CR-189346

SOFTWARE ENGINEERING LABORATORY SERIES SEL-94-001

SOFTWARE MANAGEMENT ENVIRONMENT (SME)

COMPONENTS AND ALGORITHMS

FEBRUARY 1994

(NASA-CR-189346) SOFTWARE MANAGEMENT ENVIRONMENT (SME): COMPONENTS AND ALGORITHMS (NASA. Goddard Space Flight Center) 133 p

N94-34996

Unclas

G3/61 0012607

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY SERIES SEL-94-001

SOFTWARE MANAGEMENT ENVIRONMENT (SME)

COMPONENTS AND ALGORITHMS

FEBRUARY 1994

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 to investigate 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, Software Engineering Branch

University of Maryland, Department of Computer Science

Computer Sciences Corporation, Software Engineering Operation

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect 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

Robert Hendrick (CSC)
David Kistler (CSC)
Jon Valett (GSFC)

Single copies of this document can be obtained by writing to

Software Engineering Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

This document presents the components and algorithms of the Software Management Environment (SME), a management tool developed for the Software Engineering Branch (Code 552) of the Flight Dynamics Division (FDD) of the Goddard Space Flight Center (GSFC). The SME provides an integrated set of visually oriented experienced-based tools that can assist software development managers in managing and planning software development projects. This document describes and illustrates the analysis functions that underlie the SME's project monitoring, estimation, and planning tools. *SME Components and Algorithms* is a companion reference to *SME Concepts and Architecture*, and *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules.*

Table of Contents

Table of Contents

Table of Contents

List of Illustrations

Figure

List of Illustrations

Figure

List of Illustrations

Figure

List of Illustrations

Figure

List of Illustrations

Figure

List of Tables

Table

SECTION 1—INTRODUCTION

The Software Management Environment (SME) is an interactive management tool developed under the sponsorship of the Software Engineering Laboratory (SEL) at the National Aeronautics and Space Administration's Goddard Space Flight Center (NASA/GSFC). The tool supports a key set of experience-based functions that utilize software metrics to assist software development managers in actively tracking and evaluating the status of their projects.

The SME provides a range of visually oriented features to help software managers observe the progress of an ongoing project, compare the project to other efforts or to models of how projects normally behave in the environment, predict the probable future behavior of the project, analyze the project's strengths and weaknesses, assess the project's quality relative to previous efforts, and examine "what if" scenarios by varying the project's plan. These functions rely not only on software measurement data collected for the development project by an ongoing SEL measurement program, but also on the organizational experience gained on past development projects in the environment which can be used to understand and manage current projects.

1.1 PURPOSE

This document presents a detailed description of the information and algorithms used within the SME to perform these functions for the manager. Its main purpose is to capture how the SME automates key management functions using local data and experience. As a result, the document focuses primarily on the logical steps required to accomplish those functions. Detailed implementation-specific issues (such as standard searching and sorting algorithms, methods of generating menus and windows, or steps for obtaining user input) are not addressed.

The material covered complements information appearing in two previously issued SEL documents—*Software Management Environment (SME) Concepts and Architecture* (Reference 1) and *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules* (Reference 2). Serving as a companion reference, this document provides a bridge between the two earlier documents by illustrating how one can use research results and past experience within the conceptual framework of a software management tool.

1.2 AUDIENCE

This document is intended for use by individuals and organizations interested in understanding the internal algorithms and techniques employed in SME management functions. While the SME has been constructed specifically for the flight dynamics environment at GSFC, the concepts and functionality described in this document readily apply in any software development environment. The SME can serve as a model for other software development organizations wishing to implement a similar measurement-oriented, integrated management tool based on local experience.

Individuals who require only an executive summary of the concepts and functionality of the SME may read the material in each section through the second-level headings. Those readers desiring additional information about SME components and management functions

should read each section through the third-level headings. Those who wish to examine detailed component information and algorithms can reference the entire document for a comprehensive view.

1.3 ORGANIZATION

The remainder of the document is organized as follows:

- Section 2 discusses the major components used to represent information and experience within the tool. These components serve as the elemental building blocks of data referenced by the various SME functions.
- Section 3 describes the major management functions supported by the tool and the algorithms used within those functions. These functions rely on the components described in the previous section for information on an ongoing project as well as for the collective experience from past development efforts.
- Appendix A provides an alphabetic list of all general-purpose and function-specific services defined and referenced in the document.

1.4 NOTATION

Throughout these sections, this document uses a set of standardized conventions to help the reader easily identify items that are discussed in another part of the document. These conventions are as follows:

Convention	Meaning
Measure Model	A shadowed box containing text, within a figure, is used to label information obtained from a major component defined in Section 2. The text appearing in the box identifies the name of the component.
Predicted Schedule	A rounded box containing text, within a figure, is used to label information that derives from a major component or represents an intermediate result. The text appearing in the box identifies the related information.
Convert Date to Phase	Italicized text appearing within the steps of an algorithm refers to a general-purpose or function-specific service, defined elsewhere in the document, that is used with a major component or function. Additional information on the service may be found in the section that addresses its associated component or function. Appendix A cross-references all defined services by name.

SECTION 2—COMPONENTS

Understanding the SME's functionality begins with a firm understanding of the major components used to represent information and experience within the tool. These components serve as the elemental building blocks of data referenced by the various SME functions. When characterized by the source of the information they provide to the SME, these components fall into four categories. The first is project data from the SEL database. This data encompasses measurement and planning data collected as part of ongoing SEL measurement activities for current projects, as well as historical measurements from past projects. The second is research data consisting of models, relationships, and quality definitions that describe the development environment. This information captures the behavior of normal projects in the environment and provides the basis for predicting and estimating key project parameters. The third is management rules that embody knowledge from experienced managers required to analyze measurement data and determine a project's strengths and weaknesses. These rules form the expert analysis portion of the SME and represent lessons learned in interpreting and analyzing metrics collected on past projects. The fourth is management data supplied interactively by users of the SME. This information constitutes additional data intended to support what-if scenarios or to specify subjective knowledge about projects that can only be obtained from the manager.

Table 2-1 summarizes the major components used by the SME, organized into these four basic categories by source.

Table 2-1. Major Components Used by the SME

SOURCE	COMPONENT
Project Data	Project List Measure List Profile List Project/Measure Availability List Project/Profile Availability List Schedule Data Measure Data Profile Data Estimates Data Project Characteristics
Research Data	Schedule Models Measure Models Profile Models Estimate Set Models Attribute Definitions
Management Rules	Knowledge Base Rule Base
Management Data	Alternative Plans Phase Estimates Subjective Data

2.1 PROJECT DATA

The SME relies on the SEL database as the source of project-specific measurement and planning data collected for all software projects within the local development environment. In addition to planned project schedules and estimates, the SEL database includes weekly measurements of basic items such as the effort expended on a project, the size of the ongoing project in both lines of code and number of modules, the amount of computer resources used on a project, the number of errors uncovered, and the number of changes made to the source code. Other information collected and stored in the SEL database covers more detailed measurements of development projects, including items such as number of modules designed, number of open problem reports, the source of software changes and errors, and the amount of time spent uncovering and repairing errors. In short, the SEL database provides a wide spectrum of up-to-date information on current projects, as well as historical information on past projects. Specific details on the various types of information in the database, as well as how that information is collected, may be found in *Data Collection Procedures for the Software Engineering Laboratory (SEL) Database* (Reference 3).

The SME uses project data extracted on a weekly basis from the SEL database in all of its analysis, comparison, prediction, and assessment functions. The data provides the fundamental information that characterizes and describes the behavior of current projects being tracked with the SME. Furthermore, data from completed projects provides an historical reference for making comparisons, creating models, and identifying applicable management rules.

Table 2-2 summarizes the major components referenced by the SME as project data. Each component maps to a particular type of data obtained from the SEL database and is identified with a specific purpose. As a general rule, the first five components serve to identify and locate the project data, while the last five types of components contain project-specific information for each project.

Table 2-2. SME Project Data Components

COMPONENT	PURPOSE
Project List	Identifies the names of all available projects
Measure List	Identifies the set of defined software development measures
Profile List	Identifies the set of defined profiles of each measure
Project/Measure Availability List	Identifies what measure data exists for each project
Project/Profile Availability List	Identifies what profile data exists for each project
Schedule Data	Captures the manager's planned project schedule
Estimates Data	Captures the manager's planned completion estimates
Measure Data	Captures actual project values over time of defined measures
Profile Data	Captures actual project values over time of defined profiles
Project Characteristics	Captures key objective facts that characterize a project

The following sections provide detailed information on each of these components.

2.1.1 Project List

Purpose

Identifies the names of all projects available for access through the SME.

Description

The project list is an alphabetized table containing the names of all projects, both ongoing and completed, that may be examined using the SME. The list defines all available projects that the user may choose as the project of interest. A project name can appear in the list if and only if a file containing schedule data exists for that project. The SME uses the project name as a starting point for identifying, locating, and referencing all project data associated with a project.

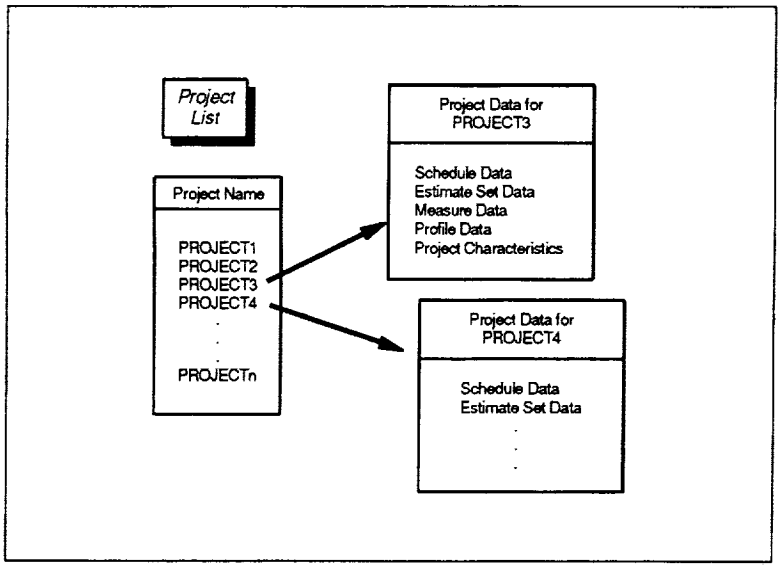


Figure 2-1. Project List for the SME

Source

Created by the SME during initialization based on the existence of project data files

Assumptions

- Each project in the list must have a schedule
- The existence of a schedule for a project implies the existence of an estimate set containing at least one nonzero estimate
- The existence of a schedule also implies the existence of nonzero measure data for at least one measure

Instances

The SME creates one project list, which exists only for the duration of the SME session.

Structure

Table with one column—project name. Each row in the table contains the name of a single project.

2.1.2 Measure List

Purpose

Identifies the set of fundamental software development measures used by the SME.

Description

The measure list is a table containing the names (and codes) of all fundamental software development measures that may be referenced using the SME. The SME defines a set of eight basic measures that managers in this environment use to track and judge project progress. The SME uses the list in locating and referencing the measure data and measure models that are available. Consolidating the names of all defined measures in one list facilitates changing or extending the list to accommodate new measures or other development environments.

Measure List

Measure Code	Measure Name
CPU	CPU Hours
EFF	Total Staff Hours
LOD	Lines of Code
MCH	Modules Changed
MOD	Module Count
RCH	Reported Changes
RER	Reported Errors
RUN	Computer Jobs

Figure 2-2. Measure List for the SME

Source

Defined as part of the SME

Assumptions

- Any measure data accessed by the SME will correspond to one of the defined measures in the list
- A one-to-one mapping exists between the defined measures and the set of measure models used for a given project type
- A one-to-one mapping exists between the defined measures and the entries in an estimate set model

Instances

The SME defines one measure list.

Structure

Table with two columns—measure code and measure name. Each row in the table defines a single measure, identified by a measure code and an associated descriptive measure name.

2.1.3 Profile List

Purpose

Identifies the types of profile data used by the SME for specific measures.

Description

The profile list is a table containing the names, codes, and associated measure of all types of profile data that may be referenced using the SME. Profile data takes an associated measure and breaks it down into two or more discrete categories. Thus, each profile must be associated with a measure. The SME defines a set of four profiles that the software uses primarily in assessing a project's overall health, stability, and reliability. These four profiles can also be used by managers to track and judge a project's progress. The SME uses the list in locating and referencing the profile data and models that are available. Consolidating the names of all defined profiles in one list facilitates changing or extending the list to accommodate new profiles or other development environments.

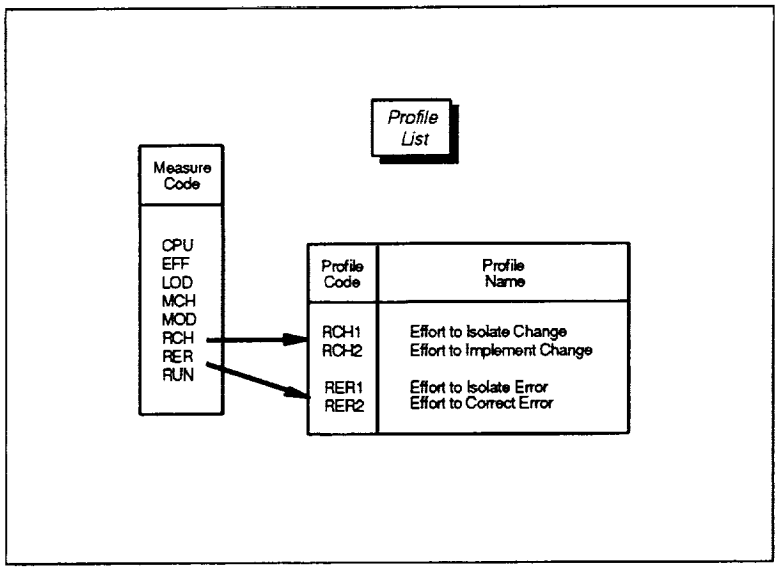


Figure 2-3. Profile List for the SME

Source

Defined as part of the SME

Assumptions

- Any profile data accessed by the SME will correspond to one of the defined profiles in the list
- A one-to-one mapping exists between the defined profiles and the set of profile models used for a given project type

Instances

The SME defines one profile list.

Structure

Table with three columns—measure code, profile code, and profile name. Each row in the table defines a single profile, identified by a unique profile code, its associated measure, and an associated descriptive profile name.

2.1.4 Project/Measure Availability List

Purpose

Identifies what measure data exists for each project.

Description

The project/measure availability list is a table of boolean flags that indicates what measure data is available for each project. Each row in the table contains information related to one project specified in the project list. Each column that is associated with a boolean flag corresponds to one measure defined in the measure list. A measure is flagged as available for a given project if and only if a file containing data for that particular measure and project exists. The SME uses the list in determining what measures are available for a project.

Project/Measure Availability List

Project Names	Measures							
	CPU	EFF	LOC	MCH	MOD	RCH	RER	RUN
PROJECT1	T	T	F	T	T	F	T	T
PROJECT2	F	T	T	T	T	T	T	F
PROJECT3	T	T	T	T	F	T	T	T
PROJECT4	F	T	T	F	T	F	F	F
.				.				
.				.				
.				.				
PROJECTn	T	T	T	T	T	T	T	T

Figure 2-4. Project/Measure Availability List for the SME

Source

Created by the SME during initialization based on the existence of project data files

Assumptions

- A one-to-one mapping exists between the rows in the table and the projects in the project list
- A one-to-one mapping exists between the columns of boolean flags in the table and the measures defined in the measure list
- Table entries flagged as "TRUE" identify the measures that are accessible by the SME for a given project

Instances

The SME creates one project/measure availability list, which exists only for the duration of the SME session.

Structure

Table of boolean flags with one row for each project in the project list and one column for each defined measure. An individual row in the table indicates which measures are available for the project identified for the row.

2.1.5 Project/Profile Availability List

Purpose

Identifies what profile data exists for each project.

Description

The project/profile availability list is a table of boolean flags that indicates what profile data is available for each project. Each row in the table contains information related to one project specified in the project list. Each column that is associated with a boolean flag corresponds to one profile defined in the profile list. A profile is flagged as available for a given project if and only if a file containing data for that particular profile and project exists. The SME uses the list in determining what profiles are available for a project. Note that if data exists for a given profile, data inherently exists for the profile's associated measure.

Project/Profile Availability List

Project Names	Profiles			
	RCH1	RCH2	RER1	RER2
PROJECT1	T	T	F	T
PROJECT2	F	T	T	T
PROJECT3	T	T	T	T
PROJECT4	F	T	T	F
.		.		
.		.		
.		.		
PROJECTn	T	T	T	T

Figure 2-5. Project/Profile Availability List for the SME

Source

Created by the SME, as needed, based on the existence of project data files

Assumptions

- A one-to-one mapping exists between the rows in the table and the projects in the project list
- A one-to-one mapping exists between the columns of boolean flags in the table and the profiles defined in the profile list
- Table entries flagged as "TRUE" identify the profiles that are accessible by the SME for a given project

Instances

The SME creates one project/profile availability list, which exists only for the duration of the SME session.

Structure

Table of boolean flags with one row for each project in the project list and one column for each defined profile. An individual row in the table indicates which profiles are available for the project identified for the row.

2.1.6 Schedule Data

Purpose

Captures a chronological record of the project's schedule as planned and periodically updated by the manager.

Description

Schedule data is a list of all schedules submitted by a manager for a project over the project's life cycle. The individual schedules in the list are maintained in chronological order by submission date with the most recent submission identifying the default "current" schedule. Each schedule in the list specifies the planned start and end dates of each phase in the software development life cycle. Since the SME follows the SEL database's use of a traditional waterfall life cycle, the SME currently uses a set of four contiguous, non-overlapping phases: design, code and unit testing, system testing, and acceptance testing. By specifying phases in this manner, the schedule implicitly defines the start and end dates for the entire project.

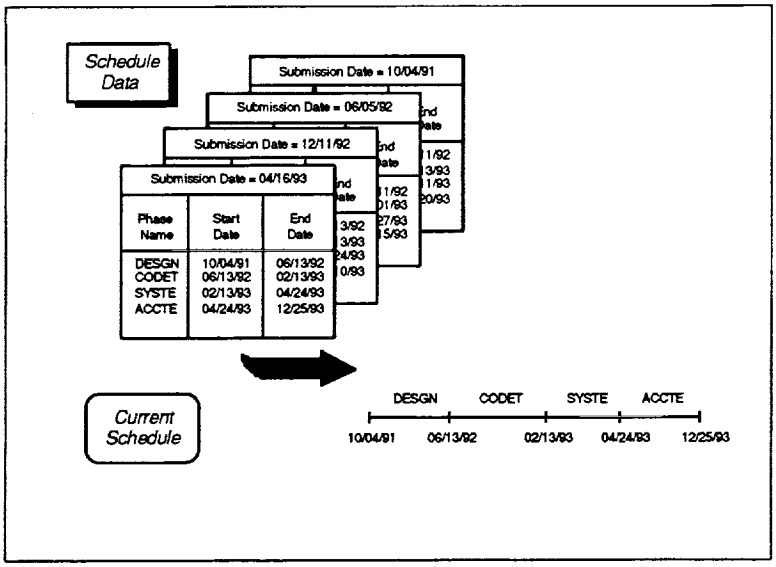


Figure 2-6. Schedule Data for a Project

Source

Collected by the SEL from the manager via Project Estimates Forms (PEFs); subsequently extracted from the SEL database for the SME

Assumptions

- Projects follow a traditional waterfall life cycle with four serial, non-overlapping phases
- The phases in a schedule map to the phases defined in the schedule model for the corresponding project type

Instances

One schedule data file is required for each project.

Structure

Collection of schedule records. Each schedule record consists of a submission date and a table with three columns—phase name, phase start date, and phase end date. Each row in the table supplies the dates for a single phase.

The following section delineates a set of general-purpose services commonly associated with schedule data.

2.1.6.1 General-Purpose Use of Schedule Data

The SME incorporates a set of general-purpose services commonly used with schedule data. The services are requested by various high-level SME functions to perform specific actions associated with schedules. These services include

- *Get Scheduled Phase Dates*—Obtains the start and end dates for a given phase from a specified schedule.
- *Get Project Dates*—Obtains the start and end dates planned for the project from a specified schedule.
- *Get Schedule*—Obtains the schedule that was in effect on a given date (if no date is specified, obtains the most recent schedule).

2.1.7 Measure Data

Purpose

Captures the actual recorded behavior over time of a fundamental software development measure such as lines of code, effort, or software errors.

Description

Measure data is a chronological record of the actual values collected on a project for a single specific measure over the development life cycle. For any given project, the SME references measure data for one or more of the eight key measures defined in the measure list. The measure values in the data are zero at the start of a project and cumulative measure values to date are subsequently recorded at a fixed sampling frequency. By convention, the SME uses measure data recorded on a weekly basis to match the sampling frequency of SEL data collection activities. The measure values stop at the most recent sampling date for ongoing projects, but continue through the end of the project for completed projects.

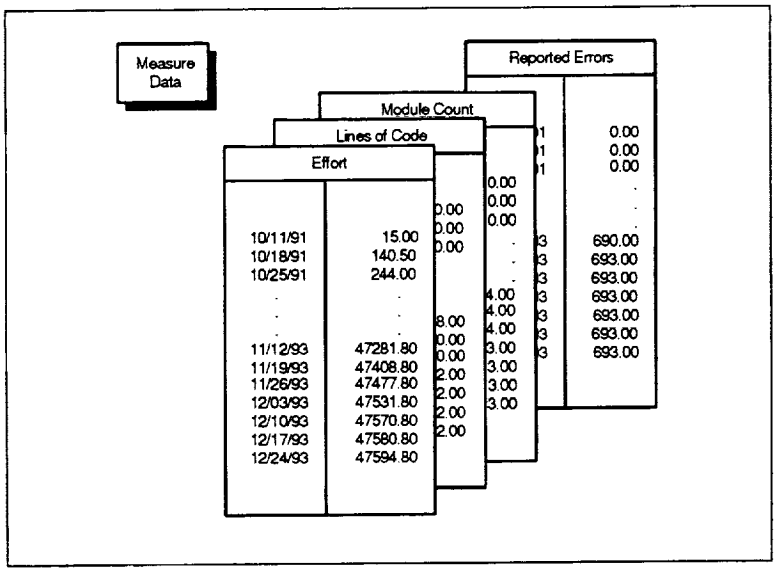


Figure 2-7. Measure Data for a Project

Source

Collected by the SEL via forms and automated data collection tools; subsequently extracted from the SEL database for the SME

Assumptions

- At project start, all measure values are zero
- The measure values are recorded on a weekly basis with one value per week (no time gaps exist in the data)
- Each project collects measure data for at least one measure

Instances

One measure data file may exist for each defined measure per project, as noted in the project/measure availability list.

Structure

Table with two columns—date of sample, measure value. Each row in the table describes the actual cumulative value observed for the measure on the sampling date.

2.1.7.1 Representative Measure Data

The SME references measure data for one or more of the eight defined measures for each project. This data encompasses

- Effort Data
- Lines of Code Data
- Module Count Data
- Computer Hours Data
- Computer Runs Data
- Changed Modules Data
- Reported Changes Data
- Reported Errors Data

The following sections present a representative set of data for the eight measures. The samples depict measurements for an ongoing project.

2.1.7.1.1 Effort Data

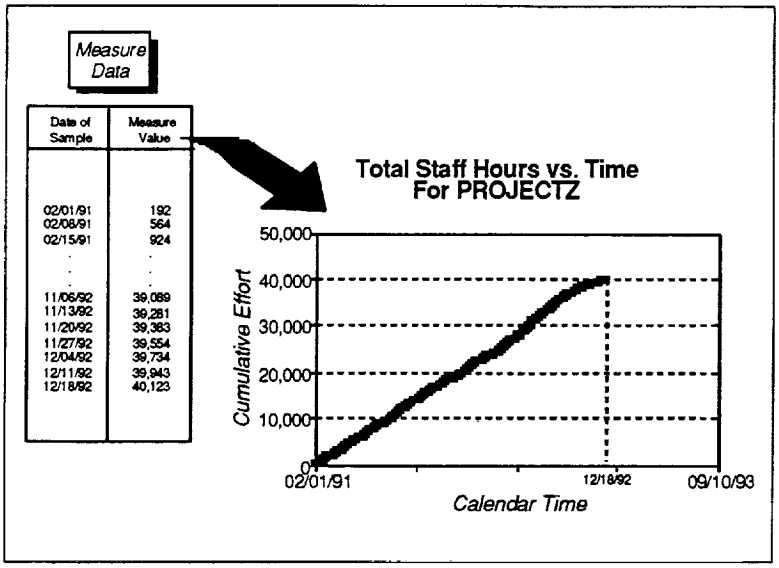


Figure 2-8. Effort Data for an Ongoing Project

Effort data provides weekly measurements of the actual expenditure of effort in staff hours on a project. The effort represents all hours expended by programmers and line management, but excludes all project management and service hours. The information is collected via SEL Personnel Resource Forms (PRFs).

Note: The measurements will typically cover the entire development life cycle from project start through project end.

2.1.7.1.2 Lines of Code Data

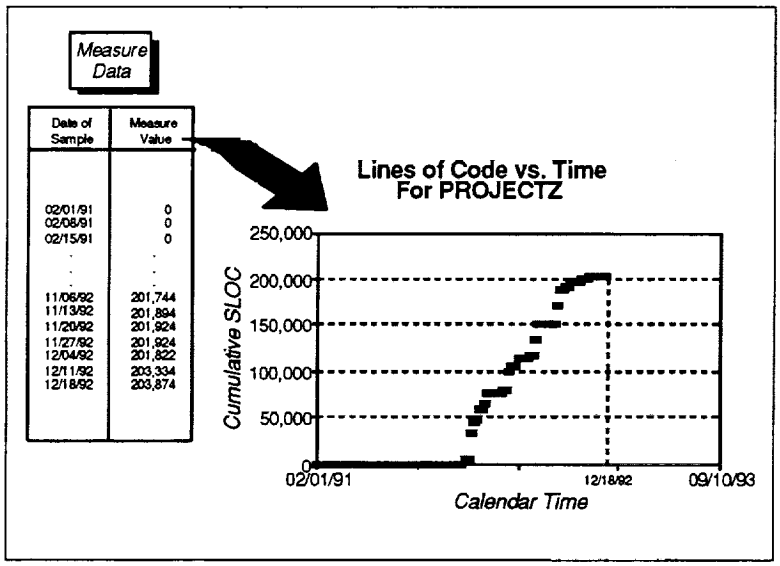


Figure 2-9. Lines of Code Data for an Ongoing Project

Lines of code data provides weekly measurements of the actual generation of lines of code in SLOC on a project. This measure reflects the number of records in the project's controlled source library. The information is collected via an automated tool that examines project libraries and is recorded on SEL Services/Products Forms (SPFs).

Note: The measurements will remain at zero until the project begins placing source code under configuration control in the project's source library. This typically occurs near the beginning of the code and unit test phase.

2.1.7.1.3 Module Count Data

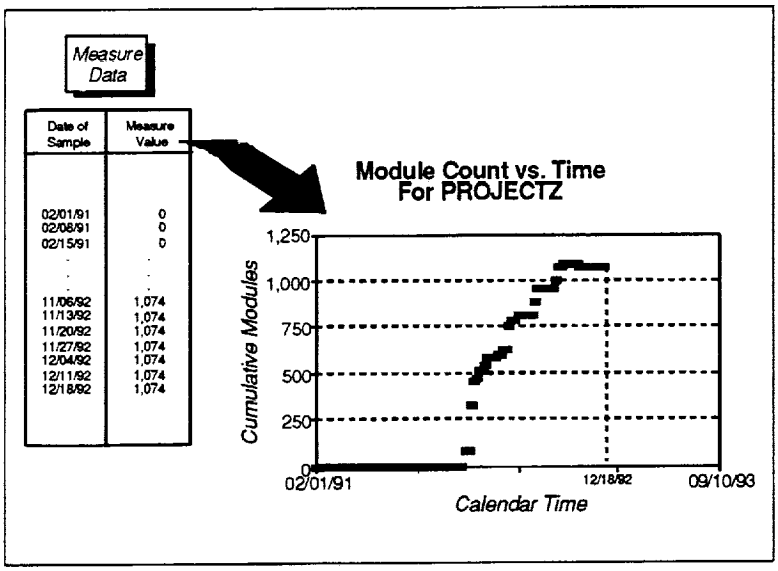


Figure 2-10. Module Count Data for an Ongoing Project

Module count data provides weekly measurements of the actual number of modules generated on a project. This measure reflects the number of members in the project's controlled source library. The information is collected via an automated SEL tool that examines project libraries and is recorded on SEL SPFs.

Note: The measurements will remain at zero until the project begins placing source code under configuration control in the project's source library. This typically occurs near the beginning of the code and unit test phase.

2.1.7.1.4 Computer Hours Data

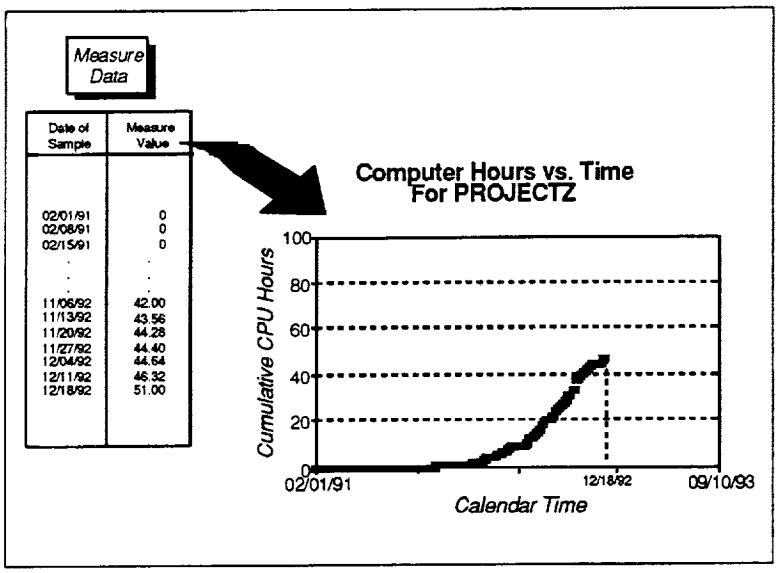


Figure 2-11. Computer Hours Data for an Ongoing Project

Computer hours data provides weekly measurements of actual computer usage in CPU hours by a project. This measure reflects values from all computers used by the project, normalized to account for different processor speeds. The information is collected by computer system accounting software and recorded on SEL SPFs.

Note: These measurements are particularly sensitive to the development process being applied, but do exhibit useful trends within similar classes of projects.

2.1.7.1.5 Computer Runs Data

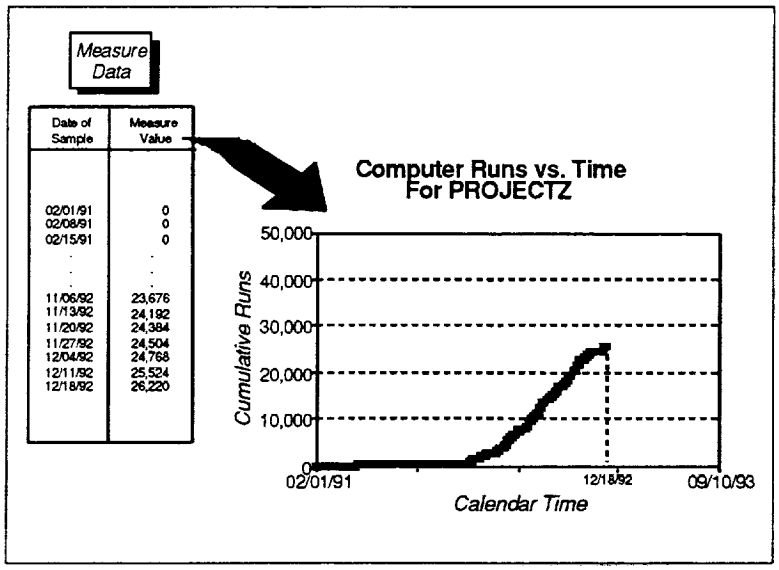


Figure 2-12. Computer Runs Data for an Ongoing Project

Computer runs data provides weekly measurements of actual computer usage in terms of the number of jobs submitted by a project. This measure reflects jobs submitted on all computers used by the project. The information is collected by computer system accounting software and is recorded on SEL SPFs.

Note: These measurements are particularly sensitive to the development process being applied, but do exhibit useful trends within similar classes of projects.

2.1.7.1.6 Changed Modules Data

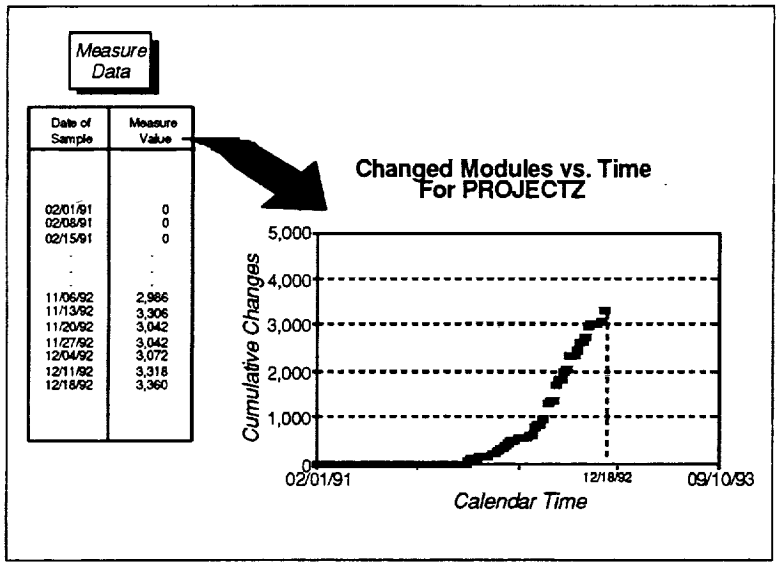


Figure 2-13. Changed Modules Data for an Ongoing Project

Changed modules data provides weekly measurements of the actual number of module changes occurring on a project. This measure reflects the number of module versions in the project's source library, minus the number of baseline members. The information is collected via an automated SEL tool that examines project libraries and is recorded on SEL SPFs.

Note: The measurements will remain at zero until the project begins modifying source code that resides under configuration control in the project's source library. This typically occurs in the code and unit test phase.

2.1.7.1.7 Reported Changes Data

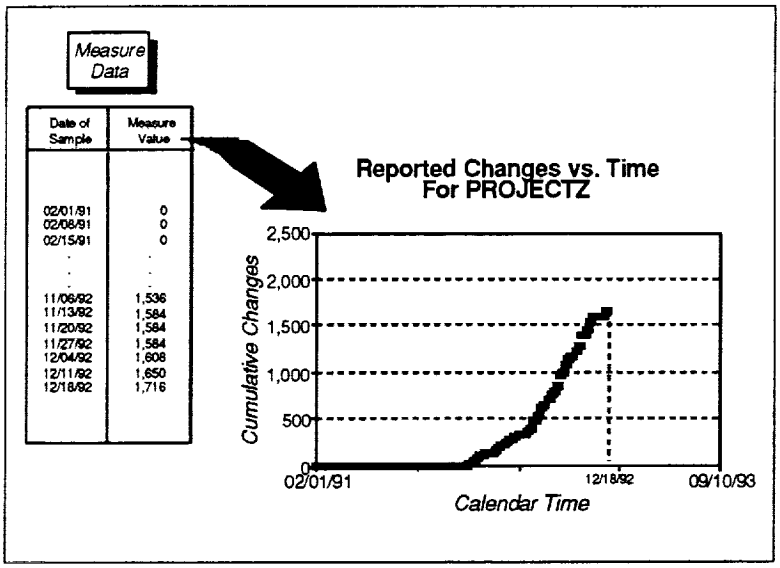


Figure 2-14. Reported Changes Data for an Ongoing Project

Reported changes data provides weekly measurements of the actual number of logical changes reported for a project. This measure reflects the number of SEL Change Report Forms (CRFs) submitted to date for a project.

Note: The measurements will remain at zero until the project begins modifying source code that resides under configuration control in the project's source library. This typically occurs in the code and unit test phase.

2.1.7.1.8 Reported Errors Data

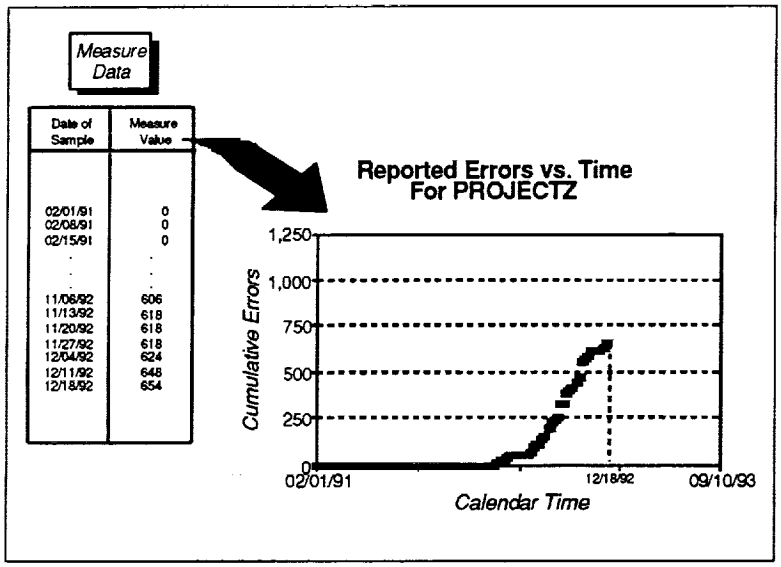


Figure 2-15. Reported Errors Data for an Ongoing Project

Reported errors data provides weekly measurements of the actual number of logical changes reported as being due to an error that occurred on a project. This measure reflects the number of SEL CRFs submitted to date on which the type of change is listed as error correction.

Note: Reported error measurements will remain at zero until the project begins correcting source code that resides under configuration control in the project's source library. This typically occurs in the code and unit test phase.

2.1.8 Profile Data

Purpose

> Captures the actual recorded behavior over time of a software development measure using an associated profile such as effort to isolate changes or effort to correct errors.

Description

Profile data is a decomposition into discrete categories of a particular development measure to further characterize that measure's behavior over the development life cycle. The profile values in the data are zero at the start of a project and cumulative profile values to date are subsequently recorded at a fixed sampling frequency. As with measure data, the SME uses profile data recorded on a weekly basis to match the sampling frequency of SEL data collection activities. The profile values stop at the most recent sampling date for ongoing projects, but continue through the end of the project for completed projects.

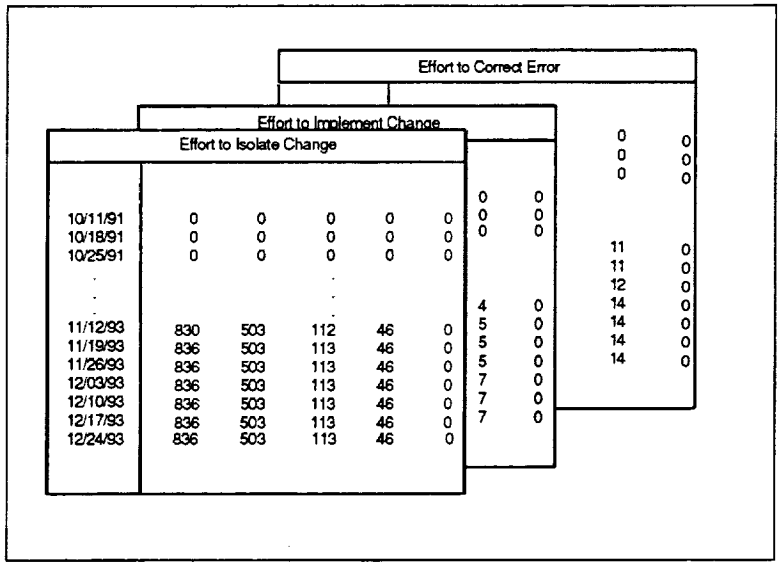


Figure 2-16. Profile Data for a Project

Source

Collected by the SEL via forms; subsequently extracted from the SEL database for the SME

Assumptions

- At project start, all profile values are zero
- The profile values are recorded on a weekly basis with one value per week (no time gaps exist in the data)
- The existence of profile data associated with a measure implies that measure data exists for that measure

Instances

One profile data file may exist for each defined profile per project, as noted in the project/profile availability list.

Structure

Table with multiple columns—date of sample, and one column per profile value. Each row in the table describes the actual cumulative values observed in the profile's defined categories on the sampling date. Additionally, the horizontal sum of the profile values taken on any given date will equal the observed value of the associated measure on the same date.

2.1.8.1 Representative Profile Data

The SME references profile data for up to four defined profiles for a project. This data encompasses

- Effort to Isolate Change Data
- Effort to Implement Change Data
- Effort to Isolate Error Data
- Effort to Correct Error Data

The following sections present a representative set of data for the four profiles. The samples depict measurements for an ongoing project.

2.1.8.1.1 Effort to Isolate Change Data

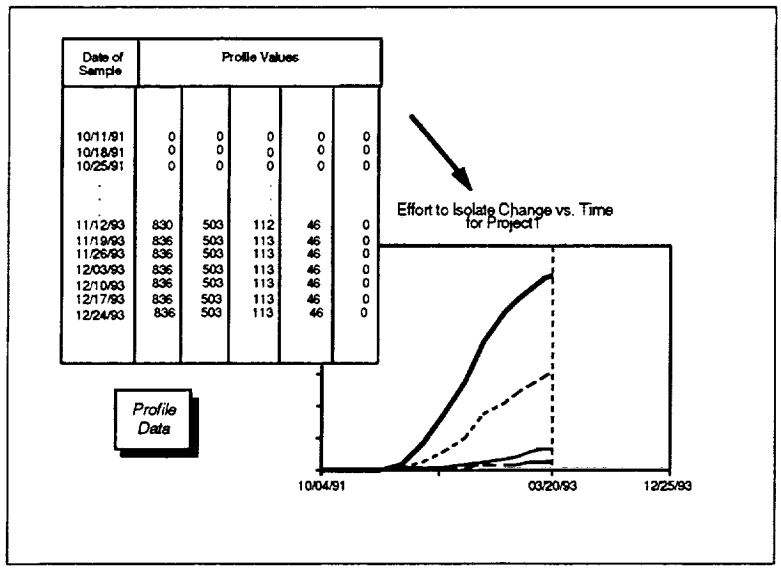


Figure 2-17. Effort to Isolate Change Data for an Ongoing Project

Effort to isolate change data provides weekly measurements that record reported changes by the effort expended in isolating the change. The profile partitions the reported change data into five categories—1 hour or less, 1 day to 1 hour, 3 days to 1 day, more than 3 days, and unknown. The information is collected via SEL CRFs.

Note: The measurements will remain at zero until the project begins modifying source code that resides under configuration control in the project's source library.

2.1.8.1.2 Effort to Implement Change Data

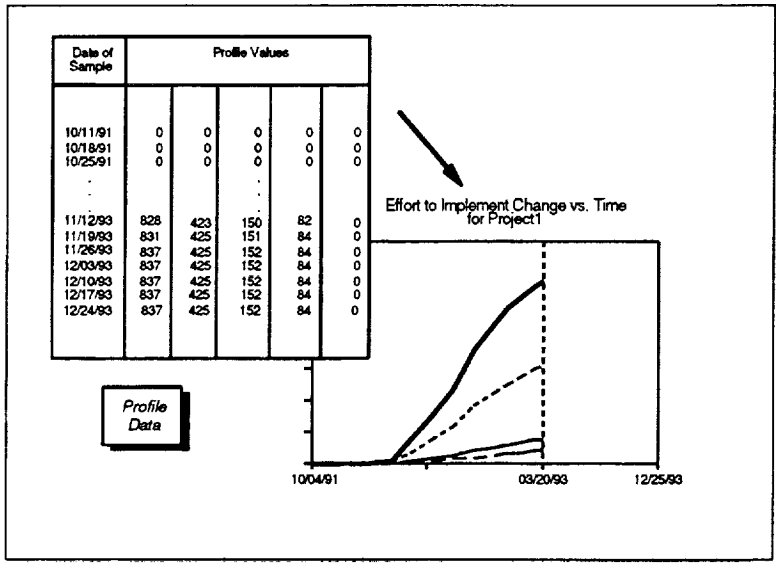


Figure 2-18. Effort to Implement Change Data for an Ongoing Project

Effort to implement change data provides weekly measurements that record reported changes by the effort expended in implementing the change. The profile partitions the reported change data into five categories—1 hour or less, 1 day to 1 hour, 3 days to 1 day, more than 3 days, and unknown. The information is collected via SEL CRFs.

Note: The measurements will remain at zero until the project begins modifying source code that resides under configuration control in the project's source library.

2.1.8.1.3 Effort to Isolate Error Data

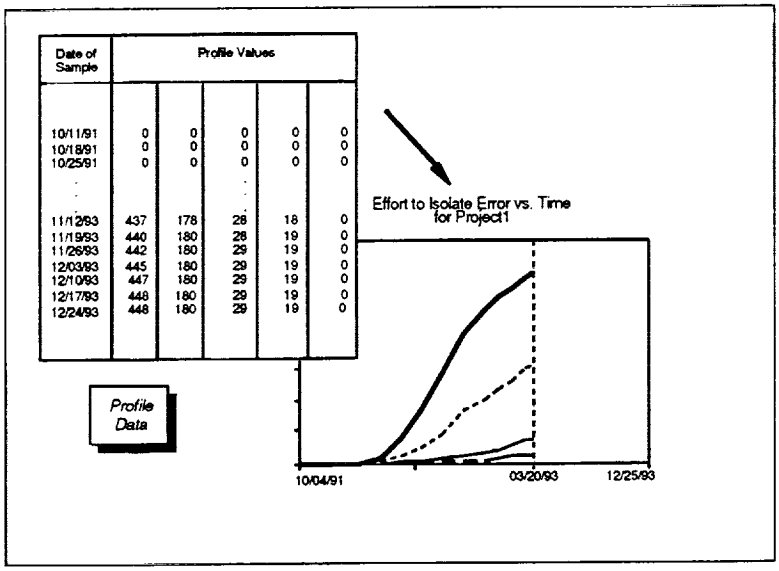


Figure 2-19. Effort to Isolate Error Data for an Ongoing Project

Effort to isolate error data provides weekly measurements that record reported errors by the effort expended in isolating the error. The profile partitions the reported error data into five categories—1 hour or less, 1 day to 1 hour, 3 days to 1 day, more than 3 days, and unknown. The information is collected via SEL CRFs.

Note: The measurements will remain at zero until the project begins correcting source code that resides under configuration control in the project's source library.

2.1.8.1.4 Effort to Correct Error Data

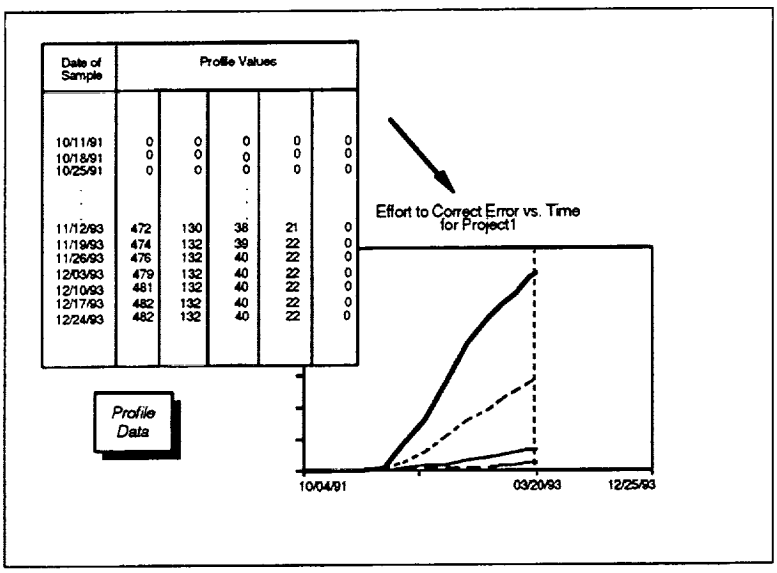


Figure 2-20. Effort to Correct Error Data for an Ongoing Project

Effort to correct error data provides weekly measurements that record reported errors by the effort expended in correcting the error. The profile partitions the reported error data into five categories—1 hour or less, 1 day to 1 hour, 3 days to 1 day, more than 3 days, and unknown. The information is collected via SEL CRFs.

Note: The measurements will remain at zero until the project begins correcting source code that resides under configuration control in the project's source library.

2.1.9 Estimates Data

Purpose

Captures a chronological record of the project's completion estimates as planned and periodically updated by the manager.

Description

Estimates data is a list of all completion estimates submitted by a manager for a project's measures over the development life cycle. The individual estimate sets in the list are maintained in chronological order by submission date, with the most recent submission identifying the "current" set of estimates. Each estimate set in the list specifies the planned completion values for all defined measures. The completion values for specific measures in an estimate set may be set to zero to indicate that the manager does not plan to collect that measure; however, at least one measure in each set of estimates must have a nonzero value.

Note: Estimates are collected from the manager for only three of the eight defined measures (lines of code, module count, and effort). Since the SME requires a manager's estimate for each measure, estimated completion values for the remaining five measures are derived by applying an estimate set model to the three values collected from the manager.

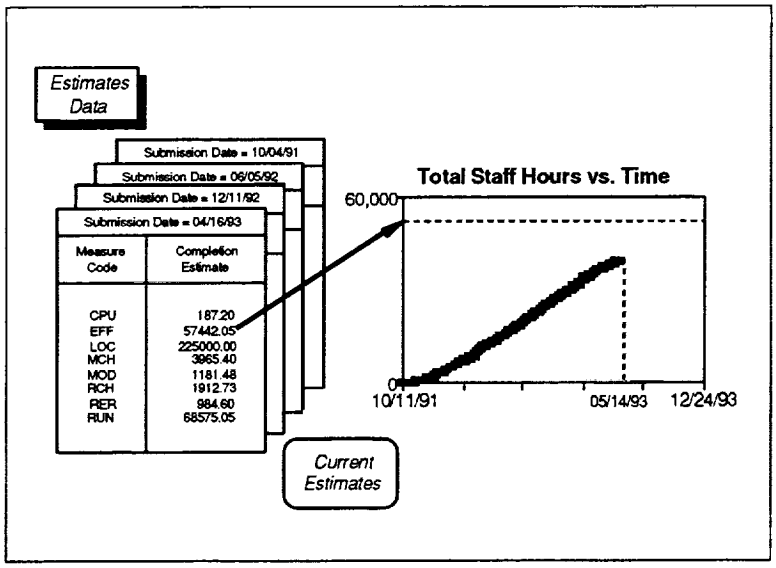


Figure 2-21. Estimates Data for a Project

Source

Collected by the SEL from the manager via SEL PEFs; subsequently extracted from the SEL database for the SME

Assumptions

- A completion value is provided in each estimate set for all measures defined in the measure list
- At least one entry in each estimate set must have a nonzero completion value

Instances

One estimates data file is required for each project.

Structure

Collection of estimate set records. Each estimate set record consists of a submission date and a table with two columns—measure code and estimated completion value. Each row in the table supplies the completion estimate for a single measure.

2.1.9.1 General-Purpose Use of Estimates Data

The SME incorporates a set of general-purpose services commonly used with estimates data. The services are requested by various high-level SME functions to perform specific actions associated with completion estimates. These services include

- *Get Estimated Completion Value*—Obtains the estimated completion value for a given measure from a specified set of estimates.
- *Get Estimates*—Obtains the set of completion estimates that was in effect on a given date (if no date is specified, obtains the most recent set of estimates).

2.1.10 Project Characteristics

Purpose

Captures a collection of key objective facts about a project that helps to characterize the project.

Description

Project characteristics data is a list of known, objective information about a project that can help to classify the project. When taken in aggregate, the key characteristics of a project compose the project type. The SME uses this project type to identify an appropriate set of models that corresponds to the project. Currently, the SME recognizes three basic characteristics—the development computer, the computer language used, and the application area. These key characteristics serve to identify the two primary classes of development projects found in the SEL environment—attitude ground support systems (AGSSs) developed in FORTRAN on IBM computers, and simulators developed in Ada on DEC computers. If the specified project characteristics fail to match these two classes or no characteristics exist for a project, the SME uses a default set of models for that project.

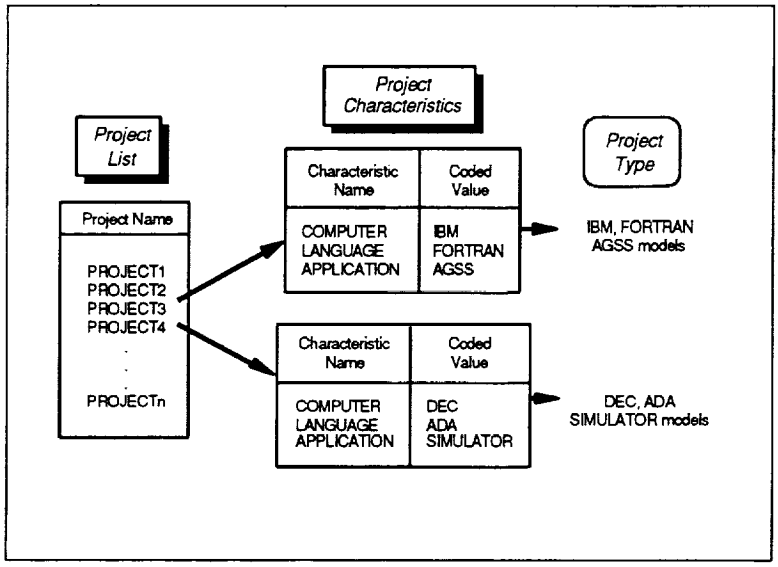


Figure 2-22. Characteristics Data for a Project

Source

Extracted from the SEL database for the SME

Assumptions

- Three key characteristics provide sufficient information to classify most projects and select appropriate models
- Default models can be used with other projects whose characteristics are not known or do not match the supported models

Instances

One project characteristics file may exist for each project.

Structure

Table with two columns—characteristic name and coded value. Each row in the table supplies the coded value for a single characteristic.

2.2 RESEARCH DATA

The SME relies on information from SEL research efforts to identify ways of representing the normal behavior of development projects in the local environment. The models and relationships used within the SME to describe normal projects derive from numerous SEL studies conducted over the years. A summary of representative SEL research results that could be applied to the SME may be found in *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules* (Reference 2).

In this context, the term model refers to a representation of how a given parameter of interest normally behaves over the software development life cycle within a specific environment. This parameter of interest may be time (as in the case of schedule models), a particular measure (as with measure models of effort or lines of code), or even a combination of measures (as in rate models of coding productivity or attribute models of correctability). Typically, these models are developed by averaging historical project data for the parameter over a set of similar, completed projects. The resultant models, normalized for project size, subsequently may be used to represent the behavior normally expected for a class of homogeneous projects having similar project characteristics. The SME currently incorporates models for the two primary classes of development project found in the SEL environment—AGSSs developed in FORTRAN on IBM computers and simulators developed in Ada on DEC computers.

The term relationship, on the other hand, refers to a representation of the correlation between various software development parameters at a specific point in the project life cycle. Due to a need for accurate planning and estimation, most relationships focus on the correlation at project completion between a pair of measures or between duration and a measure. The SME currently incorporates the relationships that exist between the completion values of each pair of measures via estimate set models provided for all supported classes of projects.

Table 2-3 summarizes the major components referenced by the SME as research data and identifies each component's purpose.

Table 2-3. SME Research Data Components

COMPONENT	PURPOSE
Schedule Models	Describes the fractional amount of time normally spent in each life-cycle phase
Measure Models	Describes the normal behavior over time of the defined measures
Profile Models	Describes the normal behavior over time of profiles defined for measures
Estimate Set Models	Describes the relationships that exist between completion values of measures
Attribute Definitions	Describes a defined set of overall project quality attributes

The following sections provide additional detailed information on each of these components.

2.2.1 Schedule Models

Purpose

Describes the amount of time normally spent in each software development life-cycle phase as a fraction of total project duration.

Description

A schedule model is a normalized representation of the fractional amount of calendar time typically expended during a development project as a function of life-cycle phase. Specific points in the life cycle are identified by the combination of a phase name and an elapsed fraction of that phase between 0 and 1.0 inclusive. The amount of time expected at those points is measured from the start of the phase and is expressed as a fraction of elapsed project duration. The sum of the total fractional time spent in all phases is 1.0.

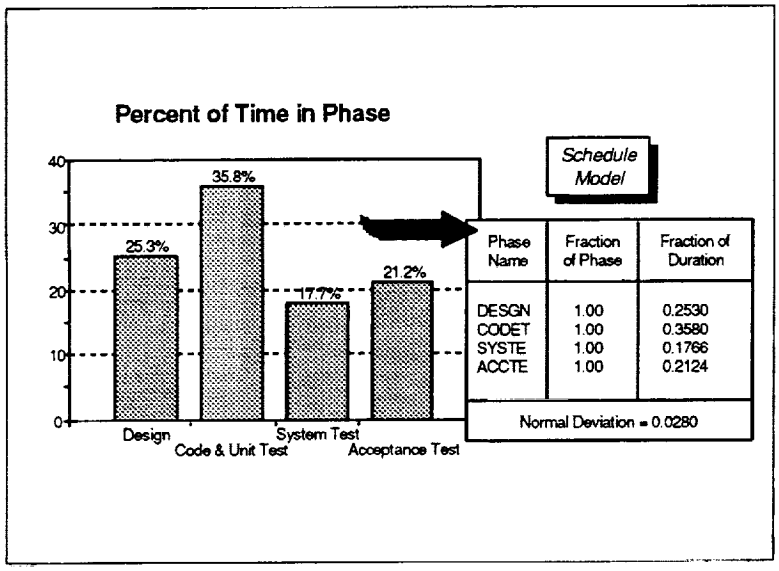


Figure 2-23. Schedule Model for IBM, FORTRAN, AGSS Projects

Source

Statistical averaging of actual schedules from a set of completed development projects

Assumptions

- The projects follow a traditional waterfall life cycle with four serial, non-overlapping phases

Instances

One model exists for each project type.

Structure

Table with three columns—phase name, fraction of phase, and fraction of duration; scalar value—normal deviation. Each row in the table describes the fractional amount of calendar time typically expended from the start of the phase through the point in the life cycle specified by the row's phase name and fraction of phase. The scalar value associated with the table represents the normal allowable deviation in a project's schedule from the tabulated fractional values.

The following sections detail the steps required to create schedule models using actual data from completed projects and present a set of general-purpose algorithms commonly used with schedule models.

2.2.1.1 Creating a Schedule Model

The schedule models used by the SME are created by normalizing and then statistically averaging actual project schedules observed on a set of one or more similar, completed development projects. The projects selected for inclusion in the set should be representative of the type of project to be captured by the model and should have the same number of phases. By first normalizing the schedules, the two-step creation process gives equal weight within the model to each contributing project regardless of size or duration.

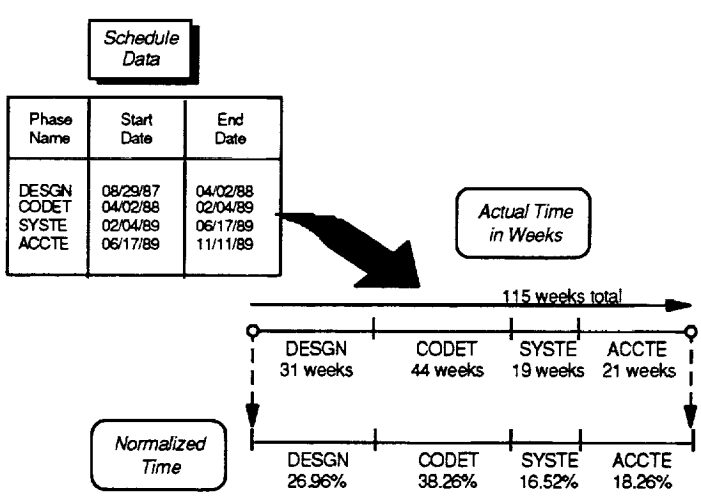
Required Data

- Schedule data (for each project in the set)

Step 1—Normalize Each Project's Schedule

For each project in the set, perform the following:

1. Calculate the actual number of weeks elapsed between the project start date and project completion date found in the schedule data ($Actual\ Weeks_{Total}$).
2. For each life-cycle phase in the schedule data, calculate the actual number of weeks elapsed between the start and end dates of the phase ($Actual\ Weeks_{In\ Phase}[i]$).
3. For each life-cycle phase, normalize the amount of time spent in the phase by the total project duration to compute the fraction of duration for that phase as
$Fraction\ of\ Duration_{In\ Phase}[i] = Actual\ Weeks_{In\ Phase}[i]\ /\ Actual\ Weeks_{Total}$



STEPS

1. Based on the schedule data, the project started on 08/29/87 and ended on 11/11/89 (i.e., start of first phase through end of last phase) for a total elapsed time of 115 weeks.

2. Based on the phase dates in the schedule data, the actual weeks spent in each phase were 31 for DESGN, 44 for CODET, 19 for SYSTE, and 21 for ACCTE.

3. Dividing the actual time spent in each phase by a total duration of 115 weeks gives normalized values of 0.2696, 0.3826, 0.1652, and 0.1826 for the defined phases (i.e., 26.96% DESGN, 38.26% CODET, 16.52% SYSTE, and 18.26% ACCTE).

Figure 2-24. Normalizing a Project's Schedule

Step 2—Average the Normalized Project Schedules

Using the intermediate results from the first step, calculate the normal values to be stored in the schedule model as follows:

1. For each life-cycle phase, average the normalized values calculated for the amount of time spent in that phase by the selected projects using

$$\text{Normal Fraction of Duration}_{\text{In Phase}}[i] = \left(\sum_{j=1}^{N} \text{Fraction of Duration}_{\text{In Phase}}[i,j] \right) / N$$

   for the ith phase, where j refers to projects 1 through N

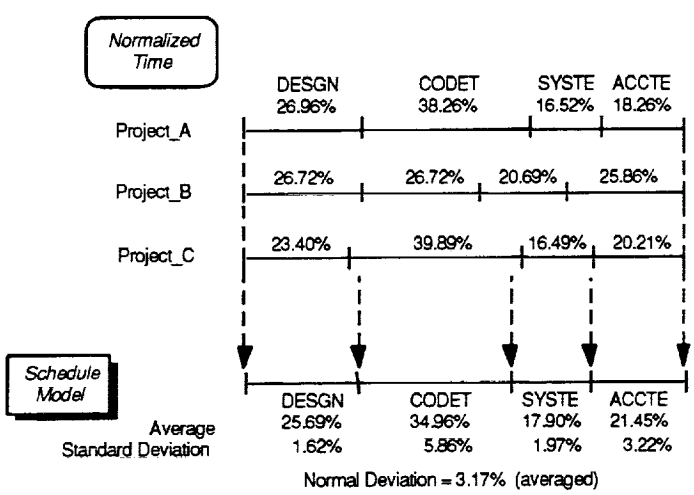
2. For each life-cycle phase, also determine the standard deviation in the normalized values calculated for the amount of time spent in phase from the average for the set of projects using

$$\text{Standard Deviation}_{\text{In Phase}}[i] = \sqrt{\left(\sum_{j=1}^{N} X[i,j] \right) / (N-1)}$$

   where $X[i,j] = (\text{Fraction of Duration}_{\text{In Phase}}[i,j] - \text{Normal Fraction of Duration}_{\text{In Phase}}[i])^2$

3. Calculate the normal deviation in the model by averaging the values of the standard deviation computed for each phase, *1* through *K*, using

$$\text{Normal Deviation} = \left(\sum_{i=1}^{K} \text{Standard Deviation}_{\text{In Phase}}[i] \right) / K$$



STEPS

1. For each life-cycle phase, the average amount of time spent in the phase is calculated. SYSTE phase values of 16.52%, 20.69%, and 16.49% for the three projects result in an average of 17.90% for that phase.

2. For each phase, the standard deviation in the normalized project values from the average is also calculated. Given the three values in SYSTE, a standard deviation of 1.97% may be computed for the phase.

3. Averaging the standard deviation computed for each phase (i.e., 1.62%, 5.86%, 1.97%, and 3.22%) results in a normal deviation of 3.17% for the model.

Figure 2-25. Averaging Normalized Schedules

2.2.1.2 General-Purpose Use of Schedule Models

The SME incorporates a set of general-purpose services commonly used with schedule models. These services are referenced freely throughout various high-level SME functions to provide needed functions associated with schedule models. The services include

- *Convert Phase to Date*
- *Convert Date to Phase*
- *Determine Normal Schedule*

The following sections discuss each of these services and detail the algorithms behind the actions they perform.

2.2.1.2.1 Convert Phase to Date

Purpose

Translates a phase name and elapsed fraction of phase into the calendar date on which the project can normally be expected to reach that phase.

Required Data

- Phase name and elapsed fraction of phase (input value)
- Project start and end dates (input value)
- Schedule model

Steps

1. Referencing the schedule model, calculate the total cumulative fraction of the project's duration normally expected through the specified phase as

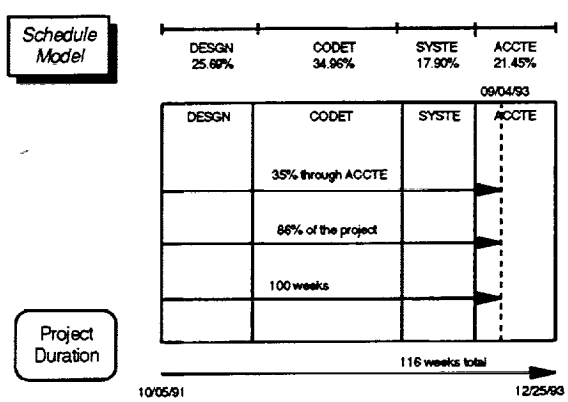
$$\text{Cumulative Fraction of Duration} = \sum_{i=1}^{k-1} \text{Fraction of Duration}_{\text{In Phase}}[i] + F * \text{Fraction of Duration}_{\text{In Phase}}[k]$$

for the kth phase and an elapsed fraction of phase equal to F

2. Calculate the planned project duration as the number of weeks between the project start and end dates. Scale the fractional value from the model by the duration in weeks to obtain the expected number of weeks into the project.

$$\text{Expected Weeks}_{\text{To Phase}} = \text{Cumulative Fraction of Duration} * \text{Project Weeks}_{\text{Total}}$$

3. Add the expected weeks to the project start date to get the expected calendar date.



STEPS

1. Based on the schedule model, 35% through acceptance testing is normally 86.2% of the total project duration (i.e., DESGN at 25.69% plus CODET at 34.96% plus SYSTE at 17.90% plus 35% of ACCTE at 21.45%).

2. Scaling this value by a project duration of 116 weeks, translates to 100 weeks into the project.

3. Adding the 100 weeks to the known project start date results in an expected calendar date of 09/04/93 for the specified phase.

Figure 2-26. Converting an Expected Phase to a Date

2.2.1.2.2 Convert Date to Phase

Purpose

Translates a calendar date specified between the project start and end dates into the phase name and elapsed fraction of phase that normally should be reached on that date.

Required Data

- Calendar date (input value)
- Project start and end dates (input value)
- Schedule model

Steps

1. Divide the weeks between the project start date and the calendar date by the total weeks between the project start and end dates to obtain the fraction of duration.

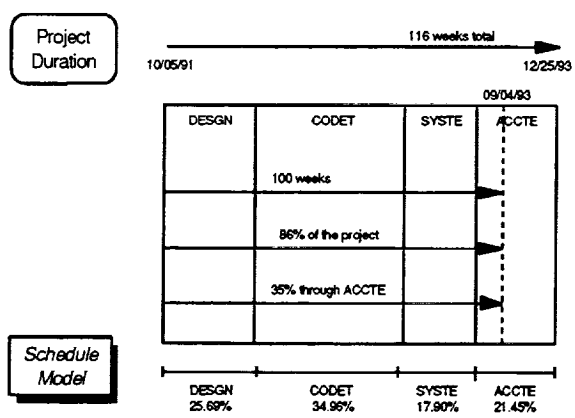
   $$\text{Fraction of Duration}_{\text{To Date}} = \text{Project Weeks}_{\text{To Date}} \;/\; \text{Project Weeks}_{\text{Total}}$$

2. Identify the phase in which the calendar date falls by serially examining the schedule model to locate the first phase k that satisfies the following:

   $$\text{Fraction of Duration}_{\text{To Date}} <= \sum_{i=1}^{k} \text{Fraction of Duration}_{\text{In Phase}}[i]$$

3. Linearly interpolate the fraction of phase k that corresponds to the calendar date as

   $$\text{Fraction of Phase} = \left(\text{Fraction of Duration}_{\text{To Date}} - \sum_{i=1}^{k-1} \text{Fraction of Duration}_{\text{In Phase}}[i]\right) \Big/ \text{Fraction of Duration}_{\text{In Phase}}[k]$$



STEPS

1. Given the project start and end dates, 09/04/93 occurs after 100 weeks; this corresponds to 86.2% of the 116-week project duration.

2. The schedule model shows 86.2% of the project duration falling in ACCTE.

3. Since 86.2% of the project duration exceeds the end of SYSTE by 7.6% (i.e., 86.2% minus DESGN at 25.69% minus CODET at 34.96% minus SYSTE at 17.90%), the date should occur at a point 35% into ACCTE (i.e., 7.6% divided by 21.45% in ACCTE).

Figure 2-27. Converting a Date to an Expected Phase

2.2.1.2.3 Determine Normal Schedule

Purpose

Scales the schedule model on the basis of a project's planned duration to generate a schedule that is considered normal for the project.

Required Data

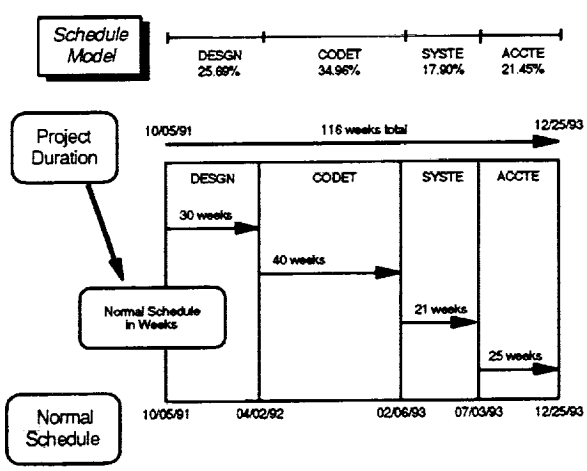
- Project start and end dates (input value)
- Schedule model

Steps

1. Calculate the planned project duration as the number of weeks between the project start and end dates ($Project\ Weeks_{Total}$).
2. For each life-cycle phase, scale the fraction of duration found in the schedule model for the phase by the planned project duration to obtain the number of weeks normally spent in the phase.

   $Normal\ Weeks_{In\ Phase}[i] = Fraction\ of\ Duration_{In\ Phase}[i] * Project\ Weeks_{Total}$

3. Beginning with the project start date, iteratively calculate the start and end dates of the life-cycle phases by incrementing the dates to account for the number of weeks normally spent in each phase.



NOTE

The figure depicts scaling a schedule model to reflect an expected total project duration of 116 weeks. In this instance, the schedule model is applied directly, as if it were a template, to the given project duration to produce a schedule that is considered normal. The resultant normal schedule indicates that if the project is typical, the DESGN, CODET, SYSTE, and ACCTE phases should take 30, 40, 21, and 25 weeks, respectively.

Given a project start date of 10/05/91, this results in the following end date for each phase: DESGN 4/02/92, CODET 2/06/93, SYSTE 7/03/93, and ACCTE 12/25/93.

Figure 2-28. Determining the Normal Schedule

2.2.2 Measure Models

Purpose

> Describes the normal behavior over time of a fundamental software development measure such as lines of code, effort, or software errors.

Description

A measure model is a normalized representation of the typical behavior of a single specific measure as a function of life-cycle phase. The SME uses a set of eight basic measure models to describe a given type of project. These models map to eight key measures defined for use with the SME that managers in this environment use to track and judge project progress. As with schedule models, specific points in the life cycle are identified by the combination of a phase name and an elapsed fraction of that phase between 0 and 1.0 inclusive. The measure value expected at those points is measured from the start of the phase and is expressed as a fraction of the total measure value at project completion. The sum across all phases of the total fractional measure value in each phase is 1.0.

Note: The SME models the ratio of any two individual measures (for example, lines of code per hour) by mathematically combining the appropriate pair of measure models to produce a resultant measure model known as a rate model. Section 2.2.2.3.4 details the steps involved in generating rate models.

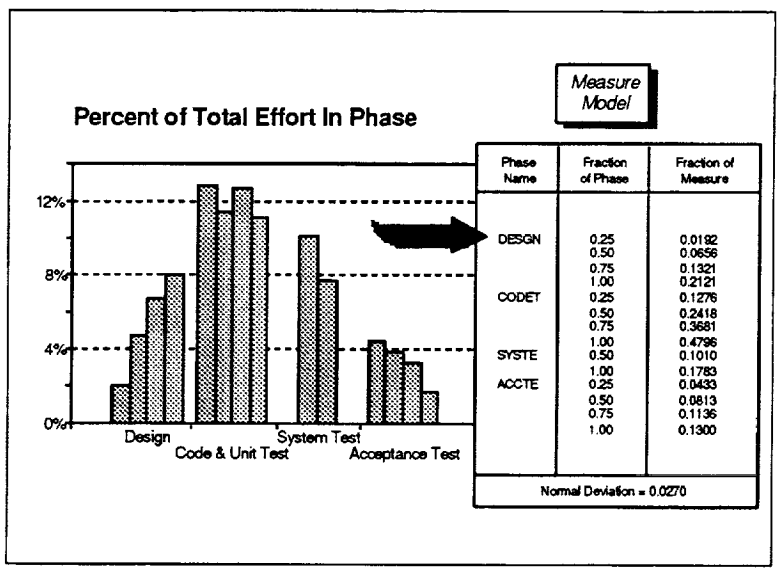


Figure 2-29. Representative Measure Model for IBM, FORTRAN, AGSS Projects

Source

Statistical averaging of actual measure data from a set of completed development projects

Assumptions

- Measure data behavior is dependent on life-cycle phase
- At project start, all measure values are zero

Instances

One model exists for each defined measure, for each project type.

Structure

Table with three columns—phase name, fraction of phase, and fraction of measure; scalar value—normal deviation. Each row in the table describes the fractional amount of the measure typically observed from the start of the phase through the point in the life cycle

specified by the row's phase name and fraction of phase. Since measures typically do not exhibit linear behavior within a phase, each phase is broken into multiple intervals for a total of 14 segments with one per row. The scalar value associated with the table represents the normal allowable deviation in the measure from the tabulated fractional values.

The following sections describe a representative set of measure models, detail the steps required to create any measure model using actual data from completed projects, and present a set of general-purpose algorithms commonly used with measure models.

2.2.2.1 Defined Measure Models

The SME defines a set of eight measure models for each supported project type. These models are

- Effort Model
- Lines of Code Model
- Module Count Model
- Computer Hours Model
- Computer Runs Model
- Changed Modules Model
- Reported Changes Model
- Reported Errors Model

The sample measure models presented below illustrate a complete set of these models for one of the supported project types—IBM, FORTRAN, AGSS projects.

2.2.2.1.1 Effort Model

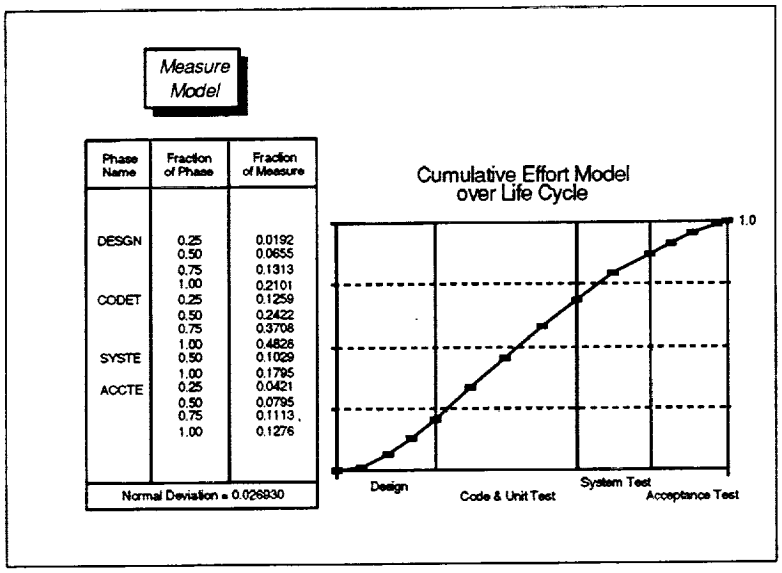


Figure 2-30. Effort Model for IBM, FORTRAN, AGSS Projects

An effort model describes how effort is normally expended as a function of life-cycle phase on a given type of project. The effort represents all staff hours expended by programmers and line management, but excludes all project management and service hours. Each supported effort model is created by statistically averaging actual data from a set of similar, completed projects.

Note: The accumulation of effort over the life cycle inherently exhibits the behavior of a monotonically increasing function.

2.2.2.1.2 Lines of Code Model

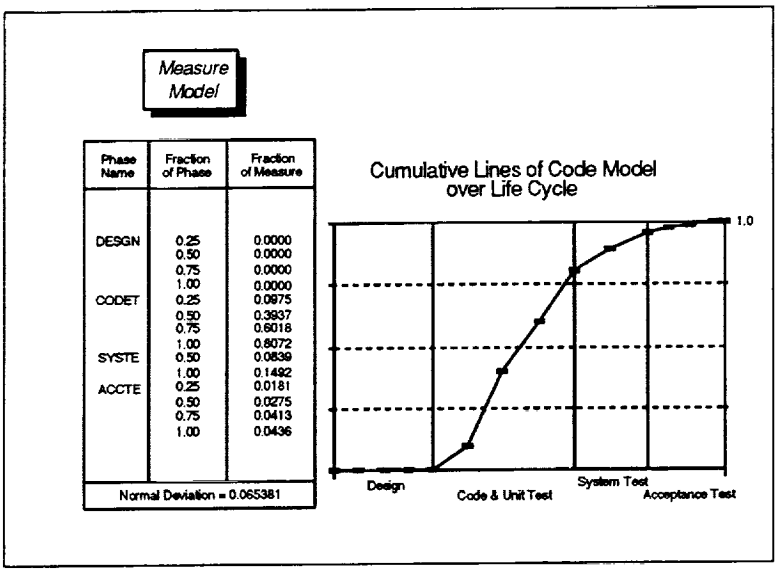


Figure 2-31. Lines of Code Model for IBM, FORTRAN, AGSS Projects

A lines of code model describes how lines of code are normally generated as a function of life-cycle phase on a given type of project. This measure reflects the number of records in the project's source code library. Each supported model is created by statistically averaging actual data from a set of similar, completed projects.

Note: The number of lines of code is expected to be zero until the beginning of the code and test phase. With some projects, the cumulative growth in lines of code may drop due to deletion of obsolete components near the end of the project.

2.2.2.1.3 Module Count Model

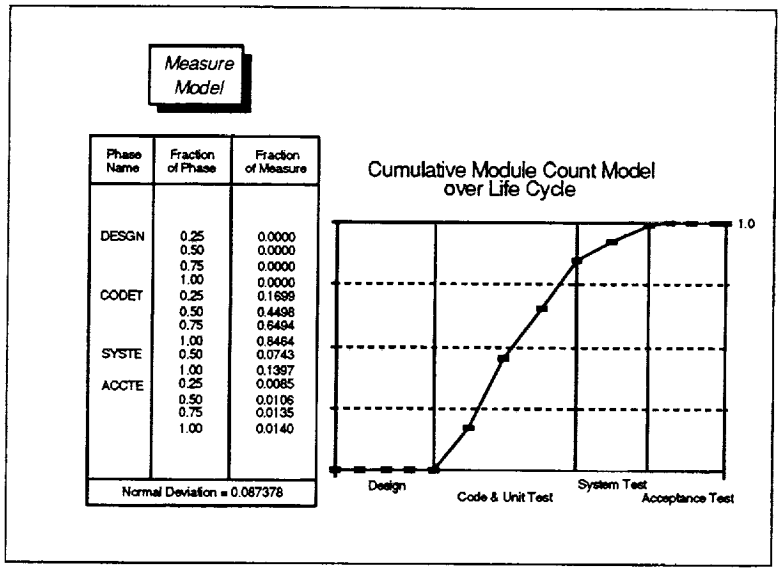


Figure 2-32. Module Count Model for IBM, FORTRAN, AGSS Projects

A module count model describes how the number of components normally grows as a function of life-cycle phase on a given type of project. This measure reflects the number of members in the project's source code library. Each supported model is created by statistically averaging actual data from a set of similar, completed projects.

Note: The module count is expected to be zero until the code and test phase. With some projects, the cumulative count may drop due to deletion of obsolete components near the end of the project.

2.2.2.1.4 Computer Hours Model

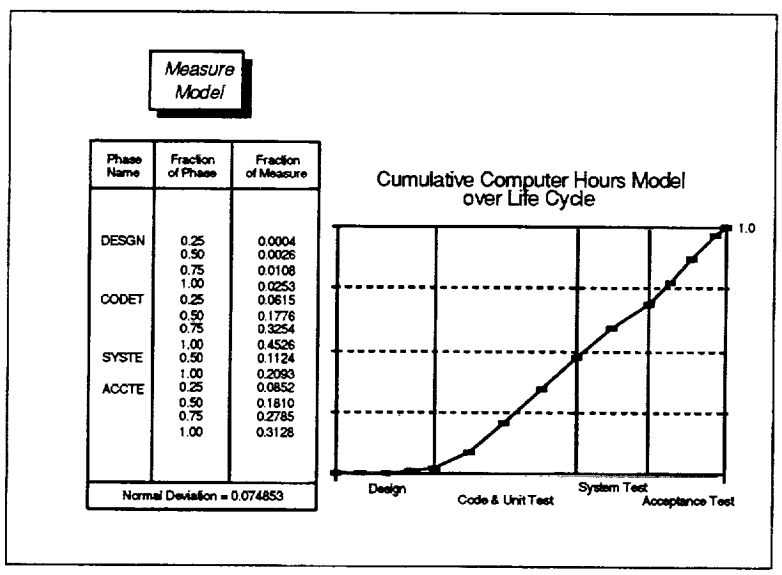


Figure 2-33. Computer Hours Model for IBM, FORTRAN, AGSS Projects

A computer hours model describes the normal usage of computer time in CPU hours as a function of life-cycle phase on a given type of project. This measure reflects values from all computers used by the project, normalized to account for processor speed. Each supported model is created by statistically averaging actual data from a set of similar, completed projects.

Note: The accumulation of computer hours over the life cycle inherently exhibits the behavior of a monotonically increasing function.

2.2.2.1.5 Computer Runs Model

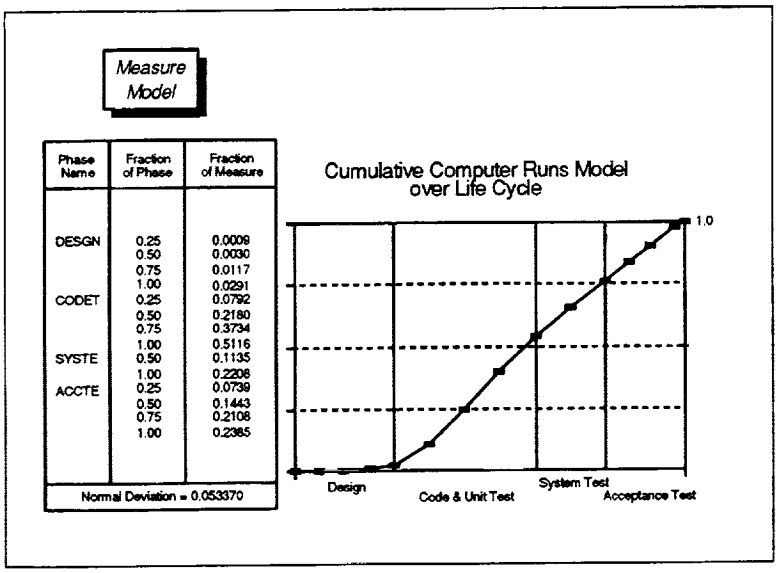


Figure 2-34. Computer Runs Model for IBM, FORTRAN, AGSS Projects

A computer runs model describes the number of computer runs normally observed as a function of life-cycle phase on a given type of project. This measure of computer resource usage reflects the number of jobs submitted on all computers by the project. Each supported model is created by statistically averaging actual data from a set of similar, completed projects.

Note: The accumulation of computer runs over the life cycle inherently exhibits the behavior of a monotonically increasing function.

2.2.2.1.6 Changed Modules Model

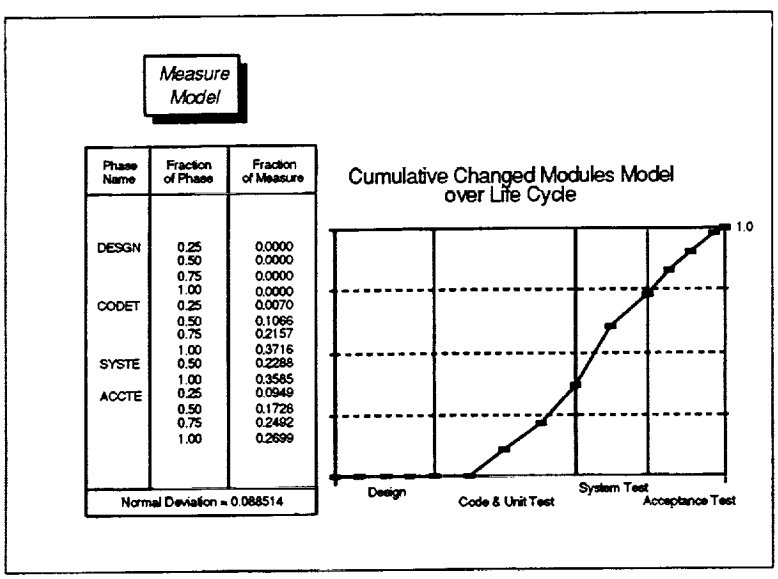


Figure 2-35. Changed Modules Model for IBM, FORTRAN, AGSS Projects

A changed modules model describes how the number of changes normally made to modules varies as a function of life-cycle phase on a given type of project. This measure reflects the number of versions of individual modules in the project's source code library, minus the number of base versions. Each supported model is created by statistically averaging actual data from a set of similar, completed projects.

Note: The number of changed modules is expected to be zero until the beginning of the code and test phase.

2.2.2.1.7 Reported Changes Model

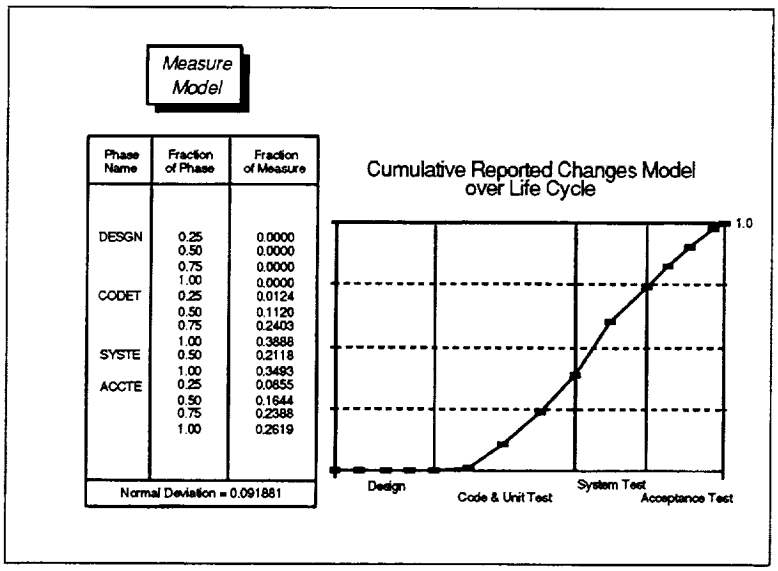


Figure 2-36. Reported Changes Model for IBM, FORTRAN, AGSS Projects

A reported changes model describes the number of logical changes normally made to the software as a function of life-cycle phase on a given type of project. This measure reflects the number of forms submitted to report a logical change to one or more related components. Each supported model is created by statistically averaging actual data from a set of similar, completed projects.

Note: The accumulation of the number of reported changes is expected to be zero until the beginning of the code and test phase.

2.2.2.1.8 Reported Errors Model

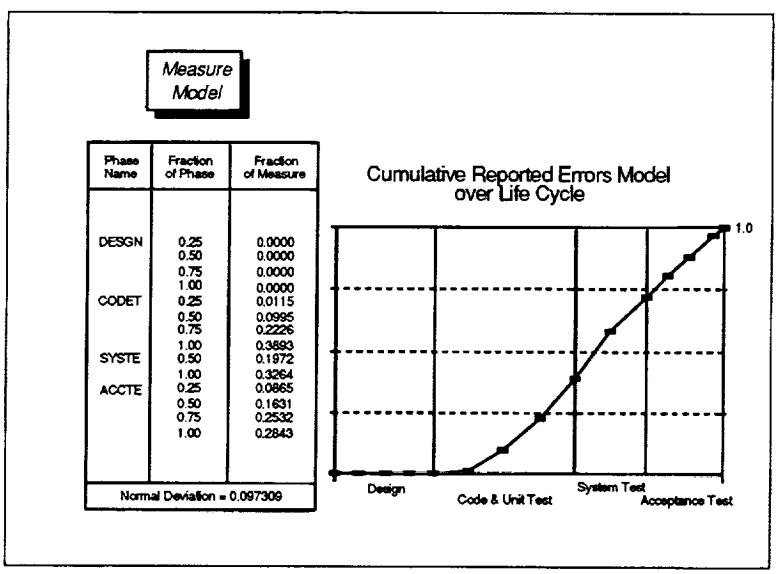


Figure 2-37. Reported Errors Model for IBM, FORTRAN, AGSS Projects

A reported errors model describes the number of logical errors normally found in the software as a function of life-cycle phase on a given type of project. This measure reflects the number of forms submitted to report a logical change that indicate the change was due to an error. Each supported model is created by statistically averaging actual data from a set of similar, completed projects.

Note: The accumulation of the number of reported errors is expected to be zero until the beginning of the code and test phase.

2.2.2.2 Creating a Measure Model

The measure models used by the SME are created by normalizing and then statistically averaging actual project measure data observed on a set of one or more similar, completed development projects. The projects selected for inclusion in the set should be representative of the type of project to be captured by the model. The algorithm may be applied to any defined measure with data. By first normalizing the measurements, the creation process gives equal weight within the model to each contributing project regardless of size or duration.

Required Data

- Schedule data (for each project in the set)
- Measure data (for the measure of interest, for each project in the set)

Step 1—Normalize Each Project's Measure Data

For each project in the set, perform the following:

1. For each life-cycle phase in the schedule data, determine the actual number of weeks from the project start date through the start date of the phase ($Actual\ Weeks_{To\ Phase}[i]$) and calculate the actual number of weeks elapsed between the start and end dates of the phase ($Actual\ Weeks_{In\ Phase}[i]$).

2. For each phase segment to include in the model, calculate the actual number of weeks from project start through the segment as

   $$Week\ Number_{Segment}[i,j] = Actual\ Weeks_{To\ Phase}[i] + F(j) * Actual\ Weeks_{In\ Phase}$$

   for the ith phase and j^{th} segment, where F(j) refers to the fraction of phase of the jth segment

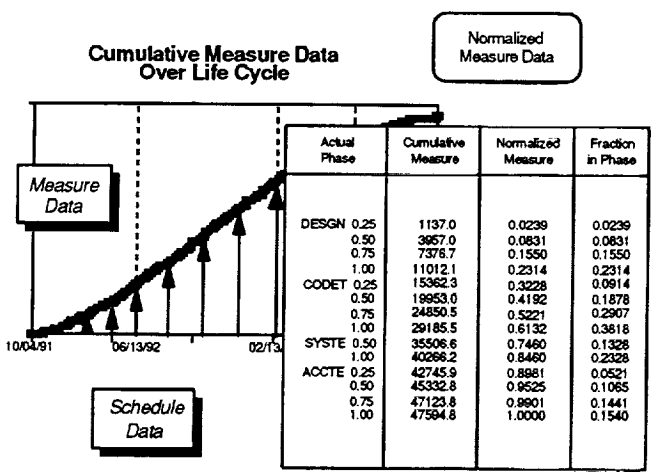
3. For each calculated week number corresponding to the desired phase segments, normalize the actual measure value for that week, measured cumulatively from project start, by the actual total measure value at project completion

   $$Fraction\ of\ Measure_{Segment}[i,j] = Actual\ Measure_{For\ Week}[i,j] \ / \ Actual\ Measure_{Total}$$

4. Adjust each computed fraction of measure value to be cumulative within phase using

   $$Fraction\ of\ Measure_{In\ Phase}[i,j] = Fraction\ of\ Measure_{Segment}[i,j] - Fraction\ of\ Measure_{Segment}[i-1, JMax(i-1)]$$

   for the ith phase and j^{th} segment, where $i > 1$ and JMax(i-1) is the last segment in the $(i-1)^{th}$ phase



Actual Phase	Cumulative Measure	Normalized Measure	Fraction in Phase
DESGN 0.25	1137.0	0.0239	0.0239
0.50	3957.0	0.0831	0.0831
0.75	7376.7	0.1550	0.1550
1.00	11012.1	0.2314	0.2314
CODET 0.25	15362.3	0.3228	0.0914
0.50	19953.0	0.4192	0.1878
0.75	24850.5	0.5221	0.2907
1.00	29185.5	0.6132	0.3818
SYSTE 0.50	35506.6	0.7460	0.1328
1.00	40266.2	0.8460	0.2328
ACCTE 0.25	42745.9	0.8981	0.0521
0.50	45332.8	0.9525	0.1065
0.75	47123.8	0.9901	0.1441
1.00	47594.8	1.0000	0.1540

STEPS

1. The number of weeks in the DESGN, CODET, SYSTE, and ACCTE phases are 30, 40, 21, and 25, respectively.

2. Each phase is broken down into 4, 4, 2, and 4 segments, respectively.

3. The cumulative value at each segment is divided by 47594.8, the cumulative total.

4. Each segment's value is converted to a value that is cumulative within phase.

Figure 2-38. Normalizing a Project's Measure Data

Step 2—Average the Normalized Measure Data

Using the intermediate results from the first step, calculate the normal values to be stored in the measure model as follows:

1. For each life-cycle phase and segment, average the normalized values calculated for the fraction of measure within phase as observed by the selected projects using

$$\text{Normal Fraction of Measure}_{\text{In Phase}}[i,j] = \left(\sum_{k=1}^{N} \text{Fraction of Measure}_{\text{In Phase}}[i,j,k] \right) / N$$

   for the ith phase and j^{th} segment, where k refers to projects 1 through N

2. For each life-cycle phase, also determine the standard deviation in the normalized values calculated for the fraction of measure observed within phase from the average for the set of projects using

$$\text{Standard Deviation}_{\text{In Phase}}[i,j] = \sqrt{\left(\sum_{k=1}^{N} X[i,j,k] \right) / (N-1)}$$

   where $X[i,j,k] = (\text{Fraction of Measure}_{\text{In Phase}}[i,j,k] - \text{Normal Fraction of Measure}_{\text{In Phase}}[i,j])^2$

3. Calculate the normal deviation in the model by averaging the values of the standard deviation computed for each phase segment, *1* through *M*, using

$$\text{Normal Deviation} = \left(\sum_{i,j=1}^{M} \text{Standard Deviation}_{\text{In Phase}}[i,j] \right) / M$$

Normalized Measure Values

Phase	Project 1	Project 2	Project 3
DESGN 0.25	0.0239	0.0164	0.0174
0.50	0.0831	0.0552	0.0582
0.75	0.1550	0.1224	0.1165
1.00	0.2314	0.2119	0.1873
CODET 0.25	0.3228	0.3736	0.3118
0.50	0.4192	0.4852	0.4526
0.75	0.5221	0.6120	0.6089
1.00	0.6132	0.7340	0.7316
SYSTE 0.50	0.7460	0.8028	0.8387
1.00	0.8460	0.8703	0.9009
ACCTE 0.25	0.8981	0.9154	0.9301
0.50	0.9525	0.9542	0.9581
0.75	0.9901	0.9788	0.9823
1.00	1.0000	1.0000	1.0000

Measure Model

Average	Standard Deviation
0.0192	0.0033
0.0655	0.0125
0.1313	0.0169
0.2102	0.0180
0.3361	0.0269
0.4524	0.0269
0.5810	0.0416
0.6929	0.0563
0.7958	0.0381
0.8724	0.0224
0.9145	0.0130
0.9549	0.0405
0.9837	0.0047
1.0000	1.0000

Normal Deviation = 0.0944 (Averaged)

STEPS

1. The normalized measure values at each segment for the three projects are averaged.

2. The standard deviation of each segment value from the respective average is calculated.

3. The average of the standard deviations is calculated to derive a normal deviation of 0.0944.

Figure 2-39. Averaging Normalized Measure Data

2.2.2.3 General-Purpose Use of Measure Models

The SME incorporates a set of general-purpose services commonly used with measure models. The services are referenced freely throughout various high-level SME functions to provide needed functions associated with measure models. These services include

- *Convert Phase to Measure*
- *Convert Measure to Phase*
- *Determine Normal Measure Guidelines*
- *Generate Rate Model*

The following sections discuss each of these services and detail the algorithms behind the actions they perform.

2.2.2.3.1 Convert Phase to Measure

Purpose

Calculates the cumulative measure value that can normally be expected at a given point in the life cycle specified by a phase name and elapsed fraction of phase.

Required Data

- Phase name and elapsed fraction of phase (input value)
- Expected measure value at project completion (input value)
- Measure model

Steps

1. Referencing the measure model, linearly interpolate the cumulative fraction of the measure's value normally expected within the specified phase as

$$\text{Fraction of Measure}_{\text{In Phase}}[k] = \text{Fraction of Measure}_{\text{In Phase}}[k,j-1] + (\text{Fraction of Measure}_{\text{In Phase}}[k,j] - \text{Fraction of Measure}_{\text{In Phase}}[k,j-1]) * (F - F(j-1)) / (F(j) - F(j-1))$$

for the kth phase, the jth segment, and an elapsed fraction of phase, F, where $F(j-1) < F <= F(j)$

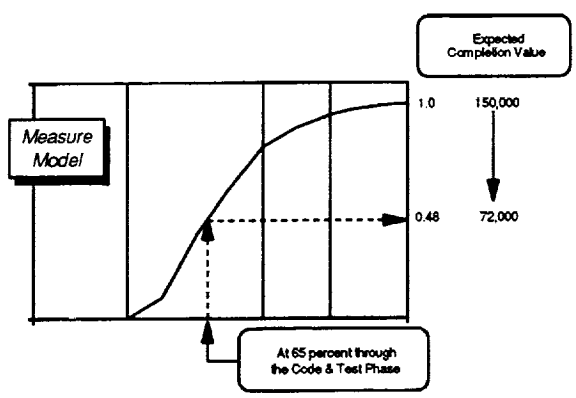
2. Also, from the model, calculate the cumulative fraction of the measure's value normally expected in any earlier phases occurring before the specified phase as

$$\text{Fraction of Measure}_{\text{Before Phase}}[k] = \sum_{i=1}^{k-1} \text{Fraction of Measure}_{\text{In Phase}}[i, JMax(i)]$$

where JMax(i) is the last segment in the ith phase

3. Obtain the expected measure value by scaling the sum of these two computed values by the specified expected measure value at project completion

$$\text{Expected Measure Value} = \text{Expected Completion Value} * (\text{Fraction of Measure}_{\text{In Phase}}[k] + \text{Fraction of Measure}_{\text{Before Phase}}[k])$$



STEPS

1. Using the measure model, at 65% through CODET the measure will normally attain a cumulative measure value equal to 48% of the expected value at project completion.

2. Given an expected project completion value of 150000 for the measure, the normal measure value to expect at this point in the project schedule is 72000 (i.e., 48% of 150000).

Figure 2-40. Converting a Phase to an Expected Measure

2.2.2.3.2 Convert Measure to Phase

Purpose

Calculates an expected phase, specified by a phase name and elapsed fraction of phase, that will normally be reached when the measure of interest attains a given value.

Required Data

- Cumulative measure value (input value)
- Expected measure value at project completion (input value)
- Measure model

Steps

1. Divide the cumulative measure value by the expected measure value at project completion to obtain the fraction of measure at the desired point in the life cycle.

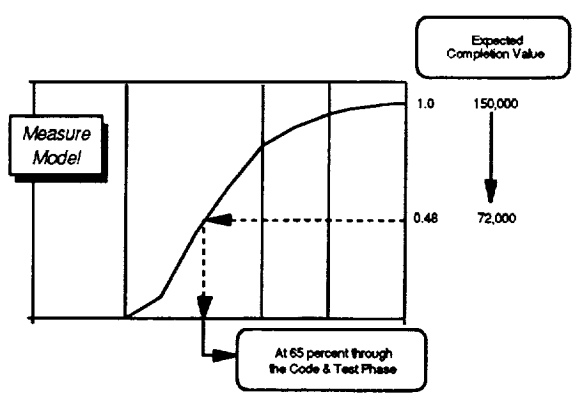
   $$\text{Fraction of Measure}_{\text{To Date}} = \text{Measure Value}_{\text{To Date}} \;/\; \text{Expected Completion Value}$$

2. Identify the phase and segment in which the fraction of measure falls by serially examining the measure model to locate the first phase k and segment j that satisfies the following

   $$\text{Fraction of Measure}_{\text{To Date}} <= \sum_{i=1}^{k-1} \text{Fraction of Measure}_{\text{In Phase}}[i, JMax(i)] + \text{Fraction of Measure}_{\text{In Phase}}[k,j]$$

3. Linearly interpolate the fraction of phase k, in segment j, that corresponds to the fractional measure value as

   $$\text{Fraction of Phase} = (F(j) - F(j-1)) * \text{Fraction of Measure}_{\text{To Date}} / (\text{Fraction of Measure}_{\text{In Phase}}[k,j] - \text{Fraction of Measure}_{\text{In Phase}}[k,j-1])$$



STEPS

1. Dividing a cumulative measure value of 72000 by the completion estimate of 150000 yields a value of 0.48.

2. The fraction of measure matching a cumulative value of 0.48 in the measure model falls within the code and test phase.

3. Linearly interpolating between the fractional measure values in the model identifies a point 65% into the code and test phase.

Figure 2-41. Converting a Measure to an Expected Phase

2.2.2.3.3 Determine Normal Measure Guidelines

Purpose

Calculates expected cumulative measure values, with upper and lower normal bounds on the values, as a function of project schedule.

Required Data

- Project start and end dates (input value)
- Expected measure value at project completion (input value)
- Schedule model
- Measure model

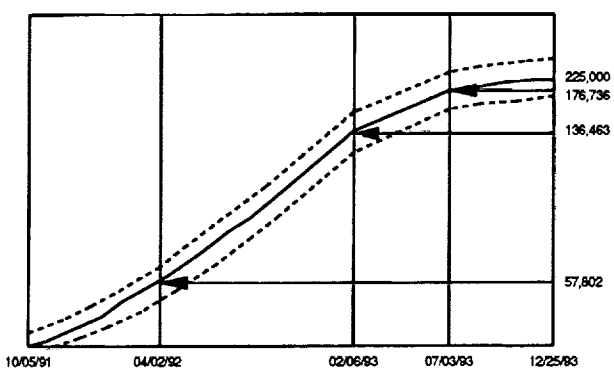
Steps

1. Use the schedule model routine *Convert Phase to Date* with the specified project start and end dates to determine the calendar dates associated with each phase and phase segment defined in the model (*Expected Calendar Date [i,j], for the ith phase and j^{th} segment*).

2. Also for each phase and phase segment, use the measure model routine *Convert Phase to Measure* with the expected completion value to determine the expected cumulative measure value for those dates from the model (*Expected Measure Value [i,j]*).

3. Compute the upper and lower normal bounds on the measure values by adding and subtracting, respectively, the normal deviation stored in the measure model from each expected measure value.

   *Normal Range [i,j] = Expected Measure Value [i,j] ± (Normal Deviation * Expected Completion Value)*



STEPS

1. For a start date and end date of 10/05/91 and 12/25/93, applying the schedule model results in intermediate phase dates of 04/02/92, 02/06/93, and 07/03/93.

2. For a completion value of 225000, applying the measure model results in intermediate values of 57802, 136463, and 176736 at these phase boundaries.

3. The normal range results from scaling the standard deviation in the measure model by 225000. Upper and lower bounds for the range are the expected measure values at each date plus or minus the scaled value.

Figure 2-42. Determining Normal Measure Guidelines

2.2.2.3.4 Generate Rate Model

Purpose

Generates a measure model, known as a rate model, that captures the typical behavior of the cumulative ratio of any two specified measures as a function of life-cycle phase.

Required Data

- Measure name for numerator (input value)
- Measure name for denominator (input value)
- Measure models (for the two specified measures)

Steps

1. For each phase and phase segment in the measure models of both the specified numerator and denominator, adjust the expected fraction of measure values to be cumulative from project start using

$$\text{Fraction of Measure}_{\text{Rate From Start}}[i,j] = \text{Fraction of Measure}_{\text{In Phase}}[i,j] + \sum_{n=1}^{i-1} \text{Fraction of Measure}_{\text{In Phase}}[n, JMax(n)]$$

   for the ith phase and j^{th} segment, where $i > 1$ and $JMax(i-1)$ is the last segment in the $(i-1)^{th}$ phase

2. Divide the fraction of measure values for the numerator by the corresponding denominator values to obtain expected rate values at each phase and segment.

3. Adjust each computed fraction of measure value to be cumulative within phase using

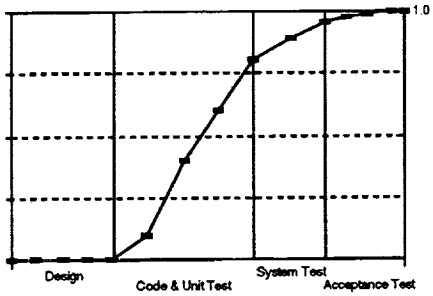
$$\text{Fraction of Measure}_{\text{Rate In Phase}}[i,j] = \text{Fraction of Measure}_{\text{Rate From Start}}[i,j] - \text{Fraction of Measure}_{\text{Rate From Start}}[i-1, JMax(i-1)]$$

4. Set the normal deviation for the rate model to the maximum absolute deviation to expect from the two individual measure models using

$$\text{Normal Deviation} = Max\left(\left| 1 - \frac{1 + \text{Normal Deviation}_{Num}}{1 - \text{Normal Deviation}_{Denom}} \right| , \left| 1 - \frac{1 - \text{Normal Deviation}_{Num}}{1 + \text{Normal Deviation}_{Denom}} \right| \right)$$

Measure Model (for Numerator)

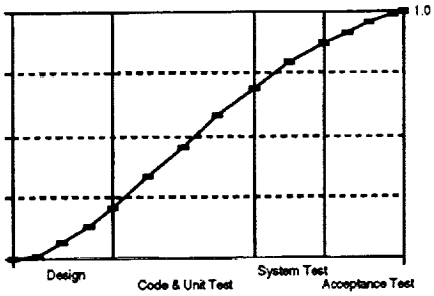
Cumulative Lines of Code Model over Life Cycle



Phase Name	Fraction of Phase	Cumulative Measure
DESGN	0.25	0.0000
	0.50	0.0000
	0.75	0.0000
	1.00	0.0000
CODET	0.25	0.0975
	0.50	0.3937
	0.75	0.6018
	1.00	0.8072
SYSTE	0.50	0.8911
	1.00	0.9564
ACCTE	0.25	0.9745
	0.50	0.9839
	0.75	0.9977
	1.00	1.0000
Normal Deviation = 0.065381		

Measure Model (for Denominator)

Cumulative Effort Model over Life Cycle

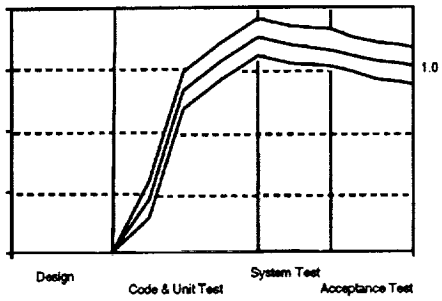


Phase Name	Fraction of Phase	Cumulative Measure
DESGN	0.25	0.0192
	0.50	0.0655
	0.75	0.1313
	1.00	0.2101
CODET	0.25	0.3360
	0.50	0.4523
	0.75	0.5809
	1.00	0.6929
SYSTE	0.50	0.7958
	1.00	0.8724
ACCTE	0.25	0.9145
	0.50	0.9519
	0.75	0.9837
	1.00	1.0000
Normal Deviation = 0.026930		

Rate Model

Phase Name	Fraction of Phase	Cumulative Rate
DESGN	0.25	0.0000
	0.50	0.0000
	0.75	0.0000
	1.00	0.0000
CODET	0.25	0.2902
	0.50	0.8703
	0.75	1.0357
	1.00	1.1648
SYSTE	0.50	1.1197
	1.00	1.0963
ACCTE	0.25	1.0655
	0.50	1.0336
	0.75	1.0142
	1.00	1.0000
Normal Deviation = 0.094866		

Cumulative Lines of Code per Effort over Life Cycle



STEPS

Generating any rate model proceeds as follows:

1. Adjust the expected fraction of measure values in both the numerator's and the denominator's measure model to be cumulative from the start of the project.

2. To compute rate model values, divide the expected value for the numerator at each model segment by its corresponding expected value for the denominator.

3. Adjust the cumulative rate model values, calculated above, to be fractional values within each phase.

4. Calculate the rate model's normal deviation based on the worst case allowed by the two individual models.

The specific example shown here illustrates generating a rate model for lines of code per hour (i.e., LOC/EFF):

The upper figure shows adjusting the measure model for LOC (i.e., the numerator of the rate) to be cumulative from the project start.

The middle figure shows adjusting the measure model for effort (i.e., the denominator of the rate) to be cumulative from the project start.

The bottom figure shows the cumulative rate model that results from dividing the LOC values by the effort values at each model segment.

Note that the rate model's normal deviation is depicted in the bottom figure as guidelines about the normal values.

Figure 2-43. Generating a Rate Model

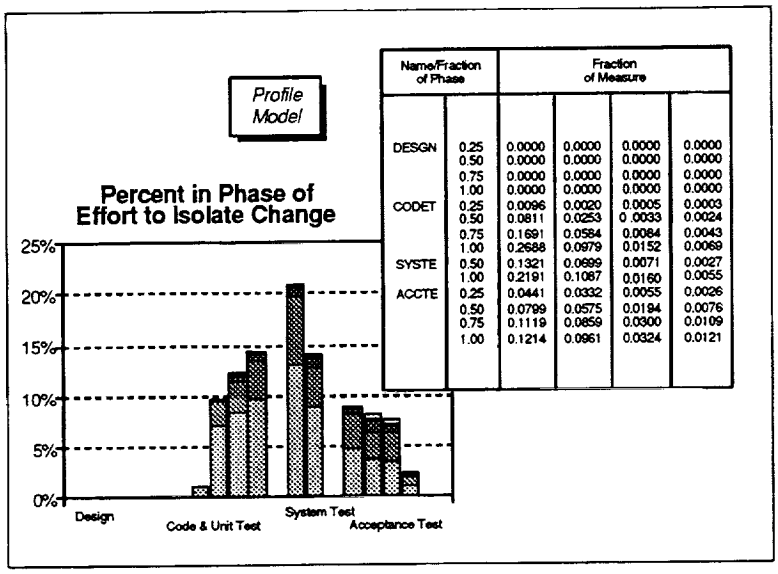
2.2.3 Profile Models

Purpose

> Describes the normal behavior over time of a software development measure using an associated profile such as effort to isolate change or effort to correct error.

Description

A profile is a breakdown of a basic measure into discrete categories that describe the behavior of the measure in greater detail. A profile model is a normalized representation of the typical behavior of a profile as a function of life-cycle phase. The SME uses four profile models to describe a given type of project. These four profile models correspond to two of the eight key measures defined for use with the SME. As with other SME models, specific points in the life cycle are identified by the combination of a phase name and an elapsed fraction of that phase between 0 and 1.0 inclusive. The value of each component expected at those points is measured from the start of the phase and is expressed as a fraction of the total component value at project completion. The sum of all components across all phases of the total fractional profile value in a phase is 1.0.



Name/Fraction of Phase		Fraction of Measure			
DESGN	0.25	0.0000	0.0000	0.0000	0.0000
	0.50	0.0000	0.0000	0.0000	0.0000
	0.75	0.0000	0.0000	0.0000	0.0000
	1.00	0.0000	0.0000	0.0000	0.0000
CODET	0.25	0.0096	0.0020	0.0005	0.0003
	0.50	0.0811	0.0253	0.0033	0.0024
	0.75	0.1691	0.0584	0.0084	0.0043
	1.00	0.2688	0.0979	0.0152	0.0069
SYSTE	0.50	0.1321	0.0699	0.0071	0.0027
	1.00	0.2191	0.1087	0.0160	0.0055
ACCTE	0.25	0.0441	0.0332	0.0055	0.0026
	0.50	0.0799	0.0575	0.0194	0.0076
	0.75	0.1119	0.0859	0.0300	0.0109
	1.00	0.1214	0.0961	0.0324	0.0121

Figure 2-44. Representative Profile Model for IBM, FORTRAN, AGSS Projects

Source

Statistical averaging of actual profile data from a set of completed development projects

Assumptions

- Profile data behavior is dependent on life-cycle phase
- At project start, all profile values are zero

Instances

One model exists for each defined profile for each project type.

Structure

Table with two fixed columns—phase name and fraction of phase—and a column for each defined component containing its fraction of measure; list of text values describing what each component represents. Each row in the table describes the fractional amount of the profile typically observed from the start of the phase through the point in the life cycle specified by the row's phase name and fraction of phase, broken down by component. As with measure models, each phase is broken into multiple intervals for a total of 14 segments with one per row.

The following sections describe a representative set of profile models, detail the steps required to create any profile model using actual data from completed projects, and present a set of general-purpose algorithms commonly used with profile models.

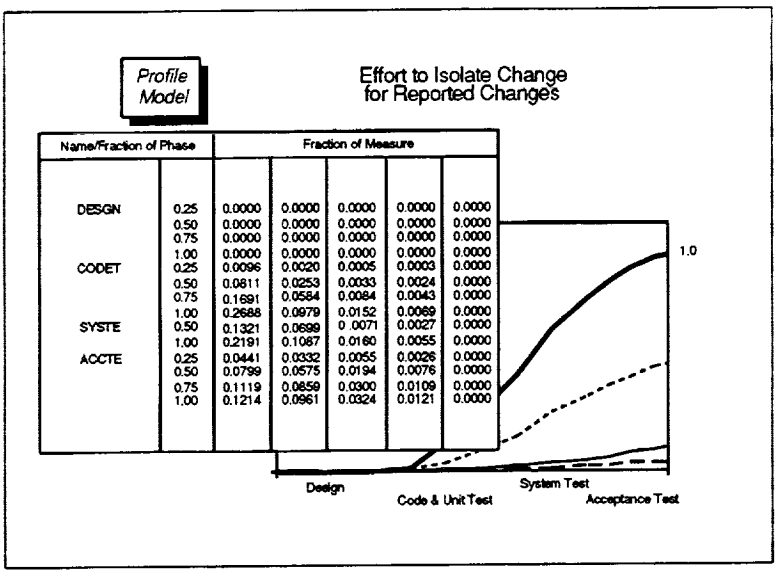
2.2.3.1 Defined Profile Models

The SME defines a set of four specific profile models for each supported project type. These models are

- Effort to Isolate Change Model
- Effort to Implement Change Model
- Effort to Isolate Error Model
- Effort to Correct Error Model

The sample profile models presented below illustrate a complete set of these models for one of the supported project types—IBM, FORTRAN, AGSS projects.

2.2.3.1.1 Effort to Isolate Change Model



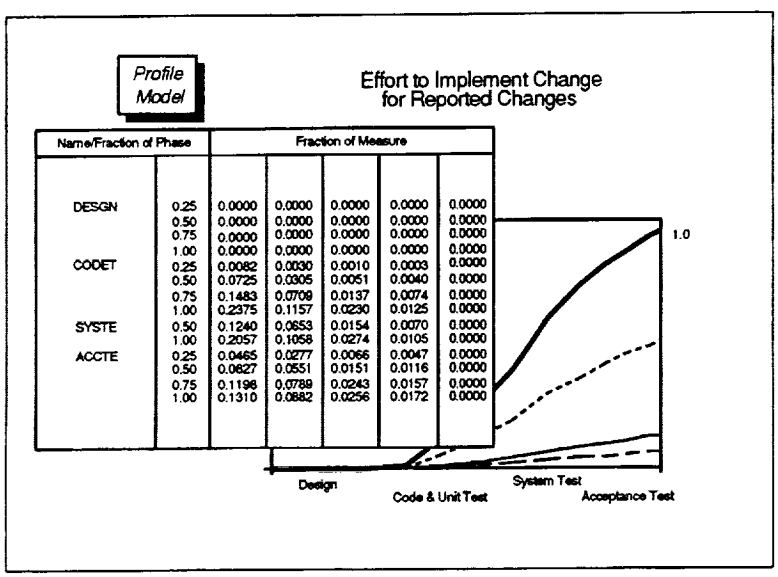
Name/Fraction of Phase		Fraction of Measure				
DESGN	0.25	0.0000	0.0000	0.0000	0.0000	0.0000
	0.50	0.0000	0.0000	0.0000	0.0000	0.0000
	0.75	0.0000	0.0000	0.0000	0.0000	0.0000
	1.00	0.0000	0.0000	0.0000	0.0000	0.0000
CODET	0.25	0.0096	0.0020	0.0005	0.0003	0.0000
	0.50	0.0811	0.0253	0.0033	0.0024	0.0000
	0.75	0.1691	0.0584	0.0084	0.0043	0.0000
	1.00	0.2688	0.0979	0.0152	0.0069	0.0000
SYSTE	0.50	0.1321	0.0699	0.0071	0.0027	0.0000
	1.00	0.2191	0.1087	0.0160	0.0055	0.0000
ACCTE	0.25	0.0441	0.0332	0.0055	0.0026	0.0000
	0.50	0.0799	0.0575	0.0194	0.0076	0.0000
	0.75	0.1119	0.0859	0.0300	0.0109	0.0000
	1.00	0.1214	0.0961	0.0324	0.0121	0.0000

Figure 2-45. Effort to Isolate Change Model for IBM, FORTRAN, AGSS Projects

An effort to isolate change model describes how effort is normally expended in isolating reported changes on a given type of project as a function of life-cycle phase. The model captures the number of reported changes to expect in five categories that are based on the effort needed to isolate the change—1 hour or less, 1 day to 1 hour, 3 days to 1 day, more than 3 days, and unknown.

Note: For any phase and fraction of phase, the sum of the fractional values across all categories equals the fractional value in the reported changes model.

2.2.3.1.2 Effort to Implement Change Model



Name/Fraction of Phase		Fraction of Measure				
DESGN	0.25	0.0000	0.0000	0.0000	0.0000	0.0000
	0.50	0.0000	0.0000	0.0000	0.0000	0.0000
	0.75	0.0000	0.0000	0.0000	0.0000	0.0000
	1.00	0.0000	0.0000	0.0000	0.0000	0.0000
CODET	0.25	0.0082	0.0030	0.0010	0.0003	0.0000
	0.50	0.0725	0.0305	0.0051	0.0040	0.0000
	0.75	0.1483	0.0709	0.0137	0.0074	0.0000
	1.00	0.2375	0.1157	0.0230	0.0125	0.0000
SYSTE	0.50	0.1240	0.0653	0.0154	0.0070	0.0000
	1.00	0.2057	0.1058	0.0274	0.0105	0.0000
ACCTE	0.25	0.0465	0.0277	0.0066	0.0047	0.0000
	0.50	0.0627	0.0551	0.0151	0.0116	0.0000
	0.75	0.1198	0.0789	0.0243	0.0157	0.0000
	1.00	0.1310	0.0882	0.0256	0.0172	0.0000

Figure 2-46. Effort to Implement Change Model for IBM, FORTRAN, AGSS Projects

An effort to implement change model describes how effort is normally expended in making reported changes on a given type of project as a function of life-cycle phase. The model captures the number of reported changes to expect in five categories that are based on the effort needed to make the change—1 hour or less, 1 day to 1 hour, 3 days to 1 day, more than 3 days, and unknown.

Note: For any phase and fraction of phase, the sum of the fractional values across all categories equals the fractional value in the reported changes model.

2.2.3.1.3 Effort to Isolate Error Model

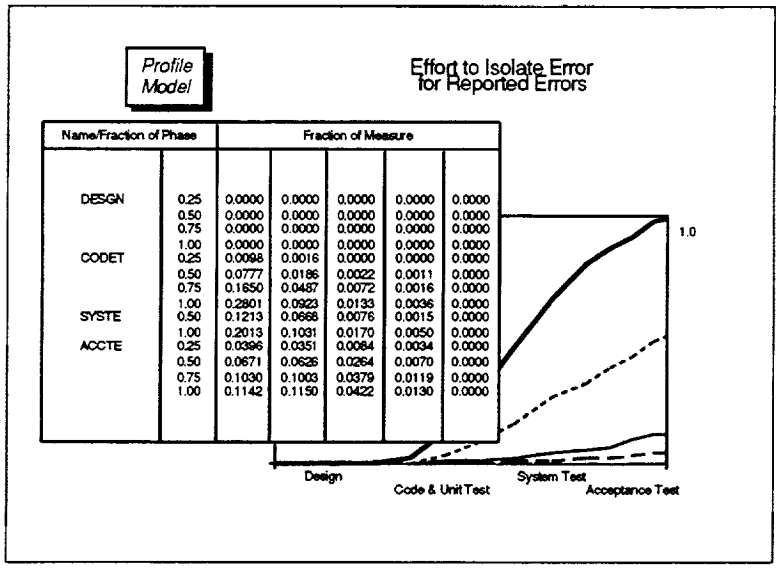


Figure 2-47. Effort to Isolate Error Model for IBM, FORTRAN, AGSS Projects

An effort to isolate error model describes how effort is normally expended in isolating reported errors on a given type of project as a function of life-cycle phase. The model captures the number of reported errors to expect in five categories that are based on the effort needed to isolate the error—1 hour or less, 1 day to 1 hour, 3 days to 1 day, more than 3 days, and unknown.

Note: For any phase and fraction of phase, the sum of the fractional values across all categories equals the fractional value in the reported errors model.

2.2.3.1.4 Effort to Correct Error Model

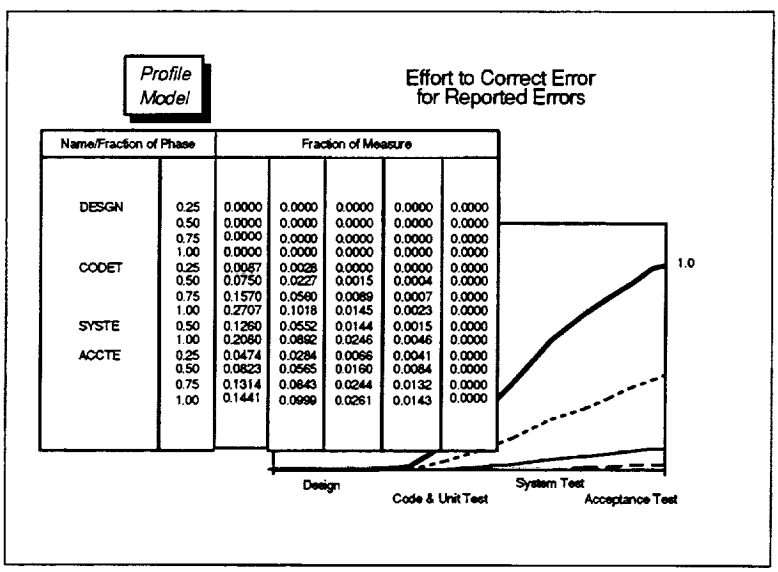


Figure 2-48. Effort to Correct Error Model for IBM, FORTRAN, AGSS Projects

An effort to correct error model describes how effort is normally expended in correcting reported errors on a given type of project as a function of life-cycle phase. The model captures the number of reported errors to expect in five categories that are based on the effort needed to fix the error—1 hour or less, 1 day to 1 hour, 3 days to 1 day, more than 3 days, and unknown.

Note: For any phase and fraction of phase, the sum of the fractional values across all categories equals the fractional value in the reported errors model.

2.2.3.2 Creating a Profile Model

The profile models used by the SME are created by normalizing and then statistically averaging actual project profile data observed on a set of one or more similar, completed development projects. The projects selected for inclusion in the set should be representative of the type of project to be captured by the model. The algorithm may be applied to any defined profile with data. By first normalizing the measurements, the creation process gives equal weight within the model to each contributing project regardless of size or duration.

Required Data

- Schedule data (for each project in the set)
- Profile data (for the profile of interest, for each project in the set)

Step 1—Normalize Each Project's Profile Data

For each project in the set, perform the following:

1. For each life-cycle phase in the schedule data, determine the actual number of weeks from the project start date through the start date of the phase ($Actual\ Weeks_{To\ Phase}[i]$) and calculate the actual number of weeks elapsed between the start and end dates of the phase ($Actual\ Weeks_{In\ Phase}[i]$).

2. For each phase segment to include in the model, calculate the actual number of weeks from project start through the segment as

   $$Week\ Number_{Segment}[i,j] = Actual\ Weeks_{To\ Phase}[i] + F(j) * Actual\ Weeks_{In\ Phase}[i]$$

   for the ith phase and j^{th} segment, where F(j) refers to the fraction of phase of the jth segment

3. For each calculated week number corresponding to the desired phase segments, normalize the actual measure value of each component for that week, measured cumulatively from project start, by the actual total measure value of all components at project completion

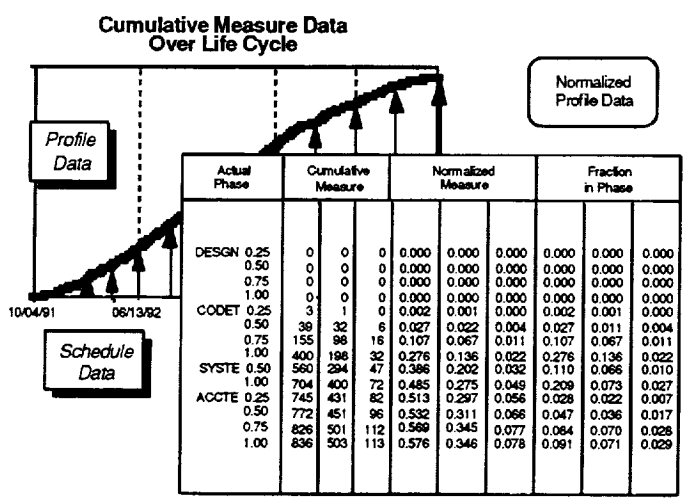
   $$Fraction\ of\ Measure_{Segment}[i,j,k] = Actual\ Measure_{For\ Week}[i,j,k] \ / \ Actual\ Measure_{Total}$$

   for the kth component (i.e., the kth profile category)

4. Adjust the computed fraction of measure values for each component to be cumulative within phase using

   $$Fraction\ of\ Measure_{In\ Phase}[i,j,k] = Fraction\ of\ Measure_{Segment}[i,j,k] - Fraction\ of\ Measure_{Segment}[i-1, JMax(i-1), k]$$

   for the ith phase, j^{th} segment, and k^{th} component,
   where $i > 1$ and JMax(i-1) is the last segment in the $(i-1)^{th}$ phase



Actual Phase		Cumulative Measure			Normalized Measure			Fraction in Phase		
DESGN	0.25	0	0	0	0.000	0.000	0.000	0.000	0.000	0.000
	0.50	0	0	0	0.000	0.000	0.000	0.000	0.000	0.000
	0.75	0	0	0	0.000	0.000	0.000	0.000	0.000	0.000
	1.00	0	0	0	0.000	0.000	0.000	0.000	0.000	0.000
CODET	0.25	3	1	0	0.002	0.001	0.000	0.002	0.001	0.000
	0.50	39	32	6	0.027	0.022	0.004	0.027	0.011	0.004
	0.75	155	98	16	0.107	0.067	0.011	0.107	0.067	0.011
	1.00	400	198	32	0.276	0.136	0.022	0.276	0.136	0.022
SYSTE	0.50	560	294	47	0.386	0.202	0.032	0.110	0.066	0.010
	1.00	704	400	72	0.485	0.275	0.049	0.209	0.073	0.027
ACCTE	0.25	745	431	82	0.513	0.297	0.056	0.028	0.022	0.007
	0.50	772	451	96	0.532	0.311	0.066	0.047	0.036	0.017
	0.75	826	501	112	0.569	0.345	0.077	0.084	0.070	0.028
	1.00	836	503	113	0.576	0.346	0.078	0.091	0.071	0.029

STEPS

1. The number of weeks in the DESGN, CODET, SYSTE, and ACCTE phases are 30, 40, 21, and 25, respectively.

2. The phases are broken down into 4, 4, 2, and 4 segments, respectively.

3. The cumulative total for each component at each segment is divided by 1452, the cumulative total of the sums of the components.

4. Each segment's value is converted to a value that is cumulative within phase.

Figure 2-49. Normalizing a Project's Profile Data

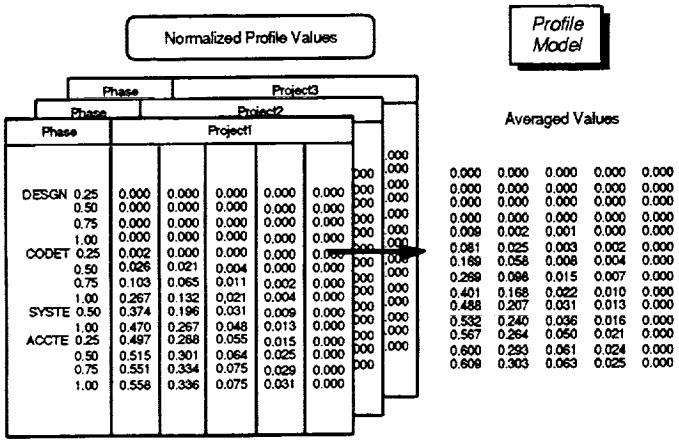
Step 2—Average the Normalized Profile Data

Using the intermediate results from the first step, calculate the normal values to be stored in the profile model as follows:

1. For each life-cycle phase and segment, average the normalized values calculated for the fraction of measure of each component within phase as observed by the selected projects using

$$\text{Normal Fraction of Measure}_{\text{In Phase}}[i,j,k] = \left(\sum_{p=1}^{N} \text{Fraction of Measure}_{\text{In Phase}}[i,j,k,p] \right) / N$$

for the ith phase, j^{th} segment, and k^{th} component, where p refers to projects 1 through N



STEPS

1. The normalized values for all components at each segment for the three projects are averaged.

2. The resultant averaged values may subsequently be stored as a profile model.

Figure 2-50. Averaging Normalized Profile Data

2.2.3.3 General-Purpose Use of Profile Models

The SME incorporates a set of general-purpose services commonly used with profile models. The services are referenced freely by SME functions to provide needed services associated with profile models. These routines include

- *Convert Phase to Profile Measure*

The following section discusses this routine and details the algorithms behind the service it provides.

2.2.3.3.1 Convert Phase to Profile Measure

Purpose

Calculates the cumulative profile vector that can normally be expected at a given point in the life cycle specified by a phase name and elapsed fraction of phase.

Required Data

- Phase name and elapsed fraction of phase (input value)
- Expected measure value at project completion (input value)
- Profile model

Steps

1. Referencing the profile model, linearly interpolate the cumulative fraction of each component's value normally expected within the specified phase as

$$\text{Fraction of Measure}_{\text{In Phase}}[k,l] = \text{Fraction of Measure}_{\text{In Phase}}[k,j\text{-}1,l] + (\text{Fraction of Measure}_{\text{In Phase}}[k,j,l] - \text{Fraction of Measure}_{\text{In Phase}}[k,j\text{-}1,l]) * (F - F(j\text{-}1)) / (F(j) - F(j\text{-}1))$$

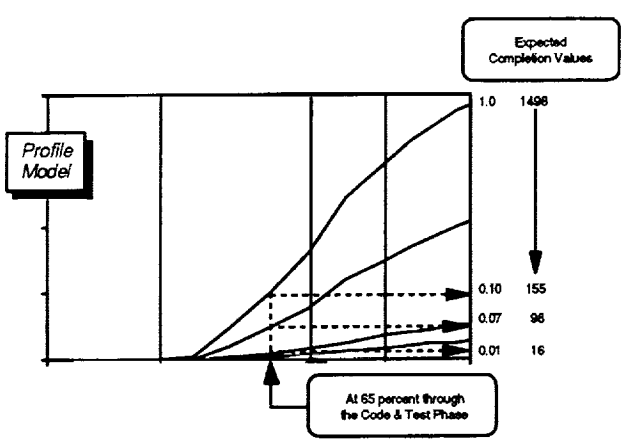
for the kth phase, the lth component, and an elapsed fraction of phase, F, where $F(j\text{-}1) < F <= F(j)$

2. Also, from the model, calculate the cumulative fraction of each component's value normally expected in any earlier phases occurring before the specified phase as

$$\text{Fraction of Measure}_{\text{Before Phase}}[k,l] = \sum_{i=1}^{k-1} \text{Fraction of Measure}_{\text{In Phase}}[i,JMax(i),l]$$

3. Obtain the expected profile vector by scaling the sum of these two vectors of computed values by the specified total expected measure value at project completion

$$\text{Expected Component Value}[l] = \text{Expected Completion Value} * (\text{Fraction of Measure}_{\text{In Phase}}[k,l] + \text{Fraction of Measure}_{\text{Before Phase}}[k,l])$$



STEPS

1. Using the profile model, at 65% through CODET the components will normally attain cumulative measure values of 1%, 7%, and 10% of the expected total value at project completion.

2. Given an expected project completion value of 1498 for the measure, the normal profile values to expect at this point in the schedule are 16, 98, and 155.

Figure 2-51. Converting a Phase to a Profile Measure

2.2.4 Estimate Set Models

Purpose

Describes the relationships that exist between the completion values of measures.

Description

An estimate set model is a normalized representation of the measure values to expect at project completion. The model implicitly captures the set of linear relationships that exist between estimated completion values for each pair of measures. The completion values in the model are normalized to 1000 lines of code, with one value for each measure defined in the measure list. The order of the measures in the model denotes the default hierarchy used by the SME in choosing a measure whose estimated completion value will be used as a scaling factor to generate the set of normal completion values.

Estimate Set Model

Measure Code	Completion Value
LOC	1000.000
MOD	5.251
EFF	255.298
CPU	0.832
MCH	17.624
RCH	8.501
RER	4.376
RUN	304.778

Figure 2-52. Estimate Set Model for IBM, FORTRAN, AGSS Projects

Source

Statistical averaging of actual measure completion values from a set of completed development projects

Assumptions

- Over the domain of the model, linear expressions are sufficient to capture the relationships between completion values
- A one-to-one mapping exists between the entries in the estimate set model and the measures defined in the measure list
- A measure model exists for each entry in the estimate set model

Instances

One model exists for each project type.

Structure

Table with two columns—measure code and completion value. Each row in the table supplies the estimated completion value per 1000 lines of code for the named measure.

The following sections detail the steps required to create estimate set models using actual data from completed projects and present a set of general-purpose algorithms commonly used with estimate set models.

2.2.4.1 Creating an Estimate Set Model

The estimate set models used by the SME are created by normalizing and then statistically averaging actual measure completion values observed on a set of one or more similar, completed development projects. The projects selected for inclusion in the set should be representative of the type of project to be captured by the model and should have measure data for each defined measure. By first normalizing the completion values, the two-step creation process gives equal weight within the model to each contributing project regardless of size or duration.

Required Data

- Measure data (for each project in the set, for each measure)

Step 1—Normalize Each Project's Completion Values

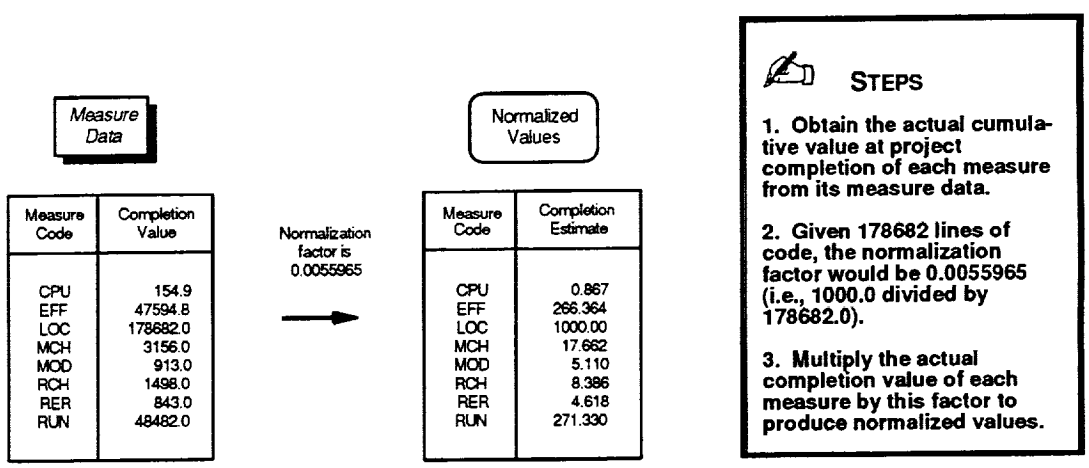
For each project in the set, perform the following:

1. For each defined measure, obtain the actual cumulative measure value at project completion from the measure data (*Actual Completion Value [i]*).
2. Calculate the normalization factor based on the actual completion value for lines of code as

   Normalization Factor = 1000.0 / Actual Completion Value$_{LOC}$

3. Normalize each measure's actual completion value using the computed factor

   *Normalized Completion Value [i] = Actual Completion Value [i] * Normalization Factor*



Measure Code	Completion Value
CPU	154.9
EFF	47594.8
LOC	178682.0
MCH	3156.0
MOD	913.0
RCH	1498.0
RER	843.0
RUN	48482.0

Measure Code	Completion Estimate
CPU	0.867
EFF	266.364
LOC	1000.00
MCH	17.662
MOD	5.110
RCH	8.386
RER	4.618
RUN	271.330

Figure 2-53. Normalizing a Project's Completion Values

Step 2—Average the Normalized Project Completion Values

Using the intermediate results from the first step, calculate the normal completion values to be stored in the estimate set model as follows:

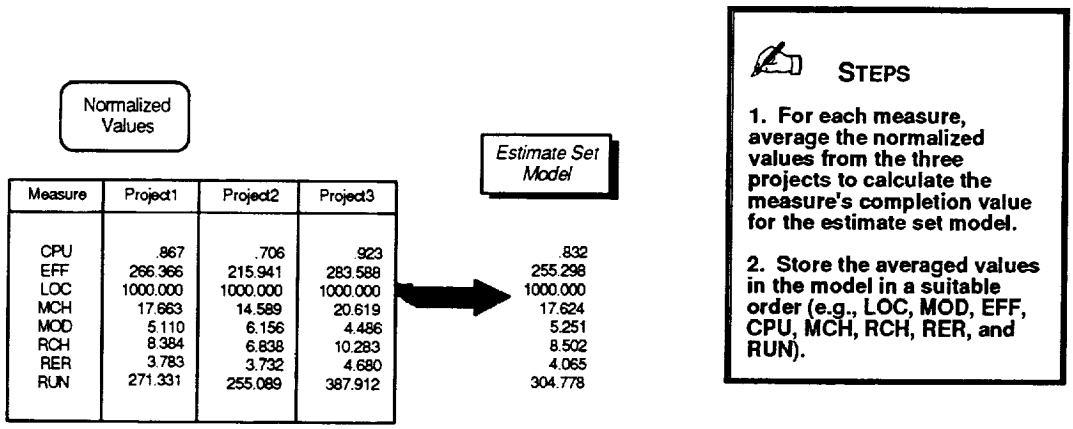
1. For each defined measure, average the normalized measure completion values for the selected projects using

$$\text{Normal Completion Value } [i] = \left(\sum_{j=1}^{N} \text{Normalized Completion Value } [i,j] \right) / N$$

   for the ith measure, where j refers to projects 1 through N

2. Store the normal completion values in the model in order of the measure's decreasing importance in determining the magnitude of a project.

   Note: By convention, the order used by the SME is lines of code (LOC), module count (MOD), total staff hours (EFF), computer hours (CPU), modules changed (MCH), reported changes (RCH), reported errors (RER), and computer runs (RUN).



Measure	Project1	Project2	Project3	Estimate Set Model
CPU	.867	.706	.923	.832
EFF	266.366	215.941	283.588	255.298
LOC	1000.000	1000.000	1000.000	1000.000
MCH	17.663	14.589	20.619	17.624
MOD	5.110	6.156	4.486	5.251
RCH	8.384	6.838	10.283	8.502
RER	3.783	3.732	4.680	4.065
RUN	271.331	255.089	387.912	304.778

Figure 2-54. Averaging Normalized Completion Values

2.2.4.2 General-Purpose Use of Estimate Set Models

The SME incorporates a set of general-purpose services commonly used with estimate set models. The services are referenced in various high-level SME functions to provide needed functions associated with estimate set models. These services include

- *Get Ratio of Estimates*
- *Determine Normal Estimate Set*
- *Get Project Magnitude*

The following sections discuss each of these services and detail the algorithms behind the actions they perform.

2.2.4.2.1 Get Ratio of Estimates

Purpose

Obtains the ratio of estimated completion values normally expected for any two specified measures.

Required Data

- Measure name for numerator (input value)
- Measure name for denominator (input value)
- Estimate set model

Steps

1. Obtain the normal completion values of the two specified measures from the estimate set model.
2. Divide the completion value of the measure for the numerator by the completion value of the measure for the denominator to obtain the normal ratio of estimated completion values.

$$\text{Ratio of Estimates}_{\text{At Completion}} = \text{Completion Value}_{\text{Num}} \ / \ \text{Completion Value}_{\text{Denom}}$$

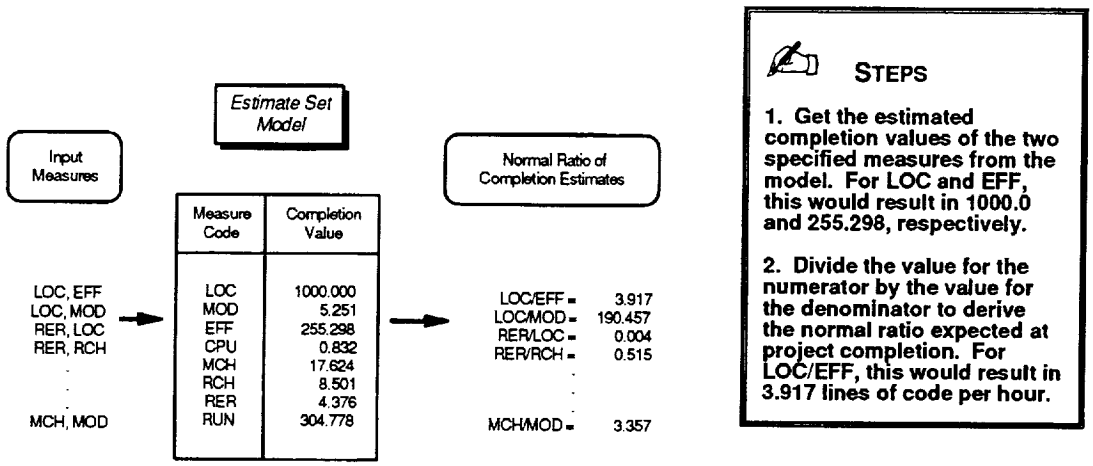


Figure 2-55. Obtaining the Ratio of Completion Estimates

2.2.4.2.2 Determine Normal Estimate Set

Purpose

Produce a full set of normal completion estimates for all measures given the expected completion value for any one measure.

Required Data

- Measure name (input value)
- Expected completion value for the measure (input value)
- Estimate set model

Steps

1. Locate the specified measure in the estimate set model and obtain the normal completion value for the measure (*Normal Completion Value*$_{Measure}$).
2. Calculate a scaling factor for the model based on the ratio of the input expected completion value for the measure to the model's normal completion value as

   Scale Factor = Expected Completion Value$_{Measure}$ / *Normal Completion Value*$_{Measure}$

3. Multiply the completion values found in the estimate set model for each measure by the calculated scaling factor to produce a set of completion estimates using

   *Completion Estimate [i] = Normal Completion Value [i] * Scale Factor*

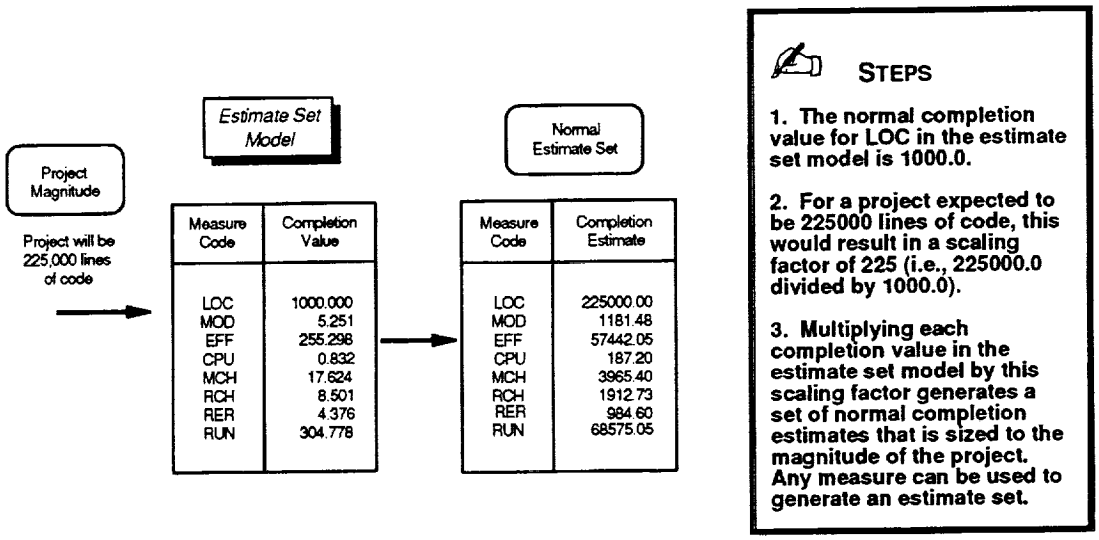


Figure 2-56. Determining a Normal Estimate Set

2.2.4.2.3 Get Project Magnitude

Purpose

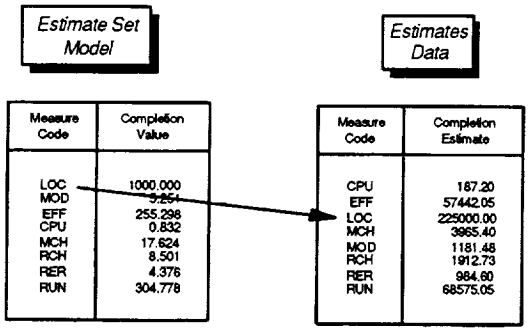
Obtains the measure and estimated completion value for the measure that is most indicative of the project's magnitude.

Required Data

- Estimate data
- Estimate set model

Steps

1. Locate the first measure in the estimate set model for which there exists a non-zero value in the project's estimate data ($PlannedValue_{Completion}$).
2. Identify the measure and return the planned completion value stored in the estimate data for the measure.



STEPS

1. LOC is the first measure in the estimate set model for which there exists a non-zero value in the estimates data, with a value of 225000.

2. This would indicate a project whose magnitude is estimated at 225000 lines of code.

3. If the estimates data contained zero values for both LOC and MOD, the algorithm would show a project whose magnitude is estimated at 57442 staff hours.

Figure 2-57. Obtaining a Project's Magnitude

2.2.5 Attribute Definitions

Purpose

Describes the set of overall project quality attributes, such as correctability and maintainability, used by the SME.

Description

The attribute definitions list is a set of associated tables that (1) identifies fundamental project quality attributes used by the SME and (2) specifies how relative ratings for those attributes are calculated. The list decomposes each attribute into one or more weighted factors and further defines each weighted factor as a function. Each function is a mathematical expression consisting of arithmetic operators, numerical constants, and variable references to specific measure or profile values. This hierarchy, in essence, captures the algorithm used to evaluate measurement data to calculate a relative rating for key project quality attributes. The SME implementation currently defines two attributes—correctability and maintainability.

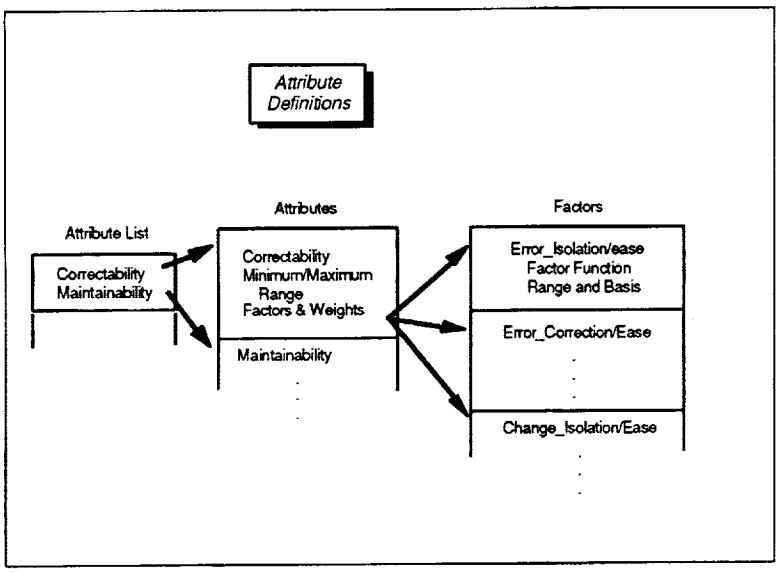


Figure 2-58. Attribute Definitions for the SME

Source

Defined as part of the SME implementation

Assumptions

- Objective measurements taken during the software development effort can be used as early indicators of project and product quality.
- The defined attribute ratings are relative to a normal project of the same project type (and are not absolute values).

Instances

The SME references one attribute definitions list.

Structure

Three tables consisting of an attribute list, a set of attributes, and a set of factors. The attribute list is a table with one column—attribute name. The names appear in alphabetical order with one defined attribute name per row. The set of attributes are described by a second table of attribute records with each record containing information on one attribute—the attribute's name, the minimum and maximum rating values, the number of underlying factors, and the name and weighting of each factor. The set of factors are

described by a third table of factor records with each record containing information on one factor—the factor's name, the maximum range of values to consider (as a percentage of the normal expected value), the function used to evaluate the factor, and the measures which must be available to evaluate the factor.

2.2.5.1 Defined Attributes

The SME defines two basic overall project quality attributes. These attributes are rated on a relative scale from -10 to +10, with 0 considered normal. Negative and positive ratings are considered below normal and above normal, respectively. The attributes are

- Correctability
- Maintainability

The following sections describe the two attributes and present a set of general-purpose algorithms commonly used with attribute definitions.

2.2.5.1.1 Correctability

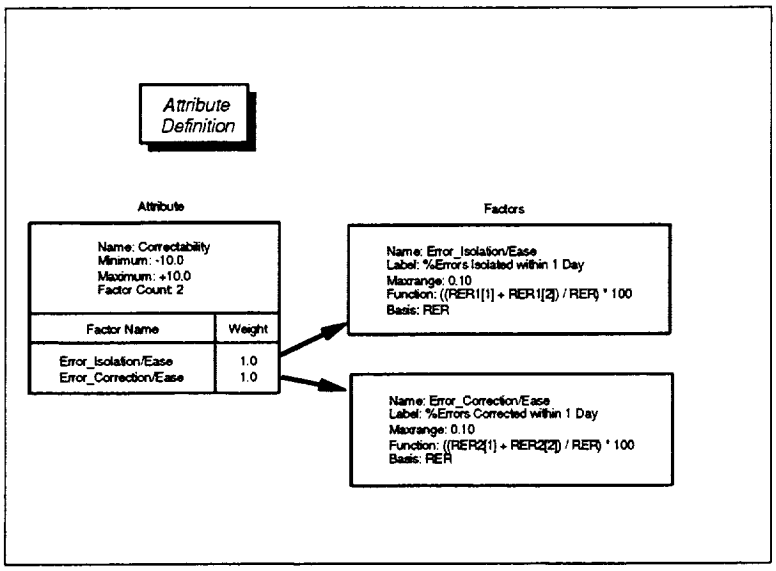


Figure 2-59. Attribute Defining Correctability

The SME rates correctability on the basis of two associated factors—the ease of isolating errors and the ease of correcting errors. Both factors rely on profile data collected on reported errors. The ease of isolating errors is calculated as the percentage of all reported errors that were isolated within 1 day. The ease of correcting errors is calculated as the percentage of all reported errors that were corrected within 1 day. After scaling, the resultant factor values are averaged to produce a relative rating on a scale of -10 to +10 for the attribute.

2.2.5.1.2 Maintainability

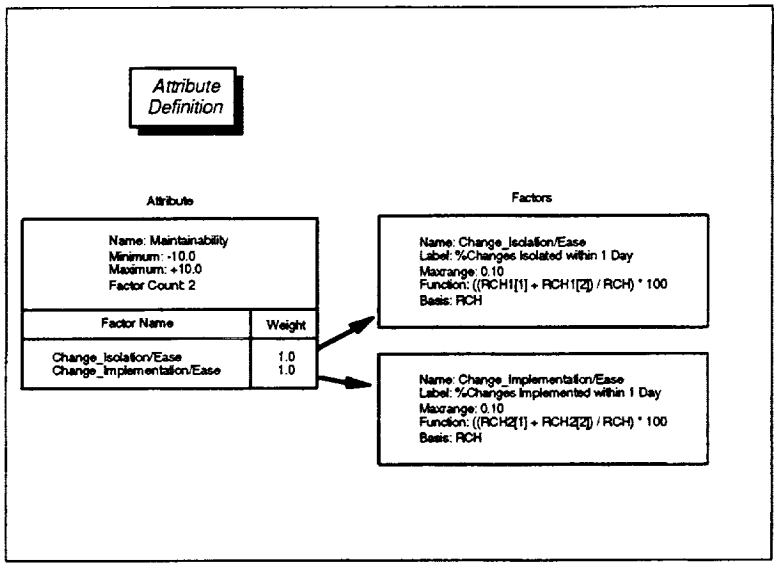


Figure 2-60. Attribute Defining Maintainability

The SME rates maintainability on the basis of two associated factors—the ease of isolating changes and the ease of implementing changes. Both factors rely on profile data collected on reported changes. The ease of isolating changes is calculated as the percentage of all reported changes that were isolated within 1 day. The ease of implementing changes is calculated as the percentage of all reported changes that were implemented within 1 day. After scaling, the resultant factor values are averaged to produce a relative rating on a scale of -10 to +10 for the attribute.

2.2.5.2 General-Purpose Uses of Attribute Definitions

The SME incorporates a set of general-purpose services commonly used with attribute definitions. The services are referenced in high-level SME functions to provide needed services associated with attribute definitions. These services include

- *Evaluate Actual Factor Value*
- *Evaluate Expected Factor Values*
- *Assess Attribute*

The following sections discuss each of these services and detail the algorithms behind the actions they perform.

2.2.5.2.1 Evaluate Actual Factor Value

Purpose

Calculates a factor's actual value as of a given date using actual project data values to evaluate the function defined for that factor.

Required Data

- Factor (input value)
- Calendar date (input value)
- Measure data (for any referenced measures)
- Profile data (for any referenced profiles)

Steps

1. For the expression in the factor's function, locate all references to measure values. Obtain the actual data value on the input calendar date from the measure data of any referenced measure.
2. For the expression in the factor's function, locate all references to profile values. Obtain the actual data value on the input calendar date from the profile data of any referenced profile.
3. Evaluate the expression in the factor's function using the actual project data values obtained (*Actual Factor Value*).

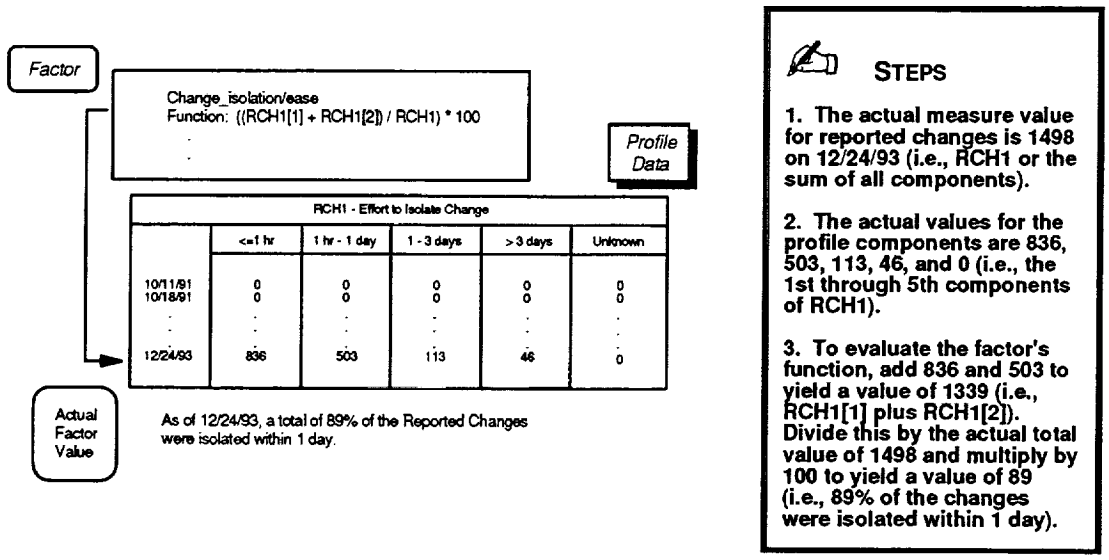


Figure 2-61. Evaluating a Factor Using Actual Data Values

2.2.5.2.2 Evaluate Expected Factor Values

Purpose

Calculates a factor's expected values as of a given date using normal model values to evaluate the function defined for that factor. The factor's expected values consist of three values that represent the normal, best, and worst cases expected for the factor.

Required Data

- Factor (input value)
- Calendar date (input value)
- Schedule data
- Schedule model (in *Convert Date to Phase*)
- Estimate data
- Estimate set model (in *Determine Normal Estimate Set*)
- Measure model (for referenced measures) (in *Convert Phase to Measure*)
- Profile model (for referenced profiles) (in *Convert Phase to Profile*)

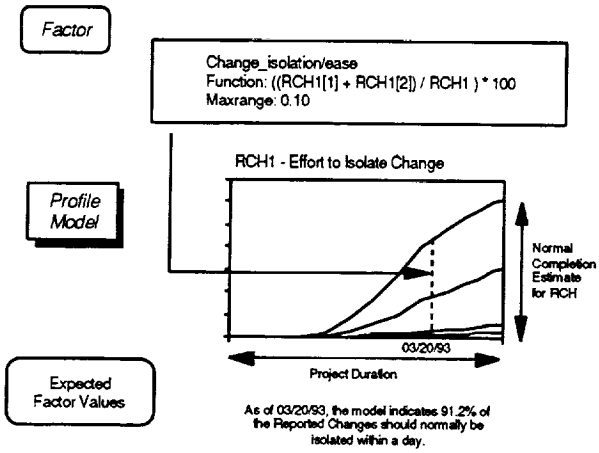
Steps

1. Use *Get Project Dates* to obtain the planned project start and end dates from the current schedule data.
2. On the basis of the project start and end dates, use *Convert Date to Phase* to translate the input calendar date to the phase and elapsed fraction of phase that normally should be reached on that date.
3. Use *Get Project Magnitude* on the current estimate data to obtain the measure and estimated completion value for the measure that is most indicative of the project's magnitude.
4. On the basis of that magnitude, use *Determine Normal Estimate Set* to create a normal set of estimates for the project.
5. For the expression in the factor's function, locate all references to measure values. For any referenced value, use *Convert Phase to Measure* to obtain the expected measure value at the desired phase and fraction of phase, given the normal completion value of the measure, from the measure model.
6. For the expression in the factor's function, locate all references to profile values. For any referenced value, use *Convert Phase to Profile* to obtain the expected profile value at the desired phase and fraction of phase, given the normal completion value of the profile's measure, from the profile model.
7. Evaluate the expression in the factor's function using the obtained model values ($Expected\ Factor\ Value_{Normal}$).

8. Use the maximum range value for the factor to compute the best and worst case expected values as

$$\text{Expected Factor Value}_{Best} = \text{Expected Factor Value}_{Normal} * (1.0 + \text{Factor Maxrange})$$

$$\text{Expected Factor Value}_{Worst} = \text{Expected Factor Value}_{Normal} * (1.0 - \text{Factor Maxrange})$$



STEPS

1. A calendar date of 3/20/93 represents 75% through the system test phase.

2. The project's magnitude is next determined to derive a normal completion estimate for reported changes of 1418.

3. Given 1418 total changes, the profile model shows that the component values at 75% of system test should be 624, 213, 53, 28, and 0.

4. When used to evaluate the factor's function, the model values yield a value of 91.2. For a maximum range value of 0.10, best and worst case expected values are 100.3 and 82.1, respectively.

Figure 2-62. Evaluating a Factor Using Normal Model Values

2.2.5.2.3 Assess Attribute

Purpose

Calculates a relative rating as of a given date for a specified project quality attribute.

Required Data

- Attribute (input value)
- Calendar date (input value)
- Factors (associated with specified attribute)

Steps

1. For each factor associated with the specified attribute, use *Evaluate Actual Factor Value*, discussed earlier, to calculate the factor's actual value as of the input calendar date (*Actual Factor Value [i], for the ith factor*).

2. For each factor associated with the specified attribute, use *Evaluate Expected Factor Values*, discussed earlier, to calculate the factor's expected values as of the input calendar date ($Expected\ Factor\ Value_{Normal}[i]$, $Expected\ Factor\ Value_{Best}[i]$, $Expected\ Factor\ Value_{Worst}[i]$).

3. Calculate the normal value for the attribute's relative rating as the average of the minimum and maximum rating values defined in the attribute with

   $$Normal\ Attribute\ Rating = (Minimum\ Rating + Maximum\ Rating) / 2$$

4. Calculate a scaling value for the attribute's relative rating as the difference between the defined maximum rating and the computed normal rating with

   $$Rating\ Scale = Maximum\ Rating - Normal\ Attribute\ Rating$$

5. For each factor, calculate the corresponding range of the expected values obtained from evaluating the factor with

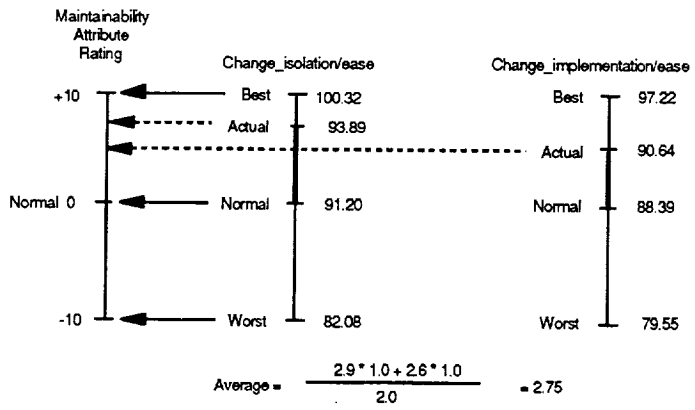
   $$Factor\ Range\ [i] = (Expected\ Factor\ Value_{Best}[i] - Expected\ Factor\ Value_{Worst}[i]) / 2$$

6. For each factor, scale the factor's actual value to match the range of values used in rating the attribute with

   $$Factor\ Rating\ [i] = Normal\ Attribute\ Rating + (Rating\ Scale / Factor\ Range\ [i]) * (Actual\ Factor\ Value\ [i] - Expected\ Factor\ Value_{Normal}[i])$$

7. Set the attribute's rating to the weighted average of the scaled factor ratings computed for each factor, *1* through *K*, using

   $$Attribute\ Rating = \left(\sum_{i=1}^{K} Factor\ Weight\ [i] * Factor\ Rating\ [i] \right) / \sum_{i=1}^{K} Factor\ Weight\ [i]$$



STEPS

1. Each factor's actual and expected values are evaluated.
2. The attribute's normal value is calculated.
3. The difference between the defined maximum rating and the computed normal rating is calculated as a scaling value.
4. The range of expected values for each factor is calculated, and the actual values are scaled.
5. The attribute's rating is set to the weighted average of the scaled factor ratings.

Figure 2-63. Assessing a Project Attribute

2.3 MANAGEMENT RULES

The SME relies on experienced software development managers in the SEL environment for the expert knowledge needed to analyze and interpret the observed behavior of projects. Capturing and applying this knowledge using expert systems techniques has been investigated by the SEL and proven feasible in this domain (References 4 and 5). Over the years, a variety of management rules and heuristics that are useful in the local environment have been collected and published in numerous SEL reports. A representative selection of these management rules may be found in *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules* (Reference 2) and in *Manager's Handbook for Software Development* (Reference 6).

Conceptually, interviewing successful software development managers to learn how they interpret certain conditions observed on a project captures reusable knowledge about evaluating a project's strengths and weaknesses. Their interpretations can then be combined or recast into specific management rules that describe the possible explanations for certain conditions. For example, one simple rule could express several possible reasons for an observed deviation in reported errors as "*If the number of reported errors is below normal, then either (1) the development team is experienced, (2) the system testing is inadequate, or (3) the problem is easier than expected.*" More complex networks or sets of these rules can be created to examine a wide range of data and provide more depth from which to draw conclusions.

The SME currently incorporates two independent approaches to capturing management rules and providing expert assistance to software development managers—a knowledge base and a rule base. The knowledge base focuses on explaining observed deviations from normal values in fundamental software development measures; the rule base concentrates on providing interpretations of the project's general status based on conditionally evaluating a series of rules.

Table 2-4 summarizes the major components referenced by the SME as management rules and identifies each component's purpose.

Table 2-4. SME Management Rules Components

COMPONENT	PURPOSE
Knowledge Base	Captures management experience that relies on objective measurements and subjective data to explain deviations in measures from normal values
Rule Base	Captures management experience that relies on a series of rules which use the observed ratios between key pairs of measures to assess the project's current status

The following sections provide additional detailed information on each of these components.

2.3.1 Knowledge Base

Purpose

Describes a collection of captured management experience that uses objective measurements and subjective data to explain deviations in measures from normal values.

Description

The knowledge base is a set of associated tables that (1) identifies possible reasons for observed deviations in a project's measures from what is considered normal and (2) specifies how to assess the probable validity and relative merit of those reasons. The list of reasons in the knowledge base are organized to associate the deviation of a measure with that deviation's possible causes. Each reason in the list is identified by an encoded reason, consisting of a causal rating and a factor name, that maps to an entry in a list of explanations used for display purposes. In assessing the reason's validity, the named factor is evaluated to produce a rating that can be compared to the causal rating. If the ratings match, the reason is a likely cause of the deviation. Each underlying factor is defined as being either objective, subjective, or dependent. Objective factors are evaluated using actual measure data, while subjective factors rely on subjective data from the manager. Dependent factors represent a weighted combination of ratings from a network of two or more factors. The SME knowledge base currently contains the reasoning needed to assess deviations in four defined measures: CPU hours, staff hours, lines of code, and reported errors.

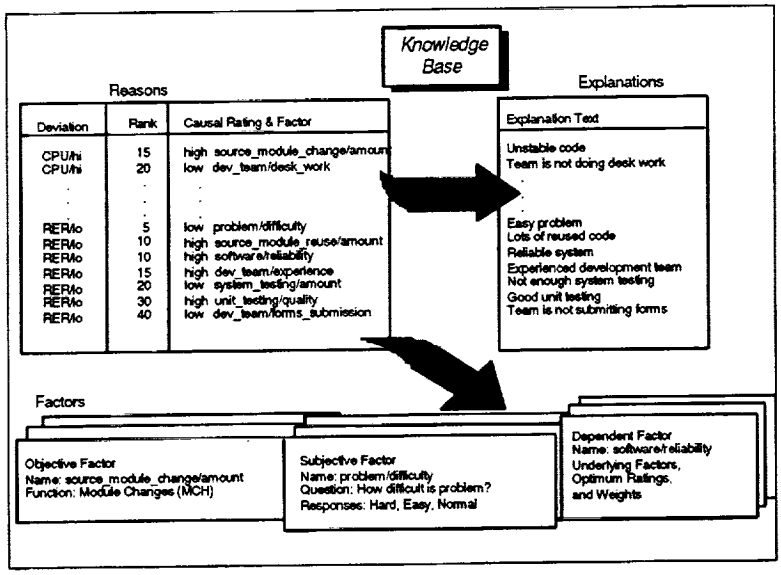


Figure 2-64. Knowledge Base for the SME

Source

Defined as part of the SME based on past experience

Assumptions

- The manager's estimated completion values accurately reflect the project's magnitude and can serve as a basis for determining what is considered normal
- The subjective data provided by the manager is rated consistently across projects

Instances

The SME has one knowledge base.

Structure

Three tables consisting of a reason list, an explanation list, and a set of factors. The reason list is a table with three columns—a deviation in a measure, the weight used to rank the

reason, and the possible reason for the deviation encoded as a causal rating and a factor name. The explanation list is a table with two columns—the encoded possible reason and the explanatory text for that reason. The set of factors are described by a third table of factor records with each record containing information on one factor. The record structure varies by the type of factor. Objective factors contain the factor name and the function used to evaluate the factor (i.e., a mathematical expression referencing specific measure values). Subjective factors contain the factor name, the question used to solicit the subjective information, and a list of acceptable responses to that question. Dependent factors contain the factor name and a list of underlying factors identified by name, weight, and optimum rating. The underlying factors referenced in a dependent factor may be objective, subjective, or dependent.

The following sections describe the specific reasoning captured in the knowledge base for assessing deviations in four specific measures and present a set of general-purpose algorithms commonly used with the knowledge base.

2.3.1.1 Captured Knowledge

The SME captures reasoning in the knowledge base for assessing deviations in four defined measures which may be either above normal (high) or below normal (low). This reasoning, discussed in detail below, addresses

- Higher than Normal CPU Hours
- Lower than Normal CPU Hours
- Higher than Normal Total Staff Hours
- Lower than Normal Total Staff Hours
- Higher than Normal Lines of Code
- Lower than Normal Lines of Code
- Higher than Normal Reported Errors
- Lower than Normal Reported Errors

2.3.1.1.1 Higher than Normal CPU Hours

The SME considers five possible reasons that could cause the number of CPU hours recorded for a project to be above normal. These reasons, in order of decreasing potential likelihood, are (1) team made up of terminal jockeys, (2) too much system testing, (3) unreliable system, (4) team is not doing desk work, and (5) unstable code. Assessing the validity of these reasons in explaining the deviation relies on evaluating the objective, subjective, and dependent factors shown below.

Possible Reasons and Explanations

Rank	Causal Rating	Factor Name	Explanation
40	High	dev_team/terminal_jockeys	Team made up of terminal jockeys
30	High	system_testing/amount	Too much system testing
25	Low	software/reliability	Unreliable system
20	Low	dev_team/desk_work	Team is not doing desk work
15	High	source_module_change/amount	Unstable code

Objective Factors

Factor Name	Function
source_code_changes/rate	RCH/LOC (Reported Changes per LOC)
source_module_change/amount	MCH (Module Changes)

Subjective Factors

Factor Name	Question	Responses (High,Low,Normal)
CM_plan/use	Is this project using/following its CM plan?	(Yes,No,N/A)
code_reading/amount	How much code reading is being done on this project?	(Lots,Minimal,Normal)
code_reading/quality	What quality rating would you assign to this project's code reading?	(High,Low,Normal)
design/stability	What level of stability would you assign to this project's design?	(High,Low,Normal)
design/quality	What quality rating would you assign to this project's design?	(High,Low,Normal)
dev_team/desk_work	Is the team completing required desk work before getting on the computer?	(Yes,No,N/A)
dev_team/terminal_jockeys	Is the team spending too much time on the computer?	(Yes,No,N/A)
librarian/use	Is this project using a librarian?	(Yes,No,N/A)
specs/stability	How would you rate the stability of the specifications for this project?	(High,Low,Normal)
system_testing/amount	How would you rate the amount of system testing being done?	(High,Low,Normal)
unit_testing/amount	How would you rate the amount of unit testing being done on this project?	(High,Low,Normal)
unit_testing/quality	What quality rating would you assign to unit testing on this project?	(High,Low,Normal)

Dependent Factors

Factor Name	Underlying Factors	Optimum Rating	Weight
CM/quality			
	CM_plan/use	High	1.0
	librarian/use	High	1.0
software/reliability			
	CM/quality	High	1.0
	code_reading/amount	High	1.0
	code_reading/quality	High	1.0
	design/stability	High	1.0
	design/quality	High	1.0
	source_code_changes/rate	Low	1.0
	specs/stability	High	1.0
	unit_testing/amount	High	1.0
	unit_testing/quality	High	1.0

Figure 2-65. Reasoning for Higher than Normal CPU Hours

2.3.1.1.2 Lower than Normal CPU Hours

The SME considers seven possible reasons that could cause the number of CPU hours recorded for a project to be below normal. These reasons, in order of decreasing potential likelihood, are (1) computer not available, (2) not enough system testing, (3) experienced development team, (4) good planning, (5) good configuration management, (6) good quality assurance, and (7) low productivity. Assessing the validity of these reasons in explaining the deviation relies on evaluating the objective, subjective, and dependent factors shown below.

Possible Reasons and Explanations

Rank	Causal Rate	Factor Name	Explanation
40	Low	computer/availability	Computer not available
30	Low	system_testing/amount	Not enough system testing
30	High	dev_team/experience	Experienced development team
20	High	planning/quality	Good planning
15	High	CM/quality	Good configuration management
15	High	QA/quality	Good quality assurance
15	Low	dev_team/productivity	Low productivity

Objective Factors

Factor Name	Function
coding/productivity	LOC/EFF (LOC per hour)
design/productivity	MOD/EFF (Modules per hour)

Subjective Factors

Factor Name	Question	Responses (High,Low,Normal)
CM_plan/quality	What quality rating would you assign to this project's CM plan?	(High,Low,Normal)
CM_plan/use	Is this project using/following its CM plan?	(Yes,No,N/A)
code_reading/use	Is this project using code reading?	(Yes,No,N/A)
computer/reliability	What level of reliability would you assign to the development computer?	(High,Low,Normal)
dev_plan/quality	What quality rating would you assign to this project's development plan?	(High,Low,Normal)
dev_team/exper_w/application	How would you rate the team's experience with the project's application?	(High,Low,Normal)
dev_team/exper_w/environment	How would you rate the team's experience with the development environment?	(High,Low,Normal)
dev_team/exper_w/language	How would you rate the team's experience with the development language?	(High,Low,Normal)
dev_team/exper_w/tools	How would you rate the team's experience with the development tools in use?	(High,Low,Normal)
dev_team/quality	How would you rate the development team's overall quality?	(High,Low,Normal)
librarian/use	Is this project using a librarian?	(Yes,No,N/A)
mgmt_plan/quality	What quality rating would you assign to this project's management plan?	(High,Low,Normal)
plan_maintenance/quality	Is the set of plans (dev, mgmt, QA, CM, and test) being kept up to date?	(Yes,No,N/A)
QA_plan/quality	What quality rating would you assign to this project's QA plan?	(High,Low,Normal)
QA_plan/use	Is this project using/following its QA plan?	(Yes,No,N/A)
staffing_plan/quality	What quality rating would you assign to this project's staffing plan?	(High,Low,Normal)
system_testing/amount	How would you rate the amount of system testing being done on this project?	(High,Low,Normal)
terminal_per_pgmr/amount	How would you rate the number of terminals per programmer?	(High,Low,Normal)
test_plan/quality	What quality rating would you assign to this project's test plan?	(High,Low,Normal)

Figure 2-66 (1 of 2). Reasoning for Lower than Normal CPU Hours

Dependent Factors

Factor Name	Underlying Factors	Optimum Rating	Weight
CM/quality			
	CM_plan/use	High	1.0
	librarian/use	High	1.0
computer/availability			
	computer/reliability	High	1.0
	terminal_per_pgmr/amount	High	1.0
dev_team/experience			
	dev_team/exper_w/application	High	1.0
	dev_team/exper_w/environment	High	1.0
	dev_team/exper_w/language	High	1.0
	dev_team/exper_w/tools	High	1.0
dev_team/productivity			
	coding/productivity	High	5.0
	design/productivity	High	3.0
	dev_team/quality	High	2.0
planning_quality			
	CM_plan/quality	High	1.0
	dev_plan/quality	High	1.0
	mgmt_plan/quality	High	1.0
	plan_maintenance/quality	High	5.0
	QA_plan/quality	High	1.0
	staffing_plan/quality	High	1.0
	test_plan/quality	High	1.0
QA/quality			
	code_reading/use	High	1.0
	QA_plan/use	High	1.0

Figure 2-66 (2 of 2). Reasoning for Lower than Normal CPU Hours

2.3.1.1.3 Higher than Normal Total Staff Effort

The SME considers six possible reasons that could cause the total number of staff hours recorded for a project to be above normal. These reasons, in order of decreasing potential likelihood, are (1) problem larger than expected, (2) low productivity, (3) unstable code, (4) poor planning, (5) inexperienced development team, and (6) incomplete specifications. Assessing the validity of these reasons in explaining the deviation relies on evaluating the objective, subjective, and dependent factors shown below.

Possible Reasons and Explanations

Rank	Causal Rate	Factor Name	Explanation
30	High	estimate/accuracy	Problem larger than expected
20	Low	dev_team/productivity	Low productivity
20	High	source_module_change/amount	Unstable code
15	Low	planning/quality	Poor planning
10	Low	dev_team/experience	Inexperienced development team
10	Low	specs/completeness	Incomplete specifications

Objective Factors

Factor Name	Function
coding/productivity	LOC/EFF (LOC per hour)
design/productivity	MOD/EFF (Modules per hour)
source_module_change/amount	MCH (Module Changes)

Subjective Factors

Factor Name	Question	Responses (High,Low,Normal)
CM_plan/quality	What quality rating would you assign to this project's CM plan?	(High,Low,Normal)
dev_plan/quality	What quality rating would you assign to this project's development plan?	(High,Low,Normal)
dev_team/exper_w/application	How would you rate the team's experience with the project's application?	(High,Low,Normal)
dev_team/exper_w/environment	How would you rate the team's experience with the development environment?	(High,Low,Normal)
dev_team/exper_w/language	How would you rate the team's experience with the development language?	(High,Low,Normal)
dev_team/exper_w/tools	How would you rate the team's experience with the development tools in use?	(High,Low,Normal)
dev_team/quality	How would you rate the development team's overall quality?	(High,Low,Normal)
estimate/confidence	What is your confidence in the project size estimate?	(High,Low,N/A)
estimate_error/direction	If you're not confident in the estimate for this project, then it is	(High,Low,Confident)
mgmt_plan/quality	What quality rating would you assign to this project's management plan?	(High,Low,Normal)
plan_maintenance/quality	Is the set of plans (dev, mgmt, QA, CM, and test) being kept up to date?	(Yes,No,N/A)
QA_plan/quality	What quality rating would you assign to this project's QA plan?	(High,Low,Normal)
specs_outstand_quest/amount	How would you rate the number of outstanding specification questions?	(High,Low,Normal)
specs_TBDs/amount	How would you rate the number of specification TBDs for this project?	(High,Low,Normal)
staffing_plan/quality	What quality rating would you assign to this project's staffing plan?	(High,Low,Normal)
test_plan/quality	What quality rating would you assign to this project's test plan?	(High,Low,Normal)

Figure 2-67 (1 of 2). Reasoning for Higher than Normal Total Staff Hours

Dependent Factors

Factor Name	Underlying Factors	Optimum Rating	Weight
dev_team/experience			
	dev_team/exper_w/application	High	1.0
	dev_team/exper_w/environment	High	1.0
	dev_team/exper_w/language	High	1.0
	dev_team/exper_w/tools	High	1.0
dev_team/productivity			
	coding/productivity	High	5.0
	design/productivity	High	3.0
	dev_team/quality	High	2.0
estimate/accuracy			
	estimate_is_high/truth	High	1.0
	estimate_is_low/truth	Low	1.0
estimate_is_high/truth			
	estimate/confidence	Low	1.0
	estimate_error/direction	High	1.0
estimate_is_low/truth			
	estimate/confidence	Low	1.0
	estimate_error/direction	Low	1.0
planning/quality			
	CM_plan/quality	High	1.0
	dev_plan/quality	High	1.0
	mgmt_plan/quality	High	1.0
	plan_maintenance/quality	High	5.0
	QA_plan/quality	High	1.0
	staffing_plan/quality	High	1.0
	test_plan/quality	High	1.0
specs/completeness			
	specs_outstand_quest/amount	Low	1.0
	specs_TBDs/amount	Low	1.0

Figure 2-67 (2 of 2). Reasoning for Higher than Normal Total Staff Hours

C-2

2.3.1.1.4 Lower than Normal Total Staff Effort

The SME considers seven possible reasons that could cause the total number of staff hours recorded for a project to be below normal. These reasons, in order of decreasing potential likelihood, are (1) staffing up too slowly, (2) easy problem, (3) problem smaller than expected, (4) experienced development team, (5) not paying attention to deadlines, (6) high productivity, and (7) problem not understood. Assessing the validity of these reasons in explaining the deviation relies on evaluating the objective, subjective, and dependent factors shown below.

Possible Reasons and Explanations

Rank	Causal Rate	Factor Name	Explanation
40	Low	staffing/amount	Staffing up too slowly
30	Low	problem/difficulty	Easy problem
30	Low	estimate/accuracy	Problem smaller than expected
25	High	dev_team/experience	Experienced development team
20	Low	mgmt_team/deadline_sensitivity	Not paying attention to deadlines
20	High	dev_team/productivity	High productivity
20	Low	problem/understanding	Problem not understood

Objective Factors

Factor Name	Function
coding/productivity	LOC/EFF (LOC per hour)
design/productivity	MOD/EFF (Modules per hour)

Subjective Factors

Factor Name	Question	Responses (High,Low,Normal)
dev_team/exper_w/application	How would you rate the team's experience with the project's application?	(High,Low,Normal)
dev_team/exper_w/environment	How would you rate the team's experience with the development environment?	(High,Low,Normal)
dev_team/exper_w/language	How would you rate the team's experience with the development language?	(High,Low,Normal)
dev_team/exper_w/tools	How would you rate the team's experience with the development tools in use?	(High,Low,Normal)
dev_team/quality	How would you rate the development team's overall quality?	(High,Low,Normal)
estimate/confidence	What is your confidence in the project size estimate?	(High,Low,N/A)
estimate_error/direction	If you're not confident in the estimate for this project, then it is	(High,Low,Confident)
mgmt_team/deadline_sensitivity	Is this team paying attention to deadlines?	(Yes,No,N/A)
problem/difficulty	How would you rate the difficulty of the problem this project is working on?	(Difficult,Easy,Normal)
problem/understanding	How would you rate the team's understanding of the problem they are working on?	(Good,Poor,N/A)
staffing/direction	If staffing is not following a plan, would you say it was	(High,Low,N/A)
staffing_plan/quality	What quality rating would you assign to this project's staffing plan?	(High,Low,Normal)
staffing_plan/use	Is this project using/following its staffing plan?	(Yes,No,N/A)

Figure 2-68 (1 of 2). Reasoning for Lower than Normal Total Staff Hours

Dependent Factors

Factor Name	Underlying Factors	Optimum Rating	Weight
dev_team/experience			
	dev_team/exper_w/application	High	1.0
	dev_team/exper_w/environment	High	1.0
	dev_team/exper_w/language	High	1.0
	dev_team/exper_w/tools	High	1.0
dev_team/productivity			
	coding/productivity	High	5.0
	design/productivity	High	3.0
	dev_team/quality	High	2.0
estimate/accuracy			
	estimate_is_high/truth	High	1.0
	estimate_is_low/truth	Low	1.0
estimate_is_high/truth			
	estimate/confidence	Low	1.0
	estimate_error/direction	High	1.0
estimate_is_low/truth			
	estimate/confidence	Low	1.0
	estimate_error/direction	Low	1.0
staffing/amount			
	staffing_is_high/truth	High	1.0
	staffing_is_low/truth	Low	1.0
staffing/quality			
	staffing_plan/quality	High	1.0
	staffing_plan/use	High	1.0
staffing_is_high/truth			
	staffing/direction	High	1.0
	staffing/quality	Low	1.0
staffing_is_low/truth			
	staffing/direction	Low	1.0
	staffing/quality	Low	1.0

Figure 2-68 (2 of 2). Reasoning for Lower than Normal Total Staff Hours

2.3.1.1.5 Higher than Normal Lines of Code

The SME considers eight possible reasons that could cause the total number of lines of code recorded for a project to be above normal. These reasons, in order of decreasing potential likelihood, are (1) problem larger than expected, (2) lots of reused code, (3) experienced development team, (4) stable design, (5) not enough unit testing, (6) high productivity, (7) poor configuration management, and (8) poor quality assurance. Assessing the validity of these reasons in explaining the deviation relies on evaluating the objective, subjective, and dependent factors shown below.

Possible Reasons and Explanations

Rank	Causal Rate	Factor Name	Explanation
40	High	estimate/accuracy	Problem larger than expected
35	High	source_module_reuse/amount	Lots of reused code
30	High	dev_team/experience	Experienced development team
25	High	design/stability	Stable design
20	Low	unit_testing/amount	Not enough unit testing
20	High	dev_team/productivity	High productivity
15	Low	CM/quality	Poor configuration management
15	Low	QA/quality	Poor quality assurance

Objective Factors

Factor Name	Function
coding/productivity	LOC/EFF (LOC per hour)
design/productivity	MOD/EFF (Modules per hour)

Subjective Factors

Factor Name	Question	Responses (High,Low,Normal)
CM_plan/use	Is this project using/following its CM plan?	(Yes,No,N/A)
code_reading/use	Is this project using code reading?	(Yes,No,N/A)
design/stability	What level of stability would you assign to this project's design?	(High,Low,Normal)
dev_team/exper_w/application	How would you rate the team's experience with the project's application?	(High,Low,Normal)
dev_team/exper_w/environment	How would you rate the team's experience with the development environment?	(High,Low,Normal)
dev_team/exper_w/language	How would you rate the team's experience with the development language?	(High,Low,Normal)
dev_team/exper_w/tools	How would you rate the team's experience with the development tools in use?	(High,Low,Normal)
dev_team/quality	How would you rate the development team's overall quality?	(High,Low,Normal)
estimate/confidence	What is your confidence in the project size estimate?	(High,Low,N/A)
estimate_error/direction	If you're not confident in the estimate for this project, then it is	(High,Low,Confident)
librarian/use	Is this project using a librarian?	(Yes,No,N/A)
QA_plan/use	Is this project using/following its QA plan?	(Yes,No,N/A)
source_module_reuse/amount	How would you rate the level of module reuse on this project?	(High,Low,Normal)
unit_testing/amount	How would you rate the amount of unit testing being done on this project?	(High,Low,Normal)

Figure 2-69 (1 of 2). Reasoning for Higher than Normal Lines of Code

Dependent Factors

Factor Name	Underlying Factors	Optimum Rating	Weight
CM/quality			
	CM_plan/use	High	1.0
	librarian/use	High	1.0
dev_team/experience			
	dev_team/exper_w/application	High	1.0
	dev_team/exper_w/environment	High	1.0
	dev_team/exper_w/language	High	1.0
	dev_team/exper_w/tools	High	1.0
dev_team/productivity			
	coding/productivity	High	5.0
	design/productivity	High	3.0
	dev_team/quality	High	2.0
estimate/accuracy			
	estimate_is_high/truth	High	1.0
	estimate_is_low/truth	Low	1.0
estimate_is_high/truth			
	estimate/confidence	Low	1.0
	estimate_error/direction	High	1.0
estimate_is_low/truth			
	estimate/confidence	Low	1.0
	estimate_error/direction	Low	1.0
QA/quality			
	code_reading/use	High	1.0
	QA_plan/use	High	1.0

Figure 2-69 (2 of 2). Reasoning for Higher than Normal Lines of Code

2.3.1.1.6 Lower than Normal Lines of Code

The SME considers five possible reasons that could cause the total number of lines of code recorded for a project to be below normal. These reasons, in order of decreasing potential likelihood, are (1) problem smaller than expected, (2) team is wasting time, (3) incomplete design, (4) poor planning, and (5) too much unit testing. Assessing the validity of these reasons in explaining the deviation relies on evaluating the objective, subjective, and dependent factors shown below.

Possible Reasons and Explanations

Rank	Causal Rate	Factor Name	Explanation
40	Low	estimate/accuracy	Problem smaller than expected
35	Low	mgmt_team/control	Team is wasting time
30	Low	design/completeness	Incomplete design
20	Low	planning/quality	Poor planning
20	High	unit_testing/amount	Too much unit testing

Objective Factors

Factor Name	Function
none	

Subjective Factors

Factor Name	Question	Responses (High,Low,Normal)
CM_plan/quality	What quality rating would you assign to this project's CM plan?	(High,Low,Normal)
design_RIDs(CDR)/amount	How would you rate the number of RIDs at CDR?	(High,Low,Normal)
design_TBDs(CDR)/amount	How would you rate the number of TBDs at CDR?	(High,Low,Normal)
dev_plan/quality	What quality rating would you assign to this project's development plan?	(High,Low,Normal)
estimate/confidence	What is your confidence in the project size estimate?	(High,Low,N/A)
estimate_error/direction	If you're not confident in the estimate for this project, then it is	(High,Low,Confident)
mgmt_plan/quality	What quality rating would you assign to this project's management plan?	(High,Low,Normal)
mgmt_team/control	Is the team wasting time and appear to lack a sense of direction?	(No,Yes,N/A)
plan_maintenance/quality	Is the set of plans (dev, mgmt, QA, CM, and test) being kept up to date?	(Yes,No,N/A)
QA_plan/quality	What quality rating would you assign to this project's QA plan?	(High,Low,Normal)
staffing_plan/quality	What quality rating would you assign to this project's staffing plan?	(High,Low,Normal)
test_plan/quality	What quality rating would you assign to this project's test plan?	(High,Low,Normal)
unit_testing/amount	How would you rate the amount of unit testing being done on this project?	(High,Low,Normal)

Figure 2-70 (1 of 2). Reasoning for Lower than Normal Lines of Code

Dependent Factors

Factor Name	Underlying Factors	Rating	Optimum Weight
design/completeness			
	design_RIDs(CDR)/amount	Low	1.0
	design_TBDs(CDR)/amount	Low	1.0
estimate/accuracy			
	estimate_is_high/truth	High	1.0
	estimate_is_low/truth	Low	1.0
estimate_is_high/truth			
	estimate/confidence	Low	1.0
	estimate_error/direction	High	1.0
estimate_is_low/truth			
	estimate/confidence	Low	1.0
	estimate_error/direction	Low	1.0
planning/quality			
	CM_plan/quality	High	1.0
	dev_plan/quality	High	1.0
	mgmt_plan/quality	High	1.0
	plan_maintenance/quality	High	5.0
	QA_plan/quality	High	1.0
	staffing_plan/quality	High	1.0
	test_plan/quality	High	1.0

Figure 2-70 (2 of 2). Reasoning for Lower than Normal Lines of Code

2.3.1.1.7 Higher than Normal Reported Errors

The SME considers five possible reasons that could cause the total number of reported errors recorded for a project to be above normal. These reasons, in order of decreasing potential likelihood, are (1) team is reporting inconsequential errors, (2) inexperienced development team, (3) poor use of methodology, (4) complex problem, and (5) unreliable system. Assessing the validity of these reasons in explaining the deviation relies on evaluating the objective, subjective, and dependent factors shown below.

Possible Reasons and Explanations

Rank	Causal Rate	Factor Name	Explanation
50	High	dev_team/nit_picking	Team is reporting inconsequential errors
25	Low	dev_team/experience	Inexperienced development team
20	Low	process_methodology/use	Poor use of methodology
15	High	problem/complexity	Complex problem
10	Low	software/reliability	Unreliable system

Objective Factors

Factor Name	Function
source_code_changes/rate	RCH/LOC (Reported changes per LOC)

Subjective Factors

Factor Name	Question	Responses (High,Low,Normal)
CM_plan/use	Is this project using/following its CM plan?	(Yes,No,N/A)
code_commenting/use	What level of code commenting is being used in this project's software?	(Lots,Minimal,Normal)
code_reading/amount	How much code reading is being done on this project?	(Lots,Minimal,Normal)
code_reading/quality	What quality rating would you assign to this project's code reading?	(High,Low,Normal)
code_reading/use	Is this project using code reading?	(Yes,No,N/A)
coding_CM/use	Is this project using formal configuration management methods during coding?	(Yes,No,N/A)
coding_QA/use	Is this project using formal quality assurance methods during coding?	(Yes,No,N/A)
design/quality	What quality rating would you assign to this project's design?	(High,Low,Normal)
design/stability	What level of stability would you assign to this project's design?	(High,Low,Normal)
design_methodology/use	Is this project using a formal design methodology?	(Yes,No,N/A)
dev_team/exper_w/application	How would you rate the team's experience with the project's application?	(High,Low,Normal)
dev_team/exper_w/environment	How would you rate the team's experience with the development environment?	(High,Low,Normal)
dev_team/exper_w/language	How would you rate the team's experience with the development language?	(High,Low,Normal)
dev_team/exper_w/tools	How would you rate the team's experience with the development tools in use?	(High,Low,Normal)
dev_team/nit_picking	Is the team reporting insignificant or cosmetic errors (nit picking)?	(Yes,No,N/A)
librarian/use	Is this project using a librarian?	(Yes,No,N/A)
problem/complexity	How would you rate the complexity of the problem this project is working on?	(Complex,Simple,Norm)
specs/stability	How would you rate the stability of the specifications for this project?	(High,Low,Normal)
specs_methodology/use	Is this project using a formal specification methodology?	(Yes,No,N/A)
testing_methodology/use	Is this project using a formal testing methodology?	(Yes,No,N/A)
unit_testing/amount	How would you rate the amount of unit testing being done on this project?	(High,Low,Normal)
unit_testing/quality	What quality rating would you assign to unit testing on this project?	(High,Low,Normal)
unit_testing/use	Is unit testing being done on this project?	(Yes,No,N/A)

Figure 2-71 (1 of 2). Reasoning for Higher than Normal Reported Errors

Dependent Factors

Factor Name	Underlying Factors	Optimum Rating	Weight
CM/quality			
	CM_plan/use	High	1.0
	librarian/use	High	1.0
coding_methodology/use			
	code_commenting/use	High	1.0
	code_reading/use	High	1.0
	coding_CM/use	High	1.0
	coding_QA/use	High	1.0
	unit_testing/use	High	1.0
dev_team/experience			
	dev_team/exper_w/application	High	1.0
	dev_team/exper_w/environment	High	1.0
	dev_team/exper_w/language	High	1.0
	dev_team/exper_w/tools	High	1.0
process_methodology/use			
	coding_methodology/use	High	1.0
	design_methodology/use	High	1.0
	specs_methodology/use	High	1.0
	testing_methodology/use	High	1.0
software/reliability			
	CM/quality	High	1.0
	code_reading/amount	High	1.0
	code_reading/quality	High	1.0
	design/quality	High	1.0
	design/stability	High	1.0
	source_code_changes/rate	Low	1.0
	specs/stability	High	1.0
	unit_testing/amount	High	1.0
	unit_testing/quality	High	1.0

Figure 2-71 (2 of 2). Reasoning for Higher than Normal Reported Errors

2.3.1.1.8 Lower than Normal Reported Errors

The SME considers seven possible reasons that could cause the total number of reported errors recorded for a project to be below normal. These reasons, in order of decreasing potential likelihood, are (1) team is not submitting SEL forms, (2) good unit testing, (3) not enough system testing, (4) experienced development team, (5) reliable system, (6) lots of reused code, and (7) easy problem. Assessing the validity of these reasons in explaining the deviation relies on evaluating the objective, subjective, and dependent factors shown below.

Possible Reasons and Explanations

Rank	Causal Rate	Factor Name	Explanation
40	Low	dev_team/forms_submission	Team is not submitting SEL forms
30	High	unit_testing/quality	Good unit testing
20	Low	system_testing/amount	Not enough system testing
15	High	dev_team/experience	Experienced development team
10	High	software/reliability	Reliable system
10	High	source_module_reuse/amount	Lots of reused code
5	Low	problem/difficulty	Easy problem

Objective Factors

Factor Name	Function
source_code_changes/rate	RCH/LOC (Reported changes per LOC)

Subjective Factors

Factor Name	Question	Responses (High,Low,Normal)
CM_plan/use	Is this project using/following its CM plan?	(Yes,No,N/A)
code_reading/amount	How much code reading is being done on this project?	(Lots,Minimal,Normal)
code_reading/quality	What quality rating would you assign to this project's code reading?	(High,Low,Normal)
design/quality	What quality rating would you assign to this project's design?	(High,Low,Normal)
design/stability	What level of stability would you assign to this project's design?	(High,Low,Normal)
dev_team/exper_w/application	How would you rate the team's experience with the project's application?	(High,Low,Normal)
dev_team/exper_w/environment	How would you rate the team's experience with the development environment?	(High,Low,Normal)
dev_team/exper_w/language	How would you rate the team's experience with the development language?	(High,Low,Normal)
dev_team/exper_w/tools	How would you rate the team's experience with the development tools in use?	(High,Low,Normal)
dev_team/forms_submission	Is the team submitting SEL forms (especially COFs and CRFs) on time?	(Yes,No,N/A)
librarian/use	Is this project using a librarian?	(Yes,No,N/A)
problem/difficulty	How would you rate the difficulty of the problem this project is working on?	(Difficult,Easy,Normal)
source_module_reuse/amount	How would you rate the level of module reuse on this project?	(High,Low,Normal)
specs/stability	How would you rate the stability of the specifications for this project?	(High,Low,Normal)
system_testing/amount	How would you rate the amount of system testing being done?	(High,Low,Normal)
unit_testing/amount	How would you rate the amount of unit testing being done on this project?	(High,Low,Normal)
unit_testing/quality	What quality rating would you assign to unit testing on this project?	(High,Low,Normal)

Figure 2-72 (1 of 2). Reasoning for Lower than Normal Reported Errors

Dependent Factors

Factor Name	Underlying Factors	Optimum Rating	Weight
CM/quality			
	CM_plan/use	High	1.0
	librarian/use	High	1.0
dev_team/experience			
	dev_team/exper_w/application	High	1.0
	dev_team/exper_w/environment	High	1.0
	dev_team/exper_w/language	High	1.0
	dev_team/exper_w/tools	High	1.0
software/reliability			
	CM/quality	High	1.0
	code_reading/amount	High	1.0
	code_reading/quality	High	1.0
	design/quality	High	1.0
	design/stability	High	1.0
	source_code_changes/rate	Low	1.0
	specs/stability	High	1.0
	unit_testing/amount	High	1.0
	unit_testing/quality	High	1.0

Figure 2-72 (2 of 2). Reasoning for Lower than Normal Reported Errors

2.3.1.2 General-Purpose Use of the Knowledge Base

The SME incorporates a set of general-purpose services commonly used with the knowledge base. The services are referenced by SME functions to provide needed services associated with the knowledge base. These services include

- *Rate Objective Factor*
- *Rate Subjective Factor*
- *Rate Dependent Factor*
- *Evaluate Reason*

The following sections discuss each of these services and detail the algorithms behind the actions they perform.

2.3.1.2.1 Rate Objective Factor

Purpose

Evaluates an objective factor as of the current date by comparing its actual value computed from measure data to its expected model value. The factor is assigned a rating (i.e., High, Low, Normal, or Unknown) and a certainty.

Required Data

- Objective factor (input value)
- Measure data (for any referenced measures)
- Schedule data
- Schedule model (in *Convert Date To Phase*)
- Estimate data
- Estimate set model (in *Determine Normal Estimate Set*)
- Measure model (for referenced measures) (in *Convert Phase To Measure*)

Steps

1. For the expression in the factor's function, obtain the actual data value of any referenced measure as of the current date from the measure data.
2. If the expression references a single measure, set the factor's actual value to the actual measure value. If the expression references a ratio of two measures, set the factor's actual value to the ratio of the two actual measure values obtained. (*Actual Factor Value*)
3. Use *Get Project Dates* to obtain the planned project start and end dates from the current schedule data.
4. On the basis of the project start and end dates, use *Convert Date to Phase* to translate the current date to the phase and elapsed fraction of phase that normally should be reached on that date.
5. Use *Get Project Magnitude* on the current estimate data to obtain the measure and estimated completion value for that measure which is most indicative of the project's magnitude.
6. On the basis of that magnitude, use *Determine Normal Estimate Set* to create a normal set of estimates for the project.
7. For the expression in the factor's function, use *Convert Phase to Measure* to obtain the expected measure value of any referenced measures at the desired phase and fraction of phase, given the normal completion value of the measure, from the measure model. ($Expected\ Measure\ Value_{Normal}$)

8. Compute the upper and lower normal bounds on any expected measure values obtained by adding and subtracting, respectively, the scaled value of the normal deviation stored in the model from each expected measure value via

$$\text{Expected Measure Value}_{High} = \text{Expected Measure Value}_{Normal} + (\text{Normal Deviation} * \text{Normal Completion Value})$$

$$\text{Expected Measure Value}_{Low} = \text{Expected Measure Value}_{Normal} - (\text{Normal Deviation} * \text{Normal Completion Value})$$

9. If the expression references a single measure, set the factor's normal upper and lower values to the expected high and low model value just obtained. If the expression references a ratio of two measures, set the factor's normal upper and lower values to the possible extremes of the ratio of the two model values via

$$\text{Normal Measure Value}_{High} = \text{Expected Measure Value}_{High}[\text{Numerator}] \,/\, \text{Expected Measure Value}_{Low}[\text{Denominator}]$$

$$\text{Normal Measure Value}_{Low} = \text{Expected Measure Value}_{Low}[\text{Numerator}] \,/\, \text{Expected Measure Value}_{High}[\text{Denominator}]$$

Note: The upper bound for the normal measure value is considered infinite (or unbounded) if the denominator of the first equation is zero. The lower bound for the normal measure value is considered unknown (or indeterminate) if the denominator of the second equation is zero.

10. Set the objective factor's rating as follows:

if (*Normal Measure Value*$_{Low}$ = *Unknown*) *or*
  if (*Normal Measure Value*$_{High}$ = *0.0*) *Factor Rating* = *Unknown*

if (*Actual Factor Value* < *Normal Measure Value*$_{Low}$) *Factor Rating* = *Low*

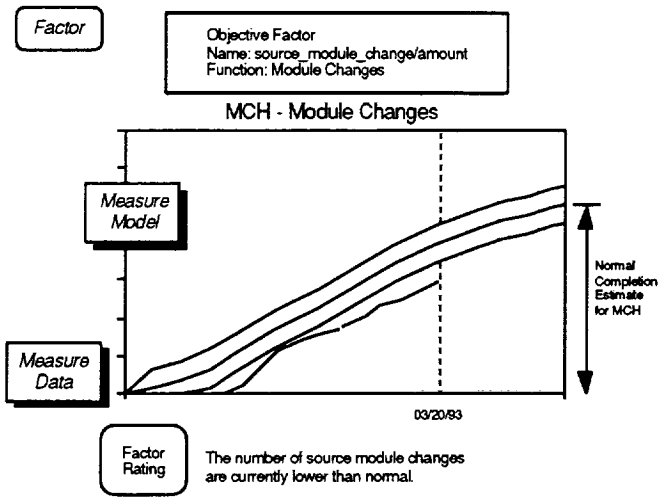
if (*Actual Factor Value* > *Normal Measure Value*$_{High}$) *Factor Rating* = *High*

otherwise *Factor Rating* = *Normal*

11. Set the objective factor's certainty as follows:

if (*Factor Rating* = *Unknown*) *Factor Certainty* = *0.0*

otherwise *Factor Certainty* = *1.0*



STEPS

1. Obtain the actual measure value for module changes as of the current date.

2. Convert the current date to an expected phase.

3. Generate a set of normal estimates for the project to obtain a completion estimate for module changes.

4. For this completion value, get the expected measure value at the desired phase and the normal range about that measure value.

5. Since the actual value is below the normal range for module changes, rate the factor as "Low."

Figure 2-73. Rating an Objective Factor

2.3.1.2.2 Rate Subjective Factor

Purpose

Evaluates a subjective factor as of the current date on the basis of the project's subjective data supplied by the manager. The factor is assigned a rating (i.e., High, Low, Normal, or Unknown) and a certainty.

Required Data

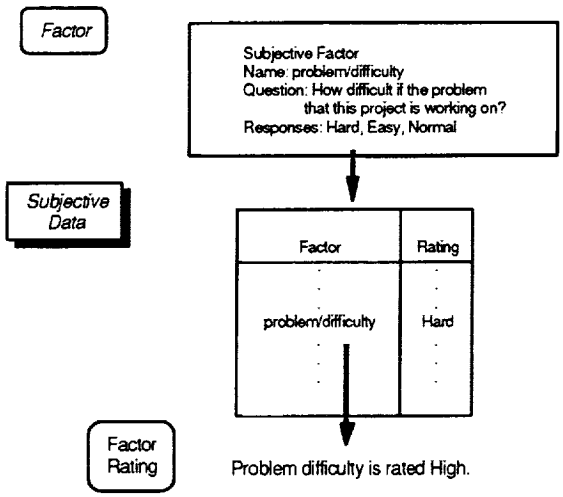
- Subjective factor (input value)
- Subjective data

Steps

1. Locate the name of the input subjective factor in the subjective data supplied by the manager for the project.
2. Translate the manager's rating for that entry in the subjective data to a factor rating of either High, Low, Normal, or Unknown on the basis of the allowable responses in the input subjective factor. (*Factor Rating*)
3. Set the subjective factor's certainty as follows:

   if (Factor Rating = Unknown) Factor Certainty = 0.0

   otherwise Factor Certainty = 1.0



STEPS

1. To rate a subjective factor (e.g., problem/difficulty), locate the factor by name in the project's subjective data and obtain its current rating.

2. Convert the factor's rating to either High, Low, Normal, or Unknown using the allowable responses defined in the factor (e.g., a value of "Hard" for problem/difficulty translates to a rating of "High").

3. If the factor was not found in the subjective data, its rating is set to "Unknown" and its certainty is set to 0.0. Otherwise, the factor's certainty is set to 1.0.

Figure 2-74. Rating a Subjective Factor

2.3.1.2.3 Rate Dependent Factor

Purpose

Evaluates a dependent factor as of the current date that consists of two or more underlying objective, subjective, or dependent factors. The factor is assigned a rating (i.e., High, Low, Normal, or Unknown) and a certainty.

Required Data

- Dependent factor (input value)
- Factors (associated with specified dependent factor)

Steps

1. For each underlying factor associated with the input dependent factor, rate the factor on the basis of its type. If the factor is objective, use *Rate Objective Factor*. If the factor is subjective, use *Rate Subjective Factor*. If the factor is dependent, recursively use this algorithm *Rate Dependent Factor*.
   (*Factor Rating[i] and Factor Certainty[i]*)

2. Assign the value of the dependent factor to the weighted average of the underlying known factor ratings obtained for each factor, *1* through *K*, using

$$\text{Factor Value} = \left(\sum_{i=1}^{K} \text{Factor Weight}[i] * \text{Factor Certainty}[i] * \text{Same}\right) / \left(\sum_{i=1}^{K} \text{Factor Weight}[i] * \text{Known}\right)$$

   where *Same* compares the underlying *Factor Rating [i]* with the *Optimum Rating [i]* and returns
   0 if *Factor Rating [i]* is Unknown or Normal
   1 if *Factor Rating [i]* matches *Optimum Rating [i]*
   -1 if *Factor Rating [i]* does not match *Optimum Rating [i]*

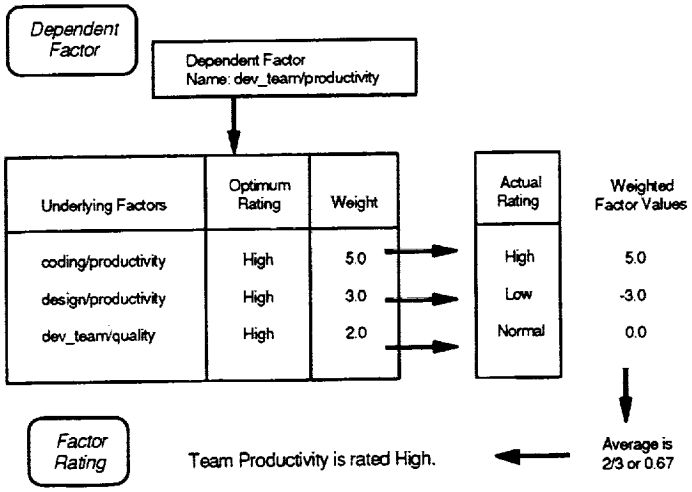
   and where *Known* examines the underlying *Factor Rating [i]* and returns
   0 if *Factor Rating [i]* is Unknown
   1 otherwise

3. Set the rating for the dependent factor by rounding the value assigned to the factor as follows:

   if (Factor Value >= 0.5) Factor Rating = High
   if (Factor Value <= -0.5) Factor Rating = Low
   otherwise Factor Rating = Normal

4. Set the certainty of the dependent factor to the weighted average of the underlying factor certainties obtained for each factor, *1* through *K*, using

$$\text{Factor Certainty} = \left(\sum_{i=1}^{K} \text{Factor Weight}[i] * \text{Factor Certainty}[i]\right) / \left(\sum_{i=1}^{K} \text{Factor Weight}[i]\right)$$



STEPS

1. Each underlying factor is evaluated based on its type (e.g., two objective factors and one subjective factor).

2. The value of the dependent factor is assigned to a weighted average of 0.67 (e.g., the sum of 5, -3, and 0 divided by 3).

3. The assigned value is rounded and translated to a rating (e.g., 0.67 is "High").

4. The certainty of the dependent factor, not depicted, is set to a weighted average (e.g., 1.0).

Figure 2-75. Rating a Dependent Factor

2.3.1.2.4 Evaluate Reason

Purpose

Calculates the relative actual ranking as of the current date for a specified knowledge base reason.

Required Data

- Reason (input value)
- Factor (associated with specified reason)

Steps

1. For the factor identified with the input reason, rate the factor on the basis of its type. If the factor is objective, use *Rate Objective Factor*. If the factor is subjective, use *Rate Subjective Factor*. If the factor is dependent, use *Rate Dependent Factor*. (*Factor Rating and Factor Certainty*)

2. Set the actual rating of the reason to the rating of the factor using

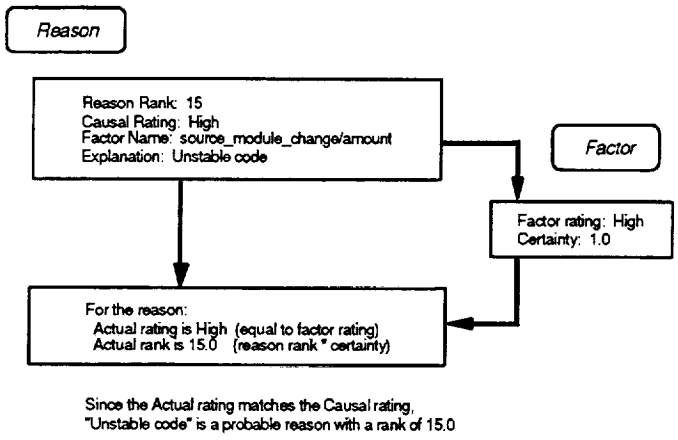
   Actual Rating = Factor Rating

3. Set the actual ranking of the reason on the basis of the reason's weighted rank and the factor's certainty using

   *Actual Rank = Reason Rank * Factor Certainty*

4. If the reason's actual rating does not match the reason's causal rating, negate the actual ranking of the reason to indicate that it is not a reason using

   *If (Causal Rating <> Actual Rating) Actual Rank = -1.0 * Actual Rank*



STEPS

1. The factor for the reason "Unstable code" is evaluated (source_module_change/-amount is rated "High" with a certainty of 1.0).

2. The actual rating of the reason is set to "High" (i.e., the factor rating) and its actual rank to 15.0 (i.e., the reason's rank times the factor's certainty of 1.0).

3. Since the actual rating matches the causal rating, actual rank is not negated and indicates "Unstable code" is a likely cause.

Figure 2-76. Evaluating a Knowledge Base Reason

2.3.2 Rule Base

Purpose

Describes a collection of captured management experience that uses a set of rules to evaluate the observed ratios of key pairs of measures to assess a project's current status.

Description

The rule base consists of two associated tables that (1) identify the set of rules to be evaluated on the basis of the present life-cycle phase and the observed deviations in the ratios of a project's measures from what is considered normal and (2) specify the interpretations to associate with those rules. The list of rules contains a series of conditions, with each condition associated with one or more possible interpretations. The interpretations are encoded and map to an entry in a list of explanations used for display purposes. Each rule in the rule base is evaluated based on the present life-cycle phase and current measure data for the project. If the rule's condition evaluates to true, the associated weighted interpretations are considered valid and added to an assertion list. Attempts to duplicate an interpretation in the assertion list result in one entry weighted to reflect both conditions. The SME rule base currently contains rules that address deviations in nine specific ratios of project measures.

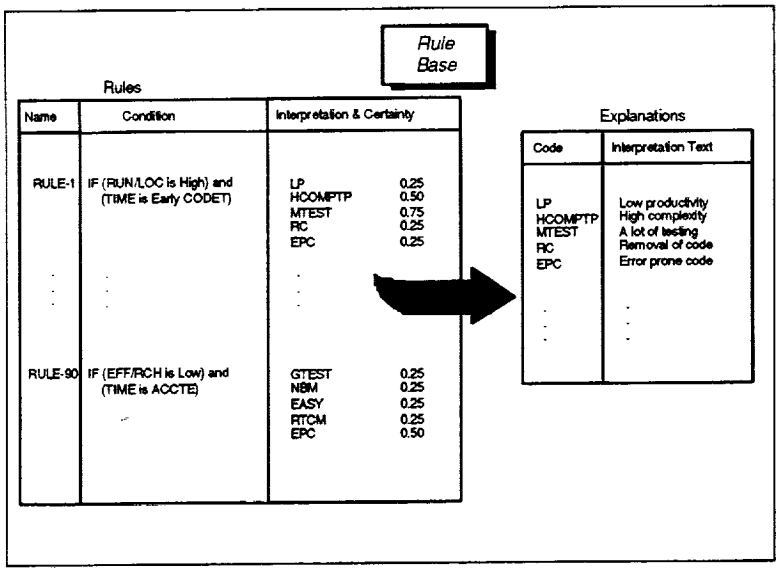


Figure 2-77. Rule Base for the SME

Source

Defined as part of the SME based on past experience

Assumptions

None

Instances

The SME has one rule base.

Structure

Two tables consisting of a rule list and an explanation list. The rule list defines all the rules in the rule base. Each record in the table describes one rule and contains the rule name, the condition to be evaluated for applying the rule, and one or more possible interpretations to consider if the rule's condition is true. Each possible interpretation in the rule consists of a pair of values—an encoded identifier and a weighted certainty. The explanation list is a table with two columns—an encoded identifier for an interpretation and the explanatory text to display for that interpretation.

2.3.2.1 Captured Knowledge

The SME captures reasoning in the rule base that encompasses deviations in nine specific ratios of measures. The deviations may be either above normal (high) or below normal (low). This reasoning covers

- Above Normal Computer Runs per Line of Code
- Below Normal Computer Runs per Line of Code
- Above Normal Computer Hours per Line of Code
- Below Normal Computer Hours per Line of Code
- Above Normal Reported Changes per Line of Code
- Below Normal Reported Changes per Line of Code
- Above Normal Total Staff Hours per Line of Code
- Below Normal Total Staff Hours per Line of Code
- Above Normal Computer Hours per Computer Run
- Below Normal Computer Hours per Computer Run
- Above Normal Reported Changes per Computer Run
- Below Normal Reported Changes per Computer Run
- Above Normal Total Staff Hours per Computer Run
- Below Normal Total Staff Hours per Computer Run
- Above Normal Computer Hours per Reported Change
- Below Normal Computer Hours per Reported Change
- Above Normal Total Staff Hours per Reported Change
- Below Normal Total Staff Hours per Reported Change

The following sections describe the specific rules captured in the rule base that address deviations in the ratios of measures and present a set of general-purpose algorithms commonly used with the rule base.

2.3.2.1.1 Above Normal Computer Runs per Line of Code

The SME considers five rules that address the case where the number of computer runs per line of code for a project is above normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-1	IF (Computer Runs per Line of Code are Above Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Low productivity	(0.25)
	High complexity	(0.50)
	A lot of testing	(0.75)
	Removal of code by testing or transporting	(0.25)
	Error prone code	(0.75)
RULE-2	IF (Computer Runs per Line of Code are Above Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Low productivity	(0.25)
	High complexity	(0.75)
	A lot of testing	(0.75)
	Removal of code by testing or transporting	(0.50)
	Unstable specifications	(0.50)
	Error prone code	(0.75)
RULE-3	IF (Computer Runs per Line of Code are Above Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Low productivity	(0.25)
	High complexity	(0.75)
	A lot of testing	(0.75)
	Removal of code by testing or transporting	(0.50)
	Unstable specifications	(0.50)
	Error prone code	(0.75)
RULE-4	IF (Computer Runs per Line of Code are Above Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Low productivity	(0.25)
	High complexity	(0.75)
	A lot of testing	(0.75)
	Removal of code by testing or transporting	(0.25)
	Unstable specifications	(0.50)
	Error prone code	(0.75)
RULE-5	IF (Computer Runs per Line of Code are Above Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	High complexity	(0.75)
	A lot of testing	(0.50)
	Unstable specifications	(0.25)
	Error prone code	(0.75)

Figure 2-78. Rules for Above Normal Computer Runs per Line of Code

2.3.2.1.2 Below Normal Computer Runs per Line of Code

The SME considers five rules that address the case where the number of computer runs per line of code for a project is below normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-6	IF (Computer Runs per Line of Code are Below Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.25)
	Influx of transported code	(0.75)
	Little or not enough online testing being done	(0.75)
	Little executable code being developed	(0.25)
	Computer problems, inaccessibility or environment constraints	(0.25)
RULE-7	IF (Computer Runs per Line of Code are Below Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	Influx of transported code	(0.50)
	Near build or milestone date	(0.50)
	Little or not enough online testing being done	(0.75)
	Little executable code being developed	(0.50)
	Computer problems, inaccessibility or environment constraints	(0.75)
RULE-8	IF (Computer Runs per Line of Code are Below Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	Influx of transported code	(0.25)
	Near build or milestone date	(0.50)
	Little or not enough online testing being done	(0.75)
	Little executable code being developed	(0.25)
	Computer problems, inaccessibility or environment constraints	(0.75)
RULE-9	IF (Computer Runs per Line of Code are Below Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	Little or not enough online testing being done	(0.75)
	Computer problems, inaccessibility or environment constraints	(0.75)
	Good testing or test plan	(0.75)
RULE-10	IF (Computer Runs per Line of Code are Below Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	Little or not enough online testing being done	(0.25)
	Computer problems, inaccessibility or environment constraints	(0.25)
	Good testing or test plan	(0.75)

Figure 2-79. Rules for Below Normal Computer Runs per Line of Code

2.3.2.1.3 Above Normal Computer Hours per Line of Code

The SME considers five rules that address the case where the number of computer hours per line of code for a project is above normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-11	IF (Computer Hours per Line of Code are Above Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Computation bound algorithms run or tested	(0.50)
	Low productivity	(0.25)
	A lot of testing	(0.75)
	Removal of code by testing or transporting	(0.25)
RULE-12	IF (Computer Hours per Line of Code are Above Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Computation bound algorithms run or tested	(0.75)
	Low productivity	(0.25)
	Unstable specifications	(0.25)
	A lot of testing	(0.75)
	Unit testing being done	(0.75)
	Removal of code by testing or transporting	(0.25)
	Loose configuration management or unstructured development	(0.75)
	Error prone code	(0.75)
RULE-13	IF (Computer Hours per Line of Code are Above Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Computation bound algorithms run or tested	(0.75)
	Low productivity	(0.25)
	Unstable specifications	(0.75)
	A lot of testing	(0.75)
	Unit testing being done	(0.75)
	Removal of code by testing or transporting	(0.50)
	Many design changes	(0.75)
	Error prone code	(0.75)
RULE-14	IF (Computer Hours per Line of Code are Above Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Computation bound algorithms run or tested	(0.75)
	Low productivity	(0.25)
	Unstable specifications	(0.50)
	A lot of testing	(0.75)
	Removal of code by testing or transporting	(0.25)
	Error prone code	(0.75)
RULE-15	IF (Computer Hours per Line of Code are Above Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Computation bound algorithms run or tested	(0.75)
	Unstable specifications	(0.50)
	A lot of testing	(0.75)
	Error prone code	(0.75)

Figure 2-80. Rules for Above Normal Computer Hours per Line of Code

2.3.2.1.4 Below Normal Computer Hours per Line of Code

The SME considers five rules that address the case where the number of computer hours per line of code for a project is below normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-16	IF (Computer Hours per Line of Code are Below Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Influx of transported code	(0.75)
	Little or not enough online testing being done	(0.75)
	Little executable code being developed	(0.75)
	Error prone code	(0.25)
RULE-17	IF (Computer Hours per Line of Code are Below Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Influx of transported code	(0.75)
	Near build or milestone date	(0.50)
	Little or not enough online testing being done	(0.75)
	Little executable code being developed	(0.75)
	Tight management plan or good configuration control	(0.75)
RULE-18	IF (Computer Hours per Line of Code are Below Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Influx of transported code	(0.25)
	Near build or milestone date	(0.75)
	Little or not enough online testing being done	(0.75)
	Little executable code being developed	(0.25)
RULE-19	IF (Computer Hours per Line of Code are Below Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	Near build or milestone date	(0.25)
	Little or not enough online testing being done	(0.50)
RULE-20	IF (Computer Hours per Line of Code are Below Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)

Figure 2-81. Rules for Below Normal Computer Hours per Line of Code

2.3.2.1.5 Above Normal Reported Changes per Line of Code

The SME considers five rules that address the case where the number of reported changes per line of code for a project is above normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-21	IF (Reported Changes per Line of Code are Above Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	Error prone code	(0.50)
	Unstable specifications	(0.25)
	Removal of code by testing or transporting	(0.50)
	Loose configuration management or unstructured development	(0.50)
	Near build or milestone date	(0.50)
RULE-22	IF (Reported Changes per Line of Code are Above Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	Error prone code	(0.75)
	Unstable specifications	(0.50)
	Removal of code by testing or transporting	(0.50)
	Loose configuration management or unstructured development	(0.75)
	Near build or milestone date	(0.50)
RULE-23	IF (Reported Changes per Line of Code are Above Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	Error prone code	(0.75)
	Unstable specifications	(0.75)
	Removal of code by testing or transporting	(0.25)
	Loose configuration management or unstructured development	(0.75)
	Near build or milestone date	(0.50)
RULE-24	IF (Reported Changes per Line of Code are Above Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	Error prone code	(0.75)
	Unstable specifications	(0.75)
	Removal of code by testing or transporting	(0.25)
	Loose configuration management or unstructured development	(0.75)
	Near build or milestone date	(0.25)
RULE-25	IF (Reported Changes per Line of Code are Above Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	Error prone code	(0.50)
	Unstable specifications	(0.50)
	Removal of code by testing or transporting	(0.25)
	Loose configuration management or unstructured development	(0.50)
	Near build or milestone date	(0.25)

Figure 2-82. Rules for Above Normal Reported Changes per Line of Code

2.3.2.1.6 Below Normal Reported Changes per Line of Code

The SME considers five rules that address the case where the number of reported changes per line of code for a project is below normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-26	IF (Reported Changes per Line of Code are Below Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Influx of transported code	(0.75)
	Near build or milestone date	(0.25)
	Good solid and reliable code	(0.50)
	Poor testing	(0.50)
	Change backlog or holding changes	(0.75)
	Low complexity	(0.50)
	Computer problems, inaccessibility or environment constraints	(0.50)
	Tight management plan or good configuration control	(0.50)
RULE-27	IF (Reported Changes per Line of Code are Below Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Influx of transported code	(0.50)
	Near build or milestone date	(0.25)
	Good solid and reliable code	(0.75)
	Poor testing	(0.50)
	Change backlog or holding changes	(0.75)
	Low complexity	(0.50)
	Computer problems, inaccessibility or environment constraints	(0.50)
	Tight management plan or good configuration control	(0.75)
RULE-28	IF (Reported Changes per Line of Code are Below Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Influx of transported code	(0.25)
	Near build or milestone date	(0.25)
	Good solid and reliable code	(0.75)
	Poor testing	(0.50)
	Change backlog or holding changes	(0.50)
	Low complexity	(0.75)
	Computer problems, inaccessibility or environment constraints	(0.25)
	Tight management plan or good configuration control	(0.75)
RULE-29	IF (Reported Changes per Line of Code are Below Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Influx of transported code	(0.25)
	Near build or milestone date	(0.25)
	Good solid and reliable code	(0.75)
	Poor testing	(0.25)
	Change backlog or holding changes	(0.25)
	Low complexity	(0.75)
	Computer problems, inaccessibility or environment constraints	(0.25)
	Tight management plan or good configuration control	(0.50)
RULE-30	IF (Reported Changes per Line of Code are Below Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Influx of transported code	(0.25)
	Near build or milestone date	(0.25)
	Good solid and reliable code	(0.75)
	Poor testing	(0.25)
	Change backlog or holding changes	(0.25)
	Low complexity	(0.75)
	Computer problems, inaccessibility or environment constraints	(0.25)
	Tight management plan or good configuration control	(0.25)

Figure 2-83. Rules for Below Normal Reported Changes per Line of Code

2.3.2.1.7 *Above Normal Total Staff Hours per Line of Code*

The SME considers five rules that address the case where the number of total staff hours per line of code for a project is above normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-31	IF (Total Staff Hours per Line of Code are Above Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	High complexity	(0.75)
	Error prone code	(0.25)
	Unstable specifications	(0.50)
	Removal of code by testing or transporting	(0.50)
	Changes hard to isolate	(0.25)
	Changes hard to make	(0.25)
	Low productivity	(0.50)
RULE-32	IF (Total Staff Hours per Line of Code are Above Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	High complexity	(0.75)
	Error prone code	(0.50)
	Unstable specifications	(0.50)
	Removal of code by testing or transporting	(0.25)
	Changes hard to isolate	(0.25)
	Changes hard to make	(0.25)
	Low productivity	(0.75)
RULE-33	IF (Total Staff Hours per Line of Code are Above Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	High complexity	(0.75)
	Error prone code	(0.75)
	Unstable specifications	(0.50)
	Removal of code by testing or transporting	(0.25)
	Changes hard to isolate	(0.50)
	Changes hard to make	(0.50)
	Low productivity	(0.75)
RULE-34	IF (Total Staff Hours per Line of Code are Above Normal) and (Project is in system test phase)	
	THEN interpretations are	
	High complexity	(0.50)
	Error prone code	(0.75)
	Unstable specifications	(0.25)
	Removal of code by testing or transporting	(0.25)
	Changes hard to isolate	(0.50)
	Changes hard to make	(0.50)
	Low productivity	(0.75)
RULE-35	IF (Total Staff Hours per Line of Code are Above Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	High complexity	(0.25)
	Error prone code	(0.50)
	Unstable specifications	(0.25)
	Removal of code by testing or transporting	(0.25)
	Changes hard to isolate	(0.50)
	Changes hard to make	(0.50)
	Low productivity	(0.75)

Figure 2-84. Rules for Above Normal Total Staff Hours per Line of Code

2.3.2.1.8 Below Normal Total Staff Hours per Line of Code

The SME considers five rules that address the case where the number of total staff hours per line of code for a project is below normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-36	IF (Total Staff Hours per Line of Code are Below Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Influx of transported code	(0.50)
	Near build or milestone date	(0.25)
	Low complexity	(0.75)
	High productivity	(0.50)
	Lack of thorough testing	(0.50)
RULE-37	IF (Total Staff Hours per Line of Code are Below Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Influx of transported code	(0.50)
	Near build or milestone date	(0.25)
	Low complexity	(0.75)
	High productivity	(0.75)
	Lack of thorough testing	(0.50)
RULE-38	IF (Total Staff Hours per Line of Code are Below Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Influx of transported code	(0.25)
	Near build or milestone date	(0.25)
	Low complexity	(0.75)
	High productivity	(0.75)
	Lack of thorough testing	(0.50)
RULE-39	IF (Total Staff Hours per Line of Code are Below Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Influx of transported code	(0.25)
	Near build or milestone date	(0.25)
	Low complexity	(0.50)
	High productivity	(0.75)
	Lack of thorough testing	(0.25)
RULE-40	IF (Total Staff Hours per Line of Code are Below Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Influx of transported code	(0.25)
	Near build or milestone date	(0.25)
	Low complexity	(0.25)
	High productivity	(0.75)
	Lack of thorough testing	(0.25)

Figure 2-85. Rules for Below Normal Total Staff Hours per Line of Code

2.3.2.1.9 Above Normal Computer Hours per Computer Run

The SME considers five rules that address the case where the number of computer hours per computer run for a project is above normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-41	IF (Computer Hours per Computer Run are Above Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	System and integration testing started early	(0.50)
	Error prone code	(0.25)
	Computation bound algorithms run or tested	(0.50)
	Large reuse or early and larger test	(0.50)
RULE-42	IF (Computer Hours per Computer Run are Above Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	System and integration testing started early	(0.75)
	Error prone code	(0.25)
	Computation bound algorithms run or tested	(0.50)
	Large reuse or early and larger test	(0.50)
RULE-43	IF (Computer Hours per Computer Run are Above Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	System and integration testing started early	(0.50)
	Error prone code	(0.25)
	Computation bound algorithms run or tested	(0.75)
	Large reuse or early and larger test	(0.25)
RULE-44	IF (Computer Hours per Computer Run are Above Normal) and (Project is in system test phase)	
	THEN interpretations are	
	System and integration testing started early	(0.25)
	Error prone code	(0.25)
	Computation bound algorithms run or tested	(0.50)
	Large reuse or early and larger test	(0.25)
RULE-45	IF (Computer Hours per Computer Run are Above Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	System and integration testing started early	(0.25)
	Error prone code	(0.25)
	Computation bound algorithms run or tested	(0.50)
	Large reuse or early and larger test	(0.25)

Figure 2-86. Rules for Above Normal Computer Hours per Computer Run

2.3.2.1.10 Below Normal Computer Hours per Computer Run

The SME considers five rules that address the case where the number of computer hours per computer run for a project is below normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-46	IF (Computer Hours per Computer Run are Below Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Unit testing being done	(0.25)
	Easy errors or changes being found or fixed	(0.25)
	Simple system	(0.25)
	New or late development	(0.25)
RULE-47	IF (Computer Hours per Computer Run are Below Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Unit testing being done	(0.50)
	Easy errors or changes being found or fixed	(0.25)
	Simple system	(0.50)
	New or late development	(0.50)
RULE-48	IF (Computer Hours per Computer Run are Below Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Unit testing being done	(0.75)
	Easy errors or changes being found or fixed	(0.25)
	Simple system	(0.75)
	New or late development	(0.75)
RULE-49	IF (Computer Hours per Computer Run are Below Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Unit testing being done	(0.75)
	Easy errors or changes being found or fixed	(0.50)
	Simple system	(0.75)
	New or late development	(0.75)
RULE-50	IF (Computer Hours per Computer Run are Below Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Unit testing being done	(0.25)
	Easy errors or changes being found or fixed	(0.50)
	Simple system	(0.75)
	New or late development	(0.75)

Figure 2-87. Rules for Below Normal Computer Hours per Computer Run

2.3.2.1.11 Above Normal Reported Changes per Computer Run

The SME considers five rules that address the case where the number of reported changes per computer run for a project is above normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-51	IF (Reported Changes per Computer Run are Above Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	System and integration testing started early	(0.25)
	Error prone code	(0.50)
	Near build or milestone date	(0.50)
	Loose configuration management or unstructured development	(0.50)
	Unstable specifications	(0.25)
RULE-52	IF (Reported Changes per Computer Run are Above Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	System and integration testing started early	(0.50)
	Error prone code	(0.75)
	Near build or milestone date	(0.50)
	Loose configuration management or unstructured development	(0.75)
	Unstable specifications	(0.50)
RULE-53	IF (Reported Changes per Computer Run are Above Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	System and integration testing started early	(0.25)
	Error prone code	(0.75)
	Near build or milestone date	(0.50)
	Loose configuration management or unstructured development	(0.75)
	Unstable specifications	(0.50)
RULE-54	IF (Reported Changes per Computer Run are Above Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	System and integration testing started early	(0.25)
	Error prone code	(0.75)
	Near build or milestone date	(0.25)
	Loose configuration management or unstructured development	(0.75)
	Unstable specifications	(0.75)
RULE-55	IF (Reported Changes per Computer Run are Above Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	System and integration testing started early	(0.25)
	Error prone code	(0.75)
	Near build or milestone date	(0.25)
	Loose configuration management or unstructured development	(0.50)
	Unstable specifications	(0.75)

Figure 2-88. Rules for Above Normal Reported Changes per Computer Run

2.3.2.1.12 Below Normal Reported Changes per Computer Run

The SME considers five rules that address the case where the number of reported changes per computer run for a project is below normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-56	IF (Reported Changes per Computer Run are Below Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.50)
	A lot of testing	(0.50)
	Poor testing	(0.50)
	Change backlog or holding code	(0.50)
	Tight management plan or good configuration control	(0.50)
RULE-57	IF (Reported Changes per Computer Run are Below Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	A lot of testing	(0.50)
	Poor testing	(0.75)
	Change backlog or holding code	(0.50)
	Tight management plan or good configuration control	(0.75)
RULE-58	IF (Reported Changes per Computer Run are Below Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	A lot of testing	(0.50)
	Poor testing	(0.50)
	Change backlog or holding code	(0.50)
	Tight management plan or good configuration control	(0.75)
RULE-59	IF (Reported Changes per Computer Run are Below Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	A lot of testing	(0.25)
	Poor testing	(0.50)
	Change backlog or holding code	(0.25)
	Tight management plan or good configuration control	(0.75)
RULE-60	IF (Reported Changes per Computer Run are Below Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	A lot of testing	(0.25)
	Poor testing	(0.50)
	Change backlog or holding code	(0.25)
	Tight management plan or good configuration control	(0.50)

Figure 2-89. Rules for Below Normal Reported Changes per Computer Run

2.3.2.1.13 Above Normal Total Staff Hours per Computer Run

The SME considers five rules that address the case where the number of total staff hours per computer run for a project is above normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-61	IF (Total Staff Hours per Computer Run are Above Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	High complexity	(0.50)
	Modifications being made to recently transported code	(0.50)
	Changes hard to isolate	(0.25)
	Changes hard to make	(0.25)
	Late design	(0.75)
	Good solid and reliable code	(0.50)
	Unstable specifications	(0.25)
RULE-62	IF (Total Staff Hours per Computer Run are Above Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	High complexity	(0.75)
	Modifications being made to recently transported code	(0.25)
	Changes hard to isolate	(0.25)
	Changes hard to make	(0.25)
	Late design	(0.75)
	Good solid and reliable code	(0.75)
	Unstable specifications	(0.25)
RULE-63	IF (Total Staff Hours per Computer Run are Above Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	High complexity	(0.75)
	Modifications being made to recently transported code	(0.25)
	Changes hard to isolate	(0.50)
	Changes hard to make	(0.50)
	Late design	(0.25)
	Good solid and reliable code	(0.75)
	Unstable specifications	(0.50)
RULE-64	IF (Total Staff Hours per Computer Run are Above Normal) and (Project is in system test phase)	
	THEN interpretations are	
	High complexity	(0.25)
	Modifications being made to recently transported code	(0.25)
	Changes hard to isolate	(0.50)
	Changes hard to make	(0.50)
	Late design	(0.25)
	Good solid and reliable code	(0.25)
	Unstable specifications	(0.75)
RULE-65	IF (Total Staff Hours per Computer Run are Above Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	High complexity	(0.25)
	Modifications being made to recently transported code	(0.25)
	Changes hard to isolate	(0.25)
	Changes hard to make	(0.25)
	Late design	(0.25)
	Good solid and reliable code	(0.25)
	Unstable specifications	(0.75)

Figure 2-90. Rules for Above Normal Total Staff Hours per Computer Run

2.3.2.1.14 Below Normal Total Staff Hours per Computer Run

The SME considers five rules that address the case where the number of total staff hours per computer run for a project is below normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-66	IF (Total Staff Hours per Computer Run are Below Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Easy errors or changes being found or fixed	(0.25)
	Error prone code	(0.50)
	A lot of testing	(0.50)
	Lots of terminal jockeys	(0.50)
	Unstable specifications	(0.50)
RULE-67	IF (Total Staff Hours per Computer Run are Below Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Easy errors or changes being found or fixed	(0.25)
	Error prone code	(0.75)
	A lot of testing	(0.75)
	Lots of terminal jockeys	(0.75)
	Unstable specifications	(0.50)
RULE-68	IF (Total Staff Hours per Computer Run are Below Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Easy errors or changes being found or fixed	(0.25)
	Error prone code	(0.75)
	A lot of testing	(0.75)
	Lots of terminal jockeys	(0.75)
	Unstable specifications	(0.75)
RULE-69	IF (Total Staff Hours per Computer Run are Below Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Easy errors or changes being found or fixed	(0.25)
	Error prone code	(0.25)
	A lot of testing	(0.50)
	Lots of terminal jockeys	(0.25)
	Unstable specifications	(0.75)
RULE-70	IF (Total Staff Hours per Computer Run are Below Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Easy errors or changes being found or fixed	(0.25)
	Error prone code	(0.25)
	A lot of testing	(0.25)
	Lots of terminal jockeys	(0.25)
	Unstable specifications	(0.50)

Figure 2-91. Rules for Below Normal Total Staff Hours per Computer Run

2.3.2.1.15 Above Normal Computer Hours per Reported Change

The SME considers five rules that address the case where the number of computer hours per reported change for a project is above normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-71	IF (Computer Hours per Reported Change are Above Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.50)
	Poor testing	(0.25)
	High complexity	(0.25)
	Changes hard to isolate	(0.25)
	Unit testing being done	(0.25)
	Computation bound algorithms run or tested	(0.50)
RULE-72	IF (Computer Hours per Reported Change are Above Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	Poor testing	(0.25)
	High complexity	(0.50)
	Changes hard to isolate	(0.25)
	Unit testing being done	(0.25)
	Computation bound algorithms run or tested	(0.50)
RULE-73	IF (Computer Hours per Reported Change are Above Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	Poor testing	(0.25)
	High complexity	(0.75)
	Changes hard to isolate	(0.50)
	Unit testing being done	(0.25)
	Computation bound algorithms run or tested	(0.75)
RULE-74	IF (Computer Hours per Reported Change are Above Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	Poor testing	(0.25)
	High complexity	(0.75)
	Changes hard to isolate	(0.50)
	Unit testing being done	(0.25)
	Computation bound algorithms run or tested	(0.75)
RULE-75	IF (Computer Hours per Reported Change are Above Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.25)
	Poor testing	(0.25)
	High complexity	(0.25)
	Changes hard to isolate	(0.25)
	Unit testing being done	(0.25)
	Computation bound algorithms run or tested	(0.50)

Figure 2-92. Rules for Above Normal Computer Hours per Reported Change

2.3.2.1.16 Below Normal Computer Hours per Reported Change

The SME considers five rules that address the case where the number of computer hours per reported change for a project is below normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

```
RULE-76   IF (Computer Hours per Reported Change are Below Normal) and
             (Project is in early code & unit test phase)
          THEN interpretations are
             Near build or milestone date                          (0.25)
             Good testing or test plan                             (0.50)
             Error prone code                                      (0.25)
             Tight management plan or good configuration control   (0.50)

RULE-77   IF (Computer Hours per Reported Change are Below Normal) and
             (Project is in middle code & unit test phase)
          THEN interpretations are
             Near build or milestone date                          (0.25)
             Good testing or test plan                             (0.75)
             Error prone code                                      (0.50)
             Tight management plan or good configuration control   (0.75)

RULE-78   IF (Computer Hours per Reported Change are Below Normal) and
             (Project is in late code & unit test phase)
          THEN interpretations are
             Near build or milestone date                          (0.50)
             Good testing or test plan                             (0.75)
             Error prone code                                      (0.75)
             Tight management plan or good configuration control   (0.75)

RULE-79   IF (Computer Hours per Reported Change are Below Normal) and
             (Project is in system test phase)
          THEN interpretations are
             Near build or milestone date                          (0.50)
             Good testing or test plan                             (0.50)
             Error prone code                                      (0.50)
             Tight management plan or good configuration control   (0.50)

RULE-80   IF (Computer Hours per Reported Change are Below Normal) and
             (Project is in acceptance test phase)
          THEN interpretations are
             Near build or milestone date                          (0.25)
             Good testing or test plan                             (0.25)
             Error prone code                                      (0.25)
             Tight management plan or good configuration control   (0.25)
```

Figure 2-93. Rules for Below Normal Computer Hours per Reported Change

2.3.2.1.17 Above Normal Total Staff Hours per Reported Change

The SME considers five rules that address the case where the number of total staff hours per reported change for a project is above normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-81	IF (Total Staff Hours per Reported Change are Above Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.50)
	Poor testing	(0.50)
	Changes hard to isolate	(0.25)
	Changes hard to make	(0.25)
RULE-82	IF (Total Staff Hours per Reported Change are Above Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	Poor testing	(0.50)
	Changes hard to isolate	(0.50)
	Changes hard to make	(0.50)
RULE-83	IF (Total Staff Hours per Reported Change are Above Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.75)
	Poor testing	(0.75)
	Changes hard to isolate	(0.50)
	Changes hard to make	(0.50)
RULE-84	IF (Total Staff Hours per Reported Change are Above Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.50)
	Poor testing	(0.50)
	Changes hard to isolate	(0.75)
	Changes hard to make	(0.75)
RULE-85	IF (Total Staff Hours per Reported Change are Above Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Good solid and reliable code	(0.25)
	Poor testing	(0.25)
	Changes hard to isolate	(0.50)
	Changes hard to make	(0.50)

Figure 2-94. Rules for Above Normal Total Staff Hours per Reported Change

2.3.2.1.18 Below Normal Total Staff Hours per Reported Change

The SME considers five rules that address the case where the number of total staff hours per reported change for a project is below normal. Conditional evaluation of the rules depends upon the current life-cycle phase of the project and results in a set of possible interpretations for the observed deviation in the specified ratio of measures.

RULE-86	IF (Total Staff Hours per Reported Change are Below Normal) and (Project is in early code & unit test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	Near build or milestone date	(0.50)
	Easy errors or changes being found or fixed	(0.50)
	Modifications being made to recently transported code	(0.50)
	Error prone code	(0.50)
RULE-87	IF (Total Staff Hours per Reported Change are Below Normal) and (Project is in middle code & unit test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	Near build or milestone date	(0.50)
	Easy errors or changes being found or fixed	(0.75)
	Modifications being made to recently transported code	(0.25)
	Error prone code	(0.75)
RULE-88	IF (Total Staff Hours per Reported Change are Below Normal) and (Project is in late code & unit test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	Near build or milestone date	(0.50)
	Easy errors or changes being found or fixed	(0.75)
	Modifications being made to recently transported code	(0.25)
	Error prone code	(0.75)
RULE-89	IF (Total Staff Hours per Reported Change are Below Normal) and (Project is in system test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	Near build or milestone date	(0.25)
	Easy errors or changes being found or fixed	(0.50)
	Modifications being made to recently transported code	(0.25)
	Error prone code	(0.75)
RULE-90	IF (Total Staff Hours per Reported Change are Below Normal) and (Project is in acceptance test phase)	
	THEN interpretations are	
	Good testing or test plan	(0.25)
	Near build or milestone date	(0.25)
	Easy errors or changes being found or fixed	(0.25)
	Modifications being made to recently transported code	(0.25)
	Error prone code	(0.50)

Figure 2-95. Rules for Below Normal Total Staff Hours per Reported Change

2.3.2.2 General-Purpose Use of the Rule Base

The SME incorporates a set of general-purpose services commonly used with the rule base. The services are referenced by SME functions to provide needed services associated with the rule base. These services include

- *Determine Phase for Rules*
- *Determine Rate for Rules*
- *Evaluate Rule*

The following sections discuss each of these services and detail the algorithms behind the actions they perform.

2.3.2.2.1 Determine Phase for Rules

Purpose

Identifies the present life-cycle phase of the current project. The result is stored in a list as an assertion, consisting of an associated key and value pair, for subsequent use in evaluating rules.

Required Data

- Current date (input value)
- Schedule data
- Schedule model (in *Convert Date to Phase*)
- Assertion list

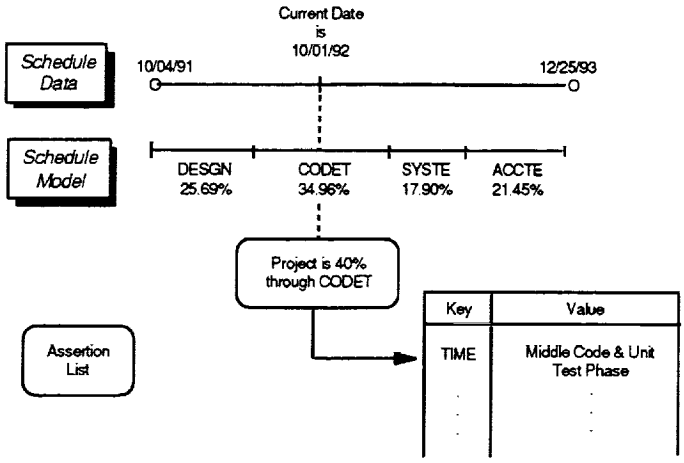
Steps

1. Use *Get Project Dates* to obtain the planned project start and end dates from the current schedule data.
2. On the basis of the project start and end dates, use *Convert Date to Phase* to translate the current date to the phase and elapsed fraction of phase that normally should be reached on that date. (*Phase Name*$_{Current}$ *and Fraction of Phase*$_{Current}$)
3. Set the assertion key and value to identify the present life-cycle phase as follows:

   Assertion Key = 'TIME'

   if (Phase Name$_{Current}$ *= 'DESGN')* *Assertion Value = 'Design phase'*
   if (Phase Name$_{Current}$ *= 'CODET') then*
   if (Fraction of Phase$_{Current}$ *<= 0.33)* *Assertion Value = 'Early code & unit test phase'*
   else if (Fraction of Phase$_{Current}$ *<= 0.66)* *Assertion Value = 'Middle code & unit test phase'*
   else if (Fraction of Phase$_{Current}$ *> 0.66)* *Assertion Value = 'Late code & unit test phase'*
   if (Phase Name$_{Current}$ *= 'SYSTE')* *Assertion Value = 'System test phase'*
   if (Phase Name$_{Current}$ *= 'ACCTE')* *Assertion Value = 'Acceptance test phase'*

4. Store the assertion in the rule base's assertion list.



STEPS

1. The project start and end dates are 10/4/91 and 12/25/93, respectively.

2. The current date of 10/01/92, based on the schedule model, maps to a point 40% into the code and test phase.

3. Since 40% through code and test falls into the middle third of the phase, set an assertion indicating that the present phase of the project is "Middle code & unit test phase."

Figure 2-96. Determining the Present Phase for the Rule Base

2.3.2.2.2 Determine Rate for Rules

Purpose

Compares the actual cumulative ratio of two specified measures to expected model values to determine if the rate is above, below, or within the range of values normally expected for the current date. The result is stored in a list as an assertion, consisting of an associated key and value pair, for subsequent use in evaluating rules.

Required Data

- Measure name for numerator (input value)
- Measure name for denominator (input value)
- Measure data (for the two specified measures)
- Schedule data
- Schedule model (in *Convert Date to Phase*)
- Measure model (for two specified measures) (in *Generate Rate Model*)
- Estimate set model (in *Get Ratio of Estimates*)
- Assertion list

Steps

1. Obtain the actual data values of the two specified measures as of the current date from the measure data. Set the actual value for the rate to the ratio of the two actual measure values as follows:

   $$\text{Actual Value} = \text{Actual Measure Value}_{\text{Numerator}} \;/\; \text{Actual Measure Value}_{\text{Denominator}}$$

2. Use *Get Project Dates* to obtain the planned project start and end dates from the current schedule data.

3. On the basis of the project start and end dates, use *Convert Date to Phase* to translate the current date of the measure data to the phase and elapsed fraction of phase that normally should be reached on that date.

4. Use *Generate Rate Model* to create a rate model for the two input measures from the corresponding measure models.

5. Use *Get Ratio of Estimates* to obtain the normal ratio of completion values for the two input measures from the estimate set model. (*Normal Completion Ratio*)

6. Use *Convert Phase to Measure* on the rate model to obtain the normal ratio of expected measure values for the two referenced measures at the desired phase and fraction of phase, given the normal ratio of completion values. ($\text{Expected Value}_{\text{Normal}}$)

7. Compute the upper and lower normal bounds on the expected ratio of values obtained by adding and subtracting, respectively, the scaled value of the normal deviation stored in the rate model to or from the expected ratio as follows:

   $$\text{Expected Value}_{\text{High}} = \text{Expected Value}_{\text{Normal}} + (\text{Normal Deviation} * \text{Normal Completion Ratio})$$

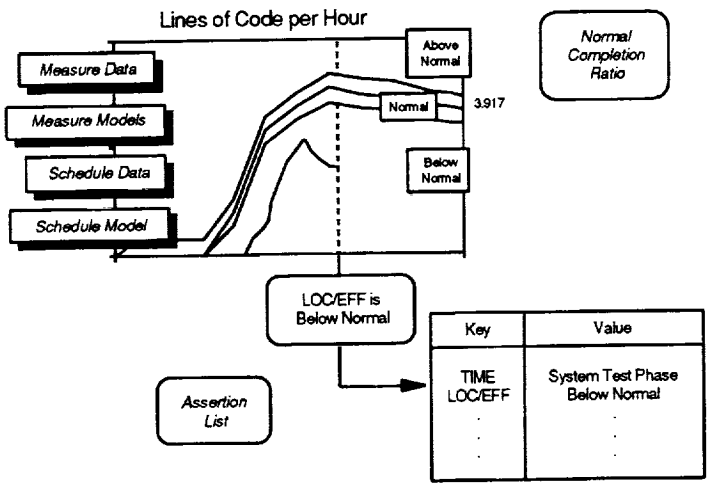
   $$\text{Expected Value}_{\text{Low}} = \text{Expected Value}_{\text{Normal}} - (\text{Normal Deviation} * \text{Normal Completion Ratio})$$

8. Set the assertion key and value to identify the results of evaluating the ratio of the specified measures as follows:

   Assertion Key = 'Measure Name$_{Numerator}$' + '/' + 'Measure Name$_{Denominator}$'

   if (Actual Value > Expected Value$_{High}$) Assertion Value = 'Above Normal'
   if (Actual Value < Expected Value$_{Low}$) Assertion Value = 'Below Normal'
   otherwise Assertion Value = 'Normal'

9. Store the assertion in the rule base's assertion list.



STEPS

1. Obtain the current phase using the schedule model.

2. Get the normal ratio of completion values for the two input measures from the estimate set model (e.g., LOC to EFF is 3.917).

3. Generate a rate model for the two measures.

4. Scaling the rate model for the normal completion ratio, get the expected value and normal range of the rate for the current phase.

5. If the actual measure ratio falls under the normal range, as shown, add an assertion that the rate is below normal.

Figure 2-97. Determining Measure Rates for the Rule Base

2.3.2.2.3 Evaluate Rule

Purpose

Evaluates a single rule in the rule base to conditionally determine the applicability of the rule's interpretations to the current project. If the rule's condition evaluates to true, each associated interpretation is stored for subsequent use in a list as an assertion, consisting of an associated key and value pair. If the rule's condition evaluates to false, no action is taken.

Required Data

- Rule (input value)
- Assertion list

Steps

1. For each expression in the rule's condition, capture the expression as an assertion consisting of an associated key and value pair. Locate the entry *k* with a matching key in the assertion list and evaluate the individual expressions, for *i* equals *1* through *N*, as follows:

   if (Expression Value[i] = Assertion Value[k]) Expression[i] = TRUE
   otherwise Expression[i] = FALSE

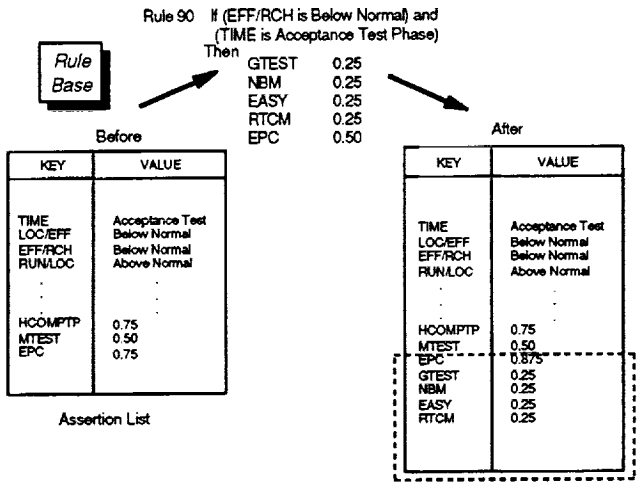
2. Evaluate the rule's condition, using boolean logic and the results obtained from evaluating the individual expressions.

3. If the rule's condition is true, express each of the rule's interpretations as assertions and determine if that interpretation already exists in the assertion list by searching for a matching key.

4. For each interpretation associated with a rule whose condition is true, store the interpretation as an assertion as follows. If the interpretation does not already exist in the list, simply add the interpretation's key and value to the assertion list. If the interpretation exists in the list as the kth entry, update the assertion value to reflect a new weighted certainty for the interpretation using

   *Assertion Value[k] = (1 - New Certainty) * Old Certainty + New Certainty*

   where *New Certainty* is the weighted interpretation's certainty used as an *Assertion Value* and *Old Certainty* is the original certainty existing in the list for that entry as *Assertion Value[k]*



STEPS

1. Each expression in the rule's condition is located in the assertion list and evaluated.

2. The rule's condition is evaluated using boolean logic and the evaluation of the individual expressions.

3. If the rule is true, each of the rule's interpretations is expressed as an assertion and searched for in the assertion list.

4. If that assertion is already in the list, the assertion's associated weight is updated. Otherwise, it is added to the assertion list.

Figure 2-98. Evaluating a Rule in the Rule Base

2.4 MANAGEMENT DATA

At times, the SME must rely on the manager using the system for additional information needed to perform a given function. This situation arises in three specific instances where the SME permits the user to interactively specify the required data. The first case involves permitting the manager to modify the project's plan by changing the current schedule and estimates for use in analyzing what-if scenarios. The second case lets a manager specify an estimate of the current phase (presumably based on information that is unavailable to the SME) for use in making predictions. Lastly, the third case solicits subjective information about a project from the manager to augment the known objective data contained in the knowledge base on the project. In each of these cases, the SME can store the information for future reference.

Table 2-5 summarizes the major components referenced by the SME as management data. Each component maps to a particular type of information obtained from SME users and is identified with a specific purpose.

Table 2-5. SME Management Data Components

COMPONENT	PURPOSE
Alternative Plans	Identifies sets of project schedules and estimates supplied by the user to support what-if scenarios
Phase Estimates	Identifies where a project is in the development life cycle on a given date as a basis for making predictions
Subjective Data	Identifies the manager's ratings of a set of subjective factors that supply additional knowledge about a project

The following sections provide additional detailed information on each of these components.

2.4.1 Alternative Plans

Purpose

Enables the user to investigate the effects of changing project plans.

Description

Alternative plans are created by the manager interactively during sessions with the SME planning function. An alternative plan consists of a schedule and a set of completion estimates that have been modified by the user in some way. The schedule has the same format as the project's current schedule, but the user might have modified one or all of the development life-cycle phase dates. Similarly, the set of completion estimates has the same format as the current completion estimates, but the user might have modified one or all of the estimated completion values. The SME provides the user with two distinct methods for creating an alternative plan, as detailed in Section 3 under planning services. Once created, the plan may be used in subsequent monitoring and overall assessment functions to investigate the effects of modifying scheduled phase dates or completion estimates.

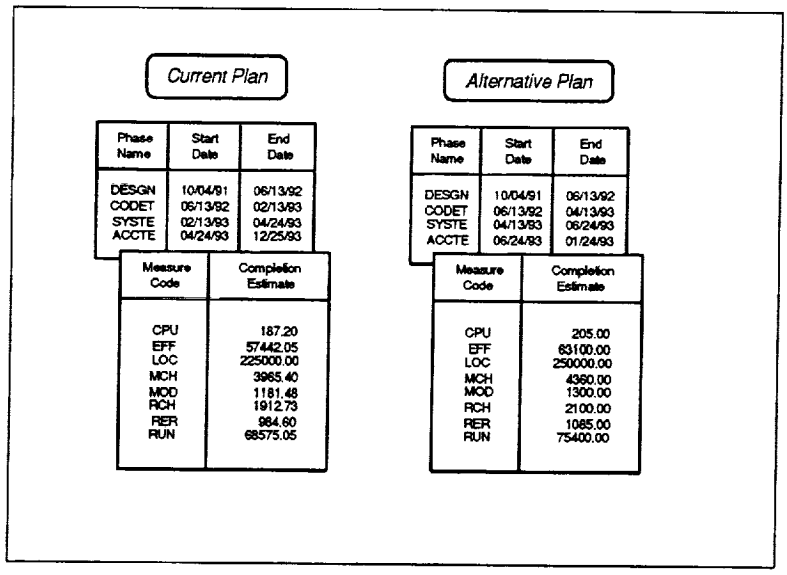


Figure 2-99. Representative Alternative Plan for a Project

Source

Created interactively by the manager using one of two methods. May be input interactively by the user, or derived from the schedule and estimate set models.

Assumptions

- The alternative schedule and estimate set will conform to the standard SME formats for schedules and estimates
- A change to either a project's schedule or its estimates constitutes an alternative plan

Instances

Multiple alternative plans can exist for any given project. Each alternative plan belongs to the manager who originally created the plan.

Structure

A schedule table with three columns—phase name, phase start date, and phase end date; an estimates table with two columns—measure code and estimated completion value.

2.4.2 Phase Estimates

Purpose

Indicates where a project is in its life cycle on a given date.

Description

Phase estimates are created by the manager interactively during sessions with the SME prediction function. A phase estimate reflects an assessment of exactly where a project is in the development life cycle on a specific date. A phase estimate contains the current date, the current phase, and the completed fraction of that phase. The SME uses a phase estimate for a given project as the basis for predicting the expected completion date and value of the measure of interest for the current project. The SME provides three distinct methods of obtaining a phase estimate, as detailed in Section 3 under the prediction function. The user may select which method should be used for the prediction.

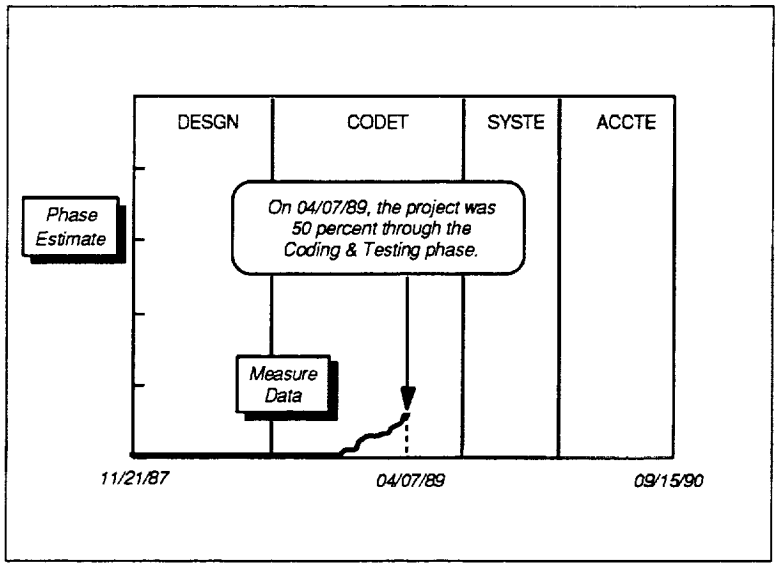


Figure 2-100. Representative Phase Estimate for a Project

Source

Selected interactively by the manager from three choices. The phase estimate may be calculated by the SME, derived from the current schedule, or input interactively by the user.

Assumptions

- The date specified in the phase estimate must be between the project start date and the current project date

Instances

Multiple phase estimates can exist for any given project. Each phase estimate belongs to the manager who originally created and used the estimate as the basis for a prediction.

Structure

Table with three columns—date, phase name, and fraction of phase complete.

2.4.3 Subjective Data

Purpose

Represents the manager's ratings of software development factors for a given project.

Description

Subjective data currently is collected from the manager interactively during sessions with the SME trend analysis function. The data consists of ratings associated with specific factors that potentially affect the software development process, such as development team experience, problem complexity, and tool usage. The SME uses these ratings in the expert system software.

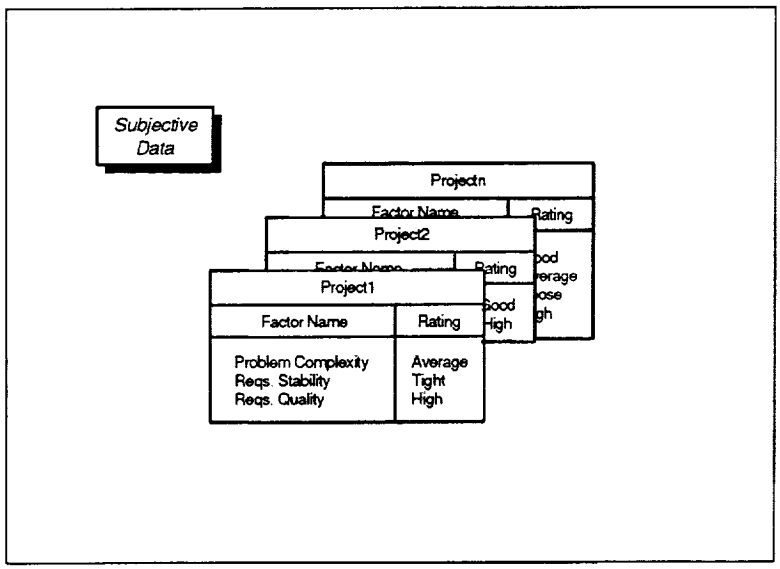


Figure 2-101. Subjective Data for Three Projects

Source

Collected interactively from the manager

Assumptions

- Each project's manager is responsible for providing subjective data on their own project that can be referenced by all SME users

Instances

One set of subjective data may exist for each project.

Structure

Table with two columns—factor name and rating. Each row in the table describes one subjective factor that exists in the knowledge base. The rating is a value that translates to either high, normal, low, or unknown.

SECTION 3—FUNCTIONALITY

The SME supports a key set of experience-based functions intended to assist software development managers in actively tracking and evaluating the status of their projects. These functions rely on the components described in the previous section for information on an ongoing project as well as for the collective experience from past development efforts that can be used to understand and manage the project. When organized by the type of service they provide for the user, these functions fall into four categories. The first relates to executive services. These functions include general high-level features that permit a user to choose a project to examine and optionally to go back in time to an earlier point in that project's life cycle. In short, these services establish the scope and context in which all subsequent SME functions will be performed. The second encompasses various monitoring services that focus on a specific measure selected by the user. These functions permit a user to observe, compare, predict, and analyze the behavior of the measure of interest. The third covers assessment services pertaining to the overall quality of the project. These functions allow a user to objectively evaluate and examine high-level quality attributes, such as correctability and maintainability, with respect to a normal project in the environment. The fourth category contains planning services that support the creation and use of alternative schedules and estimates. These functions are used for performing "what if" scenarios to explore the effects of changing a project's plan.

Table 3-1 summarizes the major functions provided by the SME, organized into four basic service categories.

Table 3-1. Major Functions Provided by the SME

SERVICE	FUNCTION
Executive	Project Selection Specification of Current Project Date
Monitoring	Measure Selection Simple Observation Comparison to a Normal Project Comparison to Manager's Plan Comparison to Other Projects Prediction Trend Analysis Profile Analysis
Overall Assessment	Attribute Evaluation Attribute Factor Examination
Planning	Use of Alternative Schedules Use of Alternative Estimates

3.1 EXECUTIVE

The executive services provided by the SME serve to establish the context in which all subsequent functions will be performed. Primarily, this involves permitting the manager to identify a project to examine by choosing one from a list of all available projects. Once the manager specifies a project of interest, any SME functions requested will reference that project.

To the user, selecting the project of interest is a simple case of choosing the name of the desired project from a list. This action, however, causes the SME to initialize a contextual environment for performing SME functions that incorporates a wide range of information related to the project of interest. This initialization includes locating and obtaining all data captured for the project, choosing the manager's current plan from the list of all submitted schedules and estimates based on the current project date, and identifying an appropriate set of models to use with the project given its known characteristics. The manager may switch to a different project at any time by choosing a new project of interest.

A second key service permits the manager to change the current date of the project of interest to view the project as it appeared at some earlier time. By default, when a project is first selected as the project of interest, the current project date is set to the latest date for which measure data exists. This lets the manager obtain the latest picture of the project from the most current information available. At times, however, the manager may wish to view the project as it appeared at an earlier point in the software development life cycle. To accommodate this, the SME allows the manager to override the default current value to effect going back in time to an earlier project date. Specifying a different project date causes the SME to update the current plan to reflect the manager's schedule and estimates in effect on that date. All subsequent SME functions requested by the manager reference that plan and artificially truncate the project's measure and profile data. The resultant picture of the project reflects what the SME would have shown on the specified date.

Table 3-2 summarizes the major functions supported by the SME under executive services.

Table 3-2. Key Executive Services Functions

FUNCTION	PURPOSE
Project Selection	Lets user select a project as the current project of interest for performing all subsequent SME functions
Specification of Current Project Date	Lets user change the current date of the project of interest to view the project as it appeared at some earlier time

The following sections provide additional detailed information on each of these functions.

3.1.1 Project Selection

Purpose

Lets the user select a single project as the project of interest for any subsequent SME functions.

Description

The project selection function displays a list of all available projects and permits the user to choose a project of interest. The SME performs its functions within the context of this particular project. Selecting a project of interest causes the SME to identify and locate for future reference all data captured for the project, the manager's current plan submitted for the project, and the appropriate models to apply to the project.

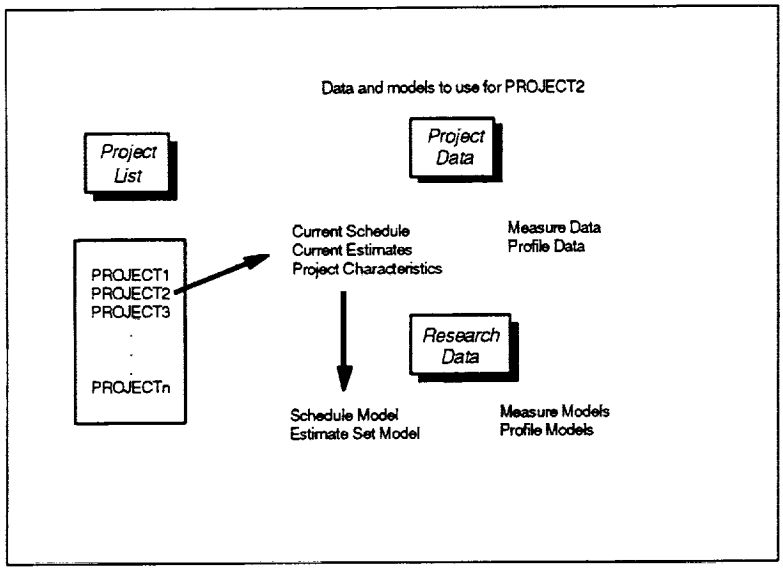


Figure 3-1. Selecting a Project of Interest

The figure illustrates the selection of a project of interest from the list of available projects. This example shows that if a user chooses PROJECT2 from the list, the SME identifies the project data associated with that project and references an appropriate set of models that match the key characteristics of the project.

Note that research data such as attribute definitions and management rules can apply to any project regardless of the project's characteristics.

Required Information

- List of available projects (project list)
- List of defined measures (measure list)
- List of defined profiles (profile list)
- List of available measures for projects (project/measure availability list)
- List of available profiles for projects (project/profile availability list)
- Planned schedule for the project (schedule data)
- Planned completion values for measures (estimates data)
- Actual data values for the available measures (measure data)
- Actual data values for the available profiles (profile data)

- Key characteristics of the project (project characteristics)
- Model of the schedule for similar projects (schedule model)
- Model of completion estimates for similar projects (estimate set model)
- Models of measure behavior for similar projects (measure models)
- Models of profile behavior for similar projects (profile models)

Key Steps

1. Select a project of interest and locate all available project data for the project.
2. Set the current plan to reflect the most recent schedule and estimates for the project.
3. Identify a set of appropriate models to use with the selected project.

3.1.1.1 Select a Project of Interest

Purpose

Allows the user to select a project from the list of all available projects. Identifies and locates all project data for the selected project.

Required Data

- Project list
- Project/measure availability list
- Project/profile availability list
- Schedule data (for project of interest)
- Estimates data (for project of interest)
- Measure data (for project of interest)
- Profile data (for project of interest)
- Project characteristics (for project of interest)

Steps

1. Display the list of available projects appearing in the project list and permit the user to select a project of interest.
2. Reference the project/measure availability list to identify the measures with data for the project. Locate the data for each available measure.
3. For each available measure, reference the project/profile availability list to identify the measure profiles with data for the project. Locate the profile data for each available profile.
4. Locate the schedule data, estimates data, and project characteristics for the project.

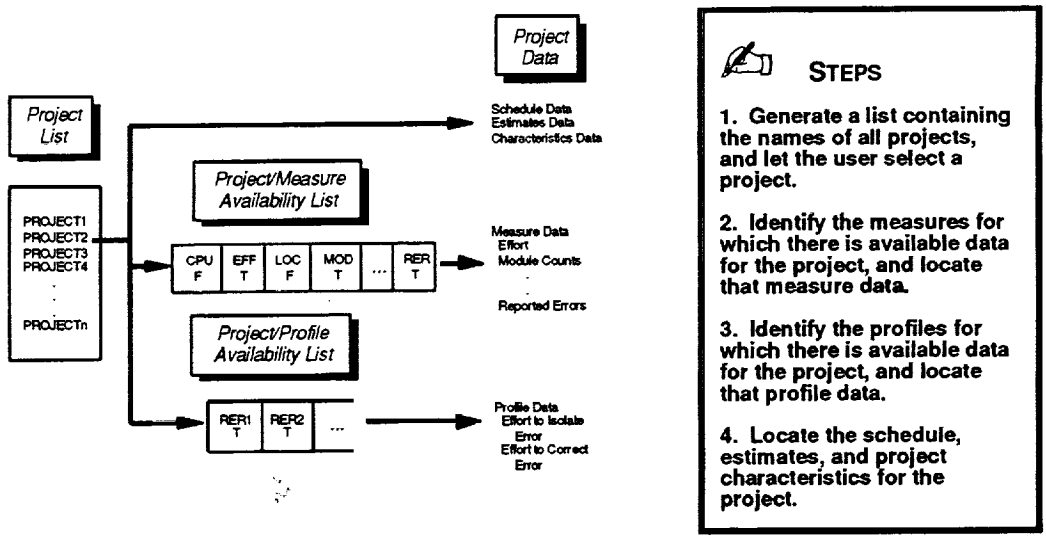


Figure 3-2. Identifying Project Data for the Project

3.1.1.2 Set Current Plan for Project

Purpose

Examines all schedules and estimates submitted by the manager for the project of interest over the development life cycle to obtain the ones that were in effect on a specified date. The identified schedule and set of completion estimates become, respectively, the current schedule and the current estimates for the project. When considered together, the selected schedule and estimates constitute the current plan submitted by the manager.

Note: When a project of interest is first selected, this service causes the most recent plan submitted by the manager to be chosen as a default. If the user subsequently changes the current project date to effect going back in time to an earlier date, that date is specified to choose a "current" plan from the past.

Required Data

- Current project date (input value)
- Schedule data (for project of interest)
- Estimates data (for project of interest)

Steps

1. Use *Get Schedule* with the input project date to obtain the most recent schedule submitted on or prior to the specified date.
2. Use *Get Estimates* with the input project date to obtain the most recent set of completion estimates submitted on or prior to the specified date.
3. Remember the resultant schedule and set of estimates, respectively, as the current schedule and current estimates for the project of interest.

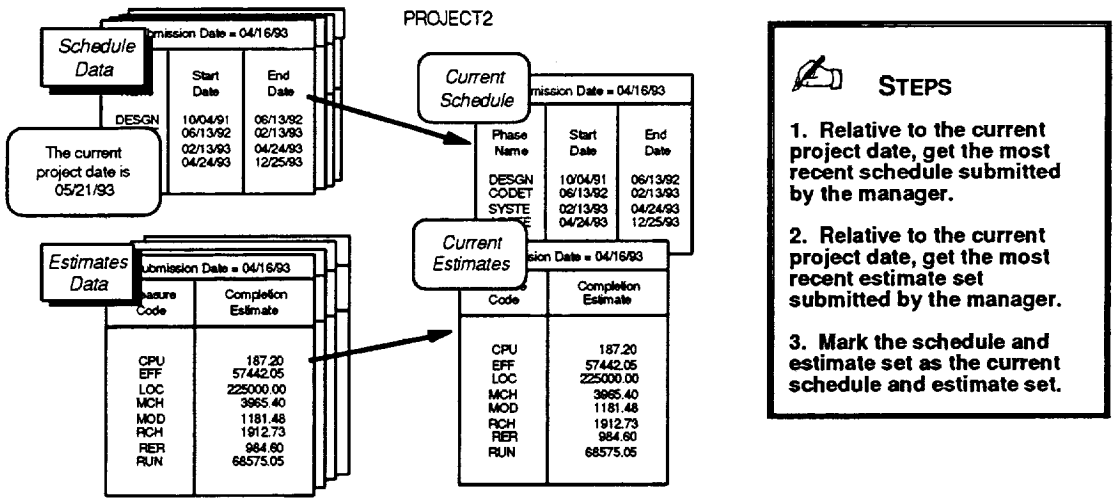


Figure 3-3. Setting the Current Plan for a Project

3.1.1.3 Identify Models to Use for Project

Purpose

Identifies and locates an appropriate set of models to use with the project of interest.

Required Data

- Project characteristics (for project of interest)
- Measure list
- Profile list
- Schedule model (suitable for the type of project)
- Measure models (suitable for the type of project)
- Profile models (suitable for the type of project)
- Estimate set model (suitable for the type of project)

Steps

1. Obtain the characteristics of the selected project of interest from its project characteristics data.
2. Concatenate the characteristics to produce a project type that identifies the appropriate models to use for the project.
3. Identify and locate the schedule model and the estimate set model that match the project type of the project of interest. (Use default models if no match exists.)
4. For each measure defined in the measure list and each profile defined in the profile list, identify and locate the measure and profile models that match the project type of the project of interest. (Use default models if no match exists.)

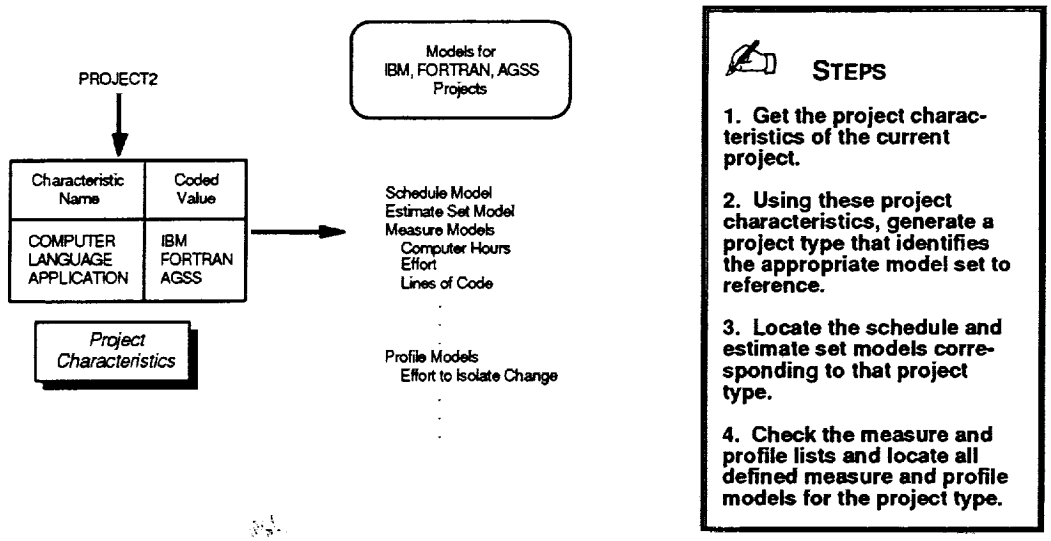


Figure 3-4. Identifying Models for the Project of Interest

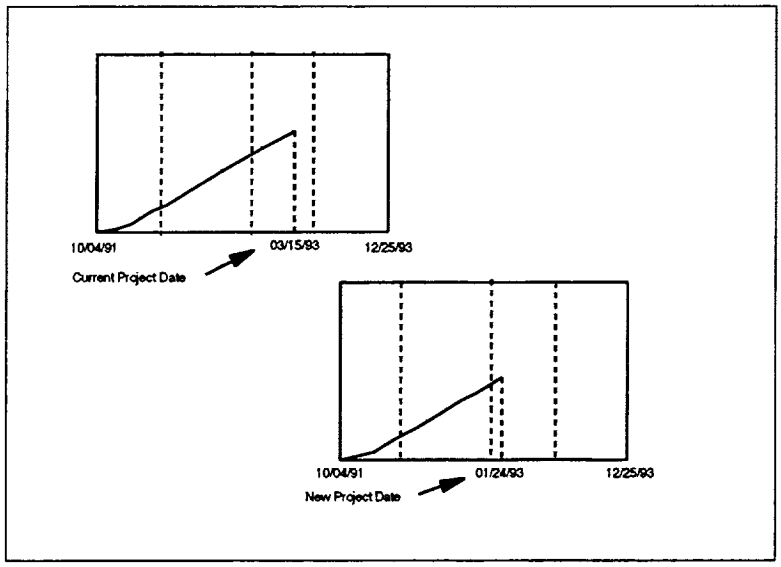
3.1.2 Specification of Current Project Date

Purpose

Lets the user change the current date of the project of interest to view the project as it appeared at some earlier time.

Description

The function permits the user to change the current project date to effect going back in time to an earlier point in the development life cycle. The user-specified date must fall between the project start date and the last date for which measure data exists (i.e., the original project date considered current). Changing the current date of the project of interest causes the SME to update the current plan to reflect the manager's schedule and completion estimates that were in effect on the specified date. Until the date is reset or the project is changed, all subsequent SME functions requested by the user for the project will reference the adjusted current date to artificially truncate any measure or profile data.



The figure illustrates changing the current date of the project of interest to reflect an earlier point in time.

Note that since an historical record of the subjective ratings used with the knowledge base for a project are not maintained over time, any updates made to these ratings can not be restored to reflect a change in the current project date.

Figure 3-5. Changing the Current Date for a Project

Required Information

- Last date for which measure data exists (current project date)
- All planned schedules for the project (schedule data)
- All planned completion values for measures (estimates data)

Key Steps

1. Obtain, validate, and remember the new date requested by the user.
2. Use *Set Current Plan for Project* (see project selection) to update the current schedule and current estimates to reflect the plan in effect on that date.

3.2 MONITORING

The monitoring services provided by the SME focus on a specific measure of interest chosen by the manager for the current project. This measure of interest may be an individual measure selected from the list of defined measures for which data exists or it may be the ratio of any two of those measures. Once the manager specifies the measure (or ratio of measures) to examine, any SME monitor functions requested will reference that measure. The manager may switch to a different measure at any time by choosing a new measure of interest.

At a basic level, the SME supports observation of the selected measure of interest by plotting its collected values as a function of time over the manager's schedule. While useful in tracking the actual work accomplished to date, this feature gives no indication of whether the project is on schedule or what work should have been accomplished. To provide such a yardstick for monitoring progress, the SME incorporates three methods of graphically comparing the observed measure values to the likely behavior of the measure based on past experience in the environment. These methods are comparison to normal project guidelines derived from models of the measure's past behavior on similar projects, comparison to a model of the measure adjusted to fit the manager's current plan, and comparison to actual measure values observed on one or more past projects.

The SME also allows the manager to predict the future behavior of the measure of interest over the project life cycle. This prediction is performed by fitting models of normal project behavior to the actual data collected on the project, thereby forecasting the probable completion date and expected completion value of the measure.

Additional monitoring services help managers identify a project's strengths and weaknesses by analyzing the current value of the measure of interest. The SME supports this monitoring function through trend analysis and profile analysis.

Trend analysis compares the current value of the selected measure to a model of the measure and uses expert systems techniques to reach conclusions that explain any deviations from the norm. This analysis uses two discrete approaches for interpreting the captured management experience and providing expert assistance to the manager. If the measure of interest is a single defined measure, the analysis relies on the knowledge base for the necessary management rules; if the measure of interest is a ratio of two measures, the analysis uses the rule base. In either case, the function examines not only the measure of interest, but a wide range of current data for the project, to reach its conclusions.

Profile analysis, on the other hand, lets managers examine and interpret the current value of the measure of interest in more detail to detect potential problems and identify improvement areas. This function displays a detailed distribution of the current measure value broken down into discrete, defined categories. Multiple profiles, or ways of categorizing the data, may be defined for each measure.

Table 3-3 summarizes the major functions supported by the SME under monitoring services.

Table 3-3. Monitoring Services Functions

FUNCTION	PURPOSE
Measure Selection	Lets the user select an available measure as the current measure of interest for performing SME monitor functions
Simple Observation	Displays the actual values observed for a measure of interest as a function of calendar time
Comparison to a Normal Project	Compares the actual values observed for a measure of interest to a model of the measure's normal behavior
Comparison to Manager's Plan	Compares the actual values observed for a measure of interest to its expected behavior given the manager's plan
Comparison to Other Projects	Compares the actual values observed for a measure of interest to the measure's behavior on other projects
Prediction	Forecasts the probable completion date and the expected completion value of the measure of interest
Trend Analysis	Displays a list of possible reasons to explain an observed deviation in the measure of interest
Profile Analysis	Displays a distribution of actual measure values within two or more discrete categories for detailed user examination

The SME provides the full range of monitoring services whenever the user selects a single measure as the measure of interest. When the user chooses to monitor a ratio of two measures, however, the SME limits the available monitoring services to observation, comparison, and trend analysis functions. This limitation arises because (1) the concept of profile analysis inherently applies only to individual measures and (2) the algorithms used in prediction currently do not accommodate the non-monotonically increasing behavior exhibited by ratios of measures.

The following sections provide additional detailed information on each of these functions.

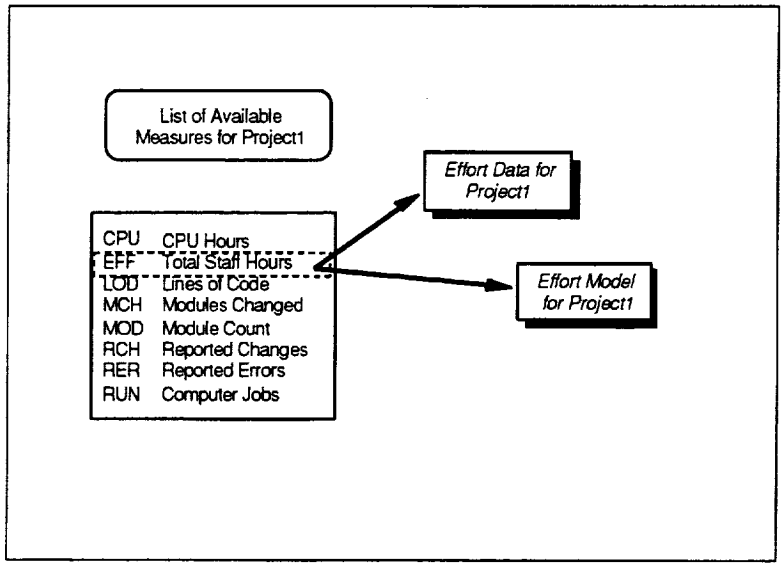
3.2.1 Measure Selection

Purpose

Lets the user select a single measure, or a ratio of two measures, as the measure of interest for any subsequent SME monitor functions.

Description

The measure selection function displays a list of all available measures and permits the user to choose a measure of interest for the current project. This measure of interest may be either a single measure for which data exists or a ratio of two such measures from the list. The SME performs all monitor functions for the current project within the context of this measure of interest. Selecting a single measure as the measure of interest simply causes the SME to identify the appropriate measure data and measure model to use in the future. Selecting two measures to serve as a ratio for the measure of interest, however, causes the SME to construct a set of measure data and a measure model for future reference that reflects the ratio of the identified measures. In this case, the SME creates the needed set of measure data from the cumulative ratios of the values recorded for the individual measures. Similarly, the SME generates a measure model to use with the ratio by combining the two models that correspond to the selected measures.



The figure depicts the selection of a measure of interest from the list of available measures. This example shows that if a user chooses "Total Staff Hours" from the list, the SME identifies the corresponding effort data and appropriate effort model for use in subsequent monitor functions.

Note that choosing two measures results in the SME combining the data and models of the individual measures to generate a composite set of data and a model for use with the ratio.

Figure 3-6. Selecting a Measure of Interest

Required Information

- List of defined measures (measure list)
- List of available measures for the current project (project/measure availability list)
- Actual data values of measures (for ratios) (measure data)
- Models of measure behavior (for ratios) (measure models)

Key Steps

1. Display a list of available measures for the current project and allow the user to select a measure of interest.
2. If the user selects an individual measure, identify the corresponding measure data and measure model to use.
3. If the user chooses two measures, construct the measure data for the ratio and use *Generate Rate Model* on the two measure models to create an appropriate model.

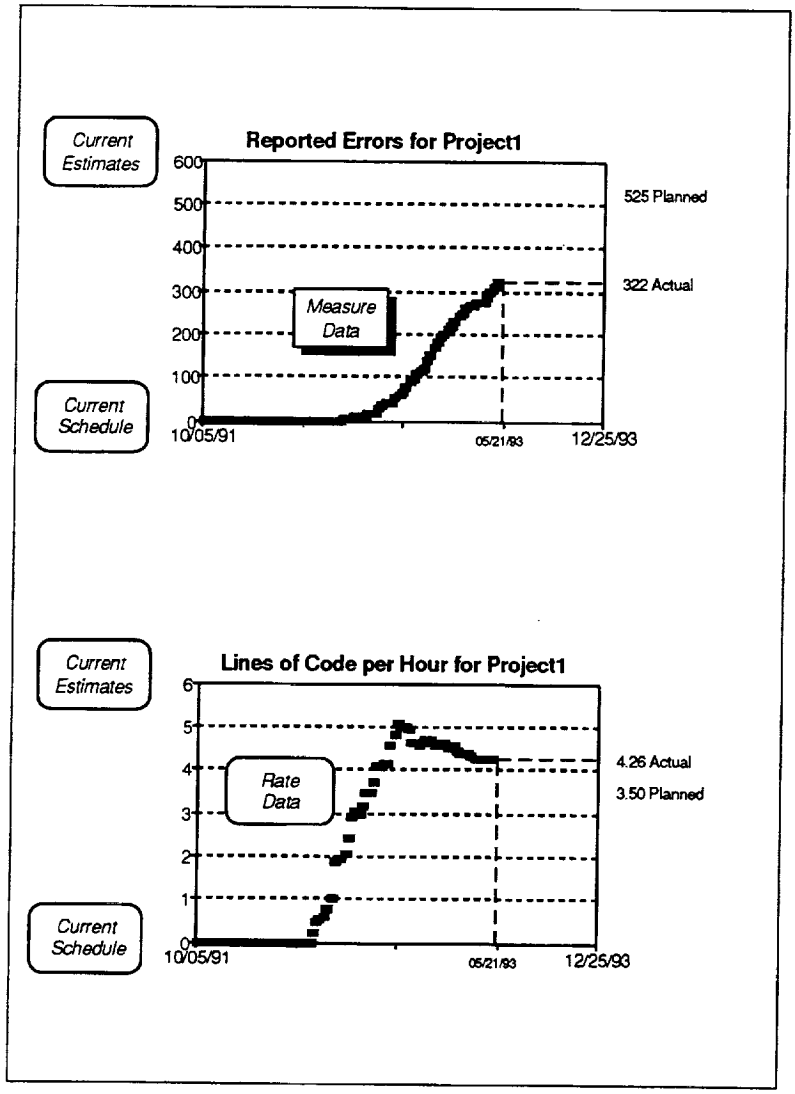
3.2.2 Simple Observation

Purpose

Displays the actual cumulative values of a measure of interest, such as effort or lines of code, as a function of calendar time.

Description

The observation function displays the actual recorded behavior of the measure of interest for the current project. The results are depicted graphically as a plot of the current measure with actual data values shown from project start through the current date. The manager's estimated completion value also appears for reference as a targeted planning value.



Note that observation applies to a ratio of two measures, as well as to a single, individual measure. This lets managers view an extended set of measures such as LOC per hour (coding productivity) and reported errors per LOC (error density).

The upper plot in the figure shows a representative observation plot of reported errors for a sample project. This example indicates that the project has reported a total of 322 errors through the current date of 05/21/93. The manager expects to see a total of 525 errors at the end of the project.

The lower plot in the figure shows a representative plot of the ratio of two measures, LOC to effort, for a sample project. This plot indicates that the project has produced 4.26 lines of code per hour through the current date. The manager plans to generate 3.50 lines of code per hour over the entire project.

Figure 3-7. Observing Actual Measure Values

Required Information

- Project start and end dates (current schedule)
- Actual data values for the measure of interest (measure data)
- Estimated completion value of the measure of interest (current estimates)

Key Steps

1. Scale and display the basic plotting area to use for observation.
2. Plot the actual measure values and the manager's planned completion value.

3.2.2.1 Scale and Display Plot Area for Observation

Purpose

Scales the plotting area to use for observation and generates the plot axes, labels, and title.

Required Data

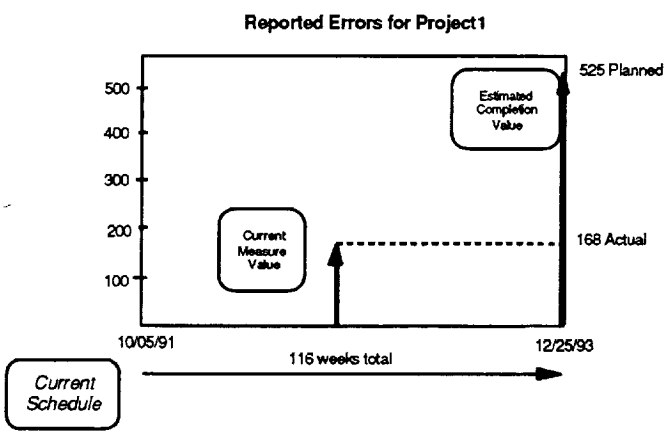
- Current schedule
- Current estimates
- Measure data (for measure of interest)

Steps

1. Use *Get Project Dates* with the current schedule to obtain the project start and end date. Calculate the number of weeks planned between these dates ($Planned\ Weeks_{Total}$).
2. Scale the plot's x-axis to the number of weeks in the project's planned schedule

   $X\text{-}Axis\ Scale = Planned\ Weeks_{Total}$
3. Use *Get Estimated Completion Value* with the current estimates to obtain the manager's planned completion value for the measure ($Planned\ Value_{Completion}$).
4. Scale the plot's y-axis to the maximum of either the manager's planned completion value or the current measure value found in the measure data.

   $Y\text{-}Axis\ Scale = Maximum\ (Planned\ Value_{Completion}, Actual\ Value_{Current})$
5. Display the basic plotting area with appropriate axes, labels, and title.



STEPS

1. The x-axis is scaled to the project duration of 116 weeks.

2. Given an estimate of 525 errors in the current estimate set, the manager's planned completion value is set to 525.

3. Based on an actual value to date of 168 errors and a planned completion value of 525, the y-axis is scaled to show 525 errors (i.e., the maximum value).

4. After mapping the fixed-sized plotting area to the computed x and y scaling factors, the basic plot is displayed.

Figure 3-8. Scaling the Observation Plotting Area

3.2.2.2 Plot Actual Data for a Measure

Purpose

Plots the actual data values of the measure of interest from project start through the current date. Adds a label to the plot for the actual measure value to date.

Required Data

- Current schedule
- Measure data (for measure of interest)

Steps

1. Initialize the starting point for plotting the actual measure data as a function of week number, to indicate the measure value is zero at week number zero using

   X-Value[0] = 0 and *Y-Value[0] = 0*

2. For each entry in the measure data through the current date, set the x and y values of the next point to plot to the week number of the sample date and its corresponding measure value as follows:

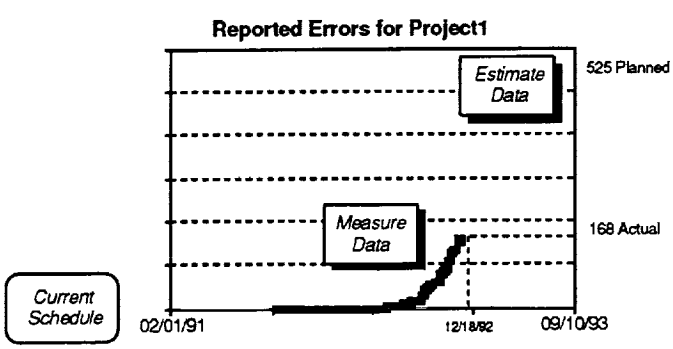
   X-Value[i] = Week(i) and *Y-Value[i] = Measure Value[i]*

   for the ith entry in the measure data, where Week(i) is the relative week number of the ith entry

3. Plot the data points computed for the actual measure data by week number, from 0 through the current week N, as a step function (i.e., plot the rise and then the run)

4. Label the x-axis with the project start and end dates from the current schedule.

5. Label the actual measure value observed on the current date at its correct height on the right side of the plotting area. ($Actual\ Value_{Current}$)



STEPS

1. The x- and y-values serve as week numbers and actual data values, respectively, for the measure data. They are initialized to zero.

2. The x-values are set to successive week numbers, while the y-values are set to the measure values for each week.

3. The points are plotted as a step function (i.e., rise and then run).

4. The actual measure value to date is labeled.

Figure 3-9. Plotting Actual Values for a Measure

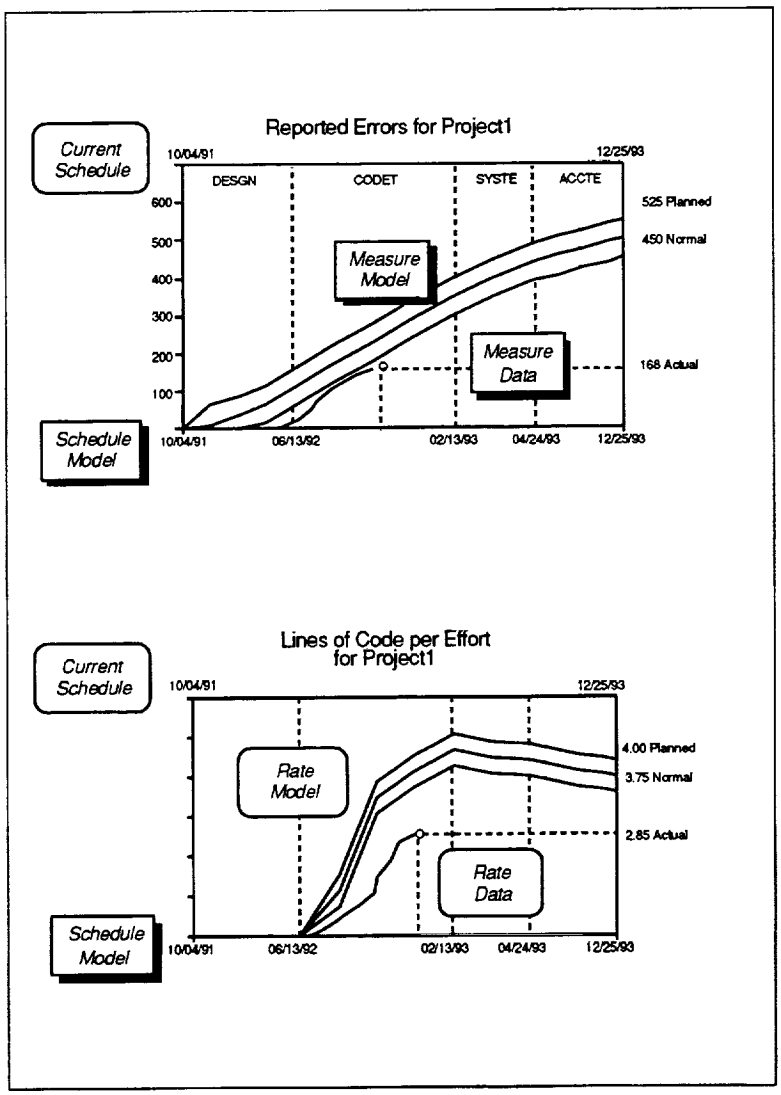
3.2.3 Comparison to a Normal Project

Purpose

Compares the actual cumulative values of a measure of interest for the current project to guidelines derived from models of the measure's normal behavior.

Description

The comparison function can visually contrast the actual recorded behavior of a measure of interest for the current project with guidelines of the measure's expected behavior for a normal project. The comparison is depicted graphically by "superimposing" a reference plot representing a normal project on an observational plot containing actual measure values.



Derived from models, the guidelines on the plot show the normal range of expected measure values as a function of a normal schedule. The normal values are scaled to reflect the size and duration of the current project. Note that comparison applies to a ratio of two measures, as well as to a single measure.

The upper plot in the figure shows a representative comparison plot of reported errors for a sample project. This example indicates (1) the project's 168 reported errors are below what is normally expected for the current date and (2) typical projects of the same size normally have 450 errors at project completion.

The lower plot in the figure compares the ratio of two measures to normal. This plot shows (1) the project's coding productivity of 2.85 LOC per hour is below normal and (2) projects of the same size normally produce 3.75 LOC per hour overall.

Figure 3-10. Comparing a Measure to Normal Guidelines

Required Information

- Project start and end dates (current schedule)
- Actual data values for the measure of interest (measure data)
- Estimated completion value of the measure of interest (current estimates)
- Model of the schedule for similar projects (schedule model)
- Model of the measure of interest for similar projects (measure model)
- Model of completion estimates for similar projects (estimate set model)

Key Steps

1. Scale and display the basic plotting area to use for comparisons with normal projects.
2. Plot the normal measure guidelines and schedule to expect for a similar project.
3. Use *Plot Actual Data for a Measure*, as in simple observation, to overlay the actual measure values and the manager's planned completion value on the plot.

3.2.3.1 Scale and Display Plot Area for Comparison to Normal

Purpose

Scales the plotting area to use for comparing a measure to normal project behavior and generates the plot axes, labels, and title.

Required Data

- Current schedule
- Current estimates
- Measure data (for measure of interest)
- Measure model
- Estimate set model

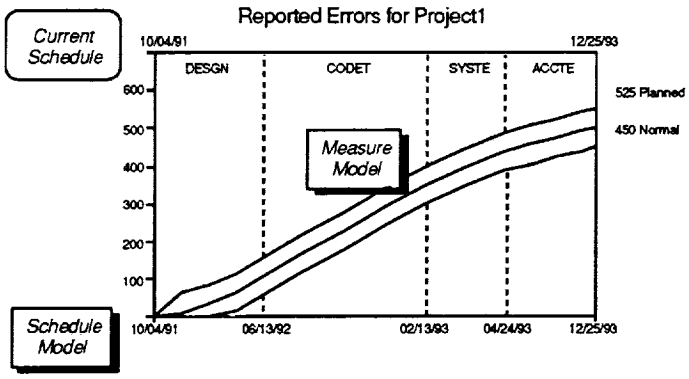
Steps

1. Use *Get Project Dates* with the current schedule to obtain the project start and end date. Calculate the number of weeks planned between these dates ($Planned\ Weeks_{Total}$).
2. Scale the plot's x-axis to the number of weeks in the project's planned schedule

   $X\text{-}Axis\ Scale = Planned\ Weeks_{Total}$
3. Use *Get Estimated Completion Value* with the current estimates to obtain the manager's planned completion value for the measure ($Planned\ Value_{Completion}$).
4. Use *Get Project Magnitude* with the current estimates to obtain the measure and estimated completion value for that measure which is most indicative of the project's magnitude.
5. On the basis of that magnitude, use *Determine Normal Estimate Set* with the estimate set model to create a normal set of estimates for the project.
6. Use *Get Estimated Completion Value* with the normal estimates to obtain the normal completion value for the measure ($Normal\ Value_{Completion}$).
7. Examine the measure model for the measure of interest and obtain the maximum fractional value expected for the measure at any point in the life cycle ($Maximum\ Value_{Model}$).
8. Compute the maximum value that the upper bound of the normal measure guidelines would attain over the life cycle as

   $Maximum\ Upper\ Range = Normal\ Value_{Completion} * (Maximum\ Value_{Model} + Normal\ Deviation)$
9. Scale the plot's y-axis to the maximum of either the manager's planned completion value or the current measure value found in the measure data or maximum upper bound value of the normal measure guidelines.

   $Y\text{-}Axis\ Scale = Maximum\ (Planned\ Value_{Completion}, Actual\ Value_{Current}, Maximum\ Upper\ Range)$
10. Display the basic plotting area with appropriate axes, labels, and title.



STEPS

1. The plot's x-axis is scaled to the project duration of 116 weeks.

2. Based on the project's magnitude, a set of normal estimates is generated. From this set of estimates, a completion estimate of 450 errors is obtained.

3. The y-axis plot is scaled to either the maximum model value, the manager's planned completion estimate, or the current measure value, whichever is greatest.

4. The plot is displayed with appropriate labels and titles.

Figure 3-11. Scaling the Comparison to Normal Plotting Area

3.2.3.2 Plot Normal Project Data for a Measure

Purpose

Plots the normal measure and schedule values to expect over the development life cycle as guidelines for the measure of interest. Adds labels to the plot for calendar dates associated with the normal schedule and for the normal measure value to expect at completion.

Required Data

- Project start and end dates (input value)
- Normal measure value at completion date (input value)
- Schedule model
- Measure model (for measure of interest)

Steps

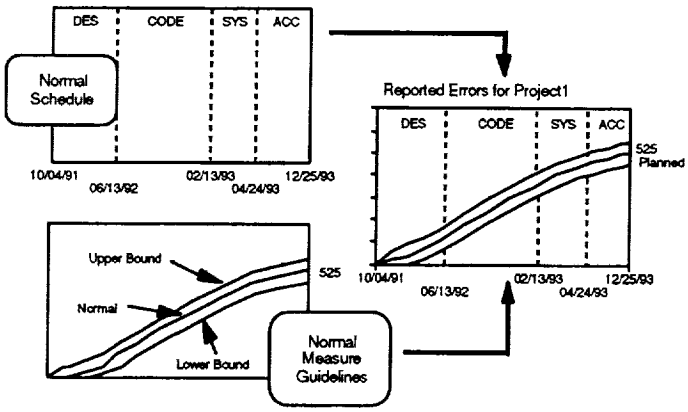
1. Use *Determine Normal Schedule* with the input project start and end dates to scale the schedule model to match the project's duration and generate a normal schedule for the current project.

2. For each phase in the normal schedule, draw a vertical line through the plotting area representing the end date of each phase. Label the names of the phases in the normal schedule across the top of the plotting area. Label relevant calendar dates under the x-axis of the plot to identify the project start date, the project end date, and the end date of each phase.

3. Use *Determine Normal Measure Guidelines* with the input normal completion value to scale the measure model and generate expected measure values, with upper and lower normal bounds on those values, as a function of schedule for the current project.

3. Plot the values computed for the normal measure guidelines over the life cycle as a shaded area consisting of three related curves—the upper bound expected for the measure, the normal measure value expected, and the lower bound expected for the measure.

4. Label the normal completion value for the measure at its correct height on the right side of the plotting area. ($Normal\ Value_{Completion}$)



STEPS

1. The project's start and end dates are used to scale the schedule model and generate a normal schedule.

2. The phase names and end dates are added to the display. A vertical line is also drawn corresponding to each phase's end date.

3. The measure model is scaled by the normal completion value for the measure, with upper and lower bounds added.

4. The normal measure guidelines are plotted as three curves shaded in between.

Figure 3-12. Plotting Normal Project Values for a Measure

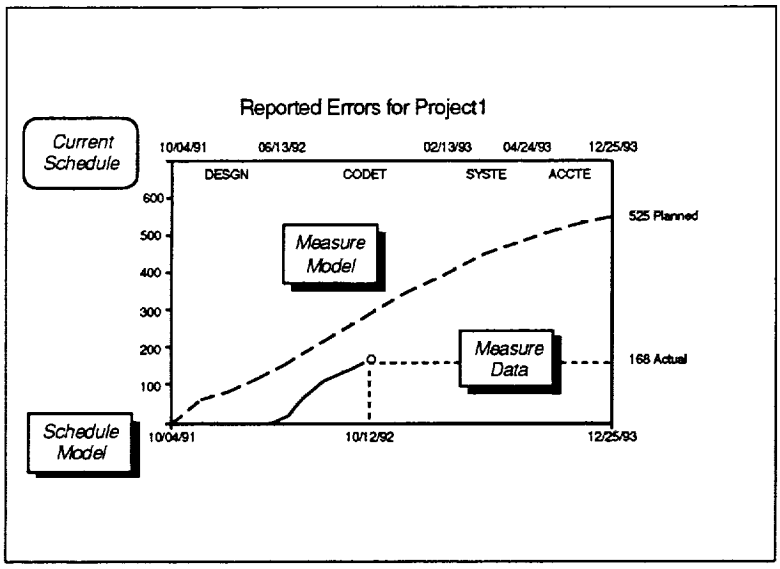
3.2.4 Comparison to Manager's Plan

Purpose

Compares the actual cumulative values of a measure of interest for the current project to the measure's expected behavior given the manager's current plan.

Description

The comparison function can visually contrast the actual recorded behavior of a measure of interest for the current project with the measure's expected behavior given the manager's current schedule and estimates. The comparison is depicted graphically by "superimposing" a reference plot representing the planned project behavior on an observational plot containing the actual measure values. Derived from the current schedule and a measure model, the reference plot shows the expected measure values as a function of planned schedule. The measure and schedule values reflect the planned size and duration of the project.



Note that the comparison also applies to the ratio of any two such measures. This permits examining an extended set of measures such as LOC per hour (coding productivity).

The figure shows a representative comparison plot of reported errors for a sample project. This example indicates (1) the project's 168 reported errors are below what can be expected as of 10/12/92 given the manager's current plan and (2) the manager plans to see 525 errors at project completion.

Figure 3-13. Comparing a Measure to the Manager's Plan

Required Information

- Planned start and end dates of each phase (current schedule)
- Actual data values for the measure of interest (measure data)
- Estimated completion value of the measure of interest (current estimates)
- Model of the measure of interest for similar projects (measure model)

Key Steps

1. Scale and display the basic plotting area to use for comparisons with the manager's plan.
2. Plot the expected measure values and current schedule given the manager's plan for the project.
3. Use *Plot Actual Data for a Measure*, as in simple observation, to overlay the actual measure values and the manager's planned completion value on the plot.

3.2.4.1 Scale and Display Plot Area for Comparison to Plan

Purpose

Scales the plotting area to use for comparing a measure to planned project behavior and generates the plot axes, labels, and title.

Required Data

- Current schedule
- Current estimates
- Measure data (for measure of interest)
- Measure model

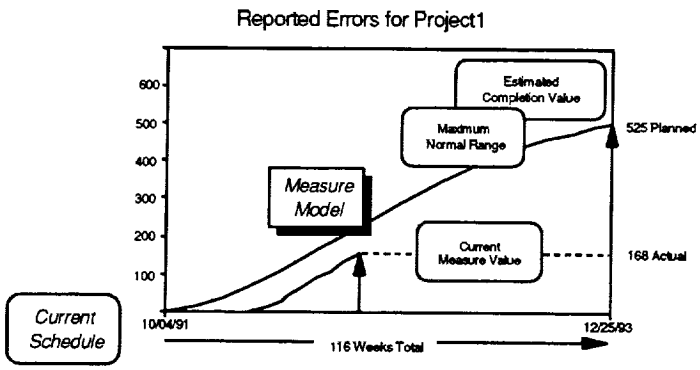
Steps

1. Use *Get Project Dates* with the current schedule to obtain the project start and end date. Calculate the number of weeks planned between these dates ($Planned\ Weeks_{Total}$).
2. Scale the plot's x-axis to the number of weeks in the project's planned schedule

   $X\text{-}Axis\ Scale = Planned\ Weeks_{Total}$
3. Use *Get Estimated Completion Value* with the current estimates to obtain the manager's planned completion value for the measure ($Planned\ Value_{Completion}$).
4. Examine the measure model for the measure of interest and obtain the maximum fractional value expected for the measure at any point in the life cycle ($Maximum\ Value_{Model}$).
5. Compute the maximum value that the planned measure would attain over the life cycle as

   $Maximum\ Expected\ Value = Planned\ Value_{Completion} * Maximum\ Value_{Model}$
6. Scale the plot's y-axis to the maximum of either the maximum expected measure value given the manager's planned completion value or the current measure value found in the measure data.

   $Y\text{-}Axis\ Scale = Maximum\ (Maximum\ Expected\ Value,\ Actual\ Value_{Current})$
7. Display the basic plotting area with appropriate axes, labels, and title.



STEPS

1. The plot's x-axis is scaled to the project duration of 116 weeks.

2. The manager's planned completion estimate, of 525 reported errors, is obtained.

3. The maximum value of the model over the life cycle is obtained. Using this value, the maximum value to expect for the planned measure is calculated.

4. The y-axis of the plot is scaled to the maximum planned measure value, the manager's planned completion value, or the actual measure value, whichever is greatest.

5. The plot is displayed with labels and titles.

Figure 3-14. Scaling the Comparison to Plan Plotting Area

3.2.4.2 Plot Planned Project Data for a Measure

Purpose

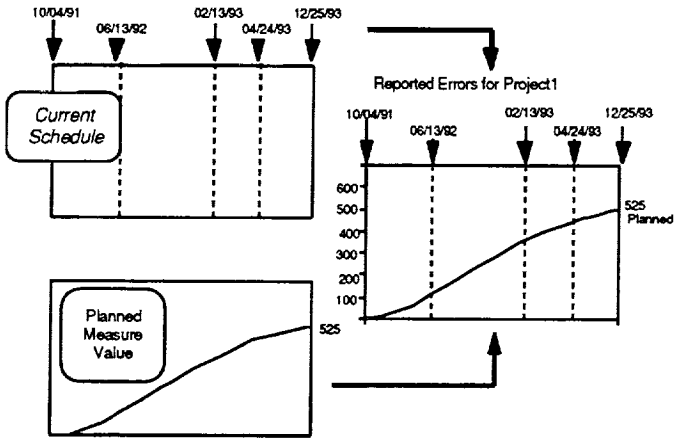
Plots the planned measure and schedule values to expect over the development life cycle for the measure of interest. Adds labels to the plot for calendar dates associated with the current schedule and for the planned measure value at project completion.

Required Data

- Current schedule
- Estimated measure value at completion (input value)
- Measure model (for measure of interest)

Steps

1. Using the dates in the current schedule, label the top of the plotting area to identify the project start date, the project end date, and the end date of each phase.
2. Calculate the planned number of weeks between the project start and end dates found in the current schedule ($Planned\ Weeks_{Total}$).
3. For each life-cycle phase in the current schedule, calculate the number of weeks planned between the start and end dates of the phase ($Planned\ Weeks_{In\ Phase}[i]$).
4. For each life-cycle phase, normalize the amount of time planned for the phase by the total project duration to compute the fraction of duration planned for that phase as $Fraction\ of\ Duration_{In\ Phase}[i] = Planned\ Weeks_{In\ Phase}[i] \ / \ Planned\ Weeks_{Total}$
5. Using these fractional values, create a schedule model that models the current project's schedule as planned by the manager.
6. Use *Convert Phase to Date* on this model of the planned schedule, specifying as input the project start and end dates, to determine the calendar dates associated with each phase and phase segment defined in the measure model ($Expected\ Calendar\ Date\ [i]$).
7. For each calendar date calculated, compute the date's relative week number as the number of weeks between the project start date and the date itself ($Expected\ Week\ [i]$).
8. Also for each phase and phase segment, use *Convert Phase to Measure* on the measure model, specifying as input the estimated completion value as planned by the manager, to determine the expected measure values that correspond to the computed dates ($Expected\ Measure\ Value\ [i]$).
9. Show the planned behavior of the measure of interest over the life cycle as a curve through the points just computed for each phase and phase segment by plotting expected measure value as a function of expected week.
10. Label the planned completion value for the measure at its correct height on the right side of the plotting area. ($Planned\ Value_{Completion}$)



STEPS

1. The manager's schedule is modeled based on the fraction of the total project duration to be spent in each phase. For this project, the fractions are DESGN 0.26, CODET 0.34, SYSTE 0.18, and ACCTE 0.22.

2. The start and end dates for each phase are determined and their relative week numbers are calculated.

3. Expected measure values at each phase segment are calculated, relative to the final completion estimate.

4. The planned behavior of the measure is plotted and labeled.

Figure 3-15. Plotting Planned Project Values for a Measure

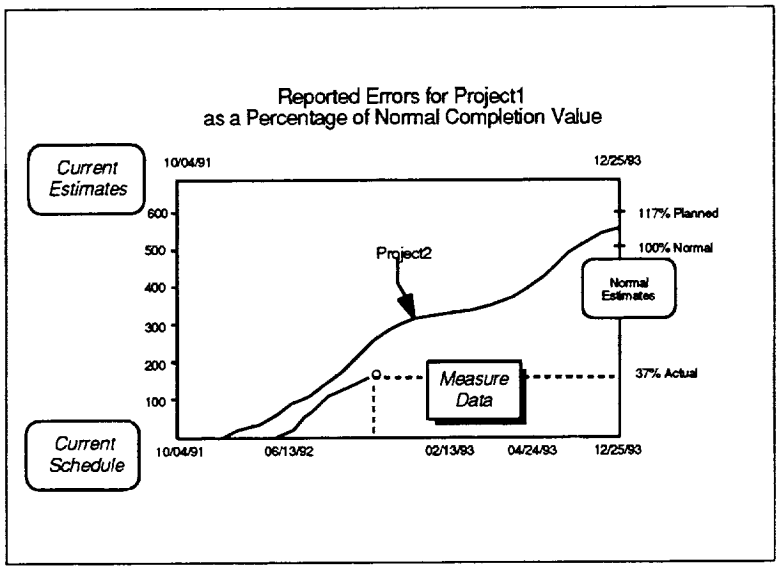
3.2.5 Comparison to Other Projects

Purpose

Compares the actual cumulative values of a measure of interest for the current project to the measure's behavior on another project.

Description

The comparison function can visually contrast the actual recorded behavior of a measure of interest for the current project with the measure's behavior as observed on other projects. The comparison is depicted graphically by overlaying reference plots of measure data from one or more selected comparison projects on an observational plot containing actual measure values for the current project. To eliminate the effects of project size, the measure values plotted for the current project and any selected comparison projects are scaled to reflect a percentage of that project's normal completion value expected for the measure. Similarly, the schedules of all comparison projects are scaled to match the duration of the current project. A comparison project may be either a completed project that reflects an earlier development effort or an ongoing project.



Note that the comparison also applies to the ratio of any two such measures. In this case, however, the measure values are plotted as absolute values and need not be scaled to reflect a percentage of the normal completion value.

The figure shows a sample plot of reported errors for Project1 and a comparison project, Project2. This example indicates (1) errors for Project1 are 37% of the total number normally expected at completion and (2) relatively more errors were reported on Project2.

Figure 3-16. Comparing a Measure to Other Projects

Required Information

- Project start and end dates (current schedule)
- Actual data values for the measure of interest (measure data)
- Estimated completion values for all measures (current estimates)
- Model of the measure of interest for similar projects (measure model)

- Model of completion estimates for similar projects (estimate set model)
- List of projects with data available for the measure (project/measure availability list)
- Characteristics data for the comparison project (project characteristics)
- Project start and end dates for comparison project (current schedule)
- Actual data values of measure for comparison project (measure data)
- Estimated completion values for comparison project (current estimates)
- Model of the measure of interest for comparison project (measure model)
- Model of completion estimates for comparison project (estimate set model)

Key Steps

1. Scale and display the basic plotting area to use for comparisons with other projects.
2. Scale and plot the actual measure values for the current project.
3. Select a comparison project with data available for the measure of interest.
4. Scale and plot the actual measure values for the comparison project.

3.2.5.1 Scale and Display Plot Area for Comparison to Other Project

Purpose

Scales the plotting area to use for comparing a measure to actual data from other projects and generates the plot axes, labels, and title.

Required Data

- Current schedule
- Current estimates
- Measure data (for measure of interest)
- Estimate set model

Steps

1. Use *Get Project Dates* with the current schedule to obtain the project start and end date. Calculate the number of weeks planned between these dates ($Planned\ Weeks_{Total}$).
2. Scale the plot's x-axis to the number of weeks in the current project's schedule.

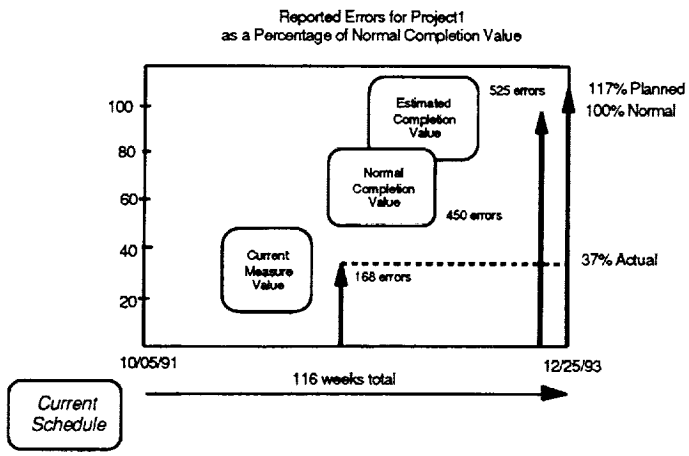
   $X\text{-}Axis\ Scale = Planned\ Weeks_{Total}$
3. Use *Get Project Magnitude* with the current estimates to obtain the measure and estimated completion value for the measure that is most indicative of the current project's magnitude.
4. On the basis of that magnitude, use *Determine Normal Estimate Set* with the estimate set model to create a normal set of estimates for the project.
5. Use *Get Estimated Completion Value* with the normal estimates to obtain the normal completion value for the measure ($Normal\ Value_{Completion}$).
6. Use *Get Estimated Completion Value* with the current estimates to obtain the manager's planned completion value for the measure ($Planned\ Value_{Completion}$).
7. Divide the manager's planned completion value by the normal completion value computed for the measure to determine the planned value as a percentage of the normal value at completion using

   $Planned\ Percent_{Completion} = (Planned\ Value_{Completion} / Normal\ Value_{Completion}) * 100$
8. Divide the current measure value found in the measure data by the normal completion value computed for the measure to determine the percentage of the normal completion value seen to date using

   $Actual\ Percent_{Current} = (Actual\ Value_{Current} / Normal\ Value_{Completion}) * 100$
9. Scale the plot's y-axis to the maximum of either 100% of the normal completion value for the measure, the current measure value found in the measure data expressed as a percentage, or the planned completion percentage.

   $Y\text{-}Axis\ Scale = Maximum\ (100, Actual\ Percent_{Current}, Planned\ Percent_{Completion})$

10. Display the basic plotting area with appropriate axes, labels, and title.



STEPS

1. The x-axis is scaled to the project duration of 116 weeks.

2. Based on the project's magnitude, a normal set of estimates is generated. From these estimates, the normal completion estimate is obtained.

3. The manager's planned completion value and the current measure value are obtained and converted to a percentage of the normal completion value.

4. The y-axis is scaled to the maximum of: 100% of the normal completion value, the manager's planned completion percentage, or the current measure value.

Figure 3-17. Scaling the Comparison to Other Projects Plotting Area

3.2.5.2 Plot Actual Data for Current Project

Purpose

Plots the actual data values of the measure of interest from project start through the current date as a percentage of the normal completion value. Adds labels to the plot to identify the percentages for the actual measure value to date, the normal measure value at completion, and the planned completion value.

Required Data

- Normal completion value (input value)
- Current schedule
- Current estimates
- Measure data (for measure of interest)

Steps

1. Initialize the starting point for plotting the actual measure data as a function of week number to indicate the measure value is zero at week number zero using

   $X\text{-}Value[0] = 0$ and $Y\text{-}Value[0] = 0$

2. For each entry in the measure data through the current date, set the x and y values of the next point to plot to the week number of the sample date and its corresponding measure value as follows:

   $X\text{-}Value[i] = Week(i)$ and $Y\text{-}Value[i] = Measure\ Value[i]$

   for the ith entry in the measure data, where Week(i) is the relative week number of the ith entry

3. Scale each y value to reflect the actual measure value expressed as a percentage of the normal completion value for the measure using

   $Y\text{-}Value[i] = (Y\text{-}Value[i] / Normal\ Value_{Completion}) * 100$

4. Plot the percentages computed for the actual measure data by week number, from 0 through the current week N, as a step function (i.e., between any two points plot the rise and then the run)

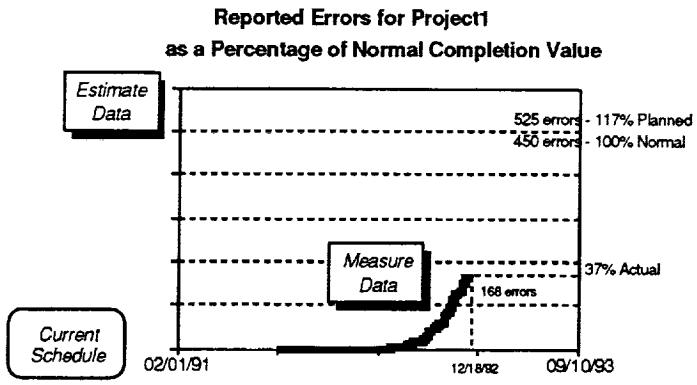
5. Label the x-axis with the project start and end dates from the current schedule.

6. Use *Get Estimated Completion Value* with the current estimates to obtain the manager's planned completion value for the measure ($Planned\ Value_{Completion}$). Scale the planned completion value to express it as a percentage of the normal value at completion using

   $Planned\ Percent_{Completion} = (Planned\ Value_{Completion} / Normal\ Value_{Completion}) * 100$

7. Label the manager's planned completion value for the measure as a percentage at its correct height on the right side of the plotting area. ($Planned\ Percent_{Completion}$)

8. Label the actual measure value observed on the current date as a percentage of the normal completion value at its correct height on the right side of the plotting area. (*Y-Value[N]*)



STEPS

1. The x-values are set to successive week numbers, while the y-values are set to each week's corresponding measure value.

2. The y-values are scaled to reflect the actual measure value expressed as a percentage of the normal completion value.

3. The points are plotted as a step function.

4. The manager's planned completion value is scaled to a percentage of the normal completion value for the measure and displayed.

Figure 3-18. Plotting Actual Values as a Percentage of the Normal Completion Value

3.2.5.3 Select a Comparison Project

Purpose

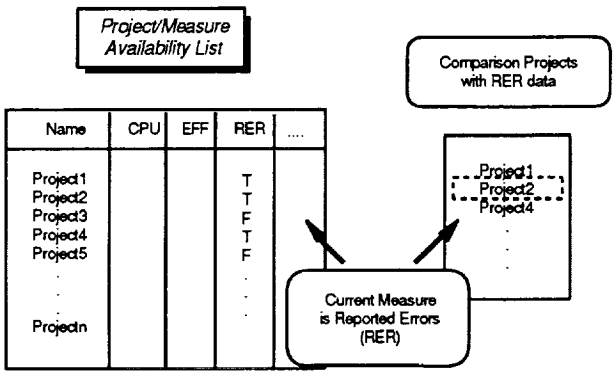
Allows the user to select a project from a list of comparison projects that have data for the measure of interest. Identifies appropriate models, as needed, for the selected comparison project whenever the project has different project characteristics from the current project of interest.

Required Data

- Project/measure availability list
- Project characteristics (for selected comparison project)
- Project characteristics (for current project of interest)

Steps

1. Examine the project/measure availability list to obtain a list of all projects that have measure data for the measure of interest.
2. Display the list of potential comparison projects and permit the user to select a project from the list.
3. Obtain the characteristics of the selected project from its project characteristics data.
4. Concatenate the characteristics to produce a project type that identifies the appropriate models for the comparison project.
5. If the project type of the comparison project differs from that of the current project, identify and locate suitable models for temporary use with the comparison project.



STEPS

1. A list of all projects having data for reported errors is generated. The user selects a project from this list.

2. Once the user selects a project, a project type is generated from the project characteristics of the comparison project.

3. If the project type of the comparison project differs from that of the current project, suitable models are identified and located for use with the comparison project.

Figure 3-19. Selecting a Comparison Project

3.2.5.4 Plot a Comparison Project for a Measure

Purpose

Plots the actual data values for a comparison project of the measure of interest as a percentage of its normal completion value. Scales the duration of the comparison project to match the planned duration of the current project of interest.

Required Data

- Planned duration in weeks (for current project of interest) (input value)
- Current schedule (for selected comparison project)
- Current estimates (for selected comparison project)
- Measure data (for selected comparison project)
- Estimate set model (for selected comparison project)

Steps

1. Initialize the starting point for plotting the actual measure data of the comparison project as a function of week number to indicate the measure value is zero at week number zero using

   $X\text{-}Value[0] = 0$ and $Y\text{-}Value[0] = 0$

2. For each entry in the measure data of the comparison project, set the x and y values of the next point to plot to the week number of the sample date and its corresponding measure value as follows:

   $X\text{-}Value[i] = Week(i)$ and $Y\text{-}Value[i] = Measure\ Value[i]$

   for the ith entry in the measure data, where Week(i) is the relative week number of the ith entry

3. Calculate the number of weeks between the project start and end dates found in the current schedule for the comparison project ($Number\ Of\ Weeks_{Total}$).

4. Scale each x value to force the duration of the comparison project to match the input planned duration of the current project of interest using

   $X\text{-}Value[i] = X\text{-}Value[i] * (Planned\ Duration_{Project\ Of\ Interest} / Number\ of\ Weeks_{Total})$

5. Use *Get Project Magnitude* with the current estimates to obtain the measure and estimated completion value for that measure which is most indicative of the comparison project's magnitude.

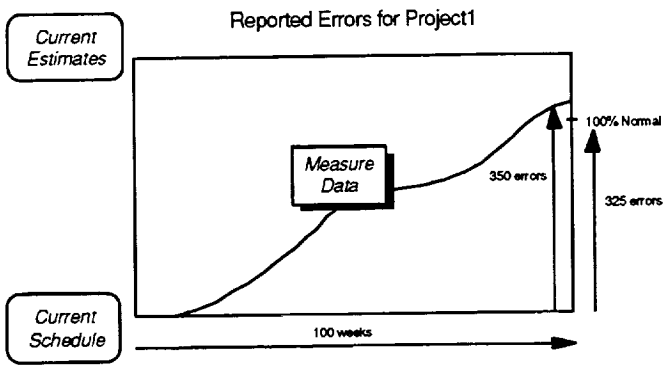
6. On the basis of that magnitude, use *Determine Normal Estimate Set* with the estimate set model to create a normal set of estimates for the comparison project.

7. Use *Get Estimated Completion Value* with the normal estimates to obtain the comparison project's normal completion value for the measure ($Normal\ Value_{Completion}$).

8. Scale each y value to reflect the actual measure value expressed as a percentage of its normal completion value for the measure using

   $Y\text{-}Value[i] = (Y\text{-}Value[i] / Normal\ Value_{Completion}) * 100$

9. Plot the percentages computed for the actual measure data by its scaled week number, for each data point 0 through N.



STEPS

1. The x-values are set to successive week numbers, while the y-values are set to each week's measure value.

2. The duration of the comparison project is scaled to match the current project.

3. Based on the comparison project's magnitude, a normal set of estimates is created for it.

4. The normal completion value for the comparison project's measure is obtained from the estimates.

5. The y-values are scaled to a percentage of the comparison project's normal value and plotted.

Figure 3-20. Plotting Comparison Project Values for a Measure

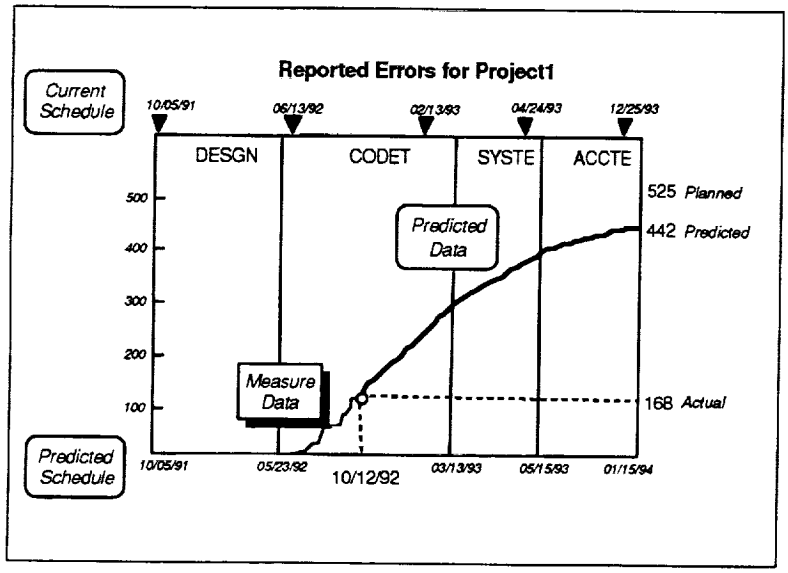
3.2.6 Prediction

Purpose

> Forecasts the probable completion date and the expected completion value of a fundamental software development measure for a given project.

Description

The prediction function forecasts the probable future behavior of the measure of interest for the current project. To accomplish this, the SME fits schedule and measure models of typical project behavior to the actual data collected for the project. The results are depicted as an extension to the observational plot for the current measure with predicted data values shown through a predicted completion date.



Note that predictions may be made for any measure of interest defined by the SME provided actual data has been collected for that measure.

The figure shows a representative prediction of reported errors for a sample project. This example indicates that the SME expects the project to finish 3 weeks behind schedule with approximately 83 fewer errors than currently planned.

Figure 3-21. Representative Prediction

Required Information

- Project start date (current schedule)
- Actual data values for the measure of interest (measure data)
- Model of the schedule for similar projects (schedule model)
- Model of the measure of interest for similar projects (measure model)
- Estimate of the life-cycle phase on a given date (phase estimate)

Key Steps

1. Obtain a phase estimate to serve as the basis for making the prediction.
2. Predict the probable completion date of the project.
3. Predict the expected measure value at project completion.
4. Predict the future measure values expected through project completion.

3.2.6.1 Obtain a Phase Estimate

Purpose

Obtains a phase estimate, based on any one of three discrete methods, that identifies where the project was in the development life cycle on a specific date.

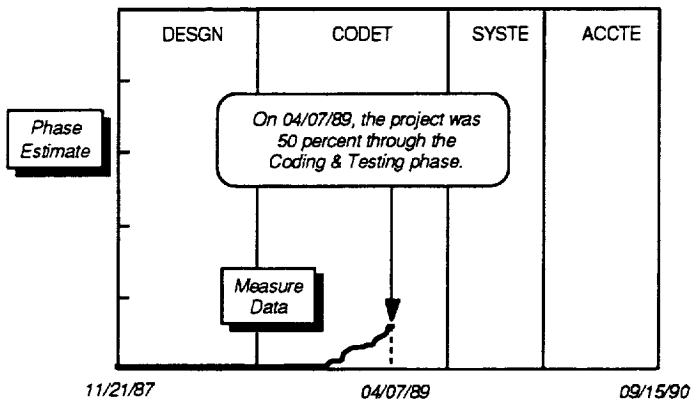
Required Data

- Current schedule (Methods 1 and 2)
- Current estimates (Method 1 only)
- Measure data (for each measure) (Method 1 only)
- Schedule model (Method 1 only)
- Measure model (for each measure) (Method 1 only)

Steps

1. Use Method 1 to analyze all available measures and calculate an overall average phase estimate for the current date.
2. Use Method 2 to examine the current schedule and derive a phase estimate from the most recently completed phase prior to the current date.
3. Allow the user to select the phase estimate resulting from either Method 1 or Method 2, or let the user interactively specify the values for the phase estimate (Method 3).

Note: To serve as a valid basis for a prediction, a phase estimate must satisfy two requirements. First, the date specified in the phase estimate must be between the project start date and the current date. Second, the value of the measure of interest as of the date specified in the phase estimate must be non-zero. These requirements ensure the existence of an objective measurement that can be extrapolated into the future.



NOTE

The figure depicts a sample phase estimate for an ongoing project. The phase estimate consists of a specific date, the life-cycle phase on that date, and the completed percentage of that phase. Non-zero measure data should exist on the specified date before the phase estimate can be used in a prediction.

Notice that the date specified does not fall exactly in the middle (at 50%) of the CODET phase, but instead indicates that the project is slightly behind schedule.

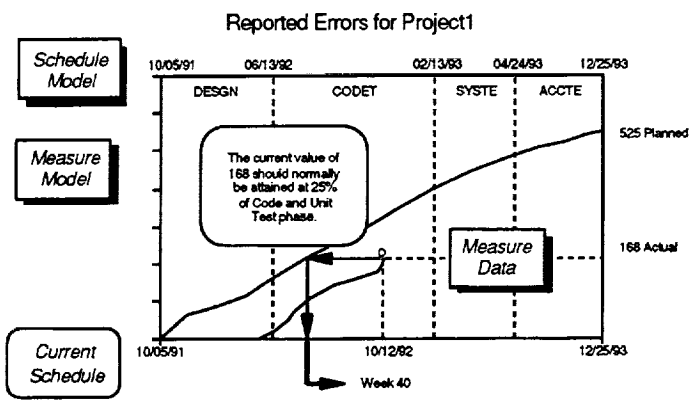
Figure 3-22. Sample Phase Estimate

Method 1—Calculated by the SME Using Phase Analysis

For each available measure, calculate the week number corresponding to the phase at which the measure normally attains its current value as follows:

1. Determine the current value for the measure from the project's measure data.
2. Determine the expected completion value for the measure from the project's estimates data.
3. Use *Convert Measure to Phase* with these values to obtain the phase and fraction of phase that is characteristic of the measure's current value from the measure model.
4. Given the project start and end dates from the current schedule, use *Convert Phase to Date* on the schedule model to determine the calendar date that matches the calculated phase and fraction of phase.
5. Compute the relative week number of this date as the number of weeks between the project start date and the calendar date (*Week Number[i]*).

Note: A measure must meet three conditions to be considered an available measure for this algorithm. These conditions are (1) data must exist for the measure as indicated by the project/measure availability list, (2) the expected completion value for the measure contained in the current estimates must be non-zero, and (3) the current value of the measure must show a positive trend by exceeding 10% of its estimated completion value.



STEPS

1. The current value for the measure is 168.

2. The expected completion value for the measure is 525.

3. The measure model indicates that this value, relative to the expected completion value, is characteristic of 25% through CODET.

4. The schedule model indicates that, relative to the project start and end dates, 25% through CODET would be 8/27/92, or 40 weeks from the project start date.

Figure 3-23. Phase Analysis for One Measure

Using the intermediate results calculated for each available measure above, obtain an overall averaged phase estimate for the current date as follows:

1. Average the week numbers computed for each available measure as indicative of the project's phase using

$$\text{Average Week} = \left(\sum_{i=1}^{K} \text{Week Number}[i] \right) / K$$

   where i refers to the available measures 1 through K

2. Obtain the calendar date corresponding to the averaged week number by adding it to the project start date.
3. Given the project start and end dates from the current schedule, use *Convert Date to Phase* on the schedule model to determine the average phase and fraction of phase that matches this calculated calendar date.
4. Set the phase estimate to reflect the averaged phase and fraction of phase as of the current date.

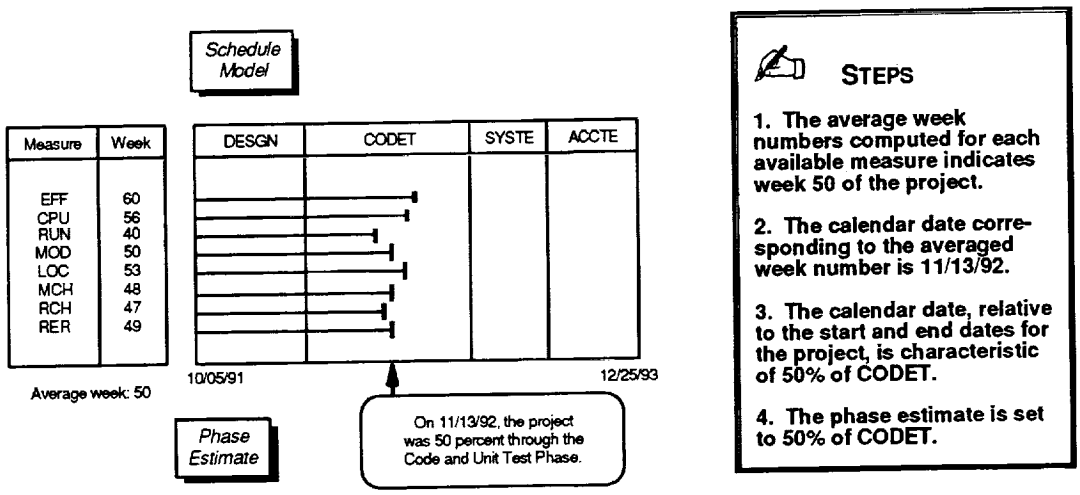


Figure 3-24. Averaging Phases from All Available Measures

Method 2—Derived from the Current Schedule

Assuming that the project's schedule is accurate and up-to-date, obtain the phase estimate from the current schedule as follows:

1. Identify the most recently completed phase prior to the current date by iteratively using *Get Scheduled Phase Dates* on each phase in the current schedule to locate the last phase whose end date satisfies the following

   Phase End Date [k] <= Current Date

2. Set the phase estimate to reflect that the identified phase was 100% complete on its scheduled end date.

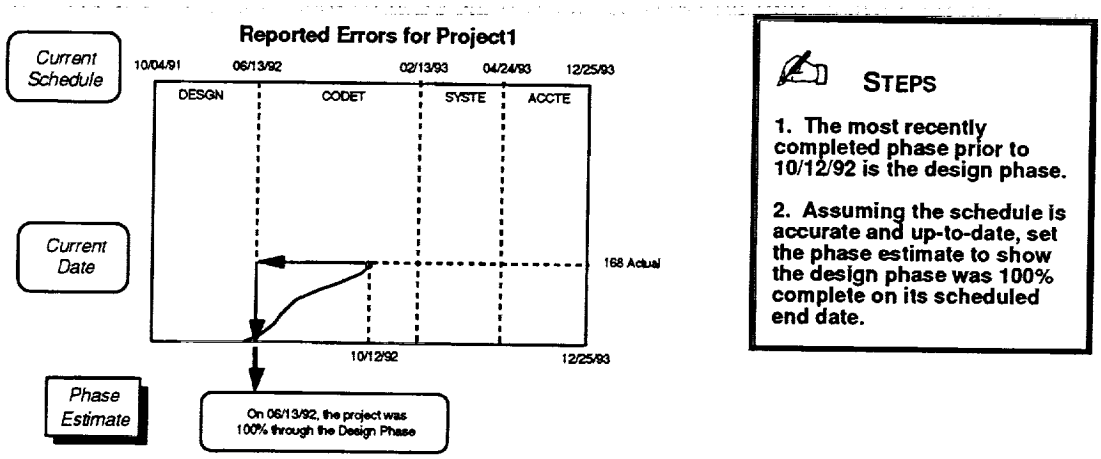


Figure 3-25. Deriving a Phase Estimate from the Current Schedule

3.2.6.2 Predict Completion Date of Project

Purpose

Predicts the probable completion date of a project on the basis of the amount of time actually expended through a known point in the project's life cycle.

Required Data

- Phase estimate (input value)
- Current schedule
- Schedule model

Steps

1. Calculate the actual number of weeks from the project start date in the current schedule through the reference date in the phase estimate ($Actual\ Weeks_{To\ Date}$).
2. Using the schedule model, calculate the fraction of the total project duration normally expended through the reference phase and fraction of phase in the phase estimate as

$$Normal\ Fraction\ of\ Duration_{To\ Date} = \sum_{i=1}^{k-1} Fraction\ of\ Duration_{In\ Phase}[i] + F * Fraction\ of\ Duration_{In\ Phase}[k]$$

for the kth phase and an elapsed fraction of phase equal to F

3. Linearly extrapolate the total number of weeks expected to be required to complete the project as

$$Predicted\ Weeks_{Total} = Actual\ Weeks_{To\ Date} \ / \ Normal\ Fraction\ of\ Duration_{To\ Date}$$

4. Obtain the predicted completion date by adding the total number of weeks predicted to the project start date.

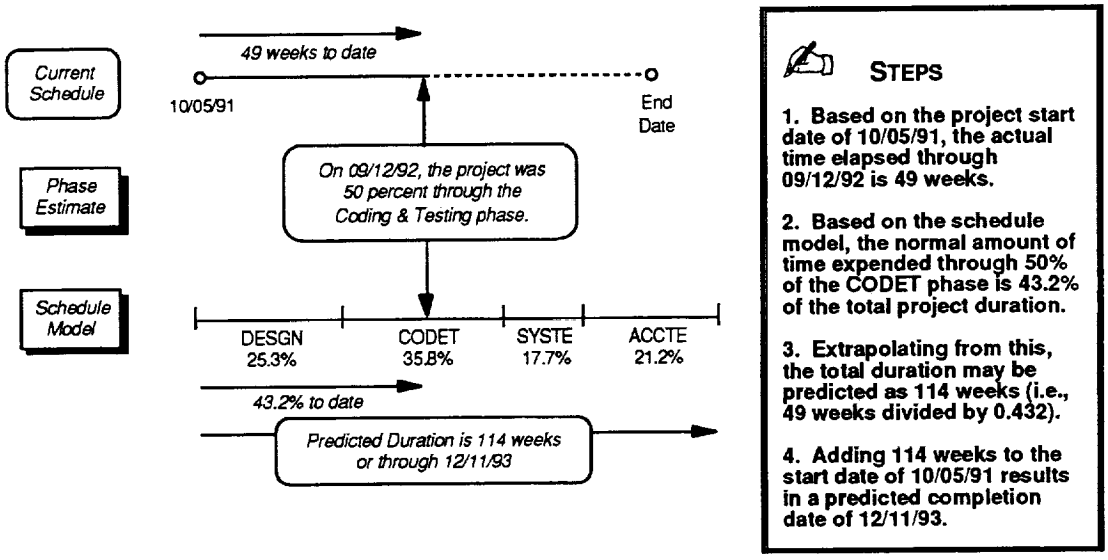


Figure 3-26. Predicting a Completion Date

3.2.6.3 Predict Measure Value at Completion

Purpose

Predicts the expected measure value at project completion on the basis of the value of the measure actually observed at a known point in the project's life cycle.

Required Data

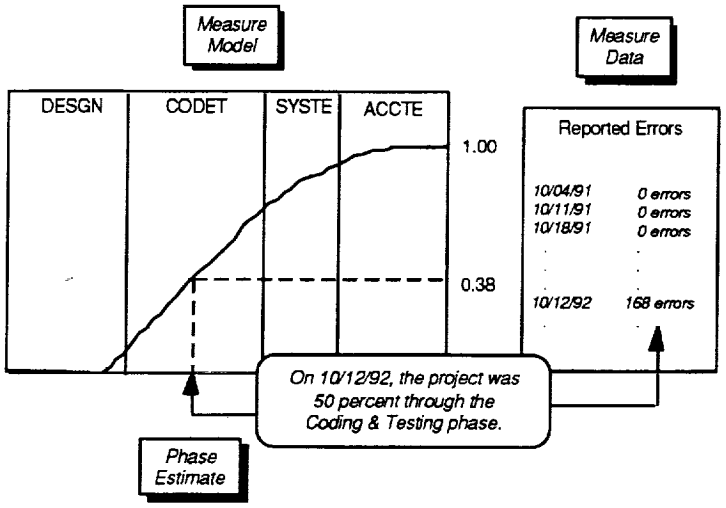
- Phase estimate (input value)
- Measure data
- Measure model

Steps

1. Obtain the actual cumulative value for the measure on the reference date in the phase estimate from the measure data (*Actual Measure*$_{To\ Date}$).
2. Use *Convert Phase to Measure* to determine the fraction of the total measure tabulated in the measure model as normally observed through the reference phase and fraction of phase in the phase estimate (*Normal Fraction of Measure*$_{To\ Date}$).

   Note: Specify an expected completion value of 1.0 for *Convert Phase to Measure* to obtain fractional, as opposed to absolute, measure values for the phase.
3. Linearly extrapolate the measure value to be expected at project completion as

   Predicted Measure$_{Total}$ = *Actual Measure*$_{To\ Date}$ / *Normal Fraction of Measure*$_{To\ Date}$



STEPS

1. Based on the measure data for the project, the actual cumulative value for the measure through 10/12/92 is 168 errors.

2. Based on the measure model for errors, the number of reported errors normally seen through 50% of the CODET phase is 38% of the total number expected at project completion.

3. Extrapolating from these values, the total number of reported errors at project completion may be predicted as 442 errors (i.e., 168 errors divided by 0.38).

Figure 3-27. Predicting a Measure's Completion Value

3.2.6.4 Predict Intermediate Values Through Completion

Purpose

Calculates predicted data values for the measure between the date specified in the phase estimate and the predicted completion date.

Required Data

- Phase estimate (input value)
- Predicted completion date (input value)
- Predicted measure value at completion (input value)
- Current schedule
- Schedule model
- Measure model

Steps

For each data point to be predicted between the date specified in the phase estimate and the predicted completion date, the SME performs the following computation:

1. Given the project start date and the predicted completion date, use *Convert Date to Phase* with the schedule model to translate the date of the desired data point into a phase and fraction of phase.
2. Given the phase and fraction of phase matching the desired date and the predicted measure value at completion, use *Convert Phase to Measure* with the measure model to determine the predicted measure value to expect at that point in the life cycle.

Note: Conceptually, this algorithm would be used to predict values for each week. In reality, however, one need only address points matching the granularity of the models.

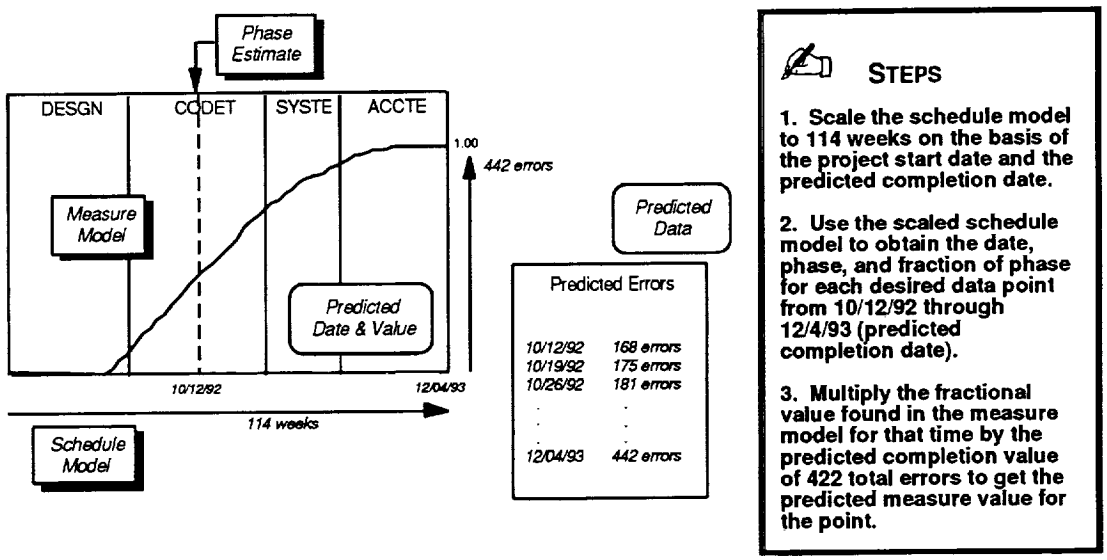


Figure 3-28. Predicting a Measure's Intermediate Values

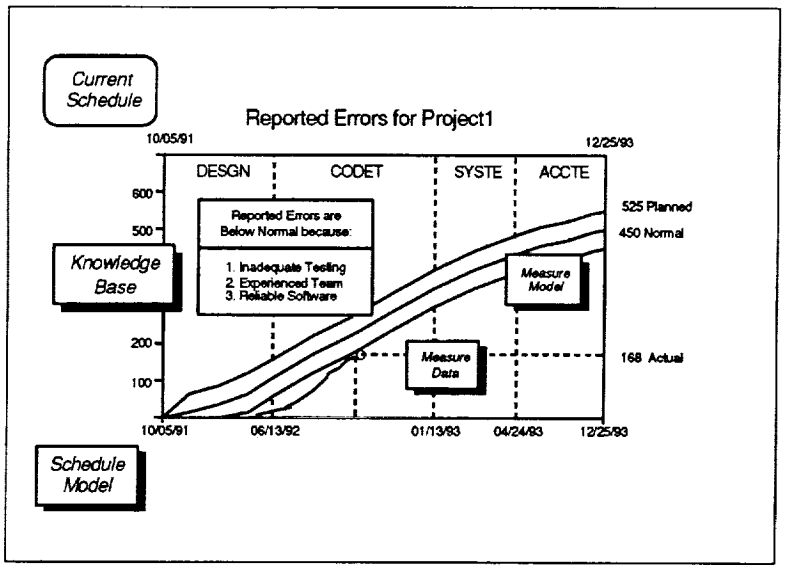
3.2.7 Trend Analysis

Purpose

Displays a list of possible reasons to explain an observed deviation in the measure of interest for the current project.

Description

The trend analysis function uses expert systems techniques to identify the probable causes of a deviation in the measure of interest. The analysis compares the current value of the measure to a model of the measure and determines if the measure's value is within an acceptable range of its expected value. If the measure falls outside of the acceptable range, the function uses captured management experience to evaluate various known information about the project and to reach conclusions to explain the deviation. The SME supports two discrete approaches for performing the analysis, a knowledge base used with individual measures and a rule base used with ratios of measures.



The figure illustrates trend analysis of a measure of interest. This example shows a list of probable causes of a lower than normal value for reported errors. Since the measure of interest is a single measure, the knowledge base is used to consider not only the current measure, but also other measure values and subjective data, in reaching these conclusions.

If the measure of interest is a ratio of two measures, the rule base is used instead of the knowledge base.

Figure 3-29. Analyzing Trends In a Measure of Interest

Required Information

- Management experience for interpreting trends (knowledge base, rule base)
- Actual data values for the available measures (measure data)
- Planned schedule for the project (schedule data)
- Planned completion values for measures (estimates data)
- Models of measure behavior for similar projects (measure models)
- Model of the schedule for similar projects (schedule model)

- Model of completion estimates for similar projects (estimate set model)
- Subjective information about the project (subjective data)

Key Steps

1. Use the knowledge base to analyze trends if the measure of interest is one measure.
2. Use the rule base to analyze trends if the measure of interest is a ratio of measures.

3.2.7.1 Analyze Trends for a Single Measure of Interest

Purpose

Uses the management experience captured in the knowledge base to identify and display the probable causes of an observed deviation in a measure from its expected value.

Required Data

- Knowledge base
- Measure data (for all measures)
- Current schedule
- Current estimates
- Measure models (for all measures)
- Schedule model
- Estimate set model
- Subjective data

Steps

1. Use *Rate Objective Factor* to rate the current value of the measure of interest as either high, low, or normal with respect to its expected model guidelines. (Treat the measure of interest as an objective factor defined in the knowledge base.)

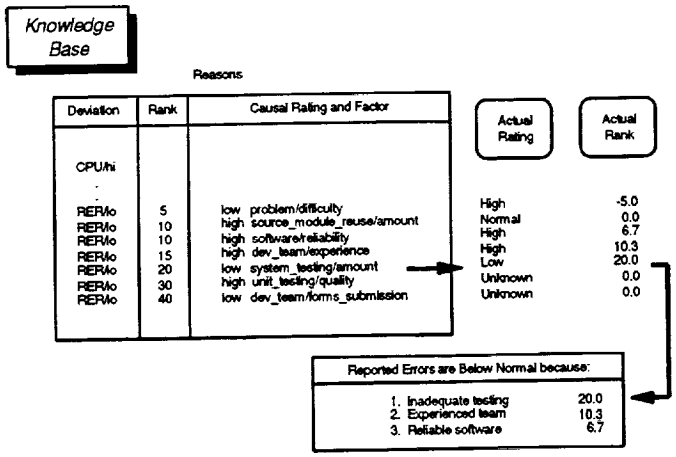
2. If the resultant rating is normal, indicate that trend analysis can not be performed when the measure of interest is within the acceptable range of normal values and quit.

3. For each reason in the knowledge base that applies to the observed deviation in the measure of interest (either lower or higher than normal), use *Evaluate Reason* to assess the probable validity and relative merit of the reason.

4. Sort the applicable reasons for the deviation by their computed relative ranking.

   Note: As computed by *Evaluate Reason*, a positive value for actual rank indicates the reason is a likely cause of the deviation in the measure. A zero value for actual rank indicates the reason is a potential cause of the deviation, but the evaluation was inconclusive. A negative value indicates the reason is not a likely cause.

5. Translate the encoded reasons into descriptive text, using the knowledge base's list of explanations, and display the sorted list of reasons and rankings for the user.



STEPS

1. If reported errors (RER) are observed to be "Low," a total of seven reasons can be found in the knowledge base to match that deviation.

2. Each possible reason, identified as a causal rating and factor pair, is evaluated to produce the reason's actual rating and rank (e.g., low system_testing/amount is indeed "Low" with a rank of 20.0).

3. The explanations that correspond to reasons with the highest actual rank are displayed for the user in descending rank order as likely or possible causes.

Figure 3-30. Analyzing Trends Using the Knowledge Base

3.2.7.2 Analyze Trends for a Ratio of Two Measures

Purpose

Uses the management experience captured in the rule base to identify and display the probable causes of an observed deviation from the expected value in a ratio of two measures.

Required Data

- Rule base
- Measure data (for all measures)
- Current schedule
- Measure models (for all measures)
- Schedule model
- Estimate set model

Steps

1. Use *Determine Rate for Rules* to rate the current value of the measure of interest (the ratio of the two measures) as either high, low, or normal with respect to its expected model guidelines.

2. If the resultant rating is normal, indicate that trend analysis can not be performed when the measure of interest is within the acceptable range of normal values and quit.

3. Use *Determine Phase for Rules* to identify the life-cycle phase that should correspond to the current date. (The result will be stored as the first assertion in a list for subsequent use in evaluating rules.)

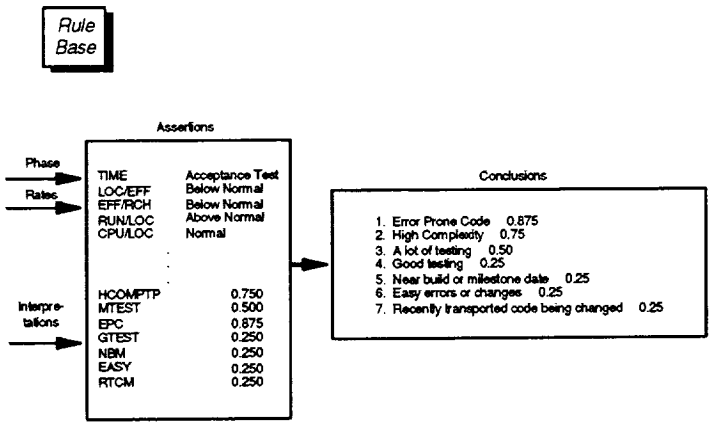
4. For each of the nine specific ratios of measures referenced by the rule base, use *Determine Rate for Rules* to determine if the ratio is above, below, or within the range of values normally expected on the current date. (The results will be stored in the assertion list for later use.)

   Note: The nine ratios of measures referenced are RUN/LOC, CPU/LOC, RCH/LOC, EFF/LOC, CPU/RUN, RCH/RUN, EFF/RUN, CPU/RCH, and EFF/RCH.

5. Use *Evaluate Rule* to evaluate each rule captured in the rule base and conditionally determine the applicability of the rule's interpretations. (Each rule that evaluates to true will have its interpretations stored in the assertion list.)

6. Sort the interpretations contained in the assertion list by their calculated certainties.

7. Translate the encoded interpretations into descriptive text, using the rule base's list of explanations, and display the sorted list of interpretations and certainties for the user.



STEPS

1. If the current ratio of measures is deviating from normal, the rule base is evaluated.

2. Evaluating the rule base (i.e., getting the current phase, evaluating nine given ratios of measures, and evaluating each rule in the rule base) results in a set of assertions.

3. In the assertions, each interpretation (e.g., a code of EPC and a certainty of 0.875) is the result of one or more rules that were true.

4. These encoded interpretations are translated, ordered by decreasing certainty, and displayed as conclusions to explain the deviation.

Figure 3-31. Analyzing Trends Using the Rule Base

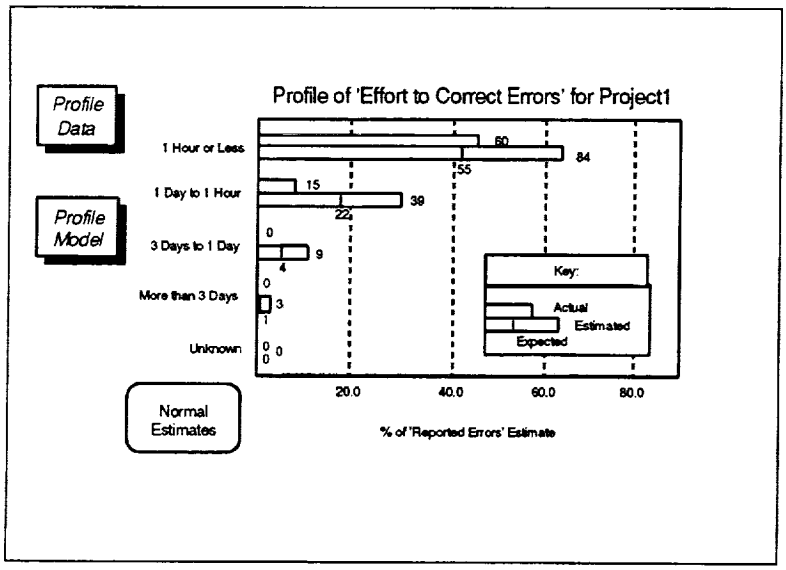
3.2.8 Profile Analysis

Purpose

Displays a distribution of the actual values recorded to date for the measure of interest within two or more discrete categories that constitute a defined profile.

Description

The profile analysis function lets users examine profile data associated with the measure of interest for the current project. Each set of profile data serves to break down the actual values of the measure into discrete categories. In effect, each profile constitutes one way of categorizing and viewing the measure's values in additional detail. Furthermore, multiple defined profiles may exist for a given measure. (The number of reported changes, for example, could be categorized based on the amount of effort required to implement the change, as well as based on the reason for the change.) The user may select any profile associated with the current measure for which profile data exists. Once a profile of interest is selected, the function obtains the current measure values in each category, the expected profile values on the current date, and the estimated profile values at project completion. The results are depicted graphically as a bar chart showing the distribution of values over the profile's defined categories.



The figure illustrates profile analysis of a measure of interest. This example shows a profile of the number of reported errors categorized into five bins by the amount of effort required to correct the error. For errors taking less than 1 hour to correct, the display indicates that (1) as of the current date 60 errors have been reported in this category while 55 errors are normally expected and (2) at project completion one should expect 84 errors in this category or 63% of the total.

Figure 3-32. Analyzing Profile Data for a Measure

Required Information

- List of available profiles for the project (project/profile availability list)
- Actual data values for the available profiles (profile data)
- Planned schedule for the project (schedule data)

- Planned completion values for measures (estimates data)
- Models of profile behavior for similar projects (profile models)
- Model of the schedule for similar projects (schedule model)
- Model of completion estimates for similar projects (estimate set model)

Key Steps

1. Let the user select a profile defined for the current measure of interest.
2. Obtain the selected profile's actual and expected values for the current date and its estimated values at completion.
3. Display the distribution of values in each of the profile's defined categories.

3.2.8.1 Select a Profile of Interest

Purpose

Allows the user to select a profile of interest from the list of all available profiles associated with the current measure.

Required Data

- Project/profile availability list

Steps

1. Examine the project/profile availability list to identify all profiles associated with the measure of interest that have data for the current project.
2. Display the list of available profiles and permit the user to select a profile of interest for subsequent examination.
3. Locate the selected profile data and profile models for the project.

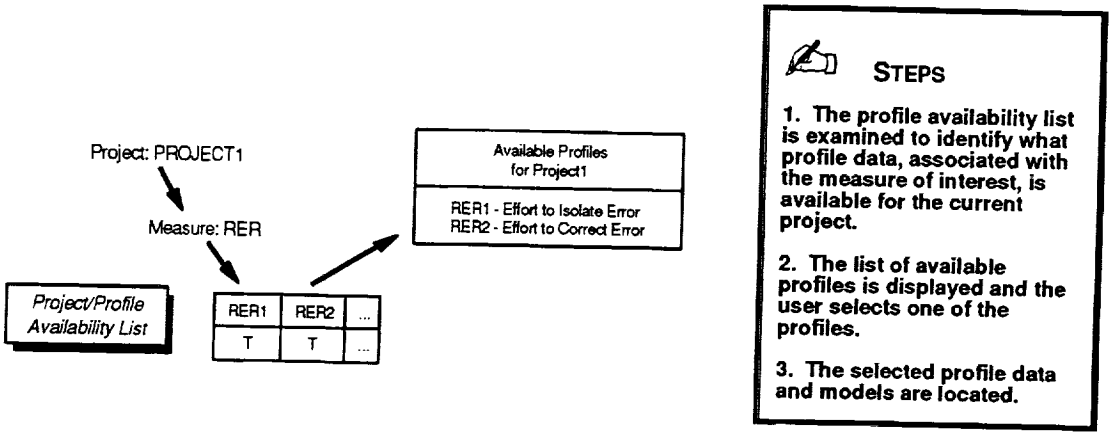


Figure 3-33. Selecting an Available Profile

3.2.8.2 Obtain Actual and Normal Profile Values

Purpose

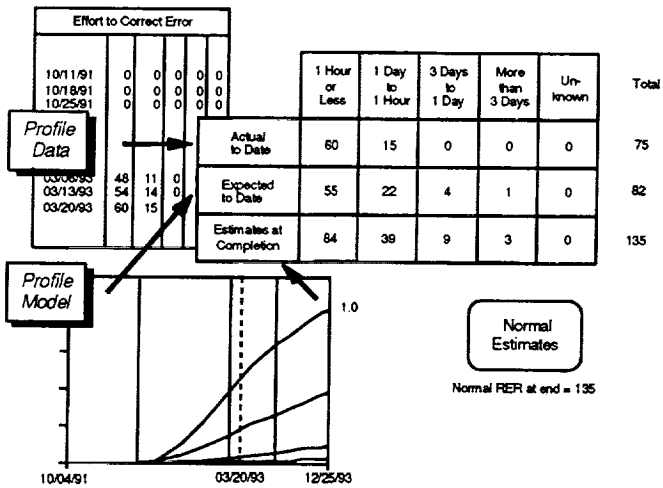
Obtains a profile's actual and expected values for the current date and its estimated values at project completion.

Required Data

- Current project date (input value)
- Profile data (for profile of interest)
- Current schedule
- Current estimates
- Profile model (for profile of interest)
- Schedule model
- Estimate set model

Steps

1. Obtain the actual values observed through the current project date for the profile of interest in each of its defined categories ($Actual\ Profile_{To\ Date}[i]$).
2. Use *Get Project Magnitude* with the current estimates to obtain the measure and estimated completion value for that measure which is most indicative of the project's magnitude.
3. On the basis of that magnitude, use *Determine Normal Estimate Set* with the estimate set model to create a normal set of estimates for the project.
4. Use *Get Estimated Completion Value* with the normal estimates to obtain the normal completion value for the profile's measure ($Normal\ Measure\ Value_{Completion}$).
5. Use *Get Project Dates* with the current schedule to obtain the project start and end date.
6. Given the project start and end dates, use *Convert Date to Phase* with the schedule model to translate the current project date into a phase and fraction of phase.
7. For this phase and fraction of phase, use *Convert Phase to Profile Measure* with the profile model, specifying the normal measure value at completion as an input scaling factor, to determine the expected profile values for the current date ($Expected\ Profile_{To\ Date}[i]$).
8. Given the project start and end dates, use *Convert Date to Phase* with the schedule model to translate the project end date into a phase and fraction of phase.
9. For this phase and fraction of phase, use *Convert Phase to Profile Measure* with the profile model, specifying the normal measure value at completion as an input scaling factor, to determine the estimated profile values at project completion ($Estimated\ Profile_{Completion}[i]$).



STEPS

1. Based on the project's magnitude, a normal set of estimates is generated. From this set of estimates, the completion estimate of 135 errors is obtained.

2. The current project date, 3/20/93, is converted to a phase and fraction of phase.

3. The expected value for each profile component on the current project date is calculated.

4. Using the phase and fraction of phase of the project's end date, the expected completion value of each profile component is calculated.

Figure 3-34. Obtaining Actual and Normal Profile Values

3.3 OVERALL ASSESSMENT

The SME enables the user to view the results of an overall project assessment of high-level quality attributes such as correctability, maintainability, and reliability. The function uses current project data along with algorithms to compute a rating for each quality attribute. The SME compares a project's objective data with models and, based on the comparisons, assigns a relative value to each one in a series of factors. Combinations of these factors are in turn evaluated to produce the attributes' overall relative quality indexes.

Table 3-4 summarizes the major functions supported by overall assessment and identifies each function's purpose.

Table 3-4. Overall Assessment Services Functions

FUNCTION	PURPOSE
Attribute Evaluation	Lets user perform an overall assessment of project quality attributes
Attribute Factor Examination	Lets user investigate the reasons the SME computed a particular attribute rating

The following sections provide additional detailed information on each of these functions.

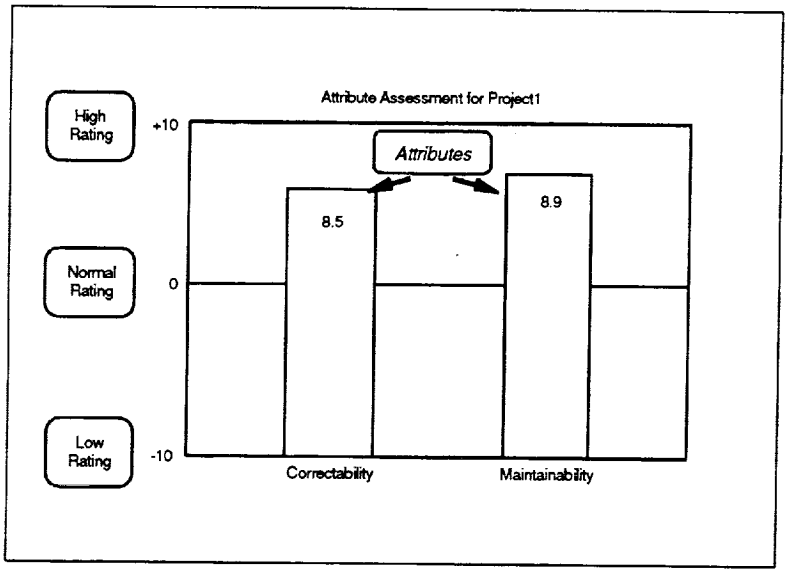
3.3.1 Attribute Evaluation

Purpose

Assigns and displays ratings of quality attributes using objective measurement data collected for the project.

Description

The attribute evaluation function uses current project data along with models and an evaluation algorithm to compute a relative value for each attribute. The values can range from negative to positive, with zero being the normal relative index. The results of the evaluation are depicted graphically as a series of vertical bars, with one bar representing each attribute. Each bar is labeled with the result of the associated attribute's evaluation.



The figure shows a representative project attribute evaluation graph. This example depicts the evaluation of two attributes, *correctability* and *maintainability*. The correctability and maintainability attributes have been evaluated at 8.5 and 8.9, respectively.

Note that the scale in the figure ranges from a low rating of -10 to a high rating of + 10, with zero considered normal.

Figure 3-35. Evaluating Project Attributes

Required Information

- List of attribute and factor definitions (attribute definitions)
- Actual data values for the available measures (measure data)
- Actual data values for the available profiles (profile data)
- Models for the available measures (measure models)
- Models for the available profiles (profile models)

Key Steps

1. Compute the relative values for each attribute.
2. Scale, display, and label the vertical bar graph in the plotting area.

3.3.1.1 Compute Relative Attribute Values

Purpose

Evaluates all defined attributes and computes their relative values.

Required Data

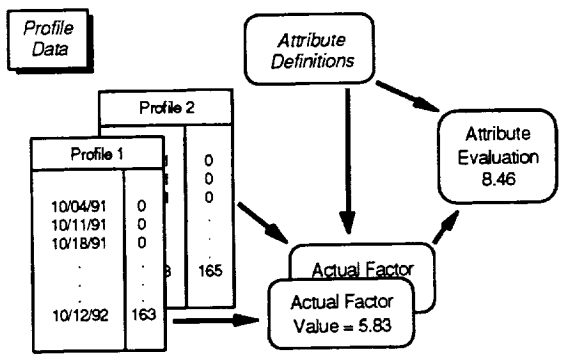
- Attribute definitions (in *Assess Attribute*)
- Measure data (in *Assess Attribute*)
- Profile data (in *Assess Attribute*)
- Measure models (in *Assess Attribute*)
- Profile models (in *Assess Attribute*)

Steps

For each attribute defined in the attribute definitions:

1. Use *Assess Attribute* to calculate a relative rating for the specified project quality attribute.

Note: The algorithm in *Assess Attribute* relies (1) on *Evaluate Actual Factor Value* to evaluate the function defined for any underlying factors using actual project data values and (2) on *Evaluate Expected Factor Values* to evaluate the function defined for any underlying factors using normal model values. The results of these evaluations are combined and scaled to produce a relative rating for each attribute.



STEPS

1. Based on the measure data for the project, the actual cumulative value for the measure through 10/12/92 is 168 errors. Based on the profile data for the project, 163 of these reported errors were isolated in less than 1 day, and 165 of these errors were corrected in less than 1 day.

2. Using information in the attribute definitions, these values are used to produce an attribute evaluation of 8.46 for correctability.

Figure 3-36. Computing Attribute Values

3.3.1.2 Scale and Display Attribute Bar Graph

Purpose

Scales and displays project attribute values and generates the plot axes, labels, and title.

Required Data

- Attribute definitions
- Attribute values (input values)

Steps

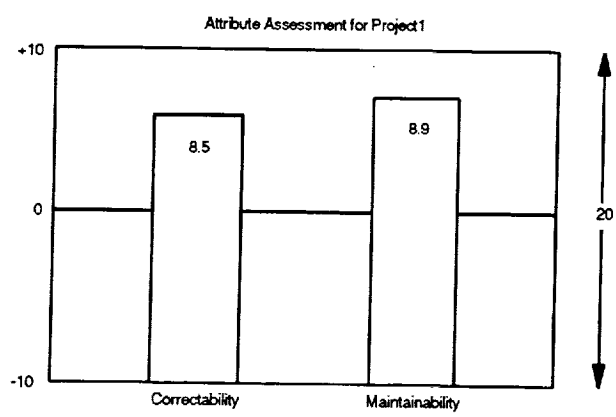
1. Scale the plot's x-axis to the number of attributes, represented by vertical bars, to be displayed for the project.

   $X\text{-}Axis\ Scale = Number\ of\ Bars_{Total}$

2. Set the plot's y-axis on a scale based on the minimum and maximum attribute values, with the average y value considered normal.

   $Y\text{-}Minimum = Bar\ Value_{Minimum}$
   $Y\text{-}Maximum = Bar\ Value_{Maximum}$
   $Y\text{-}Axis\ Range = Bar\ Value_{Maximum} - Bar\ Value_{Minimum}$
   $Normal\ Y\text{-}value = (Bar\ Value_{Maximum} + Bar\ Value_{Minimum}) / 2$

3. Display the basic plotting area with appropriate axes, labels, and title.
4. Display and label vertical bars, and display respective attribute values.



STEPS

1. Based on the information in the attribute definitions, there are two attributes to be displayed.

2. The minimum and maximum values contained in the factor definition list are -10 and +10, respectively. This defines a range of 20, with 0 being normal.

3. The basic plot is displayed on the screen with titles and labels.

4. The vertical bars are displayed on the screen, along with associated attribute values and labels.

Figure 3-37. Displaying a Bar Graph of Attribute Values

3.3.2 Attribute Factor Examination

Purpose

Displays ratings for factors that contribute to a particular attribute evaluation.

Description

The attribute factor examination function generates a vertical bar graph displaying the factors that were analyzed in arriving at the relative index of a given attribute.

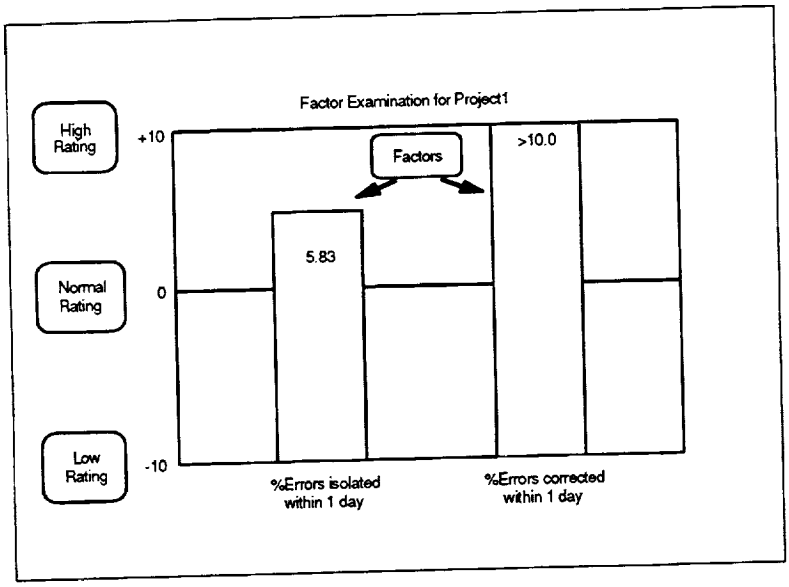


Figure 3-38. Examining Project Attribute Factors

The figure shows a representative project attribute factor graph. This example depicts the display of two factors, percentage of errors isolated in less than 1 day, and percentage of errors corrected in less than 1 day. The factors have been rated at 5.83 and 11.09, respectively.

Note that the scale in the figure ranges from a low rating of -10 to a high rating of +10, with zero considered normal.

Required Information

- List of attribute and factor definitions (attribute definitions)
- Actual data values for the available profiles (profile data)
- Models for the available profiles (profile models)

Key Steps

1. Scale, display, and label vertical bar graph in plotting area.

3.3.2.1 Scale and Display Factor Bar Graph

Purpose

Scales and displays attribute factor values and generates the plot axes, labels, and title.

Required Data

- Attribute definitions
- Attribute factor values (input values)

Steps

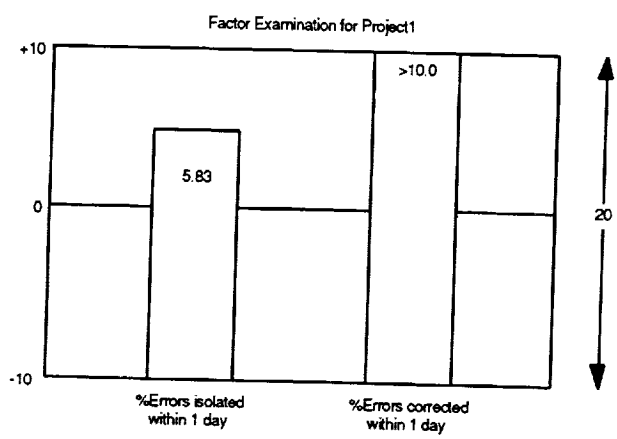
1. Scale the plot's x-axis to the number of factors, represented by vertical bars, to be displayed for the project.

   $X\text{-}Axis\ Scale = Number\ of\ Bars_{Total}$

2. Set the plot's y-axis on a scale based on the minimum and maximum factor values, with the average y value considered normal.

   $Y\text{-}Minimum = Bar\ Value_{Minimum}$
   $Y\text{-}Maximum = Bar\ Value_{Maximum}$
   $Y\text{-}Axis\ Range = Bar\ Value_{Maximum} - Bar\ Value_{Minimum}$
   $Normal\ Y\text{-}value = (Bar\ Value_{Maximum} + Bar\ Value_{Minimum}) / 2$

3. Display the basic plotting area with appropriate axes, labels, and title.
4. Display and label vertical bars, and display respective factor values.



STEPS

1. Based on the information in the attribute definitions, there are two factors to be displayed.

2. The minimum and maximum values contained in the factor definition list are -10 and +10, respectively. This defines a range of 20, with 0 being normal.

3. The basic plot is displayed on the screen with titles and labels.

4. The vertical bars are displayed, along with associated factor values and labels.

Figure 3-39. Displaying a Bar Graph of Factor Values

3.4 PLANNING

The SME enables the user to select, create, and modify alternative plans. An alternative plan consists of a schedule and a set of completion estimates. Alternative plans are created and modified by the user to investigate the effects of changing schedules and estimates. Project plans are used by the monitoring and assessment functions. The user can see the results of using an alternative plan by reexecuting these functions.

Table 3-5 summarizes the major functions supported by the planning feature and identifies each function's purpose.

Table 3-5. Planning Services Functions

FUNCTION	PURPOSE
Use of Alternative Schedules	Lets user modify phase start and end dates
Use of Alternative Estimates	Lets user modify estimated completion values

The following sections provide additional detailed information on each of these functions.

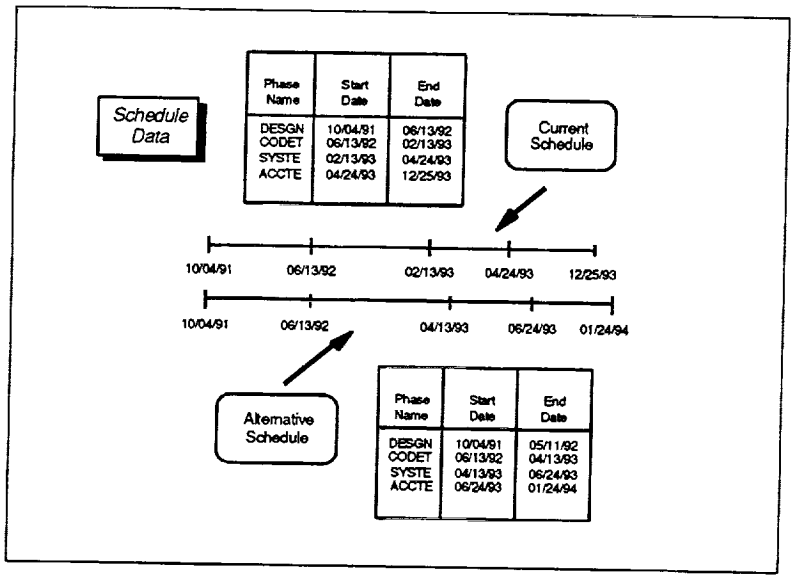
3.4.1 Use of Alternative Schedules

Purpose

Lets the user modify the phase start and end dates specified in the current schedule for use in "what-if" scenarios.

Description

A schedule is a list of serial, non-overlapping phases and their start and end dates. An alternative schedule has the same format and usage, but is created interactively by the user. Creating an alternative schedule enables the user to see the possible effects of changing some aspect of a project's schedule. Once selected, the alternative schedule becomes the current schedule for the project of interest and will be used in subsequent monitor and assessment functions. The SME provides two independent methods for creating these schedules.



The figure depicts updating a project schedule to create an alternative schedule. This example illustrates a case where the end dates of two phases, CODET and SYSTE, have slipped approximately 2 months, and the end of the ACCTE phase has been extended by 1 month.

Such a situation could arise due to problems or to periodic reassessments of the plan. Creating an alternative schedule helps the user investigate the effects of adjusting the schedule.

Figure 3-40. Sample Alternative Schedule

Required Information

- Planned start and end dates of each phase (current schedule)
- Model of the schedule for similar projects (schedule model)

Key Steps

1. Use Method 1 to allow the user to interactively specify end dates for each phase.
2. Use Method 2 to generate dates for each phase based on the schedule model.

Note: To serve as a valid alternative schedule, phase dates must be in chronological order by phase, and the project end date may not fall before the current date of the project. Additionally, the project start date is considered fixed and may not be altered.

Method 1—Entered by the User Interactively

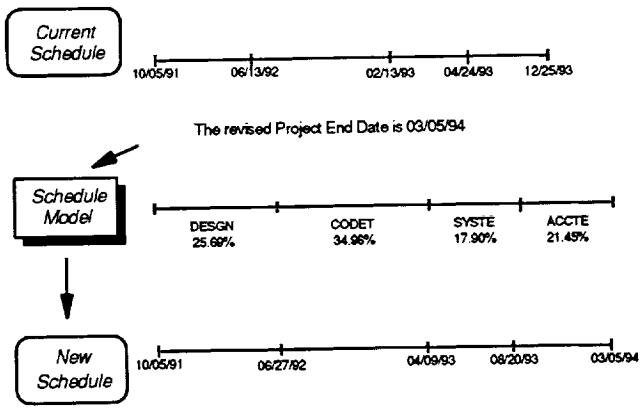
Obtain any new phase end dates interactively from the user and create an alternative schedule as follows:

1. Display the project start date and the end dates of all development life-cycle phases in the current schedule.
2. Allow the user to update the end dates of one or more phases. After the user enters a revised end date, validate and remember the entry.
3. When the user finishes updating the schedule, check all the entries to ensure that the phase dates are in chronological order and that the end date of the last phase does not precede the current project date.

Method 2—Derived from the Schedule Model

Obtain a new project end date for the project from the user and create an alternative schedule as follows:

1. Display the project start date and end date from the current schedule.
2. Allow the user to revise the end date of the project. After the user enters a new completion date, validate the entry and ensure that the date entered does not precede the current project date.
3. Using the original project start date and the new project completion date, use *Determine Normal Schedule* with the schedule model to calculate new phase dates for each life-cycle phase.



STEPS

1. The current end date of the project is 12/25/93.

2. The user enters 03/05/94 as a revised completion date for the project.

3. The schedule model is scaled based on the revised project duration to produce a new schedule.

4. The new end dates for the DESGN, CODET, SYSTE, and ACCTE phases are 6/27/92, 4/09/93, 8/20/93, and 3/05/94, respectively.

Figure 3-41. Creating a Schedule Based on a Model

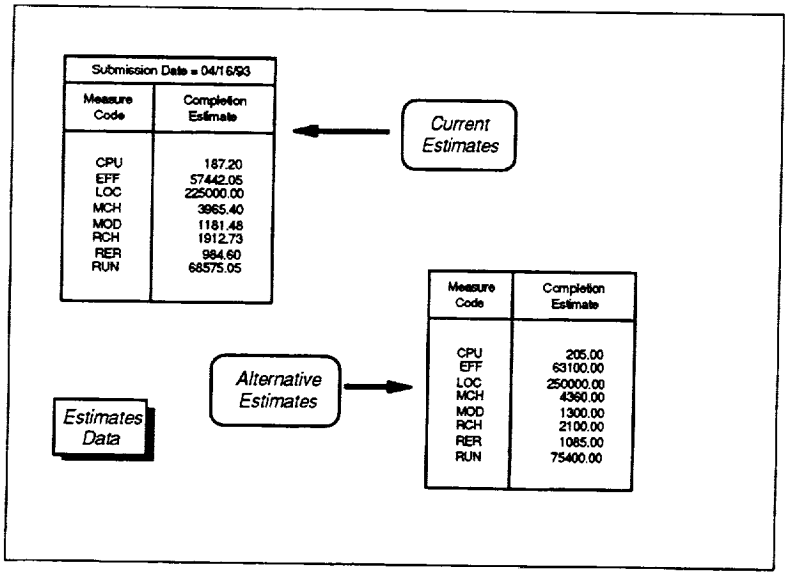
3.4.2 Use of Alternative Estimates

Purpose

Lets the user modify the estimated completion values of one or more measures for use in "what-if" scenarios.

Description

Completion estimates are a set of expected measure data values at project completion. Alternative estimates have the same format and usage, but are created interactively by the user. Creating a set of alternative estimates enables the user to see the possible results of changing any of the project completion estimates. Once selected, the alternative estimates become the current estimates and will be used in subsequent monitor and assessment functions. The SME provides two independent methods for creating these estimates.



The figure depicts updating a set of project estimates to create alternative estimates. This example illustrates a case where all estimated completion values have been revised upward by a factor of about 10% over their original values.

Such a situation could arise due to growth or to periodic reestimation of targeted completion values. Creating alternative estimates helps the user investigate the effects of changing one or more project completion estimate(s).

Figure 3-42. Sample Alternative Estimates

Required Information

- Planned completion value for each project measure (current estimates)
- Model of estimates for similar projects (estimate set model)

Key Steps

1. Use Method 1 to allow the user to interactively specify estimated completion values for each measure.
2. Use Method 2 to generate completion estimates for each measure based on the estimate set model.

Note: To serve as a valid set of alternative estimates, each estimated completion value must be a non-negative numeric value.

Method 1—Entered by the User Interactively

Obtain any new estimated completion values from the user and create a set of alternative estimates as follows:

1. Display the estimated completion values of all measures in the set of current estimates.
2. Allow the user to update the estimates for one or more measures. After the user enters a revised completion estimate, validate the entry to ensure that the value is numeric and non-negative.

Method 2—Derived from the Estimate Set Model

Obtain a new estimated completion value for one of the measures from the user and create an alternative estimate set as follows:

1. Display the estimated completion values of all measures in the set of current estimates.
2. Allow the user to choose one of the measures and to supply a new estimated completion value for that measure. After the user enters a new estimate, ensure that the value is numeric and non-negative.
3. For the chosen measure and new estimated completion value, use *Determine Normal Estimate Set* to scale the estimate set model and calculate new estimated completion values for each project measure.

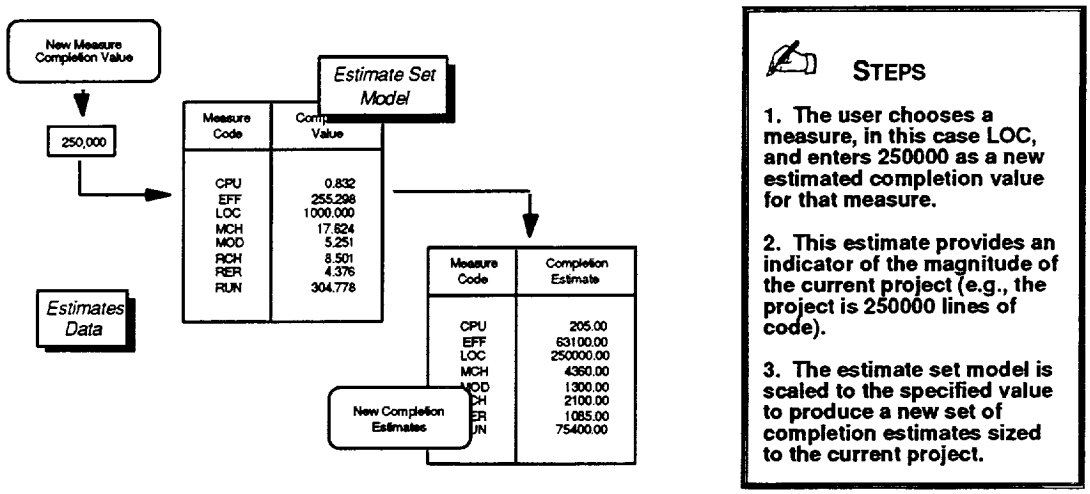


Figure 3-43. Creating an Estimate Set Based on a Model

APPENDIX A—LIST OF DEFINED SERVICES

This appendix provides an alphabetic listing (Table A-1) of all general-purpose and function-specific services defined and referenced in the document. The list can facilitate locating where a specific service is described in this document when only its name is known.

Table A-1. Cross Reference of Defined Services

NAME OF SERVICE	SECTION	COMPONENT/FUNCTION
Analyze Trends for a Ratio of Two Measures	3.2.7	Trend Analysis
Analyze Trends for a Single Measure of Interest	3.2.7	Trend Analysis
Assess Attribute	2.2.5	Attribute Definitions
Compute Relative Attribute Values	3.3.1	Attribute Evaluation
Convert Date to Phase	2.2.1	Schedule Models
Convert Measure to Phase	2.2.2	Measure Models
Convert Phase to Date	2.2.1	Schedule Models
Convert Phase to Measure	2.2.2	Measure Models
Convert Phase to Profile Measure	2.2.3	Profile Models
Determine Normal Estimate Set	2.2.4	Estimate Set Models
Determine Normal Measure Guidelines	2.2.2	Measure Models
Determine Normal Schedule	2.2.1	Schedule Models
Determine Phase for Rules	2.3.2	Rule Base
Determine Rate for Rules	2.3.2	Rule Base
Evaluate Actual Factor Value	2.2.5	Attribute Definitions
Evaluate Expected Factor Values	2.2.5	Attribute Definitions
Evaluate Reason	2.3.1	Knowledge Base
Evaluate Rule	2.3.2	Rule Base
Generate Rate Model	2.2.2	Measure Models
Get Estimated Completion Value	2.1.9	Estimates Data
Get Estimates	2.1.9	Estimates Data
Get Project Dates	2.1.6	Schedule Data
Get Project Magnitude	2.2.4	Estimate Set Models
Get Schedule	2.1.6	Schedule Data
Get Scheduled Phase Dates	2.1.6	Schedule Data
Get Ratio of Estimates	2.2.4	Estimate Set Models
Identify Models to Use for Project	3.1.1	Project Selection
Obtain Actual and Normal Profile Values	3.2.8	Profile Analysis
Obtain a Phase Estimate	3.2.6	Prediction
Predict Completion Date of Project	3.2.6	Prediction
Predict Measure Value at Completion	3.2.6	Prediction
Predict Intermediate Values Through Completion	3.2.6	Prediction
Plot a Comparison Project for a Measure	3.2.5	Comparison to Other Projects
Plot Actual Data for a Measure	3.2.2	Simple Observation
Plot Actual Data for Current Project	3.2.5	Comparison to Other Projects
Plot Normal Project Data for a Measure	3.2.3	Comparison to a Normal Project
Plot Planned Project Data for a Measure	3.2.4	Comparison to Manager's Plan
Rate Dependent Factor	2.3.1	Knowledge Base
Rate Objective Factor	2.3.1	Knowledge Base
Rate Subjective Factor	2.3.1	Knowledge Base
Scale and Display Plot Area for Observation	3.2.2	Simple Observation
Scale and Display Plot Area for Comparison to Normal	3.2.3	Comparison to a Normal Project
Scale and Display Plot Area for Comparison to Other Project	3.2.5	Comparison to Other Projects
Scale and Display Plot Area for Comparison to Plan	3.2.4	Comparison to Manager's Plan
Scale and Display Attribute Bar Graph	3.3.1	Attribute Evaluation
Scale and Display Factor Bar Graph	3.3.2	Attribute Factor Examination
Select a Comparison Project	3.2.5	Comparison to Other Projects
Select a Profile of Interest	3.2.8	Profile Analysis
Select a Project of Interest	3.1.1	Project Selection
Set Current Plan for Project	3.1.1	Project Selection

ABBREVIATIONS AND ACRONYMS

AGSS	attitude ground support system
CDR	critical design review
CPU	computer hours
CRF	change report form
EFF	total staff hours
FDD	Flight Dynamics Division
GSFC	Goddard Space Flight Center
LOC	lines of code
MCH	modules changed
MOD	module count
NASA	National Aeronautics and Space Administration
PEF	project estimates form
PRF	personnel resources form
RCH	reported changes
RER	reported errors
RID	review item disposition
RUN	computer runs
SEL	Software Engineering Laboratory
SLOC	source lines of code
SME	Software Management Environment
SPF	services/products form
TBD	to be determined

REFERENCES

1. SEL 89-103, *Software Management Environment (SME) Concepts and Architecture (Revision 1)*, R. Hendrick, D. Kistler, and J. Valett, September 1992

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

3. SEL 92-002, *Data Collection Procedures for the Software Engineering Laboratory (SEL) Database*, G. Heller, J. Valett, and M. Wild, March 1992

4. J. Valett and A. Raskin, "DEASEL: An Expert System for Software Engineering," *Proceedings of the Tenth Annual Software Engineering Workshop*, SEL-85-006, December 1985

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

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

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-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, W. A. Taylor, et al., July 1986

SEL-79-002, *The Software Engineering Laboratory: Relationship Equations*, K. Freburger and V. R. Basili, May 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-80-005, *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

10000229
0207/1994

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

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 of 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-81-110, *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-81-305, *Recommended Approach to Software Development*, L. Landis, S. Waligora, F. E. McGarry, et al., June 1992

SEL-81-305SP1, *Ada Developers' Supplement to the Recommended Approach*, R. Kester and L. Landis, November 1993

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 of 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, M. G. Rohleder, and F. E. McGarry, October 1983

SEL-82-1106, *Annotated Bibliography of Software Engineering Laboratory Literature*, L. Morusiewicz and J. Valett, November 1992

10000229
0207/1994

SEL-82-1206, *Annotated Bibliography of Software Engineering Laboratory Literature*, L. Morusiewicz and J. Valett, November 1993

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 of 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-84-003, *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 of the Ninth Annual Software Engineering Workshop*, November 1984

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

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., and V. R. Basili, May 1985

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

SEL-85-006, *Proceedings of 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

10000229
0207/1994

SEL-86-003, *Flight Dynamics System Software Development Environment (FDS/SDE) 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 et al., November 1986

SEL-86-006, *Proceedings of 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-87-009, *Collected Software Engineering Papers: Volume V*, November 1987

SEL-87-010, *Proceedings of 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-89-004, *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

10000229
0207/1994

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-89-103, *Software Management Environment (SME) Concepts and Architecture (Revision 1)*, R. Hendrick, D. Kistler, and J. Valett, September 1992

SEL-89-301, *Software Engineering Laborary (SEL) Database Organization and User's Guide (Revision 3)*, L. Morusiewicz, December 1993

SEL-90-001, *Database Access Manager for the Software Engineering Laboratory (DAMSEL) User's Guide*, M. Buhler, K. Pumphrey, and D. Spiegel, 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-90-006, *Proceedings of the Fifteenth Annual Software Engineering Workshop*, November 1990

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

SEL-91-003, *Software Engineering Laboratory (SEL) Ada Performance Study Report*, E. W. Booth and M. E. Stark, July 1991

SEL-91-004, *Software Engineering Laboratory (SEL) Cleanroom Process Model*, S. Green, November 1991

SEL-91-005, *Collected Software Engineering Papers: Volume IX*, November 1991

SEL-91-006, *Proceedings of the Sixteenth Annual Software Engineering Workshop*, December 1991

SEL-91-102, *Software Engineering Laboratory (SEL) Data and Information Policy (Revision 1)*, F. McGarry, August 1991

SEL-92-001, *Software Management Environment (SME) Installation Guide*, D. Kistler and K. Jeletic, January 1992

SEL-92-002, *Data Collection Procedures for the Software Engineering Laboratory (SEL) Database*, G. Heller, J. Valett, and M. Wild, March 1992

10000229
0207/1994

SEL-92-003, *Collected Software Engineering Papers: Volume X*, November 1992

SEL-92-004, *Proceedings of the Seventeenth Annual Software Engineering Workshop*, December 1992

SEL-93-001, *Collected Software Engineering Papers: Volume XI*, November 1993

SEL-93-002, *Cost and Schedule Estimation Study Report*, S. Condon, M. Regardie, M. Stark, et al., November 1993

SEL-94-001, *Software Management Environment (SME) Components and Algorithms*, R. Hendrick, D. Kistler, and J. Valett, February 1994

SEL–RELATED LITERATURE

[10]Abd-El-Hafiz, S. K., V. R. Basili, and G. Caldiera, "Towards Automated Support for Extraction of Reusable Components," *Proceedings of the IEEE Conference on Software Maintenance-1991 (CSM 91)*, October 1991

[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

[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

[10]Bailey, J. W., and V. R. Basili, "The Software-Cycle Model for Re-Engineering and Reuse," *Proceedings of the ACM Tri-Ada 91 Conference*, October 1991

[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

10000229
0207/1994

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

[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

[9]Basili, V. R., G. Caldiera, and G. Cantone, "A Reference Architecture for the Component Factory,"*ACM Transactions on Software Engineering and Methodology,* January 1992

[10]Basili, V., G. Caldiera, F. McGarry, et al., "The Software Engineering Laboratory—An Operational Software Experience Factory," *Proceedings of the Fourteenth International Conference on Software Engineering (ICSE 92)*, May 1992

[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

[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 J. Ramsey, *Structural Coverage of Functional Testing*, University of Maryland, Technical Report TR-1442, September 1984

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

10000229
0207/1994

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

[5]Basili, V. R., 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

[9]Basili, V. R., and H. D. Rombach, "Support for Comprehensive Reuse," *Software Engineering Journal*, September 1991

[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, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Transactions on Software Engineering*, December 1987

[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

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

[9]Basili, V. R., and R. W. Selby, "Paradigms for Experimentation and Empirical Studies in Software Engineering," *Reliability Engineering and System Safety*, January 1991

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

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

[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

10000229
0207/1994

[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

[9]Booth, E. W., and M. E. Stark, "Designing Configurable Software: COMPASS Implementation Concepts," *Proceedings of Tri-Ada 1991*, October 1991

[10]Booth, E. W., and M. E. Stark, "Software Engineering Laboratory Ada Performance Study—Results and Implications," *Proceedings of the Fourth Annual NASA Ada User's Symposium*, April 1992

[10]Briand, L. C., and V. R. Basili, "A Classification Procedure for the Effective Management of Changes During the Maintenance Process," *Proceedings of the 1992 IEEE Conference on Software Maintenance (CSM 92)*, November 1992

[10]Briand, L. C., V. R. Basili, and C. J. Hetmanski, "Providing an Empirical Basis for Optimizing the Verification and Testing Phases of Software Development," *Proceedings of the Third IEEE International Symposium on Software Reliability Engineering (ISSRE 92)*, October 1992

[11]Briand, L. C., V. R. Basili, and C. J. Hetmanski, *Developing Interpretable Models with Optimized Set Reduction for Identifying High Risk Software Components*, TR-3048, University of Maryland, Technical Report, March 1993

[9]Briand, L. C., V. R. Basili, and W. M. Thomas, *A Pattern Recognition Approach for Software Engineering Data Analysis*, University of Maryland, Technical Report TR-2672, May 1991

[11]Briand, L. C., S. Morasca, and V. R. Basili, "Measuring and Assessing Maintainability at the End of High Level Design," *Proceedings of the 1993 IEEE Conference on Software Maintenance (CSM 93)*, November 1993

[11]Briand, L. C., W. M. Thomas, and C. J. Hetmanski, "Modeling and Managing Risk Early in Software Development," *Proceedings of the Fifteenth International Conference on Software Engineering (ICSE 93)*, May 1993

10000229
0207/1994

[5]Brophy, C. E., W. W. Agresti, and V. R. 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. N., and W. W. Agresti, "Resolving the Software Science Anomaly," *Journal of Systems and Software*, 1987

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

[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., 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

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

[5]Card, D. N., F. E. McGarry, and G. T. 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

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

[5]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

[11]Li, N. R., and M. V. Zelkowitz, "An Information Model for Use in Software Management Estimation and Prediction," *Proceedings of the Second International Conference on Information Knowledge Management*, November 1993

[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. E., and W. 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

[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. L., and V. R. Basili, "An Evaluation of Expert Systems for Software Engineering Management," *IEEE Transactions on Software Engineering*, June 1989

[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

10000229
0207/1994

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

[9]Rombach, H. D., "Software Reuse: A Key to the Maintenance Problem," *Butterworth Journal of Information and Software Technology*, January/February 1991

[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

[10]Rombach, H. D., B. T. Ulery, and J. D. Valett, "Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL," *Journal of Systems and Software*, May 1992

[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

[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

[9]Seidewitz, E., "Object-Oriented Programming Through Type Extension in Ada 9X," *Ada Letters*, March/April 1991

[10]Seidewitz, E., "Object-Oriented Programming With Mixins in Ada," *Ada Letters*, March/April 1992

[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

[9]Seidewitz, E., and M. Stark, "An Object-Oriented Approach to Parameterized Software in Ada," *Proceedings of the Eighth Washington Ada Symposium*, June 1991

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

10000229
0207/1994

[11]Stark, M., "Impacts of Object-Oriented Technologies: Seven Years of SEL Studies," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, September 1993*

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

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

[10]Straub, P. A., and M. V. Zelkowitz, "On the Nature of Bias and Defects in the Software Specification Process," *Proceedings of the Sixteenth International Computer Software and Applications Conference (COMPSAC 92)*, September 1992

[8]Straub, P. A., and M. V. 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

[10]Tian, J., A. Porter, and M. V. Zelkowitz, "An Improved Classification Tree Analysis of High Cost Modules Based Upon an Axiomatic Definition of Complexity," *Proceedings of the Third IEEE International Symposium on Software Reliability Engineering (ISSRE 92)*, October 1992

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

[10]Valett, J. D., "Automated Support for Experience-Based Software Management," *Proceedings of the Second Irvine Software Symposium (ISS '92)*, March 1992

[5]Valett, J. D., and F. E. 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. R. 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

10000229
0207/1994

[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: Experiences With Syntax Editors," *Information and Software Technology*, April 1990

10000229
0207/1994

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.

[9]This article also appears in SEL-91-005, *Collected Software Engineering Papers: Volume IX*, November 1991.

[10]This article also appears in SEL-92-003, *Collected Software Engineering Papers: Volume X*, November 1992.

[11]This article also appears in SEL-93-001, *Collected Software Engineering Papers: Volume XI*, November 1993.

10000229
0207/1994

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
February 1994

3. REPORT TYPE AND DATES COVERED
Contractor Report

4. TITLE AND SUBTITLE

Software Management Environment (SME)
Components and Algorithms

5. FUNDING NUMBERS

552

6. AUTHOR(S)

Software Engineering Laboratory

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Software Engineering Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland

8. PERFORMING ORGANIZATION REPORT NUMBER

SEL-94-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, D.C. 20546–0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

CR-189346

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified–Unlimited
Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document presents the components and algorithms of the Software Management Environment (SME), a management tool developed for the Software Engineering Branch (Code 552) of the Flight Dynamics Division (FDD) of the Goddard Space Flight Center (GSFC). The SME provides an integrated set of visually oriented experienced-based tools that can assist software development managers in managing and planning software development projects. This document describes and illustrates the analysis functions that underlie the SME's project monitoring, estimation, and planning tools. SME Components and Algorithms is a companion reference to *SME Concepts and Architecture,* and *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules.*

14. SUBJECT TERMS

15. NUMBER OF PAGES
226

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT
Unlimited