IN-61
146667
P.125

NASA Conference Publication 10115

A NASA-Wide Approach Toward Cost-Effective, High-Quality Software Through Reuse

Edited by
Charlotte O. Scheper
Research Triangle Institute
Research Triangle Park, North Carolina

Kathryn A. Smith
Langley Research Center
Hampton, Virginia

Proceedings of the Second NASA
Workshop on Software Reuse held at
Research Triangle Institute
Research Triangle Park, North Carolina
May 5–6, 1992

JANUARY 1993

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-0001

Contents

List of Figures

List of Tables

PRIMARY

1. Introduction

NASA Langley Research Center (NASA Langley) sponsored a workshop on Software Reuse Tools on May 5-6, 1992, at the Research Triangle Institute (RTI) in Research Triangle Park, North Carolina. The workshop was hosted by RTI and led by Kathryn Smith of NASA Langley. Participation was by invitation only and included representatives from four NASA centers (Langley, the Jet Propulsion Laboratory, Goddard, and Johnson), COSMIC, the Air Force's Rome Laboratory, and DARPA's STARS/ASSET program. A complete list of the participants is included in Appendix A of this report.

The primary purpose of this workshop was to exchange information on software reuse tool development, particularly with respect to tool needs, requirements, and effectiveness. The objectives of this information exchange were to 1) identify critical issues and needs in software reuse and 2) identify opportunities for cooperative and collaborative research by addressing the following questions:

- How is software reuse defined?
- What are NASA's requirements?
- What will be the benefits?
- What needs to be done?
- How can results be quantified?

The participants in the workshop presented the software reuse activities and tools being developed and used by their individual centers and programs. These programs address a range of reuse issues: the creation, management, and use of repositories (or libraries); library interoperability; domain analysis; and component certification. Viewgraphs from the presentations are included in Appendix B of this document.

The participants of this workshop agreed that NASA is faced with increased construction and use of software at a time when software development costs are rising and budgetary resources are shrinking. This increased need is due in part to the exponential growth in the amount of data resulting from NASA missions that must be processed and analyzed as well as to the growth in software needs to conduct, control, and manage the missions themselves. Producing software becomes more difficult and more costly as software becomes more complex, documentation becomes more intricate, and technology undergoes rapid change. The participants concluded that a concerted effort to promote and enable software reuse is required to accomplish cost-effective software development under these conditions. This report summarizes the workshop findings and presents the group's plan for defining the goals and objectives for NASA-wide coordination of software reuse activities.

1.1. Mission and Goals

The mission of the group's proposed software reuse activities is to facilitate the construction and use of high-quality, cost-effective software. It proposes to accomplish this mision by creating a quality-conscious reuse environment at NASA that builds on the agency's past achievements in software development to accomplish today's missions.

Reuse is a *process* by which *components* created by *activities* in one software development effort are used again, with or without modification, in other software development efforts. *Components* include artifacts from all aspects of software development. These artifacts can be requirements, specifications, designs, code modules varying from low-level subroutine modules to stand-alone modules to complete subsystems, interface requirements, revision histories, component- and system-level test cases, historical performance metrics of usage and failure rates, development standards, and risk information. *Activities* include the complete range of development and maintenance activities, such as requirements analysis, design, implementation, testing, field operations, and maintenance modifications. *Process* includes both the creation of components that are capable of being reused as well as the actual reuse of components. It encompasses identification, construction, verification, storage, retrieval, and modification of components.

The group has four specific goals for its reuse activities:

1. **Enable NASA missions in the era of very limited resources.** This goal specifically addresses supporting the smaller, low-cost missions. The proposed reuse effort will accomplish this goal by putting into place a mechanism to build software better, faster, and cheaper than can currently be done.

2. **Promote and improve quality in NASA software products and processes.** Two aspects to accomplishing this goal are the application of TQM principles to foster a quality-conscious environment, and the development and

use of the processes and metrics necessary to achieve a higher SEI (Software Engineering Institute) software capability rating.

3. **Preserve, package, and exploit NASA's software legacy**. This goal will be accomplished by establishing a reuse environment that allows components from existing systems to be reused, that applies lessons learned from one system development to another, and that promotes interoperability among new and existing systems.

4. **Foster a pervasive culture of software reuse within NASA**. Such a culture is an integral part of creating a successful reuse environment. This goal can be accomplished through education and coordination. In the area of education, this proposed effort will seek to improve awareness of software reuse and to educate current and future engineers and managers. In the area of coordination, it will work to increase collaboration across NASA and to formalize and incorporate reuse into the NASA software life-cycle process.

1.2. Customers and Sponsors

Table 1.1 identifies Nasa Headquarters customers and sponsors having an existing or potential interest in software reuse efforts.

Table 1.1. NASA Headquarters Customers (C) and Sponsors (S) in Software Reuse Efforts

HQ Code	Contacts	Areas
RC	(S) Lee Holcomb	HPCC, CAS, ESS
RJ	(S) Kristin Hessenius	Aero R&T
RS	(S) Sam Vernneri	Space R&T
SMI	(C/S) Joseph Bredenkamp	Data Management
SZE	(C) Guenter Reigler	Astrophysics
SE		EOS
SS		Space Physics
SL		Planetary
SB		Life Sciences
SN		Microgravity
CU	(S) Frank Penaranda	Technology Transfer
QE	(S/C) Don Sova	Technical Standards
QR	(S/C) Alice Robinson	

2. Technical Rationale

2.1. Benefits

Results from a market analysis conducted for the ASSET repository were discussed at the workshop. This analysis determined that the perceived benefits of reuse are, in order of priority, improved cost/productivity, reduced development time, increased quality, and increased competitiveness. The surveyed users thought that the reuse approach would provide an improved development environment where prototyping, development, modification, and porting could be accomplished efficiently and more successfully. They also felt that it would provide better communication among the staff involved in the development. These improvements would lead to the projected cost/productivity and development time gains. The users felt that the quality and reliability of the software would be improved, without increasing development costs, due to the increased testing and easier maintenance that reuse would provide. Finally, they felt that the ability to produce higher quality software at less cost would let them bring a better product to the market faster than their competitors.

The findings of this market analysis agree with the general consensus that cost savings can be realized through increased software reuse. These cost savings result not only from the reuse of code, but also from the retention of software engineering knowledge and experience in a database that others can access. This allows improvement of the development process by building on past experience and lessons learned. In fact, it is now thought that the greatest benefits will probably be realized by reusing more abstract artifacts of software development than code modules, including artifacts from the process, design, and model levels [1].

2.2. Problems and Barriers

Before software reuse can be a practical reality, several issues relating to quality, cost, and usability must be resolved [2,3,4]. The goal of reusable software is to cut costs, but, depending on the application and system, this may not always be the case [5]. Practically, component retrieval costs should be less than component development costs. A previous NASA workshop [6] concluded that there was a need for economic models of reuse that could quantify the cost tradeoffs, identify the cost factors, and allow the calculation of how many times a component must be reused to justify the cost of creating and reusing it.

The ASSET market analysis concluded that barriers to software reuse exist in the lack of mature processes, standards, and tools for reuse; company cultures and attitudes based on current software development processes; the front-end investment cost of designing reusable software; unresolved legal issues such as intellectual property rights, licensing and contractual issues, and product liability; and a lack of component suppliers, maintenance and support, and concern.

The following additional items were identified as potential barriers to reuse by the participants of this workshop:

- The need for systems with unprecedented requirements
- Limited information access mechanisms
- The perception that building new software is faster than searching, identifying, retrieving, understanding, and modifying existing software objects
- A lack of methods/procedures for reuse
- No common environment for reuse
- A lack of management support
- A lack of successful case studies
- Inertia

2.3. Technical Approach

The proposed effort to promote and enable software reuse throughout NASA requires a coordinated attack on a broad set of entrenched, interrelated problems. The problem space is described in Table 2.1.

Within each of these problem areas, progress can be made by pursuing all or some of the following seven activities:

1. Define solution approach
2. Evaluate feasibility
3. Build prototype/product
4. Agree upon broad standards
5. Train
6. Distribute
7. Commercialize - enlist industry support

A matrix in which the problem areas are listed down the side, and the solution activities along the top, provides a framework for assessing progress towards widespread software reuse. In the following subsections, each problem area is briefly described. In the next section, the state-of-the-art at NASA is examined by filling in the matrix with activities currently being pursued by NASA centers.

2.3.1. Process

Achieving widespread software reuse is not simply a technological problem, nor is it simply a matter of creating and collecting a large number of reusable assets. A reuse-based approach to software engineering requires a change in the processes followed by all parties involved. This includes not only engineering processes, but also investment, acquisition, and management processes. Each of these areas presents obstacles to reuse which must be overcome. By addressing them in terms of the roles involved, the authority and interrelationships of these roles, and the procedures they would follow in a reuse-based development context, an Operations Concept of reuse can

Table 2.1. The Software Reuse Problem Space

1.	**Process**
1.1	Business Process
1.1.1	Market Analysis
1.1.2	Incentives for Reuse
1.1.3	Management Policy
1.2	Engineering Process
1.2.1	Domain Engineering
1.2.2	System Engineering
1.2.3	Software Engineering
1.3	Legal Issues
1.3.1	Acquisition Policy
1.3.2	Capitalization Policy
1.3.3	Liability
2.	**Technology**
2.1	Engineering Methods
2.1.1	Object-Oriented Methods
2.1.2	Generic Assets
2.1.3	Megaprogramming
2.2	Libraries
2.2.1	User Interfaces
2.2.2	Asset Classification
2.2.3	Asset Management
2.2.4	Library Interoperability
2.3	Measurement
2.3.1	Certification Metrics
2.3.2	Experience Metrics
3.	**Assets**
3.1	Life-cycle Products
3.1.1	Requirements
3.1.2	Designs
3.1.3	Code
3.1.4	Test Procedures
3.1.4	User Guides
3.1.4	Other Life-cycle Products
3.2	Captured Knowledge
3.2.1	Reuse Guidance
3.2.2	Reuse Experience
3.2.3	Process Models

be developed that can serve as a statement of vision against which progress can be measured.

2.3.1.1. Business Process

This category refers to the set of problems concerning the financing, acquisition, and management of reusable software and of software developed by means of large-scale reuse. It includes such issues as rights retained on reusable assets, royalty structures for the use of such assets, and liability for defects in the assets.

2.3.1.1.1. Business Analysis

The economics of reuse are not straightforward. Models of the return on investment have been developed which show the extent of reuse necessary to justify an initial investment in developing reusable software. Market analysis is necessary in order to estimate whether the projected reuse is a reasonable expectation in a given domain.

2.3.1.1.2. Incentives for Reuse

Current Government acquisition policies, as stated in the Federal Acquisition Regulations (FAR), tend to discourage reuse. For example, if a company cannot retain the rights to reusable software incorporated in a Government system, then there is no incentive for the company to invest the extra resources that reusable software requires. Similarly, development of reusable software under a Government program is implicitly discouraged, since such development requires additional resources and can drive up the cost of a single system.

2.3.1.1.3. Management Policy

Reuse-based software development requires a shift of management priorities away from "whatever it takes to make this project succeed" to a longer-term vision encompassing many projects. A domain orientation, which sees the development of a single system as one instance in the development of many similar systems, is thus required not just by engineers but also by management.

2.3.1.2. Engineering Process

This category refers to the technical procedures followed by software engineers throughout the development life cycle. Experience has shown that large scale reuse is not achieved by simply making libraries of reusable components available to developers, with no corresponding changes in the processes the developers follow. The "not invented here" (NIH) syndrome and various other obstacles to reuse necessitate a new understanding of what it means to develop software. This new understanding is encapsulated in the term Domain Engineering, which must now be added to the familiar concepts of system and software engineering.

2.3.1.2.1. Domain Engineering

A domain is a family of similar systems. It corresponds to a familiar application area or technical area, such as avionics systems, satellite systems, accounting systems, database systems, communications systems, etc. Domains can contain subdomains, which represent standard parts of a complex system (for example, the ground segment of a satellite system).

Domain engineering is a discipline that stems from the fact that, within a given domain, the same techniques, design alternatives, tradeoffs, rules of thumb, testing approaches — and other aspects of the engineering process — are frequently encountered again and again. Whether there is a formal "software reuse" program in place or not, the most effective engineers informally reuse the knowledge and techniques that they have built up through experience. Domain engineering seeks to systematize this process and make the legacy of built-up knowledge available to all members of a development team.

Domain engineering is an approach to developing systems by exploiting similarities within a given domain. Individual systems in a domain are developed by instantiating a generic architecture, which describes the common structure of systems in the domain. A good generic architecture also identifies the ways in which individual systems can vary; ideally, it provides an easily used mechanism, such as parameter instantiation, for describing the unique aspects of a new system. Through the use of the generic architecture and its instantiation mechanism, the development of strictly new software is kept to a minimum.

Domain engineering is intrinsically evolutionary: each new application yields experience that is fed back into the domain model (which consists of the generic architecture as well as the techniques and supporting knowledge necessary for using the architecture). This feedback means that the domain model — which is a model of recommended engineering practice within the domain — continually changes as requirements become more and more complex, and as improved solution techniques are discovered.

2.3.1.2.2. System Engineering

Decisions made during the system design process (for example, partitioning decisions and processor allocation decisions) can impact the feasibility of reuse during software development. Thus, the concepts of reuse and domain engineering need to be integrated into the systems engineering process as well.

2.3.1.2.3. Software Engineering

Developers must be trained to view reuse not just as an ad hoc labor-saving technique, but as part of an overall engineering discipline that minimizes risk by building on past experience. Reuse must not be relegated to the coding phase of a project. It is equally (perhaps more) important in the earlier life-cycle phases, i.e., requirements analysis and design, and can be effectively applied in other activities such as test planning and test development as well.

2.3.1.3. Legal Issues

This category refers to a range of problems that arise when reuse is attempted between organizations (e.g., Government and industry). Changes are required in the Government's acquisition policies as well as in the laws governing rights to software in the commercial arena.

2.3.1.3.1. Acquisition Policy

As already mentioned, the FAR tends to discourage reuse on the part of Government contractors. Provisions need to be made for the retention of rights to reusable software incorporated in a Government system. In addition, the way in which software is maintained may need to change, as source code for proprietary components may not be made available to a maintenance contractor.

2.3.1.3.2. Capitalization Policy

Investment in the development of reusable assets would by encouraged by a modified accounting system, in which newly written software could be amortized over a longer period of time than its development period. This would to some extent mitigate the additional expense of developing software to be reusable.

2.3.1.3.3. Liability

As software assets come to be treated more as commercial products, the question of liability for errors arises. The question is intrinsically complex because the context in which an asset is intended to be reused is typically not completely defined (formal specification is not yet widespread in the software industry). Certification is an approximate process. The question becomes even more complex when there are multiple layers of reuse, e.g., component A from organization A is reused in tool B from organization B, which is reused in system C for organization C.

2.3.2. Technology

The technological problems have been addressed more extensively, to date, than the process issues. A great deal of progress has been made in our understanding of how to develop assets that are reusable, and how to organize and present these for easy location and access by developers. Less progress has been made in the area of measurement, i.e., how do we assess the success of a reuse program? Nevertheless, significant technical problems remain in all three areas of engineering methods, libraries, and measurement.

2.3.2.1. Engineering Methods

Creating reusable assets is a technical challenge because software requirements continually evolve. Designing for reuse requires the ability to predict how requirements will change over time, and in what different contexts an asset will have to be reused. The basic software engineering goals of modularity and encapsulation improve the chances of reuse but do not by themselves solve the problem. In fact, none of the methods discussed below solves the problem, but they represent significant progress in our understanding of what makes software reusable.

2.3.2.1.1. Object-Oriented Methods

Data encapsulation and information hiding are basic techniques that aid in the definition of components that are loosely coupled to their environment (and can therefore be reused in other environments). Decomposing software in terms of "objects," which represent the entities or important "things" in a given domain, has turned out to be a systematic way of achieving data encapsulation and information hiding. This is known as object-based software development. Object-oriented development goes one step further by organizing objects into classes and subclasses. Members of a subclass inherit attributes and capabilities from the parent classes. Inheritance has been advocated as a means of achieving reuse: by having an object inherit functions from a parent class, a developer does not have to re-implement them in the subclasses. However, systematic use of inheritance has also led to difficulties in reuse and maintenance, which have been documented in the object-oriented programming literature (e.g., the proceedings of the Object Oriented Programming, Languages, Systems, and Applications — OOPSLA — Conferences). The difficulties stem primarily from the dependencies of a subclass on its parent classes. These dependencies work against the encapsulation (localization) of information that are a hallmark of good software engineering.

Organizing objects into classes and subclasses can be a useful tool in understanding a problem domain during the analysis and design phases, even if inheritance is not implemented in the programming language used. Overall, there is a consensus in the software engineering community that object-orientation supports the development of reusable software. Unfortunately, there has been very little empirical measurement performed to test this belief.

2.3.2.1.2. Generic Assets

Languages such as Ada (and now C++) allow for the definition of components that are generic, in that they are parameterized to allow their use in different contexts.

For example, a generic list package may be used to manage lists of different types of objects: the generic package is instantiated according to the particular object type to be supported in a given application.

Recently, the notion of a generic asset has been extended to encompass more than code components. Software engineers now speak of generic architectures for certain types of systems (this is the thrust of a major DARPA program - Domain Specific Software Architectures). In the context of domain engineering (see Section 2.3.1.2.1), we can even speak of generic requirements specifications.

Class hierarchies and generic assets are two methods of building in variability, so as to increase the chances of an asset being reused. In a class hierarchy, variation is accommodated by the range of subclasses of a given parent class (e.g., the varieties of a window in a windowing system). In a generic asset, variability is accommodated by means of parameters that must be instantiated in order to use the asset.

2.3.2.1.3. Megaprogramming

Megaprogramming refers to the idea of building software systems out of large building blocks, each of which represents a rich capability in its own right. The consensus in the software engineering community seems to be that this can be achieved in domain-specific contexts, where the typical architecture and building blocks of a system are well understood. Megaprogramming is, for this reason, very closely related to domain engineering.

In domains where there is a great deal of commonality from one system to another, the synthesis of the building blocks into new systems can often be described in terms of a very high-level language (VHLL); for example, architecture diagrams that refer to well-known subsystem implementations. Automated code generation plays an increasingly important and feasible role in this context to create the code that ties together the specified building blocks.

2.3.2.2. Libraries

Most of the research and development in software reuse has concentrated on the development of library systems. There are numerous issues remaining to be resolved concerning the best way to present information to the user, the most effective ways of organizing a library to facilitate finding desired assets, and the ability of multiple libraries to interoperate in a seamless fashion despite differences in their internal storage procedures and user interfaces.

2.3.2.2.1. User Interfaces

The overall problem here is to prevent a user from being overwhelmed by massive amounts of information while providing access to the assets that will meet his/her current requirements. The advent of graphics/windowing systems and of hypertext/hypermedia systems has opened many new possibilities for presenting information to the user. In addition to query-driven database searches, some systems now use hypertext techniques that allow users to browse or navigate through the contents of a library, following reference or similarity links from one asset to another. Graphical interfaces can show "neighborhoods" of closely related assets, allowing the user to grasp the overall content of the library in a visual manner.

2.3.2.2.2. Asset Classification

This problem bears directly on the ease with which users can locate assets meeting their requirements. Assets may be classified hierarchically, as in a tree structure, or by means of facets, which are independent attributes of an asset (e.g., function, author, programming language, etc.). Both overall methods present problems. Hierarchical schemes have been used in object-oriented programming systems such as Smalltalk, and have frequently proven difficult to use when the conceptual scheme assumed by the creator of the library is markedly different from that of the user. Faceted systems are frequently limited to describing superficial characteristics of an asset; for example, the function of a component may be described in a manner that leaves may questions about the operation of the component unanswered.

In addition to problems with both methods of classification, determining the specific classification of an asset is inherently problematic. The name that one person uses to describe a function may be different from the name used by someone else. Support for synonyms and similarity is therefore desirable.

2.3.2.2.3. Asset Management

Software evolves, and not all reuse will be verbatim reuse. There will be circumstances in which modified assets are submitted to a library for inclusion as a variant to the original on which it is based. In addition, if problems with reuse are reported, it may be necessary to maintain software stored in a reuse library. These and other circumstances create a problem of managing the assets in a library. Procedures and supporting technology are needed for configuration control, access control, and similar asset management tasks.

2.3.2.2.4. Library Interoperability

Widespread sharing of information among software engineers will require the ability of libraries to interoperate, so that requests at one library system can be satisfied by retrieving assets from another, perhaps geographically remote, system. The Reuse Library Interoperability Group (RIG) is currently addressing this problem.

2.3.2.3. Measurement

This area has received the least attention of all the technological aspects of reuse, and yet it is crucial to achieving any kind of objective success.

2.3.2.3.1. Certification Metrics

Various schemes have been proposed for annotating reusable assets with a certification measure — a description of the confidence the library management has in the correctness and quality of the asset. Because quality is not a precisely defined concept in software (it has different meanings on different projects), and because in the absence of formal specifications even correctness is not precisely defined, certification must be viewed as an approximate indicator rather than an absolute seal of approval. Methods for certifying reusable assets will evolve as testing theory, use of formal methods, and approaches to quality assurance evolve.

2.3.2.3.2. Experience Metrics

This category refers to the collection of measurements concerning the practice of reuse. These measurements may include how much software from a library is being reused, what percentage of new systems consists of reused code, how many successful vs. unsuccessful searches there have been in a library system over a given period of time, how many errors have been encountered in reused assets, how many modifications have been necessary in reusing an asset, what kinds and frequencies of problems have been encountered in reusing various assets, etc.

Information gathered from such measurements can be used to refine the organization of a library, improve the procedures for using the library, improve other aspects of the software development process, filter out unneeded or substandard assets from a library, and in many other ways contribute to an ongoing process improvement program.

2.3.3. Assets

The development of a sizable store of reusable assets is, obviously, key to a successful reuse program. There are two main points to be made: 1) we should be thinking of reusing life-cycle products in general, not just code, and 2) we can (and must) reuse knowledge that has accrued over the years of developing systems in a domain.

2.3.3.1. Life-cycle Products

Many products created over the course of the software development life-cycle can be reused effectively in future systems. Many researchers in the field have come to the conclusion that reusing code without reusing requirements, specifications, and designs will never lead to more than ad hoc reuse. It is the requirements and design that establish the context for code components — for example, the interfaces — a context that is either consistent or inconsistent with the assumptions of existing code components. Thus, it is in the requirements analysis and design phases that key decisions affecting the potential for reuse are made. To the extent that these decisions are consistent with those made in the past (i.e., requirements and designs are reused), the chances of successfully reusing code are increased.

In addition, there are the obvious economic benefits to be gained if a design specification, for example, can be created by means of a few modifications to an existing document. This is also a means of reducing risk on a project, since the number of decisions without precedent is reduced.

2.3.3.2. Captured Knowledge

It is sound engineering discipline to build on knowledge accumulated through prior efforts, but relatively little attention has been paid to integrating this process into a reuse framework. The advantage of doing so is that knowledge can be shared rather than remaining in the mind of a single developer. A reuse program should therefore look at ways of packaging previously accrued engineering knowledge so as to make it available to the developers of new systems.

The 1990 report of the Computer Sciences and Technology Board (CSTB) of the National Research Council strongly recommended the use of handbooks in specific disciplines as a means of packaging and transferring this kind of knowledge (Communications of the ACM, March 1990). Such "handbooks" could in fact be on-line and made available as part of a reuse environment, providing guidance on how to reuse various assets, information about past experience in reusing specific assets (lessons learned), and criteria for choosing reusable assets.

In addition, alternative process models, suitable for projects with different characteristics (e.g., size, criticality, performance requirements, etc.), could be stored and made available as part of this on-line database of knowledge. This knowledge would constantly evolve as a function of the experience metrics collected (see Section 2.3.2.3.2). In the long run, the reuse of packaged knowledge of this sort can have a great impact on software quality and productivity because they directly address the risk factors associated with software development.

2.4. State-of-the-Art

2.4.1. Current Status of NASA Efforts

The workshop identified current reuse activities at four NASA centers: Langley Research Center, the Jet Propulsion Laboratory, Goddard Space Flight Center, and Johnson Space Center. The tools resulting from these activities are described in the following sections, and the technical points of contact are summarized in Table 2.2.

Table 2.2. Technical Contacts for NASA Reuse Tools

Technical Contact	NASA Center	Tools/Programs
Kathryn Smith	LaRC	Eli (InQuisiX)
Randy VanValkenburg	LaRC	SEAL
Ed Ng	JPL	HyLite
Walt Truszkowski	GSFC	LEARN-92, KBSEE
Mike Bracken	GSFC	KAPTUR
Charles Pitman	JSC	RBSE, REAP, SimTool, PCS/ESL

NASA Langley Research Center

The Eli Software Synthesis System is an automated set of cooperating reuse tools that NASA Langley has been sponsoring. It is in its third phase of development, during which it is being commercialized as InQuisiX. The component tools are library facilities to classify, store, and retrieve reusable components; design synthesis; component checkout; file checkout; and Ada component metrics. Eli has been designed to be tailorable to specific users needs. It supports user-defined component classes and classifications and many types of attributes. The goal of this system is to automate the development and use of reusable components to make software reuse easier to accomplish.

Eli is an operational product, running under X11 on a Sun4. It has a window and menu-based user interface. It manages code, design, test case, and documentation components and performs the complete set of library functions. Additionally, it provides facilities for integrating library components into new systems under development.

The Software Engineering and Ada Laboratory (SEAL) at NASA Langley is involved in a number of efforts that will facilitate the implementation of reuse in the software development process. A domain analysis is underway that will identify the potential for reuse for the domain of interest to the SEAL. The SEAL is cooperating with the hardware and systems engineering branches at Langley to document a systems engineering approach that includes participation of software engineers from the earliest stage of development and that will advocate the development of standards for hardware, limiting the options software has to address. An object-based design methodology has been defined in the SEAL and many of the code modules actually developed are in the form of reusable, generic Ada packages. Finally, the SEAL is developing guidebooks for developing reusable Ada components/systems and for a tailorable software engineering process.

The Jet Propulsion Laboratory

HyLite is an R&D activity of JPL that is producing a tool to facilitate the construction of electronic libraries for software components, hardware parts or designs, scientific databases, bibliographies, etc. HyLite evolved from a task formerly entitled the Encyclopedia of Software Components (ESC) and its major area of applicability has thus far been software resue. HyLite has a graphical user interface (GUI) to its set of library functions. These functions include inserting new components and property knowledge, browsing and searching databases, and retrieving software from selected networks. It also contains a library of math software and a library of data structures and algorithms.

HyLite has employed advanced technology in developing its component functions. These technologies include object-oriented databases, semantic networks for classification, and automatic GUI generation. The effort is currently addressing the use of AI technologies for intelligent retrieval based on learning from experience, user models, the correction and/or completion of retrieval statements, and suggestions for alternative retrievals.

A prototype for beta testing exists for the color Macintosh. The prototype uses SuperCard, Macintosh Allegro Common Lisp, Pixel/Paint Professional, Canvas 2.0, and Think C to implement the system's functions. This prototype is currently being ported to UNIX workstations running under XWindows and will be upgraded to include the AI technologies, a history mechanism, and more complete Hypertext capabilities. Additional efforts are underway to adapt HyLite as a graphical front-end for a national software exchange experiment, to adapt it as an intelligent front-end to NAIF (a library of software tools and datasets for space flight navigation systems), and to connect to NetLib. Initial preparations are being made for commercialization.

Goddard Space Flight Center

LEARN-92 (Learning Enhanced Automation of Reuse Engineering) is an experimental project that is using conceptual clustering techniques from artificial intelligence to automatically develop a classification scheme for code components. This capability would support the domain engineer, who must create a classification scheme for components as part of the domain model. A prototype version of the tool is planned to be completed by the end of September 1992.

LEARN-92 is intended to provide the software engineer with a classification of components based on their role in the problem space (i.e., what problem they solve), rather than the solution space (how they are implemented.) The inheritance hierarchy of an object-oriented programming system, such as C++, provides a solution-space organization; this is often not very helpful to programmers who are searching for a reusable component to perform a specific function.

LEARN-92 will provide an automated mechanism for hierarchical classification of code components, based on faceted descriptions of these components. A unique aspect of the faceted descriptions is that the facet space is extendible "on the fly" by the user who is placing a component into the system. The user is encouraged, but not required, to use existing facets in describing a new component. The focus in this effort is on code components, but the classification mechanisms being implemented in LEARN-92 could work for other forms of assets as well.

KAPTUR (Knowledge Acquisition for Preservation of Trade-offs and Underlying Rationales) is a tool under development for preserving and building on NASA's engineering legacy. It captures the engineering decisions/rationales that went into the development of software assets and provides an easy-to-use environment for accessing that knowledge. The functionality implemented by KAPTUR includes entering new architectures, recording rationales, placing rationales within the context of an overall domain model, browsing alternatives, understanding decisions, and selecting for reuse.

KAPTUR supports an approach to domain engineering in which assets are organized in terms of their *distinctive features*, which represent key engineering decisions, and which are justified by *rationales*. The approach is also distinguished by the fact that it is *case-based*, i.e., actual legacy products are included in the database, not just generic models for future use. KAPTUR's approach to asset classification uses a typing structure including both domain-independent and domain-dependent asset types. Within a type, assets are classified on the basis of their features. The KAPTUR concept of feature is broader than that found in the Software Engineering Institute's Feature-Oriented Domain Analysis (FODA) method. KAPTUR employs a novel user interface approach which is based more on direct display and manipulation of the database rather than queries. A hierarchical map of alternatives and a stack of pages describing them are presented to the user in a window and menu-based format.

KAPTUR currently runs on a Sun SPARCS Station. Version 2.0 has been released, following versions 1.0 and two earlier prototypes. The system is currently being distributed to interested/potential users, and a training course on KAPTUR and Domain Analysis is being developed. The developers of KAPTUR maintain that the continuous feedback loop this type of system provides between the supplier of reusable components and the user of those components is the key to successful reuse.

The KBSEE (Knowledge-Based Software Engineering Environment) is a prototype environment to support the production of new systems by configuring generic assets stored in a domain model. It incorporates the Evolutionary Domain Life-Cycle (FDLC) model in which new systems are used to update the domain model to make it more responsive to future requirements.

The KBSEE makes reuse the central activity of the software engineering process. Development is seen as a process of identifying the required features of a new system, retrieving the assets possessing those features from the generic domain model, checking the mutual consistency of the assets, and configuring them into the new system. Specification of the required features is done by the developer; all the other steps are performed by the KBSEE.

The domain model, as stored by the KBSEE, consists of a hierarchy of generic assets, each of which possesses certain features that make it suitable or unsuitable for a given application. The generic assets are created through the process of Domain Analysis, which abstracts the functionality found in existing and planned systems in the domain.

Assets are organized into whole-part and class-subclass hierarchies. In addition, assets possess features (similar to the notion of feature in the Software Engineering Institute's FODA method), which are used to determine which assets should be retrieved to meet the requirements of a new system. Features are described as *mandatory* (must be present in any system), *variant* (one of several variants must be present in any system), or *optional* (may or may not be present).

A prototype KBSEE has been developed, and its feasibility is now being tested in the Payload Operations Control Center domain. The KBSEE effort has focused to date on the storage of generic requirements specifications and the automated configuration of requirements specifications for new systems based on the generic versions. This supports a development process that consists of configuring assets each of which can represent a complex capability in its own right. This highly automated concept of software development supported by the KBSEE makes it suitable for megaprogramming.

Johnson Space Center

The NASA Repository - Based Software Engineering Program (RBSE) directed by NASA Johnson Space Center has operated a prototype public-domain software reuse library (AdaNET) since 1989. Updates to the AdaNET architecture, including high-performance hardware and an open-systems-based library management system are reversing a trend to degraded responsiveness and capability. The RBSE is committed to making reuse part of the mainstream of software development practices and is working to achieve this by delivering and supporting a robust set of products supporting research to fill critical technology gaps, and adapting to changing customer requirements. Through the Reuse Interoperability Group, RBSE is involved in developing standards for interoperability among government-funded reuse libraries, and sees interoperability as key to expanding the base of library suppliers and customers.

In addition to RBSE, NASA's Johnson Space Center supports several activities that are related to software reuse. The Re-Engineering Application/Project (REAP) is developing an integrated reengineering environment, including methods and tools. It captures all code and as much as possible of other software life cycle products in an electronic repository and provides analysis support for abstracting, grouping, and structuring the information.

SimTool is also supporting the domain engineering process through the construction of simulations of new applications based on a library of models from the domain. Using SimTool's library of executive software components, application interfaces, and math models, the user builds an application specification. This specification identifies which components are to be integrated and how they relate to each other and the simulation.

The Parts Composition System/Engineering Script Language (PCS/ESL) provides reusable, domain-specific software parts, catalogs of parts, and libraries. The software parts consist of primitive modules and drivers/graphs. This tool lets the developer retrieve parts from the library and recombine or modify them into new, executable applications. The modules and applications are represented in the library as graphs. The ESL is a graphical language for composing complete applications from software parts, and as such is one approach to megaprogramming. A prototype of this system has been built and is being tested.

2.4.2. Assessment of State-of-the-Art

A problem area/solution activities matrix based on the framework described in Section 2.3 was created to determine and assess the current status of reuse activities at the NASA centers. The participants at this workshop filled in the matrix with respect to the reuse activities and tools being pursued by their centers. These individual results were then compiled into the matrix in Figure 2.1 using the following key to identify the individual tools:

Tool Number	Tool Name
1	Eli (InQuisiX)
2	HyLite
3	SEAL
4	RBSE
51	LEARN-92
52	KAPTUR
53	KBSEE
6	RBSE, REAP, SimTool, PCS/ESL

Notes related to individual column entires are included after the table.

This matrix provides a snapshot of existing NASA reuse activities in a framework that denotes their status with respect to the issues that this workshop identified as crucial to the successful development of a NASA-wide reuse environment. This snapshot clearly illustrates where NASA is now and provides a basis for determining where future efforts should be directed in resolving these issues.

PROCESS PORTION OF MATRIX

	A	B	C	D	E	F	G	H	I	J	K
1					Define Solution Approach	Evaluate Feasibility	Build Prototype/ Product	Agree upon Broad Standards	Train	Distribute	Enlist Industry Support
2											
3											
4	Process										
5											
6			Market Analysis				3			3	
7											
8			Incentives for Reuse		4						
9											
10			Management Policy		4				3		
11											
12											
13			Domain Engineering		4, 51	51	3, 51		52	3, 4	
14					52, 53, 6	52, 53	52, 53				
15			System Engineering		3			3		4	
16											
17			Software Engineering		51, 53	51, 53	3, 51	3	3	3, 4	
18							52, 53, 6				
19											
20			Acquisition Policy		4					4	
21											
22			Capitalization Policy							4	
23											
24			Liability							4	
25											

Figure 2.1. Current Status of NASA Efforts

TECHNOLOGY PORTION OF MATRIX

	A	B	C	D	E	F	G	H	I	J	K
1					Define Solution Approach	Evaluate Feasibility	Build Prototype/ Product	Agree upon Broad Standards	Train	Distribute	Enlist Industry Support
2											
3											
4	Technology										
5		[illegible]									
6			Object-Oriented Methods		3	2, 3, 6	2, 3	3	3	3, 4	3
7											
8			Generic Assets		53	53	3, 53	3	3	3, 4	
9											
10			Megaprogramming		53	3, 53	3, 53, 6		3	4	
11											
12		[illegible]									
13			User Interfaces		1, 4, 52	1, 2, 4	1, 2, 4			4	
14						52	52, 6				
15			Asset Classification		1, 3, 4	1, 2	1, 2, 3, 4			4	
16					51, 52, 53	51, 52, 53	51, 52, 53, 6				
17			Asset Management		1	1, 2	1, 2, 4, 6			4	
18											
19			Library Interoperability		4	2, 4	2, 6	4		4	
20											
21		[illegible]									
22			Certification Metrics		1, 4	1, 4				4	
23											
24			Experience Metrics		3, 4					4	
25											

Figure 2.1. Current Status of NASA Efforts (Continued)

ASSETS PORTION OF MATRIX

	A	B	C	D	E	F	G	H	I	J	K
1					Define Solution Approach	Evaluate Feasibility	Build Prototype/ Product	Agree upon Broad Standards	Train	Distribute	Enlist Industry Support
2											
3											
4	Assets										
5		Lifecycle Products									
6			Requirements		1, 53	1, 2 53	1, 2, 3 52, 53	3		3, 4	
7											
8			Designs		1	1, 2	1, 2, 3 52	3		3, 4	
9											
10			Code		1, 51	1, 2 51	1, 2 3, 51	3		3, 4	
11											
12			Test Procedures		1	1, 2	1, 2, 3	3		3, 4	
13											
14			User Guides		1	1, 2	1, 2, 3	3		3, 4	
15											
16			Other Lifecycle Products		1	1	1, 3	3		3, 4	
17											
18		[illegible]									
19			Reuse Guidance				3			3	
20											
21			Reuse Experience		52	52	3, 52			3	
22											
23			Process Models		4		3	3	3	3	
24											

Figure 2.1. Current Status of NASA Efforts (Continued)

PROCESS NOTES

Tool 3: SEAL

Cells 6G,J - The domain analysis of 13G should answer the market analysis question: does the potential for reuse in our domain justify the cost of reuse efforts? See 13J.

Cell 10I - SEAL management is committed to the "domain orientation," and we are seeking to educate other areas of management via classes and informal interactions.

Cells 13G,J - A "Domain Analysis" of the SEAL software application domain(s) is being conducted to reveal the commonalities between development projects. This is a deliverable under a task being conducted by the SEAL for the Code QE Software Engineering Program.

Cells 15E,H - The SEAL is cooperating with LaRC hardware and systems engineering branches to document a systems engineering approach that includes participation of software engineers in the earliest stages. The SEAL advocates limiting hardware choices, such as buses and microprocessors, to selections from a small set of agreed upon standards. This will further promote reuse of software components.

Cells 17G,H,I,J - These are addressed in Asset cells 23G,H,I,J. The referenced guidebooks will also cover the management and assurance processes.

Tool 4: RBSE

Cell 8E - RBSE participates in the Reuse Acquisition Action Team, a group which is focused on management/acquisition issues of reuse. It is sponsored by the ACM/SIGAda Reuse Working Group. The group has strong support from DoD's Executive Reuse Steering Committee and acquisition/policy officers from Army, Navy and Air Force.

Cell 10E - RAAT (See 8E)

Cell 13E - RBSE is active in developing the Software Engineering Institute's "Design for Reuse Handbook." RBSE sponsored a workshop earlier this year at the University of Houston, Clear Lake.

Cells 13-24J - AdaNET provides information about a range of reuse-related technical and non-technical issues. Information on these and other topics may be available.

Cell 20E - RAAT (See 8E)

Tool 51: LEARN-92

Cell 13 - LEARN-92 is an experimental project that is using conceptual clustering techniques from artificial intelligence to automatically develop a classification scheme for code components. This capability would support the domain engineer, who must create a classification scheme for components as part of the domain model. A prototype version of the tool is planned to be completed by the end of September 1992.

Cell 17 - LEARN-92 is intended to provide the software engineer with a classification of components based on their role in the problem space (i.e., what problem they solve rather than the solution space (how they are implemented). The inheritance hierarchy of an object-oriented programming system, such as C++, provides a solution-space organization: this is often not very helpful to programmers who are searching for a reusable component to perform a specific function.

Tool 52: KAPTUR

Cell 13 - KAPTUR supports an approach to domain engineering in which assets are organized in terms of their *distinctive features*, which represent key engineering decisions and which are justified by *rationales*. The approach is also distinguished by the fact that it is *case-based*, i.e., actual legacy products are included in the database, not just generic models for future use.

KAPTUR 2.0 has been released, following version 1.0 and two earlier prototypes. The system is currently being distributed to interested/potential users, and a training course on KAPTUR and Domain Analysis is being developed.

Tool 53: KBSEE

Cell 13 - The KBSEE is a prototype environment intended to support domain engineering; in particular, the production of new systems by configuring generic assets stored in a domain model. It is based on an evolutionary concept of domain engineering, in which new systems are used to update the domain model to make it more responsive to future requirements. A prototype KBSEE has been developed, and its feasibility is now being tested in the Payload Operations Control Center domain.

Cell 17 - The KBSEE makes reuse the central activity of the software engineering process. Development is seen as a process of identifying the required features of a new system, retrieving the assets possessing those features from the generic domain model, checking the mutual consistency of the assets, and configuring them into the new system. Specification of the required features is done by the developer; all the other steps are performed by the KBSEE.

Tool 6: Johnson Space Center Tools

Cell 13 - Domain Engineering - Two projects, ESL and SimTool, are investigating various aspects of domain architectures and reuse, and are discovering implications for the domain engineering process.

Cell 17 - Software Engineering - Three projects, REAP, FPP, and ESL, are addressing aspects of the software engineering process:

(i) REAP (Re-engineering Application Project) is developing an integrated re-engineering environment, including methods and tools.

(ii) FPP (Framework Programmable Platform) is focusing on the description, management, and control of the software development process within an integrated life-cycle environment.

(iii) ESL (Engineering Script Language) is a graphical language for composing complete applications from software parts in a reusable library, and it is investigating a process for composing applications.

TECHNOLOGY NOTES

Tool 1: Eli/InQuisiX

Cells 13, 15, 17, 19;E-G - The Eli (InQuisiX (TM) Software Synthesis System includes a graphical user interface and a library system. The library system supports classification, retrieval and management of components. InQuisiX was developed under an SBIR; the company is preparing a commercial product.

Cells 22E,F - Identify a set of measurable reuse attributes for object-oriented systems and design a prototype tool to take these measurements.

Tool 2: HyLite

Cells 6F,G - Applying object-oriented DBMS methods for software reuse.

Cells 13F,G - Applying hypermedia technology.

Cells 15-19 F,G - Applying AI techniques for navigation in databases.

Tool 3: SEAL

Cells 6E-K - An object-based design methodology has been defined in the SEAL. Applied to a flight software project and published in several papers. The guidebooks of Asset Cell 23G will define a suite of object-oriented methods to be used in the SEAL for analysis, design, and implementation. Training in these chosen methods will be given at LaRC. The SEAL provides feedback to software development tool vendors about features that are desirable.

Cells 8G-J - Many of the code modules developed in the SEAL are in the form of reusable, generic, Ada packages. Ada has been adopted as the development language for the SEAL. SEAL guidebooks for developing reusable Ada components/systems (See Asset 19G) will be the basis of reuse training for new personnel. The generic Ada packages will be made widely available via asset repositories such as COSMIC and AdaNET.

Cells 10F,G,I – The domain analysis of Process 13G will identify the feasibility of megaprogramming in our domain by determining the common building blocks in our systems. New systems will be megaprogrammed from existing reusable assets, which have been designed with standard protocols, methodologies, and hardware in mind.

Cells 15E,G – The domain analysis of Process 13G will identify attributes and facets of our domains that will enable us to develop classification schema for our reusable assets. These schema will be initially implemented using the ELI/ARCS reuse tool system developed under a LaRC SBIR.

Cell 24E – The SEAL will be identifying metrics to measure all aspects of the software development process, including reuse activities. These will be formalized in the guidebooks of Asset 23G.

Tool 4: RBSE

Cells 6-24J – AdaNET provides information about a range of reuse-related technical and non-technical issues. Information on these and other topics may be available.

Cell 13E – Trade study

Cell 13F – Feasibility study

Cell 13G – RBSE's operational reuse library component, AdaNET, has developed and operated a prototype reuse library. The system is to be upgraded this fall. System will include X-windows, MAC, and PC-based GUI.

Cell 15E – RBSE has sponsored work by Dr. David Eichmann and others to develop lattice-based classification schemes of reuse libraries.

Cell 15G – AdaNET (see 13G).

Cell 17G – See AdaNET (see 13G).

Cell 19E – RBSE provides active support and leadership to the Reuse Library Interoperability Group, an organization developing consensus-based standards for interoperability among government-funded reuse libraries.

Cell 19F - RBSE is holding discussions with another reuse library to prototype interchange of assets.

Cell 19H - RIG (see 19E).

Cell 22E - RBSE has conducted trade studies on certification metrics.

Cell 22F - RBSE is evaluating the feasibility of certification metrics with off-the-shelf tools.

Cell 24E - RBSE co-chairs the RIG technical subcommittee on metrics.

Tool 51: LEARN-92

Cell 15 - LEARN-92 will provide an automated mechanism for hierarchical classification of code components, based on faceted descriptions of these components. A unique aspect of the faceted descriptions is that the facet space is extendible "on the fly" by the user who is placing a component into the system. The user is encouraged, but not required, to use existing facets in describing a new component.

Tool 52: KAPTUR

Cell 13 - KAPTUR employs a novel user interface approach which is based more on direct display and manipulation of the database rather than queries.

Cell 15 - KAPTUR's approach to asset classification uses a typing structure including both domain-independent and domain-dependent asset types. Within a type, assets are classified on the basis of their features. The KAPTUR concept of feature is broader than that found in the Software Engineering Institute's Feature-Oriented Domain Analysis (FODA) method.

Tool 53: KBSEE

Cell 8 - The domain model, as stored by the KBSEE, consists of a hierarchy of generic assets, each of which possesses certain features that make it suitable or unsuitable for a given application. The generic assets are created through the process of Domain Analysis, which abstracts the functionality found in existing and planned systems in the domain.

Cell 10 - The highly automated concept of software development supported by the KBSEE makes it suitable for megaprogramming. The development process consists of configuring assets each of which can each represent a complex capability in its own right.

Cell 15 - Assets are organized into whole-part and class/subclass hierarchies. In addition, assets possess features (similar to the notion of feature in the Software Engineering Institute's FODA method), which are used to determine which assets should be retrieved to meet the requirements of a new system. Features are described as *mandatory* (must be present in any system), *variant* (one of several variants must be present in any system), or *optional* (may or may not be present).

Tool 6: Johnson Space Center Tools

Cell 6 - Object-oriented methods: one project, re-engineering the Mission Operations Computer to an object-oriented design, is evaluating the feasibility of using object-oriented technology in a previously assembly-language, mega-system domain.

Cell 10 - Megaprogramming: ESL is investigating exactly this type of problem, and an entire prototype has been built and is being tested.

Cells 13-19 - Libraries: NELS (NASA Electronic Library System) and RBSE (Repository-Based Software Engineering) are related projects that are building a reuse library system that addresses many of the areas on this chart.

ASSETS NOTES

Tool 1: Eli/InQuisiX

Cells 6-16, E-G - The InQuisiX system supports the reuse of many types of components including: designs, code, documentation and test procedures.

Tool 2: HyLite

Cells 6-14, F-G - Developing versatile system that can be used to manage and reuse these types of assets.

Tool 3: SEAL

Cells 6G,H,J -16G,H,J - The SEAL has adopted an "expansive" view of reuse, where all products of the life cycle may be reused and composed of reusable products. Assets will be developed following pertinent software, hardware, communications, and user interface standards. Documentation will follow the NASA Software Documentation Standard. All assets will be made widely available via asset repositories.

Cells 19G,J - A guidebook for developing reusable Ada components and systems will be developed by the SEAL. This is a deliverable under a task being conducted by the SEAL for the Code QE Software Engineering Program.

Cells 21G,J - A guidebook for transferring reusable Ada software in NASA will be developed by the SEAL. This is a deliverable under a task being conducted by the SEAL for the Code QE Software Engineering Program.

Cells 23G-J - Tailorable software engineering process guidebooks are being developed for the various SEAL software domains. These guidebooks will incorporate standard, existing methodologies and tools as much as possible. Future training for SEAL and other LaRC personnel will be tailored to these guidebooks. These guidebooks are deliverables under a task being conducted by the SEAL for the Code QE Software Engineering Program.

Additionally, an annual SEAL report is planned that will assess the scope, development processes, and transfer mechanisms for reuse of software products for NASA Ada projects.

Tool 4: RBSE

Cell 6J - AdaNET (see Technology 13G).

Cell 8J - AdaNET (see Technology 13G).

Cell 10J - AdaNET (see Technology 13G).

Cell 12J - AdaNET (see Technology 13G).

Cell 14J - AdaNET (see Technology 13G).

Cell 16J - AdaNET (see Technology 13G).

Tool 51: LEARN-92

Cell 10 - The focus in this effort is on code components, but the classification mechanisms being implemented in LEARN-92 could work for other forms of assets as well. The emphasis is due to a current need within GSFC/Code 520, where there is a growing collection of reusable C++ components being circulated among developers, and a need to organize the components in a form that makes it easy to locate reusable code.

Tool 52: KAPTUR

Cell 21 - KAPTUR provides a mechanism for the rationales for various engineering decisions to be recorded. These can include after-the-fact lessons learned from the particular decisions made.

Tool 53: KBSEE

Cell 6 - The KBSEE effort has focused to date on the storage of generic requirements specifications and the automated configuration of requirements specifications for new systems based on the generic versions. The methodology encompasses other life-cycle products as well.

3. Proposed Actions

During the wrap-up session, the workshop participants discussed ways to leverage their individual software reuse activities into a coordinated program to address NASA's software development needs and to promote software reuse as an integral part of the NASA software development process. The participants concluded that these objectives can be accomplished by coordinating their software reuse activities and marketing their activities to NASA Headquarters as a coordinated, focused program to advance software reuse throughout the NASA community. The following preliminary action items were agreed upon:

- Use this workshop document as the basis for a proposal to potential sponsors.
- Form a Software Engineering and Reuse Team focusing on NASA problems. This team is to be led by either LaRC or ARC. Team members are to include ARC, LaRC, LeRC, GSFC, JSC, JPL, MSFC, HQ, Rome Laboratory (Air Force), COSMIC, DARPA (ASSET), RBSE. This team should combine with SATWG/
SAAP Software Engineering Subpanel, chaired by E. Fridge of JSC.
- Determine customer needs for the near term. This will be accomplished by looking at existing advocacy packages, by presenting current software reuse activities to HQ, and by soliciting feedback from HQ.
- Use Code R Block Grants as a mechanism to influence software reuse in university curricula. Candidates are University of Illinois/Urbana-Champaign, Stanford University, University of Maryland, and Harvey Mudd College.

References

[1] James W. Hooper and Rowena O. Chester. *Software Reuse: Guidelines and Methods.* Plenum Press, New York, 1991.

[2] G. Caldiera and V. R. Basili. Identifying and Qualifying Reusable Software Components. *Computer*, 24:61–70, February 1991.

[3] E. J. Joyce. Reusable Software: Passage to Productivity? *Datamation*, 34:97–98, September 15 1988.

[4] B. J. Shelburne and M. J. Pitarys. Avionics Software Reusability Observations and Recommendations. In *Proceedings of NAECON*, pages 614–619, 1991.

[5] T. Biggerstaff and C. Richter. Reusability Framework, Assessment, and Directions. In *Proceedings of the Twentieth Annual Hawaii International Conference on System Sciences*, 1987.

[6] S. J. Voight and K. A. Smith. Software Reuse Issues. Conference Publication 3057, NASA, November 17-18 1988. Proceedings of a Workshop held in Melbourne, Florida.

APPENDIX A: Workshop Attendees

Wayne Bryant
NASA Langley Research Center
Mail Stop 478
Hampton, VA 23665-5225
804-864-1692
wayne@uxv.larc.nasa.gov

Kathryn Smith
NASA Langley Research Center
Mail Stop 478
Hampton, VA 23665-5225
804-864-1699
kas@csab.larc.nasa.gov

Floyd Shipman
NASA Langley Research Center
Mail Stop 478
Hampton, VA 2366-5225
804-864-1706
shipman@scab.larc.nasa.gov

Randy VanValkenburg
NASA Langley Research Center
Mail Stop 125A
Hampton, VA 23665-5225
804-864-7933
vanvalke@voyager.larc.nasa.gov

Edward Ng
Jet Propulsion Laboratory
CIT
4800 Oak Grove Dr.
Pasadena, CA 91109
818-306-6166
MS 525-3660
edng@nasamail
edward_ng@isd.jpl.nasa.gov

Jim Golej
MITRE Corporation
Mail Code PT41
Houston, TX 77058
713-333-5020

Deborah Cerino
Rome Laboratory/C3CB
Griffiss AF Base, NY 13441
315-330-2054
cerino@softvax.rl.af.mil

Sidney Bailin
CTA, Inc.
6116 Executive Boulevard
Rockville, MD 20852
301-816-1451
sbailin@cta.com

Barry E. Jacobs
Code 934
Goddard Space Flight Center
Greenbelt, MD 20771
301-286-5661
bjacobs@nssdc.gsfc.nasa.gov

Scott Clark
COSMIC
University of Georgia
382 East Broad St.
Athens, GA 30602
404-542-3265
scott@cosmic1.cosmic.uga.edu

David Dikel
Applied Expertise
1925 N. Lynn St.
Suite 802
Arlington, VA 22209
703-516-0911
ddikel@ajpo.sei.cmu.edu

Charles W. Lillie
SAIC
1710 Goodridge Dr.
McLean, VA 22102
703-749-8732
lilliec@source.asset.com

Jack Tupman
Jet Propulsion Laboratory
CIT
4800 Oak Grove Dr.
Pasadena, CA 91109
818-306-6182
MS 525-3660
jack@jade.jpl.nasa.gov

Joe Jupin
Jet Propulsion Laboratory
CIT
4800 Oak Grove Dr.
Pasadena, CA 91109
818-306-6161
MS 525-3660
joe_jupin@isd.jpl.nasa.gov

Charlotte Scheper
Research Triangle Institute
P.O. Box 12194
Research Triangle Park, NC 27709
919-541-7116
cos@rti.org

Janet Dunham
Research Triangle Institute
P.O. Box 12194
Research Triangle Park, NC 27709
919-541-6562
jrd@rti.org

Gail Loveland
Research Triangle Institute
P.O. Box 12194
Research Triangle Park, NC 27709
919-541-6330
gl@rti.org

Larry Preheim
Jet Propulsion Laboratory
CIT
4800 Oak Grove Dr.
Pasadena, CA 91109
818-306-6042
MS 525-3660
lpreheim@jpllsi.jpl.nasa.gov

Robert Baker
Research Triangle Institute
P.O. Box 12194
Research Triangle Park, NC 27709
919-541-7401
rlb@rti.org

Dave McLin
Research Triangle Institute
P.O. Box 12194
Research Triangle Park, NC 27709
919-541-5828
dmm@rti.org

Ed Withers
Research Triangle Institute
P.O. Box 12194
Research Triangle Park, NC 27709
919-541-6311
bew@rti.org

Appendix B: Viewgraphs Presented at Workshop

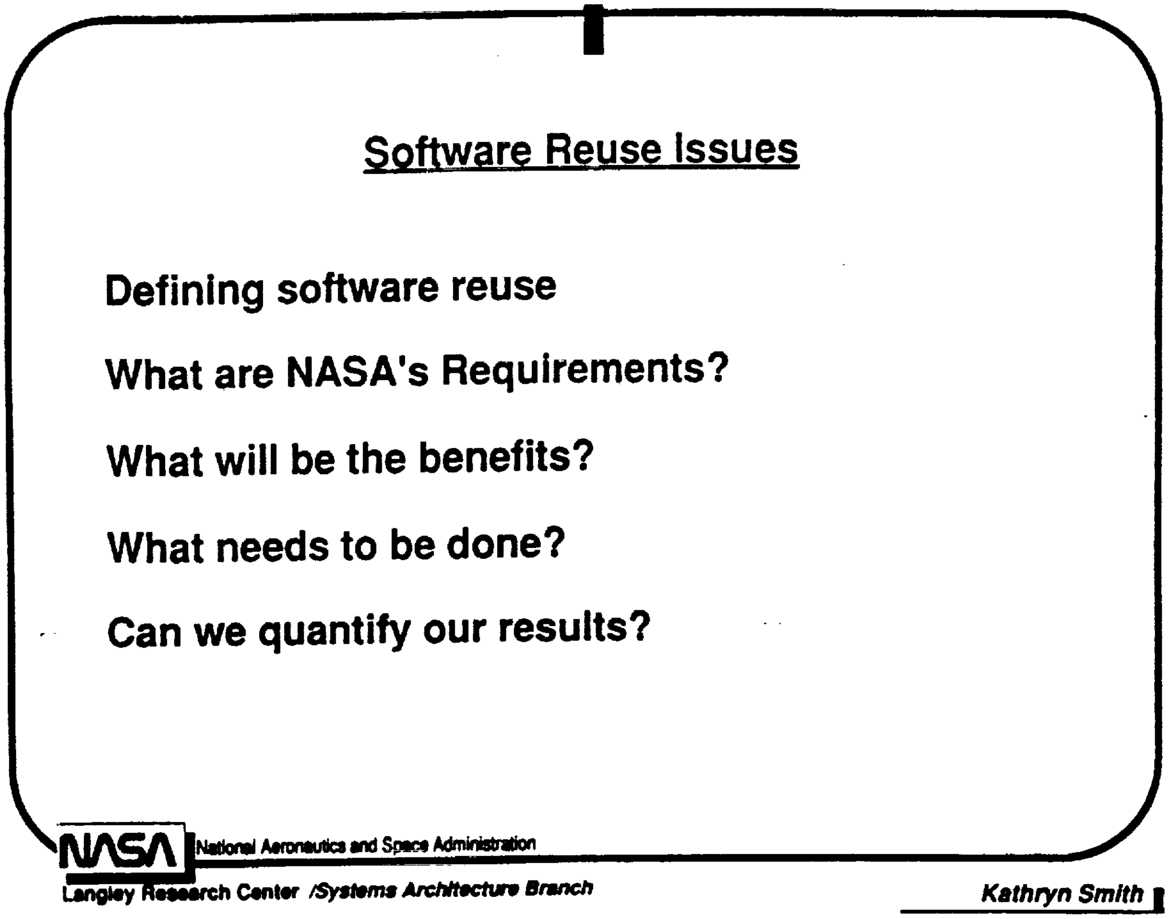
Software Reuse Tools Workshop

Workshop Objectives

Exchange Information on
- Software reuse tool development
- Software reuse tool needs, requirements and effectiveness

Identify critical issues and needs in software reuse

Identify opportunities for cooperative and collaborative research



Software Reuse Issues

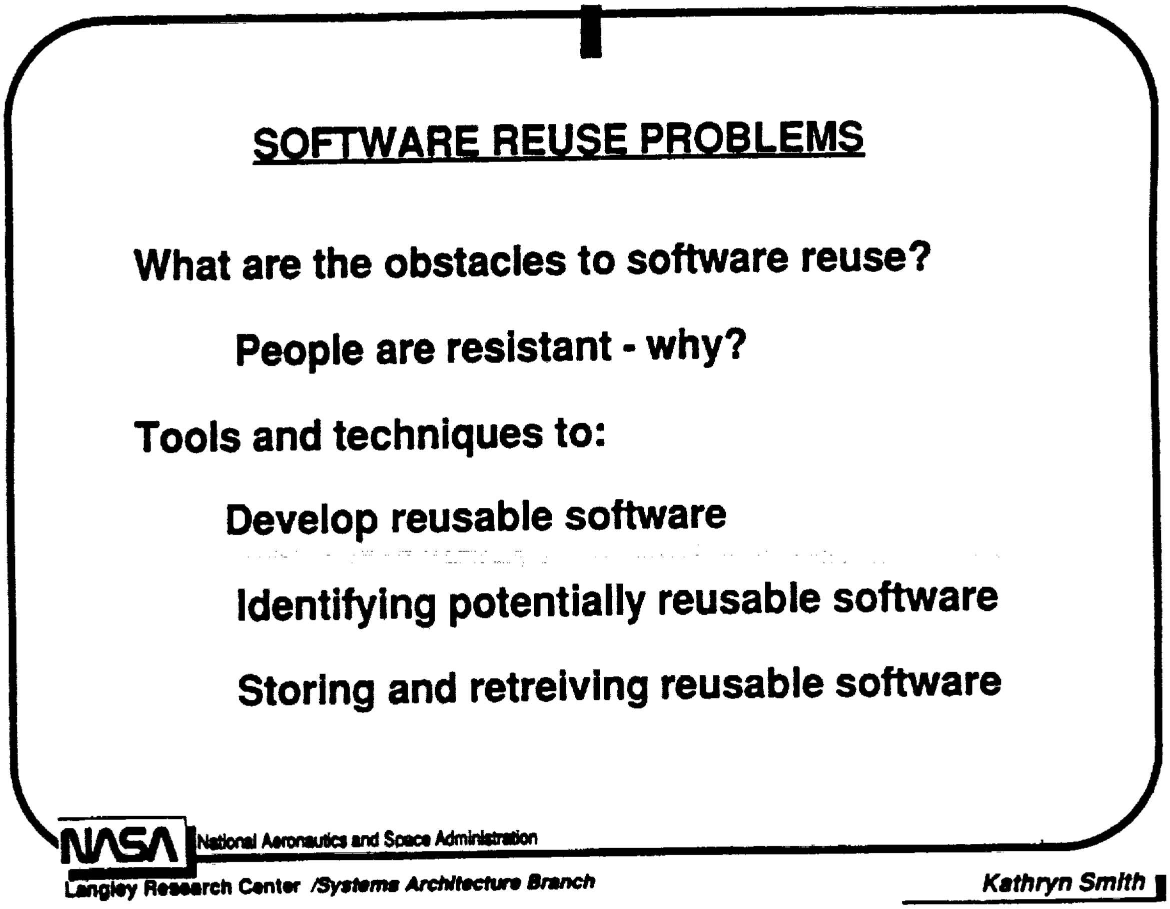
Defining software reuse

What are NASA's Requirements?

What will be the benefits?

What needs to be done?

Can we quantify our results?



SOFTWARE REUSE PROBLEMS

What are the obstacles to software reuse?

People are resistant - why?

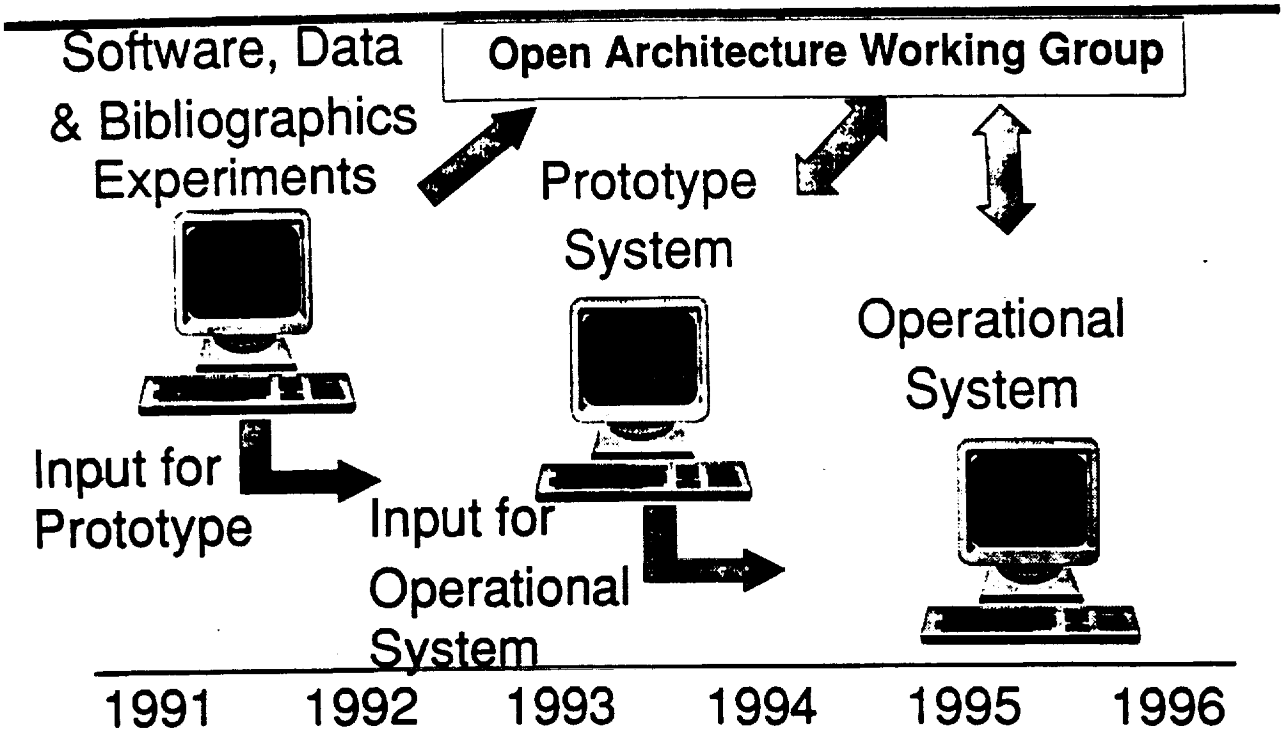
Tools and techniques to:

Develop reusable software

Identifying potentially reusable software

Storing and retreiving reusable software

HPCC Software Sharing - Schedule



Software, Data & Bibliographics Experiments

Open Architecture Working Group

Prototype System

Operational System

Input for Prototype

Input for Operational System

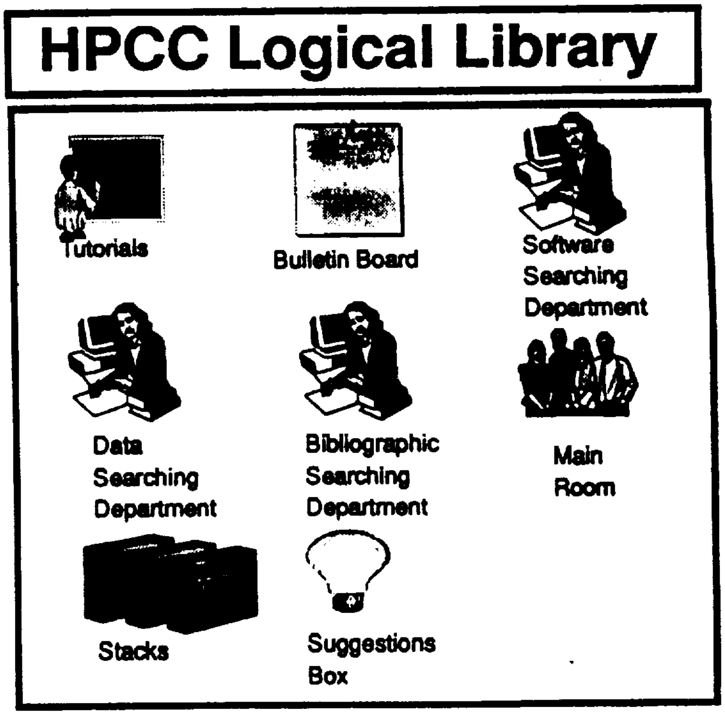
1991 1992 1993 1994 1995 1996

HPCC Software Sharing

HPCC Software Sharing Experiment- Logical Library

All HPCC participating organizations appear as part of one large "logical library".

HPCC Logical Library

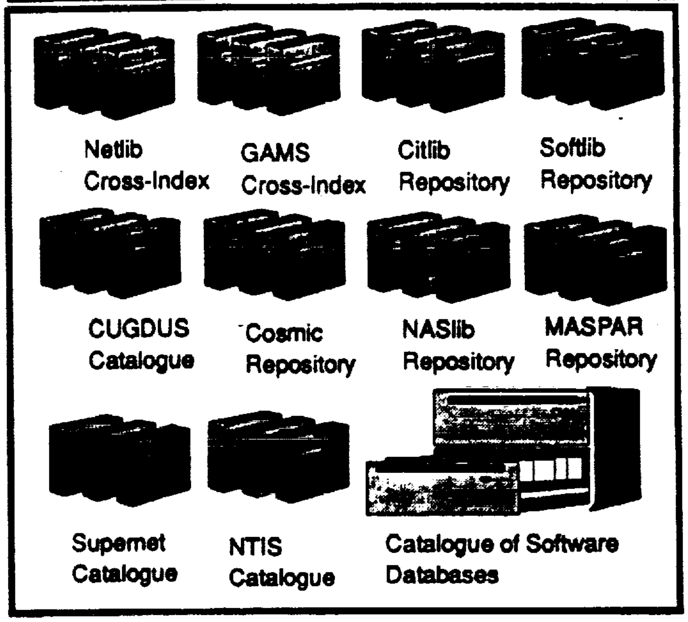

Tutorials
Bulletin Board
Software Searching Department
Data Searching Department
Bibliographic Searching Department
Main Room
Stacks
Suggestions Box

HPCC Software Sharing

HPCC Software Sharing Experiment- Software Shelf

All Software Databases are accessible either on the shelf or via the Catalogue of Software Databases.

Software Shelf



Netlib Cross-Index

GAMS Cross-Index

Citlib Repository

Softlib Repository

CUGDUS Catalogue

Cosmic Repository

NASlib Repository

MASPAR Repository

Supernet Catalogue

NTIS Catalogue

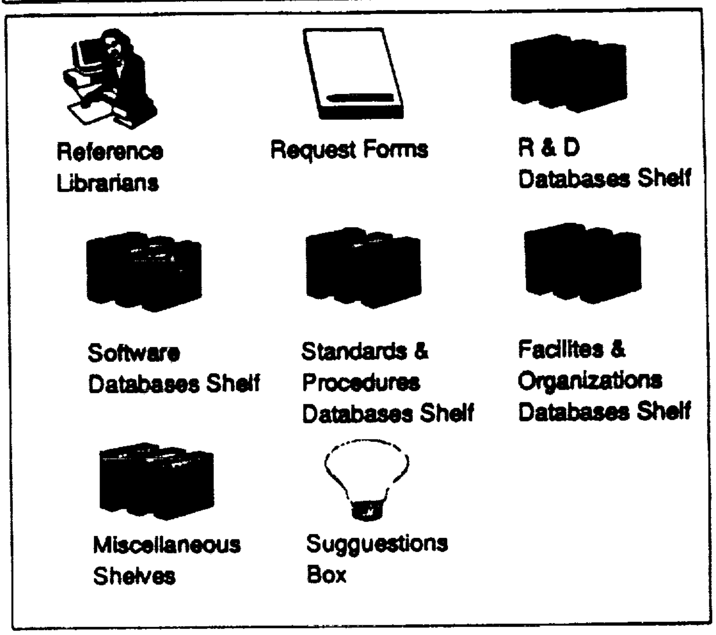
Catalogue of Software Databases

HPCC Software Sharing

HPCC Software Sharing Experiment- Software Searching Department

Software Searching Department helps users locate relevant software.

Software Searching Department



Reference Librarians

Request Forms

R & D Databases Shelf

Software Databases Shelf

Standards & Procedures Databases Shelf

Facilites & Organizations Databases Shelf

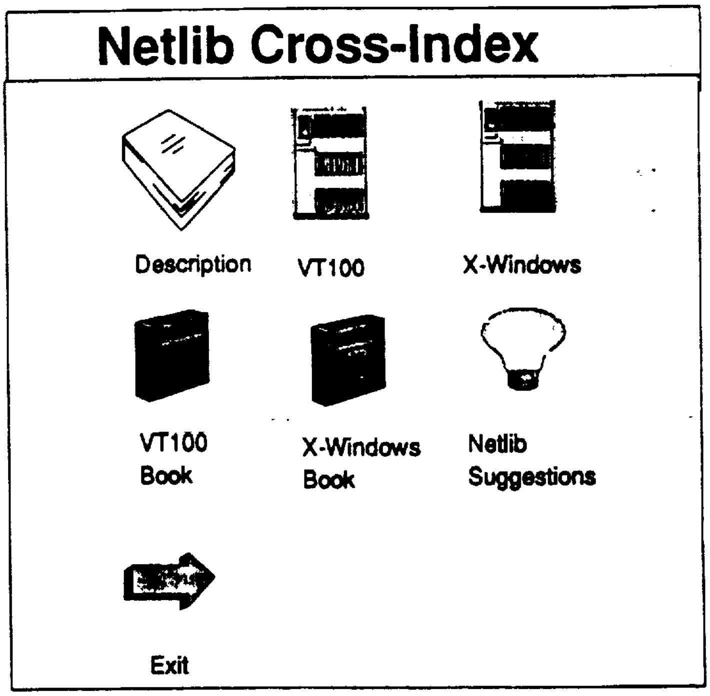
Miscellaneous Shelves

Sugguestions Box

HPCC Software Sharing

HPCC Software Sharing Experiment- Software Databases

All Logical Library holdings may have multiple user interfaces.



KAPTUR

Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationales

A Tool for Preserving and Building on Engineering Legacy

Presented by:

Sidney C. Baliin
CTA Incorporated
6116 Executive Boulevard, Suite 800
Rockville, MD 20852
(301) 816 - 1200
sbailin @ cta.com

CTA INCORPORATED

WHAT IS KAPTUR?

- KAPTUR is a tool designed to be part of a reuse-based software development environment.
- KAPTUR has gone through two phases of prototyping:
  - KAPTUR '89
  - KAPTUR '90
- Efforts are underway to bring the tool from a laboratory environment to software developers.
  - KAPTUR 1.0

CTA INCORPORATED

KAPTUR GOALS AND OBJECTIVES

SUPPORT REUSE OF SOFTWARE ASSETS

- Capture engineering decisions/rationales that went into their development

PROVIDE AN EASY TO USE ENVIRONMENT FOR ACCESSING CAPTURED KNOWLEDGE

APPLY THE ENVIRONMENT TO SUPPORT SOFTWARE REUSE IN SMEX MISSIONS

CTA INCORPORATED

RATIONALE / BENEFITS

COST SAVINGS THROUGH INCREASED SOFTWARE REUSE

RETENTION OF SOFTWARE ENGINEER KNOWLEDGE AND EXPERIENCE IN A DATABASE ACCESSIBLE TO OTHERS

IMPROVEMENT OF DEVELOPMENT PROCESS BY BUILDING ON PAST EXPERIENCE AND LESSONS LEARNED

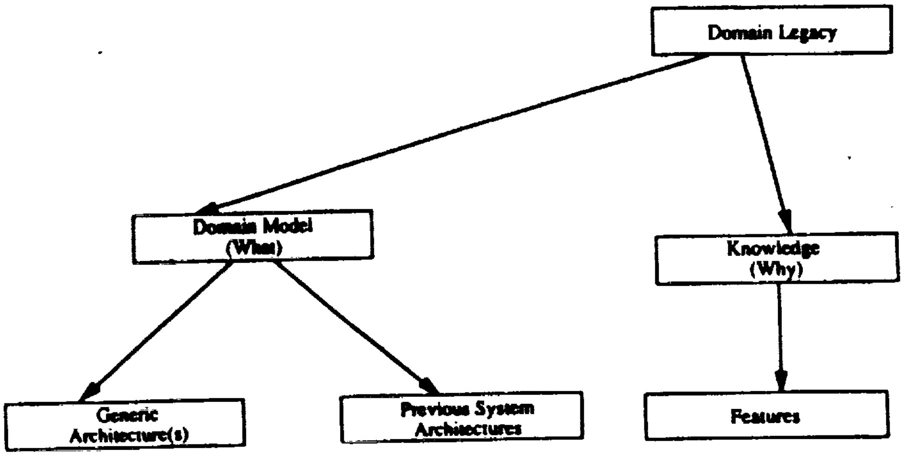
CTA INCORPORATED

HOW DOES KAPTUR ASSIST IN REUSE?

- KAPTUR handles more than code components.
  - — requirements
  - — design
  - — test (planned)
- KAPTUR keeps a <u>representation</u> of components and <u>knowledge</u> that would assist in determining which particular components to reuse.
- Components themselves aren't kept in KAPTUR.
- KAPTUR provides information on where the components are kept (not implemented).

CTA INCORPORATED

WHAT FORM DO COMPONENT REPRESENTATION AND KNOWLEDGE TAKE?



- Multiple Views

- Entity Relationship Diagram
- Classification Diagram

•
•
•

- Decisions
- Tradeoffs
- Rationales

KAPTUR not only stores representations of systems, but also stores key development decisions and the reasons behind the decisions.

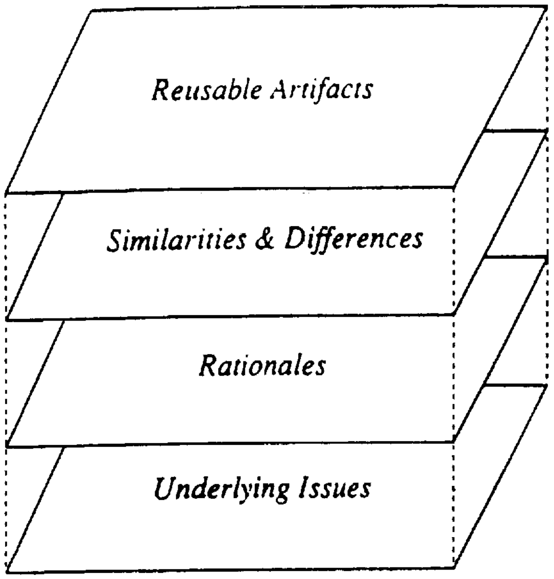
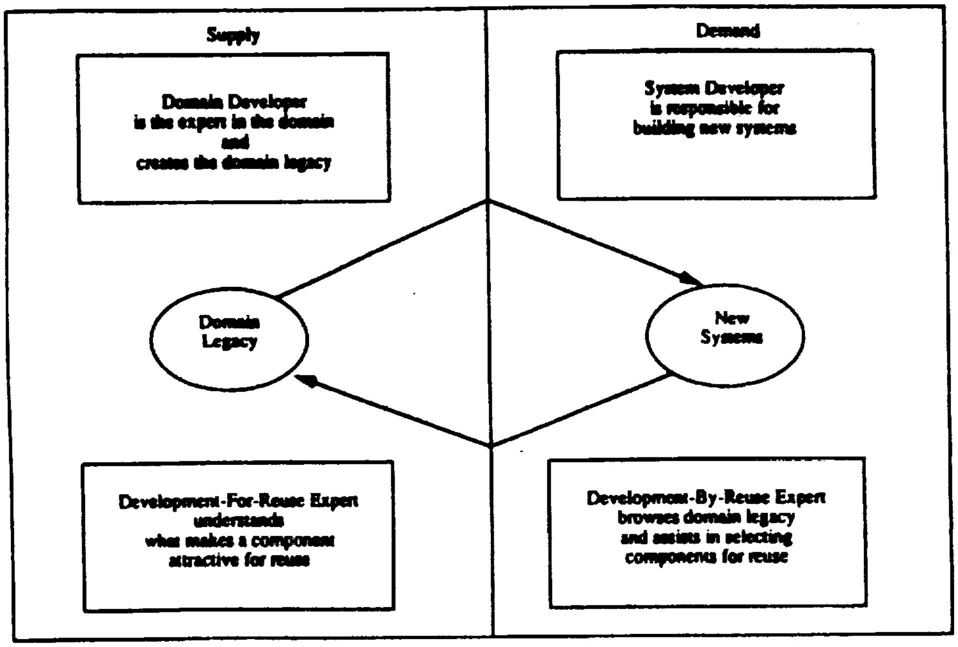


Figure 4-1: Layers in KAPTUR's Knowledge Base

HOW IS THE REPRESENTATION AND KNOWLEDGE CREATED AND USED?


Supply

Demand

Domain Developer
is the expert in the domain
and
creates the domain legacy

System Developer
is responsible for
building new systems

Domain Legacy

New Systems

Development-For-Reuse Expert
understands
what makes a component
attractive for reuse

Development-By-Reuse Expert
browses domain legacy
and assists in selecting
components for reuse

The continuous feedback loop between the supply
and the demand side is the key to reuse

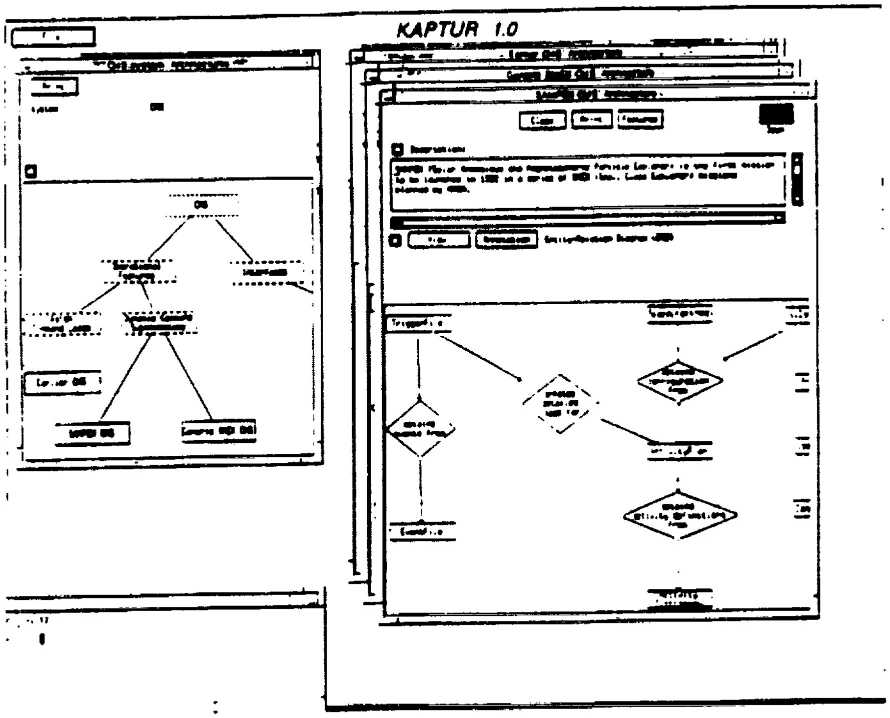
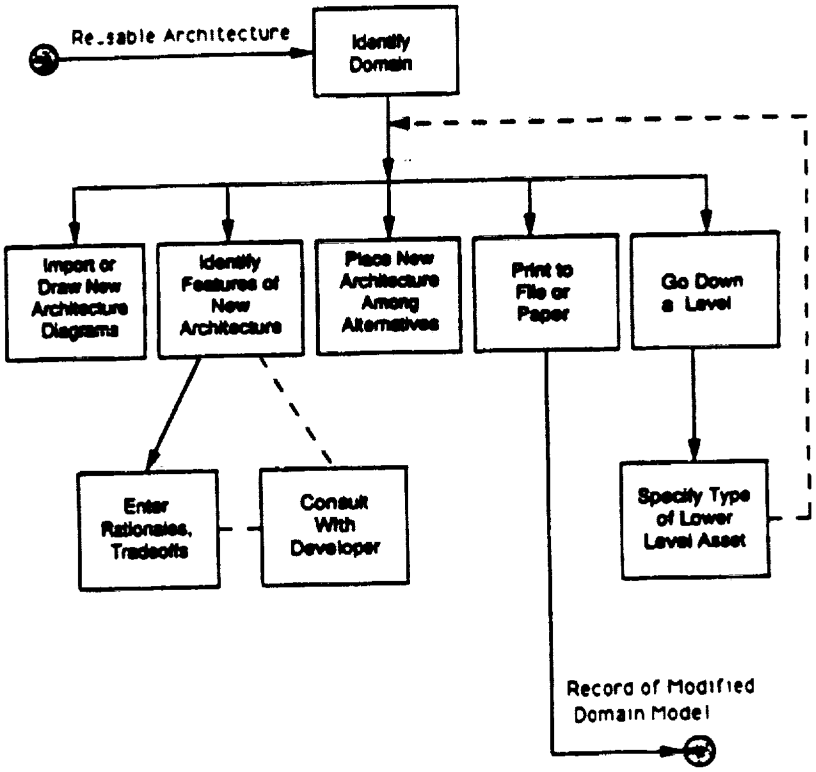
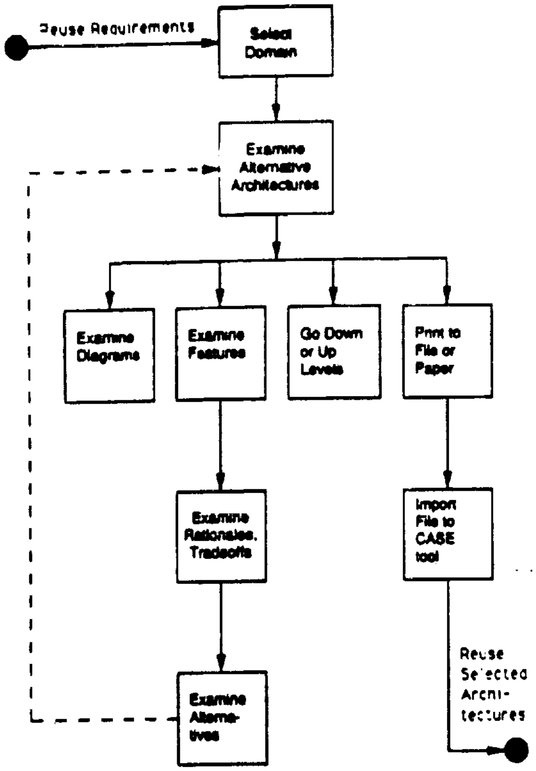


Figure 2: The Basic KAPTUR Screen Provides a Hierarchical Map of Alternatives and a Stack of Pages Describing Them


Re_sable Architecture
Identify Domain
Import or Draw New Architecture Diagrams
Identify Features of New Architecture
Place New Architecture Among Alternatives
Print to File or Paper
Go Down a Level
Enter Rationales, Tradeoffs
Consult With Developer
Specify Type of Lower Level Asset
Record of Modified Domain Model

The KAPTUR Supply Side Supports the Entering of New Architectures, Recording of Rationales, and Placement of Them Within the Context of an Overall Domain Model



The KAPTUR Demand Side Supports Browsing of Alternatives, Understanding Decisions, and Selecting for Reuse

KEY CONCEPTS IN KAPTUR

ASSET

- **Any software product that can be reused in future developments**
- **Includes systems, subsystems, objects, functions**

ARCHITECTURE

- **A description of the structure of a software asset**
- **Uses one or more graphical views**

GENERIC ARCHITECTURE

- **An architecture that can be instantiated or tailored to meet varying requirements**

DISTINCTIVE FEATURE

- **Any significant way in which an architecture differs from its alternatives**
- **The way in which an asset manifests a significant engineering decision**

KEY CONCEPTS IN KAPTUR (continued)

ALTERNATIVES MAP

- A hierarchical description of alternative architectures for a given type of asset

DOMAIN MODEL

- The legacy of knowledge about an application domain
- Packaged for easy access and reuse

SUPPLY SIDE

- The creation/maintenance of the domain model
- Incorporation of new assets as they are developed, with features and rationales

DEMAND SIDE

- Access to the domain model for the purpose of reusing the assets it contains

KAPTUR ENVIRONMENT DETAILS

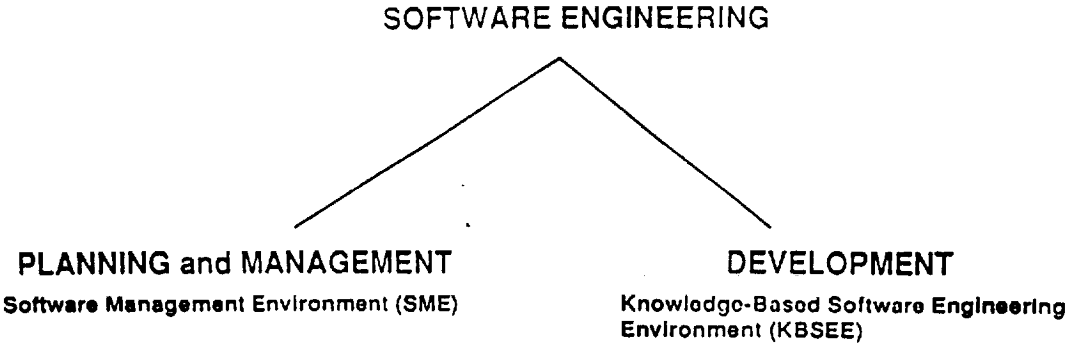
WRITTEN IN C

- Approximately 45% of code is automatically generated

USES TAE+ VERSION 5

CURRENTLY RUNS ON A SUN SPARCSTATION

- Should run on any UNIX system supporting TAE



SOFTWARE ENGINEERING

PLANNING and MANAGEMENT

Software Management Environment (SME)

DEVELOPMENT

Knowledge-Based Software Engineering Environment (KBSEE)

The Software Engineering work addresses a full spectrum of activities needed to:

1. plan, manage, and monitor the development of, and
2. provide for the efficient and effective implementation of

complex operational systems.

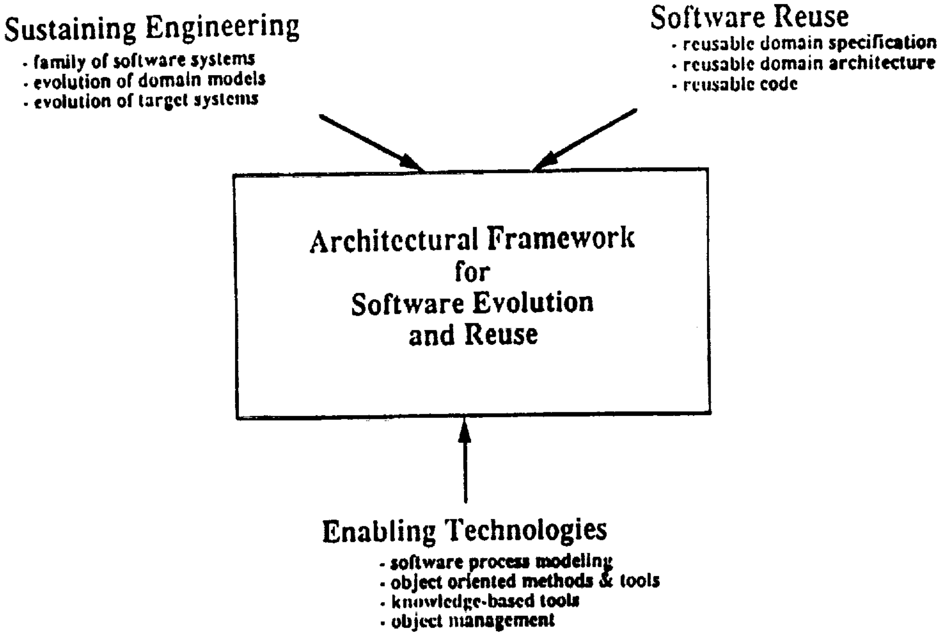
Basic
Knowledge-Based Software Engineering Environment

(KBSEE)

- Incorporates the Evolutionary Domain Life-Cycle (EDLC) Model

GOALS OF THE RESEARCH:

IN RESPONSE TO NASA SOFTWARE ENGINEERING INITIATIVE



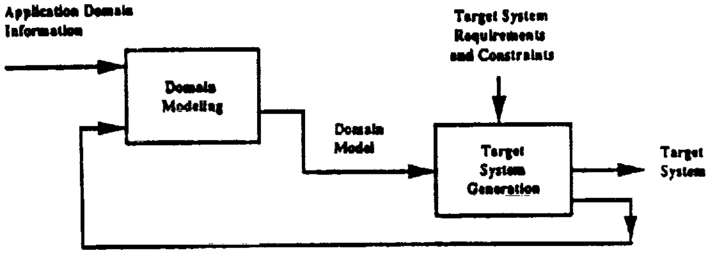
Sustaining Engineering
- family of software systems
- evolution of domain models
- evolution of target systems

Software Reuse
- reusable domain specification
- reusable domain architecture
- reusable code

Architectural Framework for Software Evolution and Reuse

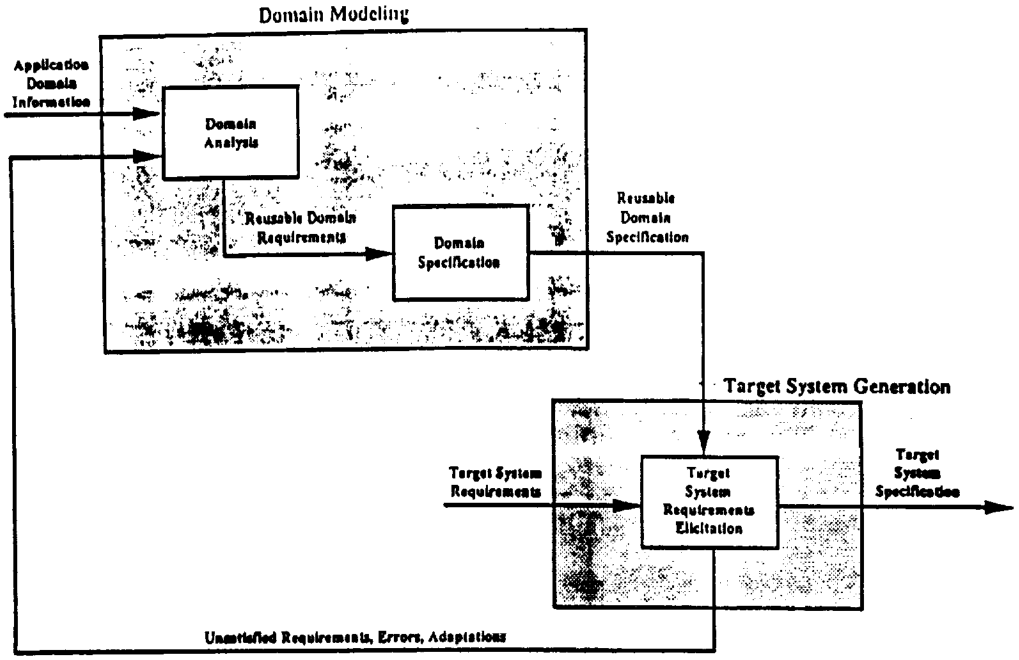
Enabling Technologies
- software process modeling
- object oriented methods & tools
- knowledge-based tools
- object management

EVOLUTIONARY DOMAIN LIFE CYCLE


Application Domain Information
Domain Modeling
Domain Model
Target System Requirements and Constraints
Target System Generation
Target System

- Makes no distinction between development and maintenance
- System viewed as evolving through several iterations
- Life-cycle for family of systems

PROOF-OF-CONCEPT EXPERIMENT


Domain Modeling
Application Domain Information
Domain Analysis
Reusable Domain Requirements
Domain Specification
Reusable Domain Specification
Target System Generation
Target System Requirements
Target System Requirements Elicitation
Target System Specification
Unsatisfied Requirements, Errors, Adaptations

EDLC PROOF-OF-CONCEPT EXPERIMENT (EPOC) GOALS

Demonstrate viability of Evolutionary Domain Life Cycle Approach

Create demonstration version of Knowledge Based Software Engineering Environment supporting EDLC

- Domain Modeling
  - Address Domain Analysis and Specification
- Target System Generation
  - Address Knowledge Based Requirements Elicitation

EDLC PROOF-OF-CONCEPT EXPERIMENT (EPOC) FEATURES

Tool Support for Developing Domain Specification

- Provide support for Domain Analysis and Specification Method
- Create multiple view graphical representation

Store Domain Specification

- Map multiple views to common underlying representation
- Store in object repository

Multiple View Consistency Checking

- Determine whether Domain Specification rules obeyed

Generate Target System Specification

- Tailored version of Domain Specification
- Knowledge Based Requirements Elicitation

EDLC PROOF-OF-CONCEPT EXPERIMENT (EPOC) APPROACH

Off-the-shelf CASE tools where appropriate

- Software Through Pictures (StP)
  - Provides graphical front end
  - Open systems architecture

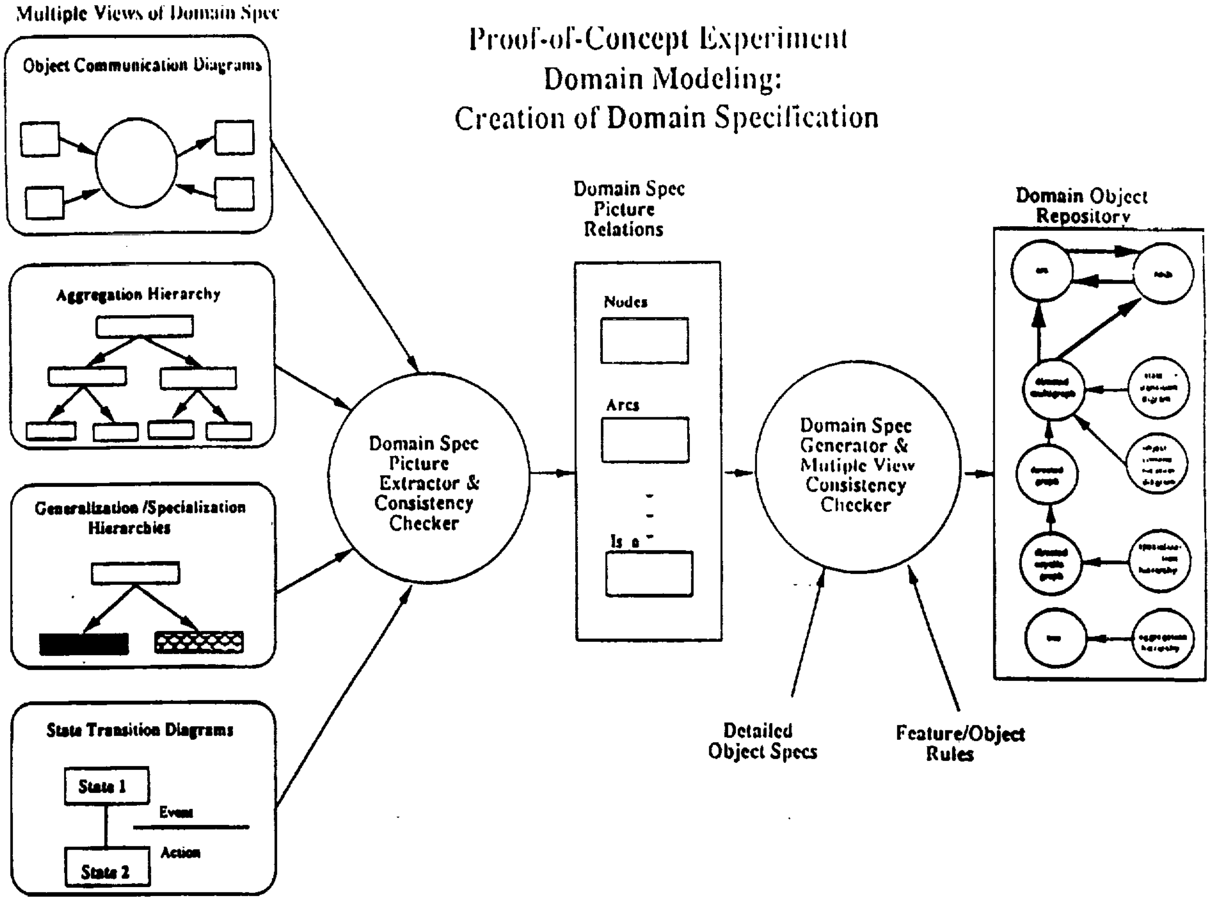
Object Oriented Programming Language Support

- Eiffel Language
  - Compiler and component library
  - Persistent object store

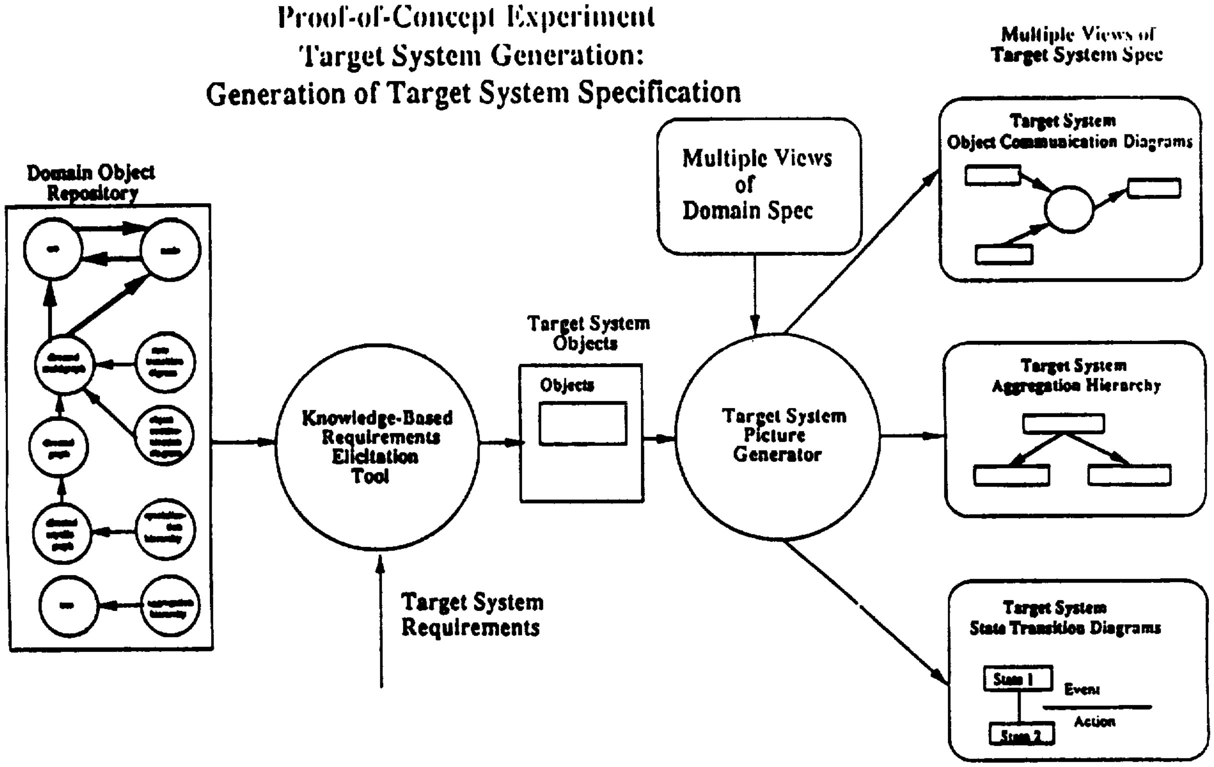
Investigate NASA developed tools where appropriate

- TAE User Interface Management System
- CLIPS knowledge based system shell

Proof-of-Concept Experiment
Domain Modeling:
Creation of Domain Specification


Multiple Views of Domain Spec
Object Communication Diagrams
Aggregation Hierarchy
Generalization /Specialization Hierarchies
State Transition Diagrams
State 1
Event
Action
State 2
Domain Spec Picture Extractor & Consistency Checker
Domain Spec Picture Relations
Nodes
Arcs
Is a
Domain Spec Generator & Multiple View Consistency Checker
Domain Object Repository
Detailed Object Specs
Feature/Object Rules

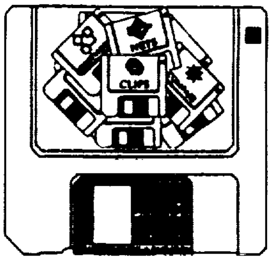
Proof-of-Concept Experiment
Target System Generation:
Generation of Target System Specification


Multiple Views of Target System Spec
Domain Object Repository
Multiple Views of Domain Spec
Target System Object Communication Diagrams
Target System Objects
Objects
Knowledge-Based Requirements Elicitation Tool
Target System Picture Generator
Target System Aggregation Hierarchy
Target System Requirements
Target System State Transition Diagrams
State 1
Event
Action
State 2

REUSE PROJECTS WITHIN JSC's SOFTWARE TECHNOLOGY BRANCH

presented at the
SOFTWARE REUSE TOOLS WORKSHOP
Research Triangle Institute
May 5-6, 1992

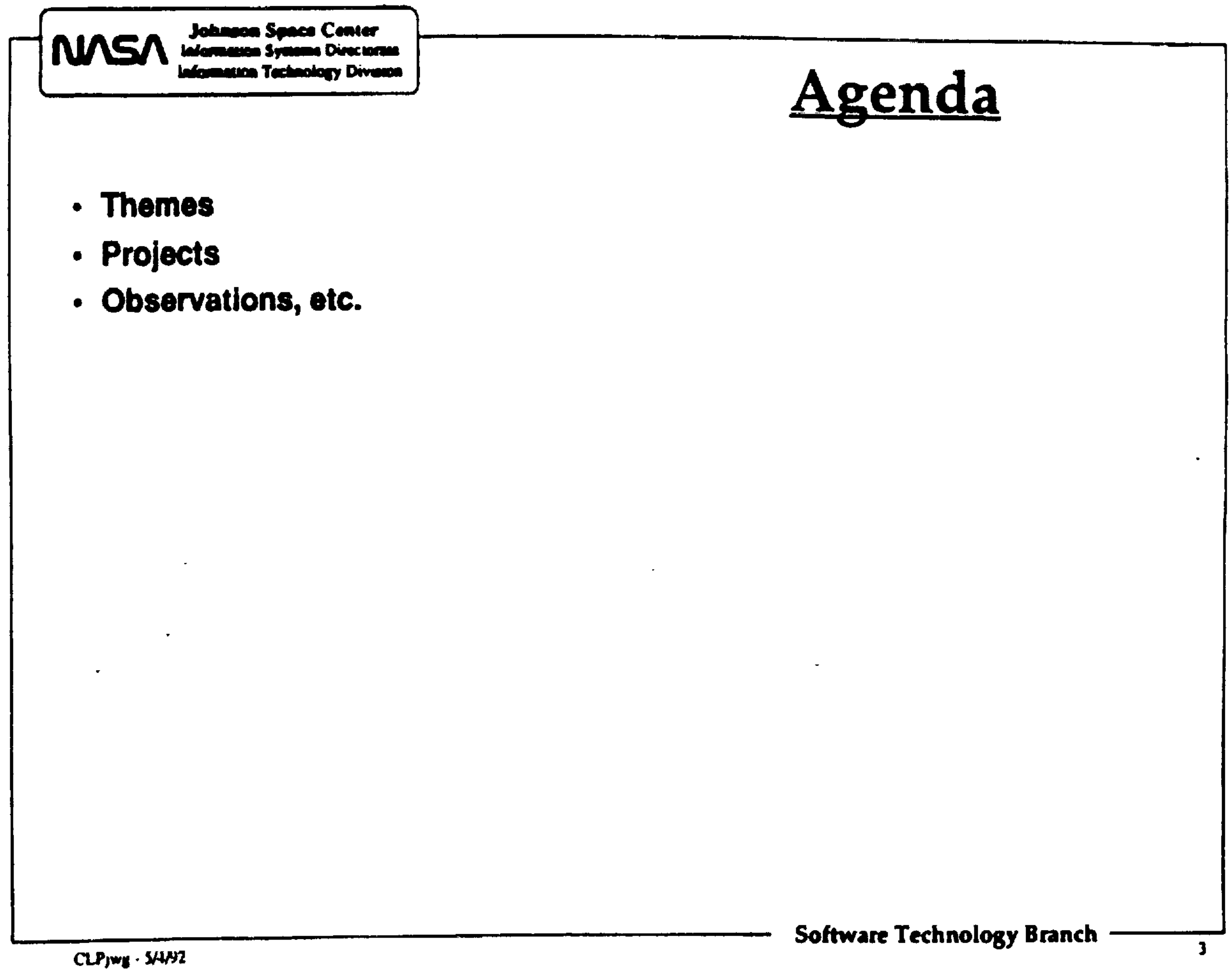
J. W. GOLEJ *, D. M. DIKEL **,
C. L. PITMAN, and E. M. FRIDGE III
NASA/Johnson Space Center/PT4



* The MITRE Corporation ** Applied Expertise, Inc.

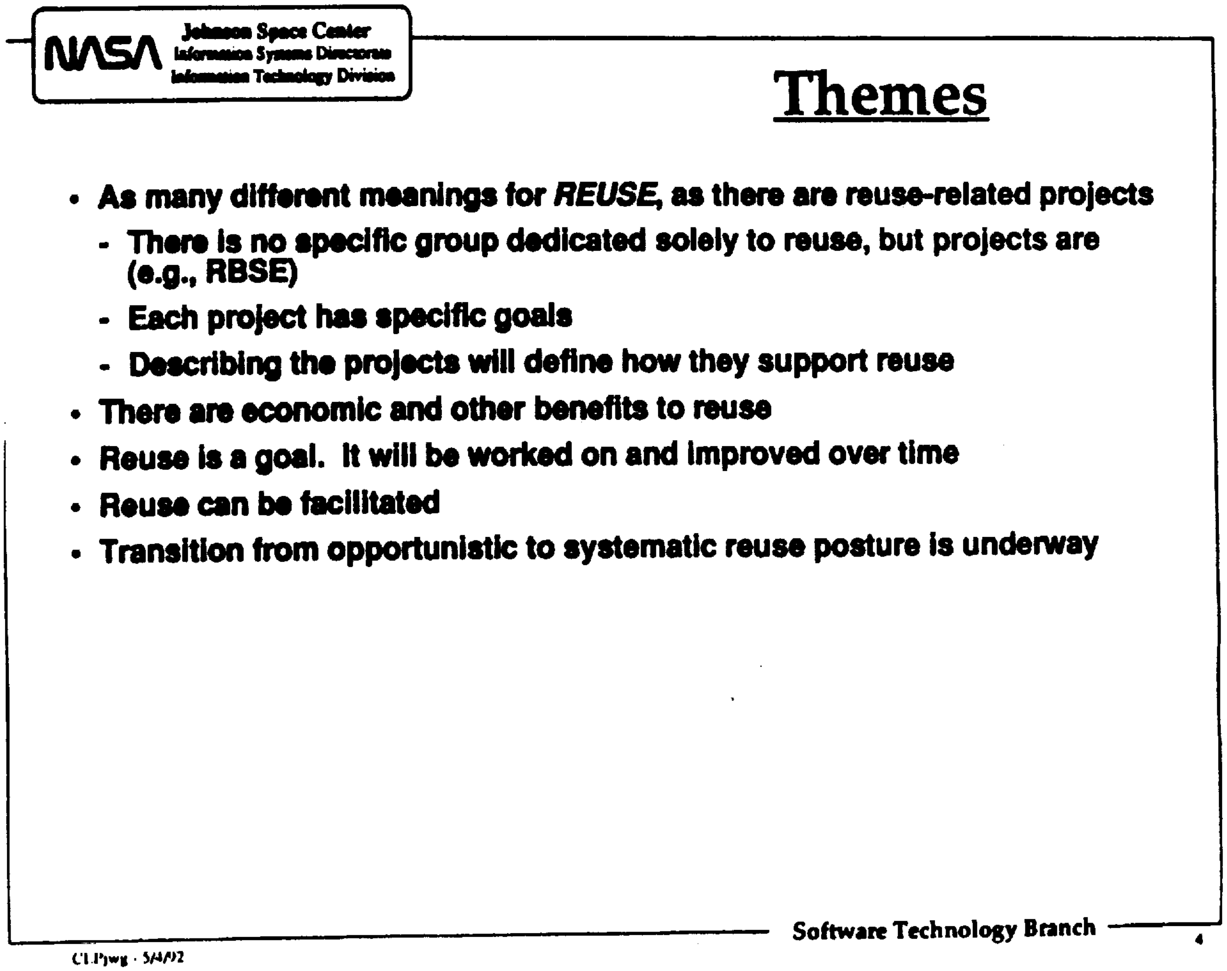
Introduction

- Presenters
  - Support Contractors, not Civil Servants
    - MITRE supports STB in Software Technology Infusion
    - Applied Expertise acts as HQ's liaison for Repository Based Software Engineering (RBSE) System
  - Representing JSC's Software Technology Branch
    - Its projects and activities
    - Its viewpoints
- Points-of-contact, NASA JSC/PT4 Houston, TX 77058
  - Ernie Fridge, (713) 483-8109
  - Dr. Charles Pitman, (713) 483-2469
- Intent is to provide a broad-brush overview of our reuse activities vs. detailing the projects' technical or other merits to
  - Stimulate discussions
  - Foster information exchange



Agenda

- Themes
- Projects
- Observations, etc.



Themes

- As many different meanings for *REUSE*, as there are reuse-related projects
  - There is no specific group dedicated solely to reuse, but projects are (e.g., RBSE)
  - Each project has specific goals
  - Describing the projects will define how they support reuse
- There are economic and other benefits to reuse
- Reuse is a goal. It will be worked on and improved over time
- Reuse can be facilitated
- Transition from opportunistic to systematic reuse posture is underway

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

REAP

project name	developer	contractor	funding source
Re-Engineering Application (REAP)	JSC/PT4	STB, Barrios, MITRE	ISD-Internal

purpose/goals

- Ease maintenance burden and provide re-engineering for JSC legacy systems
  - Huge investment in difficult-to-maintain, yet functionally responsive programs
  - Redevelopment and translation, not economically feasible or otherwise practical
  - Marketplace has historically focused on MIS
- Provide design recovery and re-engineering capabilities for re-implementing legacy systems into more maintainable, modern, and better systems
- Provide Methods, Standards, Tools, and Data integration to facilitate the re-implementation
- Focus on and leverage off COTS/GOTS tools, deviating only to provide otherwise unavailable capabilities
- Implement an extensible, open system based on standards and environments/frameworks

status

- Deliverables
  - REAP version 1.2—integrates JSC/STB and COTS tools under common presentation control for maintenance and analysis of FORTRAN to COMGEN-Standard FORTRAN
  - REAP version 2.0 [under development]—federates 1.2 with COTS tools FORTRAN and C. Includes data interchange (limited integration) between communicating tools and standards-based repository prototype
- Schedule
  - Design recovery for FORTRAN delivered JAN'92; for C by end of CY92
  - Integration standards and framework investigation NOV92
  - Integrated REAP in JUN93
  - Support for real-time information capture, other languages

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

REAP

project name	
Re-Engineering Application (REAP)	Reuse Aspects

entities

- Program design, i.e., call graphs, argument list, ...
- Requirements-potentially
- Standards
- Methodologies
- Tools and Tool Environments

operation

- Capture all code and as much as possible of other information (software life cycle products) in electronic form in a repository
- Use analysis capabilities with human expert in the loops to abstract, group, and structure the information to obtain

constraints

- Variety of usually non-standard inputs due to platform, language and other environmental differences
- Terminological differences and the lack of standard definitions for and understanding of the process and products of software development

findings

- Much valuable maintenance and design recovery support is available now
- Recovery of highest level design information from code is difficult and requires analysis
  - Domain knowledge often no longer available
  - Attaining most abstract information needs to be traded off against utility and ROI of effort
- Customer/Market driven fervor

SimTool

project name	developer	contractor	funding source
SimTool	JSC/PT4	LinCom	SBIR

purpose/goals

- Aid simulation designers in the construction, use, and maintenance of simulation applications, data, and math models from reusable parts and domain architectures
- Provide a facility that transforms a user-constructed graphic specification of a simulation into appropriate Ada code

status

- Deliverables
  - Fairly comprehensive Unix-based prototype with all functionality, on schedule
  - Application tools and some library tools currently under development
- Schedule
  - Phase I (Proof-Of-Concept) completed Summer'89
  - Phase II (Developmental) ends Fall'92

SimTool

project name	
SimTool	Reuse Aspects

entities

- Software components: executive, interface, math models
- Application Specification
- Library

operation

- To construct simulation, user builds an Application Specification by selecting an appropriate executive, application interface, and math models from among the reusable software components in library
- Application specs identify components to be integrated and relationships to each other and simulation
- Executive software components control the state of a simulation application and coordinate its execution
- Application I/I components bridge math models and executive, and establish domain architecture

constraints

- All Software Components must conform to interface and control standards
  - To assure integration
  - To encourage development of modularization, reuseability, and object orientedness
- All Software Components include attributes for component classification, data interface (units, description, initialization requirements, etc.), and control dependencies

findings

- Feasible
- Classification scheme was a major, fundamental development
  - Component Design
  - Functional
  - Domain

Mission

project name	developer	contractor	funding source
Mission	JSC/PT4	UHCL	HQ/Code R

purpose/goals

- Predictably and reliably compose models of mission and safety critical (MASC) applications and systems
- Transform models into real world systems preserving the MASC properties
- Represent the processes, products, and interfaces used to evolve and sustain the models as interrelated, traceable, and reusable artifacts spanning the computer and software engineering life cycle
- Support both precise and formal levels of modeling discipline

status

- Deliverables
  - Clear Lake Life Cycle Model (CL-LCM)
  - Object Management System (OMS)
  - Library Management System (LMS)
- Schedule
  - OMS/LMS prototype delivered FY91—continuous improvement planned through FY96
  - Life Cycle Methodology, delivery in FY93—continuous improvement planned through FY96

Mission

project name	
Mission	Reuse Aspects

entities

- Computer and software processes/models
- Computer and software products/models
- Computer and software interfaces/models
- Libraries of the above, including specific views and configurations

operation

- Evolution will always begin by updating the model and then proceed to update the existing, system so that its structure and behavior continue to mirror the MASC properties of the rigorous model
- Significantly extend traditional entity-attribute / relationship-attribute representations of semantic models
- Provide hierarchy of structures and disciplines for evolving and sustaining traceable sets of semantic models: abstract I/I specs, virtual I/I sets, stable frameworks of subsystems, and stable I/I sets

constraints

- Behavior and structure to support a rigorous discipline of composing models to tractably subject MASC subsets of the precise models to formal methods

findings

- Formal models and methods have not been tractable for large, complex distributed systems with MASC functions and components.
- Appropriate object-based disciplines offer the best hope for controlling the life cycle complexities of MASC applications and systems. Object-based disciplines are insufficient: *Precise* fine-grained models should provide extensible semantics for structure and behavior internal to objects and systems of objects
- A fine-grained OMS/LMS is needed to support such modeling

RBSE

project name	developer	contractor	funding source
Repository-Based Software Engineering (RBSE)	JSC/PT4	UHCL, et al.	HQ/Code C

purpose/goals

- Distribute technology across many industries and domains. Deliver and support robust product set
  - Commit to customer-driven quality. Consistently monitor and increase customer benefits
  - Serve high-leverage, high-impact civilian markets (i.e., aerospace, education, and mission- and safety critical) affecting U.S. competitiveness and from which technology spreads quickly
  - Introduce reuse into customers' mainstream software development practices so as to have them parallel the clarity, consistency and predictability of other engineering disciplines
  - Make reuse tools, assets, and practices more easily/economically accessible. Apply human factors engineering, explore classification schemes, develop generalized life-cycle process model, and help lead cooperative efforts
- Replace outdated architecture limiting system responsiveness and capability with open-systems-based library management system

status

- Deliverables
  - Operational prototype
  - Highly capable service organization, skilled in cataloging, managing, and delivering software assets [AdaNET Help Desk: (800) 400-1458]
  - Safer, higher quality products for NASA
- Schedule
  - Public-domain software reuse library (AdaNET) in operation since 1989
  - Enhanced functionality/performance capability in SEP92 (i.e., NASA Electronic Library System (NELS))

RBSE

project name	
Repository-Based Software Engineering (RBSE)	Reuse Aspects

entities

- Standard life cycle practices
- Standard life cycle components
- Reuse libraries, customized in technology and service

operation

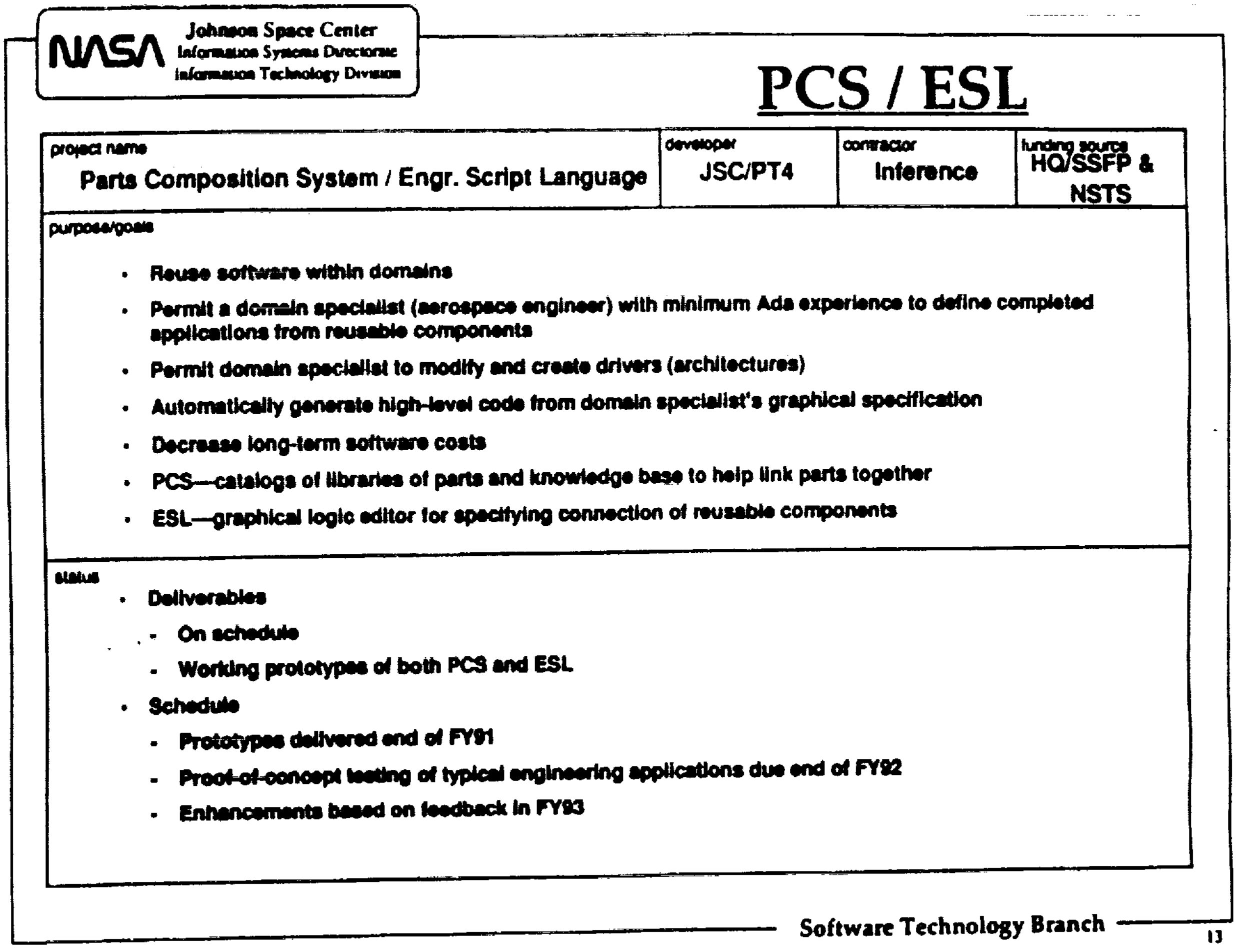
- Expand base of library suppliers and customers through interoperability
- Share advances of other repositories
- Fill critical technology gaps through research
- Adapt to changing customer requirements by integrating research results and COTS/GOTS products

constraints

- Cooperation, essential
- Breadth of R&D programs, critical while technology and practice of reuse matures
  - To encourage development of modularization, reusability, and object orientedness

findings

- Software practice lacks essential elements common to mature engineering fields. Effective reuse of common elements is necessary to approach efficiency and predictability of other disciplines
- NASA can make a major contribution to the solution—both as a source and a reuser
- There are many barriers to reuse; *no one program can solve this problem* (i.e., cooperation)



NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

PCS / ESL

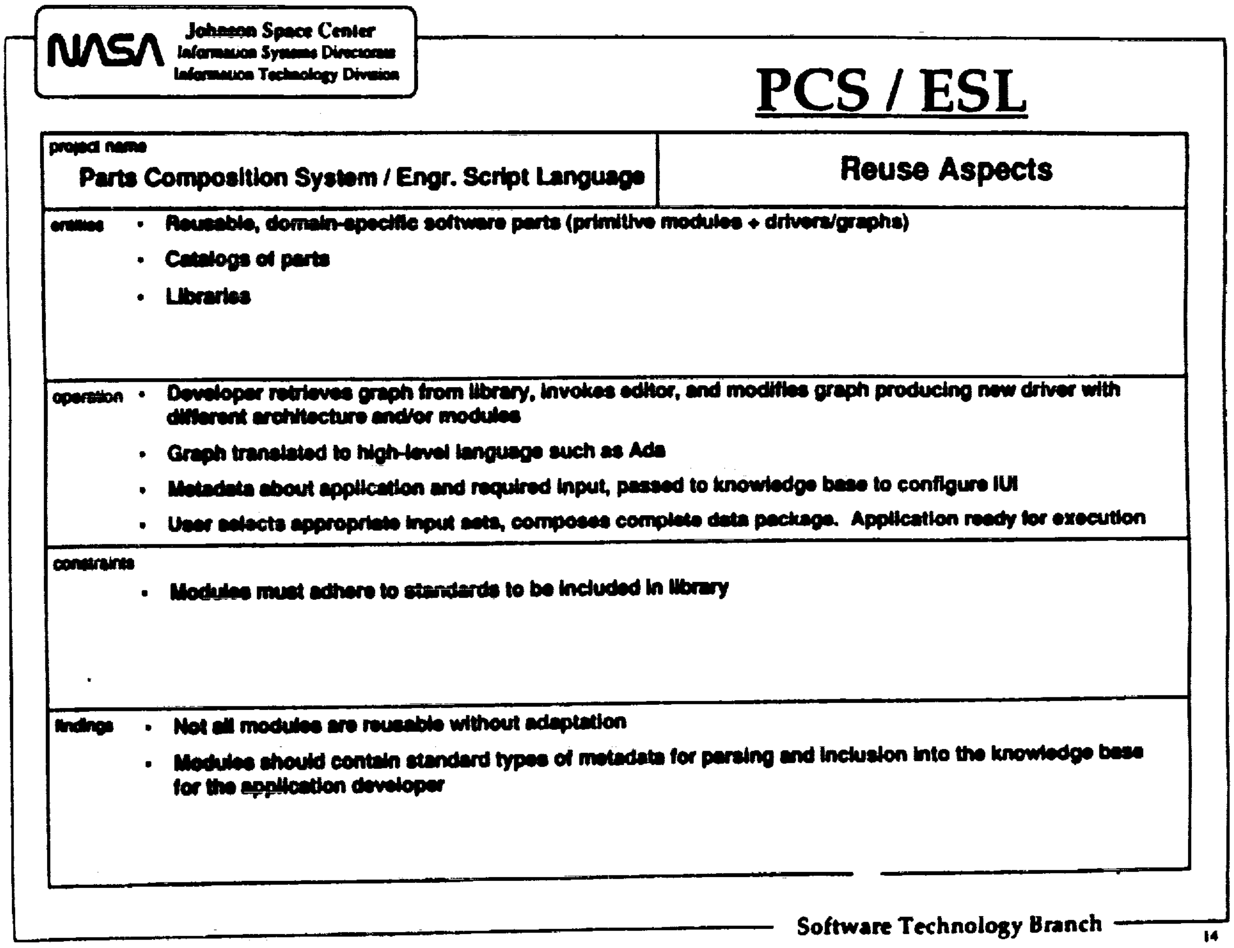
project name	developer	contractor	funding source
Parts Composition System / Engr. Script Language	JSC/PT4	Inference	HQ/SSFP & NSTS

purpose/goals

- Reuse software within domains
- Permit a domain specialist (aerospace engineer) with minimum Ada experience to define completed applications from reusable components
- Permit domain specialist to modify and create drivers (architectures)
- Automatically generate high-level code from domain specialist's graphical specification
- Decrease long-term software costs
- PCS—catalogs of libraries of parts and knowledge base to help link parts together
- ESL—graphical logic editor for specifying connection of reusable components

status

- Deliverables
  - On schedule
  - Working prototypes of both PCS and ESL
- Schedule
  - Prototypes delivered end of FY91
  - Proof-of-concept testing of typical engineering applications due end of FY92
  - Enhancements based on feedback in FY93



NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

PCS / ESL

project name	
Parts Composition System / Engr. Script Language	Reuse Aspects

entities

- Reusable, domain-specific software parts (primitive modules + drivers/graphs)
- Catalogs of parts
- Libraries

operation

- Developer retrieves graph from library, invokes editor, and modifies graph producing new driver with different architecture and/or modules
- Graph translated to high-level language such as Ada
- Metadata about application and required input, passed to knowledge base to configure IUI
- User selects appropriate input sets, composes complete data package. Application ready for execution

constraints

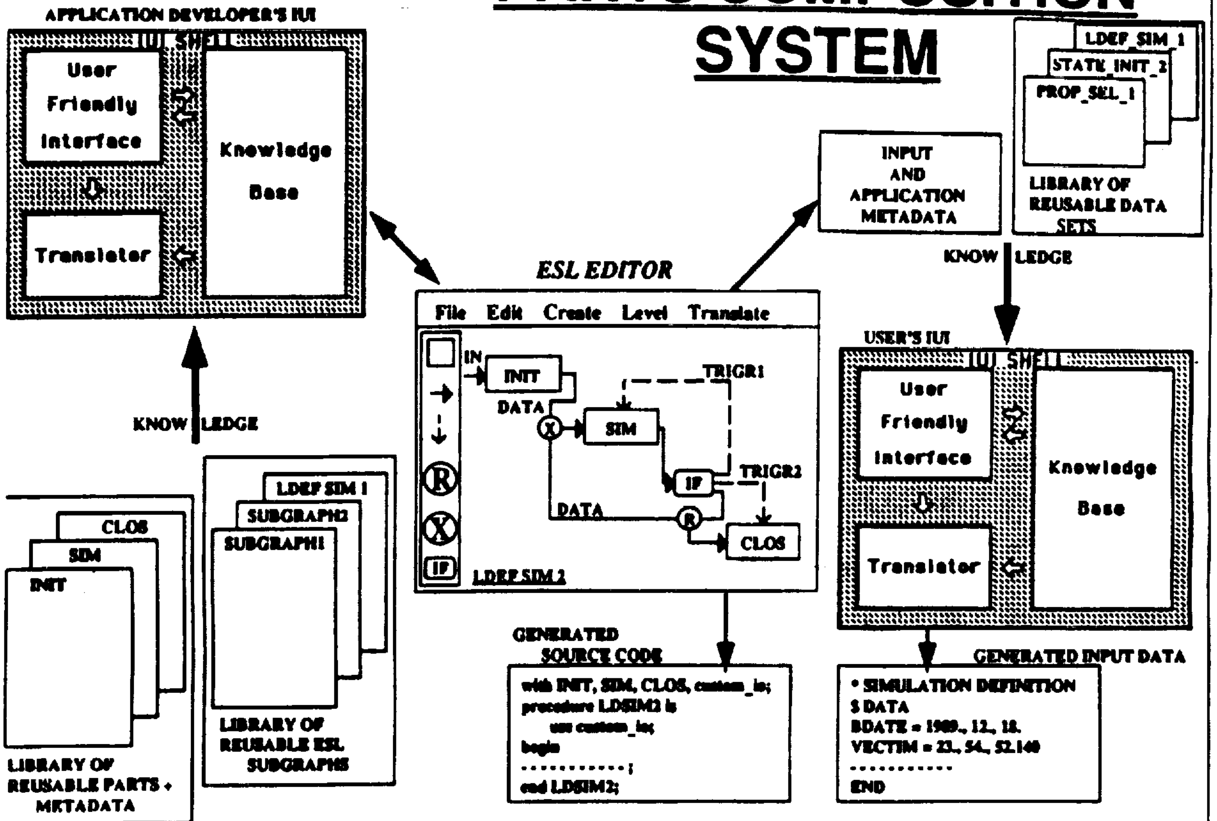
- Modules must adhere to standards to be included in library

findings

- Not all modules are reusable without adaptation
- Modules should contain standard types of metadata for parsing and inclusion into the knowledge base for the application developer

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

PARTS COMPOSITION SYSTEM



```
with INIT, SIM, CLOS, custom_io;
procedure LDSIM2 is
     use custom_io;
begin
............;
end LDSIM2;
```

```
* SIMULATION DEFINITION
$DATA
BDATE = 1989., 12., 18.
VECTIM = 23., 54., 52.140
...........
END
```

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

AN INPUT DATA PACKAGE

```
$RUN
   *   STS32-MONTE CARLO ANALYSIS/OFP2R CYCLE  - -
   *     ****************************************
   *         SYMBOL TABLE MODS
   *     ****************************************
$SYMBOL
*ADD,P       S0.SYMBOL
*ADD,P        DMS*TCLDEV.GNDSIM/SYMBOL
   *     ****************************************
   *          BEGIN CASE AND PHASE DEFINITION
   *     ****************************************
$CASE(0) /RENDEZVOUS PROFILE/
   *     ****************************************
$PHASE(10) /MC1 - EXECUTE OMS-2/
   *     ----------------------------------------
   *          SIMULATION DEFINITION
   *     ----------------------------------------
$DATA
      DICTOR     =  1
      NORBT      =  1
      IMANVV     =  1
      NV         =  2                           *TWO VEH. SIMULATED
      KDYN       =  5                           *ORBITAL SIMULATION
      BDATE      =   1989., 12., 18.            *LIFTOFF DATE
      VECTIM     =   23., 54., 52.140           *START OF SIM (MECO)
   *     ----------------------------------------
   *          MONTE CARLO INITIALIZATION
   *     ----------------------------------------
$DATA
      MTCP       =  2                           *TWO VEHICLE MONTE CARLO
      NDISP      =  1                           *NAV OFF FOR REFERENCE
*ADD,P       DMS*TCLDEV.MC-INITIAL/STS8-0      *INIT COV - REF INTO ACT
```

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

INTUIT

project name	developer	contractor	funding source
INTelligent User Interface (IUI) Development Tool	JSC/PT4	Inference, Barrios	HQ/SSFP & NSTS

purpose/goals

- Reduce the complexity of space flight simulation and its heavy input data construction and management load
- Emphasis is on aiding user in composing complete input data packages from available data groups, with minimal modifications
- Provide a user friendly graphical interface that presents information more effectively and makes interactions clearer and more efficient
- Supply an intelligent assistant (expert system) to check user input for consistency with a goal of making a program execute correctly on the first run
- Supply a case-based reasoning system to aid in searching for data sets
- Provide a generic tool configurable to any application

status

- Deliverables
  - Knowledge base extended to cover additional application domains
  - Proof-of-concept demonstration of programs running correctly on their first run
- Schedule
  - Proof-of-concept successfully demonstrated in FY91
  - Apply INTUIT to various complicated engineering application in CY92
  - Refine methodology for configuring knowledge bases in CY92

CLPrwe - 5/4/92

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

INTUIT

project name	
INTelligent User Interface (IUI) Development Tool	Reuse Aspects

entities

- Input data sets
- User interfaces

operation

- User invokes a configured INTUIT shell
- Selects an existing input data package close to his/her needs
- Selects (and modifies, if required) a few new data groups to replace some of the old
- A complete data package is translated into exact format required by the computer, so that no changes are required in the existing input structure

constraints

- INTUIT is most useful when:
  - Input structure is complicated, requiring both graphical user interface and expert system assistance
  - Input structure must be readily reconfigurable

findings

- INTUIT shell permits quick and easy building of IUIs and, therefore, efficient upgrade of complex simulation programs. Improvements in runstream setup of 50–80% (labor)
- IUIs, by making data use and reuse easier and more efficient
  - Reduce computer programming knowledge required and allow concentration on engineering domain
  - Reduce time required for user training

CLPjwg 5/4/92

Observations, etc.

- Barriers to reuse, both real and procedurally ingrained, need to be eliminated
  - Standards
  - Paradigms
  - Culture
- Incentives need to be developed and put in place
  - For reusing products developed elsewhere
  - For developing reusable products
- Infrastructure must be developed, distributed, and made easily usable and available to foster high levels of reuse of products of the software life cycle
  - Software Engineering Environments
  - Repositories/Libraries—for accessibility
- Reuse does not come free of charge, i.e., it costs to design and develop
  - Reusable items
  - Methods to make reusable components available and, then, to find, access, and utilize them

Observations, etc. (Con'c.)

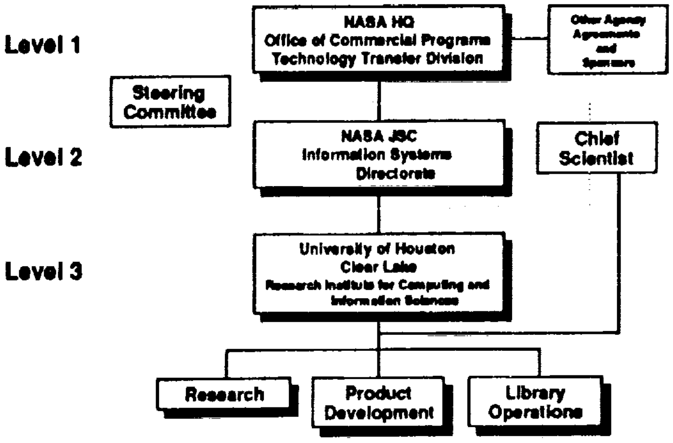
- How to design for reuse is not a given, but a developing concept
  - Optimum granularity of reuse and reusable components, if it exists
  - Domain independence or dependence, or both
- Probably the biggest payback lies with reuse at the process, design, and model levels, i.e., levels more abstract than code
- Reuse is certainly *not the proverbial silver bullet*

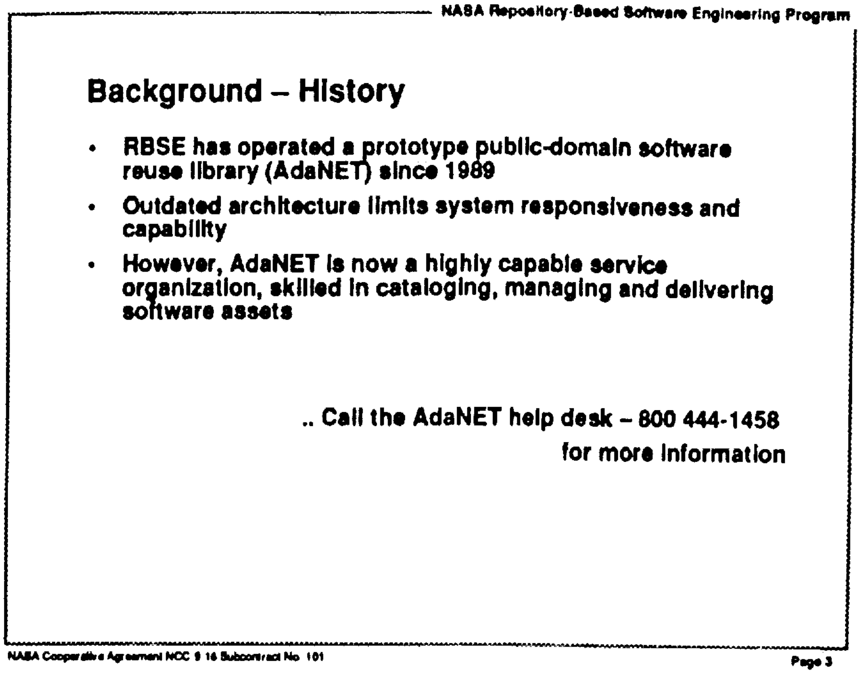
NASA's Repository-Based Software Engineering (RBSE) Program and Collaborative Efforts

NASA Software Reuse Tools Workshop
May 4-5, 1992

Dave Dikel
Applied Expertise, Inc.
1925 North Lynn Street, Suite 802
Arlington, VA 22209
703 516-0911
FAX: 516-0918
ddikel@ajpo.sei.cmu.edu
NASAMAIL: DDIKEL

Background – Management Structure


Level 1
Level 2
Level 3
NASA HQ
Office of Commercial Programs
Technology Transfer Division
Other Agency Agreements and Sponsors
Steering Committee
NASA JSC
Information Systems Directorate
Chief Scientist
University of Houston Clear Lake
Research Institute for Computing and Information Sciences
Research
Product Development
Library Operations



NASA Repository-Based Software Engineering Program

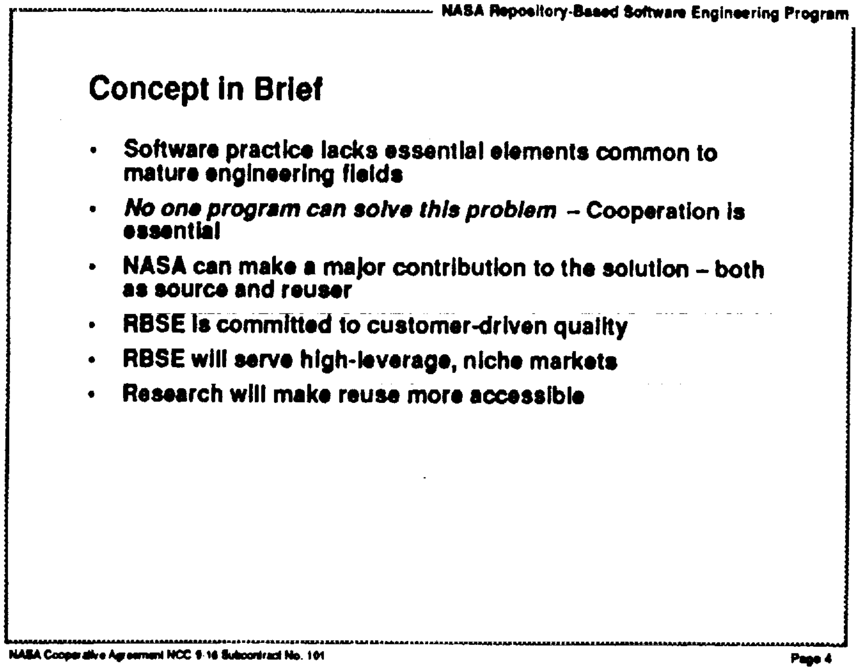
Background – History

- RBSE has operated a prototype public-domain software reuse library (AdaNET) since 1989
- Outdated architecture limits system responsiveness and capability
- However, AdaNET is now a highly capable service organization, skilled in cataloging, managing and delivering software assets

.. Call the AdaNET help desk – 800 444-1458
for more information

NASA Cooperative Agreement NCC 9 16 Subcontract No. 101

Page 3



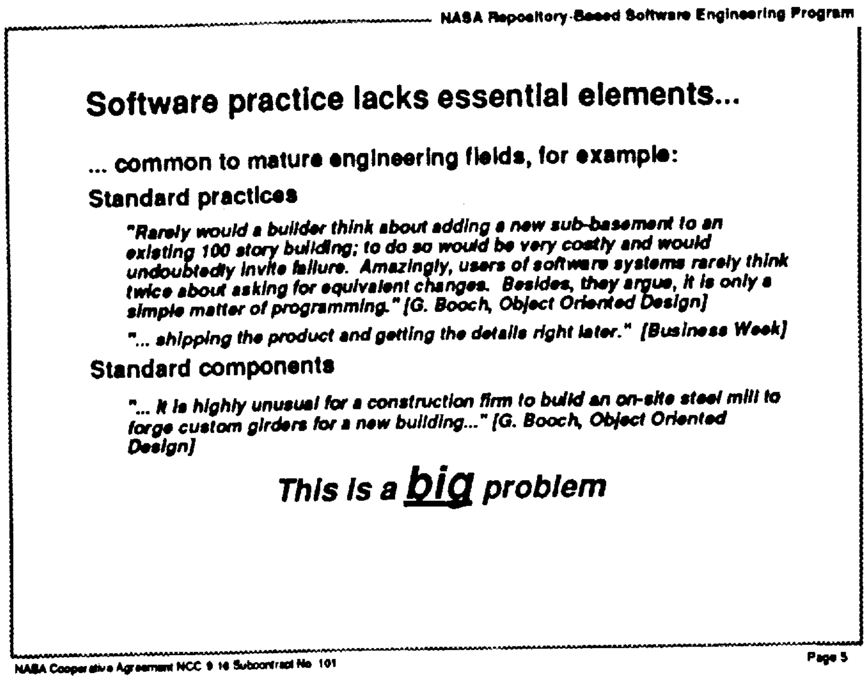
NASA Repository-Based Software Engineering Program

Concept in Brief

- Software practice lacks essential elements common to mature engineering fields
- *No one program can solve this problem* – Cooperation is essential
- NASA can make a major contribution to the solution – both as source and reuser
- RBSE is committed to customer-driven quality
- RBSE will serve high-leverage, niche markets
- Research will make reuse more accessible

NASA Cooperative Agreement NCC 9 16 Subcontract No. 101

Page 4



Software practice lacks essential elements...

... common to mature engineering fields, for example:

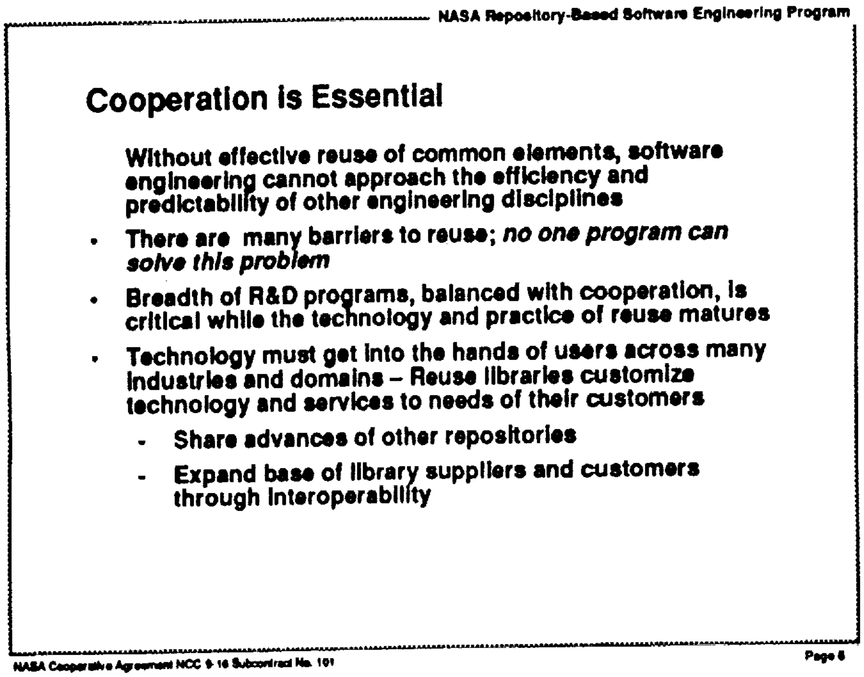
Standard practices

> *"Rarely would a builder think about adding a new sub-basement to an existing 100 story building; to do so would be very costly and would undoubtedly invite failure. Amazingly, users of software systems rarely think twice about asking for equivalent changes. Besides, they argue, it is only a simple matter of programming." [G. Booch, Object Oriented Design]*
>
> *"... shipping the product and getting the details right later." [Business Week]*

Standard components

> *"... it is highly unusual for a construction firm to build an on-site steel mill to forge custom girders for a new building..." [G. Booch, Object Oriented Design]*

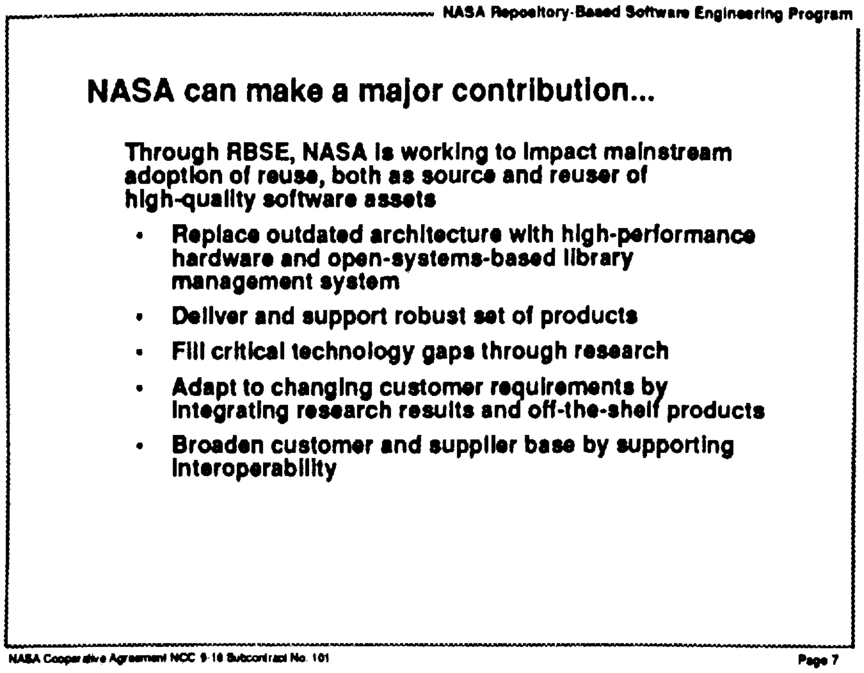
*This is a **big** problem*



Cooperation is Essential

Without effective reuse of common elements, software engineering cannot approach the efficiency and predictability of other engineering disciplines

- There are many barriers to reuse; *no one program can solve this problem*
- Breadth of R&D programs, balanced with cooperation, is critical while the technology and practice of reuse matures
- Technology must get into the hands of users across many industries and domains – Reuse libraries customize technology and services to needs of their customers
  - Share advances of other repositories
  - Expand base of library suppliers and customers through interoperability



NASA Repository-Based Software Engineering Program

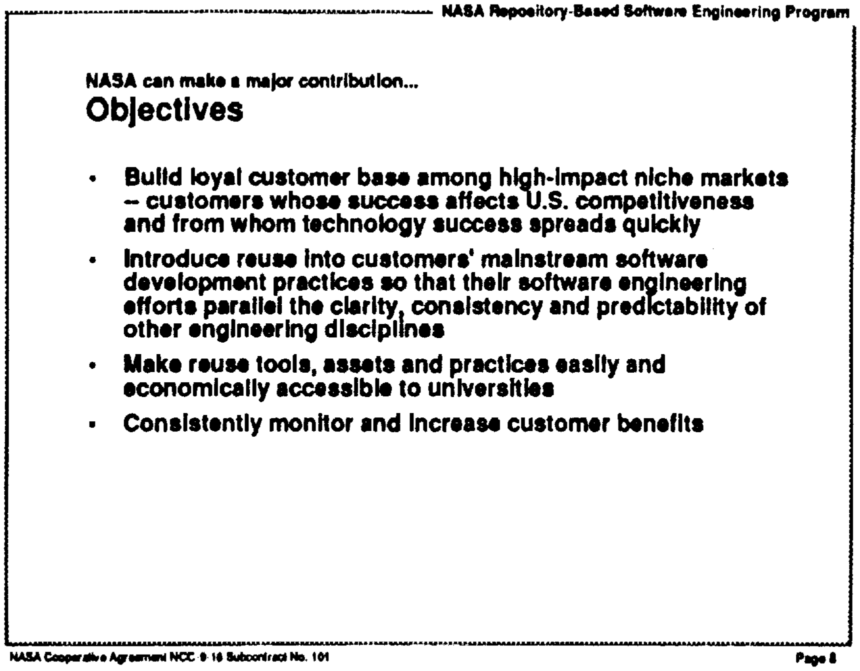
NASA can make a major contribution...

Through RBSE, NASA is working to impact mainstream adoption of reuse, both as source and reuser of high-quality software assets

- Replace outdated architecture with high-performance hardware and open-systems-based library management system
- Deliver and support robust set of products
- Fill critical technology gaps through research
- Adapt to changing customer requirements by integrating research results and off-the-shelf products
- Broaden customer and supplier base by supporting interoperability

NASA Cooperative Agreement NCC 9-16 Subcontract No. 101

Page 7



NASA Repository-Based Software Engineering Program

NASA can make a major contribution...

Objectives

- Build loyal customer base among high-impact niche markets – customers whose success affects U.S. competitiveness and from whom technology success spreads quickly
- Introduce reuse into customers' mainstream software development practices so that their software engineering efforts parallel the clarity, consistency and predictability of other engineering disciplines
- Make reuse tools, assets and practices easily and economically accessible to universities
- Consistently monitor and increase customer benefits

NASA Cooperative Agreement NCC 9-16 Subcontract No. 101

Page 8

NASA can make a major contribution...

Benefits

- Increased customer competitiveness
- Widespread dissemination of NASA-developed software assets and technology
- Graduates who are better able to engineer large, complex software systems
- Safer, higher quality products for NASA

Commitment to Customer-Driven Quality

Ensures that RBSE --

- Provides customers with what they expect and need
- Focuses on efficiency, i.e., providing products and services at a minimum cost while ever more effectively increasing bottom-line benefits to target customers
- Measures its impact using well-defined criteria

RBSE will serve high-leverage, niche markets

- NASA/civilian aerospace application domains
- Civilian mission- and software-intensive, safety-critical systems
- Educational institutions interested in reuse

Research will make reuse more accessible by...

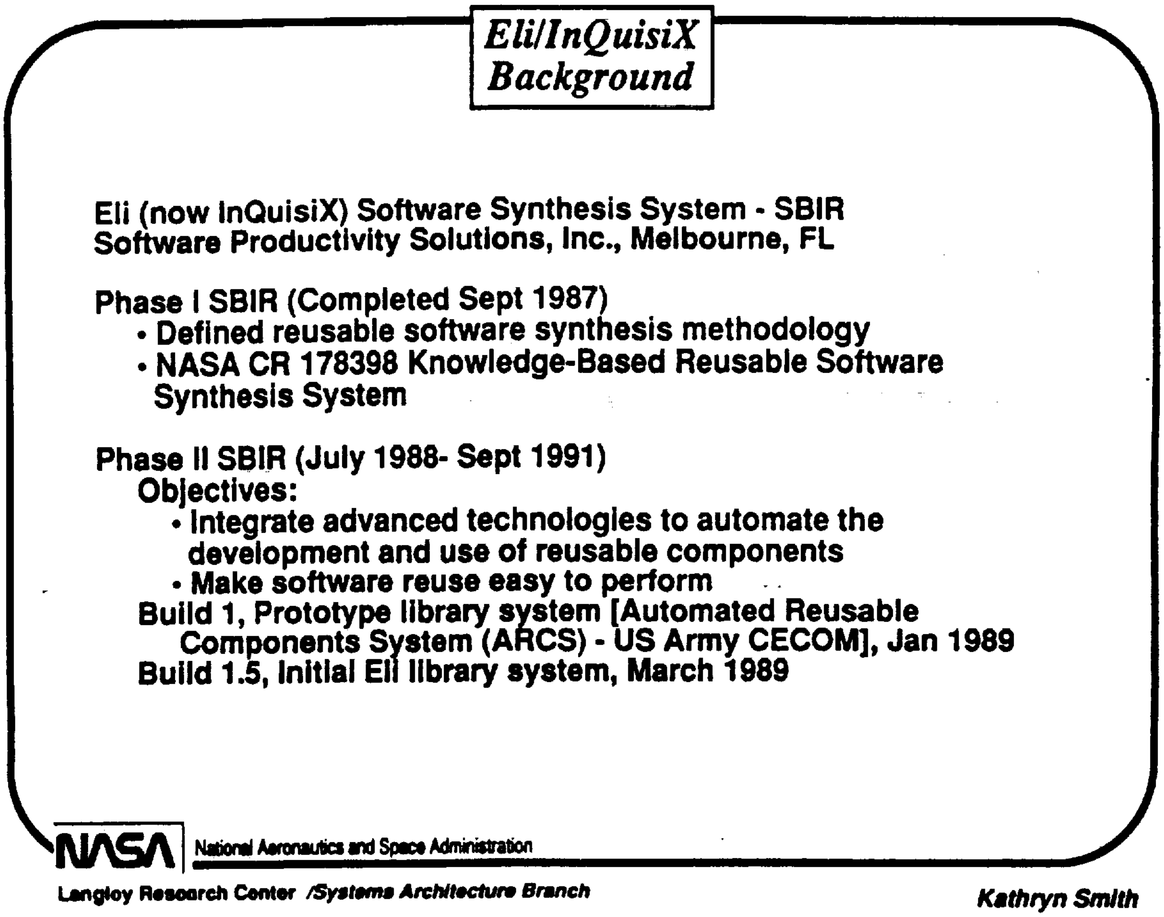
- Applying human-factors engineering
- Exploring new classification schemes
- Developing a generalized life-cycle process model
- Helping to lead key cooperative reuse efforts

Software Reuse

Software Reuse in Systems Architecture Branch Information Systems Division NASA Langley

OUTLINE

- Background
- Eli/InQuisiX overview
- Plans



Eli/InQuisiX Background

Eli (now InQuisiX) Software Synthesis System - SBIR
Software Productivity Solutions, Inc., Melbourne, FL

Phase I SBIR (Completed Sept 1987)

- Defined reusable software synthesis methodology
- NASA CR 178398 Knowledge-Based Reusable Software Synthesis System

Phase II SBIR (July 1988- Sept 1991)

Objectives:

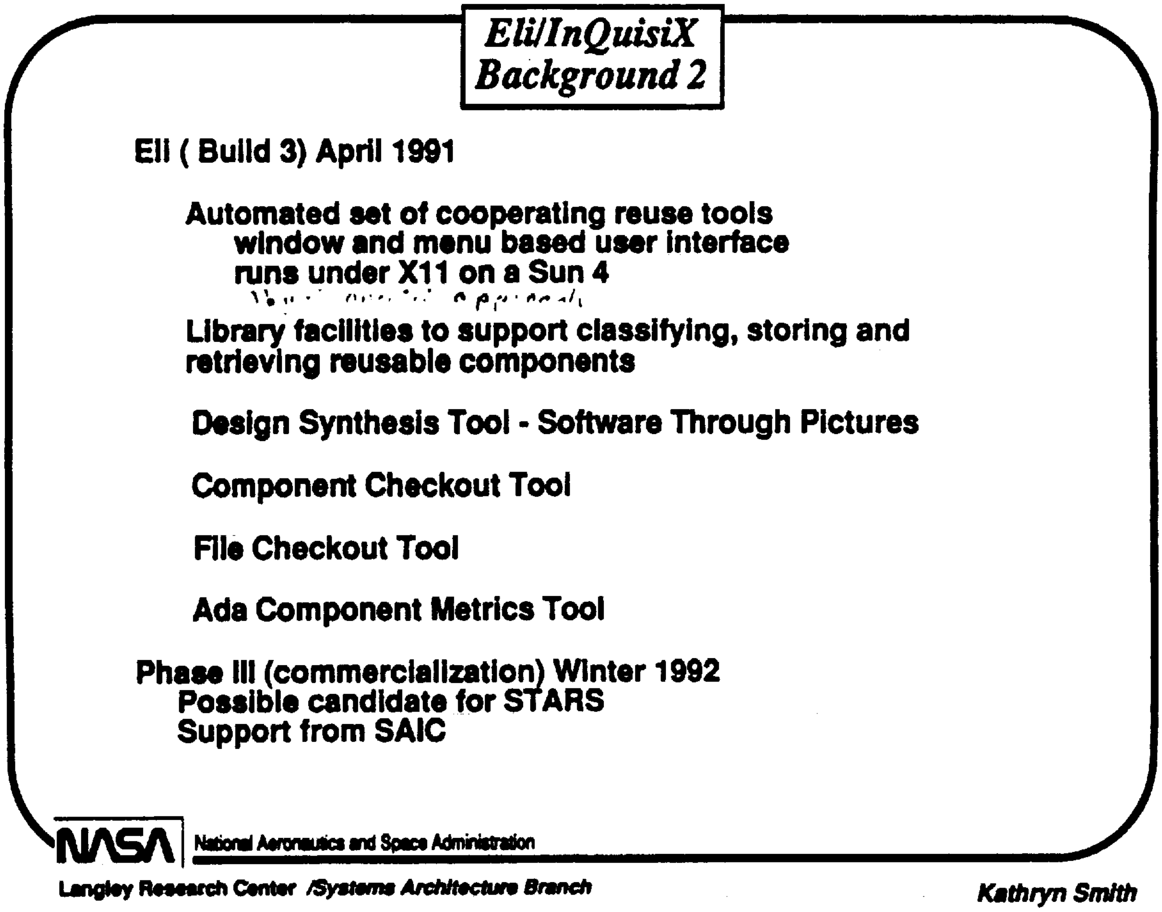
- Integrate advanced technologies to automate the development and use of reusable components
- Make software reuse easy to perform

Build 1, Prototype library system [Automated Reusable Components System (ARCS) - US Army CECOM], Jan 1989

Build 1.5, Initial Eli library system, March 1989

NASA National Aeronautics and Space Administration
Langley Research Center /Systems Architecture Branch

Kathryn Smith



Eli/InQuisiX Background 2

Eli (Build 3) April 1991

Automated set of cooperating reuse tools
window and menu based user interface
runs under X11 on a Sun 4
[illegible]

Library facilities to support classifying, storing and retrieving reusable components

Design Synthesis Tool - Software Through Pictures

Component Checkout Tool

File Checkout Tool

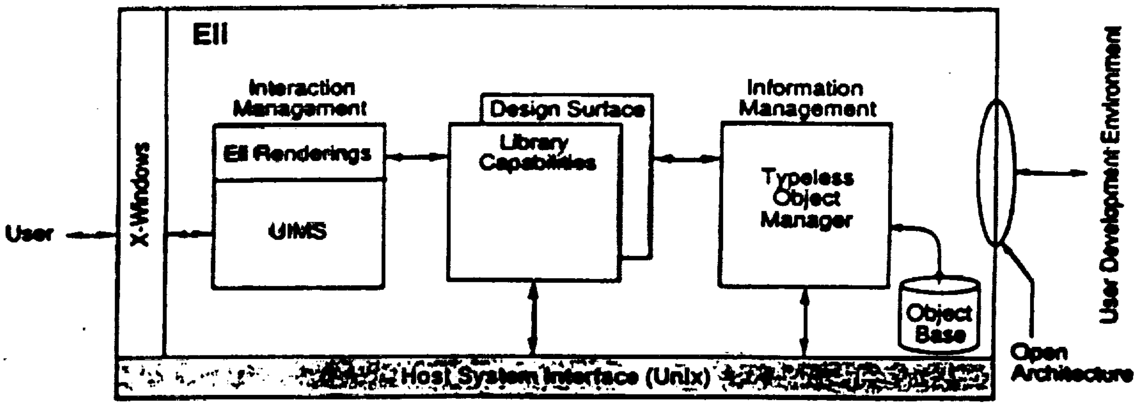
Ada Component Metrics Tool

Phase III (commercialization) Winter 1992
Possible candidate for STARS
Support from SAIC

NASA National Aeronautics and Space Administration
Langley Research Center /Systems Architecture Branch

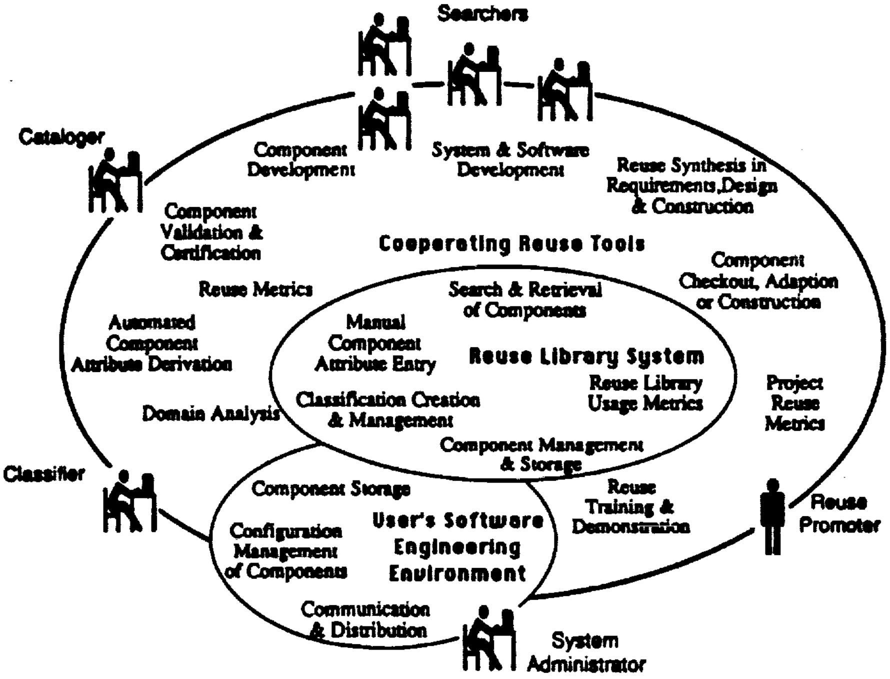
Kathryn Smith

Top-Level Eli Architecture



SPS

InQuisiX
Software Synthesis System


Searchers
Cataloger
Component Development
System & Software Development
Reuse Synthesis in Requirements,Design & Construction
Component Validation & Certification
Cooperating Reuse Tools
Component Checkout, Adaption or Construction
Reuse Metrics
Search & Retrieval of Components
Automated Component Attribute Derivation
Manual Component Attribute Entry
Reuse Library System
Reuse Library Usage Metrics
Project Reuse Metrics
Domain Analysis
Classification Creation & Management
Component Management & Storage
Classifier
Component Storage
Reuse Training & Demonstration
Reuse Promoter
Configuration Management of Components
User's Software Engineering Environment
Communication & Distribution
System Administrator

Eli/InQuisiX Software Synthesis System

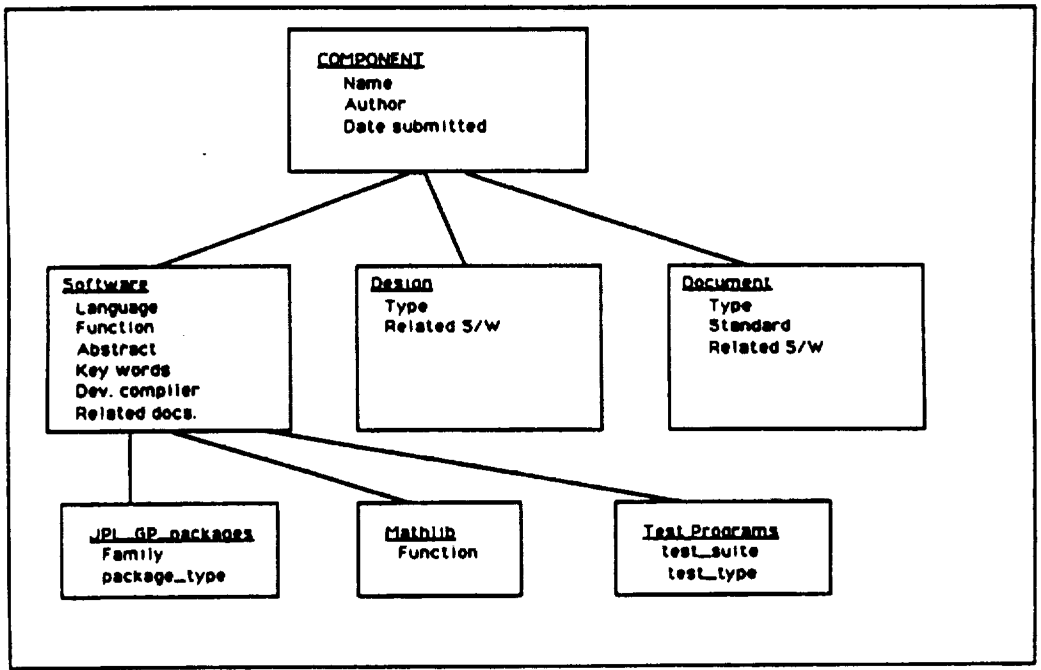
Flexible:

- User defined component classes and classifications
- User tailorable and user extensible

Supports many types of attributes

- Faceted classifications
- Text
- File
- Keywords

InQuisiX Library Classes



COMPONENT
Name
Author
Date submitted

Software
Language
Function
Abstract
Key words
Dev. compiler
Related docs.

Design
Type
Related S/W

Document
Type
Standard
Related S/W

JPL GP packages
Family
package_type

Mathlib
Function

Test Programs
test_suite
test_type

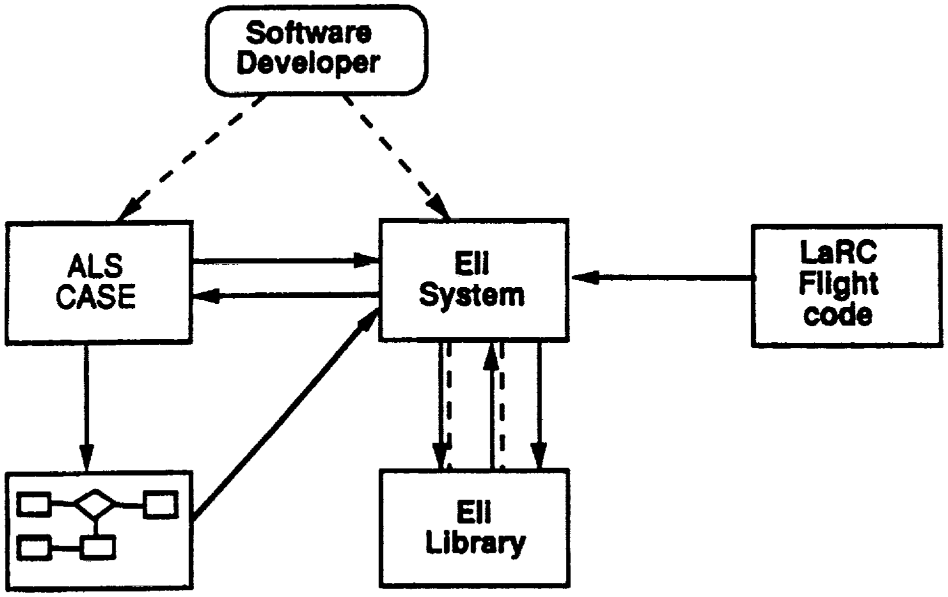
Eli/InQuisiX Plans

Serve as a beta test site for Eli/InquisiX

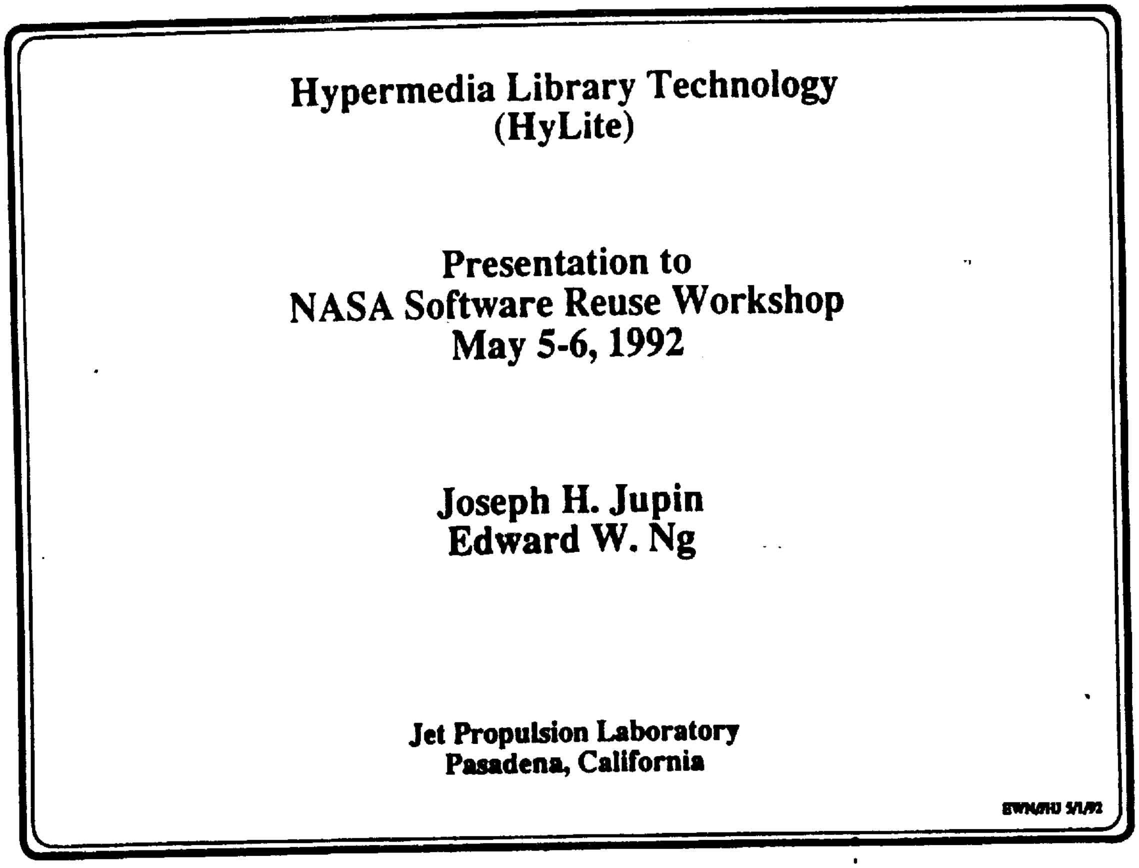
Technology Transfer

Develop interaction between InQuisiX and CSDL CASE

Eli-CSDL CASE Interaction



JPL



Hypermedia Library Technology (HyLite)

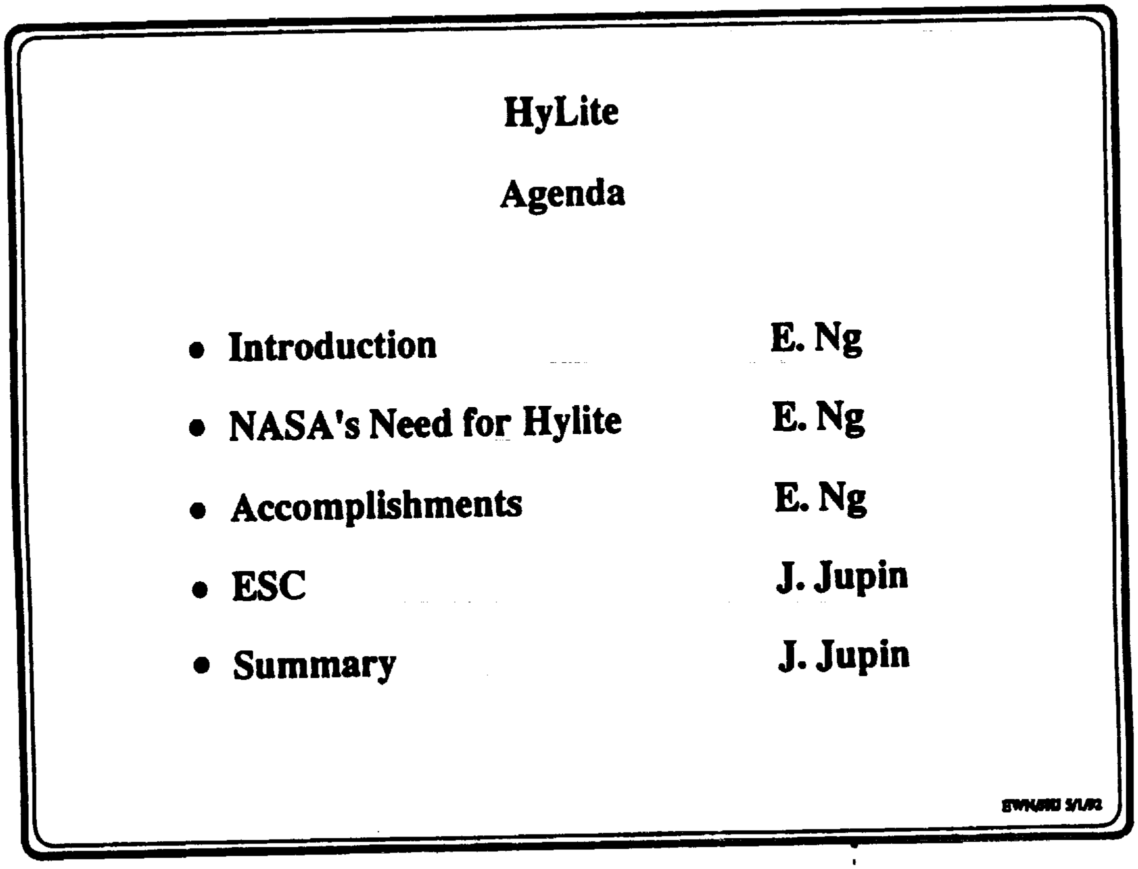
Presentation to
NASA Software Reuse Workshop
May 5-6, 1992

Joseph H. Jupin
Edward W. Ng

Jet Propulsion Laboratory
Pasadena, California

EWN/JHJ 5/1/92

JPL



HyLite

Agenda

• Introduction	E. Ng
• NASA's Need for Hylite	E. Ng
• Accomplishments	E. Ng
• ESC	J. Jupin
• Summary	J. Jupin

EWN/JHJ 5/1/92

Introduction

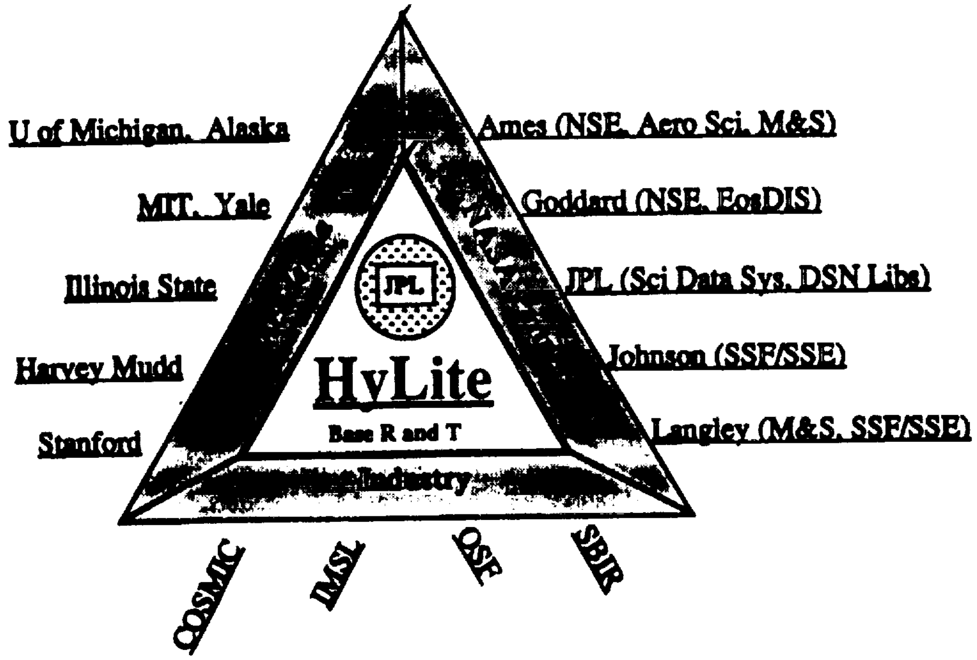
- HyLite is a research & development activity to produce a versatile system as part of NASA technology thrusts in automation, information sciences & communications.

- Useful as a versatile system or tool to facilitate the construction of electronic libraries for:

  - software components
  - hardware parts or design diagrams
  - scientific or engineering datasets or databases
  - bibliography organized by special taxonomy
  - configuration management information
  - etc...

SWN/RU 5/1/92

APPLICATIONS AND SPIN-OFFS (5 YEAR HORIZON)



JPL

~~NASA's Need for HyLite~~
HyLite for NASA Applications

- HyLite provides the potential to address a b~~or~~road range of NASA problems in the 1990's, such as,
  - scientific data deluge
  - rapidly increasing complexity in software development
  - ever growing volumes and variety of documentation
- HyLite evolved from a task formerly entitled the Encylopedia of Software Components (ESC)
- ESC was motivated primarily by the need for software reuse
- It was designed in anticipation of the "K by N by L" problem, that is, K kinds of computers, N applications, & L languages
- This presentation will focus on the software/reuse relevance of HyLite

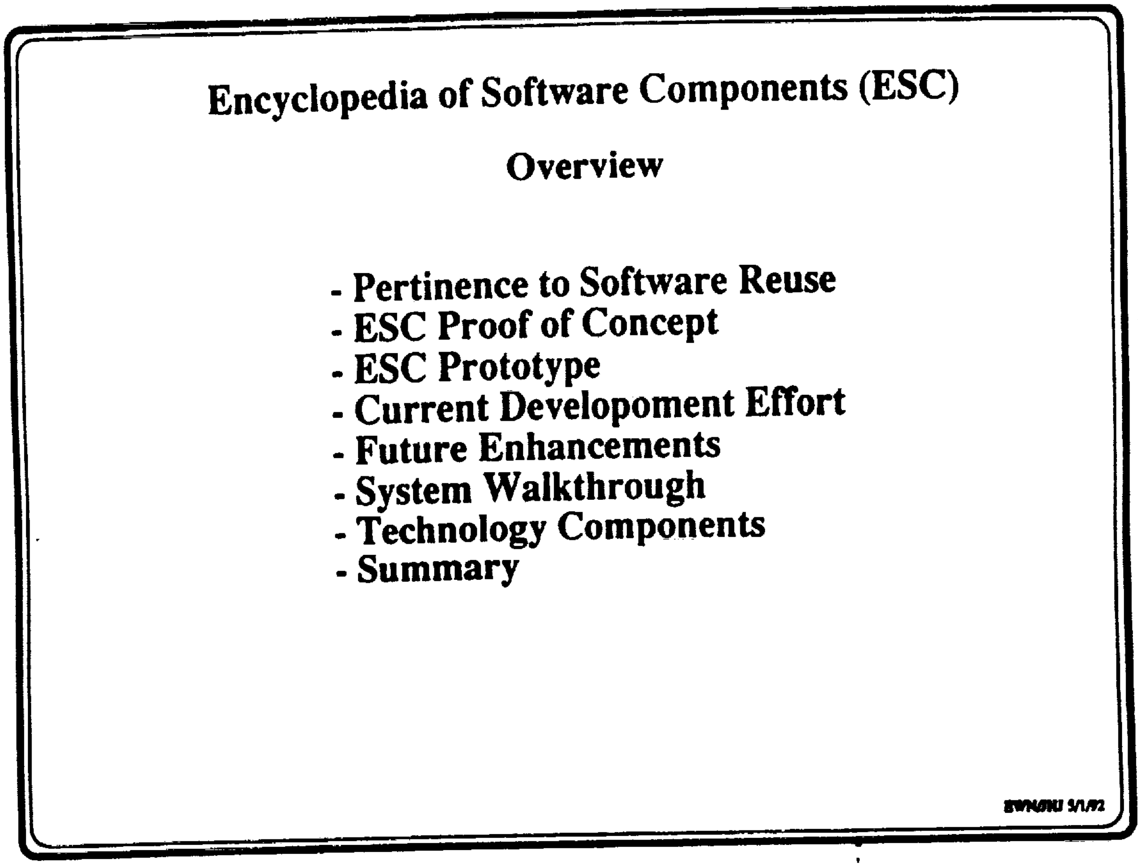
SWN/JRU 5/1/92

JPL

Hylite Accomplishments
(FY92 and Projected for FY93)

- Prototype for beta testing on color Macintoshs
- Graphical user interface (GUI) developed for inserting new components and property knowledge, for browsing and searching databases, and for retrieving software from selected networks
- Contain library of math software and library of data structures and algorithms
- Presently being adapted as a graphical front-end for national software exchange experiment
- To be adapted as an intelligent front-end to NAIF, a library of software tools and datasets for space flight navigation system
- Investigate collaborative arrangements with Ames and Langley on applications in aeronautics, materials, and structures areas
- Connect to Netlib, a very popular online software library
- Initiate SBIR contract for commercial spin-off

SWN/JRU 5/1/92

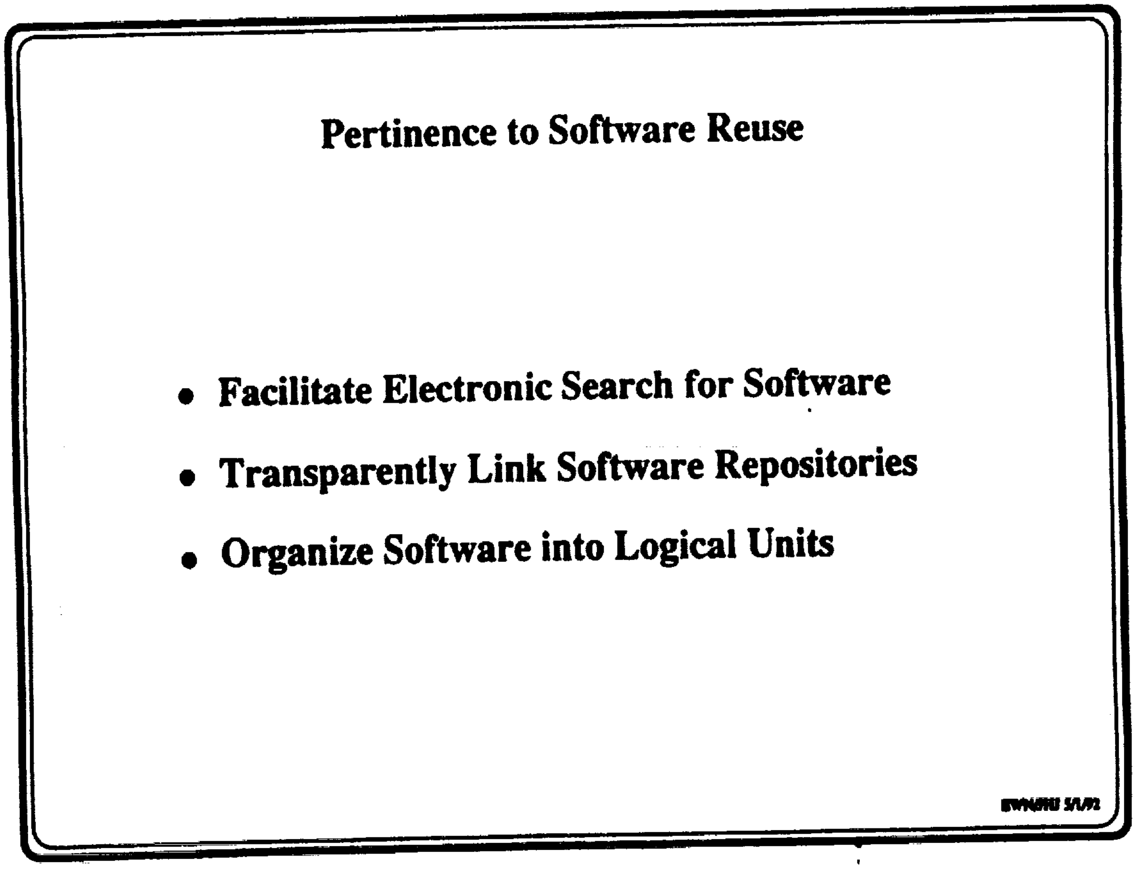


Encyclopedia of Software Components (ESC)

Overview

- Pertinence to Software Reuse
- ESC Proof of Concept
- ESC Prototype
- Current Developoment Effort
- Future Enhancements
- System Walkthrough
- Technology Components
- Summary

SWN/JU 5/1/92



Pertinence to Software Reuse

- Facilitate Electronic Search for Software
- Transparently Link Software Repositories
- Organize Software into Logical Units

SWN/JU 5/1/92

ESC Proof of Concept

- Development Environment
  - SuperCard on Macintosh
  - Think C
- Features
  - Browser
  - Publisher
  - History List
- Lessons Learned
  - Stronger programming language needed
  - Better representation for software classification
  - Software classification needed
  - Automatic GUI generation needed

ESC Prototype

- Development Environment
  - Macintosh Allegro Common Lisp
  - Think C
  - PixelPaint Professional
  - Canvas 2.0
- Features
  - Browser
  - Searcher
  - Publisher
  - Retrieval Mechanism
  - Classification Mechanism
    - -- Linnaeus
    - -- Semantic Networks

JPL

Current ESC Development Effort

- Port to Unix workstations running under the X window system
- Inclusion of AI technologies
  - Intelligent retrieval
  - Learning from experience
  - User modeling
  - Incomplete retrieval statements
  - Spelling and grammar correction
  - Automatic suggestion of alternative retrieval requests when a trieval fails
- Updating the Prototype to include other capabilities
  - History List
  - Hypertext

EWN/JTU 5/1/92

JPL

ESC System Walkthrough

EWN/JTU 5/1/92

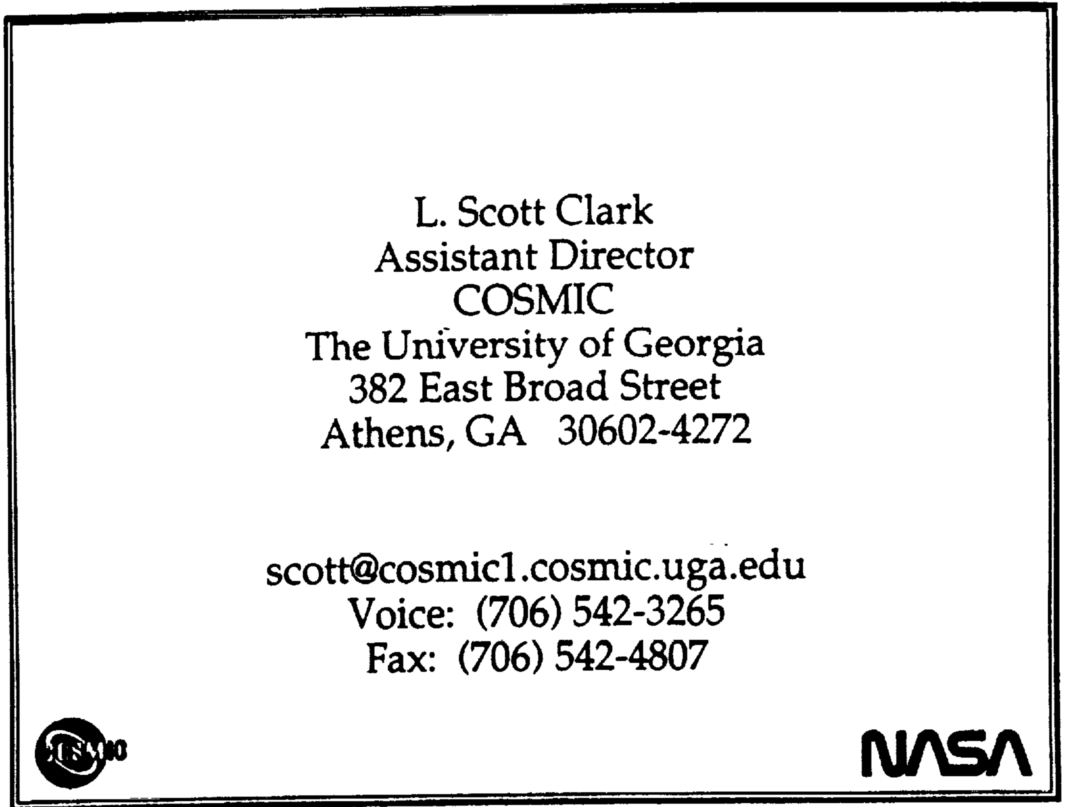
Technology Components

- Object Oriented Databases
- Classification Scheme based on Semantic Networks
- Automatic GUI generation

Summary

- HyLite represents an important area of NASA's computer science base research and development
- It is promising in significant potential pay-offs to a broad range of NASA problems
- Software resue is one important application
- With mutual leveraging among NASA Centers to industry and universities, we can make significant progress in the next 3-5 years
- JPL is strongly motivated to cooperate with other NASA Centers

COSMIC: Still Changing After All These Years

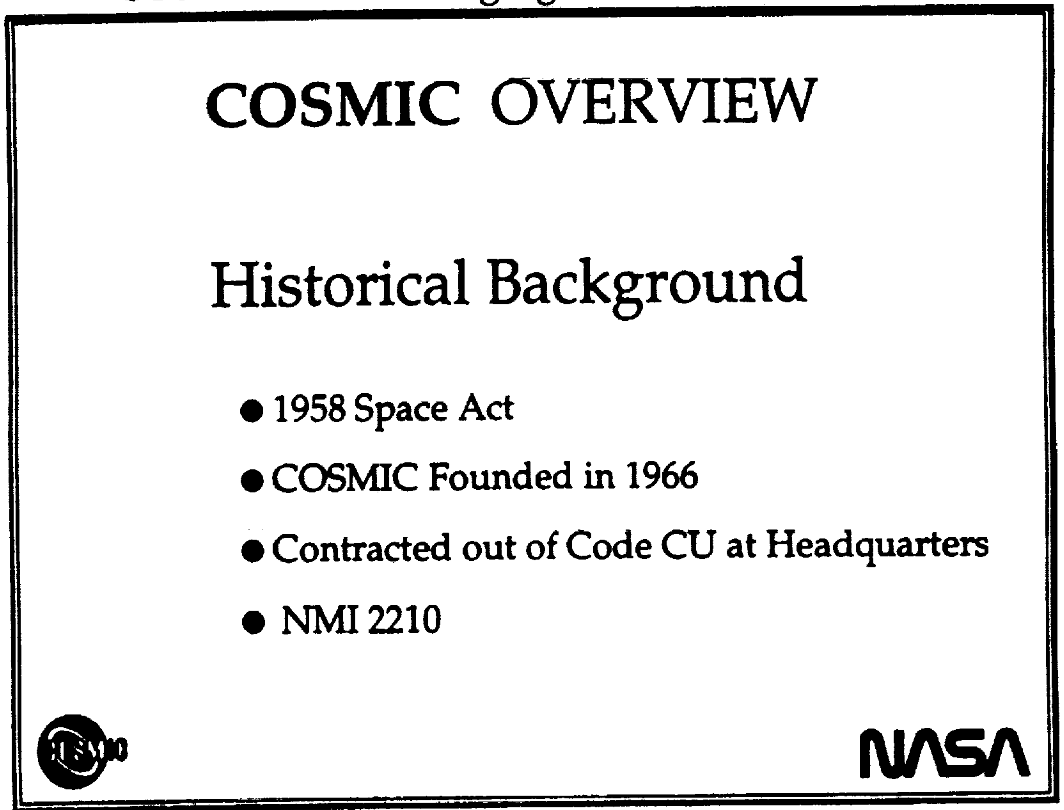


L. Scott Clark
Assistant Director
COSMIC
The University of Georgia
382 East Broad Street
Athens, GA 30602-4272

scott@cosmic1.cosmic.uga.edu
Voice: (706) 542-3265
Fax: (706) 542-4807

Software Reuse Tools Workshop (5/92) PAGE 1

COSMIC: Still Changing After All These Years



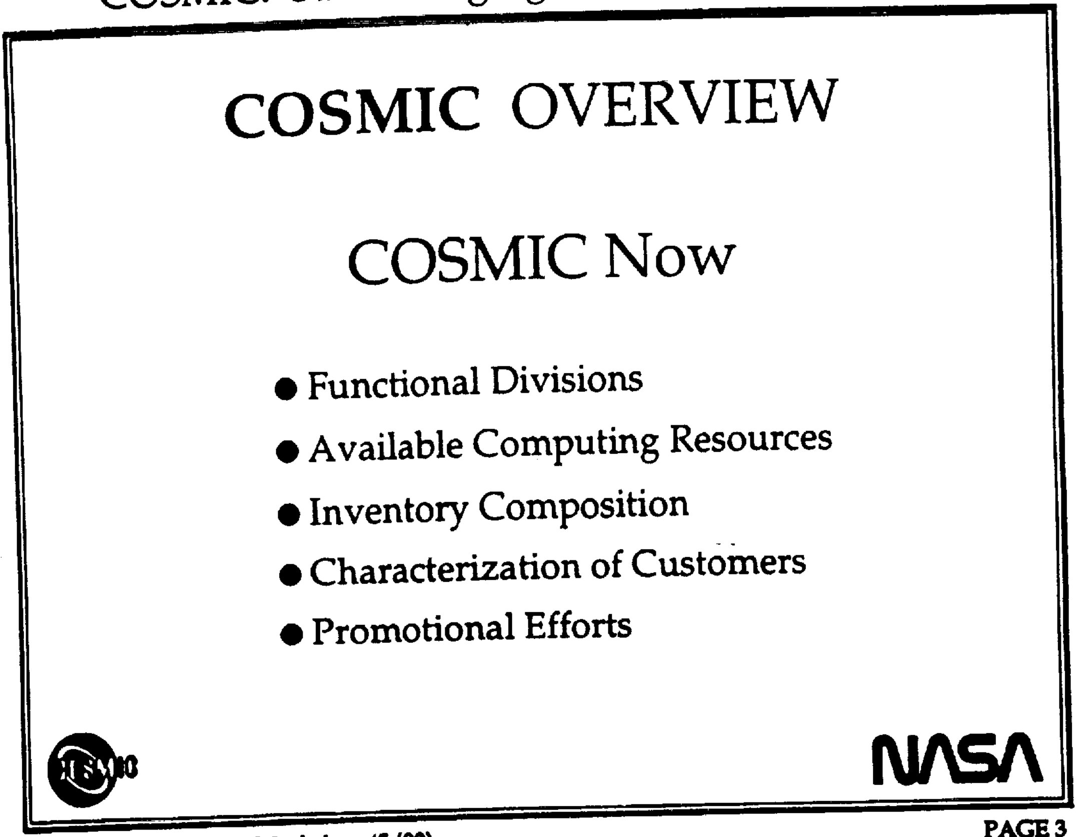
COSMIC OVERVIEW

Historical Background

- 1958 Space Act
- COSMIC Founded in 1966
- Contracted out of Code CU at Headquarters
- NMI 2210

Software Reuse Tools Workshop (5/92) PAGE 2

COSMIC: Still Changing After All These Years

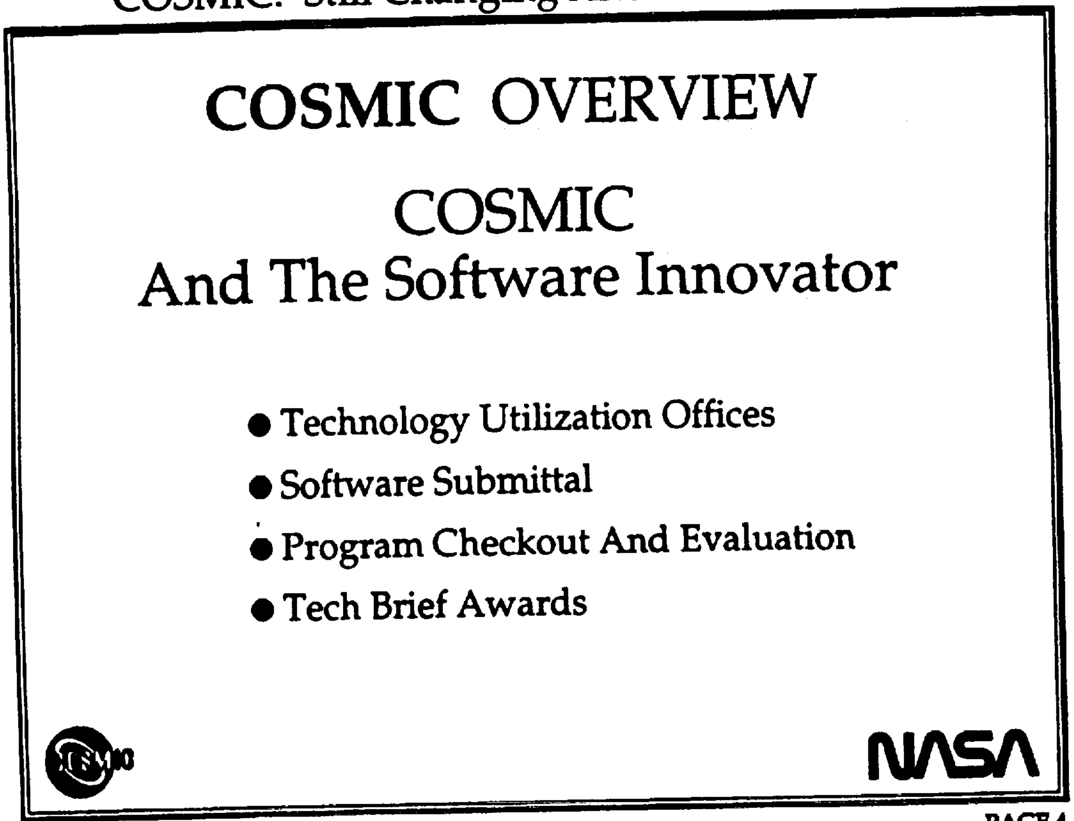


COSMIC OVERVIEW

COSMIC Now

- Functional Divisions
- Available Computing Resources
- Inventory Composition
- Characterization of Customers
- Promotional Efforts

COSMIC: Still Changing After All These Years

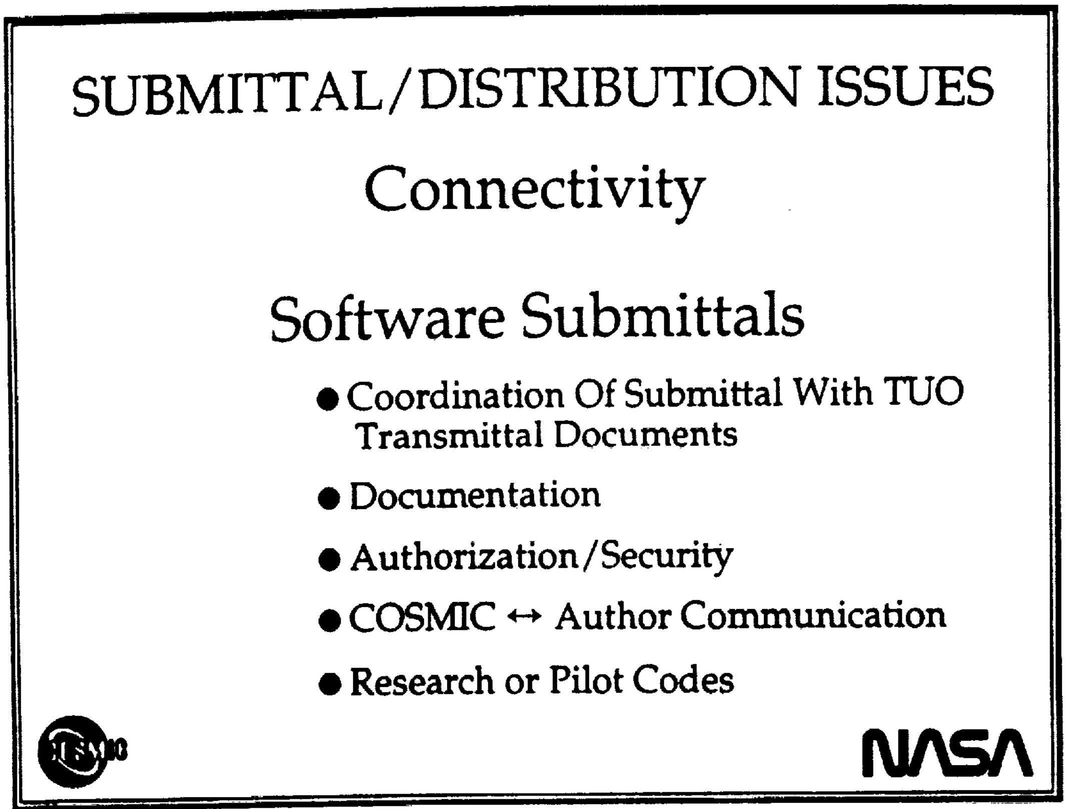


COSMIC OVERVIEW

COSMIC And The Software Innovator

- Technology Utilization Offices
- Software Submittal
- Program Checkout And Evaluation
- Tech Brief Awards

COSMIC: Still Changing After All These Years



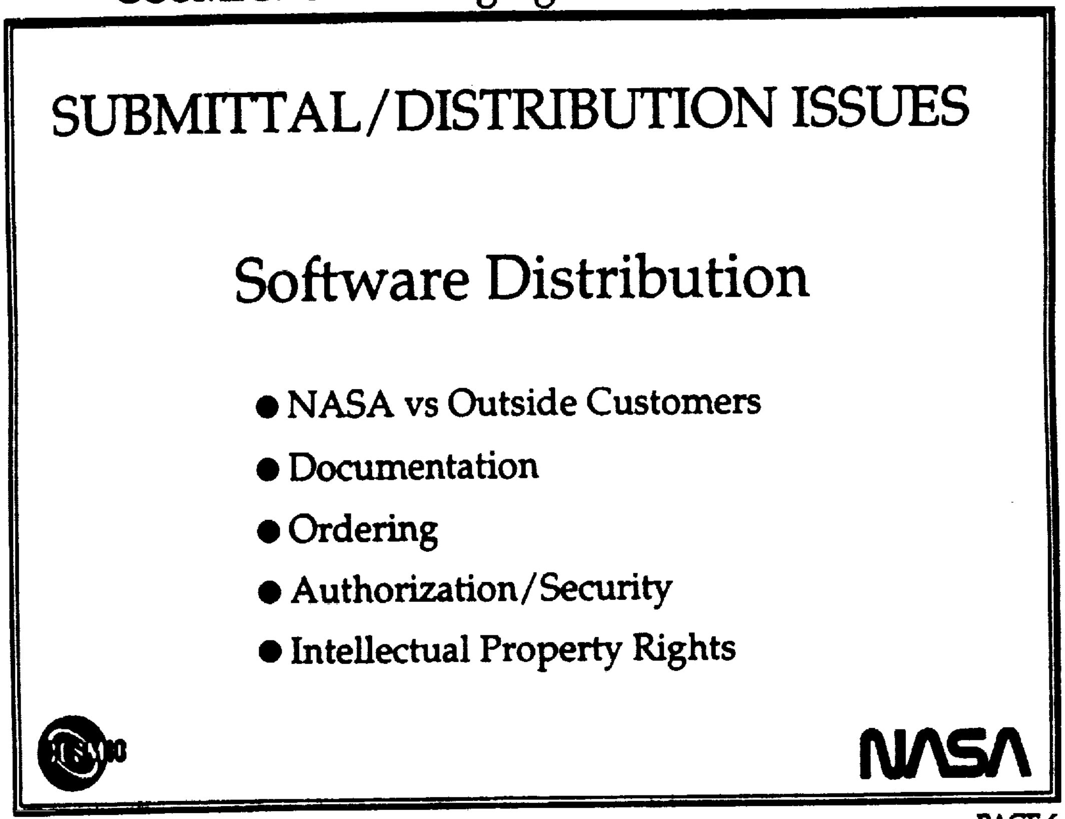
SUBMITTAL/DISTRIBUTION ISSUES

Connectivity

Software Submittals

- Coordination Of Submittal With TUO Transmittal Documents
- Documentation
- Authorization/Security
- COSMIC ↔ Author Communication
- Research or Pilot Codes

COSMIC: Still Changing After All These Years



SUBMITTAL/DISTRIBUTION ISSUES

Software Distribution

- NASA vs Outside Customers
- Documentation
- Ordering
- Authorization/Security
- Intellectual Property Rights

CERTIFICATION OF REUSABLE SOFTWARE COMPONENTS

Presentation to:

NASA Software Reuse Tools Workshop

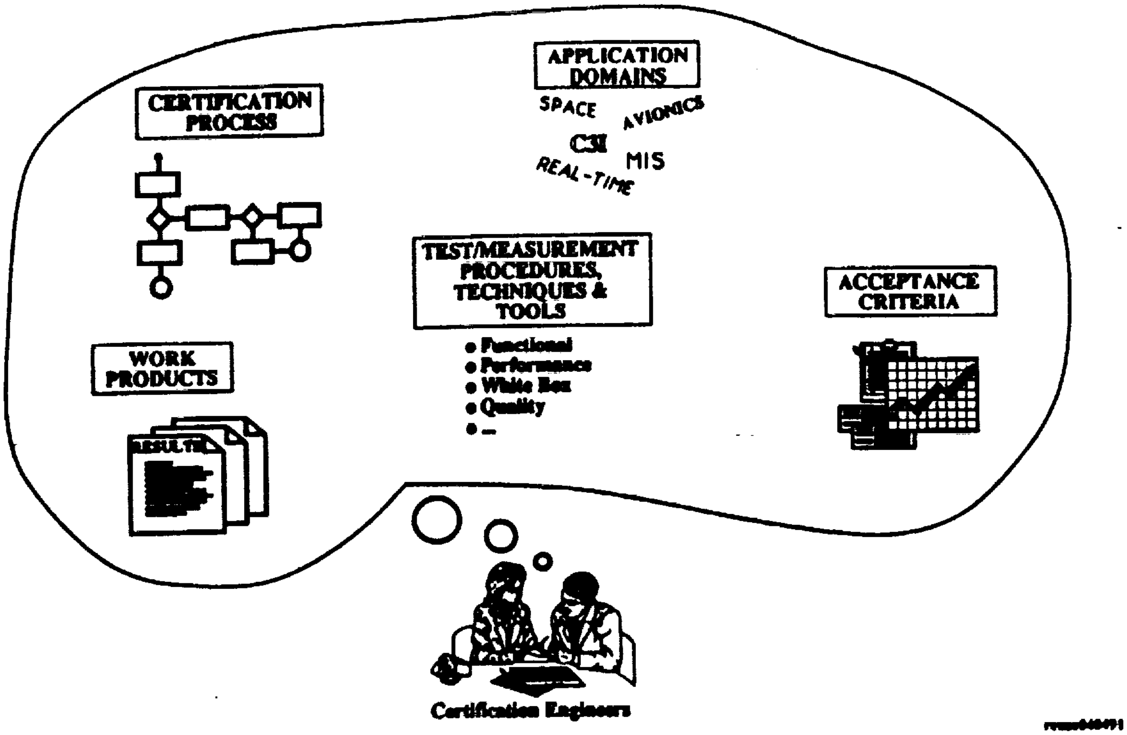
5-6 May 92
Rome Laboratory
Griffiss AFB NY 13441

Deborah Cerino/C3CB/DSN 587-2054

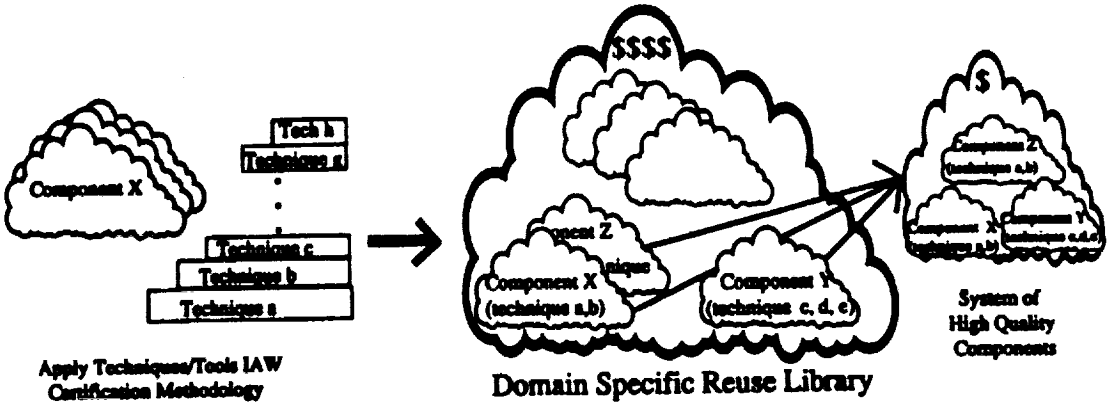
Overview

- What is Certification?
- Certification Considerations
    - Test Techniques
    - Formal Verification
    - Quality Analyses
- Research Areas
- Rome Laboratory Program Plan

Considerations For Certification Of Reusable Components



Certification Methodology for Reusable Software Components



Component X
Tech h
Technique g
Technique c
Technique b
Technique a
Apply Techniques/Tools IAW Certification Methodology
$$$$
Component Z
Component X (technique a,b)
Component Y (technique c, d, e)
Domain Specific Reuse Library
$
Component Z (technique a,b)
Component X (technique a,b)
Component Y (technique c,d,e)
System of High Quality Components

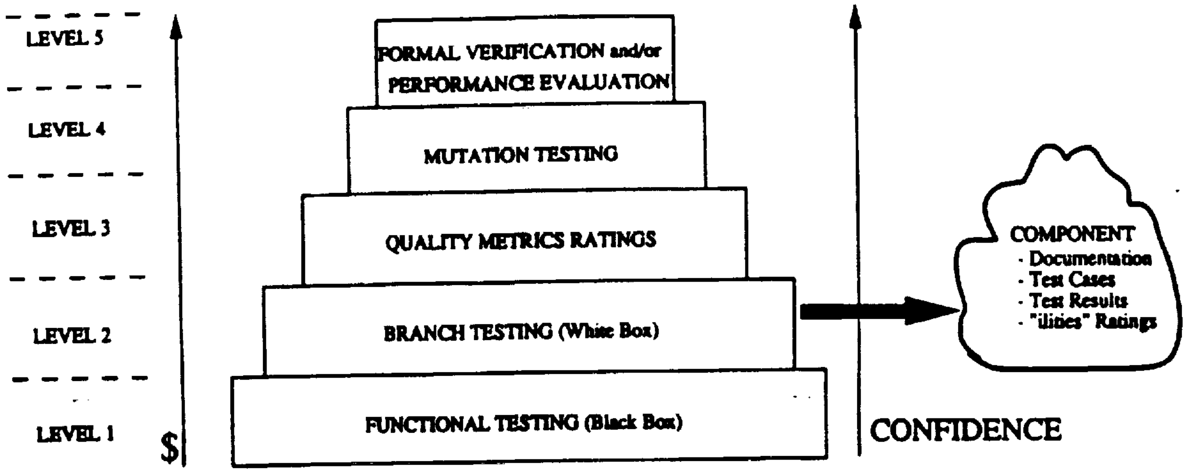
Why Certify Components?

- insure high quality
- provide degrees of confidence
- aid in reuse decisions vs development from scratch
- alleviate legal issues
- promote reuse; significant cost savings (over 50%)

What will this Program provide?

- certification process--multi-level
- advanced techniques/tools for component analyses (software test & verification, software quality assessment)
- another dimension for choosing reusable components (e.g., choose a highly tested over a poorly tested component)

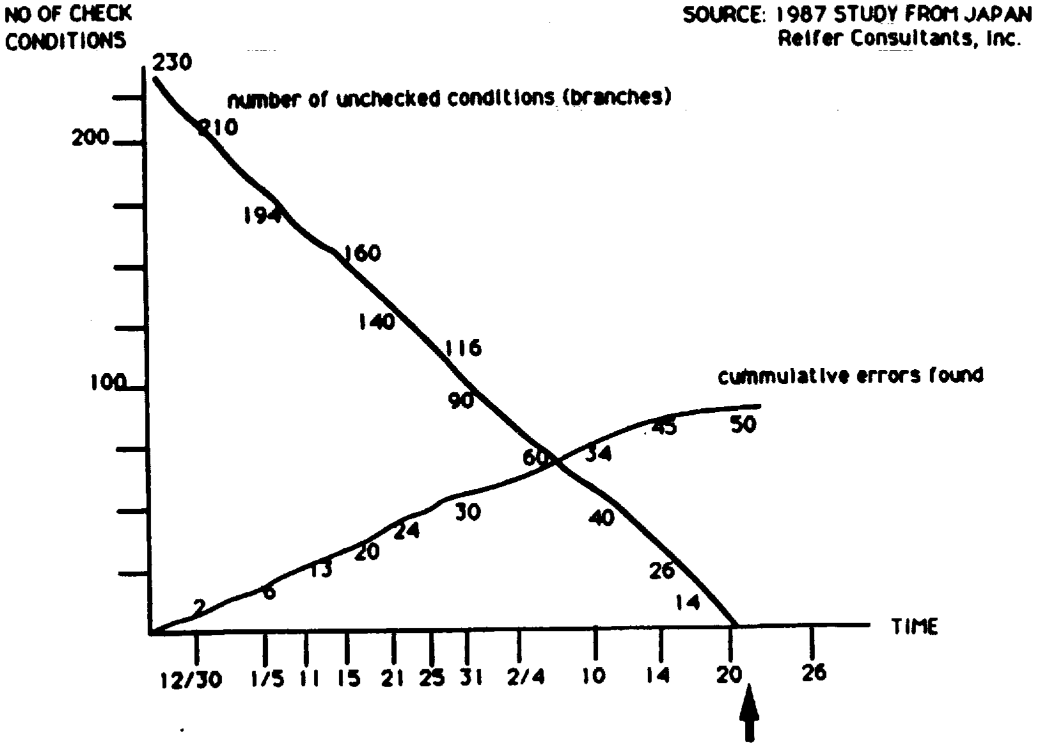
STRAWMAN CERTIFICATION STRATEGY



$ = COST TO PRODUCE & CERTIFY

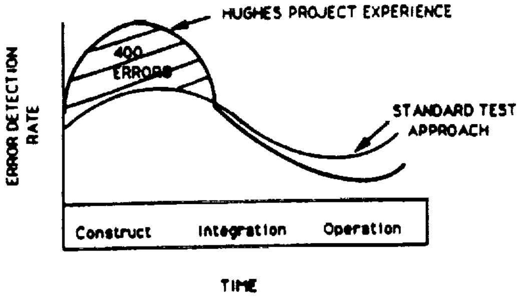
$ = COST OF PURCHASING COMPONENT

CORRELATION BETWEEN BRANCH TESTING AND ERRORS FOUND



SOURCE: 1987 STUDY FROM JAPAN
Reifer Consultants, Inc.

LIFE CYCLE COST BENEFITS USING AN AUTOMATED TEST TOOL



WHEN ERRORS ARE DETECTED EARLY		WHEN ERRORS ARE DETECTED LATER
COST: $400 \text{ errors} \times 2 \frac{\text{person days}}{\text{error}} =$ 800pd	VS	COST: $200 \text{ errors} \times 2 \frac{\text{pd}}{\text{error}} (4) + 200 \text{ errors} \times 2 \frac{\text{pd}}{\text{error}} (9)$ = 5200pd

△ 4400 pd

RL TEST & VERIFICATION TOOLS

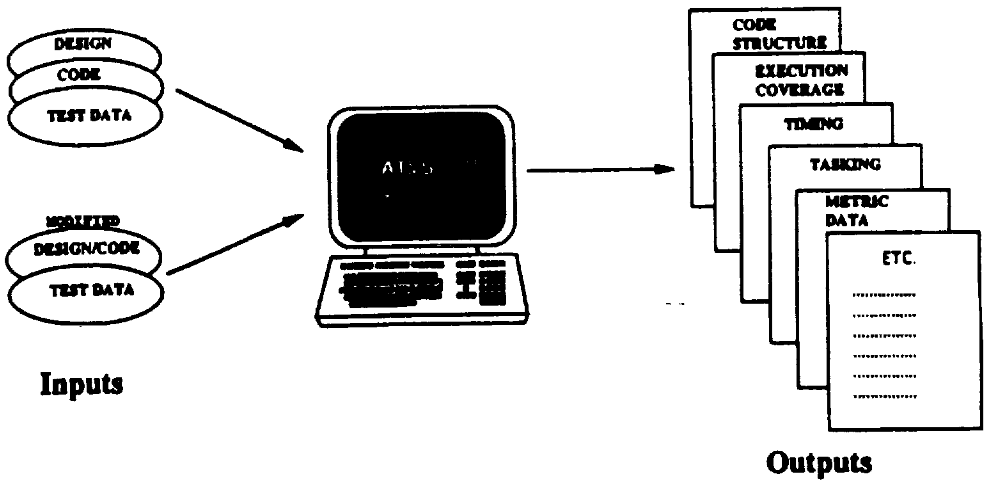
FORTRAN - 1979

JOVIAL J73 - 1983

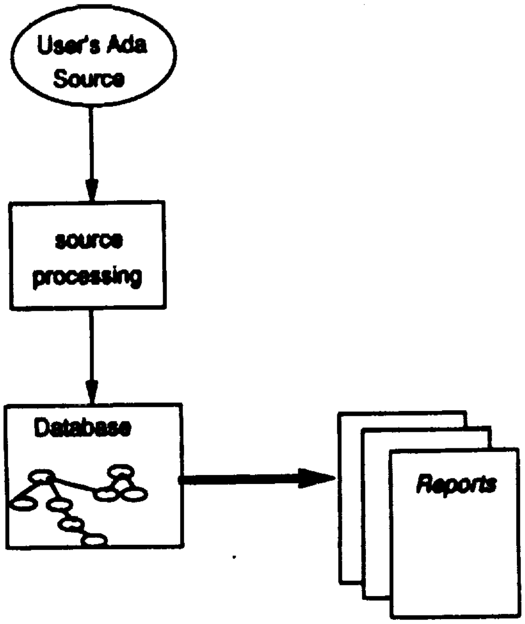
COBOL - 1983

ADA-1989

Ada Test & Verification System (ATVS)
Analyses Capabilities



STATIC ANALYSES



LOOK AT CODE STRUCTURE

What are all the variable, parameters, etc. names ?

Where are they located in the code?

Which units call/are called by other units?

What does the unit nesting look like?

How many LOC in each unit?

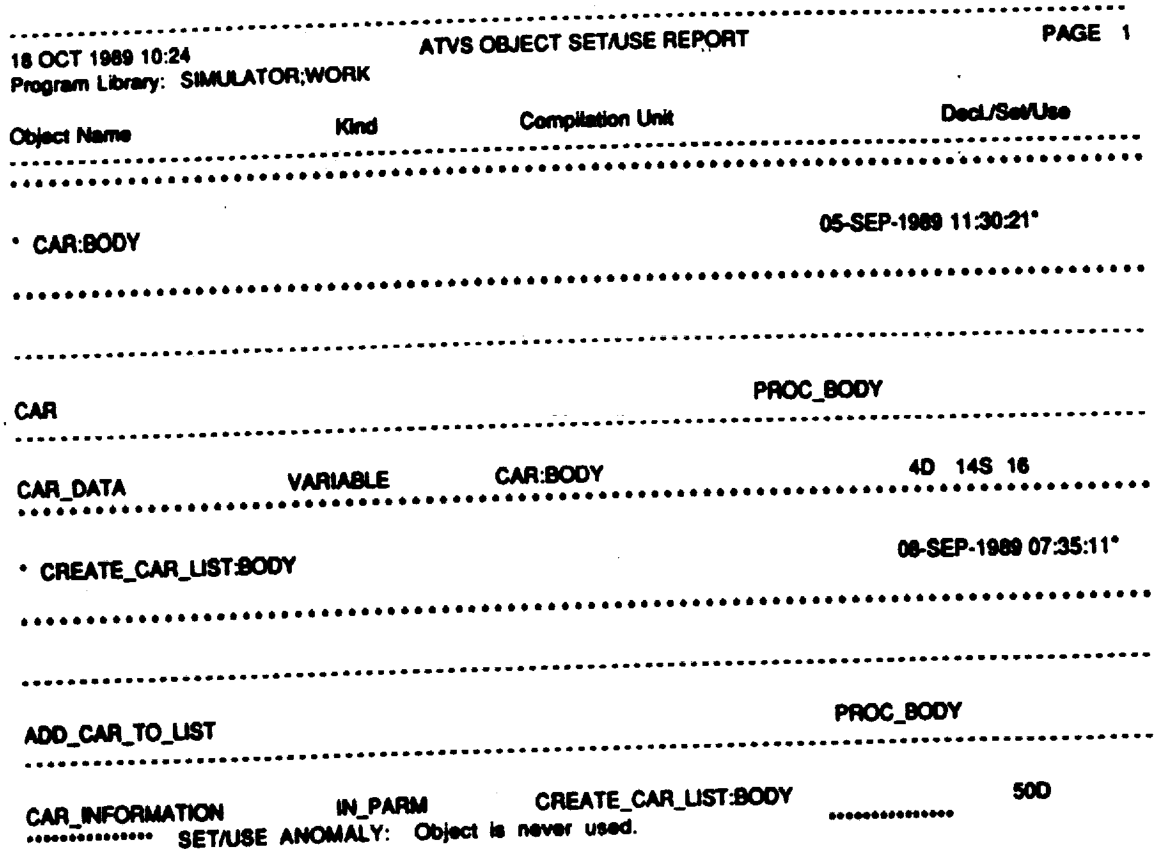
How many tasks?

How many procedures?

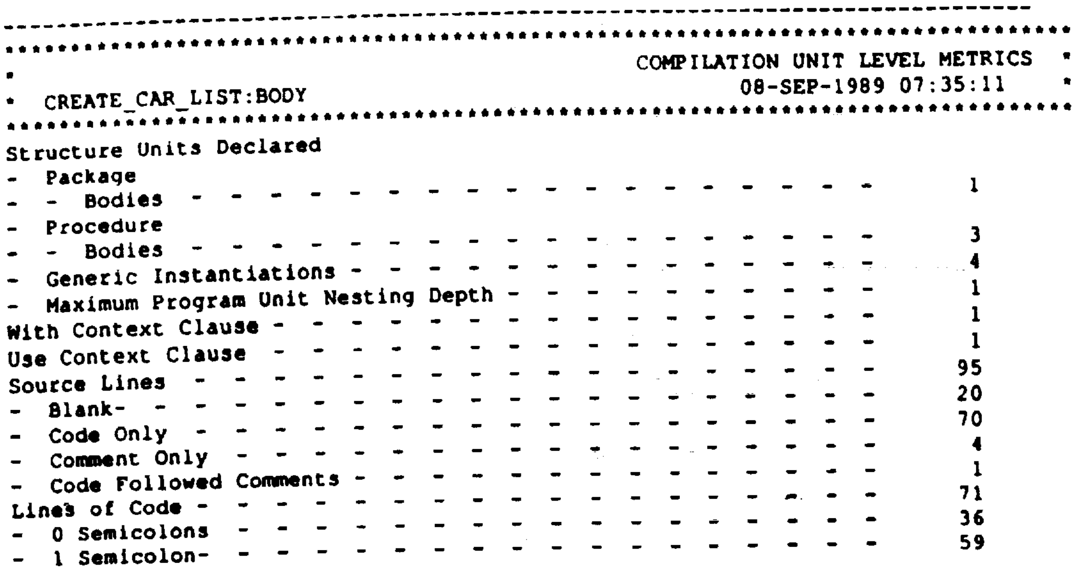
STATIC ANALYSES

BENEFIT: Identify a Potential Problem

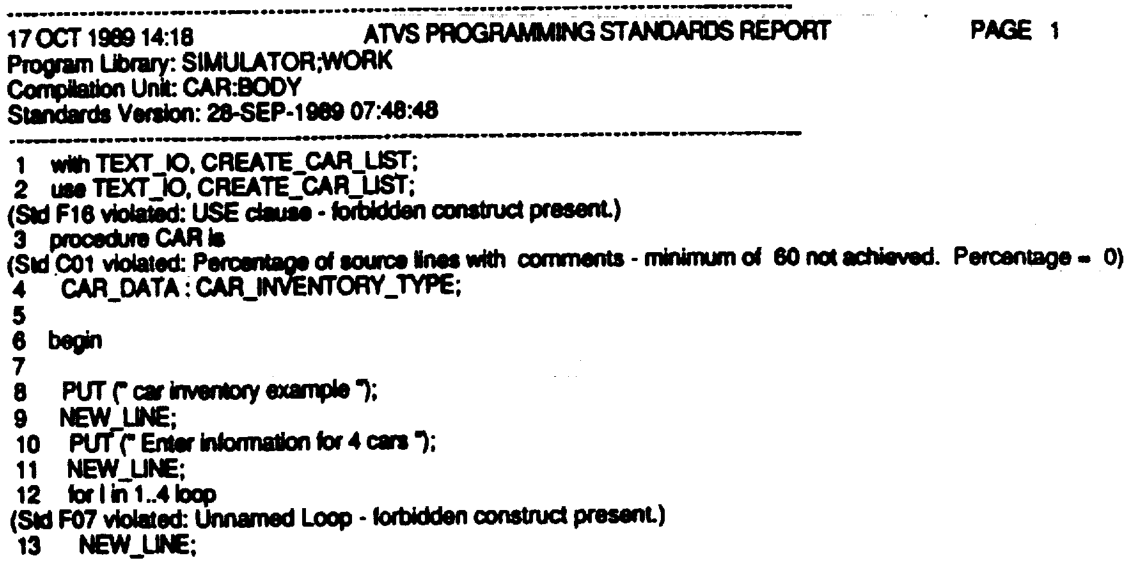
- SET/USE REPORT
- SOURCE CODE REPORT
- PROGRAMMING STANDARDS REPORT



```
..........................................................................
18 OCT 1989 10:24             ATVS OBJECT SET/USE REPORT            PAGE  1
Program Library:  SIMULATOR;WORK

Object Name            Kind         Compilation Unit              Decl./Set/Use
..........................................................................
••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

                                                    05-SEP-1989 11:30:21*
* CAR:BODY
••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

..........................................................................
                                              PROC_BODY
CAR
..........................................................................

CAR_DATA               VARIABLE     CAR:BODY                    4D  14S  16
••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
                                                        08-SEP-1989 07:35:11*
* CREATE_CAR_LIST:BODY
••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••

..........................................................................
                                                  PROC_BODY
ADD_CAR_TO_LIST
..........................................................................

CAR_INFORMATION        IN_PARM      CREATE_CAR_LIST:BODY  •••••••••••••••  50D
•••••••••••••• SET/USE  ANOMALY:  Object is never used.
```



```
-----------------------------------------------------------------------
***********************************************************************
*                                          COMPILATION UNIT LEVEL METRICS  *
*  CREATE_CAR_LIST:BODY                           08-SEP-1989 07:35:11    *
***********************************************************************
Structure Units Declared
-  Package
-  -  Bodies - - - - - - - - - - - - - - - - - - - - - - -         1
-  Procedure
-  -  Bodies - - - - - - - - - - - - - - - - - - - - - - -         3
-  Generic Instantiations - - - - - - - - - - - - - - - - -         4
-  Maximum Program Unit Nesting Depth - - - - - - - - - - -         1
With Context Clause - - - - - - - - - - - - - - - - - - - -         1
Use Context Clause  - - - - - - - - - - - - - - - - - - - -         1
Source Lines  - - - - - - - - - - - - - - - - - - - - - - -        95
-  Blank-  - - - - - - - - - - - - - - - - - - - - - - - - -        20
-  Code Only  - - - - - - - - - - - - - - - - - - - - - - -        70
-  Comment Only  - - - - - - - - - - - - - - - - - - - - - -         4
-  Code Followed Comments - - - - - - - - - - - - - - - - -         1
Lines of Code -  - - - - - - - - - - - - - - - - - - - - - -        71
-  0 Semicolons  - - - - - - - - - - - - - - - - - - - - - -        36
-  1 Semicolon-  - - - - - - - - - - - - - - - - - - - - - -        59
```



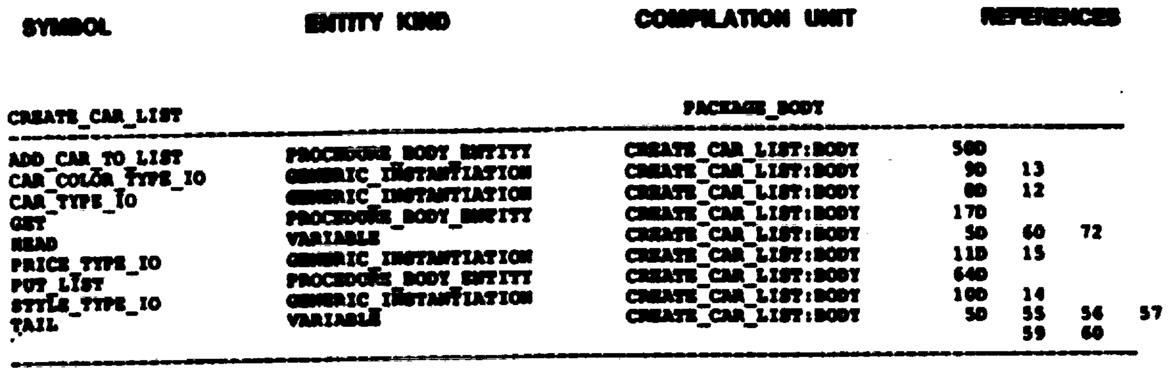
```
----------------------------------------------------------------------------
17 OCT 1989 14:18          ATVS PROGRAMMING STANDARDS REPORT          PAGE  1
Program Library: SIMULATOR;WORK
Compilation Unit: CAR:BODY
Standards Version: 28-SEP-1989 07:48:48
----------------------------------------------------------------------------
 1   with TEXT_IO, CREATE_CAR_LIST;
 2   use TEXT_IO, CREATE_CAR_LIST;
(Std F16 violated: USE clause - forbidden construct present.)
 3   procedure CAR is
(Std C01 violated: Percentage of source lines with  comments - minimum of  60 not achieved.  Percentage = 0)
 4     CAR_DATA : CAR_INVENTORY_TYPE;
 5
 6   begin
 7
 8     PUT (" car inventory example ");
 9     NEW_LINE;
 10     PUT (" Enter information for 4 cars ");
 11     NEW_LINE;
 12     for I in 1..4 loop
(Std F07 violated: Unnamed Loop - forbidden construct present.)
 13       NEW_LINE;
```

STATIC ANALYSES

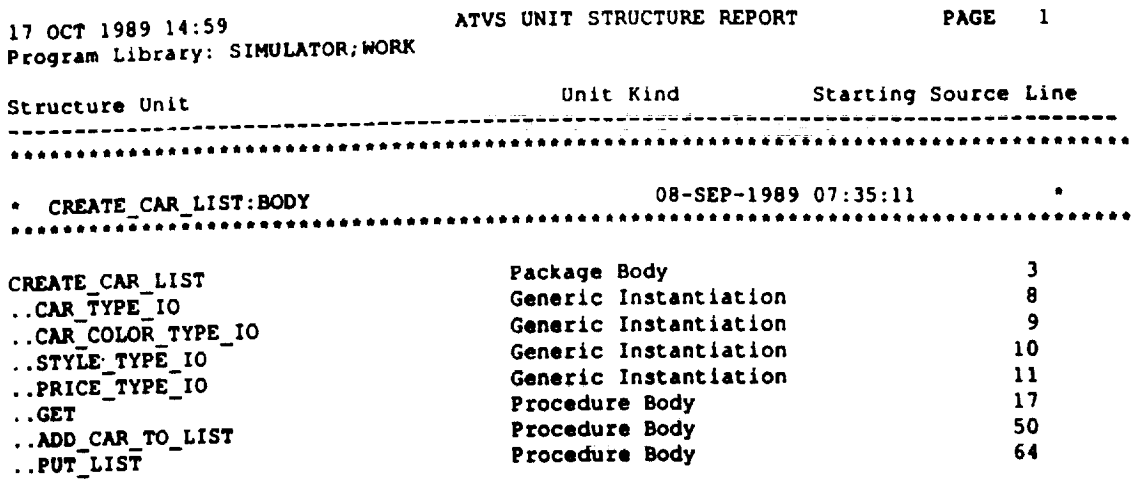
BENEFIT: Aid Maintenance of Software

- ENTITY CROSS REFERENCE REPORT
- UNIT STRUCTURE REPORT

ENTITY CROSS REFERENCE REPORT

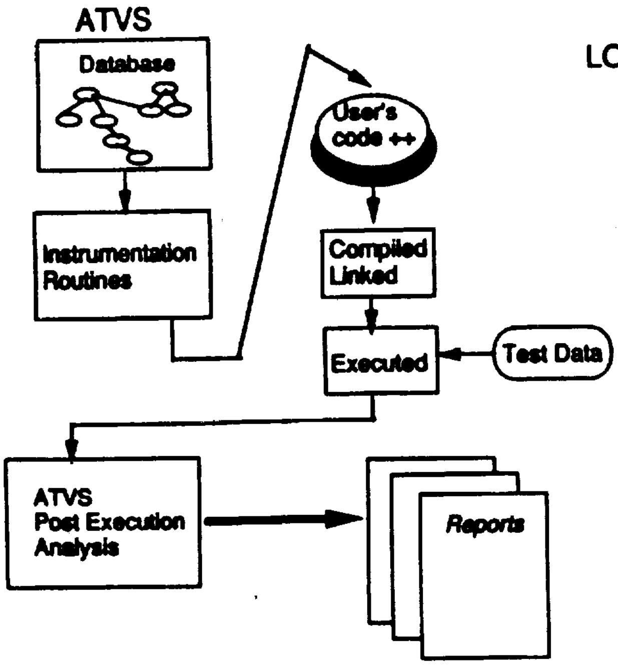


```
SYMBOL               ENTITY KIND                COMPILATION UNIT        REFERENCES


CREATE_CAR_LIST                                  PACKAGE_BODY
---------------------------------------------------------------------------------------
ADD_CAR_TO_LIST      PROCEDURE_BODY_ENTITY      CREATE_CAR_LIST:BODY    50D
CAR_COLOR_TYPE_IO    GENERIC_INSTANTIATION      CREATE_CAR_LIST:BODY     9D   13
CAR_TYPE_IO          GENERIC_INSTANTIATION      CREATE_CAR_LIST:BODY     8D   12
GET                  PROCEDURE_BODY_ENTITY      CREATE_CAR_LIST:BODY    17D
HEAD                 VARIABLE                   CREATE_CAR_LIST:BODY     5D   60   72
PRICE_TYPE_IO        GENERIC_INSTANTIATION      CREATE_CAR_LIST:BODY    11D   15
PUT_LIST             PROCEDURE_BODY_ENTITY      CREATE_CAR_LIST:BODY    64D
STYLE_TYPE_IO        GENERIC_INSTANTIATION      CREATE_CAR_LIST:BODY    10D   14
TAIL                 VARIABLE                   CREATE_CAR_LIST:BODY     5D   55   56   57
                                                                              59   60
---------------------------------------------------------------------------------------
```



```
17 OCT 1989 14:59                ATVS UNIT STRUCTURE REPORT          PAGE    1
Program Library: SIMULATOR;WORK

Structure Unit                               Unit Kind          Starting Source Line
------------------------------------------------------------------------------------
************************************************************************************

*  CREATE_CAR_LIST:BODY                               08-SEP-1989 07:35:11          *
************************************************************************************

CREATE_CAR_LIST                           Package Body                           3
..CAR_TYPE_IO                             Generic Instantiation                  8
..CAR_COLOR_TYPE_IO                       Generic Instantiation                  9
..STYLE_TYPE_IO                           Generic Instantiation                 10
..PRICE_TYPE_IO                           Generic Instantiation                 11
..GET                                     Procedure Body                        17
..ADD_CAR_TO_LIST                         Procedure Body                        50
..PUT_LIST                                Procedure Body                        64
```

DYNAMIC ANALYSES



LOOK AT RUN TIME BEHAVIOR

- Which sections of code have been executed?
- How many times has each branch been executed?
- How much time was spent in a particular section of code?
- What sequence of tasks was executed?

DYNAMIC ANALYSES

BENEFIT: Provide Test Coverage

- EXECUTION COVERAGE
  - UNIT COVERAGE REPORT
  - BRANCH COVERAGE REPORT
- TIMING REPORT
- TASKING REPORT

UNIT COVERAGE REPORT

```
_____________________________________________________________________________________
                                        | N U M B E R   O F   E X E C U T I O N S |
Comp. Unit             Line             |          |  ( Normalized to Maximum )   |
   Structure Unit        #      Kind    |  Count   |  20    40    60    80   100|
                                        |          | -----.-----.-----.-----.-----.|
_______________________________________________________|                              |
MOD_FUNCTIONS:BODY                                 |                              |
  MOD_FUNCTIONS          2    PKG BDY         0    |                              |
  CALC_LEAP_YEAR         4    FUNC BDY        3    |******************************|
  GET_DATE              16    FUNC BDY        2    |********************          |
                                                   |                              |
DATE_MANIP:BODY                                    |                              |
  DATE_MANIP             6    PKG BDY         0    |                              |
  NEXT_DATE              8    FUNC BDY        3    |******************************|
                                                   |                              |
DATE_LAB:BODY                                      |                              |
  DATE_LAB               4    PROC BDY        1    |**********                    |
  block_24              24    BLK STMT        3    |******************************|
  block_43              43    BLK STMT        3    |******************************|
                                                   |                              |
=====================================================================================
```

BRANCH COVERAGE REPORT

```
  Structure                  | Invo-    Total    Branches  Percent   |      Branches
   Unit/                     |cations  Branches  Executed  Branches  |        Not
   Line                      |                             Executed  |      Executed
==========================================================================================

  COMPILATION UNIT:  MOD_FUNCTIONS:BODY

MOD_FUNCTIONS    / 2          0        0         0          0 %   |
CALC_LEAP_YEAR   / 4          3        4         2         50 %   |  2  3
GET_DATE         / 16         2        1         1         100%   |

  COMPILATION UNIT:  DATE_MANIP:BODY

DATE_MANIP       / 6          0        0         0          0 %   |
NEXT_DATE        / 8          3        18        5         28 %   |  5  6  7  8  9  10
                                                                     11 12 13 14 15 16
```

BRANCH COVERAGE REPORT

```
**************************************************************
* MOD_FUNCTIONS:BODY          1989/ 8/ 17  66332
**************************************************************

1
2  package body MOD_FUNCTIONS is
3
4      function CALC_LEAP_YEAR( Test_Date : in Date ) return boolean is

5
6      begin
 ** Branch  1 PROGRAM UNIT START
7          if ( Test_Date.Year mod 400 = 0 ) then
 ** Branch  2 IF CONDITION TRUE
8              return True;
9          elsif ( Test_Date.Year mod 4=0 ) and ( Test_Date.Year mod 100 /=0 )
```

ATVS STATUS

- Government Version Completed (Sep 89)
- Commercial Version Currently Available - AdaQuest
  - Fully supported
  - Robust
  - POSIX/Motif Compatibility - Jul 92
  - Additional standards from <u>Ada Quality & Style Guide</u> - Jul 92

MUTATION TESTING

<u>PROGRAM A</u>

```
BEGIN
READ K
IF K < 10
THEN
J:=K+5
ELSE
J:=K+10
ENDIF
WRITE J
END
```

ORIGINAL PROGRAM

<u>PROGRAM B</u>

```
BEGIN
READ K
IF K <= 10
THEN
J:=K+5
ELSE
J:=K+10
ENDIF
WRITE J
END
```

MUTANT PROGRAM

MUTANT : A VARIATION OF THE ORIGINAL PROGRAM THAT CONTAINS A SINGLE INSERTION OR DEVIATION

MUTATION TESTING

EXISTING MUTATION TESTING SYSTEM CAPABILITIES

- ANALYZES FORTRAN CODE
- AUTOMATES MUTATION TESTING PROCESS
  (GENERATES AND EXECUTES MUTANTS)
- MAINTAINS DATABASE OF TESTING STATUS
- LOCALIZES PROGRAM ERRORS

USER RESPONSIBILITIES

- GENERATE TEST CASES
- VERIFY TESTCASE RESULTS
- ESTABLISH TEST COMPLETION CRITERIA
- IDENTIFY PROGRAM ERRORS

MUTATION TESTING

MUTANT OPERATORS - A SIMPLE TRANSFORMATION

• STATEMENT ANALYSIS

- REPLACE EACH STATEMENT BY "CONTINUE"
- REPLACE EACH STATEMENT BY "RETURN"
- REPLACE THE TARGET LABEL IN EACH "DO" STATEMENT

• PREDICATE AND DOMAIN ANALYSIS

- TAKE THE ABSOLUTE VALUE OF AN EXPRESSION
- REPLACE ONE ARITHMETIC OPERATOR BY ANOTHER
- REPLACE ONE RELATIONAL OPERATOR BY ANOTHER
- REPLACE ONE LOGICAL OPERATOR BY ANOTHER

• COINCIDENTAL CORRECTNESS

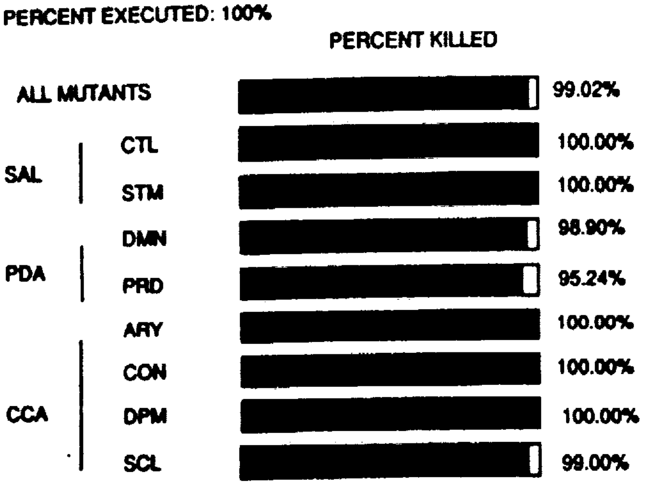
- REPLACE A SCALAR VARIABLE
- REPLACE AN ARRAY REFERENCE
- REPLACE A CONSTANT

MUTATION TESTING

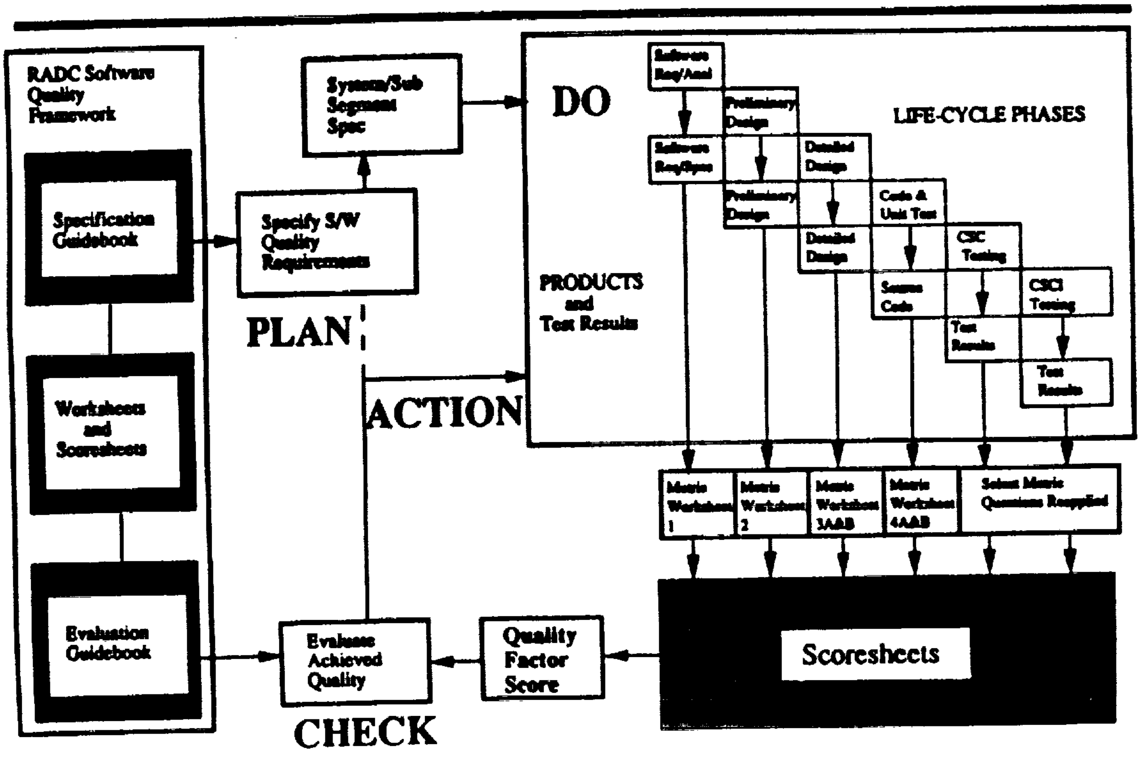
MUTANT STATUS

NUMBER OF MUTANTS GENERATED: 307

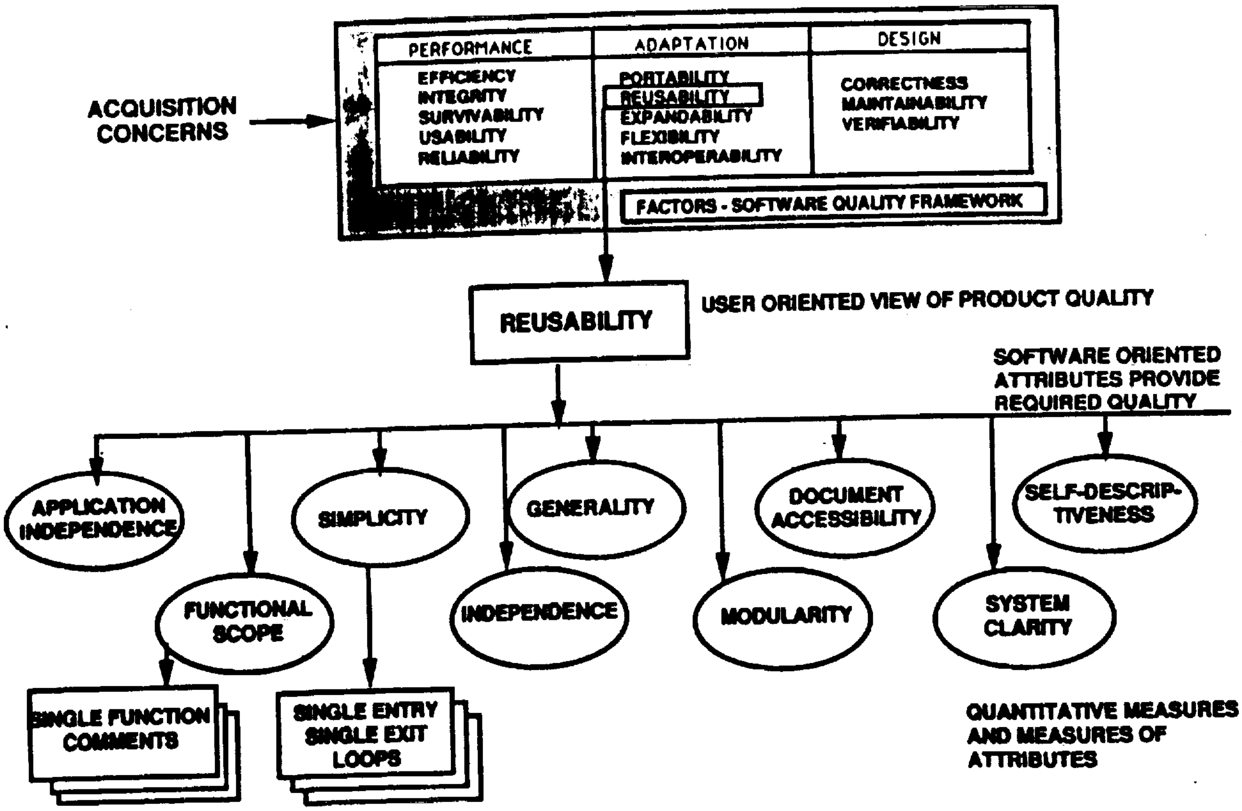
PERCENT EXECUTED: 100%



RL SOFTWARE QUALITY FRAMEWORK APPLICATION



QUALITY REPRESENTATION



ACQUISITION CONCERNS

PERFORMANCE
EFFICIENCY
INTEGRITY
SURVIVABILITY
USABILITY
RELIABILITY

ADAPTATION
PORTABILITY
REUSABILITY
EXPANDABILITY
FLEXIBILITY
INTEROPERABILITY

DESIGN
CORRECTNESS
MAINTAINABILITY
VERIFIABILITY

FACTORS - SOFTWARE QUALITY FRAMEWORK

REUSABILITY

USER ORIENTED VIEW OF PRODUCT QUALITY

SOFTWARE ORIENTED ATTRIBUTES PROVIDE REQUIRED QUALITY

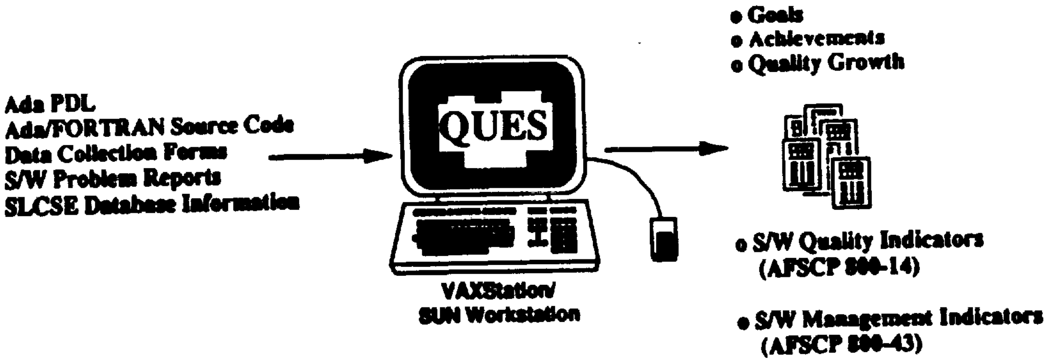
APPLICATION INDEPENDENCE
FUNCTIONAL SCOPE
SIMPLICITY
GENERALITY
INDEPENDENCE
MODULARITY
DOCUMENT ACCESSIBILITY
SYSTEM CLARITY
SELF-DESCRIP-TIVENESS

SINGLE FUNCTION COMMENTS
SINGLE ENTRY SINGLE EXIT LOOPS

QUANTITATIVE MEASURES AND MEASURES OF ATTRIBUTES

Ref: Specification Of Software Quality Attributes (RADC-TR-85-37) Vols I-III

QUality Evaluation System (QUES)



Ada PDL
Ada/FORTRAN Source Code
Data Collection Forms
S/W Problem Reports
SLCSE Database Information

QUES

VAXStation/
SUN Workstation

- Goals
- Achievements
- Quality Growth

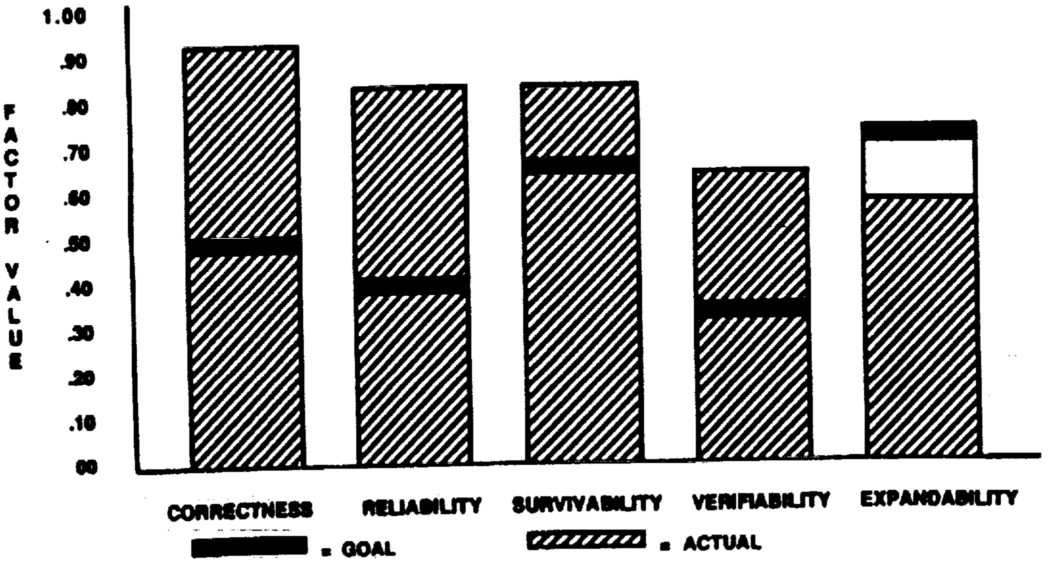
- S/W Quality Indicators (AFSCP 800-14)
- S/W Management Indicators (AFSCP 800-43)

- Automates The RADC S/W Quality Framework Evaluation Guidebook (RADC-TR-85-37, Volume III)
- Supports Acquisition Managers, Project Managers, & Engineers
- Allows Quality Goals To Be Specified
- Assesses Software Product Quality

SOFTWARE QUALITY GOAL REPORT

PROJECT:
PHASE: REQUIREMENTS
LEVEL: CSCI
ENTITY NAME: AR1.0
METRIC CALCULATION DATE: 10/05/85

DATE: 12/05/85



FRAMEWORK EXPERIENCE AND RESULTS (JAPAN)

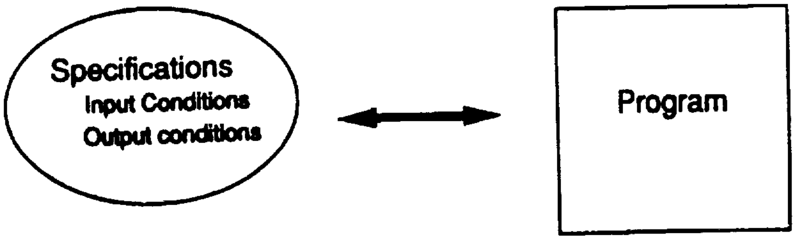
AVERAGE 3% OF DEVELOPMENT COST PER FACTOR

25% SAVINGS THRU FULL-SCALE DEVELOPMENT

51% SAVINGS AFTER 1 YEAR MAINTENANCE

Ref: "Integrating Software Quality Metrics with Software QA," Gerald E. Murine, Quality Progress, Nov 1988.
Data from Nippon Electric Company (NEC) projects. Gerald Murine president of METRIQS, Inc.

FORMAL VERIFICATION



DEFINITION: Collection of techniques that apply the formality and rigor of mathematics to the task of proving the consistency between an algorithmic solution and a rigorous, complete specification of the intent (behavior) of the solution

Develop new techniques for insertion into Certification Methodology

- **Software Fault Tolerance**
- **V & V of Artificial Intelligence components**
- **Performance assessment for real-time applications**

PROGRAM PLAN

- Develop Initial Certification Framework
    - Funded by CIM central funds
    - Contractor - RTI
    - Schedule: May 92 - Dec 92
    - Deliverables - Technical Report
        - available tools/techniques
        - approaches for information storage
        - certification framework
        - plan for application of the certification process
        - plan for cost/benefit analysis
        - plan for incentives
- Apply and Validate Certification Framework
    - Funded by RL 6.2 funds
    - Schedule Jul 93 - Jul 96
    - Deliverables -Technical Reports
        - Revised Certification Framework
        - Results of application of certification process
        - Results of cost/benefit analysis

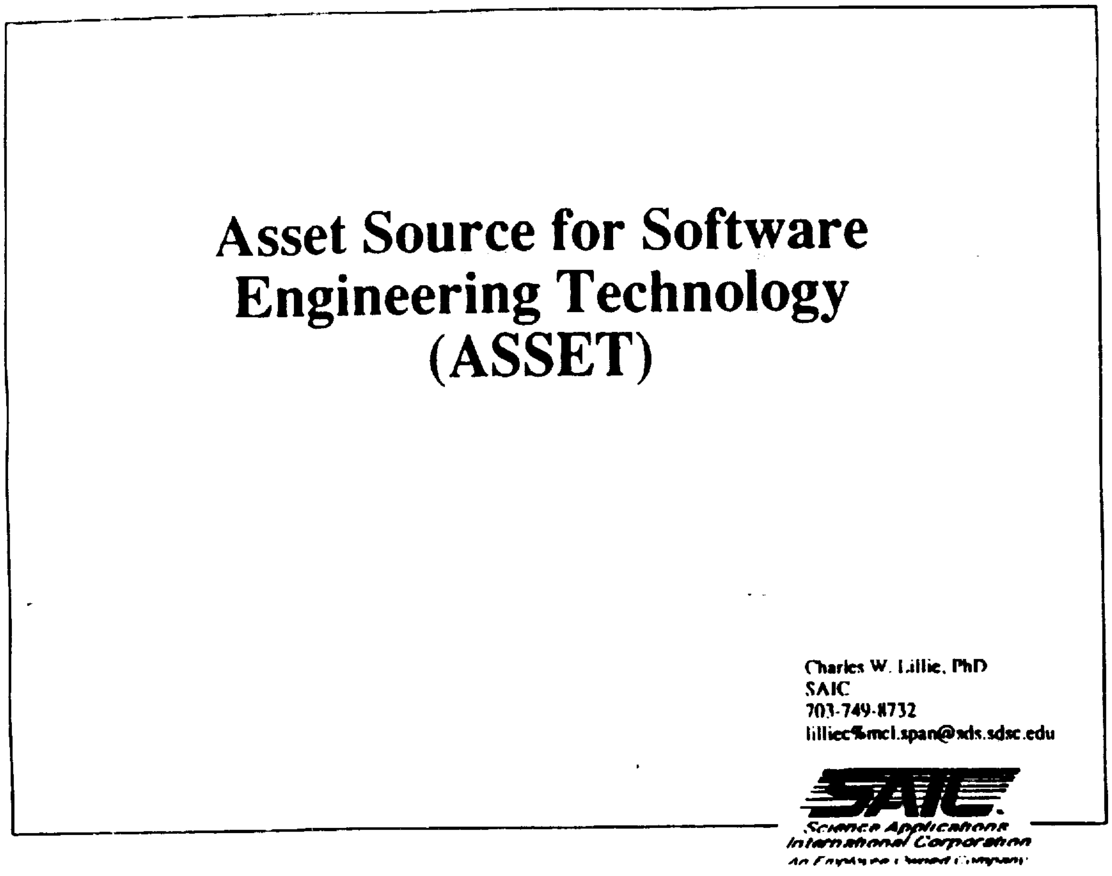
SUMMARY

<u>CERTIFICATION PROCESS & TOOLS</u>

- PROVIDES MEASURE OF CONFIDENCE IN REUSABLE COMPONENT

- PROVIDES SCALE & PERFORMANCE DATA IF REQUIRED

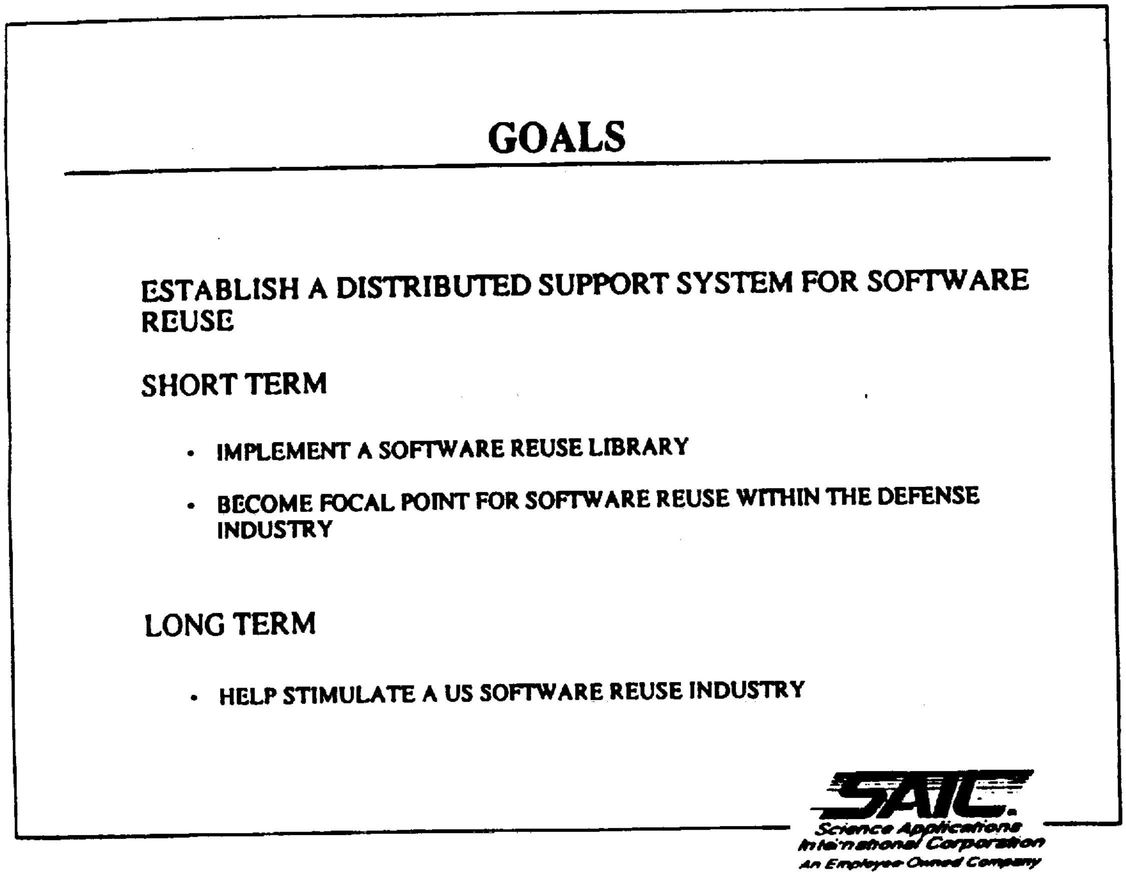
- SOUND BASIS FOR BUILD/BUY DECISIONS



Asset Source for Software Engineering Technology (ASSET)

Charles W. Lillie, PhD
SAIC
703-749-8732
lilliec%mcl.span@xds.sdsc.edu

Science Applications International Corporation
An Employee Owned Company



GOALS

ESTABLISH A DISTRIBUTED SUPPORT SYSTEM FOR SOFTWARE REUSE

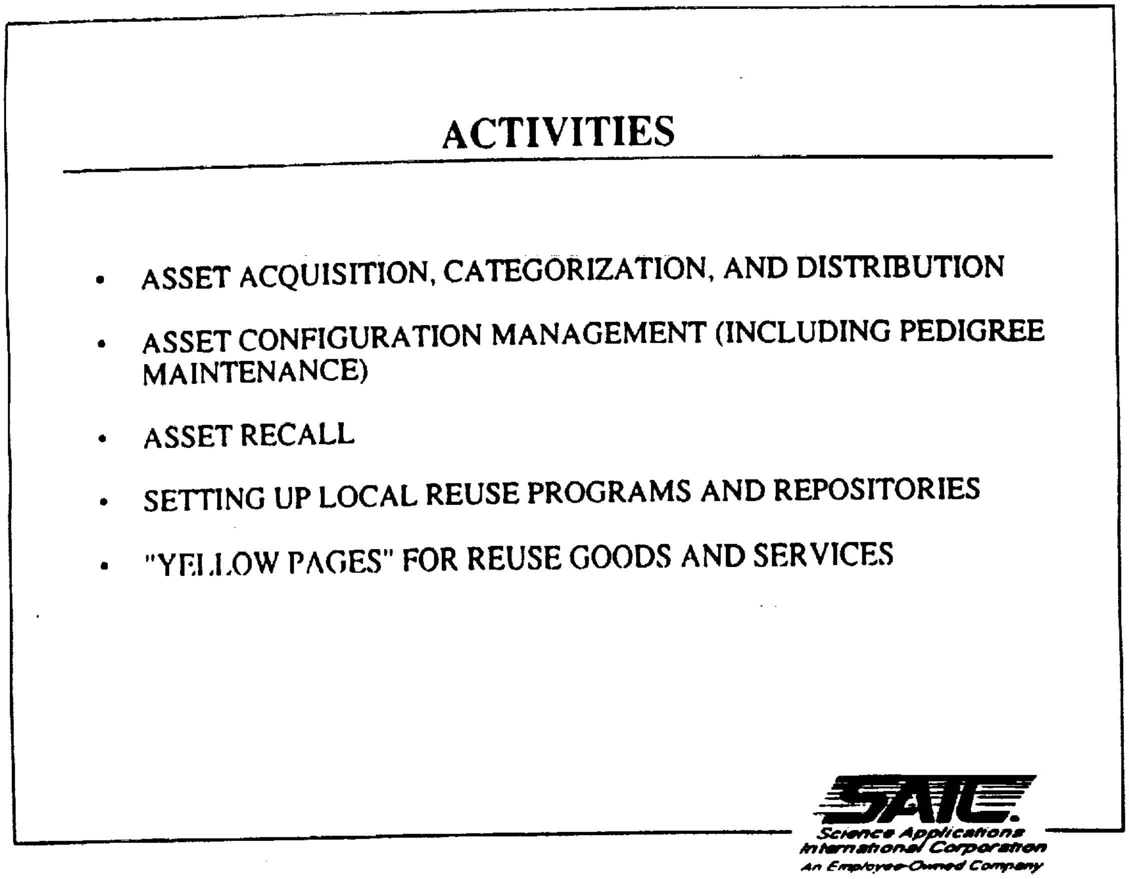
SHORT TERM

- IMPLEMENT A SOFTWARE REUSE LIBRARY
- BECOME FOCAL POINT FOR SOFTWARE REUSE WITHIN THE DEFENSE INDUSTRY

LONG TERM

- HELP STIMULATE A US SOFTWARE REUSE INDUSTRY

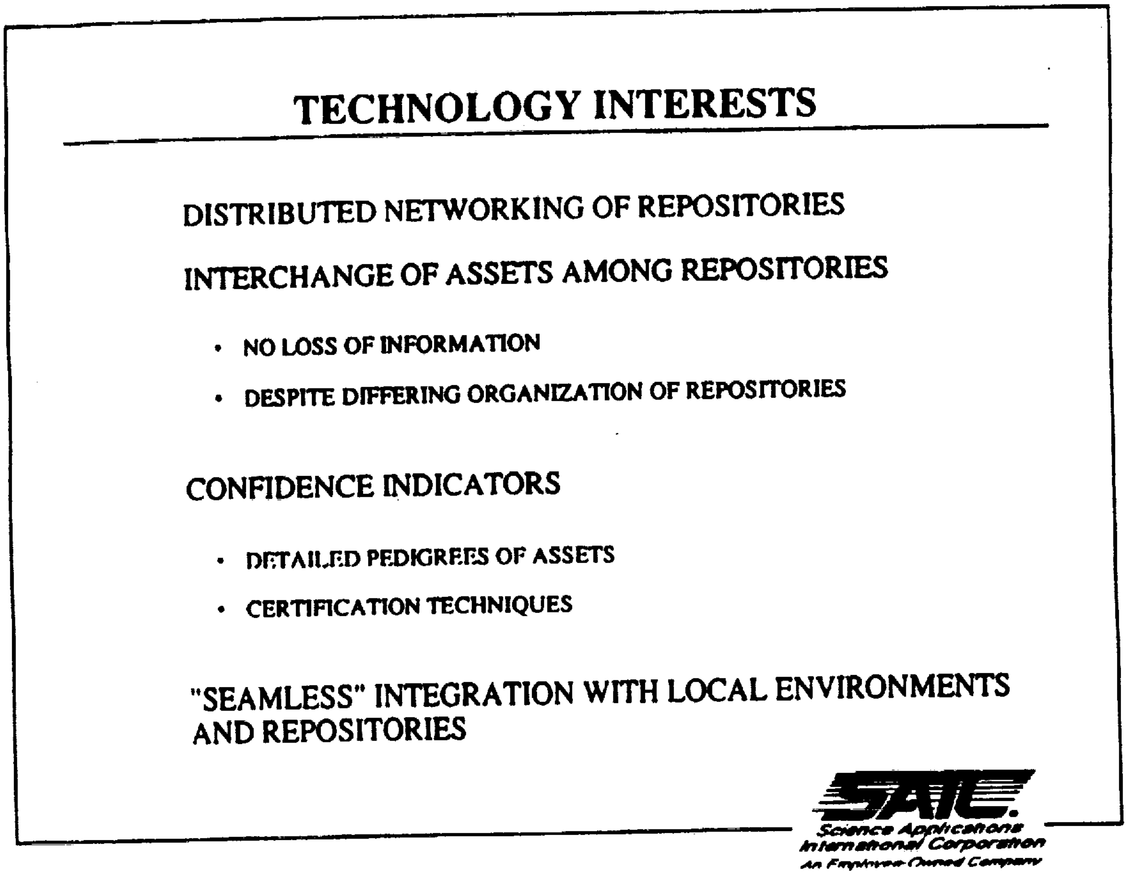
Science Applications International Corporation
An Employee Owned Company



ACTIVITIES

- ASSET ACQUISITION, CATEGORIZATION, AND DISTRIBUTION
- ASSET CONFIGURATION MANAGEMENT (INCLUDING PEDIGREE MAINTENANCE)
- ASSET RECALL
- SETTING UP LOCAL REUSE PROGRAMS AND REPOSITORIES
- "YELLOW PAGES" FOR REUSE GOODS AND SERVICES

SAIC.
Science Applications
International Corporation
An Employee-Owned Company



TECHNOLOGY INTERESTS

DISTRIBUTED NETWORKING OF REPOSITORIES

INTERCHANGE OF ASSETS AMONG REPOSITORIES

- NO LOSS OF INFORMATION
- DESPITE DIFFERING ORGANIZATION OF REPOSITORIES

CONFIDENCE INDICATORS

- DETAILED PEDIGREES OF ASSETS
- CERTIFICATION TECHNIQUES

"SEAMLESS" INTEGRATION WITH LOCAL ENVIRONMENTS AND REPOSITORIES

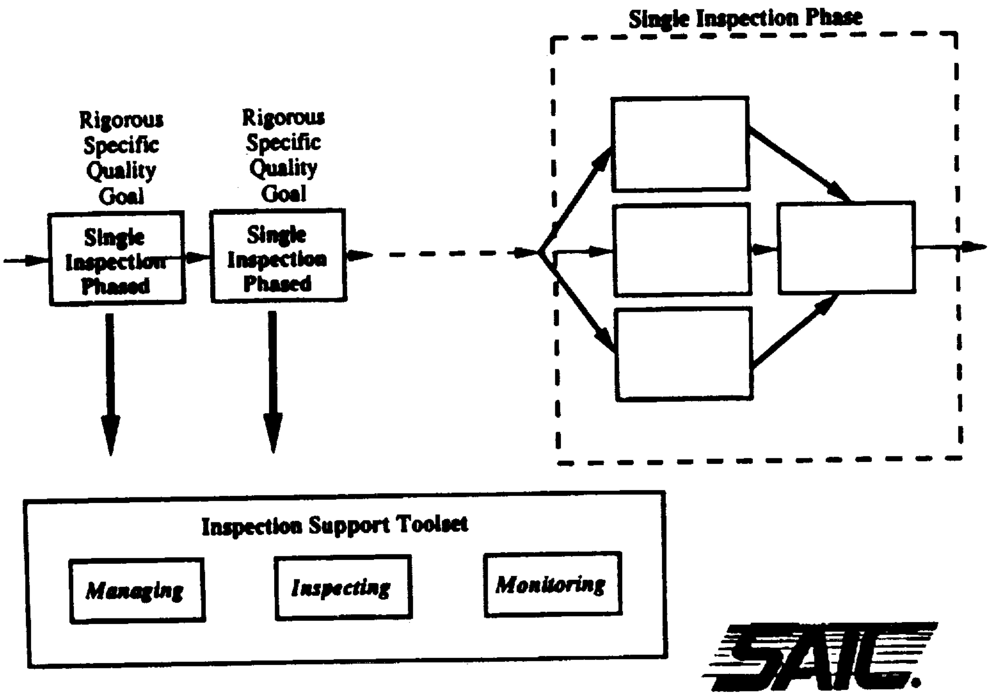
SAIC.
Science Applications
International Corporation
An Employee-Owned Company

Asset Evaluation

Evaluation Level	Description
Documented:	Offeror attests that information requirements are met.
Audited:	Librarian attests that information requirements are met and library issues are addressed.
Validated:	Librarian has examined the software engineering asset and found no errors or inconsistences.
Certified:	Librarian performed independent repeatable evaluation relative to published protocol.

SAIC
Science Applications
International Corporation
An Employee-Owned Company

Phased Inspections



SAIC
Science Applications
International Corporation
An Employee-Owned Company

NTSC Reuse Initiative

- Naval Training System Center
- Adaptation of STARS Technology
- Reuse Library Development
- Flight Simulation Domain Analysis
- Assist in Asset Moderization
- Develop Reuse Software Assessment Tool

SAIC
Science Applications
International Corporation
An Employee-Owned Company

ASSET Business Plan

<u>Market Analysis</u>

Assess current understanding of software reuse technologies, benefits, and requirements within the organizations surveyed, and their commitment to integrating software reuse into their software development process.

<u>Business Analysis</u>

Analyze business models to determine the best approach to manage a software reuse library.

<u>Business Plan</u>

Use business and market analyses to describe the transition from government funding to self-sustaining operations.

SAIC
Science Applications
International Corporation
An Employee-Owned Company

ASSET LONG RANGE PLANNING INFRASTRUCTURE

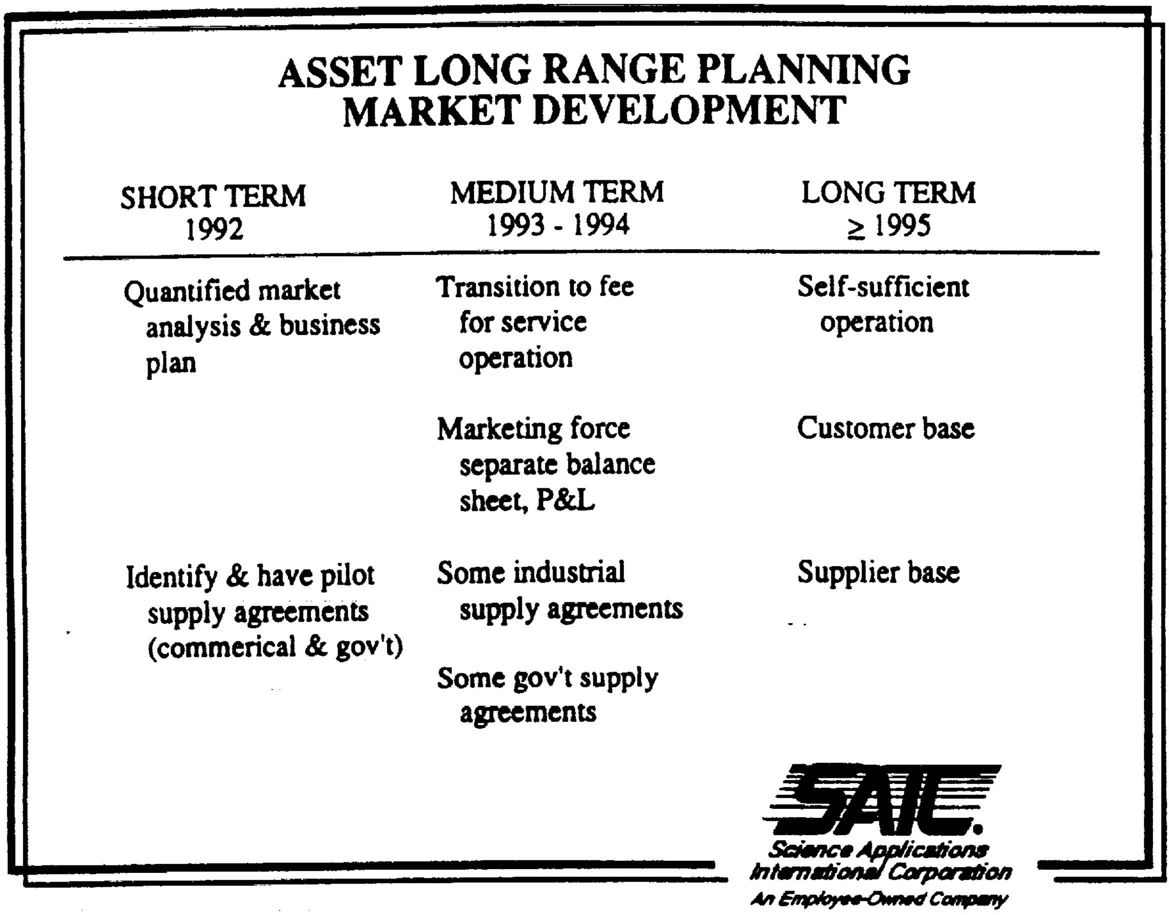
SHORT TERM 1992	MEDIUM TERM 1993 - 1994	LONG TERM ≥ 1995
Implement prelim yellow pages		
	Implement RIG yellow pages	
Install advance library mech.		
Experimental interconnection (CARDS, AdaNET)	Interconnect multi-library	Interoperability
Local Security	Network Security	Interoperability security
Survey Existing legal Work	Formulate basis for industry	
Survey electronic commerce	Formulate basis	

SAIC.
Science Applications International Corporation
An Employee-Owned Company

ASSET LONG RANGE PLANNING PRODUCTS & SERVICES

SHORT TERM 1992	MEDIUM TERM 1993 - 1994	LONG TERM ≥ 1995
STARS CDRLs STARS BB STARS NG STARS Products Other	Program Specific Products & Services Consulting Services	
	Set up local libraries Cross domain components Standards & bindings Reuse technology tools	
		Reuse Library Services

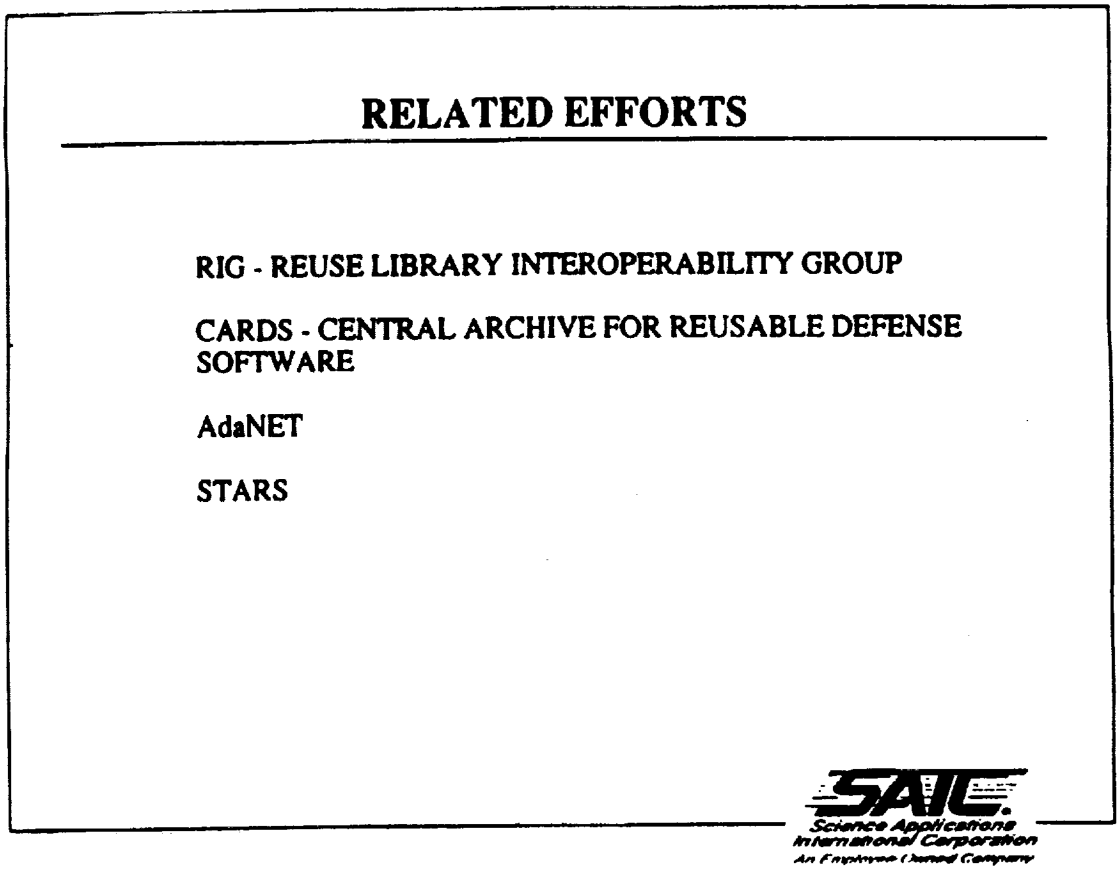
SAIC.
Science Applications International Corporation
An Employee-Owned Company



ASSET LONG RANGE PLANNING MARKET DEVELOPMENT

SHORT TERM 1992	MEDIUM TERM 1993 - 1994	LONG TERM ≥ 1995
Quantified market analysis & business plan	Transition to fee for service operation	Self-sufficient operation
	Marketing force separate balance sheet, P&L	Customer base
Identify & have pilot supply agreements (commerical & gov't)	Some industrial supply agreements	Supplier base
	Some gov't supply agreements	

SAIC
Science Applications International Corporation
An Employee-Owned Company



RELATED EFFORTS

RIG - REUSE LIBRARY INTEROPERABILITY GROUP

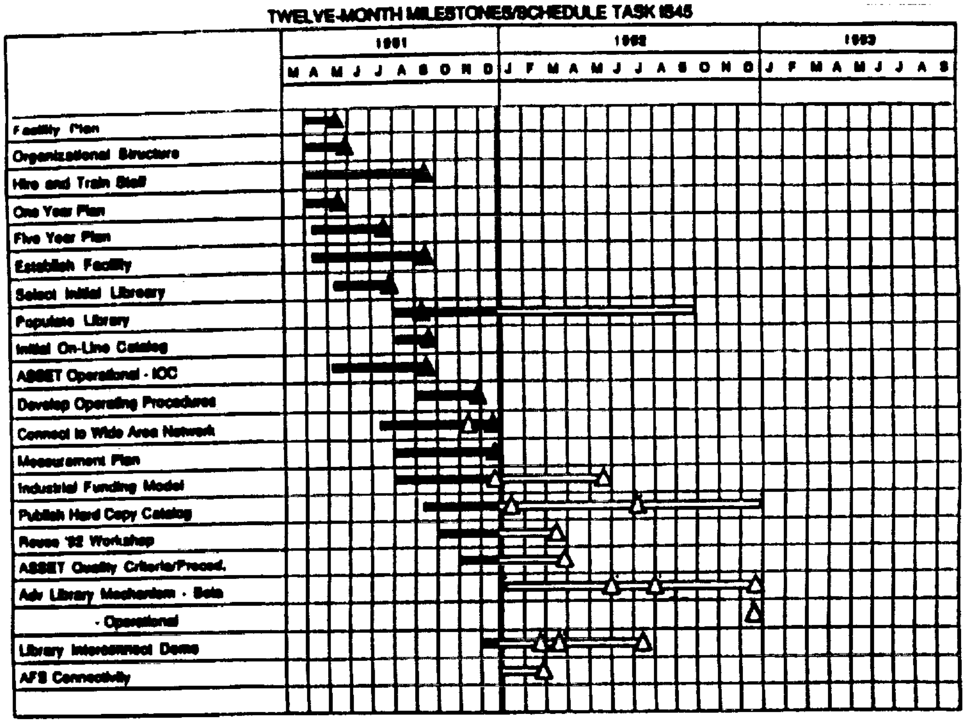
CARDS - CENTRAL ARCHIVE FOR REUSABLE DEFENSE SOFTWARE

AdaNET

STARS

SAIC
Science Applications International Corporation
An Employee-Owned Company

TWELVE-MONTH MILESTONES/SCHEDULE TASK 1545



Science Applications International Corporation
An Employee-Owned Company

Science Applications International Corporation

Re-Engineering With Reuse

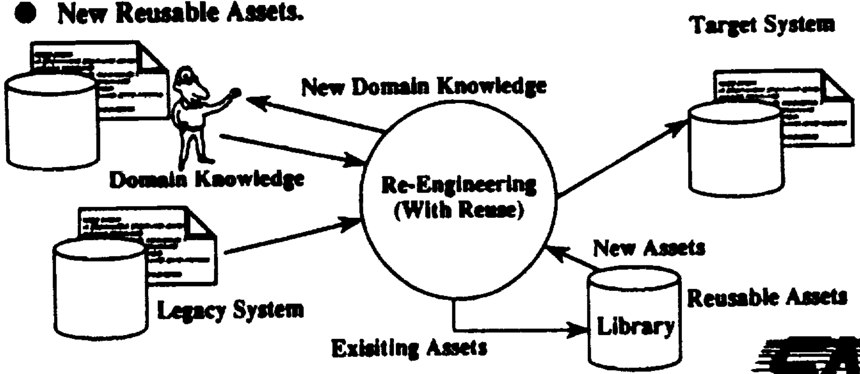
DEFINITION

A process of software analysis and development that takes as input:

- **Software artifacts from a Legacy system,**
- **Domain Knowledge (Vocabulary, Taxonomies, Models, Standards)**
- **Reuse Library**
- **New Requirements (optional)**

For the purpose of producing as output:

- **Target System of higher quality,**
- **Updated Domain Knowledge,**
- **New Reusable Assets.**

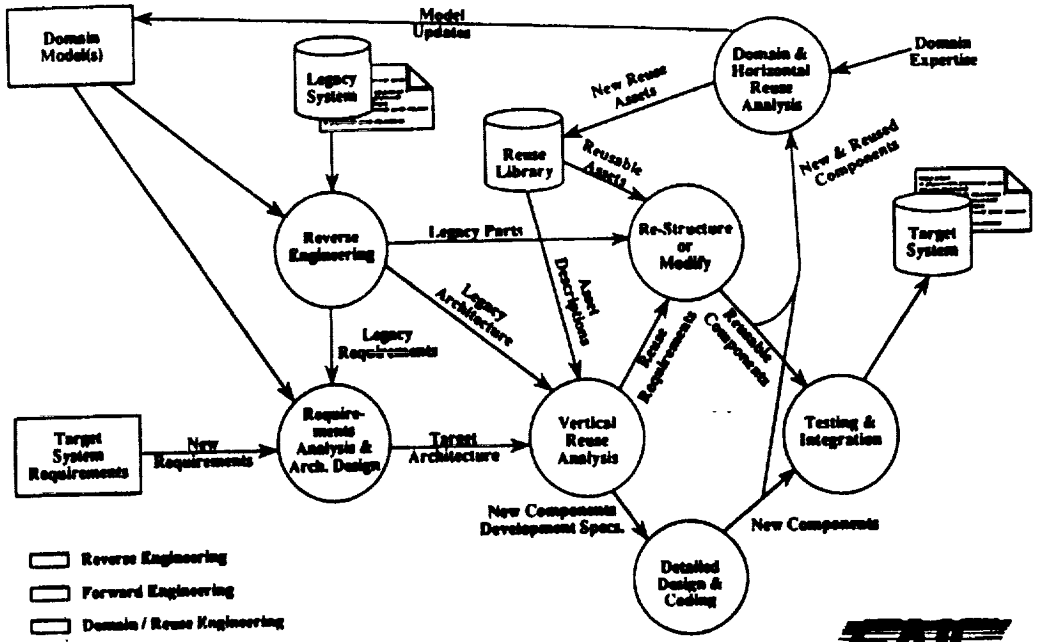


An Employee-Owned Company

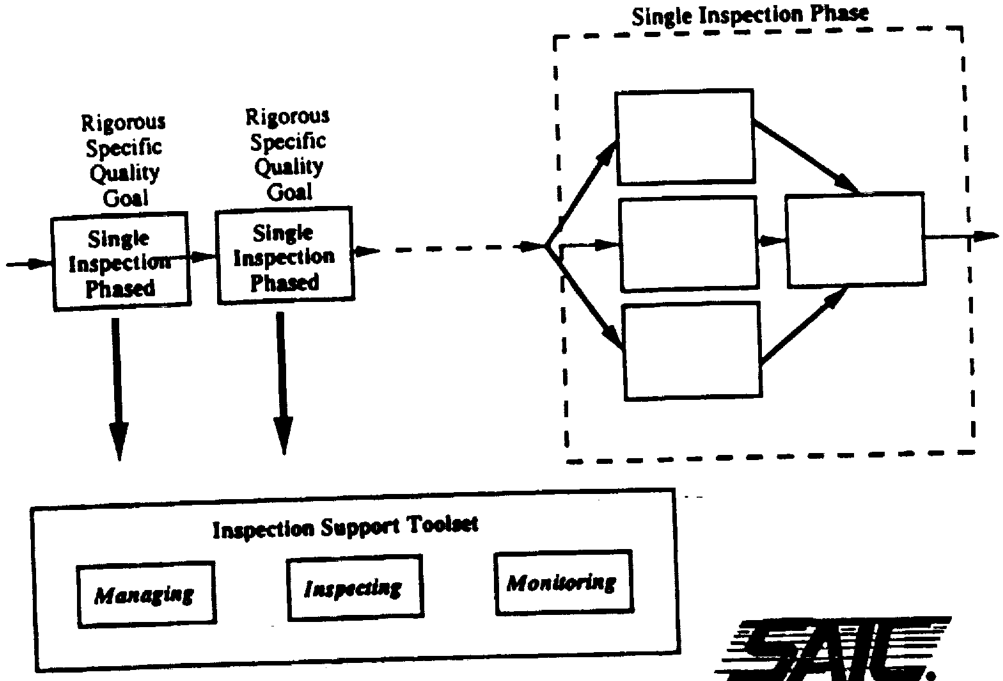
Re-Engineering

PROCESS *(Continued)*

SAIC's Domain-sensitive, reuse-oriented approach to Re-engineering efficiently produces modernized (target) systems from existing (Legacy) systems with by-products consisting of reusable assets and persistent domain knowledge.



Phased Inspections



Single Inspection Phase

Rigorous Specific Quality Goal

Single Inspection Phased

Rigorous Specific Quality Goal

Single Inspection Phased

Inspection Support Toolset

Managing

Inspecting

Monitoring

SAIC. Science Applications International Corporation An Employee-Owned Company

NTSC Reuse Initiative

- Naval Training System Center
- Adaptation of STARS Technology
- Reuse Library Development
- Flight Simulation Domain Analysis
- Assist in Asset Moderization
- Develop Reuse Software Assessment Tool

SAIC. Science Applications International Corporation An Employee-Owned Company

Science Applications International Corporation

Re-Engineering With Reuse

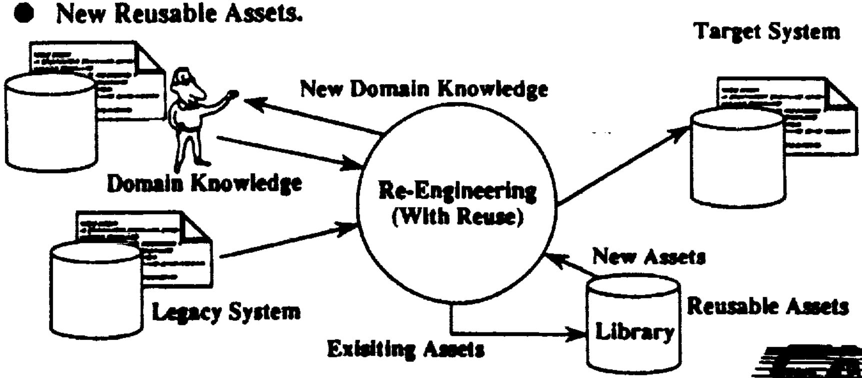
DEFINITION

A process of software analysis and development that takes as input:

- Software artifacts from a Legacy system,
- Domain Knowledge (Vocabulary, Taxonomies, Models, Standards)
- Reuse Library
- New Requirements (optional)

For the purpose of producing as output:

- Target System of higher quality,
- Updated Domain Knowledge,
- New Reusable Assets.



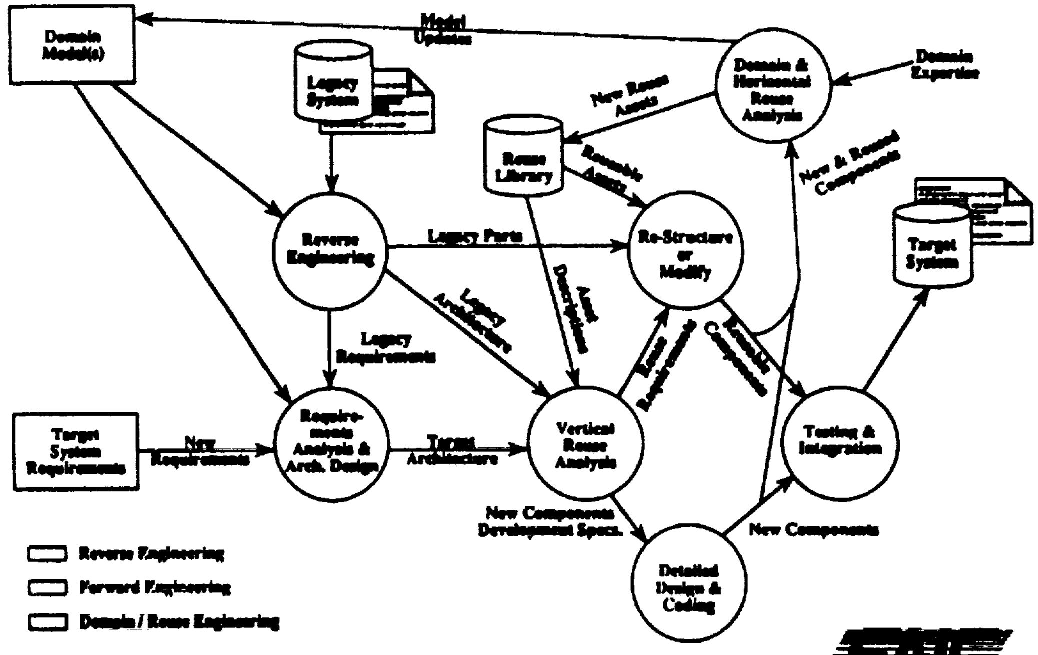
SAIC. An Employee-Owned Company

Science Applications International Corporation

Re-Engineering

PROCESS (*Continued*)

SAIC's Domain-sensitive, reuse-oriented approach to Re-engineering efficiently produces modernized (target) systems from existing (Legacy) systems with by-products consisting of reusable assets and persistent domain knowledge.



SAIC. An Employee-Owned Company

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
January 1993

3. REPORT TYPE AND DATES COVERED
Conference Publication

4. TITLE AND SUBTITLE
A NASA-Wide Approach Toward Cost-Effective, High-Quality,
Software Through Reuse

5. FUNDING NUMBERS
WU 505-64-10-02

6. AUTHOR(S)
Charlotte O. Scheper and Kathryn A. Smith, Editors

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-0001

Research Triangle Institute
Research Triangle Park,
North Carolina 27709

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CP-10115

11. SUPPLEMENTARY NOTES
Charlotte O. Scheper: Research Triangle Institute, Research Triangle Park, NC
Kathryn A. Smith: Langley Research Center, Hampton, VA

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited

Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

NASA Langley Research Center sponsored the second Workshop on NASA Research in Software Reuse on May 5-6, 1992 at the Research Triangle Park, North Carolina. The workshop was hosted by the Resarch Triangle Insitute. Participants came from the three NASA centers, four NASA contractor companies, two research institutes and the Air Force's Rome Laboratory. The purpose of the workshop was to exchange information on software reuse tool development, particularly with respect to tool needs, requirements, and effectiveness. The participants presented the software reuse activities and tools being developed and used by their individual centers and programs. These programs address a wide range of reuse issues. The group also developed a mission and goals for software reuse within NASA. This publication summarizes the presentations and the issues discussed during the workshop.

14. SUBJECT TERMS
software reuse, software development, software upositories, software libraries

15. NUMBER OF PAGES
124

16. PRICE CODE
A06

17. SECURITY CLASSIFICATION OF REPORT
unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
unclassified

20. LIMITATION OF ABSTRACT