



Distilling Intent from Code

Behavioural specification for legacy analysis
and modernisation

Henry Garner, CTO & AI Chapter Lead **JUXT**
June 2026

Velocity

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

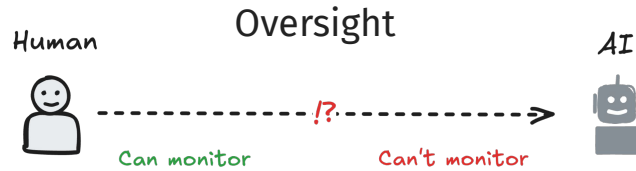
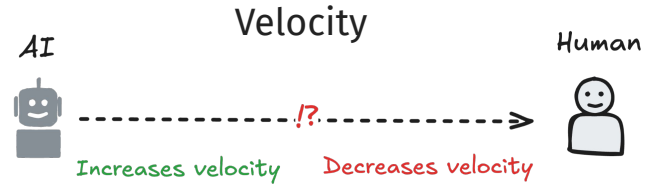
Oversight

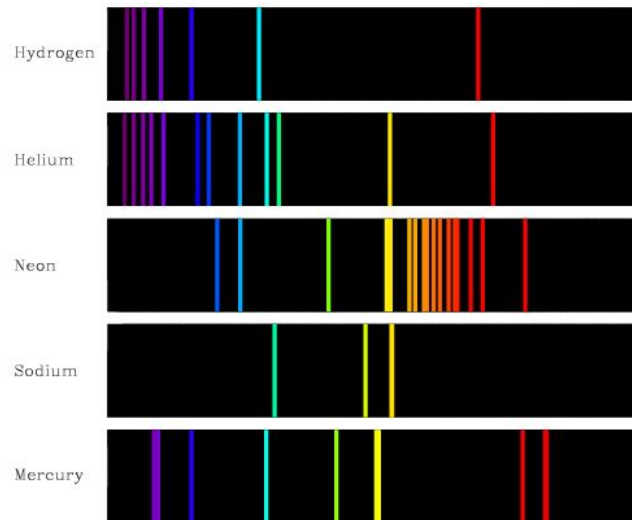
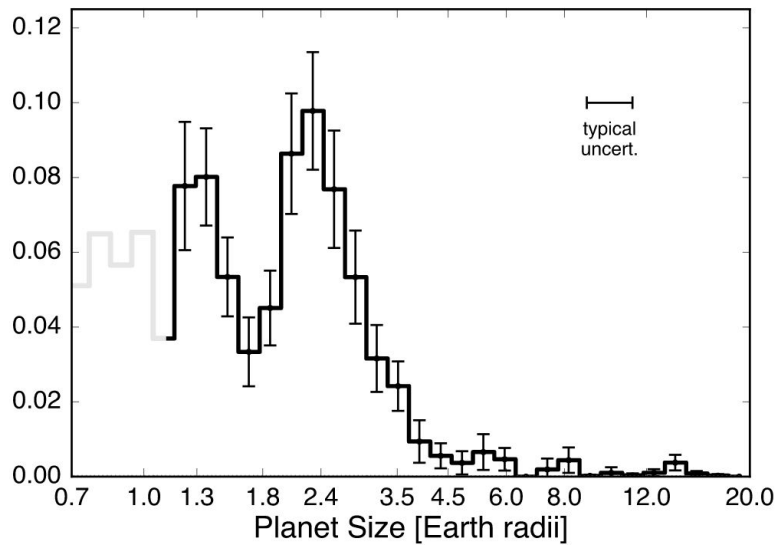
AI systems must be effectively overseen by natural persons

EU AI Act, Article 14

The human has been given an impossible task

Lisanne Bainbridge, *Ironies of Automation*





The Hacker News

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

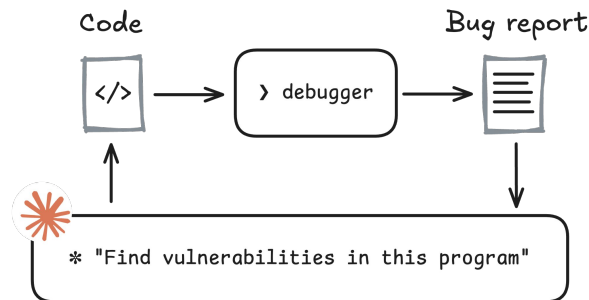
Ravie Lakshmanan May 23, 2026

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”

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

SEJ 29 May 2025

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

Jinyao Guo^{*1} Chengpeng Wang^{*1} Xiangzhe Xu¹ Zian Su¹ Xiangyu Zhang¹

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; Calcano et al., 2009; Sui and Xue, 2016; Shi et al., 2018a). Unfortunately, software systems are often non-executable and even uncompileable 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 fail

14 Oct 2025

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

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

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 task of postcondition generation *nl2postcond*. Our evaluation shows that (1) LLMs are generally effective

“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
Questions • Deferred Specs

```
1  -- allium: 3
2
3  enum QuoteStatus { quoted | ordered | rejected | expired }
4  enum ProductCode { equity | fx | bond | derivative }
5
6  entity Client {
7    id: Integer
8    allowed_products: List<ProductCode>
9    remaining_credit: Integer
10   daily_turnover_limit: Integer
11
12   -- Relationship: every Quote that names this Client
13   quotes: Quote with client = this
14
15   -- Projection: today's orders for this client
16   todays_orders: quotes where status = ordered and same_day(placed_at, now)
17 }
18
19 entity Quote {
20   id: Integer
21   client: Client
22   product: ProductCode
23   notional: Integer
24   status: QuoteStatus
25   placed_at: Timestamp when status = ordered
26 }
27
```

```
28 given {
29   quotes: List<Quote>
30   clients: List<Client>
31 }
32
33 rule PlaceOrder {
34   when: OrderRequested(quote)
35
36   requires:
37     quote.status = quoted
38     quote.product in quote.client.allowed_products
39     quote.notional <= quote.client.remaining_credit
40
41   ensures:
42     quote.status = ordered
43     quote.placed_at = now
44 }
45
46 invariant ClientWithinDailyTurnoverLimit {
47   -- Sum of notional across today's ordered quotes per client
48   -- must not exceed that client's daily turnover limit.
49   for c in Clients:
50     sum(c.todays_orders, q => q.notional) <= c.daily_turnover_limit
51 }
52
```



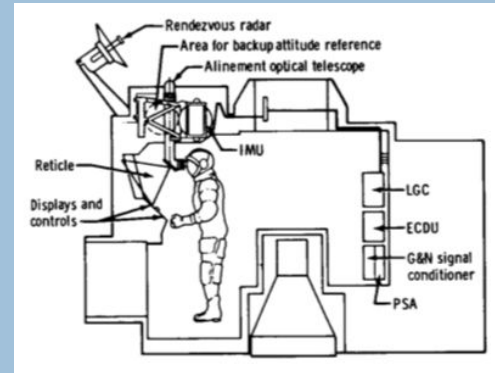
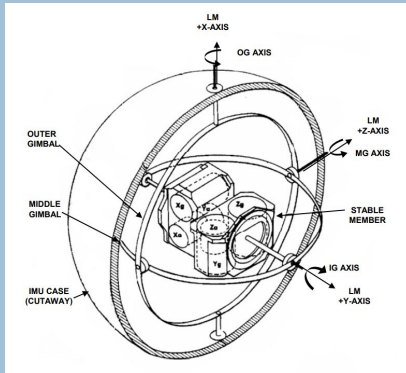
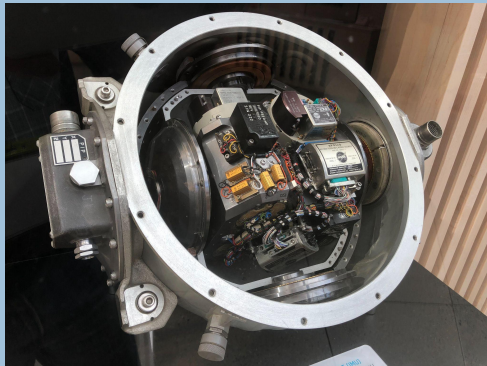
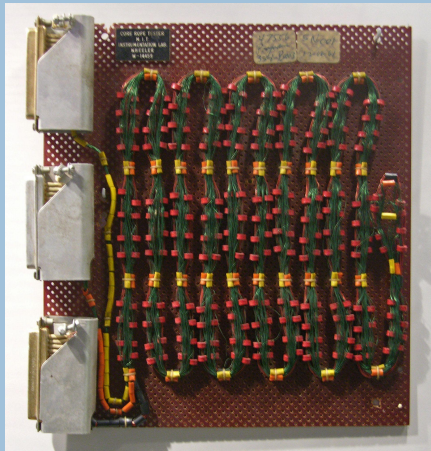
```
53 {
54   "category": "projection",
55   "description": "Verify projection Client.todays_orders filters correctly",
56   "id": "projection.Client.todays_orders",
57   "source_construct": "Client.todays_orders",
58   "source_span": {
59     "end": 553,
60     "start": 193
61   }
62 },
139 {
140   "category": "invariant",
141   "description": "Verify invariant ClientWithinDailyTurnoverLimit holds after
142   every state-changing rule that touches constrained entities",
143   "id": "invariant.ClientWithinDailyTurnoverLimit",
144   "source_construct": "ClientWithinDailyTurnoverLimit",
145   "source_span": {
146     "end": 1332,
147     "start": 1071
148   }
149 }
```

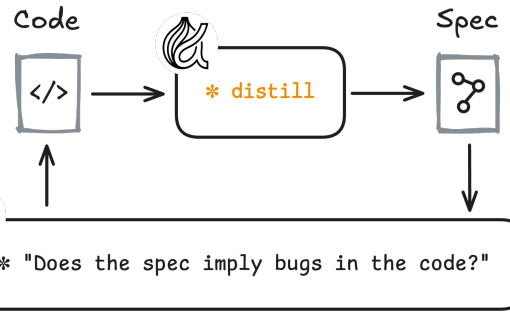
```
120 {
121   "category": "rule_failure",
122   "dependencies": {
123     "entities_read": [
124       "Quote"
125     ],
126     "entities_written": [
127       "Quote"
128     ],
129     "trigger_source": "external"
130   },
131   "description": "Verify rule PlaceOrder is rejected when requires clause fails",
132   "id": "rule-failure.PlaceOrder.1",
133   "source_construct": "PlaceOrder",
134   "source_span": {
135     "end": 992,
136     "start": 842
137   }
138 },
```



TAPE HEAD RUN 4 VERIFICATION C4164 TEST RUN DATE 10/23/68 EDIT DATE 23PCT68 SDS TIME 1640 AGC TIME 1639:56 10 777 PAGE 1																			
PROGRAM	VERB 01	NRUN 16	01	37777	R2	R3	00026	FLASH 0	DSPTAB +11	00000									
PROGRAM	VERB 01	NRUN 16	01	37777	R2	R3	00026	FLASH 0	DSPTAB +11	00000									
SDS	REFERENCE MATRIX	ASC	REFERENCE MATRIX	ASC	ATTITUDE MATRIX														
***1077304	**79107081	**45108113	**11022304	**79159081	**45108113	**11026094	**79164018	**45108113	**11026094	**79164018	**45108113	**11026094	**79164018	**45108113					
***11018884	**39839706	**03688201	**31108880	**39838706	**03688201	**31108884	**39838706	**03688201	**31108884	**39838706	**03688201	**31108884	**39838706	**03688201					
REFORMAT	ER	DIPHAL	ANGLES	SDS	CDU	ASC	CDU	ASC	SHAFT	SDS	SHAFT	X	ERRN	LBS	DR	LBS	ALT	LBS	CR
Z4	**0000	ALFA	**+00	**+00	**+00	**+00	**+00	**+00	SDS	TRUN	SDS	TRUN	V	ERRN	**+00	**+00	**+00	**+00	**+00
V4	**0000	AREA	**+01	**+01	**+01	**+01	**+01	**+01	**+01	**+01	**+01	**+01	**+01	**+01	**+01	**+01	**+01	**+01	**+01
STATE	VECT	Rx	BY	RZ	ASS	BY	VX	VY	VZ	SS	TIME								
SDS	CM	S726386+1	2910736+6	-1666192+7	6633727+3	-3911+80	5801+87	-3917+87	7784+94	164+42									
ASC	CM	S726386+1	2910736+6	-1666192+7	6633727+3	-3911+80	5801+87	-3917+87	7784+94	164+42									
DELTA			**+00	**+00	**+00	**+00	**+00	**+00	**+00	**+00									
SDS	LM	S726666+1	2944205+0	-1676134+5	660504+8	-391+80	5789+80	-3907+81	7742+87	164+42									
ASC	LM	S726666+1	2944205+0	-1676134+5	660504+8	-391+80	5789+80	-3907+81	7742+87	164+42									
DELTA			**+00	**+00	**+00	**+00	**+00	**+00	**+00	**+00									
POSITION	LM	LM	LM	ACT	ARRN	ARRN	ARRN	ARRN	ARRN	ARRN	RANGE	DATE							
SDS	CM	**+1483	**+1821	284629+3	282611+2	282712+6	282712+6	282712+6	282712+6	282712+6	**+028								
SDS	LM	**+1483	**+1821	284629+3	282611+2	282712+6	282712+6	282712+6	282712+6	282712+6	**+028								
FLASMO	00000	FLASMO	1 00000	FLASMO	2 00000	FLASMO	3 10000	FLASMO	4 00000	FLASMO	5 40000	FLASMO	6 00000	FLASMO	7 00000				
FLASMO	8 00000	FLASMO	9 00000																
REFLAGS	00111	REFLAGS	00111	REFLAGS	00111	REFLAGS	00111	REFLAGS	00111	REFLAGS	00111	REFLAGS	00111	REFLAGS	00111				
FAILSOP	00000	FAILSOP	00000	FAILSOP	00000	FAILSOP	00000	FAILSOP	00000	FAILSOP	00000	FAILSOP	00000	FAILSOP	00000				
REFLAGS	77776	REFLAGS	11111	REFLAGS	11111	REFLAGS	11111	REFLAGS	11111	REFLAGS	11111	REFLAGS	11111	REFLAGS	11111				
CHAN	13	00100	CHAN	14	00000	CHAN	30	37273	CHAN	31	37777	CHAN	32	77777					
CDU	**000000	CDU	**1078433E+01	CDU	**000000	CDU	**191768V0	ADST	**301218A										
ADST	**102643A	ADST	**14075797	AK	**000000	AK	**271782L3	AK	**118027878										
THETAD	**+161307+	THETAD	**+436890	THETAD	Z	10+8398E	TIS	2370+800	TEST	1	**000000								
TEST	**+1666667+	TEST	**1666667+	STAR	Z	**03726E+01	STAR	Z	**03726E+01	STAR	Z	**000000							
STAR	R51	**+06377+	STAR	R51	**+06377+	STAR	R51	**+06377+	STAR	R51	**+06377+	STAR	R51	**+06377+					
STAR	R51	**+06377+	STAR	R51	**+06377+	STAR	R51	**+06377+	STAR	R51	**+06377+	STAR	R51	**+06377+					
VERT	Z	**+07175	VERT	Z	**+07175	VERT	Z	**+07175	VERT	Z	**+07175	VERT	Z	**+07175					
THETAD	Z	**+030000	THETAD	Z	**+030000	THETAD	Z	**+030000	THETAD	Z	**+030000	THETAD	Z	**+030000					
LAUNCH	Z	**+073786C	LAUNCH	Z	**+073786C	LAUNCH	Z	**+073786C	LAUNCH	Z	**+073786C	LAUNCH	Z	**+073786C					
THETAD	Z	**+1111	THETAD	Z	**+1111	THETAD	Z	**+1111	THETAD	Z	**+1111	THETAD	Z	**+1111					
Z	CE	**+01	Z	CE	**+01	Z	CE	**+01	Z	CE	**+01	Z	CE	**+01					
TZ	8471	TZ	8471	TZ	8471	TZ	8471	TZ	8471	TZ	8471	TZ	8471	TZ	8471				
W	DSG/SE	**+05	W	DSG/SE	**+05	W	DSG/SE	**+05	W	DSG/SE	**+05	W	DSG/SE	**+05					
ASZ	CH/SE	**+06	ASZ	CH/SE	**+06	ASZ	CH/SE	**+06	ASZ	CH/SE	**+06	ASZ	CH/SE	**+06					
ATSM	RZ	**+133	ATSM	RZ	**+133	ATSM	RZ	**+133	ATSM	RZ	**+133	ATSM	RZ	**+133					

# Page 801			
175			
176	CAF	TWO	# WCHPHASE = 2 ----> VERTICAL: P65, P66, P67
177	TS	WCHPHOLD	
178	TS	WCHPHASE	
179	TC	BANKCALL	# TEMPORARY, I HOPE HOPE HOPE
180	CADR	STOPRATE	# TEMPORARY, I HOPE HOPE HOPE
181	TC	DOWNFLAG	# PERMIT X-AXIS OVERRIDE
182	ADRES	XOVINFLG	
183	TC	DOWNFLAG	
184	ADRES	REDFLAG	
185	TCF	VERTGUID	
186			
187	STARTP67	TC	NEWMODEX # NO HARM IN "STARTING" P67 OVER AND OVER
188	DEC	67	# SO NO NEED FOR A FASTCHG AND NO NEED
189	CAF	ZERO	# TO SEE IF ALREADY IN P67.
190	TS	RODCOUNT	
191	CAF	TEN	
192	TCF	VRTSTART	
193			
194	STABL7	CAF	BIT13 # IS UN-ATTITUDE-HOLD DISCRETE PRESENT?
195		EXTEND	
196		RAND	CHAN31
197		CCS	A
198		TCF	GUILDRET # YES ALL'S WELL
199			
200	P66NOW?	CS	MODREG
201		AD	DEC66
202		EXTEND	
203		BZF	RESTART?





BURN_BABY_BURN
 PINBALL_GAME_BUTTONS_AND_LIGHTS
 TRASHY_LITTLE_SUBROUTINES

```

311  -- NVSUB = internal program interface -----
312
313  rule NvsubRequest {
314    when: NvsubCall(verb_code, noun_code)
315    requires: dsky.busy = false
316    ensures:
317      dsky.current_verb = verb_code
318      dsky.current_noun = noun_code
319      dsky.busy = true
320      dsky.busy_owner = `calling_job`
321      dsky.input_source = internal
322      VerbDispatch(verb_code: verb_code, noun_code: noun_code)
323  }
324
325  rule NvsubBusyWait {
326    when: NvsubCall(verb_code, noun_code)
327    requires: dsky.busy = true
328    ensures:
329      JobSleep(job: `calling_job`)
330
331    @guidance
332    -- The caller enters JOBSLEEP and will be woken when the
333    -- current DSKY user completes. There is no timeout on this
334    -- wait. If the busy owner dies without releasing the DSKY,
335    -- the waiting job sleeps indefinitely.
336  }
337
338  rule DskyRelease {
339    -- The busy owner releases the DSKY, waking any waiting job.
340    when: DskyRelease(owner)
341    requires:
342      dsky.busy = true
343      dsky.busy_owner = owner
344    ensures:
345      dsky.busy = false
346      dsky.busy_owner = null
347  }
  
```

```

430  -- GyroTorquing (IMV: 455) -----
431
432  rule GyroTorque {
433    -- Sends gyro torquing pulse commands. Reserves the gyros,
434    -- enables power supply, and dispatches pulses per axis.
435    when: GyroTorque(command: GyroTorqueCommand)
436
437    requires:
438      imu.mode != caged
439      imu.gyros_busy = false
440
441    ensures:
442      imu.gyros_busy = true
443      GyroTorqueStarted()
444  }
445
446  rule GyroTorqueBusy {
447    -- Gyros already reserved by another torquing operation.
448    -- Caller sleeps until LGYRO is cleared.
449    when: GyroTorque(command: GyroTorqueCommand)
450
451    requires: imu.gyros_busy = true
452
453    ensures:
454      JobSleep(job: calling_job())
455  }
  
```

!!

MIT/DL PROGRAM CHANGE ROUTING SLIP

- COLOSSUS 2E
- COLOSSUS 3
- COLOSSUS 3A
- COLOSSUS
- LUMINARY 1D
- LUMINARY 1E
- LUMINARY 1F
- LUMINARY

- MIT Approved PCN
- MIT Approved ADR
- NASA Approved PCR
- NASA Approved PCN
- NASA Approved Software Anomaly
- MIT Approved Software Anomaly

A. Coding

Begin coding immediately

ACTION: _____

Program Supervisor's Approval: _____

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

B. GSOP Preparation

Prepare GSOP revisions for MDRE 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: _____

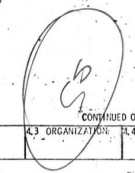
Date: 7-8-70

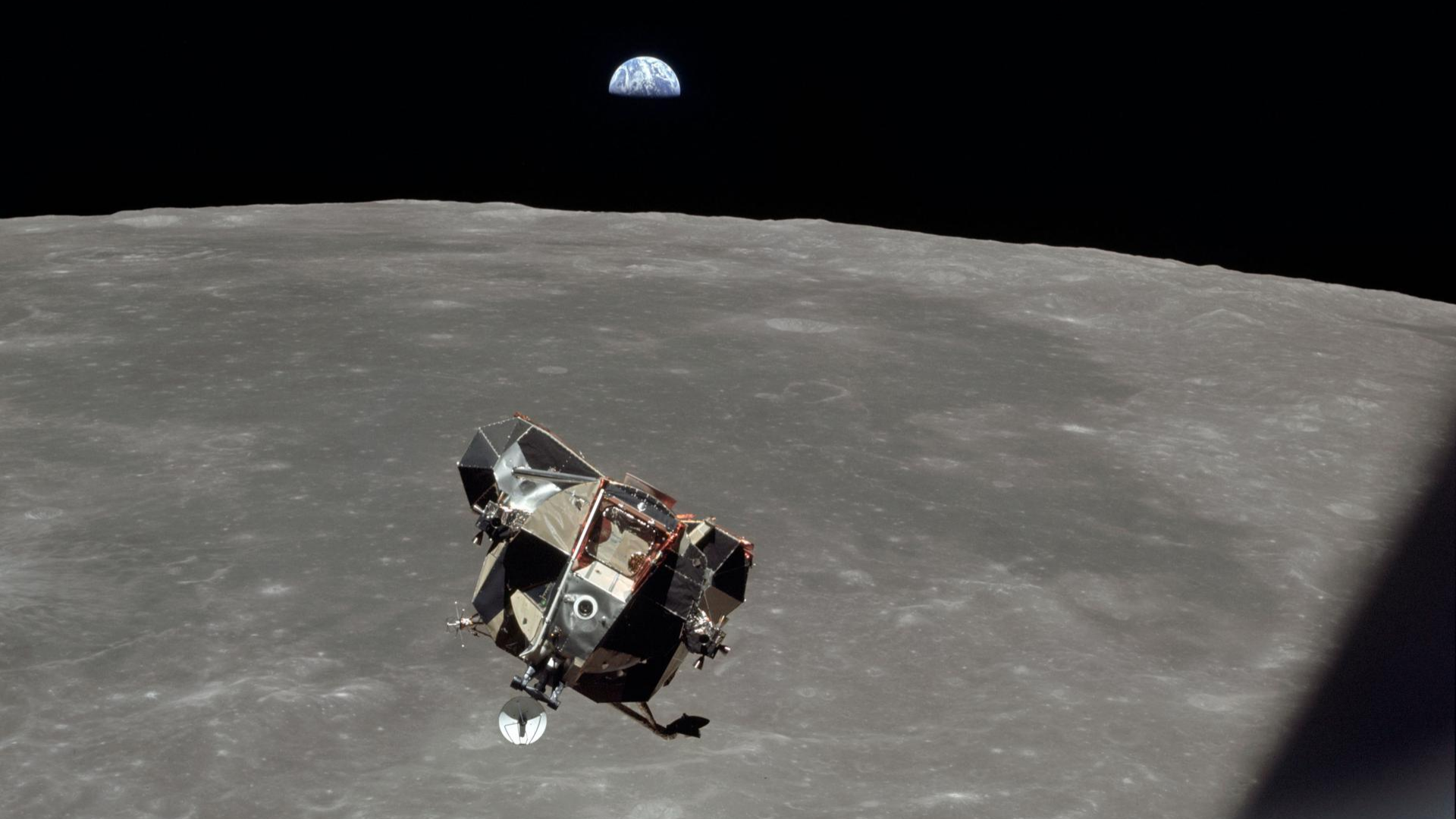
PCR/PCN # _____
ANOMALY # L-1D-02
ADR # _____

Bob Crelli

Margaret Hamelton

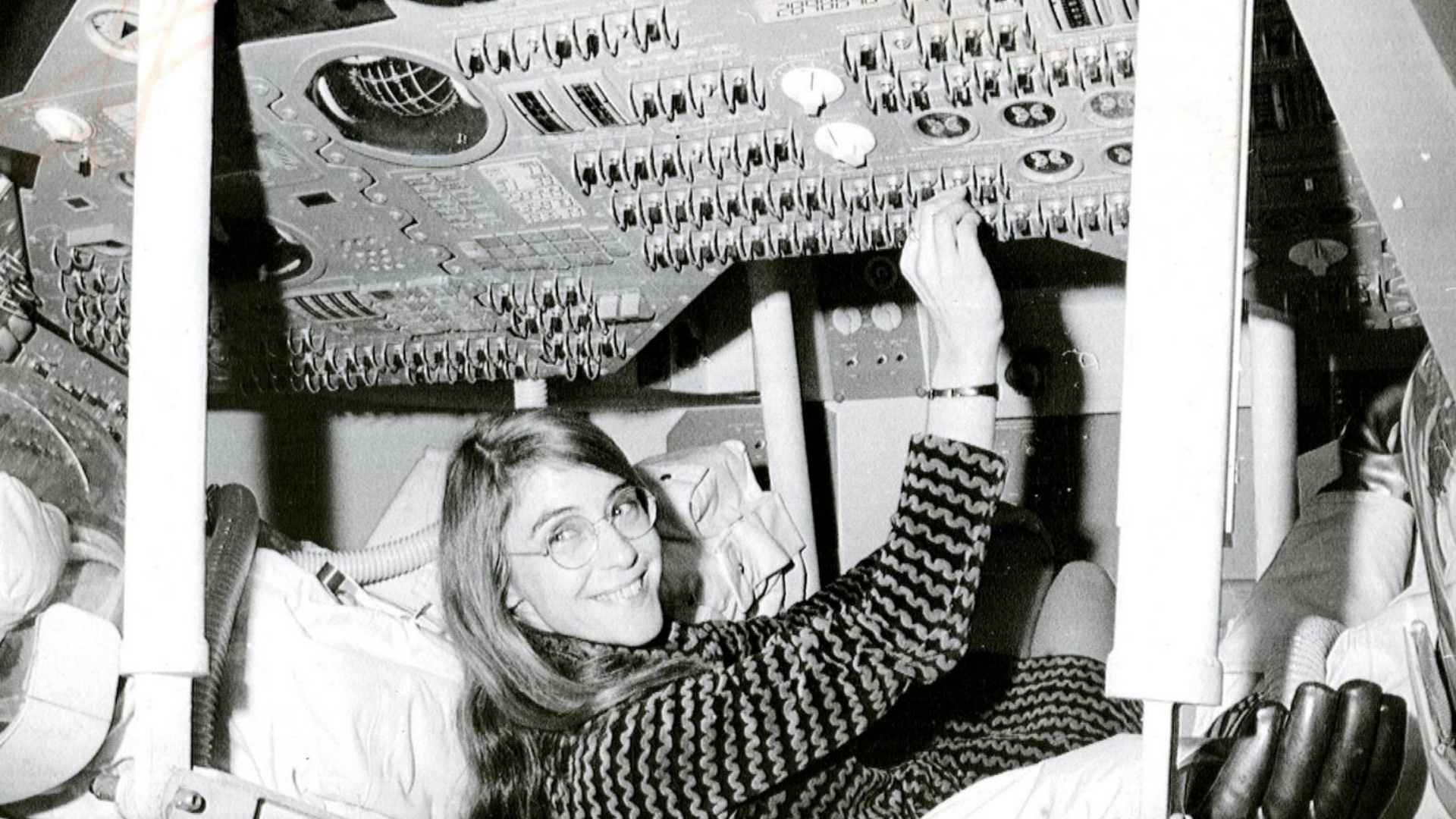
MIT/IL SOFTWARE ANOMALY REPORT

MIT/IL SOFTWARE ANOMALY REPORT		NSC REPORT NO. <u>L-1D-02</u>	
1.1 AUTHOR: <u>N. Barnert/P. Rye</u>	1.2 ORGANIZATION: <u>MIT/CSDL</u>	1.3 DATE: <u>5/19/70</u>	1.4 ORIGINATOR CONTROL NO. <u>LUMINARY</u>
1.5 DESCRIPTION OF ANOMALY: <u>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.</u>			
CONTINUED ON PAGE			
1.6 DESCRIPTION OF RUN: <u>Satanche Level 3 test</u>			
CONTINUED ON PAGE			
- MIT ANALYSIS -			
2.1 CAUSE: <u>"LGYRO" not zeroed if pulse torquing ends due to cage or coarse align; subsequent jobs attempting torquing will stack up via JOBSLEEP.</u>			
CONTINUED ON PAGE			
2.2 RECOGNITION: <u>31202 Abort</u>			
CONTINUED ON PAGE			
2.3 MISSION EFFECT: <u>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.</u>			
CONTINUED ON PAGE			
2.4 AVOIDANCE PROCEDURE: <u>Do not coarse align during Average G; or else set GCOMPSW negative. Zero LGYRO whenever cage is pushed.</u>			
CONTINUED ON PAGE			
2.5 RECOVERY PROCEDURE: <u>Set GCOMPSW negative (terminates compensation).</u>			
CONTINUED ON PAGE			
2.6 PROGRAM CORRECTION: <u>Clear LGYRO and wake possible sleeping jobs before exiting pulse torquing routine.</u>			
CONTINUED ON PAGE			
2.7 RECOMMENDED DISPOSITION (Fix, Work-around, etc.): <u>Fix for Luminary 1E</u>			
CONTINUED ON PAGE			
2.8 RECOMMENDED RE-TESTING: <u>Level 2 test.</u>			
CONTINUED ON PAGE		2.9 MIT/IL SIGNATURE: <i>[Signature]</i>	2.10 DATE: <u>7-8-70</u>
3.1 NASA DIRECTION:		4.1 CLOSING ACTION TAKEN:	
CONTINUED ON PAGE			
3.2 NASA/IL SIGNATURE:	3.3 ORGANIZATION:	3.4 DATE:	4.2 SIGNATURE:
CONTINUED ON PAGE		CONTINUED ON PAGE	









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.

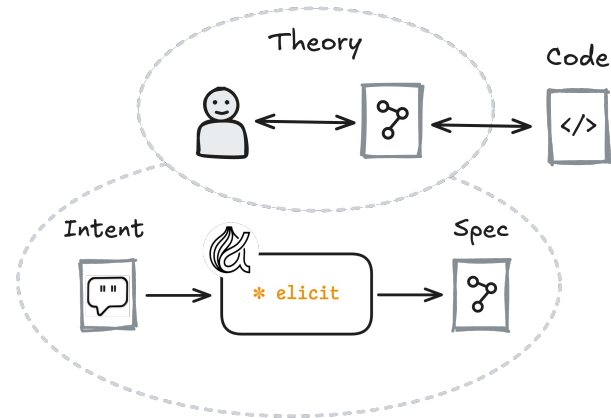
“PROGRAMMING AS THEORY BUILDING”

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 GOTOP00H; 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.

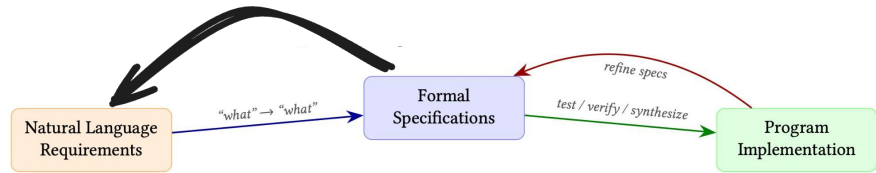
> |

Status quo: AI code generation without formalisation

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]

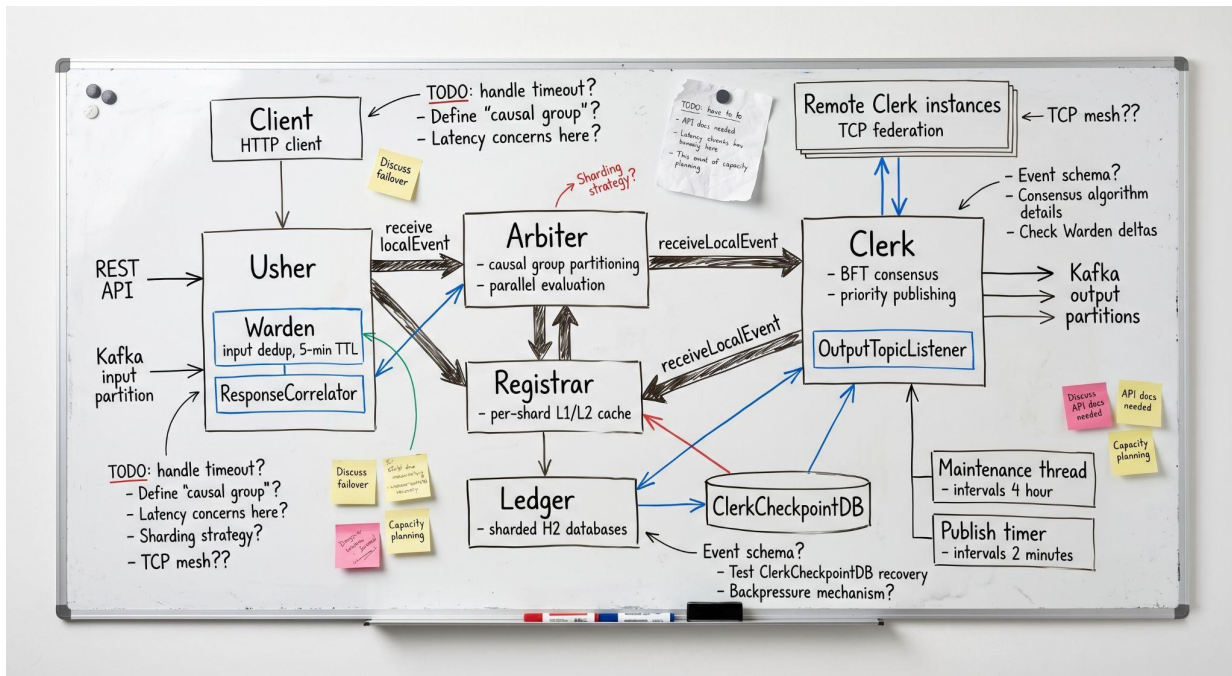


Proposed: intent formalisation bridges the gap



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

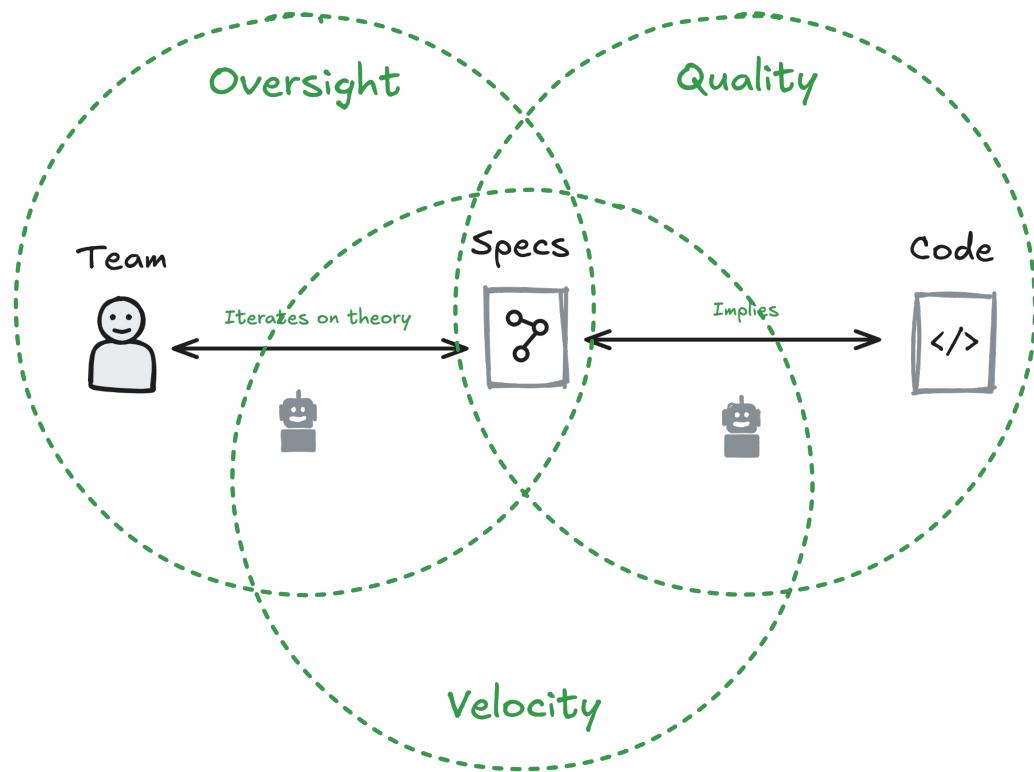
accessibility, accountability, accuracy, adaptability, administrability, affordability, agility, analyzability, auditability, autonomy, availability, compatibility, composability, confidentiality, configurability, convenience, correctness, credibility, customizability, debuggability, degradability, determinability, 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...









Grid Dynamics

trusted engineering partner for digital transformation

Thank you

<http://juxt.github.io/allium>

<http://github.com/juxt/allium>