להנדס את סוכני הקוד כמו כל מערכת אחרת

Zach Lloyd מ-Warp: המעבר מפרומפטינג מקומי ל-software factory שמוגדר כקוד, נמדד ומשפר את עצמו

Zach Lloyd · AI Engineer · 20 בנובמבר 2026
משך: כ-18 דקות · YouTube
סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה קצרה (כ-18 דקות) מכנס AI Engineer. מרצה בה Zach Lloyd, מייסד ו-CEO של Warp, שהוביל בעבר את ההנדסה של חבילת Google Docs. Warp מוכרת בעיקר בזכות ה-agentic terminal בקוד פתוח שלה, ולפי Lloyd יש לו כ-65,000 כוכבים ב-GitHub וכמעט מיליון מפתחים. היום החברה בונה פלטפורמה שבעזרתה ארגונים מקימים software factories פנימיים.

הטענה המרכזית: לעבוד עם coding agents כמו שרוב המפתחים עובדים היום, כלומר מקומית ובפרומפט אינטראקטיבי, זה מוגבל בתפוקה האנושית וקשה לשלוט בו מבחינת עלות, אבטחה ו-governance. לפי Lloyd, סוכנים הם primitive הנדסי חדש וצריך לבנות סביבם מערכת: מפעל שמוגדר כקוד, פועל בלולאה סגורה ונמדד לאורך כל הדרך. מכאן נגזרים כלי העבודה החדשים של factory engineering: evals, לולאות self-improvement, benchmarks ו-A/B testing.

איך לקרוא את המסמך

- **1. תקציר מנהלים:** הטענות המרכזיות, כל אחת עם חותמת זמן, ומה לא נאמר בהרצאה.
- **2. למה פרומפטינג מקומי לא מתרחב:** שלוש הסיבות לצאת מהלפטופ.
- **3. מה זה software factory:** ההגדרה של Lloyd ושלושת המרכיבים שלה.
- **4. הכלים וה-stack:** הכישורים החדשים של המהנדס ושש שכבות התשתית.
- **5. הדמו:** factory as code, live view של כל הסוכנים, מטריקות עלות ו-online evals.
- **6. Self-improvement ו-benchmarks:** סוכנים שפותחים PR נגד הגדרת המפעל, ובדיקה על הדאטה שלך.
- **7. מה עושים עכשיו, ו-8. מילון מונחים.**

הערת מקור המסמך הופק אוטומטית מהתמלול של ההקלטה ומהשקפים שלה. ייתכן ששמות ומונחים שובשו בתמלול האוטומטי (ASR), ובמקומות כאלה הם מסומנים כלא ודאיים. הציטוטים מועתקים מילה במילה ונבדקו מול חותמת הזמן שלהם. לא נוסף שום מידע ממקורות חיצוניים.

1. תקציר מנהלים

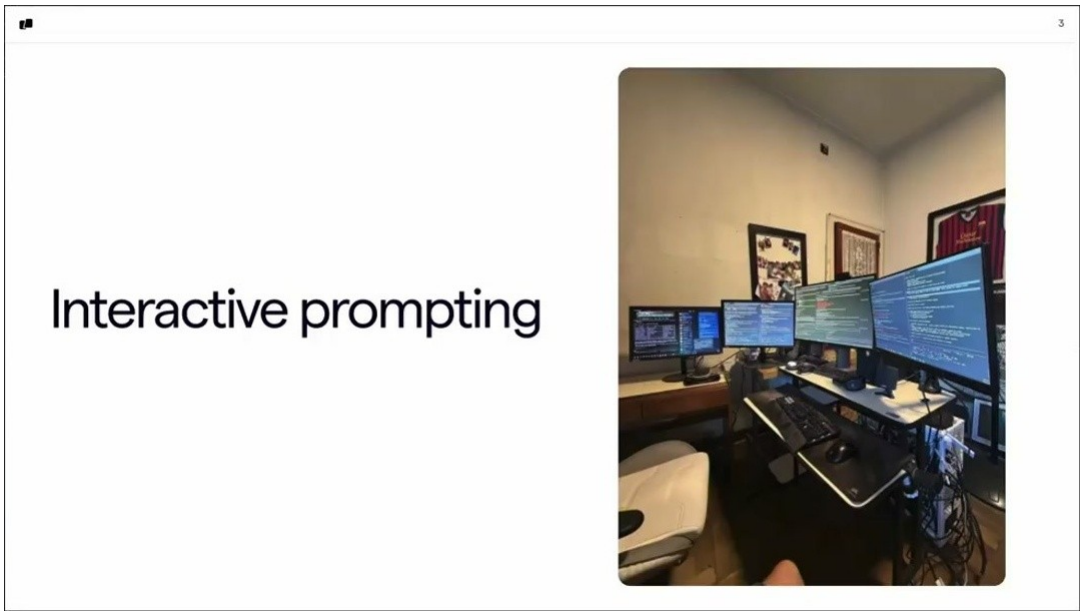
Lloyd מציג את ה-software factory כשלב הבא אחרי סוכנים מקומיים, ומגדיר אותו בצורה מצומצמת ומחייבת: מערכת שמוגדרת כקוד, פועלת בלולאה סגורה ומשפרת סוכני קוד בענן באמצעות מדידה מתמשכת. בחלק השני של ההרצאה הוא מדגים את זה על Warp Factories, ומדגיש שהעקרונות תקפים גם למי שבונה מפעל בעצמו.

- **פרומפטינג אינטראקטיבי חוסם אוטומציה:** אם כל סוכן צריך שמישהו יפעיל אותו ידנית, התפוקה מוגבלת ביכולת האנושית. מעבר לזה, כשכל מפתח בוחר לעצמו מודלים ו-skills, קשה לשלוט בעלות ובאבטחה.
- **ההגדרה המחייבת:** software factory הוא "code-defined closed-loop system" למדידה ולשיפור מתמשכים של סוכנים בענן. אוסף אוטומציות, "AI teammates" או skills ו-MCPs משותפים הם לדבריו רק primitives.
- **הכול כקוד, בסגנון Terraform:** סוכנים, MCPs, workflow, triggers ומקום הריצה, הכול מוגדר בקבצים דקלרטיביים. ככה אפשר לפתוח branch, לעשות rollback ולאפשר self-improvement.
- **מדידה במקום vibes:** בלי מדידה אי אפשר לדעת אם מקבלים את האיכות הטובה ביותר בעלות הנמוכה ביותר.
- **Stack של שש שכבות:** factories as code, data and context, compute (sandboxing), inference (מודלים ו-harnesses שאפשר להחליף), orchestration, ומעליהן ממשק API/MCP-first.
- **Online evals על כל הסשנים:** סוכן נוסף מדרג ריצות שכבר הסתיימו. בדוגמה שהוצגה ה-eval מחפש טסטים מיותרים, failure mode נפוץ אצל סוכני קוד.
- **Self-improvement בצורת PR:** סוכנים מזהים דפוסים בריצות שנכשלו ומציעים תיקון בצורת PR נגד הגדרת המפעל.
- **Benchmarks על הדאטה שלך:** לוקחים משימות אמיתיות ומריצים אותן שוב מול קונפיגורציות שונות של מודל, harness או skills, כדי לראות את ה-trade-off בין עלות לאיכות.
- **טענת תוצאה בלי מספרים:** לדברי Lloyd, Warp שיפרה "drastically" את התפוקה והורידה עלויות בעזרת הכלים האלה, אבל הוא לא נתן נתונים.

מה לא נאמר

- **אין מספרים לטענת ההצלחה:** לא נמסר כמה גדלה התפוקה ובכמה ירדה העלות ב-Warp. בדמו הופיעו סכומים (למשל עלות ל-PR), אבל לא הוסבר אם אלה נתונים אמיתיים.
- **אין פירוט על בקרת ה-self-improvement:** לא נאמר מי מאשר את ה-PRs שהסוכנים פותחים, ואם הם עוברים benchmark לפני merge.
- **קוד המפעל הוצג רק בקצרה:** Lloyd עצמו אמר שהקוד "not going to be that interesting". לא הוצגו סכמה או פורמט קונקרטיים.
- **אין failure stories:** הדוגמה היחידה לכשל הייתה טסטים מיותרים. לא סופר מה נשבר בדרך ומה היו עושים אחרת.
- **אין מחיר ולא נאמר לאיזה גודל ארגון זה מתאים:** לא נאמר מתי המפעל מצדיק את ההשקעה מול עבודה מקומית.

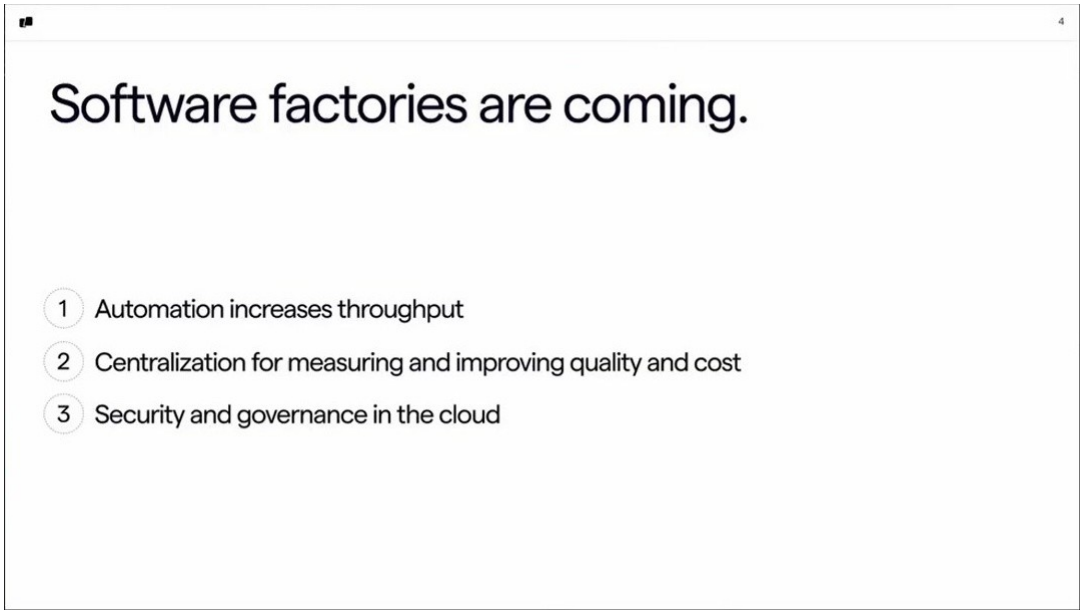
2. למה פרומפטינג מקומי לא מתרחב



השקף "Interactive prompting": עמדת עבודה של מפתח עם כמה מסכים. כך רוב המפתחים עובדים היום עם סוכני קוד.

Lloyd מתחיל מהמצב הנוכחי. רוב המפתחים וצוותי הפלטפורמה שהוא פוגש מריצים סוכנים מקומית, לפעמים כמה במקביל, מתוך הטרמינל או מתוך agentic development environment. כששאל את הקהל, רוב האנשים עדיין עובדים ככה, אבל לא כולם. כששאל מי בונה או משתמש ב-software factory פנימי, הוא העריך שהקבוצה הזו כבר גדולה כמעט כמו הקבוצה הראשונה.

הוא מציג שלוש סיבות לשינוי. הראשונה היא תפוקה: "you're kind of limited by human throughput really". השנייה: כשכל אחד בוחר לעצמו מודלים ו-skills, קשה לוודא שהסוכנים יעילים מבחינת עלות ופרודוקטיביות. השלישית היא בעיית אבטחה ו-governance.



"Software factories are coming": שלוש הסיבות למעבר, אוטומציה שמעלה תפוקה, ריכוזיות שמאפשרת למדוד ולשפר איכות ועלות, ואבטחה ו-governance בענן.

3. מה זה software factory: ההגדרה

לפני שהוא מגדיר מה זה מפעל, Lloyd מסביר מה זה לא: אוסף אוטומציות לא מתואמות, צוות של "AI teammates", ספריית skills ו-MCPs משותפת, או swarm אישי של סוכנים. לדבריו כל אלה primitives שימושיים, אבל הם לא מצב היעד.

> **Zach Lloyd** — I define it as a code-defined closed-loop system for continuously measuring and improving cloud coding agents.

- **מוגדר כקוד:** כל הסוכנים, איך הם מופעלים, ה-workflow מ-triage עד spec, אילו MCPs זמינים ואיפה הסוכנים רצים. הכול בקבצים דקלרטיביים "in a Terraform style".
- **לולאה סגורה:** בעצם ה-SDLC. רעיונות מגיעים מאנשים או מסיגנלים אוטומטיים, והסוכנים מעבירים אותם דרך triage, implementation, verification ו-monitoring. ככל שהסוכנים רצים יותר, אפשר לייעל את הלולאה יותר.
- **הכול מדיד:** "If you're just going on vibes in terms of how you deploy coding agents, you can't be confident that you're actually getting the best, highest quality outputs at the best cost."



"Measurement > Vibes": גרף של תוצאות benchmark שבו כל מודל מסומן כנקודה לפי עלות ממוצעת ושיעור הצלחה. שמות המודלים בשקף קטנים מכדי לקרוא אותם בוודאות.

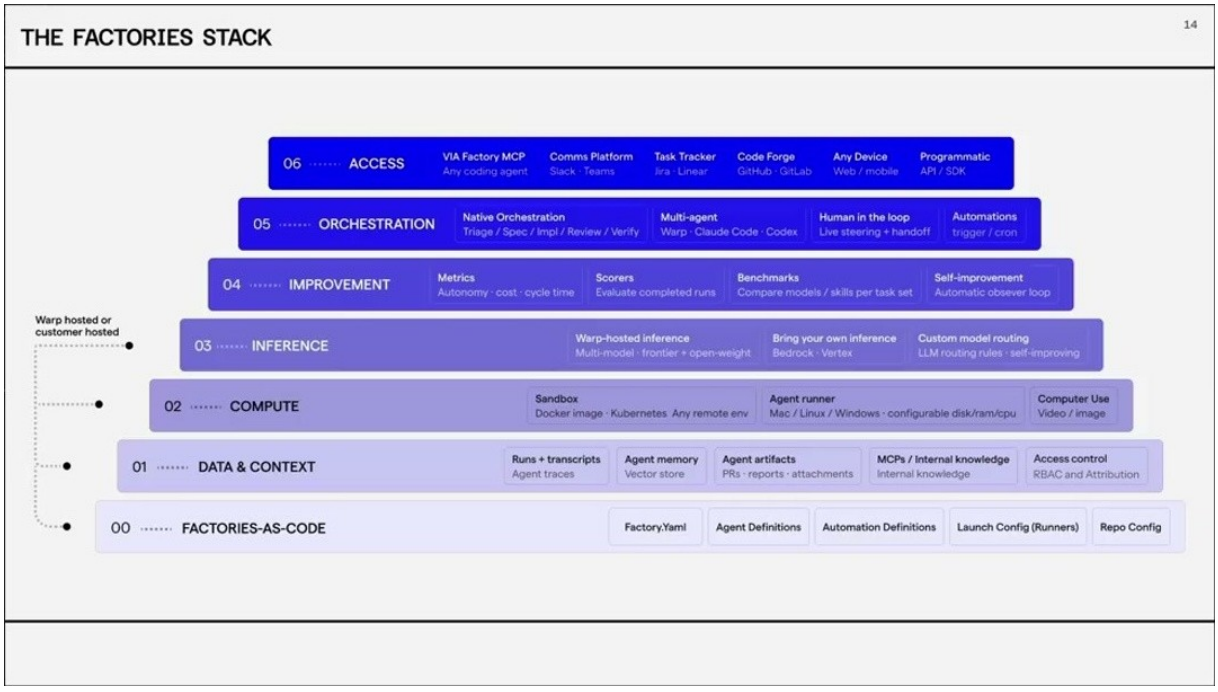
על המסר הזה Lloyd חוזר לאורך כל ההרצאה: סוכנים הם primitive הנדסי חדש, וצריך לגשת אליהם כמו מהנדסים. "To me, it's not about growing a garden or following a recipe."

4. הכלים החדשים וה-stack של המפעל



"The factory engineering mindset": שני עקרונות, למדוד הכול בעצמך על הדאטה שלך, ולשפר באופן מתמשך.

Lloyd משווה את זה לתקופה שלו ב-Google. מה שהבדיל אז בין מהנדסים מצוינים למהנדסים סבירים היה שליטה ב-debuggers, ב-profilers וב-Chrome DevTools. ב-factory engineering יש ארגז כלים חדש: evals, לולאות self-improvement שבאמת עובדות, כתיבת benchmarks ו-A/B testing של המפעל. הוא מדגיש שלא מדובר בהחלפה של product engineering. המהנדס עדיין אחראי לכך שהמוצר פותר בעיה אמיתית, אבל כל מהנדס יצטרך ללמוד גם איך לשפר את המערכת, ולא רק לבנות את המוצר.



"The Factories Stack": פירמידה של שכבות, מ-Factories-as-code ו-Data & Context בבסיס, דרך Compute, Inference, Orchestration ו-Improvement, ועד Access בראש.

שכבה	מה היא נותנת לפי Lloyd
Factories as code	branching של קונפיגורציות, rollback, ובעיקר בסיס ל--self improvement
Data & context	תיעוד של כל השיחות, התוצרים, הסרטונים וצילומי המסך של הסוכנים. זה חומר הגלם לשיפור
Compute	sandboxing לאבטחה ולשחזוריות, רצוי multi-platform

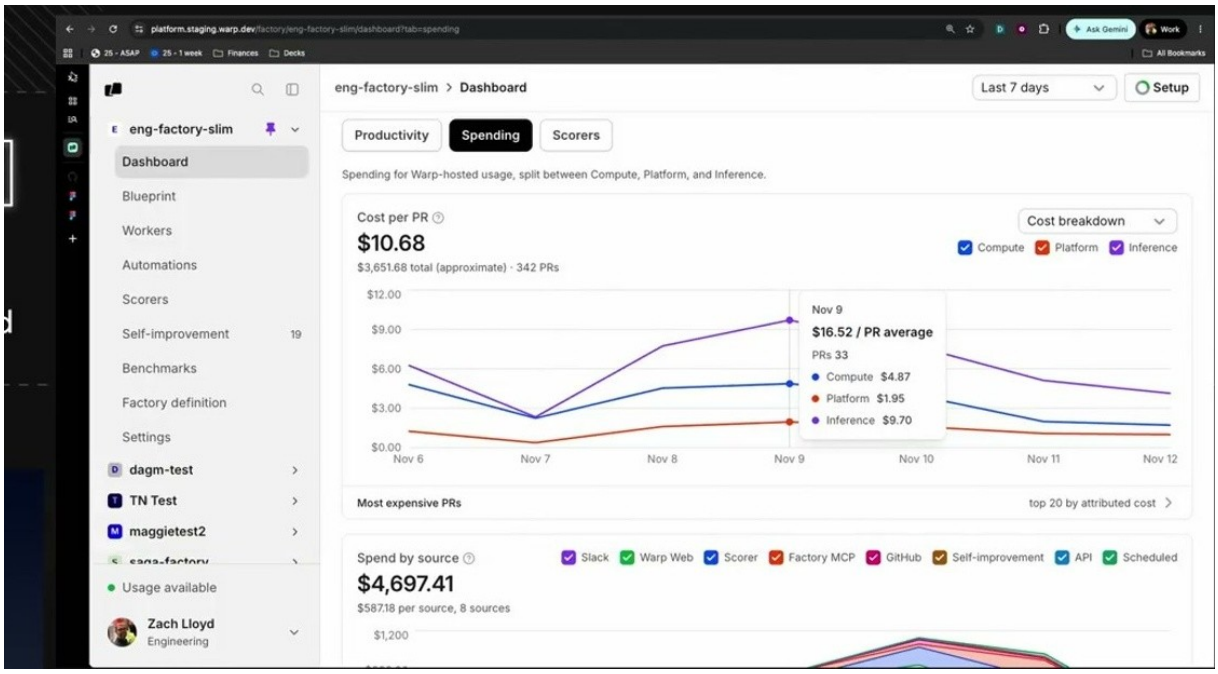
שכבה	מה היא נותנת לפי Lloyd
Inference	מודלים ו-harnesses הם "a degree of freedom to test":" Claude Code, Codex, Pi, Warp, או harness פנימי עם הקונטקסט של הארגון
Orchestration	מזיזה עבודה בתוך המפעל: "In some ways, the least interesting part, but you need that."
API / MCP first	גישה לאנשים ולסוכנים אחרים, ואינטגרציה עם Slack, Linear, Jira ו-GitHub

שימו לב לתמלול בתמלול כתוב "Cloud Code". מההקשר של רשימת ה-harnesses, כמעט בוודאות הכוונה ל-"Claude Code. "Pi" מופיע כשם של harness ולא ברור אם התמלול שלו מדויק.

5. הדמו: מקוד המפעל ועד online evals

Lloyd עובר ל-Warp Factories ומדגיש שאותם עקרונות תקפים גם למי שבונה מפעל בעצמו. בבסיס יש הגדרה דקלרטיבית של המפעל: policies, repos שהוא ניגש אליהם, triggers ו-blueprint של ה-workflow. גם כל שאר רכיבי המפעל מוגדרים בקוד.

השכבה הבאה היא "live view" Google Docs style של כל הסוכנים שרצים אצל כל מהנדסי Warp. אפשר להיכנס לסשן של מהנדסת בשם Maggie ולראות את השיחה, את התוצרים ואת המקור שממנו הופעל הסוכן, במקרה הזה Slack. לפי התמלול הסוכן רץ על "Opus 5.5 on the high plan", אבל זה כנראה שיבוש של שם המודל וההגדרה ולא כדאי להסתמך עליו. לדברי Lloyd, הערך האמיתי הוא לא בצפייה עצמה אלא באיסוף המרוכז של המידע.



דשבורד Spending ב-Warp Factories: גרף של Cost per PR (בערך $10.68) לאורך זמן, ופירוט של Spend by source (בערך $4,697.41). לא הוסבר אם הנתונים אמיתיים.

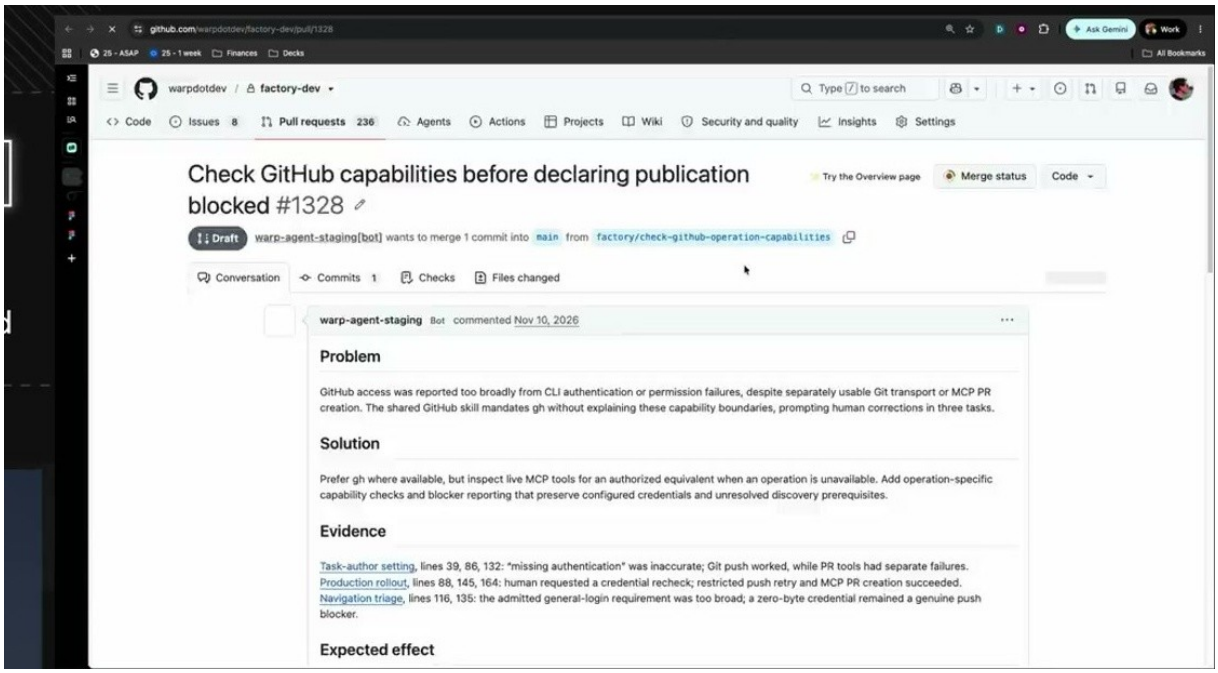
כשהדאטה מרוכז, אפשר למדוד בלי להתקין שום דבר על הלפטופים. מודדים מטריקות קלאסיות כמו DORA ו-cycle time, מדדי אוטומציה כמו מספר האינטראקציות האנושיות ל-PR ותפוקה, וגם עלות. "if you don't have this data, you're kind of flying blind."

השכבה שמעליה היא online evals: סוכן נוסף עובר על הסשנים שהוקלטו ומדרג אותם לפי מדד מסוים. הדוגמה של Warp היא scorer שמזהה טסטים מיותרים, failure mode נפוץ של סוכני קוד. השאלה הבאה היא למה הסוכן כותב יותר מדי טסטים, ואיזה מנגנון במפעל אפשר לשנות כדי לתקן את זה.

6. Self-improvement ו-benchmarks על הדאטה שלך

ב-self-improvement, את הפלט של ה-scorers בוחנים סוכנים ולא אנשים. הסוכן מסתכל על הריצות שנכשלו, מחפש דפוסים ומציע פתרון. Lloyd רואה בזה את התמורה העיקרית של הגדרת המפעל כקוד:

> — **Zach Lloyd** And then the form that that solution takes is as a PR against your factory definition.



PR שסוכן פתח נגד הגדרת המפעל: "Check GitHub capabilities before declaring publication blocked #1328", עם הסעיפים Problem, Solution, Evidence ו-Expected effect.

הכלי האחרון הוא benchmarks. בכל פעם שיוצא מודל, skill או harness חדש, מופיעות טענות שהוא "הכי טוב". לדברי Lloyd: "The only way to answer those questions for real is to test on your own data". בונים סט benchmark ממשימות אמיתיות ומריצים אותן שוב מול קונפיגורציות שונות של מודל, harness או מפעל. התוצאה נראית כמו גרף של SWE-bench או Terminal Bench (בתמלול: "Sweebench"), ומטרתה להראות את ה-trade-off בין עלות לאיכות. רוצים להיות ברביע השמאלי העליון, אבל על הדאטה שלכם. בודקים לא רק מודלים, אלא גם שאלות כמו "אם נוריד את ה-skills האלה, המפעל ישתפר או יחמיר?".



שקף הסיום בכחול: "Prompt your agents" ו-"Manage your agents" מחוקים, ומתחתם "Engineer your agents".

> **Zach Lloyd** — I know we're not writing code, but I'm still using all the engineering skills that I spent my whole career learning.

7. מה עושים עכשיו

- **לתעד כל ריצה של סוכן:** שיחות, artifacts וצילומי מסך, במקום מרכזי. בלי הדאטה הזה אין חומר גלם ל-evals או ל-benchmarks.
- **להעביר את הקונפיגורציה לקוד:** להתחיל במה שכבר קיים, כמו MCPs, skills, triggers ובחירת מודל. כך אפשר לעשות branch ו-rollback.
- **לבנות eval ראשון אחד:** לבחור failure mode שמכירים, כמו טסטים מיותרים בדוגמה של Warp, ולהריץ scorer על מספר אמיתיים.
- **למדוד עלות ל-PR ומספר אינטראקציות אנושיות ל-PR:** אלה שני מדדים שאפשר להתחיל לעקוב אחריהם מיד, מעבר ל-DORA.
- **benchmark פנימי לפני החלפת מודל:** לבנות סט קטן ממשימות אמיתיות ולהשוות עליו עלות מול איכות, לפני שמאמצים מודל או harness חדש.

שאלה פתוחה	למה זה חשוב
מי מאשר את ה-PRs של self-improvement, ולפי מה?	בלי gate יש סיכון ל-regression במפעל עצמו
האם ה-PRs עוברים benchmark לפני merge?	זה החיבור החסר בין שני הכלים שהוצגו
מה גודל השיפור בפועל ב-Warp?	נאמר רק "drastically", בלי מספרים
איך בונים benchmark מייצג מדאטה פנימי?	replay של משימות דורש סביבה שאפשר לשחזר. אולי זה הקשר ל-sandboxing, אבל לא פורט
מאיזה גודל צוות זה משתלם?	עלות התשתית מול עבודה מקומית לא נדונה

8. מילון מונחים

מונח	משמעות
Software factory	מערכת שמוגדרת כקוד ומריצה סוכני קוד בענן בלולאה סגורה, עם מדידה ושיפור מתמשכים
Factory engineering	הדיסציפלינה של שיפור המפעל עצמו באמצעות evals, benchmarks, A/B ו-self-improvement
Factories as code	הגדרה דקלרטיבית של סוכנים, workflow, triggers ו-policies, בדומה ל-Terraform
Closed loop	מחזור ה-SDLC שבו כל ריצה מייצרת דאטה שמשפר את הריצות הבאות
Harness	שכבת הסוכן שעוטפת את המודל, למשל Claude Code, Codex או Warp
Sandboxing	סביבת ריצה מבודדת לסוכן, שמשפרת אבטחה ושחזוריות
Orchestration	השכבה שמזיזה עבודה בין שלבי המפעל
Online evals	דירוג אוטומטי של סשנים שכבר רצו, על ידי סוכן נוסף
Self-improvement	סוכנים שמנתחים כשלים ופותחים PR נגד הגדרת המפעל
Benchmark	סט משימות קבוע שמשווים עליו קונפיגורציות לפי עלות ואיכות
DORA metrics	מדדי ביצועים מקובלים בהנדסה, למשל cycle time
MCP	Model Context Protocol, פרוטוקול לחיבור כלים ומקורות מידע לסוכנים