N94-11437

STGT Program: Ada Coding and Architecture Lessons Learned

Paul Usavage and Don Nagurney
GE Management and Data Systems Engineering,
P.O. Box 8048, Phila. PA, 19101
Phone: (215) 354–3165
Fax: (215) 354–3177

STGT (Second TDRSS Ground Terminal) is currently halfway through the System Integration Test phase (Level 4 Testing). To date, many software architecture and Ada language issues have been encountered and solved. This paper, which is a transcript of the presentation at the December 3rd meeting, attempts to define these lessons plus others learned regarding software project management and risk management issues, training, performance, reuse, and reliability. Observations are included regarding the use of particular Ada coding constructs, software architecture trade-offs during the prototyping, development and testing stages of the project and dangers inherent in parallel or concurrent Systems, Software, Hardware and Operations Engineering.

Introduction

STGT is the first major Ada development program for M&DSO, which has developed other large ground stations in FORTRAN and C. In addition to the use of Ada, GE Management and Data Systems Operations faced other software development risks in the implementation of STGT. Some of these risk items are itemized below:

- A heavily distributed system (>30 processing nodes and >100 workstations in previous ground stations)
- High real-time system content (vs. 40% real-time, 60% batch processing)
- First on a DEC/VAX platform (vs. IBM mainframes and Sun/Unix workstations)
- High-availability/high-reliability architecture (99.99% availability required)
- High hardware content (>350 racks of ground communication equipment)
- Heavily automated, X-Windows, workstation-based user interface
- First artificial intelligence (AI) based hardware fault detection/fault isolation
- Short development lead time (3 years from start to delivery)

Risk items like the above don't usually translate into the impossible, they just have a way of eating into cost and schedule margins. Several steps were taken to mitigate the risks involved. An Ada Core Team was formed prior to program startup to develop language expertise. An Ada training program was developed and its completion required for all software engineers employed on the program. Despite these efforts, many lessons were learned on the job through prototyping, development and testing. This paper is intended to be a chronicle of these risk issues and (hopefully) their resolutions.

Project Composition

The Second TDRSS (Tracking and Data Relay Satellite System) Ground Terminal (STGT) is a new ground station and an upgrade to an existing ground station in White Sands, New Mexico. These ground stations will provide command and data communications from user control facilities through the TDRS, and on to the various user satellites and the Space Shuttle.

The breakdown of thousands of source lines of code developed for each Computer Software Configuration Item (CSCI) for the project is shown in Table 1.

CSCI	Size (Lines of Code)	Thousands of Hours[1]	LOC / Hour
TTC (Satellite Control)	100k	115k	0.86
DIS (Communication)	76	79	0.96
USS (Ground Equip.)	71	58	1.22
EXC (Scheduling)	26	23	1.13
WKS (Workstation Interface)	152	56	2.70
COM (Infrastructure)	23	37	0.62
MDS (Development Env.)	100	36	2.77
SIM (Simulators)	40	40	1.00
Totals	**588**	**444**	**1.32**

Note:
1 - Requirements Analysis through Software Test

Descriptions of the CSCIs are as follows:

- TTC:
  Tracking, Telemetry and Command CSCI, responsible for controlling the Tracking and Data Relay Satellites (TDRS) used by NASA to relay user satellite and space shuttle telemetry and command data. Responsible for commanding the satellite, monitoring its health, and controlling the ground antenna in order to point at the satellite.

- DIS:
  Data Interface Subsystem, responsible for interfacing with the NASA communication network, accepting scheduling orders from NASA, and switching the inputs and outputs from the ground station to data links between STGT and the other NASA locations.

- USS:
  User Services Subsystem, responsible for controlling most of the ground communications equipment (GCE) and supporting communications to the TDRS and to various user satellites.

- EXC:
  Executive, responsible for scheduling of a single Space to Ground Link Terminal (SGLT) controlling a single TDRS satellite. There will be six SGLTs overall in the two ground station installations.

- WKS:
  Workstation, responsible for operator interface, including intelligent graphically-oriented displays, operation alert messages and operator commanding capabilities.

- COM:
  Common Run-time Environment, provides common capabilities across all computers including communications within and between computers, data logging, startup/shutdown/ failover control, and device driver interfaces.

- MDS:
  Maintenance and Development Subsystem, provides COTS tools for development and maintenance environment, database displays/editors, and configuration management software.
- SIM:
  Simulations, provides simulations of the NASA scheduling interface, ground hardware, and the TDRS. Simulators are used in testing, training and problem investigation.

Software Architectural Issues

Architectural Reuse

STGT attempted a high level of reuse and incorporated reuse into its architecture. and in many ways succeeded. Attempts were made in object-oriented design, some of which succeeded in providing reliable, understandable, reusable products, and some of which only caused major headaches. Those that were problematic were usually related to lack of understanding of the scope and breadth of the situations in which the code would be reused, the computers on which the code would run, and the environments on these computers. For example, code reused on a workstation found itself in a rather different environment than on the large VAXes, due to lack of availability of large local databases.

Reuse was attempted on both large and small scales. Small-scale reuse was of course more easily planned than large-scale reuse. Large-scale reuse was more likely to result in complicated error conditions, where different subsystems (and their engineers and programmers) wanted to operate in different ways but were constrained by identical implementations due to code reuse.

Ada Reuse

In the early days we had "reuse evangelists" who proposed massive, complex, self-initializing generics for everyone. Almost every case that was ever implemented was later disabled, deleted, gutted or otherwise rewritten. Generics proved very difficult to debug using a source-level interactive debugger, relatively slow to execute in real-time, and very hard to write. Elaboration time-initialization code was also difficult to debug and prone to exception handling difficulties. Simple generics, on the other hand, were often very effective and easy to reuse. Complicated generics (including generics within generics within generics) were seldom worth the cost unless the designer and sole user were one-and-the-same, and the designer was well above-average in terms of proficiency and experienced at writing generics. That's not to say that we didn't have proficient programmers. With 100 or more programmers, just don't expect everyone to be a generics expert and design generics well.

Our best use of complex Ada generics involved data logging and retrieval software. This software utilized a high number of generics starting with primitive types (strings, integers, reals) and built up by instantiation into complex, compound record structures in various sizes and formats. This worked very well, provided a single designer/programmer was responsible for both the generic capability and it's uses.

Other good choices for Ada generics were design elements which clearly had a high degree of parallelism, such as our communications package which treated all messages the same, regardless of individual message formats. These even utilized declare blocks, which instantiated the generic on-the-fly for differing sizes or other record discriminants based on run-time values. These met with good success and surprisingly good performance on VAX Ada. Poor choices included the hardware simulators, which attempted a very high degree of generics (>50% of code was within a generic) which suffered from severe performance penalties and lack of flexibility in dealing with specific hardware behaviors.

Coding to a common source template is actually a low-tech form of reuse that should not be overlooked. It worked very easily (as long as the template was correct) and served to promulgate good examples for coding and error-handling. Templates were used for declaring, sending and receiving message objects. They worked well, until limitations in the templates were found. A more extensive effort in developing the template would have payed off handsomely in our experience.

Avoid "Monolithic" Ada packages. Trying to be all things to all people will most likely be nothing to anybody. Thinking that object-oriented design translates into "throw everything into one package" is similarly misguided. Use a layered approach instead. Define a package with just type definitions. Then define a package that provides basic operations on these types. Define higher-level packages as necessary to define more complex operations, building on lower-level packages. A careful architecture like this can help you reap big reuse benefits as new uses are found.

Following this approach allows different programs to access the object at different layers of abstraction. Some just need a type definition. Others need basic routines to manipulate the types. Some need advanced routines composed out of basic routines. Others could benefit from automatic initialization of objects at elaboration time (tends to be very trouble-prone, should be carefully controlled by a standards committee). All uses of a complex object, especially potential future ones, may suffer if the only view presented is a single complete monolithic view. A program wishing access to a type definition ends up with pages of "hidden", unused code and data, and maybe even automatic creation/initialization of objects at startup time, referencing databases defined on one computer and not others.

Variable-length strings were another good reusable package. We implemented them with a generic package, pre-instantiated sized to 256 characters. Use of a pre-instantiated package allowed easier sharing of types. However, this also encouraged waste (programmers were encouraged to use 256-byte strings where only 16 characters were necessary).

Ada Architecture issues

Error handling was our number-one architecture problem. We definitely could have benefitted from better up-front design and more prototyping. Ada tasks complicated the error handling picture drastically.

There is a lot of functional overlap between the capabilities provided by Ada exceptions and those provided by VAX/VMS Condition handlers. There were points of interference or undesirable interplay between the two as well. You need to design error handling into all system service calls. Know which exceptions are worth handling, and which you WANT to be unhandled (because they show up obvious coding or environment problems).

Taking advantage of the operating system's capabilities for calling stack tracebacks on unhandled exceptions, for example, can provide lots of power for debugging. These are especially useful if integrated into the debugging environment, as is the case with most DEC/VAX software.

Concurrency

Ada Tasks

Much fear was generated during early design phases concerning the trade-offs between concurrent operating system processes, and concurrent Ada tasks. During implementation, use was made of both single and multiprocessor machines, with varying results. Software testing and modification history have allowed us to construct better guidelines for process versus task trade-offs. In many cases, processes were used as an aid to work breakdown rather than based on strong architectural need. In some cases these choices caused problems later, and limited the range of available solutions for requirement or design changes. Ada tasking would have been more flexible.

However, increased use of Ada tasking would have required a different development support structure. This support structure would have had to allow separate development and testing of task-based functional work packages independently. The tasks could then be integrated into a single process resulting in a more reliable system.

In general, tasks were well-used and caused relatively few problems. Among the problems were prioritization, blocking, proliferation of tasks required to synchronize between other tasks, and increased rigor in defining/testing the tasking architecture. Tokens (Ada "private" objects containing pointers and flags used in the interface packages between application and service layers) were used to define message addresses. These later became a problem since they were not designed to be shared, yet were shared in some application programs among various tasks. The sheer number of tokens used in the system prevented us from embedding a task within all token types for synchronization (because of the amount of memory used for task stacks, etc.), but we later embedded "token in-use" flags to help detect instances of sharing. Earlier recognition of the problem would have allowed a range of more elegant solutions.

The following are some additional observations regarding Ada tasks:

- Task context switches are a LOT faster than process context switches. If you're thinking of adding more processes, tasks are better. However, processes are easier to split up the work among multiple independent programmers. Tasks in the same process require more programmer coordination during development.
- Tasks are like lawyers. If you have no tasks, you probably won't need any. However, once you have two tasks, you will probably need another five or ten more to handle coordination between those two tasks plus synchronize any shared inputs, outputs, resources, etc. This means that if you start out thinking that you'll write a program with a few tasks, you'll probably end up writing lots. However, this didn't appear to have been a problem. The number of tasks did not affect performance as long as they were event-driven. You may have to spend more time maintaining relative priorities of tasks as the number increases.
- We avoided PRAGMA TIME_SLICE, since we understood it to add significant overhead. We were successful in avoiding it. Several times we were tempted to use it to alleviate other tasking problems, but it was never absolutely necessary and in the end was successfully avoided.
- Multiprocessor problems were encountered, which required us to use PRAGMA SHARED and PRAGMA VOLATILE, which are implementation dependent. These relied on architecture-dependent features of VMS processors. The features worked well in our two-CPU environment.
- We would have liked to prioritize different entry points in the same task (e.g. to handle the same type of rendezvous, but from different sources), but Ada doesn't allow it. We found a kludgy way of doing it. Instead of attempting reuse, we should have duplicated the task code (i.e. via task types) and prioritized them differently. Maybe we did this because we were attempting excessive reuse, or we were afraid of proliferating tasks. Simpler would have been better.
- We worried a lot about "fairness" of tasking, however all fears appeared to be groundless. If you're worried about fairness of tasking, what you really may be worried about is that you need more CPU power. Or you may have tasks polling when instead you need to turn them around into an interrupt-driven approach.
- Beware of non-reentrant servers, services, etc. Accesses to Rdb, the relational database we used, had to be serialized by routing all task's requests through a single Rdb server task (gateway) which in turn provided

the sole control of the Rdb server. This is a fairly common problem interfacing with non-Ada facilities for which you should watch. Our COTS Graphical User Interface (GUI) non-reentrancy problem was solved with the opposite approach. We ran four copies of it, one for each operator window.

- There was still some question for us about what Delay 0.0 really did, or if it was necessary. It was documented as a way to break the execution of a long-running task and allow a context switch to another waiting task. When we attempted to verify this behavior through benchmarks however, we met with mixed results. We eventually opted not to use the feature. Instead we broke problematic long-running tasks into multiple shorter tasks.

  We also had reports of problems with the fairness of allocation of CPU time among tasks. When we investigated with benchmarks, however, all we found were problems with the benchmarks. For each case of purported probems with Delay 0.0 and tasking fairness, programmers who thought they had a problem with an Ada feature were instead using too much CPU time. The ultimate fix was to rearchitect the program to respond to events or Asynchronous System Traps (ASTs) rather than poll.

Compile-time vs. Run-Time Binding

- You can use unchecked_conversion to convert between system.address and object_access types. You'd better be very careful when using this, though. A LOT of errors were committed in this area. Need careful code review and on-the-job indoctrination, perhaps through programmer peer group inspections/walkthroughs, etc. Watch out for things like unintentionally overlayed objects and other C code type pointer errors.
- Anytime you use access types or system addresses in variables it opens the door for memory leaks around allocation/deallocation.
- The Ada compile-time binding of record types was an early problem when data logging record types were very volatile. Many low-worth recompilations were performed. Configuration management and test computer system performance were impacted by the need to accept the many new executables images that were generated. A run-time-binding architecture might have been better in these highly volatile report-writing cases. Once the formats stabilized, the structure did provide for ease of checking. Compilation tests for code impact to changing interface or record format become both routine and precise.

Message Passing Architectures

Ada Interface Definitions

Internal interface definitions, between computers and software subsystems, were captured in Ada. In most cases, representation clauses were not used. Instead the message record definition code was reused in each subsystem. Software configuration management mechanisms ensured that interfaces were modified consistently. This was reliable since all computers used the same hardware architecture and the same compiler.

Platform Dependencies

Operating System Dependencies

Many unknown, unforseen platform dependencies cropped up during the development and test phases. In many cases, these problems were the most astounding and difficult to predict of any we encountered. There is a high degree of functional overlap between the Ada compiler/language run time environment (VAX/VMS Ada 2.2–41 at this writing) and the host/target operating system (VMS 5.5–1). This overlap caused problems in error handling; Ada exception handling interfered with the generation of otherwise automatic operating system calling–tree tracebacks. It also appeared in process management (computer operators couldn't reliably cancel processes with some types of tasking structures), and debugging (generics and tasks increased difficulty of source–level debugging and thus were unpopular with programmers). While many of these are platform–dependent, they point to the overall problem of overlap between Ada's functionality and the functionality of the operating system upon which it's running. If you're running on a bare–bones processor, or a primitive operating system, then there may be little or no problem. Using a sophistcated and feature–rich operating system like VAX/VMS, on the other hand, can lead to limitations and unforeseen problems when you use Ada's advanced features and the operating system's advanced features in the same program.

We ended up having our DEC consultants write a sophisticated assembler routine embedded in each executable which detects unhandled exceptions in any task, forces a traceback, and terminates the image. This has provided us with vastly improved turn–around time for fixing fatal errors found during testing.

Some particulars we found:

- The VAX Ada Run Time Library disables certain features of VMS (like the capability of a computer operator to stop a process gracefully, unless you've coded–in your own user–defined exception handler and a means to signal termination). Also, VAX Ada's memory deallocation/stack unwind during exception propogation interfere with VMS's capability to do a call tree traceback, which would otherwise have shown a stack dump from the line raising the problem all the way back up to the top of the program. This was especially troublesome when some tasks failed due to unhandled exceptions, (coding errors), but other tasks and the process as a whole, continued to function, making it difficult to detect and isolate the problem.
- Writing debug or error messages using Put_Line caused a performance problem in real–time processes, when all tasks in the process hang behind an operating system output request

queued to the disk device. We couldn't tolerate this in many of our hard–real–time executables, so we converted these into shared memory messages between the real–time processes and a lower priority server process, who performed output on behalf of the real–time processes.

- We used tuned Record Management Services (RMS) Input/Output instead of vanilla Ada TEXT_IO or SEQUENTIAL_IO. This was because of the need for heavy–duty tuning, including buffering control and management. We implemented a Mixed I/O–like capability using discriminated records, where each record in the file contained its own embedded record format identifier. This worked quite well, except when the formats were under early development and changed often. Then backward compatibility of current software and previously archived data files became tedious.
- SHARED images (a sophisticated VMS Feature) would have been good to use in certain areas where reusable code made up almost a Megabyte of each executable image, but the integration with Ada was not smooth. By the time we developed a good working approach we had to abandon it because of the retrofit cost. This might have helped Ada's performance some, in decreasing the memory required. If it could have been done earlier with benefits amortized over more of the development phase, it would have saved money and time. We had initial misgivings about the ability to debug an installed shared image, which later appeared to have been unfounded.
- VMS has a very nice software pseudo–interrupt capability (Asynchronous System Traps or ASTs). The Ada run–time library uses these to do it's own synchronization, and instead converts each application AST into a task rendezvous. As a result, running Ada as a part of a "real" AST such as in a call from a device driver written in another language was a difficult proposition (couldn't use tasking, perform any I/O, etc.). However, the run time libary's conversion of ASTs to tasks (PRAGMA AST ENTRY) was quite accessible to programmers. Tasks seemed to be quite easy (and even natural) to use for this purpose. This enabled anyone to make use of ASTs, whereas without this we probably would have had to restrict their use to an elite group of the most experienced programmers.
- Make use of platform capabilities. Don't be an Ada zealot, thinking you have to write pure Ada code and duplicating functionality otherwise available more cheaply or efficiently in the operating system (100% code portability wasn't an issue for us – and it may not be for you either). Examples are character and numeric utilities. Just write good (portable) package specs, and implement the bodies of these in the most efficient manner, even utilizing operating system service calls or non–Ada utility packages. This is especially appropriate on complex instruction set computers (CISC) like the VAX. You can always rewrite

C-5.

the bodies for each new platform to which you port. That way you've addressed performance, reusibility and reduced risk while making good progress and leveraging the capabilities and strong points of your underlying platform.

COTS Dependencies and Integration

During the proposal phase of STGT we identified several areas where Commercial Off-The-Shelf (COTS) software could be used. We then deleted costs based on the difference between developing the application from scratch and the cost of the COTS product. However, the following concerns arose:

- We did not allocate necessary additional costs to continually evaluate and incorporate periodic updates/upgrades of these COTS products. This turned out to be a big ticket item over the life of STGT.

- Purchase good quality COTS bindings. This is a LOT of work. Availability/maturity of Ada bindings should be a significant discriminator during COTS evaluations (e.g., XWindows/Motif binding problems, Distributed File Service (DFS) bindings, device driver bindings, etc.). As usual, productivity may be gained for many at the expense of hard work by a few, or by the purchase of a proper bindings. Consider the trade-offs.

Performance

Ada Performance Characteristics

Many performance problems were encountered which required various mitigation approaches. Performance modelling was only as good as the input received (much guess work was necessary early on in the life-cycle). This lead to big surprises and varying types of late changes. Eventually larger CPUs and more memory were purchased.

There appears to be a SERIOUS dichotomy in Ada between coding for performance and coding for what most consider to be a "good" Ada style. "Good" Ada was subject to our interpretation of the current literature and to the lessons developed during prototypes by the Ada Core Team. What might be considered "good" Ada of course will change over time. Examples are:

- The generic string package was pre-instantiated for (discriminated record structures) of 256 bytes. This affords maximum reusability and similarity, but appears to waste memory and disk space due in certain cases to needlessly large structures.
- Proponents of "good" Ada often stress deeply nested procedure calls for modularity and reuse. "Fast" Ada is often relatively flat, with a shallow call depth.
- "Good" Ada makes maximum use of local variables. "Fast" Ada allocates

variables once in package bodies, then carefully reuses them within package procedure and function bodies.

- "Good" Ada makes maximum use of Generics. "Fast" Ada avoids complex generics.
- Good Ada makes minimum use of implementation-dependent PRAGMAs. Fast Ada utilizes some PRAGMAs, e.g., PRAGMA ELABORATE to force elaboration of packages before the routines are called for real-time execution.
- As a result of the apparent quandry between "good" and "fast" Ada, it seems that Ada right out of the object-oriented training book can be quite slow. You either need to allocate a bigger CPU, know very accurately the performance characteristics in advance, or plan on a tuning phase to increase the performance of your code once it's written.

Schedule pressures made us opt for the quickest solutions in most cases, that is, larger CPU's. We had some success in optimizing Ada for performance. In some cases the re-coding or reimplemetation of a component saved 50-100% of CPU or Memory resources. In one case it saved a factor of 5X CPU for a compute-intensive satellite orbit prediction function.

Configuration Management

Ada Configuration Management

- Ada dependencies are GRAPHS, most library structures/directory hierarchies are TREES. Therefore, if you lock yourself into a library structure that mimics the Ada dependency structure, you'll be disappointed eventually. We used a simple tree of SHARED code at the top, with CSCIs or subsystems below.
- Sublibraries were used versus the VAX Ada Compilation System (ACS) ENTERED units. This allowed automatic recompilation for dependent units when root units changed. The downside was that massive recompilations were forced when not all dependent libraries (and groups using those libraries) were ready to see the change. An alternative approach might have been to develop a tool for automatically re-entering changed units into dependent libraries. That also could have allowed for library dependencies more complicated than a tree.
- We used separate/duplicate libraries to reflect differing levels of software test maturity. For instance, we had one shared set of libraries in which developed code. We only updated the reused components of that library once a week. People affected by interface changes only had to support (or suffer) changes once a week.
- We could have used hierarchical libraries for test, but the computational requirements were too great. Our development CPU resources were never great enough to compile the same source code multiple times for different hierarchical libraries supporting different test maturities. Consequent-

ly all tests were forced to the same maturity – fresh from the programmer.

- We had to write a program to extract a cross-reference containing "where-used" information. ACS did not provide this information.

Ada Compilation Performance

We did a LOT of work to improve compilation speed. Some of the things we did were:

- Faster CPUs – went from VAX 8250s (1.5 MIPS) to VAX 6610s (25 MIPS).
- More memory – from 64 to 256 Mb
- Tuning of system quotas, batch queues etc.
- RAM DISK and/or semiconductor disk for shared code Ada library (most critical compilation library)
- Spread I/O over multiple disks to reduce bottlenecks
- We didn't persue but maybe should have experimented further with the effects of smaller and larger directory/library sizes on compilation speed.

Ada Compiler

- We found relatively few bugs. Most were in code generation, a few for floating point types and others which optimized away variables or code. One involved different Ada library unit interfaces depending on whether code was compiled in debug or non-debug. All were resolved in quite good order by excellent DEC support. The lesson was that compiler maturity (for VAX/VMS Ada) was not a risk factor. We also learned that run-time (vs. compile-time) bindings for certain rapidly and persistantly changing functions would have been a much better design from an operational and CM point of view.
- On the other hand, the maturity of ACS was less evident. We have had numerous problems and "features". A good Ada Program Support Environment would be greatly appreciated. We wrote 30,000 lines of "tool" and configuration management scripts. This is significantly more than we anticipated supporting. A good COTS tool available in a timely manner would have been a big productivity enhancement.
- The design of our parent libraries and sublibraries were important. We found ourselves re-creating libraries because library parent/child relationships were hard-coded rather than logical. We redid all libraries with PSEUDO-DEVICE logicals so that successive changes were less painful.

Project Management

Equally as important as the Ada lessons learned were the lessons we learned in managing and controlling a large Ada software development effort. Some of these lessons are:

Standards

- Our Software Standards and Practices Manual (SSPM) was HUGE. Far too big to be understood or enforced.

- Should have made better use of automated standards checkers or pretty-printer tools.
- Should have tailored the Language Sensitive Editor (LSE) more aggressively for our local standards and included more templates
- Standards should be issued, proven, taught, understood, reviewed, reproven, and well documented before any code is written.

Architecture and Schedule

- Allocate the Right CSCIs. We changed the allocation of CSCI's early in the development effort. Changes (reallocations) are difficult to make.
- Avoid Early Split into CSCI Production Groups. We set up a Work Breakdown Structure (WBS) and Management structure on day one. Therefore shifting of work from CSCI to CSCI became a continuous struggle. Work overall system architecture first before parochialism sets in. Set up a mechanism to provide for the overall project good at expense of an individual group.
- Avoid the pressure to accelerate schedules. Believe the "Rule of Tens" (errors found in a later phase take 10 times longer to fix). Missed goals can not be made up. Insist on operation's concepts and equipment (mission equipment) designs prior to software designs.
- View interfaces as a "contract" not as a goal. Interfaces that change are painful.
- Understand tools required and decide on their use well in advance of needs. We developed Configuration Management DCL on-the-fly, did not understand the complexities of Ada CM, and shared interface packages (which are a good idea, but caused massive recompiles). Understand and plan the role of tools throughout the whole lifecycle.
- Define and stick to a fixed methodology. We were guilty of making it up as we go. Much of the heritage we had from our Ada Core Team did not scale up into larger development efforts. Tools did not easily transition between phases.
- Do more prototyping – especially for performance. Make performance estimates based on Executed Lines Of Code (ELOCs) from actual prototypes rather than from Lines Of Code (LOCs) written or predicted to be written. Consider living (non_throwaway) prototypes for broadly used "infrastructure" code.
- Use the right language for the right function. We made some changes to use macro assembler in some critical high frequency applications. Device driver type functions were very slow in Ada as was the high use interprocess communication processes.
- Put Some Teeth into allocating and enforcing performance requirements. We allocated only very high level requirements to the CSCIs for CPU and Memory performance. These were not allocated to lower level components and were therefore untestable and unenforceable.

- Do Code Walkthrus – set aside a team to execute. We relied on peer reviews of code. This became a significant schedule pressure on the CSCI who concentrated more on their own efforts then in a thorough review of another CSCIs code.
- Understand and don't underestimate the entire domain. Understand the performance aspects of the COTS products and prototype their use. Errors in COTS are harder to fix because of 3rd party involvement. Work with COTS can begin earlier since design effort is usually not required. The effects of the operating system and hardware platform are significant, protopying and an early start is recommended.
- Know what you are buying and where to use it. For Example, Booch components were excellent at improving productivity. However know their performance characteristics before deciding where to use them and other similar COTS software.
- Hire Experts – utilize vendor consultants. On site expertise is the best way to fix problems and to get preferential access to vendor guru's and other experts. Often you fix problems before they happen, since consultants can help you with that most difficult assessment, determining what it is that you don't know.

These Lessons Learned represent only a small subset of the potential data that can be gleaned from GE's experience on STGT. The main lesson to take away from this paper is that the language, platform, COTS products, tools, etc. are just a means to an end and in themselves are responsible for neither success nor failure.

STGT Ada Lessons Learned

STGT
Second TDRSS
Ground Terminal

Dec 2-3, 1992

NASA Goddard Software Engineering Laboratory

Software Engineering Workshop

STGT Project
Ada Lessons Learned

Tod Kahrli
Bill Manley
Scott Brown
Brian Bauman
Paul Usavage
Don Nagurney

Chart 1

STGT Ada Lessons Learned
Agenda

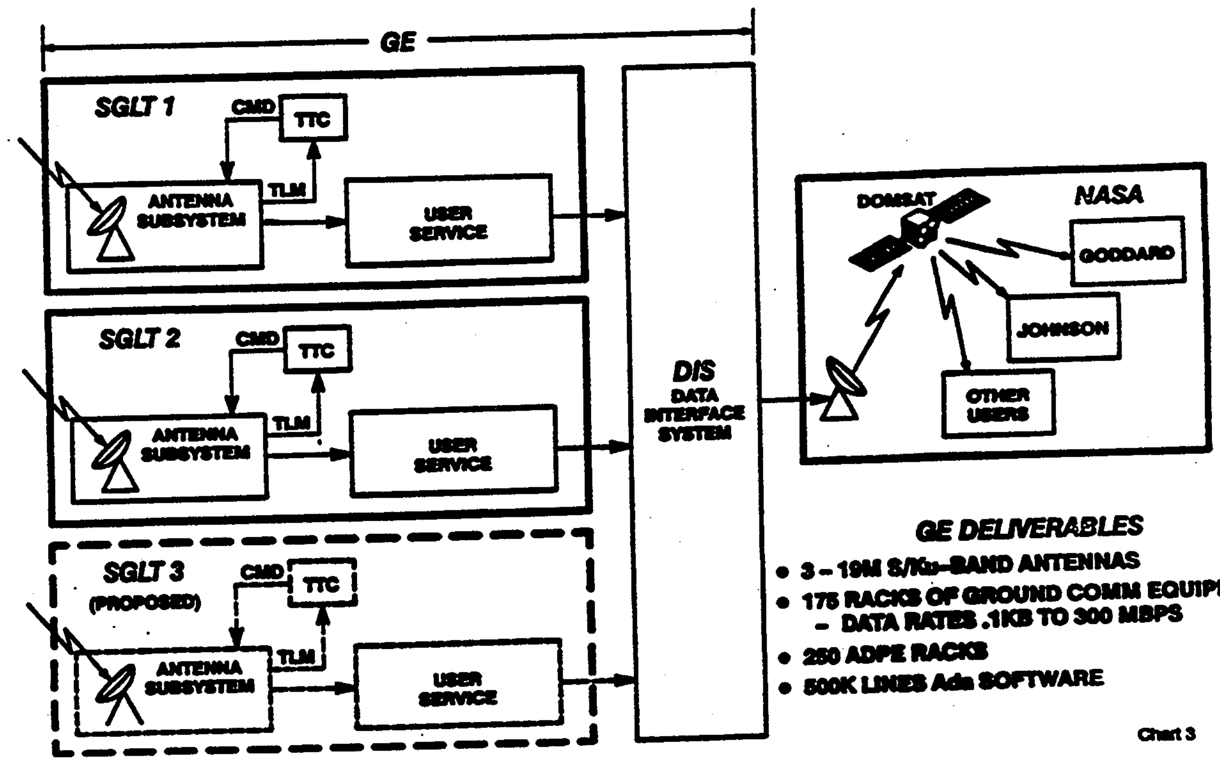
STGT
Second TDRSS
Ground Terminal

Dec 2-3, 1992

- **Project Overview**
- **Software Configuration**
- **Software Metrics**
- **Ada Project Management Lessons Learned**
  - **Project Schedule/Structure**
  - **General Issues**
  - **Performance/Sizing**
  - **Reusability**
- **Ada Lessons Learned**
  - **Generics**
  - **Tasking**
  - **COTS/Platform Dependencies**
  - **Package Structuring/Record Formats**
  - **Exceptions**

STGT Ada Lessons Learned
Project Overview

STGT
Second TDRSS Ground Terminal
Dec 2-3, 1992



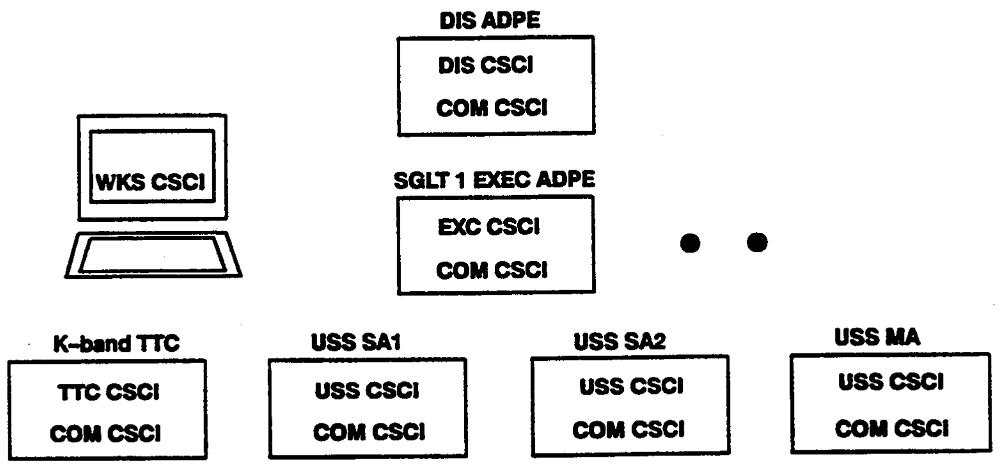
GE DELIVERABLES

- 3 - 19M S/Ku-BAND ANTENNAS
- 175 RACKS OF GROUND COMM EQUIPMENT
  - DATA RATES .1KB TO 300 MBPS
- 250 ADPE RACKS
- 500K LINES Ada SOFTWARE

Chart 3

STGT Ada Lessons Learned
Software Configuration

STGT
Second TDRSS Ground Terminal
Dec 2-3, 1992



COM CSCI	Common Services
DIS CSCI	NCC Interface
EXC CSCI	SGLT Scheduling
USS CSCI	Equipment CMD/MON
TTC CSCI	TDRS/Antenna Control
WKS CSCI	Operator Interface
SIM CSCI	SMTF Simulation
MDS CSCI	SMTF Tools

Chart 4

STGT Ada Lessons Learned
Software Metrics

STGT
Second TDRSS Ground Terminal
Dec 2-3, 1992

CSCI	Size (LOC)	Hours[1]	LOC/Hour
TTC	100000[3]	115000[2]	.86
DIS	76000[3]	79000	.96
USS	71000[3]	58000	1.22
EXC	26000	23000	1.13
WKS	152000	56000	2.7
COM	23000	37000	.62
MDS	100000	36000	2.77
SIM	40000	40000[3]	1.0
Total	588000	444000	1.32

1 – Requirement Analysis thru Software Test
2 – Includes Cost of Common Ground Control/Monitor and Fault Detection
3 – Includes Common Ground Control/Monitor and Fault Detection

Chart 5

STGT Ada Lessons Learned
Ada Project Management
Lessons Learned

STGT
Second TDRSS Ground Terminal
Dec 2-3, 1992

- Ada Project Management Lessons Learned
  - Project Schedule/Structure
  - General Issues
  - Performance/Sizing
  - Reusability

STGT Ada Lessons Learned
Ada Project Management
Lessons Learned
Project Schedule/Structure

STGT
Second TDRSS Ground Terminal
Dec 2-3, 1992

- **Allocate the Right CSCIs**
  - **Changes (Reallocation) are Difficult to Make**
- **Avoid Early Split into CSCI Production Groups**
  - **Work Overall System Architecture First**
  - **Set up a Mechanism to Provide for the Overall Good at Expense of an Individual Group**
- **Avoid the Pressure to Accelerate Schedule**
  - **Believe the "Rule of Tens"**
  - **Missed Goals Can Not Be Made Up**
  - **Insist on Operation's Concepts and Equipment Designs Prior to Software Designs**
- **View Interfaces as a "Contract" not as a Goal**
  - **Interfaces That Change Are Painful**

Chart 7

STGT Ada Lessons Learned
Ada Project Management
Lessons Learned
General Issues

STGT
Second TDRSS Ground Terminal
Dec 2-3, 1992

- **Understand Tools Required and Decide on their Use Well in Advance of Needs**
  - **CM Developed DCL on-the-fly : did not understand the Complexities of Ada : Shared Interface Packages (A Good Idea) Caused Massive Recompiles**
  - **Understand and Plan the Role of Tools Throughout the Whole Lifecycle**
- **Understand and Don't Underestimate the Entire Domain**
  - **COTS**
  - **DEC/VMS**
- **Prototype and Utilize Prototype Code Everywhere**
- **Hire the Right People Then Train/Train/Train**

Chart 8

STGT Ada Lessons Learned
Ada Project Management
Lessons Learned
General Issues

STGT
Second TDRSS
Ground Terminal
Dec 2-3, 1992

- **Define and Stick to A Fixed Methodology**
  Define in Advance and Don't Experiment
  Educate User's
- **Keep the SSPM Simple - Useful and Easy to Enforce**
- **Do Code Walkthrus - Set Aside a Team to Execute**

Chart 9

STGT Ada Lessons Learned
Ada Project Management
Lessons Learned
Performance/Sizing

STGT
Second TDRSS
Ground Terminal
Dec 2-3, 1992

- **More Prototyping - Estimates Based on Executed LOCs**
- **Complex Generics Proved to be Extremely Slow**
- **Understand Compile and Link Process (e.g. Compiler Eliminates Dead Code But Linker Does Not)**
- **Use the Right Language for the Right Function**
- **Bad Ada Is Real Baaaaad**
- **Put Some Teeth Into allocating and enforcing Performance Requirements**

Chart 10

STGT Ada Lessons Learned
Ada Project Management
Lessons Learned
Reusability

STGT
Second TDRSS Ground Terminal
Dec 2-3, 1992

- **Know What You are Buying and Where to Use It**
  Booch Components – Not Optimized for Performance
- **Don't Attempt High Level Generics Yet**
  Ground Equipment Simulation Is the Wrong Choice
- **Provide for Project Wide Reuse Czar**
  Avoid Parochialism
  Proactive Search for Opportunities

Chart 11

STGT Ada Lessons Learned
Ada Lessons Learned

STGT
Second TDRSS Ground Terminal
Dec 2-3, 1992

- **Ada Lessons Learned**
  - **Generics**
  - **Tasking**
  - **COTS/Platform Dependencies**
  - **Package Structuring/Record Formats**

Chart 12

STGT Ada Lessons Learned
Ada Lessons Learned
Generics

STGT
Second TDRSS Ground Terminal
Dec 2-3, 1992

- **Can be a Performance Problem**
- **Are to Debug with Interactive Source Level Debugger**
- **Keep Small : Don't Attempt a Reusable Ground Station**
- **Restrict Usage to Types as Formal Parameters**
- **Keep Them out of the Hands of Amateurs**
  - **Limit to Your Most Experienced People**
  - **Review/Review/and Then Again – Prototype Performance**

Chart 13

STGT Ada Lessons Learned
Ada Lessons Learned
Tasking

STGT
Second TDRSS Ground Terminal
Dec 2-3, 1992

- **Mistrusted at First – Found Many Appropriate Uses**
  - **Understand the Target Environment/Prototype**
- **Provide for Terminate Alternatives – Make Sure a Parent can Terminate Children**
- **Exceptions Must Be Propagated Upward (Free Running Tasks Need Some Control)**
- **Don't Substitute Tasks Where Procedures Would Suffice**
- **When Using Tasks – Centralize Control (one writer)**
- **If You Plan on a Few Expect Many More**

Chart 14

STGT Ada Lessons Learned
Ada Lessons Learned
COTS/Platform Dependencies

STGT Second TDRSS Ground Terminal
Dec 2-3, 1992

- **Understand Compiler/Linker and Their Interaction**
  - **Don't Count on Default Order of Elaboration**
- **Understand The Whole Domain**
  - **VMS Services Better Than Ada Features**
- **Pick COTS With Ada Bindings (Avoid Multiple Translations)**
- **SQLMODS Proved to Be Workable Interface**
  - **Imbedded SQL was Impossible to Debug**
- **Hire Experts – Utilize Vendor Consultants**
- **Product Upgrades are Large Undertakings and Come at the Most Inopportune Times**
  - **Properly Plan for and Fund Product Upgrades**
- **Avoid the Creation of Processes Without Justification**

Chart 15

STGT Ada Lessons Learned
Ada Lessons Learned
Package/Record Formats

STGT Second TDRSS Ground Terminal
Dec 2-3, 1992

- **Limit Scope of Packages – Don't Try to Encapsulate and Entire Object in One Package**
  - **Use Multiple Packages – Each With a Purpose**
  - **Know the Intended Use of the Packages (e.g. Senders vs Receivers)**
  - **Avoid Monoliths**
- **Don't Put Database Access into Interface Packages**
- **Don't Combine Loosely Related Types**
- **Create Null Instances of a Type as an Initial Value**
- **Avoid String Types – Usually Masking an Enumerated Type**
- **Renaming – Many Differences of Opinions: Be Careful**

Chart 16

STGT Ada Lessons Learned
Ada Lessons Learned
Exceptions

STGT
Second TDRSS
Ground Terminal
Dec 2-3, 1992

- **Use Only For Real Errors – Very Expensive for Use As GOTOs**
- **"When Others" obscures origin of exceptions**
- **Understand and Plan for Unhandled Exceptions**
  - **Tracebacks and Stack Dumps are Good Debugging Tools**
  - **Process/System Dumps Have Their Place**
- **Specify and Design Expected Levels of Error Handling**

Chart 17

STGT Ada Lessons Learned
Summary

STGT
Second TDRSS
Ground Terminal
Dec 2-3, 1992

- **Project Pressures Force Old Habits to Return**
- **Solidify Interfaces Under Penalty of Death**
- **Prototype Everything and Always**
- **Enforce Performance Allocations**
- **Focus Reuse and Dedicate Resources**
- **Restrict Generics**
- **Don't Be Afraid of Tasks**
- **Understand the Domain – and Hire Where Necessary**
- **Limit Scope of Packages**
- **Be Prepared to Upgrade COTS**

Panel: Is Ada Dying?

Marv Zelkowitz, University of Maryland, Facilitator

Stu Feldman, Executive Director of Computer Systems Research, Bellcore

John Foreman, Director of STARS Program, Department of Defense

Susan Murphy, AAS Software Manager, IBM

Tom Velez, President and CEO, CTA

Panel: Is Ada Dying?

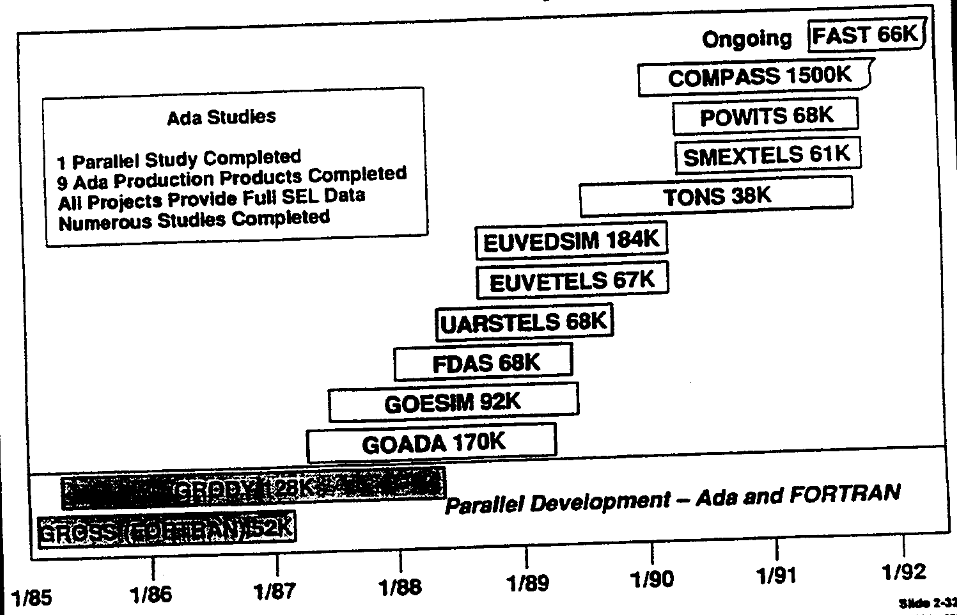
- Facilitator:
  - Marvin V. Zelkowitz, NIST/CSL and Department of Computer Science, University of Maryland
- Panelists:
  - Stu Feldman, Executive Director, Computer Systems Research, Bellcore
  - John Foreman, Director of STARS Program, DARPA
  - Susan Murphy, AAS Software Manager, IBM FSC
  - Tom Velez, President and CEO, CTA

SEL interest in Ada

- Why SEL interest in Ada?
  - SEL has broadest experience with Ada within NASA
  - SEL has collected much data on the use of Ada (as well as many other technologies)
  - SEL has analyzed Ada usage from various perspectives (e.g., see last few Workshop proceedings)
- Results of SEL studies:
  - Value of Ada not unconditionally shown
  - Need to assess current status and plan future processes

PRECEDING PAGE BLANK NOT FILMED

SEL Ada Projects



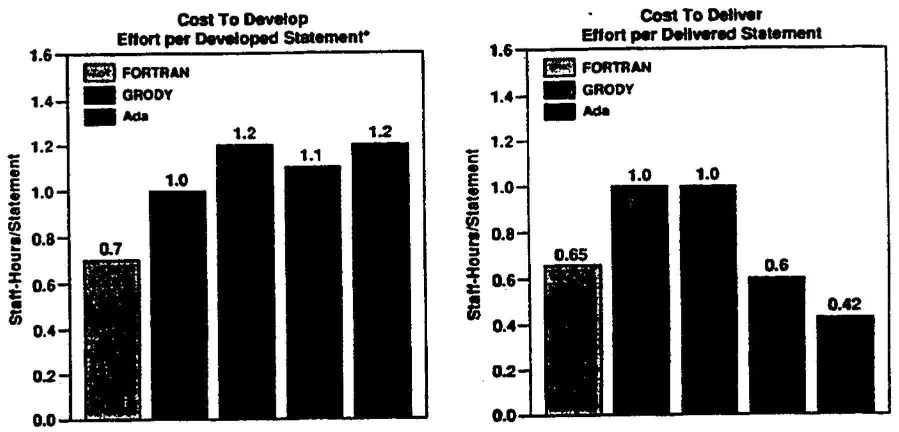
Ada Studies

1 Parallel Study Completed
9 Ada Production Products Completed
All Projects Provide Full SEL Data
Numerous Studies Completed

Slide 2-32
10000284-g050

SOFTWARE ENGINEERING LABORATORY — SEL

Ada (and OOD) Impacts on Cost



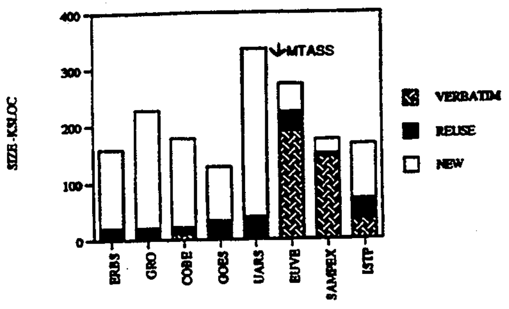
*Ada : Developed Size = 100% New + 30% Old Statements
FORTRAN: Developed Size = 100% New + 20% Old Statements

- Development cost per statement has been no cheaper for Ada
- Resue potential of Ada is significant
- Reuse cost factor has changed in Ada systems

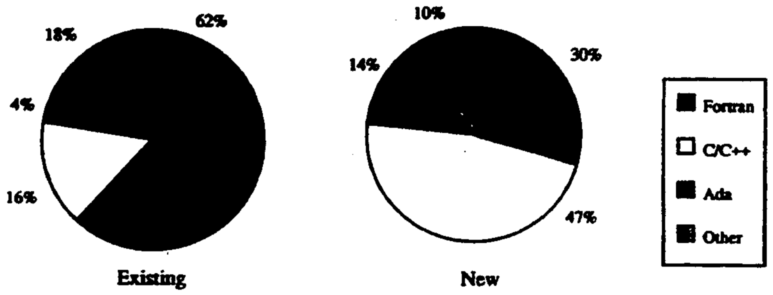
Slide 2-33
10000284-g051

SOFTWARE ENGINEERING LABORATORY — SEL

But FORTRAN reuse is also growing



Language use in Code 500 at Goddard



NASA IBM mainframe Ada evaluation

- Need more development and testing support
  - Two compilers evaluated
  - Multiple source file compilation limited
  - Ada library can be corrupted
  - Inflexible Ada library manager
  - Need better debugger
  - One compiler failed to even compile some modules
- Need improvement in error handling and error messages
- Need improvement in performance
- Result: Could not use IBM mainframe for large-scale NASA Goddard development

Onboard embedded Ada application

- Goal: Dual 1750A processors with shared memory to handle onboard navigation
- Environment: TI 1750A hardware, Tartan cross compiler system on VAX
- Problems: Intermittent communication and shared memory problems. Hardware and software vendors could not solve problems.
- Resolution: Had to fly uniprocessor system with reduced functionality.

Positive attributes of Ada

- Language syntax and semantics are in mainstream language design – an outgrowth of FORTRAN, ALGOL and Pascal
- Language features to aid in large system design, reuse and maintenance (e.g., packages, tasking, exceptions, generics)
- Over 250 validated compilers
- University use growing – 14 Ada textbooks and use at perhaps 10% of U.S. universities (from: November, 1992 Comm. of the ACM)
- Millions of lines of Ada code for commercial non-military applications – Examples: Shell Oil for exploration, Motorola for cellular telephones, Boeing for 747-400, GE for automated steel manufacturing, NTT (Japan) for commercial telecommunications applications, Nokia SoftPlan (Finland) for a banking system, plus others
- Ada-9X revision to solve many of the lingering problems

Negative attributes of Ada

- Hard to learn to use well
- Lack of production quality compilers
- Performance penalty in certain critical applications
- Doesn't handle object oriented design – Impact of C++

Observations

After 10 years of development ...

- Growth of courses and textbooks in Ada seems very slow.
- Does not seem to be a large scale movement to Ada within non-DoD segments of the industry. Most examples are anecdotal.
- Ada does not yet seem ready within the large mainframe environment at Goddard.
- Yet, seems to be a natural attraction to C and C++. Both have attained huge unsupported growth.

Will there be supported Ada products in 10 years?

Summary of issues

- "Many of the perceived problems with Ada were due to the immaturity of early implementations, rather than flaws of the language itself. Some of these perceptions linger, even though mature Ada implementations are available today and most of the previously identified shortcomings have disappeared." – Erhard Ploedereder, Comm. of the ACM, Nov., 1992
- Is Ada today an economically viable language for building software systems?
- If so, for what class of projects is it appropriate?
- If not, what criteria are needed for determining the economic viability of Ada (and when should they be met)

Panel organization

- Opening statements:
  - What is your position and why?
  - What are the objective or subjective criteria supporting your position?
  - What actions should the principles be taking (i.e., DoD, NASA, contractors) and what will Ada be in the next century?

- Each panelist will talk for up to 10 minutes; then a 5 minute comment by panelists on other statements; then general comments or questions from workshop attendees

Feldman 1

Uses and Future

Niche	1980	⇒	2000
Commercial	COBOL	+4GL	+QUERY LANGUAGE
+ C++			
Scientific/Engineering	FORTRAN	+C	FORTRAN 90, C++
Systems	ASM, C	C	C++
Prototyping	LISP, SMALLTALK	C, PROLOG ...	
Embedded/Real Time	ASM, ADA	C, ADA	C++, ADA
S/W Engineering	ADA		C++, ADA, ...?
CS Research	C, LISP	C++, CLOS, ML	

Feldman 2

Sociology

Lifecycle
- Born/Stillborn
- Born Again?

Nurture
- Phoenix/Bride of Frankenstein?

Kinship
- None Allowed

Support System
- Ada Industry $\propto \frac{d^n}{dt^n}$ Defense Budget

Ecology
- Niches and Competition

Feldman 3

Unproven Comparisons

Software Maintainability

Ada > C
Ada > C++

$C \left\{ \begin{matrix} > \\ ? \\ < \end{matrix} \right\} C{+}{+}$

Language Complexity

Ada 9X > FORTRAN 90 > C++ >> C ~ FORTRAN 77

Simple - Compiler Difficulty

Ada 9X > Ada >> C++ >> C

Excellent - Compiler Difficulty

C++ $\gtrsim$ C >> Adas > FORTRAN

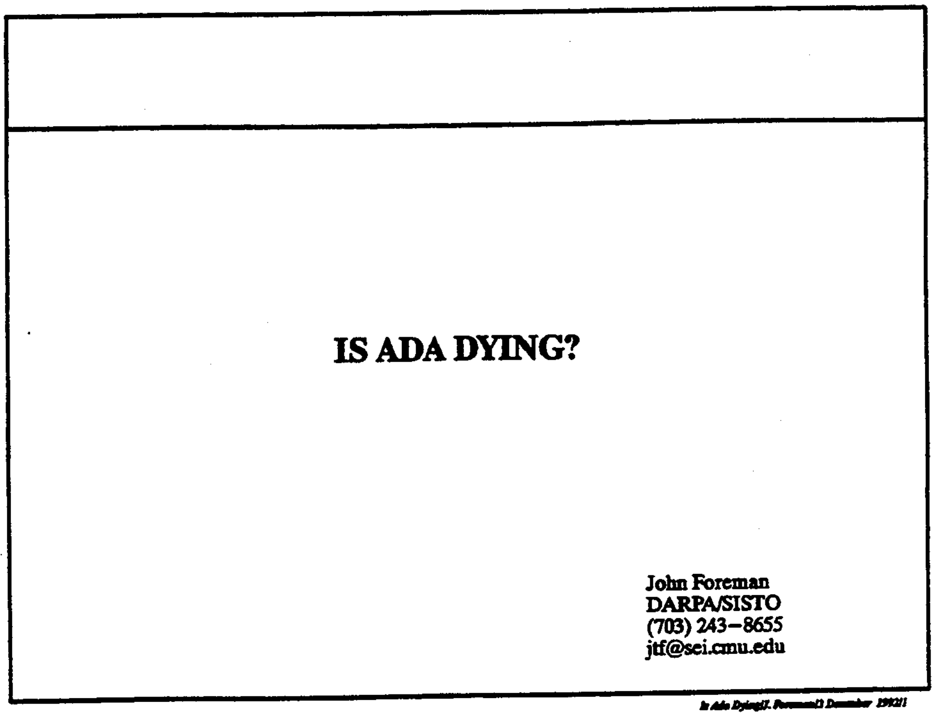
Feldman 4

Ada Properties

+ Complete
+ Supported
+ Sponsored
+ Real-Time
+ Software-Engineering

? Configuration Support

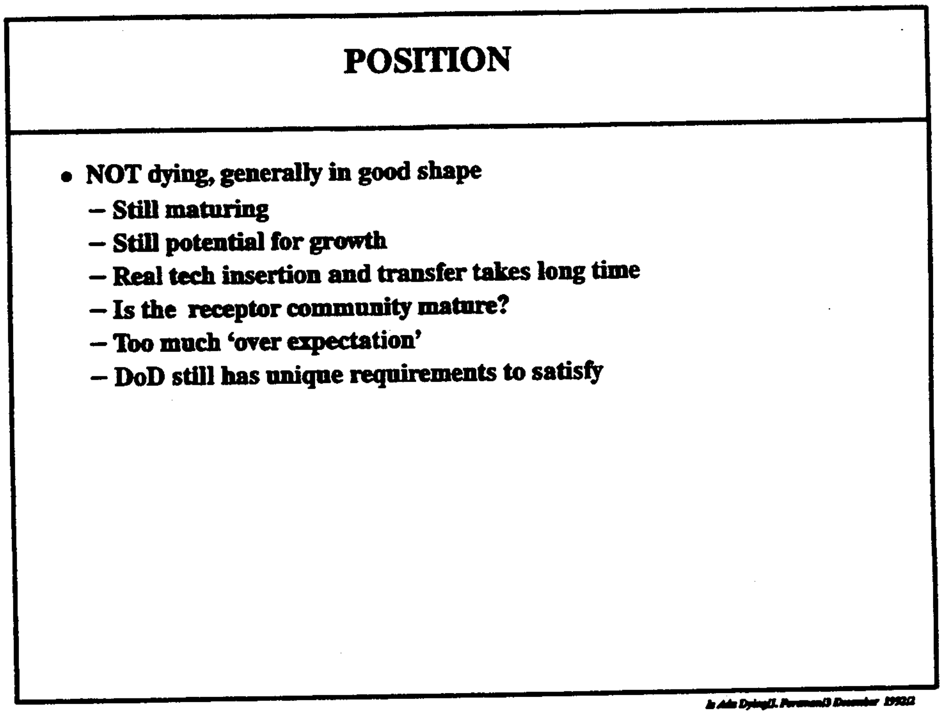
- Syntax
- Garbage Collection
- Complexity
- Software Support
- Use in Systems ("open")
- Love



IS ADA DYING?

John Foreman
DARPA/SISTO
(703) 243–8655
jtf@sei.cmu.edu

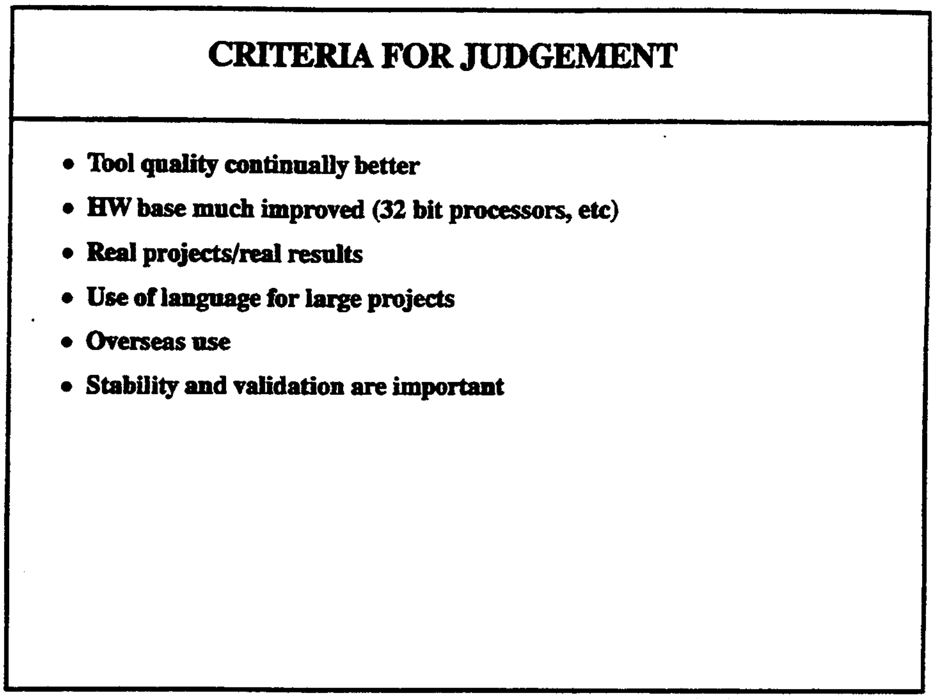
Is Ada Dying?/J. Foreman/3 December 1992/1



POSITION

- NOT dying, generally in good shape
  - Still maturing
  - Still potential for growth
  - Real tech insertion and transfer takes long time
  - Is the receptor community mature?
  - Too much 'over expectation'
  - DoD still has unique requirements to satisfy

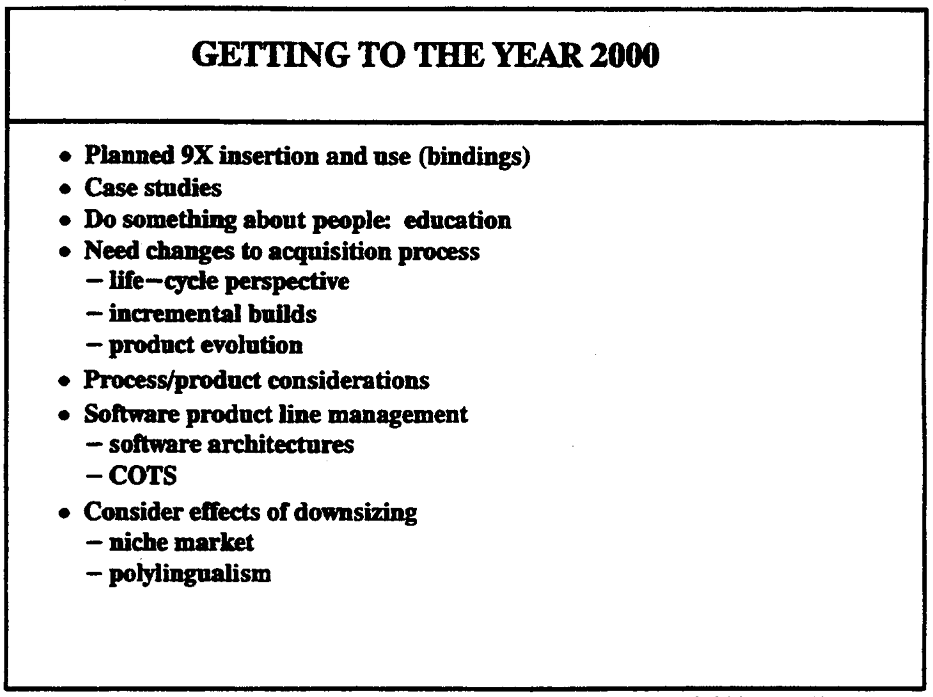
Is Ada Dying?/J. Foreman/3 December 1992/2



CRITERIA FOR JUDGEMENT

- Tool quality continually better
- HW base much improved (32 bit processors, etc)
- Real projects/real results
- Use of language for large projects
- Overseas use
- Stability and validation are important

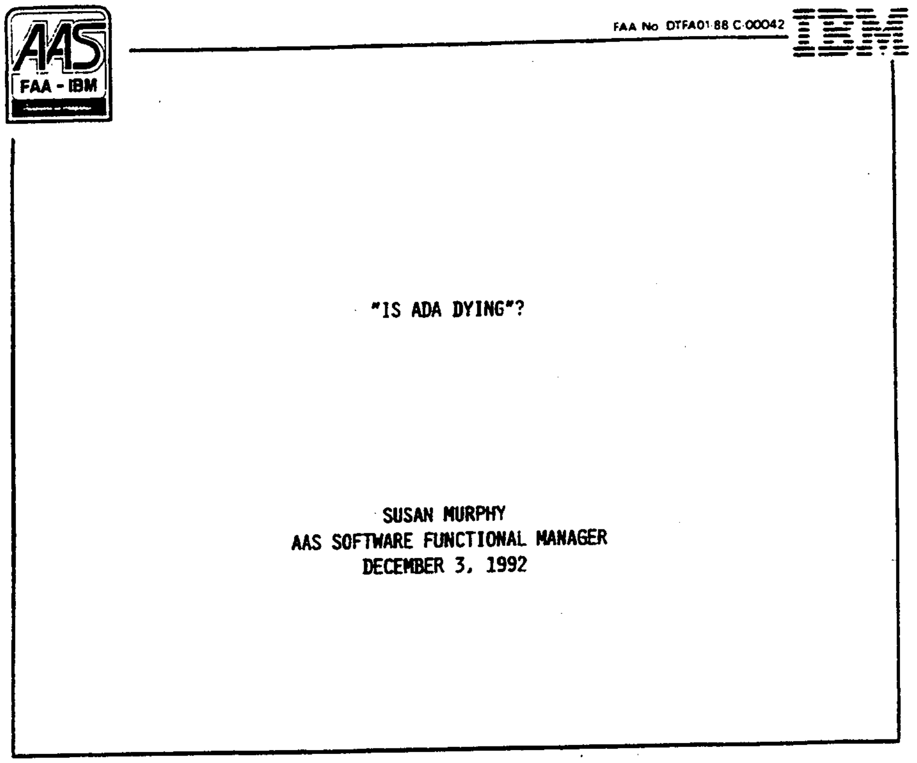
Is Ada Dying/J. Foreman/3 December 1992/13



GETTING TO THE YEAR 2000

- Planned 9X insertion and use (bindings)
- Case studies
- Do something about people: education
- Need changes to acquisition process
  - life–cycle perspective
  - incremental builds
  - product evolution
- Process/product considerations
- Software product line management
  - software architectures
  - COTS
- Consider effects of downsizing
  - niche market
  - polylingualism

Is Ada Dying/J. Foreman/3 December 1992/14

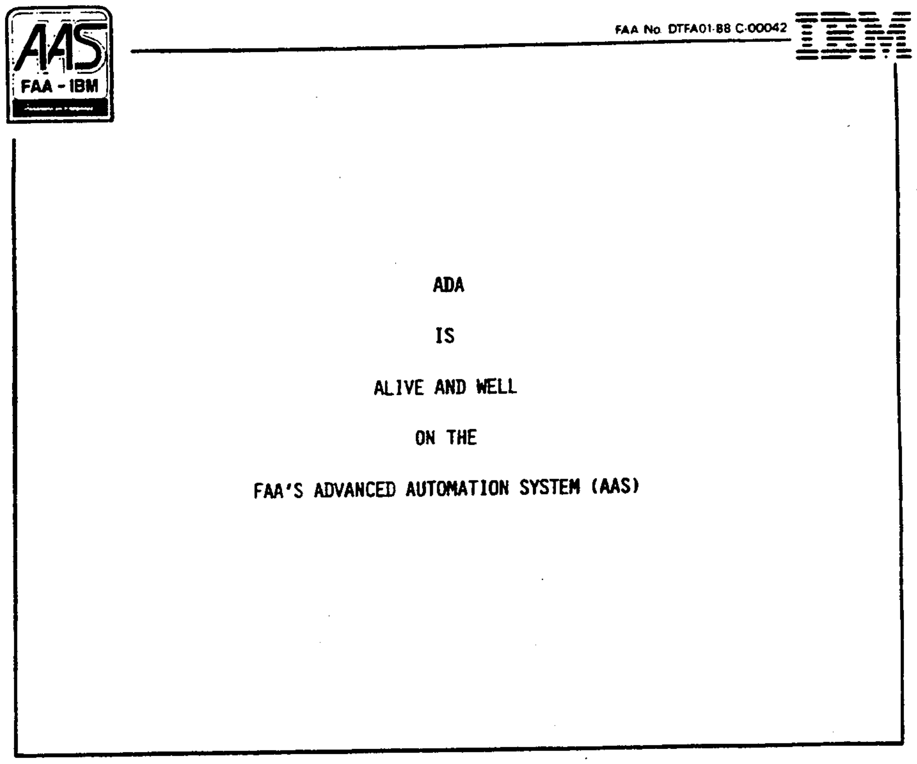


AAS FAA - IBM

FAA No DTFA01-88-C-00042 IBM

"IS ADA DYING"?

SUSAN MURPHY
AAS SOFTWARE FUNCTIONAL MANAGER
DECEMBER 3, 1992



AAS FAA - IBM

FAA No DTFA01-88-C-00042 IBM

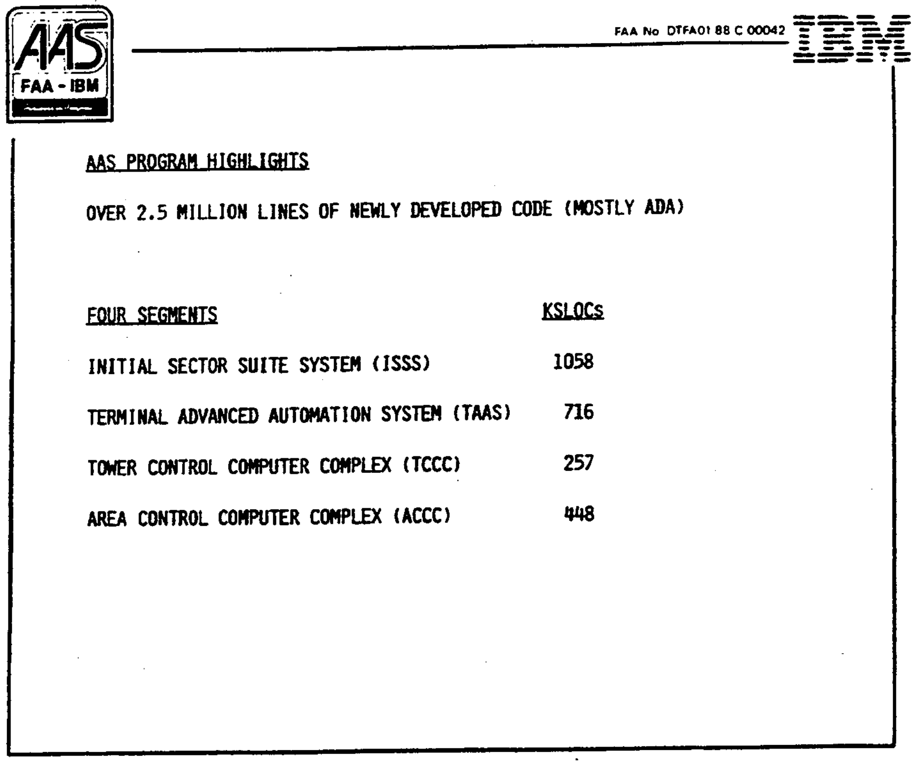
ADA

IS

ALIVE AND WELL

ON THE

FAA'S ADVANCED AUTOMATION SYSTEM (AAS)

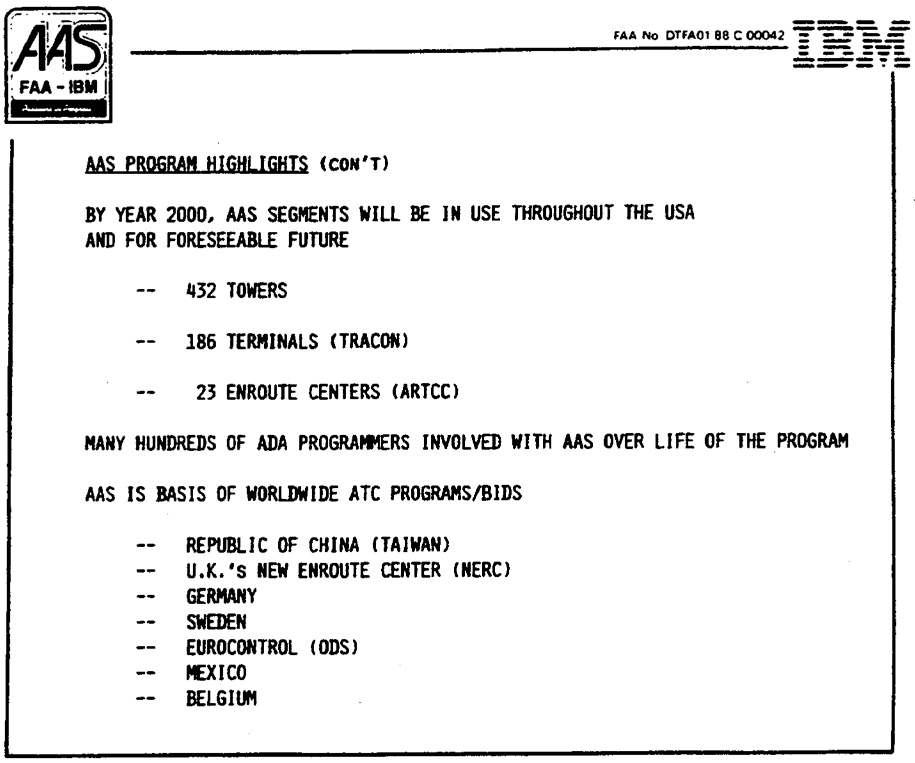

AAS FAA - IBM

FAA No DTFA01 88 C 00042 IBM

AAS PROGRAM HIGHLIGHTS

OVER 2.5 MILLION LINES OF NEWLY DEVELOPED CODE (MOSTLY ADA)

FOUR SEGMENTS	KSLOCs
INITIAL SECTOR SUITE SYSTEM (ISSS)	1058
TERMINAL ADVANCED AUTOMATION SYSTEM (TAAS)	716
TOWER CONTROL COMPUTER COMPLEX (TCCC)	257
AREA CONTROL COMPUTER COMPLEX (ACCC)	448


AAS FAA - IBM

FAA No DTFA01 88 C 00042 IBM

AAS PROGRAM HIGHLIGHTS (CON'T)

BY YEAR 2000, AAS SEGMENTS WILL BE IN USE THROUGHOUT THE USA
AND FOR FORESEEABLE FUTURE

-- 432 TOWERS

-- 186 TERMINALS (TRACON)

-- 23 ENROUTE CENTERS (ARTCC)

MANY HUNDREDS OF ADA PROGRAMMERS INVOLVED WITH AAS OVER LIFE OF THE PROGRAM

AAS IS BASIS OF WORLDWIDE ATC PROGRAMS/BIDS

-- REPUBLIC OF CHINA (TAIWAN)
-- U.K.'S NEW ENROUTE CENTER (NERC)
-- GERMANY
-- SWEDEN
-- EUROCONTROL (ODS)
-- MEXICO
-- BELGIUM

FOR ADA TO GROW:

ADA 9X MUST BE FULLY DOWNWARD COMPATIBLE WITH ADA 83
(NO CODING CHANGES REQUIRED)

ELSE

- THESE PRODUCTION SYSTEMS WILL NOT TRANSITION TO ADA 9X
- HUNDREDS OF ADA PROGRAMMERS WILL NOT EVOLVE TO USE OF ADA 9X FEATURES

CTA INCORPORATED

AIR FORCE ADA PROJECTION

MAJOR TOM CROAK, USAF

	1991 SURVEY*	1995 PROJECTION
COBOL	40%	20%
ADA	10%	40%
FORTRAN/ JOVIAL	30%	25%
C	3%	10%
OTHER	17% (450 LANG'S.)	5% (250 LANG'S.)

THERE HAVE BEEN NO ADA WAIVERS SINCE JULY 1990

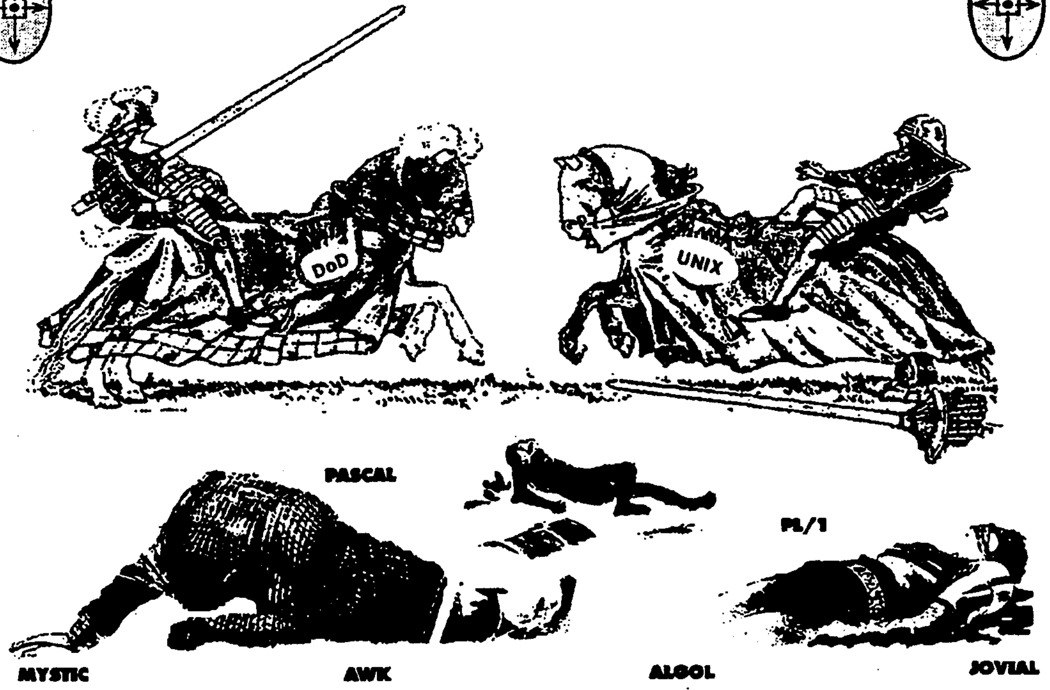
*ALL OPERATIONAL SYSTEMS; ADDITIONAL 32M OF ADA CODE UNDER DEVELOPMENT

CTA INCORPORATED

ADA INFORMATION CLEARINGHOUSE

ADA PROJECTS		EXAMPLES
• ACADEMIA	4	"SUB-SIM" ATTACK SUB SIMULATOR
• ARMY	62	ADVANCE FIELD ARTILLERY TACTICAL DATA SET (AFATDS)
• NAVY	220	ADVANCE SURVEILLANCE WORKSTATION
• MARINE CORPS	41	NAVAL FLIGHT RECORD SUBSYSTEM
• AIR FORCE	151	ADVANCED TACTICAL FIGHTER (F22)
• COMMERCIAL	111	BOEING 777
• GOV'T. (NON-DoD)	58	ADVANCED AUTOMATION SYSTEM (AAS)
• INTERNATIONAL	68	NETHERLANDS TELEPHONE CONTROL & MONITORING SYSTEM
• OTHER DoD	7	SINGLE CHANNEL OBJECTIVE TACTICAL TERMINAL (SCOTT)
TOTAL	722	

Ada vs C++



ADA & C++ - BUSINESS CASE ANALYSIS*

ADA		C++
28 COMPANIES W/VALIDATED PRODUCTS	MARKET AVAILABILITY	18 VENDORS OFFER C++
GOV'T. CONTROLLED/ANSI & ISO STANDARDS	STRONG STADARDIZATION	NO VALIDATION OR STANDARD EXIST
YES	CROSS COMPILATION	NO
22 UNIVERSITIES & 13 DoD INSTALLATIONS	EDUCATION/TRAINING	4 UNIVERSITIES
78.8	FEATURE COMPARISON** (OUT OF 100)	63.9
210 (SLOC/MM) (153 DATA POINTS)	PRODUCTIVITY (NORM: 183 ALL LANG.	187 (SLOC/MM) (38 DATA PTS.)
65 ($/SLOC) (153 DATA POINTS)	COST (NORM: 70 ALL LANGUAGES)	55 (23 DATA PTS.)
	AVG. ERROR RATES (PER KSLOC)	
24 (153 DATA PTS.)	INTEGRATION (33: NORM ALL LANGUAGES)	31 (23 DATA PTS.)
1 (153 DATA PTS)	FORMAL QUAL TEST (3: NORM ALL LANGUAGES,)	3 (23 DATA PTS.)
	ADA COCOMO COST ANALYSIS	
1631 (23% HIGHER)	MIS	1324
1738 (24% HIGHER)	C3 SYSTEMS	1401

* BASED ON U.S. AIR FORCE STUDY
** BY SEI FOR APPLICATIONS INFORMATION/C3 SYSTEMS

CTA INCORPORATED

ADA AN EIGHTEEN YEAR SCOREBOARD

OBJECTIVE	RESULT	SCORE
SINGLE (DoD-1) HOL	WE (CTA) SEE ADA MANDATED IN VIRTUALLY 100% OF DoD RFPs	+
SUPPORT MODERN SOFTWARE ENGINEERING TECHNIQUES	YES: THROUGH STRONG TYPING PACKAGING, AND OTHER FEATURES	+
PROVIDE AN "ADA" ORIENTED PROGRAMMING ENVIRONMENT	NO: CLEARLY, THE PROMISES OF CAIS, APSE, NOT REALIZED	-
INCREASE OF PRODUCTIVITY	NO CLEAR, CONCLUSIVE RESULTS - APPARENT RESULT IS SAME AS OTHER LANGUAGES	NEUTRAL
DECREASE LC SOFTWARE MAINTENACE (EVOLUTION) COST	EVIDENCE IS POSITIVE - LESS ERRORS IN O&M	+
STANDARDIZATION	YES: ANSI & ISO	+
CONTROLLED, STABLE COMPILER IMPLEMENTATION	YES: THROUGH GOV'T. SUPPORT	+
CLEAR "GRASS ROOTS" USAGE (IN COMMERCE, ACADEMIA)	NO: CERTAINLY NOT LIKE "C"	-

<u>OVERALL RESULT: POSITIVE</u>

Appendix A: Attendees

Abd-El-Hafiz, Salwa K., University of Maryland
Addelston, Jonathan D., Planning Research Corp.
Agresti, Bill W., MITRE Corp.
Aikens, Stephen D., DoD
Allen, Julia, Software Engineering Institute
Allen, Russ, IRS
Anderman, Al, Rockwell SSD
Anderson, Barbara, Jet Propulsion Lab
Anderson, Jim, IRS
Angier, Bruce, Institute for Defense Analyses
Arnold, Robert S., Sevtec
Astill, Pat, Centel Federal Services
Austin, James L., IRS
Ayers, Everett, Ayers Associates

Bachman, Scott, DoD
Bacon, Beverly, Computer Sciences Corp.
Bailey, Carmine M., McDonnell Douglas
Bailey, John, SEL
Balick, Glenn, DoD
Barbara, Edward K., U.S. Air Force
Barbour, Ed, U.S. Air Force
Barnes, Bruce H., National Science Foundation
Barnette, Randy, Hughes STX
Barnhart, Don, Boeing Aerospace Co.
Basch, Bill, Boeing Computer Support Services Co.
Basili, Vic, University of Maryland
Bates, Bob G., Lockheed Space Operations
Baumert, John H., Computer Sciences Corp.
Bearchell, Deborah J., Computer Sciences Corp.
Beatty, Kristin, IIT Research Institute
Belle, Jeffery C., Logicon, Inc.
Beswick, Charlie A., Jet Propulsion Lab
Billick, Ron, Bell Atlantic
Binegar, Scott, Computer Sciences Corp.
Biondi, Marisa, IRS
Bishop, Steven, Naval Air Warfare Center
Bisignani, Margaret, MITRE Corp.
Bissonette, Michele, Computer Sciences Corp.
Blackwelder, Jim, Naval Surface Warfare Center
Blagmon, Lowell E., Naval Center for Cost Analysis
Blankenship, Donald D., U.S. Air Force
Blankenship, Gordon, U.S. Air Force
Bloodgood, Pete, IRS
Blough, Lyn, Computer Sciences Corp.
Blum, Bruce I., Applied Physics Lab
Bogdan, Robert J., Computer Sciences Corp.
Boger, Jacqueline, Computer Sciences Corp.
Boland, Dillard, Computer Sciences Corp.
Bond, Jack, DoD
Boon, Dave, Computer Sciences Corp.
Booth, Eric, Computer Sciences Corp.
Borger, Mark W., Software Engineering Institute
Boyce, Glenn W., MITRE Corp.
Bozenski, Richard, DoD
Bozoki, George J., Lockheed
Bradley, Stephen, MMS Systems
Bradshaw, Royce, NATO
Brandt, Thomas C., Software Engineering Institute
Bredeson, Mimi, Space Telescope Science Institute
Briand, Lionel, University of Maryland
Brill, Gary, IRS
Brisco, Phil C., Hughes STX
Brown, Robert E., Hughes Aircraft Co.
Brownsword, Lisa L., Computer Sciences Corp.
Brownsword, Robert J., Rational
Bruhn, Anna, Jet Propulsion Lab
Bullock, Steve, IBM
Bunch, Aleda, Social Security Administration
Burell, Billie, IBM
Burns, Patricia, Computer Sciences Corp.
Butler, Sheldon, Computer Sciences Corp.
Butterworth, Paul, Hughes STX
Button, Janice, DoD
Button, Judee, IRS

Caldiera, Gianluigi, University of Maryland
Calvo, Robert, Paramax Aerospace Systems
Cantalupo, Joy, IIT Research Institute
Capraro, Gerald T., Karman Sciences
Card, Dave, Computer Sciences Corp.
Carlin, Catherine M., Dept. of Veterans Affairs
Carlisle, Candace, NASA/GSFC

PRECEDING PAGE BLANK NOT FILMED

Carlson, J., Computer Sciences Corp.

Carpenter, Maribeth B., Software Engineering Institute

Carruthers, Mary W., IIT Research Institute

Carter, Mike, U.S. Air Force

Cecil, Robert W., Computer Sciences Corp.

Cheramie, Randy, Loral Space Information Systems

Cheung, Helen, Tandem Computers, Inc.

Chiem, I-Ming Annie, Computer Sciences Corp.

Chimiak, Reine A., NASA/GSFC

Chittister, Clyde, Software Engineering Institute

Chiverella, Ron, PA Blue Shield

Cho, Kenneth, U.S. Air Force

Choquette, Carl, IIT Research Institute

Choudhary, Rahim, Hughes STX

Christophe, Debou, Alcatel-Elin Research Centre

Chu, Martha, Computer Sciences Corp.

Chu, Richard, Loral AeroSys

Church, Vic, Computer Sciences Corp.

Clapp, Judith A., MITRE Corp.

Clark, Carole A., Dept. of Veterans Affairs

Clark, Peter G., TASC

Clarke, Margaret J., IBM

Coleman, Carolyn, IIT Research Institute

Condon, Steven E., Computer Sciences Corp.

Connor, David, Computer Sciences Corp.

Cook, John F., NASA/GSFC

Coon, Richard, Computer Sciences Corp.

Cornett, Lisa K., U.S. Air Force

Couchoud, Carlton B., Social Security Administration

Cover, Donna, Computer Sciences Corp.

Crafts, Ralph E., Ada Software Alliance

Creecy, Rodney, Hughes Aircraft Co.

Crehan, Dennis J., Loral AeroSys

Creps, Dick, Paramax Aerospace Systems

Cuesta, Ernesto, Computer Sciences Corp.

D'Agostino, Jeff, The Hammers Co.

Dabrowski, Christopher, NIST

Daku, Walter, Paramax Aerospace Systems

Daney, William E., NASA/GSFC

Dangerfield, Olie B., Computer Sciences Corp.

Daniels, Charles B., Paramax Aerospace Systems

Daniels, Helen, IRS

Davis, Ann, Computer Sciences Corp.

Davis, C., Computer Sciences Corp.

Day, Nancy A., Naval Surface Warfare Center

Day, Orin, Hughes STX

Decker, William, Computer Sciences Corp.

Denney, Valerie P., Martin Marietta

Dhaliwal, Avtar, SEER Systems Corp.

DiNunno, Donn, Computer Sciences Corp.

Dikel, David, Applied Expertise, Inc.

Diskin, Barbara N., Census Bureau

Diskin, David H., Defense Information Systems Agency

Diven, Jeff, IRS

Doland, Jerry T., Computer Sciences Corp.

Dolgaard, Jon, Sunquest Information Systems

Donnelly, Richard E., DoD

Dortenzo, Don, Fairchild Space Co.

Dowen, Andrew, Jet Propulsion Lab

Drake, Anthony M., Computer Sciences Corp.

Driesman, Debbie, Computer Sciences Corp.

Duncan, Scott P., BELLCORE

Duniho, Mickey, DoD

Dunn, Joseph, Computer Sciences Corp.

Durek, Tom, TAD Consulting

Duvall, Lorraine, Syracuse University

Dyer, Michael, IBM

Edelson, Robert, Jet Propulsion Lab

Edlund-O'Mahony, Sheryl J., USA, ISSOCW

Eichmann, David, University of Houston-Clear Lake

Ellis, Walter, IBM

Elovitz, Honey, MITRE Corp.

Elston, Judson R., Boeing Aerospace Co.

Elwood, Todd W., Computer Sciences Corp.

Emerson, Curtis, NASA/GSFC

Emery, Richard D., Vitro Corp.

Engelmeyer, William J., Computer Sciences Corp.

Evanco, William, MITRE Corp.

Evers, J. W., Paramax Aerospace Systems

Fagan, Michael, Michael Fagan Associates

Faller, Ken, HTASC

Farah, Jocelyne, U.S. Air Force

Farrell, Mary Ann, Logicon, Inc.

Farrell, William T., DSD Laboratories, Inc.

Fauerby, John, Computer Sciences Corp.

Feldman, Stuart, BELLCORE

Ferguson, Frances, Stanford Telecommunications, Inc.

Ferrigno, Peter M., RJO-Enterprises, Inc.

Fink, Mary Louise A., Treasury Department

Finley, Doug, Paramax Aerospace Systems

Fleming, Barbara

Fleming, Judy K., IBM

Foreman, John, Software Engineering Institute

Forsythe, Ron, NASA/Wallops Flight Facility

Fouser, Thomas J., Jet Propulsion Lab

Fox, Raymond, DoD

Franklin, Jude E., Planning Research Corp.

Friedman, Seymour R., MITRE Corp.

Fuentes, Wilfredo, Logicon, Inc.

Gallagher, Barbara, DoD

Gaylord, Jerry, IIT Research Institute

Gehrmann, Paul, IBM

Geil, Ester, Westinghouse

Geil, Leana M., Dept. Of Veterans Affairs

Gieser, Jim, Paramax Aerospace Systems

Gillam, Michael, OAO Corp.

Gire, Carey, Loral AeroSys

Giusti, Ronald V., MITRE Corp.

Glascock, Robin, Tandem Computers, Inc.

Glass, Robert L., Computing Trends

Godfrey, Sally, NASA/GSFC

Gogia, B. K., Datamat Systems Research, Inc.

Golden, John R., Rochester Institute of Technology

Golding, Annetta, Census Bureau

Gordon, Del, Paramax Aerospace Systems

Gormally, John M., TRW

Gosnell, Arthur B., U.S. Army Missile Command

Gotterbarn, Donald, East Tennessee State University

Graham, Robert P., U.S. Air Force

Gray, Carmella, CRM

Gray, James H., Computer Sciences Corp.

Green, David, Computer Sciences Corp.

Green, Scott, NASA/GSFC

Greene, Joseph B., Booz, Allen & Hamilton, Inc.

Gregory, John G., Westinghouse

Grondalski, Jean F., Computer Sciences Corp.

Groveman, Brian S., Computer Sciences Corp.

Gu, Dechang, North Carolina A&T State University

Guillebeau, Pat, New Technology, Inc.

Gupta, Lakshmi, Loral AeroSys

Hall, Dana L., SAIC

Hall, John E., DoD

Hall, Ken, Computer Sciences Corp.

Hall, Susan M., SofTech, Inc.

Halpine, Scott, Loral AeroSys

Halterman, Karen, NASA/GSFC

Hankins, Dick, General Dynamics

Hanna, Susan, Beckman Instruments, Inc.

Harrington, Keith, U.S. Air Force

Harris, Barbara, IRS

Harris, Bernard, NASA/GSFC

Harris, Mary, Hughes Aircraft Co.

Hashmi, Awais A., Digital Systems

Hatch, Ada, IRS

Hausler, Philip A., IBM

Hazle, Marlene, MITRE Corp.

Hearn, Rick, Ollila Industries

Heller, Gerard H., Computer Sciences Corp.

Hendrick, Christine, Computer Sciences Corp.

Hendrick, Robert B., Computer Sciences Corp.

Hendrzak, Gary, Booz, Allen & Hamilton, Inc.

Hetmanski, Christopher, University of Maryland

Hill, Ken, Paramax Aerospace Systems

Hilldrup, Kerry C., Hughes STX

Hills, Frederick, Software Productivity Consortium

Hladry, John, Boeing Computer Support Services Co.

Ho, N., Computer Sciences Corp.

Hoffman, Dan, University of Victoria

Hoffman, John C., Sunquest Information Systems

Hoffmann, Kenneth, Ryan Computer Systems

Holmes, Barbara, CRM

Holmes, Joseph A., IRS

Hover, Karen E., Martin Marietta

Hsiah, K., Computer Sciences Corp.

Huang, Bing, FAA

Hull, Larry, NASA/GSFC

Hung, Joshua C., FAA

Huza, Marilyn, IRS

Hynes, Lois, IRS

Ilgenfritz, Charles, IRS

Ippolito, Laura, NIST

Iscoe, Neil, EDS Research, Inc.

Iskow, Larry, U.S. Census Bureau

Jackson, Ann, University of Victoria

Jackson, Lyn, Logicon, Inc.

Jackson, Steve, U.S. Air Force

James, Chris, Computer Sciences Corp.

James, Jason S., DoD

Jay, Elizabeth M., NASA/GSFC

Jeletic, Jim, NASA/GSFC

Jeletic, Kelly A., NASA/GSFC

Jenkins, John O., City University

Jenkins, Jr, Robert, Computer Sciences Corp.

Jepsen, Paul L., Jet Propulsion Lab

Jessen, William J., General Electric

Jilek, Simi S., U.S. Dept, of Energy

Johnson, Kent A., Software Productivity Consortium

Jones, Christopher C., IIT Research Institute

Jones, Deborah M., FAA

Jones, Mel, Applied Expertise, Inc.

Jones, Nancy A., MITRE Corp.

Jordano, Tony J., SAIC

Kavanagh, Dennis M., Computer Sciences Corp.

Kelly, John C., Jet Propulsion Lab

Kelly, Sharon C., Harris Corp.

Kemp, Dennis, Hughes STX

Kemp, Kathryn M., NASA/HQ

Kester, Rush, Computer Sciences Corp.

Kieckhefer, Ron, Computer Sciences Corp.

Kim, Seung, Computer Sciences Corp.

Kirkendall, Thomasin, NIST

Kirkpatrick, Diane, Ball Aerospace

Kistler, David M., Computer Sciences Corp.

Klitsch, Gerald, Computer Sciences Corp.

Knapp, Andy, Bell Atlantic

Knoell, Roger, U.S. Air Force

Koeser, Ken, Vitro Corp.

Konopka, Joseph J., Computer Sciences Corp.

Kontson, Kalle R., IIT Research Institute

Kopperman, Stuart, Systems Research & Applications Corp.

Kosloski, Joe, IRS

Kovin, Steven E., Computer Sciences Corp.

Kramer, Nancy, Viar & Co. /Dyncorp

Kramer, Teresa L., DoD

Kristof, Dave, U.S. Air Force

Kudlinski, Robert A., NASA/LaRC

Kurihara, Tom, Logicon, Inc.

Lai, R., Chi Tau, International S/W Process Constellation

Lal, Nand, NASA/GSFC

Lam, Vincent, IMS

Lane, Sherry, CRM

Lang, John, Computer Sciences Corp.

Langston, James H., Computer Sciences Corp.

Laubenthal, Nancy, NASA/GSFC

Lawlor, Tom, Bell Atlantic

Lawrence, Raymond, LMcDonnell Douglas

Lawrence-Pfleeger, Shari, MITRE Corp.

Ledford, Rick, McDonnell Douglas

Lee, Raymond H., Computer Sciences Corp.

Lee, Thomas S., Paramax Aerospace Systems

Lehman, Meir, Imperial College of Science

Lemmon, Doug, University of Maryland

Leone, Rick, Hughes STX

Levitt, David S., Computer Sciences Corp.

Lewicki, Scott A., Jet Propulsion Lab

Li, Ningda Rorry, University of Maryland

Liebrecht, Paula, Computer Sciences Corp.

Lijewski, Mike, Hughes STX

Likness, Mark, Martin Marietta

Lindsay, Orlando, Computer Sciences Corp.

Lindvall, Mikael, Linkoping University

Lippens, Gary A., U.S. Air Force

Litz, Deborah, DoD

Liu, Jean C., Computer Sciences Corp.

Liu, Kuen-san, Computer Sciences Corp.

Loesh, Bob E., Software Engineering Sciences, Inc.

Loftin, Donald R., GE Aerospace

Long, Roger, LTASC

Loomis, Todd, Booz, Allen & Hamilton, Inc.

Loy, Patrick H., Loy Consulting, Inc.

Lucas, Janice P., Dept. of Treasury

Luczak, Ray, Computer Sciences Corp.

Lupinetti, Martin, Computer Sciences Corp.

Luppino, Fred, IBM

Lyle, William, TASC

Maccannon, Cecil, FAA

Madden, Joseph A., U.S. Air Force

Majane, John A., EG & G WASC, Inc.

Manicka, Gary, Hughes STX

Manter, Keith, Computer Sciences Corp.

Marciniak, John, Computer Technology Associates, Inc.

Marcoux, Darwin, DoD

Marquiss, Terri, Computer Sciences Corp.

Martin, Carol, TRW

Mashiko, Yasuhiro, University of Maryland

Mauney, Mike, Census Bureau

Maury, Jesse, Omitron, Inc.

McConnel, Pat, IRS

McCreary, Julia M., IRS

McGarry, Frank E., NASA/GSFC

McGarry, Mary Ann, IIT Research Institute

McGarry, Peter, General Electric

McGovern, Dan, FAA

McKay, Judith A., Census Bureau

McKinney, Cathy, IRS

McKinney, Jimmie, USAFISA

McNeill, Justin F., Jet Propulsion Lab

McSharry, Maureen, Computer Sciences Corp.

Mehta, Shilpa, American Systems Corp.

Meick, Douglas, Library of Congress

Mendonca, Manoel G., University of Maryland

Merry, Paul, Harris Space Systems Corp.

Miller, Ronald W., NASA/GSFC

Miller, Sharon E., AT&T Bell Lab

Miller, Terrence, Project Engineering, Inc.

Mills, John P., Booz, Allen & Hamilton, Inc.

Mohallatee, Michael, Computer Sciences Corp.

Moleski, Laura, CRM

Moleski, Walt, NASA/GSFC

Moniuszko, Charles, DoD

Moore, Betty, IRS

Moore, Kathryn J., USAISSDC-A

Moore, Paula, NOAA/SPOx3

Moortgat, Jean-Jacques, Booz, Allen & Hamilton, Inc.

Morasca, Sandro, University of Maryland

Morgan, Elizabeth, Bendix Field Engineering Corp.

Morusiewicz, Linda M., Computer Sciences Corp.

Mostoller, Brad, Sunquest Information Systems

Moxley, Fred I., DISA/CFS

Mucha, John F., IRS

Muckel, Jerry, Computer Sciences Corp.

Mulville, Daniel, NASA/HQ

Munkeby, Steve, Martin Marietta

Murphy, Susan, IBM

Murtha, Kimberly N., Sunquest Information Systems

Myers, Philip I., Computer Sciences Corp.

Myers, Robert M., MITRE Corp.

Narrow, Bernie, Bendix Field Engineering Corp.

Nassau, Dave, Applied Expertise, Inc.

Newman, Phillip A., NASA/GSFC

Nishimoto, Theresa, Coopers & Lybrand

Nola, Charles L., NASA/MSFC

Noonan, Carolina, Computer Sciences Corp.

Noone, Estelle, Computer Sciences Corp.

Norcio, Tony F., University of Maryland Baltimore County

O'Brien, Robert L., Paramax Aerospace Systems

O'Connor, Sean, Martin Marietta

O'Neill, Don

O'Neill, Patrick, U.S. Army AMSAA

O'Neill, Peter, PA Blue Shield

Ohlmacher, Jane A., Social Security Administration

Okupski, Scott, U.S. Air Force

Olson, Lenoard, Hughes STX

Osifchin, Tammy, Hughes Aircraft Co.

Padgett, Kathy, Census Bureau

Page, Gerald, Computer Sciences Corp.

Pajerski, Rose, NASA/GSFC

Palmer, Regina, Martin Marietta

Paluzzi, Paul, Computer Sciences Corp.

Pang, Les, FAA

Panlilio-Yap, Nikki M., IBM Canada Ltd.

Park, Robert, Computer Sciences Corp.

Patrick, Debora, IIT Research Institute

Patterson, F., G., NASA SSFPO

Patton, Kay, Computer Sciences Corp.

Pavnica, Paul, Treasury/Fincen

Pecore, Joseph N., Vitro Corp.

Peeples, Ron L., IBM

Pendergrass, Vicki, NASA/GSFC

Pendley, Rex, Computer Sciences Corp.

Peng, Wendy, NIST

Peng, Yuh-Fen, Computer Sciences Corp.

Perry, Brendan, Hughes STX

Peters, Jeffrey, U.S. Air Force

Pettijohn, Margot, IRS

Philpot, Donn E., Technology Applications, Inc.

Philpot, Fred, Dept. of the Air Force

Plett, Michael E., Computer Sciences Corp.

Plonk, Glenn, DoD

Polly, Mike, Raytheon

Poms, B., Computer Sciences Corp.

Porter, Adam A., University of Maryland

Potter, Marshall R., Dept. of the Navy

Pottinger, David L., SAIC

Powers, Larry, Unisys Corp.

Presbury-Bush, Anna, DoD

Preston, Dick

Provenza, Clint, Booz, Allen & Hamilton, Inc.

Quann, Eileen S., Fastrak Training, Inc.

Quindlen, Brian, Computer Sciences Corp.

Quinn, Harold, Computer Sciences Corp.

Rager, William J., Computer Sciences Corp.

Rahmani, Donna, Computer Sciences Corp.

Rajlich, Vaclav, Wayne State University

Raney, Dale, LTRW

Ransdell, William G., Research Triangle Institute

Ray, Julie, New Technology, Inc.

Raymond, Jack, Computer Sciences Corp.

Reddy, Swami, Hughes STX

Reed, Lee Scott, Software Engineering Institute

Regardie, Myrna L., Computer Sciences Corp.

Reifer, Don J., Reifer Consultants, Inc.

Repsher, Marie, IRS

Rhoads, Thomas E., Computer Sciences Corp.

Rhodes, Tom, NIST

Rice, Mary K., USAF

Ridgeway, Roland M., NASA/HQ

Risser, Gary E., Dept. of Veterans Affairs

Rizer, Stephani, NAWC-AD

Rizzello, John, Loral AeroSys

Roberts, Becky L., CBIS Federal Inc.

Roberts, Geraldine, Computer Sciences Corp.

Robertson, Laurie, Computer Sciences Corp.

Robinett, Susan, Systems Research & Applications Corp.

Robinson, Alice B., NASA/HQ

Rohr, John A., Jet Propulsion Lab

Rombach, H. Dieter, University of Kaiserslautern

Rose, Lois A., Bell Atlantic

Rosenberg, Linda H., NASA/GSFC

Roth, Karen, Paramax Aerospace Systems

Rouff, Chris, NASA/GSFC

Roy, Dan M., Software Engineering Institute

Rudiger, Karen S., Boeing Computer Support Services Co.

Russell, Wayne M., GTE

Russo Waters, Olga, Logicon, Inc.

Rymer, John, IBM

Sabarre, Nick, IRS

Sahady, Phil, Booz, Allen & Hamilton, Inc.

Saisi, Robert, ODSD Laboratories, Inc.

Salwin, Arthur, MITRE Corp.

Sanden, Bo I., George Mason University

Santiago, Richard, Jet Propulsion Lab

Schappelle, Sam, IBM

Schilling, Mark, Project Engineering, Inc.

Schneidewind, Norman F., Naval Postgraduate School

Schoen, Bill, IIT Research Institute

Schuler, Pat M., NASA/LaRC

Schwartz, Karen D., Government Computer News

Schwarz, Henry, NASA/KSC

Scott, Rhonda M., IBM

Seaman, Carolyn B., University of Maryland

Seaver, David P., Project Engineering, Inc.

Seidewitz, Ed, NASA/GSFC

Shammas, Barbara A., IRS

Sheets, Teresa B., NASA/GSFC

Sheppard, Sylvia B., NASA/GSFC

Shirey, Carl L., ITT

Shockey, Donna, IRS

Short, Cathy, IRS

Siddalingaiah, Vimala, Computer Sciences Corp.

Siegel, Karla, MITRE Corp.

Simenson, Norman, FAA

Singer, Carl A., BELLCORE

Singh, Prakash, EER Systems Corp.

Singleton, Frank L., Jet Propulsion Lab

Skrivan, James A., Boeing Computer Support Services Co.

Sledge, Carol, Software Engineering Institute

Smith, David, Computer Sciences Corp.

Smith, Diana, IIT Research Institute

Smith, Donald, NASA/GSFC

Smith, Shawn D., American Systems Corp.

Smith, Vivian, AFAA

Snook, Judy, Computer Sciences Corp.

Song, Fu-Fu, Computer Sciences Corp.

Sorensen, Steven, Martin Marietta

Sova, Don, NASA/HQ

Spangler, Alan R., IBM

Spence, Bailey, Computer Sciences Corp.

Spencer, Mike, Naval Air Warfare Center

Spool, Peter R., Siemens Corporate Research, Inc.

Sporn, Patricia A., NASA/HQ

Squires, Burton E., Mnemonic Systems Inc.

Srivastava, Alok, Computer Sciences Corp.

Stanton, Faye, IRS

Stark, Michael, NASA/GSFC

Stauffer, Mike P., General Electric

Stevens, Jan, Systems Research & Applications Corp.

Stewart, Barbara C., U.S. Air Force

Strano, Caroline, FAA

Sugumaran, Vijayan, George Mason University

Svara, Allan C., USAF/7th, Comm, Group

Swain, Barbara, University of Maryland

Szulewski, Paul S., Draper Labs, Inc.

Tasaki, Keiji, NASA/GSFC

Tausworthe, Robert C., NASA/JPL

Tavakoli-Shiraji, Iraj, George Mason University

Tervo, Betsy, Computer Sciences Corp.

Thackrey, Kent, Planning Analysis Corp.

Theeke, Patrick, Electronic Warfare Associates, Inc.

Theofanos, Mary, Martin Marietta

Thomas, Donna C., Computer Sciences Corp.

Thomas, Isac, Computer Sciences Corp.

Thomas, William, MITRE Corp.

Thomen, Mark, IBM

Thrasybule, Wesner, Computer Sciences Corp.

Tipparaju, Suri, Hughes STX

Tisnado, Gilberto M.

Tran, Dennis A., MITRE Corp.

Tran, Tuyet-Lan, Jet Propulsion Lab

Trujillo, Nelson W., NDU/IRMC

Truong, Son, NASA/GSFC

Tsagos, Dinos

Tupman, Jack R., Jet Propulsion Lab

Ullman, Richard, Hughes ST Systems Corp.

Usavage, Paul, General Electric

Valett, Jon, NASA/GSFC

Valleni, Bob R., TRW

Van Meter, David, Logicon, Inc.

Van Verth, Patricia B., Canisius College

VanHorn, Wendy J., IRS

Varklett, Vanessa, IIT Research Institute

Vaughan, Joe, Social Security Administration

Vause, David G., IBM

Vazquez, Federico, Computer Sciences Corp.

Velez, Tom E., Computer Technology Associates, Inc.

Verducci, Anthony J., AT&T Bell Lab

Viola, Ken W., IRS

Voit, Eric, Bell Atlantic

Votta, Lawrence G., AT&T Bell Lab

Wagoner, Raelene, Systems Research & Applications Corp.

Waligora, Sharon R., Computer Sciences Corp.

Wallace, Charles J., Integrated Systems Analysts, Inc.

Wallace, Dolores, NIST

Walsh, Bob, IRS

Walsh, Chuck, NASA Center for Aerospace Information

Waszkiewicz, Mary Lily, Computer Sciences Corp.

Weber, Paul A., Technology Planning, Inc.

Weiss, Peter, Arthur D., Little, Inc.

Weiss, Sandy L., GTE

Wells, Robert, Computer Sciences Corp.

Werling, Richard, Software Productivity Consortium

Wessale, William, CAE-Link Corp.

Weston, William, NASA/GSFC

Weszka, Joan, IBM

Wheeler, J. L., Computer Sciences Corp.

White, Cora P., New Technology, Inc.

Whitehead, John W., NAVSEA 06D3

Whitfield, Josette, IIT Research Institute

Whitman, Cynthia B., USAISSDC-A

Wilkins, Elsie C., USAFISA

Williamson, Jim, Sunquest Information Systems

Wilson, Jim, Applied Expertise, Inc.

Wilson, Randy D., Naval Center For Cost Analysis

Wingfield, Lawrence D., Computer Sciences Corp.

Wisdom, Rex, U.S. Air Force

Wise, Charles F., Technology Applications, Inc.

Wong, Sha, IMS

Wong, Yee, Computer Sciences Corp.

Wood, Richard, Computer
Sciences Corp.

Wood, Terri, NASA/GSFC

Woodward, Herbert P., TRW

Worley, Patricia W., Boeing
Computer Support
Services Co.

Wortman, Kristin, Hughes
STX

Yin, Sandra, IRS

Youman, Charles, SETA
Corp.

Young, Andy, Bendix Field
Engineering Corp.

Yu, Anna, NC A&T State
University

Zavaleta, Henry M.,
Computer Sciences
Corp.

Zaveler, Saul, U.S. Air Force

Zelkowitz, Marv, University
of Maryland

Zimet, Beth, Computer
Sciences Corp.

Zucconi, Lin, Lawrence
Livermore National
Laboratory

Zvegintzov, Nicholas,
Software Maintenance
News Inc.

Appendix B: Standard Bibliography of SEL Literature

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

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-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

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

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

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-89-201, *Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 2)*, L. Morusiewicz, J. Bristow, et al., October 1992

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

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-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

[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

[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

[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

[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

[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

[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

[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

[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

[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

NOTES:

[0]This document superseded by revised document.

[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.