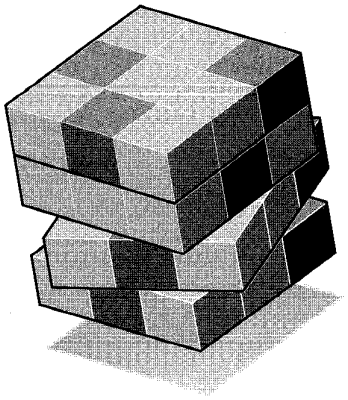

An OO Project Management Strategy

Babak Sadr
PARTA Corporation

Patricia J. Dousette
Litton Data Systems

This strategy solves many problems typically encountered during large OO projects by creating specialized work teams and by dividing projects into strategic and tactical areas.

Object technology is seen by many as the Yellow Brick Road to improved productivity, reliability, maintainability, and software reusability. Because of this, the software development community rushed to adopt OT.

As with all technological breakthroughs, OT has yielded successes and failures. The failures have been due largely to inexperience and a lack of planning by project managers for the transition to OT.

OT is tremendously helpful in achieving strategic goals and fulfilling business needs. However, because it is relatively new, it requires a transition plan, and its implementation must be closely supervised.

NEEDS ASSESSMENT

We developed our strategy to manage OO projects while working on a project that involved more than 100 engineers and development centers throughout the world. However, the strategy can be tailored for use in projects of any size.

Our strategy grew out of the challenges we faced, including the need to overcome coordination, logistical, and communication problems caused by having development personnel in various locations. In addition, to minimize development risks, we used Barry Boehm's spiral development model,[1] which calls for OO projects to be developed iteratively.

We also designed our strategy to solve the type of problems we, and many others, have encountered during OO projects[2]:

- A lack of coherency in methodology and process across the enterprise causes backward incompatibility and product-line integration problems. For customers, it reduces plug-and-play capabilities and makes it harder to upgrade to newer product lines.
- Architectural instability and incompleteness cause schedule delays and results in software components that must be redesigned for each development iteration. This also results in throwaway code and, therefore, low developer morale.
- A lack of staff familiarity with OT requires a substantial investment in training. A software developer typically needs six months of training to understand OO methodology, C++, and development tools. This must be included in the development schedule to avoid project delays.
- Staff inexperience with OT leads to an overly complex system architecture and implementation, which can degrade maintainability, performance, and reliability.
- Because it is new, distributed OT software development is hard to learn and use. Support tools are generally slow and unreliable, so their use will hurt system performance.

0018-9162/96/$5.00 © 1996 IEEE

- Lack of a code reuse strategy leads to redundant implementations.
- Developers who inadequately evaluate off-the-shelf products' applicability to their projects may spend time developing capabilities when existing products already provide them (thus reinventing the wheel). They may also use inappropriate product features in system design, which may cause performance and reliability problems.
- Not having an OO test environment and strategy affects software implementation stability and reliability.

PROJECT ORGANIZATION

We recommend that a project be divided into *strategic* and *tactical* processes. This division will determine the way you organize and manage the project.

Strategic process

This process addresses global concerns that have systemwide ramifications.[3] Strategic concerns include reuse strategy, the definition of project deliverables, and the system architecture, which guides system development. A technical manager coordinates strategic activities.

Tactical process

This process defines the development team's day-to-day operations.[3] Tactical concerns include software analysis, design, implementation, and testing, which are carried out iteratively until each object's dynamic and static behavior has been fully defined and implemented. A project manager coordinates tactical activities.

A project should be divided into strategic and tactical processes. This division will determine the way you organize and manage the project.

Staff organization

The traditional organizational hierarchy for a software development project does not work well for a large OO project, particularly one that has team members spread out in a variety of locations.

It is better to use a group of cohesive, specialized, and focused teams, each working on its own set of strategic and tactical activities. This divides and thereby reduces problems caused by the project's complexity and requirements, and also minimizes dependencies, which makes the teams more self-sufficient. In addition, this maximizes the degree to which tasks can be performed in parallel, thereby reducing dead time and increasing productivity.

PROJECT PLANNING

In keeping with our strategy, you should divide project planning into strategic and tactical focus areas.

Strategic planning

You use strategic planning to develop a stable system architecture, which provides the road map for developing an OO system.

Strategic project planning partitions project development into three phases. The setup phase and architecture definition phase generate the system architecture and requirements, while the development phase implements the software. Dividing your project into three phases partitions its logistical and technical needs, which also reduces risk factors.

Figure 1 provides an overview of strategic planning activities, showing the order in which teams perform various activities.

SETUP PHASE. In this phase, system engineers specify system requirements, which will be used in the next phase to design the system architecture. The rest of the development organization works on the project's infrastructural needs by, for example, undergoing training, preparing standards, establishing a documentation control system, and installing a configuration management system.

We assume here that project management, systems engineers, and support personnel (such as test, software quality assurance, and software configuration management) have already received OT training.

Since team dependencies are not critical in the setup phase, teams can move on to the next phase's activities as soon as they accomplish their objectives for this phase.

• *Systems engineering team:* The systems engineers define system requirements based on customer and marketing needs. For consistency, the format of the system requirements and specifications should be based on a development standard, such as the IEEE Software Standards[4] or the US Department of Defense's Mil-Std-498.[5] Although these standards do not explicitly support the OO paradigm, they add structure and definition to any development process.

Generally, there is no object model during requirements definition, so it is difficult and usually unnecessary to prepare object specifications at this time.

• *Development teams:* Team members are trained in the areas of analysis, design, language, and tools. Developers who do not have OO backgrounds generally need three months of training and three months to refine and sharpen their new skills.

They should attend a class to learn about OO analysis and design, and such methodologies and notations as the Unified Modeling Language, the Object Modeling Technique, Booch, and Real-time Object Oriented Modeling (ROOM).[2-3, 6-7]

An OO programming class should relate OO concepts to the features of the chosen language. Developers can use OO languages such as C++, Java, and Ada 95 to implement an object model. By taking a class that emphasizes OO programming instead of language syntax, developers will be better able to move from analysis to implementation.

In addition, developers will need software training from vendors on the use of software configuration management and CASE tools, which are essential parts of an organization's infrastructure.

For companies that will be using distributed objects and applications based on CORBA (Common Object Request Broker Architecture), formal training in these areas is also essential.[8]

Meanwhile, developers should be trained in software standards and processes, which define the rules for development activities.

We have found that about 80 hours of classroom train-

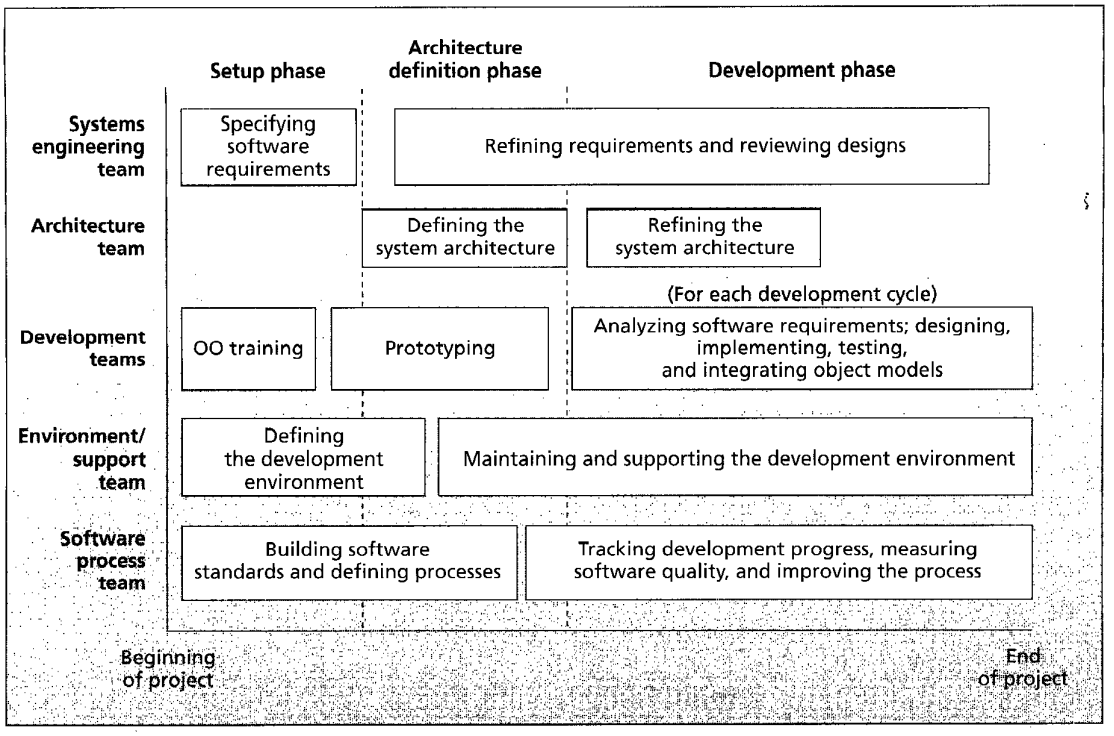

Figure 1. Overview of strategic development program. You divide strategic planning into three phases—setup, architecture definition, and development. You also divide the people working on your project into teams. The chart shows what each team works on during each phase, with the earliest activities on the left.

ing per person is necessary for a smooth transition to OT. Training costs can be significant and must be planned for.

• *Environment/support team:* This team installs and tests new computing systems, software products, and tools. These include the compiler and its supporting development environment, development frameworks and tools (such as GUI builders), an OO CASE tool, document generation software tools, and software for tracking and reporting problems. The environment/support team must integrate commercial software tools into the development environment and solve compatibility problems before formal development begins.

If the organization does not have an adequate communications system, the environment/support team must establish it now. E-mail, electronic file transfer, telephone conferencing and videoconferencing capabilities, and fax machines are critical, particularly when project members are in different locations.

The environment/support team also must establish an electronic documentation control system. A configuration management tool works well as the foundation for the documentation control system. Such tools help the organization control the data and documents that will be produced and disseminated later in the development cycle.

• *Software process team:* The SPT plans, tracks, and oversees software development. The team builds an OO project's development standards and processes around the OO paradigm. Development organizations must have formal software standards and processes to promote communication and avoid confusion in the execution of development tasks.[9,10] This improves the quality of products and organizational processes.

The SPT, which should consist of OO and software process experts, also specifies the criteria for measuring the effectiveness of managerial and technical activities.

Specifically, the team works on software-design document formats, which capture an object model's dynamic and static behavior. Their content should be object-focused rather than procedural, in contrast to traditional design documents.

The SPT also works on the organization's software coding standard, which creates a homogeneous style and presentation format for all program source files and which can support a formal test environment by creating test interfaces that are not intrusive.[11]

By standardizing source file content, layout, and documentation style, team members can use software tools to generate high-level documents, such as a software reference manual, from source files. These tools streamline the documentation process by incorporating function comment blocks in source files and by extracting and exporting applicable information to a word processor file.[11]

In addition, the SPT builds software test standards, which identify the testing strategy and environment for deliverable software. The team also specifies development processes by identifying the organization's teams, membership, roles, responsibilities, and intergroup relations.

ARCHITECTURE DEFINITION PHASE. In this phase, the system architecture is developed from the system requirements. A specialized team of systems engineers and developers focus on defining a stable system architecture.

Initiation of system development without this stable framework greatly increases development risk and leads to quality-related problems and nonreusable components.

While the architecture team is designing the system architecture, other teams are preparing for the development phase. In addition, software processes, standards, and development methods are prototyped and instantiated. This refines key processes, reveals obstacles and problems, resolves organizational and staffing needs, and gives development teams more experience with OT and support tools.

• *Architecture team:* This team consists of systems engineers and designers who analyze system requirements and define the system architecture. By defining the major software components (such as subsystems, modules, and their interfaces), the system architecture defines the framework from which development teams will work. For projects based on Boehm's spiral model,[1] project management uses this framework to specify each development cycle's software deliverables.

• *Development teams:* These teams develop software prototypes. For instance, one team may create a preliminary GUI specification.

Because each phase builds on the previous phase, the project must have a technical management team to prioritize and focus on each phase's key areas.

By creating prototypes, developers gain experience with OO analysis, design, and programming, as well as support tools. Although these work products are not reusable, the process lets developers validate basic concepts in key areas of the system and generate early feedback on technical issues and development processes.

• *Environment/support team:* This team continues to establish a mature development environment, and defines and develops the environment for unit and integration testing.

• *Software process team:* This team refines software standards and processes by validating them. Selected development teams test the standards on a limited basis. For example, the GUI team will develop the software for the GUI prototype based on the specified coding standards. This approach provides early feedback on the effectiveness of the standards and processes.

DEVELOPMENT PHASE. Using the system architecture, the technical management team begins a series of development cycles, using Boehm's spiral model.[1] For each development cycle, management identifies subsystems and modules that must be developed, tested, and integrated. Meanwhile, processes and standards are further refined.

At the end of each development cycle, project teams should meet to discuss the obstacles they encountered and the lessons they learned. This creates a system for tracking problems and resolutions.

• *Systems engineering team:* Systems engineers turn their attention to refining and updating software requirements. They also determine whether object models created by the development teams comply with system requirements.

• *Development teams:* During each development cycle, these teams analyze the software requirements and design object models for the architectural components on which they are working. Peer reviews are conducted to ensure that each component design provides the features and interfaces required to meet the client component's needs.

The development teams also implement object models and subject them to unit and integration testing.

• *Environment/support team:* This team provides the traditional system "housekeeping" and administrative support, and also maintains configuration management of object models, documentation, and source code.

• *Software process team:* This team performs risk assessment for future development cycles, tracks the progress of development, and collects data for measuring software quality.

Tactical planning

Because each phase builds on the previous phase, the project must have a technical management team to prioritize and focus on each phase's key areas. Companies will decide many of the details of tactical planning based on their standard operating procedures. However, some general principles apply.

SETUP PHASE. Without proper logistical support by the environment/support team, a project may encounter severe configuration and integration problems in the development phase. Meanwhile, the software processes that the SPT is putting in place must streamline development by specifying only processes and tasks that are necessary to enhance work on the project. These processes must be tested on a limited basis and then must be refined before they are fully adopted.

ARCHITECTURE DEFINITION PHASE. Project management must make sure the architecture team addresses nine key issues. This will refine the system architecture and establish the details of the software architecture.

• *Performance:* For real-time applications, an object model must encompass the system's timing aspect, which includes the real-time computational and operational requirements for the applicable subsystems and modules.[6] The development teams accommodate performance requirements in the component design, thus minimizing the need for subsequent redesigns and modifications.

The software-development platforms, tools, and methodologies used must support the system performance requirements. For example, you must determine the effect of a CORBA platform on a distributed application's performance prior to selecting the platform.

• *Error handling:* When an error occurs, the objects within a module may need to notify objects in other modules. Sharing and propagating errors is part of the modules' design interface, which means error detection and recovery are critical. Development teams can accomplish this by using a systemwide interface provided by a standardized error detection and reporting mechanism.

• *Fault tolerance:* Using the system requirements, the architecture team specifies fault modes and identifies the affected modules' behavior.

• *Concurrency:* In the case of distributed systems, the object model must address concurrency issues and incor-

porate strategies for avoiding deadlocks caused when multiple processes try to access an object.

• *Connectivity:* A network interface may affect the system architecture. By considering data bandwidth and related connectivity issues during high-level analysis and design, the architecture team can avoid architectural design flaws and shortcomings.

• *User interface:* An easy-to-use interface is a key marketing requirement, so the user interface's design should start in the early stages of system development. This lets customers work with the interface early enough in the process to provide useful feedback. This also gives the interface time to evolve and mature before reaching the market.

• *Off-the-shelf products:* The use of these products can significantly reduce development time. In addition, the use of generic interfaces can minimize dependencies on vendor products.

A project team should learn about off-the-shelf operating systems, hardware platforms, development tools, software development environments, databases, and other tools that can be used on the project.[2]

For the software components that will use off-the-shelf products, the team must specify design requirements and guidelines that will minimize dependencies between the products and the components under development. This will minimize the number of changes that product upgrades will cause.

• *System requirements traceability:* Analyzing requirements and verifying a design is tedious in large-scale system development. Manual verification is neither feasible nor desirable. Instead, you should use a database to map system and software requirements to the appropriate modules. Using an off-the-shelf database, the requirements are documented in requirements traceability matrix tables (see the sidebar "Requirements traceability"). These tables typically list, for each testable requirement, its source, its title and description, the design component to which it is allocated, the class and object that implements it, and the test reference that verifies it.

The quality assurance team also uses the matrix to verify object models' adherence to system requirements. Most importantly, by specifying relevant requirements for a module, the matrix lets development teams focus their OO analysis and design on the applicable requirements.

• *Interoperability:* To achieve interoperability, the common features and capabilities of a company's product lines are specified in a set of core components. Using core software components in various products enhances software reusability and enterprise coherency.

DEVELOPMENT PHASE. In the development phase, the development teams develop an object model for their assigned modules and subsystems. Because this phase can be divided into several development cycles, you must institute appropriate reviews to ensure coherency between the object modules developed in each cycle.

• *Design coherency and reuse strategy:* You should adopt a reuse strategy early in the design process. For example, you should review the object models created by different development teams for common design patterns.[10] The creation of common libraries could eliminate redundancies and enhance coherency across module boundaries.

Because developers use an iterative approach to design and implement OO projects, you should review the design of new modules in terms of previously developed class libraries, to identify similarities and differences.

Changes to an existing class library may require substantial interface and design changes. You should evaluate the extent of the redevelopment, retesting, and revalidation that would be necessary to determine whether it would be better to modify and reuse a class library or to develop a new library.

• *Standard class libraries:* Some classes in the model can be implemented by using standard class libraries. In terms of maintainability, testability, reusability, and portability, this is better than using custom implementations.

• *Legacy software:* Using existing non-OO software libraries eases a development organization's transition to OO methodology. An OO software wrapper can use an adapter class to hide the use of a non-OO legacy component in the system implementation. This adapter class does not provide any additional functionality except for minor and hidden data transformations that its member functions may perform to accommodate the use of existing products.[12] If the legacy software implementation is later replaced by an internal implementation or another vendor's tool, the adapter class keeps this from affecting the rest of the software design.

Requirements traceability

Requirements traceability is an important part of any OO development strategy. You use the process, which begins in the requirements analysis phase, to verify the correct and complete implementation of the system software.

Requirements analysis allocates system requirements to software components. The correct allocation of these requirements can be documented using a commercial database that produces a requirements traceability matrix (RTM). The database provides automation and flexibility.

The RTM provides the project's design and implementation history and must be maintained for the entire project. In the matrix, you list information for each testable requirement, such as the requirement's source, its title and description, the design component (such as the subsystem or module) to which the requirement is allocated, the class and object that implements the requirement, and the test reference that verifies the requirement.

The matrix specifies relevant requirements for a module and thus lets developers focus on the applicable requirements. The RTM also lets the quality assurance and test teams verify whether object models adhere to system requirements.

If an RTM is not used, the requirements traceability and system verification processes will be disorganized and probably ineffective.

OUR STRATEGY IS BASED on our experiences and is not meant to be used like a recipe in a cookbook. You must build on this blueprint and customize it to meet the needs of your projects and your work environment.

In addition, if you are considering an OT project, you should hire an industry expert on OO adaptation and training. This expert can help design a transition program for your company.[2] ■

References

1. B. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, May 1988, pp. 61-72.
2. M.F. Fayad, W. Tsai, and M.L. Fulghum, "Transition to Object-Oriented Software Development," *Comm. ACM*, Feb. 1996, pp. 108-121.
3. G. Booch, *Object-Oriented Analysis and Design with Applications*, Addison-Wesley, Reading, Mass., 1994.
4. *IEEE Software Standards*, IEEE, Piscataway, N.J., 1994.
5. *Mil-Std-498, Military Standard: Defense System Software Development*, US Dept. of Defense, Washington, D.C., 1995.
6. B. Selic, G. Gullekson, and P. Ward, *Real-Time Object-Oriented Modeling*, John Wiley and Sons, New York, 1994.
7. G. Booch and J. Rumbaugh, *Unified Method for Object-Oriented Development Documentation Set, Version 0.8*, Rational Software, Santa Clara, Calif., 1995.
8. *Common Object Request Broker Architecture (CORBA): Architecture and Specification, Version 2.0*, Object Management Group, Framingham, Mass., 1995.
9. *ISO-9000, International Standard: Quality-Management and Quality-Assurance Standards*, ISO, Geneva, 1987.
10. M. Paulk and C. Weber, *Key Practices of the Capability Maturity Model*, Software Eng. Inst., Pittsburgh, 1993.
11. B. Sadr, *Fundamentals and Applications of Object-Oriented Programming Using C++*, UCLA Academic Publishing Services, Los Angeles, 1995.
12. E. Gamma et al., *Design Patterns: Elements of Object-Oriented Software*, Addison-Wesley, Reading, Mass., 1994.

Babak Sadr *is an electrical engineer at PARTA Corp., a software development consulting and research company. He also teaches OO-related classes at UCLA. His expertise is in software development, parallel processing, and OO design. He received a BS and an MS in electrical engineering from the University of Southern California.*

Patricia J. Dousette *is a senior technical staff member at Litton Data Systems, where she is a member of the Advanced Products Group. She also teaches programming language courses at UCLA. Her expertise is in software process definition and instantiation. She received a BS and an MS in mathematics from California State Polytechnic University, San Luis Obispo, and California State University, Los Angeles, respectively.*

Contact Sadr at (818) 889-5333 or Dousette at (818) 597-5388.