

Loop Engineering מעקרונות יסוד: לולאות agent שעובדות בעולם האמיתי

למה Ralph loop עיוור מייצר PR-ים שאף אחד לא קורא, ואיך control theory מאפשרת
לבנות לולאות קטנות, מדידות וניתנות להכוונה

Kyle Mistele (HumanLayer) · AI Engineer World's Fair
מחדש ב-X על ידי Codez · 2026-07-05

משך: כ-17 דקות · X

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה של כ-17 דקות בשם Loop Engineering from First Principles, שהעביר Kyle Mistele מ-HumanLayer. השם וה-handle @0xblacklight מופיעים בשורת התחתית של השקפים. לפי הלוגו בשקפים, ההרצאה ניתנה ב-AI Engineer World's Fair. הסרטון הגיע אלינו מ-repost ב-X, והכיתוב שם (Senior AI engineer) לא בא מההרצאה עצמה.

הטענה המרכזית: התעשייה התאהבה בלולאות שמזרימות prompt ל-coding agent שוב ושוב (Ralph loop). הלולאות האלה מייצרות PR-ים עצומים שאיש לא קורא, וזה לא מתאים לצוותים שעובדים על מערכות קריטיות. החלופה היא agent control loop: מגדירים set point, מודדים בעזרת sensor, ו-controller בוחר שינוי קטן ש-actuator מבצע. אחר כך מודדים שוב. Mistele מדגים את זה על הגירה הדרגתית של RPC API ל-Effect ב-HumanLayer, כולל ההרצאה ב-GitHub Actions, קובץ feedback שאפשר להכווין דרכו, ו-flow control שמונע הצטברות של PR-ים.

איך לקרוא את המסמך

- 1. **תקציר מנהלים:** הטענות המרכזיות והראיות, עם חותמות זמן.
- 2. **הבעיה:** Ralph loops, השיח על write-only code, ולמה קוד גרוע יקר מתמיד.
- 3. **Control theory:** המודל (sensor, set point, controller, actuator) ואיך מתרגמים אותו ל-agents.
- 4. **Case study, חלק א':** הגירה ל-sensor, Effect מבוסס ast-grep ו-disturbance dampener ב-CI.
- 5. **Case study, חלק ב':** ה-controller, ה-actuator ו-golden patterns.
- 6. **הרצה בפרודקשן:** feedback file, GitHub Actions, עם flow control, /iterate, והאצה.
- 7. **מה לעשות עכשיו:** צעדים מעשיים וטבלת שאלות פתוחות.
- 8. **מילון מונחים:** המונחים באנגלית שמופיעים במסמך.

הערות מקור המסמך נוצר אוטומטית מהתמליל של ההקלטה ומהשקפים שחולצו ממנה. התמלול האוטומטי (ASR) עלול לשבש שמות ומונחים. למשל Cloud Code הוא Claude Code, והשקף מציג Boris Cherny ולא Boris Turney. איפה שלא היה ודאי, ציינו זאת במפורש. הציטוטים מועתקים מילה במילה מהתמליל ונבדקו מול חותמת הזמן שלהם. לא הוספנו מידע ממקורות חיצוניים.

1. תקציר מנהלים

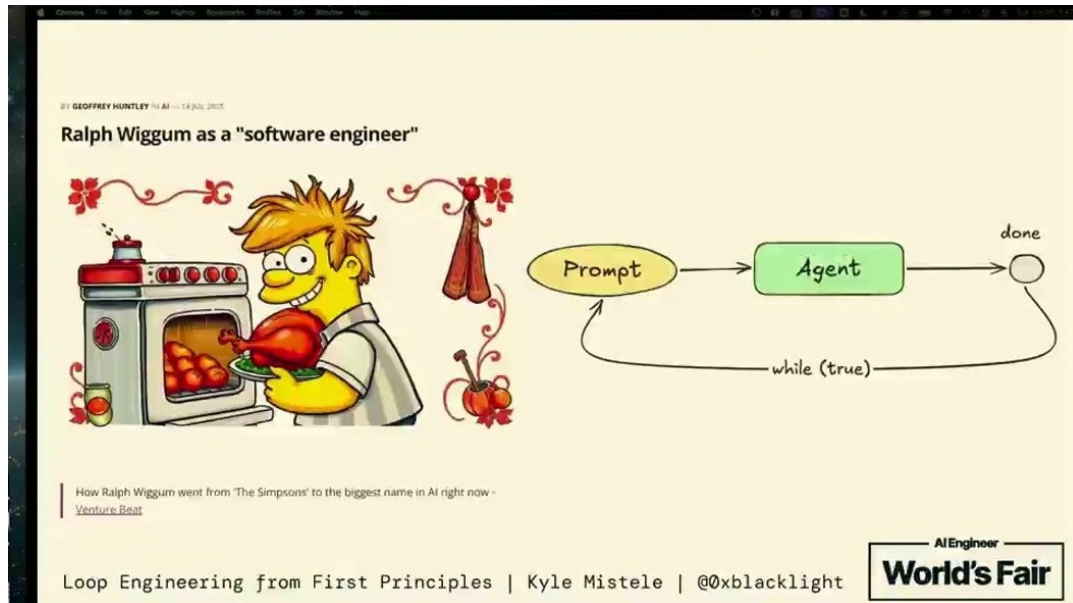
Mistele לא מתנגד ללולאות. HumanLayer עצמה בונה אותן. הוא טוען שלולאה טובה צריכה להתנהג כמו control loop: לשנות את המערכת בצעדים קטנים ומדידים, לקבל feedback ולתת לבני אדם להמשיך לקרוא את הקוד. רוב ההרצאה היא מתכון מעשי שמבוסס על הגירה אמיתית שהצוות מריץ.

- **לולאה עיוורת מייצרת PR-ים שאיש לא יקרא:** גם עם verifiers ושישה code review agents, התוצאה היא עדיין "40,000 [0:00] line PRs that just nobody wants to read". לדבריו, Ralph מתאים למי שלא עובד בצוות ולא עובד על מערכות קריטיות.
- **השיח על write-only code עוד לא הוכיח את עצמו:** הוא מזכיר שתיקון ה-terminal flicker של Claude Code לקח שישה חודשים ושם-OpenClaw יש בעיות יציבות. לעומת זאת, לדבריו, העלות ברורה מאוד למי שאין לו תקציב tokens בלתי מוגבל [3:01].
- **Control theory היא המסגרת הנכונה:** Set point, sensor, measured error, controller, actuator ומדידה חוזרת. אותו דפוס מופיע ב-thermostat, ב-Kubernetes autoscaling, ב-laC, ב-autovacuum של Postgres וב-virtual DOM של React [4:31].
- **קודם עוצרים את הדימום, אחר כך מגירים:** סורקים את main פעם אחת עם ast-grep, שומרים רשימת הפרות ממוינת ב-version control ובודקים שכל PR חדש לא מוסיף הפרות. הוא קורא לזה disturbance dampener [9:06].
- **אין לתת ל-agent לעשות עבודה של קוד דטרמיניסטי:** את ה-controller אפשר לבנות עם bash ו-jq, או עם ast-grep שבוחר את ה-procedure הקטנה ביותר כדי להקטין סיכון [10:38].
- **ה-skill הוא העיקר:** golden patterns שנכתבים ביד לפני שמשחררים את ה-agent, כי agents הם "pattern replicators" [12:10].
- **מספיק CI קיים:** workflow ב-GitHub Actions מריץ איטרציה אחת (sense, control, actuate) פעם ביום ופותח PR קטן. לא צריך cluster חדש [12:10].
- **הכוונה זולה דרך קובץ:** קובץ feedback ב-markdown שמנוהל ב-version control, ותגובת /iterate על ה-PR שמעדכנת גם את הקוד וגם את הקובץ [13:42].
- **Flow control מונע עבודה כפולה:** אם יש PR פתוח עם התווית של הלולאה, ה-workflow נעצר. כך יש לכל היותר PR פתוח אחד לכל לולאה [15:14].

מה לא נאמר

- **אין מדדי הצלחה:** לא נאמר כמה procedures כבר הוגרו, כמה PR-ים נדחו, ומה העלות בכסף או ב-tokens לכל ריצה.
- **לא צוין coding agent:** "Bring your CLI coding agent of choice". לא נאמר באיזה agent או מודל HumanLayer משתמשת.
- **Controller מבוסס telemetry הוצג כאפשרות בלבד:** בחירת procedure לפי שגיאות או פערים ב-APM הוצגה במילים "what we could do". לא נאמר שהצוות מפעיל את זה בפועל.
- **ה-skill שפורסם לא מזוהה:** הוא מזכיר skill שאפשר לנסות, אבל השם והקישור לא מופיעים בתמליל ולא בשקפים שחולצו.
- **הביקורת על Claude Code ו-OpenClaw אנקדוטלית:** היא נשענת על דוגמאות בודדות ולא על השוואה שיטתית.

2. הבעיה: Ralph loops וקוד שאיש לא קורא



Ralph Wiggum as a "software engineer": תרשים של prompt שמזרם ל-agent בתוך לולאת while (true)

Mistele פותח בלולאת Ralph, שלדבריו פורסמה ביולי והפכה ויראלית בינואר. בתמליל השם של המחבר מופיע Jeff Huntley, וכל הנראה מדובר ב-Geoffrey Huntley, אבל לא אימתנו זאת. הוא מבהיר שזו לא התקפה: Ralph הוא "[0:00] a sharp tool that works very well for certain types of problems". אבל רוב המהנדסים עובדים בצוות, על מערכות עם לקוחות, רגולציה ו-SLA, ולא יכולים לשלוח PR של 40 אלף שורות ישר לפרודקשן.



ציוץ של Peter Steinberger על לולאות codex שרצות על כל commit, מוצאות באגים ופותחות PR עם תיקון

משם הוא עובר לשיח הנוכחי. Peter Steinberger אומר שלא צריך יותר לכתוב prompts ל-agents, רק לתכנן לולאות שעושות את זה. OpenClaw (בתמליל Open Clause) בנוי כולו על לולאות, "It's loops all the way down". [1:30] בשקף הבא מופיע ציטוט של Boris Cherny, ה-creator של Claude Code, שלפיו התפקיד שלו היום הוא לכתוב לולאות.



שקף ציטוט: (Boris" I don't prompt Claude anymore. I write loops — and the loops do the work. My job is to write loops." (Boris Cherny, Creator of Claude Code)



DevGuild: Write-Only Code Summit. הכנס שמקדם את הרעיון שקוד שנוצר בלולאות לא צריך לקרוא

השלב הבא בשיח הוא write-only code: יש כל כך הרבה קוד שאי אפשר לקרוא אותו, אז לא קוראים בכלל. זו התזה של הכנס בשקף ("[1:30] a conference that was here in town last month"). Mistele מטיל ספק: "it's not clear how well this works yet" [1:30]. לדבריו, תיקון ה-flicker בטרמינל של Claude Code לקח שישה חודשים, וצוות OpenCode כתב renderer בחלק קטן מהזמן הזה. הוא מוסיף שמה שבטוח הוא שזה יקר למי שלא עובד במעבדת frontier.

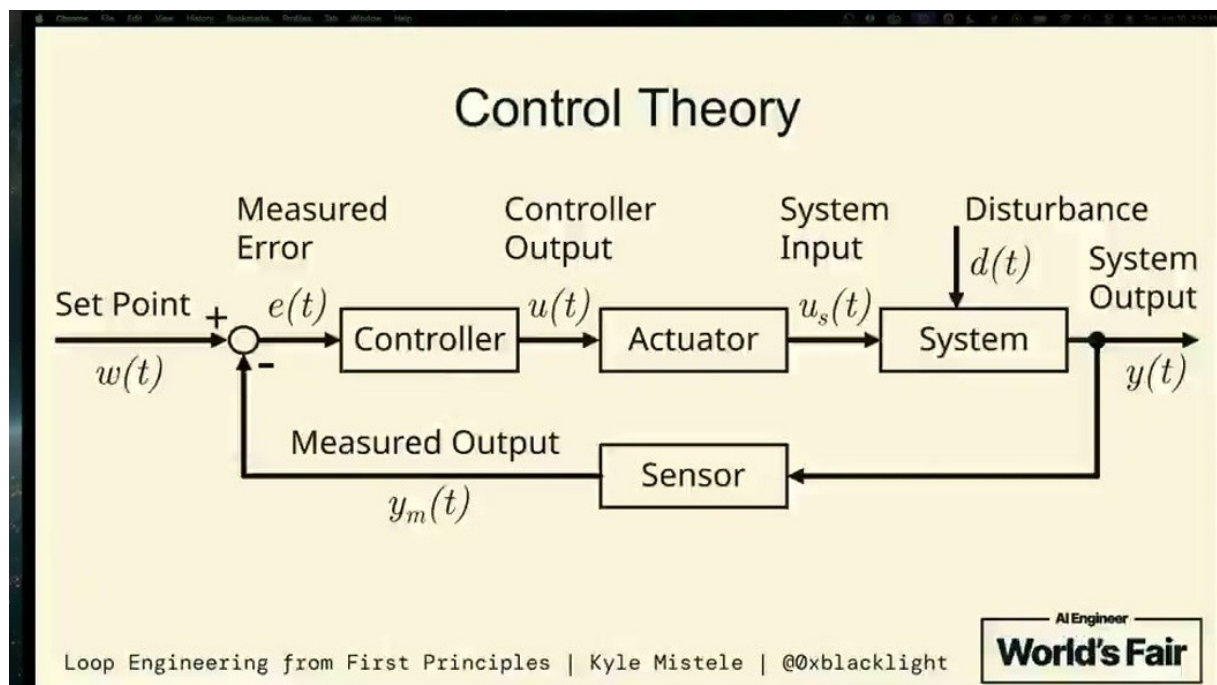


שקף: "Bad code is the most expensive it's ever been", לצד צילום של הדובר (מיוחס ל-Matt Pocock לפי התמליל)

Matt Pocock [3:01] Bad code is much more expensive in the age of agents **Kyle Mistele — בשם** .than it has ever been at any point in the past

התזה "we can design loops and still read the code. In fact, we can design loops that make it easier to" read the code because the loops are making the code better." [3:01]

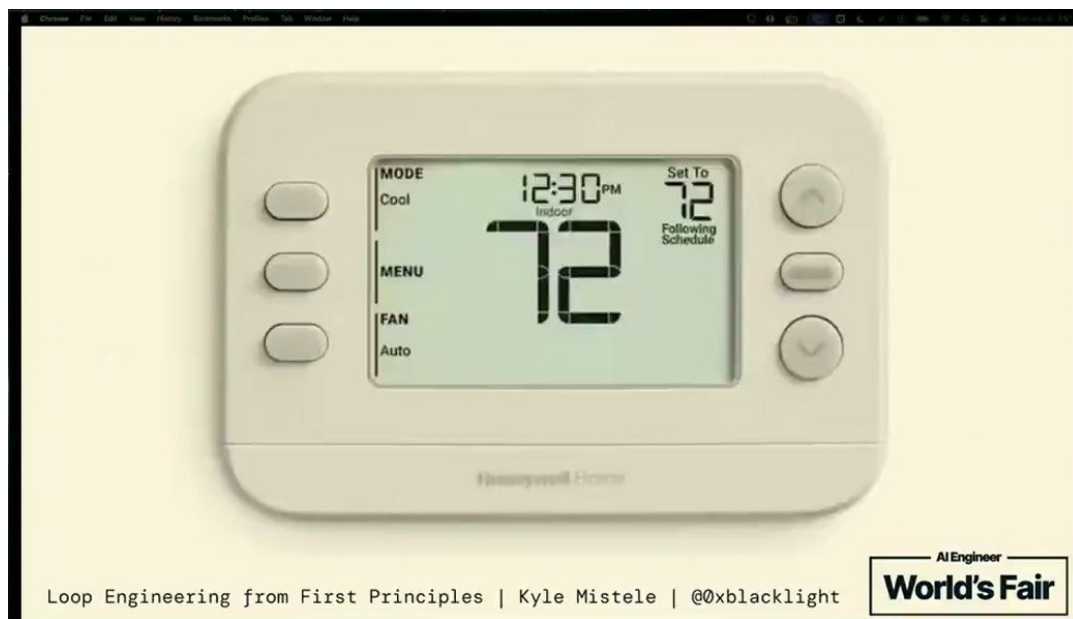
3. Control theory כמסגרת ללולאות agent



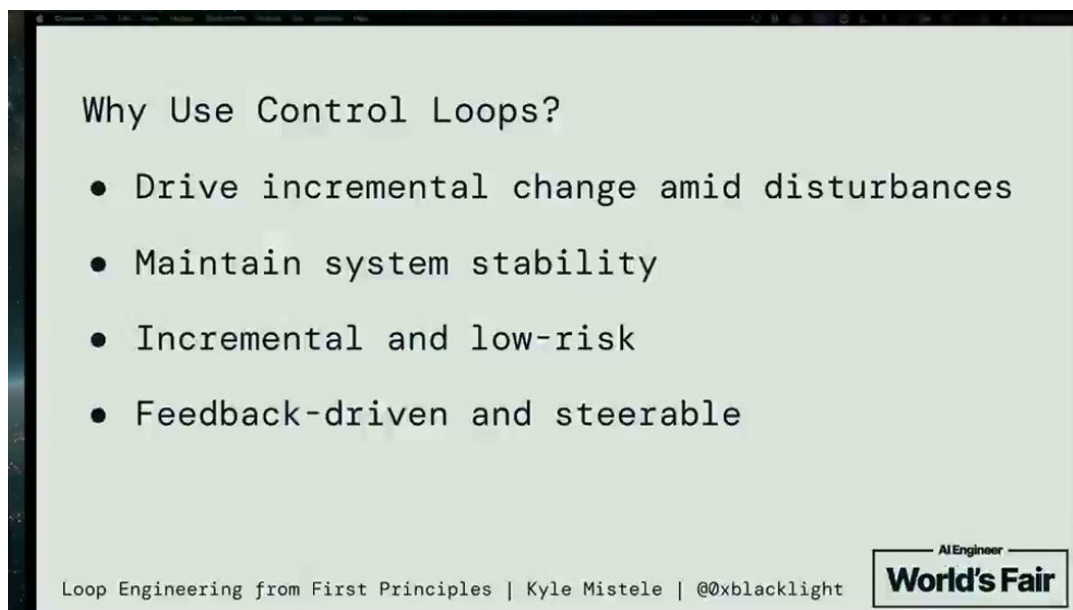
תרשים בלוקים של Set Point, Measured Error, Controller, Actuator, System עם Disturbance, ו-Sensor שמחזיר Measured Output

Control theory עוסקת בהובלה של מערכת דינמית, ובמקרה הזה codebase, למצב יציב או אופטימלי שרוצים להגיע אליו. ה-sensor מודד את המצב הנוכחי. ה-set point הוא המצב הרצוי, וההפרש ביניהם הוא ה-measured error. ה-controller הופך את השגיאה ל-control signal, כלומר לשינוי קטן. ה-actuator מבצע את השינוי על

המערכת, שבינתיים סופגת disturbances, ואז מודדים שוב. Mistele מספר שאחיו התאום הוא מהנדס אווירונאוטיקה ושכר שומרים על מטוסי קרב באוויר, אבל מוסיף שזה פשוט יותר ממה שנדמה [4:31].



Thermostat ביתי (72 מעלות), הדוגמה היומיומית ל-control loop



Why Use Control Loops?: שינוי הדרגתי למרות disturbances, יציבות המערכת, צעדים קטנים בסיכון נמוך, והכוונה מבוססת feedback

הוא מונה דוגמאות מהיום-יום: Kubernetes autoscaling, infrastructure as code (מצב רצוי, מצב נוכחי, שינוי איטרטיבי), Postgres של autovacuum וה-virtual DOM של React. לולאת בקרה מתאימה כשיש מערכת שרוצים לשנות, בעיה שאפשר למדוד ודרך לקבל feedback על השינוי. היא משנה את המערכת בהדרגה ולא "risk blowing everything up" [4:31].

מכאן נגזרת ההבחנה המרכזית: "control loops are the opposite of what I'm gonna call a blind Ralph loop". לדבריו, ה-Ralph-ים הטובים כבר מיישמים control theory, ו-Ralph נועד מלכתחילה ככלי הוראה שחלק מהאנשים קראו מילולית מדי. הבעיה השנייה היא ש-Ralph הוא "just a bash loop" ולכן לא הדרגתי [6:03].



Hybrid Sensor + Controller: הפלט של react-doctor של Aiden Bai, שמאתר בעיות בקוד React וגם ממליץ מה לתקן קודם

בעולם של agents הגבולות בין הרכיבים מטשטשים. sensor יכול להיות דטרמיניסטי (חוקי grep, AST, ESLint) או לא-דטרמיניסטי (agent עם skill וחוקים בשפה טבעית), או pipeline שמשלב את שניהם. react-doctor (בתמליל AidenBuy; בשקף Aiden Bai) הוא sensor ו-controller בבת אחת: הוא מראה את כל הבעיות בקוד ה-React ואומר אילו שלוש לתקן קודם. ה-controller וה-actuator יכולים להיות agent יחיד שמחליט ומבצע באותו context window. Mistele מדגיש את ה-controller במיוחד: בלי controller מכויל היטב "we might make too large of a change all at once, or we might make the wrong change entirely" [7:34].

שימושים נוספים שהוא מציע: תאימות ל-OpenAPI spec של מישור אחר, תאימות של שרת MCP לגרסה העדכנית של ה-spec, mirroring בין Python ל-TypeScript, ותחזוקה של fork מול upstream. לפני שמתחילים יש לשאול שלוש שאלות: האם יש משהו מדיד, האם אפשר לשנות בצעדים קטנים, והאם אפשר לקבל feedback על איכות השינוי [7:34].

4. Case study, חלק א': הגירה ל-Effect, sensor-disturbance dampener ו-

הדוגמה היא לולאה פנימית ב-HumanLayer שמגירה בהדרגה את ה-RPC API ל-Effect. הצוות אימץ את Effect קודם בקוד שנוטה ל-race conditions ועכשיו מרחיב אותו לשאר ה-codebase. "The syntax is really weird. We're psychos, we really like it" [7:34].

את ה-sensor הם בנו עם ast-grep ולא עם grep או ast-grep. agent לא תלוי בשפה, עובד בנפרד מה-TypeScript config ומחוקי ESLint, ומתאים ל-monorepo רב-לשוני. הנימוק לכך שהוא נפרד מהם: "if you're a TypeScript developer you have watched Claude disable those with inline comments" [9:06]. חוק פשוט מאתר procedures שעוד לא הוגרו. הפלט כולל כ-50 שדות לכל הפרה, ולכן מצמצמים לארבעה שדות וממיינים בצורה דטרמיניסטית.

לפני שמתחילים להגיר, צריך לעצור את הדימום. מריצים סריקה מלאה אחת על main, שומרים את רשימת ההפרות הממוינת ב-version control, ובכל PR חדש בודקים אם ה-branch הוסיף procedure שלא הוגרה. ה-disturbance במקרה הזה הוא חברי הצוות, שממשיכים להכניס קוד חדש שנוצר עם Claude ("Claude slop"). Mistele מודה שזה לא רכיב קלאסי של control loop, "we can kind of like squint at it a little bit and call it a disturbance dampener" [10:38].

5. Case study, חלק ב': golden-controller, actuator patterns

ה-controller הפשוט ביותר בוחר את ההפרה הראשונה ברשימה בעזרת `bash` ו-`jq`. גרסה חכמה יותר משתמשת ב-`ast-grep` כדי לבחור את ה-procedure הקטנה ביותר, כי שינוי קטן מקטין סיכון. אפשר גם לתת ל-agent להחליט, אבל:

— **Kyle Mistele [10:38]** I don't think you should ever send an agent to do deterministic code's job, but you certainly can.

הגרסה המתקדמת שהוא מתאר מסתמכת על `telemetry`. בוחרים את ה-procedures עם הכי הרבה שגיאות, הכי מעט instrumentation או פערים ב-APM, ומעבירים ל-actuator גם את הנתונים האלה. כך ה-agent משפר את הקוד ולא רק מתרגם אחד לאחד. זה הוצג כאפשרות ("what we could do"), לא כמשהו שכבר פועל.

ה-actuator הוא "[10:38] just an agent plus a skill". עם כל `CLI coding agent`. את רוב הזמן כדאי להשקיע ב-skill, ולא חייבים לעשות את כל העבודה מראש: משפרים אותו לאורך הזמן לפי מה שעובד. ב-HumanLayer כותבים ביד golden patterns, דוגמאות אידיומטיות לפני שמשחררים את ה-agent, כי "agents are just pattern replicators". בלי דוגמאות כאלה מקבלים את מה שכתוב בתיעוד או את מה שה-agent למד מהאינטרנט [12:10]. ה-skill כולל response template. בסוף הריצה ה-commit, ה-push ופתיחת ה-PR נעשים בצורה דטרמיניסטית, והתשובה האחרונה של ה-agent משמשת כתיאור ה-PR.

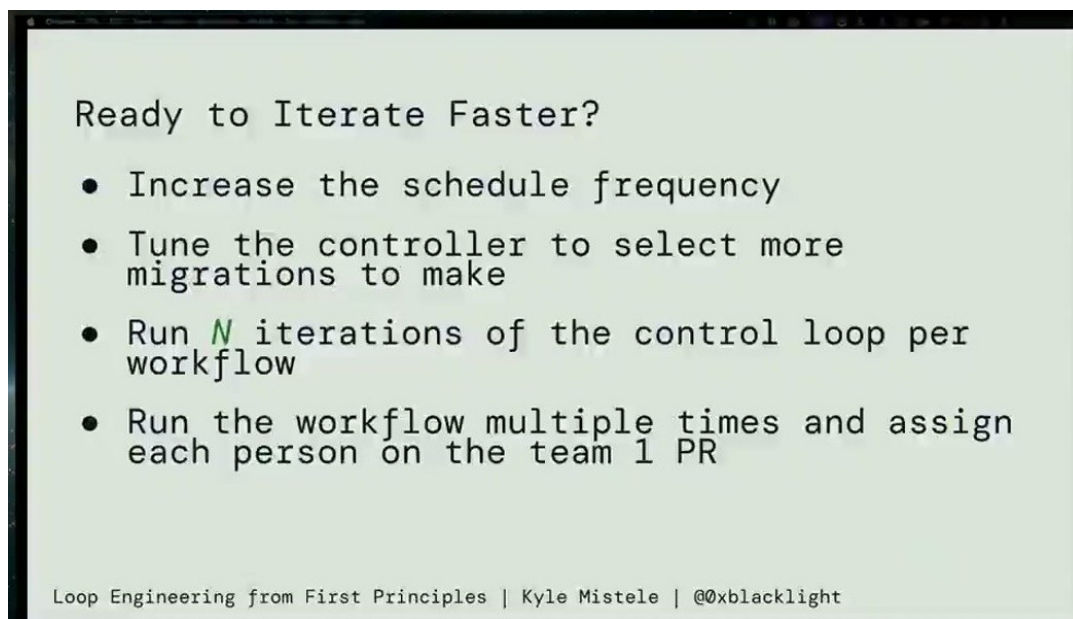
6. הרצה בפרודקשן: GitHub Actions, feedback file ו-flow control

את הלולאה מריצים ב-CI שכבר קיים: GitHub Actions, GitLab או CircleCI. ל-CI כבר יש גישה לקוד ולסודות, וגם יכולות dispatch ו-scheduling, ולכן "[12:10] We don't need a new cluster for this". ה-workflow מריץ איטרציה אחת (sense, control, actuate), פותח PR ומתוזמן לפעם ביום. כל בוקר מחכה PR קטן בסיכון נמוך.

מה השתבש: בפעם הראשונה הם כיבו את הלולאה. כל תיקון דרש לעשות checkout ל-branch, לשנות את ה-skill ואת הקוד, ואז commit ו-push. זה היה "[12:10] really high friction".

- **Feedback file:** קובץ markdown שמנוהל ב-version control ונטען בצורה דטרמיניסטית ל-context של ה-actuator בכל ריצה, אחרי ה-[13:42] controller.
- **תווית לכל לולאה:** כל workflow מסמן את ה-PR-ים שלו, כך שכשרצות כמה לולאות כל אחת מגיבה רק לתגובות על ה-PR-ים שלה.
- **/iterate:** תגובה על ה-PR מפעילה את ה-workflow, שטוען את ה-diff, התגובות, ה-review comments והתיאור, ומנחה את ה-agent לתקן את הקוד וגם לעדכן את קובץ ה-feedback. כך רואים את ההיסטוריה של ההנחיות ואפשר לעשות להן revert.
- **Flow control:** לפני checkout והתקנת dependencies בודקים אם יש PR פתוח עם התווית של הלולאה. אם יש, עוצרים. הבעיה המקורית: כשהצוות היה אצל לקוח או בנסיעה, ה-PR-ים הצטברו, שכפלו עבודה ונכנסו לקונפליקטים [13:42].

— **Kyle Mistele [15:14]** No human reviewed the last output, so there's no reason to stack up even more work for humans to review. This way we have exactly one PR at most open per loop at a time



Ready to Iterate Faster?: להגדיל את תדירות התזמון, לכוון את ה-controller לבחור יותר הגירות, להריץ N איטרציות לכל workflow, ולהריץ את ה-workflow כמה פעמים עם PR לכל אחד בצוות

האצה: "I have 150 RPC procedures to migrate. If I do one at a time, it's gonna take six months" [15:14]. אפשר לתת ל-controller לבחור שלוש או חמש procedures בכל פעם. אפשר לבחור כמה ולבצע כל אחת ב-implementation phase נפרד עם context window משלה, מה שלדבריו זול ואמין יותר. ואפשר להריץ את ה-workflow ארבע פעמים ולתת PR לכל אחד מארבעה אנשים בצוות. לסיכום הוא מציג לולאה שמשפרת את הקוד בהדרגה, שהצוות עדיין קורא את הקוד שלה, שיש לה flow control אדפטיבי ושקל להכווין אותה תוך כדי ריצה [15:14].

7. מה לעשות עכשיו

- **בחרו property מדיד אחד:** הגירה, תאימות ל-spec או ניקוי של anti-pattern. ודאו שאפשר למדוד אותו, לשנות אותו בצעדים קטנים ולקבל feedback.
- **בנו sensor דטרמיניסטי מחוץ ל-linter:** חוק grep ast-שה-agent לא יכול לבטל בהערה inline, עם פלט מצומצם וממוין.
- **עצרו את הדימום לפני שמגירים:** baseline של הפרות ב-version control ובדיקה ב-CI שכל PR לא מוסיף הפרות.
- **Controller דטרמיניסטי ושמרני:** בחירה של השינוי הקטן ביותר, ולא של agent.
- **השקיעו ב-golden patterns skill:** שונתבים ביד ו-response template שמשמש כתיאור ה-PR.
- **הריצו ב-CI הקיים פעם ביום:** עם תווית לכל לולאה, PR פתוח אחד לכל היותר, וקובץ feedback שמתעדכן דרך /iterate.
- **האיצו רק כשיש ביטחון:** יותר פריטים לכל ריצה, כל אחד ב-context נפרד, או חלוקה בין אנשי הצוות.

שאלה פתוחה	למה זה חשוב
כמה עולה ריצה של לולאה, ומה שיעור ה-PR-ים שמתקבלים בלי תיקון?	בלי המספרים האלה קשה להצדיק את ההשקעה מול הגירה ידנית או codemod.
האם ה-controller מבוסס ה-telemetry באמת פועל ב-HumanLayer?	זה הרעיון החזק בהרצאה, אבל הוא הוצג כאפשרות בלבד.
איך מונעים מקובץ ה-feedback להתנפח ולהכיל הנחיות סותרות?	הוא נטען לכל ריצה, ולכן הגודל והאיכות שלו משפיעים על התוצאות.
מה קורה כשהשינוי ההדרגתי חוצה כמה procedures שתלויות זו בזו?	בחירת "הקטנה ביותר" לא מתאימה לשינויים שצריכים להיכנס יחד.
מה שם ה-skill שפורסם ואיפה הוא נמצא?	הוא הוזכר בסוף ההרצאה, אבל השם והקישור לא מופיעים בתמליל.

8. מילון מונחים

פירוש	מונח
לולאת bash שמזרימה את אותו prompt ל-coding agent שוב ושוב עד שמגיעים לתוצאה	Ralph loop
לולאת בקרה: מדידה, השוואה למצב הרצוי, שינוי קטן ומדידה חוזרת	Control loop
המצב הרצוי של המערכת לגבי property מסוים	Set point
הרכיב שמודד את המצב הנוכחי, דטרמיניסטי (,ast-grep ESLint) או מבוסס agent	Sensor
ההפרש בין המצב הנמדד לבין ה-set point	Measured error
הרכיב שמתרגם את השגיאה לשינוי הקטן הבא (control signal)	Controller
הרכיב שמבצע את השינוי. כאן: coding agent עם skill	Actuator
שינויים חיצוניים שקורים במקביל, כמו קוד חדש של חברי הצוות	Disturbance
בדיקת CI שמונעת מ-PR-ים חדשים להוסיף הפרות כלי חיפוש והתאמת דפוסים על AST, לא תלוי בשפה	Disturbance dampener
ספריית TypeScript לטיפול בשגיאות ובאפקטים, שאליה HumanLayer מגירה את ה-RPC API	ast-grep
Effect	
דוגמאות קוד אידיומטיות שנכתבות ביד כדי שה-agent ישכפל אותן	Golden patterns
חבילת הנחיות ודוגמאות שנטענת ל-agent	Skill
קובץ markdown ב-version control עם הנחיות מצטברות ללולאה	Feedback file
מנגנון שעוצר את הלולאה כשיש PR פתוח שעוד לא עבר review	Flow control
הגישה שלפיה קוד שנוצר על ידי agents לא נקרא על ידי בני אדם	Write-only code
Application Performance Monitoring, ניטור ביצועים ושגיאות של האפליקציה	APM