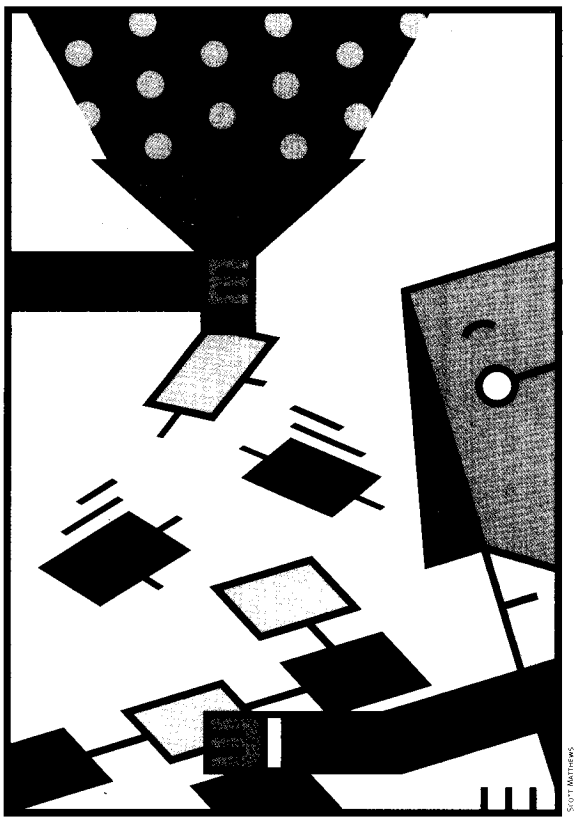
The OOP survival guide
Argila, Carl A
Computerworld; Aug 22, 1994; 28, 34; ABI/INFORM Global
pg. 89

In Depth

●REPLACING 1s and 0s with objects is easy. Mastering the mind-set for object-oriented programming is hard. Here to smooth your way is

THE OOP SURVIVAL GUIDE



SCOTT MATTHEWS

So you've decided to abandon traditional application development methods and take on new objected-oriented ways. Before you begin, bear in mind you're facing far more than a technical challenge. Once your developers master the programming skills and languages that object-oriented projects demand, you will need to create a workplace that will help them use those skills effectively.

By CARL A. ARGILA

That's not easy to do. You are likely to discover, just as Niccolo Machiavelli did, "*There is nothing more difficult to take in hand, more perilous to conduct or more uncertain in its success than to take the lead in the introduction of a new order of things.*" To help you take the lead in the new object-oriented order of things, here's a 12-step survival guide.

STEP 1

Decide why you are making the move

THE KEY REASON TO INTRODUCE object-oriented programming (OOP) methods is to reduce the time it takes your development staff to create business applications over the long haul. At the heart of any OOP strategy is the reuse theory: The objects and systems designed for one application can be redeployed in new configurations for others. Bear in mind that you won't reap "reuse" benefits until the second or even third project. Once you have articulated this message to the development team, put it in writing. Make it no longer than one page.

OOP SURVIVAL GUIDE, *page 92*

OOP SURVIVAL GUIDE, *from page 89*

STEP 2

Accept that the technology is here to stay

NOW THAT YOU ARE ADOPTING OOP methods, it's highly unlikely that you'll revert to traditional ones. Make sure your application developers know this. Because that may not go over well, I advise a gentle but firm approach. Don't dictate, "Thou shalt use objects — or else." Instead, ask each programmer to generate a list of OOP pros and cons, then meet with each member individually to discuss items on their lists. Use this information to create well-balanced project teams, made up as evenly as possible of pro and con members. In other words, don't put all the negative people in one group. Address in a group meeting the top con items voiced by staff members.

STEP 3

Take stock of your assets

OOP REQUIRES SPECIFIC SKILLS. Which ones are available in your organization, which ones can your staff acquire through training and which ones do you need to bring in from outside? Know the answers to these questions before you replace your Cobol-proficient staff with an army of Smalltalk or C++ programmers. Your staff is likely to have a wealth of knowledge about the business units they serve. Their expertise in communicating with users and capturing business knowledge is crucial to the project's success; you may decide to keep them on board for that part of the process and outsource the actual programming effort.

STEP 4

Don't develop a full-scale pilot project

FULL-SCALE PILOT PROJECTS take a long time and can cost hundreds of thousands of dollars. If they fail, companies often abandon — for good — the very technology the pilot intended to promote.

To minimize the likelihood of that, set up what I call a "symbiojet." This minipilot project of sorts has a symbiotic relationship to an existing mission-critical application, such as a title and escrow management system that could benefit from an object-oriented database. The database portion of the system could be spun off as a symbiojet. If the object-oriented project doesn't work the first time around, you can fall back on the existing relational database. The point: Choose a project that can fail without ruining the company.

STEP 5

Establish meaningful metrics

SUCCESSFUL PROJECTS are well-monitored. You need to measure the progress, quality and efficiency of your OOP project on a regular basis, then report back to those involved. Doing so helps spot trouble before it gets out of hand. Steady progress reports boost morale.

There are many ways to measure progress. One project manager I know established a metric based on the number of compiler invocations per person, per module and per unit of time. As the staff develops a module and the module nears completion, this metric should approach zero, and modules that don't conform to this pattern should be singled out for attention. Note that you have to use metrics carefully, making sure you monitor the right things. For instance, using the metric described above to check on attendance would destroy its value, not to mention demoralize your staff.

STEP 6

Expect resistance to change

IN REPLACING OLD DEVELOPMENT METHODS with new ones, you are likely to encounter all manner of behaviors.

Expect to see "creative avoidance," i.e., "We can't do this. We don't have the right CASE tool, the right methodology." In some cases, a committee will form with a mission to identify that perfect tool or flawless methodology. Be aware this may be a stalling technique. Another strategy: "malicious compliance," i.e., "You want objects, I'll give you objects." In this insidious game, players work to make the project fail, taking every instruction literally and looking for loopholes.

Note that creative avoiders are usually well-intentioned. Training will increase their comfort levels. As for the others, just hope their behavior stops.

STEP 7

Focus on object-oriented programming alone

DON'T INTRODUCE OTHER TECHNOLOGIES at the same time you implement OOP. Doing so is asking for trouble.

A few years back, one of my clients made this mistake. In launching a major telecommunications project, it mandated the use of computer-aided software engineering (CASE) tools for application development. Simultaneously, it switched from Cobol to C and from a mainframe to a Unix platform, replacing a single processor with a client/server network. The project failed. The compiler wasn't working. There was a problem with the client/server transaction processing system. The reasons were endless, and all legitimate. If it's imperative — and sometimes it is — to introduce two technologies simultaneously, run concurrent pilots. Integrate them only after you understand both thoroughly.

STEP 8

Solicit advice from experienced veterans

WHILE IT'S TRUE THAT companies have just begun to take advantage of object-oriented technology (see chart at right), that's no excuse for wandering aimlessly through uncharted territory. Training programs and seminars devoted to the topic are full of people who may have already solved the problems you are facing. Network with them. Join discussion groups on the Internet. Call your local IS association. Contact your vendor. There's help out there, and you have to look for it. The point is this: Learn lessons from those who have completed successful OOP development efforts.

STEP 9

Create your information model with care

AT THE OUTSET OF THE PROJECT, you need to create an information model, which will establish a basic set of objects on which to base future systems. It's critical to get the model right. The following story illustrates why:

A European country replaced a legacy system designed to track government benefits with an object-oriented application. The programmers identified object classes with names such as "pension" and "beneficiary." The system worked well until it was time to update the "legislative rules" embedded throughout. Programmers had never created a legislative rules object class. So instead of issuing one command to perform a global update, they had to change each one step by step. That defeats the point of OOP.

STEP 10

Specify a time frame and stick to it

THE PROJECT DELIVERY DATE never slips. Period. If it does, the technology you are trying to promote will be deemed a failure. If you can't meet the deadline you have established, reduce the scope of the project. That may mean reducing, limiting or restricting the scope of the features offered. The important thing is to deliver the critical pieces on time.

Getting the first project out when promised is a real morale booster and will serve you well in the long run. Remember, your long-term goal is to develop all your applications using object-oriented methods.

Looking Ahead: Object-Oriented Programming

1993 *to* 1997	• Applications based on Microsoft's Object Linking and Embedding appear. • Visual Basic is the most common development environment for object-oriented programming.
1994 *to* 1999	• Wide use of OOP begins. • Systems integrators deliver applications based on the technology. • IS departments begin to outsource OOP projects.
1996 *to* 1999	• Object infrastructure appears. • OOP is commonplace.

Source: Forrester Research, Inc., Cambridge, Mass.

STEP 11

Promote the new culture you have built

NOW IS THE TIME to establish the basic mechanism for reusing objects in your next programming effort in your organization. To be truly effective, that effort needs to be coordinated at the corporate level. It doesn't begin so much with reusing the objects your programmers have developed as with the corporate commitment to continue using object-oriented technology.

How do you make this happen? One data services company, for instance, instituted a program in which developers receive a onetime financial reward when the company accepts a given object in its reuse library. They receive royalties when other developers reuse their objects in other projects. This way, reuse has become part of the company's culture.

STEP 12

Conduct a postmortem

ASK YOUR TEAM: What went right? What went wrong? What should be done differently next time?

You may discover that your software development team needs to be organized differently. You may find, for instance, that it makes sense to split into two groups: object builders and system builders. The first creates individual objects. They are programmers at heart. They may not care how those objects are ultimately used, but they love a technical challenge.

System builders, on the other hand, thrive on understanding the business processes and assembling collaborations of objects that build systems to serve the business well.

You will no doubt also find that your comfort level with object-oriented technology has grown enormously. That, more than anything, should encourage reuse and help generate the productivity gains the technology is designed to promote. ■

Argila is a software engineering consultant in Pico Rivera, Calif. He offers consulting and training services in systems analysis and design, development methodologies, project management and computer-aided software engineering. He can be reached at (800) 347-6903 or on the Internet at carl@acm.org.