DOI:10.1145/2347736.2347752

Article development led by acmqueue
queue.acm.org

Quality happens only when someone is responsible for it.

BY POUL-HENNING KAMP

A Generation Lost in the Bazaar

THIRTEEN YEARS AGO, Eric Raymond's essay, "The Cathedral and the Bazaar,"[2] redefined our vocabulary and all but promised an end to the waterfall model and big software companies, thanks to the new grass-roots open source software development movement. I found the book thought-provoking, but it did not convince me. On the other hand, being deeply involved in open source, I couldn't help but think that it would be nice if he was right.

The book I brought to the beach house this summer is also thought-provoking, much more so than Raymond's book (which it even mentions rather positively): Frederick P. Brooks's *The Design of Design*.[1] As much as I find myself nodding in agreement and as much as I enjoy Brooks's command of language and subject matter, the book also makes me sad and disappointed.

Thirteen years ago also marks the apogee of the dot-com euphoria, where every teenager was a Web programmer and every college dropout had a Web startup. I had genuine fun trying to teach some of those greenhorns about the good old-fashioned tricks of the trade—test-restoring backups, scripting operating-system installs, version control, and so on. Hindsight, of course, is 20/20 (that is, events may have been less fun than you remember), and there is no escaping that the entire dot-com era was a disaster for IT/CS in general and for software quality and Unix in particular.

I have not seen any competent analysis of how much bigger the IT industry became during the dot-com years. My own estimate is that—counted in the kinds of jobs that would until then have been behind the locked steel doors of the IT department—our trade grew by two orders of magnitude, or if you prefer, by more than 10,000%.

Getting hooked on computers is easy—almost anybody can make a program work, just as almost anybody can nail two pieces of wood together in a few tries. The trouble is that the market for two pieces of wood nailed together—inexpertly—is fairly small outside of the "proud grandfather" segment, and getting from there to a decent set of chairs or fitted cupboards takes talent, practice, and education. The extra 9,900% had neither practice nor education when they arrived in our trade, and before they ever had the chance to acquire it, the party was over and most of them were out of a job. I will charitably assume that those who managed to hang on were the most talented and most skilled, but even then there is no escaping that as IT professionals they mostly sucked because of their lack of ballast.

The bazaar meme advocated by Raymond, "Just hack it," as opposed to the carefully designed cathedrals of the pre-dot-com years, did, unfortunately, not die with the dot-com madness, and today Unix is rapidly sinking under its weight.

I updated my laptop. I have been running the development version of FreeBSD for 18 years straight now, and compiling even my Spartan work environment from source code takes a full day, because it involves trying to make sense and architecture out of Raymond's anarchistic software bazaar.

At the top level, the FreeBSD ports collection is an attempt to create a map of the bazaar that makes it easy for FreeBSD users to find what they need. In practice this map currently consists of 22,198 files that give a summary description of each stall in the bazaar—a couple of lines telling you roughly what that stall offers and where you can read more about it. Also included are 23,214 Makefiles that tell you what to do with the software you find in each stall. These Makefiles also try to inform you of the choices you should consider, which options to choose, and what would be sensible defaults for them. The map also conveniently comes with 24,400 patch files to smooth over the lack of craftsmanship of many of the wares offered, but, generally, it is lack of portability that creates a need for these patch files.

Finally, the map helpfully tells you that if you want to have www/firefox, you will first need to get devel/nspr, security/nss, databases/sqlite3, and so on. Once you look up those in the map and find their dependencies, and recursively look up their dependencies, you will have a shopping list of the 122 packages you will need before you can get to www/firefox.

Modularity and code reuse is, of course, A Good Thing. Even in the most trivially simple case, however, the CS/IT dogma of code reuse is totally foreign in the bazaar: the software in the FreeBSD ports collection contains at least 1,342 copied and pasted cryptographic algorithms.

If that resistance/ignorance of code reuse had resulted in self-contained and independent packages of software, the price of the code duplication might actually have been a good trade-off for ease of package management. But that was not the case: the packages form a tangled web of haphazard dependencies that results in much code duplication and waste.

Here is one example of an ironic piece of waste: Sam Leffler's graphics/libtiff is one of the 122 packages on the road to www/firefox, yet the resulting Firefox browser does not render TIFF images. For reasons I have not tried to uncover, 10 of the 122 packages need Perl and seven need Python; one of them, devel/glib20, needs both languages for reasons I cannot even imagine.

Further down the shopping list are repeated applications of the Peter Principle, a belief that in an organization where promotion is based on achievement, success, and merit, that organization's members will eventually be promoted beyond their level of ability. The principle is commonly phrased, "Employees tend to rise to their level of incompetence." Applying the principle to software, you will find that you need three different versions of the Make program, a macroprocessor, an assembler, and many other interesting packages. At the bottom of the food chain, so to speak, is libtool, which tries to hide the fact that there is no standardized way to build a shared library in Unix. Instead of standardizing how to do that across all Unixen—something that would take just a single flag to the `ld(1)` command—the Peter Principle was applied and made it libtool's job instead. The Peter Principle is indeed strong in this case—the source code for devel/libtool weighs in at 414,740 lines. Half that line count is test cases, which in principle is commendable, but in practice it is just the Peter Principle at work: the tests elaborately explore the functionality of the complex

/*You are not expected to understand this*/

```
## Whether `make' supports order-only prerequisites.
AC_CACHE_CHECK([whether ${MAKE-make} supports order-only prerequisites],
  [lt_cv_make_order_only],
  [mkdir conftest.dir
   cd conftest.dir
   touch b
   touch a
cat >confmk << 'END'
a: b | c
a b c:
        touch $[]@
END
   touch c
   if ${MAKE-make} -s -q -f confmk >/dev/null 2>&1; then
     lt_cv_make_order_only=yes
   else
     lt_cv_make_order_only=no
   fi
   cd ..
   rm -rf conftest.dir
])
if test $lt_cv_make_order_only = yes; then
  ORDER='|'
else
  ORDER=''
fi
AC_SUBST([ORDER])
```

solution for a problem that should not exist in the first place. Even more maddening is that 31,085 of those lines are in a single unreadably ugly shell script called configure. The idea is that the configure script performs approximately 200 automated tests, so that the user is not burdened with configuring libtool manually. This is a horribly bad idea, already much criticized back in the 1980s when it appeared, as it allows source code to pretend to be portable behind the veneer of the configure script, rather than actually having the quality of portability to begin with. It is a travesty that the configure idea survived.

The 1980s saw very different Unix implementations: Cray-1s with their 24-bit pointers, Amdahl UTS mainframe Unix, a multitude of more or less competently executed SysV+BSD mashups from the minicomputer makers, the almost—but not quite—Unix shims from vendors such as Data General, and even the genuine Unix clone Coherent from the paint company Mark Williams.

The configure scripts back then were written by hand and did things like figure out if this was most like a BSD- or a SysV-style Unix, and then copied one or the other Makefile and maybe also a .h file into place. Later the configure scripts became more ambitious, and as an almost predictable application of the Peter Principle, rather than standardize Unix to eliminate the need for them, somebody wrote a program, autoconf, to write the configure scripts.

Today's Unix/Posix-like operating systems, even including IBM's z/OS mainframe version, as seen with 1980 eyes are identical; yet the 31,085 lines of configure for libtool still checks if <sys/stat.h> and <stdlib.h> exist, even though the Unixen, which lacked them, had neither sufficient memory to execute libtool nor disks big enough for its 16MB source code.

How did that happen?

Well, autoconf, for reasons that have never made sense, was written in the obscure M4 macro language, which means the actual tests look like the example in the accompanying figure.

Needless to say, this is more than most programmers would ever want to put up with, even if they had the skill, so the input files for autoconf happen by copy and paste, often hiding behind increasingly bloated standard macros covering "standard tests" such as those mentioned earlier, which look for compatibility problems not seen in the past 20 years.

This is probably also why libtool's configure probes no fewer than 26 different names for the Fortran compiler my system does not have, and then spends another 26 tests to find out if each of these nonexistent Fortran compilers supports the -g option.

That is the sorry reality of the bazaar Raymond praised in his book: a pile of old festering hacks, endlessly copied and pasted by a clueless generation of IT "professionals" who would not recognize sound IT architecture if you hit them over the head with it. It is difficult to believe today, but under this embarrassing mess lies the ruins of the beautiful cathedral of Unix, deservedly famous for its simplicity of design, its economy of features, and its elegance of execution. (*Sic transit gloria mundi* etc...etc...)

One of Brooks's many excellent points is that quality happens only if somebody has the responsibility for it, and that "somebody" can be no more than one single person—with an exception for a dynamic duo. I am surprised that Brooks does not cite Unix as an example of this claim, since we can pinpoint with almost surgical precision the moment that Unix started to fragment: in the early 1990s when AT&T spun off Unix to commercialize it, thereby robbing it of its architects.

More than once in recent years, others have reached the same conclusion as Brooks. Some have tried to impose a kind of sanity, or even to lay down the law formally in the form of technical standards, hoping to bring order and structure to the bazaar. So far they have all failed spectacularly, because the generation of lost dotcom wunderkids in the bazaar has never seen a cathedral and therefore cannot even imagine why you would want one in the first place, much less what it should look like. It is a sad irony, indeed, that those who most need to read it may find *The Design of Design* entirely incomprehensible. But to anyone who has ever wondered if using m4 macros to configure autoconf to write a shell script to look for 26 Fortran compilers in order to build a Web browser was a bit of a detour, Brooks book offers well-reasoned hope that there can be a better way.

Related articles on queue.acm.org

Open vs. Closed: Which Source is More Secure?
Richard Ford
http://queue.acm.org/detail.cfm?id=1217267

The Hyperdimensional Tar Pit
Poul-Henning Kamp
http://queue.acm.org/detail.cfm?id=2108597

Broken Builds
George Neville-Neil
http://queue.acm.org/detail.cfm?id=1740550

References

1. Brooks, F. *The Design of Design*. Addison-Wesley Professional, 2010.
2. Raymond, E. *The Cathedral and the Bazaar*. O'Reilly Media, Sebastapol, CA, 1999.

Poul-Henning Kamp (phk@FreeBSD.org) has programmed computers for 26 years and is the inspiration behind bikeshed.org. His software has been widely adopted as under-the-hood building blocks in both open source and commercial products. His most recent project is the Varnish HTTP accelerator, which is used to speed up large Web sites such as Facebook.