Take a Flying Leap: The Plunge into Object Technology

by Donald Firesmith

Preparing for your first project using a new and unfamiliar technology is always risky, and object technology is no exception. While most of the promised benefits have been achieved on one project or another, many projects have met with only limited success, and some have suffered major setbacks. Most of the problems can be attributed to a lack of proper preparedness, both by management and technical staffs. While there have certainly been technical problems with the maturity of parts of the technology (e.g., methods, CASE tools, compilers, object-bases[1]), the main risk factors have been managerial and are to be expected whenever people must master a new technology and paradigm.

1 The next generation of databases, specifically designed for storing objects and classes.

This article offers a brief road map for managers preparing for their first project using object technology. Figure 1 provides a list of interrelated steps to take to maximize the probability of project success.

OBTAIN INITIAL TRAINING AND EXPERT CONSULTING

Whenever managers and developers are transitioning to a new technology where previous training and experience is of

limited value and may even be misleading, the first thing to do is to get help. There is nothing like having an experienced guide when one is exploring new territory and wants to avoid pitfalls and get to the other side with a minimum of risk and worry. Although many companies are just now starting their transition to object technology, others have already successfully developed multiple projects using object-oriented (OO) methods, languages, and databases.

A reasonable investment up front in consulting and training is one of the key steps for risk management that a wise manager can make. This consulting and training should be directed at management and lead engineers who must provide the leadership required to transition their staffs rapidly to the new and better technology. It should emphasize the differences between the new and older technologies, the risks associated with the transition process (including the risks of not transitioning), and the initial risk management steps necessary to ensure a successful transition.

OBTAIN MANAGEMENT COMMITMENT

Transitioning to any new technology is expensive in terms of cost and schedule, and initial quality may suffer. The more

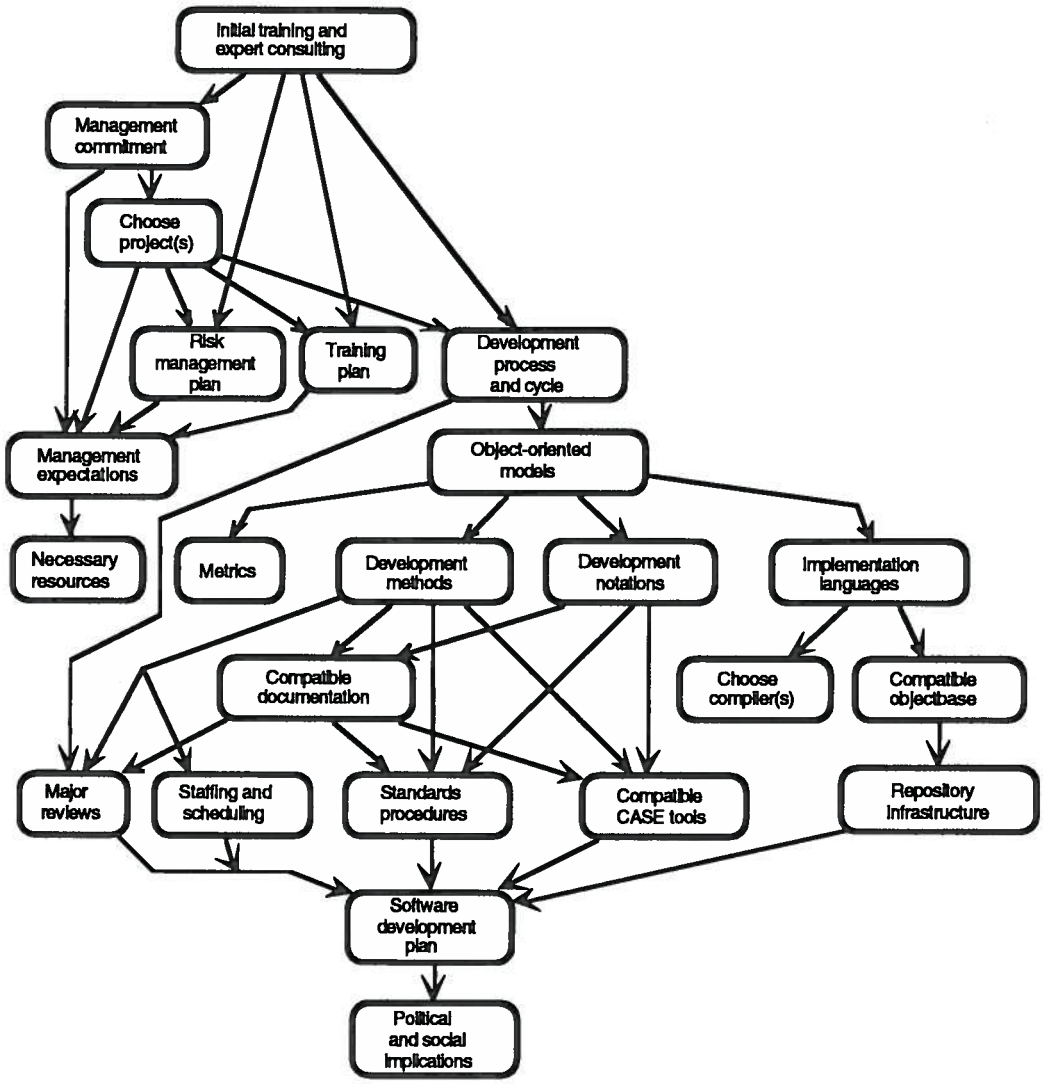

Figure 1: Order of management tasks

powerful the new technology, and the greater the differences between the old and new technologies, the more difficult the transition becomes. The difference between object technology and the older structured approaches is comparable to the difference between the early structured approaches and the undisciplined development they replaced.

Studies have shown that the full benefits of object technology gradually develop during the course of several projects, and developers (especially experienced ones) often require an average of three months to a year to become proficient. The payback is not something that occurs in the first six months. Project and corporate management commitment in terms of cost, schedule, and leadership is therefore critical if success is to be achieved in any reasonable time frame. The initial training and consulting should be used to help obtain the necessary

DF: I have a problem with OOD as OO development. I think the world sees it as OO design and that will confuse the issue.

DF: I don't know what this phrase means.

management commitment.

CHOOSE THE INITIAL PROJECT(S)

The choice of first projects is paramount. Because of the many inherent risks in using a new and unfamiliar technology, these projects should not be critical to the success of your organization. If a project is too important, there's a natural tendency to be overly conservative about the technology transition, which can be very dangerous. When crossing a narrow chasm, a step-by-step approach may be the surest and quickest way to the rocks on the other side, but sometimes the safest approach is to study the problem well and then take a flying leap to the other side.

For example, some managers have advocated the use of C++ as the OO language of choice for C programmers, reasoning that the transition will be easy and quick because their developers already know a subset of the new language. After mastering C++, they would introduce an OO development method and maybe think about objectbases once they have matured. Unfortunately, a recent study [7] has shown that of 23 C++ projects performed by seven firms, not one took full advantage of the OO capabilities (e.g., inheritance) of C++. Other experts have advocated a more radical transition strategy involving first teaching the developers a pure OO language (e.g., Smalltalk or Eiffel) to break the developers of their functional decomposition habits before venturing back to C++.

The initial project should be important enough to have sufficient corporate visibility to ensure adequate resources and influence on future projects, but not so important that the project manager is unwilling to take the necessary risks required for a successful transition. The project should be small enough and of short enough duration (e.g., 6 to 18 months) so as not to delay the transition of other projects, but not so small that lessons learned are inapplicable to real projects and that an inadequate number of developers obtain the necessary experience to support the next wave of OO projects.

The project should be in the same application domain as future projects that are slated to use object technology. The success of future projects, however, should not depend on the software produced by the initial project, because its quality will tend to be lower than that produced on later projects.

The initial training and consulting should be used to help determine the initial projects. Remember that management commitment to the proper use of object technology on these initial projects is critical.

DEVELOP THE PROJECT RISK MANAGEMENT PLAN

Developing a detailed project risk management plan is one of the first steps managers should take when preparing to transition to a new technology. Those that fail to plan, plan to fail. The initial training and consulting should be used to identify and prioritize all significant risks associated with the transition to object technology (e.g., limited experience, lack of training, learning curve effects, need for new CASE tools and compilers, maturity of the technology, appropriateness of the various methods to the application domain, risks of delay, and the limited window of opportunity due to the transition of competitors). The risk management plan should also address planned risk avoidance and management activities, including budget, schedule, and assignment to the responsible personnel. Management commitment must include the mandate and resources to identify and mitigate these risks.

DEVELOP THE PROJECT TRAINING PLAN

Very few companies have fully transitioned all relevant personnel to object technology. Most staff members have only a rudimentary understanding of the technology and its implications; many have extensive experience with older, incompatible methods (e.g., structured analysis and design) and require a certain amount of reeducation. The managers of initial projects should expect that significant training is necessary to bring people up to speed and to keep up with this rapidly evolving technology.

Don—comment from Ed Yourdon:
Please be more specific—which development cycles? Can you suggest books or references for the reader who wants more details? or make a forward reference to the relevant section in the article?

Article 5,

iterative and recursive

A detailed project training plan should be developed and implemented that incorporates initial classroom training to supply needed knowledge, continuing on-the-job training to develop necessary skills, refresher and advanced courses, an adequate supply of books and journals, and attendance at appropriate conferences and seminars. Management commitment must include the mandate and resources to train all relevant personnel properly: management, system and software engineers, testers, maintenance programmers, quality assurance and configuration management personnel, and relevant customer and independent verification and validation (IV&V) personnel.

CHANGE MANAGEMENT EXPECTATIONS

Experience has created certain management expectations regarding how a project should be run. Many managers expect some variety of the classic waterfall development cycle: software with a specific (typically functional decomposition) structure in terms of separate functions and data, traditional metrics (e.g., source lines of code) and reviews (e.g., preliminary design review, detailed design review), certain standard staffing curves and schedules, and so on.

Unfortunately (or fortunately if one considers the need to change the way we do business in light of the ongoing software crisis), many of these expectations are inappropriate on projects using object technology. Because of the importance of iteration and recursion, new development cycles are replacing the waterfall. Because different development methods produce different intermediate products according to different cycles and schedules, the number and type of formal reviews change.

Traditional development standards and procedures are inappropriate for the new methods and must be updated or replaced. Many of the CASE tools that have recently been acquired to support structured development methods are no longer useful and must be either upgraded or replaced. Since most OO development methods are highly graphical, large-screen workstations are replacing dumb terminals. Because of the use of OO (e.g., Eiffel) and object-based (e.g., Ada83) languages during analysis and design, compilers are needed much earlier during development.

The initial training and consulting can help transition management expectations and avoid problems caused by misplaced management fears.

OBTAIN THE NECESSARY RESOURCES

Many projects have suffered because the necessary resources were not expended to get the job done. The initial training and consulting, the risk management plan, and the training plan should all address the magnitude and optimal scheduling of the resources needed to transition to object technology properly. Management commitment means little if the necessary resources are not allocated. While managers of failed projects too often blame their staffs or the immaturity of the technology, it is management's responsibility to ensure that the right technology (e.g., CASE tool, compiler, object-base) is used and that the engineering staff is properly trained and capitalized to perform the tasks demanded of it.

EVALUATE, CHOOSE, AND TAILOR THE COMPATIBLE DEVELOPMENT PROCESS AND CYCLE

The waterfall development cycle and document-driven development process have long been recognized as suboptimal for the efficient development of quality software [1]. OO development methods tend to be highly iterative and globally recursive; this has a major impact on the overall development process and cycle.

Iteration is typically used to correct or improve existing software and occurs when some or all of the steps of the method are reapplied to the same product to modify it. Recursion is used to develop new software incrementally and occurs when all the steps of the method are repeated to generate new products, typically at the next lower level of abstraction.

Since many projects contain

No consensus exists with regard to the impact of object technology on function points although some research in this area has recently started.

Call Don Reifer

Don—comment from Ed Yourdon: What about function points? Are they applicable? Is IFPUG working to extend the concept of "object points"?

hundreds if not thousands of objects and classes, it is not practical to identify, analyze, or design them all at once. Global recursion (i.e., recursion that spans multiple activities such as requirements analysis, design, and programming) is typically used on projects employing OO development methods to develop incrementally small (i.e., 3 to 15) collections of objects and classes, variously referred to as subassemblies, subsystems, clusters, or subjects.

By using iteration and global recursion, software can be incrementally verified, validated, and developed at significantly reduced cost and risk. The question is: How much iteration and global recursion will be used? Some managers force OO development into the waterfall development cycle; others try to separate analysis and design, even though the same paradigm, method, models, and notation are often used for both. Still others use a totally iterative and recursive cycle (i.e., analyze a little, design a little, code a little, test a little, integrate a little) in which different development teams are simultaneously performing different activities on different subassemblies of objects and classes.

The choice of development process and cycle is critical because it will greatly impact many of the remaining preparation steps. The initial training and consulting should help managers and technical leads determine the optimum development process and cycle for the application size, overall schedule, and contractual requirements. Effort should be made to tailor customer standards if these will prevent the proper use of object technology.

EVALUATE, CHOOSE, AND TAILOR THE COMPATIBLE OBJECT-ORIENTED MODELS

Modeling is a fundamental part of software requirements analysis and design, especially on projects using object technology, because objects are software models of application domain entities. Object technology uses a different set of models than structured approaches use. While the names and number of models vary from OO method to method, there is significant overlap. These models document the static architecture and dynamic behavior of the software in terms of the objects, classes, subassemblies of objects and classes, resources of objects and classes (including attributes, operations, and exceptions), and the various relationships between them.

The software development team should evaluate the various OO modeling techniques available, choose those that are adequate and appropriate for their application domain, and, if necessary, tailor them for use on the project.

DETERMINE WHAT METRICS TO COLLECT

Because managers cannot control what they cannot measure [4], metrics are a necessary part of software engineering and engineering management. Metrics are especially important on projects using a new technology because past experience is of limited value. Unfortunately, the traditional metrics (e.g., source lines of code and number of units successfully tested) need to be either replaced (e.g., with number of classes, number of methods per class, levels of inheritance) or reinterpreted (e.g., classes rather than functions as units) on projects using object technology.

The metrics collection process and schedule of collection also require modification. Several military projects seeking to transition to object technology have suffered because of government requirements concerning the collection of inappropriate metrics tied to inappropriate development cycles. Project managers have even mandated inappropriate development methods based on obsolete development processes, cycles, models, and methods.

This risk factor is one of the most intractable of object technology. Whenever there is a technology transition, metrics (and associated cost and schedule models such as COCOMO) naturally suffer. Such models, used for estimating effort and schedule, are based on past data that is often inappropriate for the new technology. Unfortunately, little data has yet been collected and published

Don: does ADM3 need a trademark symbol in you bio? 

concerning OO metrics. Many of the relevant papers are suspect because they are not based on actual data. Managers should collect and analyze their own data, at least until an adequate sample size of relevant data points has been collected, analyzed, and published.

EVALUATE, CHOOSE, AND TAILOR THE COMPATIBLE DEVELOPMENT METHOD(S)

Over 30 different methods have been published for performing OO domain analysis, requirements analysis, design, and testing. Some methods appear to be primarily academic exercises, whereas others have been developed by experienced developers for use on real projects. Some methods are designed for sequential MIS software and address such issues as interfacing with relational databases, while other methods are oriented for real-time embedded applications and address such issues as concurrency and timing. Some methods are relatively simple, with limited capabilities, whereas others are quite sophisticated for use under all reasonable circumstances. Some methods are either quite old (and possibly obsolete) by OO development standards or so new as to be essentially ready for alpha testing and debugging. Other methods are into their third generation and as mature as any methods used in the structured world.

The project development team must therefore evaluate the different methods using numerous, appropriate evaluation criteria. The team must also choose methods (or a method) that cover the entire development cycle for use on the project. Finally, the development team should probably tailor the chosen methods for the specific application and the corporate culture of the developers. Because CASE tools should support the methods rather than drive the choice of methods, this step should occur prior to the choice of CASE tools (with iteration being used to adjust the methods as necessary). The project training plan may need to be updated to ensure adequate training on the project methods.

EVALUATE, CHOOSE, AND TAILOR THE COMPATIBLE NOTATIONS

Different OO methods often use slightly different notations for the same purpose (e.g., the semantic net of ADM3™ [5] is similar to the semantic networks of Berard [2] and the information model of Shlaer and Mellor [8]). Some of these notations are more expressive and easier to understand than others; some notations are supported by production-quality CASE tools, while others are not. The project development team should evaluate the available notations (e.g., for compatibility with the needs of the chosen method), choose an appropriate and sufficient set of notations, and (if necessary) tailor them for project use. The project training plan may need to be updated to ensure adequate training on the project notation set.

EVALUATE, CHOOSE, AND TAILOR THE COMPATIBLE DOCUMENTATION CONTENT AND FORMAT

Traditional documentation content and format requirements (e.g., see [6]) were typically developed to document procedural software that was functionally decomposed. They are usually not properly organized to document frameworks and assemblies consisting of subassemblies of objects and classes, nor are requirements to document the important relationships (e.g., aggregation, associations, inheritance) between objects and classes typically present.

Based on expert consulting and the initial training, the project development team should evaluate the currently available documentation standards, both traditional and object oriented. The team should then choose the appropriate documentation standards and tailor them for project use based on the choice of methods and notations. Note that the choice of the development process and cycle, as well as the choice and scheduling of the major reviews, will impact which parts of the associated documents must be completed for which reviews.

DETERMINE THE MAJOR REVIEWS

Because different OO methods and notations produce different intermediate products (e.g., documents, code) according to different processes and development cycles, the development team should determine the optimum number and types of reviews (e.g., incremental in process reviews), what should be reviewed at each major review, the acceptance criteria for passing each review, and the associated metrics for determining if the project is on schedule if the reviews are to be used as project milestones. The customer should be involved in this step due to the contractual nature of many formal reviews.

STAFFING AND SCHEDULING

Staffing and scheduling are affected by the iterative and recursive development cycle of most projects using object technology. Because the software is incrementally developed in an "analyze a little, design a little, code a little, test a little, integrate a little" manner, programmers and testers are needed earlier than is normal for waterfall projects. OO development allows managers to violate Fred Brooks's [3] dictum by adding additional staffing when projects get behind schedule. New development teams can be safely added as the number of subassemblies of objects and classes grows over time due to recursion (i.e., because of the low coupling among subassemblies due to the encapsulation of attributes and operations within objects and classes).

Scheduling is also affected by the iterative and recursive development cycle. The major in-process reviews become the intermediate milestones of each build or release, with progress tracked in terms of earned value (e.g., number of completed objects and classes). While the builds and releases can be scheduled as part of project planning, the scheduling of individual subassemblies of objects and classes is typically done on a case-by-base basis, because the subassembly architecture is not developed prior to the analysis and design of the objects and classes in the subassemblies. Objects and classes in the current subassembly must often be temporarily left incomplete because they must send messages to as yet unidentified objects and classes in lower-level subassemblies. Child subassemblies are thus typically incrementally identified and developed to supply these objects and classes and cannot be scheduled in advance.

Finally, managers should know that the traditional cost and effort models (e.g., COCOMO) appear to be inappropriate for estimating OO projects. As with any major technology transition, statistical models based on past data become suspect as the relevancy of the data diminishes, while new statistical models are of limited accuracy because of their small sample size.

DEVELOP COMPATIBLE STANDARDS AND PROCEDURES

Many organizations have organizationwide software development standards and procedures for use on individual projects. Unfortunately, many of these institutionalize older software development cycles and methods, such as the waterfall cycle and functional decomposition. These standards and procedures are therefore inappropriate for OO development and must be replaced. The corporate software engineering organization or the project software development team should develop new standards and procedures that are consistent with the chosen software development process, cycle, methods, notation, documentation content and format, major reviews, and implementation languages.

The initial versions of these standards and procedures should be developed prior to the start of the project and incorporated with the project software development plan for two reasons: first, to ensure that the development organization is properly prepared to begin the development effort (far too many organizations spend the first third of the project trying to determine how to proceed rather than actually developing the software) and, second, to allow the customer to evaluate the potential development organizations adequately.

Because new standards and procedures require use and developer feedback to become

production quality, planning should include both project validation and upgrades of the standards and procedures during the course of the project. The responsible parties will have to walk a fine line between fixing clear problems in the methods, notations, standards, and procedures and lowering developer productivity due to the constantly changing rules of the game.

EVALUATE AND OBTAIN COMPATIBLE CASE TOOLS

CASE tools should support the development process, methods, notations, and documentation content and format standards. Many CASE tools however support only the older software development approaches and are inappropriate for developing and documenting OO specifications, designs, and software. Too many managers have tried to use CASE tools designed for structured analysis and design for OO development, with disastrous results.

Fortunately, there are now many CASE tools designed for OO development, though they vary greatly in production quality, cost, support, and capabilities. Some only support a single OO development method and its associated notations, while others support over 20 different development methods and their associated notations. Because one can choose notations relatively independently of the methods and because most CASE tools do not enforce the actual steps of the methods, the software development team should evaluate the capabilities of CASE tools primarily in terms of the models and notations rather than the specific methods supported. The upper and lower CASE tools should be integrated and consistent with the implementation language and compilers.

EVALUATE AND CHOOSE THE COMPATIBLE IMPLEMENTATION LANGUAGE(S)

The implementation language should at least be object based (e.g., Ada83) or object oriented (e.g., Smalltalk, C++, Eiffel). While a complete comparison of the relative strengths and weaknesses of various languages is beyond the scope of this article, several factors should be considered:

- ☆ Application domain issues (e.g., real time, embedded)
- ☆ Project size and complexity
- ☆ Team size and distribution
- ☆ Compiler availability and quality
- ☆ Understandability and risk

For example, a pure OO language (e.g., Smalltalk, Eiffel) forces the software engineers to develop the OO mindset and is less likely to be misused than a hybrid language (e.g., Ada83, C++). Some languages (e.g., Ada83, Eiffel) support software engineering and are easier to read and maintain than others (e.g., C++). Some languages are better for embedded, real-time applications (e.g., Ada83) than others due to their support for concurrency, exception handling, and interrupt handling.

The project software development team should first prioritize the desired and necessary language features. It should then evaluate the various object-based and OO languages before choosing the optimum set for the project. Initial training and expert consulting can support this task, especially if the project development team has limited experience with object-oriented programming languages (OOPLs).

EVALUATE AND OBTAIN COMPATIBLE COMPILER(S)

OOPLs are usually supported by multiple vendors selling compilers and interpreters for multiple platforms and operating systems. There is also often a great discrepancy between the quality and capabilities of various compilers and associated tools such as debuggers, language-sensitive editors, class browsers for navigating inheritance structures, and the like. Vendors and compilers must be carefully and thoroughly evaluated.

Technical issues that should be considered include, but are not limited to, language features supported, validation to national and international standards, compiler and object code performance, capacity is-

sues, and integration with CASE tools. Nontechnical issues that should be considered include cost, vendor's reputation and longevity, vendor's market share and customer base, percent of vendor's business derived from OOPL support, vendor's strategic plans, vendor's financial status and resource allocation, and vendor's support. Many developers confuse the limitations of their compilers and associated tools with the limitations of the language and consequently have a harder time properly transitioning to full command of the language.

EVALUATE AND OBTAIN COMPATIBLE OBJECTBASE

There are a great many reasons to use an objectbase rather than a relational database on a project using object technology [9], including consistency with the object paradigm, compatibility with the OO software design, massive performance improvements when dealing with the complex data often found on projects using object technology, storage of the objects' operations as well as attributes, and so on. Several vendors currently sell production-quality objectbases, the next generation of databases designed for the storage of persistent objects and classes. The project development team should evaluate the various objectbases and, where practical, obtain an objectbase that is compatible with the implementation language and hardware platform that were chosen.

DEVELOP THE REUSE REPOSITORY INFRASTRUCTURE

One of the main advantages of and reasons for using object technology is the significant increases in reuse that it provides. However, this reuse is not automatic and requires significant planning and resources. Class libraries of low-level components are typically provided with the compilation system, but reusable application domain objects and classes must still be built from scratch. Projects need all the standard repository infrastructure associated with massive reuse in any language (e.g., configuration management, librarian, browsing software), as well as additional capabilities associated with object orientation (e.g., class browsers and configuration management that understands the implications of inheritance).

DEVELOP THE PROJECT SOFTWARE DEVELOPMENT PLAN

A detailed OO software development plan can be produced only after all the issues discussed in this article have been properly considered. It should be created as part of the initial proposal effort, delivered to the customer for developer evaluation, and maintained for the duration of the project. The risk management and training plans can be kept as separate documents or incorporated into the overall software development plan.

PREPARE FOR POLITICAL AND SOCIAL IMPLICATIONS

The productivity of the software engineers and the quality of their OO software is strongly determined by their position on the learning curve. It is therefore the responsibility of all managers to lead their staffs to the new, better way of developing software and to ensure that paradigm shift from functional decomposition (and event partitioning) to object orientation occurs as rapidly as practical. While some developers take to object orientation like ducks to water (especially those whose neurons have not been hardened by years of experience with older methods and languages), others must be dragged kicking and screaming into the next millennium. Some modern-day Luddites may actually seek to sabotage the technology transition which is perceived as threatening to many managers and developers.

Be prepared to recognize that political and social factors are often more important than technical ones when paradigm shifts are involved. Because ignorance often leads to fear, provide more training than has been the norm. Managers should be leaders rather than administrators. Plan to capture lessons learned in real time and spread success stories to motivate those developers who

must see the benefits of objects and classes with their own eyes. Use object technology as a way to empower the developer with the most potent technology currently available. Be flexible during the learning process and willing to upgrade the methods and tools as the technology changes. Above all, do not be afraid to use object technology as a powerful weapon in the ongoing war against the software crisis. After all, if you keep using the same old approach, you will get the same old result.

CONCLUSION

Many important factors must be carefully considered if adequate project planning is to occur. Because initial planning has many important ramifications for the development effort that follows and may have significant contractual implications, planning should take place early in the project and should be submitted to the customer organization as part of the project proposal. Since this planning is critical to project success, wise managers will seek expert advice before starting any project using a new technology. Object orientation is no exception.

REFERENCES

1 Agresti, William W., ed. *New Paradigms for Software Development.* Washington, D.C.: IEEE Computer Society Press, 1986.

2 Berard, Edward V. "Object-Oriented Requirements Analysis." Berard Software Engineering, 18620 Mateney Rd., Germantown, MD 20874, 1991.

3 Brooks, F. P. *The Mythical Man-Month.* Reading, MA: Addison-Wesley, 1975.

4 DeMarco, Tom. *Controlling Software Projects: Management, Measurement, and Estimation.* Englewood Cliffs, NJ: Yourdon Press/ Prentice Hall, 1982.

5 Firesmith, Donald G. *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach.* New York: John Wiley, 1992.

6 Joint Logistics Commanders Computer Software Management Subgroup. *Defense Systems Software Development.* Washington, D.C.: U.S. Department of Defense, February 29, 1986.

7 Deputy Assistant Secretary of the Air Force. Institute for Defense Analyses. "Ada and C++: A Business Case Analysis." *Communications, Computers, and Logistics,* Washington, D.C., July 1991.

8 Shlaer, Sally, and Stephen J. Mellor. *Object-Oriented Systems Analysis: Modeling the World in Data.* Englewood Cliffs, NJ: Prentice Hall, 1988.

9 Taylor, David A. *Object-Oriented Information Systems: Planning and Implementation.* New York: John Wiley, 1992.

Donald Firesmith is president of Advanced Software Technology Specialists, a Midwest consulting and training company that specializes in object technology. His book, Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach, *is scheduled for publication this fall by John Wiley. He runs 1-, 5-, and 10-day courses for both management and technical personnel in object technology.*

Before starting his own company, Firesmith was the software methodologist of a million-plus line project using object-oriented design. He is the developer of ADM3™, an object-oriented software development method for large, complex, and even real-time projects that is currently supported by two CASE vendors. He speaks regularly at CASE-World, SIGAda, and other industry meetings. He was founding chair of the SIGAda Software Development Standards and Ada Working Group and received an Air Force commendation for his work making DoD-STD-2167A more compatible with Ada and OOD. His current interests are in object-oriented systems analysis, domain analysis, and testing.

Mr. Firesmith can be reached at Advanced Software Technology Specialists (ASTS), 17124