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

Cracking Software Reuse

Diomidis Spinellis

[Newton] said, "If I have seen further than others, it is because I've stood on the shoulders of giants." These days we stand on each other's feet! — Richard Hamming

Sometimes we encounter ideas that inspire us for life. For me, this was a Unix command pipeline I came across in the '80s:

```
tr -cs A-Za-z '\n' |
tr A-Z a-z |
sort |
uniq |
comm -23 - /usr/dict/words
```

This command will read a text document from its standard input and produce a list of misspelled words. It works by transforming all nonalphabetic characters into new lines, folding uppercase letters to lowercase, sorting the resultant list of words, removing duplicates, and finally printing those words that don't appear in the system dictionary.

Our cryptic pipeline is remarkably portable and efficient. By fixing the system dictionary's location, which has moved over the years, I successfully tested the pipeline on modern FreeBSD and Linux systems. Impressively, on one of those increasingly common multiprocessor machines, the pipeline utilized 1.25 of the two available processors—a feat, even by modern standards. However, when I first saw the pipeline, portability and multiprocessor utilization weren't on my radar screen. What impressed me was how five straightforward commands running on a relatively simple system that supported a few powerful abstractions could achieve so much.

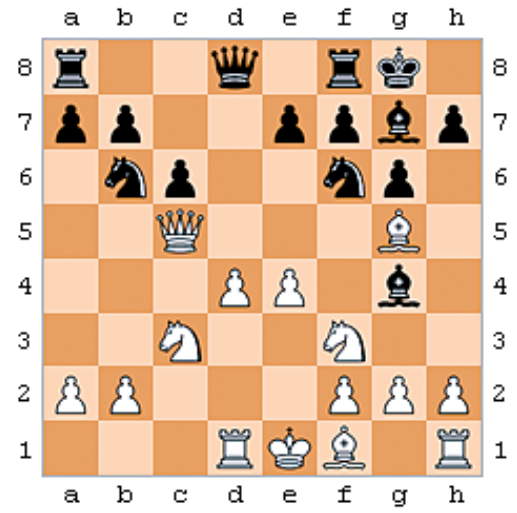
I hoped life in computer science would be a series of such revelations, but I was in for disappointment. Things appeared to be going downhill from then on. I saw systems become increasingly complex, the tools that first impressed me fall into disuse, and the concept of reuse preached more than practiced. A "hello world" program in the then shiny new X Window System or Microsoft Windows was a 100-line affair. I felt our profession had hit a new low when I realized that a particularly successful ERP (enterprise resource planning) system used a proprietary in-house developed database and programming language. It seemed Hamming was right.

An unexpected picture

Yet, progress moves in surprising ways. Nowadays, I'm proud of our achievements and optimistic about our future. Look at figure 1, depicting a position of what became known as the Game of the Century: a chess game played between Donald Byrne and 13-year old Bobby Fischer on 17 October 1956. Although the game was remarkable, so is the ecosystem behind the picture.

The picture on the left comes from the Wikipedia article on the game (http://en.wikipedia.org/wiki/The_Game_of_the_Century_(chess), as of 21 October 2006). To create it, one of the article's 65 contributors wrote the layout appearing on the figure's right, using a readable and concise domain-specific minilanguage. Despite what you might think, this chessboard description language isn't an inherent part of MediaWiki (Wikipedia's engine). Instead, it's a MediaWiki template: a parameterized, reusable formatting element. About a dozen people wrote this particular template, using MediaWiki's low-level constructs, such as tables and images.

 0740-7459/07/$20.00 © 2007 IEEE



```
{{Chess diagram small|=
| tright
|
|=

 8 |rd|  |  |qd|  |rd|kd|  |=
 7 |pd|pd|  |  |pd|pd|bd|pd|=
 6 |  |nd|pd|  |  |nd|pd|  |=
 5 |  |  |ql|  |  |  |bl|  |=
 4 |  |  |  |pl|pl|  |bd|  |=
 3 |  |  |nl|  |  |nl|  |  |=
 2 |pl|pl|  |  |  |pl|pl|pl|=
 1 |  |  |  |rl|kl|bl|  |rl|=
    a  b  c  d  e  f  g  h

| The position after 11. Bg5.
}}
```

Figure 1. A chess board diagram and its layout description.

Digging deeper, we'll find that MediaWiki consists of about 175,000 lines of PHP (PHP: hypertext preprocessor) code using the MySQL relational database engine. A rough count of C/C++ source code files in the PHP and MySQL distributions gives us 740,000 and 1.8 million lines, respectively. And underneath, we'll find many base libraries on which PHP depends, the Apache and Squid server software, and a multimillion-line-large GNU/Linux distribution. In all, we see a tremendously complex system that lets hundreds of thousands of contributors cooperatively edit two million pages—and still manages to serve more than 2,000 requests each second.

How we won the war

We must be doing something right. Having Wikipedia's software components freely available has surely helped, but there's more than that in our recent successes. One important factor is that we've (almost) sorted out the technology for reuse. Huge organized archives, pioneered by the Comprehensive TeX Archive Network (CTAN) and popularized by Perl's Comprehensive Perl Archive Network (CPAN), let us publish and locate useful components. The package management mechanisms of many modern operating systems have simplified the installation and maintenance of disparate components and their intricate dependencies. Programming languages now offer robust namespace management mechanisms to isolate the interactions between components. Shared libraries have matured, providing us with vital savings in memory consumption: on a lightly loaded system, I recently calculated that shared libraries saved 97 percent of the memory space that we'd need without them. Widely used platforms, such as Microsoft's .NET and the Java Platform Enterprise Edition, have also helped code reuse by integrating into their libraries everything but the kitchen sink. In all, the factors determining our return on investment from the components we reuse have moved in the right direction: modern components, like our chess description language, offer more and demand less.

A second important factor of our successes is the emergence of new types of collaboration. Version-control systems, bug-management databases, mailing lists, and wikis form the glue of modern large development teams. At the same time, code repositories, RSS feeds, automatic software update systems, and more mailing lists bring together component producers and consumers. Claiming that the Internet has revolutionized software development might sound farfetched, but we've got to remember that 20 years ago, systems of the size we've seen were developed only by NASA and large defense contractors, not volunteers working in their spare time.

Like many of my generation, one of my early sources of inspiration, predating the Unix pipeline, was Star Trek's USS Enterprise. I marveled its intricate technology but always wondered how it was built and maintained, especially when pieces of it got torn apart in battles. Clearly, the development model that gave us the Doric beauty of Unix and its tools couldn't be extended to cover the Enterprise's baroque complexity. I used to think that I would have to take that particular aspect of the Star Trek offering with a pinch of salt. Now I see that we computing professionals are developing an ecosystem where large, intricate systems can grow organically. And this is another truly inspiring idea.

Post your comments online by visiting the column's blog: **www.spinellis.gr/tools**

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 Quality: The Open Source Perspective* (Addison-Wesley, 2006). Contact him at dds@aueb.gr.