TOWARDS UNDERSTANDING SOFTWARE – 15 YEARS IN THE SEL

Frank McGarry
Rose Pajerski
GODDARD SPACE FLIGHT CENTER

ABSTRACT

For 15 years, the Software Engineering Laboratory (SEL) at NASA/Goddard Space Flight Center (GSFC) has been carrying out studies and experiments for the purpose of understanding, assessing, and improving software, and software processes within a production software environment. The SEL comprises three major organizations:

• NASA/GSFC	Flight Dynamics Division
• University of Maryland	Computer Science Department
• Computer Sciences Corporation	Flight Dynamics Technology Group

These organizations have jointly carried out several hundred software studies, producing hundreds of reports, papers, and documents [Reference 1]—all describing some aspect of the software engineering technology that has undergone analysis in the Flight Dynamics environment. The studies range from small controlled experiments (such as analyzing the effectiveness of code reading versus functional testing) to large, multiple-project studies (such as assessing the impacts of Ada on a production environment). This paper will summarize the key findings that the sponsoring organization (NASA) feels have laid the foundation for ongoing and future software development and research activities.

I. BACKGROUND

In 1976, NASA/GSFC initiated an effort to carry out experiments within the Flight Dynamics area to attempt to measure the relative merits of the numerous software technologies that were both available and claimed to be significant 'improvements' over currently used practices. Although significant advances were being made in developing new technologies, such as structured development practices, automated tools, quality assurance approaches, and management tools, there was very limited empirical evidence or guidance pertaining to applying these promising, yet immature methods. Primarily to address this situation, the Software Engineering Laboratory (SEL) was formed.

The SEL was formed as a partnership between NASA, the University of Maryland, and Computer Sciences Corporation. This working relationship has been maintained continually since 1976 with relatively little change to the overall goals of the organization during its entire history. In general, the goals have matured; they have not been changed. The goals can be itemized as follows:

1. **Understand**—Improve the insight that exists in characterizing the software process and its products in a production environment,

2. **Assess**—Measure the impact that available techniques have on the software process. Determine which techniques are appropriate for the environment and can improve the software,

3. **Infuse**—After identifying process improvements, package the technology in a form to be applied and useful to the production organization.

The approach taken to attain these three generalized goals has been to apply potentially beneficial techniques to the development of production software and to measure the process and product in reasonable detail to assess quantifiably the applied technology. Measures of concern, such as cost, reliability, and/or maintainability, are defined as the organization determines the major near- and long-term objectives for its software development process improvement program. Once those objectives are determined, the SEL staff designs the experiment, that is, defines the particular data to be captured and the questions that must be addressed in each experimental project.

All of the experiments conducted by the SEL have occurred within the production environment of the Flight Dynamics software development facility at NASA/GSFC. This software can be characterized as scientific, nonembedded, relatively complex software. Projects are typically developed in FORTRAN, although about 25 percent of the projects utilize another language such as Ada, C, or PASCAL. The duration of each effort normally runs from 2 to 3-1/2 years. with an average staff size of approximately 15 software developers. The average size of one of these projects runs approximately 175,000 source lines of code (counting commentary), with about 25 percent reused from previous development efforts. Since this environment is relatively consistent, it is conducive to the experimentation process. In the SEL, there exists a homogeneous class of software, a stable development environment, and a very controlled, consistent management and development process.

The following three major functional organizations support the experimentation and study within the SEL environment:

1. Software developers, who are responsible for producing the flight dynamics application software.

2. Software engineering analysts, who are the researchers responsible for carrying out the experimentation process and producing study results

3. Data base support staff, who are responsible for collecting, checking and archiving all of the data and information collected from the development efforts.

Since its inception in 1976, the SEL has carried out studies and experiments involving nearly 100 flight dynamics projects. Detailed data have been collected and studied, and numerous reports and journal papers have been produced. From all of this analysis and from all of these studies, seven key points have been identified that reflect insight gained by the SEL and which are the guiding principles for future development and research within this organization. These seven key points, described under EXPERIENCES below, address nearly every aspect of the activities within the SEL experimentation process and should provide some guidance to other organizations involved with software development and/or software engineering research.

II. EXPERIENCES

Point 1: Measurement is an Essential Element of Software Process Improvement

It is imperative that software measurement be an integral component of any software process assessment or process improvement program. Although this point may seem obvious to most, there is evidence that occasionally organizations may initiate 'process improvement' efforts without fully developing considerations and plans for applying measurement. In addition to providing some mechanism for determining the baseline characteristics of the software process before any change is adopted, the measurement aspect is necessary to gauge the impact of any change to the software process. Not only does an organization need to understand if and by how much software 'quality' is improving through enhanced software processes, but even the more elementary assessment as to whether there is any consistency within an organization's process (is it measurable) as well as 'is change observable?' are points addressed only through measurement.

The SEL has focused on software measurement as a tool to aid in determining the effect that changes to the software process may have on attributes of concern (cost, quality, reliability, ...). In addition to this, it has become evident that both the measurement process as well as the measurements themselves are extremely valuable software management tools. The Flight Dynamics environment has adopted the SEL measurement process as an integral component of the development standards and applies key measures in planning and tracking the progress of projects. There are eight key measures that the Flight Dynamics software organization has adopted as essential management aids [Reference 2].

One additional point that has become apparent for the SEL after 15 years of software measurement is that the adoption of an effective measurement program is not cost prohibitive. In fact, the measurement collection process can be essentially a zero cost impact to the development project—provided that a well thought-out set of measures is adopted rather than an ill-conceived large number of measures. The most significant cost attributable to a measurement program is that of processing and effectively analyzing/utilizing the information—and this must be done. It should be obvious that the mere process of collecting measures will be of absolutely no value (even negative value) unless the information is analyzed and factored back into the software process itself. Although the cost impact to the development projects themselves can be near 0-percent overhead, the cost of processing and analyzing information as part of an effective process improvement program will add 10 percent to 15 percent of the development cost.

Point 2: Many Diversions Exist to a Successful Process Improvement Program

Most software organizations have either attempted or at least seriously considered adopting a software measurement program. Unfortunately, there are too few examples of projects or companies in general that have sustained an effective measurement program. Many reasons exist to explain why such a critical element of software engineering consistently fails, and the SEL has experienced most of the significant impediments and pit-falls that can discourage the use of measurement programs. Three of the most significant diversions that the SEL has experienced and which seem to plague numerous other software organizations include the following:

1. **Excessive planning/replanning**—If someone is serious about starting a measurement program, it is more important to get started with a very small effort as opposed to developing the full set of measures, tools, analysis approaches, etc. The key is to start small and grow with experience—but at least start.

2. **Over-Dependence on Statistical Analysis**—Although the use of analysis tools is certainly required in applying measurement to the software process, there are occasions when the analysts attempt to uncover more information from available data (measures) than is reasonable. Intuition is an excellent starting point for the analysis process, and it is certainly enhanced or challenged by statistical information; but there is danger in assuming that the mathematical interpretation of some quite inexact figures can lead to a more accurate conclusion than the figures dictate. Too often, the common sense of experienced software developers and managers is ignored in favor of statistics produced with possibly flawed, misinterpreted, or missing data.

3. **Looking Under the Lamp-Post**—As in the case of the person who has lost a coin in the dark part of a street, but chooses to search for it beneath the lighted lamp-post because it is easier to see, software engineers occasionally address those software topics that are easiest to study as opposed to those that are the real problems for software development/management. There has been significant effort put into studying, rebuilding, and modifying such tools as code analyzers, auditors, converters, graphical design aids, etc., when there is doubt as to the real driving need for such small modifications to very old and well-understood technology. Excessive studies continue to be conducted on antiquated complexity metrics and on 15-year-old cost-modeling techniques, when there are extremely difficult areas to be addressed.

such as design measures, software specification tools and analyzers, and integrated environments.

Point 3: People Are the Most Important Resource/Technology

In reviewing the results of the numerous studies and experiments that the SEL has conducted over the past 15 years, it is apparent that the most effective technologies, that result in the most significant benefit, are those that leverage the skills of the software developers themselves. Numerous studies outside of the SEL environment have shown that the productivity of individuals can easily vary by as much as a factor of 10 to 1. In addition to this fact, SEL studies have indicated that those methods and tools that emphasize human discipline are far more effective than those that merely attempt to take work away from the developers.

Such software techniques as code reading, inspections, walk-throughs, and all aspects of 'Cleanroom' are examples that have been shown to be extremely effective [Reference 3]. All of these are directed toward maximizing the potential of individuals as opposed to removing the individual from the process.

Point 4: Environmental Characteristics Should Dictate Selected Software Engineering Techniques

Experiences in the SEL have verified the expectation that standards, methods, and, in general, all software engineering approaches must be tailored to specific environments. Although the point seems to be obvious, we as practitioners and software engineers often attempt to apply a new technique or method expecting certain improvements without first analyzing whether the methodology is addressing the needs of the environment. For example, if a development organization historically produces highly reliable, well-tested software, then there is probably little benefit to be derived from modifying the testing approach by applying an automated test generator.

Additionally, it must be understood that all software environments evolve with time and undergo some level of change. Because of this, the overall process must be continually observed to identify changing and evolving practices in order to respond with the most appropriate modifications to methods, tools, etc.

Point 5: Automation is an Instrument of Process Improvement, Not a Replacement for Process Understanding

As was mentioned previously, the foundation of the process improvement paradigm is that of understanding the software process and associated products— which may then lead to assessment and to process improvement. Automated tools may provide some help in understanding this process, but too often we expect the automation process to resolve problems that we don't clearly understand in a manual sense. If a software developer or manager cannot clearly represent and grasp some process manually, the application of a software tool will only make the process less understood and more ill defined. This overreliance on automation is occasionally exemplified by organizations that move too swiftly in the adoption of CASE (or related technology) before the overall development characteristics are analyzed and the need for automated tools is identified. Another example can be seen in the attempts of managers to use code analyzers, auditors, and automated complexity analyzers to gain insight into 'complexity' without being able to discern this trait in any of the products or processes.

Although it is unwise to try to automate immature processes or to apply tools where no tool is needed, there are excellent examples of tools and overall automation that reflect significant advances in applying this technology to recently maturing disciplines. Such an example is the recent development of the 'Software Management Environment (SME)' [Reference 4] which is used by the Flight Dynamics Division at NASA/GSFC to automate the use and interpretation of historical software data, models, measures, and intuition toward the management of active software projects.

Point 6: Heritage of the Environment Will Strongly Influence the Software Process

It seems rather obvious to say that a development environment has its own characteristics of process and process improvement and that the heritage of this environment will certainly influence the development of project after project, but the level to which the past performance of a software organization dominates even the use of significantly different technology is quite surprising. It is the most prevalent influence that the SEL has seen in its environment where evolving, new technology is continually applied to observe impacts to the software process, and major changes to methodology are continually made.

For example, the technology impact from the introduction of Ada into the SEL environment has been under study since 1985 when the first Ada system was developed. One of the early expectations was that there would be a significant change to the effort distribution over the implementation (design, code, test) period for these Ada systems in comparison with previous FORTRAN systems. To date, this has not been observed in the SEL—effort distributions based on these activities have remained essentially the same and continue to reflect past SEL experience. Since changes to an established development process occur slowly, the changes themselves tend to evolve over time as more experience is gained with the new technology. As expected, the use of various Ada constructs (generics, packages, typing, tasking) in the more recent Ada projects is considerably different than in earlier systems.

Point 7: Software Can Be Measurably Improved Through Appropriate Use of Available Technologies

Possibly the most important point evinced as a result of the 15 years of study within the SEL is that software (both the process and products) can be quantifiably improved through the selected application of methods, tools, and models that exist today. It has often been argued that since 'the human being' is the dominant factor in any software project, the modification or application of any approach to the development process cannot be observed nor can it have any significant impact on improving measures of importance.

Experience has verified the fact that researchers often attempt to apply and measure, to extremely detailed levels, techniques that may not be 'measurable'; however, it has also shown that overall trends are definitely measurable when the measurement process becomes an integral part of the applied methodology. As was described previously, because a specific software technology may not be applicable to all environments, each environment must clearly define its goals, strengths, and weaknesses before it attempts to observe positive impacts from some modified approach.

There are specific methodologies that the SEL has applied and measured over a long period of time and that have been verified as having positive impact on the cost, reliability, and overall quality of software within the Flight Dynamics environment. Such techniques include 'Reading' (as applied to design, code, and test), Ada, object-oriented development, design criteria (e.g., strength), measurement, and many others. There are software practices that will significantly and measurably improve the software within any specific environment.

III. OVERALL COST/IMPACT OF THE MEASUREMENT EFFORTS IN THE SEL

For 15 years, NASA has been funding these efforts to carry out experiments and studies within the SEL. There has been significant cost and general overhead to this effort, and a logical question that is asked is 'Has it all been worth it?' The answer is a resounding YES. Not only has the expenditure of resources been a wise investment for the Flight Dynamics area within NASA, but members of the SEL strongly believe that such efforts should be commonplace throughout the Agency as well as throughout the software community. The benefits far outweigh the cost.

Since the SEL's inception in 1976, NASA has spent approximately $14 million dollars in the three major support areas required by this type of study environment: research (such as defining studies and analyzing results), technology transfer (such as producing standards and policies), and data processing (such as collecting forms and maintaining data bases). Additionally, approximately 50 staff-years of NASA personnel effort has been expended on the SEL. During this same time period, the Flight Dynamics area has spent approximately $130 million on building operational software, all of which has been part of the study process to some degree.

During the past 15 years, the SEL has certainly had significant impact on the software being developed in the local environment, and there is strong reason to believe that many of the results and studies of the SEL have had favorable impact on a domain broader than just the NASA Flight Dynamics area. Examples of the changes that have been observed include the following:

1. The 'manageability' of software has improved dramatically. In the late 1970s and early 1980s, this environment experienced wide variation from project to

project in productivity, reliability, and quality. Today, however, the SEL has excellent models of the process; has well-defined methods; and is able to predict, control, and manage the cost and quality of the software being produced.

2. The cost per line of new code has decreased somewhat (about 10 percent), and at first glance this may imply that the SEL has failed at improving productivity. Although the SEL finds that the cost to produce a new source statement is nearly as high as it was 14 years ago, there is appreciable improvement in the functionality of the software, as well as tremendous increases in the complexity of the problems being addressed. Also, there has been an appreciable increase in the reuse of software (code, design, methods, test data, etc.), which has driven the overall cost of the equivalent functionality down significantly. When we merely measure the cost to produce one new source statement, the improvement is small; but when we measure overall cost and productivity, the improvement is significant.

3. Reliability of the software has improved by 35 percent. As measured by the number of errors per thousand lines of code (E/KSLOC), the Flight Dynamics software has improved from an average of 8.4 E/KSLOC in the early 1980s to approximately 5.3 E/KSLOC today. These figures cover the software phases up through and including acceptance testing (beginning of operations). Although the operational and maintenance data are not nearly so extensive as the development data, the small amount of data available indicates significant improvement in that area as well.

4. Other measures include the effort put forth in rework (changing, fixing, etc.) and in overall software reuse. These measures also indicate a significant improvement to the software within this one environment.

In addition to the common measures of software (cost, reliability, etc.), there are many other major benefits derived from such a 'measurement' program as that in the SEL. Not only has our understanding of software significantly improved within the research community, but this understanding is apparent throughout the entire development community within this Flight Dynamics environment. Not only have the researchers benefited, but it is obvious that the developers and managers who have been exposed to this effort are much better prepared to plan, control, assure, and, in general, develop much higher quality systems. One view of this entire program is that it is a major 'training' exercise within a large production environment, and the 800 to 1000 developers and managers who have participated in development efforts studied by the SEL are much better trained and effective software engineers.

REFERENCES

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

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

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

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

VIEWGRAPH MATERIALS

FOR THE

F. MCGARRY PRESENTATION

TOWARDS UNDERSTANDING SOFTWARE

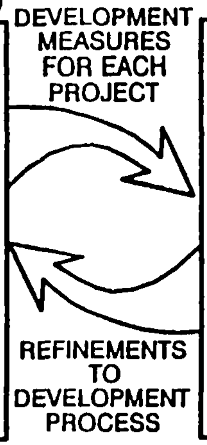
15 YEARS
in the
Software Engineering Laboratory (SEL)

Frank McGarry
Rose Pajerski
and SEL Staff

SEL ENVIRONMENT

DEVELOPERS
(DEVELOP FLIGHT DYNAMICS S/W)

STAFF	150-250 (FTE)
TYPICAL PROJECT SIZE	- 150-200 KSLOC
ACTIVE PROJECTS	- 6-10 (AT ANY GIVEN TIME)
PROJECT STAFF SIZE	- 15-25 PEOPLE
1976-1990	- 75 PROJECTS


DEVELOPMENT MEASURES FOR EACH PROJECT

REFINEMENTS TO DEVELOPMENT PROCESS

S/W ANALYSTS
(STUDY PROCESS)

STAFF 5-10 RESEARCHERS

FUNCTION
- SET GOALS/QUESTIONS/ METRICS
  - DESIGN STUDIES/ EXPERIMENTS
- ANALYSIS/RESEARCH
- REFINE S/W PROCESS
  - PRODUCE REPORTS/ FINDINGS

1976-1990 - 250 REPORTS/DOCUMENTS

DATA BASE SUPPORT (MAINTAIN/QA SEL DATA)

STAFF 2-5 (FTE)

FUNCTION
- PROCESS FORMS/DATA
- QA
- RECORD/ARCHIVE DATA
- MAINTAIN SEL DATA BASE
- OPERATE SEL LIBRARY

SEL DATA BASE

FORMS LIBRARY

REPORTS LIBRARY

1976-1990 - OVER 150,000 "FORMS"

F. McGarry
NASA/GSFC
Page 11 of 32

A498.009

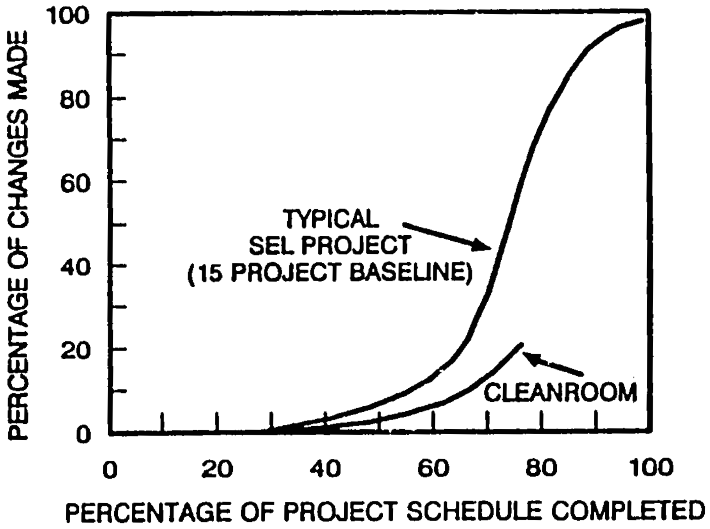
MEASUREMENT IS AN ESSENTIAL ELEMENT OF S/W PROCESS IMPROVEMENT

①

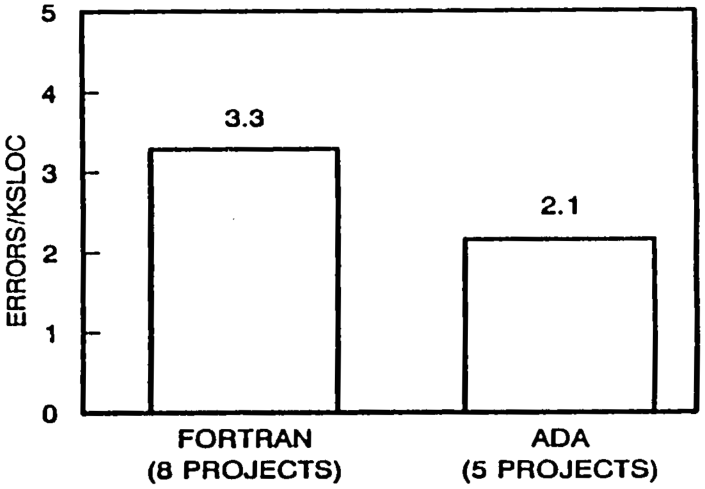
- MEASURES DEFINE PROCESS/PRODUCT BASELINE AND GAUGE CHANGE
  - ONLY MEANS OF PROVIDING UNDERSTANDING
  - WITHOUT MEASUREMENT CANNOT DETERMINE CHANGE/IMPROVEMENT
- MEASURES - SIGNIFICANT ASSET TO S/W MANAGEMENT/DEVELOPMENT
  - VITAL FOR PLANNING/ESTIMATING
  - PROVIDES INSIGHT TO HEALTH OF PROJECTS
- MEASUREMENT IS <u>NOT</u> COST PROHIBITIVE
  - EXISTS SMALL/CRITICAL SET OF MEASURES
  - CRITICAL SET LESS THAN 2% IMPACT TO PROJECT
  - BENEFITS FAR OUTWEIGH THE OVERHEAD

MEASURES - GAUGING CHANGE AND IMPROVEMENT IN THE SEL

1-1



OBSERVING IMPACTS OF PROCESS CHANGE



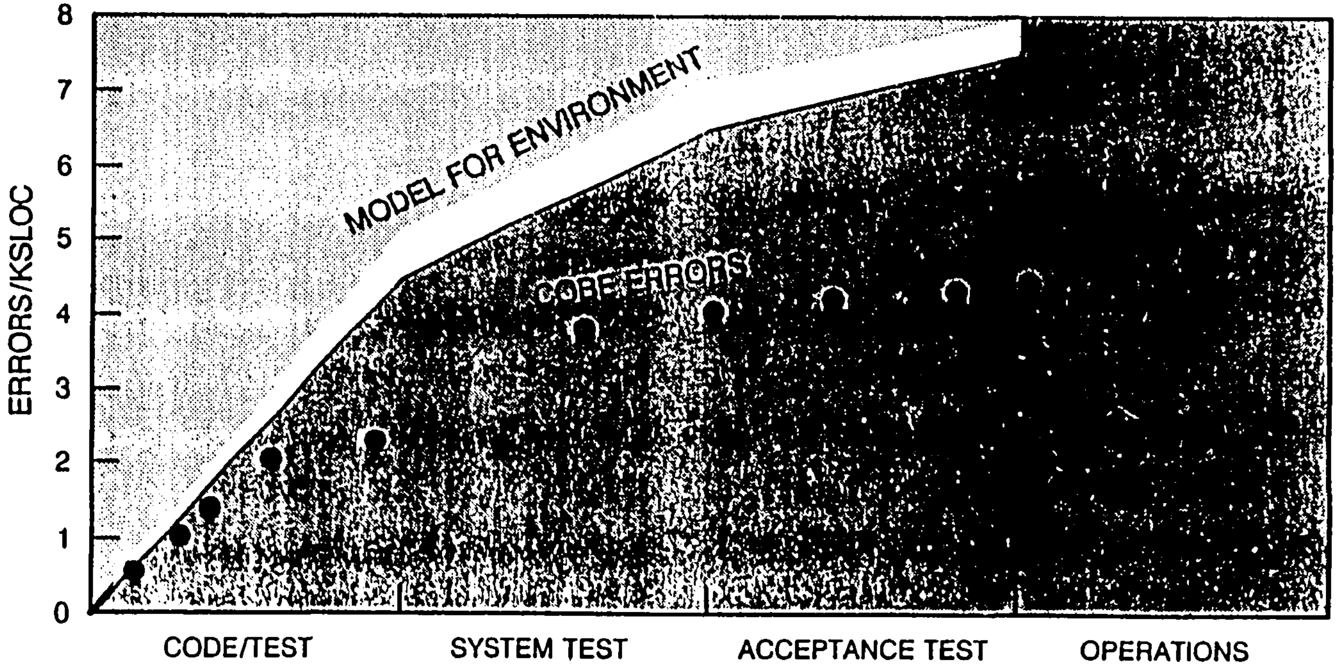
DETERMINE IMPROVEMENT DUE TO PROCESS CHANGE

F. McGarry
NASA/GSFC
Page 13 of 32

• MEASUREMENT AS A MANAGEMENT AID

1-2

TRACKING "COBE" RELIABILITY



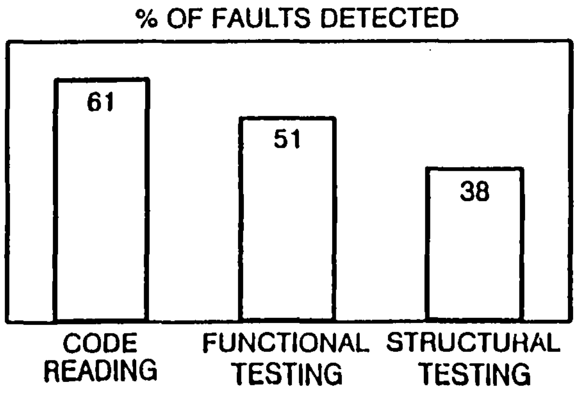
MEASURING ERROR RATES CAN PROVIDE EARLY INDICATION OF SOFTWARE QUALITY

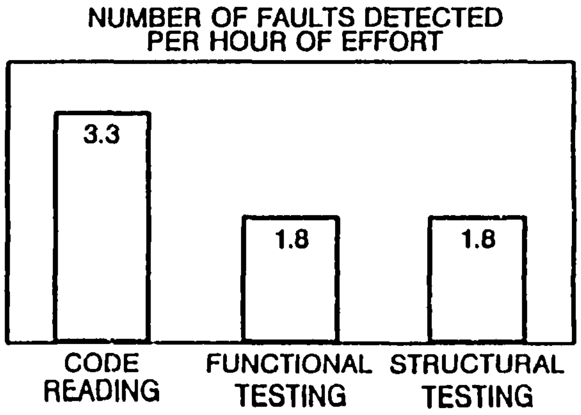
PEOPLE ARE MOST IMPORTANT RESOURCE/TECHNOLOGY

③

TEST TECHNIQUES EXPERIMENT DESCRIPTION

- 3 APPROACHES STUDIED
  - CODE READING
  - FUNCTIONAL TESTING
  - STRUCTURAL TESTING
- 32 PEOPLE PARTICIPATED (GSFC, UM, CSC)
- 3 UNIT-SIZED (100 SLOC) PROGRAMS SEEDED WITH ERRORS





EFFECTIVE TECHNOLOGY SHOULD FOCUS ON "PERSONNEL" POTENTIAL

MANY DIVERSIONS EXIST TO A SUCCESSFUL PROCESS IMPROVEMENT PROGRAM

②

(DIVERSIONS THE SEL HAS BEEN THROUGH)

- EXCESSIVE PLANNING/REPLANNING
  - JUST DO IT/START SMALL
  - LEARN WITH EXPERIENCE
  - RELY ON LOCAL STANDARDS (E.G., TERMINOLOGY)
- OVER DEPENDENCE ON STATISTICAL ANALYSIS
  - INTUITION IS A VERY USEFUL STARTING POINT
  - MAKE USE OF SUBJECTIVE DATA
- LOOKING UNDER THE LAMP POST
  - CODE ANALYZERS/CONVERTERS
  - COMPLEXITY METRICS
  - DESIGN GRAPHIC AIDS

ENVIRONMENTAL CHARACTERISTICS SHOULD DICTATE SELECTED SOFTWARE ENGINEERING TECHNIQUES

④

- SPECIFIC MEASURES/TECHNIQUES MAY NOT APPLY TO ALL "DOMAINS"

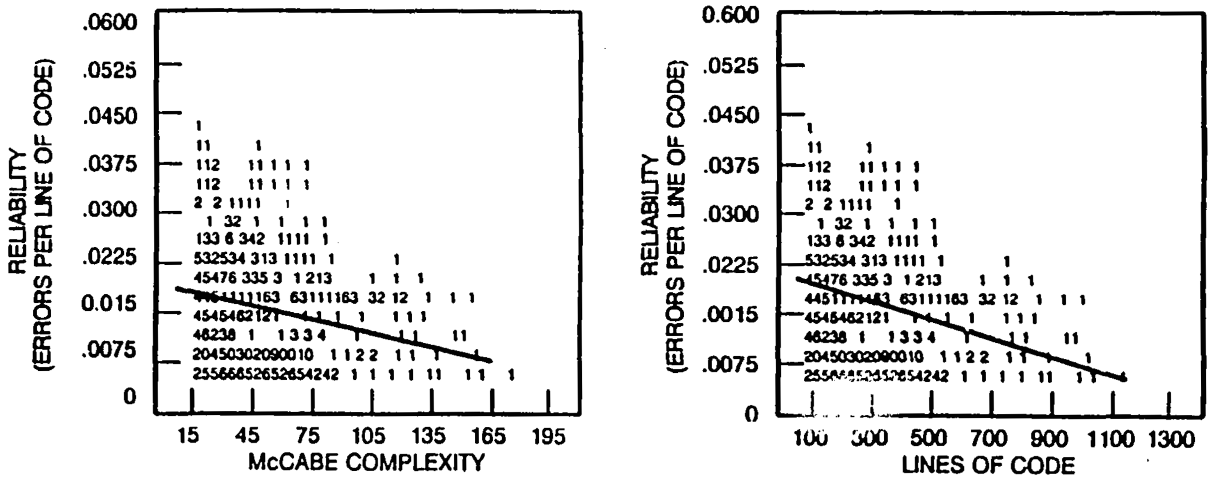
- AS ENVIRONMENT EVOLVES, METHODOLOGIES SHOULD FOLLOW (AND LEAD)

- TAILOR STANDARDS/POLICIES

SPECIFIC MEASURES MAY NOT APPLY TO ALL "DOMAINS"

(4-1)

SOFTWARE MEASURES IN THE SEL



CORRELATIONS

	TOTAL LINES	EXECUTABLE LINES	McCABE COMPLEXITY	HALSTEAD LENGTH
HALSTEAD LENGTH	0.85	0.91	0.91	1.00
McCABE COMPLEXITY	0.81	0.87	1.00	
EXECUTABLE LINES	0.84	1.00		
TOTAL LINES	1.00			

SAMPLE OF 688 MODULES

CHARACTERISTICS OF EFFECTIVE POLICIES

4-2

STANDARDS/POLICIES MUST BE:

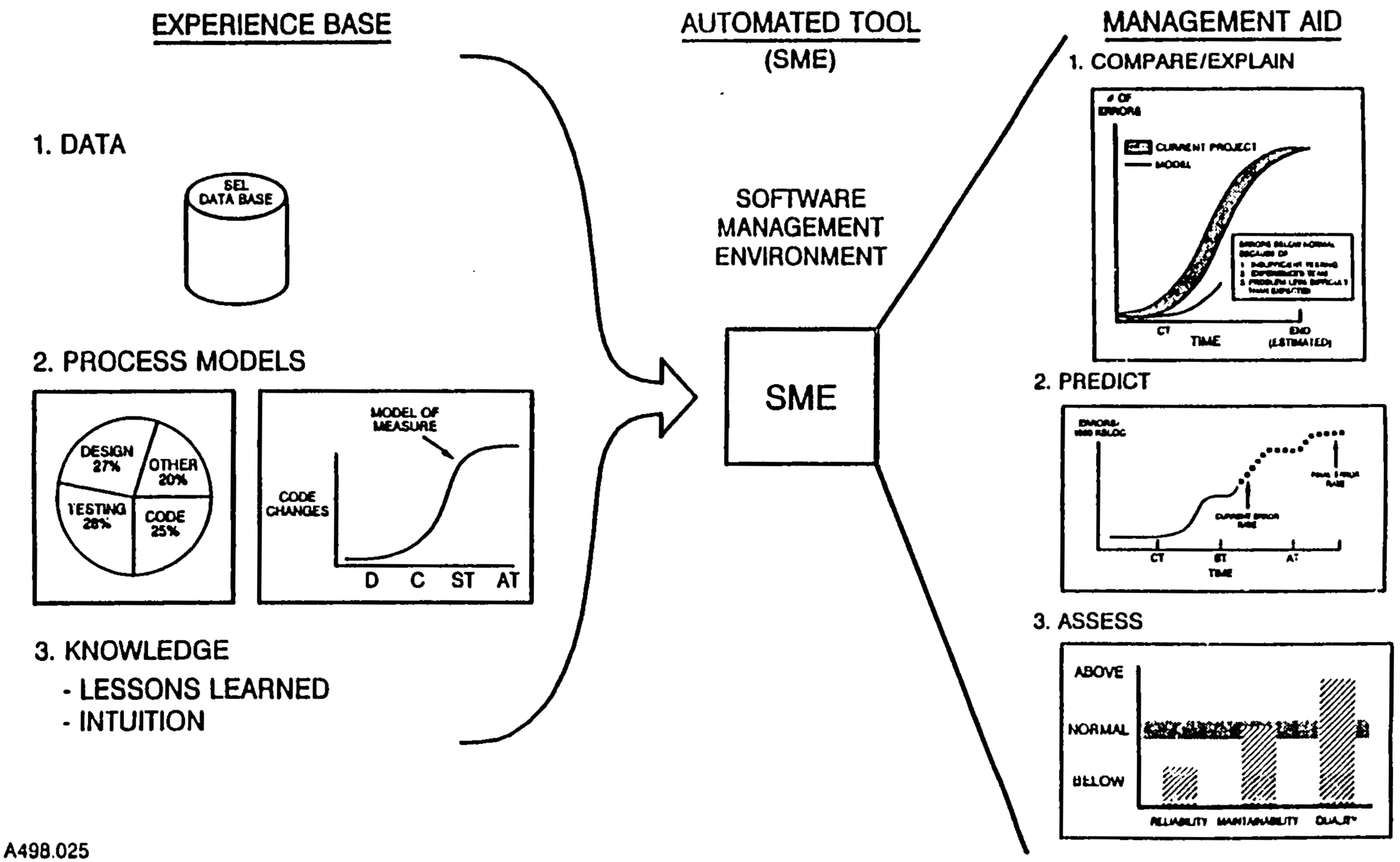
1. WRITTEN
   - MAY BE COMBINED (GENERIC AND TAILORED)
   - CAREFULLY "PRESENTED"
2. UNDERSTOOD
   - NOT TO INCLUDE EXCESSIVELY ALIEN TECHNOLOGY
   - TRAINING OFTEN REQUIRED
3. "LEGACY-BASED"
   - DERIVED FROM NEED/LEGACY
   - CONTINUALLY EVOLVING
   - ALL ELEMENTS ARE "DEFENDABLE"
4. ENFORCED
   - SUPPORTED BY MANAGEMENT
   - LIMITED "DETAIL"
5. MEASURABLE
   - OBSERVABLE (CAN TELL IF IT'S BEING FOLLOWED)
   - REQUIRES SELF-EVALUATION

AUTOMATION IS AN INSTRUMENT OF PROCESS IMPROVEMENT (NOT A REPLACEMENT FOR PROCESS UNDERSTANDING)

⑤

- TOOLS CAN PROVIDE SIGNIFICANT BENEFIT TO WELL-DEFINED EXPERIENCE BASE (E.G., SME IN THE SEL)
- "IMMATURE" PROCESSES ARE NOT AUTOMATABLE
  (IF YOU CAN'T DO IT MANUALLY - DON'T TRY TO AUTOMATE IT)
  (E.G., OVER RELIANCE ON CASE/ANALYZERS/AUDITORS/ MEASUREMENT TOOLS)
- EFFECTIVE TOOLS MUST ADDRESS DEFINED PROCESS NEED
  (MATCH SOLUTION TO PROBLEM)
  (E.G., OVERUSE OF CODE TRANSLATOR/CODE ANALYZERS/ TEST GENERATORS, ...)

(5-1) AUTOMATING A WELL-UNDERSTOOD "EXPERIENCE BASE" IN THE SEL
(SOFTWARE MANAGEMENT ENVIRONMENT (SME))



EXPERIENCE BASE

1. DATA

2. PROCESS MODELS

3. KNOWLEDGE

- LESSONS LEARNED
- INTUITION

AUTOMATED TOOL
(SME)

SOFTWARE MANAGEMENT ENVIRONMENT

SME

MANAGEMENT AID

1. COMPARE/EXPLAIN

2. PREDICT

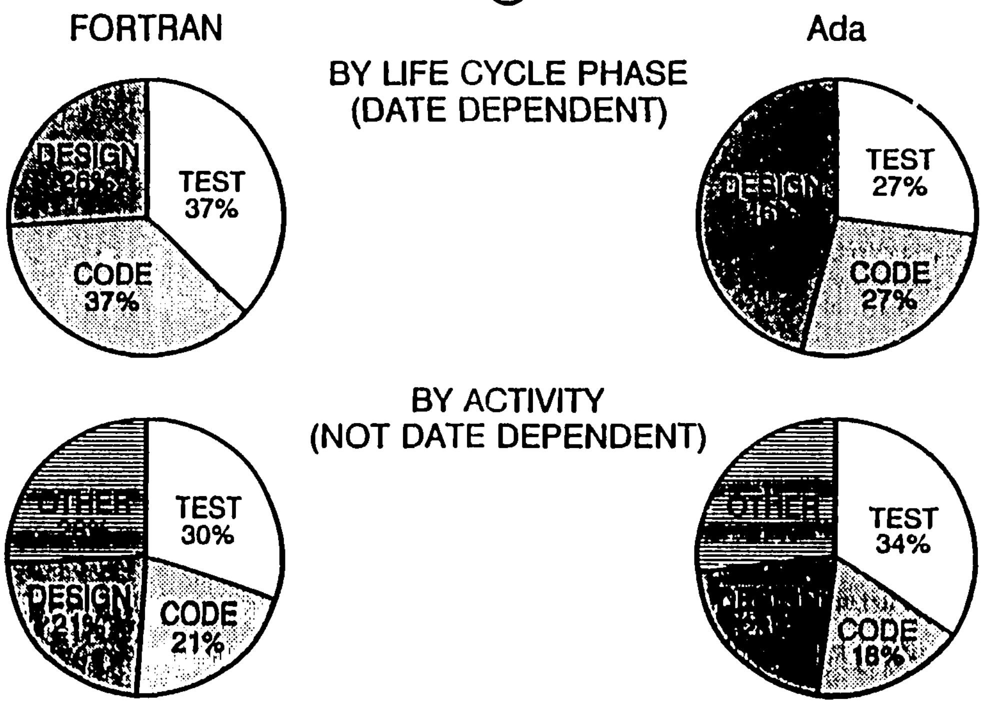
3. ASSESS

F. McGarry
NASA/GSFC
Page 21 of 32

A498.025

HERITAGE OF ENVIRONMENT WILL STRONGLY INFLUENCE PROCESS

6



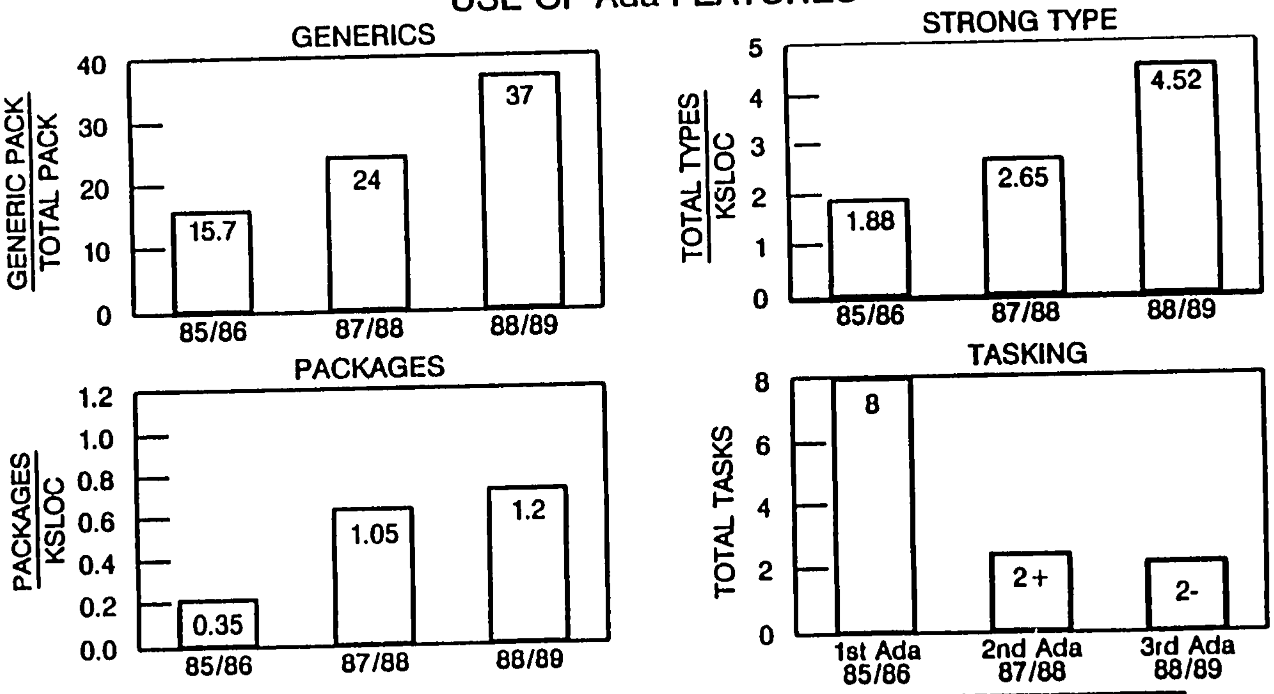
MAJOR DATES CHANGED (CDR, ...) BUT EFFORT DISTRIBUTION STILL QUITE SIMILAR

A488.018 *BASED ON 5 Ada AND 8 FORTRAN PROJECTS OF SIMILAR TYPE IN THE SEL

SIGNIFICANT PROCESS CHANGE REQUIRES SIGNIFICANT EFFORT/TIME

6-1

USE OF Ada FEATURES



- USE OF Ada FEATURES CHANGES APPRECIABLY WITH EXPERIENCE
- NOT ALL FEATURES APPROPRIATE FOR APPLICATION

SOFTWARE CAN BE MEASURABLY IMPROVED THROUGH APPROPRIATE USE OF AVAILABLE TECHNOLOGIES

⑦

EXAMPLES IN ONE ENVIRONMENT (SEL)

TECHNOLOGY	DEMONSTRATED IMPACT
- "READING"	REPEATEDLY SHOWN TO IMPROVE SOFTWARE RELIABILITY (NO ADDITIONAL COST)
- DESIGN CRITERIA (STRENGTH)	DEMONSTRATED TO PRODUCE MORE ERROR FREE SOFTWARE
- Ada	SIGNIFICANT COST BENEFIT THROUGH REUSE
- OOD	REUSE
- CLEAN ROOM	SIGNIFICANT IMPROVEMENT IN RELIABILITY AND PRODUCTIVITY (ALSO RESOURCE CONSUMPTION DOWN)
- MANAGEMENT/ MEASUREMENT	MAJOR IMPROVEMENT IN PLANNING, ADJUSTING AND CONTROL - COST ESTIMATION - SCHEDULE ESTIMATION

ASSESSING "STRENGTH" AND "SIZE" AS A STANDARD FOR DESIGN

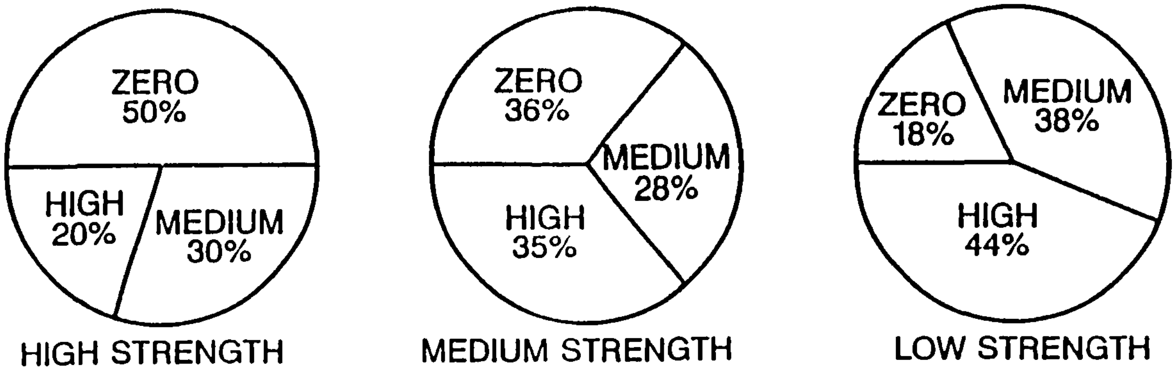
7-1

EXPERIMENT:

- 450 FORTRAN MODULES (ACROSS 4 SYSTEMS - OVER 20 DEVELOPERS)
- DETAILED COST AND ERROR DATA ON ALL MODULES
- DETERMINE RELATIONSHIPS: "STRENGTH" TO RELIABILITY AND "SIZE" TO RELIABILITY

RESULTS:

FAULT RATE FOR CLASSES OF MODULE STRENGTH



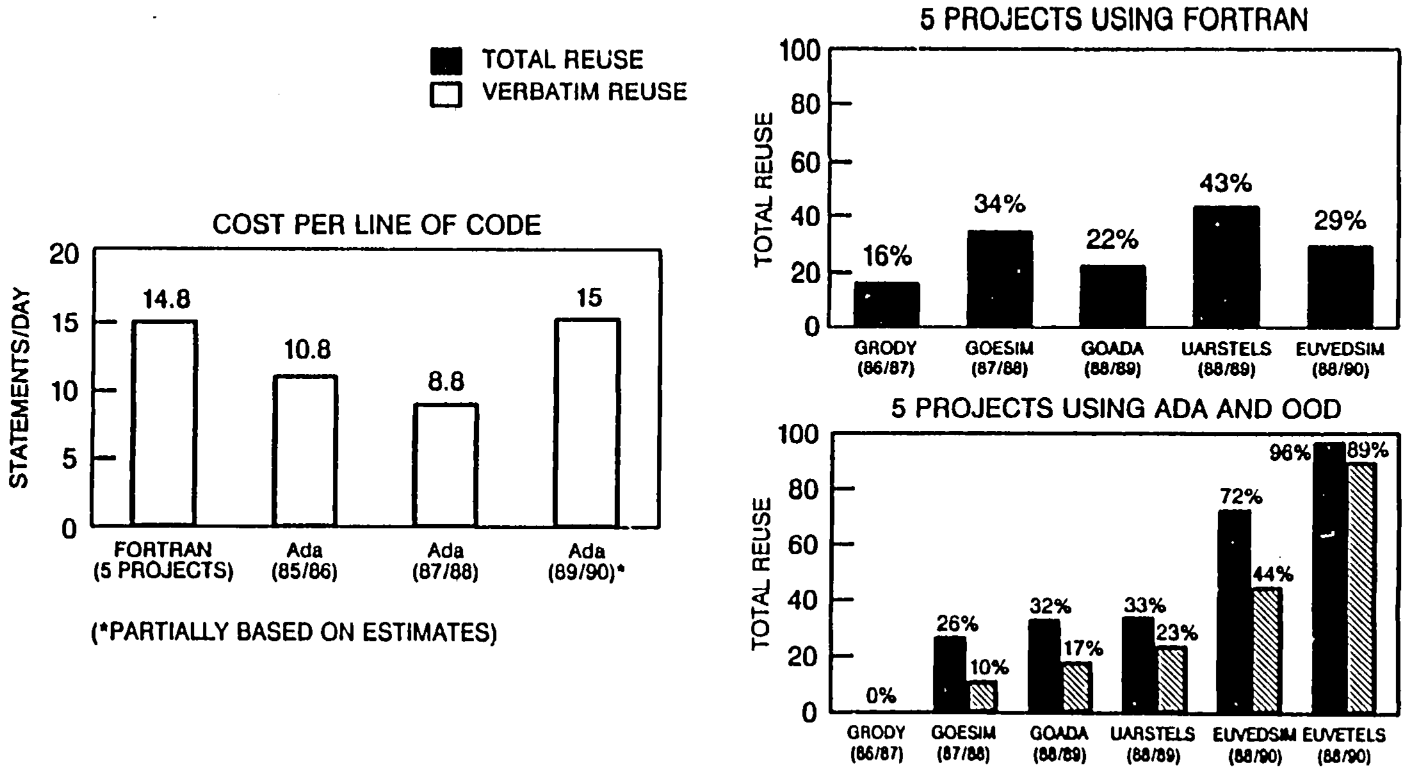
HIGH STRENGTH MEDIUM STRENGTH LOW STRENGTH

A498.024

DESIGN MEASURES SUMMARY

- GOOD PROGRAMMERS TEND TO WRITE HIGH-STRENGTH MODULES
- GOOD PROGRAMMERS SHOW NO PREFERENCE FOR ANY SPECIFIC MODULE SIZE
- OVERALL, HIGH-STRENGTH MODULES HAVE *A LOWER FAULT RATE AND COST LESS* THAN LOW-STRENGTH MODULES
- OVERALL, LARGE MODULES COST LESS (PER EXECUTABLE STATEMENT) THAN SMALL MODULES
- FAULT RATE IS NOT DIRECTLY RELATED TO MODULE SIZE

Ada (AND OOD)* IMPACTS ON "COST" FROM SEL EXPERIENCES



(*PARTIALLY BASED ON ESTIMATES)

1. DEVELOPMENT COST PER STATEMENT HAS BEEN NO "CHEAPER" FOR ADA
2. REUSE POTENTIAL OF Ada IS SIGNIFICANT

*ALL Ada PROJECTS APPLIED OOD TECHNIQUES

HAS THE EFFORT BEEN WORTH IT?
(1975 - 1990)

• SEL EXPENDITURES (1990 DOLLARS)	
- RESEARCH SUPPORT (UNIVERSITY) (EXPERIMENTATION, ANALYSIS, RESEARCH, REPORTS, ...)	$2.5M
- RESEARCH AND TECH TRANSFER (CSC) (ANALYSIS, RESEARCH, REPORTS, OVERHEAD TO DEVELOPMENT PROJECTS)	$5.5M
- DATA PROCESSING AND GENERAL SUPPORT (CSC AND OTHERS) (PROCESS/QA DATA, SEL DATA BASE, REPORTS, ...)	$6.0M
• PRODUCTION SOFTWARE (FLIGHT DYNAMICS) DEVELOPED (CSC AND NASA)	$130M

HAS THE EFFORT BEEN WORTH IT?

(1975 - 1990)

IMPACT OF SEL RESEARCH*

	1976 - 1980	1986 - 1990
MANAGEABILITY	• COMPLETE DEPENDENCE ON PERSONNEL CAPABILITY • WIDE VARIANCE IN COST/QUALITY • NO GUIDANCE FOR SELECTING METHODS	• PROCESS-MODELED AND EFFECTIVE • SOFTWARE MORE PREDICTABLE, CONSISTENT • RATIONALE FOR METHODS USED EXISTS
COST PER LINE OF CODE	≈ 24 SLOC/DAY	≈ 24 SLOC/DAY
RELIABILITY (UNIT TEST THRU ACCEPTANCE)	8.4 E/KSLOC	5.3 E/KSLOC
CODE REUSE	15-25%	25-35%
REWORK	35-40% OF TOTAL EFFORT	20-30%

*PROBLEM COMPLEXITY AND SUPPORT ENVIRONMENT HAVE ALSO CHANGED SIGNIFICANTLY

HAS THE EFFORT BEEN WORTH IT?

FINAL OBSERVATIONS

- "OUR" UNDERSTANDING OF SOFTWARE HAS IMPROVED SIGNIFICANTLY (WE DO SOFTWARE BETTER)
- CONTRIBUTIONS TO SOFTWARE RESEARCH AND DEVELOPMENT (MEASUREMENT, MANAGEMENT, EXPERIENCE BASE, ...)
- PROFESSIONAL DEVELOPMENT OF DEVELOPERS, MANAGERS, RESEARCHERS
- "NEW" AWARENESS BY MANAGERS, DEVELOPERS (SOFTWARE CAN BE ENGINEERED)

ONGOING/FUTURE ACTIVITIES FOR THE SEL

GENERAL

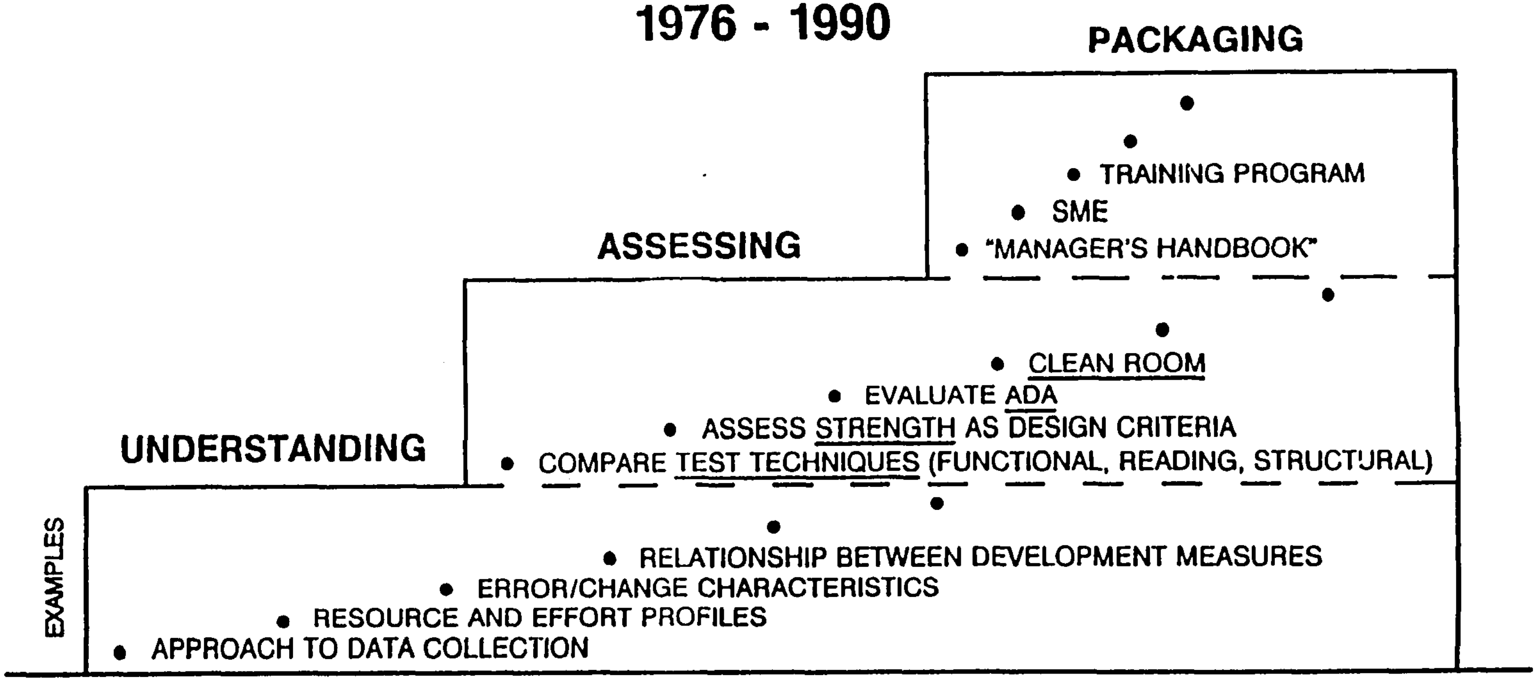
- CONTINUE EVALUATION OF "PROCESS IMPROVEMENT"
  - G/Q/M
  - EXPERIMENTATION
  - REFINEMENT
- DOMAIN ANALYSIS FOR "EXPERIENCE BASE"
  - RELEVANCE TO OTHER "ENVIRONMENTS"
  - CHARACTERIZING DOMAIN OF "EXPERIENCE BASE"
- EXPANSION OF LIFE CYCLE ANALYZED
  - MAINTENANCE
  - SPECS/REQUIREMENTS
  - EXPANDED MEASUREMENT

CURRENT/NEAR FUTURE STUDIES

- CLEAN ROOM (3 ACTIVE PROJECTS)
- Ada (3 PROJECTS)
- OOD (1 ONGOING EXPERIMENT - 2 PLANNED)
- CASE (1 ACTIVE PROJECT)
- REUSE (USING EXISTING SEL PROJECTS)
- MAINTENANCE (3 PROJECTS FOR ANALYSIS)
- TESTING STRATEGIES (EXISTING SEL PROJECTS)
- MEASUREMENT (CHARACTERIZING DESIGNS)

STUDIES IN THE SEL

1976 - 1990



PACKAGING

- ●
- ●
- TRAINING PROGRAM
- SME
- "MANAGER'S HANDBOOK"

ASSESSING

- ●
- ●
- CLEAN ROOM
- EVALUATE ADA
- ASSESS STRENGTH AS DESIGN CRITERIA
- COMPARE TEST TECHNIQUES (FUNCTIONAL, READING, STRUCTURAL)

UNDERSTANDING

EXAMPLES

- ●
- ●
- RELATIONSHIP BETWEEN DEVELOPMENT MEASURES
- ERROR/CHANGE CHARACTERISTICS
- RESOURCE AND EFFORT PROFILES
- APPROACH TO DATA COLLECTION

ACTIVITIES

1976 - 1980	1980 - 1986	1986 - 1990
● DEFINE PROCESS	● INITIAL "RELATIONSHIPS"	● PROCESS IMPROVEMENT ENVIRONMENT
● CALIBRATE "PROCESS ENVIRONMENT"	● EXPERIMENTS	● FULL TECHNOLOGY ASSESSMENT
● DEFINE MEASURES/MEASUREMENT	● REFINE MEASURES/MEASUREMENT	● FULL USE OF MEASUREMENT

EVOLVING TO AN EFFECTIVE
"PROCESS IMPROVEMENT" ENVIRONMENT