AI Agents and Software Engineering

CS350 Introduction to Software Engineering

Shin Yoo


Discussions and forums
Is Software Engineering dead ?
Reddit · r/careerguidance · 10+ comments · 6 months ago

Hmm.

tl;dr: Not dead, but changing very fast.

And here are some relevant points.

- Professional Angle
  - SE was not about generating code anyway.
  - To really benefit from agents, you will need SE knowledge.
- Research Angle
  - Now with the help from agents, there are more software to build.
  - We need to cover a vast new ground provided by LLMs and agents.
- Whenever things move fast, chances are you have to work harder :)

SE was not about generating code, anyway

…because coding actually takes up not much…?

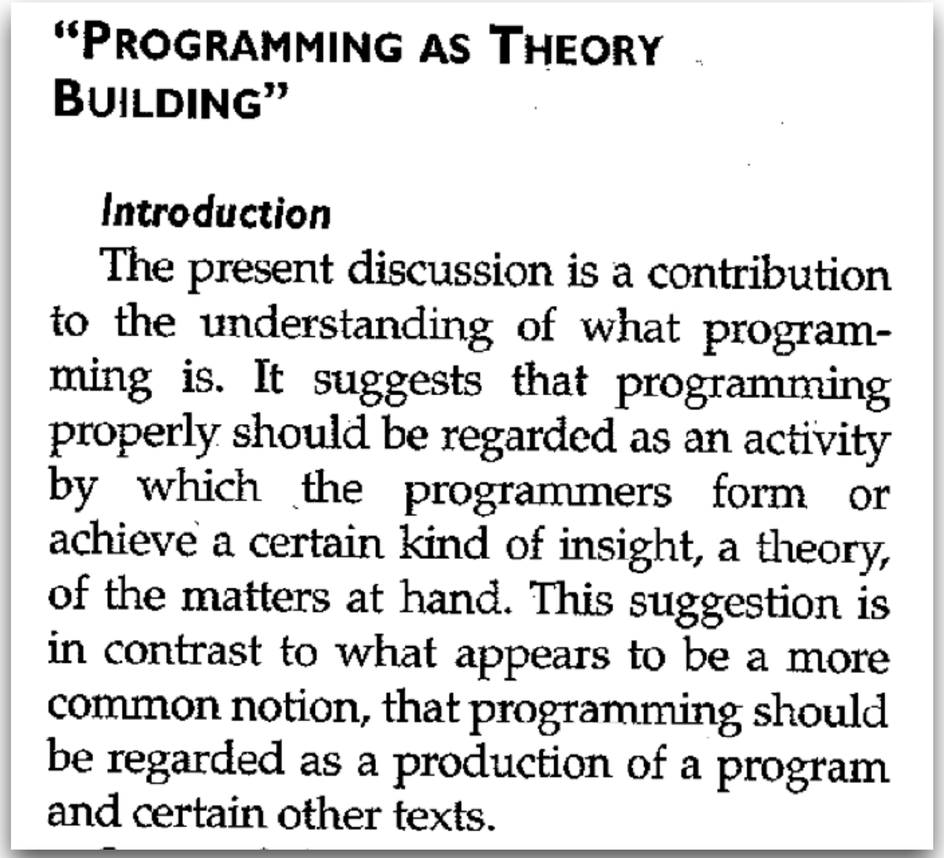
For some years I have been successfully using the following rule of thumb for scheduling a software task:

⅓ planning
⅙ coding
¼ component test and early system test
¼ system test, all components in hand.

SE was not about generating code, anyway

…because the real objective is not coupled to code…?

- Peter Naur's Theory Building
- Programming is not production of programs.
- Programming is the activity by which programmers form or achieve a certain kind of insight or theory.



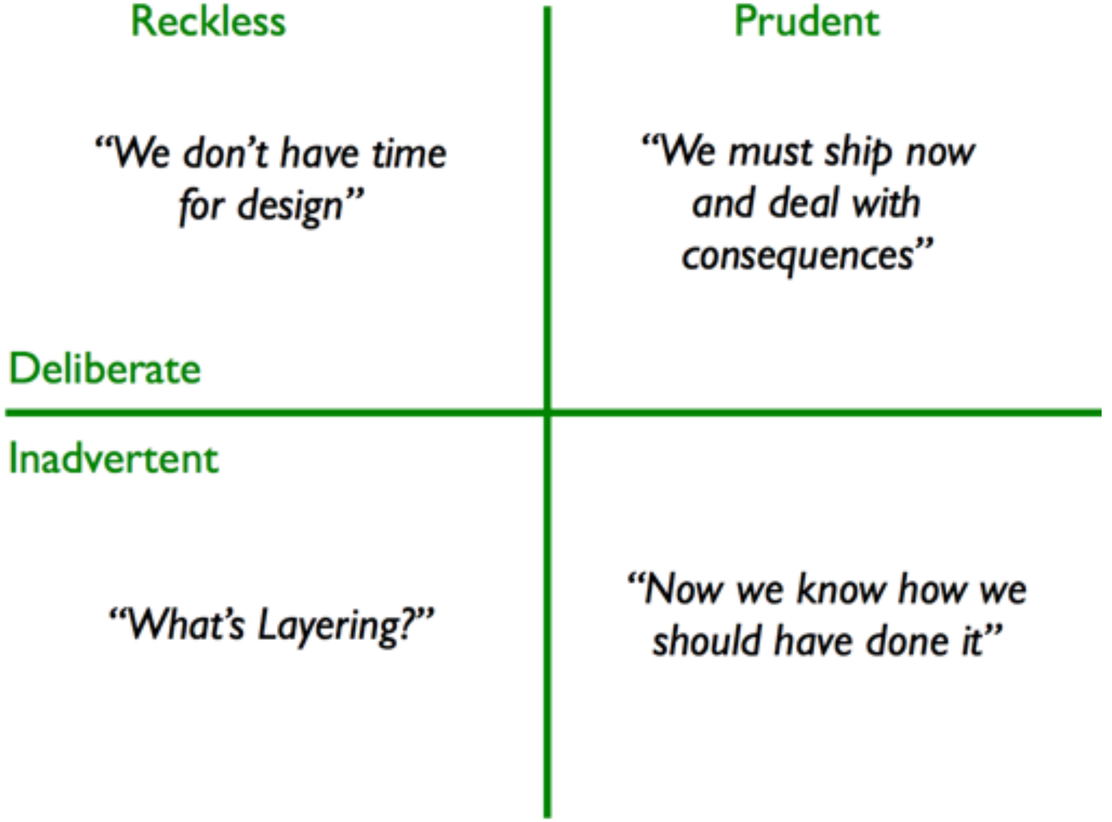
"PROGRAMMING AS THEORY BUILDING"

Introduction

The present discussion is a contribution to the understanding of what programming is. It suggests that programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is in contrast to what appears to be a more common notion, that programming should be regarded as a production of a program and certain other texts.

Technical Debt

- Technical debt (also known as design debt or code debt) is a qualitative description of the cost to maintain a system that is attributable to choosing an expedient solution for its development (Wikipedia).
- Prudent/Reckless, Deliberate/ Inadvertent quadrant (Martin Fowler)



https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

Cognitive & Intent Debt

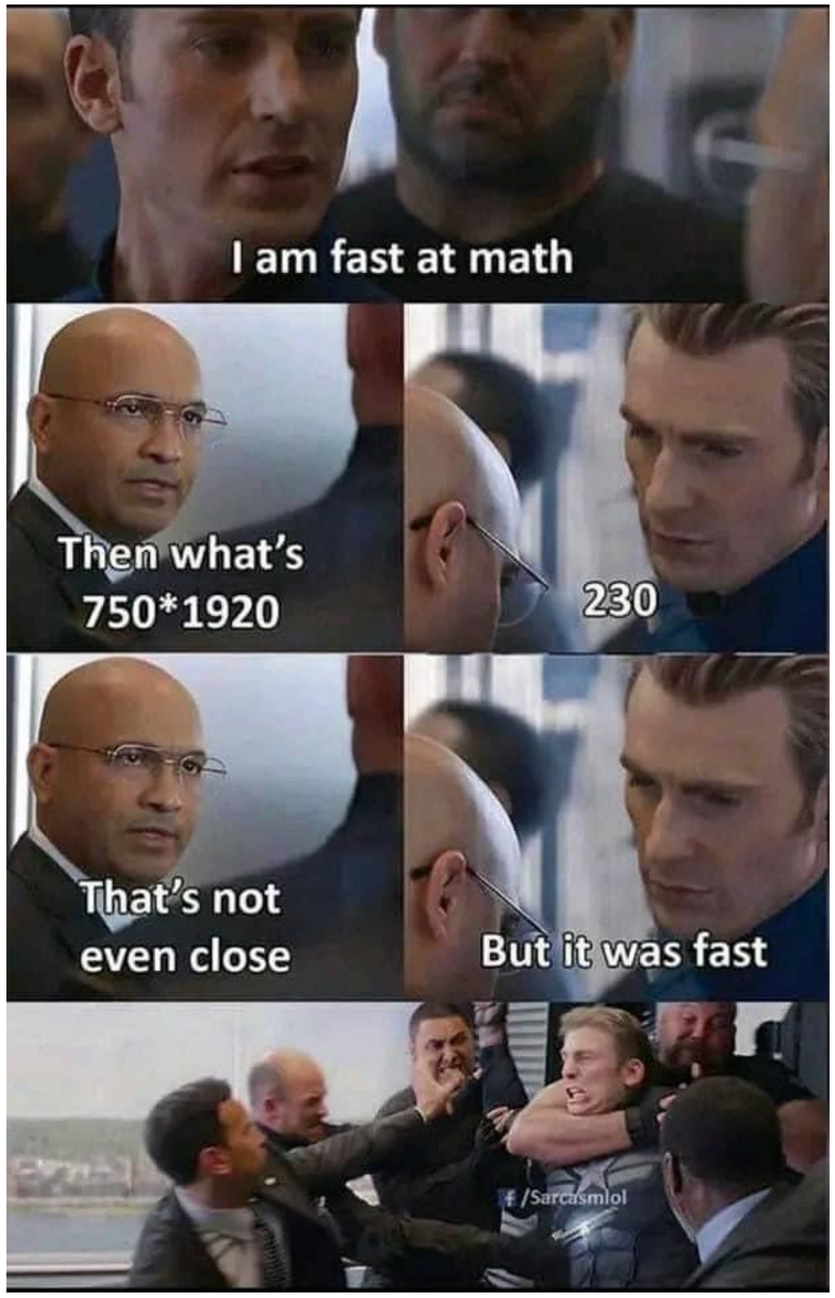
Storey, 2026 (https://arxiv.org/abs/2603.22106)

- Cognitive Debt: “erosion of shared understanding across a team”
- Intent Debt: “absence of externalized rationale that developers and AI agents need to work safely with code”

Triple Debt Model

Storey, 2026 (https://arxiv.org/abs/2603.22106)

- **Technical debt** lives in code. It accumulates when implementation decisions compromise future changeability. It limits how systems can change.
- **Cognitive debt** lives in people. It accumulates when shared understanding of the system erodes faster than it is replenished. It limits how teams can reason about change.
- **Intent debt** lives in artifacts. It accumulates when the goals and constraints that should guide the system are poorly captured or maintained. It limits whether the system continues to reflect what we meant to build and it limits how humans and AI agents can continue to evolve the system effectively.


I am fast at math
Then what's 750*1920
230
That's not even close
But it was fast
/Sarcasmlol

Economic Argument

- A new technology initially threats human jobs, but in the end creates more jobs: a task becoming easier means more people now want to do it.
  - https://newsletter.pragmaticengineer.com/p/state-of-the-job-market-2026
- Token-maxxing may not be sustainable: to optimize the cost-benefit, you need to understand your task and use structured harnesses - simply letting agents to do their jobs won’t be enough.
  - https://fortune.com/2026/05/28/tokenmaxxing-is-dead-companies-didnt-get-the-roi-from-ai-they-wanted-to-see/

The Importance of SE Knowledge

- Coding agents are not perfect yet - you cannot push a button and get everything you want (otherwise, token-maxing would’ve worked).
- They are fast in generating code, and dramatically more accurate than any previous code generation technique (evolutionary computation, grammar-based program synthesis…). However, they lack sufficient level of domain knowledge and design capabilities.
- It would be much more powerful if someone can guide them properly :)

Claude Code Tools

(a broad summary)

- Git Worktree Management
- Read (cat, head, tail)
- String edit
- Search: glob, grep, LSP
- Task management
- User interaction
- Skills & Sub-agents

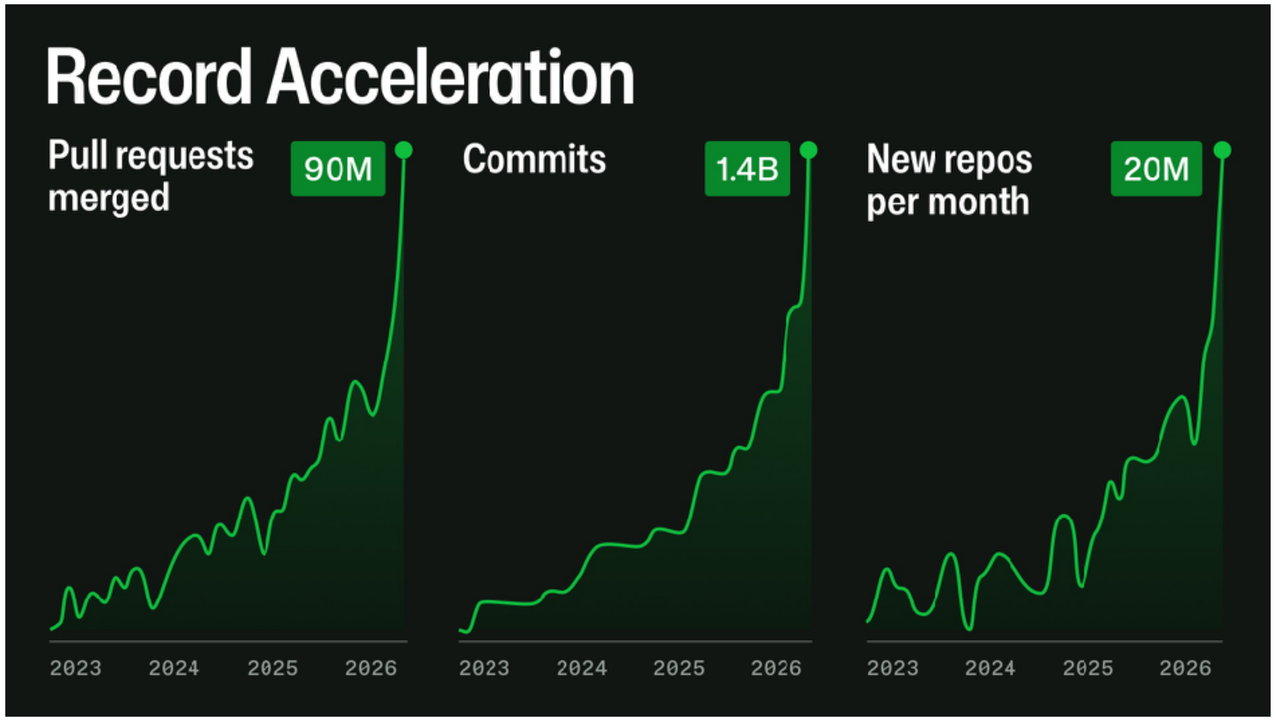
In some sense, all low level actions.
We delegate actual engineering decisions to the pertaining of the foundation models.

You can extend the tool library here but knowledge of specific techniques would help a lot.

More SW to Build

First, more of the same

- We will have more of everything.
- Increasingly, the bottleneck is the human (especially code review).
- How will we adapt to this scale?

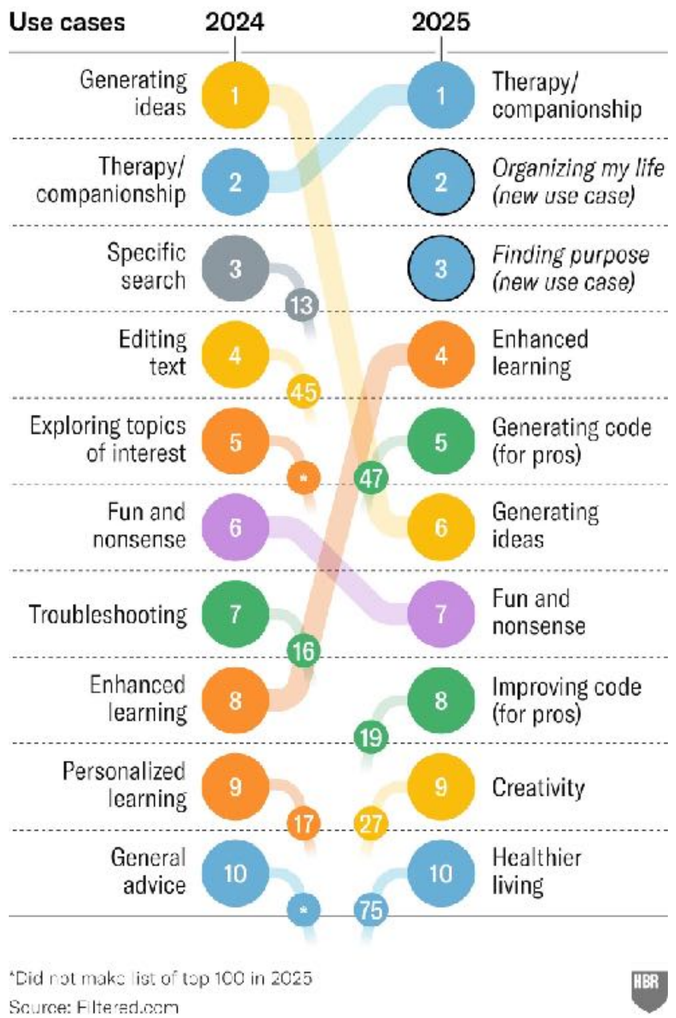


https://github.blog/news-insights/company-news/an-update-on-github-availability/

More SW to Build

Second, more of what we do not know about

- There will be completely different types of software: agents are actually the representative example!
- LLMs as interface will force us to consider both old and new quality criteria: correctness (under randomness and vagueness), safety (emotional and psychological)…



“[Natural language prompts | Agents | Specs] is the **new layer of abstraction.**”

A compromise that seems to appease to both the machinists and the artisans.

New Layer, Different Reactions

Compilers / HPL

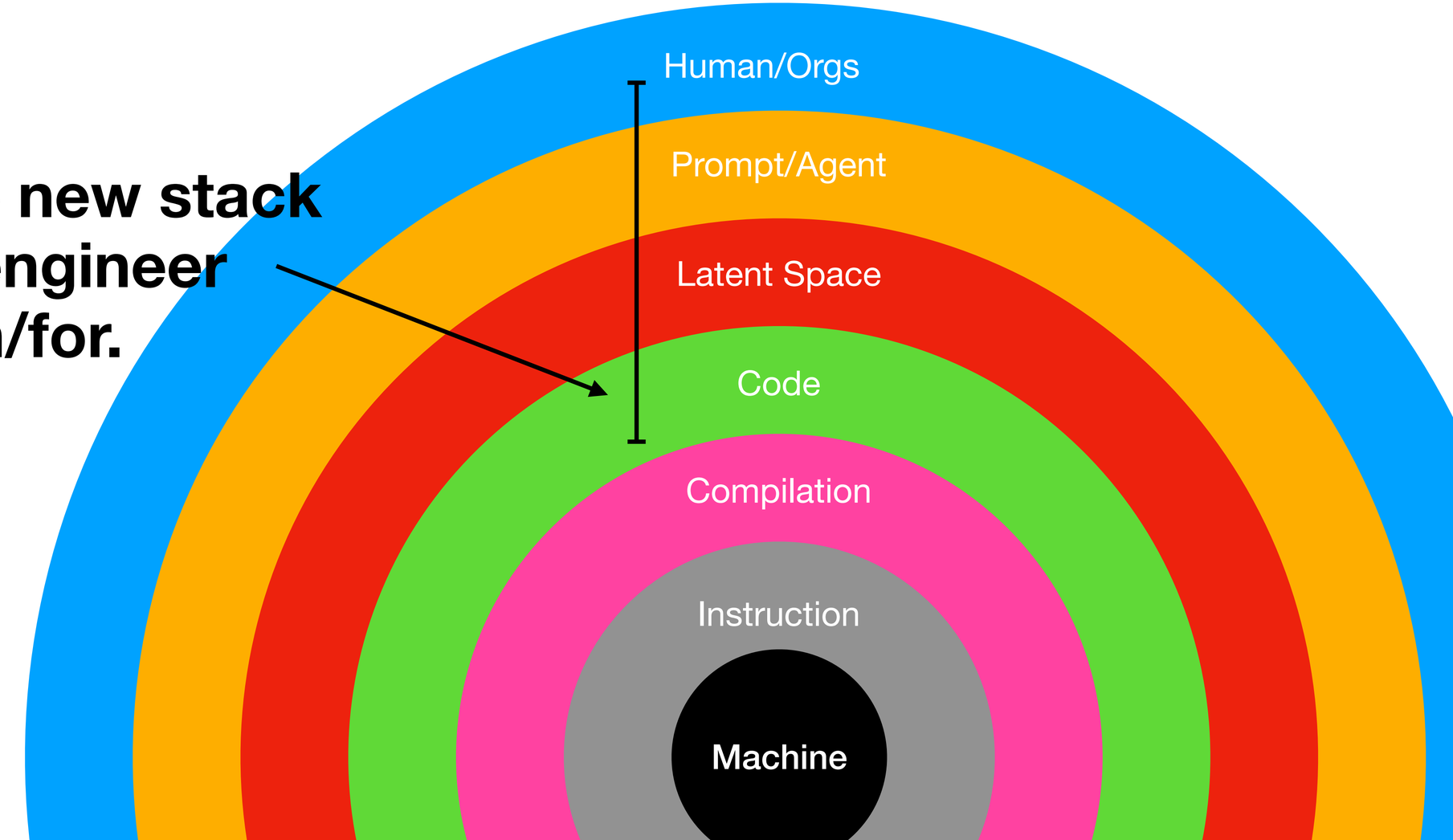
- Initially not entirely popular 🤓
- Rigorous research and **engineering** went into **compilers**.
- The whole layer is **traceable**, even **verifiable**.
- Program analysis **techniques that check the outcomes**.

LLMs / Agents

- Even better than sliced bread!
- **Vibes** seem to play a critical role in the whole discourse.
- Evaluation focuses solely on **end-to-end** input/output, skipping the layer itself.
- Testing of the new layer remains largely **unexplored**.

Our playground got so much bigger!

This is the new stack we must engineer in/through/for.



(New) Targets of Investigation

Application Layer

- Completely new types of software
- Human impact
- Everything executable

Trajectory Layer

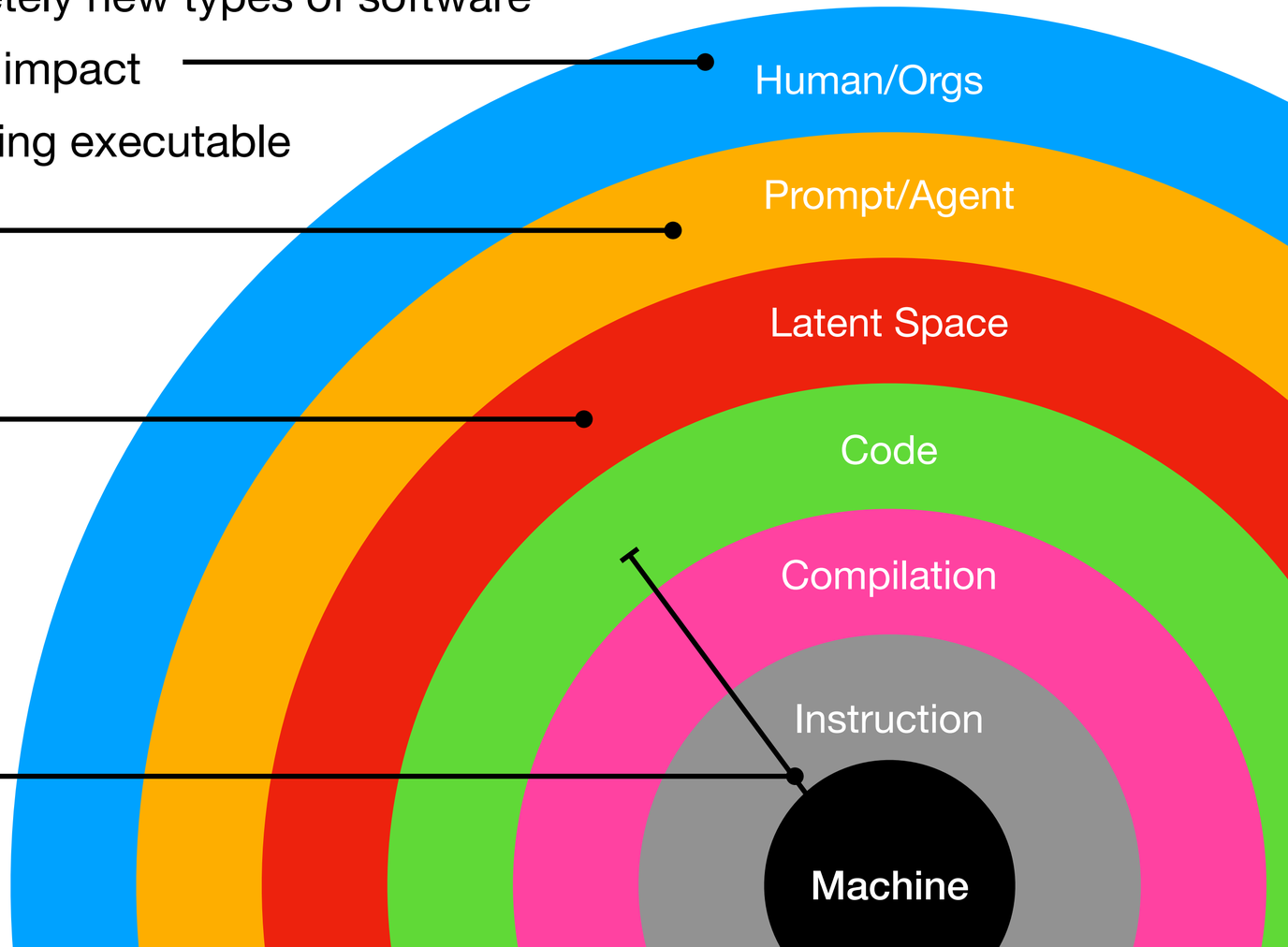
- Neuro-symbolic
- High-level semantics
- Mix of logic over LLM and tools

LLM Layer

- Latent/Continuous
- Mechanistic Interpretability
- Logic on Data-flow

Code to Machine Stack

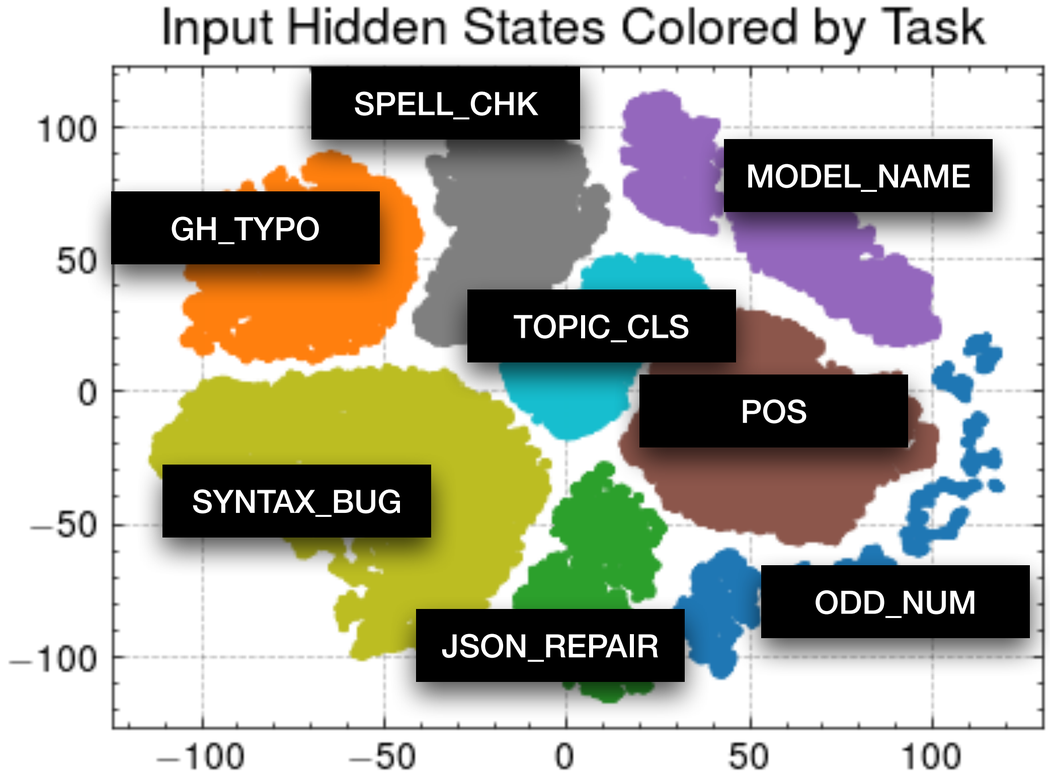
- Symbolic/Discrete
- Interpretable
- Logic on Control-flow

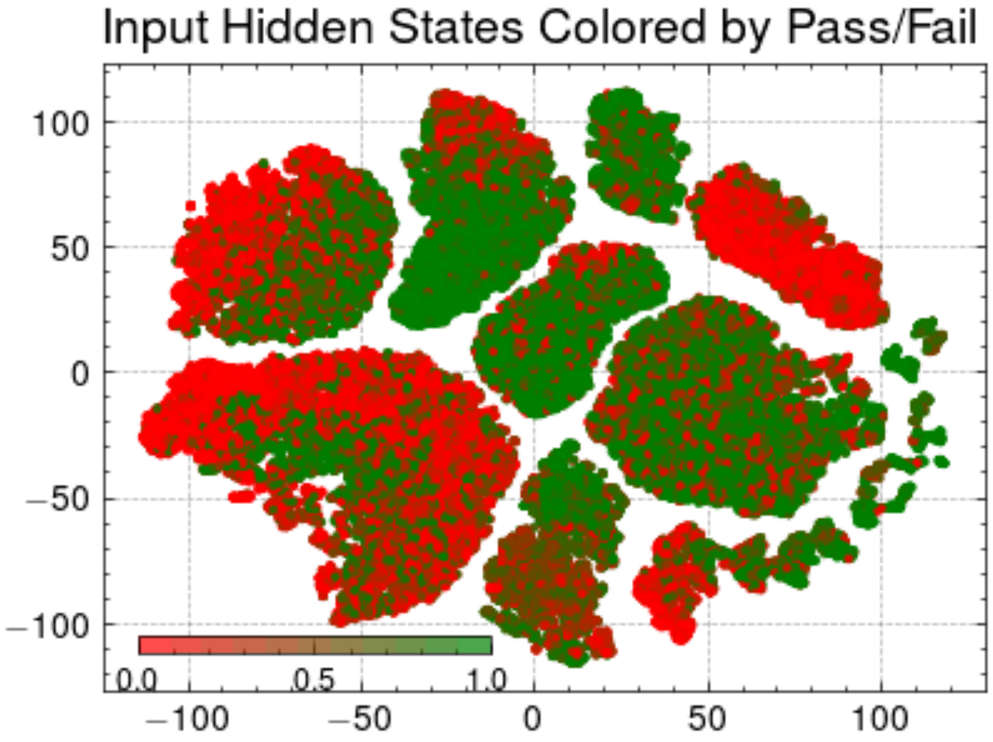


(One quick example)

Task-specific Modelling of Input Distribution

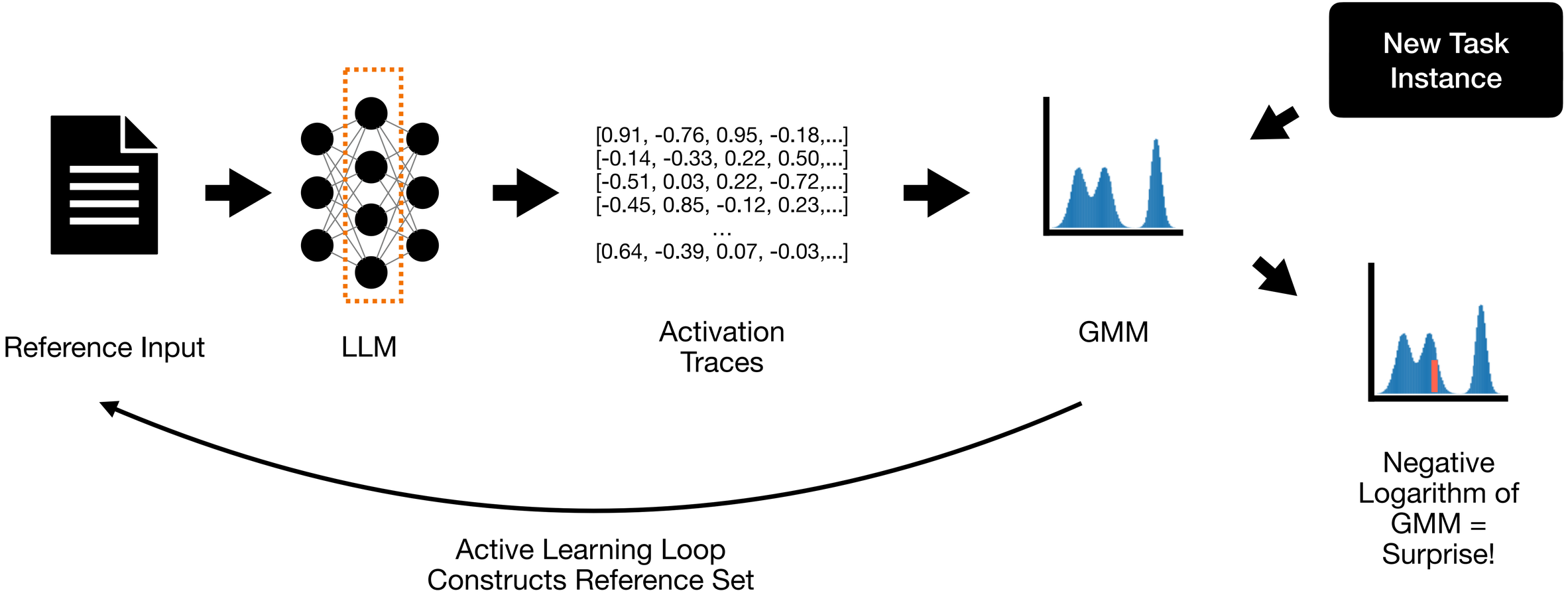
Clotho (FSE 2026 https://arxiv.org/abs/2509.17314)





Surprise Adequacy Strikes Again (SA^2)

(this time, w/ task-specific reference sets)



Clotho can predict failures reasonably well.

Average ROC-AUC $\simeq$ 0.7

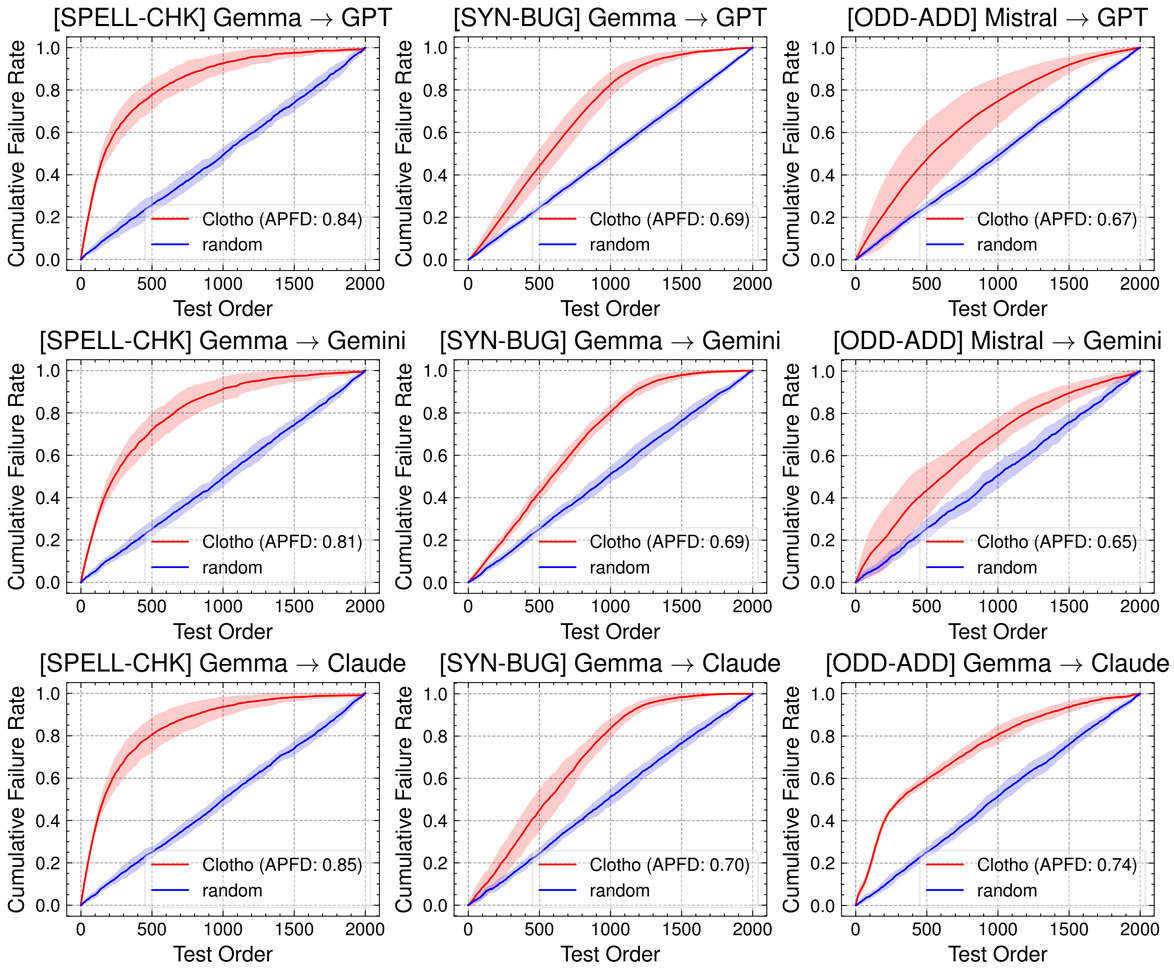
	ROC-AUC					
	Gemma 2 9B		Llama 3.1 8B		Mistral 7B	
Task ID	Clotho	MDSA	Clotho	MDSA	Clotho	MDSA
ODD-ADD	**0.714** ± 0.04	0.609 ± 0.01	**0.790** ± 0.02	0.609 ± 0.02	**0.883** ± 0.01	0.823 ± 0.03
GH-TYPO	0.737 ± 0.05	**0.756** ± 0.02	0.731 ± 0.02	**0.754** ± 0.01	**0.587** ± 0.02	0.575 ± 0.01
JSON-FIX	**0.764** ± 0.02	0.642 ± 0.01	**0.690** ± 0.02	0.583 ± 0.01	**0.793** ± 0.01	0.702 ± 0.02
MODEL-EX	**0.696** ± 0.07	0.595 ± 0.02	**0.758** ± 0.05	0.684 ± 0.02	**0.788** ± 0.02	0.702 ± 0.02
POS-TAG	**0.648** ± 0.04	0.568 ± 0.01	**0.601** ± 0.02	0.516 ± 0.01	**0.571** ± 0.02	0.508 ± 0.01
SPELL-CHK	0.869 ± 0.04	**0.871** ± 0.01	0.748 ± 0.03	**0.776** ± 0.01	0.649 ± 0.02	**0.663** ± 0.01
SYN-BUG	**0.905** ± 0.02	0.836 ± 0.01	**0.714** ± 0.02	0.593 ± 0.01	**0.717** ± 0.01	0.679 ± 0.01
TOPIC-CLS	**0.640** ± 0.04	0.588 ± 0.01	**0.618** ± 0.04	0.557 ± 0.02	**0.580** ± 0.03	0.539 ± 0.01
Average	**0.746** ± 0.10	0.683 ± 0.11	**0.706** ± 0.07	0.634 ± 0.09	**0.696** ± 0.11	0.649 ± 0.10

Without actual generation!

Scores transfer across models!

Clotho (FSE 2026 https://arxiv.org/abs/2509.17314))

- SA computed by Clotho shows moderate correlation with pass rates in SLMs.
- However, if you prioritize inputs according to SA computed by Clotho, and run inference on proprietary LLMs, the ordering still finds failing inputs earlier.



Summaries (for this lecture)

Is that SE or ML?

- ML: theoretical/empirical findings of relationship between out-of-distribution-ness and correctness
- SE: scaling it up, controlling human labeling cost, understanding failure modes…
- Arguably, software engineering is about quality of software, and is NOT defined as the sum of all its methodologies. Rather, the methodologies evolve as new types of software emerge.

Conclusion (for CS350)

- Unless we produce a real AGI (whatever that means) tomorrow, we will keep using software systems, and the knowledge of best practices and effective techniques related to SDLC will remain relevant (regardless of who writes the actual code).
- To borrow from Naur, software engineering is about how to build effective & accurate theory about our systems in an efficient way.
- There will be more types of software to be written.
- Think very hard before you write (or prompt) anything :)