

Quality Software

Correctness vs. Quality

Moon's First Maxim: The process of developing a system uncovers information about the system that no one could have known at the offset. [Richard Moon]

Moon's Second Maxim: Development methodologies that do not support iterative development are doomed to failure. [Richard Moon]

Brook's Law: **Adding manpower to a late software project makes it later.**

The Japanese treat every product as an ongoing experiment and are constantly engaged in improving it. [Peters]

Find out what was wrong, try to understand why it had gone wrong, and then break down the corrective process into modest steps. [David Halberstram, The Reckoning]

Idea

Software Status Bulletin contains every known problem, but doesn't highlight the disastrous ones.

Explicit Design Criteria

Programmers need guidelines to help them make difficult tradeoffs while programming. The criterion is reliability first, then compatibility, performance, and finally features. Without such leadership from management, programmers cannot be expected to produce a consistent and dependable style of program.

Do automatic batch regression testing of each new version. The tests are designed to abort if anything goes wrong, or update a results file if they make it to the end. Run the test suite every night to check that day's changes. Ideally, each bug uncovered should be verified with a batch job that reproduces it. Once the bug is corrected, the test will pass and will ensure that old bugs do not creep back in by accident.

Frequent User Testing

The biggest danger is that you will deliver a working system that doesn't do what the client needs or wants. Even when code does what the user wants, it never does it exactly right on the first pass. To minimize this risk, send pre-releases of revised software to selected clients early. A pre-release program is much like a regular release, including

updated manuals and on-line help. Seek out clients who will treat this software as harshly as they would an official new product. Find the best candidate by searching in the tech-support records. The benefit is mutual: Get objective feedback, and the clients often get very quick solutions to their problems

Programmers Should Take Technical Support Calls.

There is nothing like hearing directly from an irate user of a piece of software that you wrote to motivate you to improve it.

1. Use of source control
 - a. Positive: We use one.
 - b. Open: How often checked in? What are the conditions to check in? Does it include comments and bug fix reference?
 - c. Recommendation
 - i. Recommended Daily
 1. Get latest
 2. Compile
 3. Run Unit Tests
 4. If Tests pass then start developing
 5. Check out
 6. Develop
 7. Compile
 8. Run Unit Tests
 9. Get Latest (Always do a Get Latest before checking in as code you didn't change could break your code)
 10. Compile
 11. Run Unit Tests
 12. Check In if all tests passed
 13. Run Unit Tests to confirm everything is working
 - ii. Automated get latest script
 - iii. Automated regression testing
2. Writing specifications
 - a. Capture for the engineering and customer what the code will do
 - b. Guidelines for the developer when there are missing / conflicting requirements.
3. Feature complete milestones
4. Each developer estimates time to task
5. Enforce code ownership. Bugs should be fixed by the original source
6. One step builds
7. Automated builds
8. Use best tools
9. Use dedicated testers