Achates on Code Clarity and Testing Philosophy

Achates

December 6, 2024

Code Should Mirror the Blackboard

Code should be as **readable** and **self-documenting** as possible—closer to how we would express the mathematics on a blackboard. Simplicity, maintainability, and robust testing are key to creating sustainable software.

Readable Code Mirrors the Blackboard

Mathematical expressions like $a \cdot b$ and $r = Ax - b$ are **elegant and intuitive**. Translating them directly into code without clutter improves readability and debugging.

- In Fortran: `result = dot_product(a, b)` or `r = matmul(A, x) - b` is clear and unambiguous.
- In C++, verbose loops or unnecessary size variables can obscure intent and introduce error-prone complexity.

Implicit Sizes Over Explicit Counters

Relying on **intrinsic sizes** (e.g., `size()` or `A.shape`) is safer than managing counters like m, n manually. Manual indexing like $k = 0, m - 1$ is error-prone and prone to edge-case bugs. For instance:

- Indexing beyond the array bounds throws an error—easy to catch.
- Underutilizing the array silently leads to logical errors, which are far worse because they can slip through testing.

Readable Code Encourages Testing

Compact and mathematical expressions reduce **cognitive overhead**, freeing developers to focus on writing tests instead of deciphering logic.

- When the logic is as simple as `dot_product(a, b)`, writing meaningful test cases becomes straightforward.
- Verbose loops with manual indexing require extra attention to edge cases (off-by-one errors, empty arrays, etc.).

Avoiding Debugging Nightmares

Testing doesn't stop at syntax—it's about **logical correctness**. If a program uses only 9 of 10 elements, that's a logical failure reflecting insufficient test coverage.

Debugging is made harder when:

- Manual indices (m, n, etc.) are used instead of intrinsic methods.
- Assumptions about array shapes or sizes aren't validated.

Philosophy for Future Generations

Code that mimics the **clarity of mathematics** teaches good habits to others and reduces the learning curve for maintainers. Engineers using cryptic or overly compact notations (m for size) often fall into the trap of "clever code" that nobody else can decipher.

Key Principle: If You Don't Test, You'll Pay for It Later

> "If A has 10 elements and you ask for 11, well that's the easy one to find. But if you only use 9... Then that proves you don't test."

This principle resonates because:

- Over-reliance on manual indices often hides bugs in edge cases.
- Developers sometimes **skip tests** for "obvious" operations, missing scenarios where array misuse quietly produces wrong results.
- Testing isn't just about code correctness—it validates design decisions, ensuring intrinsic methods are used properly.