

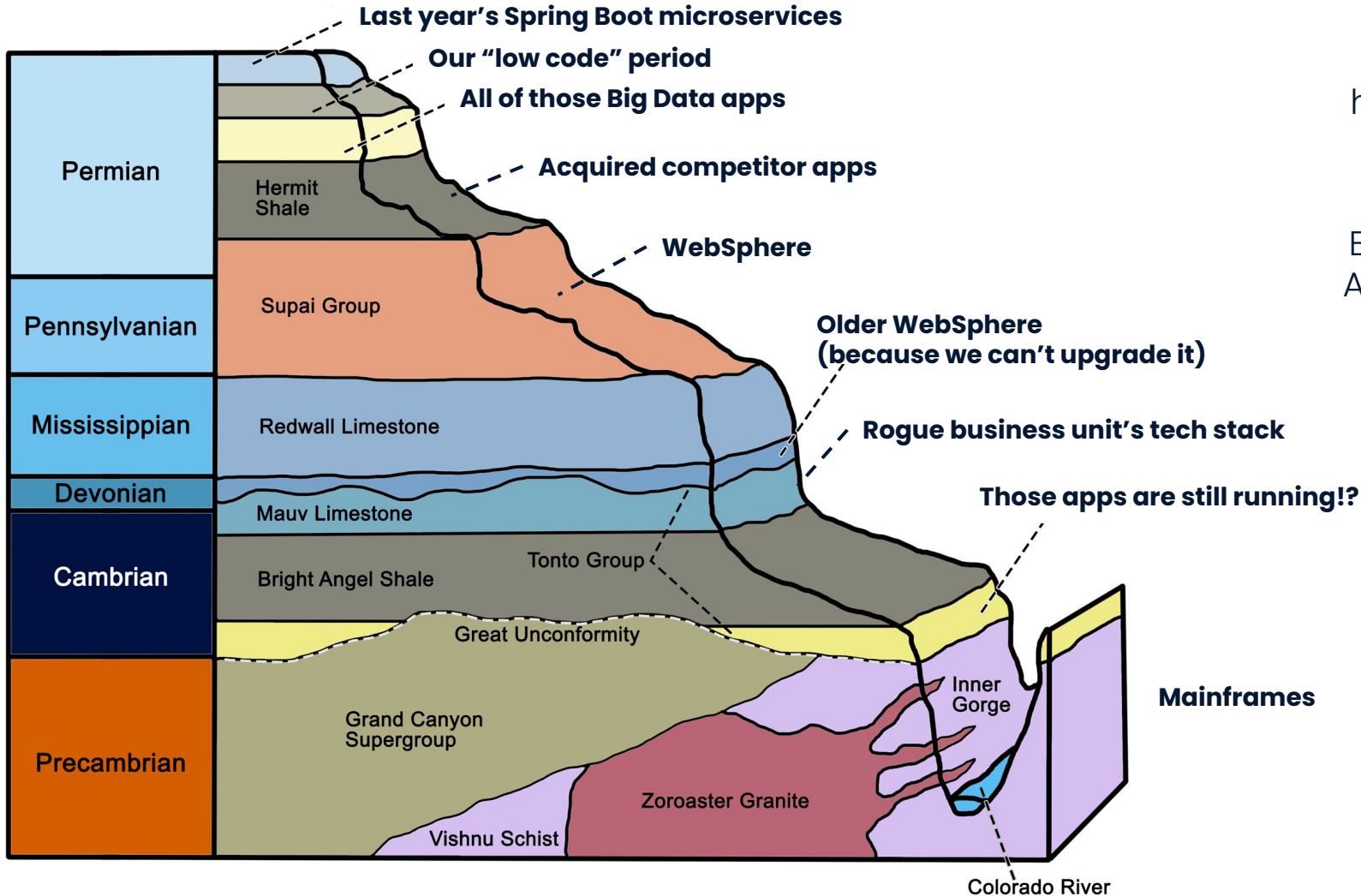
# DevTools or... Agent Tools?

April 2026





# An enterprise isn't just one application!



Our work is navigating thousands of heterogeneous interconnected apps without breaking things.

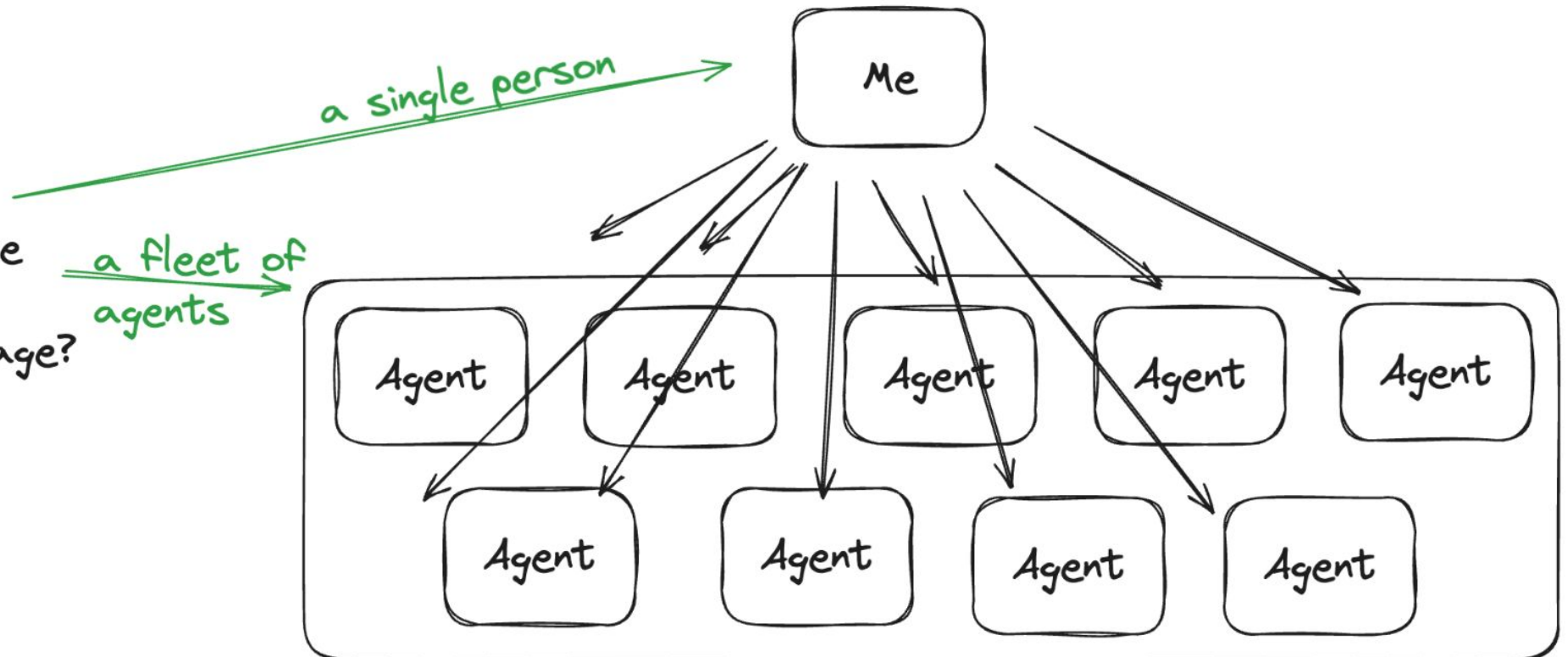
But things aren't completely random. An enterprise's portfolio accumulates and evolves in layers, following industry shifts, acquisitions, and leadership changes.

Original Source: <https://earthathome.org/ho/sw/rocks-cp/>



# The point of greatest leverage is shifting

If I could improve the efficiency of one only one of these, where is the point of greatest leverage?



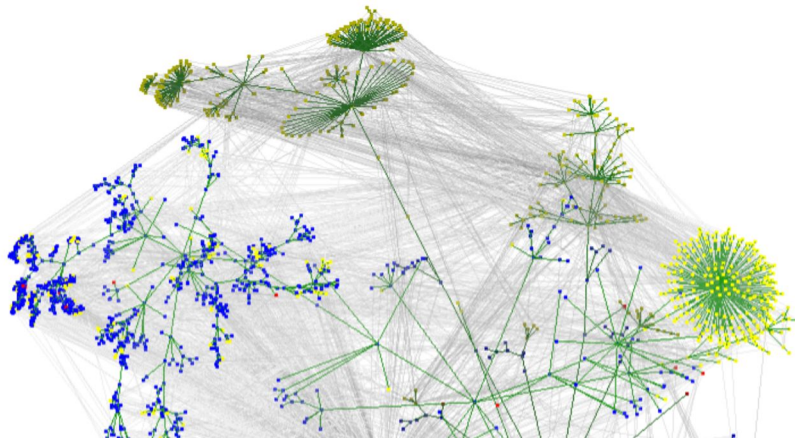


# Moderne: Foundational technology

## 1 Your code as structured data

### Lossless Semantic Tree (LST)

Compiler-accurate code model with type attribution, all dependencies, and other metadata resolved—how code should always be stored.



## 2 Customizable access to the data

### Programmable recipes

Structured queries in the form of recipes can easily be spun up by agents to safely search and manipulate the LST.

Recipes can be driven through Moderne automation or agents.

## 3 Multi-repo architecture

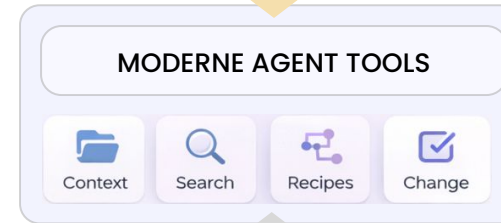
### Global and org-level code management

Distributed architecture and serialized LSTs enable visibility and control across entire codebases.

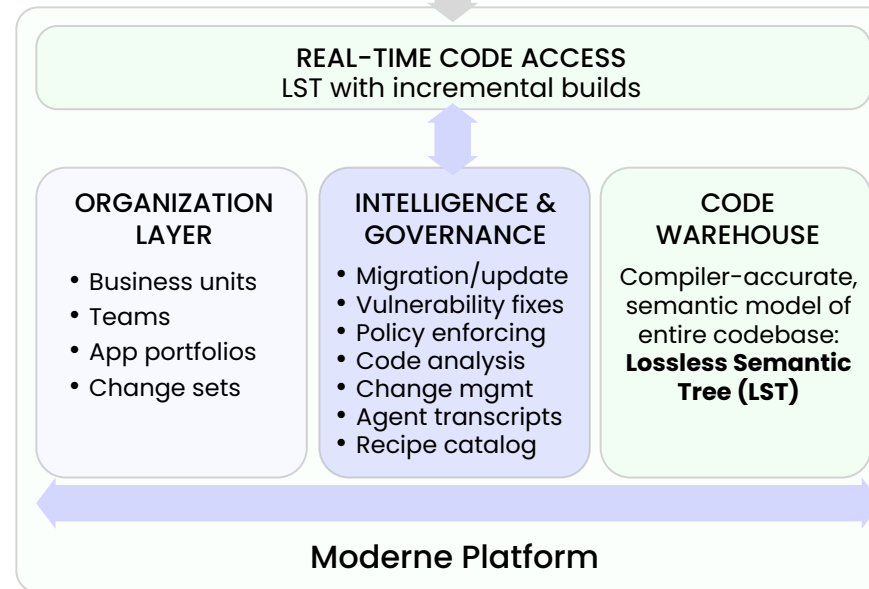


# Where modern software engineering operates

Scaling modernization, security, and AI-driven development through shared intelligence and governed execution.



*Enabling coding agents to work faster, more accurately, and more token-efficiently –replacing GPU for CPU.*



*Driving code intelligence, governance and transformation at enterprise scale.*

*Foundational code model for precise context and accurate change.*

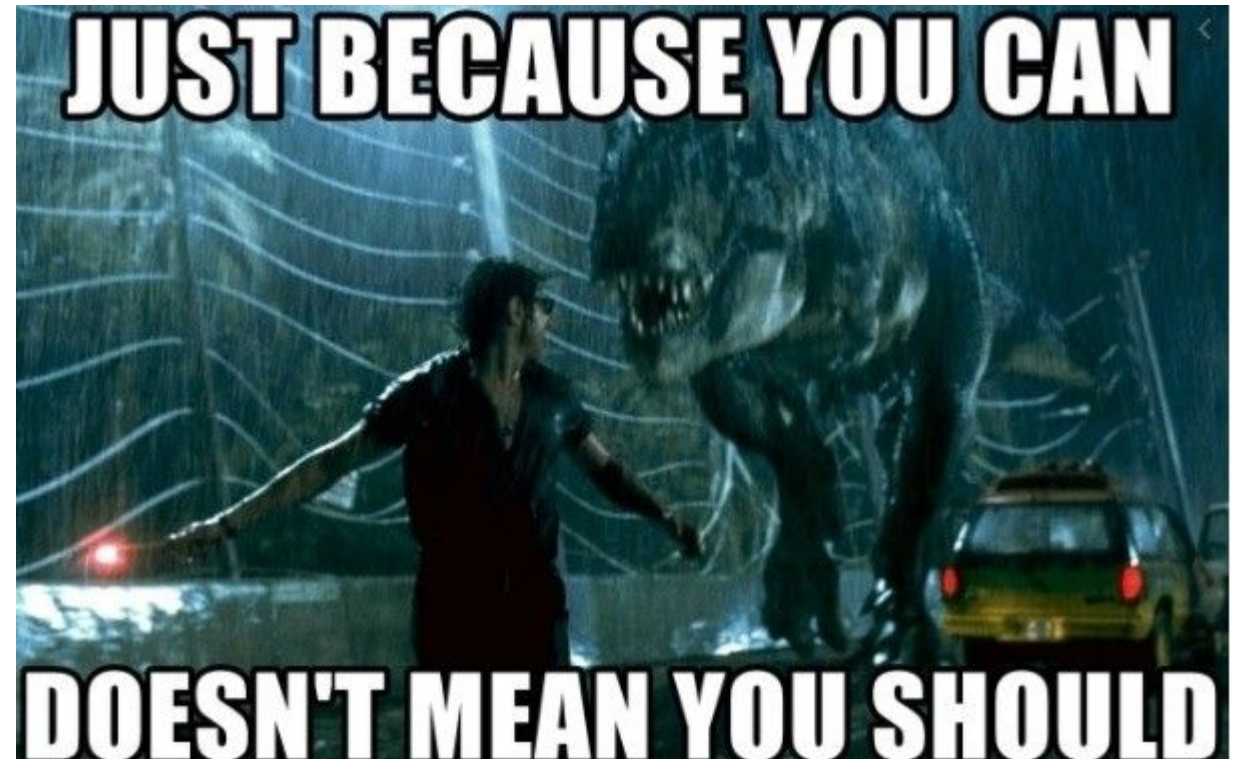


# Just because you can do it with an LLM...

We're starting to look for ways to exchange GPU for CPU...

to be a better friend to our agent by offloading some of its work onto not just cheaper models but better tools...

Just because we *can* throw something through the weights of a model, doesn't mean we always *should*.





# The coming price shock

Can everybody sense the trend?

Early 2025

Mid 2025

Early 2026

The screenshot shows a 'Plan usage limits' dashboard with three main sections: 'Current session', 'Weekly limits', and 'Extra usage'. Red arrows from the text 'Early 2025', 'Mid 2025', and 'Early 2026' point to the 'Current session', 'Weekly limits', and 'Extra usage' sections respectively.

Section	Usage	Resets
Current session	0% used	Resets in 3 hr 55 min
Weekly limits (All models)	30% used	Resets Thu 1:59 PM
Weekly limits (Sonnet only)	0% used	You haven't used Sonnet yet
Extra usage	8% used	Resets Mar 1

Additional details from the 'Extra usage' section:

- Turn on extra usage to keep using Claude if you hit a limit. [Learn more](#)
- \$41.74 spent
- Monthly spend limit: \$500
- Current balance: \$8.25 (with [auto-reload on](#))
- Buttons: [Adjust limit](#), [Buy more](#)



## MODERNE'S ADAPTATION:

An Agent Tools company is one that identifies tool gaps and closes them... so the GPUs can relax a bit.



ACT 1: “Prethink” context  
ahead of an interaction.



# Don't blame the agent... blame the context.

When we ask an agent to understand architectural patterns, patterns of dependency usage, etc. it requires a *lot* of grepping and reading.

It's one of the big sources of time and token waste.

```
> If I modify the Order entity, what other services might be affected

• I'll explore the codebase to understand the Order entity and its dependencies across services.

• Explore(Find Order entity dependencies)
  └ Done (55 tool uses · 60.3k tokens · 1m 40s)

* Finagling... (1m 52s · ↓ 6.4k tokens)

> █

esc to interrupt
```



# Don't blame the agent... blame the context.

On a regular cadence, before an agent session even begins, there is an opportunity to use CPU only static analyzers and other tooling to “prethink” out the relationships between components.

Recent activity > Results X6ZNY9ICR

Results for Default

### Update Prethink context (no AI)

Finished

Results | Repository status | Visualizations | Data tables

**Summary**

Started	2 minutes ago	Files changed	144
Running time	1m 13s	Repositories with results	11
Files searched	6,115	Est. time savings	18 hours

All | Success | Error | Search

**Results**

- finos (4 repositories)
  - symphony-bdk-java @ main (20 results)
  - spring-bot @ spring-bot-master (16 results)
  - messageml-utils @ main (14 results)
  - symphony-wdk @ master (3 results)
- Netflix (2 repositories)
  - ribbon @ master (17 results)
  - photon @ master (15 results)
- spring-projects (2 repositories)
  - spring-petclinic @ main (21 results)
  - spring-data-commons @ main (3 results)
- apache (1 repository)
  - maven-doxia @ master (14 results)

Commit all results

finos/symphony-bdk-java @ main

Filter | Settings | Collapse all | Commit only selected results

.moderne/context/caLm-architecture.json

Large diffs are not displayed by default. Load diff

.moderne/context/architecture.md

```
@@ -0,0 +1,100 @@
1 + # Architecture
2 +
3 + ## FINOS CALM architecture diagram
4 +
5 + FINOS CALM (Common Architecture Language Model) architecture diagram showing services,
6 + level system architecture and component relationships.
7 +
8 + ## Data Tables
9 +
10 + ### Data assets
11 + **File:** [ `data-assets.csv` ](data-assets.csv)
12 +
```



# Don't blame the agent... blame the context.

The end result is both a quicker result...

- ▶ If I modify the Order entity, what other services might be affected
- I'll check the Moderne Prethink architecture context to understand service dependencies related to the Order entity.
- Read 2 files (ctrl+o to expand)
- Let me search the Prethink context files for Order-related entries.
- Searching for 4 patterns... (ctrl+o to expand)
- ★ **Crafting...** (thought for 5s)



# Don't blame the agent... blame the context.

```
Order events would require updates
```

---

```
Service: Payment  
Component: PaymentOrderStatusVm  
Why Affected: Contains orderId, orderStatus – changes to order status handling could break integration
```

---

```
Service: Rating  
Component: OrderExistsByProductAndUserGetVm  
Why Affected: Queries order existence by product/user – depends on order data structure
```

```
Messaging Dependency
```

```
The webhook service subscribes to the order Kafka topic:  
Topic: ${webhook.integration.kafka.order.topic-name}  
Consumer: OrderEventInbound.onOrderEvent()
```

```
Any changes to the Order event schema would require corresponding updates in the webhook service.
```

```
Recommendation
```

```
Before modifying the Order entity:  
1. Review all OrderVm, OrderPostVm, and related DTOs in the order service  
2. Check the Kafka event schema consumed by the webhook service  
3. Verify payment service integration via PaymentOrderStatusVm  
4. Confirm rating service's order existence check remains compatible
```

...and fewer tokens.

```
└ Context Usage
```

```
🌀 🌀 🌀 🌀 🌀 claude-opus-4-5-20251101 · 29k/200k tokens (15%)
```

```
🌀 [ ] [ ] [ ] [ ] [ ]
```

```
[ ] [ ] [ ] [ ] [ ] Estimated usage by category
```

```
[ ] [ ] [ ] [ ] [ ] 🌀 System prompt: 2.8k tokens (1.4%)
```

```
[ ] [ ] [ ] [ ] [ ] 🌀 System tools: 15.9k tokens (7.9%)
```

```
[ ] [ ] [ ] [ ] [ ] 🌀 Memory files: 413 tokens (0.2%)
```

```
[ ] [ ] [ ] [ ] [ ] 🌀 Skills: 816 tokens (0.4%)
```

```
[ ] [ ] [ ] [ ] [ ] 🌀 Messages: 10.8k tokens (5.4%)
```

```
[ ] [ ] [ ] [ ] [ ] [ ] Free space: 136k (68.2%)
```

```
[ ] [ ] [ ] [ ] [ ] [ ] [ ] Autocompact buffer: 33.0k tokens (16.5%)
```



# ACT 2: Better search tools



# “Trigrep” – Trigram Search

Most coding agents use either internal grep (ripgrep or “rg”) or a vector search.

Trigram based searches offer a much faster alternative if their indexes can be maintained.

Execution Time



**338ms**

Avg Grep Time

**25ms**

Avg Trigrep Sim Time

**13.5x**

Avg Speedup



# “Trigrep” – Trigram Search

Expensive subsequent reads happen because while a grep or vector search narrows the search scope textually, the agent must still prove to itself that this method has a *receiver* type that it is looking for.

```
static {  
    if (Platform.isMac() || Platform.isLinux()) {  
        TICKS_PER_SEC = ParseUtil.parseLongOrDefault(Execu  
            defaultLong: -1L);  
        if (TICKS_PER_SEC == -1) {  
            LOG.info("Unable to determine ticks per second  
        }  
    } else {  
        LOG.info("Cumulative cpu time in seconds is unsupp  
    }  
}
```



# Lossless Semantic Tree (LST) access patterns

“Find Spring API endpoints”

Rather than using a grep, the agent is using a smarter tool.

It has exchanged GPU for CPU.

```
GitHub Copilot v0.0.415
Copilot uses AI. Check for mistakes.

• ⚡ No copilot instructions found. Run /init to generate a copilot-instructions.md file for this project.
• Loaded env: 1 MCP server, 1 skill
• Loaded env: 1 MCP server, 1 skill
> Find Spring API endpoints
⦿ The user wants to find Spring API endpoints in this Spring PetClinic project. Let me use the code intelligence tools to find controller annotations and endpoint mappings.
• find_annotations
  ↳ {"recipe": "org.openrewrite.java.search.FindAnnotations", "searchResults": [{"fi...
• find_annotations
  ↳ {"recipe": "org.openrewrite.java.search.FindAnnotations", "searchResults": [{"fi...
• find_annotations
  ↳ {"recipe": "org.openrewrite.java.search.FindAnnotations", "searchResults": [{"fi...
```

Uses an OpenRewrite recipe rather than an expensive grep + read



# Lossless Semantic Tree (LST) access patterns

The result is accurate, fast,  
and (token) cheap.

The recipe often possesses  
far greater semantic  
understanding of the code  
pattern than would be  
apparent from just a  
grep+read.

- Here are all the Spring API endpoints in the PetClinic application:

## OwnerController

Method	Path	Handler
GET	/	welcome()
GET	/owners/new	initCreationForm()
POST	/owners/new	processCreationForm()
GET	/owners/find	initFindForm()
GET	/owners	processFindForm()
GET	/owners/{ownerId}/edit	initUpdateOwnerForm()
POST	/owners/{ownerId}/edit	processUpdateOwnerForm()
GET	/owners/{ownerId}	showOwner()

## PetController (@RequestMapping("/owners/{ownerId}"))

Method	Path	Handler
GET	/owners/{ownerId}/pets/new	initCreationForm()
POST	/owners/{ownerId}/pets/new	processCreationForm()
GET	/owners/{ownerId}/pets/{petId}/edit	initUpdateForm()
POST	/owners/{ownerId}/pets/{petId}/edit	processUpdateForm()



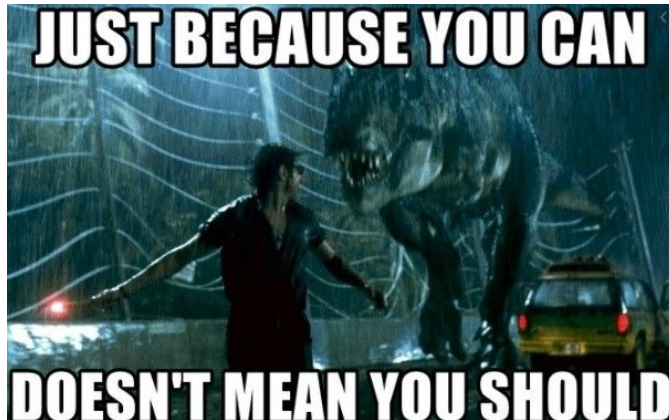
# ACT 3: Better editing tools



# Lossless Semantic Tree (LST) editing patterns

Something like “Upgrade to Java 25” is a complex migration that *can* be accomplished by an agent, but at what cost?

Remember...





# Lossless Semantic Tree (LST) editing patterns

Rather than grepping, reading, and editing its way through thousands of files, a coding agent can reach for one of the thousands of OpenRewrite or Moderne recipes that do small and big things well.

```
- Repository: /Users/jon/conductor/workspaces/rewrite/seattle-v2
- LST store: .moderne/mcp/lst

> Upgrade to Java 25

● moderne - search_recipes (MCP)(query: "upgrade to Java 25")
  └─ {
    "query" : "upgrade to Java 25",
    "totalCount" : 53,
    ... +44 lines (ctrl+o to expand)

● Found the recipe. Let me get the details and then run it.

● moderne - learn_recipe (MCP)(recipe: "org.openrewrite.java.migrate.UpgradeToJava25")
  └─ {
    "name" : "org.openrewrite.java.migrate.UpgradeToJava25",
    "displayName" : "Migrate to Java 25",
    ... +38 lines (ctrl+o to expand)

● Running the Migrate to Java 25 recipe now. This is a comprehensive recipe with replacements, and plugin upgrades.

moderne - run_recipe (MCP)(recipe: "org.openrewrite.java.migrate.UpgradeToJava25")
  └─ Preparing recipe...
     └─ ██████████ 40%

* Envisioning... (thinking)
  └─ Tip: Press Shift+Enter to send a multi-line message
```



# Lossless Semantic Tree (LST) editing patterns

~3 minutes and 30k tokens later, the migration is complete.

● The recipe completed successfully. Here's a summary of what Migrate to Java 25 applied across 709 files (+5,763 / -6,856 lines):

Key Transformations

Recipe	Files	Description
InstanceOfPatternMatch	352	x instanceof Foo + cast → x instanceof Foo foo
ReplaceUnusedVariablesWithUnderscore	257	Unused lambda params/catch vars → _
PathsGetToPathOf	186	Paths.get(...) → Path.of(...)
StringFormatted	140	String.format(...) → "...".formatted(...)
OptionalNotPresentToIsEmpty	75	!optional.isPresent() → optional.isEmpty()
UseTextBlocks	50	Multi-line string concatenation → text blocks
SwitchCaseReturnsToSwitchExpression	24	Switch statements → switch expressions
SwitchCaseAssignmentsToSwitchExpression	18	Switch assignments → switch expressions
MigrateMainMethodToInstanceMain	18	public static void main → instance void main()
SequencedCollection	15	Use new SequencedCollection APIs
IteratorNext	15	Iterator patterns → modern alternatives
ReplaceSystemOutWithIOPrint	13	System.out.print → Java 25 IO.print
UpgradeJavaVersion	build files	Java source/target → 25

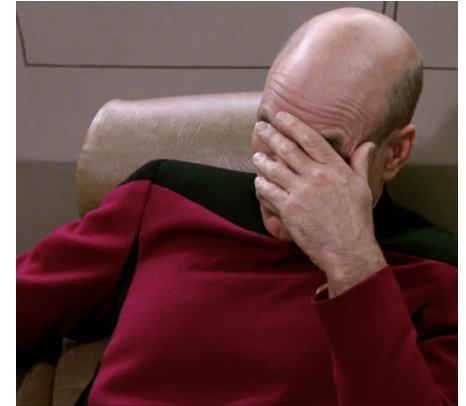
\* Brewed for 2m 41s



# Lossless Semantic Tree (LST) editing patterns

I asked the most expensive frontier model to do the same work *without* code intelligence tools.

It declared success after updating some dependencies and not touching the source code. Which is *ok*. I can just keep saying “do more”.

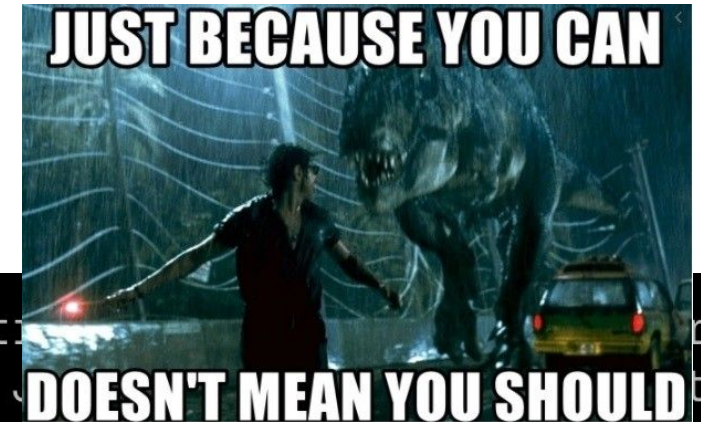


```
> You missed the bulk of the work which is to uplift all of the source code to the latest Java 25 coding patterns
● You're right – the build dependency updates are just the trivial part. The real work is updating the parser implem
  handle new Java 25 language features. Let me explore what Java 25 brings and what the current rewrite-java-25 modu
+ Accomplishing
```



# Lossless Semantic Tree (LST) editing patterns

Hundreds of thousands of tokens, and perhaps as important ***time*** get burned up trying to figure out what the recipe already knows.



- You're right – the build dependency updates are just the tip of the iceberg. I need to handle new Java 25 language features. Let me explore what's new in the JDK. The c
- Running 3 agents... (ctrl+o to expand)
  - └ Explore (Explore rewrite-java-25 source code) · 33 tool uses · 76.8k tokens
    - └ Done
  - └ Task (Research Java 25 language features) · 21 tool uses · 49.1k tokens
    - └ Fetch: <https://openjdk.org/jeps/512>
  - └ Explore (Diff Java 21 vs 25 parser visitors) · 21 tool uses · 37.6k tokens
    - └ Done



# Lossless Semantic Tree (LST) editing patterns

And then we spend another couple hundred thousand tokens...

- Got it – the real task is modernizing the Java source code across *all* modules (text blocks, switch expressions, records, etc.). That's a very different and large scope.
- Entered plan mode  
Claude is now exploring and designing an implementation approach.
- 2 Explore agents finished (ctrl+o to expand)
  - └ Explore Java 8 baseline config · 42 tool uses · 68.2k tokens
    - └ Done
  - └ Survey Java 8 patterns in codebase · 41 tool uses · 68.3k tokens
    - └ Done



# Lossless Semantic Tree (LST) editing patterns

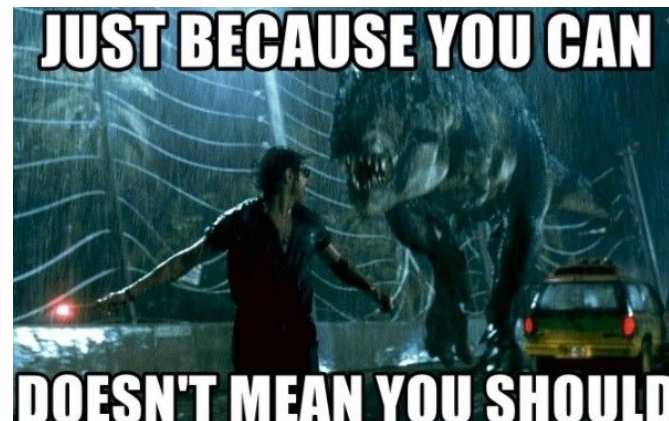
## Execution strategy

Work one pattern at a time across the entire codebase using grep/replace, then verify compilation. This is more efficient than module-by-module because each pattern is a consistent mechanical transformation.

For each pattern:

1. Grep for occurrences across all modules
2. Apply transformations file by file
3. Run `./gradlew assemble` to verify

*Oh no, not grep again!*



Remember...





# Lossless Semantic Tree (LST) editing patterns

I gave up after 45 minutes. I know the agent could continue plowing through this.

- All four main tasks are complete. Let me review what remains from the plan - switch expressions and text blocks were listed as lower priority patterns. Let me check if there are any obvious switch statement candidates.
- User answered Claude's questions:
  - └ The main Java 25 upgrade patterns are done (instanceof, Collections factories, .toList(), .formatted()). The plan also listed switch expressions and text blocks as lower priority. Do you want to continue with those, or is this good? → Continue with all remaining



# Lossless Semantic Tree (LST) editing patterns

But we had already burned through 61M tokens so far.

The point is not that an LLM shouldn't drive the migration, it should do as much heavy lifting with a recipe first and then mop up the leftovers with GPU.

And where there's repeatability to a migration, there is a reason to save off recipes and share them.

Source	Turns	Input Tokens	Output Tokens
Main conversation	431	46.2M	50k
Streaming events	328	14.3M	2k
Subagents	12	643k	24k
<b>Total</b>	<b>771</b>	<b>61M</b>	<b>76k</b>

*For incomplete work*



# ACT 4: Identify tool gaps by paying attention to transcripts



# Mining data from chat transcripts

Remember this chart?

It's hard to get an agent to go against its system prompt.

One trick is to simply measure the alternative whenever you see the agent make a suboptimal choice.

Then... *BRING THE RECEIPTS!*

The agent eventually listens.

Execution Time





Thank you

Modernize, secure, and maintain  
your entire codebase with Moderne