level of output. Other short-term variable costs, such as the salaries of intermediate-level supervisory personnel, may vary in steps, such as when an extra shift is added or subtracted, or in some other disproportionate fashion, with the level of output.

Some would argue that these definitions are not sufficiently precise. For example, the neoclassical definition of cost implies optimality; i.e., the minimum cost of producing certain outputs with given input prices and technology [24]. This economic definition is precise, and any cost modeling approach should search for minimum cost, but none of the existing FEA approaches explicitly model cost as frontiers. We do not address these more theoretical issues in this paper.

It is important to note that the above definitions are different from the cost classifications most often used by cost accountants: direct and indirect costs. Fultz [17] defines *a cost objective as any function for which cost is accumulated*, and *a direct charge is one that is incurred for a specific cost objective*. Typical cost objectives are products, services, etc. As noted by Fultz (see page 9), both the Federal Procurement regulations and the Armed Service Procurement regulations provide a definition of indirect costs:

> An indirect or overhead cost is one which because of its incurrence for common or joint objectives is not readily subject to treatment as a direct cost. Minor direct cost items may be considered to be indirect costs for reasons of practicality. After direct costs have been determined and directly charged to contracts or other work as appropriate, indirect costs are those remaining to be allocated to several classes.

By our accepted cost definitions, direct costs are probably variable; although there may be parts of direct costs that are fixed. Indirect costs clearly contain large segments that are variable. These are the cost that accountants label as *semi-variable*. The challenge facing FEA practitioners is to analyze budget and *Unit Cost* data that are either total expenditures or classified into direct and indirect categories, and model the cost of processing activity workload as fixed and variable cost. The expansion of this concept permeates the remainder of this paper.

C. Financial and Managerial Accounting

The issues relating to accounting systems are critical to understanding the FEA process. These issues are discussed in some detail by Kaplan [23]. The issues were also discussed in [5], but it is important that the basic issues be reiterated.

Financial accounting data are compiled to satisfy the needs of the organization's external constituents; e.g., stockholders, taxing authorities, contract auditing agencies, etc. These data are not compiled with managerial decision making in mind. If financial accounting data are used to support organizational decisions, distortions are likely to occur.

The main point of this discussion is related to the cost definitional issues already discussed. If a manager makes a decision that alters any activity relevant to the production of organizational output, then cost should change. By definition this is variable cost. As noted, it is reasonable to assume that direct costs are variable. If direct costs comprise a large portion of total cost (i.e., 90% or more), then direct costs are probably a good approximation for variable costs.

An example of cost distortion was uncovered in a conversation that one of the authors had with an IBM executive at a recent DoD workshop on cost/performance measurement [8]. IBM produces a number of products at one of its manufacturing facilities. Using traditional cost accounting methods, some of these products are indicated as being more profitable than others. Two of these products were examined in detail. Product 1, a heavy consumer of computer resources in the production process,

was believed to be very profitable. Product 2, a light consumer of computer resources in the production process, was believed to be significantly less profitable.[5]

For accounting purposes, computing resources were treated as an indirect cost, and they were allocated to the products in proportion to the number of units produced. Note, computing costs are clearly variable as they are described in this example, but the allocation procedure does not treat them as variable. When management properly traced computing costs to the products that were incurring the costs, the distortion was discovered. The relative profitability of the two products was reversed; i.e., product 2 was really the more profitable product.

The distortions occur when indirect costs are a large proportion of total costs and semi-variable costs are significant. Under this scenario, if direct costs are used to approximate variable costs, the approximation is often poor since large portions of variable costs are not properly considered. In the early years of manufacturing, direct costs were a large percentage of total cost, hence financial accounting data could be used in managerial decision making with minimum distortions. However, the trend in defense industry indirect costs has been steadily increasing. In fact, in many industries indirect costs are as high as 50% of the total (see, for example the discussion in [25]).

The above discussion is important for many obvious reasons, and one less obvious reason. The DoD Comptroller's *Unit Cost* program reorganizes budget data into direct and indirect categories by output measure. This is a major limitation of using the unit cost data to support FEA development. The relevant data for FEA are variable costs by the activities that go into the production of an output measure.

VI. MODELING ISSUES

Modeling is probably the least understood of all of the FEA issues. Our experience has been that modeling means different things to different FEA practitioners. The purpose of this section is to clarify some of the issues relating to process, cost, and resource modeling. The intent is to relate the types of models to activity-based costing.

---

[5] A similar example is presented by Strassmann [37, p. 71]

A. Models and Process Flows

Giordano and Weir [18] note the following about systems and the modeler's problem:

> A *system* is an assemblage of objects joined in some regular interaction or interdependence. The modeler is interested in understanding how a particular system works, what causes changes in the system, and the sensitivity of the system to certain changes. He or she is also interested in predicting what changes might occur and when they occur.

The above scenario is identical to that faced by the functional manager with a BRE task.

Business reengineering is a complex process. Complex organizations are characterized by many inputs and outputs. The process of transforming inputs into outputs may span a number of interconnected activities with many causal relationships. Presumably, as work flows through the activities the inputs are applied in such a way that value-added is generated; the objective of FEA and BRE is to understand and measure this value-added.

As previously mentioned, it is possible to provide a pictorial representation of a particular business process with a process flow chart. The process flow chart may provide some insight into how inputs interact with outputs at the activity level, but FEA requires quantification of costs, benefits, and risks. The process flow chart does not provide this information. To quantify costs, benefits, and risks for the function; the decision maker must quantify costs, benefits, and risks at the activity level.

This requirement should not be trivialized. The function is complex, data are scarce, and resource requirements and availability change over time. It is a rare decision maker who can report activity cost or resource consumption at a particular point in time, much less project these numbers one year into the future. The relationships among the activities must be analyzed in a *systems analysis* context, since it is impossible for the functional manager to analyze the complex system relationships subjectively. It would appear that some form of simplification is required.

A model is a selective abstraction of reality. The purpose of a model is to simplify and explain, and there are many types of models. For example, the physical process flow chart is a pictorial model of a function. It simplifies the system in that it

stresses the importance of critical activities and interrelationships, while suppressing those activities that are relatively unimportant.[6]

IDEF is another type of pictorial model. IDEF provides a logical representation of how activities interact; however, IDEF stops short of predicting costs, resources, and benefits at the activity level. Modifications of IDEF allow the functional manager to attach (as an attribute) costs, benefits, and risks at the activity level, but one still has to predict costs, benefits, and risks at the activity level. This idea is discussed below in detail.

B. Quantitative Decision Models

Since the functional manager must quantify costs, benefits, and risks, a quantitative decision model is required. Each decision model contains decision variables, and the process of making a decision is equivalent to assigning numbers to the decision variables. The task of the modeler is to understand how these decision variables relate to costs, benefits, risks, or any other important criteria.

If the relationships among the decision variables and the decision criteria can be represented with a system of equations, then one has a mathematical model. For most complex systems (e.g., business process flows), it may be possible to specify equations but impossible to solve them in closed form; hence simulation will probably be required. In order to predict the costs, benefits, and risks associated with a complex system, a simulation model may be the only practical solution.[7]

---

[6] One could argue that all activities are important from the point of view of producing final output. Perhaps they are; however, that is not the point of this discussion. Many variables impact a particular decision; so many that it is impossible for the manager to understand how they relate to one another. The basic idea in modeling is to identify those variables that have the most impact on the decision criteria while judiciously suppressing those that have little impact. In this sense some variables and/or activities may be more important than others.

[7] This is not the appropriate forum for a lengthy discussion of simulation models, nor model classification. A good discussion of the relevant issues is provided in [15]. Chapter 1 is particularly relevant for this discussion.

C. Simulation Models, Process Flows, and FEA

Here we combine the apparent disparate parts and discuss the logic of a simulation-based approach. We also explain how this approach extends the activity-based costing work initiated by the Corps of Engineers [11].

The baseline process model describes the current systemic relationships, and it also predicts the costs, benefits, and risks associated with maintaining the baseline in future years. The key word is prediction; that is, the FEA assigns costs, benefits, and risks to activities over time. FEA is dynamic.

The above assignment or *tracing* problem could be solved in a number of ways. For example, the functional manager could subjectively assign cost to a particular activity and in many applications a subjective assignment may be sufficient. Consider the case of a data processing center that provides services for several business processes. The center may not accumulate cost by process; i.e., the center has a budget that does not provide visibility by process. The manager may have to determine subjectively the proportion of data processing services to be traced to each process. However, this is not the same as guessing. The assignment is based on expert judgment, group consensus, Delphi, or whatever subjective methodology is deemed appropriate. If the subjectively estimated data is used as model input, the estimate can be refined over time as the analyst observes how sensitive the model output is to the estimated input.

Additional accuracy and believability can be obtained by collecting data on the uncertain activities. This is the logic of the activity-based costing (ABC) approach. The functional manager collects data on costs, benefits, and risks at the activity level. This data, collected mainly by interview with historical support, requires a new way of thinking by the functional manager, since the DoD does not currently collect data at the activity level. However, if the functional manager is successful, the benefits are obvious. Much insight is gained about baseline operations and visibility into nonvalue-added activities is obtained.

The ABC approach is superior to subjective estimation, but it stops short of analyzing bottlenecks, feedback, or other sensitive cost drivers within the process flow.

This point becomes particularly important when alternative process flows are analyzed. It is impossible to collect historical data on processes that do not currently exist, yet the FEA methodology requires that costs, benefits, and risks be assigned to alternative activities as well as the baseline.

The Byrnes et al. [5] approach solves this problem by using a dynamic cost/resource[8] simulation model. Hence, there is a third way to provide data at the activity level - as the generated output of a simulation model. The use of a model provides insight into baseline process flow relationships and allows the analyst to perform sensitivity analyses on critical parameters. In addition, it is almost impossible to quantify risks without a model.

For analyzing alternative process flow configurations, a model is a necessity. The ABC approach advocates *zeroing out* nonvalue-added activities, and perhaps assigning costs to new investment initiatives. To compute the average total cost of processing the baseline workload with the alternative process flow configuration, the functional manager multiplies the average unit cost by the average workload and sums the estimated costs across all activities in the process flow.

The modeling approach extends the ABC approach in the following way. If some current activities are eliminated, and some new activities are added; the costs and risk may be altered for every activity in the process flow. That is, cost accumulation is nonlinear, and new technologies applied to activities may cause bottlenecks at other activities. Since the alternative system does not exist, the only method for analyzing alternative system bottlenecks and feedback is to use a resource model. Hence, a resource model is useful for predicting costs and risks for the activities in the alternative process flow.

In addition, the simulation model can be used to project costs, benefits, and risks over time. It is possible to apply ABC methods over time by linking a sequence of

---

[8] In this document we shall use the term *resource model* to mean any simulation model used to predict resources, cost, schedule, or other relevant economic variables. We note that the simulation model could be discrete or continuous (see [15], chapter 1). The advantages and disadvantages of each type of model are discussed in the section on risk analysis.

spread sheets; perhaps one for each year in the planning horizon. However, the basic problem still exists - FEA process flow charts are a representation of a nonlinear dynamic system, and they should be analyzed as such.

We are not criticizing the ABC approach. In fact, some ABC work must be completed before a resource simulation model can be constructed and for many functions a simulation model may not be necessary. For example, consider a scenario where the baseline process model, augmented with activity-based costs, indicates that savings potential is low. Why then incur the additional cost of constructing a model? BRE is a methodology for identifying *major*[9] savings. The extraction of minor savings is really a Total Quality Management issue.

D. Conclusions

The FEA methodology is characterized by three major tenets:

1. process flow analysis,
2. activity-based costing,
3. resource modeling that includes risk analysis.

It is our belief that resource models are a necessity. The estimator assigns cost to activities while the analyst models cost at the activity level. The addition of modeling extends FEA from the realm of cost estimating to the realm of economic analysis.

VII. COST MAPPING ISSUES

A. What Are the Issues?

On the surface, the issues seem simple. The output of a FEA must map in two directions. First, the functional manager must map the FEA results to an agreed-upon standard cost breakdown structure. This is defined by the Institute for Defense Analyses (IDA) with its Functional Economic Analysis Model.[10]

---

[9] We make no attempt to define what is meant by major savings. In their BRE efforts, James Martin and Company uses the term *10x savings*. The assignment of a dollar value to the term *major* is obviously a DoD policy decision.

[10] The IDA model provides a standard format for displaying FEA output. The IDA model summarizes FEA data in a spreadsheet for a presentation of the risk adjusted discounted

The second mapping that must be addressed by the functional manager involves DoD budgets. Projected savings (if realized) imply budget reductions. Since the FEA identifies where savings occur and their dollar amount, the manager must map the FEA output to the organization's budget. These mappings are not well understood, and additional research is needed here.

B. Mapping to the IDA Model

The IDA cost structure is very general since it was designed for application to any function. Cost is segregated into direct and indirect categories, and it is the task of the functional manager to display the function's cost according to the IDA structure.

FEA output summarizes cost into fixed and variable (as opposed to direct and indirect) categories. A portion of indirect costs is variable; hence, costs must be reclassified prior to mapping FEA output to the IDA model. The most popular methodology for addressing this problem is what accountants call the *account classification procedure*[11] (see, for example, [22], pp. 94-97).

The account classification procedure requires that the cost analyst classify each category of cost as either fixed or variable. In effect, the analyst subjectively maps the direct and indirect costs onto the fixed and variable costs; however, this mapping can be extremely difficult.

In fact the problem is more difficult than the standard account classification procedure, because the mapping is in the reverse direction. Instead of determining what parts of direct and indirect costs are fixed and variable, the analyst must determine how to reconstruct direct and indirect costs from the fixed and variable FEA output. It is impossible to understand the reverse mapping without first understanding the forward mapping.

One final issue needs to be discussed with respect to this mapping problem. If the analyst estimates cost at the activity level using the same cost structure as the IDA

---

cash flow of the projected savings associated with each alternative. The IDA model also permits risk analysis over the elements of the generic cost structure.

[11] Other methods for estimating fixed and variable costs are discussed in [2].

model, then there is no mapping. Total cost is found by summing the cost elements across the activities. Unfortunately, as previously discussed, as activities are altered in the formation of FEA alternatives, distortions may occur. The nature of the distortion cannot be stated because it varies from function to function. If a function has high indirect costs, the distortion may be considerable; but if most costs are direct, the distortion may be slight.

C. Mapping to DoD Budgets

The standard DoD budget does not display costs by direct and indirect nor variable and fixed categories. The new DoD *Unit Cost* budget presents the unit costs allocated to predefined output measures, but this budget is still not the answer to the FEA mapping problem. There are three major issues that relate to Unit Costs.

First, the Unit Cost boundary does not always agree with the boundary of the function under study. When the boundaries do not coincide, the functional manager either ignores the Unit Cost boundary and collects supplemental data or abides by the Unit Cost boundary and the FEA is constrained.

Second, the Unit Cost outputs do not always agree with the outputs defined by the functional manager. In many ways this is uninteresting, since a modeling approach to FEA does not directly use cost by output. The relevant costs for modeling are the costs associated with the activities that produce functional output.

Third, the mapping problems still exist. The FEA output should project costs in variable and fixed categories. The IDA model accepts direct and indirect inputs, but the Unit Cost structure is not the same as the IDA cost structure.

In our opinion, Unit Cost is an important budgeting tool, but it complicates (as opposed to simplifying) the FEA process. However, if functional managers are forced to use Unit Costs in constructing their FEAs, we believe that a mapping is possible. This is certainly an area where additional research is needed.

VIII. RISK ANALYSIS ISSUES

A. Definition of Risk

In the decision analysis literature a clear distinction is made between risk and uncertainty. In any decision problem, three assumption can be made regarding the true state of nature, ϕ. First, if ϕ is known with certainty, the decision is straightforward. This situation is called *decision making under certainty*. Second, if there is total ignorance about the value of ϕ, we are said to be making a *decision under uncertainty*. The final assumption characterizes most management decisions. If we can assess a probability distribution for various values of ϕ, the situation is classified as *decision making under risk*.

The above taxonomy is useful and precise, but unfortunately it often confuses managers when they are trying to understand and manage risk. For the typical manager, risk results from *uncertainty* about the value of a decision or process variable; e.g., cost or schedule. The manager may be able to characterize this uncertainty by providing an interval estimate or even by specifying a probability distribution. The confusion is caused by the unfortunate use of the word, *uncertainty*. As used by the risk manager, it does not imply ignorance of the state of nature. Henceforth, we define risk as uncertainty in the value of a decision variable or a parameter. If the decision variable is cost, then we are describing cost risk. If the uncertainty is associated with evolving a new technology to provide a greater level of performance, then we are describing technical risk.

This approach to risk classification appears to be consistent with the *risk facets* advocated by the Defense Systems Management College [13]. DSMC identifies five facets of risk that are necessary to segment and manage the cost, schedule, and performance issues faced on a project:

1. Technical (Performance Related),
2. Supportability (Performance Related),
3. Programmatic (Environment Related),
4. Cost,
5. Schedule.

A similar classification could be designed for FEA, but the primary risk facets appear to be technical and cost.

For the definitions of *Risk Analysis* and *Risk Management*, we borrow from Charette [7]:

> The process of identification, estimation, and evaluation of risk is called risk analysis. Risk analysis is used to identify potential problem areas, quantify risks associated with these problems, and generate alternative choices of actions that can be taken to reduce risk.
>
> The planned control of risk and monitoring the success of control mechanisms is termed risk management. Risk management is involved with making a decision about the risk(s) after it has been analyzed. Four elemental tools are needed to effectively apply risk management. They are (a) standards against which performance can be measured; (b) information to monitor actual performance; (c) authority to make required adjustments and corrections when they are needed; and (d) competence to implement the best solution from available alternatives.

The above definitions are consistent with the Business Reengineering process. Risk analysis is associated with the FEA component of BRE, and risk management is mainly associated with the implementation and monitoring component of BRE.

B. Primary Risk Analysis Issues

There are at least three issues that need additional study in the area of risk analysis:

1. Where should the risk analysis be performed? Should it be performed at the disaggregated activity level, or should it be performed at the more aggregate 'total cost' level?
2. How should risk be computed? Are analytical methods appropriate, or is Monte-Carlo simulation required?
3. What is the appropriate risk probability distribution?

Each issue is now discussed. But to motivate the discussion, several items need to be clarified. By our definition, risk is synonymous with probability. We assume that the analyst provides sufficient information about the estimated random variable, so that an

interval estimate may be developed. This implies that the analyst has to provide the following information for each resource or cost estimate:

1. A measure of central tendency; e.g., the expected value of the estimate,
2. An estimate of the minimum and maximum of the random variable,
3. A measure of dispersion; e.g., the variance or standard deviation.

The total risk associated with a baseline or alternative cost model requires a combining of the individual risk distributions to estimate the risk distribution of total cost.

Given the above, the classification of risk into different types is less meaningful. For example, it has been noted that there are two types of risk that should be considered: a) implementation risk associated with implementing new technology through a selected alternative, and b) uncertainty in cost estimation. For our purposes the two types of risk are treated the same as long as the risk probability distribution can be specified. If it is impossible to specify the risk probability distribution, the uncertain variable is not considered in FEA.

C. Where Should the Risk Analysis Be Performed?

The obvious answer is that the analyst should disaggregate the process to the appropriate level to understand the risk. If this means that risk must be modeled at the task level, then so be it. If the aggregation level is too high, critical high-risk activities may not receive appropriate attention in sensitivity analyses. However, there is an equal danger in disaggregating to low levels. Resource managers often have difficulty in estimating the required input data (i.e., mean, min, max, and variance) at low levels. Hence, much analyst judgment must go into the selection of the appropriate level.

The general consensus is that the risk analysis should be done at the activity level, but the appropriate activity disaggregation level is problem specific. A good rule-of-thumb is to only disaggregate on activities that are controllable; i.e., if the decision maker has no control over an activity, model the risk of that activity at the highest possible level.

A second rule-of-thumb involves the identification of the most critical sensitivity parameters. For example, in many FEAs, the wage rate is a parameter that

affects almost every activity. In deciding how far to disaggregate, one would certainly want to disaggregate to a level that provides visibility of the sensitive parameters; e.g., workload bottlenecks, error rates, high cost inputs, etc.

Of course, the analyst must consider data availability, data believability, and interview time required. These issues must be considered relative to savings potential in deciding how far to disaggregate the process flow activities. If savings potential is low, a good rule-of-thumb is to model risk at a high aggregation level. For those areas of the process flow model that management controls and believes that significant savings are possible, disaggregation would seem to be appropriate. The implication of this discussion is that many of the modeling issues relating to risk analysis require judgment in modeling. There is definitely an *art* component in what we are advocating.

D. How Should Risk Be Computed?

It is our hypothesis that the method used to compute risk is not a major issue. We do however agree that a model of the organization's process flow is necessary to provide meaningful resource and cost projections and that this model will have to be some type of simulation model; i.e, the process flow model is a nonlinear feedback-control system. For any simulation model, either discrete or continuous, the risk analysis output is a technicality. The difficult part of the analysis is understanding the organization's process flows.

Once the simulation model is constructed the following three-step procedure could be used for risk analysis:

1. Estimate the partial derivative of the measure of effectiveness with respect to each parameter. Rank the parameters according to sensitivity; i.e., the most sensitive parameter is ranked at the highest level.
2. For those parameters that are most sensitive, try to assess their uncertainty.
3. Those parameters that have high sensitivity and high uncertainty are the primary candidates for risk analysis.

Again, this three-step procedure re-enforces the idea that there is an *art* component to risk analysis.

There are issues that relate to the choice of a simulation methodology. Discrete simulation provides much flexibility in modeling process flows, and the risk analysis is a natural by-product of the simulation. However discrete simulation can require significant computer resources, especially in performing sensitivity analyses. The number of runs required for a statistically significant estimate of the mean response is large, and the number of runs required for risk analysis may be very large.

Continuous simulation requires little computing, but it provides less flexibility, and the risk analysis is not a natural by-product of the simulation. The risk analysis is approximate, and the error in approximation is an issue that has not been fully explored.

E. What Is the Appropriate Risk Probability Distribution?

Given the nature of FEA data, this does not seem to be an issue of overriding importance. However, the issue can be of extreme importance in simulation modeling. For example, in discrete simulation, it is easy to generate pseudo-random deviates from any distribution for which the inverse cumulative distribution function can be represented in closed form. For other distributions, the process can be more difficult as well as time consuming. The amount of computational effort required is also problem specific; i.e., a process flow network with 100 activity nodes will require more effort than an aggregate cost structure with 15 cost categories.

The Byrnes et al. [5] approach avoids the discrete simulation problems by using an approximation to the sum of beta distributions.[12] This approximation assumes that sums of beta distributions and products of beta distributions may be approximated as beta distributions. Note, that this is not the same as saying that sums or products of beta variates are beta distributed. The wording is crucial, since 'fitting' a beta distribution to a resulting sum of random variables is not the same as requiring that the sum of beta variates have a beta distribution.

The approach is clearly an approximation. For the problems examined to date, the approximation appears to yield satisfactory results, but more work is needed in

---

[12] This approximation approach was developed, coded, and tested by Henry A. Neimeier of The MITRE Corporation.

analyzing its accuracy. However, the beta distribution is extremely flexible in that it provides a good fit to most classical probability distributions. One major exception is the log-normal distribution, which requires a transformation prior to fitting the beta distribution.

Of course, a major advantage of using the approximation is that discrete simulation is no longer required to perform the risk or sensitivity analyses. For the beta risk distribution assumption, the analyst can analytically estimate total system risk by tracking four parameters at the activity level: the mean, the maximum, the minimum, and the standard deviation. The primary disadvantage is that the properties of the approximation are open research questions.

F. Conclusions

Much progress has been made in understanding risk at the activity level, but additional work is needed. The relevant research issues relate to the appropriateness of the beta distribution and the accuracy of the risk approximation. Other general research issues relate to discrete versus continuous simulations of the process flow models. If discrete simulation is selected, the most important research issues are relate to convergence and statistical significance.

IX. RELATIONSHIP TO THE ABBREVIATED METHODOLOGY

The abbreviated approach[13] is a results oriented methodology that is designed to force a decision with respect to a particular issue; e.g., a migration decision to establish the *CIM Functional Baseline*. The procedure is often discussed as a specific application of FEA.

The main strength of the procedure is that it forces a quick decision; i.e., it *jump-starts* the CIM process and minimizes "analysis paralysis." The main weakness of the procedure is that it is based on limited analyses of business functions or processes. In short, the abbreviated methodology is results oriented, and it may lead to suboptimal re-

---

[13] This approach has been called the abbreviated FEA by others. We avoid this terminology because the approach does not have the characteristics associated with FEA; i.e., process flow analysis, activity-based costing, and resource modeling.

engineering implementations in the long-run. It should be noted, however, that this issue should be of little concern in the definition of a FEA methodology. The abbreviated procedure is a separate type of analysis, and as long as the functional manager understands its purpose, strengths, and limitations, then it could be carefully used to support specific decisions. However, we believe that the CIM migration decision should be contained within BRE alternatives. Hence, as BRE policy is defined and as BRE management teams gain experience in performing FEAs, we hope that the abbreviated procedure is abandoned.

The suboptimality of the abbreviated procedure can be easily understood within the context of the CIM migration decision. Assume n potential migration alternatives and a target alternative selected by the FEA component of a BRE study. The optimal migration alternative is the intermediate state that minimizes the cost of achieving the BRE target. That is, the migration decision and the reengineering decision represent an optimal decision set, hence they should be selected so that they are consistent with Bellman's principle of optimality [3], as described in [26]:

> An optimal set of decisions has the property that whatever the first decision is, the remaining decisions must be optimal with respect to the outcome which results from the first decision.

This principle is intuitive when considered within the context of the migration decision.

Suppose candidate system # 1 is selected as the migration system, and the CIM functional baseline is established, based on system # 1. A BRE study is then initiated, and after analysis a BRE alternative is selected. Assume, at this point, it is noted that the implementation of the BRE alternative would have been less costly and/or more efficient if system # 2 were selected as the migration system; i.e., the decision of selecting system # 1 violates the principle of optimality.

The main point of the discussion is as follows. In the absence of constraints (political or otherwise), the migration system should be considered in the BRE analysis. However, the decision process is constrained. If the constraint is that a quick decision must be made, and if the functional manager is willing to accept the possibility that the quick decision may be suboptimal in the long-run, then the abbreviated procedure may be

appropriate. We see the abbreviated procedure as a constraint on the *optimal* FEA methodology, and it should be presented as such.

More technically, the constraint precludes n-1 arcs in a network representation of the migration process. This eliminates n-1 migration states from being considered when defining the BRE alternatives, leading to a potential suboptimal global decision. Denardo ([14], chapter 2) provides a simple mathematical presentation of the problem using directed networks.

We understand the intent of the abbreviated procedure, but we think that more work is needed to define the circumstances under which it would be appropriate.

X. RELATIONSHIP TO INFORMATION ENGINEERING

We see Information Engineering (IE) as being primarily associated with the design, development, and operation of information systems. Our view of the proper role for information engineering parallels that presented by Richmond [30]. In general the IE process has an information system orientation, as opposed to a functional orientation. Since FEA is the primary analysis tool to support organizational reengineering, FEA is primarily associated with organizational strategy/planning; a primary input into the IE process. We acknowledge that the boundaries are blurred; hence, some aspects of FEA could overlap some components of information strategy planning and business area analysis.

In summary, FEA is the primary economic analysis methodology to support organizational reengineering. The organizational reengineering analysis supports the organizational strategy/planning process. Once the organizational reengineering alternative is selected, information engineering is intimately involved in the implementation of the selected alternative.

XI. CONCLUSIONS

This paper discussed issues that need further clarification in the evolving DoD FEA methodology. While process flow analysis and activity-based costing are generally

accepted procedures, cost and resource simulation modeling has not been widely accepted by the functional manager.

We have discussed issues related to simulation modeling, and we have emphasized the activity risk analysis aspects of modeling. The paper suggests that simulation modeling (including risk analysis) is a primary FEA component, and if modeling is ignored there is potential for cost, benefit, and risk estimation errors. This is especially true in organizational processes that are characterized by feedback and resource reallocations.

XII. REFERENCES

[1] Bakke, Nils Arne and Roland Hellberg. *Relevance Lost? A Critical Discussion of Different Cost Accounting Principles in Connection With Decision Making for Both Short and Long Term Production Scheduling*, International Journal of Production Economics, Vol. 24 (1991), 1-18.

[2] Balut, Stephen J., Thomas P. Frazier, and James Bui. *Estimating Fixed and Variable Costs of Airframe Manufacturers*, P-2401. Alexandria, Virginia: Institute for Defense Analyses, 1991.

[3] Bellman, Richard. *Dynamic Programming*. Princeton: Princeton University Press, 1957.

[4] Brimson, James A. *Activity Accounting*. New York: John Wiley, 1991.

[5] Byrnes, Patricia, Anthony Hartman, Robert Judson, Henry Neimeier, and Joseph Platenkamp. *A Functional Economic Analysis Reference Methodology*, MTR 91W00198. McLean, Virginia: The MITRE Corporation, 1992.

[6] Cellier, F.E. *Continuous System Modeling*. Heidelberg: Springer-Verlag, 1991.

[7] Charette, Robert N. *Software Engineering Risk Analysis and Management*. New York: McGraw-Hill, 1989.

[8] Cheslow, Richard C. and J. Richard Nelson. *The Executive Workshop on Cost/Performance Measurement, Volume I: Executive Summary*, P-2321. Alexandria, Virginia: Institute for Defense Analyses, 1989.

[9] Cloos, John J. and James D. McCullough. *New Accounting Systems and Their Effects on DoD Cost Estimation*, P-2343. Alexandria, Virginia: Institute for Defense Analyses, 1989.

[10] Cooper, Robin and Robert S. Kaplan. *Profit Priorities from Activity-Based Costing*, Harvard Business Review (May-June, 1991), 130-135.

[11] *Cost-Based Activity Modeling Project Results*. Proposal #91-20 Presented to the Information Technology Policy Board, September 4, 1991.

[12] D. Appleton Company. *Corporate Information Management Process Improvement Methodology for DoD Functional Managers*. Fairfax, Virginia: D. Appleton Company, 1992.

[13] Defense Systems Management College. *Risk Management: Concepts and Guidance*. Fort Belvoir, Virginia: DSMC, 1989.

[14] Denardo, Eric V. *Dynamic Programming: Models and Applications*. Englewood Cliffs: Prentice-Hall, 1982.

[15] Emshoff, James R. and Roger L. Sisson. *Design and Use of Computer Simulation Models*. New York: Macmillam, 1970.

[16] Fisher, Gene H. *Cost Considerations in Systems Analysis*. New York: American Elsevier, 1970.

[17] Fultz, Jack F. Overhead: What It Is and How It Works. Cambridge, Massachusetts: Abt Books, 1980.

[18] Giordano, Frank R. and Maurice D. Weir. *A First Course in Mathematical Modeling*. Monterey, California: Brooks/Cole, 1985.

[19] Hammer, Michael. *Reengineering Work: Don't Automate, Obliterate*, Harvard Business Review, (July-August, 1990), 104-112.

[20] Johnson, H. Thomas. *Activity-Based Management: Past, Present, and Future*, The Engineering Economist, Vol. 36 (1991), 219-238.

[21] Kadota, Takeji. *Charting Techniques*. In Gavriel Salvendy, Editor, The Handbook of Industrial Engineering. New York: John Wiley, 1982.

[22] Kaplan, Robert S. and Anthony A. Atkinson. *Advanced Management Accounting*. Englewood Cliffs: Prentice-Hall, 1989.

[23] Kaplan, Robert S. *New Systems for Measurement and Control*, The Engineering Economist, Vol. 36 (1991), 201-218.

[24] Lovell, C.A. Knox and Peter Schmidt. *A Comparison of Alternative Approaches to the Measurement of Productive Efficiency*. In A. Dogramaci and R. Färe, Editors, Applications of Modern Production Theory: Efficiency and Productivity. Boston: Kluwer Academic Publishers, 1988.

[25] McCullough, James D. and Stephen J. Balut. *Defense Industry Indirect Costs: Trends, 1973-1982*, P-1909. Alexandria, Virginia: Institute for Defense Analyses, 1986.

[26] Nemhauser, George L. *Introduction to Dynamic Programming*. New York: John Wiley, 1966.

[27] O'Guin, Michael C. *Activity-Based Costing: Unlocking Our Competitive Edge*, Manufacturing Systems, Vol. 8, No. 12 (1990), 35-43.

[28] Pryor, Tom. *Activity-Based Management for a Competitive Advantage*, Mimeo Lecture Notes, 1991.

[29] Quade, E.S. *Analysis for Public Decisions*, Third Edition. New York: North-Holland, 1989.

[30] Richmond, Ken. *Information Engineering Methodology: A Tool for Competitive Advantage*, Telematics and Informatics, Vol. 8 (1991), 41-47.

[31] Riebel, Paul. *Einzelkosten- und Deckungsbeitragsrechnung*, Grundfragen e. markt.- u. entscheidungsorientierten Unternehmensrechnung, 5. Auflage. Wiesbaden: Gabler Verlag, 1985.

[32] Sage, Andrew P. *Introduction to Systems Engineering Methodology and Applications*. In Andrew P. Sage, Editor, IEEE Press, Systems Engineering: Methodology and Applications. New York: John Wiley, 1977.

[33] Sassone, P. and W. Schaffer. *Cost-Benefit Analysis: A Handbook*. New York: Academic Press, 1978.

[34] Schmalenbach, Eugen. *Buchführung und Kalkulation im Fabrikgeschäft*. Leipzig: Gloeckner, 1928.

[35] SRA Corporation. *Corporate Information Management Functional Economic Analysis Guidebook*. Arlington, Virginia: SRA Corporation, 1993.

[36] Stöppler, S. and D.C. Mattfeld. *Kosten- und Deckungsbeitragsflussanalyse in Allgemeinen Bezugsobjektnetzwerken*, Paper Presented at the International Conference on Operations Research, Vienna, Austria, 1990.

[37] Strassmann, Paul A. *The Business Value of Computers: An Executive's Guide.* New Canaan, Connecticut: The Information Economics Press, 1990.

[38] Streitferdt, Lothar. *Cost Management.* In E. Grochla and E. Gaugler, Managing Editors, Handbook of German Business Management. Stuttgart: C.E. Poeschel Verlag, 1990.

[39] U.S Air Force Systems Command. *Integrated Computer-Aided Manufacturing (ICAM) Function Modeling (IDEF0).* USAFSC: Wright-Patterson AFB, Ohio, 1981.

Using IDEF0 in Functional Economic Analysis

Minder Chen*
Edgar H. Sibley*+

Department of Decision Sciences and MIS*
Department of Information and Software Systems Engineering+
George Mason University
Fairfax, VA 22030 USA

I. INTRODUCTION

The use of the IDEF (a system definition methodology) has been mandated by the DOD's Director of Defense Information for documenting the process and data models of an FEA (Functional Economic Analysis) and for all Business Process Improvement Programs. IDEF stands for ICAM definition language. This methodology derives, in its present form, from the Integrated Computer-Aided Manufacturing (ICAM) program, where a standard specification methodology was needed for describing manufacturing processes and data; a family of methods has been developed under the ICAM program. Particularly, IDEF0 and IDEF1x are applicable in the conducting FEA study.

A very brief review of the FEA process is given first. Then, because there have been questions about the use of IDEF *versus* certain other methods and automated tools, we attempt to answer the following questions:

- What is the IDEF methodology?
- What is the relative value of such a methodology and what are the tools currently available to support them?
- What is the difference between IDEF and the many Information Systems (IS) methods and their supporting CASE (Computer Aided System Engineering) tools?

II. The Business Reengineering Process

The terms Business Case Analysis (BCA) is used interchangeably with the term Functional Economic Analysis (FEA). In DOD, functional managers have to conduct an FEA to justify and defend investment projects on their information systems. They have to take into consideration factors such as total ***cost*** of the investment decision, the amount of ***risk*** associated with the decision, and the ***benefit*** of the decision toward business goals. FEA is a method for analyzing and evaluating management practices, alternative process improvements, and investments [18]. The focus of the FEA study is to model and analyze the business activities of an organization and its supporting infrastructure.

Obviously, not the least problem in conducting FEA is in determining the scope or contour of the business being analyzed: Does it encompass all business or only certain parts? The scope of FEA study deals with DOD defined functional areas (those under the control of a Functional Manager), using them as a departure point for an initial cut at scoping the area or unit under consideration.

Once a set of specific units or offices of a function area has been identified, we have to ask the following two refined questions:

- What interactions are there between this portion of the business and others? and
- How can the parts be isolated so that the boundaries are not confusing to the analysts and the allocation of the savings is not disputed by the various business areas?

The overall FEA process consists of five steps:

1. **Create an initial baseline business process model.** An initial business process model is created by interviews or having users and functional area managers participate in a facilitated business process modeling workshop. Then the baseline business process model is created with the help of functional area users, managers, systems analysts, cost analysts, and other experts. A pictorial process model is developed to help visualize the results and understand the analysis.

2. **Determine activity-based performance and cost data.** Cost and performance data are determined for the initial process model and used as a baseline. A high-level data model for the functional area may also be developed at this stage to facilitate the redesign of the business process models. The performance data include the volume and frequency of inputs, mechanisms, and primary outputs of each of activities in the process model. The consumption of inputs and the utilization of mechanism can be translated into cost data for each activity. Cost elements of all organizational units should be allocated to each activity and then be aggregated to derive the total cost of each activity. Using one primary output of the process, we can also calculate the unit cost for each activity, i.e., dividing the total cost for each activity by the volume of the primary output of the process.

3. **Develop alternative business process models.** The project team evaluates the baseline business process model to identify opportunities for streamlining and redesigning processes. Activities that are not contributing to the added-value of the process and consume a lot of resources are potential candidate for consolidation, simplification, or elimination. A set of criteria, such as generalized coupling and cohesion, can be used to evaluate the baseline and alternatives.

4. **Evaluate alternative businesses process models.** Alternative business process models are evaluated based on cost and performance data, such as the investments compared with the baseline. Performance data, such as volume and frequency of inputs and outputs as well as response time of processes is used to build a simulation model to analyze the dynamic behavior of the alternatives. Costs are also derived.

5. **Present and select one of the alternatives.** A final set of alternatives is presented to top management based on the standard cost structure used in the organization. DOD uses a cost structure that is defined in IDA Template. A recommended alternative is identified. Top management will select one alternative.

Once a FEA study has been the approved, the selected business process redesign should be implemented. The implementation plan and strategies should be carefully formulated and the information systems that have been identified to support the redesigned process should also be designed and built accordingly.

III. The IDEF Family of Methods

The IDEF is a family of methods to support the modeling of complex systems. The term "system" used in the field of systems modeling means: a usually

large complex of interconnected parts with an organized array of individuals and parts forming and working as a unit. One of the criteria for choosing a system modeling method is primarily that it be appropriate to model the type of system under consideration. It must also be:

- Easy to learn and use by its intended users.
- Capture information of the target system in a structured way so that the information can be further analyzed by computer programs.
- Support decomposition of the system into a hierarchical organization (e.g., a decomposition) of components with different levels of abstraction.
- Have software tools to support the documentation, storage, and analysis of the models in applying the methodology.

A. IDEF History and Its Evolution

The first of the IDEF methods was called IDEF0 [12]. This is a functional modeling method, somewhat similar to a conventional procedural and control flow chart with aspects of a general data flow diagram. A second method (IDEF1) is based on Chen's Entity-Relationship-Attribute model for conceptual data base design. IDEF1 Extended (IDEF1x) was developed under the Integrated Information Support System (IISS) project, under the ICAM Program, as a data modeling technique to describe common data model subsystem. The primary contract is General Electric Company. The final report was delivered in November 1985. This has since been extended, mainly by D. Appleton Company (DACOM). Other members of the IDEF family include a simulation language (IDEF2) developed mainly by Pritsker & Associates. It can be used to simulate a design, thereby representing time varying behavior of resources in a system. There is an on-going effort to extend the IDEF suite of methods [13]. Other members of the IDEF family (IDEF7 to IDEF14) are in review by the IDEF User Group. In this report, the discussion is focused on IDEF0 and IDEF1x and their applications in FEA.

A model is expected to answer questions about the requirements and initial design decisions of a system. A model should have a clearly defined boundary. A model has a viewpoint and purpose. A system's IDEF0 model consists of a set of

interrelated IDEF0 diagrams, texts, and tables.

Activities and things are two object types that can be modeled in IDEF0. Activities are functions and processes performed by the system. Things include data (e.g., a blueprint, customer orders) and non-data objects (e.g., parts, machine tools, an automated system). The interactions between things and activities are called connections. IDEF0 diagrams have boxes and arrows. Boxes represent activities and they are placed in a diagram as a staircase to indicate the dominance of one activity over the other. Arrow-headed lines, also called arrows, are connections that represent interactions of things with activities. Arrows connect boxes together to indicate the constraints of one activity imposed on another. The basic construct of an IDEF0 diagram is depicted in Figure 1. An example of a highest level IDEF0 diagram, i.e., A-0 diagram, is shown in Figure 2.

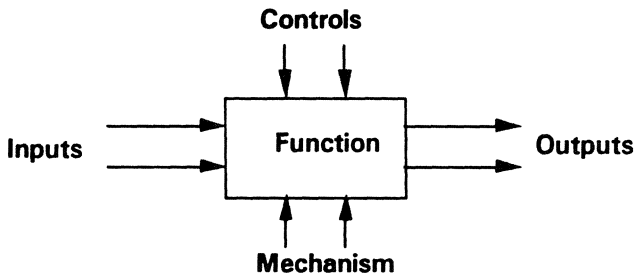


Figure 1. The Basic Constructs of an IDEF0 Diagram

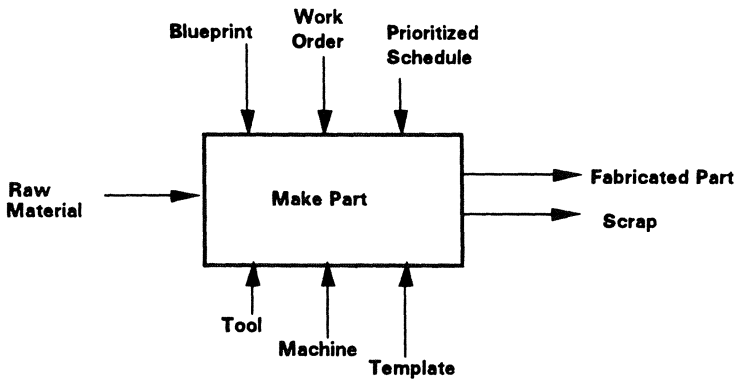


Figure 2. An Example of an A-0 Diagram

A IDEF0 diagram also contains the following information:

- Title
- Author(s) of the diagram
- Project Name
- Date of creation
- Date of last revision
- Diagram status: Working, Draft, Recommend, or Publication
- A list of readers and dates when the readers read the diagram
- Context: Position of the diagram in the parent diagram, e.g., *None* for A-0 diagram, *Top* for first-level decomposition diagram, or  to show the relative position of the parent process in the decomposition diagram that it belongs to.
- Node Number: A unique identifier of a box in an IDEF0 diagram. It consists of project abbreviation, "/", Node index number of the parent node, and box number of the node in the diagram. For examples: FEA/A1, FEA/A13, FEA132.
- Notes: Substantive comments about the diagram, numbered from 1 to 12.
- C-Number: An IDEF0 diagram's unique identifier (e.g., MC 002) that consists of an author's initials (or unique identifier) and a sequence number. The C-number of a previous version of the diagram is enclosed in parentheses to provide link to prior work.

A system can be modeled as a set of interrelated IDEF0 diagrams, texts, and glossary, called IDEF0 model. The highest-level diagram, called A-0 diagram, represents the whole system. An A-0 diagram has only one box and is annotated by PURPOSE and VIEWPOINT. The PURPOSE of a system's IDEF model is to answer the reason why the model was developed. A system can be described from several VIEWPOINTs (i.e., that of a person or an organizational unit). An IDEF0 model should, however, be developed from only one particular viewpoint.

The role of the arrows is important in IDEF0 diagram:

- Arrows represent how boxes influence or constrain each other. A box can send its outputs to other functions for further transformation, provide controls that govern what other functions must perform, or serve as mchanisms for carrying out other functions.

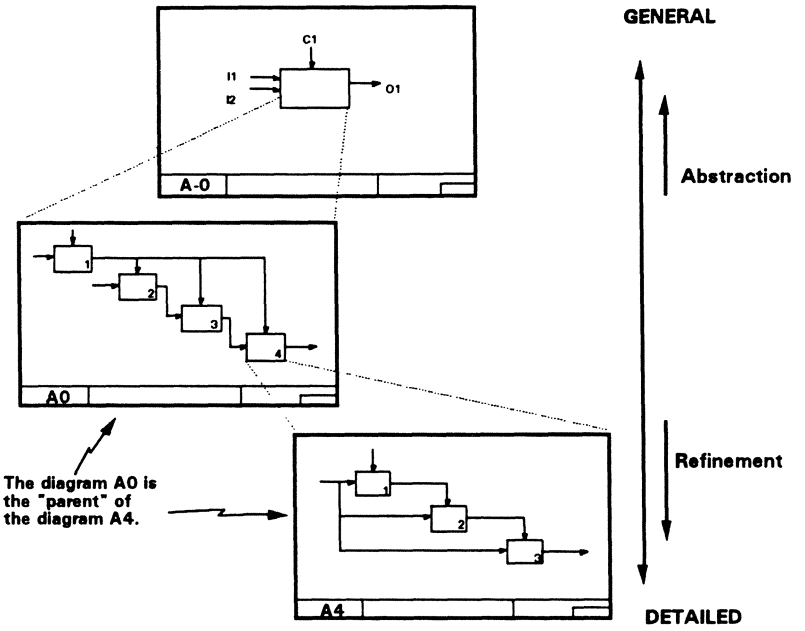


Figure 3. The Hierarchical Structure of an IDEF0 Model

- The side of the box at which an arrow enters or leaves determines the role of an arrow (i.e., a thing) related to the box. These roles include input (left), control (top), output (right), and mechanism (bottom) They are referred to as ICOM in IDEF0 diagrams. An IDEF function and its related ICOMs can be interpreted as:

***"Inputs* are transformed by the *function* into *outputs* according to *controls*, using *mechanisms*."**

- Input: Describe resources or data that are needed to perform the function and are transformed by the function into outputs.
- Control: Describe the conditions, rules, procedures, or circumstances that govern the execution of the function. An arrow is a control unless it obviously serves only as input. Each function should have at least one control arrow. Most of controls are in the form of data.
- Output: Data or non-data objects that are transformed by the function and leave the boundary of the process are called outputs. An output of a function can be used as inputs, controls, or mechanism of other functions.
- Mechanism: Define the actors (i.e., supporting mechanisms) that carry out the function. A mechanism may be a person, an organizational unit, a physical device, a computer program, etc.
- Arrow branches: Branches that are not labeled are assumed to contain all the things carried by the arrow before the branch. Branches that are

labeled could contain some or all of the things carried by the arrow before the branch.

- Arrow joins: Branches that are not labeled are assumed to contain all the things carried by the arrow after the join. Branches that are labeled could contain some or all of the things carried by the arrow after the join.

IDEF0 requires 3 to 6 boxes in any one diagram except A-0 diagram. IDEF0 tends to be used as a physical model because you can describe the implementation *mechanisms* of systems functions. However, if we only use and focus on the inputs, outputs, and controls of the systems functions, IDEF0 can be used to provide a logical (i.e., essential) model of the system. IDEF0 diagram can be used to show material flows.

An IDEF0 model consists of a set of hierarchically organized IDEF0 diagrams. A function can be decomposed into a detailed IDEF0 diagram. The hierarchical structure of an IDEF0 is shown in Figure 4. The arrows connected to a function are carried down to the function's decomposed IDEF0 diagram to maintain consistency and completeness.

IV. Tools Supporting IDEF and FEA

There are several tools that could be used to support the application of IDEF and FEA. Many techniques and methods used in an FEA study are complex and a large amount of structured information is generated in the definition process. Selecting appropriate tools and integrating them to support these techniques and methods are critical to the successful implementation of an FEA.

A. Computer-Aided Diagramming Tools for IDEF0

Currently there are several COTS that support IDEF0 and IDEF1x. Some basic information of these IDEF tools is listed in Table 1. There are several approaches that an organization can take in providing computer-aided diagramming tools to support the use of IDEF methods. All of these may be alternatives:

1. Select the best tool from the marketplace and make it as a standard tool to

be used in FEA.

2. Recommend a list of tools as long as they support existing IDEF methods.
3. Recommend use a CASE shell to generate diagramming tools that support IDEF to meet the specific requirements in applying IDEF for FEA.

Table 1. A Sample of Tools for IDEF0 and IDEF1x

Product	Methodology	Vendor	Platforms
IDEFine-0	IDEF0	Wizdom Systems, Inc.	Sun, PC, Apollo, VAX
Design/IDEF	IDEF0 IDEF1 IDEF1x	Meta Software Co.	Mac, MS Windows, Unix
AI0 AI1 AI1x	IDEF0 IDEF1 IDEF1x	Knowledge Based Systems, Inc.	PC
IDEF/Leverage	IDEF0 IDEF1x	D. Appleton Co.	PC, VAX
ERWin	IDEF1x	Logic Works	MS Windows

One advantage of the latter alternative is that CASE shells provide a flexible approach to supporting existing and future methods required in the FEA process. CASE shells are software tools that allow users to define a system modeling methods and then generate a run-time environment to support the specification of a target system using the method. There are several commercially available CASE shells in the marketplace. It will be a worthwhile effort to evaluate these CASE shells. The benefit of the CASE shell is that, it allows the customization and evolution of the method. Existing IDEF tools might need appropriate extensions to support the modeling and analysis. Using CASE shell to generate tools to support IDEF will allow continuous improvement of IDEF and smooth integration of IDEF with tools needed in the FEA and in the downstream systems development process. Examples of such CASE shells are: Intersolv's XL/Customizer, VSF's Virtual Software Factory, and CADWare's Foundry.

The work of certain standards committees, such as the IDEF Users Group and the Electronic Industries Associate's CDIF (CASE Data Interchange Format) Technical Committee may soon make it possible to pass designs (i.e., IDEF models) from one vendor's tool to another.

B. Tools That Support the FEA

There are several tools that could be used in the FEA process. Using IDEF for business process modeling under the context of FEA is a collaborative effort among functional area users, managers, cost analysts, information engineers, and business engineers. It is a change process because the re-engineering business process may dramatically change how work will be done in the future. Team-oriented techniques, such as Joint Application Design, may be used to encourage proper participation of all parties involved in the process to ensure all the requirements and concerns are surfaced and addressed. One of the critical success factors in using JAD is the skill of the facilitator (i.e., session leader). However, skillful facilitators are hard to find. Emerging collaboration technologies can be used to assist facilitate business process modeling workshop. Collaboration technology can be used to provide anonymity, equal participation, and complete documentation of workshop outcomes. It is not going to replace a good facilitator, but it can be used to improve the effectiveness of business process modeling.

Conducting FEA could be a very political process. *Group facilitation techniques* and *collaboration technologies* can be used to assist to support business process modeling meetings. Group support systems, such as GroupSystems and VisionQuest can be used as a front-end requirements elicitation tool that captures initial specifications of a business process [3]. These specifications can be transferred to IDEF tools for the construction of formal IDEF models. We have developed a Collaboration Technology Laboratory at George Mason University, that can be used to demonstrate this approach.

The IDA Template can be used to present the final cost analysis of various alternative business process models with the baseline model. Cost data collected for

various alternatives and for the baseline can be presented in different formats (e.g., graphics and tables) based on a specific cost breakdown structure. Two major cost items are: Operations Costs and Management & Support Cost. Each of them is broken into down into the major life cycle phases and expense types. Cost data of baseline and alternatives are entered into a cost Data Sheet based risk and the detailed cost breakdown structure over a six-year period. For alternatives, each estimated cost item has the high and low values. It is a way to express the risk or uncertainty involved in the estimation. A Risk Adjusted Discounted Cash Flow (RADCF) analysis will be performed on each alternative. The results are presented in graphics and tables.

V. Comparison of IDEF Methodology with Dataflow, ERA, and Other Techniques

A system can be modeled from multiple viewpoints. However, in IDEF0 you

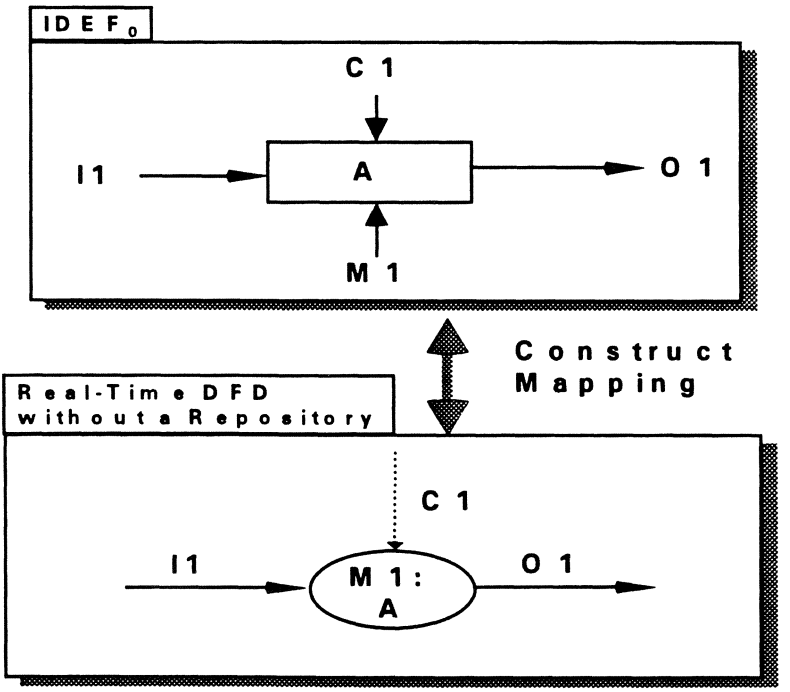


Figure 4. Mapping Between IDEF0 to Real-time DFD without a Repository

are only allowed to model a system from a single viewpoint. A system model consists

of a set of interrelated diagrams, texts, and tables. Data Flow Diagrams (DFD) can be used to describe both the logical and physical aspects of an existing (AS-IS) system or a new (TO-BE) system, just as can IDEF0. When a DFD is used to specify the physical aspect of a system, the implementation mechanism can be amended to the processes in the diagram as depicted in Figure 4. Mechanism can also be captured in the corresponding entry of a process object in the repository as shown in Figure 5.

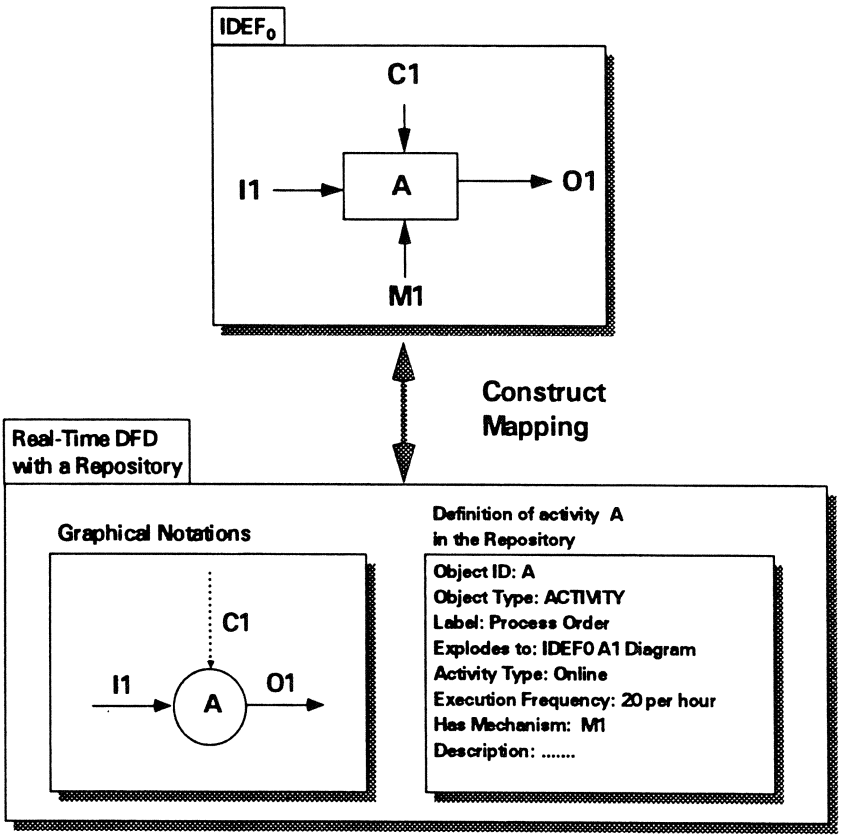


Figure 5. Mapping Between IDEF0 and Real-time DFD with a Repository

The weaknesses of IDEF0 include:

1. The external entities are not explicitly represented, and thus the interaction of the system with external entities is only represented by the inputs, outputs, controls,

and mechanism of the system.

2. Boxes in IDEF0 are forced to be placed as a *staircase* pattern to show the dominance among them. However, there are situations where several functions may have the same dominance. The author of an IDEF0 diagram still has to lay out these functions as if they were different in their dominance.
3. The glossary of IDEF0 is a simple description of objects used in the IDEF0 diagram. It can not represent additional attributes of the objects or relationships of the objects with other objects. A more powerful repository is needed to integrate IDEF0 diagram with IDEF1x diagram, and with other modeling tools used in the FEA process.
4. INPUT, OUTPUT, MECHANISM, and CONTROL are terms used to describe how things (i.e., data and materials) are related to a function or activity under study. An output of a function can be used as an input, control, or mechanism of another function. Controls are usually in the form of information. It is not very easy for an author of an IDEF0 model to determine the role of a thing that interacts with the function.

Currently, there is no *repository* concept in IDEF techniques. Objects (e.g., functions, inputs, and outputs) are defined in a glossary that capture only limited attributes of the objects. Structured relationships among objects are not captured. The integration of IDEF suite of methods can be achieved by using a repository system that can support the data integration among tools that support IDEF methods and related tools for FEA. Currently, IDEF does not provide a predefined set of attributes and relationships for further descriptions of the characteristics of IDEF objects and their relationships to other objects. Due to this deficiency, if IDEF has to be used in FEA, a meta-model of the IDEF has to be defined to serve as a foundation for implementing repository-based IDEF tools [5]. Important information required for FEA should be defined in the meta-model. The following are some suggestions:

1. Cost and resource utilization data could be associated with MECHANISM. These include volume, frequency, unit cost, etc.
2. Performance data could be associated with INPUTS, OUTPUTS, and PROCESSES. Examples of performance data include response time, throughput, etc.
3. Things can be categorized into data and non-data objects such that we can link data objects that are used as inputs, controls, or outputs of functions in IDEF0 with IDEF1x diagrams or with other definitions of data structures.

VI. Using IDEF for Business Process Modeling

In discussing ISDOS project in retrospect, one author has said that there was major industrial apathy due to a lack of the use of PSL/PSA generated documents for implement the target system [16] -- thus the users saw it only as a complicated documentation device. If the only purpose of using IDEF in FEA is to document the business process model, the potential benefits of using it will be limited. We believe that IDEF or its alternative techniques should be used for continuous business improvement and for the downstream systems development activities. If a technique or tool is used only for documentation, no one is likely to have a vested interest in keeping the business model up-to-date. The effort spent in building the business process models may be wasted.

The business expertise embedded in business models is a valuable asset that should be exploited by functional area users and managers with the assistance of business analyst. Once the model has been developed, it could be used by functional area users and managers to:

1. support continuous business improvement based on the performance criteria established in the model,
2. assist the business reengineering process in making structural changes,
3. navigate and explore the model in order to understand the goals and objectives of the organization,
4. guide the development of IS models to ensure that the IS are aligned with the business objectives,
5. perpetuate a common mental model of the business, and
6. form the basis for delivering information in the business context.

Business process models should be integrated with information systems models because many critical business functions are supported by IS. An integrated model should be used not only for the design/redesign of business and its IS, but the users to deliver the information [2, 4].

VII. Conclusions

Training is one of the important factor in successfully implementing a methodology. It is particularly critical in implementing IDEF as a front end modeling technique to support business engineering/reengineering and to facilitate FEA because these processes are driven by end users and managers. They are usually not familiar with the systematic approach in analyzing and designing business systems and information systems. However, their involvement is politically smart. It is the only way to capture the information and knowledge about the businesses and their operations. By tapping into front-line users' and managers' expertise and creativity about how they may improve their work processes, we will have a better chance to introduce dramatically improvements in the processes. Training on IDEF in the FEA should be given to both business analysts and users/managers of various agencies. *Real-time training* should be given so that people with the training can apply the techniques, methods, and tools in their jobs.

The FEA is a change process. It changes the way we think about information systems investment and development processes and the users' and managers' roles in this process. It is a change of mind set instead of a change of notations. Therefore, it is more important to educate the agencies about the underlying principles Corporate Information Management (CIM) first [1, 17], then familiar them with FEA processes. IDEF and other related techniques, methods, and tools are important mechanisms to carry out the vision of CIM, but they should be introduced after the CIM principles and FEA process has been fully understood.

VIII. References

[1] Brewin, B., "CIM: Corporate Information Management," white paper, Federal Computer Week, September 1991, pp. 1-15.

[2] Chen, M., Liou, Y. I., and Weber, E. S., "Developing Intelligent Organizations: A Context-Based Approach to Individual and Organizational Effectiveness," to appear in *Organizational Computing.*

[3] Chen, M. and Nunamaker, J. F., Jr., "The Architecture and Design of a

Collaborative Environment for Systems Definition," *Data Base,* Vol. 22, No. 1/2, Winter/Spring 1991, pp. 22-29.

[4] Chen, M., Nunamaker, J. F., and Weber, E. S., "The Use of Integrated Organization and Information Systems Models in Building and Delivering Business Application Systems," *IEEE Transactions on Knowledge and Data Engineering,* Vol. 1, No. 3, 1989, pp. 406-409.

[5] Chen, M. and Sibley, E. H., "Using a CASE Based Repository for Systems Integration," *The 24th Annual Hawaii International Conference on System Sciences*, January 8-11, 1991.

[6] *Cost-Based Activity Modeling Project Results,* Proposal #91-20, presented to ITSB, September 4, 1991.

[7] Gulden, G. K. and Reck, R. H., "Combining Quality and Reengineering for Operational Superiority," *Indications,* September/October, 1991, Vol. 8, No. 1, pp. 1-9.

[8] Hammer, M., "Reengineering Work: Don't Automated, Obliterate," *Harvard Business Review*, July/August 1990, pp. 104-112.

[9] Hammer, M., "Why We Need Both Continuous and Discontinuous Improvement," *Indications,* September/October, 1991, Vol. 8, No. 1, pp. 6-7.

[10] IDA, *Functional Economic Analysis of DoD Functions,* Users Manual, Institute of Defense Analysis, 1991.

[11] *Integrated Computer-Aided Manufacturing (ICAM) Architecture Part II, Vol. IV - Function Modeling Manual (IDEF0)*

[12] Marca, D. A. and McGowan, C. L., *SADT: Structured Analysis and Design Technique,* New York: McGraw Hill, 1987.

[13] Mayer, R. J. and Painter M. K., "The IDEF Suite of Methods for System Development & Evolution," working paper, Knowledge Based Systems Inc., 1991.

[14] Robson, G. D., *Continuous Process Improvement: Simplifying Work Flow Systems,* New York: The Free Press, 1991.

[15] Sibley, E. H., "An IDEF Family Portrait," working paper, George Mason University, 1988.

[16] Sibley, E. H., "The Evolution of Approaches to Information Systems Design Methodology," in *Information Systems Design Methodologies: Improving The Practice,* edited by Olle, T. W., Sol, H. G., and Verrijn-Stuart, A. A., North-Holland, 1986, pp. 1- 17.

[17] Strassmann, P. A., *The Business Value of Computers,* New Canaan, CT: The Information Economics Press, 1990.

[18] Yoemans, M., *Functional Policies for Corporate Information Management,* presentation at IDEF Users Group Symposium, October 16, 1991.

Performance Evaluation Gradient

Henry Neimeier
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102 USA

I. INTRODUCTION

The Infrastructure Engineering Directorate of the Defense Information Systems Agency (DISA) Center for Information Management is responsible for defining an information utility to provide data processing, storage, and value-added services to DOD users. One of the key questions on how to structure the utility is whether to contract for services or provide them within DOD. The primary goal of PEG is to aid in answering this question. Present procurement regulations and practices lead to long procurement and installation delays when compared to commercial practice. The age of the government installed base is far higher than in industry (see references 1&2). These effect the potential cost of government-provided services. Key information utility questions include:

- What are the key parameters that affect the outsourcing decision?
- What products or services are more efficiently outsourced ?
- What products or services are more cost effectively provided by the government ?
- How do government procurement delays affect costs ?
- How does the age of installed equipment affect costs ?
- What is the most cost effective equipment turnover rate ?

A secondary goal of PEG is to provide the basis for a Functional Economic Analysis of promising products or services. FEA is a methodology for modeling the costs, benefits, and risks associated with alternative investment and management practices, and is the primary decision support methodology for DOD business re-engineering. Since

July 1991 functional managers have been tasked with preparing FEAs for all proposed information technology alternatives. One approved methodology is defined in reference 3. A companion document describes an abbreviated FEA process that can be used in initial alternative definition (see reference 4). A FEA requires projection of workload, system cost, and performance five years into the future. It uses a discounted present value distribution of savings as a measure of effectiveness. Effectively projecting performance into the future in the fast-changing information technology environment requires a system dynamics model. PEG uses dynamic modeling to provide the discounted present value savings distribution.

II. MODEL DESCRIPTION

The model is implemented in version 2.0 of the i Think graphical simulation language. A full description is contained in reference 5. Each graphical symbol has an underlying supporting equation and documentation. Figures 1 and 2 give the graphical representation of the model for both government and commercial sectors. The continuous simulation model is a linked set of nonlinear integral difference equations. Boxes represent accumulations (integrals). Rate valves (circles with T at top- derivatives) control conserved flows in the double line pipes. Conserved flows such as units and dollars take time to move. Information flows represented by single lines are instantaneous. Clouds represent the external environment. Circles represent auxiliary equations. The complete model is documented in reference 6.

"i Think" has a sensitivity run capability that automatically performs a series of runs at specified input parameter values. This greatly simplifies the gradient calculation, since runs for each individual parameter automatically can be performed. Quadric surface fits require even more runs, so future development will be greatly aided by this feature.

Figure 1 shows the government sector and figure 2 shows the commercial sector. The commercial sector structure adds salvage, depreciation, Return On Equity (ROE) and profit to the government sector. Otherwise, the model structure is the same.

Of course, some of the parameter values are different. Note similar variables in the commercial sector have the same abbreviation as the government sector followed by a "C". Both sectors have a fee-for-service cost per unit (CPUopn & CPUopnC) that is calculated over the simulation time.

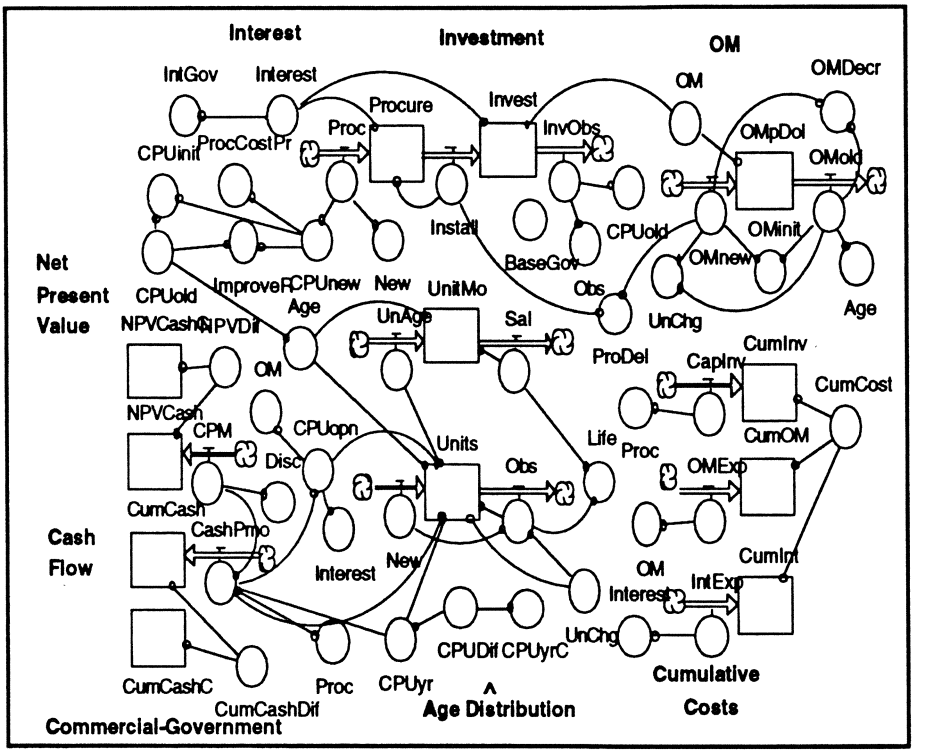


Figure 1. Government Sector Model

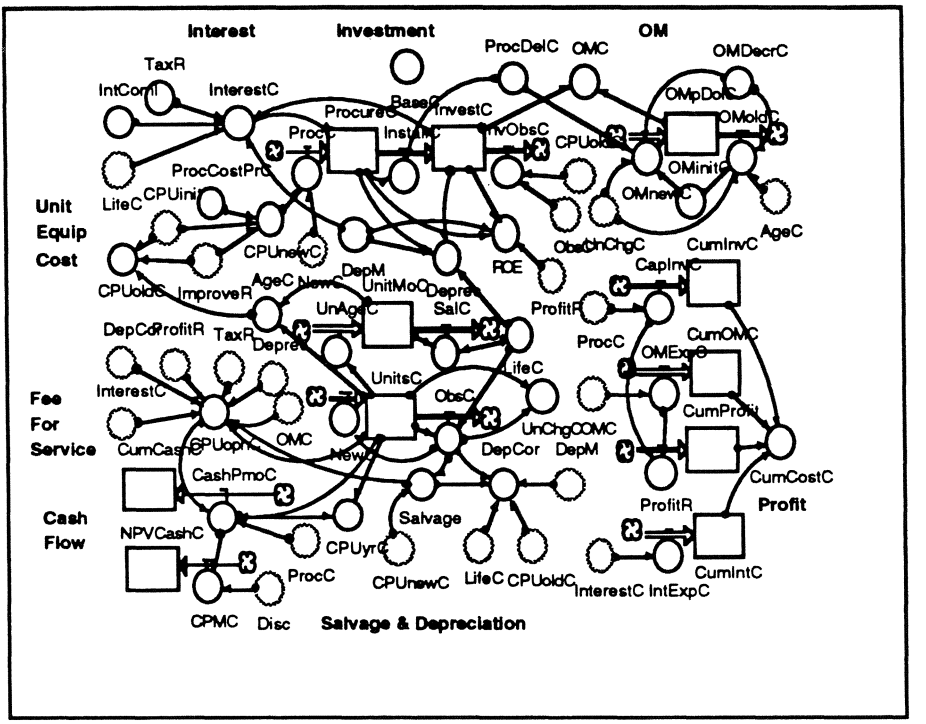


Figure 2. Commercial Sector Model

A. Measures Of Effectiveness

The two primary output measures of effectiveness used in the model are cumulative cash flow over the operations period, and discounted cash flow over the operations period. The operations period can be any length. For the fast-changing LAN environment we chose 60 months. Functional economic analysis uses discounted cash flow as its prime measure. The government discount rate of 10 percent was chosen, although other values are easily set.

B. Key Controllable Parameters

The Local Area Network (LAN) services area was chosen as an example case. This area has significant savings potential (see reference 7). LAN equipment and maintenance

costs are decreasing in excess of twenty percent per year. LAN services is one of the fastest growing components in the federal ADP budget. Operations and maintenance are a significant proportion of total costs. We estimated the present operating point (baseline parameter values) for both DOD and commercial service provision.

In the PEG model, cumulative cash flow is a nonlinear function of several input parameters. A baseline operating point was chosen and each parameter was individually varied from this operating point. The parameters that have the same values (operating point values in parenthesis) for both government and commercial sectors include:

- Initial LAN port unit equipment cost ($2000)
- Equipment price reduction (20 percent per year)
- Initial operation and maintenance cost per dollar invested (50 cents per year per dollar investment)
- Decrease in operation and maintenance cost (10 percent per year)
- Procurement cost percentage of unit equipment cost (4 percent)
- Equipment lifetime (36 months)
- Proportion of total unit equipment cost paid during the procurement and installation interval (25 percent)
- Discount rate for net present value calculation (10 percent)

Parameters that have different government and commercial values include:

- Average installed base age (30 months government, 15 months commercial)
- Procurement processing delay (12 months government, 3 months commercial)
- Interest rate (9 percent government, 16 percent commercial)

The parameters that only apply to commercial operations include:

- Commercial federal and state tax rate (40 percent)
- Equipment salvage value (25 percent of new equipment price)

- Depreciation life (twice equipment life- straight line depreciation)

The model easily accommodates changes in any of the parameter values.

III. MODEL RESULTS

No inflation rate was assumed for the model runs. Figure 3 shows how the yearly per port cost for both commercial and government operation decrease over the course of the simulation. The initial 1992 cost per port per year is $3200. The curves are almost identical since the operating point is close to break even cost. This includes a 60 percent commercial profit on sales.

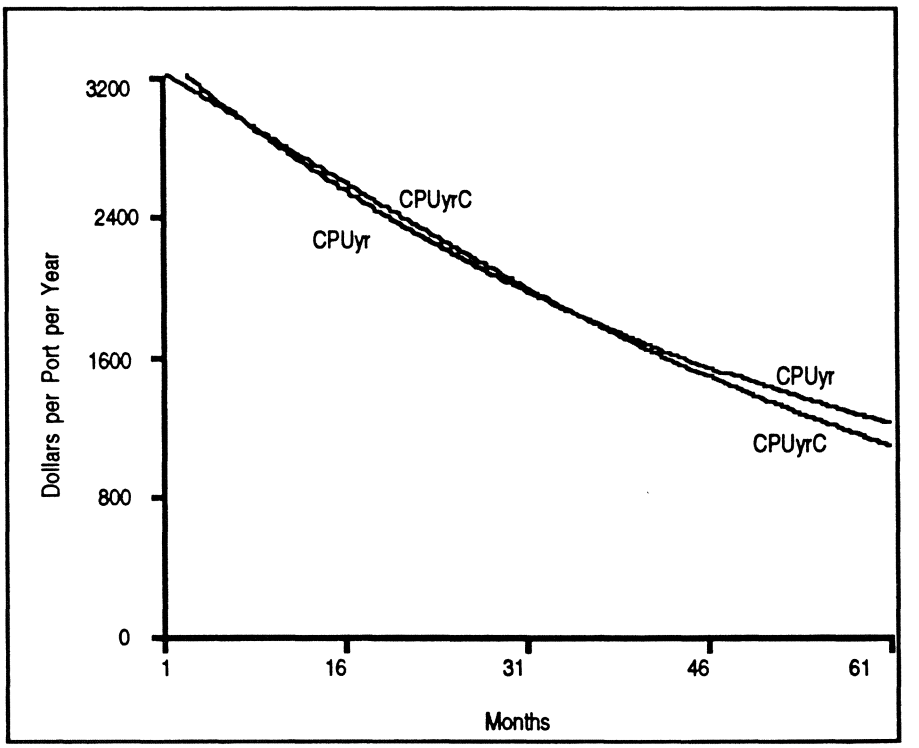


Figure 3. Yearly Port Cost

Figure 4 shows the cumulative government cost picture. Operations and maintenance are by far the largest cost category, followed by cumulative investment and interest. Note that installed capital investment (Invest) decreases throughout the simulation. More costly old equipment is replaced by less costly new equipment. The number of installed units is fixed at 30,000 for both government and commercial sectors. Future releases will investigate growth or reduction in installed base.

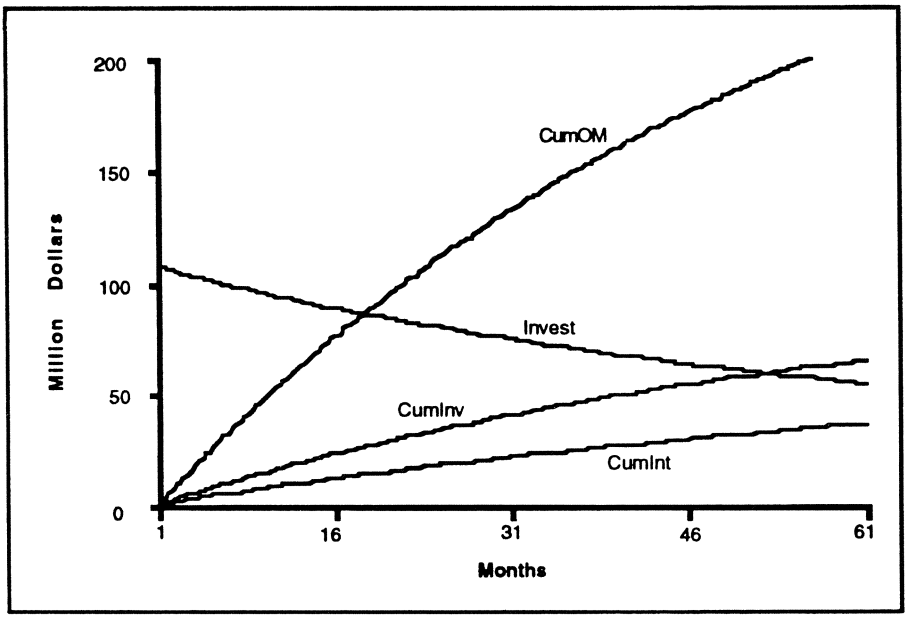


Figure 4. Cumulative Government Costs

Figure 5 shows the cumulative commercial picture. The largest cumulative cost is for operations and maintenance followed closely by profit. Cumulative investment cost over the operating period is next.

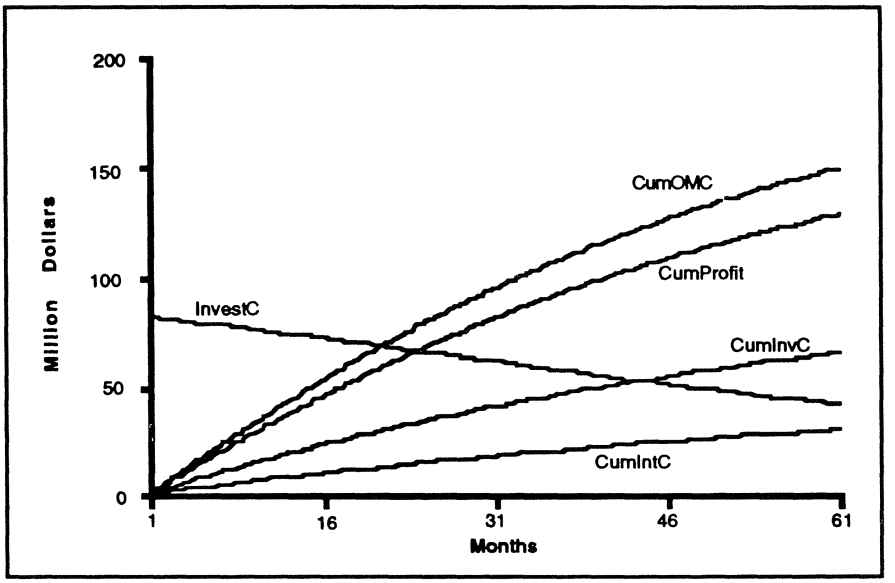


Figure 5. Cumulative Commercial Costs

Figure 6 shows the operations and maintenance cost per dollar invested for both government and commercial. Both show a significant decrease over the simulation period. Since capital investment is also decreasing, there is a substantial reduction in operations and maintenance costs. This is shown by a significant reduction in the slope of the cumulative operations and maintenance cost curves in figures 4 and 5.

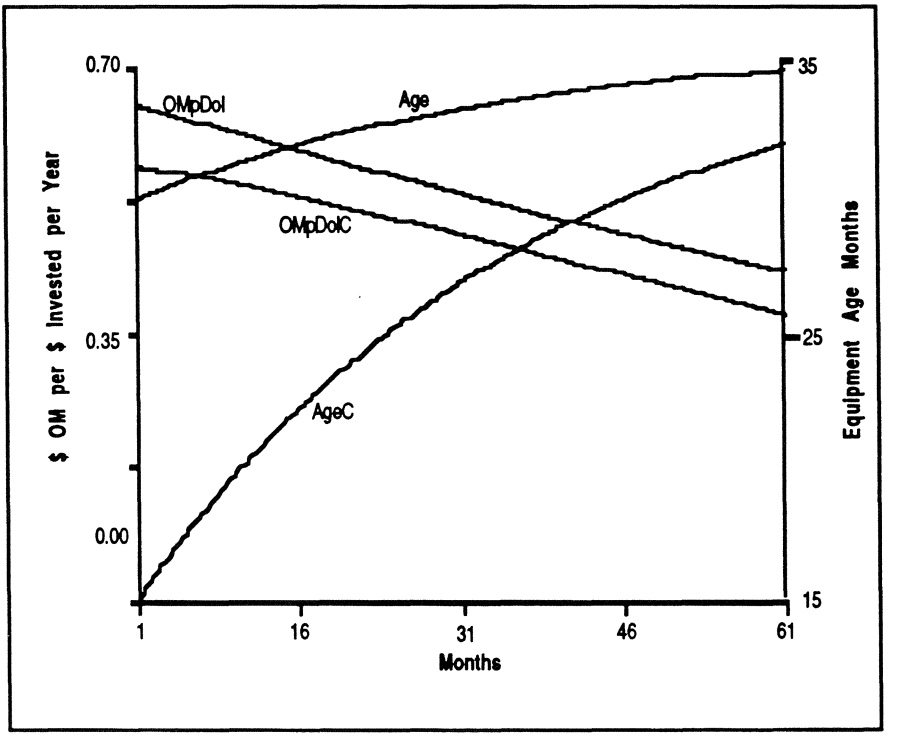


Figure 6. Operations and Maintenance Per Dollar Invested And Equipment Age

The operating point equipment life time is set to 36 months. Note that both government and commercial equipment age increases over the simulation period but does not reach 36 months. Thus steady state in equipment age is not reached in the simulation period.

The overall system boundary for this model is drawn around DOD rather than the entire government. The depreciation tax loss helps the commercial firm at the price of reduced Treasury income. This is equivalent to a Treasury subsidy to DOD outsourcing.

A. Factor Sensitivity Analysis

Over a five year operation period, the break-even commercial profit is 60 percent of sales. The break-even profit is where commercial and DOD operation are equally costly. If vendors are willing to accept a profit below the break-even point, then it is more efficient for the government to outsource the modeled services. This result is sensitive to the price reduction role, the installed government base age, and the procurement delay. The examples case assumes a 20 percent annual price reduction rate in equipment cost. For a commodity product or service, with no equipment price reduction, the break-even profit is 27 percent of sales. Greater rates of equipment price reduction result in higher break-even profits. A 30 percent price reduction rate produces a 92 percent break-even profit. If the average DOD installed base age is reduced from 30 months to the commercial value of 15 months, break-even commercial profit is reduced from 60 percent to 37 percent of sales. A reduction of DOD procurement delay from 18 months down to the commercial value of 3 months reduces total LAN procurement, operations, and maintenance costs by ten percent. These results are based on the commercial tax benefits available from depreciation and interest expense write-offs. In effect, the Federal Treasury is subsidizing DOD outsourcing.

Figure 7 shows the cumulative commercial cash flow advantage for various government procurement delays at the operating point. The operating point was chosen near the boundary where DOD and commercial service provision are equally costly. At twelve months or more government procurement delay outsourcing is superior. At nine months or less procurement delay DOD service provision is superior (assumes 60 percent commercial break-even profit on sales).

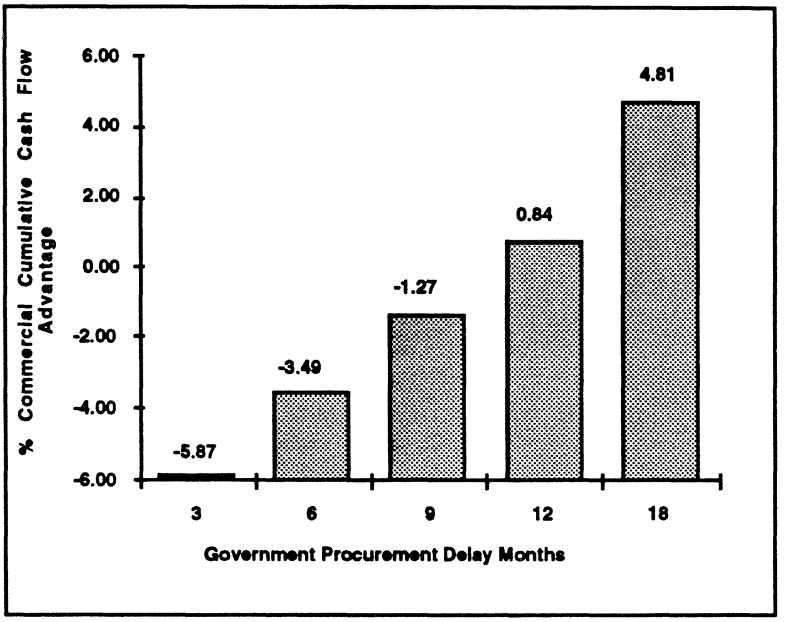


Figure 7. Commercial Cumulative Cash Flow Advantage Versus Government Procurement Delay At LAN Operating Point

Figures 8 and 9 show the percent increase in cumulative government and industry cash flow for various equipment lifetimes at the operating point. The optimal equipment lifetime is approximately 36 months for industry and 42 months for government. The commercial salvage value, depreciation and interest expense write-offs lead to a shorter optimal commercial equipment life time. Longer or shorter equipment lives than the optimum result in increased costs. The optimal equipment life is dependent on the operating point. Faster equipment price reduction rates and greater operation and maintenance costs as a proportion of investment lead to shorter optimal equipment life times. Higher procurement cost proportions and interest rates lead to longer optimal equipment life times. The reciprocal of equipment life time in months is the equipment turnover rate per month.

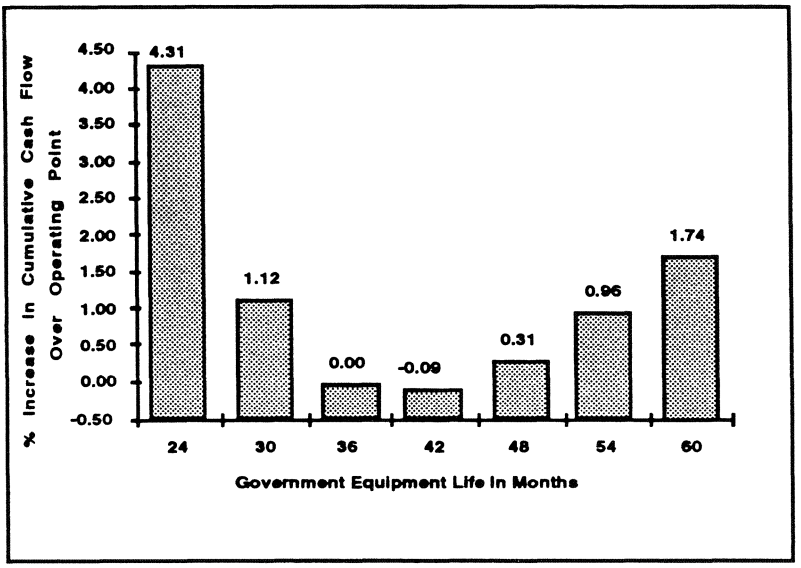


Figure 8. Cumulative Cash Flow Versus Government Equipment Life At LAN Operating Point

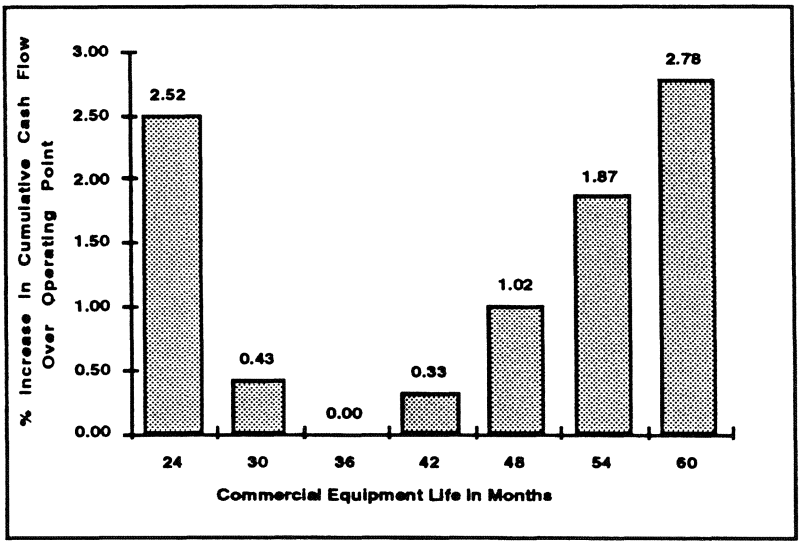


Figure 9. Cumulative Cash Flow Versus Commercial Equipment Life At LAN Operating Point

B. Gradient

The gradient gives the change in output per unit change in selected parameter value when all other parameters are held fixed. It is the partial derivative of the output measure relative to each controllable model parameter. The model output measures are the cumulative cash flow and the discounted cash flow. The gradient gives the normal to the hyperplane tangent to the iso-cost surface (see figure 10). It is the direction of fastest increase in cost. Figures 11 and 12 show the cumulative cash flow gradients for government and commercial operation. For example, in Figure 11 every 1 percent increase in government procurement cost proportion (ProcCost percent) of equipment price results in 1.64 percent increase in total cumulative cash flow (cost) over the operations period. The factor abbreviations, full names, and factor change for gradient calculation (in parenthesis) are as follows:

- Life6moDec, equipment lifetime decrement (6 months)
- Proc mo, procurement and installation time increase (1 month)
- Interest%, Interest rate increase(1%/year)
- Price %, equipment price increase (1% / year)
- OM/$Init%, initial annual operations and maintenance dollars as a percentage of investment dollars increase (1%)
- OM/$inc%, increase in operations and maintenance costs (1%/year)
- ProcCost%, increase in procurement processing cost as a percentage of equipment price (1%)
- Profit%, increase in profit as a percentage of sales (1%)
- Tax%Dec, decrease in tax rate (1%)

• DepLife%Dec, decrease in depreciation life time (1%)

• Salvage Dec, decrease in salvage percentage (1%)

The dot product of the gradient with a proposed policy vector, which represents a plan to change parameter values, gives the cost impact of that policy. Gradients and policy vectors should be investigated at various operating points in future extensions of the model. Since the actual iso-cost surface is curved, the gradient only holds for a small region around the operating point. Another extension would be to fit a quadric surface to a set of model runs. This would aid in determining minimal cost operating points and apply over a larger operating region. A simple three-parameter example is given in the next paragraph. Reference 8 describes a full performance evaluation surface model developed as an extension to this paper.

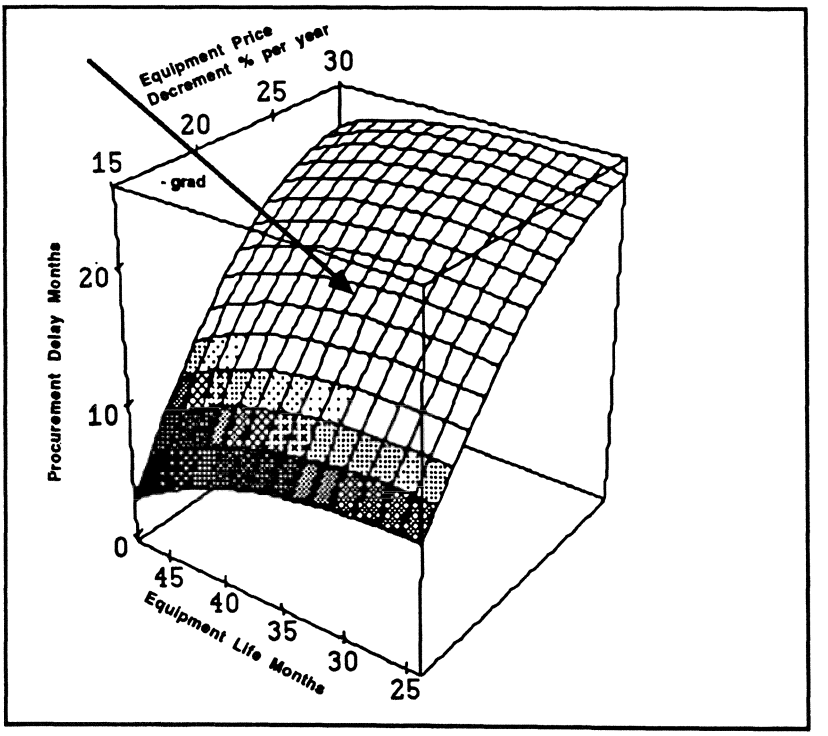


Figure 10. 60 Percent Cost Reduction Surface and Gradient

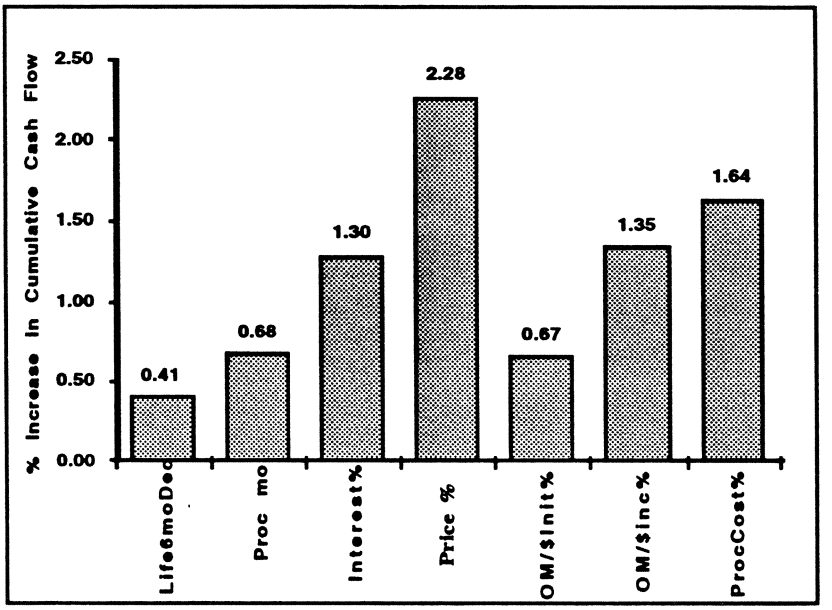


Figure 11. Government Cumulative Cash Flow Gradient

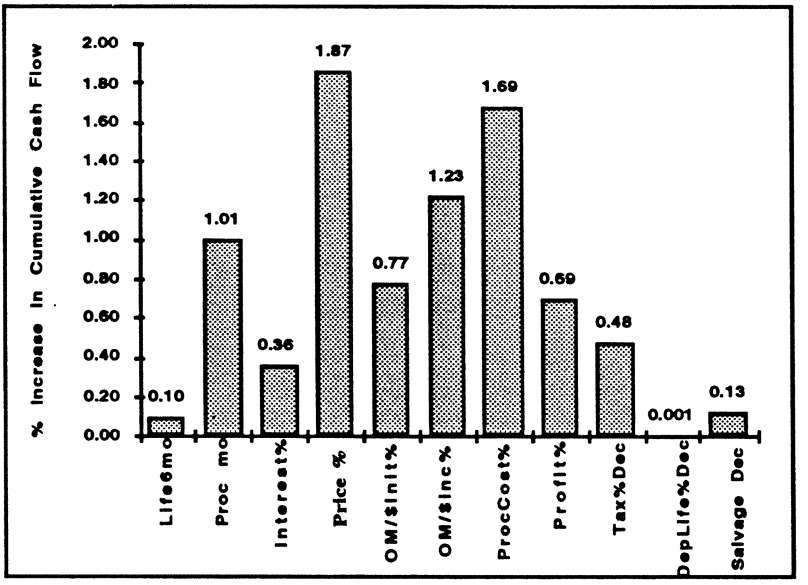


Figure 12. Commercial Cumulative Cash Flow Gradient

C. Cost Hyper Surface

A complete factorial design of 27 PEG runs was completed to vary equipment lifetime, procurement delay and decrement in equipment price. The output measure was total cost per unit at the end of the five year run divided by initial cost per unit at the start (C). The following parameter abbreviations and values were used.

- (life) equipment life in months [24,36,48]
- (dec) decrement in equipment price percent per year [10,20,30]
- (delay) procurement processing delay in months [3,12,21]

The following quadric surface was least squares fit to the commercial data.

$$C = .701 - .003\ life - .032\ dec + .013\ delay + .00006646\ life^2 + .0004526\ dec^2 + .00007181\ life\ dec - .0001424\ life\ delay$$

All terms were highly significant (.0008 t probability level for life coefficient , .0001 t probability level for all other factor coefficients). The multiple correlation coefficient squared was 99.7 percent, so only 0.3 percent of the data variability was not explained by the equation. To obtain the 40 percent total cost surface (60 percent reduction), we set C=.4 and solve for procurement delay (delay) in terms of equipment life (life) and decrement in equipment price (dec).

$$delay = (301 - .003\ life + .00006646\ life^2 - .032\ dec + .0004526\ dec^2 + .00007181\ life\ dec) / (.013 - .0001424\ life)$$

To obtain the gradient (grad) we take the partial derivatives of C relative to life, dec, and delay. The partial derivatives are shown below along with numerical values in parenthesis at the following point on the 60 percent cost reduction surface:

- 36 month equipment life,
- 20 percent decrement in equipment price,

- 16.27 month procurement delay on the (40 percent cost reduction surface for life of 36 months and annual equipment price decrement of 20 percent).

$$\partial C/\partial life = -.003 - .0001424\ delay + .00007181\ dec + .00013292\ life$$

$$(.000904472)$$

$$\partial C/\partial dec = -.032 + .0009052\ dec + .00007181\ life$$

$$(-.0113108)$$

$$\partial C/\partial delay = .013 - .0001424\ life$$

$$(.0078736)$$

The gradient (grad) gives the direction of most rapid cost increase, and is perpendicular to the cost surface at the operating point. We are interested in the negative gradient, i.e. the direction of most rapid cost decrease. Figure 10 shows the cost surface and perpendicular gradient. Though we can only show 3 dimensional surfaces, the mathematical techniques apply equally to higher dimensions. An extension would be to fit a quadric surface to all parameters (14 dimensional) and calculate the gradient. Setting the partial derivatives equal to zero we can find the least cost and highest cost operating points. The Mathematica software package (reference 9) considerably aided the calculations and plotting
performed in this section.

IV. CONCLUSIONS

The PEG model can quantitatively evaluate the economics of outsourcing. The model is general and can separately be applied to a wide variety of product categories. Products are categorized by their similarity in average parameter values. A by-product of the model is a cost per unit output versus simulation time. This should aid in setting fee-for-service algorithms.

At the assumed operating point, the break-even commercial profit is 60 percent of sales. Thus outsourcing LAN services seems superior. However this is based on some key assumptions that should be checked:

- Average government installed base age of 30 months versus 15 months for commercial (37 percent break-even profit for both 15 months)
- DOD system boundary that does not account for the depreciation tax losses to the Treasury
- No government outsourcing monitoring costs (not contained in the model)
- There is no DOD salvage value - change in regulations or equipment refurbishment and reuse could change this assumption
- LAN equipment price reductions of 20 percent per year (higher growth rates lead to a higher break even profit)

Significant government cost reduction can be obtained from a reduction in procurement delays (7 percent of total costs by reducing delay from 12 months to 3 months). The optimal government equipment life time is approximately 42 months (for the assumed operating point). Lower capability growth rates and smaller operations and maintenance decrement rates would lead to longer optimal equipment life times.

LAN services are only one category of product. Other categories should be defined for products with different characteristics (different average parameter values operating point) such as:

- Equipment price decrement rate
- Operations and maintenance cost proportion
- Procurement cost proportion and time delay

Products can be ranked by the break even commercial profit on sales. The highest profit products are the prime candidates for outsourcing.

A by-product of the model is cost per unit versus time. This should guide setting fee-for-service algorithms. The model provides simplified sensitivity analysis, so the impact of changes in cost of any model parameter can be determined. Key parameters to be investigated include:

- Age of installed equipment base
- Growth rate in installed equipment units
- Equipment lifetime
- Improvement in equipment price and operations and maintenance cost per dollar invested
- Initial equipment costs and annual operations and maintenance per dollar invested
- Corporate tax rates and depreciation schedules
- Interest rate

During model runs both installed base age and growth in installed base units were very sensitive parameters. These should be investigated fully in future extensions.

A three-dimensional cost reduction surface and its gradient were presented. The technique should be extended to all the key parameters. An optimal operating point can

be determined from the surface equation by setting its partial derivatives to zero. It should be possible to get an analytic solution to the model differential equations. This would both speed calculations and simplify the gradient calculation.

The dot product of the gradient vector with a policy vector (plan to change parameter values) gives an estimate of the improvement obtainable by that policy. The fastest cost improvement is in the negative gradient direction. However some parameters are not controllable such as interest rate and corporate tax rate. Others will require process changes that take time such as legal requirements in procurement process and the government procurement process itself. Thus the policy vector differs practically from the gradient. The potential policy vector envelope should be explored in future work.

V. REFERENCES

[1] December 1990, *A Five Year Plan For Meeting The Automatic Data Processing And Telecommunications Needs Of The Federal Government,* Office of Management and Budget, Washington, D.C.

[2] H.Neimeier, July 1991, *Architecture Performance Evaluation (APE)- Model,* MTR-91W00091, The MITRE Corporation, McLean, Virginia.

[3] P. Byrnes, H.Neimeier, et. Al., January 1992, *A Functional Economic Analysis Reference Methodology*, MTR-91W00198, The MITRE Corporation, McLean, Virginia.

[4] H. Neimeier, February 1992, *MITRE Abbreviated Functional Economic Analysis Tool (MAFEA)*, MTR-92W0000037, The MITRE Corporation, McLean, Virginia.

[5] B. Richmond, et. Al.., 1991, *i Think , The visual thinking tool for the 90's: i Think User's Guide*, High Performance Systems Inc., Hanover, New Hampshire.

[6] H.Neimeier, February 1992, *Performance Evaluation Gradient (PEG),* MTR-92W0000038, The MITRE Corporation, McLean, Virginia.

[7] S.Ficklin et.Al., March 1992, *Premises Services Savings Assessments,* WP 92W0000098, The MITRE Corporation, McLean, Virginia.

[8] H.Neimeier, April 1992, *Performance Evaluation Surface (PES),* The MITRE Corporation, McLean, Virginia.

[9] S.Wolfram, 1988, *Mathematica : A System for Doing Mathematics by Computer,* Addison-Wesley Publishing Company, Redwood City, California.

DEFENSE BLOOD STANDARD SYSTEM FUNCTIONAL ECONOMIC ANALYSIS: A CASE STUDY

Carla von Bernewitz
and
Marty Zizzi
Vector Research, Incorporated
901 South Highland Street
Arlington, Virginia 22204 USA

The Defense Blood Standard System (DBSS) Functional Economic Analysis (FEA) was the first FEA to be accepted by the Department of Defense (DoD) Director of Defense Information (DDI), Paul Strassmann. It was one of the first FEAs, also known as business cases, to analyze functional area costs and to apply business process reengineering in structuring alternatives to the current way of doing business. The DBSS FEA examines the business processes and practices of the DoD blood management function and analyzes changes in those practices or supporting information technologies.

I. OVERVIEW

A FEA examines current and proposed operations and expected financial results prior to the decision to invest in any new business practices and associated information technologies. It quantifies functional area costs, benefits, and risks, and adjusts the dollar amount for the time value of money. It may also aid in managing the business changes and associated benefits resulting from an investment decision.

A FEA allows the current and forecasted costs of baseline operations over a stated economic life to be compared to alternatives associated with management initiatives. Sunk costs are not included in a FEA. A FEA provides a uniform basis for analyzing and comparing alternatives to the current way of doing business. It applies to decisions involving proposed and existing business methods and current and proposed information technologies. A FEA provides support for and input to the decision-making process by comparing investment in various alternatives to a baseline, where each alternative is defined as a management initiative leading to potential functional area savings.

To present the quantitative data necessary to support the decision to invest in new business methods and information technologies, baseline costs are gathered and presented in the format required by the DDI. For the DBSS FEA, life cycle costs of continuing operations were forecasted in millions of then year (current) dollars over a 12-year economic life, with a seven-year planning phase, both beginning in Fiscal Year 1991 (FY91).

In order to perform the DBSS FEA responsively, a model and a set of spreadsheet tools were developed for compiling cost and benefits data for the baseline and selected alternatives. As noted above, costs are depicted in then year dollars, the conventional way DoD budgetary data are portrayed. However, the completed analysis requires a calculation of ROI which is calculated from discounted, or present value, cash flows, rather than from then year values themselves. The tools perform risk-adjusted present value

comparisons and compute the ROI using 10% mid-year discount factors.

The assessment of risk in a proposed project or alternative is perhaps the most difficult and yet most critical task that faces managers charged with decision making. Risk analysis attempts to account for underlying forecast error. Placed in financial terms, risk analysis allows senior managers to view the potential advantages of a proposed alternative balanced against the chances that predicted outcomes may not occur. The analytical methodology used to quantify and describe risk in the DBSS FEA uses a probability density function to model the risks. A lognormal form of this probability density is applied to create a variety of risk profiles that the functional expert can evaluate in terms of their view and experience. The lognormal form is useful because it is flexible yet quantitatively valid for a variety of perceived situations.

II. METHODOLOGY

The DBSS FEA was developed and iteratively refined in the sequence of six steps shown in figure 1 and listed below:

- determine scope of functional area;
- identify business practices;
- gather baseline costs;
- define new business practices, alternatives and associated benefits;

- perform financial simulations and analyze costs; and
- make recommendation(s) and plan for transition.

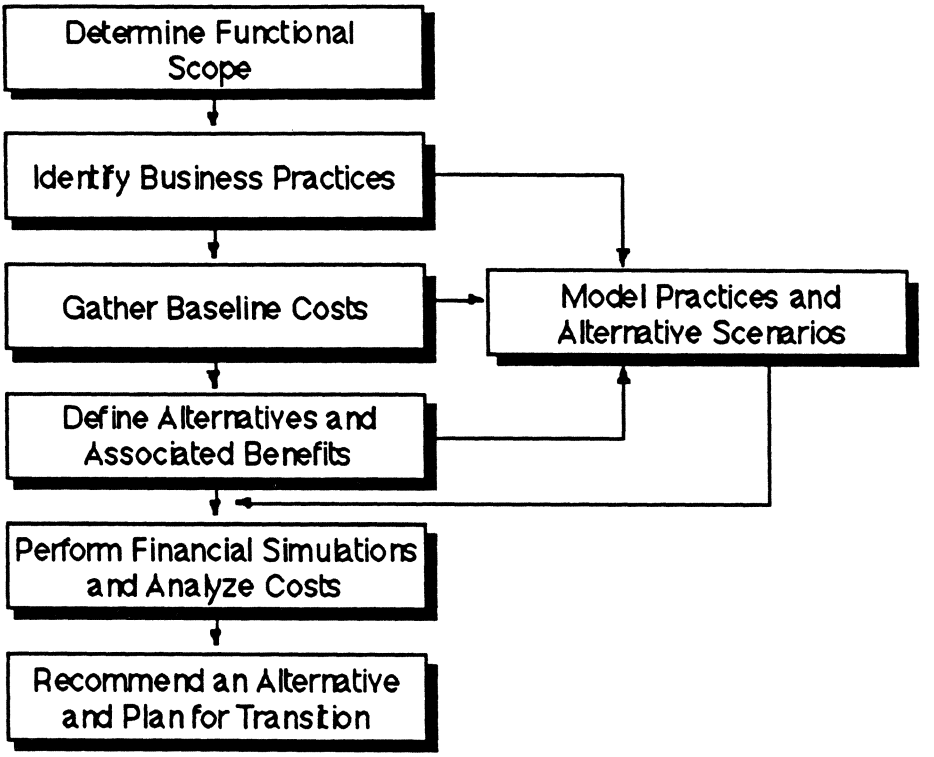


Figure 1. Methodology

These steps are explained in detail in the following paragraphs. Modeling is described as an analytical technique rather than a separate step.

A. Scope

Scoping the functional area in which the FEA will be performed is the first and perhaps the most challenging step. FEAs are performed for functional areas, rather than information systems, locations, or organizations. Functional areas may contain supporting information technology (systems) and other programs of interest to the FEA. Often, a functional area analysis crosses traditional budget or program definitions such as civilian personnel support for a variety of functions on a military base or post. A FEA provides the actual context in which business is conducted and therefore a realistic basis under which the business case will be analyzed.

During the scoping process, it is helpful to refer back to traditional strategic planning products, such as the mission and vision. Such products help define the boundaries, direction, and activities of the functional area. A high-level DoD enterprise model has been defined, which also aids in the scoping process. Doctrine, policy, and guiding principles provide guidance as well. It may be necessary to revisit the scope several times, especially as baseline costs are gathered for the function.

B. Business Practices

Once the scope of the functional area has been defined, the business practices applicable to the functional activities, functions, and processes are identified. Whereas functions describe *what* is performed, business practices describe *how* a function is performed.

To describe the functions, processes, and business practices of the functional area, a business process model is built. For the DBSS FEA, a business process model was built using Information Engineering (IE) techniques such as Function Decomposition Diagrams (FDDs) and Entity Relationship Diagrams (ERDs). The Integrated computer-aided manufacturing Definition (IDEF) technique has since been chosen by the DDI for all business process modeling.

Current business practices, their deficiencies, and supporting information technologies are first identified. Next, new business practices in the form of improvements and changes to current business practices are defined. Policy and statutory modifications may be required to support new business practices. New information technologies that are required to support and implement new business practices also are delineated.

The application of information systems and information technologies comes into play only after revised business practices have been examined thoroughly and agreed upon. The FEA does not automatically imply the application of information technology in a recommendation or solution. Information systems and technology can, however, make possible changes in business methods that would have been otherwise infeasible. For example, bar codes allow supply items to be labeled, tracked, and reviewed automatically with an integrated data entry mechanism that reduces clerical error. Smart cards carry complete and accurate records more efficiently than bulky folders. The decision to use information technology may be driven, however, by a business need for new ways of doing business, such as

lowering costs, or finding more accurate and timely ways of delivering products from vendors to final customers.

C. Baseline Costs

Baseline costs represent the cost of doing business as usual. Gathering the costs of maintaining and operating the functional area is the next step in a FEA. The costs are termed enterprise costs to convey the concept of an ongoing business enterprise. These are the costs needed to operate the entire functional area.

For the DBSS FEA, baseline costs were either gathered from DoD budget documents, extracted from the Medical Expense and Performance Reporting System (MEPRS) of the Military Health Services System (MHSS), or calculated from probabilities and costs for infectious diseases from MHSS data reports and literature citations from organizations such as the Centers for Disease Control (CDC). All baseline enterprise costs were modeled and aggregated into the five FEA categories required by the DDI:

- Personnel;
- Facilities;
- Materiel;
- Information Technology; and
- Other.

Personnel costs are defined in terms of dollars, although the initial personnel data were obtained as end strengths or Full Time Equivalents (FTEs). The budgeted authorized number of military officers, enlisted personnel, and civilians is identified for the functional area. Composite DoD military compensation rates from the President's Budget are used to calculate personnel costs from the authorized number of personnel. Specific personnel costs may be substituted when this detail is available and serve to more accurately describe this important cost category.

Facilities costs are based upon square footage requirements to operate the enterprise. Materiel costs include supplies and equipment. Information Technology costs are those that account for existing information systems in the baseline and new information systems in the alternatives.

The "Other" data category contains costs that are not already allocated to one of the previous four categories. For the DBSS FEA, the Other category was subdivided into two cost categories, Miscellaneous and Liability. Liability costs represent the health care and litigation-induced settlement costs for transfusion-transmitted diseases.

There are several additional ways data may be classified for these five cost categories. Within each category, Operations costs and Management and Support (M&S) costs may be defined. Operations relates to those costs in direct support of delivering or producing the products and services of the functional area. M&S refers to those costs not directly related to the end products of the functional area. M&S costs are thus indirect costs and may be viewed

as overhead costs. For the DBSS FEA, once total costs were confirmed, the percentage of M&S and Operations costs were identified. The allocation between operations and overhead may be an estimate, since accounting principles employed may not support a definitive derivation of overhead costs.

With the exception of Personnel, costs in the five cost categories may also be classified into more than one type of funds or appropriation. Examples are Operations and Maintenance (O&M), Other Procurement (OP), and Military Construction Program (MCP). O&M costs are budgeted costs used to maintain current resources. OP costs, also called procurement costs, are funds that are used for acquiring additional or new resources other than facilities, at costs that exceed a set dollar threshold. The threshold for FY91 was $15,000. MCP costs are used for major facility acquisitions. These costs are included in the baseline where they are planned, programmed, and funded. Additional investment costs for alternatives are not included in the baseline.

D. Alternatives

The foundation for consideration of alternatives is new business practices. These may be adopted from other successful enterprises, or developed specifically for the particular FEA. The alternatives are defined in terms of the functions and new business practices they support, and the information technology support that is required to render them feasible. The baseline is not considered an alternative; it is the status quo of doing business as usual. Three or more separate and distinct alternatives

were required to support the DBSS FEA; only two are currently required for FEAs. These alternatives may vary by the number of functions and business practices implemented, their rate of implementation of business practices, or their degree of information technology support.

After the alternatives have been described, the costs associated with them are enumerated. Investment costs for each of the alternatives are delineated. Benefits, which are defined as reductions in baseline costs, are quantified by the functional experts. Risk profiles are chosen for each alternative and the baseline as well.

E. Financial Simulations

A modeling tool supports the financial simulations that project risk adjusted discounted cash flows (RADCFs) for the baseline and each of the alternatives. This adjustment for risk allows expected value, low, and high ROIs, costs, and benefits to be calculated. The tool converts FY91 dollars into then-year (or current dollars) and net present value dollars over the FEA life cycle and calculates the ROI. Detailed and summarized financial costs and benefits in the required five cost categories, total costs and benefits, and ROIs are calculated. The tool also generates charts and graphs similar to the one shown in figure 2 that depict summarized financial simulations data.

An analysis of the ROIs, risk, and life cycle costs and benefits of the alternatives compared to the baseline is performed once the financial simulations have been executed. While one alternative may have appeared preferable before simulating, the results from the simulations may lead to a

different conclusion. The RADCFs are also examined, since the high and low values dictated by the risk profile may support different decisions than the expected value does.

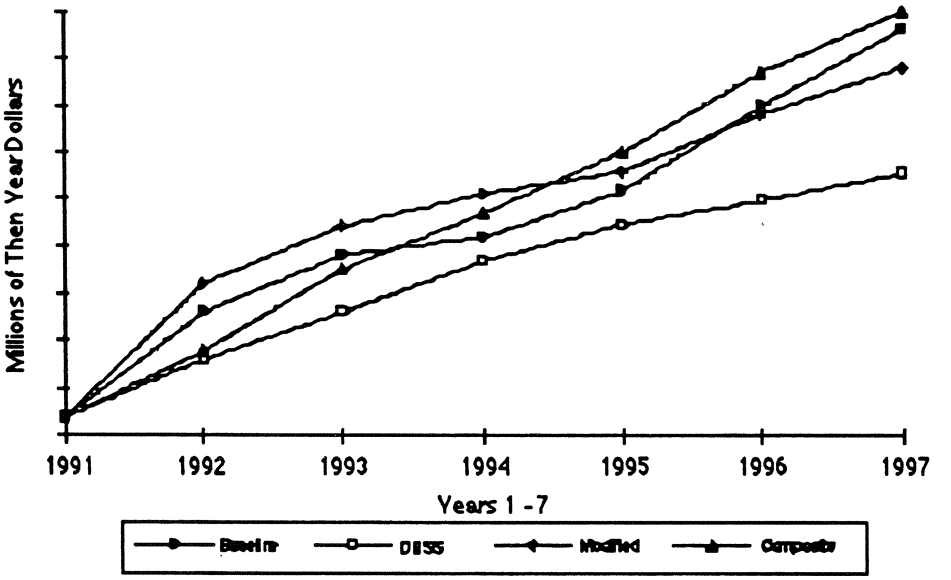


Figure 2. Financial Simulations

F. Recommendations

The FEA concludes with a recommendation based on the results of the overall analysis. The analysis is summarized using a variety of graphical aids and is presented to the

executive decision maker. A plan for transitioning or migrating to the recommended alternative is then prepared.

III. Modeling

In analyzing a business case, several interactive models are built, as shown in figure 3. All FEAs, however, begin with the business process model, which describes the activities and practices of the functional area. Using various costing techniques, such as Activity Based Costing (ABC), a model of the costs required to operate the functional area is developed. The following paragraphs discuss the cost models and the application of risk that is required for FEAs.

In order to perform a FEA, it is useful to take advantage of the spreadsheet software commercially available for personal computers. Electronic spreadsheets facilitate the collection and creation of various data needed to perform the analysis. The application of a variety of modeling software packages designed to support the financial simulations that project RADCFs for the baseline and selected alternatives is desirable. Alternatively, a financial simulation tool can be developed that is expressly tailored for a particular FEA.

A tailored tool was developed to support the DBSS FEA. The tool performs associated risk-adjusted net present value comparisons. It forecasts then-year costs for each of the major FEA categories and computes net present values and ROIs. Risks are considered at the summary level for each alternative. Within an alternative, the same risk profile is applied to all cost categories. Risk profiles are

selected by functional users. All costs are presented in millions of then year dollars to one decimal place for the entire 12-year FEA life cycle.

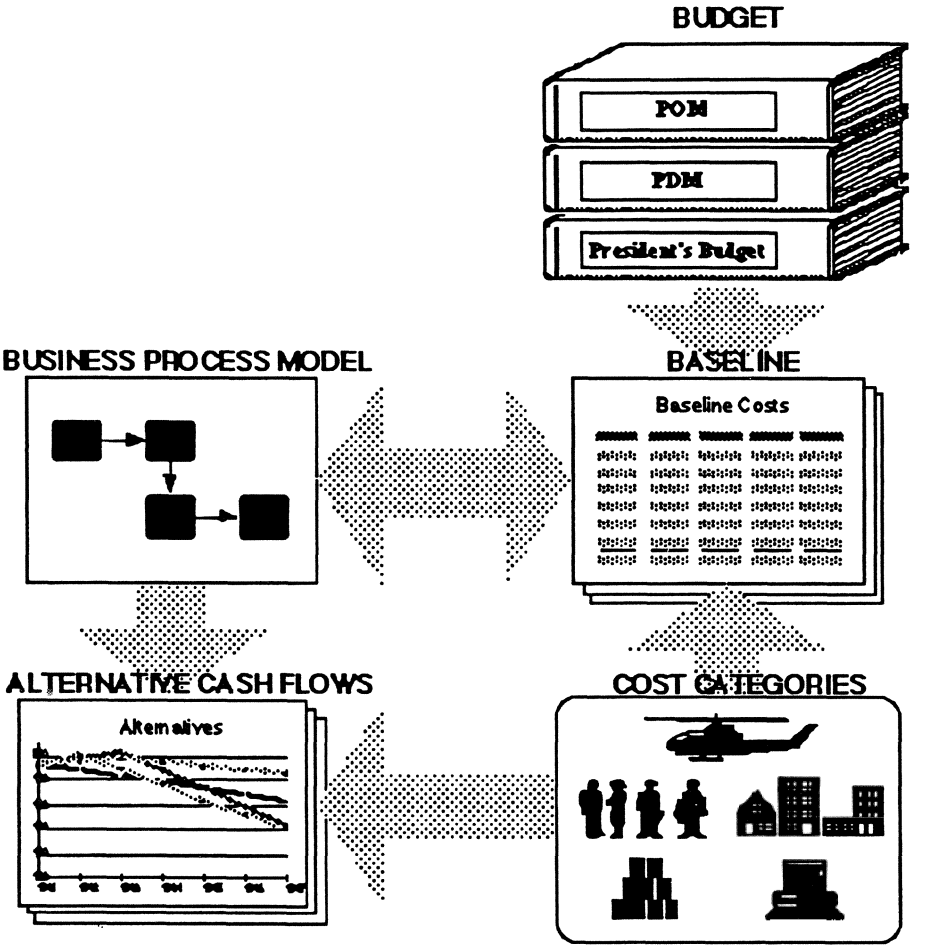


Figure 3. Modeling

A. Risk

The model assumes that the risk inherent in the baseline and each alternative can be identified and modeled in terms of a normalized probability density function of cost multipliers. The shape of cost multiplier distributions can be varied to accommodate specific *a priori* assumptions of risk. A set of risk distributions for information technology is suggested by Mr. Paul Strassmann in chapter nine of The Business Value of Computers. Mr. Strassmann indicates the suggested distributions are based upon his experience with multiple systems, but are not based upon rigorous statistical analysis. The use of the lognormal form of these distributions permits flexibility in assisting the functional expert to describe the environment surrounding risk while maintaining quantitative validity. Thus, a risk profile can be constructed that fits the perceptions and experiences of the functional expert as to how risk might vary depending on the alternative selected. It can depict that costs are stable and well controlled, highly volatile, or somewhere in between.

After discussions and consultations with the functional experts for all alternatives, each risk profile is mathematically derived. A risk profile is a distribution of cost multipliers and their associated probabilities. The probability of each cost multiplier is based on its frequency in 2,000 sample observations. The resulting profile depicts the cost multipliers on the x-axis and the probability of the multiplier on the y-axis.

The risk profiles were constructed using a three-parameter equation of a lognormal form as specified in

Distributions in Statistics, Volume 1, chapter 14. The parameters used were the standard deviation, the mean, and a shift parameter. In this application, the "expected value" of a risk profile was interpreted as the mode of that distribution. This definition of expected value was driven by an understanding that budget planners would be motivated by their targeted authorizations -- the most likely amounts. They would not be motivated by mathematical computations that do not reflect their nominal, or budgeted, amounts. Profiles designed this way imply that the cash flows developed with the functional experts would be the likely outcome in a one-time draw of that alternative. If multiple draws were conducted, however, then the simulation model shows the distribution of all outcomes, including the most likely outcome. Risk is specified by the lower and upper bounds of the probability density function (standard deviation parameters) and its shape.

The creation of each risk profile followed the same process. The first step was to determine the nominal budget amounts. As stated above, these were interpreted as the "expected," or most likely values. The next step was to determine the known volatility of a cost estimate for an alternative. This high-level analysis of the interdependencies of cost elements and of the potential weaknesses of an alternative was the basis for determining the endpoints of its cost multiplier distribution. A better understanding of the character of the risk facing an alternative was gained at this stage as well. It was this understanding of the character of the risk that was used to mathematically define the shape of the risk distribution. As endpoints and the mode of the curve were predefined, it

was a matter of adjusting the standard deviation and the shift parameter to obtain the appropriate shape.

In aggregating costs for a particular alternative, the model used in the DBSS FEA assumes that elements from each category are perfectly correlated. This assumption is conservative in the sense that outcomes calculated for total project cost depict a wider dispersion than if category costs were randomly or only partially correlated. If costs are completely uncorrelated, the probabilities associated with the extremes for aggregated costs would be less, as independent samples of costs for each category when added would tend toward the expected value. The perfect correlation approach was selected for computational simplicity, ease of implementation in a spreadsheet environment, and consistency with the belief that some correlation exists between categories. All results for this business case must be considered with the above assumptions in mind.

The impact of the relative risk of the baseline and each alternative becomes apparent as the distributions of benefits and ROIs are calculated. The model calculates ROI as the quotient of the baseline discounted cost minus the alternative discounted cost (including investment) divided by the alternative discounted investment. This calculation is performed for every possible combination of the derived cost multipliers for the baseline and alternatives (twenty values for the baseline and each alternative resulting in 400 total observations). The frequency distribution of these individual ROI values is supported by the underlying risk for each alternative, since each alternative, as well as the baseline, already is influenced by specific risk

profiles. The calculation includes the cost of the investment in the alternative. Thus, the resultant ROI can be considered a net value.

B. Cost Factors

The model for this FEA uses deflators from the Office of the Secretary of Defense (OSD) Comptroller, and real growth rates for cost elements to depict costs as either constant year or then-year (current) dollars. The analysis in then year dollars shows amounts that relate directly to budgeted dollar requirements. The analysis in constant year dollars removes inflation from the picture and reveals the effects of real growth on costs. When data were obtained for FYs other than FY91, the appropriate deflators and growth rates were applied.

The model uses a mid-year discount factor of 10% as provided in the Office of Management and Budget guidance (Circular A79-Revised, 27 March 1972). Discounting cash flows is a method to adjust dollar amounts to show the cost of capital or the opportunity cost of the cash flow. The opportunity cost of cash expenditures is the return on the next best available alternative. The discount rate is generally accepted as a proxy for the return available in the government long-term bond market, which is generally considered the most reasonable investment alternative. Use of the government long-term bond rate assumes that the risk preference of the investor is indifferent to the level of investment risk between the alternative and the government long-term bond market. If the investor's risk preference allows a relatively higher level of investment risk than

that of the government long-term bond rate, it may be appropriate to adjust the discount rate accordingly.

C. Life Cycle

The DBSS FEA uses FY91 as the beginning of the life cycle. The length of the analysis is 12 years, which represents the economic life of the business practice changes and supporting information technologies. The economic life may range from 12 to 20 or more years. This 12-year period includes a seven-year planning life cycle with an additional 5 years of residuals. Residuals allow benefits beyond the planning phase to be captured and included in ROI calculations. All costs are detailed by year from the base year FY91 through FY97, the end of the planning period. Residual values are calculated for FY98 through FY2002.

IV. REFERENCES

[1] Bureau of Labor Statistics, C*onsumer Price Index Urban Consumers*, Medical Components unadjusted May 91/May 90 annualized rates.

[2] *Circular A79-Revised*, 27 March 1972.

[3] *Department of Defense Corporate Information Management*, ASD (C^3I), April 1991.

[4] *Distributions in Statistics*, Volume 1, Chapter 14, pp 112-114.

[5] *OMB Circular A-79-Revised*, 27 March 1972.

[6] Strassmann, Paul A., Director of Defense Information, To the House Appropriations Committee Defense Subcommittee, 24 April 1991.

[7] Strassmann, Paul A., *The Business Value of Computers*, 1991.

[8] Strassmann, Paul A., *The Goals and Directions of DoD Corporate Information Management Briefing*, 19 June 1991.