TRACING THE INVISIBLE HAND: CD ROM AND THE VALUE OF INFORMATION

We spent a wonderful two days at Microsoft’s recent CD ROM conference, but we had the disquieting feeling many people were barking up the wrong tree. How many megabytes? What standard? What access techniques? Such questions abounded. Few people -- with such notable exceptions as Bill Gates, Ted Nelson and James Burke -- said much about the global implications. At the other end of the spectrum, few people showed concern for the mundane business issues that will inevitably confront them: How should the stuff be priced? What kinds of support, distribution channels, marketing techniques are appropriate?

The global questions are tougher to ask, let alone answer. On the one hand they deal with the structure of information, and on the other with its value. Does information have any value until it’s defined and structured? What’s the value of information about the information? What’s the value of a stack of magazines without any index? It’s the value of the information minus the value of the time someone will take to find it, multiplied by the less-than-100-percent probability the information you want is in there.

What’s the value of an oil lease? That depends on the likelihood that oil will be found. And what’s the value of that information? It depends both on the cost of drilling to find out as well as on the price of the oil that may be found.

So how does information acquire its value? The future will require new standards defining what information is so that it can be transferred and valued. Long run, as physical production becomes ever more optimized and distribution, not production, becomes the primary economic task, assets will be defined by a bank or a law, rather than by physical possession or stakes in the ground.

NEXT MONTH: IBM ANNOUNCES...
EXCERPTS FROM THE PROGRAMMER'S BOOKSHELF™

...from "The Programmer's Travel Guide," on getting to the airport

The from-scratch, procedural programmer: [This section contains a long list of instructions tailored for a variety of different locations.] From Rickeys Hyatt, turn north on El Camino. If you have lots of time, you may prefer to use 280. Turn left at Page Mill Road. Otherwise, proceed north to University Avenue, bearing right at the Holiday Inn and driving through the underpass before reaching the train station. From the Stanford Park Hotel...

The procedural programmer with reusable code: Drive north on El Camino past Stanford until you reach the Stanford Park Hotel. Now follow the instructions on page 32.

The object-oriented programmer: Have your assistant arrange transportation for you. [If you're a team leader, your secretary will call the limo service. If you're one of the troops, the office manager will make a reservation on the Airport Connection van.]

The Trees programmer: Select the tree of all the routes to the airport. Examine all the routes, and delete each one that's longer than those previously examined. Follow the route that remains.

The AI hacker: Anytime you see a sign for the airport, follow it. (It helps if you're in a car.) If you're at the airport, park and get out.

The editor: Find a friend who's willing to drive you.

...from "The Programmer's Cookbook," on making soup

The from-scratch, procedural programmer: Get a heavy pot, and fill it with water. Turn on the heat. Meanwhile, chop some onions. Get another pot, add butter, and fry the onions...

The reusable-coder: Follow instructions for soup stock on page 49. Fold in sauteed onions, following instructions on page 129...

The object-oriented programmer: Find a can of Campbell’s soup. Follow the instructions on the can. [Note the instructions for thicker soup, which advise you to replace water with milk.]

The Trees programmer: Assemble all possible ingredients for soup. For each possible combination of ingredients, rate desirability according to your criteria (e.g. taste, cost, nutrition, preparation time). Return the sub-optimal items to the icebox, and cook the rest.

The AI hacker: If it tastes good, put it in the pot. If the result is liquid and hot and tastes good, it's soup. (If it's moist and warm and tastes good, it's stew; if it's dry and hot and tastes okay, it's hash.)

The editor: Call a restaurant for reservations.
OBJECT-ORIENTED DESIGN

There's a movement afoot. In the first wave of the personal computer, computers were personal but not personable -- collections of chips on boards, kits, and the odd mainframe or mini in the hands of a single user. The persons using them were few and experienced; the machines were too expensive and complex (if crude by current standards) to be widespread.

Along came the Apple II and the IBM PC. The hackers were quickly outnum-
bered by people who used computers not as the flexible machines they are, but as appliances -- spreadsheet appliances, word-processing appliances, list-management appliances.

Now we're on the edge of the third wave, which will enable more of those appliance users to become hackers. The key to that lies at least in part in object-oriented programming and the concepts that come with it. It's not that most users are going to become object-oriented coders; the guts of object-oriented programs are as tough and kludgy as any other kind to write. But graphically rich programs built with object-oriented techniques hold the promise of letting end-users manipulate and customize their software instead of merely adapting to something off the shelf. (And consider all the 1-2-3 coders who don't even think of themselves as such.)

An objective perspective

This article discusses object-oriented programming and profiles the players. Object-oriented programming is a technique, not a market. It provides benefits in design quality and time, and in code size; its major negative is slower execution time, a diminishing concern as hardware gets more powerful. Clearly, given market trends, object-oriented programming makes sense, for virtually all the same reasons as computer-aided software engineering, with which it shares some features.

Naturally, object-oriented programming's virtues are not universally recognized. For starters, non-users tend to confuse it with AI or with the use of Smalltalk, a notably slow if elegant language. The availability of object-oriented programming support in supersets of the C language such as Objective C and C++, and in Pascal-based implementa-
tions such as Actor and Apple's Object Pascal, should help change these attitudes and foster object-oriented programming's adoption.

Object-oriented programming won't solve the world's programming prob-
lems, but it's a significant addition to a growing array of programming techniques that are becoming vital to productivity as a larger and larger proportion of the country's workers acquire computers and need power over their software to become optimally productive. Most won't be users of object-oriented programming techniques per se, but of the resulting friendly, customizable software.

And in the meantime object-oriented programming already offers benefits both to developers and to users of the resulting programs (who are frequently one and the same): After an initial start-up period, it makes software easier to develop and enhance. Specifically, it makes "friendly" interfaces easier to implement. And it helps builder-users to create easy customization fa-
cilities, so that even relatively inexperienced users can customize their

Release 1.0 24 March 1987
software to some extent. Fostered by and fostering object-oriented programming is the popularity of graphics-intensive user interfaces, which are tough and tedious to program the old way (ask any Mac or Windows programmer!). The ability to marry UNIX and graphics is especially exciting.

Yet we still suspect (unhappily) that those who see users taking control are intelligent people who tend to project a little too much of themselves into the world around them. There's a distinction between the user as engineer and the user as end-user; so far object-oriented programming empowers mostly the latter. We wish it weren't so, but our optimism fades every time we see someone stare quizzically at a train or bus schedule and then hand it to us, asking for help. (Yes, it's happened several times.) The market for PC-as-appliance may always beat the market for pc-as-tool. Still, object-oriented programming will help us make more controllable appliances.

What is it?

What constitutes object-oriented programming? A number of related concepts: Objects and classes; inheritance (the ability of a class to pass on its properties to a subclass); information-hiding (also known as encapsulation, the notion that objects hold their own methods or procedures for responding to generic commands, or "messages," from the outside); and dynamic binding (the avoidance of "binding" or attaching procedures to objects until a program is actually executed, which allows changes in classes, objects or procedures without disruption). All these foster the reusability of code, so that software wheels need not be reinvented -- just improved upon!

Talking clearly about object-oriented programming is difficult; it is a loosely defined set of approaches to software design and programming, with different vocabularies and practices in currency. Not only do developers argue about whether to do it, they argue about what it is. And not only do they argue about what it is, but about what it should be.

Object-oriented programming in context

Object-oriented programming implements the general concept of seeing things and functions as objects. That concept is around everywhere -- in how we think, in how we sometimes represent things in hierarchies and diagrams. Like the gentleman who had spoken prose all his life, we do it naturally, whenever we think about things abstractly.

But its implementation in concrete software programming techniques is much rarer. For starters, execution time in Smalltalk, an early and still the most widespread object-oriented language, is notoriously slow, and Smalltalk once ran only on specialized Xerox hardware. Second, few people knew how to use Smalltalk or understood the concepts behind it. Indeed, traditional programmers pride themselves on their ability to translate people's object-oriented thinking into linear, procedural implementations. The lack of object-oriented design tools and languages (until recently) that can handle such abstract constructs is in part why programming is such a tough job.

Now, the growing use of object-oriented languages with pre-compilers that translate object-oriented designs into procedural source languages is making life easier. The languages allow us to think OOPs-ly, raising productivity for initial design and prototyping and subsequent implementation and main-
tenance, but the familiarity of the underlying language keeps the program-
mers comfortable and reduces retraining requirements.

OOP and CASE

Ironically, object-oriented programming is useful for designing CASE tools, since the images in a CASE diagram can conveniently be built as objects. But the traditional metaphor of CASE, where data "flows" from procedure to procedure, is orthogonal to that of object-oriented programming, where commands flow from object to object, and are carried out according to the kind of object that receives them. But the boxes and the links on the CASE screen must follow a variety of rules, all of which can easily be represented in the object-oriented metaphor. (Meanwhile, each of the boxes could represent an object-oriented program, but the fusion becomes awkward.)

The most offputting aspect of object-oriented design for traditional developers is the loss of control it implies. Consider the spectrum from batch programs to interactive ones to object-oriented programs. Batch programs are a sequence with a beginning and an end (perhaps with some branches and loops along the way). Reluctantly, programmers gave up some control to let end-users interact with programs. Although their code was still procedural, user input in part controlled the execution of modules of a program. Now, with object-oriented programming, the developers must cede even more control -- or, to put it more positively (if they could only see it!), they avoid having to handle the details. Object-oriented design lets the program, not the developer, decide the way in which things happen -- and it invites both unplanned messages and new objects (i.e., interference) from third parties. In the traditional schema, a "finished" application is finished -- at least until someone takes it down for maintenance and messes with the source code.

Software designers will have to learn to trust other people's work if they are to get much done in the future -- much as a builder uses other people's girders and window frames, a computer maker uses other people's ICs, or an editor uses other people's reporting. They must feel comfortable with indirect control: It's the difference between telling your children what to do step by step, and bringing them up right and hoping they'll behave. (In object-oriented terms, you give them the right procedures to follow.)

Objects in context

Objects comprise everything in a program -- both data structures and procedures (nouns and local definitions of verbs). Typically they exist -- are instantiated -- as a member of a class. Each object (instance of a class) has its own data, but it shares the data fields and methods (procedures for carrying out messages it may receive) of its class. For example, sales employee is a subclass of the class of employee. Sales employees have attributes inherited from the employee class, such as length of service and age, but they have a different method for determining compensation, which depends on sales and commissions, fields specific to the sales employee class. Each individual salesperson object has his own identity, as reflected by the particular data in the fields acquired from his class, as well as the acquired procedures, or behaviors, of his class.

A familiar visual example of the creation of an object is the "stationery" metaphor used in the Lisa and some Macintosh applications: A spreadsheet is peeled off a stack of spreadsheet paper, instantiating a new spreadsheet.
object and the associated data structures and commands. Similarly, the underlying operation of many drawing programs is that when a user selects an object from the palette, he sends a message "Create yourself." When he chooses a point on the screen and clicks, he says, "Draw yourself here." Further actions specify the size and other characteristics of the object.

Inheritance

While objects are instances of a class, subclasses are new classes that inherit or specialize their features (fields and methods) from their super-class in a hierarchical structure. Assuming you start out with good classes (whether your own or from a third-party class library), inheritance leads to better, faster code.

Once created, a class need not be reinvented, but can instead be reused, edited, particularized through inheritance and overriding; only the differences need be specified. Rather than edit source code, a builder can define new classes that inherit the structures and procedures of existing ones. And because of encapsulation, which keeps objects and procedures in discrete chunks, he needn't worry that these changes will affect the rest of the program. All this not only saves coding time, but helps ensure consistency and avoid bugs. Moreover, human nature dictates that reusable procedures will be written more carefully than code destined to be used only once, and end up not just bug-free but faster, cleaner and smaller than throwaway code.

(Some clarifications: Some object-oriented languages, such as Object Logo, don't make the distinction between classes and objects, and allow inheritance from other objects. (The copy function in most drawing programs is an example of this; you can create a specific instance of a circle and then create additional instances of it.) More commonly objects are considered instances of their parent class, from which they acquire their properties. Inheritance is also called specialization, as when the class of sales employees specializes from the class of all employees. And of course there is disagreement on the importance of these various concepts. For example, some people consider Ada an object-oriented language, while most say that its lack of inheritance and dynamic binding disqualifies it.)

(Further clarifications: Object-oriented programming and artificial intelligence have enough in common that they tend to get incorrectly confused. LISP as a language makes it easy to do either AI or object-oriented programming, and many people confuse LOOPS or Flavors, object-oriented extensions of LISP, with the underlying language. Meanwhile, object-oriented programming makes it easy to build the pretty interfaces that are frequently associated with AI. Both OOP and AI eschew procedurality. Yet neither is a subset of the other: You can have either separately, or both together.)

Encapsulation

The particulars of each object are hidden (encapsulated) so that the programmer does not need to deal with them. Each class contains its own methods or procedures for responding to "messages," or commands, which need not know what specific kind of object it's addressing. (I.e., a circle object and a rectangle object would respond differently to the same message: "Draw yourself." ) Procedures within objects are activated by receiving a message that says, in effect, "Do your thing." The procedure's "thing" depends on
its class, so the object sending the message is not aware of how the message will be carried out. (Messages do contain specific information about what is to be done, just not about how.)

**OOP and DBMS**

Let's compare a class hierarchy, which holds information about objects, and a relational schema, which holds information about data. The data in a relational table may be used to instantiate an object, and the table itself could be an object, with procedures such as open, display, etc. For example, Juan is an instance of the class of sales employee. The sales-employee class has internal procedures, such as an SQL call, to get the data describing him (and Alice, another sales employee). That procedure selects the appropriate information from one or several tables (although until recently most object-oriented programs simply dealt with a list of information in memory -- one reason traditional application developers may be suspicious of the approach). Thus Juan's data are not stored as an object; they are "instantiated" as necessary from the table (or other data structure) where they are stored. If a message arrives asking for Juan's salary (from the payroll object, for example), the sales employee class has a procedure to get Juan's sales data and commission rates, and determine how much Juan has earned.

How does it get those? Maybe they're stored by department, and the procedure has to look at Juan's department and get the data from that table. Or maybe they're kept by product line. Only the individual procedure knows...

Alice, on the other hand, is a special kind of sales employee -- a sales manager. Her class, a subclass of sales employee, overrides the salary procedure inherited from the sales-employee class, and pays her on the basis of her department's profitability. Sales manager could instead be a subclass of the managers class, but (in most implementations of object-oriented programming) not both. In either case, the subclass knows where to go to inherit its procedures.

Thus the hierarchy of classes is independent of whatever persistent data structures exist -- although its procedures must know how to manipulate them. When the data structure changes, the appropriate procedures are changed once, and then ripple through the system as needed by inheritance. (In less abstract terms, a class consists of data as does any other element of code, but that's another matter. In a coming issue, we will explore the related world of object-oriented data base management systems.)

**Binding -- early and late**

Encapsulation means that the programmer doesn't need to know the exact procedures to be used in response to any given message to a class. (Although he may in fact have written those procedures, he could instead have taken them from a class library.) Late, or dynamic, binding means that the program doesn't need to know either -- that is, the actual procedures to be used in response to messages are not determined until runtime. Even at that point, it's not the program that "knows," but just the instantiated object and its procedures. That leaves maximum flexibility in the program, although it does mean extra activity (i.e. slightly slower performance) at runtime. It also means that anytime a class or its procedures are changed, the impact remains strictly local; the rest of the program doesn't need to know, and can carry on merrily as before.
Several vendors use optimizing compilers/linkers that examine the program and determine which data types and procedures are never overridden or specialized and can therefore be bound early; others leave this option to the programmer. In concrete terms, early binding means that a general message, such as "Determine salary," is bound to the specific procedure needed to carry it out (given that there's only one way to do so).

Such early binding speeds up execution, since the procedure doesn't need to be bound each time it's used. But it also means that any subsequent changes to the program -- such as adding a new kind of employee with a new pay scale -- require the early binding to be undone and the program to be recompiled. That's not a big problem for a developer, but it makes it difficult to further modify, upgrade or extend the program -- a capability which is one of the major advantages of object-oriented programming. (If you know exactly what you want, of course, you can do it right the first time and never change it. Hah!) Consider the analogy between data independence, independence of an application from the data structure it addresses, as offered by a data base management system; and encapsulation, independence of modules within a program from each other.

Graphics

In addition to facilitating the development of large, complex programs, object-oriented programming is well suited for building graphics-based, interactive programs. It's bad enough having to determine the functionality of the program (and most object-oriented programming languages allow for the inclusion of blocks of C, Pascal, or other languages), but try simultaneously to handle menus, dialogue boxes, prompts and all the other pretty arcana users are learning to expect. How much easier to say, "Window, draw thyself!" and have the window not only draw itself but know how to position itself on the screen in relation to other window objects, how to size itself to fit, how to scroll, how to be active or inactive, etc. Then the developer need worry only about what goes inside the window. Who wants to handle traffic when the real job is ensuring that everything ends up where it belongs, whatever route it takes?

Execution time

Committed object-oriented programmers charge that claims of execution-time penalties for object-oriented programs are spurious, arguing that any slowness is simply the appropriate cost of better coding techniques and cleaner code, rather than a deficiency in implementation. Apple's Larry Tesler, for one, says, "We find that there's no noticeable difference in speed at runtime, especially if you pre-bind," he says. "It varies either way by a couple of percent, no more." We'd rather not get in the middle of this argument, but it's clear that performance is becoming less and less of an issue with increasingly powerful hardware -- and ease of programming and ease of use are becoming more important.

Potential value and realized value -- class libraries and tools

Object-oriented programming languages by themselves can offer benefits in design and development time. In addition, commercial object-oriented programming environments also offer ancillary programming tools (such as PPI's Vici) and extensive libraries of classes -- in effect, program modules that
save the user the trouble of constructing and debugging his own, yet allow
him great freedom to customize or add to them (inheriting and overriding
various procedures and data structures). Shared throughout a design team or
a company, they can raise productivity and help enforce consistency both in
programming practices and in the ultimate look and feel of the resulting
programs. Shared or sold throughout a design community, they can provide a
broad consistency among programs so that users can benefit from reduced
training and confusion, and so that vendors compete on grounds of function-
ality, not of interface. Typical class libraries include program interfaces
and graphical objects -- windows, menus, cursors -- as well as data struc-
tures -- words or paragraphs, cells and ranges in a spreadsheet, etc.

Sinning on weekdays

There's disagreement not just about the benefits of object-oriented design
and programming, but about when they kick in. In fact, those disagreements
represent people's varying needs more than they represent fundamental dif-
fferences of opinion. If a program is likely to be used frequently by many
people, maximum speed and stability are important. If it's likely to be
changed frequently in comparison to the frequency and broadness of use, then
the flexibility offered by object-oriented programming is worth paying for
in slight runtime penalties. We can draw the curve, but personal preju-
does determine the scale.

Thus, one is more likely to use object-oriented programming for an in-house
development effort, which will be subject to continual updating. Yet ob-
ject-oriented programming doesn't make much sense for small programs without
an appropriate class library handy; if there is one around, then object-
oriented programming makes everything much quicker. Finally, it makes no
sense to convert to object-oriented programming in the middle of a project
or, unfortunately, to use it to fix or maintain a program that was once con-
sidered "finished."

Tool vs. environment

Of course, it's not a choice that needs to be made once and for all unless
you operate within a seamless object-oriented environment: You can use a
variety of approaches within a single project. Many object-oriented pro-
grammers like the ability to resort to a procedural language (including
assembler) to write the primitives -- i.e. the procedures that actually
carry out the actions, especially when speed or complex calculations are at
issue. These are the people who see object-oriented programming as a tool,
with the benefits of reusability of code, resolution of complexity, etc. On
the other side are those who favor it as an environment, within which the
user gains control over malleable software.

The yes-buts

While much of the fear of loss of control may sound like the self-serving
whines of insecure programmers, object-oriented programming is not a pan-
cea. Aside from not being suited for computation-intensive tasks, it does
hide things so that bugs are hard to decipher and fix. Even allowing for
vendors' claims that their class libraries are well-used and debugged, soft-
ware is also susceptible to bit errors from hardware malfunctions, which are
much harder to find when a user can't quite trace what's going on anyway.

Release 1.0

24 March 1987
That's why the development environment -- the browsers and debuggers -- matter so much.

Another problem is cited by Bat Davidge, a consultant who's tried a number of different object-oriented extensions to C: "When you complain, they tell you to go back and read the original papers." But that academic attitude is disappearing as object-oriented programming becomes the province of commercial vendors -- and commercial users.

THE LANGUAGES

The Smalltalk language was developed at Xerox PARC by Alan Kay and a host of others in the Seventies. It is still the most widely used object-oriented language, with a user base we estimate at about 15,000 (including the Digitalk implementation). Coming up with good numbers is impossible, since many people have dabbled with it, especially in academics. Clearly the number of users is larger than the number of copies sold, while the number of serious users (as opposed to experimenters) is far less.

The specific Smalltalk-80™ product owned by Xerox is an environment as well as a language, and is designed to be used that way. It has an extensive library of 250 classes and about 4000 procedures. Except for Digitalk, which has developed its own clone, most Smalltalk vendors license the product from Xerox.

By contrast, the object-oriented C implementations are going after a much broader but less committed audience. Their success will lead the success of object-oriented programming in general. These C superset languages translate into standard C code, and do not need any runtime version. As yet, they lack much in the way of graphics support. In essence, the programs they generate are built to run as applications rather than enrichments to the Smalltalk environment. For example, the C-based object-oriented languages currently offer no garbage collection, working on the assumption that an application will be started, run, and closed, at which time the garbage will be collected. The "environments," by contrast, may run and interact with users all day, requiring the collection of leftover objects that are no longer needed.

C++, an object-oriented superset of the C language, was developed by Bjarne Stroustrup at AT&T (which has used it to write a million lines of code). In the object-oriented C world, C++ plays the role of Smalltalk, in that it resold under license in various implementations by third parties, while Objective C is an independent product conforming to the C standard. The new ANSI standard for C is widely expected to include a number of object-oriented extensions, which will be reflected in new versions of C++ and Objective C. But whatever ANSI and AT&T do with C itself, vendors of the languages will add increasing value with development tools and class libraries.

Of course, one doesn't need an object-oriented language to build object-oriented programs (just as one does not need LISP to do AI), but it helps!
### SOME OBJECT-ORIENTED PROGRAMMING LANGUAGES

<table>
<thead>
<tr>
<th>Language</th>
<th>Price</th>
<th>Target</th>
<th>First ship</th>
<th>Base (est.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>$995</td>
<td>PC/Windows</td>
<td>Feb 87</td>
<td>400 (incl. betas)</td>
</tr>
<tr>
<td>Whitewater</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Advantage C++</strong></td>
<td>$495/$695</td>
<td>PC (DOS/Xenix)</td>
<td>Oct 86</td>
<td>1000</td>
</tr>
<tr>
<td>Lifeboat</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Designer C++</strong></td>
<td>$950</td>
<td>Sun</td>
<td>Nov 86</td>
<td>200</td>
</tr>
<tr>
<td>Oasys</td>
<td>1200-4000</td>
<td>MicroVAX/VAX</td>
<td>Dec 86</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>1200-4000</td>
<td>AT&amp;T 3B</td>
<td>Dec 86</td>
<td>100</td>
</tr>
<tr>
<td><strong>Object Pascal</strong></td>
<td>$125</td>
<td>Macintosh</td>
<td>Jun 85</td>
<td>1000</td>
</tr>
<tr>
<td>Apple</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Objective C</strong></td>
<td>$2500</td>
<td>UNIX, VMS</td>
<td>late 84</td>
<td>/3500</td>
</tr>
<tr>
<td>PPI</td>
<td>500</td>
<td>PC-DOS</td>
<td>Jun 85</td>
<td>\</td>
</tr>
<tr>
<td><strong>Smalltalk-AT</strong></td>
<td>$995</td>
<td>PC-AT</td>
<td>Mar 86</td>
<td>200</td>
</tr>
<tr>
<td>Softsmarts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Smalltalk 0.X</strong></td>
<td>$50*</td>
<td>Macintosh</td>
<td>Jun 85</td>
<td>1000</td>
</tr>
<tr>
<td>Apple</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Smalltalk/V</strong></td>
<td>$99</td>
<td>PC</td>
<td>Aug 86</td>
<td>7000</td>
</tr>
<tr>
<td>Digitalk</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Smalltalk/80</strong></td>
<td>$5000</td>
<td>Sun other 68000s</td>
<td>Nov 86</td>
<td>&lt;100</td>
</tr>
<tr>
<td>ParcPlace</td>
<td></td>
<td></td>
<td>soon</td>
<td></td>
</tr>
<tr>
<td><strong>Smalltalk-80</strong></td>
<td>(bundled)</td>
<td>4400 series</td>
<td>Aug 84</td>
<td>2000**</td>
</tr>
<tr>
<td>Tektronix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Smalltalk-80</strong></td>
<td>$5000</td>
<td>D-machines</td>
<td>84</td>
<td>200</td>
</tr>
<tr>
<td>Xerox</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*not supported

**not necessarily in use, since it comes bundled with the hardware at a total cost between $15,000 and $24,000

Most prices shown are heavily discounted for volume purchases and purchases by academic institutions.

The figures shown for estimated sales are highly suspect, and do not include quantities given to educational institutions.

Because comparisons of class libraries are odious (one man's class is another man's reject), we have left class libraries out of this table and discuss them only in the accompanying text. Yet they form an important part of the value of any object-oriented programming toolset.

*Release 1.0*  
24 March 1987
"Object-oriented programming is basically all about relinquishing control, passing the buck. The trick is to design trustworthy objects so you feel confident delegating." --- Larry Tesler

APPLE

Apple has had a long association with object-oriented programming. In 1980 it hired Larry Tesler directly from the Smalltalk group at PARC, and promoted him to vice president of advanced technology late last year. In 1982, Apple built a Pascal-like language called Clascal for the Lisa. Working in conjunction with Pascal author Niklaus Wirth, the company subsequently developed Object Pascal for the Macintosh in 1985.

Apple's current object-oriented development environment for the Mac costs $350 (ex hardware): $125 for Object Pascal, $125 for the Macintosh Programmer's Workbench, and $100 for MacApp ("The Expandable Macintosh Application"), a sample application with a library of graphics and other classes that take much of the drudgery out of building Macintosh applications. (In addition, Apple charges $100 per year for a license allowing unlimited distribution of MacApp applications, but restricting them to use on Apple Mac machines, a less onerous limitation with the pending arrival of the Mac II.) Object Pascal/MacApp users can also benefit from the existence of the MacApp Developers Association, headed by former architect Carl Nelson, a man who clearly likes to think in pictures. The group has its own newsletter, and Object Librarian Kurt Schmucker (yes, the same one who works for PPI) solicits member contributions to a shared collection of reusable classes.

Object Pascal has been around since 1985 and probably approaches 1000 active users; MacApp, which most of them will use, went into beta test last summer and came out formally early this year. By now, of course, most Macintosh developers have built their own libraries of Mac interface routines (for the non-object-oriented) or objects. And forget the runtime issue: Object Pascal does not compile as fast as Lightspeed C. Programmers ignore the time they save with reusable classes; the time they spend working doesn't faze them as much as the time they spend watching the computer work.

But we expect many of them will adopt Object Pascal and the MacApp library over time, as will many new Mac developers. As one of them said at the first MacApp Developers meeting, anyone who has tried it the old way understands the need to do it the new way!

Written in Object Pascal, the MacApp classes can also be used with C and assembly language, although the user has to do a little more work to define and use the classes to full advantage (until Apple releases a long-awaited object-oriented version of C). Apple takes a strictly non-sectarian view of object-oriented programming. Object Pascal, for example, includes an optimizing compiler/linker that forces early binding by examining the code to see which classes/procedures can be early-bound, and then doing so.

Apple also offers but does not support an Apple-unique version of Smalltalk descended from an early version distributed by Xerox under favorable terms.
Digitalk, based in Los Angeles, was started in 1983 by Jim Anderson, a pioneer on compiler optimization theory and a consultant to a variety of computer makers, to spread the concepts of object-oriented programming to the masses. Unlike all the other Smalltalk vendors listed here, Digitalk pays no royalties to Xerox/ParcPlace (see below). Digitalk's competitors tend to sniff at Digitalk's Smalltalk/V. They liken it to Turbo Pascal or Turbo Prolog, and profess worry that it will define the language "by dint of sheer numbers." And they make the distinction between it and the Smalltalk-80™ environment in which Xerox asserts a proprietary interest. Says Adele Goldberg of Xerox/ParcPlace: "Smalltalk/V doesn't have the look and feel of Smalltalk-80. It's in the interests of the industry to have the language available at a variety of levels."

Digitalk's first product (January 1985) was Methods, a version of Smalltalk without graphics that sold for $250. Last summer it released Smalltalk/V -- with graphics, and costing only $99. "We carefully honed it to fit," says Anderson. "The competition carries the baggage of 10 years of research."

As it happens, this honing gives Smalltalk/V superior performance to Smalltalk-80 on a PC AT -- albeit of a more limited set of functions. Although users are building production applications with Smalltalk/V, its class library of about 100 objects with 1800 procedures (about half the size of the "real" Smalltalk's) also limits its power. If the procedures are the ones you need, fine; if not, you lose much of the benefit of reusable code. Smalltalk's development environment -- browsers and debuggers -- is also limited; its debugger stops at errors, but doesn't let a user step through the operation of a program.

Unlike Methods, which was limited to 640K, Smalltalk/V supports virtual memory and can swap objects out to disk or above the 640K barrier with a RAM disk. Digitalk has also added support for multi-processing, useful for discrete-event simulations, and a Prolog compiler. With sponsorship from Olivetti, Digitalk is working on an 80286-specific version that will run in protected mode with access to up to 16 megabytes of RAM.

LIFEBOAT/HUDSON TECHNOLOGIES

Lifeboat, discussed in our September 24 issue, is the exclusive distributor of Advantage C++, a version of AT&T's C++ for the PC and compatibles developed by Glockenspiel of Dublin, Ireland. Advantage C++ has the advantage of the perceived AT&T seal of approval, if not of AT&T's support. The Advantage C++ pre-compiler/translator translates its user's code into either Lattice or Microsoft C, and takes up about 400K.

At the moment, Advantage C++ offers little in the way of a class library, but in May Lifeboat will offer one to support Windows development. Lifeboat is also beta-testing PforCe++, an extensive library of about 400 graphics, data structure and other classes from Phoenix Technologies, which already sells PforCe (a library of C routines) through Lifeboat. Finally, the company will distribute a free library of Smalltalk-like classes developed at the University of North Carolina.

Release 1.0 24 March 1987
Advantage C++ (like other C++es) offers as a feature what PPI considers a liability -- automatic early binding unless the builder-user declares otherwise. Lifeboat also plans to ship a debugger for C++ later this year.

OASYS

To generalize about all the companies mentioned here, Oasys probably serves the most developer-oriented customer base -- that is, professional C programmers enticed by the lure of productivity gains from object-oriented programming, as opposed to end-users who want to build or customize their own environments (the engineers and PC users), and graphics whizzes who want some way to do whizzy graphics easily (the Mac, Actor and Smalltalk groups). Based in Cambridge, MA, Oasys sells a variety of languages and other software engineering tools and utilities, taking on whatever role is appropriate: developer, remarketer, publisher, distributor, or consultant. About two years ago, "working on our gut" rather than any expressed customer demand, says director of marketing Gregory Kee, the company licensed a number of C++ implementations from Glockenspiel. (These include rights to the PC version for the U.S., which it subsequently reassigned to Lifeboat, which is more familiar with the low end of the market.)

Now, says Kee, Oasys's boothmate at recent trade shows has been miffed because 80 percent of the traffic is due to Designer C++. Those customers tend to be existing C users (typical of Oasys's current base) who are eager to try it out. They're especially excited at the prospect of designing graphical user interfaces with a minimum of effort -- something that won't really happen until there's a good library of interface classes. Oasys currently offers no libraries, although it refers customers to a number of public-domain sources and to fellow users. By May it hopes to ship a debugger, with a library to follow shortly.

PRODUCTIVITY PRODUCTS INTERNATIONAL

Objective C, from Productivity Products International (discussed in our 24 September issue along with Lifeboat), was developed in 1983 by Brad Cox and Tom Love, both from ITT. They founded their company around Objective C and have devoted years and millions of dollars to building up the surrounding programming environment, Vici, and a substantial class library. Kurt Schmucker, an object-oriented programming expert who gives seminars for PPI (and author of "Object-Oriented Programming for the Macintosh" from Hayden), calls the library's classes Software-ICs, making the valid point that software engineers can operate just like circuit designers, selecting packages of functionality that act together in known ways to perform tasks of considerable complexity -- and without the designer knowing much about the internals of each integrated circuit. (In this context, perhaps, specific software routines and procedures are ASICs, or application-specific ICs.)

Like C++, Objective C is a pre-compiler; i.e., it compiles Objective C code into C source code that can be compiled by most regular C compilers. While Lifeboat considers the C++ early-binding a feature, Productivity Products International, which aims to sell an environment rather than just a language, considers its avoidance of same a feature. PPI sells to people looking for a total development environment, whereas Lifeboat expects its users to use other C tools such as Microsoft's CodeView.
Objective C includes a class library of about 30 classes with 300 methods, and is working on an interface library with 50 graphics and other classes and more than 600 methods. For $2500 (or $500 in a PC version due out soon), PPI adds Vici, an Objective C interpreter targeted at developers who like to interact with their code. While both Advantage C++ and Objective C are supersets of C (although each vendor casts aspersions on the purity of its competitor), Objective C's extensions give it some of the flavor of Smalltalk, while C++ adheres more strictly to C conventions of thought and structure.

XEROX/PARCPLACE

ParcPlace Systems is an about-to-happen spinoff of Xerox PARC, run by Smalltalk team manager Adele Goldberg. (It was Goldberg who reluctantly gave Steve Jobs one of the demos that strongly influenced Apple's Lisa and Mac.)

PARC, of course, is where Alan Kay developed Smalltalk in the Seventies, and it holds a special place in the hearts of object-oriented programmers everywhere. Recognizing that PARC's role is research and not development or sales (Palo Alto Research Center), Xerox officials are currently negotiating the terms of the spinoff and the amount of Xerox's equity in ParcPlace. The unit has ported Smalltalk, originally implemented on the Xerox D-machines (Dorado, Dolphin, and Danditiger) to the Sun, Apollo and a variety of other 68000-based machines, including the Macintosh. It also licenses Softsmarts to make and sell an 80286 version.

The current version, the Smalltalk-80\textsuperscript{tm} environment, offers the advantages of an extensive graphics-rich class library, powerful browsers, debuggers and other programming tools, and a cross-system development environment, so that you can develop on one system and run on a multitude of others. For people who believe in object-oriented programming (as opposed to those who use it), Smalltalk-80 is the ideal environment.

Goldberg plans to "take the basic Smalltalk system and apply it to programming in other languages and integrate it with databases and operating systems." She will also unbundle the system, so that developers can use only the class libraries appropriate to their needs, and users can run the resulting applications in a runtime environment. The company currently has 18 people and should grow to 30 by year-end.

Xerox will continues to sell Smalltalk-80 for its own D-machines, frequently as part of customized systems, as well as related applications such as NoteCards, The Analyst, and Humble.

SOFTSMARTS

Softsmarts sells its authorized version of Smalltalk-80 for $1000, in part reflecting the current royalty of $200 it pays Xerox for each copy (which may change with the spinoff of ParcPlace). Although positioned differently, Softsmarts' Smalltalk-AT competes with Digitalk's $99 Smalltalk/V. Smalltalk-AT runs on the PC AT in protected mode, and thus is able to take up 2 megabytes of memory without swapping.

Release 1.0

24 March 1987
"The typical ethic in programming is, 'I'm a stud; I wrote all this code,'" says Softsmarts vp of R&D Steve Burbeck. "But with Smalltalk, you're better off; you can browse through and rewrite stuff that's already there." Code is reusable only if you can find it; that's the strength of Smalltalk-80's browsers, which are inherited, so to speak, by the Softsmarts port.

The company's founders worked together at the Linus Pauling Institute of Science and Medicine in Palo Alto, where they used a Smalltalk environment to do their work. When the AT arrived, they saw and took an opportunity to match their favorite environment with a mass-market vehicle.

TEKTRONIX

Tektronix was one of the first outside users of Xerox's Smalltalk, along with Apple and with DEC and Hewlett-Packard, who were less enthusiastic. In 1984, the company ported Smalltalk to its own 4404, a 68000-based "AI" workstation, originally for internal use. It showed the system as a product at the Austin AAAI meeting in the summer of 1984. The response persuaded the company to follow up with versions for the 68020-based 4405 and 4406 in 1985, says Jeff McKenna, technical applications manager at Tek's AI machines group and a one-time employee of Sorcim. "I get real jazzed about object-oriented programming," he says. "PCs with 1-2-3 -- those aren't personal computers; they're clones. The vendors all lock up their software because they don't want it stolen. Smalltalk is the first software malleable enough to let users build their own."

Tek's version is fairly close to the one on Sun sold by ParcPlace, says McKenna, but Tek's graphics look faster. Although the system was originally positioned as an AI machine, says McKenna, it's now being used extensively in software engineering and prototyping, factory planning, circuit and board layout, scheduling, and a variety of other applications that are really systems design of one kind or another.

"For us internally, the key use is in interface design," says McKenna. "It's getting so complicated that you can't specify it anymore; you've just got to show it, prototype it. It's not simulation; you actually program it. And suddenly they're discovering that the prototype runs fast enough...it is the application!"

THE WHITewater GROUP

If Microsoft Windows is to be successful, it must be made easier to develop for -- but Microsoft is currently too busy working on DOSeS and applications. Filling the breach is The Whitewater Group of Evanston, IL. Founded by Mark Achler, a former marketing manager for the Apple IIC, and Chuck Duff, who wrote Typing Tutor III for Kriya Software, The Whitewater Group is Chicagoland's response to Silicon Valley. It is funded by the Illinois Business Innovation Fund and the Evanston Business Investment Corporation, and housed in Northwestern University's business incubator.

Duff is the author of Neon, a Forth-based object-oriented language for the Macintosh. But, he notes, even though it is frequently used for real-time systems and executes efficiently, "Forth just isn't acceptable to the main-
Selling object-oriented programming is tough enough! Accordingly the company's first product, Actor, uses a Pascal-like syntax. To alleviate the problems of slow execution, it compromises by forcing early binding in easy-to-determine cases and also helps the user to do so manually. (The manual binding especially is a slightly dangerous approach, since it isn't automatic, and therefore must be redone by hand, carefully, whenever a program is changed.)

The Whitewater Group committed to Windows a year ago, and it is now both fostering and exploiting Windows' entry into the marketplace. Developing for Windows is tough; Actor makes it a little easier. The system makes it relatively easy to use the Windows interface, with simple manipulation of windows, dialogue boxes, scrolling, file-handling and other routines, leaving the builder-user free to focus on his program's functions.

The presence of Windows is also Actor's biggest problem: The two of them together (300K for Actor plus 100K for Windows) take up so much space that there's little room left for the programs Actor is building (the runtime version is about 50K smaller). On the other hand, the object-oriented approach produces less code, since so much of it is reused at runtime, and Actor's garbage collection facilities also dynamically keep memory use as low as possible. Still, both Windows and Actor will come into their own with the arrival of DOS for the 386.

Meanwhile, Reuters is working with Whitewater to create a version of Actor for the Sun; separately, another part of Reuters has developed a Windows application that many observers consider slow. Whitewater itself is working on an application framework -- a sample application along the lines of MacApp -- that would be virtually environment-independent. By using Whitewater's classes, the developer could build a single application with easy access to procedures for a variety of environments including Windows on the PC and NeWS and X-Window on the Sun, as well as links to commercial data bases and other data structures.

MICROSOFT

Microsoft isn't listed as a vendor on page 11, but it gets space here as an enthusiastic user of object-oriented programming, and a likely seller in the of object-oriented tools in the future. Says chairman Bill Gates: "This is not a product announcement, but we believe in that stuff. We are going to adopt some of those C++-like extensions. There's no magic. You can do it with today's programming languages, but it's hard to do. If you're doing good coding, you're already doing it. Back when we wrote BASIC for those small machines, with very tight code and lots of code-sharing, we were using this stuff."

The advent of Windows, and the expansion of its products in general, provide a perfect opening for object-oriented programming at Microsoft. Graphics-rich environments are incredibly complex to program without some ability to make the graphics objects take care of themselves. Indeed, Microsoft has built its own object-oriented extensions to the C language. For Microsoft Word, as one example, we saw a printout of 2004 methods written in C and organized by their classes according to object-oriented conventions.
Microsoft chief architect Charles Simonyi has invited Apple's Larry Tesler up to give his troops a talk on object-oriented programming's virtues. Says Simonyi: "C++ was love at first sight. We build a very nice structure for all the objects floating around in Word and Excel. Object-oriented programming is a guide, a source of inspiration. We observe the concept in our coding practice, but less in the specific language. We no longer have to convince anyone here of its goodness, but we still have to satisfy them with the right tools to express it. So far all we have is pidgin C++.

The good news is that the paradigm is so powerful that it can be justified even across a single application, let alone throughout a development group, says director of development Jeff Harbers.

Object-oriented programming began with Simula in 1967, and was furthered by Alan Kay when he developed Smalltalk in the early Seventies at PARC. Other important players include Carl Hewitt at MIT, who developed the notion of actors. For history, we refer you to the August 1981 issue of Byte magazine on Smalltalk, as well as the five-year anniversary (August 1986) issue on object-oriented programming in general. "Smalltalk-80: Bits of history, words of advice" (Addison-Wesley) provides an eclectic anthology of readings compiled by Glenn Krasner of PARC. (This is also known as "the green book," in line with the culture's visual imagery.)

As for the future, the first annual OOPSLA (for Object-Oriented Programming Systems, Languages and Applications) attracted 1000 people instead of the 250 expected last fall. Next fall's will likely be larger.

Lotus is actively using an object-oriented language in the development of one of its next products, and it has an extra person on the development team whose job is to make sure the group's class library will be suitable for dissemination elsewhere within Lotus. When the project is over, he will work with some other development groups within Lotus to help them adopt object-oriented programming techniques. Index Technology, a leading computer-aided software engineering vendor, considers object-oriented programming a promising market, "and we are investigating how to incorporate it in our future products," says vp of Sales and Marketing Chris Grejtak. NeXT has advertised in the San Jose Mercury for technical writers who understand object-oriented programming...

Release 1.0

24 March 1987
SOFTWARE GLASNOST: OPEN APPLICATIONS

Just as machines with slots are "open" to third parties if not to cloners, so are applications with object-oriented programming hooks open to customers and third parties who might want to add in new procedures and data types (objects). Offering a macro language is just a start (cf. the original 1-2-3). A macro language allows a user to execute routines that already exist in the underlying program, whereas an object-oriented program allows a user not just to use but also to alter those routines and write new source code to suit his purposes more closely. The middle ground, one that provides considerable support for application extension, is the provision of programming interfaces and associated tools.

Lotus has further opened 1-2-3 just this way with its Developer Tools. The builder/user gets a set of access methods and entry points, but not the actual source code to fool around with. This protects the original application and makes sure the extensions will work with as long as both the application developer and the extender stick to the interface rules -- an encapsulation of sorts in object-oriented terms. "It's like dealing with a giant object that you can send messages to but can't inherit from," says Lotus systems architect Adam Hertz.

Could a vendor sell more applications by making them open, even if it lost some utilities business, just as a hardware vendor loses some add-in memory sales but gains overall? Several application vendors think so.

Lotus Galaxy

Lotus recently showed off its follow-on to Jazz, Galaxy, to a number of press and other critics. Those on the West Coast loved it: It makes full use of the Mac (especially the Mac II), with such features as polar charts and other wonderful graphics, it has macros, and it's fast. On the East Coast, Lotus was berated for Galaxy's lack of full document-processing capabilities (indexing, spell-checker, etc.), which some took to mean that high-end management users likely won't adopt it, so it can't trickle down to low-end users either.

From our perspective, the most interesting thing about the product is its programmability. Systems architect Adam Hertz notes that the Jazz group's original expectations of naive users proved to be way off-base: The lack of macros was the major customer complaint. So the Galaxy team took a hint from the demand for the 1-2-3 developer tools and the success of Framework's FRED language. Galaxy includes not just a macro language, but substantial tools for linking to Galaxy routines, linking in new user source code, and creating custom objects such as menus or dialogue boxes. These tools don't go quite as far as the 1-2-3 Developer Tools, which allow access to 1-2-3's internal data structures, but they make it as easy as possible to extend and customize the product's functionality, short of supplying source code -- or using object-oriented programming.

Do users care more about full-featured document-processing (which happens to be a strength of Framework) or about programmability? Presumably Galaxy, promised for this summer, will find its natural market: Smaller than it could have been with both advantages, but considerably larger than Jazz.

Release 1.0 24 March 1987
Forefront/FrameWork/Fred

Forefront's Framework is perhaps the most widely owned semi-object-oriented environment. Forefront's parent Ashton-Tate has shipped roughly 200,000 copies of Framework, which includes the FRED language (for FRAME EDitor). FRED was developed by Robert Carr in 1983 to build the industry's most elegant integrated program, precisely because of the object-oriented way in which it integrates its parts. He regrets that he never got around to using FRED to reimplement Framework, and perfecting FRED as the fully object-oriented language with inheritable classes that he would have liked it to be.

Yet object-oriented metaphors pervade the program. What is an outliner but a way of treating text as hierarchies of text objects? (Living Videotext's MORE, which transforms outlined text into trees and back again, illustrates this point clearly.) And the notion of hierarchies extends beyond just the text. Each frame in Framework is a class; subclasses of the frame include spreadsheet, text/wp, file manager and graphics frames, which can send each other messages in classic object-oriented style.

Printing a Framework document means to command the topmost frame to print itself; that message is sent on to each object (frame) comprising the document, and there you have it: a hybrid document that includes text, graphics, tables, spreadsheets, whatever.

Framework has the advantage of an extremely rich library of routines in the guise of a set of integrated applications and modules that can be run as is, or modified and extended by a user who takes the trouble to learn FRED.

Polygen Centrum

Polygen's Centrum is another, less-known example of the new openness in software. Exemplifying the sensible trend towards shoulder-based development, Polygen is a version of Applix's Alis targeted towards scientists, and marketed mostly on VAXen and Suns. We've all heard by now that scientists have word processing, budgeting and record-keeping needs just like the rest of us... but they're notoriously reluctant to put aside UNIX, FORTRAN, Edlin, etc. in favor of a normal word-processor or filing program. That reminds us of the 1-2-3 fanatics who insist on using it for word-processing, file management, scheduling.

So why not give these folks what they want? Polygen is embracing this concept from the start. Chairman Jeff Wales, a former electronics engineer, wanted to find the chemical scientist's equivalent of the analyst's spreadsheet or the engineer's circuit diagram -- a medium both for development of ideas and for communication of the results. The answer: The scientific research paper. Somewhat of a mixed metaphor because scientists do so many different things, Polygen's Centrum is a document-processing environment that contains embedded applications for calculations, data management, graphing, molecular design and other tasks -- and that allows users or third parties to add their own. It solves two problems: It lets the scientists operate in a seamless environment rather than switch tasks or, worse, machines, and it supports them where they want support while leaving them free to build job-specific programs as they see fit.
"We use the words object-oriented programming around here a lot," says Polygen chief architect Ron Kalin, "and we follow many of the disciplines. But basically we mean it as a synonym for data-abstraction" -- without the class hierarchies and inheritance that characterize strict object-oriented programming. Instead, the system has a table of object types, and defines structures and procedures for each object. (The ability to do this is one of the reasons Polygen chose Applix's Alis.) The company may ultimately switch to some C-based object-oriented language, says Kalin, noting that since Centrum is written in C the move would not be tremendously disruptive.

RELEASE 0.5: MANAGING PROJECTS, HIGH AND LOW

Trees, from Avyx

When we were first learning about AI, tree editing ranked up there along with truth maintenance, garbage collection and multiple worlds on our list of favorite terms. So we were delighted to encounter Avyx's TREES-pls, with words such as branch, graft, prune, and high-water marks (maximum storage requirements). It also includes a library of TREES utilities for scheduling and resource management called FOREST-pls.

TREES, of course, is a tree- or hierarchy-oriented language. It's far from being an end-user tool and not likely to be a huge commercial success, but it will probably find a devoted following of users who find it ideal for modeling and planning such tasks as scheduling, allocation, layout, staffing, route planning, warehouse stocking, fleet management, and other logistical challenges. TREES and FOREST each sell for $995; the core-scheduling module, a scheduler with no interfaces or reporting facilities, sells for $1995. The PC-based language/development environment originated as a language for operations management used by partners John Willoughby and Jo Anne Gardner in work for NASA planning the shuttle project.

While LISP manipulates lists and Prolog executable declarations, TREES manipulates forked lists, or hierarchies -- the sort of bi- and multi-furcated objects found in outlines, organization charts...and trees. The branches of a tree can represent constrained resources converging dynamically on the completion of a project, rather than the usual static point-expansion where a tree consists of a breakdown of tasks and subtasks. Yes, you could also get there by backward-chaining, but Trees offers the programmer more control and a more natural way of doing things, says Willoughby.

TREES lets you manipulate the trees and their nodes (or branches), grafting and pruning as you go. While project management concentrates on schedules and predecessor-successor relationships, Trees focuses more on the management of activities and resources, even when there's no specific project to be completed (software maintenance, for example, which never ends). The resources in question are too scarce to lie idle for long, and their use must be optimized, frequently across projects.

Resources may have complicated interactions: For example, the third shift may not produce as much as the first shift at the same plant, although fixed costs per unit will be lower than for single-shift operation. Different
combinations and permutations of nodes generate different results. TREES- 
pl is designed to process combinations, permutations, rankings -- that is, 
structured list or hierarchies.

Avyx president John Willoughby hopes eventually to build the world's best 
activity modeling tool, but rather than keep his underlying technology pro-
prietary he hopes to make it a standard for this type of task. His larger 
system will then incorporate users' models, in the best of open style.

InstaPlan, from InstaPlan

Just as NewViews (Release 1.0, 2 December) gives one a new perspective on 
accounting, so does InstaPlan give you a new perspective on project manage-
ment. Both are the "personal tool" version of applications that generally 
have more of a factory feel to them. InstaPlan offers a variety of views of 
a project, usually two to the screen at a time, including calendars, spread-
sheets matching activities and resources, GANTT and PERT charts, and loading 
and resource views. The overall structure builds a project as a hierarchy, 
from the overall task down to particular subtasks and individual resources. 
It's a nice way to look at a project, formally known as Work Breakdown 
Structure (a module Software Publishing has just added to Harvard Total 
Project Manager). Professional project managers (such as those who might 
use TREES or even less power-user types) will easily pick holes in Insta-
Plan, just as accountants can fault NewViews for its lack of built-in 
accounting modules and structures.

InstaPlan is designed more as a thinking and communication tool -- for plan-
ning and presenting relatively simple projects -- and there it fits very 
well. It's ideal for people like us, who are too busy doing instead of 
planning to sit down and use a full-scale project-management system.
THE NEXT GENERATION: DOS AND ANTE-DOS

With Microsoft making its plans for 80386 DOS clearer (or at least clearly further out than many people had been hoping), attention is shifting to other means of bridging the gap. In particular, the hypervisors and operating systems promised last fall (Release 1.0, 2 December) are starting to ship, and to be used.

Microsoft will be selling an early version of the DOS 286 toolkit within six months, the company says. That does not mean that the final version of the OS itself will be available, but merely that it’s fixed enough that developers can start writing applications for it. The DOS itself should be out sometime early in 1988, followed shortly by the 80386 version.

Why so long? In part because Microsoft is writing two OSes at once, and the 80286 is the harder one. (The 80386 comes second for marketing reasons, we imagine. Moreover, one of them had to stand by while the other was put into final test and release.) The 80386 DOS will be able to rely on the 80386 for emulation of the 8088 environment, while the 80286 version must do so in software. The major difference between the two OSes to the developer is memory management. The difference between the two hardware environments also includes the 80386’s 32-bit instruction set, which developers can use now with the appropriate compilers.

Taking steps

Not everyone is waiting passively for the 80386 DOS. There are ways and ways to benefit from its presence without the use of 386 DOS (WinDOS, we like to call it). Those start, of course, with UNIX/Xenix, which requires a relatively full rewrite of DOS programs, but which looks like an increasingly appealing option now that UNIX and Xenix are converging. Then there are the approaches that offer memory management while still relying on DOS for I/O (Softguard and Phar Lap OS extensions, for execution of code above the 640K barrier), various forms of EEMS (for swapping of code) or EMS (mostly just for swapping of data), and compilers that generate 80386 instructions within the 640K space.

Remember, of course, that just plopping a 640K-limited program into 2 megabytes or even 16 megabytes of memory isn’t going to do much -- unless, of course, the program has previously been working with (and hampered by) heavy use of memory-management tricks and can now just stretch out in the extra space. Consider Ansa’s Paradox, which has 600K of code, and could easily be converted to take full advantage of all the real memory it can get.

Oracle, which earlier had written a limited PC version of its mainframe Oracle dbms, has now done a substantial readaptation to put a full-featured version onto the 386, using its own memory-management system to bypass DOS. Except for the underpinnings, it’s identical to the mainframe/mini version rather than to the current PC version, although it still uses the PC’s 16-bit instruction set (for the moment).

Focusing on speed rather than size, Symantec has gone ahead and implemented an 80386-specific version of Q&A that still fits within 640K, but with 386 instructions generated by Metaware’s High C compiler. Since Q&A’s Intelligent Assistant is compute-intensive, the product benefits substantially,
with performance on its natural-language queries improved by 10 to 30 percent. Moreover, the code size is approximately 15 percent smaller because of the more efficient 80386 instructions. Space for data is further cleared by use of the LIM EMS spec to store data but not executing code in RAM above the 640K memory barrier. However, Q&A 386 still relies on DOS 3.x for memory management and other OS functions.

Finally, we hear that Novell is working with The Software Link’s MOS-386, presumably on some kind of multi-tasking 386-based server.

UNIX

Many folks are looking on UNIX with renewed favor, especially now that AT&T is taking an active role in promoting the creation of a standard implementation of this heterogeneous beast across many machines, and Apple will be offering UNIX on the Mac II this summer. For the moment, that will still the only way to get multi-tasking on the Mac.

Also promoting UNIX will be IBM’s RT PC. IBM last month announced the machine that should have been (as did Apple with its stunning Mac II). Little noticed, however, was the repositioning of this more powerful system. In short, the RT PC is now IBM’s standard-architecture multi-user machine both in the small-business and the scientific markets, as well as a technical/professional workstation. The system’s networking and data management capabilities have been significantly improved, along with virtually every metric of performance.

Yet precisely because it is standard-architecture (AT&T’s nominal standard, no less), the RT PC has been neglected in IBM’s recent announcement of its Systems Application Architecture. IBM assures us that the RT PC will support most of Systems Application Architecture and vice versa. But when push comes to shove, the RT will have to support industry standards because that’s its designated role.
SYSTEMS APPLICATION ARCHITECTURE

Perhaps one of the least understood but most important concepts floating around is IBM's Systems Application Architecture. Like IBM's SNA (Systems Network Architecture), this is not a product but a concept, an architecture, a specification by which products can be implemented. Contrary to widespread impression, it will not magically allow all IBM software to run on any IBM machine, but rather will provide a set of specifications so that future software written to its programming interfaces and by its rules will be easily portable among three architectures: the Intel x86 line, the System 3X, and the 4300/370 line. Developers observing the specs will henceforth offer similar interfaces from environment to environment, but in the meantime IBM and others will have to spend a lot of time rewriting existing front-ends.

Just as an operating system insulates an application from its hardware (in this context current PC-DOS is just an I/O system), Systems Application Architecture will insulate higher-level application specs from details of implementation in different environments. It will enable developers to work at a high level, independent of target machines, operating and communications systems, and data base architectures. Its value lies not within single applications, but in the interfaces and commonalities among them.

Allowing applications to be easily ported to other machines doesn’t guarantee that you’d want to. (Who wants to run IMS on a PC, anyway?) More interesting, Systems Application Architecture will enable applications to cooperate across architectures, using common programming interfaces. Many applications will be decomposed into constituent parts, each for execution in the appropriate environment. For example, most interface activity will be implemented on the PC, while heavy-duty number-crunching will occur on a back-end mainframe, data base transactions will occur on a data base machine, and some expert system modules will operate on an inferencing machine. But the PC, or some local machine, will edit and compose users' data base queries for the dbms machine, so as to limit costly, time-consuming communications.

Such cooperative processing is the model espoused by Sybase, with its Data-Server architecture. Oracle has recently adopted it too in its new line of dbmses for networks, with a workstation and a server component; the server need not be 386-based but could also be a standard mini or mainframe running Oracle. (While Oracle says it currently has no arrangement with IBM for its 386 PCs, Oracle’s compatibility with DB2 means that it de facto fits handily into Systems Application Architecture.)

We see more and more developers glomming on to such a cooperative architecture. One of the most general examples is Network Innovations, which offers a link between UNIX/SQL dbmses and standard PC/DOS applications such as 1-2-3 and dBASE. Answer Systems is providing similar connectivity between PCs and IBM’s mainframe-based DB2.

The IBM documents make interesting reading, since they implicitly outline which products are strategic... DB2 makes the grade, as does PROFS.
Symbolics recently held an analysts’ briefing to tell the world it had never beaten its wife: It always had the standards they are clamoring for, but hadn’t thought to promote the fact. Like Apple but more so, Symbolics has suffered from its image as an offbeat system unsuitable for normal use. After suffering from a deep slump last year, the company has reorganized, brought in a new president (Brian Sear, with a long career in electronics-company management); pulled back to its Cambridge headquarters from an expansion to luxurious new digs in Concord, MA; and dropped surplus people.

The message is: Whatever it is, we’ve got it -- SNA, Pascal, Ada, Fortran, CO-- well, maybe not COBOL, but everything else. Just last week Symbolics hired marketing director Annie Brooking away from Sun, where she certainly had a chance to learn about standards and commercial markets.

Sear plans to launch an aggressive advertising campaign making the valid points that its machines aren’t as expensive as everyone thinks, and that they do support standards as well as a uniquely wonderful and productivity-enhancing AI programming environment. With its new positioning clear, the company is now moving forward, working on such projects as a C language, further communications enhancements, and some kind of co-processor arrangement with a 386 chip.

Such a standard/LISP box would be easy enough to build -- and an attractive offering in a specialized market -- but the key to the effort will be the establishment of logical rather than physical connections between the two environments. The biggest complaint about LISP machines is still their inability to work tightly with standard data bases and operating environments. If only Symbolics could just nestle neatly within IBM’s Systems Application Architecture.

Palladian shifts

Meanwhile, one of Symbolics’ most visible independent software vendors, Palladian Software, has moved its Financial Advisor, renamed and repositioned as Management Advisor, onto a more standard machine -- the Apollo 3000 series. (It will also remain on the Symbolics and the TI Explorer.) The price of the software -- $66,000 per user before quantity discounts down to $5,000 for the 25th user -- stays the same, but the hardware cost is lower, between $25,000 and $30,000, vs. about $50,000 for a Symbolics. Performance is close to that of the TI Explorer and a little less than that of the Symbolics, says Jim McGowan, the company's new senior vp of marketing and sales, formerly with IBM. More important, there is a large installed base of Apollo machines, and they can run MS-DOS, 1-2-3, etc.

Because the new version is implemented entirely in Common LISP (Lucid LISP on the Apollo), it could probably move quite easily to still other machines, such as Suns and 386es. Yet the problem of tight interconnection won’t go away simply because of the hardware platform. Palladian won’t commit to trading data easily back and forth between Management Advisor and 1-2-3, although we assume such a capability is in the works. Nor does MA work closely with standard dbmses -- although we might note the close ties between Palladian and Bachman Information Systems, which has lots of dbms SQL expertise. (Cooper is also chairman of Bachman.)

Release 1.0

24 March 1987
PHONE NUMBERS

Larry Tesler, Harvey Alcabes, Apple, (408) 973-2219, 996-1010
John Willoughby, Avyx, (303) 790-0514
Carl Nelson, Carl Nelson Associates, (206) 252-6897
Jim Anderson, Digitalk, (213) 645-1082
Robert Carr, Forefront, (408) 735-0740
Eddie Currie, Lifeboat/Hudson Technology, (914) 332-1875
Chris Grejtak, Rich Carpenter, Index Technology, (617) 491-2100
Gary Cole, InstaPlan, (415) 325-2551
Adam Hertz, Lotus, (617) 577-8500
Tandy Trower, Rob Dickerson, Microsoft, (206) 882-8080
Greg Kee, Gasya, (617) 491-4180
Gen Shklar, Oracle, (415) 598-8000
Phil Cooper, Jim McGowan, Palladian, (617) 661-7171
Adele Goldberg, ParcPlace Systems, (415) 494-4380
Joel Schwarts, Ron Kalin, Polygen, (617) 890-2888
Dennis Sisco, Productivity Products International, (203) 426-1875
Steve Burbeck, Softsmarts, (415) 327-8100
Gordon Eubanks, Symantec, (408) 253-9600
Brian Sear, Symbolics, (617) 577-7500
Jeff McKenna, Tektronix, (503) 627-5278
H.D. Grupenzusammen, Vienna Institute, (44) 123-5711
Mark Achler, Chuck Duff, The Whitewater Group, (312) 491-2370

COMING SOON...

• The IBM announcements...
• Natural-language processing.
• Text: Structure and search.
• Nitty-gritty experts.
• And many more...

Release 1.0 is published at least 15 times a year by EDventure Holdings, 375 Park Avenue, New York, NY 10152; (212) 758-3434. Editor & publisher: Esther Dyson; associate publisher: Sylvia Franklin; circulation manager: Hyacinth Frederick. Copyright 1987, EDventure Holdings Inc. All rights reserved. No material in this publication may be reproduced without written permission. However, we gladly arrange for reprints or bulk purchases.

Release 1.0 24 March 1987
March 26-29

March 25-28
SOFTWARE PUBLISHERS ASSOCIATION SPRING CONFERENCE - Oakland, CA. Including "Corporate users on critical issues" -- not just kid stuff. Contact: Ken Wasch, (202) 452-1600.

March 29-April 1
ADAPSO SPRING MANAGEMENT CONFERENCE - Orlando, FL. Worth flying from Oakland to Orlando for. With a panel on development tools moderated by Paul Hessinger of Computer Task Group, a panel on AI tools moderated by Lou Odette of Apex, a panel on the 80386 moderated by Esther Dyson, as well as panels on software maintenance, micro-mainframe links, and other compelling topics. Call Sheila Wakefield, (703) 522-5055.

March 30-April 2
Ninth annual international conference on software engineering - Monterey. Sponsored by IEEE and other societies. With Robert Balzer, USC ISI; Larry Druffel, Software Engineering Institute (formerly with Rational); Ram Banin, Codesmith; Gerry Davis, Schroeder, Davis & Orliss; and others. Also, a "Tools Fair." Contact: Larry Druffel, (412) 268-7740.

March 30-31
Solutions showcase - Los Angeles. Sponsored by Ashton-Tate, to feature dBASE and other vertical market applications. Contact: Jeanne Jalan, (213) 538-7783.

March 30-April 2
Interface '87 - Las Vegas. Sponsored by the Interface Group, for corporate, government and institutional users. Contact: Keith Westerman or Linda Hanson, (617) 449-6600.

April 5-9

April 6-8
Facsimile and image communications systems conference - Boston. With speakers from Datacopy, Fujitsu, etc. Sponsored by CAP International. Contact: Jean O'Toole, (617) 837-1341.

April 6-9

April 7

April 8
AI Satellite Symposium - your place. Third in a series, with a focus on "AI productivity." Speakers include Ed Feigenbaum, TI's own George Heilmeier, Alan Kay, Herb Schorr, Doug Lenat, James Martin. Sponsored by Texas Instruments. Call (800) 527-3500 for information on hooking up.

April 15  NYNEX at the analysts - New York City. Sponsored by the New York Society of Security Analysts. Contact: Judy Fontana, (212) 344-8450.

April 16  American Software at the analysts - New York City. Sponsored by the New York Society of Security Analysts. Contact: Judy Fontana, (212) 344-8450.

April 20-22  CD-ROM vs. micrographics - Monterey, CA. The transition from microfilm to CD-ROM. Sponsored by the Institute for Graphic Communications. Contact: Gail Montgomery, (617) 2679425.

April 21  Hogan Systems at the analysts - New York City. Sponsored by the New York Society of Security Analysts. Contact: Judy Fontana, (212) 344-8450.

April 22  American Management Systems at the analysts - New York City. Sponsored by the New York Society of Security Analysts. Contact: Judy Fontana, (212) 344-8450.

April 23  MCI Communications at the analysts - New York City. Sponsored by the New York Society of Security Analysts. Contact: Judy Fontana, (212) 344-8450.

April 22-24  AEA financial conference for public companies - Boston. Meet with as many as you can manage of 50 electronics/computer companies; mostly for financial types. Contact: Dave McKell at (415) 857-9300.

April 22-24  AI Long Beach - Long Beach, CA. Contact: Jim Hay at Tower Conference Management, (312) 668-8100.

April 25-26  Softeach - Chicago. Sponsored by Softsel. For information, call (800) 325-9189, (314) 225-1724, or (416) 629-2222.


May 6-8  ICP Million-Dollar Awards & Conference - Indianapolis. Sponsored by International Computer Programs, Inc. With the one and only Larry Welke, plus an agenda of sales & marketing topics. Contact: Judy Fary, (314) 844-7461.

May 7-9  Fortune/Seybold Desktop Productivity Conference - New York City. The usual stellar array. Call Beth White or Sandy Seybold, (213) 320-9151, or Carol Federighi, (408) 297-0888.

Release 1.0 24 March 1987
<table>
<thead>
<tr>
<th>Date</th>
<th>Event Description</th>
<th>Location</th>
<th>Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 18-21</td>
<td>Navy Microcomputer Conference - Virginia Beach, VA. Sponsored by the Navy Regional Data Automation Center.</td>
<td>Virginia Beach</td>
<td>Karla Rowlett at (804) 444-8486.</td>
</tr>
<tr>
<td>May 21-22</td>
<td>TECHNOLOGICAL SUPPORT FOR WORKGROUP COMPUTING - New York City. Sponsored by NYU.</td>
<td>New York City</td>
<td>Marge Olson, (212) 285-6077.</td>
</tr>
<tr>
<td>May 27-29</td>
<td>CASE '87 - Cambridge, MA. &quot;First international workshop on computer-aided software engineering.&quot; Sponsored by Index Technology, Purdue and Northeastern Universities, Boston ACM; real-world but nonpartisan.</td>
<td>Cambridge</td>
<td>Elliot Chikofsky, (617) 491-2100.</td>
</tr>
<tr>
<td>June 1-3</td>
<td>SPRING COMDEX - Atlanta. Incorporating winter Comdex, by popular request.</td>
<td>Atlanta</td>
<td>Linda Yogel, (617) 449-6600.</td>
</tr>
<tr>
<td>June 15-18</td>
<td>National Computer Conference - Chicago. Sponsored by AFIPS and a host of other societies.</td>
<td>Chicago</td>
<td>Martha Byrne at (800) NCC-1987 or (703) 620-8925.</td>
</tr>
<tr>
<td>July 13-17</td>
<td>AAAI-87 - Seattle, WA. So good, they made it earlier this year.</td>
<td>Seattle</td>
<td>Claudia Mazzetti at the American Association for Artificial Intelligence, (415) 328-3123.</td>
</tr>
</tbody>
</table>

Release 1.0 24 March 1987


September 21-23  Conference on software maintenance - Austin, TX. Sponsored by several professional societies. Contact: Roger Martin, National Bureau of Standards, (301) 921-3545.

September 27-30  ADAPSO management conference - Colorado Springs, CO. Contact: Sheila Wakefield, (703) 522-5055.

October 4-7  Computer Services Forum - Baltimore. The standard in Wall Street software events. Call Chris Mortensen, (301) 727-1700.

October 4-8  OOPSLA '87 - Orlando, FL. The second annual conference on object-oriented programming, sponsored by ACM and chartered by Adele Goldberg (ParcPlace) and Chet Wisinski (PPI). If you liked this issue, you'll love OOPSLA! Contact: Jerry Archibald, (914) 789-7695.


Please let us know of any other significant events we should include.

Release 1.0  24 March 1987
SUBSCRIPTION FORM

Please enter my subscription to Release 1.0 at the rate of $395 per year in the U.S. and Canada. Overseas subscriptions are $475, airmail postage included. Payment must be enclosed. Multiple-copy rates on request.

Name ____________________________________________

Title ____________________________________________

Company _________________________________________

Address __________________________________________

City ____________________ State _______ Zip _________

Telephone ________________________________________

Please fill in the information above and send with your check payable to: EDventure Holdings Inc.
375 Park Avenue, Suite 2503
New York, NY 10152

If you have any questions, please call us at (212) 758-3434

Sylvia Franklin
Associate Publisher

Release 1.0 24 March 1987