Object - Based Computing

Published Monthly

A programmer's guide to object based computing on the NeXT Computer.

PREMIER ISSUE

VOLUME 1, NUMBER 1, JUNE 1992

editorsDesk

Welcome to our first issue

The first issue of any new publication is a difficult time. While the vision for starting the publication, and its purpose is clear -- in our case to provide a forum for NeXT programmers to discuss object-oriented programming and interpersonal computing (Object-Based Computing for short) -- the publication has no readers as yet.

Any good publication is considered to be so because it has carefully listened to the needs of its readers. As a new publication we have no readers to guide us toward "the good,." so our first attempt should be seen as an initial throw at the dart board -- a first attempt to judge distance and speed.

For us to improve our aim, and become a valuable asset to you, we need your judgement. Please send mail, electronic messages, and do not hesitate to call with comments on our publication. We want to know what you think of what we have done and, more importantly, what you want us to do to improve.

In the months ahead I plan to use this column to tell you about the comments I have received and let you know what we are doing to act upon them. Each month I will be trying to incorporate a few of your comments into the newsletter so that we can do a better and better job of hitting the bull's-eye with the NeXT programming community.

The best way to reach us is through electronic mail. If you do not currently have access to mail I strongly encourage you to change this situation. There are a number of different services which can provide mail access from your machine to the CIX network (or Commercial Internet) and from there to the rest of the world. We happen to use PSI which costs a mere $25 per month for store and forward mail access.

Thank you for becoming a reader of OBC and I will look forward to your messages.

Ted Shelton, Editor
Object-Based Computing
ems@its.com

thisIssue

A little about the articles in this issue.

This first issue of Object-Based Computing owes a great deal to Dan McCreary and his Minneapolis based NeXT development group. A conversation with Dan four months ago first caused me to think of creating this newsletter. And now, our first issue contains three articles from members of his new company, Integrity Solutions, Inc.

The first of a series of six articles on object-oriented computing appears in this issue. Written by Dan McCreary of Integrity Solutions, Inc. this series represents the first part of a new book on Object-Based Computing which focuses on the NeXT computer. It is a pleasure to include this serialized version of an outstanding book in our publication.

By Lionel P. Aboulkheir, also a member of the Integrity Solutions staff, we have an article on the "Linguistics of Object-Oriented Programming."

Finally, a second article by Dan McCreary exploring some of the reasons behind the industry switch to object oriented development environments titled "Object-Oriented Programming: The Probable Person Period."

I hope that you enjoy reading this first issue of OBC as much as we have enjoyed starting this new enterprise.

nextIssue

Object-Based Computing (pt. 2)
NeXT/OOP: A Case Study
usernamed: A Network Daemon
Flexible Object Design

index

features

400 WEST ERIE, SUITE 402
CHICAGO, IL 60610

Object-Based Computing

Dan McCreary - Integrity Solutions

(This is the first of six articles on Object-Oriented programming)

When [the Mark 1 was] first built, a program was laboriously inserted and the start switch pressed. Immediately the spots on the display tube entered a mad dance. In early trials it was a dance of death leading to no useful result, and what was even worse, without yielding any clue as to what was wrong. But one day it stopped and there, shining brightly in the expected place, was the expected answer.

F.C. Williams, June 21, 1948

All of us, professionals as well as laymen, must consciously break the habits we bring to thinking about the computer. Computation is in its infancy. It is hard to think about computers of the future without projecting onto them the properties and the limitations of those we think we know today. And nowhere is this more true than in imagining how computers can enter the world of education.

Seymour Papert, **Mindstorms**

Keeping Competitive in the Information Age

The quotes above are significant in two ways. The first quote gives us a glimpse at the staggering rate that computers have progressed in the last four decades. The second, by Seymour Papert, shows that we must constantly re-think the way we solve problems because computers constantly remove the limitations we have become used to. As we move into an information based economy we find that the more an individual thinks of and uses computers as a competitive tool, the greater the advantage gained.

The popular media in our society often talks of the dawning "Information Age" though rarely is it defined. What do we mean when we talk about the information age? Table 1 is an outline of some of the ages of man.

There are two interesting ideas that we can learn from this table. First, from early primitive times through the industrial revolution and finally into the present information age we can see that the **rate** of change is increasing.

And second, with each new age, the skills needed to excel in our society have changed. To succeed in early times we needed to be strong, fast and have endurance for tracking our prey. But as we started specializing we became mutually dependant and started to live in larger social units. This caused languages to flourish and placed demands on parts of our brains that required the ability to communicate through oral and written symbols. Because of our written culture, knowledge began to accumulate. Each generation built on the work of previous generations and added their own new contributions. Now we are at a stage in history where in order to make significant contributions to our knowledge base we often spend a third of our lives extracting the existing information from our written heritage. One of the most valuable components of our society is our educated human resources. We are now in the information age. An age where in the near future, most of the population of our planet will be involved in the creation, dissemination and translation of information.

Age	Years Ago	New Skills Emphasized
Hunter/Gatherer	1,000,000?	Strength, Speed, Endurance
Agriculture	100,000?	Understanding Environment
Skilled Craftsman	1,000?	Large Motor Skills
Industrial Revolution	200?	Small Motor Skills
White Collar Worker	100?	Communication Skills
Information Age	30?	Abstract Reasoning

Table 1: The Ages of Man

Skills for the Information Age

What are the skills we need to be competitive in the information age? Many say that one of the essential skills will be the ability to derive meaning from data. Meanings can be found by creating and combining views of data that give us insight into underlying trends. We call these views abstractions.

If we look at the way people have used computers in the past, we see that there is a natural tendency for people to take things they understand well and create abstractions of them on the computer. Examples of this include the word processor, which is an abstraction of the standard office typewriter, and the spreadsheet which is a abstraction of a calculator. In each case, our computer based abstractions allow us new freedoms to manipulate objects in a more flexible way. For example, with a typewriter, if we wanted to change the margins on a page we would have to re-type the entire page. With a word processor we have the ability to make many changes anywhere in a page without having to re-enter the text of that page.

Creating Innovative Abstractions

Although the word processor is a great productivity tool, it provides no real new innovations into the way people think. Innovative tools can be created when we realize our computer based abstractions do not have the same restrictions that the real world objects have. That is why the first spreadsheet was considered the first innovative use of a personal computer. The author of the first PC based spreadsheet (VisiCalc for the Apple II) realized that the restrictions of a single calculator were artificial. Why not have an entire array of them? Why not let them all pass

on their calculations to other cells? And with each of these innovations, we not only change the way we interact with our abstractions on the computer but we change the way we think. We change our cognitive models. We begin to mentally visualize a series of calculations in a mental picture of a spreadsheet, even when we are far away from the computer screen. And that is what object oriented programming is all about: It drastically changes the way people think. They learn to create mental abstractions of objects around them. They find relationships between old and new objects. They learn how objects interact. And they make new discoveries about the world around them because of the new ways the computer has taught them to think.

Lets examine the spreadsheet example again. Current implementations of spreadsheets force the user to put data and calculations in rows and columns. Relationships are based on **location**. But we are not limited to a matrix of characters on most of the new bit mapped screens. We should be able to create relationships based on connections between data and the calculations on that data. By dragging these objects around on the display we change the way we view the data but do not change the relationships between the data and the calculations we perform on it. We now have a new set of constraints. These constraints do not include restricting data to location. We can change a row of data to be a column and we can move the column around on the page. Now our cognitive models of understanding the relationships between the data and their operations must change based on the new constraints.

Our task is to learn two skills. The first is to be able to quickly create computer based abstractions of the world around us and integrate these abstractions into real world problems. The second is to have the ability to understand the power of the new computer systems and not limit our abstractions to be the same limits as the ones in the physical world or on other less powerful computer systems. We need to teach the ability to use creativity to create innovative views of our world. This ability to "break the chains" of previous abstractions and go a step beyond is one of the essential skills of the information age.

Decade	Method	Objects	Programmers
1950s	Switches/Paper Tape	Machine Instructions	100
1960s	Punch Cards	Assembly Language	1,000
1970s	Line Editor	FORTRAN code	10,000
1980s	Screen Editor	Structured code	1,000,000
1990s	Mouse	Tool-kits	100,000,000
2000	Voice(?)	Ultra Tool-kits	1,000,000,000

Table 2: Methods of Software Development

Object Based Computing is Creating Abstractions

So what does this have to do with Object Based Computing? Object Based Computing is the process of creating and integrating graphic abstractions of the world around us. It is not just a programming technique but a new way of structuring information and programs. It is a new cognitive style[1]. And if we are to be competitive in the information age we need to master these new cognitive styles.

The Blurring Distinction Between Computer Programmers and Computer Users

As we create more of these objects with graphical interfaces we will also find that they are not only powerful but they can be easy to use. One of the most significant aspects of object oriented programming is that by encapsulating information you build programming systems that require very little training to use. These will consist of "tool-kits" of objects which are used by pointing to them with a mouse and dragging them into the application you are building. The objects are then connected together to create entire applications. Software tool-kits will dramatically increase the number of people who can create programs. Just as the way the user friendly computers allow non-technical people to use computers, object based computing will allow non-technical people to create, customize and integrate objects into their environment. And as table 2 shows, tool-kits are just the latest step in the evolution of how we create programs.

The striking fact from this table is that every ten years there have been significant changes in the ways we create programs. Most programmers today associate punch cards with history books. And perhaps by the year 2000 we will think of procedural programming in the same way: something that was a necessary part of our evolution but so primitive we wouldn't wish it on our worst enemy.

By the year 2000 I expect to see object based tool kits so powerful that the ones we are developing in research labs today will seem like toys. By then we will take for granted the presence of full color interactive video objects, objects that will communicate with our kitchen appliances or our stereo. We will have access to remote objects at our offices, banks, libraries and our friends in remote locations. These objects will run over widely distributed fiber optic networks exchanging information at billions of bits per second. I only wish I had a crystal ball to tell us how we will create programs in 2050!

After we learn object-oriented programming techniques and start teaching others, we can put these tools in the hands of more people. We can empower them to contribute new objects to our culture at a rate faster then we could have imagined a generation ago. The distinction between **users** of programs and **creators** of programs will blur. To be competitive, the layperson will be required to customize programs to fit their problem. Finding the correct object to fit the problem

1. Presentation by Ed Barbonie, Summer 1989

will be much like going to the library to check out a book. We will be able to search large databases listing the features and connections to objects. We might even be able to rent objects that will be able to communicate with the rest of our computing environment. We will see changes in the cognitive skills we need to effectively use computers. The way we teach and learn these skills will be vastly different.

What is Object Based Computing?
Object based computing is a term brought to my attention by David Stutz, a NeXT Systems Engineer from the Chicago region. In the context of this text we shall define object based computing as

"The process of creating and manipulating computer based abstractions of the world around us."

Although that may seem a rather general definition, object oriented programming techniques will guide us in this process. Object based computing deals not just with the creation of new objects but also the re-use of and integration of existing objects to solve new problems. Object based computing will allow systems integrators to quickly build custom applications to solve specialized problems. Object based computing will end our current era of monolithic application based computing where we rely on a single program to solve our problems. Instead of searching for large complex turn-key existing software to solve our specific problems we will be able to create new applications to solve a variety of problems.

Benefits of Object Based Computing
The users of the objects we create will not care as much about the techniques we use to develop the objects as much as the benefits such objects bring them. Object based computing systems have the following benefits:

Customization.
A typical application program which runs in an object based computing environment is actually a collection of cooperating objects. These objects are either supplied by the computer manufacturer, third party software developers or could be in the public domain. If a user's needs do not match the application being run the user has the opportunity to edit the application to add or delete objects which conform to the user's needs. Other features of object based computing systems allow you to add and override characteristics of objects you wish to change.

Integration.
In an object based computing environment all objects are tied together by a simple communication system called a message passing system. Messages are simple structures which contain information that is exchanged between all types of objects local to the system as well as remote objects. Messages are language independent so objects can be created in C, LISP, FORTRAN or assembly language. Messaging allows new features to be integrated into existing programs without changing the original program. This allows programs to be easily integrated and allows old programs to take advantage of new technology without a great deal of change. An example of this might be the process of integrating voice recognition into an existing mail program. By adding a voice recognition object to your application you could simply tell it to redirect its output of recognized commands to the mail system using voice messages. You would not have to change the mail program.

Reusability.
On an object based computing platform, all objects are related in a hierarchy to the other objects in the system. This is known as Inheritance. For example, all objects which have dimensions on the screen are a sub-set of the "View" object.

OBJECT-BASED COMPUTING SUBSCRIPTIONS

Annual subscriptions are $28 for a hard copy version, but there will be an introductory rate until June 30th of $12. International subscriptions are 50% more. The email version is free to people who have hard copy subscriptions and request that the newsletter be sent by email in addition to or instead of the regular version. If you want to pay by check or money order (preferable) just send the necessary information payable to:

Information Technology Solutions, Inc. 400 W. Erie Suite 402 Chicago, IL 60610

If you want to pay by Visa or Mastercard you can send the necessary information by email, regular mail or fax. Send email to: gwkiv@its.com, or fax to (312)664-8409.

The following info is what I need (if you are sending the information by way of email please type the information without labels, it makes it easier to import it into Dataphile): Name, Address, Phone Number, Fax Number, NextMail, Asciimail (if no Nextmail) and Credit card type, expiration, and number(if paying by credit card).

All view objects inherit the program code which manipulates sizes of the objects on the screen and the order they are to be displayed. We re-use this object over and over for every new object we create which must be displayed on the screen. Creators of new objects no longer must start from scratch. By re-using existing code which has already been debugged and tested we find the productivity of system integrators rises dramatically.

Standardization.
In an object based environment there is often a rich set of objects or something called an application kit which has been created by the vendor. The application kit allows a core set of objects to be used by every developer in all of their programs. For example, all programs which require the user to select a new font style may be able to use standard objects which query the user for new font information. This is great for developers because they don't have to re-invent a new font object each time they wish to include font capabilities in their program. It is also very important for the users. Users need to know how to interact with only one font object. Once the users have learned how to change the font in one program they have instantly learned how to change the font in every program which uses the standard font object. This offers a very consistent environment across all programs you use. The time it takes to learn how to use a new program is dramatically less because the user expects the same menus, and the objects will respond exactly the same way they did in other programs you have used.

Sharing Objects.
One of the important differences between integrated Object Based Computing environments and traditional PC programming environments is that Object Based Computing systems have many of the objects and tools to manipulate the objects included as an integral component of the operating system. They are not added by third party software developers as an afterthought. Because of this, all users can assume a basic common denominator that the base objects are always there and can build new objects on top of them. This lowest common denominator assures a certain level of compatibility which allows the exchange of objects with others who have the same system. Compare this to the problems developers of PC software have with sharing objects. They first must pick a third party object oriented compiler and development system. Then they must purchase a library of objects which use the compiler. After they have added objects to the system they can only exchange those objects with users who have made the same identical choices. Each step quickly narrows the potential base of other people who can share the objects. So perhaps the greatest innovation Object Based Computing systems will give us is the unique ability to use and integrate a large public library of very high level objects.

Ease of Use.
Most, if not all, of the new Object Based Computing environments rely heavily on the use of graphical interfaces. A software company developing a library of objects can put a graphics front end onto the individual objects and encapsulate the routines of the object as a iconic button which a user can manipulate. These objects can then be integrated into tools that assemble the objects together in a point and click environment. Users view a subroutine library as a graphic "palette" of objects which are used to "paint" the user interface of an application similar to a painter using a pallet of colors to paint. Using a subroutine library is not a matter of going through the processes of editing, compiling, debugging, etc. It is the process of dragging icons and connecting them together. The potential user community of a subroutine library is not limited to programmers, but is now open to anyone who can point and click a mouse! The distinction between a programmer, a systems integrator and a user will begin to blur. This will allow a new level of extensibility to exist with computer programs. The user will be empowered with the ability to customize their environment to meet their own specific needs without the assistance of computer programmers or systems integrators.

Reliability.
Because objects communicate using carefully pre-defined and tested messages, they have a consistent, reliable and repeatable behavior. The creator of the objects can assume that the data types of all incoming messages is correct because the types of all message arguments are checked by the compiler. Because data that updates the state of an object enters through a clearly defined interface, the range checking can be performed before the state of the object is changed. Objects can avoid the problems of internal corruption when some new and unforeseen end case has occurred. Objects protect the data inside of them

by allowing the creator of the objects to automatically be a gate-keeper.

Extensibility.
It is often trivial to extend objects and add new features as well as modify existing features without changes in the objects we started with. The principal way of doing this is to extend and override messages that are directed to existing objects. Inheritance and delegation allow the creator to customize how messages are routed through objects. Proprietary objects can also be extended and integrated without the user having access to the original developer's source code.

Leveraging Powerful Servers.
One of the side benefits of multi-tasking environments is the ability to have a large group of powerful servers available in the background while your main program is running. Servers provide what their name implies: a group of specialized services you don't want to reproduce in your own objects. Some examples of servers are SQL database servers (which take in database queries and return reports), display servers (which take in draw commands and return graphic images), and computational servers (which do mathematical calculations). The essential observation is that because of the messaging architecture all objects we create can be servers. They can respond to informational requests and provide data. This is also known as the client-server model. Object Based Computing platforms which have integrated networking can easily extend this model to servers running on other processors as well as over local and wide area networks.

Network Integration.
When personal computers first became popular it was rare to find them integrated into a company wide network. Recently demands are being placed upon desk-top computers to quickly and transparently access large amounts of information within a company and on wide area networks such as the Internet. By using an object based computing environment with integrated networking you can take advantage of distributed computing very quickly. Transparent networking means every object can access every byte of information on every mass storage device on a network. This means sharing is easy and local users are far less likely to spend time duplicating information. In the end, productivity will rise and costs will fall.

From Procedural Programming to Object Oriented Programming
I am assuming that most readers have been exposed to procedural programming before they have access to Object Based Computing. Until systems as powerful as the NeXT computer reach the hands of the K-12 students, this will not change. I have found that there are several pitfalls common for those who grew up with only a procedural programming perspective.

Object oriented programming using toolkits is very easy. It involves actions such as selecting objects, dragging icons, and creating connections by drawing lines. These steps can be mastered by people with no previous programming background. In contrast, creating new objects or extending the objects requires rethinking many of the traditional procedural programming assumptions. It can still be mastered by people with a minimum of programming background, but the way we create these new structures is radically different than simply learning the syntax of another procedural programming language.

When I was a college student I learned Pascal. My first real job required that I learn C. It took me just a few days to learn the syntax differences between Pascal and C. But going from C to Objective C was a great deal more difficult, and not because Objective C has a great deal of new syntax to learn. In fact, Objective C only adds a handful of new language constructs. The real difference is how object oriented programming forces you to change the way you think. It took several months of studying textbooks and reading other people's programs before I understood the power of the concepts. But once I did, my mind caught fire with a new understanding. I felt I had a new powers. I could create programs with far greater complexity in a much shorter time. I hope that the same feeling of power also comes to the reader of this book. I have worked hard to make this happen for you in hours, not months.

Top-Down Design
One of the hardest techniques to teach is the way an experienced object oriented programmer partitions a problem into manageable pieces. Researchers such as the great psychologist Jean Piaget[1] have showed that most people learn new concepts in two ways: by comparing them to existing concepts and by decomposing

1. *Principals of General Psychology*. John Wiley and Sons, Inc., 1980. p. 292.

these new concepts into other structures. Piaget called these processes assimilation and accommodation. He felt that they are fundamental to the way we learn new concepts. Seymour Papert also showed that we can understand the way students learn procedural programming languages such as LOGO by using these models.[1] Papert quotes George Polya[2] who urges that whenever approached with a problem we ask two questions: Can this problem be subdivided into simpler problems? Can this problem be related to a problem I already know how to solve? These are also the principal design techniques that we use when we are faced with a task of object creation.

Suppose your task was to create a program that modeled the functions of a plant.The experienced object oriented programmer would take a look at the plant and partition it into structures such as leaves, the stem, and flowers. These structures would then be analyzed further until the problems could be represented by other objects or by lower level data structures. In contrast, many other design methods start by using these lower level structures and keep assembling them until they start simulating the higher level structures. They have a collection of data structures and a collection of algorithms and they build programs by combining them in whatever way best suits their problem. The problem is that it can take a great deal of effort to put the lower level pieces together before you see the high level structures. Later, if your high level structures don't fit the problem you often need to start over from scratch. Object based computing allows you to do a rapid prototype first, and only when the high level structures have been validated do you need to implement the low level structures. This later step is known as the code polishing phase.

All of this theory is perhaps interesting but won't mean much until you run into these problems yourself.

Next month's article will discuss tools for building applications in an Object-Based Computing development environment.

1. Seymour Papert. *Mindstorms: Children, Computers and Powerful Ideas*. Basic Books, 1980. This book is highly recommended for anyone teaching object based computing.
2. G. Polya. *How to Solve It*. Doubleday-Anchor, Garden City, N.Y., 1954.

Linguistics of Object-Oriented Programming and NeXT Objective-C Protocols

by Lionel P. ABOULKHEIR, Integrity Solutions, Inc.

As you become familiar with the concept of Objects and messaging, you begin to build more and more sophisticated objects. This brings you to the most difficult part of every elaborate system of communication, its linguistics.

Because the Objective-C messaging metaphor has brought a new dimension to programming languages, it has come closer to the general concept of a real language. Objective-C has added new levels of complexity to programming, even as it simplifies it. Just like a real language, Objective-C must be expanded when it becomes difficult to communicate a new concept using the existing linguistic structures. For example, in the NeXTSTEP 3.0 implementation of Objective-C, the addition of protocols is one of the features that was needed to make programs that are more powerful than those that have been made in the past.

The art of using the object-based paradigm, as opposed to the technique of object-oriented programming, pertains to the language: the messages, the syntax, the words... You can think of objects as entities which you can send memos to (messages), and then they can reply with a memo of their own.

However, just sending memos back and forth is not true communication. Problems are solved much more easily when people have a frame of reference to work within and more complex ways of communicating.

In real world discussions about problems to be solved, such as a mathematical problem, you must first provide a context, and a hypothesis, and follow some procedure along which intermediary results would be proved; eventually, it would lead to some conclusion, or message, which could be the starting point for a new larger procedure. It is the same kind of procedure you would use if you want to invite your neighbor to a party. First, you would have to contact him. Then he would probably have to talk to his wife, maybe arrange his schedule, and make any necessary steps to give you an answer at some point. If you want to invite all of your neighbors, you will have to do the same thing with each one of them.

In other words, goals which are dependent on many other actions make it very difficult, if not impossible, to deliver the necessary information within a single message. You will have to follow a certain procedure, or protocol, to get a single reply: you cannot get a reply from your neighbor, until he consults with his wife; nor can you start your party if nobody knows about it.

The same idea applies to the language you use for the objects you create. An elaborate object would have to know some sort of information before it can reply to a complex message. And when a number of objects cooperate, they will have to get this information in a certain order. This is what brought about the NeXT Objective-C Protocol in the 3.0 release. As stated in NeXT compiler 3.0 Release Notes:

(see the release notes for more technical information)

> *Protocols allow you to organize related method into groups that form high-level behaviors. This gives library builders a tool to identify sets of standard protocols, independent of the class hierarchy. Protocols provide*

language support for the reuse of design (i.e. interface), whereas classes support the reuse of code (i.e. implementation). Well designed protocols can help users of an application framework, when learning or designing new classes.

The necessity for such an extension probably arose from the DBKit design. How could one send a complex SQL expression using object orientation? SQL is itself a language, and as such contains many possible messages. If a set of objects are used to represent this language, they will have to cooperate in a certain fashion: To message a Select statement one has to provide the name of the tables it involves (Entities in DBKit), the name of the columns that are to be selected (Attributes) and the where clause, which may include joins (DBProperties). Before the database object (a DBDatabase) can reply to a select message (the selectData: method which is never sent directly), the internal state of all these objects has to be set (by sending them proper messages). Without protocols this sort of application would be a almost impossible.

Because of Objective-C's features, such as the newly introduced protocols, dynamic binding, and Objective-C's use of distributed objects, NeXT is becoming the leader in this kind of communication. Just as in the real world, as the lines of communication are improved, the end result is improved as well; and protocols are one way NeXT has created to make objects communicate with each other in more sophisticated ways, without making objects any more complex to create.

Object Oriented Programming: The Probable Person Period
Dan McCreary: Integrity Solutions

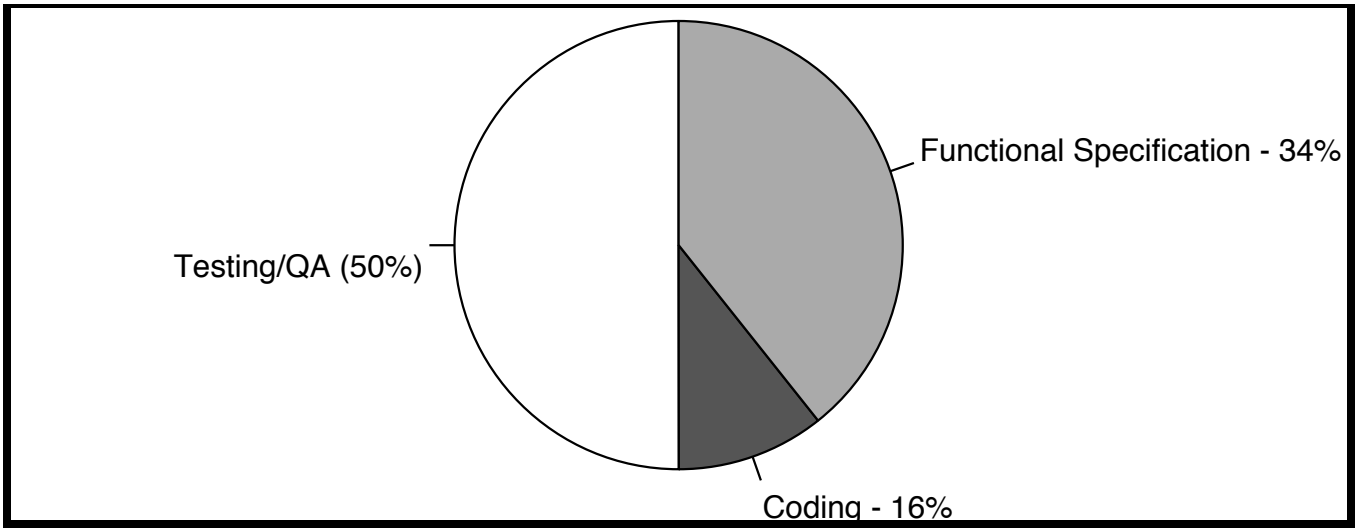
In his book The Mythical Man Month, Frederick P. Brooks, Jr. gives his description of the percentages of resources devoted to various aspects of the development of large software projects. From this analysis we can create the graph seen at the bottom of this page.

He further breaks the testing and quality assurance into sub sections of component test and system testing each of which takes about 25% of the total project resources. Some of us who have never built production systems are often shocked by the fact that only about 1/6th of the total time is dedicated to the programming part of a project. Some of us who have just finished getting final acceptance on software projects wonder why we didn't learn from our earlier mistakes that testing really takes at least three times as long as the coding.

Brook's book was published in 1975. Most of his examples are taken from IBM System 370 mainframe operating system software written is assembly language. The rules of the game have changed a lot since then. This article will look at how Object oriented programming has changed the makeup of resource allocation for large software projects.

First lets look at some of the claims made by the object oriented software vendors and see how these changes affect the total software schedule. EDS did a study a while back showing a 14:1 productivity gain by using an object oriented system over traditional systems. We have often heard our NeXT sales rep. talk about how NeXTSTEP programmers are 10 times more productive. Lets assume for a moment that their claims are right.

If we take the graph above and cut the coding time down by a factor of 10 where does that leave us? Let's assume a 100 hour project before our team learned the principals of object oriented programming. After sending our team to object oriented programming school and allowing them to spend a year getting familiar with the NeXTSTEP AppKit (you can't re-use what you don't know exists) our coding task just went from 16 hours down to 1.6 hours and the total project time went from 100 hours down to 85.6 hours, a 14% drop in the total project time. Probably not enough savings to convince the MIS director to replace all their IBM-PC systems with



Resource Allocation for Software Projects

NeXT systems. But the story is not so simple. Lets see how NeXTSTEP also affects some other aspects of the total schedule.

Functional Specification

One of the first productivity gains we notice is the way our staff uses Interface Builder to help us create functional specifications. We have developed a set of "mirage building tools" that allow us to quickly build a program that looks to an untrained user like the program is actually running off a real database. It is in fact just using ASCII files that we modify with any ASCII test editor. But, to the user, they can actually sit down and see how the user interface will function when they click on a button, browser, menu or window.

Interface Builder and our internally built mirage tools allow us to quickly build and modify user interfaces based on our customers needs. In a week or two we can often go through dozens of mock ups and ask the end users their preferences. When we are done we have a program that serves as our functional specification. After we design and implement our database and hook up the user interface the user still gets the same user interface only now it is actually doing what they want. No surprises. They get exactly what they asked for. Now that doesn't mean that end users don't change their mind. They always do. But with Interface Builder these changes are easy to make and the user can often see their requested changes the next day.

This method must be contrasted to the traditional method discussed by Brooks where very large functional specifications are meticulously written and constantly reviewed before the first line of code is written. This methodology was critical because it was so hard to change a program once it was created. By using object oriented design principals these changes are not disasters. The rigorous functional specification rules enforced by most of the large accounts firms that also do custom software don't apply to systems built with NeXTSTEP. Our experience has been that these firms spend more time in contract negotiations with our customers then we spend writing the programs. When mission critical applications need to be written, customers want solutions that they know will solve their problems, not a 100 page contract that they need to review.

In general I have found that by using the tools provided with NeXTSTEP, like interface builder, and by adding our own mirage builder tools we have reduced the time for functional specification by 60%.

So the 34 hours we originally spent on functional specification is now down to 20 hours.

The ability to rapidly re-arrange user interfaces does have one draw back. If the end users know how easy it is to change user interfaces they invariably ask us to do it and get upset when we want to bill them additional fees after the user interface has been finalized. Our solution is to ask users to initial each page of functions specification and set their expectations up front. Because we can change a user interface easily does not mean that the underlying database structures will not need to be changed when a user changes a functional specification..

Testing and Quality Assurance

One of the principal benefits of object oriented programming is enhanced reliability. By clearly controlling the access to state variables of your objects by the methods the object designer provides, the burden of correctness is moved from the shoulders of the user to the shoulders of the person who created the object. Fortunately for us, with more then 50 pre-tested classes in the NeXT AppKit, most of our testing job is done for us before we write our first line of NeXTSTEP code. Our job as programmers is to re-use as much of pre-tested code as possible. By minimizing the number of new lines of code we create we minimize the amount of testing we have to do.

Besides reusing pre-tested modules we also are forced to pay a great deal of attention to the interfaces between our objects by the standards enforced by good object oriented design methodologies. Many of the complex problems that used to come up in system test are the results of unexpected interactions between objects that have not been tested together previously under a new data set. By rigorously defining the methods used to change the instance variables of each object we control these interactions and can easily debug the side effects as they happen.

My experience with unit testing (testing of the individual objects) is that it is usually done in half the time and that system testing can be done in one third the time. This means that our 50 hours dedicated to testing and quality assurance is now down to about 21 hours, bringing the total time for the project from 100 hours down to 20+1.6+21=42.6 hours. This suggests that projects done with NeXTSTEP can be done in 42% of the time that they might take using other systems.

The savings can be increased even further by having a very clear functional specification to start. All of our work porting existing applications fit into this category.

Code Maintenance Costs

This analysis does not consider the fantastic cost savings we have found due to dramatically lower maintenance costs. We have found over and over again that new programmers can be assigned the tasks of fixing bugs or adding new features to existing programs with very little training. We have found that bugs are much easier to isolate because all the code in an objects file only deals with maintaining the state of its instance variables. Much of our code is inherently self documenting. It is because of the inherent structure of the NeXTSTEP software architecture that mission critical applications developed using the NeXTSTEP environment are much easier to maintain and extend.

Summary: Object oriented design and programming techniques can be applied to all phases of a software project. From needs analysis, functional specification, coding, unit testing and systems test, all stages of a project can be dramatically improved not just in schedules but in the better end results: the program being written actually solves the problems for the end users. Once object oriented design principals, tools and languages are in place the use of a large pre-tested class library can dramatically cut development time and reduce maintenance costs.

objectCatalog

Companies currently marketing programming tools and objects for the NeXT Developer:

Impact Software
210 Lake Street
Ithaca, NY 14850

(607) 277-8623

email: impact@impact.shamen.com

Objective Technologies
Suite 1502
7 Dey Street
New York, NY10007

(212) 227-6767

email: lg@object.com

RDR, Inc.
Suite 350
10600 Arrowhead Dr.
Fairfax VA 22030

(703) 591-8713

email: info@rdr.com

Threaded Technologies
339 Wiltsee Avenue
Loveland OH 45140

(513) 677-2106

email: !uunet!tti!dennisg

We would be happy to list your company -- please let us know how to list your company and what objects are available from your company. Filling out the object submission form on the next page will also help us to provide a complete list of resources for the object based NeXT developer.

In future issues an important part of this publication will be a descriptive list of commercial and "shareware" objects available to the NeXT Developer community.

Your participation in creating this marketplace of ideas is essential: we'd like to make it easier ---

TEMPLATE FOR CATALOG SUBMISSIONS

In order to make your lives a little easier we will be using the same template as NeXT for submissions to our objectCatalog. The information can be mailed, faxed or emailed to gwkiv@its.com

Object Category

Product Name

Product description here (up to 100 words)

Company Name or University

Street Address

City, State, Zip code, Country

Phone:

E-Mail address:

Pricing Information (including educational discount, if applicable):

Source code available:

Support available:

On-line Help (if a UI object):

Documentation Availability:

Localization Support:

Availability:

MAIL ADDRESS:

Geordie Korper
Object-Based Computing
400 West Erie, Suite 402
Chicago, IL 60610

FAX: 312.664.8409

EMAIL: gwkiv@its.com

Please use one of these category names for consistency and clarity:
UI Object
Information Display Object
Analytical Tool (includes objects to be used with Mathematica)
Networking & Telecommunications Object
Device Driver & Communications Object
Multimedia Object
Document & Publishing Object
Database Object
Financial Object
Tool & Utilities -- General
Business Graphic
Digitized Sound
Graphic
3D & Rendering Tool
Voice & Speech Object
Telephony & ISDN Object
Text & Language Tool
Scripting & Macro Object
Indexing & Retrieval Object
Workflow Object

Deadline for object submissions:

June 15, 1992

ENCLOSED:

The Premier Issue of *Object-Based Computing* -- A resource for the NeXT Programmer

ARTICLES:

Object-Based Computing (Part 1) *by Dan McCreary*

Linguistics of Object-Oriented Programming *by Lionel Aboulkheir*

Object Oriented Programming: The Probable Person Period *by Dan McCreary*

nextIssue
Object-Based Computing (pt. 2)
NeXT/OOP: A Case Study
usernamed: A Network Daemon
Flexible Object Design

Object-Based Computing is published monthly by Information Technology Solutions, Inc.

Information Technology Solutions, Inc. - -- 400 West Erie, Suite 402 - -- Chicago, IL 60610 - -- (800) 394-4487 - -- gwkiv@its.com

Object - Based Computing

400 WEST ERIE, SUITE 402
CHICAGO, ILLINOIS 60610