An Update on Software Updates

David J. Brown, *Queue* Editorial
Advisory Board Member

Editor Ed Grossman passed me the pen this month to tell you about our topical focus on software updates. When I raised the idea at the *Queue* editorial advisory board meeting several months ago, it was because I think the way that software is now being delivered to us is quite interesting. Things have changed. Nowadays you much less often expect to “install” whole systems or indeed even individual applications. Instead you’ve probably become quite used to the idea of accepting ongoing updates to what you already have—a behavior of happily receiving incremental changes to your software systems and/or applications by what I like to describe as the process of “bit dripping” from vendors over the Web.

Clearly the access and deployment metaphor that the Web provides has been an important enabler for this. Given the relatively high-speed communications infrastructure most of us now have (DSL, cable modems, etc.), we can expect good-sized lumps of code to come over the wire to us expediently. So you’re probably already quite accustomed to these new software delivery mechanisms.

But have you stopped to think about how all this works, and where we are going as things move forward? Will we soon be in a situation where our software systems and applications are almost totally “liquid”? You can already see some signs of this. Certainly there are some challenges, and I’ll bet also some interesting problems. So what are those issues? Here’s how I framed it to the *Queue* board at the time:

Today’s software systems and applications are often complex compositions of many hundreds or thousands of component parts. These constituent parts are each evolving, but at different rates. How does one update such a software system—whether to fix bugs, add new functionality, and/or to work within different operating environments—in a timely, convenient, and palatable way (e.g., without destabilizing its existing use and users)?

I thought you might find it interesting to hear from a number of folks involved in both building and using update-based mechanisms about how their world has changed: What does it take to build and deliver software this way, *and* what does it look like to receive and main-

The way

SOFTWARE IS
DELIVERED HAS
CHANGED.

tain systems given this approach.

When we began talking to some experts in this area, there was an immedi-

ate attraction to the topic of patching and patch management. Of course, a key focus for “updates” is as a vehicle for fixing bugs (“defect” management) and/or to address security liabilities. But I have to admit to being somewhat surprised by the degree of gravitation to that problem.

I’m also very interested in software updates as a means to deliver new functionality incrementally. You know, the kind of thing that we tend to see in the open source communities (among others): “Let me have the latest bits for package xyz.” And I’m especially interested in the idea that you could construct a system configuration tailored to your needs, consisting of “just” that functionality you want, with a high degree of certainty that the whole combinatorial cocktail, once assembled, will work properly. But I suppose that the topic of “software updates” is as broad as the various purposes to which updates may be put. I expect we’ll pursue some of these other purposes, especially the incremental delivery of functionality to *extend* existing systems, as we go forward.

Nevertheless, our initial engagement tells me that patching and the patch management problem are a major concern to many of you. So we’ve decided to run with that a bit. As a result, you’ll get a view in this issue of software updates from the perspective of a major system software vendor who has to develop and deliver them, and of the large enterprise IT systems manager who has to receive and deploy them.

Somewhat iconic when one thinks of software updates is Microsoft’s Software Update for Windows. We were very lucky to have Joseph Dadzie, who heads Microsoft’s software update team, describe the whole lifecycle of software updates as seen from the Microsoft perspective. In “Understanding Software Patching,” he provides a comprehensive overview of the surprisingly many things that you have to think about when you’re building and delivering updates. I think you’ll find his description quite informative.

Following Dadzie's discourse, which primarily focuses on the patch provider's perspective, is "Patching the Enterprise" by George Brandman, a practical account of what dealing with software updates looks like on the receiving end. Brandman is an executive director in Morgan Stanley's information technology department. He deals with the patching problem "in the large," as they say. Although you and I probably don't normally deal at his scale, it's quite useful to know what problems crop up when you're managing software updates for a collection of systems—perhaps for an entire organization—just as it's seen from an IT perspective in most commercial enterprises.

We wrap up our special report with a piece more to the point of supporting extensibility in software systems. In "On Plug-ins and Extensible Architectures," Dorian Birsan of Eclipse discusses plug-ins as an architectural mechanism for system software evolution. Whereas patches have typically been used by a single vendor to deploy updates or fixes to its own software, plug-ins offer an a priori design for a software system around which more than one software vendor can provide extensions. This is also well worth your reading, and it starts to address the question that I hope we may follow somewhat further in subsequent issues: How do you enhance the functional-

ity of a complex software system that is already deployed using these incremental delivery mechanisms?

One final note—those of you who are committed readers of *Queue* will probably remember Alex Bell's earlier article: "Death by UML Fever" (March 2004). As it was one of *Queue*'s most popular articles to date, we are delighted that Bell is back this month with a reprise, "UML Fever: Diagnosis and Recovery," which describes a 12-step program for recovery from this dreaded (and apparently epidemic) disease.

I hope you find this material, as well as *Queue*'s regular monthly content, engaging, rewarding, and useful. ☺

DAVID J. BROWN is senior staff engineer and chief technologist for the System Interfaces Department, Solaris Engineering Group, at Sun Microsystems. He is currently involved in the development of Sun's Web Services architecture, following recent involvement in Sun's Linux compatibility and open source strategies. Before coming to Sun, Brown was a member of the research staff at Stanford University, where he worked with Andy Bechtolsheim on the prototype SUN Workstation. He later was a founder of Silicon Graphics and subsequently joined Steve Bourne to establish the Workstation Systems Engineering Group for DEC. Brown has a Ph.D. from Cambridge University.