ה-Agents הפכו את הקוד לחינמי, אבל התחזוקה שלו לא הוזלה

איך מנהלים codebase שנכתב ברובו על ידי coding agents בלי לטבוע באנטרופיה ובלי למחוק הכול: ניקוי קטן כל שבוע, עם ראיות

Dmitry Buykin (Maersk) · AI Engineer · 11 בנובמבר 2026
משך: כ-18 דקות · YouTube
סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה של כ-18 דקות מ-AI Engineer (על השקפים מופיע הלוגו Code Summit). מרצה בה Dmitry Buykin מ-Maersk, שמתחזק מערכת גדולה בייצור שרובה נכתבה על ידי coding agents. המערכת רצה ב-110 מדינות, ובה כ-10,000 תרחישים של הוראות שכתובות כ-prose. לאורך כל ההרצאה הוא משתמש במטאפורה של Thanos ואבני האינסוף: שש אבנים, ומאחורי כל אחת שאלה (Time, Space, Reality, Mind, Power, Soul). דוגמה אחת חוזרת בכל הפרקים: שינוי הסכום של הזמנה שכבר אושרה, 1,250 מול תקרה של 1,000.

הטענה המרכזית: agents הוזילו את כתיבת הקוד, אבל את האנטרופיה הם הוזילו עוד יותר. Buykin מתאר שתי ברירות גרועות, לטבוע בכאוס או לעשות snap (למחוק ולשכתב) ולשבור דברים שאנשים משתמשים בהם, ומציע דרך שלישית: להישאר מתחת ל-breach line, הקו שמעליו יש יותר שינויים ממה שמישהו מסוגל לבדוק. עושים את זה בניקויים קטנים כל שבוע, עם לולאות משוב מובנות, evals שבבעלות מומחים, guardrails בזמן ריצה וראיות שאפשר לספור.

איך לקרוא את המסמך

- **1. תקציר מנהלים:** הטענות המרכזיות עם ראיות וחותמות זמן, וגם מה לא נאמר.
- **2. פרולוג:** ה-snap שלא החזיק, ה-breach line ושש השאלות.
- **3. הכללים שאנשים נושאים:** prose כשלושה רבעים מהמערכת, ולמה רק אנשים יכולים לקבוע מה נכון.
- **4. הנתיבים שחוזרים:** מתי prose הופך לקוד, ו-decision table שבבעלות המומחים.
- **5. החלטה הופכת לפעולה:** מה המודל רואה, ושש בקרות לפני כל כתיבה.
- **6. מעבר לקשב של אדם אחד:** שחיקה, מכשירי מדידה, לולאות משוב ו-ledgers.
- **7. ראיות לשינוי ומה ראוי לשרוד:** מהירות האימות כצוואר בקבוק, ריפליי של משימות אמיתיות וגיזום שבועי.
- **8. מה עושים מחר:** 18 השאלות, צעדים מעשיים ושאלות פתוחות. בסוף יש מילון מונחים.

הערת מקור המסמך הופק אוטומטית מהתמליל ומהשקפים של ההקלטה. ייתכן שזיהוי הדיבור (ASR) שיבש שמות ומונחים. הציטוטים מועתקים מילה במילה ונבדקו מול חותמת הזמן שלהם. לא נוסף שום מידע ממקורות חיצוניים.

1. תקציר מנהלים

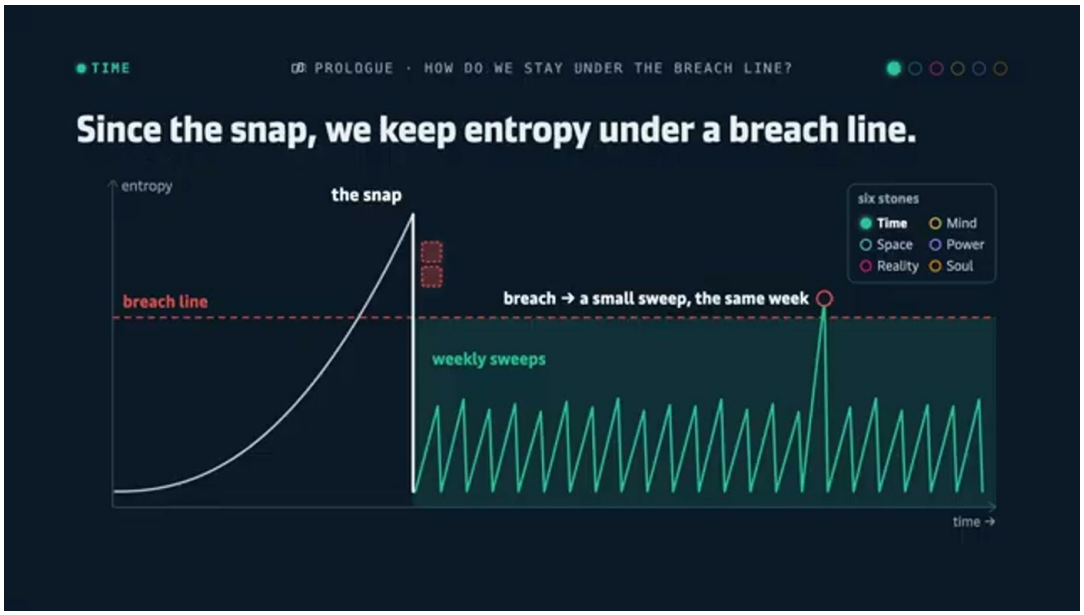
Buykin לא מדבר על איך לכתוב קוד מהר יותר עם agents. הוא מדבר על מה קורה כמה חודשים אחרי: טסטים עוברים בזמן שהייצור נשבר, כללים סותרים זה את זה, וכמות השינויים גדלה מעבר למה שמישהו מסוגל לסקור. התשובה שלו היא תהליך ולא כלי: לקבוע מי מחליט מה נכון, לצמצם את מה שהמודל רואה ומה מותר לו לשנות, לבנות לולאות משוב שלא עוברות דרך הראש של אדם אחד, ולמחוק רק כשיש ראיות.

- **אנטרופיה, לא כתיבה, היא הבעיה:** "Coding agents made code cheap. They made entropy even cheaper.". שכתוב גורף (snap) לא החזיק מעמד, כי ה-agents המשיכו לכתוב והבלגן חזר. מכאן הרעיון של breach line וניקוי שבועי.
- **רוב המערכת היא prose, וגם הוא צריך הנדסה:** לפי הערכתו כשלושה רבעים מהמערכת הם prose. ב-mutation testing על משפטים "almost none turned red", ובשקף: 10 מתוך 11 משפטים שבורים עברו את כל הטסטים.
- **אנשים קובעים מה נכון, לא טסטים:** מומחי עסק חותמים על הטקסט, ומפתחים חופרים ב-traces כדי להבין איך המערכות מחוברות. "Business tells us what, we work out how."
- **prose הופך לקוד רק כשהוא חוזר ועקבי:** walk שחוזר על עצמו ומתנהג אותו דבר בכל המדינות הופך ל-workflow tool. המומחים מחזיקים את ה-decision table, והמודל רק בוחר שורה. אם אף שורה לא מתאימה, אדם לוקח את המשימה.
- **שליטה מתחילה במה שהמודל רואה:** כשנתנו למודל לבחור מה לטעון, הוא טען הכול והמציא בדיקת תמחור. עכשיו ה-runtime מעביר לו שלוש עובדות, ולפני כל כתיבה יש שש בקרות.
- **הקשב של הארכיטקט הוא משאב שצריך להגן עליו:** הוא נשחק, הפסיק לקרוא כל שינוי ובנה מכשירים שמודדים את קצב השינוי. בדיקה חדשה הורצה קודם על כמה אלפי sessions מהעבר, ונמצא שהייתה מתריעה על פחות מ-1% מהם.
- **מהירות האימות היא צוואר הבקבוק:** "My agent spent almost 9 tenths of that time on a change just checking it.". בשקף מופיע 89%. בבדיקת backlog התברר שרק כרבע מהפריטים היה אפשר באמת לסגור.
- **סוקר נפרד בלי ההנמקה של הכותב:** agent שני בודק לפי checklist ולא רואה את ההנמקה של ה-agent שכתב, ולכן אי אפשר לשכנע אותו להסכים. במהלך חצי שנה הוא הפך לרוב הבדיקות של Buykin.
- **גיזום שבועי במקום snap:** ratchet על מדדים, ומחיקה רק עם ראיות. וריאנט שיוצא משימוש לוקח איתו את הטסטים שלו. "I haven't had to snap again."

מה לא נאמר

- **אין מספרי עלות או היקף:** לא נאמר מה גודל ה-codebase, כמה אנשים בצוות ובאילו coding agents או מודלים משתמשים.
- **אין מדדי לפני ואחרי:** לא הוצגו נתונים על ירידה בתקלות או באירועים בייצור. הטענה "לא נדרש snap נוסף" נשענת על התרשמות אישית.
- **חלק מהשיטה עוד לא קיים:** שני gauges עובדים ושישה מתוכננים. גם השלב שבו המודל לוקח החלטות קטנות ו-workflows מחברים ביניהן "עוד לא נבנה".
- **המספר 89% מגיע ממקור אחד:** זו היסטוריית ה-prompts של המרצה עצמו, בלי baseline ובלי השוואה.
- **הכלים לא הוצגו:** ה-compiler וה-linter ל-prose, ה-ledgers ומסך המשוב מתוארים ברמה הקונספטואלית בלבד. אחד המסכים מסומן בשקף כ-illustrative, ולא ברור אם דוגמת ההזמנה לקוחה מהמערכת האמיתית.

2. פרולוג: ה-snap שלא החזיק, וה-breach line

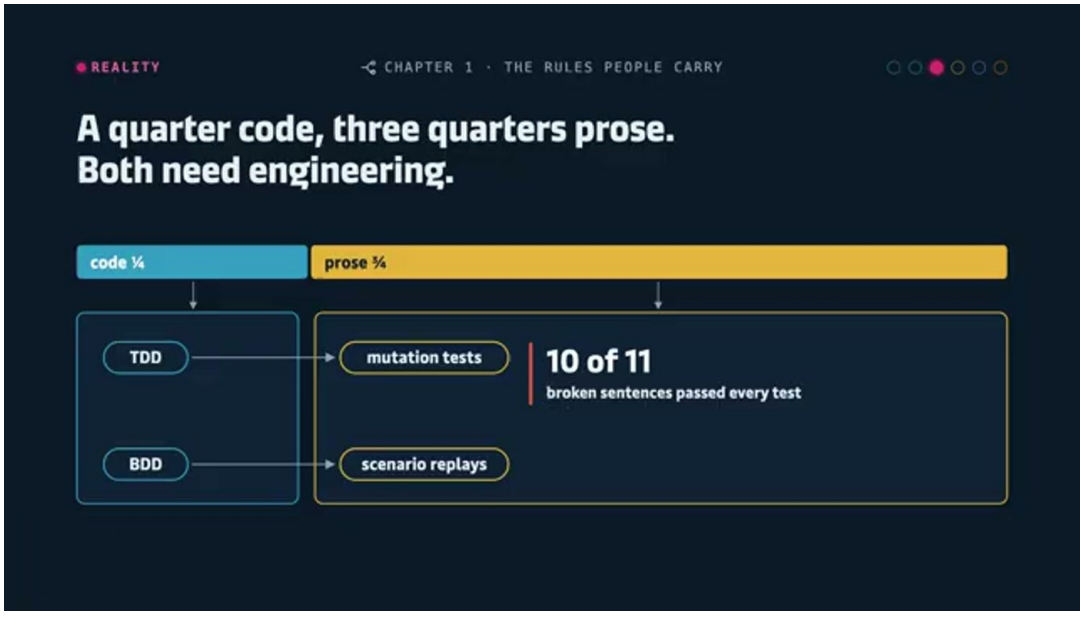


גרף אנטרופיה לאורך זמן: עלייה חדה עד ה-snap, ואחריו דפוס של שיני מסור (weekly sweeps) מתחת ל-breach line. ליד כל חריגה כתוב: breach → a small sweep, the same week. בפינה מופיע מקרא של שש האבנים.

Buykin מספר שפעם אחת הוא עשה snap. כשהבין מה ה-agents שלו עשו, הוא מחק טסטים ו-boilerplate שאף אחד לא השתמש בהם, נאלץ לשכתב את כל הפרויקט ואיבד בדרך חלקים שימושיים. מכאן הדילמה שלו: "Drown in chaos or break what people use?". גם השכתוב לא החזיק: "The agents kept writing and the mess grew back."

לכן הוא סימן קו. מעליו יש יותר שינויים ממה שמישהו מסוגל לבדוק. האפשרות השלישית היא להישאר מתחתיו בעזרת ניקויים קטנים כל שבוע ולולאות משוב שבנויות לתוך המערכת. כל פרק בהרצאה מוקדש לאחת מ-"six questions": Time (איך נשארים מתחת לקו), Space (לאן מותר לשינוי להגיע), Reality (האם זה באמת נכון), Mind (מה המודל יודע), Power (מה מותר לשנות) ו-Soul (מי שופט, ובאיזה מחיר).

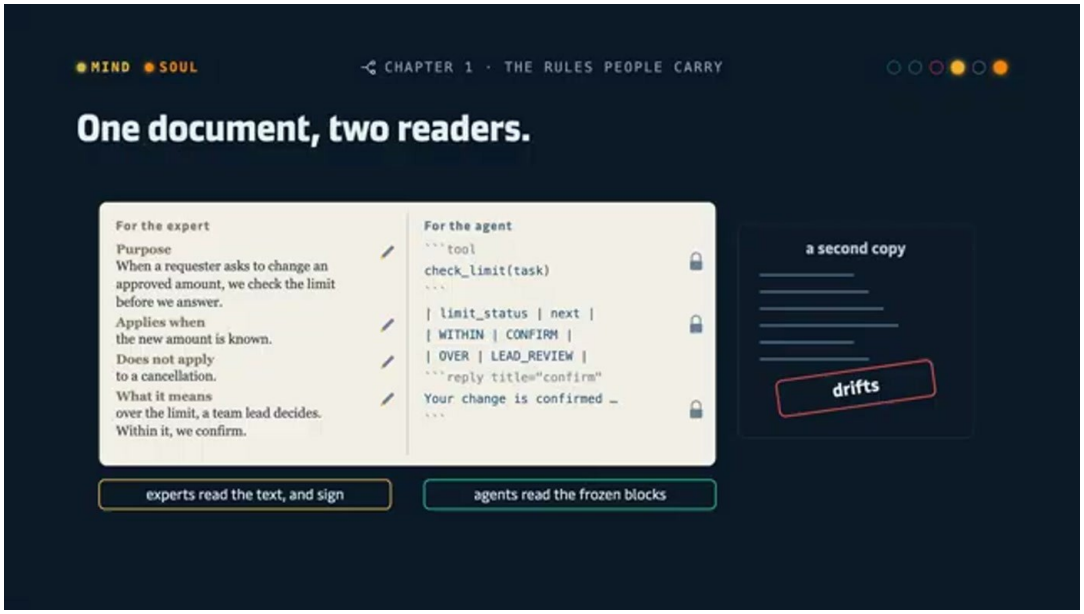
3. הכללים שאנשים נושאים



A quarter code, three quarters prose: את הקוד מכסים TDD ו-BDD, את ה-prose מכסים mutation tests ו-scenario replays. בשקף מודגש: 10 of 11 broken sentences passed every test.

במערכת יש שני סוגי agents: coding agent שכותב את ההוראות, ו-task agent שמבצע אותן בייצור. "We run this in 110 countries with about 10 000 scenarios of prose instructions". כל משימה נוחתת ב-slice, כלומר תהליך אחד במדינה אחת, וכברירת מחדל שינוי נשאר בתוך ה-slice שלו. זו שאלת ה-Space. מאחר שלפי הערכתו כשלושה רבעים מהמערכת הם prose, ניסו להחיל עליו mutation testing: שוברים משפט בכוונה ובודקים אם טסט כלשהו נכשל. כמעט אף טסט לא נכשל. המסקנה שלו: "With agents, a rewrite is easy. The hard part is knowing what has to survive it". בקוד, טסטים יכולים לענות על השאלה הזו. ב-prose רק אנשים יכולים.

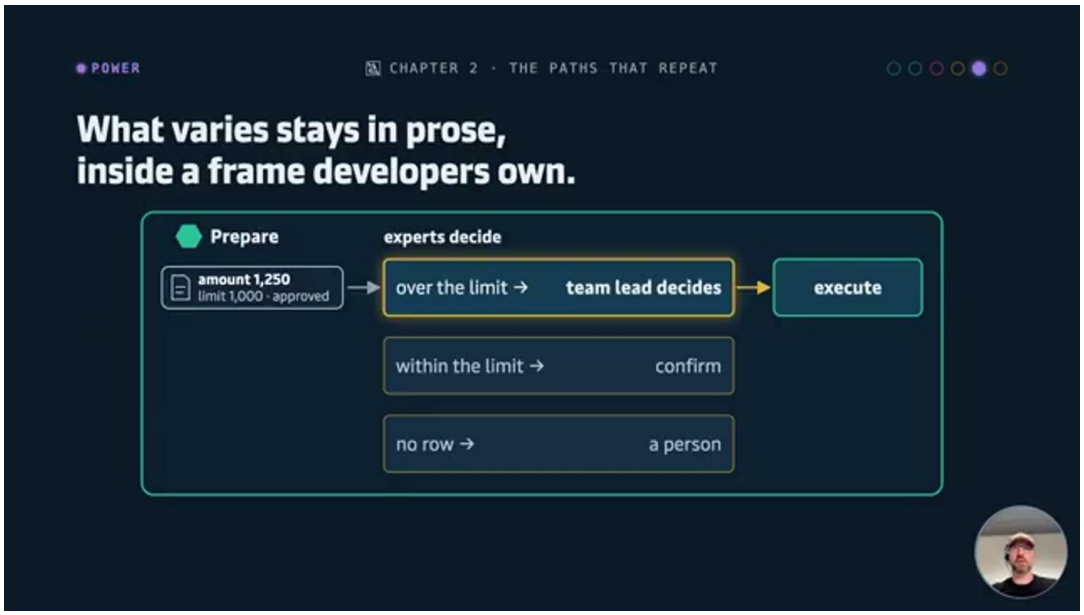
המשפט "check with the team" מסתיר שאלה אמיתית: האם זה שלב שדורש אדם, או שאפשר להפוך אותו לאוטומטי? המומחים לא יודעים לענות, כי הם לא רואים איך המערכות מחוברות בפנים. לכן המפתחים חופרים ב-traces וב-logs בעזרת ה-coding agents ומחליטים לפי מה שהם מוצאים. בדוגמה של ההזמנה, ההחלטה של ה-team lead נשארה שלב אנושי. מה שהגביל אוטומציה היה אילו נתונים אפשר לחבר. "Developers see what's under the water. And that's where most bugs live".



One document, two readers: מצד אחד טקסט למומחים, שקוראים וחותמים עליו. מצד שני בלוקים קפואים ומדויקים שה-agent קורא. עותק שני של המסמך מסומן drifts.

מכאן נגזר הכלל: מסמך אחד לשני קוראים. ה-agents קוראים את הבלוקים, ולכן הם חייבים להיות מדויקים. המומחים קוראים את הטקסט שמסביב וחותמים עליו. "We keep one copy. The second one we drift.". זה עובד גם בכיוון ההפוך: גם הקוד צריך תיאור בשפה פשוטה בשביל המומחים. ערוץ המשוב העיקרי הוא מסך שבו כל מומחה סוקר משימה בדיוק כפי שה-agent ראה אותה. ההערה שלו הופכת לשינוי הבא של ה-coding agent, והשינוי נבדק על אותה משימה. לפי השקף, ההערה נסגרת רק עם ראיה מה-deploy. השורה התחתונה של הפרק: אף אדם יחיד לא יודע מה נכון. קבוצת מומחים מחליטה, ולכן מכניסים אותם לתהליך מההתחלה.

4. הנתיבים שחוזרים: איפה עדיין צריך שיקול דעת



What varies stays in prose, inside a frame developers own: שלב Prepare מכין את העובדות (amount 1,250, limit 1,000). אחריו decision table של המומחים: over the limit → team lead decides, within the limit → confirm, no row → a person. ובסוף execute.

מומחים מתקנים משימה אחת בכל פעם, וה-agents מריצים את אותו prose כל היום. לכן עוקבים אחרי כל תוצאה, אם התקבלה ואם נדחתה, והדיוק מראה איפה אנשים עדיין צריכים להסתכל. דוגמה לבאג שקט: ב-prose התקרה היא 1500 ובקוד היא אלף. אותה הזמנה תקינה במקום אחד וגבוהה מדי במקום אחר, "and nothing throws an error". ועוד תופעה: "even a line like do not call x teaches the model that X exists.". ל-prose אין compiler, אז הם בנו אחד: קודם הופכים את ה-prose לניתן לפירוק (typed blocks, templates, decision tables), ואחר כך מריצים lint על כל קובץ שהשתנה, כמו בקוד.

lint לא מונע מעותקים להתרחק זה מזה, ולכן עוקבים אחרי walks, כלומר המסלול ש-agent עובר בתוך ה-prose. walk שחוזר על עצמו הופך ל-workflow tool אחד, וה-prose שתיאר אותו נמחק. הניסיון הראשון החמיר את המצב: ה-prose עדיין הזכיר את השלבים שבתוך הכלי, וה-agents הריצו אותם שוב. זה עבד רק כשב-prose נשארו ההחלטות בלבד. הכלל היום: walk הופך לקוד רק כשהוא חוזר וגם מתנהג אותו דבר בכל המדינות. הקוד מכין את העובדות ומבצע את ההחלטה. "Experts own the decision table and the model only picks a row.". הגבול זז לשני הכיוונים: הם העבירו "hundreds of lines of expressions" מהקוד ל-prose, כי המומחים לא הצליחו לקרוא אותם. regex עדיין מתאים לזיהוי אות קריטי, וכשיש הרבה כאלה הם עוברים ל-config.

5. החלטה הופכת לפעולה: מה המודל רואה ומה מותר לו לשנות



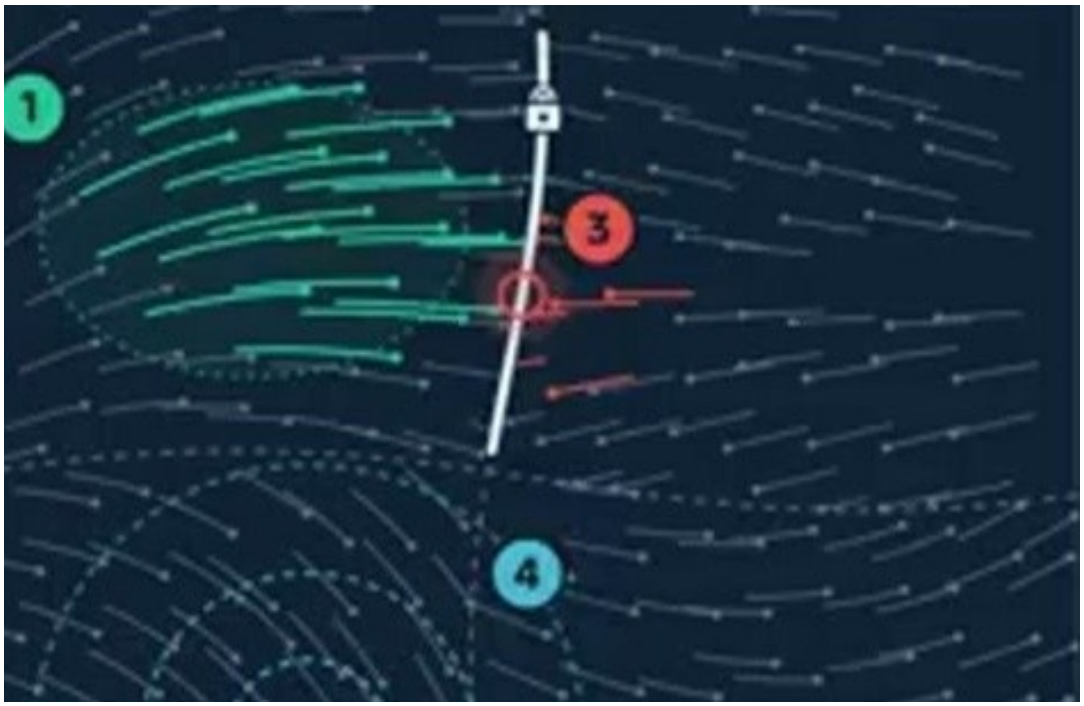
Six controls before any write lands: route policy (האם ה-slice הזה רשאי לפעול בכלל), instruction allowlist, write gating, consent של אדם, journal שמבטיח שהפעולה רצה ונרשמת פעם אחת, ו-readback שבודק שהכתיבה אכן נחתה.

בעבר Buykin חשב ששליטה פירושה הרשאות ל-agent. היום, לדבריו, השליטה מתחילה במה שהמודל רואה (Mind). כשנתנו למודל לבחור מה לטעון, הוא טען הכול, ואז המציא בדיקת תמחור רק כי מחירים הופיעו בהקשר. עכשיו ה-runtime מחליט מה המודל רואה: בדוגמה של ההזמנה זה שלוש עובדות, והרשומה המלאה נשארת אצל ה-runtime. כשמדובר בסט נתונים גדול, המודל מעביר handle, שם קצר לנתונים שה-runtime כבר מחזיק, וה-runtime מאמת אותו לפני כל כתיבה. כך ההקשר נשאר קטן אבל מספיק להחלטה, וה-runtime גם בודק שהוא עקבי.

אבל גם מודל שיש לו העובדות הנכונות יכול לגרום נזק (Power). לכן כל שינוי נבדק פעמיים: בזמן הפיתוח ושוב ב-runtime, ורק קוד פותח את מסלול הכתיבה. לפני כל כתיבה עומדות חמש חומות, והשישית בודקת שהכתיבה נחתה. אם כתיבה נראית כאילו נכשלה, ה-agent מנסה שוב, וה-journal מוודא שהיא תרוץ פעם אחת בלבד.

> — **Dmitry Buykin** Whatever sits in its context becomes a reason to act.

6. מעבר לקשב של אדם אחד

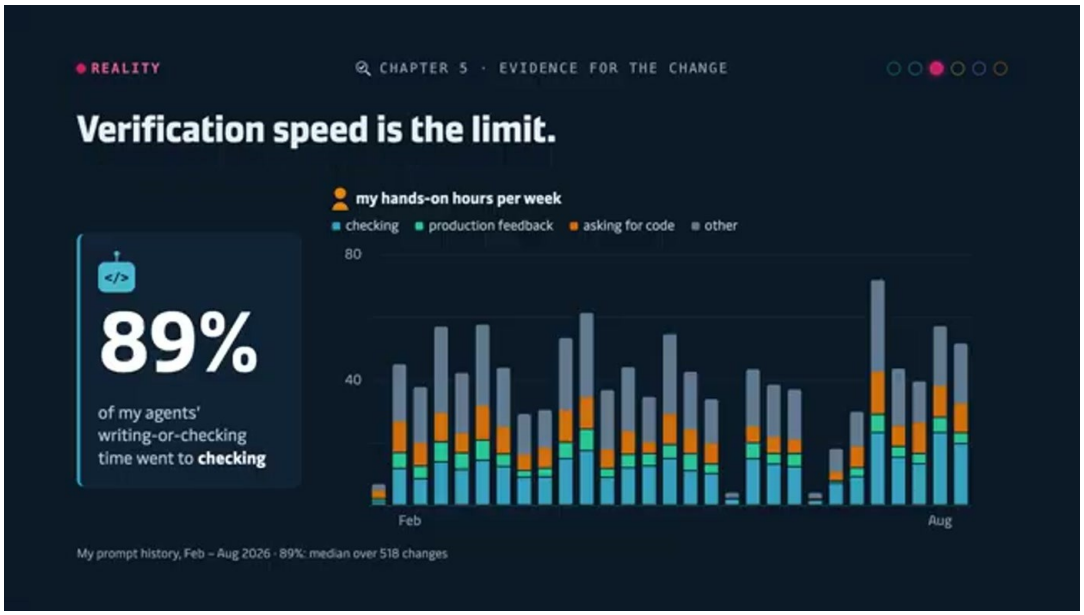


When changes outran my attention, I watched the derivatives: מפת זרימה עם מערבולות לצד ארבעה מכשירים, rate per domain, boundary leaks, contract diffs ו-blast radius (מיושם כ-query). בתחתית: Friction is the signal.

"All those controls protect the system. Nothing protected my attention". בהתחלה Buykin קרא כל שינוי, וכך המערכת אימצה את ההרגלים שלו. כשהנפח עבר את מה שאדם אחד מסוגל לקרוא, הוא נשחק והפסיק. כל tech lead מכיר את המצב שבו כל שינוי מחכה לו, ולדבריו agents פשוט מביאים אותך לשם בתוך שבועות במקום שנים. המדד הראשון היה הוא עצמו: אם הוא עייף ולא מסוגל להסביר מה השתנה, זה האות לעצור, גם כש-"The dashboard can be green". במקום לקרוא כל diff הוא בנה מכשירים שעוקבים אחרי קצב השינוי. לפני כל בדיקה חדשה הם הריצו את הכלל שלה על כמה אלפי sessions מהעבר: "It would have fired on less than 1%."

אות בודד ממתינים לו עד שהוא חוזר, גם כשהוא מגיע ממומחה. אשכול של אותות בכמה slices שונים הוא סיפור אחר, ואז מתקנים פעם אחת את המושג שמאחוריו. "Left alone, an agent keeps adding additions until today's tasks pass." (ייתכן שב-ASR המילה conditions הפכה ל-additions; השקף מראה גידול מ-3 ל-17 תנאים). כש-slice אחד נכשל, ההערה של המומחה הופכת ל-eval, ו-agent סורק את ה-siblings, כלומר הווריאנטים של אותו תהליך במדינות אחרות, כדי למצוא את אותה תקלה. התיקון מוחל רק במקומות שבהם התקלה אומתה. "At any time 10 to 20 of these loops are open". כל לולאה כותבת ל-ledger, כך ששום דבר לא תלוי בזיכרון שלו. ה-agents שוכחים הכול בסוף ה-session, ולכן הזיכרון נשמר בקבצים, בשני סוגים: מה קרה, ומה נכון עכשיו. ה-session הבא מתחיל ממה שנכון עכשיו.

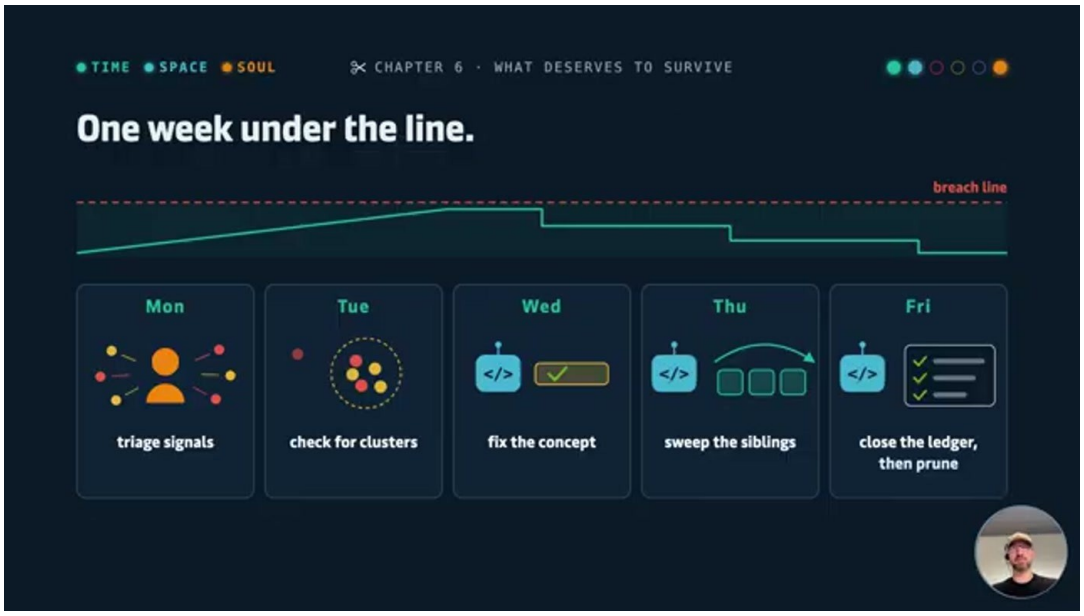
7. ראיות לשינוי, ומה ראוי לשרוד



Verification speed is the limit: בשקף כתוב 89% of my agents' writing-or-checking time went to checking, ולצדו גרף עמודות של שעות עבודה בשבוע לפי סוג העבודה (checking, production feedback, asking for code ועוד).

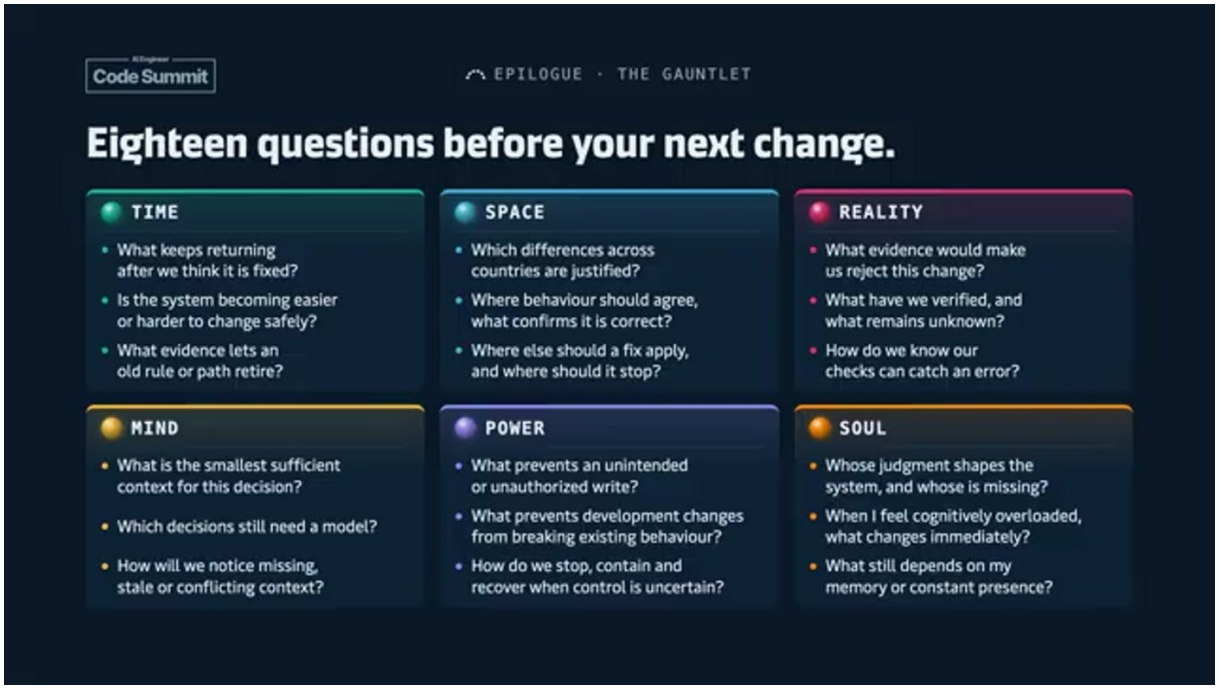
"I used to trust Green". ירוק פיתה אותו כי הבדיקה היא המקום שבו הולך הזמן. לכן הם מדרגים ראיות לפי מה שהן מוכיחות. בראש הסולם נמצא replay על משימות אמיתיות, ובשקף מופיעים אחריו production measurement, קריאת קוד ב-file:line, מה שאדם עשה בפועל, טסטים, legacy code ו-docs. שמות וניחושים נמצאים מתחת לסולם. לפני שהנתונים מגיעים כותבים מה יוכיח שהם טועים, וכל בדיקה חדשה חייבת להיכשל פעם אחת בכוונה לפני שסומכים עליה. "AI refactors fail quietly". אחרי refactor כל branch חייב לנחות בדיוק פעם אחת. בדוגמה של ההזמנה, ה-branch של סכום מעל התקרה אבד, וההזמנה הייתה עוברת בלי ה-team lead. לכן מקפיאים סט של משימות אמיתיות ומריצים אותו על הגרסה הישנה ועל החדשה. "If no real task takes a branch, we call that no evidence".

לבדיקה עצמה: "You can't trust one agent to judge its own work.". agent שני בודק לפי checklist ובלי ההנמקה של הכותב. coding agents גם נוטים לסיים את רוב ה-batch ולדלג בשקט על השאר, במיוחד כשההקשר מלא, וה-ledger חושף את זה. בשקף: 84% מהפריטים מסומנים כמכוסים, אבל רק 27% אפשר לסגור. ובפרק האחרון, על מה ששורד: טסטים השאירו בחיים וריאנטים מתים, כי ה-agents שמרו כל אפשרות שעוד הוזכרה באיזה טסט. היום "a retired variant takes its test with it". ratchet פירושו שהמונה יכול רק לרדת. כל מה שאף משימה אמיתית לא מגיעה אליו מקבל שם וראיה לפני שהוא נמחק.



One week under the line: ביום שני triage signals, בשלישי check for clusters, ברביעי fix the concept, בחמישי sweep the siblings, ובשישי close the ledger ואז prune.

8. מה עושים מחר



Eighteen questions before your next change: שלוש שאלות לכל אחת מהאבנים Time, Space, Reality, Mind, Power ו-Soul. למשל: What keeps returning after we think it is fixed?, What prevents an unintended or unauthorized write?, What still depends on my memory or constant presence?

- **לבחור שאלה אחת מתוך ה-18:** "pick one your own repository can't answer yet. That's where to start".
- **למדוד כמה מה-prose מכוסה בטסטים:** לשבור משפטים בהוראות ל-agent (mutation testing) ולבדוק אם טסט כלשהו נכשל.
- **להקטין את ההקשר:** שה-runtime יעביר למודל רק את העובדות שההחלטה צריכה, ו-handles במקום סטים גדולים.
- **לחסום כתיבה בקוד:** allowlist, write gating, consent, journal שמבטיח ריצה אחת ו-readback.
- **להריץ replay למשימות אמיתיות לפני ואחרי כל refactor של AI:** ולוודא שכל branch נוחת בדיוק פעם אחת.

- **סוקר נפרד:** agent שני עם checklist, בלי ההנמקה של הכותב, ו-ledger שמראה על מה דילגו.
- **מקצב שבועי ומחיקה עם ראיות:** וריאנט שיוצא משימוש לוקח איתו את הטסטים שלו.
- **להתחיל לפני השחיקה:** "Use your stones while you can still read every change."

שאלה פתוחה	למה היא פתוחה
איך נראה ה-compiler וה-linter ל-prose בפועל?	תוארו רק עקרונות (typed blocks, templates, decision tables). לא הוצג מימוש ולא כלי זמין.
האם השיטה מורידה תקלות בייצור?	לא הוצגו מדדים של לפני ואחרי, רק ההתרשמות שלא נדרש snap נוסף.
מה עושים ששת ה-gauges המתוכננים?	"Two gauges run today and six more are next.", בלי פירוט.
מתי המודל יקבל החלטות קטנות בעצמו?	השלב הזה עוד לא נבנה.
איך מתמודדים עם ההטיות של מי שמתקן את המערכת?	"ours copies whoever corrects it": הבעיה הוצגה, אבל לא נאמר איך מאזנים אותה.

9. מילון מונחים

מונח	משמעות
Breach line	הסף שמעליו יש יותר שינויים ממה שאפשר לבדוק. המטרה היא להישאר מתחתיו.
Snap	מחיקה ושכתוב גורפים של קוד, טסטים ו-boilerplate.
Prose	הוראות בשפה טבעית ל-agents. לפי ההערכה, כשלושה רבעים מהמערכת.
Slice	תהליך אחד במדינה אחת. שינוי נשאר בתוכו כברירת מחדל.
Mutation testing	שבירה מכוונת של משפט או קוד כדי לבדוק אם הטסטים יתפסו את זה.
Walk	המסלול ש-agent עובר בתוך ה-prose.
Decision table	טבלת החלטות שבבעלות המומחים. המודל רק בוחר שורה בה.
Handle	שם קצר לנתונים שה-runtime מחזיק, במקום להכניס אותם להקשר.
Write gating / Journal / Readback	חסימת כתיבה כשאין הרשאה, הבטחה שהפעולה תרוץ פעם אחת, ובדיקה שהכתיבה נחתה.
Blast radius	היקף ההשפעה של שינוי. נמדד באמצעות query.
Siblings	וריאנטים של אותו תהליך בשווקים ובמדינות אחרות.
Ledger	רישום מתמשך של לולאות, פריטים וראיות, כדי שדבר לא יהיה תלוי בזיכרון של אדם.
Replay	הרצה חוזרת של משימות אמיתיות על הגרסה הישנה ועל החדשה.
Ratchet	מדד שמותר לו רק לרדת.
Eval	בדיקת איכות אוטומטית שנבנתה מהערה של מומחה.