TASK: PV03
CDRL: A018
25 October 1993

STARS Conceptual Framework For Reuse Processes (CFRP) Volume I: Definition Version 3.0

Informal Technical Data

STARS-VC-A018/001/00
25 October 1993

TASK: V03
CDRL: A018
October 25, 1993

INFORMAL TECHNICAL REPORT

For

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

STARS Conceptual Framework for Reuse Processes (CFRP)
Volume I: Definition
Version 3.0

STARS-VC-A018/001/00
October 25, 1993

Data Type: Informal Technical Data

CONTRACT NO. F19628-93-C-0130

Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

The Boeing Company,
Defense & Space Group,
P.O. Box 3999, MS 87-37
Seattle, WA 98124-2499

IBM,
Federal Systems Company,
800 N. Frederick Pike,
Gaithersburg, MD 20879

Unisys Corporation,
12010 Sunrise Valley Drive,
Reston, VA 22091

TASK: V03
CDRL: A018
October 25, 1993

Data ID: STARS-VC-A018/001/00

Distribution Statement "A"
per DoD Directive 5230.24
Authorized for public release; Distribution is unlimited.

Developed by: Unisys Corporation

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Scheme (DoD Directive 5230.24) unless otherwise indicated. Sponsored by the U.S. Advanced Research Projects Agency (ARPA) under contract F19628-88-D-0031, the STARS program is supported by the military services, SEI, and MITRE, with the U.S. Air Force as the executive contracting agent.

TASK: V03
CDRL: A018
October 25, 1993

INFORMAL TECHNICAL REPORT
STARS Conceptual Framework for Reuse Processes (CFRP)
Volume I: Definition
Version 3.0

Approvals:

Teri L. Payton 10/28/93

Program Manager *Teri Payton* *Date*

(Signatures on File)

TASK: V03
CDRL: A018
October 25, 1993

INFORMAL TECHNICAL REPORT
STARS Conceptual Framework for Reuse Processes (CFRP)
Volume I: Definition
Version 3.0

Change Record:

Data ID	*Description of Change*	*Date*	*Approval*
STARS-VC-A018/001/00	Version 3.0: CFRP process categories further refined.	25 October 1993	*on file*
STARS-UC-05159/001/00	Version 2.0: CFRP revised to incorporate Reuse Management and Reuse Engineering idioms.	13 November 1992	*on file*
STARS-SC-04040/001/00	Version 1.0: Original Issue	14 February 1992	*on file*

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	25 October 1993	Informal Technical

4. TITLE AND SUBTITLE
STARS Conceptual Framework for Reuse Processes (CFRP)
Volume I: Definition
Version 3.0

5. FUNDING NUMBERS
F19628-93-C-0130

6. AUTHOR(S)
Boeing
IBM
Unisys

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

8. PERFORMING ORGANIZATION REPORT NUMBER
STARS-VC-A018/001/00

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of the Air Force
Headquarters Electronic Systems Center

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
A018

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution "A"

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document, *STARS Conceptual Framework for Reuse Processes (CFRP), Volume I: Definition, Version 3.0*, defines version 3.0 of the STARS CFRP. It supercedes an earlier CFRP definition document, *STARS Reuse Concepts, Volume I - Conceptual Framework for Reuse Processes (CFRP), Version 2.0.* This document is a new version of the earlier document, describing a new version of the CFRP. The document has been retitled to more directly reflect its CFRP subject matter, but the version sequencing has been retained.

This document is Volume I of a two volume set. Its companion volume, *STARS Conceptual Framework for Reuse Processes (CFRP), Volume II: Application, Version 1.0*, provides guidance in how to apply the CFRP.

14. SUBJECT TERMS

15. NUMBER OF PAGES
85

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Contents

List of Figures

List of Tables

Prologue

This document, *STARS Conceptual Framework for Reuse Processes (CFRP), Volume I: Definition, Version 3.0*, defines version 3.0 of the STARS CFRP. It supercedes an earlier CFRP definition document, *STARS Reuse Concepts, Volume I - Conceptual Framework for Reuse Processes (CFRP), Version 2.0*. This document is a new version of the earlier document, describing a new version of the CFRP. The document has been retitled to more directly reflect its CFRP subject matter, but the version sequencing has been retained.

This document is Volume I of a two volume set. Its companion volume, *STARS Conceptual Framework for Reuse Processes (CFRP), Volume II: Application, Version 1.0*, provides guidance in how to apply the CFRP.

Version 3.0 of the CFRP is significantly different from version 2.0 in several ways. The most prominent differences are briefly summarized below:

- The *Domain Selection* process category in the Reuse Planning process family has been renamed *Scoping*. In addition, its meaning has been generalized to encompass definition of the overall technical and organizational scope of a reuse program. This involves selecting the key domains of focus in the program and planning how the domain assets will be applied in selected application product lines. It also involves delineating the reuse program's organizational boundaries and spheres of influence.
- The *Process Observation* and *Process Evaluation* process categories in the Reuse Learning process family have been renamed *Project Observation* and *Project Evaluation*, respectively. This renaming makes these categories more consistent with other Reuse Management categories that focus on the CFRP notion of reuse projects, and it also removes the implication that learning focuses narrowly on processes, rather than processes, products, and infrastructure.
- The *Software Architecture Development* process category in the Asset Creation process family has been renamed *Domain Architecture Development* to better reflect its scope and role in developing domain architecture assets.
- The *Application Generator Development* and *Software Component Development* process categories in the Asset Creation process family have been merged into a single category called *Asset Implementation*. This category encompasses the development of software components and other non-architecture assets such as reusable requirements and test software, and also addresses the development of assets that can generate such application products.
- *Library Usage Support* and *Asset Brokering* process categories have been added to the Asset Management process family. These categories represent important activities in supporting and encouraging the utilization (as well as creation) of assets. These activities may have been seen as implicit in the *Library Operation* category in earlier versions of the CFRP, but are now viewed as important enough to be treated as explicit process categories.
- The *Library Metrics Collection* and *Asset Metrics Collection* process categories have been removed from the Asset Management process family. The CFRP addresses process and product metrics collection in the Plan-Enact-Learn idiom, and specific metrics collection

categories in the Asset Management family were viewed as redundant. However, aspects of the *Asset Metrics Collection* category that involved recording intrinsic asset characteristics for the benefit of asset utilizers have been incorporated into *Asset Cataloging.*

- The *Asset Evolution* and *Library Evolution* process categories have been removed from the Asset Creation and Asset Management process families, respectively. The CFRP addresses process and product evolution implicitly, through the continual application of the Plan-Enact-Learn idiom. Therefore, no explicit process categories to address process and product evolution are necessary.
- The *Asset Requirements Determination* process category in the Asset Utilization process family has been renamed *Asset Criteria Determination* to avoid unnecessary confusion resulting from excessive use of the already overloaded word "requirements". The meaning of the category remains unchanged.
- The *Integration of Assets with Application* process category in the Asset Utilization process family has been renamed *Asset Integration* to shorten and simplify the name. The meaning of the category remains unchanged.

Both CFRP volumes will be revised and re-released periodically to reflect the lessons learned in defining and applying the CFRP and to address comments from reviewers of the documents, both internal and external to STARS.

We thus encourage trial application of the CFRP and solicit reader review and comments as input to future revisions of both volumes. Please submit comments to:

CFRP Comments
c/o Dick Creps
Unisys Government Systems Group
Dept. 7670
12010 Sunrise Valley Drive
Reston, VA 22091

Phone: (703) 620-7100
Fax: (703) 620-7916
E-mail: cfrp@stars.ballston.paramax.com

1 Introduction

This document was produced by the Software Technology for Adaptable, Reliable Systems (STARS) program on behalf of the U.S. Department of Defense (DoD) Advanced Research Projects Agency (ARPA). The document was developed by a STARS working group consisting of members from each of the three STARS prime contractor teams, the MITRE Corporation, and the Hewlett Packard Company (a STARS Technology Transition Affiliate). The DoD Software Engineering Institute (SEI) also contributed significantly to the document during the early stages of its development.

1.1 Purpose

The principal purpose of this document is to define a conceptual framework that describes reuse in terms of the processes involved. This STARS Conceptual Framework for Reuse Processes (CFRP) is intended to:

- articulate, and promote understanding of, STARS reuse concepts,
- identify reuse processes that are candidates for detailed definition,
- provide techniques for modeling interrelationships among reuse processes, and
- identify organizational and management issues associated with reuse and provide a framework for managing change in that context.

In addition, this document elaborates on the STARS vision and mission with respect to reuse and provides a context for understanding STARS reuse products and plans.

1.2 Applicability and Scope

The CFRP is intended to be generic with respect to domains, organizations, economic sectors, methodologies, and technologies. Furthermore, even though this document focuses discussion of the CFRP by describing it primarily in the context of software engineering, CFRP concepts should generally be applicable to reuse in any information-intensive context, such as technical documentation, general information retrieval, and business, scientific, or personal information management.

The CFRP is a reuse process framework; its scope is limited to identifying the processes involved in reuse and describing at a high level how those processes operate and interact. In so doing, it offers an indication of how some reuse processes can be integrated with other processes and how the transition to reuse can be managed from a process perspective. The CFRP does **not** prescribe how reuse should be implemented in any particular context, nor does it include processes that do not directly contribute to reuse. It thus should not be interpreted as providing prescriptive process definitions or imposing a particular life cycle model. However, in separate efforts, STARS is developing detailed reuse process definitions and life cycle models that are consistent with the CFRP.

In general, legal, business, and acquisition aspects of reuse are outside the scope of this document.

1.3 Audience

This document is targeted to readers having one or more of the following roles in their organizations:

- *Program/Project Planner* – Responsible for planning the objectives, strategy, processes, infrastructure, and resources for software engineering programs or projects. Interested in incorporating domain-specific reuse into those programs/projects.
- *Process Engineer* – Responsible for defining, instantiating, tailoring, installing, monitoring, administering, and evolving software engineering process models. Interested in defining reuse processes or integrating them with overall life cycle process models.
- *Reuse Advocate* – Responsible for keeping abreast of reuse concepts, technology, and trends and promoting the establishment/improvement of reuse capabilities and practices within an organization. Interested in understanding how new concepts and technology can be applied to accelerate reuse adoption.

Different portions of this document may appeal most strongly to one segment of the audience or another. However, the entire document should be of some interest to all readers, because all the audience segments are interdependent and each can benefit from the variety of perspectives presented in the document.

1.4 Relationship to Other Work

This document is Volume I of a two volume set. The other volume, *STARS Conceptual Framework for Reuse Processes (CFRP), Volume II: Application, Version 1.0* [Sof93c], directly supplements Volume I by providing initial guidance in how the CFRP can be applied. It is recommended that Volume II be read in addition to Volume I to gain increased understanding of the CFRP and its applicability. (Note that throughout the remainder of this document, Volume I will be referred to informally as the "CFRP Definition document", and Volume II will be referred to as the "CFRP Application document".)

The CFRP is derived from and elaborates on the STARS vision with respect to reuse. It reflects and abstracts the experiences of software reuse efforts both within and outside of STARS. It also draws on organizational and management concepts and STARS process concepts. The CFRP has been reviewed by individuals in government, industry, and academia, and by working groups at the 1991 and 1992 Annual Workshops on Software Reuse (WISRs) and the 1991 Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). The CFRP reflects a consensus view among its varied authors, based on their own experience, feedback from reviewers, and lessons learned through early CFRP application.

The CFRP thus provides a conceptual foundation, framework, and set of high level requirements for the reuse technology products (processes and supporting tools) needed to accomplish the STARS reuse mission. STARS has already produced specific reuse process definitions consistent with the CFRP, such as the Reuse-Oriented Software Evolution (ROSE) Process Model [Sof93d], the Organizational Domain Modeling (ODM) Process Model [Sof93a], the STARS Reuse Library Process

Model [Sof91c], the STARS Composite Process Model [Sof91b], and the ASSET Criteria and Implementation Procedures for Evaluation of Reusable Software Engineering Assets [Ass92b]. The STARS asset library mechanisms provide automated support for many of the process categories identified in the CFRP.

The CFRP also provides a framework or basis for some of the products that are needed to transition reuse-related technology into use. For example, the STARS Reuse Strategy Model (RSM) [Sof93b], a mechanism to help projects assess their current reuse capabilities and develop strategies for improving them, draws heavily on the CFRP as a basis for characterizing reuse capabilities.

However, for an organization undertaking a reuse program, the CFRP and the associated technology cited above must be augmented with additional information to motivate and promote the adoption of reuse. Such information can include case studies, rationale, lessons learned, economic models and data, and specific guidance about how to implement a reuse program. Products providing these kinds of information may be consistent with the CFRP and even use it as an organizing principle, but extend beyond the scope of the CFRP itself. Examples of such products are the Reuse Adoption Guidebook [Vir92a] developed by the Virginia Center of Excellence for Reuse and Technology Transfer (VCOE), and the Direction Level Handbook [Cen93a], Acquisition Handbook [Cen92], and Franchise Plan [Cen93b] developed by the DoD Central Archive for Reusable Defense Software (CARDS) program.

1.5 Document Organization

This volume is organized as follows:

- Section 1 (this section) provides introductory material that defines the purpose, scope, and audience of the document and establishes its relationship to other reuse-related work.
- Section 2 describes the context in which the CFRP was developed and the factors that motivated its development.
- Section 3 describes the CFRP in detail.
- Section 4 summarizes the CFRP in terms of the key themes it embodies.
- Appendix A describes a set of CFRP process modeling techniques enabling construction of a wide variety of CFRP process configurations.
- The Glossary defines the key terms used in this volume.
- The References section provides bibliographic entries for all documents referenced in this volume.

2 Context and Motivation

A number of factors have influenced and motivated the work that STARS is doing in the area of software reuse. This section discusses these factors from two viewpoints. The first provides a brief assessment of current reuse practice and presents an overview of recent reuse-related technical and management trends that have influenced STARS work and are already impacting reuse practice to varying degrees. The second viewpoint provides insight into the STARS vision, mission, and strategy as seen specifically from a reuse perspective. The section closes by describing a set of objectives for this document, derived from these motivating factors.

2.1 Reuse Practices and Trends

As has been frequently documented (e.g., in [U.S89]), the practice of software development in the United States has become increasingly plagued by high cost, poor productivity, and poor or inconsistent quality. One of the reasons for this problem is that many software development organizations continue to recreate the same or similar systems over and over again, rather than treating their previously-developed systems and previous experiences as assets to be captured, nurtured, and evolved so that they can contribute directly to future development activities. That is, these organizations are practicing little or no *software reuse.*

In particular, the development of software systems for the U.S. Department of Defense (DoD) historically has not been very reuse-oriented. There are many reasons, all valid from one perspective or another, for software reuse not to have become standard practice within DoD and its contracting organizations. These reasons are wide-ranging, involving many different kinds of issues: technical, cultural, contractual, organizational, economic, legal, political, and so on. When reuse does occur in the DoD context, it is typically through individual initiative, rather than in response to a deliberate plan and well defined processes. Such reuse is likely to involve code and perhaps design information rather than complete sets of assets covering the entire life cycle. Furthermore, reuse is usually isolated within a single organization – typically among a small number of similar projects – and the information that is reused in these circumstances often requires significant modification because it was not designed for reuse.

However, this situation appears to be steadily changing. The barriers that have inhibited reuse are gradually diminishing and, due to a downturning and more competitive economic climate, it is becoming increasingly critical for organizations to overcome those barriers. The result has been a recent substantial increase in the number of industry and government initiatives focusing on establishing reuse programs. For example, the DoD has developed a Reuse Vision and Strategy [DoD92] and has established a Reuse Executive Steering Committee and two working groups addressing reuse technical and management issues. The DoD has additionally established a Software Reuse Initiative, which is a federation of three reuse-related programs: STARS, the Air Force Central Archive for Reusable Defense Software (CARDS) program, and the Defense Information Systems Agency (DISA) Joint Interoperability Engineering Organization (JEIO) Center for Information Management (CIM) Software Reuse Program (SRP). The DISA/JEIO/CIM SRP, in particular, has undertaken a number of activities to directly support the realization of the DoD Reuse Vision and Strategy. In addition, the Army, Navy, and Air Force have each issued plans for accelerating adoption of reuse within the individual services. Prior to these developments, DoD organizations

such as the Joint Integrated Avionics Working Group (JIAWG), the Strategic Defense Initiative Office (SDIO), and the Army Communications and Electronics Command (CECOM) established reuse initiatives. The Air Force has established, in tandem, the CARDS program to develop a blueprint for domain-specific, architecture-based reuse, and the Portable, Reusable, Integrated Software Modules (PRISM) program to develop a generic command center prototyping capability based on CARDS facilities and processes. Individual companies have begun to formalize reuse and to develop reusable components for competitive advantage or commercial purposes (e.g., TRW's Universal Network Architecture Services, EVB's GRACE components).

This increased reuse activity has built upon, and often contributed to, a number of technical and management advances that form the theoretical and practical foundations for reuse. Foremost among these advances is the emergence of domain analysis and domain engineering as distinct fields of study. These disciplines focus on methods and tools for analyzing families of systems (*domains*) and capturing, organizing, and evolving information about the domains (encoded in *domain models, architectures, generators,* and *components*) so that it can be exploited to support reuse-based system development and evolution. This technology area is still somewhat immature, but current work is paving the way for more generalized techniques to support domain-specific reuse (see [PDA91] for an overview of this technology area and [Sof93a, Sof91c, DIS93, Vir92b, KCH+90, IHD+90, PW92, Sha91, Sha89, Bai89a, MCP+88] for further information). The DoD Advanced Research Projects Agency (ARPA) is explicitly focusing on domain engineering issues through its Domain Specific Software Architecture (DSSA) program [MG92]. In addition, the software maintenance and reengineering communities are shifting increasingly towards a view of their disciplines that is more directly compatible with general domain analysis and domain engineering principles [Bas90, Big89, Nav93].

Another strong influence on modern reuse practices is the work underway to address automated support for domain-specific reuse, including reuse library technology. Theoretical aspects of reuse libraries have been explored significantly (e.g., [PDF87, FG90]), and a number of processes and tools have been developed to support them (e.g., [Reu90, Sof91c, SWT89]). As a result, several major efforts are underway to establish operational domain-specific reuse libraries (e.g., the STARS Asset Source for Software Engineering Technology (ASSET) [Ass92a], the CARDS Generic Command Center library [Cen93c], the DISA/JEIO/CIM Defense Software Repository System (DSRS), and the Reusable Ada Avionics Software Packages (RAASP) library). There has been other important work focusing on reuse automation, as well, ranging from application generation technology to reuse-based software environments (e.g., [Cle88, Bai89b, Sca92]).

In addition, there have been a number of recent advances in disciplines that are not generally recognized as being reuse-oriented, but that focus on many of the same goals as reuse and are beginning to influence reuse practice, particularly from a management perspective. Among these influences are concepts from the software process modeling and management community (e.g., [CKO92]), Continuous Process Improvement (CPI) and Total Quality management (TQM) principles (e.g., [Dem86]), and advances in organizational and learning theory, including the concept of the Learning Organization (e.g., [Sen90]).

2.2 STARS Reuse Vision, Mission, and Strategy

As indicated by the reuse-related activity cited above, a trend towards reuse-based software engineering is already underway, both within DoD and its contracting organizations and in general industry. This is evidenced clearly in the DoD Reuse Vision defined in the DoD Reuse Vision and Strategy document [DoD92]:

> The DoD vision for reuse is to drive the DoD software community from its current "re-invent the software" cycle to a process-driven, domain-specific, architecture-centric, library-based way of constructing software.

To help accelerate the trend towards reuse-based software engineering consistent with this vision, ARPA is sponsoring the STARS program. To focus its efforts, STARS has defined its own *vision* describing the desired future state of DoD software engineering practice, a *mission* defining the STARS role in realizing the vision, and a *strategy* defining specific activities that STARS will undertake to pursue the mission. The STARS vision and mission, taken together, are highly consistent with the DoD Reuse Vision.

The STARS vision is:

> Within 15 to 20 years, the DoD will have institutionalized a disciplined architectural and engineering-based approach to the development and evolution of software-intensive systems.

The STARS mission is:

> To provide DoD the technological, management, and transitional basis to influence and enable a paradigm shift to a process-driven, domain-specific reuse-based approach to software-intensive systems development and evolution.

The paradigm described above is a STARS interpretation of a new software engineering paradigm that has been dubbed *megaprogramming* within ARPA. In the paragraphs below, this paradigm is further interpreted, specifically from a reuse perspective, to construct a STARS "reuse vision" that defines what *reuse-based* means and how the process-driven and domain-specific aspects of the paradigm constrain that meaning.

Being *reuse-based* means that the standard approach to software-intensive system development and evolution is to derive new and modified systems principally from existing assets rather than to create the systems anew. Reusable assets are thus a central concept of the reuse vision, and they imply a need for processes to create such assets, manage them (typically in some form of asset library), and utilize them to produce new systems. Assets include not only the software code components most commonly associated with reuse, but also other kinds of information such as:

- Reusable forms of other software products; e.g., requirements specifications, architectures, designs, test procedures

- Application domain knowledge; e.g., models, data dictionaries, algorithms
- Process definitions; e.g., for managing asset libraries, for developing application systems
- Rationale; e.g., for the inclusion of features, services, objects, and/or algorithms in a system; for the selection of one architecture or design over another.

Being *domain-specific* means that the reusable assets, the development processes, and the supporting technology are appropriate to (perhaps tailored for) the application domain for which the software is being developed. STARS is focusing on reuse within application domains because we believe that is where the greatest reuse impact will be achieved. Application domains are generally considered to be broad in scope, for example C^3I, and to comprise subdomains. These subdomains may be unique to the application domain or common across several domains. A domain-specific software architecture with standard interfaces is central to domain-specific reuse, in that it provides a framework for creating assets and constructing systems within a domain. The effectiveness of domain-specific assets depends on a number of factors, including the maturity of the application domain and the investment applied to create the assets. As a domain matures, it generally becomes more stable and better understood, thus increasing the likelihood that assets will be reusable in a large number of systems. However, even in mature domains, asset reusability and quality will be maximized only if suitable investment has been applied to identify key reuse opportunities and develop assets that best exploit those opportunities.

Being *process-driven* means that software engineering is done in accordance with well defined, repeatable processes that are subject to continuous measurement and improvement and enforced through management policies. Computer-assisted software engineering environments provide, at a minimum, automated definition of the processes and guidance in applying them, and in some cases may provide a significant degree of automated process enactment and enforcement.

The high-level STARS strategy for effecting the megaprogramming paradigm shift is to:

- Demonstrate the benefits of domain-specific reuse in familiar DoD application contexts,
- Support the transition from the current paradigm in such a way as to reduce risks in DoD's evolution to domain-specific reuse-based software engineering, and
- Ensure that basic reuse support capabilities, both processes and technologies, are available and validated for use.

The STARS program is building on the results of previous reuse efforts, working together with ongoing government and industry programs, and undertaking additional initiatives to implement this strategy and help make realization of the vision and mission possible.

2.3 Document Objectives

Successful pursuit of STARS objectives requires a common understanding of what process-driven, domain-specific reuse-based software engineering (i.e., the megaprogramming paradigm) means. Thus, one key motive behind this document is to develop a common, consensus view of the reuse

aspects of the paradigm among STARS personnel and within a broader community. This implies that the document should provide an expanded and unified elaboration of the STARS reuse vision and its implications, in a way that properly reflects and integrates relevant ongoing work from various reuse-related disciplines.

These are ambitious objectives, and there are numerous possible approaches to the problem. Other work has strived to explore the conceptual underpinnings of reuse (e.g., [BR87]), establish broadly-scoped reuse process models (e.g., [BCC92]), examine reuse from a management perspective (e.g., [BB91]), or establish reuse adoption and maturity models (e.g., [Vir92a]). A key ingredient that is missing from such work is a unifying conceptual model that provides a common context for understanding the technical and management aspects of reuse in terms of the *processes* that are involved. Such a *conceptual process framework* could provide a common basis for establishing, comparing, contrasting, measuring, and improving reuse-related processes among varied organizations. A framework that satisfies these needs should meet many if not all of the following goals:

- Establish a common reuse terminology,
- Emphasize consideration of reuse issues from a process perspective,
- Define a classification scheme for reuse processes,
- Formalize and normalize essential reuse process characteristics and interrelationships,
- Encompass both technical and management aspects of reuse,
- Emphasize the importance of formulating reuse strategies in the context of specific application domains and product lines,
- Emphasize domain-specific architecture-based approaches to reuse,
- Establish measurement, learning, and change as integral to reuse,
- Emphasize the need for investment in infrastructure,
- Establish the brokerage role as fundamental to reuse,
- Accommodate small-scale as well as large-scale software engineering efforts,
- Accommodate complex, reuse-driven interactions among organizations,
- Provide flexible mechanisms to support modeling and configuration of reuse processes,
- Be generic with respect to domains, technologies, organizational structures, and economic sectors.

To address these objectives, STARS has developed the Conceptual Framework for Reuse Processes (CFRP) described throughout the remainder of this document.

3 CFRP Description

The STARS CFRP defines a context for considering reuse-related software development processes, how they interrelate, and how they can be composed and integrated with each other and with non-reuse-related processes to form reuse-oriented life-cycle process models that are tailorable to organization needs.

The CFRP consists of dual, interconnected "process idioms" called *Reuse Management* and *Reuse Engineering*, as depicted in their most basic form in Figure 1. These idioms describe distinctive patterns of activity that are inherent to the organizational and management aspects and the product engineering aspects of reuse, respectively. The CFRP process idioms are further decomposed into "process families" (shown in Figure 1) and these in turn are decomposed into "process categories". Table 1 shows the full decomposition structure of the CFRP.

Figure 1 also shows the major inputs to and outputs from the CFRP. Inputs to the CFRP are:

- **Market Forces** – new market trends, competitive developments, new technologies, emerging standards, and other factors that impact perception of marketplace needs
- **Assets** – existing reusable units of information relevant to a domain or set of domains; e.g., domain models and architectures, software components, application generators, test cases, processes
- **Software Systems** – systems in domains of interest that can impart legacy knowledge about the domains and feed domain analysis or reengineering efforts to produce domain assets or new application systems
- **Domain Knowledge** – information about domains that can be imparted in a variety of ways, other than via the legacy systems
- **Technology** – technological capabilities that can contribute to the reuse infrastructure within an organization and can be applied to establish or automate reuse processes
- **Organizational Context** – the business strategies, policies and procedures, expertise, technological capabilities, cultural legacies, etc., of the set of organizations involved in a reuse effort

Outputs from the CFRP are:

- **Software Systems** – application systems built using domain assets
- **Assets** – new or refined reusable units of information relevant to a domain or set of domains

The arrows in Figure 1 represent, at a high level, the directions of information flow, influence, and feedback among the process families. They also, to some degree, indicate sequencing among families, although sequencing can vary substantially depending on specific circumstances, and activities within different families often will operate in parallel. The meaning of the arrows is discussed in more detail in the subsections below that describe the individual idioms.

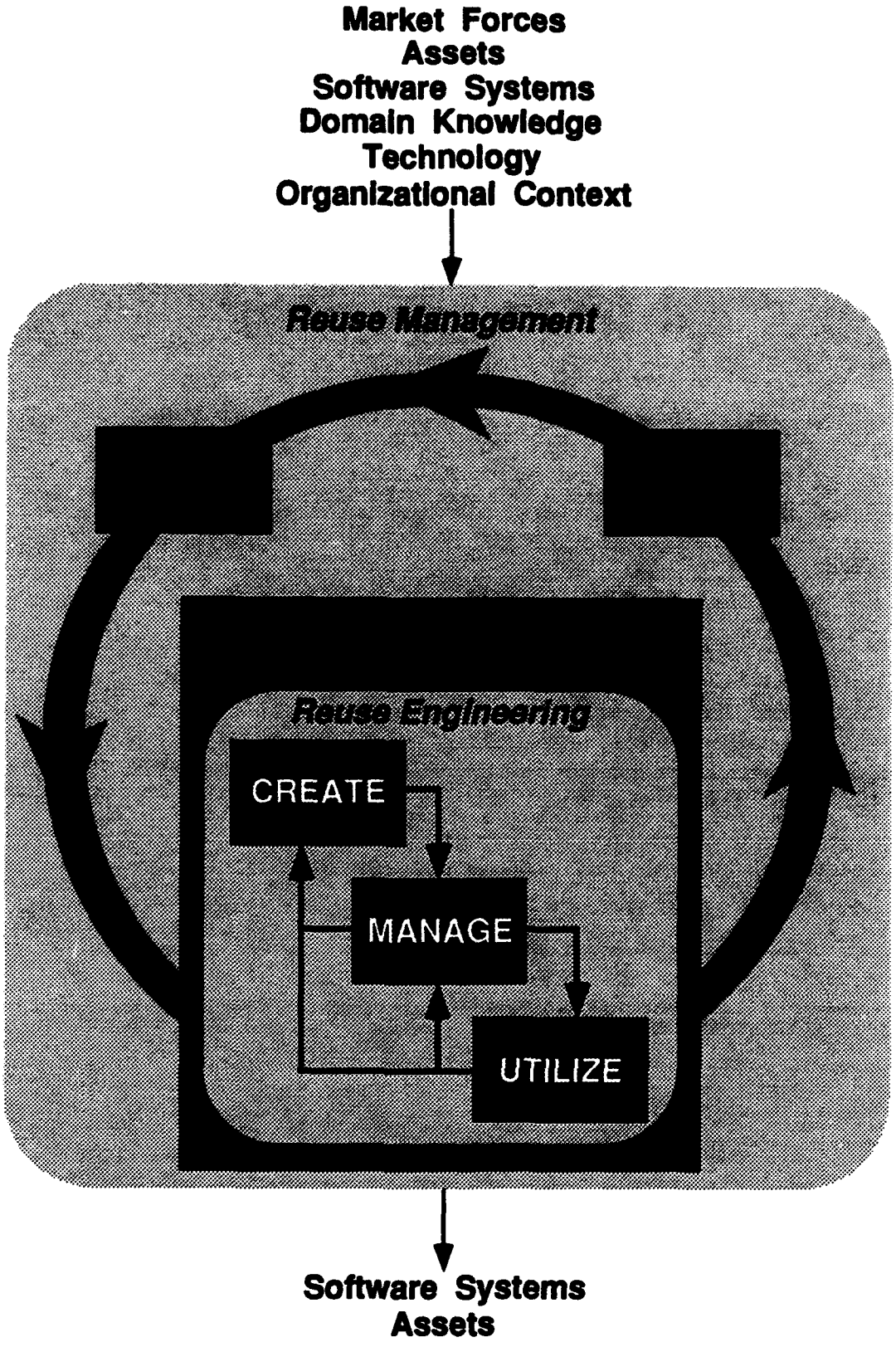

Figure 1: STARS Conceptual Framework for Reuse Processes

Reuse Management

- **Reuse Planning**
  - Assessment
  - Direction Setting
  - Scoping
  - Infrastructure Planning
  - Project Planning
- **Reuse Enactment**
  - Project Management
  - Infrastructure Implementation
- **Reuse Learning**
  - Project Observation
  - Project Evaluation
  - Innovation Exploration
  - Enhancement Recommendation

Reuse Engineering

- **Asset Creation**
  - Domain Analysis and Modeling
  - Domain Architecture Development
  - Asset Implementation
- **Asset Management**
  - Library Operation
  - Library Data Modeling
  - Library Usage Support
  - Asset Brokering
  - Asset Acquisition
  - Asset Acceptance
  - Asset Cataloging
  - Asset Certification
- **Asset Utilization**
  - Asset Criteria Determination
  - Asset Identification
  - Asset Selection
  - Asset Tailoring
  - Asset Integration

Table 1: CFRP Decomposition

The **Reuse Management** idiom describes a cyclic pattern of activity addressing the establishment and continual improvement of reuse-oriented activities within an organization by emphasizing learning as an institutional mechanism for change. Learning in this context means actively evaluating and reflecting on behavior to effect positive change. The idiom consists of the following three process families:

- *Reuse Planning* processes establish objectives and strategies for reuse within and across selected domains and application product lines; plan an interconnected set of reuse projects consistent with the objectives and strategies; and plan infrastructural capabilities to facilitate project performance and evolution.
- *Reuse Enactment* processes manage active reuse projects (e.g., allocate resources to them, initiate and retire them, monitor their performance); regulate day-to-day project performance; and ensure that a reuse infrastructure sufficient to meet the needs of the projects is established and maintained.
- *Reuse Learning* processes evaluate reuse project performance relative to local and global objectives and investigate and recommend approaches to effect evolutionary or revolutionary enhancements in reuse capabilities; they can be viewed as an institutional mechanism for managing improvement and innovation.

The **Reuse Engineering** idiom describes a "chained" pattern of activity that addresses reuse-related product development and reuse and explicitly recognizes the role of the broker as a mediator

between asset producers and consumers. The idiom consists of the following three process families:

- *Asset Creation* processes produce and evolve domain models and domain assets, including requirements and architecture assets, application generators, and software components.
- *Asset Management* processes acquire, describe, evaluate, and organize assets produced by Asset Creation processes, make those assets available to utilizers as a managed collection, and provide services to promote and facilitate reuse of the assets.
- *Asset Utilization* processes reuse the assets made available by the Asset Management processes by identifying, selecting, and tailoring desired assets and integrating them to construct application systems within target domain(s).

The Reuse Management and Reuse Engineering idioms represent reuse-specific specializations or adaptations of more general forms of organizational activity that can be termed the "Learning Loop" and the "Brokered Marketplace", respectively. The CFRP specializes these more general idioms to facilitate adaptation of a wide body of management and organizational theory to the software reuse problem.

Separating reuse processes into distinct Reuse Management and Reuse Engineering idioms within the CFRP not only cleanly distinguishes between these two aspects of reuse-related activity, but also enables very flexible configuration of reuse processes via a set of CFRP process composition techniques. These techniques, described in detail in Appendix A, can be used to construct a wide variety of reuse-specific process configurations reflecting different planning levels, organizational structures, and interaction patterns. Figure 1 shows one possible CFRP configuration, termed the "canonic" configuration. This configuration provides the most intuitive, straightforward view of the CFRP, wherein one set of Reuse Engineering activities is controlled directly by a set of Reuse Management activities. The canonic configuration is thus used throughout Section 3 as the basis for describing idiom and family interrelationships and information flows. However, other kinds of configurations are discussed briefly in some places to make key points about certain processes.

The CFRP idioms and families (in association with the CFRP composition techniques) represent a complete, cohesive system of reuse processes. That is, reuse processes can be adequately described at a high level by the classes and patterns of activity formed by CFRP idioms and families. Furthermore, eliminating a process family or otherwise substantially altering the relationships among the families or idioms could significantly alter the basic meaning of the CFRP and jeopardize its cohesiveness. The CFRP process families take on their unique characteristics largely by virtue of participating in a network of interactions with other families, as captured in the structure of the idioms. For example, Reuse Planning is defined with respect to its interactions with accompanying Reuse Enactment and Reuse Learning families. Similarly, Asset Management functions would have little meaning without the surrounding Asset Creation and Asset Utilization context.

On the other hand, the specific CFRP process categories defined in this document are considered to be less critical to the cohesiveness of the CFRP and are thus open to interpretation and tailoring. Some organizations may employ reuse processes that cannot be readily mapped to current CFRP categories, and they may wish to define additional categories. Alternatively, some organizations may elect to repartition the processes within the families to establish a different set of categories that is better suited to modeling that organization's processes. In general, the mapping of processes

to categories will vary across organizations and will change over time within any organizational context. The CFRP can thus be viewed as flexible and tailorable at the category level and likely to evolve over time to reflect experiences and alternative viewpoints.

The CFRP idioms, families, and categories form a hierarchical classification scheme for reuse-oriented processes. In addition, the CFRP describes how those processes can interrelate and interact. The CFRP can thus be viewed as a domain model and high-level architecture for the reuse process domain (with variability at the process category level, as noted above). As is true of domain models and architectures in general, the CFRP is intended to promote understanding and reuse within its domain. That is, the CFRP addresses not only reuse processes, but also reuse *of* those processes, by providing a framework for the definition and composition of process assets. However, the CFRP process elements do not cover all processes involved in software engineering, and the reuse of many processes addressed by the CFRP will involve integrating them with other non-reuse-oriented processes to compose total life cycle models tailored to the needs of an organization or project. The CFRP Application document addresses this and other applications of the CFRP in significant detail.

The two subsections that follow describe the Reuse Management and Reuse Engineering process idioms and their constituent process families and categories in greater detail. These subsections are not greatly dependent on one another and can be read in either order, depending on reader interest. Each subsection presents an overview of the idiom and then provides three subsections that describe each of the idiom families in significant detail. Each of these family subsections presents an overview of the family and then provides a set of subsections that describe each process category within the family.

3.1 Reuse Management Process Idiom

The Reuse Management process idiom describes a cyclical pattern of planning, enactment, and learning processes fundamental to domain-specific reuse-based software engineering. Reuse Management can be viewed as a systematic "learning loop" for managing innovation and improvement of the reuse capabilities within a reuse program. The Reuse Management idiom encompasses the following major functions, distributed throughout its three constituent process families, Reuse Planning, Reuse Enactment, and Reuse Learning:

- Determining and evolving the objectives, strategy, and scope of a reuse program, resulting in selection of a set of suitable domains and product lines in which to apply reuse within an organization;
- Planning, establishing, monitoring and evaluating Reuse Engineering (Asset Creation, Asset Management, and Asset Utilization) projects addressing the selected domains and product lines;
- Incorporating reuse processes into the organization's overall software engineering processes and policies;
- Planning, implementing and improving the organization's reuse infrastructure;
- Managing improvement, innovation, and research on reuse processes, tools, and domains within the organization.

Conventional software engineering focuses on the production of application systems, and Reuse Engineering widens this notion to include production of managed collections of reusable software assets (i.e., asset bases) addressing specific application domains. Reuse Management adds a strategic level above Reuse Engineering by selecting the key domains and product lines of interest for a reuse program, establishing reuse-related projects focusing on those domains and product lines, and learning about those projects and other aspects of the reuse program in systematic ways.

In planning reuse-related projects, Reuse Management emphasizes interconnections among the projects. Reuse by its nature implies interdependence and cooperation among organizations and individuals, so project interconnections must be explicitly planned and managed to ensure reuse effectiveness. In addition, since these interconnections can have significant structural and cultural impact within participating organizations, Reuse Management addresses issues of technology transfer, organizational change and restructuring, and evolution of engineering capabilities. Also, whereas Reuse Engineering processes establish a kind of asset marketplace internal to a reuse program, Reuse Management addresses the exchange of technology, assets, and domain knowledge with organizations external to the program. Reuse Management activities also address the planning, implementation, and evolution of the reuse infrastructure necessary to support the engineering projects within the program.

Reuse Management learning processes are consistent with the incremental, evolutionary improvement approaches exemplified by Continuous Process Improvement (CPI) and Total Quality Management (TQM). Reuse Management therefore supports definition of explicit, measurable objectives for reuse, collection of metrics necessary to evaluate success in achieving those objectives, and reflection on the data to improve the reuse approach in subsequent engineering activity. However, Reuse Management goes beyond such approaches by incorporating emerging general theories of organizational learning [Sch83, Bee72, Sen90] that have been adapted to the reuse-based software engineering context. Reuse Management thus includes processes for analyzing and synthesizing qualitative project experience and lessons learned, research results, and market advances to generate new discoveries and unanticipated insights. Such discoveries and insights, which can yield enhancements of a more revolutionary than evolutionary nature, emerge largely as a result of learning processes that explicitly encourage innovation.

Because of this integrated approach to learning and innovation, the Reuse Management idiom can be viewed as a kind of "maturity engine" that promotes continual increases in the maturity of reuse processes, technology, and overall reuse capabilities within an organization.

Idiom Elements and Their Interactions

The Reuse Management idiom consists of three process families, decomposed into a total of 11 process categories, as shown in Table 1. These families and categories are described in significant detail throughout the remainder of this section.

The following key concepts provide a useful basis for describing and understanding the relationships between elements of the Reuse Management idiom.

- **Reuse Program** – A *reuse program* is the set of activities and capabilities encompassed by a particular instance of the Reuse Management idiom. Each reuse program has a defined

scope representing the program's technical and organizational boundaries. The program scope includes the program's Reuse Management activities, a set of enacted reuse projects (and/or nested reuse programs), a set of selected domains and application product lines defining the technical context for the enacted activities, and an organizational context in which the overall program activities are performed.

- **Reuse Project** – A *reuse project* is any collection of Reuse Engineering activities that is managed and enacted as a unit by Reuse Management processes. Reuse projects as discussed in this document generally involve activities from only one of the Reuse Engineering process families. Each project may thus focus distinctly on Asset Creation activities, Asset Management activities, or application engineering activities involving Asset Utilization.

- **Reuse Cycle** – Because the primary flows of information within the Reuse Management idiom are from Reuse Planning $\Longrightarrow$ Reuse Enactment $\Longrightarrow$ Reuse Learning $\Longrightarrow$ Reuse Planning, reuse programs are conceived of as sequences of iterations, or cycles, through the three process families. Each *reuse cycle* (or *reuse program cycle*) represents one pass through the various Reuse Management steps in a particular reuse program.

- **Reuse Initiative** – The CFRP Reuse Management idiom is applicable both in organizations that are transitioning to reuse-based approaches and in organizations in which reuse has become ingrained. Management of the transition phase in which a reuse program is initially established will differ in some ways from the ongoing activities that sustain the program. A reuse program thus consists of an initial reuse cycle, termed a *reuse initiative*, to manage the transition phase, followed by a number of sustaining reuse cycles.

Although this document presents Reuse Planning as the first family in the Reuse Management idiom, reuse program cycles can begin with Reuse Enactment (representing a prototyping, exploratory style) or Reuse Learning (emphasizing research and empirical data collection from existing practice). Also, the term "reuse cycle" may be somewhat misleading, in that it may imply that Reuse Management activities move strictly sequentially between Reuse Planning, Enactment, and Learning. Even though the overall pattern is cyclical, there may be considerable concurrent activity among the families, particularly in stable reuse programs. While reuse projects are being enacted, observation and learning about the projects and infrastructure may be underway, and initial planning activity may have begun for the ensuing cycle or for some projects that are not completely synchronized with others.

The above definitions of "reuse program" and "reuse project" are most readily understood in terms of the canonic CFRP configuration shown in Figure 1. In this configuration, the outer Reuse Management idiom and the enacted Reuse Engineering families map clearly to the conventional, intuitive notions of a reuse program managing a set of reuse projects within a single organization. However, when considering more complex CFRP configurations (as discussed in Appendix A), the concepts of "reuse program" and "reuse project" are open to somewhat broader interpretations.

For example, any Reuse Management idiom can be modeled as having other Reuse Management idioms nested recursively within it (as in Figure 11 in Appendix A); each of these, by the above definition, is a "reuse program". Such CFRP configurations can reflect the hierarchical structure of an organization (e.g., company $\longrightarrow$ division $\longrightarrow$ department $\longrightarrow$ project $\longrightarrow$ engineer), with the Reuse Management idioms at each level of nesting representing a particular organizational level. The concept of a "reuse program" is thus scalable to any organizational level, and needn't reflect

only the static, intuitive view implied in Figure 1. Reuse projects are similarly scalable to reflect the scope of the work managed at different organizational levels, ranging from division-level reuse efforts down to individual engineering tasks.

In a situation involving nested reuse programs, the planning decisions made in any given program are considered to be "inherited" by its nested programs to establish a planning context for those programs. The nested programs thus are subject to the objectives, strategies, project plans, lessons learned, etc., that are produced at the higher level. The nested programs may refine, or perhaps even contradict, the inherited planning decisions, depending on the degree of planning freedom they enjoy. This planning freedom is generally a function of the management style prevailing within the organization.

Although the remainder of this section describes the CFRP in terms of the canonic CFRP configuration and the straightforward view of reuse programs and projects that it suggests (with the occasional exception where appropriate), some readers may wish to consider the impact of other CFRP configurations when reading this section.

The following subsections describe the Reuse Management process families in more detail.

3.1.1 Reuse Planning Process Family

The Reuse Planning process family encompasses both strategic planning and tactical, project-oriented planning within a reuse program. Strategic planning processes establish reuse program strategies that can span multiple reuse cycles (although each cycle may evolve the strategies). Strategic planning determines how organization resources should be applied to initiate and sustain reuse-based development in accordance with overall business objectives. Strategic *reuse* planning differs from traditional strategic or project planning, in part because of the need to consider the organization's key domains of interest. One key strategic reuse planning function, which augments traditional product line planning within an organization, is to select the key domains of focus for the reuse program and determine how the domain assets will support the organization's product engineering efforts.

Based on strategic domain and product line planning decisions, the tactical, project-level planning processes plan specific Reuse Engineering projects. These can include Asset Creation projects to create assets in the selected domains, Asset Management projects to manage the assets appropriately, and product engineering (Asset Utilization) projects to utilize the created assets. Reuse project planning processes generally plan reuse project activities for the duration of one cycle and replan relevant activities in each subsequent cycle, based on the results of Reuse Learning activities. The project planning activities address topics such as project processes, project interconnections, project infrastructure requirements, and resource planning. These planning activities can differ from conventional planning, due to the novel ways in which Reuse Engineering projects can be interconnected to exploit and evolve domain resources, and the way in which the CFRP promotes definition and tailoring of reuse-based project processes through integration of CFRP-consistent reuse process assets with existing process models.

Another key focus of Reuse Planning is the *reuse infrastructure* that is required to transition to and sustain a reuse-based software engineering approach. Infrastructure planning can focus both

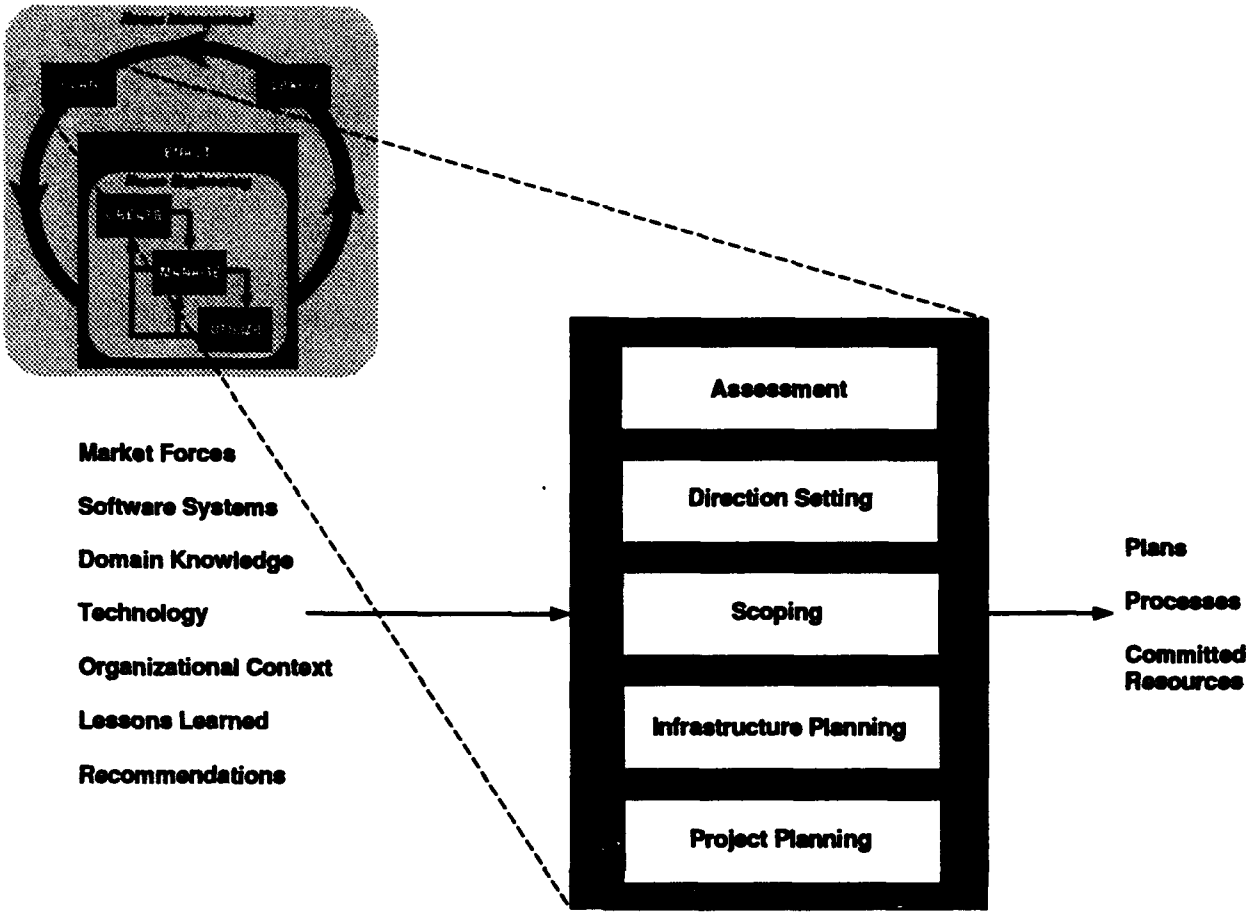

Figure 2: Reuse Planning Process Family

on long-term strategic capabilities and on the short-term requirements of individual projects.

As shown in Figure 2, the inputs to the Reuse Planning family include:

- **Market Forces** characterized by new market trends, competitive developments, new technologies, emerging standards, and other factors that impact perception of marketplace needs
- **Software Systems** in domains of interest that can impart legacy knowledge about the domains and feed domain analysis or reengineering efforts to produce domain assets or new application systems
- **Domain Knowledge** that can be imparted in a variety of ways (other than the legacy systems) to provide information about domains
- **Technology** that can contribute to the reuse infrastructure within an organization and can be applied to establish or automate reuse processes
- **Organizational Context** that includes the business strategies, policies and procedures, expertise, technological capabilities, cultural legacies, etc., of the set of organizations involved in the reuse program

- **Lessons Learned** that provide feedback from the Reuse Learning process family about preceding program cycles
- **Recommendations** from the Reuse Learning family regarding potential new domains and enhancements to the reuse program strategy, reuse infrastructure, and reuse processes

The outputs from the Reuse Planning family include:

- **Plans** for the reuse program, including program objectives and strategies, program scoping decisions, reuse infrastructure plans, individual reuse project plans (with planned interrelationships), and criteria and metrics for evaluating the program
- **Processes** to be employed by the projects
- **Committed Resources** to support the projects, in terms of funding, staffing levels, expertise, equipment, etc.

The subsections that follow describe each of the Reuse Planning process categories shown in Figure 2.

3.1.1.1 Assessment Process Category

Assessment processes characterize the current state of reuse practice within an organization, the readiness of the organization as a whole (or of specific groups) for practicing reuse-based software engineering, and the reuse technology and expertise available both internal and external to the organization. In addition, Assessment processes help to identify key opportunities for applying reuse to improve business practices and possibly gain competitive advantage.

Assessment draws upon the learning insights gained from previous reuse program cycles. Assessment processes thus form an important bridge between the Reuse Learning and Reuse Planning processes, enabling accumulated lessons to be used, not just learned. Learning results to be considered can include feedback about the state of reuse practice within the organization, specific recommendations (with rationale) based on evaluation of current practice, detailed lessons learned, and new discoveries and innovations. Depending on the degree of up-front planning desired, additional assessment activities internal to the reuse program can be initiated; for example, interviewing key personnel to assess experience and expertise in various domains, or inventorying software artifacts available for potential reengineering. Assessment activities can also take into consideration business needs and constraints as seen from a broader organizational perspective, and can analyze information from external sources addressing market factors such as competitive challenges, new technologies, emerging standards, developments within relevant application domains, and so on. Assessment processes synthesize the information from these various sources to establish a context for other planning processes.

While some assessment activities will be performed in ways that are unique to the needs and culture of an organization, other assessment activities can follow established or emerging guidelines developed within the reuse research community. For example, models are being developed that enable organizations or projects to formally assess their reuse capabilities in terms of particular dimensions or aspects of reuse [Sof93b, Vir92a].

3.1.1.2 Direction Setting Process Category

Direction Setting processes define specific objectives for the reuse program, strategies for achieving those objectives, and criteria for evaluating how successfully the objectives have been met. These planning elements can vary considerably from one organization to another, depending on factors such as overall business goals and high-level software engineering policies. For example, objectives and strategies for the following types of organizations may differ substantially: a company seeking to market reusable components or develop systems based on them, a DoD program office establishing a reuse program for a given application domain, a contracting organization developing or integrating custom systems, or a system maintenance organization.

Objectives and strategies can also address different stakeholder perspectives within an organization, at varying levels of detail. For example, reuse program objectives can include high-level business objectives (e.g., win certain contracts because of reduced cost and demonstrated domain capabilities), more detailed business or management objectives (e.g., increase productivity, reduce cost, or improve time-to-market by certain percentages), or technical objectives (e.g., modernize the technical infrastructure, establish domain capabilities to ease maintenance and improve quality).

The strategies developed to achieve program objectives are used to plan and guide the program's Reuse Engineering projects and reuse infrastructure. The strategies employed can, among other things, emphasize investment in different areas. One strategy might involve focused, capital-intensive Reuse Engineering efforts (e.g., domain analysis, investment in generative technology) in business areas where sufficient return on investment is projected and risk is acceptable, while another strategy might emphasize more broad-based efforts to cultivate reuse more naturally through tools, education, incentives, organizational redesign, and integration of reuse into organizational policies and procedures.

The models that are now emerging in the reuse community to support the Assessment process category (see above) also can offer a basis for Direction Setting. These models can be used not only to assess current capabilities, but also to define objectives for making specific improvements to those capabilities. Furthermore, by describing both current and desired capabilities in common terms, the models provide a useful context for developing strategies to transition to the desired capabilities. Such models also can assist in defining reuse program success criteria by giving planners guidance in deriving questions (and perhaps metrics) for evaluating whether the objectives are being met. Use of such models results in the direct coordination of Assessment and Direction Setting processes.

Since the objectives and strategies may depend in part on decisions about the specific domains and reuse-driven product lines on which the reuse program will focus (and vice versa), Direction Setting and Scoping processes may require close coordination. This will ensure that the strategies of the reuse program and the scope of its activities are mutually consistent.

Also, in practice, reuse-specific Direction Setting processes must be integrated with more general goal setting and strategy development processes within the program's organizational scope to integrate and prioritize reuse issues in a broader business context.

3.1.1.3 Scoping Process Category

Scoping processes define the overall scope of the reuse program by delineating the program's tech-

nical and organizational boundaries. The technical scope is defined in terms of the domains and product lines to be encompassed by the reuse program, while the organizational scope is defined in terms of the program's organizational context and management influence. A program scope's level of detail will generally be greater when the organizational scope is narrow, reflecting the finer-grained planning decisions made at lower levels within an organization.

Scoping processes draw upon information from various sources (particularly Assessment processes) to define a program scope that supports the reuse objectives and strategies defined by Direction Setting. As part of this activity, Scoping processes can focus on a variety of issues, such as candidate domains and product lines to target within the program, organizations or groups that could be involved in the program, potential asset and infrastructure suppliers and/or consumers in the program context, and other market or business factors that could influence decisions about program scope.

The major aspect of Scoping in the CFRP context that distinguishes it from more conventional planning approaches is selection of the key domains of focus for the reuse program and development of a planned approach for applying assets in those domains to selected application products or product lines. The selected domains may be targeted for Asset Creation efforts or chosen based on availability of an existing asset base. The process of selecting applicable domains may be tightly intertwined with the mapping of those domains to application products, because domain relevance to existing or planned product lines may be a key domain selection criterion.

A first step in domain selection is to identify and characterize promising candidate domains, based on the organization's business interests, key areas of expertise, and existing legacy systems. Some of this activity may be performed by Assessment processes. Criteria for identifying promising domains can be quite extensive; examples include [HCKP89, JHD+90]:

- The domain is well-understood and includes codified experience that can predict technology and provide domain expertise,
- The domain is based on predictable technology that will not make the reusable assets obsolete before the investment in their development can be recovered,
- Domain expertise is available to support Asset Creation, or relevant assets are already available in the domain, and
- The domain addresses key functional needs within existing and planned application products.

After candidate domains have been identified, they are further evaluated by applying additional or refined criteria, prioritized appropriately. This results in selection of one or more domains for which Reuse Engineering activities are to be initiated (or for which external sources of assets are to be selected). Goals of the domain selection process include:

- Identify the highest payoff, lowest-risk domains from among the identified candidates,
- Demonstrate sufficient promise in domains for which suitable assets are not available, to gain commitment for Asset Creation efforts, and

- Document the selection process and rationale so that it can be reused by subsequent domain selection processes.

In general, the initial domain identification activity focuses on the general technical feasibility of a domain in supporting reuse within the organization. The actual domain selection process focuses on criteria and trade-offs addressing more strategic issues, such as the relative priority of the domain in the current environment of an organization, the utility of the domain in supporting specific organization product lines, and economic factors such as projected return on investment. These issues should be considered in the context of factors such as domain stability and organizational priorities for near-term vs. long-term payoff.

3.1.1.4 Infrastructure Planning Process Category

Infrastructure Planning processes (a) identify needs for various types of support that are common among planned reuse projects, and (b) develop plans for establishing a shared reuse infrastructure to satisfy those needs. Reuse infrastructure can be informally divided into its technical, organizational, and educational aspects:

- *Technical infrastructure* includes computer and communication equipment, support tools and environments to automate reuse processes, technical support functions such as configuration management and quality assurance, reusable process definitions that are applicable across projects, technical standards and guidelines, etc.
- *Organizational infrastructure* includes staff members, office facilities, non-technical organizational support functions, organizational policies and procedures, organization and project history information, funding models, incentive strategies, institutionalized project role definitions, reuse-oriented task forces, transition teams and steering committees, etc.
- *Educational infrastructure* includes capabilities for developing and evolving necessary knowledge and skills among individual workers in an organization.

These aspects of infrastructure may not be as cleanly divisible as implied above. For example, process definitions may be codified in organizational policies and procedures. Furthermore, individual infrastructure elements will generally be interdependent and have implications on one another. For example, manually enforced reuse policies can eventually become automated by support tools, and introduction of new tools invariably creates new requirements for reuse education and training.

The infrastructure needs within a reuse program are closely tied to the program's reuse strategies, the pre-existing infrastructural capabilities, the organization's projected capacity for change, and the specific project strategies and processes. It is important to be realistic in planning the infrastructure so as not to introduce change at a greater rate than an organization can accommodate. Hence, Infrastructure Planning should involve short-term planning that addresses the specific infrastructure capabilities that can be accommodated in the current reuse cycle, as well as longer-term planning that addresses incremental insertion of more advanced capabilities over a span of several reuse cycles.

Infrastructure Planning and Project Planning are interdependent processes, since infrastructure needs can be fully determined only in the presence of project strategies and processes, and project plans describing the specific infrastructure capabilities to be used by each project can be finalized only in the presence of infrastructure plans. Thus, Infrastructure Planning and Project Planning should generally be closely coordinated and performed in parallel or iteratively.

Also, since many reuse infrastructure elements identified above overlap with traditional infrastructure needs in an organization, the reuse Infrastructure Planning processes must be coordinated with planning for the organization's overall infrastructure.

3.1.1.5 Project Planning Process Category

Project Planning processes plan the reuse program's Reuse Engineering projects in detail. This first involves determining which specific projects to plan, based on the domains and application product lines targeted for reuse activity by Scoping processes. Asset Creation projects are identified to produce desired domain capabilities, and application engineering projects are identified to incorporate Asset Utilization processes that exploit the domain capabilities. In addition, a set of Asset Management projects is identified that will most effectively serve the needs of the (possibly multiple) Asset Creation and Asset Utilization projects. In any of these cases, new projects may be established or existing projects may be evolved, as appropriate. Once the individual projects are identified, dependencies and information flows among the projects are defined explicitly. The resulting plans can be considered a "project architecture" for the reuse program, specifying the relationships and interconnections of the various Reuse Engineering projects in CFRP terms.

Some of the projects identified via the above approach may be external to the organization establishing the reuse program. For example: external Asset Creation projects may exist that are already producing relevant assets in a target domain; external clients may exist for domain assets that the organization is producing; relevant assets may be available through an external organization providing Asset Management services in a target domain. To the extent that such external involvement is consistent with the organizational scope and strategy of the reuse program, Project Planning processes should evaluate such opportunities (perhaps identified by Assessment or Scoping processes) and incorporate external projects into the "project architecture" appropriately. The Project Planning activities described below relate to projects that are within, or can be directly influenced by, the planning organization.

Project Planning is responsible for establishing specific objectives for each identified project, and for defining the metrics to be used to evaluate the effectiveness of the projects relative to those objectives and the objectives of the overall reuse program. A variety of metrics can be defined relating specifically to assets and reuse. Examples include:

- The number of assets produced in each domain
- The percentage of each application product that was directly derived from domain assets
- The percentage of assets in each domain that have been reused at least once
- The number of times each asset has been successfully (or unsuccessfully) reused
- Reliability and maintainability metrics for the assets and resulting application products

- The number of assets that have been extracted from or submitted to a library
- The number of regular users of an asset library
- The rates of change in the above metrics over time

A key Project Planning activity is definition of the specific processes that will be employed on each project. In a mature, stable reuse program, these processes may be institutionalized as policies and procedures within the organizational infrastructure, requiring only minor tailoring for each project. In less stable circumstances, such as in a reuse initiative, reuse-based processes generally will not be available within the organization and will have to be developed from scratch or obtained (and probably adapted) from other sources. The CFRP supports the definition and reuse of reuse process assets to facilitate project process definition and tailoring, so if an external process asset library is available containing such assets, they can be integrated with existing conventional life cycle processes to incorporate needed reuse capabilities.

Another aspect of Project Planning is the identification of reuse infrastructure capabilities (aside from process definitions) that will be used by each project to support the processes it will employ. In addition, plans should be developed for tailoring the identified capabilities to suit the needs of each project, if such tailoring is necessary.

The final stage of Project Planning is to plan the reuse projects' resource needs, budgets, and schedules in detail and then obtain the necessary commitment to implement the plans. Commitment should be obtained from both higher level management and the technical staff members whose buy-in will ultimately determine whether or not reuse practice is really improved by the program. Obtaining this commitment can be difficult, due not only to technical considerations and resource requirements, but also resistance to change. Iteration among Direction Setting, Scoping, and Infrastructure and Project Planning processes will often be required before the necessary commitment can be obtained.

3.1.2 Reuse Enactment Process Family

The Reuse Enactment process family addresses initiation, performance, and retirement of the various reuse-related projects planned by Reuse Planning. Reuse Enactment processes also include supervisory activities, such as project control, monitoring, mid-course corrections, and local adaptation of processes and policies to accommodate project-specific circumstances.

Although Reuse Enactment processes can, in general, manage and control any project or process (including nested Reuse Management processes or even non-reuse-oriented processes), the emphasis in this section is on management of a set of interrelated projects employing Reuse Engineering processes. The managed projects and their interrelationships reflect the "project architecture" produced by Project Planning.

As shown in Figure 3, the inputs to the Reuse Enactment family include:

- **Plans** for the reuse program, including program objectives and strategies, program scoping decisions, reuse infrastructure plans, individual reuse project plans (with planned interrelationships), and criteria and metrics for evaluating the program

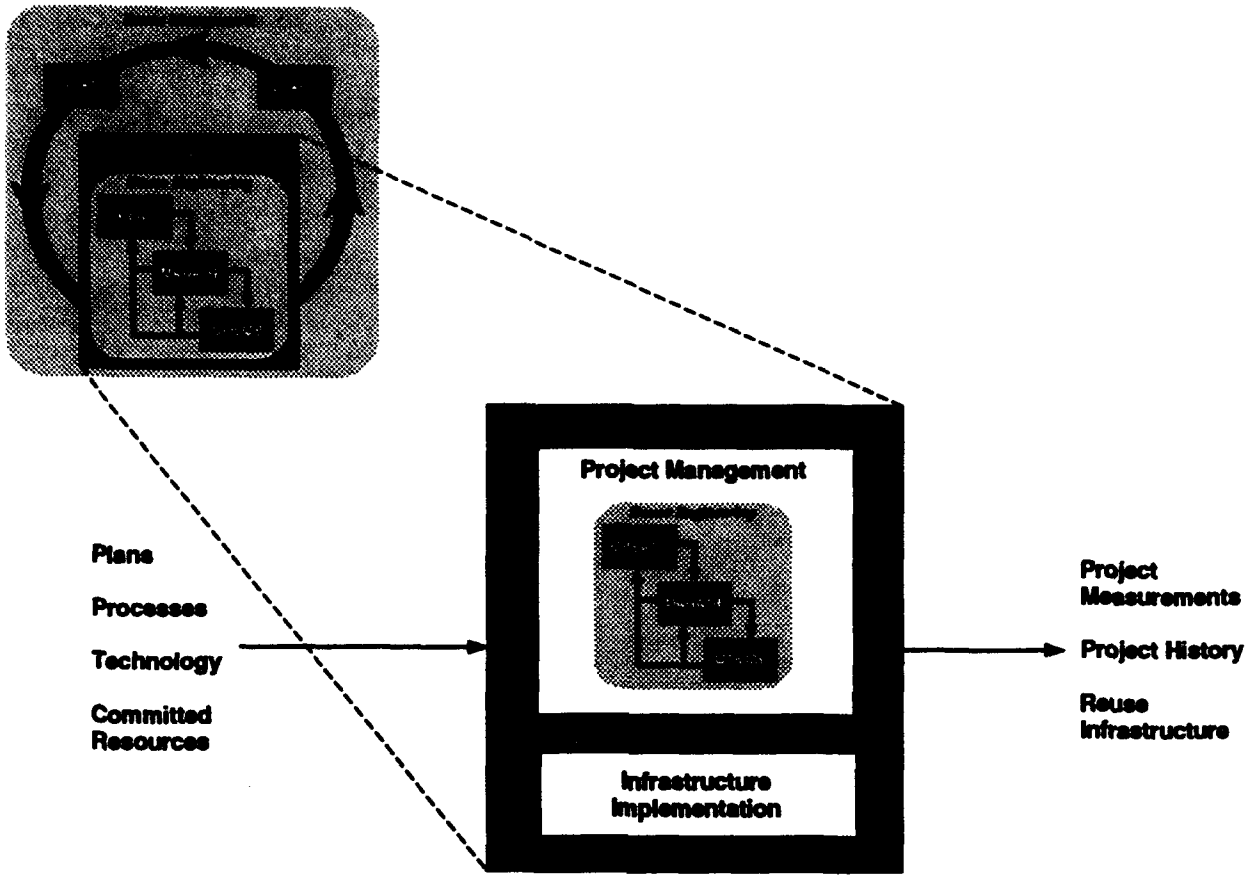

Figure 3: Reuse Enactment Process Family

- **Processes** to be employed by the projects
- **Technology** that can contribute to the reuse infrastructure and can be applied to establish or automate reuse processes
- **Committed Resources** to support the projects, in terms of funding, staffing levels, expertise, equipment, etc.

The outputs from Reuse Enactment processes include:

- **Project Measurements** collected during project enactment to measure the effectiveness of the project processes, products, and infrastructure
- **Project History** that provides qualitative historical information about the project processes, products, and infrastructure
- **Reuse Infrastructure** that is implemented and evolved within Reuse Enactment

The subsections that follow describe each of the Reuse Enactment process categories shown in Figure 3.

3.1.2.1 Project Management Process Category

Project Management processes establish a temporal context for reuse project activities and perform detailed project supervision. Temporal Project Management tasks include project initiation, performance, and retirement, in recognition of the fact that projects and processes have distinct lifetimes within an organization. Supervisory tasks include project control and monitoring.

Project processes can be viewed as generating output that includes not only explicitly planned products, but also project measurements and history of various kinds. Detailed performance history and rationale, adjustments in performance relative to plans, and metrics are all secondary or "learning-oriented" rather than "product-oriented" results of project enactment. Reuse Management, in controlling any project, should prioritize and balance the product-oriented and learning-oriented activity to ensure that each is adequately emphasized.

Project Management includes the following specific functional areas:

- *Project initiation* activities include allocation and tailoring of technical reuse infrastructure capabilities to specific projects; allocation of project resources committed during Reuse Planning; staffing, training, and team formation for the specific organizational units involved in the projects; and application of incentives, funding strategies, and other relevant organizational policies and mechanisms to the projects.

- *Project performance*, at the core of Reuse Enactment, is where the processes being enacted are "hooked in" to the Reuse Management idiom and actually performed by individual staff members. Project performance activities are distinct from the processes being enacted in that they involve tactical decisions about *how* the work will be performed on a detailed, day-to-day basis. At one level, project performance activities can be viewed as individualized techniques for filling in the minute implementation details that are generally missing from project processes established in Project Planning.

- *Project control* activities intervene with project performance to optimize overall project performance relative to reuse program and project objectives and constraints. Project control is the "management" function (in the most conventional sense) for the projects being performed. Project control activities may dynamically adjust project budgets, schedules, staffing, processes, infrastructure, task assignments, etc., in response to performance deficiencies identified during project enactment.

- *Project monitoring* activities capture "learning-oriented" information from the projects as they are performed. Some of this information provides detailed feedback to project control activities to support needed project adjustments. Such information can include data collected in accordance with some or all of the reuse project metrics identified in Project Planning. Project monitoring activities also supply information to Reuse Learning processes. This information includes all reuse program and project metrics data, as well as more qualitative project history information such as a record of the process steps that were applied and rationale for the decisions that were made and the workproducts that were created.

- *Project retirement* activities can include termination procedures for the project, capture of essential project results, elimination of information needed solely to maintain the project on an ongoing basis, debriefing of experts on the knowledge gained during the project, and

archival of the key results and knowledge as part of the reuse infrastructure. Since every process produces knowledge as well as products, and some of this knowledge can only be finalized with project completion, a managed project retirement process should be an explicit part of a reuse-based engineering paradigm. This will enable each project to become part of an established reuse program legacy that may be assessed or reused in future reuse cycles.

Project control activities may involve project re-direction — not merely simple adjustments during performance to better meet project objectives, but also updates to original planned projections, budgets, milestones, schedules, and technical approaches, or even revisions of the objectives based on experience or rapidly shifting conditions. Such mid-course corrections should be documented, with associated rationale, so that later evaluations of project success can take into account any substantive changes to the plans or objectives. In the case of projects where the learning output is prioritized higher than the product output (such as in training or educational settings, controlled R&D experiments, or shadow projects), re-direction may be discouraged or prohibited to ensure that needed lessons are learned.

3.1.2.2 Infrastructure Implementation Process Category

Infrastructure Implementation processes ensure that reuse infrastructure capabilities are established and evolved in accordance with infrastructure plans and evolving project needs. Infrastructure Implementation can be divided into the technical, organizational, and educational aspects of reuse infrastructure identified in the description of the Infrastructure Planning category above.

- The reuse *technical infrastructure* includes software process and method descriptions, tools and environments to provide automated support for reuse processes, technical standards and guidelines, technical support functions (e.g., configuration management, quality assurance), and computer hardware and other technical resources needed by reuse projects.

  To be considered part of the reuse infrastructure, tools should specifically support reuse processes and should be applicable across multiple projects within the reuse program's scope. Reuse support tools may include tools developed specifically to support reuse (e.g., modeling tools to be used by domain analysts during Asset Creation), or may be general software engineering tools adapted to serve reuse needs (e.g., code inspection tools that are used for asset understanding). Implementing the technical infrastructure may involve acquiring technology through external organizations (e.g., commercial vendors) or developing it in house if appropriate.

  Existing technical support functions within an organization may be called upon to perform similar roles in a reuse context, and they will need to be adapted appropriately to support reuse. For example, a quality assurance department that is assigned responsibility for Asset Certification will need to adapt their existing processes to take reuse-specific factors into account.

- A key aspect of the reuse *organizational infrastructure* is the body of institutionalized policies and procedures that regulate various tasks and activities within an organization. An appropriate set of policies and procedures should be selected and tailored (as allowed) for use on the reuse program. Policies and procedures may overlap with the technical infrastructure in that

they may mandate the use of specific processes, methods, guidelines, standards, and tools within a broad organizational context. Over time, reuse-specific practices should be captured in policies and procedures as they become more widespread throughout an organization.

Organizational infrastructure implementation also involves activities to standardize reuse-oriented project role definitions, establish funding models and incentive strategies that promote reuse, and establish reuse-oriented task forces, transition teams, steering committees, and so on. Funding models can include reuse-oriented software cost estimation models, domain engineering return-on-investment models, etc. Incentive strategies can include reward systems for investment in reuse and for creation and utilization of assets, publication of success stories and testimonials attesting to the benefits of reuse, etc.

Another facet of the organizational infrastructure is the ongoing capture and maintenance of organizational and project history information. This can include project plans, results, experiences, lessons learned, and other information that can be of use to future planning efforts. A sophisticated way of dealing with this issue is to manage such information in an "organizational asset library" that includes not only historical information, but also other aspects of the infrastructure, such as process definitions, policies and procedures, and so on. In this context, Reuse Management activities can be viewed as also playing Asset Creation, Management, and Utilization roles with regard to these organizational assets.

Organizational infrastructure activities also address infrastructure issues such as building a staff of suitable size with suitable reuse skills and experience, providing adequate office facilities, and establishing organizational support functions (e.g., photocopying and documentation support).

- The reuse *educational infrastructure* is needed to develop and maintain the reuse skills and capabilities needed by individual workers in the context of a reuse program. Transitioning to reuse-based software engineering will require cultivation of new technical and interpersonal skills to accommodate new project roles, new processes and tools, and new forms of team interaction. Existing personnel may be assigned to the new reuse-oriented roles, or new personnel may be hired to address specific needs. In either case, training in reuse concepts and skills will be needed, though the needs will vary from one reuse project to the next, depending on the background of the workers involved. The organization will need to either establish an in-house training capability to accommodate these educational needs, or acquire training from external sources.

The above view of Infrastructure Implementation suggests a general bottom-up approach for "promoting" infrastructure capabilities from initially narrow use to increasingly widespread and standardized use, based on their increasing level of acceptance and generality over time. In this approach, reuse capabilities (e.g., processes, guidelines, tools) are acquired or developed for use on individual projects (perhaps even by individual engineers in their daily work). These capabilities are monitored and evaluated as part of Infrastructure Implementation, and the capabilities that are considered general (or generalizable) enough to be reusable for other projects are "promoted" to the common infrastructure. As they achieve progressively more widespread use, they may be further promoted by being incorporated into organizational policies and procedures that institutionalize their use throughout the organization.

3.1.3 Reuse Learning Process Family

The goal of Reuse Learning processes is to enhance the plans, performance, and overall reuse capabilities of a reuse program, based on reuse project results and other relevant factors. Reuse Learning achieves this goal in two major ways:

1. Identifying opportunities for improvement in the reuse program by evaluating reuse project performance relative to program and project objectives.
2. Exploring opportunities for innovation that can be applied to the reuse program by analyzing project history and experience in light of external factors (e.g., emerging technologies, changes in domain requirements) to produce unanticipated discoveries and insights.

The results of Reuse Learning are fed back to Reuse Planning processes in the form of recommendations for the next reuse program cycle. Such recommendations can address overall program objectives and strategies (e.g, increase projected cost savings from reuse, apply reuse more in software maintenance contexts), program scope (e.g., consider specific new domains, drop or rescope existing domain efforts, expand existing reuse-driven product lines), reuse infrastructure (e.g., obtain new tools, evolve existing processes, establish additional training), or reuse projects (e.g., adjust individual project processes or tool choices, modify project interconnections, make greater use of external asset sources). Reuse Learning results can also be propagated outside the program to other reuse programs and organizations or to the reuse research community.

Learning is relevant to any activity emphasizing continual improvement, but it is particularly vital to a managed approach to software reuse. Reuse is founded on the notion that the results of prior work can be reapplied in current and future work. This only holds, in general, if future needs are continually analyzed, work is performed to anticipate those needs, and workproducts are continually evolved in concert with those needs. Such an approach requires continual learning about the workproducts, the needs they must target, and the overall context in which the work is performed. Although such learning can occur informally, the CFRP includes Reuse Learning as a separate process family to encourage planners and managers to explicitly designate resources for learning activities, rather than to have them occur in an ad hoc and unsupported manner. The CFRP, by explicitly distinguishing between the production and learning aspects of software engineering processes, encourages planners to directly confront and balance the trade-offs between product-oriented and learning-oriented activities.

In addition to the product- or goal-driven approaches to learning that are characteristic of conventional improvement methods such as TQM, Reuse Learning emphasizes a "reflective" learning mindset [Sch83]. With this mindset, process and product improvement opportunities are identified not only through measurements relative to specific objectives, but also through reflective observation of the processes and products to reveal new insights, lessons, and discoveries. This approach can yield valuable results, such as new ideas for methods and strategies to apply in reuse activities, suggestions for redesign of the technical infrastructure, ideas for new assets addressing higher levels of abstraction, and new training concepts to better prepare personnel for reuse tasks. For example, the rationale for decisions made by asset creators in developing a domain architecture, combined with feedback from asset utilizers in using the architecture to build applications, may lead to the realization that a process is needed for developing architectures having a completely different archi-

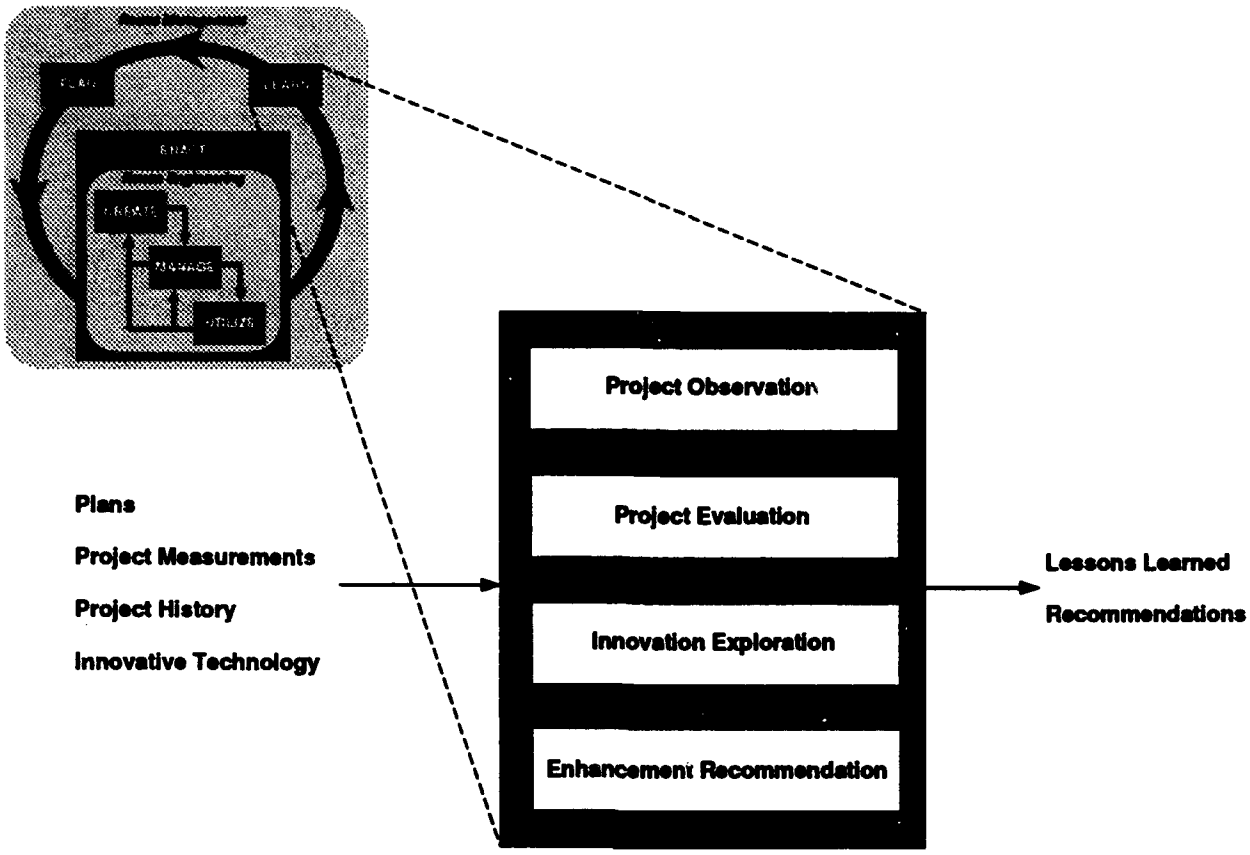

Figure 4: Reuse Learning Process Family

tectural style. The new process would not be an incremental improvement of the existing process, but a substantially new and different process.

As with the CFRP idioms and families in general, the Reuse Learning family is relevant at each level of planning within an organization, and across the software life cycle. At the personal level, Reuse Learning processes can be applied to improve individual workers' performance and skills through systematic planning and reflection in the context of day-to-day work. At the project team level, Reuse Learning can play a vital role in the continual evolution of project processes and products. At the level of the organization as a whole, Reuse Learning can focus on continually enhancing reuse capabilities across a broad range of domains and projects.

As shown in Figure 4, the inputs to the Reuse Learning process family include:

- **Plans** for the reuse program, including program objectives and strategies, program scoping decisions, reuse infrastructure plans, individual reuse project plans (with planned interrelationships), and criteria and metrics for evaluating the program
- **Project Measurements** collected during project enactment to measure the effectiveness of the project processes, products, and infrastructure
- **Project History** that provides qualitative historical information about the project processes, products, and infrastructure

- **Innovative Technology** from sources internal or external to an organization that can have impact on the processes, infrastructure, or assets within the reuse program

The outputs from the Reuse Learning family include:

- **Lessons Learned** from the current program cycle that can contribute to the overall reuse program experience base and influence ensuing planning activities
- **Recommendations** regarding potential new domains and enhancements to the reuse program strategy, reuse infrastructure, and reuse processes

The subsections that follow describe each of the Reuse Learning process categories shown in Figure 4.

3.1.3.1 Project Observation Process Category

Project Observation processes gather information about enacted reuse projects, package it as appropriate, and make it available for other Reuse Learning processes. The information can be used to evaluate project performance relative to objectives or can provide a basis for Innovation Exploration. The information may also be considered to have legacy value by other reuse programs or the reuse community at large. The project information can be gathered in a variety of ways, including observations via instrumented tools and environments; reflective observation on the part of project team members; solicitations of information from project team members through interviews, questionnaires, debriefings, etc.; or observations by outside researchers. One set of information that should always be gathered by Project Observation processes is metrics data generated in Project Management in accordance with the project metrics defined in Project Planning. Some project observations will inevitably be performed in parallel with reuse project activity, rather than taking place after completion of the project.

Whereas TQM and similar approaches emphasize detection of problems such as breakdowns in information flow within the work environment, a reuse-based approach also involves searching for common sequences of engineering tasks that could be encapsulated in reusable form to promote process reuse. Observers can learn to identify such common sequences of operations, but this involves a set of skills quite different from general software engineering skills. As a reuse program matures, these process observation skills should become increasingly ingrained and integrated within the overall software engineering paradigm. A mundane example of such skills is an experienced Unix programmer's ability to recognize recurring sequences of system commands that can be encapsulated in shell scripts.

3.1.3.2 Project Evaluation Process Category

Project Evaluation processes compare the project results that are generated in Reuse Enactment (and gathered in Project Observation) with the reuse program and project objectives and success criteria developed in Reuse Planning. This comparison yields a detailed evaluation of the successes and shortcomings of the reuse program in the current reuse cycle. The results of the evaluation

directly support the generation of recommendations for incremental improvements in subsequent reuse cycles.

It is important to distinguish the Project Evaluation processes that support overall reuse program evolution from the Project Management processes that apply mid-course corrections to projects during enactment. Project Evaluation processes assess the performance of projects relative to their objectives to support evolution of reuse program and project plans in the context of an overall reuse program cycle. They *do not* provide feedback directly to Project Management processes to intervene in the ongoing enactment of current projects. Note, however, that even though Project Evaluation can be viewed conceptually as occurring after the evaluated projects are completed, in practice the evaluation activities may be concurrent with Project Management activities. They may even be performed by the same people, and those people should understand the distinctions between the processes.

3.1.3.3 Innovation Exploration Process Category

A primary goal of Reuse Learning processes is to support the evolution of an organization's reuse-based software engineering capabilities. Innovation Exploration processes address this goal in a different way than Project Evaluation, by gathering, generating, analyzing, and testing new ideas, discoveries, and innovations to generate recommendations for potentially dramatic improvements.

One source of inspiration for Innovation Exploration processes is the results of the Project Evaluation activities. When discrepancies between objectives and performance are discovered, hypotheses or interpretations of probable causes for the discrepancies are often proposed. In general, such hypotheses should be explored before becoming the basis for concrete planning in subsequent reuse program cycles. Innovation Exploration provides the necessary managed-risk environment for testing these hypotheses. Many R&D groups perform this kind of role within software engineering organizations, but perhaps in a less managed or focused manner than is suggested here.

Innovation Exploration activities can also draw from Project Observation results. These results can include project history, rationale, and lessons, as well as interesting artifacts that emerged during engineering activity even though they were not anticipated in project plans. Here, exploratory activity might involve collecting further data to assess hypotheses based on the engineering results, or establishing experiments to build on those results in innovative ways to yield substantial new discoveries and insights. In the context of such activity, Innovation Exploration can serve as a key resource within Reuse Management for gathering and evaluating potential assets, domain knowledge, and technology innovations from external sources (e.g., the research community, the commercial marketplace, publicly available asset bases). Conversely, Innovation Exploration can serve as a conduit for exporting internal innovations and research results to the external community.

A key question that may need to be addressed as a result of Innovation Exploration activities is whether to modify program objectives that have not been attained, or introduce technical or policy innovations that can increase the program's overall reuse capability to make the objectives attainable. Potential recommendations that could result from Innovation Exploration to increase reuse capability include dramatic changes to reuse tools, environments, and processes; restructuring of organizational policies and procedures; changes in emphasis in organizational training materials; and nomination of ripe new target domains. By their very nature, Innovation Exploration processes promote substantial, and often revolutionary, change, whereas Project Evaluation processes

tend to generate more incremental improvements. Together, Innovation Exploration and Project Evaluation processes can provide a flexible set of mechanisms for promoting a variety of different kinds of change in an organization.

3.1.3.4 Enhancement Recommendation Process Category

Enhancement Recommendation processes propagate the verified results of Project Evaluation and Innovation Exploration to the Reuse Planning family of processes. Enhancement Recommendation may often emphasize consideration of the *technical* feasibility or desirability of new reuse approaches, leaving more management- and business-oriented considerations to Reuse Planning. Such analysis will generally be performed in the context of information available from within the immediate program scope (e.g., plans, project results, and innovations from the current reuse cycle). Enhancement Recommendation can also identify assets and domain resources that could be made available to other reuse programs within or outside an organization.

Enhancement Recommendation may involve synthesizing the results of a number of separate investigative activities into concrete recommendations for the next planning cycle. Enhancement Recommendation activities should appraise and trade off potential enhancements and innovations as completely as possible before putting forth specific recommendations. Just as the process of obtaining commitment within Project Planning may result in a number of iterations to revisit program objectives and strategies, so the enhancements suggested by Innovation Exploration and Project Evaluation may be initially rejected as insufficient or inappropriate, thus initiating a further round of investigation. It is left to Reuse Planning functions to make final decisions on adoption of the recommendations, depending on circumstances that prevail at the time of planning. Since the planning cycle may be quite long-term, market forces, technology, and other factors could shift significantly between the time recommendations are developed and the commencement of a new reuse program cycle. On the other hand, there could be short-term feedback from Reuse Planning that could trigger additional investigation of enhancements.

3.2 Reuse Engineering Process Idiom

The Reuse Engineering idiom describes a pattern of activity that addresses the creation, management, and utilization of reusable assets in support of domain-specific reuse-based software engineering. Because Asset Management serves in a kind of brokerage role between Asset Creation and Asset Utilization, the idiom can be viewed as a reuse-specific specialization of a more general Producer-Broker-Consumer idiom that reflects common marketplace interactions.

The Reuse Engineering idiom embodies the following general principles, which are characteristic of the idiom as a whole and impact each of its individual process families:

- Domain analysis and the resultant models and architectures permeate all aspects of Reuse Engineering and are critical for establishing the domain-focused mindset that is essential to the success of a domain-specific reuse program. Because domain models and architectures encapsulate an organization's domain knowledge in ways that can be directly utilized to build strategic application products, they will, over time, have fundamental impact on how

an organization thinks about its business and marketplace, how it builds its products, and how it applies its investment dollars (e.g., to improve and broaden its asset base).

- Reuse-based approaches are inherently quality-oriented. Reusable assets must be of high quality to encourage their repeated reuse. Asset libraries therefore should carefully manage asset quality by establishing well-defined quality criteria, ensuring that only assets of acceptable (or higher) quality are installed, and making information about the quality of assets available to reusers. Also, since assets will be of value to an organization over significant periods of time spanning multiple application projects, engineering processes should systematically generate feedback about assets to promote continual improvement of asset (and thus application) quality over time.

- The asset brokerage role is fundamental to reuse. Reuse is often described in terms of a simple dichotomy between "domain engineering" and "application engineering", and asset management is typically just lumped into one of these two areas. However, this approach limits flexibility when, for example, working with applications that cross domain boundaries or when creating assets in two or more domains that have redundant subdomains. Explicitly recognizing the importance of the asset management role in mediating and promoting interactions between asset creators and utilizers results in substantially greater flexibility in configuring processes and underlying infrastructure and in evolving them over time.

- The CFRP is neutral with respect to particular reuse approaches and technologies and recognizes that they should be allowed to evolve over time within an organization in response to a variety of factors (e.g., application technology changes, architecture evolution). However, it is important that the specific approaches and technologies that are used among interdependent organizations be compatible across the Reuse Engineering families. For example, asset creators should produce application generators only if asset utilizers are ready and equipped to use them.

Idiom Elements and Their Interactions

The Reuse Engineering idiom consists of three process families, decomposed into a total of 16 process categories, as shown in Table 1. These families and categories are described in significant detail throughout the remainder of this section.

Within the Reuse Engineering idiom, there are two principal kinds of information flow among the process families (as depicted without labels in Figure 1). One is the flow of assets and associated information from Asset Creation to Asset Management and in turn, in managed form, to Asset Utilization. The other is the flow of feedback about assets, reuse services, and other related information from Asset Utilization to Asset Management and in turn to Asset Creation. Note that Figure 1 also shows a direct flow from Asset Utilization back to Asset Creation, in recognition of the fact that utilizers may often inform creators directly about significant shortcomings in the domain model or architectures or in specific domain components. However, it is also possible for Asset Management to be the mediator for all feedback between utilizers and creators, as may sometimes be the case with centralized libraries that provide Asset Management services spanning multiple domains. Feedback from Asset Utilization that is targeted specifically to Asset Management commonly addresses the quality of the Asset Management services themselves or focuses on relatively minor problems with assets that can be resolved within Asset Management.

Control flow among the Reuse Engineering families can be directly related to the information flows described above, in terms of responding to feedback or to requests to manage or utilize assets. In general, however, control flow among the Reuse Engineering families is difficult to characterize, because it can differ greatly from organization to organization. In an organization that has a mature reuse program underway, there will likely be Asset Creation, Asset Management, and Asset Utilization projects in operation simultaneously and often asynchronously. It is usually only in the early stages of a domain-specific reuse program that any clearly sequential high-level flow of control among the families may exist; e.g., an Asset Creation project may be started to produce an initial asset base, and after there is a critical mass of assets, an Asset Management project may begin operation, at which point Asset Utilization may begin. However, even this scenario may not be typical in practice, since many organizations, before beginning formal Asset Creation efforts, will establish some form of reuse library capability to attempt to reuse existing artifacts as much as possible.

The data and control flows among the families are important to understanding how assets and asset libraries evolve through feedback and refinement. However, there are no specific asset or library evolution process categories defined within the Reuse Engineering idiom. This is because evolution and refinement of products and processes is considered to be an implicit, natural phenomenon within the CFRP, driven by continual application of the learning processes that are fundamental to the Reuse Management idiom. In effect, each process has "built-in" capabilities for evolving its own products, via higher-level or embedded learning processes that respond to feedback and lessons learned from the development and use of the products. Such evolution can be gradual (e.g., addressing identified shortcomings or bugs in individual assets; adjusting the library data model to better address user needs) or can be more dramatic (e.g., generalizing or combining assets into larger-scale assets; producing new, complementary assets to address evolving application requirements; developing new tools to further automate composition of applications using particular sets of assets).

Similarly, there are no explicit metrics collection processes in Reuse Engineering because of the metrics definition, collection, and analysis activities inherent in the Plan–Enact–Learn mechanisms within Reuse Management. This implies that the specific metrics needed by an organization with regard to Reuse Engineering processes and products (e.g., library usage metrics) are planned, collected, and evaluated by Reuse Management processes.

The above discussion of data and control flow generally assumes a straightforward configuration of projects within the different families, where all of them are operated and controlled within the same organization. As discussed in significant detail in Appendix A, substantially more complex configurations among Asset Creation, Management, and Utilization projects are possible. These can span organizational boundaries and involve complex interconnection patterns among organizations. The data and control flows in each of these kinds of situations is likely to be highly idiosyncratic and needs to be considered carefully on a case-by-case basis during planning.

3.2.1 Asset Creation Process Family

The goals of the Asset Creation process family are to (a) capture, organize, and represent knowledge about a domain, and (b) use that knowledge to develop reusable assets that can be applied to produce families of systems (and other products) within the domain. Asset Creation can be viewed

as consisting of:

1. *Domain Analysis and Modeling* processes that analyze, abstract, and model the characteristics of existing and envisioned application products within a domain in terms of what the products have in common (their "commonality") and how they may vary (their "variability"). This information is captured in a set of *domain models.*
2. *Domain Architecture Development* and *Asset Implementation* processes that produce reusable assets providing domain capabilities that reflect particular product commonalities and ranges of variability defined by the domain models. These assets form a domain *asset base.*

In general, the assets produced by such an approach support a range of capabilities identified during domain analysis, and should thus be applicable in multiple system contexts. In contrast, traditional software development practices emphasize development of point solutions that satisfy exactly one set of system requirements. As a simple example, countless implementations of the "stack" abstract data type have been developed to address some unique set of application requirements. These implementations, in general, require some adaptation or reengineering to perform correctly in different applications. However, some programming languages, such as Ada, permit the definition of a generic stack package that can be tailored to form a specific stack implementation by supplying parameters such as the type of element to be stacked or the maximum number of elements to be stacked. The parameters that the package supports are, in effect, identified through a form of domain analysis for the stack domain (although the package designers may not be aware of that). Such a generic package will generally be more reusable (i.e., applicable to a broader range of systems) than a package in which the parameter values are hard coded.

Domain models and asset bases, as presented in this document, are logically at different levels of abstraction and serve different purposes. The primary role of domain models within Asset Creation is to assist in determining (and perhaps recording decisions about) which assets should be produced and the range of characteristics they should support. For that reason, domain models focus on describing the commonality and variability among existing and envisioned systems, rather than on describing the systems themselves. The assets that are developed using the domain models are at a lower level of abstraction: they describe or implement (or, in the case of application generators, generate) system work products. The assets often accommodate some degree of variability, but with a principal objective not of describing the variability, but of providing mechanisms to resolve it during Asset Utilization to produce specific system solutions.

Domain models can have broader goals than is implied above. For example, they can serve as a general organizational repository of domain knowledge and experience to support domain evolution and learning. They can thus provide a framework for recording information that has broader organizational purposes than just the production of software assets or systems (e.g., information concerning business planning, project and proposal histories, corporate experience, etc.). The domain models and the general information they contain can themselves be viewed as assets from this perspective. To focus the discussion, however, this section emphasizes the somewhat narrower role that domain models play in support of asset development.

An alternative way of viewing Asset Creation is from the perspective of software life cycle stages or products. This view is orthogonal to the "domain model vs. asset base" perspective described above, in that it applies equally to both domain models and assets. In this view, domain models can

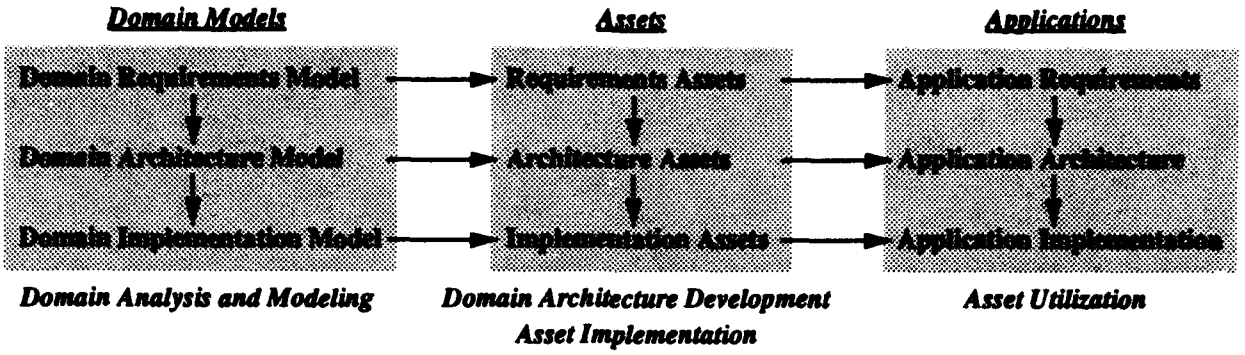

Figure 5: Relationships Among Reuse Products and Processes

be divided logically into separate models that focus on describing the commonality and variability among different classes of software life cycle products within a domain. This approach can be useful because it (a) reflects the natural abstraction layerings among life cycle products, (b) can ease traceability among the models, assets, and existing and future system life cycle artifacts, and (c) can reduce the complexity of the domain modeling process through decomposition. For the purposes of this discussion, the following three general classes of domain models are considered, reflecting the requirements definition and analysis stage, the architecture/design stage, and the implementation stage:

- *Domain requirements models* that describe the overall scope and context of the domain and the range of potential requirements on the operational characteristics of domain applications.
- *Domain architecture models* that describe the range of potential software architectures and designs that can satisfy domain requirements.
- *Domain implementation models* that describe the range of potential implementation assets that can satisfy domain requirements and architecture constraints.

Similarly, the asset base can be viewed in terms of life cycle stages or products, by mapping assets into the conventional life cycle product categories of requirements, architecture/design, and implementation. As with the domain models, there is a cascading of constraints from requirements through architecture to implementation. That is, at each level, assets are constrained by the range of possibilities admitted at the higher level. For example, each implementation asset ideally should satisfy some need identified within one or more architecture assets. Note that implementation assets can extend beyond the scope of operational system software, to include items such as system simulators, test drivers and data, user documentation, and so on.

Figure 5 illustrates the two dimensions of Asset Creation discussed above (i.e., the domain model/asset base view and the software life cycle stage view) by showing, in somewhat simplified form, the relationships among various products. The shaded areas represent domain models, assets, and application products, as seen from left to right. Arranged vertically within each of these shaded areas are the more specialized products that address each of the major life cycle stages discussed above. The arrows in the figure indicate constraints, dependencies, or influences among products. In general, the downward arrows indicate constraints imposed by products at one level that must be satisfied by products at a lower level. The arrows pointing to the right indicate

similar kinds of constraints, and also indicate derivation and usage relationships. For example, an application product may be derived (e.g., instantiated, adapted, generated) from an asset; a generator asset may generate an application product by directly using commonality and variability information encoded in a domain model.

The names at the bottom of each shaded area in Figure 5 indicate which CFRP process categories are principally involved in producing the three major types of products. The various domain and application products and processes are discussed in more detail in the corresponding process category descriptions within this document. For the sake of simplicity, the figure omits a number of details, such as the role of Asset Management and various secondary relationships among products.

As the above discussion of Figure 5 illustrates, there is potentially a very rich web of interrelationships between and among the domain models and assets that should be recorded and managed. For example, it is important to record relationships and constraints such as derivation and traceability (e.g., code asset A satisfies some specific aspect of architecture asset B, and also covers some specific subrange of variability in the domain implementation model) and compatibility or incompatibility (e.g., code asset A will work together with code asset B but not code asset C in the context of architecture asset D).

Such information can be recorded in a variety of ways. The relationships among domain models could be stored in the domain models themselves and/or in some form of higher-level model. Some relationships among assets could be stored within some of the assets themselves (e.g., as part of the architecture assets), but can be viewed more logically as stored within higher-level *asset models* that describe the overall structure of the asset base. Relationships between the domain models and assets (e.g., to characterize the ranges of variability that the asset base addresses) could be stored in the domain models and/or asset models. In any event, the extensive interrelationships among the domain models and assets imply a need for considerable interaction, iteration, and feedback among the Asset Creation process categories (as well as the other Reuse Engineering processes that use Asset Creation products), both as the models and assets are initially developed and as they continually evolve over time.

Note that the distinctions drawn between the different classes of domain models and assets described above, though conceptually straightforward in relation to the Asset Creation process categories, are nevertheless strictly logical in nature. Any particular approach to Asset Creation may partition or organize the models, assets, and processes differently or give them different names, but the approach will likely be mappable, at some level, to the concepts described in this section.

As shown in Figure 6, the inputs to the Asset Creation family include:

- **Assets** relevant to a domain or set of domains, created previously by Asset Creation processes
- **Software Systems** in domains of interest that can impart legacy knowledge about the domains and feed domain analysis or reengineering efforts to produce domain assets
- **Domain Knowledge** that can be imparted in a variety of ways (other than the legacy systems) to provide information about domains
- **Market Forces** characterized by future market trends, competitive developments, new technologies, emerging standards, and other factors that can impact technology and requirements forecasting

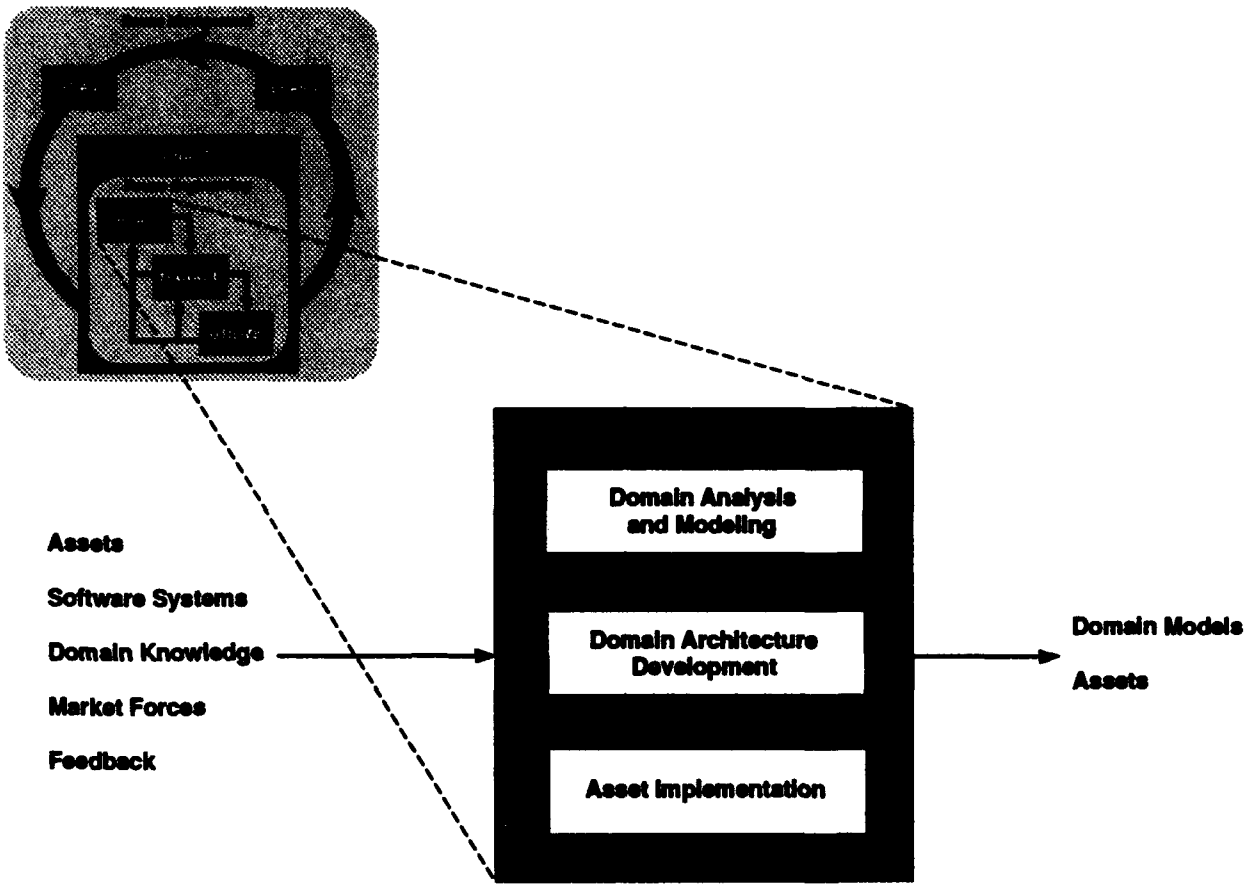

Figure 6: Asset Creation Process Family

- **Feedback** from Asset Management and Asset Utilization processes concerning the assets supplied by Asset Creation

The outputs from the Asset Creation family include:

- **Domain Models** that describe commonality and variability within a domain, from a variety of perspectives
- **Assets** that can be reused to construct application system products within a domain

The subsections that follow describe each of the Asset Creation process categories shown in Figure 6.

3.2.1.1 Domain Analysis and Modeling Process Category

The Domain Analysis and Modeling process category focuses on producing engineering models to support the development of domain assets. Other activities that are sometimes associated with domain analysis, but which focus more on organizational and management-oriented issues (e.g., domain selection), are addressed in the CFRP within Reuse Management process categories such as Assessment and Scoping.

The principal goal of Domain Analysis and Modeling is to develop a set of domain models, such as the domain requirements, architecture, and implementation models discussed above, that can assist in the process of *asset specification* (i.e., deciding which assets to develop, and the range of features or capabilities they should support). Such models focus on describing system characteristics in terms of their commonality and variability within the domain, and they may include a wide variety of information that supports such modeling or is otherwise relevant to asset specification.

To establish the most reusable and effective asset base, domain analysts often must adopt a very broad view of what is relevant to the problem. One technique for scoping the analysis is to consider the domain from the perspective of a variety of different "stakeholders" who each work with the domain in some capacity. Such a stakeholder analysis can help clarify and prioritize the aspects of the domain that are most important; can establish lines of communication with stakeholders that can assist with knowledge acquisition, model validation and evolution, and community buy-in; and can provide a basis for determining how best to model the domain and how to partition the models most effectively. Some typical stakeholders include: the application end users, maintainers, and installers; the application developers, testers, and configuration managers; and the domain analysts and domain engineers themselves.

Although there has been a recent proliferation of domain analysis methods (e.g., [Sof93a, DIS93, PDA91, Sof91c, KCH+90, Sof91a, JHD+90]), there remains little consensus on specific techniques. Domain analysis in general remains a substantial research topic and it will likely be a while before any strong community consensus emerges on domain analysis objectives, scope, approach, and products. However, many existing domain analysis processes have in common the following general activities:

- Reverse engineering,
- Knowledge acquisition,
- Technology and requirements forecasting,
- Domain modeling, and
- Asset specification.

These activities are described in the following paragraphs.

Reverse Engineering

One very important source of information that can be used to construct domain models is the set of legacy systems that is available within the domain. To extract expertise already encoded in these legacy systems, they are often analyzed using reverse engineering and design recovery techniques. These techniques aid in identifying basic domain concepts and vocabulary, commonality and variability in the domain, and, where several versions of the same system are available for analysis, responses to changes in technologies and methodologies over time. Increasing the number of systems analyzed and compared within a domain will likely enrich the domain models and thus increase the overall depth and breadth of applicability of the resultant domain assets.

For the purposes of this discussion, design recovery is considered a subset of reverse engineering. Design recovery methods extract high level design information from existing systems, while other reverse engineering methods extract lower level design and implementation information. All of these methods may, in full, be more applicable to the development of assets that can be reengineered from existing system artifacts, but the methods also clearly apply to domain analysis. Design recovery can be used to discover, validate, and record basic domain concepts and requirements from a user's and customer's perspective. Design recovery can also help to identify issues and alternatives regarding architecture, design, and implementation methods and technology from a developer's perspective. Lower level reverse engineering techniques can similarly be used to extract information about design, implementation, and technology alternatives and can help to reveal the impact such choices have on one another and on issues such as system performance, timing, and sizing.

Knowledge Acquisition

One key objective of domain analysis is to capture and formalize domain knowledge that would otherwise remain only in people's heads. Such knowledge is useful not only because it represents expertise and experience in the domain, but also because it can be used to verify or corroborate the information obtained through the analysis of existing systems. Domain experts can also assist in defining domain abstractions at various levels and in providing domain rationale, examples, and rules of thumb.

Processes to support knowledge acquisition in domain analysis can be adapted from knowledge acquisition techniques used by expert system developers, interviewing techniques used for systems analysis and requirements elicitation, and general methods used for in-depth interviewing in any discipline. Knowledge acquisition remains a craft area that is ripe for innovation in methods and tools, both to promote completeness and consistency of the acquired knowledge and to automatically structure that knowledge in useful ways.

Technology and Requirements Forecasting

In order for reuse to remain viable within typical domains over a period of years so that a return on the investment in asset creation will be fully realized, forecasting of future trends in domain requirements, architectures, and supporting technology is essential. This does not necessarily mean predicting these trends in detail, but being prepared to accommodate changes in a manner that will allow smooth evolution and modernization of assets over time. This forecasting activity can involve consulting domain and technology experts, keeping abreast of evolving standards and underlying technology, and staying directly involved in relevant domain and technology professional communities via conferences, journals, and so on. Technology forecasting can play a key role in domain analysis by identifying strategic technology requirements, and it can also play an important role in identifying technology alternatives and evolution paths for architecture and implementation assets at a more tactical level.

If domain knowledge acquisition is a craft, technology and requirements forecasting qualifies as an art. Short term forecasts of 9 months to two years can often be developed with a reasonable amount of confidence. Long term forecasts of more than two years are more difficult to develop with any confidence, although this can vary from domain to domain.

Domain Modeling

During and after the gathering of domain information by reverse engineering, knowledge acquisition, and forecasting activities (and any other information gathering activities deemed necessary), the information is integrated into a set of domain models that can be used to support asset specification and development. The information gathering activities often produce data in raw form, and this data must be distilled, synthesized, and further analyzed to capture essential domain characteristics. This is usually done in an ad hoc manner because few (if any) general methods exist that support comprehensive model synthesis of this nature. Determining which domain characteristics are "essential" in this context is also generally an ad hoc process, but one good general criterion is importance in the eyes of domain stakeholders.

Domain models bound and scope the domain, capture fundamental domain concepts and vocabulary, and describe the commonality and variability that can exist among elements of the domain. The elements in question here can be wide-ranging; examples include:

- end-user requirements,
- functional features and capabilities,
- architecture and design characteristics,
- implementation algorithms and techniques,
- development processes, methods, and technologies,
- development and operational context,
- user and developer organizations, skills, and expertise, and
- applicable standards.

Some of these elements are associated with particular life cycle stages, and can be described by domain models focusing on those life cycle stages (e.g., the domain requirements, architecture, and implementation models discussed earlier, or other life cycle partitionings, as appropriate). Other elements from the above list have impact across the life cycle or define overall domain context, and could be described in additional models or shared across multiple models. The various models generally include a variety of relationships and constraints among elements (within and possibly across models), and may also include information about how those relationships and constraints vary under differing circumstances.

As with other aspects of Domain Analysis and Modeling, model representation techniques can vary widely, but techniques that readily represent variability are particularly useful. These can include taxonomic modeling, object-oriented modeling, domain-specific language definition, and rule-based modeling. Domain models may also include conventional system modeling representations (e.g., data flow models, entity-relationship models, structure charts) as a context for abstracting and describing commonality and variability.

Domain models can also be annotated with various anecdotal and heuristic information about the domain. This can include information such as design rationale, rules of thumb, examples, and

domain experience gathered during development or maintenance activities. Such information can provide valuable, though informal, advice to asset specifiers and developers. It can also be of considerable value to asset utilizers during Asset Selection, when encoded in a library data model or in individual asset descriptions.

Domain models will ultimately be of little use if they are not validated in some way to establish confidence in their correctness and utility. Processes supporting model validation include walkthroughs, expert and stakeholder reviews, consistency and completeness checking by tools supporting specific representations, and prototyping and trial application of assets derived from the models.

Asset Specification

The asset specification process involves deciding and specifying which assets should be developed, and the range of features or capabilities they should support. One issue to consider in Asset Creation is where the asset specification process resides in relation to other processes. Asset specification could be viewed as a final stage of domain analysis, but could just as readily be viewed as an aspect of the asset development processes. In practice, the asset specification process will likely be very iterative and interact with a number of other processes. It may also be interleaved among a number of processes (e.g., specification of the architecture assets could be part of Domain Analysis and Modeling, but code asset specification may be considered part of Domain Architecture Development because it may not be possible to fully specify the code assets until the architecture assets are developed). For ease of presentation in this document, asset specification is addressed as part of Domain Analysis and Modeling.

Domain models generally describe a broad range of potential asset capabilities, as well as a complex set of constraints and interdependencies. In general, it is neither feasible nor desirable to develop an asset base that accommodates the entire range of possibilities. A number of criteria can be used to decide which asset capabilities and configurations to implement, either immediately or in accordance with some evolution plan. These can include factors such as development cost, number of future applications affected, maintenance impact, overall long-term cost savings, market conditions and customer impact, perceived stability of the technologies or architectures, coherence of sets of assets in addressing common ranges of requirements and architectural constraints, and so on. The result of analyzing the domain models with respect to the chosen criteria is a set of asset specifications that reflect the relevant decisions made about the assets to be developed. These specifications can be encoded within the domain models or asset models, or can exist separately.

In general, it is useful to consider asset specification issues from the perspective of the domain stakeholders discussed above. They will provide several specific contexts for considering cost and benefit and will also open the process to multiple relevant viewpoints.

3.2.1.2 Domain Architecture Development Process Category

The CFRP treats domain architecture assets differently from other assets by addressing their development in a separate process category. The principal reason for this is the important role that domain architectures play in facilitating domain-specific reuse by defining implementation frameworks for constructing systems within a domain. The development of other forms of assets is addressed by the Asset Implementation process category.

The goal of Domain Architecture Development is to produce a set of domain architecture assets that satisfy the architecture asset specifications produced during Domain Analysis and Modeling. These specifications identify which architecture assets will be produced and the variability they will accommodate. The variability encompassed by the architecture assets should be consistent with constraints expressed in the domain architecture model and the requirements assets (Figure 5 illustrates this context). In turn, architecture assets establish a set of *architectural constraints* on potential application architectures and implementation assets (and thus implicitly on application implementations) within the domain.

The domain architecture assets are adaptable software architectures or designs that define the structure of application systems or subsystems within a domain. Each domain architecture may consist of a number of different views capturing a number of different perspectives on the overall architecture. Examples of such views are functional decomposition models, entity-relationship models, dependency graphs, data flow models, control flow models, and state transition models. For the purposes of this discussion, an architecture is defined in terms of the following general constructs:

- a set of software system elements, which may include both processing and data elements
- interfaces for each element
- a set of element-to-element connections, collectively forming interconnection topologies
- the semantics of each connection
  - the meaning of static connections (e.g., between data elements)
  - protocols describing information transfer across dynamic connections, in terms of element interfaces (general classes of protocols include procedure call, pipe, message passing, etc.)

Each architectural view presents some specific filtered subset of this information. In addition, certain architectural traits (e.g., particular connection topologies, interaction patterns, etc.) can be said to represent certain architectural *styles* or *idioms*[1] [Sha91, PW92]. It is generally important to maintain a consistent style within an architecture, at least in the context of a particular view.

The specific approaches that can be taken to develop and represent architectures vary widely, and no broad consensus is evident in this area. However, there has recently been a sharply increased level of activity in this field (e.g., [Cen93c, MG92, PW92, Sof91a, Sha91, Sha89, Bai89a, Kam89]), both in system and domain contexts. In particular, the DoD DSSA and CARDS programs are working towards some level of consensus, at least within limited communities, on architecture representation. Among the many issues that recent work in this area is attempting to address are the qualities of an architecture that make it "good" (e.g., adaptable, extensible, reusable) and the qualities that distinguish a domain architecture from a system architecture. Current approaches to domain architecture development generally tend to mirror system design methodologies, and it is an open issue whether fundamentally different approaches are needed to adequately support domain-specific reuse. As with design methodologies, object-oriented approaches to architecture development are becoming popular, and layering within architectures is another common technique. A recent phenomenon is the emergence in the commercial marketplace of tools that directly support

[1] Not to be confused with CFRP process idioms.

methods for message-based, architecture-driven system design and implementation (e.g., TRW's Universal Network Architecture Services (UNAS) product set).

Reverse engineering of legacy systems to recover and subsequently generalize their designs can play an important role in architecture development. Reverse engineering and design recovery methods can be used at a more detailed level than in domain analysis to identify a software system's modularization, the relationships among the structural elements, data usage and flow patterns, control flow patterns, and scoping information. This information can be used as a basis for defining new architectures and abstracting them to accommodate variability.

In general, there could be many architecture assets for a given domain, reflecting different sets of requirements or addressing different levels of abstraction. For example, within a broadly scoped domain, there may be a high level architecture that describes, rather coarsely, how systems within that domain are constructed. In the context of that architecture, there may be a number of specialized subdomains, and for each of these subdomains, there may be a variety of alternative detailed architectures, each reflecting a different set of implementation technologies, performance requirements, and so on.

Architectural alternatives may be expressed in terms of distinct architectures, or by defining individual architectures that encompass some range of variability. In either approach, the architecture assets may accommodate variation with respect to a number of factors, including the architectural constructs listed above. If the variability within a single architecture becomes unwieldy to represent, creation of separate, alternative architectures may be advisable. One approach to managing complex variations in architecture assets is to use application generators, which are described as part of the Asset Implementation process category below. Generators can encapsulate architectural variability very effectively. For example, a generator can allow a user to interactively configure an application architecture in high-level terms while the generator ensures architecture completeness and consistency at the more intricate, detailed level. In fact, generators can encapsulate an architecture so fully that the user need not make explicit architectural decisions; in such cases, all such decisions can be determined implicitly from application requirements that the user specifies.

The Domain Architecture Development and Asset Implementation processes collectively may record a variety of information about assets as they are developed. This information can include: constraints such as those depicted in Figure 5; asset specifications describing how the assets satisfy those constraints (including descriptions of asset variability); development trade-offs, decisions, and rationale; instructions and heuristics for applying the assets, and various other asset characteristics, interdependencies and constraints; Logically, such information is stored in *asset models* that describe the overall structure of the asset base, at a higher level of abstraction than the assets themselves. For example, such models can capture the various dependency and coherence relationships that define implementation alternatives associated with an architecture (e.g., by identifying implementation assets that satisfy particular requirements or architectural constraints, defining dependencies and compatibility constraints among implementation assets, and describing how asset choices may impact tailoring decisions for other assets). Physically, such information might not be stored in separate asset models, per se (e.g., some of it could be embedded within the architecture assets themselves), but it is useful to consider the information to be logically distinct from the assets.

3.2.1.3 Asset Implementation Process Category

The goal of Asset Implementation is to produce the non-architecture (requirements and implementation) assets in the asset base. These assets are developed to satisfy asset specifications, produced during Domain Analysis and Modeling, that identify which assets will be produced and the range of variability they will accommodate. This variability must be consistent with constraints expressed in the domain requirements and implementation models and the architecture assets (Figure 5 illustrates this context). Requirements assets establish a set of *requirements constraints* on application requirements and on architecture and implementation assets within the domain. Implementation assets establish a set of *implementation constraints* on application implementations.

This category encompasses two major asset implementation perspectives:

- Production of *software components* that have the same general form as application products and can be reused directly (perhaps with some tailoring) to produce application products in whole or part. The term "software component" is used broadly here to include not only code components, but a variety of other life-cycle products, including reusable requirements, detailed designs, test cases, documentation, and so on.
- Production of *application generator* tools that accept specifications of desired application characteristics and generate application life-cycle products (of any type) that reflect those characteristics.

Some high-level decisions can be made within Reuse Planning processes (e.g., Infrastructure Planning) with regard to the general approaches, technologies, or methods that will applied in developing certain classes of assets. For example, strategic decisions can be made to emphasize a generator-based approach in general. However, Asset Implementation processes should assume responsibility for making final decisions on such issues for individual assets, based on finer-grained considerations than are apparent at the strategic planning level. A variety of factors, such as technology maturity, domain maturity and stability, technology compatibility among assets and application products, and initial cost vs. long-term cost savings, can enter into such decisions. Consideration of these factors may sometimes result in decisions not to implement certain assets, in which case custom development or "non-reuse-based" reengineering of corresponding application products will be required.

The remainder of this section provides an overview of software component and application generator asset development processes.

Software Component Development

The goal of software component development is to develop a set of reusable software components (in accordance with the broad definition above) that satisfy particular asset specifications. This is generally done either by developing the components from scratch or through "reuse-based" reengineering of legacy artifacts. In either case, the processes employed should emphasize good software engineering practices and principles such as separation of concerns and information hiding and should observe general guidelines for reusability and quality. Techniques used to incorporate variability into the assets will vary greatly depending on the methods and tools (e.g., programming

languages) used. The analysis, design, and coding guidelines that are available for different methods or programming languages may be used to help guide the component development processes. It may also be useful to consider implementing functions, data, and modularizations that exceed the needs of some systems or accommodate more variability than is immediately required, in order to build in flexibility and future growth. Also, in addition to simply building the components, software component development processes should produce supporting material such as reuse and maintenance documentation, and possibly items such as test drivers and test data if no separate assets of that nature are specified for development.

A distinction is made above between "reuse-based" and "non-reuse-based" reengineering. Much of the reengineering that is done today is non-reuse-based, in the sense that it is code-oriented and "point-to-point". That is, it focuses on reengineering code from one application so that it operates within another, usually very similar, application, without much consideration for how the code might apply to additional applications. This is often done to convert an application from one programming language to another without significantly affecting data structures, modularization, or control flow, and it is usually done using a simplistic language-to-language translation approach.

Increasingly, however, reengineering is being done in a more reuse-based manner, to improve the reusability (and quality, maintainability, etc.) of the legacy components so that they apply not only to a single new system, but also to a variety of other potential systems. This approach often involves substantial restructuring of the legacy system and is ideally done in the context of domain models and architectures that may have been derived in part through design recovery of the system. This approach is initially more costly than the "point-to-point" approach, but substantial cost savings can be realized if a series of similar, but significantly varying, systems will be built in the domain. In a system maintenance and evolution context, the various versions of a system can be viewed as a series of similar systems, and a reuse-based reengineering approach can thus serve as a sound basis for a long-term system maintenance and evolution strategy [Big89, Bas90, Vir92b]. One valid approach is to apply reuse-based reengineering techniques incrementally, targetting one key subsystem after another over time.

Application Generator Development

The goal of application generator development is to produce tools that accept specifications of desired properties of a target application and generate application life-cycle products that reflect the specified properties. The objective is to allow an application engineer to specify "what" is desired (in terms that are native to the domain and familiar to the engineer) rather than detailing "how" the desired effect is to be achieved, as is the case in more conventional development. This "what" orientation can be termed requirements-based. Two examples of the domain-specific nature of generator input languages are:

- A generator supporting interactive construction of graphical user interfaces might allow the direct specification of user interface abstractions such as menus, popups, buttons, etc.
- A generator addressing the chemical process control domain might support direct specification of control law concepts, symbols, and terminology.

Application generator development depends on the results of other Asset Creation processes. The language used to specify the required properties of the generated products should be derived from

domain models (particularly the domain vocabulary). Such languages may have been developed during domain modeling to characterize appropriate aspects of commonality and variability within the domain, or they may need to be developed or refined as part of the application generator development process. Also, knowledge about the form and variability of the application products to be generated must be either embedded within the generator assets during development or obtained from external sources, such as domain models or other assets during generator execution. Code generators often presume a tightly constrained range of architectural contexts, so generators are often closely associated with particular architecture assets.

Complete applications are generally composed from code assets, newly-developed custom code components, and generated subsystems. Some application generators, known as composition tools, provide automated support for part or all of this composition process by composing and configuring individual components in accordance with an application engineer's specifications to form (sub)systems that meet application needs.

The specific generation techniques and technologies that are available and understood within an organization are key factors in determining the development processes that will be employed to implement a generator. Techniques used may include textual or graphical language design, language translation techniques, meta-generation techniques (i.e., generation of a generator using a higher-level tool), knowledge-based and expert system techniques, and so on. Although somewhat specialized, development of an application generator is similar to development of any software system. Therefore, software engineering principles, reuse principles, sound design methods, validation, and testing are all vital to generator development.

3.2.2 Asset Management Process Family

The goal of the Asset Management process family is to establish managed collections of reusable assets, give Asset Utilization processes access to those assets, and promote and support the reuse of the assets. The Asset Management family functions in a brokerage role between Asset Creation and Asset Utilization processes, in that it establishes a kind of marketplace supporting and encouraging asset distribution and creation, based on availability and needs.

The Asset Management processes fall into two general classes: processes that focus on acquiring, installing, and evaluating individual assets in a library, and processes that focus on developing and operating libraries that house collections of assets, provide access to those assets, and support their utilization.

Asset Management may appear to overlap in some ways with the Reuse Management idiom, particularly with regard to reuse infrastructure. Asset Management processes and capabilities can be viewed as just another form of infrastructure serving the needs of the more product-oriented Asset Creation and Asset Utilization processes. This issue is accentuated by the fact that "organizational assets" (e.g, plans, policies, processes, history) are generally treated as part of the reuse infrastructure, while library support technology is generally considered the province of Asset Management. The following general guidelines can help clarify these matters:

- There can be logically distinct Asset Management processes within a reuse program, each of which manages assets addressing distinct organizational needs. For example, one set of Asset

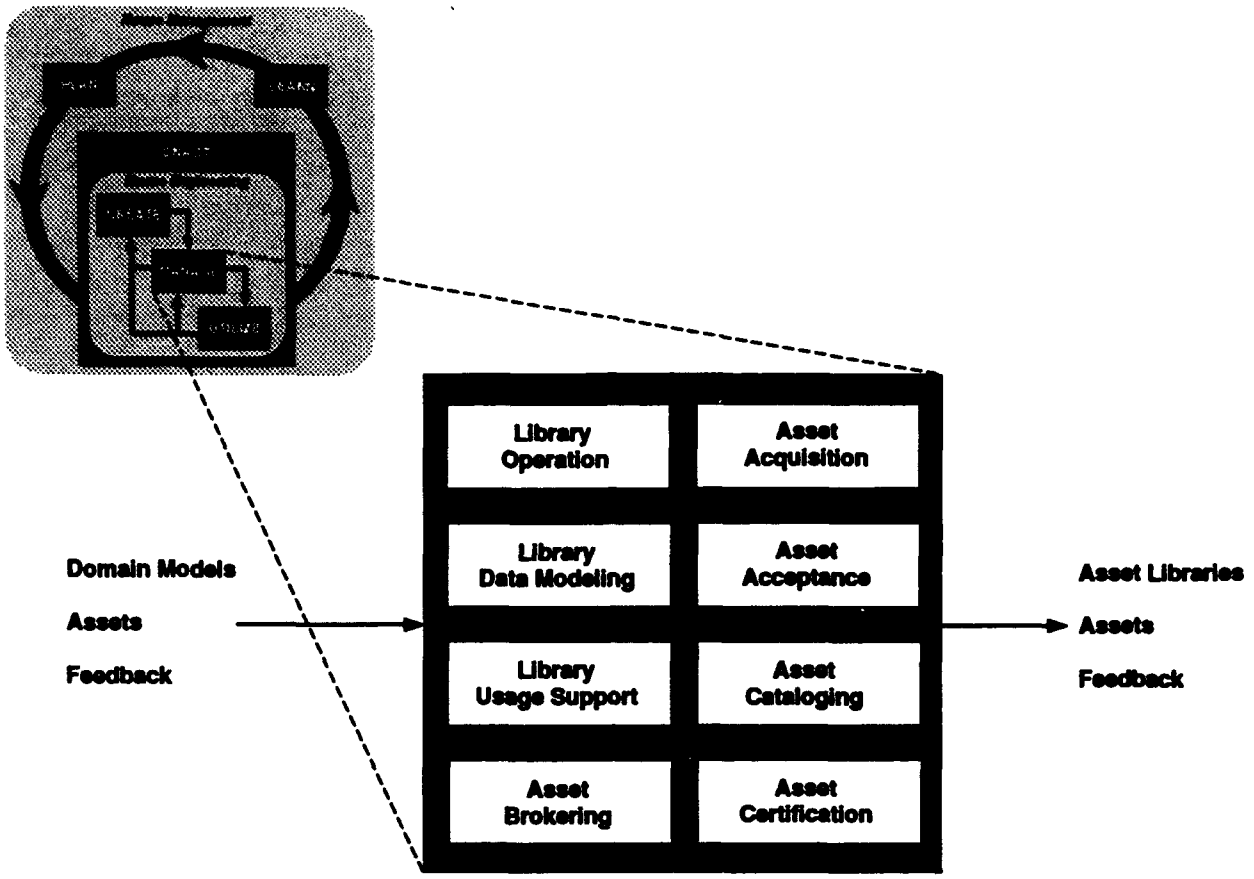

Figure 7: Asset Management Process Family

Management processes (enacted as shown in the canonic CFRP in Figure 1) can manage assets relevant to production of application products within an organization's product line. Meanwhile, another set of Asset Management processes (recursively embedded within Infrastructure Implementation processes – see Appendix A for more information on recursion) can manage "organizational assets" relevant to planning and other management functions.

- Asset Management addresses the selection and support of technology that is inherently asset-, library-, or domain-specific, or tailored to be so. Realistically, however, such activity is generally influenced to some degree by the overall infrastructure policy and technology planning constraints imposed by Reuse Management processes.

Although the term "library" is used throughout this section (and, in fact, throughout the document) to refer to entities that house managed asset collections, this term is not meant (except where noted otherwise) to connote any particular technological approach. A library need not even be automated to effectively manage a collection of assets and serve a useful mediator role between Asset Creation and Asset Utilization processes. The STARS Asset Library Open Architecture Framework (ALOAF) document [Sof92] includes an Asset Library Reference Model that describes the general characteristics of asset libraries without any strong technological bias (although it does assume some degree of automation), and consultation of that reference model may be useful (though not required) when reading the process category descriptions below.

As shown in Figure 7, the inputs to the Asset Management family include:

- **Domain Models** that describe commonality and variability within domains, from a variety of perspectives
- **Assets** relevant to a domain or set of domains
- **Feedback** from Asset Utilization processes concerning the assets and services provided by Asset Management

The outputs from the Asset Management family include:

- **Asset Libraries** that each provide an organizing scheme and descriptive information for a collection of assets, along with a set of services and capabilities for identifying, evaluating, and reusing the assets
- **Assets**, managed in the libraries, that can be reused to construct application systems in relevant domains
- **Feedback** concerning the assets supplied by Asset Creation processes

The subsections that follow describe each of the Asset Management process categories shown in Figure 7.

3.2.2.1 Library Operation Process Category

The goal of Library Operation processes is to ensure the availability and accessibility of the library and its associated assets for Asset Utilization. This can involve a variety of activities, such as:

- Administration and operation of the physical library facility,
- Acquisition, installation, operation, and maintenance of computer system hardware and software, including the basic library software if that is not already part of the established reuse infrastructure,
- Library access control and security,
- Configuration management of library contents,
- Periodic archiving and backup of library contents, and
- Support for interoperation with other libraries.

These activities are strictly operational in nature, and organizations may consider many or all of them to be aspects of Infrastructure Implementation. However, there are aspects of these activities that relate specifically to asset libraries and merit attention from an Asset Management perspective. For example:

- **Access Control**
  Access to asset libraries and individual assets or sets of assets may need to be restricted. Access restrictions can be levied to enforce a wide variety of constraints, including data rights (e.g., license) restrictions, government security laws or policies, and company proprietary policies. Such restrictions can be applied to individual users or groups of users, based strictly on identity or on attributes such as user role. Some typical library user roles, corresponding to some of the Asset Management and Asset Utilization processes, include: *library data modeler*, *asset cataloger*, *asset certifier*, *library operator*, *asset utilizer*, and *asset broker*. Where access restrictions are needed, libraries should support strong user identification and authentication policies and mechanisms, in concert with administrative procedures for managing information about users.

- **Configuration Management**
  Version and configuration control is an important and potentially complex set of processes within a domain-specific reuse-based life cycle. All assets, ranging from high level requirements and architecture assets to the lowest-level implementation assets, must be kept mutually consistent to ensure the consistency and integrity of applications built from the assets. Assets are typically longer-lived than system artifacts, with the potential for creation of multiple versions or variations of the same asset, each of which must be maintained concurrently. The consistency of relationships between each particular variation of an asset and other library assets (and their variations) also needs to be maintained. Because these processes manage information about assets, they may be tightly intertwined with (or considered part of) Library Data Modeling and Asset Cataloging processes.

- **Library Interoperation**
  Some libraries require procedures and mechanisms for interoperating with other asset libraries. Such interoperation may take a variety of forms, including accessing assets in the other libraries in a direct and "seamless" manner, importing assets from those libraries for local installation, or exporting local assets to those libraries. Before such interoperation can be effective, the policies, data models, and interfaces of the participating libraries may need to be evaluated. If the policies and data models of the libraries are similar and they support standard library interfaces, interoperation may simply be a matter of maintaining network connectivity. However, as the library policies, data models, or interfaces diverge, the potential for interoperability between the libraries will become progressively lower, ranging from the ability to automate the exchange of assets, to maintaining electronic catalogs (sometimes called "Yellow Pages") of external libraries and assets, to simply publishing paper catalogs and requesting that assets be sent from one library to another on tangible media.

3.2.2.2 Library Data Modeling Process Category

The goal of Library Data Modeling processes is to develop a data model for describing assets within a library. This *library data model* filters and synthesizes the domain models, asset models, and assets produced by Asset Creation to capture the structure of the information needed specifically to support Asset Utilization. The specific approach taken to produce the library data model is thus highly dependent on the characteristics of the Asset Creation products and on the objectives of the library in supporting Asset Utilization. If the library's objectives are to provide a basic search and retrieval capability for individual assets, it should suffice for the library data model to include mainly taxonomic information derived from one or more domain models. If the objectives

are more ambitious, to include more direct support for system composition, the library data model must integrate additional elements of the domain models, elements of asset models describing various asset interrelationships, and possibly elements of specific assets (e.g., architecture assets). In addition to incorporating information produced by Asset Creation processes, the library data model also codifies information that specifically addresses Asset Management needs. For example, a model could include data elements to support library and asset metrics collection, record asset certification information, and capture asset submittal and user feedback data.

One of the basic elements of any library data model is the classification scheme, or taxonomy, that is used to partition library assets into different classes or categories. Classification knowledge can be represented in a variety of ways, including indexing schemes, simple hierarchies, faceted schemes, object-oriented class hierarchies, and semantic networks [FG90]. One powerful technique that can be used within a library data model is the concept of multiple classification schemes that provide library users with alternative taxonomic views of the domain and thus provide alternative browsing and querying strategies for locating assets. Taxonomic models can be especially useful when they are combined with non-taxonomic (e.g., architectural) information to produce integrated library data models enabling navigation from multiple perspectives.

3.2.2.3 Library Usage Support Process Category

In addition to simply operating a library and making assets available within it, Asset Management should provide a set of library services that anticipate and address specific asset utilizer needs. These services support usage of the library and of individual assets. Some of these services could be considered extensions of the *Library Operation* activities discussed above, but are distinguished from those activities because of their direct support for library and asset usage. Examples of these kinds of services are:

- The production of conventional catalogs, in paper or electronic form, that describe the assets in the library and enable prospective users to quickly assess library contents without direct library access,
- The collection and generation of asset data in a number of different formats, including a variety of media (e.g., paper, on-line files, video) and data representations (e.g., plain ASCII text, Postscript, executables, specific tool data formats),
- The distribution of assets to library customers in a variety of formats via a number of different physical media, in accordance with distribution restrictions,
- The operation of electronic and/or telephone hot lines to accept and resolve user requests, suggestions, complaints, and other feedback, and
- The operation of electronic bulletin boards and mail systems to enable sharing of information among library users.

Another key aspect of Library Usage Support is the selection (or development), tailoring, and integration of library tools that are specific to the domain or to particular types of assets. This activity, often closely associated with Library Data Modeling, involves determining users' library interaction needs in terms of the kinds of assets and descriptive information that will be in the

library and the kinds of tools that will be needed to inspect, evaluate, and utilize the assets and associated information. Such tools can support activities such as: viewing assets, understanding assets, testing assets, executing assets for evaluation, providing views of asset interrelationships and architectural structures, extracting sets of closely related assets, composing assets to build applications, and so on. In general, there should be coordination between Reuse Management and Reuse Engineering processes to ensure that tools that prove useful within a domain context are absorbed into the established reuse infrastructure over time, and perhaps broadened or generalized to apply across domains.

An often critical form of Library Usage Support is direct, personal assistance to users in understanding library capabilities, finding and assessing relevant assets, and utilizing those assets for desired purposes. These kinds of consultation services can be likened to the role of the traditional librarian in conventional book libraries. Such services can be automated to some degree, but are usually most effective when rendered in person by knowledgeable individuals. This person-to-person approach can effectively lower the technological barriers to reuse that typical library software presents to many users.

A related form of services is asset subscription, which allows users that "subscribe" to a particular asset to be informed of all changes to that asset (and, optionally, to be given updated versions of it) as it evolves. The kinds of asset changes about which a user can be notified include identification or resolution of errors, changes in classification, development of new variations, and addition of new usage history data.

3.2.2.4 *Asset Brokering Process Category*

Without planning and management that anticipates asset needs, Asset Creation efforts may not produce assets that satisfy the needs of Asset Utilization efforts. Furthermore, even when assets are available and appropriate for application system needs, prospective asset utilizers could be unaware of the assets, or could be aware but unmotivated to consider them for reuse. Careful planning at the reuse program level can address these problems to some degree by clearly specifying producer and consumer responsibilities and establishing global reuse promotion and incentive efforts.

However, these program-level measures are only part of the formula to adequately motivate creation and utilization of appropriate assets. Such measures generally need to be augmented with more localized efforts that take into account observations about (a) needs that are not being met by asset creators and (b) assets that are not being exploited by asset utilizers. These efforts involve activities to monitor asset flows, interactions, and feedback among Asset Creation, Management, and Utilization processes, and apply that knowledge to work proactively with all concerned parties to improve effectiveness in particular areas. Such activities can be viewed as *asset brokering* processes. Some general examples of asset brokering approaches are:

- Based on observed asset utilization and application construction patterns and feedback, determine areas where additional assets addressing some portion of a domain would yield substantial benefit to application engineers. Notify relevant asset creators and encourage them to create new assets addressing the identified needs. Alternatively, identify new external sources of relevant assets that can be acquired and brought under Asset Management control (or simply accessed directly by asset utilizers in their external form) to address application

needs.

- Based on asset utilization patterns and feedback, determine which known assets are generally not being reused (even though they nominally address application needs), and why they are not being reused. If the problem is that the asset utilizers perceive shortcomings in the assets, inform asset creators of the perceived shortcomings and encourage modification of the assets or other appropriate actions to remedy the problem. If the problem is that the asset utilizers are unaware of the assets or have not properly assessed their benefits, work with the utilizers to improve their awareness or knowledge of the assets.
- Monitor asset creation and application engineering activity within and across various organizations. Identify assets or potential assets in existing or even new (i.e., previously unmanaged) domains that can be applied to ongoing or future application engineering efforts. Among various projects and organizations, encourage acquisition and management of new assets, creation of new assets from promising applications, and utilization of those assets within the application engineering efforts.

Of course, such activity must be coordinated with Reuse Management processes to ensure that the work being encouraged is within the scope of existing reuse program plans or that the plans are modified appropriately to accommodate any additional work.

3.2.2.5 Asset Acquisition Process Category

The principal goal of Asset Acquisition processes is to obtain assets from external sources (including other asset libraries) to support Asset Utilization activities. Asset Acquisition obtains assets that appear to be good candidates for inclusion in an asset library, based on the domain and architectural requirements embodied in the library data model (or potential extensions to that model). Such candidate assets can then be assessed more strictly by Asset Acceptance processes to determine whether or not they should be made available to utilizers.

In an ideal situation, Asset Acquisition is coordinated directly with Asset Creation processes designed to produce domain models and assets that comprehensively address some set of Asset Utilization (and thus application system) needs. In such cases, Asset Acquisition typically involves little more than obtaining the assets from the suppliers so that they can be made available to utilizers through the asset library. In less ideal situations, where there is less direct coordination with Asset Creation processes or those processes do not perform so comprehensive a role, Asset Acquisition involves locating and acquiring needed assets through other means.

In these latter situations, external sources (possibly identified by Reuse Planning or Asset Brokering processes) should be exploited. Such sources can include external asset libraries, projects developing systems within a relevant domain, commercial or government off-the-shelf products, external individuals or organizations that voluntarily submit candidate assets, and so on. Acquiring assets from such sources can present a variety of problems, however, including difficulty in obtaining enough information about the assets to reliably evaluate and catalog them. For example, when assets are submitted voluntarily, repeated requests may be necessary to obtain sufficent descriptive information. Similar in some ways is the acquisition of assets from an external library, wherein differences in the data models between the local and external libraries must be resolved in order to obtain adequate descriptions of the assets. The degree to which acquisition of assets from external

libraries can be automated is directly related to the homogeneity of the local and external library data models and the degree of interoperability among the libraries, as discussed in [Sof92].

In addition to supporting Asset Utilization processes, Asset Acquisition may also support Asset Creation activities by obtaining candidate assets that come "close enough" to satisfying the domain and architectural requirements to justify the cost of modifying the assets or extending the requirements. Asset Acquisition may also obtain legacy software that does not satisfy the requirements but could be useful input to Asset Creation processes to help evolve the domain models and assets. To support Asset Creation in these ways, the Asset Acquisition and Asset Creation processes should be coordinated to ensure that the software products being acquired are relevant to the creation activities.

3.2.2.6 Asset Acceptance Process Category

The goal of Asset Acceptance processes is to ensure that an asset that is a candidate for inclusion in a library satisfies relevant policy, legal, and domain-specific constraints.

The purpose of library policy constraints is typically to ensure that assets in a library satisfy at least minimal criteria for quality and suitability for use in Asset Utilization activities. Such constraints are generally imposed internally by an organization. They are often expressed in terms of requirements on the descriptive information that accompanies a candidate asset, and sometimes in terms of requirements on the asset itself. Requirements on the descriptive information are often expressed in terms of an asset submittal form that defines the minimal set of information that must be submitted with an asset in order for it to be cataloged in the library. The submitted information may then be subjected to further criteria to determine if the asset is acceptable for inclusion in the library, such as criteria on the format of the asset, its source language, its degree of documentation, and so on. The asset itself may also undergo examination; for example, it may be subjected to metrics analysis or testing to assess asset quality relative to some basic criteria.

Legal constraints are generally imposed by external organizations and focus primarily on restricting the access, distribution, or use of an asset, independent of its perceived technical quality or suitability. Consideration of legal constraints is particularly important for assets acquired from external sources such as public, government-supported, or commercial asset libraries, or sometimes even other projects within the same organization. In any of these cases, licenses, patents, copyrights, distribution rights, liability requirements, royalties, and other related issues may complicate or restrict the ability to reuse a particular asset.

Domain-specific asset acceptance constraints are derived from the domain models (and perhaps some assets, such as architecture assets) produced by Asset Creation processes. These constraints establish qualification criteria with respect to the functions, interfaces, interactions, side effects, performance, etc. (sometimes called the "form, fit, and function") of candidate assets relative to domain needs. Assets must satisfy some set of these criteria in order to be considered acceptable for reuse in some context within the domain and thus qualify for inclusion in the library.

3.2.2.7 Asset Cataloging Process Category

The goal of Asset Cataloging processes is to incorporate accepted assets into a library to make

them accessible to library users. Asset Cataloging is generally broken down into the following three steps:

- *Asset classification* is the process of determining where an asset belongs within the library classification scheme that is created by the Library Data Modeling processes described above. Once the appropriate place(s) in the scheme is/are found, the asset is said to be classified. For example, classification within a faceted scheme might involve identifying a facet term, or set of terms, for the asset.
- *Asset description* is the process of creating, capturing, or adapting all the other information that is needed to describe the asset in the context of the library data model, once the asset has been classified. Asset Description can involve a wide variety of activities to collect and record needed information, such as capturing general or domain-specific asset properties; generating objective measures of asset size, reusability, quality, and complexity; and transcribing and interpreting asset submittal information. Asset Description may also involve identifying dependencies on and relationships to other assets, as determined from domain or asset models developed during Asset Creation. Some of the collected descriptive information may need to be tested for correctness or validated against library standards, to the extent that the information is not checked during the Asset Acceptance process.
- *Asset installation* is the process of installing the classified and described asset in the library system. This involves capturing the asset and its descriptive information in some kind of data base or other persistent store, linking this information to the library data model and other library information as appropriate, and performing other environment-specific operations to make the asset available to asset utilizers.

3.2.2.8 Asset Certification Process Category

The ultimate goal of Asset Certification processes is to guarantee that the assets satisfy their requirements without error. From a practical standpoint, Asset Certification is an iterative evaluation process that gradually approaches but may not achieve that ultimate goal. Various categories of certification can be defined, each associated with particular sets of certification criteria. To be assigned a particular certification category, assets must satisfy the corresponding criteria. These categories can be arranged in hierarchical "levels", wherein each level represents a greater degree of certification than those below it, or they can be considered more independent, each representing some particular important quality for an asset to achieve; some mixture of both is also possible. In the more hierarchical approach, an asset at a particular level is considered more trusted than the assets at lower levels in the sense that there is increased confidence that it meets its requirements without error.

Asset Certification typically begins after Asset Acceptance and Asset Cataloging have occurred. However, it is generally an ongoing process that continues while an asset is available through, and evolves within, the library. The certification categories that an asset has been assigned at any given time are typically included as part of the asset description.

Some of the criteria that can be applied during Asset Certification may also be appropriate during Asset Acceptance; the specific processes in which particular criteria are applied will reflect organization preferences and needs. Certification criteria need not be strictly formal and measurable to

be useful, although greater care needs to be taken in applying criteria that are relatively informal or heuristic in nature. Examples of general certification criteria include:

- Completeness
  - Does the asset include associated life-cycle products, such as requirements, design, code, test cases and data, analysis and verification artifacts, and documentation?
  - Are the asset's dependencies and (in)compatibilities documented?
  - Is the asset accompanied by other products on which it is dependent?
- Reliability and quality
  - How good are the results of asset testing, inspection, and review?
  - How good is the asset's development and/or runtime performance?
  - Has the asset undergone formal verification? Of what nature?
  - Does the asset come with results of independent review, evaluation, or accreditation? How good are the results?
- Ease of reuse
  - Does the asset come with reuse instructions or reuse support tools?
  - What is the asset's reusability, understandability, portability, complexity, maintainability, debuggability, etc., as determined via metrics, inspection, or other methods?
- Adherence to guidelines and standards
  - Which development or reusability guidelines does the asset adhere to?
  - Which standard or required processes, methods, languages, or tools were used to develop or verify the asset?
  - Which standard or required architectures, interfaces, etc., does the asset satisfy?
- Asset history
  - Does the asset come with a documented history of user experiences?
  - Does the asset come with a documented development and maintenance history?
- Maintenance and support
  - Does the asset come with maintenance or support agreements?
  - What is the extent and quality of the asset support?
- Known problems or limitations
  - Does the asset have a documented set of known bugs or faults?
  - Does the asset have critical limitations in functionality, performance, compatibility, etc.?
- Legal/contractual constraints
  - Is the asset known to have legal or contractual constraints that limit its accessibility, applicability, or (re)distribution?

3.2.3 Asset Utilization Process Family

The goal of the Asset Utilization process family is to construct new application products (e.g., requirements, design, code, tests, documentation) using previously developed assets. Asset Utilization encompasses the reuse-based aspects of the process that is becoming increasingly known as "application engineering". Asset Utilization generally involves determining a set of criteria to use in selecting assets for reuse, identifying suitable candidate assets in the context of those criteria, selecting and tailoring assets to meet the criteria, and integrating the tailored assets with the target application.

The domain and asset models developed in the Asset Creation family support Asset Utilization processes by encoding a variety of domain information that can be reused when building systems within a domain. The Asset Management family makes this information directly available to Asset Utilization processes through the library data model, the associated asset descriptions and assets, and a collection of library capabilities or services for accessing and processing the information. This information and corresponding set of services can support Asset Utilization in a number of ways by, for example:

- Providing access to general system requirements and characteristics within the domain to establish a context for identifying and understanding domain assets;
- Providing access to architecture assets that can be used as architectural frameworks for constructing target applications;
- Providing access to specific implementation assets that satisfy certain domain, architectural, or implementation constraints, and describing how those assets can vary to satisfy a range of such constraints; and
- Providing guidance or assistance (at varying levels of automation) in finding, evaluating, or applying assets to construct target application products.

The Asset Utilization process categories were chosen to be independent of any particular method or life-cycle process. That is, these categories are at a level of abstraction that enables them to be integrated with any method to make that method more reuse-based without substantially disrupting or compromising its other inherent qualities. However, as Asset Utilization processes become integrated with particular methods and supported by particular technologies, these should be coordinated with corresponding Asset Creation methods and tools to ensure a consistent strategy. One key issue to acknowledge in this area because of its pervasive impact is the dichotomy between *system composition* and *system generation* techniques.

These two techniques differ in a number of ways, and their differences are particularly apparent in the Asset Creation and Asset Utilization families. In Asset Utilization, differences between these techniques impact several categories, with the strongest impact appearing in Asset Tailoring. This is because a generator asset is a tool that generates system artifacts that eventually get integrated with the target application, and to generate those artifacts, the asset must be tailored through some form of specification language. The specification language is typically non-procedural in nature (though not necessarily textual) and is generally domain-specific. That is, the specifications that can be expressed in the language specify requirements or constraints on the generated system in

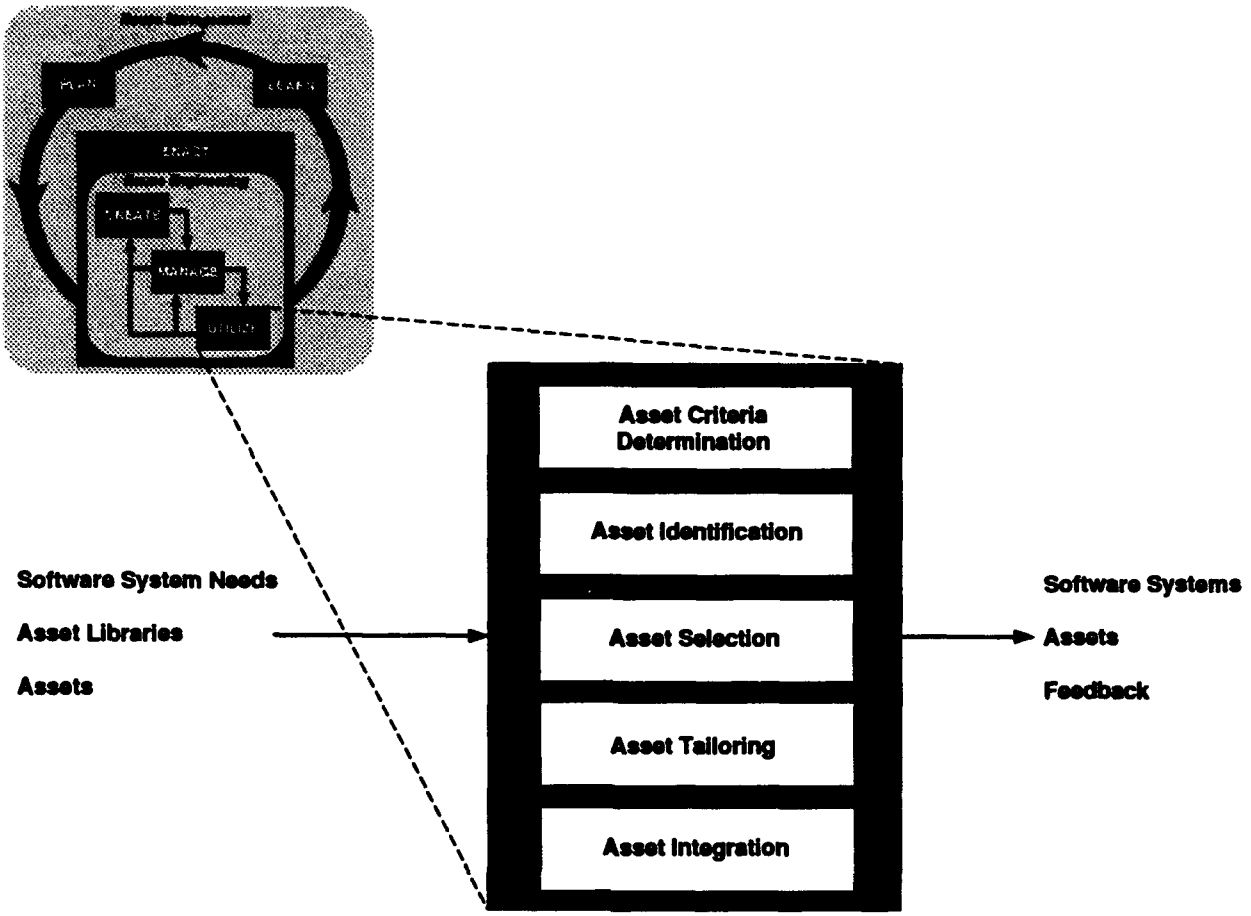

Figure 8: Asset Utilization Process Family

domain-specific terms. This generally requires a much more extensive and sophisticated tailoring process than is the the case when tailoring an Ada generic package via generic parameters, for example. Generators are also different because a product of the asset, rather than the asset itself, is integrated with the target application, unlike conventional software components.

The viewpoint taken within the Asset Utilization category descriptions that follow is that system composition is generally considered the principal utilization technique and system generation is in some sense subordinate in this respect. This is because, even in circumstances where generation is used heavily, complete application systems are typically composed from generated subsystems, software component assets, and some components developed from scratch. The situation where an application is generated in its entirety can be viewed as a degenerate case of the composition process, where there is only one element to compose to form the application. As a result of this somewhat composition-centered viewpoint, the category descriptions focus principally on the composition perspective, but do point out ways in which generation differs where appropriate.

As shown in Figure 8, the inputs to the Asset Utilization family include:

- **Software System Needs** that identify requirements for desired application systems in relevant domains; these may include reuse-related requirements or criteria addressing specific

asset characteristics

- **Asset Libraries** that each provide an organizing scheme and descriptive information for a collection of assets, along with a set of services and capabilities for identifying, evaluating, and reusing the assets
- **Assets**, managed in the libraries, that can be reused to construct application systems in relevant domains

The outputs from the Asset Utilization family include:

- **Software Systems** and other application products that are constructed from assets, in whole or in part
- **Assets** that are improved or synthesized during Asset Utilization and made available to Asset Management and Asset Creation so that they can be considered for incorporation into the asset base
- **Feedback** concerning the assets and services supplied by Asset Creation and Asset Management processes

The subsections that follow describe each of the Asset Utilization process categories shown in Figure 8. Note that, although the Asset Utilization process categories are applicable to the reuse of assets that span the application life cycle, most of the examples in the category descriptions that follow are presented in terms of code assets for convenience. Also, for brevity, the term "library data model" as used throughout the remainder of this section refers to both the data model and associated asset descriptions, unless otherwise noted.

3.2.3.1 Asset Criteria Determination Process Category

The goal of Asset Criteria Determination processes is to establish a set of constraints or criteria for identifying, selecting, and tailoring assets that can be applied to the construction of application products. Since assets can address the entire system life cycle, the asset criteria may apply to a variety of different kinds of assets that address various aspects of the application. For the purposes of this discussion, the criteria apply to requirements assets, architecture assets, and implementation assets, as discussed in Section 3.2.1.

The asset criteria generally are derived from three sources: (a) the domain and asset information encoded in the library data model, (b) the overall user requirements or constraints imposed on the target application, and (c) the life cycle products produced for the application, each of which constrains products produced further downstream in the life cycle. These sources of information apply to Asset Criteria Determination in the following ways:

- Overall application requirements are used to derive criteria that are applied to the selection of domain requirements assets, which are then reused to construct detailed system requirements.
- Detailed system requirements are used to derive criteria that are applied to the selection of domain architecture assets, which are then reused to construct system designs.

- Detailed system requirements and designs are used to derive criteria that are applied to the selection of domain implementation assets, which are then reused to construct system implementations.

As the above implies, it is in general neither possible nor desirable to determine the criteria for all assets at once. An iterative or phased approach is appropriate, wherein criteria are first determined for requirements assets, which ultimately cascade down into architecture and implementation asset criteria via reuse at the requirements and architecture levels, respectively. This is not to imply, however, that the process is necessarily rigid or highly sequential. Iterative prototyping and assessment of results at all stages of this process can be a key factor in producing "good" asset criteria and thus in eventually producing "good" system products that satisfy customer needs.

Regardless of the specific steps that are taken, the Asset Criteria Determination process ultimately produces a set of criteria to be used in identifying, selecting, and tailoring assets that address one or more aspects of an application. Each of the criteria produced is considered to be in one of two categories:

- *Shallow* criteria that are applied by Asset Identification processes to identify a set of candidate assets for further evaluation. In general, shallow criteria are those which can be applied through direct appeal to the library data model via library browsing and query operations. Examples of such criteria are:
  - Assets within a particular category in a domain taxonomy,
  - Assets that can implement a particular architectural element,
  - Assets that have been certified at a particular level,
  - Assets written in a particular programming language, and
  - Assets that work in a particular development or target environment.
- *Deep* criteria that are applied by Asset Selection processes to evaluate candidate assets and select particular assets to be reused. Deep criteria cannot be readily applied by inspecting only the library data model. Instead, they focus on asset characteristics requiring more in-depth inspection and analysis of individual assets. Examples of such criteria are:
  - Assets that operate properly in a particular testbed environment,
  - Assets exhibiting a particular range of performance given certain test data,
  - Assets that meet subjective architectural quality standards, and
  - Assets that are sufficiently adaptable and reusable, as determined through trial reuse.

These categories of criteria are difficult to characterize in any greater detail because of the fact that they are defined in terms of the contents of the library data model, which can vary widely in breadth and depth from library to library. Some further insight into the nature of these criteria is provided in the Asset Identification and Asset Selection process category descriptions below.

3.2.3.2 Asset Identification Process Category

The goal of Asset Identification processes is to identify a set of candidate assets that meet shallow asset criteria. These candidate assets are provided to Asset Selection processes that assess the assets relative to deeper criteria.

As noted above, shallow asset criteria can be applied directly by browsing or querying the library data model. The specific techniques used to browse and query the data model are highly dependent on the structure and representation of the model and the support tools that are provided by Asset Management to access the model. Common approaches include facet- and keyword-based queries, hierarchical and hypertext-style browsing, and dialog-based browsing wherein the user is guided through the library by answering questions that elicit the asset criteria. These approaches can be complementary within the same library. For example, the understanding of the data model acquired while browsing can help the engineer to formulate queries; conversely, the results of queries may help direct the engineer's attention towards portions of the model that are best browsed (possibly with on-line assistance) to achieve full understanding of certain assets and their interrelationships.

If the library data model is relatively complete, in that it reflects thorough domain and asset models, it can be quite an effective vehicle for identifying a narrow, focused, and complementary set of assets that will require little additional evaluation to determine if they are suitable for the target application. That is, in such cases, the "shallow" criteria used to identify assets can in fact be quite deep, reflecting the depth and richness of the model. Libraries based on such deep models enable fine-grained resolution of asset information and may thus be amenable to tools that perform automatic composition of assets to produce application products, based on the specification of relatively deep asset criteria by the user. In such cases, Asset Identification and Asset Selection processes may both be fully encapsulated by the tool.

It may often be the case, however, that the library data model is relatively superficial, in that it does not encode very thorough domain requirements or architecture information or enables only very coarse classification of assets. Under such circumstances, Asset Identification can be challenging and will often yield too few or too many assets for further consideration. In these cases, it can be useful to broaden or narrow the asset criteria systematically to eventually yield a candidate asset set of sufficient and manageable size.

At times, analysis of the library data model may indicate that other domains or other libraries need to be browsed or queried to obtain useful assets. In such cases, library interoperability comes into play. Other domain-specific libraries, which may be locally or remotely located, will need to be browsed and/or queried in a manner similar to the initial library, even though they may have substantially different data models. A high level of interoperability between libraries will allow engineers to perform these activities without being strongly aware that such differences exist or that the libraries may be widely distributed on a network.

3.2.3.3 Asset Selection Process Category

The goal of Asset Selection processes is to further evaluate the assets identified by Asset Identification processes to select those which best meet target system needs and should thus be tailored for integration into the application. The asset evaluation is done by applying "deep" asset criteria that are not readily applicable through direct appeal to the library data model. The criteria may

vary greatly, depending on the breadth and depth of the data model; they may initially require basic judgements about high-level asset capabilities due to the lack of resolution of the data model, or they may focus at the outset on low-level deciding factors such as performance and level of assurance.

Asset evaluation can involve a wide variety of activities, depending on the asset criteria and the nature of the assets. These activities can take place entirely in the context of the library if the library is well equipped with supporting tools and data, or they can be performed by extracting assets from the library and evaluating them outside the library context. Asset evaluation often begins with a thorough analysis of an asset's description, as well as analyses of the asset itself and of related assets and other supporting data. The asset description information to be analyzed might include asset attributes and abstracts, detailed asset interface descriptions, usage histories and problem reports, metrics and quality data, and a variety of other items. The assets themselves can be inspected in their raw source form (e.g., the source code of an Ada package specification) or can be viewed using alternative methods with the assistance of appropriate tools, such as hypertext systems, design tools, graphics tools, and word processors. The dynamic behavior of assets can also be evaluated. Inspection of behavioral specifications may be sufficient for this, but other approaches include the use of tools to simulate the behavior of the asset under realistic conditions or the use of test harnesses to actually execute the asset with representative data to provide live, hands-on feedback. The latter approach may be useful for better understanding both the functional and non-functional characteristics of the asset. In particular, non-functional characteristics such as performance are often addressed unconvincingly, if at all, in the static asset description, and are best understood through hands-on use.

Assets are also often evaluated in terms of their quality and assurance. This may involve the inspection or the on-line computation of a variety of metrics for an asset, and may also involve the tracing and inspection of corroborating assurance information, such as test results, formal specifications, formal or informal proofs, and the results of certification or accreditation processes which the asset (in the context of systems or subsystems of which it was a part) may have undergone.

Another key factor in evaluating assets is their ease of use, both from an end user and a reuser perspective. The reuser, in particular, must be concerned with the variability the asset will accommodate, its tailorability, its ease of modification (if modification is necessary), and its compatibility with other assets for purposes of integration. Such judgements can sometimes be made through inspection or simple trial usage of the asset, but may also sometimes require more full-fledged attempts to tailor the asset and integrate it with other portions of the application. This may call for some iterative prototyping activity among the Asset Selection, Asset Tailoring, and Asset Integration processes.

After undergoing the above evaluation activities (and possibly more), an asset may be judged to provide an appropriate set of capabilities; provide them in an appropriate manner with adequate documentation, assurance, and performance; be sufficiently variable, tailorable, and integratable; and meet any other criteria that may be applied, including subjective or intuitive judgements made by application engineers based on their individual experience with the domain. If only one asset satisfies the criteria in a particular functional area, it will be selected for tailoring and integration with the application. If there is more than one such asset, additional criteria (often subjective) are generally applied to select the one "best" asset. atively, more than one can be selected and each can be tailored and integrated with the app n to assess which is best through practical

experience.

If no assets fully satisfy the criteria, an assessment must be made of whether any particular asset is "close enough" to the criteria to justify its reuse. Alternatively, the criteria may be reassessed and adjusted to better accommodate the assets, thus emphasizing a "reuse first" approach in which application requirements are allowed to be compromised (within limits) to favor a reuse-based solution. An organization may establish economic criteria for making such judgements, based on economic models that weigh the cost of modifying or otherwise adapting reusable assets (and possibly fielding systems that don't fully meet a priori requirements) against the cost of implementing system functionality from scratch or reengineering it from existing systems.

3.2.3.4 Asset Tailoring Process Category

The goal of Asset Tailoring processes is to customize assets that have been selected for reuse (or use them in customized ways) so that they satisfy target system requirements. Assets typically require at least some tailoring during reuse. Tailoring generally comes in two forms, either or both of which may be applied to any given asset:

Anticipated Target system needs lie within the range of variability anticipated for an asset during Asset Creation; the asset encapsulates the variability through some set of *tailoring interfaces* (such as parameters); these interfaces can be used to resolve the variability to meet target system needs.

Unanticipated Target system needs lie outside the range of variability anticipated for an asset during Asset Creation (e.g., there is a need for new or alternative features where no variability was anticipated); the asset thus provides no relevant tailoring interfaces that can be applied; the asset is modified to address the unanticipated target system needs.

To perform *anticipated* tailoring, an engineer must understand the range of variability an asset may accommodate and how the asset's tailoring interfaces are used to select among the variations. This information should be included in the library in the form of "reuse instructions" for the asset, which may be augmented by examples. Parameterization (interpreted broadly) is the mechanism most commonly used for anticipated tailoring. Examples of parameterization include:

- run-time parameters that are passed to the asset procedurally or via messages during system execution,
- specifications, macros, data files, or command-line arguments that are interpreted at run-time to produce desired behavior (e.g., initialization files, document style sheets),
- compile-time parameters that are passed to the asset to produce a system-specific instantiation of the asset (e.g., Ada-style generics),
- installation parameters that control system-specific configuration of the asset (e.g., variables controlling conditional compilation), and
- specification of product requirements, expressed in some domain-specific language, that are given to an application generator to generate the desired products.

In addition to parameterization, another technique that can be used for anticipated tailoring is hand modification of the asset in accordance with precise instructions. An asset that can be tailored in this manner is typically called a *template.*

Note that the tailoring of application generator assets is treated here as a very sophisticated form of parameterization, wherein the entire specification given to the generator is considered to be a parameter or set of parameters that controls the characteristics of the generated products. In some cases, the generated products themselves may also need to be tailored in anticipated ways, such as through installation parameters or run-time customization parameters.

Unanticipated tailoring is more of an ad hoc process in which the engineer assesses the asset's shortcomings relative to system needs and then employs whichever strategies are appropriate to tailor the asset to those needs. This often involves hand modification of the asset to add desired features or remove undesired features. Modifications may also be needed to address issues such as performance, environment compatibility, and safety, reliability, or other quality factors.

For generator assets, unanticipated tailoring typically involves modifying the generated products to meet some system requirement unforeseen by the asset creators. This can be effective, but should be pursued with great caution due to the typically poor understandability and modifiability of the generated products and due to the fact that the modifications will have to be redone any time the products are regenerated. An alternative approach is to modify the generator itself to accommodate greater variability or to satisfy some specific system requirement. Ideally, this should be done by notifying asset creators (possibly including commercial vendors) that a need exists, so that they can modify the generator and related assets to benefit other application engineering efforts.

Some experimentation may be appropriate while tailoring assets using any of the above techniques, to ensure that the tailoring is done most effectively, particularly if the relevant target system requirements are not well understood in advance. At times, the results of such experimentation may affect decisions made during Asset Selection, causing some iteration between the selection and tailoring processes to occur. In the worst case, the full implications of an asset's limitations may not become clear until unanticipated tailoring of the asset is undertaken. At that point it may be appropriate to revisit the decision of whether to reuse the asset, adjust the asset criteria to accommodate the asset's limitations or admit a broader set of assets, or develop the desired capability from scratch, depending on a variety of technical and economic considerations.

3.2.3.5 Asset Integration Process Category

The goal of the Asset Integration processes is to ensure that assets that have been selected for use in an application system and tailored to meet system needs are properly integrated with other system elements. Many Asset Tailoring activities are often considered aspects of system integration, in part because some assets' *tailoring* interfaces are made available through the same mechanism as their *integration* interfaces (e.g., run-time parameters in procedure calls or messages). However, for the purposes of this discussion, asset integration is the process of making tailored assets work with other system elements in the context of a system architecture, and is logically distinct from tailoring. While tailoring focuses more on the adaptation issues local to a particular asset, integration addresses adaptation and consistency issues global to an entire system or subsystem. In this view, the tailoring process can be subservient to the integration process in the sense that integration may require several iterative refinements of tailoring to ensure that all system needs are being met.

Even in relatively mature reuse environments, integration often involves the development of integration modules (sometimes called "glue code") to allow system components to interoperate when asset tailoring is inappropriate or insufficient for that purpose. One of the most commonplace integration strategies is *encapsulation*. In this approach, an asset that does not present the desired interfaces to its would-be clients is encapsulated by code that does present the desired interfaces and also transforms the data passed through those interfaces into (and out of) formats that the embedded asset can understand and process appropriately.

In mature reuse environments, system integration is often done in the context of a domain architecture that has been tailored to reflect target system needs and therefore provides a framework for integrating the tailored implementation assets, newly developed modules, and other legacy components in accordance with well-defined interfaces. In such cases, integration can be straightforward and require little glue code or other interface matching techniques. In fact, in these circumstances, the processes to identify, select, tailor, and integrate assets are all highly automatable, and tools may hide or blur these processes or the distinctions between them.

A somewhat more common situation is that there exist portions of a domain for which there are good architectures and implementation assets, and other portions of the domain for which such assets are not available or do not match target system needs very well. Such situations can occur when domain-specific reuse is being practiced in subdomains of a larger domain in which new systems are being developed or legacy systems are being evolved in otherwise traditional ways. In such cases, problems that arise in integrating the reuse-based and non-reuse-based portions of the system can be addressed using encapsulation techniques. An alternative approach, in the legacy system context, is to reengineer the non-reuse-based portions of the system to accommodate the interfaces in the reuse-based portions. Such reengineering may, but need not, be done in the context of an domain analysis or architecture development effort addressing the entire domain; if it is, it should be considered an Asset Creation effort.

Another common situation is that a system needs to be built in a domain for which there are no architecture assets, but there are a number of implementation assets available (each with idiosyncratic interfaces) that perform desired functions. In such cases, a system-specific design can be developed that incorporates the needed functions and the available assets can be encapsulated or modified as appropriate to work within that design.

4 Summary

Section 3 describes the STARS CFRP in significant detail. It does this in a methodical way, by describing the CFRP hierarchically in terms of the process idioms, their constituent process families, and the process categories within the families. To facilitate this approach, the CFRP is described principally from the perspective of the "canonic" CFRP configuration depicted in Figure 1. Appendix A supplements this view by describing the flexible process modeling features of the CFRP that support construction of alternative CFRP-consistent process configurations.

In general, Section 3 and Appendix A focus on detailed aspects of CFRP elements or composition techniques, rather than offering a more global view of the fundamental concepts, or *themes*, inherent in the CFRP. These key themes are listed below for reader reflection. They are divided into themes about the nature of reuse as it is expressed in the CFRP, and themes about the CFRP model itself and its particular qualities and characteristics, which are specifically relevant to how it is applied.

The themes about reuse as expressed in the CFRP are:

- A reuse-based approach to software engineering should be driven by well-defined processes that are repeatable and can be evolved in manageable ways.
- Software reuse has both management and engineering dimensions, whose key activities, and their interrelationships, are captured in the CFRP Reuse Management and Reuse Engineering idioms and their constituent process families.
- The CFRP process categories provide a definition of the specific activities involved in a process-driven, domain-specific reuse-based approach to software engineering.
- Reuse should be applied as a "first principle". That is, reusable products should always be considered as the basis for work before creating new products. In addition, experiences, processes, and workproducts should always be recorded for learning, and the workproducts should be designed to facilitate reuse.
- Learning and managed change, based on measurement, history, and innovation, are essential to reuse.
- Technical, organizational, and educational infrastructure is essential to reuse, and must be designed and managed to support it.
- A domain-specific, architecture-driven approach to reuse will yield the greatest reuse impact, and is thus important to address from both an engineering and a management perspective.
- The asset producer, broker, and consumer roles are important, separable aspects of reuse that form distinctive patterns of activity within the Reuse Engineering idiom.

The themes about the CFRP and its applicability are:

- The CFRP is generic with respect to domains, technologies, organizational structures, management styles, and economic sectors.

- The CFRP addresses only reuse processes, and these processes must be integrated with overall planning and engineering practices to be effective.
- The CFRP provides a process modeling language featuring composition techniques that support construction of complex, reuse-oriented process configurations.
- The CFRP is highly scalable. In general, it can be applied in an organization of any size, at any organizational level.
- The CFRP is a domain model and high level process architecture for the reuse process domain. It provides a basis for the analysis of reuse processes (and associated products) and the definition of reusable process assets.

In the companion volume to this document, *STARS Conceptual Framework for Reuse Processes (CFRP), Volume II: Application*, guidance is provided for applying the CFRP to support various forms of reuse planning. The document offers additional insight into the CFRP and its associated themes, primarily by putting the CFRP into a more concrete context and showing how it can be used to address practical concerns. STARS strongly encourages you to read the CFRP Application document to gain this additional insight and to begin applying the CFRP in your own organization.

A CFRP Composition Techniques

Section 3 of this document focuses on describing the individual CFRP process elements (the reuse process idioms, families, and categories). These individual elements offer significant insight into reuse processes, but they gain synergistic value when viewed as modular building blocks that can be used to construct a wide variety of reuse-specific process configurations reflecting different planning levels, organizational structures, and interaction patterns. The CFRP is described in Section 3 primarily in the context of one such configuration, the "canonic" CFRP configuration depicted in Figure 1. However, many other configurations are possible.

To support the construction of process configurations, the CFRP includes not only the process idioms and families already described, but also a set of "composition" rules or techniques that can be used to connect the idioms and families together in a variety of ways. These techniques provide a flexible and scalable composition approach enabling CFRP models to capture aspects of reuse-based engineering practice not easily described with traditional software life cycle models, as discussed below.

A full understanding of how to apply the CFRP for planning, process engineering, or evaluation and comparison depends on an understanding of the CFRP composition techniques. These techniques may be of particular interest both to process engineers interested in process configuration and composition and to planners interested in organizational design and interaction.

The CFRP Application document discusses the CFRP process modeling techniques of tailoring, integration, and instantiation, as well as composition. The composition techniques are considered an intrinsic part of the CFRP model, and they are thus described in this document as part of the CFRP definition. The other techniques, though important in applying the CFRP, are not considered intrinsic to the CFRP model, and are thus described in the CFRP Application document.

The following subsections describe the CFRP composition techniques of *linkage*, *recursion*, and *overlapping* in significant detail. In general, the techniques are illustrated by realistic examples showing how they can be useful.

A.1 Linkage

To model potentially complex interaction patterns among reuse projects in CFRP configurations, the CFRP allows for one-to-many, many-to-one, and many-to-many linkages among process families. These techniques are described below, with illustrative examples:

- **One-to-many mapping** Figure 9a depicts a situation in which single Asset Creation and Asset Management projects have a set of multiple Asset Utilization "clients" for the reusable assets developed. This is, in fact, almost a defining case for reuse, where the intention is to create assets that can be used across multiple systems. In modeling this situation, the CFRP encourages explicit documentation of the intended scope of applicability for the assets being developed. Rather than designing for reusability in the abstract, asset creators can design to an expected set of target systems. This configuration could apply to, for example, a single organization performing product-line management by creating domain assets relevant to the

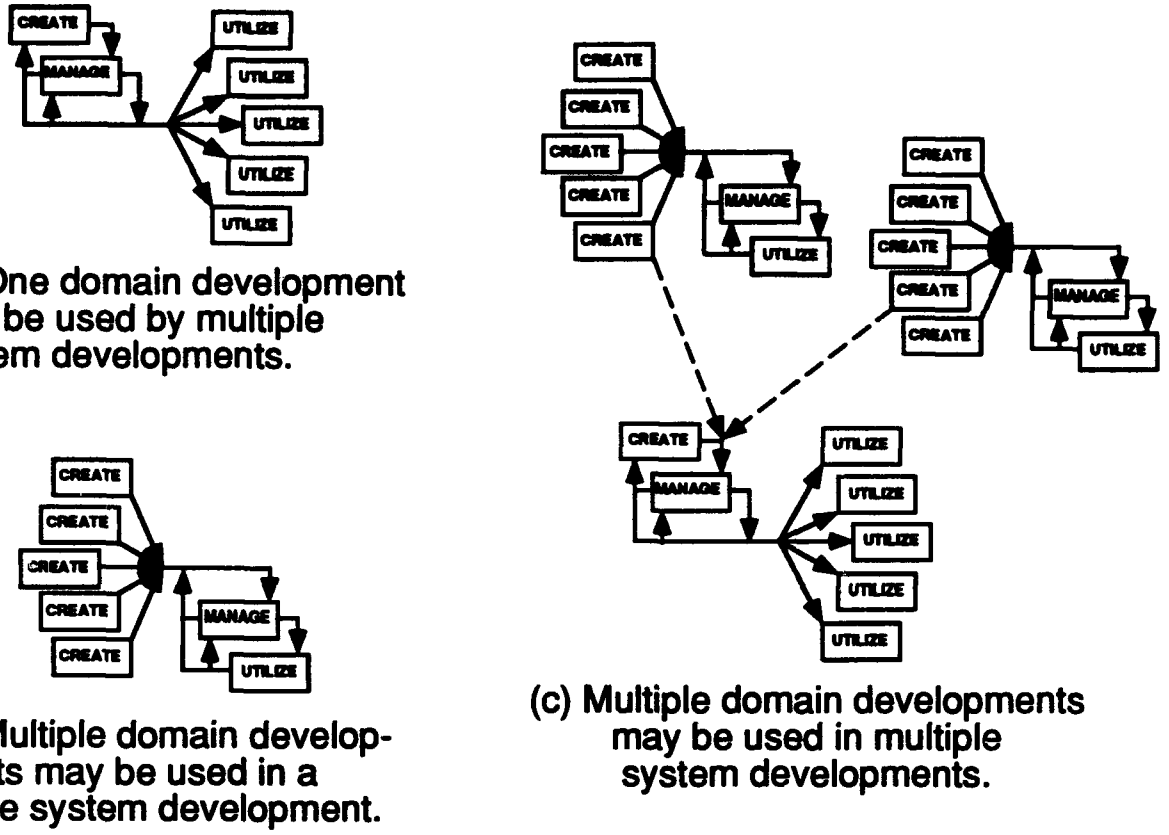

Figure 9: Linkage of Reuse Engineering Processes

product line, managing those domain assets, and building a family of products in the product line using the managed assets. (Nobeltech provides a real-life example of this approach with its shipboard command and control systems [Rat91].) Alternatively, the same configuration could apply to a software vendor marketing assets in a particular domain to a set of external clients.

- **Many-to-one mapping** Figure 9b depicts a situation in which a single Asset Utilization project integrates assets that are managed by a single Asset Management project, but developed by a number of separate Asset Creation projects. This configuration is typical of large-scale systems, where system integrators need to merge assets from multiple domains to implement and maintain a single complex system or family of systems. Boeing's use of subcontractors for commercial avionics provides a real-life example. In the figure, the separate Asset Creation projects could belong to distinct subcontractors. This configuration emphasizes the need for reuse technology, design methods, and infrastructure that allow assets from multiple Asset Creation projects to be managed in common collections and readily integrated. It also points to the need for investment in standard architectures and interfaces to facilitate system composition and integration.

- **Many-to-many mapping** Both examples described above assume the presence of a single Asset Management project in the configuration. In fact, just as there may be multiple Asset Creation and Asset Utilization projects within the scope of a reuse program, there may be multiple Asset Management projects as well. The rationale for such a configuration becomes

apparent in a realistically complex picture of a reuse program spanning multiple domains and several system development efforts, as depicted in Figure 9c. This situation suggests an open marketplace of asset producers, brokers, and consumers. Each Asset Management project receives assets from multiple Asset Creation projects, and two Asset Creation projects provide assets to multiple Asset Management projects. (The dashed lines represent assets moving across organizational boundaries.) The lower "cluster" in the configuration can represent an organization integrating in-house assets with external assets to create a product line. A configuration not shown, but equally possible, would involve the Asset Utilization projects drawing from multiple Asset Management projects across organizational boundaries, as well.

This diagram serves to illustrate that the brokerage role of Asset Management can be indispensable in a multi-domain, multi-project, multi-organization marketplace. Without a separately configurable Asset Management capability, the asset creators or the asset utilizers in this scenario would bear the burden not only of performing basic Asset Management functions, but also of managing multiple, redundant lines of communication to related projects. The Asset Management projects here serve as central focal points for managing these complex interactions. This diagram also suggests that reuse programs will have an ongoing need for negotiating the import and export of assets into and out of the program scope.

This subsection has illustrated one-to-many, many-to-one and many-to-many relationships among Reuse Engineering families. Similar linkages are possible among Reuse Management families; however, the real-world situations in which such modeling constructs would be useful are less intuitive and are not discussed here.

A.2 Recursion

The linkage relationships discussed above involve multiple CFRP families at similar functional and organizational levels. These CFRP families are connected with other families in the same idiom or with families in other, similar idioms. These linked process families are distinct activities coordinated via the peer-to-peer exchange of information. That is, in general, there is no hierarchical interaction or control among the elements.

A different CFRP composition technique enables the modeling of hierarchical relationships among CFRP elements. This technique, termed "recursion" (a special use of the term in the CFRP context), allows CFRP idioms to be nested or embedded within a process family in another idiom. The embedded idiom in a recursion relationship is termed the "child" idiom, and the embedding family (or the idiom of which it is a member) is termed the "parent". The intended semantics of recursion is that enacting the processes of a parent family involves enacting processes of the embedded idiom as sub-processes.

There are twelve distinct recursion relations possible in the CFRP (by embedding each idiom respectively in any of the three Reuse Management families or in any of the three Reuse Engineering families). Some of these relations are more meaningful in practice than others, and each implies a unique set of issues. The following paragraphs discuss a selected subset of these recursion relations.

The canonic CFRP configuration that is used as a reference throughout Section 3 is a result of the recursion relation depicted in Figure 10a. In this configuration, in which a Reuse Engineering

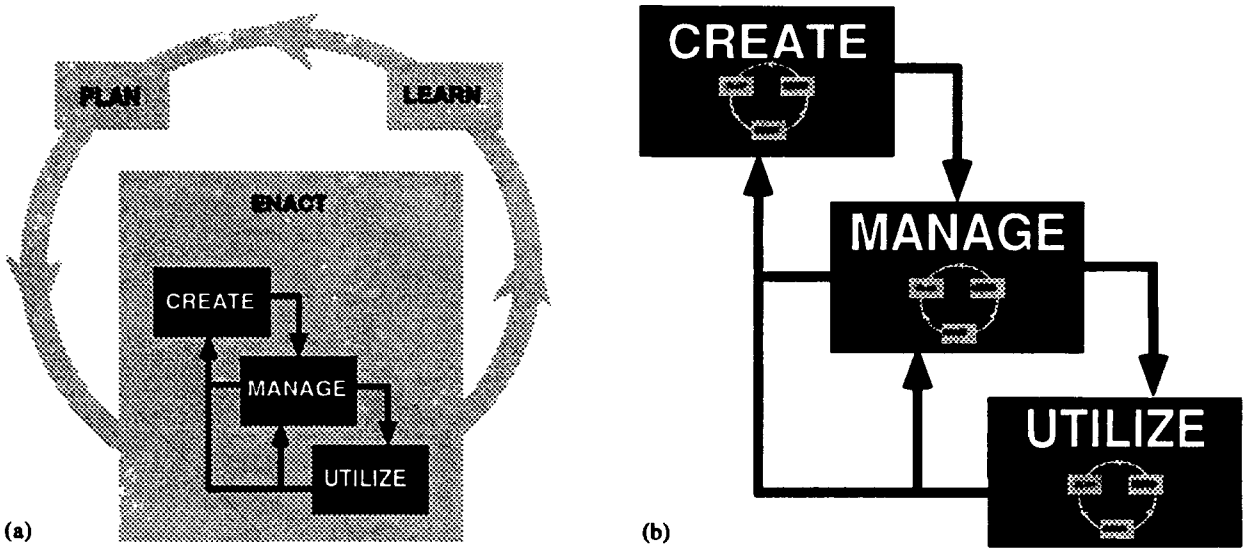

Figure 10: Mutual Recursion of Reuse Management and Reuse Engineering

idiom is embedded within the Reuse Enactment family of a parent Reuse Management idiom, Reuse Management acts as a regulator or overseer of Reuse Engineering activities. This configuration is generally interpreted in this document as representing a reuse program that operates as follows: Reuse Planning scopes the Reuse Engineering effort and plans the processes and interconnections of the Reuse Engineering projects; Reuse Enactment embeds the projects and adds a number of project management functions, such as metrics collection and project monitoring and redirection; Reuse Learning collects and generates lessons learned, evaluates performance relative to objectives, and abstracts processes and methods to improve the next reuse program cycle.

Recursion of Reuse Management within the various families of Reuse Engineering is depicted in Figure 10b. Such recursion generally implies introduction of a different level of planning and decision-making, often at a different organizational level. In this example, the parent families represent the processes of particular reuse projects, with the three inner Plan-Enact-Learn loops representing learning-oriented approaches to individual project tasks. The inner learning loop in Asset Creation, for example, could represent an engineer planning, performing, and improving his own reuse methods on a specific project task. An outer learning loop surrounding the three Reuse Engineering families (as in Figure 10a) would represent a reuse program that focuses more on managing the overall projects and the interactions among them, at a higher organizational level.

Recursion of Reuse Management within the Reuse Enactment family of a parent Reuse Management idiom (shown in Figure 11a) enables modeling of nested reuse programs reflecting hierarchical scopes of planning within an organization. Such recursion can represent (a) levels of planning and decision-making responsibility, reflecting organizational structure, and (b) differences in planning timescales as one ascends the hierarchy, with lower-level planning loops addressing short-cycle planning, and higher-level loops addressing longer, strategic cycles involving perhaps ten- to twenty-year projections. Reuse planners, rather than making arbitrary assumptions about organizational structure, can use the CFRP in this way to model the structural levels that appear relevant to reuse within the organization. The nested Plan-Enact-Learn loops can be visualized as sets of interlocking gears that transmit change at a rate appropriate to the given organizational level or

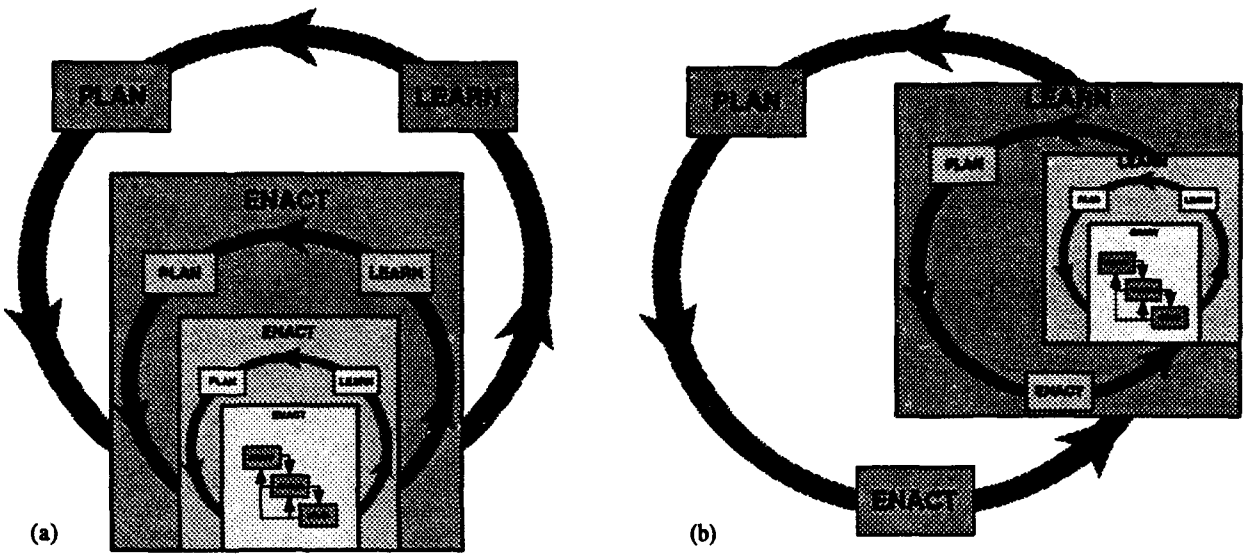

Figure 11: Recursion of Reuse Management within Reuse Management

scope. The managed approach to change inherent in the Reuse Learning family allows potential changes to percolate up the hierarchy of the organization as well as down.

Several CFRP process category descriptions in Section 3 implicitly suggest opportunities for the use of recursion. For example, within an Innovation Exploration activity in Reuse Learning, a Reuse Engineering activity might be established to conduct controlled engineering experiments. This could be modeled as an embedded Reuse Engineering idiom within the Reuse Learning family. Alternatively, if the experimentation involved a series of iterations or cycles (independent of the longer cycle of the parent Reuse Management idiom), this situation could be modeled as an embedded Reuse Management idiom within Reuse Learning. The Reuse Engineering idiom could then be embedded within the lower-level Reuse Management idiom. Figure 11b shows a similar, but more complex situation, in which an additional Reuse Management layer is embedded within Reuse Learning. This illustrates, for example, that Innovation Exploration activities may be multiply nested, reflecting layers of abstraction that occur naturally in R&D work.

Certain configurations, such as recursion of Reuse Management within the Reuse Planning family, seem to threaten an infinite regression, a non-converging series of planning activities. (What does it mean to plan the Planning, and where does such planning terminate?) Since such problems can occur in real-life planning situations, however, it is interesting to note that the CFRP model has predictive value in anticipating these potential problem areas.

A.3 Overlapping

Another important way that process idioms can be composed is through "overlapping", where a process family within one idiom simultaneously serves in a different family role within a separate idiom. Overlapping does not mean the mere sharing of two distinct functions by a single individual or group within an organization, but rather one specific activity simultaneously filling two independent functional roles in the context of logically separate idioms. As with recursion, there are a

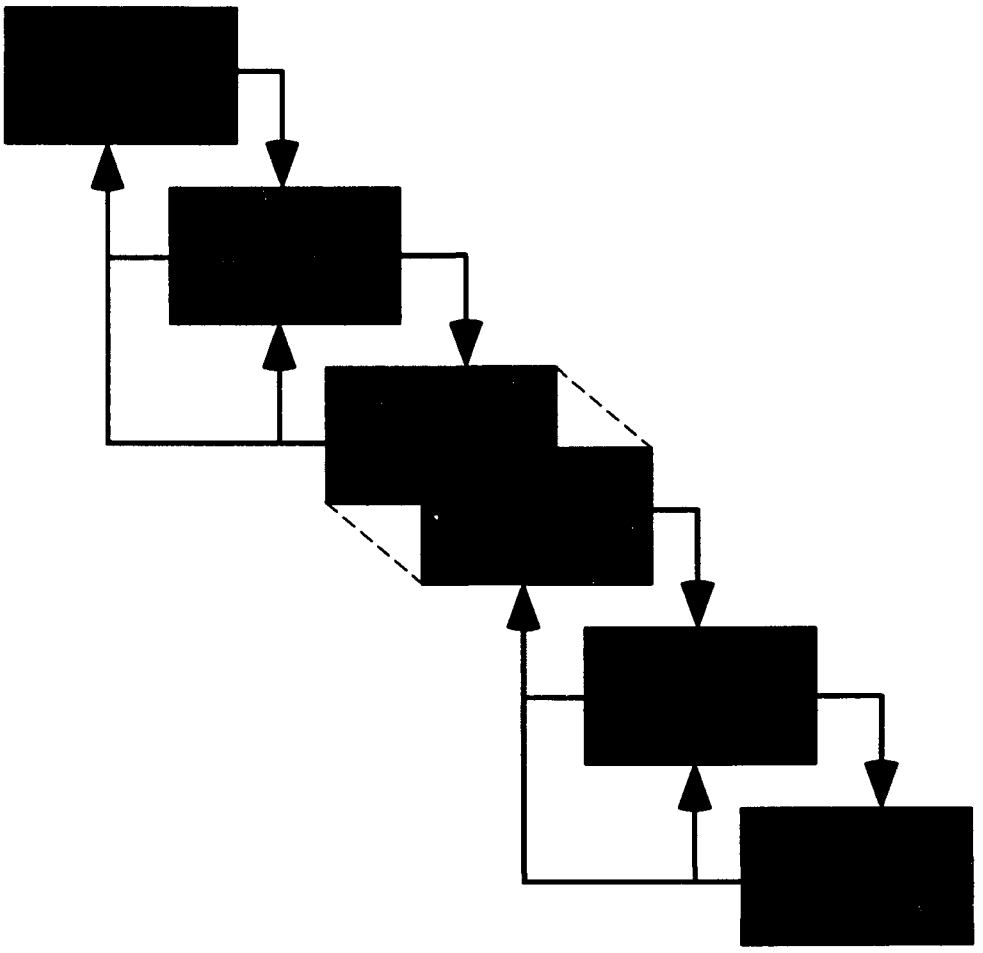

Figure 12: Cascading of Reuse Engineering Processes

number of overlapping relations possible, between idioms of the same or different types. Here, only one overlapping relation will be discussed in detail to illustrate the general concept. This relation is a significant special case termed "cascading".

Cascading

Cascading involves overlapping of the Asset Utilization family of one Reuse Engineering idiom with the Asset Creation family of another Reuse Engineering idiom, as depicted in Figure 12. As an example, consider the implementor of a windowing system that will be marketed as a reusable asset with a published programming interface. This person simultaneously enacts Asset Utilization activities in one Create-Manage-Utilize context (as a reuser of existing, lower-level window management capabilities) and enacts Asset Creation activities in another Create-Manage-Utilize context (as creator of the new windowing system that will be reusable by others, both to establish a window-based environment and to create higher-level window-based applications).

Cascading can iterate through multiple levels, in effect reflecting the "food chain" phenomenon that occurs naturally in many areas within software development, where a chain of value-added resellers build layers of application-specific enhancements to an underlying technical foundation.

Each layer, by adding capabilities that did not exist in previous layers, establishes a new context for creating reusable assets at a higher level of abstraction.

It is important to differentiate the cascading relation from typical producer-consumer relationships in conventional software life cycles (e.g., relationships between requirements and design, design and code, etc.). Processes in a cascading relation are not simply passing workproducts among one another as part of a development cycle. The asset creator, rather than providing products from previous life cycle stages that are prerequisites for the asset utilizer's work, provides reusable assets that enable products in the current life cycle stage to be developed more productively and at higher levels of abstraction.

The cascading relation may become increasingly useful for modeling a number of situations that will become more prominent in software engineering in the future. As more software products are marketed as assets rather than systems, and more users become value-added resellers, the modeling of cascading relationships as an integral part of reuse project planning will become more important.

Glossary

application engineering The development or evolution of a system to meet particular application requirements. In a domain-specific reuse-based environment this generally involves determining the criteria for selecting domain assets in the context of the application requirements; identifying, selecting, and tailoring assets to meet the criteria and requirements; and integrating the tailored assets into the application system.

application generator A software tool that generates software work products from non-procedural user specifications of desired capability.

architecture See *software architecture.*

asset A unit of information of current or future value to a software development or maintenance enterprise. Assets can include a wide variety of items, such as software life cycle products, domain models, processes, documents, case studies, research results, presentation materials, etc.

asset base A coherent set of assets, addressing one or more domains and residing in one or more asset libraries.

asset certification The process of determining to what extent an asset can be trusted to satisfy its requirements without error.

asset description Information about an asset that is useful for identification and selection of the asset by a user (e.g., author, functionality, usage history, dependencies, etc.) and is organized in accordance with a library data model.

asset library A set of assets and associated services for accessing and reusing the assets. A library typically consists of assets, corresponding asset descriptions, a library data model, and a set of services (manual or automated) for managing, finding, retrieving, and reusing assets. Such services can include reuse consultation services.

asset library interoperability The ability of two or more distinct and possibly heterogeneous asset libraries to dynamically access one another's assets, asset descriptions, data models, and/or library services.

asset model A data model describing relationships and constraints among assets in an asset base. An asset model describes the overall structure of the asset base.

asset qualification An aspect of the Asset Acceptance process that involves determining whether assets qualify for inclusion in an asset library based on domain-specific criteria such as conformance to a domain architecture.

asset specification The process of determining (and specifying) which assets should be developed in a domain and the range of features or capabilities they should support.

asset understanding The process of thoroughly analyzing an asset and its description in order to grasp the functionality being provided as well as the constraints and limitations on its use.

brokerage A function performed by a process when serving as an intermediary between other processes to facilitate or focus interaction and information exchange among those processes.

cascading In the CFRP context, a process modeling technique that enables overlapping of the Asset Utilization family of one Reuse Engineering idiom with the Asset Creation family of another Reuse Engineering idiom. Cascading relationships can reflect "food chain" relationships characteristic of value-added software resellers.

certification category A category to which an asset is assigned to indicate the degree of confidence that has been established about the asset relative to some criteria, such as functional correctness, reliability, or adherence to standards.

component Some part of a software life cycle product. A component may be subdivided into other components. A "complete" component includes both the component data itself and all related information that is needed to use it.

composition A process, technique, or method that involves combining elements of some kind of system or model to form larger, composite elements. For example, the elements can be software system components (in which case the process is known as *system composition*) or process model elements (e.g., the CFRP process model composition techniques).

constraint A functional or operational requirement for a system or model that limits the set of candidate solutions.

design The process of defining the software structure, components, modules, interfaces, and data for an application system to satisfy specified requirements.

design recovery The part of reverse engineering that involves analyzing system components and relationships to identify design elements, their interrelationships and interactions, and their design principles, requirements, and constraints. See *reverse engineering.*

domain An area of activity or knowledge. A number of different classification schemes have been proposed for domains; some of the classes of domains that have been identified include: application, horizontal, vertical, technology, computer science, execution, etc.. Figure 13 graphically depicts relationships among some of these classes of domains, as elaborated in the text below.

application domain The knowledge and concepts that pertain to a particular computer application area. Examples include battle management, avionics, C^3I, and nuclear physics. Each application domain can be decomposed into more specialized subdomains where the decomposition is guided by the overall purpose or mission of systems in the domain. For example, C^3I may be decomposed into C^3I for land operations, for sea operations, for air operations, etc.

horizontal domain The knowledge and concepts that pertain to particular functional capabilities that can be utilized across more than one application domain. Examples include user interfaces, database systems, and statistics. Most horizontal domains can be decomposed into more specialized subdomains where the decomposition is often guided by characteristics of the solution software. Distinguishing characteristics can include architectural style (e.g., functional, object-oriented, data-oriented, control-oriented, declarative, etc.), conceptual underpinning (e.g., relational or hierarchical data models), or required hardware. For example, user interface capabilities can be subdivided into those which support ANSI terminals and those which support bit-mapped, mouse input devices.

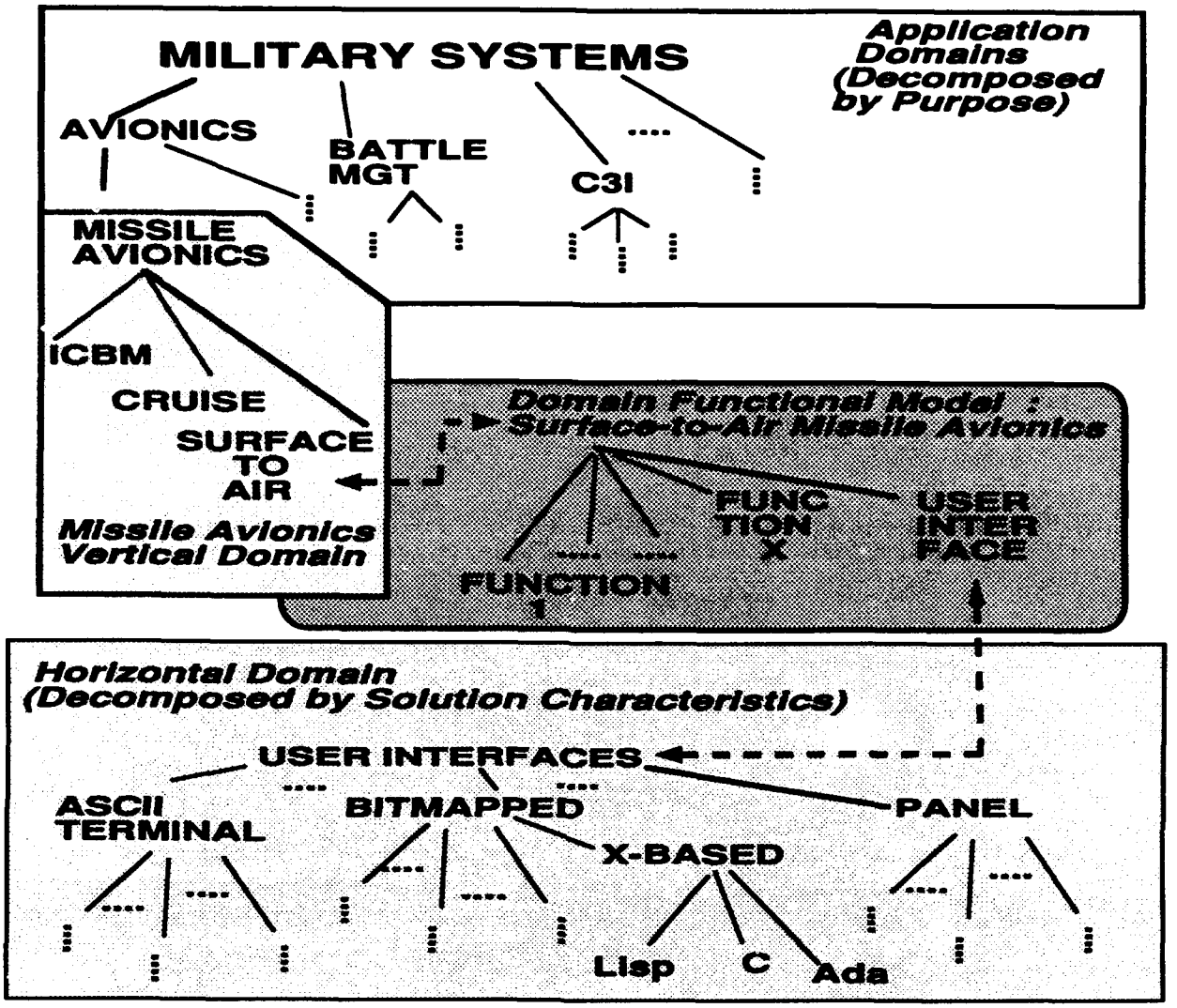

Figure 13: Types of Domains

vertical domain The knowledge and concepts that pertain to a class of systems addressing a particular functional need within an application (sub)domain. This class of systems can be organized as a hierarchy of functions. Also, horizontal (sub)domains may address functional requirements as described/modeled within the vertical domain functional hierarchy.

domain analysis The process of identifying, collecting, organizing, analyzing, and modeling domain information by studying and characterizing existing systems, underlying theory, domain expertise, emerging technology, and development histories within a domain of interest. A primary goal is to produce domain models to support the development and evolution of domain asset.

domain engineering The development and evolution of domain-specific knowledge and assets to support the development and evolution of application systems in a domain. Includes engineering of domain models, architectures, components, generators, methods, and tools.

domain knowledge Information about domains that exists in a variety of forms, including legacy systems and human expert knowledge.

domain model A definition of the characteristics of existing and envisioned application products within a domain in terms of what the products have in common (their "commonality") and how they may vary (their "variability").

domain requirements model A description of (a) the overall scope and context of a domain, and (b) the range of potential requirements on the operational characteristics of domain applications.

domain architecture model A description of the range of potential software architectures and designs that can satisfy domain requirements.

domain implementation model A description of the range of potential implementation assets that can satisfy domain requirements and architecture constraints.

domain-specific language A machine-processable language (the terms of which can be derived from a domain model) that is used to describe application system characteristics or capabilities within a domain. Often is used as input to an application generator.

domain-specific reuse Reuse in which the reusable assets, the development processes, and the supporting technology are appropriate to, and perhaps developed or tailored for, the application domain for which a system is being developed.

generation A technique or method that involves generating software work products from non-procedural user specifications of desired capability.

generator See *application generator.*

legacy systems Software systems in domains of interest that can impart legacy knowledge about the domains and feed domain analysis or reengineering efforts to produce domain assets or new application systems.

library data model The information (sometimes called meta-data) that describes the structure of the data in an asset library.

library mechanism A software system that provides a generic library system framework which can be tailored and perhaps extended to support asset libraries with specific capabilities.

life cycle The stages a software or software-related product passes through from its inception until it is no longer useful. Note that this definition shifts the usual definition of life cycle, which is based on the life of a *system*, to a more general concept covering the lifetime of a software *product.*

life cycle model A model describing the processes, activities, and tasks involved in the development and maintenance of software and software related products, spanning the products' life cycles.

life cycle stage One phase of a software life cycle addressing some particular aspect of a product's development or evolution. Examples include requirements analysis, design, implementation, and test. Each stage may produce individual *life cycle products* representing the results of that stage.

linkage In the CFRP context, a process modeling technique that enables representation of complex interaction patterns among reuse projects. Linked projects are distinct activities coordinated via the peer-to-peer exchange of information.

megaprogramming A software engineering paradigm that emphasizes process-driven, domain-specific reuse-based approaches to software development and evolution, automated by support tools and environments.

method A series of steps, actions, or activities that use a defined set of principles to bring about a desired result.

methodology A set or system of methods and principles for achieving a goal such as producing a software system.

process A description of a series of steps, actions, or activities to bring about a desired result. The process may be expressed at various levels of abstraction, reflecting the various degrees of precision appropriate at different organizational levels and at different stages in a software life cycle. Depending on the level of abstraction at which a process is described, it may or may not include well-defined inputs, intermediate products, constraints, needed resource descriptions, outputs, and testable criteria for starting, stopping, and moving on to the next step in the series.

process category A class of processes having common objectives, inputs, outputs and overall approach, but variable in methodology, complexity, formality, etc. The lowest level structural element type in the CFRP.

process-driven software engineering An approach in which software is developed or evolved in accordance with well defined, repeatable processes that are subject to continuous measurement and improvement and are enforced through management policies.

process family A collection of different process categories that together support a major functional capability. The middle level structural element type in the CFRP; decomposed into CFRP process categories.

process idiom A distinctive pattern of process activity. The top level structural element type in the CFRP; decomposed into CFRP process families.

project history Qualitative historical information about project processes, products, and infrastructure captured during project enactment.

query A request for identification of a set of assets, expressed in terms of a set of criteria which the identified items must satisfy.

recursion In the CFRP context, a process modeling technique that enables CFRP idioms to be nested or embedded within a process family in another idiom. Enacting the processes of a CFRP family involves enacting processes of any recursively embedded idioms as sub-processes.

reengineering The process of examining, analyzing, restructuring, and/or re-implementing existing computer software to address evolving requirements, implementation technologies, etc.

requirement A condition or capability that must be met or possessed by a software system or software-related product.

reuse The application of existing information. In software engineering, reuse usually involves the application of information encoded in software-related workproducts. A simple example of the reuse of software work products is reuse of subroutine/subprogram libraries for string manipulations or mathematical calculations. A simple example of the reuse of information not encoded in software workproducts is consultation with a human expert to obtain desired knowledge.

reuse-based software engineering An approach to software-intensive system development and evolution in which new and modified systems are constructed principally from existing software assets rather than through new development.

reuse cycle One pass through the Reuse Planning, Enactment, and Learning families in a particular reuse program.

reuse infrastructure The collection of capabilities that is needed to support and sustain reuse projects within a reuse program. Includes tools and technology; organizational structure, policies, and procedures; and education and training.

reuse initiative The first Reuse Management cycle of a reuse program, in which the program is initially established.

reuse library See *asset library.*

reuse program The set of activities encompassed by (and including) a particular instance of the Reuse Management idiom.

reuse program scope The technical and organizational boundaries of a reuse program. This includes delineation of the program's key domains of focus and application product lines.

reuse project A collection of Reuse Engineering activities that is enacted as a unit by Reuse Enactment processes.

reuse strategy A strategy for instituting and evolving reuse capabilities to satisfy overall objectives within an organization.

reverse engineering The process of analyzing a computer system's software to identify components and their interrelationships. See *design recovery.*

software architecture The high level design of a software system or subsystem. An architecture is defined in terms of the following general constructs:

- a set of software system elements, which may include both processing and data elements
- interfaces for each element
- a set of element-to-element connections, collectively forming interconnection topologies
- the semantics of each connection
  - the meaning of static connections (e.g., between data elements)
  - protocols describing information transfer across dynamic connections, in terms of element interfaces (general classes of protocols include procedure call, pipe, message passing, etc.)

software engineering environment (SEE) The computer hardware, operating system, tools, and encoded processes and rules that an individual software engineer works with to develop a software system.

specification A document or formal representation that prescribes, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a software system or software-related product.

tailoring The process of adapting products for application in new, specific situations. The adapted products may have been designed to anticipate specific forms of tailoring or may be tailored in unanticipated ways.

technique A set of procedures, rules, or skills that can be applied to achieve a particular class of results.

technology Technological capabilities that can contribute to the reuse infrastructure within an organization and can be applied to establish or automate reuse processes.

traceability The characteristic of software-related products that identifies and documents the derivation path (upward) and allocation/flowdown path (downward) of requirements and constraints.

variability The extent to which an element or characteristic of a domain may vary among individual systems or products in the domain.

References

[Ass92a] Asset Source for Software Engineering Technology (ASSET). ASSET Operations Plan. IBM STARS Technical Report CDRL 05602-001, Asset Source for Software Engineering Technology (ASSET), Morgantown, WV, June 1992.

[Ass92b] Asset Source for Software Engineering Technology (ASSET). Criteria and Implementation Procedures for Evaluation of Reusable Software Engineering Assets. IBM STARS Technical Report CDRL S45.11, Asset Source for Software Engineering Technology (ASSET), Morgantown, WV, March 1992.

[Bai89a] Sidney Bailin. Generic POCC Architectures. Report prepared for NASA Goddard Space Flight Center, Computer Technology Associates, Laurel, MD, April 1989.

[Bai89b] Sidney Bailin. The KAPTUR Environment: An Operations Concept. Technical report, CTA Incorporated, Rockville, MD, 1989.

[Bas90] Victor Basili. Viewing Maintenance as Reuse-Oriented Software Development. *IEEE Software*, 7(1):19–25, January 1990.

[BB91] Bruce Barnes and Terry Bollinger. Making Reuse Cost-Effective. *IEEE Software*, 8(1):13–24, January 1991.

[BCC92] Victor Basili, Gianluigi Caldiera, and Giovanni Cantone. A Reference Architecture for the Component Factory. *ACM Transactions on Software Engineering and Methodology*, 1(1):53–80, January 1992.

[Bee72] Stafford Beer. *Brain of the Firm.* Herder and Herder, New York, NY, 1972.

[Big89] Ted Biggerstaff. Design Recovery for Maintenance and Reuse. *IEEE Computer*, 22(7):36–49, July 1989.

[BR87] Ted Biggerstaff and Charles Richter. Reusability Framework, Assessment, and Directions. *IEEE Software*, 4(2):41–49, March 1987.

[Cen92] Central Archive for Reusable Defense Software (CARDS). Acquisition Handbook. Unisys STARS Technical Report STARS-AC-04105/001/00, US Air Force Materiel Command, Electronic Systems Center, Hanscom Air Force Base, MA, October 1992.

[Cen93a] Central Archive for Reusable Defense Software (CARDS). Direction Level Handbook. Unisys STARS Technical Report STARS-AC-04104/001/00, US Air Force Materiel Command, Electronic Systems Center, Hanscom Air Force Base, MA, March 1993.

[Cen93b] Central Archive for Reusable Defense Software (CARDS). Franchise Plan. Unisys STARS Technical Report STARS-AC-04116/001/00, US Air Force Materiel Command, Electronic Systems Center, Hanscom Air Force Base, MA, March 1993.

[Cen93c] Central Archive for Reusable Defense Software (CARDS). Technical Concept, Command Center Library. Unisys STARS Technical Report STARS-AC-04107A/002/00, US Air Force Materiel Command, Electronic Systems Center, Hanscom Air Force Base, MA, March 1993.

[CKO92] Bill Curtis, Marc I. Kellner, and Jim Over. Process Modeling. *Communications of the ACM*, 35(9):75–90, September 1992.

[Cle88] J.C. Cleaveland. Building Application Generators. *IEEE Software*, 5(4):25–33, July 1988.

[Dem86] W.E. Deming. *Out of the Crisis.* MIT Press, Cambridge, MA, 1986.

[DIS93] DISA/CIM Software Reuse Program. Domain Analysis and Design Process, Version 1. Technical Report 1222-04-210/30.1, Defense Information Systems Agency Center for Information Management, 701 South Court House Road, Arlington VA 22204, March 1993.

[DoD92] DoD Software Reuse Initiative. DoD Software Reuse Vision and Strategy. Technical Report 1222-04-210/40, Center for Software Reuse Operations, Alexandria, VA, July 1992.

[FG90] W.B. Frakes and P.B. Gandel. Representing Reusable Software. *Information and Software Technology*, 32(10), December 1990.

[HCKP89] Robert R. Holibaugh, Sholom G. Cohen, Kyo C. Kang, and Spencer Peterson. Reuse: Where to Begin and Why. In *Proceedings of Tri-Ada '89*, pages 266–277, New York, NY, October 1989. Association for Computing Machinery.

[JHD+90] A. Jaworski, F. Hills, T. Durek, S. Faulk, and J. Gaffney. A Domain Analysis Process. Technical Report DOMAIN_ANALYSIS-90001-N, Software Productivity Consortium, 2214 Rock Hill Road, Herndon VA, 22070, January 1990.

[Kam89] Gary M. Kamsickas. An Objective Look at the Modularization and Standardization of Training Systems. In *Proceedings of the Eleventh Interservice/Industry Training Systems Conference*, 1989.

[KCH+90] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, November 1990.

[MCP+88] D. McNicholl, S. Cohen, C. Palmer, J. Mason, C. Herr, and J. Lindley. Common Ada Missile Packages – Phase 2. Technical Report AFATL-TR-88-62, Air Force Armament Laboratory, Eglin AFB, FL, November 1988.

[MG92] E. Mettala and M. Graham. The Domain-Specific Software Architecture Program. In *Proceedings of the DARPA Software Technology Conference 1992.* Defense Advanced Research Projects Agency, April 1992.

[Nav93] Naval Surface Warfare Center Systems Reengineering Strategies Working Group. Draft Reengineering Taxonomy and Process. Technical Report NSWCDD/TR-92/290, Naval Surface Warfare Center, Silver Spring, MD 20903, August 1993.

[PDA91] R. Prieto-Diaz and G. Arango, editors. *Domain Analysis and Software Systems Modeling.* IEEE Computer Society Press, Los Alamitos, CA, 1991.

[PDF87] R. Prieto-Diaz and P. Freeman. Classifying Software for Reusability. *IEEE Software*, 4(1):6–16, January 1987.

[PW92] D.E. Perry and A.L. Wolf. Foundations for the Study of Software Architecture. *ACM Software Engineering Notes*, 17(4):40–52, October 1992.

[Rat91] Rational. Foundation for Competitiveness and Profitability: FS2000 System, Rational, and Ada. Case study, Rational, Santa Clara, CA, 1991.

[Reu90] Reusable Ada Products for Information System Development (RAPID). Final RAPID Center Reusable Software Component (RSC) Procedures. Softech Technical Report 3451-4-326/4, US Army Information Systems Engineering Command, Ft. Belvoir, VA, June 1990.

[Sca92] Walt Scacchi. Process-Driven Environments as Reusable Application Development Frameworks. In *Proceedings of the Fifth Annual Workshop on Software Reuse*, November 1992.

[Sch83] Donald Schon. *The Reflective Practitioner: How Professionals Think in Action.* Basic Books, New York, NY, 1983.

[Sen90] P.M. Senge. *The Fifth Discipline.* Doubleday/Currency, New York, NY, 1990.

[Sha89] Mary Shaw. Large Scale Systems Require Higher-Level Abstractions. In *Proceedings of the 5th International Workshop on Software Specification and Design*, Pittsburgh, PA, May 1989.

[Sha91] Mary Shaw. Heterogeneous Design Idioms for Software Architectures. In *Proceedings of the 6th International Workshop on Software Specification and Design*, Los Alamitos, CA, October 1991. IEEE Computer Society Press.

[Sof91a] Software Productivity Consortium. Ada Quality and Style: Guidelines for Professional Programmers. Technical Report SPC-91061-N, Version 02.00.02, Software Productivity Consortium, Herndon, VA, 1991.

[Sof91b] Software Technology for Adaptable Reliable Systems (STARS). Domain Specific Environment Repository — Composite Paradigm Report. Unisys STARS Technical Report STARS-SC-03068/001/00, Advanced Research Projects Agency, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, May 1991.

[Sof91c] Software Technology for Adaptable Reliable Systems (STARS). Reuse Library Process Model. IBM STARS Technical Report CDRL 03041-001, Advanced Research Projects Agency, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, July 1991.

[Sof92] Software Technology for Adaptable Reliable Systems (STARS). Asset Library Open Architecture Framework (ALOAF), Version 1.2. Unisys STARS Technical Report STARS-SC-04041/001/02, Advanced Research Projects Agency, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, August 1992.

[Sof93a] Software Technology for Adaptable Reliable Systems (STARS). Organization Domain Modeling (ODM), Volume I – Conceptual Foundations, Process and Workproduct Descriptions, Version 0.5 – DRAFT. Unisys STARS Technical Report STARS-UC-05156/024/00, Advanced Research Projects Agency, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, July 1993.

[Sof93b] Software Technology for Adaptable Reliable Systems (STARS). Reuse Strategy Model: Planning Aid for Reuse-based Projects. Boeing STARS Deliverable D613-55159, Advanced Research Projects Agency, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, July 1993.

[Sof93c] Software Technology for Adaptable Reliable Systems (STARS). STARS Conceptual Framework for Reuse Processes (CFRP), Volume II: Application, Version 1.0. Unisys STARS Technical Report STARS-VC-A018/002/00, Advanced Research Projects Agency, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, September 1993.

[Sof93d] Software Technology for Adaptable Reliable Systems (STARS). The Reuse-Oriented Software Evolution (ROSE) Process Model, Version 0.5. Unisys STARS Technical Report STARS-UC-05155/001/00, Advanced Research Projects Agency, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, July 1993.

[SWT89] James J. Solderitsch, Kurt C. Wallnau, and John A. Thalhamer. Constructing Domain-Specific Ada Reuse Libraries. In *Proceedings of the 7th Annual National Conference on Ada Technology*, 1989.

[U.S89] U.S. House of Representatives. Bugs in the Program: Problems in Federal Government Computer Software Development and Regulation. U.S. House of Representatives Staff Study, Committee on Science, Space, and Technology, September 1989.

[Vir92a] Virginia Center of Excellence for Software Reuse and Technology Transfer (VCOE). Reuse Adoption Guidebook. Technical Report SPC-92051-CMC, Software Productivity Consortium, Herndon, VA, November 1992.

[Vir92b] Virginia Center of Excellence for Software Reuse and Technology Transfer (VCOE). Domain Engineering Guidebook. Technical Report SPC-92019-CMC, Version 01.00.03, Software Productivity Consortium, Herndon, VA, December 1992.