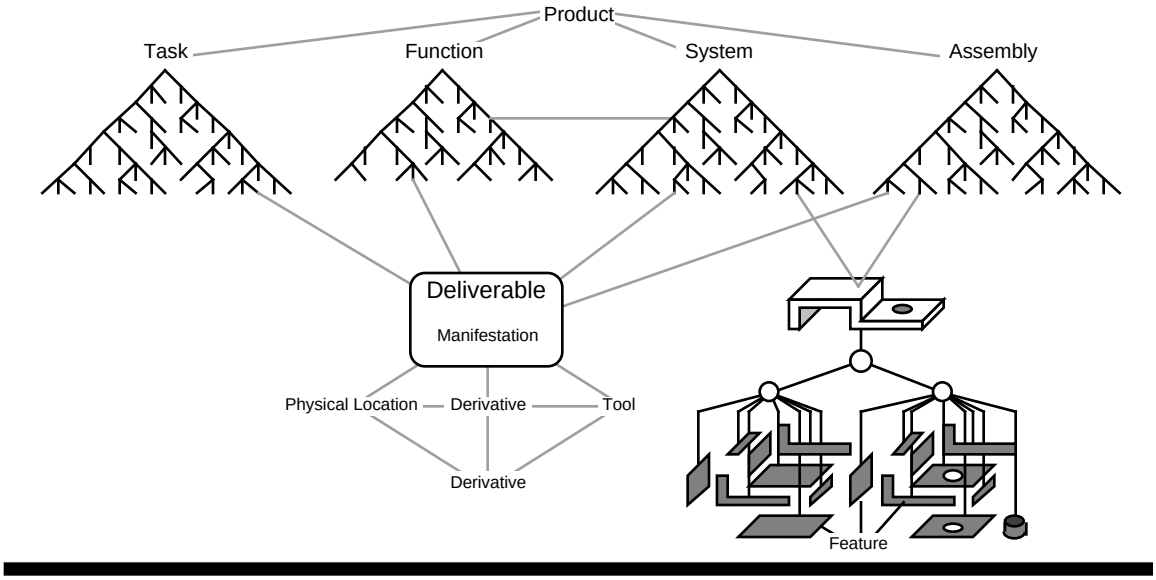
Enterprise Integration and Management



William T. Holmes

Library of Congress Cataloging-in-Publication Data

Holmes, William T. (date)
Enterprise Integration and Management/William T. Holmes
Includes Index.
ISBN 0-914743-02-3
1. Computer Integrated Manufacturing
2. Concurrent Engineering
3. Integrated Product Development
4. Computer-Aided Design (CAD)
5. Computer-Aided Analysis (CAA)
6. Computer-Aided Engineering (CAE)
7. Computer-Aided Systems Engineering (CASE)
8. Computer-Aided Software Engineering (CASE)
9. Computer-Aided Manufacturing (CAM)
10. Computer-Aided Logistics Support (CALS)
11. Solid Modeling
12. Process Improvement

Published by
Matrix Design Publications
P.O. Box 3336
Lennox, California 90304
Printed in the United States of America

Copy privileges are a function of the format purchased.

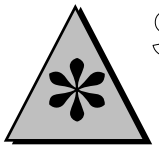
Format	Copy Permission	Retail Price
Vello-Bind™	Not to be copied in whole or in part	$30
Three-ring	May be copied without modification	$60
3.5" Diskette	Diagrams only with credits — MacDraw II, PICT, others by request	$80

General Dynamics Convair Division has unlimited copy rights.

2 3 4 5 6 7 8 9 10

ISBN 0-914743-02-3

Distributor:



Synergy Enterprises
P.O. Box 36, Escondido, Calif. 92033-0036
619-672-9836 voice and FAX

Enterprise Integration and Management

William T. Holmes

Matrix Design Publications
P.O. Box 3336
Lennox, California 90304

Contents

1. Introduction

The accolades I received from representatives of IBM, DEC, Matra DataVision, Prime, SDRC, the Mechanical Engineering Department, Michigan State University and others for the "Solid Modeling Brief" motivated me to combine the best of my related work into a useful book for enterprise integration and management purposes. The Solid Modeling Brief is but a small portion of this work.

This book is intended to help business be as efficient and responsive as they can be with the tools available to them. No business can afford to develop and maintain all of its tools for long, so an equally important intent is to provide a view of the business process that illustrates to commercial tool providers the similarity of the process, regardless of the products involved. By simplifying the information relationships, this view of the process demonstrates that the market for sophisticated and comprehensive enterprise integration tools justifies the investment required to develop them.

A large system integration enterprise with complex products is the model for this demonstration, because it represents the most difficult communication and coordination problem. It involves nearly all engineering and manufacturing disciplines. Its information integration and tool requirements are the most extreme. If the needs of such an enterprise can be satisfied with an integrated tool set, the needs of any enterprise can be satisfied with the same tool set.

Good tools are not the only ingredient for successful enterprise integration and management. The prospective buyers of integration tool sets must change their business process to take complete advantage of the tools. They must see the business process as a continuous process rather than organizations performing functions. They must view their resources as independent contributors to the process rather than organizational possessions.

The current political and business environments are described to establish an understanding of how American business has declined, and how it must change to meet new challenges and capitalize on new opportunities. A generic business process is described as a template for all of the product lines of any business. A way to simplify the information relationships required to support an integrated design and manufacturing enterprise is described. Then all of the various resources (computers, tools, humans, machines and facilities) required to conduct the business process of a modern enterprise are described in the context of the generic business process and integrated information paradigm.

2. Environment

Like any organism, an enterprise must either adapt to its environment or die. Those that can adapt quickly to rapidly changing technology and markets survive those that cannot. As products become more complicated and the market place more dynamic, the command and control business management paradigm has became increasingly inappropriate. The following is a view of the dynamic business environment of the 1990s.

2.1. Old Paradigm

Businesses tend to emulate the government bureaucracy imposed on them. They have to staff functions to promulgate government regulations as company policy, complete government forms and collect government taxes. Defense businesses have the additional burden of the military bureaucracy. Consequently, American business resembles the command and control government that resulted from the Great Depression and World War II. Increasing product complexity significantly aggravated management problems.

2.1.1. Specialize

Before computers were readily available to automate much of the design and analysis process, design teams had to increase in number to accommodate more complex products. Individual engineers or clerks were dedicated to manage the product configuration, the program schedule, the changes to the product, the contract requirements, the subcontractor requirements, the contract documents, etc. As the workload or scope of work grew, the individuals asked for help. Entire organizations grew around them. As these "support" organizations grew, the communications among them and with the core engineering and manufacturing organizations degraded.

Forms were created to facilitate accountable communications. Individuals were dedicated to see that the forms were completed. Others were dedicated to ensure that they were completed correctly. Organizations were created to control the creation of forms.

2.1.2. Re-Organize

The "Matrix management" organizational approach had people report to both a Functional manager and a Program manager. After it failed to improve the responsiveness of human resources to the Program managers, the Program managers physically removed people from their functional groups (marketing, design, analysis, purchasing, etc.), and "co-located" them. That only substituted one communication problem (product team communication) for another (communication within a discipline), and aggravated the overhead rate by changing facilities and moving as much as one-half of the human resources annually as Program resource demands changed. Many key personnel had two or more desks, because their experience was desired by more than one

Program manager. The "lessons learned" by members of a discipline were confined to a Program and seldom benefitted other members of that discipline and the Programs to which they were assigned. The analysis tools and procedures that were once refined within functions were re-invented for each Program. Software written for a Program was not architected to be useful on subsequent Programs.

2.1.3. Compartmentalize

Each new subdivision of responsibilities and tasks spawned new controls. More individuals were dedicated to drawing vaults, change control boards and task implementation boards. Whole organizations like Quality Assurance (QA) and Integrated Logistics Support (ILS) were established to convince the military that a defense contractor was serious about resolving production and design quality problems, but such organizations only added more layers to the bureaucracy. The fundamental communication, coordination and product design problems remained.

This was the legacy inherited from World War II by the next generation of managers. With computers at their disposal, but little in the way of communications, they employed computers to speed the work of support organizations like accounting and payroll. Engineers belatedly replaced their slide rules with computers for analysis purposes. Each department managed "their" data and printed reports for their internal (downstream) customers.

2.1.4. Computerize the Compartments

Even when computer security and data management capabilities allowed originators and users of the data to access and update the information without jeopardizing it, the support fiefdoms persisted. They insisted that the job they performed was too complicated, specialized or troublesome for engineers to do themselves. The time required for engineers to communicate their purchasing needs or configuration changes to the support organizations with forms was more than time required for engineers themselves to complete a purchase order or change a configuration on-line.

This was a grand mistake. It reinforced organizational fiefdoms and delayed the effective use of computers for integration by industry for twenty years. Had these computer systems been made accessible throughout the company, their cryptic, department-specific user interfaces would not have been tolerated. Company practices would have been questioned and simplified. The degree of integration and information sharing we strive for today would have been achieved long ago.

Now the peripheral functions do little more than update their various parochial computer systems in a vain attempt to make "their" data match the reality of engineering and manufacturing process. These systems are generally only paper tracking systems. They contain no data that is not also on some paper form. In addition, they contain form numbers, revision letters and dates to

facilitate synchronization with other forms. This is redundant data in a computer system.

Although many of these systems include "edit checks" that preclude the entry of nonsense data, support organizations do not have engineers perform the data entry directly. Instead, the engineers complete forms that tell clerks what to input to the computer, doubling the work and overhead cost and increasing schedule.

2.1.5. Sacrifice Product Quality and Cost for Schedule

Engineers have always known that they should concern themselves with the reliability, producibility and maintainability of their designs as well as their performance. Unfortunately, the bureaucracy was unresponsive to the schedule demands of Program managers, so something had to give. That something was product quality and producibility. The attitude was "We'll fix that later if we survive our more immediate problems." Consequently, bad designs were dumped on manufacturing. Manufacturing either returned them to Engineering with a change request, or employed exotic manufacturing technology to avoid the cost of a design change or avoid extending the schedule.

2.2. New Paradigm

The trends we see in government have direct parallels in business. We are witnessing the demise of authoritarian regimes and authoritarian businesses. Individuals and businesses are being liberated from government regulation. Individuals are being liberated from nonsense business procedures.

2.2.1. Trends in Politics

With all of their power to control personal and economic freedom, the authoritarian regimes are disintegrating. The long queues for some products and the warehouses full of unwanted products attest to the impracticality of centralized planning and the wonder of the free market. There are just too many variables involved in the process of supplying viable products to consumers.

Draught can quickly render a five-year production plan obsolete. Slogans do not motivate business or workers for long. Privilege (guaranteed market, good housing, seats at the ballet) is a poor substitute for profit motive. Despite good intentions, the authoritarian regimes have wreaked economic and environmental havoc on their countries.

The fundamental assumption underlying the authoritarian paradigm is *imperfect man*. Authoritarians assume that humans are fundamentally stupid or corrupt. If left to their own self-serving devices, they are at best inefficient and at worst self-destructive. A few intelligent men must contrive laws to control the behavior of these individuals for their own good or the "common good." If there is a

problem, it must be caused by too much freedom or the abuse of it. This assumption underlies the actions of all authoritarians, be they fascists, communists or socialists.

The world is reverting to the libertarian paradigm that was prominent in the United States of America in the 17th and 18th centuries. The fundamental assumption underlying that paradigm was *perfect freedom* (Howard Freeman): humans are fundamentally intelligent and good. If left to act in their own interest, they will directly and indirectly benefit their fellow man. If there is a problem, search for an existing law that restricts freedom. Eliminate the restriction and the problem will be eliminated.

2.2.2. Trends in Business

Despite all of their lobbying power to use government to restrict competition with licenses, regulations and import/export duties, authoritarian businesses are failing. They cannot deliver the products that are in demand. They have warehouses full of unwanted products. There are just too many resources and variables to *manage*.

Innovative competition and changing technology can render development and production plans obsolete before their enactment. Slogans do not motivate workers for long. Privilege (office size, parking space) is a poor substitute for profit motive. Profit disconnected from market demand will fail to motivate the production of the right products at the right time. Despite good intentions, the command and control managers have invited the growth of equally destructive labor unions. Their customers have grown dissatisfied with their products, and they are rapidly losing market share to their more agile competitors.

The fundamental assumption of authoritarian business managers is that employees are fundamentally unmotivated or corrupt. If left to their own self-serving devices, they will be unproductive, or steal materials or time from the company. A few intelligent men must contrive standard practices, control access to supplies, tools, parts and material and install time clocks to control the behavior of these individuals for their own good and the good of the company. If there is a problem, it must be caused by too much freedom or the abuse of it, so a procedure is created or modified, and a new form is created and dispensed.

Businesses are reverting to the libertarian paradigm. Many case studies demonstrate that paternalistic management, like paternalistic government is counter-productive. Employees are fundamentally motivated and good. If left to act in their own interest, they will directly and indirectly benefit the company. If there is a problem, search for an existing rule that restricts employee freedom. Eliminate the restriction and the problem will be eliminated.

Complacent business managers are awakening to a world economy and intense competition. Some continue to use government tariffs, duties and regulations to retard the inevitable, but that behavior only delays the demise of

an infirm business at the expense of the consumer, the taxpayer and entrepreneurial businesses. Enlightened business managers are collapsing their management hierarchy and treating their employees like adults. They have learned that hierarchically distant management cannot know enough about a problem to make a good decision. Employees who share the company vision make the best decisions concerning the problems that confront them hourly.

The employees of enlightened businesses are attuned to their customers and responsive to the marketplace. They control their process. They can stop it, change it and expend company funds to improve it. They are trained in many functions of the process and spontaneously react to minimize the effect of bottlenecks without management intervention. They establish their own measures of success. They vary their work hours and location as the situation dictates. They determine for themselves who does what work and when. Often they determine how much is re-invested in the business, how much is spent on labor and who gets what pay. They select their peers and their team leaders. They are self-coordinated, because they share the same vision for their company. They are motivated to perform because they want to be proud of their products and benefit from the profits. They perform significantly better than their shackled peers.

2.2.3. Economy of Scale

New technology has nullified the *economy of scale* concept that characterized the industrial age. Small, specialized steel plants are rapidly capturing market share from the old behemoths. Personal computers and easy-to-use programs have eliminated the need for specialized or centralized clerical, accounting, billing, payroll and reporting organizations. The minimum effective size of a business is no longer constrained.

Interpersonal communications limit the effective size of teams. The need to keep employee profit connected with market demand limits the effective size of a profit center within a business. As a business grows it must subdivide to maintain effective communications and market sensitivity. Profit centers become customers and suppliers of other profit centers who in turn supply to the original customer.

As a source of products, material or services, the profit centers may become indistinguishable from preferred vendors or subcontractors. They may become independent businesses. Various forms of voice, video and data communications will enable the teams and profit centers to share information and coordinate their activities, regardless of their geographic location.

The teams consist of individuals, who often can afford to own all the resources they require to perform and manage their work. The minimum effective size of a business is an individual.

2.2.4. Price

Price is still the most efficient consumer/supplier communication tool available. Price (wages) is still the best way to communicate resource needs and match resources to the job. Price becomes the communication key between departments when they grow and separate into independent enterprises, so why shouldn't price also be the way to communicate demand for intermediate deliverables among functions (departments) within a company? The use of price to coordinate the internal activities is explored in the Resources section.

2.3. New Business Drivers

The aforementioned political trends have a direct impact on business. Domestic and international competition are increasing as are international market opportunities. The impact of the political trends is particularly acute for defense contractors. The threat of peace will reduce the demand for defense products.

Gone are the days of exclusive cash-cow production contracts. No longer are they the expected result of winning a design competition. Design competitions may not be won by lobbying the U.S. Congress.

Gone are the cost-plus product development contracts. No longer are mistakes of little concern, because the customer would pay for them. New contracts will be fixed-price.

Gone are the days when shoddy products delivered on schedule earned a delivery incentive award, and fixing the products after delivery earned more profits. New products will only be sold under warranty.

According to the September 19, 1989 "Financial World":

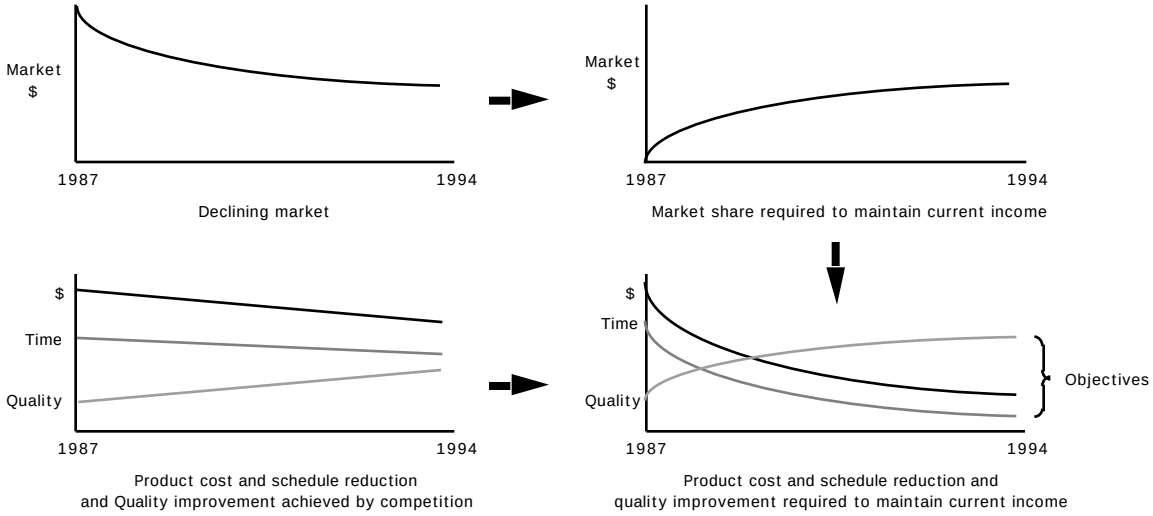
> "… since 1985 defense spending in the U.S. declined 15% in real terms. Conservative estimates project an annual 2% to 5% real decline well into the next decade. … Northrup posted a $78 million loss in the second quarter and immediately slashed 3,000 from its labor force. Hughes Aircraft … announced plans to trim 6,000 from its payroll. … McDonnell Douglas surprised Wall Street with a stunning $48 million loss.
>
> "What it boils down to is shakeout and consolidation [that] appears to be well underway. … the number of firms doing business with the Department of Defense (DoD) has plummeted 60% in just four years. … In the past three years, such well-known companies as Honeywell, Sperry, Gould and Goodyear have dumped their defense divisions …
>
> "To make matters worse, the raft of procurement reforms begun by Congress and [the] DoD is starting to sting. Increasing competition, smaller progress payments, mandated second sourcing, lower federal R&D funding and rigid fixed-price contracts have crimped margins and

made weapons building a high-risk venture. …Fixed-price contracts … have left earnings time bombs in company backlogs where those contracts were underpriced. … Since 1985, for instance, Hughes, United Technologies, Boeing and Northrup have together swallowed nearly $900 million in cost overruns.

"No surprise then that cash flow for most major contractors has declined nearly 35%. Or that net margins are down. Or that balance sheets are deteriorating as the industry loads up with debt to fund its increasing cash demands.

"…European defense contractors have managed to build themselves up so that they are competing nearly head to head with their U.S. counterparts. The once-huge U.S. arms trade surplus with Europe has been trimmed from $7 billion to $1 billion."

To maintain its current level of income in a declining market, a business must increase its market share at the expense of its competitors. Its competitors, however are reducing costs, reducing time to market and improving quality to increase or maintain their market share. Therefore its cost and time to market reduction and quality improvement objectives must be extraordinary.



Furthermore, future defense products may be developed entirely at the expense of the contractor. The cost of development would be recouped only after the delivery of a quantity of product. This fact, coupled with high interest rates, will further erode already thin profit margins.

The new emphasis on quality and cost to own was manifested in various DoD initiatives. For example, the Willoughby Templates (transition to production) reasserted the need to consider production as well as performance issues (function and reliability) during the design of a product. How a product meets its original requirements must be demonstrated. The Computer Aided Logistics (CALS) initiative reasserted the need to consider maintenance as well as production and performance issues during the design of a product. Product

assembly and maintenance information must be made electronically available to supply and maintenance depots, ships at sea and troops in the field. *Concurrent* or *Simultaneous* Engineering is touted as a means to improve time to market while considering manufacturing and maintenance issues during design. Total Quality Management (TQM) says it all in fewer words: *continuously improve*.

Although this combination of events has been particularly acute for the defense industry, they have directly or indirectly caused commercial enterprises to reconsider their business process. Enterprises that produce largely commercial products are embracing CALS, Concurrent Engineering and TQM to compete internationally and satisfy customers who are demanding quality products. The cost to own products is considered more important than the cost to buy them.

Each of these business drivers are described in detail below. Although they are directed to defense contractors, more commercial enterprises are beginning to adopt them as practical business practices.

2.3.1. Willoughby Templates

DoD directive 4245.7-M, Transition from Development to Production (Willoughby Templates) presents guidelines developed by the Department of Defense. These guidelines encourage improvement in the product development processes, particularly during its transition to production. Many of the recommended changes are reflected in the Process Architecture.

The Willoughby Templates imply that product data should be accessible. This need is reflected in the Information Architecture, which makes product data readily available for producibility, reliability and supportability as well as performance purposes. The design subsection of the Willoughby Templates is organized into thirteen categories:

1. Design Requirements
2. Trade Studies
3. Design Policy
4. Design Process
5. Design Analysis
6. Parts Selection and Materials Selection
7. Software Design
8. Computer-Aided Design (CAD)
9. Design for Testing
10. Built-In Test (BIT)
11. Configuration Control
12. Design Reviews
13. Design Release

A analysis and technical manuals are addressed in the Logistics subsection. The following examples illustrate criteria affecting the business process.

2.3.1.1. Design Requirements

The designation of detailed design requirements is singularly important in the discussion of design activities. The requirements definition process starts with concept formulation, iterates with trade studies and ends in refined mission/environmental profiles. This results in the firm requirements necessary for full scale development. Producibility considerations:

- Do design engineers have manufacturing knowledge or experience?
- Does design policy include producibility?
- Are manufacturing engineers involved early in the design process?
- Is engineering involved in developing the manufacturing plan?

Requirements for joint engineering/manufacturing participation throughout the full-scale development phase are among the most critical. They minimize the risk associated with the transition from development to production.

2.3.1.2. Trade Studies

The DoD definition of trade studies is the evaluation of concepts, policies, techniques, methods, and systems in terms of their cost and effectiveness. Trade study considerations:

- Are trade studies iterative from concept through full scale development (FSD)?
- Have trade study results identified the risks associated with new technologies?
- Has the producibility of each design alternative been considered in a separate trade study?
- Have the salient design standards been passed on to subcontractors?

2.3.1.3. Design Policy

A design policy is a statement supported by controlled engineering manuals, procedures, or guidelines, that attempts to reduce risk by implementing fundamental design principles and practices. Where (corporate design policies) exist, they often lack substantive direction regarding best design practices.

Design polity considerations:

- Are lessons learned reflected in the design policy or guidelines?
- Is the design process treated in the design policy guidelines and standards?

2.3.1.4. Design Process

The engineering design activities that are necessary for product development are often treated as discrete functional activity, with little or no involvement of the other plant functions (e.g., manufacturing or production engineering). As a result, the design of the product meets performance specifications at the completion of development, but does not allow for the limitations of manufacturing processes and procedures found on the factory floor. Hence, the apparently mature product configuration does not survive rate production

without performance degradation, and significant redesign is required for efficient production.
Design process considerations:

- Does the policy of the contractor include producibility as part of design reviews?
- Are manufacturing and producibility personnel involved in the design process?
- Are proven manufacturing processes being used whenever possible, with trade studies performed to justify the use of new technology?
- Are design and manufacturing engineers co-located during development?

2.3.1.5. Design Analysis

As the design process progresses, analytical techniques guide the continuing effort to arrive at a mature design. While the design process concerns the actual additions, deletions and changes to the design, design analysis evaluates the ability of the design to meet performance specifications at low risk. Those analyses oriented to the reduction of design risk include, but are not limited to, stress and strength, worst case tolerance, sneak circuit, failure modes and effects, and thermal analyses. These are the checklist items:
Design analysis considerations:

- Has continuous design analysis throughout the design process been specific?
- Do division standards identify design analysis as an integral part of the design process?
- Are design engineers required to participate in and use the results of design analyses?
- Has proper balance been achieved between design analysis and testing?

2.3.1.6. Parts and Materials Selection

Design engineers often apply parts and materials too close to maximum rated stress levels and may also specify nonstandard parts in an effort to meet performance requirements. The uncontrolled use of these techniques leads to high risk during testing and operational use. They decrease operational readiness, and increase logistics support systems complexity.
Parts and material selection considerations:

- Does the contractor have an established set of de-rating criteria that all engineers must use?
- Will part operating temperatures be determined by thermal survey measurements?
- Are the results of thermal analyses and thermal survey measurements being used in the design process?

2.3.1.7. Software Design

Modern weapon systems have become increasingly dependent upon software for their operation. No cost-effective procedure exists for eliminating failures, or even accurately measuring the failure rate due to software. Software design practices must follow a disciplined process similar to proven hardware design practices. Trade analyses can disclose significant life-cycle cost savings. They result in the proper allocation of hardware and software roles. They can minimize the difficulty of isolating and correcting design problems.
Software design considerations:

- Are hardware/software allocations assigned soon after preliminary design trades are completed?
- Are hardware/software interfaces clearly defined?
- Is the draft of the users' manual outline scheduled for completion before the start of programming?
- Are reports available for review at the detailed design and coding level?

2.3.1.8. Computer-Aided Design (CAD)

Here the Willoughby Templates directly address the role of computing, and in particular the use of design workstations in developing reliable products. Many design tools and analysis techniques that will facilitate the design process are not used by contractors. They do not impact the product meaningfully. Design tools that will facilitate the design process and yield a producible product are available. The use of such equipment decreases the length and cost of reliability development testing. It decreases the cost for tooling and test equipment. It eliminates redesign efforts due to producibility issues. It reduces the risk during the transition from development to production.
CAD/CAM database considerations:

- Design specifications including mission profile, performance limits and requirements, and reliability requirements.
- Design standards and rules that support (Division) policies.
- Verified libraries of preferred electronic parts with both performance and physical characteristics, including tolerance
- Preferred mechanical parts
- Previously manufactured and qualified assemblies
- Materials, processes, and finishes
- Manufacturing processes, standards, and rules
- Design data, including analytical results
- Manufacturing data, including design release status, test status, test and failure analyses, and manufacturing yield and trend analysis
- Tool design
- Control of design release and configuration.

Computer-Aided Design considerations:

- Is the use of CAD a Division policy?
- Are individual terminals (sized for basic text processing and display functions) available for each engineer?
- Are interactive graphics terminals provided for groups of engineers?

- Does a formalized training program exist for introducing engineers to CAD?
- Is a common and up-to-date database available containing parts and materials information as well as design engineering information?
- Is CAD included in the Division's overall modernization strategy?

2.3.1.9. Design for Testing

To provide for efficient and economical manufacturing, test and inspection capabilities must be incorporated in the design. Past development projects have neglected to consider the need for production and field test capabilities during the early design phase. Attempting to add these capabilities later is difficult and costly, especially in those cases where production has begun. These are the relevant checklist items:

- Is a division policy for design-for-testing in effect?
- Have production test guidelines been established before full scale development?
- Have trade studies been done during design to establish relative levels of built-in test/automatic test equipment/manual testing?
- Is automatic test equipment being selected/designed concurrently with the prime system?

2.3.1.10. Built-In Test

The continuing increase in complexity of military systems has imposed additional operational, maintenance, and logistics burdens on our military organizations. Unfortunately, these organizations are concurrently experiencing a reduction of both manning and skill levels of operators and maintenance personnel. Consequently, Built-In Test (BIT) monitoring and fault isolation capabilities must be incorporated as integral features of system design. BIT should also be used as part of the manufacturing process to verify proper functioning at various levels of assembly.
BIT considerations:

- Have BIT design details been identified and included as part of initial design efforts?
- Have BIT requirements been passed on to subcontractors and vendors?
- Have production test and integration personnel been involved in initial BIT design and trade-off efforts?

2.3.1.11. Configuration Control

A smooth transition from development into production requires that the design be documented and its 'configuration' carefully controlled. Only then can the final planning for production, installation, maintenance, and logistics be completed. Configuration control must be maintained throughout the life cycle of the equipment to maximize operational availability and minimum support costs.

Configuration control considerations:
- Have configuration control procedures been tailored to the product's relative complexity?
- Are configuration control requirements flowed down to all subcontractors?
- Does the status accounting system allow for information feedback from the field?
- Has an effective quality assurance change verification system been established?
- Are technically qualified personnel involved in configuration management?

2.3.2. Design Reviews

Formal design reviews often become a forum for providing an overview of the overall hardware design, rather than an in-depth technical assessment of design maturity. Design reviews must be performed by technically competent personnel if the technical risk of proceeding to the next phase of the development process is to be accurately assessed.
Design review considerations:
- Does the contractor have a policy identifying procedures for internal reviews as well as customer required reviews?
- Is emphasis being placed on technical interchange meetings between contractor and customer rather than large-scale reviews?
- Are reviews being done by qualified technical experts who can challenge the design and assess risks?
- Are technical design review schedules established based on design progress?

2.3.2.1. Design Release

At some point design must cease and the product definition must then be released to manufacturing. Then a detailed design review can occur. A meaningful configuration audit can be performed. Producibility, system operation and maintenance documentation can be evaluated.
Design release considerations:
- Has engineering scheduled design releases with manufacturing and purchasing?
- Has technical risk been judged to be acceptable before design release?
- Does the management system control pre-released drawings?
- Have critical drawings been identified?
- Is the release of critical drawings properly scheduled to meet requirements?
- Does the design release process require concurrent review by all disciplines?

2.3.2.2. Logistics Support Analysis

Logistics Support Analysis (LSA) defines readiness and support related performance parameters for the systems engineering process. It affects the design of the weapon and weapon support system. It provides accurate weapon system support requirements information for use in acquiring operational phase resources.
LSA considerations:

- Does the effort start with the initial design?
- Is LSA integrated into the design analysis and design review process?
- Are the engineering analyses coordinated (trade-off analysis) to achieve a cost-effective impact on design as early as possible?
- Does the LSA provide quantitative parameters used in the system and design specifications?

2.3.2.3. Technical Manuals

The technical manual development process involves the translation of engineering work and design analysis into an operations and maintenance information. The engineering change process must involve the technical manuals. The use of CAD/CAM databases in the technical manual process provides this link. Their use improves technical manual accuracy and preparation efficiency.

The information used to develop technical manuals is usually extracted manually from a variety of sources at high cost and error rate. They are mostly prepared using only word processing techniques. Integrating technical manual preparation with CAD/CAM/LSA databases will improve their accuracy and increase productivity.

2.3.3. CALS Initiative

Closely related with the Willoughby Templates in its intent is the DoD Computer-Aided Acquisition and Logistics Support (CALS) initiative. It strongly influences the Process and Information Architectures. CALS is a cooperative effort to establish a technical infrastructure and standards that will reduce the cost of operating, supporting, and maintaining aerospace and military equipment and assure the timely availability of replacement parts and technical data. It involves government agencies contractors, and virtually all subcontractors and suppliers. The proposed solution is to make product data available in digital form to all who need it.

Although the focus of CALS is on logistics, it requires a computer-based product definition. An electronic format allows the instantaneous dissemination of product data anywhere it is needed. Any paper manifestation of product data is to be generated from the electronic master. The electronic product model is to be created early in the design process and used throughout the manufacturing process. That same data is used by the logistic functions for product

deployment and repair. The scope and objectives of CALS are described in the following paragraphs.

2.3.3.1. Scope

The CALS initiative is divided into three areas:

- Technical database definition and access
  - Product definition database
  - Integrated support database
- Digital data interchange
  - Product definition data
  - Logistics Support Analysis (LSA)
  - Technical manuals
  - Training materials
  - Technical plans and reports
  - Operational feedback data
- Integration of processes
  Reliability and Maintainability (R&M) integration in CAD/CAE

2.3.3.2. Objectives

The stated objectives of CALS are:

1. Accelerate integration of RAM design tools into contractor CAD/CAE systems.
2. Automate contractor processes for generating logistic technical information.
3. Rapidly increase the ability of the DoD to receive, distribute and use technical information in digital form.

2.3.3.3. Core Requirements

Recognizing that CALS cannot be implemented all at once, its implementation is divided into two phases.

2.3.3.3.1. Phase 1

The initial focus is on:

- A few major logistics applications
- Available technology and standards
- A "records transfer" environment

2.3.3.3.2. Phase 2

Subsequent effort will be focused on:

- a wider range of design, manufacturing, and logistics applications,
- more advanced technology and standards,
- a centroid of advanced data models,
- an on-line access environment (contractor and sub-contractors will maintain on-line databases for direct access by the customer).

The Department of Defense is encouraging and actively participating in the creation of an emerging product definition standard, the Product Data Exchange Specification (PDES). PDES compliant products will enable the customer to more readily use contractor product definition data as an integral part of field operations.

2.3.4. Concurrent Engineering

Concurrent engineering is an attempt to shorten the engineering schedule while assuring that reliability, producibility and maintainability concerns are addressed. Much of this can be accomplished by reducing the bureaucracy. More can be accomplished by making designs available before their completion. Various analyses and manufacturing process, tool and fixture design should be performed as much in parallel with the design of the product as possible.

Concurrent engineering challenges the notion that only a designer knows when a design is sufficiently complete for disclosure. Originally, this paternalistic approach was used to minimize the waste of time of an analyst or tool designer when a design was expected to change radically. Gradually it became an excuse by designers to hide a design until it was complete to avoid criticism and the burden on change. This *information hiding* makes it difficult for someone to suggest a change that might favorably affect the functionality, reliability, producibility or maintainability of the product before the design is committed to production. Designers are saved some embarrassment and the time required to change their designs, but the quality of the product suffers.

The reality is that no one but the user of the information knows what might be of value. It is the responsibility of the information user to decide when and when not to invest time in a premature design and risk the resource costs consumed by the effort. Even if a design has only one line or arc, it may be what is needed for an analysis or a constraint. A designer should only indicate the degree of completeness or the likelihood of change of a design. Information hiding of any kind has no place in the business of the future.

2.3.5. Total Quality Management

The aforementioned initiatives fall under an umbrella initiative known as Total Quality Management (TQM), which promotes continuous improvement. As soon as one improvement is completed, start on the next one.

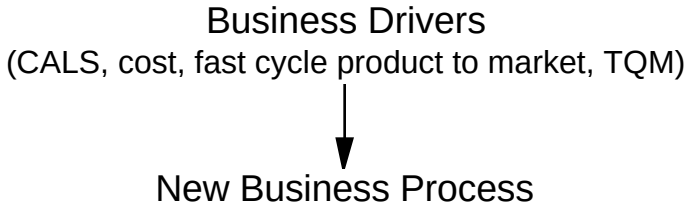
2.3.6. New Business Drivers Summary

Unfortunately for defense contractors, until the transition from cost-plus to fixed-price contracts is complete, the defense contractor must continue to contend with micro-management problems. This is especially true of labor accounting. Having to contend with the worst of the old world as well as the worst of the new world exacerbates the survival problem for defense contractors. Commercial

companies have no constraint on their adoption of new business processes other than their own bureaucracy.

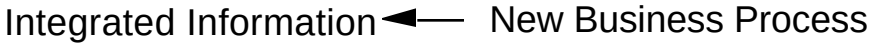
3. Basic Relationships

The new business drivers, competition and common sense dictate fundamental changes in the way business is conducted. The design and manufacturing process must be simplified and when practical, automated. The new business process accommodates concurrent engineering and disciplined product management. It allows the highest product quality to be attained. It can be used to identify, fix and learn from quality problems so that the business can continuously improve.

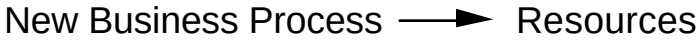


This new business process has been generalized and described in the Generic Process section.

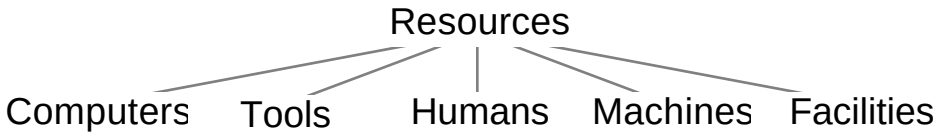
Derived from this generic process are the necessary data relationships needed to conduct the process. To conduct concurrent engineering, the process management and product definition data must be shared among the participants. To support the process, the data that results from the process must be readily shared and yet be adequately controlled to assure that access is authorized and that the version of data accessed is appropriate for its use. Past attempts to relate subsets of the program and product definition data have resulted in complex data relationships, which prove to be unnavigable. A relatively simple method of relating this data is described in the Information Integration section.



Skills are required to conduct the process. These are embodied in resources.

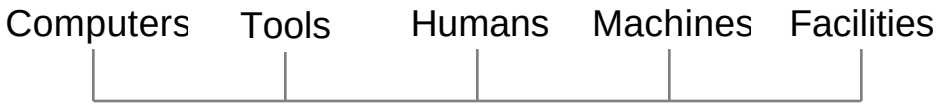


The computer, tool, machine, human or facility resources required to design and manufacture a product may be owned by the prime contractor, a subcontractor or a partner.



The resources are interdependent. For example tools cannot be useful unless employed by computers, machines or humans. Similarly, human resources are

more productive when they employ tools and machines. Computers, tools , machines and humans need facilities for protection, sustenance and a home.

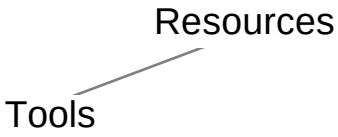


The rapidly changing marketplace demands that product development efforts (Programs) rapidly grow and contract. Programs or portions thereof may need to be conducted with various degrees of secrecy during portions of the life-cycle of the products. Key resources often work on many Programs and cannot be re-located. To support this business environment, a network of computing resources that can be quickly scaled to meet the affordable performance needs of each Program. These computing resources are described in the Computing Resources section.

Information integration imposes additional requirements on the computing resources (distributed data management, shared data, product and tool configuration management, version control…). To make the computing resource requirements as simple as possible, it may impose certain restrictions on the computer-based tools (Unix/POSIX).

A *backplane* or *framework* concept is the most advanced approach to integrate the process, information and resource aspects of an enterprise. Like the backplane of a computer with a series of printed circuit assemblies inserted in it that can be readily replaced with other circuit boards as needed to adapt to changing needs, the framework provides for the control and exchange of data among the tools "inserted" into it. The tools are described in the Tool Resource section. The framework also has a *frontplane* that provides a consistent interface to the human and machine resources utilizing the framework.

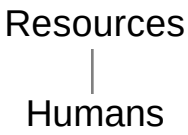
Integral to the current and future process are tools.



These may be drills, milling machine cutters or computer-based tools, known as application programs or software tools. Application programs are currently dominated by analysis, simulation and design tools. Key to the development of a complete and unambiguous product definition is a solid modeling tool. A modern set of computer-based tools is required to capitalize on solid models

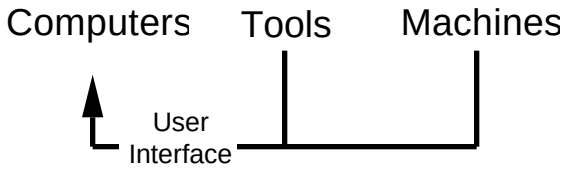
and maximize the productivity of each employee. These tools are described in detail in the Tool Resources section.

The key resources are human. Without them, none of the other resources can be directly or indirectly made to work.

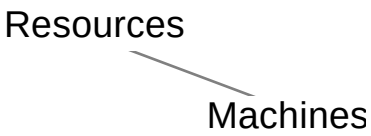


The training of humans is recognized as the largest cost of any process change. That cost is often aggravated by employee turn-over. The need to train employees to use many different tools, and to be re-trained in their use as new versions are released further aggravates training costs. The changes in roles, responsibilities and organization of human resources required to support the new business process are described in the Human Resources section.

To maximize individual productivity and minimize training costs, a user interface that provides common system functions in a consistent manner is required. This user interface requirement impacts the requirements for both the computer tools and the computers on which the tools must run.



Machines are most important to the manufacturing aspect of an enterprise. They move the end-effector to cut the metal. They clean, insert and apply the solder to assemble printed circuit boards.

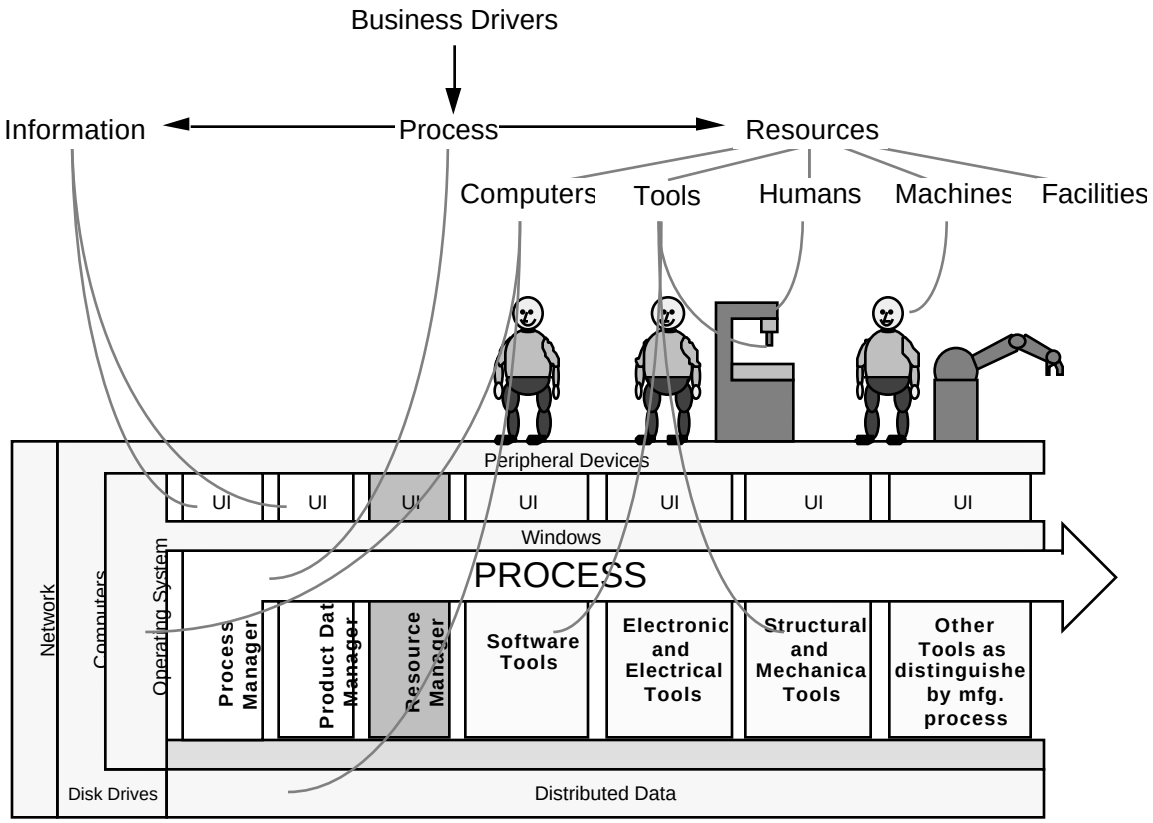


They make the human resources that make them and control them more productive.

Facilities provide a secure and comfortable place for all of the other resources to call home.

Facilities supply sustenance (water, oxygen, oil, food, fuel) and remove waste (sewage, removed material). The Facilities required to support the new business process are described in the Facilities Resources section.

How each of these may manifest themselves is depicted in the following diagram.



The tools, humans and machines are "inserted" in or "removed" from the framework as the need arises. Each may be readily interleaved in the process. The framework approach is one of many aspects of computing described in the Computing Resources section.

4. Generic Process

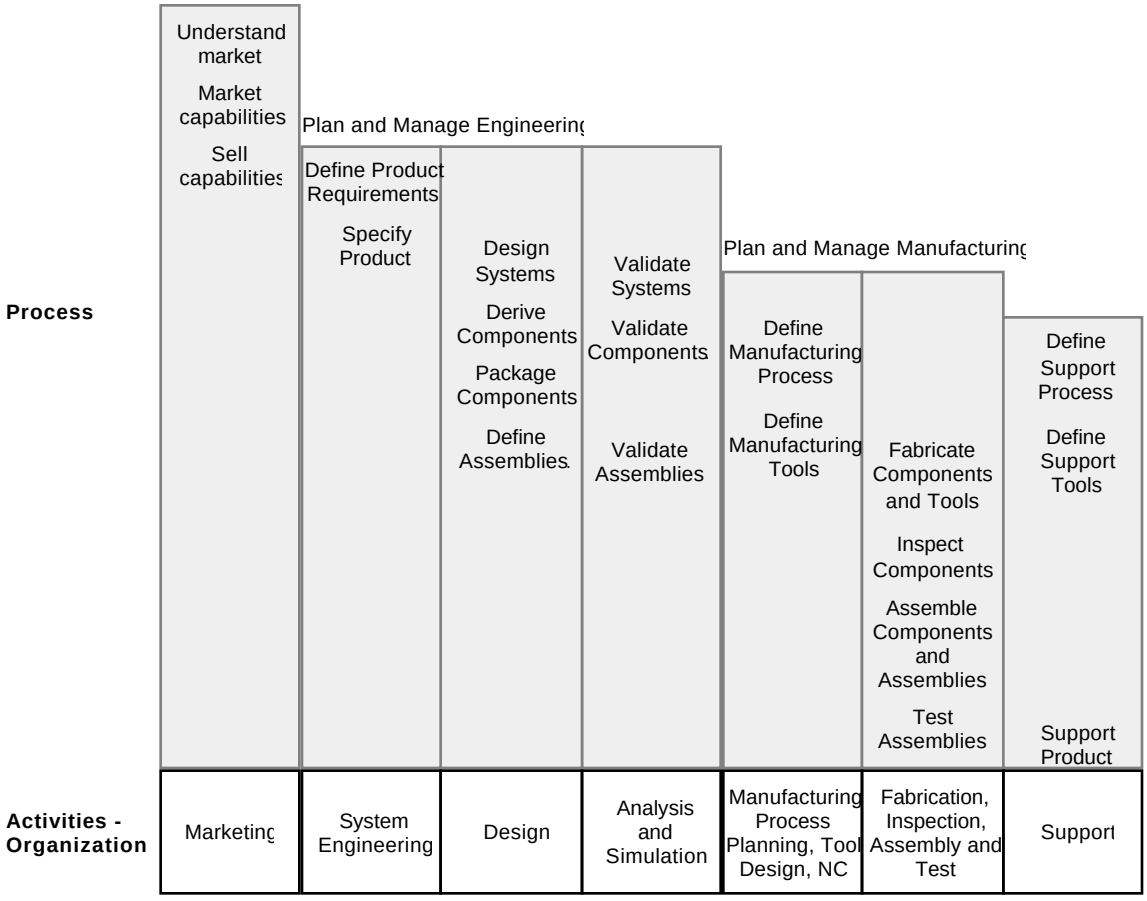
All of the bad practices of the current business process, like inappropriate or inadequately defined requirements or products, serial development, islands of automation, authoritative management, micro-management and their adverse effects on the cost of product ownership, market share and profit alluded to in the Introduction could be delineated at length. So too can their specific remedies like employee empowerment, concurrent engineering, multi-media communications, data and tool integration. However, the most meaningful way to relate the information in this book is by way of the product development and manufacturing process.

The author has developed his share of activity and information models. Most were done to at least partially describe the as-is process for a specific business. For its details one must peruse the flow diagrams buried in standard practice manuals and situation-specific addendum. The as-is business process for a major enterprise is too confused, bureaucratic and convoluted and situation-dependent to be comprehensively described. That futile effort will not be repeated here.

Instead a generic process is used as the framework for this book. The generality of the process is a function of the degree of detail (granularity) at which it is described. At high levels of granularity, the process is the same regardless of the product. At low levels of process granularity, many of the tasks associated with the process are a function of the product. For example, electronic design and validation tasks and their information and tools are pertinent only to products with electronic systems. Compilation tasks are pertinent only to products with software components.

The data relationships required to support the generic process are described in the Information section. The resources required to conduct it are described in the Resources section. The Resources section is the largest, because it includes descriptions of all the resources and tasks required to conduct the process for a sophisticated product. A product that includes many system types (software, electronic, electrical, hydraulic, structural, mechanical, thermal, aerodynamic …).

Although the process may produce different products or "services", it is essentially the same for all businesses. Some enterprises are involved in the entire process. Others specialize in a part of it. Each activity described can be replaced or augmented with the purchase of the deliverables that would otherwise result from the activity. The process described here has no regard for the quantity of product instances made or their intent (prototype or production). It is generic.



4.1. Plan and Manage Engineering

Before the process can begin for a new product or the modification of an old one, it must be planned. The funds and resources must be allocated, scheduled and applied to at least the most immediate part of the process. Large plans should be decomposed into projects, tasks, subtasks until the work defined is small enough for an individual resource to perform. This hierarchy of work is entitled the *Task Breakdown Structure (TBS)*. It is the product most used by product development (Program) or project managers. It provides individuals with accurate information about the process so they can coordinate their activities without the need for management intervention.

Throughout the process there are *deliverables*. They may be information deliverables on various media, or physical deliverables from one specialized

enterprise to another (external deliverables), or from one group to another (internal deliverables). The result of one subtask is a deliverable for another subtask. This interdependence on deliverables requires that the tasks be coordinated. Resource time should not be wasted waiting for deliverables. To compress overall time, as many tasks as practical should be performed in parallel.

4.2. Define Product Requirement

Some enterprises specialize in portions of the process like market research, design, manufacturing or maintenance. Regardless of the extent of their involvement in the process, every enterprise examines its marketplace to determine the requirements for a successful new product or a profitable modification of an existing product. Whether by intensive market research and analysis or intuition, the requirements for a product are delineated. The larger the magnitude of the investment the more formality and detail will be invested defining the requirements.

This is the *Define Requirements* part of the process. These are behavioral requirements. They are usually expressed in qualitative terms that describe the function of the product (transport, feed, house, clothe, enlighten, entertain …). This description can be readily communicated between a potential customer and a supplier. It should be as detailed as practical. However, it should not specify *how* the behavior is to be manifested, which tends to bias a design and potentially exclude better designs.

The behavioral descriptions of each aspect of the product may be decomposed into sub-descriptions. The descriptions should include time, sequence and other dependencies. All the descriptions should be organized into a *Function Breakdown Structure (FBS)*. The FBS is the customer view of the product. Each behavioral description is a deliverable to the next step in the process.

The result of this exercise is the "as required" baseline configuration.

4.3. Specify Product

The requirements must be converted from a qualitative description to a quantitative specification before meaningful design and analysis can be performed. If a product is to be used in an office environment, that environment must be described in terms of temperature, humidity, shock, vibration and available electrical power. If a product is to be eaten, the allowable limits of its weight, color, texture and moisture content must be described.

Specifications will often conflict, so the sophisticated enterprises with longer time-to-market opportunities will correlate specifications to detect those which conflict. Then the product will be re-specified such that the net effect of the specifications will likely be an optimum product. Although the conflicts can be

resolved during the *Design System* part of the process, specification compromise is less expensive than resolving design conflicts later.

The product specifications initially correlate with the behaviors they are to exhibit. Consequently, the specifications should be organized in a *System Breakdown Structure (SBS)* like that of the FBS to provide a home for each specification deliverable. The SBS is the engineering view of the product. As the product behavior is described to more detail, the product specifications can be specified to more detail.

The result of this exercise is the "as specified" baseline configuration.

4.4. Design Systems and Derive Components

Candidate systems are defined to meet the specifications. These may be electronic, software, hydraulic, structural, mechanical or any combination of many system types. As the product systems are specified to more detail, the systems that are to perform the behaviors can be defined to more detail and exactness. Initially, the system definitions may be manifested as simple schematics. Ultimately, they may be manifested as complex solid models. In any event, the systems must be decomposed into subsystems and the subsystems decomposed into components, each with a corresponding set of specifications and manifestations. The system design deliverables are associated with the same SBS nodes as their specifications.

Components may breakdown into primitive solids or into surfaces called *boundary elements*, or both. These representations are organized in a *Component Breakdown Structure (CBS)*. The boundary elements may be grouped in to *features*. The CBS provides an engineering or manufacturing view of components as a function of the features defined. To analyze the structural, thermal, electrical, acoustic and other kinds of interaction among systems, subsystems and components, their connectivity must be known. *Connection features* may be defined for this purpose.

4.5. Verify Systems and Components

Each subsystem is analyzed to determine if it will in fact meet its specification. Subsystems are compared to determine which is the better one. This comparison is done firstly with isolated subsystems and secondly with the subsystems in combination with other subsystems. Ideally each subsystem is evaluated on the basis of its impact on the entire product, because subtle subsystem interactions can significantly affect the total product cost, performance or other selection criteria. Unfortunately, the cost of fully integrated analysis is such that more myopic comparisons may be conducted. Entire product configurations are similarly compared to determine which is the better product relative to the specifications. Unselected systems should be retained in the event that the selected design is found to be unsatisfactory later in the process.

4.6. Package Components

As component designs are validated relative to their specifications, the components are arranged spatially. Like puzzle pieces, electronic components are arranged in two dimensions to maximize their density without violating connectivity constraints. Thermally sensitive components may be forced to be as far as possible from hot components. Radio frequency sensitive components may be forced to be as far as possible from those that emit electromagnetic radiation.

Similarly, three-dimensional models of circuit board assemblies and hydraulic, electric, fuel and other system components are spatially arranged for maximum product density, optimum weight distribution and other criteria. For example, radiation sensitive components should be separated from radiating components. Less reliable components should be more accessible than more reliable components. The position and orientation of hydraulic or fuel system components that are connected with tubes should have priority over the position and orientation of electrical system components connected by wires. Similarly, the arrangement of electrical system components connected by fiberoptic or large cables should be given priority over electrical components connected by more easily bent wires. Given this maintainability and producibility criteria, proper packaging is in one sense an extension of the validation part of the process.

This is what is commonly called an *electronic mockup*. Once the components are arranged, they may be modified and structural system components added to facilitate assembly. *Assembly features* (face/face, slot/key, hole/pin) may be defined to facilitate assembly.

The assembly commands of future modelers will reference assembly features. For example, the selection of a pin feature of a part followed by an *insert* command followed by the selection of a hole feature of another part would be all that is needed to properly orientate the parts relative to each other. Even after three-dimensional packaging is as automated as two-dimensional electrical circuit assembly arrangement is today, assembly features will facilitate the definition of assembly instructions for human or machine resources. Grasp and assembly features will be particularly useful vision-dependent robots.

4.7. Define Assemblies

As components are arranged, their spatial tolerances and assembly sequence can be defined. Wide tolerance bands minimize fabrication and assembly costs, but the accumulation of tolerances can make assembly impossible. This organization of assembly sequence and component spatial information is maintained in an *Assembly Breakdown Structure (ABS)*. The ABS provides the manufacturing and support views of the product.

4.8. Validate Assemblies

Ideally, the components are modeled as solids with tolerance envelopes so components that can potentially occupy the same space at the same time can be easily detected. The viability of the assembly sequence and the movement of mechanical assemblies can be similarly validated.

The result of the foregoing four steps is the conventional "as designed" baseline configuration, but often the assembly validation is not done.

4.9. Plan and Manage Manufacturing

Before the manufacturing part of the process can begin for a new product or the modification of an old one, it must be planned. The funds and resources must be allocated, scheduled and applied. As components are defined and arranged, the manufacturing part of the process (purchasing, fabrication, inspection, assembly and test) can be defined and managed by extending the TBS to include manufacturing tasks.

4.10. Define Manufacturing Process

Component *fabrication features* are defined to provide the basis for the fabrication (deposit, mill, turn, weld) part of the process. Fabrication processes require intermediate definitions of the components as they are transformed from workpieces (blocks of material) into finished parts. The addition of coatings, serial numbers and the like, may be part of the process.

Component *inspection features* are defined to provide a basis for inspecting the parts as they are fabricated or purchased. Additional *assembly features*, like grasping features, may be required. System *test features* are defined for assemblies that comprise systems for which there are specifications that can be used as test criteria. All of these feature definitions are associated with the CBS of the respective components. The assembly features are related by way of assembly commands.

The manufacturing operations required to fabricate, inspect, assemble and test the product are added to the TBS as recurring tasks for each part and assembly instance required. Inspection and test tasks may not be required for every instance of a part or system assembly. Process control measures must be established.

4.11. Define Manufacturing Tools

As fabrication and assembly processes are defined, the special tools necessary to hold and cut components during fabrication, hold and probe them during inspection, assemble them during assembly, or connect with them during testing can be defined. Ideally, the manufacturing resources have been modeled as solids. Then the contacting surfaces of the tools can easily be

defined by orienting the worst case tolerance variation of a component or assembly model with the work space of the selected machine, and subtracting both models from a tooling "workpiece."

Manufacturing tools that cannot be purchased must be specified, designed, validated and manufactured. That process is no different than the one used to develop and produce a product. A tool can be an intermediate deliverable within the process or a product in itself, depending on the viewpoint of its producer.

The result of the foregoing Plan and Manage Manufacturing through the Define Manufacturing Tools steps is the conventional "as planned" baseline configuration. The result of all of the foregoing steps is the concurrent engineering "as designed" baseline configuration.

4.12. Fabricate or Purchase Components

As the materials and manufacturing tools become available, fabrication can be conducted according to the operations delineated in the appropriate fabrication task in the TBS.

4.13. Inspect Components

As the components and inspection tools become available, inspection can be conducted according to the operations delineated in the appropriate inspection task in the TBS.

4.14. Assemble Components and Assemblies

As the inspected components and assembly tools become available, assembly can be conducted according to the operations delineated in the appropriate assembly task in the TBS.

4.15. Test Assemblies

As the assemblies and test tools become available such that a system exists, testing can be conducted according to the operations delineated in the appropriate testing task in the TBS.

The result of the foregoing four steps is the "as manufactured" baseline configuration. If the product must be shipped partially assembled or additional packaging is required for safe delivery, there would be an "as delivered" configuration that may include packaging materials and instructions, shipping and handling instructions, uncrating and assembly instructions and assembly tools.

4.16. Plan and Manage Support

Before the support part of the process can begin for a new or modified product, it must be planned. The funds and resources must be allocated, scheduled and applied. Given the product definition, the support part of the process can be defined and managed by extending the TBS to include the appropriate support tasks.

4.17. Define Support Process

As the product instances are tested, they can be delivered. The behavioral requirements delineated in the FBS indicate replenishment needs (fuel, oil, toner). The reliability specifications in the SBS indicate maintenance needs. Accordingly, replacement component and assembly quantities are estimated and added to the number of instances of each required to produce the product. Field and depot replenishment and maintenance tasks and their intervals are defined. Each such task may take advantage of the component inspection and assembly and system assembly test features defined earlier. They can be referenced in the definition of replenishment and maintenance operations. These operations are scheduled and added to the TBS as recurring tasks for each product instance.

The behavioral requirements delineated in the FBS may also indicate likely repair requirements. Repair operations can be delineated and added to the TBS, but not scheduled until the need arises.

4.18. Define Support Tools

As the ABS is defined, the special tools necessary to assemble, disassemble and test assemblies or hold and probe components can be defined. Many of the manufacturing tools can be used or copied and modified for support purposes. Repairs that do not involve the simple replacement of parts (plug hole in hull) often require tools unique to the repair operations.

Support tools that cannot be purchased must be specified, designed, validated and manufactured with a process no different than the one used to define the product and its manufacture.

The result of the foregoing four steps is the "as maintained" baseline configuration.

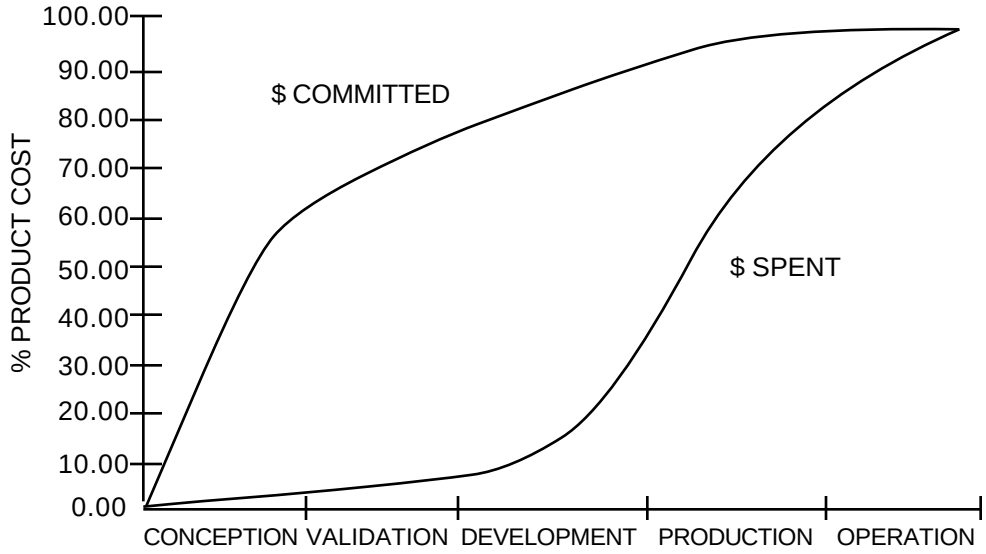
4.19. Support Product

As the supplies and support tools become available, refurbishment, maintenance and repair tasks can be conducted according to the operations delineated in the corresponding support tasks in the TBS.

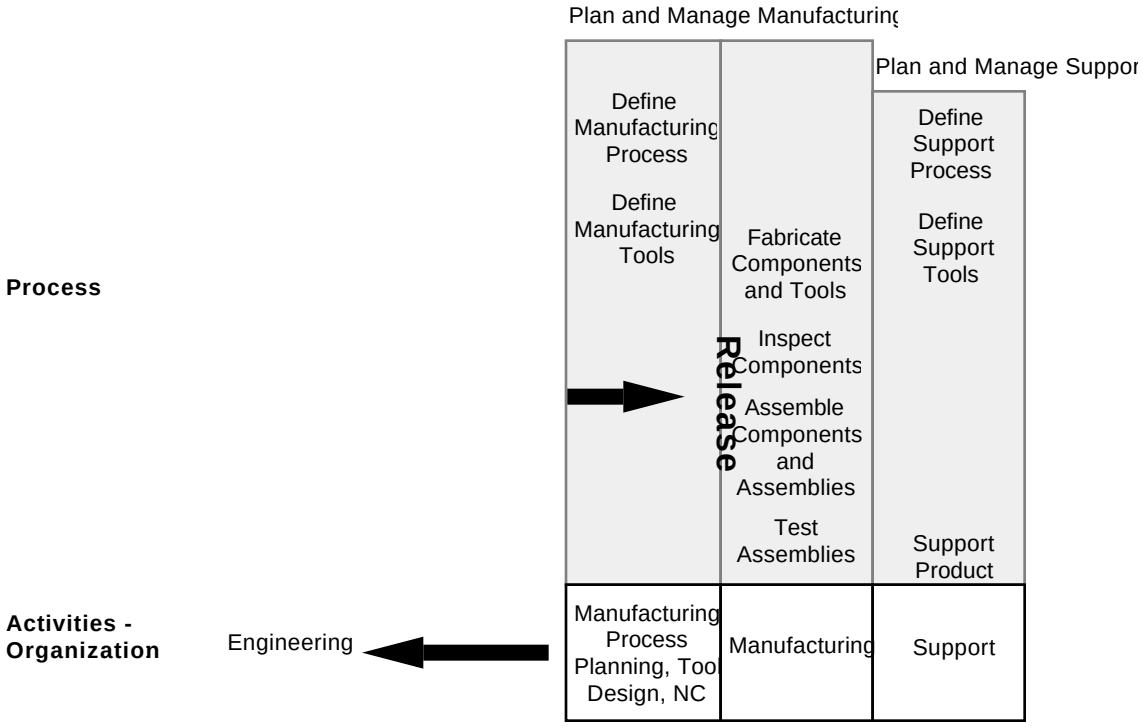
As replacement products become available, dispose of the older models in an environmentally sound manner.

4.20. Process Implementations

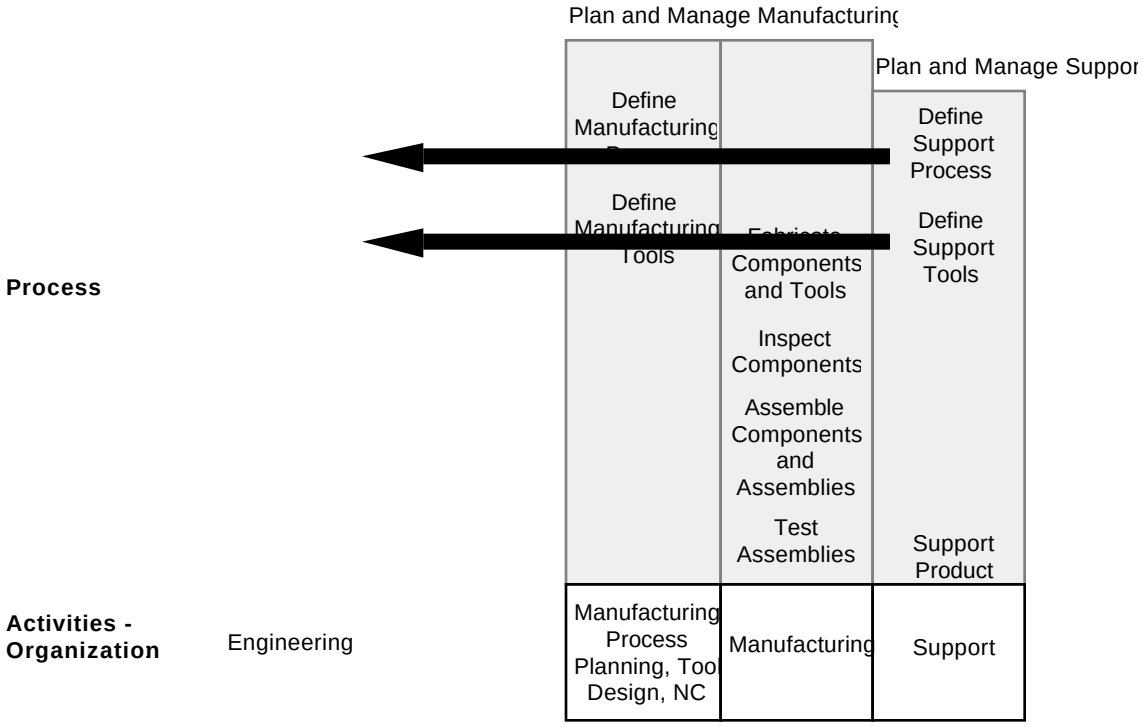
It may not be practical to change the entire process at once, so where should change begin? Should it be in manufacturing where the costs and potential returns are large, or in engineering, where the expense is relatively small? The traditional implementation of the process often delays pre-manufacturing tasks until after a formal engineering release of the product definition to manufacturing, so bad designs are not detected until they are already part of the formal manufacturing process. A bad design that would cost $10 to fix during the design phase will cost $100 to fix during the manufacturing phase, $1,000 to fix after manufacturing and before delivery and $10,000 to fix after delivery. 85% of the cost of a product for its life-time is committed before the release of its design for production.



Obviously, the initial focus of a process improvement effort should be to reduce the bureaucracy and provide the integrated tools necessary to support the thorough definition and validation of a product before its release for rate production. Configuration management, change control, labor accounting and the like should be a natural consequence of using the tools. The product definition must include all of the Pre-Manufacturing activities (process planning, tool design, numerical control programming, etc.) to be complete. This postpones release until all the manufacturing processes and tools are defined and validated. Doing the pre-manufacturing tasks as concurrently as practical with the engineering tasks will maximize the "window of opportunity" for the product.



Similarly, the support analysis should be performed as early as possible to determine the supportability (maintainability, repair) of a design before a commitment to manufacture it is made. The design is not released for production until it has "earned the right to release."



The preoccupation with performing pre-manufacturing tasks during design is most pertinent to low-rate or short-term manufacturing. For high rate, long term manufacturing, it may in fact be more practical to build the factory necessary to produce a market dominating design. Market domination is often worth the capital investment as the Japanese automakers and the Saturn line demonstrate.

4.21. Process Interruptions

Although there is one and only one process, it may have business or contractual interruptions. The assumptions upon which the requirements of a product are based may be rendered invalid by unforeseen events. The process may be broken into segments to provide a pause in the process sufficient to examine the work accomplished, the degree of compliance with the product requirements on which it was based or the viability of those requirements before another major investment is made to continue the process.

A *system requirements review* may interrupt the process near the end of product conceptual design and validation phase before an investment in more detailed design and development of prototypes is made. A *preliminary design review* may interrupt the process near the end of product preliminary design and validation phase before an investment in even more detailed design is made. A *detail design review* may interrupt the process before an investment in manufacturing tools is made.

4.22. Process Iterations

The process described thus far does not include iterations. Iterations are required, because the product development process is one of discovery and optimization. Each optimization disrupts the continuous flow of the process.

The process is like a mountain stream. When viewed from afar it looks like a ribbon of smooth laminar flow in a singular direction. When looked upon more closely, large boulders (process steps) are visible. The eddies (design iterations) induced by the boulders make the stream turbulent. An even closer inspection reveals rocks and pebbles. Around each is also an eddy (analysis iteration). They all blend with and contribute to the eddies around the boulders whose eddies contribute to the behavior of the entire stream.

Move a pebble, rock or boulder and measure the impact of that change as a function of the down stream turbulence. The larger the object or the further it is up stream, the more the downstream effect. Changes to higher level (major system) or earlier product definitions have similarly disruptive effects.

At times two or more streams may merge into one. The turbulence created at their intersection is like the resource conflict and resolution that occurs when two or more products are manufactured in the same facility at the same time.

Each swirl around an eddy is another iteration of the portion of the process. Each iteration delays the delivery of its deliverable. The net effect, however, is a better product.

4.23. Information Hiding

Concurrent engineering is an attempt to shorten the engineering schedule beyond that which may be done by eliminating the engineering bureaucracy, and still insure that reliability, producibility and maintainability concerns are addressed. By making designs available before their completion, analysis and manufacturing process, tool and fixture design can be performed as much in parallel with the design of the product as possible. The notion that only a designer knows when a design is sufficiently complete for disclosure to do anyone any good is inconsistent with concurrent engineering.

Originally, this paternalistic approach was used to avoid having analysts or tool designers invest time in a design that was likely to change. Later, it was used as an excuse by designers to hide a design until it was complete. Unfortunately, that behavior also makes it difficult for someone to suggest a change that might make the product more functional, reliable, producible or maintainable. The designers were saved the embarrassment and time required to change their designs, but the quality of the product suffered.

The reality is that no one but the user of the information knows what might be of value. The information user must decide whether or not to invest time in a premature design and risk the cost of that labor and associated resource costs. Even if a design has only one line or arc, it may very well be exactly what someone else needs for an analysis or as a constraint on their own design. Information hiding of any kind has no place in the business of the future.

4.24. The Goal

The Goal, A Process of Ongoing Improvement (ISBN 0-88427-061-0, PR9510.9.G64G6 1986 823 86-12566 by Elizahu M. Goldratt and Jeff Cox, Avraham Y. Goldratt Institute, 57 Trumbull Street, New Haven, Ct. 06510, 203-624-9026) shatters many traditional beliefs about how to manage a business process for maximum market share and profit. Three parameters are important to this endeavor:
Throughput - the rate at which the system generates money through sales,
Inventory - all of the money the system has invested in purchasing parts for the products it intends to sell, and
Operational Expense - all of the money the system spends to convert inventory into throughput.

To make money, seek to minimize inventory and operating expenses while maximizing the throughput of the entire operation. To accomplish this, one must contend with dependent events (an event or series of events must occur before another can begin) and statistical fluctuations (many events cannot be precisely predicted). Ignore productivity of discrete non-bottleneck events. Minimize bottlenecks (increase their throughput with additional tooling), move them to the

front of the process and/or off-load them to alternate or less-efficient machines or subcontractors. Pace the upstream process rate according to the ability of a bottleneck to accept input from it. Pace the downstream (final assembly) rate according to the ability of a bottleneck to produce output. Pace all processes that run parallel with the bottlenecked process (feed final assembly) according to the bottlenecked process. Don't waste bottleneck time on potentially bad parts: inspect them before hand. Treat the parts that result from a bottleneck like gold.

Process parts for latest product first. Try to meet or exceed delivery schedule to increase customer satisfaction and potentially increase market share.
Decrease lot sizes to avoid in-process inventory and event delay (machine inactivity due to wait for a large lot). Reduce process times:

Setup time - the time a part spends waiting for a resource while it is being prepared to work on the part (usually a small percentage of total time).

Process time - the time a part spends being modified into a new, more valuable form (usually small percentage of total time).

Queue time - the time a part spends in line for a resource while the resource is busy working on something ahead of it (usually a large percentage of total time, especially on bottlenecks).

Wait time - the time a part waits for another part so they can be assembled (usually a large percentage of total time, especially on non-bottlenecks).

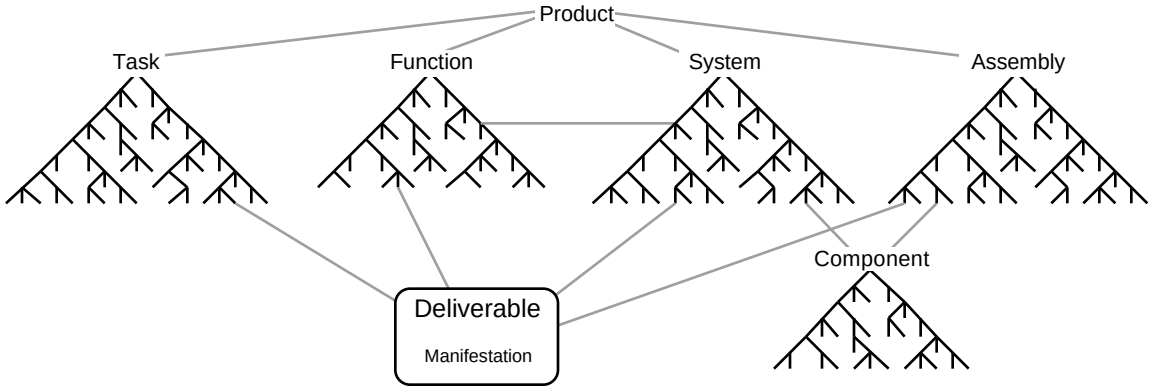
Queue and Wait time — Focus on them. Reducing lot sizes will reduce queue and wait time (usually over half of the total time).

5. Information Integration

To conduct the engineering, manufacturing and support process as efficiently and timely as possible, a coordinating context must be established. That context is represented in the foregoing diagram by the Plan and Manage Engineering, Manufacturing and Support parts of the process.

To support the new business process, data is related by way of the fundamental views of the process and the product. Product development (Program) management views the data from the perspective of the work to be performed. The customer views the data from the perspective of the behavior or functions that the product is to perform. Engineering views the data from the perspective of the systems that are devised to perform the product functions. Manufacturing and Support view the data from the perspective of the components of the systems and their arrangement as assemblies.

These views are respectively called the process Task Breakdown Structure (TBS), the product Function Breakdown Structure (FBS), the product System Breakdown Structure (SBS), the product Component Breakdown Structure (CBS) and the product Assembly Breakdown Structure (ABS). The FBS, SBS and ABS could also be called product behavior, engineering and manufacturing breakdown structures, respectively.



There is a lot of data associated with the specification, design and manufacture of a product, especially one as complex as a modern air vehicle. The relationships among the data are equally complex. With the Breakdown Structures, the data relationships are simplified by maintaining only two relationships for any deliverable that results from the conduct of the process.

A deliverable is anything exchanged between individuals, groups, divisions, companies, prime- and sub-contractors or contractors and their customers. Subtasks in the TBS are often dependent on the receipt of deliverables from other subtasks. A deliverable is related to a bottom node in the TBS and a node in one and only one of the other Breakdown Structures.

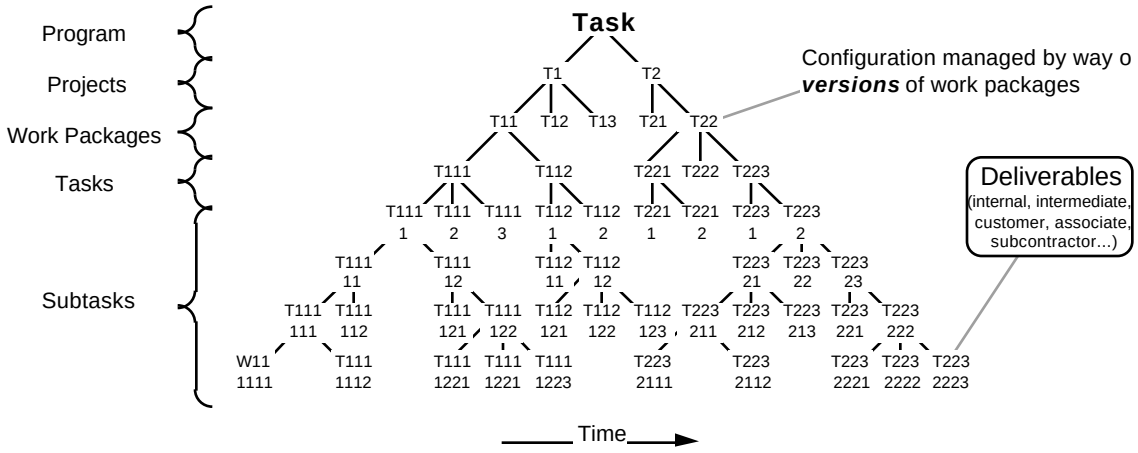
A node in the SBS is explicitly related to a node in the FBS (this system performs that product function). The SBS breaks down into subsystems and components. The components break down according to their CBS. The components are assembled into assemblies which assemble into the product. The SBS and ABS are thereby related to one another by way of their common components.

These relationships make it possible to navigate through the Breakdown Structures to find any data associated with the product (FBS, SBS, CBS and ABS) or the process (TBS) by knowing any one thing about the product or process. Part numbers are not necessary search criteria.

Each of these Breakdown Structures and their relationships are described in more detail in their respective sections.

5.1. Management View

The TBS is a means of coordinating the activity of many and diverse resources throughout the process. Whether the problem is the development of a new product or service or the modification of an old product or service, the problem is typically decomposed into its constituent elements to make it tractable. The TBS is a decomposition of the development process into tasks sufficiently small that they can be performed by a single resource (human, machine, computer...).



To the top node of the hierarchy is related the name of the Program and any appropriate attributes. The purpose of a Program is to develop a product under contract with another enterprise or within an enterprise. Contract information is related to this node. The customer could be the Department of Defense (DoD), the management of an enterprise, the Marketing or Product Planning organizations thereof, another enterprise or an individual.

To make large Programs manageable and to maximize the effectiveness of interpersonal communications, Programs are often divided into *projects*. Some projects, especially those involving changes to an existing design or product

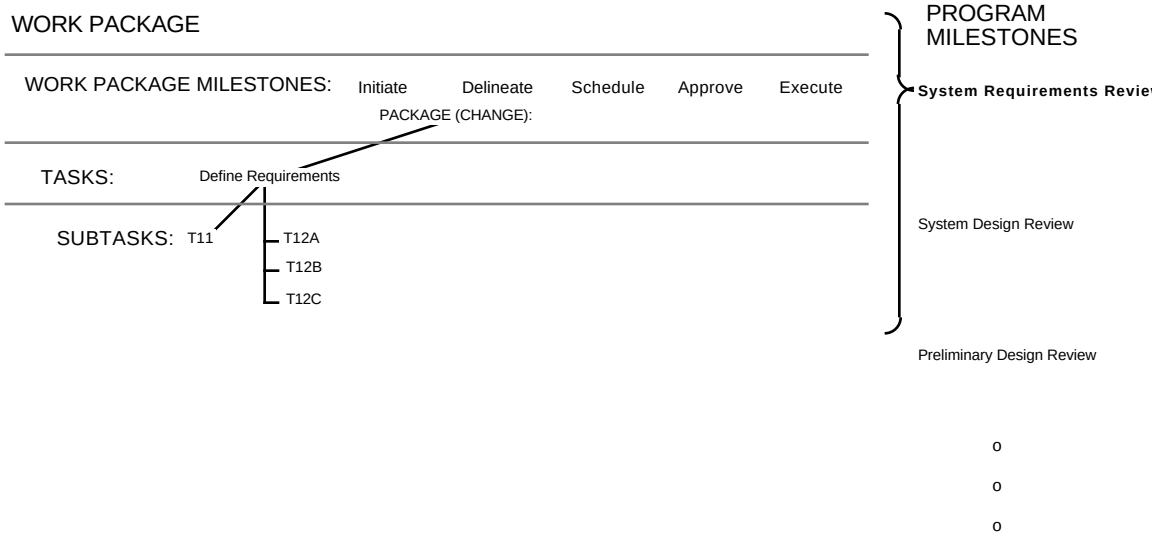
are separate contracts or subcontracts. A project typically contends with one, but may contend with more than one portion of a product.

This portion (system, assembly or group) of a product is characterized in a *work package.* A workpackage correlates with a contract or a portion of a contract (lowest level of a Work Breakdown Structure). The costs of performing the work are cumulated to the workpackage level for accounting or billing purposes. The workpackage costs are cumulated to the project and program levels for project and program management purposes. A workpackage correlates directly with the business process and could consist of tasks that involve any portion of it.

The tasks described in the Generic Process section appear to be limited to sequential processing from the coarse perspective of a product. When viewed from the granular perspective of a component however, parallel processing can be achieved. Subtasks can even be defined for portions of parts if more parallelism is desired.

To further increase parallel processing, sequential tasks can begin before a preceding task is completed. To mitigate the risk involved with such an approach, the human resource responsible for a deliverable to a subsequent task should indicate its degree of completeness. For this reason, a *likelihood of its change* attribute should be associated with every deliverable.

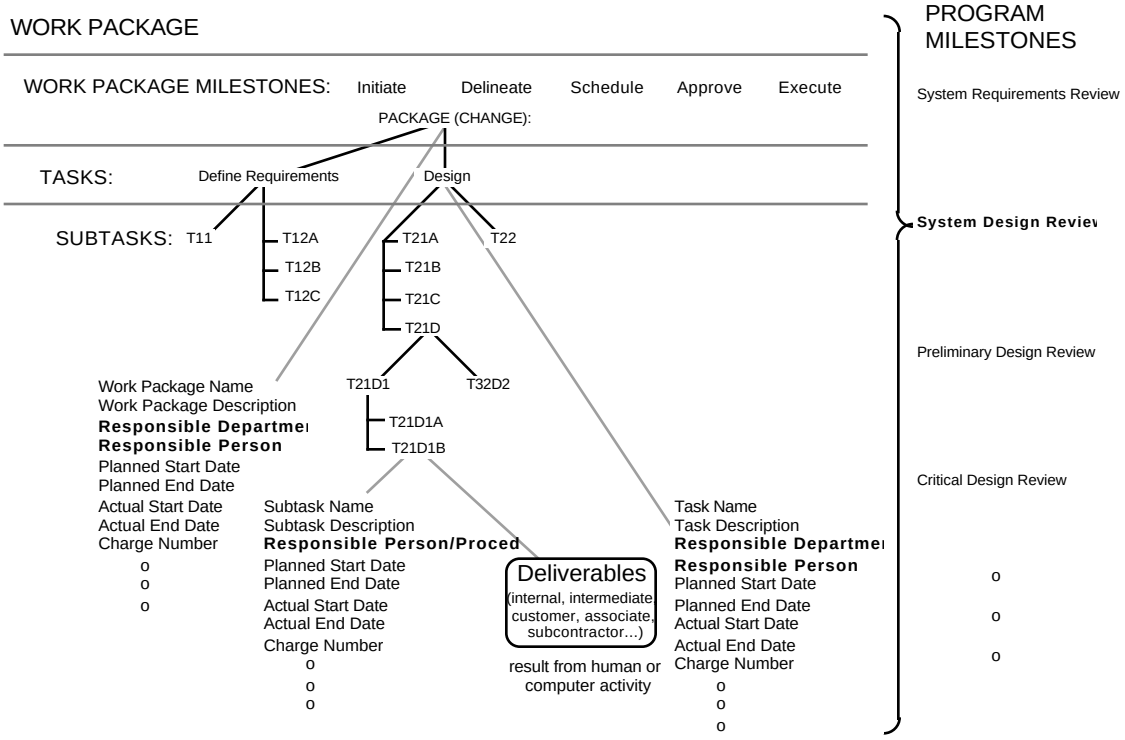
Workpackages have milestones associated with them. They indicate the initiation of the workpackage, the resolution of the work to be performed, the scheduling of the work, the resolution of any resource conflicts, and all approvals required before the work can begin. Ideally, no approvals should be necessary. Individuals should be free to establish a workpackage or task or subtask and begin work when they think it is appropriate.



The completion of the "final" milestone for a task in a workpackage should result in or contribute to the completion of a Program milestone for a phase of the

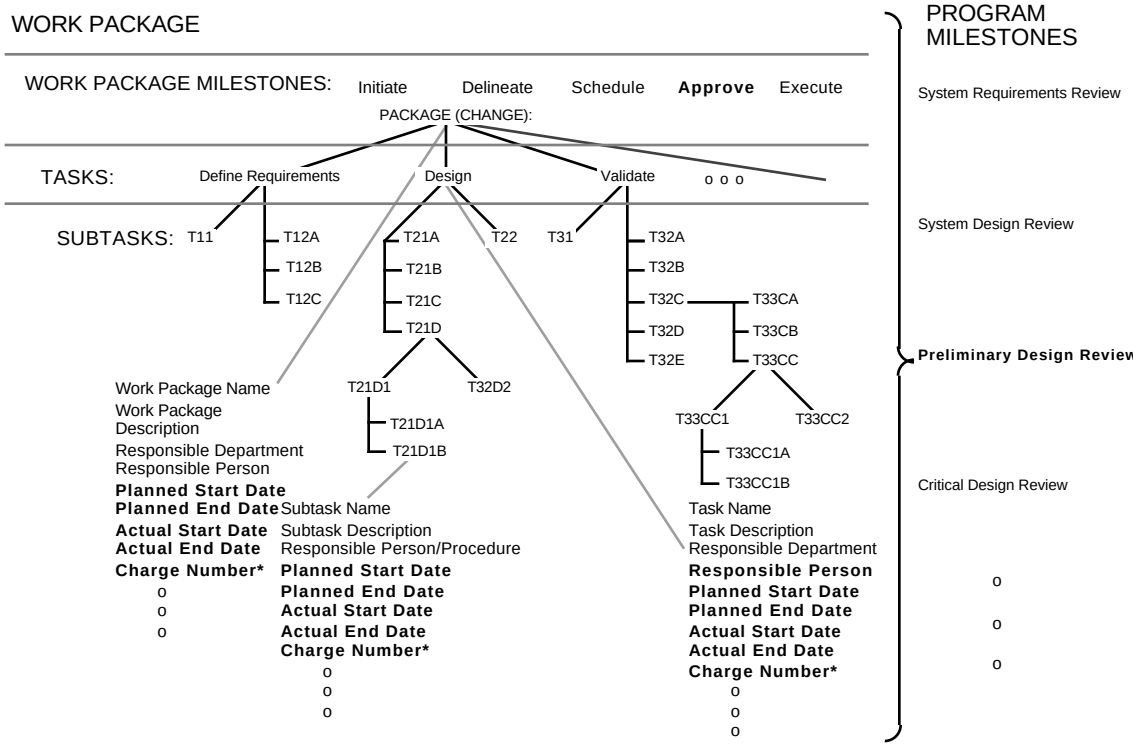
Program. Additional tasks are added to a workpackage to extend it to support other Program milestones.

Associated with each activity is a resource that has responsibility for the activity. A *program manager* is responsible for a Program. A *project manager* is responsible for a project. A *cognizant engineer* is responsible for a workpackage. A *responsible engineer* is responsible for a task within a workpackage. An *individual engineer* or computer program is responsible for a subtask within a task or subtask.



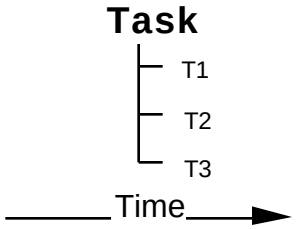
Associated with each activity is a description of the work. There are brief Program, project, workpackage, task and subtask descriptions. The planned and actual start or end dates shown in the diagram could be start dates and durations.

Work should be performed only if it results in a required deliverable to another subtask in the process or to the customer. Associated with the lowest subtask in the hierarchy is a definition of a deliverable and the skills needed to produce the deliverable. The deliverable has a unique and perhaps hidden identifier, a name, a short description and a location associated with it even before it exists. The location is a function of the deliverable type. The location could be a computer file name and path name or a building column number and bin number. When something appears on a computer network or in a bin with that unique identifier, it can readily be identified as the anticipated deliverable.

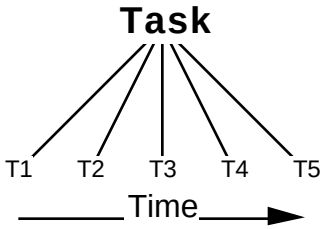


When the subtask is scheduled, a specific resource with the necessary skills is committed to the subtask. Costs are forecast by multiplying the cost per unit of time of a resource by the planned duration of its use. Actual costs are computed using the actual resources and durations. The planned or actual durations or costs can be cumulated up the TBS to whatever level is of interest.

If more than one resource is required to produce the deliverable of a subtask, the deliverable and its subtask should be decomposed. The decomposition can be parallel. For example, each section of a report could be defined as a deliverable (T1, T2, T3) and the original task changed to a compilation task during which the sections are assembled into the originally defined deliverable.



The decomposition can be sequential. For example, intermediate definitions of a part and corresponding subtasks (operations; T1 - T5) that correspond to individual resources can be defined. The result of all of the subtasks is the originally defined deliverable.



The latter approach is common during the transition to production of a product. For example, a workpiece must go through many transformations performed by different resources before it achieves the shape defined by its design. Each transformation results in a new, intermediate manifestation of the intended part that must be handled and sometimes stored. Such synthetic parts or assemblies are given a new identification (synthetic part number) at each such step. Intermediate models of the part may be derived to help the resources recognize when the desired intermediate physical manifestation has been achieved (e.g., model used to establish inspection criteria).

The ability to delineate subtasks to any level of detail permits the responsibility for many subtasks to be assigned to computers. Human and computer tasks can be interleaved, which makes design rule checking (batch analysis invocation), electronic mockup (model transformation invocation) and the like practical.

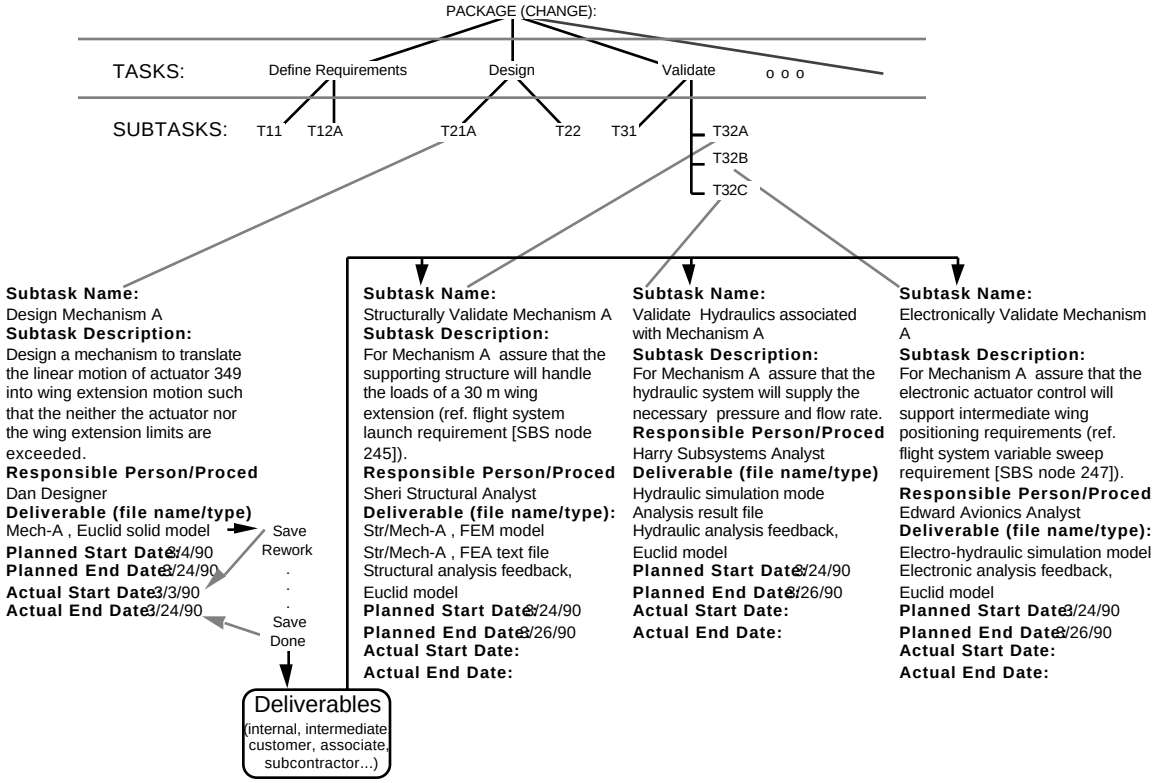
Once a workpackage is initiated, defined and approved, actual work may commence. Labor accounting numbers are accordingly displayed. Some accounts should be provided to which exploratory work can be charged when there is no appropriate product-related account.

If resources are compensated for their time on the basis of time worked instead of deliverables delivered, then some form of time-based accounting is required.

All labor accounting and resource utilization data may be based on the TBS and the actual duration of subtasks performed by a resource. Labor accounting would then be a natural consequence of performing subtasks. There would be no need to display charge numbers for manual charging purposes. There is no need for individuals to report the time expended by themselves or their tools.

Changes to a workpackage are limited to the originators of the information or those who assume responsibility for it.

The lowest level subtasks are associated with one another by way of deliverables as well as their common parent subtask or task. The deliverable resulting from the conduct of one subtask is a required input for one or more other subtasks.



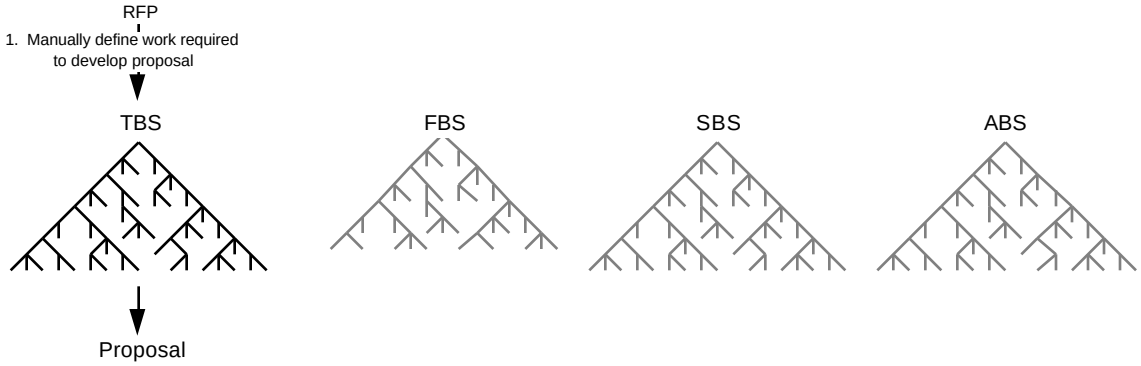
The need for explicit human action to indicate progress relative to schedule can be avoided. Task initiation, progress and completion may be indicated by tool invocation and file saving activity. Although file size as a function of the deliverable type might be used as an indication of the degree of completion, a *likelihood-of-change* indicator manually provided by the author of the deliverable may be more useful to those awaiting the deliverable.

Skills provide a convenient way of establishing "soft links" between work. They eliminate the need to categorize or abstract resources. If a subtask requires a certain set of skills and a certain proficiency in each, then a table of resources can be queried according to the skill/proficiency search criteria of the subtask. If a qualified resource is available during the desired time, it can be committed

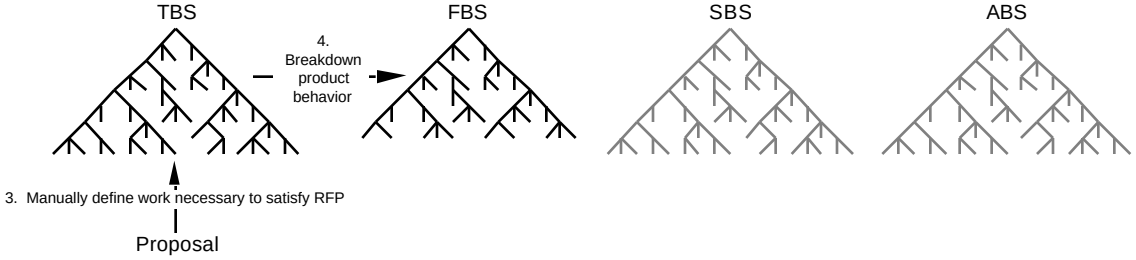
(hard linked) to the subtask. If that time conflicts with a previous commitment of that resource, then a *resource conflict* results.

Since first-come-first-served or who-can-yell-the-loudest may not result in resource assignments that most benefit the enterprise, a means of resolving resource conflicts and reassigning resources must be provided. Resource resolution can be automated by simulating the impact of a delay in the use of the resource on the time-to-market of each product. The estimated profit of each product times the delay will yield the actual enterprise profit impact of each resource assignment alternative.

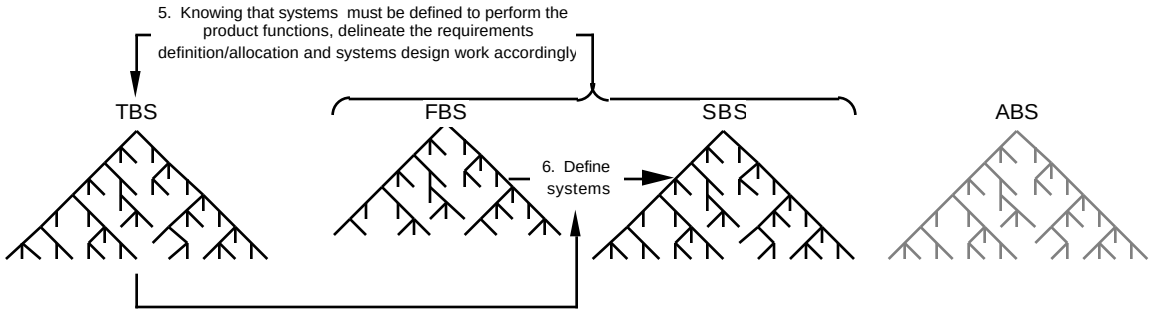
The tasks required to develop each breakdown structure and the deliverables related thereto can be inferred from a predecessor structure. From a Request For Proposal or Product Development Plan can be inferred the tasks necessary to prepare a proposal in response to the request.



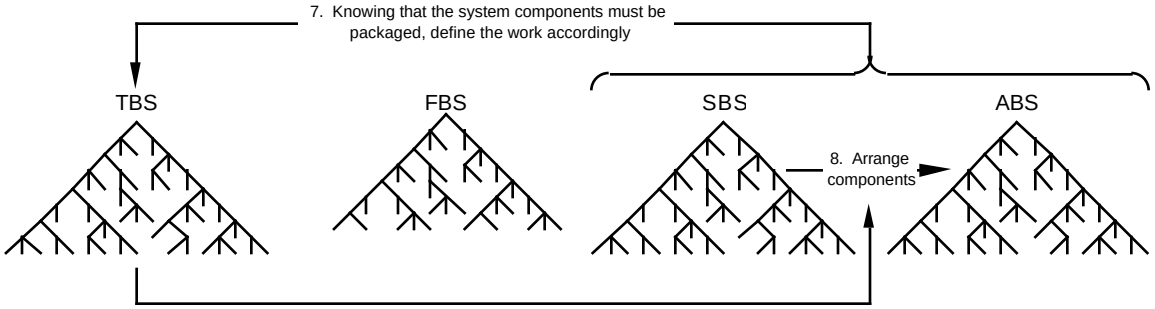
The execution of the proposal tasks results in a significant refinement of the TBS and an initial description of product behavior in the FBS.



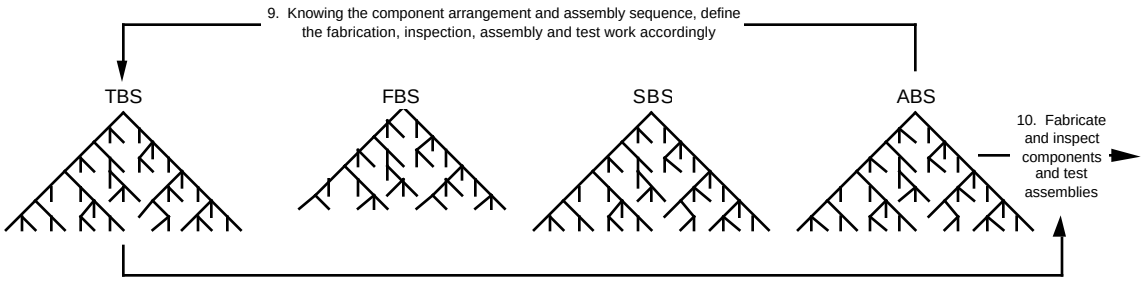
Given the product behavior, the tasks necessary to define and allocate the system requirements, and design the systems and their components according to those requirements are obvious.



Given the product systems and components, the tasks necessary to arrange them within the constraints of the external and internal mold lines are obvious.

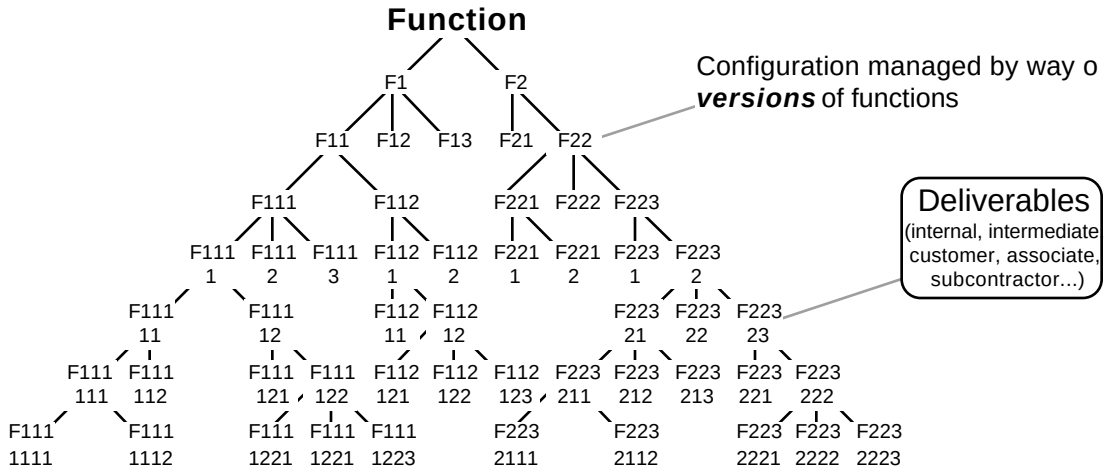


Given the arrangement of the components and their assemblies, the tasks necessary to manufacture and support them are obvious.



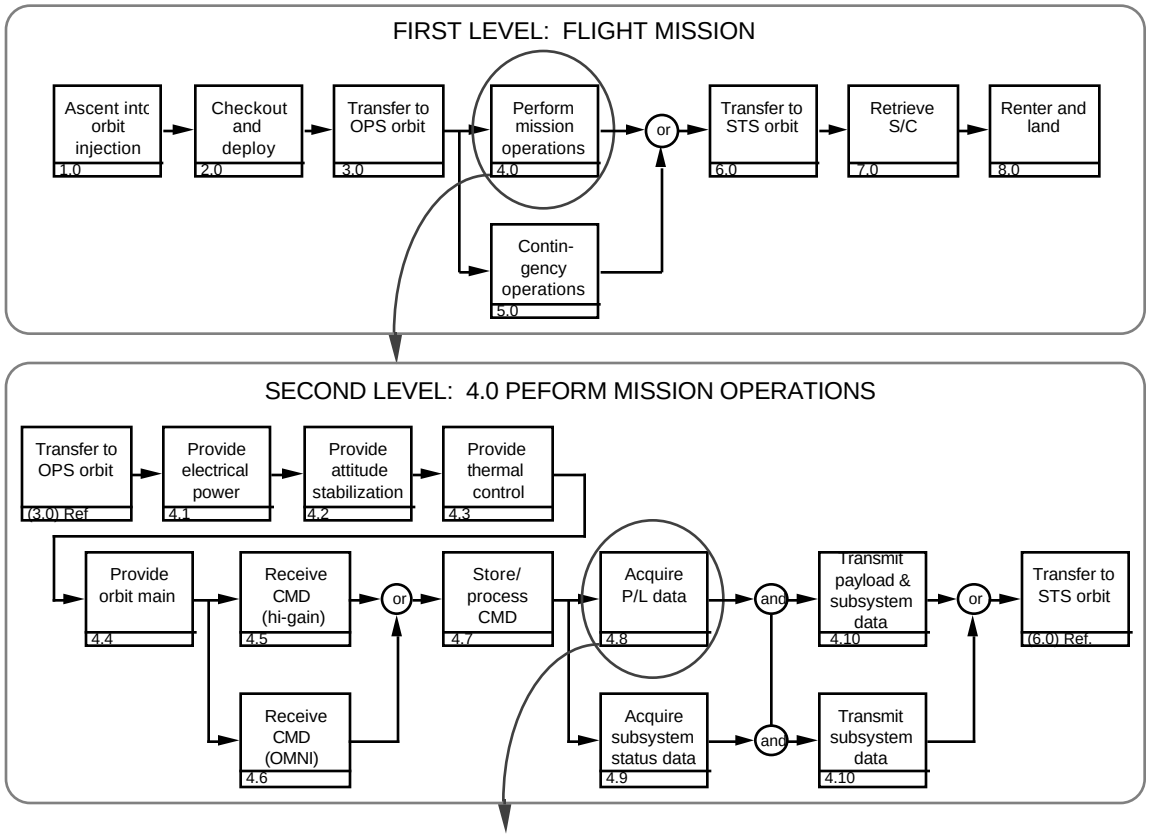
5.2. Customer View

Product requirements are qualitatively described in terms of the behavior or function a product is expected to perform. The function of a product should be broken down to the extent that product functions can meaningfully be converted to system requirements. For example, the qualitative functions of a computer might be: transport continentally and operate in an office environment.



Versions of product functional definitions should be managed such that product versions can be traced to them. Requirements, operational analyses and mission analyses exemplify some of the deliverables that would be associated with the nodes of a FBS.

The FBS is not a simple hierarchy. It is a network of elements with Boolean options and time dependencies. It is the definition of product behavior to the level of detailed operations.

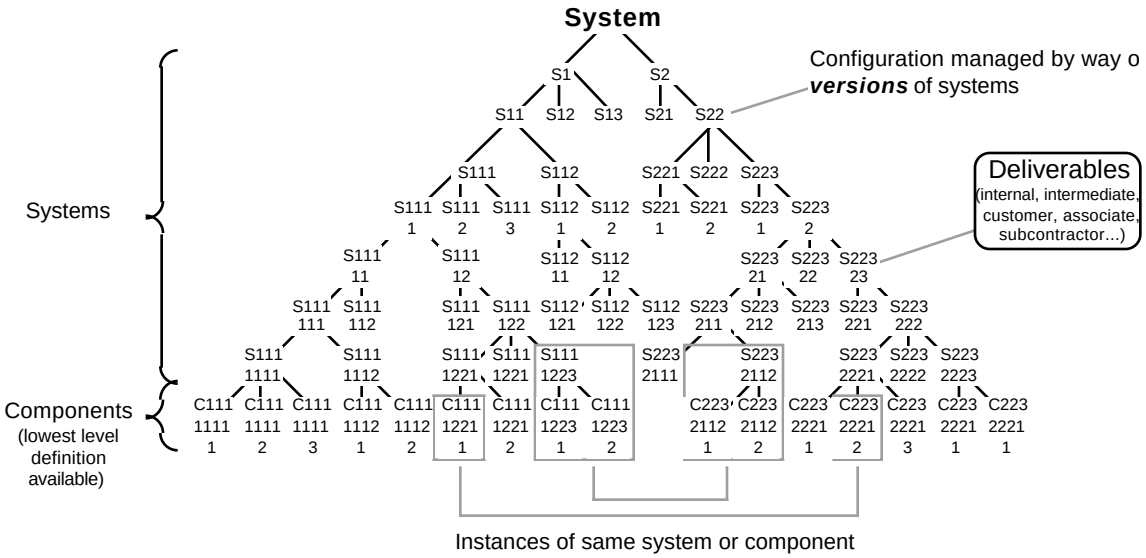


5.3. Engineering View

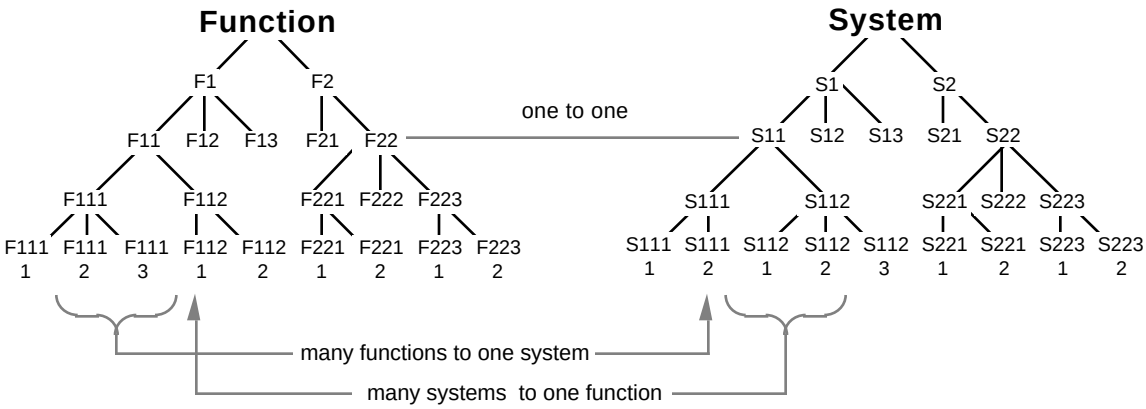
As product functions are defined and decomposed in the FBS, the qualitative behavior can be converted to quantitative system requirements appropriate for the fidelity of the design effort, and can be correspondingly defined and decomposed in the SBS before a design effort is begun. For example, the quantitative system requirements for a computer that is sold in the united States and operates in an office environment might be:

non-operating temperature = -20 to 180 degrees F,
non-operating shock load = 5g in vertical axis and 3g in the lateral axes,
operating temperature = 50-80 degrees F,
operating shock load = 2g in vertical axis and 5g in the lateral axis.

This gives designers constraints and analysts a measure by which the design can be validated.

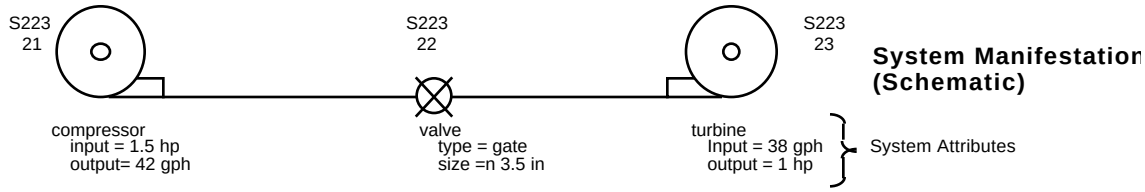


The text and two-dimensional graphic deliverables that define system requirements are intitially associated with the nodes in the SBS that correspond with the nodes in the FBS. A link is manually established between the descriptions of behavior and the systems they specify.



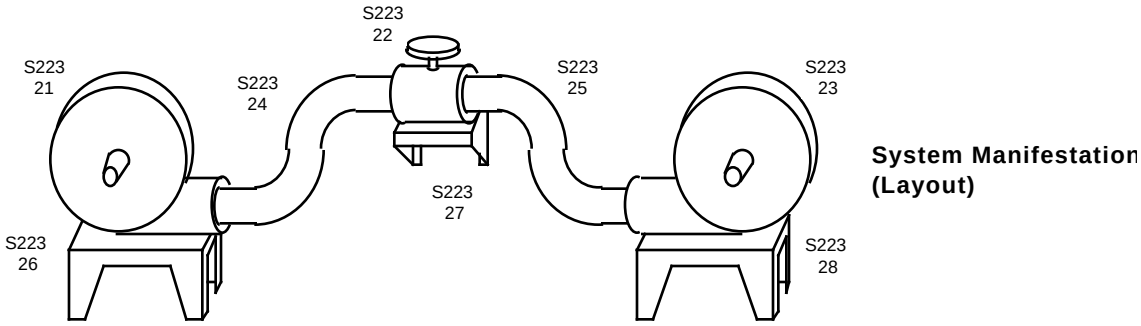
The FBS would be a satisfactory means of organizing the system requirements were it not for the degradation of the one-to-one relationship between product functions and systems into a many-to-many relationship as system design progresses. Designers often realize that if they make a system perform one more operation, it could perform two functions. A set of three systems to perform two functions may make more sense. If two instances of the same system are required to perform two functions, the quantity of the same part is increased. Larger part quantities usually lower per unit cost. These are two reasons for maintaining a SBS as a separate view of a product. A third reason is automated packaging.

The manifestations of the systems (schematics and other two-dimensional models, wire frame, surface and solid three-dimensional models) are deliverables that should be related to the same node in the SBS to which the system requirements are related.



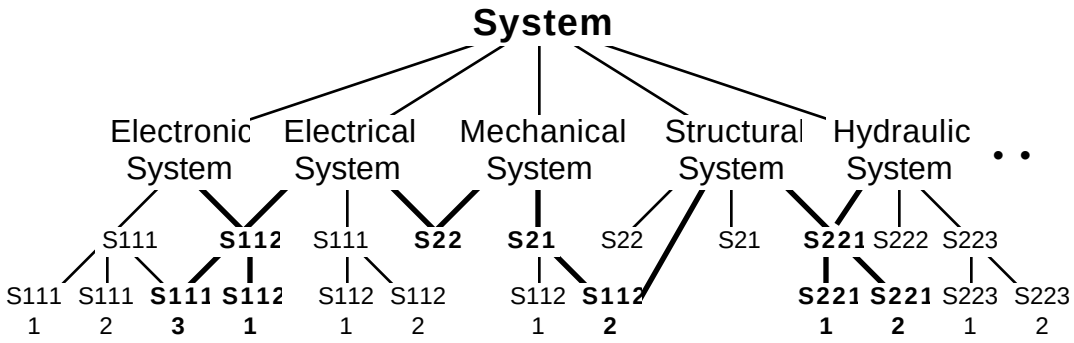
A system may have many manifestations. Early in the design, a simple schematic may be an adequate definition of a system for validation purposes. For example, the impact of the pipe run on the analysis may be small compared to other parameters, so simplifying assumptions about pressure losses can be made. The attributes of the components shown in the schematic may be sufficient for an analysis to compare various system designs to determine which is the better approach (trade study), and worthy of refinement.

As more accurate analyses are required to improve the confidence that the system design will, in fact, meet the requirements, more detailed system manifestations are required. The fidelity of the design and analysis data increases.

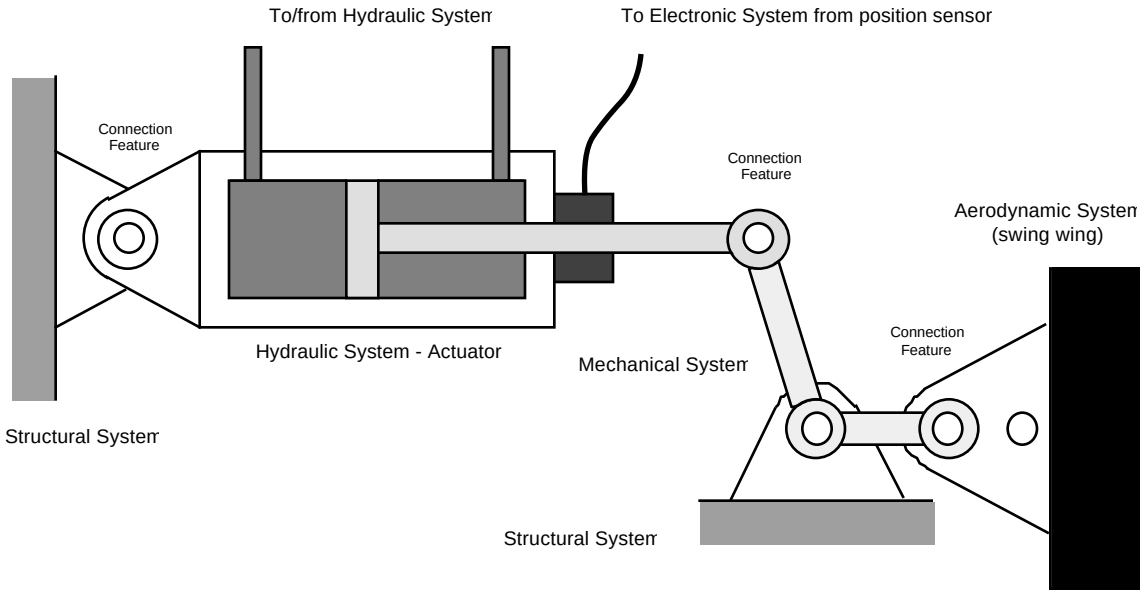


At any point in time, the lowest level in the SBS is a *component*. It may only be the most detailed breakdown of a system available at the time (engine), or it may be a system that will be purchased as an assembly and never disassembled, or it may truly be an indivisible component, which cannot be disassembled without damage. The further breakdown of systems that are purchased as assemblies can be had from the manufacturers of those systems.

The SBS is not a simple hierarchy. It is a complex network of interrelated systems.



To simulate or test systems at each level in the SBS properly, the interdependencies among the components of the systems must be defined. The interdependent components can be included in the simulation or test, or their effect on the system may be estimated and simulated.



Consequently, connectivity information is maintained in the SBS exclusive of any spatial (position and orientation) information.

The maintenance of an SBS as an engineering view distinct from a manufacturing view is normal for software and electronic systems, but unusual for structural, mechanical, electrical, hydraulic, fuel and other systems. It is often erroneously concluded that the components of a system must be adjacent to other components or located in certain areas of a product. Then it is wondered why structural products typically grow during their development when electronic systems shrink. For example, if an airplane is to fly farther, its fuel capacity must be increased, which adds weight, so the wings must be larger, which adds weight and drag to the extent that the engines must be bigger...

One reason why electronic systems shrink is because the connectivity of electronic components is defined before packaging decisions are made. Automated packaging software is then used to position and orient electronic components on a circuit board of a specified shape and size such that the copper required to connect the components and the printed circuit board area is minimized. Other criteria like maintaining a minimum component separation distance for assembly or maintenance purposes, or keeping thermally sensitive components away from hot components, can be used to further constrain an automated packaging tool.

Even for the relatively simple two-dimensional case of packaging circuit boards, automated packaging tools tax computing resources. Three-dimensional automated packaging tools will have to contend with many more constraints and much more complex interrelationships. For example, fuel lines should have routing priority over wire harnesses, assemblies should be oriented to minimize connection lengths, less reliable components should be located where they are more accessible.

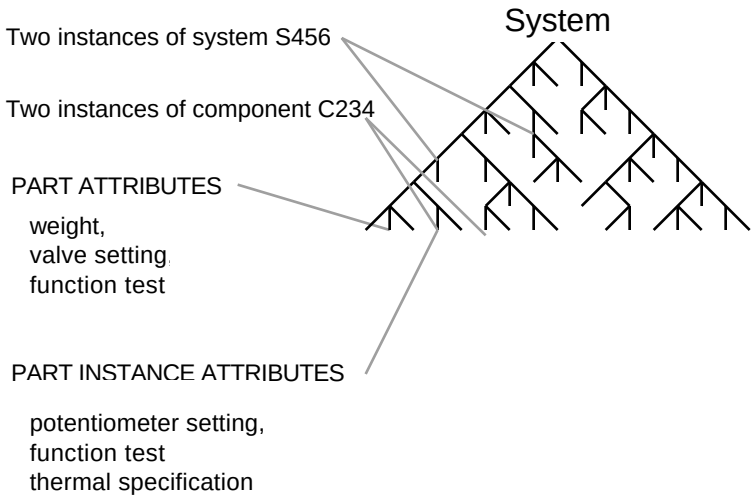
Besides having to accommodate a third spatial dimension, an automated packaging tool should accommodate a fourth dimension – time. In anticipation of the time when computing technology and tools will be adequate to the task of three-dimensional automated packaging, the connectivity of the system components must be maintained separate from their position in assemblies. Until then, this separation of systems and assemblies will promote better manual packaging of products.

5.3.1. Part Attributes (SBS)

Associated with the systems, subsystems and components are part attributes. Part attributes are those characteristics of a part that do not change from instance to instance (use) of a part in the product. Shock, vibration, thermal, electromagnetic interference, electromagnetic pulse and other system specifications for each operating environment and volume and aggregate weight are typical of system and subsystem part attributes. These are properly associated with the SBS node of a system or component. Working fluid, valve or potentiometer settings are subsystem part attributes when there is only one instance of them or all instances are the same. Material is a common component part attribute.

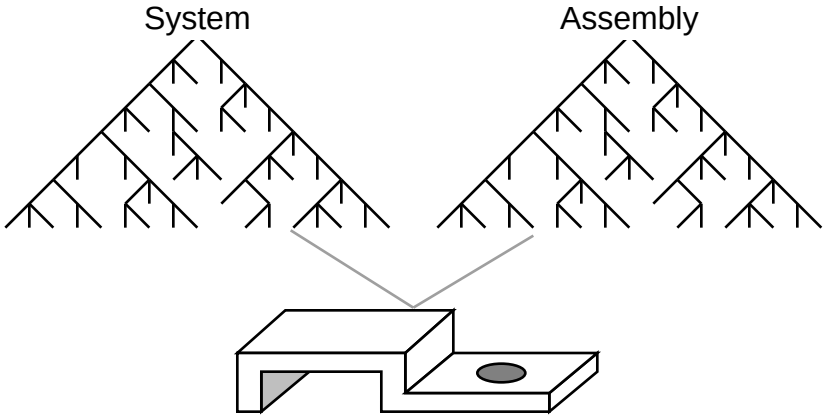
5.3.2. Part Instance Attributes (SBS)

Serial number is a common system, subsystem and component part instance attribute, because its value must vary from instance to instance. Working fluid, valve or potentiometer settings may be part instance attributes if, in fact, they vary among their instances in the product. System specifications may also be instance attributes because they may specify the valve or potentiometer settings.

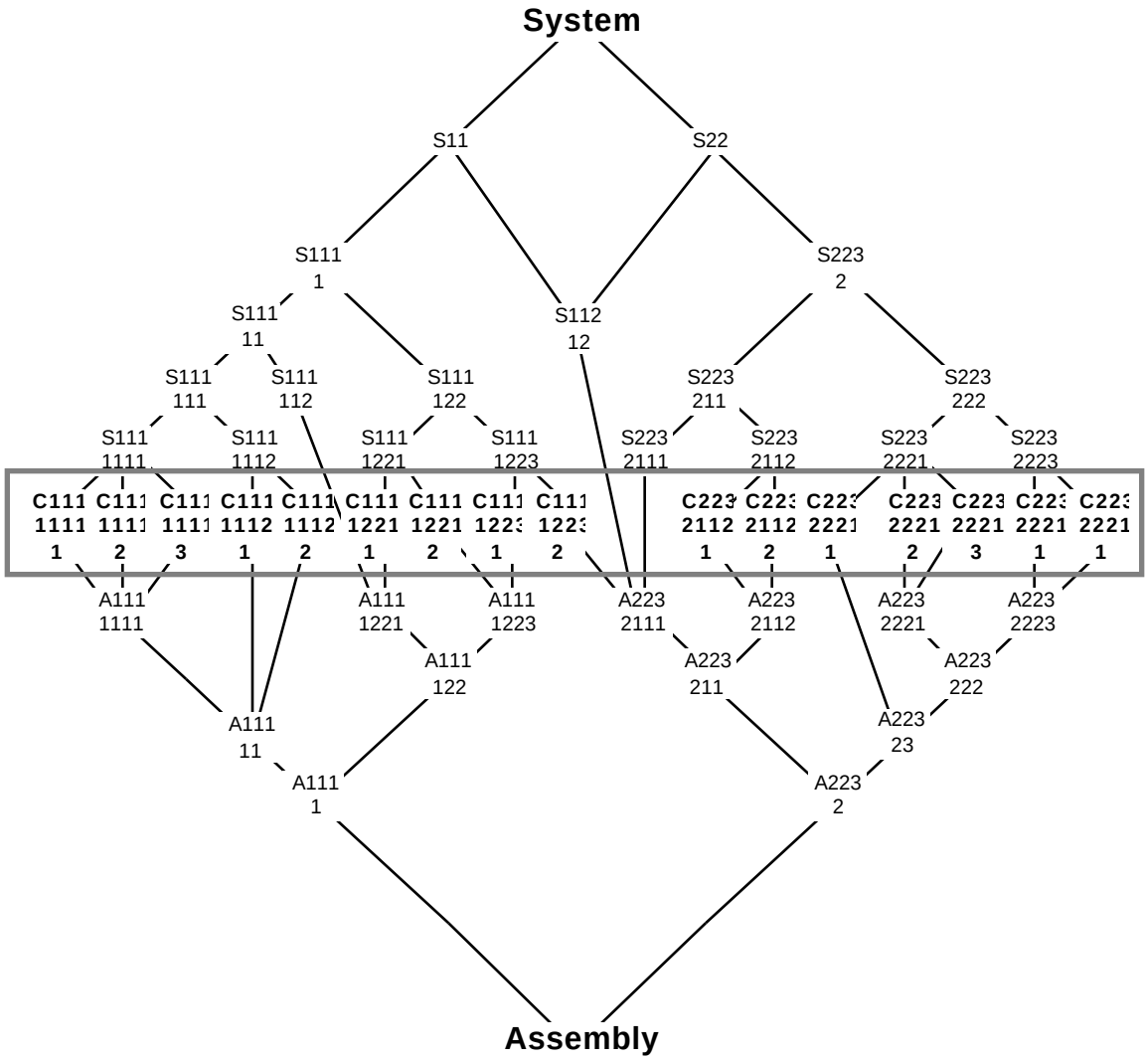


5.4. System and Manufacturing View

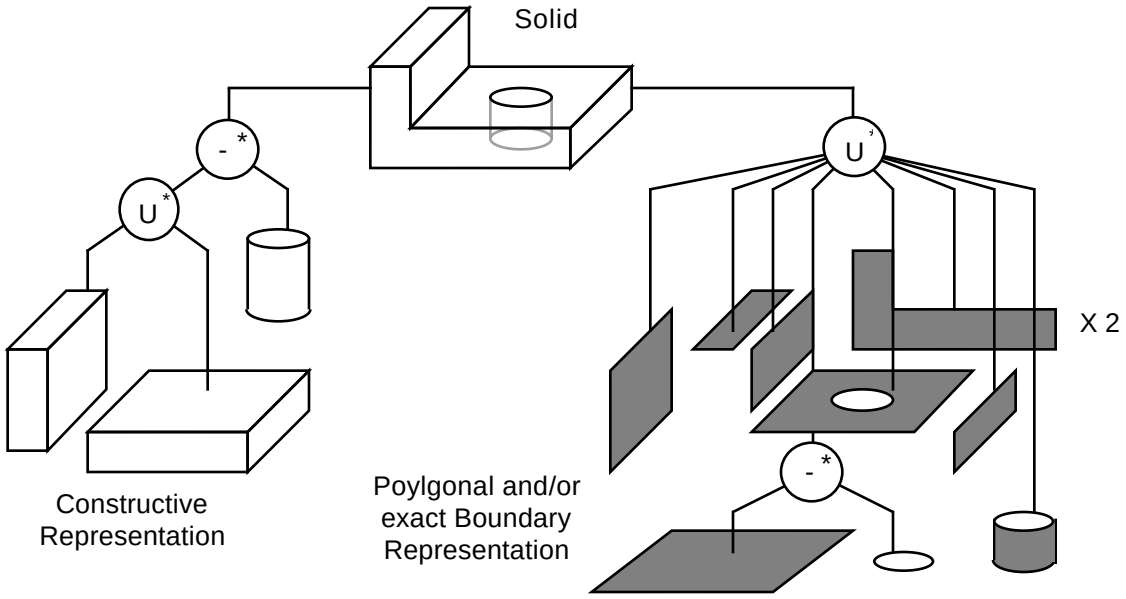
Components are common to the bottom nodes of both the SBS and the ABS.



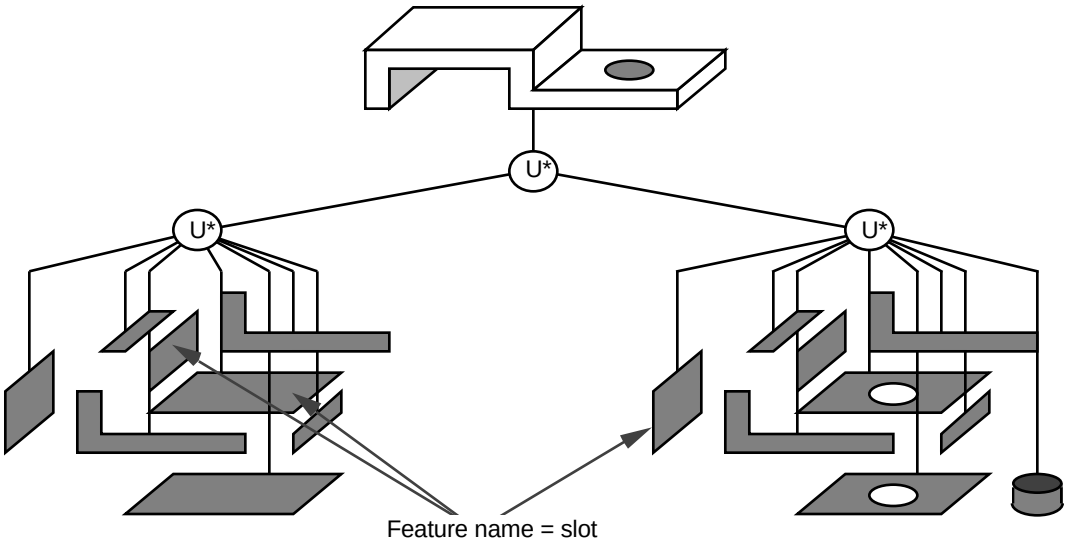
Consequently, they link the two Breakdown Structures.



The components themselves breakdown into primitive solid objects in solid models or surfaces in surface models or lines in wire frame models. Solid models are the least ambiguous. The portions of the primitive solid objects that define the exterior of a part are the boundary elements of the part. The boundary elements are comprised of edges, which define the surface bounded by the edge. The edges are comprised of a chain of lines or curves. The order of those lines or curves indicates the direction to the interior of the part.



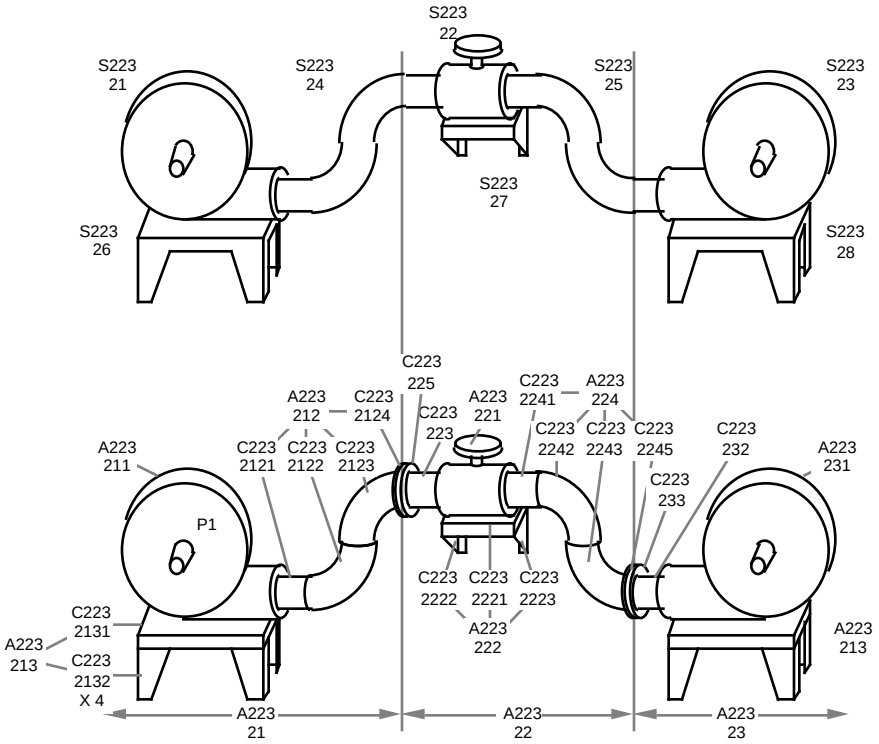
Geometric modeling systems usually maintain both representations. The constructive solid geometry (CSG) representation is better for editing purposes. The boundary representation (B-rep) is better for display and feature definition purposes. Features can be used to designate which surfaces of a part are connected to other parts of other systems. They can be used to designate loaded surfaces and load points for stress analysis. They can be grasping by a robot, clamped for a machining operation, measured during an inspection operation. They can designate what surface will have what finish or treatment (paint) or tolerance. (Features are often what arrows point to on drawings.)



Feature name = slot
Attribute name = surface finish
value =±0.001
units = inches
Attribute name = fabrication cost
value =30.356
units = dollars
•
•
•

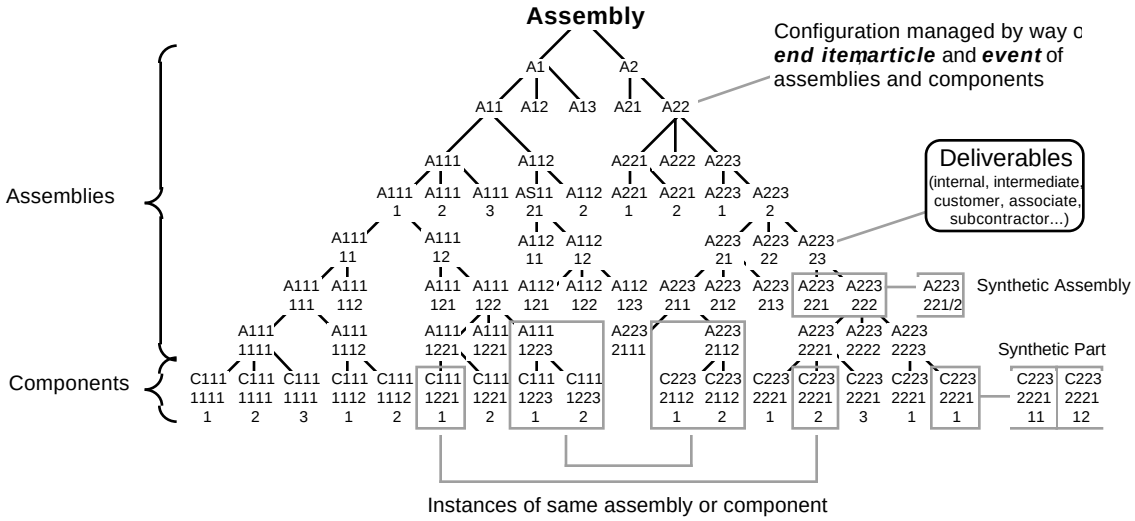
5.5 Manufacturing View

Flight control, fuel or hydraulic systems cannot be assembled as complete systems before being installed as part of the product. Instead they must be assembled along with the components of other systems into subassemblies, until the product is assembled. Some systems cannot exist in a usable or testable state until most of the product is assembled, and all of the necessary electrical, hydraulic and fuel connections have been made.



As the systems are shown to provide the desired functionality, the optimum arrangement of their components within weight, balance, ease of assembly and maintenance constraints is determined. Then the components are grouped into reasonable assemblies and the assemblies are carved into reasonable subassemblies. Additional components, like the pipe flanges (C222124, C223235, C223245, C223233) may be required solely for mating system components that are part of different manufacturing assemblies.

It is in the ABS that the spatial relationships (position and orientation) of the components and the sequence of their assembly are maintained.



As system definitions are augmented by assembly definitions, parts that were originally conceived to be single castings (A223233 in the previous diagram) may instead be assembled from many component parts (C2232131 and four instances of C2232132). Neither they nor the pipe flanges should be treated as synthetic parts.

If the system shown was originally defined as one manufacturing assembly (S223) by the design function, and the manufacturing function had to, in fact, install it as three assemblies (A22321, A22322 and A22323), then the additional part hierarchy indenture level caused by the existence of the three intermediate assemblies could be maintained as a synthetic assembly. However, in the context of concurrent engineering, the design should reflect these manufacturing realities. In the event a change must be made after a design is released for production, the change process should be responsive enough to insure that such synthetic assemblies do not persist for long before they are incorporated into the design. There they can be identified as part of a system and involved in a re-analysis of the system, if necessary, to determine their impact on the performance of the system.

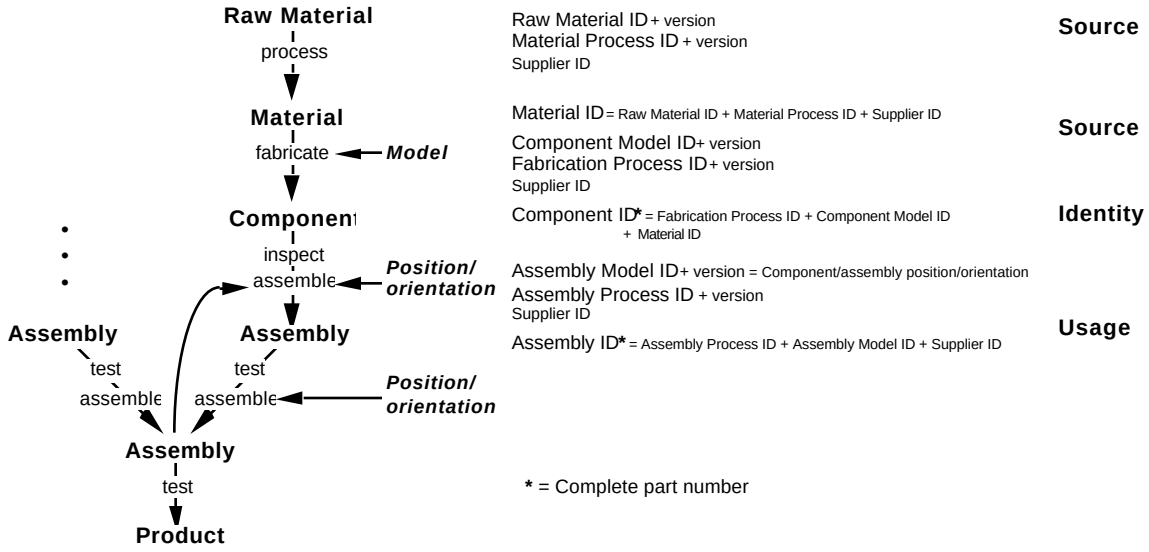
Intermediate parts, like workpieces and incomplete parts on their way to becoming deliverable parts, will persist as synthetic parts. Some assemblies, like those awaiting painting or labeling, will persist as synthetic parts. Most of the intermediate assemblies that are commonly treated as synthetic assemblies, however, should be incorporated into the design of the product.

5.5.1. Part Identification

Material will go through many transformations before it becomes a component part (cut from raw stock, face, profile, pocket, drill, weld, etc.). Although Just-In-Time is a goal, some of these synthetic parts will have to be temporarily stored.

Consequently, synthetic parts or at least their bins must be uniquely identified with synthetic part numbers. These numbers are what distinguish *as-designed* parts from *as-planned* parts. Associated with each transformation is an operation. Operation instructions are found in process plans. Synthetic part numbers should be the process plan number (as-designed part number) concatenated with the operation number.

The following is a comprehensive way of uniquely identifying parts throughout their evolution.

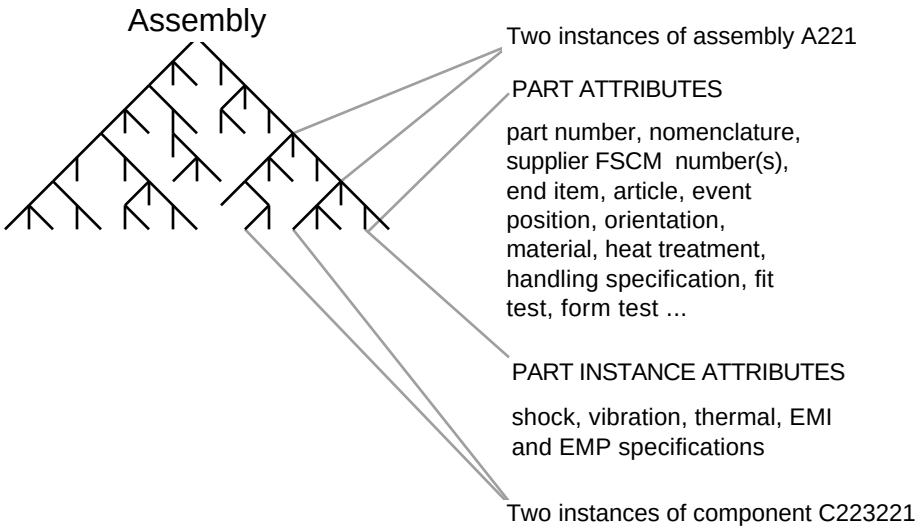


5.5.2. Part Attributes (ABS)

Component or assembly identification and End Item (product model) numbers are part attributes. The supplier of the part may be a part attribute if all instances of the part in the product were provided by the same supplier.

5.5.3. Part Instance Attributes (ABS)

Part instance attributes are those characteristics of a part that are different for one or more instances of the part in a product. Since no two parts can occupy the same space, component and assembly position and orientation relative to the next higher level of assembly, are part instance attributes. If more than one supplier provides the same part for various instances of it in a product, then the supplier name is a part instance attribute. When more than one instance (Article) of a product is involved and their configurations begin to differ, then Article and Event are part instance attributes.

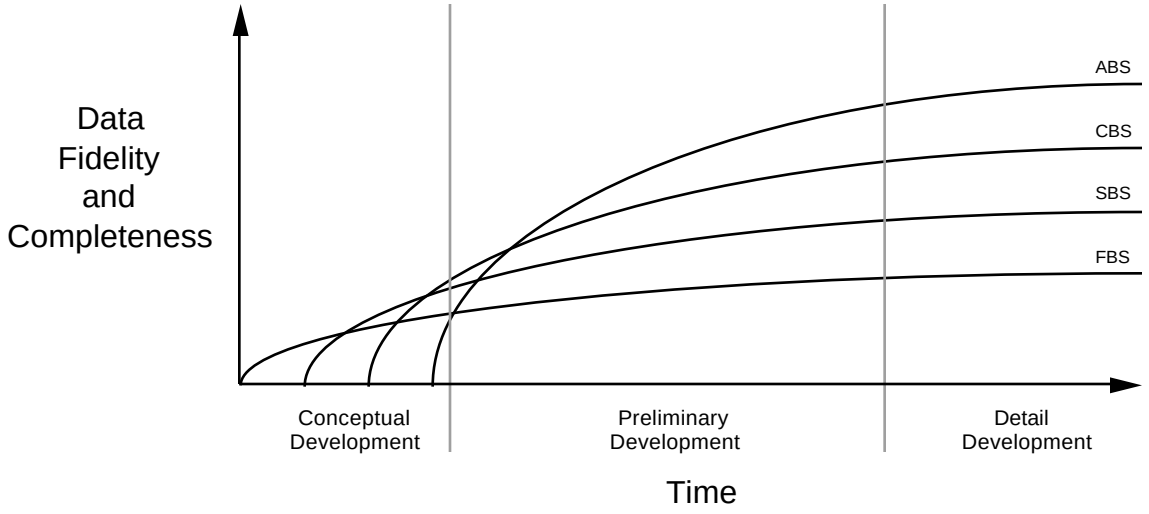


5.6. Support View

The support view is a combination of the customer (FBS), engineering (SBS) and manufacturing (ABS) views of the product. In the FBS is found the mission statement and the environmental characteristics. From the environmental characteristics, sources of damage may be surmised and quantified. System definitions and dependencies can be found in the SBS. They provide the information needed for fault detection and isolation. The ABS provides the information needed to determine what must be removed in what order to repair, refurbish or otherwise maintain the product.

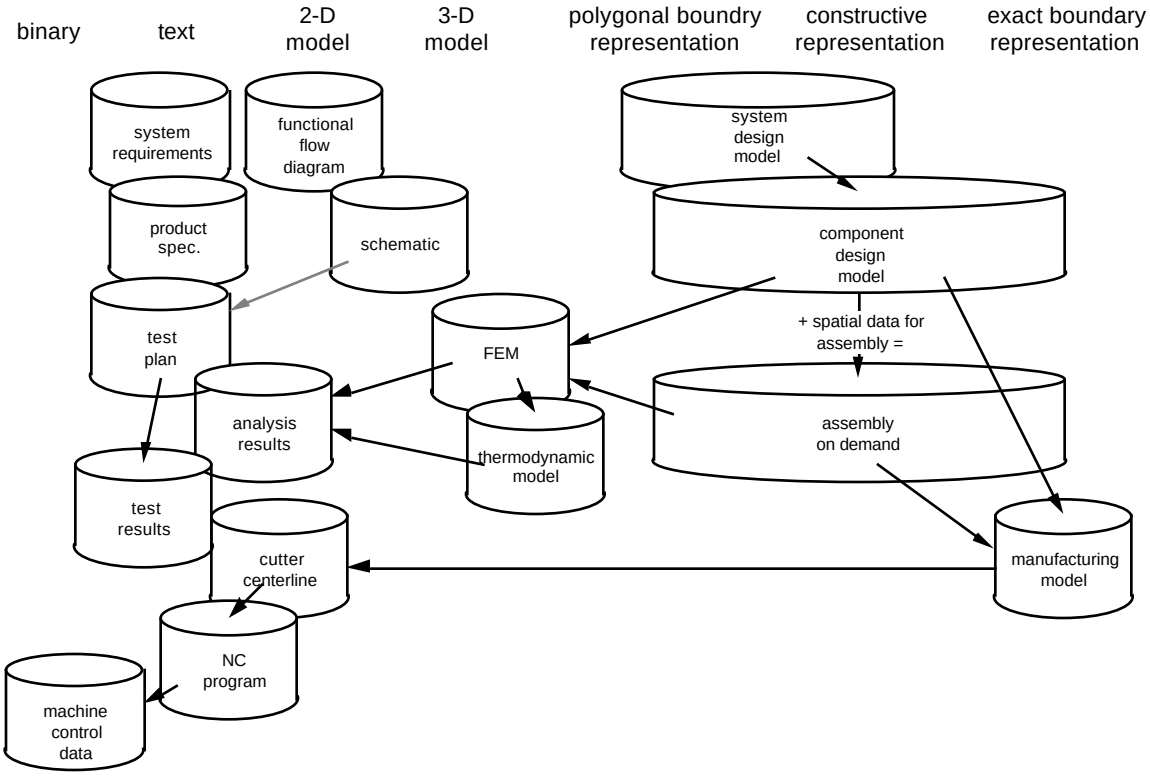
5.7. Data Amount and Fidelity

The amount of data grows through the life of a product. The fidelity of the product definition data increases throughout the engineering phase of a product. The development of the Breakdown Structures is neither parallel nor sequential. The level of definition detail of the ABS will lag that of the CBS, which will lag that of the SBS, which will lag that of the FBS.

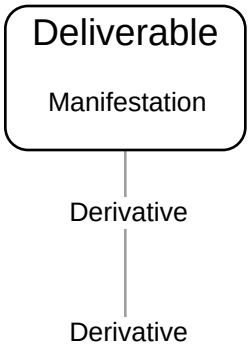


5.8. Derivatives

Various manual and computer-augmented means are used to create and disseminate the text, 2-D graphics, 2-D models and 3-D models used to define and communicate the product definition among people and computers throughout the process. Examples of the various models involved in the process are depicted as computer files in the following diagram. Some are related to one another by way of the Function, System and Assembly Breakdown Structures. Others are copies of models used for one purpose, which have been modified or used as a constraint for models derived for another purpose. Many deliverables may be *derived* from other deliverables.

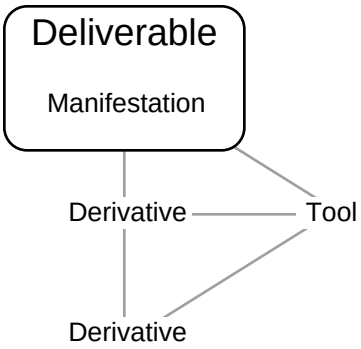


This chain of relationships from manifestations to derivatives of manifestations to derivatives of derivatives, must be maintained if the cost of a proposed change is to be accurately assessed. Only then will all items that may be affected by a change to a manifestation be known. Only then will all items that may be affected by a change to a requirement that affects a manifestation be known. Only then will what needs to be changed be known should a change be approved.



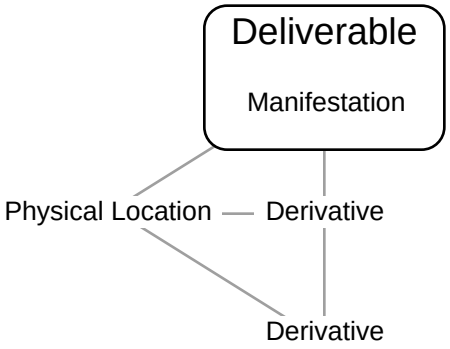
5.9. Tool Used

The knowledge of what tools were used to create or modify a manifestation or derivative must be maintained. Then in the event that it is belatedly discovered that a revision of a software tool was corrupting data, the corrupted data can be quickly isolated.



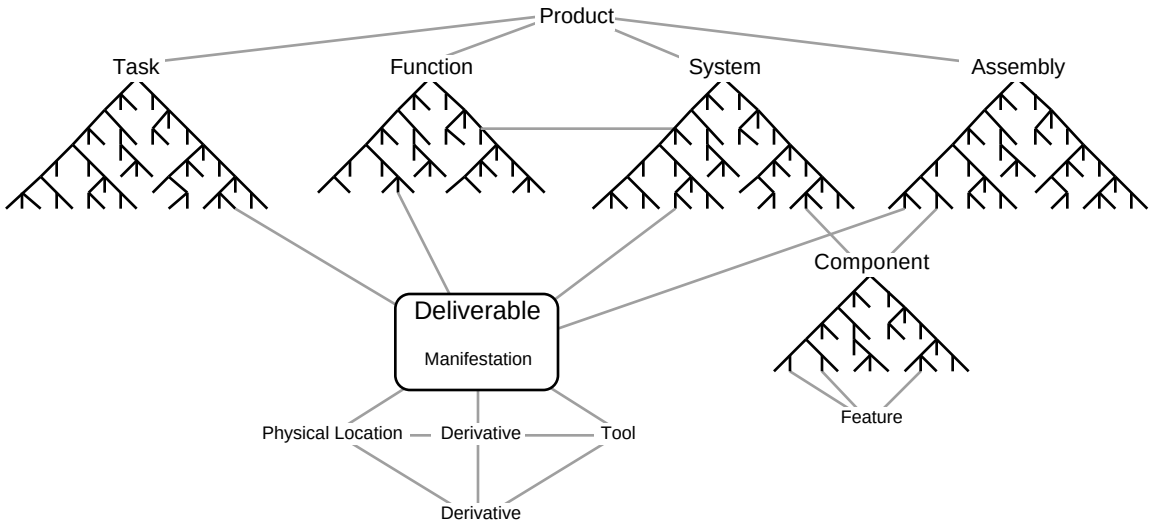
5.10. Physical Location

Whether to repair or discard bad deliverables, or find them for use in the process, the physical location of each deliverable should be known. For this purpose, a fourth link to each deliverable is required.



5.11. Data Navigation

Only five relationships must be maintained for each deliverable. One is to the lowest subtask in the TBS that defined and authorized the work to create the deliverable. A second is to the breakdown structure (FBS, SBS, CBS or ABS) to which the deliverable is most reasonably related. The third is to the deliverables that were derived from it. The fourth is to the tool that was used to create it. The fifth is to its physical location.



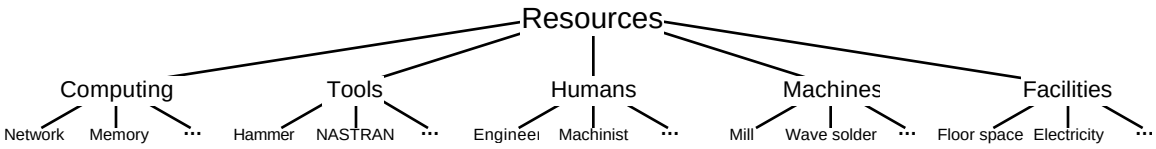
These relationships establish the simplest method known of navigating throughout all the Program and product data generated throughout the life of a product. A part number is not required to find information. Knowing any one bit of information will lead to any other bit of information.

5.12. Configuration Management

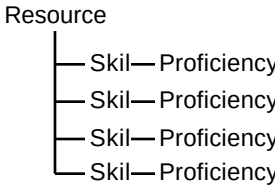
There will likely be versions of the deliverables, derivatives, the tools used to make them and the Breakdown Structures used to relate them. Some changes will be effective on different parts of a product. Other changes will be effective during different parts of its life. Consequently, the configuration management of the product must include all the data structures and tools used to define, manufacture and support the product.

6. Resources

The resources needed to conduct the process are a function of the product. They may be many and varied. Like the information hierarchies, there could be resource hierarchies, but they are of little value.



All resources should be characterized only by their skills, their proficiency in each skill and the cost per unit time of their use. Every resource has a set of skills and a proficiency in each skill.



6.1. Computing Resources

Computing resources include everything related to computers and their communications. Computers are are a particularly pervasive resource. Computers consume facility resources. The facility resource costs and computer maintenance costs are the recurring costs for a computing resource. Computers have memory, permanent storage, processing and network access skills. They have memory and permanent storage proficiencies in terms of its size (Mbytes) and speed (transfer rate). They have processor proficiencies in terms of size (word length) and speed (MHz). They have network proficiencies in terms of size (band-width) and speed (Mbits/sec).

6.2. Tool Resources

Tool resources are those which are used by other resources to produce a deliverable. Human resources use hand tools, like hammers. Machines use machine tools like a horizontal cutter or a holding fixture. Both may use software tools, but human resources dominate the use of interactive software tools. A hammer has hitting, peening and fastener removal skills. A hammer has a hitting proficiency in terms of moment arm (handle length), head weight and impact area. A NASTRAN computer program tool has static and dynamic finite element analysis skills. Its proficiency is described in terms of the size of the model it can analyze and the speed with which it can perform the analysis on a nominal computing resource.

6.3. Human Resources

Human resources are critical to the functioning of all the resources. They make, buy, install, invoke, use and maintain all of the other resources. Human resources have mechanical design, analysis documentation and tool use skills and proficiencies in each.

The information about who is doing what with what resource is cumulated up the management hierarchy. The higher managers have wider perspective, but less detail. The information they receive must be filtered and compressed. They do not have the time to understand it to the detail necessary to make an informed decision. Those on the bottom who are intimately aware of the details have a limited perspective. They usually do not have the authority to make decisions. When they do, their lack of scope may lead to bad decisions, because they are unaware of the business strategy.

After peeling away the layers of symptoms that obscure the real problems, at the core of every problem is a lack of personal freedom. In many instances, company policies are dictated by government regulations that required the aerospace industry to be costly and wasteful. Literally, an act of Congress is needed to make the simplest of process improvements. In most instances however, imaginary company policy or government regulations are used as excuses to avoid change. When this fallacy is challenged, a layer of organizational or people excuses is proffered to avoid change. People fight to stay in a rut about which they bitterly complain. It can take the demise of the company to induce real change.

6.4. Machine Resources

Machine resources are the mills, lathes, material handling, component placement and test equipment and the like. Manual machines consume human resources. Automated machines consume computer and human resources. Both consume facility resources. Milling machines have material removal skills. Their proficiency is described in terms of number of axes of motion, the speeds of those motions, maximum travel, bed size, cutting power and speed.

6.5. Facility Resources

All of the other resources are dependent on one or more facilities resources. Facilities resources have physical support (land), protection (enclosure), cooling, heating and electrical power skills and proficiencies in each (enclosure area, volume, cooling capacity, etc.).

6.6. Resource Correlation

Each task requires a set of skills and the proficiency necessary to accomplish the task (produce a deliverable) on time and at minimum cost. When there is no direct correlation between the skills and proficiency of a resource and the skills

and proficiency needed to produce a deliverable, a combination of resources are required to produce the deliverable. When this circumstance is recognized, the task must be decomposed into discrete subtasks for each resource as described in the Information Integration section. In that way, resources will be explicitly correlated with each other by way of the deliverable dependency among the subtasks.

This relationship provides the basis for determining the value of the correlation of resources by their physical proximity (relative location). Resources that directly transfer deliverables should be physically adjacent to minimize the cost of transfer. For digital deliverables, the transfer cost is low, regardless of the distance. For immobile resources, the information is only useful for determining the cost of the process. However, if the process cost is high enough and the duration of the subtask is long enough, even resources determined to be immobile should be moved.

6.7. Resource Evolution

All resources are acquired or developed, improved or at least maintained and retired or destroyed. Nearly all resources can

gain skills (machine operation or in-process inspection probe),
gain proficiency in old skills (advanced accounting class, new bearings),
lose skills (loss of memory skill due to stroke or disk drive failure) and
lose proficiency (poor response due to degraded eyesight, or backlash of warn gears).

Some skills become rare or otherwise increase in demand. Others are no longer useful to the process. Consequently, the amount that can competitively be charged for the time of a resource varies throughout its evolution.

6.8. Resource Value

To determine accurately the price of a product given a desired profit, or vice versa, its cost must be known. The cost of a product is that of its raw materials and all the resources employed to convert the raw materials into a viable product. The cost of a resource is its acquisition cost distributed over the expected useful life of the resource plus its recurring cost. The recurring cost of a resource is a combination of its maintenance (wages, oil), repair (medical, replacement parts) and improvement (training, enhancements) costs. The acquisition and installation costs of human resources are low compared to their recurring costs. The opposite is more nearly true of all other resources. There are two approaches to resource valuation.

6.8.1. Authoritarian Approach

The use of resources described in the Information Integration section is from the conventional perspective in which an enterprise effectively owns and directly controls all of its resources. Resources are *assigned* to perform tasks. In this case, the value of the resources is a function of their acquisition and recurring costs. Their value is not a function of their demand (annual merit increases do

not accurately reflect current value). This valuation method results in over- or under-utilization, unless a resource management tool like that described in the Tools Resources section is employed. The resource management tool must be continually updated with current resource availability, skills and proficiency if it is to be of value.

Without such a sophisticated Resource Manager, the true cost of resources are not accurately known. Hence, the true cost of the product and the profit are not accurately known. Seemingly inexpensive resources may in fact be very expensive when their less obvious costs, like medical expenses, early burn-out, maintenance personnel on overtime and poor product quality are considered.

6.8.2. Libertarian (Free Market) Approach

The resources of commercial suppliers are indirectly controlled by their customers by the price the customers are willing to pay for the products or deliverables that result from the use of those resources, or the price the suppliers are willing to bid to supply the deliverables. The value of the resources of a supplier change as a function of the price the supplier can demand for their use. As the price rises, other suppliers are attracted to the same market. The additional supply forces the original suppliers to reduce the price of their products in competition with the new suppliers, effectively reducing the value of the resources involved.

As the price declines, suppliers who find the price unprofitable leave the market, reducing supply. This establishes the potential for a price increase by the remaining suppliers should the demand for the deliverables remain or increase. Others invest in improved resources to maintain or increase their profits at the lower price.

The more successful enterprises try to commit their suppliers to low prices over the life of the deliverables. This approach can backfire if the resource costs of the supplier rise to the extent that the supplier can no longer afford to supply the products as contracted. A viable price/supply balance can only be achieved in a free-market environment where the prospect of late, poor quality or no deliverables is less likely.

A free-market approach to resource valuation and allocation can be applied to the resources of an enterprise if the principle of private property is employed. By selling, leasing or renting the non-human resources of the enterprise to its human resources, the human resources can competitively bid for work for themselves and their resources. Supply and demand will not only guarantee the minimum price for the deliverables, but also cause the true cost of the resources to be accurately reflected in the price of the deliverables. If the bids for some work are high compared to external commercial alternatives, then additional or more productive resources can be acquired to increase the internal supply. Otherwise the deliverables should be bought from the external supplier.

This approach eliminates the need for a sophisticated Resource Management tool and the cost of its maintenance. All that is really needed is a way to communicate market (Program) needs to the human resources. The Task Breakdown Structure (described in the Information Integration section) is adequate for that task.

6.9. Computing Resources

The computing resource technology changes so rapidly that new products appear daily. It is futile to try to provide a specific description of computing resources. This qualitative description should suffice to guide those responsible for the acquisition, installation and maintenance of computing and related resources.

6.9.1. Requirements

The business process must quickly adapt to new products. Consequently, the computing resources of an enterprise must quickly adapt to the needs of its business process. This adaptation should be in the smallest units practical to minimize the possibility of excess computing capacity, and therefore excess cost, or insufficient computing capacity, and therefore inadequate support of the process.

Computing resource changes should minimally impact the users of those resources, and therefore minimize training costs. To further reduce training costs and maximize productivity, the use of the computing resources should be intuitive or automatic. They should be optimized for their users. The responsiveness of the computing resources should be consistent such that they do not disturb the thought process or timing of its human or non-human users, respectively.

Computing resource changes should also minimally impact the facility resources, and therefore minimize computing resource relocation costs. Since most facility resources must accommodate human resources, the computing resources should be compatible with human resources and vice versa. No special environmental control (temperature, humidity, sound), power supply, structural support or structural rigidity or damping should be required to support the computing resources. Doorways and elevators, as well as cooling requirements, limit the size of the computing resources.

The value and volatility of the data and tools (application programs) residing on the computing resources is such that it is advisable to have at least one copy stored on a geographically remote computing resource.

The deliverables developed on a computing resource by one function in the process must be shared with other functions by way of their computing resources. If process schedule or timing requirements dictate that computing resources are to process subtasks cooperatively (distributed parallel processing), they must also share control information. The sharing of control information is also required if the activities of tool and human resources are to be interleaved with those of computing resources. This necessitates a means of communicating deliverables and control information among computing resources. The Task Breakdown Structure described in the Information section can contain all the control information and the metadata about the deliverables

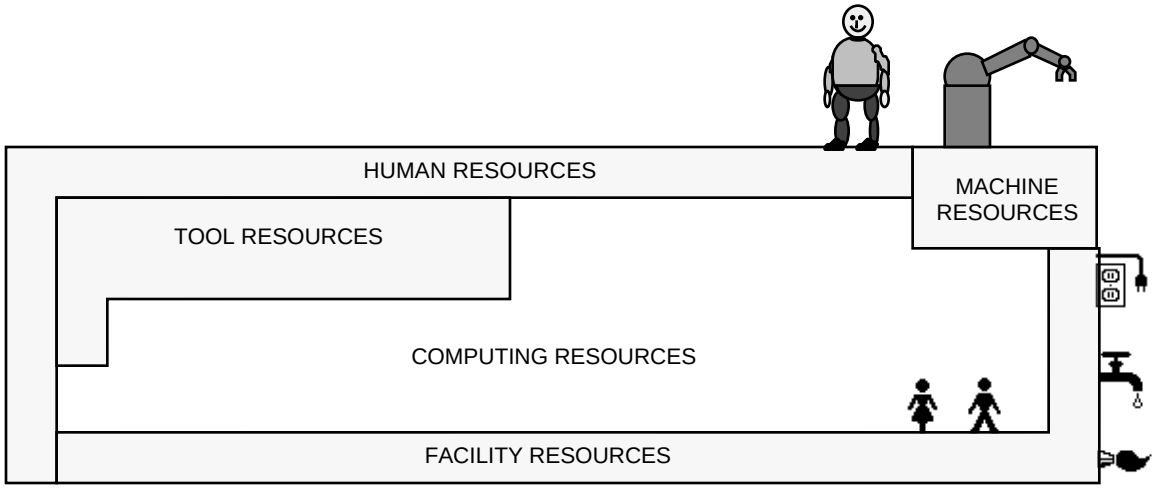
necessary to support such cooperative processing. The deliverables can be found by way of the other Breakdown Structures as well.

Conversely, computing resources must be capable of performing a minimum set of skills and proficiencies because they may be isolated from the other computing resources. Such isolation may be done accidentally due to a resource failure, or intentionally should the nature of a deliverable or software tool be such that it cannot be made accessible to other resources (classified).

6.9.2. Architecture

The requirements may be met with a collection of networked computers. Neither the computers nor the network components will likely be made by the same manufacturer. It will be a heterogeneous environment of computing resources.

The relationship of computing resources to the other resource types is represented in the following diagram.



This diagram uses adjacency to show how the computing resources interact.

6.9.3. Communications

Until brain-to-brain communication is viable, some combination of sight, sound, touch, motion, position and orientation must be exchanged among human resources for effective communication to occur. Sight includes images (photograph), simulated images (shaded solid or surface models with shadows, reflections and other effects), wire frame models, graphics or text. The sights may be animated (video or computer simulation). Sound includes voice, music and noise (cutter chatter). Sound cannot exist without a time domain. It is always 'animated'. Touch includes pressure, temperature, humidity and vibration cues. Motion is the feeling of velocity (hair movement, temperature differential due to uneven evaporation of sweat) and acceleration or

deceleration (inner ear and joint movement). Position and orientation are derived from a combination of sight, sound, touch and motion cues.

If the human resources are remote from one another, their communication must be converted to a medium that can be transmitted over their intervening distance. The communication may be converted to analog signals and transmitted by way of copper wires, optical cable, the atmosphere or satellites. To avoid distortions due to interference, the analog signals may carry digital information. Digital data may be checked and corrected to insure that what was sent was received.

With the right peripherals, it is currently possible to communicate sight, sound, touch, motion, position and orientation. Sight can be communicated using various computer, video and holographic displays. Sound can be communicated using audio speakers. Touch can be communicated using actuators, blowers and radiant or piezoelectric heating and cooling devices. Motion can be communicated using visual cues, but true motion can be better communicated using a six degree of freedom flight simulator. Position and orientation can be communicated verbally if an experience has been shared (northeast corner of Hollywood and Vine looking south).

If computers are in the communication path, the information must be in digital form. Analog signals like voice or video must be digitized into discrete values over a time domain.

The bandwidth of the communication medium may not allow the communication to occur in "real-time," so the information must be stored, reconstructed and presented later than its actual occurrence. The information may be compressed and decompressed to get more information through a limited band width communication medium. Even with data compression and decompression, real-time communications may not be possible or practical.

Not all forms of communication are practical or necessary for the design and manufacture of a product. In the interest of cost, one must always ask, "What is the minimum communication technology needed to communicate the necessary information unambiguously within a time that will not distort the information or make it too costly?"

Processing consistency is important to human and machine users of a computing resource. If a distributed heterogeneous computing environment is involved, digital data must be accessible by physically remote computers. The software tools used to process the data must be similarly accessible. The execution of the tools on the data should proceed as if they were local resources. Consequently, the network should have sufficient capacity (bandwidth) to function as though the computers were directly (channel) connected.

6.9.3.1. Interaction Methods

There are various ways computer resources can communicate: master/slave, client/server and peer-to-peer. Each is discussed briefly here.

6.9.3.1.1. Master/Slave

This is the classic multi-user environment in which dumb terminals rely entirely on the services of a "mainframe" or "host" computer. As its name implies, it isn't much fun for the human resources, who must compete with their peers for host cycles. When the host "goes down," so does everyone.

Obstructive access and data security mechanisms must be employed to protect the investments of users from other users. A utilization charge rate based on CPU, memory, disk access/storage and printer output is required to minimize over-utilization as well as pay for the resource. Hence, an account number is required. Unfortunately, it can take days or even weeks to get the required charge number, approvals and access privileges.

As personal computers became available, so did terminal emulation software. An informal client/server environment ensued in which the host computer became an uncooperative server.

6.9.3.1.2. Client/Server

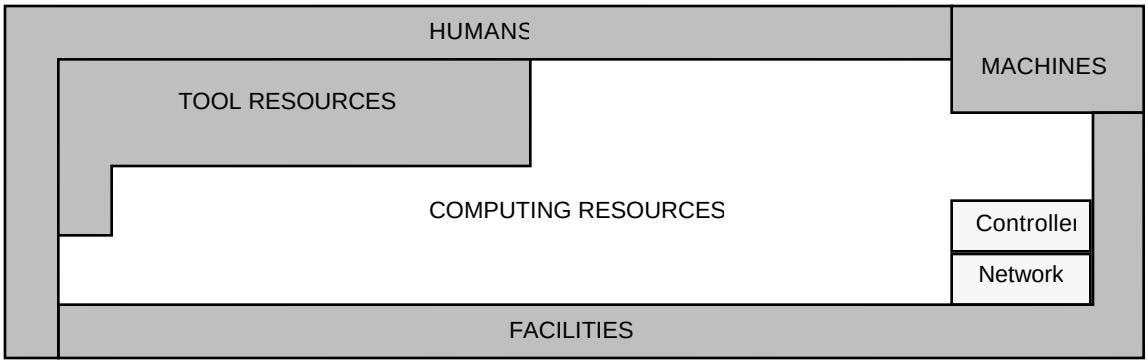
This is currently the most widely supported model. Much of the technology discussed in the remainder of this section supports it. At its most rudimentary, a client may merely transact with a server on behalf of its human resource. More sophisticated implementations have the client and server continually interacting and load sharing.

6.9.3.1.3. Peer-To-Peer

Clients and servers readily swap roles according to the circumstance.

6.9.4. Network

The network is the backbone of the computing architecture. To minimize incompatibilities among networks and computers and the cost to accommodate them, network standards should be adhered to whenever possible.



From the perspective of a computer, the network is just another input/output device. Beyond the computer is a labyrinth of wires and communication equipment that are unappreciated by the computer.

6.9.4.1. Standards

Various communication standards are widely accepted by industry. They include the Open Systems Interconnect (OSI) model of the International Standards Organization (ISO), the Systems Network Architecture (SNA) of IBM, the Digital Network Architecture (DNA) of DEC and the Manufacturing Automation Protocol/Technical Office Protocol (MAP/TOP). ISNA, DNA and MAP/TOP all parallel the ISO OSI model to some degree. The protocols are tested by the Corporation for Open Systems. A common protocol must minimize the cost of creating, expanding or linking communication networks. It will likely be selected from the options specified by the Open System Interconnect (OSI) body.

OSI specifies a comprehensive framework of communications standards covering all aspects of information exchange among computer systems and their respective applications. It establishes protocol standards in seven levels, or *layers*, of data communications and data management functionality. A sub-set of these may be selected for actual implementation..

Each layer contains modules that specify and define a separate aspect of the communication function. Within each module are protocols that define message formats and the rules for message exchange and network management between communicating systems.

The seven OSI layers are the *Physical*, *Datalink*, *Network*, *Transport*, *Session*, *Presentation* and *Application* layers.

6.9.4.1.1. Physical

The Physical Layer is concerned with the mechanical and electrical transmissions of signals among computer systems. The standards specify connectors, modulation and encoding techniques. The IEEE 802 standards for Local Area Networking support the physical and link layers of the OSI model. The selection of a proper subset of the IEEE 802 standards for implementation depend entirely upon the application.

The physical network medium may be an electrical conductor like copper, an optical conductor like glass fibers, microwaves or some other means of communicating electrical signals. The medium should minimally impact the facility resources, yet support the speed and bandwidth of the connected computing resources.

The facility resources required to support voice communications among human resources are often based on networks of copper conductors (unshielded small gage twisted pair). This often drives the selection of a network conductor, but telephone communications may also be performed on computer network media (shielded or fiberoptic cable). The selection of a suitable network medium is a function of the distance between using resources, the communication flow rate required, the physical environment and the electromagnetic environment.

Hence, the network medium is likely to be heterogeneous. The following options are listed in order of the cost of the medium. Microwave transceivers will likely be used over long distances or under circumstances that make a physical link impractical (rapid movement of workstations in a building that is not pre-wired). Fiberoptic cables will likely be used where a high data flow rate is required or where high levels of electromagnetic radiation exist. Shielded copper cable will likely be used where a high data flow rate is required or where moderate levels of electromagnetic radiation exist. Unshielded copper wires will likely be used where they already exist, or where low data flow rate is required and low levels of electromagnetic radiation exist.

Consideration should be given to the fact the network medium may also be used to communicate voice, video, and security, safety and environmental information.

6.9.4.1.2. Datalink

The Datalink Layer establishes an error-free communications path between network nodes over the physical medium. It manages access to the communications channel and ensures the proper sequencing of transmitted data. A node is any junction in the network. Each node has a unique address and a computer. The computer may be of any type (workstation, file server, communication specialist), as long as it supports network activity. Data Links can be Ethernet, DDCMP or HDLC.

6.9.4.1.3. Network

The Network Layer supports the allocation and interpretation of node addresses. The network layer establishes the path between communicating nodes. It routes messages through intervening nodes to their destination and controls the flow of messages between nodes.

6.9.4.1.4. Transport

The Transport Layer provides source node to destination node control of a communication session once the path has been established. This layer allows processes to exchange data reliably and sequentially, independent of which computers are communicating or their location in the network.

6.9.4.1.5. Session

The Session Layer manages the dialog. It establishes and controls system dependent aspects of communications sessions between specific nodes in the network.

6.9.4.1.6. Presentation

The Presentation Layer masks the differences of varying data formats between computers of different vendors. This layer transfers data in a manner that is independent of the computer, performing appropriate conversions at each computer.

6.9.4.1.7. Application

The Application Layer provides services that directly support such user and application tasks as file transfer, remote file access and data management.

6.9.4.2. Hierarchy

A network hierarchy is a means of minimizing telecommunication costs. All the computers in a group of computers can communicate to other computers via a specially equipped computer in each group. Such a network architecture corresponds with the "keep the data as close as possible to its primary user" philosophy.

Some eighty percent of the work in an enterprise is concentrated in local processing by work groups using specialized resources. Their deliverable and control communication requirements are such that their computing resources may be grouped. The remaining twenty percent of deliverable and control data is shared among multiple groups of resources (functional departments). This relationship may change as multi-discipline teams are employed. The network should enable distributed processes and data sharing among the resources employed throughout the process.

Consequently, a combination of wide area, value-added, regional, local area and cell network types may be required to support the process of an enterprise.

6.9.4.2.1. Wide Area

A Wide Area Network (WAN) is a network which interconnects defined regional networks. It provides data and voice communications between distant sites which are not within the same defined region. This is supported by using satellite links and T-1 leased lines. Other public networks provided by companies like Telenet, Tymnet, and Accunet are also available to support WANs using X.25 packet-switching.

6.9.4.2.2. Value Added

WANs may be extended or interconnected by way of a Value Added Network (VAN). Subscribing to a public VAN extends network capabilities to other involved parties of the business process. Communicating to subcontractors and customers are strong examples of the business requirement to subscribe to a VAN. The government CALS initiative and standards on Electronic Document Interchange (EDI) provide justification and enabling capabilities to utilize a public VAN to support effective communications in an extended enterprise environment.

6.9.4.2.3. Regional

The regional network interconnects host processors, workstations, and terminal devices at all levels of the hierarchical network between remote sites within a geographic area. Regional networks normally consist of 56 Kbyte Digital Data Service (DDS) and T-1 leased lines from a local telephone carrier. When appropriate, multiplexed wideband links are used for ease of network management and cost benefit reasons.

6.9.4.2.4. Local Area Network

The Local Area Network (LAN) is usually based on a *backbone* that is capable of high speed data communications with transfer rates exceeding millions of bits per second (Mbps) Local Area Networks are used to link cells and major computer nodes, using bridges, gateways or routers as appropriate.

A backbone identifies the highest level of networking within a geographic area. The backbone may be implemented over existing broadband resources. It may use fiberoptics where additional bandwidth is required. The backbone offers flexibility and connectivity to enhance the efficiency of a distributed computing environment.

Local Area Networks optimize data sharing and computer utilization in work groups. Appropriate clustering supports fast, high-volume traffic in a localized area, and a lower volume of inter-network traffic outside the local area. Local area networks also provide optimized sharing of peripherals. LANs are

particularly well-suited for departmental or project related functions which require a dedicated network (classified processing).

The physical media for LANs is usually building distribution twisted-pair wiring, except where bandwidth requirements mandate coaxial cable. Fiber may be used for bandwidth or interference reasons.

LAN configurations should adhere to IEEE 802.3 Ethernet (or where required, IEEE 802.5 Token Ring) specifications. Digital Equipment Corporation's DECnet may be implemented for new LAN configurations based on IEEE 802.3 Ethernet.

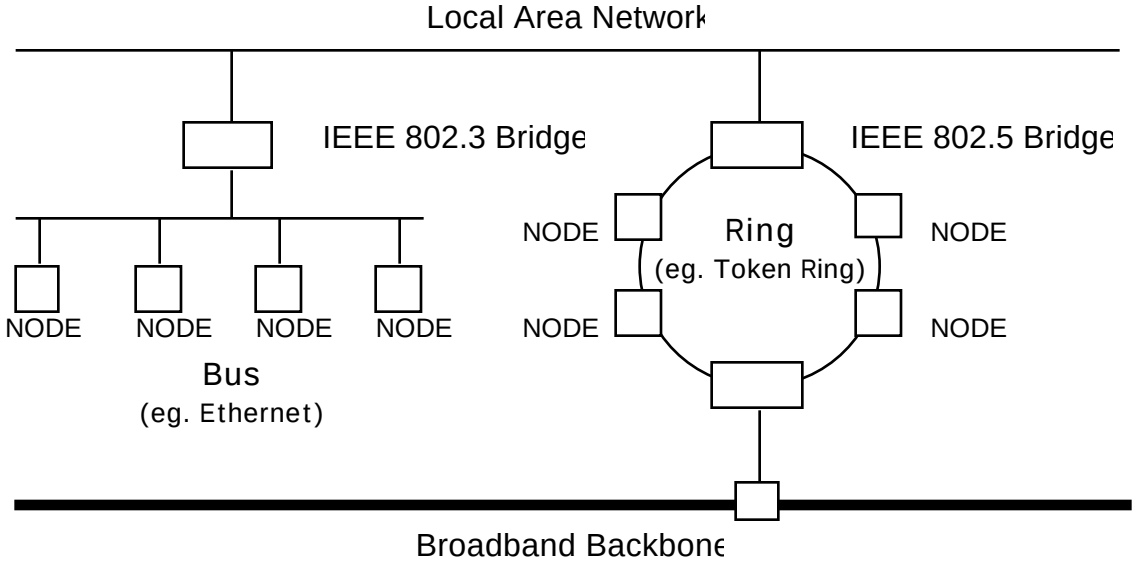
TCP/IP should not be implemented unless an OSI compliant upper-layer protocol product is unavailable.

LocalTalk networks may utilize existing twisted-pair wiring (PhoneNet implementation). LocalTalk LAN workstations may access external resources via a gateway node configured with IEEE 802.3 Ethernet (and if required, TCP/IP capabilities).

Apollo LAN workstations may access external resources via a gateway node configured with IEEE 802.3 Ethernet or TCP/IP. Until Apollo cell networks can utilize existing building distribution twisted-pair wiring, the LAN cabling should only span a limited area.

6.9.4.2.4.1. Bridges, Gateways and Routers

Bridges and gateways provide an extension of this hierarchical topology to lower level networks such as IBM's SNA and Apple's LocalTalk. Digital Equipment Corporation's Digital Network Architecture (DNA) spans both the backbone and local work groups (LANs) in a common environment.



Filtering bridges may be used to interconnect a LAN to a broadband network backbone.

Bridges are protocol-independent and transparent in the sense that cell networks with dissimilar upper-layer protocols can pass packets of information from cell to cell via the LAN. They have the flexibility to restrict local traffic on a particular cell so it does not flood the data highway with unnecessary data traffic, yet will automatically allow the flow of data to specific remote cells when absolutely necessary. Bridges improve overall network reliability and extend the distance over which communications between cells can be supported.

Gateways perform protocol and address translation to enable two dissimilar domains to communicate.

The use of bridges, gateways and routers as interfaces to a broadband backbone network retain LAN identification and provide isolation of networking levels. These bridges, gateways and routers should support DECNET, SNA, TCP/IP, and XNS higher-level protocols. Applicable LANs are IEEE 802.3 Ethernet, IEEE 802.5 Token Ring, Apollo Ring and LocalTalk.

Routers perform the same task as bridges and gateways, but require that the protocols above the OSI Network Layer be identical.

Data switches allow for flexible connectivity for terminal and host environments. A single terminal device can have switched access to several hosts, or several terminals can have switched access to a single host. Dedicated connections can be established as required. The two data switch standards are the AT&T System 85/75 and the Infotron INX4400. AT&T is the emerging standard, which integrates voice and data over the same wires. Infotron is the *de facto* standard.

A user requiring both telephone and asynchronous data transmission may have a telephone installed with an Asynchronous Data Unit (ADU). The ADU gives the telephone data communication as well as voice capabilities over a single set of wires to the PBX switch. Transition to the PBX as the standard for asynchronous communications is advisable.

The Integrated Systems Digital Network (ISDN) standards will provide solutions to data, voice and video (multimedia) communications.

6.9.4.2.4.2. Broadband, Baseband and Fiberoptic

Broadband Local Area Networks (LANs) are capable of supporting multiple communication services using basic cable television (CATV) technology (shielded cable). CATV technology utilizes frequency division multiplexing. It provides a wide range of services over a single coaxial cable, including video, voice and data. However, transmission rates over the broadband are limited to 5 Mbps or 10 Mbps per communication service, depending on the number of subchannels assigned for carrying the service as well as the communications equipment used.

Baseband coaxial cable backbones are quite different from broadband CATV technology for the support of backbone communication services. Baseband backbones restrict the maximum length of backbones and are only capable of supporting a single communications service. However, this restriction is overcome by its ability to transfer information at a minimum 10 Mbps.

Hyperchannel, a product of Network Systems Corporation may be used as a backbone. It supports high speed (200M bps over four coaxial trunks, 50M bps per baseband coaxial cable) bulk host-to-host file transfers between an enterprise computer (IBM 308X) and various mini- and micro-computers (workstations) from different vendors (heterogeneous computing environment) using proprietary communications software. If large blocks of data are to be moved on a regular basis, a high speed baseband backbone is advisable Hyperchannel may be a viable backbone network product until it is converted to or supplanted by an OSI compliant product.

Communication services have long been provided by electrons in copper cables. Now photons are being used in fiberoptic cables. Fiberoptic cable consists of a very thin glass core surrounded by reflective and supporting layers of material. It transmits the light generated by a diode laser. One fiber is adequate for asynchronous communications under 1000 meters. Two fibers are necessary for synchronous communications or for bi-directional asynchronous communications over 1000 meters.

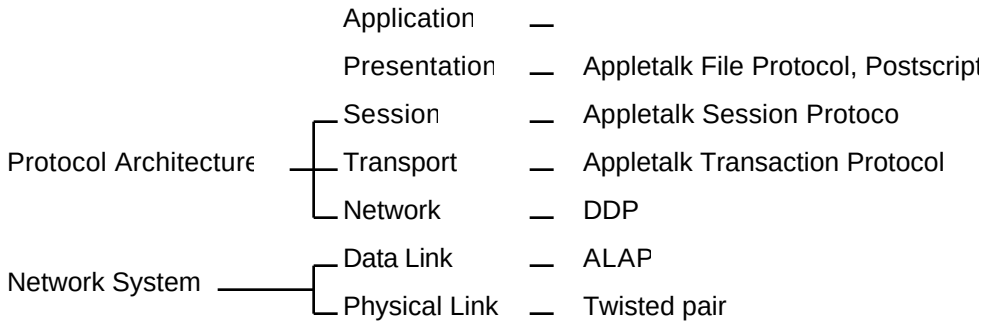
The use of fiberoptics for a communications backbone for a Local Area Network is ideal for many reasons. Present technology provides data transmission speeds in the order of hundreds of megabits and soon to be gigabits per second. This is possible because there is less signal attenuation than radio frequency signals. The use of fiberoptics technology is also more secure. It does not radiate signals in any form, and is immune to electromagnetic interference. It provides high integrity of data transmissions at very high speeds. Security breaches are also easily detected when the fiber has been illegally tapped since this would dramatically decrease the signal intensity and increase the error rate.

Unfortunately, communication interface units and field installation tools for fiberoptic communication systems are not well developed. It is difficult to design and implement a viable fiberoptic backbone. Standards are being developed for the use of fiberoptic technology in communications. It is only a matter of time before the industry can take practical advantage of the backbone capacity of fiberoptic cable. The American National Standards Institute (ANSI) is in the process of passing a Fiber Distributed Data Interface (FDDI) standard. It will establish a full 125 Mbps backbone using redundant Token Ring technology.

6.9.4.2.4.3. AppleTalk LAN

AppleTalk is a network system consisting of cabling systems, network components and network services. LocalTalk (.25 Mbps), Ethernet, Token Ring,

FDDI, etc are cabling systems. Routers (bridges), gateways, Macintosh computers, personal computers (IBM and clones) and DEC VAX computers are network components. AppleShare (Macintosh file server), LaserShare (Macintosh print spooler), etc. are network services.



6.9.4.2.4.3.1. Applicability

AppleTalk may be used on all Local Area Networks at 230.4 Kbps, using the LocalTalk (telephone cable), or 10 Mbps, using EtherTalk (coax).
The newer AppleTalk Data Stream Protocol (ADSP) consists of:

Network	16 bit static (managed),
Node number	8 bit dynamic (computer finds unique node number),
Socket	8 bit,
Zone	logical/arbitrary way to group nodes,
Bridges	Internet routers.

6.9.4.2.4.3.2. Functionality

Terminal emulation
File transfer
    One VMS file = one Macintosh file or
    One VMS file = one Macintosh disc
Virtual disk
File serving
Print serving and
Mail serving

6.9.4.2.4.3.3. Hardware Options

Asynchronous - terminal emulation, file transfer, virtual disk, performance limited.
Direct Ethernet - 2 to 5 times faster than asynchronous, all services, use Kinetics Etherport SE, SC, or Etherport II, the Dove Box or Apple EtherTalk.
LocalTalk bridge to Ethernet - all services, performance good, use Kinetics Fastpath (fastest), which uses AppleTalk (dynamic), IP (static), or Cayman Gatorbox, which uses Columbia AppleTalk packages (AppleTalk inside of Internet Protocol).

AppleTalk for VMS is a standard. Its ridge performance problems will go away with AppleTalk 3.0. AppleTalk File Protocol (AFP) will be standard for the Macintosh/VAX. MACworkstation, X Windows and DECwindows will be supported. ADSP will be used for distributed data processing.

3/1. from Network Innovations, a subsidiary of Apple Computer will be the standard communication language. It will provide distributed data access for desk-top applications through networks to host computers (IBM VM or DEC VMS) and their databases. It has 35 SQL-based data manipulation verbs (statements). They allow the application programmer to ignore the network and concentrate on application to application programming (client/server). CL/1 is written in C. It is tightly integrated with the DEC Command Language (DCL). It will produce output data mapping as well as the data needed by a client application. CL/1 is also linked with HyperCard.

6.9.4.3. Cell

Data transfers need not occur in real-time except those occurring between process controllers driving relatively autonomous devices (direct numerical control machines, robotic devices) and the controlling (cell) computer.

Cell network configurations provide a specific network service. They are favored for departmental use. Examples of cells or subnets include segmented Ethernets (thin, standard, twisted-pair), Token Ring, Apollo ring and AppleTalk.

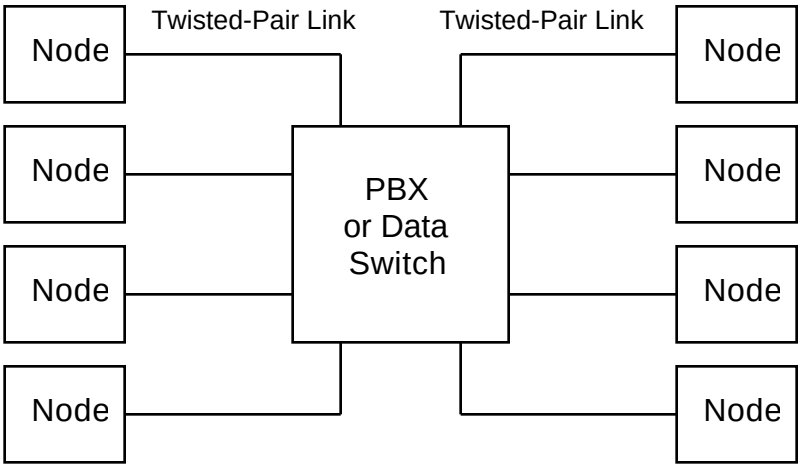
Minicomputers, workstations or even personal computers may be used as the interface between a LAN and a cell. They may be a cell controller. A cell controller controls the devices in the cell and informs any human operators involved with the cell of its activity. The cell controller cumulates tasks and task sets (repeat this task n times) as they are received. It distributes information to the operators and the various machine and robot controllers such that their end-effector actions and motions are coordinated. Depending on the type of cell is network implemented, a bridge or gateway may be used as an interface to the LAN.

The cell controller should have sufficient disk capacity for 24 hours of work. This should provide ample time for a failure external to the cell to be corrected before cell operations are affected. The cell controller should also have a removable disk or other means to acquire data for the cell should an extended network failure occur.

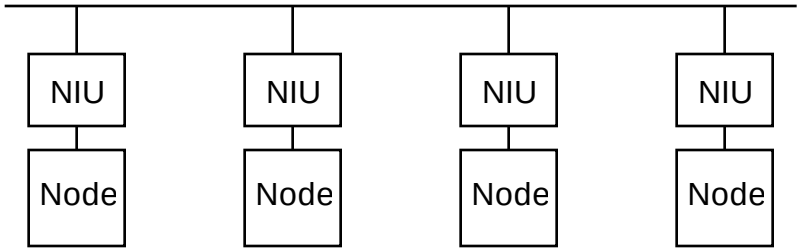
6.9.4.4. Node

Nodes are terminal devices which include file servers, personal computers, workstations, and intelligent peripherals (printers, plotters). They may be individual devices or interfaces to a cell network. ASCII terminals are connected to the network environment through a routing device such as a digital PBX, data switch, or intelligent LAN unit (NIU).

LOGICAL NETWORK STAR CONFIGURATION



BUS NETWORK CONFIGURATION



6.9.4.5. Network Management

Network management increases in importance as networks and the applications running on them become more distributed in a heterogeneous computing environment. Network management covers the administration, configuration, performance, fault isolation, and security management of the network. OSI standards will enable the development of common management information services. These services will allow inter-operability among network management applications across a heterogeneous computing environment.

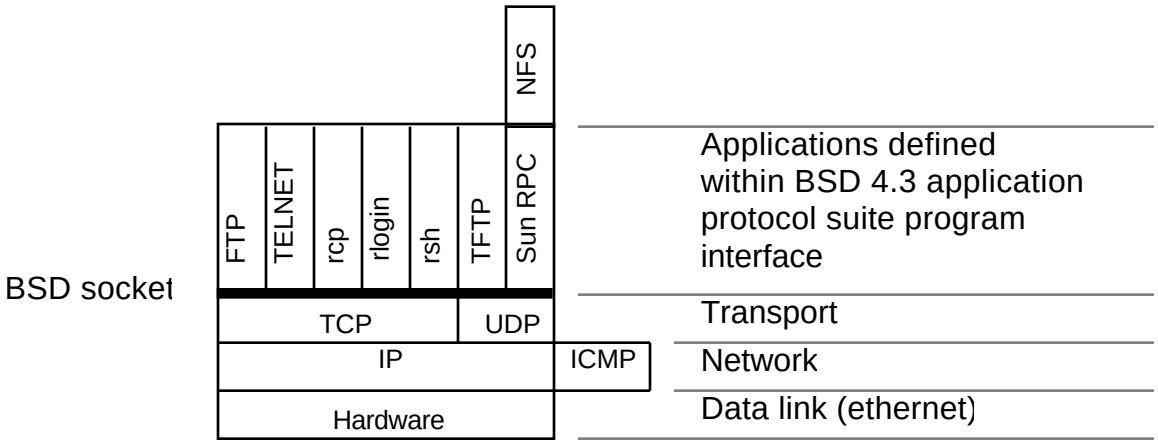
System configuration and traffic will be monitored from a centralized facility having the technical capability to administer the network. Departmental or project LANs may assume some responsibilities for network administration as an extension of the centralized network management facility. Until the OSI standards have been completed, vender provided network management

"turnkey" applications will be used. IBM's Netview is capable of monitoring network components to the node level. DEC's VAX Cluster NCP can determine "reachable" resources within a particular VAX cluster.

The network management tool should support non-stop networking by performing tests, detecting faults, disabling failed elements, re-routing traffic, indicating state, gathering status, providing displays and report generators for network administrators and notifying users of network and node status. The network management of the future should provide enterprise wide management of the distributed computing environment, and be accessible from anywhere in the network.

6.9.4.6. Unix Communication Services

The current implementation of these protocol standards in the Unix environment is in part by way of TCP/IP. BSD 4.3 has the better networking subsystem. Ultrix 3.0 is essentially BSD 4.3 plus corrections, new functions and better performance. Not included with Ultrix 3.0 are syslog, xns, tn3270, TCP/IP used over uucp or hyper, but it does include slip, news, rn and gated (public domain software).



6.9.4.6.1. Protocols

Transmission Control Protocol (TCP) is a stream protocol (no records) in which a sequenced packet of bytes is transmitted using one or more network paths. It will re-transmit if bytes are missing according to the checksum at the receiving station. It must consume data as it arrives and re-combine bytes into records. It uses port-to-port connections, flow control (windows) and acknowledgements.

Internet Protocol (IP) supports 32-bit host addresses with 4 bytes in three forms:

Class A - $N_0H_1H_2H_3$ - $N<128$

Class B - $N_1N_2H_1H_2$ - $128 \leq N < 192$ for large sites

Class C - $N_1N_2N_3H_1$ - $192 \leq N < 255$ for most small sites

It also supports adaptive routing and fragmentation/re-assembly of 1500-byte Ethernet packets that target machines utilize whenever communication is available.

User Datagram Protocol (UDP) is a datagram protocol that is unreliable (no promises, can run out of buffer space and discard data). It preserves records, but the consuming program must check for lost data and byte order (packets delivered in the order they arrive). It has an optional data checksum.

Internet Control Message Protocol (ICMP) performs routing management ("Look out, this node is going down") link management ("Receiving host buffers are full."). It throttles traffic, provides error messages (destination unreachable, time expired), and makes low-level calls (ping).

ARP is the acronym for Address Resolution Protocol. It maps between INET an address and its Ethernet address.

6.9.4.6.2. Services

File Transfer Protocol (FTP) has high start-up overhead and low transfer overhead. It is built on TCP and has a lot of functionality (e.g., multiple files transferred in one operation).

Trivial FTP (TFTP) has low start-up overhead and moderate transfer rates.

Simple Mail Transfer Protocol (SMTP) supports electronic mail related commands (Send mail).

TELephone NETwork (TELNET) supports remote logon as if the terminal were a local device, using RS-2323 for long distance communications.

6.9.4.6.2.1. services on Unix:
rep - remote copy - mimics cp, allows 3rd party copy,
rlogin - remote login - includes terminal type and
rsh - remote execution
require commonality of name space.

Remote Procedure Call (RPC) of Sun Computers allows low-level routines to be executed remotely.

Network File System (NFS) allows:
remote file systems to be "mounted" to local system,
remote printing (lpr),
remote execution (rsh, rexec, SUNRPC),
name saves (bind, yellow pages),
terminal servers (Telnet, Local Area Transport protocol),
network oriented window system (x:, MIT's Project Athena is socket application).

6.9.4.6.3. Data Links

Data Links can be Ethernet, DDCMP or HDLC.

6.9.4.6.4. Programming

Subnets shield internal company organization from the outside world. They allow an internal collection of networks to appear as one to the outside world. They use a Class B address ($N_1N_2H_1H_2$) like a Class C address ($N_1N_2N_3H_1$), where N_1N_2 are the Class B network for the outside world, H_1 is for a gateway that routes the external and internal world to Internets, and H_2 is the address for a host on the internal network.

Socket addressing requires 16 bits to be unique to host and protocol. Socket numbers < 1024 are associated with some privileged application that will not necessarily respond. A socket can only be opened by root (most powerful Unix user). The socket is the terminator for communications at a computer. Sockets are addressable software data structure established by the Unix kernel. Pipes can be sockets. In fact some pipes are implemented as two sockets.

6.9.4.6.5. Callable System Routines

socket() - set of primitives that create, bind (application has its own network address) and connect socket (active user). It can listen, accept (passive user), read, write and close. It obtains socket from system (# = port number). It returns a descriptor with three arguments:
6.9.4.6.5.1. protocol family (AppleTalk ...),
6.9.4.6.5.2. type (stream or data handler) and
6.9.4.6.5.3. which protocol to select from family according to type.
bind() - associates an address with a socket (socket address data structure setup). It cannot use a busy socket. If a socket number is given as 0, the system will select a socket number.
connect() - client initiation of stream protocol.
accept() - server initiation of stream protocol.
sent() or sentto() - send data on socket. Can only be used in the case of a connected socket.
sentto() - send data on socket. Can only be used with an unconnected socket.
recv()/recvfrom() - receive data from socket.

6.9.4.6.6. Network Management

RIP - Routed Information Protocol - dynamic local routing
Exterior Gateway Protocol
Setup - Kernel configuration

6.9.4.6.7. Naming

Berkeley Internet Name Domain (BIND).

6.9.5. Computers

To be compatible with enterprises that experience dynamic organizational changes, computing resources must be readily expandable and contractible. The large, mainframe-based computers that traditionally provided computing resources interactively by way of terminals or batched by way of paper reports have performance and cost increments (step functions) that are too large for a dynamic enterprise. Programs often cannot afford computing resources during their start-up phase, because they must pay for excess computing capacity. If they grow rapidly, they suffer with too little computing capacity. It takes too much time to justify the large cost involved in increasing the capacity of mainframe computers. Furthermore, the peripheral devices that are available to mainframe users are limited.

A network of workstations and file and application servers allows computing capacity to be increased or decreased in small increments of cost and performance. Programs can readily change the computing cost and performance according to budget and schedule constraints. High performance computers can be moved or otherwise applied where the greatest benefit is derived. The computing resources are under the direct control of the Program. They can be isolated for security reasons. However, such isolation may increase the cost of computing due to the need to duplicate computing resources and data in a secure environment that might otherwise be available via the network.

To minimize installation and relocation costs, the computers must be compatible with an office environment. They should require no extra air conditioning or electrical power. The heat load they add to the environment should be no more than that generated by humans occupying the same space.

Computers are normally microprocessor-based. They normally have a bus into which processor, memory and peripheral control cards (printed circuit assemblies) are inserted. The peripheral control cards may interface any combination of many different peripheral devices to the computer. They may be output devices like

- interlaced (TV, VCR),
- non-interlaced two- and three-dimensional image displays,
- pen impact dot-matrix electrostatic ink jet or laser plotters or printers,
- sound generators,
- analog or digital signal generators,
- stepper or synchronous motor drivers for machine or robot control.

They may be input devices like

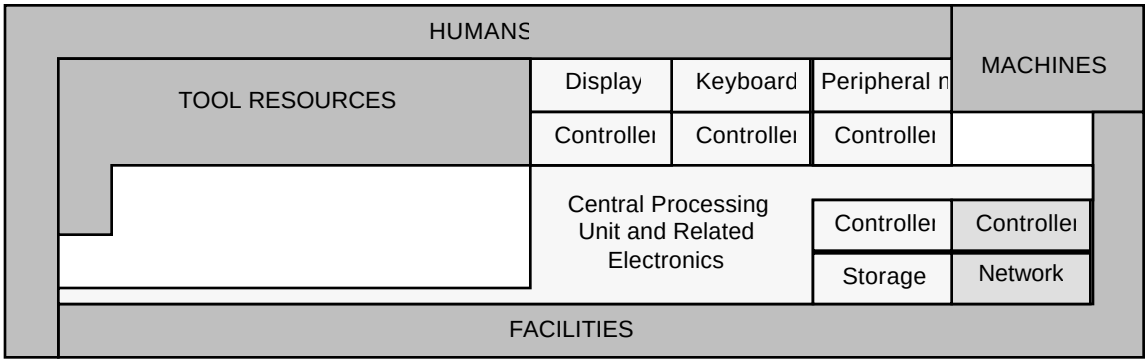
- keyboards,
- mice,
- trackballs,
- two- or three-dimensional joy-sticks,
- touch-sensitive screens or pads,
- laser disks,
- CD-ROMs,

CD or video cameras,
digital scanners,
analog or digital signals from test equipment,
limit switches from machines, or
head, eye or brain potential position sensors for advance look-and-select devices.

They may be input and output devices such as
magnetic disk drives or tape drives,
WORM (Write Once, Read Many) optical drives,
RAM disk "drives",
video or audio tape machines, or
telephone interaction devices.

A typical computer includes a non-volatile storage medium, like a magnetic disk, and a disk controller. They provide a permanent place to store computer programs or data. That which is in computer memory, usually volatile Random Access Memory (RAM), is lost when the computer is turned-off.

A typical computer includes a means of inputting and outputting data. Input is provided by way of a keyboard. Output is provided by a display monitor and its controller. Other output or input peripherals (n) can be added as well. All interface to the central processing unit.



Computers can be classified as personal computers, departmental computers and enterprise computers. Their skills and proficiency can vary widely. A combination of general and special purpose computers that can work cooperatively as well as independently, is currently the better computing resource solution.

6.9.5.1. Personal

Personal computers provide dedicated processing power. As a result, the computer and its assortment of peripherals can be optimized for a likely or intended interactive software tool set. The user interface can be more sophisticated (intuitive, friendly) than can be practically provided on terminals from a minicomputer or mainframe computer. As such, it reduces training costs and increases the productivity of their users.

Examples of personal computers are the Macintosh from Apple Computer and Personal Computer from IBM or their clones. Apple Computer introduced the first commercially successful intuitive and highly productive graphical user interface with windows and pull-down menus. Apple Computer uses a comprehensive library of programming tools to help induce its software tool developers to develop products that have a consistent look and feel. Similar user interfaces are now available from Microsoft (Windows) or IBM (Presentation Manager) or Digital Equipment Corporation (X-Windows), but the lack of programming standards allows the user interfaces of the software tools that run on these computers to be annoyingly inconsistent.

Personal computers normally provide word processing, spreadsheet and two-dimensional graphic design capabilities. Many now also provide *multimedia* capabilities, like coordinated text, 2-D and 3-D graphics, images, video, and sound. They are alternately available with the Unix operating systems (A/UX, Xenix).

Higher performance or more expensive personal computers are often called *workstations*. The power of workstations is such that, they can support the execution of three-dimensional modeling and advanced engineering analysis in a time that is acceptable to an interactive user of those software tools. Workstations typically use a Unix operating system, like AT&T Unix System V with Berkeley 4.2 extensions. They are evolving toward the Portable Operating System Interface for Computer Environments (POSIX) as embodied in the proposed IEEE P1003.1 POSIX operating system specification.

Both the personal computer and workstation can be used as dedicated computers or as network components. As part of a network, they as well as departmental and enterprise computers can share some of the processing burden of other than their direct users. They can cooperatively process distributed processing tasks. They can be diskless and dependent on the permanent storage facilities of a network file server. They can be departmental computers. They are usually maintained by their users. There is little or no support staff needed.

Adding memory or storage capacity to personal computers or upgrading personal computers to workstations represents the smallest incremental change in computing resource capacity available. The costs of such performance changes are relatively small as well. Consequently, personal computers can be more closely *tuned* to the computing needs of their users than can the departmental or enterprise computing resources.

6.9.5.2. Departmental

A function in the business process is often organized into a *department*. To facilitate the sharing of data and software tools that are common to a function, mini-computers or workstations are often dedicated as a network resource. As such, a departmental computer is often called a *file server*. The server may also provide access to special-purpose peripherals, large amounts of departmental

data, or support computation-intensive software tools, like array or vector processing applications. A computation server may be configured from a loose coupling of two or more computers. Departmental computers require a small support staff.

6.9.5.3. Enterprise

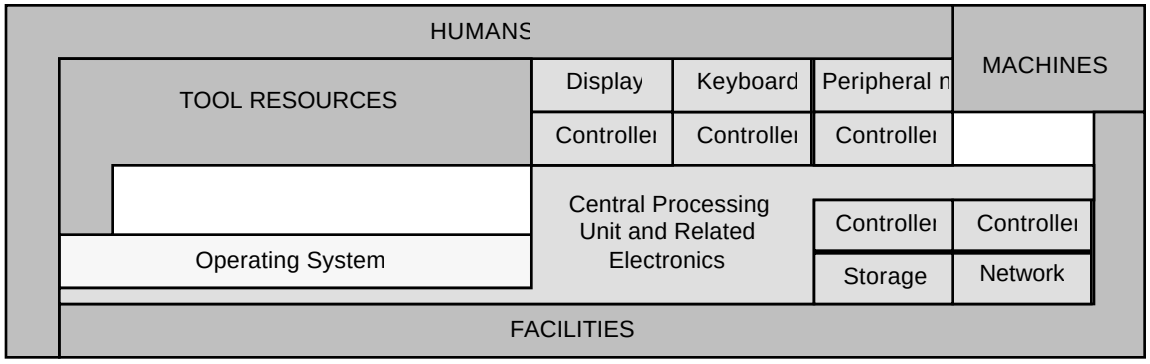
Enterprise computers normally support enterprise planning, finance and payroll functions. They can support large amounts of data. As a shared resource supporting interactive software tools via terminals, enterprise computers provide annoyingly inconsistent processing response times. They may exclude additional users, charge for the resources used and are decidedly less friendly than personal computers. They often require a large support staff. An IBM 3080 and a VAX 9000 are examples of enterprise computers.

Enterprise computers can perform computation-intensive tasks better than personal or departmental computers. The IBM 3090 and Cray computers are examples of computation-oriented enterprise computers. Typically, these processors are accessed for large scale computation by personal computer users who are willing to pay for more power than that which is available from their departmental computer.

Increasing or reducing enterprise level computing resources involves a large incremental increase or decrease in computing power and cost. Since computing needs vary in small increments, enterprise computing resources seldom match the need. As enterprise computing resources are saturated, they become annoyingly slow or inaccessible, but the usage cost is low. Eventually, they are replaced or upgraded. The resulting speed and access is good, but the excess capacity increases the usage cost. The upgrade must be subsidized in some manner or reversed to keep users from abandoning the enterprise computing resource.

6.9.5.4. Operating System

The computers normally have operating system software through which the computer and its peripherals may be accessed directly by application or data management software. Operating system functions include task scheduling, processing, storage and protection of data files, peripheral communications and resource management.



To take advantage of existing computing resources and better price/performance purchase options, it is desirable to have software tools that are readily portable among a variety of computers from a variety of vendors. Conversely, software tool suppliers want the largest possible market in which to sell their products. A standard operating system or tool interface are ways to establish a large market from a disjoint collection of small markets. This has advantages for the competitive computer supplier as well as the software tool supplier and their customers.

To simplify the communication problem among heterogeneous general purpose computers, reduce the cost of implementing core framework software on multiple computers and allow the most cost-effective computer to be used with a particular software tool. The operating system must be one that is or will likely be a standard. Unix is the closest operating system to being a standard. It is the preferred operating system until POSIX becomes an available standard.

The POSIX standard specifies a set of run-time functions and procedures. Software tools that use only this set will be portable across operating system environments. For example, A POSIX-compliant program will run on VMS, Unix and Ultrix without modification. To minimize the cost of a transition to POSIX, any Unix purchased should be POSIX compliant, or at least its vendor should have a low-cost migration path to POSIX and beyond POSIX to OSF-1.

Unix is preferred but not mandatory as the operating system for computers which perform the function of a file or computation server.

If Unix is unavailable for a critical interactive tool, the tool should at least use an X Windows implementation of the common command interpreter. This command interpreter should correspond to that in the OSF Standard Unix or AT&T System V Release 4.0 Unix.

6.9.5.4.1. Recovery

To minimize the need for computer support personnel and the cost of an unavailable computer, it is desirable that all autonomous operating computers be able to recover from an environmental (air conditioning, power) failure or a tool (application program) failure as soon as the disrupting condition (temperature, humidity, line power sensors required) ceases.

6.9.5.5. Utilities

Utility software may be specific to an operating system or acquired separately. They are quite diverse and not easily depicted on a diagram like the one used throughout this section to graphically relate computing resources. The following utilities help maximize the performance of computing resources and protect the data thereon.

6.9.5.5.1. Archive

As newer versions of data accumulate, it is desirable to free disk space by using a utility program for deleting or transferring little used and older versions of data to less expensive media (tape) for storage in less expensive facilities.

6.9.5.5.2. Optimization

If there is not enough contiguous space on a disk to store a file, the file is *fractured* as required to store it in the spaces that are available. After the saving and deletion of many files, eventually the data, programs and free space on a disk will become so fragmented that the computer must spend an inordinate amount of time finding free space large enough to accommodate even small files. This can severely degrade performance. The time required to fracture and aggregate fractured programs and data when they are used further degrades performance.

Optimization software will check for directory inconsistencies, bad blocks or other problems before it attempts to optimize a disk. It may repair the damage or advise that a disk repair tool be used. Then it will use what free space is available to rearrange the bits of data files and programs on a disk until they and the free space are contiguous.

6.9.5.5.3. Repair

Much use of a relatively full disk will cause excessive data fragmentation. Such fragmentation can confuse the extent file, catalog file, volume bitmap and boot block of a disk. A disk repair tool can often correct these problems as well as detect bad blocks on a disk and mark them so they cannot be used. Sometimes an extent or catalog file may become so badly fractured that it cannot be repaired. Then the only alternative is to copy the data and programs to another disk or storage medium (backup the disk) and re-initialize the disk.

6.9.5.5.4. Backup

Backup tools copy data on workstations or personal computers to other storage medium or to a designated departmental or enterprise level computer. Backups can be performed on command or at a time.

The backed-up data is usually compressed to minimize the transfer time and storage space required. To minimize the backup time, the transfer is usually contiguous (streamed). A directory of where certain data is stored is usually the last data written to the backup medium. If a loss of data occurs, the same backup tool can be used to restore the data.

Full backups copy and compress everything on a disk. Incremental backups copy and compress only that which has changed since the last full or incremental backup. Incremental backups should be performed daily. Full backups should be performed weekly. To protect data from loss due to physical damage (fire, earthquake, theft), backup data should be geographically separated from the storage medium from which it was derived.

The ability to perform remote backups is desirable. Backups on unmanned workstations, file servers or departmental computers can then be accomplished from a departmental or enterprise computer where operators are normally present.

6.9.5.6. Security

The utilities described thus far do not protect from the intentional destruction or copying of data. That is the job of security software. There are many degrees of security. The necessary degree is a function of the sensitivity of the data or processing involved. It may be non-proprietary, proprietary or company private from an industrial security perspective, and/or confidential, secret , etc. from a national defense perspective. Computing resources consisting of distributed personal computers, workstations and file servers can easily accommodate classified processing tasks. The data or computers involved are simply isolated from the rest of the enterprise.

Only portions of a product may be classified. During the evolution of a Program, data is often declassified. Metadata about the classified data must be maintained to assure those working in the unclassified environment that there are no "holes" in the product definition. A home for classified data must be provided when it is declassified.

6.9.5.6.1. Unclassified

Update and access protection can be provided via user-id and password pairs that are entered manually or by way of identification badges or other means that uniquely identify the user. These ID/password pairs may control access to a data entity,

field (or object),
set of fields (record),
set of records (file),
set of files (database),
the tool (application program) used to access the data or
the computer on which the tool or data reside.

Security should not be imposed at a session, workstation or tool level, because the data is not protected from methods that can access the data directly. Security constraints should be imposed at the lowest level of data practical.

Typically, data can be viewed by many people concurrently, but it can be changed by only the originator or a designated *owner* of the data. Although some feel that multiple concurrent update capabilities are required to support concurrent engineering, it is not practical. There is really no need for it. A product is always decomposed into parts. Typically designers and analysts are responsible for collections of parts (systems or assemblies). Seldom is there more than one designer or analyst working on one contiguous part. If there is a requirement for concurrent update to one part model, then the model can be further decomposed into groups of features. Each of these could be updated by only one of the designers at a time. The ability to change owners is all that is needed.

6.9.5.6.2. Classified

If defense related work is to be performed on a computer, approval from the Defense Investigative Service (DIS) must be obtained. Some of the rules that may apply are listed here.

- Determine which data and data processing equipment must be protected and how.
- Comply with company as well as Government requirements.
- Control access to the system (visual, physical and electromagnetic).
- Plan for controlling potential damage to the system.
- Ensure that the system resists compromise of its controls through misuse or manipulation of data.
- Restrict the use of the system resources to authorized individuals only.
- Limit those individuals to using only the system resources required to do their jobs.
- Assure that physical access to transmitted data in not possible.
- Encrypt sensitive data. This involves the use of a key and technology approved under the national cryptographic standard. Intervals for changing the key are contractually specified.
- Assign duties and keep records such that individuals can be held accountable for their actions.
- Assign no one to sensitive combinations of resources. Change job assignments frequently.
- The operating system should be capable of preventing one program from interfering with or modifying another.

If the computer is to be used for non-classified work periodically, *sanitization* procedures must be followed. They involve putting the operating system, application programs and all classified data on a removable medium, storing it and any printer ribbons in a safe or other appropriate enclosure, reformatting all fixed disks and powering the computer off to delete anything in its memory.

6.9.5.6.3. Secret

If secret work is to be performed on a computer, the National Security Classification *Tempest* specifications and guidelines apply. Tempest involves the containment of emanations from a computer or any of its peripherals. Emanations are signals of varying frequencies which can carry recoverable information.

Normally a shielded vault is used to contain and ground any emanations from the computer related equipment in it. Some computers, peripherals and their cables have been Tempest-certified. They need not be in a shielded vault, but they must be protected from physical access.

Transmissions between Tempest facilities are secure if no recoverable information is produced from a connection. This is often accomplished using a fiber cable contained in a pressurized tube. Puncture of the tube automatically terminates all communication.

Security software like ACF2, TOP SECRET and RACF is required to protect computing disk, tape, system software, application programs, and control the access or use of peripherals. The user ID and password described in the Data Protection section is normally sufficient to identify the user uniquely, but passwords must be changed within 60 days. Those not updated are voided.

The security software should limit users to only those resources for which they are authorized. It should log all data accesses and issue a violations report to the security administrator. It should disable a user ID with three violations. It should log the opening of files which contain sensitive data (e.g., estimating data) even when the access is authorized. It should log the use of systems utilities which have the ability to bypass the security software. It will restrict access to all resources unless a special grant of access is given. This is the opposite of leaving access open unless specifically protected.

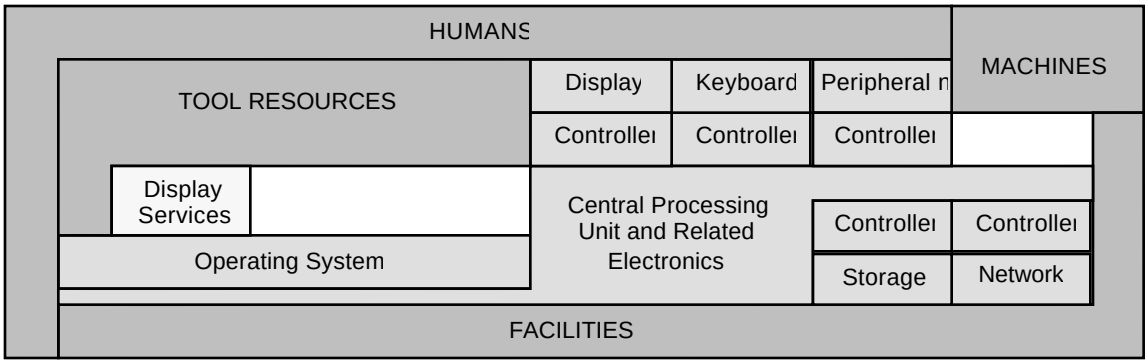
6.9.5.6.7. Display Services

Originally, everything that the operating system was unable to do had to be accomplished by an application program. File handlers, display drivers … everything had to be written by the application programmer. That made even simple computer tools very large and complex and established a practical limit to tool sophistication.

Each these lower level functions were similar from tool to tool and computer to computer. Programmers developed libraries of these functions and

programmed their tools so the functions could be re-used with little or no modification. Some programmers exchanged their function libraries with other programmers.

Finally, some computer manufactures, most notably Apple Computer, provided the functions as part of the computer (firmware). Each tool developer could then benefit from a set of low-level functions and thereby devote relatively more time to high level tool functions.

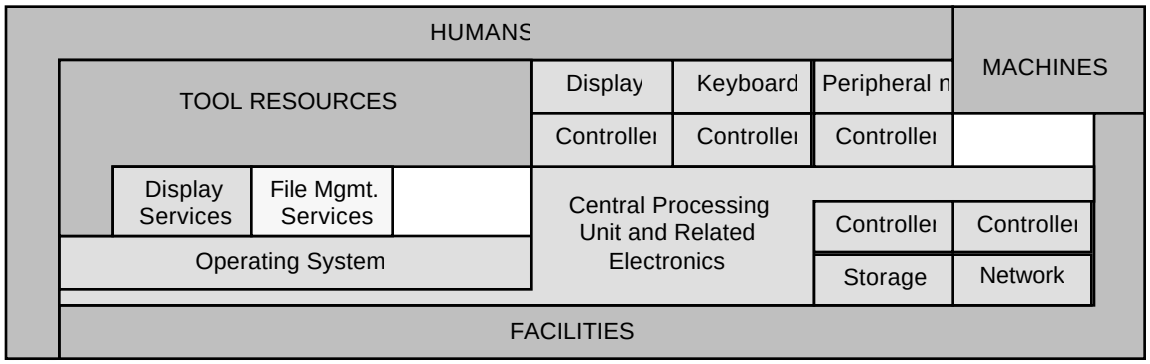


With the advent of X-Windows and Microsoft Windows, the user interfaces of other computers are becoming more like that of the Macintosh. Tool developers who use X-Windows need only maintain one user interface. It will work on any workstation that supports the X-Windows standard (X.11). They thereby minimize their development and maintenance costs while retaining a broad market. Even Apple Computer is supporting X-Windows.

Most computer suppliers are evolving their products beyond X-windows to the Motif user interface standard. It will further reduce the training costs associated with learning to use a new computing resource or software tool. Motif will make it even easier to develop sophisticated, interactive computer tools.

6.9.5.6.8. File Management

Files are collections of data. They are read sequentially (like a tape) from the beginning of the file until the data (characters, sequence of bits) of interest is found. Variations on this theme are the Indexed Sequential Access Method (ISAM) and the Virtual Sequential Access Method (VSAM). These methods provide indexes to selected data to speed its access. Opening files, putting data into files, getting data out of files and closing files are basic file management services.



File sharing in a single computer multi-user environment (mainframe computer) is relatively straight forward. Read or write access is a function of file ownership, which is controlled by user identification and password. Files would simply be copied from the catalog of one user to that of another.

File sharing in a distributed computing environment (network of mini- and micro-computers) requires a more sophisticated file management system. It may take the form of a centralized repository with distributed access, or a directory for all the disks in the network (yellow pages). It may be homogeneous (only computers of the same make and one or more types) or heterogeneous (computers of various makes and types).

The repository approach is like a library. It loans copies of files to distributed users, assuring that there is always a master copy safely stored on the central computer. If the "original" is checked out by the originator or an authorized designate, later borrowers are not granted access. They are notified that the file is being updated and by whom. When the file is checked in, its revision number or date is incremented. If a copy is checked out, it cannot be checked in by the same name. It becomes a derivative file. It may be a new version of the data, like a design variation. It may be a constraint on the design of an adjacent part. It may be the basis for an analysis model. The facilities of the central computer are used to archive and back-up the repository.

In a decentralized approach, the file control system must control access to the files on each of the disks of the many computers in the network. When workstation computers can be turned off or disconnected from the network, distributed file management can be a difficult task. Backup and archival activity must be triggered remotely or performed by the individuals who use each

computer. To avoid the cost of back-up and archival media for each workstation, the data can be copied across the network to a node with such storage devices.

Some distributed file control systems also use file status (e.g., in-work, approved, released) as control criteria. For example, the revision numbers of *in-work* files are not updated. The revision numbers are incremented only after files have entered the approval process or have been released.

The File Control System (FCS) developed by the Data Systems Division of General Dynamics is a centralized repository on an IBM mainframe computer. The FCS supports heterogeneous, distributed access, ownership access control and file status promotion by non-owner users. EDCS™ from Digital Equipment Corporation is an example of a homogeneous, distributed, password controlled access file management system with file status promotion by non-owner users.

The older file control systems only relate ownership and status to files. They do not understand the relationships among the files. They do not even maintain the derivative relationship.

More recent file control systems like InfoManager™ from EDS and EDL™ from CDC include a product structure for relating the files. Product configuration management can be performed with these systems. They include tool encapsulation and invocation facilities. These help automate the check-in and check-out functions and assure that the control system is aware of the significance of all files created by those tools. Simple file management capabilities are inadequate for this task. Such functionality requires sophisticated *data management* capabilities.

6.9.5.6.9. Data Management

A database is a place to store and retrieve information. A data manager does so in a manner that is more granular and sophisticated than that provided by file management systems. Records and fields of data within a file can be directly accessed by a data management system. Data management systems include logging, back-up and recovery facilities as well as data definition, manipulation and protection facilities.

Database management systems store data in individual files and among multiple files. They may use file management systems for file protection and access control. Database management systems may organize information in hierarchical (IMS™), network (IDMS™) or inverted file (M204™) structures, relational tables (ORACLE™) or as object entities (Objectivity™).

Hierarchical structures are like pyramids with higher level entities being composed of (resting on) other entities in the next level below. They can find what goes into what quickly (parts of assemblies in a product structure), but slow when an update is performed. They must search the entire database to

find all occurrences of something that is not hierarchically defined, like an attribute (owner, status) of a part. For example, "find all parts created by Fred Smith" would be very time consuming to do with a database defined as a part hierarchy.

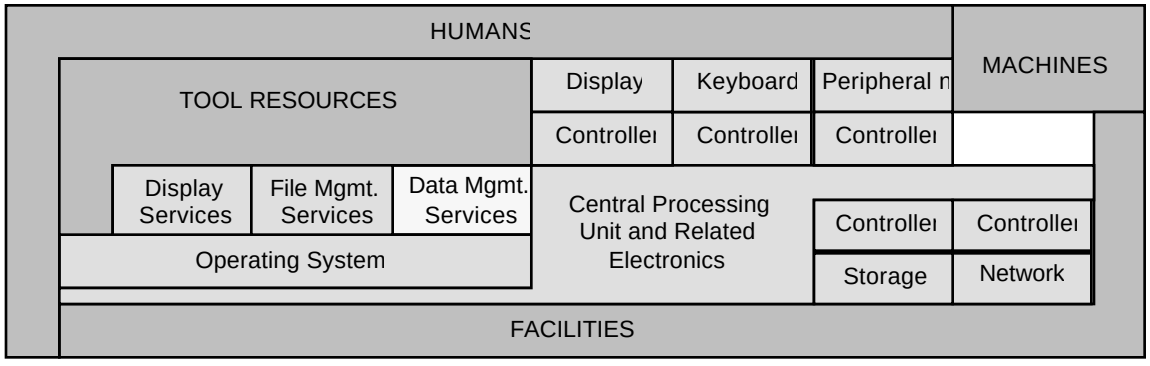
Network structures allow direct relationships between the nodes of a hierarchy (leaves, children) that avoid having to climb the hierarchy to a common node and dive back down to access the desired data. This is a useful technique for improving the performance of often-used searches, but its use must be limited. It cannot be done for all combinations for all such relationships without degrading the overall performance of the system.

Relational structures are relatively simple to understand and use. They can be viewed as little more than many related spreadsheets. As a consequence, users can develop their own relational data management tools with little or no help from a programmer. The strict rules ("algebra") by which relational operations are conducted make relational data management unambiguous, but the performance of a relational system is worse than the other methods mentioned thus far.

Inverted file structures are the inverse of hierarchical structures. As a consequence, they perform updates relatively faster than their access can be performed. Inverted files system like M204™ can be used to imitate all of the data structures mentioned thus far.

All of the data management schemes discussed thus far require that the data types and their relationships be pre-defined. New data of the defined type can be added and their values and relationships changed, but new data types or relationships cannot be added without re-programming the system.

Object-oriented data management OODM allows new objects and relationships to be defined at any time. OODM provides *object persistence*. It keeps objects around after they have been created by the object-oriented programming language used by a software tool. OODM is described in more detail later.



If the application is table-oriented or otherwise simple, and it does not require a high-performance database manager, then a relational database management system may be used. Oracle™ is a popular relational database management

system, especially for distributed computing applications that must run on computers from different vendors (heterogeneous). Even though data management tool development is relatively easy with relational systems, the use of a prototyping and development tool like HyperSQL™ or Omnis-5™ is recommended.

If a graphic data definition and manipulation language is preferred or the application is interactive, or a prototype is desired, especially those that are to run on a Macintosh computer, Double Helix™ or 4th Dimension™ is recommended. If the prototype is to run on a VAX™ computer or on a network of Macintosh and/or VAX™ computers, Helix/VMS™ is recommended.

If a text-oriented data manipulation language is to be employed, a standard form of SQL (no vendor extensions) is recommended.

If multi-dimensional modeling is involved, then an OODM is the logical choice.

6.9.6. Object-Oriented Data Management

Much of what follows was extracted by Maurice Pratt of The J.I.A. Management Group from material written by Ontologic, Inc.

The object-oriented database management approach provides the benefits of object-oriented programming, namely *type abstraction*, *inheritance* and *polymorphism*. Type abstraction means that all objects are typed and they define (and implement) their own behavior. The system can be viewed as a set of components upon which operations can be performed. Where components have only minor differences in behavior or properties, their types can be defined with a super-type relationship. Then the sub-type can inherit all the common behavior of the super-type. Only the different behavior need be defined. Polymorphism means that multiple data types can have the same logical operation defined for them and each of them can implement the behavior appropriately. The same named operation is mapped to the appropriate method for the particular type. The emphasis of the object-oriented approach is on a high level of commonality and reusability with a consistent sharing of services via inheritance.

An object-oriented database not only contains all data objects, but also all of the *meta* information needed to describe the objects, their methods and all system information. Objects may have a structure or may be of variable length. Object database systems explicitly support both types. Complex objects are stored as one unit (no splitting into sub-components).

Object management has been performed by the application programs. Many do not provide object persistence. All the objects created during the session are lost when the computer is turned-off. Object-oriented database management systems (O-O DBMSs) are now available for object-oriented programmers to use as a

database management and graphics system for a design or manufacturing application,

tool to coordinate the usage and inter-relationships of versions of objects, and
basis for integrating several computer tools.

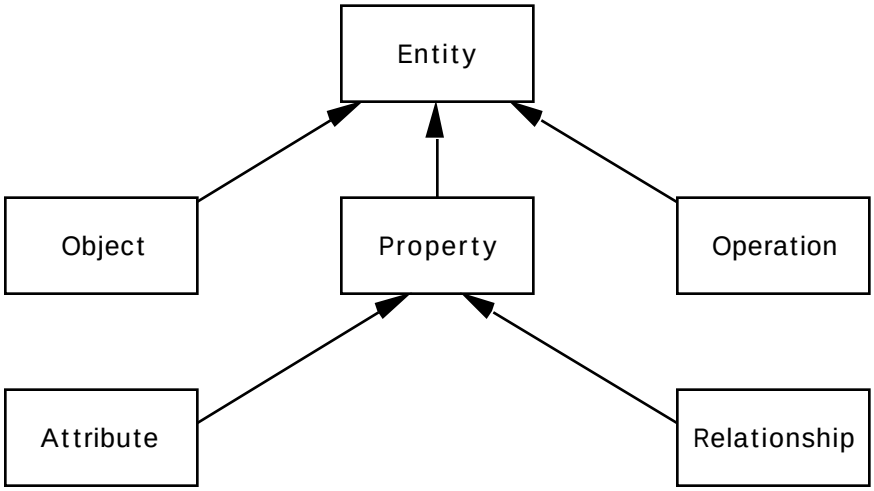
O-O DBMSs support all types of engineering and manufacturing data efficiently, including records, text and graphics. Some employ advanced compiler technology to obtain very high performance, running as much as 100 times faster than relational databases in design applications. Most employ a version and alternative mechanism to track and coordinate the relationships and dependencies of objects as they evolve.

The object model provides an efficient mechanism for data integration and exchange. By bundling the complete semantics of an object along with the data, it is far easier to share and exchange information. The following is a generalized description of O-O DBMSs.

A database definition language (DDL) is used to define types. A database manipulation languages (DML) is used to program operations and to access objects from applications. DMLs are normally implemented as embedded extensions to standard programming languages such as C, Ada and Fortran. The use of an embedded extension to a programming language turns the language into an object-oriented language.

Type definitions are compiled and stored in a library analogous to the "metadata" of a conventional DBMS or the "symbol table" of a compiler. The library differs from the symbol table in that it does not go away at the end of compilation.

High level types are part of the system.



These types define operations that are inherited by each of their subtypes. *Property* defines 'Get_value' and 'Set_value' operations that are inherited by all properties, even those defined on types declared by the programmer. The

Get_value and Set_value operations are automatically inherited by each new object type defined.

Conventional database management systems carried liabilities when they were used as the foundation for design applications (solid modeling). They exhibit inadequate modeling power. Their static schemas do not handle design evolution. The complex consistency constraints that are of central importance in design support applications cannot be specified. They rely on the notion of global consistency, which is not consonant with the gradual evolution of large designs through long term states of only partial consistency. They have poor performance when dealing with complex, aggregate objects.

6.9.6.1. Modeling Power

Commercial database applications may involve a hundred record types. It is possible to look at a diagram of the database schema and understand it. Design support applications may involve a thousand record types, so the simple, one-level schema organization characteristic of conventional database management systems, becomes incomprehensible. Complexity is classically dealt with by abstraction.

Three abstractions which are almost second nature to the way we think are:

an-instance-of
a-kind-of, and
a-part-of.

Certain things are recognized as being true of all lions, for instance, and therefore need not postulate each separately for each lion. Each lion is an-instance-of the type LION, and therefore inherits four legs, a tail, a mane, etc. Lions are a-kind-of jungle cat and we can therefore modularize our knowledge still further. "Mane" would remain unique to type LION. Those things which are true of all jungle cats need not be repeated specifically for lions, leopards, panthers, etc. Finally, the left leg of a lion is a-part-of the lion as a whole, so if the lion moves to the other side of a field, it can be assumed his leg did also; the position information for each piece of the lion need not be recorded separately. The lion is an assembly. The abstractions allow knowledge to be modularized and organized, considerably simplifying the model.

Conventional database management systems (Codasyl or Relational) currently support only one of these abstractions, an-instance-of. In a Codasyl DBMS any record is an instance of a record type. In a Relational DBMS any tuple is an instance of a Relation template. Artificial intelligence knowledge representation systems are the first computer-based modeling systems to deal with very complex problem domains. They have always supported the other two abstractions, as well: a-kind-of and a-part-of. It is in these systems (MIT's FRL, Stanford's KRL and CMU's SRLP) that the notion of inheriting properties and operations down a-kind-of hierarchies first emerged. These early systems were built in LISP as concept demonstrations. They had limitations that made them inappropriate for non-academic use. Performance on traditional machines with interpretive implementations of LISP was poor. They were also limited to

databases that would fit into a LISP workspace, the size of the virtual memory required by a program. Furthermore, the databases were unique to the program using them. The closest these systems came to a notion of the independent long-term existence of the database was to copy the workspace of a process to permanent storage (disk), and allow the subsequent resumption of that workspace by a new invocation of the program. There was no provision for sharing the database among multiple processes, certainly not among processes running on separate workstations in a Local Area Network.

O-O DBMSs support all three types of abstraction characteristic of Artificial intelligence knowledge representation systems, but do so with high performance in the context of shared databases.

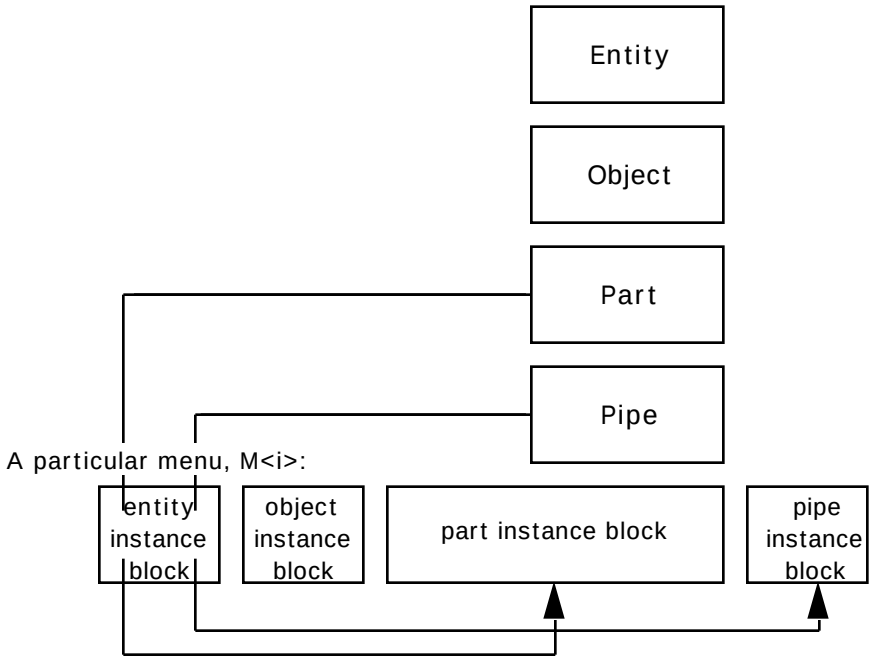
6.9.6.1.1 An-Instance-Of

Objects are instances of types. The type defines the set of properties that each instance carries and the set of operations that can be done on an instance. This is termed the specification of the type. The type also defines a representation for instances of the type (typically in terms of lower-level types or subtypes) and implementations of each of the supported operations.

Programs are constructed as sets of type managers with high level types invoking operations on lower-level types. The type mechanism allows arbitrary object types, not just records to be defined. Complex data types, such as display lists, raster images and documents are supported. The formal distinction between specification and representation allows the type programmer to export only the behavior that is consonant with the abstract object type being modeled. For example, the only retrieval operation permissible on a stack is to "pop" the top element off the stack. This strict separation of the abstract object from its representation is the basis for the high degree of modularity characteristic of applications written in object-based languages (or even in an object-based "style" using a traditional language). It is also one of the key reasons why the performance of an O-O DBMS is high. Since users of a type have no knowledge of its representation, it is possible to build type-specific representations that are highly efficient.

6.9.6.1.2 A-Kind-Of

The power of the a-kind-of abstraction is in the notion of inheritance. If a PIPE is a-kind-of PART, then it inherits all the properties and operations defined on PART. They need not be reprogrammed for PIPE. An individual that is an instance of a lower-level type (in this case PIPE) is also an instance of each type that is a supertype of its most immediate type (e.g., PART, OBJECT, ENTITY). The representation of the individual is a set of instance blocks. There is one for each type of which it is an instance, immediately or indirectly. Each instance of PIPE inherits a portion of its representation (specifically an instance-block) from PART.



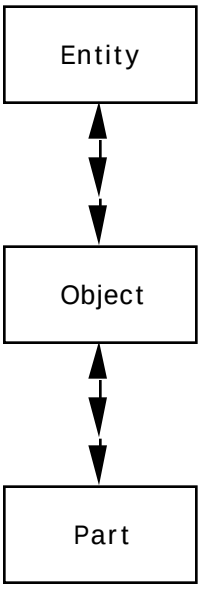
Since the operations it inherits from PART have been programmed to work with the PART instance block, operation inheritance is straightforward at the implementation level as well as at the conceptual level. Yet, a type manager is free to redefine the representation of its instances. It can provide subtype-specific implementations of operations inherited from its supertype(s) to improve performance. The extreme of this is when a type defines the semantics of a set of operations without a representation or implementation. Then each subtype is required to provide its own representation and implementations of the operations that work with the subtype specific representation.

A-kind-of abstraction clarifies the structure of large programs involving many similar types and simplifies programming. It is not unusual for an object-based implementation of a large subsystem to be as small as one-third the size of a more traditional implementation.

6.9.6.1.3 A-Part-Of

The a-part-of abstraction is ubiquitous in design support applications. Things are made up of other things. Design is in fact often a process of specifying the subcomponent structure of a complex artifact.

With an O-O DBMS it should be possible to name the kind of abstract object that occurs at each level of aggregation, e.g., CHIP, CIRCUIT, MODULE, BOARD. The a-part-of abstraction should not be limited to defining a hierarchy between objects. It should support a lattice. It should support property inheritance both up and down the hierarchy.



Each word represents an object type. Each arrow represents a specific instance of the a-part-of relationship. The a-part-of relationship is bi-directional. It can be referred to by either of two names, depending on whether the orientation is that of the component ("a-part-of"), or that of the assembly ("consists-of"). The object types at each level of the hierarchy are named. They are identified in the schema. This allows the type programmer to define properties and operations that make sense for each type. Record-variants, which have different structures depending on which kind of object the record is being used to represent are not required. This can dramatically improve the clarity of the application code that interacts with the database. The maintenance programmer knows by inspection which type the code is dealing with at which instant. It is not invisible, buried in code that does run-time tests on record-variant discriminator fields.

The ability to handle lattices as well as hierarchies is also important in design applications that aggregate objects in two or more ways. Good examples are the logical-physical aggregation hierarchies illustrated for Electronic Computer Aided Design and Technical Documentation applications. In both cases the hierarchies touch at their lowest levels. This is characteristic of a wide class of design support applications. In systems that supported only hierarchical

aggregation, the lowest level components have to be replicated to allow aggregation into different hierarchies. With an O-O DBMS they can share low-level components without requiring the replication of data. This can dramatically reduce storage requirements and remove from the application all the burden of keeping replicated copies of the same data coordinated.

Support for type-specific property value inheritance both up and down the a-part-of hierarchy is another feature of O-O DBMSs. It can reduce storage requirements and improve performance for systems that use many small objects. Each character of a document can be treated as a separate object. An individual character may have 4 to 8 typographic properties (font, size, face, etc.). Most are actually determined by a specification on a much higher level object in the logical hierarchy, like *SECTION_BODY*. If values for these properties can be "inherited" by an object at a lower level in the hierarchy, then they need not be physically stored as part of the representation of each character. A performance-related side-effect of property value inheritance down an a-part-of hierarchy is that *value* and *Set_property_value* operations on the types in the a-part-of hierarchy can be refined to cache current property values in a *state machine*. From there, they can be rapidly retrieved without going to the underlying database.

6.9.6.3. Design Evolution

Conventional database management systems are "shadow" systems. They track the real world. Since the real world has one current state, the database has one current state. That makes sense in the commercial applications these systems were designed to handle. If you are booking reservations for a show and the theater has 200 seats, it does no good to consider a theoretical theater with 220 seats. Twenty people are going to be left at the gate.

In design support applications, however, it is precisely these versions and theoretical alternatives that must be tracked. O-O DBMSs support both *versions* and *alternatives*.

6.9.6.3.1. Versions

O-O DBMSs do not actually store a representation for each instance per se. An instance in the O-O DBMS is actually modeled as a set of distinct versions. There may be a "released" version, a "beta test" version, and a version under development. Individual versions of the same conceptual object may have important differences in property values (range of temperature over which the version is stable, known problems, clock speed, etc.).

The real database in a conventional DBMS is the audit log. The database is really just a cache containing the most recent version of each object in the database. The problem with attempting to use these systems as a foundation for design support subsystems is that there is no way for application programs to access older versions of objects. In a typical commercial DBMS, an application program identifies an object either by a "key value," or by an

"associative retrieval expression." In either case the database returns the most recent version of the object when it is given a key value. This makes it impossible to get at an older version. There is no way of asking for it.

The inability to access versions is why a conventional DBMS cannot support the functionality needed for a design support application. The ability to compare and contrast successive versions of an object or a mutually consistent set of objects is critical. The DML and the query language of a traditional commercial DBMS give the application programmer no way to formulate the questions. The only option is to "circumvent" the system by rolling the whole database back to a point in time at which the version wanted was extant. Then copy the needed version out of the database to an operation system file. Then roll the database forward to its current state, and compare the version in the file with the version in the database. Even then the programmer must know what point in time the version he wants was extant. There is no way of naming individual versions. What the programmer really needs are sets containing mutually consistent versions of several interrelated objects. At best, a programmer is forced to step outside of the system to do what is needed. At worst, the time involved in rolling the database back and then rolling it forward again is so prohibitive that the programmer either builds a version-naming scheme on top of the database, or does not provide this functionality.

By contrast, O-O DBMSs automatically track the evolution of an object through successive versions, and lets the user address them using an object-id of the form:

<conceptual object id>[<version id>].

The <version id> is optional. If it is not supplied, the O-O DBMS should return the most recent version by default.

The most recent version is normally stored in a fully articulated form. Older versions are stored using a backward differential representation. This results in very low storage overhead for older versions. In design support applications it is not unusual to be able to store the 20th earlier version at a 5% incremental overhead in storage space.

The O-O DBMS programmer or user has control over when a set of property value changes is considered significant enough to constitute a new version of an object. This same control extends to how changing one object affects related objects. For example, if object X refers to object Y as the value of one of its properties, does the creation of a new version of Y create a new version of X as well? The most frequent example of this is in a-part-of hierarchies. When a new version of a lower-level component in an a-part-of hierarchy is created, the designer may want new versions to be automatically created at their higher-level aggregate. An example would be a report consisting of two chapters. A new version of Chapter 2 has been created. It percolated up the a-part-of hierarchy to create a new version of the report. Note that the new version of the report shares the single extant version of Chapter 1 with its predecessor.

There are cases when a small change to a low-level component should not ripple through an entire hierarchy, creating a new version of everything above it. The O-O model allows the type definer to control when and how far up the type hierarchy versions are to percolate. Percolation can also be enabled or disabled on an instance-by-instance basis at run time.

6.9.6.3.2. Alternatives

Each version may have a single predecessor and a single successor (linear version path), but complex designs often evolve in complex ways. An O-O DBMS handles design complexity by allowing version paths to fork into alternatives. Alternatives can be individually addressed using the version-id scheme introduced earlier. They may, therefore, be individually modified, compared and contrasted.

A high-level conceptual object like a system design is represented as the root node of an a-part-of hierarchy that ties it to all of its subordinate module designs together. The evolution of lower-level objects common to two alternative versions of the high level object is handled by having both versions of the high-level object refer to the same versions of lower-level objects. They both automatically see the most recent versions of lower-level objects as they are modified.

6.9.6.4. Type Evolution

The interaction between versions and the a-part-of hierarchy allows it to interact with the an-instance-of hierarchy. This gives an O-O DBMS some unique strengths while retaining its conceptual simplicity. No new concepts need to be introduced to handle the evolution of types as well as the evolution of instances. Each version of an instance is tied to the latest version of the type that existed at the time the instance was created.

The representation and even the semantics of successive versions of the same type may change somewhat. There are problems applying operations defined on a new version of a type to instances of an older version of the type and vice versa. This becomes particularly troublesome when there are operations that iterate over all instances of a type, irrespective of the version of that type with which they were created. An O-O DBMS handles these situations by allowing the type programmer to define filter operations that will either permanently convert an older instance to the most recent version of a type, or temporarily create an alternate representation. Then older or newer versions of the operations of the type can execute correctly on it.

In some cases this is not possible. For example, a Get_value operation on properties that were not defined at the time the instance was created and for which no value has subsequently been stored is impossible. Even in this case, the notion of a filter operation allows the type programmer to supply a default operation that will return an UNKNOWN value. This is better than having the

DBMS crash due to incompatibilities between the representation assumed by an operation and the actual representation of the instance involved.

The problem of evolution is very real and costly in large application programs, particularly those that are successful. Such systems may evolve incrementally over 3 to 5 years, and may grow to 200,000 lines of baroque code. Even relatively minor changes may affect tens of modules.

6.9.6.5. Partial Consistency

Concurrency control in business database management systems is based on the notion of global consistency. A transaction takes the database from one globally consistent state to another. There may be some temporary inconsistency during the transaction. For example, during the transfer of funds from a savings account to a checking account, there is an inconsistency after the savings account has been debited and before the checking account credited. This does not change the model, because none of the intermediate states of the transaction are visible to other users of the database until the transaction completes.

In a design database this is too restrictive. A design database may not achieve a globally consistent state for weeks or months. In fact, it may never do so over the period that it is useful in supporting the design process. The point at which it achieves consistency is by definition the point at which the design is complete. At any particular point of its evolution, versions of portions of a design may be consistent with other portions of the design. However, conventional database management systems do not keep track of this dynamically changing set of partial consistencies. The result is building designs where the heating ducts go through the elevator shafts.

By contrast, an O-O DBMS has facilities that give the builder of design support applications powerful ways of modeling the constraints, ways that may be unique to the application. These include declarative specifications of the state of an operation that require an object on which it is going to operate to be available before it will execute correctly. They require a notion of consistency that suffices for tracking the state of particular versions of objects and the degree to which they are consistent with versions of other objects. These constraint definition capabilities can be very useful to the builder of a design support application, because modeling constraints can be as important as modeling data.

6.9.6.6. Performance

One of the complaints about using conventional database management systems for engineering design applications is poor performance. O-O DBMSs provide a higher level of functionality than do conventional database management systems. Although the usual consequence is a lower level of performance, there are reasons why performance in an object-based DBMS can be dramatically better than in a conventional DBMS. One is the knowledge

of the semantics of an object type, which allows the type programmer to exploit constraints unique to that type. The other is the distinction between semantics and representation, which allows the representation to be tailored for performance.

Knowing semantics can yield better performance. In a research effort at the San Jose Research Lab of IBM, a CAD/CAM system was built on top of the Relational DBMS test vehicle of IBM called SYSTEM-R. One of the principal stumbling blocks reported in that effort was its poor performance with complex aggregate objects. Retrieving a simple 4AND gate electronic circuit comprised of three 2AND gates, required separate access to 22 tuples in 5 separate relations. If the database was local to the workstation, access time was bad. If the data had to be requested across a LAN from a database server, the time delays were intolerable. One solution was to add to the relational model a notion of entities that had unique identifiers. Another was to add a notion of aggregation, so the designer could specify precisely what lower-level components comprised a particular circuit. The result was a model like the O-O DBMS a-part-of hierarchy. The a-part-of abstraction is built into an O-O DBMS, so the O-O DBMS knows that the 4AND is a composite object. Therefore, it can be brought into memory or retrieved from a shared database server as a whole with a single request, not dozens of individual requests.

Keeping the distinction between semantics and representation straight allows tailoring representations for performance. O-O DBMSs support different representations for different types, different representations for different instances of the same type, and single instances that may have two representations at once.

Each of these has important performance implications. Conventional database management systems have no notion of semantics. They support only a single representation of type: "record." For many kinds of objects, like text and graphics, record is a hopelessly inappropriate representation. Attempts to apply relational database management systems to office automation applications are embarrassing. People try making a record out of each character, stuff a fixed number of characters in the successive fields of a set of records (ORACLE), put into the field of a record a pointer to a Unix file containing the text (INGRES, IDB), or to break the model and introduce another storage type ("blobs" in JRD).

A system that allows each type to build its representation out of whatever lower-level types are appropriate is required. O-O DBMSs supply a rich set of low-level representation types (structure, array, set) as well as a suite of higher-level representation types tailored to graphics and text_pixel_array, descriptor_list, text_block, etc. The type programmer is free to define type-specific representations that are efficient for the types of objects involved, e.g., pixel_array for raster images that are going to be manipulated with BITBLT operations.

It is also possible for the type programmer to define two or more alternate representations for a type. Any given instance of that type may be based on

either of these representations. This allows efficient handling of cases in which the instance-level variation is large enough to warrant separate performance treatment. For example, the default representation for small sets might be a linked list, but when the set grows to several thousand members, a B-tree representation may be more efficient. The type programmer can specify daemon routines that automatically convert an instance from one representation to another when the second representation becomes more efficient.

Another advantage of O-O DBMs is that they allow a single instance of a type to have two or more representations simultaneously and vector operations to whichever representation is the more efficient. A good example is nodes in the image composition tree representing a complex picture. If the subpictures the nodes represent can be stored as both raster-images and descriptor-images, then operations such as Move can be done quickly with graphics processor hardware on the raster-image. The descriptor-image representation could be revised after the fact. Essentially one representation could act as a cached version of another, capitalizing on the different performance possibilities of different operations defined on the same type.

The data structures used to represent objects are stored Object Memory. Object Memory consists of a per-workstation Active Memory, and a per-network Inactive Memory. Inactive Memory is logically global to the network. Physically, it is partitioned into areas, each of which is mapped to a partition on a disk pack mounted on some node of the net.

The implementation of Active Memory has a large impact on performance. Active Memory is implemented as a shared virtual memory segment mapped into the address space of all processes on the workstation that are using the O-O DBMS. There is no IPC or context switch overhead when making a database call. There is no "copying" of data from DBMS buffers into application program buffers, or "piping" of records from DBMS to application. This can make a dramatic difference in performance. An IPC call in Unix takes 10-14 milliseconds, in VMS 7 milliseconds or AOS 10 ms – nearly one-third of a disk access.

Protection is traded for performance. Since object memory is mapped directly into the virtual memory address space of the processes using them, it is possible for an errant application program to overwrite the representation of objects in active memory. This concession makes sense because the O-O DBMS will not be something that is directly available to end-users. The O-O DBMS is distributed as a foundation on top of which the application builder constructs a discipline-specific application (e.g., semi-custom VLSI design, Structural Design). The designer interacts with the application. The application interacts with the O-O DBMS. In this sense, the environment in which the Object Manager executes is a closed one. The application developer will presumably have eliminated any problems with the system before its delivery to customers. They should be comfortable that the software no longer needs to be protected from itself.

During the development stage, the application builder may prefer to make the opposite trade - protection for performance. This option is provided in O-O DBMSs. The O-O DBMS may be invoked as a separate process. Its applications are linked with a checkout library that makes IPC calls to the O-O DBMS. In this case, object memory is mapped into the address space of the O-O DBMS. It is not directly accessible to the application process.

Programming with type managers that export a well-defined set of operations, but do not make the representation of the object visible outside of the type manager tends to result in significantly more error-free code. Getting an application to compile using a strongly-typed object-based language takes longer than with a more permissive language. However, once it compiles, it runs. The 50-60% of the program development cycle usually spent on debugging shrinks to 10-20%.

6.9.6.7. Object-Oriented DBMS Applications

The following application descriptions exemplify the advantages of O-O DBMSs. They indicate that the programs that implement design support applications can be much smaller than they would otherwise have to be using older, less functional graphics modeling systems. Consequently, these design support applications can be brought to market in less time with more functionality and with lower maintenance and enhancement costs over the life of the application.

6.9.6.7.1. Graphics

Some O-O DBMSs, like an earlier product from Ontologic called Vbase, have an extendable, three-dimensional hierarchical graphics subsystem. It serves as a base for more extensive graphics subsystems written by applications developers. It also serves as a model of how the O-O DBMS can be used to integrate display list and pixel graphics with other engineering data representations.

The O-O DBMS implements a basic set of types (e.g., ENTITY, OBJECT, PROPERTY, OPERATION, etc.) and exports operations on these types to the graphics subsystem. The graphics subsystem adds a layer of more specific types (e.g., WINDOW, CANVAS, DESCRIPTOR_NODE, etc.) and exports operations on these types to application subsystems. Application subsystems are implemented as a third layer of still more specific types. One application may define types used in VLSI design; another may define types used in Mechanical CAE; a third may define types used in Architectural Engineering (e.g., BEAM, COLUMN, TRUSS).

At the lowest level of the graphics subsystem, the "drawing primitives" (e.g., CIRCLE, LINE, ARC, RECTANGLE) are simply another set of primitive object types. The type LINE, for instance, has a well defined set of properties (location, length, line-style, width …) and a small set of operations (draw, undraw …), like

the type INTEGER with its properties ('value') and its operations ("add," "subtract," "multiply," etc.).

The image composition level of a graphics subsystem is supported by the O-O DBMS. Hierarchically structured images, instancing support, and the ability to merge raster-based and vector-based subpictures into a single picture are provided by abstraction. Hierarchically structured images are based on the a-part-of relationship. Instancing is based on the an-instance-of relationship. The ability to merge raster-based images into higher level composite descriptor-based images is predicated on the a-kind-of relationship.

6.9.6.7.2. High Level Hierarchical Image Composition

There have been three major standardization efforts in vector-based display list graphics: CORE, GKS, and most recently PHIGS. The CORE standard was both two-dimensional (2-D) and three-dimensional (3-D), but made little provision for interactive graphics. The GKS standard was intended for the emerging interactive graphics tools. It currently defines a 2-D system. A 3-D extension is under development. PHIGS supports both 2-D and 3-D and is also interactive. Its principal departure from the earlier standards is that where both CORE and GKS build images out of a flat collection of segments, PHIGS allows segments to be structured into a multi-level image.

In all three standards a picture is built-up from segments. Segments consist of a series of drawing primitives (instructions to a graphics processor to draw lines, fill rectangles, draw polygons, shade areas, etc.). Segments may also contain attributes and transformations. Attributes determine such things as line color, line style, the pattern with which surfaces are to be shaded, etc. Transformations define how the primitives collected into a segment are to be positioned in the rest of the picture. For example, the drawing primitives that define a bicycle wheel might be defined in a coordinate space unique to the wheel. When the wheel drawing is incorporated into the drawing of the entire bicycle, each wheel has to be positioned within the coordinate system of the bicycle. It may have to be scaled or rotated as well. These transformations are termed modeling transformations. Both are conducted by placing a 3x3 (for 2-D) or 4x4 (for 3-D) matrix into the first entry of the segment.

In either of the two older graphics systems, CORE or GKS, the stored graphics model (the "descriptor list") is not capable of reflecting that hierarchical decomposition. A CORE or GKS image is created out of a flat series of segments. Each is logically a leaf node of the image hierarchy. If the application builder wants to provide the ability to move a subpicture (e.g., change the angle of the front fork and move the front wheel forward 8 millimeters), the application program will have to remember which drawing segments were involved in the wheel. It must adjust the transformation matrices at the head of each of these segments accordingly. The structural decomposition of the picture must be maintained by the program, either in a separate data structure, or in the logic of the subroutine calling structure. It cannot be carried by the declarative structure of the display list.

In PHIGS, the display list assumes this function. It is possible to create a hierarchy of segments that decompose the drawing in exactly the same way the object itself is decomposed. Transformation information is "inherited" down the hierarchy. The position of the spoke holes in the hub is defined relative to the coordinate system of the hub. The position of the hub is defined relative to the coordinate system of the wheel. This makes it simpler to move/delete/replace entire subtrees within the picture. This is a very frequent operation in design support systems. It allows the designer using the system to modify his design by modifying the picture through which the design is presented to him.

Such applications often involve 100,000-200,000 lines of code and represent 40-50 man years of programming. If the underlying graphics modeling system provides direct support for creating pictures as hierarchies, the cost to construct new applications can be radically lower, and time-to-market radically shorter.

In the PHIGS draft standard, the structure of a picture display list is not limited to a strict hierarchy. If the same sub-object appears in the picture several times, it need be represented but once as a single segment. The positioning information that controls how it is "instanced" into the larger picture is detached from the segment proper. It is logically attached to the relationships between the shared object and the objects of which it is a part.

This capability is provided by way of the abstraction capability underlying the graphics subsystem. For example, the arcs from node to node are an a-part-of relationship (or conversely, the Consists-Of relationship depending on whether you are looking up from the spoke or down from the wheel). Position information can be attached to that relationship by way of the type hierarchy: RELATIONSHIPs are a-kind-of PROPERTY. PROPERTIES are a-kind-of ENTITY, and any ENTITY can have PROPERTIES. Therefore RELATIONSHIPs can have PROPERTIES. In this case the transformation information that controls how the coordinate system in which the spokes are defined is mapped into the coordinate system in which the wheel as a whole is defined.

6.9.6.7.3. Integration of Vector and Raster Graphics

There have historically been two approaches to graphics: traditional vector-oriented, display list graphics, and the newer Xerox STAR style of graphics. They arose from different technologies. Display List graphics originated in the time of stroke vector graphics terminals. The primitives of display list graphics are line, rectangle, etc. These are combined into segments for display. Image composition is done by placing matrices at the head of each segment. These matrices define how the coordinates of the primitives in the segment are to be mapped into the coordinates of the image.

At Xerox's Palo Alto Research Center, a different way of looking at graphics was defined in the 70s by the BITBLT operation. The Xerox workstation prototypes had large bit-mapped displays. Pictures were inherently thought of as arrays of pixel values. Images were composed by pasting pixel arrays together and

combining them with logical functions (and/or, xor,...). This was particularly natural for text. Rather than drawing each character as dozens of small lines, each character was defined as an array of pixel values (a different array for each font and size combination). These arrays were then copied into rows to develop lines of text.

Since memory is linear, what was intended to appear on the screen as a single rectangular block of pixels, could not be stored in memory as a rectangle. Pasting one conceptual rectangle beside another required fairly sophisticated support to deal with the internal representation of the pixel arrays. This was encapsulated in the BITBLT instruction. It first appeared in the Xerox Alto in the early '70s and has since become a standard feature of most raster graphics terminals and workstations, whether in microcode or silicon. The Xerox style of graphics (windows, menus, icons) saw its first commercial use in the Xerox STAR. Mass market penetration came the next year with the Apple Macintosh.

Unfortunately, these two styles of graphics are still desperate. It is not infrequent to find a Window Manager in the style of the Xerox model, and a CORE package that supports display list graphics. Unfortunately, there is no way of incorporating subpictures defined as pixel arrays into the display list hierarchy.

In an O-O DBMS, these two styles of graphics can be smoothly integrated. As was noted earlier, pictures can be defined as a tree of segments. The key difference between this and the display list technology is that the nodes are not restricted to be descriptor lists. They may be either descriptor lists or pixel arrays.

6.9.6.7.4. Libraries of Pre-Defined Types

Many design support applications give the designer access to libraries of pre-defined components. These libraries typically contain drawings of the components, or may contain the 3-D geometry information from which drawings can be derived. The designer then "instances" these pre-defined components into the image, always by giving them a particular position in the drawing, sometimes by scaling or rotating them as well.

Common examples are libraries of chip types in logic design applications. The library may define several different types of memory chips of different capacity (64 Kb, 256 Kb), different configurations (64K by 1, 16K by 4), different performance, different electrical characteristics, different heat dissipation. The designer of a memory array card selects the most suitable chip type from the library and creates instances of it in the design.

The templates stored in the library are called types. The individuals placed into the design are instances of those types. The types define the properties carried by instances of the type. Instances define values for these properties. In some cases, the type may constrain the range of property values that are legal for an instance (e.g., power dissipation of 20-40 milliamperes). This makes the

specific value a function of how the designer chooses to use the instance in the particular design. In other cases it may specify a value that is true for all instances of the type (e.g., width 0.04 mm, height 1.20 mm). This is an example of the an-instance-of relationship discussed earlier combined with property-value constraints and type-defined property values.

The a-kind-of abstraction is also central to the library notion. For example, chip libraries may contain literally hundreds of different chips. A common way of organizing them is to group them into a-kind-of hierarchies. Then a designer can find a set of chips of potential interest and select the most appropriate chip.

If the selected chip type is used several times in the design, the lattice structured a-part-of and a-kind-of relationships are useful. For example, on a memory array card there may be several dozen instances of the same chip type, each driven exactly the same way. They differ only in their location on the board. If the display list representation extracts the positioning information as part of the a-part-of relationship between the board and the chips it contains. Then the same display list segment can be used to draw all the chips.

6.9.7. Data Dictionary

To avoid confusion, the names and meaning of data must be understood. This is data about data, *metadata*. Metadata provides the unique identity of data and its units (inches, meters) so that it will be properly used.

Metadata is usually kept in a data dictionary for coordinating application development, particularly the avoidance of redundant data entities. The data that must be known to a data dictionary for it to be effective, need only be that which is shared among computer tools. The dictionary must contain metadata about independently developed computer-based tools. Hence, it must accommodate data with the same name but different meanings (synonyms). This is usually done by using the tool name as a prefix for the data name to identify it uniquely. Conversely, the data dictionary must accommodate data with different names but the same meaning (homonyms). This is usually done by way of aliases. Since product description data "evolves," the data dictionary must accommodate *versions* of data with the same name and description.

The data dictionary must be "active" if it is to be aware of new data as it is defined. Before data is saved for the first time on a local storage device by a tool, the tool must provide the name and description of the data to the data dictionary. If the name or the description indicates that the data exists, the tool should so notify its user, and give the user the opportunity to change the name, description or version of the data before re-invoking the save command. If there is a conflict with the information in the data dictionary, then the metadata cannot be registered with the data dictionary. The tool is not allowed to store the data.

A data dictionary can also be used to manage product data, but in either case, a data dictionary is most useful when it is used with a data directory.

6.9.8. Data Directory

Should any human or computer resource want to access data, they need to know where to look for it. A data directory provides that service. It should indicate the

network,
node (computer) in the network,
storage device (disk) on that node,
file on the storage device and
field or object

in which the data is to be found. In some cases, the nodes maintain internal path names, so the data directory need only concern itself with the node. If a data request is presented to such a node, the node computer will find the requested data according to its internal directory.

The data directory must be "active". It must be made aware of the location of new data as soon as it is stored.

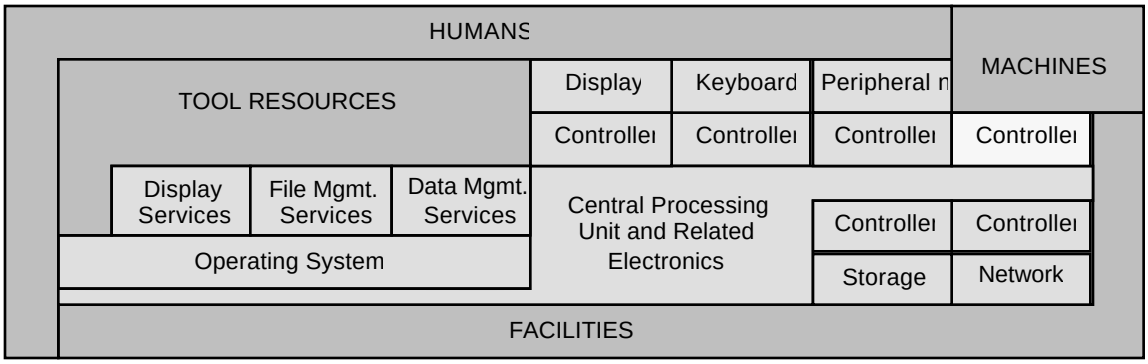
The use of a data directory need not be restricted to digital data. Some data like sound and image data may be more effectively stored on audio or video tape or film. Legacy and other data may not be in digital form. It may remain on paper or microfilm indefinitely, but must be maintained as part of the product definition for products that are still in service.

Data names are redundant to data dictionaries and directories, and their interaction with tools is similar. Consequently, data dictionary and directory functions are often combined into one tool called a data dictionary/directory. A data dictionary/directory can be used to manage the product and tool data.

A separate data/dictionary should be used for each Program. Appending the Program name to a data entity is a simple way of distinguishing enterprise data among multiple Programs. It avoids the bureaucracy involved with reconciling entity names among Programs.

6.9.9. Machines

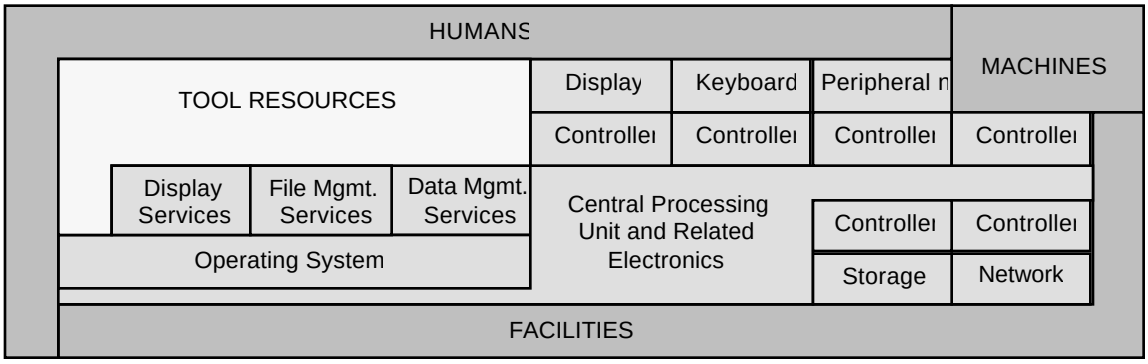
Computers are used to manipulate machines as well as displays. There are a wide variety of computer controllable machines (mills, lathes, robots, missiles, automobiles, etc.). Each machine model has a unique controller.



Consequently, the specific instructions for a controller are generated (post-processed) from a general representation of machine motion. The Compact II and Automatic Programmed Tool (APT) programming languages are commonly used to create generic tool motions and specify machine settings (feed, speed, coolant). The machines themselves are described in the Machine Resources section.

6.9.10. Tools

Tools conduct or help human resources conduct work. Tools that help define and coordinate the work are required in addition to the tools needed develop deliverables.

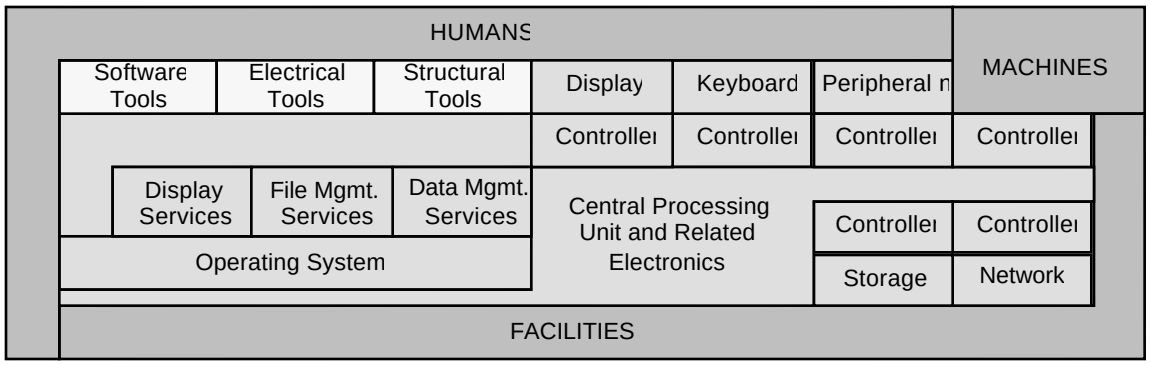


These tools are described in the Tool Resources section. A synergistic benefit can be derived from integrating their use.

6.9.10.1. Tool Integration

Efforts to integrate tools were naturally organized along manufacturing lines of demarcation. Software, electronic/electrical and structural/mechanical products

have distinctly different manufacturing processes. *Tool domains* were the result of this orientation.



These domains of integration are aggregations of smaller domains organized along disciplinary and sub-disciplinary lines of demarcation. Within the structural domain for example, aeronautical engineering integrated their tools. Within aeronautical engineering the subsonic, transonic and supersonic tools were often integrated separately. Within the software domain, Computer Aided Software Engineering (CASE) tools were integrated as a function of the software language (FORTRAN, COBOL, C, etc.).

Sometimes this integration takes the form of a single, large software tool with many different functions. For example, a vendor of tools for electronic systems may try to integrate printed circuit board layout, analysis and packaging into a single tool. Each function is a different subroutine or set of subroutines in a single computer program (software tool).

This approach soon becomes cumbersome and costly to maintain. Even if portions of a tool can be separately compiled and debugged, they have to be linked with the remainder of the program before the entire tool can be exercised. As each area (design, analysis, manufacturing) became a sophisticated speciality, separate tools were developed for each discipline in each domain. As a result, each part of the organization of an enterprise could do their job well, but there was no net benefit to the enterprise. The benefits gained by the isolated tools were lost in the effort to transfer data among the tools, or from tools to human resources without computer tools, or vice versa.

This predicament prompted attempts to automate the transfer of data among the tools. This it called *tool interfacing*. There are various degrees of tool interfacing from file conversion to common databases.

In every case, the tool deals directly with the computer on which it is hosted. The tool vendor must re-host the tool onto other computers to expand the market for the tool. Tool customers must buy new computers to have a place to host tools that will not run on their existing computers. This predicament prompted the call for a standard operating system and the move to Unix as the *de facto* standard.

To further improve the net benefit to the enterprise, the elimination of any *slack time* is becoming more important ("fast cycle," "time to market"). Slack time is the period between the availability of data and its use by a subsequent function in the business process. This requires some degree of tool control. Autonomous (batch) and interactive tools must be automatically invoked. Their termination must be detected. The preparation and dispersal of their data must be automated. This is called *tool integration*.

Key benefits of integrated tools are data consistency and the elimination or mitigation of redundant data entry by human resources. For example, a designer need not notify a data dictionary/directory tool of the existence of a new model or inform a scheduling tool that a task has begun. In an integrated environment, the very act of creating a new model with a design tool and saving its file for the first time would trigger a query for additional model, file or task attributes. The meeting of those constraints would trigger the automatic notification of the data dictionary/directory and the scheduling tools.

The data of unintegrated tools is usually inconsistent, because from the perspective of a designer, the "real world" is the design tool. It is not some annoying product data or process management tool. Consequently, the data of what are felt to be superfluous tools is not updated regularly. It is periodically not representative of the reality of the designer.

There are various degrees of tool integration from a library of programming functions to formal application programming interfaces to frameworks. The more sophisticated forms of tool integration involve an intervening layer of software that isolates the tools from the computing and data management environments. Although this eliminates the need for a standard operating system from the perspective of tool vendors and users, a standard operating system would simplify the task of developing tool integration environments.

Tool Interfacing and Tool Integration and their various implementations are described in the following sections.

6.9.10.2. Tool Interfacing

Tools may be interfaced by way of entire files or the data therein.

6.9.10.2.1. File Transfer

Tools may be interfaced by transferring files among the tools. This tool interfacing approach requires no modification of existing programs. It may require the development of data format conversion utilities. These may be treated as separate tools or as part of a tool (subroutine). There are various ways to transfer files.

6.9.10.2.1.1. Reformatting Utility

Tools may be interfaced by copying and converting data by way of an intermediate data format conversion utility to the specific format of each tool. Such utilities are usually unidirectional. They import or export files, not both.

The utility must know the position of the data in the file and perform any necessary units conversion. It may have to operate mathematically on the data and combine it with other data or otherwise manipulate it to make it be exactly as the using tool expects.

This approach is the most common. It requires import and export data format conversion utilities for every pair of tools so interfaced. All affected data format conversion utilities must be changed each time the data format of a tool is changed. This may also require that the data access method of the other tool of the interfaced pair be changed. Although the cost to implement may be relatively low (no concurrence among tool developers required), the cost to maintain reformatting utilities is high.

6.9.10.2.1.2. Neutral Format

The need for a data format conversion utility for every tool pair can be reduced with international or *de facto* file exchange standards. This approach requires each tool vendor to provide conversion utilities for importing and exporting data in the *neutral format*. Applications of data are unpredictable, so such neutral file formats must be robust to be comprehensive.

The Initial Graphics Exchange Specification (IGES) is an approved U.S.A. and European standard for the transfer of the geometric and drafting data typical of structural/mechanical Computer Aided Design, Computer Aided Drafting, Computer Aided Analysis, Computer Aided Engineering and Computer Aided Manufacturing systems (tool sets). The Electronic Data Interchange (EDI) is an international format for electronic data transfer among dissimilar electronic design and analysis tools.

TIFF, PICT and MacPaint™ are three of the many viable *de facto* graphics data transfer standards for bit-mapped (paint and video) images and line (arc, circle...) drawings. Many popular graphics programs (MacDraw™, Canvas™, Superpaint™, ColorEyes™) will *import* more than one or more *foreign* formats. Similarly, MacWrite™ and Word™ have become *de facto* text processing standards and many text editors or word processors will import their formats. For example, SoundEdit™ is a *de facto* standard for storing and transferring sound files.

6.9.10.2.2. Data Transfer

Tools may be interfaced by transferring a portion of a file – data. Data may span more than one file. This tool interfacing approach usually requires some modification of existing programs, but allows a lower *granularity* of data to be

transferred. An entire file does not have to be converted to an acceptable format. Only that portion of the file that is required, needs to be reformatted. There are various ways to transfer data.

6.9.10.2.2.1. Cut-And-Paste

An interactive form of data transfer is the cut-and-paste method of manually selecting a portion of a text or graphic file and *cutting* or *copying* it to a temporary holding area (clipboard); then *pasting* the contents of the clipboard into a different file. This is an intuitive, easy to learn method of data transfer made popular by the Macintosh computer, but it requires a human operator.

6.9.10.2.2.2. Live Links

By providing the path name (network and file address) of pasted data to a tool that keeps track of such information, a *live link* can be maintained between the source and target documents. It can insure that the target document has the most current data (Excel spreadsheet or MacDraw graphic pasted into Word document). Each time the source file of pasted data is changed, the pasted portion is automatically updated in the target document. This avoids the necessity of having to repeat the cut-and-paste operation manually each time a source document is changed.

6.9.10.2.2.3. Common Database Access Method

Access routines that use the Standard Query Language (SQL) can access the data of any database management system that supports SQL. Using vendor extensions to SQL will eliminate this versatility. Unless data administration is performed rigorously, independently developed tools will seldom have the same name for identical data. SQL routines developed later must use the data name appropriate for each system it searches. If the data names or path name changes, the access tools must be modified, accordingly. This is similar to the problems associated with the file transfer method, but it involves only the data transferred, not the entire file.

One way to compensate for different data names is to have the tool first query a data dictionary to determine what data name aliases exist. Then have it incorporate any found aliases into its query or update. With this approach, new tools or their modifications should not be implemented until the data dictionary data is made current. This delay can be avoided with the use of an active data dictionary. The degradation to tool performance resulting from the extra query cannot be avoided.

One way to compensate for different path names is to use a data directory in a manner like that described for the use of a data dictionary. Some data dictionaries include directory (where used) information. Although the use of a dictionary/directory simplifies tool development and maintenance, it does not improve tool performance. Furthermore, a staff of human resources must maintain the metadata in the data dictionary and/or directory accurately.

Data dictionary and/or directories may be "active". Like live links between the files, links can be established between tools and a data dictionary and/or directory This would automate the update of the metadata whenever a tool is modified. It reduces, but does not eliminate the data dictionary and/or directory maintenance.

SQL is a data manipulation language. It is not a data definition language. It provides no means for a tool to create new data of a type not already defined. The data definition language of the source tool must be used to define new data.

6.9.10.2.2.4. Data Format Standard

PDES/STEP is an evolving standard for defining product data schema. This standard is part of the Computer Aided Acquisition and Logistics Support (CALS) initiative. Under the CALS initiative the U.S. government will require contractors to support the transfer of digital data among contractors and DoD organizations, including operational units.

The PDES implementation plan identifies four levels of capability. The first two levels of the PDES implementation are extensions of IGES. The third level is direct data access by way of database management systems. The fourth level will support knowledgebase related access, like that required by expert systems. Proprietary data is not inherently protected from this method of data exchange if direct access, like that promoted by DoD, is granted. Until the issue of protecting proprietary data is adequately addressed by PDES, the data of one company should be selectively duplicated onto an *isolation computer* for access by DoD or other companies. The isolation computer is physically disconnected from the outside world and then connected to the resources of a company. The pertinent data is transferred to the isolation computer. The isolation computer is then disconnected from the inside world and connected to the outside world for access by DoD or other companies

Many PDES/STEP proponents advocate PDES/STEP as the native data format for new tools (common DBMS approach). The disadvantage of this approach is that it will hamper innovation. Tool performance may suffer if a sub-optimal data storage method is used. For these reasons, most tool vendors are planning to provide data reformatting utilities to import or export data in PDES/STEP format, yet retain the storage formats that are optimum for their products.

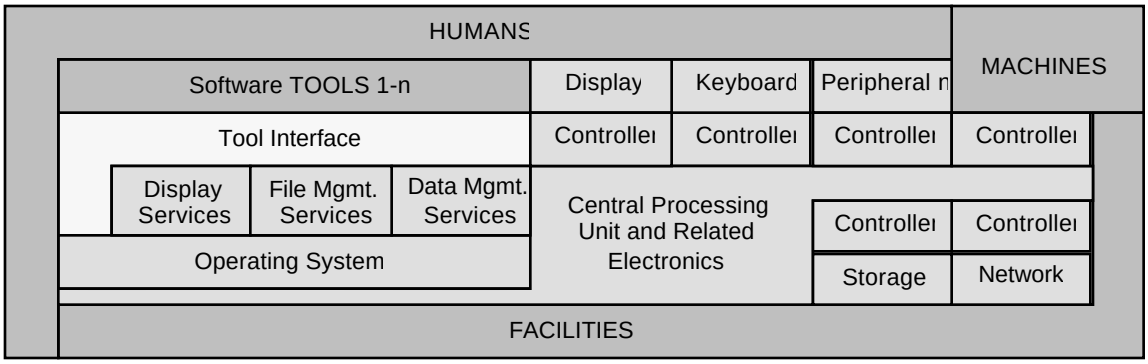
Another CALS requirement is the Standard Generalized Markup Language (SGML). It is to be supported by all tools that create documentation (text and two-dimensional graphics) that may need to be transferred. Until SGML is widely supported, other transfer standards may be used for text and graphic data as described in the Neutral Format section.

6.9.10.2.2.5. Common Database Management System

More data integration can be achieved by having the software tools use the same DBMS rather than a common file format or a common data schema. This approach requires extensive modification of existing tools. This approach is most appropriate to new suites of tools. This approach minimizes normal maintenance, but the cost of changing DBMSs is such that it is often not done when it should be for the tools to remain competitive.

6.9.10.3. Tool Integration Approaches

As noted previously, the computer is assuming more and more of the functions that had been developed for each application program (computer tool). As the degree of tool integration increases, higher-level data management and tool control services are being provided as part of the computing resource. These services are normalized across different brands of computers by tool interface environments that run in such a heterogeneous computing environment.



The Tool Interface layer may be implemented as a proprietary development standard within a company or as an international standard. It may have various degrees of sophistication.

6.9.10.3.1. Application Programming Interface (API)

All the tool interface methods described thus far exclude any means of tool control. SQL is an example of a relatively standard API, but it is limited to data manipulation. Most APIs include various high-level routines for data definition, display and tool control as well as manipulation.

APIs that include high-level, yet useful routines, are usually the most successful. If the APIs simplify the overall programming task, it is in the interest of the programmer to use them. More sophisticated tool integration efforts require more sophisticated APIs, like those that support *triggers* and *constraints*. For example, the invocation of a tool may be pre-defined to occur upon the termination of a predecessor tool in a process or upon the appearance of certain data or a change in its status.

Tool developers have tried to satisfy all the needs of their discipline oriented customers with ever larger tool sets amassed from a combination of proprietary and third-party tools. The most successful vendor was the one with the most complete set of integrated tools. Each tool developer devised a proprietary AIS to reduce the cost of tool integration and maintenance.

Eventually some tool providers realized they could not provide all of the functions their customers desired and stay apace of the technical evolution of the tools. Those with an established API made it available to third party tool developers. The high level routines available with the API reduced the tool development cost to the third party tool developers. They accelerated the expansion of integrated tools that could be marketed by the vendor with the API.

Mentor Graphics adopted this approach from its inception for its integrated electronic Computer Aided Engineering (CAE - design and analysis) environment. Atherton Technologies created the Atherton Tool Interface Standard (ATIS) as an API specifically for the Ada Computer-Aided Software Engineering (CASE) environment. It does software configuration management according to MIL-STD 2167A. Rockwell has promoted the integration of the Digital VAXset of software development tools into the Atherton framework.

Digital Equipment Corporation worked with Atherton Technologies to make Atherton's API more robust. The result of that work, A Tool Interface Standard (ATIS) is promoted by DEC as an international standard. Both have made their API's "open" to encourage their adoption as an international standard.

DEC offered ATIS to various standards bodies for acceptance as a standard for tool integration. The CASE Information Services (CIS) industry consortium was created to refine the ATIS standard. It was submitted to the X3H4 ANSI standards committee. The CIS represents 30 CASE user and computer companies. The ATIS was endorsed as the working document for the Information Resource Dictionary System (IRDS) by the members of the American National Standard X3.138 committee. The IRDS is supported by a single layer within the ATIS. The government (CALS) demand for IRDS is keeping the committee purposeful.

The CIS is considering the relationship of ATIS to various proprietary systems such as Mentor Graphic's Concurrent Design Environment (CDE). The CIS is also considering layering the ATIS on the Portable Common Tools Environment (PCTE) standard being developed by the European Computer Manufacturer's Association (ECMA) for the Unix environment.

Like other vendors, DEC is complementing the basic data access and control provided by its API with an Integrated Project Support Environment (IPSE). The IPSE provides even higher level data structure definition and manipulation facilities, including comprehensive version control and configuration management services. The IPSE concept and thrust are primarily directed at the CASE environment, but is applicable to the other application areas (electrical/electronic and structure/mechanical) as well.

Recently the API approach has appealed to the structural/mechanical disciplines as a way to integrate their tools. Efforts are underway to take advantage of the configuration management and version control facilities of the APIs and IPSEs developed for software, and apply them to the structural/mechanical tool integration problem (SDS of Matra DataVision).

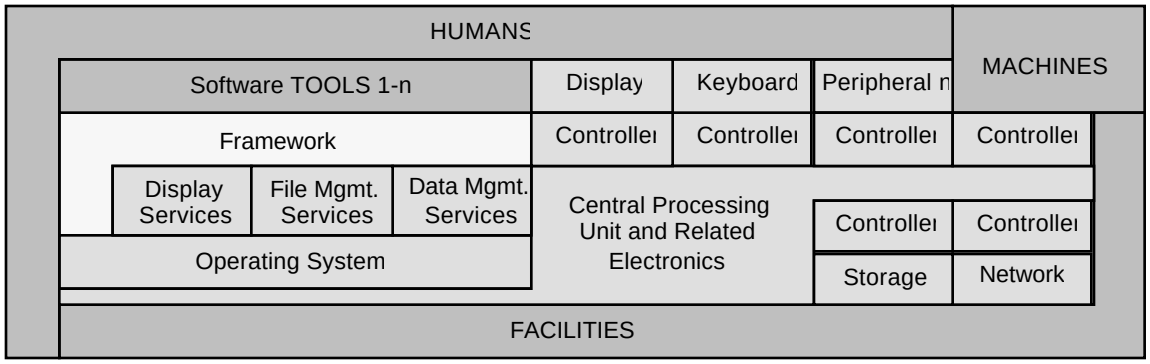
This is a favorable trend for many businesses whose products involve a combination of many system types (software, electrical, electronic, structural, mechanical, hydraulic, etc.). "Islands of automation" have grown around each discipline. They improve the performance of each business function, but demonstrate little net benefit, because the islands are not interfaced. The islands must be blended and their tools integrated.

If multiple vendors are to integrate their tools into a cooperative set for these businesses, they must agree on a standard API (ATIS, Revision 8 of Mentor Graphics). If they want their tools to be easier to learn and use, they must agree on a user interface convention ("Mac-like").

The success of the Macintosh environment has prompted many to examine tool functions to determine what else may be sufficiently common to warrant the development of a common support environment, one that goes beyond that of the Macintosh computer. Though perhaps less common than the Macintosh Open, New, Save, Save-as, Cut, Copy, Paste and Duplicate functions, there are, in fact, many significant functions that are common to many tools. These include data display, version control and configuration management. The degree of computing resource interaction required to support this degree of integration sophistication has prompted the emergence of *frameworks* or *backplanes*.

6.9.10.3.2. Frameworks

Electronic assemblies on printed circuit boards plug into computer backplanes so they can exchange data and control information through a standard bus. Frameworks allow software tools to be "plugged" into them to share data and coordinate their actions through a sophisticated API. The appeal of frameworks is that the tools can easily be unplugged and replaced with better tools as they become available. This not only makes the enterprise with a framework more adaptable to the changing needs of its marketplace, but also reduces the cost of implementing new tools. This is especially true if the framework encourages a consistent user interface.



A framework is a further expansion of the shared display and data management services described thus far. These services continue to evolve as a function of technology and standards.

The framework is based on conventional computing resources. At least one computer with an operating system, database management system, and input (keyboard, mouse), output (display) and storage devices (disk) is required to support a framework. Some frameworks can utilize multiple different database management systems as appropriate for the tool set in the framework. If more than one computer is involved, they must be able to communicate with one another via a data communications network. It is through the input and output devices that human resources interact with the framework and their tools. It is through communication adapters that machine controllers interact with the framework and the tools.

In order for tools and humans to access and use objects, knowledge of their location and characteristics must be maintained in a dictionary/directory. This dictionary/directory may be centralized on one node (computer) in the network or *shadow* copies of it may be distributed throughout the network. Portions of the dictionary/directory may be hierarchically distributed. The existence of an object may be known, but its component objects are only known to a portion of the dictionary/directory. That portion may be distributed to another node in the network where the data physically resides. The central dictionary/directory must query the node with the lower level information to respond to questions posed about it. Else it must copy the data to another node for complete access.

The relationships of objects to one another (parent, child, manifestation, derivative, originating tool, owner) must also be maintained if objects are to be used to find other objects. Knowledge of how the objects are individually stored must also be maintained if they are to be presented together with the proper view orientations.

Several commonly used services are provided by the framework for tools or human resources to use. For example, many tools may need to notify human resources via electronic mail or telephone of tasks started or completed and their results. Many kinds of data may need to be browsed graphically. To avoid having to maintain access criteria for each tool, a single access control service can be used and be maintained.

Common *core services* like solid modeling kernels, shading algorithms, graphical display routines and text editors can be made available as libraries of code for directly linking with an individual tool. For performance reasons, such core services will likely not be accessible by way of the standard framework access method (AIS). They will more likely be a collection of high-level functions that can be called upon by any tool to create or manipulate an object.

The Open Software Foundation (OSF) selected the following technologies to be among the core services of its Distributed Computing Environment (DCE):

- Distributed Name Service (DECdns) is a global directory system for computer networks is based on the Network Computing System from Hewlett-Packard. It stores information about objects in the network. It makes it possible for resources to be used without knowing their physical location in the network.
- Remote Procedure Call (DECrpc) is a programming tool for transparently distributing applications across a global network. It is based on the Network Computing System (NCS) from Hewlett-Packard. The DECrpc extends the NCS functionality in the areas of transport independence, name service independence, ISO standards support, and support for large data processing applications.
- Distributed Time Service (DECdts) is a fully distributed service from Digital Equipment Corporation (DEC). It provides precise, fault-tolerant clock synchronization for systems in local and wide area networks.
- Concert Multi-thread Architecture (CMA) is a set of services that support the development of efficient client-server parallel processing applications.

These services provide performance gains by enabling distributed applications to overlap network activity with other work. They contribute to the standardization of framework services.

Although the Design Automation Conference (DAC) is primarily concerned with electronic design, it provides a forum for vendors to display their framework products and promises. Recognizing the need for framework standards, attendees of the DAC formed the CAD Framework Initiative (CFI), which is now an industry consortium (IBM, DEC, HP, Mentor Graphics...). Mentor Graphics

participates in the Object Management Group and the CALS test network. The ATIS model has been presented to the CFI, but Mentor Graphics does not subscribe to it. The Mentor CASE environment is "totally different" from ATIS. Although the CFI is considered to be a good idea by all of its participants, there is no driving force like the government or a strong market demand like OSF to keep the consortium on a purposeful track. The proliferation of proprietary frameworks may be evidence of the disarray of the CFI.

There is no equivalent to ATIS for process control. The Concurrent Design Environment (CDE) of Mentor Graphics is equivalent and elements of EDL from Control Data Corporation. They provide a proprietary means of process definition and control.

6.9.10.3.2.1. Encapsulation

The less sophisticated forms of tool integration into frameworks are called *encapsulations*. Encapsulation is a means with which to "plug" a tool into a framework without significantly modifying the tool. The tool is enveloped in a *shell* or *husk* of interface software that makes it appear to the tool that it is still being invoked by a human resource and is still using its old data management scheme when in fact it is invoked by the framework and uses the data management facilities of the framework. An encapsulation can also simplify the user environment by hiding unnecessary functions and complication by saving user preferences, saving contexts and automating complex data flows.

If the encapsulation is limited to preparing data for the tool and invoking it, no modification to the tool is required. Even with such a crude encapsulation, a Process Manager (see Tool Resources) may restrict the use of the tool and control its data input. Without modifying the Save command of the tool, the process manager cannot be informed of what data was outputed from the tool. A Product Data Manager (see Tool Resources) will be unaware of its significance to the product definition. Without a modified Quit command, the process manager is dependent on the action of a human resource to inform it of the completion of a subtask. A human resource must perform the extra step of informing the process and product managers.

Encapsulated tools will likely have lower performance, because they will incur the overhead of the encapsulation software. To minimize this problem, encapsulated tools can perform data management in the framework at a higher data object/element level (file or record) than would integrated tools. Integrated tools perform data management at the low data object/element level (field). The higher level of data management may be insufficient for proper data integration among the tools in the framework. This is especially true among the tools whose data models have common elements or features. Furthermore, the encapsulation will have to change every time the tool or the framework changes.

Although encapsulation has its disadvantages, it has the advantage of time. An existing tool can be plugged into a framework more quickly as an

encapsulation. Consequently, existing tools will likely be encapsulated into the framework first, and later integrated.

This two-step integration process can be avoided with tools that are currently under development or are undergoing a major revision. Tool developers are most susceptible to the influence of framework suppliers and buyers. Tool developers do not want to be in the encapsulation business forever. Encapsulated tools are far more expensive and less rewarding than are fully integrated tools.

It is in the interest of framework suppliers and buyers to make the framework such a success that the tool developers are motivated to integrate with the framework. Tool integration services should be provided to make it as easy as possible for tool developers to integrate their tools into a framework.

Some tools will likely always be encapsulated. Special purpose tools will never have a sufficiently large market to justify the cost of integration. They and those which require special purpose computers will likely only be encapsulated at the tool invocation level. For example, framework utilities may be developed to extract data from the framework and format it specifically for a tool that runs only on a Cray computer. The framework would transmit the data to the Cray, invoke the tool on the Cray, collect the results, load them into the framework and resume integrated operations.

6.9.10.3.2.2. Exits

The next level of tool integration into frameworks involves the edition of *user exits* or the minor modification of existing commands like the Save command, so the Process Manager can be automatically notified of task termination and the Product Data Manager can be automatically notified of file or data activity. Then, both can accordingly update their metadata. With this degree of encapsulation, the Product Data Manager can control the output of a tool, and where the data is located. An example of the use of user exits is described in the beginning of the Tool Integration section.

6.9.10.3.2.3. Full Integration

To integrate a tool fully into a framework, extensive modification of the tool will likely be required. This will allow direct access to low granular data and their relationships. Control over some of the internal functions of the tool may be exerted.

Full integration requires that the tool developers replace their user interface with one which is a framework standard. If the framework standard is ODA/ODIF Motif or its precursor, X-windows, then this conversion effort is likely already underway or complete for the tool vendors who are aggressively trying to expand the market for their tools. They already support windowing standards, regardless of the operating system.

This is a major effort however, and may have significant performance implications. Many tool vendors are resisting complete integration and opting for what is essentially active metadata integration. In this case, the tool uses the framework to find files or large objects and informs the framework of the existence of new files and large objects, but it uses proprietary data management services to manipulate its small objects (solid primitives, surfaces, edges). For example, the existence of a part created using a solid modeling tool may be made known to the framework, but the constructive solid components of the part may only be known to the solid modeling tool.

In most cases data management above the level of features (arbitrary collections of boundary elements) will be adequate. The feature relationships among dissimilar tools will not need to be retained. For example, a printed circuit assembly may be mounted to a relatively rigid structure. Since the thermal expansion coefficient for the structure differs from that of the printed circuit board, thermal and structural analyses are performed to insure that this situation does not cause the board to warp and its circuits to crack as the printed circuit assembly warms. The hole features of the printed circuit board, defined as circles in an electrical CAD system, will be converted to a cylindrical boundary elements when the circuit board is converted to a solid model in a mechanical CAD system for packaging purposes. The hole feature is directly related to the threaded cylinder feature of the screw that goes through it, and indirectly by way of the screw to the threaded hole feature in the boss on the structure that supports the printed circuit board. The model is defined entirely in one tool before it is modeled for the analysis. The result of this analysis may be that the holes have to be larger and the screws tightened with less force to allow the board to move as it expands relative to its supporting structure. This knowledge can be communicated to the circuit board and structure designers so they may accordingly change their designs.

Eventually however, data at the level of features of parts will have to be maintained in the framework, or at least related within the framework if they are separately retained in the databases of the tools. For example, the hole feature of the aforementioned printed circuit board may be used as a ground or thermal sink. In this circumstance, it should be directly related to the cylinder/cap feature of the screw that goes through it and indirectly related by way of the screw to the threaded hole feature in the boss. The effect of the structure on the electrical and thermal dissipation of the printed circuit board can then be properly simulated. In this case, conductivity attributes would also have to be associated with these feature relationships.

The inconsistent availability of low level (feature) data will adversely impact the usefulness of shared data. Having the data of one tool available at the feature level is of little value if the data of a tool with which it must share data is only available at the part level. Tool integration efforts should be performed in a manner that promotes a consistent increase in the granularity of data among the various tools that may share it.

6.9.10.3.2.4. Advantages

The added cost of integrating a tool into a framework is offset by several factors. Firstly, someone is more likely to buy a tool if it is integrated with other tools. Secondly, a lot of software that would normally have to be developed, maintained and distributed with each new release of a tool need only be called from among the many framework services. Thirdly, if the API used by the framework is a standard used by other frameworks, then the market for the tool will expand to include that of all such frameworks and all of the computers on which they run.

What of the effort invested in porting tools to Unix? What about those customers who are not yet or may never be framework advocates?

Porting to Unix may not be in vain. It is not yet clear whether frameworks or APIs will ever completely shield tools from operating systems. Furthermore, some tools and their Unix computer may be treated as a unit (composite object) by a framework. The framework may package data and commands, send them to the computer/tool, retrieve the results and unpackage them for use by other tools in the framework. This would be especially true of batch programs.

The push for Unix may also not be in vain. Unix is important to cost-effective framework portability.

Tool vendors will have the best of both the framework and non-framework worlds if the framework suppliers make their frameworks portable and license their framework to the tool vendors. Then the tool vendors can continue to sell their tools to non-framework customers as if the tool were still independent of a framework when it is in fact fully integrated into a framework.

6.9.10.3.2.5. Applied to Data Transfer To/From Subcontractors

Ideally, the computing resources of subcontractors should be integrated with the computing resources of their prime contractor (system integrator) to the extent that their contributions to the process cannot be distinguished from any other function of the enterprise. They should interact directly with the framework of the prime contractor.

Text, binary and bit-mapped (facsimile) files would be exchanged via electronic mail. The data would include Request for Quote (REQ), Response to RFQ, Purchase Order, Purchase Order Acknowledgement, Order Status, Shipping Notice and Request For Assistance (RFA) information. If the contracted work entailed design, then the subcontractor would have access to the digital models that constrain the design. If the contracted work entailed manufacturing, then the subcontractor would have access to the manufacturing plans and the digital models from which end-effector motions could be defined.

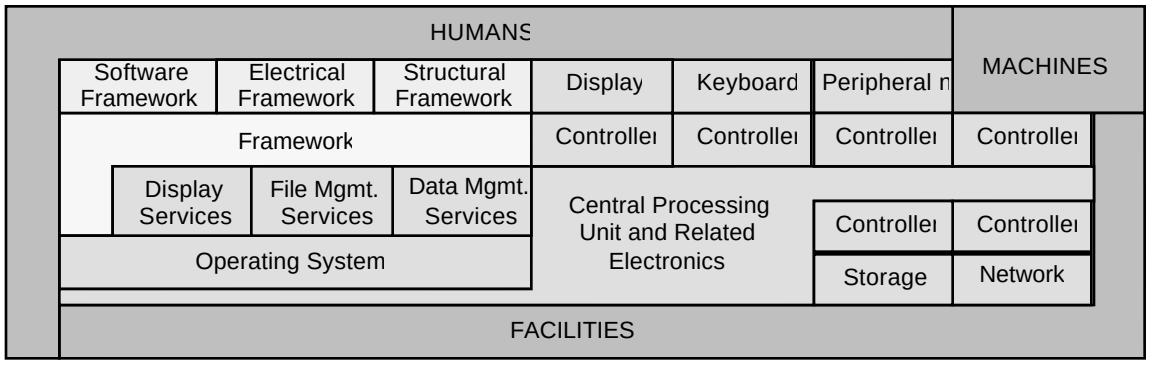
6.9.10.3.2.6. Framework Wars

It was hoped that the operating system war could be ended and some application program (tool) mobility among computers could be attained by forcing the computer suppliers to support Unix as a standard operating system. Then the better price/performance computers could be purchased for the better tools with little regard for computer/tool compatibility. As that goal was being attained, computer vendors reasoned that with Unix and the advent of POSIX, they could no longer discriminate themselves from one another with their proprietary computer operating systems. Computer system vendors realize that software sells computers and integrated tools sell more computers. As evidenced by the success of Macintosh, tools integrated with no more than similar Save, Open, Close and Cut and Paste functions (similar "look and feel") have a favorable impact on computer sales.

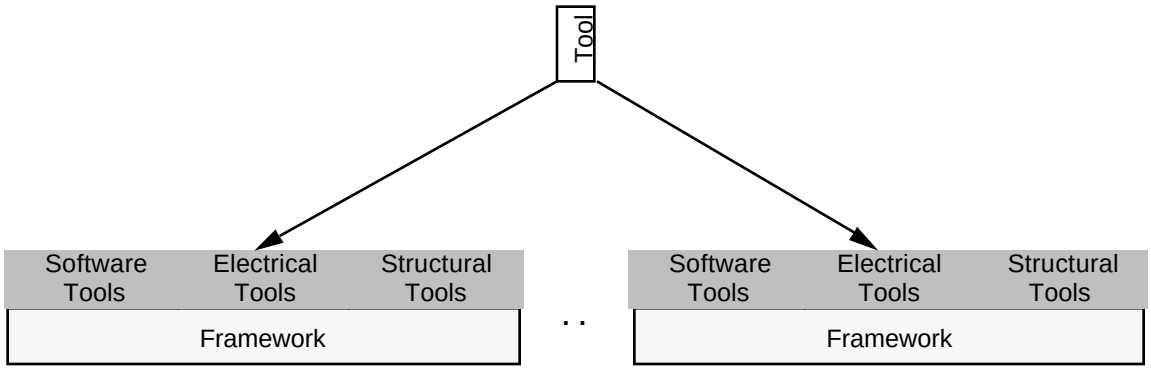
For a framework to be successful, it must be widely accepted by tool developers. To be widely accepted by tool developers, the framework must provide all of the desired functions and performance. It must be easy to encapsulate into or integrate tools with the framework. The framework should not limit the market of the tools. It should expand the market for them.

6.9.10.3.2.6.1. Framework Buyer Dilemma

The framework buyer who considers the structural design tools of Matra DataVision, the electrical design tools of Mentor Graphics and the software design tools of Software Through Pictures as the "best of the breed" must choose less than the best tools, or interface frameworks like Software Through Pictures, Mentor Graphics and Euclid into a third, enterprise-wide framework.

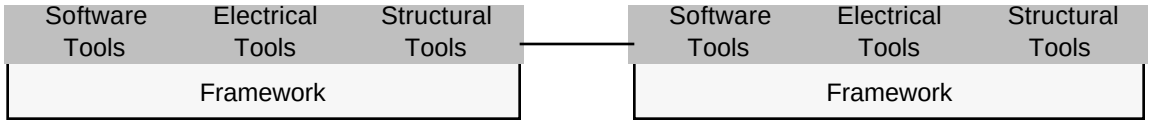


One way to avoid this dilemma is to make the ATIS a true international standard. It would leverage the efforts of tool developers to encapsulate or integrate their tools into any framework that complied with the standard. It would broaden the market for the products of the developers who hosted their tools on an ATIS framework without correspondingly increasing their costs. This would cause other developers to abandon non-ATIS frameworks for those that employed the ATIS, which would in-turn force the suppliers of non-ATIS compliant frameworks to make them ATIS compliant.



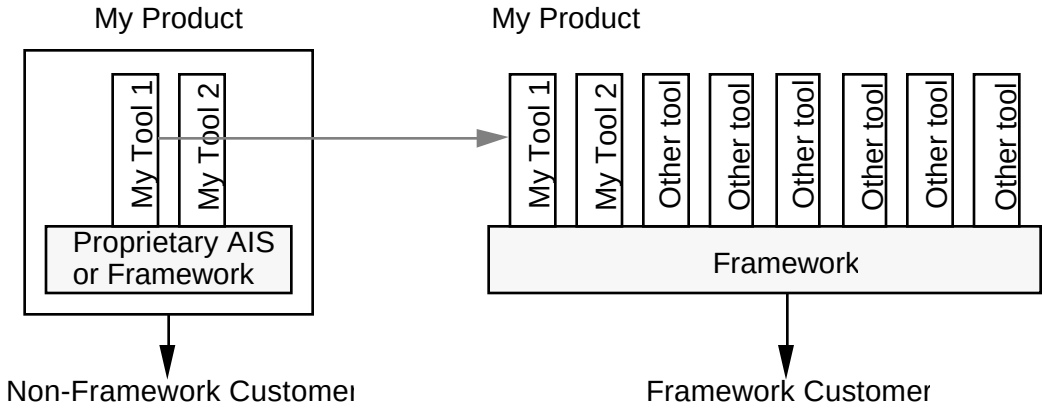
It is in the interest of the framework supplier and buyer to promote to standards committees and other computer vendor participants the proposed standard on which the framework is based. It is especially in the interest of the framework supplier and buyer to promote the proposed standard to the tool developers, not only to insure that it is chosen as a standard, but also to insure that the standard will, in fact, be used by the tool developers.

Until ATIS is a standard, the tool buyer who insists on having the better tools must encapsulate one framework into another or bridge frameworks by way of interface tools in each framework that communicate and translate between the frameworks.

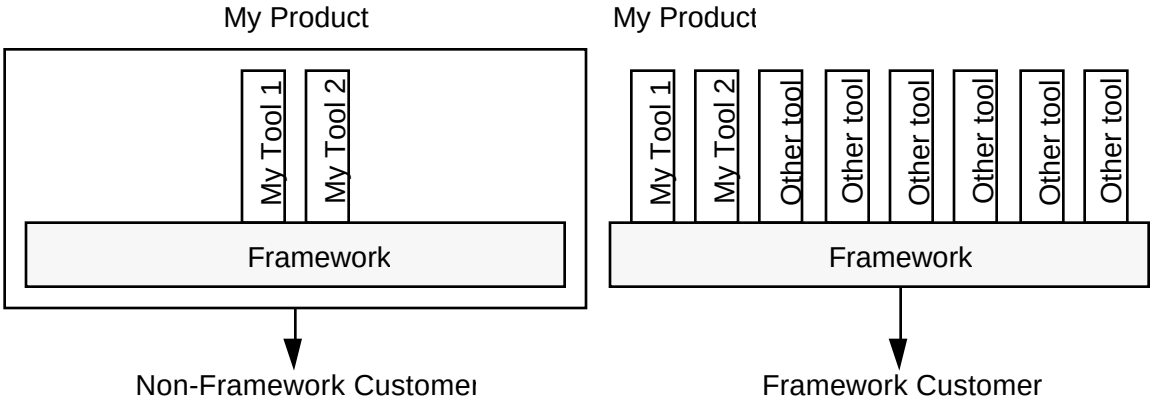


6.9.10.3.2.6.2. Framework Tool Developer Dilemma

If the tool developer wants to continue to serve the traditional non-framework market and expand into the framework market, the cost may be prohibitive. At least two products or versions must be maintained.



To avoid this problem, the framework should run on a broad number of computers, and the framework should be licensed for sale by the tool vendors.

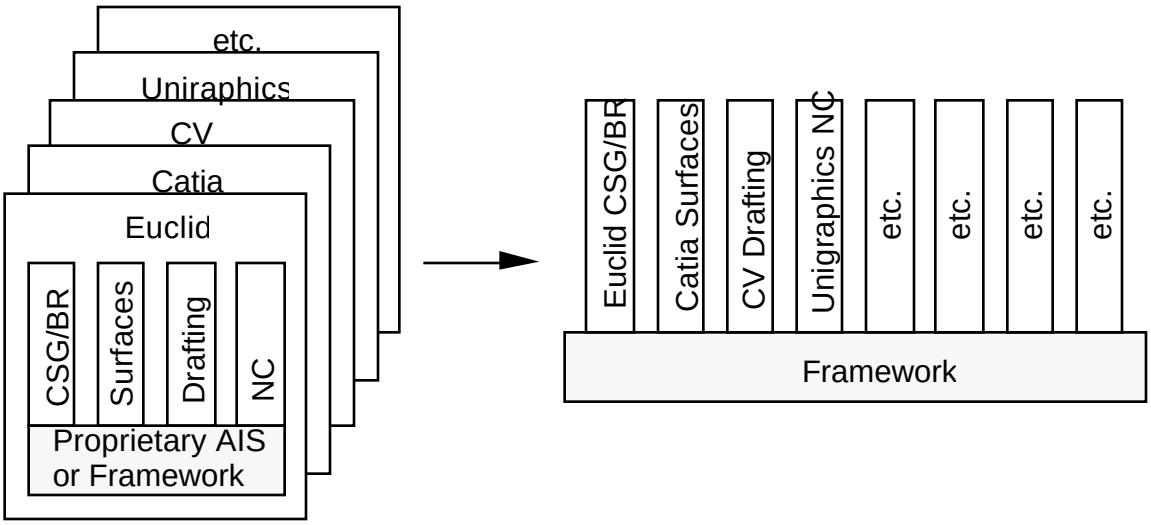


This would eliminate the need for tool developers to maintain their tools both in a framework and as a self-contained product. The framework with just their tools could be sold by the tool vendor as a self-contained product when it, in fact, is bundled within a framework. Obviously, the single vendor tool set framework license should be restricted to the tools of the developer. It should cost less than the license for a multi-vendor tool set implementation. Having the framework run on any Unix computer and licensing it to tool developers would not only minimize the risk to the tool developers of encapsulating into or integrating with a framework, but it would also sell a lot of frameworks. It would distribute the development cost over many tool and framework customers, which would allow the framework price to be reduced, which would attract more developers and buyers to the framework.

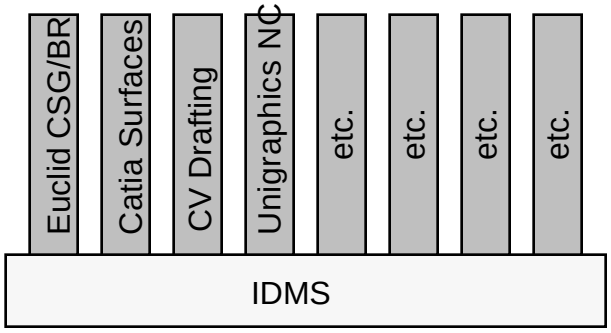
To further encourage tool developers to encapsulate and eventually integrate their tools into a framework, the framework vendor should provide tool encapsulation and integration services to tool vendors.

6.9.10.3.2.7. Tool Decomposition

Tools that are really a collection of tools bundled as one product with various options, like Euclid, IDEAS and All-In-One should be unbundled. Each useful piece thereof should be separately encapsulated or integrated into the framework. This finer degree of tool granularity would allow framework tool buyers to choose the optimum combination of tools for their environment. They would not have to live with bad portions of a bundled tool to have its good portions. They would not have to buy tool sets which have tools that are redundant with those of other tool sets (e.g., drafting, shaded display, text editing tools). They would only need to buy the best solid modeling kernel, shading or drafting tools, regardless of their source.



Though mitigated with the implementation of Motif, the user interface is still different for each tool. This necessitates training for each tool and retraining for each new release of each tool. The next step toward complete integration is the decomposition of tools into their constituent algorithms and their execution under a single user interface.



Such a decomposition will effectively result in many more "tools". Most will have no user interface or data management facilities to call their own. A technology known as Hyperknowledge™ can effectively integrate all the algorithms into what appears to a user to be a single tool with a single user interface. The user interface is oriented toward requirements definition and feedback to the user concerning the design, analysis and manufacturing preparation effect of the requirements as interpolated or extrapolated by Hyperknowledge™. This form of integration is exemplified in the Intelligent Design and Manufacturing Synthesis (IDMS) system of OMNIGON, Inc. (P.O. Box 96970, San Diego, California 92196-9070).

6.9.11. Programming Languages

Much of what is in this section was learned from Digital Equipment Corporation (DEC).

Object-oriented programming is the method of choice. C++ is the *de facto* object-oriented programming language. Other dominant programming languages are COBOL for business applications and FORTRAN and C for scientific and graphic applications. To minimize software maintenance costs and maximize the reusability of code modules (objects or subroutines), structured programming techniques can be applied to the use of any programming language.

6.9.11.1. Graphics Programming

The Programmers' Hierarchical Interactive Graphic Standard (PHIGS) and the extended standard for 3-D graphics (PHIGS+) are graphics programming standards. The PHIGS standard embodies structure concepts, editing and search capabilities. It provides structure network posting and a consistent approach to structuring graphics elements for display, particularly attributes and transformations. It defines uniform methods of handling output primitives, input classes, and attributes used in transfers between applications and between applications and devices. It includes primitives to support wire frame graphics. It defines a fill area set primitive and an annotation text primitive to support the requirements of 3-D graphics applications. It defines polyline, polymarker, text, and generalized drawing output primitives. It also supports atomic output primitive functions at the device level, such as circles, rectangles, and arcs. It provides a model of the abstract graphical workstation.

PHIGS+ provides a transformation pipeline, handles clipping and transformations and stores data in a defined coordinate system for perspective and parallel views.

The PEX committee standard (PHIGS/PHIGS+Extension to the core X Window System) will provide a common programming interface for 3-D graphics for both Graphics User Interface functions in the X environment and for graphics applications.

6.9.11.2. Object-Oriented Programming

Much of what is in this section was learned from Digital Equipment Corporation (DEC).

Programs traditionally work on data structures like record and array, and operations like assignment and expression evaluation. Traditional programming has forced the programmer to think in terms of low-level abstractions that are close to the level of the hardware on which the program executes. The advantage of an object-oriented view of programming (or data management) is that it offers the programmer a higher level of abstraction from which to work. This simplifies the job of programming.

This is the key reason why the word "object" is becoming such a buzz word. Ten years ago it was argued whether programmers should work at the level of abstraction supported by assembly languages (named variables and mnemonic representations for machine instructions), or at the level of abstraction supported by high level languages (COBOL, FORTRAN, ALGOL, PASCAL, C, etc.). The argument was settled decisively by the fact that programs written in high level languages took nearly one-third the time to develop as their counterparts in assembly language. Subsystems written in high level languages often had a life-cycle maintenance cost one-fifth that of subsystems written in assembly language. Object-oriented programming promises to provide a programming improvement of the same magnitude.

This 5:1 increase in productivity, and a 10:1 reduction in life-cycle support costs results from

- a high degree of modularity, enforced by the software,
- the ability to inherit and refine types, letting programmers reuse modules instead of writing from scratch and
- a high level of abstraction which lets programmers work at the level of their problems rather than the details of a tool.

Computers and programming languages have always had rudimentary notions of object types in them. Most have built-in support for a small number of low-level types: CHAR, INT, FLOAT. These were called "data types." They are also "types" in the more general sense of the word. They have instances ('3' is an instance of the type INT). They each support a well-defined set of type-specific operations.

Object-based programming extends this notion of types and instances to higher-level types. The programmer is no longer restricted to working with pre-defined types. Types can be defined such that their behavior and properties closely resemble the real-world entities they are intended to model.

Object-oriented ideas have been emerging with different terminology in three subdisciplines of programming:

- database management,
- artificial intelligence and
- programming languages.

In the database world, object-oriented systems are called "semantic data models." Examples are TAXIS (Univ. of Toronto), SDM (MIT), DAPLEX (CCA) and GEM (Servo-Logic). In the artificial intelligence world they are called semantic nets, or knowledge representation systems. Examples are FRL (MIT), KRL (Stanford) and SRL (CMU). In the programming language community, support for object-based languages has begun to appear both in applicative languages like LISP (the FLAVORS mechanism at Symbolics; GLISP at Stanford) and Smalltalk, as well as more traditional algebraic languages (Ada, C, Modula).

Conventional data models are not expressive enough. They provide no integration of data and procedural information. They perform poorly for *graph-walking* (pointer chasing) applications (CAD, CIM). Their transaction and recovery mechanisms (locking and logging) have too much overhead. One model is needed for data and programs. The object-oriented paradigm provides that model.

The attributes of an object-oriented programming environment listed in order of importance are methodology, language, programming environment, type library and persistent storage (database). The most important aspects of object-oriented programming are *data abstraction* and *inheritance* . Data abstraction allows behavior to be separate from implementation. It supports abstract data types. For example, *shape* is an abstraction of

Case 1 - circle
Case 2 - triangle
Case 3 - square

Abstraction provides a higher-level interface.

Objects are well-defined data structures that consist of fields that refer to other objects and a set of operations that allow data to be manipulated. The object data model contains properties and operations. Properties model the state of an object. Operations define its behavior. Objects are self-identifying. Objects invoke *operations* of other objects. An operation has two parts:

interface (message) and
implementation (method).

Processes (text scanner) and intangible concepts (window manager) can be modeled as objects.

Types classify objects. The chief role of a type is to define the properties and operations (behavior) of its instances. Each object is an instance of a *type*, Type is a unit of modularity, not procedures.

Types are related to one another in subtype/supertype hierarchies. A subtype "inherits" all the operations and properties defined on its supertype. Type hierarchies are useful. For example, type *person* may have subtypes

employee, student, and
part-time, full-time.

Types should hide information and reduce interdependencies. A challenge is to choose where to put behavior: in a new type or in some existing type? Some rules follow.

Reusability - Factor-out behavior that would be useful in more than one context.
Complexity - Provide complex operations in a separate type.
Applicability - Provide behavior in a relevant type.
Implementation knowledge - Implement an operation that needs to know about the internals of a type in that type.

Inheritance allows data types to share definitions. To use inheritance effectively, one should subtype for

Specialization
type: input device
subtype: mouse, tablet, keyboard;
Implementation
type: dictionary
subtype: hash table, property list and
Combination
type: teacher, student
subtype: teaching assistant.

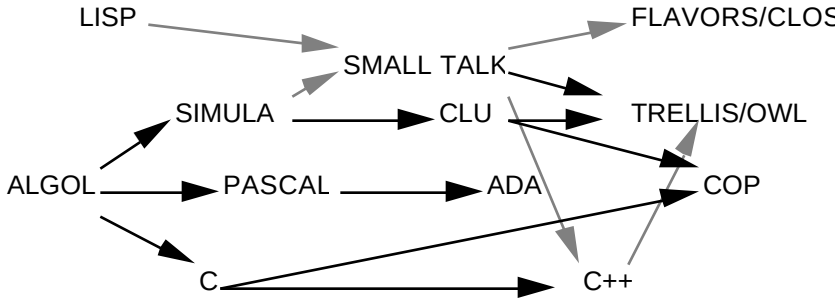
Subtype should not be used for aggregation.

Only Trellis/Owl, C++ and Smalltalk are truly object-oriented. Object C is an extension/hybrid of C. When selecting an object-oriented programming language, one should look for

compile time type checking,
parameterized types,
iterators,
exception handling and
storage management.

6.9.11.2.1. Evolution of Object-Oriented Programming Languages

Like most programming languages, object-oriented programming languages have evolved.



6.9.11.2.1.1. C++

C++ was developed by one person at Bell Labs. It is "C with class." It is a super-set of C. It makes concessions to C syntax, has strong typing and will have multiple inheritance. It has no parameterized types or exception handling or automatic storage management (no garbage management), but it has *constructors/destructors*. Large public domain object libraries are under development. Language development is controlled by one person. C++ is precompiled into C. The precompilation is not human readable. C++ compilation results in an efficient implementation.

6.9.11.2.1.2. Object C

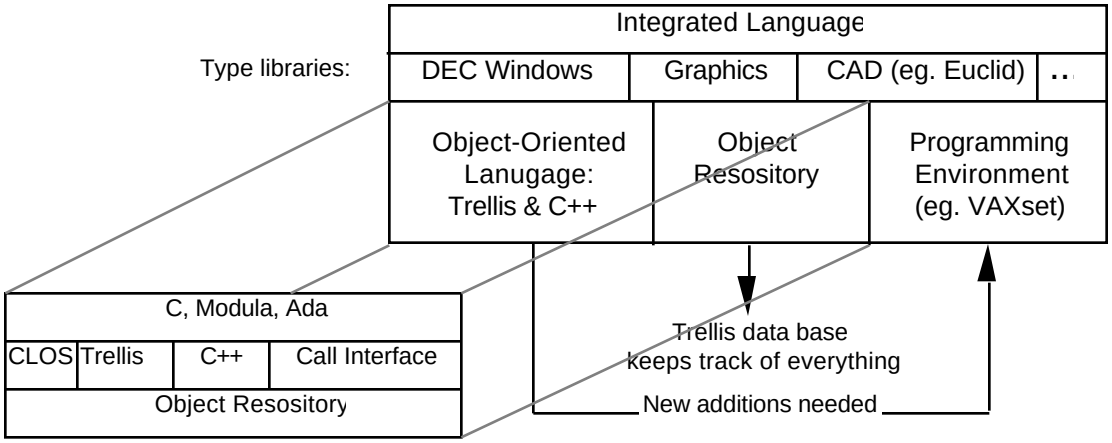
Object C from Stepstone is a melange of Smalltalk and C. It has a good library and a single source (unlike C and Smalltalk).

6.9.11.2.1.3. Smalltalk-80

Smalltalk-80 from Xerox is a mature language and system. It is primarily for experimental programming. It has an extensive library, a sophisticated programming environment, and a unique, but simple syntax. Object-oriented concepts permeate Smalltalk and are *designed-in*. Variables and slots are not typed, they are *checked-out* at run-time. There are many unofficial variants of Smalltalk. A standards committee has been formed.

6.9.11.2.1.4. Trellis/Owl

Trellis/Owl is the proprietary object-oriented programming language of DEC. It consists of over 30,000 lines of code. It has the message passing, late binding and inheritance of Smalltalk. It is strongly typed. It was inspired by CLU. It has ALOGL syntax, iterators, parameterized types, exception handling, automatic garbage collection, programming environment, call-out to any language (e.g., Fortran) and compiles into machine code.



Trellis/Owl is beginning to be accepted by development groups at DEC. One hundred copies were distributed for "research" purposes, but recently DEC has

decided to productize Trellis/Owl. This coincides with the selection of Objectivity as the object repository (see Object-Oriented Data Management).

Although Trellis/Owl could be used very productively independent of the VAXtools, they are still worthwhile additions to a Trellis/Owl programming environment.

6.9.11.2.1.5. EIFFEL

EIFFEL is like Trellis/Owl, but has a less sophisticated programming environment.

6.9.11.2.1.6. FLAVORS and CLOS

FLAVORS and CLOS (Common LISPS) are extensions to LISP. They have some compromises and non-uniformity. FLAVORS has several dialects. CLOS is being standardized. It has multiple inheritance with priority scheme for resolving inheritance conflicts. It does run-time checking.

6.9.11.2.1.7. ADA

ADA is not an object-oriented programming language. It may allow abstract data types.

6.10. Tool Resources

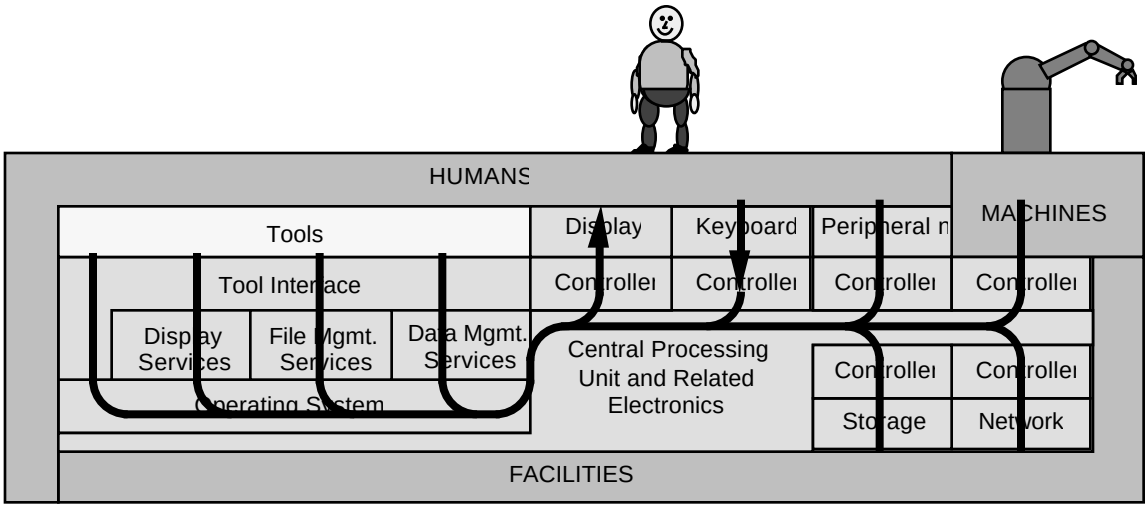
As the granularity of the product definition increases, the product definition differentiates into the major system categories of software, electrical/electronic and structural/mechanical products. This differentiation is a consequence of specialization and the practical need to emphasize different portions of the process. The isolation of the various disciplines (aerodynamics, structures, fuel systems, etc.) from one another resulted in differing vocabularies. Different names were used to describe essentially the same activities in the process. Slightly different ways of performing configuration management and measuring and reporting progress evolved. Accordingly, different tools were developed and used for common as well as dissimilar process practices. Efforts to integrate tools concentrated in one discipline to the exclusion of the others. However, there is a strong dependency among the software, electrical and structural systems of a product.

The cost to design software is high relative to the cost to manufacture and supply it. The cost to make an electronic system and test it is low relative to the cost to analyze it with comparable confidence. The opposite is more often true of structural/mechanical products. The cost of designing structural/mechanical parts is low relative to the cost of their manufacture.

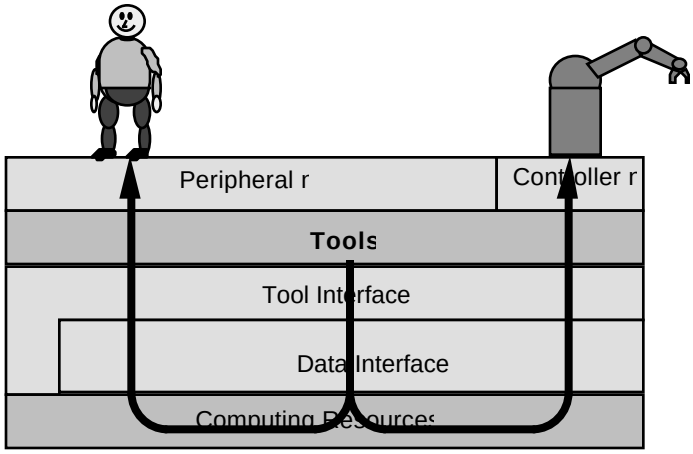
As a consequence, software tools tended toward requirements gathering, prototyping and interactive graphic design. They initially ignored configuration management needs, but later evolved the worst case requirements for configuration management. Electrical tools tended to concentrate on two-dimensional system design documentation (drafting) tools. As the designs became more complex, the tools tended toward system simulation and packaging (automated placement and routing) tools. Structural/mechanical tools initially tended toward analysis and two-dimensional assembly documentation. As the algorithms became available, the tools tended toward three-dimensional and parametric system design and analysis. These tools and their evolution are described in later sections.

Efforts to integrate the tools were naturally organized along the same lines of demarcation. Application interface "standards", *backplanes* or *frameworks*. were initially used to integrate Computer Aided Software Engineering (CASE) tools. Later, they were proposed as a way to integrate all tools. Businesses that involve a combination of many system types (software, electrical, electronic, structural, mechanical, hydraulic, etc.) must have their tools integrated across the disciplines.

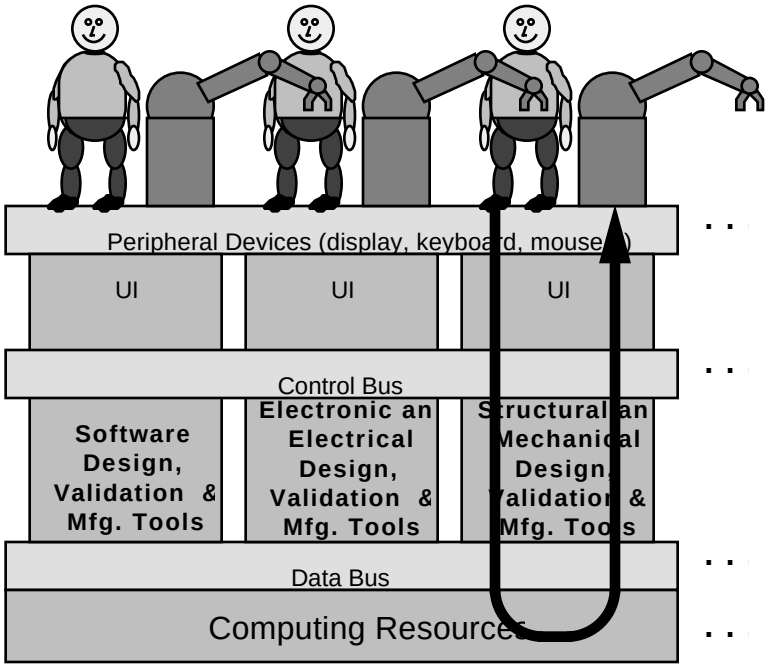
The interaction among the various resources, including the tool resources was depicted in the Computing Resources section with the following diagram. In this instance, the flow of data and control is shown. Most flows are bi-directional as indicated by the lack of any arrows.



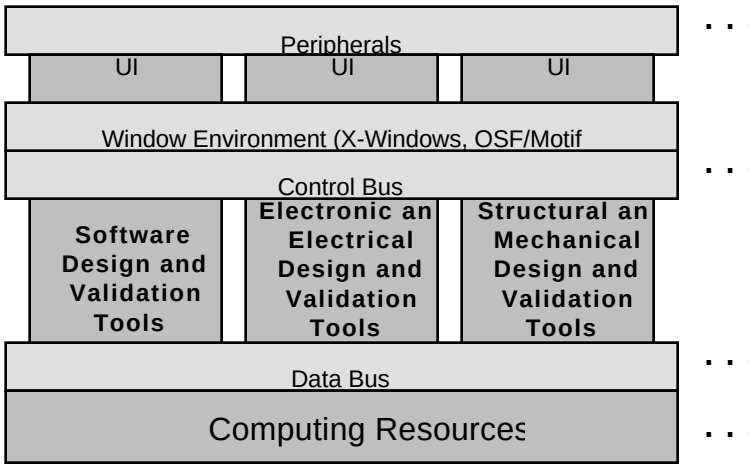
If the controllers are generalized and the Human, Machine and Facility resource categories are removed, a diagram more pertinent to tools results. In this case, only the flow of information to human resources and the flow of control data to machine resources is shown.



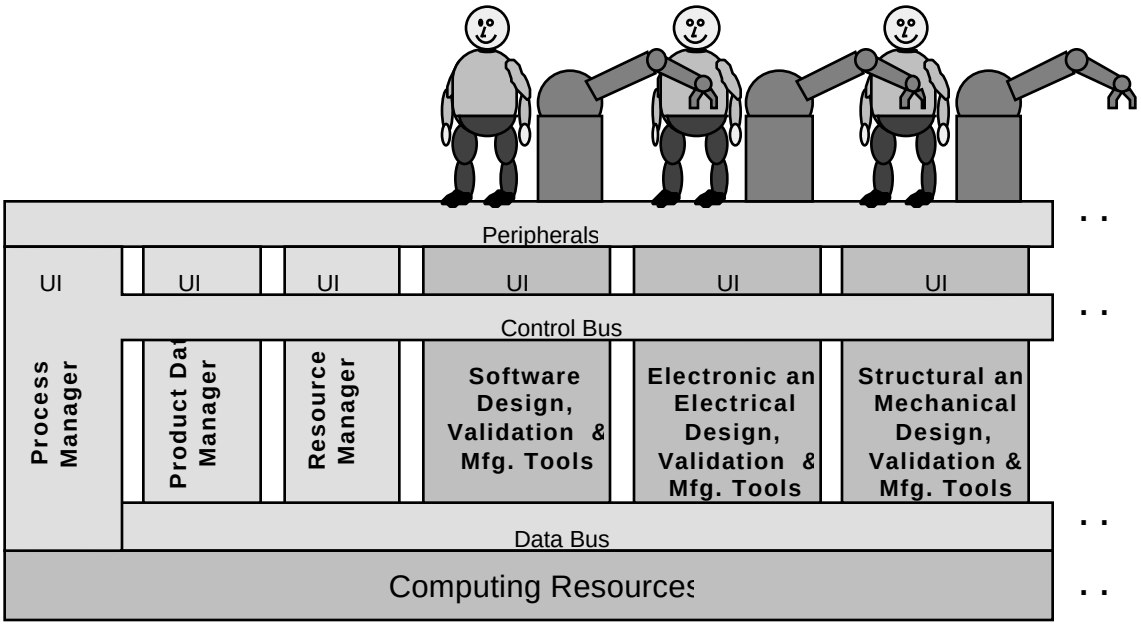
The following diagram emphasizes the tool categories (sets, product lines) that have evolved. What appears to be a differentiation of the business process into software, electrical and structural processes, each with its own requirements definition and allocation, system design and validation, manufacturing and support activities is, in reality, only a differentiation of the tools used in the process. The tools may change as a function of the product, but the fundamental process remains the same.



The flow of control and data is from the user through some peripheral device (or controller in the case of a machine tool), through a tool-specific user interface to the computing resources and back. As described in the Computing Resources section, a standard windowing system can be used by tool vendors to reduce the amount of the user interface that has to be developed and maintained. Sharing data through a similarly standard programming interface like the A Tool Interface Standard (ATIS) will also reduce software development costs. Having a mechanism for tool control will improve tool integration.



Three tools that are not common to most tool sets or framework depictions are Process, Product Data and Resource Management tools. The Process Manager improves the efficiency and concurrency of the use of the other tools. The Product Data Manager provides a common context for all the tool, machine and human resources. The Resource Manager provides a means to find resource combinations for specific types of work. It avoids resource over- or under-utilization and identifies resource conflicts. Each is described in more detail later.



Although much of this discussion has focused on software tools, the resource management concepts are similar in other tool categories. Tool cribs allow machine tools like drills, cutters and fixtures to be shared among various machines. A common database or application programming interface are similarly used to facilitate the sharing of data among software tools. Just as data may be distributed, the tools need not be maintained in one geographic location to achieve the goal of the tool crib. Tool availability and whereabouts only need to be known to those who need the skills embodied in a tool.

To conduct the tasks delineated in the Task Breakdown Structure (Information Integration section), a variety of skills are required. Since no one tool has all of the skills needed to conduct the business process, many tools are required. Descriptions of some of these tools follow. Most are described in terms of their use in the business process.

6.10.1. Hand Tools

Hand tools cannot be used by a machine. Hammers, files, saws, drill motors, "skill" saws, oxygen/acetylene welders are examples of hand tools. They each have particular skills. They may be used by more than one human resource.

6.10.2. Machine Tools

Machine tools are those tools which can normally only be used by a machine. Machine tools are held by the *end-effector(s)* of a machine. Tools are an extension of the end-effectors of machines. Machine tools that may be clutched by end-effectors are:

- Cutter Assemblies (cutter plus extensions and adapters)
- Cutters
  - Horizontal mill
  - Vertical mill
  - Drill
  - Ream
- Routers
- Carbon Electrodes (erosion milling)
- Graspers
- Welders
- Riveters
- Touch Sensors (measurement, assembly, disassembly, positioning)
- Water Jet (cutting)
- Lasers (cutting, steriolithography, measurement)
- Sound generators (autoconsolidation)
- Paint Dispensers

A single machine may have many end-effectors and cooperatively use many machine tools. The end-effectors of many machines in a cell may cooperate as well.

End-effectors are often not the only part of a machine that is capable of motion. The table or bed on which a workpiece is secured may be capable of moving in one or more axes (left/right, in/out, up/down, rotate around any of those axes) to present a new part or a different face of a part to an end-effector and its tool. In some machines, two tables alternate to present a new part to the tool(s) while the finished part is safely removed. A new workpiece is secured on the inactive table. Bed motion instructions must coordinate with those of related end-effectors.

A lathe is an example of a machine in which the bed, or *chuck*, in this case, does most of the work. The end-effector needs to move only in two axes (left/right and in/out) to present its simple cutting tool to cut a workpiece held by the rotating chuck.

In addition to the major motion devices there are coolant dispensers, chip removal conveyers, vacuums and other devices on a machine whose actions must be coordinated with the other devices on the machine by way of machine instructions.

Machine instructions must also instruct a human resource to perform tasks for the machine or ask the human resource to intervene if unanticipated circumstances arise with which the machine instructions cannot cope.

6.10.3. Fixtures

General purpose fixtures like clamps and vices may be used on many different machines and by human resources for many different parts. They have many skills. Special purpose fixtures are made for a specific part during one or more operations on it using one or more machines. They may have fewer skills, but fewer of them are needed to perform a task that would otherwise require many clamps and a lot of human resource time to position and secure them manually.

6.10.4. Software

Application software (software tools) are used to aid or automate portions of the business processes. Unless there is an intervening layer of software (ATIS) between a software tool and an operating system, the tool will only run on the platforms and operating systems for which it was designed.

6.10.4.1. Acquisition

The cost of "competitive edge" software is high if it cannot be shared. The cost of its maintenance is such that little, if any, is performed on "in-house" tools. As a consequence, what was once an advanced product quickly becomes obsolete. It is eventually replaced by a commercial software tool.

It is advisable to purchase all software tools rather than develop and maintain them, because the cost of development and maintenance of commercial software tools is distributed among many hundreds of customers. The maintenance of the product is paid by way of software license agreements by the many owners of the licenses or with a portion of the sales price. As long as the product continues to attract customers, there will be income for the continued improvement of the product, which should attract more customers.

Every effort must be made to locate, acquire, integrate and use adequate commercial software tools before consideration is given to developing proprietary software tools. If no acceptable software tool exists, then the software tool suppliers should be pressed to divulge their plans. If a new product is expected to meet the requirements as much as a year beyond the expected date of release of a software tool developed by the enterprise, the development should not be undertaken.

Only when no commercial alternative can be foreseen, and no commercial software supplier will undertake the development of a necessary software tool, should software be developed by an enterprise. Even then, the development should be undertaken jointly with one or more commercial suppliers with the understanding that the resulting tool will be marketed, sold and maintained by the commercial supplier. Royalties or advance usage can compensate the enterprise for its investment. If the enterprise develops software entirely on its own, it should seek a commercial outlet for the software, so that the software will be maintained as a viable product.

Large companies should strive to obtain site licenses for purchased software instead of licenses that are specific to an instance of the software on a computer.

Better yet, buy file server-based software on a concurrent-user basis. For example, a "four-seat" license would allow anyone anywhere on a network to use the tool until there are four concurrent users. The software restricts a fifth user from using it until one of the earlier four users "release" the software. If contention becomes a problem, buy more seats. Sell seats when they are in excess.

This arrangement allows an enterprise to increase and decrease its software tool inventory to meet its real needs at minimum cost and do it quickly. The cost-effectiveness and convenience of this approach encourages companies to comply with copyright restrictions, which is in the interest of the supplier. The addition or deletion of concurrent users could be done with code supplied with the tool, by the tool supplier on-site, or by the tool supplier remotely by modem.

Another advantage to file server based software is that it reduces the number of computers that must be upgraded with new releases of tool software. A tool on the computers in a network can be easily upgraded in one operation by a human resource, which prevents tool, network and peripheral (laser printer) disruptions due to version incompatibilities.

6.10.4.2. Development

If the enterprise chooses to develop business or product software, the most modern and highest level (generation) of software development tools available should be used. Fourth generation languages are being challenged by graphical (fifth generation) languages like those provided by Odesta and Symantec, especially for rapid prototyping purposes.

Prototyping software tools as a means to determine their requirements is encouraged. It elicits requirements that may otherwise remain unknown until late in the development process, when the cost to change is much higher. Prototyping is especially useful when the expected tool users are not computer literate. Their exercise of a prototype will help them learn what is possible. Then they can define optimal and realistic requirements. A prototype unambiguously represents the requirements of a proposed tool.

Computer Aided Software Engineering (CASE) tools like Software Through Pictures (STP) are usually software tool sets that are utilized to develop other software (instructions compiled and run on general purpose computers), or firmware (loaded and run in an integrated circuit), according to customer requirements. The customer requirements can be thoroughly defined using a tool like RDD-100 from Ascent Logic. CASE tools usually include graphic software design, code templates, logic verification aids, configuration management and documentation tools.

The rapid demand for CASE tools spawned the development of many tools by many small companies. None of them supported the entire software development process. Even when laboriously used as a set of tools, many functions were missing or redundant. Their proprietary command and data structures prohibited their integration. Application interface "standards" were devised to solve the integration problem. This was the genesis of frameworks, which are described in the Computing Resources section.

6.10.4.3. Data Management

No software tool should use a proprietary data manager. Data management related development tools should be limited to those that facilitate the use of a standard application programming interface, like A Tool Interface Standard (ATIS). The ATIS will provide concurrent access to one or more database management system as described in the Computing Resources section.

6.10.4.4. Migration Aids

Software tools to help the migration of data from tools based on proprietary or non-framework database management systems (DBMSs) to framework-based DBMSs are recommended to ease the transition from legacy software tools to a framework environment. Should the DBMS that initially underlies a framework be relational, software tools to help the migration from the relational DBMS to an object-oriented DBMS in the framework should be required of the vendor to ease that transition.

6.10.4.5. Class Definition

Class definition software tools allow a developer to complement, create or modify a hierarchy of objects, and complement, define or modify their methods (behavior) and their implementation (structure of the data and fields).

6.10.4.6. Mathematical

Software tools should utilize standard mathematic function libraries. Such functions include scalar product, vector product, matrix operations and trigonometric functions (sine, cosine, tangent, etc.). Common geometric functions like distance (point/point, point/line, point/surface, etc.), length (line, curve), area, centroid, inertias and intersections (line/line) should also be used.

Having software tools use the same functions helps to assure that the same operation performed by different tools has the same result. Such functions are an example of the direction software tools can and should take by way of a framework. Many of the products that may be encapsulated in a framework will include tool sets (drafting, surface modeling, routing) that are redundant with those of other products. Each of those tools will have redundant functions (display wire frame model, translate, rotate, insert dimension). The tools and their functions should eventually be decomposed and treated as mathematical functional libraries are currently treated.

6.10.4.7. General Use

These software tools provide functions that are commonly used by everyone involved in the business process.

6.10.4.7.1. Word Processor

Word processors like MacWrite, Microsoft Word and Word Perfect have become indispensable for quality written communications. They reduce the cost of changing text or managing large documents.

Word has a feature called QuickSwitch which provides a degree of automated document configuration management. Excel spreadsheets or Superpaint, MacPaint or MacDraw pictures, or their portions can be copied and pasted (paste link) into Word documents with *live links*. Macintosh System 7.0 makes this feature available to all Macintosh-based tools. The Compound Document Architecture (CDA) of Digital Equipment Corporation supports live links among all the tools that conform to the CDA, like DECview and DECwrite.

6.10.4.7.2. Presentation Graphics

Two-dimensional drawing tools like MacDraw, Canvas and Draw Perfect have become indispensable to those who often prepare view graph or slide presentations. Each line, arch, box or text object retains its identity so it can be edited.

Those who need more creative latitude use two-dimensional paint programs, like MacPaint, Fullpaint and Color MacCheese. Once pasted into the work space, paint objects loose their identity. They must be re-identified using rectangular or arbitrary cutting tools before they can be moved, copied or pasted. Canvas is one exception to this norm. It retains the object identity of pasted images.

Canvas and Superpaint offer a combination of drawing and painting capabilities.

6.10.4.7.3. Spreadsheet

Spreadsheet tools have become very powerful, adaptable and easy to use. Many of the software tools listed in the category of Mathematical Software tool kit are available with a modern computer-based spreadsheet like Wing-Z™, Microsoft Excel™ or Lotus 1-2-3™.

6.10.4.7.4. Electronic Mail

Electronic mail allows information to be disseminated to many people quickly regardless of their geographic location. The store and foreword capabilities of electronic mail tools avoids the problem of "telephone tag." The messages are

usually stored once on a computer or on a network of computers. All-In-One™, Microsoft Mail™ and Oracle Mail™ from Digital Equipment Corporation, Microsoft Inc. and Oracle Inc., respectively, are examples of electronic mail tools. Microsoft Mail will even issue a telephone-like ring to alert the recipient of new mail. It will also allow files of any type to be "attached" to a message.

6.10.4.7.5. Voice Mail

Voice mail is a telephone enterprise version of an answering machine. To avoid "telephone tag," messages can be left by the caller for the recipient. Some systems will send messages to many people on a distribution list of telephone numbers much like electronic mail text is distributed.

Some tools like Microsoft Mail allow messages to be expressed in text, voice or a combination of data types.

6.10.4.7.6. Desk Management

Although there are many commercial desk top management products available, HyperCard is probably the most renowned example of a tool for managing telephone numbers, appointments, disks, files and other information associated with a conventional office as "cards" in "stacks" that can be sorted and browsed. The collection of its standard stacks is equivalent to many organizer tools.

6.10.4.7.7. Project Management

Project managers and those who like to plan the use of their time will use project management tools, like MacProject to quickly and graphically create task dependency diagrams. More powerful project management tools like QuickNet, Artimus and P2 are too difficult to use for a general use tool. Quicknet may be used for product development Program task planning and scheduling purposes. Artimus and P2 are similarly appropriate for entire enterprises.

6.10.4.7.8. Custom Data Management and Decision Support

There should be no custom tools unless there is not likely to be a commercial alternative. However, the cost of developing tools with products like HyperCard™, Helix/VMS™ or 4th Dimension™, which require no programming experience is such that they may be used to develop a temporary tool until a commercial alternative is available. The temporary tool may in fact be used as a requirements prototype by a commercial developer. This approach will be especially viable if the data management and decision support products interface directly with the framework via ATIS.

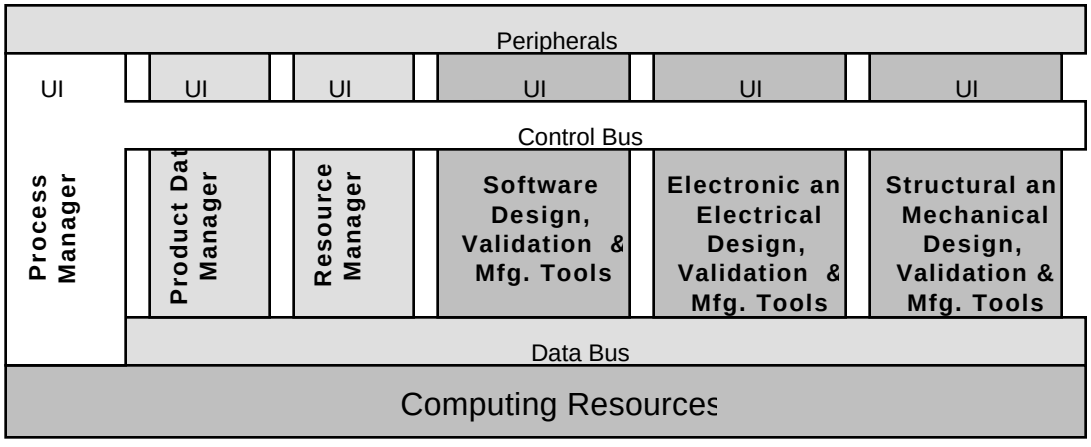
6.10.4.7.9. Graphic Feedback

Communicating questions, problems or suggested changes to the originator of a design in a distributed computing environment in the context of the original design has been a problem. Analysts write reports and verbally attempt to describe the part of a part which concerns them using text. Designers must interpret the text and determine what part of the part to which the feedback applies.

Until DECview 3-D™, there has been nothing equivalent to "red-lined drawings" for three-dimensional models. With DECview 3-D™, analysts, tool designers or the like, can display a copy of the original design, orient it for their purpose, and superimpose notes and arrows on the model for review by the designer. By providing the feedback in the context of the design, designers can immediately recognize their model and can quickly orient it to view the questionable area.

6.10.4.7.10. Process Manager

A Process Manager allows human resources to define, extend in breadth or detail, modify, review or schedule tasks as a function of their authority. Once invoked by a user or triggered by time or some other condition (constraint met), the Process Manager invokes those functions it can cause to be performed. It notifies human resources to invoke the functions it cannot cause to be performed. Some of the subtasks defined may include the transfer of a software tool and data to the workstation to be used by the person responsible for a scheduled subtask. If a change is approved that affects on-going work, human resources are notified and computer programs stopped automatically (electronic stop work order). This can be done according to phase-in criteria (use existing inventory of parts before switching to replacement part) that will minimize scrap while achieving the goals of the change.



The Process Manager uses and maintains in the framework the Task Breakdown Structure (TBS), which is described in the Information Integration section. The TBS may be examined and changed using the graphical browser, which is described later.

An enterprise must first define the way it will do business in terms of a set of allowable work categories and a sequence of workpackage and task milestones. Each product development effort (Program) must define the way it will do business in terms of an appropriate subset of the allowable milestones and work categories. The program milestones will likely differ according to the customer (commercial, NASA, Navy or Air Force contract).

Any original design work will normally be delineated as a set of projects. Each project pertains to a particular aspect of a product (e.g., safety, navigation, fuel system). Each project may consist of one or more workpackages. Each defines work that is pertinent to a particular part or area of the product (navigation section, cockpit). The traditional Work Breakdown Structure (WBS) can be derived from the TBS, which should be consistent with the work proposed.

Any change activity can be defined as new workpackages, or incorporated as revisions or additions to existing workpackages. A significant change, like extend range, may be defined as a separate project.

Original design-related projects and workpackages or the entire TBS of successful Programs can be copied and modified for use by new Programs. This will not only save a lot of work, but also promote the better practices. If prior work is similar to proposed work, it will provide a good indication of the cost, schedule and resource requirements of the proposed work.

For example, a Program manager (human resource) may copy the TBS of a similar Program, modify its program description, delete and add projects. A statement that the projects defined were for another Program and should be modified as appropriate for the new Program can be included in the copied project descriptions automatically.

Project managers (human resource) would modify the project definitions copied by the Program manager, or copy or originate their own project definitions as appropriate. When originating a project the project manager will

describe the project,
delineate the workpackages involved and
estimate the duration of each workpackage.

Project manager labor costs would be charged to that project as a project definition task.

Similarly, *cognizant engineers* (human resource) assigned to a project can copy and modify or originate workpackages. When originating a workpackage the cognizant engineer will

describe the workpackage,
select the appropriate work category and workpackage (initiation, definition, approval) and task (design, analyze, prepare for manufacturing ... produce) milestones to the extent reasonable,
delineate the new, replacement and replaced items involved,
identify the organizational entities (departments) responsible for each task, and

estimate the duration of each task.

The Process Manager will automatically specify the file name for any new manifestation, derivative or other deliverable as described in the Information Integration section. Upon completion, the cognizant engineer will update the status of the workpackage. The status change will trigger electronic mail and perhaps voice mail messages to all the departments Cognizant engineer labor costs would be charged to that workpackage as a workpackage definition task.

By default the managers of the departments are the *responsible engineers* (human resource) for workpackage tasks until they delegate the responsibility to someone in their department. Each responsible engineer will with the aid of individual engineers

refine the task description,
delineate the sequential and parallel subtasks necessary to complete the task,
describe each subtask,
identify the deliverables of the lowest-level subtasks,
select the manifestations appropriate for each deliverable,
select the appropriate derivatives of the manifestations,
select the appropriate validations.
select skills and proficiencies needed to conduct each subtask,
estimate the time required of each skill to conduct the subtask.

When finished, the responsible engineer will update the status of the task. This will trigger electronic mail and perhaps voice mail messages to the cognizant engineer. The cognizant engineer should then begin a dialogue about the task definition and duration. Responsible engineer labor costs would be charged to that task as a task definition task.

The responsible and cognizant engineers and project managers may simulate the effect of their respective tasks, workpackages and projects on the enterprise and identify resource conflicts and slack time or unrealistic schedules by supplying a start or end date. The Process Manager will consult the Resource Manager (described later) for resources that have the specified skills and proficiencies and are available to conduct the defined subtasks during the period indicated.

If the cumulative time required of a single skill exceeds the subtask duration estimated by the responsible engineer the cumulative time required by a workpackage or exceeds the workpackage duration estimated by the project manager, the Process Manager issues a warning accordingly. If the cumulative time required of a single skill is less than the subtask duration estimated by the responsible engineer, the skill is assumed to be required "part time." The Process Manager will alert the responsible engineer to the possibility of condensing a subtask. Similarly, if the cumulative time required by a task is less than the task duration estimated by the cognizant engineer, then the cognizant engineer is alerted to the possibility of condensing a task.

The responsible and cognizant engineers will confer to resolve any problems they find with the workpackage definition. Once satisfied, each person involved

with the workpackage will approve the workpackage definition. Finally the cognizant engineer will approve the workpackage or submit it to some other authority (project manager) for approval.

Once approved, a workpackage may be invoked according to the project, or if no project, according to the Program schedule. At that instant, the Process Manager confers with the Resource Manager to bind resources to each subtask. Resource conflicts are resolved at the appropriate level of management (responsible engineers, cognizant engineers, project managers or Program managers).

Account numbers appropriate for the work may appear in the workpackage description at the workpackage, task and subtask levels. The related human resources are notified of pending work. They can contest the reasonableness of the schedule and intent of their subtask, and renegotiate the subtask with the responsible engineer. Individual engineers may also create a subtask and associate it with a task. The responsible and cognizant engineers are notified accordingly.

As the work commences, any stop or lift stop orders deduced by the Process Manager from the workpackage description will be executed by the Process Manager. The product Function, system and assembly breakdowns may be extended by the individuals assigned to subtasks which impact the Breakdown Structures. Additional manifestations and derivatives may be added to the workpackage definitions. Other aspects of the workpackage can readily be revised as more is learned. As the individual engineers and computer programs complete their subtasks, the computer dates will be posted as actual completion dates. When all of the subtasks of a task are completed, the task is complete, and so forth until the workpackage, project and program are complete.

Although it will be tedious to prepare the first workpackages, it will eventually only involve changes to copies of old workpackages that involved similar work. The better workpackages to copy may be found by way of the similarity of the attributes of the deliverables involved in the work. Even the first workpackages will be developed and a work definition and schedule consensus and resource conflicts will be resolved far more quickly than is currently possible, because work definition development, reviews and approvals will be performed interactively.

The ability to define work to the level of an individual resource, allows the activity of all resources (human, computer and machine) to be interleaved. Designers, analysts and analytical tools can be coordinated. Machine operators and machines can be coordinated as pairs of resources and as members of manufacturing cells. Manufacturing cells can be similarly coordinated.

The context of the process is maintained in the Task Breakdown Structure by the Process Manager. The Function, System and Assembly Breakdown

Structures of the product are maintained by the Product Manager (described below). All of the deliverables (manifestations, derivatives) that may be affected by a change (parts, specifications, requirements, analyses, tests, kits) can be instantly assessed and displayed by the Process Manager. The investigative part of impact analysis is rendered essentially automatic. Those developing the workpackage for a change need only select from the displayed lists those items that should, in fact, be included in the change.

Accountability (audit trail) is provided by the Process Manager. The Process Manager may utilize the Product Data Manager to determine when deliverables are connected (SBS) to or spatially near (ABS) each other. Then the Process Manager can notify the appropriate resources that the work they are performing may affect or be affected by the work of others.

Resources typically do not work at the same task day after day. A design engineer may change a current design version, develop a new version, or analyze a design all in the same day and within the scope of the same project. The Process Manager makes context changes minimally disruptive. A human resource can stop working on one task and start working on another without losing their place, even among different projects or Programs.

The Process Manager does all of this with no data redundancy, unless it is manifested by more than one tool.

6.10.4.7.10.18. Process Manager Scenario

Upon Login at workstation, a user will either be in command mode and invoke the Process Manager front-end on the workstation, or the Process Manager front-end will be invoked automatically when the workstation is empowered. The Process Manager front-end will switch the user directly to the Process Manager on a file server. Upon the selection of a Process Manager icon, an integrated software tool or an encapsulation script will be executed. If the tool is not integrated, the encapsulation gets the data necessary for the task, reformats it as necessary, transmits it to the workstation, and goes into a wait state. At the workstation, the software executes or an encapsulation script invokes the tool and opens the files that were received. The tool will be active until terminated by the user. Any "saves" of shared data (deliverable) from an encapsulated tool are intercepted by the encapsulation, reformatted and saved in an appropriate permanent storage facility. Any "saves" of data that will not be shared are left as work files on the workstation to be managed by the software tool as usual. Upon termination the user is asked for an estimate of the degree of completion of the deliverable (maturity) or its likelihood of change. When the tool is terminated by the user, the reverse occurs.

The intimate involvement of the Process Manager in the process allows actual process activity to be correlated with scheduled events at a subtask level. Process activity can be cumulated up the Task Breakdown Structure to whatever level is of interest for an accurate assessment of progress or cost. It is done in a manner that does not inhibit the freedom of the user to conduct work.

It avoids the need for most users to learn a separate project management tool (described above) and report their progress to it.

It is likely that each function or cell will have its own Local Area Network and computing platforms optimized for that function or cell. Hence, it may be reasonable to distribute the parts of the TBS (workpackages, tasks and subtasks) that pertain to the function or cell to a file server on its LAN. The higher levels of the Program schedule would reside on the LAN of the program management team. If views of the Program schedule require more detail, the Process Manager would seek the latest data through the network from the contributing functions.

This approach has several benefits. It gives the process functions some local autonomy, which allows them to continue to coordinate their internal work regardless of what might happen to the larger network. It improves the overall performance of the network, because the data is physically located nearest its use. It eliminates the need for a Global Access Facility or the like to aggregate data from the Process or Product Manager of each Program for bulk purchases. The top level Process and Product Managers can also assess the net impact of the various Program schedules on one or more manufacturing functions. Each cell will have all of the information it needs to determine whether it should bid for more work.

6.10.4.7.11. Work Broker

Because of the difficulty of finding suitable suppliers or subcontractors, many enterprises have chosen to limit their selection to a few suppliers and subcontractors and establish a relationship with them. This relationship may range from the name of the supplier or subcontractor on an approved list of suppliers and subcontractors to having blanket contracts and direct access to company data.

While this approach may ease the development of contracts and improve the flow of information and the timeliness of part delivery, it limits competition and may thereby increase part cost. Subcontractors perceive it as enslavement. An alternative is to utilize a work brokerage much like temporary services are used.

In this case, the approval of a workpackage triggers the Program Manager to send requests for interest messages, according to the work described in approved workpackages to one or more international work brokerage systems (e.g., GEISCO). The work may involve one or more functions of the process (define requirements, design, validate, handle material, fabricate, inspect, assemble, test).

The request for interest messages would include the delivery dates as extracted from the Process Manager image of the schedule of the Program. Any necessary contract boiler plate text would be automatically forwarded with the request for interest message. If the respondent is not known to the Resource Manager as an approved supplier, the Process Manager answers with a

message that tells the respondent how to become an approved supplier. Otherwise, the Process Manager queries the Product Manager for all the product definition data necessary for the respondent to complete the task or subtask, and sends a request for bid message containing that data to the approved respondents.

Prime contractor engineering and manufacturing functions and subcontractors periodically query the work brokerage. They compare their capabilities profile and work load with the task or subtask descriptions. Those capable of performing the work respond with an affirmative message. The respondent may not, in fact, be capable of performing the work immediately, but may invest in more capability to be able to perform the work in time to meet the delivery schedule.

The respondents compare their capabilities profile with the product definition data. Those capable of performing the work respond with a bid message that includes the cost of the work and the time to complete it.

The Process Manager adjusts (penalizes) the cost of each bid according to any schedule variance and past performance data (quality, cost and schedule compliance). Based on the relative adjusted costs, the Process Manager selects one or more winners from among the respondents. It then sends consolation messages to the losers and work authorization messages to the winner(s).

If there are too few respondents within a specified time, the Process Manager so notifies the cognizant engineer of the workpackage, who must then initiate a change to the workpackage to relax the associated requirements or the schedule.

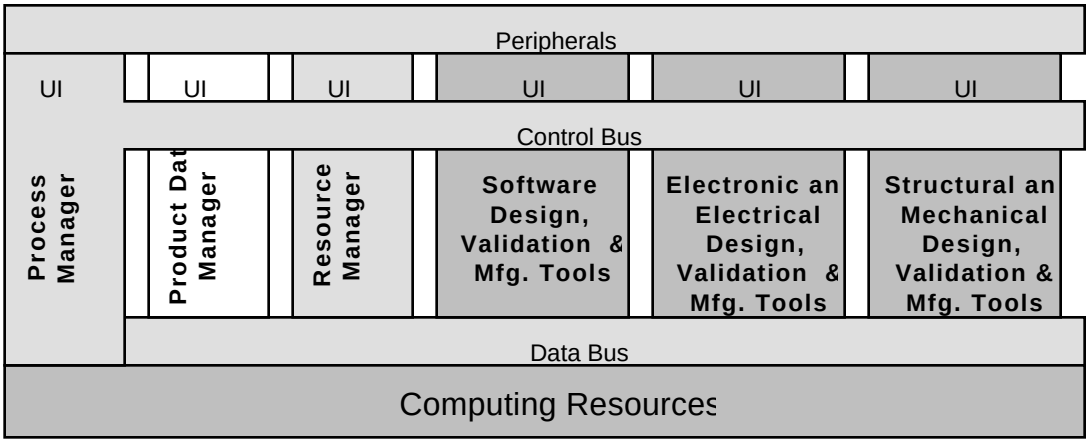
When a winning respondent delivers according to its subtask and the corresponding receiving subtask completes, the Process Manager triggers the finance system to make an electronic funds transfer to the account of the respondent for the services rendered, according to the amount defined in the subtask.

The product definition data provided will vary as a function of the task or subtask. For example, the product definition data may be

- the product requirements text for a design subtask,
- a design model for an analysis or pre-manufacturing subtask,
- a design model and a process plan for a manufacturing subtask,
- a derivative of the design model for an analysis subtask or
- a numerical control program that executes the process plan in the context of the model for a manufacturing subtask, or
- the derived machine control data that resulted from post-processing the numerical control program for a manufacturing subtask on a specific machine.

6.10.4.7.12. Product Data Manager

The Product Data Manager maintains the Function, System and Assembly Breakdown Structures, the digital deliverables defined with the Process Manager, the whereabouts of all deliverables, their derivation (source manifestation or derivative and the tool used) and their attributes. It provides for the change control and configuration management of the Breakdown Structures, deliverables and tools as described in the Information Integration section.



By way of the Product Data Manager, the resources (human and software tools) relate and reconcile interdependent and often conflicting product performance, reliability, producibility and maintainability objectives. A continuous and complete definition of the evolving product from conception through production is provided by way of the deliverables created by the resources and managed by the Product Data Manager. Traceability from requirements through design, analysis and test, and the design decisions that resulted from trade studies are maintained throughout the life of a product by way of the Product Data Manager. This eliminates the duplication of effort that resulted from treating the product development phases as separate processes. The Product Data Manager permits the retrieval and use of the proper version of product data in the many ways alluded to in the Information Integration section. It does so without data redundancy, unless its functions are performed by multiple tools.

6.10.4.7.12.1. Change Control

The relationship between derived deliverables (copied and modified), the models from which they were derived, and the tool(s) used to create the derivation, can be established automatically. This automatic linkage applies to derivatives of derivatives, ad infinitum. A change to a feature of a derivative that is dependent on some feature of the deliverable from which it was derived (source) without a change to the corresponding source feature must be either not permitted or automatically flagged as a potential discontinuity in the product definition. If consistent business rules cannot be established and enforced for the scope of a change throughout the product structure, including features and

attributes, then the scope of a change cannot be automatically determined. It must be defined by a human resource.

When the status of a model, derivative, reference document or any other deliverable known to the Product Data Manager reaches a sufficient level of completeness to warrant its use for rate production, it must be bonded such that it can never be changed. Its official name (e.g., part number) cannot be re-used. It can only be copied under a new name. Changes can only be made to the copy. The name of the deliverable bonded will persist in the data structure indefinitely, or until there is effectively no use for that deliverable. For example, when all the deliverables related to a model, like the NC program and finite element model derived from a solid model of a part, cease to exist, there is no dependency on the model. Then it too can be purged. The longevity of some products like the DC-3 may require that these deliverables and their relationships be maintained for more than 50 years. This does not imply that the entire model or any other deliverable has to be available on-line. Only the name and location (archive) of the deliverable need be maintained on-line.

The knowledge of what user created or modified a deliverable, what user changed the status of a deliverable and when those actions were performed must be maintained for accountability purposes. Many status changes are really "approvals". There can be multiple sequential and parallel approvals.

6.10.4.7.12.2. Configuration Management

A configuration management tool is typically built on top of or integrated with a change control tool. The configuration management tool must support the association of versions of deliverables with specific events during the life of specific Articles of a product. An Article is uniquely identified with a number (serial, tail, hull or chassis). A product is uniquely identified with a model, End Item or configuration item number. The Vehicle Identification Number (VIN) of an automobile for example, is the concatenation of the model and chassis number. It uniquely identifies an automobile.

While there may be many versions of deliverables, only one version of the design is used in an Article of a product at one time. That time is the period between events. Events may involve a change to the configuration (replace navigation section) or the addition, condition or position of a part (fill with fuel) of an instance (Article) of a product or of the entire product (shipped to another facility for the addition of pyrotechnics).

If another version of a system is used on subsequent Articles of a product, and the new version requires a different part or a different version of a part whose form, fit (ABS) or function (SBS) differs from that of the old part, then the unique identifier of the new part must change. This indicates that it is not interchangeable with the old part (see Information Integration section for part identification description). If the new part affects the form or fit of its assembly or function of its system, then the part number of the assembly and/or system must change as well. The part number change must percolate up the assembly and

System Breakdown Structures until the assemblies and systems are, in fact, interchangeable. Thenceforth, the part numbers may remain unchanged. The various used and unused versions remain intact as part of the product structure until they are discarded by their originator or a designate.

One or more product Articles may be retrofitted with new versions of parts. Changes introduced during the production of the product are not made to all of the Articles of a product. Some Articles may have already been produced, delivered or destroyed. Sometimes it is not necessary to introduce a new part until all the old parts have been used. The effectivity of the change must be specified by Event and Article for each product. With the relationships managed by the Product Data Manager, parts that are discovered to be prone to failure or are otherwise bad can be readily identified.

The version histories of all the Breakdown Structures must be maintained. Configuration control must be applied at every level in the Breakdown Structures. Each component, system, assembly, function and task must have a version number associated with it as an attribute. If a deliverable or any of its attributes change, its version number must correspondingly change. A part must inherit the largest version number (or newest date) of the features which comprise it. A system or assembly must inherit the largest version number of the components of which they are comprised. A product must inherit the largest version number of the systems or assemblies of which it is comprised.

This version number inheritance technique allows computer programs as well as humans to determine that a change has occurred (version number is larger than the one previously known to an individual or tool). It indicates what has changed, regardless of the level in the hierarchy interrogated.

The version number does not necessarily correlate with the computer file versions that are used to identify the versions of files on a computer. It should be a deliverable attribute that is internal to the Product Data Manager. The Product Data Manager should shield the user with a more friendly and useful interface to version information. Most users are not interested in version numbers, Event numbers or Article numbers per se. They want to see the latest or previous deliverable, or what deliverable is used for a certain circumstance. The users should be able to ask for versions according to such criteria. The Product Data Manager should make the inferences necessary to provide the appropriate version of the requested deliverable.

The ability to have and track versions allows a greater degree of concurrency than what might otherwise be possible. One designer could be working on one version of a deliverable while another copies it and proceeds to modify it as a different deliverable or another version of the same deliverable.

Version number inheritance and the fact that version numbers can be used throughout the life of a product will cause version numbers to become large, especially at the assembly or product levels. An alternative is to use the computer date and time as the version number instead of an integer number.

Nearly all platforms use date and time to help their users maintain their files. Some make date and time functions available to software tools as well. This approach would ease the implementation of a version number technique, were it not for the fact that not all platforms provide a date and time comparison function. An integer version number is far easier and faster for tools to compare than a date and time version number. Although each platform will not save files with the same date and time, a network of platforms could cause problems with this approach unless their clocks are synchronized or they use a network clock.

The *effectivity* of the systems, components, features and assemblies can be expressed by *End Item* (product), *Article* (serial, hull or tail number) and *Event* (manufacture, test, refurbish, etc.). A specific version of a product is *effective* for a specific Event in the life of one or more Articles of an End Item as are the corresponding versions of all the assemblies and their components.

The effectivity of a specific item may be determined by following its version parentage (items with the same version number) up the product hierarchy to the product level where the applicable Event and Article can be found. If the version parent path leads to more than one Event and/or Article, then the item is *effective* on more than one Event and/or Article. For efficiency reasons, it may be necessary to store effectivity data at intermediate levels in the product structure. Some product, assembly, part or feature versions may never be effective as real products, assemblies, parts or features. Alternate parts should only be associated with their next assembly by way of a preferred part designation.

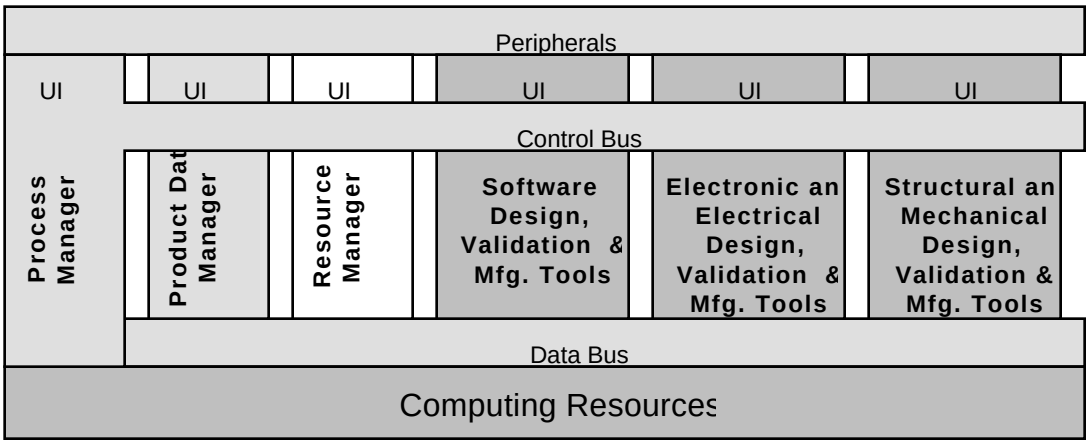
Configuration management allows people and processes to hold, track, manipulate and communicate all of the relevant information about a specific design. It provides an environment where data about design iterations can be shared. It provides the means of establishing the proper data context for work. Multiple configurations can exist to capture the state of a system at different points in time or to capture variants of the system. Product lines, as well as products, can have configurations.

6.10.4.7.13. Graphical Browser

A graphical browser is not explicitly shown on a diagram, because it is pervasive. It provides the capability to graphically traverse and penetrate the Process (TBS) and product Breakdown Structures (FBS, SBS and ABS). A graphical browser will enable users to navigate intuitively to information of interest via these Breakdown Structures. It will allow authorized users to edit the Breakdown Structures, including the creation, deletion, extension and decomposition of a breakdown structure. It will allow breakdown structure nodes to be linked. It will allow all or portions of one or more Breakdown Structures to be displayed in a single window at any level is desired. It will provide the capability to retrieve, display and store objects associated with a selected Breakdown Structure node.

6.10.4.7.14. Resource Manager

The Resource Manager manages the inventory of resources and the skills of those resources. The inventory may be limited to the resources of an enterprise or extend to include those of its subcontractors. As new computers, tools, people, machines and facilities are acquired by the enterprise, their unique identity, physical location, skill set and proficiency must be made known to the Resource Manager.



Directly or by way of the Process Manager, the Resource Manager makes known to resource consumers what skills are available to develop or manage the development of a deliverable as defined in the Task Breakdown Structure. When a task is scheduled, the Resource Manager lists what resources with the needed skills and proficiencies are available when they are needed. If more than one task requires a skill that can only be provided by one resource, then a resource conflict exists. The Resource Manager notifies those requiring the skill accordingly. They must resolve the conflict by changing schedules, proficiency or skill requirements, or by using a set of resources whose collective skills are equivalent to that of the originally desired resource.

6.10.4.8. Special Purpose

These are the many commercially available software tools or the functional requirements for those that are needed. Although only software, electronic/electrical and structural/mechanical systems are discussed at any length in this section, this is not meant to imply that they are the only systems that merit discussion. These three domains have unique manufacturing processes. The functions related to their design, validation and manufacturing process are representative of those of the multitude of other systems (hydraulic, pneumatic, pyrotechnic, optical, thermodynamic, aerodynamic, etc.). To describe each would be largely redundant.

Tools that apply to all domains are described first. The term *validation* as used in the description of tools in each domain includes analysis and test.

6.10.4.8.1. Requirements Definition/Allocation

RDD-100 from Ascent Logic was originally intended to be a software requirements definition and management tool. It is useful for the definition, allocation and maintenance of electrical, structural, mechanical and other system requirements. It is an appropriate requirements definition and allocation tool for all domains and system types.

6.10.4.8.2. Materials Selection

Many commercially accessible databases are available for a variety of materials. Unfortunately, they are not integrated. Metallics, ceramics and various plastics databases must be searched to discover all the material options for a particular design requirement before the optimum material can be selected. Most do not include acquisition costs, let alone the costs associated with storing or fabricating the material. Some materials, like thermoplastic composites, require expensive storage conditions to retard degradation.

Many materials suppliers provide databases of materials information. Some provide expert tools. They ask questions, and based on the answers, guide the user to an appropriate adhesive for example. Naturally, they tend to guide the user to the selection of a material available from the supplier.

Some industry associations provide less parochial databases, but they often are masses of material property data. There is no intelligent human interface to help the user to quickly reduce the number of materials choices. Designers cannot really be expected to sift through a maze of strength, creep and other charts that vary according to environmental conditions, read all the caveats to identify a few candidate materials from among 40,000 metals and 20,000 plastics, and then research and compare acquisition, handling, storage and fabrication costs and availability in order to select the optimum material. Help is needed.

Some of the factors to be considered when selecting a material are paraphrased from a table in Materials and the designer by E. H. Cornish:

FACTOR	PARAMETERS
Aesthetic	Color possibilities
	Optical clarity
	Surface finish possibilities
	Freedom to shape in smooth curves
	Lightness or heaviness
	Surface properties
Safety Aspects	Toxicity
	Flammability and possibility of smoke emission
	Avoidance of sharp points and edges
	Shielding of electrical parts
Environmental	Effect of temperature on properties
	Degradation due to thermic and electromagnetic radiation
	Weathering
	Moisture permeability
	Resistance to chemical attack from acids, alkalies, solvents

	Resistance to solvent stress cracking
	Influence of active environments (vibration, thermal expansion) on fatigue
	Fatigue effect
	Ductile/brittle transitions
	Influence of humidity on creep and electrical properties
	Resistance to biological attack
Mechanical	Tensile modulous under static and dynamic conditions
	Creep and creep rupture at various temperatures and stress levels
	Tensile strength
	Stress relaxation at various temperatures and strain levels
	Dynamic stiffness at various temperatures, frequencies and stress levels
	Resistance to wear
	Compressive strength
	Tear resistance
	Hardness
	Impact strength at various temperatures
	Notch sensitivity
	Effect of surface finish
	Friction properties
	Dynamic fatigue at various temperatures and stress levels
Processing	Stiffness
	Strength
	Forming options (bend, beat, explosive, inject)
	Material removal options (cut, grind, erode)
	Assembly options (fasteners, adhesives, compression bonding)
Thermal	Shrinkage from the mold
	Thermal expansion
	Specific heat
	Dimensional stability
	Upper and lower temperature limits in service
	Heat conductivity
	Softening temperature
Acoustic	Vibration absorption
	Damping capacity
Electrical	Electrical strength (ac and dc)
	Tracking resistance (ac and dc)
	Volume and surface resistivity
	Dielectric constant at a range of frequencies
	Loss tangent at a range of frequencies
	Conductivity
	Arc resistance
Magnetic	Susceptibility
	Coercively
	Permanence

Consequently, the optimum material is seldom selected.

The aforementioned services are offered for free or at cost, so there is little room for complaint. Commercial services can offer the better of both: industry-wide data and good user interfaces.

The cost of maintaining this information for a specific enterprise in a specific format is prohibitive. Commercial services can reduce the cost by spreading it among many customers.

6.10.4.8.3. Part Selection

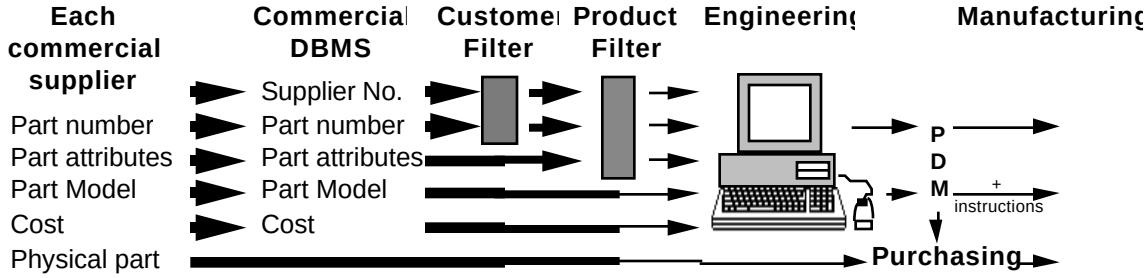
Bolts are representative of a family of parts, which could be maintained in a standard parts library. Defining every bolt used in a product as a solid model would be expensive. The shaft length, shaft diameter and thread pitch feature attributes are the variables among a class of bolts. The head diameter and height and driver geometry are functions of the shaft diameter. Bolt length is limited by bold diameter. All of these attributes can be defined parametrically. From this information, specific instances of bolts can be automatically derived. Most fasteners and other often used parts (tube and pipe fittings) and assemblies (connectors, valves, switches) can be economically managed in this manner.

Enterprises cannot afford to maintain an accurate database of parts. Consequently, several part database suppliers are vying for prominence as the *parts broker*. Manufacturers cannot afford to supply the parts libraries of every major customer with current information, especially when each demands that it be provided in a different format. Consequently, many are supplying part number, attribute, properties, cost and availability information to parts brokers. Customers then subscribe to the database of the broker to get the latest parts information.

Since many manufacturers use electrical and structural/mechanical design tools to develop their products, their models can also be up-loaded to the parts broker database for direct use by those designing new products. No longer do the designers have to recreate the part model. Unfortunately, the three-dimensional models of the manufacturers are being converted to two-dimensional images or CAD models (IGES) for the parts broker systems. The parts brokers have been encouraged to use solid models as the basis of future versions of their products. All the other model types can be derived from a solid model as appropriate for each customer.

The parts brokers provide search tools which facilitate the selection of parts according to their part number or any combination of attributes (material, strength, etc.). The names (numbers) of specifications and military standards can be part attributes as well. Hence, part attributes can also be used to find all the parts that satisfy specified requirements as well as meet physical criteria.

Part supplier, number and attributes can also be used as filter criteria to eliminate from consideration those parts that have been shown to be unreliable or prone to failure. The filter criteria is also a function of the environment for which the product is intended. The more part attributes specified in the query, the smaller will be the set of parts selected for further consideration.

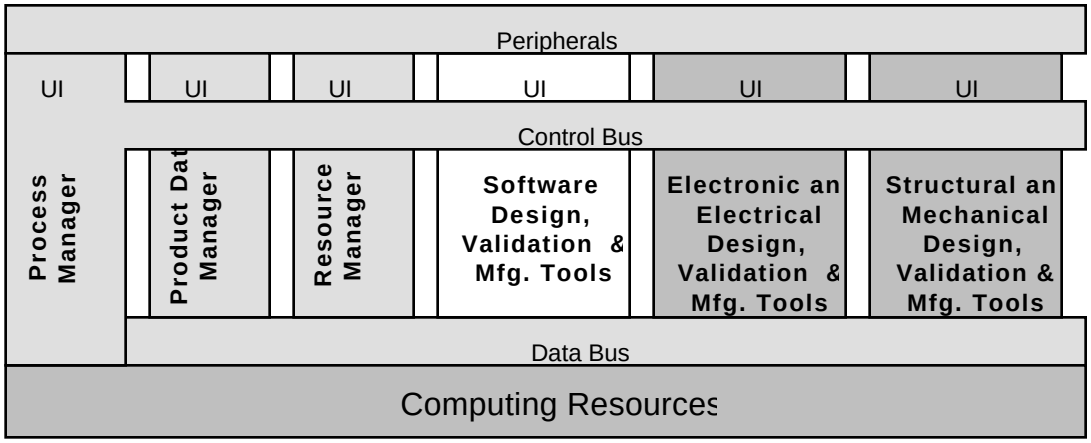


Designers need only specify a few attributes and their values for a broad part class (fastener) to have all the allowable parts displayed. Not only can their models and range of allowable physical characteristics be displayed, but also all the standards with which they are compliant and any caveats in their use can be displayed. Designers would then select the most desirable standard part, supply the values for its additional physical attributes, and have the corresponding geometry generated for placement in a systems or assembly design.

Sometimes a part with parameters in between those of "off-the-self" parts may be considered worth the price of custom manufacturing. Parametrically defined parts allow the part attributes and physical characteristics of imaginary parts to be inferred from the part attributes of known parts. If a designer is looking for a part and finds that none of those defined thus far are suitable, the designer can disable the software function that restricts selections to standards, specify the parameters of concern, and let the system interpolate a suitable part accordingly. The attributes may be interpolated as a function of the shape of the part or vice versa. The imaginary part may then be inserted in a system design and its effect on the system can be compared with the effect of standard parts to determine if a custom part is in fact worthwhile.

6.10.4.8.4. Software Design, Validation and Manufacturing Tools

These are largely what is known as Computer Aided Software Engineering (CASE) tools. Software Through Pictures is a popular software design and documentation tool.



There are no software analytical tools that compare with the electrical or mechanical analytical tools. Currently, software validation largely consists of compilation, run, debug and modify iterations. This is equivalent to a manufacture, use and re-manufacture cyclic process. Design and manufacturing, rapidly done (rapid prototyping) is the best method currently available for the development and testing of unambiguous requirements and product alternatives.

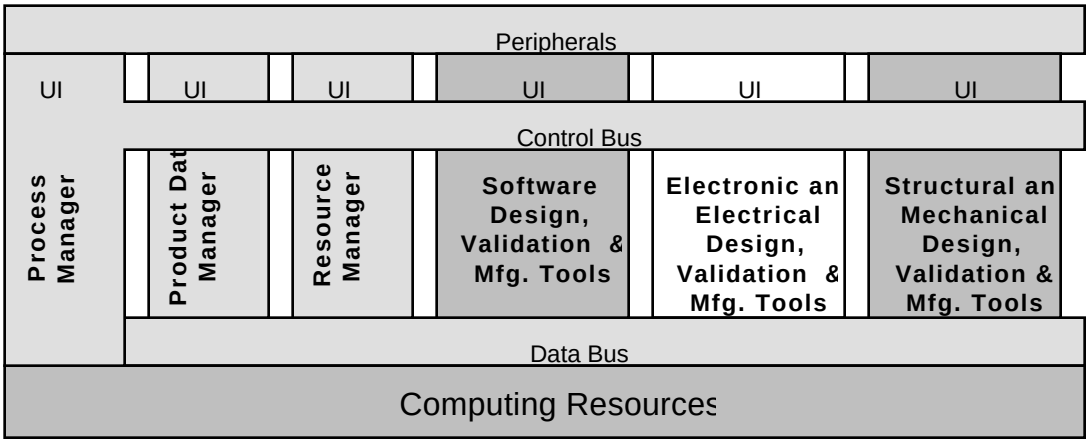
Software fabrication is the compilation of the software. Software assembly is the linking of the software modules into *object code*. Shipping is the copying and distribution of object code. Delivery is the installation of the compiled and linked software (object code) on the memory of a particular computer.

The true behavior of software cannot be simulated separate from its operational environment: electronic circuits and the electrical systems that interconnect them. As products become more complex, it will become essential that electronic/electrical hardware simulations be stimulated by simulations of the software that is to run on the hardware. Else subtle anomalous behavior will not be discovered before the cost to eliminate it becomes many multiples of the cost to fix the problem before hardware is built.

6.10.4.8.5. Electrical and Electronic Design, Validation and Manufacturing Tools

These are the tools used for integrated and printed circuit design, simulation, analysis, layout (detail design) and arrangement and routing (packaging). Mentor Graphics and Daisy are commercial examples of tools for electrical and electronic design and validation. Although a full complement of analog and digital simulation tools are available, the construction and testing of prototype

circuit boards is still a common practice because the manufacturing process is relatively inexpensive.



The manufacture of electrical systems largely involves the fabrication of circuit boards and discrete components. Conductor material (copper) is etched from circuit boards or layers of semiconductor are etched to achieve the desired connectivity among discrete components. The fabrication of discrete components involves the assembly of structural, thermal and electrical materials or the deposition of impurities in semi-conductors to induce the desired effect. These components are then packaged and assembled on the circuit board. Testing is usually imposed at each step.

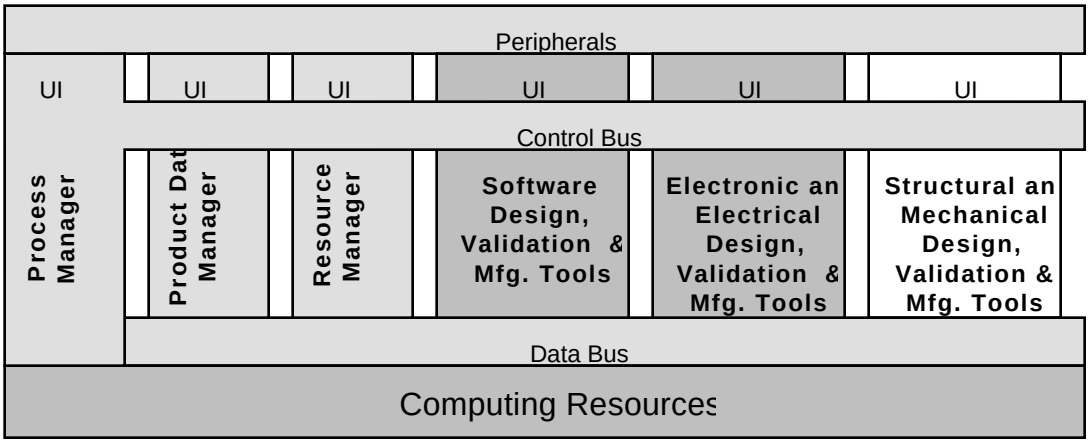
A result of virtually every new electronics design is the need for nonstandard fixtures or "special test equipment" to verify that the design meets the requirements. Hence, an electronic design should include interface, stimulus and response information that may be used by test equipment designers.

Where once the input stimulus for electronic circuits was exclusively switches activated and deactivated by people or mechanical systems, now software plays a predominant role. Hence, the need to have software and electrical simulations interact. Control is where the software and electronic domains interact.

Although software occupies no space, the medium that maintains the digital patterns that represent software (ROM, disk, bubble memory, tape), does occupy space. Space is where the electronic/electrical and structural/mechanical domains interact. Given a printed circuit board profile and the placement of components on the board in two dimensions and the thickness of the board and the heights of the components from a component library, a solid modeling tool can be used to extrude the components and board into a full three-dimensional solid model of the assembly. Wire harnesses can be similarly simulated by tubes. The arrangement of the electronic/electrical system among the other system types can then be simulated. The model will also be useful for assembly and maintenance simulations.

6.10.4.8.6. Structural and Mechanical Design, Validation and Manufacturing Tools

These are the tools used to create and validate two- and three-dimensional System, component and assembly geometry and annotate it with text.



Here again, the behavior of one system cannot be accurately predicted without considering its interaction with other systems. The external loads of air vehicles are caused in part by the aerodynamic and flight control systems. The external loads of water and land vehicles are similarly caused. Mechanical systems (suspension system, actuators) affect the structure that must resist it. Heat generated by other systems can distort and weaken structure. Vibration generated by other systems can fatigue structure. Structural distortions can alter the behavior of mechanical, aerodynamic, fluid dynamic, optical and acoustical systems. Consequently, comprehensive system simulations should be at the level of the entire product. Cumulating them from system-specific or assembly simulations can be misleading.

6.10.4.8.6.1. Master Dimensions (Lofting) Tool

The external and internal wetted surfaces of a product are often the first to be defined, especially if the product moves or is otherwise subject to fluid dynamic constraints. These surfaces constrain the structural system. The structural system constrains all the other systems and the arrangement of their components. The size of the product may vary parametrically, but its proportions will likely have to remain unchanged if the aerodynamic or hydrodynamic characteristics of the product are to meet its requirements.

Master dimension systems use some of the most sophisticated surface definition algorithms available to describe explicitly the exposed surfaces of a product. They include algorithms for computing variable off-set surfaces to provide constraints for structural designers. *Non-dimensional* constraining geometry (lines plotted to scale in the view plane of the drawing) is generated for the designers and draftspersons who still use manual techniques (drafting tables).

Some of the master dimension systems evolved into wireframe and surface design systems that designers could use. Some now include solid modeling capabilities (CATIA). Many solid modeling systems have evolved to include sophisticated data management functions (Euclid). Master dimension systems that were based on mainframe computers can run on workstations. The tool needs of the loftsperson and structural/mechanical designer can now be satisfied by one tool.

With the addition of *fuzzy surfaces*, this combination of tools will allow the design to proceed concurrently from the outside in (gradual dedication of volume to function) and from the inside out (incorporation of off-the-shelf parts into the design).

6.10.4.8.6.2. Parametric Design Tool

A solid a modeler should ideally allow model geometry to be defined such that variations in one geometric variable could be immediately reflected in other geometric variables within a part, like a spreadsheet, and among parts within an assembly, like linked spreadsheets. Unfortunately, making solid model geometry parametric can strain computing capacity. Some viable design products have arisen from parametric design tools with limited geometric definition capability. These products were targeted at those involved in preliminary design, where the product definition is sufficiently abstract to be represented by a parametric design tool. As the product design becomes more refined, however, the limited capabilities of the parametric design tools soon become apparent. Only products with few systems and components can be accommodated.

Solid modeling tools can be as inappropriate for preliminary and conceptual designers as parametric design tools are inappropriate for detail designers. A preliminary designer may be willing to specify that a bolt will be used, but not willing to specify its type or diameter, let alone its thread size. A conceptual designer may not be willing to commit to the use of any specific fastener, because an adhesive may still be a viable option. If the designers involved during the detail phase are to capitalize on earlier work, the interface between parametric and solid modeling tools must be seamless or solid modeling tools must incorporate parametric design capabilities. Feature, part and assembly attributes specified in the earlier phases must be useful to later phases. Their abstractions should provide a useful basis for more detailed manifestations.

Derived geometry must retain its parametric links to the manifestation or derivative from which it was derived unless they are explicitly broken. Attributes associated with the geometry (thermal emission, volume) should change according to geometric changes or other influences.

As algorithm efficiencies and computing capacity improve and the cost of computing resources declines, parametric design will become an integral part of solid modeling tools. The parametric design of complex features will become

appropriate for the detail design of systems, components and assemblies, and still be appropriate for preliminary and conceptual design activity.

6.10.4.8.6.3. Solid Modeling Tool

Many thanks to Richard Fridshal and Steve Sheldon for the refinement of my solid modeling education.

A key software tool is a solid modeler and the tools normally associated with a solid modeler. A solid modeling tool supports the design of structures, mechanical devices and linkages, wire harnesses and tubing. A solid modeler usually also supports visualization (shaded surfaces), component and assembly arrangement, documentation, validation (volume properties) and numerical control programming.

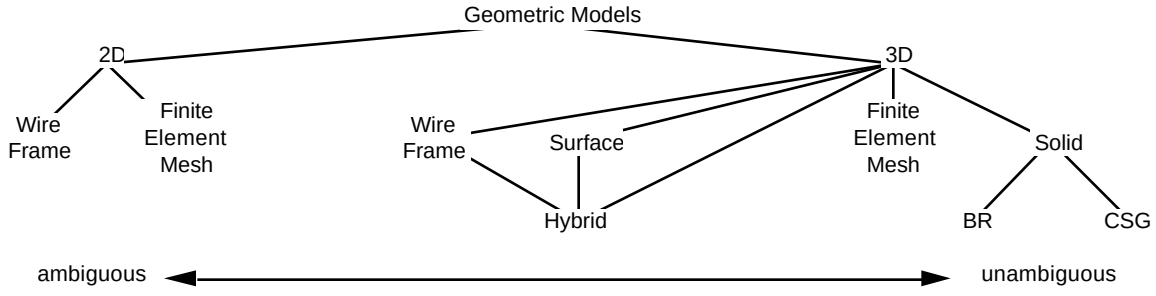
Euclid from Matra DataVision is a solid modeler with a complementary surface modeler. CATIA™ from Dessault is a popular surface modeler with solid modeling capabilities. Both are predominantly used for structural and mechanical design.

The users of electrical/electronic design systems (schematic capture, component placement) get printed circuit board constraints from the solid models. They supply printed circuit assembly data that can be converted to a solid model for arrangement purposes. Various other software tools are used by down-stream functions to expand and validate the product definition for performance, manufacturing design, reliability, producibility and maintainability purposes.

Solid modeling serves to render a design less ambiguous. It supports "transition to production." Solid modeling allows components and assemblies to be easily visualized, and component interferences to be easily detected.

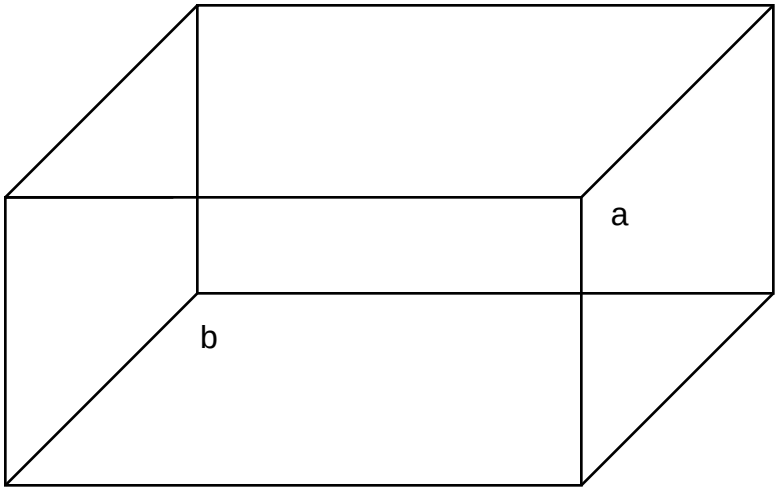
6.10.4.8.6.3.1. Modeler Types

Major mathematical innovations were responsible for the evolution from modelers which were limited to two-dimensional (2-D) wire frame to three-dimensional (3-D) wire frame, surface and solid modelers. As the following diagram indicates, the ambiguity of the models declined as the mathematical rigor and model content improved.



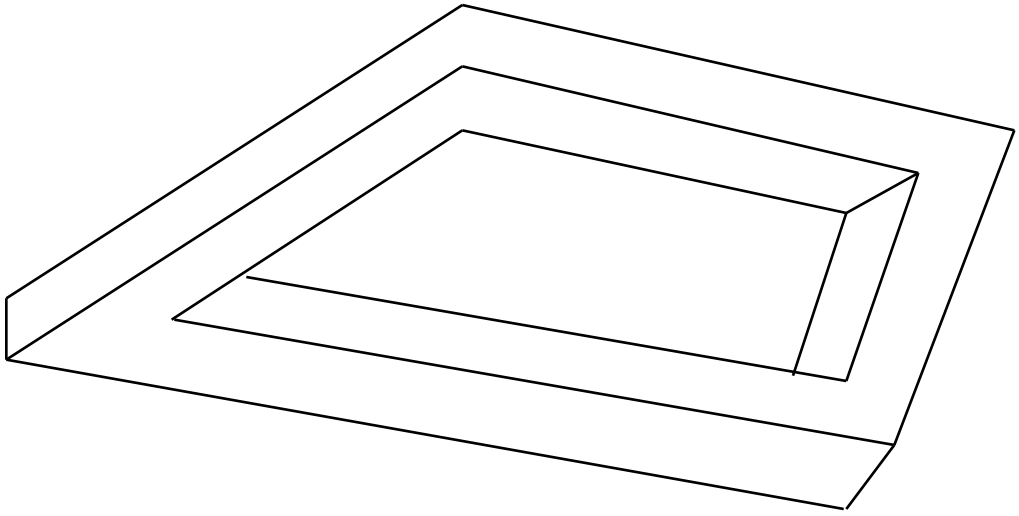
2-D wire frame models emulate a traditional drawing, and share its ambiguity. It is a tedious job for humans to check a drawing or the model from which it was plotted for errors. The arrangement of the views on a drawing give someone versed in drawing convention the relationships needed for them to imagine a 3-D model. Only recently have those conventions been compiled into a knowledge base sufficiently robust for a computer program to construct an accurate wire frame model from a paper drawing. This capability is known as *rectification* as provided by Metagraphics, Inc. As sophisticated as it is, the computer program must still ask a human to confirm the assumptions made by the computer program about simple ambiguities, and to reconcile difficult ambiguities.

The earlier mathematics, display technology and computer processing power provided fairly expensive 2-D drafting systems and 3-D wireframe systems. Wireframe models only represent the edges of the physical objects being modeled. They do not provide enough information to determine whether a point is outside, on the surface, or inside an object. Unless line intensity varies with depth or circular polarization, red/blue color encoding and complementary polarized or colored "3-D" glasses must be used to provide a depth cue. Wireframe models are not only ambiguous to software tools, but also visually ambiguous to humans.



Which corner is in front, a or b?

Although wireframe models can be accurately interpreted from the data used to display them, the models are useless for the calculation of mass properties or the generation of fabrication instructions (generative NC). Wire frame modelers can also be used to create *nonsense* objects:



Surface modelers require users to define a planar or non-planar profile of lines, arcs or splines (uniform, non-uniform or rational B-spline or Bezier) to constrain a surface. The surface can be *quadric* (second order polynomial) or *sculptured* (third through n-th order polynomial). Sculptured surfaces are like thin membranes stretched upon a bent wireframe (profile) by weights (control points). The surface of a non-planar profile can vary significantly as a function of control point values, much like the surface tension on a membrane will affect its shape. This variability is what complicates the transformation of some surface data from one form to another. It results in the inexact replication of data when it is used on different systems.

Surface modeling describes part surfaces but not their interiors. Surface models can be used to perform limited interference checking where the surfaces of parts partially penetrate one another, but surfaces that are wholly within a surface model will not be detected. Humans are still required to examine surface models to determine if there is, in fact, an interference problem.

Unlike surface modelers, which require the designer to insure that the various surfaces on a model are contiguous. Solid modelers start with a solid and use topology rules to guarantee that all the surfaces are stitched together properly. A solid model has enough information to determine whether a point is outside, on the surface, or inside an object. A surface model does not. Automation requires a complete and unambiguous product definition. Solid models provide the foundation for that definition.

6.10.4.8.6.3.2. Construction, Representation and Display

To describe any modeler,

the *construction* techniques available to define a model,
the *representation* of the model,
the *display* techniques and other
applications of the model
must be distinguished.

Two methods for representing solid models in a digital database are the constructive solid geometry (CSG) representation and the boundary representation. A solid model may be *constructed* (created, designed, input) using techniques that are independent of the representation of the model. Unfortunately, many people assume that model construction techniques are limited to those that may be inferred from their representation technique (CSG or boundary). Such is not the case.

6.10.4.8.6.3.2.1. Construction

The model *construction* sequence of a CSG modeler is: select/copy and size primitive shapes, orient them spatially as desired, and issue the modeler command(s) which will union, difference or intersect the primitives as desired. As a consequence of the commands, the Boolean operators and references to the primitives used are stored in a CSG *tree*. The CSG tree can be readily edited to change a model.

The model *construction* sequence of a boundary modeler is: define points, define lines/curves with those points, define surfaces with those lines and curves. Then issue commands that designate the surfaces as the boundary of a solid, or that *extrude*, *rotate* or *sweep* a defined face into a third dimension to obtain a solid. As a consequence of the commands, a graph of the topology and geometry is stored.

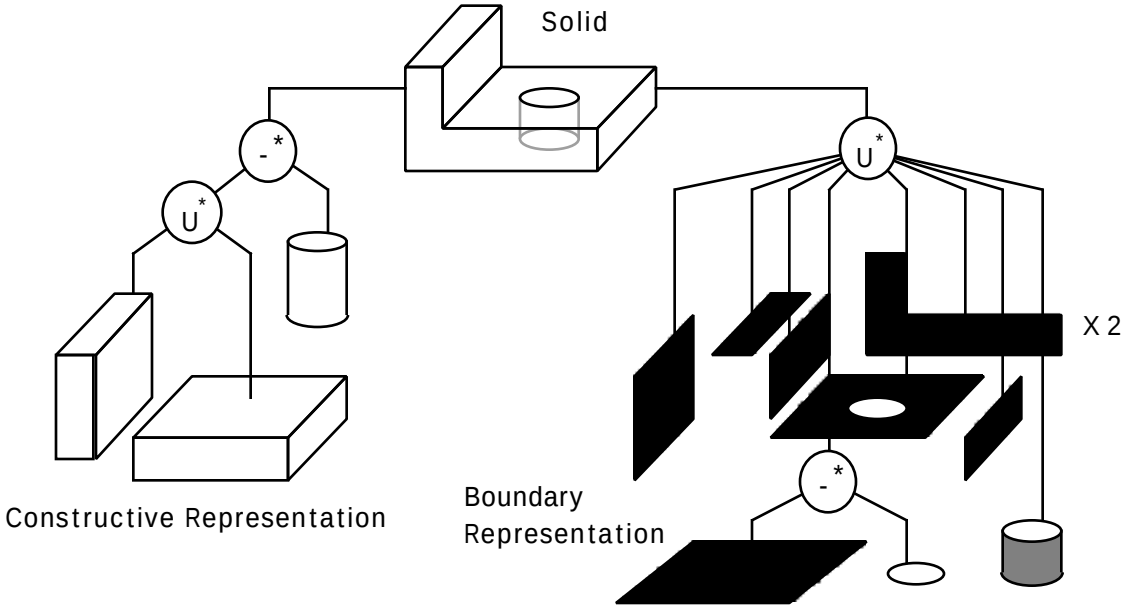
Some CSG modelers also allow a 2-D profile to be swept to define a volume that can be used as a primitive object in subsequent Boolean operations. Many boundary modelers allow CSG model construction techniques to be used as well. Only their representations differ.

Many modelers provide both construction options to minimize design time. Many maintain both the CSG for editing purposes and boundary representations for display and machining purposes. Some do it all.

6.10.4.8.6.3.2.2. Representation

The CSG *representation* is a hierarchy of *Boolean* operations (e.g., union, intersection, and difference) on *objects*. Objects are used to describe the results of combining primitive objects or *primitives*, like the blocks and cylinder shown in the lower left portion of the following diagram. Using Boolean operations, like union (U*) the blocks and difference (-*) the cylinder will result in the solid L-bracket object or part shown. With many CSG modelers, these objects (L-bracket) may be used in subsequent Boolean operations to form more complex intermediate objects, which eventually are combined to form a

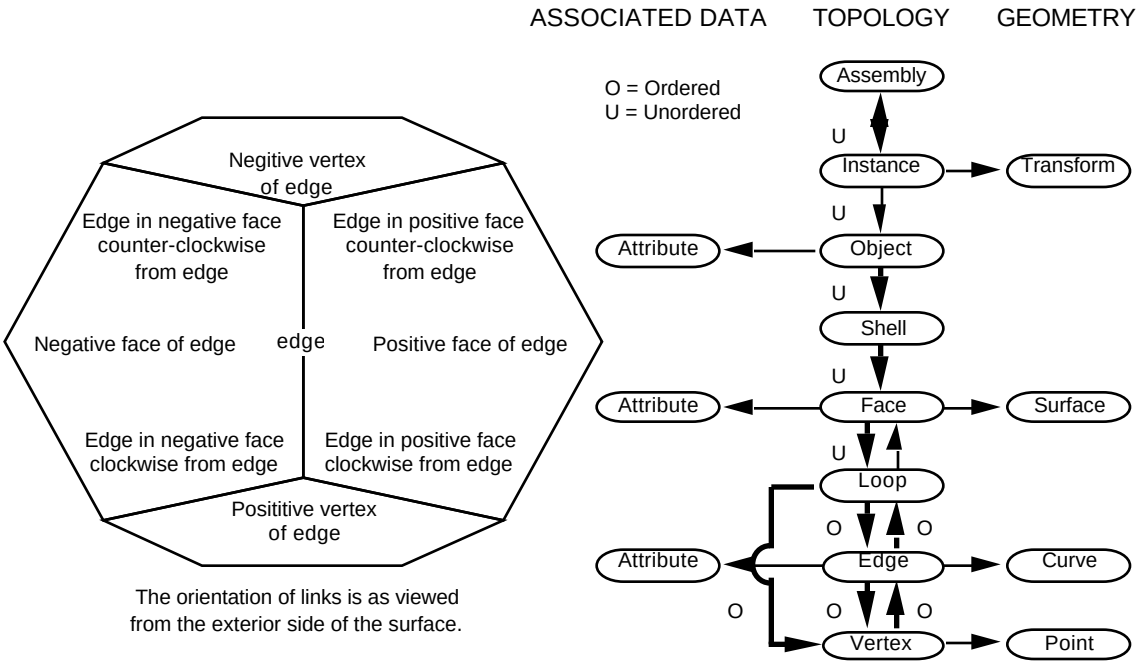
component part. Objects are the least common denominator of pure CSG modelers. Only the operations, the names of the primitives used and their parameters (instance size, relative position, orientation) are stored in a pure CSG or *constructive* representation.



Boundary modelers represent the same part by way of its external surfaces, or *boundary elements* as shown in the lower right portion of the foregoing diagram. Like objects, boundary elements can be unioned (U*) and differenced (- *) to define the part. Only the names and definitions of the boundary elements and their parameters (instance relative position, orientation) are stored in a boundary representation.

The boundary representation is a graph of vertices, edges, and faces, which represent the *topology* of boundary model. Each face is bounded by edges. The edges are bounded by vertices. The vertices are located by points. The *sense* (sequence of the line elements) of the face indicates the inward or outward direction of the face, and, hence, the inside or outside of a solid bounded by boundary elements.

The traditional Winged Edge Topology and a robust, virtual representation of the element relationships of a virtual boundary modeler are shown in the following diagrams.



"Hybrid" modelers extend the breakdown of parts to objects and then to the boundary elements that define the primitive objects. They may allow boundary elements to be defined independent of objects as well. These modelers may maintain or dispose of objects as intermediate geometric representations once the boundary model is derived. Those which retain this construction history as part of the representation can make model editing easier.

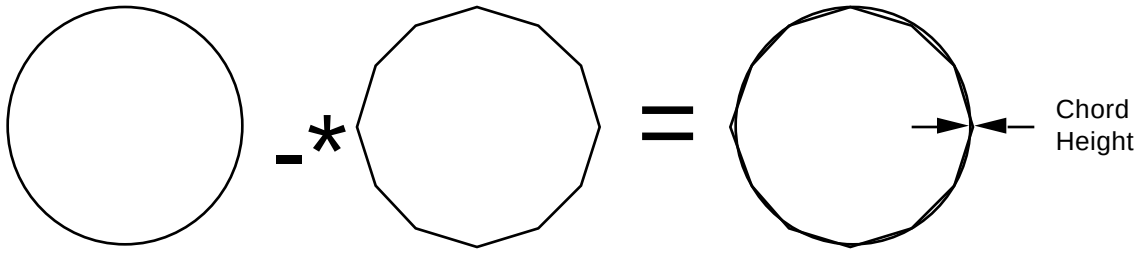
Both the CSG and boundary representations are unambiguous, but not unique (a feature or object can be created many ways). The CSG representation makes it easy to ensure that the part makes sense. It provides a concise representation for data storage and analysis. It allows the dimensions of primitive objects to be adjusted (tweaked) to the extent that the part still makes sense. The CSG representation makes it difficult to display line drawings, or have feature attributes like surface finish meaningfully associated with elements of the model (parts of a primitive may have different finishes).

The boundary representation makes it difficult to ensure validity (nonsense parts may be possible). It provides a verbose representation. It allows boundary elements to be tweaked to the extent that they preserve the topology. The boundary representation also makes it easy to display line drawings, associate tolerance information like surface finish with the elements of a model.

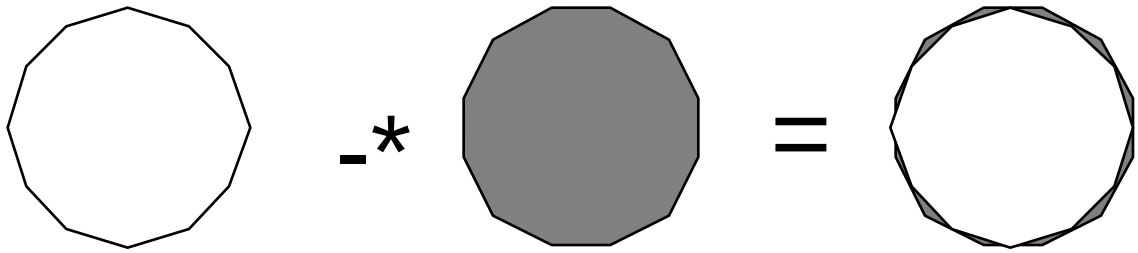
Some solid modelers are *faceted* modelers. They *represent* their curved faces or the curved surfaces of their primitives with many small polygons. Other modelers use analytically exact surfaces to represent all their faces or

primitives. Some resort to polygons for some but not all surfaces, primitives and Boolean operations. Many resort to polygons for *display* purposes only, because it is less compute intensive.

The use of polygon *representations* can have adverse consequences for fabrication and interference analysis if the number of facets are insufficiently numerous to approximate the intended surface for many applications. This is normally only a procedural problem, because most modelers allow user control over the number of facets. A common measure of the adequacy of the approximation is *chord height* or *crown tolerance*. Some modelers allow the number of facets to be specified as chord height. Unfortunately, some modelers do not allow the tolerance to differ among the geometric elements in a single model.



The real problem with facets arises when faceted primitives are differenced or intersected with one another. For example, when two circular areas are differenced from one another, the result is zero area (null set). When two polygons that approximate the same circular area, but with a different number of facets, or with the same number of facets rotated relative to one another, are differenced from one another, the result will be the intersection volumes of the polygons rather than the null set.



Such Boolean results will mislead interference analyses, generative NC and other applications.

Non-Uniform Rational B-spline (NURB) surfaces are popular for master dimension systems. NURBs allow the faces or surfaces of a solid model to be represented by a single surface type (plus topology), regardless of its complexity. This offers some advantage to those programming modelers to evaluate Boolean operations. However, the evaluation algorithms themselves, especially those dealing with coincidence and tangency can be very difficult to develop. Even the simplest objects, like a cylinder can also be represented by NURBs. NURBs are more computation intensive than the simple analytical

equation for a cylinder. Once a NURB evaluator is written that works, however, it works for all objects. It may be practical to implement it in firmware or even silicon. Then the processing inefficiencies related to simple objects become relatively meaningless.

6.10.4.8.6.3.2.3. Display

Graphic feedback of each construction operation is normally provided by the modeling tool. The *display* of the model is in itself a software tool. It is dependent upon the *representation*. A model defined by a boundary representation is easier to display than one defined by a CSG representation, because faces can be layered relative to the viewpoint of the observer and the obscured faces discarded for display purposes. With a CSG representation, both the "front" and "back" sides of the Booleaned primitives must be evaluated relative to the observer to determine what part of which primitive should be displayed.

6.10.4.8.6.3.2.4. Three Representations Are Required

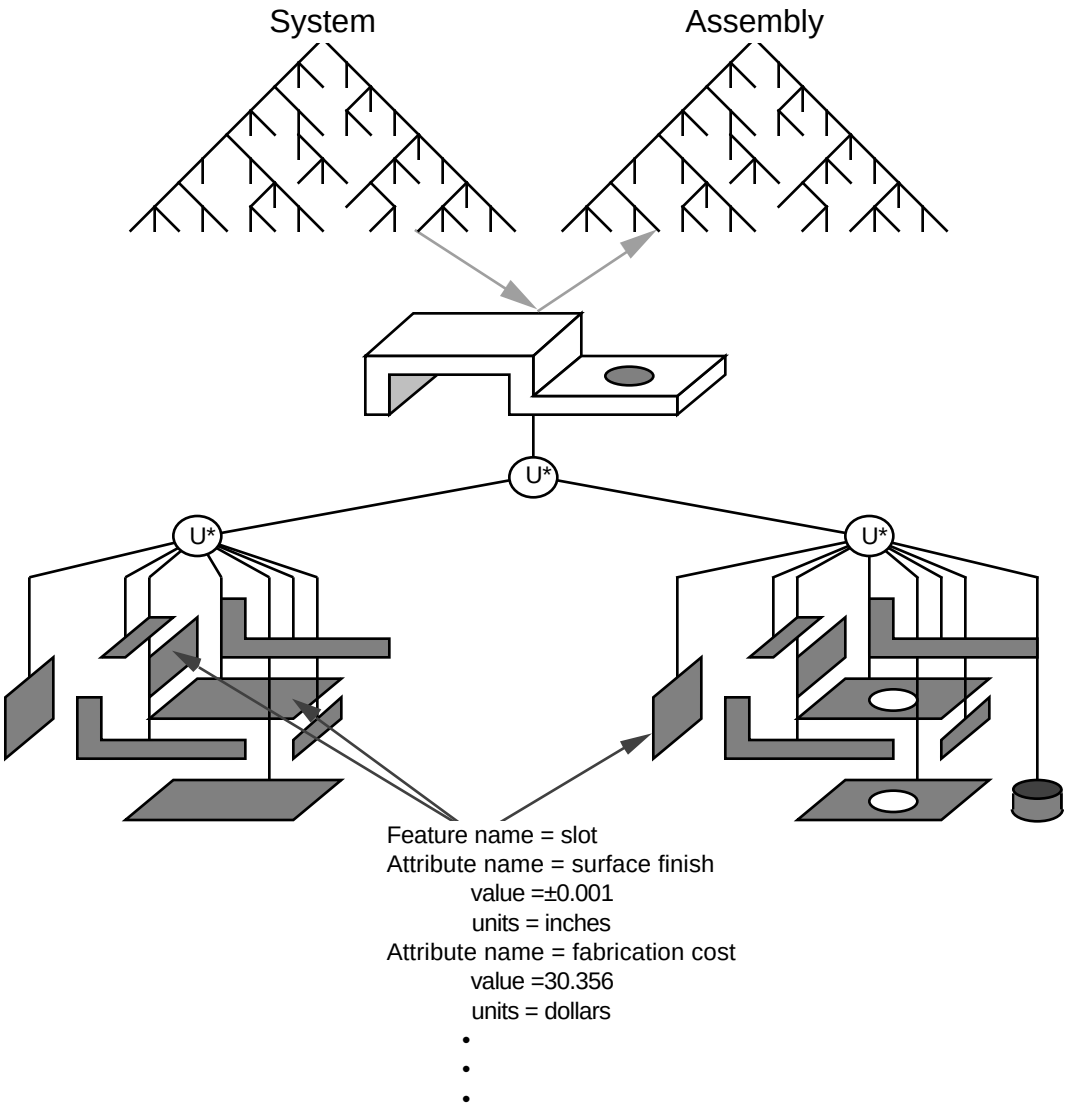
The constructive representation facilitates model editing. A polygonal boundary representation facilitates fast display. An exact boundary representation supports downstream uses of the model, like validation analyses and feature inference for generative process planning and numerical control programming. All three representations are required.

6.10.4.8.6.3.3. Features, Parts, Assemblies and Their Attributes

A product is assembled from major assemblies, assembled from minor assemblies, assembled from components. There are commonly seven intermediate levels of assembly from the perspective of a prime contractor or system integrator. Purchased items are considered to be component *parts*, when they are, in fact, *assemblies* (valves, potentiometers). In reality, there are many more levels in the assembly hierarchy than is typically managed by a prime contractor or system integrator.

In the context of this discussion, component *parts* cannot be disassembled without destroying the part. Component parts, like printed circuit boards or integrated circuits are not homogeneous, but once assembled they cannot be readily disassembled. By this definition, they are component parts. Everything else is an assembly.

The following diagram depicts the breakdown of a product into its constituent systems and subsystems, which breakdown to component parts, which assemble into assemblies. Parts can be bulkheads, brackets, printed circuit boards, resistors, paint, sealant or weldment. The parts consist of *objects* and/or boundary *elements*. The boundary *elements* may represent *features* individually or in groups.



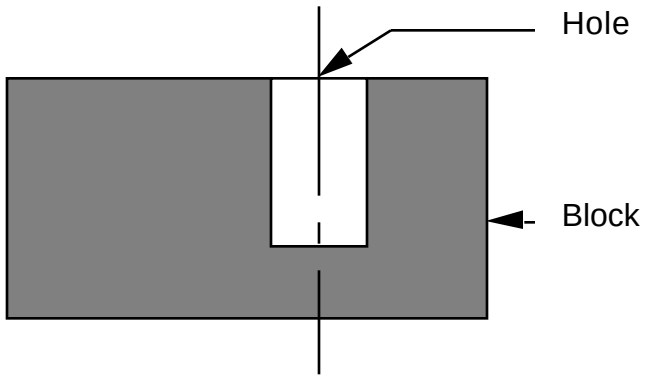
Constructive solid objects are not shown in this diagram because they cannot be consistently used to represent design features, let alone other feature types. Only boundary elements and groups of boundary elements can be used to describe all feature types.

6.10.4.8.6.3.3.1. Fabrication Features

Part *design* features include plates, holes, fillets, rounds, slots, webs and stiffeners. Some of these may also be *analysis* features. Pocket, fillet, round

and weldment are *fabrication* features. These feature types are related to one another by way of the boundary elements that are common to both. For example, the boundary elements of a pocket (manufacturing feature) are derived from the boundary elements of the plate and stiffeners (design features) that bound the pocket.

If a cylindrical CSG primitive is *differenced* (subtracted) from a block CSG primitive such that it penetrates only one side of the block, the result is something that looks like a hole. The result as far as a CSG representation is concerned, is a block minus a cylinder. If a circle is extruded into the block, the result is something that also looks like a hole. The result as far as a boundary representation is concerned, is a modification to the surface of the block to include a cylindrical surface enclosed with a circular plane at one end and open at the other end. What humans intuitively call a hole is identified as such by neither modeler.



The result of the differencing or extrusion operation must be specified by a "make hole" command/or identified by the human operator to be a "hole", or a computer program must analyze the model to make that inference. The feature that resulted from the operation may be given the *name* : "HOLE #1." The name can be a unique identifier of the feature or just one of its many *feature attributes*.

Features, parts, assemblies and products can have attributes associated with them. Typically an attribute consists of a name of the attribute (diameter tolerance), its value in this instance (±0.0003) and a unit of measure (inches). Reamer #45 is an example of a feature attribute without a unit of measure. An attribute can also be the name of a computer file or reference document source and its location. Attributes are categorized into classes or types, but such classification will not be discussed here.

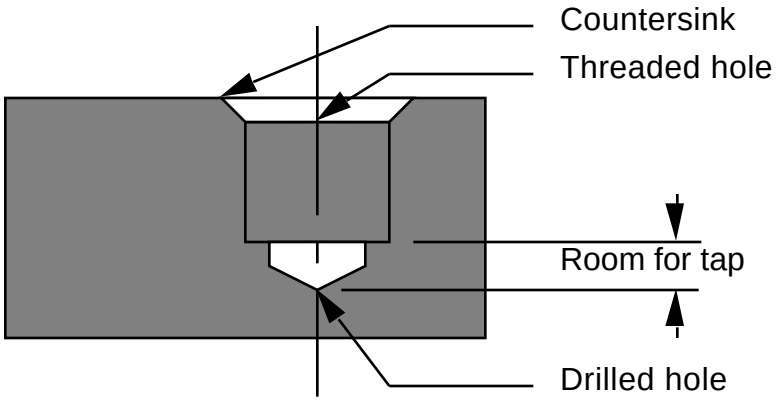
The geometry (positions of centerline and diameter points) that represents this feature that we now call HOLE #1 is accurately maintained by the modeler. Its dimensions can be computed and displayed. However, much more information must be associated with the feature to make it completely unambiguous for analysis, handling, fabrication, inspection, assembly, test and maintenance purposes. Diameter, length and perpendicularity tolerances are examples of

attributes that would be appropriate for this feature. Surface finish, location and parallelism tolerances are examples of attributes of a surface (feature) of the block or the hole.

If the hole is to be a threaded hole, a thread size or tap number must be associated with it as another feature attribute. Adding all the necessary feature attributes to each feature is similar to annotating a drawing, except that the data in a solid model is readily useful to downstream functions. (Ever try handing a drawing to a robot?) Annotation, whether on a drawing or with a modeler, is a lot of work.

An alternative is to define more sophisticated (complex, compound) features, like threaded holes. These could be differenced with the block as before, but all the information pertinent to a threaded hole would already be associated with it. It would not have to be added each time the hole is used in the definition of a product. It would be readily available in a library of standard features.

Furthermore, the geometry of the feature need not be a simple cylinder. It could consist of cones and a helix to represent accurately the end of the drilled hole, its tapered entry or countersink, and the threads as shown below. The more representative a feature is of the real thing, the tighter a design can be without problems like an unwanted penetration (threads of adjacent holes interfere). Regardless of its uses (instances), such a feature need be defined only once. These pre-defined feature, are to what current "feature modelers" are limited.



The bolt that goes into the hole could be similarly defined once, and used in many places. It would consist of additional objects to define the head and driver geometry (slot, Phillip's, hex, etc.). So why bother to define both a bolt and its hole? Why not define a bolt as the part it is, and simply difference it with the parts it intersects to make complementary holes? Sets of bolts could be used to make a hole pattern with one differencing operation.

The set of attributes that help define the bolt unambiguously for its analysis and manufacturing purposes must be distinguished from the set of attributes that help to define the hole that results from a differencing operation with a workpiece. Furthermore, the dimensions of the hole must be slightly larger to accommodate the bolt. The hole must be deeper than the bolt to accommodate

the tap with which the hole is threaded. The hole must include a conical end to accommodate the drill bit with which the hole is drilled. The solid modeler must be smart enough to use the attributes of the hole for a differencing operation rather than those of the corresponding bolt the user selected for the operation. Otherwise the user must specify which set of feature attributes to use.

This correspondence between a part and the geometry needed to mate the part with another part can be extended to all parts, like rivets, keys and key-ways. It can even be extended to include assemblies like bearing balls and inner and outer races and the hole into which the assembled bearing must be mounted. The mating geometry that corresponds to a part or a portion of an assembly is a form of arrangement criteria known as a *connection feature* described in the Arrangement Tool and Information Integration sections.

This is not quite like differencing a cylindrical primitive from a block, but the construction technique is the same. This technique can be extrapolated to include manufacturing features, like pockets, but in that context it limits the construction techniques available to designers. Sweeps, for example, could not be allowed, because there is no equivalent manufacturing feature or operation.

To solve a functional problem, designers think of webs and stiffeners, not the pockets of material that might be removed from a workpiece to make the part. The part may instead be made by fastening separately made webs to a plate, or it may be built-up from many layers of pre-cut composite material. There are many different manufacturing features that can correspond to a design feature. The actual fabrication method may be undefined the moment the design is conceived. Consequently, it is not wise to extrapolate this correspondence of constructive geometry with manufacturing features. This is the fallacy of what has come to be known as *feature design*.

6.10.4.8.6.3.3.2. Assembly Features

See Assembly Tool description.

6.10.4.8.6.3.4. Feature Design Versus Feature Recognition

The use of solid model data by another software tool is not as simple as many people believe. Many see a Boolean operation, like the subtraction of a cylinder from a block or the addition of two parts, and imagine fabrication and assembly operations. Sorry, there is no consistent correlation between a design operation and manufacturing features. Does the design operation "difference cylinder from block" correlate with the manufacturing operation "drill hole," "punch hole," "mill hole" or "drill and ream hole"?

To avoid some of these problems, feature design systems limit a designer to a few pre-defined the fabrication features (hole, pocket). Such limitations can be particularly frustrating for conceptual and preliminary designers. Even during detail design, designers think of function rather than manufacturing operations.

Structural designers think in terms of plates and stiffeners when designing a bulkhead. They do not think of pockets. If stiffeners were extruded from a pattern from a plate, a feature modeler would have stiffeners machined separately and welded to a plate. It would not recognize the pockets inherent in the design and have the panel milled from a block.

Fabrication features will seldom be compatible with analysis or process planning features. It is not enough to know how parts are oriented in an assembly. A robot must be told how to get them there without any collisions. Clearly, the features needed by each discipline to perform its function must be inferred from the design features.

In order for a software tool to make the necessary inferences, it must have access to the edges and faces which constitute the solid model. For this reason, a solid modeler must include an exact boundary representation. The polygonal boundary representation is inadequate for this purpose as described in the Representation section. A generative process planning or numerically controlled machine programming tool can interrogate the representation to infer the manufacturing features, and then infer an optimum sequence for their fabrication. An analysis tool can interrogate the exact boundary representation to assess the functionality of the model relative to its requirements. A collection of analysis tools (design rules checking) can interrogate the representation to assess how producible and maintainable the design will be.

A complex evaluation scheme is required for a computer to make these inferences from solid models. Some representations are common to more than one modeler due to the lineage of the modeler. However, the evaluation scheme is usually unique for each modeler, because the programmatic interfaces and command languages are unique for each modeler. They also do not provide the high-level commands the software tool developers would like.

6.10.4.8.6.3.5. Standard Tools Interface for Solid Modelers

A standard tools interface has been specified to alleviate the difficulty of developing tools that use the data generated by solid modelers: Volume I - Introduction and Rationale and Volume II - FORTRAN procedures of the Computer-Aided Manufacturing International (CAM-I) Applications Interface Specification (AIS), R-86-GM-01.1, CAM-I, 611 Ryan Plaza Drive , Suite 1107, Arlington, Texas 76011, 817-860-1654. The AIS currently only deals with geometry and topology. It does not encompass dimensioning, tolerancing or features.

An interface that is compliant with a complete AIS will not only make software tool development easier, but also provide a degree of independence from any one modeler. If all the purchased or developed software tools use an AIS compliant interface instead of interfacing directly with a modeler, the modeler can theoretically be replaced without having to modify any of the tools.

The AIS puts the burden on the solid modeler vendors to provide a consistent software tool interface. Like the Initial Graphics Exchange Specification (IGES), the AIS will not become a viable standard unless all the potential buyers of solid modelers announce that they will buy only those modelers that comply with the AIS. (Extensions to IRDS to support solid models, dimensioning and tolerancing based on ATIS may be a more viable standard.)

6.10.4.8.6.4. Assembly Tool

Currently, a designer must specify the position and orientation of each component in an assembly to make them appear to fit together in the most compact arrangement possible. During this process, the designer must consider ways in which to secure the parts to each other and to the structural system while minimizing tubing and wire harness lengths, maintaining the required mass distribution, keeping replaceable parts accessible and a host of other criteria. The designer must measure the distances between mating surfaces of the parts to ensure that they are properly aligned (parallel or tapered offset, etc.). In some solid modelers selected geometry can be aligned relative to one another, but this is still a tedious process.

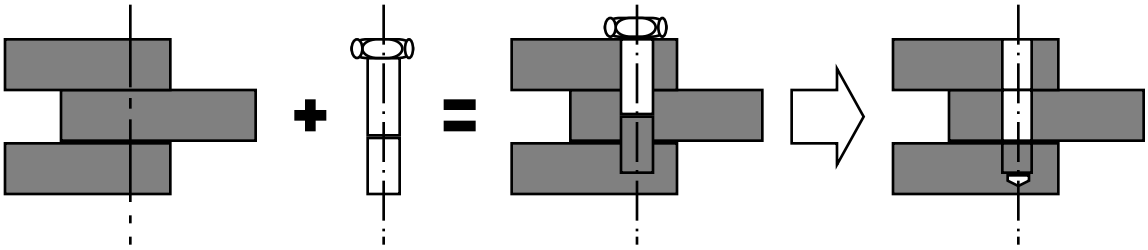
To reduce the scope of the arrangement problem and perhaps reduce the number of variables with which a designer must concurrently contend, major systems are often allocated to a volume of the product (forebody, mid-section). Although this approach may make the job easier or even possible, it often results in excess volume in some portions of a product and insufficient volume in other portions. To contend with insufficient volume, designers resort to more exotic and expensive system or assembly solutions.

Until an automated arrangement tool is available, the manual arrangement process could be improved with the introduction of assembly features and assembly commands. The assembly tool should allow a designer to select boundary elements of a part and group them as an assembly feature of that part. The process would be repeated for all the assembly features of the part and every other part in an assembly. Then the designer should be able to select an assembly feature on one part (key, bolt shaft), issue an assembly command like *insert* and select an assembly feature on another part (key-way, hole). The assembly tool should then compute the rotation and translation necessary to move and position the part such that the one assembly feature is inserted into the other assembly feature as indicated. *Place against/offset* would be another assembly command.

Tolerances can be inferred from the assembly command and the geometry. Tolerances for each assembly feature should default to a standard appropriate for the circumstance and the material. However, the designer should be able to override the defaults and specify the tolerance. Interference and tolerance analysis tools could be an integral part of an assembly tool to provide immediate feedback to the designer, or they could be separate tools run after an assembly has been created. In the interest of performance, the latter is the

preferred case: use nominal dimensions to create assemblies until an optimal assembly is selected, then assign tolerances to the assembly features and use interference and tolerance analysis tools to identify the geometry or tolerances that must be changed.

The assembly tool should reduce the time a designer spends finding, selecting and modeling standard parts and specifying their corresponding manufacturing features. A designer should only need to specify an assembly, like the three plates shown below.



The designer could use a *part selection tool* to specify part characteristics, like those of a bolt fastener. Ask the part selection tool to display candidate bolts. Select a bolt, and specify the bolt diameter, but not the shaft or thread length as shown.

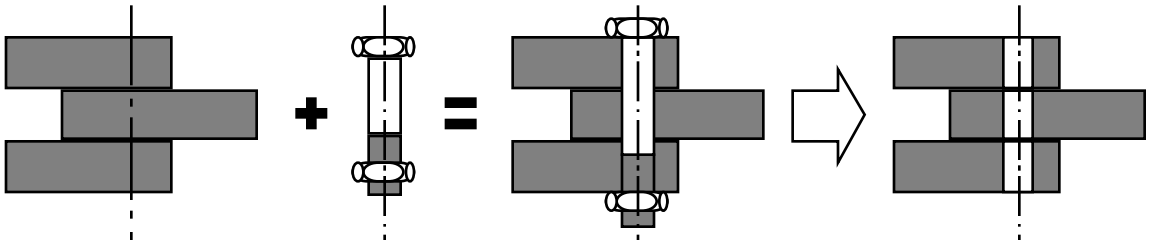
The designer would use the assembly tool to *insert* the bolt in the plate assembly with a certain *fit* (lose, tight) *such that* it does not penetrate the lower plate. The lower plate is specified as an attachment. The assembly tool should use the parametric description of the bolt available from a *part selection tool* to select a standard bolt length that will not penetrate the lower plate and a standard thread length that will match or exceed the thickness of the lower plate. The part selection tool should revise the bolt geometry (based on its parametric solid model of the bolt) and supply a name (part number) and associated part attributes to the assembly tool. The assembly tool would then display the solid model of the standard bolt in the assembly.

A design rules checking tool could be invoked by the assembly tool to assure that the diameter of the bolt selected by the designer is sufficient for the worst case shear loads. The bearing area could be determined by the design rules checking tool from the solid model. The shear strength of the plates and the bolt could be determined from a standard parts and materials library. The tension on the bolt could be determined from its torque specifications. The design rules checking tool could then invoke a structural validation tool to determine the shear loads. Then the design rules checking tool could ascertain the viability of the design. A similar validation could be performed for edge thickness concerns. Fatigue could similarly be evaluated using a structural dynamic validation tool.

The part selection tool should associate the parametric manufacturing features that correspond to the bolt with the bolt. The assembly tool should insert (difference) the two instances of the corresponding hole manufacturing feature

and the one instance of the corresponding threaded hole manufacturing feature into the plate models. When the bolt is disassembled from the plate assembly, the hole and threaded hole features should remain behind with their respective plate models as shown.

Similarly, the designer may specify a fastener assembly like a bolt and nut as shown.



The designer could ask the part selection tool to present candidate bolts, select a candidate bolt, specify the bolt diameter, but does not specify shaft or thread length as shown.

Use the assembly tool to *insert* the bolt in the plate assembly with the specified *fit*, *such that* it penetrates the lower plate with sufficient length to allow the corresponding nut to be used to secure the assembly (plus three threads beyond the nut). The assembly tool should use the part selection tool as described above to select an appropriate bolt. The part selection tool should revise the bolt information, so the assembly tool will accurately display the standard bolt and nut in the assembly as shown.

In this case, three instances of the corresponding hole manufacturing feature must be inserted into the plate models. When the nut and bolt are disassembled from the plate assembly, the hole features should remain behind with their respective plate models as shown.

Manufacturing features associated with parts could instead be associated with their assemblies. For example, a manufacturing engineer may decide that the tolerance requirements could better be met by drilling through the three plates while they are in their assembled positions rather than drill the plates separately. The holes were designed as features of the parts. The manufacturing engineer need only notify the responsible design engineer of the needed change, and the designer (or the manufacturing engineer if given the authority by the responsible design engineer) can simply use a solid modeling tool to edit the CBS (CSG tree) to break the links between the hole features and their plates, and establish a link between the hole features and the assembly of the three plates.

The designer should be able to override the linear translation and rotation of a part into its assembled position as calculated by the assembly tool. The designer should be able to specify a trajectory that avoids other parts or fixtures

during assembly. This trajectory could then be used by a robot to assemble the part, or remove it for maintenance purposes.

6.10.4.8.6.5. Composites Design Tool

The ability to tailor the stress and strain characteristics of a structure using composites and more exotic materials has important benefits for product design. However, it presents special problems for a solid modeler. With composite materials, the modeler must deal with many parts with uniform thickness and complex shapes assembled (bonded) into a single part. The fiber orientation of each layer usually varies from layer to layer, so the tensile, bending and compressive strength of each part is directional. Having certain fiber orientations at certain levels in the assembly will cause the assembly to deform in predictable and beneficial ways to known loads (flexibility tailoring of aircraft wings - bend down and twist negative under load; see Structural Validation Tool section). To take advantage of this, the composites design tool of a solid modeler must retain the unique identity (fiber orientation) of each layer (part) of a composite part (assembly) for structural analysis tools to use.

In the composites design process, the overall part shape is defined according to external constraints (connections, weight, inertia). Then the number of layers and the fiber orientation necessary to achieve the desired strength, behavior and shape is defined. This may result in a modification of the part shape. A composites design tool should support this design process. It should generate the numerical control program required to direct the movement of an end-effector to cut (laser, water jet, oscillating knife, router) each ply of the composite material.

6.10.4.8.6.6. Flat Pattern Design Tool

This software tool should automatically generate flat patterns for deformed composite and sheet metal (hydro-pressed) as well as simple bent planar parts. Pattern offsets must be a function of the material and the geometry of the part. For change control purposes, the flat pattern should be automatically associated with the model from which it was derived. In addition to fiber orientation, the derivation must consider fiber re-alignment during hydro-pressing.

6.10.4.8.6.7. Arrangement Tool

Automated two-dimensional arrangement has long been available for the placement of electrical components and the routing of their connections (copper traces). With such a tool, the copper and printed circuit board area are minimized. Hot or radiating components are placed away from components that cannot tolerate heat or radiation. Automated three-dimensional arrangement will be possible with solid model data, eliminating the need to arrange component models manually.

The system definitions available from the System Breakdown Structure (SBS) described in the Information Integration section indicate the general connectivity of systems, subsystems and their components. Once the system definitions have been driven to the detail of component parts, and their specific connection features (see solid modeling tool section) are established, the three-dimensional component models can be automatically rotated and positioned. The router would strive to minimize the lengths and bends of connecting components (wires, tubes).

Wire harnesses and tube runs need not be specified. Only the connectors and (pin) connectivity need be specified. The wire harnesses and tubes can be defined as a result of the arrangement operation, which would iteratively invoke an automatic routing tool until the arrangement achieved the desired result.

The arrangement criteria could include weight and volume minimization, balance (center of gravity) and survivability (ballistic strike protection) requirements. The weight and volume minimization criteria would require that the lengths of connecting components (wires, tubes) be minimized. Wire and tube lengths must also be minimized to minimize electrical and fluid resistance. Tubing bends should be minimized to minimize the energy required to push fluids through the tubes. Pipes would have routing precedence over tubing and tubing should have precedence over wire harness, if fabrication cost minimization were a criteria. The less reliable components would migrate toward the outside of the product, particularly toward access doors if maintenance costs were the criteria.

The arrangement constraints could include external and internal surfaces (mold lines) and physical exclusion (no two parts can occupy the same space at the same time). Kinematics (mechanical linkage, force and acceleration requirements) must be preserved. Volumes swept by mechanical linkages must not violate the volumes of other parts. The attachment points and connectors of off-the-shelf (existing) parts and assemblies (connection features) must be used.

Although the arrangement tool is automated and the assembly tool is not, the tolerance discussion in the assembly tool section applies to the arrangement tool as well.

Only after the component parts have been optimally arranged should their assembly sequence (Assembly Breakdown Structure) be defined. Only then should the attachment points (assembly features) be defined for those component parts for which attachment points could be left undefined during the arrangement process. If acceptable attachments cannot later be defined for some parts, then acceptable but perhaps interfering attachments must be defined. Then the three-dimensional arrangement program must be re-run to force the arrangement to accommodate practical component attachments. As a result of interference and tolerance analyses, some component attachments may have to be modified to take advantage of more optimum arrangements.

Three-dimensional arrangement is an iterative process. It should be done periodically throughout the product definition process. The more often it is done early in design, the less likely major problems will arise during later activities, and the more likely the product volume will be minimized. The use of connectivity features will allow the arrangement tool to arrange even abstract representations of components or systems. As components are defined to a detail sufficient for their unambiguous manufacture, connection features will become assembly features or at least correlate with them.

As excess volume appears during the arrangement process, the external surface definition should be shrunk, and the arrangement tool re-run to see if all the components can still be made to fit into the smaller volume. If so, the structural, fuel and other systems can be reduced proportionally, saving weight and material cost. If not, then the volume constraints must be relieved, or the external (wetted) surface violated. This iterative process could be automated by specifying volume reduction criteria, like "preserve wetted surface proportions" (same shape but smaller).

6.10.4.8.6.8. Automatic Interference Checking Tool

An automatic three-dimensional interference checking tool could warn a designer when parts interfere during or after a manual assembly process. It would evaluate the product geometry as defined, and identify all part intersections (intersection flashes or assumes a different color). Such a tool should be an integral part of an arrangement tool.

6.10.4.8.6.9. Tolerance Analysis Tool

The arrangement, routing and interference checking tools described use nominal tolerances when spatial relationships are defined. A tolerance analysis tool should apply various permutations and combinations of individual part feature tolerances to the packaged product (or a portion thereof). It should re-invoke the interference checking software tool to determine the effect of the tolerances in each situation. Where no interference problems exist, the tolerance analysis tool should try relaxing the feature tolerances until an interference problem is detected. It should then identify what tolerances could be relieved to reduce the cost of fabrication and assembly. The tolerance analysis tool should iterate with the arrangement and interference checking tools to optimize the arrangement without tightening the tolerance requirements. It should indicate what tolerances should be tightened to avoid a tolerance build-up problem.

6.10.4.8.6.10. Tolerance Reality Check Tool

As fabrication processes are defined with a manufacturing process planning tool, the fabrication processes are associated with fabrication features. Where the boundary elements of fabrication features coincide with the boundary elements of assembly features, a *tolerance reality check* can be performed. If the manufacturing process chosen for a fabrication feature is such that the

tolerance assigned to an assembly feature boundary element is unrealistic, this should be identified by the tolerance reality check tool. Then the tolerance associated with the assembly feature must change or the fabrication process must be changed.

6.10.4.8.6.11. Assembly Simulation (Electronic Mockup) Tool

This software tool should animate the assembly, removal and replacement of parts and assemblies in a packaged product (with the movement of any mechanisms defined with the kinematics software tool). It should optionally include the movement of foreign objects (tools, end-effectors, hands, etc.) within the assembly and highlight collisions.

6.10.4.8.6.12. Design Validation Tools

Product performance, reliability and maintenance requirements can be satisfied in many ways. Each of those solutions may involve many different design features. Each design feature corresponds by way of its boundary elements to one or more manufacturing features or a portion of a manufacturing feature. Each manufacturing feature has unique producibility (material handling, fabrication, inspection, assembly and test) requirements associated with it. Each of them is impacted by the materials involved. The net effect of these implications must be determined to select the optimum design solution.

Only gross qualifiers like weight, cost, status and quantity can be associated with parts. Even part cost can be appropriately associated with a part only if the part is to be purchased "off-the-shelf." The cost of a part to be manufactured must be determined from the accumulated cost of the material and each operation performed on it, including tool wear.

It costs money and time to buy the bulk material. It costs more money and time to cut or forge a workpiece from the bulk material. It costs money to store the material and workpiece, and move them about the factory. It costs money to buy or develop tooling for the workpiece or part. It costs time and money to set-up the workpiece for fabrication and inspection and for assembly and test. These costs can be associated with the part by way of its manufacturing features and the process and resources used to create them. Each process operation uses skills for the duration of that operation, which relate to the employment of valued resources for a time, which equate to cost and schedule.

Tools and fixtures (tooling) are among the resources described in the Resource section. Like the other resources, they have unique skill sets. The skill sets of some tooling are such that that it is used only to manufacture the instances of one part. Other tooling may be used on more than one part. Both are deliverables. More accurately, a feature of the tooling is related to a feature of a part. The cost of acquiring, refurbishing or replacing that tooling should be

distributed among as many instances of one or more features as possible to reduce the effective cost of that tooling.

This is but one example of the costs that can be cumulated up the System and Assembly Breakdown Structures to determine the cost of the product, or the cost of any subsystem or subassembly. The impact of each design feature on the overall cost of the product can thereby be assessed.

Not all design features or their corresponding manufacturing features are known when design decisions must be made. Significant commitments must be made when system designs are largely abstractions of the product definition before it is sufficiently complete for production purposes. In this case the system requirements and attributes can be used to find similar existing systems for which there is manufacturing, maintenance and cost data. Estimates of the cost of each attribute of a new design can be interpolated between or extrapolated from that experience for design optimization purposes.

An alternative is averaging the cost of part classes. To provide cost feedback for those involved in the preliminary and conceptual phases of design, the costs associated with a class of features like bolt holes, can be averaged to establish a cost for that class of features for preliminary design purposes. The costs of all fastener types must be similarly averaged to derive a cost appropriate for the conceptual phase. Adhesives would be included in even a broader class of fastening devices. These costs would be attributes of comparably abstract features used to define the product during each phase of the design. As the design becomes more detailed and component parts are designed, then the experience with individual manufacturing and maintenance features can be used to make good design decisions.

It is one thing to validate a design by way of simulation or test. It is another to use validation tools to determine which among the many design versions is the optimum design. To this end a suite of validations tools must be run on each version of the product.

With a parametric product definition, the validation tools could run iteratively, creating their own versions of the product design. They could iterate toward an optimum solution for the product. They could employ a solid modeler to determine the arrangement and its inertial effects. However, requirements often conflict. The desire to make a product lighter to improve its performance conflicts with the desire to retain its strength.

A quality design performs as required and is reliable, producible and maintainable. Performance, producibility, reliability and maintainability each have a cost associated with them. There are large costs associated with failing to meet a performance requirement. There are costs associated with over-riding government or enterprise or Program part standards. Schedule also has a cost associated with it, be it a function of progress payments or a product "window of opportunity," or customer satisfaction. In order for a designer or

optimization tool to trade design options, a single measure of merit must be provided as feedback. That measure is cost.

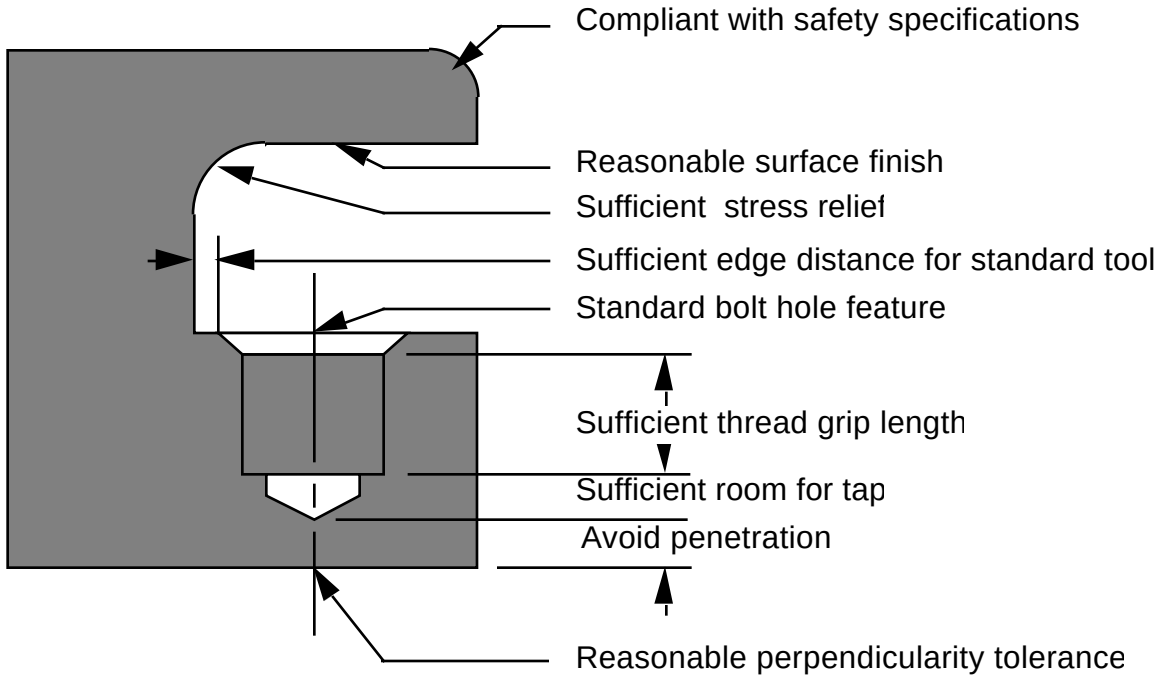
The following validation tools are a few examples of those which may be used to determine if a design meets its requirements. While the cost of design versions provide a measure of merit for trade studies, validation tools identify the designs that meet or exceed their requirements.

6.10.4.8.6.12.1 Design Rules Checking Tool

Design rules checking is a phrase that was once limited to informing a designer that a non-standard part or tool was specified on a drawing. With the advent of solid modelers, the concept has expanded to include producibility criteria (corner radii too small, edge distance too little) and maintainability (special tool required to remove part). These criteria can only be usefully associated with features. Their common measure is cost.

Solid models expand the practical domain of design rules checking. The minimum costs and cost multipliers as a function of adjacent features need only be maintained at the feature level. The ability to inherit costs up the product structure is inherent in the data structure of many solid modelers. However, component costs are more accessible as part of the System and Assembly Breakdown Structures maintained by the Product Data Manager. Hence, part costs should be cumulated from their feature costs and maintained as an attribute of the component.

The minimum cost of an isolated feature is the same for all of its occurrences if the same material is involved. Feature costs increase when the operations needed to make or inspect a part or test an assembly are complicated by the other features of a part or those of the parts adjacent to it. The following figure is an example of an unnecessarily costly part.



The fabrication cost would be reduced if the threaded hole could go through the "bottom", and be drilled and threaded from the "bottom". If not, perhaps the obstructing flange could be removed, added later or drilled to provide sufficient access to make the threaded hole from the "top" of the part. Each of these process costs increase with material hardness or abrasiveness. Each increase in cost as the tolerance requirements become more severe.

Unless an access hole is provided in the overhanging flange, the assembly and maintenance costs associated with this part are higher than they need be. A special tool (bent driver) must be tediously employed to install or remove the bolt that goes into the hole.

A design rules checking tool should enforce user-defined rules on model geometry and attribute values. In a batch environment, it may invoke sophisticated analyses to determine the appropriate geometric constraints or attribute values for the part, system or assembly model being checked. It should use the attribute values of the deliverable being checked to determine which analyses should be invoked.

In an interactive environment, a design rules checking tool would rely on relatively simple range or value constraints (part number attribute value matches that of an approved part, fillet radii should be one of a set of radii which correlate to standard cutters, etc.). The interactive user would tell the design

rules checking tool which analyses to invoke. The design rules checking tool should use the attribute values of the deliverable being checked to determine whether the requested analysis is possible, and notify the user accordingly. When a simple constraint check involves a search of large databases, it may be delegated to the batch environment. The tool should facilitate such performance tuning if an on-line environment is to be properly supported.

Since the Process Manager allows subtasks to be defined on-line, and a subtask can involve the invocation of a validation tool, there is no need for an interactive Design Rules Checking Tool per se. Similarly, any batch checking should be keyed to one or more subtasks in the Task Breakdown Structure (Information Architecture) managed by the Process Manager. The Process Manager would trigger the validation tool the moment the conditions for the invocation of that subtask (preceding subtask complete, model is not likely to change, scheduled start time, etc.) are met. The ability to associate all the analysis programs that can be executed in batch to such subtasks and have the Process Manager invoke them eliminates the need for a separate batch design rules checking tool as well.

Consequently, the role of a design rules checking tool is reduced to that of a tool that determines what validation subtasks are appropriate for a System, component or assembly. That tool is called the Validation Selection Tool.

6.10.4.8.6.12.2. Validation Selection Tool

Designers commonly determine which analysis and test functions should be involved in the validation of a design. Depending on the experience of the responsible engineer and the circumstances, insufficient or inappropriate analyses and tests may be conducted. A conservative engineer may request more numerous and more sophisticated and expensive analyses and tests than are necessary. The same engineer under schedule pressure may do the opposite.

To assure that the analyses and tests that are necessary to validate the design are conducted, and no more, a validation selection tool is desirable. This tool could use system requirements and system, part and assembly attributes to determine which analyses or tests should be performed to validate that a deliverable does, in fact, meet its requirements.

Once the necessary validations are determined, this tool could invoke the Process Manager and establish subtasks for each validation. It may even use deliverable dependencies among the subtasks to specify subtask sequencing. Initially, this tool would be invoked from within the Process Manager by responsible engineers who are delineating the subtasks for the task for which they are responsible.

6.10.4.8.6.12.3. Mass (Volume) Properties Analysis Tool

This tool calculates the surface area, area moments of inertia, volume and volume moments of inertia of a solid model. It stores the results so that section properties and mass properties can be obtained for other tools. If a material density value is available by way of the value of a material attribute, then the mass and mass moments of inertial of the model can be computed. If the value for the acceleration due to gravity is also available, then the weight of the model can be calculated as well.

6.10.4.8.6.12.4. Mesh Generation Tool

A mesh generator should automatically generate a mesh from a solid or surface model. The generated finite elements should be compatible with structural dynamic, thermodynamic and fluid dynamic validation tools or be the basis for their derivation. This would eliminate the need to have and to train human resources to use the many mesh generators that are peculiar to each validation tool.

6.10.4.8.6.12.5. Static and Dynamic Structural Validation Tool

NASTRAN, SUPERTAB, PATRAN, MARC, ANSYS and Strain Generation are popular structural analysis tools. Finite element analysis tool capabilities should include linear and non-linear static, dynamic steady state, random and frequency response and transient behavior analysis for heat transfer and fluid as well as structural problems. It should support design optimization. It should determine plane stress and plane strain with symmetric or unsymetric loading.

Finite element modeling elements types should include isoparametric, linear, parabolic and cubic. General shells, laminated composite or sandwich shells, thick shells, solid beams, spars, springs, mass elements and rigid elements should be also be supported.

Given the mass properties of a structural design and a specific load regime, the deformation of the structure can be calculated and used to determine its effect on the loads. The iterative invocation of validation tool sets is exemplified in more detail in the Aerodynamic Validation Tool section.

6.10.4.8.6.12.6. Mechanics Validation Tool

Mechanics validation tools should use the solid or abstract models of mechanical systems, their connection features, material attributes and mass distribution to determine the behavior of mechanical systems. ADAM is a popular mechanics validation tool.

This software tool should allow one or more sets of at least ten elements each to be interconnected and interdependently moved according to user defined constraints. The constraints should include range and degree of motion relative

to mechanical connection features that may connect with structural or other mechanical connection features. Six degrees of freedom are possible at each connection. Each element may be defined as a solid model or represented by simple abstractions like those used in mechanical schematics. The element definitions may be mixed. Hence, the element definitions may or may not include the volumetric information necessary for collision detection or arrangement purposes. Inertias and stress/strain relationships may be associated with linkage elements with or without volumetric information associated with them. This would allow mechanical system inertia, deflections and excess stresses to be determined early in the design process without the need for a solid model of the mechanical system.

Model motion should be definable by time as well as position. The tool should display the mechanism at any user selected time or position. I should calculate joint displacements, velocities and accelerations on demand. Given mass, it should also calculate joint forces. A color change should indicate when and where these values exceed allowable limits.

Where volumetric information is available, trajectory volumes should be computed for collision detection purposes. The offending link should change colors while it interferes with another part. The kinematic validation tool should optionally union the trajectory volumes to determine the total volume swept by the linkage elements in motion (envelope). Then it can be saved and used as a representation of an assembly of the mechanical system for arrangement purposes.

Where material property information is available, the kinematic validation tool should optionally compute the deflection of each linkage element and the net effect of all linkage element deflections on the mechanical assembly under specified loading conditions.

6.10.4.8.6.12.7. Thermodynamics Validation Tool

Thermodynamics validation tools should use the assembly models (spatial relationships) of a product, the attributes of their components and their connectivity to represent the product or its parts for thermodynamic analyses purposes. The mesh provided by a mesh generation tool may be adequate for most thermodynamic radiation or convection analysis purposes. Sinda and Thermal Networking are popular thermodynamic analysis tools.

6.10.4.8.6.12.8. Aerodynamics Validation Tool

Gross abstractions of vehicles are used during conceptual design. Surface models (lines/loft) are usually used during preliminary and detail design. With the advent of solid models, aerodynamics validation tools should use the wetted portions of the solid models of a product to represent the product for aerodynamic (lift, drag, pressure, loads) validation purposes during detail design. The mesh provided by a mesh generation tool may be adequate for most purposes. VSAero is a popular tool for determining aerodynamic loads.

Aerodynamics, structures and mass properties validation tools should work in harmony to facilitate design optimization for an entire mission rather than for a few discrete parts thereof. Given the mission profile and a structural design, the aerodynamics and mass properties validation tools can be executed to provide loads and inertias to the static and dynamic structural validation tools. The structural validation tools would then determine the deflection of the structure and its dynamic behavior. The aerodynamics and mass properties validation tools would then be re-executed to determine the air loads of the deformed structure and so forth until the rate of change of air loads due to structural change declined to a tolerable level. This process could be automatically repeated by a Process Manager to provide data at intervals throughout the mission to determine where is the maxima of the structural load regime. The process could be repeated for smaller intervals in those regimes until the true maxima is determined.

If the structure is made of composite materials, then the interaction of these three validation tools can be used to determine the composite fiber orientation needed to achieve aerodynamically desirable wing bend, twist and camber under load.

6.10.4.8.6.12.9. Stability Validation Tool

Static and dynamic stability validation tools should use the results from mass properties (balance, inertias), aerodynamic (air loads), propulsion (thrust vector, torque), control system (control rate and amplitude) and static and dynamic structural (twist, bending, dampening) validation tools to model the product for stability analysis purposes.

6.10.4.8.6.12.10. Signature Validation Tool

Gross abstractions of vehicles are used during conceptual design. Surface models (lines/loft) are usually used during preliminary and detail design. Signature validation tools should use the solid or surface models of a product and their connections and material attributes to represent the product for signature analyses (microwave, optical, acoustical) purposes. The mesh provided by a mesh generation tool may be adequate for most purposes. MISCAT is a popular radiation signature analysis tool.

6.10.4.8.6.12.11. Reliability Validation Tool

Obviously, the more reliable a system or component is, the more available (ready to perform mission) and useful the product will be, and the less likely that costly maintenance will have to be performed. To minimize input data development, the data generated from the finite element modeling and analysis subtasks and the design models should be used by a tool like Reliability And Maintainability using Computer Aided Design (RAMCAD) to generate data relevant to life estimates. Failure mode characterization using RAMCAD can be a significant input to maintainability and supportability analyses.

Designing circuits to perform at maximum ratings is considered poor design practice under most circumstances. Conservative designers usually de-rate a device to fractions of the maximum values. If de-rating is desired, all values are multiplied by their associated de-rating factors. The de-rating factors are obtained from a de-rating specification. Each contractor creates and maintains a separate de-rating policy. However, Program specific de-rating guidelines do not always cover the full spectrum of electronic components. The knowledge required to properly de-rate components for which there are no guidelines is the intellectual property of a few experts. This makes it a candidate for a rule-based expert system. The de-rated parameters derived by the expert system should be related to the corresponding subsystems or components in the SBS so that they can be used for the validation of those subsystems or components.

6.10.4.8.6.12.12. Producibility Validation Tool

A producibility validation tool need only invoke a generative process planning tool and get a positive result to determine if a part or assembly is producible. The generative process planning tool would determine the skills necessary to fabricate and inspect a part or assemble and test an assembly. The producibility validation tool would then invoke the Resource Manager to determine if resources with those skills will be available when they are needed.

More important are estimates of the production cost of alternate designs. If a resource set is selected from the Resource Manager and established as a constraint on the generative process planning tool, then the manufacturing cost of a design can be calculated by invoking the generative process planning tool. It would determine the time of each resource required to perform the operations necessary to manufacture the item. Multiplying those times by the resource cost ($/hour) would result in the cost to produce a design. Doing this for each design variant will identify which is the most producible.

Until there is a generative processing tool, a rule-based producibility validation tool must be used. Such a tool would notify a designer when fabrication features (holes, fillets) require the use of non-standard tools, when bend radii is tool small to avoid cracking, when holes are too close to an edge, etc. (see Design Rules Checking section).

6.10.4.8.6.12.13. Maintainability Validation Tool

Maintenance may involve any collection of subtasks from replenishing fluids to major disassembly, repair, replacement and re-assembly subtasks. Maintenance subtasks may have to be performed by resources with limited skills anywhere and under any conditions from the relatively benign office environment to the manufacturing floor to the north pole or under water. Consequently, no special resources (tools, humans) should be required to perform a maintenance operation. The operation should be as simple and fast as possible.

Simulations of maintenance operations are necessary during the early design phases to insure that the product will be maintainable before the design is committed. A tool like RAMCAD would be appropriate for this purpose.

6.10.4.8.6.13. Design Documentation Tool

Solid modelers may one day automatically generate drawings with very little help from a human resource. Most three-dimensional (3-D) wireframe, surface and solid modeling tools include a drafting tool. The drafting tool should allow human resources to derive two-dimensional (2-D) representations from 3-D models and embellish them as drawings. Drawing annotation that is unique to the drawing and cannot be derived from the attributes of the 3-D model can be added with the drafting tool.

Transferring views of 3-D models to an inexpensive 2-D modeler for drafting functions reduces the cost of drafting. However, for data consistency purposes, the drafting tool should be part of or at least linked to the modeler. Then changes to the model will be automatically reflected in the drawing.

If changes to drawing geometry can be made independent of the 3-D model from which it was originally derived, or vice versa, then the drawing can differ from the model. Those validating, fabricating, inspecting, assembling or maintaining the part on the basis of the drawing will generate failure and fit problems for those who base their work on the model and vice versa.

Ideally, it should be impossible to modify derived geometry independently of the model. The drafting tool should require that the 3-D modeling tool be the only means to change the geometry depicted in a drawing. If the derivation process results in an independent drawing model, then care must be taken to assure that changes to geometry are always made to the 3-D model and the corresponding drawing geometry is re-derived. The drawing tool should insist that any changes to the geometry be performed on the source model.

Model changes should automatically be reflected in the dimensions depicted in the drawing. The drawing tool should notify its user of annotations that could not be automatically updated. These usually have no relationship to the geometry, so little modification to the drawing should be required.

The actual paper or mylar drawings that are plotted from the drawing model present a similar synchronization problem. The drafting tool should always print a warning that the drawing was derived from a computer-based model that is considered to be the "master drawing." No changes are to be made to any printed drawing. All changes are to be incorporated into the computer-based master and the drawing re-plotted.

For government customers, the dimensioning and tolerancing capabilities of the drafting tool should comply with ANSI Y14.5, DOD-STD-100 and ANSI Y14.26.3. Maintenance requirements for drawings and associated lists, databases, microfilm and referenced documents may be required. Electronic

approval procedures should provide for the entry of the names or signatures of the approvers into the signature and revision blocks of the affected drawings by whatever means that will assure accountability. Line conventions and letters may vary for CAD prepared drawings as long as legibility requirements, like those imposed for microfilming purposes by MIL-M-9868, are met.

When magnetic tape or other digital media is specified as the physical media for a deliverable, it should conform to MIL-D-28000. If IGES engineering drawing data files are involved, they should be Class II tool data subsets. If printed circuit board descriptions are involved, their description and form should conform to ANSI/IPC-D-350 or MIL-D-28000. If raster engineering drawing data files are involved, they should comply with MIL-STD-1840.

When digital product definition data is exchanged between dissimilar systems in the format of the Initial Graphics Exchange Specification (IGES) per ANSI Y14.26M, copies of the drawings derived from such digital data should include the following legend inserted in a box under the last entry under the revision block:

[IGES - ( )]

Inside the parentheses are to be inserted the IGES version number.

6.10.4.8.6.14. Manufacturing Preparation Tools

The following tools are used to design the manufacturing part of the process.

6.10.4.8.6.14.1. Manufacturing Process Planning Tools

A manufacturing process plan differs from an engineering plan only in the skills required to conduct the subtasks, the granularity of the subtasks (operations) and the repetitiveness of the subtasks (make 20 of these). Consequently, the Process Manager could be used to delineate and sequence manufacturing tasks, subtasks and operations and maintain them in the Task Breakdown Structure (see Information Integration section). However, the scheduling of subtasks at the various manufacturing stations can be very complicated. A Manufacturing Resource Planning (MRP) tool may have to be used.

6.10.4.8.6.14.1.1. Variant Process Planning Tool

If no generative process planning tool is available, a variant process planning tool should be used. With variant process planning, old or *standard* plans are recalled as a function of the characteristics (attributes) of the part being processed. The plan most appropriate for the specific part is copied and manually modified, considering part peculiarities and current manufacturing capabilities (skills). It is then issued as the process plan for the new part.

6.10.4.8.6.14.1.2. Generative Process Planning Tool

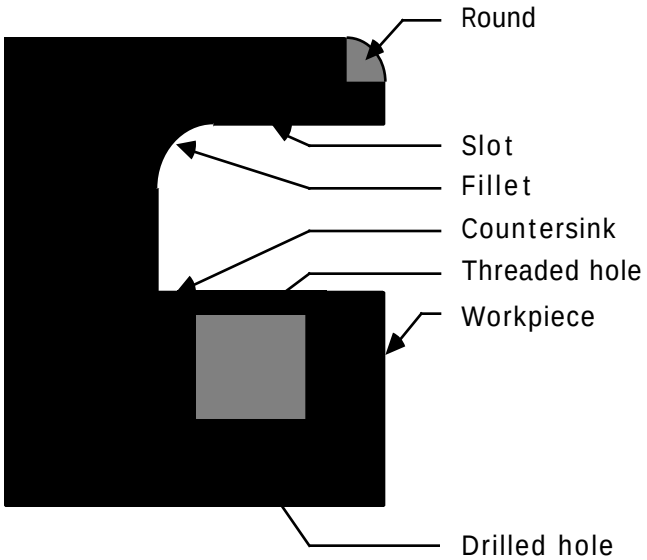
Generative process planning is now a common phrase. It was invented to describe the automatic generation and sequencing of fabrication and assembly

instructions. A software tool evaluates a part and goes through much the same thought process as a human planner to arrive at an optimum process plan.

As used here, generative process planning includes the handling, inspection, test and maintenance functions of the manufacturing process as well as the fabrication and assembly functions. The instructions may be human readable for manual fabrication and assembly, or in digital form for the computers that control the shop floor, cells and the handling, fabrication, inspection, assembly and test machines that comprise the cells.

The various handling, fabrication, inspection, assembly, test and maintenance operations have detailed activities associated with them. These operations typically relate to features of parts. A tapped hole, for example, may involve rough and finish drilling operations, a countersink operation and a threading operation. Each operation may involve different tools. Handling (set-up), inspection, assembly and test operations must be defined as well. These operations may be associated with parts or their manufacturing features. The actual conduct of the operations are specified by Numerically Controlled machine (NC) programs as described in the Generative Numerical Control Tool section.

There is often more than one set of manufacturing operations that can be used to make a feature. For each set, the manufacturing operations required to make an isolated feature are the same for all uses of that feature. These operations may become more involved when the feature is used in a compromising situation. The material handling, fabrication, inspection, assembly and test processes may also be complicated by other features of a part or adjacent parts in an assembly. For example, the operations involved in making a tapped hole may become very difficult and complicated if it can only be done from one side, and that side is obstructed as shown in the following figure.



Solid modelers make generative process planning practical because they allow features to be defined. Operations that are only meaningful at the feature, part or assembly level can be maintained at that level. The machine and human resource instructions for a fabrication, inspection, assembly, test or maintenance operation are a deliverable of a process planning subtask. That deliverable is associated with the CBS (fabrication or inspection) or ABS (assembly or test) as appropriate for the operation it defines.

The manufacturing schedule is derived from the fabrication and assembly sequence and maintained in the TBS. Ideally, the instructions are executed such that resources are bound to the subtask and parts are fabricated and inspected just in time for their assembly and test. Resource unavailability may, however, disrupt the ideal. Disruptions to disruptions lead to the use of a commercial MRP II system.

There can be many ways to produce (drill, punch, etch) a manufacturing feature (hole). More than one process plan can be derived for the same part. The generative process planning tool will likely be invoked during design to validate the producibility of each design and help provide an estimate of its cost. It should use the resource set with the optimum combination of skills and proficiencies. It will likely be invoked during manufacturing to accommodate the unavailability of resources or the availability of new resources.

The generative process planning tool should infer from the component model geometry the manufacturing features necessary to fabricate a part to match the design model within its tolerance requirements. Given the features and the assigned resources, it should strategize an optimum fabrication sequence. It should similarly infer inspection features and inspection operations. It should generate all the steps necessary to handle and restrain the material and any intermediate (synthetic) parts.

The generative process planning tool should be smart enough to require a spotface operation before a drill operation that is not perpendicular to a surface. It should identify the need for multi-use fixtures and tools by scanning the model database for parts with similar manufacturing and inspection features that would require such fixtures and tools. If no compatible fixtures are found, it should invoke the generative fixture design tool to create the tool, or alert the fixture design function.

Similarly, the generative process planning tool should infer from the spatial relationships of parts in an assembly the manufacturing and test features necessary to assemble and test the assembly. It should generate all the steps necessary to handle and fixture (jig) the parts for assembly and test, including fixture design, assembly (robotic) programming and test programming. It should identify the need for any intermediate (synthetic) assemblies and accommodate their handling and storage.

It should be smart enough to invoke a robot simulation tool to help determine the optimum assembly procedure. It should identify the need for special fixtures

by scanning the modeler database for assemblies with similar manufacturing features. If needed fixtures are not found in inventory or are not on order, it should invoke the generative fixture design tool, or alert the fixture design function. It should select the optimum assembly and test process plans for the shop floor conditions anticipated when the parts are to be assembled and tested.

The generative process planning tool should also use fabrication and manufacturing feature relationships to identify incompatible operations. For example, associated with the fabrication features of a flange might be a welding process or the use of a relatively inaccurate drilling process for the holes in the flange. The fabrication features have one or more boundary elements in common with an assembly feature of the flange. The assembly feature of the flange is used to mate with an assembly feature on a mating part. Associated with the assembly of those two assembly features is a tolerance. The generative process planning tool should examine the fabrication processes associated with each of the fabrication features that share boundary elements with an assembly feature to determine if the fabrication process is capable of fabricating the feature with sufficient accuracy to assure that its related assembly feature will assemble according to the assembly tolerance.

6.10.4.8.6.14.1.3. Parts Nesting Tool

The flat patterns that result from a sheet metal design tool should be used by a parts nesting tool. The parts nesting tool should use material attribute values and bulk material size to select which parts to position and orient the pattern on bulk material such that material waste is minimized. It should generate end-effector motion or punch instructions to cut the parts from the bulk material.

6.10.4.8.6.14.2. Manufacturing Tool Design Tools

Both fixtures (jigs and clamping devices) and tools (drills, reamers, cutters) are often called *tooling*.

6.10.4.8.6.14.2.1. Tool Design Tool

Tool designers use part geometry as the requirement for the design of special tools. For example, a mill specially shaped to cut many part fabrication features in one operation.

If no generative tool design tool exists, extensions of the model construction and parametric design tools to include features appropriate for tool design, like helixes (drills) and tool related parameters like shaft type and draft angle would be desirable. The resulting tool should include or have access to a library of material properties for both tools and workpieces, a library of machine parameters and a library of tool holders. It should present this information on demand in a format conducive to good tool design. After a tool designer has defined a tool, the tool definition could be given to the generative process

planning and NC programming tools to generate the human and machine resource instructions necessary to fabricate the tool.

6.10.4.8.6.14.2.2. Generative Tool Design Tool

The generative tool design tool should automatically generate an optimum tool design based on the geometry of the feature it is to make, the material(s) it is to cut and the range of machine parameters (feed rate, spindle rpm) available. The resulting tool definition could be given to the NC programming function or generative NC tool by way of the generative process planning tool to generate the machine instructions necessary to fabricate the tool.

6.10.4.8.6.14.2.3. Fixture Design Tool

Fixture designers use part geometry, machine beds, the factory floor and adjacent parts in assemblies as constraints on the design of fixtures.

If no generative tool design tool exists, extensions of the model construction and parametric design tools to include features appropriate for fixture design would be sufficient. A fixture design tool should include or have access to a library of machining area (mill bed, coordinate measuring machine bed, factory floor) and available attachment geometry (slots, pin holes) or functions (vacuum) to provide a factory constraint on fixture design. It should access the process plan to determine which machining area and finished part are involved. It should scan the process plans to find those with similar feature requirements to locate existing fixtures that may be appropriate for the new part. From those fixtures, it should select the one to be used for this part or used as a baseline to design a fixture for this part. It should present the most appropriate fixture, the machine bed and the part, or intermediate part, to the fixture designer in a way that is conducive to good fixture design. After a fixture designer has defined a fixture, the fixture definition could be given to the generative NC programming tool by way of the generative process planning tool to generate the fixture tool according to the optimum process determined by the generative process planning tool.

Normally, a modified fixture must be saved under a new name to preserve the fixture definition from which it was derived. If the owner of the source fixture determines that the modified fixture can be used for both the old and new parts, it can be saved under the old name.

6.10.4.8.6.14.2.4. Generative Fixture Design Tool

A generative fixture design tool needs to know the process by which a part is to be fabricated, inspected, assembled or tested. It needs the geometric constraints of the part and the machine on which the part is to be fabricated, inspected, assembled or tested. It needs to know the material to be cut, the cut rate, and the tool type so it can determine the loads that will be applied to the part by the machine. It needs to know the tolerance requirements of the part to determine the stiffness requirements of the fixture. If the tool access

requirements are based on tool centerline only, then the generative fixture design tool will need tool geometry information as well.

Given this information, the generative fixture design tool should automatically generate a fixture design. The resulting fixture definition could be given to the generative NC programming tool by way of the generative process planning tool to generate the human and machine resource instructions necessary to fabricate the fixture.

6.10.4.8.6.14.2.5. Mold Design Tool

If no generative mold design tool exists, an extension of the model construction and parametric design tools to include features appropriate for mold design, like material supply ports and mold release angle could be used. This tool should include or have access to a library of material properties for molds, casting and release material, a library of mold machine parameters (pressure, feed rate, temperature) and a library of mold holders (fixtures). It should present this information on demand in a format conducive to good mold design. After a mold designer has defined a mold, the mold definition should be given to the NC programming tool by way of the generative process planning tool to generate the machine instructions necessary to fabricate the mold.

6.10.4.8.6.14.2.6. Generative Mold Design Tool

The generative mold design tool should automatically generate a mold design (model and attributes), including the necessary release offsets, given the geometry of the part it is to form, the material(s) involved and the geometry of the mold machine. The resulting mold definition could be given to the generative NC programming tool by way of the generative process planning tool to generate the machine instructions necessary to fabricate the mold.

6.10.4.8.6.14.3. Machine Programming Tools

Machines (mills, lathes, robots) are the resources used to perform the fabrication process. Tooling is usually required to facilitate the process

6.10.4.8.6.14.3.1. Machine Programming Tool

Machine programming is an extension of process planning into the details of material modification (machining, forming, etching) and part assembly. It is an extension of process plan instructions into the digital world of Numerical Control (NC - numerically controlled machine), robotics and cell control.

Fabrication and assembly features and assembly commands dictate certain handling, fabrication, inspection, assembly, test and maintenance operations. They do not define specifically how a particular machine tool will rough-cut and finish-cut a part, or how a robot might assemble or disassemble an assembly. The features and commands are machine-independent attributes of parts and assemblies.

Each machine tool or robot has particular attributes (skills) like horsepower, feed rates, spindle speeds and head or end-effector positioning and orientation options and limits. Certain types of machines can perform certain operations or combinations of operations. To minimize the impact of a machine failure or a resource conflict that would require that a different machine or a combination of machines be used, a factory must be adaptable.

To provide this adaptability, the machine programming process evolved the following sequence.

Define the part geometry.
Define cutter or end-effector motion.
Define the machine parameters (feed, speed, coolant) for that motion.
Convert (postprocess) the motion and parameter instructions into a language that a specific machine will understand (machine control data - MCD).
Communicate the machine-dependent instructions to the designated machine.

The first three steps are done by human resources using the APT (Automatically Programmed Tool), Compact II or other machine programming language. The conversion is done with a post-processor. The communication was done with paper or mylar punched tape.

Most NC shops have replaced the paper and mylar tapes with Direct Numerical Control (DNC) computers and networks. They distribute blocks of MCD to Computer Numerical Control (CNC) computers, which feed the machine tools MCD on demand, as if a new reel of tape were being mounted. This improvement keeps MCD in digital form. It eliminates the need for tape libraries. It eliminates tape reader acquisition and maintenance and the machine inactivity associated with tape changes.

Computer Aided Manufacturing (CAM) systems have added graphics and some automation to the first three steps. Graphic systems have made it easier to define and visualize cutter motions. Some motions, like those required to remove the material from a pocket, are automated once the drive and check surfaces are defined by an NC programmer. The systems provide graphic feedback of the cutter motion, but most motions and all the machine parameters must be specified by an NC programmer.

Recently, solid modeling tools have not only made the interpretation of design intent less ambiguous, but have also facilitated more powerful machine programming tools (complex pockets) and more realistic renderings of the part throughout the fabrication, inspection, assembly or test process. Although this reduces machine programming time and makes it easier to detect errors, it is still a tedious and error-prone manual process.

Some modelers are unintegrated with solid and surface modelers. Some require that the solid be converted to surfaces for machining programming purposes. This is an undesirable extra step in the process.

Complex surfaces can be difficult for a machine programming tool, especially if many surfaces are involved. The machine programming tool should enforce the topology rules of the model. It should not have to concern itself with surface boundaries that are not tangent or coincident. The modeler should resolve such situations.

Libraries of tools (drills, cutters), end-effectors (grips, drill and router motors, spray heads), machine parameters (feed, speed, travel, reach, swing, degrees of freedom), and fixtures whose attributes can be copied and inserted into machine control programs, would facilitate the machine programming effort. The library data should be presented with the part or assembly models and fixtures on demand in a format conducive to good machine programming (animation).

The assembly tool commands used during the design part of the product development process should simplify the robot programming effort.

Most of the CAM systems generate a dense list of point-to-point moves that can be input to the APT post-processor. Some CAM systems transfer machine parameters as well. Unfortunately, some graphic systems only generate machine-specific MCD. This is particularly true of printed circuit board drilling, electrical component placement and robotic related systems. There is an effort to rectify this situation so APT can be used for all machine tools to provide a degree of machine independence for all operations.

6.10.4.8.6.14.3.2. Machine Program Verification Tool

Regardless of the manual programming tool used, programming errors are not obvious. Hence, the need for an *NC Verification* tool. Until a generative machine control tool is available, an end-effector motion (cutter path) verification tool is required. This tool should use the part, tool, fixture and machine geometry to simulate the machining process as specified by a machine control program. It should indicate by way of color changes areas where the material would be under- and over-cut, or fixtures (clamp) or the machine (bed) would be damaged. The effect of tool wear and deflection and machine characteristics (slop, inertia) should be included in the simulation to eliminate any need for part program proofing on real machines.

6.10.4.8.6.14.3.3. Generative Machine Control Tool

Generative NC and Automated NC are common phrases used to describe the automation of the process of strategizing material removal (milling). Its initial focus was the automatic generation of material removal machine control commands as a function of the geometry of the finished part and its attributes. As used here, generative NC is not limited to conventional NC machine tools

(lathes, mills). It includes any manufacturing operation that is dependent on the geometry of a part, like material handling, material removal, part inspection, assembly and test.

Generative NC tools automatically infer from the part geometry the cutter or end-effector motion and machine parameters necessary to optimize the material removal process. A complete set of machine and operator instructions result from the execution of the program. The instructions can be for a class of machine resources or a specific machine. If they are generic, the format of the instructions is usually in APT. This preserves the machine independence provided by APT, allowing existing post-processors to be used to derive the machine control data from the APT program. If they take advantage of machine-specific features, the use of APT may only be a matter of convention. The program must be run on the targeted machine.

Solid modelers make generative NC practical, because the geometry is accurate and unambiguous. Cutter or end-effector motions that are only meaningful at the feature level can be maintained with the appropriate features.

Human sight can be easily fooled. Objects which look perfectly correct on a display may, in fact, have discontinuities. Discontinuities drive APT processors, NC machines and robots crazy. For this reason, modelers that do not enforce coincidence and tangency constraints, whose representations are faceted, or which resort to polygons for some of their Boolean operations provide unacceptable input data for generative NC tools.

A generative machine control program tool should automatically generate the machine control program necessary to handle, fabricate or inspect a part or assemble parts. The generative machine control program tool may be invoked by the generative process planning tool to determine the feasibility of using one or more machines for a particular process.

The machine cutter or end-effector control language generated by the machine programming tool should generate source control language in a standard language, like APT. Then it can be post-processed on demand for any of the machines capable of performing the operations dictated by the program. Modeler post-processors are undesirable.

6.11. Human Resources

This discussion follows from the Current Trends in Business section of the Introduction.

6.11.1. Manage Less

People tend naturally to be helpful and cooperative. They naturally seek the best help available. When they are busy, they will not accept responsibility for additional work They abide by their prior commitments. When they over-commit themselves or the unexpected happens, they work extra hours.

In unenlightened businesses, people are constrained from cooperating with one another by the requirement to first check with the boss ... who will not be in until next week ... who must check with his boss .... The impediment to highly productive *teams* (aka *concurrent engineering*) is management.

In an enlightened business there is nothing for middle management to do. They return to front-line management roles, or to the technical jobs they would not have left were it not for the money, or move to entirely new jobs. That nagging feeling that they are just messengers and not really contributing to the business process is replaced by a true sense of purpose and job satisfaction.

Supervisors really cannot be effective unless they know enough about the technology employed by their function to converse intelligently with the people in their employee. They must also know enough about the business of their customer to converse with their customers intelligently. Total all the administrative and police duties that have become the responsibility of a supervisor and it becomes apparent why good supervisors are a rare breed.

Their secretaries are often frustrated by the need to have the supervisor sign a document before even trivial actions can be undertaken. Many supervisors have encouraged their secretaries to forge their signatures on certain documents, or with verbal authorization to sign any document.

The technical and administrative duties of a traditional supervisor are not only overwhelming, but also represent a mental schism. Few people are inclined or capable of performing both technical and administrative jobs. Both are or very nearly are full time jobs, so why not split an impossible along different lines?

Supervisors and secretaries should be replaced with technical lead and administrative assistant teams. The technical lead would be "in charge." The administrative assistant would have signature authority on all non-technical documents. Their pay should be based solely on merit. In a smoothly running office, one administrative assistant may be able to support two or more technical leads. With the freedom to work when there is work and not to work when there is not, peak work load conditions can be accommodated without additional personnel.

The following table lists many of the tasks currently assigned to supervisors. A check in one or more of the three columns entitled "TECHNICAL LEAD," "ADMINISTRATIVE ASSISTANT" and "EMPLOYEE" denotes a task appropriate for that role.

TASK	TECHNICAL LEAD	ADMINISTRATIVE ASSISTANT	EMPLOYEE	COMPUTER
Identify problems			x	
Define requirements			x	
Distill solution			x	
Plan product	x			
Plan business		x		
Define tasks	x			
Allocate work	x			
Track Work		x		x
Counsel technically	x			
Counsel personally	x	x		
Market products and services	x			
Establish vision	x		x	
Educate about process		x		
Provide retirement info./status		x		
Provide medical plan information		x		
Provide insurance information		x		
Distribute mail		x		
Establish labor account		x		
Authenticate labor account		x		
Educate about labor accounting		x		
Record labor			x	
Schedule vacation			x	
Authorize vacation			x	
Track vacation		x		x
Track sick leave		x		x
Track actuals versus budget		x		
Provide pro/demote evidence	x		x	
Provide pro/demote forms		x		
Provide merit in/decease evidence	x		x	
Provide merit in/decease forms		x		
Provide termination evidence	x		x	
Provide termination forms		x		
Select candidates	x		x	
Schedule candidates		x		
Complete hire forms		x		
Relocate personnel		x		
Move or repair facilities		x		
Provide supplies		x		
Plan career			x	
Rotate jobs			x	
Provide security briefing		x		
Provide ethics briefing		x		

Although this would be a step in the right direction, the role of front-line management should shift from that of a supervisor or foreman (nursemaid and police officer) to that of a

- facilitator of discussion within a team,
- coordinator among teams and

- seeker of new methods and technology for the team to consider making part of their role in the business process.

The team leader leads by way of the Socratic Method: by asking the right questions of the right people at the right time. The team leadership role may rotate among team members. The administrative assistant is just another member of the team. When other team members need a change of pace, they may perform some of the duties of the administrative assistant or other team members.

Top management continues to seek business opportunities and financing, and collect market and competitive information. Instead of limiting that information to strategy meetings with a few top level managers, it is shared with the entire work force. Top management works with the employees to establish a common vision for the business, a strategy for its realization and the incentives to motivate the employees and management to achieve that vision. Top management also works with the employees to establish measures of success for each work cell and individual. The measures should help identify the incentives that will motivate them to contribute to the overall success of the enterprise. Gone are counter-productive micro management, micro accounting and micro reporting practices. (Recommend reading: "Managing without Managers" article in September-October 1989 Harvard Business Review by Richard Semler, from his book entitled "Turning the Tables.")

For authoritarian business to be competitive in the new business environment, they must evolve quickly and radically. Here are some recommendations.

6.11.2. Un-Organize

Consider all employees, computers, machines and facilities to be one large resource pool. They all have skills. No two employees and few computers and machines have exactly the same skills or the same proficiency in a particular skill. Specific skills or skill sets are required to perform specific work or sets of work.

Program managers, *project manager*s, *cognizant engineers* and *responsible engineers* are human resource consumers. *Individual engineers* consume all other resource types.

Other than by way of who knows who or what in an organization, there is no way for someone to find needed resources. Many skills are not utilized, because qualified resources or their skills are unknown to the consumers. A *Resource Manager* tool as described in the Tool Resources section would help resource consumers be cognizant of available skills, aid in the allocation of those skills and help reconcile resource conflicts as work is scheduled or schedules change.

It is possible to **require** the Human Resources Department to update a Resource Manager database as employees hire or terminate or gain skills or proficiency. It is possible to **require** the Industrial Engineering Department to

update a resources database as machines are acquired, sold, upgraded or degrade. It is possible to **require** the Information Resource Management Department to update a resources database as computers are acquired, sold or upgraded, they are not inherently motivated to do so. It would be an onerous task to maintain all of this information in a database for resource consumers to access. Inaccurate or untimely information could adversely affect a human resource, and result in a lawsuit.

A Resource Manager is appropriate for authoritarian businesses. However, it can also be a tool for libertarian businesses when it is used by the resources to market their skills and make them aware of opportunities. In this case the Resource Manager that is more like an electronic bulletin board. If someone wants work performed, let them describe the deliverable, skills, proficiency and due date to it. Let the human resources describe their skills and the skills of their tools to the Resource Manager. Let the Resource Manager identify possible matches between the work and the resources. Let the resources bid for the work. Let the resource consumers contract with the resources.

The Resource Manager would be like a work broker and personnel broker combined. There is no need to **require** anyone to maintain the data known to the Resource Manager. The maintenance of accurate data is the vested interest of all the participants.

If many resources bid for the work, the competition may force the price down. If few apply for the job, the price may be forced to increase. If none apply, or the price is too high, the project manager may degrade the proficiency requirements, allow a combination of resources to perform the work. If time permits, seek resources from outside the enterprise … by way of the Resource Managers of subcontractors.

Human resources should have the freedom and responsibility to acquire skills or improve their proficiency in the skills that are the most demanded or interesting. They should have the freedom to acquire (buy, lease, rent) tools (machines, computers, software) that will make them more proficient or effectively give them new skills. They should be free to team and subcontract the work, and share the cost and benefit of tools. They should be free to generalize their skills to be qualified to perform a lot of different work. They should be free to specialize to increase their value for a specific kind of work.

Some human resources may create work for themselves or demand a higher price by inventing new capabilities or procedures that improve the business process. To encourage innovation, enterprise "patents" should be issued and protected by the enterprise, or issued as a contract between a human resource and all other human resources. Then it can be protected by way of private or public civil courts. These may eventually become national or international patents. All patents can be bought, sold or leased (royalty payments).

Similarly, resource consumers should be free to pay a higher price for the skills of resources that are known to be reliable and who accurately represent their

proficiency. resource consumers should be free to take a chance on an unknown resource to lower costs. However, wise project managers will only do so when there is sufficient slack time in the schedule to permit recovery from a bad choice.

Resource consumers are also human resources. They have a certain skill set and inclination. They too should be free to bid for work and acquire tools. They may share their experience with other resource consumers, which may cause the proficiency rating of certain skills of a resource to change or become less credible. They may buy proficiency ratings from those who specialize in that service. This is a likely job for those who were functional managers. They may also be funded by groups of project managers to add or improve scarce skills. Resource consumers may team with other project managers in retain idled resources in anticipation of their need at some future date or to keep their better resources from working for their competition.

At the discretion of the human resources, enterprise management may advertise for work to improve the appearance, health, safety or functionality of enterprise-owned facilities. Groups of human resources may advertise for and pay someone to acquire new tools that can be shared (leased or rented) by many human resources. Some human resources may specialize in acquiring new tools for lease or rent to other human resources. Some may specialize in the maintenance of information resources (communication networks, mainframe computers) and charge for their use. Others may specialize as effective team leaders or administrative assistants.

6.11.3. Toss the Time Clock

With human resources being paid by the subtask or "by the piece" instead of by the hour, there is no justification for time clocks or set working hours. Empower the human resources to communicate how, when and where they want to get their jobs done most efficiently.

6.11.4. Flatten and Divide

Collapse the management hierarchy into two levels: Manager and team leaders. If the number of team leaders is appropriate for the number of human resources and there are too many team leaders or members in one organization for effective communication, then divide the organization into separate profit centers with separate managers.

6.11.5. Reallocate

How do machines, fixtures and machine tools bid for work? Sell, lease or rent all non-human resources (tool cribs and tools, stockrooms and inventory, fixtures, computers, furniture, research, design, analysis, material handling, fabrication, inspection, assembly and test equipment) to the highest bidder among the human resources. The human resources will naturally acquire only those resources that cost-effectively improve their productivity, and hence their competitiveness. The human resources will use their resources to acquire more work or acquire more

dollars per hour for their work, or both. Those resources that are not acquired should be sold to outside interests. Else they waste floor space which could otherwise generate income for the enterprise.

Allocate discretionary funding for the enterprise beyond management to each human resource. Let them improve their productivity by acquiring new tools, fixing old tools, improving their working environment or whatever. If someone believes that something beyond their means would significantly benefit the enterprise, they can lobby others and convince them to pool their discretionary funding to acquire more expensive items or services.

To give the human resources the option not to spend funds allocated to them annually, make them accountable for the best return on those funds. Allow the funds to be invested in an interest-earning account. Allow the funds to cumulate over multiple years. If the human resource thinks a greater return can be had by investing the funds in the enterprise, then that risk should be taken. Else the funds earn the expected interest rate. The human resources that consistently earn a higher return on their investment relative to the investments of other human resources will be able to demand a higher pay rate and/or attract more investment funds.

6.11.6. Privatize

The enterprise with the better location and facility will attract the better human resources. Medical, child care and pension benefits may also be used to attract human resources, but pension benefits may tend to attract the less innovative or adaptive human resources.

If an enterprise wants to attract the most innovative and adaptive human resources, and avoid the cost of being an employer, like the cost of managing pension funds and medical plans and paying social taxes, it should convert its human resources into subcontractors. It should help individuals or groups of human resources establish themselves as independent businesses with a name of their choosing. The parent enterprise can help the sibling businesses find health, life and disability insurance, financial resources and investment plans individually or as a group.

The parent enterprise can become a *Mall of Subcontractor*: a building full of tenant sibling businesses. Let the siblings mark their floor space and erect business signs and offices. Let them bid higher rents for the better parking spaces. Their costs thereby shift from overhead to direct. The incentive of having their own business will increase their productivity.

Rent, lease or donate two personal computers or workstations to the sibling businesses. Put all the business software (electronic mail, accounting, billing, payroll, taxes, telephone answering/messaging/broadcast) they need to conduct a small business competitively on one computer. Put all the software they need to run their tools (structural, electrical or software design, documentation, analysis, manufacturing) on the other. Connect the computers together so one can be the back-up for the

other. Connect the computers to the enterprise subcontractor computer network or an internal network for access to work requirements (Resource Manager) and for coordination with others. Allow them to connect to other networks of their choosing as well.

Bill the sibling businesses for the use of the floor space (including allocated fire and liability insurance), utilities (electricity, heating, air conditioning, telephone) and toilet facilities. Let them worry about their own materials, material handling, fixtures, tools, machine maintenance, telephone answering service, theft protection and clerical help. Let sibling businesses or external companies compete to provide such commodities and services.

If some of the sibling enterprises fail to compete, then other sibling enterprises or external enterprises may bid for their assets and floor space, and perhaps hire the less entrepreneurial human resources.

If unused floor space exists, allow any sibling enterprise or outside enterprise to bid for the floor space. Let them install additional machines of their own. If the parent enterprise has insufficient work to occupy a sibling enterprise fully, let it seek work from outside the parent enterprise.

Malls can be as large as the facility permits, but they should not be geographically distributed. Separate facilities should be separate Malls. Individual buildings can be Malls. The Malls should remain open 24 hours a day, 7 days a week.

This approach is not limited to manufacturing enterprises. It can be applied to engineering enterprises, or any enterprise for that matter. Some companies are both. With the separate contracts for design and manufacturing becoming the norm, it may make sense to make Engineering and Manufacturing (Operations) separate cost centers even if they are in the same facility.

Pay per deliverable, not per hour. This should hold true for deliverables among functions in the process (sibling enterprises and subcontractors) as well as for deliverables to the parent enterprise (system integrator).

Those managers who do not participate in the sibling enterprises may become Mall managers. The number of personnel involved in the administration of the Malls must be minimized if the Malls are to be competitive. Very low rents are required to improve the odds that the sibling companies will survive their formative years. Many of the manufacturing managers and human resources may be employed by the engineering sibling companies to help them make their designs more producible.

The parent enterprise should qualify the process of the sibling enterprises as the parent enterprise would qualify that of any subcontractor or supplier.

6.11.7. Communicate

The parent enterprise should use a computer network to interact with all sibling companies and outside suppliers, subcontractors and associates, so work requirements need only be placed on one bulletin board (Resource Manger) for all to see. This will make the transition from an enterprise work bulletin board to an international work bulletin board easier.

Companies like General Electric Information Services (GEIS) may expand beyond international electronic mail exchanges among "incompatible" enterprise electronic mail systems into *work brokerages*. Companies that need work performed will subscribe to such an exchange to broadcast work requirements world-wide to get the best price, quality and response possible. Companies that want work will subscribe to receive such information and respond via their electronic mail system. Not to subscribe will be to go out of business.

This electronic information exchange service could include all kinds of data (requirements, specifications, graphical and geometric data and machine motion data) as attachments to messages. It could perform data format and language conversions from that of the sender system to the that of receiver "on the fly." Such a system would support not only product development and manufacturing, but also product support (CALS). Eventually the parent enterprise will be using such a service from all the possible perspectives (prime contractor, subcontractor, associate producer, partner, buyer, seller...).

A similar service will be available to help businesses find and acquire existing parts (Information Integration section), including their specifications and geometric models for direct inclusion in designs before their acquisition. It will become the "one stop shopping center" for parts. For a manufacturer not to advertise its parts on the "parts bulletin board" would mean certain business failure. This will insure that all available parts are on the bulletin board.

6.11.8. Disperse

Use the communication network to allow the sibling enterprises to disperse to their home or whatever location is most amenable to their productivity. Clerical, engineering and non-polluting fabrication sibling enterprises can be conducted within private residences. This will avoid the pollution, auto and highway wear, traffic congestion and time lost commuting. It will save petroleum and other resources. It will make the parent enterprise a "good neighbor" in its community. Given the freedom to work where and when they want will maximize the productivity of human resources and the effective use of all other resources.

Use the communication network as an international *work brokerage*. Expand the request for bid and bid work/resource matching process into a world-wide activity.

6.11.9. Start

Such radical changes will take time. Until then, some relatively easy changes can be made to begin the process of change.

Eliminate all privileged parking (supervision, employee of the month...).

Eliminate the executive dining room or convert it into a Total Quality Management (TQM) educational center.

Eliminate the correlation of office size or location to management position. As facilities are modified, replace closed offices with standard low partition "cubicle" work spaces.

Disassociate position from apparel. Encourage comfortable, functional and safe clothing.

Until each employee becomes a contractor, equate cost to demand. As soon as the demand for a resource surpasses an average of 40 hours per week for human resources, increase the cost of that resource. Temporarily increase their pay accordingly. The increased cost will reduce demand and protect valuable resources from over use (burn-out). Long term demand trends will indicate when the base pay of a human resource should be increased to retain the resource, and when the base pay is unjustifiably high.

The allowable average for hours per week for machines, computers, facilities and other resources would depend on the resource. Use the excess funds derived from the higher rates to acquire additional temporary or permanent resources. Do the reverse if the demand falls below optimum levels.

6.11.10. Change

Keep changing for the better. Not to improve continuously is to be destined for oblivion.

6.12. Machine Resources

Machines are large and small, heavy and light, fragile and tough. They may
- move material (conveyers, robots),
- cut material (saw, water jet, laser),
- shape material (mill, turn, mold, electro-erode, etch),
- add material (paint, glue, weld, wave solder, electro-deposit),
- measure material (acoustic, optical or laser range finders, load cells
- assemble material (composite lay-up) or parts (inserters, robots) or
- test systems (bed-of-nails circuit board testers, volt/ohm meters).

Some perform a combination of these functions. The key attributes of machines are their modes of control, set-up and their adaptability.

The function of many machines can be controlled by computing resources or paper or mylar tapes. These can be replaced with computer-based controllers. However, other computer controlled or human controlled machines or human resources are usually required to
- move material or parts to and from the machine,
- install jigs and tools on the machine and
- secure to and remove from the working surface of the machine the material or parts that are to be operated upon (fabricated, inspected, assembled, disassembled, tested or delivered) by the machine.

Combinations of machines may be co-located into cells and coordinated by a combination of computing and human resources.

Both controlling resources require instructions. The human resources require their instructions in audio and/or visual form. Time is required for the human resources to understand the instructions. The computing resources require a set of specially formatted, digitally encoded instructions for which virtually no learning or transmission time is required.

Human resources are expensive and sometimes unavailable, unreliable (breaks, sickness) and inconsistent (mood, physical condition). They are inventive, dexterous and adaptive. They are better suited for creative and non-repetitive tasks. They can interpret incomplete or ambiguous instructions. The time required to create instructions for human resources is small compared with the time required to program computing resources.

Computing resources must have complete and unambiguous instructions. However, computing resources are relatively inexpensive, available and reliable. Their behavior is consistent. Hence, repetitive or at least predictable tasks are more appropriate for computing resources. Therefore, machines that are employed doing repetitive tasks should be computer controlled. Machines that are employed doing creative tasks or which must adapt quickly should be human controlled.

Regardless of whether a computer or human resource controls a machine, that part of the business process which is dependent on the machine must stop and

wait for the resource to be alerted to the need for action. It must understand the action that is to be performed, act and finally complete the action before the process can continue. Computing resource can do it much faster than a human resource. Consequently, the more machines that are computer controlled, the more automated, timely and predictable the engineering and manufacturing process can be made.

Machines can be much more expensive than the computer or human resources that control them. Some combination of the productivity or the quality of work, or the longevity of a machine must justify its expense. Otherwise a combination of human resources and tools is likely the more appropriate alternative.

6.13. Facility Resources

Facility resources are those which house, support, energize, cleanse, cool, heat and light the other resources. Facilities resources include land, buildings, utilities, sewers, fire protection and the like. Some of the less obvious attributes of good facilities are listed here.

6.13.1. Air Conditioning

The temperature, humidity, particulate and chemical contamination of the air supplied to resources must be within the limits tolerable to them. Given the diversity of such requirements, some resources must be physically isolated. They must be provided with filters, different sources of air or additional air conditioning.

6.13.2. Lighting

Although fluorescent lighting is less expensive than incandescent lighting when left on for extended periods, some incandescent or ambient lighting is required to avoid injury. Some people have grabbed rotating parts in lathes and cutters in mills, because machines rotating at 60 cycles per second appear to be motionless when illuminated with only fluorescent light. Light sources should be easily moved to promote the mobility of the resources that are dependent on the light.

6.13.3. Protection

Protection from natural elements (rain, wind, sun) is required for many resources. Physical access constraints are necessary to prevent resource, personal property and information theft or damage. Visual access constraints are required to prevent information theft. Electromagnetic transmissions must be contained to prevent information theft (see Computing Resources section). Electromagnetic interference constraints are required to prevent a disruption of certain resources, especially communication resources. Depending on the frequency and intensity, internally (machines) or externally (jet aircraft) generated sound can be disruptive to computer, tool and machine as well ashuman resources.

6.13.4. Space

If a resource is to be contained within an enclosure, a volume of space that will accommodate it and its installation and access requirements is required. Ingress and egress volumes (minimum cross section times turn length) must accommodate the largest resource anticipated, except those that are expected to be disassembled for movement. Special consideration must be given to the safe egress of human resources, particularly those with motive constraints. Partitions should be moveable. Large portals (double doors, freight elevators) or alternate portals (removeable exterior walls or roof sections) should exist to accommodate large resources.

6.13.5. Stability

A foundation must be sufficiently rigid and stable to be a platform for resources that are sensitive to movement or alignment. For example, computer disk drives can tolerate little motion, especially abrupt motion like shock. Upper floors are often so flexible that a heavy person walking on one part of the floor can cause enough vibration on remote parts of the floor to cause disk drive heads to contact their rotating media, destroying disks. One solution is to analyze the structural dynamics of the floor and relocate the vibration-sensitive resources to more stable node points.

Large gantry milling machines do not have enough rigidity in their frames to maintain machining accuracy if the foundation warps. Foundation warping can be caused by changes in hydraulic pressure in the soil under the foundation, which can be caused by ocean tides or seasonal rain. One solution is to replace the foundation under such machines with large-reinforced, autonomous footings.

Earthquakes can induce motion in all floors of a building. Foundations isolated from earth can dampen building movement and minimize structural and resource damage.

6.13.6. Supplies

A means of supplying oxygen for combustion and respiration must be provided. Natural gas, petroleum and food must be supplied for conversion to energy. Oil and water must be supplied for lubrication, cooling and waste removal. Water must be supplied for hydration. Electrical power not generated internally must be supplied externally. Electrical and optical conduits must be provided for internal and external digital and analog data, voice, sound and video communications. The sources for these supplies or their conduits must be mobile if the resources dependent on them are to be readily moved to allow the enterprise to adapt to new business drivers quickly.

6.13.7. Waste

The sewage of human resources, material waste of material removal machines, chemical waste of integrated circuit and circuit board fabrication and the heat generated by all the resources that is not recycled must be removed. Due to building code restrictions, waste conduits tend to be the most difficult or expensive to move. They tend to immobilize the resources that require them.

Depending on the size of an enterprise and the proximity of its resources, the waste products of some resources may useful to other resources or the process in which they are involved. For example, furnace and autoclave waste heat may be used to remove the chill from the space of human resources.

7. INDEX

8. ACKNOWLEDGEMENTS

It is impossible to acknowledge everyone who contributed to the 26 years of diverse civil, aerospace and commercial industry experience and knowledge that has been distilled into this book. What was learned from publications and reports was small compared to what was learned personally or learned from the successes and failures of myself, friends, business associates and affiliates. Any sources not specifically mentioned have faded into the oblivion of a memory that was concerned with solving problems rather than writing a book. Where memory served, the source of a concept is credited in the section or paragraph that describes the concept. My apologies to those who inadvertently did not receive recognition they deserve. Please contact me so you can be appropriately acknowledged in the next edition.

9. ABOUT THE AUTHOR

As a stock boy, deliveryman, draftsman (Air Force Civil Engineering, Rocketdyne), and designer (NASA, Fight Operations and Advanced Design, Edwards), he earned his way through college. He received a Bachelor of Science degree with honors in Aeronautical Engineering from California State Polytechnic University in 1971, where a degree of proficiency in all the basic manufacturing methods was required. He delayed his career for three months while he was the "transportations systems engineer" for National Science Foundation interdisciplinary team (GY-9159). After one year of basic aerodynamics work (Rockwell International, B-1 Division), he began to seek computer solutions to manual wind tunnel data reduction and display methods. That earned him a job doing advanced aerodynamics (Los Angeles Aircraft Division) where he was initially involved with mainframe-based Computer-Aided Analysis (CAA) and later assigned the task of acquiring minicomputer-based CAA and Computer-Aided Design (CAD) capabilities and integrating them. In 1978 that work expanded to include analysis integration and data management (General Dynamics, Convair Division). During 1980-81 he examined the commercial engineering and manufacturing process as a product marketing engineer (Hewlett-Packard, San Diego Division). In 1982 he defined the requirements for a successful file control system for the management of CAD and other engineering data (General Dynamics, Data Systems Division). From 1983 through 1985 he was the project manager for a corporate-wide CAD/CAM Database Management System, and its subcontractor (Computer Corporation of America). His most recent assignment was as the process and information architect for a major enterprise integration and management project. Since 1976 he has devised better engineering and manufacturing processes and information systems to reduce product cost and improve product quality.

Most Recent Testimonials

"Bill has provided an excellent road-map for CAD/CAM integration. He has made a strong technical contribution in the development and implementation of various integration projects. … Strengths: technical knowledge & skills, dedication and loyalty, highly productive and proactive, vision and "big-picture" perspective, creativity and originality." - KW

"Bill Holmes [is the] chief architect of Convair's CIM Plan. His responsibilities include identifying faults and weakness in the traditional ways of performing aerospace product development, devising an improved process that will overcome the existing problems, and identifying opportunities to apply computer technology to the new streamlined process. Bill Holmes has been a leader and one of the most articulate spokesmen for the IMS team's efforts to devise plans and strategies for establishing the "new Convair". … He has conceptualized ways of organizing product and program information that hold the promise of enabling a responsive, efficient, and accurate product development life cycle at Convair." - RJF

"DEC and GD went through a prolonged period where we could not agree on an architectural approach. Bill essentially rescued the situation with his insights and ability to communicate difficult technical concepts. He made numerous presentations to various levels of people in both companies, and kept elaborating on the backplane concept until he had sold it throughout the Program.

"Bill … has an extraordinarily insightful understanding of the engineering end of the defense business. He is an excellent communicator of difficult concepts. He readily adapts to new technologies. … We need his brain to be applied to the problems on the manufacturing side of the business.

"Your memo was just excellent! Your frankness and honesty in packing those ideas and thoughts are to be admired. I had no idea that such a deeply philosophical and thoughtful person was lurking behind that friendly mask!" - PCM

"Well said, timely I might add, too!!! May your wonderful mind continue to chart a course within the nether world of CIM processes which no organization has ever gone before!!!" - JCM