TECHNOLOGY INSERTION: ESTABLISHING AN OBJECT-ORIENTED LIFE-CYCLE METHODOLOGY

John A. Anderson

Computer Sciences Corporation
CSC Technology Center
3160 Fairview Park Drive
Falls Church, Virginia 22042

ABSTRACT

Computer Sciences Corporation (CSC) is establishing a comprehensive object-oriented life-cycle methodology for Ada software development. The development and introduction of this methodology include many technical, managerial, and logistical challenges that go beyond creating and/or selecting software analysis and design techniques. Like any other large project, such an endeavor must be planned with a specific set of approaches and deliverables, milestones to measure progress, and evaluation criteria to determine quality. This paper identifies the components of a comprehensive life-cycle methodology and the necessary steps required to introduce it into an existing development environment. Recognizing that such a methodology cannot be created and established instantaneously, this paper suggests criteria for making trade-offs during the creation, introduction, and refinement of these life-cycle methodology components. Finally, the paper reports CSC's progress in developing and inserting its life-cycle methodology into practice.

BACKGROUND

Thorough research and experience can establish a set of techniques or tools that could greatly improve the software development process, increasing the quality of the product and the productivity of development staff. Unfortunately, the effective and efficient introduction of innovative technology into the mainstream of software development is as much a challenge as the creation of the innovation itself. For an organization to easily accept and apply new methods, techniques, or tools, they must fit neatly into the current method of operation and be reasonably similar to those they replaced.

The Ada language program is an example of the difficulty in introducing new technology. Although the Ada language [DOD83] supports powerful software engineering principles such as abstraction and information hiding, its early applications resulted in software differing little from that of FORTRAN or Pascal. These experiences signalled the need for more effective design methods to exploit the features of the language, and object-oriented design became in vogue. In recent years, large Government projects have continued to exhibit problems with Ada and object-oriented design, not because Ada and object-oriented design concepts are improper, but because they have not been effectively integrated with existing development standards and functional requirements analysis.

The Government and software contractors must recognize that efficient and effective software development requires more than establishing a set of independent methods and techniques to be applied at various phases of the life cycle. CSC is establishing a comprehensive object-oriented life-cycle methodology to control the complexity of software development, leading to the economic production of reliable and efficient Ada software. A comprehensive life-cycle methodology cannot be established instantaneously; CSC will achieve such a goal by incrementally creating and introducing the methodology into CSC's existing development environments and projects.

ADOPTING A STRATEGY

Software development periodicals, conferences, and forums present abundant powerful techniques, methods, and new approaches for solving the software crisis. Relatively few of these are effectively applied and adopted by the software development community at large. In fact, one source states that only about one per cent of organizations fully deploy a manual methodology [CASE90]. However, inadequate deployment is not necessarily a reflection of the technical merit of the methodologies. Achieving acceptance of new technology is as much of a challenge as developing the innovation itself. Project managers are rightfully hesitant to increase the risk on their projects by adopting techniques lacking a proven track record.

Creating a comprehensive methodology, introducing it into an existing corporate structure, and having it accepted by software developers are formidable and potentially expensive tasks to say the least. A powerful technique for risk reduction is to pattern the development plan after

successful case studies of other technology insertion programs. One of the most successful technology insertion efforts was Thomas Edison's introduction of the incandescent light bulb into American society.

The light bulb was an innovation that was not guaranteed success. At the time of its invention, electricity was commonly considered a mysterious and dangerous power. The introduction of electrical lighting into every home in the United States was considered incredible and impractical. Edison recognized the potential of harnessing such power and attacked the problem with a well-planned strategy. Reasonable milestones were identified to establish proof of concept (lighting of his laboratory in Menlo Park), to stabilize and support the technology (establishment of a company to invent and manufacture electrical generation, distribution, and application devices), and to demonstrate the practicality of deployment (electrification of the first neighborhood). As with innovative software techniques, the effort related to inventing the light bulb was minor compared to that of establishing and stabilizing the technology support. Edison's "90% perspiration" was invested in identifying and establishing the infrastructure necessary to support commonplace household incandescent lighting, thus feeding the "10% inspiration" of inventing the tools necessary to achieve his goal. This monumental effort spanned a broad spectrum of difficulty from designing and inexpensively manufacturing relatively simple appliances such as replaceable light bulbs and sockets, to inventing and installing more sophisticated apparatuses such as generators, meters, and distribution mechanisms.

Using Edison's success as a model for introducing an object-oriented development methodology, methodology investigations should apply the greatest proportion of the effort to establishing an infrastructure for its support. The development of the infrastructure will drive the technology: identifying specific technology support deliverables, assisting prioritization, and inspiring more innovations.

REUSE, REPLACE, RENEW

The world is just now recognizing how precious and limited our natural resources are, and that to survive we must recycle. Software projects constantly suffer from a shortage of resources, and management is similarly recognizing the necessity of reusable software components. However, software code reuse is only a small part of solution. To substantially reduce the effort (and thus the cost) necessary to solve large software problems, a much more global software recycling mechanism must be established. Software recycling must encompass not only code (or product) reuse, but all levels of information reuse. Most software development efforts include the expense of creating and establishing their software development infrastructure [HUMP88], a tedious, resource-consuming process that delays solving the software problem at hand. The establishment of a comprehensive object-oriented life-cycle methodology will substantially decrease this expense and allow personnel and capital resources to be expended more efficiently. Further, reuse is not a complete recycling method; a comprehensive methodology must also include a feedback mechanism that completes the cycle and guarantees continued process improvement.

Ironically, despite their overall goal for increasing productivity, methodology investigations often have even tighter budgets than software development projects, and thus a more intense requirement for information reuse in Freeman's categories of tech-transfer and development knowledge [FREE83]. The areas with the most significant potential within tech-transfer and development knowledge are process reuse and experience reuse. Process reuse relates to the definition of a developmental process and the techniques applied during software development, including the definition of "meta-products" such as deliverable documents, project standards, and support tools. Experience reuse is a tougher challenge, that of tapping the variety and depth of experience of the practitioners throughout a project or corporation.

PROCESS REUSE

Standards are not a result of reuse, reuse is the result of standards. Standards are the foundation of process reusability because they document the intended software process, and when effective can determine the quality of that process. An effective methodology must establish and maintain a set of standards to document its very definition. Technology insertion requires special attention to standards, because the standards are more likely to be volatile. Management and practitioners must be aware of the role of standards. On any project, standards must be considered a quality filter to identify deviations, not a constraint from doing the job correctly. When a technology is being introduced, any deviations are from an intended model, not from a perfected model. Quality assurance organizations must be especially careful to identify and evaluate the reasons for frequent deviations, determine if the standards really discriminate quality, and feed this information back into the standards definition.

Whether a software methodology is intended for Government projects or not, it makes sense to adopt or address Government standards and requirements. Government standards are defined to control expenses, and they result from years of experience and problem solving. Although far from perfect, Government standards represent a substantial investment of time. Developers should capitalized upon this experience.

Recently a standard software development process has been established in DOD-STD-2167 [DOD85] and revised in DOD-STD-2167A [DOD88a]. The life-cycle model described in DOD-STD-2167A establishes uniform requirements for software development that are applicable throughout the system life cycle and provide the basis for Government insight into a contractor's software development, testing, and evaluation efforts. However, although DOD-STD-2167A provides a standard vocabulary for the life cycle and defines the minimal requirements for recommended deliverable software development products, it was not intended to impose a development process upon software contractors. (To do so could significantly increase software development costs and inhibit innovative techniques.) The standard vocabulary and recommended set of deliverables must be (re)used as a framework for establishing a life-cycle methodology for economical reasons: alternate life-cycle models and vocabulary require substantial investment and justification on Government contracts, and addressing established standards provides a broader

base of potential (re)users. Unfortunately, DOD-STD-2167A's focus on products has resulted in software development being often viewed only in terms of the separate deliverable products for each individual life-cycle phase. The processes applied during software development are just as important as the products, and just as interrelated. A comprehensive methodology must define, reuse, and refine both the products and processes of software development and also transcend computer science to consider the issues related to managing the people, products, and processes applied during software development.

Similarly, software quality standards have been recently established in DOD-STD-2168 [DOD88b]. Although DOD-STD-2168 defines a framework for a software quality assurance program, it does not impose many specific requirements on the contractor. The framework requires a set of well-defined activities and products and requires evaluation criteria and responsibility to be assigned to each. Applying the DOD-STD-2168 framework to the activities and products of DOD-STD-2167A produces a voluminous set of requirements for a comprehensive life-cycle methodology. For example, Figures 1 through 3 illustrate the processes and products that must be defined for the Requirements Analysis through Detailed Design phases. In addition to these processes and products, a life-cycle methodology must also define the transition strategy between phases, ranging from the technical transformation of information between phases to the management of teams of practitioners changing activities at varying rates.

The final aspect of process reuse in a comprehensive methodology definition is automated support. Current trends in Computer Aided Software Engineering (CASE) support show promise, but an integrated support system will be necessary to achieve significant gains in productivity [CASE90]. An automated support system must collect and correlate the voluminous data about a system's requirements, design, and implementation; ensure the internal consistency of the products of each phase; assist transition between phases; maintain the connections among the information of all software phases; and monitor progress and assist management in strategic decisions. Until integrated support systems are achieved, a set of recommended tools or job aids must be created and applied. Practical examples include checklists for inspections and manual or semi-automated progress reporting mechanisms.

EXPERIENCE REUSE

Experience reuse, however difficult, can be achieved within a particular organization, and—in combination with process reuse—from one organization to another. Experience reuse is the most important mechanism for advancing and refining technology to achieve its goals. The essential requirements for experience reuse are communication, cooperation, and a recognition that practitioners are capable of controlling quality themselves.

Experience reuse begins with creating standards and procedures that constitute the new methodology, assuming that the methodology is based on past project experience. New technologies and approaches will change critical aspects of the development process; however, many traditional developmental processes and products must be reused until the new approach's impact can be determined. This reuse capitalizes on the experience built into those standards and demonstrates that process reuse in the form of standards is equivalent to experience reuse. Application of new technology also heightens the need for management experience reuse. Deploying new approaches effectively requires extensive experience to manage the trade-offs between strictly enforcing traditional standards and procedures and freely leaving innovation "to the experts."

Once the new methodology is defined for a project, management and technical staff must often be trained to ensure effective application. The expense of live, in-house industrial training can be justified more easily by exploiting the collective experience of the participants. Less expensive alternatives to live training, such as books or video tape, are static, limited in scope, one-way communication mechanisms that are seldom completed or taken seriously. Live training can and should encourage a controlled exchange of experiences and "war stories" to reinforce and illuminate the technology transferred. Because most of the components of new methods were designed as solutions to address common software development problems, these exchanges are easily initiated. Further, in-house training may be a rare opportunity for entry level practitioners to openly interact with their more experienced senior counterparts; the training session can be an environment for team building and solving problems related to the specific challenges of a particular project. Additionally, pilot training programs that introduce new methods and products are excellent for generating initial feedback on techniques, standards, and job aids.

As the new methodology is applied on a project, experience with the process and products must be fed back into the methodology. Feedback will determine the effectiveness of and changes required in the processes, products, and/or their media. The lessons of the "Quality Revolution" [SEID89] in manufacturing must also be reused and applied in the software industry. A culture that includes a systematic way for practitioners to contribute to product and process quality is essential for quality achievement, increased productivity, and improved developer morale. In essence, the practitioners themselves must become the methodologists and quality assurance specialists.

OPTIMIZATION AND TRADE-OFFS

The previous sections describe a monumental set of requirements necessary to establish a comprehensive life-cycle methodology. Creating and validating processes and products for a new methodology require experience, refinement, and time; all of these require funding. Fiscal responsibility requires that such IR&D funding be limited until the pay-off for such an investigation is proven. Prudent planning requires that the investigation of the methodology be carefully optimized to ensure the most critical areas are addressed first. No matter how well planned the methodology investigation is, opportunities and unexpected challenges will arise. This section identifies criteria for optimizing a methodology investigation and resolving the trade-offs in planning that will inevitably be required.

1. A "Best of Class" approach optimizes the initial construction of the

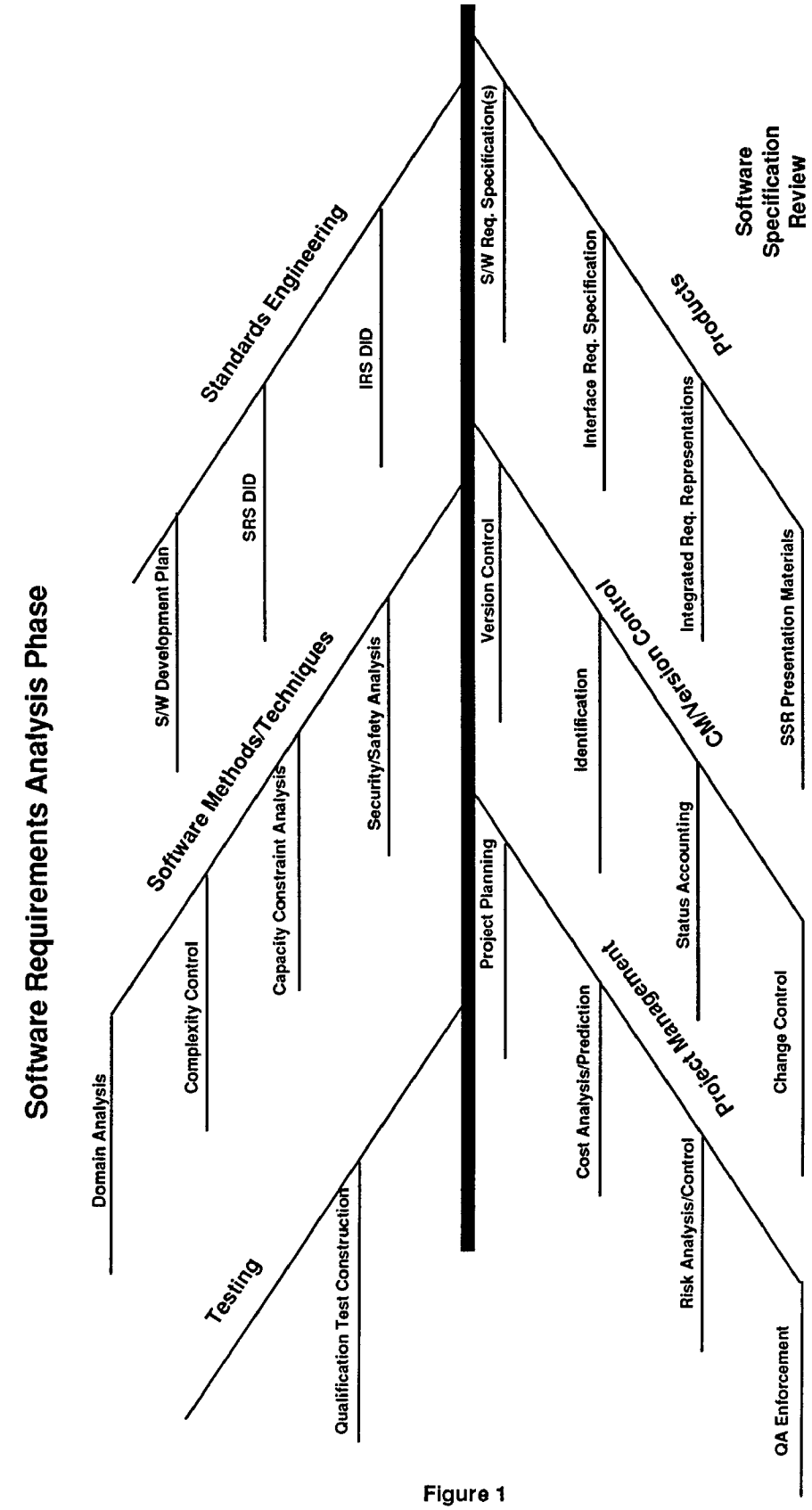
Software Requirements Analysis Phase
Standards Engineering
Software Methods/Techniques
Testing
Project Management
CM/Version Control
Products
Software Specification Review
S/W Development Plan
SRS DID
IRS DID
S/W Req. Specification(s)
Interface Req. Specification
Integrated Req. Representations
SSR Presentation Materials
Domain Analysis
Complexity Control
Capacity Constraint Analysis
Security/Safety Analysis
Version Control
Identification
Status Accounting
Change Control
Qualification Test Construction
Project Planning
Cost Analysis/Prediction
Risk Analysis/Control
QA Enforcement

Figure 1

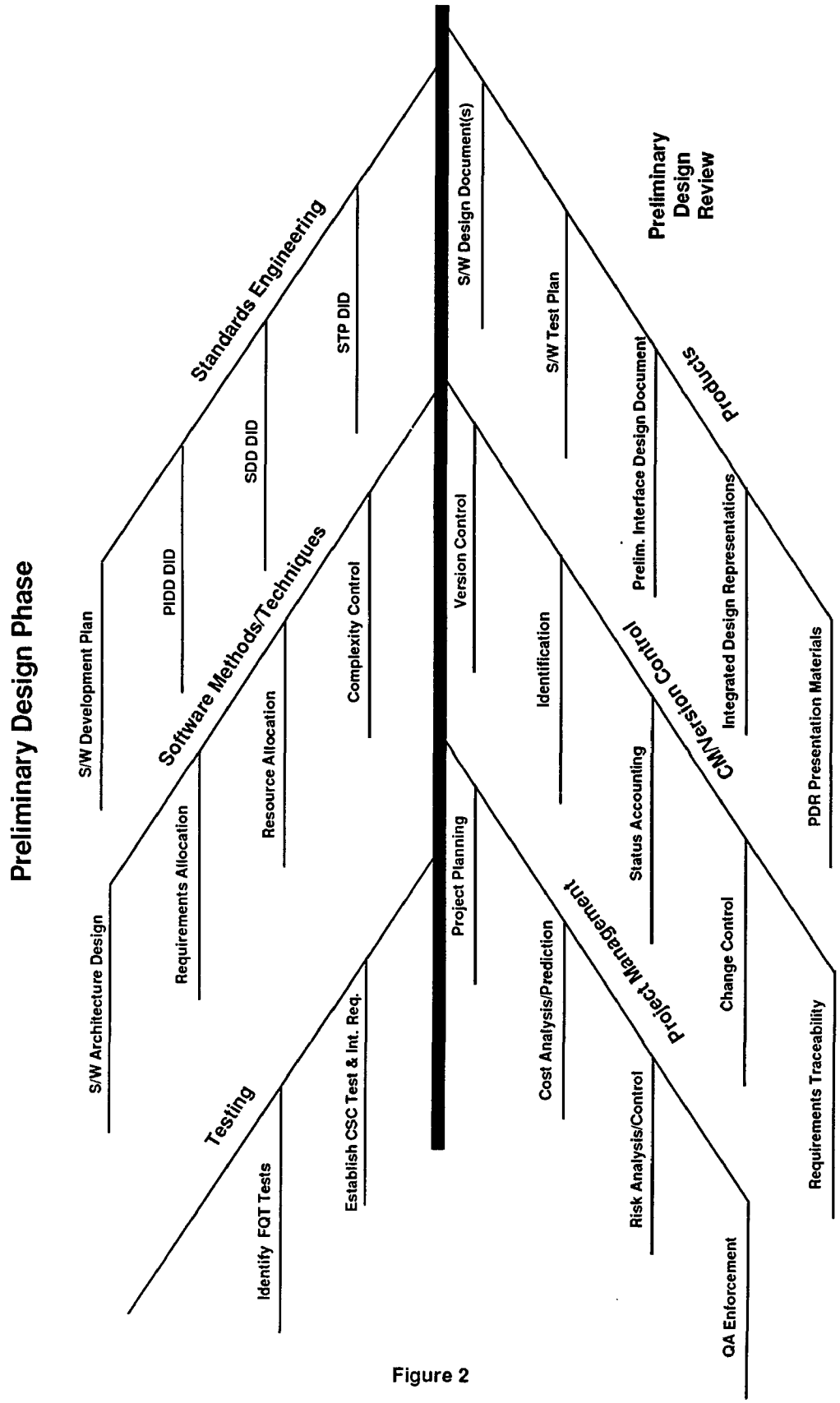
Preliminary Design Phase
Standards Engineering
Software Methods/Techniques
CM/Version Control
Project Management
Testing
Products
S/W Development Plan
PIDD DID
SDD DID
STP DID
S/W Design Document(s)
S/W Test Plan
Preliminary Design Review
S/W Architecture Design
Requirements Allocation
Resource Allocation
Complexity Control
Version Control
Identification
Prelim. Interface Design Document
Integrated Design Representations
PDR Presentation Materials
Identify FQT Tests
Establish CSC Test & Int. Req.
Project Planning
Cost Analysis/Prediction
Status Accounting
Change Control
Requirements Traceability
Risk Analysis/Control
QA Enforcement

Figure 2

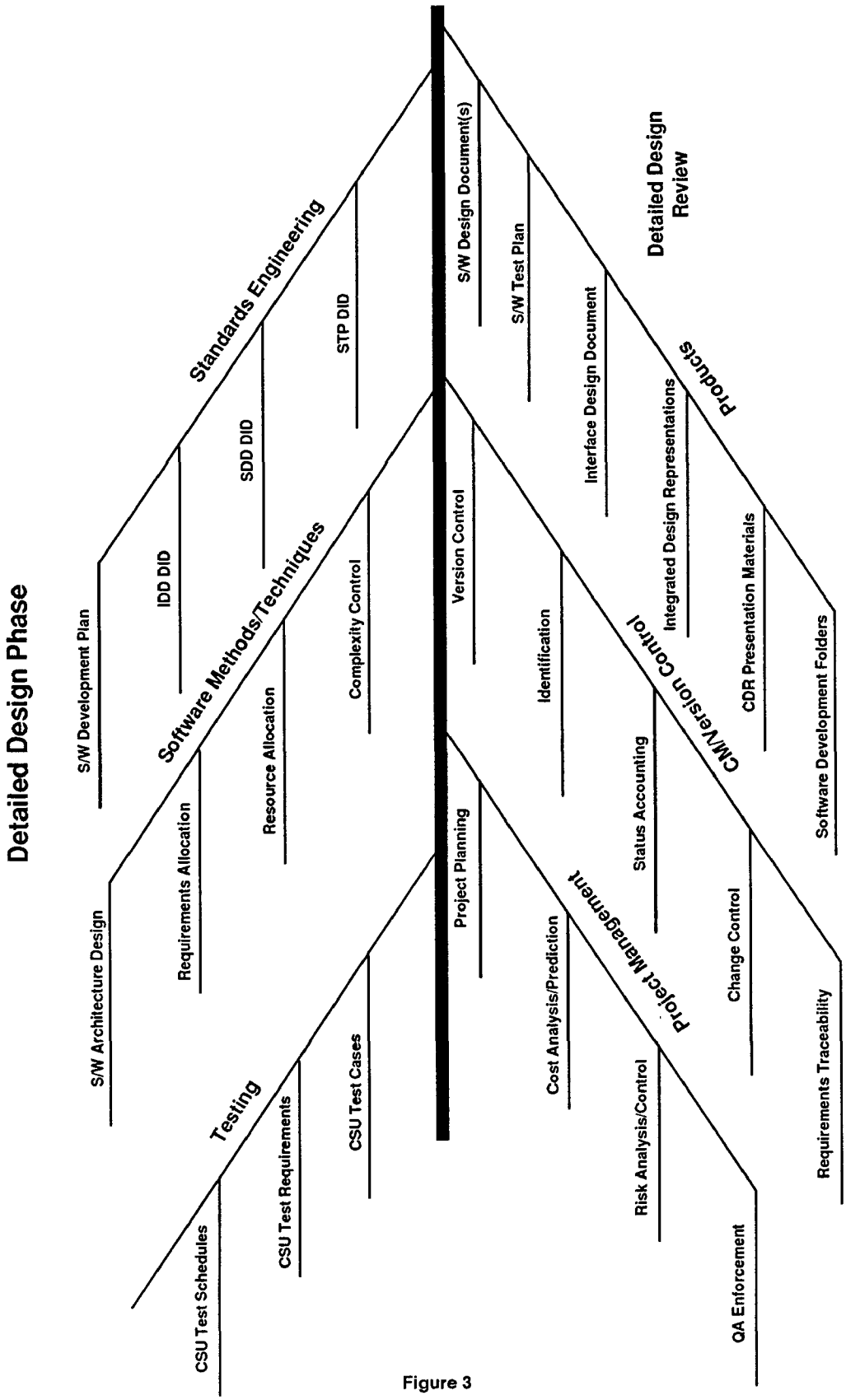
Detailed Design Phase
Standards Engineering
S/W Development Plan
IDD DID
SDD DID
STP DID
Software Methods/Techniques
S/W Architecture Design
Requirements Allocation
Resource Allocation
Complexity Control
Testing
CSU Test Schedules
CSU Test Requirements
CSU Test Cases
Project Planning
Cost Analysis/Prediction
Risk Analysis/Control
QA Enforcement
Project Management
Change Control
Requirements Traceability
CM/Version Control
Version Control
Identification
Status Accounting
Integrated Design Representations
CDR Presentation Materials
Software Development Folders
Products
S/W Design Document(s)
S/W Test Plan
Interface Design Document
Detailed Design Review

Figure 3

methodology.

Process reuse must occur even in the earliest phases of methodology investigation. The "Not Invented Here" stigma must be eliminated. The "Best of Class" approach, named after a similar manufacturing quality program at Ford Motor Corporation [SEID89], examines industry's techniques and products to determine their best subcomponents and tries to improve them. Where they cannot be improved, they are adopted as is. (The existence of standards and/or voluminous documentation for an approach indicates thorough research or application.) The selected or refined quality artifacts are then integrated with existing corporate standards into a life-cycle methodology. Because software development problems are wide-spread and being attacked throughout the industry, continued monitoring for additional innovations and better approaches must be an ongoing process.

2. The front-end of the development life cycle is the most important area to be defined.

An appropriate strategy must focus on the earliest, most critical phases of software development. The cost of error detection and correction escalates in orders of magnitude if left to the later stages of development and deployment [CASE90]. Further, the later phases of the life cycle manipulate the more concrete artifacts of software development such as code, test plans, and test data, and thus can be managed by traditional standards and practices. The front-end manipulates more abstract concepts such as requirements and design. Defining techniques, development products, and quality evaluation criteria for these otherwise intangible concepts solidifies the life cycle approach and permits further refinement and review.

3. A baseline configuration of the methodology must be defined and maintained as soon as possible.

As stated above, a methodology is defined in terms of its standards. Without the establishment of a baseline configuration, it is impossible to improve on the methodology, either by further augmentation or by refinement based on application experience. The methodology may initially be defined by references to articles and training materials. Eventually, standards must be constructed, documented, and maintained to establish a foundation for process reuse. This baseline configuration for software development support establishes the foundation for a repeatable development process as defined by [HUMP89].

4. Feedback is the most powerful mechanism for quality improvement.

Establishing an experience base for portions of the methodology is a higher priority than the full augmentation of the methodology. A fully documented methodology that has not yet been applied will seldom accomplish the quality goals intended and is unlikely to be adopted by project management. Incremental methodology development based on and refined by project experience will ensure a quality development process over the long run and identify detailed support requirements unanticipated by the original methodology developers. Projects applying portions of the life-cycle methodology can thus demonstrate the power and flexibility of techniques without committing to the risks of adopting an entirely new methodology.

Additional feedback can be obtained through internal and external publications, seminars, and review sessions. Proprietary issues must not interfere with the dissemination of the technology. Wide-spread distribution of the methods and techniques will stimulate necessary criticism, and, if the techniques and standards are effectively refined, acceptance. Broad acceptance of the methodology will further stimulate others to augment and possibly automate the methodology in a cooperative atmosphere.

5. Software management issues should be considered to be as important as technology issues.

Software development management and risk mitigation must be considered high priority areas for methodology development. The software crisis is by definition as much a management problem as a technology problem: "software that is nonresponsive to user needs, unreliable, excessively expensive, untimely, inflexible, difficult to maintain, and not reusable" [DIJK72]. Recognizing that software development with traditional methods will invariably be late and over-budget, few project managers want to be the first to adopt and apply a new methodology. New technologies increase risk and management will not adopt them if the project control mechanisms are ill-defined.

6. Deliverable documents are the most important standards to be defined.

The software documents required for delivery on Government contracts have rigorous minimum requirements. Constructing a mapping between the information elicited by the analysis and design techniques and their corresponding documentation vehicles, the Software Requirements Specification [DOD88c] and the Software Design Document [DOD88d], will identify additional methodology requirements. Upon completion, the existence of such a mapping lends credibility to the use of the techniques by demonstrating that they are necessary and sufficient to meet government requirements. Further, the mapping facilitates the construction of deliverable documents, allowing document production to be streamlined. Finally, the mapping establishes requirements for interfaces between CASE support tools and automated document generators.

The most comprehensive of DOD-STD-2167A deliverable documents is the Software Development Plan (SDP), as defined by Data Item Description DI-MCCR-80030A [DOD88e]. The SDP describes a contractor's plans for conducting software development, including standards and procedures, management, and contract work. Once an effective SDP is constructed referencing the documented standards in the methodology baseline, the methodology can be proposed on larger projects with reduced risk.

MILESTONES FOR TECHNOLOGY INSERTION

The previous sections outline the components of a comprehensive life-cycle methodology and the strategies for developing and introducing the

methodology into the mainstream of an existing organization. Such a plan is not complete without specific milestones and evaluation criteria to monitor its progress. This section describes milestones that can be used to determine the quality of the technology development and insertion.

Level 1: Technical Investigation

This is the initial stage of methodology development and the precursor to technology insertion. A project or organization enters this stage upon recognizing that there must be a better approach to solving its problems. Activities in this stage include literature searches, correspondence with consultants, selection of new methods and techniques, and experimental application of techniques. Some organizations may internally fund complete software development projects within the scope of technical investigations. Incremental methodology development requires continued concurrent technology investigation for some areas of the methodology, while others advance to Level Two.

Level 2: Minimal Project Support Established

This stage of technology insertion requires two conditions: enough documentation of the methodology exists to support projects outside of the methodology investigation, and a project is willing to accept the risks related to applying a new methodology or methodology subcomponent. If either of the pre-conditions is not complete, it will become quickly evident. If a project applies a methodology without sufficient support, the project management and technical practitioners will not waste time in saying so. And of course, not being able to convince project management and practitioners to adopt the methodology indicates that insufficient technical and management documentation exists to support them. Projects in this stage of technology insertion, although outside the scope of the methodology investigation, are often experimental or low risk efforts. Examples may include internal tool development or Government-funded methodology experiments.

Level 3: Assisted Methodology Application

This stage of technology insertion corresponds with the adoption of the methodology by a software project that must deliver a product to an outside customer. The adoption is based on a well-defined software development process, and the risk is mitigated by personnel support from the methodology investigation team. This liaison between methodologists and practitioners permits a steady stream of feedback and quick refinement of techniques, standards, and job aids.

Level 4: Initial Independent Methodology Application

This level of technology insertion signifies that the documentation of the methodology is sufficient for independent practitioners to apply it independent of the methodology investigation team. Level Four can only be achieved after Level Three insertion efforts have provided sufficient feedback to create a Repeatable Process for software development (as defined by Humphrey [HUMP89]). CASE and automated management support may exist for some or all of the techniques defined in the methodology. The methodology investigation should continue to correspond with these projects for two major reasons: in a crisis, the project may inappropriately abandon or alter methodology components; and independent application will identify additional support requirements that were not needed as long as the methodology investigation team was present and in control.

Level 5: Maturing Independent Methodology Application

At this level of insertion, technology is successfully in place. Projects that have adopted the development methodology independently refine it to their specific environments. Projects that can effectively control their projects while refining the processes to meet their own specific needs have achieved Humphrey's process maturity level of a Defined Process. At this stage, the methodology will continue to mature within the application project's organization.

PROGRESS AT CSC

CSC has sponsored the initial investigation for the comprehensive object-oriented life-cycle methodology through Internal Research and Development (IR&D). The first year effort focused on Level One technology investigation activities. CSC's IR&D effort (1988-89) proved that a comprehensive object-oriented life-cycle methodology for Ada software development is feasible and made excellent progress toward its establishment. The team investigated and evaluated work in object-oriented software development, selected and applied techniques of various approaches, and established the Object-Oriented Requirements Analysis and Design (OORA/D) method for Ada software. The greatest amount of resources was allocated to the selection and refinement of an Object-Oriented Requirements Analysis (OORA) method that creates multiple, interconnected models of system requirements, organized around the entities in the problem domain. As part of that first year effort, the OORA/D method was successfully applied to the development of a small Ada application in order to provide preliminary validation and to establish software products for further analysis. The first year results were published [ANDE89], and received substantial technical acceptance by the Ada community.

The second year of the project (1989-90) built upon the progress of the first year and eventually achieved limited Level Two and Level Three technology insertion. Standards for OORA processes and products were established based on prototype documents from the initial year investigation; projects outside the scope of the IR&D project applied the OORA/D methodology on their own test projects; and the design method was documented in the form of a workshop that permits project members to apply the method to their own project in place of contrived exercises. The design method was created using the "Best of Class" concept and thus can be considered a hybrid built on the process experience of many other projects. A particularly significant risk-reduction characteristic of the design method is that the entire design is graphically defined in terms of an abstraction hierarchy of interacting objects before the creation of Ada

specifications. This initial logical design can be constructed rapidly and provides early management insight into the size and technical intricacies of the application. The logical design is then transformed into a physical Ada software system; however, any programming language can be used.

Publication and application have elicited significant feedback to refine the OORA/D methodology. Construction of OORA standards was possible because existing documentation and training were available on the particular method selected before the techniques were applied. Standards were defined for the analysis process and the interim documentation produced during that process [NASA90], as well as the final DOD-STD-2167A compliant SRS that documents them as a Government deliverable [WARD90]. The object-oriented design workshop has been delivered, and the method was applied to several external projects. Although the hybrid nature of the design method deterred standards construction, the initial workshops contributed significantly to method refinement. The final deliverable design product is the most concrete and tangible form of a design because it immediately precedes Ada coding. Therefore it was possible to define management control structures for its development and maintenance [ANDE90] and to tailor the DOD-STD-2167A Software Design Document [DAHL90].

In keeping with the approach presented in this paper, the CSC Technology Center encourages discussion regarding this technology insertion approach and its comprehensive object-oriented life-cycle methodology. Managers, practitioners, and researchers may contact the author for further information, including access to standards, training, development support, and updates to the methodology.

ACKNOWLEDGEMENT

The author wishes to express his appreciation to Dr. Pamela A. Miller, whose inspiration and unyielding concern for quality made this paper possible.

REFERENCES

[ANDE89] Anderson, John, & Holland, Lance, & McDonald, Jane, & Scranage, Elaine, "Automated Object-Oriented Requirements Analysis and Design," Proceedings of the Sixth Washington Ada Symposium, June 26-29, 1989. Association for Computing Machinery, Inc.

[ANDE90] Anderson, John A. & Dahlke, Carl E., "Controlling Complexity in Ada Design Representation," Proceedings of the Ada-Europe Dublin 1990 Conference 12-14 June 1990, published in Ada: Experiences and Prospects, Cambridge University Press, Ada Companion Series, 1990.

[CASE90] Case, A., "What is an Analysis and Design Tool Worth?," Software Engineering Strategies, File: Strategic Planning SPA-238-394, Gartner Group, Inc., March 30, 1990.

[DAHL90] Dahlke, Carl & Anderson, John A., "Tailoring The DOD_STD-2167A Software Design Document To Support Layered Abstractions," Proceedings of the Structured Development Forum XI, April 30-May 3, 1990.

[DIJK72] Dijkstra, E. W., "The Humble Programmer" (Turing Award Lecture), Communications of the ACM, Vol. 15, No. 10 (October 1972), Association for Computing Machinery, Inc.

[DOD83] U.S. Department of Defense, Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A-1983. 17 February 1983.

[DOD85] U.S. Department of Defense, Military Standard Defense System Software Development, DOD-STD-2167. 4 June 1985.

[DOD88a] U.S. Department of Defense, Military Standard Defense System Software Development, DOD-STD-2167A. 29 February 1988.

[DOD88b] U.S. Department of Defense, Military Standard Defense System Software Quality Program, DOD-STD-2168. 29 April 1988.

[DOD88c] U.S. Department of Defense, Military Standard Defense System Software Development Data Item Description for the Software Requirements Specification, DI-MCCR80025A. 29 February 1988.

[DOD88d] U.S. Department of Defense, Military Standard Defense System Software Development Data Item Description for the Software Design Document, DI-MCCR80012A. 29 February 1988.

[DOD88e] U.S. Department of Defense, Military Standard Defense System Software Development Data Item Description for the Software Development Plan, DI-MCCR80030A. 29 February 1988.

[FREE83] Freeman, Peter, "Reusable Software Engineering: Concepts and Research Directions," Tutorial: Software Reusability, IEEE Computer Society Order No. 750, IEEE Computer Society Press, 1987.

[HUMP87] Humphrey, W. S., & Sweet, W. L., A Method for Assessing the Software Engineering Capability of Contractors, September, 1987, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213.

[HUMP89] Humphrey, Watts S., Managing the Software Process, Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[NASA90] NASA Goddard Space Flight Center, "Flight Software Systems Branch (FSSB) Software Standards and Procedures (FSSP) Manual" Standards 800 - 804, control number B202A3-25, March 29, 1990.

[SEID89] Siedor, Collin & Byerly, Steve, producers, "The Quality Revolution," (Television program), Dystar Television, Inc., 1989.

[WARD90] Ward, Elaine S., & Anderson, John A., "Documenting Object-Oriented Requirements Analysis Understandably for DOD-STD-2167A," Proceedings of the Structured Development Forum XI, April 30-May 3, 1990.