# כל הקוד הוא slop (חוץ משלי)

איך צוות Grok Bot הגיע ל-11,000 PRs בחודש עם שיעור revert של כחצי אחוז:
ארכיטקטורה, lint rules ואימות במקום עוד prompts

Lauren Tan (poteto) · AI Engineer · 14.11.2026

משך: כ-18 דקות · YouTube

סיכום אוטומטי — AI Briefing (aibriefing.dev)

# על המסמך הזה

זו הרצאה קצרה של כ-18 דקות מ-AI Engineer Code Summit. כך כתוב על המסך, והכנס מוצג בחסות Google DeepMind. המרצה היא Lauren Tan, שמוכרת בכינוי poteto (בתמלול יוצא Potato). היא Principal Engineer ב-SpaceX, ובתיאור הסרטון החברה מופיעה כ-SpaceXSI. Tan עובדת על Grok Bot, אפליקציית Electron שמשמשת ממשק להרצה ולתזמור של agents. חלק מהדוגמאות מגיעות מ-Cursor: היא מתארת בעיות שהם ראו שם, וכנראה שם עבדה קודם, אבל היא לא אומרת את זה במפורש.

הטענה המרכזית: התפקיד של מהנדס תוכנה היום הוא לשמור על איכות הקוד, ואפילו לשפר אותה, תוך הגדלה דרמטית של התפוקה. הדרך לשם לא עוברת דרך עוד הוראות ב-agents.md. היא עוברת דרך השקעה בסביבה של ה-agent: קוד נקי שה-agent מעתיק ממנו, ארכיטקטורה ו-lint rules שאוכפים את הדרך הנכונה, ושכבות של אימות runtime ואימות פורמלי. התוצאה שהיא מציגה היא 11,000 PRs לפרודקשן בחודש, שיעור revert של כחצי אחוז וביצועים שהשתפרו.




*שקופית הפתיחה על הבמה: הכותרת all code is slop עם כיתוב משנה בסוגריים והשם poteto aka Lauren Tan, על רקע המיתוג של AI Engineer Code Summit*

### איך לקרוא את המסמך

- **1. תקציר מנהלים:** הטענות המרכזיות, הראיות לכל אחת ומה חסר בהרצאה.
- **2. פירמידת האמון:** ארבע שכבות ההשקעה בסביבה של ה-agent ולמה הקוד עצמו הוא הקלט החשוב ביותר.
- **3. Dune וארכיטקטורה שנאכפת:** איסור useEffect והפרדה מבנית בין main ל-renderer.
- **4. הערות מול lint rules:** למה נאסרו 99% מההערות בקוד ומה בא במקומן.
- **5. אימות:** Lean, TLA+, feature map, runtime verification ו-property-based testing ל-UI, ולמה רוב הטוקנים הולכים לשם.
- **6. התוצאות:** המספרים של צוות Grok Bot.
- **7–8. פעולות ומילון:** מה לעשות מחר, שאלות פתוחות ומונחים.

**הערת מקור** המסמך הופק אוטומטית מהתמלול של ההקלטה ומהשקופיות שחולצו ממנה. התמלול האוטומטי (ASR) משבש שמות ומונחים. שם המוצר, למשל, מופיע כ-GrokBot, GraphBot, GroffBot, Grotbot ו-GrowlitheBot. כאן נבחר Grok Bot לפי תיאור הסרטון. הציטוטים מועתקים מילה במילה מהתמלול ונבדקו מול חותמת הזמן שלהם. לא נוסף מידע ממקורות חיצוניים. מההרצאה חולצו רק שש תמונות, וחמש מהן מאותה שקופית הדגמה.

# 1. תקציר מנהלים

Tan טוענת שאת הקפיצה בתפוקה ש-agents מאפשרים אפשר לנצל בלי לטבוע ב-slop, בתנאי שמפסיקים לתקן את ה-agent במילים ומתחילים לתקן אותו בקוד. כל תיקון הופך לבדיקה דטרמיניסטית, כל anti-pattern נחסם ארכיטקטונית, והאימות תופס את מה שנשאר. בסוף התהליך כמעט לא נשאר מה לבדוק ב-code review, וה-agents ממזגים את רוב העבודה בעצמם.

- **אמון נבנה בסביבה ולא ב-prompt:** פירמידה הפוכה. בבסיס שתי שכבות דטרמיניסטיות (קוד וארכיטקטורה; lint, טיפוסים ובדיקות), מעליהן אימות, ובראש שכבה דקה של פרוזה (rules, skills, agents.md).
- **הקוד הוא הקלט הגדול ביותר של ה-agent:** כל workaround, קוד משוכפל או tech debt ייראה ל-agent הבא כמו הדפוס המקובל, והוא יעתיק אותו. לכן בסיס הקוד צריך להיות נקי ככל האפשר.
- **לאסור במקום להסביר:** ב-Dune נאסר useEffect בקוד מוצר, והייבוא בין electronRenderer ל-electronMain חסום על ידי dependency graph analyzer. הבעיות האלה פשוט לא קיימות יותר.
- **הערות הן פתח מילוט:** ב-Cursor ה-agents כתבו הערות שהצדיקו timers אימפרטיביים כתיקון מהיר, וה-timers התפשטו בקוד. ב-Grok Bot נאסרו 99% מההערות, ובמקומן יש lint rules עם הודעות שגיאה שמכוונות לשורש הבעיה.
- **כל תיקון של ה-agent הופך לבדיקה דטרמיניסטית:** אחרי מספיק סבבים לא נשאר מה לבדוק, וה-agents ממזגים את רוב ה-PRs שלהם בעצמם.
- **code review הוחלף בבדיקת ראיות אימות:** סרטונים, צילומי מסך, דיאגרמות ואימות פורמלי. TLA+ משמש ל-model checking של זרימות, Lean להוכחת אלגוריתמים ו-Bombadil ל-property-based testing של UI.,
- **רוב הטוקנים הולכים לאימות:** Cindy (שם פרטי בלבד) מדווחת ש-70% מהטוקנים היומיים שלה הולכים לאימות, ו-Tan אומרת שאצלה התמונה דומה.,
- **המספרים:** 11,000 PRs לפרודקשן בחודש אחד, גרסה יציבה כמעט כל יום, כ-227 תורמים באוקטובר, revert יציב סביב 0.5% והאפליקציה נעשתה מהירה יותר.,
- **קודם יסודות ורק אחר כך factory:** לולאה פנימית שמייצרת slop תייצר עוד slop כשממכנים אותה. אצלם כל release מפעיל fuzzing אוטומטי שפותח PRs לתיקון וממזג אותם אוטומטית.

### מה לא נאמר

- **אין baseline ל-revert:** נאמר שהשיעור ירד יחסית ועומד על כחצי אחוז, אבל לא נמסר מה היה לפני ה-agents ואיך נמדד revert.
- **אין מספרי ביצועים:** נאמר ש-startup, הקלדה ומעבר בין צ'אטים נעשו מהירים יותר, בלי מדדים ובלי השוואה.
- **אין עלות:** נאמר שזריקת טוקנים על הבעיה יקרה ושרוב הטוקנים הולכים לאימות, אבל אין תקציב, עלות ל-PR או גודל צוות.
- **כמה באמת ממוזג בלי אדם:** נאמר ש-most product code לא דורש review עמוק, בלי אחוז. גם אין נתונים על איכות ה-PRs שה-fuzzer ממזג אוטומטית.
- **היקף האכיפה:** לא נמסר כמה lint rules קיימים, כמה זמן לוקח להוסיף כלל או כמה באגים בדיוק מצאו TLA+, Lean ו-Bombadil (רק quite a few ו-a few).
- **3,000 מול 11,000:** בפתיחה Tan מספרת ששלחה בעצמה 3,000 PRs בחודש שעבר (בתמלול: 3,000 Ploycrest), ובהמשך מציגה 11,000 לכל הצוות. היחס בין הליבה לתורמים מזדמנים לא פורט.

# 2. מ-slop לזהב: אמון ופירמידת האמון

Tan פותחת בהשוואה ל-coding machines, מהנדסים חריגים בתפוקה שלהם. הדוגמה שלה היא Brett Taylor (כך בתמלול; כנראה Bret Taylor), שלפי הסיפור כתב מחדש את Google Maps בסוף שבוע. לדבריה, עם agents כל אחד יכול להיות coding machine. את מה שאחרים קוראים לו software factory היא מעדיפה לתאר במונח אחד: אמון. לא אמון עיוור, אלא "investing in your agents environment so you can trust that your agents are going to do the right thing". לדבריה לקח לה שישה חודשים להגיע למצב שבו היא שולחת אלפי "a lot of PRs, them in my sleep", ולאחרים זה ייקח פחות.

המודל שלה הוא פירמידה הפוכה של השקעה. השכבות הרחבות למטה הן דטרמיניסטיות ומגבילות את ה-agent כך שיעשה את הנכון כברירת מחדל. הפרוזה, שדורשת שיקול דעת, נמצאת בקודקוד הצר.

| שכבה | מה נכלל בה | התפקיד שלה |
|---|---|---|
| קוד וארכיטקטורה (דטרמיניסטי) | API design, ארכיטקטורה, refactoring | להפוך את הדרך הנכונה לדרך הקלה ביותר עבור ה-agent |
| ניתוח סטטי ובדיקות (דטרמיניסטי) | lint rules, type systems, compilers, סוגי בדיקות שונים | לחסום anti-patterns לפני שהם נכנסים |
| אימות | formal-ו runtime verification verification | למצוא באגים, לבדוק invariants ולוודא שה-agent הבין את הכוונה |
| פרוזה | rules, skills, agents.md | רק למה שבאמת דורש שיקול דעת |

> **Lauren Tan** — So whenever you're correcting your agent, think about how you can make that a deterministic check instead of adding another line to your rules or your agents.md because it's far more reproducible when it's in code and it lets you trust it more.

## 3. הקוד כגינה: Dune וארכיטקטורה שנאכפת

הדימוי שלה הוא גינה. ב-2024 היא הגיעה לעבודה ובדקה שלושה PRs שכתבו עמיתים, והיום המספר קרוב ל-300. כל מהנדס הוא עכשיו צוות של מאות או אלפי agents, והקוד מגיע בזרם רציף שמרגיש כמעט אורגני. האילוצים וה-guardrails הם יסודות הבטון שהקוד החדש צומח סביבם. מנגד, workarounds ו-anti-patterns הם עשבים שוטים: ה-agent הבא מעתיק אותם, כי הם נראים כמו הדפוס המקובל, והם מתפשטים מהר.,

את הרעיון הזה הם מממשים ב-Dune, framework צד לקוח שבנו בשביל Grok Bot. Tan מתארת אותו כמשהו כמו Next.js לאפליקציות Electron, ואומרת שטכנית הוא לא מעניין במיוחד. מה שמעניין בו הם האילוצים וה-lint rules שמקודדים לקחים מבניית אפליקציות desktop. הכלל המנחה: כל מה שדורש ניואנס כדי להשתמש בו נכון מוביל בסוף ל-slop (בתמלול נשמע slow).

| Anti-pattern | מה קרה | הפתרון |
|---|---|---|
| useEffect ב-React | כלי שדורש הרבה ניואנס (יש שאומרים שצריך PhD כדי להשתמש בו), ו-agents משתמשים בו לא נכון | נאסר בקוד מוצר. מותר רק במקומות מסוימים בתוך ה-APIs של Dune |
| ייבוא קוד main thread ל-render thread (נצפה ב-Cursor) | long tasks ו-jank | שתי תיקיות עליונות בשם electronRenderer ו-electronMain. ייבוא ביניהן חסום על ידי dependency graph analyzer ו-lint rules |
| timers אימפרטיביים: setTimeout, requestAnimationFrame, requestIdleCallback (ב-Cursor) | טלאי על התנגשות בין הסגנון האימפרטיבי של VS Code לסגנון הדקלרטיבי של React. הרבה garbage collection ו-jank | lint rules עם הודעות שגיאה שמפנות לתיקון שורש הבעיה (פרק 4) |

> **Lauren Tan** — So our solution has been to just ban it in product code. And it's really only allowed in some places inside of Dune APIs. And effectively, this problem has now been categorically eliminated.

## 4. להחליף הערות ב-lint rules

הממצא המפתיע ביותר בהרצאה הוא ש-agents משתמשים ב-markdown, כלומר בתיעוד, בהערות קוד, ב-rules וב-skills, כדי לכתוב הצדקות לכך שהם לא עושים את הדבר הנכון. במקרה אחר הם לוקחים תיקון קטן ומקבעים אותו ככלל שאסור לשבור לעולם. הדוגמה מ-Cursor: ה-agents השתמשו ב-timers כתיקון מהיר, וכתבו לידם הערות שמסבירות למה זה מוצדק. בפועל ה-timers טשטשו בעיה ארכיטקטונית. הם התפשטו בכל הקוד והביצועים נפגעו. Tan מסכמת שבחירת הארכיטקטורה פגעה גם בביצועי הלקוח וגם בביצועי ה-agent.,

ב-Grok Bot הסיקו את המסקנה ואסרו 99% מההערות. ההערות מתיישנות מהר, אנשים ממילא לא קוראים comment slop, והן נותנות ל-agent פתח מילוט כשהדבר הנכון קשה. במקומן יש lint rules עם הודעות שגיאה מועילות שאומרות ל-agent לתקן את שורש הבעיה. העיקרון הכללי: כל תיקון של agent הופך לבדיקה דטרמיניסטית,

או לשינוי ב-API ובארכיטקטורה שמעלים את הבעיה לגמרי. אחרי מספיק סבבים "you'll realize there's nothing left to review", ואפשר לתת ל-agents למזג את רוב ה-PRs שלהם.

> **Lauren Tan** And the worst part about comments though is that they're not enforceable. So what — seems well-intentioned, like I'm going to tell the next agent not to do something, never ends up actually being anything that you can trust.

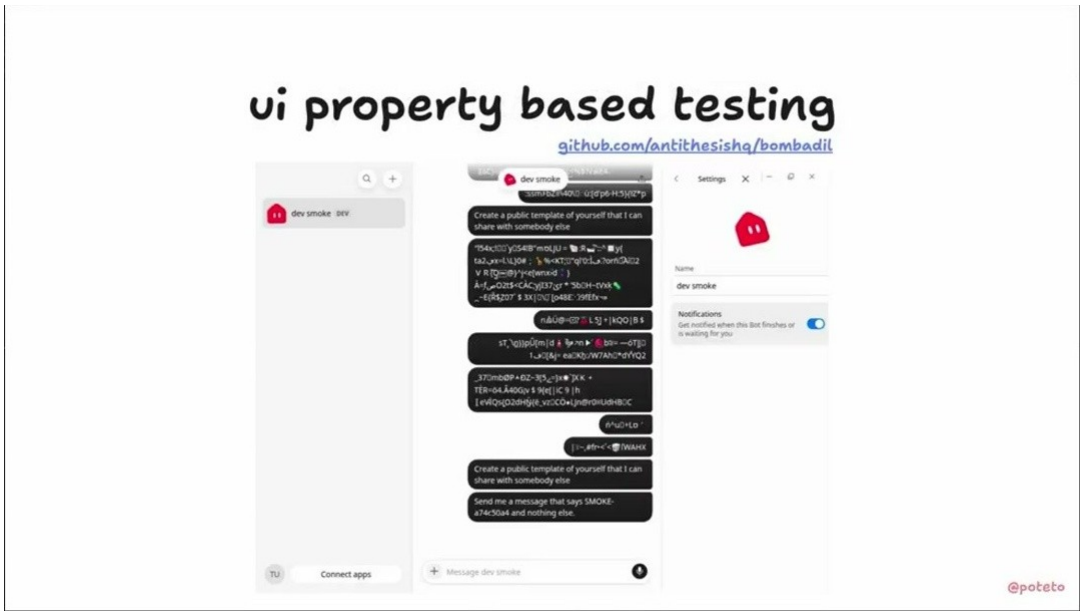
## 5. אימות: runtime, פורמלי ו-UI

לדבריה, code review אצלם הצטמצם לבדיקה של ראיות אימות: סרטונים, צילומי מסך, דיאגרמות ותוצרי אימות פורמלי. הבסיס הוא runtime verification, כלומר לתת ל-agent להריץ את הקוד בעצמו ולהשתמש באותם כלי debugging של מפתחים, למשל Chrome DevTools Protocol. ה-skills של runtime verification כוללים feature map, מעין sitemap של כל הפיצ'רים באפליקציה מנקודת המבט של המשתמש. כך ה-agent לא צריך לנחש למה הכוונה ב-"the thingy in the left sidebar", ונוצרת שפה משותפת. בתוך ה-skills יש גם CLIs ייעודיים שה-agent מריץ שוב ושוב עם אותו driver. זה חוסך טוקנים, ואת ה-CLI עצמו אפשר לבדוק.,




*שקופית ui property based testing עם קישור GitHub ל-Bombadil: הקלטה מואצת של אפליקציית צ'אט שמופעלת אוטומטית, עם רשימת הודעות וסרגל צד*

השכבה הבאה היא אימות פורמלי. ב-TLA+ הם עושים model checking לזרימות משתמש מרכזיות, למשל התראות על פני כמה מכשירים ופלטפורמות. ב-Lean הם מוכיחים אלגוריתמים, למשל בספריית virtualization שכתבו בעצמם, כדי לוודא שאין flickering או jittering. לדבריה כבר נמצאו כך לא מעט באגים, אבל "it's certainly not a silver bullet and we're treating it as another tool, another layer in our agents environment". לאחרונה הם התחילו להשתמש גם ב-property-based testing ל-UI עם ספרייה בשם Bombadil, שכבר תפסה כמה באגים. ההדגמה הוצגה כסרטון מואץ, ואלה פריימים ממנו.




*פריים נוסף מהדגמת Bombadil: מסך הצ'אט עם פאנל הגדרות פתוח בצד, כולל מתג (toggle). ה-property-based testing עובר בין מצבי ממשק שונים*




*פריים מהדגמת Bombadil: חלון בסגנון command palette עם רשימת פריטים מעל שכבת רקע אפורה, אחד המצבים שהבדיקה מגיעה אליהם*

כדי להמחיש את ההשפעה Tan מביאה ציוץ של Cindy (שם פרטי בלבד, ללא פרטים נוספים). לפני חצי שנה Cindy כתבה שאינה יכולה לסמוך על ה-agents שלה בגלל באגים עדינים שנכנסו ל-merge. היום, לדבריה, 70% מהטוקנים היומיים שלה הולכים לאימות. Tan אומרת שזה תואם את מה שהיא רואה אצלה. לדבריה אף אחת מהטכניקות לא חדשה. הן פשוט היו יקרות מדי להבנה, לכתיבה ולתחזוקה, ו-agents הורידו את המחיר. "really we're entering a golden age of software testing and verification because agents let us test everything to a much greater degree than used to be economical". היא מזכירה גם skill בשם State Chart בתוסף שלה, pstack (בתמלול potato stack), שמציג state machines מנקודת המבט של המשתמש. לדבריה קל לבדוק אותו לצד הסרטונים וצילומי המסך.




*פריים מהדגמת Bombadil: חלון הגדרות או דיאלוג פתוח מעל האפליקציה. זה סוג ראיית האימות (וידאו של ריצה אוטומטית) שמחליף קריאת diff*

## 6. התוצאות: 11,000 PRs ו-0.5% revert

| מדד | ערך |
|---|---|
| PRs לפרודקשן של צוות Grok Bot בחודש שעבר | 11,000 |
| PRs ש-Tan שלחה בעצמה בחודש שעבר | 3,000 (בתמלול: Ploycrest) |
| קצב גרסאות יציבות של הלקוח | בערך פעם ביום |
| אנשים שתרמו לפחות PR אחד באוקטובר | כ-227 |
| שיעור revert | כ-0.5%, יציב, ונמוך יותר במונחים יחסיים |
| PRs לבדיקה ביום: 2024 מול היום | 3 מול כ-300 |
| זמן ההשקעה עד המצב הנוכחי | כשישה חודשים |
| חלק הטוקנים שהולך לאימות (Cindy) | 70% |

מעבר למספרים, Tan מדגישה שהביצועים של Grok Bot לא רק נשמרו אלא השתפרו מאז ההשקה בכמה ממדים: startup, הקלדה, פתיחת צ'אטים חדשים ומעבר בין צ'אטים. זה למרות שמדובר באפליקציית Electron, שיש לה תקרת ביצועים, ולמרות שתורמים לה גם תורמים מזדמנים בלי הרבה הקשר. לדבריה היא כבר מהירה בהרבה ממשקי agent אחרים, אבל לא הוצגו מדדים. הדוגמה הקיצונית לאוטומציה: בכל release חדש Grok Bot מקים אוטומטית Cursor project שמבצע fuzzing לכל ה-build. הוא מריץ את האפליקציה כמו משתמש אמיתי, פותח PR לתיקון לכל באג שהוא מוצא וממזג אותו אוטומטית. לדבריה זה אפשרי כי בסיס הקוד מספיק טוב כדי ש-PRs יהיו טובים כברירת מחדל.

> — **Lauren Tan** But actually, the number of PRs that get reverted has actually gotten lower on a relative basis and is steady at about half a percent.

## 7. מה לעשות מחר בבוקר

- **לפני כל תיקון ל-agent, לשאול אם אפשר להפוך אותו לקוד:** lint rule, טיפוס או שינוי API עדיפים על עוד שורה ב-agents.md.
- **להתחיל ב-lint rule שחוסם מקרים חדשים, ואז לנקות את הישנים:** זה הסדר שהיא ממליצה עליו. אחרת ה-agent ימשיך להעתיק את הדפוס הישן.
- **לזהות API שדורש ניואנס ולחסום אותו:** המקבילה של useEffect אצלכם. לאסור בקוד מוצר ולעטוף ב-API בטוח.

- **לאכוף גבולות דרך מבנה התיקיות:** שמות תיקיות שמבטאים הקשר, וייבוא שנחסם על ידי dependency graph analyzer.
- **לצמצם הערות ולשים במקומן הודעות שגיאה:** ההודעה צריכה להפנות את ה-agent לשורש הבעיה ולא להצדיק טלאי.
- **לבנות runtime verification עם feature map ו-CLI קבוע:** שה-agent יריץ את האפליקציה, ישתמש ב-DevTools ויחזיר סרטון או צילום מסך כראיה.
- **לנסות אימות פורמלי במקום אחד שכואב:** זרימה מבוזרת ל-TLA+ או אלגוריתם עדין ל-Lean, בתור עוד שכבה ולא בתור silver bullet.
- **לא לבנות software factory לפני שהיסודות מוכנים:** "if your inner coding loop produces slop, then automating that loop just produces more slop".

| שאלה פתוחה | למה זה חשוב |
|---|---|
| מה היה שיעור ה-revert לפני ה-agents, ואיך הוא נמדד? | בלי baseline קשה להעריך את הטענה ש-0.5% הוא שיפור |
| איזה אחוז מה-PRs ממוזג בלי בדיקה אנושית? | הטענה ש-most product code לא דורש review עמוק לא קיבלה מספר |
| מה עולה האימות בטוקנים ובכסף? | אם 70% מהטוקנים הולכים לאימות, צריך לדעת אם זה כלכלי בצוות קטן יותר |
| מה שיעור התקלות ב-PRs שה-fuzzer ממזג אוטומטית? | זו הלולאה הכי אוטונומית שתוארה, ואין עליה שום נתון |
| כמה זמן לוקח לכתוב ולתחזק lint rule חדש? | כל התזה נשענת על כך שזה זול. ההשקעה הכוללת הייתה שישה חודשים |
| האם הגישה עובדת מחוץ לאפליקציית Electron/React? | כל הדוגמאות מגיעות מסטאק אחד |

# 8. מילון מונחים

| מונח | פירוש |
|---|---|
| slop | קוד באיכות נמוכה שנכתב בלי מחשבה, בהקשר הזה בעיקר על ידי agents |
| software factory | מערך אוטומטי שבו agents כותבים, בודקים וממזגים קוד בהיקף גדול |
| lint rule | כלל ניתוח סטטי שחוסם דפוס קוד ומחזיר הודעת שגיאה |
| agents.md / rules / skills | קבצי הנחיות בשפה טבעית ל-agent, שכבת הפרוזה בראש הפירמידה |
| useEffect | Hook של React להרצת side effects. שימוש נכון בו דורש ניואנס |
| main / renderer thread | שני התהליכים של אפליקציית Electron. ייבוא קוד main ל-renderer גורם ל-jank |
| dependency graph analyzer | כלי שמנתח את גרף הייבוא וחוסם תלויות אסורות |
| jank | גמגום וקפיצות בממשק בגלל משימות ארוכות או garbage collection |
| runtime verification | ה-agent מריץ את הקוד בעצמו ובודק את ההתנהגות בפועל |
| feature map | מפה של כל הפיצ'רים מנקודת המבט של המשתמש. נותנת ל-agent ולמפתח שפה משותפת |
| Chrome DevTools Protocol | הפרוטוקול שדרכו אפשר לשלוט בדפדפן או ב-Electron ולבדוק אותם |
| formal verification | הוכחה או בדיקה מתמטית שתכונות מסוימות של המערכת מתקיימות |
| TLA+ | שפת מפרט ל-model checking של מערכות ושל זרימות מבוזרות |
| Lean | proof assistant להוכחת נכונות של אלגוריתמים |
| property-based testing | בדיקות שמייצרות קלטים או רצפי פעולות אקראיים ובודקות שתכונה מסוימת תמיד מתקיימת |
| Bombadil | ספרייה ל-property-based testing של UI שבה הצוות משתמש |
| revert rate | אחוז ה-PRs שבוטלו אחרי שמוזגו |
| inner / outer loop | הלולאה הפנימית: כתיבה ובדיקה של שינוי בודד. הלולאה החיצונית: תזמור של הרבה agents |

| מונח | פירוש |
| --- | --- |
| fuzzing | הפעלה אוטומטית ואקראית של האפליקציה כדי למצוא באגים |
| pstack | התוסף של Tan ל-Cursor, שכולל בין השאר את ה-skill State Chart |