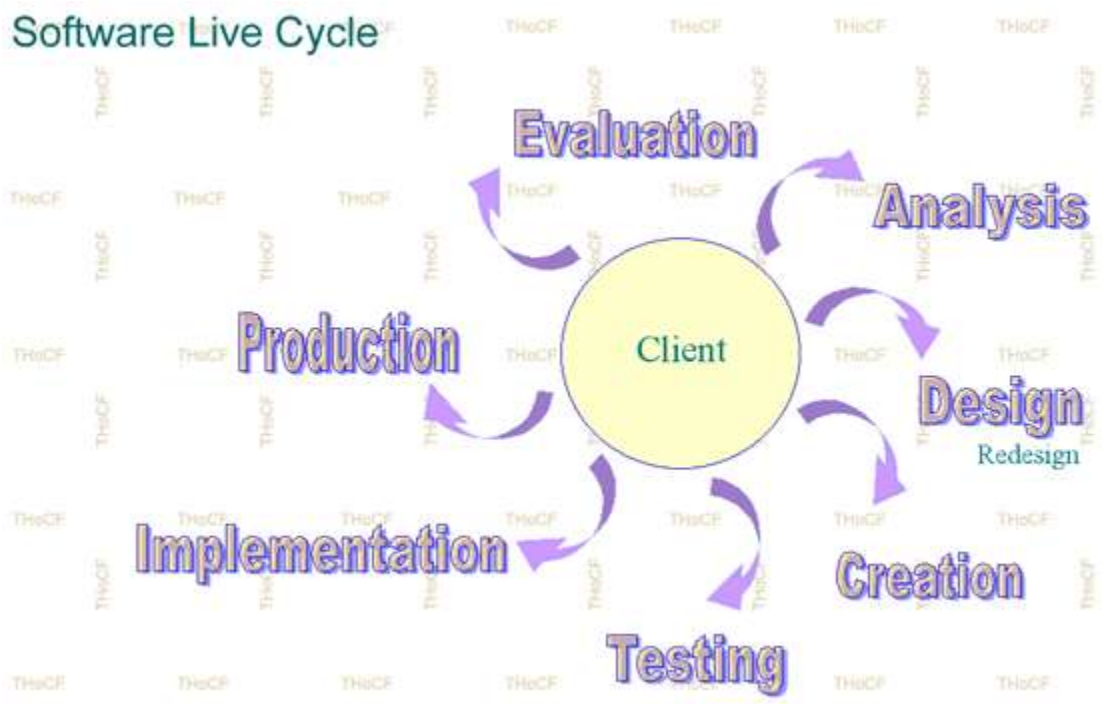
[*Note: This document has been modified from the original by the Saylor Foundation*]

Introduction to Software History
by Cornelis Robat, Editor



First Steps

This part will be different from the History of the computer, no chronological travel through software-land, but a collection of articles and assays on software.

Software has a long history and as far as the facts are known to us we will give them to you. When missing stories, data, or other information are shared to us they will be put on this site. If you have any comments of suggestions regarding this page or any other page please do not hesitate to contact us.

A simple question: "What is software?" A very simple answer is: Hardware you can touch, software you can't. But that is too simple indeed.

But when talking about software you talk about programming and programming languages. But about producing and selling the products made by programming (languages) as well.

There are over 300 different ("common") computer languages in existence, apart from the various dialects stemming from one of them. Most of them can be classified in definable groups, but others don't belong to anything. Some because they are rather new or the use of them was or is never wide spread and only used by a small specialized professionals or groups of scientists requiring these dialects. This is often the case with a specific language that was designed for just one purpose, e.g. telecommunication or supercomputing.
Some languages are even dead languages, some others are revived and expanded upon again, and there are ones that constantly rejuvenate. In the latter case a programmer is sometimes wondering whether he or she is not just upgrading to a newer version but instead learning a complete new language.

How It All Started

It shouldn't be a big surprise that the creation or software also went in large but distinguishable steps. Compared with hardware there were fewer developments that went parallel or overlapping. In rare cases developments were reinvented sometimes because the development or invention was not published, even prohibited to be made public (war, secrecy acts etc.) or became known at the same time and after (legal)discussions the "other" party won the honors.

The earliest practical form of programming was probably done by Jaquard (1804, France). He designed a loom that performed predefined tasks through feeding punched cards into a reading contraption. This new technology allowed carpets and tissues to be manufactured with lower skills

and even with fewer people. The little kid sitting under the loom changing rods and other things vanished. One single person could now handle a loom. That this met resistance from the weavers leaves no question. The same thing happened in England during the industrial revolution there. Even a movement came up called: Luddites (anti-technology or just concerned citizens fighting for their bread?)

This picture shows the manufacturing

of punched cards for looms

The technology of punched cards will later be adapted by (IBM's) Recording and Tabulating Company to process data.

The situation was still a one on one game: a problem needed to be solved thus a machine was built. (Pascal, Babbage, Scheultz & Son) And when some sort of instruction was needed a sequence was designed or written and transferred to either cards or mechanical aids such as wires, gears, shafts actuators etc.. To call that programming? Well, according to our definition yes it was.

First there was Ada Lovelace, writing a rudimentary program (1843) for the Analytical Machine, designed by Charles Babbage in 1827, but the

"**What was first: software or hardware**"

Frankly this is a matter of philosophy, or simpler: how you look at it.

Nonetheless this question is difficult to answer:

With the early computers "the idea" did not express

machine never came into operation.

Then there was George Boole (1815-1864), a British mathematician, who proved the relation between mathematics and logic with his algebra of logic (BOOLEAN algebra or binary logic) in 1847.

This meant a breakthrough for mathematics. Boole was the first to prove that logic is part of mathematics and not of philosophy.

A big step in thinking too.

But it will take **one hundred years** before this algebra of logic is put to work for computing.

John Von Neumann working at the Institute for Advanced Study developed in 1945 two important concepts that directly affected the path of computer programming languages:

The first concept became known as "shared-program technique" (7). This technique states that the actual computer hardware should be simple and not need to be hand-wired for each program. Instead, complex instructions should be used to control the simple hardware, allowing it to be reprogrammed much faster.(8)
The second concept was also extremely important to the development of programming languages. Von Neumann

itself neither in software nor in just hardware but was broadly interpreted: computing was problem based.
Remember that the first computers were designed for just one single task. One problem - one computer or machine or contraption.

The autonomous - something that could theoretically "run" by itself on any machine - software idea came only after these single task computers were already history.

That is why can be said:

The first computers that were built, represented in the same time the software as well as the hardware

More precise: one could build a machine to have it solve a problem automatically. By doing that you translated an idea into a mechanical expression that

called it "conditional control transfer"(7) (www.softlord.com). This idea gave rise to the notion of subroutines, or small blocks of code that could be jumped to in any order, instead of a single set of chronologically ordered steps for the computer to take. The second part of the idea stated that computer code should be able to branch based on logical statements such as IF (expression) THEN, and looped such as with a FOR statement. "Conditional control transfer" gave rise to the idea of "libraries," which are blocks of code that can be reused over and over.(8)

ran the built in problem-solving mechanism by itself.

So this question cannot be answered when you regard software as a non-integrated standalone idea or resource. In that case: hardware was first.

But for the hardware to run you needed an idea or problem to be translated into a mechanical expression. In other words software solidified into a machine. In this case software was first.

the first draft on the EDVAC

(the following paragraphs are cited from http://www.princeton.edu/~ferguson/adw/programming_languages.shtml)

It took Claude Shannon (1916-2001) who wrote a thesis (*A Mathematical Theory of Communication* in the *Bell System Technical Journal* -1948) on how binary logic could be used in computing to complete the software concept of modern computing.

In 1949, a few years after Von Neumann's work, the language Short Code appeared . It was the first computer language for electronic devices and it required the programmer to change its statements into 0's and 1's by hand. Still, it was the first step towards the complex languages of today. In 1951, Grace Hopper wrote the first compiler, A-0. A compiler is a program that turns

the language's statements into 0's and 1's for the computer to understand. This lead to faster programming, as the programmer no longer had to do the work by hand.(9)

In 1957, the first of the major languages appeared in the form of FORTRAN. Its name stands for FORmula TRANslating system. The language was designed at IBM for scientific computing. The components were very simple, and provided the programmer with low-level access to the computers innards. Today, this language would be considered restrictive as it only included IF, DO, and GOTO statements, but at the time, these commands were a big step forward. The basic types of data in use today got their start in FORTRAN, these included logical variables (TRUE or FALSE), and integer, real, and double-precision numbers.

Though FORTAN was good at handling numbers, it was not so good at handling input and output, which mattered most to business computing. Business computing started to take off in 1959, and because of this, COBOL was developed. It was designed from the ground up as the language for businessmen. Its only data types were numbers and strings of text. It also allowed for these to be grouped into arrays and records, so that data could be tracked and organized better. It is interesting to note that a COBOL program is built in a way similar to an essay, with four or five major sections that build into an elegant whole. COBOL statements also have a very English-like grammar, making it quite easy to learn. All of these features were designed to make it easier for the average business to learn and adopt it.

In 1958, John McCarthy of MIT created the LISt Processing (or LISP) language. It was designed for Artificial Intelligence (AI) research. Because it was designed for such a highly specialized

field, its syntax has rarely been seen before or since. The most obvious difference between this language and other languages is that the basic and only type of data is the list, denoted by a sequence of items enclosed by parentheses. LISP programs themselves are written as a set of lists, so that LISP has the unique ability to modify itself, and hence grow on its own. The LISP syntax was known as "Cambridge Polish," as it was very different from standard Boolean logic (Wexelblat, 177) :

x V y - Cambridge Polish, what was used to describe the LISP program
OR(x,y) - parenthesized prefix notation, what was used in the LISP program
x OR y - standard Boolean logic

LISP remains in use today because its highly specialized and abstract nature.

The Algol language was created by a committee for scientific use in 1958. It's major contribution is being the root of the tree that has led to such languages as Pascal, C, C++, and Java. It was also the first language with a formal grammar, known as Backus-Naar Form or BNF (*McGraw-Hill Encyclopedia of Science and Technology*, 454). Though Algol implemented some novel concepts, such as recursive calling of functions, the next version of the language, Algol 68, became bloated and difficult to use (www.byte.com). This lead to the adoption of smaller and more compact languages, such as Pascal.

Pascal was begun in 1968 by Niklaus Wirth. Its development was mainly out of necessity for a good teaching tool. In the beginning, the language designers had no hopes for it to enjoy widespread adoption. Instead, they concentrated on developing good tools for teaching such as a debugger and editing system and support for

common early microprocessor machines which were in use in teaching institutions.

Pascal was designed in a very orderly approach, it combined many of the best features of the languages in use at the time, COBOL, FORTRAN, and ALGOL. While doing so, many of the irregularities and oddball statements of these languages were cleaned up, which helped it gain users (Bergin, 100-101). The combination of features, input/output *and* solid mathematical features, made it a highly successful language. Pascal also improved the "pointer" data type, a very powerful feature of any language that implements it. It also added a CASE statement, that allowed instructions to to branch like a tree in such a manner:

CASE *expression* OF
    possible-expression-value-1:
        statements to execute...
    possible-expression-value-2:
        statements to execute...
END

Pascal also helped the development of dynamic variables, which could be created while a program was being run, through the NEW and DISPOSE commands. However, Pascal did not implement dynamic arrays, or groups of variables, which proved to be needed and led to its downfall (Bergin, 101-102). Wirth later created a successor to Pascal, Modula-2, but by the time it appeared, C was gaining popularity and users at a rapid pace.

C was developed in 1972 by Dennis Ritchie while working at Bell Labs in New Jersey. The transition in usage from the first major languages to the major languages of today occurred with the transition between Pascal and C. Its direct ancestors are B and

BCPL, but its similarities to Pascal are quite obvious. All of the features of Pascal, including the new ones such as the CASE statement are available in C. C uses pointers extensively and was built to be fast and powerful at the expense of being hard to read. But because it fixed most of the mistakes Pascal had, it won over former-Pascal users quite rapidly.
Ritchie developed C for the new Unix system being created at the same time. Because of this, C and Unix go hand in hand. Unix gives C such advanced features as dynamic variables, multitasking, interrupt handling, forking, and strong, low-level, input-output. Because of this, C is very commonly used to program operating systems such as Unix, Windows, the MacOS, and Linux.

In the late 1970's and early 1980's, a new programing method was being developed. It was known as Object Oriented Programming, or OOP. Objects are pieces of data that can be packaged and manipulated by the programmer. Bjarne Stroustroup liked this method and developed extensions to C known as "C With Classes." This set of extensions developed into the full-featured language C++, which was released in 1983.

C++ was designed to organize the raw power of C using OOP, but maintain the speed of C and be able to run on many different types of computers. C++ is most often used in simulations, such as games. C++ provides an elegant way to track and manipulate hundreds of instances of people in elevators, or armies filled with different types of soldiers. It is the language of choice in today's AP Computer Science courses.

In the early 1990's, interactive TV was the technology of the future. Sun Microsystems decided that interactive TV needed a

special, portable (can run on many types of machines), language. This language eventually became Java. In 1994, the Java project team changed their focus to the web, which was becoming "the cool thing" after interactive TV failed. The next year, Netscape licensed Java for use in their internet browser, Navigator. At this point, Java became the language of the future and several companies announced applications which would be written in Java, none of which came into use.

Though Java has very lofty goals and is a text-book example of a good language, it may be the "language that wasn't". It has serious optimization problems, meaning that programs written in it run very slowly. And Sun has hurt Java's acceptance by engaging in political battles over it with Microsoft. But Java may wind up as the instructional language of tomorrow as it is truly object-oriented and implements advanced techniques such as true portability of code and garbage collection.

Visual Basic is often taught as a first programming language today as it is based on the BASIC language developed in 1964 by John Kemeny and Thomas Kurtz. BASIC is a very limited language and was designed for non-computer science people. Statements are chiefly run sequentially, but program control can change based on IF..THEN, and GOSUB statements which execute a certain block of code and then return to the original point in the program's flow.

Microsoft has extended BASIC in its Visual Basic (VB) product. The heart of VB is the form, or blank window on which you drag and drop components such as menus, pictures, and slider bars. These items are known as "widgets." Widgets have properties (such as its color) and events (such as clicks and double-clicks) and are central to building any user interface today in any language. VB is

most often used today to create quick and simple interfaces to other Microsoft products such as Excel and Access without needing a lot of code, though it is possible to create full applications with it.

Perl has often been described as the "duct tape of the Internet," because it is most often used as the engine for a web interface or in scripts that modify configuration files. It has very strong text matching functions which make it ideal for these tasks. Perl was developed by Larry Wall in 1987 because the Unix sed and awk tools (used for text manipulation) were no longer strong enough to support his needs. Depending on whom you ask, Perl stands for Practical Extraction and Reporting Language or Pathologically Eclectic Rubbish Lister.

Programming languages have been under development for years and will remain so for many years to come. They got their start with a list of steps to wire a computer to perform a task. These steps eventually found their way into software and began to acquire newer and better features. The first major languages were characterized by the simple fact that they were intended for one purpose and one purpose only, while the languages of today are differentiated by the way they are programmed in, as they can be used for almost any purpose. And perhaps the languages of tomorrow will be more natural with the invention of quantum and biological computers.
<end quote>

Also the hardware needed to make jumps ahead like the following:

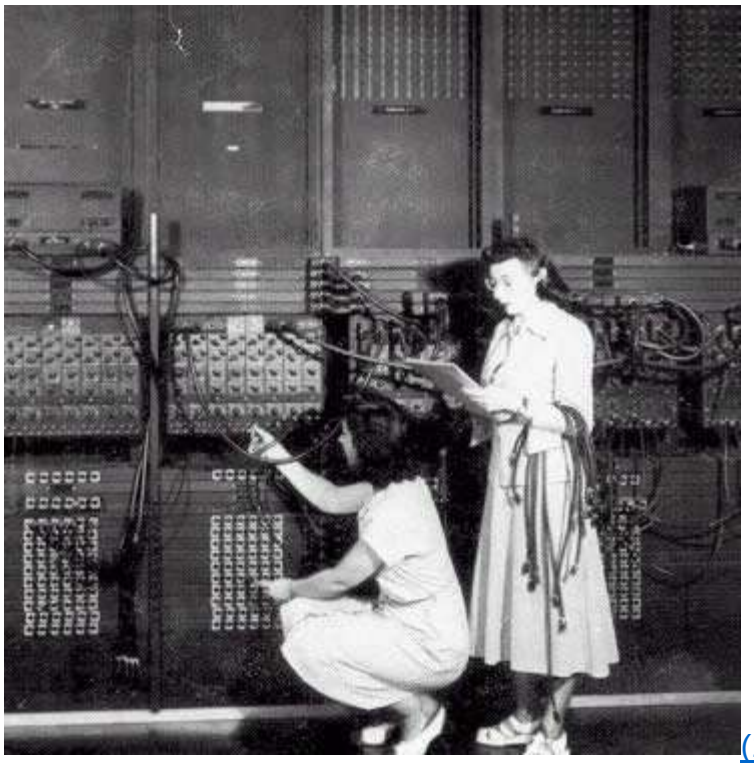
It took Zuse to create the first binary programmable computer,

relay based.
The Bomba, originally built by Polish engineers to crack the Enigma code, pushed the envelope again
The colossus built by people from Bletchley Park (near London, UK) for the same purpose
Atanasov and Berry designed the ABC computer, a binary computer, as Zuse's was, but now 100% electronic.
And not to forget the ENIAC by Eckert and Mauchly and a team made up of many others

Now things were in place to start off with the information age.

Enter the Information Age

In the beginning of the so called "Information Age" computers were programmed by "programming" direct instructions into it. This was done by setting switches or making connections to different logical units by wires (circuitry).



(1)

Two women wiring the right side of the ENIAC with a new program

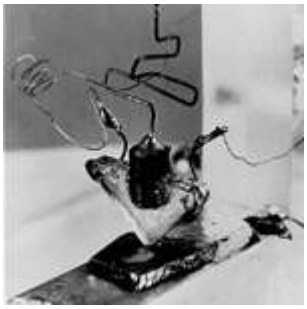
(US Army photo, from archives of the ARL Technical library, courtesy of Mike Muuss)

Programming like this was nothing else but rewiring these huge machines in order to use all the options, possibilities and calculations. Reprogramming always meant rewiring.

In that way calculations needed days of preparations, handling thousands of wires, resetting switches, plugs etc. (in the most extreme case that is). And the programmed calculation itself just took a few minutes. If the "programming" of wiring did not have a wrong connection, the word bug was not in used yet, for programming errors.
The coding panels very much looked like that a few telephone switchboards hustled together, and in fact many parts actually came from switch boards.

With the invention of vacuum tubes, along with many other inventions, much of the rewiring belonged to the past. The tubes replaced the slow machines based on relays.

The above picture displays an early electronic switch called a triode.



The first contact point transistor

When the transistor was invented this again replaced a technology: vacuum tubes.

When Shannon reinvented or better rediscovered the binary calculus in 1948 and indicated how that could be used for computing a revolution started. The race was on!

Programming the new tools (or toys?)

Programming the first binary computer was still not an easy task and much prone to mistakes. First programming was done by typing in 1's or 0's that were stored on different information carriers. Like paper tapes, punched hole cards, hydrogen delay lines (sound or electric pulse) and later magnetic drums and much later magnetic and optical discs.

By storing these 0's and 1's on a carrier (first used by Karl Suze's X1 in 1938) it was possible to have the computer read the data on any later time. But mistyping a single zero or one meant a disaster because all coding

(instructions) should absolutely be on the right place and in the right order in memory. This technology was called absolute addressing.

An example:

1010010101110101011

If this is a sequence for switches it means switch one on, switch two off etc. etc.

In simple language:

Panel	**1**	**function: enter house**
Switch 0	**1**	**open the door**
Switch 1	**1**	**put the lights on**
Switch 2	**0**	**close the door (please)**

In fact the protocols for programming the machines in this way looked very much like that.

In the early 50's programmers started to let the machines do a part of the job. This was called automatic coding and made live a lot easier for the early programmers.

Soon the next step was to have the program to select the proper memory address instead of using absolute addressing.

The next development was to combine groups of instruction into so called words and abbreviations were thought up called: opcodes (Hopper 1948)

Absolute addressing:

the programmer instructs the machine at what location of the memory (valve, relay, transistor) to store a value

Machine Language

Opcode works like a shorthand and represents as said a group of machine instructions. The opcode is translated by another program into zero's and one's, something a machine could translate into instructions.

But the relation is still one to one: one code to one single instruction. However very basically this is already a programming language. It was called: assembly language.

An example:

Label	Opcode	Register
CALC:	**STO**	**R1, HELP0**
	STO	**R2, HELP2**
	LD	**R3, HELP1**
	ADD	**R3, HELP2**
	LD	**R4, HELP1**
	SUB	**R4, HELP2**
	RSR	**SP, 0**
HELP1:	**DS**	**2**
HELP2:	**DS**	**2**

This piece of assembly code calculates the difference between two numbers.

Subroutines

Soon after developing machine languages and the first crude programming languages began to appear the danger of inextricable and thus unreadable coding became apparent. Later this messy programming was called: "spaghetti code".
One important step in unraveling or preventing spaghetti code was the development of subroutines. And it needed Maurice Wilkes, when realizing that "a good part of the remainder of his life was going to be spent in finding errors in ... programs", to develop the concept of subroutines in programs to create reusable modules. Together with Stanley Gill and David Wheeler he produced the first textbook on "The Preparation of Programs for an Electronic Digital Computer".(6)
The formalized concept of software development (not named so for another decade) had its beginning in 1951.

Below is an example of how subroutines would work.

Start of program the main "menu" first subroutine	Begin program; ***Main***; Printf ("Hello World"); DoSomethingElse() Printf ("Hello World"); (end of program) Function **DoSomethingElse**; Add two numbers;

	Return OK
back to the main menu second subroutine *with a parameter (contents of what to print)* back to procedure: main	Function ***Printf***(what_to_print) Open channel to printer interface; Initialize printer; Send "what_to_print" to printer; Send page feed to printer; Close printer interface; Return OK

This program would print "Hello World" twice on two different pages.

By re-using the **Printf** subroutine a possible error in this routine would show up only once. An enormous advantage when looking for errors. Off course the*Open, Initialize, Send,* and *Close* "commands" in this **Printf** function are also subroutines.

Fortran

The next big step in programming began when an IBM team under John W. Backus created FORTRAN - FORmula TRANslator 1952. It could only be used on their own machine, the: IBM 704. But later versions for other machines, and platforms were sold soon after. Until long past 1960 different CPU's required another kind instruction set to add a number, thus for each different machine a different compiler was needed. Typically the manual came years later in 1957!
Rewiring of machines to reprogram them now definitely belonged to the past!

THE FORTRAN
AUTOMATIC CODING SYSTEM
FOR THE
IBM 704 EDPM

Programming language

FORTRAN soon became called a programming language. So why calling a set of some predefined instructions a programming language?

Because some characteristics of a language are met:

- It must have a vocabulary - list of words
- It must have a grammar - or syntax
- It must be unique, both in words and in context

All the above criteria were easily met for this - strictly defined- set of computer instructions.

An example:

Let's presume communication with a computer can be accomplished. Then how would you tell it to add two numbers in simple terms?

human	computer
Add 2 and 2	
Show me the answer	print 2+2

Depending on what (dialect of) computer language you use it could look different:

human	computer
Add 2 and 2	answer := 2+2;
Show me the answer	printf ("%d\n", answer);

The above example is standard ANSI C.

And by the time when Cobol, Common Business Oriented Language, was published in 1960 by the Codasyl committee, (Hopper was a member) the term Computer Language was a fact.

In the meantime hardware developments raced literally ahead.

Already computers were connected to teletype machines to expedite the entry of programs. In the late 1960's the first video interfaces were connected. The network was invented. And floppy disks, hard drives etc. made live for programmers a lot easier.

As mentioned you could not simply run FORTRAN on any machine. It had to be rewritten for each particular machine if the type of processor was different. In in that early days ALL types were different. This did not promote the development of programming at all!

Enter "C"

C came into being in the years 1969-1973 and was developed by Dennis Richey and David Kerningham both working at the Bell laboratories.(3)

And the magic word was portability.

Parallel at the development of computer languages, Operating Systems (OS) were developed. These were needed because to create programs and having to write all machine specific instructions over and over again was a "waste of time" job.
So by introducing the OS 'principle' almost all input and output tasks were taken over by the OS.

Such as:

- writing data to and from memory,
- saving data on disks,
- writing data to screens and printers,
- starting tapes,
- refreshing memory,
- scheduling specific tasks
- etcetera, etcetera.

(More on operating systems in another chapter)

As the common computer languages had trouble to be translated from one machine to another the OS's had to take the same hurdle every time a new machine was developed.

The need ad pressure for a common portable language was enormous.

There were some unsuccessful projects aimed to solve this problem, like Multics, a joint venture of MIT, General Electric, and Bell Labs. And other developments at DOD's in different countries. But they all came either too late or became too complex to succeed.

But the demise of Multics inspired Dennis Ritchie and Brian Kernigham to develop C in 1972. This was and still is a very strict language that stayed close enough to the machine's internal logic and structure. If you were allowed to say that. This new language was reasonable well to read and understand by humans. And because of this combination the language is fast, compact and became very popular amongst system programmers and commercial software manufacturers.

With that language they also developed UNIX, a generic operating system.

The power of C was that the language had a small language base (vocabulary) but leaned heavily on what they called libraries. Libraries contain machine specific instructions to perform tasks, like the OS does. These libraries were the only parts that had to be redesigned for different machines, or processor families. But, and that was C's strength, the programming interface/language remained the same. Portability was born. Source code could be reused, only to be recompiled when it had to run on other machines.

A classic example of C, printing "Hello World" on your screen:

```
/* helloworld.c */

main()
{
printf('Hello
World\n");
}
```

In another chapter this routine will be shown to make the various differences in language visual.

Soon other language manufacturers sprang on the bandwagon and the software industry leaped ahead. The industry took off like a rocket!

Footnotes & References

1 http://www.library.upenn.edu/special/gallery/mauchly/jwm8b.html

2 picture c.robat

3 Dennis M. Ritchie Bell Labs/Lucent Technologies, Murray Hill, NJ 07974 USA

4 http://wuarchive.wustl.edu/doc/misc/lang-list.txt

5 This is mentioned in the 1978 ACM History of Programming Languages FORTRAN session.

6 from: ei.cs.vt.edu

7 www.softlord.com

8 http://www.princeton.edu/~ferguson/adw/programming_languages.shtml

9 www.byte.com