Alex Groce (agroce@gmail.com), Oregon State University

Edward R. Tufte's *The Visual Display of Quantitative Information* is so well known that it may need little introduction to most readers of this column. That it is a remarkable book, the most widely-read and beloved examination of "data graphics," is uncontroversial.

Why, however, is it particularly a book for software engineers? In one sense, it is not particularly for software engineers, since the qualities that make it useful to software engineers for the most part apply to all scientists and engineers. Why should every engineer or scientist read Tufte? Because we are often required to present complex data to each other, to non-technical audiences, and even to ourselves. Unfortunately, in most cases our training in computer science does not give us much experience in doing so, and the sophistication of data graphics in most textbooks is limited at best. This is particularly sad in an era when computers make the production of high-quality graphics easier than ever before. Of course, computers also make the production of terrible graphics easier than ever before, so the blessing is not unmixed.

Tufte offers guidance in how *not* to produce terrible graphics. While the text of the book is superb, much of Tufte's method is to present a vast, dominating (the book's substance, not the reader), sequence of large-sized examples of data graphics, mostly chosen for their excellence, occasionally chosen to demonstrate the way not to do things. The figures are eye-catching and presented at an excellent size for close examination (this is a large book that won't fit on a small shelf). They cover a historical period ranging from the Napoleonic wars, which inspired Minard's justly praised depiction of the terrible losses suffered during Napoleon's Russian campaign of 1812, up to infographics from the 1980s (we are, thankfully, spared anything from USA Today).

In fact, the way Tufte uses actual data graphics to discuss the construction of good graphics and avoidance of bad graphics suggests that software engineering could benefit from a book like this. It is pleasant to imagine *The Textual Display of Algorithmic Information*, a new textbook adopted by every university (or even every elementary school), which takes classic examples of what to do (and what not to do) in code, across a variety of purposes and languages, and wraps these examples up with the ribbon of a compelling examination like Tufte's. The readability and maintainability of code is analogous to visual appeal in data graphics; correct, efficient execution maps to the faithfulness of a graphic to the facts of the underlying data. A graphic can be beautiful but misleading, and code can be easy-to-read but buggy. Code can also be "correct" but nearly impossible to read, or ugly in concept (when a more elegant and concise approach would be equally effective). Even when our computer science textbooks do include code examples, they are usually contrived for the book, not historical examples with real context and a variety of developers. The examples are seldom faulty in an interesting way, and very seldom beautiful, striking, or even aesthetically acceptable (in the majority of texts). Tufte's book serves not only as an education in how to make data graphics, it serves as an education in how to read data graphics. It is not unreasonable to claim that learning to read diverse code would certainly benefit most students of software engineering.

Most of us spend more time looking at data graphics (or at lines of code) than we spend producing them.

Confession time: my own papers contained almost no graphics, for the first seven to eight years of my research career. In model checking, the normal state of affairs at the time was to use a table of detailed runtimes, memory usage, and so forth. Including a column with averages or percentage improvements was a wild and crazy innovation, and rather indicative of a second-rate mind. In fact, given the tiny number of data points available in the typical paper (and lack of statistical substance), it is not clear graphs would have been of much use to readers, even if present. Since then, I've used larger data sets, and produced more data graphics, as my research moved from model checking only to software testing and analysis using a wider variety of techniques (even my model checking papers tend to contain at least one data graphic, these days). My more recent papers, however, have hardly made use of the rich set of techniques Tufte presents. Most research papers outside of the visualization field itself (and perhaps human-computer interaction), in my experience, consist of scatterplots (with regression lines), box-and-whiskers plots, or simple bar charts that present the same kind of information as model checking research circa 2000-2010's ubiquitous tables. When data graphics are present in non-academic documents in software engineering, they tend to be of these same limited forms (but are probably rarer than in contemporary scientific papers in the field).

Does this mean Tufte isn't very useful to software engineers, after all? Does it at least mean that Tufte's many fans in the field aren't actually getting much out of his work other than a pleasant feeling that they may *aspire* to elegant displays of information, even if they never actually get around to producing them? No! It rather means that while the charming and unique nature of many of Tufte's examples is missing from most real-world usage in the field, the essential points in Tufte can be applied just as well to bread-and-butter information that is not inherently hard to display. Avoiding "chart junk" and using proper scaling, using effective labeling, and other issues are perhaps even *more* critical for simple graphics than more elaborate displays of many complex variables at once. When there is only simple data to display, obscuring it is a worse crime than when even an expert might fail to tell the story with a picture. Moreover, the growing importance of Geographic Information System (GIS) analysis in some subfields of computer science does require new sophistication in effective presentation of information, and there is no better place to start than Tufte.

The lesson, that error in complex systems is forgivable, but error in simple systems is a crime, can also apply to software. Most of us may not solve the most difficult problems of software design; we often produce systems of limited complexity, with high similarity to something we have built in the past. Nonetheless, lessons learned in the context of much more ambitious and clever systems, with harder-to-achieve goals, can help us avoid common mistakes in even the "simplest" work. A good carpenter may build a beautiful ornate mantlepiece; a great carpenter knows how to make a simple, sturdy table a work of art. Tufte is essential for being both good and great.