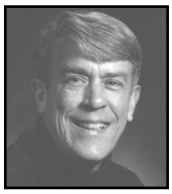

Robert L. Glass

The Loyal Opposition

Maintenance: Less Is Not More

Maintenance has been one of my hot-button topics for more decades than I care to remember—a thread of concern that has run throughout my career continuously. In the 1950s I, like most programmers, began my career doing maintenance. During the 1960s I realized that the things I'd learned while *maintaining* software were the most important things to remember while *building* software. In the 1980s, colleague Ron Noiseux and I wrote a book on the subject (*Software Maintenance Guidebook*, long since out of print—no one wanted to buy a book on the subject). Now, in the 1990s, I occasionally speak about, and teach classes in, maintenance.

Show Me the Maintenance

If this decade's mantra is "show me the money," those of us in software will find it in maintenance. Lifecycle data shows that maintenance is where we spend the biggest chunk of practitioner time and money.

But that's not the view. Few computing academics teach maintenance, even in software engineering programs. It seems to be a topic most computing professionals would like to sweep under the rug. It's time to give the maintenance devil its due.

Legacy Systems Need Love, Too

One keyword for this issue's focus is "legacy." We've adopted that word to describe the aging, often antiquated software systems that soldier on year after year. The dictionary defines legacy as "something handed down by a predecessor." Right on. But then it adds an example of the word's use that struck me as an almost curious afterthought: *a legacy of distrust*. Even in the dictionary, legacy systems get no respect.

Given that modifying old artifacts to serve new purposes is precisely what this issue's focus is about, I think it's especially appropriate that I address maintenance in a similar way: not by writing all-new prose, but by modifying and plugging in something I wrote awhile

EDITOR: Robert L. Glass • Computing Trends • rglass@indiana.edu

0740-7459/98/$10.00 © 1998 IEEE

back. What follows, then, are some snippets derived from a couple of essays roughly five years old: one called "Software Maintenance is a Solution, Not a Problem" (previously published in several places, including my 1991 book *Software Conflict*), and another called "Methods and Maintenance," from a 1993 Software Reflections column I wrote for *Managing System Development*. These abbreviated and intertwined legacy essays retain much of their original content, but have been repackaged in a glossy, new, user-friendly interface.

Because software maintenance adds functionality, it's a solution, not a problem.

Maintenance Isn't Bad, It's Just Misunderstood

Is software maintenance a problem? Today's standard answer is "You bet." The standard rationale for that standard answer is "Look how much of our budget we're putting into software maintenance. If we'd only built the software better in the first place, we wouldn't have to waste all that money."

I disagree. The standard answer is wrong because the standard rationale is wrong. Software maintenance isn't a problem, it's a solution. What we're missing by viewing maintenance as a problem is the special significance of these two pieces of information:

- The software product is "soft" and thus easily changed compared to other, "harder" disciplines.
- Software maintenance is far less devoted to fixing errors than to making improvements.

Software maintenance is a solution rather than a problem because it changes old artifacts by adding new functionality. Indeed, 60 percent of maintenance changes provide solutions, while only 17 percent fix errors. That's not a bad solution-to-problem ratio.

Thoroughly Modern Maintenance

What about the criticism that we could avoid maintenance if we got software right the first time? There's a piece of research that addresses this argument with results that will surprise those who believe maintenance is a waste of money.

Sasa Dekleva of Chicago's DePaul University ("The Influence of the Information Systems Development Approach on Maintenance," *MIS Quarterly*, Sept. 1992, pp. 355-372) unearthed this contrarian finding: The more modern the methods you use in building software, the more time you spend maintaining the resulting product. Dekleva surveyed practitioners who built and then maintained software using several modern systems development approaches. These approaches included software engineering, which he identified as the use of process-oriented structured methods; information engineering, the use of data-oriented methods; prototyping; and CASE tools. The survey went out to 500 data processing managers at Fortune 500 companies and generated a 22 percent response rate.

Two of Dekleva's findings were predictable: Modern methods lead to both more reliable software and less frequent software repair. But his third was not: Modern methods lead to more total maintenance time. When Dekleva obtained that surprising finding, he puzzled over it. Eventually he determined that modern methods lead to more maintenance because the resulting software products are more maintainable and easier to enhance.

This increasing maintenance cost effect didn't kick in immediately after the new software went into production. For the first year of use, these modern-method-built systems actually incur lower maintenance costs, just as we would predict intuitively. The maintenance workload increases slowly, as users acclimate to the system, flush out its early errors, and realize what additional capabilities they would like.

Better Maintenance, Not Less

Dekleva's study confirms that if you build a system well, it will be maintained more than if you don't. Why? Because maintenance is about product enhancement, and a well-built system is easier to enhance.

So if you still believe "we can eliminate maintenance by doing our development jobs better," Sasa Dekleva and I have news for you. That's not how it works.

When it comes to software maintenance, it's increasingly clear that we should be doing more of it, not less. Maintenance is a unique advantage that the discipline of software brings to the table. Maintenance can be best served not by searching for ways of obliterating it, as some have suggested, but by searching for ways of doing it more effectively and efficiently. ❖