RL-TR-92-345, Vol VI (of seven)
Final Technical Report
December 1992

SYSTEM ENGINEERING CONCEPT DEMONSTRATION, Trade Studies

Software Productivity Solutions, Inc.

Andres Rudmik, Edward Comer, Sharon Rohde

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-92-345, Volume VI (of seven) has been reviewed and is approved for publication.

APPROVED: Frank A. LaMonica

FRANK S. LAMONICA
Project Engineer

FOR THE COMMANDER: John A. Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1992	Final Feb 90 - Jul 92

4. TITLE AND SUBTITLE
SYSTEM ENGINEERING CONCEPT DEMONSTRATION,
Trade Studies

5. FUNDING NUMBERS
C - F30602-90-C-0021
PE - 62702F
PR - 5581
TA - 18
WU - 54

6. AUTHOR(S)
Andres Rudmik, Edward Comer, Sharon Rohde

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Productivity Solutions, Inc.
122 4th Avenue
Indialantic FL 32903-1697

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3CB)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-92-345, Vol VI
(of seven)

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Frank S. LaMonica/(315) 330-2054

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This final technical report documents the objectives, activities, and results of Air Force contract F30602-90-C-0021, entitled "System Engineering Concept Demonstration." The effort, which was conducted by Software Productivity Solutions, Inc., with McDonnell Douglas Corporation - Douglas Aircraft Company and MTM Engineering Inc. as subcontractors, demonstrated and documented the concept of an advanced computer-based environment which provides automation for Systems Engineering tasks and activities within the Air Force computer-based systems life cycle. The report consists of seven (7) volumes as follows: I) Effort Summary, II) Systems Engineering Needs, III) Process Model, IV) Interface Standards Studies, V) Technology Assessments, VI) Trade Studies, and VII) Security Study.

This Volume (Volume VI - Trade Studies), describes several studies that were performed to 1) reduce technology risks by assessing emerging, enabling technologies that are presumed to be mature enough for application in the 5-7 year time frame, 2) make the system concept concrete and visible through realistic demonstrations using realistic data, and 3) reduce technology transfer risks through evaluation of usability and the degree to which the envisioned systems engineering automation addresses real needs.

14. SUBJECT TERMS	15. NUMBER OF PAGES
System Engineering, System Life Cycle Tools, System Life Cycle Environment	122
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

1. Introduction

This document is Volume 6 of the Final Technical Report of the System Engineering Concept Demonstration, contract F30602-90-C-0021 for the Air Force Rome Laboratory (RL). The document is organized into five sections:

- Environment Studies
- SECD Environment and Rationale
- Technology Demonstrations
- Prototypes
- Risk Evaluation

Environment Studies discusses frameworks and their integration, factors affecting tool integration, and user interface technologies. The SECD Environment and Rationale discusses a model of an environment, Catalyst building blocks and interfaces, computer system environment requirements and environment focus. Technology Demonstrations documents the identification, investigation and assessment of enabling technologies and issues which were considered critical to establishing a system engineering environment. The section, Prototypes, documents three scenarios developed at SPS during the SECD effort that demonstrated the advanced capabilities and enabling technologies of an automated system engineering environment. Risk and Cost Analysis identifies the risks and strategies to reduce the risks, and the estimated cost of the demonstration and validation of the CSCIs required to automated a system engineering environment.

The goals of the environment studies, technology demonstrations and prototypes were three fold: to reduce technology risks by assessing emerging, enabling technologies that are presumed to be mature enough for application in the 5-7 year time frame; to make the system concept concrete and visible through realistic demonstrations using realistic data; and to reduce technology transfer risks through evaluation of usability and the degree to which the system addresses real needs.

Risk factors examined were performance, market acceptability, usability, maintainability, evolvability, unit cost and feasibility of developing a system with the projected maturity of technology within cost and schedule. The technology demonstrations and the prototyping also helped to solidify the operational concepts developed during the effort.

2. Environment Studies

Key architectural trade studies were conducted for two topics:

- Environment frameworks
- User interface architectures

The following subsections provide pertinent technical background concerning these topics and outline key decisions.

2.1. Environment Frameworks

The role of an environment framework is to integrate software products and engineering methods, as well as support organizational aspects of development. Figure 2.1-1 shows that as the level of integration increases, the ability to support the definition and enforcement of products, methods, and roles also increases.

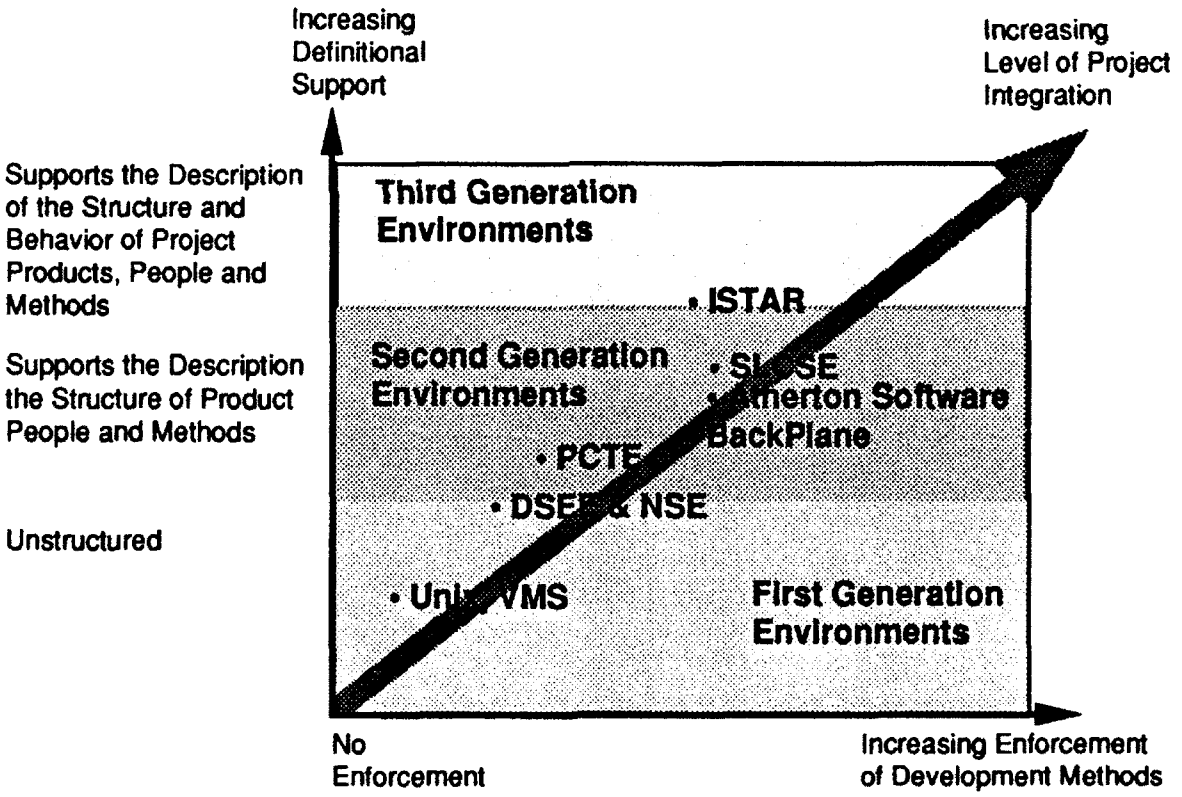


Figure 2.1-1. Levels of framework integration

The level of framework integration is used as the basis for classifying environments. Each higher level generation subsumes the lower level.

First generation environments provide the user with a simple file system and tool set. Software engineering disciplines are applied manually. Product structures or development methods are enforced through the use of tools and manual administrative procedures. Unix and VAX/VMS are examples of first generation environments.

First generation environments emphasize the availability of tools to support the development and maintenance of products. These environments place little emphasis on supporting the definition and enforcement of user roles, product structures, and software engineering methods. The primary form of integration in these environments occurs through tools which maintain and enforce product structures and may partially support a software engineering method. The disadvantage of such limited integration is that each tool must have intimate knowledge about these structures. This limitation makes it difficult to incorporate new tools, transport tools to different environments, or change the products.

Second generation environments support the definition of the product structures (objects and relationships) that represent real world objects more naturally and enforce conformance to these structures during development. A key aspect of second generation environments is that the product structure rules are defined in the environment. These product definitions are then enforced by the environment rather than through tools, thus eliminating many of the disadvantages of first generation environments.

A common approach to providing second generation capabilities is to use a database that keeps the description of the method and organizational structures employed within a software engineering environment. The key issue is that the environment supports the description of the relevant project structures but relies on tools to define the semantics of those structures.

In addition to the capabilities supported by second generation environments, third generation environments support the description of the semantics of a project's data, methods, and organizational aspects. The framework is responsible for enforcing these semantics. Tools are simply operations defined for different objects. They become indistinguishable from the objects themselves; new tool development is a specialization of existing objects. This paradigm represents the convergence of major disciplines in computer science and software engineering, including databases, programming

languages, knowledge representation, software engineering environment frameworks, and software engineering methodologies.

Third generation environments provide a much higher degree of project integration. Products, methods, and users are all part of a single conceptual framework; thus, the framework can guide and control project work, providing significant user productivity and product quality gains. For example, ISTAR [LEH86] supports a contract model of software development which includes a method model, a product model, and a user model.

Framework integration is a measure of the ability to insert tools into the framework so that they share information, are controlled uniformly, provide a consistent look and feel, and support the project methods. There are four fundamental forms of integration:

1. **Information Integration**: The framework's ability to support the sharing of data and the meaning of the data.
2. **Control Integration**: The framework's ability to manage the execution of tools.
3. **User Interface Integration**: The framework's ability to provide a common user interface across environment tools.
4. **Method Integration**: The framework's ability to control the usage of tools to conform to a particular development method.

An increased awareness of the problems of tool integration has spawned numerous efforts by governments and industry to define framework integration standards. Framework interface standards are a major factor affecting ones ability to integrate tools into a framework. Although standards are important, there are a number of other factors that need to be considered with today's frameworks given that the standards are not universally accepted nor supported.

The following discussion of framework integration technologies and approaches is illustrated with examples from existing commercial frameworks. In some cases, examples are taken from framework components, such as SUN Microsystems' NSE and Apollo's DSEE, that provide a specific framework capability. The following systems are used to illustrate framework technologies:

1. **Apollo DSEE**: (Domain Software Engineering Environment) A powerful configuration management and product building facility that provides a high degree of automation for software product management in a computer network.

2. **Atherton Software Backplane**: A commercial environment framework that provides project data management, advanced graphic user interface, work flow management, and configuration management.

3. **CAIS-A**: (Common APSE Interface Set): A proposed set of Ada interfaces and interface semantics for the tool support layer, CAIS-A provides a modified entity-relationship project database with multiple inheritance, hierarchical transactions, mandatory security, and naming flexibility. CAIS-A is a proposed DoD standard whose prototype implementations are under development.

4. **PCTE**: (Portable Common Tool Environment) A software engineering framework that provides a standard set of tool support layer interfaces. The PCTE interfaces support entity-relationship project databases, user interface services, and process and transaction management .

5. **SLCSE**: (Software Life Cycle Support Environment) A software engineering environment that supports data and process integration. The SLCSE framework provides an entity-relationship project database; common, menu-based user interface; simple transaction management; and user management.

6. **SUN NSE**: (Network Software Environment) NSE provides facilities for configuration management, linking services for building relationships among data managed by tools, and a user interface.

2.1.1. Information Integration

Information integration is critical to tool interoperability. Tool interoperability is necessary to achieve a high degree of automation and synergy among tools within the environment. Figure 2.1.1-1 illustrates different mechanisms that can be employed to share information among tools.

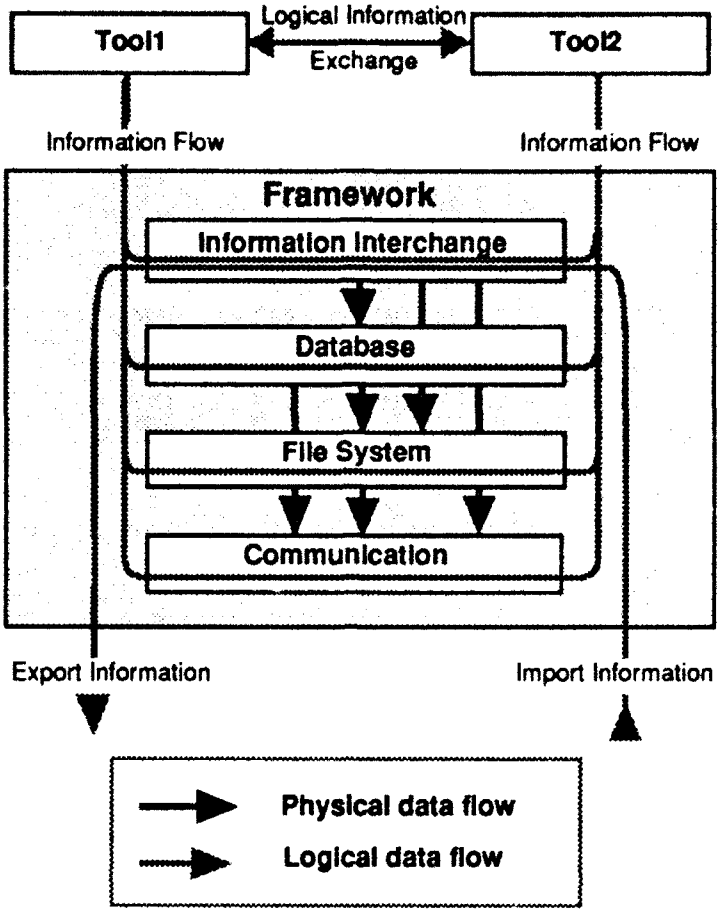


Figure 2.1.1-1. Framework information integration mechanisms

The logical need for tools to communicate and share information with each other within an environment can be met by a framework through the utilization of different mechanisms. Standard *information exchange* services allow data to be interchanged between tools using standard data interchange formats. A common framework *database* allows tools to share data by utilizing common data access interfaces and employing a common description of the data items. A *file system* allows data to be shared, but it does not provide common descriptions of the data. Tools can exchange data by *communicating* with each other.

2.1.1.1. Information Interchange

A framework should support standard information interchange formats for all common data types employed by tools in the environment. This support

should include framework utilities for converting from one format to another, standard framework information interchange services interfaces, and "clipboard" facilities. These clipboard facilities can be used as temporary buffers to allow information presented by one tool to be transferred into another tool using a simple cut and paste operation.

Frameworks must also support the interchange of information with tools in other frameworks. Most of the issues arise from differences in the environment frameworks. These differences can occur in tool interfaces, data model and data management services, communication services, and standards or lack of standards concerning the project data to be transferred between environments.

Issues

1. **Proliferation of Interchange Formats:** In many areas, information interchange standards do not exist. Different framework vendors have chosen to make public an external form of interchange standards to encourage the development of additional tools that can coexist with the vendors' tools. This has resulted in several incompatible external information interchange formats.
2. **Lack of Interchange Interface Standards:** Besides needing standard, external formats, the framework should provide interfaces that allow tools to exchange information. For example, a clipboard provides a temporary buffer for exchanging information between tools.
3. **Information Loss:** Information is lost if some of the data exported by one tool is not received by the importing tool. There are three places where information loss may occur:
   a) The external format cannot encode the information to be exported.
   b) Data is lost in the transmission between tools.
   c) The importing tool is incapable of decoding all the information in the external exchange format.
4. **Reduced Usability:** The usability of the data by the importing tool is largely determined by its ability to understand the data. Even if data is received without information loss, the importing tool may be unable to understand the data it has received. For data exchanged between two tools to be usable by the importing tool, it is necessary to also exchange descriptions of the data as well. An obvious, but not completely satisfactory solution, is to exchange data at a higher level. For example, Ada libraries could be exchanged as source libraries.

5. **Lack of Automation:** The ability to exchange project data between tools is impractical unless the export and import functions are automatable. Manual conversions are not only error-prone, but they are also impractical given the effort involved. Consequently, the framework services that support the export and import of project data are critical.

Approaches

1. **Interchange Format Standards:** The use of standard interchange formats allows data to be interchanged by having tools agree on the format of a common, external representation for a particular type of information. For example, the Programmer's Hierarchical Interface Graphics System (PHIGS) supports the exporting and importing of graphics. There are a number of defacto as well as emerging industry and national standards for document exchange formats. External forms for environment databases are available for exporting and importing data among different instances of the same environment type.
2. **Interchange Format Conversion Utilities:** As different framework vendors provide public interchange formats, utilities will be needed to convert information exported in one format to information in a format that is compatible with the importing tool.
3. **Interchange Services Framework Interfaces:** A framework should provide information interchange services through standard interfaces. These interfaces would allow a tool to construct data that conforms to the interchange standard. The interfaces would also allow data to be imported and exported.
4. **Interchange Clipboards Services:** The Apple Macintosh™ illustrates the power of a simple clipboard mechanism that supports standard graphic and text data. Graphics and text can be cut from any application and pasted into another.

   A framework should provide both private and public clipboard mechanisms. Private clipboards allow data to be cut and pasted under the tool's control. Public clipboards allow data to be transferred between two tools executing simultaneously.
5. **Shared Data Dictionary:** For the importing tool to understand the data it has received, the data definitions that describe the data must be imported as well. This problem is not limited to software engineering environments and is fairly common in the information management domain. A database and data model-independent data description

technology is emerging with the Information Resource Dictionary System (IRDS) standards. If both the exporting and importing environments conform to the IRDS standard, then they should be able to share data descriptions even if their databases are different.

6. **Exchange Deliverables:** A common method for exchanging data between environments is at the level of deliverable work products. The exporting environment is responsible for assembling the deliverable in the form of hard copy documents and magnetic media containing the system source, tests, and binary load modules. The receiving organization must unload the magnetic media and manually place the data into their environment.

Assessment

The ability to import and export information between tools is an important service that must be provided by an environment framework. The level of success that can be achieved in this area varies greatly among frameworks and the types of data being interchanged. The ability to exchange textual data is maturing, while the ability to exchange other kinds of data, including graphics, is often problematic.

There is a rapid evolution by software tool vendors to allow graphic renderings produced by CASE tools to be exported into documents through the use of defacto data interchange formats. The identification and exportation of graphics is often done manually, thus making it difficult to maintain consistency between continually evolving diagrams and documentation of systems which includes these diagrams. Although information interchange is useful, it does not address the problem of maintaining consistency between data managed by different tools. Other information interchange formats which have been considered have gained little acceptance by tool vendors. For example, Descriptive Intermediate Attributed Notation for Ada (DIANA) has yet to become a universal interchange form for Ada programming tools.

The ability to exchange detailed data structures is limited to environments that support identical data models and data description languages. As systems like the Information Resource Dictionary System (IRDS) become widely accepted standards, the ability to exchange data between environments that employ different models will also become possible. The most common form of data exchange between environments is the exchange of deliverable work products.

2.1.1.2. Database Integration

Database integration implies that tools share data and a description of the data. It also provides a uniform mechanism for managing and accessing the data so that the data is readily available, correct, and protected from intentional or unintentional access.

Issues

1. **Data Model:** A software engineering framework database must support data models that are sufficiently expressive to represent the complex data structures that occur in engineering projects. Both the CAIS and PCTE employ variants of the entity-relationship-attribute model. Other environments are experimenting with the object-oriented data models. [WOL88] An evolution towards more powerful data models will allow project data and project knowledge to be maintained within the project database.

   Increased power in the data model requires increased sophistication in the database management system as well as increased computing resources. The primary drawback to employing project databases based on these higher level models is that they are emerging technologies and will require several years to mature.

2. **Data Redundancy:** Storing duplicate data results in system inefficiencies and inconsistencies when a single change needs to be reflected in several places. Data duplication occurs in a software engineering environment for many reasons. A user may want a copy of some data with the intention of modifying the data. If the original data item needs to be updated, it is possible that copies of the data may also need to be updated. A framework should provide mechanisms that support data redundancy.

3. **Data Replication:** Data replication maintains the logical appearance of a single data item with an implementation that can make copies of the data item to improve data availability and survivability. Data replication requires that data locking be handled in a distributed fashion to ensure consistency of data values when updates are made. In addition, updates need to be propagated to all replicated instances of a data item before one is permitted to access an updated value. Supporting data replication reduces network performance and increases system complexity.

4. **Data Consistency:** Data consistency implies that the data values within the database conform to the rules about their values. Transaction

mechanisms are used to take a database from one globally consistent state to another. In an engineering database, this definition of consistency is not entirely appropriate. A typical engineering database may not achieve a consistent state for weeks or months, and in some cases may never reach a globally consistent state. The database should allow the user to abort an incomplete transaction and undo or redo completed ones as a way of backtracking under the user's control.

5. **Data Integrity:** Integrity specifications describe the rules under which a data item value is well formed. Integrity checking includes conformance to data typing or dynamic triggering of constraint checking routines. Each of these mechanisms introduces a performance overhead for making changes to framework data.

6. **Security:** Depending on the project requirements, the framework security model may need to support user authentication, discretionary and mandatory access control, and encryption. Security mechanisms become increasingly more complicated as the data model becomes richer. In addition, not all data within a project will require the same level of access control. All of these mechanism impose an overhead. However, if higher levels of security are required, then these mechanism need to be considered. To date, most of these mechanisms have yet to be applied to software development frameworks.

7. **Distribution:** Distribution is the ability of a network node to maintain global knowledge of the total network in order to make local decisions. The ability to maintain global knowledge requires has significant network communication overhead and delay. Data distribution includes the allocation of data to nodes in a network, location transparent access to data, replication of data to improve availability and survivability, decomposition of database queries into subqueries that can be sent to different nodes, management of locks and names on a network wide basis, check pointing, recovery, and initializing of nodes after a failure. These mechanisms introduce complexity and have significant performance implications.

8. **Recovery:** Recovery is the activity of ensuring that the framework can restore itself to some previous consistent state after a failure. An environment framework must provide tools and procedures to automate system recovery after a failure. Manual recovery procedures will be inadequate as the data complexity managed by the framework increases.

9. **Archiving/Restoring Data:** Archiving and restoring data is a difficult problem when the framework supports higher level data models. It becomes more difficult to determine the set of objects and their

relationships that need to be saved. Furthermore, the restoration process must insert both objects and relationships into an existing structure. Both of these problems are not fully understood at this time.

10. **Concurrency Control:** Concurrency control is the activity of coordinating the actions of processes that concurrently access shared data and could potentially interfere with each other. The simple models of concurrency control employed in file based systems may prove to be totally inadequate when higher level data models are employed. When a large number of objects are accessed by an operation, there can be a substantial performance overhead in implementing concurrency control (especially in a distributed environment) and knowing exactly what must be locked.

11. **Database Migration:** An environment framework must support dynamic changes to the data definition describing the structure of entities, relationships, and behavior of data without taking the system off-line. A change to a data definition can entail changes in storage allocation for entities, adjusting existing data values and generating new ones, and restructuring relationships among objects within the database. The data management facilities within the framework must provide tools to support these activities and ensure that the data migration activities result in a consistent database.

12. **Engineering Data:** Engineering applications require that the database supports the manipulation of complex objects and the need to store graphic, binary, and textual data of varying sizes. Most existing commercial databases support a limited set of data types and do not handle large and varying size data types.

13. **Versioning:** Engineering data evolves by successively refining information about a system. The database must support versioning so that different temporal states of an object can be referenced to recreated an earlier state of the information about a system. One issue with versioning involves deciding which versioning model is appropriate. The strategy of immutable objects is gaining acceptance because it eliminates some difficult problems of maintaining database consistency. The notion of immutable objects is that once an object is committed, its values are time invariant. The versioning model is intimately related to the database concurrency control and transaction mechanisms.

Approaches

1. **Common Database Interface Set:** A strategy for information integration involves the use of a common set of data management services interfaces. This approach is widely recognized as an effective means for integrating tools within an environment. A critical factor for the success of this approach is the adoption of a standard interface set and its acceptance by software tool vendors. CAIS and the PCTE are examples of frameworks that provide a set of interfaces for data management services.

2. **Common Data Dictionary and Multiple Databases:** The level of data integration can be enhanced by using a common data dictionary to provide a common understanding of the structure of the data, even though the data itself may be stored in physically distinct databases. It is common practice to develop data processing applications that access multiple databases through the use of a common data dictionary. The data dictionary becomes a single point of reference for all the project data definitions. A good example of this approach is the Information Resource Dictionary System (IRDS) being developed by the National Institute of Science and Technology. The IRDS supports the design of efficient programs and databases that share data. The IRDS uses a meta-data representation based on the Entity-Relationship-Attribute data model.

3. **External Links Mechanism:** Current environment framework databases do not provide adequate performance for kinds of data managed by CASE tools (e.g. intermediate forms). Consequently, most CASE vendors use a highly optimized and proprietary, application-specific data management system to support their tools. One problem with integrating these tools with other tools is the inability to establish traceability between information maintained by different systems. One strategy for dealing with this problem is to provide an external database that has a simple tool-independent interface for registering data dependencies. The SUN NSE provides a linking service that allows tools which conform to the link interface protocols to establish relationships between data stored in their respective databases without having to directly access each others' databases.

Assessment

Information integration is critical for providing effective automation within an environment. Current framework standards efforts and emerging commercial frameworks are addressing the issue of providing standard

database interfaces. The PCTE is the most advanced commercially available framework that supports a distributed entity-relationship project database. At the current level of PCTE funding and the high level of industry participation, the PCTE will continue to evolve and mature rapidly. The PCTE is commercially available on most of the common computer platforms.

The PCTE standard, used by vendors to design applications, comes in several variations, PCTE 1.5, PCTE + and ECMA PCTE:

- PCTE version 1.5 is partially implemented as a commercial product called Emeraude.
- PCTE + specifies the future features of PCTE. The current thrust provides a "neutral" operating system and security options. As of July 1991, two development teams are building implementations of PCTE+, but no products are commercially available.
- The ECMA PCTE has made minor modifications to PCTE+. Currently, its standards group is soliciting comments on the modified standard. Digital and IBM are developing a commercial products, but ECMA PCTE has no product announcements as of July 1991.

2.1.1.3. File System Integration

Overview

Host operating systems files are still the least common denominator for interchanging data between tools. For example, in the UNIX operating system, tools communicate with each other through ASCII text files.

Issues

1. **Meaning Embedded in Tools:** A major disadvantage of using files as the medium of integration is that the meaning (i.e., structure and interpretation) of data is encoded in tools. This creates a critical dependency on the specific, physical implementation details of the data stored in the files.
2. **Multiplicity of File Systems:** Tools that use a host file system will inevitably be dependent on how the host files are named and accessed.

Even though a good design and the use of a standard language such as Ada will reduce the impact of porting tools, there are a number of differences between file systems on different hosts that reduce the effectiveness of using files as a vehicle for inter-tool communication.

3. **Database:** File systems, in general, do not address the issues identified in the discussion concerning Data Base Integration.

Approaches

1. **Hierarchical File System:** Hierarchical files systems are the most common means for managing data within current software development environments.
2. **Network File System (NFS):** NFS provides a location transparent facility for accessing files in a computer network. Tools that were designed to utilize the local file system can run without change and access files located at different computers within the network, provided that the network is using NFS.

Assessment

Hierarchical file systems are still the dominant mechanism for sharing data between tools. This is partly due to the absence of efficient engineering databases that can be used to manage fine granularity data. As standards become accepted by tool vendors, there will be a slow migration of tools towards these standards.

2.1.1.4. Data Communication Integration

Data communication is a dominant means of integrating data processing applications. In the data processing community, it is common practice to have applications share databases across a network, communicate with different terminal types, and share files across a network. This data communication technology has not matured to the same level within software engineering environment frameworks. Peer-to-peer communication protocols, file transfers programs, data management systems, electronic mail, and computer terminals emulation are examples of data communication.

To achieve communication between entities in a computer network, these entities must adhere to the same protocol. A protocol includes the syntax of the information exchange; the semantics of addressing, routing, and correcting the data; and the communication bandwidth and sequencing of units of data interchanged.

Issues

1. **Plethora of Communication Protocols:** A large installed base of computer hardware and software uses existing protocols. The data processing community uses industry standards such as SNA and DECNET. The DoD requires the use of TCP/IP. The international community and the standards bodies are evolving OSI into a future standard communications protocol.
2. **Standards Still Evolving:** The OSI ISO/ANSI standardization has managed to define most of the lower five levels of the ISO protocol stack. The remaining two upper layers are still under development. Unfortunately, the upper layers will have the greatest impact on environment frameworks.

Approaches

1. **TCP/IP:** A series of DoD military standards have been developed to allow communication between dissimilar computer hosts supplied by different computer vendors. These standards include Internet Protocol (IP) MIL-STD-1777, Transmission Control Protocol (TCP) MIL-STD-1778, File Transfer Protocol (FTP) MIL-STD-1780, Simple Mail Transfer Protocol (SMTP) MIL-STD-1781, and TELNET Protocol MIL-STD-1782. The DoD is committed to migrate from DoD protocol standards to international standards. This process will be slow because of the large installed computer and software base within the DoD.
2. **OSI:** The OSI reference model defines a seven-layer communication architecture called the Open Systems Interconnection (OSI) model. This architecture provides for a precise definition of the functionality at the interface to each layer. Each higher layer hides details managed by lower layers providing a suitable abstraction for communication at that level. As OSI comes into widespread use, there will be an increasing migration to efficient hardware/firmware implementation of the OSI protocol stacks providing for highly reliable and efficient inter-computer communications.
3. **DECNET:** DECNET, which is developed by Digital Equipment Corporation, is closely tracking the OSI standard and modifying DECNET to comply with OSI.
4. **SNA:** SNA is a public communication protocol developed by IBM primarily for the data processing industry. Although SNA is a dominant force in the data processing community, SNA does not play a significant role in supporting software development environments, except for terminal communication services.

5. **Mailboxes:** Mailboxes provide an asynchronous communication vehicle among a number of processes by which one or more processes can write to and read from a shared mailbox. Mailboxes tend to be less efficient and are typically not used when high bandwidth communications are required.
6. **Sockets:** A Socket is a named end point of communication between processes on the same or different machines in a Unix host network. A common use of sockets is to implement a server client model of communication between applications within a workstation network.
7. **Pipes:** A pipe is a Unix facility for redirecting text I/O. Pipes support tool composition, provided the tools operate on text files. It is the user's responsibility to ensure that the specified composition is meaningful.

Assessment

The major data communication protocols include IBM's SNA, DEC's DECNET, and OSI, which is an ISO/ANSI standard. Several computer vendors are moving towards the OSI standard. The major standard in Europe, OSI is gaining more interest in the United States.

TCP/IP is the major standard employed by the U.S. government and is supported by most computer vendors. TCP/IP will eventually be replaced by OSI. But until then, TCP/IP will coexist with OSI and other communication protocols. Consequently, an environment framework must support TCP/IP as well as other major protocols used in a particular organization.

Data communication issues become more important with geographically distributed development organizations. More advanced protocols will be needed as we move away from file-based environments to database based environments. In the latter case, file transfer is an inadequate means for communicating between environments or remote hosts within a common environment. More advanced protocols will be required to access remote environment databases and to perform transactions that involve multiple databases.

2.1.1.5. Levels of Information Integration

Information integration is the degree of interoperability that can be achieved between tools within an environment. Interoperability is defined as the ability of tools to exchange their data in a form usable by other tools without conversion. In order to effectively use this data, tools must share an understanding of both the structure and the meaning of the data. The

different levels of information integration that apply to both databases and data communications as the enabling mechanisms are the following:

1. Framework maintains the structure and meaning of data
2. Framework maintains the structure of data and tools encode the meaning
3. Tools encode the structure and meaning of data

The level of information integration supported by a framework is dependent on its support for maintaining a description about the structure and meaning of the data to be shared by tools. Conversely, the degree to which tools are integratable into the framework depends on whether or not they use the framework's descriptions of the data. Both of these issues are dependent on standards for describing the structure and meaning of data and on interface standards for accessing the data.

1. **Framework maintains the structure and meaning of data:** The highest level of framework support for information integration is achieved when the framework maintains a complete description of the structure and meaning (behavior) of the data. Presently, there are no commercial frameworks that provide this capability. The difficult problem is the inability to maintain a description of the meaning of the data. Object-oriented databases provide a facility for defining both the structure and behavior of objects, which is necessary for providing a complete application-independent description of the data.
2. **Framework maintains the structure of data and tools encode the meaning**: The second level of framework support for information integration is to share the responsibility for maintaining the description of the data with tools. Using this approach, the framework maintains a description of the data's structure while tools encode the algorithms that define the data's behavior. Several advanced environment frameworks, such as PCTE, Atherton Software Backplane, and SLCSE, support this level of interface. A limitation of this approach is that as the environment becomes more complex, it becomes more difficult to manage the data's behavior since tools define the behavior.
3. **Tools encode the structure and meaning of data:** At the lowest level, the framework provides minimal support for information integration. In this case, tools encode both the structure and meaning of the data. This approach is the dominant one employed today in frameworks where files are the primary data storage mechanisms.

The three approaches to information integration depict an evolution of environment frameworks to support higher levels of information integration. Tools become simpler as the framework becomes more expressive. Consequently, tools will become easier to integrate.

2.1.2. Control Integration

Control integration allows the operation of tools to be controlled uniformly from within the environment. Control implies the ability to initiate and terminate the execution of processes and to synchronize their execution.

In recent years, information integration has been emphasized at the expense of control integration within an environment. Recent work at Brown University has demonstrated that control integration can rival information integration in effectiveness measured by the degree of cooperation that can be achieved between tools. [RIE88] Control integration will become a more important consideration as more environment capabilities require the computing resources of several computers. The Domain Software Engineering Environment (DSEE) has already demonstrated a substantial reduction in the time required to compile large Ada programs by distributing the recompilation to workstations within the network. With high bandwidth reliable network communications, it is becoming attractive to distribute the computing load within the network.

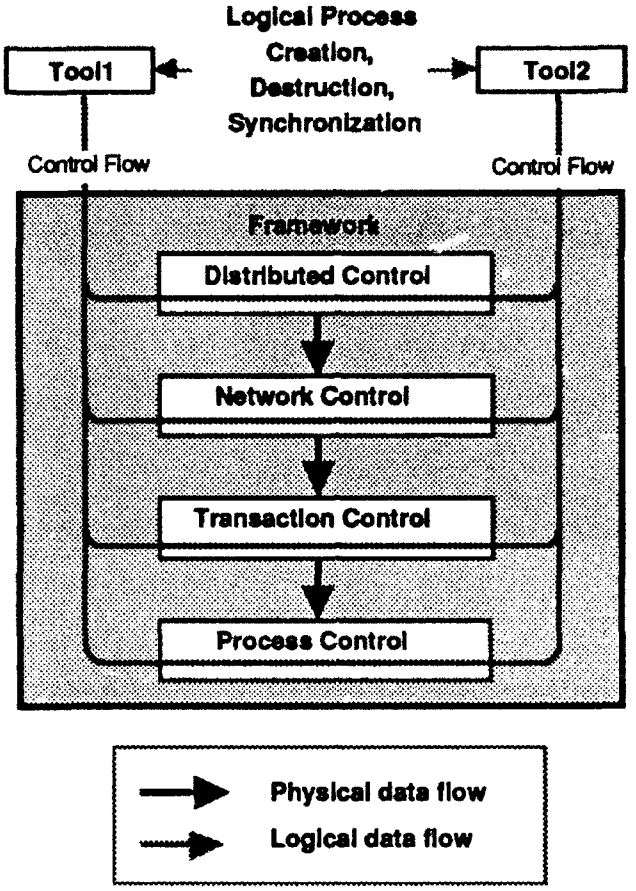


Figure 2.1.2-1. Control information integration mechanisms

2.1.2.1 Process Control

All operating systems provide a level of process management that allows one to create, destroy, and synchronize processes. Different systems provide process models that differ in what rights are inherited by child processes, the semantics of performing I/O, and the mechanisms for interprocess communication and synchronization.

Issues

1. **Interaction with other Mechanisms:** In considering framework process models, the interaction of the process model with other environment mechanisms such as security, transactions, naming, distribution, and inter-process communications must be examined.

Several problems have arisen from the interaction of different models. For example, the PCTE binds processes to transactions. Tools designed in Ada that use tasks for independent subfunctions cannot bind transactions to their natural unit of execution (i.e., task). Consequently, the ability of tool developers to use Ada tasking in their tools must be limited. This problem is being considered for future revisions to the PCTE process model.

2. **Adequacy of the Process Model**: With networked and distributed computing, tools can be decomposed into cooperating processes that run on different platforms. In doing so, several process model implications must be considered:
   - The context inherited by child processes includes visibility to framework objects; access rights; the ability to suspend, resume, and abort the execution of other processes; setting the priority of processes; and visibility into the parent process's status and state.
   - Inter-process communication and synchronization mechanisms should allow users to construct reliable and efficient networked applications. A high degree of location transparency is desirable.
   - I/O blocking causes problems when multiple Ada tasks are timesliced within one process.
   - Lightweight processes allow tools to utilize a large number of processes without incurring the large overhead for process context switching.

Approaches

1. **Traditional Process Control:** Most common operating systems used today support a simple process hierarchy in which child process inherit the context of parent processes. There usually is a limit to the number of processes that can be created. There is a more practical performance limit on the number of processes that execute concurrently. Both limitations are caused by context switching overhead.
2. **CAIS Process Control:** CAIS processes are objects whose attributes contain process status information and whose relationships establish the context of the process relative to other processes and objects within the environment. The existence of process nodes is independent of whether they are running, suspended, or aborted.
3. **PCTE Process Control:** PCTE processes are not modeled as objects. Consequently, the PCTE provides a separate set of interfaces to manage processes. This approach is difficult when writing a generic tool to

uniformly access a wide variety of objects managed within the framework. Another problem occurs when a process is terminated: all information about the process is lost. PCTE+ has proposed a change to the PCTE model to make processes first-class objects.

Assessment

Framework process models and capabilities will need to evolve considerably from the time-sharing heritage of most current operating systems. These limitations must be addressed to support multi-processor, parallel computing architectures of the future. The next generation of CASE tools will exploit processes to provide a high degree of parallelism required for analysis-intensive tools. An evaluation of environment frameworks should address these needs to ensure that they are feasible through a natural migration path of the framework.

Advanced frameworks model processes as objects that can be created, deleted, and accessed as objects. The advantage to this approach is a significant simplification of tools that manipulate processes. Another advantage is that objects, like processes, can be distributed and replicated within the framework to exploit multi-processor and parallel-processor architectures.

2.1.2.2. Transaction Control

Transaction control supports data integrity in the presence of hardware or software failures. Transactions treat a sequence of operations as an atomic operation such that they are all completed successfully. If any one of the operations fails, then the effect is as though none of the transactions were performed. Transactions are required to preserve integrity within engineering databases.

Issues

1. **Long Duration Transactions:** Transactions may last for hours, days, and weeks, or in some cases, they may never even be completed. Engineering transaction mechanisms must provide considerable flexibility to users and tools over their behavior.
2. **Nested Transactions**: Nested transactions are a natural consequence of how engineering activities are structured. Nested transactions is a new technology in which most engineers have very little experience.
3. **Interaction of Versioning with Transactions**: Versioning and concurrency control interact strongly with transactions. Consequently, most researchers now advocate the concept of immutable objects. An object is said to be immutable if its state cannot be changed after its

values are committed. Approaches that allow object values to change dynamically have problems with concurrency control and place inconvenient restrictions on the user's ability to access objects.

4. **Transaction Models:** There are many transaction models that describe how processes, operations, and objects are bound to a transaction. More experience is needed to evaluate these models.

5. **Convenience:** The transaction mechanism should not hinder development. For example, a user should not lose results as a result of a mistake or failure in a particular tool. The environment framework should checkpoint the database periodically. In addition, the framework database should allow the user to abort incomplete transactions and undo or redo completed transactions as a form of backtracking under the user's control.

Approaches

1. **Pessimistic Transaction Model:** The pessimistic transaction model assumes that multiple users needing to access a common object must always lock the object before accessing it. Traditional data processing databases employ the pessimistic transaction model. The pessimistic model is inappropriate for engineering applications because of the long duration of these transactions and the problems that may be caused by locking all users from reading an object that is participating in a transaction.

2. **Optimistic Transaction Model:** The optimistic transaction model assumes that users seldom, if ever, need to access and update a common object. Consequently, locks are not placed on the object during the transaction. During the commit, conflicts are resolved manually by users deciding which conflicting transactions need to be aborted. Since these conflicts rarely happen, this is an insignificant problem.

3. **SLCSE Transaction Model:** SLCSE provides two kinds of transaction bindings. The first approach allows a user to request an exclusive lock on a given entity or relationship type, after which all instances of the object or relationship type participate in the transaction. The second alternative is similar to the traditional, single level, pessimistic transaction mechanism in which all instances that are touched during the transaction participate in the transaction.

4. **PCTE Transaction Model:** PCTE (version 1.4) processes are initiated in the context of an activity (i.e., transaction). Acquired resources and locks held on these resources are associated with an activity. An activity can be initiated internally to a single process or extended to

include several generations of processes. Consequently, the PCTE model binds processes to activities. Ada tools that use tasking to perform separate functions that should behave as transactions would function improperly if the Ada run-time manages tasks within a PCTE process.

Assessment

A framework must support transactions to ensure the integrity of the data. This is becoming increasingly important as the complexity of the data that is managed within development projects increases. Traditional transaction mechanisms are too restrictive. Several transaction support functions should be provided by an environment framework:

- optimistic transaction model to impose the least amount of restrictions during development
- pessimistic model for critical updates
- multi-level undo and redo and returning to baselines
- transaction binding model is compatible to tools that use Ada tasking
- nested transactions
- tool interfaces for transaction management

Transaction support becomes increasingly important as more tools are integrated into an environment. The failure of any tool in a long automated process can leave the database in a corrupted state. Consequently, it is necessary to protect the data by being able to undo the effect of the execution of these tools.

2.1.2.3. Network Control

Overview

Modern frameworks provide services that support a higher level of control integration to integrate tools by directing their execution remotely. The framework supports this integration by maintaining a dictionary that identifies the tools and their interface protocols. A designated tool is usually in control and will direct the execution of other tools. The primary distinction between this higher level control facility and lower level process controls is that the framework maintains information essential to performing the integration.

Issues

1. **Lack of Control Standards:** Network control mechanisms (e.g., remote procedure calls) have been used for a number of years in experimental and production systems. Until recently, there has not been much interest in standardizing these control mechanisms.

2. **Limitation of Current Development Languages & Tools:** In order to fully utilize the power of control capabilities, it is necessary to employ languages and tools that support the control paradigm.

Approaches

1. **Apollo Network Computing Architecture (NCA):** The Apollo Network Computing Architecture (NCA) employs a set of tools that supports an object-oriented approach to developing distributed applications. NCA is based on a model of clients, objects, and brokers (or agents). A client requests a service from the network, and servers provide these services. The role of the broker is to enable the transaction between the client and server.

2. **Remote Procedure Calls (RPC):** The remote procedure call (RPC) facility extends the local procedure call mechanism to a distributed computing environment. With the RPC mechanism, application code can be distributed among multiple processors in a highly transparent manner. An RPC run-time library is used to route the messages that implement the RPC. An RPC facility has several requirements:
   - The RPC facility must be portable to multiple hosts.
   - NCA must be placed in the public domain and recommended for standardization.
   - The run-time facility should be implementable on top of the common protocols.
   - Automatic error checking and recovery should be provided by the run-time system.
   - The run-time library should minimize the message traffic.
   - Tools should be available to all the popular programming languages to support the generation of both Client and Server stubs.
   - No new language constructs should be required by NCA applications.

- The system must provide mechanisms for automatic data translation between unlike machines.
- The architecture should support light-weight processes (processes with minimal context switching overhead) for massively parallel architectures.

Remote graphics, remote debuggers, network batch and queuing services, and remote database servers are examples of network computing applications.

3. **Selective Message Broadcasting:** In selective broadcasting, all tools talk to a central message server, which then broadcasts messages to active tools known to the message server. Tools communicate by sending messages to the server and receiving those messages that match their registration with the server. This simple facility can provide a high level of integration between tools.

The Selective Message Broadcasting approach has been used in the FIELD programming environment developed at Brown University. [REI88] This approach allows existing tools to be integrated effectively with minimal effort. This approach can easily be extended to incorporate new tools. In FIELD, messages can be sent either asynchronously or synchronously. In addition, messages can be used to coordinate and direct the execution of different tools and shared information among tools. For example, if a user is working at an intelligent editor and wants to set a breakpoint in the source, the editor must be capable of sending a message to the debugger asking it to set the specified break point. If the user wants to find all occurrences of a specific variable, the editor can send a message to the cross-referencing utility. Finally, when the user want to compile the source, the editor can send a message to the "make" tool to compile and link the appropriate program components.

The selective message server approach allows tools to access different databases by providing an active database server that can respond to requests for information from other tools. This approach compartmentalizes the environment data, simplifying both the design of data servers and tools that use the data. A drawback of this approach is that it does not solve the environment data integrity problem in which responsibility for project data integrity is dispersed among different databases.

A tool is integrated by building a shell that communicates with the message server around the tool. The effort to develop the shell is partially determined by the nature of the tools' command interface. Tools that have a consistent grammar for these commands would be easier to integrate than tools which use an ad-hoc command notation. The problem of building these shells is

equivalent to being able to define command patterns and rules for recognizing these patterns.

The FIELD message server uses the Unix TCP-domain sockets. These sockets allow messages to be managed across the network, allowing tool operation to be integrated across multiple machines in the network.

Assessment

Network Computing Architecture and Selective Message Broadcasting are similar approaches that provide an object-oriented paradigm for integrating environment functionality. NCA allows tool developers to build new applications that better utilize network resources and support improved communication and cooperation between tools on different host. Selective Message Broadcasting provides a high degree of integration of tools that do not share data directly. A principle advantage of this technique is that it can be used to integrate existing tools that provide a reasonable, structured command interface.

2.1.2.4. Distributed Control

Overview

The major distinction between network control and distributed control is subtle, but fairly significant. Network control provides location transparency. Distributed control provides location transparency as well as replicated control functionality at nodes of the network where each node maintains global control. The information at the global control level enable local control decisions that are in the best interest of the network.

Issues

1. **Technology Maturity:** The distributed control technology is still a research topic whose maturity is a number of years away.
2. **Technology Complexity:** Distributed control requires that each node in the network makes decisions based on global knowledge of the network. In order to maintain this global knowledge, the nodes must continuously inform each other of changes and attempt to maintain a high degree of consistency of this knowledge.

Approaches

At this time, distributed control is impractical for environment frameworks. There is a significant performance overhead for maintaining consistent global control information at each node. In addition, there are a number of

problems such as node re-initialization, network recovery, single node recovery and continued operation after network failure. These are a sample set of problems that add significant complexity to systems implementing distributed control. Because of the significant increase in complexity and high cost of development, distributed control is not supported within most current frameworks.

Assessment

It is unclear if a distributed solution warrants increased complexity and cost. With increasing network bandwidth, conventional logging and error recovery techniques, and network computing, the required access to data and overall system performance can be achieved without the expense and complexity of distributed control.

2.1.2.5. Levels of Control Integration

Control integration deals with the ability to manage the execution of tools by the framework so that tool functionality can be composed to define new and more complex functions. Control integration includes the management of the creation, destruction, and synchronization of processes within the framework. The levels of control integration that can be provided by a framework are the following:

1. Framework supports control and data flow description and execution
2. Framework supports control flow description and execution
3. No control integration

The level of control integration supported by a framework depends the framework's ability to manage resources across the network, its ability to register tools with the framework, and its ability to define a model of dependencies between work products and tool executions. Control integration is an enabling capability for automating software development.

1. **Framework supports control and data flow description and execution**: Advanced frameworks can schedule the executions of tools to produce the required software products. These frameworks support the definition of tool mappings of input products to output products and schedule the executions of tools as needed to generate required work products. These frameworks ensure that generated products are consistent with their dependent products and schedule the regeneration of products either on demand or as background processes. DSEE provides this capability for configuration management and for building the deliverables.

2. **Framework supports control flow description and execution**: At this level of control integration, the framework allows users to define a sequence of tool executions that can be carried out automatically by the framework. This form of control integration is supported by most operating systems shells.
3. **No control integration**: A framework at this level provides limited, if any, capabilities for control integration.

An important aspect of control integration is the ability to manage resources, including the execution of processes within a computer network. In order to support more advanced CASE technologies, the computing and information resources of the entire network should be exploited. As network-based frameworks become commonplace, the ability to effectively manage the distribution of processes and data within the network will be critical to making effective use of the network. Many aspects of managing a software project on a single computer host become increasingly complex with networked-based, distributed computing.

2.1.3. User Interface Integration

2.1.3.1 Tool Integration

In the last few years, there has been an increased emphasis on the environment user interface. This emphasis results from advances in workstation technology and improving price/performance ratios to where increasing number of companies are acquiring workstations for software development. Software developers are now aware of graphic, object-oriented user interfaces and view products that do not provide such an interface as antiquated and ineffective. Consequently, computer and software vendors have emphasized the development of graphic user interfaces for their products resulting in a proliferation of user interfaces. Consequently, each computer system and tool has its own style of user interface. The use of menus, mouse buttons, icons, scrollbars, and function keys varies greatly between frameworks and tools.

Good user interfaces are difficult to design and implement. For example, the Macintosh user interface represents a large effort involving human factors experts, software designers, and people who had developed and used similar user interfaces at Xerox Parc. The Macintosh has been successful in achieving widespread acceptance of its interface as a point of comparison for other user interfaces. The Macintosh user interface is successful for several reasons:

1. Standard software interfaces implemented in a highly optimized ROM toolbox. Anyone bypassing the toolbox will encounter significant development effort and performance penalties.
2. Apple has published standards for using the graphics toolbox and seeded the market with programs that illustrated these standards.
3. The Macintosh user community is not tolerant of new products that do not conform to the Macintosh user interface standards.

An increasingly important environment framework role is the support for and enforcement of a uniform, well engineered user interface. Computer vendors recognize this need and are starting to promote their own proprietary solutions.

Issues

1. **Lack of Commonality across Tools:** It is common for different tools to use mouse buttons, menus, and icons in ways which are incompatible with each other.
2. **Standards at Too Low a Level:** Current user interface standards, such as X-Windows, are at too low a level to influence the look and feel of tools within an environment. Higher level standards are needed for tool developers to ensure a consistent look and feel among tools.
3. **Human Factors:** Simply using graphics, menus, and icons does not guarantee a good user interface. The most important ingredient is the use of a sound model of human-machine interaction that is reflected in the user interface implementation by tools. A number of tools are extremely difficult to use even though they employ all of the accepted user interface mechanisms. For example, one of the popular document publishing systems available on most workstations has a Macintosh-like interface. Unfortunately, simple operations, such as cutting and pasting a piece of text, may require as many as 27 actions by the user.

   The design of good user interfaces requires a study of the work flow patterns of system users. The design also requires a consistent model of interaction so that the user can intuitively perform a new action based on past experience using the tool.
4. **Performance:** Developing an efficient set of higher level user interface libraries is a major undertaking. A number of graphics-based tools available on high powered workstations suffer from poor performance when manipulating complex graphics. There is a critical need for

higher level user interface libraries that have been engineered for performance.

5. **Complexity:** Advanced user interface libraries are complex. This complexity arises from several factors:
   - support simultaneous update within multiple windows
   - intercept random user events in different contexts and dispatch these events to the appropriate application event handlers
   - provide a generic architecture for doing, undoing, and redoing user commands
   - allow arbitrary nesting of windows, panes, and subpanes; each with its own coordinate space and associated controls
   - provide the ability to perform common operations such as moving and sizing a collection of complex objects
   - provide uniform support for color and shading for all user defined objects

Approaches

1. **Object-Oriented User Interface Libraries:** Conventional programming cannot cope with this complexity. New user interface paradigms based on object-oriented programming show considerable promise in being able to manage this complexity and supporting optimizations for performance. Object-oriented user interface libraries are now becoming available. Future frameworks should incorporate these libraries for use by tool vendors.
2. **Open Look:** Open Look, announced in April 1988 by AT&T, will be the standard user interface for Unix System V (version 4.0). Open Look was designed for AT&T by Sun Microsystems and based on a technology licensed from Xerox. It is designed to be independent of the hardware and software that it runs on. Open Look was designed to accommodate different keyboards, mice, and screen resolutions. In a manner similar to the Macintosh toolbox, Open Look provides an extensive set of higher level user interface toolkit routines for the application developer. A goal of Open Look is to provide consistency between applications so that users can easily switch between applications. For example, throughout the system and across applications, a given mouse button is used for only one function. The first two Open Look toolkits will be available early in 1989. The version from AT&T is called XT+, while Sun Microsystems's version is called View2. Both of these toolkits will be implemented on top of

X-Windows. Sun is also planning to provide an Open Look toolkit called the NeWS Development Environment (NDE), which is based on the NeWS window system.

3. **OSF/Motif:** As part of a movement to establish standards for user interfaces, the Open Software Foundation (OSF), in 1988, asked major software developers to submit graphic user interface technologies for consideration as part of a standard operating environment for UNIX. To most people's surprise, the OSF chose pieces from three companies - DEC, Hewlett-Packard, and Microsoft. OSF's product, Motif, looks like Microsoft's Presentation Manager (PM), uses parts of the DEC and Hewlett-Packard application program interface (as well as the three-dimensional windows from Hewlett-Packard's NewWave), and is based on the X Window System. OSF/Motif, OSF's first offering, is a graphical user interface combining the following elements:

   - A toolkit
   - A presentation description language
   - A window manager
   - A style guide

   Together, these four elements, the toolkit, the presentation description language, the window manager, and the style guide, provide a standard of user interface behavior for applications.

   OSF/Motif runs on the workstations of Hewlett-Packard, Digital Equipment Corporation, Sun Workstation, and Interactive. Like most software products in their early stages, some bugs exist, and are being uncovered by users. OSF/Motif is currently in Version 1.0; Version 1.0.3 was released with some fixes to bugs, with Version 1.1 soon to be distributed.

   Motif is fast becoming an industry de facto standard. Following the announcement of Motif, many companies announced support for the OSF standard and began tweaking their graphic user interface software to be compatible. For example, as of April 1990, over 600 companies have licensed OSF/Motif source code, representing 25 countries and 80% of worldwide computer suppliers. Endorsements include The European Economic Community, '88 Open Consortium, General Motors, American Airlines, and the Marriott Corporation. OSF/Motif is more popular than Open Look, with 73% of software vendors working in UNIX having elected to support OSF/Motif. Presently, Motif runs on more than 100 hardware platforms and 38 operating systems [WAG90].

The U.S. Government's response to OSF/Motif has been very positive as indicated by the acceptance by the U.S. Air Force and U.S. Navy, and OSF/Motif's references in U.S. Government RFPs.

4. **DECwindows:** DECwindows is DEC's user interface toolkit for providing a common user interface across machines based on VMS, Ultrix, and MS-DOS. DECwindows is implemented on top of X-Windows, which should further promote the portability of the user interface. DECwindows is supported by a toolkit called X User Interface Toolkit, which provides a set of tools for application development; a user interface style guide; and bindings for Ada, Pascal, C, and FORTRAN.

5. **SLCSE User Interface:** A design goal of SLCSE was to provide a consistent user interface across a range of terminal types and workstations. The SLCSE user interface is data driven and linked via a message handler to isolate the user interface from changes. For example, the introduction of a new tool into SLCSE is handled by specifying the tool name, its parameters, and the roles that are allowed to use the tool. This approach provides an easy facility for integrating a new tool into the SLCSE framework. SLCSE supports three levels of tool integration with its user interface.

    1. The user interface libraries can be compiled directly into a tool to provide a SLCSE conforming look and feel to the tool. A number of system tools, such as mail, *dir, copy, delete,* conform to the SLCSE interface. A tool writer has the option of developing new tools to adhere to the SLCSE interface.

    2. SLCSE provides a surrogate user interface process that supports a standard user interface protocol for SLCSE tools. Tools that use this protocol are isolated from host and terminal dependencies.

    3. Tools imported into the SLCSE environment are simply invoked by the SLCSE user interface and will display their existing user interface.

Assessment

Providing a consistent look and feel across tools within the environment is becoming increasingly important as end users recognize the benefits of such interfaces. It appears that industry pressure is reducing the number of interfaces to a small set of competing solutions which includes DECwindows, OSF/Motif and Open Look. Both of these systems are highly portable and compatible with modern networked environments. There is probably room for a small set of high level user interface standards. Existing environment

frameworks are expected to evolve to embrace at least one of the industry standard interfaces. When evaluating a framework for its user interface capabilities, several issues must be considered:

1. conformance to the look and feel of an industry standard
2. conformance to industry standard lower level interfaces to support tool portability
3. user interface toolkit that assists in achieving the standard look and feel
4. user interface style guidelines to ensure that tools developed for the framework conform to the required look and feel
5. libraries of user interface components that can be used to rapidly assemble sophisticated user interfaces
6. generic user interface and tool architecture that can be used to rapidly develop and integrate user interfaces into new tools; several object-oriented tool architectures can be used for rapid development of complex user interfaces

2.1.3.2. Levels of User Interface Integration

User interface integration is measured by how well an environment provides a consistent look and feel across all tools within the environment. A high degree of user interface integration has been hard to achieve in situations were tools are developed independent of the environment framework. There has recently been an increased emphasis on the need to standardize the look and feel of the user interface within an environment. Open Look, OSF/Motif, and DECwindows attempt to define a vendor-independent, standard framework interface set as well as conventions for the look and feel of tools built using that standard.

The following are levels of user interface integration:

1. User interface look and feel defined and enforced by framework.
2. Consistency due to adherence to standards
3. Each tool provides its own look and feel

User interface integration has been hard to achieve because of the lack of standards or approach for user interface integration. Within the past few years, the success of the Macintosh has been attributed to having mastered the problem of user interface integration to the extent that the majority of applications share a common look and feel. Similar efforts are now

underway to provide the same level of integration within environment frameworks.

1. **User interface look and feel defined and enforced by framework:** The framework defines a set of interfaces at sufficiently high level that tools that use those interfaces will conform to a common look and feel. Additional guidelines should dictate the appropriate application of these standards.
2. **Consistency due to adherence to guidelines:** Some frameworks have guidelines or a tradition of a certain kind of user interface. For example, Unix tools tend to conform to the Unix style of single letter command parameters.
3. **Each tool provides its own look and feel:** Since there are no standards and guidelines, each tool is free to choose its own look and feel. Unfortunately, this provides an inconsistent user interface that is difficult to master.

To date, frameworks have failed to achieve a consistent look and feel. This failure is partially due to the lack maturity of user interface technologies and the emphasis on low level user interface standards like X-Windows. Effective user interface integration will not occur until frameworks begin to provide higher level user interface libraries and interfaces.

2.1.4 Method Integration

As defined earlier, a framework includes those services that are provided ubiquitously within the environment. Method support capabilities typically fall into this category. When evaluating a framework's support for methods, it is necessary to examine tool integration capabilities and method tailorability and extensibility.

A major problem with many of today's CASE tools is the inability to integrate these tools into an organization's methods. One method that must be considered is the software development life cycle, which has subordinate methods for configuration management, quality management, project management, and development. Each of these subordinate methods extends across the entire life cycle and can encompass a large number of tools.

The degree of method tailorability supported by a framework is important to the acceptability of a framework within an organization. Methods have considerable variation among organizations. Different organizations employ different tools to support their methods. The methods employed by an organization will change over time as their business changes or, in some

cases, as the tools for doing business change. Several methodological issues should be considered when selecting an environment framework.

A number of recent environments provide some level of support for software engineering methods. SLCSE provides the ability to define roles which determine the data objects that a user can access and the tools available. ISTAR, developed by Imperial Sciences of England, supports a method based on the "contract model." The contract model is a formal development method that decomposes a large project into a hierarchy of subcontracts. The ISTAR framework supports this model directly, but allows users to integrate application-specific tools as required.

The field of method modeling has been receiving increasing publicity in the last five years. In the plenary session of the 1987 Software Engineering Conference, the speaker challenged the software engineering community to place a greater emphasis on software engineering method modeling, formalization, and automation. There has been a renewed interest in method research, as illustrated by the number of prototype frameworks with method support presently being developed.

Issues

1. **Method Models:** The technology for modeling and formalizing methods is still a research topic, with some limited application in a few environments such as ISTAR.
2. **Roles:** A number of environments have provided a limited form of support for methods through the use of roles. Roles provide some control over what different users can see and do within an environment based on their responsibilities and authority. The concept of role seems to incorporate dimensions of access control and data view mechanisms.
3. **Tool Integration:** The environment framework must provide tools and interfaces to support methods. Furthermore, development and management tools must be integrated into the framework to become integral parts of the method. This implies that the framework must exercise control over the operation of tools.

Approaches

1. **SLCSE:** SLCSE supports DoD-STD-2167A by identifying all the data that is required for each development phase. Tools populate the database in the course of development. An automatic documentation generation tool can be used to extract information from the database to

produce the required reports. The SLCSE roles restrict users to the data and tools required for a particular DoD-STD-2167A activity.

2. **Atherton Software Backplane:** The Atherton Software Project Softboard provides work flow management capabilities. For example, the action management subsystem allows users to initiate, respond to, and keep track of actions over the life of the project.

Assessment

Method modeling and support is still a research topic. A number of frameworks provide limited method support, such as roles in SLCSE and action tracking in the Atherton Software Backplane. Even these minimal facilities can prove to be useful in tailoring the environment to better support a particular software method. To date, these limited forms of method support do not cause major issues with tool integration.

2.1.4.1 Configuration Management Integration

Overview

Configuration management, which also includes change management, is a development activity that is performed across the life cycle. A configuration management method must support problem tracking, configuration identification and auditing, approval procedures, and tracking and generation of all deliverables associated with a product.

Issues

1. **Automation:** Configuration management is a difficult and complex task that requires automation to be used effectively. This automation involves managing information about developing product configurations, tracking problem reports, and generating deliverables.
2. **Visibility:** Most environment users must access development products that are under configuration management. The framework configuration management services should provide visibility to this information.
3. **Tool Integratability:** A function of a configuration management system is to generate the software products required in a delivery of a system. These products include source, load modules, documents, test data, and software development files. The ability to integrate tools includes the capabilities to invoke and control the operation of tools, handle failures, and distribute construction of a software system for delivery.

4. **Tailorability:** The sequencing of steps employed in performing configuration management and scheduling tools, as well as the kinds of information maintained about configurations, should be tailored to suit the organization.
5. **Extensibility:** The configuration management facilities provided by a framework will require some customization which goes beyond the tailoring just described. Extension is typically handled by modifying or replacing parts of the system. Access to lower level interfaces and the availability of details about the data definitions used by the configuration management tools are two issues which should be considered.

Approaches

1. **Apollo DSEE:** DSEE supports configuration threads and a powerful make facility. DSEE takes advantage of the network and distributes a number of configuration management functions within the network.
2. **SUN NSE:** NSE provides both revisioning and versioning to provide a fine grained control over how changes are managed. Revisions are made for major changes while versions are made for minor changes. NSE provides a collection of tools to support the creation of new revisions and versions. A "make file" facility is available for building deliverables.

Assessment

Configuration management is a mature technology. Configuration management products have evolved for the past twenty years and are now in a state of high refinement.

Performing configuration management functions efficiently within a network requires the ability to distribute these management functions within the network. For example, configurations should not be kept on one node. In larger developments, significant disk contention and network availability problems may arise. For example, if anything goes wrong with the node that has all the configuration data, then work on the project is impacted. Frameworks should be able to distribute the data and the major configuration management functions within the network.

2.1.4.2. Levels of Method Integration

A method is the sequencing of activities that must be carried out in order to produce a desired deliverable. The ability to define or tailor a method implies

that the framework can support the carrying out of activities as defined by the method. In defining a method, the framework must support the registration of tools that participate in the method, identification of work products that are input to and produced by each activity, and people and their roles in the activity. A few proprietary and commercial frameworks [LEH86] support the definition and tailoring of methods. These frameworks have typically been applied to developments with rigorous methodological constraints.

Method integration relies on all the forms of integration just discussed. When defining a method, the work products and necessary tools must be identified. In addition, users must be directed to follow a sequence of activities defined by the method. Consequently, effective method support cannot be performed until the other forms of integration are in place. The two levels of method integration are the following:

1. **Method Definition and Enforcement:** The framework supports the specification of a method and enforces the method during development.
2. **No Method Support:** Most frameworks do not provide method support.

There seems to be a spectrum of method support provided by environments. However, most fall short of providing an effective and flexible level of method integration. In order to achieve effective method integration, the types of integration just described must be in place. Method modeling has become a popular research topic in the last few years. Areas being investigated include knowledge-based method modeling and method programming. Method modeling technology is in its infancy and will need several years of research and practical experience to mature.

2.2 Factors Affecting Tool Integration

This section briefly examines a number of orthogonal factors based on the design and implementation of the framework rather than on the framework technology itself. For example, the style of data management interfaces can affect tool integration independent of the data model supported. Tools that do not conform to the style of usage may be more difficult to integrate.

2.2.1 PCTE PACT Experience

There is now a growing body of experience by developers who have tried to integrate tools into advanced frameworks. For example, the ESPRIT PACT project, which built an integrated toolset on top of the PCTE interfaces, found the need to develop a layer on top of the PCTE interfaces and use these new

services for developing tools. One of the problems the PACT developers found when they started to develop tools using the PCTE was that the PCTE interfaces were at too low a level. Consequently, the PACT project introduced another layer called the PACT Common Services layer. The Common Services layer implemented some of the fundamental models that the PACT project wanted to reflect through all of the tools used within the environment. By factoring out this functionality and embedding it into a Common Services layer, the tools became simpler and it became easier to provide a single, consistent model of user interaction.

Figure 2.2.1-1 illustrates the PACT architecture. The innermost circle represents an implementation of the PCTE interfaces. The next layer, the Common Services layer, provides services common to a number of tools. The Common Services layer eases the tool writer's task by reducing the amount of code to be written. It also provides a higher degree of consistency amongst tools. The PACT project found the Common Services layer to be an important mechanism for tool integration support and recommended that these services be used by all tools. This experience introduces an interesting dilemma in that the level of tool portability is now the Common Services layer and not the PCTE layer. In order to transport tools, it is also necessary to transport the Common Services Layer. Furthermore, it is unclear if tools built to the PCTE interfaces could be integrated

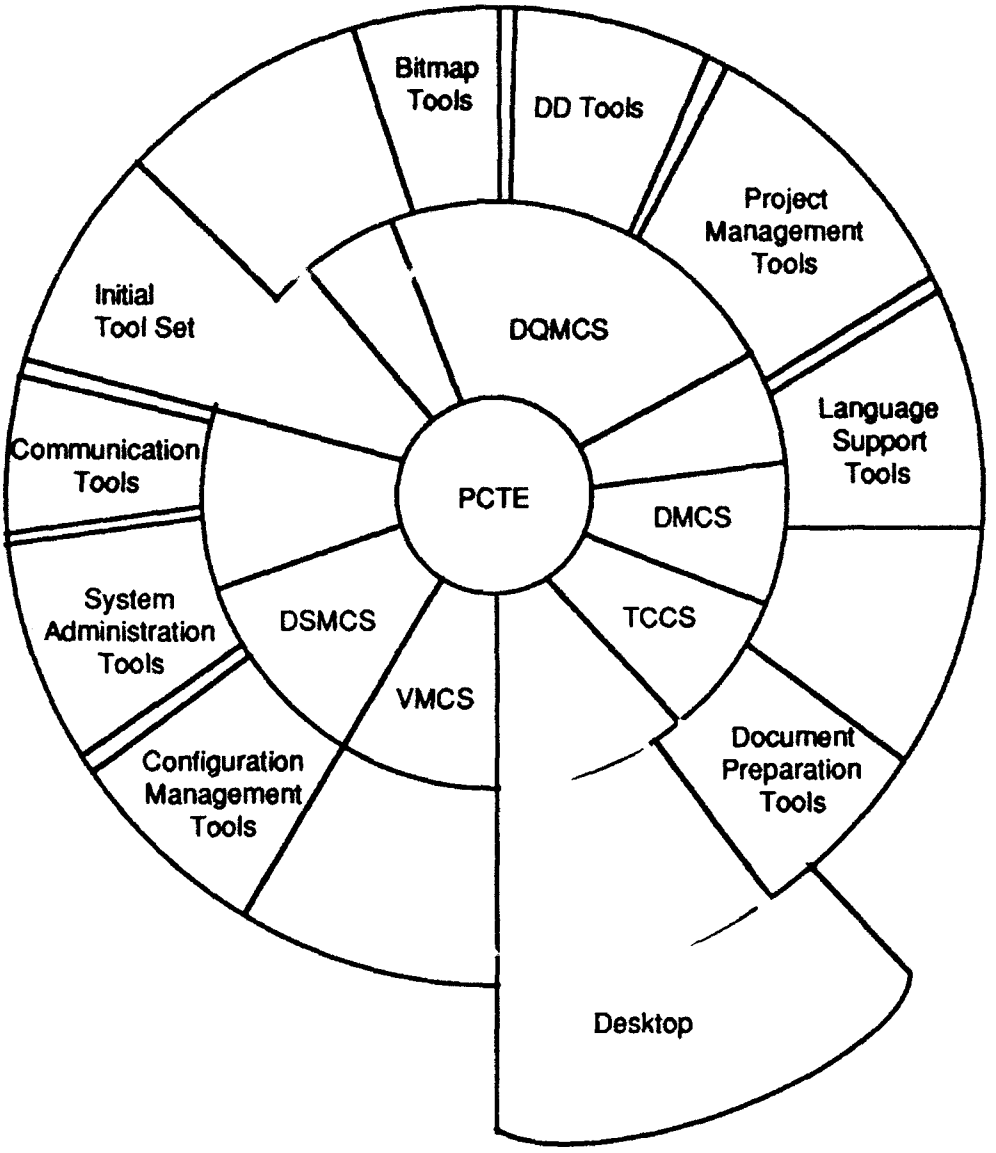


Figure 2.2.1-1. PACT Architecture

with tools developed on top of the Common Services layer. The PACT Common Services include the following:

1. **Dialogue Management Common Services (DMCS):** The DMCS supports a higher level model of man-machine interaction than that provided by the PCTE interfaces. Associated with the DMCS is a set of guidelines for those aspects of the PACT user interface that are not directly supported by the DMCS.

2. **Data Query and Manipulation Common Service (DQMCS):** The DQMCS provides a high level query processing capability on top of the PCTE primitive Object Management services. Clearly, tools that employ these services will require the presence of the DQMCS on each PCTE instance on which those tool run. The results of queries are relationship tables. The PACT experience has shown that these relationship tables provide a useful exchange format between tools.
3. **Version Management Common Service (VMCS):** The VMCS provides basic facilities for the management of versions of single and groups of related objects.
4. **Document Structure Common Services (DSCS):** The DSCS provides facilities for tools that create, access, and manage documents in the PCTE object base.
5. **Tool Composition Common Services (TCCS):** The TCCS provides facilities for defining new tool functionality by composing primitive tools.

The outermost circle represents the application specific tools that populate the environment to support the project disciplines. The Desktop is the PACT user interface that includes a graphics interface for data query and manipulation.

2.2.2. Framework Architecture

The framework reference model defines a tool support layer that provides the functionality and interfaces required for tool development. The architecture must explicitly address the problems of integrating tools. Simply providing a set of low level standard interfaces is insufficient. In general, tools interact at a much higher level. For example, the PACT Common Services layer provides a set of services that are more appropriate for the tool writer than the PCTE standard interfaces. This experience points out that many of today's thrusts of defining low level framework interfaces is an inadequate solution to tool transportability and integratability.

It seems inevitable that tools will make some assumptions about higher level services. An architecture that also defines interfaces for services at that level will enhance the ability to effectively integrate tools. The argument for this is that as the framework services increase in power, the environment tools become simpler. Simple tools are typically easier to integrate than complex tools. In addition, the more that tools rely on the framework services, the easier it is to integrate these tools. The experience gained from PACT confirms this premise.

2.2.3. Open Architecture

An open architecture includes both a published detailed description of all the framework interfaces as well as a philosophy of openness. The latter is perhaps the most critical in that it determines the nature of these interfaces. The converse of this is also true for tool developers. Tool developers also need to develop tools with a philosophy of openness to make it easier to integrate tools into an environment.

Frameworks that embody the philosophy of openness should minimize the need for tools to become dependent on particular framework services. For example, if a framework provides configuration management services, its interfaces should minimize the tool's dependence on the particular model of configuration management supported. A tool should be directed to perform its function on framework designated objects rather than requiring the tool to provide a configuration management model with specific information in order to access the object.

The problems of tool integration will not be solved by low level interface standards such as those defined by the CAIS-A and the PCTE. Instead, these problems will be solved by having framework and tool vendors adopt a philosophy of openness at higher levels of framework functionality. Areas where this openness should be considered include configuration management, user interface paradigms, documentation management, and database querying.

2.2.4 Conformance to Standards

Conformance to national, international, DoD, and industry defacto standards will aid in tool integration. Many of the current standards are at too low a level. Although one can argue that low level standards provide the most flexibility and are an aid to tool transportability, they really do not adequately address problems of tool integration. Tools that are developed using higher level framework services would no longer be portable unless those services where also ported.

Most current framework standards are evolving around the entity-relationship data model which some believe is too weak. Several prototype environment efforts are experimenting with object-oriented environment frameworks. [VIN88] [WOL88] Object-oriented frameworks will eventually replace the current entity-relationship based models.

2.2.5. Tools to Support Integration

A measure of Framework maturity is the availability of tools to support tool integration. There are some areas where tools would be advantageous.

Tool registration: Registering a tool into the environment includes defining the type of tool and its parameters to the framework user interface to allow the tool to be invoked. In addition, the tool registration should integrate the tool into the methods supported by the environment.

Name Conflict Resolution: Tools are typically developed with an assumption about the names of objects and object attributes. Tools that support the definitions of new views or aliases for accessing framework-managed data should allow new tools to be integrated without modification. The CAIS-A interfaces include a mechanism for resolving name conflicts through a view mechanism. The PCTE allows one to define subschemas to resolve conflicts.

Database Schema Editor and Migration: New tools may require additional data types or additional attributes on existing data types. A framework tool that supports extension of the schema, as well as migrating the existing database to conform to the new schema, is needed when integrating new tools. Database migration is a difficult problem when integrating new tools. Most environment frameworks do not provide adequate support for database migration. Consequently, the tool integrator may have to write utilities that migrate the database from its current state to a new state that is compatible with the new tool.

2.3. User Interface Technologies

The user interface is a critical component of an environment. To a large extent, it determines the perceived value and effectiveness of the tool. The existence of various technologies requires a user interface that employs a separation of concerns between intermediate representation of information and the presentation of that information. These technologies include:

- extensibility in the types of information presented to users
- incorporation of new display paradigms
- multiple views for editing

There must be a separation of device-dependent input and manipulation of internal information. This will allow the user to tailor the input mechanism. A good architecture will allow the functionality to evolve independently of the user interface.

The following sections are a survey of different paradigms and architectures for designing interactive user interfaces. All of these approaches are based on the object-oriented paradigm.

2.3.1. Model-View-Controller Paradigm

The Smalltalk Model-View-Controller (MVC) paradigm [KRA88] evolved from a desire to build interactive applications where one maximizes the portability and reusability of the components that support the interactive graphics. Figure 2.3.1-1 illustrates the three-way separation of concerns supported by this paradigm.

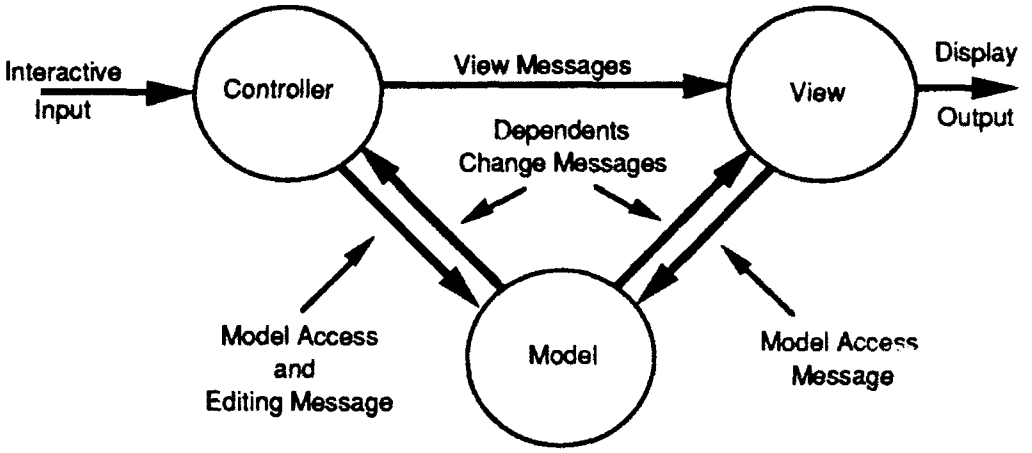


Figure 2.3.1-1. Model-View-Controller User Interface Paradigm

The three main components of the MVC paradigm include the following:

1. A Model is the object that maintains application-specific state information. The Model does not have knowledge about how it is being displayed or how its inputs are being handled. A Model has operations to access and change its state. Whenever a Model changes its state, the Model sends messages to all of its dependents (associated Views and Controllers) notifying them that the Model has changed.
2. A View is responsible for determining how to present the model. A View will revise its rendering of the Model whenever it receives a message indicating that the Model has changed state. In addition, some of the user inputs may affect the View (e.g., scrolling), requiring it to update its rendering of the Model. The View queries the Model to determine the Model's state. The View then adjusts its rendering of the Model accordingly. In general, a Model can have multiple Views associated with it.

3. Each Controller has an associated View. The Controller mediates between input devices (e.g., mouse, keyboard, menus, scrollbar), the Model, and its associated View. Whenever a significant event occurs at an input device, the Controller logic determines whether the event should just change its associated View (e.g., scroll the view), or if the event is a directive to change the state of the Model. Messages are sent to either the Model or View object as appropriate.

The separation of concerns in this approach is that Models know that they have Views and Controllers associated with them. However, Models do not have explicit knowledge about the functions performed by these Views and Controllers. This architecture has several advantages:

- The Model portion of the applications can be developed independent of the input device and language. This capability makes the application highly portable, which is a desirable attribute of Catalyst.
- The application is unaware of how may views are active at any given time. Therefore, the architecture is extendable, allowing new views to be added without modifying the application Model. This capability allows Catalyst to be extended to provide new renderings of the software being edited. It also allows new presentation styles so that the quality indicators can be inserted.
- The architecture allows editing of one Model view while automatically updating the other views whenever a change is made to the Model. This capability satisfies the Catalyst requirement to support multiple, editable views of the software being developed.
- The architecture allows tailoring of the user interface in a manner which is independent of the model. Consequently, the style of interaction can easily be changed without affecting the behavior of the tool.

2.3.2. Artist Paradigm

The Arcadia-1 Software Engineering Environment project has adapted a user interface called Chiron [YOU88, MYE83]. Chiron is based on the Artist paradigm of structuring user interfaces, as shown in Figure 5-6. The Artist paradigm is a variant of access-oriented programming, where each invocation of an operation on a model has the side effect of invoking an operation to update the view.

An artist adds new state information to keep track of the model's rendering. The artist also attaches additional behavior to the existing operations, either through specialization or through the use of triggers. With this approach,

neither the syntax nor the semantics of the operations on the model are changed. Multiple artists can be defined by simply repeating this process. An advantage of the artist approach is that it can be implemented using a language like Ada to provide data abstraction capabilities.

2.3.3. Access-Oriented Paradigm

Another technique that supports the multi-view, simultaneous update paradigm is access-oriented programming [STE86]. In access-oriented programming, the fetching or storing of data can cause procedures to be invoked. Access-oriented programming differs from object-oriented programming in the area of actions and side-effects. In object-oriented programming, when one object sends another object a message, the receiving object may change state as a side-effect. In access-oriented programming, when one object changes its state, a message may be sent as a side-effect. Of course, one could employ a development paradigm that incorporates both object-oriented and access-oriented programming.

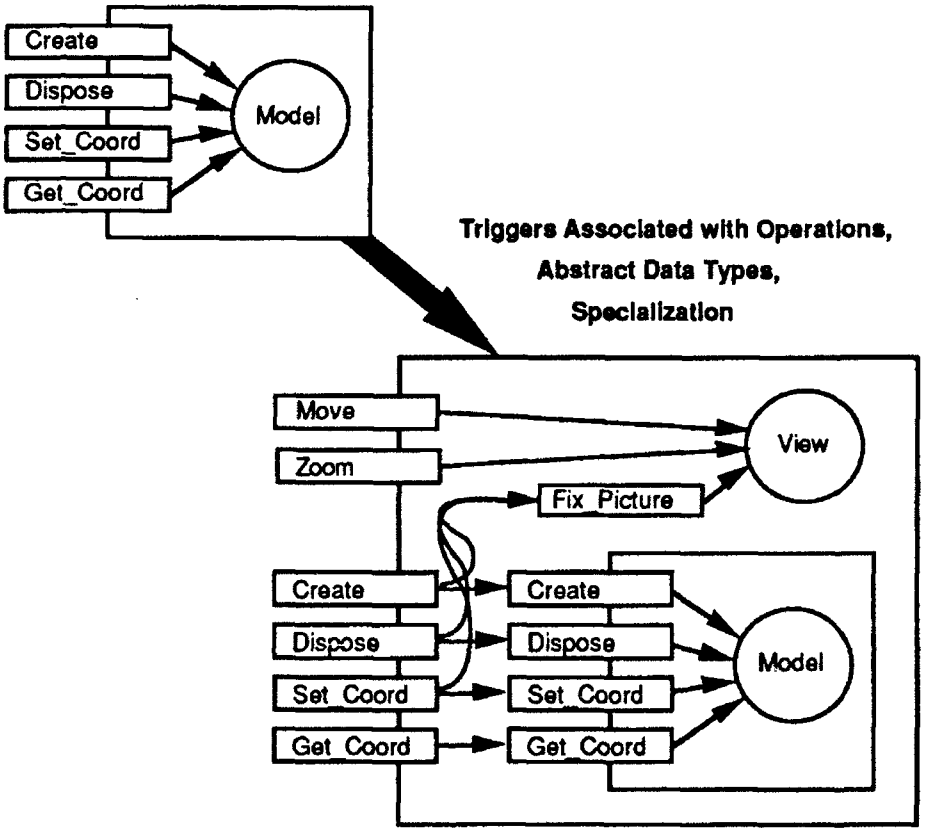


Figure 2.3.3-1. Artist User Interface Paradigm

The Loops language [STE86] allows procedures to be dynamically associated with data so that these procedures are invoked when data is fetched and stored. An application of this approach is to attach gauges (e.g., Catalyst quality indicators) to objects to monitor their state. For example, in Loops, one can simply create an instance of the appropriate gauge and send a message which will attach itself to the desired object property. Subsequent changes in the value of that property will register on the gauge as a side effect of that change.

2.3.4. Constraint-Oriented Paradigm

The constraint-oriented paradigm employs a capability supported by some object-oriented systems in the ability to define property values of graphical objects in terms of the property values of other graphical objects. For example, the fact that two software packages have a dependency can be described by a constraint that forces them to remain graphically connected, even when one of the packages is moved. In this case, whenever the properties of a graphical object change, the relevant constraints are automatically enforced, and the affected graphical objects are automatically redrawn. Examples of languages that support these constraints are Coral [SZE88] and the Constraint Window System (CWS) [EPS88]. Constraints have been used for other drawing programs, such as SketchPad [SUT63] and ThingLab [DUI86]. SketchPad's constraints allow lines to have specific relationships to other lines. ThingLab extends SketchPad to provide a general simulation environment.

Constraints can be expressed between data objects and graphic objects so that whenever the value of the data object changes, the effects of those changes are propagated to the property values of the graphic objects. These objects, in turn, update themselves. There is a difference between constraint-oriented and access-oriented paradigms. In constraint-oriented programming, constraints are value expressions constraining the property value of one object in terms of the property values in other objects. On the other hand, access-oriented programming uses triggers to identify procedures that are invoked whenever a specified property value changes. By incorporating constraints as a language-supported mechanism, some clever optimizations can be performed by compilers to provide efficient implementations of the constraint mechanisms [SZE88, EPS88]. The architecture of Coral programs is shown in Figure 2.3.4-1.

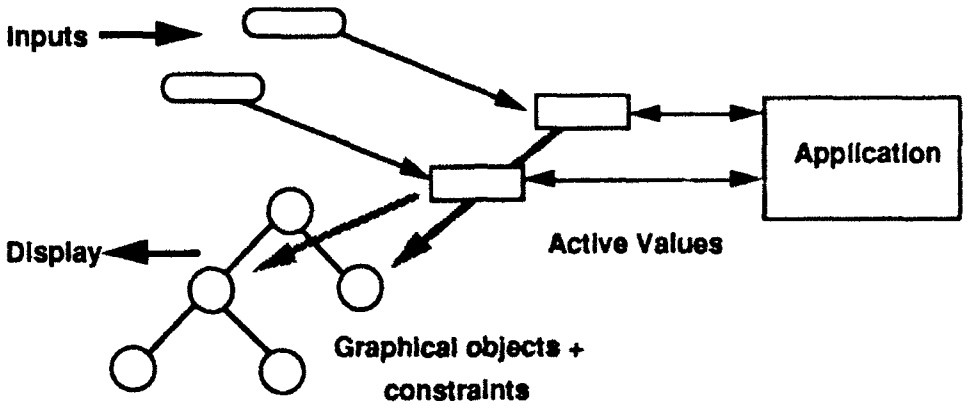


Figure 2.3.4-1. Constraint-Based User Interface from Coral

The graphical objects have defined constraints to ensure that the graphical renderings of these objects conform to the constraints Whenever these values change, the graphical renderings are updated accordingly. An active value is a data value plus a list of objects and procedures that depend on that value. When the state active value is changed, the graphic objects are informed so that they can be redisplayed. Procedures are then invoked to notify the application program of the change. In Coral, interactors handle the information from input devices and update the active values accordingly.

An advantage of the constraint-oriented approach is that constraints are declarative and enforced uniformly for all objects to which the constraints apply. The concept of active values provides a clean way of separating input and output portions of the user interface. The concept also separates the user interface from the rest of the program.

3. SECD Environment and Rationale

Catalyst is a set of highly adaptable and configurable software "building blocks" in the form of interactive software tools and environment frameworks, that, when integrated with an installed computer system (hardware plus system software) and other software in the users' environment (tools and frameworks), will provide an automated support system for systems engineering.

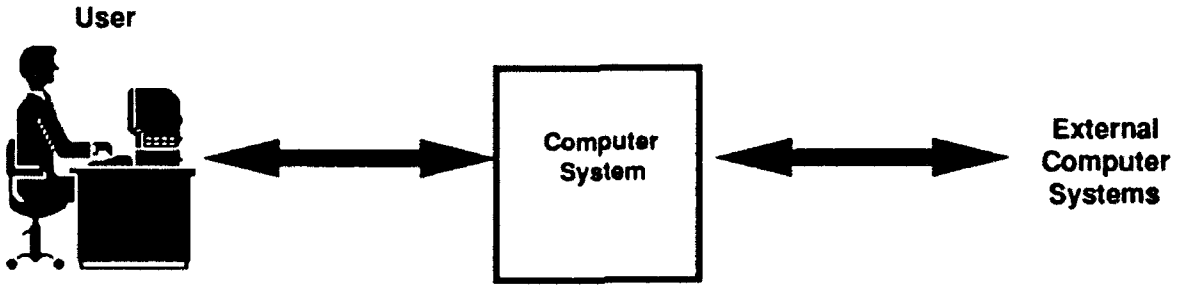
Catalyst is a software system. Catalyst cannot, by itself, be considered a systems engineering environment because it depends on the existence of an underlying computer system and certain additional software in order to be operational.

The following subsections discuss the principal Catalyst concepts. We will start by describing what an environment is, providing definitions for key terms (e.g., *environment*, *tool* and *framework*) that will be used later and providing a conceptual model of an automated environment for future use. Next the concept of Catalyst consisting of "building blocks" is explained.

3.1 Model of an Environment

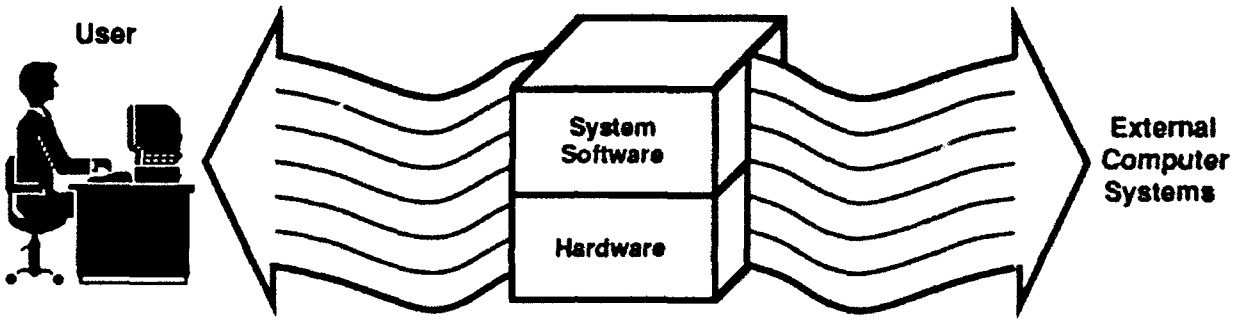
A *environment* consists of a computer system plus a set of software tools which are designed to provide an encompassing breadth of support for some human endeavor. In this case, the human endeavor to be supported is that of systems engineering.

A *computer system* consists of a set of hardware and operating system software for processing, data storage, user interface, communications and resource management. Computer systems interact with users and communicate with external computer systems.

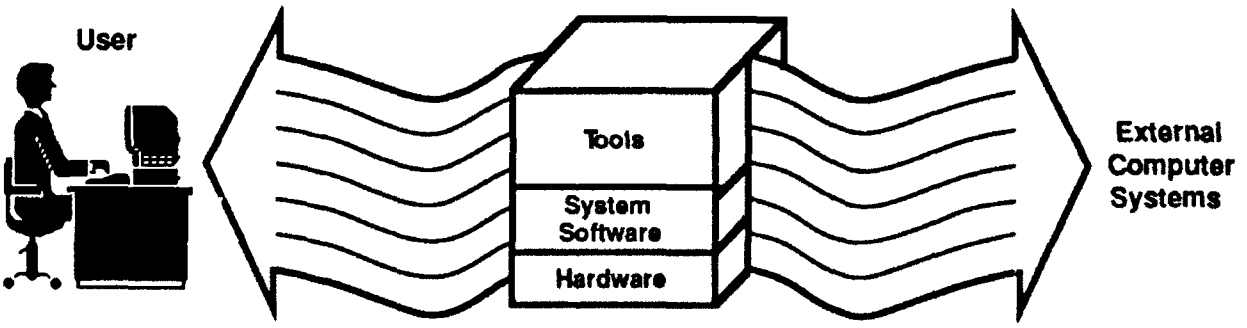


The hardware and operating system forms the lowest level *virtual machine* that interfaces to users and tools. It is called a virtual machine because the partitioning between hardware and specific software is transparent. To the users and tools, it is merely a collection of processing, data storage, user interface, communications and resource management services.

In the diagram below, the communication with users and external computer systems is shown as a "ribbon" to illustrate that such communication involves both the system software and hardware.



A *tool* is software which serves to automate some specific aspect of the endeavor. Tools also use the services provided by the virtual machine to accomplish their function. Tools utilize the virtual machine services to communicate with the user and with external computer systems. Tools also use the virtual machine services for inter-tool communication and for information sharing amongst tools.



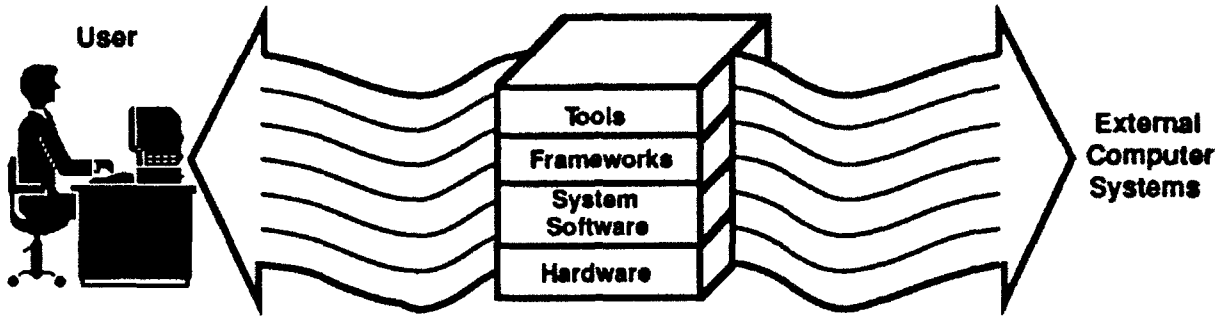
A *framework* is an underlying infrastructure of an environment that, through a set of common software services, enhances the commonality and integration of tools, and thereby improves the collective use of these tools for a specified endeavor.

A framework is, itself, a collection of software tools which provide common services that are shared amongst other software tools. Frameworks extend and enhance the services of the virtual machine, typically providing higher level, more powerful services for users and tools. Frameworks typically enhance the integration and/or commonality of tools through shared use of these common services. A framework typically provides the following types of services:

- Information management
  - Higher level data model access
  - Data interchange
  - Configuration management
- User interface
  - Higher level user interface primitives
  - User access to framework services
- Communication
  - Higher level protocols for external communication
  - Inter-tool communication mechanisms
- Resource management
  - Tool execution and control
  - User and tool access control
  - Higher level computing resource management

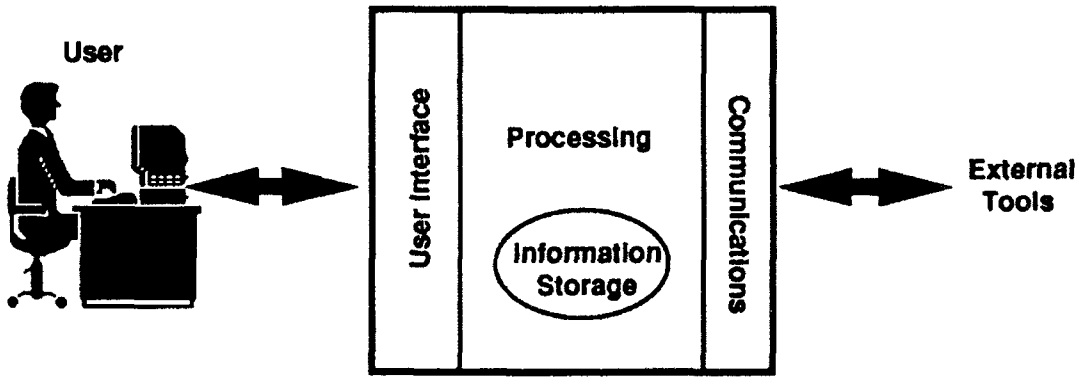
The framework common services often work in concert with a tool in accomplishing a particular tool function and is often not itself visible to the user. This latter, more transparent view of a framework, is better described as a layer or infrastructure beneath the tool. A layered model portrays frameworks as being ubiquitous (i.e., being or seeming to be everywhere at the same time; omnipresent).

A framework may consist of a layering of various other frameworks. This layering serves to provide high and more powerful common services for tools. It is important to note that in these cases, a single function of the tool may involve support from many layers of frameworks beneath it.

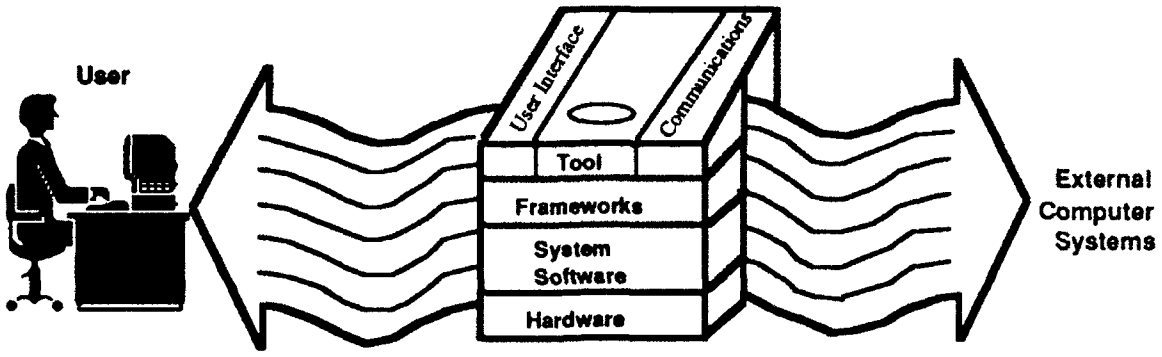


Tools typically have the following four components:

1) a user interface
2) a communications interface (to other tools)
3) a processing component (which provides the underlying capabilities/functionality
4) an information component (for retaining information between invocations)



The services of the virtual machine (hardware plus system software) and the services of frameworks may facilitate the user interface, communications, information management, and/or the processing components typically found in other tools. While providing services, frameworks can also implement protocols and support policies (e.g., UI frameworks promote look-n-feel policies by implementing specific protocols).



The processing component of a tool may, in fact, contain and/or utilize other tools. Tools may contain other tools; some tools automate the use of lower level tools. In addition, software tools can be manipulated as data. As a result, tools may serve to generate, translate, or modify other tools.

3.2 Catalyst Building Block Concept

The systems engineering automation focuses on the development of high payoff *building blocks* for a systems engineering environment, rather than attempting to specify a complete and comprehensive automation of the systems engineering process. These "building blocks" consist of highly adaptable and configurable software tools and environment frameworks, that, when integrated with an installed computer system (hardware plus system software) and other software in the users' environment (tools and frameworks), provide an automated environment for systems engineering.

A complete, monolithic systems engineering environment (i.e., all tools and all required frameworks) is considered infeasible for a number of reasons:

- The systems engineering process, as practiced across the numerous types of mission-critical systems, is too broad to practically automate in a comprehensive fashion.
- There is a high degree of variability in the systems engineering process across organizations that precludes a single environment solution.
- Each organization already has a number of tools, frameworks and computing hardware that it has experience with and will want to continue to use.
- A single monolithic environment is not considered commercially viable; organizations prefer to pick and choose their tools from multiple sources.

The building block approach seeks to identify a set of widely applicable and highly adaptable tools and environment frameworks that can be integrated with other COTS (Commercial-Off-The-Shelf) tools and frameworks, and internally-developed tools and frameworks to form an organization's systems engineering environment.

Following this concept, an organization's systems engineering environment will resemble distributed, cooperating *islands of automation* that are extensively adaptable. It is our hypothesis that automated tools transition faster and more effectively if they are flexible enough to simply automate the users' existing process, rather than requiring the users to change the way they do business. As a result, the tools and building blocks that are specified will be *ultra-flexible*, supporting adaptation along many dimensions, allowing organization-specific definition of:

- Process and life cycle
- Organization structure and roles
- Methods and techniques
- Information and representations
- Work flows and work products
- Policies and procedures

Moreover, the tools and building blocks must be portable to a number of computing platforms and support integration with the organization's installed base of COTS and internal tools.

Each tool and framework building block should strive for power and simplicity within a narrow scope, yet be highly interoperable with other tools and framework building blocks. An integrated set of highly effective, single purpose tools is better than a single, monolithic environment that attempts to solve all problems. The integration mechanisms between these tools, whenever possible, should be ubiquitous (using the framework building blocks), rather than being a recognizable layer to the user.

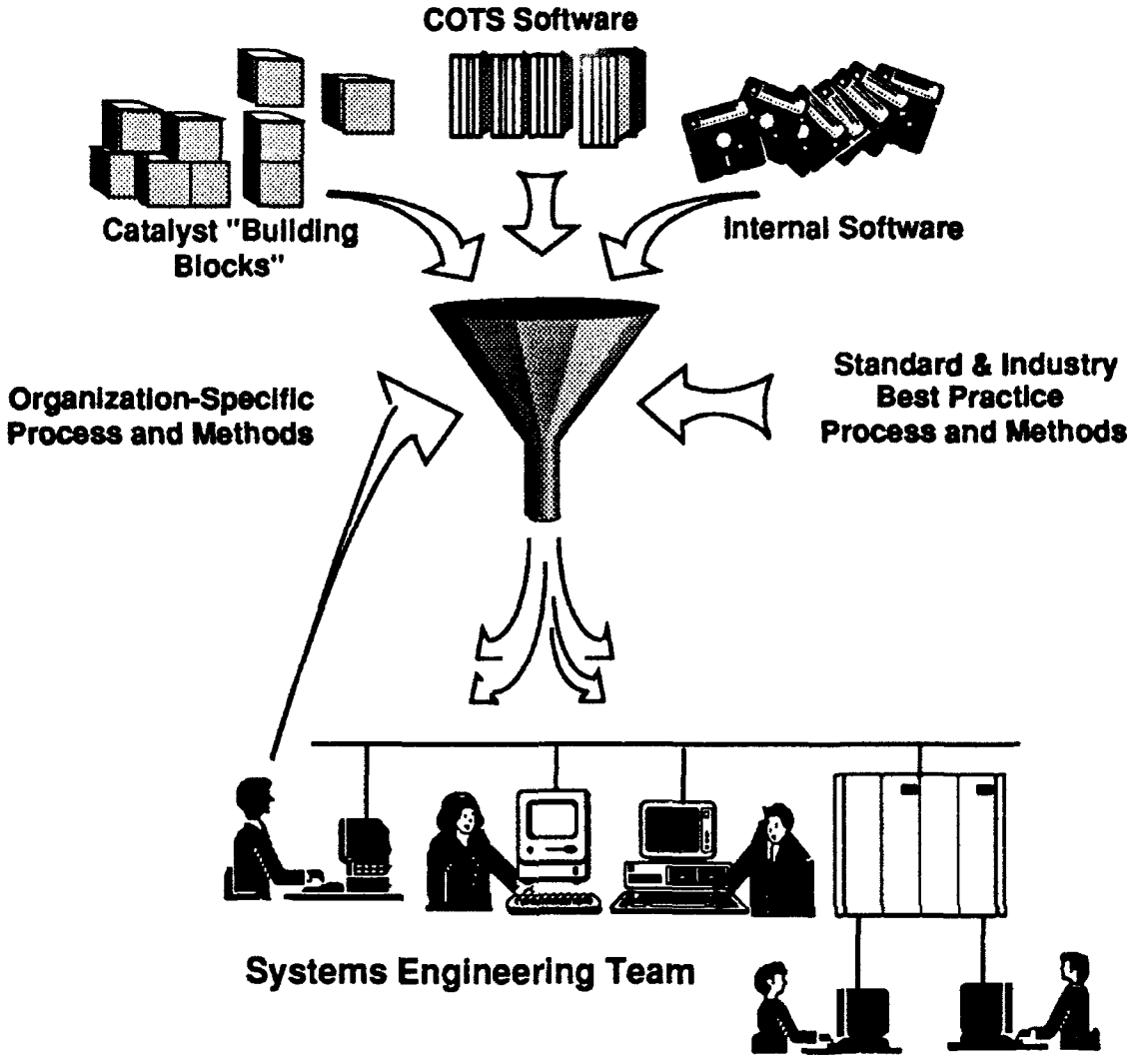


Figure 3.2-1. Building block concept for systems engineering automation

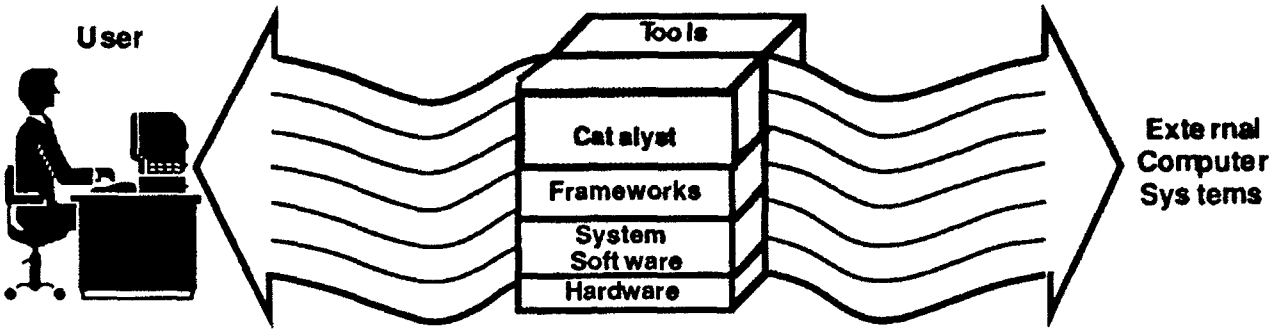
3.3 Catalyst Environment Interfaces

Catalyst has the following interfaces:

- User's host computer system, consisting of the computer hardware and system software
- User's installed frameworks
- User's installed tools
- Users (employing the services of the user's host computer system hardware, system software and possibly installed frameworks)

- External Systems (employing the services of the user's host computer system hardware, system software and possibly installed frameworks)

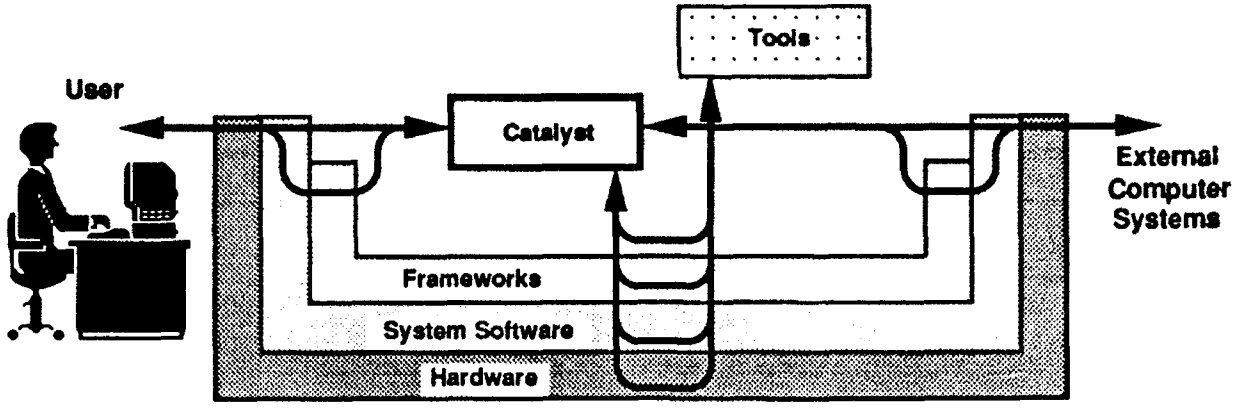
Note that the latter two interfaces are layered on top of the user's computer hardware, system software and frameworks. The program interfaces of the Catalyst software involve only the system software, user frameworks and user tools.



Communication between Catalyst and the user may be accomplished through direct use of system software services (e.g., terminal drivers) or through use of user interface frameworks (e.g., X Windows, Motif), that in turn use the underlying system software and hardware to communicate with the user.

Similarly, communication between Catalyst and external systems may be accomplished through direct use of system software services (e.g., low level communication protocols such as X.25 or Ethernet) or through use of communication frameworks in the form of higher level communication protocols (e.g., NFS, TCP/IP), that in turn use the underlying system software and hardware to communicate with the external systems.

Catalyst may communicate with other tools, either directly (e.g., procedural interfaces), through use of system software services (e.g., process-to-process message services), or through frameworks (e.g., tool access protocols). The underlying software services and frameworks will typically provide facilities to communicate with tools that execute on distributed computer hardware in the system.



3.4 Computer System Environment Requirements

Catalyst must be adaptable to support the systems process across a wide range of computer system platforms and configurations, including the following:

- Standalone computer system with explicit communications to external systems
- Host-workstation configuration
- Heterogeneous computing network
- Multiple heterogeneous computing networks with internetwork communications

3.5 Environment Focus

The *Catalyst* automation focuses on high payoff systems engineering automation, specifically in the areas of:

- **Modeling and specification meta-tools**, a set of highly adaptable and configurable tools that may be used for a variety of tasks and in a number of different contexts, particularly those early life cycle requirements and system design specification activities.
- **Concurrent engineering groupware**, tools specifically designed to promote and enhance multi-person, and particularly multi-disciplinary, interactions and collective work flows.
- **Integration mechanisms**, frameworks that increase the effectiveness of multi-tool work flows and promote information sharing between the systems engineer and the various specialty roles.
- **Environment administration support**, tools assisting the installation, adaptation and extension of the environment.

- **Process automation**, providing specialized support for defining, planning, tracking, enforcing and improving the systems engineering process.

The determination of the highest payoff area was determined by the market survey, process modeling and field interview activities.

4. Technology Demonstrations

As defined in the SECD SOW, the Technology Demonstrations identified tools and enabling technologies which were considered critical to establishing an automated system engineering environment. The following technologies were demonstrated for the SECD Project Team:

- FIELD (Friendly, Integrated Environment for Learning and Development), a system with message-based integration
- Versant, an object-oriented database product
- ArborText, an electronic document production product using SGML
- Automated Access Experiment (AAE), a SPS delivered system to RL for distributed, heterogeneous computer systems
- InQuisiX, a SPS meta-tool
- InSight, a classification and search engine productized by SPS and SAIC
- Momenta, a Pen-Based Computer
- PRICE (Parametric Review of Information for Cost and Evaluation), a cost analysis tool

The following subsections describe each of these technology demonstrations.

4.1. Friendly, Integrated Environment for Learning and Development (FIELD)

Message broadcasting is an integration technology that is the foundation of FIELD (Friendly, Integrated Environment for Learning and Development) developed by Steven P. Reiss, Brown University, Providence, Rhode Island [REI90]. FIELD is an open collection of tools that uses selective broadcasting and source annotations as an integration mechanism. FIELD emphasizes visualization tools and standard workstation interfaces.

The requirements for the development of FIELD were extensible, simple enough for novices, rich enough for research. FIELD needed to accommodate multiple processes and machines, create an excellent student environment, and be inexpensive to build and maintain. Integration required that the tools must interact with each other directly; dynamic information must be shared; all source action should be through a single view; and specialized information must be available to all tools.

In the architecture of selective broadcasting, the central message server exists for the clients to send/receive messages and command requests and information. Clients can be added or removed. Messages between the server and the client are strings and conventions for encoding are used (e.g., WHO CMD system args, ID WHAT system information). In the operation of a message, clients register patterns and tools send messages to the server. The patterns can be a "scanf-based" pattern matching, exact normal text matches, or argument matches. The tools can rebroadcast their messages selectively, or use "printf" formatting for sending calls. Policies are declared for users, tools, patterns, variable, and load requests. Additional policies can be expressed for complex actions using path expressions.

Messages can be asynchronous (i.e., the sender continues immediately) or synchronous (i.e., the sender can wait for the first non-null reply or he can wait for all receivers). The messages are decoded by the server and the sender receives the arguments and the identification of the reply. The server is responsible for services for the tools and is central to the environment. The server maintains a working directory for all tools, a check and a start for services, and consistency within the environment.

FIELD offers cross referencing to create an interactive, fully relational database. The user can define queries and FIELD retrieves the set of desired entities. FIELD offers browsing capabilities using a class hierarchy, selective inclusion and exclusion of entities, and local graphs for a node. Graphical views of the class hierarchy show class relationships, member information, member inheritance, and window information. Program execution can be traced as messages are sent between client and server for communication.

FIELD is used extensively at Brown University for research and instruction. In January 1991, Digital Equipment Corporation commercialized FIELD as Fuse version 1.0 (i.e., Field using Motif). DEC Fuse version 1.1, subsequently released in November 1991, runs on Ultrix and Sun OS. Fuse uses message broadcasting technology, also known as the object request broker, and is the basis for Softbench and Tooltalk, two COTS tools.

4.2. Versant

Versant is a COTS object-oriented database, developed by Versant, Menlo Park, CA. Object-oriented technology is improving programmer productivity by a factor of 10 and database performance by a factor of 100 [RUM91]. After nearly forty years of refinements in software development, the capabilities of software still lag far behind the power of hardware, and the gap continues to widen every year. Versant's goal is to change this trend.

Historically, hardware has shown dramatic improvements because most computers are now designed and assembled out of standard components such as integrated circuits and printed circuit boards. With the advent of object technology, software can also enjoy the benefits of construction by assembly.

Object technology can now supersede the monolithic, hand-crafted programs of years past with collections of reusable software components. This goal will allow programs to be assembled from existing components in a fraction of the time and cost of traditional software development. It will also result in much more robust, flexible software than we have today.

Using objects, methods, and classes, an object-oriented approach reuses classes and builds models of how a system works. This model can then serve as a framework upon which any number of applications can be constructed. And, if the way an organization does business changes, the framework can be modified without having to rewrite all the applications that depend upon it. Building models increases the reusability of the classes developed while making applications easier to modify.

Once a development team has built a solid foundation of reusable models and frameworks, new applications should contain no more than 20% new code. It is the construction of these frameworks that is the most significant investment in object technology and the major source of benefit as well. Versant enables the user to build these models and frameworks.

Object-oriented technology also offers important advantages in the construction of databases and the large information systems of which they are a part. One advantage of objects for information systems is that they are not identified by their contents. Each object has a unique internal code called an object identifier that is automatically assigned by the system. These object identifiers have useful properties:

- Object identifiers provide a very efficient way of accessing objects. They are almost as fast as the address pointers used in some languages and early database management systems, but they are less vulnerable to relocation than pointers.
- Each object has a unique identity throughout its life span, even if all of its contents change. This is very convenient for information systems because it eliminates the need to define and maintain unique keys based on the contents of entries.
- Composite objects actually "contain" other objects by storing object identifiers in their variables. Containment by reference allows individual objects to participate in any number of composite objects. It

also allows composite objects of any complexity to be retrieved very quickly. At the same time, the structure of composite objects is easily changed because the structure is defined in the objects themselves, not in the database management system.

Object technology has made possible a new type of database that is rapidly eclipsing the capabilities of previous database technologies. For example, object databases can store any kind of information that can be digitized, including pictures, diagrams, sound speech, and video images. Object databases can handle information of any complexity, including multi-level engineering drawings and compound documents, with live links among text, spread sheets, and graphs. No matter how complex, anything stored in an object database can be defined as a new class of object. As such, it can be stored, retrieved and processed as a unitary entity.

Because objects contain methods as well as data, information stored in object databases can manage its own access. This simplifies every application that uses this information by eliminating all the redundant data-access functions. The manner is which data is stored can be changed without having to modify, recompile and redeploy every application that relies on that data.

The use of object identifiers to form the links within compound objects allows these objects to be retrieved far faster than access based on foreign keys and time-consuming joins. In fact, object databases have been shown to outperform relational databases when dealing with complex information.

Vendors of relational databases are attempting to match these advantages by adding object-line extensions to their products. However, as long as they maintain the basic relational model and its reliance on joins and foreign keys, the performance of a true object database cannot be matched.

The future of information systems belongs to object technology. All the major vendors, from IBM in hardware to Microsoft in software, have announced plans to build their future products around object technology. A major industry consortium, the Object Management Group, is working with over a hundred membership companies to standardize the technology so that objects from different vendors can be mixed and matched freely in the same information system.

The scale of software systems today demands this kind of openness. No one vendor can possibly supply every component of an enterprise-wide information system. The common language of objects, and the disciplines imposed by message-based interactions, will make it possible for companies to

The days of the monolithic application are numbered. Just as hardware made the transition from closed, proprietary architectures to open systems, so too, will software move to systems composed of objects from different vendors collaborating to produce a unique solution. Object technology is the breakthrough that makes this transition possible.

Versant carries an impressive customer list that includes U.S. Navy, Wright-Patterson Air Force Base, General Electric, Hewlett Packard, Hughes, IBM, Martin Marietta, McDonnell Douglas, Nippon, and Toshiba Company.

4.3. ArborText

ADEPT, (Adaptable Electronic Publishing Technology), is a COTS tool sold by ArborText Inc., Ann Arbor, MI, and Longwood, Fl. The ArborText ADEPT Series is an adaptable, electronic documentation system for creating, editing and publishing SGML documents. The software's capabilities conform to CALS requirements by encoding the documents in SGML [SEY90].

The software includes SGML; the Publisher and SGML; an Editor, authoring tools for SGML documents; a Document; an Architect, a tool for writing and validating DTDs; a portability package for graphics conversion, and a tool for reading and writing MIL-M-1840A tapes.

ADEPT provides a structured documentation system with interchangeable formats, document portability, system-wide consistency, and increased productivity. ADEPT runs on the Sun, Hewlett-Packard and DEC workstations. The software meets the requirements for an automated system engineering. ADEPT supports the requirement for Work Product Display, Output and Editing Support. ADEPT supports the editing of a complete work product in a WYSIWYG view. ADEPT supports the output of formatted work products to external document formatting, production and publishing systems in compliance with CALS using SGML. ADEPT supports convenient access of an external tool to edit information and provides an on-line thesaurus and spell-checker.

ADEPT has a built-in interactive and continuous parser that helps the author stay within the bounds established by the DTD and greatly reduces or eliminates the need for correcting errors when a document is completed. A document can be labeled an "active" and can be programmed to change, without user intervention, based on changes to its environment. The document is programmable through the software's command language, based upon C and UNIX C shell syntax.

ADEPT is a proven SGML product with a wide DoD and commercial customer base. For example, Computer Sciences Corporation (CSC) is a user of ArborText products to form a large-scale information network to streamline the process of building and maintaining weapons systems. ADEPT is streamlining Mobil Exploration and Producing Services' information exchange to ensure that basic engineering standards are applied consistently throughout the organization. Hewlett-Packard Company uses ADEPT to create documentation for their products, and ranks among the largest single-time sales of licenses within the UNIX operating system-based electronic publishing industry.

4.4. Automated Access Experiment (AAE)

Automated Access Experiment (AAE) developed for Rome Laboratory (RL), was a set of network administration support tools that demonstrated access to a heterogeneous multimedia database. These tools supported the definition and maintenance of data structures (views) representing information resources accessible via the network. The prototype system, developed in Ada and *Classic-Ada*, demonstrated transparent read-only access to multimedia databases used by Air Force intelligence agencies. The AAE used an Entity Relationship network schema as the unifying model for data description. An advantage of using the ER data model is the existence of well-defined rules for transforming it to a hierarchical, network, or SQL model. SPS prototyped a subset of the IRDS data dictionary systems to support the experiment. The prototype was completed in September 1991, and continues as a building block for future technology development at SPS.

AAE attacked one of the most important problems in computer information technology in the 1990s - maintaining and using diverse data storage and retrieval systems that support large organizations. The typical large organization has a collection of heterogeneous and distributed computers that store data in multi-media types and formats. The tasks identified in the AAE effort were to design, implement and demonstrate tools to support transparent access to network-resident data regardless of type, structure, location or storage of media. The experiment was demonstrated in a well-defined test bed.

SPS developed a system, Automated Transparent Object Management System (ATOMS) to solve the problem of distributed, heterogeneous computers. ATOMS was characterized by the following characteristics: provided transparent access to distributed, heterogeneous, multimedia data; provided powerful user-defined access to data through data views and queries; supported data correlation and fusion, controlled user access to data and identified classified data; provided a modern bit-mapped user interface;

demonstrated the concept of modeling heterogeneous, multimedia data using the Entity-Relationship (ER) Model; and was scaleable and extensible without extensive reconstruction of the existing software.

The ATOMS server architecture includes a data server, that is, each data source is encapsulated by a TCP/IP network server process. Each data server is responsible for translating between the native data manipulation language of the data source and the ATOMS ER language. Each physical database and service type residing on an ATOMS node is accessed through a network server and runs in the background, waiting for service requests. The network servers allow a user on one machine to retrieve data from databases located on any ATOMS machine on the network, and further allows the user to log onto any ATOMS machine on the network and access their own schema views and correlations.

Attributes can be retrieved from a data source and can be of the type string, float, integer, boolean, file (dialogic), file (tiff), file (xbm) and file (text). Attributes tiff and xbm hold graphics which can be displayed in ATOMS. An attribute dialogic is a message attribute which enables the user to listen via a telephone.

ATOMS provides a correlation mechanism to support the intelligent information correlation and fusion. A correlation allows an intelligence analyst to mark a set of one or more entities and/or relationships as correlated and then record a textual description of their ignificance. Correlations are stored for later retrieval and modification.

The ATOMS solution to the problem of distributed access uses several levels of ER schema information. Although not intended to be a secure system, ATOMS does maintain and present DoD security classification information to identify classified material (i.e., unclassified, confidential, secret, top secret). ATOMS keeps a log file of all changes to the local schemas, network schemas, network views correlation sets, users and access control lists.

ATOMS has a friendly, graphical, mouse-based and window-oriented user interface. ATOMS uses features such as pull-down menus, pop-up dialog and alert boxes, check boxes, option buttons and scrollable window panes. ATOMS exists inside the X Window System™. Windows allow the user to manipulate schemas, subschemas, correlations and view definition.

4.5. Reliable Specification Execution Tool (RSET)

In this Phase I SBIR contract with NASA Langley, SPS developed a reliable object-oriented specification technique that supports the formal incremental,

and humanly verifiable description of system behavior. SPS' specification technique is based upon the proven Box Structures methodology, invented by Dr. Harlan Mills. SPS has been awarded Phase II to develop and demonstrate the technology with Dr. Mills as our consultant.

In the Phase II follow-on contract, SPS is developing tools to support specification activities as well as the simulation of systems from their specifications. SPS recommends the use of value, existence, and temporal constraints to model a wide range of system behaviors, including concurrency, timing dependencies, and real-time systems. SPS is using a pragmatic approach to support a wide variety of constraint definitions including linear and non-linear equations, equations defined by tables, and equations incorporating functions.

For Phase III, SPS is planning development of the technology previously sponsored by NASA Langely. The productization is directed towards an extendable and tailorable engineering drawing and documentation tool, or a meta-tool, called Insight. InSight is an advanced prototype meta-tool with user tailorable graphics and graphic semantics. InSight's users can associate information with graphic objects and tailor that information as well as the presentation format and the generation of documents. The tailoring process is easy to use and non-threatening. InSight supports concurrent engineering by allowing team members to share work products in a highly cooperative manner, and can integrate with existing office automation and engineering tools. Consequently, Insight is adaptable to existing and new engineering methods that an organization may employ.

InSight is based upon constraint technology and offers the following advantages:

- Provides a natural expression of relationships between objects
- Is Highly suited to integrate with CAD and CASE applications.
- Has high object access performance using intelligent caching and clustering, virtual memory, fast disk drives
- Has optimized performance for constraint evaluation
  - Multiple re-evaluations and critical constraints
  - Space and maintenance overhead of constraint-defined dependencies

The initial build of InSight is demonstratable and future products are planned to increase intelligence and enhance existing capabilities. SPS sees a meta-tool as a critical component of an automated system engineering environment.

SPS is cultivating growth markets for continued development of InSight to provide better value-added products for the automation of the system engineering environment.

4.6. InQuisiX™

InQuisiX™ is a classification and search engine for software reuse developed, jointly, by SPS and Science Applications International Corporation (SAIC) under a Phase III SBIR program. InQuisiX is in its sixth year of research and development, and initially started as a pair of Phase I and Phase II SBIRs through NASA Langely Research Center and U.S. Army CECOM. SPS and SAIC have formed a strategic partnership to facilitate software reuse, a concept that offers the best near-term potential for lowering the cost and improving the quality of software-intensive systems.

A software reuse library is a catalyst for integrating reuse into the development process. A software library of assets is tangible and understandable by managers and engineers alike. Collecting assets naturally leads to defining reuse policies and procedures for developing, verifying, managing, distributing, and reusing assets. Classifying assets naturally leads to domain analysis activities which naturally leads to development of high payoff assets. The reuse library serves as a focal point for promoting reuse in an organization and for sharing reuse experiences and lessons learned.

InQuisiX is an ultra-flexible reuse library system that was designed to be integrated into an organization's existing environment. The reuse library has three major roles, the Librarian, the Software Development Subscribers and the System Administrator as seen in Figure 4.6-1. The Librarian gathers domain analysis studies, software components from other sources, designs a classification scheme, and catalogues software components. The Software Development Subscribers can browse, search and retrieve components. They also develop software components internally and route to the Librarian as well as route search, requests and responses to a Software Development Environment.

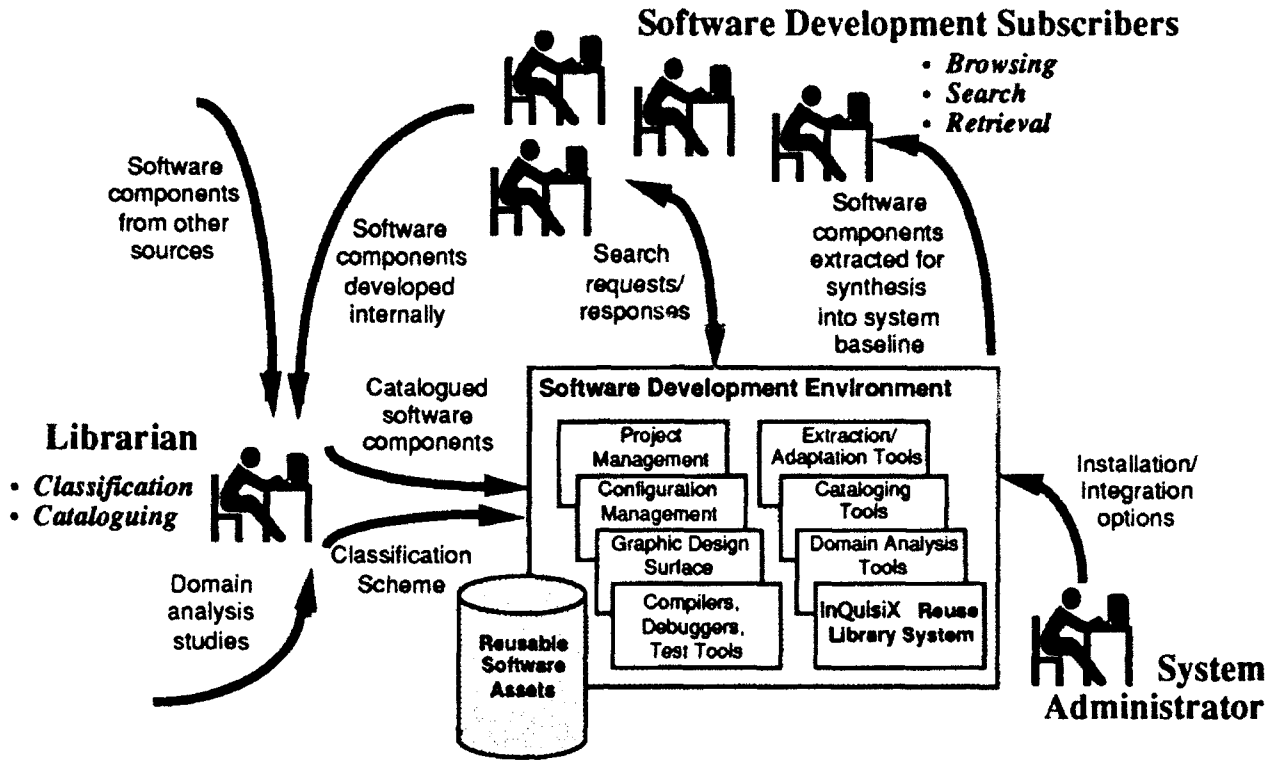


Figure 4.6-1 Reuse Library Concept of Operations

The Software Development Environment is a collection of cooperative tools that can include project management, configuration management, graphic design, compilers, debuggers, test tools, extraction/adaptation tools, cataloging tools, domain analysis tools, and InQuisiX itself. The Software Development Environment can be any environment (e.g., SLCSE). The Software Development Environment provides software components for the Software Development Subscribers for synthesis into a system under development. The System Administrator is responsible for installing and integrating the Software Development Environment.

InQuisiX supports management of heterogeneous classes of assets. User-defined component classes represent the finds of components that may be catalogued, search for, examined, extracted and reused. Each component class may require different information to be captured in the form of attributes. A component class hierarchy can be annotated with the attributes defined at each class level.

Because the effectiveness of a reuse library is enhanced by an integrated set of facilities to support a wide variety of user-defined classifications, InQuisiX supports the following different classification techniques at the same time:

- Faceted classifications (controlled vocabulary)
- Keyword indexing (uncontrolled vocabulary)
- Free text indexing information retrieval
- Characteristics-based attributes
- Metrics-based criteria (i.e., precision, recall)
- Organizational taxonomies (using search trees)
- Relationships between components

The classification process enhances the users' ability to organize, describe, discriminate, browse, search, understand, evaluate, retrieve, extract, adapt, and apply.

InQuisiX is equipped with a flexible set of cataloguing facilities designed to minimize the effort associated with the entry of component information. For example, attribute information can be entered with user-defined layouts of forms; bulk text and graphic attributes remain externally stored and may be indexed by the reuse library system; users may integrate custom tools for automated derivation and cataloguing of components attributes; custom migration tools may move component information from existing libraries into InQuisiX; and components may be moved between libraries using the import/export facility.

InQuisiX provides effective searching by offering an array of different search facilities directed by the user. These search facilities include iterative querying, free text associative query, Boolean query, form-based query with user-defined layouts, wildcard text matching, and powerful browsing facilities.

A product such as InQuisiX offers valuable potential as part of an automated system engineering environment. As of June 1992, InQuisiX is nearing the end of beta testing, and will be commercially available from SPS/SAIC before the end of 1993.

4.7. Momenta Pen-Based Computer

Momenta Corporation is Silicon Valleys' newest glamour company. Founded in September 1989 with $1 million in capital and ten initial

employees, it grew to 100 employees by August 1991. With 30 million dollars in startup and an all-star management team, Momenta launched its product in October 1991 [MAG91]. Momenta has 11 major investors located in the U.S. Singapore, Taiwan and Japan. The company is headed by Silicon Valley veteran Kamran Elahian, founder of Cirrus Logic and CAE Systems.

The Momenta Computer is controlled via a pen-like stylus which can be used, like a mouse, to issue commands, draw graphics, and select text. The pen can also be used to enter data. Momenta was designed to bridge the gap between pen-only systems and keyboard-only desktop and laptop computers, hence the term "pentop." Also, the machine can be trained to recognize neatly-printed block letters. No portable machine can yet recognize hand-writing. You can use the pen to write in cursive style, but the software won't recognize the actual letters. It will, however, store handwritten notes for later reference.

The machine is a 11 3/4-inch-by-10 1/2-inch unit which contains all the electronics, including a 40 megabyte hard disk, 4 megabytes of RAM, and a megabyte of ROM, plus the connectors and the screen. The unit is shaped like a wedge, rising from about 1 1/4 inches at the front to 2 1/2 inches at the back. The top surface is the screen, and the slope makes a nice angle for writing with the pen. The screen is hinged along the front edge, and the screen can be tilted up to a convenient viewing angle.

The screen itself is an 8-by-6-inch VGA (video graphics array) reflective liquid crystal display, with black characters on a yellowish background. One slider switch on the right edge controls brightness; another controls the volume of a small built-in speaker. The display is not illuminated, which gives the advantage of portability - it will run for six to eight hours on 10 standard AA batteries. Momenta promises a back-lit model in 1992 to improve the readability of the screen in ambient light [SHA91].

Unlike some other pen-based systems, the Momenta Computer comes with a plug-in keyboard. The company is quick to admit that the keyboard is better for entering large amounts of text. A recent update to the Momenta operating system has enhanced and fine tuned the touch and efficiency of the pen interface, however. In addition to the external keyboard, the Momenta Pen-Based Computer is able to display an on-screen image of a keyboard. The stylus can be used like fingers of your hands to "press" keys on this on-screen or "soft keyboard" [OCO91].

The Momenta interface is simple, graphic, and customizable. The user can arrange the screen icons to suit his way of working. The user can add new icons to symbolize new applications. Default icons include a calendar,

notetaker, address book, memo, presenter, PenCell, In/Out, Mark-up, handprinting trainer, quick handprinting editor, DOS to Momenta, paper tray, and software catalog.

The machine is designed for executives, professionals, sales people, journalists and other white-collar productivity workers who spend a great deal of their time in meetings or on-the-move. Notebook and laptop computers are fine for working in an office, but they are too obtrusive for meetings. Keyboard sounds stop the flow in information in a meeting, and laptop screens block part of the view. The Momenta is used as pen and paper.

Momenta has a built-in data modem for instant access to remote databases, networks, or electronic mail systems, and FAX machines. Momenta transmits and receives at 9,600 bits per second, and the modem operates at 2,400 bits per second and meets the new MNP 5 and V.42 communications standards. The FAXing process is paperless with Momenta since the FAX appears as an electronic image on the screen. Using a "mark-up" application, the FAX can be edited by writing your comments directly on the screen. When finished, the user can ship the revised FAX back to the sender or to someone else, without the transfer of paper. Momenta will also create a coversheet, using the address book to fill in the details. Prepared FAXes can be stored in an electronic "in/out box" and then transmitted later. By transmitting the FAX image to yourself to a nearby FAX machine, a hard copy is available with the same quality as a laser printer.

Momenta can also go wireless if a phone line is not available, Momenta hooks to a portable phone via a cellular connection cable. Momenta features built-in support for the new SPCL protocol, which compresses data and saves money on the transmission.

The Momenta runs on standard MS-DOS 5.0 and Windows 3.0 software on a 386X computer at 20Hz. Momenta has developed an interface that allows the use of the pen to issue commands and enter data even on DOS programs not designed for a pen. The pen can also be used in place of a mouse. Rather than just serve as a pen-based machine for DOS software, Momenta has taken great strides to develop its own "pencentric" operating environment called the Momenta Software Environment (MSE). This environment which runs on top of MS-DOS, provides a user interface especially designed for pen-based programs [REI91].

Bundled programs with a Momenta machine include a basic word processor, a spreadsheet, a charting program, a presentation graphics package, a calendar and an address book. Unlike a traditional drawing package, the presentation software, called Presentor, doesn't require pre-selection of the type of object

created. Instead, it automatically converts crude hand drawings into perfect line, circles, rectangles or squares.

Programs designed specifically for the machine are written using the Momenta Application Development Environment (MADE), a custom implementation of Smalltalk's object-oriented programming environment. Momenta's multitasking, object-oriented environment provides shared objects such as printing, scrolling and text editing. This conserves memory and disk space as well as software development time.

The Momenta Software Environment also offers a unique "pencentric" interface, called Command Compass. Command Compass gives the user the ability to manipulate or modify information by striking the pen in a specified direction on the screen. The Command Compass, a circular menu of command options, eliminates the hand and eye movements necessary when using pull-down menus that might be located far from the object that the user is trying to manipulate. Instead, the Command Compass is located on the object being manipulated. To move an object, you move the pen to invoke the move command and then move the object to the location where you want it. The command and the action itself are part of the same pen movement, reducing hand and eye movement.

At the moment, unfortunate problems concerning managerial and business difficulties are shadowing the success of Momenta.

4.8. Parametric Review of Information for Cost and Evaluation (PRICE)

Parametric Review of Information for Cost and Evaluation (PRICE) is a family of fully supported parametric models that are applicable to cost and schedule estimating requirements for hardware and software. Four models in the PRICE family are available: PRICE H™, PRICE HL™, PRICE M™, and PRICE S™. PRICE H estimates costs and schedules for the development and production of hardware products and systems. PRICE HL estimates maintenance and support costs of hardware products and systems. PRICE M estimates development and production costs and schedules of microcircuit components (chips) and electronic assemblies (boards). PRICE S estimates costs and schedules for the development and support of computer software.

PRICE models use a parametric approach to cost estimating. Parametric cost modeling is based on cost estimating relationships (CERs) that make use of product characteristics (such as hardware weight and software language) to estimate costs and schedules. The CERs in the PRICE models have been determined by statistically analyzing thousands of completed projects where

the product characteristics and project costs and schedules are known. These projects encompass a wide range of products and companies, in the U.S. and abroad. As a result, the PRICE models contain hundreds of CERs that work together to compute the costs and schedules for developing hardware and software products. Parametric techniques have wide business applications, and are a proven method of providing fair and reasonable cost estimates.

The parametric models were designed to produce cost estimates for projects in a fraction of the time required for a conventional estimate. Since the product characteristics can be defined long before the material and labor requirements, the PRICE models permit accurate estimates to be obtained early-on. Early management and engineering decisions can be based on these realistic estimates.

With PRICE, the time and labor required for cost estimating is reduced, and pre-development cost and schedule can be estimated. PRICE users obtain required cost estimates in one-tenth to one-twentieth of the time required for a conventional estimate. Cost iteration for the same product can be obtained in minutes using PRICE. Cost risks associated with uncertainties in the project can be explored (i.e., "what-if" questions).

This early-on estimating capability makes PRICE an excellent tool for design engineers, since much of a project cost is locked in by early engineering decisions. PRICE can provide engineers with the cost information needed to develop minimum-cost designs.

Other benefits of PRICE are found in day-to-day business operations and in maintaining the competitive edge:

- Developing auditable, repeatable estimates
- Establishing cost targets
- Performing cost/scheduling tradeoff studies
- Establishing work center budgets
- Developing estimate-to-complete
- Supporting bid/no-bid decisions
- Supporting subcontractor/vendor negotiations
- Making teaming choices
- Measuring new technology insertion costs
- Analyzing competitor product costs

Use of PRICE parametric models benefits management, engineering, manufacturing, and finance.

Since 1975, PRICE Systems has been the international standard for cost and schedule estimating models and services, designed by professionals for use by professionals. PRICE Systems maintains a technical operations staff at seven U.S. and international locations dedicated to supporting the PRICE user. The PRICE Systems Headquarters is located at Moorestown, NJ and is manned twenty-four hours a day, every day. At any time, should a user need assistance, a member of the technical operations staff will respond to the requirement.

5. Prototypes

The motivation for developing prototypes was to demonstrate an automated system engineering environment. The development of the prototypes helped to reduce program risk, demonstrate environment synthesis from building blocks, demonstrate multi-discipline concurrent engineering support and experiment with key user interactions. Rather than attempting to provide comprehensive coverage for automation of all system engineering activities, the SECD Team chose to focus on automation of high payoff activities.

The SECD Team chose three scenarios that most adequately represented critical activities in the system engineering process:

- Requirements Flowdown Scenario
- Tradeoff Scenario
- Timeline Scenario

The test set for the Requirements Flowdown Scenario was an integration of the Global Positioning System (GPS) for the upgrade of the A-10 Aircraft provided by Frank LaMonica, RL. The Test set for the Tradeoff Scenario was an architecture tradeoff for the Air Transport Theater/ Special Operation Aircraft (ATT/SOA) provided by Alberto Ortiz, McDonnell Douglas, Douglas Aircraft Corporation. The test set for the Timeline Scenario was the method to define the sequence of an air /sea battle attack, allocation to systems and subsystems, and define system loading provided by Hank Stuebing and Rich Pariseau, Naval Air Warfare Center (NAWC).

These three scenarios automated the more generic systems engineering functions and excluded activities that are specific to the type of application, the level of system being engineered, and the organization of the methods being applied.

The demonstration testbed for the prototype consisted of the following hardware and software:

- (2) Macintosh II fx, personal computers representing the desktop systems engineering workstations, running System 7 Software to provide the state-of-the-art operating systems for the Macintosh
- One Pentop Notebook, Momenta 386SX, a pen-based computer, for use in meetings, on travel, or for information

- One 386 PC, a personal computer networked to the Macintoshes and the Momenta to demonstrate a heterogeneous environment
- (2) Voice Navigator II and Voice Record to demonstrate voice record and voice commands with a desktop and a headset microphone
- MacFlow, MacFlow Symbol Sampler & MacFlow More Translator for process modeling
- MacSchedule Plus1.0.4 for project scheduling
- MacDraw Pro for graphics
- MacProject II 2.5 for project planning
- MORE 3.0 for outlining
- Microsoft Office 1.5, includes word processing and spreadsheet facilities
- MathCAD for the Macintosh for mathematical engineering computations
- Meeting Maker for meeting scheduling
- Retrospect Remote for remote system backup
- MiniCad+ Software for engineering drawings
- Microsoft Mail for electronic mail
- FileMaker Pro for flat records of data
- Remember for personal calendars and reminders
- Power Windows Software for layout of screens
- UserLand Frontier v1.0 for scripting of scenarios

Figure 5-1 illustrates the configuration of the testbed for the prototype scenarios.

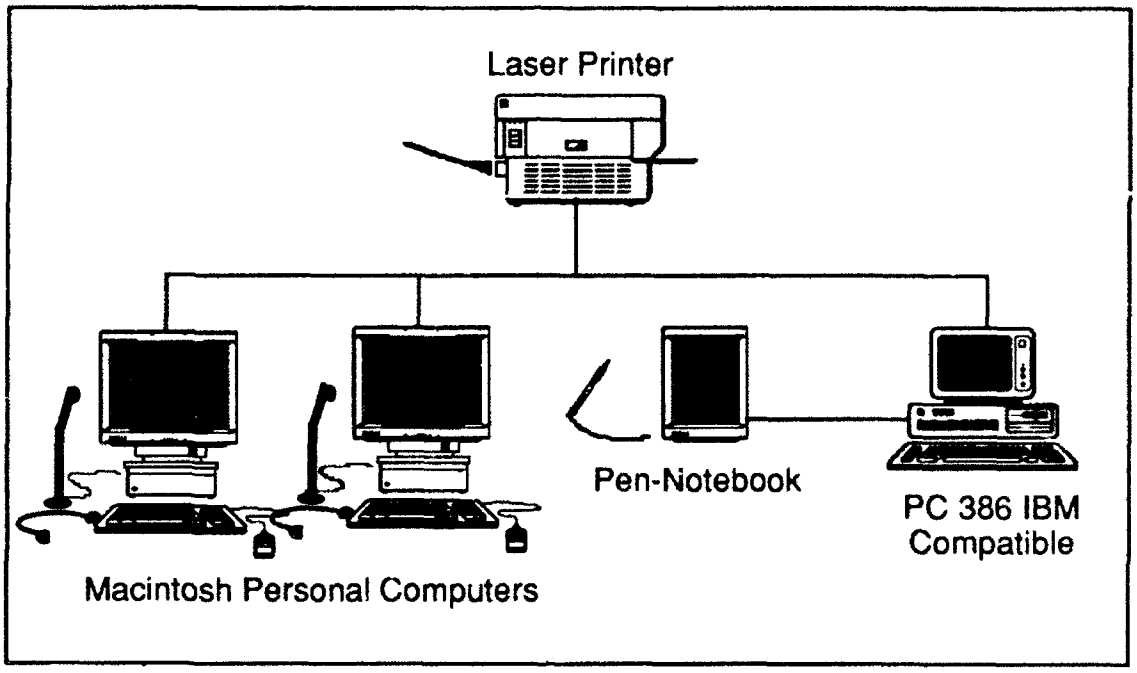


Figure 5-1. Testbed Configuration for the Scenarios

The hardware and software selected provided support in the areas of engineering, management, and communication as specified in the operational concept and requirements of an automated system engineering environment. Selection of the hardware and software was made by researching the needs, analyzing the features, ease of use, availability, and cost of each item purchased.

5.1. Requirements Flowdown

In performing a flowdown of requirements, the Systems Engineer is faced with managing complex information during iterative analysis, partitioning and allocation of requirements. The Systems Engineer must derive the system requirements from the mission requirements to develop a system specification while maintaining consistency and completeness. The rationale for this decision process may not be documented, and consequently, is lost for future reference. Additionally, if a requirement must be changed, the impact of that change on other requirements must be analyzed before a change decision is made. The change must be documented and work products, budgets and schedule must be revised. Automating this process in an integrated development environment is critically needed to improve the productivity and quality of system engineering of complex systems.

The following concepts were demonstrated in the Requirements Flowdown:

- Tracing and tracking mission requirements to system requirements to system design (partitioning) to subsystem requirements allocation
- Networking, communication, and data exchange in a heterogeneous, distributed computing environment (Macintosh, Pen notebook computer, and 386 PC)
- Cooperative integration of COTS tools with synthesis of information from multiple sources
- Analysis of the impact of changes to a requirement

The development of the details of the scenario was driven by the following Catalyst requirements:

- Partitioning, Allocation and Traceability Support, SYSREQ - 108
- Work Management Support, SYSREQ - 115
- Work Product Development Support, SYSREQ - 110
- Information Management Support, SYSREQ - 111
- Information Formalization Support, SYSREQ - 104
- Decision Making Support, SYSREQ - 107
- Impact Analysis Support, SYSREQ - 109
- Versioning and Baseline Support, SYSREQ - 112
- Information Dissemination and Communication Support, SYSREQ - 114

The activities of the Requirements Flowdown Scenario were divided into Acts and Scenes. The players of the Requirements Flowdown Scenario were a Systems Engineer, an Electrical Engineer, a Software Engineer and a Program Manager. An overview of the storyline of the Requirements Flowdown Scenario was as follows:

- Act I, Scene 1 - The Systems Engineer on the Macintosh workstation

  Initialize the process of deriving the mission requirements by parsing the mission statement

- Act I, Scene 2 - The Systems Engineer on the Pen-Based Computer

  Edit the mission statement and annotate, sketch the system architecture and formalize the System Block Diagram

- Act I, Scene 3 - The Systems Engineer on the Macintosh workstation

Download the System Block Diagram form the Pen-Based Computer to the Macintosh workstation and place in the System Specification

- Act II - The Systems Engineer's Workstation

  Derive a set of specifications from a mission statement to develop a System Specification, use multiple views of the data for his analysis

- Act III, Scene 1 - The Systems Engineer on the Macintosh workstation

  Finalize his draft of the System Specification and routes to his team members.

- Act III, Scene 2 - The Electrical Engineer's Workstation

  Receives the communication and electronic document, makes his suggested corrections, and forwards, with a message, to the Systems Engineer

- Act III, Scene 3 - The Systems Engineer's Workstation

  Monitor the progress of the comments to the System Specification, view annotations by the Electrical Engineer, the Software Engineer, and the Program Manager, and resolve the differences. The updated System Specification is distributed to the Project Team.

- Act IV, Scene 1 - The Navigation Specialist's Workstation

  Discovers a requirement that is not "do-able" by his mathematical analysis using a COTS tool integrated into the environment, notifies the Systems Engineer

- Act IV, Scene 2 - The Systems Engineer's Workstation

  Views the analysis of the Navigation Specialist, and earmarks the requirements that must be changed. Engineering Change Proposal is generated by the environment, and change is committed, all project documents updated.

The following steps defined the activities in the requirements flowdown scenario:

1. The scenario, background and data set were introduced.
2. A scanned image of the original mission requirements was viewed and organized into specific requirements in the database.
3. Using an outliner, the systems engineer elaborated the mission requirements and organized them into a set of system requirements reflected in the database.

4. Conferring verbally with members of the systems engineering team, the system engineer sketched the system architecture on a pen-notebook computer and later downloaded this data to his desktop workstation.
6. The database was updated with the system partitioning and the systems engineer allocated the requirements to the various subsystems.
7. Responding to a change in mission requirements, the systems engineer uses the automated environment to identify the impacted requirements and subsystems.

As evidenced from the demonstration of the Requirements Flowdown Scenario, an environment to automate these activities supports consistency and completeness to identify, partition and allocate requirements, analyze impacts, as well as provide version control of documents and project information after changes have been committed. Automating laborious, yet critical activities of the requirements flowdown process, would help to improve the quality of systems engineering in a concurrent environment.

5.2. Tradeoff Scenario

In performing a tradeoff, systems engineers and specialty engineers concurrently analyze alternatives against specific criteria spanning multiple engineering disciplines. After their analysis, the decisions, both technical and programmatic, must be clearly communicated. Currently, this process may be ad hoc, and capturing the history and rationale of the decisions is often lost and difficult to recreate.

To address the issues and needs for automating the activities of a tradeoff, the following concepts were demonstrated in the Tradeoff Scenario:

- Provide an automated tradeoff harness for alternative analysis
- Provide a standard framework for evaluating alternatives against chosen criteria
- Integrate, link and streamline the activities of a tradeoff
- Support tasking of individuals for multi-disciplinary analysis
- Provide an interface to external analysis tools and mathematical solver tools and receive data from them
- Synthesize and display analysis for various alternatives to support decision making

- Record the alternatives, results, decisions and rationale for later reference
- Maintain traceability of decisions to the system definition
- Support reviewing and revisiting tradeoff results further down stream
- Use voice annotation (during tasking) and voice commanding (to view analysis results), improving productivity and reducing labor costs

The development of the details of the scenario was driven by the following Catalyst requirements:

- Analysis Support, SYSREQ - 106
- Tasking Support, SYSREQ - 102
- Monitoring Support, SYSREQ - 103
- Decision Making Support, SYSREQ - 107
- Partitioning, Allocation, and Traceability Support, SYSREQ - 108
- Information Dissemination & Communication Support, SYSREQ - 114
- Electronic Meeting Support, SYSREQ - 116
- Planning and Tracking Support, SYSREQ - 101
- Information Formalization Support, SYSREQ - 104
- Work Management Support, SYSREQ - 115

The activities of the Tradeoff Scenario were divided into Acts and Scenes, much like a play. The players of the Tradeoff Scenario were a Systems Engineer and two Specialty Engineers, an Avionics Engineer and a Cost Analyst. An overview of the storyline of the Tradeoff Scenario was as follows:

- Act I, Scene 1 - Systems Engineer's Workstation

  Define a new tradeoff, create a workspace, browse references, partition tradeoff, assign tasks, communicate assignment

- Act I, Scene 2 - Specialty Engineer's Workstation

  Management of work priorities, analyze using specialty tools, communicate task completed

- Act II, Scene 1 - Systems Engineer's Workstation

Monitor work flow, browse interim work products, assign tasks, communicate assignment

- Act II, Scene 2 - Another Specialty Engineer's Workstation

  Management of work priorities, browse interim work products, analyze using specialty tools, communicate task completed

- Act III - Systems Engineer's Workstation

  Monitor work flow, communicate task completion, capture decision and rationale, schedule review of decision

The following steps defined the development of specific activities in the Tradeoff Scenario:

1. The tradeoff case, background, and data set was introduced.
2. Using various textual and graphic tools, the systems engineer documented the various alternatives to be considered.
3. Using a spreadsheet, the systems engineer defined the various numeric criteria to be analyzed.
4. The systems engineer tasked various members of the systems engineering team to derive specific data for the tradeoff.
5. Via electronic mail, the various members received their taskings, supporting data with voice annotation from the systems engineer.
6. Each member of the team, utilizing existing tools and models, derived their respective data.
7. The systems engineer electronically received the inputs and synthesized them into the original spreadsheet of criteria.
8. The systems engineer viewed graphs and overlays of the data to make a decision, commanding the computer by voice to show different views.
9. The systems engineer made the decision, captured rationale by voice, and distributed the decision electronically to the various team members.

As evidenced from the demonstration of the Tradeoff Scenario, an environment which automates these activities would have the following benefits:

- Solidify operational concept and user interactions through demonstration, reducing program risk
- Use of emerging technologies and standards

- Support for multi-discipline, concurrent engineering support
- Synthesis from environment building blocks
- Increased productivity and reduced cost through integration

5.3 Timeline Scenario

The prototype scenario demonstrates the operational concept of Catalyst in performing a timeline analysis. Typically, systems engineers and specialty engineers conduct timeline analyses using pencil and paper. However, the following problems are encountered using manual methods:

- Methods for analysis are often informal or proprietary and the use of a COTS tool is not feasible.
- Relationships of data and time constraints are difficult to track.
- Changes to data and time constraints are performed manually and are laborious and error-prone.
- Implementation of the method is often a group activity over several months.
- Methods often span enginering of multiple systems and subsystems.

The timeline scenario demonstrated that Catalyst is a solution to these existing problems. As indicated by the three shaded areas in Figure 5.3-1, there are general purpose tools to support informal methods and special purpose tools to support formalized methods.

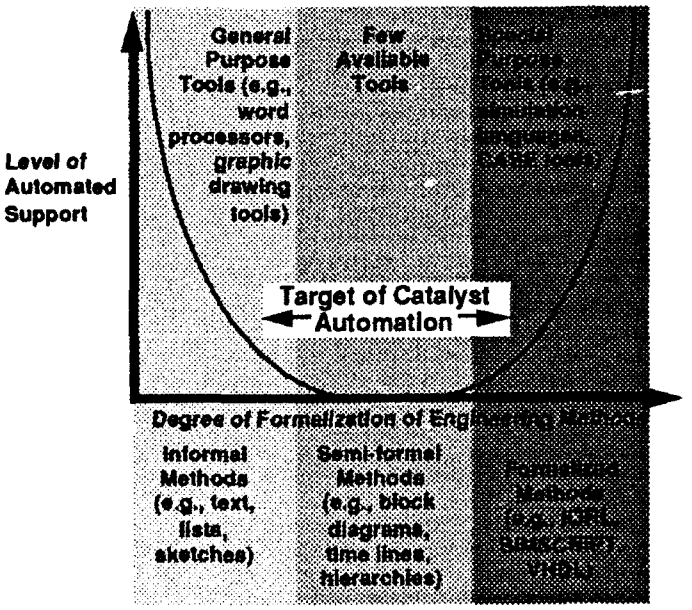


Figure 5.3-1 Catalyst Automation Target

However, there is a large gap between these two categories of tools . Few tools are available to support semi-formal methods (block diagrams, timelines and hierarchies). This gap is the target area of Catalyst.

Catalyst has the following features to support solving the types of problems listed above:

- Integrates into an existing environment
- Manages complex information and provides process guidance
- Provides a visual user interface to define models
- Presents unified views of information
- Uses meta-tool technology to provide implementations of many methods (i.e., the timeline is one of many possible implementions of the meta-tool)

InSight, a meta-tool in development by SPS, was added to the existing prototype testbed to demonstrate this scenario. The meta-tool was customized to automate the method used at NAWC. Objects that make up the method were shown, constraint language method rules that define the method semantics were browsed, and the user interface was customized to the user's method.

The development of the details of the scenario was also driven by the following Catalyst requirements:

- Planning and Tracking Support, SYSREQ - 101
- Information Formalization Support, SYSREQ - 104
- Work Product Development Support, SYSREQ - 110
- Modeling Support, SYSREQ - 105
- Analysis Support, SYSREQ - 106
- Decision Making Support, SYSREQ - 107
- Partitioning, Allocation and Traceability Support, SYSREQ - 108
- Information Management Support, SYSREQ - 111
- Versioning and Baseline Support, SYSREQ - 112

The activities of the Timeline Scenario were divided into Acts and Scenes. The players of the Timeline Scenario were a Systems Engineer, a Specialty

Engineer, and an Operational Analyst. An overview of the storyline of the Timeline Scenario was as follows:

- Act I - The Specialty Engineer's Workstation

  Define the Air/Sea Battle Attack

- Act II - The Systems Engineer's Workstation

  Formulate Conceptual Design of Battle, identify functions and allocate to subsystems

- Act III - The Systems Engineer's Workstation

  Define System Loading, visually execute timeline with selected systems in a degraded mode

The following steps defined the development of specific activities in the Timeline Scenario:

1. The scenario and the data set is introduced to show the basis for the timeline analysis
2. Using the meta-tool, the Specialty Engineer instantiates existing objects and creates new objects in a origin-based grid. Attributes and constraints are added to objects with a intuitive, user-friendly user interface.
3. Using the meta-tool, the Systems Engineer elaborates the functions that must be accomplished within the timeline in the form of functional flow diagrams.
4. The Systems Engineer electronically illustrates this functional analysis to other members of the systems engineer team.
5. Using the meta-tool, the Systems Engineer derives a conceptual design and allocates functions to the subsystems.
6. Using the meta-tool, the Systems Engineer leads a joint, distributed development of loading snapshots as elaborations of the timeline, functional flow and conceptual design.

As evidenced from the demonstration of the Timeline Scenario, an environment to automate these activities has the following benefits:

- Automated control of sequencing information within a concurrent engineering environment
- Knowledgeable environment for functional analysis and allocation of requirements

- Maintenance of system subsystem details and requirements when operating in a degraded mode

In summary, the feasibility, usability, and marketability of Catalyst has been fortified by the various trade-off studies and analyses, and the prototype concept and technology demonstrations that were performed.

6. Risk Evaluation

6.1 Risk Identification

We have identified the following risks for the *Catalyst* environment:

- **Scope** - Attempting to address a scope that is either too broad or too ill-defined is a risk to the successful specification and later implementation of the environment.
- **Usability** - The 1997 environment must be highly usable to insure technology transfer and marketability. The two key factors that contribute to usability are environment capabilities and user interface desirability and performance.
- **Performance** - Performance is the key to the usability and acceptance of the completed environment. However, performance requirements which are too stringent will add significant risk to the development project.
- **Degree of adaptability** - The environment must be sufficiently adaptable to successfully automate an organizations' existing processes. There is a high degree of variability in the roles, methods, work products and work flows that must be accommodated.
- **Scalability** - The environment must be able to automate the development of systems involving hundreds or even thousand of personnel. Current environments have had significant problems in scaling up to support very large developments.
- **Feasibility** - Implementing the specified environment by 1997 is a risk that should be tracked closely. It is a trade-off against incorporating both "cutting edge" and state-of-the-art technology.
- **Maintainability** - Our experience indicates that we must insure that the system engineering environment is easy to maintain and will suffer minimum impact by COTS component upgrades. For example, one of the problems in marketing SLCSE is its maintainability. SLCSE is tightly integrated with a host of COTS software products (e.g., VMS, DECNET, SMARTSTAR, RDB/Sharebase). Any time one of the COTS components is upgraded, many hours of maintenance may be required to make SLCSE functional again.

6.2 Risk Analysis and Abatement Strategies

Prudent planning and management dictates that potential problems be identified as early as possible and alternatives be enumerated to reduce the related program risk. The following analysis was accomplished according to the Air Force Systems Command and Air Force Logistics Command entitled, "Acquisition Management Software Risk Abatement," AFSC/AFLCP 800-45.

This approach to risk analysis identifies four risk areas: 1) Performance, 2) Support, 3) Cost, and 4) Schedule. The analysis process is targeted to software development. Support risks were not analyzed since they largely depended on a yet to be determined support concept and the capabilities of the development contractor.

As described in AFSCP/AFLCP 800-45, there are several methods for evaluating the risk of a given area. Risk can be determined by (1) taking the average value (2) incorporating weighting factors or (3) taking the driver with the highest probability of occurrence values and using that as the overall value. For this risk analysis process, we shall use the last option, the highest value. This value is then mapped into a summary risk assessment matrix which indexes the probability of occurrence along with the impact to identify an overall risk value of high, moderate, low, or none.

Performance Risk Assessment Table

PERFORMANCE DRIVERS	Impossible to Improbable (0 or 1)	Probable (2)	Frequent (3)	Value
REQUIREMENTS				
Complexity	Simple or easily allocatable	Moderate, can be allocated	Significant or difficult to allocate	3
Size	Small or easily broken down into work units	Medium, or can be broken down into work units	Large, cannot be broken down into work units	2
Stability	Little or no change to established baseline	Some change in baseline expected	Rapidly changing or no baseline	1
PDSS	Agreed to support concept	Roles and missions issues unresolved	No support concept or major unresolved issues	1
R&M	Allocatable to hardware and software components	Requirements can be defined	Can only be addressed at the total system level	1

PERFORMANCE DRIVERS	Impossible to Improbable (0 or 1)	Probable (2)	Frequent (3)	Value
CONSTRAINTS				
Computer Resources	Mature, growth capacity within design, flexible	Available, some growth capacity	New development, no growth capacity, inflexible	N/A
Personnel	Available, in place, experienced, stable	Available, but not in place, some experience	High turnover, little or no experience, not available	N/A
Standards	Appro-priately tailored for application	Some tailoring, all not reviewed for applicability	No tailoring, none applied to the contract	N/A
GDE/GFP	Meets requirements, available	May meet requirements, uncertain availability	Incompatible with system requirements, unavailable	N/A
Environment	Little or no impact on design	Some impact on design	Major impact on design	2
Performance Envelopes	Operation well within boundaries	Occasional operation at boundaries	Continuous operation at boundaries	2

PERFORMANCE DRIVERS	Impossible to Improbable (0 or 1)	Probable (2)	Frequent (3)	Value
TECHNOLOGY				
Language	Mature, approved HOL used	Approved or non-approved HOL	Significant use of assembly language	1
Hardware	Mature, available	Some development or available	Totally new development	1
Tools	Documented, validated, in place	Available, validated, some development	Unvalidated, proprietary, major development	N/A
Data Rights	Fully compatible with support and follow-on	Minor incompat-ibilities with support and follow-on	Incompatible with support and follow-on	1
Experience	Greater than 4 years	Less than 4 years	Little or none	N/A
DEVELOPMENT APPROACH				
Prototypes and Reuse	Used, documented sufficiently for use	Some use and documentation	No use or documentation	0
Documentation	Correct and available	Some deficiencies, available	Non-existent	0
Environment	In place, validated, experience with use	Minor modifications, tools available	Major development effort	N/A
Management Approach	Existing product and process controls	Product and process controls need enhancement	Weak or non-existent	N/A
Integration	Internal and external controls in place	Internal or external controls not in place	Weak or non-existent	N/A

Cost Risk Assessment Table

COST DRIVERS	Impossible to Improbable (0 or 1)	Probable (2)	Frequent (3)	Value
REQUIREMENTS				
Size	Small, noncomplex, or easily decomposed	Medium, moderate complexity, decomposable	Large, highly complex, or not decomposable	2
Resource Constraints	Little or no hardware-imposed constraints	Some hardware imposed constraints	Significant hardware-imposed constraints	2
Application	Non-real-time, little system interdepen-dency	Embedded, some system interdepen-dency	Real-time, embedded, strong interdependency	2
Technology	Mature, existent, in-house experience	Existent, some in-house experience	New or new application, little experience.	2
Requirements Stability	Little or no change to established baseline	Some change in baseline expected	Rapidly changing or no baseline.	1

COST DRIVERS	Impossible to Improbable (0 or 1)	Probable (2)	Frequent (3)	Value
PERSONNEL				
Availability	In place, little turnover expected	Available, some turnover expected	High turnover, not available	N/A
Mix	Good mix of software disciplines	Some disciplines inappropriatel y represented	Some disciplines not represented	N/A
Experience	High experience ratio	Average experience ration	Low experience ratio	N/A
Management Environment	Strong personnel management approach	Good personnel management approach	Weak personnel management approach	N/A

COST DRIVERS	Impossible to Improbable (0 or 1)	Probable (2)	Frequent (3)	Value
REUSABLE SOFTWARE				
Availability	Compatible with need dates	Delivery dates in question	Incompatible with need dates	N/A
Modifications	Little or no change	Some change	Extensive changes	N/A
Language	Compatible with system & PDSS requirements	Partial compatibility with requirements	Incompatible with system or PDSS requirements	0
Rights	Compatible with PDSS & competition requirements	Partial compatibility with PDSS, some competition	Incompatible with PDSS concept, noncompetitive	1
Certification	Verified performance, application compatible	Some application compatible test data available.	Unverified, little test data available	N/A
TOOLS AND ENVIRONMENT				
Facilities	Little or no modification	Some modifications, existent	Major modifications, non-existent	N/A
Availability	In place, meets need dates	Some compatibility with need dates	Nonexistent, does not meet need dates	N/A
Rights	Compatible with PDSS & development plans	Partial compatibility with PDSS & development plans	Incompatible with PDSS & development plans	1
Configuration Management	Fully controlled	Some controls	No controls	N/A

Schedule Risk Assessment Table

SCHEDULE DRIVERS	Impossible to Improbable (0 or 1)	Probable (2)	Frequent (3)	Value
RESOURCES				
Personnel	Good discipline mix in place	Some disciplines not available	Questionable mix and/or availability	N/A
Facilities	Existent, little or no modification	Existent, some modification	Nonexistent, extensive changes	N/A
Financial	Sufficient budget allocated	Some questionable allocations	Budget allocation in doubt	3
NEED DATES				
Threat	Verified projections	Some unstable aspects	Rapidly changing	N/A
Economic	Stable commitments	Some uncertain commitments	Unstable, fluctuating commitments	3
Political	Little projected sensitivity	Some limited sensitivity	Extreme sensitivity	1
GFE/GFP	Available, certified	Certification or delivery questions	Unavailable and/or uncertified	N/A
Tools	In place, available	Some deliveries in question	Uncertain delivery dates	N/A
TECHNOLOGY				
Availability	In plane	Some aspects still in development	Totally still in development	2
Maturity	Application verified	Some applications verified	No application evidence	2
Experience	Extensive applications	Some applications	Little or none	N/A

SCHEDULE DRIVERS	Impossible to improbable (0 or 1)	Probable (2)	Frequent (3)	Value
TOOLS AND ENVIRONMENT				
Definition	Known, baselined	Baselined, some unknowns	Unknown, no baseline	N/A
Stability	Little or no change projected	Controllable change projected	Rapid or uncontrollable change	N/A
Complexity	Compatible with existing technology	Some dependency on new technology	Incompatible with existing technology	1

In summary, the risk weights as defined by AFSCP/AFLCP 800-45 have the following values:

Risks	Weight
Performance	13
Schedule	12
Cost	9

Drivers for performance risk are complex requirements and allocations, large size environment, though with manageable subsystems, adaptability to the users' environment significantly impacts the design, environment must perform when stressed with large applications and many users. Drivers for schedule risk are doubtful budget allocation and commitment, and availability and stability of some technologies. Drivers of cost risk are the large size of the environment, some hardware-imposed constraints, some system dependencies, and some required emerging technologies.

Primary risks are the size and complexity of the environment, adaptability of the environment, dependence on some emerging technologies, and questionable sufficient and stable budgets.

Usability is also a risk. Catalyst must be highly usable to insure technology transfer and marketability. Catalyst's primary users are only marginally

computer literate. New user interface paradigms and technologies are being applied. Performance will greatly impact usability.

The following risk abatement strategies are recommended:

- An object-oriented architecture will reduce size and allow adaptability to be accommodated.
- Prototyping and incremental development will address technology, performance and usability risks.
- Maximum use of commercial-off-the-shelf solutions will minimize development budget requirements and focus attention on difficult technical issues
- Building-block approach will allow development flexibility to accommodate funding constraints.

REFERENCES

[ACL88] Acly, E. "Looking Beyond CASE." *IEEE Software* Mar. 1988: 39-43.

[AMS76] Amster, S., et al. "An Experiment in Automatic Quality Evaluation of Software." *Proceedings of the Symposium on Computer Software Engineering* 1976: 171-97.

[ARN82] Arnold, R.S. and D.A. Parker. "The Dimensions of Healthy Maintenance." *Proceedings of the 6th International Conference on Software Engineering* 1982.

[BAH86] Bahlke, R. and G. Snelting. "The PSG System: From Formal Language Definitions to Interactive Programming Environments." *ACM Transactions on Programming Languages* Oct. 1986: 547-76.

[BAI81] Bailey, C. and W. Dingee. "A Software Study Using Halstead Metrics." *ACM SIGMetrics Performance Evaluation Review* Spring 1981: 189-97.

[BAK79] Baker, A. and S. Zweben. "The Use of Software Science in Evaluating Modularity Concepts." *IEEE Transactions on Software Engineering* Mar. 1979: 110-20.

[BAK80] Baker, A. and S. Zweben. "A Comparison of Measures of Control Flow Complexity." *IEEE Transactions on Software Engineering* Nov. 1980: 506-11.[BOO83] Booch G., *Software Engineering with Ada,* Benjamin Cummings, 1983.

[BAN81a] Bandyopadhyay, S. "A Study on Program Level Dependency of Implemented Algorithms on Its Potential Operands." *ACM SIGPLAN Notices* Feb. 1981: 18-25.

[BAN81b] Bandyopadhyay, S. "Theoretical Relationships Between Potential Operands and Basic Measurable Properties of Algorithm Structure." *ACM SIGPLAN Notices* Feb. 1981: 26-34.

[BAS81] Basili, V. and T. Phillips. "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory." *ACM SIGMETRICS Performance Evaluation Review* Spring 1981: 95-99.

[BER80] Berlinger, E. "An Information Theory Based Complexity Measure." *Proceedings of the National Computer Conference* 1980: 773-79.

[BES82] Beser, N. "Foundations and Experiments in Software Science." *ACM SIGMETRICS Performance Evaluation Review* Fall 1982: 48-72.

[BOE81] Boehm B.W. *Software Engineering Economics,* Prentice-Hall, Inc., 1981.

[BOH75] Bohrer, R. "Halstead's Criterion and Statistical Algorithms." *Proceedings of the 8th Annual Computer Science/Statistics Interface Symposium* 1975.

[BOO83] Booch G., *Software Engineering with Ada,* Benjamin Cummings, 1983.

[BOU88] Boudier G., Gallo F., Minot R., Thomas M.I., "An Overview of PCTE and PCTE+," *Proceedings of the 3rd ACM Symposium on Software Development Environments,* Boston, Nov., 1988.

[BOW78] Bowen, J. "Are Current Approaches Sufficient for Measuring Software Quality?" *Proceedings of the Software Quality Assureance Workshop* 1978: 148-55.

[BOW79] Bowen, J. "A Survey of Standards and Proposed Metrics for Software Quality Testing." *IEEE Computer* Aug. 1979: 37-42.

[BRA85] Brady, J.T. "A Theory of Productivity in the Creative Process." *Proceedings of the 1st International Conference on Computer Workstations* Nov. 1985: 70-9.

[BUL74] Bulgut, N. and M. Halstead. "Impurities Found in Algorithm Implementations." *ACM SIGPLAN Notices* Mar. 1974.

[BUL85] Bull, et al. *PCTE: A Basis for a Portable Common Tool Environment, (Functional Specifcations)* 2 vols., 3rd ed., 1985.

[BUX79] Buxton, J.N. *Requirements for Ada Language Integrated Computer Environments: Preliminary `Stoneman'* Department of Defense, Washington, D.C. Nov. 1979.

[BUX80] Buxton, J.N. *Requirements for Ada Programming Support Environments: Stoneman* Department of Defense, Washington, D.C. Feb. 1980.

[CAI86] *Common Ada Programming Support Environment (APSE) Interface Set (CAIS)*, DoD Std 1838, Oct. 9, 1986.

[CAI88] *Common Ada Programming Support Environment (APSE) Interface Set (CAIS), Revision A*, Proposed DoD Std. 1838A, May 20, 1988.

[CAN83] Cantone, G., A. Cimitile, and L. Sansome. "Complexity in Program Schemes: The Characteristic Polynomial." *ACM SIGPLAN Notices* Mar. 1983: 22-31.

[CHA85] Charette, B., H. Steubing, and M. Dynum. *Panel 4: Software Engineering Environment Standard Interfaces* 16 Sept. 1985.

[CHA79] Chapin, N. "A Measure of Software Complexity." *Proceedings of the National Computer Conference* 1979: 995-1002.

[CHA83] Chaudhary, B. and H. Sahasrabuddhe. "Two Dimensions of Program Complexity." *International Journal of Man-Machine Studies* 1983: 505-511.

[CHE78] Chen, E. "Program Complexity and Programmer Productivity." *IEEE Transactions on Software Engineering* May 1978: 187-94.

[CHI83] Childs, D., D. Whalen, and L. Keith. *U.S. Army Intelligence Center and School (USAICS) Software Analysis and Management System (USAMS) Evaluation of SREM/REVS/RSL and PSL/PSA* 1983: 89-98.

[CHI87] Chikofsky, E., ed. "The Information Resource Directory System: A Standard Bus to Join the Segmented Environment Together." *Proceedings of the First International Workshop on Computer-Aided Software Engineering* May 1987: 517-22.

[CHR81] Christensen, K., G. Fitsos, and C. Smith. "A Perspective on Software Science." *IMB Systems Journal* 1981: 372-87.

[COO82] Cook, M. "Software Metrics: An Introduction and Annotated Bibliography." *ACM SIGSOFT Software Engineering Notes* Apr. 1982: 41-60.

[COU81] Coulter, N. "Applications of Psychology in Software Science." *COMPSAC Proceedings* 1981: 50-1.

[COU83] Coulter, N. "Software Science and Cognitive Psychology." *IEEE Transactions on Software Engineering* Mar. 1983: 166-71.

[CUR79a] Curtis, B. "In Search of Software Complexity." *Proceedings of the Workshop on Quantitative Models of Software Reliability, Complexity, and Cost* 1979: 95-106.

[CUR79b] Curtis, B., S. Sheppard, and P. Milliman. "Third Time Charm: Stronger Prediction of Programmer Performance by Software Complexity Metrics." *Proceedings of the Fourth International Conference on Software Engineering* 1979: 356-60.

[CUR79c] Curtis, B., et al. "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics." *IEEE Transactions on Software Engineering* Mar. 1979: 95-104.

[DEY79] DeYoung, G. and G. Kampen. "Program Factors as Predictors of Program Readability." *COMPSAC Proceedings* 1979: 668-73.

[DOL87] Dolk, D.R. and R.A. Kirsch II. "A Relational Information Resource Dictionary System." *Communications of the ACM* Jan. 1987: 48-61.

[DON75] Donzeau-Gouge, V., et al. "A Structure-Oriented Progrm Editor." *INRIA Technical Report* 1975.

[DOW87a] Dowson, M. "Integrated Project Support with IStar." *IEEE Software* Nov. 1987: 6-15.

[DOW87b] Dowson, M. "ISTAR and the Contractual Approach." *Proceedings of the Ninth International Conference on Software Engineering* 1987: 287-88.

[DUI86] Duisberg, R.A. "Animated Graphical Interfaces Using Temporal Constraints." *CHI'86 Conference Proceedings* Apr. 1986: 131-36.

[DUN80] Dunsmore, H. and J. Gannon. "Analysis of the Effects of Programming Factors on Programming Effort." *Journal of Systems and Software* 1980 : 141-53.

[DYS87] Dyson, P.B. "Open Architecture for Computer-Aided Software Engineering (CASE) Environments." *Proceedings of the First International Workshop on Computer-Aided Software Engineering* May 1987: 26-8.

[E&V2-84] "Ada Programming Support Environment Evaluation and Validation." *AFWAL/AAAF Workshop Proceedings* Aug. 1984.

[ELL85] Ellis, J.T. *One Approach for Evaluating The Distributed Computing Design System (DCDS)* 25 Sept. 1985.

[ELS76] Elshoff, J. "Measuring Commercial PL/I Programs Using Halstead's Criteria." *ACM SIGPLAN Notices* May 1976: 38-46.

[ELS78] Elshoff, J. "An Investigation Into the Effects of the Counting Method Used on Software Science Measurements." *ACM SIGPLAN Notices* Feb. 1978: 30-45.

[EPS88] Epstein, D. and W.R. LaLonde. "A Smalltalk Window System Based on Constraints." *OOPSLA Conference Proceedings* Nov. 1988: 83-94.

[FITS80] Fitsos, G. "Vocabulary Effects in Software Science." *COMPSAC Proceedings* 1980: 751-56.

[FITZ78] Fitzsimmons, A. and T. Love "A Review and Evaluation of Software Science." *ACM Computing Surveys* Mar. 1978: 3-18.

[FUN76] Funami, Y. and M. Halstead. "A Software Physics Analysis of Akiyama's Debugging Data." *Proceedings of the Symposium on Computer Software Engineering* 1976: 133-38.

[GAF80] Gaffney, J. "Maximize Design Effort and Minimize Program Control Complexity - To Maximize Software Development Productivity." COMPSAC Proceedings 1980: 225-28.

[GAL86] Gallo F., Minot R., Thomas M.I., "The Object Management System of PCTE as a Software Engineering Database Management System," *Proceedings of the Second ACM Symposium on Practical Software Development Environments*, ACM Sigplan Notices, Jan. 1987.

[GAN86] Gannon, J.D., E.E. Katz, and V.R. Basili. "Metrics for Ada Packages: An Initial Study." *Communications of the ACM* July 1986: 616-23.

[GAR84] Garlan, D.B. and P.L. Miller. "GNOME: An Introductory Programming Environment Based on a Family of Structure Editors." *Proceedings of the ACM SIGSOFT/SIGPLAN Software Symposium on Practical Software Development Environments* Apr. 1984: 65-72.

[GIL75] Gilb, T. "Software Metrics - The Emerging Technology." *Data Management* July 1975: 34-7.

[GOL85] Goldfine, A. and P. Konig. *A Technical Overview of the Information Resource Dictionary System* Apr. 1985.

[GOR76] Gordon, R. and M. Halstead. "An Experiment Comparing FORTRAN Programming Times with Software Physics Hypothesis." Proceedings of the National Computer Conference 1976: 935-37.

[GOR79] Gordon, R. "Measuring Improvements in Program Clarity." *IEEE Transactions on Software Engineering* Mar. 1979: 79-90.

[GRA77] Grace, A. "The Dimensions of Complexity." *DATAMATION* Sept. 1977: 315-18.

[GRE76] Green, T., et al. "Program Structures, Complexity, and Error Characteristics." *Proceedings of the Symposium on Computer Software Engineering* 1976: 139-54.

[GRO82] Gross, D., et al. "Complexity Measurement of Electronic Switching System (ESS) Software." *ACM SIGMETRICS Performance Evaluation Review* Fall 1982: 75-85.

[HAA83] Haas, M. and J. Hassell. "A Proposal for a Measure of Program Understanding." *ACM SIGCSE Bulletin* Feb. 1983: 7-13.

[HAL72] Halstead, M. "Natural Laws Controlling Algorithm Structure?" *ACM SIGPLAN Notices* Feb. 1972: 19-26.

[HAL77] Halstead, M.H. *Elements of Software Science.* Elsevier, New York, 1977.

[HAL87] Hall, A. "Tool Interfaces in Integrated Project Support Environments." *Proceedings of the Ninth International Conference on Software Engineering* 1987: 289-90.

[HAM82] Hamer, P. and G. Frewin. "M.H. Halstead's Software Engineering Science - A Critical Examination." *Proceedings of the 6th International Conference on Software Engineering* 1982: 197-206.

[HAM85] Hammons, C. and P. Dobbs. "Coupling, Cohesion, and Package Unity in Ada." *Ada Letters* May-June 1985: 49-59.

[HAN78] Hansen, W. "Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count)." *ACM SIGPLAN Notices* Mar. 1978: 29-33.

[HAR87] Harrison, W. "RPDE3: A Framework for Integrating Tool Fragments." *IEEE Software* Nov. 1987: 46-56.

[HAR88] Harris, D., et al. *The Knowledge-Based Requirements Assistant Final Technical Report.* Vol. I, RADC, Feb. 1988.

[HAR81a] Harrison, W. and K. Magel. "A Complexity Measure Based on Nesting Level." *ACM SIGPLAN Notices* Mar. 1981: 63-74.

[HAR81b] Harrison, W. and K. Magel. "A Topological Analysis of Computer Programs with Less Than Three Binary Branches." *ACM SIGPLAN Notices* Apr. 1981: 51-63.

[HAR82] Harrison, W., et al. "Applying Software Complexity Metrics to Program Maintenance." *IEEE Computer* Sept. 1982: 65-79.

[HAR82] Hartman, S. "A Counting Tool for RPG." *ACM SIGMETRICS Performance Evaluation Review* Fall 1982: 86-100.

[HEI85] Heimbigner, D. and D. McLeod. "A Federated Architecture for Information Management." *ACM Transactions on Office Information Systems* July 1985: 253-78.

[HEN71] Hensen, W. "Creation of Hierarchic Text with a Computer Display." *Ph.D. dissertation, Computer Science Dept., Stanford Univ.* June 1971.

[HEN81a] Henry, S. and D. Kafura, "Software Structure Metrics Based on Information Flow." *IEEE Transactions on Software Engineering* Sept. 1981: 510-18.

[HEN81b] Henry, S., D. Kafura, and K. Harris. "On the Relationships Among Three Software Metrics." *ACM SIGMETRICS Performance Evaluation Review* Spring 1981: 81-8.

[HEN84] Henry, S. and D. Kafura. "The Evaluation of Software Systems' Structure Using Quantitative Software Metrics." *Software Practice and Experience* June 1984: 561-73.

[HEN87] Henderson, P.B. and Notkin, D. "Integrated Design and Programming Environments." *IEEE Computer* Nov. 1987: 12-16.

[HUM87] Humphrey, W.S. and W.L. Sweet. "A Method for Assessing the Software Engineering Capability of Contractors." *Technical Report, CMU/SEI-87-TR-23, ESD-TR-87-186, Carnegie-Mellon University, PA.* Sept. 1987.

[JEN82] Jensen, H. and K. Varian. "A Comparative Study of Software Metrics for Real Time Software." *COMPSAC Proceedings* 1982: 96-9.

[JON78] Jones, T. "Measuring Programming Quality and Productivity." *IBM Systems Journal Vol. 17, No. 1* 1978: 39-63.

[JOR80] Jorgensen, A. "A Methodology for Measuring the Readability and Modifiability of Computer Programs." *BIT, #20* 1980: 394-405.

[KAF81] Kafura, D. and S. Henry. "Software Quality Metrics Based on Interconnectivity." *Journal of Systems and Software, #2* 1981: 121-31.

[KAM86] Kamel, Z. and Rudmik, A. *Project Data Portability Study,* Developed for Naval Air Development Center, Warminister, PA 18974-5000, GTE Communication Systems Corp, June, 1986.

[KRA88] Krasner, E.G. and S.T. Pope. "A Cookbook for Using the Model-View Controller User Interface Paradigm In Smalltalk-80." *Journal of Object-Oriented Programming* August-Sept. 1988: 26-49.

[LAL71] LaLonde, W.R., E.S. Lee, and J.J. Horning. " An LALR(k) Parser Generator." *IFIP Congress 71 Proceedings* 1971: 153-57.

[LAS79] Lassez, J.L. and D. Van der Knijff. "Evaluation of Length and Level for Simple Program Schemes." *COMPSAC Proceedings* 1979: 688-94.

[LAS81] Lassez, J.L., et al. "A Critical Examination of Software Science." *Journal of Systems and Software, #2* 1981: 105-12.

[LAU82] Laurmaa, T. and M. Syrjanen. "APL and Halstead's Theory: A Measuring Tool and Some Experiments." *ACM SIGMETRICS Performance Evaluation Review* Fall 1982: 32-47.

[LEB84] Leblang, D.B. and R.P. Chase, Jr. "Computer- Aided Software Engineering in a Distributed Workstation Environment." *ACM SIGSOFT Engineering Notes/SIGPLAN Notices* May 1984: 104.

[LEB87] Leblang, D.B. and R.P. Chase, Jr. "Parallel Software Configuration Management in a Network Environment." *IEEE Software* Nov. 1987: 28-35.

[LEH86] Lehman, M.M. "Approach to a Disciplined Development Process - The ISTAR Integrated Project Support Environment." *ACM SIGSoft Software Engineering Notes* Aug. 1988: 28-33.

[LEH87] Lehman, M.M. and W.M. Turski "Essential Properties of IPSEs." *ACM SIGSoft Software Engineering Notes* Jan. 1987: 52-5.

[LIP82] Lipow, M. "Number of Faults per Line of Code." *IEEE Transactions on Software Engineering* July 1982: 437-39.

[LIS82] Lister, A., "Software Science - The Emperor's New Clothes?" *Australian Computer Journal* May 1982: 66-70.

[LOV76] Love, T. and B. Bowman. "An Independent Test of the Theory of Software Physics." *ACM SIGPLAN Notices* Nov. 1976: 42-9.

[LOV77] Love, T. "An Experimental Investigation of the Effect of Program Structure on Program Understanding." *ACM SIGPLAN Notices* Mar. 1977: 105-13.

[MAC88] Maclaren, R.H. *Ada Run-Time and DAPSE: Experiences, Issues, and Recommendations* 18 Mar. 1988.

[MAG81] Magel, K. "Regular Expressions in a Program Complexity Metric." *ACM SIGPLAN Notices* July 1981: 61-5.

[MAG82] Magel, K. "A Theory of Small Program Complexity." *ACM SIGPLAN Notices* Mar. 1982: 37-45.

[MAG91] Lawrence J. Magid, "The Best of Both Worlds?" *Bay Area Computer Currents.* October 22-November 4, 1991. p. 28-34.

[MAR81] Markham, D., J. McCall, and G. Walters. "Software Metric Application Techniques." *COMPSAC Proceedings* 1981: 38-45.

[MAR86] Marcus, M., et al. "DAPSE: A Distributed Ada Programming Support Environment." *Proceedings of the IEEE Computer Society Second International Conference on Ada Applications and Environments* 1986: 115-25.

[MCC76] McCabe, T.J. "A Complexity Measure." *IEEE Transactions on Software Engineering* Dec. 1976: 308-20.

[MCC79] McCabe, T. "A Complexity Measure." *IEEE Transactions on Software Engineering* Dec. 1979: 308-20.

[MCC78] McClure, C. "A Model for Program Complexity Analysis." *Proceedings of the 3rd Conference on Software Engineering* 1978: 149-57.

[MCT80] McTap, J. "The Complexity of an Individual Program." *Proceedings of the National Computer Conference* 1980: 767-71.

[MED81] Medina-Mora, R. and P. Feiler. " An Incremental Programming Environment." *IEEE Transactions on Software Engineering* Sept. 1981: 472-82.

[MIC80] Michelson, M. and M.N. Wegman. "PDE1L: The PL1L Program Development Environment: Principles of Operation." *Res. Rep. RC8513, IBM T.J. Watson Research Center, NY* Nov. 1980.

[MOH79] Mohanty, S. "Models and Measurements for Quality Assessment of Software." *ACM Computing Surveys* Sept. 1979: 251-76.

[MON87] Monarch, I. and J. Carbonell. "CoalSORT : A Knowledge-Based Interface," *IEEE Expert,* 1987: 39-53.

[MYE75] Myers, G.J. *Reliable Software Through Composite Design.* Van Nostrand Reinhold, New York, 1975.

[MYE77] Myers, G. "An Extension to the Cyclomatic Measure of Program Complexity." *ACM SIGPLAN Notices* Oct. 1977: 61-4.

[MYE78] Myers, G.J. *Composite/Structured Design.* Van Nostrand Reinhold, New York, 1978.

[MYE83] Myers, B.A. "A System for Displaying Data Structures." *Computer Graphics* July 1983: 115-25.

[NAG73] Nagasaka, K. "On Minimal-Program Complexity Measure." *Proceedings of the 6th International Conference on Systems Science* 1973: 477-79.

[NAI82] Naib, F. "An Application of Software Science to the Quantitative Measurement of Code Quality." *ACM SIGMETRICS Performance Evaluation Review* Fall 1982: 102-28.

[NAS85] Nash, S.H. and S.T. Redwine, Jr. *IDA Paper P-1842: Information Interface Related Standards, Guidelines, and Recommended Practices, (SEE-INFO-004)* 1985.

[NAV82] Department of the Navy, Naval Material Command (NAVMAT). "A Software Engineering Environment for the Navy." *Report of the NAVMAT Software Engineering Environment Working Group* 31 Mar. 1982.

[NIE86] Nielsen, K.W. "Task Coupling and Cohesion in Ada." *Ada Letters* July-Aug. 1986: 44-52.

[NOR88] Norcio, A.F. and J. Stanley. *Adaptive Human-Computer Interfaces.* Naval Research Laboratory, NRL Report 9148, Sept. 1988.

[OCO91] O'Connor, Rory J. "Pen Players Boost the Ante." *Pentop.* Volume 1, Number 1. November/December 1991.

[OTT78] Ottenstein, L. "Further Evaluation of Ana Error Hypothesis." *ACM Software Engineering Notes* Jan. 1978: 27-8.

[OTT79] Ottenstein, L. "Quantitative Estimates of Debugging Requirements." *IEEE Transactions on Software Engineering* Sept. 1979: 504-14.

[OTT81] Ottenstein, L. "Predicting Numbers of Errors Using Software Science." *ACM SIGMETRICS Performance Evaluation Review* Spring 1981: 157-65.

[OUL79] Oulsnam, G. "Cyclomatic Numbers Do Not Measure Complexity of Unstructured Programs." *Information Processing Letters* Dec. 1979: 207-11.

[OVI80] Oviedo, E. "Control Flow, Data Flow, and Program Complexity." *COMPSAC Proceedings* 1980: 146-52.

[PAY81] Payne, A. "A Basis for Software Physics?" *ACM SIGPLAN Notices* Aug. 1981: 27-30.

[PAY86] Payton, T. *Architectural Description of the SDC Common Software Environment (SDC-CSE)* Feb. 1986.

[PCT86] Bull, GEC, ICL, Nixdorf, Olivetti, Siemens, *PCTE: A Basis for a Portable Common Tool Environment, Functional Specifications, Fourth Edition,* 1986.

[PCT88] GIE Emeraude, Selenia, Software Sciences Ltd., *PCTE+ Ada and C Functional Specifications,* Issue 2, July 8, 1988.

[PEE81] Peercy, D. "A Software Maintainability Evaluation Methodology." *IEEE Transactions on Software Engineering* July 1981: 343-51.

[PER86] Perkins, J.A., D.M. Lease, and S.E. Keller. "Experience Collecting and Analyzing Automatable Software Quality Metrics for Ada." *Proceedings of the 4th Annual Conference on Ada Technology* Mar. 1986.

[PER87] Perkins, J.A. and R.S. Gorzela. "Experience Using an Automated MEtrics Framework to Improve the Quality of Ada Software." *Proceedings of the 5th Annual National Conference on Ada Technology, 4th Washington Ada Symposium* Mar. 1987.

[PIW82] Piwowarski, P. "A Nesting Level Complexity Measure." *ACM SIGPLAN Notices* Sept. 1982: 44-50.

[POT82] Potier, D., et al. "Experiments with Computer Software Complexity and Reliability." *Proceedings of the 6th International Conference on Software Engineering* 1982: 94-103.

[PRC85] Planning Research Corporation, Government Information Systems. *Architectural Description of the Automated Product Control Environment (APCE)* 23 Dec. 1985.

[PRO82] Proctor, C. "A Software Quality Metrics Study." *COMPSAC Proceedings* 1982: 187.

[RAM87] Ramanathan, J. and V. Venugopal. "Integration Paradigms in Software Environments." *Proceedings of the First International Workshop on Computer-Aided Software Engineering* May 1987: 245-47.

[REE87] Reed, K. "Practical Software Engineering Environments: Report on the ACM SIGSoft/SIGPlan Software Engineering Symposium." *ACM SIGSoft Software Engineering Notes* Jan. 1987: 56-62.

[REI85] Reiss, S. "PECAN: Program Development System that Supports Multiple Views." *IEEE Transactions on Software Engineering* Mar. 1985: 276-84.

[REI88] Reiss S. "Integration Mechanism in the FIELD Environment." *Brown University Computer Science Department, Technical Report No. CS-88-18* Oct. 1988.

[REI90] Reiss, S. "Connecting tools using message passing in the FIELD program development environment." IEEE Software 1990.

[REI91] Rienhardt, Andy. "Momenta Points to the Future." *Byte.* November 1991. p. 48-49.

[RID85] Riddle, W. E. and J. C. Wileden. *Environment Extensibility Impact on the STARS SEE Architecture (SEE-ARCH- 007)* Apr. 1985.

[RID86] Riddle, W. E. and L. G. Williams. "Software Environments Workshop Report." *ACM SIGSoft Software Engineering Notes* Jan. 1986: 73-102.

[RUD85] Rudmik, Andres and D.H. Vines. *Software Development and Maintenance Environment (SDME) Architecture Description* 31 Oct. 1985.

[RUD86] Rudmik, A and D. Lubeck. *Integrated Project Support in Third Generation Environments,* Proceedings of the Ninteenth Annual Hawii COnference on System Sciences.

[RUM91] Rumbaugh, James. M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, NJ. 1991.

[SAL82] Salt, N. "Defining Software Science Counting Strategies." *ACM SIGPLAN Notices* Mar. 1982: 58-67.

[SCH85] Schmiedekamp, C. *The Impact of Multi-Lingual Support on Software Engineering Environment Architecture* 14 Nov. 1985.

[SCH81] Schneider, G., R. Sedlmeyer, and J. Kearney. "On the Complexity of Measuring Software Complexity." *Proceedings of the National Computer Conference* 1981: 317-22.

[SCH82] Schnurer, K. "Product Assurance Program Analyzer (PAPA), A Tool for Program Complexity Evaluation." *ACM SIGMETRICS Performance Evaluation Review Fall* 1982: 73-4.

[SCH79] Schneidewind, N. "A Case Study of Software Complexity and Error Detection Simulation." *COMPSAC Proceedings* 1979: 843-48.

[SCH78] Schneider, V. "Prediction of Software Effort and Project Duration." *ACM SIGPLAN Notices* June 1978: 49-59.

[SCH81] Schwartz, W. "Software Science Measures Language Level Increase Provided by IBM Series/1 Assembler Structured Programming Facility." *COMPSAC Proceedings* 1981: 59-65.

[SEE84] *Information Interfaces in a Software Environment, SEE- INFO-001* 27 May 1984.

[SEY90] Seybold Report of Publishing Systems. Volume 20. Number 7. December 24, 1990.

[SHA91] Shapiro, Ezra. "A Peek at the New Momenta." *Pentop.* Volume 1, Number 1. November/December 1991.

[SHE79] Shen, V. "The Relationship Between Student Grades and Software Science Parameters." *COMPSAC Proceedings* 1979: 783-87.

[SHE83] Shen, V., S.D. Conte, and H. Dunsmore. "Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support." *IEEE Transactions on Software Engineering* Mar. 1983: 155-65.

[SHN77] Shneiderman, B. "Measuring Computer Program Quality and Comprehension." *International Journal of Man-Machine Studies,* #3 1977: 465-78.

[SIM87] Simmel, S. S. and P. J. Gagne. "Two Fundamental Abstractions for Software Engineering Environments." *Proceeding of the First International Workshop on Computer-Aided Software Engineering* May 1987: 258-60.

[SOF86] *A Guidebook for Software Measurement Data Collection* Software Technology for Adaptable, Reliable Systems (STARS), May 1986.

[SPI87] Spilke, H. and R.P. Chase, Jr. "Toward Software Configuration Management in the Heterogeneous Environment." *Proceedings of the First International Workshop on Computer-Aided Software Engineering* May 1987: 802-6.

[STA82] Stankovic, J.A. "Good System Structure Features: Their Complexity and Execution Time Cost." *IEEE Transactions on Software Engineering* July 1982: 306-18.

[STE86] Stefik, M.J., D.G. Bobrow, and K.M. Kahn. "Integrated Access-Oriented Programming into a Multiparadigm Environment." *IEEE Software* Jan. 1986: 10-18.

[STE88] Stefanescu, C. M. and H.D. Rombach. *A Development Methodology for Distributed Ada Applications* 14 April 1988.

[SUN81] Sunohara, T., et al. "Program Complexity Measure for Software Development Management." *Proceedings of the 5th International Conference on Software Engineering* 1981: 100-6.

[SUT63] Sutherland, I.E. "SketchPad: a Man-Machine Graphical Communication System." *AFIPS Spring Joint Computer Conference* 1963: 329-46.

[SZE88] Szekely P.A. and B.A. Myers. "A User Interface Toolkit Based on Graphical Objects and Constraints." *OOPSLA Conference Proceedings* Nov. 1988: 36-45.

[SZU81] Szulewski, P., et al. "The Measurement of Software Science Parameters in Software Designs." *ACM SIGMETRICS Performance Evaluation Review* Spring 1981: 89-94.

[TAM83] Tamine, J. "On the Use of Tree-like Structures to Simplify Measures of Complexity." *ACM SIGPLAN Notices* Sept. 1983: 62-9.

[TBE86] Teledyne Brown Engineering. *Technology for the Automated Generation of Systems (TAGS)* Jan. 1986.

[TEI81] Teitelbaum, T. and T. Reps. "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment." *Communications of the ACM* Sept. 1981: 563-73.

[TEN88] Tenma, T., et al. "A System for Generating Language Oriented Editors." *IEEE Transactions on Software Engineering* Aug. 1988: 1098-109.

[THA81] Thadhani, A.J., "Interactive User Productivity." *IBM Systems Journal*, Vol. 20, No. 4 1981, 407-423.

[TRO81] Troy, D. and S. Zweben. "Measuring the Quality of Structured Designs." *Journal of Systems and Software*, #2 1981: 113-20.

[TRW87] TRW System Development Division, Defense Systems Group. *Distributed Computing Design System* 18 June 1987.

[VAN78] Van der Knijff, D. "Software Physics and Program Analysis." *Australian Computer Journal* 1978.

[VIN90] Vines, D. and T. King. *Gaia: An Object-Oriented Framework for an Ada Environment*. 1990.

[WAG90] Wagner, Mitch. "OSF To Add More PM Compatibility to Motif." *UNIX Today!* 1990: 14-18.

[WAL79] Walsh, T. "A Software Reliability Study Using A Complexity Measure." *Proceedings of the National Computer Conference* 1979: 761-68.

[WAL87] Walker, J.H., et al. "The Symbolics Genera Programming Environment." *IEEE Software*, Nov. 1987: 36-45.

[WEI74] Weissman, L. "Psychological Complexity of Computer Programs." *ACM SIGPLAN Notices* June 1974: 25-36.

[WEI86] Weiser, M. and B. Shneiderman. "Human Factors of Software Design and Development." *Handbook of Human Factors/Ergonomics*, John Wiley & Sons Ltd., 1986.

[WEI87] Weiderman N., Haberman N., et al. "Evaluation of Ada Environments." *Technical Report CMU/SEI-87-TR-1; ESD-TR-87-101*, March 1987.

[WHI81] Whitworth, M. and P. Szulewski. "The Measurement of Control and Data Flow Complexity in Software Designs." *COMPSAC Proceedings* 1981: 735-43.

[WIL76] Wilcox, T.R., A.M. Davis, and M.H. Tindall. "The Design and Implementation of a Table Driven, Interative Diagnostic Programming System." *Communications of the ACM* Nov. 1976: 609-16.

[WOL88] P Graphite: An Experiment in Persistent Typed Object Management." *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, November 1988.

[WOO79] Woodward, M., M. Hennell, and D. Hedley. "A Measure of Control Flow Complexity in Program Text." *IEEE Transactions on Software Engineering* Jan. 1979: 45-50.

[WOO79] Woodfield, S. "An Experiment on Unit Increase in Problem Complexity." *IEEE Transactions on Software Engineering* Mar. 1979: 76-8.

[WOO81] Woodfield, S., V. Shen, and H. Dunsmore. "A Study of Several Metrics for Programming Effort." *Journal of Systems and Software, #2* 1981: 97-103.

[YAU80] Yau, S. and J. Collofello. "Some Stability Measures for Software Maintenance." *IEEE Transactions on Software Engineering* Nov. 1980: 545-52.

[YOU88] Young M., et al. "Design Principles Behind Chiron: A UIMS for Software Environments." *Proceedings of the 10th International Conference on Software Engineering,* Apr. 1988: 367-76.

[ZOL77a] Zolnowski, J. and D. Simmons. "A Complexity Measure Applied to FORTRAN." *COMPSAC Proceedings* 1977: 133-41.

[ZOL77b] Zolnowski, J. and D. Simmons. "Measuring Program Complexity." *COMPCON Proceedings* 1977: 336-40.

[ZOL80] Zolnowski, J. and D. Simmons. "Measuring Program Complexity in a COBOL Environment." *Proceedings of the National Computer Conference* 1980: 757-66.

[ZOL81] Zolnowski, J. and D. Simmons. "Taking the Measure of Program Complexity." *Proceedings of the National Computer Conference* 1981: 329-36.

*U.S. GOVERNMENT PRINTING OFFICE: 1993-710-033-60217

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C^3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.