www.griddynamics.com

Grid Dynamics

Spectroscopy for Software

Measuring Legacy Code with AI

Henry Garner, CTO & AI Chapter Lead JUXT

June 2026

METR

Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity

CONTRIBUTORS
Joel Becker, Nate Rush, Beth Barnes, and David Rein

DATE
July 10, 2025

SHARE
Copy Link arXiv Citation

We are Changing our Developer Productivity Experiment Design

CONTRIBUTORS
Joel Becker, Nate Rush, Tom Cunningham, David Rein, and Khalid Mahamud

DATE
February 24, 2026

SHARE
Copy Link Citation

Speed

Developers who used GitHub Copilot completed the task 55% faster than those who didn’t

GitHub

AI coding tools make developers slower but they think they’re faster

The Register

Quality

81% of those who use AI for code review saw quality improvement

Qodo

AI-authored code contains worse bugs than software crafted by humans

The Register

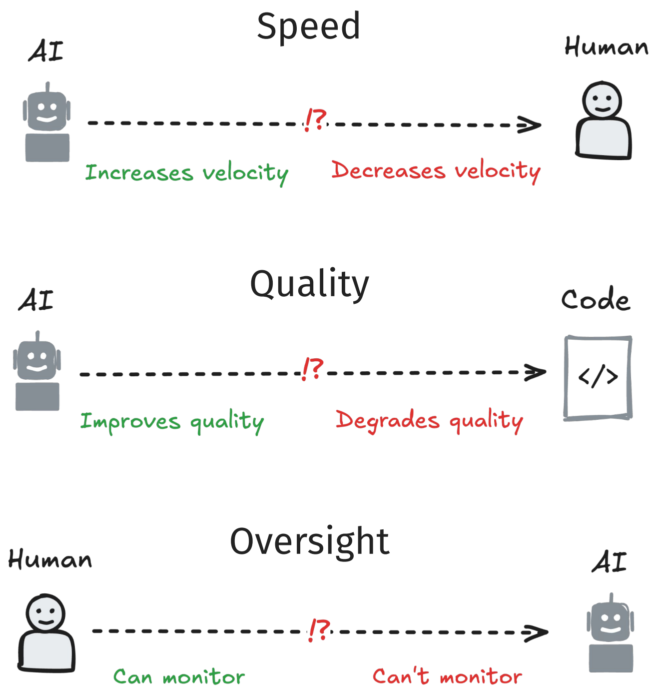
Oversight

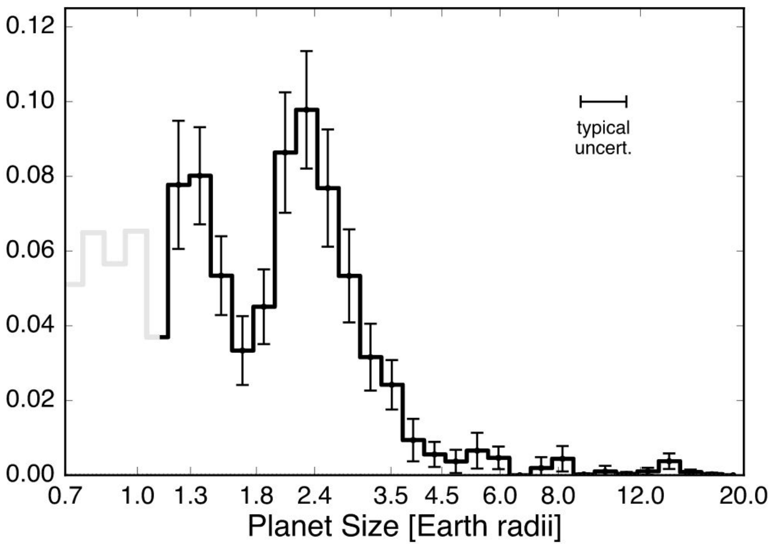
AI systems must be effectively overseen by natural persons

EU AI Act, Article 14

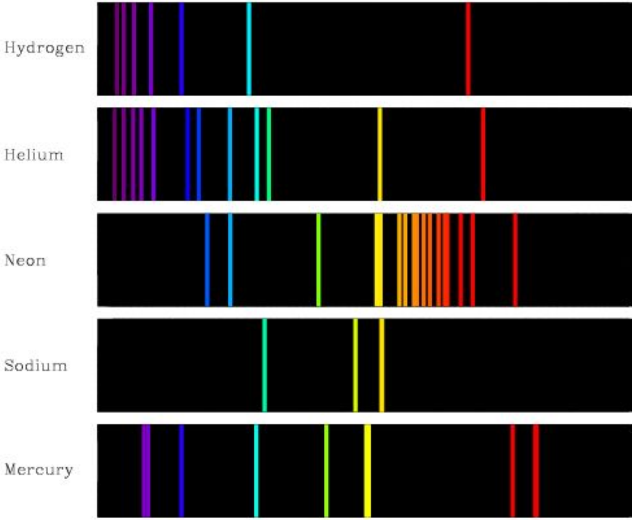
The human has been given an impossible task

Lisanne Bainbridge, Ironies of Automation


Speed
AI
Human
!?
Increases velocity
Decreases velocity
Quality
AI
Code
</>
!?
Improves quality
Degrades quality
Oversight
Human
AI
!?
Can monitor
Can't monitor



0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.7
1.0
1.3
1.8
2.4
3.5
4.5
6.0
8.0
12.0
20.0
Planet Size [Earth radii]
typical uncert.



Hydrogen
Helium
Neon
Sodium
Mercury

#1 Trusted Cybersecurity News Platform | Followed by 5.70+ million

The Hacker News

Home | Vulnerabilities | Cyber Attacks | Expert Insights | Awards

Claude Mythos AI Finds 10,000 High-Severity Flaws in Widely Used Software

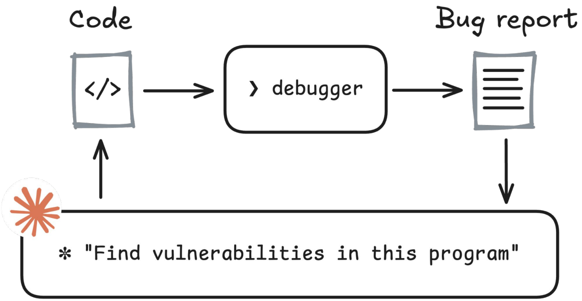
Ravie Lakshmanan May 23, 2026

MEMOS

Treasury Secretary and Federal Reserve Chair Warn Bank CEOs About Cybersecurity Risks Posed by Anthropic's New AI Model

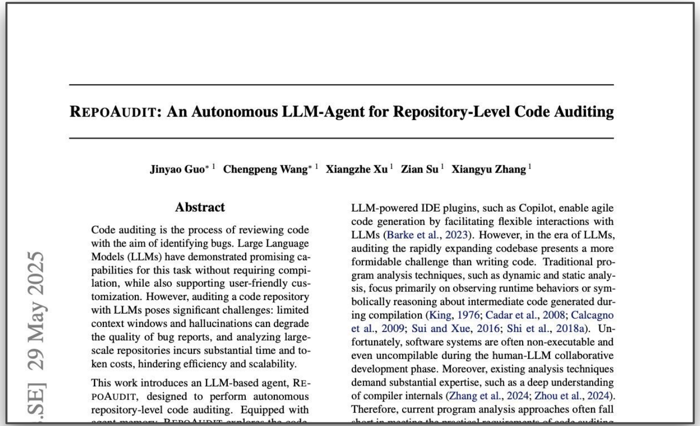
Bank of England, FCA and HM Treasury Warn Financial Firms Over Frontier AI-Linked Cyber Threats After PRA Warning

By Ewan Scott | May 17, 2026 | Economy & Business, Featured, News & Analysis



"A migration to Rust … would shrink one very important risk surface, but it would not neutralize … leaks that are "working as coded" and still unsafe"

Penligent AI, Claude Mythos Preview and the New Zero-Day Era, 2026



REPOAUDIT: An Autonomous LLM-Agent for Repository-Level Code Auditing

Jinyao Guo[*1] Chengpeng Wang[*1] Xiangzhe Xu[1] Zian Su[1] Xiangyu Zhang[1]

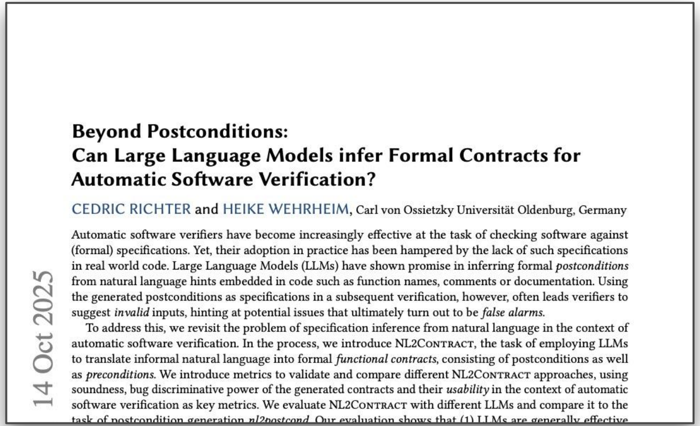
.SE] 29 May 2025

Abstract

Code auditing is the process of reviewing code with the aim of identifying bugs. Large Language Models (LLMs) have demonstrated promising capabilities for this task without requiring compilation, while also supporting user-friendly customization. However, auditing a code repository with LLMs poses significant challenges: limited context windows and hallucinations can degrade the quality of bug reports, and analyzing large-scale repositories incurs substantial time and token costs, hindering efficiency and scalability.

This work introduces an LLM-based agent, REPOAUDIT, designed to perform autonomous repository-level code auditing. Equipped with

LLM-powered IDE plugins, such as Copilot, enable agile code generation by facilitating flexible interactions with LLMs (Barke et al., 2023). However, in the era of LLMs, auditing the rapidly expanding codebase presents a more formidable challenge than writing code. Traditional program analysis techniques, such as dynamic and static analysis, focus primarily on observing runtime behaviors or symbolically reasoning about intermediate code generated during compilation (King, 1976; Cadar et al., 2008; Calcagno et al., 2009; Sui and Xue, 2016; Shi et al., 2018a). Unfortunately, software systems are often non-executable and even uncompilable during the human-LLM collaborative development phase. Moreover, existing analysis techniques demand substantial expertise, such as a deep understanding of compiler internals (Zhang et al., 2024; Zhou et al., 2024). Therefore, current program analysis approaches often fall



Beyond Postconditions:
Can Large Language Models infer Formal Contracts for Automatic Software Verification?

CEDRIC RICHTER and HEIKE WEHRHEIM, Carl von Ossietzky Universität Oldenburg, Germany

14 Oct 2025

Automatic software verifiers have become increasingly effective at the task of checking software against (formal) specifications. Yet, their adoption in practice has been hampered by the lack of such specifications in real world code. Large Language Models (LLMs) have shown promise in inferring formal *postconditions* from natural language hints embedded in code such as function names, comments or documentation. Using the generated postconditions as specifications in a subsequent verification, however, often leads verifiers to suggest *invalid* inputs, hinting at potential issues that ultimately turn out to be *false alarms*.

To address this, we revisit the problem of specification inference from natural language in the context of automatic software verification. In the process, we introduce NL2CONTRACT, the task of employing LLMs to translate informal natural language into formal *functional contracts*, consisting of postconditions as well as *preconditions*. We introduce metrics to validate and compare different NL2CONTRACT approaches, using soundness, bug discriminative power of the generated contracts and their *usability* in the context of automatic software verification as key metrics. We evaluate NL2CONTRACT with different LLMs and compare it to the

"Analyzing data-flow facts along feasible program paths within individual functions … detects 185 new bugs in high-profile projects, among which 174 have been confirmed or fixed."

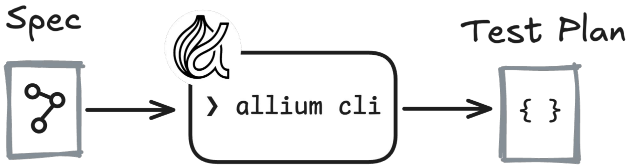
RepoAudit

"Our evaluation shows that LLMs can translate natural language descriptions into non-trivial contracts that accurately capture the developer's intent."

Beyond Postconditions

Entities • Rules • Invariants • Surfaces • Actors • Guidance

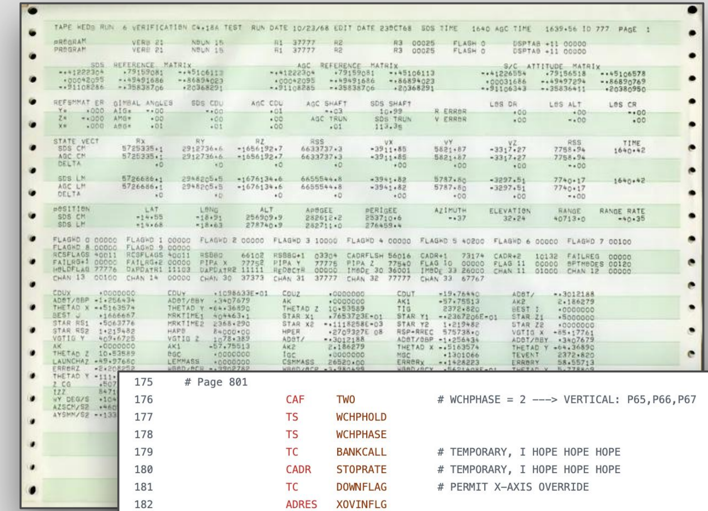
```
-- allium: 3

enum QuoteStatus { quoted | ordered | rejected | expired }
enum ProductCode { equity | fx | bond | derivative }

entity Client {
    id: Integer
    allowed_products: List<ProductCode>
    remaining_credit: Integer
    daily_turnover_limit: Integer

    -- Relationship: every Quote that names this Client
    quotes: Quote with client = this

    -- Projection: today's orders for this client
    todays_orders: quotes where status = ordered and same_day(placed_at, now)
}

entity Quote {
    id: Integer
    client: Client
    product: ProductCode
    notional: Integer
    status: QuoteStatus
    placed_at: Timestamp when status = ordered
}

given {
    quotes: List<Quote>
    clients: List<Client>
}

rule PlaceOrder {
    when: OrderRequested(quote)

    requires:
        quote.status = quoted
        quote.product in quote.client.allowed_products
        quote.notional <= quote.client.remaining_credit

    ensures:
        quote.status = ordered
        quote.placed_at = now
}

invariant ClientWithinDailyTurnoverLimit {
    -- Sum of notional across today's ordered quotes per client
    -- must not exceed that client's daily turnover limit.
    for c in Clients:
        sum(c.todays_orders, q => q.notional) <= c.daily_turnover_limit
}
```



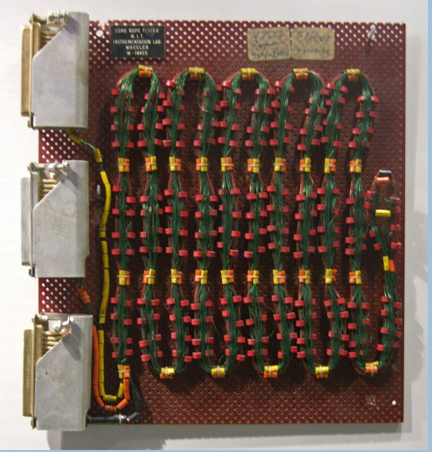

```json
{
  "category": "projection",
  "description": "Verify projection Client.todays_orders filters correctly",
  "id": "projection.Client.todays_orders",
  "source_construct": "Client.todays_orders",
  "source_span": {
    "end": 553,
    "start": 193
  }
},
```

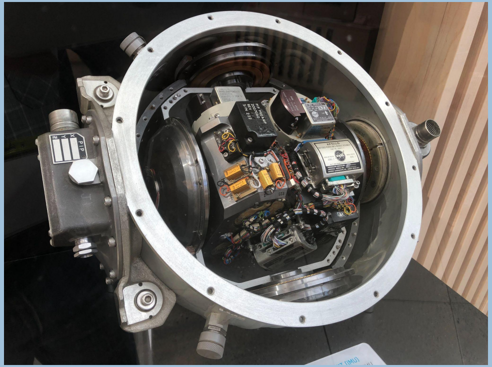
```json
{
  "category": "invariant",
  "description": "Verify invariant ClientWithinDailyTurnoverLimit holds after every state-changing rule that touches constrained entities",
  "id": "invariant.ClientWithinDailyTurnoverLimit",
  "source_construct": "ClientWithinDailyTurnoverLimit",
  "source_span": {
    "end": 1332,
    "start": 1071
  }
}
```

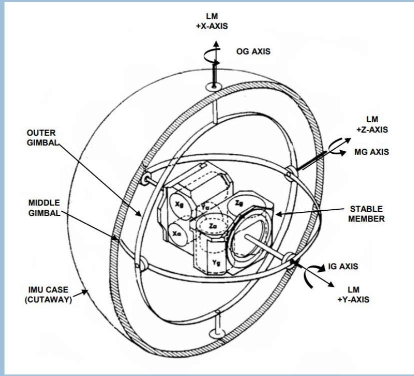
```json
{
  "category": "rule_failure",
  "dependencies": {
    "entities_read": [
      "Quote"
    ],
    "entities_written": [
      "Quote"
    ],
    "trigger_source": "external"
  },
  "description": "Verify rule PlaceOrder is rejected when requires clause fails",
  "id": "rule-failure.PlaceOrder.1",
  "source_construct": "PlaceOrder",
  "source_span": {
    "end": 992,
    "start": 842
  }
},
```

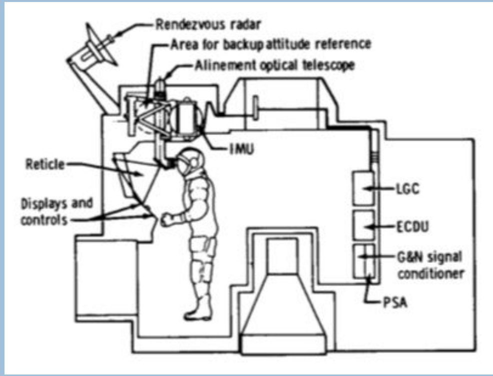


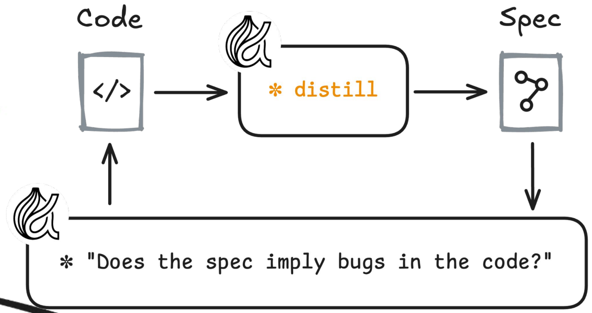
```
# Page 801
                CAF     TWO             # WCHPHASE = 2 ---> VERTICAL: P65,P66,P67
                TS      WCHPHOLD
                TS      WCHPHASE
                TC      BANKCALL        # TEMPORARY, I HOPE HOPE HOPE
                CADR    STOPRATE        # TEMPORARY, I HOPE HOPE HOPE
                TC      DOWNFLAG        # PERMIT X-AXIS OVERRIDE
                ADRES   XOVINFLG
                TC      DOWNFLAG
                ADRES   REDFLAG
                TCF     VERTGUID

STARTP67        TC      NEWMODEX        # NO HARM IN "STARTING" P67 OVER AND OVER
                DEC     67              # SO NO NEED FOR A FASTCHNG AND NO NEED
                CAF     ZERO            # TO SEE IF ALREADY IN P67.
                TS      RODCOUNT
                CAF     TEN
                TCF     VRTSTART

STABL?          CAF     BIT13           # IS UN-ATTITUDE-HOLD DISCRETE PRESENT?
                EXTEND
                RAND    CHAN31
                CCS     A
                TCF     GUILDRET        # YES ALL'S WELL

P66NOW?         CS      MODREG
                AD      DEC66
                EXTEND
                BZF     RESTART?
```






LM
+X-AXIS
OG AXIS
OUTER
GIMBAL
LM
+Z-AXIS
MG AXIS
MIDDLE
GIMBAL
STABLE
MEMBER
IG AXIS
IMU CASE
(CUTAWAY)
LM
+Y-AXIS


Rendezvous radar
Area for backup attitude reference
Alinement optical telescope
IMU
Reticle
LGC
Displays and
controls
ECDU
G&N signal
conditioner
PSA



BURN_BABY_BURN

PINBALL_GAME_BUTTONS_AND_LIGHTS

TRASHY_LITTLE_SUBROUTINES

```
-- NVSUB — internal program interface ----------------------

rule NvsubRequest {
    when: NvsubCall(verb_code, noun_code)
    requires: dsky.busy = false
    ensures:
        dsky.current_verb = verb_code
        dsky.current_noun = noun_code
        dsky.busy = true
        dsky.busy_owner = `calling_job`
        dsky.input_source = internal
        VerbDispatch(verb_code: verb_code, noun_code: noun_code)
}

rule NvsubBusyWait {
    when: NvsubCall(verb_code, noun_code)
    requires: dsky.busy = true
    ensures:
        JobSleep(job: `calling_job`)

    @guidance
        -- The caller enters JOBSLEEP and will be woken when the
        -- current DSKY user completes. There is no timeout on this
        -- wait. If the busy owner dies without releasing the DSKY,
        -- the waiting job sleeps indefinitely.
}

rule DskyRelease {
    -- The busy owner releases the DSKY, waking any waiting job.
    when: DskyRelease(owner)
    requires:
        dsky.busy = true
        dsky.busy_owner = owner
    ensures:
        dsky.busy = false
        dsky.busy_owner = null
}
```

!!

```
-- Gyro Torquing (IMPULSE) ------------------------------------

rule GyroTorque {
    -- Sends gyro torquing pulse commands. Reserves the gyros,
    -- enables power supply, and dispatches pulses per axis.
    when: GyroTorque(command: GyroTorqueCommand)

    requires:
        imu.mode != caged
        imu.gyros_busy = false

    ensures:
        imu.gyros_busy = true
        GyroTorqueStarted()
}

rule GyroTorqueBusy {
    -- Gyros already reserved by another torquing operation.
    -- Caller sleeps until LGYRO is cleared.
    when: GyroTorque(command: GyroTorqueCommand)

    requires: imu.gyros_busy = true

    ensures:
        JobSleep(job: calling_job())
}
```

MIT/DL PROGRAM CHANGE ROUTING SLIP

PCR/PCN #
ANOMALY # L-1D-02
ADR #

- [ ] COLOSSUS 2E
- [ ] COLOSSUS 3
- [ ] COLOSSUS 3A
- [ ] COLOSSUS___
- [ ] LUMINARY 1D
- [x] LUMINARY 1E
- [ ] LUMINARY 1F
- [ ] LUMINARY___

- [ ] MIT Approved PCN
- [ ] MIT Approved ADR
- [ ] NASA Approved PCR
- [ ] NASA Approved PCN
- [ ] NASA Approved Software Anomaly
- [x] MIT Approved Software Anomaly

A. Coding

(✓) Begin coding immediately — Bob Covelli

ACTION:

Program Supervisor's Approval: Margaret Hamilton

( ) Do not code until new GSOP material has been approved by the MIT Mission Design Review Board (MDRB) and distributed.

B. GSOP Preparation

( ) Prepare GSOP revisions for MDRB consideration

ACTION:

- [ ] Technical Committee Meeting not required.
- [ ] Technical Committee Meeting(s) held on
  Attendees:

C. KSC Testing and Checkout

( ) Review for possible impact on KSC testing and checkout

ACTION:

D. Other Programs Affected

( ) Review for corresponding changes in

ACTION:

Special Instructions

Project Manager R. H. Larson
Date 7-8-70

MIT/IL SOFTWARE ANOMALY REPORT

MSC REPORT NO.: L-1D-02
PROGRAM: LUMINARY
PROGRAM REVISION: ALL

1.1 ORIGINATOR:	1.2 ORGANIZATION:	1.3 DATE:	1.4 ORIGINATOR CONTROL NO.
N. Barnert/P. Rye	MIT/CSDL	5/19/70	

1.5 DESCRIPTION OF ANOMALY:
It is possible that a core set overflow (31202 Abort) will result from a "bad" return from pulse torquing due to IMU cage or coarse align.
CONTINUED ON PAGE

1.6 DESCRIPTION OF RUN:
Satanche Level 3 test
CONTINUED ON PAGE

- MIT ANALYSIS -

2.1 CAUSE:
"LGYRO" not zeroed if pulse torquing ends due to cage or coarse align; subsequent jobs attempting torquing will stack up via JOBSLEEP.
CONTINUED ON PAGE

2.2 RECOGNITION:
31202 Abort
CONTINUED ON PAGE

2.3 MISSION EFFECT:
Software restart caused by 31202 abort cleans up the problem, but further calls (by gyro compensation during average G) will refill the core sets, causing repeated overflows.
CONTINUED ON PAGE

2.4 AVOIDANCE PROCEDURE:
Do not coarse align during Average G; or else set GCOMPSW negative.
Zero LGYRO whenever cage is pushed.
CONTINUED ON PAGE

2.5 RECOVERY PROCEDURE:
Set GCOMPSW negative (terminates compensation).
CONTINUED ON PAGE

2.6 PROGRAM CORRECTION:
Clear LGYRO and wake possible sleeping jobs before exiting pulse torquing routine.
CONTINUED ON PAGE

2.7 RECOMMENDED DISPOSITION (Fix, Work-around, etc):
Fix for Luminary 1E
CONTINUED ON PAGE

2.8 RECOMMENDED RE-TESTING:
Level 2 test.
CONTINUED ON PAGE

2.9 MIT/IL SIGNATURE: R. H. Larson
2.10 DATE: 7-8-70

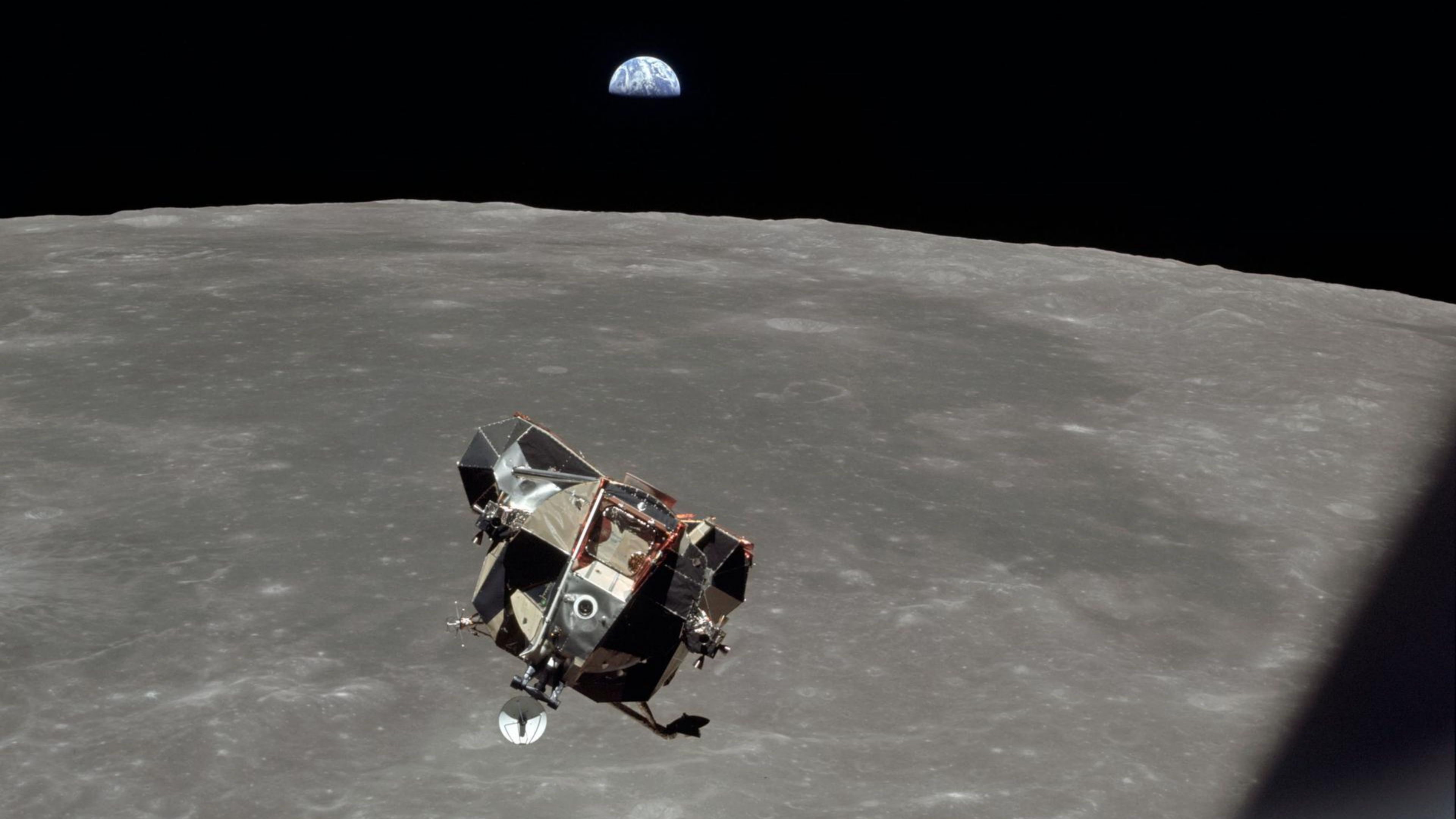
3.1 NASA DIRECTION:
CONTINUED ON PAGE

4.1 CLOSING ACTION TAKEN:
6
CONTINUED ON PAGE

3.2 NASA/MSC SIGNATURE:	3.3 ORGANIZATION:	3.4 DATE:	4.2 SIGNATURE:	4.3 ORGANIZATION:	4.4 DATE:

MSC Form 1409 (Rev May 68)

PAGE 1 OF



Peter Naur, Programming as Theory Building • 227

PETER NAUR, PROGRAMMING AS THEORY BUILDING

Peter Naur, widely known as one of the authors of the programming language syntax notation "Backus-Naur Form" (BNF), wrote "Programming as Theory Building" in 1985. It was reprinted in his collection of works, *Computing: A Human Activity* (Naur 1992).

This article is, to my mind, the most accurate account of what goes on in designing and coding a program. I refer to it regularly when discussing how much documentation to create, how to pass along tacit knowledge, and the value of the XP's metaphor-setting exercise. It also provides a way to examine a methodology's economic structure.

In the article, which follows, note that the quality of the designing programmer's work is related to the quality of the match between his theory of the problem and his theory of the solution. Note that the quality of a later programmer's work is related to the match between his theories and the previous programmer's theories.

Using Naur's ideas, the designer's job is not to pass along "the design" but to pass along "the theories" driving the design. The latter goal is more useful and more appropriate. It also highlights that knowledge of the theory is tacit in the owning, and so passing along the theory requires passing along both explicit and tacit knowledge.

Here is Peter Naur's way of saying it.

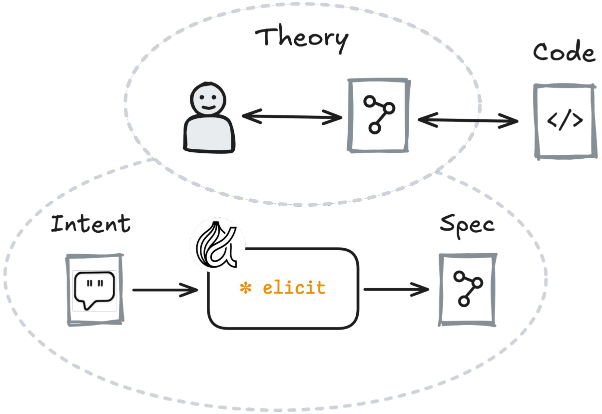
"PROGRAMMING AS THEORY BUILDING"

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.

Some of the background of the views presented here is to be found in certain observations of what actually happens to programs and the teams of programmers dealing with them, particularly in situations arising from unexpected and perhaps erroneous program executions or reactions, and on the occasion of modifications of programs. The difficulty of accommodating such observations in a production view of programming suggests that this view is misleading. The theory building view is presented as an alternative.

A more general background of the presentation is a conviction that it is important to have an appropriate understanding of what programming is. If our understanding is inappropriate we will misunderstand the difficulties that arise in the activity and our attempts to overcome them will give rise to conflicts and frustrations.



"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. "

"The death of a program happens when the programmer team possessing its theory is dissolved."

Peter Naur, Programming as Theory Building

❯ let's add a program for the crew to orient the spacecraft so that the moon is visible from the window

Write(Comanche055/P79.agc)

Wrote 111 lines to Comanche055/P79.agc

Added P79, a MOON VIEW ORIENTATION PROGRAM for the CSM, and wired it into Comanche055/MAIN.agc:33.

What it does:

- V37E79E starts the program. Crew enters a time-of-view via V06N34.

- Integrates the CSM state to that time (CSMPREC), gets the Moon position (LUNPOS), forms the unit line-of-sight in REFSMMAT, then rotates it into stable-member coordinates and stores it in POINTVSM.

- Sets SCAXIS to the W1 (left rendezvous window) boresight in body coordinates – 31.78 degrees up from +X toward +Z – and calls R60CSM, which uses VECPOINT to compute gimbal angles and flashes V06N18 for the crew to approve

the auto maneuver.

- Termination via V34 returns through GOTOPOOH; the TRACKFLG is restored on exit.

Caveats: P79 isn't in the original rope, so the header marks it as a modern addition. The W1 boresight constant uses the standard Block II window cant angle but I haven't cross-checked it against CSM_GEOMETRY.agc – worth

verifying before any assembly attempt, along with confirming REFTOSMM, GOFLASHR, and INTWAKE symbol names match the Comanche055 labels (the Luminary side uses slightly different spellings).

❯ **/allium** let's add a program for the crew to orient the spacecraft so that the moon is visible from the window

Listed 2 directories (ctrl+o to expand)

Before writing this, a few questions worth resolving so the spec isn't vague. Let me peek at how existing programs are structured.

Read 1 file (ctrl+o to expand)

A few things worth pinning down before writing the rules, because each changes the spec materially:

1. **Which window?** The CM has five windows with different lines of sight. Should this be parametric (crew picks a window) or fixed to one (e.g. the left rendezvous window)?

2. **Continuous tracking or one-shot?** Manoeuvre once to put the Moon in the window, or hold the Moon in the FOV as the spacecraft drifts?

3. **Crew confirmation before the manoeuvre fires?** The pattern elsewhere (P40) uses V50 PLEASE PERFORM to gate the manoeuvre. Same here?

4. **Program number?** P52 is IMU align, P22/P23 are taken. Free in the CM range – P24 or a P5x slot. Any preference, or should I just pick a plausible unused one?

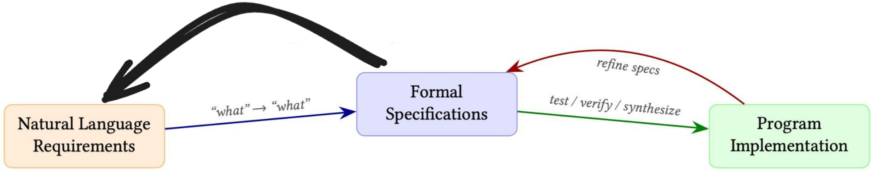
If you'd rather I just commit to defaults (parametric window, one-shot with V50 gate, P24), say "defaults" and I'll write it.

❯ |

```
WHEN [condition] THE SYSTEM SHALL [behaviour]
IF [condition] THEN THE SYSTEM SHALL [behaviour]
WHILE [state] THE SYSTEM SHALL [behaviour]
WHERE [feature included] THE SYSTEM SHALL [behaviour]
THE SYSTEM SHALL [behaviour]
```

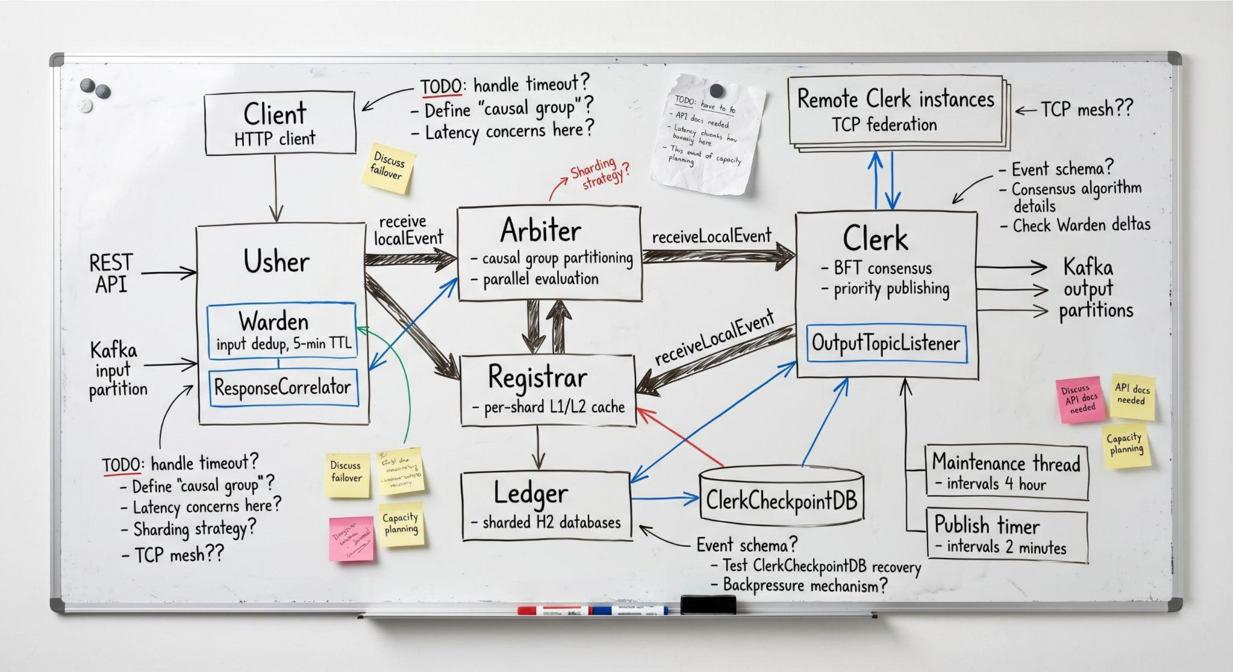
Status quo: AI code generation without formalisation

Proposed: intent formalisation bridges the gap



Intent Formalisation: the key challenge that will determine whether AI makes software more reliable or merely more abundant.

Shuvendu Lahiri, Microsoft Research, 2026



Formal Methods is a debuggable design… It can catch the bugs before they've been implemented. An hour of modeling will catch issues that days of writing tests will miss.

Hillel Wayne, The Business Case for Formal Methods (2020)

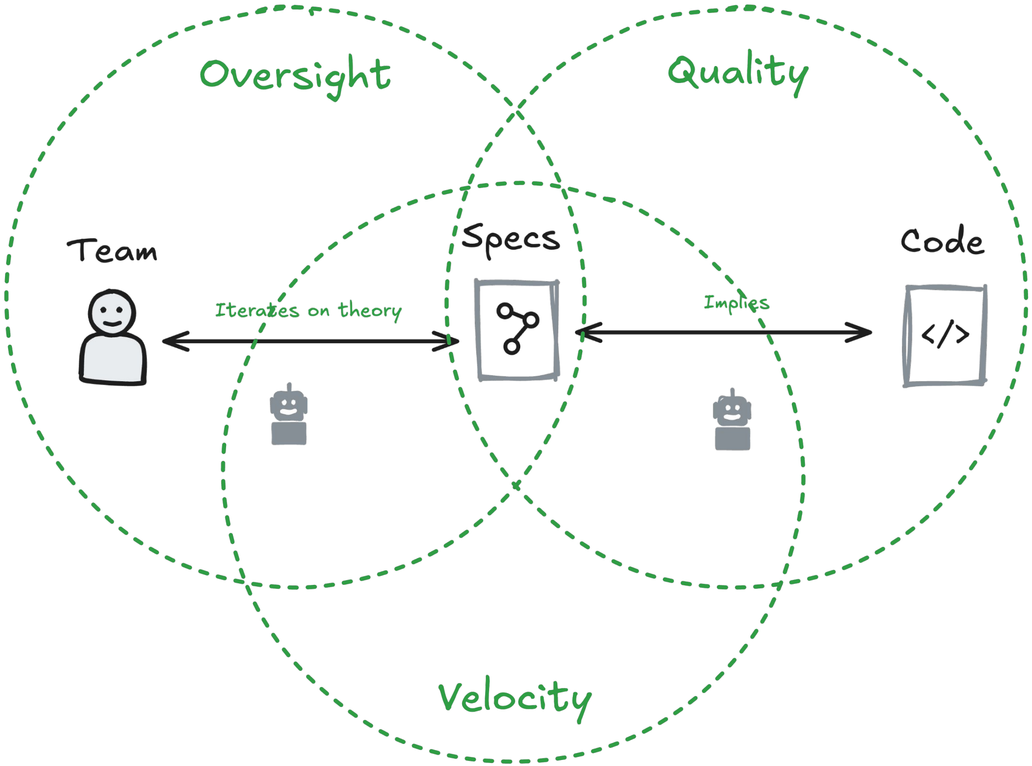
The -ilities

accessibility, accountability, accuracy, adaptability, administrability, affordability, agility, analyzability, auditability, autonomy, availability, compatibility, composability, confidentiality, configurability, convenience, correctness, credibility, customizability, debuggability, degradability, determinability, demonstrability, dependability, deployability, discoverability, distributability, durability, effectiveness, efficiency, elasticity, evolvability, extensibility, failure, transparency, familiarity, fault-tolerance, fidelity, flexibility, inspectability, installability, integrity, interactivity, interchangeability, interoperability, intuitiveness, learnability, localizability, maintainability, manageability, mobility, modifiability, modularity, observability, operability, orthogonality, portability, precision, predictability, process, capabilities, producibility, provability, recoverability, redundancy, relevance, reliability, repairability, repeatability, reproducibility, resilience, responsiveness, reusability, robustness, safety, scalability, seamlessness, self-sustainability, serviceability, securability, simplicity, stability, standards, compliance, supportability, survivability, sustainability, tailorability, testability, timeliness, traceability, transparency, ubiquity, understandability, upgradability, usability

The -ilities

accessibility, accountability, accuracy, adaptability, administrability, affordability, agility, analyzability, **auditability**, autonomy, availability, compatibility, composability, confidentiality, configurability, convenience, **correctness**, credibility, customizability, debuggability, degradability, determinability, demonstrability, dependability, deployability, discoverability, distributability, durability, effectiveness, efficiency, elasticity, evolvability, extensibility, failure, transparency, familiarity, fault-tolerance, fidelity, flexibility, **inspectability**, installability, integrity, interactivity, interchangeability, interoperability, intuitiveness, learnability, localizability, **maintainability**, manageability, mobility, modifiability, modularity, observability, operability, orthogonality, portability, precision, predictability, process, capabilities, producibility, provability, recoverability, redundancy, relevance, reliability, repairability, repeatability, reproducibility, resilience, responsiveness, reusability, robustness, safety, scalability, seamlessness, self-sustainability, serviceability, **securability**, simplicity, stability, standards, compliance, **supportability**, survivability, sustainability, tailorability, testability, timeliness, traceability, transparency, ubiquity, understandability, upgradability, **usability**

But then again...



Oversight
Quality
Team
Specs
Code
Iterates on theory
Implies
</>
Velocity

Grid Dynamics

trusted engineering partner for digital transformation

Thank you

Get Allium

http://juxt.github.io/allium

http://github.com/juxt/allium

Grid Dynamics Holdings, Inc.

6101 Bollinger Canyon Road,

Suite 465 / San Ramon, CA 94583

+1 650-523-5000

info@griddynamics.com

www.griddynamics.com