by Cameron Laird

ROBERT BURGER

Balancing Act for SCM

In upcoming columns I look at several domains–network management, geographic information systems (GIS), which make and annotate maps of the real world, and so on–where proprietary systems dominated during the '90s. Has open source made a difference in these areas? Does proprietary software continue to lead the markets, or are new development models such as open source "shaking up" the products available to end-users? Think about software configuration management (SCM) products, for example; is the top SCM software open source or closed?

Maybe neither. That's the reality Larry McVoy, founder of BitMover (`http://www.bitmover.com`) is out to create. Not only does he aim for his company's BitKeeper (BK) product to be the best-engineered SCM, he also wants to create a new business model for software. BitMover boasts a high ratio of interesting industry stories to employees. Although it has only a handful of engineers, their aggregate experience is deep. The second half of this column profiles the BK software they support and enhance. Before turning to that, though, it's important to understand BM's business model, and a little bit of the history that brought them this far.

McVoy thinks software has the potential to benefit three constituencies:

- Producers
- Consumers who are willing to pay ("paying customers")
- Consumers who are not willing to pay ("free customers")

Plenty of businesses and individuals are happy to pay for software. The benefit the software gives them–attractively printed reports from existing information stores, for example–exceeds the license fee and all other costs involved in use of the software. These are the people who funded the "shrink-wrap" explosion of the '80s and '90s.

Buyers and sellers traditionally have a stake in a product. Software is special in that people neither giving nor receiving money can still be important to the health of the market. Free customers contribute not money, but vital engineering information back to the producers. Closed-source development of proprietary products puts money in the treasuries of producers, and delivers value to paying customers. It does nothing for free customers. McVoy argues, moreover, that closed source "leaves money on the table"–it thwarts potential contributions from volunteers, for example. While conventional wisdom has it that open-source volunteers engineer patches and enhancements, McVoy claims their greatest contribution is in testing. "No testing department can ever hope to match the testing that happens as a result of one post to freshmeat.net," says McVoy. It is not unheard of to have 100,000 downloads in a matter of days after such a post.

Open source is also sub-optimal. McVoy says no organization which derives its revenue solely from open source has ever invented anything truly new. By his analysis, "For every good idea that comes out, there are hundreds

or thousands that end up in the trash can. Someone has to pay for the bad ideas and open source simply doesn't have the margins to do so." McVoy recognizes his point of view is unpopular with open-source zealots. He insists, though, that open source simply can't capture enough revenue to fund the labor involved in conceiving, developing, and polishing a product the way paying customers expect it to. Although open source creates tremendous value for free customers, non-developers don't receive the packages they want and understand.

SCM is like trash disposal. Although there's no glamour to it, it's valuable–valuable enough to inspire plenty of hollering when it doesn't work right.

McVoy has meditated at length on these different models. He has an analysis of open vs. closed source that's beyond the scope of this column. In fact, one way to think about BitMover is as a realization of his ideas on development models.

In particular, one of BitMover's aims is to explore a part of "business-model" space that McVoy considers more promising. He's out to combine the engineering excellence of open source with the consumer values and business sustainability of proprietary products. McVoy calls his hybrid "business source."

Profits with the BKL

The BitKeeper License (BKL) and associated technical mechanisms are his vehicle for reaching that happy marriage. Source code for his company's products is available; engineers can comment on and correct it. He also provides for modification and redistribution of source code, just as with such open-source licenses as the GNU Public License (GPL) and Perl's Artistic License. The BKL "hooks" customers, though, by requiring changes either to pass specific regression tests, or to be logged publicly. Customers who don't want to do their business in the open pay for a more traditional license. Their fees finance the maintenance of this "mixed use business model."

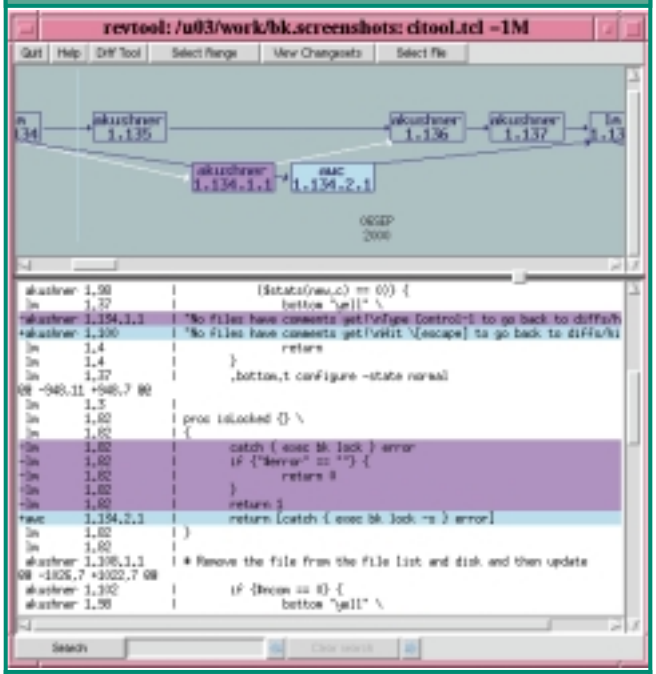
It's an interesting construction. McVoy has applied lessons he's learned at such high-profile employers as Lachman, SCO, Sun, SGI, Cobalt, and Google to launch what he intends as a fresh way to produce software. He claims a certain level of success already; through careful control of BM's growth, it's been a profitable company since 1999. Read the details of the BKL at `http://www.bitkeeper.com/4.4.2.html`.

Zero-price software still appears to be the asymptote our industry is approaching. Software copy protection, dongles, and even the fanciest micropayment schemes cooked up by IBM and other industry heavyweights seem like feeble jokes, or, at best, specialty items. Will BM reverse this trend? I don't know. It appears to be sustainable in its own business, though, and that constitutes newsworthy and healthy competition for both open- and closed-source companies.

Put aside for a moment BitMover, the business model. Is BitKeeper useful as a software product?

It's an engineer's dream.

Figure 1. revtool



revtool browses the history of a project or file. This screenshot captures part of the history of BK itself. Button clicks give immediate access to such functionality as diff comparisons between selected revisions, and display of check-in comments.

Deluxe Source Control

SCM is a bit like trash disposal. Although there's no glamour to it, it's valuable–valuable enough to inspire plenty of hollering when it doesn't work right. BK does a lot that's right. That reflects, in part, the background of its engineers, mostly "ex-kernel hackers," in McVoy's characterization. SCM companies rarely attract or retain that kind of talent, because most programmers don't find the domain "sexy" enough.

The main SCM products pertinent to *Server/Workstation Expert* readers are a few old-line no-charge UNIX-oriented products, including SCCS, RCS, and CVS, and pricey alternatives from companies such as Continuus, Perforce, Rational, and so on. The latter all present graphical user interface (GUI) views for ease of use.

Good SCM is a hard problem. Manipulation of source code is what developers do for eight or more hours daily, and any faults in SCM are sure to cause pain in at least some working situations. BK seems to bring more pleasure than pain.

Commercial SCM is notorious for its high "activation energy." Vendors generally recommend training and often dedication of one full-time administrator. Real-life installations often take a week of preparation before users see their code moving into and out of the system correctly. Quite apart from

licensing fees, it can take thousands of dollars of invested time to understand a specific SCM well enough to judge its fitness for a particular organization.

BK avoids that hazard. One BK Web page explicitly mentions "the five-minute test" of whether a product can do something useful in the first fraction of an hour after installation. BK can. It's easy to do the sensible things engineers generally want to do with SCM. While BK is compatible with traditional UNIX-style SCM commands like SCCS and RCS in its syntax, it builds in such modern conveniences as an extensive help system (available at `http://www.bitkeeper.com/5.1.html`) and responsive GUI screens (see Figures 1, 2, and 3).

Scalable and Comprehensive

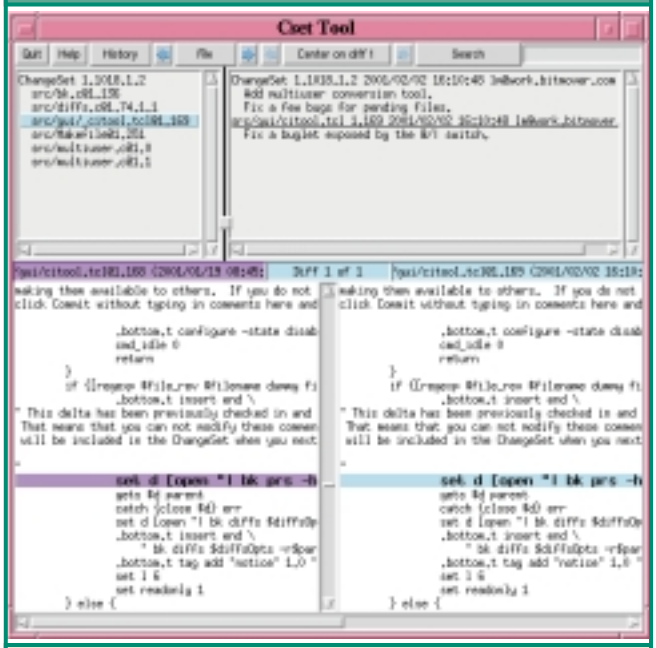
BK pays off even bigger after the first five minutes. The unit of BK transactions is the "changeset." This has enormous advantages over the file-oriented CVS model. Suppose, for example, that you change the signature of one C-coded function, defined in one source file, and invoked in another. With CVS and comparable managers, you check in changes to two different files, and there's no way to enforce synchrony between the changes. The inevitable result is that someone has to write "by hand" that "any f1.c after version 1.53 needs f2.c of version 2.18 or later." By contrast, BK "checks in" changesets. BK manages transactions so that it's natural to "roll back" change 710, which simultaneously reverts both f1.c and f2.c.

Moreover, BK keeps complete information about its changesets, and doesn't discard "merged data." This means with BK it's easy to start with a baseline of, say, change 700, "roll in" changes 710, 711, and 714, and roll out 694. The effect is to create a set of source files with specific, properly synchronized corrections and enhancements. While that's the promise of all SCM products, most of them stumble when they encounter complex changes involving different developers working on multiple, overlapping files. BK gives correct results for all such operations. (For more of the technical underpinnings, see "Tree Tagging," Page 44.)

Not only does BK give correct results, it gives them quickly. McVoy and his staff have gone to the trouble of testing specific operations other products also do. While all the comparisons he related to me were favorable to BK, I agree with him that they're appropriate and meaningful. As another BM Web page correctly says, "The problem with most configuration management systems is they don't scale. They all work great for 1-5 developers, but they tend to fall apart when you have 1,000 developers. BitKeeper's architecture is inherently scalable, so what works for five developers works equally well for 1,000 or 10,000." Complex merges that take several minutes with other products take BK seconds, in my own tests.

There's more to BK's scalability. The uniformity of BK's changeset model makes it practical to define subteams of developers, give them their own subrepositories, and manage their checkins separately from a main line of development. One of the benefits of the tutorial is its instruction in such novel processes. Along with these complex project hierarchies that are beyond the grasp of other SCM products, the tutorial even describes effective peer-to-peer (P2P) development models.

Figure 2. csettool



csettool browses changesets.

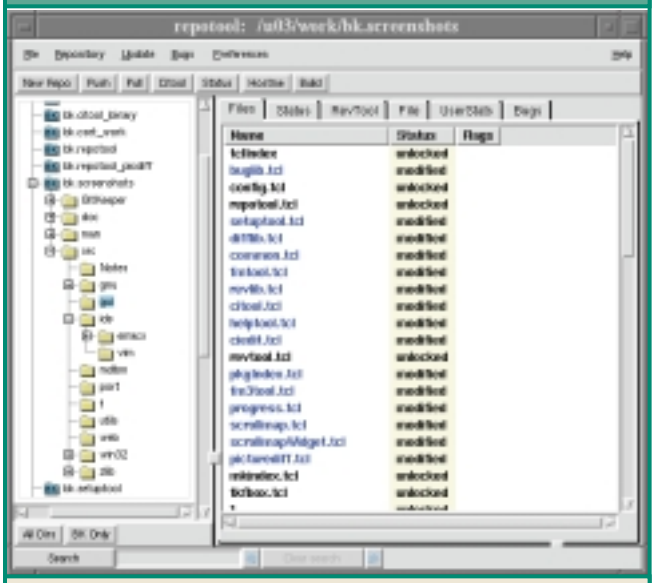
Transport Neutrality

BK's communication and security features support yet another aspect of scalability. A typical project in the year 2001 might have three engineers in an office in California, another one or two on the U.S. East Coast, and several others scattered over Europe, some with sporadic network connections. BK makes it easy to maintain local repositories, synchronized by any of such transports as `ssh`, SMTP, and HTTP. Among other things, this makes BK practical in heavily and heterogeneously firewalled situations. Developers concentrate on the source code, as they should, and BK takes care of the details of managing all the interactions. This is why I call BK an engineer's dream–it does all the things I knew 20 years ago I wanted SCM to do for me.

So, should you be using BK? I'm still not sure. CVS has an enormous "penetration" in the open-source world, and it is hard overcoming such a leader. The other commercial products also boast plenty of features, and I have enough experience with several of them to know how many problems they solve. Also, BK has one serious conceptual problem, a consequence of what I call "semantic ignorance." BK's current changeset model preserves time sequences. If change B happened after change A, all rollbacks and rollforwards with B necessarily have A. The result is the creation of "false dependencies" in large projects with many unrelated files. Think, for example, of the entire FreeBSD CVS repository, which includes source for thousands of different applications. An engineer might want to work only with device-driver enhancements; because these are interleaved in time with corrections to the authentication source, though, BK labors to "linearize" changes. This forces people who want the

authentication changes to also take the device-driver enhancements. Although there's no reason to believe BK introduces any errors in this situation, it's not as convenient and responsive as one might imagine.

Figure 3. repotool



repotool is like a master project integrated development environment (IDE) browser. It ties together all the other tools. This view displays the state of different source files: locked, modified, and so on. A right button-click opens a source file within an editor of your own choice.

McVoy says his engineering team already has designed a correction for this situation of large, weakly-coupled repositories. He expects to implement the fix no later than release 3.0 of BK. I modularize my work in a way to avoid this blemish. If BK continues to hold up in my tests, it's what I'll choose. Its high-performance, correct "changeset" semantics, and transport flexibility solve the problems that plague competing SCM products. Future installments of "Open Sources" may return for a more detailed look at BK. I suspect its scriptable "triggers" and their use in policy automation will make a particularly apt follow-up subject.

Reference and Acknowledgments

For more info about SCM, start with "Dave Eaton's Software Configuration Management Index" at `http://www.daveeaton.com/scm/`. If you are active in open-source development, you'll want to make a modest investment in the *CVS Pocket Reference*: `http://www.oreilly.com/catalog/cvspr/`.

Thanks to Aaron Kushner of BitMover for his help with BK. Also, special thanks this month go to Leam Hall and Bryan Oakley. Bryan, famous for the high-quality GUI widgets he's given away to the Tcl/Tk programming community, first recommended BitMover as an interesting story. After months of quiet productivity, Phaseit somehow had hardware faults in five (!) different hosts in a two-week interval. Leam rushed a replacement desktop allowing this column to make publication.

***Cameron Laird** is vice president of Phaseit Inc. (`http://www.phaseit.net`). Like McVoy, he had a string of successive development jobs which all involved a necessary detour through setting up a "homegrown" configuration management system. Unlike McVoy, he did not set up a company to productize the lessons he learned.*

Tree Tagging

One way to look at BitKeeper is as a triumph of wise data structures. Its performance on some of the complex operations working developers typically want is flabbergasting. How does it do so well?

One of BK's secrets, in McVoy's words, "is that each change represents not only *what changed* but the *context* in which it changed. Another way to say that is that each changeset is a snapshot of the whole tree." The state of a project at changeset 3.700 includes not only all changes in 3.700, but also all files *not* changed in 3.700.

SCM vernacular recognizes this reality in talk of "tagging the tree." Most SCM products can roll backwards only to points that have been distinguished by a separate operation of "tagging the tree," which identifies a well-defined state. What happens in practice is that trees aren't tagged. Doing so is cumbersome and unrewarding. Because it isn't done, rollbacks are fragile and approximate. This slashes the power of SCM down to the rather feeble "version control" that most organizations actually practice.

BK's trees are *always* tagged–automatically. And it happens nearly for free, because the structures that capture state are so apt. BK tags in msecs, even for large trees. Other systems can take minutes, as they tag by walking through the file system. BK keeps even tag or context information in its own repository, so it can be blindingly fast.

I've worked with SCM products that make ingenious shortcuts. When I first heard about BK, I assumed it was vulnerable to the data corruption I experienced with competing products. For an ancient analogue of the "Windows vs. Linux" battles, ask RCS and SCCS users about the advantages of forward vs. reverse differences. Earlier, less reliable hard-disk drives inflamed engineers' passions about such algorithmic differences, if only because they'd experienced the misery of restoring corrupted revision histories.

BK, though, apparently isn't so delicate. In fact, BK's design deliberately incorporates redundancy that protects it from file system corruptions to the point that it detects and resists errors in the network file system (NFS).