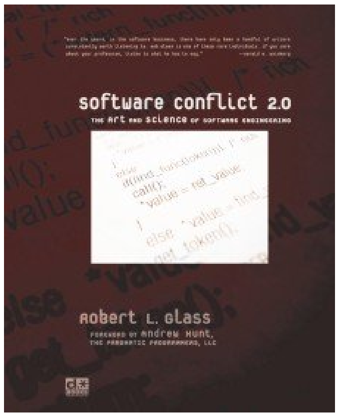
Book Excerpt



Software Conflict 2.0: The Art and Science of Software Engineering

by Robert L. Glass

ISBN 0977213307; US $29.99 / UK £22.99; 308 pages

Software Maintenance is a Solution, Not a Problem

Is software maintenance a problem?

Today's standard answer is "You bet it is."

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 on maintenance."

Well, I want to take the position that this standard answer is wrong. It's wrong, I want to say, because the standard rationale is wrong.

The fact of the matter is, software maintenance isn't a problem, it's a solution!

What we are missing in the traditional view of software as a problem is the special significance of two pieces of information:

1. The software product is "soft" (easily changed) compared to other, "harder," disciplines.
2. Software maintenance is far less devoted to fixing errors (17 percent) than to making improvements (60 percent).

In other words, software maintenance is a solution instead of a problem because in software maintenance we can do something that no one else can do as well, and because when we do it we are usually building new solutions, not just painting over old problems. If software maintenance is seen as a solution and not as a problem, does that give us some new insight into how to do maintenance better?

I take the position that it indeed does.

The traditional, problem-oriented view of maintenance says that our chief goal in maintenance should be to reduce costs. Well, once again, I think that's the wrong emphasis. If maintenance is a solution instead of a problem, we can quickly see that what we really want to do is more of it, not less of it. And the emphasis, when we do it, should be on maximizing effectiveness, and not on minimizing cost.

New vistas are open to us from this new line of thinking. Once we take our mindset off reducing costs and place it on maximizing effectiveness, what can we do with this new insight?

The best way to maximize effectiveness is to utilize the best possible people. There is a lot of data that supports that conclusion. Much of it is in the "individual differences" literature, where we can see, for example, that some people are significantly better than others at doing software things:

Debugging: some people are 28 times better than others.

Error detection: some people are 7 times better than others.

Productivity: some people are 5 times better than others.

Efficiency: some people are 11 times better than others.

The bottom line of these snapshot views of the individual differences literature is that there is enormous variance between people, and the best way to get the best job done is to get the best people to do it.

This leads us to two follow-on questions:

1. Does the maintenance problem warrant the use of the best people?
2. Do we currently use the best people for doing maintenance?

The first question is probably harder to answer than the second. My answer to that first question is "Yes, maintenance is one of the toughest tasks in the software business." Let me explain why I feel that way.

Several years ago I coauthored a book on software maintenance. In the reviewing process, an anonymous reviewer made this comment about maintenance, which I have remembered to this day:

Maintenance is:

- intellectually complex (it requires innovation while placing severe constraints on the innovator)
- technically difficult (the maintainer must be able to work with a concept and a design and its code all at the same time)
- unfair (the maintainer never gets all the things the maintainer needs. Take good maintenance documentation, for example)
- no-win (the maintainer only sees people who have problems)
- dirty work (the maintainer must work at the grubby level of detailed coding)
- living in the past (the code was probably written by someone else before they got good at it)
- conservative (the going motto for maintenance is "if it ain't broke, don't fix it")

My bottom line, and the bottom line of this reviewer, is that software maintenance is pretty complex, challenging stuff.

Now, back to the question of who currently does maintenance. In most computing installations, the people who do maintenance tend to be those who are new on the job or not very good at development. There's a reason for that. Most people would rather do original development than maintenance because maintenance is too constraining to the creative juices for most people to enjoy doing it. And so by default, the least capable and the least in demand are the ones who most often do maintenance.

If you have been following my line of reasoning here, it should be obvious by now that the status quo is all wrong. Maintenance is a significant intellectual challenge as well as a solution and not a problem. If we want to maximize our effectiveness at doing it, then we need to significantly change the way in which we assign people to it.

I have specific suggestions for what needs to be done. They are not pie-in-the-sky theoretical solutions. They are very achievable, if management decides that it wants to do them:

1. Make maintenance a magnet. Find ways to attract people to the maintenance task. Some companies do this by paying a premium to maintainers. Some do this by making maintenance a required stepping stone to upper management. Some do this by pointing out that the best way to a well-rounded grasp of the institution's software world is to understand the existing software inventory.

2. Link maintenance to quality assurance. (We saw this in the previous essay.)
3. Plan for improved maintenance technology. There are now many tools and techniques for doing software maintenance better. (This has changed dramatically in the last couple of years.) Training and tools selection and procurement should be high on the concerned maintenance manager's list of tasks.
4. Emphasize "responsible programming." The maintainer typically works alone. The best way to maximize the effectiveness of this kind of worker is to make them feel responsible for the quality of what they do. Note that this is the opposite of the now-popular belief in "egoless programming," where we try to divest the programmer's personal involvement in the final software product in favor of a team involvement. It is vital that the individual maintainer be invested in the quality of the software product if that product is to continue to be of high quality.

There they are...four simple steps to better software maintenance. But note that each of those steps involves changing a traditional software mindset. The transition is technically easy, but it may not be socially or politically quite so easy. Most people are heavily invested in their traditional way of looking at things.

If we are to get there at all, however, there is one vital first step which must be taken. It is the step that started off this essay.

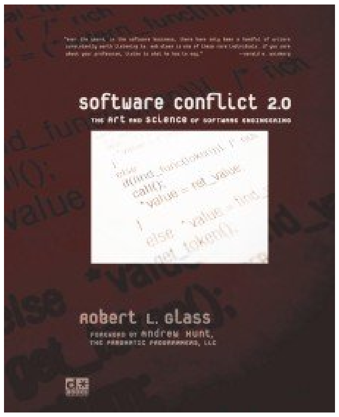
We must see that software maintenance is a solution, not a problem. If we agree to that, then we have opened the door to profound changes in how we software people do our business. Think about it.

###

About the Author

Robert L. Glass held his first job in computing in 1954. Author of over 25 books, he is one of the true pioneers of the software field. He is the editor and publisher of *The Software Practitioner*, and also writes regular columns for *Communications of the ACM* and *IEEE Software*. In 1995 he was awarded an honorary Ph.D. from Linkoping University of Sweden, and in 1999 he was named a Fellow of the ACM professional society. His unique viewpoint and timeless writings have for decades offered insights to practitioners, managers, professors, entrepreneurs, researchers, and students alike.

About the Book



Title	*Software Conflict 2.0: The Art and Science of Software Engineering*
Author	Robert L. Glass
Foreword	Andrew Hunt, Pragmatic Programmers, LLC First edition Foreword by Donald J. Reifer, Reifer Consultants Inc.
ISBN	0977213307
Pages	308
Price	$29.95 U.S. / UK£22.99

As loyal Robert Glass readers have come to expect, *Software Conflict 2.0* takes up large themes and important questions, never shying away from controversy. Robert Glass has a unique perspective, owing partly to his longevity in the field, partly to his breadth and depth of experience as a practitioner, and partly to his experiences on multiple continents crossing back and forth between the worlds of the university and the professional programming shop.

No matter what unique corner of the software engineering world you call home—be it aerospace or e-commerce—whether you are a researcher, hardcore coder, consultant, or manager, *Software Conflict 2.0* tackles questions and conflicts that you will recognize. Bob Glass's wide and deep perspective on the art and science of software engineering will widen and deepen your own perspective.

The first edition of *Software Conflict* was published circa 1990 and, until now, has been out of print for some time. Why? Mainly because that’s the normal pattern for software books: a new book is hot when it hits the streets, but then trends change, paradigms shift, and eventually the publisher stops placing orders with the printer. As hundreds of new books are published every year, a real treasure can be buried in the shifting sands.

Sometimes the significance of a software book transcends the endless cycle of trends and revolutions. In fact, some of the great software books continue to be discussed even decades after their original publication. Why do people keep reading these "dated" software engineering books?

Because the insights of these great books are timeless, as valid today as they were yesterday. Because these insights help us become better software professionals, better researchers, better managers. And because the writings of a computing pioneer like Robert L. Glass might just reveal something about where we are today and where we're headed.

Software Conflict 2.0 features six new essays by Robert Glass and a new Foreword by Andrew Hunt of the Pragmatic Programmers.

Order now at `www.DeveloperDotStar.com` or wherever books are sold.