Chapter 1
Introduction

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
On the Mark 1, one of the first computers to use a stored program

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 the information based economy we find that people who use computers to their advantage will be more competitive.

What do we mean when we talk about the information age? Table 1 is an outline of some of the ages of man.

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

There are two interesting facts 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 their lives extracting the existing information from our written heritage. One of the most value component of our societies 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.

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 create 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 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 take 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 them into real world problems. The other 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.

Object Based Computing is Creating Abstractions

So what does this have to do with object based computing? Object based computing is the process of create 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.

Decade	Method	Objects	Number of 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

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

1 Presentation by Ed Barbonie, Summer 1989

will think of procedural programming in the 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 tools 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 the techniques in this book 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 the problem. Finding the correct object to fit the problem will be much like going to the library to check out a book. We will be able to search large database listing the features and connections to objects. We might even be able to rent objects that will be able 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 base 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 with just with the creation of new objects but also the re-use of and integration of existing objects to solve new problems. This book will guide you in the use of current programs that manipulate these objects as well as teach you to create your own objects. 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 solves our specific problem we will be able to create new applications to solve a variety of problems.

Benefits of Object Based Computing

This book covers the techniques used in object based computing systems. The users of the objects we create will not care as much about the techniques we use to develop them as much as benefits it brings them. Object based computing systems have the following benefits:

Customization. A typical application program which runs in a object based computing environment is actually a collection of cooperating objects. These objects are 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 both 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 together 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 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. All view objects inherit the program code which manipulates size of the objects on the screen and what order they are to be displayed in. 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 de-bugged 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 everyone 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 only 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 PCs is the Object Based Computing systems have many of the objects and the 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 with the problems developers of PCs 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 to share objects with. 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 graphical "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 with. 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 extendibility to exist with computer programs. The user will be empowered with the ability to customize their environment to their own specific needs without the assistance of computer programmers or system 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 check 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 gatekeeper.

Extensibility. It is often trivial to extend objects and add new features as well as modify existing features without changing 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 to have access to the original developers 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 tool-kits 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 re-thinking many of your 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 then 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 lot 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 problems into manageable pieces. Researchers such as the great psychologist Jean Piaget[2] have showed that most people learn new concepts in two ways: by comparing them to existing concepts and by decomposing these new concepts into other structures. Piaget called these processes assimilation and accommodation. He felt that they are the 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[3] Papert quotes George Polya[4] 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 technique that we use when we are faced with a task of object creation.

---

[2] *Principals of General Psychology* (John Wiley and Sones, Inc, 1980) p. 292

[3] Seymour Papert, *Mindstorms: Children, Computers and Powerful Ideas* (Basic Books, 1980). This book in highly recommended for anyone teaching object based computing.

[4] G. Polya, *How to Solve It* (Garden City, N.Y.: Doubleday-Anchor, 1954)

Suppose your task was to create a program that modeled the functions of a plant (Figure 2).

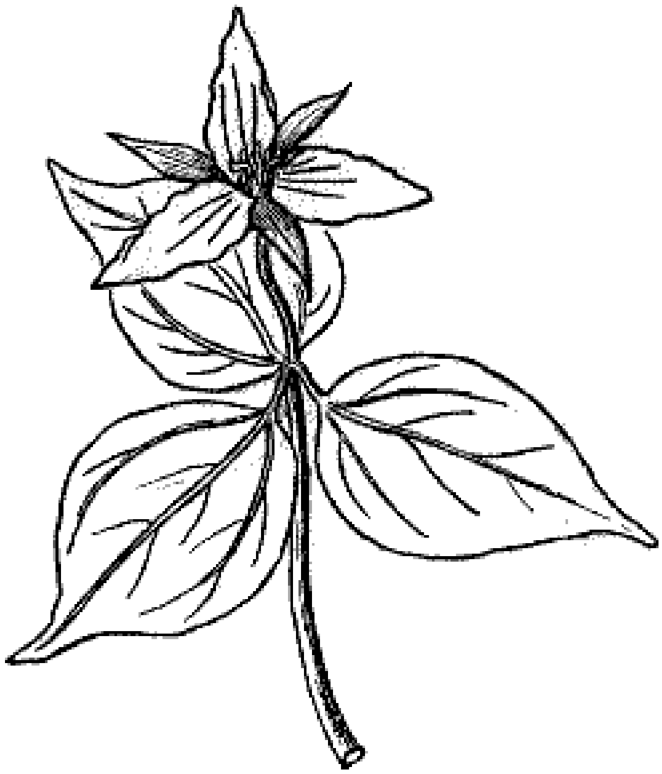

Figure 2: Decomposing The Structure of a Plant

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. The best way for us to do that is to get you started creating an object right away. The next chapter will introduce you to one of the tools for building applications and it will be followed by the actual creation of your first object.