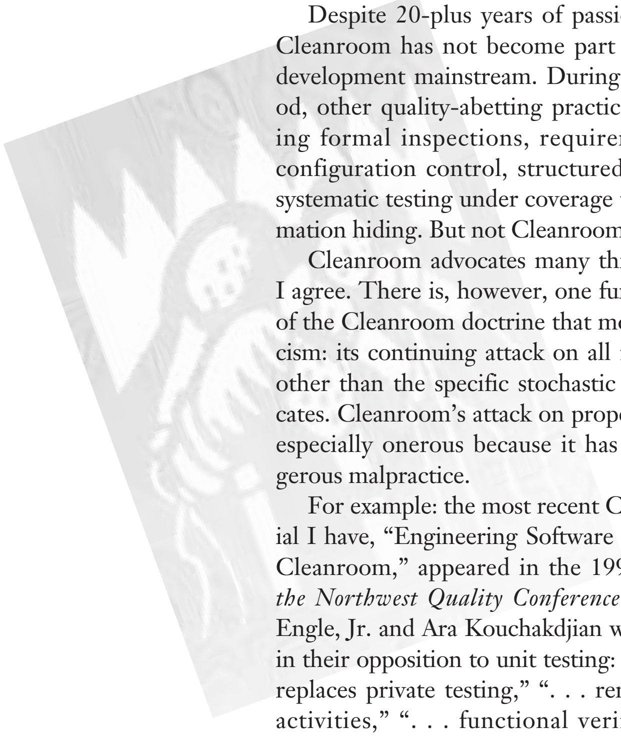
Cleanroom Process Model: A Critical Examination

A forum for exchanging ideas, philosophy, and experience.

Editor: Tomoo Matsubara
Matsubara Consulting
1-9-6 Fujimigaoka,
Ninomiya Naka-gun,
Kanagawa 259-01
Japan
matsu@sran125.sra.co.jp



SINCE HARLAN MILLS INTRODUCED IT more than 20 years ago, the Cleanroom process model has enjoyed considerable—and unwarranted—favorable publicity. A self-promoting article on Cleanroom seems to be an annual event in *IEEE Software*. Almost nothing critical of Cleanroom has been published. Is it because Cleanroom is beyond reproach or because the best would-be critics have not taken Cleanroom seriously?

Despite 20-plus years of passionate advocacy, Cleanroom has not become part of the software development mainstream. During this same period, other quality-abetting practices have, including formal inspections, requirements analysis, configuration control, structured programming, systematic testing under coverage tools, and information hiding. But not Cleanroom.

Cleanroom advocates many things with which I agree. There is, however, one fundamental tenet of the Cleanroom doctrine that moves me to criticism: its continuing attack on all forms of testing other than the specific stochastic testing it advocates. Cleanroom's attack on proper unit testing is especially onerous because it has promoted dangerous malpractice.

For example: the most recent Cleanroom tutorial I have, "Engineering Software Solutions Using Cleanroom," appeared in the 1995 *Proceedings of the Northwest Quality Conference*. In it, Charles Engle, Jr. and Ara Kouchakdjian were unequivocal in their opposition to unit testing: ". . . verification replaces private testing," ". . . removal of testing activities," ". . . functional verification but no debugging . . . before independent statistical usage testing," "No private testing," "Development team typically does not compile or execute the code."

Cleanroom's attack on unit testing is ill-informed as to current best testing practices. It is based on a strawman testing model decades out of date. Its claimed superiority is based on grievously flawed attempts at controlled experiments. Cleanroom's attack contradicts known testing theory as well as common sense. For these reasons I am confident that Cleanroom is, and will remain, an underdeveloped and overexposed minority practice.

HISTORICAL PERSPECTIVE. We must step back a few decades to understand what motivated Mills to propose Cleanroom as a solution to the perennial software quality problem. Software engineering did not yet exist as a discipline. Programming was ego-intensive and idiosyncratic. Almost everything was sacrificed to supposed efficiency and memory conservation. There was no distinction between testing and debugging. Notions of what constituted effective testing were primitive compared to what we know today: the "best" testing was generally considered to be exhaustive testing of all possible inputs, which was obviously impractical and often theoretically impossible. Dynamic instruction modification was common. GOTOs were rampant. Notions of decent structure were as yet unformed. Strong typing was generally unknown and programmers who understood the notion objected to it. Documentation was pitiful. Software components were often huge by today's standards: 50,000 assembly language instructions without a single subroutine and with in-line device drivers. Formal inspections did not exist and the closest thing we had to it—reviews—were often simply disorganized hostility sessions. All this could be excused if the software had been on time, within budget, and reliable, but even among the best practitioners of the day software was late, expensive, and very, very bad.

Cleanroom is an underdeveloped and overexposed minority practice.

Mills believed that bug prevention was more cost-effective than bug detection and correction. That was true then. It is true today. Mainstream developers and testing advocates such as myself acknowledge this. However, Mills saw testing, which he viewed as synonymous with debugging, as an unstructured hack through the code in hopes of finding a bug or two. He raised the ludicrous prospect of exhaustive testing as the only alternative (other than Cleanroom) to aimless hacking. Indeed, compared to the so-called testing/debugging practices of the day, Cleanroom *was* an improvement. Almost anything would have been.

Today, Cleanroom advocates for the most part still parade that exhaustive testing strawman. They don't acknowledge three decades of testing progress, testing theory, and the widespread practices based on that theory. The Cleanroom literature's testing notions are stuck back in the '60s.

CLEANROOM FLAVORS. Cleanroom, as its apologists are quick to point out, is not a monolithic movement. Advocates come in all stripes, from ultraconservatives who permit only stochastic testing (to measure reliability) to revisionists who permit some additional testing.

There is considerable variability in what is or is not included under the Cleanroom umbrella. Some advocates, following Mills, insist that programmers not compile their own code. Most condemn unit testing; others permit it for "experimental purposes." I don't know where actual Cleanroom practices may be on this spectrum, but the literature is still distinctly orthodox (see, for example, Richard Linger's article, "Cleanroom Process Model," *IEEE Software*, March 1994). No programmer compilation or unit testing is still the idealized goal.

MAKING FAIR COMPARISONS. The published Cleanroom rhetoric is unequivocal in its condemnation of "testing." I use quotes because Cleanroom's testing strawman is nothing like the testing practices, especially unit testing, advocated by the testing community. When advocates "compare" Cleanroom with testing, it is always with the obsolete hacking model of a quarter-century ago. Cleanroom is never measured against

- ♦ proper unit testing done under coverage standards,
- ♦ people trained in testing techniques,
- ♦ shops that use test design and execution automation technology, or
- ♦ organizations that do proper integration testing.

The very notion of coverage is deplored in Cleanroom's literature. Whenever the term appears it is as "coverage testing" and invariably pejorative (see, for example, Richard Cobb's and Harlan Mills' article, "Engineering Software Under Statistical Quality Control," *IEEE Software*, Nov. 1990). I suppose if your strawman is awful enough, anything can look good by comparison.

PERMISSIBLE "TESTING." The only "testing" permitted under the orthodox Cleanroom doctrine is stochastic testing to determine the program's expected failure rate. Indeed, Cleanroom rejects testing's fundamental purpose: to find bugs. Thus, Cleanroom rejects both common sense and testing theory. You cannot find a bug unless you execute the code that has the bug.

Because Cleanroom rejects all notions of coverage and the bug-finding purpose of testing, it appears to be content to allow software to be fielded with untested code. This is dangerous. Reliability testing, by design, focuses on high-probability paths; most serious and potentially dangerous bugs occur on the low-probability, error-condition paths.

Last year's publication of Michael Luy's monumental *Handbook of Software Reliability Engineering* (IEEE Computer Society Press/McGraw-Hill) was a watershed. The best thinkers on the subject have put down what they know. However, as important as the material is, SRE's caveats and problems are more important—especially the difficulties of obtaining meaningful user profiles. Cleanroom's literature treats SRE as if it has the same solidity as hardware reliability engineering. This is naive, and contrary to known facts and expert opinion. If, as Cleanroom advocates, the sole purpose of testing is stochastic testing to determine failure rates, then they have, by their own doctrine, ruled out Cleanroom's use for the many applications for which current SRE is either problematic or impossible.

EXPERIMENTAL VALIDATION? Over the years there have been several oft-cited and often republished studies that purport to demonstrate Cleanroom's superiority over unit testing and indeed, the software development mainstream. These studies, such as that by Victor Basili and Scott Green (see *IEEE Software*, July 1994) and that by Richard Selby and colleagues (see *IEEE Transactions on Software Engineering*, Sept. 1987), while well-intentioned and undoubtedly honestly conducted, are invalid. Both attempted to conduct controlled experiments to compare Cleanroom's effectiveness to that of "testing," but several things went wrong.

1. The subjects knew that they were part of an experiment (the experiments were not even single-blind, never mind double- or triple-blind).
2. "Testers" had no training in proper testing methods.
3. Testers did not use coverage tools.
4. Testers apparently used no automation methods.
5. Nothing in the experimental procedure prevented cheating (such as testing and debugging by traditional means and then retrospectively subjecting the already debugged code to the Cleanroom treatment).

As of 1995, a total of 24 projects, ranging in size from 1,820 LOC to 350 KLOC, have been cited in support of Cleanroom. It is not known which of these projects used orthodox Cleanroom doctrine and which allowed unit testing.

The Cleanroom literature's testing notions are stuck back in the '60s.

However, even assuming that all were truly orthodox without programmers' compilation or unit testing, 240 projects, never mind 24, is not enough data to make any kind of statistical inference whatsoever. Proper statistical methods are based on epidemiology, with each project representing one data point. The statistically valid supporting evidence for Cleanroom, despite its continual republication (which doesn't add to the data set), is nil.

CLEANROOM CHALLENGE. If testing is as bad as Cleanroom advocates claim, they should be able to refute testing theory by exposing its flaws using mathematically rigorous methods. They should be able to prove the worthlessness of its axiomatic foundations, its models, and its proof methodologies. The Cleanroom literature is constant in its intemperate attack on testing, and therefore, by implication, on testing theory. Yet not one Cleanroom author has ever published a paper criticizing testing theory or any part of it. I challenge Cleanroom authors to begin their rebuttal by refuting Gourlay. Then they can work their way through Gerhart, Goodenough, Hamlet, Harrold, Howden, Korel, Laski, Morell, Ntafos, Ostrand, Podgursky, Rapps, Richardson, Soffa, Tai, Voas, Weiser, Weyuker, White, and Zeil.

Otto Vinter has developed an elegant experimental method for comparing software development processes that does not suffer from the problems I've noted here ("The Prevention of Errors Through Experience-Driven Test Efforts," Delta Report D-259, Jan. 1996; http://www.esi.es/ESSI/Reports/All/10438). I call Vinter's method retrospective analysis. Experimental blindness is not an issue because the subjects never know they are part of an experiment; all analysis is done after the fact. For example, to compare methods A and B, develop software using method A, then field it, recording all discovered bugs for a year or so. You would then apply method B to the code and see which of those bugs B would have found. Vinter did this to experimentally show the cost-effectiveness of various unit-testing coverage criteria. A complete experiment would then apply method A to a B project retrospectively.

So, to Cleanroom advocates, I offer a formal challenge: take any orthodox Cleanroom project and submit it to Vinter's retrospective analysis. As a variation, explain why we should not remove any code not covered under the prescribed stochastic testing. Develop your software using Cleanroom. Then let us test it further under coverage tools (can't hurt, can it?). We'll then remove all uncovered code. A repetition of your stochastic tests will yield the same mean time between failures; and by the Cleanroom doctrine, this revised code should be equally acceptable. The challenge, part two: agree or explain the fallacy in this reasoning.

Cleanroom, despite 25 years of publicity, has not become part of the development mainstream.

THE BOTTOM LINE. Cleanroom, despite 25 years of publicity, has not become part of the software development mainstream. The most likely reason is that it is a profoundly flawed and dangerous concept, and rational software developers have rejected it out of common sense rather than an unwillingness to adopt new methods. I understand Cleanroom's appeal: it promises to rid us of 50 percent of software development labor—the most onerous 50 percent at that. But if Cleanroom has a future, it will have to earn it. As a packaged methodology, it will have to earn it in the marketplace and slug it out with other purveyors of packaged methodologies. Conversely, if it claims to be a science, Cleanroom researchers will have to engage in scientific debate and provide formal proofs of the mainstream testing fallacies they so enthusiastically point out.

In either case, it is high time that we stop giving Cleanroom free publicity and the respect due to valid science until, if ever, it has earned the fanfare and respect. ◆

A more extensive version of this essay with references can be found in the March/April 1995 Journal of the Society for Software Quality *and the 1995* Proceedings of the Pacific Northwest Quality Conference *("The Cleanroom Process Model: A Critical Examination").*

Boris Beizer is an international software testing and quality assurance consultant and author of 12 books, including Software Testing Techniques *and* Software System Testing and Quality Assurance. *His latest book is* Black Box Testing, *an introduction to testing technology. Beizer was testing director for the US FAA's Weather Message Switching Center and several other large communications systems. He can be reached at bbeizer@acm.org or bbeizer@sprintmail.com.*