

Error-Free Software Remains Extremely Elusive

Robert L. Glass

... in which I present evidence to oppose the notion that software can commonly be built error-free.

One of the more interesting questions in software engineering today is "Is it possible to produce error-free software?"

Advocates of various technological advances tend to argue that, with whatever approach they are advocating, error-free software is indeed possible. Software old-timers are, at best, dubious.



A recent study casts important light on this issue. That study sets out to explore the related question "Can state-of-the-art software engineering methods guarantee reliability?" As it turns out, perhaps surprisingly, the answer was "no."

Two approaches, similar results

In the study, two practitioner teams implemented a single real-time personnel-badge-reading application, employing two very different software development approaches.¹ The "CMM4" team used a waterfall-based, relatively traditional, approach, relying on UML (Unified Modeling Language) and Rational Rose and their Capability Maturity Model Level 4 status to produce the required product reliability. The "FORMAL" team used formal methods, including the functional programming language Haskell, and Specware, a tool for formal specification.

Both teams aimed to produce a product

that was reliable 90+ percent. When the smoke had cleared, both teams believed their product met its reliability requirements. Both were quite wrong!

The researchers divided the reliability test results into two failure categories:

- Level 1 failures bring the system to a critical state (in general, these failures were unrecoverable). The required reliability was 99 percent.
- Level 2 failures are less severe; they can be worked around. The required reliability was 90 percent.

CMM4 was reliable at only the 56-percent level for its Level 1 tests and at the 97-percent level for Level 2. FORMAL hit 77 percent for Level 1 and 65 percent for Level 2. So, CMM4's results were less successful because their failure rate for the most severe results was worse, although CMM4 did pass the test for Level 2 failures.

What went wrong?

CMM4 misjudged the set of customer requirements for the product, pronounced that set "one of the best ever," and implemented it without further questions. No designers or implementors analyzed the requirements, which made spotting their flaws (predominantly ambiguity) impossible.

As time passed, and the team found itself increasingly unable to meet the "generous" schedule and cost constraints, they sacrificed Level 4 processes and reverted to Levels 1

Continued on p. 103

Continued from p. 104

and 2. A junior programmer had difficulty using some new object-oriented tools, so the team conscripted some senior-level "heroes" to try to recover from the problem. They also tried to scale down the CMM Level 4 approaches to match this smaller project's needs; that scaling down, according to the authors, proved "inadequate."

On the positive side, the authors claim that "documentation of the process was very good and the code should be easy to maintain."

FORMAL, in contrast, did a largely excellent job with the requirements analysis. However, they lost some of the requirements owing to a lack of process.

Things began to fall apart when the two senior people who did all the project's tasks got into the coding phase. The expected automatic code generation from specifications, performed with Specware, was "a major undertaking," with "technical incompleteness" (for example, "a simple assignment statement needed refinement by the ... tool supplier"). Code generation consumed so much time and resources that, once again, the team could not meet the expectedly generous schedule. In this case, the team sacrificed the specification-theorem-proving that it had planned to perform. And, unlike the other team, "the development process was poorly documented." The study's authors make no comment about the code's maintainability.

The authors are reluctant to generalize their findings, on the grounds that the tasks were too simple to be representative. However, it is apparent that, if a predefined reliability figure could not be met on a fairly simple project, no matter which of the two development approaches were taken, then achieving such reliability targets on more complex assignments is unlikely.

Which, many would say, leads to a fairly straightforward answer to our original question. No matter how traditionally conservative or sophisticatedly avant garde the software development approach, it is unlikely

that we can, at this point in the field's history, build error-free software.

Inadequate estimation; inadequate processes?

Before we leave this study's findings, let's examine more closely two facets of the results that I mentioned earlier in passing:

- Both teams failed to meet their cost and schedule requirements, even though these requirements were originally conceived to be generous.
- Both teams tried to shortcut processes they had expected to use, to play catch-up with their schedule problems.

These results are not, of course, unusual for a software project. Cost and schedule failures, I would assert, far exceed any other kinds of software failures in practice. That the teams expected that these targets were generous, however, makes this situation particularly interesting. Apparently, even with favorable estimates, software people have an exceedingly difficult time achieving predicted targets. No wonder software projects frequently fail to achieve the normally terribly optimistic targets set for them.

The shortcutting presents an even more interesting issue. When a team of software practitioners eschews its processes under schedule pressure, there are two possibilities:

- They are responding poorly to that pressure.
- They are shucking processes that they believe to be of marginal value anyway.

We have no clue as to which of the two held in this study. You could easily imagine that the first possibility is true. But I have one concern about that. In a recent *Software Development* magazine conference presentation, Karl Wiegers asserted that software people should "never let anyone talk you into doing a bad job." I believe that most practitioners would agree with that position and that few would willingly shortcut a process they believed was important. Because of that, I can't help but wonder about the second possibility. Did the CMM4 team really believe that Level 4 processes were an improvement over Levels 1 and 2? Did the FORMAL team really believe that the specification-theorem-proving they had intended to do was important? Perhaps a follow-up to this study could answer those questions.

Reference

1. C. Smids, X. Huang, and J.C. Widmaier, "Producing Reliable Software: An Experiment," *J. Systems and Software*, vol. 61, no. 3, 1 Apr. 2002, pp. 213-224.

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 curriculum of software practice." Contact him at rglass@indiana.edu; he'd be pleased to hear from you.

Copyright and reprint permission: Copyright © 2003 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; (714) 821-8380; fax (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—\$43/34/56 US print/electronic/combination; go to <http://computer.org/subscribe> to order and for more information on other subscription prices. Back issues: \$10 for members, \$20 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 #R125634188. Canada Post Publications Mail Product (Canadian Distribution) Sales Agreement Number 0487805. Printed in the USA.