Keynote Address

DONALD E. KNUTH

Donald E. Knuth
Computer Science Department
Stanford University
Stanford, CA 94305

Notes on the Errors of TEX

At this 10th TUG meeting, my goal is to describe a longish paper that I recently had some fun writing.[1] That paper [6] contains a complete listing of the errors I noted down as I was developing and maintaining the TEX system during a period of more than 10 years. I've always believed that one of the best ways to learn is by a process of trial and error; hence I decided that the presentation of a true-to-life list of errors might be the best way to help other people learn the lessons that my experiences with TEX have taught me. And I suspected that the people at this conference might be as interested in this history as anybody is.

Of course no single project can be expected to illuminate all the aspects of software development. But the error log of TEX seems to provide useful data for understanding the problems of crafting a medium-size piece of software. It's hard to teach students the concept of "scale" — the enormous difference between textbook examples and larger systems — but I think that a reasonable appreciation of the complexity of a medium-size project can be acquired by spending about two hours reading through a complete log such as the one in [6].

My error log begins with all the corrections made while debugging the first version of TEX, which was a program consisting of approximately 4,600 statements in an Algol-like language. The log ends with all the changes I made as TEX was becoming a stable system, as TEX began to have more than a million users on more than a hundred varieties of computers. By studying the log you can see all the stages in the evolution of TEX as new features replaced or extended old ones — except that I did not record the changes I made when I re-wrote the original program TEX78 and prepared the final one, TEX82.[2]

Altogether the error log contains 865 entries so far. I've tried to analyze this data and to introduce some structure by assigning each of the errors to one of 15 categories:

A Algorithmic Anomaly
B Blunder, Botch
C Cleanup for Consistency
D Datastructure Debacle
E Efficiency Enhancement
F Forgotten Function
G Generalization, Growth
I Interactive Improvement
L Language Liability
M Mismatch between Modules
P Promotion of Portability
Q Quest for Quality

[1] The preparation of this paper was supported in part by National Science Foundation grant CCR-86-10181.

[2] Those changes were summarized briefly in another publication for early users [1].

R Reinforced Robustness
S Surprising Scenario
T Trivial Typo

Categories A, B, D, F, L, M, R, S, T are *bugs*, which definitely needed to be removed from the code; categories C, E, G, I, P, Q are *enhancements*, which improved TEX but were not obligatory. I consider both bugs and enhancements to be *errors*, for if I had designed a perfect system in the beginning I would not have made any of these changes and my error log would have been empty.

The most important lessons I learned can be summarized in the following theses, which my paper [6] defends and explains in detail:

1. TEX would have been a complete failure if I had merely specified it and not participated fully in its initial implementation. The process of implementation constantly led me to unanticipated questions and to new insights about how the original specifications could be improved.
2. TEX would have been much less successful if I had not used it extensively myself. In fact, when TEX was new I thought of 100 ways to improve it as I was typesetting 700 pages over a period of several months, at a nearly constant rate of one enhancement per 7 pages typed.[3]
3. TEX would have been much less successful if I had not put considerable effort into writing a user manual for it myself. The process of explaining the language gave me views of the system that I never would have perceived if I had merely designed it, implemented it, and used it.
4. TEX would have been much less successful if I had not scrapped the first system and written another system from scratch, after having the benefit of several years' hindsight.
5. TEX would have been much less successful if I had not had the voluntary assistance of dozens of people who regularly gave me feedback on how to improve everything. The network of volunteers eventually became worldwide, perhaps because I decided that TEX should be in the public domain.
6. I recommend that everybody keep an error log such as the one I kept for TEX. The amount of extra time required is negligible (less than 1%), and the resulting records help us to understand ourselves and our fallible natures.
7. The methodology of structured programming reduced my debugging time to about 20% of what it was under my habits of the 60s. Furthermore, structured programming gave me enough confidence in my code that I did not feel the need to test anything for six months, until the entire system was in place and ready for testing. Therefore I saved considerable time by not having to do any prototyping.
8. Although certain features of programming languages can justly be considered harmful, we should not expect that eliminating such features will eliminate our tendency to err. For example, 12 of my errors can be ascribed to misuse of **goto** statements [3]; but that accounts for only 1.4% of the total, and I also made mistakes when using **while**, **case**, **if-then-else**, etc.
9. TEX proved to be highly reliable and portable because it was subjected to a "torture test," which is quite different from anything a sane user would write but which really tries hard to make the system fail. We should strive energetically to find faults in our own work, even though it is much easier to find assurances that things are OK.

My experiences agree well with Peter Naur's hypothesis [7] that programming is "theory building," and they strongly support his conclusion that programmers should be accorded high professional status. But I do not share Naur's unproved assertion that "reestablishing the theory of a program merely from the documentation is strictly impossible." On the contrary, I believe that improved methods of documentation (which I have called "literate programming" [4, 2]) are able to communicate everything necessary for the maintenance and modification of programs. I think it's fair to claim that more than 100 people, perhaps more than 1000, now understand the "theory" of the TEX program after merely reading its documentation [5]. For I have seen numerous examples of electronic communications in which many people have demonstrated such knowledge by making excellent special-purpose extensions to the existing code and by giving highly appropriate advice to users.

Therefore I now look forward to making further errors in my next project.

---

[3] On the other hand, the new ideas ceased when I went on to type hundreds of additional pages; 700 was enough! I got most of the later suggestions from other people, and I was able to appreciate them because of my own experiences.

Bibliography

[1] Beeton, Barbara, ed. "TEX and METAFONT: Errata and Changes," (dated 09 September 11, 1983). Distributed with *TUGboat* 4, 1983.

[2] Bentley, Don. "Programming Pearls." *Communications of the ACM* 29:364–369, 471–483, 1986.

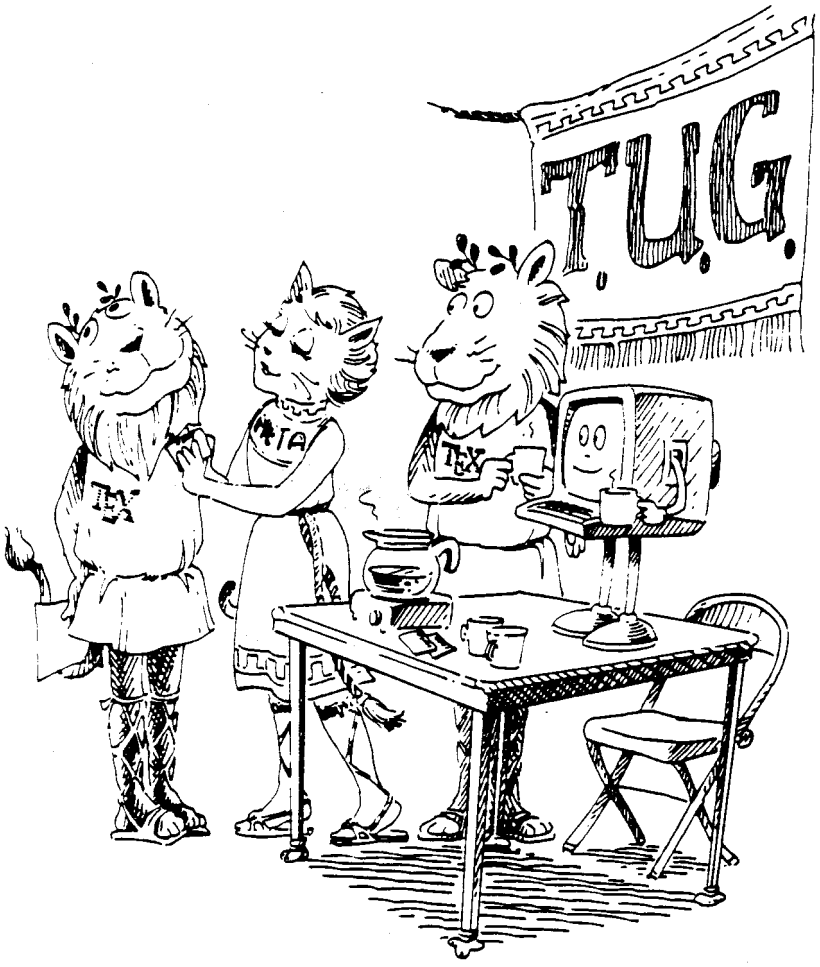
[3] Knuth, Donald E. "Structured Programming with **go to** Statements." *Computing Surveys* 6:261–301, December 1974. Reprinted with revisions in *Current Trends in Programming Methodology*, Raymond T. Yeh, ed., 1:140–194, (Englewood Cliffs, N.J.: Prentice-Hall, 1977); also in *Classics in Software Engineering*, Edward Nash Yourdon, ed., pp. 259–321, (New York: Yourdon Press, 1979).

[4] Knuth, Donald E. "Literate Programming." *The Computer Journal* 27:97–111, 1984.

[5] Knuth, Donald E. *TEX: The Program.* Reading, Mass.: Addison-Wesley, 1986.

[6] Knuth, Donald E. "The Errors of TEX." *Software Practice & Experience* 19:607–785, 1989.

[7] Naur, Peter. "Programming as Theory Building." *Microprocessing and Microprogramming* 15:253–261, 1985.



TEX Users Group

Tufts University, July 20–23, 1986

P. T. Barnum Auditorium