design

Doing it my way: a lone TEXer in the real world

Jean-luc Doumont
JL Consulting
Achterenberg 2/10
B–3070 Kortenberg, Belgium
email: JL@JLConsulting.be

abstract
While a world-renowned standard in many academic fields, Don Knuth's much acclaimed typesetting system is almost unknown in most parts of the real world, where many a document designer has achieved professional success without ever hearing (let alone pronouncing) the word "TEX". Outside academia, the lone TEXer faces not only compatibility headaches, but also outright incomprehension from his customers, colleagues, or competitors: why would anyone want to use TEX to produce memos, two-color newsletters, full-color brochures, overhead transparencies, and other items – in short, anything but books that contain a lot of mathematics?

As a consultant in professional communication, I have been using TEX for all documents I have produced for my clients and for myself during the last ten years or so. Though it has turned out to be most successful, this approach is seen by most as a mere idiosyncrasy. And yet, the systematic use of my own TEX and PostScript programming gives me three unequalled advantages over using off-the-shelf software: I travel light, I can go anywhere I please, and I guarantee I'll get there.

Introduction

Being someone who (among other activities) produces documents for a living, I often have to discuss software issues with clients, colleagues (or competitors, as the case may be), and service bureaus or print shops. When I say I use "TEX," these people usually first ask me to repeat, then express their surprise. Some immediately dismiss it as "never heard of"; others first ask what exactly it is, before wondering why I would want to use something "that nobody else uses" instead of a WYSIWYG, "professional" application (the one *they* use, no doubt).

My using TEX (and PostScript) for virtually all documents I produce, successful as it may turn out, has puzzled many a TEX user, too. Is TEX really the best tool for documents other than long, structured, or heavily mathematical ones? Is a direct-manipulation, integrated application not better suited to producing short newsletters, graphs, or overhead transparencies? The most qualified answer is likely to be, "Well, it depends."

This paper relates the experience of a long-time lone TEXer in the real world. It explains choices, points out advantages and limitations, and draws lessons from the experience.

Approach and opportunities

The very varied documents my company produces can be grouped into two categories. Some are in-house documents, such as letters, reports, and teaching materials (handouts, lecture notes, overhead transparencies). The others are documents we create on behalf of clients, including newsletters, brochures, corporate reports, and overhead transparencies. Documents of both categories may be black-and-white, two-color, or full-color ones, printed in traditional (offset) or modern (digital) ways, in small or large runs.

To be able to guarantee the quality of the documents we produce for our clients, we have opted to give them non-editable deliverables only – in practice, paper, film, or ready-to-print electronic files (in PostScript, for example). We thus strive to preserve the quality of both the writing and the typesetting against two sources of undesirable alterations: unwise last-minute modifications by the clients themselves, often ruining a consistency they may not readily perceive, and accidental changes caused by an all-too-theoretical "seamless conversion" between platforms or versions of a given software application.

Our insistence on not giving away editable files is a tough one to maintain (admittedly, it has suffered exceptions): clients logically consider as theirs the text they have paid us to write or the format they have paid us to design and produce. Yet our name in the list of credits constitutes our best – if not our only useful – marketing tool: we cannot afford to take chances. We will, of course, ensure that what we create addresses the client's needs while attaining the quality level we strive for.

Though our approach is dictated by quality standards, not ease of production, it does simplify our work – to some extent, that is. Because we need not exchange editable files back and forth with external parties, we are free to choose not only the platform but also the software tools with which we work. More accurately, we must be able to process incoming ASCII files and occasional images in GIF or JPEG format, and to produce PostScript files for laser printers, imagesetters, or others still – all in all, minor constraints. Our software tools are therefore essentially limited to an ASCII editor, an image processor, and, of course, a TEX environment, which we use to produce virtually all our paper documents.

Surprise and frustration

The choice of TEX as all-purpose tool for producing very varied documents surprises those who know TEX and puzzles those who do not. Clearly, it does stem from my previous experience with TEX in an academic setting (Stanford University, which happens to be TEX's cradle), and from my technical education (applied physics) and consequent preference for analytical thinking. Admittedly, it may also partly originate in my pronounced taste for computer programming and, more generally, in my peculiar habit of wanting to (re)do things my own way. Still, the perspective provided by some ten years of successful professional practice vindicates my choice: over the years, I have satisfactorily moved to TEX for more and more types of documents.

Until I entered the "real world," I failed to realize how poorly known TEX is, at least over in Europe. It is known by the more technically minded participants at the training programs I teach in academia and national research centers, though even these people are often unclear about what TEX really is and typically confuse TEX and LATEX. In contrast, it is an unknown entity to clients who entrust us with document-creation projects, even those in research organizations, typically from the Corporate Communication or Public Relations department. These clients, used to mainstream direct-manipulation packages, have difficulties understanding the nature of TEX, the difference between TEX itself and its implementation on a specific platform, and the fact that TEX cannot do much (relatively speaking) until one programs it.

Irrelevant as it may seem, given the lack of compatibility issues in our case, the choice of TEX in an essentially non-TEX world proved a liability to our credibility. I had expected that clients who did not know TEX would be content to judge the tool by the result; I was wrong, for at least two reasons. First, most of our clients are little able to judge the quality of a typeset page; their eyes somehow seem blind to stretched out lines, uneven line spacing, and other basic points. Yet the eventual readers of the documents may well notice the difference (if unconsciously), so producing high-quality pages still matters. Second, our clients judge merit by popularity and hence distrust or even disdain what they do not know; they figure that, if they have not heard of TEX, then it cannot possibly be any good. Consequently, and independently from what they see, they are suspicious of the pages I typeset with TEX. Ironically, they seem infinitely more reassured when I simply tell them I use "a system I programmed myself" rather than "the system developed by Professor Donald E. Knuth".

Distrust of the unknown goes beyond the clients. Many of the print shops with which clients sometimes ask me to work dislike my giving them PostScript files instead of editable ones. PostScript being akin to Greek to them, they somehow feel deprived of their power and will not admit I have done half of the work for them. When anything goes wrong at any stage of the printing process, they are quick to pin the blame on me, insisting to the client that I use a software application nobody else uses, "so it's bound to create problems". (As an extreme case, one renowned print shop even resorted to this excuse after mistakenly using a Pantone color other than the one I had specified in writing.)

People who do understand what TEX is, after some explanations, wonder why on earth I prefer using such an intricate system rather than a much more user-friendly (and better known) package. At first, the advantages of TEX were so intuitively obvious to me I was unable to put them into words. Extolling the line-breaking algorithms, positioning accuracy, or logical markup was pointless: these people either saw no benefit in the features or insisted they could do all of that with their own text processor. To a point, they were right, of course, even if we all feel that "it's not the same".

A little help from my friends

At a loss for words in the defense of TEX, but convinced from experience that it suited my work so well, I set out to ask other users why they preferred TEX. I contacted the makers of my own TEX implementation, posted questions on `comp.text.tex`, and asked around. The answers were varied but can be grouped into five categories: output quality, flexibility, text-based formats, reliability, and portability.

Not surprisingly, many who answered my post on `comp.text.tex` summed it all up by saying that TEX's output "simply looks better," especially as regards mathematics. Among others, they pointed to such features as transparent use of optical sizes, better hyphenation algorithms, and line-breaking algorithms that act on whole paragraphs, not line by line.

Besides the quality of its output, users love TEX for its flexibility, allowing one to extend its capabilities almost ad infinitum. Some mentioned virtual fonts, custom hyphenation patterns, and non-European languages. Others lauded TEX's superior capabilities for floating inserts and cross-references. Following the theme of the present TUG conference, some also underlined the parallel writing of printed and HTML documentation.

Less expectedly perhaps, users praised TEX's ASCII roots. A plain-text format, they said, allows easy manipulation with any text editor, easy generation of TEX files by other applications (including preprocessors), and small files (hardly larger than the actual text) that compress well. Some also mentioned the small size of their TEX implementation, compared to that of popular text processors.

TEX's text files, batch operation, and stability over time were seen as the basis of its reliability and portability. In contrast to those of integrated applications, the TEX (source) files can be damaged only by the text editing, not by the formatting, the displaying, or any other downstream operation. Moreover, source files typeset with TEX ten years ago can be typeset again today – on any platform – to produce the exact same output, in the exact same device-independent format. TEX files, being plain ASCII, can also safely be exchanged by electronic mail across the world.

Besides the above observations, the (occasionally emotional) discussion on `comp.text.tex` as a result of my post pointed to two interesting differences. Trivial as these may seem in retrospect, they helped me understand much of the religious wars around software: they are the differences between a software tool's

- claimed and actual capabilities, and
- its potential and actual use.

Reliability and portability are typical issues that differentiate between claimed and actual capabilities. There is no intrinsic reason why an integrated application should be unreliable, although increased complexity and fast-changing versions certainly increase the odds. Similarly, there is no intrinsic reason why the claimed interchangeability between platforms and between versions should turn out untrue. Yet a pragmatic point of view suggests otherwise.

Furthermore, the fact an application offers a given feature does not imply that its users actually use it – often a question of usability. As an extreme example, any graphical application that allows one to position characters precisely anywhere on the page can produce output that matches TEX's – but at what cost? TEX, or so its users say, not only makes some operations even possible at all, it also makes many operations easier than direct-manipulation software.

Actual use is also largely affected by stability. In never-ending debates on software, protagonists who claim their own tool to be more efficient than that of others may well *all* be right, for the tool each masters best is the most efficient for him or her. Yet mastery requires time and users feel little motivated to learn to master a tool that will soon change: fast-evolving software, with pressure or even obligation to upgrade regularly, may offer ever-increasing capabilities, but discourages in-depth learning.

As a final point, the `comp.text.tex` discussion also clarified WYSIWYG (What You See Is What You Get), a concept often confused with that of direct manipulation. If WYSIWYG denotes faithful correspondence between screen view (what you see) and paper output (what you get), then some `.dvi` viewers are the closest to WYSIWYG one can get. Many people, however, actually mean that what they *do* affects the output in a way they can instantly *see*, a characteristic of direct-manipulation software. For accurate positioning (for example, to align two words exactly), direct manipulation may not be the most practical approach.

Agreed, but...

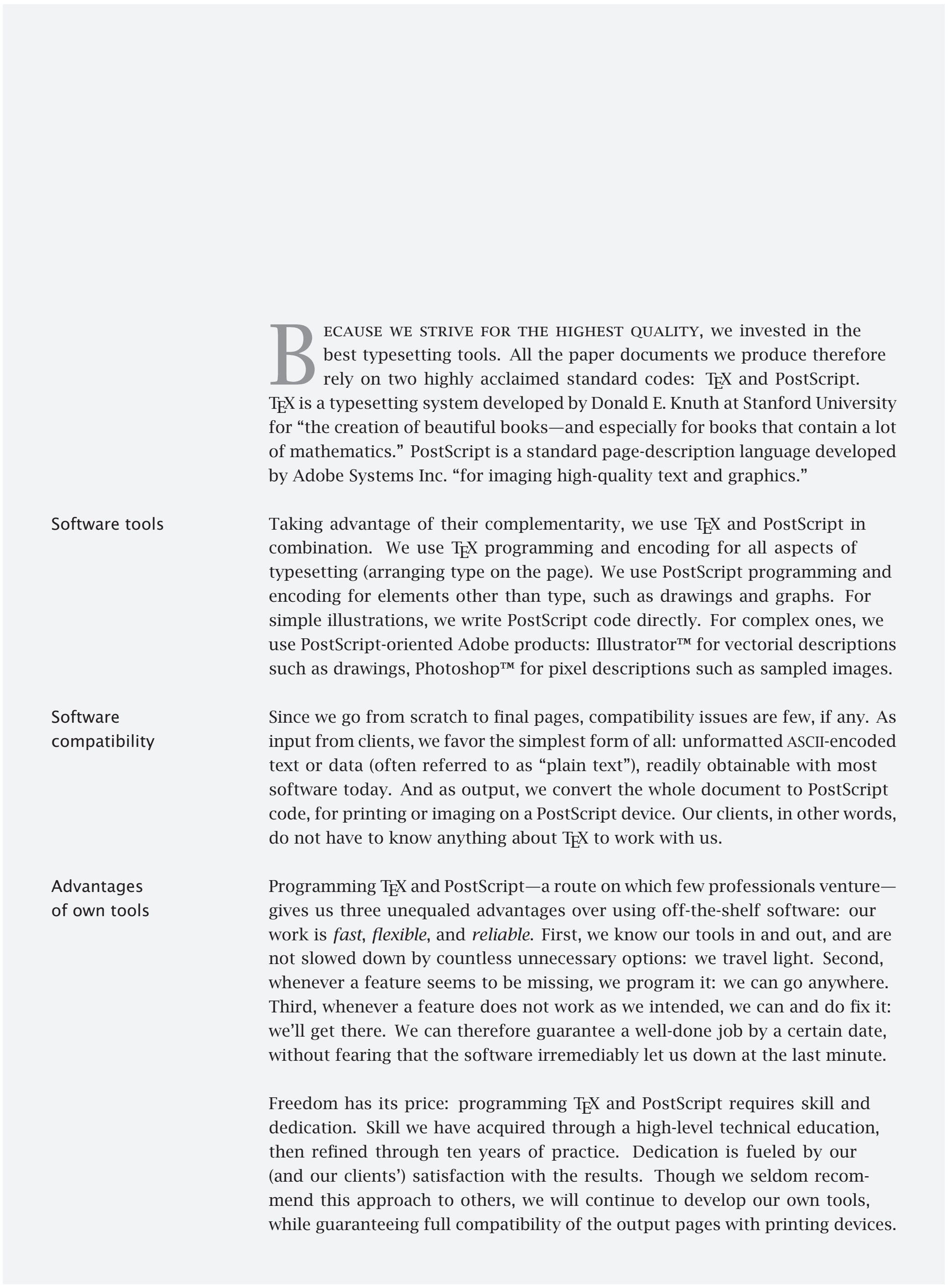
Answers from other TEX users expressed some of my intuitions in words, clarified some of my own mental shortcuts, and triggered new thoughts. I agreed on all advantages presented, but felt something was still missing. Most users enthusiastically put forward that TEX is unequaled for *some* documents; very few suggested they were, like me, using TEX for *all* documents they produce.

Users see TEX as particularly suited to the production of large, structured documents. Its batch process, ability to input files in other files, and virtually unlimited capabilities (such as number of paragraph styles) make large projects easier to handle. The separation of contents and visual appearance that it encourages (via macro programming or use) allows one to focus more easily on content, structure, and style.

To my surprise, many a TEX user I talked to admitted to resorting to direct-manipulation applications for short, less-structured documents (letters being the typical example) because "It's so much easier." I wonder what, specifically, they find easier. Letters – business letters, that is – may be seen, if not as one large, structured document, at least as a uniform collection of documents. Consistency, therefore, is as important to the 250 business letters I write every year as a one-off long report. Such a level of consistency, it seems to me, is more easily and more reliably achieved with TEX than with direct-manipulation software. As an example, the instruction `\JL99001 TME/MDe` typesets a letter header with my company's logo, my name, the date (in the proper language), the letter's reference number (here, `99001`), and the complete address block of my addressee (here, person `MDe` from company `TME`, as specified in a data file).

The major perceived obstacle to using TEX, to which some `comp.text.tex` members rapidly pointed, is its steep learning curve. TEX, or rather TEX packages, are often quite immediate to *use*. Despite a marked aversion for computer programming, my business partner used my TEX macros very readily, even without the user manual I never

The two pages on our using TeX and PostScript, out of
a short document explaining our approach to our clients.



BECAUSE WE STRIVE FOR THE HIGHEST QUALITY, we invested in the best typesetting tools. All the paper documents we produce therefore rely on two highly acclaimed standard codes: TeX and PostScript. TeX is a typesetting system developed by Donald E. Knuth at Stanford University for “the creation of beautiful books—and especially for books that contain a lot of mathematics.” PostScript is a standard page-description language developed by Adobe Systems Inc. “for imaging high-quality text and graphics.”

Software tools

Taking advantage of their complementarity, we use TeX and PostScript in combination. We use TeX programming and encoding for all aspects of typesetting (arranging type on the page). We use PostScript programming and encoding for elements other than type, such as drawings and graphs. For simple illustrations, we write PostScript code directly. For complex ones, we use PostScript-oriented Adobe products: Illustrator™ for vectorial descriptions such as drawings, Photoshop™ for pixel descriptions such as sampled images.

Software compatibility

Since we go from scratch to final pages, compatibility issues are few, if any. As input from clients, we favor the simplest form of all: unformatted ASCII-encoded text or data (often referred to as “plain text”), readily obtainable with most software today. And as output, we convert the whole document to PostScript code, for printing or imaging on a PostScript device. Our clients, in other words, do not have to know anything about TeX to work with us.

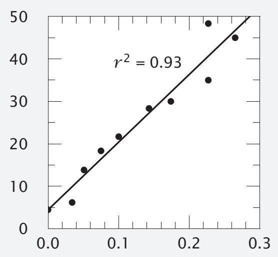
Advantages of own tools

Programming TeX and PostScript—a route on which few professionals venture—gives us three unequaled advantages over using off-the-shelf software: our work is *fast*, *flexible*, and *reliable*. First, we know our tools in and out, and are not slowed down by countless unnecessary options: we travel light. Second, whenever a feature seems to be missing, we program it: we can go anywhere. Third, whenever a feature does not work as we intended, we can and do fix it: we’ll get there. We can therefore guarantee a well-done job by a certain date, without fearing that the software irremediably let us down at the last minute.

Freedom has its price: programming TeX and PostScript requires skill and dedication. Skill we have acquired through a high-level technical education, then refined through ten years of practice. Dedication is fueled by our (and our clients’) satisfaction with the results. Though we seldom recommend this approach to others, we will continue to develop our own tools, while guaranteeing full compatibility of the output pages with printing devices.

The beauty of TEX

TEX (pronounced "tech" or "teck") is not so much a word processor as a programming language, complete with typed variables, block structure, executable statements, and a powerful macro facility. The TEX compiler turns a one-dimensional source program into a two-dimensional typeset page. Below is an example of source code (right) and resulting output (left).



```
\draw[\white]{8\pc}{8\pc}
\xrange03{10} \xaxis01{10}34
\yrange0{50}1 \yaxis0{10}151
\markerdata
0.034 6.1667  0.051 13.833  ...  \enddata
\linedata
0.0 4.45  0.3 52.28 \enddata
\an{0.2}{35}{$r^2=0.93$}rb
\enddraw
```

TEX's advantages are numerous. Dedicated to high-quality typesetting, it produces the best output available today, especially for such tasks as kerning lines, hyphenating paragraphs, and displaying mathematics. It allows accurate positioning (better than 1 µm) and can tackle virtually any language, in any alphabet. As a clearly defined language, it is platform-independent and stable over time, allows a high level of automation and extension, nicely separates contents from format (thus encouraging logical design, rather than visual one), and works with plain-text source files (smaller, easier to manipulate, edit, transfer, and compress, and much harder to mangle by accident than word-processor files).

The example graph above exemplifies some of TEX's advantages. The plain-text code is compact and compiles fast to produce a consistent graph. Typeface, tick lengths, and line widths are defined at the top of the document, and apply by default to all graphs. The graph is exactly eight interlines high, and is shifted down by $^1/_4$ of an interline; bottom labels are aligned to a line of text, contributing to the overall harmony of the page.

Because typesetting is a complex process, so, unavoidably, is TEX. While merely using existing macros is simple—much simpler, in fact, than using a word processor—writing one's own set of macros is not. Though we would personally settle for nothing less, we consider TEX a tool for professionals and a priori do not recommend its widespread business use around the office.

got around to writing. By contrast, TEX, or rather the fine programming of TEX, is not that immediate to *learn*, as confirmed by Knuth's "dangerous bend" signs in *The TEXbook* (1984). The same business partner was often at a loss when something unexpected occurred: the mental paradigm she had built through practice was insufficient to understand those cases.

Conclusion

Faced with people who equate "most popular" with "best", I stopped saying I work with TEX in casual conversation. Today, if people ask, I usually say I use "my own system" and refrain from explaining further. If people insist, I give them a short document that explains my company's approach, including its choice of software tools (Figure 1). Because they may care little about output quality and because our approach bypasses portability issues, the two pages on TEX and PostScript emphasize the three other categories mentioned above: we say it allows our work to be *fast*, *flexible*, and *reliable*.

While I have been a TEX enthusiast for over ten years, I have never been a TEX evangelist. More importantly, while I am convinced that using TEX my own way is one of my best professional moves, I have never advised anyone to develop his or her own package from scratch, let alone use mine. Some clients, who do notice the superior output TEX allows me to produce, have asked what software tool I was using in order "to buy it for themselves". I have to disappoint them, telling them my "tool" could not really be bought. Other clients, who had heard about TEX, asked me for the documents' source files, so they could convert them automatically to HTML. How could their HTML converter know, for example, that my TEX command `\Cs[137]` produces ^{137}Cs?

Using TEX in the real world, where time and money matter much, may require a dedicated TEX wizard. A well-oiled macro package may save considerable time and money by yielding consistently beautiful documents fast. It may, however, not account for the admittedly limited but nonetheless important special cases. In those cases, real-world users may want to call upon a TEX wizard, sometimes on short notice. How severe a limitation this is depends on management strategy. Learning TEX, in my case, certainly paid off, but it was quite an investment, one that was eased by personal motivation but that may not make sense from a pure business point of view. Still, it has given me an edge in many demanding professional cases, in which I can – actually – do what others can not.

I may remain a lone traveler, but my mind is made up: I will go on traveling light, going anywhere I please, and resting assured I'll get there.

Reference

Knuth, Donald E.. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1984.