ה-agent שלכם מתחיל כל session כמו עובד חדש: Sonar Vortex

איך מכניסים ל-agent הקשר על הפרויקט לפני שהוא כותב, ובדיקה דטרמיניסטית אחרי כל עריכה

Killian Carlsen-Phelan (Sonar) · AI Engineer · 30.10.2026

משך: כ-18 דקות · YouTube

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאת Expo קצרה מכנס AI Engineer בניו יורק (Sutton Room, 14 באוקטובר 2026, לפי הכיתוב על המסך). Killian Carlsen-Phelan מ-Sonar מציג את Sonar Vortex, plugin שמתחבר ל-coding agent קיים (Claude Code, Codex, Antigravity ואחרים). הוא נותן ל-agent מפה של ה-codebase ואת ה-guidelines של הפרויקט עוד לפני שהוא כותב, ומריץ ניתוח של SonarQube על כל קובץ שה-agent עורך.

הטענה המרכזית: בלי הקשר, כל session של agent מתחיל כמו עובד חדש שמחפש את דרכו ב-repo, שורף tokens ומשכפל בעיות שכבר קיימות בקוד. הפתרון שמוצע כאן הוא שכבה דטרמיניסטית, ולא עוד AI שבודק AI. בדמו מוקלט Vortex תופס באג הרשאות בקובץ זמני שהבדיקות של ה-agent עצמו לא תפסו, ואפילו דוחה את התיקון הראשון שה-agent הציע.

איך לקרוא את המסמך

- **1. תקציר מנהלים:** הטענות והמספרים בעמוד אחד, וגם מה לא נאמר.
- **2. הבעיה:** למה כל session מתחיל מאפס ולמה מגלים את הבעיה רק ב-PR.
- **3. הפתרון:** שני השלבים, Guide ו-Verify, ואיך Vortex נשען על סריקת ה-CI הקיימת.
- **4. הדמו:** משימת Java אמיתית, צעד אחר צעד.
- **5. דטרמיניזם והתיקון שלא היה תיקון:** למה AI שבודק AI לא מספיק, ומה קרה כשה-agent ניסה להסתמך על ברירת מחדל.
- **6. מה לעשות עכשיו:** צעדים מעשיים וטבלת שאלות פתוחות.
- **7. מילון מונחים:** המונחים באנגלית שמופיעים במסמך.

הערת מקור המסמך הופק אוטומטית מתוך התמליל והשקפים של ההקלטה. ייתכן שזיהוי הדיבור שיבש שמות ומונחים (למשל, בתמליל מופיעים 'Cloud Code' ו-'Clutter' במקום Claude Code ו-Claude, ו-'Sonar Cube' במקום SonarQube). הציטוטים מועתקים מילה במילה ונבדקו מול חותמת הזמן שלהם. לא נוסף שום מידע ממקורות חיצוניים.

1. תקציר מנהלים

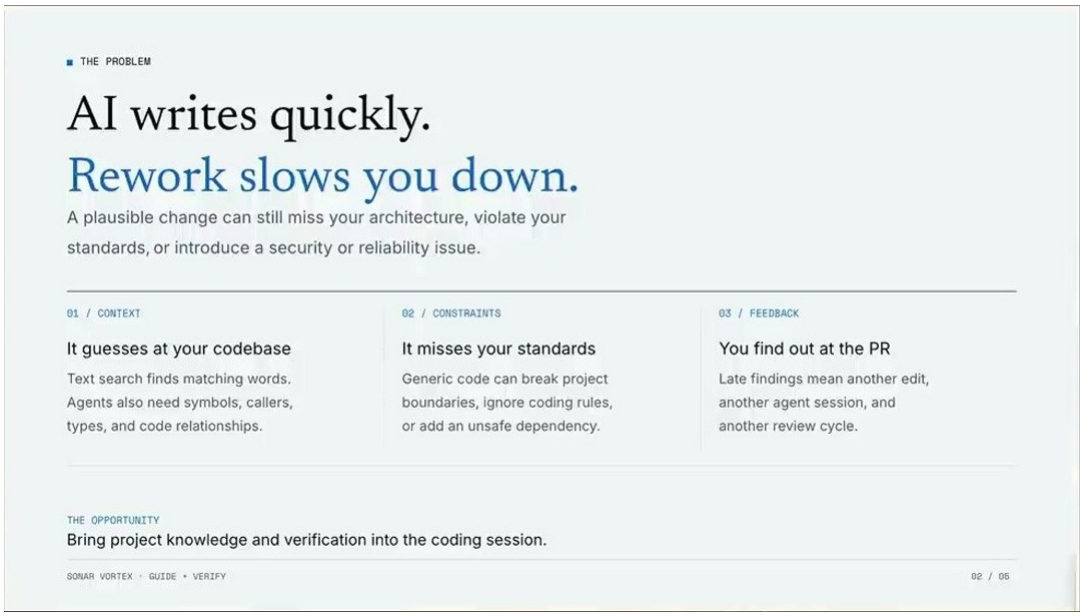
זו הרצאת מוצר של ספק, ולכן כדאי לקרוא אותה ככזו: הבעיה מוצגת היטב, הדמו קונקרטי ומשכנע, והמספרים מגיעים בלי מתודולוגיה. הרעיון ההנדסי שראוי לקחת ממנה לא תלוי במוצר: להזין את ה-agent בהקשר לפני שהוא מתחיל לכתוב, ולהריץ בדיקה דטרמיניסטית בתוך הלולאה עצמה ולא רק ב-PR.

- **כל session של agent מתחיל מאפס:** ה-agent לא מכיר את המוסכמות, את ה-guidelines ואת חוקי הארכיטקטורה. לכן הוא מחפש הקשר ב-repo, וזה עולה זמן ו-tokens. הדובר מתאר את זה כ-"a bit of a roll of the dice".
- **ה-agent משכפל בעיות קיימות:** בכל codebase יש בעיות, וה-agent ישחזר את הדפוסים שהוא רואה. מעבר לזה, הוא גם יכניס בעיות חדשות.
- **פער אמון:** לפי סקר של Sonar בקרב יותר מ-1,100 מפתחים, 96% לא סומכים לגמרי על קוד שנוצר ב-AI, ורק 48% מאמתים אותו במלואו לפני push.
- **Vortex עובד בשני שלבים:** ב-Guide ה-agent מקבל ניווט סמנטי בקוד, ארכיטקטורה ו-guidelines. ב-Verify כל עריכה מפעילה בדיקה מהירה, ובסוף ה-turn רצה בדיקה עמוקה של data flow בין קבצים,.
- **המספרים של Sonar:** 36% פחות tokens בממוצע, בעיקר במשימות refactoring, ו-44% פחות תקלות production אצל המשתמשים. לא הוצגה מתודולוגיה.
- **אין צורך בסריקה חדשה:** Vortex משתמש במה שסריקת SonarQube ב-CI כבר למדה על הקוד, כך שבדיקה של קובץ בודד נשענת על ידע על כל הפרויקט.
- **ההקשר הוא הגורם החשוב ביותר:** לדברי הדובר, אחרי בחירת המודל, ההקשר שה-agent מקבל בהתחלה הוא "the single greatest lever" לביצועים של agent.
- **דטרמיניסטי ולא AI בודק AI:** כל הרצה על אותו קוד מחזירה את אותה תוצאה, וזה מה שמאפשר governance.
- **Vortex דחה את התיקון הראשון:** ה-agent הסתמך על ברירת המחדל של Java להרשאות. Vortex דרש הרשאות מפורשות והמשיך לחסום עד שזה תוקן.

מה לא נאמר

- **אין מתודולוגיה:** לא הוסבר על איזה מדגם, מול איזה baseline ובאיזו תקופה נמדדו 36% ו-44%. גם לא ברור אם הירידה בתקלות נובעת מ-Vortex עצמו או מהשימוש ב-SonarQube באופן כללי.
- **"Very low false positive rate":** הטענה הופיעה בלי מספר.
- **תמחור ודרישות:** לא נאמר אם נדרש SonarQube Server או Cloud, איזה רישוי צריך, ומה קורה בפרויקט שאין בו עדיין סריקת CI.
- **Latency:** לא נמסר כמה זמן מוסיפה הבדיקה המהירה בכל עריכה, או הבדיקה העמוקה בסוף ה-turn.
- **הבדיקה העמוקה לא הוצגה:** ניתוח ה-data flow בסוף ה-session הוזכר אבל לא הודגם.
- **השוואה ישירה:** לא הוצגה הרצה של אותה משימה בלי Vortex, רק טענה שזה "takes longer".

2. הבעיה: כל session מתחיל מאפס



שקף 'The Problem': AI writes quickly, rework slows you down. שלוש עמודות: It guesses at your codebase, It misses your standards, You find out at the PR. בתחתית: Bring project knowledge and verification into the coding session.

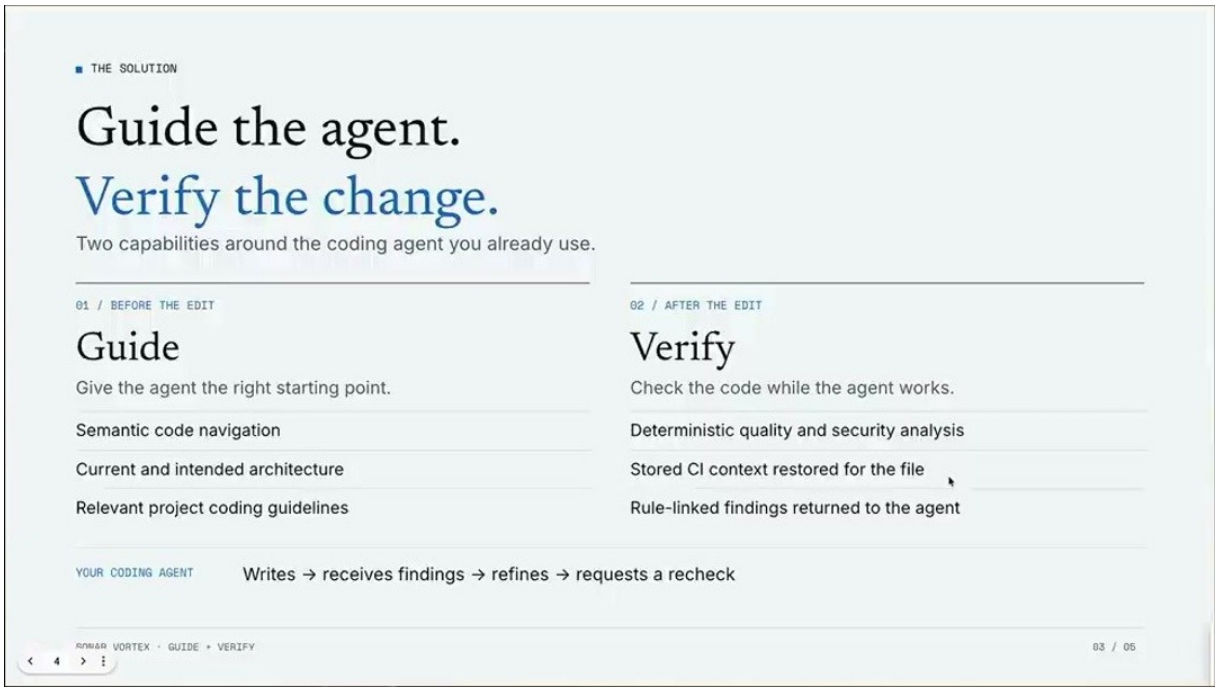
הדובר פותח עם האנלוגיה שנתנה להרצאה את שמה. agent שמקבל משימה כמו "תבנה API endpoint חדש" צריך קודם להבין איך בנויים ה-endpoints הקיימים. לפעמים הוא מוצא את ההקשר מהר, ולפעמים הוא מסתובב בכל ה-repo. כך או כך, זה עולה זמן ו-tokens, ולדבריו זו "a very fixable problem".

הבעיה השנייה חמורה יותר: אם יש בעיות בקוד, וכמעט תמיד יש, ה-agent ישכפל אותן. כשהוא מכניס בעיות חדשות ואף אחד לא תופס אותן, כמעט אי אפשר לשמור על איכות הקוד. השקף מסכם את זה בשלושה כשלים: ה-agent מנחש איך בנוי ה-codebase, מפספס את הסטנדרטים, ואנחנו מגלים את זה רק ב-PR, ואז מתחיל עוד סבב של תיקון וסקירה.

> **Killian Carlsen-Phelan** — It is like a new hired employee into your project. It does not know your existing project conventions. It does not know your coding guidelines.

הדוגמה שמלווה את כל ההרצאה: באג שקשה לכתוב עליו בדיקה. קובץ זמני נוצר בלי הרשאות מפורשות, וזה מסוג הדברים ש-senior reviewer יתפוס "only... if they're explicitly looking for it, if at all". וגם אם הוא תופס, העבודה חוזרת למפתח או ל-agent לעוד סבב של תיקון, push, סקירה וניתוח. הנתון שהדובר מביא כדי להמחיש את הפער: 96% לא סומכים לגמרי על קוד AI, ורק 48% מאמתים אותו במלואו. ובלשונו: "That terrifies me that we are just blindly setting off code into production without verifying it".

3. הפתרון: Guide the agent, Verify the change



שקף 'The Solution': שתי יכולות סביב ה-coding agent הקיים. Guide כולל semantic code navigation, ארכיטקטורה נוכחית ומתוכננת ו-coding guidelines של הפרויקט. Verify כולל ניתוח איכות ואבטחה דטרמיניסטי, שחזור הקשר ה-CI עבור הקובץ, ו-findings שמקושרים לחוקים וחוזרים ל-agent. הלולאה: Writes → receives findings → refines → requests a recheck.

כדי להסביר את ההבדל הדובר משתמש באנימציה. בלי Vortex, ה-agent הוא כדור שקופץ על פני רשת של קבצים, ובכל קפיצה בודק קובץ נוסף, בזמן שמד ה-tokens עולה. כשהוא טועה, מגלים את זה ב-PR, ה-review נכשל והעבודה חוזרת אליו. עם Vortex, לפני שה-agent כותב משהו הוא מקבל את ה-guidelines ומפה של הקשרים בקוד: ארכיטקטורה, ספריות מאושרות וסטנדרטים, שהצוות מגדיר בעצמו. משם הוא הולך ישר לקבצים הנכונים.

בשלב ה-Verify, כל שימוש של ה-agent בכלי העריכה שלו מפעיל בדיקה מהירה. לפני שהוא מסיים את ה-turn, הוא מקבל הוראה להריץ בדיקה עמוקה יותר לאיתור בעיות אבטחה ו-data flow בין קבצים. ה-finding חוזר ל-agent, הוא מתקן, ו-Vortex בודק שוב. מבחינת אינטגרציה: Vortex בנוי על ה-CLI ועל כלי ה-MCP של Sonar. הוא תומך ב-Claude Code, Codex ו-Antigravity, ול-agents מותאמים אישית או פתוחים יש MCP clients שאפשר להגדיר.

> **Killian Carlsen-Phelan** — it spends 36% fewer tokens on average, especially on tasks like refactoring existing code bases. People who use it have a 44% reduction in production outages.

הנקודה הארכיטקטונית שהכי מעניינת: Vortex לא סורק את הפרויקט מחדש. SonarQube פועל כבר כמעט שני עשורים כשלב ב-pipeline, ובונה גרפים של הקוד לניתוח אבטחה, maintainability ו-data flow. הידע הזה כבר שמור בשרתים, ו-Vortex מעביר אותו ל-agent. "Your CI already has a SonarCube scan. And the Vortex-reuses what that scan has learned." לכן בדיקה של קובץ בודד יכולה להתחשב בשאר הקוד.

4. הדמו: משימה אמיתית

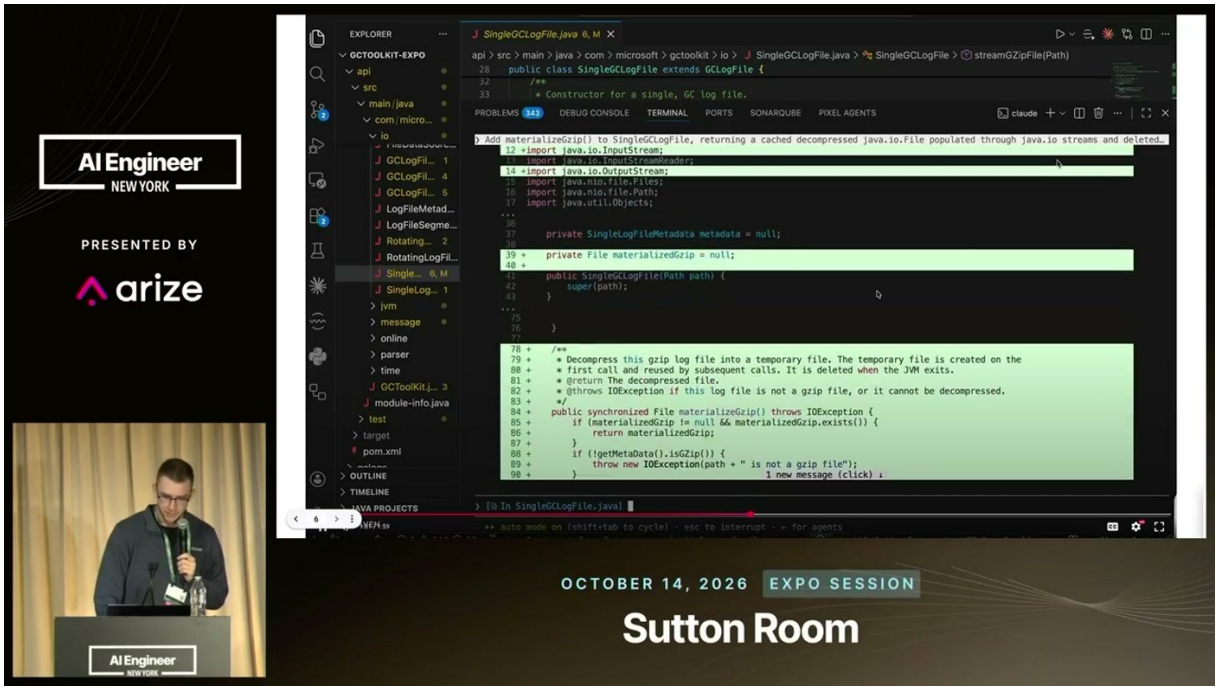
```
 7 +import java.io.IOException;
 8 +import java.io.OutputStream;
 9 +import java.nio.charset.StandardCharsets;
10 +import java.nio.file.Files;
11 +import java.nio.file.Path;
12 +import java.util.zip.GZIPOutputStream;
13 +
14 +import static org.junit.jupiter.api.Assertions.assertEquals;
15 +import static org.junit.jupiter.api.Assertions.assertSame;
16 +
17 +public class SingleGCLogFileTest {
18 +
19 +    @Test
20 +    void materializeGzipReusesCachedFile(@TempDir Path tempDir) throws IOException {
21 +        String content = "[0.010s][info][gc] Using G1\n";
22 +        Path gzip = tempDir.resolve("gc.log.gz");
23 +        try (OutputStream out = new GZIPOutputStream(Files.newOutputStream(gzip))) {
24 +            out.write(content.getBytes(StandardCharsets.UTF_8));
25 +        }
26 +
27 +        SingleGCLogFile logFile = new SingleGCLogFile(gzip);
28 +        File first = logFile.materializeGzip();
29 +        File second = logFile.materializeGzip();
30 +
31 +        assertSame(first, second);
32 +        assertEquals(content, Files.readString(first.toPath()));
33 +    }
34 +}
(files that changed while this command ran, by the command or by another process; a convenience view, not a review or audit of
the command)
Allowed by auto mode classifier
```

ה-IDE בתחילת ה-session: ה-agent עובד על מחלקת Java לטיפול בקובץ GC log, וב-diff בירוק רואים את הקוד והבדיקות החדשים.

הדמו הוקלט ערב לפני ההרצאה. כבר בפתיחת ה-session רואים ש-SonarQube עלה מעצמו: ה-CLI זוהה, מערכת ה-hooks זוהתה ו-secrets detection פעיל. לדברי הדובר, ה-plugin מונע דליפת secrets ל-GitHub ולזיכרון ההקשר של ה-agent. הוא מדגיש: "I didn't have to paste any Sonar instructions into the prompt."

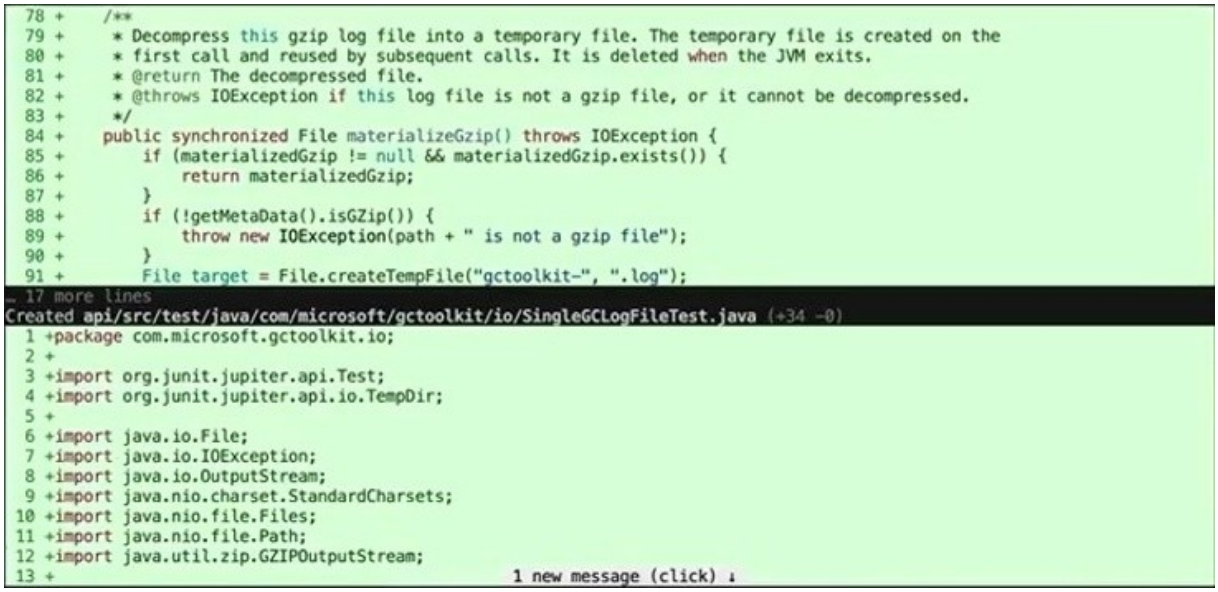
המשימה: לפרוס קובץ log מכווץ לקובץ זמני, להשתמש שוב באותו קובץ בכל פעם, ולכתוב בדיקה שמוכיחה שהוא אכן משמש שוב. לפני שורת קוד ראשונה, ה-agent שואל את Sonar שתי שאלות: איפה המחלקה נמצאת (Context Navigation, שמחזיר את מבנה ההורה והילד של המחלקה ואת הדפוסים הקיימים), ומה ה-guidelines של Java בפרויקט (Sonar Context Guidelines). דרישות שהצוות הגדיר בעצמו נכנסות כבר בשלב הזה.

> **Killian Carlsen-Phelan** — outside of model choice, the context in steering from the get-go that agents receive is the single greatest lever that you can manipulate for agentic performance.



המימוש: יצירת הקובץ הזמני ושיטה לפריסת קובץ ה-log, עם Javadoc שמתאר את השימוש החוזר בקובץ.

שלב ה-Guide לקח כמה שניות. ה-agent בדק את הבדיקה הקיימת והמשיך למימוש. בלי Vortex, לדברי הדובר, אותה משימה לוקחת יותר זמן כי ה-agent צריך לחפש עם grep. ה-agent כתב בעצמו test suite. הקובץ הזמני נוצר בשורה קטנה אחת. אחרי שהבדיקות עברו, ה-agent הריץ Vortex analysis על שני הקבצים ששינה. הניתוח הזה רץ אחרי כל עריכה או הוספה של קובץ, בלי שצריך לדרוש אותו.

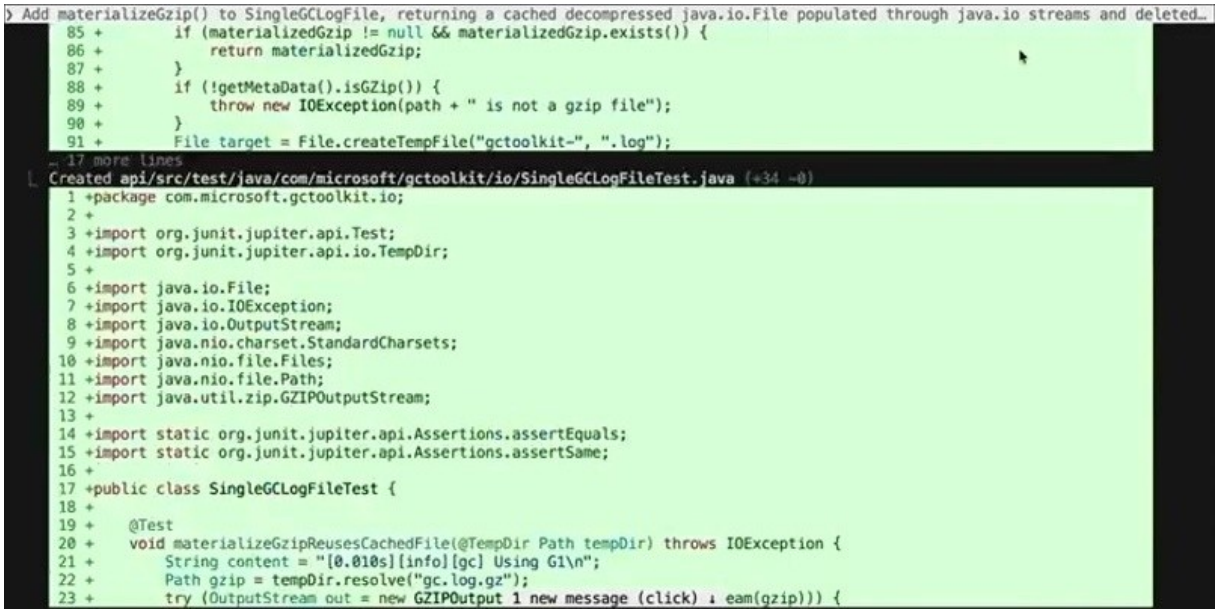


ה-agent מוסיף את קוד הבדיקה ואת ה-imports בקובץ הבדיקות, לפני הרצת ניתוח ה-Vortex על שני הקבצים שהשתנו.

5. לא AI בודק AI, והתיקון שלא היה תיקון

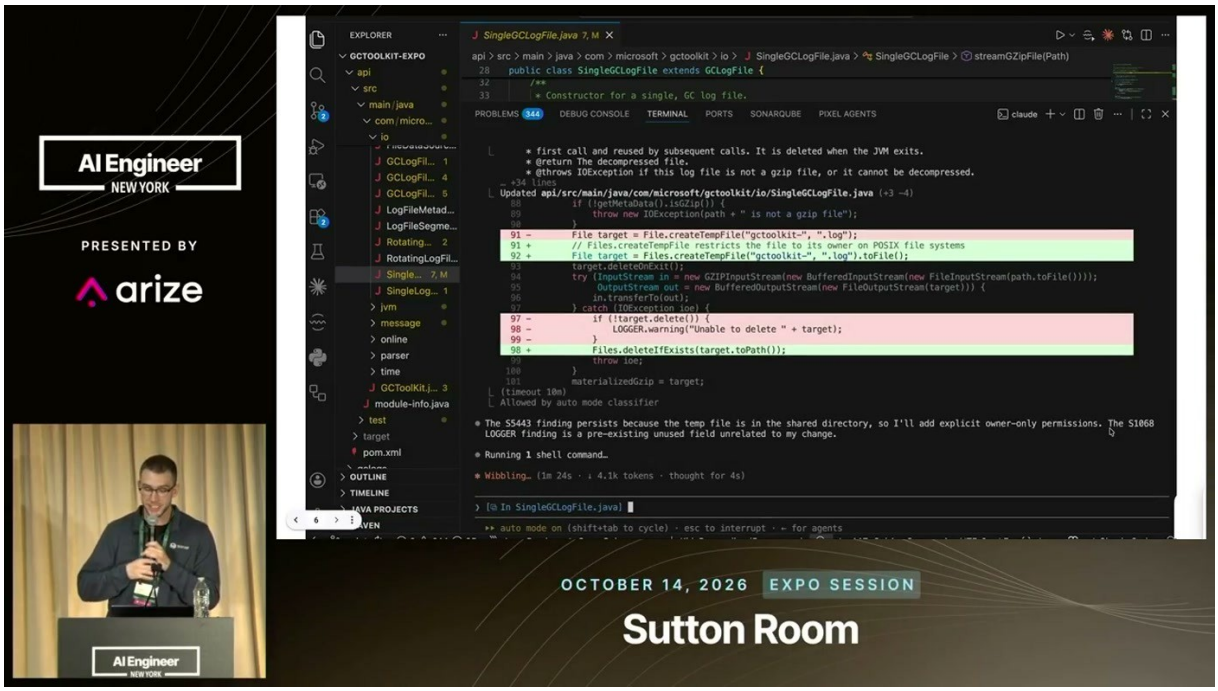
כאן מגיע הטיעון העקרוני של ההרצאה. לדברי הדובר, AI שבודק את עצמו הוא "one of the least reliable ways to guarantee quality". agents חושבים אחרת כשהם כותבים וכשהם סוקרים, ו-agents אחרים נוטים לפספס ולהזות כשהם סוקרים קוד קיים. שכבה דטרמיניסטית מחזירה בכל הרצה את אותה תוצאה, ולכן היא "reliable, governable guidance and verification that is compliant and predictable". הוא לא פוסל AI review: ל-Sonar יש גם כלי כזה (שמו בתמליל הוא 'Guitar', כנראה שיבוש של ASR, ולא ברור מה השם הנכון). הוא מתאר את Vortex כשכבה דטרמיניסטית שיושבת בין ה-AI שכותב לבין ה-AI review.

> **Killian Carlsen-Phelan** — One point as well, this is not another AI checking AI. And it's certainly not it checking itself.



אחרי הניתוח: ה-agent מעדכן את קוד יצירת הקובץ הזמני בתגובה ל-findings של Vortex.

הניתוח החזיר שישה findings: שניים בשורות שה-agent כתב עכשיו וארבעה בקוד שהיה קיים קודם. כאן מופיעה החלטת עיצוב חשובה. בקוד חדש Vortex מחייב תיקון, ובקוד קיים הוא רק מציג את הממצא לידיעה, כדי לא להציף. "We did not engineer Vortex to be super noisy." אחד מהממצאים בקוד החדש הוא באג האבטחה מהפתיחה: "Claude put the log in the default temp folder without saying who can read it", ובהרבה שרתים התיקייה הזו משותפת.



ה-diff של התיקון השני: הסרת הגרסה הקודמת (באדום) והגדרת הרשאות מפורשות לקובץ הזמני, אחרי ש-Vortex דחה את התיקון הראשון כי הממצא עדיין קיים.

> **Killian Carlsen-Phelan** — The first fix leaned on a Java default. And on Mac and Linux, the default is already owner only. But the rule that we set in, which is a high bar that we set, is that the permissions must be set explicitly.

זה הרגע הכי משכנע בדמו. Claude תיקן, Vortex רץ שוב וקבע שהממצא נשאר כי הקובץ עדיין נמצא בתיקייה משותפת. בתיקון השני Claude הגדיר הרשאות מפורשות, והניתוח רץ שוב. בסוף ה-session, כשה-agent מחזיר את השליטה, רץ ניתוח data flow עמוק. במילים של הדובר, ה-agent "asked sonar first. When sonar flagged its fix, it went further and continued to remediate itself."

6. מה לעשות עכשיו

- **בדקו כמה הקשר ה-agent שלכם מקבל בהתחלה:** המסר שאינו תלוי במוצר הוא שהקשר בתחילת ה-session (ארכיטקטורה, ספריות מאושרות, guidelines) מקטין את כמות העבודה החוזרת. כדאי לבדוק מה ה-agent יודע לפני שהוא מתחיל לחפש עם grep.
- **העבירו בדיקות דטרמיניסטיות לתוך הלולאה:** אם יש לכם כבר linters, SAST או SonarQube ב-CI, שווה לחבר אותם ל-hook שרץ אחרי כל עריכה של ה-agent, ולא רק ב-PR.
- **הפרידו בין קוד חדש לקוד קיים:** לחייב תיקון רק על קוד חדש ולהציג את מה שבקוד הקיים לידיעה. זה דפוס שעוזר לשמור על רעש נמוך.
- **דרשו כוונה מפורשת ולא ברירות מחדל:** הדוגמה של הרשאות לקובץ זמני מראה שקוד יכול להיות נכון בסביבה אחת ולא בטוח בשרת משותף. זה סוג החוקים שכדאי לאכוף אוטומטית.
- **אם אתם לקוחות SonarQube:** שווה לבחון את Vortex ולמדוד בעצמכם את חיסכון ה-tokens ואת ה-latency, במקום להסתמך על 36%.

שאלה פתוחה	למה זה חשוב
איך נמדדו 36% פחות tokens ו-44% פחות תקלות?	בלי baseline ומדגם אי אפשר לדעת אם המספרים רלוונטיים לפרויקט שלכם.

שאלה פתוחה	למה זה חשוב
מה שיעור ה-false positives בפועל?	בלולאה שבה ה-agent נדרש לתקן כל ממצא, false positive עולה ב-tokens ובסבבים.
כמה latency מוסיפה הבדיקה אחרי כל עריכה?	ב-session ארוך עם עשרות עריכות זה מצטבר.
מה קורה בפרויקט בלי סריקת SonarQube ב-CI?	Vortex נשען על הידע שהסריקה כבר צברה.
מה הדרישות והתמחור?	לא נאמר אם נדרש Server, Cloud או רישוי מסוים.
איך מגדירים guidelines מותאמים?	נאמר שאפשר להזריק דרישות של הצוות, אבל לא הוצג איך.

7. מילון מונחים

מונח	פירוש
Coding agent	כלי AI שכותב ועורך קוד באופן אוטונומי בתוך ה-IDE או הטרמינל, כמו Claude Code או Codex.
Session	הפעלה של ה-agent. כל אחת מתחילה בלי זיכרון של הפרויקט.
Tokens	יחידות הטקסט שהמודל מעבד. הן קובעות את העלות ואת ניצול חלון ההקשר.
Sonar Vortex	ה-plugin של Sonar שמחבר את SonarQube ל-agent ומספק את שלבי ה-Guide וה-Verify.
SonarQube	הפלטפורמה של Sonar לניתוח סטטי של קוד, שרצה בדרך כלל כשלב ב-CI.
Guide / Verify	שני שלבי Vortex: מתן הקשר לפני הכתיבה, ובדיקה אחרי כל עריכה.
Deterministic	ניתוח שמחזיר בכל הרצה את אותה תוצאה על אותו קוד, בניגוד לסקירה של מודל.
Data flow	ניתוח של מעבר הנתונים בקוד, כולל בין קבצים, לאיתור בעיות אבטחה.
Hook	נקודת הרחבה שמריצה פעולה אוטומטית, למשל בדיקה אחרי כל עריכה של ה-agent.
MCP	Model Context Protocol, פרוטוקול לחיבור כלים ומקורות מידע ל-agents.
CLI	ממשק שורת פקודה.
Secrets detection	זיהוי של מפתחות וסיסמאות כדי שלא ידלפו לקוד או להקשר של ה-agent.
False positive	ממצא שגוי, כלומר התרעה על בעיה שאינה קיימת.
PR (Pull Request)	בקשת מיזוג קוד שעוברת סקירה לפני שהיא נכנסת לענף הראשי.
Refactoring	שינוי מבנה הקוד בלי לשנות את ההתנהגות שלו.