

**Lessons learned
on the topic**

“The Origins of Software”

By

Armin Hornung and Sandra Lemon

Definition

"There will be no software in this man's army!" (Eisenhower, 1947)

Just as this notable quote was probably not meant to be about the sense of software which we have today, the meaning of “software” was very vague in the beginning. So, the first thing to clarify when outlining the origins of software, and arguably the most difficult, is the actual definition of software. In the early days a definition could not be easily agreed upon, because the machines of that time were always for a distinct purpose. The machines incorporated both software and hardware at the same time and the line separating the two was just not obvious.

Today, the distinction is much more obvious. Software can generally be seen as the implementation of the task that the computer has to finish, working on the physical hardware, or put simply, *“software is everything that is not hardware”*. This commonly accepted definition still has its limitations. It implies that images, music, documents, data, etc. are all software. Conceptually, software may be better defined as the interface between the hardware and data, which the whole system processes and generates.

The early days

The first idea of software can be observed in 1804, with the loom of Jacquard. Paper cards with punched holes controlled the weaving of the pattern on the loom, enabling more complex patterns and faster production times. His machine created the first need for software and with it, the first negative reaction to programmable machines. The high-tech loom changed the weaving profession, effectively lowering the required skill set and limiting the number of people needed to operate. The loom demonstrated three aspects of software: logical structure, representation and the interaction with the physical device, a concept that would loosen throughout the evolution of software.

Jacquard’s most significant invention was perhaps that of the punch cards – they would remain in use for well over a hundred years. Some significant uses would

be Hollerith's tabulating machine for the U.S. Census, the UNIVAC and the machines built by IBM.

In 1837, Charles Babbage designed and planned to build the Analytical Engine, a programmable calculator. The Analytical Engine was supposed to be driven by programs on punch cards, enabling the machine to perform all operations possible to modern computers. The idea of programming the machine with the cards originally came from Ada Lovelace, the protégé of Babbage, who is often regarded as the world's first programmer. However, just like the Difference Engine, the Analytical Engine failed and was never built during Babbage's lifetime.

In those days, the software – mainly consisting of instructions by punch cards – was always specific to one kind of machine. Only many years later would the software evolve to become largely independent from the hardware.

Software in Theory

"A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal Turing Machine." (Turing, 1948)

In 1936 Alan Turing invented the theoretical design of a “Turing Machine”, proving that the Halting problem is unsolvable [6]. The Turing Machine was designed to function as a systematical working human, using only three operations: read from a tape, write to a tape, and move the read/write head. With only these operations, the Turing Machine can compute anything that a functioning computer is able to calculate, and the Church-Turing thesis [7] extends its capabilities by conjecturing that any function calculable in the common sense can be computed by a Turing Machine.

Similar to a Turing Machine, but operating in a simpler way, is a Finite State Automaton (FSM), a construct also widely used in theoretical computer science. Being more powerful, a Turing Machine can simulate a FSM, and the language acceptable by a FSM (*Regular Languages*) is a subset of the language acceptable by a Turing Machine (*Recursively Enumerable Languages*). In theory, a real computer is closer to a FSM because the amount of memory is limited. Practically, however, the number of possible states a computer can be in is so large, that regarding all theoretical questions such as the Halting Problem, it can be regarded as a Turing Machine [8].

The theoretical implications of a Turing Machine are vast, but the significant feature in this instance is the *software* that arose from the conception of a Turing Machine. A Turing Machine proposed that a mechanical machine would simply execute a set of instructions, which could then be easily copied or moved to a new machine. Additionally, it proposed a so-called *Universal Turing Machine* that could read and simulate the behavior of other machines, given as input on the tape. These thought experiments had many direct influences on the years to follow, even though it took some time for them to be realized, and can be seen as the earliest conceived notions of compilers and interpreters, which were finally

implemented in 1952, and emulators and simulators, which were only recently implemented efficiently in 1997.

The first electronic computers

*"The most frequent mistake is the assumption that progress in those early days was slow and plodding and that not much was happening in the field."
(Glass, 1998)*

When the ENIAC, the first large digital computer that could be re-programmed, was built in 1937, there was still no concept or need for portable software. In fact, there was no place where the instructions for running the machine were stored. Instead, every time the ENIAC was to compute a new problem, it had to be *set up* anew. And that meant re-cabeling all the hardware units manually, which could take days.

The manual setup was changed with the *Whirlwind* at MIT (completed in 1951), and the *Mark I* at Harvard (completed in 1944), where punched cards were being used to determine the order of operations. It was the Harvard Mark I that spun the modern creation of software in the United States. Grace Hopper was shared by the Navy to create programs for the Mark I and it was in her daily routines that she saw the need for easier reuse of code and later the compiler. Her ideas were not perfect but seemed to create a series of events which would quickly lead us to the modern idea of compiles and assemblers. Laning and Zierler created the first assembler in 1954, around the time that the first languages were being created and used.

FORTAN was released by IBM in 1957 and COBOL was popularized by the United States Government in 1960. It was during this time of innovation that the first examples of open source occurred. SHARE was a group of IBM users that joined forces to, if nothing else, share frustrating experiences. They managed to create many libraries of code, reducing the amount of redundant work between members. As the group grew, SHARE and IBM seemed to form a symbiotic relationship, in that SHARE created more profits for IBM and IBM in return placed great weight on SHARE's opinions and preferences. It was a great example of a developer community.

During these first years of software development, the study and perfecting of algorithms was quickly becoming its own subject area. Limitations of punch cards and tape reels forced extra processing time just to sort the files in an accessible manner, occupying 25 percent of all processing time. In order to be effective, code had to be precise and compact. Some of the best algorithms in place today were created and implemented prior to the 1970s. This, along with the constant improvements in hardware and the realization that computers can be used for more than just mathematics, created a growing need for programmers, directly creating a need for formal education in the area. Colleges began implementing computer science departments in the 1960s. The first programs

seemed to focus on languages and practical applications while contemporary programs focus more on theories and practices. The term “software engineer” arose in 1968, when people speculated that lack of engineering (theoretical foundations and disciplines) was causing the software crisis. The debate of computer programmer versus software engineer continues today.

Perhaps the most important occurrence in the advent of software was the unbundling of software from hardware by IBM in 1969, under pressure by the U.S. Government, and the rise of software companies. The industry slowly lost its focus on hardware, creating an expectation of reliable software. Likewise, the costs of software development started to exceed the costs for hardware.

Unix & C

“C was already implemented on several quite different machines and OSs, Unix was already being distributed on the PDP-11, but the portability of the whole system was new.” (Ritchie, 2003)

Between 1969 and 1972, the programming language C was developed by Dennis Ritchie, amongst others, at AT&T's Bell Labs, just like its direct ancestor B. Strangely enough, the initial motivation was to enable the programmers to play the video game *Space Travel* on a PDP-11. As it happened, and almost at the same time, the operating system Unics [*sic*] was being developed at Bell Labs, written in Assembler and for a PDP-11 machine. The researchers soon found that the finished high level language C would enable them to make Unics, in 1973 labeled UNIX, portable to almost any other machine, so most parts of it were then rewritten for that purpose.

The success of both C and UNIX was closely tied together. The fact that AT & T, being a regulated monopoly, was not able to sell UNIX for profit, along with its high portability, led to the fast acceptance of UNIX and C everywhere. Additionally, the source was published for a nominal fee, what led to a constant improvement of the operating system.

The impressive networking, file-handling and user management capabilities of UNIX influenced the early days of the Internet and many operating systems which are widely used today, such as Linux, *BSD or MacOS. Similarly, C was soon standardized, adapted to the PC and used by system programmers everywhere. Together with Bjarne Stroustrup's extension C++, C is probably the most used programming language today.

The impact of the *Origins* of Software

Software, in the modern form, made a great impact during its first few decades. Even though investment into a software company was not regarded as equally rentable as in a hardware start-up, the development of software companies

spawned many high-paid and equally respectable professions, some rather successful companies and even some movies (although maybe not so successful). Software has formed an everlasting place in society, because humanity could never return to life without it. Some people even hypothesize that we are already living in the age of artificial intelligence because humans are now capable of doing things not normally humanly possible.

While we had to learn that the origins of software can not be clearly distinguished from the hardware (originally, during our preparation on the topic, we would have thought to determine the first software appearing after the first compilers), it will be even harder to try to predict the future of software and hardware. Will further levels of abstraction evolve?

In any case, we speculate that software is still in its infancy and the origins of software will gradually come to include the present times. It's fascinating to be a part of history, especially one that has had such an impact on the world.

Bibliography

- [1.] *Ceruzzi, Paul E.: A History of Modern Computing.*
Cambridge: The MIT Press, 2003.
- [2.] *Glass, Robert L. In the Beginning: Recollections of Software Pioneers.*
IEEE Computer Society Press, 1998.
- [3.] **“Software – Know More.”**
http://www.01webdirectory.com/software_info.htm cited 29 October 2006.
- [4.] **“History of Computing: Software”**
<http://www.thocp.net/software/software.htm>
- [5.] **“Software History Center”**
<http://www.softwarehistory.org/>
- [6.] *Alan Turing: On Computable Numbers, with an Application to the Entscheidungsproblem*, 1926
- [7.] *Stephen Kleene: “Recursive Predicates and Quantifiers”*, 1943
- [8.] *Marvin Minsky: “Computation, Finite and Infinite Machines”.*
Prentice-Hall, Inc., N.J., 1967