/N-61
P 94

SOFTWARE ENGINEERING LABORATORY (SEL) RELATIONSHIPS, MODELS, AND MANAGEMENT RULES

FEBRUARY 1991

(NASA-TM-105513) SOFTWARE ENGINEERING LABORATORY (SEL) RELATIONSHIPS, MODELS, AND MANAGEMENT RULES (NASA) 94 p CSCL 09B

N92-18123

Unclas
G3/61 0068914

NASA
National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY (SEL) RELATIONSHIPS, MODELS, AND MANAGEMENT RULES

FEBRUARY 1991

NASA
National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1976 and has three primary organizational members:

NASA/GSFC, Systems Development Branch

The University of Maryland, Computer Science Department

Computer Sciences Corporation, Systems Development Operation

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effects of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.

The major contributors to this document are

William Decker (CSC)
Robert Hendrick (CSC)
Jon Valett (GSFC)

Single copies of this document can be obtained by writing to

Systems Development Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

This document captures over 50 individual SEL research results, extracted from a review of published SEL documentation, that can be applied directly to managing software development projects. Four basic categories of results are defined and discussed—environment profiles, relationships, models, and management rules. In each category, research results are presented as a single page that summarizes the individual result, lists potential uses of the result by managers, and references the original SEL documentation where the result was found. The document serves as a concise reference summary of applicable research for SEL managers.

Table of Contents

List of Tables

Table

SECTION 1 – INTRODUCTION

The Software Engineering Laboratory (SEL) was established in 1977 to support research in the measurement and evaluation of the software development process (Reference 1). The SEL is a cooperative effort of the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC), Computer Sciences Corporation (CSC), and the University of Maryland. Under its sponsorship, numerous experiments have been designed and executed to study the effects of applying various tools, methodologies, and models to the software development efforts in flight dynamics applications.

The SEL has been researching and evaluating software development methodologies for over 13 years. This research has provided valuable insight into the software development process of one particular organization. By collecting detailed software development data and recording that data in a software engineering data base (References 2 and 3), the SEL has been able to characterize and understand the development process within that organization. Many of the research results that have been published by the SEL over the years can be applied directly to managing software development projects in the SEL environment.

To help promote successful software development practices in this environment, the SEL recognized the potential of providing the experience of previous projects—the data, research results, and knowledge of experienced software managers—to the managers of ongoing projects. Research efforts were undertaken to determine if and how SEL experience and data could be effectively incorporated into an automated tool to provide insight into a project. Reference 4 describes an experimental, automated tool, the Software Management Environment (SME), currently in development.

As part of the tool development, the SEL conducted a thorough review of published SEL research. This review revealed several categories of data and findings that are fundamental to the construction and use of a tool to support the management of software development projects. These results are also useful to managers outside of an automated tool; in fact, many of the results are simply formal presentations of data used by managers on a regular basis.

This document contains a representative set of the research results extracted during the review of SEL documentation that can be used by software development managers. As a result, the document can serve as a concise reference summary of applicable research for SEL managers. (Each research result lists one of the source documents, References 5 through 14, where additional detailed information may be found.)

Although this research applies to projects within the SEL, other organizations may use the results as a starting point in their efforts to understand their own environments. The results provide a fundamental set of knowledge that illustrates the types of data an organization should assemble and suggests how to use the data.

The remainder of this document describes SEL research results organized by section into four basic categories: environment profiles, relationships, models, and management rules. Section 2 describes profiles containing typical values that characterize software development projects in the SEL environment. Section 3 describes relationships between various software development measures that are essential to successful planning and estimation activities. Section 4 describes models that capture the expected behavior of software development measures over the course of the development life cycle. Section 5 describes management rules that can help in diagnosing problems and evaluating the status of software development projects.

Each section first defines and describes the result type and then presents a series of representative samples of the result type. The results appear as single-page displays that include the following information: the result type, a text description of the result, a list of management activities to which the result can be applied, the SEL document reference, and a tabular and/or graphic representation of the result.

Table 1-1 presents a complete list of all single-page results presented in the document by title with the page number of the result. Table 1-2 contains a cross-reference of the research results grouped by specific factors of interest to managers. Note that individual results may appear several times in the list if the result relates to more than one factor.

Table 1-1. Index of Results (1 of 2)

Table 1-1. Index of Results (2 of 2)

Table 1-2. Research Results Grouped by Factor (1 of 3)

FACTOR	TYPE OF RESULT	DESCRIPTION OF RESEARCH RESULT	RESULT NUMBER
Changes	Profile	CHANGE RATE DURING IMPLEMENTATION	Result 2-2
	Model	SOFTWARE CHANGES PER LINE OF CODE BY PHASE	Result 4-9
	Model	SOFTWARE CHANGES PER COMPUTER RUN BY PHASE	Result 4-12
	Model	PROGRAMMER HOURS PER SOFTWARE CHANGE BY PHASE	Result 4-14
	Model	COMPUTER TIME PER SOFTWARE CHANGE BY PHASE	Result 4-15
	Model	TRENDS IN EFFORT TO CHANGE BY PHASE	Result 4-17
	Model	TRENDS IN CHANGES PER COMPUTER RUN BY PHASE	Result 4-18
	Model	TRENDS IN COMPUTER TIME PER CHANGE BY PHASE	Result 4-19
	Management Rule	DEVIATIONS IN CHANGES PER LINE OF CODE	Result 5-7
Computer Runs	Relationship	COMPUTER RUNS VS. LINES OF CODE	Result 3-9
	Relationship	COMPUTER RUNS VS. EFFORT	Result 3-10
	Model	COMPUTER RUNS PER LINE OF CODE BY PHASE	Result 4-8
	Model	PROGRAMMER HOURS PER COMPUTER RUN BY PHASE	Result 4-11
	Model	SOFTWARE CHANGES PER COMPUTER RUN BY PHASE	Result 4-12
	Model	COMPUTER TIME PER COMPUTER RUN BY PHASE	Result 4-13
	Model	TRENDS IN CHANGES PER COMPUTER RUN BY PHASE	Result 4-18
	Management Rule	VARIATIONS IN EFFORT TO CHANGE	Result 5-1
Computer Time	Relationship	COMPUTER TIME VS. LINES OF CODE	Result 3-11
	Model	COMPUTER USE BY PHASE	Result 4-5
	Model	COMPUTER TIME PER LINE OF CODE BY PHASE	Result 4-10
	Model	COMPUTER TIME PER COMPUTER RUN BY PHASE	Result 4-13
	Model	COMPUTER TIME PER SOFTWARE CHANGE BY PHASE	Result 4-15
	Model	TRENDS IN COMPUTER TIME PER CHANGE BY PHASE	Result 4-19
	Management Rule	HIGH COMPUTER USE	Result 5-4
	Management Rule	DEVIATIONS IN COMPUTER USE PER LINE OF CODE	Result 5-5
Documentation	Profile	PAGE DISTRIBUTION BY DOCUMENT TYPE	Result 2-7
	Profile	COST OF USER DOCUMENTATION	Result 2-8
	Relationship	DOCUMENTATION PAGES VS. LINES OF CODE	Result 3-12
	Relationship	DOCUMENTATION PAGES VS. MODULES	Result 3-13
	Relationship	DOCUMENTATION PAGES VS. EFFORT	Result 3-14
Duration	Profile	AVERAGE PROJECT DURATION	Result 2-1
	Relationship	DURATION VS. LINES OF CODE	Result 3-3
	Relationship	DURATION VS. MODULES	Result 3-4
	Relationship	DURATION VS. EFFORT	Result 3-5
	Relationship	SIZE, EFFORT, AND SCHEDULE BY PHASE	Result 3-15
Effort	Profile	AVERAGE PROJECT EFFORT	Result 2-1
	Profile	EFFORT DISTRIBUTION BY STAFF CATEGORY	Result 2-3
	Profile	EFFORT DISTRIBUTION BY ACTIVITY	Result 2-4
	Profile	ERROR DISTRIBUTION BY EFFORT TO CORRECT	Result 2-6
	Relationship	EFFORT VS. LINES OF CODE	Result 3-1
	Relationship	EFFORT VS. MODULES	Result 3-2
	Relationship	DURATION VS. EFFORT	Result 3-5
	Relationship	STAFF SIZE VS. EFFORT	Result 3-6
	Relationship	COMPUTER RUNS VS. EFFORT	Result 3-10
	Relationship	DOCUMENTATION PAGES VS. EFFORT	Result 3-14
	Relationship	SIZE, EFFORT, AND SCHEDULE BY PHASE	Result 3-15

Table 1-2. Research Results Grouped by Factor (2 of 3)

FACTOR	TYPE OF RESULT	DESCRIPTION OF RESEARCH RESULT	RESULT NUMBER
Effort (Cont'd)	Relationship	EFFORT ADJUSTMENT FOR COMPLEXITY	Result 3-16
	Relationship	EFFORT ADJUSTMENT FOR TEAM EXPERIENCE	Result 3-17
	Relationship	EFFORT ADJUSTMENT FOR SCHEDULE TYPE	Result 3-18
	Model	EFFORT BY PHASE	Result 4-2
	Model	EFFORT ACTIVITY BY PHASE	Result 4-3
	Model	PROGRAMMER HOURS PER LINE OF CODE BY PHASE	Result 4-7
	Model	PROGRAMMER HOURS PER COMPUTER RUN BY PHASE	Result 4-11
	Model	PROGRAMMER HOURS PER SOFTWARE CHANGE BY PHASE	Result 4-14
	Model	UNCERTAINTY IN EFFORT AND SIZE ESTIMATES BY PHASE	Result 4-16
	Model	TRENDS IN EFFORT TO CHANGE BY PHASE	Result 4-17
	Management Rule	VARIATIONS IN EFFORT TO CHANGE	Result 5-1
	Management Rule	VARIATIONS IN SIZE AND EFFORT ESTIMATES	Result 5-3
	Management Rule	DEVIATIONS IN LINES OF CODE PER STAFF HOUR	Result 5-8
Errors	Profile	ERROR RATE DURING IMPLEMENTATION	Result 2-2
	Profile	ERROR RATE DURING SYSTEM TESTING	Result 2-2
	Profile	ERROR RATE DURING ACCEPTANCE TESTING	Result 2-2
	Profile	ERROR RATE DURING MAINTENANCE/OPERATIONS	Result 2-2
	Profile	ERROR DISTRIBUTION BY ERROR CLASS	Result 2-5
	Profile	ERROR DISTRIBUTION BY EFFORT TO CORRECT	Result 2-6
	Model	ERROR RATE IN EACH PHASE	Result 4-6
Experience	Profile	AVERAGE APPLICATION EXPERIENCE OF MANAGERS	Result 2-1
	Profile	AVERAGE OVERALL EXPERIENCE OF MANAGERS	Result 2-1
	Profile	AVERAGE APPLICATION EXPERIENCE OF TECHNICAL STAFF	Result 2-1
	Profile	AVERAGE OVERALL EXPERIENCE OF TECHNICAL STAFF	Result 2-1
	Relationship	EFFORT ADJUSTMENT FOR TEAM EXPERIENCE	Result 3-17
Process	Profile	AVERAGE CODING RATE	Result 2-2
	Profile	NOMINAL MAINTAINABILITY INDICATORS	Result 2-2
	Profile	NOMINAL RELIABILITY INDICATORS	Result 2-2
	Relationship	PRODUCTIVITY VS. CODE REUSE	Result 3-7
	Relationship	PRODUCTIVITY VS. MODULE REUSE	Result 3-8
Product Size	Profile	AVERAGE LINES OF CODE DELIVERED	Result 2-1
	Relationship	EFFORT VS. LINES OF CODE	Result 3-1
	Relationship	EFFORT VS. MODULES	Result 3-2
	Relationship	DURATION VS. LINES OF CODE	Result 3-3
	Relationship	DURATION VS. MODULES	Result 3-4
	Relationship	COMPUTER RUNS VS. LINES OF CODE	Result 3-9
	Relationship	COMPUTER TIME VS. LINES OF CODE	Result 3-11
	Relationship	DOCUMENTATION PAGES VS. LINES OF CODE	Result 3-12
	Relationship	DOCUMENTATION PAGES VS. MODULES	Result 3-13
	Relationship	SIZE, EFFORT, AND SCHEDULE BY PHASE	Result 3-15
	Model	LINES OF CODE BY PHASE	Result 4-4
	Model	PROGRAMMER HOURS PER LINE OF CODE BY PHASE	Result 4-7
	Model	COMPUTER RUNS PER LINE OF CODE BY PHASE	Result 4-8
	Model	SOFTWARE CHANGES PER LINE OF CODE BY PHASE	Result 4-9
	Model	COMPUTER TIME PER LINE OF CODE BY PHASE	Result 4-10
	Model	UNCERTAINTY IN EFFORT AND SIZE ESTIMATES BY PHASE	Result 4-16

Table 1-2. Research Results Grouped by Factor (3 of 3)

FACTOR	TYPE OF RESULT	DESCRIPTION OF RESEARCH RESULT	RESULT NUMBER
Product Size (Cont'd)	Management Rule	VARIATIONS IN SIZE AND EFFORT ESTIMATES	Result 5-3
	Management Rule	DEVIATIONS IN COMPUTER USE PER LINE OF CODE	Result 5-5
	Management Rule	DEVIATIONS IN LINES OF CODE	Result 5-6
	Management Rule	DEVIATIONS IN CHANGES PER LINE OF CODE	Result 5-7
	Management Rule	DEVIATIONS IN LINES OF CODE PER STAFF HOUR	Result 5-8
Reuse	Profile	AVERAGE REUSE PERCENTAGE	Result 2-2
	Profile	COST OF REUSE	Result 2-9
	Relationship	PRODUCTIVITY VS. CODE REUSE	Result 3-7
	Relationship	PRODUCTIVITY VS. MODULE REUSE	Result 3-8
Schedule	Relationship	EFFORT ADJUSTMENT FOR SCHEDULE TYPE	Result 3-18
	Model	SCHEDULE BY PHASE	Result 4-1
Staffing	Profile	AVERAGE STAFF SIZE (FTE)	Result 2-1
	Profile	PEAK STAFF SIZE	Result 2-1
	Profile	TOTAL STAFF SIZE (INDIVIDUALS)	Result 2-1
	Profile	EFFORT DISTRIBUTION BY STAFF CATEGORY	Result 2-3
	Relationship	STAFF SIZE VS. EFFORT	Result 3-6
	Management Rule	DEVIATIONS IN STAFFING PLAN	Result 5-2

SECTION 2 – ENVIRONMENT PROFILES

The term "environment profile" refers to a SEL research result containing typical values that characterize software development in the SEL environment. These profiles describe what constitutes a normal project under typical conditions. They include a wide range of factors relevant to this environment and serve as a basis for better understanding the SEL development process.

The projects studied by the SEL generally involve the development of scientific, ground-based, interactive graphics software with moderate reliability and response requirements. Representative applications provide spacecraft support in the areas of attitude determination, attitude control, maneuver planning, orbit adjustment, and mission analysis.

An environment profile captures values that characterize one or more relevant aspects of the development process in the SEL environment. These profiles serve as a basic piece of management data by providing a standard for managers to use in evaluating and planning new projects. Managers also use profiles as a baseline for assessing the effects of improvement initiatives.

The following paragraphs describe three groups of environment profiles identified through SEL research efforts.

- <u>Characteristic values</u> describe nominal SEL values related to product, process, performance, and quality factors. These profiles provide a standard for determining if a project is representative of the environment with respect to overall project-specific features such as effort, size, duration, or staffing levels. They additionally provide a means for evaluating a project's performance and quality with regard to factors such as productivity, reliability, and maintainability.

- <u>Distribution profiles</u> describe the typical allocation of a key software development measure or resource classified according to its type. Examples of these profiles include distributions for effort by reported activity and for errors by effort to correct.

- <u>Cost profiles</u> describe heuristics that express typical values for costs that may reduce or increase normal expenditures from the basic development costs. Examples of these profiles include the cost of reuse and cost of user documentation.

The following pages present a selection of representative environment profiles published by the SEL.

PROCESS AND PRODUCT CHARACTERISTICS

Result 2-1

TYPE: Characteristic Values

DESCRIPTION:

The profile contains typical values related to process and product factors that are characteristic of software development in the SEL environment.

APPLICATION:

Comparison – Permits overall project-specific factors to be compared with past projects to determine if a project is representative of the environment.

Assessment – Serves as a basis for judging how typical or atypical a project is with respect to factors used in diagnosing problems (e.g., team is more experienced than normal, project size is smaller than normal).

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. A-1

REPRESENTATION:

Characteristics	Low	Average	High
Duration			
Time (months)	19	24	43
Cost			
Effort (staff-years)	3	14	32
Size			
Delivered code (KSLOC)	31	107	246
Staff			
Average staff (FTE)	4	8	15
Peak staff (FTE)	5	13	30
Individuals (total)	6	22	44
Application Experience			
Managers (years)	4	9	15
Technical staff (years)	2	4	7
Overall Experience			
Managers (years)	10	14	19
Technical staff (years)	4	6	9

NOTES: Type of software: *Scientific, ground-based, interactive graphic*
Machines: IBM 4341 and DEC VAX 780, 8600, and 8810
Sample: 10 FORTRAN (with 15% in Assembler) and 3 Ada projects
Staff-Year = 1864 effort hours

PERFORMANCE AND QUALITY CHARACTERISTICS

Result 2-2

TYPE: Characteristic Values

DESCRIPTION:

The profile contains typical values related to performance and quality factors that are characteristic of software development in the SEL environment.

APPLICATION:

Planning – Provides guidance in anticipating values for performance and quality measures that can be expected on a project.

Comparison – Permits software quality and performance factors to be compared with previous projects to measure the relative effect of improvement initiatives.

Assessment – Serves as a basis for evaluating an ongoing project with respect to performance and quality factors that relate to productivity, reliability, and maintainability.

Assessment – Provides indicators to help detect problems in the quality of software development efforts.

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. 3-3, Table 3-2, p. 6-8

Measures and Metrics for Software Development, SEL-83-002, pp. 4-12 through 4-22, Table 4-3

"Experiences in the Software Engineering Laboratory (SEL) – Applying Software Measurement," F. E. McGarry, S. R. Waligora, and T. P. McDermott, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, SEL-89-007, pp. 4, 5, 19

REPRESENTATION:

Characteristics	Nominal Values
Productivity	
Coding rate	3.3 developed SLOC per hour
Reuse percentage	30% of code is "reused"
Reliability	
Error rate	4 errors per developed KSLOC during implementation
Error rate	2 errors per developed KSLOC during system testing
Error rate	1 error per developed KSLOC during acceptance testing
Error rate	0.5 errors per developed KSLOC during maintenance/operations
Change rate	14 changes to components per developed KSLOC during implementation
Maintainability	
Effort to change	85% of all changes classified as "easy" or "very easy"
Effort to repair	85% of all repairs classified as "easy" or "very easy"

NOTES: Type of software: Scientific, ground-based, interactive graphic
Machines: IBM 4341 and DEC VAX 780, 8600, and 8810
Language: FORTRAN (with 15% in Assembler)

TYPE: Distribution Profile

DESCRIPTION:

The profile contains a distribution of total effort for the development process by staff category.

APPLICATION:

Planning – Provides a high-level cut at a staffing plan to identify typical management and support resources required.

Comparison – Permits overall staffing mix to be compared with past projects to determine if a project is representative of the environment.

Observation – When applied to an estimate of total effort, provides estimated completion values for tracking the expenditure of effort by staff category.

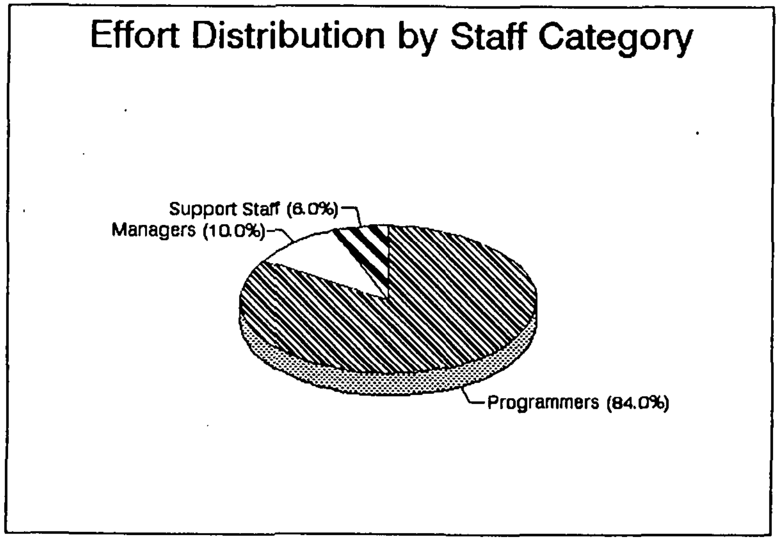
SOURCE:

An Approach to Software Cost Estimation, SEL-83-001, pp. 4-10, A-4

REPRESENTATION:

Staffing Category	% of Total Effort
Programmers	84
Managers	10
Support Staff	6

NOTE: *Support staff consists of administrative and publications support personnel.*



6199

EFFORT DISTRIBUTION BY ACTIVITY

Result 2–4

TYPE: Distribution Profile

DESCRIPTION:

The profile contains a distribution of the total hours of effort expended by reported activity for both FORTRAN and Ada projects.

APPLICATION:

Planning – Provides a high-level understanding of where programmers can be expected to expend effort.

Observation – When applied to an estimate of total effort, provides anticipated completion values for tracking the expenditure of effort by reported activity.

SOURCE:

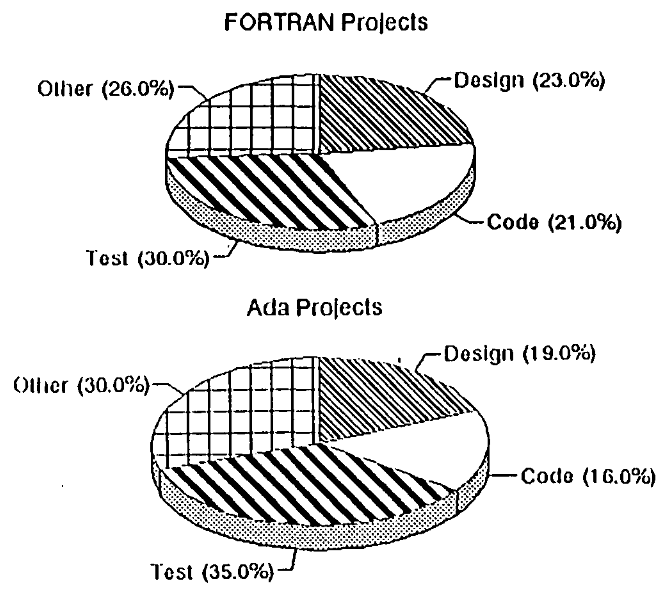
Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. 6-4

REPRESENTATION:

Reported Activity	% of Total Hours	
	FORTRAN	Ada
Design	23	19
Code	21	16
Test	30	35
Other	26	30

NOTE: Total hours expended are based on programmers providing weekly data attributing their time to the activity that they felt they were actually doing (no matter what development life-cycle phase they were in at the time).

Effort Distribution by Activity



6199

ERROR DISTRIBUTION BY ERROR CLASS

Result 2-5

TYPE: Distribution Profile

DESCRIPTION:

The profile contains a distribution of errors classified by type of reported error for both FORTRAN and Ada projects.

APPLICATION:

Planning – Provides a high-level understanding of the proportional number of errors expected to be encountered by reported class.

Observation – When applied to an estimate of total errors, provides anticipated completion values for tracking the number of reported errors uncovered to date in each error class.

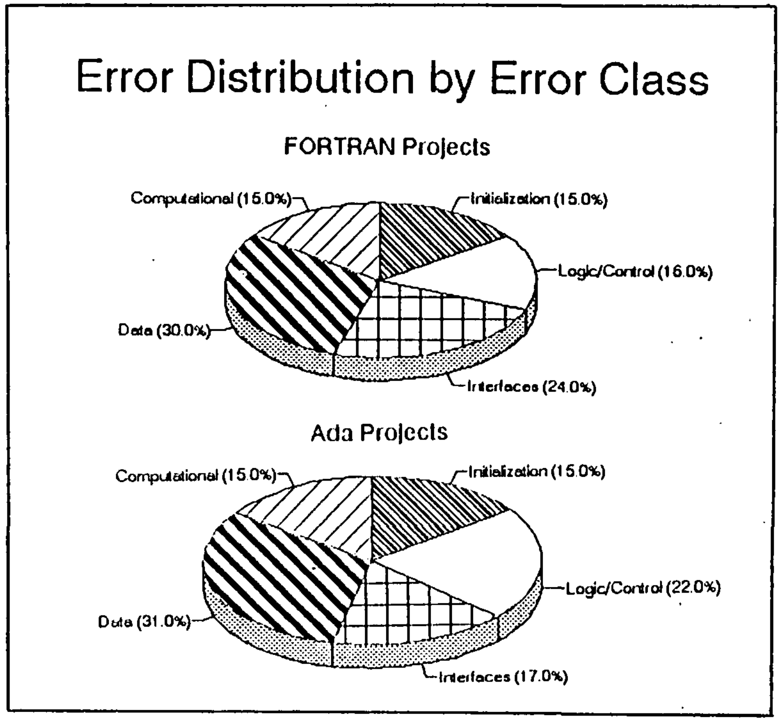
Assessment – Detects problems by observing deviations from the standard distribution in errors uncovered by reported class. (This usage assumes a distribution that is relatively constant over the project life-cycle.)

SOURCE:

"Experiences in the Software Engineering Laboratory (SEL) – Applying Software Measurement," F. E. McGarry, S. R. Waligora, and T. P. McDermott, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, SEL-89-007, pp. 4, 17

REPRESENTATION:

Error Class	% of Reported Errors	
	FORTRAN	Ada
Initialization	15	15
Logic/Control	16	22
Interfaces	24	17
Data	30	31
Computational	15	15



ERROR DISTRIBUTION BY EFFORT TO CORRECT

TYPE: Distribution Profile

DESCRIPTION:

The profile contains a distribution of errors classified by the effort required to correct the error.

APPLICATION:

Planning – Provides a high-level understanding of the amount of effort that can be expected to be required to correct errors.

Observation – When applied to an estimate of total errors, provides estimated completion values for tracking the number of reported errors by effort to correct.

Assessment – Provides an overall indicator for projecting the quality of the product being produced with respect to reliability, maintainability, and correctability.

Assessment – Detects problems by observing deviations from the standard distribution to judge relative difficulty encountered in correcting errors. (This usage assumes a distribution that is relatively constant over the project life-cycle.)

SOURCE:

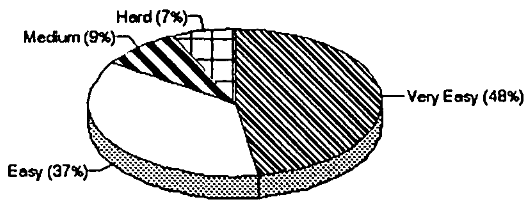
"What Have We Learned in 6 Years?", F. E. McGarry, *Proceedings of the Seventh Annual Software Engineering Workshop*, SEL-82-007, pp. 6, 24

REPRESENTATION:

Effort to Correct	% of Total
Very Easy	48
Easy	37
Medium	9
Hard	7

NOTE: The classifications defined for "effort to correct" reflect the actual effort expended, as reported by the programmer, when making a change. They do not reflect the subjective judgment of the programmer.

Error Distribution by Effort to Correct



6199

PAGE DISTRIBUTION BY DOCUMENT TYPE

Result 2–7

TYPE: Distribution Profile

DESCRIPTION:

The profile contains a distribution of page counts for project documentation classified by the type of document.

APPLICATION:

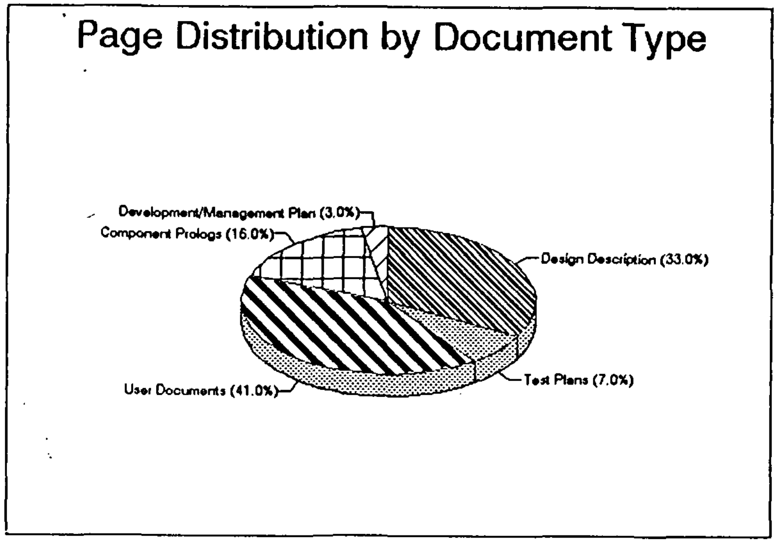
Planning – Provides a high-level understanding of the proportional number of pages expected to be needed for project documentation based on the type of document.

SOURCE:

An Approach to Software Cost Estimation, SEL-83-001, p. 3-12

REPRESENTATION:

Document Type	% of Total Effort
Design Description	33
Test Plans	7
User Documents	41
Component Prologs	16
Development/Management Plan	3



6199

COST OF USER DOCUMENTATION

Result 2–8

TYPE: Cost Profile

DESCRIPTION:

The profile describes additional documentation costs (over the basic development cost) based on the level of user documentation required.

NOTE: This result does not imply that documentation is unnecessary. The profile simply illustrates the relative cost associated with various documentation alternatives.

APPLICATION:

Planning – Provides guidance for estimating the cost associated with producing user documentation.

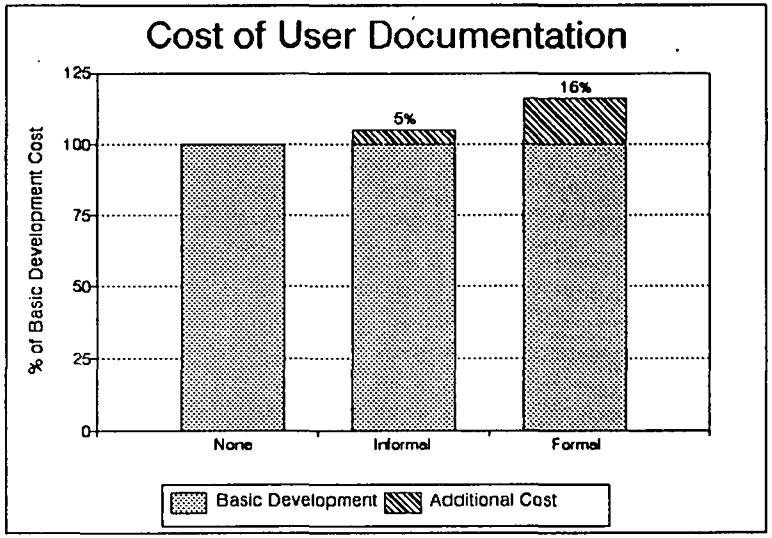
SOURCE:

An Approach to Software Cost Estimation, SEL-83-001, p. 3-12

REPRESENTATION:

Document Level	Additional Cost
No User Documents	0%
Informal User Documents	5%
Formal User Documents	16%

NOTE: Additional cost is expressed as a percentage of the basic development cost.



6199

TYPE: Cost Profile

DESCRIPTION:

The profile describes the cost of reusing a software module compared to the cost of developing a new module. This cost varies as a function of the extent of the modifications required to reuse the module.

APPLICATION:

Planning – Provides guidance for estimating the cost reduction that can be expected when software modules are reused with little or no change. (The number of developed modules, typically used as a basis for estimating costs, may be expressed as N + 0.20 * R, with N being new or extensively modified modules and R being slightly modified or old modules.)

SOURCE:

Manager's Handbook for Software Development (Revision1), SEL-84-101, p. 3-8, Table 3-8

An Approach to Software Cost Estimation, SEL-83-001, p. 3-17, Table 3-7

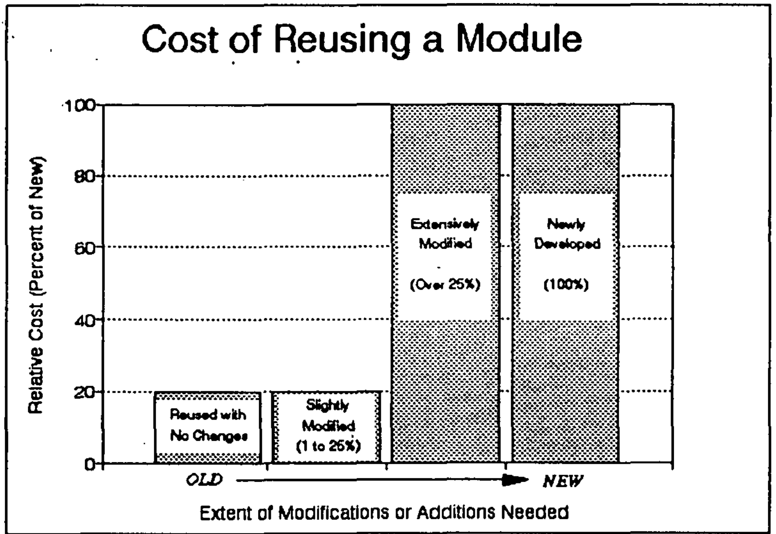
REPRESENTATION:

Module Classification	Percent of Code Modified or Added	Relative Cost
New	100	100
Extensively Modified	>25	100
Slightly Modified	1-25	20
Old	0	20

where

"Percent of Code" is the percentage of the module's code that must be modified or added for reuse.

"Relative Cost" is expressed as a percentage of the cost of developing a new module.



6199

SECTION 3 – RELATIONSHIPS

The term "relationship" refers to a SEL research result that describes the correlation between various software development measures at a specific point in the project life cycle. These relationships provide a method for projecting the values of unknown development measures and costs from information that is more readily available or more accurately known.

Extensive SEL research has been conducted to identify key software development measures in this environment and to quantify the relationships that exist between these measures. Since accurate estimation and planning are essential in successfully managing the development process, many SEL studies focus on estimating project completion values such as total effort, product size, and expected duration. These studies generally apply statistical analysis to historical project data to obtain useful relationships.

Managers employ these results to facilitate and standardize the planning and estimation process. Without a set of relationships derived from an understanding of the environment and from historical data, planning and estimation become largely guesswork. As a result, relationships between key development measures or costs are considered basic and necessary management data.

The following paragraphs describe three distinct groups of relationships identified through SEL research efforts.

- <u>General relationships</u> are time-independent equations that describe the correlation between key software development measures at project completion. These relationships provide a method for projecting a desired value based on the known or estimated values of other measures. For example, an equation that expresses total staff-hours as a function of lines of code (LOC) may be used to estimate the effort required for a project of a given size.

  When planning a project, experienced managers (or estimators) follow an established procedure that employs these relationships. Generally, this process involves (1) estimating the size of the software product, (2) converting the size estimate to an estimate of total effort, and (3) determining an expected duration and practical staffing level for completing the project.

  Since a set of relationships should be inherently consistent for the environment, a manager's use of relationships in this process implicitly detects and helps correct potential planning problems. For example, a relationship that expresses duration as a function of LOC may identify the impracticality of targeting the software delivery for a specified date without a reduction in scope or size.

- <u>Phase-dependent relationships</u> are time-independent equations, typically captured as a set of equations in a table, that apply to specific life-cycle phases during the project. They provide a method of reestimating the completion values of key software development measures for ongoing projects based on the most accurate and latest information available.

  As prescribed in the SEL environment, managers reestimate the values for key development measures at the end of each life-cycle phase. The basic values to be reestimated consist of effort, schedule duration, and project size. SEL research has identified an optimum set of relationships for each phase to accomplish this reestimation.

  These relationships comprise a system intended for use with specific data that are available when the relevant phase completes. For example, the most accurate measure of project size available through the requirements analysis phase may be the number of subsystems. After detailed design, when the system has been decomposed to a finer level, the most accurate measure of project size may be the number of modules. The discrete sets of relationships reflect the increasing granularity and decreasing uncertainty that occurs as the life-cycle progresses.

- <u>Adjustment factors</u> describe relationships expressing guidelines for determining multiplicative factors that may be applied to refine estimates. These factors account for differences in the problem, process, or environment that vary significantly from typical development conditions and that will increase or decrease nominal project expenditures.

  These relationships are represented in a tabular format that can be used to calculate the appropriate adjustment factor when atypical conditions arise. Estimates from other relationships, derived for nominal projects, are multiplied by the computed factor to revise the estimate upward or downward as needed.

  Examples of these relationships include guidelines for adjusting effort estimates to reflect problem complexity or development team experience.

The following pages present a selection of representative relationships published by the SEL.

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between total effort and source lines of code.

<u>NOTE</u>: This result may be counterintuitive since many popular equations used outside of the SEL environment have an exponent greater than 1.

APPLICATION:

Planning – Estimating the total effort required for a project based on a measure of the project size.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

Observation – Providing a target completion value for scaling effort models used in monitoring the expenditure of effort over the development life-cycle.

SOURCE:

The Software Engineering Laboratory: Relationship Equations, SEL-79-002, p. 13

RELATED RESULTS:

[1,2]"Finding Relationships Between Effort and Other Variables in the SEL," V. R. Basili and N. M. Panlilio-Yap, *Collected Software Engineering Papers: Volume III*, SEL-85-003, pp. 4-4, 4-5

REPRESENTATION:

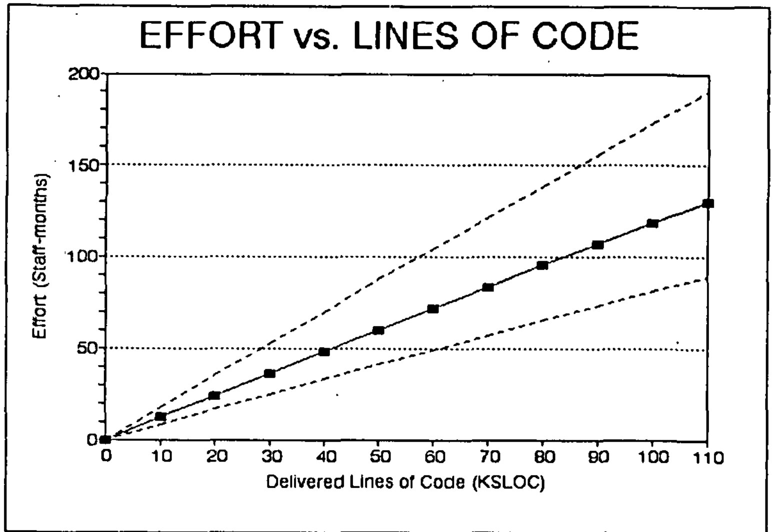
$$E = 1.266 * L^{0.986}$$

where

E is total effort in staff-months (with 172 hours per staff-month)

L is total delivered lines of code in KSLOC

<u>NOTE</u>: uncertainty = 0.456
$E_{upper\ bound} = E * (1.0 + uncertainty)$
$E_{lower\ bound} = E / (1.0 + uncertainty)$



[1] Describes alternative relationships, $E = 4.372 + 1.430 * devlines$ and $E = 5.497 + 1.500 * newlines$, that relate total effort to source lines of code in developed KSLOC and new KSLOC, respectively.

[2] Article also appears in *Proceedings of the Ninth International Computer Software and Applications Conference*, October 1985.

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between total effort and module count.

APPLICATION:

Planning – Estimating the total effort required for a project based on a measure of the project size.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

Observation – Providing a target completion value for scaling effort models used in monitoring the expenditure of effort over the development life-cycle.

SOURCE:

The Software Engineering Laboratory: Relationship Equations, SEL-79-002, p. 15

RELATED RESULTS:

[1]*The Software Engineering Laboratory: Relationship Equations*, SEL-79-002, p. 19

REPRESENTATION:

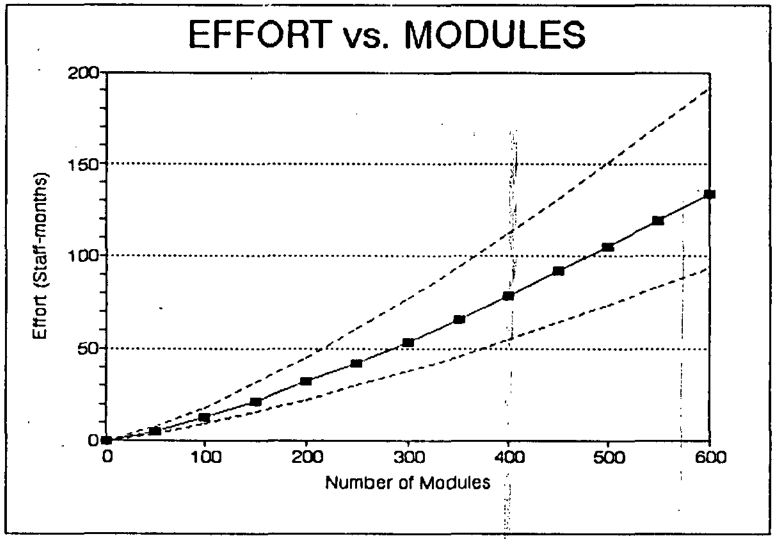
$$E = 0.029 * M^{1.319}$$

where

E is total effort in staff-months (with 172 hours per staff-month)

M is total number of modules in project (defined as any separately compilable entity)

NOTE: uncertainty = 0.431
$E_{upper\ bound} = E * (1.0 + uncertainty)$
$E_{lower\ bound} = E / (1.0 + uncertainty)$



[1] Describes an alternative relationship, $E = 0.052 * DM^{1.277}$, that relates total effort to the number of developed modules.

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between project duration and source lines of code.

APPLICATION:

Planning – Estimating the total time required to complete a project based on a measure of the project size.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

Observation – Providing a time scale for schedule models used to monitor software development measures such as effort and lines of code.

SOURCE:

The Software Engineering Laboratory: Relationship Equations, SEL-79-002, p. 32

RELATED RESULTS:

[1] *The Software Engineering Laboratory: Relationship Equations*, SEL-79-002, p. 36

REPRESENTATION:

$$D = 5.450 * L^{0.203}$$

where

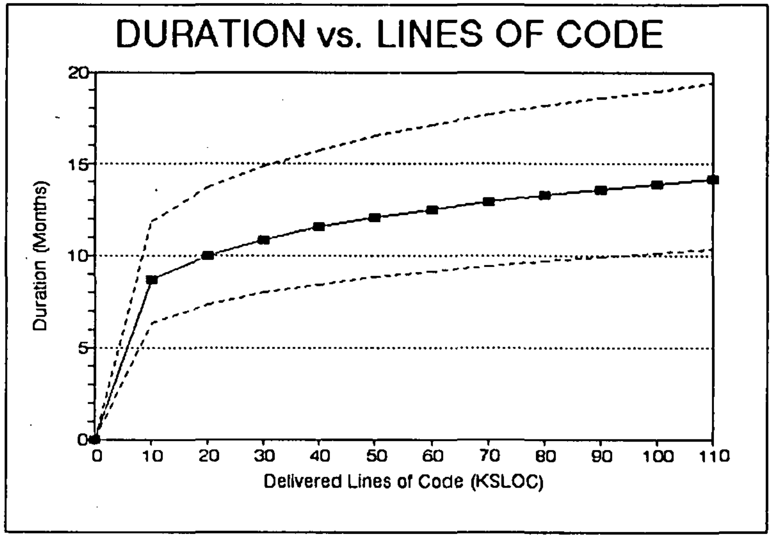
D is project duration in months (from start of project to end of acceptance testing)

L is total delivered lines of code in KSLOC

<u>NOTE</u>: uncertainty = 0.367

$D_{\text{upper bound}} = D * (1.0 + \text{uncertainty})$

$D_{\text{lower bound}} = D / (1.0 + \text{uncertainty})$



[1] Describes an alternative relationship, $D = 4.836 * DL^{0.257}$, that relates duration to source lines of code in developed KSLOC.

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between project duration and module count.

APPLICATION:

Planning – Estimating the total time required to complete a project based on a measure of the project size.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

Observation – Providing a time scale for schedule models used to monitor software development measures such as effort and lines of code.

SOURCE:

The Software Engineering Laboratory: Relationship Equations, SEL-79-002, p. 32

RELATED RESULTS:

[1]*The Software Engineering Laboratory: Relationship Equations*, SEL-79-002, p. 36

[1] Describes an alternative relationship, $D = 1.835 * DM^{0.346}$, that relates duration to the number of developed modules.

REPRESENTATION:

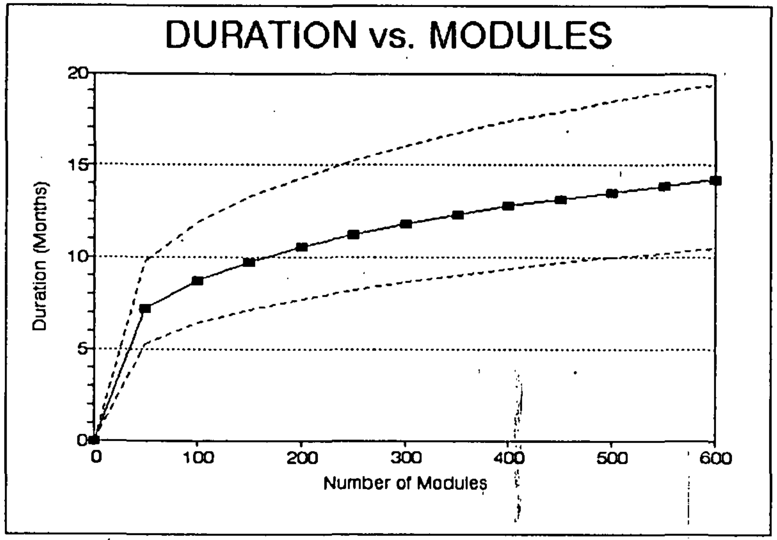
$$D = 2.453 * M^{0.275}$$

where

D is project duration in months (from start of project to end of acceptance testing)

M is total number of modules in project (defined as any separately compilable entity)

<u>NOTE</u>: uncertainty = 0.361
$D_{upper\ bound} = D * (1.0 + \text{uncertainty})$
$D_{lower\ bound} = D / (1.0 + \text{uncertainty})$



TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between project duration and total effort.

APPLICATION:

Planning – Estimating the total time required to complete a project based on a measure of total effort.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

Observation – Providing a time scale for schedule models used to monitor software development measures such as effort and lines of code.

SOURCE:

The Software Engineering Laboratory: Relationship Equations, SEL-79-002, p. 41

REPRESENTATION:

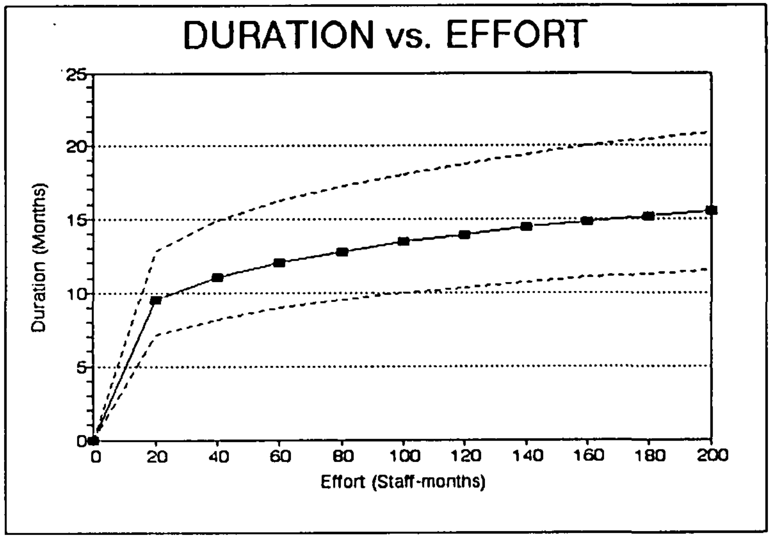
$$D \doteq 5.104 * E^{0.210}$$

where

D is project duration in months (from start of project to end of acceptance testing)

E is total effort in staff-months (with 172 hours per staff-month)

NOTE: uncertainty = 0.346
$D_{upper\ bound} = D * (1.0 + uncertainty)$
$D_{lower\ bound} = D / (1.0 + uncertainty)$



STAFF SIZE vs. EFFORT

Result 3-6

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between average staff size and total effort.

APPLICATION:

Planning – Estimating the average staff size required for a project based on a measure of total effort.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

SOURCE:

The Software Engineering Laboratory: Relationship Equations, SEL-79-002, p. 41

REPRESENTATION:

$$S = 0.198 * E^{0.786}$$

where

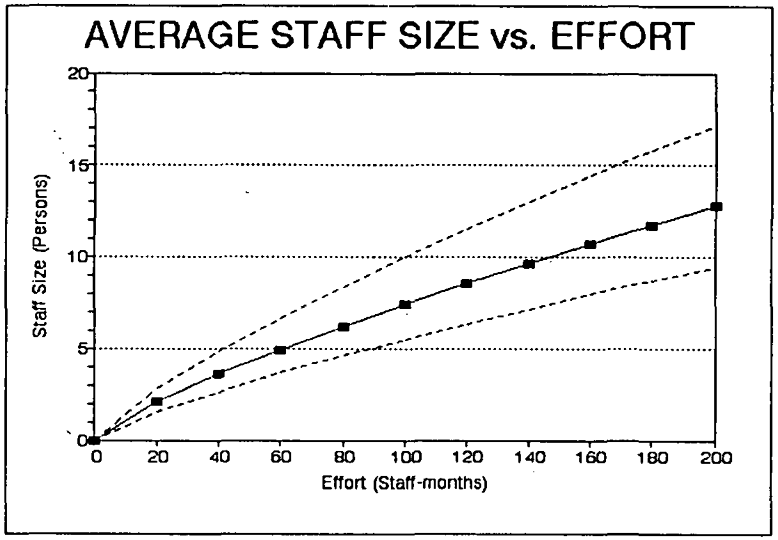
S is average staff size in staff-months per month (defined as total effort divided by duration or E/D)

E is total effort in staff-months (with 172 hours per staff-month)

<u>NOTE</u>: uncertainty = 0.347

$S_{upper\ bound} = S * (1.0 + uncertainty)$

$S_{lower\ bound} = S / (1.0 + uncertainty)$



6199

PRODUCTIVITY vs. CODE REUSE

Result 3-7

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between productivity and code reuse.

APPLICATION:

Planning – Estimating the productivity required for a project based on an expected value for code reuse.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

SOURCE:

The Software Engineering Laboratory: Relationship Equations, SEL-79-002, p. 21

REPRESENTATION:

$$P = 678.311 * (DL/L)^{-0.730}$$

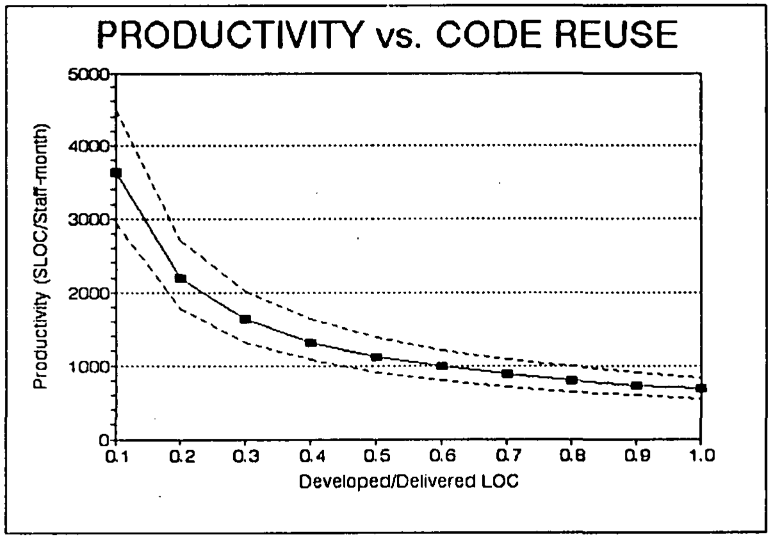
where

- **P** is productivity in delivered SLOC per staff-month (defined as total delivered SLOC divided by total effort, or L/E, with 172 hours per staff-month)
- **DL** is total developed lines of code in KSLOC (excludes reused code)
- **L** is total delivered lines of code in KSLOC

<u>NOTE</u>: uncertainty = 0.235

$P_{upper\ bound} = P * (1.0 + \text{uncertainty})$

$P_{lower\ bound} = P / (1.0 + \text{uncertainty})$



6199

PRODUCTIVITY vs. MODULE REUSE

Result 3-8

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between productivity and module reuse.

APPLICATION:

Planning – Estimating the productivity required for a project based on an expected value for module reuse.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

SOURCE:

The Software Engineering Laboratory: Relationship Equations, SEL-79-002, p. 21

REPRESENTATION:

$$P = 677.760 * (DM/M)^{-0.708}$$

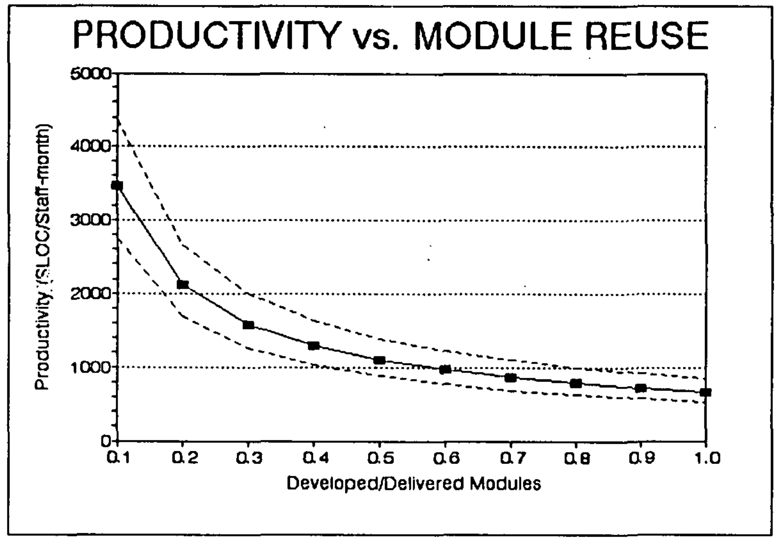
where

P is productivity in delivered SLOC per staff-month (defined as total delivered SLOC divided by total effort, or L/E, with 172 hours per staff-month)

DM is total number of developed modules in project (excludes reused modules)

M is total number of modules in project

NOTE: uncertainty = 0.258
$P_{upper\ bound} = P * (1.0 + uncertainty)$
$P_{lower\ bound} = P / (1.0 + uncertainty)$



6199

COMPUTER RUNS vs. LINES OF CODE

Result 3–9

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between number of computer runs and source lines of code.

APPLICATION:

Planning – Estimating the computer runs required for a project based on a measure of total effort.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

Observation – Providing a target completion value for scaling measure models used in monitoring the expenditure of computer resources over the development life-cycle.

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. 3-5

RELATED RESULTS:

[1,2] "Finding Relationships Between Effort and Other Variables in the SEL," V. R. Basili and N. M. Panlilio-Yap, *Collected Software Engineering Papers: Volume III*, SEL-85-003, p. 4-6

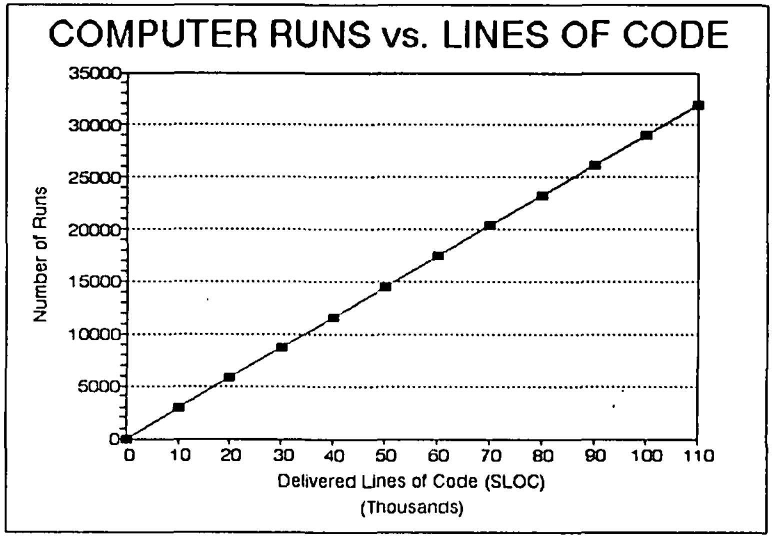
REPRESENTATION:

$$R = 0.29 * L$$

where

R is total number of computer runs

L is total delivered lines of code in SLOC



[1] Describes an alternative relationship, numruns = -108.274 + 150.879 * devlines, that relates the number of computer runs to source lines of code in developed KSLOC.

[2] Article also appears in *Proceedings of the Ninth International Computer Software and Applications Conference*, October 1985.

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between number of computer runs and total effort.

APPLICATION:

Planning – Estimating the computer runs required for a project based on a measure of total effort.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

Observation – Providing a target completion value for scaling measure models used in monitoring the expenditure of computer resources over the development life-cycle.

SOURCE:

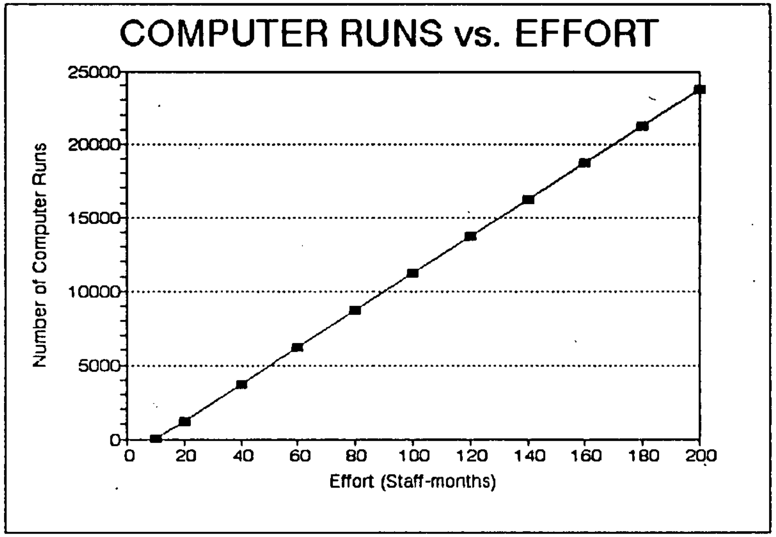
[1] "Finding Relationships Between Effort and Other Variables in the SEL," V. R. Basili and N. M. Panlilio-Yap, *Collected Software Engineering Papers: Volume III*, SEL-85-003, p. 4-5

REPRESENTATION:

$$E = 9.951 + 0.008 * numruns$$

where

E	is total effort in staff-months (with 160 hours per staff-month)
numruns	is total number of computer runs (includes every batch job submittal and every interactive terminal sign-on)



[1] Article also appears in *Proceedings of the Ninth International Computer Software and Applications Conference*, October 1985.

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between computer CPU time and source lines of code.

APPLICATION:

Planning – Estimating the computer CPU time required for a project based on a measure of the project size.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

Observation – Providing a target completion value for scaling measure models used in monitoring the expenditure of computer resources over the development life-cycle.

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. 3-5

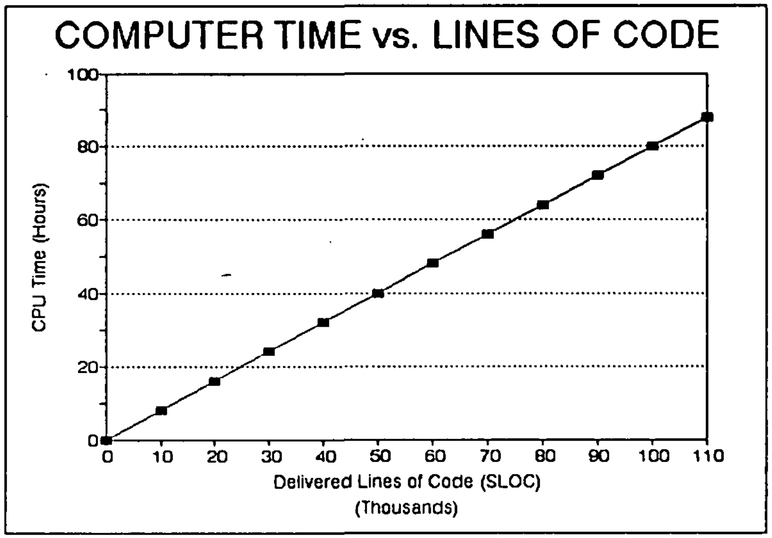
REPRESENTATION:

$$H = 0.0008 * L$$

where

H is total hours of computer CPU time

L is total delivered lines of code in SLOC



DOCUMENTATION PAGES vs. LINES OF CODE — Result 3–12

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between pages of documentation and source lines of code.

APPLICATION:

Planning – Estimating pages of documentation for a project based on a measure of the project size.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. 3-7

RELATED RESULTS:

[1]*An Approach to Software Cost Estimation*, SEL-83-001, p. 3-11

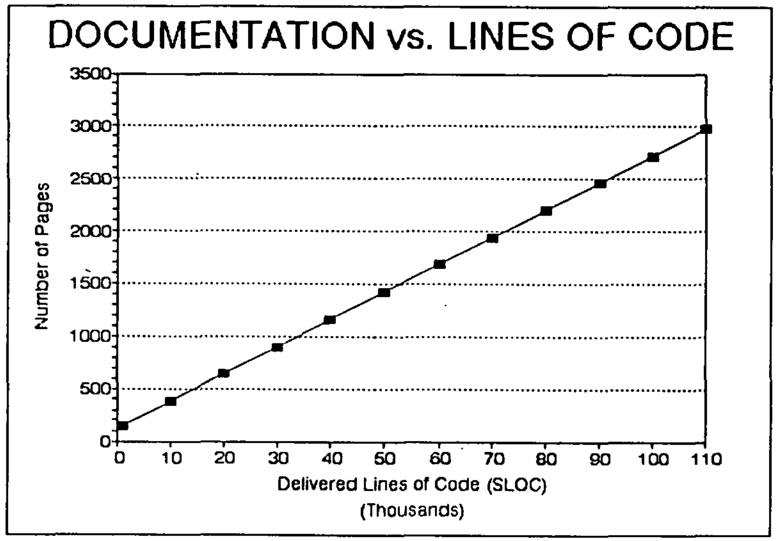
REPRESENTATION:

$$P = 120 + 0.026 * L$$

where

P is total pages of documentation (includes the requirements analysis report, design documents, system description, and user's guide)

L is total delivered lines of code in SLOC



[1] Describes an alternative relationship, P = 0.04 * L, that relates pages of documentation to source lines of code in developed SLOC.

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between pages of documentation and module count.

APPLICATION:

Planning – Estimating pages of documentation for a project based on a measure of the project size.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

SOURCE:

The Software Engineering Laboratory: Relationship Equations, SEL-79-002, p. 27

RELATED RESULTS:

[1]*The Software Engineering Laboratory: Relationship Equations*, SEL-79-002, p. 27

[1] Describes an alternative relationship, $DOC = 359.300 * DM^{0.279}$, that relates pages of documentation to the total number of developed modules.

REPRESENTATION:

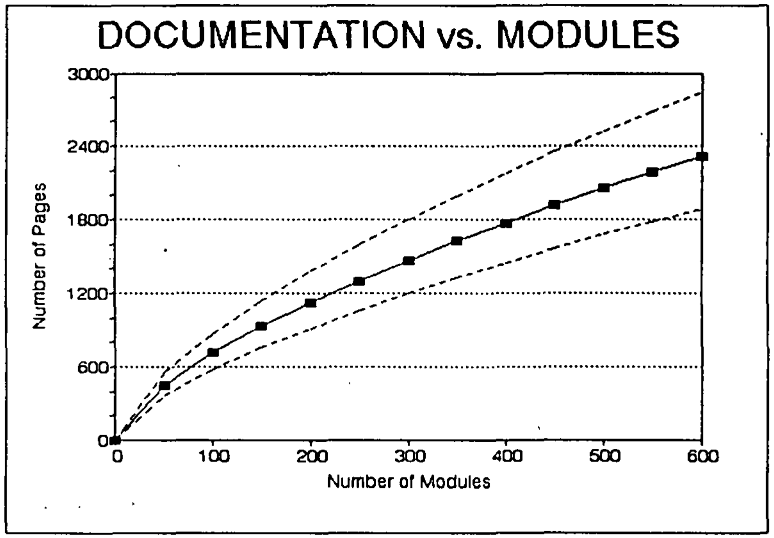
$$DOC = 33.585 * M^{0.662}$$

where

DOC is total pages of documentation (includes program design, test plans, user's guides, system description, and module description)

M is total number of modules in project (defined as any separately compilable entity)

NOTE: uncertainty = 0.227
$DOC_{upper\ bound} = DOC * (1.0 + uncertainty)$
$DOC_{lower\ bound} = DOC / (1.0 + uncertainty)$



DOCUMENTATION PAGES vs. EFFORT

Result 3–14

TYPE: General Relationship

DESCRIPTION:

The relationship describes the correlation between pages of documentation and total effort.

APPLICATION:

Planning – Estimating pages of documentation for a project based on a measure of total effort.

Planning – Validating values in a set of project estimates for consistency and reasonableness when used in conjunction with other relationships.

SOURCE:

[1] "Finding Relationships Between Effort and Other Variables in the SEL," V. R. Basili and N. M. Panlilio-Yap, *Collected Software Engineering Papers: Volume III*, SEL-85-003, p. 4-6

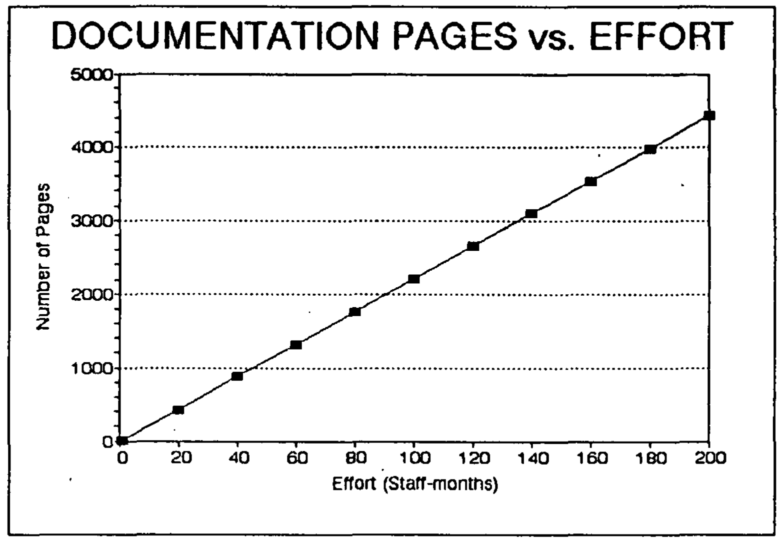
1 Article also appears in *Proceedings of the Ninth International Computer Software and Applications Conference*, October 1985.

REPRESENTATION:

$$E = 0.581 + 0.045 * docpages$$

where

E	is total effort in staff-months (with 160 hours per staff-month)
docpages	is total number of pages of documentation (includes program design document, development plan, test plans, system description, module descriptions, and user's guide)



SIZE, EFFORT, AND SCHEDULE BY PHASE

Result 3–15

TYPE: Phase-Dependent Relationships

DESCRIPTION:

The relationships describe a set of equations captured in a table that serve as a system for reestimating completion values for key software development measures at the end of each life-cycle phase.

APPLICATION:

Planning – Reestimating product size, total effort, and project duration based on specific data at recommended points in the life-cycle of ongoing projects.

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101 p. 3-3, Table 3-2

REPRESENTATION:

End of Phase	Relationship	Uncertainty
Requirements Analysis	$L = 11000 * SS$ $E = 3000 * SS$ $D = 83 * SS$	0.75
Preliminary Design	$L = 190 * M$ $E = 52 * M$ $D = 1.45 * M$	0.40
Detailed Design	$DL = 200 * DM$ $E = 0.31 * DL$ $D = 0.0087 * DL$	0.25
Implementation	$L = 1.26 * L_{Cur}$ $E = 1.43 * E_{Cur}$ $D = 1.54 * D_{Cur}$	0.10
System Testing	$E = 1.11 * E_{Cur}$ $D = 1.18 * D_{Cur}$	0.05

where

L	is total size in SLOC
E	is total effort in staff-hours
D	is total duration in weeks per staff member (based on a full-time employee's average work week with 1864 hours annually)
SS	is total number of subsystems in project
M	is total number of modules in project
DM	is total number of developed modules in project (defined as N + 0.2*R, with N being new modules and R being reused modules)
DL	is developed lines of code in SLOC
L_{Cur}	is size in SLOC through the current phase
E_{Cur}	is effort expended in staff-hours through the current phase
D_{Cur}	is schedule expended in calendar weeks through the current phase

NOTE: Uncertainty applies to effort or size estimates as follows:

$Estimate_{upper\ bound} = Estimate * (1.0 + uncertainty)$

$Estimate_{lower\ bound} = Estimate\ /\ (1.0 + uncertainty)$

EFFORT ADJUSTMENT FOR COMPLEXITY

TYPE: Adjustment Factor

DESCRIPTION:

The relationship provides a multiplicative factor for adjusting effort estimates based on complexity guidelines.

APPLICATION:

Planning – Permits an estimate of the total effort required for a project to be adjusted to account for complexity as determined by indicators of past experience with the problem and environment.

Planning – Classifies a project with respect to complexity based on the development team's experience with the application and the computing environment.

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. 3-4, Table 3-3

An Approach to Software Cost Estimation, SEL-83-001, p. 4-5, Table 4-2

REPRESENTATION:

Project	Environment	Multiplier
OLD	OLD	1.0
OLD	NEW	1.4
NEW	OLD	1.4
NEW	NEW	2.3

where

Project Type — is considered OLD when the development team has more than 2 years experience with the application area (e.g., orbit determination, simulator)

Environment Type — is considered OLD when the development team has more than 2 years experience with the computing environment (e.g., IBM 4341, VAX 8810)

EFFORT ADJUSTMENT FOR TEAM EXPERIENCE

Result 3-17

TYPE: Adjustment Factor

DESCRIPTION:

The relationship provides a multiplicative factor for adjusting effort estimates based on years of team experience.

APPLICATION:

Planning – Permits an estimate of the total effort required for a project to be adjusted to account for years of team experience with the application.

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. 3-4, Table 3-4

An Approach to Software Cost Estimation, SEL-83-001, p. 4-5, Table 4-3

REPRESENTATION:

Team Years of Application Experience	Effort Multiplier
10	0.5
8	0.6
6	0.8
4	1.0
2	1.4
1	2.6

where

Team Years is the average of all team member's years of application experience weighted by the member's participation on the team

with

Application experience is defined as prior work on similar applications (e.g., attitude and orbit determination).

Member's participation is defined as time spent working on the project as a proportion of total project effort.

EFFORT ADJUSTMENT FOR SCHEDULE TYPE

Result 3-18

TYPE: Adjustment Factor

DESCRIPTION:

The relationship provides a multiplicative factor for adjusting effort estimates based on schedule type.

APPLICATION:

Planning – Permits an estimate of the total effort required for a project to be adjusted to account for a faster or slower schedule than normally used in the environment.

Planning – Characterizes the type of schedule for a project based on an expected work rate and complexity classification.

SOURCE:

An Approach to Software Cost Estimation, SEL-83-001, pp. 4-4, 4-7, Tables 4-4 and 4-5

REPRESENTATION:

Schedule Characterization	Effort Multiplier
Fast	1.15
Average	1.00
Slow	0.85

where Schedule Characterization is determined from the following table:

Complexity Class	Derived Schedule Type for Work Rate		
	Fast	Average	Slow
OLD/OLD	> 0.24	0.24-0.16	< 0.16
OLD/NEW	> 0.17	0.17-0.10	< 0.10
NEW/OLD	> 0.17	0.17-0.10	< 0.10
NEW/NEW	> 0.11	0.11-0.07	< 0.07

with

Work Rate — calculated in developed KSLOC of executable code per calendar week

Complexity Class — expressed as a project/environment type pair (e.g., OLD/NEW)

(Project Type is OLD when the development team has more than 2 years experience with the application area; Environment Type is OLD when the development team has more than 2 years experience with the computing environment.)

SECTION 4 – MODELS

The term "model" refers to a SEL research result that describes the expected behavior of a software development measure as a function of time. For example, a research result containing a tabular listing of the fraction of errors detected during each development phase can be considered a "model of error detection." This type of research result has been described and applied in many SEL studies.

SEL research studies employ models to represent a "typical" project. The study results are commonly based on analyzing the comparison of some project of interest to the model. In almost all cases, the models have been obtained by averaging the behavior of the measure over several projects.

A model is a basic piece of management data. Models provide the standard or guidelines that managers use to judge the status of a project. By comparing the evolution of a measure to its expected behavior, the manager can assess a project's health and predict the measure's future behavior. Models of resource measures, such as effort or computer use, also can be used for planning.

The following paragraphs describe five groups of models.

- A <u>schedule model</u> describes how much time is allocated to each phase of the software development life cycle. In the SEL environment, most research results are obtained from projects in the requirements analysis through acceptance testing life-cycle phases.

  A schedule model is combined with each of the other types of models to provide a time scale for depicting the "typical" project in the environment.

- A <u>basic measure model</u> describes the behavior over time of a fundamental software development measure such as LOC, effort, or software errors.

  A manager uses basic measure models for tracking those software development measures that are directly related to the magnitude of the specific project being monitored. An estimate of the completion value of the measure for the project (see Section 3) is required since each model expresses a measure's behavior on a normalized scale.

- An <u>environment model</u> describes the behavior over time of a ratio of two basic software development measures such as computer resource usage per job, LOC per staff hour, or errors per LOC.

  Environment models are useful when a manager tracks a project's behavior as it compares to other projects in the environment. The effects of project size are minimized through the use of ratios. These models are characteristic of the environment and not a single project; therefore, they are not normalized but are expressed in absolute units.

- An **uncertainty model** describes the changing confidence interval for resource estimates over time. As a software development project evolves through specification, design, and implementation, more information becomes available about the eventual size of the completed project. As the project evolves, the manager's estimates of the resources required to complete it become more certain. The model describes the manner in which the uncertainty decreases.

- A **subjective model** is a description of specific features of a software development measure's probable behavior over time. The development measures are either basic measures or ratios of basic measures. SEL research describes these models with text and in some cases they are illustrated by a sketch. Subjective models are based on the experiences of managers in a particular software development environment. Subjective models are not quantitative; they describe the characteristics of a measure's behavior using words such as "constant," "increasing," or "starts to decrease."

  Subjective models describe significant features of the behavior of measures that signal to the manager when a development process is working correctly or incorrectly. In SEL literature, subjective models are often paired with rules that indicate problems. For example, a subjective model that describes a measure's behavior as "constant in magnitude" is another view of a rule that states that a rapidly changing measure is an indicator of a problem.

The following pages describe a selection of models published by the SEL.

SCHEDULE BY PHASE

Result 4–1

TYPE: Schedule Model

DESCRIPTION:

The model identifies the amount of time normally allocated to each life-cycle phase as a percentage of total project duration.

APPLICATION:

Planning – Provides guidance in selecting dates for major reviews and milestones.

Comparison – When used with an estimate of project duration, provides a time scale for depicting typical projects in the environment.

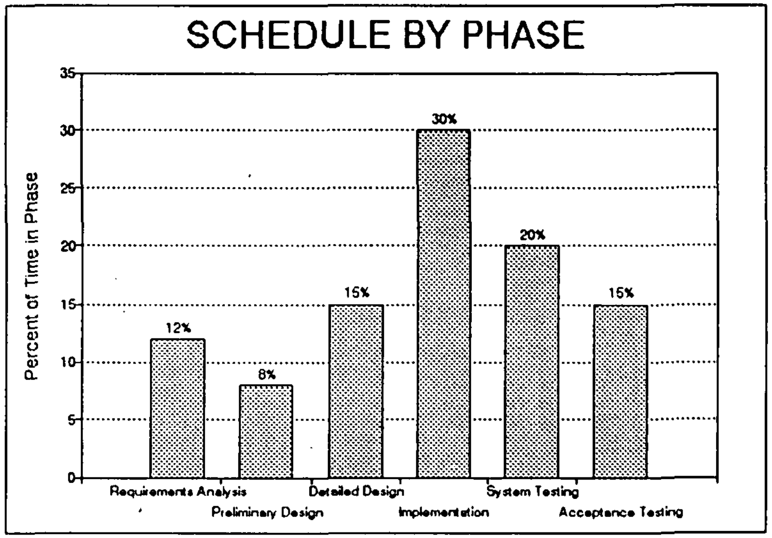
Prediction – Extrapolating the schedule of an ongoing project into the future given the time required to complete past life-cycle phases.

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. 3-1

REPRESENTATION:

Life-Cycle Phase	Percent of Time in Phase
Requirements Analysis	12
Preliminary Design	8
Detailed Design	15
Implementation	30
System Testing	20
Acceptance Testing	15



6199

EFFORT BY PHASE — Result 4–2

TYPE: Basic Measure Model

DESCRIPTION:

The model identifies the effort normally allocated in each life-cycle phase as a percentage of total effort.

APPLICATION:

Planning – Provides guidance in allocating effort to phases of the development life-cycle.

Prediction – Extrapolating the effort required in future phases based on the known effort expended in past phases.

Analysis – Estimating the fraction of the current phase that is elapsed based on the fraction of the total estimated effort that has been expended to date.

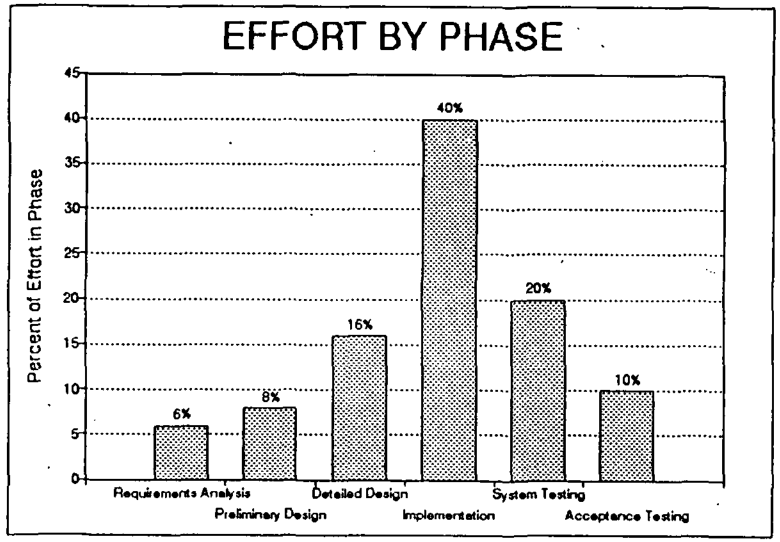
Assessment – Detecting problems by observing deviations of actual staff-hours from guidelines based on the model.

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. 3-1

REPRESENTATION:

Life-Cycle Phase	Percent of Time in Phase
Requirements Analysis	6
Preliminary Design	8
Detailed Design	16
Implementation	40
System Testing	20
Acceptance Testing	10



TYPE: Basic Measure Model

DESCRIPTION:

The model illustrates how effort expended on specific activities is distributed throughout the development life-cycle.

APPLICATION:

Planning – Providing guidance in allocating effort to phases of the development life-cycle.

Comparison – Providing a standard breakdown of effort by activity based on previous projects for comparison with current projects.

Assessment – Detecting problems by observing deviations from the standard activity breakdown.

SOURCE:

[1] "The Effectiveness of Software Prototyping: A Case Study," M. V. Zelkowitz, *Collected Software Engineering Papers: Volume VI*, SEL-88-002, pp. 2-6 through 2-8

REPRESENTATION:

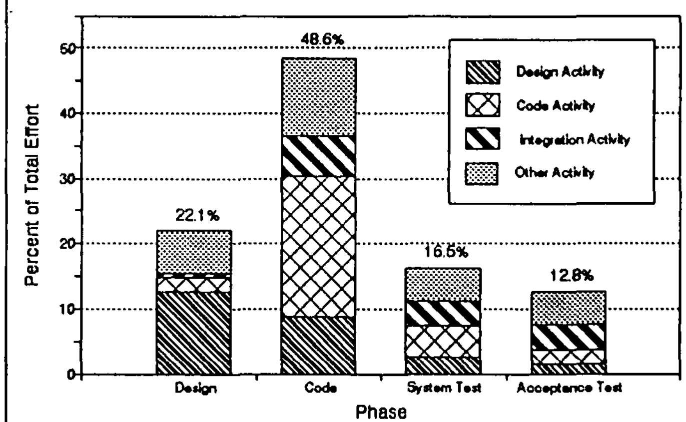
Percentage of Activity in Each Phase

Phase	Design Activity	Code Activity	Integration Activity	Other Activity
Design	49.2	6.9	4.7	23.1
Code	34.1	70.3	43.4	41.2
System Test	10.3	15.9	26.1	17.8
Acceptance Test	6.4	6.9	25.8	17.9
	100.0	100.0	100.0	100.0

Percentage of Total Effort for an Activity by Phase

Phase	Design Activity	Code Activity	Integration Activity	Other Activity
Design	12.6	2.1	0.7	6.7
Code	8.7	21.4	6.5	11.9
System Test	2.6	4.8	3.9	5.1
Acceptance Test	1.6	2.1	3.9	5.2
	25.5	30.4	15.0	28.9

Activity by Phase



1 Article also appears in *Proceedings of the 26th Annual Technical Symposium of the Washington, D.C. Chapter of the ACM*, June 1987.

TYPE: Basic Measure Model

DESCRIPTION:

The model identifies the growth pattern of source code produced by a development project.

APPLICATION:

Prediction – Extrapolating the code to be produced in future phases given the known size of code produced in past phases.

Analysis – Estimating the fraction of the current phase that is elapsed based on the fraction of the total estimated code size that has been produced to date.

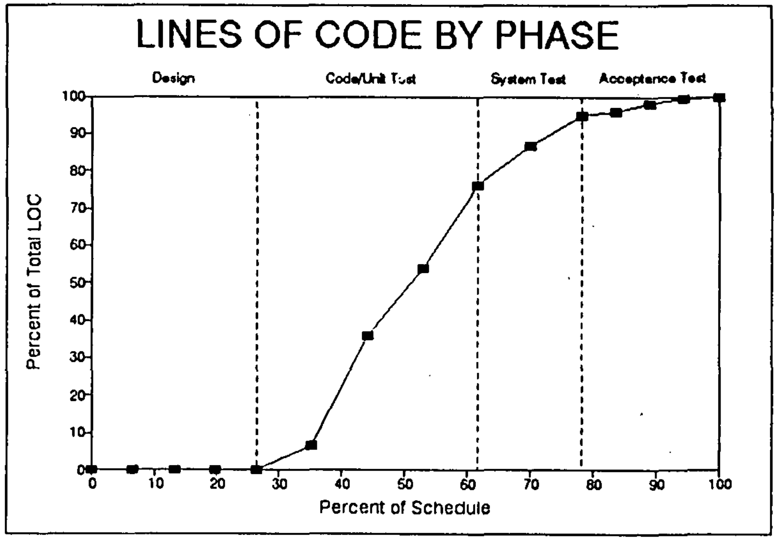
Assessment – Detecting problems by observing deviations of actual code size from guidelines based on the model.

SOURCE:

"Experiences in the Software Engineering Laboratory (SEL) – Applying Software Measurement," F. E. McGarry, S. R. Waligora, and T. P. McDermott, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, SEL-89-007, pp. 4, 18

REPRESENTATION:

Phase	% of Phase	% of Total Lines
Design (Start)	0	0.00
	25	0.00
	50	0.00
	75	0.00
Code/Unit Test (Start)	0	0.00
	25	6.86
	50	36.05
	75	53.99
System Test (Start)	0	76.28
	50	86.82
Acceptance Test (Start)	0	94.88
	25	96.09
	50	98.14
	75	99.58
End	100	100.00



6199

TYPE: Basic Measure Model

DESCRIPTION:

The model illustrates how the demand for computer time varies over the development life-cycle for FORTRAN and Ada projects. The amount of computer utilization (CPU time) is expressed as a percentage of the average weekly use for the entire project.

<u>NOTE</u>: The maximum value attained should not exceed three times the weekly average.

APPLICATION:

Planning-Provides guidance in anticipating computer resources required at various points in the development life-cycle.

Prediction – Extrapolating the total computer time expected to be used from the known amount of computer use to date.

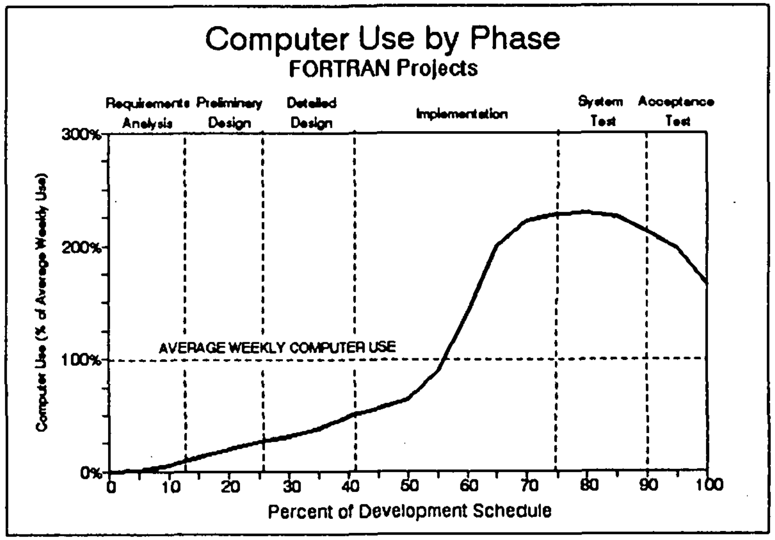
Analysis – Estimating the fraction of the current phase that is elapsed based on the fraction of the total estimated computer time that has been used to date.

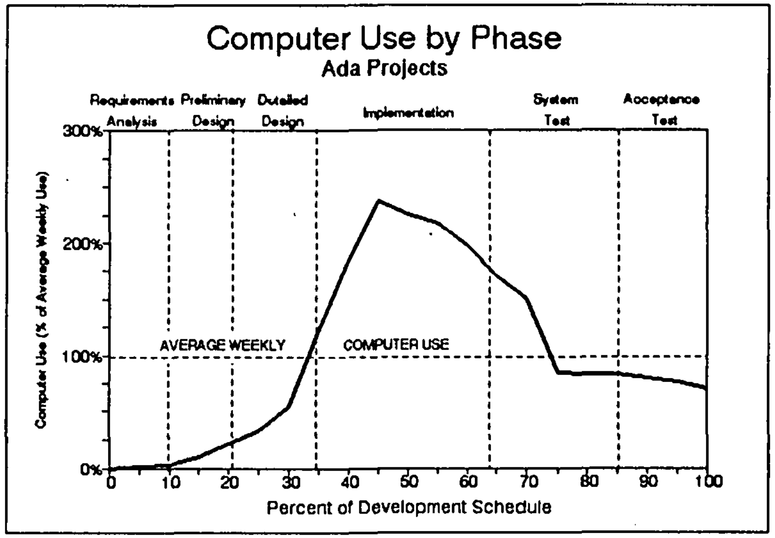
Assessment – Detecting problems by observing deviations of actual computer time used from guidelines based on the model.

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. 3-6

REPRESENTATION:





ERROR RATE IN EACH PHASE

Result 4–6

TYPE: Environment Model

DESCRIPTION:

The model describes the behavior of the error rate that can be expected in each life-cycle phase. The error rate is expressed as a ratio of the number of errors detected in the phase to the total estimated size of the project in developed lines of source code.

APPLICATION:

Assessment – Provides a characteristic pattern of behavior for the error rate that can be compared to actual measurements to help judge the relative reliability of the software being produced.

Assessment – Detecting problems by observing deviations of the error rate from values given by the model. (Failure of the error rate to decline during testing may suggest that many undiscovered errors remain in the software.)

SOURCE:

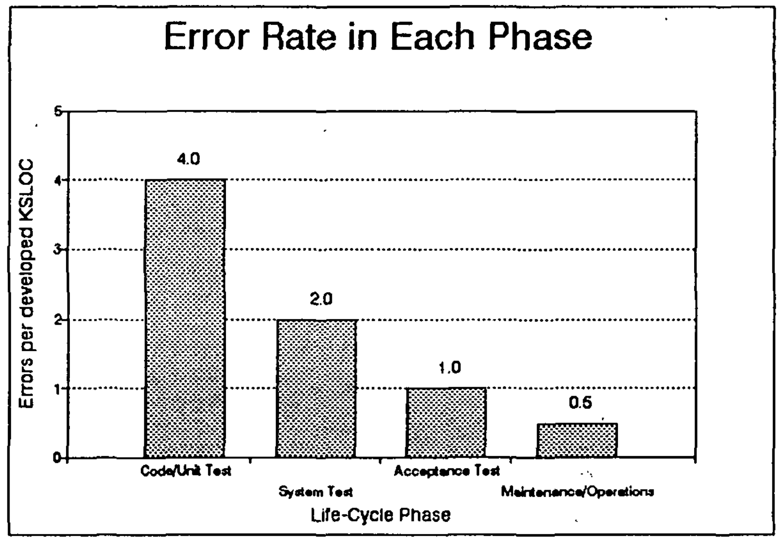
"Experiences in the Software Engineering Laboratory (SEL) – Applying Software Measurement," F. E. McGarry, S. R. Waligora, and T. P. McDermott, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, SEL-89-007, pp. 4, 5, 19

Manager's Handbook for Software Development (Revision 1), SEL-84-101, p. 6-8

REPRESENTATION:

Phase	Errors Detected Per Developed KSLOC
Code/Unit Test	4
System Test	2
Acceptance Test	1
Maintenance/Operations	0.5

<u>NOTE</u>: This implies that a 50% reduction in the error detection rate for a given phase is expected for each subsequent phase.



6199

PROGRAMMER HOURS PER LINE OF CODE BY PHASE

Result 4–7

TYPE: Environment Model

DESCRIPTION:

The model describes the behavior of the cumulative ratio of programmer hours per line of source code.

APPLICATION:

Prediction – Extrapolating the final value of the ratio based on the current observed value and the values given by the model.

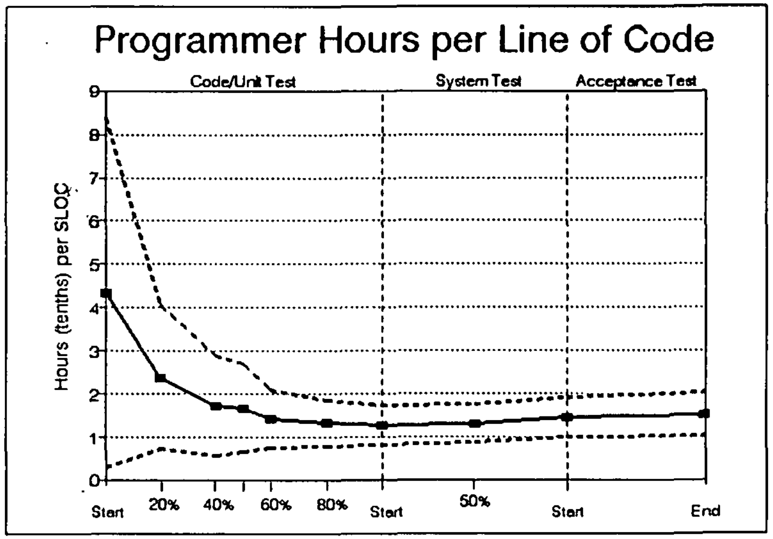
Assessment – Detecting problems by observing deviations of the ratio from values given by the model.

SOURCE:

Monitoring Software Development Through Dynamic Variables (Revision 1), SEL-83-106, p. 47

REPRESENTATION:

Phase	Hours (in tenths) per Line of Code	Standard Deviation
Start coding	4.340	4.037
20% coding	2.379	1.673
40% coding	1.718	1.155
50% coding	1.683	1.026
60% coding	1.426	0.664
80% coding	1.324	0.539
Start system testing	1.260	0.443
50% system testing	1.312	0.447
Start acceptance testing	1.456	0.459
End acceptance testing	1.531	0.505



6199

COMPUTER RUNS PER LINE OF CODE BY PHASE

Result 4-8

TYPE: Environment Model

DESCRIPTION:

The model describes the behavior of the cumulative ratio of computer runs per line of source code.

APPLICATION:

Prediction – Extrapolating the final value of the ratio based on the current observed value and the values given by the model.

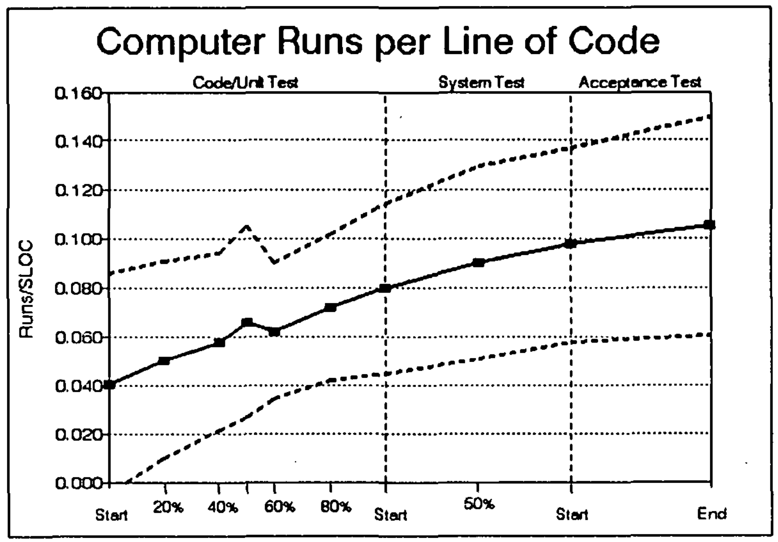
Assessment – Detecting problems by observing deviations of the ratio from values given by the model.

SOURCE:

Monitoring Software Development Through Dynamic Variables (Revision 1), SEL-83-106, p. 48

REPRESENTATION:

Phase	Computer Runs per Line of Code	Standard Deviation
Start coding	0.0401	0.0454
20% coding	0.0503	0.0402
40% coding	0.0575	0.0362
50% coding	0.0659	0.0392
60% coding	0.0622	0.0281
80% coding	0.0716	0.0298
Start system testing	0.0795	0.0350
50% system testing	0.0899	0.0393
Start acceptance testing	0.0972	0.0397
End acceptance testing	0.1050	0.0444



6199

SOFTWARE CHANGES PER LINE OF CODE BY PHASE

TYPE: Environment Model

DESCRIPTION:

The model describes the behavior of the cumulative ratio of software changes per line of source code.

APPLICATION:

Prediction – Extrapolating the final value of the ratio based on the current observed value and the values given by the model.

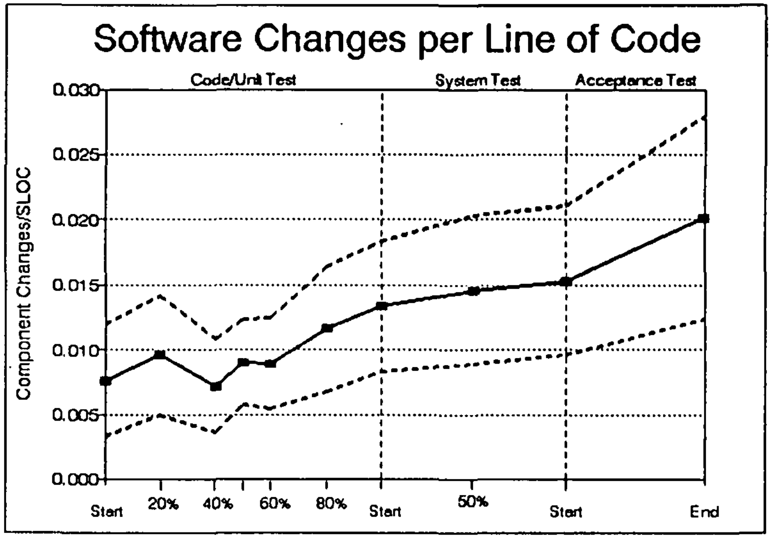
Assessment – Detecting problems by observing deviations of the ratio from values given by the model.

SOURCE:

Monitoring Software Development Through Dynamic Variables (Revision 1), SEL-83-106, p. 49

REPRESENTATION:

Phase	Software Changes per Line of Code	Standard Deviation
Start coding	0.0076	0.0043
20% coding	0.0096	0.0046
40% coding	0.0072	0.0036
50% coding	0.0090	0.0033
60% coding	0.0089	0.0035
80% coding	0.0116	0.0048
Start system testing	0.0133	0.0050
50% system testing	0.0146	0.0057
Start acceptance testing	0.0153	0.0057
End acceptance testing	0.0201	0.0078



6199

COMPUTER TIME PER LINE OF CODE BY PHASE

Result 4–10

TYPE: Environment Model

DESCRIPTION:

The model describes the behavior of the cumulative ratio of computer time per line of source code.

APPLICATION:

Prediction – Extrapolating the final value of the ratio based on the current observed value and the values given by the model.

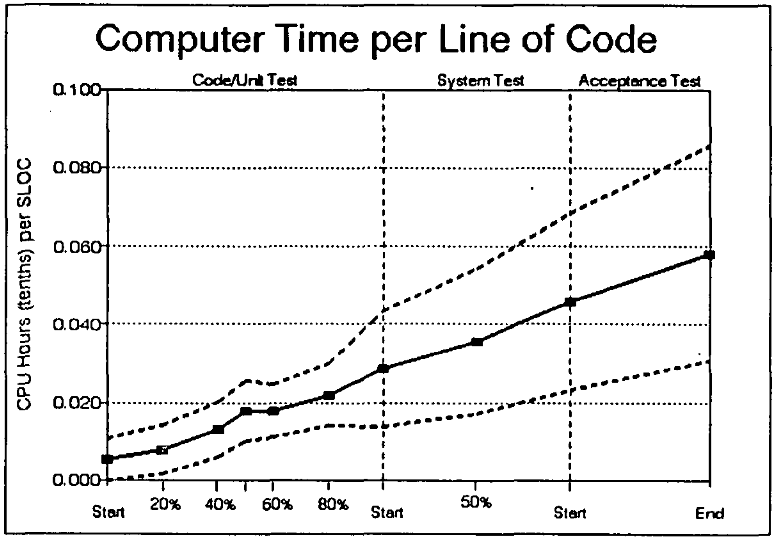
Assessment – Detecting problems by observing deviations of the ratio from values given by the model.

SOURCE:

Monitoring Software Development Through Dynamic Variables (Revision 1), SEL-83-106, p. 50

REPRESENTATION:

Phase	CPU Hours (tenths) per Line of Code	Standard Deviation
Start coding	0.0053	0.0054
20% coding	0.0077	0.0062
40% coding	0.0130	0.0071
50% coding	0.0178	0.0079
60% coding	0.0178	0.0068
80% coding	0.0220	0.0080
Start system testing	0.0287	0.0150
50% system testing	0.0357	0.0185
Start acceptance testing	0.0458	0.0226
End acceptance testing	0.0580	0.0275



6199

PROGRAMMER HOURS PER COMPUTER RUN BY PHASE

Result 4–11

TYPE: Environment Model

DESCRIPTION:

The model describes the behavior of the cumulative ratio of programmer hours per computer run.

APPLICATION:

Prediction – Extrapolating the final value of the ratio based on the current observed value and the values given by the model.

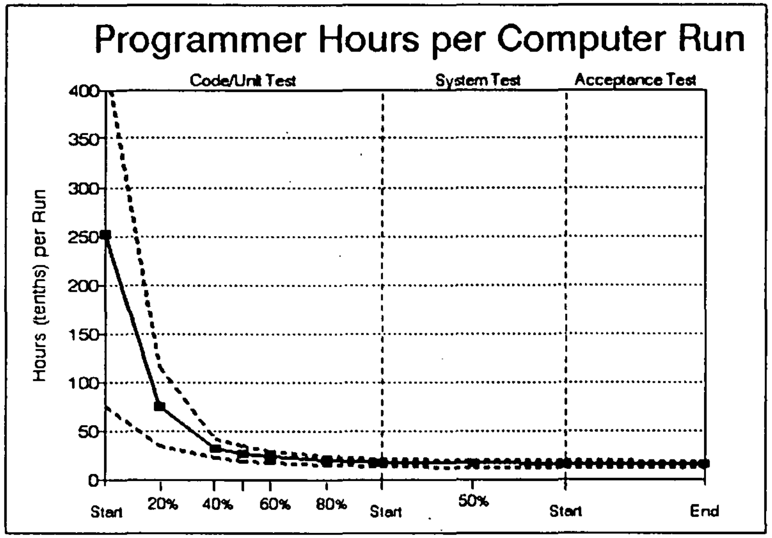
Assessment – Detecting problems by observing deviations of the ratio from values given by the model.

SOURCE:

Monitoring Software Development Through Dynamic Variables (Revision 1), SEL-83-106, p. 51

REPRESENTATION:

Phase	Hours (in tenths) per Computer Run	Standard Deviation
Start coding	251.82	176.62
20% coding	75.26	40.18
40% coding	31.96	9.40
50% coding	26.71	7.97
60% coding	23.59	6.37
80% coding	19.35	5.07
Start system testing	17.46	4.32
50% system testing	16.41	3.94
Start acceptance testing	15.87	3.68
End acceptance testing	15.45	3.59



6199

SOFTWARE CHANGES PER COMPUTER RUN BY PHASE

Result 4–12

TYPE: Environment Model

DESCRIPTION:

The model describes the behavior of the cumulative ratio of software changes per computer run.

APPLICATION:

Prediction – Extrapolating the final value of the ratio based on the current observed value and the values given by the model.

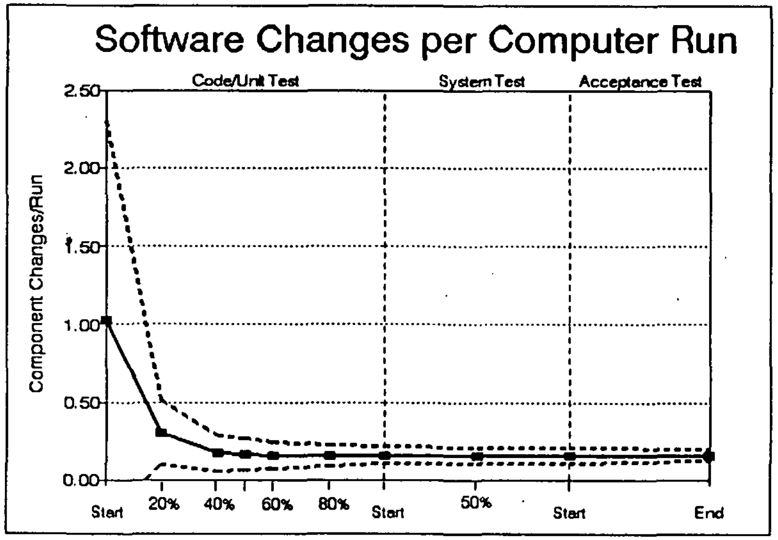
Assessment – Detecting problems by observing deviations of the ratio from values given by the model.

SOURCE:

Monitoring Software Development Through Dynamic Variables (Revision 1), SEL-83-106, p. 52

REPRESENTATION:

Phase	Software Changes per Computer Run	Standard Deviation
Start coding	1.0240	1.2750
20% coding	0.3052	0.2073
40% coding	0.1714	0.1140
50% coding	0.1670	0.0999
60% coding	0.1583	0.0863
80% coding	0.1588	0.0691
Start system testing	0.1604	0.0561
50% system testing	0.1552	0.0492
Start acceptance testing	0.1563	0.0476
End acceptance testing	0.1611	0.0386



6199

COMPUTER TIME PER COMPUTER RUN BY PHASE

Result 4–13

TYPE: Environment Model

DESCRIPTION:

The model describes the behavior of the cumulative ratio of computer time per computer run.

APPLICATION:

Prediction – Extrapolating the final value of the ratio based on the current observed value and the values given by the model.

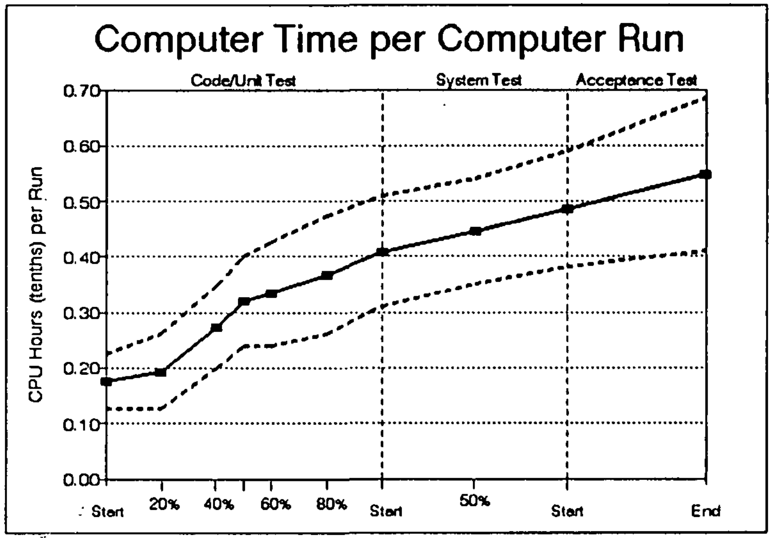
Assessment – Detecting problems by observing deviations of the ratio from values given by the model.

SOURCE:

Monitoring Software Development Through Dynamic Variables (Revision 1), SEL-83-106, p. 53

REPRESENTATION:

Phase	CPU Hours (tenths) per Computer Run	Standard Deviation
Start coding	0.1760	0.0498
20% coding	0.1944	0.0686
40% coding	0.2740	0.0740
50% coding	0.3197	0.0808
60% coding	0.3333	0.0935
80% coding	0.3681	0.1061
Start system testing	0.4095	0.0990
50% system testing	0.4453	0.0945
Start acceptance testing	0.4857	0.1029
End acceptance testing	0.5472	0.1371



6199

PROGRAMMER HOURS PER SOFTWARE CHANGE BY PHASE

Result 4–14

TYPE: Environment Model

DESCRIPTION:

The model describes the behavior of the cumulative ratio of programmer hours per software change.

APPLICATION:

Prediction – Extrapolating the final value of the ratio based on the current observed value and the values given by the model.

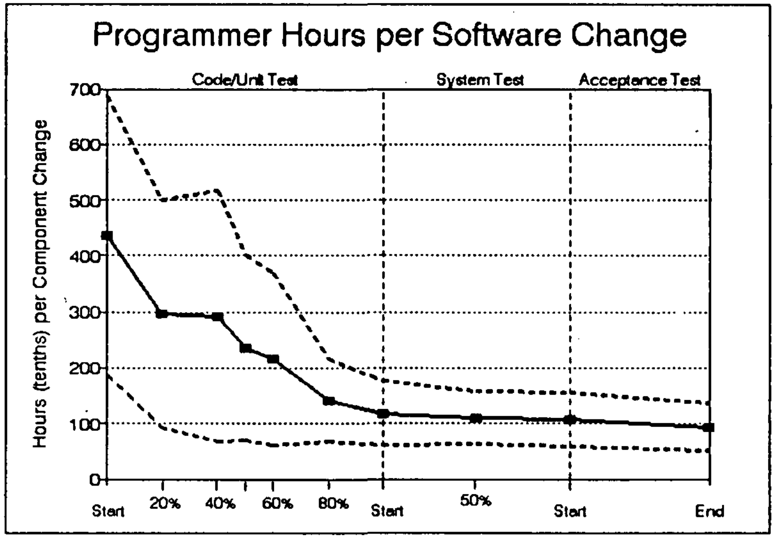
Assessment – Detecting problems by observing deviations of the ratio from values given by the model.

SOURCE:

Monitoring Software Development Through Dynamic Variables (Revision 1), SEL-83-106, p. 54

REPRESENTATION:

Phase	Hours (in tenths) per Software Change	Standard Deviation
Start coding	437.50	249.57
20% coding	297.29	204.09
40% coding	292.66	224.14
50% coding	235.36	166.41
60% coding	215.22	154.85
80% coding	141.77	75.62
Start system testing	118.81	58.36
50% system testing	110.54	47.61
Start acceptance testing	106.91	47.66
End acceptance testing	94.03	41.60



6199

TYPE: Environment Model

DESCRIPTION:

The model describes the behavior of the cumulative ratio of computer time per software change.

APPLICATION:

Prediction – Extrapolating the final value of the ratio based on the current observed value and the values given by the model.

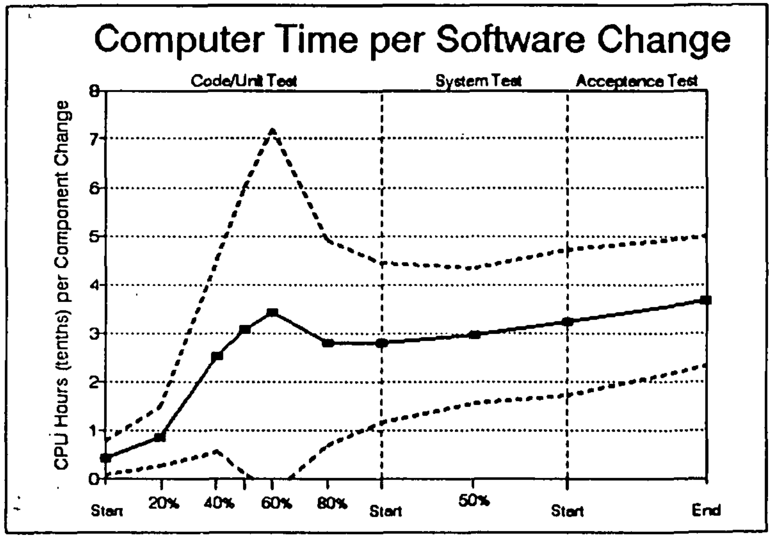
Assessment – Detecting problems by observing deviations of the ratio from values given by the model.

SOURCE:

Monitoring Software Development Through Dynamic Variables (Revision 1), SEL-83-106, p. 55

REPRESENTATION:

Phase	CPU Hours (tenths) per Software Change	Standard Deviation
Start coding	0.426	0.346
20% coding	0.871	0.616
40% coding	2.542	1.977
50% coding	3.058	2.962
60% coding	3.437	3.749
80% coding	2.802	2.112
Start system testing	2.804	1.643
50% system testing	2.950	1.404
Start acceptance testing	3.219	1.501
End acceptance testing	3.652	1.334



UNCERTAINTY IN EFFORT AND SIZE ESTIMATES BY PHASE

Result 4-16

TYPE: Uncertainty Model

DESCRIPTION:

The model describes the relative uncertainty to expect in effort and size estimates made for a project at various times in the software development life-cycle.

NOTE: Values for effort and size should be reestimated at the end of each phase as additional information becomes available about the eventual size of the completed project. The model reflects the manner in which the uncertainty in estimates decrease as more accurate data are known.

APPLICATION:

Planning – Provides a range of confidence for an initial estimate of effort or size and for any reestimates made during subsequent phases of the software development life-cycle.

SOURCE:

Manager's Handbook for Software Development (Revision 1), SEL-84-101, pp. 3-2, 3-3

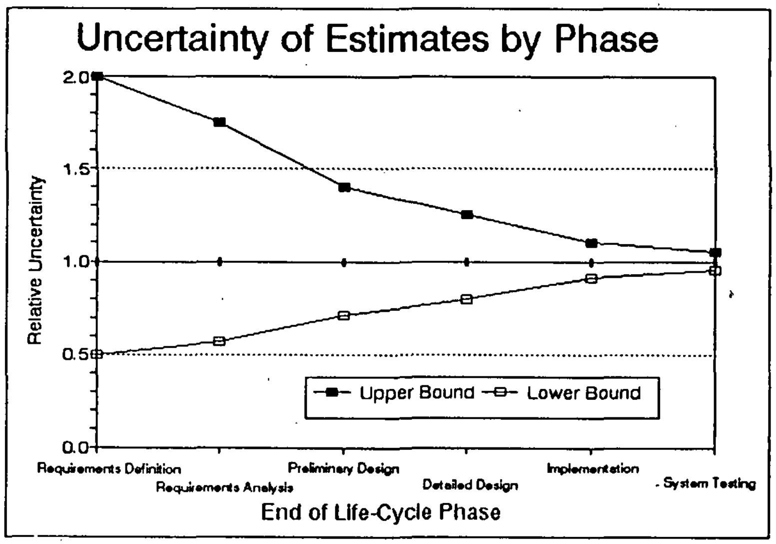
REPRESENTATION:

End of Phase	Estimate Uncertainty	Lower Bound	Upper Bound
Requirements Definition	1.00	0.50	2.00
Requirements Analysis	0.75	0.57	1.75
Preliminary Design	0.40	0.71	1.40
Detailed Design	0.25	0.80	1.25
Implementation	0.10	0.91	1.10
System Testing	0.05	0.95	1.05

where:

The uncertainty may be applied to effort or size estimates made at the end of each named life-cycle phase as follows:

$$Estimate_{upper\ bound} = Estimate * (1.0 + uncertainty)$$
$$Estimate_{lower\ bound} = Estimate / (1.0 + uncertainty)$$



TRENDS IN EFFORT TO CHANGE BY PHASE

Result 4-17

TYPE: Subjective Model

DESCRIPTION:

The model describes the behavior of the measure of effort to make a software change.

APPLICATION:

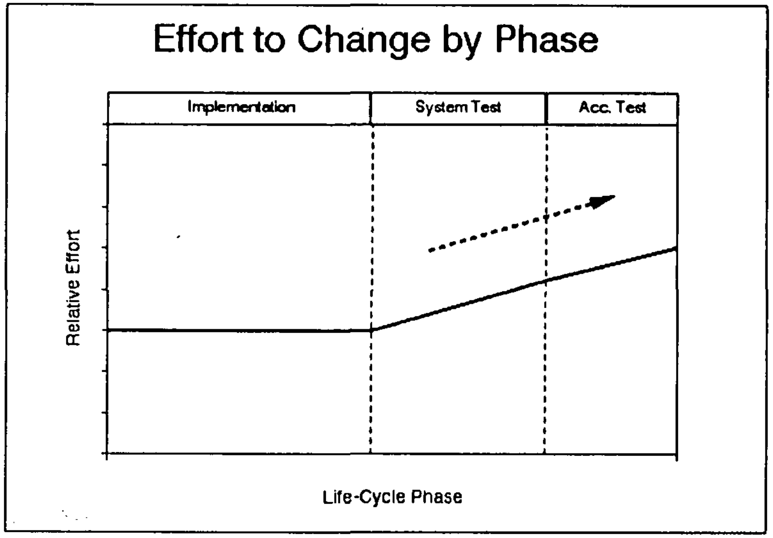
Assessment – Providing a characteristic pattern of behavior for the effort to make a software change that can be compared to actual project behavior to detect problems.

SOURCE:

[1]"Technical Summary – 1982: Report to the National Aeronautics and Space Administration," V. R. Basili, *Collected Software Engineering Papers: Volume II*, SEL-83-003, p. 2-12

REPRESENTATION:

"The amount of manpower to make a change should increase toward the end of the project and be stable at the beginning."



[1] Report originally prepared as a University of Maryland Technical Memorandum dated December 1982.

TRENDS IN CHANGES PER COMPUTER RUN BY PHASE — Result 4–18

TYPE: Subjective Model

DESCRIPTION:

The model describes the behavior of the ratio of software changes to computer runs.

APPLICATION:

Assessment – Providing a characteristic pattern of behavior for the ratio of software changes to computer runs that can be compared to actual project behavior to detect problems.

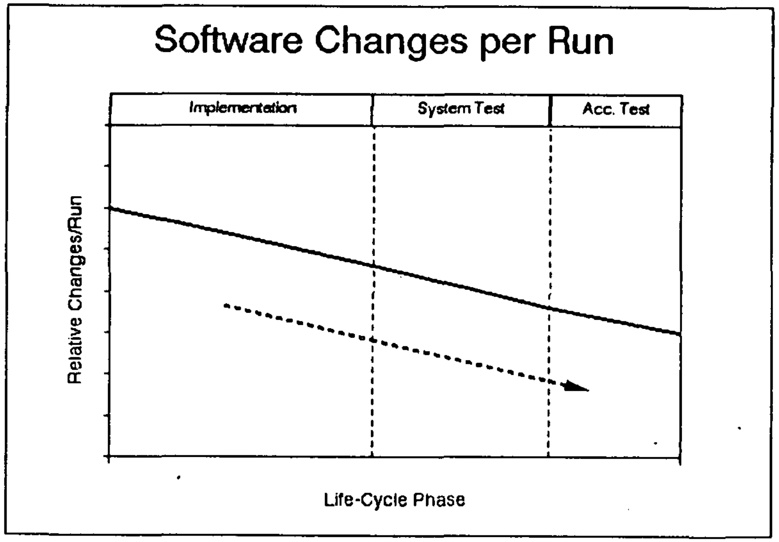
SOURCE:

[1] "Technical Summary – 1982: Report to the National Aeronautics and Space Administration," V. R. Basili, *Collected Software Engineering Papers: Volume II*, SEL-83-003, p. 2-12

1 Report originally prepared as a University of Maryland Technical Memorandum dated December 1982.

REPRESENTATION:

"The ratio of changes to computer runs should decrease as the project evolves."



TRENDS IN COMPUTER TIME PER CHANGE BY PHASE

Result 4–19

TYPE: Subjective Model

DESCRIPTION:

The model describes the behavior of the measure of computer time used to make a change.

APPLICATION:

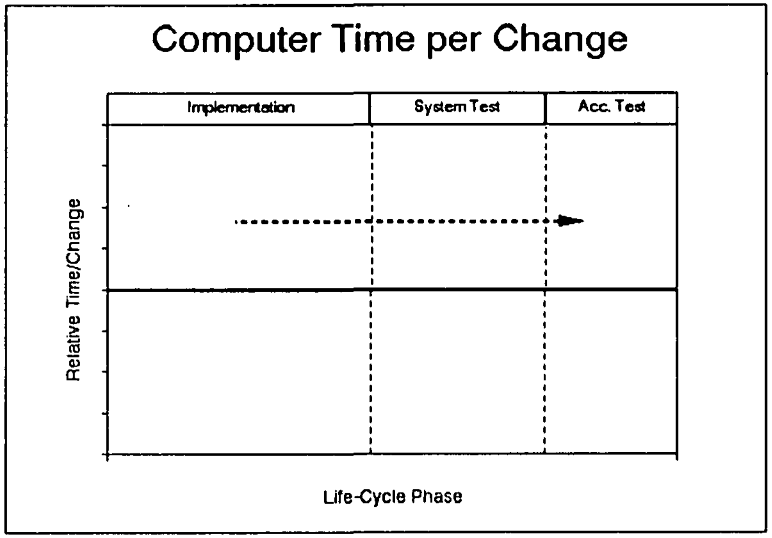
Assessment – Providing a characteristic pattern of behavior for the computer time used to make a change that can be compared to actual project behavior to detect problems.

SOURCE:

[1]"Technical Summary – 1982: Report to the National Aeronautics and Space Administration," V. R. Basili, *Collected Software Engineering Papers: Volume II*, SEL-83-003, p. 2-12

REPRESENTATION:

"The amount of computer time spent on detecting and correcting a given change will remain constant."



1 Report originally prepared as a University of Maryland Technical Memorandum dated December 1982.

SECTION 5 – MANAGEMENT RULES

The term "management rule" refers to a SEL research result that specifies how to interpret the observed behavior of a software development measure. These rules are formal statements typically used in evaluating data collected during ongoing development efforts. Rules can stand alone or can be combined to work together to give more detailed conclusions.

Automating the use of management rules is a recent area of interest in the SEL, although many of the rules listed here were recorded in early SEL research. Managers have always applied informal rules to the process of assessing a project's status.

The following paragraphs describe three groups of rules.

- Independent rules stand alone and are applicable to a single specific situation. An observation is interpreted with no mention of when the observation is made during the life cycle nor is the measure compared to a model to aid in drawing the conclusion.

  Independent rules are typical of what managers think of as "rules of thumb" or "project management lore." They usually state something about the overall status of the project.

- Model-dependent rules require a standard, or model, to which a measure is first compared. Based on the comparison and an observed deviation from the standard, an interpretation is presented. This type of rule was implied by the "assessment" applications described for models in Section 4. This type of rule requires both a model of the measure and possibly an estimate of the completion value for the measure.

  During the life of a project, a manager charts software development data to monitor progress. By comparing the actual progress to a model, the manager can spot deviations earlier. Model-dependent rules describe the consequences of a particular deviation based on observations of previous projects.

- Phase-dependent rules are model-dependent rules that are made more elaborate by introducing the life-cycle phase into the rule. This allows different conclusions to be drawn based on the time at which the measure deviated from the standard.

Sets of rules (independent, model-dependent, and/or phase-dependent) can be created to provide more depth on which to draw conclusions. While each rule will probably not result in the same conclusion, the advantage of a set of rules comes from "voting" the conclusions. The one conclusion that is most prevalent (or better yet, in the majority) has a good likelihood of being appropriate to the project's situation. This is similar to managers "taking everything into account."

The following pages describe a selection of rules published by the SEL.

VARIATIONS IN EFFORT TO CHANGE

Result 5-1

TYPE: Independent Rule

DESCRIPTION:

The rule describes the type of behavior of the measure of effort to make a software change that indicates a poor quality development process.

APPLICATION:

Assessment – Detecting the lack of a quality development process by observing the described behavior.

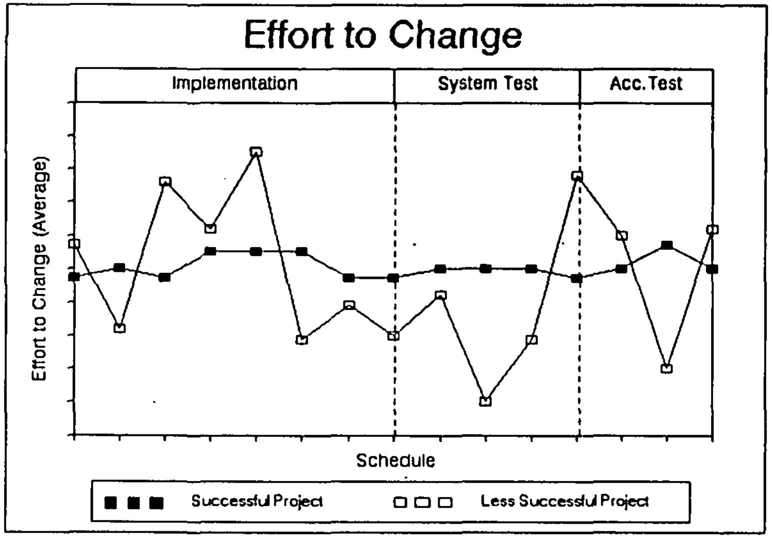
SOURCE:

[1]"Technical Summary – 1982: Report to the National Aeronautics and Space Administration," V. R. Basili, *Collected Software Engineering Papers: Volume II*, SEL-83-003, p. 2-12

REPRESENTATION:

"Projects deemed less successful by subjective analysis have sharp changes in the amount of manpower spent per change."

If
    the effort to make a change is varying sharply
then
    the project will be less successful



[1]Report originally prepared as a University of Maryland Technical Memorandum dated December 1982.

DEVIATIONS IN STAFFING PLAN

Result 5-2

TYPE: Independent Rule

DESCRIPTION:

The rule describes the type of behavior of the actual staffing level that indicates a poor *quality development process.*

APPLICATION:

Assessment – Detecting the lack of a quality development process by observing the described behavior.

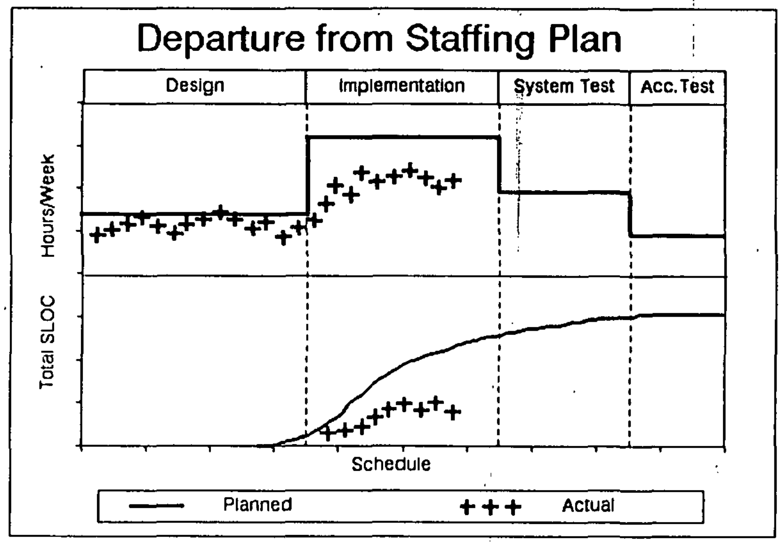
SOURCE:

Measures and Metrics for Software Development, SEL-83-002, p. 4-22

REPRESENTATION:

"Departures from planned staffing usually indicate that production will depart from plan, too."

If
the staffing level is varying from the plan
then
production will not match the plan



6199

VARIATIONS IN SIZE AND EFFORT ESTIMATES

Result 5-3

TYPE: Independent Rule

DESCRIPTION:

The rule describes the type of behavior of estimates of system size and total effort that indicates a poor quality development process.

APPLICATION:

Assessment – Detecting the lack of a quality development process by observing the described behavior.

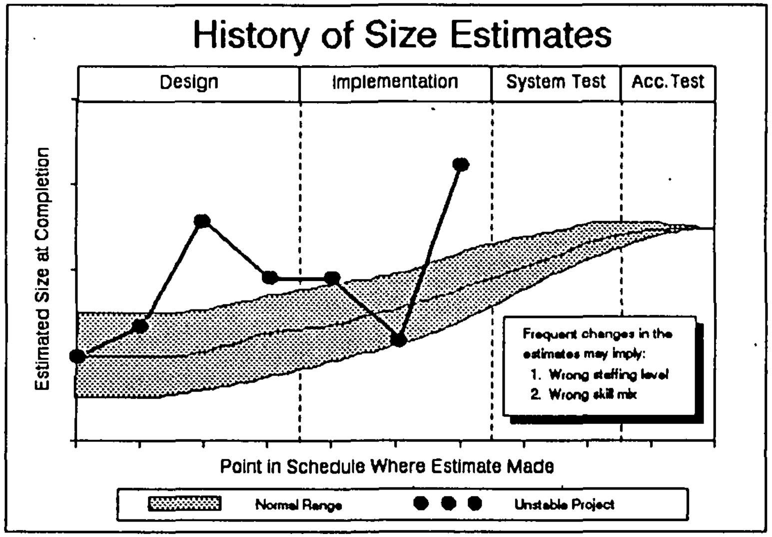
SOURCE:

Measures and Metrics for Software Development, SEL-83-002, p. 4-22

REPRESENTATION:

"Frequent changes in size and cost estimates can imply that the estimates are being adjusted to fit an inappropriate staffing level or that the skill mix of the development team is inappropriate to the task."

If
    size or effort estimates are changing frequently
then
    the estimates are being adjusted to match the wrong staffing level or
    the development team has the wrong skill mix



6199

HIGH COMPUTER USE

Result 5–4

TYPE: Model-Dependent Rule

DESCRIPTION:

The rule describes the type of behavior of the measure of computer use that indicates a poor quality development process.

APPLICATION:

Assessment – Detecting the lack of a quality development process by observing the described behavior.

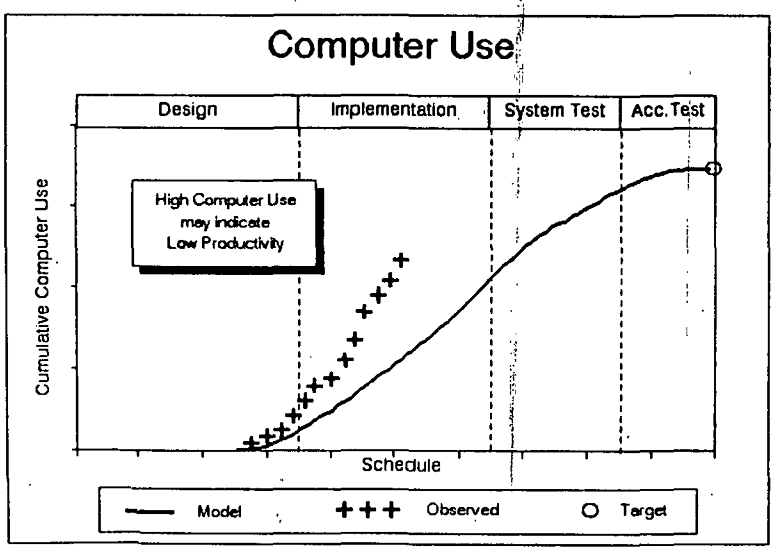
SOURCE:

[1]A Software Technology Evaluation Program," D. N. Card, *Collected Software Engineering Papers: Volume III*, SEL-85-003, p. 2-3

REPRESENTATION:

"Extensive computer use is associated with low productivity."

If
computer use is higher than expected
then
productivity will be low



[1] Article originally appeared in *Annais do XVIII Congresso Nacional de Informatica*, October 1985.

DEVIATIONS IN COMPUTER USE PER LINE OF CODE

TYPE: Phase-Dependent Rule

DESCRIPTION:

The rule describes the type of behavior of the measure of computer use that indicates a poor quality development process.

APPLICATION:

Assessment – Diagnosing the underlying reasons if the computer use per line of code is observed to depart from normal behavior.

SOURCE:

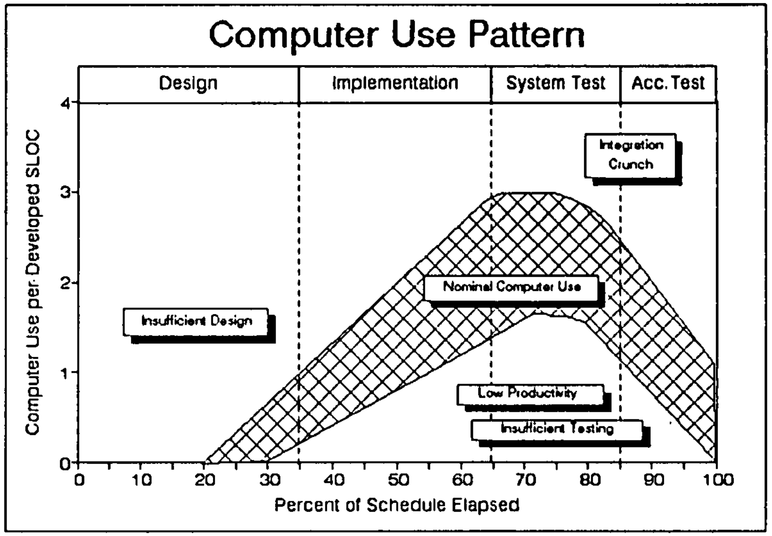
Measures and Metrics for Software Development, SEL-83-002, pp. 4-6, 4-9

REPRESENTATION:

If
the phase is design and computer use is high
then
there is insufficient design effort being expended

If
the phase is implementation or test and computer use is low
then
there is insufficient testing being done or
there is low productivity

If
the phase is test and computer use is high
then
the project is in an integration crunch



DEVIATIONS IN LINES OF CODE

Result 5-6

TYPE: Phase-Dependent Rule

DESCRIPTION:

The rule describes the type of behavior of the measure of software size that indicates a poor quality development process.

APPLICATION:

Assessment – Diagnosing the underlying reasons if the lines of code measure is observed to depart from normal behavior.

SOURCE:

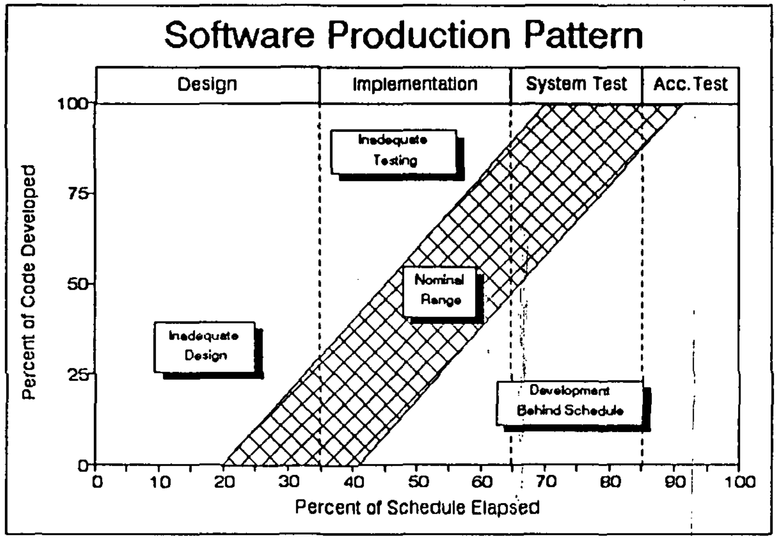
Measures and Metrics for Software Development, SEL-83-002, pp. 4-7, 4-8

REPRESENTATION:

If
  the phase is design and lines of code is high
then
  there is inadequate design effort being expended

If
  the phase is implementation and lines of code is high
then
  there is inadequate unit testing being done

If
  the phase is implementation or test and lines of code is low
then
  the development effort is behind schedule



DEVIATIONS IN CHANGES PER LINE OF CODE

Result 5–7

TYPE: Phase-Dependent Rule

DESCRIPTION:

The rule describes the type of behavior of the measure of change rate that indicates a poor quality development process.

APPLICATION:

Assessment – Diagnosing the underlying reasons if the change rate measure is observed to depart from normal behavior.

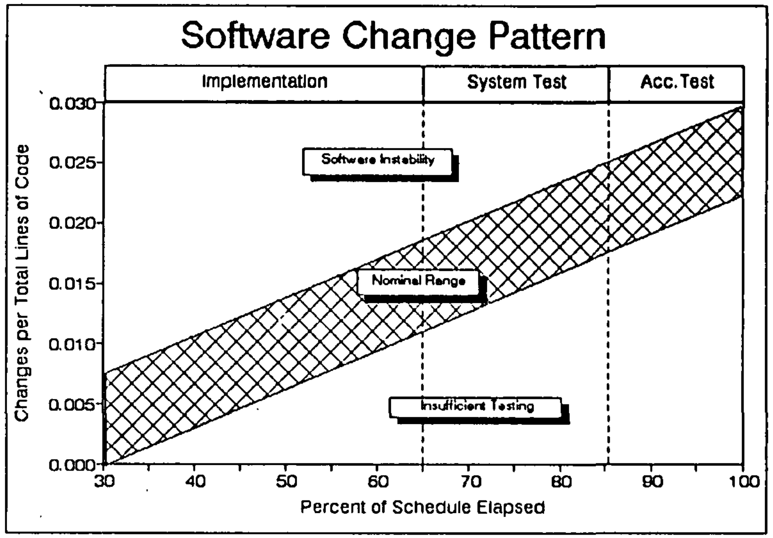
SOURCE:

Measures and Metrics for Software Development, SEL-83-002, pp. 4-10 to 4-12

REPRESENTATION:

If
  the phase is implementation or test and changes per line of code is high
then
  the software is unstable

If
  the phase is implementation or test and changes per line of code is low
then
  there is insufficient testing being done



6199

DEVIATIONS IN LINES OF CODE PER STAFF HOUR

Result 5–8

TYPE: Phase-Dependent Rule

DESCRIPTION:

The rule describes the type of behavior of a measure of productivity that indicates a poor quality development process.

APPLICATION:

Assessment – Diagnosing the underlying reasons if the productivity measure is observed to depart from normal behavior.

SOURCE:

Measures and Metrics for Software Development, SEL-83-002, pp. 4-20, 4-21

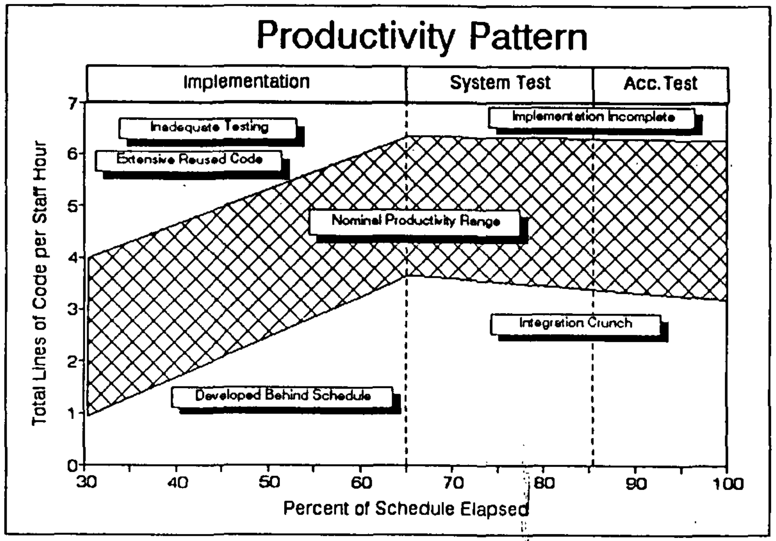
REPRESENTATION:

If
 the phase is implementation and lines of code per hour is high
then
 there is extensive code being reused or
 there is inadequate unit testing being done

If
 the phase is implementation and lines of code per hour is low
then
 the development effort is behind schedule

If
 the phase is test and lines of code per hour is high
then
 the implementation is incomplete

If
 the phase is test and lines of code per hour is low
then
 the development is in an integration crunch



6199

GLOSSARY

CPU	central processing unit
CSC	Computer Sciences Corporation
FTE	full-time equivalent
GSFC	Goddard Space Flight Center
KSLOC	source lines of code in thousands
LOC	lines of code
NASA	National Aeronautics and Space Administration
SEL	Software Engineering Laboratory
SLOC	source lines of code
SME	Software Management Environment
TBD	to be determined

REFERENCES

1. SEL-81-104, *The Software Engineering Laboratory*, D. N. Card, F. E. McGarry, G. Page, et al., February 1982
2. SEL-81-101, *Guide to Data Collection*, V. E. Church, D. N. Card, and F. E. McGarry, August 1982
3. SEL-87-008, *Data Collection Procedures for the Rehosted SEL Database*, G. Heller, October 1987
4. SEL-89-003, *Software Management Environment (SME) Concepts and Architecture*, W. Decker and J. Valett, August 1989
5. SEL-79-002, *The Software Engineering Laboratory: Relationship Equations*, K. Freburger and V. R. Basili, May 1979
6. SEL-83-001, *An Approach to Software Cost Estimation*, F. E. McGarry, G. Page, D. N. Card, et al., February 1984
7. SEL-83-002, *Measures and Metrics for Software Development*, D. N. Card, F. E. McGarry, G. Page, et al., March 1984
8. SEL-83-106, *Monitoring Software Development Through Dynamic Variables (Revision 1)*, C. W. Doerflinger, November 1989
9. SEL-84-101, *Manager's Handbook for Software Development (Revision 1)*, L. Landis, F. E. McGarry, S. Waligora, et al., November 1990
10. SEL-83-003, *Collected Software Engineering Papers: Volume II*, November 1983
11. SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985
12. SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988
13. SEL-82-007, *Proceedings of the Seventh Annual Software Engineering Workshop*, December 1982
14. SEL-89-007, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, November 1989

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, *Proceedings From the First Summer Software Engineering Workshop*, August 1976

SEL-77-002, *Proceedings From the Second Summer Software Engineering Workshop*, September 1977

SEL-77-004, *A Demonstration of AXES for NAVPAK*, M. Hamilton and S. Zeldin, September 1977

SEL-77-005, *GSFC NAVPAK Design Specifications Languages Study*, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-005, *Proceedings From the Third Summer Software Engineering Workshop*, September 1978

SEL-78-006, *GSFC Software Engineering Research Requirements Analysis Study*, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, *Applicability of the Rayleigh Curve to the SEL Environment*, T. E. Mapp, December 1978

SEL-78-302, *FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3)*, W. J. Decker and W. A. Taylor, July 1986

SEL-79-002, *The Software Engineering Laboratory: Relationship Equations*, K. Freburger and V. R. Basili, May 1979

SEL-79-003, *Common Software Module Repository (CSMR) System Description and User's Guide*, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, *Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment*, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, *Proceedings From the Fourth Summer Software Engineering Workshop*, November 1979

SEL-80-002, *Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation*, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-003, *Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study*, T. Welden, M. McClellan, and P. Liebertz, May 1980

SEL-80-005, *A Study of the Musa Reliability Model*, A. M. Miller, November 1980

SEL-80-006, *Proceedings From the Fifth Annual Software Engineering Workshop*, November 1980

SEL-80-007, *An Appraisal of Selected Cost/Resource Estimation Models for Software Systems*, J. F. Cook and F. E. McGarry, December 1980

SEL-80-008, *Tutorial on Models and Metrics for Software Management and Engineering*, V. R. Basili, 1980

SEL-81-008, *Cost and Reliability Estimation Models (CAREM) User's Guide*, J. F. Cook and E. Edwards, February 1981

SEL-81-009, *Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation*, W. J. Decker and F. E. McGarry, March 1981

SEL-81-011, *Evaluating Software Development by Analysis of Change Data*, D. M. Weiss, November 1981

SEL-81-012, *The Rayleigh Curve as a Model for Effort Distribution Over the Life of Medium Scale Software Systems*, G. O. Picasso, December 1981

SEL-81-013, *Proceedings From the Sixth Annual Software Engineering Workshop*, December 1981

SEL-81-014, *Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL)*, A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-81-101, *Guide to Data Collection*, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-104, *The Software Engineering Laboratory*, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

SEL-81-107, *Software Engineering Laboratory (SEL) Compendium of Tools*, W. J. Decker, W. A. Taylor, and E. J. Smith, February 1982

SEL-81-110, *Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics*, G. Page, F. E. McGarry, and D. N. Card, June 1985

SEL-81-205, *Recommended Approach to Software Development*, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

SEL-82-001, *Evaluation of Management Measures of Software Development*, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-004, *Collected Software Engineering Papers: Volume 1*, July 1982

SEL-82-007, *Proceedings From the Seventh Annual Software Engineering Workshop*, December 1982

SEL-82-008, *Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory*, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, *FORTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1)*, W. A. Taylor and W. J. Decker, April 1985

SEL-82-105, *Glossary of Software Engineering Laboratory Terms*, T. A. Babst, F. E. McGarry, and M. G. Rohleder, October 1983

SEL-82-906, *Annotated Bibliography of Software Engineering Laboratory Literature*, P. Groves and J. Valett, November 1990

SEL-83-001, *An Approach to Software Cost Estimation*, F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, *Measures and Metrics for Software Development*, D. N. Card, F. E. McGarry, G. Page, et al., March 1984

SEL-83-003, *Collected Software Engineering Papers: Volume II*, November 1983

SEL-83-006, *Monitoring Software Development Through Dynamic Variables*, C. W. Doerflinger, November 1983

SEL-83-007, *Proceedings From the Eighth Annual Software Engineering Workshop*, November 1983

SEL-83-106, *Monitoring Software Development Through Dynamic Variables (Revision 1)*, C. W. Doerflinger, November 1989

SEL-84-101, *Manager's Handbook for Software Development, Revision 1*, L. Landis, F. McGarry, S. Waligora, et al., November 1990

SEL-84-003, *Investigation of Specification Measures for the Software Engineering Laboratory (SEL)*, W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, *Proceedings From the Ninth Annual Software Engineering Workshop*, November 1984

SEL-85-001, *A Comparison of Software Verification Techniques*, D. N. Card, R. W. Selby, Jr., F. E. McGarry, et al., April 1985

SEL-85-002, *Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team*, R. Murphy and M. Stark, October 1985

SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985

SEL-85-004, *Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics*, R. W. Selby, Jr., May 1985

SEL-85-005, *Software Verification and Testing*, D. N. Card, C. Antle, and E. Edwards, December 1985

SEL-85-006, *Proceedings From the Tenth Annual Software Engineering Workshop*, December 1985

SEL-86-001, *Programmer's Handbook for Flight Dynamics Software Development*, R. Wood and E. Edwards, March 1986

SEL-86-002, *General Object-Oriented Software Development*, E. Seidewitz and M. Stark, August 1986

SEL-86-003, *Flight Dynamics System Software Development Environment Tutorial*, J. Buell and P. Myers, July 1986

SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986

SEL-86-005, *Measuring Software Design*, D. N. Card, October 1986

SEL-86-006, *Proceedings From the Eleventh Annual Software Engineering Workshop*, December 1986

SEL-87-001, *Product Assurance Policies and Procedures for Flight Dynamics Software Development*, S. Perry et al., March 1987

SEL-87-002, *Ada Style Guide (Version 1.1)*, E. Seidewitz et al., May 1987

SEL-87-003, *Guidelines for Applying the Composite Specification Model (CSM)*, W. W. Agresti, June 1987

SEL-87-004, *Assessing the Ada Design Process and Its Implications: A Case Study*, S. Godfrey, C. Brophy, et al., July 1987

SEL-87-008, *Data Collection Procedures for the Rehosted SEL Database*, G. Heller, October 1987

SEL-87-009, *Collected Software Engineering Papers: Volume V*, S. DeLong, November 1987

SEL-87-010, *Proceedings From the Twelfth Annual Software Engineering Workshop*, December 1987

SEL-88-001, *System Testing of a Production Ada Project: The GRODY Study*, J. Seigle, L. Esker, and Y. Shi, November 1988

SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988

SEL-88-003, *Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis*, K. Quimby and L. Esker, December 1988

SEL-88-004, *Proceedings of the Thirteenth Annual Software Engineering Workshop*, November 1988

SEL-88-005, *Proceedings of the First NASA Ada User's Symposium*, December 1988

SEL-89-002, *Implementation of a Production Ada Project: The GRODY Study*, S. Godfrey and C. Brophy, September 1989

SEL-89-003, *Software Management Environment (SME) Concepts and Architecture*, W. Decker and J. Valett, August 1989

SEL-89-004, *Evolution of Ada Technology in the Flight Dynamics Area: Implementation/Testing Phase Analysis*, K. Quimby, L. Esker, L. Smith, M. Stark, and F. McGarry, November 1989

SEL-89-005, *Lessons Learned in the Transition to Ada From FORTRAN at NASA/ Goddard*, C. Brophy, November 1989

SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989

SEL-89-007, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, November 1989

SEL-89-008, *Proceedings of the Second NASA Ada Users' Symposium*, November 1989

SEL-89-101, *Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 1)*, M. So, G. Heller, S. Steinberg, K. Pumphrey, and D. Spiegel, February 1990

SEL-90-001, *Database Access Manager for the Software Engineering Laboratory (DAMSEL) User's Guide*, M. Buhler and K. Pumphrey, March 1990

SEL-90-002, *The Cleanroom Case Study in the Software Engineering Laboratory: Project Description and Early Analysis*, S. Green et al., March 1990

SEL-90-003, *A Study of the Portability of an Ada System in the Software Engineering Laboratory (SEL)*, L. O. Jun and S. R. Valett, June 1990

SEL-90-004, *Gamma Ray Observatory Dynamics Simulator in Ada (GRODY) Experiment Summary*, T. McDermott and M. Stark, September 1990

SEL-90-005, *Collected Software Engineering Papers: Volume VIII*, November 1990

SEL-91-001, *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules*, W. J. Decker, R. Hendrick, and J. Valett, February 1991

SEL-RELATED LITERATURE

[4]Agresti, W. W., V. E. Church, D. N. Card, and P. L. Lo, "Designing With Ada for Satellite Simulation: A Case Study," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

[2]Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," *Program Transformation and Programming Environments*. New York: Springer-Verlag, 1984

[1]Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," *Proceedings of the Fifth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1981

[1]Basili, V. R., "Models and Metrics for Software Management and Engineering," *ASME Advances in Computer Technology*, January 1980, vol. 1

Basili, V. R., *Tutorial on Models and Metrics for Software Management and Engineering*. New York: IEEE Computer Society Press, 1980 (also designated SEL-80-008)

[3]Basili, V. R., "Quantitative Evaluation of Software Methodology," *Proceedings of the First Pan-Pacific Computer Conference*, September 1985

[7]Basili, V. R., *Maintenance = Reuse-Oriented Software Development*, University of Maryland, Technical Report TR-2244, May 1989

[7]Basili, V. R., *Software Development: A Paradigm for the Future*, University of Maryland, Technical Report TR-2263, June 1989

[8]Bailey, J. W., and V. R. Basili, "Software Reclamation: Improving Post-Development Reusability," *Proceedings of the Eighth Annual National Conference on Ada Technology*, March 1990

[8]Basili, V. R., "Viewing Maintenance of Reuse-Oriented Software Development," *IEEE Software*, January 1990

[1]Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?," *Journal of Systems and Software*, February 1981, vol. 2, no. 1

[1]Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," *Journal of Systems and Software*, February 1981, vol. 2, no. 1

[3]Basili, V. R., and N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," *Proceedings of the International Computer Software and Applications Conference*, October 1985

[4]Basili, V. R., and D. Patnaik, *A Study on Fault Prediction and Reliability Assessment in the SEL Environment*, University of Maryland, Technical Report TR-1699, August 1986

[2]Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Communications of the ACM*, January 1984, vol. 27, no. 1

[1]Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," *Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics*, March 1981

Basili, V. R., and J. Ramsey, *Structural Coverage of Functional Testing*, University of Maryland, Technical Report TR-1442, September 1984

[3]Basili, V. R., and C. L. Ramsey, "ARROWSMITH-P—A Prototype Expert System for Software Engineering Management," *Proceedings of the IEEE/MITRE Expert Systems in Government Symposium*, October 1985

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," *Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost*. New York: IEEE Computer Society Press, 1979

[5]Basili, V., and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," *Proceedings of the 9th International Conference on Software Engineering*, March 1987

[5]Basili, V., and H. D. Rombach, "T A M E: Tailoring an Ada Measurement Environment," *Proceedings of the Joint Ada Conference*, March 1987

[5]Basili, V., and H. D. Rombach, "T A M E: Integrating Measurement Into Software Environments," University of Maryland, Technical Report TR-1764, June 1987

[6]Basili, V. R., and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, June 1988

[7]Basili, V. R., and H. D. Rombach, *Towards A Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment*, University of Maryland, Technical Report TR-2158, December 1988

[8]Basili, V. R., and H. D. Rombach, *Towards A Comprehensive Framework for Reuse: Model-Based Reuse Characterization Schemes*, University of Maryland, Technical Report TR-2446, April 1990

[2]Basili, V. R., R. W. Selby, Jr., and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," *IEEE Transactions on Software Engineering*, November 1983

[3]Basili, V. R., and R. W. Selby, Jr., "Calculation and Use of an Environment's Characteristic Software Metric Set," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

Basili, V. R., and R. W. Selby, Jr., *Comparing the Effectiveness of Software Testing Strategies*, University of Maryland, Technical Report TR-1501, May 1985

[3]Basili, V. R., and R. W. Selby, Jr., "Four Applications of a Software Data Collection and Analysis Methodology," *Proceedings of the NATO Advanced Study Institute*, August 1985

[4]Basili, V. R., R. W. Selby, Jr., and D. H. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering*, July 1986

[5]Basili, V., and R. Selby, Jr., "Comparing the Effectiveness of Software Testing Strategies," *IEEE Transactions on Software Engineering*, December 1987

[2]Basili, V. R., and D. M. Weiss, *A Methodology for Collecting Valid Software Engineering Data*, University of Maryland, Technical Report TR-1235, December 1982

[3]Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, November 1984

[1]Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," *Proceedings of the Fifteenth Annual Conference on Computer Personnel Research*, August 1977

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," *Proceedings of the Software Life Cycle Management Workshop*, September 1977

[1]Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," *Proceedings of the Second Software Life Cycle Management Workshop*, August 1978

[1]Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," *Computers and Structures*, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," *Proceedings of the Third International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1978

[5]Brophy, C., W. Agresti, and V. Basili, "Lessons Learned in Use of Ada-Oriented Design Methods," *Proceedings of the Joint Ada Conference*, March 1987

[6]Brophy, C. E., S. Godfrey, W. W. Agresti, and V. R. Basili, "Lessons Learned in the Implementation Phase of a Large Ada Project," *Proceedings of the Washington Ada Technical Conference*, March 1988

[2]Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

[2]Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982

[3]Card, D. N., "A Software Technology Evaluation Program," *Annais do XVIII Congresso Nacional de Informatica*, October 1985

[5]Card, D., and W. Agresti, "Resolving the Software Science Anomaly," *The Journal of Systems and Software*, 1987

[6]Card, D. N., and W. Agresti, "Measuring Software Design Complexity," *The Journal of Systems and Software*, June 1988

Card, D. N., V. E. Church, W. W. Agresti, and Q. L. Jordan, "A Software Engineering View of Flight Dynamics Analysis System," Parts I and II, Computer Sciences Corporation, Technical Memorandum, February 1984

[4]Card, D. N., V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," *IEEE Transactions on Software Engineering*, February 1986

Card, D. N., Q. L. Jordan, and V. E. Church, "Characteristics of FORTRAN Modules," Computer Sciences Corporation, Technical Memorandum, June 1984

[5]Card, D., F. McGarry, and G. Page, "Evaluating Software Engineering Technologies," *IEEE Transactions on Software Engineering*, July 1987

[3]Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

[1]Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," *Proceedings of the Fifth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1981

[4]Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan, "An Approach for Assessing Software Prototypes," *ACM Software Engineering Notes*, July 1986

[2]Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," *Proceedings of the Seventh International Computer Software and Applications Conference*. New York: IEEE Computer Society Press, 1983

[5]Doubleday, D., *ASAP: An Ada Static Source Code Analyzer Program*, University of Maryland, Technical Report TR-1895, August 1987 (NOTE: 100 pages long)

[6]Godfrey, S., and C. Brophy, "Experiences in the Implementation of a Large Ada Project," *Proceedings of the 1988 Washington Ada Symposium*, June 1988

Hamilton, M., and S. Zeldin, *A Demonstration of AXES for NAVPAK*, Higher Order Software, Inc., TR-9, September 1977 (also designated SEL-77-005)

Jeffery, D. R., and V. Basili, *Characterizing Resource Data: A Model for Logical Association of Software Data*, University of Maryland, Technical Report TR-1848, May 1987

[6]Jeffery, D. R., and V. R. Basili, "Validating the TAME Resource Data Model," *Proceedings of the Tenth International Conference on Software Engineering*, April 1988

[5]Mark, L., and H. D. Rombach, *A Meta Information Base for Software Engineering*, University of Maryland, Technical Report TR-1765, July 1987

[6]Mark, L., and H. D. Rombach, "Generating Customized Software Engineering Information Bases From Software Process and Product Specifications," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989

[5]McGarry, F., and W. Agresti, "Measuring Ada for Software Development in the Software Engineering Laboratory (SEL)," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

[7]McGarry, F., L. Esker, and K. Quimby, "Evolution of Ada Technology in a Production Software Environment," *Proceedings of the Sixth Washington Ada Symposium (WADAS)*, June 1989

[3]McGarry, F. E., J. Valett, and D. Hall, "Measuring the Impact of Computer Resource Quality on the Software Development Process and Product," *Proceedings of the Hawaiian International Conference on System Sciences*, January 1985

National Aeronautics and Space Administration (NASA), *NASA Software Research Technology Workshop* (Proceedings), March 1980

[3]Page, G., F. E. McGarry, and D. N. Card, "A Practical Experience With Independent Verification and Validation," *Proceedings of the Eighth International Computer Software and Applications Conference*, November 1984

[5]Ramsey, C., and V. R. Basili, *An Evaluation of Expert Systems for Software Engineering Management*, University of Maryland, Technical Report TR-1708, September 1986

[3]Ramsey, J., and V. R. Basili, "Analyzing the Test Process Using Structural Coverage," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

[5]Rombach, H. D., "A Controlled Experiment on the Impact of Software Structure on Maintainability," *IEEE Transactions on Software Engineering*, March 1987

[8]Rombach, H. D., "Design Measurement: Some Lessons Learned," *IEEE Software*, March 1990

[6]Rombach, H. D., and V. R. Basili, "Quantitative Assessment of Maintenance: An Industrial Case Study," *Proceedings From the Conference on Software Maintenance*, September 1987

[6]Rombach, H. D., and L. Mark, "Software Process and Product Specifications: A Basis for Generating Customized SE Information Bases," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989

[7]Rombach, H. D., and B. T. Ulery, *Establishing a Measurement Based Maintenance Improvement Program: Lessons Learned in the SEL*, University of Maryland, Technical Report TR-2252, May 1989

[5]Seidewitz, E., "General Object-Oriented Software Development: Background and Experience," *Proceedings of the 21st Hawaii International Conference on System Sciences*, January 1988

[6]Seidewitz, E., "General Object-Oriented Software Development with Ada: A Life Cycle Approach," *Proceedings of the CASE Technology Conference*, April 1988

[6]Seidewitz, E., "Object-Oriented Programming in Smalltalk and Ada," *Proceedings of the 1987 Conference on Object-Oriented Programming Systems, Languages, and Applications*, October 1987

[4]Seidewitz, E., and M. Stark, "Towards a General Object-Oriented Software Development Methodology," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

[8]Stark, M., "On Designing Parametrized Systems Using Ada," *Proceedings of the Seventh Washington Ada Symposium*, June 1990

[7]Stark, M. E. and E. W. Booth, "Using Ada to Maximize Verbatim Software Reuse," *Proceedings of TRI-Ada 1989*, October 1989

Stark, M., and E. Seidewitz, "Towards a General Object-Oriented Ada Lifecycle," *Proceedings of the Joint Ada Conference*, March 1987

[8]Straub, P. A., and M. Zelkowitz, "PUC: A Functional Specification Language for Ada," *Proceedings of the Tenth International Conference of the Chilean Computer Science Society*, July 1990

[7]Sunazuka, T., and V. R. Basili, *Integrating Automated Support for a Software Management Cycle Into the TAME System*, University of Maryland, Technical Report TR-2289, July 1989

Turner, C., and G. Caron, *A Comparison of RADC and NASA/SEL Software Development Data, Data and Analysis Center for Software*, Special Publication, May 1981

Turner, C., G. Caron, and G. Brement, *NASA/SEL Data Compendium, Data and Analysis Center for Software*, Special Publication, April 1981

[5]Valett, J., and F. McGarry, "A Summary of Software Measurement Experiences in the Software Engineering Laboratory," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

[3]Weiss, D. M., and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data From the Software Engineering Laboratory," *IEEE Transactions on Software Engineering*, February 1985

[5]Wu, L., V. Basili, and K. Reed, "A Structure Coverage Tool for Ada Software Systems," *Proceedings of the Joint Ada Conference*, March 1987

[1]Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," *Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science*. New York: IEEE Computer Society Press, 1979

[2]Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," *Empirical Foundations for Computer and Information Science* (Proceedings), November 1982

[6]Zelkowitz, M. V., "The Effectiveness of Software Prototyping: A Case Study," *Proceedings of the 26th Annual Technical Symposium of the Washington, D. C., Chapter of the ACM*, June 1987

[6]Zelkowitz, M. V., "Resource Utilization During Software Development," *Journal of Systems and Software*, 1988

[8]Zelkowitz, M. V., "Evolution Towards Specifications Environment: Experience With Syntax Editors," *Information and Software Technology*, April 1990

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," *Proceedings of the Software Life Cycle Management Workshop*, September 1977

NOTES:

[1]This article also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I*, July 1982.

[2]This article also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II*, November 1983.

[3]This article also appears in SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985.

[4]This article also appears in SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986.

[5]This article also appears in SEL-87-009, *Collected Software Engineering Papers: Volume V*, November 1987.

[6]This article also appears in SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988.

[7]This article also appears in SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989.

[8]This article also appears in SEL-90-005, *Collected Software Engineering Papers: Volume VIII*, November 1990.