AI as a Learning Tool

Chronicle SOFTWARE

Chronicle Software

Enabling Technology for
Trading Systems

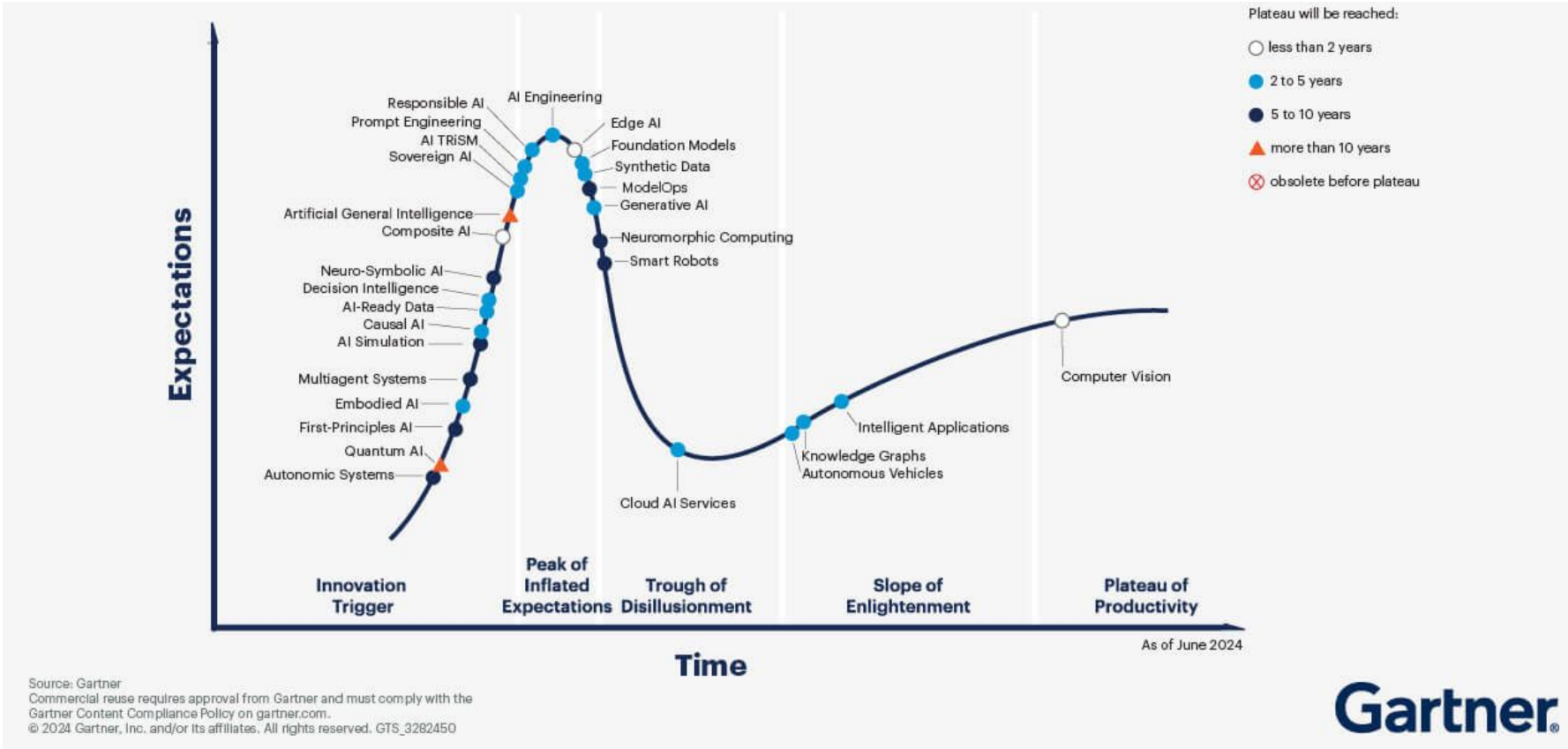
Trusted. Tested.
Transformative.

Our philosophy is to go faster by doing less, focusing only on what is necessary.

“Perfection is achieved, not when there is nothing more to add, but when nothing is left to take away.”

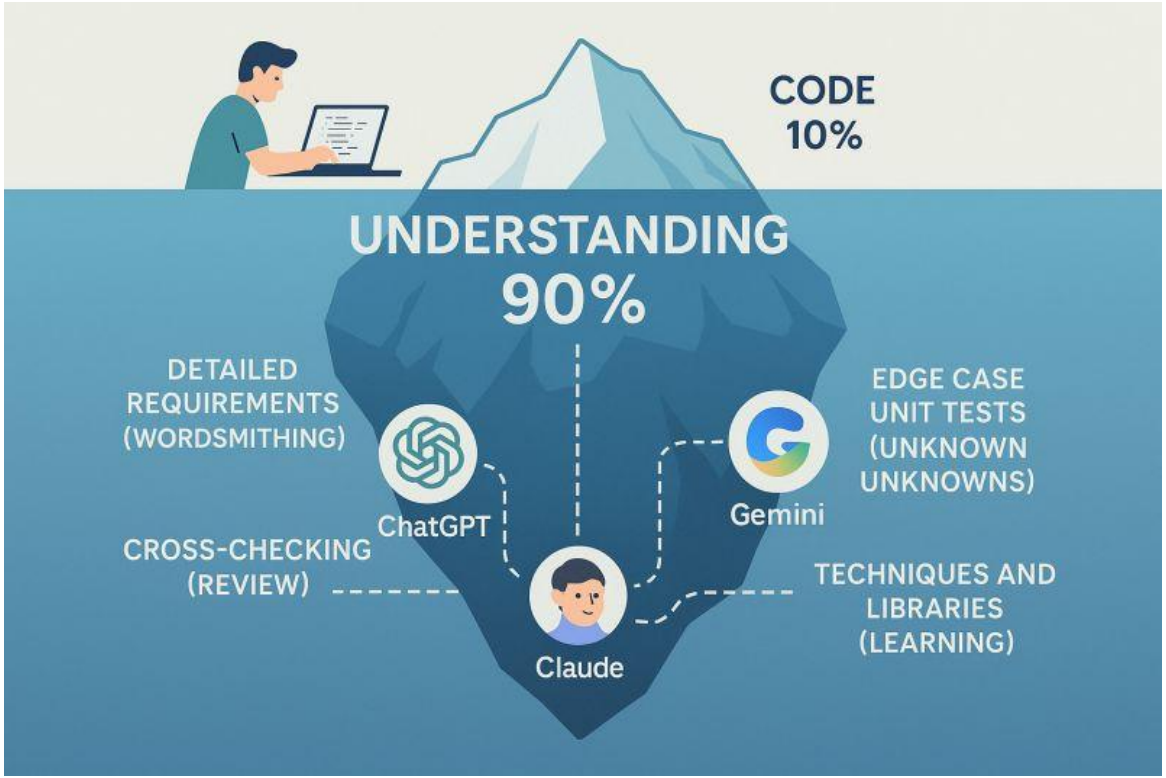
— Antoine de Saint-Exupéry, Airman's Odyssey

Gartner Hype Cycle for AI 2024 | Generative AI, 2 - 5 years away?



Gartner

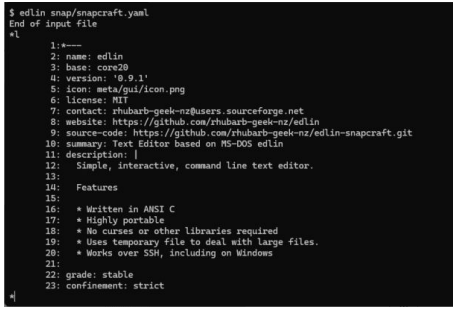
Understanding the Problem is 90% | Writing code is 10%



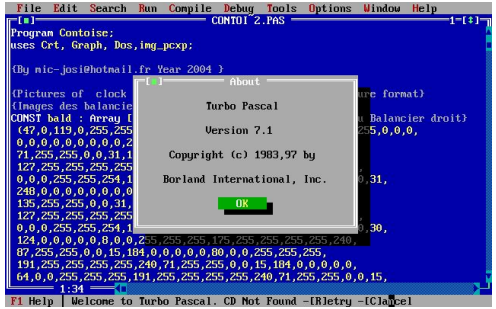
Chronicle SOFTWARE

Tooling | Many Enhancements

Edlin ‘84

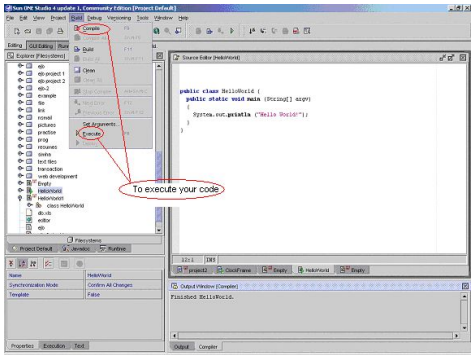


Turbo Pascal ‘87

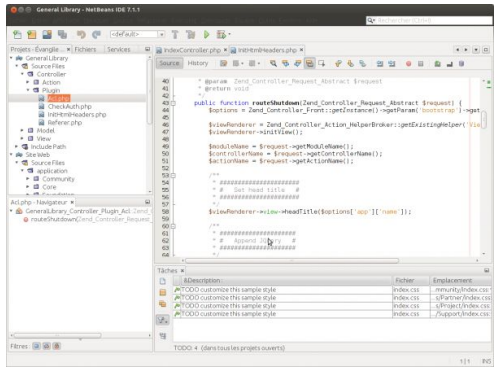


vi ‘89

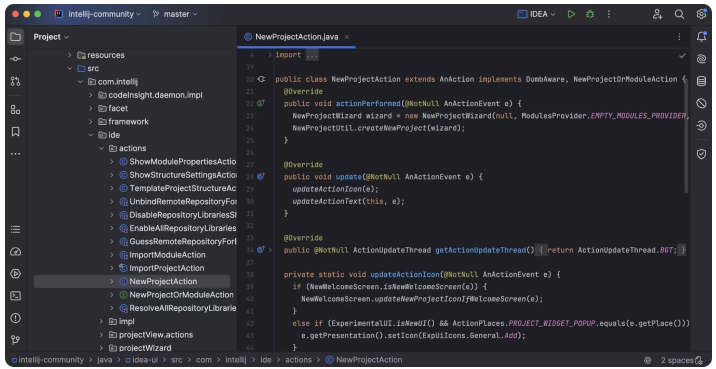
Forte for Java ‘99



NetBeans 2001



IntelliJ 2004



Chronicle SOFTWARE

From a simple prompt, AI can produce very different results.

https://blog.vanillajava.blog/2025/07/asking-multiple-ai-to-optimise-same-code.html

Given a more specific prompt they produce more consistent results

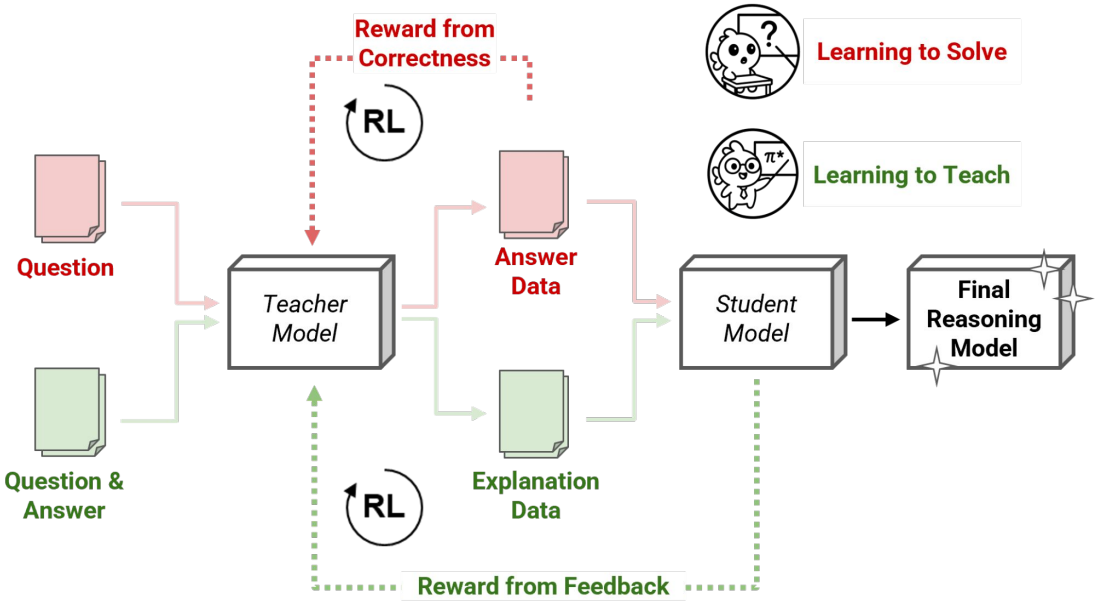
https://blog.vanillajava.blog/2025/07/improving-prompt-to-ai-to-get-better.html

As **AIs are statistical** based on enormous datasets, this can offer value in generalising knowledge. They **require significant review by an expert** to get quality results.

Generative AI follows an information path, relying on **contextual information.** They are very dependent on how you prompt them and what **you have provided or asked** before.

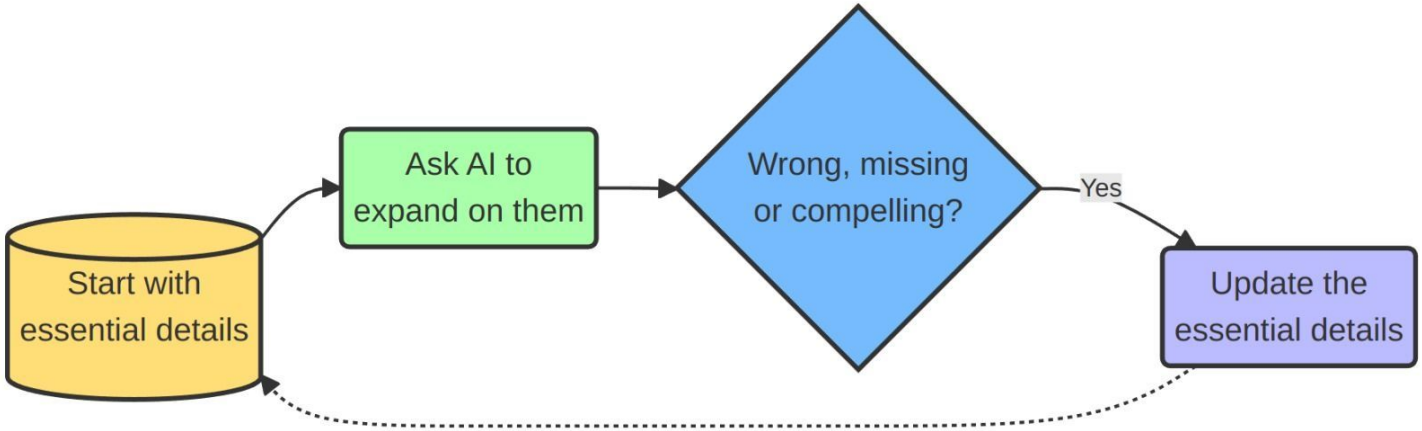
Learning By Teaching | Development is about Continuous Learning

The best way to learn is to teach others, also effective for AI. https://sakana.ai/rlt/



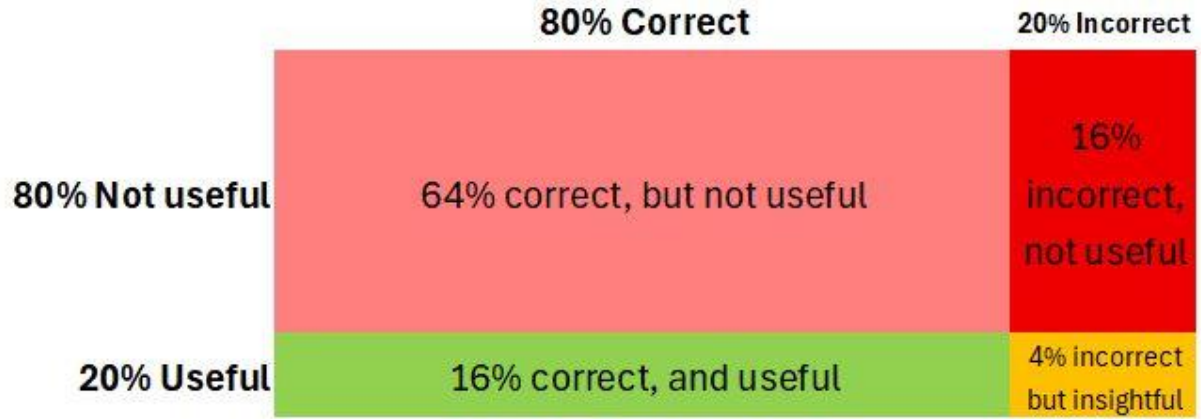
Chronicle SOFTWARE

We can use the same approach by focusing on improving the essential prompt as the retained product. Our messaging and documentation is concise and human moderated.



Correct vs Useful | Removing the fluff

Be ready to review and delete most of what AI produces. For moderate complex tasks, AI might be 80% correct, but only 20% useful.
Even when it's wrong it can be insightful



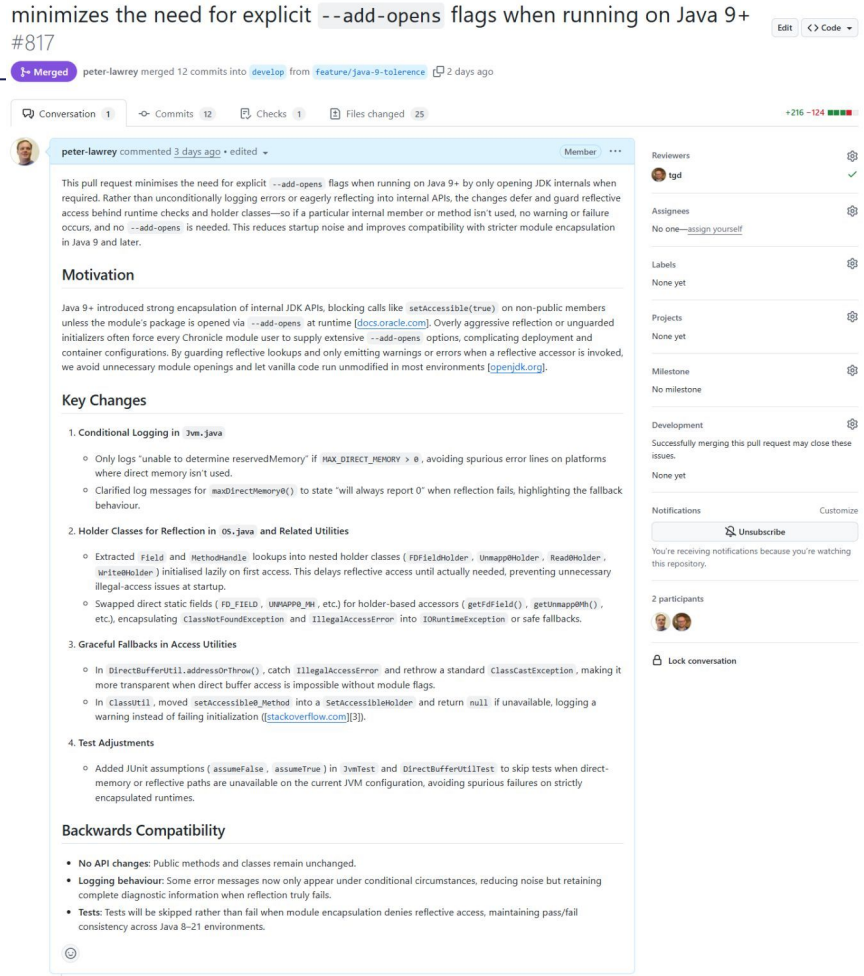
Chronicle SOFTWARE

Using Prompts as Meta-Programming | Some Tips

Stage	AI Generated Content
Prototype	When something is better than nothing, and it can produce a skeleton with very little effort.
Early Development	Saves time, but you will likely rewrite it. Requirements documentation helps keep the AI results stable and relevant.
Late Development	It gives you the bandwidth to focus on high level things, such as maintainability. You can trade time for quality.
Production / Support	Useful for covering gaps in functionality

Human-in-the-Loop | AI for Review

In github you can get a `git diff` by adding .diff to the PR. You can then ask AI for a pull request description adding a short description of your own.



minimizes the need for explicit `--add-opens` flags when running on Java 9+ #817

Edit · Code

Merged · peter-lawrey merged 12 commits into `develop` from `feature/java-9-tolerence` · 2 days ago

Conversation 1 · Commits 12 · Checks 1 · Files changed 25 · +216 −124

peter-lawrey commented 3 days ago • edited · Member

This pull request minimises the need for explicit `--add-opens` flags when running on Java 9+ by only opening JDK internals when required. Rather than unconditionally logging errors or eagerly reflecting into internal APIs, the changes defer and guard reflective access behind runtime checks and holder classes—so if a particular internal member or method isn't used, no warning or failure occurs, and no `--add-opens` is needed. This reduces startup noise and improves compatibility with stricter module encapsulation in Java 9 and later.

Motivation

Java 9+ introduced strong encapsulation of internal JDK APIs, blocking calls like `setAccessible(true)` on non-public members unless the module's package is opened via `--add-opens` at runtime [docs.oracle.com]. Overly aggressive reflection or unguarded initializers often force every Chronicle module user to supply extensive `--add-opens` options, complicating deployment and container configurations. By guarding reflective lookups and only emitting warnings or errors when a reflective accessor is invoked, we avoid unnecessary module openings and let vanilla code run unmodified in most environments [openjdk.org].

Key Changes

1. **Conditional Logging in `Jvm.java`**
   - Only logs "unable to determine reservedMemory" if `MAX_DIRECT_MEMORY > 0`, avoiding spurious error lines on platforms where direct memory isn't used.
   - Clarified log messages for `maxDirectMemory0()` to state "will always report 0" when reflection fails, highlighting the fallback behaviour.
2. **Holder Classes for Reflection in `OS.java` and Related Utilities**
   - Extracted `Field` and `MethodHandle` lookups into nested holder classes (`FDFieldHolder`, `Unmapp0Holder`, `Read0Holder`, `Write0Holder`) initialised lazily on first access. This delays reflective access until actually needed, preventing unnecessary illegal-access issues at startup.
   - Swapped direct static fields (`FD_FIELD`, `UNMAPP0_MH`, etc.) for holder-based accessors (`getFdField()`, `getUnmapp0Mh()`, etc.), encapsulating `ClassNotFoundException` and `IllegalAccessError` into `IORuntimeException` or safe fallbacks.
3. **Graceful Fallbacks in Access Utilities**
   - In `DirectBufferUtil.addressOrThrow()`, catch `IllegalAccessError` and rethrow a standard `ClassCastException`, making it more transparent when direct buffer access is impossible without module flags.
   - In `ClassUtil`, moved `setAccessible0_Method` into a `SetAccessibleHolder` and return `null` if unavailable, logging a warning instead of failing initialization ([stackoverflow.com][3]).
4. **Test Adjustments**
   - Added JUnit assumptions (`assumeFalse`, `assumeTrue`) in `JvmTest` and `DirectBufferUtilTest` to skip tests when direct-memory or reflective paths are unavailable on the current JVM configuration, avoiding spurious failures on strictly encapsulated runtimes.

Backwards Compatibility

- **No API changes:** Public methods and classes remain unchanged.
- **Logging behaviour:** Some error messages now only appear under conditional circumstances, reducing noise but retaining complete diagnostic information when reflection truly fails.
- **Tests:** Tests will be skipped rather than fail when module encapsulation denies reflective access, maintaining pass/fail consistency across Java 8–21 environments.

Reviewers: tgd

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

Development: Successfully merging this pull request may close these issues. None yet

Notifications: Customize · Unsubscribe · You're receiving notifications because you're watching this repository.

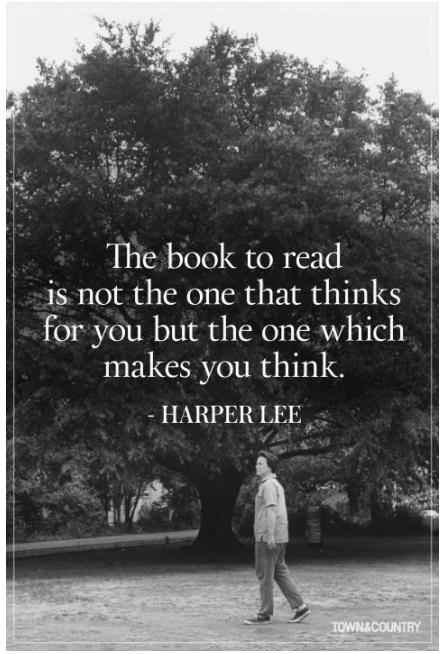
2 participants

Lock conversation

Human-in-the-Loop | Experts Still Fundamentally Required

All the approaches I have outlined require an expert

- to specify and review the requirements.
- for subjective or numeric analysis.
- to check the results.
- for domain knowledge.



Chronicle SOFTWARE

Human-in-the-Loop | Experts Still Fundamentally Required

AI enhance developer learning and enjoyment .

Understanding the problem, ensuring quality, and making architectural decisions remain human responsibilities

The developers who benefit most from AI are those who approach it with eyes open, ready to delete most of what it generates if needed, prepared to invest effort in guidance and verification, and determined to use AI to augment good engineering practices, not shortcut them.

Chronicle SOFTWARE

It can speed up repetitive work and provide inspiration, and it can make coding feel more enjoyable by offloading drudge.

But it doesn't fundamentally change the nature of software development as a thinking intensive, design centric activity. Writing code was never the true bottleneck; understanding requirements, devising solutions, and maintaining quality are where the real effort lies, and those are the aspects where human skill remains irreplaceable.