

Vibe coding בפרודקשן — איך עושים את זה באחריות

חוקר מ-Anthropic מסביר איך מוותרים על קריאת הקוד בלי לוותר על השליטה במוצר:
מגבילים את ה-vibe coding ל-leaf nodes, בונים בדיקות אימות, ומשמים כ-PM של
Claude

Erik Schluntz (Anthropic) · אירוע Code w/ Claude (לפי שקופית הפתיחה) · הועלה מחדש ב-X על ידי Himanshu Kumar ב-21.4.2026, תאריך ההרצאה המקורית לא מצוין

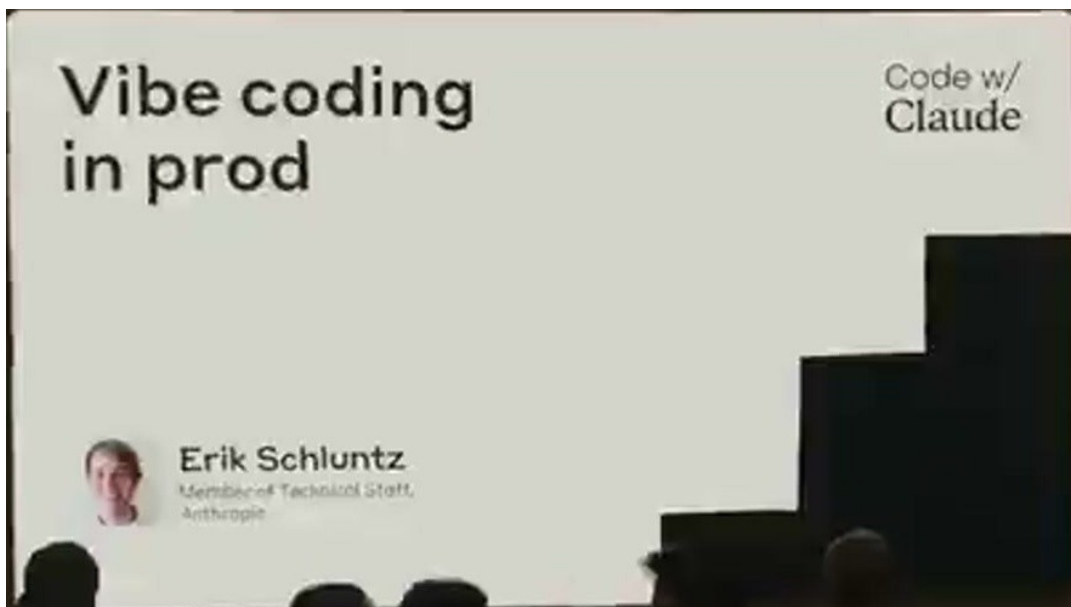
משך: כ-31 דקות · X

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה של כ-15 דקות ואחריה כ-15 דקות של שאלות מהקהל, בשם "Vibe coding in prod", מאירוע Code w/ Claude של Anthropic. הדובר מציג את עצמו כ-Eric, חוקר ב-Anthropic שעוסק ב-coding agents ושותף לכתיבת Building Effective Agents. השקופית הראשונה מזהה אותו כ-Erik Schluntz. הכיתוב של הפוסט ב-X, שלפיו הדובר הוא "Head of Claude Code" ש"הוציא 49 פיצ'רים ביומיים", לא מופיע בשום מקום בהרצאה. מה שהוא כן מספר: כשהיד שלו הייתה בגבס במשך חודשיים, Claude כתב את כל הקוד שלו.

הטענה המרכזית: בגלל הקצב האקספוננציאלי שבו המודלים משתפרים, מהנדסים יצטרכו להפסיק לקרוא כל שורת קוד, כמו שהפסקנו לקרוא את ה-assembly שהקומפיילר מייצר. איך עושים את זה באחריות? שוכחים שהקוד קיים, אבל לא שוכחים שהמוצר קיים. מוצאים שכבת הפשטה שאפשר לאמת בלי להכיר את המימוש, מגבילים את ה-vibe coding ל-leaf nodes שאף רכיב לא תלוי בהם, ומשקיעים בהכנת context ל-Claude כמו PM שמכין מסמך לעובד חדש. ההוכחה שהוא מביא: PR של כ-22 אלף שורות שנכתב ברובו על ידי Claude ומוזג ל-codebase של RL בפרודקשן.



שקופית הפתיחה: "Vibe coding in prod" עם הלוגו Code w/ Claude והשם Erik Schluntz מ-Anthropic

איך לקרוא את המסמך

- 1. **תקציר מנהלים:** הטענות המרכזיות עם חותמות זמן, ומה לא נאמר בהרצאה.
- 2. **מהו vibe coding ולמה זה חשוב:** ההגדרה של Karpathy, הדוגמאות שהשתבשו, והגרף של METR.
- 3. **לשכוח את הקוד, לא את המוצר:** האנלוגיות למנהלים, שכבת ההפשטה שאפשר לאמת, וההסתייגות לגבי tech debt ו-leaf nodes.
- 4. **להיות ה-PM של Claude:** איך נראה תהליך התכנון של 15 עד 20 דקות לפני ההרצאה, ולמי הגישה הזו לא מתאימה.
- 5. **Case study:** ה-PR של 22 אלף שורות ל-codebase של RL, ואיך הצוות וידא שהוא תקין.
- 6. **סיכום ההרצאה ו-Q&A:** ארבעת העקרונות, ותשובות על למידה, אבטחה, compact, TDD ו-Cursor.
- 7. **מה עושים מחר:** צעדים מעשיים וטבלת שאלות פתוחות.
- 8. **מילון מונחים:** המונחים באנגלית שמופיעים במסמך.

הערות מקור המסמך הזה הופק אוטומטית מהתמלול ומהשקופיות של ההקלטה. ייתכן ששמות ומונחים שובשו בתמלול האוטומטי (ASR). למשל, "Barry Zang" הוא כנראה Barry Zhang, ו-"Andre Carpathie" הוא Andrej Karpathy. כפי שמופיע בשקופית. הציטוטים מועתקים מילה במילה מהתמלול ונבדקו מול חותמת הזמן שלהם. לא נוסף מידע ממקורות חיצוניים, וגם הכיתוב של הפוסט ב-X לא שימש כמקור.

1. תקציר מנהלים

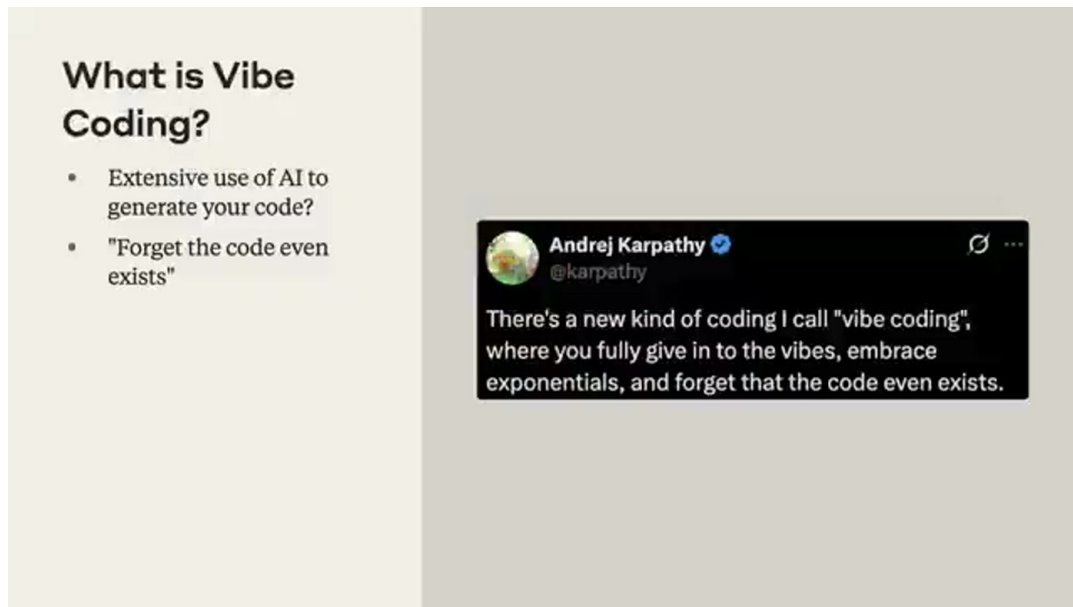
ההרצאה היא בעיקרה טיעון שיטתי: vibe coding הוא כבר לא רק צעצוע לפרויקטים צדדיים. בקרוב זו תהיה מיומנות הכרחית, כי המודלים יספקו יחידות עבודה גדולות מכדי שאדם יוכל לקרוא אותן שורה אחרי שורה. הדובר לא מציע לוותר על בקרה. הוא מציע להעביר אותה מקריאת הקוד לאימות של התנהגות המוצר, ולתחום את הסיכון לחלקים במערכת שאף רכיב לא תלוי בהם.

- **Vibe coding זה לא "להשתמש הרבה ב-AI"**: מי שעובד עם Cursor או Copilot בלולאת משוב צפופה לא עושה vibe coding. לפי ההגדרה של Karpathy, המהות היא "forget that the code even exists" [0:00, 1:31].
- **הסיבה לטפל בזה עכשיו היא הקצב האקספוננציאלי**: "The length of tasks that AI can do is doubling" every seven months. כרגע מדובר בערך בשעה של עבודה, ובעוד שנה-שנתיים זה יגיע ליום או לשבוע [3:03].
- **העיקרון המנחה**: "we will forget that the code exists, but not that the product exists". צריך שכבת הפשטה שאפשר לאמת, כמו CTO שכותב acceptance tests או PM שמשתמש במוצר [4:34, 6:09].
- **ההסתייגות היא tech debt**: היום אין דרך למדוד tech debt בלי לקרוא את הקוד, ולכן ה-vibe coding מוגבל ל-leaf nodes. הליבה הארכיטקטונית נשארת באחריות של מהנדסים [6:09, 7:44].
- **להיות ה-PM של Claude**: הדובר משקיע 15 עד 20 דקות בשיחה נפרדת עם Claude כדי לחקור את הקוד ולבנות תוכנית, ורק אחר כך מריץ אותה. לדבריו, אחוז ההצלחה אז "very, very high" [10:50].
- **זה לא מתאים לכולם**: "I don't think that vibe coding and prod is for everybody". מי שאין לו רקע טכני לא יודע לשאול את השאלות הנכונות [10:50].
- **Case study: PR** של +22,160 שורות ל-codebase של RL בפרודקשן. הושקעו ימים של עבודה אנושית, השינויים התרכזו ב-leaf nodes, נבנו stress tests, וה-outputs תוכננו כך שאדם יוכל לבדוק אותם. התוצאה: יום עבודה במקום שבועיים [12:23, 13:55].
- **טיפים מעשיים מה-Q&A**: לבקש שלושה end-to-end tests ולקרוא קודם את הבדיקות [23:05, 24:37]. לעשות compact כשמגיעים לנקודת עצירה טבעית [27:40]. להתחיל עם Claude Code ולסיים תיקונים נקודתיים ב-Cursor [29:15].
- **אבטחה**: התשובה שלו היא שמי שמכיר את ה-context יודע מה מסוכן. במקרה הפנימי שהציג, המערכת הייתה "fully offline" [20:00].

מה לא נאמר

- **הכיתוב ב-X לא נתמך בהרצאה**: אין בה "Head of Claude Code", אין "49 פיצ'רים ביומיים" ואין "לא כתב קוד ביד כבר חודשים". הדובר מציג את עצמו כחוקר ב-Anthropic.
- **אין מספרים על התהליך**: "very, very high success rate" נאמר בלי נתון. לא נאמר כמה מתוך 22 אלף השורות עבר review אנושי, ואין פרטים על ה-stress tests עצמם.
- **חוסר עקביות קטן בהערכת החיסכון**: הדובר אומר גם "saved us a week's worth of human time" וגם "one day of time instead of two weeks" [13:55].
- **אין פתרון ל-tech debt**: הוא מודה שאין כיום דרך לאמת tech debt בלי לקרוא את הקוד, ולא מציע כלי או מדד.
- **שאלה שלא נענתה**: על הרצה של כמה Claude Code במקביל, Git worktrees ו-stacked PRs הוא לא ענה בכלל. הוא ענה רק על Cursor ועל היכרות עם codebase חדש [29:15].
- **אין מתכון אבטחה כללי**: דוגמת האבטחה שלו נשענת על כך שהמערכת offline, וזה לא מתאים לרוב מוצרי הפרודקשן.

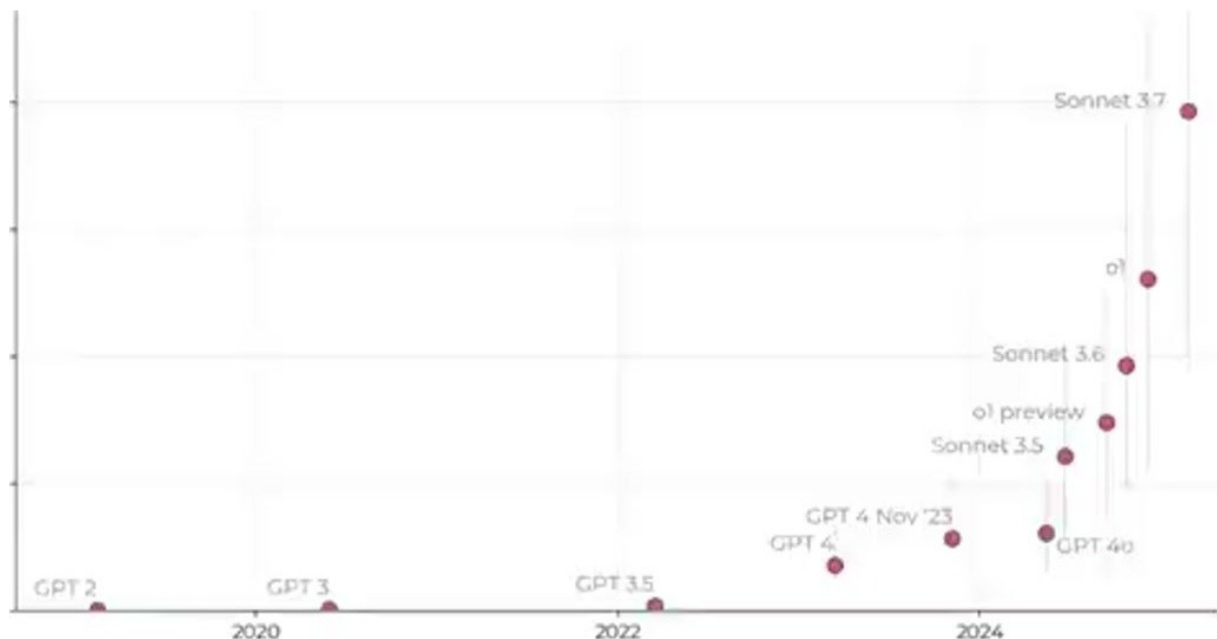
2. מהו vibe coding ולמה זה חשוב



השקופית "What is Vibe Coding?" עם הציוץ של Andrej Karpathy: "fully give in to the vibes, embrace exponentials, and forget that the code even exists"

Schluntz מתחיל בהבחנה בין שימוש אינטנסיבי ב-AI לבין vibe coding. מי שעובד עם Cursor או Copilot ומקבל ממנו רוב הקוד, אבל נשאר בלולאת משוב צפופה, לא עושה לדבריו vibe coding. החלק שחשוב בהגדרה של Karpathy הוא "[1:31] 'forget the code even exists'. בגלל זה vibe coding הלהיב גם אנשים שלא מגיעים מהנדסה, שפתאום יכלו לבנות אפליקציה שלמה לבד.

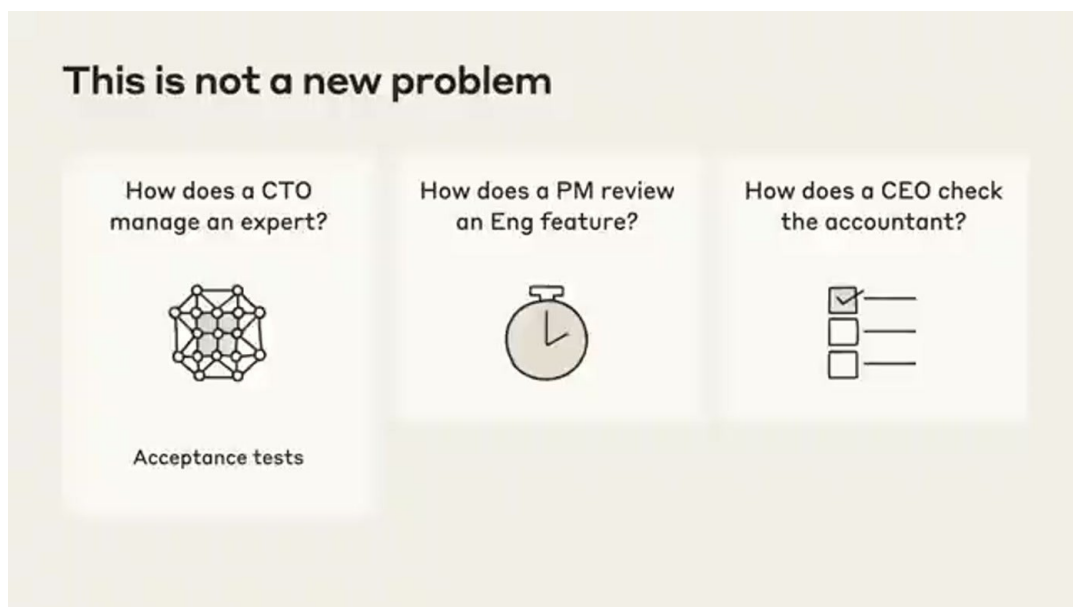
בשקופיות מופיעים גם הצד השלילי וגם הצד החיובי. מצד אחד, ציוץ של מפתח שבנה SaaS עם Cursor ונפרץ, והתמלול מתאר את מה שקרה: "maxed out usage on my API keys", עקיפה של המנוי ויצירת רשומות אקראיות ב-DB [1:31]. מצד שני, ההצלחות היו בעיקר בתחומים שבהם הסיכון נמוך, כמו המשחק שנבנה ב-vibe coding ומכניס 52,000 דולר בחודש לפי השקופית. השאלה שהוא מעלה: למה בכלל להתעסק בזה אם ההצלחות הן צעצועים? התשובה שלו היא הקצב האקספוננציאלי. כשמודל יספק שבוע עבודה בבת אחת, "there is no way that we're going to be able to keep up with that if we still need to move in lockstep" [3:03] זה לקומפיילרים: בהתחלה מפתחים קראו את ה-assembly שהקומפיילר הוציא, ובשלב מסוים זה פשוט הפסיק להיות אפשרי.



גרף של METR: "The length of tasks AIs can do is doubling every 7 months". אורך המשימה בהצלחה של 50%, לפי תאריך שחרור המודל, מ-GPT-2 ועד Sonnet 3.7 שמגיע לכשעה

Erik Schluntz [3:03] The length of tasks that AI can do is doubling every seven months. right now — we're at about an hour

3. לשכוח את הקוד, לא את המוצר

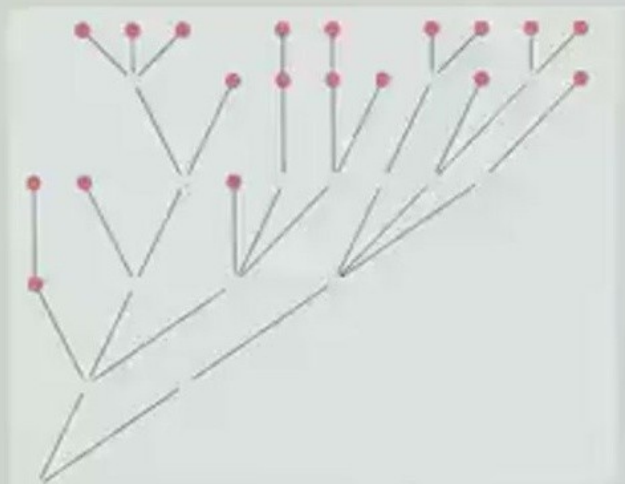


השקופית "This is not a new problem" עם שלוש שאלות: איך CTO מנהל מומחה (acceptance tests), איך PM בודק פיצ'ר הנדסי (משתמש במוצר), ואיך CEO בודק את רואה החשבון (בדיקה מדגמית של עובדות)

התשובה שלו לשאלה "how will we Vibe Code in prod" היא "we will forget that the code exists, but not that the product exists" [4:34]. הוא מדגיש שזו לא בעיה חדשה. CTO כותב acceptance tests למומחה שעובד אצלו גם בלי להבין את המימוש. PM משתמש במוצר כדי לוודא שהוא מתנהג כמו שציפה. CEO בודק מדגמית את העובדות שהוא כן מבין, וכך בונה אמון במודל הפיננסי כולו. בלשונו, "managing implementations that you yourself don't understand is actually a problem as old as civilization" [6:09]. כל מנהל כבר מתמודד עם זה, ומהנדסי תוכנה פשוט רגילים להבין את כל ה-stack עד הסוף. המסקנה המעשית, שמופיעה גם בשקופית: "Find an abstraction layer you can verify".

Caveat: Tech Debt

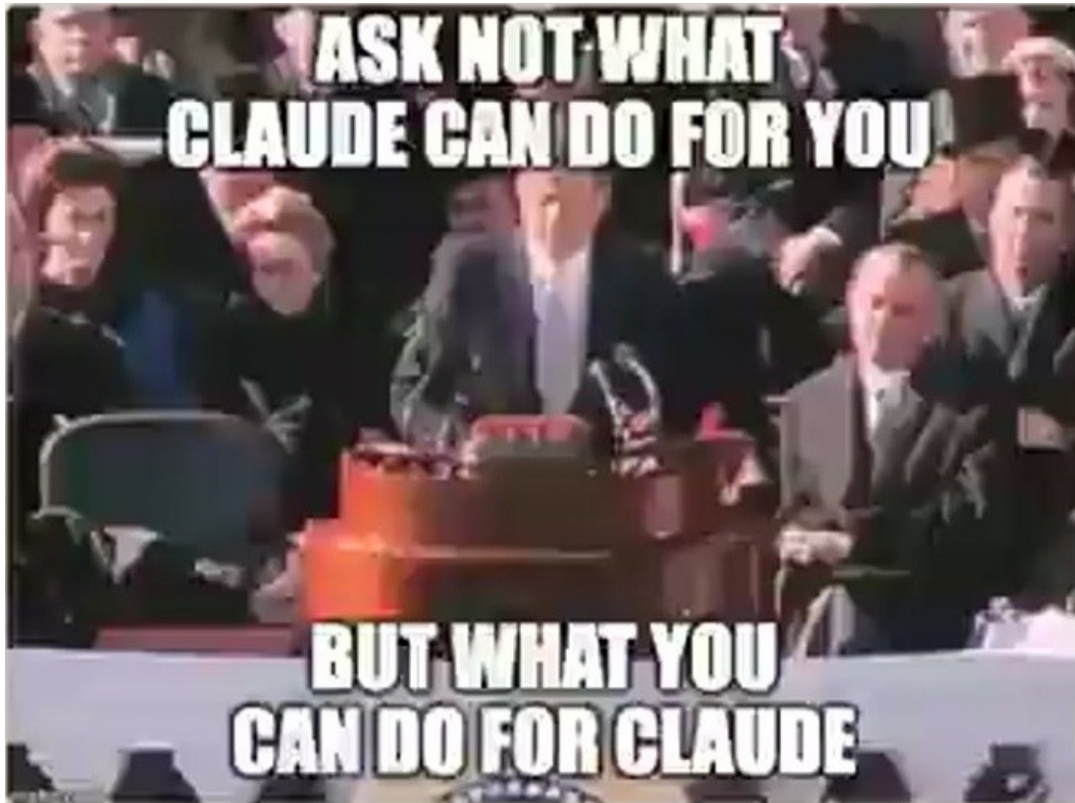
- Extensibility cannot be verified
- Focus on "Leaf Nodes"
- **Core Architecture** must still be understood
- But... the models are getting better all the time...



השקופית "Caveat: Tech Debt": עץ שבו הענפים והגזע לבנים והעלים (leaf nodes) כתומים. הנקודות: Extensibility cannot be verified, Focus on Leaf Nodes, Core Architecture must still be understood, But the models are getting better all the time

ההסתייגות היחידה שלו היא tech debt: "right now there is not a good way to measure or validate tech debt without reading the code yourself" [6:09], leaf nodes- ל vibe coding את ה- tech debt, הנזק מוגבל, כי לא סביר שהם ישתנו או שייבנו עליהם דברים נוספים. הענפים והגזע הם הארכיטקטורה שמהנדסים עדיין צריכים להבין לעומק ולשמור עליה גמישה וניתנת להרחבה [7:44]. לדבריו הגבול הזה זז: בשבוע-שבועיים האחרונים בתוך Anthropic, עם מודלי Claude 4, הוא נתן להם הרבה יותר אמון ממה שנתן ל-3.7 [9:19].

4. להיות ה-PM של Claude



השקופית "How to succeed at Vibe Coding": "Think like a PM: What guidance or context would a new employee need?" לצד
נאם "Ask not what Claude can do for you, but what you can do for Claude"

העצה המרכזית שלו: "[9:19] ask not what Claude can do for you, but what you can do for Claude". מי שעושה vibe coding משמש בפועל כ-PM של Claude, ולכן צריך לשאול איזה context עובד חדש היה צריך כדי להצליח: סיור ב-codebase, דרישות, אילוצים. אף אחד לא מצפה שעובד ביום הראשון שלו יממש פיצ'ר על סמך משפט אחד, וככה גם לא כדאי לעבוד עם Claude.

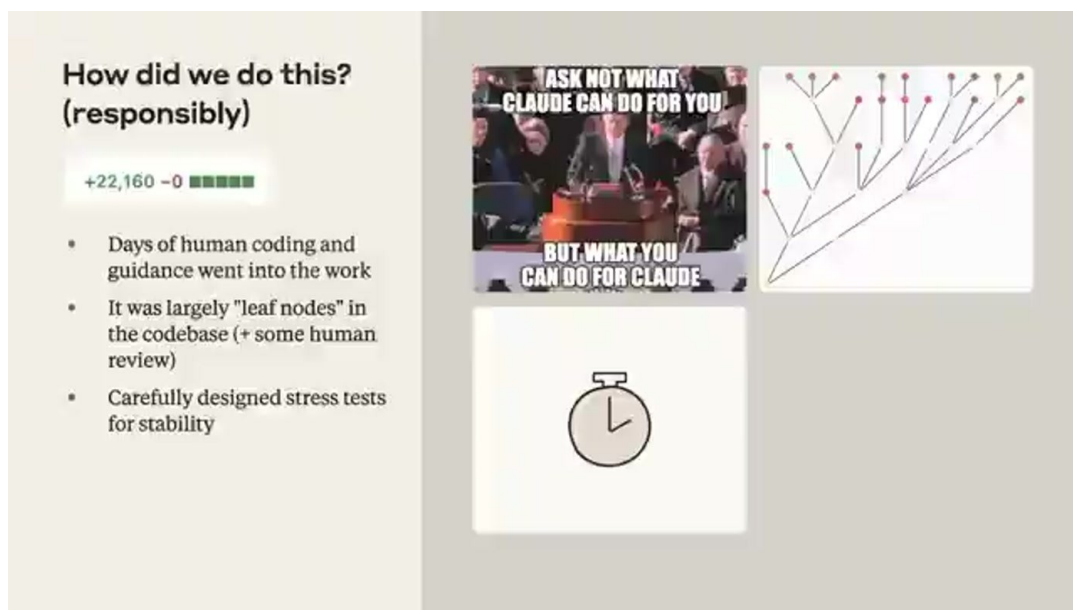
בפועל, 15 עד 20 הדקות האלה הן לא כתיבה ידנית של prompt. זו שיחה נפרדת עם Claude, שבה הוא חוקר את הקוד, מאתר את הקבצים הרלוונטיים, ושניהם בונים יחד תוכנית: מה הוא רוצה, אילו קבצים ישתנו ואילו דפוסים צריך לשמור. את התוצר הזה הוא מעביר ל-context חדש, או פשוט אומר "[10:50] let's go execute this plan". כמה פירוט לתת? זה תלוי במה שחשוב לו. כשהימוש לא משנה לו, הוא מתאר רק דרישות. כשהוא מכיר את הקוד, הוא מציין מחלקות ומפנה לפיצ'ר דומה. הוא מזהיר ש-"our models do best when you don't over-constrain them" [18:29]. ויש גבול ברור: אנשים בלי רקע טכני לא צריכים לבנות עסק מאפס בדרך הזו, כי הם לא יודעים לשאול את השאלות הנכונות [10:50].

Erik Schluntz [10:50] When I'm working on features with Claude, I often spend 15 or 20 minutes — collecting guidance into a single prompt and then let Claude cook after that

5. PR Case study: של 22 אלף שורות ל-codebase של RL



השקופית "Case Study: We merged a 22k line change to our RL codebase, written heavily by Claude" עם ה-diff מ-GitHub: +22,160 -0



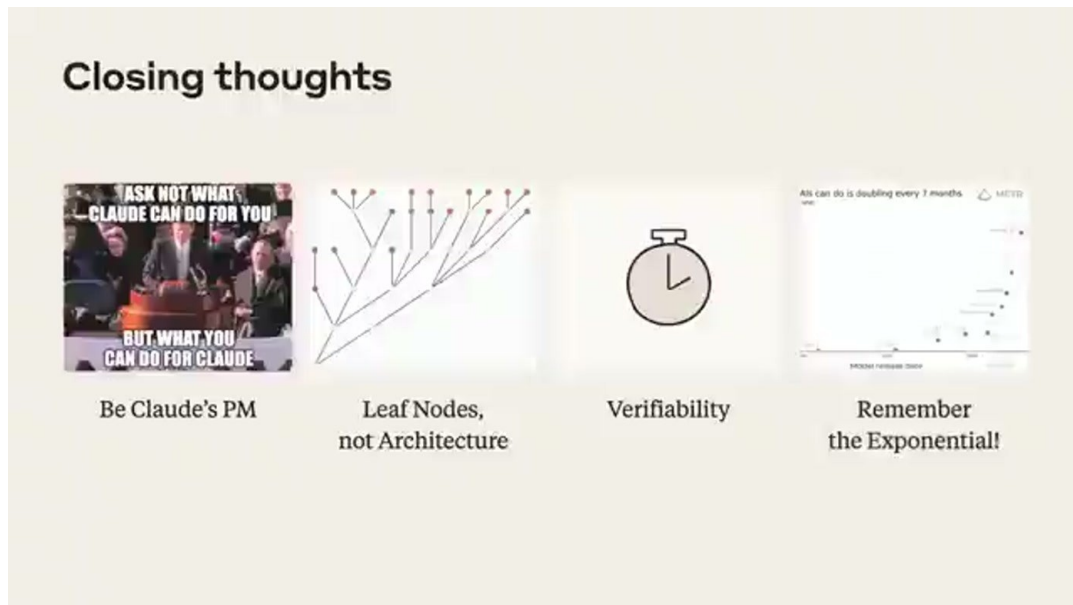
השקופית "How did we do this? (responsibly)": ימים של קוד והכוונה אנושיים, ריכוז השינוי ב-leaf nodes עם קצת review אנושי, ו-stress tests שתוכננו בקפידה

"We recently merged a 22,000 line change to our production reinforcement learning code base" [10:50]. לדבריו, ה-screenshot הוא ה-diff האמיתי מ-GitHub. הוא מפרט ארבעה דברים שאפשרו את זה. ראשון, המערכת [12:23]. שני, השינוי התרכז ב-leaf nodes, והחלקים שצריכים להישאר ניתנים להרחבה עברו review אנושי כבד. שלישי, נבנו stress tests ליציבות שרצו לאורך זמן, כי יציבות הייתה הדאגה הגדולה ביותר. רביעי, המערכת תוכננה מראש עם inputs ו-outputs שאדם יכול לבדוק בקלות, כך שאפשר היה לאמת שהיא נכונה בלי לקרוא את כל המימוש.

בשקופית הבאה (k_043) מופיעה השורה התחתונה: "we were just as confident in this change as any other,". לדבריו, הרווח החשוב יותר הוא שינוי באופן החשיבה. כשהעלות השולית של תוכנה יורדת, משימות של שבועיים הופכות למשהו שאפשר פשוט לעשות. שימו לב: הוא גם מזכיר שהמערכת הזו [20:00] "fully offline", וזה חלק מהסיבה שהסיכון היה נמוך.

Erik Schluntz [13:55] And now suddenly, when something costs one day of time instead of two — weeks, you realize that you can go and make much bigger features and much bigger changes

6. סיכום ההרצאה ו-Q&A



השקופית "Closing thoughts" עם ארבעה עקרונות: *Be Claude's PM*, *Leaf Nodes not Architecture*, *Verifiability*, *Remember the Exponential*

ארבעת העקרונות שבהם הוא סוגר: להיות ה-PM של Claude, לעשות vibe coding ב-leaf nodes ולא בארכיטקטורה, לחשוב איך לאמת שהשינוי נכון בלי לקרוא את הקוד, ולזכור את הקצב האקספוננציאלי. היום עוד אפשר לא לעשות vibe coding, אבל בעוד שנה-שנתיים מי שמתעקש לקרוא או לכתוב כל שורה יישאר מאחור: "you are going to become the bottleneck if we don't get good at this" [15:26]. אחרי זה הגיעו כ-15 דקות של שאלות, ובהן הפרטים המעשיים ביותר בהרצאה.

- **איך לומדים בלי להתאמץ על syntax:** לדבריו אפשר ללמוד מהר יותר, למשל לשאול את Claude למה בחר בספרייה מסוימת. "people that are lazy are not going to learn". מחזור של החלטה ארכיטקטונית יכול להתקצר משנתיים לחצי שנה, כלומר פי ארבעה יותר לקחים באותו זמן [18:29, 16:58].
- **אבטחה:** מי שמבין את ה-context יודע מה מסוכן. המקרה הפנימי היה offline. הוא היה רוצה לראות "frameworks" provably correct, שבהם auth ותשלומים כבר בנויים ורק ה-UI נכתב ב-vibe coding, כמו Claude Artifacts שמוגבל ל-[20:00, 21:33].
- **TDD:** Claude נטה לכתוב בדיקות שתלויות מדי במימוש. הוא מבקש במפורש "just write three end-to-end tests": happy path ושני מקרי שגיאה. לפעמים הבדיקות הן החלק הראשון, או היחיד, שהוא קורא [24:37, 23:05].
- **Embrace exponentials:** הכוונה היא לא רק שהמודלים ישתפרו, אלא שהם ישתפרו מהר יותר ממה שאנחנו מסוגלים לדמיין. הוא מצטט את "machines of loving grace is not science fiction. It's a product roadmap" מהשיחה של Dario ו-Mike Krieger [26:08].
- **Compact:** הוא עושה compact או פותח session חדש בנקודת עצירה טבעית, כמו שמתכנת יוצא להפסקת צהריים. קודם הוא מבקש מ-Claude לכתוב את התוכנית למסמך, ואז "gets rid of 100k tokens" ונשארים כמה אלפי [27:40].
- **Cursor מול Claude Code:** "I'll start things with Claude code and then I'll use Cursor to fix things" הוא קודם מבקש מ-Claude Code להראות לו איפה דברים קורים ואילו מחלקות לבדוק, קט". בקוד שהוא לא מכיר, הוא קודם מבקש מ-Claude Code להראות לו איפה דברים קורים ואילו מחלקות לבדוק, כדי לבנות לעצמו תמונה בראש [30:48, 29:15].

7. מה עושים מחר

- **למפות את ה-codebase:** לסמן אילו רכיבים הם leaf nodes שאפשר לבנות ב-vibe coding, ואילו הם גזע שדורש review מלא.
- **להכניס שלב תכנון נפרד:** להקדיש 15 עד 20 דקות לחקירה עם Claude ולתוכנית כתובה, ורק אז להריץ ב-context חדש.
- **לתכנן מראש איך יאומת השינוי:** inputs ו-outputs שאדם יכול לבדוק, ו-stress tests ליציבות, עוד לפני שמבקשים מ-Claude לכתוב קוד.
- **לבקש בדיקות מינימליסטיות:** שלושה end-to-end tests ולקרוא אותם ראשונים. אם הבדיקות נראות נכונות ועוברות, זו נקודת אמון טובה.
- **לנהל את ה-context:** לעשות compact בנקודות עצירה טבעיות, אחרי שהתוכנית נשמרה במסמך.
- **לא לעשות vibe coding במקומות רגישים:** auth, תשלומים ורכיבים שחשופים לאינטרנט, אלא אם יש מישהו שיודע לשאול את שאלות האבטחה הנכונות.

שאלה פתוחה	מה נאמר בהרצאה
איך מודדים tech debt בקוד שאף אחד לא קרא?	לא נאמר. הדובר מודה שאין כרגע דרך לעשות את זה.
מה היה היחס בין review אנושי לקוד שלא נבדק ב-PR של 22 אלף השורות?	נאמר רק "heavy human review" על החלקים שצריכים להרחבה. אין מספרים.
האם הגישה מתאימה גם למערכות online שחשופות למשתמשים?	הדוגמה הייתה offline. לגבי השאר נאמר רק שצריך להבין את ה-context.
איך עובדים עם כמה agents במקביל, worktrees ו-stacked PRs?	השאלה נשאלה ולא נענתה [29:15].
איפה עובר הגבול בין leaf node לגזע כשהמודלים משתפרים?	הגבול זז עם Claude 4, אבל לא הוצג קריטריון [9:19].

8. מילון מונחים

מונח	משמעות
Vibe coding	פיתוח שבו נותנים למודל לכתוב את הקוד ו"שוכחים שהקוד קיים", לפי ההגדרה של Karpathy
Leaf nodes	רכיבים בקצה המערכת שאף רכיב אחר לא תלוי בהם
Core architecture	הגזע והענפים של המערכת, שרכיבים אחרים נבנים עליהם
Tech debt	חוב טכני: קוד שקשה להרחיב או לתחזק
Acceptance tests	בדיקות קבלה שבודקות התנהגות בלי להכיר את המימוש
End-to-end tests	בדיקות שבודקות תרחיש שלם מהקלט ועד הפלט
TDD	פיתוח שמתחיל מכתיבת הבדיקות
Stress tests	בדיקות עומס ויציבות שרצות לאורך זמן
Compact	דחיסה של היסטוריית השיחה ב-Claude Code לכמה אלפי tokens
Context	כל המידע שהמודל מקבל לפני שהוא מבצע משימה
RL	Reinforcement learning, למידת חיזוק
PR / diff	בקשת מיזוג ב-GitHub וההבדלים בקוד שהיא מכילה
METR	הגוף שפרסם את הגרף על אורך המשימות שמודלים מסוגלים לבצע (לפי השקופית)
Claude Artifacts	קוד front-end ש-Claude כותב ומוצג ישירות בתוך Claude AI
Git worktrees	כמה תיקיות עבודה של אותו repository, מאפשרות לעבוד בכמה ענפים במקביל