Anecdotes

The Beginnings of TECO

Dan Murphy

Editor: Dave Walden

I came up with the name TECO in 1962 while sitting in Ye Hong Guey, my favorite restaurant in Boston's Chinatown during my undergraduate years at the Massachusetts Institute of Technology. I had been working on a little program to help me write other programs, a common activity in the day when a computer (the Digital Equipment Corporation PDP-1) would be delivered with almost no software. My program had gotten to the point that it needed a name, so my friend Pete Peterson and I were kicking around possibilities and their acronyms.

Years later, TECO would be known to mean Text Editor and Corrector and so described in its documentation. However, the original meaning—the one we came up with that night at the corner of Oxford and Beach streets—was Tape Editor and Corrector.

Interactive computer use on the PDP-1

We used the term ''tape'' because punched paper tape was the only medium for the storage of program source on our PDP-1. There was no hard disk, floppy disk, magnetic tape (magtape), or network. There was no operating system, only the paper tape reader and punch and a console typewriter. The paper tape reader was fast (or so it seemed at the time), reading 300 characters per second. The punch was considerably slower at 20 characters per second. The console device was an office-style electric typewriter that had been interfaced to the PDP-1 for character-at-a-time input and output.

The PDP-1 was the first computer built and sold by DEC and thus the leading edge of the minicomputer revolution. It was anything but ''mini'' by today's standards, however; it occupied several tall cabinets in its minimum configuration. All that, and it had a main memory of only 4,096 18-bit words (effectively 8 Kbytes). This was core memory made of tiny magnetic toroids strung on thin wires in a 3D mesh. It had a cycle time of 5 microseconds. The processor executed most instructions in two cycles (10 microseconds)—one cycle to fetch the instruction from memory, and the second to load and/or store the memory operand. Some instructions did not reference memory and so required only one cycle.

All this notwithstanding, it was exciting to actually be able to sit at the console and use the computer interactively. My prior experience was with mainframe class machines like the IBM 709 or 7090. Those required preparing a program on punched cards, submitting the deck to the computer center for a batch run, and getting a paper listing of the results the next day. With the PDP-1, the code-compile-debug cycle was shortened from hours to minutes. I could do it several times on the machine in a one-hour session.

In a smart move for a young corporation, DEC had donated a PDP-1 to the Research Laboratory for Electronics (RLE) at MIT, and it was sitting in a room more-or-less available for anyone, including undergraduates, to play with. There were soon many takers. Because only one person could be using the PDP-1 at a time, computer time quickly became scarce, and it was still necessary to do as much program preparation as possible offline. We did this with Frieden Flexowriters, typewriter-like devices with a paper tape reader and punch. Programs had to be handwritten first because editing capabilities on the Flexowriter were nonexistent. The most you could do was nullify a previously punched character by backing up the tape punch and over punching the same line.

Paper tape was more convenient than punched cards in some ways. It was lighter than cards for a given number of lines of program, and if you accidentally dropped it, it did not shuffle all your lines of code. This was also its biggest weakness. When you needed to modify a program—add, change, or delete a few lines—you could not just replace those lines as you did with the cards; you had to punch a whole new tape. Hence, editing on the Flexowriter was a painstaking process of reading an existing source tape and simultaneously punching a new one, while stopping at each point where changes were to be made to type new material and skip over unwanted old material.

It was also possible to edit on the computer, which is what we did while debugging and recompiling during our slots of computer time. The editor on the PDP-1 had a simple line-oriented interface that let users step through an existing paper tape, adding or replacing lines to produce a new generation of the source. Using this editor consumed precious computer time, however, and it used the whole computer to do a ''simple'' task, so it was known as Expensive Typewriter.

That Expensive Typewriter could only add, delete, or replace entire lines of text annoyed me considerably. It was a bothersome fact of punched cards that making any change on a line required retyping that whole line on a new card, and I didn't see why that limitation needed to exist for this new medium where there was no physical separation between one line and the next other than the carriage-return character.

The birth of TECO

Thus, I decided to write a program that, like Expensive Typewriter, would let me copy a paper tape

 Published by the IEEE Computer Society 1058-6180/09/$26.00

while making changes along the way and would let me make those changes character by character. Accordingly, the first few commands would include these types of functions:

- delete a character,
- delete a line (it was still convenient to work on lines sometimes),
- insert text (any amount from one character to multiple lines), and
- search for text at the point where you wanted to make changes.

TECO was not initially intended to be an interactive, online editor. Because of the scarcity of computer time, online editing was still considered somewhat squanderous, and we really did not want to wait until our next time slot to get the next source revision ready.

Therefore, TECO was intended to be used offline, on the Flexowriter, where we would prepare a separate paper tape with the commands needed to edit our existing source tape. As people often say, ''it seemed like a good idea at the time.'' In other words, the idea was to hand-mark changes on your source listing and enter the TECO commands necessary to make those changes on the Flexowriter (i.e. find a spot, delete, insert, move on). Then, when our computer time came, we would load TECO and read in our commands tape, so TECO could read our old source tape and quickly punch a new one.

Hence, in the beginning, TECO was a language for describing changes to a source program or, more generally, a stream of text. This turned out to be its real power as time passed and its evolution continued.

TECO becomes interactive

In the short term, however, offline command preparation proved to be impractical. It was impossible for most ordinary mortals, myself included, to consistently prepare command tapes that would work right. Like any other program, they would have bugs that would only become evident while on the computer. Thus, I added a mode to TECO early on that would dispense with reading the command tape first and simply let us enter commands at the console typewriter. Needless to say, this was an entirely different experience. We could enter one or a few commands, make sure the result was right, and then move on.

Although quite interactive, TECO was not a screen editor at this point. Although our PDP-1 did have a CRT display device, neither the paradigm nor the code existed for displaying text. Hence, source was kept on paper printouts, and editing online meant looking at a few lines at a time typed out on paper.

With interactive editing, a number of other enhancements were soon required. It quickly became clear that moving through a source tape in only one direction was still awkward. An especially easy, but painful mistake would be to mistype a search command so that the text you wanted would not be found and the copying would continue to the end of the tape. To solve this problem, I decided it would make sense to read a chunk of the source into memory where a number of edits could be made before committing it to the new paper tape. The natural unit for this was one typewritten page. Most sources were paginated to this size because we were using letter-size fanfold paper stock in the Flexowriters and console typewriter.

This was a big help. The usual search was limited to the current page, so a failed search was no problem. Just go back to the top of the page and try again. Plus, we could make some changes, see how they looked, and if still not right, do some further edits.

Page-at-a-time editing required new commands for keeping track of the location that editing would take place (the point), for moving to the next page or to the end of the tape, and for searching that was not limited to the current page. This search still carried the risk of going too far, but since it was used less often and only to get you to the right page, that risk was much lower.

You might ask, why only one page at a time? Why not read in the whole source tape? As I noted earlier, our PDP-1 had a total memory component of about 8 Kbytes, and that had to hold the entire TECO program as well as whatever text was being edited. It was not until many years later that memories became large enough that yanking in the whole file became practical and routine.

One enhancement in that direction became possible when the RLE PDP-1 had a swapping drum added to it. Although this device was not large enough to serve as permanent storage for programs or sources, it was fast and could be used to work around the limited main memory size. For TECO, I implemented a kind of paging mechanism

Alleged Patent Infringement

As simple as it seemed, even at the time, this method of command processing became the subject of a patent infringement case some years later. This was in the late 1980s, and I was working for DEC in the VAX/VMS group. One day, I got a call from someone in the DEC legal office. It seems that IBM was being sued by some small company because of alleged patent infringement by the IBM PC. Some basic programming techniques were asserted as patented IP by this company, and in looking to defend itself, IBM had done a search for prior art. One of the issues was character-based command dispatching, and somehow, IBM found TECO, which was originally implemented years before the claims of this challenger. IBM identified me as the original implementer, traced me to DEC, and contacted the DEC legal department to ask for assistance. I consulted with some IBM attorneys and was looking forward to a full-blown trial. I never got the call though, so I presume the case was settled well short of that.

that allowed the main editing buffer to be many times larger than could fit in memory. I would draw on some of these paging ideas a few years later in the implementation of other multiuser systems culminating in TENEX and TOPS-20.[1,2]

Punched paper tape remained in common use through at least the late 1960s on a number of systems, but TECO supported other media almost as soon as it became available on any machine we used. One of the other PDP-1s at MIT had a magnetic tape drive, and I quickly added a magtape driver to TECO. Again, there was no operating system, or even common I/O libraries at that time, so each program had separate commands and I/O routines added for each device.

Magtape was never practical for program storage, of course, even though it was wonderfully fast compared to paper tape. Much better was DECtape (or MicroTape), the block-addressable small-reel tape system eventually added to the PDP-1 and available on all DEC systems for many years. DECtape had a basic file system that supported a modest number of named files on each reel and the ability to rewrite files any number of times. This required TECO to grow the ability to open a file by name, not just yank a stream from a serial input device.

Even when timesharing and disk-based file systems came into common use, TECO editing continued to be a page at a time for the most part. Conservative memory use was still important, and for all its power, TECO was economical to run.

The TECO language

I did not give much thought to the TECO's ''human interface'' design at the beginning. The concept really didn't exist around program tools at that point. The most commonly used interactive program was the DDT debugger,[3] and it had a set of single-letter commands with numeric arguments preceding. For example, to examine location 123 in your program, you would type `123/`. DDT would act immediately on the slash and print the contents, so the line would then appear as `123/ lac foo`. TECO followed that form with commands such as

- `5d`, meaning delete five characters;
- `3c`, meaning move forward three characters;
- `10l`, meaning move forward 10 lines;
- `2k`, meaning kill (delete) two lines; and
- `iTEXT<term>`, meaning insert ''TEXT,'' or everything from the initial `i` to the special text termination character.

As I noted earlier, I based the initial design of the TECO commands on entering them offline, but they continued to work well when interactive use became the rule. (See the sidebar for an interesting side note on character-based command dispatching.)

From that relatively simple start, I went on to add further enhancements. Some of them were around making the language more consistent and general. For example, we did not need a separate command to move the pointer in reverse (the original `r` command); a negative argument to the character-move-forward command was quite natural. The same applied to commands that operated on lines—`-3l` meant move backward three lines, and `-5k` meant kill the preceding five lines. `0` (zero) meant the current line, so `0l` meant to move to the beginning of the current line. A number of simple constructs quickly became a habit for any TECO user, such as using `0kk` to kill all the current line regardless of where the pointer is on it. (TECO uses the convention that the default argument is `1` for most commands.)

TECO becomes a programming language

By late 1962, TECO had about the level of capabilities that simple editors would have for years to come, but TECO's development didn't stop there. Many of the subsequent enhancements were done simply because something occurred to me that I could easily do or someone suggested something, and it seemed like a cool idea. Our mindset was that cool ideas were fun to pursue, even if we couldn't think of a practical use or need right away.

More significant additions—the ones that began to give TECO some power as a programming language or text engine—included text buffers and the looping construct. TECO handled simple program modification fairly well at that point, but it did not offer a way to do significant restructuring (e.g., move a sizeable amount of text from one place to another or replicate a segment of text a number of times). To provide this, I added the concept of text variables—that is, a variable that would hold an arbitrary section of text. The command to move a range of text into a text variable was `x`. (I was running low on unused letters by then.) Its prefix argument(s) identified a range of text in the buffer in the same way as the `k` command, and the single letter following the `x` was the variable name from the set a–z, 0–9. `g` (get) was used to copy the contents of a text variable into the main buffer at the editing point and could be used any number of times to replicate copes of the text. Despite the lack of a mnemonic association for `x`, it quickly came to seem natural, almost intuitive. To `x` something became as easily spoken and understood by TECO users as ''store'' would have been.

Then, given that we had the ability to store any text in a variable, I implemented a command to execute a text variable as a TECO command string. An optional argument could be given in the form of a preceding expression. Thus, TECO came to have a form of macro or callable function, and it followed from the original concept that a concise string of text would represent commands to modify other text.

Around that same time, I also found that I wanted to be able to make repetitive and systematic changes to the source text. This included simple cases like changing all occurrences of ''foo'' to ''fie'' (which would later be supported by the replace command) and things beyond simple search and replace, such as ''find every line with 'foo' in it and add a comment at the end.'' This gave rise to the original loop construct: `( commands boolean ; commands )`.

The loop would be executed zero or more times until the test was false. A search variant was available that returned true only if the text was found, so search and modify loops were straightforward. The loop-begin and loop-end commands appeared as parentheses with a dot in the middle on the PDP-1. Subsequent ASCII-based versions of TECO used angle brackets for this purpose.

Integer variables were also added around this time, again from the single-letter set a–z, 0–9. `u` (use) would store its argument into the variable, and `q` (quantity) would stand as the stored value in an expression. The same variable name could have both a text and numeric value; the command determined which was being referenced. Because simple arithmetic expression were accepted, a loop to run, say, five times could be written as

```
5ua
( qa; commands qa-1ua )
```

Spaces and line breaks were not syntactically significant. As this simple quasi-example shows, TECO was nothing if not terse. Fairly complex loops and other command sequences could be written in TECO, and mostly looked like line noise. TECO was one of the first languages to spawn the practice of handing someone a one-line string of near gibberish and asking with a grin, ''tell me what it does.'' Nonetheless, TECO made it onto Jean Sammet's list of known programming languages sometime in the late 1960s.[4]

There have been innumerable comments over the years about how obscure the TECO language is. Few would dispute the premise that it is easier to write than to read. As an editor however, its brevity was one of its advantages. Regular users quickly learned the more frequent command sequences and did not even need to think about them. Especially after WYSIWYG editors became common, the claim that ordinary users could only handle verbose, line-oriented editors disappeared. This was hotly debated within DEC for some years, pitting developers against the business and marketing folks. Today, if we considers all the obscure keystroke sequences available in the most commonly used software, with modifiers Control, Alt, Shift, and others, TECO seems almost simple.

TECO moves to new machines

Within a few months, TECO was being used in several PDP-1s around MIT. Besides the first one in RLE, there was one being used for analysis of bubble chamber photographs by Professor Martin Deutsch. In my last undergraduate year or so, Deutsch hired me to do further work on TECO and some other programming tools that I had developed.

A PDP-1 had also been installed in the new Tech Square building where Marvin Minsky had his AI lab. This was also where the first DEC PDP-6 at MIT would be installed in 1965, and it become the first machine to which TECO would be ported. I did not do that port myself; it was done by Stewart Nelson, Richard Greenblatt, and Jack Holloway, who had been using the AI PDP-1 and were among the first to get their hands on the new PDP-6. Some enhancements had already been made to the PDP-1 version by the AI group, and it was used heavily there. Hence, porting it to the new PDP-6 was an urgent project. I had earlier been offered a part-time job by DEC to do that port for their new, still-in-development machine, but I was trying to finish my senior year without flunking out, so wisely I declined.

PDP-1 TECO was written in assembly language, and simply recompiling the source, even with a few edits, was impossible. However, the PDP-6 had a far more powerful and general instruction set, so much of TECO could be simply (if only manually) translated instruction by instruction while still remaining in assembly language. One big difference was the change of character set. The PDP-1 used an encoding called FIO-DEC (the Frieden paper tape code as adopted by DEC). The PDP-6 used a Teletype Model 35 or 33, which supported ASCII (upper-case only).

Another enhancement begun in the AI PDP-1 TECO and made fundamental on the PDP-6 was to use the display to show the text around the editing point. This was not yet on-screen editing in the current sense; commands were still entered and echoed on the console teletype, but the feedback to the user was much enhanced. It wasn't until some years later that CRT-based terminals became common. Until then, most users could only use TECO in the traditional way on a hard-copy device.

Many other enhancements were made in the next few years, fostered by the PDP-6's much larger memory, greater power, and programming ease. The command set for that version can still be found with a Web search today and, as far as I know, was by far the most complex version of TECO ever in common use. Although there were numerous other ports for other machines in the following years, most of them were much more basic and similar to the PDP-1 version. Even the version of TECO included by DEC in its initial release of the PDP-10 (DecSystem-10) was more basic. Bob Clements (at DEC at the time) had taken a relatively early version of the PDP-6 AI TECO, stripped it down to remove things that were not available on most installations (e.g., the expensive CRT display) or to timesharing terminal users, and that became the version shipped by DEC on the PDP-10 and later PDP-6 installations. It was also the version that came into my hands when Bolt Beranek & Newman (BBN) got a PDP-10 (1969), which I would evolve from then through about 1986.

The Spirit of Open Source

From the time of TECO's original development through the PDP-6 port, we were all happy to have DEC or anyone else take the programs we had written to use or even sell. For an undergraduate like myself, having people use and like them was the reward. Besides, nobody had any idea how or whether software could be copyrighted or otherwise protected. This was the open-source model in action long before that term came into use.

Starting with the PDP-6, TECO was included in the base software set of every system that DEC offered, including the PDP-8, PDP-11, DecSystems 10 and 20, and VAX/VMS. It also continued to be heavily used and enhanced by the hackers[5] in Minsky's AI lab, eventually including Richard Stallman. In the early to mid-1970s, Stallman would make some key enhancements to TECO that allowed it to become a fully interactive, WYSIWYG-style onscreen editor. A key piece of this implementation was an enhancement where an individual keystroke could be made to immediately invoke a macro (i.e., a function to be executed). This included the normal graphic keys, so typing an `a` in normal text entry mode invoked a function to insert `a` into the buffer and, of course, update the onscreen display. This resulted in many users creating keystroke-triggered editing operations, and before long, an effort was made to collect the most useful of them into a set that could be documented and used by others. This became the original Editor Macros (Emacs).

TECO would continue to be the text engine for EMACS for several years, but it was

ultimately replaced by a Lisp-like text manipulation language known as Emacs-Lisp.

Several versions of TECO for soft-copy terminals would also be done, including one by me called VTECO for TENEX and TOPS-20. A number of dialects of the TECO command language also grew up, as various branches of development and evolution continued in different places. The DEC Users Group (DECUS) had a TECO Special Interest Group (TECO-SIG). Although DEC products continued to be where TECO was most consistently available and used, it was ported to many other products as well. The lineage of many of these versions is now difficult or impossible to determine.

Reflections

Clearly, TECO has had numerous users over the years. A relative few wrote or attempted to write actual programs in TECO. One of my early exercises was to write an arithmetic expression compiler in TECO. This would take an expression like `x = a + (b − c)*3/d` and translate it to a sequence of PDP-1 assembler instruction to perform the computation. I never attempted to make it a practical compiler though; it was mainly a way to test the completeness of the TECO command set and a cool hack. Other known TECO programs included an early email interface—a set of TECO macros from Larry Roberts at ARPA that let you selectively read messages from your email inbox.

Although this level of TECO hacking was unusual, many users would come up with their own customizations in the form of handy command sequences that could be saved as TECO macros and quickly invoked in an ordinary editing session. Unlike the keystroke captures in some of today's editors, TECO macros were plain text, so they could be edited like any other text to debug and enhance them. With that and the ability to use programming constructs like loops and conditional searches at any time, TECO offered capabilities that many former TECO users, myself included, find lacking in contemporary editors. Things I used to do in TECO on the fly must now be done more laboriously, or if that isn't tolerable, not done in the editor at all but rather using other tools such as Grep, Sed, Awk, and Perl.

For anyone who would like to run TECO now, at least occasionally, there are versions available on the Web for most common systems.[6]

References and notes

1. D.G. Bobrow et al., ''TENEX: A Paged Time Sharing System for the PDP-10,'' *Comm. ACM,* vol. 15, no. 3, 1972, pp. 135–143.
2. See Dan Murphy's TENEX and TOPS-20 papers at http://tenex.opost.com/
3. DDT originally stood for ''DEC Debugging Tape'' as, like all programs for the PDP-1, it was loaded from a paper tape. DEC later changed that to ''Dynamic Debugging Technique'' as paper tape faded away.
4. Jean Sammet's list appeared periodically in the *Comm. of the ACM* (see vol. 19, no. 12, 1976).
5. S. Levy, *Hackers: Heroes of the Computer Revolution,* Doubleday, 1984.
6. Two good places to find them are http://almy.us/teco.html and the Wikipedia article on TECO at http://en.wikipedia.org/wiki/Text_Editor_and_Corrector.

Readers may contact Dan Murphy at dan.murphy@dlmmx.com.

Contact department editor David Walden at annals-anecdotes@computer.org.