Session 1: The SEL

Frank E. McGarry, NASA/Goddard

Vic Basili, University of Maryland

Michael Stark, NASA/Goddard

EXPERIMENTAL SOFTWARE ENGINEERING: SEVENTEEN YEARS OF LESSONS IN THE SEL

F. McGarry

SOFTWARE ENGINEERING BRANCH
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771
(301) 286-5048

ABSTRACT

This paper describes seven key principles developed by the the Software Engineering Laboratory (SEL) at the Goddard Space Flight Center (GSFC) of the National Aeronautics and Space Administration (NASA). For the past 17 years, the SEL has been experimentally analyzing the development of production software as varying techniques and methodologies are applied in this one environment. The SEL has collected, archived, and studied detailed measures from more than 100 flight dynamics projects, thereby gaining significant insight into the effectiveness of numerous software techniques (e.g., References 1, 2, and 3), as well as extensive experience in the overall effectiveness of "Experimental Software Engineering". This experience has helped formulate follow-on studies in the SEL, and it has helped other software organizations better understand just what can be accomplished and what cannot be accomplished through experimentation.

INTRODUCTION

The Software Engineering Laboratory (SEL) was established in 1976 as a joint venture among Goddard Space Flight Center (GSFC) of the National Aeronautics and Space Administration (NASA), the University of Maryland, and Computer Sciences Corporation to study software technologies as applied to production software in the Flight Dynamics Division. The goal was to measure the effects of various methodologies on the development process and then to use on ensuing projects those techniques that proved favorable. During its 17 years of existence, the SEL has experimented with more than 100 development efforts, collecting detailed information on each. For each project (experiment), a goal or set of goals was defined; then a plan was developed to ask the necessary questions and collect the necessary measures for the particular project. This approach has come to be called the Goal-Question-Metric (GQM) paradigm (Reference 4).

The projects studied are all of the flight dynamics class, with similar levels of complexity. They ranged in size from four thousand to five thousand source lines of code (KSLOC) to more than 1.5 million source lines of code (MSLOC), the typical project being around 150 KSLOC and requiring about 25 staff years to develop. The relative homogeneity of the class of systems studied in the

SEL is particularly attractive to experimental software engineering.

The SEL has generated more than 200 papers (Reference 5) documenting the studies that have been completed. Each of these typically reports on some particular method that was studied (e.g. Ada, OOD, IV&V, testing techniques), but some of the reports also reflect major lessons learned about overall approaches to software experimentation and research. Many papers, for instance, have been written about software metrics, obviously an integral part of this process but not its major driver. This kind of information is probably as important as the specific study results pertaining to a software development approach. Seven principles emerge from this activity as critical, in the SEL's view, for any organization pursuing efforts in experimental software engineering.

2. The Seven Principles of Experimental Software Engineering

In the course of conducting more than 100 experiments during the past 17 years, the SEL has collected and archived more than 135 megabytes (MB) of measurement data; it has documented its results in more than 200 reports and papers. The many valuable successes, as well as the mistakes, encountered during this activity have taught the SEL a great deal about the overall process of experimental software engineering (Reference 3). The information presented here is based on the 100 production projects studied, some 200,000 forms collected and analyzed, data and subjective information from 800 to 1000 people, and more than 15 years of experience with various levels of software engineering experiments.

The seven key principles that the staff of the SEL has derived are these:

1. Improvement is characterized by continuous, sustained, and methodical change, not by waiting for some technology breakthrough.
2. Experimental data analysis must be addressed in a specific context/domain.
3. The goal of experimental software engineering must be self-improvement, not external comparisons.
4. Data collection must not be the dominant element of process improvement; analysis and application are the goal.
5. Data are uncertain and fallible; you must design experimentation to accept those facts.
6. There must be a separate organizational element—not the development organization itself—to package experience.
7. Effective packaging must be experience based.

2.1. Principle 1: Improvement is characterized by continuous, sustained, and methodical change, not by waiting for some technology breakthrough.

The SEL originally expected to identify specific technologies having a potential for remarkable improvements in productivity, but now, after so many years of study, we see that any genuine improvement results from a slow evolutionary process in which change is guided by experience and learning. Many software technologies, such as Ada, reuse, OOD, SADT, CASE, and integrated environments, were initially expected to improve software development by orders of magnitude, but in fact there is no realization of attaining N to 1 gains in productivity through some specific approach.

Overall, the SEL has found that any single software technique must be integrated into a repeatable process, along with a means of measuring effectiveness and providing feedback to the development organization, before it can have a lasting favorable effect on the development process in an organization. This concept of continuous improvement, of course, is similar to the concept of TQM and other improvement paradigms; in this environment, it has led to impressive gains in reliability (65 percent improvement in 17 years), in reuse (some classes rising from a 25-percent average to more than a 60-percent average), and in productivity of new software (25 percent greater in 1993 than in 1980),

but it is important to note that these gains have been small incremental changes guided by the continuous examination of completed projects.

2.2. Principle 2: Experimental data analysis must be addressed in a specific context/domain.

One of the most significant barriers to successful experimental software engineering, and to data analysis in particular, is the failure of the analysts to completely understand and factor in the context from which the information was taken. Measurement data are a very attractive device to the software researcher, but unless the domain is understood very misleading or erroneous conclusions will be drawn. It is imperative to understand the characteristics of the project from which the data were extracted.

Figure 1 depicts data that characterize the level of reuse for projects in the SEL during the past 8 years. The data represent all projects, developed on several different platforms, of varying size, using different methodologies, and—most importantly—having different goals. Without understanding the contexts of the data, one could easily conclude that the level of reuse has remained essentially constant in the SEL, and, on the average, that could be true. Figure 2, however, depicts the same data with projects that used OOD identified by circles. These

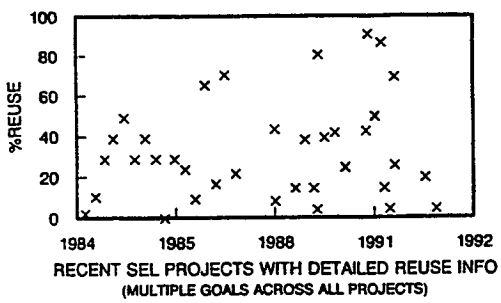


Figure 1. Recent SEL Projects With Detailed Reuse Information

projects, all of a similar class, had a focus on reuse and OOD. This chart indicates, therefore, that the particular technology (OOD) is showing a favorable trend for software reuse, but also that many more circumstances (context) must be studied before that effect can be completely understood.

Unfortunately it is nearly impossible to capture every relevant piece of information pertaining to the context of the project and store it on the database. The analysts must understand not only the environment, the staff, the management style, the project goals, and the like; they must also understand any extenuating circumstances that could have caused certain characteristics. This can only be done by spending time with the developers and managers; very questionable conclusions can be drawn from bulk data alone. Because only limited information pertaining to the context can be captured in the database, the SEL discourages external usage of the database.

2.3. Principle 3: The goal of experimental software engineering must be self-improvement, not external comparisons.

Any process-improvement program is intended to guide the evolution of change toward some set of objectives within the organization. The principal goal is self improvement; it is not comparison with other domains. When software organizations focus on external comparisons, they can easily lose sight of their own goals.

Also, problems may not be similar from one domain to another, and comparisons across domain

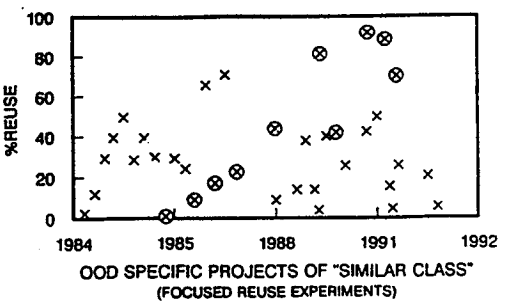


Figure 2. OOD-Specific Projects of Similar Class

boundaries are always uncertain. As an organization understands the general characteristics of its own process and products, some high-level attributes may be compared generally, but that must be treated as a secondary goal, not as the primary goal of any experimental software engineering effort.

If the focus of any process improvement should be on the local domain, it makes little sense to expend effort generating broadly accepted standard definitions (lines of code, errors, etc.,) or building national databases.

The experience of the SEL indicates that effort is better spent in efforts to understand the organization's own domain, its own set of definitions, its own software data, and its own goals. As the organization matures, it may begin to map its own characteristics into the given characteristics of other domains for some possible high- level comparisons, but, again, this is a secondary process of no certain value.

Consequently, the assumption that more broadly populated databases, local or national, will result in greater insight into experimental software engineering is not valid. Rather than continuously adding new data to a database, the key goal must be generating more focused data of higher quality.

2.4. Principle 4: Data collection must not be the dominant element of process improvement; analysis and application are the goal.

Software measurement is often viewed as a goal in itself, and it is sometimes assumed to be a measure of success. This erroneous perception of measurement is often sustained by the tremendous publicity given to measurement as a goal: the conferences, papers, studies, guidebooks, etc., that support the conclusion that measurement itself is a goal worth pursuing and ignoring the question of which application the measurement information will be applied to and why.

As long as measurement is viewed as a goal in itself, and not as a means to an end, the measurement program will be doomed to failure. In fact, without a clear application, there is no reason to collect the measurement data. This fact in itself should minimize the amount of data that any software organization attempts to collect. But often, measurement programs are instituted so that measurement data can be used to qualify complexity and general characteristics of software without a prior definition of just what criteria are to be acceptable and for what reason. This usually leads to the collection of excessive amounts of data and very questionable (forced) interpretation of some of the information.

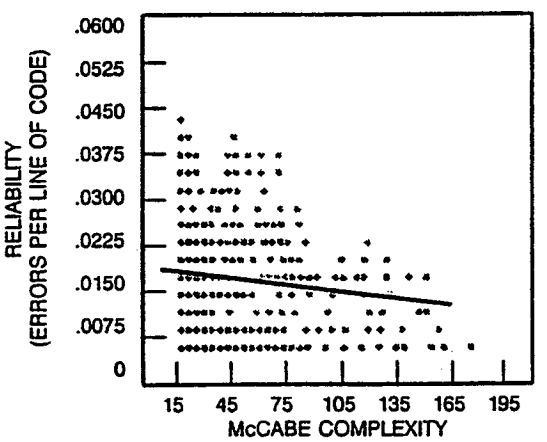
As a rule of thumb, there are three functional areas in typical measurement programs: data collection, data analysis, and general support. The bulk of any effort put into a significant software measurement program must be directed at analysis, not at collecting and processing the information. In the SEL, the typical analysis function consumes approximately 60 percent of the total measurement effort. The routine data collection is less than 10 percent of the effort, and the overall processing, archiving, quality assurance, an so forth runs about 30 percent of the effort.

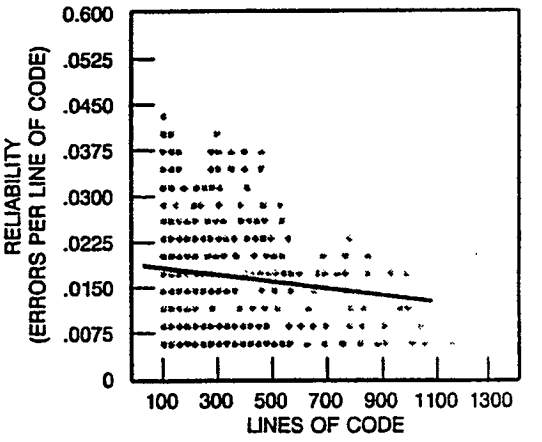
2.5. Principle 5: Data are uncertain and fallible; you must design experimentation to accept those facts.

Many inexperienced researchers see any software metrics database populated with production software characteristics as a reservoir of answers for empirical studies in software engineering technology. Carrying out large numbers of statistical analyses on very large databases with hundreds or thousands of projects—such as computing numerous correlation coefficients on every set of parameters that can be made available—can easily lead to very erroneous implications. The context, goals, subjective information, and domain understanding are as important as the amount of data available for study.

Figure 3 is an example of one study that the SEL carried out in 1985. By merely running a large number of statistical correlations on a class of FORTRAN modules, one could almost conclude that large modules (as counted by lines of code) are better (less prone to error) than smaller ones, and that modules of higher complexity (as measured by McCabe complexity) are more reliable than smaller ones. These data, taken in a very limited context, differed completely from subsequent studies that later showed it to be inconclusive.

In addition to the fact that such data, processed with appropriate statistical analysis and correlation coefficients, will not generate new insights automatically, researchers also find, inevitably, that data are faulty, missing, inconsistent, or otherwise unus-





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

Figure 3. Software Measures the SEL

able, perhaps to such a degree that the database will be viewed as worthless and fully corrupted. This is a universal occurrence; one cannot assume that any organization can produce complete, accurate, and consistent software development data. Completely accurate data is precluded by the tremendous ambiguity and uncertainty inherent in the software process itself. Even the best data-collection procedures, contending with varying terminology, the subjective nature of much of the information, and the limited resources that can be spent on producing such information, can produce at the most only a view of general trends. There are at least five key points that must be considered in handling data for experimental software engineering:

1. The context of the information.
2. Defined goals of the process and organization.
3. Subjective information from the developers and managers.
4. Measurement data.
5. Qualitative analysis of the data and information.

Each of these is vital in carrying out valid studies in experimental software engineering. To assume that the data itself can provide any more insight into the process than any of the other factors is not true.

2.6. Principle 6: There must be a separate organizational element – not the development organization itself – to package experience.

Many measurement programs and process improvement programs fail because the developers are expected to use any collected set of measures and apply the data toward self improvement. In fact, successful developers have a single goal: to produce quality software on a given schedule and for a given budget. They have neither the time nor the interest to develop measurement programs or to start writing new processes for ensuing projects. Because of this, a process improvement program is more likely to be effective when a separate organization is established specifically to acquire, assess, synthesize, and feedback data to the developers. The developers are then free to produce the software, having only to provide the small amount

of additional information that this new organization needs to carry out the process improvement studies.

In addition to the analysts who are responsible for the synthesis and feedback to the developers, there should also be a support organization to handle the processing of data, quality assurance, archiving, library maintenance, and so forth; such a group is invaluable to the overall success of the process improvement efforts.

2.7. Principle 7: Effective packaging must be experience based.

One of the key lessons that the SEL has learned over the past 15 years is that the developers have excellent insight as to what processes are useful and appropriate and which are of minimal use or even detrimental. They are therefore best qualified to produce software policies or processes for a development group. The experiences and general insight of the development teams must be incorporated into any attempt to generate processes or to carry out process improvement to any helpful degree.

As part of the paradigm of the experience factory (Reference 6), the first major step is to understand the local environment. This implies not only gathering data from development projects but also listening to the teams that produce software in that environment. Thus, the strengths, weaknesses, needs, and successes of the environment serve as a basis for useful processes captured in the form of standards or training programs, and—most important—the processes themselves will reflect the experiences of the development organization.

As developers gain experience with defined standard processes for particular domains, they will be able to judge better the impact of specific elements of the process. The experience packagers must adjust the process in response to observations that some elements (perhaps even something as specific as the design review process) are of no value; this includes changes to standards, training, tools, and general management practices.

ACKNOWLEDGEMENT

Kevin Orlin Johnson of CSC carried out the complete editing, integration, and organizing of this paper in its final form.

REFERENCES

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

2. McDermott, T., et al., *Gamma Ray Observatory Dynamic Simulator in Ada (GRODY) Experiment Summary*, SEL-90-004, September 1990.

3. Card, D. N., McGarry, F. E., and Page, G. T., "Evaluating Software Engineering Technologies," *IEEE Transactions on Software Engineering*, July 1987.

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

5. Morusiewicz, L., and Valett, J., *Annotated Bibliography of Software Engineering Laboratory Literature*, SEL-82-1106, November 1992.

6. Basili, V. R., and Caldiera, G., "Methodological and Architectural Issues in the Experience Factory," *Proceedings of the Sixteenth Annual Software Engineering Workshop*, SEL-91-006, December 1991.

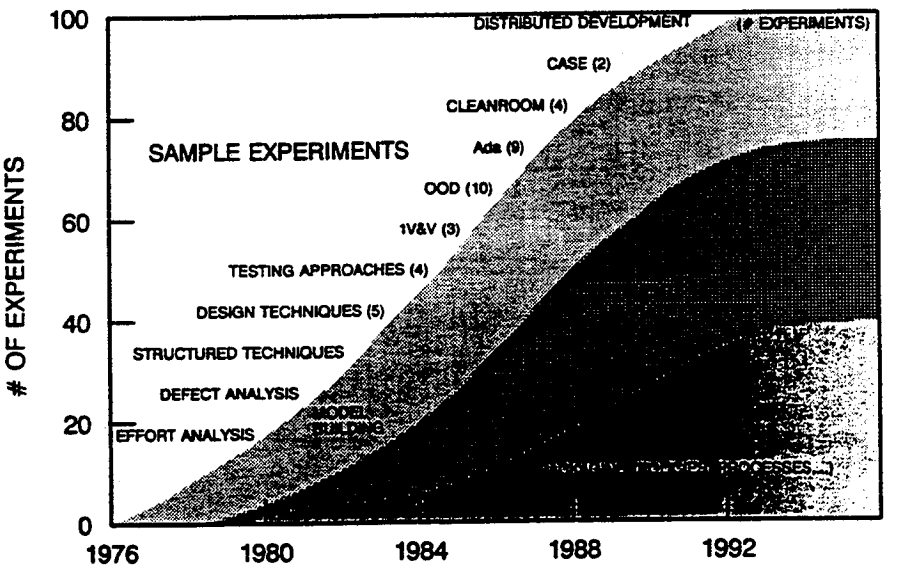
EXPERIMENTAL SOFTWARE ENGINEERING

17 YEARS OF LESSONS IN THE SEL

FRANK McGARRY
NASA/GSFC

G498.001

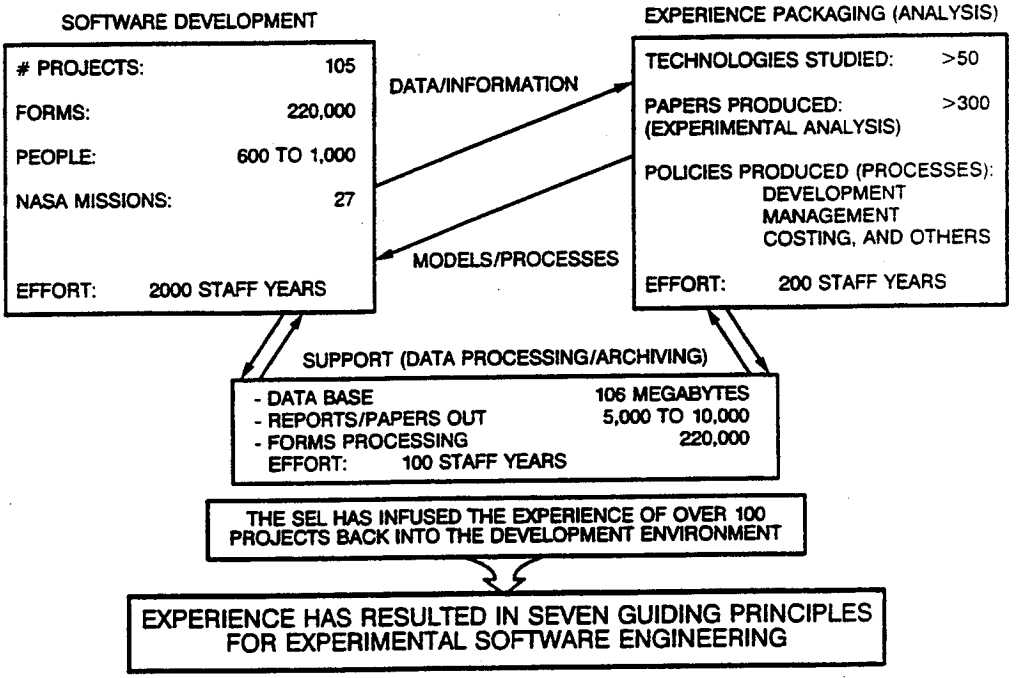
EXPERIMENTAL SOFTWARE ENGINEERING IN THE SEL
BACKGROUND



SEL HAS CONDUCTED OVER 100 EXPERIMENTS
USING NASA PRODUCTION SOFTWARE PROJECTS

G498.002

SUMMARY OF ACTIVITIES
(1976 - 1992)



G498.003

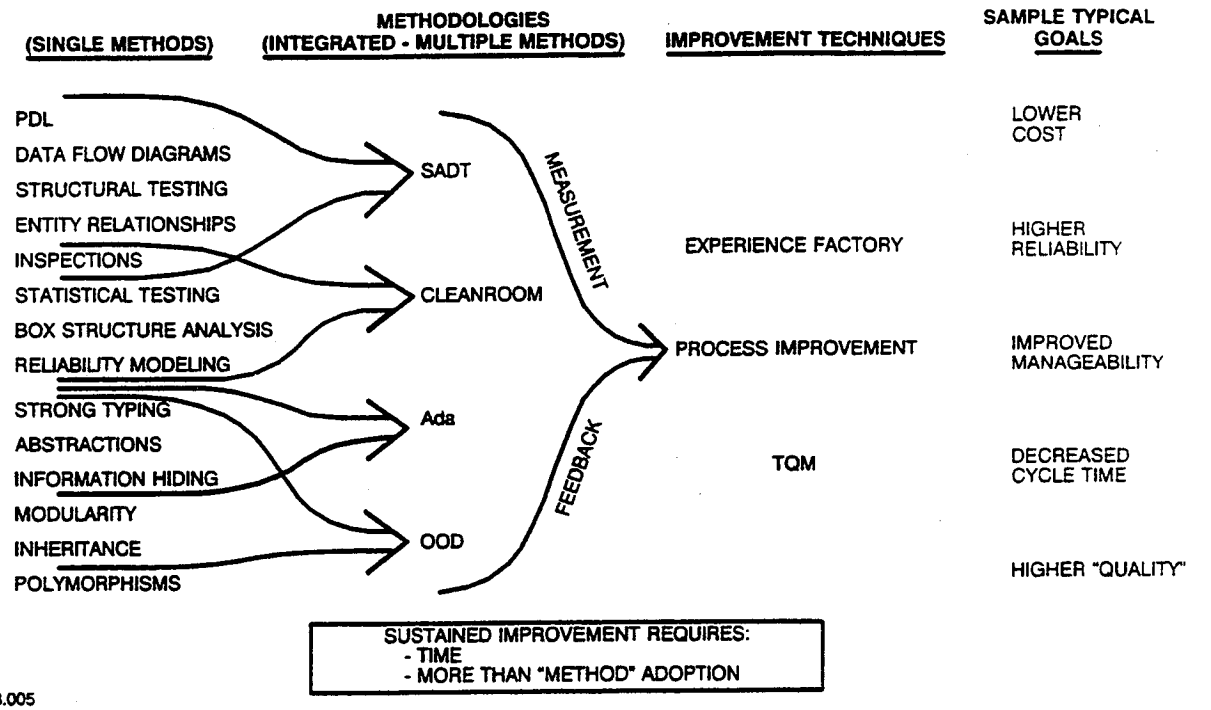
① IMPROVEMENT IS CHARACTERIZED BY CONTINUAL, SUSTAINED, AND METHODICAL CHANGE; NOT BY TECHNOLOGY BREAKTHROUGH

- WILL NOT ATTAIN "N TO 1" IMPROVEMENT (WAITING ON UNIQUE TECHNOLOGY)
- EFFECTIVE CHANGE/IMPROVEMENT MUST BE DRIVEN BY EXPERIENCE (MUST FOCUS ON SPECIFIC "DOMAIN")
- SINGLE "TECHNIQUES" ARE INCOMPLETE ANSWERS
- CHANGE MUST BE MEASURED AND DEMONSTRATED

SHOULD A TECHNOLOGY BREAK THROUGH OCCUR, EXPERIENCE DRIVEN ORGANIZATIONS WOULD BE IDEALLY POSTURED TO ADOPT.

G498.004

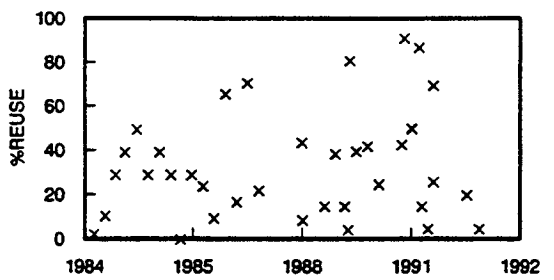
ELEMENTS OF IMPROVED SOFTWARE



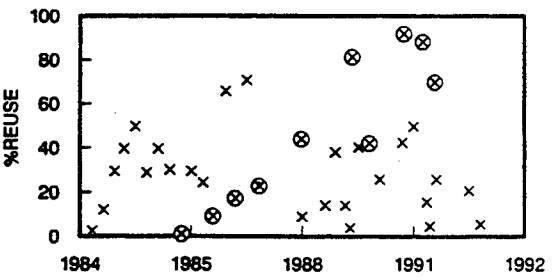
G498.005

② EXPERIMENTAL DATA ANALYSIS MUST BE ADDRESSED IN SPECIFIC CONTEXT (DOMAIN)

- DATA OUT OF CONTEXT WILL BE MISLEADING AND RESULT IN ERRONEOUS IMPLICATION
- EXTREME CAUTION REQUIRED IN ANY GENERALIZATION OR IN EXTERNAL COMPARISONS
- SHARING OF DATA IS OF VERY LIMITED BENEFIT



RECENT SEL PROJECTS WITH DETAILED REUSE INFO
(MULTIPLE GOALS ACROSS ALL PROJECTS)



OOD SPECIFIC PROJECTS OF "SIMILAR CLASS"
(FOCUSED REUSE EXPERIMENTS)

G498.006

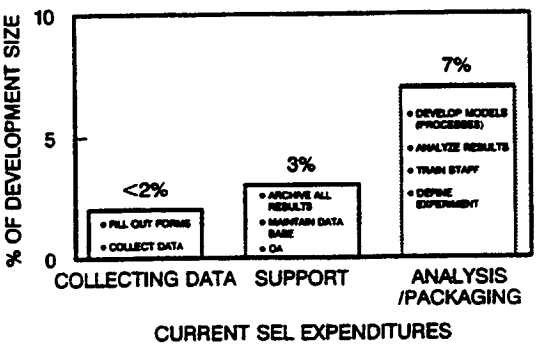
③ GOAL OF EXPERIMENTAL SOFTWARE ENGINEERING MUST BE SELF IMPROVEMENT, NOT EXTERNAL COMPARISONS

- COMPARING WITH EXTERNAL DOMAINS HAS VERY LIMITED VALUE
  - YOUR LOCAL DOMAIN IS THE WHOLE WORLD
  - EXTERNAL COMPARISONS OFTEN MISLEADING, INCONCLUSIVE, AND TIME CONSUMING
  - ANY COMPARISON MUST BE ADDRESSED AT EXTREMELY HIGH LEVEL (e.g., COMPARATIVE PROCESSES OR DEFECT RATE ON IDENTICAL PRODUCTS)
- NATIONALIZING THE EXPERIMENTAL PROCESS IS WRONG FOCUS
  - NATIONAL STANDARDS NOT OF SIGNIFICANT CONCERN (e.g., LINE OF CODE)
  - NATIONAL "DATA BASE" OF MEASUREMENT DATA NOT NEEDED (NOT AT THIS TIME)
- DEVELOP YOUR OWN DEFINITIONS (e.g., LINE OF CODE, ERROR CLASSES, etc.)
  - DEVELOP LOCAL DEFINITIONS (e.g., LINE OF CODE, ERROR CLASSES, etc.)
  - AS YOU MATURE, DEVELOP TRANSFORMATIONS (e.g., 1 LINE OF EXECUTABLE CODE ~2 PHYSICAL LINES)
- KEY TO IMPROVED RESEARCH IS NOT BROADER POPULATED DATA BASES (BETTER QUALITY AT LOCAL LEVEL; NOT MORE DATA)

G498.007

④ DATA COLLECTION (MEASUREMENT) MUST NOT BE DOMINANT ELEMENT OF PROCESS IMPROVEMENT; ANALYSIS/APPLICATION IS THE GOAL

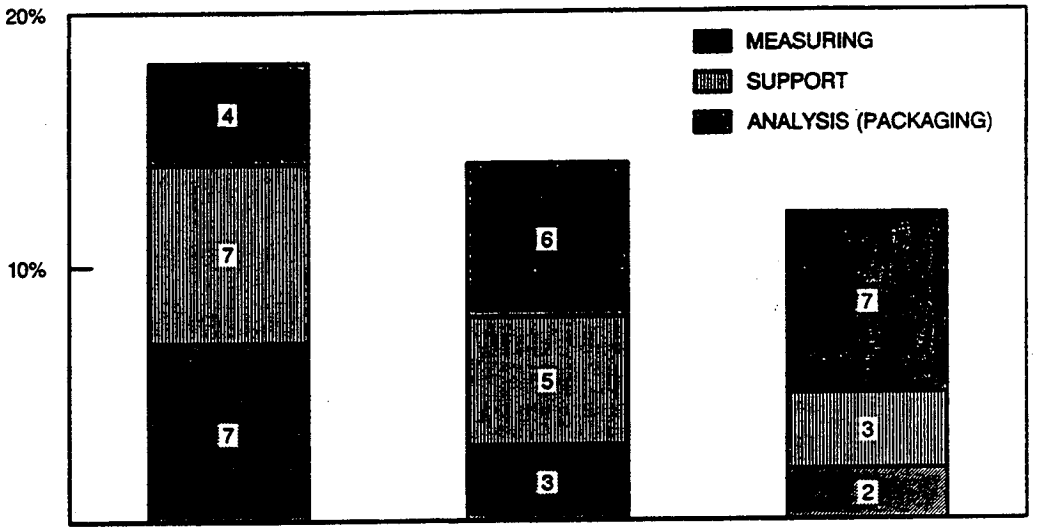
- EXCESSIVE EMPHASIS OCCASIONALLY PUT ON SEARCHING FOR "THE MEASURE"
  - AS MANAGEMENT AID, USE SIMPLE, PROVEN DATA (e.g., EARNED VALUE)
  - DON'T SEARCH FOR KEY "THRESHOLD" MANAGEMENT INDICATOR
- APPLICATION/USE OF MEASURES MUST DOMINATE COLLECTION/PROCESSING OF MEASURES
  - MORE "LESSONS LEARNED" ARE WRITTEN THAN READ
  - MINIMIZE MEASUREMENT TO DAT EXPLICITLY REQUIRED
- USE OF MEASUREMENT IS TOWARD "ENGINEERING" AND "UNDERSTANDING"
  - GOALS MUST BE DEFINED EXPLICITLY (e.g., ARE INSPECTIONS LOWERING ERROR RATES?)
  - DEVELOPERS HAVE RIGHT TO KNOW "WHY?"



CURRENT SEL EXPENDITURES

G498.006

GETTING PRIORITIES CORRECT (AS % OF THE DEVELOPMENT COST)



SEL "IMPROVEMENT" COST AS % OF DEVELOPMENT COST
(COSTS INCLUDE SME AND SOME PURE RESEARCH EFFORTS)

G498.009

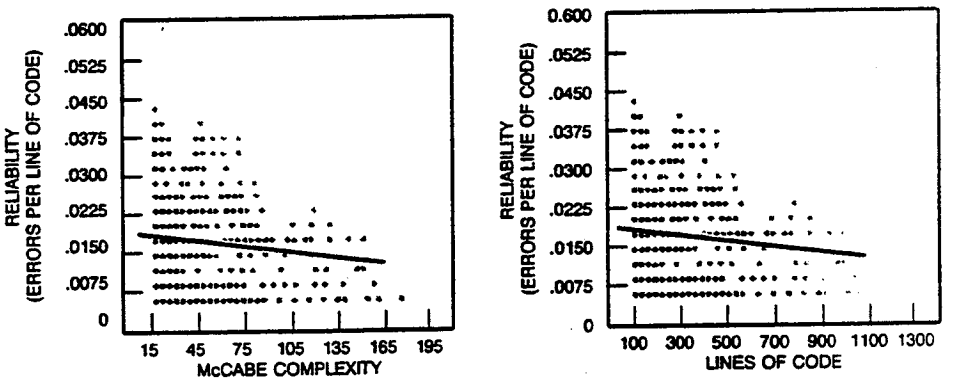
⑤ UNCERTAINTY/FALLIBILITY OF MEASUREMENT DATA IS FACT; DESIGN EXPERIMENTATION TO ACCEPT THAT

- MEASUREMENT DATA CANNOT BE TREATED AS EXACT OR COMPLETE INFORMATION
  - CONTEXT + DEFINED GOALS + SUBJECTIVE INFO + DATA + QUALITATIVE ANALYSIS (ALL PART OF THE EQUATION)
- RUNNING 1000 CORRELATION STUDIES ON 1000 MEASURES WILL CERTAINLY PRODUCE SOME QUESTIONABLE "BREAKTHROUGH"
- SUBJECTIVE INSIGHT EXTREMELY VALUABLE
- MULTIPLE EXPERIMENTS NECESSARY
  - SINGLE STUDIES CAN BE MISLEADING
  - CANNOT PROMISE QUICK INSIGHTFUL FEEDBACK

G498.010

DO NOT PLACE UNFOUNDED CONFIDENCE IN RAW MEASUREMENTS DATA

SOFTWARE MEASURES IN THE SEL



CORRELATIONS

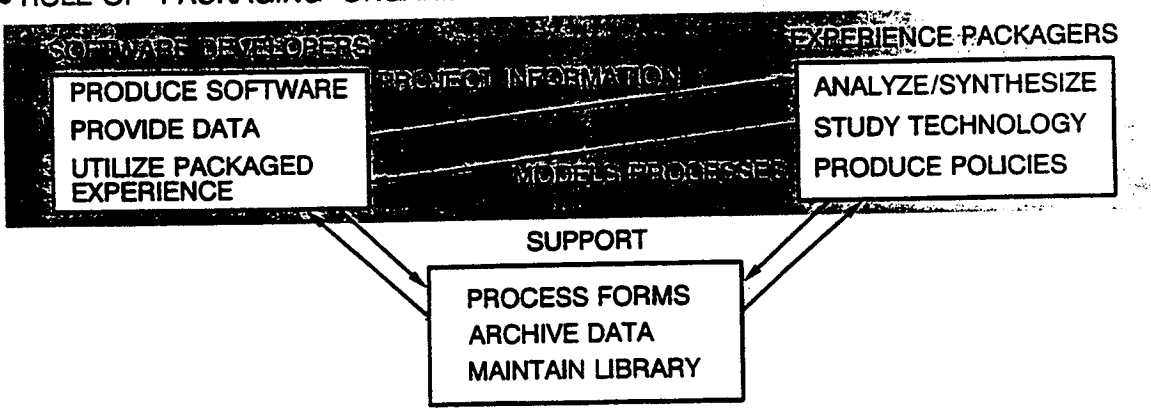
	TOTAL LINES	EXECUTABLE LINES	McCABE COMPLEXITY	HALSTEAD LENGTH
HALSTEAD LENGTH	0.85	0.91	0.91	1.00
McCABE COMPLEXITY	0.81	0.87	1.00	
EXECUTABLE LINES	0.84	1.00		
TOTAL LINES	1.00			

SAMPLE OF 688 MODULES

A498.023

⑥ MUST ESTABLISH SEPARATE ORGANIZATIONAL ELEMENT TO ADDRESS EXPERIENCE PACKAGING (CANNOT BE THE DEVELOPMENT ORGANIZATION)

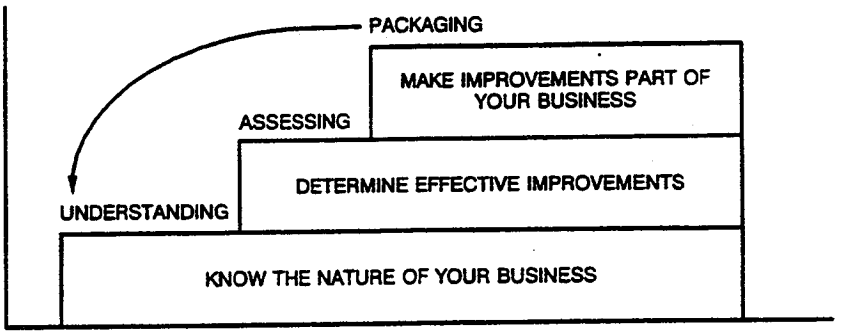
- DEVELOPERS CANNOT BE RESPONSIBLE FOR ANALYSIS/PACKAGING
  - THEY HAVE OTHER PRIORITIES
  - WILL HAVE LIMITED INTEREST
  - BURDEN MUST BE SHARED
- ROLE OF "PACKAGING" ORGANIZATION IS TO ANALYZE, SYNTHESIZE, AND PACKAGE



G498.011

⑦ EFFECTIVE PACKAGING MUST BE EXPERIENCE BASED

- EXPERIENCES OF DEVELOPERS MUST BE REFLECTED IN "PACKAGE" (e.g., STANDARDS, PROCESS,...)
- STANDARDS MUST REFLECT KNOWLEDGE/ENVIRONMENT/EXPERIENCES OF YOUR DOMAIN
- COOKBOOKS CAN AND HAVE BEEN PRODUCED AND ARE EXTREMELY VALUABLE

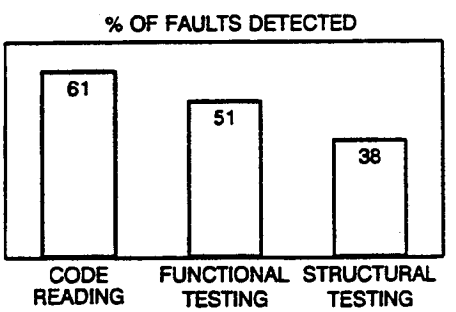


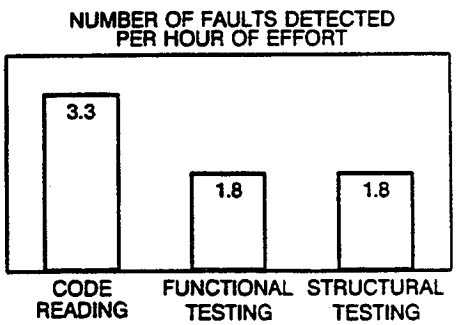
G498.012

POLICIES AND STANDARDS SHOULD REFLECT DOMAIN SPECIFIC EXPERIENCES AND CHARACTERISITICS

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

A498.022

SUMMARY

- EXPERIENCE-BASED IMPROVEMENT IS DOABLE FOR SOFTWARE
- UTILIZE THE OPINION/EXPERIENCE OF THE DEVELOPMENT ORGANIZATION
  - THEY SHOULD DRIVE THE PROCESS
- DOMAINS/CONTEXTS MUST BE REALIZED
  (ALL SOFTWARE IS NOT THE SAME)
- EXPERIENCE FACTORY STRUCTURE ENABLES SELECTION
  AND ADOPTION OF EVOLVING TECHNOLOGY (e.g. OOD)

G498.013