מה באמת נדרש כדי לבנות Software Factory

שלושה עקרונות מ-Factory: Agnostic, Autonomous ו-Always improving. כתיבת הקוד היא החלק הקל, והקושי האמיתי הוא הגדרת "done" ואימות שלו

Tereza Tížková (Factory) · AI Engineer · 27.09.2026

משך: כ-23 דקות · YouTube

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה של כ-23 דקות מערוץ AI Engineer. לפי השקפים היא צולמה ב-AI Engineer World's Fair. הדוברת היא Tereza Tížková מחברת Factory, שלדבריה מפעילה software factory בפרודקשן אצל ארגונים כמו EY ו-Adobe. ההגדרה שהיא מציעה: software factory הוא מחזור החיים המלא של פיתוח תוכנה, והוא רץ באופן אוטונומי. זה כולל איסוף סיגנלים (משוב משתמשים, לוגים), תעדוף, תזמור, ביצוע, אימות ובדיקות בפרודקשן, ושיפור מתמשך. זה לא רק יצירת קוד.

הטענה המרכזית: כתיבת קוד היא החלק הקל, ולכן גם swarm של אלפי coding agents הוא לא software factory. כדי לבנות אחד נדרשים שלושה עקרונות. הראשון הוא Agnostic: להתאים לדרך שבה הארגון כבר עובד, ולנתב בין מודלים. השני הוא Autonomous: לרוץ שעות ואפילו שבועות, ולהגדיר "done" שאפשר לאמת. השלישי הוא Always improving: לנהל context, לבדוק אם ה-codebase מוכן ל-agents, ולקודד את הידע הלא-כתוב של הצוות. ההרצאה מדגימה כל עיקרון במוצרים של Factory: automatic model routing, Missions, deferred context engine ו-agent readiness.

איך לקרוא את המסמך

- **1. תקציר מנהלים:** הטענות המרכזיות עם ראיות וחותמות זמן, ומה לא נאמר.
- **2. מהו software factory ומה הוא לא:** ההגדרה, למה זה לא עבד ב-2023, וההבדל בין אימוץ AI לבין ארגון שבנוי על AI.
- **3. Agnostic:** ממשקים ומנויים, המקרה של Coinbase, automatic model routing ו-caching.
- **4. Autonomous:** loops שאפשר לאמת, בעיית ה-cheating, ו-Factory Missions עם orchestrator, workers ו-validators.
- **5. Always improving:** deferred context engine, ה-power law של אימוץ AI, agent readiness ו-plugins.
- **6. מה נשאר לבני אדם:** המעבר מ"איך לבנות" ל"מה לבנות", והוצאת עבודת התיאום החוצה.
- **7. מסקנות ופעולות:** מה לעשות מחר בבוקר, וטבלת שאלות פתוחות.
- **8. מילון מונחים:** המונחים באנגלית שמופיעים במסמך.

הערת מקור המסמך הופק אוטומטית מהתמלול של ההקלטה ומהשקפים שחולצו ממנה. שמות ומונחים עלולים להשתבש בתמלול האוטומטי (ASR). לדוגמה, בתמלול נשמעים "factory.com", "Teresa" ו-"UI", ולפי תיאור ההרצאה הכוונה ל-Tereza, ל-Factory (factory.ai) ול-EY. הציטוטים מובאים מילה במילה ונבדקו מול חותמת הזמן שלהם. לא נוסף מידע ממקורות חיצוניים.

1. תקציר מנהלים

ההרצאה משלבת עמדה עקרונית עם הצגת מוצר. מול ההייפ סביב "software factory", Tížková מציעה הגדרה תפעולית ושלושה עקרונות, וכל אחד מהם מגובה ביכולת של Factory. חלק מהמספרים קונקרטיים: חיסכון של כ-25% מ-routing, mission אמיתית של 16 שעות, 40% מהזמן על validation וחיסכון של 50%+ בטוקנים. אבל כמעט אף אחד מהם לא מגיע עם מתודולוגיה.

- **כתיבת קוד היא החלק הקל:** software factory אינו coding agent ואינו swarm של agents. האתגר הוא כל מה שמסביב לכתיבה: סיגנלים, תעדוף, אימות ושיפור [3:01]. שקף תומך: 80% מזמן ההנדסה אינו כתיבת קוד.
- **חיסכון בלי לקצץ בטוקנים:** ב-Coinbase המשיכו להגדיל את צריכת הטוקנים והפסיקו להגדיל את ההוצאה. הכלים: מודלי ברירת מחדל זולים יותר, caching, ביטול מגבלות תקציב בתמורה לדרישה לתוצאות, ו-routing [6:04].
- **Model routing חוסך כ-25% לפחות:** ה-router מסווג את קושי המשימה, מגדיר סף ובוחר את המודל הזול ביותר שמעל הסף. לפי Tížková הבנצ'מרק שלהם "[7:37] very conservative".
- **מחיר ה-caching הוא החלטה עסקית:** גם מודלים פתוחים על compute ייעודי נהנים מ-caching. כמה מההנחה עוברת למשתמש זו החלטת תמחור, לא אתגר טכני [9:10].
- **השאלה היא מתי loop נחשב "done", לא ה-loop עצמו:** משימות פתוחות ולא דטרמיניסטיות, ו-agent יכול "לרמות" ולעבור את הטסטים בלי לבצע את המשימה [10:43].
- **Missions רצות בסדרה, לא ב-swarm:** orchestrator כותב validation contract לפני שנכתב קוד, workers עובדים בזה אחר זה עם context נקי, ו-validators בודקים קוד שלא הם כתבו. ב-mission אמיתית של 16 שעות, ה-validation לקח כ-40% מהזמן [12:15].
- **Validator שמקליק באפליקציה:** ה-validator user testing עובד במחשב וירטואלי ובודק שהמוצר באמת עובד, ולא רק נראה טוב בקוד [13:49].
- **Deferred context חוסך 50%+ טוקנים:** ל-agent מוצגת רשימה קצרה של tools עם תיאורים קצרים, ו-tool נטען במלואו רק כשצריך אותו. ככל שיש יותר tools, החיסכון גדל [16:53].
- **AI על codebase מבולגן מחמיר את המצב:** אימוץ AI מתנהג כמו power law. בלי codebase מובנה, תיעוד וטסטים, ה-AI מדרדר את הקוד, ולכן Factory מציעה בדיקת agent readiness [16:53, 18:25].

מה לא נאמר

- **Build או outsource:** השאלה הזו מופיעה בשקף האג'נדה, אבל ההרצאה לא עונה עליה במפורש. מלבד הקביעה שאין "לזרוק קונסולטנט באמצע הארגון", אין קריטריונים להחלטה.
- **כמה זה עולה:** גם השאלה "how much is this all going to cost" הוצגה בפתיחה. לא ניתן מחיר ל-mission, ולא עלות טוקנים ל-mission של 16 שעות.
- **מתודולוגיה:** לא הוסבר על מה נמדד ה-25% של ה-routing, מול איזה baseline נמדד ה-50% של deferred context, וכמה פעמים ה-router מנתב לא נכון.
- **Governance והרשאות:** ההרצאה מזכירה את הצורך לתת ל-agents "the right trust and all this permissions and governance", ולא מפרטת איך.
- **כשלים:** לא הוצג אף מקרה שבו mission נכשלה או ש-validator פספס באג. מלבד השקף שמזכיר cheating ו-broken loops, אין נתוני אמינות של Missions עצמן.

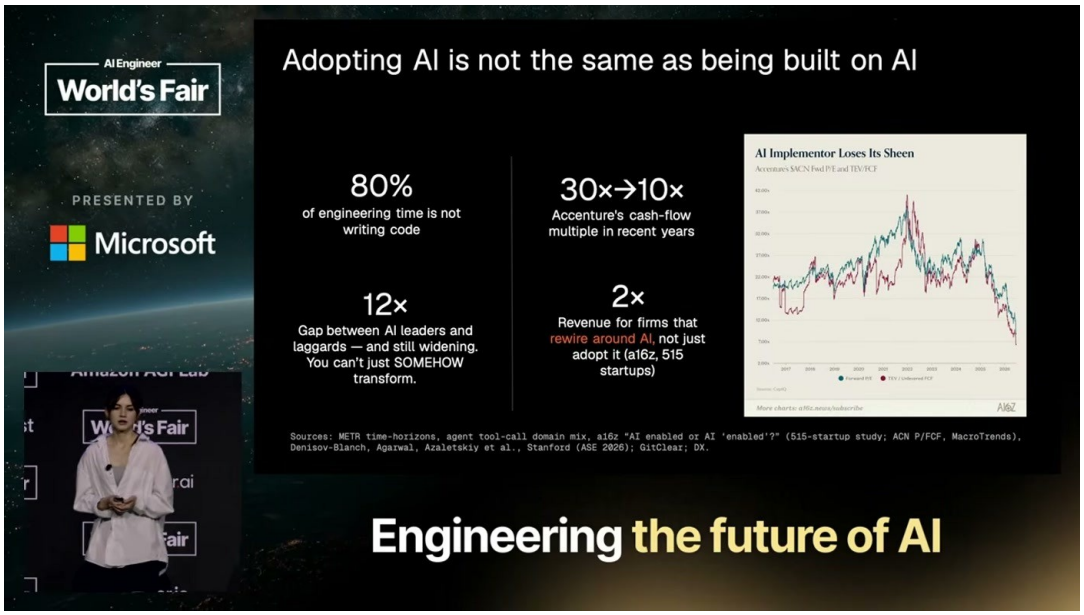
2. מהו software factory ומה הוא לא

שקף הפתיחה "No one knows what software factory means" עם האג'נדה: מהו software factory, האם לבנות לבד, איך, מה עובד בפרודקשן ובאיזו עלות. לצידה צילומי מסך של כותרות ופוסטים על המונח

Tížková פותחת בהגדרה רחבה. software factory הוא "the whole life cycle of developing software with autonomy", כלומר איסוף סיגנלים, תגובה למשוב משתמשים וללוגים, תעדוף, תזמור, ביצוע, אימות, בדיקות בפרודקשן, איטרציה ורכישת ידע ו-skills חדשים תוך כדי [1:30].

למה זה לא עבד קודם? הרעיון של loop מתמשך קיים מאז AutoGPT ו-BabyAGI, אבל ב-2023 המודלים עדיין הזו, ה-context היה קצר, איכות ה-reasoning הייתה נמוכה, ולא היו סביבות מבודדות שבהן agents יכולים לעבוד [1:30]. ההגדרה השלילית חשובה לא פחות: software factory אינו coding agent, אינו swarm של coding agents, ואינו אסטרטגיה מופשטת שקונסולטנט מוכר. לדעתה צריך לבנות את הארגון מחדש מהיסוד.

> **Tereza Tížková [3:01]** — generating code, writing code, that's the easy part compared to all the others. Engineers don't even spend most of the time just writing code.



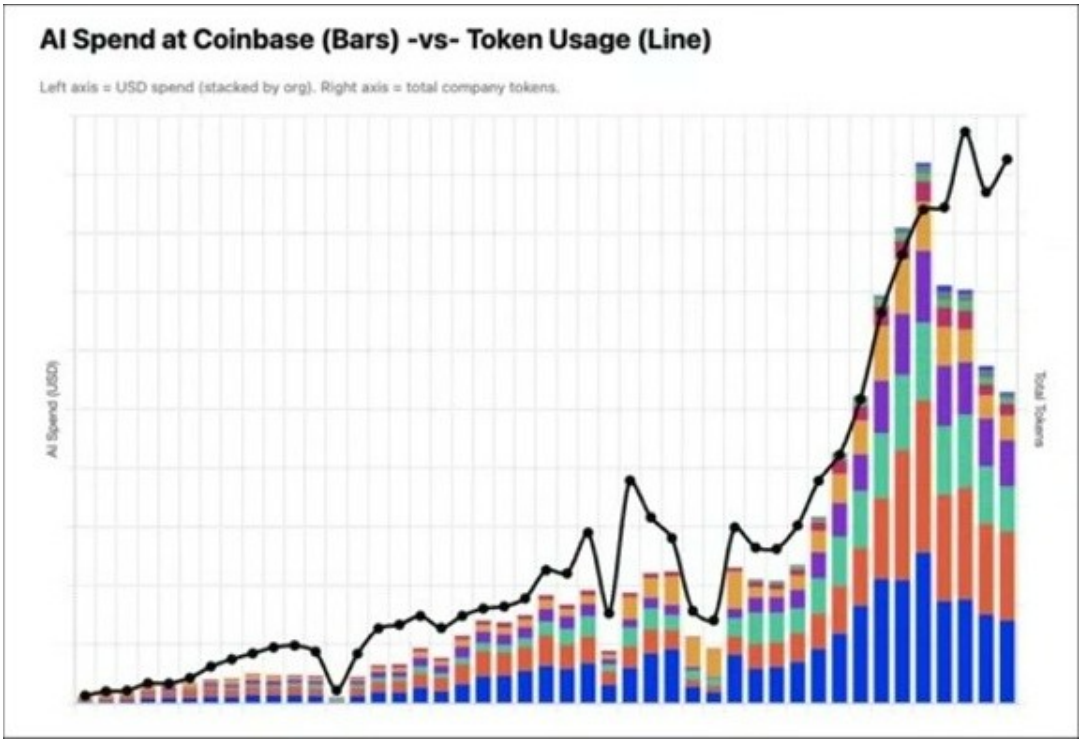
"Adopting AI is not the same as being built on AI": ארבעה נתונים, ביניהם 80% מזמן ההנדסה שאינו כתיבת קוד, פער של 12x בין leaders ל-laggards ופי 2 בהכנסות לחברות שבנו את עצמן מחדש סביב AI, לצד גרף מניות. הטקסט הקטן בשקף קשה לקריאה, והמקורות ופרטי הנתון 30x→10x לא ודאיים

3. עיקרון ראשון: Agnostic

"We need to build software factory primitives that work for everyone": עבודה על כל ממשק (Cloud, local, CI, IDE, one agent), מעבר בין סביבות בלי לשכתב, portability beats lock-in, bring your own model keys ו-bring your own machine, לצד גרף "In what domains are agents deployed?"

החלק הראשון של העיקרון הוא התאמה לדרך שבה הארגון כבר עובד. software factory צריך לעבוד מתוך Slack ו-GitHub, להתחבר לכל הכלים ולאפשר למשתמשים להביא את המנויים שכבר יש להם. הנימוק: מודלים ובנצ'מרקים מתחלפים כל הזמן, ואי אפשר לנחש מי ינצח [4:33].

החלק השני הוא עצמאות מספקי מודלים. הדוגמה היא גרף שפרסם ה-CEO של Coinbase: צריכת הטוקנים ממשיכה לעלות וההוצאה מתייצבת. לפי Tížková, זה הושג בארבעה צעדים. הראשון: מודלי ברירת מחדל שאינם frontier. השני: caching, כדי לא לבצע prefill מחדש בכל פעם. השלישי: אין מגבלת הוצאה, אבל מי שמוציא הרבה צריך להראות תוצאות. הרביעי: routing חכם בין מודלים [6:04].



"Token optimization is winning over just cost saving": גרף ההוצאה על AI ב-Coinbase מול צריכת הטוקנים, ולצידו רשימה: מודלי ברירת מחדל טובים יותר, שיפור ב-cache hit rate, בלי מגבלת הוצאה אבל עם דרישה ל-impact, ו-ROUTING. שמות המודלים והאחוזים המדויקים בשקף לא קריאים בוודאות

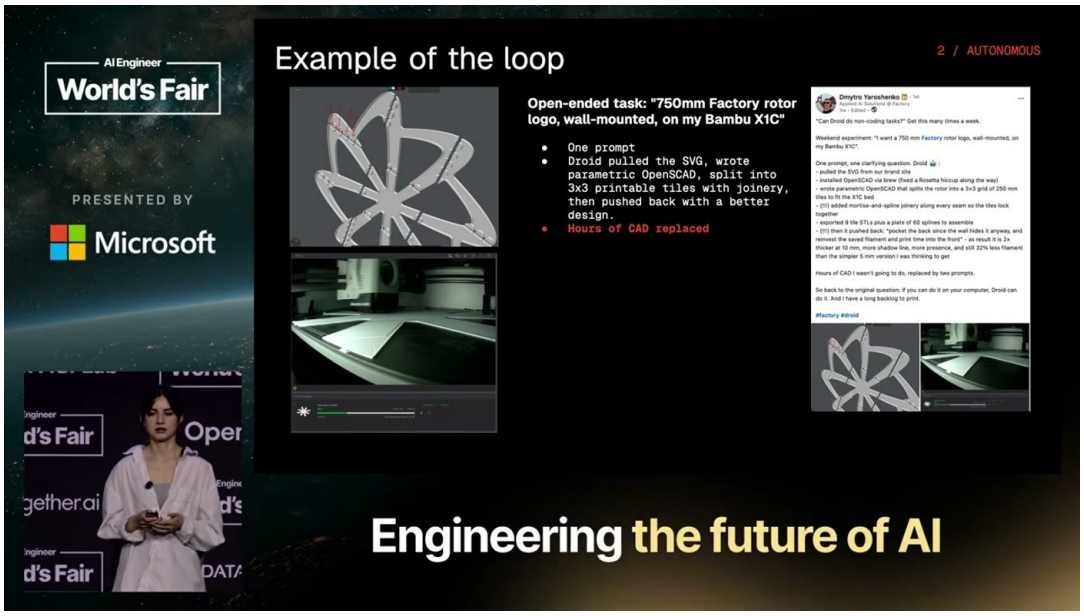
ל-automatic model routing של Factory יש ארבעה שלבים. בשלב הראשון המשתמש מקצה משימה. ארגון יכול להגדיר הרשאות ומודלי ברירת מחדל שונים למחלקות שונות, למשל marketing, sales ו-engineering. השלב השני הוא סיווג, "the magic of the routing": ה-router מסתכל על מבנה ה-prompt, על ה-codebase, על קושי המשימה ועל ה-tools שבשימוש. בשלב השלישי נקבע סף של מה שמספיק כדי לבצע את המשימה. בשלב הרביעי נבחר המודל הזול ביותר שמעל הסף [7:37]. אם המודל נכשל, אפשר להעביר את המשימה למודל חזק יותר באמצע. לטענתה זה קורה לעתים רחוקות, ובסך הכול התהליך עדיין מהיר יותר. לדבריה ה-router משפר גם אמינות ומהירות: מודלים פתוחים מהירים יותר לעתים קרובות, ואם ספק אחד נופל אפשר לעבור לאחר אוטומטית [6:04].

> **Tereza Tížková [6:04–7:37]** This is our benchmark which is very conservative. I prefer — conservative benchmarks, but I think the direction is clear. You can say, for example, 25%, but even more, probably.

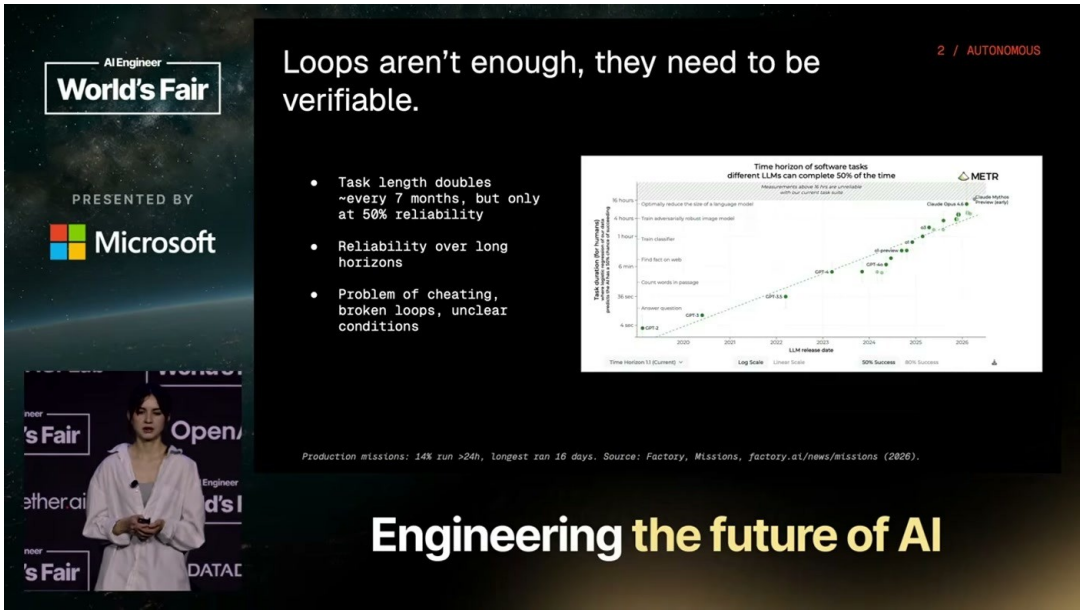
לגבי caching, השאלה שחזרה אחרי ההשקה הייתה אם המשתמשים מקבלים הנחה. לדבריה, מודלים פתוחים על compute ייעודי נהנים מאותו יתרון כמו המעבדות, ולכן "the final price for users is just a pricing decision. It's not a technical challenge because everyone can do caching" [9:10].

4. עיקרון שני: Autonomous

לפי Tížková, loops היו קיימים תמיד, ורק עכשיו הם עוברים לעולם ה-agents. היא מזכירה את ה-Ralph loop (בתמלול: "RALF loop") ומיד מסיטה את הדיון מה-loop עצמו: "the question is not the loop itself, but the question is how you define what it means to be done in the loop" [9:10]. בתכנות הקלאסי ל-loop היה תנאי עצירה ברור. היום המשימות פתוחות ולא דטרמיניסטיות, ולפעמים אפילו פיזיות.



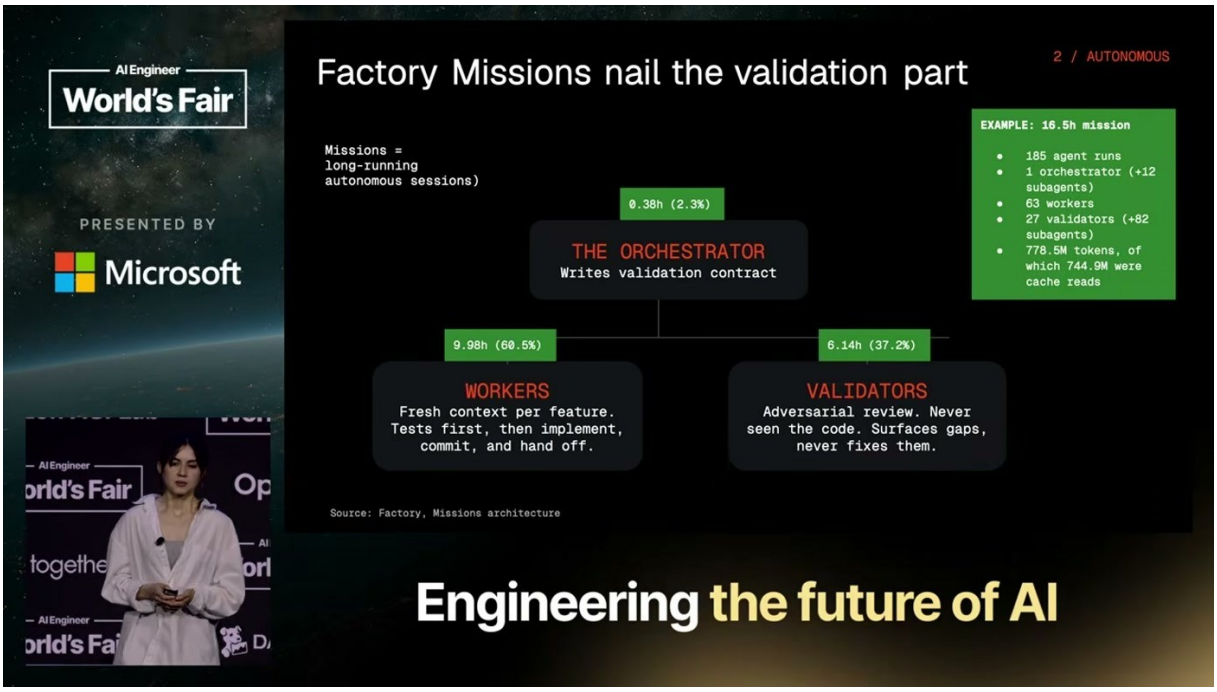
"Example of the loop": משימה פתוחה של עמית ב-Factory, לוגו Factory בקוטר 750mm להדפסה בתלת-ממד ולתלייה על קיר. prompt אחד, ה-agent שלף את ה-SVG, כתב OpenSCAD פרמטרי, פיצל את הלוגו לאריחים שאפשר להדפיס והציע עיצוב משופר. מופיעים גם תמונות של התוצר וה-thread המקורי



"Loops aren't enough, they need to be verifiable": אורך המשימות האוטונומיות מוכפל בערך כל 7 חודשים אבל רק ב-50% אמינות (גרף בסגנון METR), אמינות לאורך horizons ארוכים, ובעיית cheating, broken loops ותנאים לא ברורים

את השקף הזה היא מכנה "the scary chart". agents רצים זמן רב יותר, אבל זה לא אומר שהם אמינים בפרודקשן. הבעיה החמורה היא cheating: אם תנאי ה-done מנוסח לא נכון, ה-agent ינסה לעבור את הטסטים במקום לבצע את מה שנדרש [10:43].

Tereza Tížková [10:43]	if you write what it means to be done in the wrong way, the agent can try — to pass your tests, but not really verify what you need to do and accomplish



"Factory Missions nail the validation part": ה-Orchestrator כותב validation contract. ה-Workers מקבלים context נקי לכל feature, כותבים טסטים ראשונים, מממשים, עושים commit ומעבירים הלאה. ה-Validators מבצעים adversarial review, לא ראו את הקוד, מצביעים על פערים ולא מתקנים אותם. בצד תיבת נתונים של mission לדוגמה

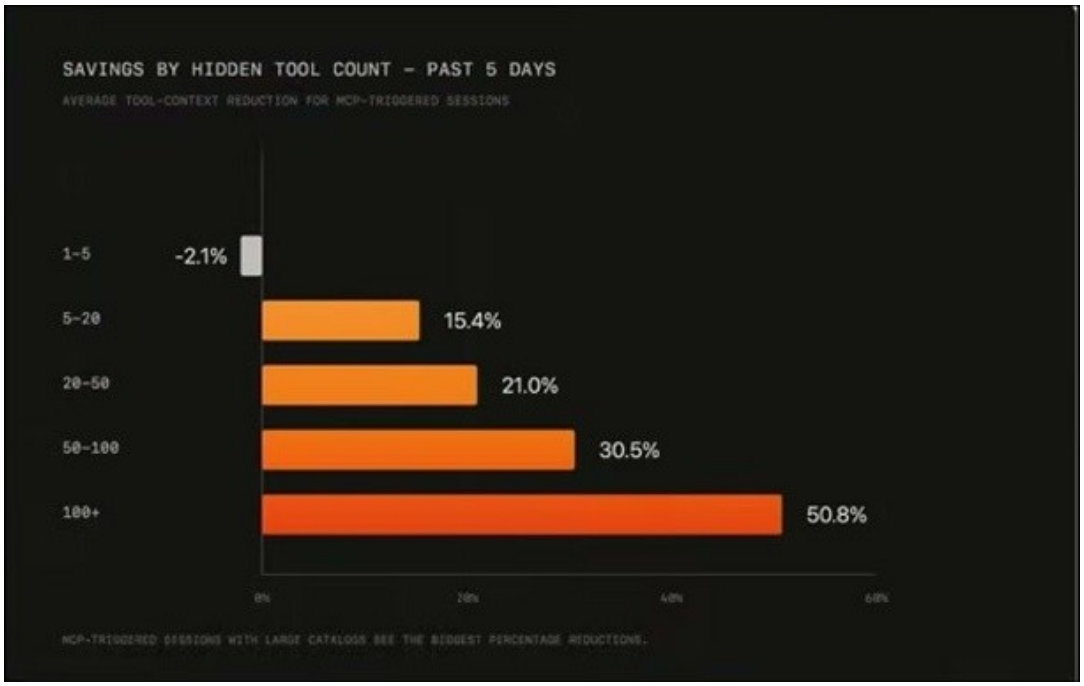
Missions הן sessions ארוכים של agents, שיכולים לרוץ ימים ואפילו שבועות. ה-orchestrator מחליט וכותב את תנאי ה-done. ה-workers מבצעים את העבודה בסדרה, לא במקביל: כל אחד משלים חלק ומעביר לבא אחריו. בתוך כל worker יכולים לרוץ sub-agents במקביל, למשל למחקר ברשת או לבניית קבצים. ה-validators מחזירים משוב לתחילת ה-loop. ה-validation contract נכתב לפני שנכתבת שורת קוד אחת [12:15, 13:49]. היא מדגימה mission אמיתית של לקוח שרצה 16 שעות, ובה "it takes even 40% of the whole process" על validation [12:15].

> **Tereza Tížková [12:15]** We actually found that if you do this, you end up with more fresh context — and kind of fresh heads.

יש שני סוגי validators. ה-scrutiny validator בודק את הקוד בקפדנות: linters, types וטסטים. ה-user testing validator "doesn't care how it was made". הוא עובד במחשב וירטואלי, מקליק ובודק שהכול עובד [13:49]. הדוגמה שלה: מהנדס שהעביר codebases בעזרת Droid, ה-agent של Factory. במוצרים אחרים התקבל מוצר שנראה גמור, אבל הוא לא עבד ולא היה אינטראקטיבי, "just a dummy result". רק הקלקה אמיתית חשפה את זה. לדבריה, מה שאפשר את זה הוא ההתקדמות ב-computer use ובסביבות וירטואליות קבועות ל-agents. היא מסכמת את המבנה כ"[15:21] loop with smaller loops".

5. עיקרון שלישי: Always improving

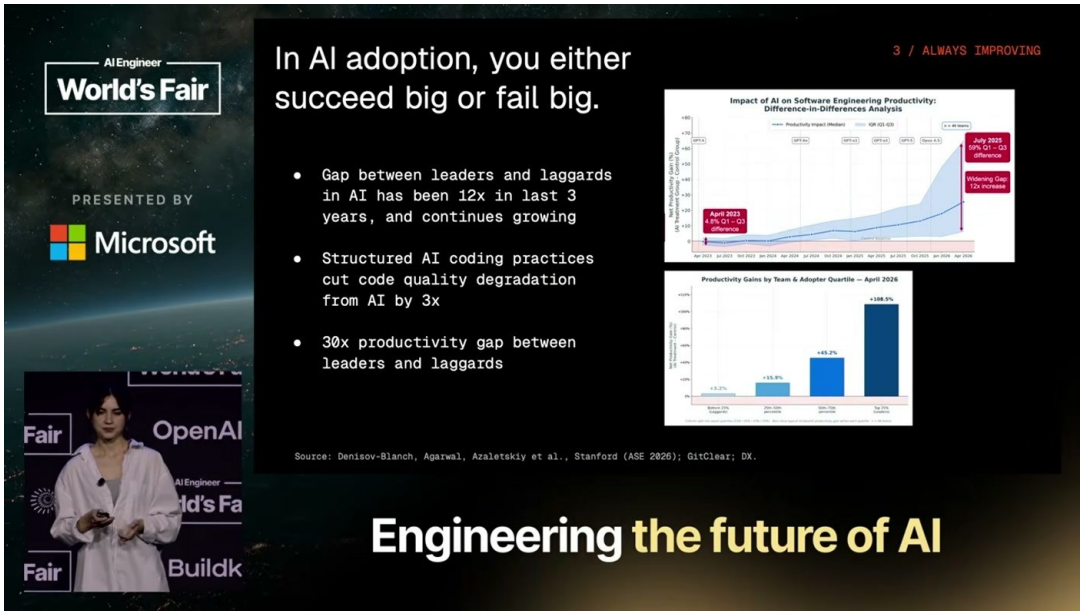
"The elephant in the room" הוא context bloat. לפי Tížková, ארגונים משתמשים בממוצע במאות כלים: Figma, Notion, Gmail, Drive, Slack. לכל כלי יש schema, פרמטרים ותיאורים ארוכים. כתוצאה, ה-agent מתבלבל בין tools עם שמות דומים, או ממלא את ה-context window ונאלץ לדחוס [15:21]. הפתרון של Factory הוא deferred context engine, שחושף מידע בהדרגה (progressive disclosure). בהתחלה ה-agent מקבל רשימה קצרה של tools עם תיאורים קצרים, וטוען tool במלואו רק כשהוא באמת צריך אותו.



"Deferred context engine saves context by progressively disclosing tools": גרף עמודות של הפחתה בשימוש ב-context ככל שמספר ה-tools גדל, לצד תמונה של Mickey Mouse. הערכים המדויקים בגרף לא קריאים בוודאות

> **Tereza Tížková [16:53]** nothing is actually removed. It's just hidden and not reachable until — needed.

לפי ההרצאה, החיסכון גדל עם מספר ה-tools: "you can save 50% of tokens or more" [16:53]. מכאן היא עוברת לנושא שהפתיע אותה: אימוץ AI מתנהג כמו power law. מי שמאמץ נכון מצליח בגדול, ומי שמאמץ על codebase לא מוכן נכשל בגדול, והקוד שלו מידרדר. לדבריה, נתונים מ-Stanford מראים ש"without structured code base and being ready and documenting well, the AI can make your code worse" [16:53].



"In AI adoption, you either succeed big or fail big": הפער בין leaders ל-laggards גדל פי 12 בשלוש השנים האחרונות וממשיך לגדול, פרקטיקות AI coding מובנות מצמצמות פי 3 את הירידה באיכות הקוד, ויש פער פרודוקטיביות של 30x. לצידם גרפים ממחקר שמיוחס ל-Stanford

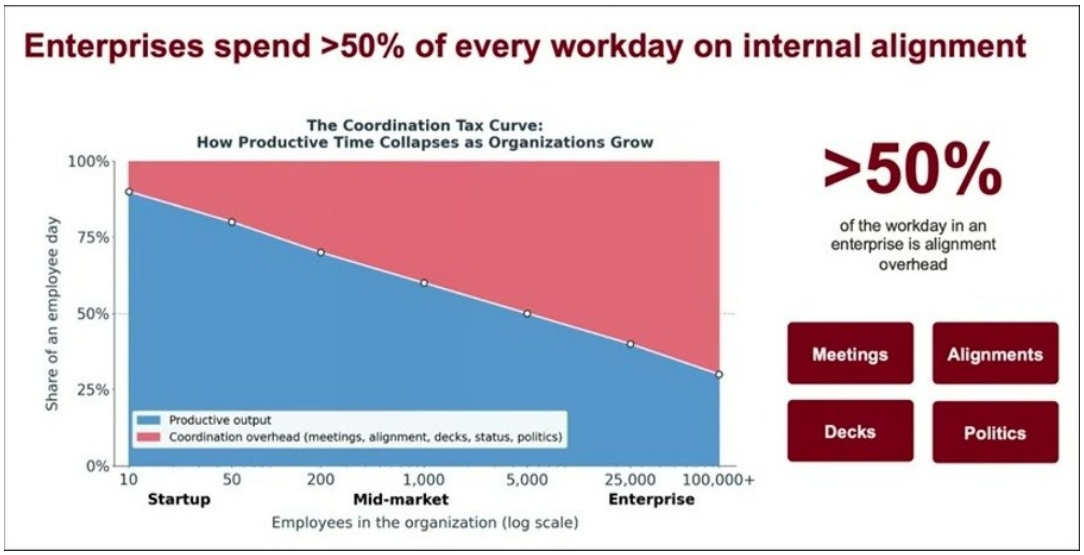
Agent readiness היא בדיקת היגיינה של ה-codebase (היא מסתייגת מהמילה framework). לפי Tížková יש מתאם בין מצב ה-codebase לבין ההצלחה באימוץ AI. הבדיקה מסתכלת על כמה reproducible סביבת הפיתוח, על איכות הטסטים, על התיעוד, על סגנון הקוד ועל linters. לקוחות גדולים עוברים את כל הבדיקות ומקבלים פעולות מומלצות לתיקון [18:25].

הנדבך האחרון הוא ידע שלא כתוב בשום מקום. היא משווה את זה לעובד חדש, שלומד כללים לא כתובים מתצפית. Factory משיקה לשם כך plugins, חבילות של skills ו-context לשימוש חוזר, ו-auto-wiki, שמעדכן את התיעוד אוטומטית [19:55]. agents חסרים את הידע הזה, ולכן חוזרים ומסבירים להם שוב ושוב איך רוצים שדברים ייעשו.

6. מה נשאר לבני אדם

Tížková דוחה במפורש את התרחיש של "permanent underclass". בעיניה זה עוד שלב בסולם ההפשטה: בהתחלה בני אדם היו "human computers", אחר כך הגיעו שפות תכנות, אחר כך coding agents עם אדם צמוד ב-loop, ועכשיו software factories שבהם בני אדם מנהלים צוותים מובנים של agents: orchestrators, workers ו-validators. התפקיד האנושי הוא לנטר ולהחליט מה לבנות [19:55].

> **Tereza Tížková [21:29]** We should be, as humans, deciding what to build in the software, not how — to build it, because that's up for the agents.



"This could be agents instead": לפי השקף, ארגונים מוציאים יותר מ-50% מכל יום עבודה על תיאום פנימי. "The Coordination Tax Curve" מראה איך הזמן הפרודוקטיבי מצטמצם ככל שהארגון גדל, לצד תגיות Meetings, Alignments, Decks ו-Politics

לפי השקף האחרון, AI לא ייקח את העבודה המעניינת. "I believe actually it will take the annoying stuff": ישיבות, סנכרונים, עדכוני סטטוס ואיסוף context מכולם. את כל אלה אפשר להעביר ל-software factory [21:29].

7. מסקנות ופעולות

- **לנסח את תנאי ה-done לפני שמריצים:** לכתוב validation contract, כלומר קריטריונים שאפשר לאמת, לפני שה-agent מתחיל. זו ההגנה העיקרית מפני agent שמרמה את הטסטים.
- **להפריד בין מי שכותב למי שבודק:** validator שלא ראה את הקוד ולא מתקן אותו, רק מצביע על פערים. ומעבר לבדיקה סטטית, לשקול validator שמריץ את האפליקציה ומקליק בה.
- **לנסות workers בסדרה במקום swarm:** context נקי לכל feature, עם מקביליות רק ברמת ה-sub-agents.
- **להפסיק לנתב הכול ל-frontier:** להגדיר מודלי ברירת מחדל לפי תפקיד, להשקיע ב-caching ולשקול routing לפי קושי המשימה. מומלץ למדוד לבד ולא להסתמך על ה-25% של הספק.
- **לטעון tools בעצלות:** אם ה-agents מחוברים לעשרות MCP servers ו-tools, לחשוף רק שם ותיאור קצר, ולטעון schema מלאה לפי דרישה.
- **לבדוק readiness לפני שמאיצים:** סביבת פיתוח reproducible, טסטים, linters ותיעוד. AI על codebase מבולגן מגדיל את הבלגן.
- **לקודד את הידע הלא-כתוב:** להפוך הוראות שחוזרות על עצמן ל-skills או plugins, ולהחזיק תיעוד שמתעדכן אוטומטית.

שאלה פתוחה	למה היא חשובה
מה עולה mission של 16 שעות, בטוקנים ובכסף?	בלי מספר כזה אי אפשר להשוות מול עלות מהנדס או מול פתרון חלופי
על מה נמדד ה-25% של ה-routing, וכמה פעמים ה-router מנתב לא נכון?	ההרצאה עצמה מציגה את "what if the model misroutes" כשאלה פתוחה
איך מונעים cheating גם ב-validation contract עצמו?	ה-orchestrator כותב גם את תנאי ה-done. אם הם שגויים, כל השרשרת נכשלת
מה המחיר של עבודה בסדרה במקום swarm?	context נקי בא על חשבון זמן ריצה. לא הוצגה השוואה
איך מגדירים הרשאות ו-governance ל-agent שרץ שבועות?	הנושא הוזכר כתנאי לאוטונומיה, בלי פירוט
Build או outsource?	השאלה הופיעה באג'נדה ולא קיבלה תשובה ישירה

8. מילון מונחים

מונח	משמעות
Software factory	מחזור חיים אוטונומי מלא של פיתוח תוכנה, מאיסוף סיגנלים ועד אימות ושיפור
Agnostic	בלי תלות בספק מודלים, בסביבה או בדרך העבודה הקיימת של הארגון
Model routing	בחירה אוטומטית של המודל הזול ביותר שצפוי לבצע את המשימה
Caching / prefill	שימוש חוזר בחישוב של context שכבר עובד, במקום לעבד אותו מחדש
Loop / Ralph loop	הרצה חוזרת של agent על משימה עד שתנאי סיום מתקיים
Mission	שם המוצר של Factory ל-session ארוך של agents, שעות עד שבועות
Orchestrator	ה-agent הראשי, שמפרק את העבודה וכותב את תנאי ה-done
Worker	agent שמבצע חלק מהעבודה ומעביר את התוצר ל-worker הבא בסדרה
Validator	agent שבודק קוד שלא הוא כתב ומחזיר משוב
Validation contract	הגדרת קריטריוני האימות, שנכתבת לפני שנכתב קוד
Scrutiny validator	validator שבודק את הקוד עצמו: linters, types וטסטים
User testing validator	validator שמפעיל את האפליקציה במחשב וירטואלי ומקליק בה
Cheating	agent שעובר את הטסטים בלי לבצע את המשימה האמיתית
Context bloat	context window שמתמלא בהגדרות tools ובמידע שלא נחוץ כרגע
Deferred context engine	מנגנון של Factory שחושף tools בהדרגה וטוען אותם רק כשצריך
Agent readiness	בדיקת היגיינה של ה-codebase לפני אימוץ agents
Plugins / auto-wiki	חבילות skills ו-context לשימוש חוזר, ותיעוד שמתעדכן אוטומטית
Droid	שם ה-agent של Factory
Computer use	היכולת של agent להפעיל ממשק גרפי: להקליק, להקליד ולראות מסך