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

Project Asset Portability

Diomidis Spinellis

It's said that real computer scientists don't program in assembler; they don't write in anything less portable than a number two pencil. Joking aside, at the end of the 1970s, the number of nonstandard languages and APIs left most programs tied to a very specific and narrow combination of software and hardware. Entire organizations were locked in for life to a specific vendor, unable to freely choose the hardware and software where their code and data would reside. Portability and vendor independence appeared to be a faraway, elusive goal.

Fortunately, several standardization efforts pinned down languages like C, C++, and SQL as well as interfaces like Posix and ODBC. So, if you're careful and not too ambitious, you can now implement large systems that will run on several different platforms, such as Windows, Linux, and Mac OS X. The Apache Web server, Perl interpreter, and TeX typesetting system are prominent examples of this reality. Furthermore, Java—with its standardized application binary interface and rich set of class libraries—is now conquering many new portability fronts. I routinely move Java programs, Ant build files, and HSQLDB scripts between different operating systems and processor architectures without changing an iota.

However, these victories, impressive as they are, shouldn't distract us from the fact that we're fighting the last war. Nowadays, a software system's program source code makes up only a small part of its assets stored as bits—taking up a larger percentage are specifications, design diagrams, application server deployment scripts, build rules, version history, documentation, and regression tests. Figure 1 shows the relative size of four asset classes that I was able to easily measure in the FreeBSD operating system. The figure shows that the source code is actually third in size after the version control system's historical data and the outstanding and solved issues database. (The different assets shown in the figure are stored in textual format, so their sizes are directly comparable.) Notice that only the source code and the documentation—less than 25 percent of the total assets—are relatively portable between different tools. The version history and the issues are stored in tool-specific formats that hold the project hostage to their corresponding tools.

Where do you stand?

How is the situation in your organization? Can you easily move your design diagrams from one UML-based tool to another, change the repository where your specifications are stored, run your regression tests on a different platform, or switch to a different configuration management system? Unless your organization still uses paper records for these tasks (in which case your problems are in a totally different league), chances are you dread even the thought of changing the tools you use.

Yes, in modern organizations, tool flexibil-

0740-7459/06/$20.00 © 2006 IEEE

ity is becoming increasingly important. We can't afford to have a project's assets locked in proprietary data formats: software projects often change hands through corporate acquisitions and reorganizations, and we're increasingly outsourcing development. If a project's new home lacks the corresponding tools or engineers trained to use them, development will most likely continue on the lowest common denominator—the source code. All effort put into specifications, design, configuration management, and testing will be lost. A similar thing happened in the previous era of nonstandard languages. Organizations sometimes had to run program binaries on top of thick layers of emulation because the new hardware lacked the compilers and tools required for working with the source code. Those organizations faced a rude shock when they had to verify and fix their software for year-2000 compliance.

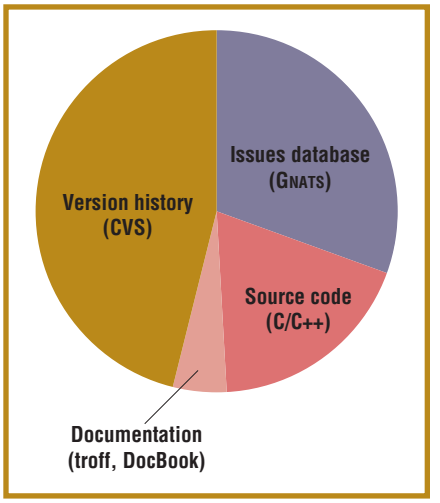

Figure 1. Relative size of different assets in the FreeBSD operating system project.

An open market

The *portability* of a project's nonsource code assets means more than allowing those assets to move freely between different organizations and developers. Portability also means that a marketplace for tools can evolve without the artificial restrictions of the vendor lock-ins imposed by incompatible data formats and associated switching costs. Such an environment would let different tools compete on their actual technical merits, without the artificially cozy protection of their installed base's captive audience. Furthermore, engineers could experiment with different tools to evaluate their merits, and, hopefully, exploit interoperability, using multiple tools to profit from their complementary strengths. Competition might also lower the cost of the corresponding tools, making them accessible to a wider community of users.

Remember the Unix wars? Many, now defunct, Unix vendors damaged themselves and their entire industry as they fiercely battled to lock in their customers with proprietary extensions. We don't want the software tool industry to suffer a similar ordeal.

Realizing the importance of the portability of a project's noncode assets just gets us out the door. Our first step should then be to inventory those assets to appreciate the problem's extent. Next, we should devise and standardize simple, powerful, and comprehensive formats for moving these assets between different tools. To minimize the effect of vendors' "embrace, extend, and extinguish" tactics, we should organize interoperability exhibitions, where vendors could compete on how well their tools work with each other. We should aim to make moving project data in its portable format as painless as editing the same text file with two different editors. As a counterexample, I understand that the interoperability of UML design tools through XMI (XML Metadata Interchange) is woefully inadequate right now—we must do better than that.

Won't the standardization I'm proposing put an end to tool innovation? By the time you read this column, Marc Rochkind's paper, "The Source Code Control System," will be 30 years old. However, modern configuration management systems don't differ radically from Rochkind's SCCS. Issue management systems are also competing on trivialities. Let's face it: a whole generation of tools has matured. So, it's high time to end gratuitous differences in the project's asset data schema and interchange format and let the tools compete on stability, usability, performance, and data manipulation capabilities. Databases, compilers, and Web browsers have all flourished under a regime of standardized interfaces; it's time to give our tools the same chance.

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.