

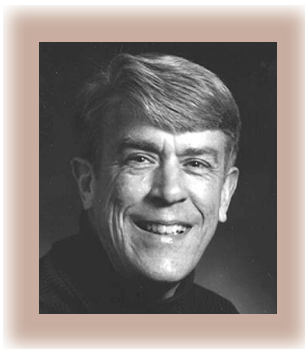
Learning to Distinguish a Solution from a Problem

Robert L. Glass

... in which I make a plea for more respect for software maintenance and software maintainers

This column is about software maintenance. Don't quit reading, now! I know that seems like a pretty boring topic. But it's also a vitally important and, more to the point, a totally misunderstood one.

I've been flaming about software maintenance for what seems like several decades. Someone comes out and says something completely stupid about maintenance, and once again I leap into the fray to try to set the record straight.



The latest stupidity

InformationWeek is one of my favorite popular-press computing magazines. It doesn't engage in the hype and excess of most of its kin, and I deeply respect it for that. But it was an editorial in the 24 March 2003 issue of *IW* that set me off this time. In that issue, Bob Evans, *IW*'s editor in chief, starts off on a crusade about what he calls "The Monster in the Basement." As you read on in his editorial, you come to realize that he's talking about legacy software. Not only does he call it a monster, he says "it's strangling innovation and progress," "it's a recipe for atrophy, irrelevance, and ultimately collapse," and the increase in the amount of maintenance over the years is a "dangerous trend."

IW and its editor aren't the first to pick on maintenance, of course. Over the years, it's been a favorite topic for those who see it as an impediment to progress. One notable in the field even paraphrased the "don't automate, obliterate" phrase to apply it to maintenance

of legacy code, implying that we ought to throw away the old and rebuild anew. And if you read Evans's editorial carefully, that's what he's advocating as well.

The buck stops here

We in the field seem, in many ways, to reinforce that negative view of maintenance:

- We tend to assign the newest and greenest programmers to maintaining our legacy programs. We assign the best and brightest to new development.
- We eschew the old and the ancient as being out of date. We evolve toward new paradigms and new concepts and are eager to leave the old ones—including old code—behind.
- We teach programmers how to write new programs, but we rarely if ever teach them how to read old ones.
- We talk about software maintenance as if it were a problem and think we should do less of it.
- We evolve and promote methodologies to guide our new software development. We rarely provide tools and techniques, let alone methodologies, to maintenance programmers.
- Once a piece of new software goes into production, we breathe a sign of relief, break up the team, and prepare to move on. We rarely hold postimplementation reviews, gather past project data for future learning experiences, or organize what some academics call *experience factories* to capitalize on lessons learned.

Continued on p. 111

Continued from p. 112

In short, we generally behave as if the maintenance of legacy software is a Bad Thing, one that we believe would go away if only we would engage in enough Good Acts.

The facts of maintenance

All of that is backwards! Somehow, over the few years that software has been a productive, useful discipline, we keep trying to kill the goose that lays more golden eggs than any other in the software barnyard. By disrespecting the maintenance of legacy software, we've turned our collective back on (a) what most people in software do, (b) what most of the money in software is spent on, and (c) our greatest opportunity to please our clients and customers.

But there's more. In disrespecting maintenance, we also ignore some of the most important and perhaps surprising facts in the software field. In *Facts and Fallacies of Software Engineering* (Addison-Wesley, 2003), I list these three frequently forgotten but fundamental facts about maintenance:

- Maintenance typically consumes 40 to 80 percent of software costs. So, it's probably software's most important lifecycle phase.
- Enhancement is responsible for roughly 60 percent of software maintenance costs. Error correction is roughly 17 percent. So, software maintenance is largely about adding new capability to old software, not fixing it.
- Better software engineering development leads to more maintenance, not less. (To completely understand this counterintuitive fact, you'll have to read the book! But basically, this concerns backlogs of maintenance activities such that, if a maintenance task takes less time, you can address more of the backlog.)

Given those facts, I conclude that maintenance is a *solution*, not a *problem* (another frequently forgotten but fundamental fact, and the rallying cry

I'd like you to take away from this column!). And, if it's a solution, we should be doing more of it, not less.

History repeats itself

Let me tell you a little story about maintenance, one that's very personal and yet generalizable to the whole field. One of software's most articulate spokespersons, when he was fresh out of college, was asked by his company to discard an old, legacy software product and produce a new version of it. He accepted the assignment with alacrity, looking forward to putting to work all those gleaming new concepts he had learned in computing academe.

As time festered on, he realized to his horror that the task was approaching insurmountability. No one knew all the requirements for the old product (those who did had left the company long ago). No one knew enough about the old code's design to reengineer it to recreate those requirements. No one seemed to understand how much work had gone into the old product, so the expectation existed that with new and gleaming techniques, the re-creation would somehow take an insignificant amount of time. And, as time passed, no one in management held much patience for the mushrooming task of discarding the old and re-creating it.

In the end, the new product was thrown away, a dead loss, because it

couldn't be made to do what the old product had done. The old, legacy product was placed back in maintenance, soldiering productively on into the future.

If that were the only such story in software history, it would make an interesting anecdote, but nothing more. But I've seen this story repeated time after time. Brilliant software engineers such as David Parnas and his team were unable to reproduce, using new techniques, the software functionality for the A-7 aircraft on a sufficiently timely basis. Hordes of software specialists at a leading banking institution were unable to produce a new, enhanced version of their old and seemingly obsolete main product because they couldn't wrap their arms around its complex functionality. In my view, some of the worst disasters in computing history have involved trying to rebuild those old legacy systems.

Now, is legacy software really "the monster in the basement"? Or is it, instead, the silent majority of software work, quietly cranking out valid, useful results, decade after decade?

You know what my answer is. ☺

Robert L. Glass is editor emeritus of Elsevier's *Journal of Systems and Software*, the publisher and editor of the *Software Practitioner* newsletter, and a certified "premier curmudgeon of software practice." Contact him at rglass@indiana.edu; he'd be pleased to hear from you.

Copyright and reprint permission: Copyright © 2004 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limits of US copyright law for private use of patrons those post-1977 articles that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Dr., Danvers, MA 01923. For copying, reprint, or republication permission, write to Copyright and Permissions Dept., IEEE Publications Admin., 445 Hoes Ln., Piscataway, NJ 08855-1331.

Circulation: *IEEE Software* (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1314; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 1730 Massachusetts Ave. NW, Washington, DC 20036-1903. Subscription rates: IEEE Computer Society members get the lowest rates and choice of media option—\$44/35/57 US print/electronic/combo; go to <http://computer.org/subscribe> to order and for more information on other subscription prices. Back issues: \$20 for members, \$116 for nonmembers (plus shipping and handling). This magazine is available on microfiche.

Postmaster: Send undelivered copies and address changes to Circulation Dept., *IEEE Software*, PO Box 3014, Los Alamitos, CA 90720-1314. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to 4960-2 Walker Rd., Windsor, ON N9A 6J3. Printed in the USA.