N86 - 30367

PANEL #4

EXPERIMENTS WITH ADA

D. Roy, Century Computing Inc.
M. McClimens, Mitre Corporation
W. Agresti, Computer Sciences Corporation

Daniel M Roy
Century Computing, Inc.

Abstract

A 1200 line Ada source code project simulating the most basic functions of an operations control center was developed for code 511. We selected George Cherry's Process Abstraction Methodology for Embedded Large Applications (PAMELA) and DEC's Ada Compilation System (ACS) under VAX/VMS to build the software from requirements to acceptance test. The system runs faster than its FORTRAN implementation and was produced on schedule and under budget with an overall productivity in excess of 30 lines of Ada source code per day.

Author current address:
Century Computing Incorporated,
8101 Sandy Spring Rd.
Laurel, Md. 20707
(301) 953 3330

1 BACKGROUND

The Multi-satellite Operations Control Center branch (MSOCC), code 511, has embarked on an effort to improve productivity in the development and maintenance of Operations Control Center (OCC) systems. This productivity effort is addressing a range of issues from equipment and facilities improvements to the development and acquisition of tools and the training of personnel.

Century Computing's previous work on MSOCC's productivity improvement program, identified the Ada language as a promising technology, and recommended evaluating Ada on a small "pilot project" related to MSOCC applications [Century-84].

2 PURPOSE OF THE STUDY

The objective of the study was to evaluate the applicability of Ada and its development environment for MSOCC. Metrics were identified for this evaluation, along with an approach to collecting the data required for these metrics. The evaluation was based on using Ada to re-develop from scratch a small scale, real-time project related to MSOCC applications: an Application Processor (AP) benchmark system.

3 DESCRIPTION OF THE AP BENCHMARK SYSTEM

An AP is a computer that performs the functions required by a satellite operations control center. The AP Benchmark system was previously developed to simulate the characteristics of a typical MSOCC's AP software system [CSC/SD-83]. Like most AP software, the Benchmark was developed in FORTRAN with some supporting assembly language.

The AP Benchmark software simulates the following AP functions:

- Reads a telemetry data stream from tape - meters the frequency of tape reads to simulate various data rates.
- Decommutates the telemetry data.
- Performs some limit checking on the data.
- Displays some of the telemetry data on CRT screens.
- Simulates the history and attitude data recording processes.
- Simulates strip chart recorders and associated functions.
- Gathers statistics on the above process and generates reports.

4 DESCRIPTION OF THE ADA PILOT PROJECT

The pilot project began with a reverse engineering phase to construct requirements from the existing FORTRAN code. Then, a staged approach was used to develop the software, using Ada for all project phases:

- We used Ada as a Data Definition Language to produce a data dictionary during the requirements analysis phase. A special package, the "TBD" package (fig. 1) helped in the top down design of the data structure.

- We used Ada as a Program Specification Language very early in the project and easily prototyped the data flow. The Process Abstraction Methodology tools [Cherry-84] (see appendix B) produced a tasking model that worked at first try (fig. 2a and b). The preliminary and detailed design templates we created (fig. 3a and b) proved to be very useful for enforcing good practices.

- We used Ada as a Program Design Language [IEEE-990] (fig. 4) and refined the PDL into detailed Ada code in the usual staged manner. The DCL tools and templates for Ada construct, developed at the onset of the project, had a dramatic impact on productivity and code consistency.

- We enjoyed the elegance of Ada as an implementation language and used most of its features (attributes, generics, exception handlers, etc.)

- Full assessment of the DEC ACS tools was beyond the scope of this study, but we appreciated the built-in configuration control tool, the automatic recompilation system and the symbolic debugger [DEC-85].

The total re-development approach we followed (from requirements to final tests) led us to believe that we could produce a still more efficient design. Actually, the PAMELA methodology design rules detected several extraneous tasks in the current AP benchmark model, but we decided to respect the existing global structure as the model was built to represent the typical CPU load of an actual OCC.

```
--------------------------------------------------------------------
Package TBD is          --| Decision deferral package --*
--| Raises:
--|     None
--| Overview:   --| Purpose:
--|     This is an improvement over Intermetrics' TBD package and IEEE 990
--|     recommendations about decision deferral techniques.
--| Effects:    --| Description:
--|     The distinction is clarified between types, variables and values.
--|     The naming is more consistent (enum_i, component_i ...) and more
--|             readable (scalar_variable intead of scalarValue)
--|     There are more definitions (enum_type, record_type)
--|     Better compatibility with BYRON (or search utility processing)
--| Requires:   --| Assumptions:
--|     Please only "WITH" this package.  By systematically specifying
--|     "TBD.x" items, it is easier to assess the stage of development of
--|     a compilation unit.
--| Notes:
--|    Change log:
--| Daniel Roy   9-AUG-1985     Baseline
--                                                                   --|
    subtype scalar_type is integer range integer'first .. integer'last; --|
    scalar_variable : scalar_type;                                   --|

    type access_type is access integer;                              --|
    access_variable : access_type;                                   --|
                                                                     --|
    type record_type is record                                       --|
        component_1 : integer := 0;                                  --|
        component_2 : integer := 0;                                  --|
        component_i : integer := 0;                                  --|
        component_p : integer := 0;                                  --|
        component_n : integer := 0;                                  --|
    end record;                                                      --|
    record_variable : record_type;                                   --|
                                                                     --|
--      Inspired by IBM PDL stuff
    Condition,CD : Boolean := true;                                  --|
                                                                     --|
--      Queues services
    type queue_type is array (array_index_type) of integer;          --|
    type queue_ptr_type is access queue_type;                        --|
--
end TBD;        --| --*
```

Fig. 1: Excerpt from the TBD package

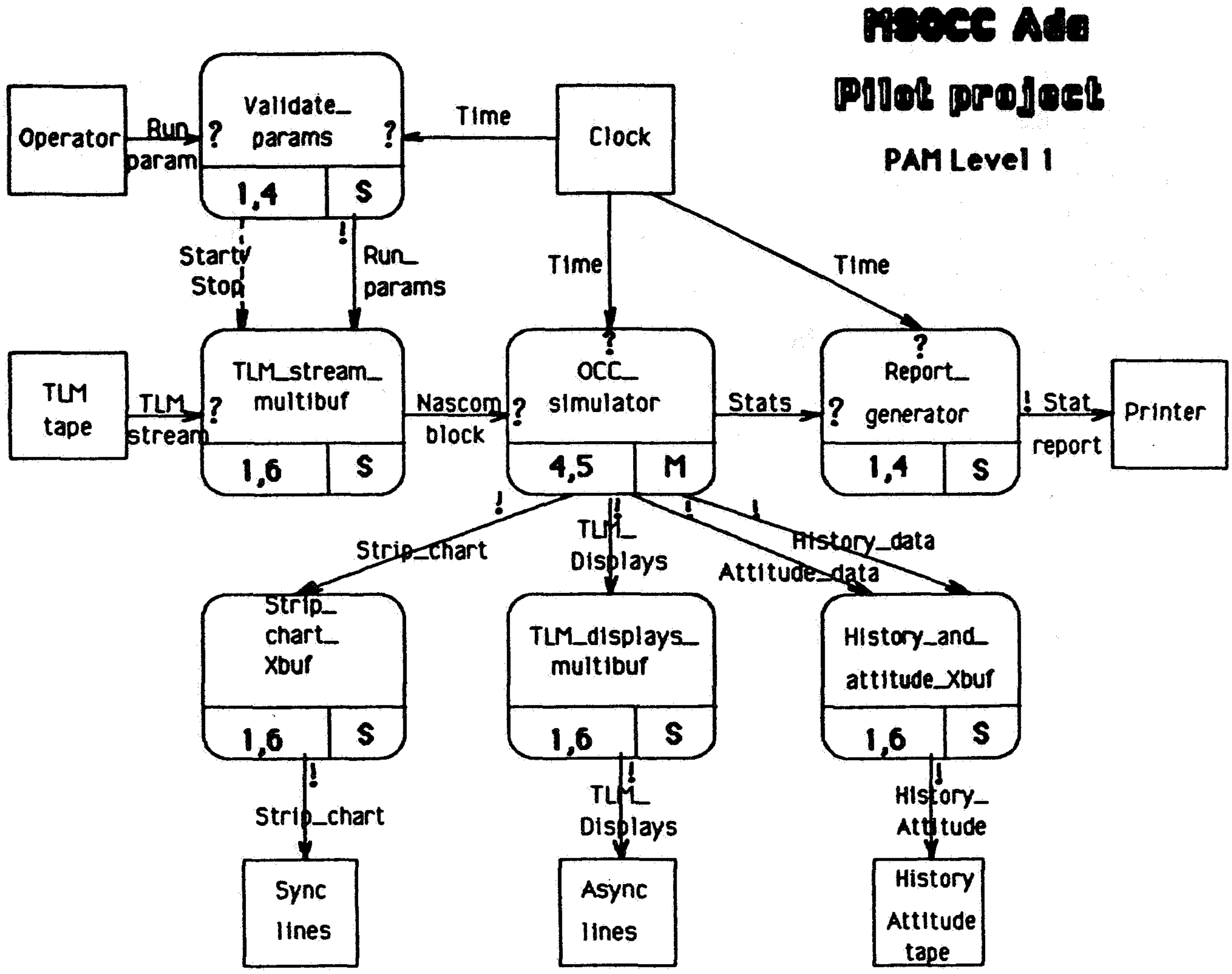


Figure 2a: PAM decomposition level 1

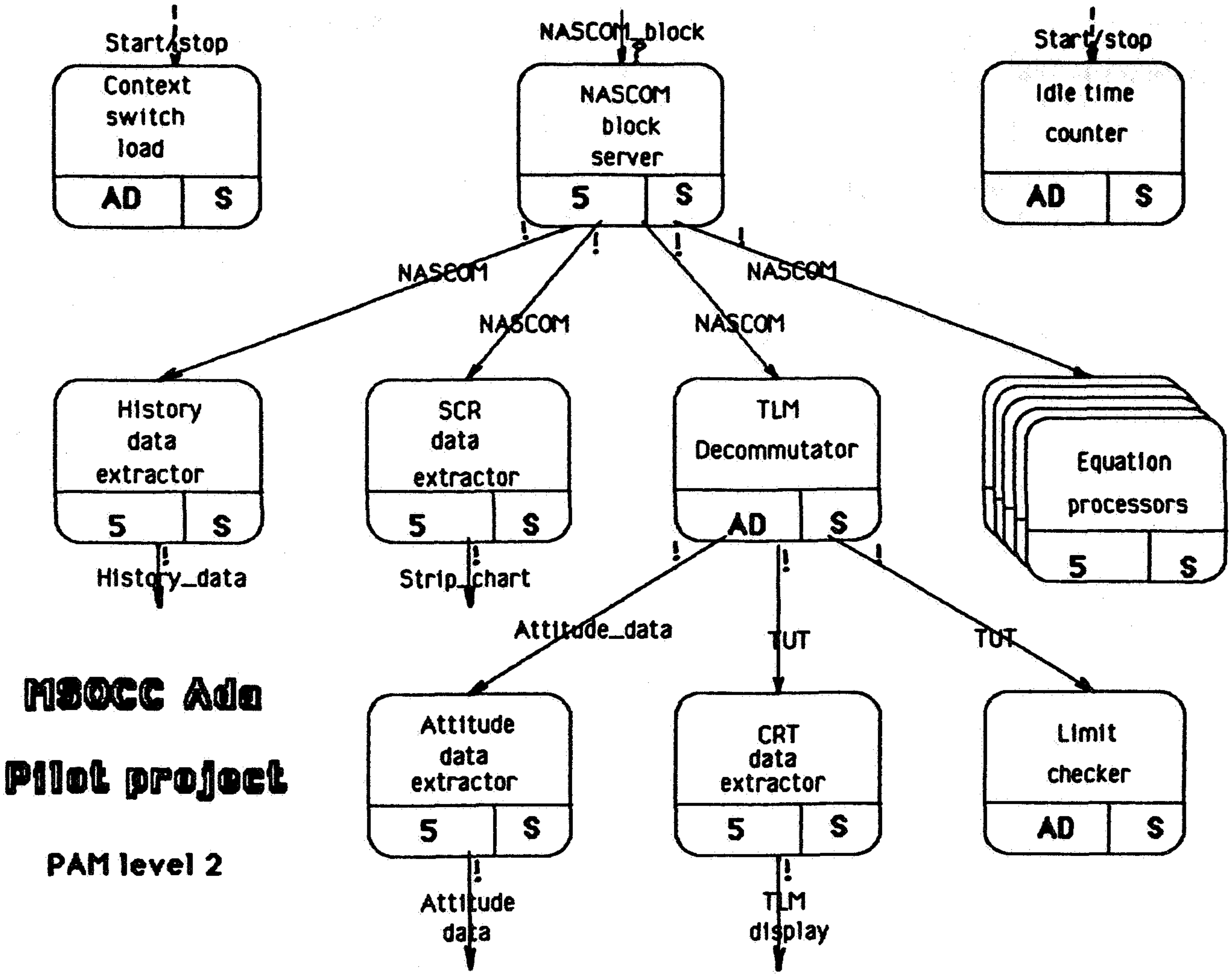


Figure 2b: PAM decomposition level 2

```
-----------------------------------------------------------------
--
   procedure P (        --| synopsis --*
       param_1 : IN some_type := some_constant ; --| description --*
       param_n : OUT some_type                  --| description --*
       ) ;                              --| --*
```

Fig. 3a: Preliminary design template for procedure (proc spec)

```
-----------------------------------------------------------------
```

```
-----------------------------------------------------------------
   separate ( )         --| --*
   procedure body P (   --| Short synopsis.  Must be the same as in body. --*
       param_1 : IN some_type := some_constant ; --| description --*
       param_n : OUT some_type                  --| description --*
       ) is                             --| --*
--|
-- ****** Cut and paste from specification.  Use Gold D for rest of DOC. ******
--
-- Packages

-- types

-- subtypes

-- records

-- variables

-- functions

-- procedures

-- separate clauses

   begin        --| --*
       null;
   end P ;      --| --*
```

Fig. 3b: Detailed design template for a procedure (proc body)

```
-----------------------------------------------------------------
```

```
----------------------------------------------------------------------
package body user_interface is          --| Isolate user interface --*
--
    function inquire_int (               --| Emulate DCL verb for integers --*
        prompt : string         --| --*
        ) return inquired_var_type is    --| --*
        inquired_var : inquired_var_type ;      --* The variable we'll return
    --
    begin       --| inquire_int --*
        --* Displays "prompt (min..max): "
        for try in 1.. max_nr_errors loop       --* until good value or else
    --
            begin       --* <<exception_block>>
                --* Get unconstrained value
                --* Validate and translate unconstrained value
                return inquired_var ;    --| --*
        --
            exception   --* recoverable exception when invalid input
                when data_error | constraint_error =>    --*
                    --* display "try again" message
            --| end exception    --*
            --
            end ;       --* <<exception_block>>
        end loop;       --* until good value or else
    --
        exception       --* catch all handler
        --
            when others =>      --*
                raise;  --*
    end inquire_int ;   --| --*
```

Fig. 4: PDL extracted from code by PDL tool

----

5 RESULTS SUMMARY

Some of the objectives of the evaluation were to determine what is required to train software engineers to use Ada, to define adequate metrics to measure productivity and quality gains and to assess the current Ada development environment.

5.1 Training

We found that Ada is sufficiently complex that we kept learning throughout the pilot project, and even beyond. We also found that none of the standard training devices (seminars, books, computer aided instruction) could alone address the broad range of issues that really are at the heart of the problem:

In the Ada era, a comprehensive education in the software engineering principles that form the basis of the Ada culture must replace ad-hoc training in the syntactic recipes of a language.

That is why we recommend a variety of continuous education measures in our report: Assuming adequate familiarization with modern software engineering practices, at least 4 person-week is the minimum minimorum training time. This time includes teaching a methodology adapted to Ada and 50% hands on experiments under the supervision of an expert.

5.2 Metrics And Data Collection Approach

After a review of established research in the areas of metrics and data collection, a brief paper outlining the metrics approach was issued. The metrics work of the NASA Software Engineering Laboratory was the key input [McGarry-82].

Simple DCL tools were built to gather the metrics data and comprehensive logs of errors, problems and interesting solutions were maintained on-line and are part of the deliverables.

5.3 Productivity

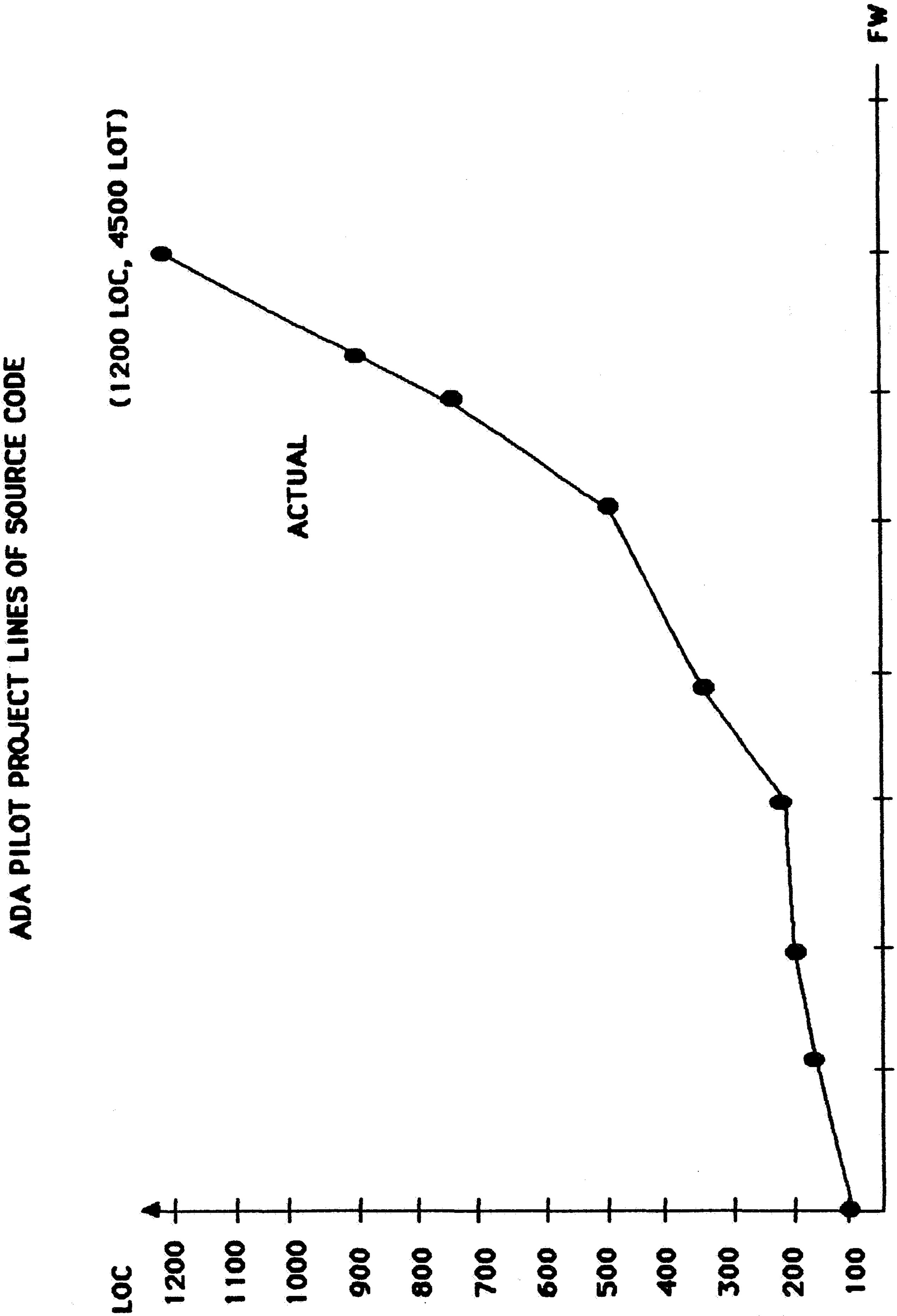
Our productivity during the seven weeks coding period averaged 32 lines of Ada source code (LOC) per day and nearly 130 lines of text (LOT) per day (includes embedded documentation, comments and blank lines). We experienced a low point of 10 LOC per day at the beginning of the coding phase, and reached a peak of 90 LOC and 370 LOT per day during the final week (fig. 5). Averaged over the whole 18 weeks of development (including reverse engineering with DeMarco before PAM, tools development, two seminars, compilers installation, etc.) productivity still remains above 13 LOC and 50 LOT per day.

Although formal verification techniques were not employed, intense validation testing discovered two errors, both due to subtle differences between our implementation and its FORTRAN precursor. A detailed log of all the problems we had at various phases of the implementation was kept on-line.

Those productivity and quality results are interesting data points, but they must be taken with the following caveat:

- We were re-implementing a working system.
- Our deliverables did not include all standard documentation.
- We did not produce a performance prediction study.
- We did not perform a deadlock avoidance study.
- Unit testing was not up to the standards we would have applied to an operational system.
- We sometimes abandoned early our search for better solutions.
- When a problem arose we did not always research why.
- More than 90% of the code was written by a single individual.

On the other hand, we wrote much more scaffolding and experimental ("throw away") software than a normal project would require.



ADA PILOT PROJECT LINES OF SOURCE CODE
(1200 LOC, 4500 LOT)
ACTUAL
LOC
1200
1100
1000
900
800
700
600
500
400
300
200
100
FW
16
17
18
19
20
21
22
23
24

Figure 5

5.4 Compilers Experience

We first used Century's NYU Courant Institute Ada interpreter on our VAX 11/750 for training and tools development. We quickly became frustrated with this system.

Thanks to NASA's cooperation, we got some exposure to the Telesoft compilers and the DEC Ada Compilation System (ACS).

We then installed Softech's Ada Language System (ALS) on another NASA VAX. Our conclusion was that the current performance problems of the ALS made it unsuitable in light of our schedule constraints.

In the end we were granted access to code 520's test version of DEC's Ada Compilation System (ACS) under VMS 4.1 which we used to develop most of the pilot project. It is clear to us that the ACS made the timely completion of our project possible and that, in general, the quality of the development environment significantly impacts software development productivity.

As delivered, the Ada pilot project features about the same number of statements as its FORTRAN precursor (about 1200) but is larger in the number of lines of text (4,500 vs 2,000). Image sizes are comparable (about 170 kbytes for Ada vs about 200 kbytes for FORTRAN).

Even though it is difficult to compare run time performance on the very different computer environments we used, **our preliminary results seem to indicate that the Ada code runs faster than its FORTRAN counterpart.** We suspect that our good results may be due to the fact that some data elements could be directly addressed in Ada and not in FORTRAN. Nevertheless, this is a completely unexpected result that is even contrary to popular belief. We think it speaks for the high quality of DEC's ACS and the adequacy of the chosen methodology (the Process Abstraction Methodology for Embedded Large Applications).

6 CONCLUSIONS

Ada is clearly a step forward in the software industry's search for a better programming language for real-time and embedded systems. Ada also represents significant advancements in the field of practical programming language development.

Furthermore, the Ada Programming Support Environment (APSE) and the Software Technology for Adaptable Reliable Systems (STARS) initiative will support the language with an impressive set of evolving tools.

But even with these features, it is possible to develop poor software in Ada. In fact, packaging, generics, multitasking and, above all, representation clauses (that allow direct access to the hardware!) will have to be closely controlled by competent project managers because these features are powerful, hence dangerous. Moreover, those powerful features provide another dimension of design decision. We feel that a methodology that helps the software engineer allocate function and data structures to packages and tasks is necessary.

Ada should prove to be an excellent tool in the hands of competent and properly trained software developers. It will not be a panacea, compensating for inadequate methods or training, but it will be beneficial if properly applied.

In that context, we make the following predictions relative to the future of Ada:

1. The momentum of the Department of Defense will make Ada a reality. The last time that DoD backed a language (COBOL), the language became, and still is, the most popular in the world.

2. **There will be major false starts in the use of Ada, especially when the aerospace contractors tackle large projects with newly trained programmers.** Ada itself will become the focus of these projects, leaving the target application in second place.

3. The "reality" of Ada will be delayed due to the immaturity of the compiler technology, expense of computer resources, and the training problem.

4. There will be major difficulties at both ends of the programmer competency scale. Many of the brightest programmers will tend to produce overly complex designs, using every possible feature of the language; the application itself becoming a side issue. Many of the less competent programmers will never really understand the Ada technology.

5. Programmer productivity will decrease (relative to conventional languages) before it eventually increases.

6. Universities will eventually produce proficient Ada software engineers, using the language as a basis for teaching all the traditional computer science courses. (This day is getting near. We recently polled area universities and found Ada present in every computer science curriculum.)

7 A FINAL NOTE

In July 1985, following the recommendation of the APSE Beta Test Site Team headed by Dr. McKay (University of Houston at Clear Lake), NASA officially adopted Ada as the language of choice for all flight software of the space station program.

APPENDIX A

BIBLIOGRAPHY

[Century-84] Century Computing Inc., "Software Tools and Methodology Study for NASA MSOCC", Laurel, Md., June 1984.

[Cherry-84] George W. Cherry, "Advanced Software Engineering with Ada", Seminar notes , 1984.

[Cherry-85] George W. Cherry, "The PAMELA (TM) Methodology, A Process-oriented Software Development Method for Ada.", To be published.

[CSC/SD-83] Computer Science Corporation, "Gamma Ray Observatory Era Application Processor Benchmark User's Guide", Update 1, Doc. No. CSC/SD-83/6101UDI, January 1984.

[DEC-85] Digital Equipment Corporation, "Developing Ada Programs On VAX VMS", February 1985.

[IEEE-990] IEEE working group on Ada PDL (990), "Ada PDL draft recommended practice", 5 March 1985.

[McGarry-82] Frank McGarry et al., "Guide to Data Collection", SEL-81-101, NASA GSFC, August 1982.

[Methodman-82] Peter Freeman, Anthony Wasserman, "Software Development Methodologies and Ada", National Technical Information Service, ADA 123-710, November 1982.

APPENDIX B

THE PROCESS ABSTRACTION METHODOLOGY

"The Process Abstraction Methodology for Embedded Large Applications (PAMELA or PAM for short) is a real-time software development method which takes full advantage of Ada's features of type abstraction, process abstraction, exception handling, top-down separate compilation, and bottom-up separate compilation.

Because the PAMELA method recognizes that abstract processes as well as abstract data types are ideal modules for programming in the large, the method is process-oriented as well as object-oriented.

The method is primarily a top-down, outside-in method; but it allows and encourages the bottom-up generation or incorporation of software components (library units).

The PAMELA method contains guidelines to ensure that program units are reusable or portable or both reusable and portable. It also contains guidelines to ensure superior real-time performance (for example, guidelines to ensure that the minimum number of necessary tasks are defined)." [Cherry-85]

"The process abstraction methodology (PAM) is based on the concept of a hierarchical structure of processes. The process as a data transforming element and data flow as a connection link between processes are central concepts in this method." [Cherry-84]

At first glance, the PAMELA methodology "process graphs" (fig. 2a and 2b) look very much like DeMarco's Data Flow Diagrams. The major difference however, is that in any data driven methodology, there is no apparent synchronization between the processes nor any explicit representation of the synchronization between the flow of data and the processes. In a process graph, the processes communicate by the Ada rendez-vous mechanism. Because the concepts of data flow and task to task synchronization are part of the semantics of the Ada rendez-vous, PAM's process graphs overcome one of the major limitations of data flow diagrams for real-time applications. This makes PAMELA applicable to the requirements analysis phase. Most importantly, PAMELA defines a limited number of "process idioms" and provides rules for their use. These rules guide the analyst in a very smooth transition between requirements analysis and preliminary design. It is this author's personal style to indicate the applied rules by their number on the process graph. For instance, the symbols [1,6 | S] at the bottom of the TLM_stream_multibuf box in fig. 2a, indicate that this Single thread process (S), results from a user's requirement to provide an asynchronous interface (rule 1) of an application independent and hardware dependent nature (rule 6). The "?" and "!" show which process requested or originated the data flow, a control information vital to real-time applications (but specifically forbidden on DeMarco's DFDs).

During the preliminary design phase, the hierarchy of process graphs is mapped to Ada constructs such as abstract data types (type definition, procedures and functions), packages and tasks specification objects by a small set of simple rules. These rules encourage the re-use of library units. To simplify, multiple thread processes are mapped to packages. These packages encapsulate the single thread processes mapped to Ada tasks. "The leaves of the tree of this hierarchical structure are the procedures and functions invoked by the single thread processes." [Cherry-85]

In the detailed design phase, Ada PDL is entered in the preliminary design object bodies. This PDL is then refined into Ada code.

We found that PAMELA builds on proven modern software engineering techniques (DeMarco, Parnas, Hoare, Myers) to provide a very smooth transition between all software development phases; a quality deemed fundamental in the methodman document [Methodman-82]. Furthermore, "PAMELA uses all of Ada's advanced features (generics, packages, tasks, exceptions, and both forms of separate compilation) wisely and effectively. PAM adds a welcome limitation, form, and rationale to the use of Ada's many features which, without a suitable design and programming discipline, can and likely will be used in bizarre, ineffective, and inefficient ways." [Cherry-84]

THE VIEWGRAPH MATERIALS

of the

D. ROY PRESENTATION FOLLOW

Century Computing

EVALUATION OF ADA

FOR MSOCC

DANIEL ROY

CENTURY COMPUTING, INCORPORATED

953-3330

Century Computing

ORIGINS OF CODE 511's ADA STUDY

- o CENTURY'S "SOFTWARE TOOLS AND METHODOLOGY" STUDY FOR MSOCC IN 1984
- o ADA IDENTIFIED AS A PROMISING TECHNOLOGY
- o NEED TO EVALUATE ADA FOR MSOCC
  - ASSESS THE ADA LANGUAGE
  - DEMONSTRATE ITS USE ON A SMALL PILOT PROJECT

REQUIREMENTS ANALYSIS

D. Roy
Century Computing, Inc.

PILOT PROJECT: REQUIREMENTS ANALYSIS

o WE PERFORMED A STANDARD STRUCTURED ANALYSIS FIRST

- USING TEXT EDITOR TEMPLATES
- AND EXISTING VMS TOOLS (SEARCH, RUNOFF, EDT)

o USING THE TOOLS, WE PRODUCED

- A FULL DATA DICTIONARY
- ALL MINI SPECIFICATIONS

o WE DOCUMENTED THE PROBLEMS OF SA WITH REALTIME APPLICATIONS

OPCON

OPCON is the benchmark software's operator interface (>OPCON-val-op-int). It also controls the initial activation and the shutdown of the system's other tasks.

SPECIFICATION

```
Level-1-single-tasks is (       EVEPRT, -- Events printer
                                TIMLOD) -- CPU time loader

Begin

     1.  Prompt operator for Run-params

     2.  Activate OCC simulator            -- >OPCON-ver-OCC-act

     3.  for task in Level-1-single-tasks

         1.  Activate task                 -- >OPCON-ver-st-act

     4.  end loop

     5.  for i = 1 to IDLE-number-tasks

         1.  Activate IDLE-i               -- >OPCON-ver-idle-act

     6.  end loop

     7.  delay req-run-time                -- >OPCON-ver-run-time

     8.  Shutdown all activated tasks

     9.  delay 1 second           -- See note 2 >OPCON-ver-shut-time

    10.  Print stat-report (PRTRPT) -- >OPCON-val-stat-rep

end
```

Fig 4-3: Minispec example built with the tools

Century Computing

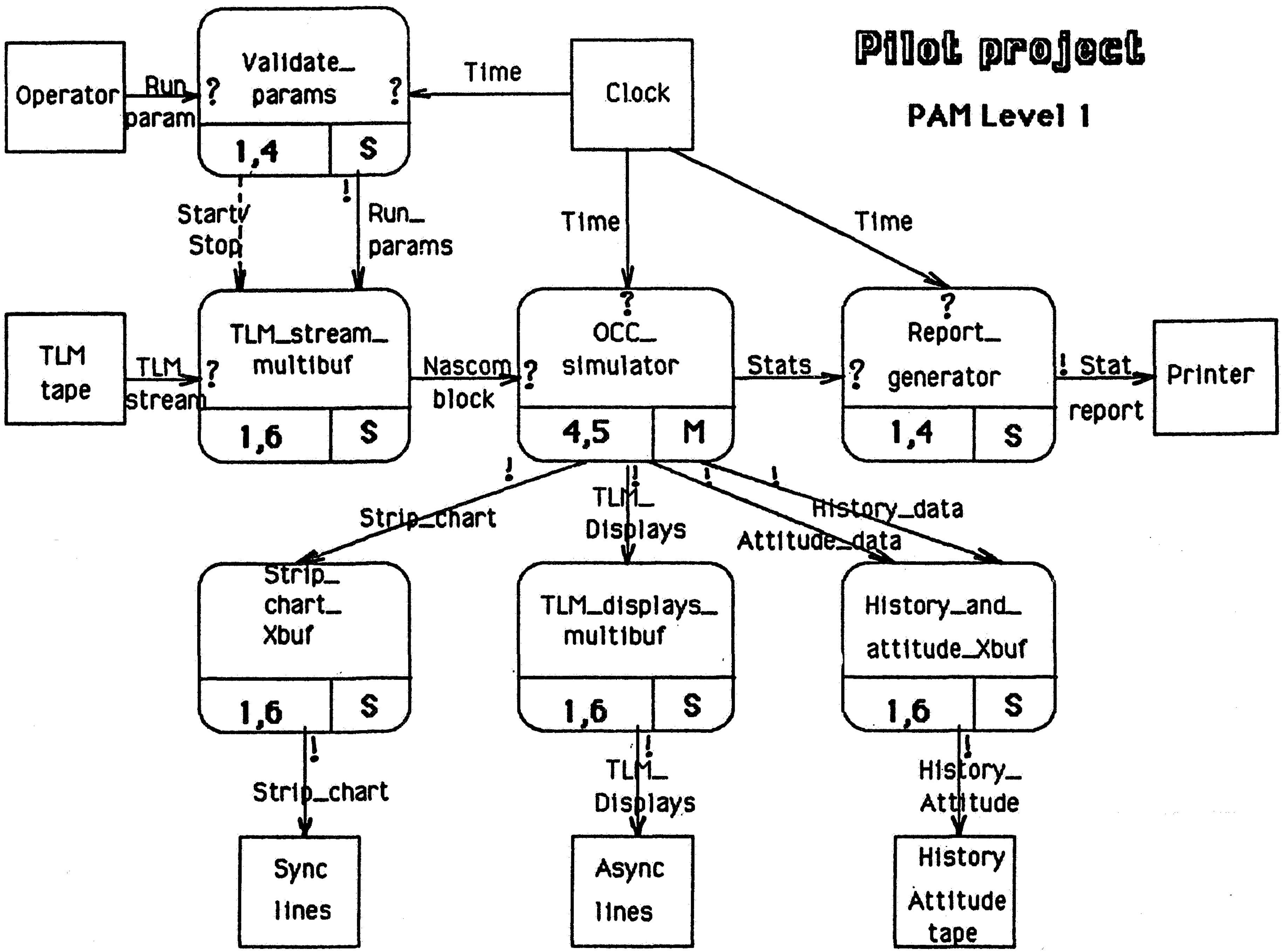
THE PROCESS ABSTRACTION METHODOLOGY

FOR EMBEDDED LARGE APPLICATIONS

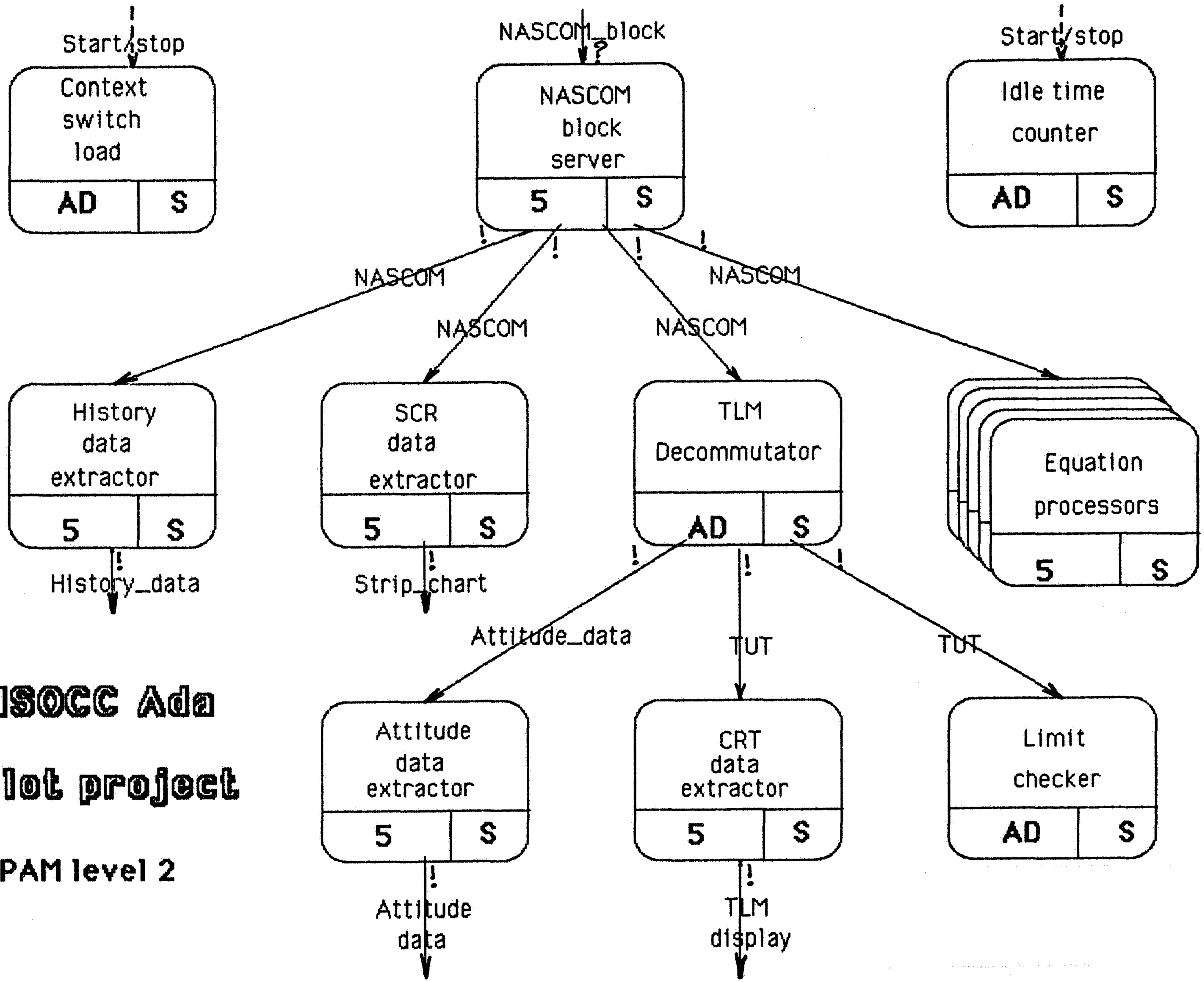
(PAMELA OR PAM)

MSOCC Ada Pilot project

PAM Level 1



D. Roy
Century Computing, Inc.



MSOCC Ada

Pilot project

PAM level 2

Century Computing

PRELIMINARY DESIGN

BARON preliminary design help

```
GOLB B => BARON TBD package              GOLD C => --| (doc), --* (PDL)
GOLD D => Bring in DOC template          GOLD E => Task entry
GOLD F => Function                       GOLD H => This text
GOLD P => Package                        GOLD S => Procedure
GOLD T => Task                           GOLD W => Bring WITH$EBP file in
GOLD X => Exception

GOLD > => half tab adjust right (*)      GOLD < => half tab adjust left (*)
GOLD TAB => half tab                     GOLD DEL => delete half tab (**)

(*) Must select range first like you would for tab adjust (control T)
(**) Careful, really does "delete" 4 times.

                BE SHORT IN PRELIMINARY DESIGN DOCUMENTATION

Algorithm:
        Can be ref to textbook and other biblio.
Effects:        --| mini-spec:
        Describes module functional requirements (more detailed than overview).
Errors:
        Describes error messages issued by module.
Modifies:       --| Side effects:
        Lists non-local variables modified (x.all. Access values, Global var).
Notes:
        User oriented description of dependencies, limitations, version
        number, status (prel des, code, etc.).  Limit change log to
        package level.
Overview:       --| Purpose:
        Describes module usage in very general terms.
Raises:
        Lists the exceptions that can be raised and not handled by module.
Requires:       --| Assumptions:
        Warns designer and user about limitations of implementation.
Synchronization:
        Describes synchronization requirements, tasks termination conditions,
        rendezvous time-outs, deadlocks prevention and other tasking reqs.
Tuning:         --| Performances:
        Specify timing and performance requirements.  Addresses performance
        issues that user can control.
```

Fig. 4-10: Preliminary design tool help

```
Package TBD is          --| Decision deferral package --*
--| Raises:
--|     None
--| Overview:    --| Purpose:
--|     This is an improvement over Intermetrics' TBD package and IEEE 990
--|     recommendations about decision deferral techniques.
--| Effects:     --| Description:
--|     The distinction is clarified between types, variables and values.
--|     The naming is more consistent (enum_i, component_i ...) and more
--|             readable (scalar_variable intead of scalarValue)
--|     There are more definitions (enum_type, record_type)
--|     Better compatibility with BYRON (or search utility processing)
--| Requires:    --| Assumptions:
--|     Please only "WITH" this package.  By systematically specifying
--|     "TBD.x" items, it is easier to assess the stage of development of
--|     a compilation unit.
--| Notes:
--|    Change log:
--| Daniel Roy   9-AUG-1985     Baseline
--                                                                     --|
--      Constants
    some_constant : constant := 1;                                     --|
    positive_constant : constant := 10;                                --|
    negative_constant : constant := -10;                               --|
    real_constant : constant := 1.0;                                   --|

--      Defer decision about type (real),(discrete(enum,integer)), subtype
--      (natural,defined subtypes), range etc... that belong to detail design
    subtype some_type is integer range integer'first .. integer'last;   --|
    subtype scalar_type is integer range integer'first .. integer'last; --|

--      Distinguishes between type, variable and value (enum_1).
--      By convention (consistent with math notation) n is last.
--      Should be Enumeration_... all over for consistency.
--      But this is so much more comfortable.
    type enum_type is (enum_1, enum_2, enum_i, enum_p, enum_n);        --|
    enum_variable : enum_type := enum_1;                               --|
                                                                       --|
--      Keep consistency with enum_type
    type record_type is record                                         --|
        component_1 : integer := 0;                                    --|
        component_2 : integer := 0;                                    --|
        component_i : integer := 0;                                    --|
        component_p : integer := 0;                                    --|
        component_n : integer := 0;                                    --|
    end record;                                                        --|
    record_variable : record_type;                                     --|
                                                                       --|
--      Inspired by IBM PDL stuff
    Condition,CD : Boolean := true;                                    --|
                                                                       --|
--      Queues services
    type queue_type is array (array_index_type) of integer;            --|
    type queue_ptr_type is access queue_type;                          --|
--
end TBD;        --| --*
```

Century Computing

DETAILED DESIGN

```
--------------------------------------------------------------------------
--
    procedure P (          --| synopsis --*
        param_1 : IN OUT some_type := some_constant ;    --| description --*
        param_n : IN OUT some_type                 --| description --*
        ) ;                                --| --*
```

Fig. 4-7: Preliminary design template for procedure (proc spec)

```
--------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
    separate ( )           --| --*
    procedure body P (     --| synopsis.  Must be the same as in body. --*
        param_1 : IN OUT some_type := some_constant ;    --| description --*
        param_n : IN OUT some_type                 --| description --*
        ) is                               --| --*
--|
-- ****** Cut and paste from specification.  Use Gold D for rest of DOC. ******
--
-- Packages

-- types

-- subtypes

-- constants

-- records

-- variables

-- functions

-- procedures

-- separate clauses

    begin          --| --*
        null;
    end P ;        --| --*
```

Fig. 4-8: Detailed design template for a procedure (proc body)

```
--------------------------------------------------------------------------
```

```
separate (mbuf) --| --*
task body P is   --| processing task --*
--|
    procedure process_block (   --| Do something useful --*
        inp_ptr : IN data_ptr_type;      --| for input blocks --*
        outp_ptr : IN data_ptr_type      --| for output block --*
        ) ;                              --| --*
--
    procedure put_blocks (      --| Dump block queue --*
        Queue : IN out_Q_type   --| Where all output blocks are queued --*
        ) ;                              --| --*
--
begin   --| P --*
--
<<exception_block>>     --*
    begin       --* for recoverable exceptions
    --
    << till_EOF >>        --| loop until all input tasks are terminated --*
        while TBD.CD loop       --* Verification:
        << build_out_Q >>       --| loop until EOF or output queue full --*
            while TBD.condition loop    --* Verification:
                --* get in_ptr (RV with I tasks)
                process_block (in_ptr, out_ptr);        --*
                --* build queue
            end loop;   --* build_out_Q
            --
            put_blocks (out_queue);     --* watch EOF case
        end loop;       --* till_EOF
        --
            exception   --| --*
                when others     =>      --| --*
        --
        --| end exception;      --*
        --
    end ;       --* <<exception_block>>
--
    exception   --| --*
        when others     =>      --| --*
--
--| end exception;      --*
--
end P ; --| --*
```

CODE AND TEST

```
                       BARON code help
                       ---------------

Gold A Access type                         Gold M Modulo statement
Gold B Block statement (range,rename)      Gold N NEW (instantiations/access/tasks)
Gold C Case statement                      Gold P Package use examples
Gold D Bring in doc template               Gold R Record (variable clause)
Gold E Entry statement                     Gold S Procedure (declaration and code)
Gold F Function (declaration and code)     Gold T Tasks (select,terminate)
Gold G Generics (overloading)              Gold U Predefined attributes
Gold H This HELP menu                      Gold W ?
Gold I IF-THEN-ELSE statement              Gold X Exception (raise)
Gold L Loop statements

GOLD > => half tab adjust right (*)        GOLD < => half tab adjust left (*)
GOLD TAB => half tab                       GOLD DEL => delete half tab (**)

(*) Must select range first like you would for tab adjust (control T)
(**) Careful, really does "delete" 4 times.
```

Fig. 4-15: Code and unit test tools built-in help

```
--------------------------------------------------------------------------------
-
<<label>>       --*
    select      --*
        --* task.entry (params);
    or | else   --*
        --* delay (time_out) | any_other_statement
    end select; --* <<label>>
```

Fig. 4-20a: Entry call template copied in program

```
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Selective entry call (no more that 2 alternatives !)
<<TLM_in>>      --*  calls TLM_stream_multibuf.do_you_have_a_block ?
    select      --*
        TLM_stream_multibuf.do_you_have_a_block (nascom_block_Xbuff);
    else        --*
        --* increment TLM_stream_multibuf overrun
        TLM_stream_multibuf_stat.increment (overrun);
    end select; --* <<TLM_in>>

Selective WAIT (any number of alternatives)
<<scr_loop>>    --* Accept and send block
    loop        --*
        select  --*
            accept here_is_a_block (     --| Accept NASCOM block --*
                nascom_block_Xbuff : IN nascom_block_Xbuff_type --| --*
                ) do    --| --*
                local_block := nascom_block_Xbuff ;
            end here_is_a_block ;        --| --*
            --* calls strip_chart_multibuf.here_is_a_set !
            put_line ("SCR_data_extractor saw a block");
        or      --*
            terminate;  -- could be delay for time-out
        end select;     --*
    end loop;   --* scr_loop
```

Fig. 4-20b: The examples buffer for task entries

```
--------------------------------------------------------------------------------
```

Century Computing

METRICS

METRICS

- o SOFTWARE ENGINEERING LABORATORY WORK WAS AT THE BASIS OF OUR METRICS EFFORT
- o DEVELOPED SIMPLE TOOLS IN DCL (LOGGER, LOC COUNTER)
- o DEVELOPED A REFINED WBS
- o KEPT ON LINE SEVERAL FILES DOCUMENTING OUR EFFORT
  - WEEKLY REPORTS
  - PROBLEMS.ADA (INCLUDING COMMENTS .LIS, ETC...)
  - GREAT.ADA (FOR NICE FEATURES OF THE LANGUAGE AND COMPILER)

RESULTS

	Hours	%
Training	253	22.9
Requirements	105	9.5
Design	93	8.4
Code/test	335	30.3
Tools dev	319	28.9

Fig. 4-17: Development data

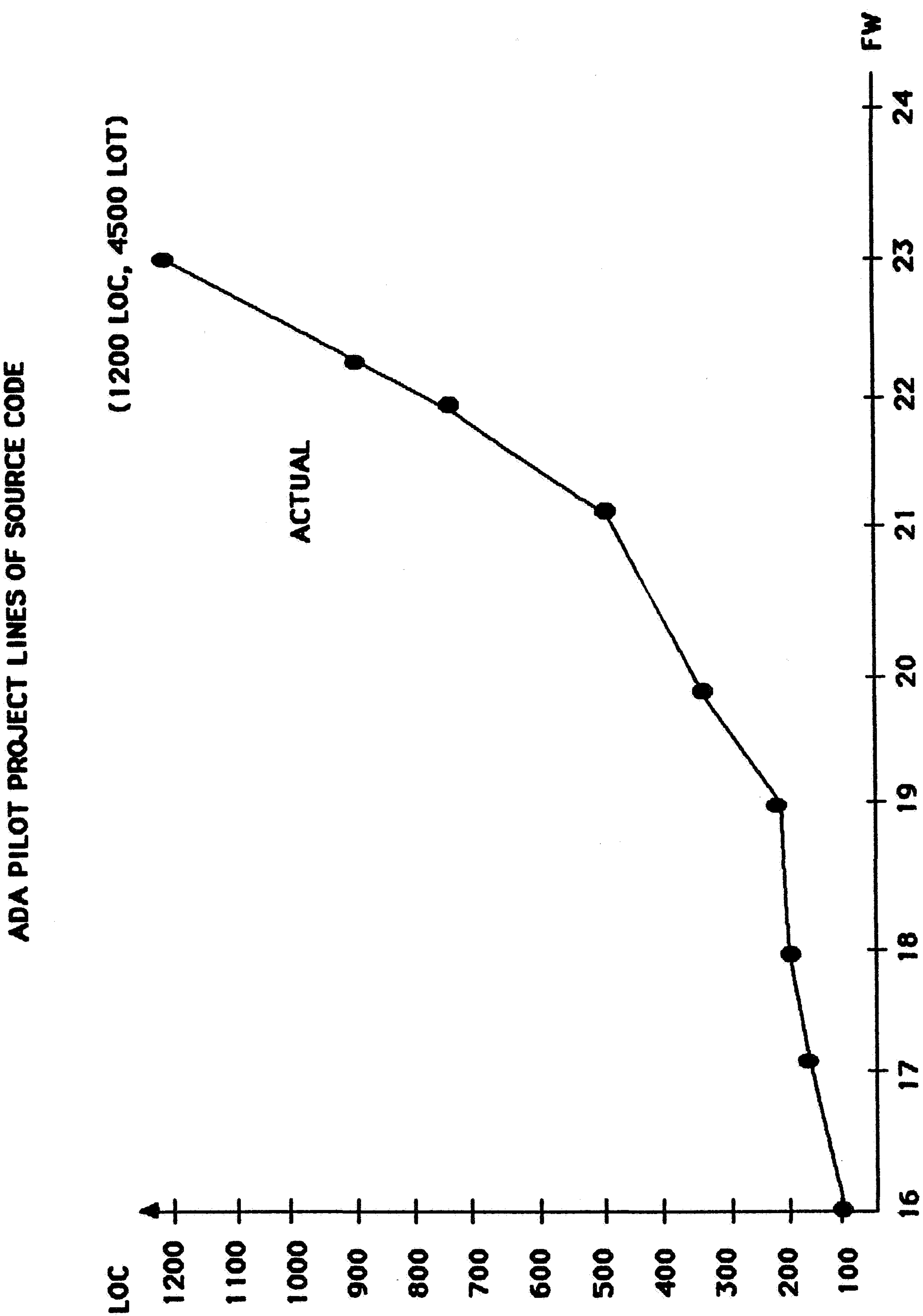

ADA PILOT PROJECT LINES OF SOURCE CODE
(1200 LOC, 4500 LOT)
ACTUAL
LOC
1200
1100
1000
900
800
700
600
500
400
300
200
100
16
17
18
19
20
21
22
23
24
FW

Figure 2-1

Century Computing

CONCLUSIONS

- o ADA VERSATILITY (PSL, DDL, PDL, IL)
- o ADA COMPLEXITY (NEED FOR A METHODOLOGY AND TOOLS)
- o START TRAINING NOW (SE FIRST, ADA SECOND)
- o PILOT PROJECT IS THE WAY TO GO (NO SCHEDULE PRESSURE)
- o TASKING WORKED WELL FOR US (TASK TYPES, I/O CONCURRENCY)
- o PAMELA WORKED VERY WELL FOR US (PRODUCED EFFICIENT DESIGN)
- o DEC ACS IS A SUPERB IMPLEMENTATION