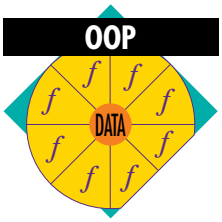


Why Invest in Object-Oriented Programming?

By Dan Hattenberger

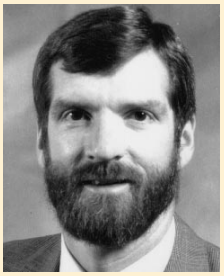
Object-Oriented Programming (OOP) seems to be the latest programming technology with promises of increased programming productivity, quality, and flexibility. However, software development managers charged with making money, and Information Technology (IT) managers charged with living within a budget need to know whether investments in new technologies will pay for themselves in either increased revenue or cost savings. This article examines OOP from a business perspective with some real-world examples.

Object-Oriented Programming (OOP) can provide considerable benefits to many programming companies or departments. Any software development organization that is plagued with numerous updates and changes, offers (or wants to offer) customized software solutions, or tries to reuse software in order to save money should consider object-oriented programming.

But before they do, that organization needs to address some real-world issues:

- **Initial investments:** Getting started
- **Ongoing costs:** New ways of doing business
- **Savings:** Productivity, quality, and reduced maintenance
- **Potential revenue:** Making money with OOP

Although many OOP articles and books address the first three issues, some discuss only the savings, and only a few address how to make money with OOP. This article addresses each of these topics, discusses some good ways to approach them, sets some realistic expectations for each based upon history and case studies, and outlines ways to get into OOP without breaking the bank.



Dan Hattenberger

Initial Investments

The three largest initial investments are training, software, and equipment. The first, and most important, is training.

Training

Programmers accustomed to traditional programming (where programs reside in one place and data resides in another—all development follows a design-code-test sequence) must learn to think in terms of objects and classes. Dr. Tom Love, a well-known OO consultant and author, recommends an education plan for OOP that should include the following elements[1]:

- OO Concepts (1 week)
- Language (2 weeks)
- Pilot Project (4 to 6 weeks)
- Initial Design (1 week)
- Initial System (20 to 40 weeks)

This plan reflects system development versus application development. However, business partners experienced in OO development also recommend the same elements with shorter times. OO Concepts should include basic OO concepts as well as analysis and design. New OO programmers must understand OO analysis and design. Without this paradigm shift, they revert to traditional methods in a new language and never

make the transition to "object-think."

Language training can be done many ways. Love suggests learning in two weeks. At IBM Rochester, however, where large portions of OS/400® are developed by using C++, programmers learn the language through a university course with one four-hour session each week plus assigned exercises over a span of eleven weeks. Programmers then take a twelve-week, four hours per-week design and analysis class. This new training method enacted after the first attempt at language training—a two-week course—failed because students could not absorb all the material at once, and they had forgotten much of it when they tried to use it.

The next phase, the pilot project, is extremely important. Programmers learning OOP should be assigned a meaty project that will challenge them without breaking the company in case it fails (as many projects do). An experienced object-oriented programmer (a mentor) should help the new programmer in this effort, providing suggestions for good OO designs that take advantage of the class libraries and that foster reusability. This will be described in more detail later.

Software and Tools

The second investment is in software and tools. OOP tool and framework development is in full swing today. By the middle of 1996, developers should have an excellent selection of development tools, a range of frameworks, and basic OOP support on all platforms. Ever-increasing competition and rising demand make it difficult, however, to predict the pricing of these tools and frameworks.

A software development company or department contemplating OOP needs at least one person in training today who can evaluate and select tools and frameworks. Many tools are available today, including evaluation versions. When evaluating tools, keep in mind that many are available in single-copy and team versions. Evaluate the team version—it is generally more powerful and is designed for multiple programmers.

Equipment

The last consideration is hardware for development. OOP tools run on PCs, and, depending upon the tools chosen, require lots of memory and power. Visual development tools, such as VisualAge™, are extremely powerful and allow the computer to do much of the work. Therefore, they require a powerful computer such as a 50 MHz (or higher) system with 20 MB (24 MB or 32 MB is better) of memory, and at least 540 MB of disk. Less robust tools cost less and require less computer horsepower. In general, the more powerful the tool, the more it costs to purchase and run.

Ongoing Costs

There are always costs involved in writing and maintaining applications. The large number of ready-to-use classes can take a significant amount of time to learn. To leverage reuse, programmers must design reusable parts, which takes even more time, particularly for inexperienced OO programmers. In fact, a company's first project often takes longer than a traditional implementation (sometimes as much as twice as long). The real cost savings happen in succeeding projects as the reusable libraries grow. Programmers will always create new objects and methods, so reuse is never 100%, and the amount differs for everyone. Over time, the ongoing costs for learning the classes and doing OOP design decline with more experience.

The three largest initial investments are training, software, and equipment.

Savings

Most OOP benefits are defined in terms of savings or increased productivity. Unfortunately, most of the benefits of any programming technology—3GL, 4GL, CASE, code generators, Graphical User Interface (GUI) tools—are defined this way. After a while, they all sound the same. What makes OOP different and what can it provide?

Code Reuse

The significant savings come from reuse. OOP "languages" are more than compilers; they include class libraries, editors, browsers, and debuggers. The class libraries contain thousands of pre-defined classes designed for reuse and tested for quality. These libraries contain only very generic classes—the trick is to use these small classes to create other, more robust classes that define objects specific to the customer world. As the number of "business-specific classes" grows, the application programmer reuses more code and writes less (OOP productivity comes from coding less, not faster), and the savings increase with each succeeding project.

Many software companies are trying to do this today but lack the appropriate tools. They enforce code reuse, scan existing programs, recall programs from old tapes, or just plain remember what they did last time. On the other hand, OO

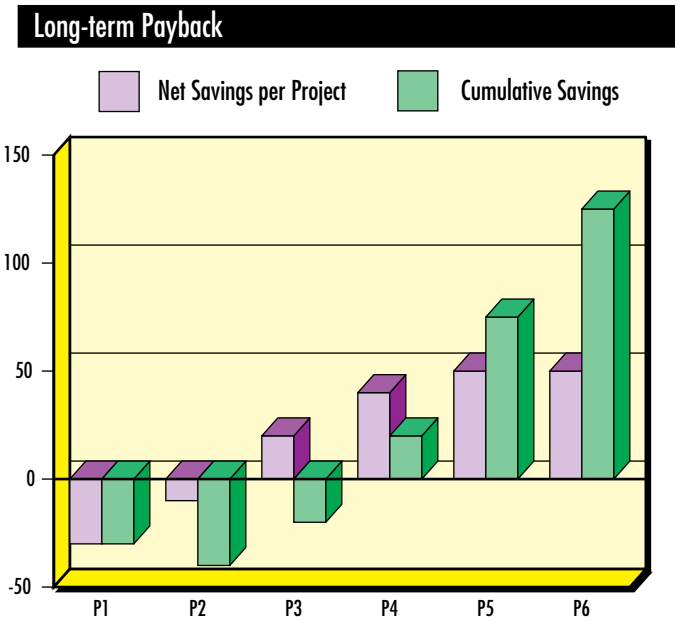


Figure 1. The long-term payback of OOP vs. traditional programming

Amount of Reuse	Development	Effort to Reuse	Development Effort	Savings
	(person-months)	(person-days)	(person-years)	(%)
0	100	0	8.3	0
10	90	10	7.5	9
30	70	30	6.0	28
50	20	80	2.0	76
80	20	80	2.0	76

Figure 2. Estimated effort to create a system of 100 classes

programmers use a browser to search for existing classes that fit the current need.

Figure 1 compares the potential savings of OOP versus traditional programming. Although the first few projects show negative savings as the reusable parts are developed, savings begin to accumulate after that.

Frameworks are large, reusable parts that come in several varieties and contain the logic, flow, and most of the key business objects that make up the backbones of applications. Programmers customize the framework by providing additional objects, modifying the objects supplied, and/or rearranging the parts and flow. By starting with larger parts created by experienced OOP programmers and designers, development shops can move to object-oriented applications with less effort and less risk. The key to success is to shop around for the framework that most closely matches the application requirements in function, platform portability, and ease of adaptation.

Productivity Gain

How much productivity gain can be expected? In one example[2], a new realty company in California wanted a state-of-the-art application suite that included a listing database, market analysis, client management, personal marketing, agent messages/status, and an interface to the Multiple Listing Service. An experienced OO programming team installed all the applications in 45 days and 1,000 person-hours. They succeeded because they made extensive use of an existing class library that they knew very well.

In a carefully controlled study[3], Electronic Data Systems (EDS) Corporation compared traditional programming to OOP. They replicated an existing manufacturing system that was originally implemented in PL/I and a relational database. EDS formed a team of Smalltalk programmers comparable in experience to the PL/I team. This team started with the same set of specifications, wrote the system, and tested it with the original test suite. The Smalltalk team completed the project in 3.5 months (versus 19 months for the PL/I team), with 10.4 person-months (versus 152), and in 22,000 lines of Smalltalk (versus 265,000 lines of PL/I). The EDS example shows a key characteristic of OOP productivity: the reduction in effort (14:1) closely resembles the reduction in code (12:1). This example is a widely quoted one, but might not be typical (14:1 is unusually high).

A better example is a company that develops command and control systems for the Swedish defense services[3]. They were faced with providing similar, but different, command, control, and communications applications across different ships. Recognizing early that they needed modularity and flexibility, they used an object-oriented design to model the shipboard system while ensuring development of reusable software components for future use.

Because the applications were designed for reusability, succeeding implementations were easier and cheaper. In fact, after completing two installations, this company estimated a 6:1 productivity improvement in succeeding installations, due mainly to 70% code reuse.

Managers contemplating OOP should consider the amount of potential reuse in their application development. The chart in Figure 2 shows some estimates of cost reductions for different levels of code reuse[1]. The savings ratios are not as high as the reuse ratios, because there are other costs associated with design, integration of parts, and testing of the whole application.

About 20 years ago, the IT manager of a large corporation stated, "Users never know what they want—they only know what they don't like when they see it." Customers seldom do a good job of defining their needs at first. Analysts define the requirements and provide specifications, designers design applications, programmers code the application, and then, often several months later, the customer gets a first look at the product. It was probably shortly after one of these "first look" sessions when the IT manager made her statement.

OOP design is an iterative process of designing, prototyping, examining or testing, then repeating those steps as necessary. Reusable classes, particularly the GUI classes, make it easy to prototype the front end of an application and review it with the customer at the beginning of the development cycle. As more code is added to the prototype, other reviews are possible (highly recommended). These iterative prototype/review cycles help eliminate costly rewrites due to changing customer desires or changes in the customer business.

Maintenance

Poor quality is expensive in terms of rework and customer sales. A survey taken several years ago revealed that customers often made a product choice based upon talking to someone who had already purchased the product. Even though there are few quantitative measurements of OOP quality versus traditional program quality, new OO programmers often notice an increase in quality—fewer bugs are created and most are more easily fixed when using OOP. Many companies do not rigorously measure quality during the development cycle, but most programmers tend to feel that initial quality is higher with OOP.

Many software development companies and departments attack costs in application development, but ignore the costs of maintenance and upgrades later. Based upon informal polls at presentations and classes, most programmers and managers estimate that programmers in their companies spend from 50% to 80% of their time on maintenance, meaning that only 20% to 50% is spent doing new development. Other studies indicate that only 30% of programming time is spent on new development, 14% on fixing bugs, 14% in adapting applications to environments, and 42% in adding to the application (enhancements)[4]. So, in one way or another, as much as 70% of a programmer's time can be spent on maintenance. Over the life cycle of an application, the total cost of maintenance can be four to eight times the original development cost.

Maintenance never seems to make programs smaller—every change generates more code. And more code generates the possibility of more bugs. The effect is similar to compound interest. OOP reduces this code bulk while contributing fewer bugs. This combination reduces both the number of update iterations and the effort required for each. Over the life of an application, these savings are substantial. In the real estate example cited earlier, maintenance during the first year was limited to half a person, but that included three major revisions. This aspect of OOP is being measured, but the results are not yet available. One thing is clear—managers striving to cut costs should aim at the biggest contributor: maintenance.

Managers striving to cut costs should aim at the the biggest contributor: maintenance.

Revenue

Save, save, save! All software companies and IT departments want to save money by producing high-quality software and services. However, if the customers stop buying the product, the software company dies. If the end users turn elsewhere for computing needs, the IT department is budgeted out of existence. Cutting costs will only help these businesses to survive. To expand, they need ways of increasing revenues or convincing someone else to increase the budget. How can OOP help?

Increased revenues generally result from expanding into wider markets or from increasing current market share. Most companies widen their software market by increasing the scope or number of applications to cover more business segments, by increasing the number of platforms supported, or by doing both. Increased market share results from either offering more value-add, outselling the competition, or a combination of both.

Platform Portability

Moving to more development platforms requires software portability. Today, industry-wide standards are nearly completed for Smalltalk and C++, and both languages are available on most

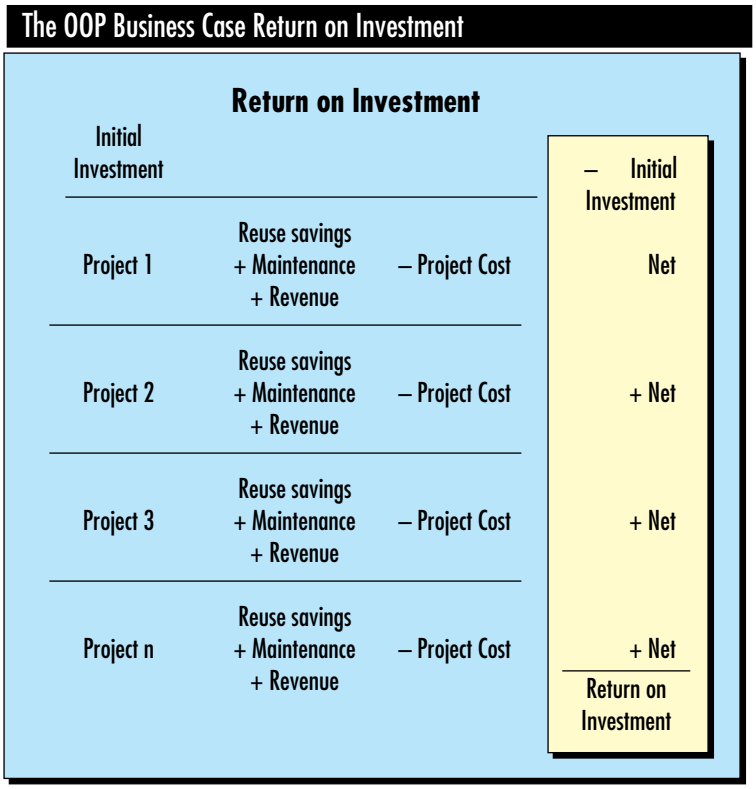


Figure 3. A postive return on investment is built over time

system platforms. With a traditional language, these two facts imply portability. With OOP, applications are made up of objects and statements. The standards address the statements, but not the class libraries from which objects are created. Therefore, developers shopping for an OOP language need to consider what platforms the language provider supports and how consistent the class libraries are across these platforms. Most providers, including IBM, are trying to create libraries that are consistent across a variety of platforms, thereby allowing developers to develop and test on one platform and easily move the application to another one.

The Object Management Group (OMG), a consortium of hundreds of software companies, has already drafted standards for object interoperability across languages and platforms. The Common Object Request Broker Architecture (CORBA), from OMG, defines standards that will allow different objects created in different languages on different platforms to communicate with each other consistently and cleanly. At this point, portability becomes reality. IBM developed System Object Model (SOM) based upon these standards.

The tools and languages currently available and those being developed support client/server applications and graphical user interfaces. While non-programmable terminals are supported, the technology is geared toward flexible GUIs through intelligent workstations connected to data and application servers. Many developers with host-based applications are being pressured to connect to the desktop PC as end users become more comfortable with iconic, graphical interfaces. OOP languages generally come with a rich library of GUI classes that make it relatively easy to create, and later change, the user interfaces.

Icons map well to GUI objects. With a visual programming tool such as VisualAge, programmers can construct or modify screen layouts quickly and accurately. The resulting applications are much more appealing than text-based applications, making them much easier to sell.

Customization

The greatest justification for moving to OOP lies in customization. Customers have made the transition from, "I will take whatever you can offer" to "Here is my list of requirements—meet it, or I can probably find it elsewhere." While a product's functions and features are still key buying factors, many customers want, need, and, most importantly, will pay for applications tailored to their specific needs.

OOP enables programmers to quickly add to and change an existing object without tearing the application apart. The new subclass contains only the changes, inheriting the rest of the data and methods from the original class (superclass). Later, the programmer can use the browser to scan through these newly created classes to see what changes were made. Encapsulation and good OOP design enforce clean object interfaces—it is more difficult to create "spaghetti code" and "patch on patch" messes, even in the heat of customer crises. Polymorphism allows the interface to the surrounding application to remain the same in most cases, isolating changes and reducing the chances of introducing a new problem while fixing another.

For developers doing customization today, this technology is a natural next step to increase the number of customers that can be supported. For others, OOP might be the way to raise their value add to the customer, close more business, and make more of that lucrative services revenue.

The Business Case

Anyone considering object-oriented programming should remember that the benefits are primarily long-term ones measured across a number of projects. It will take time to build up a library of reusable parts. But as programmers gradually learn their way around the class libraries, the application offerings will gradually expand. Figure 3 shows a summary of the business case.

What's Next?

Even when the business case shows a positive return, software companies must have a plan to move from one technology to another without disrupting the revenue stream. IT managers must make the transition without a loss of service and within budget. Most developers who have successfully made the transition used one of two methods.

Method 1. The most popular method is to take a small subset of people, remove them from day-to-day responsibilities, send them to some classes, give them a project to work on as part of their training, and find a mentor who can help steer them in the right direction. Using a mentor will greatly decrease the learning time and greatly increase the chances for eventual success. New OOP programmers might create what they think is an elegant, object-oriented design but, without experience, cannot even determine whether it is good or bad. This is where a mentor comes in. Some companies hire mentors, and others contract with them to varying degrees throughout the first application design. After the initial design, many companies begin to add others to the team, using experienced programmers to mentor their peers.

Method 2. The second method is a quicker method for companies that face tighter deadlines. These companies hire skilled mentors and OOP programmers up front to do the initial design and much of the high-level implementation. While this is happening, other employees begin training and are added to the team later for the bulk of the implementation. For this to be successful, it is important to employ skilled, experienced (and successful) programmers from the start. The investment level in doing so is high—good OOP people are not inexpensive—but the success ratio is also very high.

Regardless of the method chosen, those managing an OOP project must be trained along with the programmers. The iterative, prototyping nature of design and implementation does not match up well with the traditional design/code/test checkpoint system. Tossing out an initial prototype design is not a setback—progress has been made. On the other hand, managers must recognize when iteration becomes over-analysis. Measurements change from lines of code written to code reused.

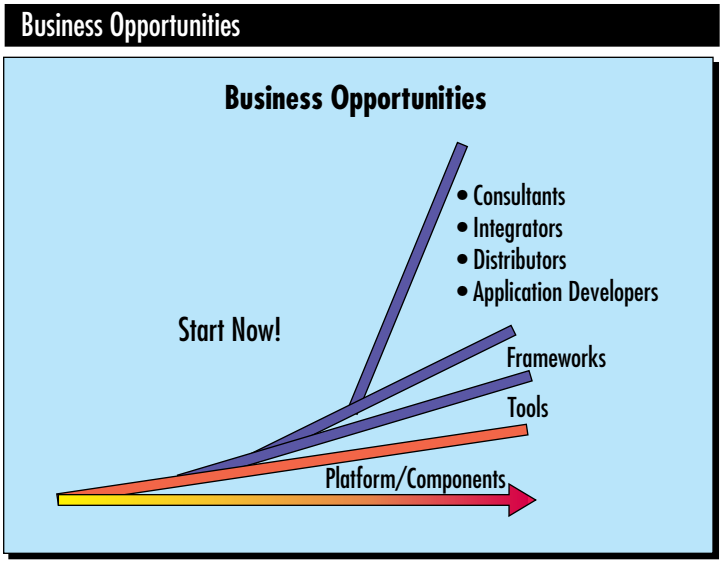


Figure 4. OOP is creating many business opportunities

Many management adjustments are required. Often, companies follow the "recipe for disaster"—they set an aggressive schedule, use inexperienced and recently trained people, try to save money by not hiring anyone with skills, track the project like any other important project, and, in the end, blame the failure on the technology (or the tools or other individuals). Programmers and managers who are asked to take on such a challenge (or already have) should stop and reconsider reality. It takes time to develop OOP skills, and these skills are essential for success. One way to turn these companies around is to hire a good OOP consultant to evaluate the project, educate management, and recommend changes.

Summary

The object-oriented business is expanding rapidly, and, like PCs, the offerings will only grow in number and function, and prices will constantly change over the next several years.

Figure 4 shows the OO industry as it matures. The platform and basic components are nearly in place, and frameworks are now under development. Once this occurs, widespread use will begin to increase rapidly. The companies that invest today to understand this technology will be

in the best position to evaluate these new offerings and take advantage of them quickly. This technology is like a train in motion—you can hop on anytime, but you have a better chance if you get a running start. At the very least, even if you are not yet sold on object-oriented programming, make sure you understand it enough to make a good decision.

1. Love, Tom. *Object Lessons: Lessons Learned in Object-Oriented Development Projects*. New York, NY: SIGS Books, Inc. ISBN 0-9627477-3-4.
2. Harmon, Paul and Taylor, David A. *Commercial Applications of Object-Oriented Technologies*. Reading, MA: Addison-Wesley Publishing Company. ISBN 0-201-63336-1.
3. International Data Corporation. *Object Technology: A Key Software Technology for the '90s* (white paper). Steve McClure, New Software Technology, at International Data Corporation.
4. Goldstein, Neal and Alger, Jeff. *Developing Object-Oriented Software for the Macintosh: Analysis, Design, and Programming*. Reading, MA: Addison-Wesley Publishing Company. ISBN 0-201-57065-3.

Dan Hattenberger, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Hattenberger recently moved from the AS/400 Laboratory in Rochester to join the Object Deployment group within Solution Developer Operations to create OO solutions for AIX and OS/2 . He has a BA in Mathematics from St. Mary's College of Minnesota.