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

Tool Writing: A Forgotten Art?

Diomidis Spinellis

Merely adding features does not make it easier for users to do things—it just makes the manual thicker. The right solution in the right place is always more effective than haphazard hacking. —Brian W. Kernighan and Rob Pike

In 1994, Shyam Chidamber and Chris Kemerer defined a set of six simple metrics for object-oriented programs. Although this number swelled to above 300 in the years that followed, I had a case where I preferred to use the original classic metrics for clarity, consistency, and simplicity. Surprisingly, none of the six open source tools I found for collecting metrics fit the bill. Most calculated only a subset of the metrics, had specific dependencies on other software, required tweaking to make them compile, or were horrendously inefficient. Although none of the tools correctly calculated the classic Chidamber and Kemerer metrics in a straightforward way most included numerous bells and whistles, such as graphical interfaces, XML output, and bindings to tools such as Ant and Eclipse.

As an experiment, I decided to create a tool to fit my needs. In the process, I discovered something important: writing stand-alone tools that you can combine efficiently with others to handle more demanding tasks appears to be becoming a forgotten art.

Going the Unix way

For my design ideal, I chose the filter interface found in most Unix-based tools. Unix tools are built following a set of simple design principles (see Brian W. Kernighan and Rob Pike's *The Unix Programming Environment* [Prentice Hall, 1984] and Eric Raymond's *The Art of Unix Programming* [Addison-Wesley, 2003]):

- Each tool is each responsible for doing a single job well.
- Functionality should be placed where it will do the most good.
- Tools generate textual output that other tools can use—for example, the program output won't contain decorative headers and trailing information.
- Tools can accept input generated by other tools.
- The tools are capable of performing stand-alone execution, without user intervention.

Tried and true

In general, these principles are easy to adopt. The 1979, Seventh Edition Unix `cat` command is only 62 lines long; the corresponding `echo` command is just 22 lines long. Despite their Spartan implementation, tools designed following these principles easily become perennial classics, which we can combine with others in powerful ways. For example, you can still use the 9-line, 30-year-old version of the Sixth Edition Unix `echo` command as a drop-in replacement in 5,705 places in the 2005 version of the FreeBSD operating system source code. You'd need the 26-year-old, slightly more powerful Seventh Edition version in another 249 instances. Nowadays, a suite of Unix tools is also widely available in open source implementations for systems such as Linux, Windows, BSD, and Mac OS X.

Something new

I created my metric tool, named ckjm (Chidamber and Kemerer Java Metrics), using the design principles I outlined earlier (see www.spinellis.gr/sw/ckjm). The tool operates on a list of compiled Java classes (or pairs consisting of an archive name followed by a Java class) specified as arguments or read from its standard input. It then prints to its standard output a single line for each class, containing the class name and the six metrics' values. This design lets you use pipelines and external tools to select the classes to process or to format the output (have a look at the tool's Web site for specific examples). Given ckjm's simplicity and minimal features (for example, I ignored irrelevant interfacing requirements), I wasn't surprised to find it both more stable and more efficient than the other tools I'd tried.

Temptation calls

A month after I put the tool on the Web, I received an email from a brilliant young Dutch programmer. He'd enhanced my tool, integrating it with the Ant Java-based build tool and adding an XML output option. He'd also provided a couple of XSL (Extensible Style Sheet Language) scripts that transformed the XML output into nicely set HTML. Although the code was well written and the new facilities appeared alluring, my initial reply wasn't exactly welcoming.

The perils of tool-specific integration

Integrating ckjm with Ant sounds like a good idea until you consider the dependencies this type of integration creates. With the proposed enhancements, the tool's source code imports six different Ant classes, creating a dependency between one general-purpose tool and another. What would happen if we also integrated ckjm with Eclipse and a third graphics-drawing software package? Through these dependencies, ckjm would become highly unstable: any interface change in any of the three tools would require a change in ckjm.

The functionality provided by the imported Ant classes is certainly useful; it gives us a generalized and portable way to specify sets of files. However, providing it in one tool violates the principle of adding functionality where it would do the most good. Many other tools would benefit from this facility, so it makes sense to put Ant's `DirectoryScanner` class into a more general tool or facility.

Ant interfaces usually provide services for performing tasks that most modern operating systems already support as general-purpose abstractions. These abstractions include a process's execution, specification of its arguments, and redirection of its output. Creating a different, incompatible interface for these facilities is not only gratuitous, it also relegates venerable tools developed over the last 30 years to second-class citizens. This approach simply doesn't scale. We can't require each tool to support every other tool's peculiar interfaces, especially when there are existing conventions and interfaces that have withstood the test of time. We gain a lot if the tools we implement—no matter if we implement them in C, Java, C#, or Pearl—follow the conventions and principles I outlined earlier.

The problems of XML output

Adapting a tool for XML output is less troublesome because XML solves some real problems. Unix tools' typeless, textual output can become a source of errors. If a tool's output format changes, tools further down the pipeline will continue to happily accept and process their input assuming that it follows the earlier format. We'll only realize that something is amiss if we see that the final results don't match our expectations. Additionally, we can represent only so much using space-separated fields and line-oriented records. XML lets us represent more complex data structures in a generalized and portable way. XML also lets us use powerful, general-purpose verification, data query, and data manipulation tools.

However, because XML intermixes data with metadata and abandons the simple, textual, line-oriented format, it shuts out most tools in a Unix programmer's tool bench. XSL transformations might be powerful, but because they're implemented within monolithic all-encompassing tools, any nonsupported operation becomes exceedingly difficult to implement. Using Unix tools, if we want to perform a topological sort on our data to order a list of dependencies, `tsort` lets us do exactly that; if we want to spell check a tool's output, we can easily do it by adding the appropriate commands to our pipeline.

Moreover, the implementation of XML-based operations appears to be orders of magnitude more verbose than the corresponding Unix commands. As an

Call for participation

13th IEEE Int. Requirements Engineering Conference

http://www.re05.org

August 29th – September 2nd 2005, **Paris, La Sorbonne**, France

Engineering Successful Products

Steering Committee Chair:
Roel Wieringa
U. Twente, Netherlands

General Chair:	**Program Chair:**
Colette Rolland	Joanne Atlee
U. Paris 1-Panthéon Sorbonne, France	*U. of Waterloo, Canada*

RE'05 will emphasize the crucial role that requirements play in the successful development and delivery of systems, products, and services.

Technical and Industrial program: The main conference will run from August 31 to September 2 and will feature parallel sessions of technical papers, mini-tutorials, and research-tool demonstrations.
Workshops, Tutorials and a Doctoral Symposium will be held August 29-30.

The list of the **twelve workshops** and **nine tutorials** is available at the RE'05 web site : http://www.re05.org/
Deadlines for workshop papers range from end of May to mid-June. See the conference web site for details.

Keynote speakers:
Jean-Pierre Corniou, CIO at Renault, author of "The Knowledge Society", Hermès, 2002, 1999 IT Manager of the Year award from the French press, president of the CIGREF (Club Informatique des Grandes Entreprises Françaises)
Daniel Jackson, Professor of Computer Science at MIT, Chair of The National Academies study on "Sufficient Evidence? Building Certifiably Dependable Systems"
Suzanne Robertson, Independent consultent, author of 'Mastering the Requirement Process' (Addison-Wesley, 1999) and Requirement-Led project Management (Addison-Wesley, 2005), co-founder of the Atlantic Systems Guild

RE'05 Organizing Committee hopes to meet you in Paris to share experiences in requirement engineering, as well as to appreciate our famous French cooking and good wine.

Register at www.re05.org **Early bid until July 31st 2005**

experiment, I asked an experienced colleague to rewrite an `awk` one-liner I used for finding Java packages with a low abstractness and instability value into XSL. The 13-line code snippet he wrote was certainly less cryptic and more robust than my one-liner. However, in the context of tools we use to simplify our everyday tasks, I consider the XSL approach unsuitable. We can write a one-liner as a prototype and then gradually explore how to enhance it. Writing 13 lines of XSL isn't a similarly lightweight task.

Although adding XML output to a tool might seem enticing, it's the first step down a slippery slope. If we did add direct XML output (ckjm's documentation already listed a 13-line `sed` script to transform its output into XML), why not allow the tool to write its results into a relational database via Java Database Connectivity (JDBC)—surely we'd end up with a more robust and efficient result than by combining existing tools. After that, what about adding a database configuration interface, chart output, a GUI environment, a report designer, a scripting language, and, who knows, maybe the ability to share the results over a peer-to-peer network?

Realpolitik

With all that said, the Ant integration and XML output will be part of ckjm by the time you read these lines—probably as optional components. Ralph Waldo Emerson famously wrote that "A foolish consistency is the hobgoblin of little minds." Spreading ideology by alienating users and restricting a tool's appeal sounds counterproductive to me. Nevertheless, the next time you ask for tighter integration or a richer I/O format for a tool, consider if you can accomplish what you're asking for in a more general fashion and what this new feature will cost your environment in terms of stability, interoperability, and orthogonality.

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.