Object-Oriented Programming and the Creator

Author: Doug Sharp
Subject: Chemistry
Date: 10/10/1998

Abstract: One of the most modern forms of computer programming is called "object-oriented" programming. This method of building computer applications has revolutionized the software industry. When the methodology is followed correctly, it saves time and money in development costs. The DNA code follows this same methodology, except that ordinary computer programming is one-dimensional; the code that makes up life is three-dimensional in scope. This three-dimensional shape has meaning, almost like a language, to solve the problems of replenishing, reproducing and constructing resources in the body. In addition, the lack of legacy code in living systems is evidence that there was one designer, one standardized genetic language, and this has the signature of the Creator God all over it.

The Evolution of the Computer

In the past thirty years, there has been a tremendous revolution in the computer industry. One of the first computers I worked with was an analog vacuum tube specialized trainer for the Navy that took up an entire building, and was programmed in FORTRAN II. In spite of its great size, this computer was very limited in its capability, and was programmed to simulate a war game. In the 1960's and 1970's, the basic method of computer input was punch cards, magnetic tape or punch tape. Most computer programs at that time were built around the concept of reading cards in a batch, processing them in a linear fashion, and producing a report, calculation, or some other result. Many of these early computer programs were written as one huge section of code. Because of their complexity, they were incomprehensible to most other programmers, and sometimes after a period of time, even the author couldn't figure out what he had written. This created a maintenance nightmare.

Many of the early computer programs were written in Assembler language, which was just about one step above machine code. The advantage of Assembler language was
speed and flexibility; the disadvantage is that it took a lot more code to do what you wanted and that resulted in a far more cryptic application. In 1971, the J.W. Knapp company in downtown Lansing had a RCA system that read cash register tapes and input the transactions into the computer. RCA sold their computer business to General Electric, and eventually GE sold their computer business to Honeywell. But many Honeywell systems needed to retain the architecture to support systems that originated with RCA.

I remember working for a manufacturing company who bought their brand new Univac that could actually run two programs at once in its 128k of memory. With 20 megabytes of disk we were sure we could run the entire business, especially since we could now store as many removable 10 megabyte disks off-line as we wanted. This computer still ran its applications in a batch update mode, using RPGII as the programming language. It stored the information on the disk in indexed sequential files, input through the computer terminal. During that time, similar small machines used that same concept: the IBM System 3, System 34, Burroughs B800 and DEC PDP-11. In addition, expensive mainframes comprised most of the computing power for large corporations. This included the IBM 360 and 370, Burroughs large systems, Control Data Corporation, Honeywell, NCR, and later on, the supercomputer Cray-1.

**Structured Programming**

Late in the 1970's and early in the 1980's, a new philosophy of program design called *structured programming* was created. This idea did away with the long, linear monolithic paragraph of code that jumped from one piece to another with GOTO statements. Instead, programmers sectioned off clearly defined pieces of code into short paragraphs and called them from the main module. Rather than using Assembler language, third generation languages such as COBOL, PASCAL and BASIC became popular. In addition, more sophisticated and faster computer hardware made relational databases possible, allowing storage of information on diskettes and hard disks. In the late 1970's and early 1980's it was quite expensive to do this, but the payoff was tremendous. No longer did you need to punch a card just to store 80 characters of information. Now you could store and retrieve tens of thousands of records and make them available at a computer terminal.

The relational database model made it possible to index files and build on-line computer applications to maintain them. Instead of keying records into a batch file, verifying them, sorting them, then running them sequentially against the file on disk to do updates, you could retrieve a record by index, look at it, make changes to it, and save it on the spot. But since the computer disk was still so expensive, everyone still needed to rely upon printouts to see the status of their information. It would be a long time before computers were inexpensive enough to be able to afford to have information at everyone's fingertips.

**The Personal Computer**

In 1976, on board ship in the Navy, our Combat Information Center got one of the first personal computers ever built. It was a Tektronix 4051 "programmable calculator" that cost over $7,000, but it had 32k of memory, a cartridge tape to store its programs, and the
BASIC programming language. Soon after, the Altair became a popular machine for labs, and Apple, Radio Shack, Commodore, Texas Instruments and even Timex introduced small computers that were affordable for many families. An obscure company called Smoke Signal Broadcasting introduced the one of the first 32-bit PC’s around 1979. Then, in 1981, IBM introduced its first personal computer and gave Microsoft a gift in allowing it to develop an open operating system. VisiCalc was the first spreadsheet, WordStar was the most popular word processor, and Dbase was to become the most popular database on these early computers. Die-hards like me pooh-poohed these puny machines, preferring to work with more powerful minicomputers and mainframes.

Eventually, though, these "toys" that we thought didn't have enough power to be useful for more than entertainment began to take over the corporate marketplace. Soon it became a race between the proprietary Apple computer and the IBM clones, which really became a Microsoft market. Today the Microsoft world has crowded out the minicomputer market, and is beginning to encroach on the massive mainframe arena.

**Object Oriented Programming**

With the introduction of graphical operating systems such as Windows, a new concept of programming has emerged. Programmers now design applications by putting together different pieces of already-written code that has been tested before, and each piece of code is called an "object." Objects can have properties, such as color, shape, size, and data type. An example would be a box on the screen that might contain a dollar amount. You could resize the text box, change its color, change the way it displays the dollar figure or formats it. You could then perform operations on that dollar amount, check for an "event" such as a click of the mouse that might trigger such an operation, or move it around on the screen. Because of the underlying operating system environment behind it, you can do all of this by setting properties rather than writing a lot of code. Not only that, you can create an application with an underlying database behind it, using all pre-programmed templates based upon the structure of the database.

In the old assembler language days, such a system would have been impractical if not impossible. Every machine language step would have had to be accounted for in the assembler program, and the size and complexity would be horrendous.

But much of that complexity is already built into the operating system of the computer that runs these "object-oriented" programs. The reason many of the functions that used to take months to program now seem easy and can be done within days, even hours, is that everything has been standardized and made to look simple by hiding the complexity in behind-the-scenes code. In addition, the economics of owning and operating computer systems have been revolutionized, due to hardware costs that are plummeting. It is commonplace for an average computer buyer in 2002 to have one gigabyte of RAM and 80 gigabytes of disk on a machine that costs 1/40 of what our poor Univac system did back in 1978. A life span of an average computer before it becomes obsolete is three years. This trend shows no sign of changing.
Evolution of Computer Compared to Evolution of Life

Now what I just described to you above might be called the "evolution" of the modern computer system. But there are differences between what I just described to you and the supposed "evolution" of life. The first major difference is that it is clear that intelligent designers built these computers. We know who these people were, and we have documented how it started off simple and became more complex. The next difference: as the computers became more complex, they became smaller, and the complexity became microminiaturized. But a far greater complexity can be found at the molecular level of life. Computer programming done by human beings is still in a linear fashion, but the genetic code of life is three-dimensional, with meaning and language not only in the sequence of characters, but in their orientation in space.

Next we notice that as the power of computing got greater, reusable components were created and became standardized. This is certainly true in life, but is a whole lot more efficient than Bill Gates ever could dream. Biologists call this homology and believe this to be evidence for evolution. Instead it is powerful evidence that an intelligent designer, far more powerful and intelligent that man, assembled all of these reusable components for the production of life, and efficiently borrowed the same ideas and patterns for different purposes. This is a direct parallel to object-oriented programming, only taken to a far new dimension beyond our capability. Wings all have similar designs in bats, birds, bees, and flying reptiles, but evolutionists claim there is no relationship, and these homologous features evolved independently. But these features have too many similarities to have arisen independently and randomly. This favors intelligent design by a creator God.

No Legacy Code

One of the biggest problems facing computer technology is the rapid obsolescence of systems. In the current market, a year or even six months can make a tremendous difference as to the usefulness of a software package or system. It used to be said of IBM that dealing with them was like sleeping with an elephant--you may stay warm but you have to watch out when he rolls over. Now that is no longer true. It is now Microsoft who is the behemoth bedfellow.

Most vulnerable to obsolescence are the massive intertwined proprietary mainframe systems that people depend upon for their daily business. A simple problem like the Year 2000 glitch can cause repercussions that cascade from module to system to enterprise. Corporate offices are still populated with obsolete computers, mainly because office procedures are still dependent upon them. "Legacy code" refers to systems that have been around for a while that still are in use, but the inner workings of the code are no longer fresh in the minds of those who designed it. Though the new paradigms of software development were built on the mistakes and lessons of the past, remnants of these old systems, some even built in the early 1970's still exist in production today.
If we compare the evolution of the computer to evolution of life, though, we find a different situation. At the basic building block level we find the same efficient coding structure in the "simplest" forms of life as we do in human beings. There is no hierarchy of complexity at the molecular level. In fact, it is the same structured code throughout life. This is in vast contrast to computer systems where the complexity of languages changed drastically from year to year. How many new systems are now written in FORTRAN, Neat-3, Jovial, SNOBOL-4, RPGII, APL, PL1, or GW-BASIC? There are very few, because the operating environment for those languages is no longer being actively supported. How many computer systems are still around that use VisiCalc, WordStar, PC Write or Dbase II? Not many, I'll bet. But life's computer language exists in all organisms from the largest to the smallest, and it is all standardized, operating in exactly the same manner. There is no "learning" as time went on, no legacy code left as a remnant. My observation is that God got it right in the first place and started out with a three-dimensional object-oriented code for living systems that needed no upgrading.

**Mutations and Genetic Variation**

What about mutations, those coding mistakes left when a DNA molecule is damaged? Since these are normally repaired by the living system or otherwise weeded out, it shows the resiliency of the originally created genetic code. Mutations are often mistaken as evidence for evolution. But because the effect of mutations are deleterious, they are analogous to computer bugs, and the chance of them causing a new species is much like a typing mistake in a computer program giving rise to a new application relevant to the task at hand.

The lack of legacy code in living systems and the inability for mutations to explain new characteristics in living systems are two powerful pieces of evidence to bolster belief in a creator God.