Experience with a Software Engineering Environment Framework

by
R. Blumberg, A. Reedy, and E. Yodis
Planning Research Corporation

1.0 Introduction

This paper describes PRC's experience to date with a software engineering environment framework tool called the Automated Product Control Environment (APCE). The paper presents the goals of the framework design, an overview of the major functions and features of the framework, a summary of APCE use to date, and the results and lessons learned from the implementation and use of the framework. Conclusions are drawn from these results and the framework approach is briefly compared to other software development environment approaches.

2.0 Framework Goals

The APCE was developed to reduce software lifecycle costs. The approach taken was to increase automation of the software lifecycle process and thereby to increase productivity. It was felt that maximum cost reduction could be achieved for the short term by attacking three major problem areas:

- automation of labor intensive but routine administrative tasks

- provision of an overall control, coordination, and enforcement framework and information repository for existing tools

- provision for maximum framework portability, distributability, and data interoperability with the bounds of performance constraints

A distinction was made between tools and the environment. In the PRC view, tools are active elements in the software lifecycle process. They create or modify (document or software) components, test components, or order the execution of groups of tools upon components. The environment or framework, on the other hand, is a more passive element. It provides for overall control, coordination, and enforcement and acts as an information repository. This distinction is important because it serves to separate environment or framework issues from tool issues. PRC wanted to build a framework which could incorporate existing tools. In this way, PRC could build on the excellent work done by others in the tool arena in a timely fashion.

3.0 APCE Overview

The APCE provides automation for:

- real-time project status tracking and reporting

- configuration management of software, documentation, and test procedures

- requirements traceability and change impact traceability

- test bed generation, component integration, and system integration

A brief overview of how the APCE is organized to support these functions and how the APCE is designed to support portability, distributability, and interoperability is given below.

3.1 Automation and Control

As suggested by Stoneman [1], a database provides the integrating mechanism for the environment framework. The database design incorporates a flexible model of the software development process. Project definition information based on this model is entered into the database during project initialization, and this information is used to control the project and provide the basis for automated tracking and configuration management. The project definition is divided into three components as illustrated in Slide 3 (APCE Entities).

User groups are identified as managers, developers (those who create products), or testers; multiple roles are allowed. The organizational hierarchy is also described so that project problem reports can be automatically forwarded up the chain of command if they are not promptly dealt with. Products, both documents and software, are described in terms of their component structure and are associated with software lifecycle phases which are also entered into the APCE database. Slide 4 (APCE - DOD Documentation and Review Sequence) illustrates the lifecycle phases as specified in Mil-Std 2167.

The levels of integration describe the hierarchy of the test and integration processes that all products (documents or software) must go through. This testing process allows for the enforcement of project standards and quality assurance. The APCE uses the product structure and test procedures developed by the testers to automatically create testing baselines and test harnesses as required.

3.2 Portability, Distributability, and Interoperability

The APCE approach to support for portability, distributability, and interoperability is based on three architectural features:

- o APCE Interface Set (AIS)
- o data-coupled design
- o open system architecture

These features are illustrated in Figure 1 (APCE Static View), which shows the APCE as part of a Software Engineering Environment (SEE).

The APCE subsystems and data management capabilities depend on a standard set of interfaces to system services called the AIS. These interfaces define a Stoneman Kernal Ada® Programming Support Environment (KAPSE) like layer for portability purposes. The AIS allows a mapping to existing operating system services. If the needed level of support is not directly available from the host operating system, then an extra layer of software is created to satisfy the requirement. Existing operating system services are not duplicated. The AIS is not based on an implicit model like the Common APSE Interface Set (CAIS) [2].

The data-coupled design provides for both control and distributability. All project information is stored in the framework database. The database controls the activities of the APCE functional subsystems since they do not interface directly but interact through the database. Users do not directly manipulate the database; they affect the database contents indirectly through interaction with the functional subsystems. The database is designed to minimize information exchange, so data is distributable (without replication). The functional subsystems are also distributable since they are controlled by the database contents. The database design is controled by the framework and hidden from the users. Thus, integrity and interoperability of data is ensured.

The open system architecture approach means that the APCE allows the use of existing host tools, including management scheduling and costing tools. The APCE does not interface directly with the tools but rather controls tool invocation and the tool products. Both existing and future tools can be used within the APCE framework without alterations.

4.0 Results

The APCE has been used on a variety of in-house and client projects over the past 21 months. It has been used in-house at PRC to support proposal and document production as well as software development and lifecycle maintenance projects. The framework has also been installed for Army, Navy, and Air Force clients. In one example client installation, APCE features were used to bring a software system under configuration control for a Navy software support activity. The

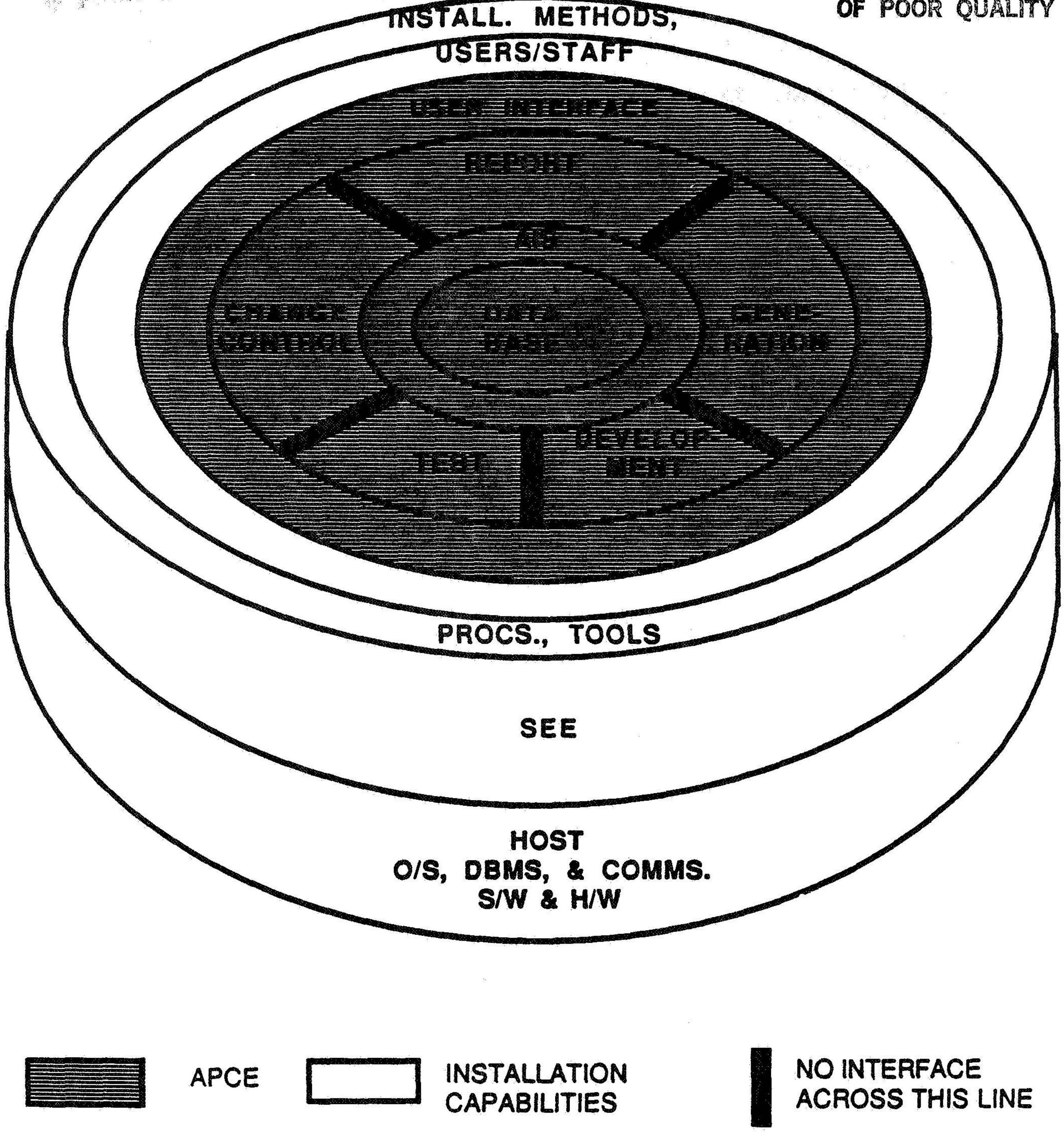


FIGURE 1: APCE STATIC VIEW

The full project team for Project 1 consisted of 9 persons, including a manager, 2 computer system scientists, 1 system analyst, 1 analyst, and 4 associate analyst/programmers. Two of the associate analyst/programmers acted as the test team. All of the other team members, except the manager, functioned as APCE developers. The senior staff members were quite experienced with 10 to 15 years experience each. The junior staff members were all new college graduates with no commercial programming experience and no VAX experience. The APCE allowed all personnel to be extremely productive despite their learning curve with a new machine and a new environment.

4.2 Cost/Benefit Analysis

PRC has conducted a cost/benefits analysis of APCE use for one of our clients. This client needed configuration management and lifecycle maintenance control for mission critical software. PRC developed plans for both a manual and a APCE controlled development support facility and plans for transitions to these facilities. A estimation of both the transition costs and the annual recurring resource costs was performed for both facilities. The results of the analysis are given on Slides 6 (Level of Effort Analysis) and 7 (Cumulative Cost Comparison).

The estimated times for transition to both the manual and the APCE controlled facilities were the same (3 months). The activities involved in the transition period involve the establishment and implementation of policies and procedures and, in the case of the APCE controlled facility, the installation of software and training. As shown on Slide 6 (Level of Effort Analysis), the cost for transition in terms of effort was slightly more for the APCE controlled facility. However, the total labor months required for the first year and following years were very much less for the APCE controlled facilities.

Slide 7 (Cumulative Cost Comparison) shows the total cumulative costs of the two facilities projected over a two year period. The larger initial costs for the APCE controlled facility is caused by the APCE licensing fees. The cumulative costs of the manual facility surpass the costs of the APCE controlled facility after seven months (4 months after transition). The cost savings achieved by the APCE facility are due to the increased automation of the control, tracking, and configuration management functions. The estimates did not include cost savings due to increased productivity of developers and testers.

4.3 Portability

The framework has proven very easy to rehost. Part of this ease is due to the design features of the AIS and part is due to rigid enforcement of coding standards for the transportable portions of the APCE. To rehost the APCE on a new machine, all that is necessary is to reimplement the AIS functions. The APCE transportable subsystems have been written in C using coding standards designed to eliminate use of "non-standard" features of the language. The C programming language was orginally framework is now being used to continue control throughout the maintenance cycle, including the incorporation of module upgrades supplied by other contractors. These various applications of the framework have resulted in rehosting of the APCE to a variety of different hardware configurations. This experience in using the APCE has allowed PRC to collect the data on productivity, transportability, and distributability presented below.

4.1 Productivity

At the National Security Industrial Association (NSIA) DOD/Industry Software Technology for Adaptable, Reliable Systems (STARS) Program Conference in April 1984 [3, pg. L-21], the NSIA Industry Study Task Group reported that the average productivity for U.S. software development projects was 200 lines of code per labor month. This works out to a little over 10 lines per day. On unclassified projects with APCE control, PRC has recorded productivity in excess of 120 lines per day. Slide 5 (Example Projects) gives the productivity figures collected for three PRC in-house projects under APCE control. (Client projects are not far enough along to report meaningful productivity figures.) Productivity in these three projects was an order of magnitude greater that the average reported for industry as a whole.

All of the reported projects used a high level programming language (HOL). Project 1 was the initial development of a software system. This system has been maintained under APCE control. The figures given for Project 1 reflect only the developers' labor and do not count time for the manager or the testers (who basically functioned as Quality Assurance personnel). Productivity during upgrades was equivalent or better than that experienced during the initial development. Further details of Project 1 are given below. Project 2 was an upgrade to an existing system under APCE control. This upgrade included full documentation. Project 3 was a prototyping activity, and is somewhat atypical since only partial documentation (e.g., no formal users manual) was produced. The figures given for Project 2 and Project 3 include the testers' time. These projects were small, so the same personnel functioned as both developers and testers.

Project 1 was a four month project to develop system software in the C programming language. The development host was a VAX 11/780 and the non-APCE tools used are commercially available for the VAX. The project products included: System Engineering Plan, Acceptance Test Plan, Functional Description, Preliminary Design Specification, Detailed Specification (22,000 lines of Ada PDL), Operators Manual, and Users Guide in addition to 58,297 lines of source code. In addition, 660 test procedures were developed and used to test the components of the products. (The test procedure development and test time is not included in the productivity figures given for Project 1.) Some of these test procedures were used to enforce the project specific coding and PDL standards.

chosen because it was available on a wide range of host machines. However, it has caused some problems because there are no standards for C. In the process of transporting, some features of C that were assumed to be commonly implemented turned out to be system specific. A single version of the transportable software is maintained that runs on all supported machines. (Future plans call for conversion to Ada as soon as there are Ada compilers on a sufficiently wide range of machines.)

The APCE is now running on the following machines: VAX 11/780 with VMS, ROLM and Data General with AOS/VS, IBM with MVS, and Intel 310 with XENIX®. Slide 8 (Rehost Efforts) presents a summary of rehosting experiences to date.

4.4 Distributability and Interoperability

The environment data is interoperable because the framework controls the database structure and because the framework controls only the products of tools rather than interfacing directly with the tools. The toolsets available on different hosts may differ, but equivalent functionality is usually available. Filters and standard forms can be used to adjust for differences between specific tools. For example, different editors sometimes embbed control characters in the text. Filters are used at PRC head quarters to move text among the VAX EDT editor, the IBM PC Wordstar editor, and the Macintosh MacWrite editor. A standard, plain text form has been established so that only one new filter needs to be written to introduce another editor.

Project data has been proven to be interoperable between different framework installations. Software and documentation have been routinely developed on one installation and then transferred together with documentation, traceability and configuration management information, and project history information to a different installation on different hardware with no problems. This feature has proven useful in allowing project work to proceed in parallel with the APCE rehost to new hardware. That is, the early phases of a project can begin under APCE control on one machine while the APCE is rehosted to the desired development host. When the rehost is complete, the project can be transfered to its own host.

The framework was designed to function in a distributed, heterogeneous hardware environment. Both the database and the processing may be distributed. Work currently underway will allow distribution of developer processing to IBM PCs and Macintoshes connected to a VAX via a local area network. Future plans call for full distribution of both processing and data.

5.0 Conclusions

The preliminary results presented above provide good evidence that the APCE approach can achieve its goals. The framework increases productivity, allows use of existing tool's without modification, and is easy to transport. PRC management has been impressed enough to make the

APCE a company standard. The task of technology insertion into large projects has begun. Because of its flexiblity, the APCE can be introduced into existing projects without undue disruption. Most of the transition problems are in the areas of training. The use of the APCE does involve understanding of some basic concepts. During the next few years, more data will be collected on the benefits of using this type of environment framework.

The APCE framework approach is in contrast with other environment approaches both in the areas of goals and of benefits. Many other recently developed environments, such as the Ada Language System (ALS) [4], have a very different set of goals. One of the goals of the ALS is to provide a minimal set of transportable tools including a retargetable Ada compiler. Much of the effort expended in the ALS development has been to develop tools, especially the Ada compiler. Many of the benefits expected from the ALS are the benefits derived from the use of a standard toolset and command language.

The approach taken by the ALS does not allow the use of non-ALS tools. To work with the ALS, existing tools must be rehosted to the ALS KAPSE and rewritten in Ada, if necessary. The ALS tools are transported by rehosting the ALS KAPSE on new hardware just as the APCE framework is transported by rehosting the AIS on a new operating system. The ALS approach means that there will be significant lead time before the ALS has a reasonably full tool set. Further, features such as full configuration management and project reporting must be added as tools to the ALS. These important productivity tools are not part of the minimal toolset. Important aspects of the ALS approach, such as productivity and portability, have yet to be proven. The problem of distribution was not directly addressed in the first version of the ALS.

The ALS approach may work for organizations such as the U.S. Army that wish to standardize as much as possible on a minimal tool set and a limited selection of standard hardware. However, for a contractor with a wide variety of client and internal standards, methodologies, and hardware, a much more flexible approach is necessary. The APCE framework is an example of a viable alternative approach.

References

[1] Requirements for Ada Programming Support Environments ("Stoneman"),
Department of Defense, February, 1980

[2] Proposed MIL-STD-CAIS,
Department of Defense, 31 January 1985

[3] Proceedings First DOD/Industry STARS Program Conference,
NSIA, 30 April - 2 May 1985, San Diego, CA

[4] Architectural Description of the Ada Language System (ALS),
Joint Service Software Engineering Environment (JSSEE) Report No.
JSSEE-ARCH-001, 3 December 1984

[5] Architectural Description of the Automated Product Control Environment,
Draft, 4 October 1985

THE VIEWGRAPH MATERIALS

for the

R. BLUMBERG PRESENTATION FOLLOW

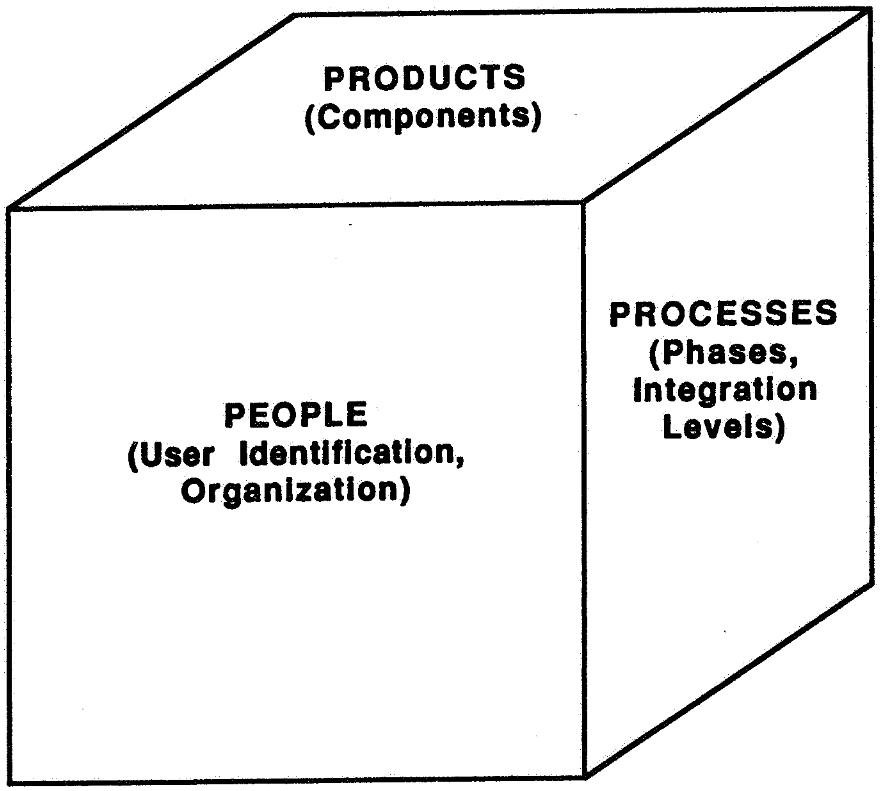
Experience with a Software Engineering Environment Framework

R. Blumberg
A. Reedy
E. Yodis

Automated Product Control Environment

- Goals
  - Environment vs tools
  - Automation
  - Portability, distributability, interoperability
- Functions
  - Project reporting
  - CM
  - Tracking
  - Test and integration

APCE Entities

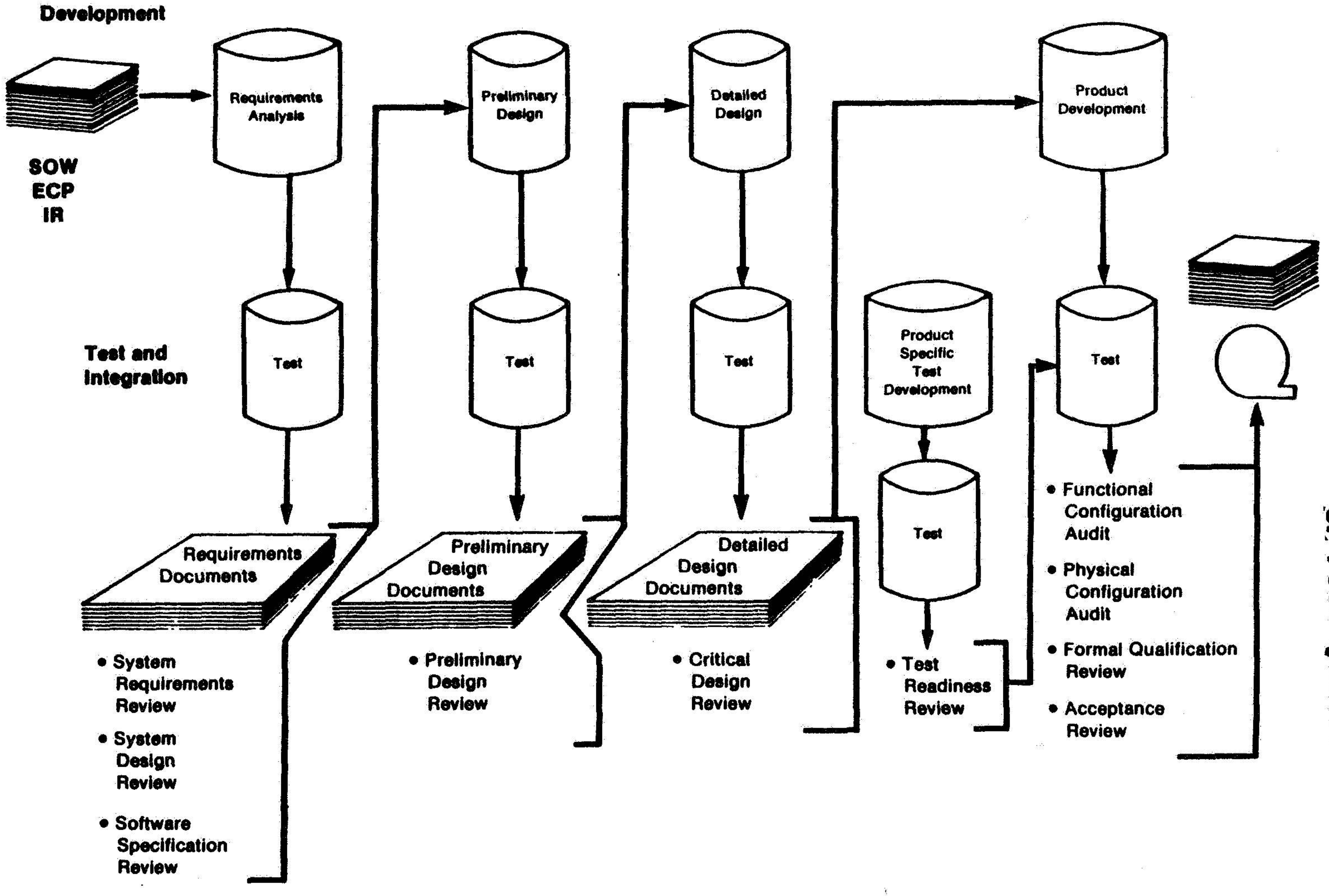


PRODUCTS
(Components)

PEOPLE
(User Identification,
Organization)

PROCESSES
(Phases,
Integration
Levels)

APCE - DOD Documentation and Review Sequence


Development
SOW
ECP
IR
Requirements Analysis
Preliminary Design
Detailed Design
Product Development
Test and Integration
Test
Test
Test
Product Specific Test Development
Test
Test
Requirements Documents
Preliminary Design Documents
Detailed Design Documents
• System Requirements Review
• System Design Review
• Software Specification Review
• Preliminary Design Review
• Critical Design Review
• Test Readiness Review
• Functional Configuration Audit
• Physical Configuration Audit
• Formal Qualification Review
• Acceptance Review

ORIGINAL PAGE IS OF POOR QUALITY

Example Projects

Project	LOC / day	Total LOC	Total personnel	Programming Language
Project 1	121	58,297	6	“C”
Project 2	217	13,024	1.5	“C”
Project 3	384	30,750	2	Pascal “C”

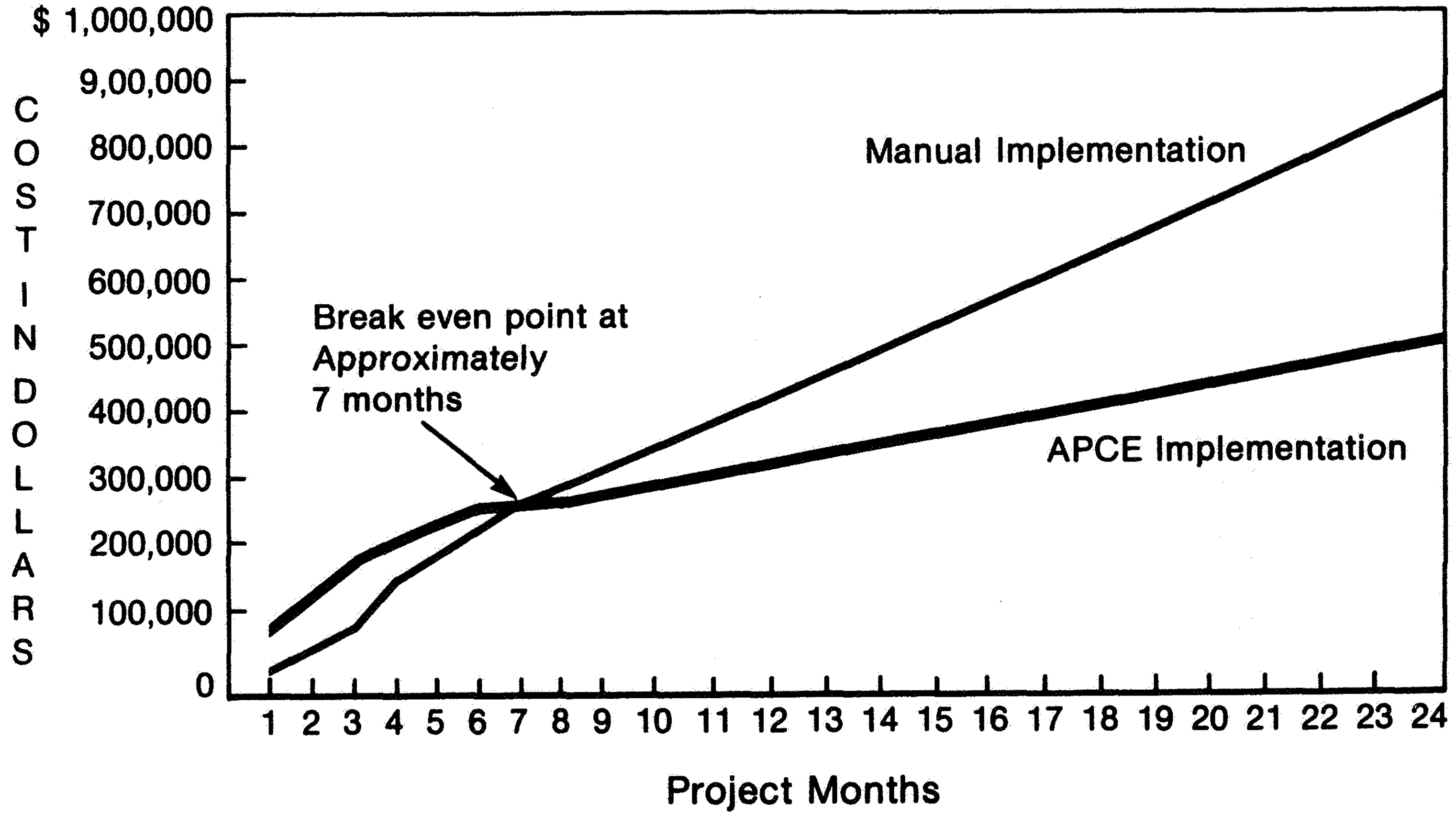
Level of Effort Analysis

First year resource costs	Manual Environment	APCE Environment
Transition activities (months 1 to 3)		
Organization	2.25	3.75
Configuration management	3.75	2.75
QA / testing	4.00	3.75
Development	1.00	1.00
Total transition activities	11.00	11.25
Recurring resource costs (months 4 to 12)	45.00	18.00
Total first year	56.00	29.25
Annual recurring resource costs		
Configuration management	15.00	6.00
QA / test management	21.00	6.00
PSL librarian	12.00	0.00
Coordinator	12.00	0.00
APCE on - site support	0.00	12.00
Total	60.00	24.00

*All values expressed in staff months

prc

Cumulative cost comparison



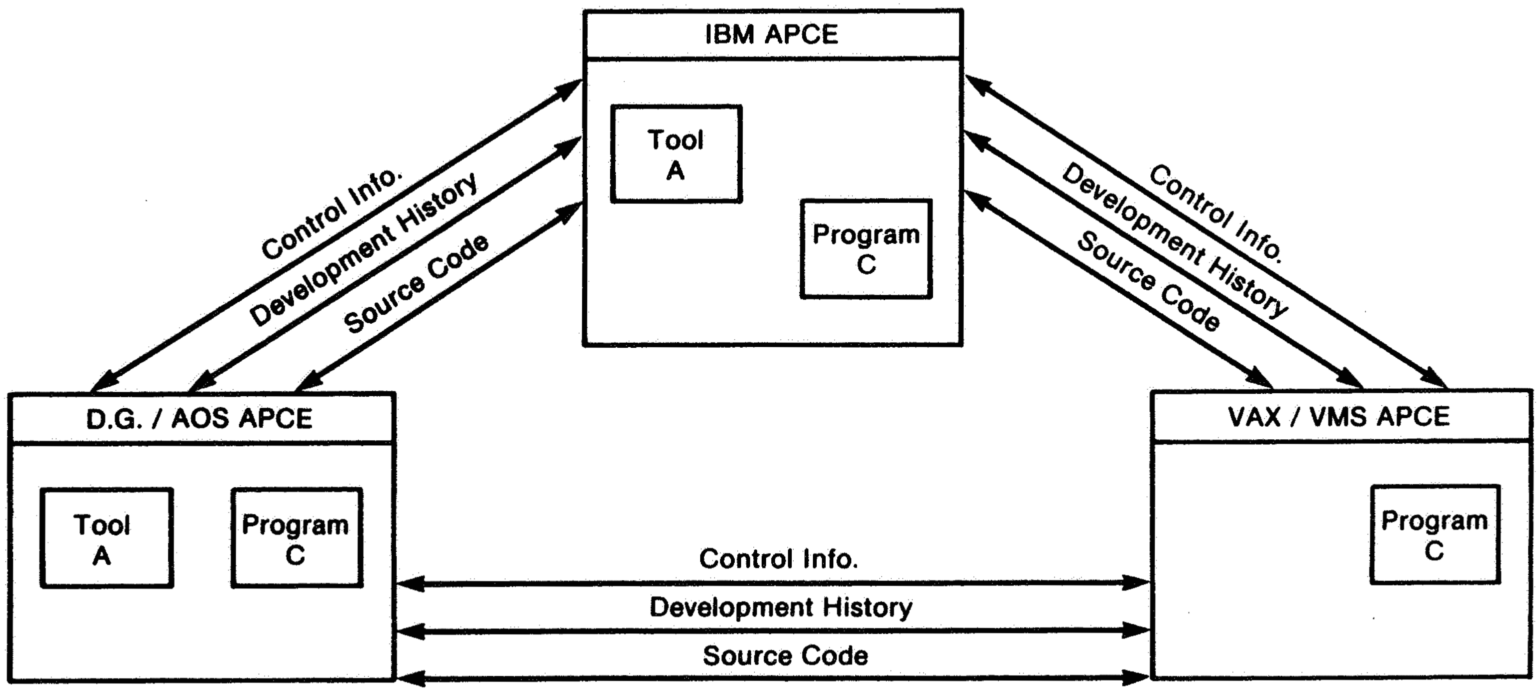
Break even point at
Approximately
7 months

Rehost Efforts

H / W	Calender Time	Team Size
ROLM - DG AOS / VS	1 Month	4 APCE analysts
IBM / MVS	2½ months	5 APCE analysts
Intel 310 XENIX	2 months	1 APCE analyst 2 site personnel

Note: VAX / VMS original host

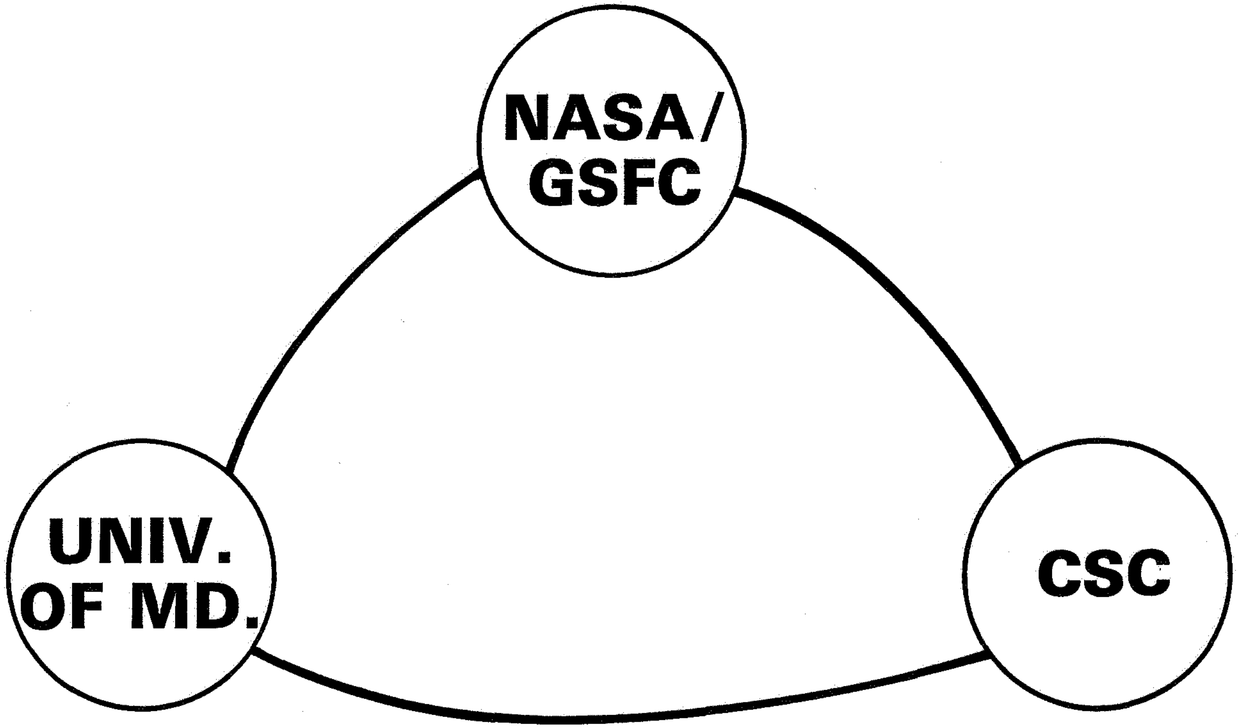
APCE Interoperability & Portability



Summary

- Different approach
- Allows use of existing tools
- Good preliminary results
- Acceptance as a corporate standard

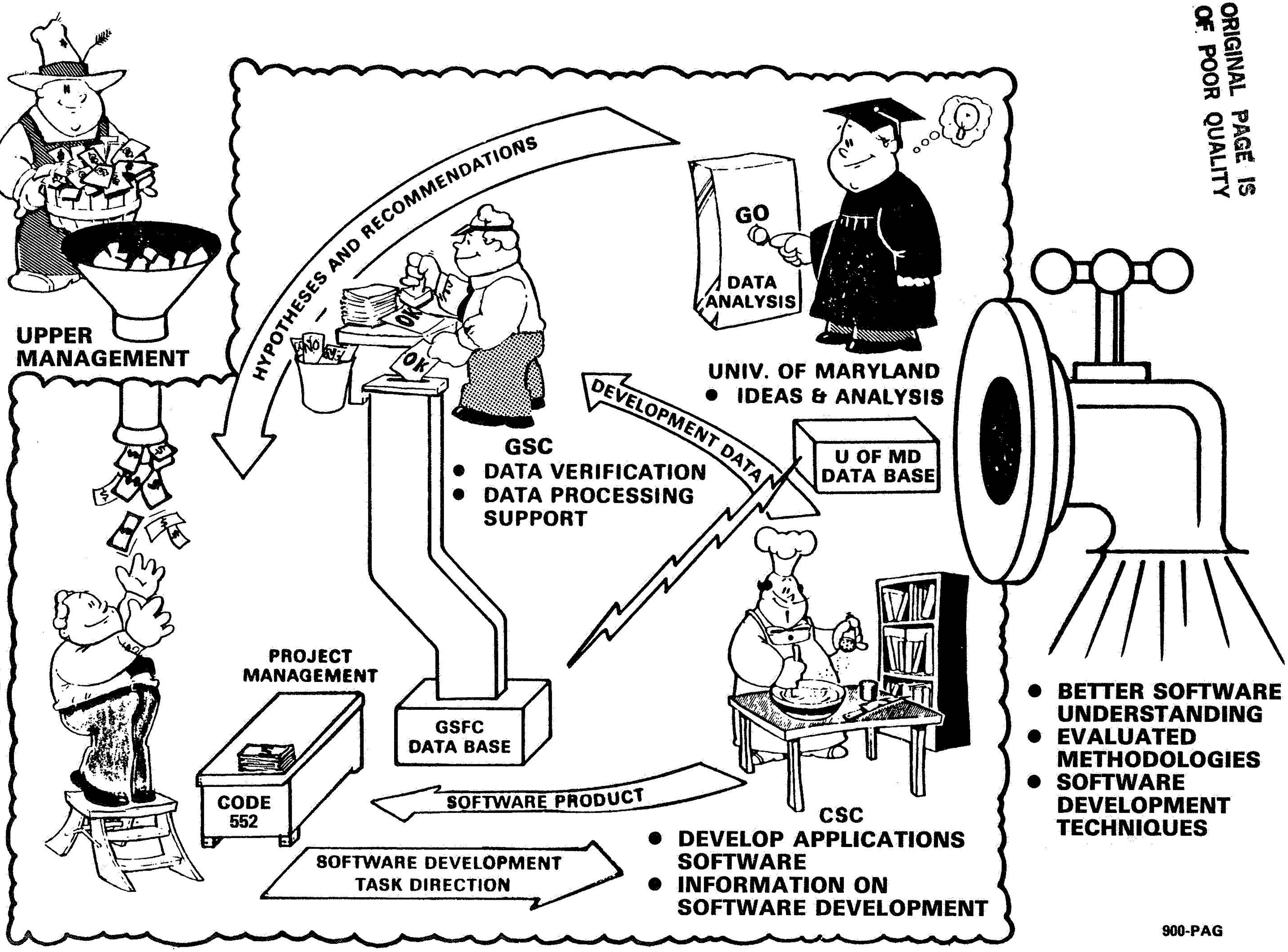
SOFTWARE ENGINEERING LABORATORY



SOFTWARE ENGINEERING LABORATORY

- **WHEN:** ESTABLISHED IN 1976 BY NASA/GSFC
- **WHY:** TO IMPROVE ITS SOFTWARE DEVELOPMENT PROCESS
- **HOW:**
  1) MEASURE SOFTWARE DEVELOPMENT PROCESS
  2) EVALUATE EXISTING TECHNOLOGY
  3) TRANSFER SUCCESSFUL TECHNOLOGIES INTO THE DEVELOPMENT PROCESS

STRUCTURE OF THE SEL


UPPER MANAGEMENT
HYPOTHESES AND RECOMMENDATIONS
OK
OK
GO
DATA ANALYSIS
UNIV. OF MARYLAND
• IDEAS & ANALYSIS
DEVELOPMENT DATA
U OF MD DATA BASE
GSC
• DATA VERIFICATION
• DATA PROCESSING SUPPORT
PROJECT MANAGEMENT
CODE 552
GSFC DATA BASE
SOFTWARE PRODUCT
SOFTWARE DEVELOPMENT TASK DIRECTION
CSC
• DEVELOP APPLICATIONS SOFTWARE
• INFORMATION ON SOFTWARE DEVELOPMENT
• BETTER SOFTWARE UNDERSTANDING
• EVALUATED METHODOLOGIES
• SOFTWARE DEVELOPMENT TECHNIQUES
900-PAG

DEVELOPMENT CHARACTERISTICS

LANGUAGES	FORTRAN (85%) MACROS (15%)
TYPE	SCIENTIFIC GROUND-BASED INTERACTIVE NEAR-REAL-TIME
SIZE	TYPICALLY ~65,000 SLOC (2,000 TO 160,000)
SCHEDULE	16 TO 25 MONTHS (FROM START OF DESIGN TO START OF OPERATIONS)
STAFFING	6 TO 18 PERSONS
COMPUTERS	IBM MAINFRAME (PRIMARILY) VAX-11/780 PDP-11/780

900-PAG-(128*)

SCOPE OF SEL ACTIVITIES
(1977 — 1985)

- COLLECTED DATA FROM MORE THAN 50 PROJECTS
  - OVER 2 MILLION LINES OF CODE PRODUCED
  - OVER 200 DEVELOPERS PARTICIPATED
  - OVER 30,000 FORMS SUBMITTED
- STUDIED ABOUT 50 STATE-OF-THE-ART TECHNOLOGIES
- PRODUCED MANY TOOLS, STANDARDS, AND MODELS FOR USE BY DEVELOPERS
  - RECOMMENDED APPROACH TO SOFTWARE DEVELOPMENT
  - MANAGER'S HANDBOOK FOR SOFTWARE DEVELOPMENT
  - AN APPROACH TO SOFTWARE COST ESTIMATION
  - SOFTWARE TEST AND VERIFICATION PRACTICES

900-PAG-(128*)