Editor: Diomidis Spinellis ■ Athens University of Economics and Business ■ dds@aueb.gr

The Tools at Hand

Diomidis Spinellis

The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities. —*Edsger W. Dijkstra*

With a shovel excavator, its operator can effortlessly move 720 tons of earth with a single movement. On the other end of the spectrum, a VLSI fabrication plant lets designers create elaborate submicron structures. Without tools, a car factory's thousands of employees can't accomplish much; with tools, they can assemble a car in 18 effort-hours. Sometimes, tools can even subsume their operators' importance. The violinist Ivry Gitlis, considered one of the most talented musicians of his generation, said of his Stradivarius: "I have a violin that was born in 1713. I don't consider it my violin. Rather, I am its violinist; I am passing though its life." Tools are clearly an important and defining element of any profession and activity: tools help us move boulders and atoms; tools help us reach the Moon and our soul.

This new *IEEE Software* column aims to explore the interplay between software development and the tools we apply to the problem. Skilled craftsmen set themselves apart from amateurs by the tools they use and the way they employ them. As a professional, I feel I'm getting a tremendous boost in my productivity by appropriately applying tools to the software construction problems I face every day. I also often find myself developing new tools, both for my personal use and for wider distribution. Column installments will discuss specific software construction activities from the standpoint of the tools we can employ—the tools of our trade. Specific topics I plan to address include editing, compiling, documentation, debugging, testing, configuration management, issue tracking, the development environment, tool building, and domain-specific tools. Of course, this is your column as much as it is mine, so I welcome your suggestions for different topics or viewpoints; email me at dds@aueb.gr.

Underspending on development tools

So, how do the tools of our trade measure up? Pathetically, by many measures. Although the software industry is large and, dare I say it, mature, the software tool industry is still in its infancy. This becomes readily apparent if we consider the cost of the tools we use. A 720 ton-rated shovel excavator is so expensive that the company selling it also provides financing. The cost of the VLSI fabrication plant effectively dictates the manufactured chips' product cycles. In comparison, software development tools cost at most a few thousands of dollars per seat. Economists track capital expenditures as a way to judge a country or sector's economic future. On the radar screen of these statistics, the cost of software development tools wouldn't amount to a single blip.

To substantiate the claim of capital underspending in our industry, I used Standard & Poor's COMPUSTAT global database to compare the capital expenditures of some industries we software engineers often admire and look to as role models against our own. Look at Table 1's

 0740-7459/05/$20.00 © 2005 IEEE

numbers. The semiconductor industry's capital expenditures amount to 23 percent of its revenue. This is how it has succeeded in following Moore's law for more than 30 consecutive years. The car industry's robotic factories, seen as patterns for emulation by proponents of software assembly plants, soak up 8 percent of its revenues. Even the nomadic heavy-construction industry—our perennial favorite when we compare software engineering to bridge building—spends on capital equipment nearly twice the percentage of revenues that our own custom software construction (programming services) firms spend.

I hear you saying that software's economies are different: we can duplicate software at a zero marginal cost, so the low cost of tools reflects the realities of their distribution rather than their intrinsic value. I only wish this was true—that we're all buying expensively developed tools at rock-bottom prices. I can vouch from experience that the effort our industry puts into developing software development tools is apparently miniscule. A couple of years ago I developed UMLGraph, a prototype of a declarative UML diagramming tool, and made it available over my Web site. I wrote the tool's first version over a single weekend, yet I regularly receive email from enthusiastic users. This fact definitely doesn't reflect on my programming brilliance, but says a lot about the state of the art in diagramming software and the amount of cash employers are willing to spend on purchasing diagramming (and conceivably other software development) tools.

What would happen if an established tool vendor with deep pockets decided to build a software development tool by investing the kind of money associated with a chip plant? (Mind you, I recognize the difference between chip *production* and software *design*; my argument concerns capital expenditures over the entire product life cycle.) According to Intel financier Arthur Rock, the cost of capital equipment to build semiconductors doubles every four years. Currently, a chip plant under construction costs over US$2.5 billion. To put this number in perspective, consider that it represents about 13,000 software development effort-years. This is almost three times the effort invested in the development of OS/360 (5,000 effort-years) and, according to my calculations, almost equal to the development effort of the Windows NT line, up to and including Windows 2000.

Investing this kind of money on a design tool could buy us round-trip, model-based software development that actually works under realistic conditions. If we invested this money into a compiler, we could get type-checking integrated across the presentation, application logic, and database layers or the ability to generate provably correct and efficient code. We could also have at our hands debuggers that can execute a program forward and backward; editors that let us navigate between diagrams and source code, effortlessly performing sophisticated refactoring operations; and infrastructure to test an application's GUI delivered as part of our integrated development environments.

To get a picture of the lag between what's theoretically possible and what tools provide in practice, scan the pro-

Table 1

Capital expenditures in different industries

Industry	Revenue (US$ billion)	Capital expenditure ($ billion)	CE/R (%)
Semiconductors	430,360	99,577	23.1
Motor vehicles	1,094,157	90,042	8.2
Heavy (nonbuilding) construction	143,957	6,187	4.3
Prepackaged software	105,356	3,402	3.2
Programming services	18,216	438	2.4

ceedings of the last five Programming Language Design and Implementation (PLDI) and International Conference on Software Engineering (ICSE) conferences. You'll see how few of the results reported there are now commercially available to developers for everyday use.

Underused tools

As if our underspending on software development tools wasn't worrisome enough, a related problem in our profession is our failure to use the most appropriate tools for a given task. Here's my list of 10 Software Tool Sins:

10. Maintaining the source code's API documentation separately from the source code.
9. Failing to integrate automated unit testing in the development process.
8. Using paper forms, email folders, and Post-it notes to track pending issues.
7. Painstakingly analyzing a source code change's effects manually when the compiler and the language's type system can do the job more reliably.
6. Refusing to learn how existing tools can be made to work together through scripting or a shell interface.
5. Ignoring or (worse) silencing compiler warning messages.
4. Maintaining isolated copies of the source code base for each developer, and performing version control and software configuration management using email attachments or those trendy USB dongles.
3. Locating definitions of program entities through a mixture of guesswork and sequential scanning through the source code.
2. Adding temporary print statements in the source code instead of using a debugger.
1. Performing mechanical, repetitive editing operations by hand.

I often spot mature developer colleagues committing the number one offense in the list by the sound of their keyboard: the click-clack-clack, click-clack-clack, click-clack-clack typing pattern gives them away. This sin is inexcusable, as (free) editors with sophisticated text-processing capabilities have been available for over 30 years. Other sins, such as number two, are admittedly a mixture of tool immaturity and developer laziness. The Linux 2.4 kernel contains 65,000 `printf` or `printk` statements, the FreeBSD kernel another 17,000. We can explain many of these statements by most debuggers' poor support for embedded and system software development—a shortcoming that's becoming increasingly important as more and more software is developed for embedded devices. In my experience, many other sinful habits can be traced back to our university days. Academia often regards the dirty mechanics of software development as a less-than-respectable activity. Software tools get in the way when teaching introductory programming courses, and would take valuable time away from discussing lofty theories when teaching software engineering. So, students are left on their own, many graduating and still writing their software with Windows Notepad.

There's really no need to end this column in a sullen mood. Software is a great lever. What little our industry has invested in tool development has provided us with numerous admirable and sophisticated tools. The many volunteers working on free and open source software projects are further increasing our choices for mature development environments and tools. It's up to us to make the best of what's available, and—why not—contribute back to the community.

Diomidis Spinellis is an associate professor in the Department of Management Science and Technology at the Athens University of Economics and Business and the author of Code Reading: The Open Source Perspective (Addison-Wesley, 2003). Contact him at dds@aueb.gr.