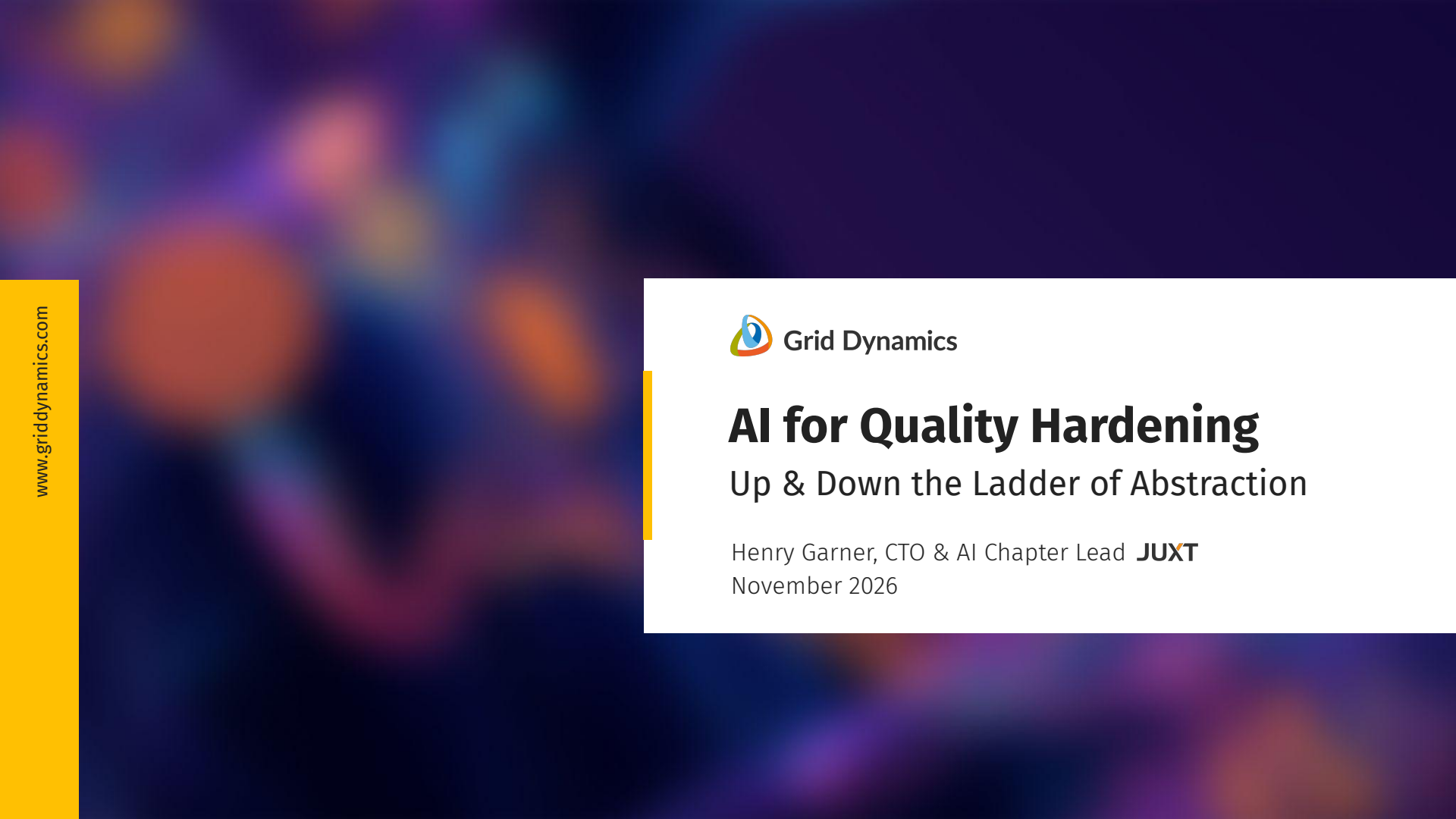
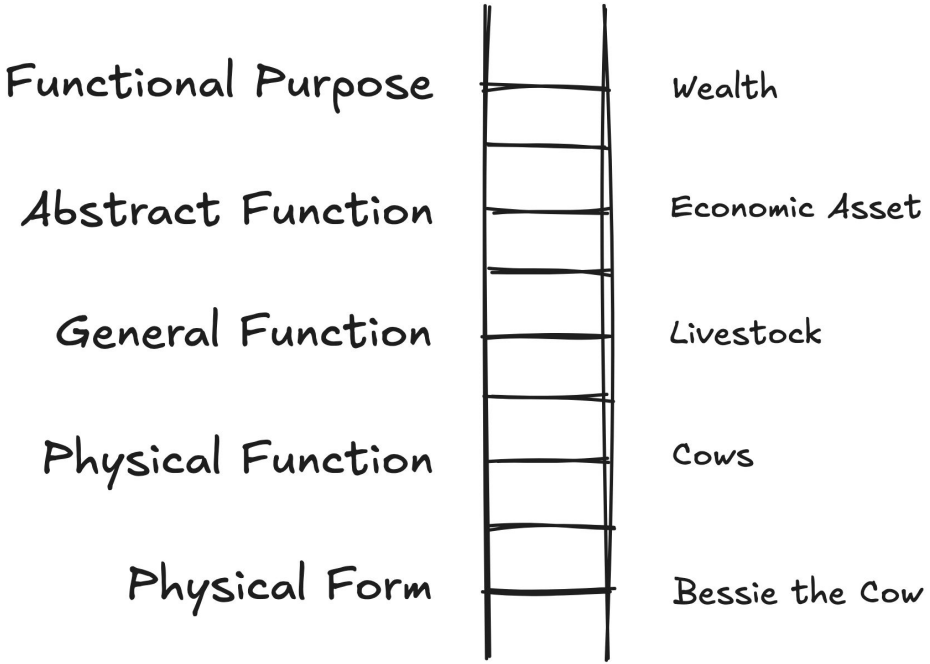

www.griddynamics.com
Grid Dynamics
AI for Quality Hardening
Up & Down the Ladder of Abstraction
Henry Garner, CTO & AI Chapter Lead JUXT
November 2026

Hayakawa's Ladder of Abstraction



"You're supposed to build a system that prompts itself"

Daisy Hollman, Anthropic

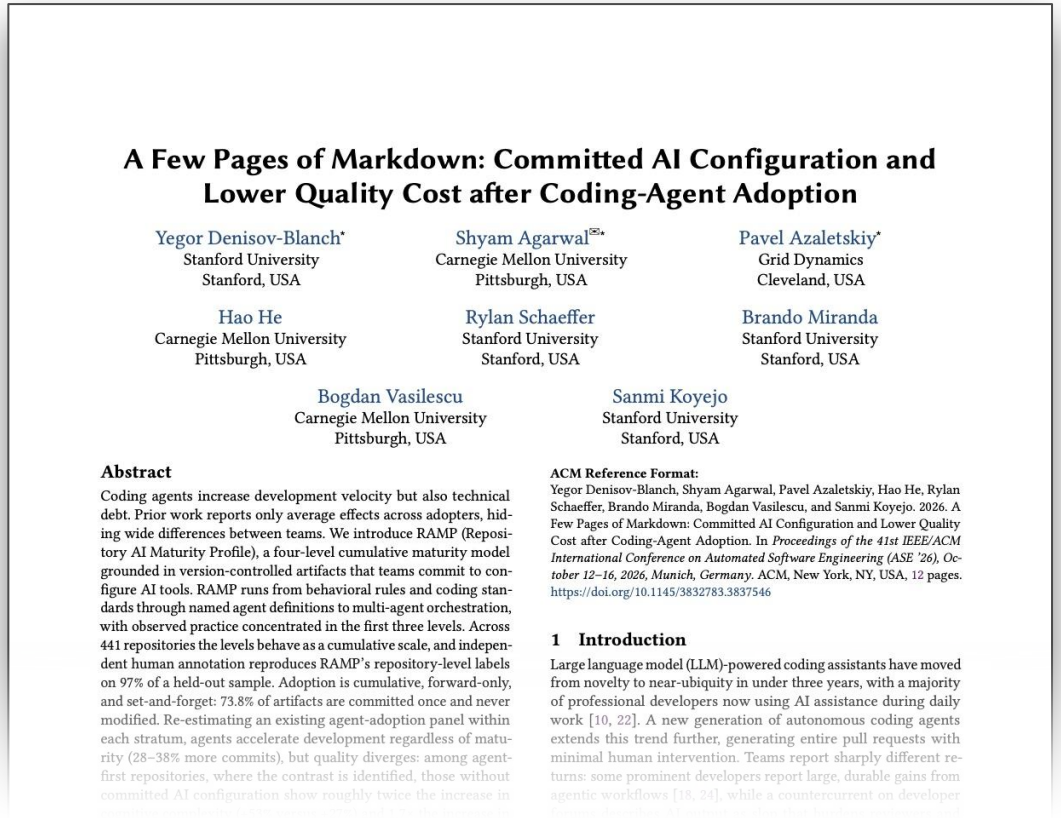
"We're all kind of mini-CEOs at this point"

Guillermo Rauch, Vercel

"Go make $1 million on a retail web platform in a few months"

Mustafa Suleyman, CEO Microsoft AI

Just Published: Our RAMP Research



A Few Pages of Markdown: Committed AI Configuration and Lower Quality Cost after Coding-Agent Adoption

Yegor Denisov-Blanch*
Stanford University
Stanford, USA

Shyam Agarwal✉*
Carnegie Mellon University
Pittsburgh, USA

Pavel Azaletskiy*
Grid Dynamics
Cleveland, USA

Hao He
Carnegie Mellon University
Pittsburgh, USA

Rylan Schaeffer
Stanford University
Stanford, USA

Brando Miranda
Stanford University
Stanford, USA

Bogdan Vasilescu
Carnegie Mellon University
Pittsburgh, USA

Sanmi Koyejo
Stanford University
Stanford, USA

Abstract

Coding agents increase development velocity but also technical debt. Prior work reports only average effects across adopters, hiding wide differences between teams. We introduce RAMP (Repository AI Maturity Profile), a four-level cumulative maturity model grounded in version-controlled artifacts that teams commit to configure AI tools. RAMP runs from behavioral rules and coding standards through named agent definitions to multi-agent orchestration, with observed practice concentrated in the first three levels. Across 441 repositories the levels behave as a cumulative scale, and independent human annotation reproduces RAMP's repository-level labels on 97% of a held-out sample. Adoption is cumulative, forward-only, and set-and-forget: 73.8% of artifacts are committed once and never modified. Re-estimating an existing agent-adoption panel within each stratum, agents accelerate development regardless of maturity (28–38% more commits), but quality diverges: among agent-first repositories, where the contrast is identified, those without committed AI configuration show roughly twice the increase in

ACM Reference Format:
Yegor Denisov-Blanch, Shyam Agarwal, Pavel Azaletskiy, Hao He, Rylan Schaeffer, Brando Miranda, Bogdan Vasilescu, and Sanmi Koyejo. 2026. A Few Pages of Markdown: Committed AI Configuration and Lower Quality Cost after Coding-Agent Adoption. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE '26), October 12–16, 2026, Munich, Germany.* ACM, New York, NY, USA, 12 pages.
https://doi.org/10.1145/3832783.3837546

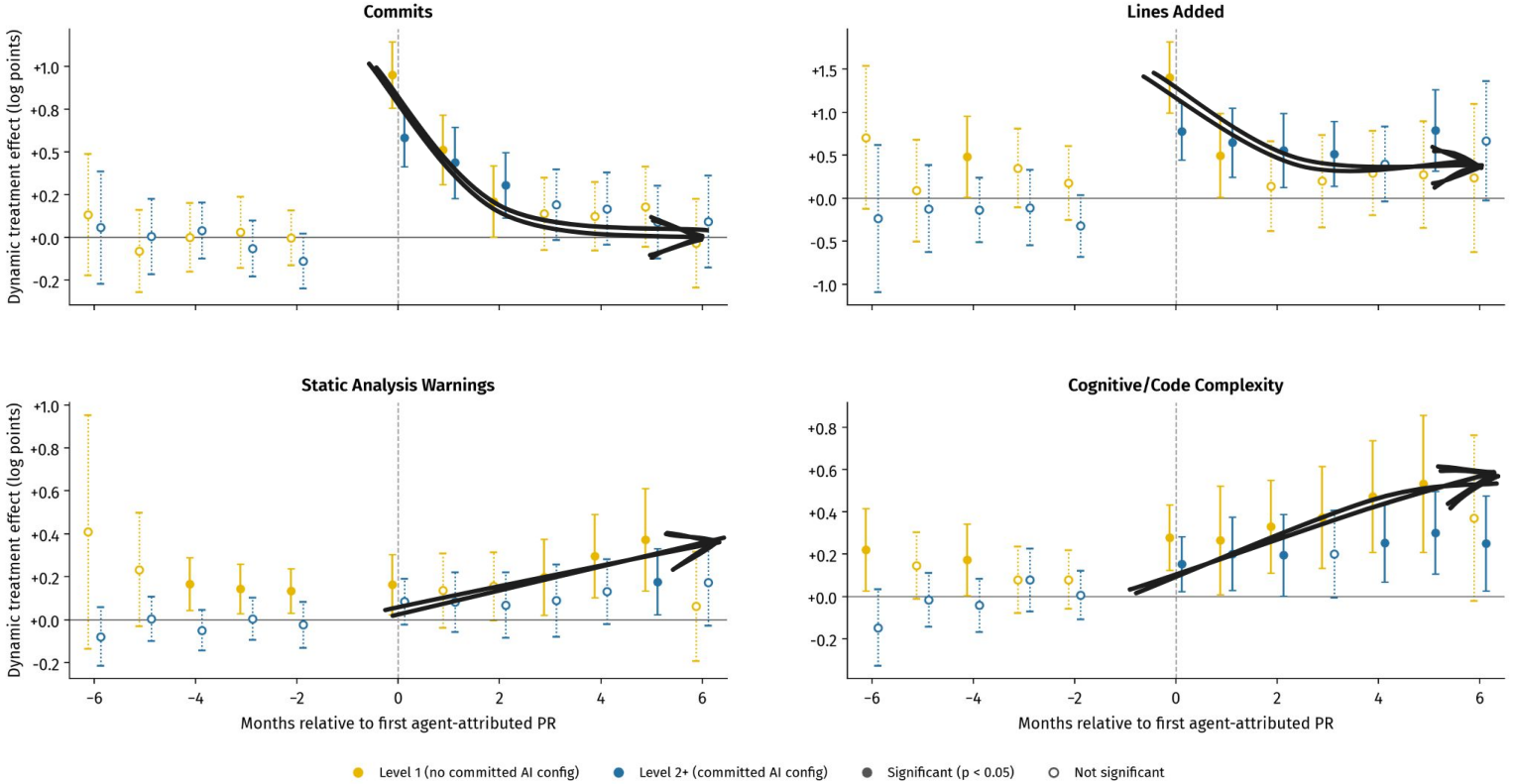
1 Introduction

Large language model (LLM)-powered coding assistants have moved from novelty to near-ubiquity in under three years, with a majority of professional developers now using AI assistance during daily work [10, 22]. A new generation of autonomous coding agents extends this trend further, generating entire pull requests with minimal human intervention. Teams report sharply different returns: some prominent developers report large, durable gains from agentic workflows [18, 24], while a countercurrent on developer

- 509 repositories analysed
- We tracked AI adopters and lookalike non-adopters month by month, and measured the adopters' change over and above the non-adopters
- Stratified based on config:
  - L1 236
  - L2 210
  - L3 23
  - L4 40

https://arxiv.org/abs/2608.25241

AI Adoption Effects



Isolated Fixes != Maintenance

SWE-CI: Evaluating Agent Capabilities in Maintaining Codebases via Continuous Integration

Jialong Chen[1,3,*], **Xander Xu**[2,*], **Hu Wei**[2], **Chuan Chen**[1,†], **Bing Zhao**[2,†]

[1]Sun Yat-sen University, [2]Alibaba Group, [3]Skylenage

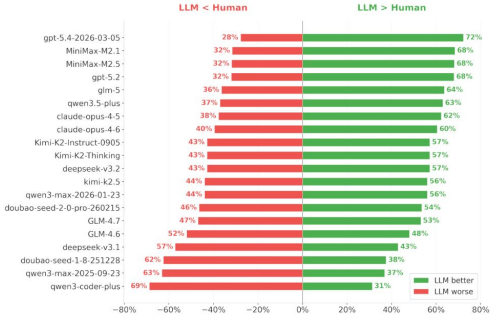
Hugging Face: https://huggingface.co/datasets/skylenage/SWE-CI

Github: https://github.com/SKYLENAGE-AI/SWE-CI

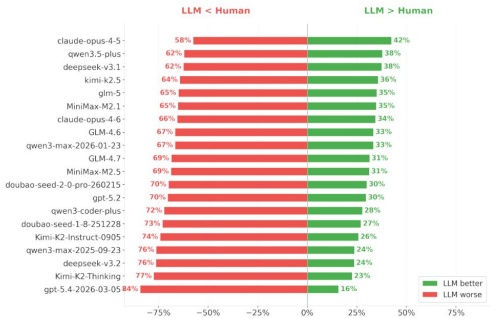
Abstract

Large language model (LLM)-powered agents have demonstrated strong capabilities in automating software engineering tasks such as static bug fixing. However, in the real world, the development of mature software is typically predicated on complex requirement changes and long-term feature iterations — a process that static, one-shot repair paradigms fail to capture. To bridge this gap, we propose **SWE-CI**, the first repository-level benchmark built upon the Continuous Integration loop, aiming to shift the evaluation paradigm for code generation from static, short-term *functional correctness* toward dynamic, long-term *maintainability*. The key insight is simple: **Maintainability can be revealed by tracking how functional correctness changes over time.** The benchmark comprises 100 tasks, each deriving from a real-world code repository with a development history spanning an average of 233 days and 71 consecutive commits. SWE-CI requires agents to systematically resolve these tasks through dozens of rounds of analysis and coding iterations. SWE-CI provides valuable insights into how well agents can sustain code quality throughout long-term evolution.

Pylint score



MI score



IBM Bank of Z

COBOL

https://github.com/IBM/Bank-of-Z

Sample banking application written in COBOL.

30k lines of COBOL
700 lines of Assembler
0 lines of tests

On every deposit, withdrawal or transfer, the amount is parsed into a floating-point double, then moved into a two-decimal field with no rounding so it truncates toward zero.

On about six percent of ordinary inputs the stored amount ends a penny low.

Is this a defect or intentional behaviour?

```
component Posting
  entity Post
  state entered(Post) : Money
  state stored(Post)  : Money

  invariant rounds_in_customers_favour means every p ::
    stored(p) >= entered(p) and stored(p) - entered(p) < 0.01
end
```

Apache Fineract

https://fineract.apache.org/



Open source core banking application written in Java.

850k lines of Java
540k lines of application code
310k lines of tests

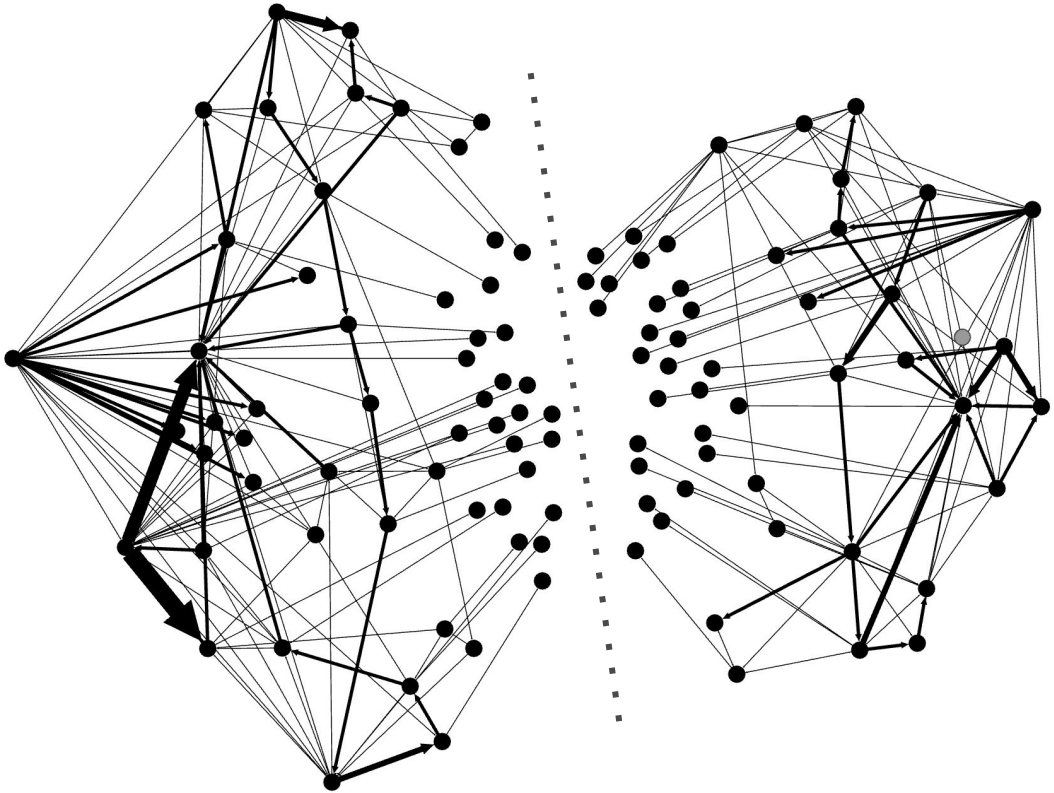
When a loan product's payment allocation is configured, unrecognised rule names are parsed to null. The validator only checks that there are twelve distinct entries, so a typo e.g. DUE_PRINCIPLE for DUE_PRINCIPAL is accepted as eleven real types plus a null.

A validated configuration causes payments to fail

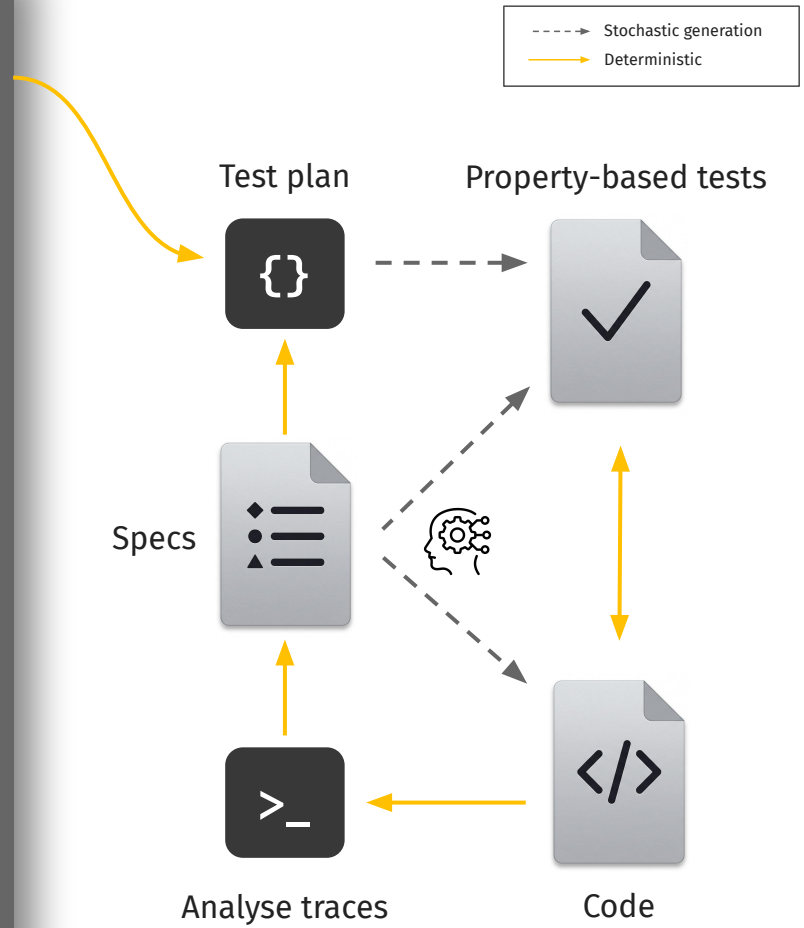
```
component AllocationConfig
  entity AllocationType
  entity Entry
  state entry_type(Entry) : AllocationType

  invariant completeness means every a : AllocationType ::
    some e : Entry :: entry_type(e) = a
  invariant distinct_types means every e1 : Entry :: every e2 : Entry ::
    (entry_type(e1) = entry_type(e2)) implies (e1 = e2)
end
```

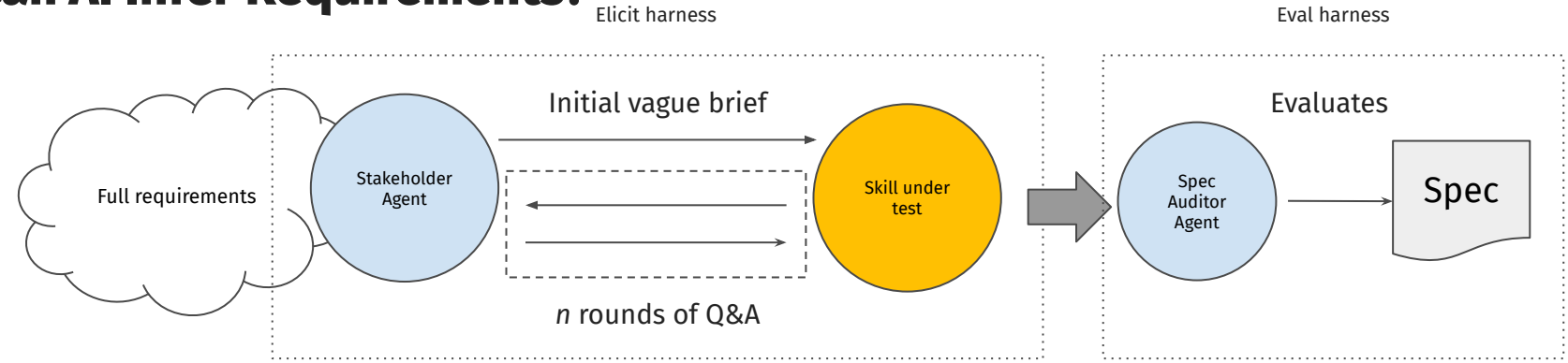
Code Property Graphs



```
{
  "obligations": [
    {
      "category": "vacuity",
      "construct": "a trace/state with at least one `AllocationType` (non-empty population)",
      "expect": "holds",
      "from": "every a :: ...",
      "obligation": "`completeness` is checked against a non-empty population, not vacuously true",
      "oracle": "the quantified body is evaluated at least once (population >= 1)",
    },
    {
      "category": "safety",
      "expect": "holds",
      "obligation": "invariant `completeness` holds: some e : Entry :: entry_type(e) = a",
      "oracle": "some e : Entry :: entry_type(e) = a",
    },
    {
      "category": "vacuity",
      "construct": "a trace/state with at least one `Entry` (non-empty population)",
      "expect": "holds",
      "from": "every e1, e2 :: ...",
      "obligation": "`distinct_types` is checked against a non-empty population, not vacuously true",
      "oracle": "the quantified body is evaluated at least once (population >= 1)",
    },
    {
      "category": "safety",
      "construct": "a state with entry_type(e1) = entry_type(e2) true",
      "expect": "holds",
      "from": "distinct_types: entry_type(e1) = entry_type(e2) implies e1 = e2  [implication]",
      "obligation": "`distinct_types` holds when its antecedent is true",
      "oracle": "e1 = e2",
    },
    {
      "category": "boundary",
      "construct": "a state with entry_type(e1) = entry_type(e2) false",
      "expect": "holds",
      "from": "distinct_types: entry_type(e1) = entry_type(e2) implies e1 = e2  [implication]",
      "obligation": "`distinct_types` is vacuously satisfied when its antecedent is false",
      "oracle": "e1 = e2 is not required",
    }
  ]
}
```

Can AI Infer Requirements?



3 tasks, based on 3 Apache Fineract subsystems: loan allocation, loan schedule, savings interest.
Unseen requirements were based on real Fineract config options, not invented gotchas.

Same stakeholder agent (Sonnet) for each system, same initial brief.
The same ‘blind’ auditor (Opus) grades each spec on # hidden requirements captured correctly.

- **14** hidden requirements for each task
- **3 times** we run the evaluation process for each skill
- Final score is the mean of the 3 runs as a %

Can AI Infer Requirements?

	Loan Allocation			Loan Scheduling			Savings Interest				
Skill	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**	**Total %**	**Qs (avg)**
Tessl	11	8	8	14	8	9	7	5	8	**61.9%**	6.9
Superpowers	8	13	8	9	7	10	5	10	9	**62.7%**	9.6
Kiro	9	10	11	9	8	9	8	8	8	**63.5%**	9.8
Vanilla prompt	10	13	9	12	9	10	6	7	6	**65.1%**	8.9
Spec-kit	11	10	12	8	9	10	8	9	6	**65.9%**	13.4
BMAD	12	11	11	11	11	8	6	9	7	**68.3%**	14.0
AIUP	11	11	10	11	11	10	8	9	8	**70.6%**	13.7
Allium	14	11	13	13	14	14	12	11	13	**91.3%**	22.0

Table shows how many off-default requirements were elicited by the skill across 3 separate candidate tasks, 3 runs each task. All counts out of 14.

Requirements Capture Trade-offs

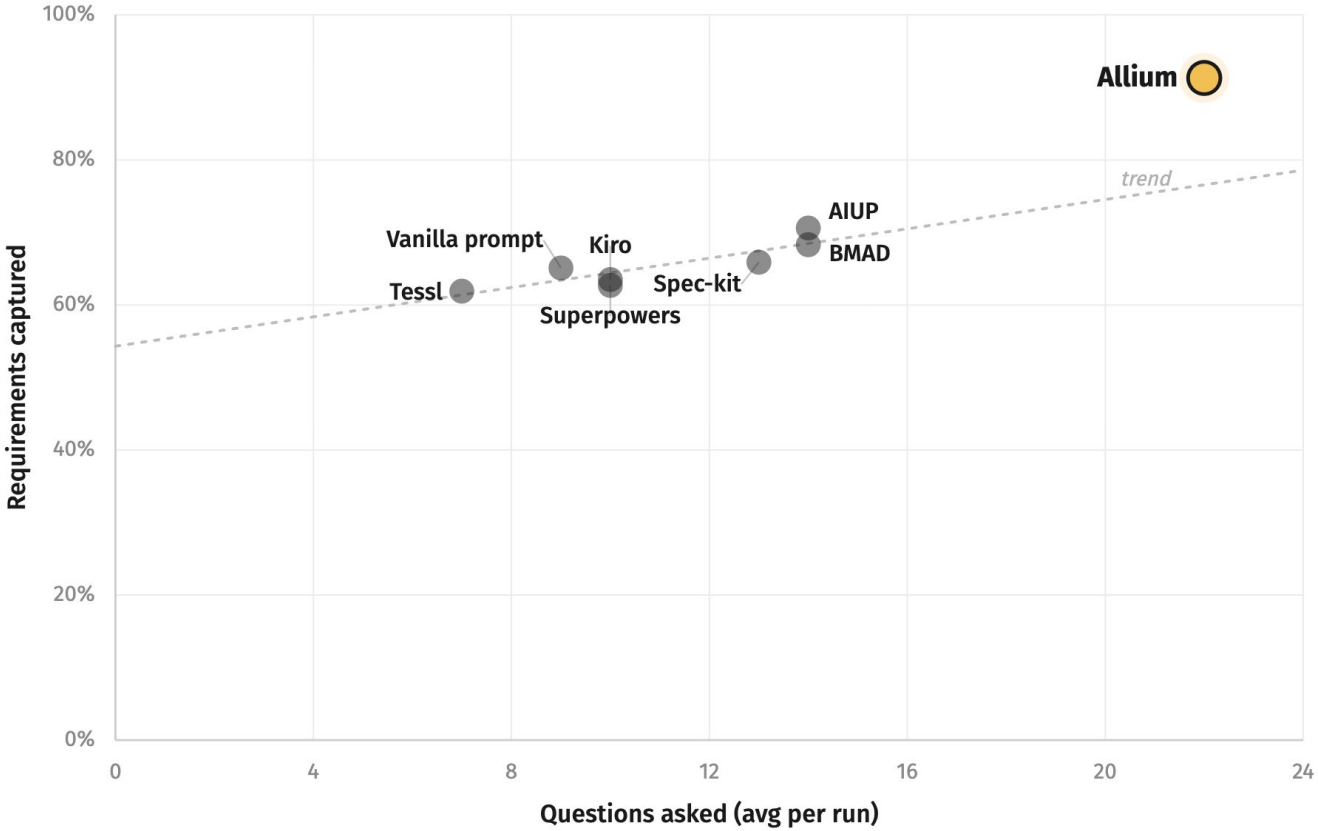
Others

- LIMIT: Maximum 3 [NEEDS CLARIFICATION] markers total
- you MAY ask up to TWO more... Do not exceed five total questions
- Maximum of 5 total questions across the whole session
- Make informed guesses

Allium

- Ask for the result shape directly, field by field. 'What must this return or report, exactly, every field?' Get the complete list, not a representative one.
- Thresholds and limits. Minimums, maximums, caps, and tolerances below which something is waived, ignored, or treated as nothing.
- Do not accept 'yes' without specifics. 'You said yes, candidates can reschedule. How many times? Is there a limit? What happens after that?'

Questions Asked Correlates with Requirements Captured



CVE Fixes & Quality Hardening at a Tier 1 bank



The ask: patch CVEs across **100+** Python & C# applications, some **1m+ LOC**



The stretch: Improve team's DORA metrics via high quality tests

- Shorter change lead time
- Lower change failure rate



The solution:

Distill behavioural specifications from code:

- Capture **observed** application behaviour
- Combine with **inferred** developer intent

Triage code & tests to **separate bugs from unintended features**

Automate high-quality test and code generation:

- Highlight **incorrect code** and **incorrect tests**
- **Isolate the fix** and **demonstrate** with existing QA tooling
- Update code **safely** and **securely** via automated PR

Real examples flagged **automatically** by spec distillation:

Likely bugs

- Calculation of mean uses incorrect denominator
- Semantic ID is incorrectly parsed in critical area
- Import will always report "0 rows imported"

Logic ambiguities

- Zero / NaN round-trip destroys data semantics
- Validation silently skips certain errors
- Core entities have no defined lifecycle

Brittle code

- Deduplication depends on unspecified file order
- Date parsing mixes day-first and month-first
- Missing data columns default to zero

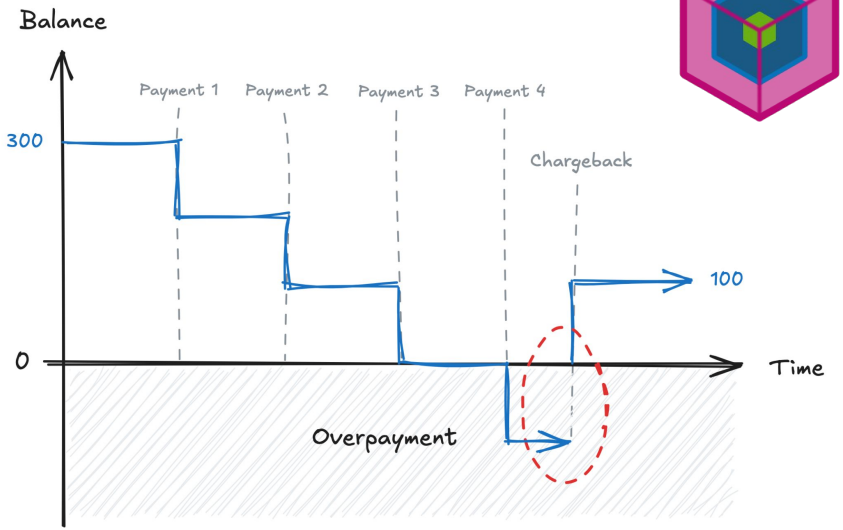
Domain questions

- Is zero ever a legitimate aggregate value?
- Should investigated issues be suppressible?
- Should recurring issues link to predecessors?

Fineract Replay Overpayment Bug

After any reversal or backdated change, it replays a loan's whole transaction history to recompute balances. On the live path, the running credit balance (the overpaymentHolder) is correctly seeded from the loan's real overpaid amount. On the replay path, overpayments are ignored.

A reversed overpayment results in a fresh debt




```
component Reprocessing
  entity Run
  state inputs(Run)       : Number
  state final_owed(Run)   : Number
  state final_credit(Run) : Number

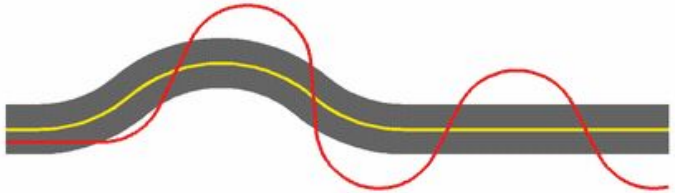
  invariant replay_reaches_the_same_state means every a : Run :: every b : Run ::
    inputs(a) = inputs(b) implies (final_owed(a) = final_owed(b) and final_credit(a) = final_credit(b))
end
```

An Alternative Ladder of Abstraction

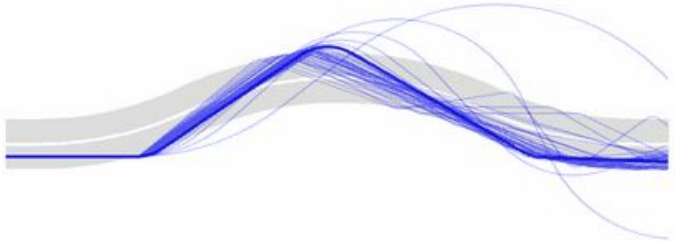
time: **0**
turning rate: **2°**
bend angle: **40°**



time: *
turning rate: **1.5°**
bend angle: **40°**

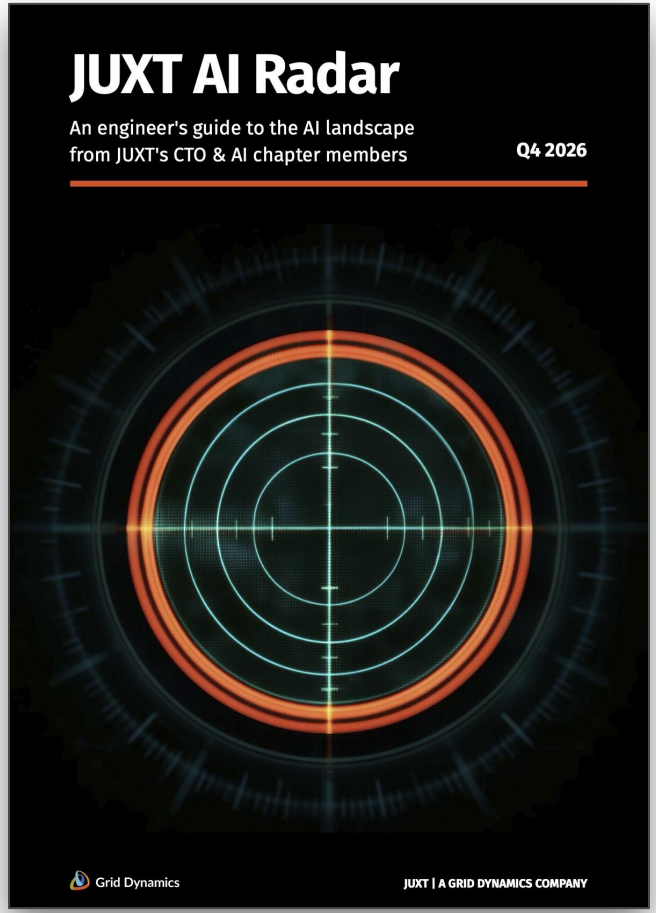


time: *
turning rate: *
bend angle: **25°**



https://worrydream.com/LadderOfAbstraction/

Grab a copy!


JUXT AI Radar
An engineer's guide to the AI landscape
from JUXT's CTO & AI chapter members
Q4 2026
Grid Dynamics
JUXT | A GRID DYNAMICS COMPANY



Grid Dynamics

trusted engineering partner for digital transformation

Thank you

Get Allium

https://allium-lang.org

http://github.com/juxt/allium