כל ה-agents של הצוות שלך מגיעים לאותו קוד בבת אחת

Patrick Debois מסביר למה continuous delivery לא נבנה ל-software factory שכמה אנשים עובדים בו במקביל: הוא מציע ענף main אחד ל-agents ולוח משימות אחד לאנשים

10 · AI Engineer · Patrick Debois (Tessl) בנובמבר 2026
משך: כ-23 דקות · YouTube
סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה קצרה (כ-23 דקות) של Patrick Debois מ-Tessl בערוץ AI Engineer. הנושא הוא מה שהוא קורא לו multiplayer software factory: מצב שבו כמה אנשים וכמה coding agents עובדים בו-זמנית על אותו codebase ועל אותו לוח משימות. הוא מבחין בין solo factory על הלפטופ, שבו המפתח מחליט וממזג בעצמו ולמעשה משמש כזיכרון של כל התהליך, לבין factory משותף שבו כולם מנסים למזג ל-main באותו זמן.

לטענתו, כשהמקביליות הופכת מחריג לשגרה, ה-pipeline הסדרתי של CI/CD נשבר: הרבה יותר PRs, rebase בלי סוף ו-CI איטי ויקר יותר. לכן צריך לעבור מ-continuous delivery ל-concurrent delivery. ההרצאה בנויה משני חלקים: צד הקוד (one main, איך agents מתחלקים בעבודה וממזגים) וצד האנשים (one board, החלטות, review ותקשורת). זו הרצאת חזון וסקירת דפוסים. אין בה מדידות ואין בה מקרה בוחן מפורט.

איך לקרוא את המסמך

- **1. תקציר מנהלים:** הטענות המרכזיות בעמוד אחד, ומה לא נאמר.
- **2. הבעיה:** המעבר מ-solo factory ל-multiplayer ומה מתחיל להישבר.
- **3. ענף main אחד:** איך מחלקים עבודה, תופסים משימה, מבודדים סביבה וממזגים בחכמה, כולל קונפליקט כוונה ו-speculative merge trains.
- **4. לוח אחד לאנשים:** agent בתוך הצ'אט, משתתפים שאינם מהנדסים, החלטות כאפשרויות לבחירה ו-decision ledger.
- **5. Review לפי סיכון:** לא כל שינוי מקבל אותה רמת בדיקה.
- **6. תקשורת וארגון:** הפצת שינויים לכל תפקיד, skills לשגרה וה-factory team.
- **7–8. פעולות, שאלות פתוחות ומילון מונחים.**

> **הערת מקור** המסמך הופק אוטומטית מהתמלול של ההקלטה ומהשקפים שחולצו ממנה. ה-ASR עלול לשבש שמות ומונחים: בתמלול Tessl מופיע כ-"TESOL" וכ-"Tesla", AST graph מופיע כ-"ASD graph" ו-merge trains כ-"merge strains". הציטוטים הועתקו מילה במילה ונבדקו מול חותמת הזמן שלהם. לא נוסף מידע ממקורות חיצוניים.

1. תקציר מנהלים

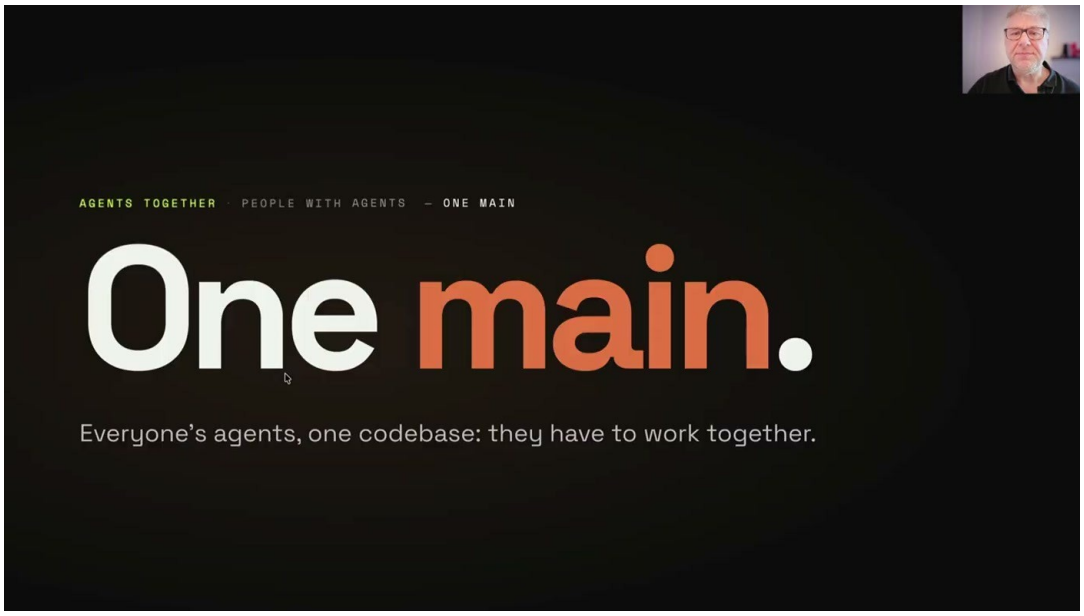
הטענה המרכזית: ברגע שצוות שלם מריץ coding agents, הבעיה כבר לא "agent אחד שכותב קוד טוב". הבעיה היא תיאום, גם בין agents שמתחרים על אותו main וגם בין אנשים שמנהלים את כולם. Debois מציג קטלוג דפוסים לשני הצדדים וחוזה שתיווצר שכבת תשתית חדשה וצוות חדש שינהל אותה.

- **מספר ה-PRs קופץ פי 10 בערך:** כש-factory משותף עולה לאוויר, מגיעים הרבה יותר PRs, יותר קונפליקטים, יותר rebase ו-CI איטי ויקר יותר. בתור נכנסים גם תיקוני תקלות של agents ותחזוקה של ה-factory עצמו.
- **Concurrent delivery במקום continuous delivery:** צריך לתכנן מראש למקביליות כשגרה, ולא לטפל בה כחריג.
- **לחלק את העבודה לפני שכותבים קוד:** לפי תחום (authentication, payment), בעזרת גרף ה-AST, ועד רמת קובץ ופונקציה. agent תופס כרטיס ומודיע אילו קבצים ישנה.
- **להקפיא לכל agent את הסביבה שלו:** worktree, מכונה, context, גרסת API נעולה, snapshot של ה-DB ו-digital twins לעולם החיצוני.
- **קונפליקט כוונה מסוכן יותר מקונפליקט קוד:** שני PRs שעוברים merge וטסטים, אבל אחד עבר לסנטים והשני מדווח בדולרים. agents צריכים לבדוק עקביות של כוונה.
- **Speculative merge trains:** בודקים את ה-PRs שבתור בזוגות, כך ש-PR אחד שנכשל לא תוקע את כל השאר.
- **agent בצ'אט במקום prompt:** מכניסים agent לשיחה של הצוות, הוא מחדד את הכרטיס מכל המשתתפים, ואנשי מכירות, שיווק ותמיכה יכולים להתניע פיצ'ר בעצמם.
- **Review לפי סיכון ולא רק לפי איכות:** שינוי בתיעוד יכול לעבור בלי בדיקה אנושית, שינוי ב-authentication דורש review אנושי, ויש גם gradual release ושחזור זול.
- **Factory team חדש:** צוות דמוי platform team שאחראי על ה-skills, ה-context והמוניטורינג של ה-factory.

מה לא נאמר

- **אין מספרים מעבר ל"פי 10":** לא נמסרו נתונים על זמני CI, עלויות או אחוזי קונפליקטים, לא לפני השינוי ולא אחריו.
- **אין שמות כלים לרוב הדפוסים:** speculative merge trains, בדיקת intent ו-decision ledger מוצגים כרעיונות. הכלי היחיד שנקרא בשמו הוא HelixML, ו-Tessl מוזכר כ-factory של הדובר.
- **לא ברור מה כבר עובד ב-Tessl:** הדובר אומר "We're not there yet", ולא מפריד בין מה שהם מריצים בפועל לבין מה שהוא רק מתאר.
- **אין דיון בכשלים:** לא סופר מקרה שבו בדיקת intent או מיזוג אוטומטי של agent טעו, ומה עלה התיקון.

2. מ-solo factory ל-multiplayer: הבעיה



שקף הפתיחה של החלק הראשון: "One main." עם תת-הכותרת "Everyone's agents, one codebase: they have to work together".

Debois מתאר שלוש דרגות. בראשונה מפתחים מריצים factories על הלפטופ. בשנייה הם משתפים setup: context, skills ו-loops. בשלישית, ה-multiplayer factory, אנשים עובדים יחד באותו לוח משימות ועל אותו codebase. לדבריו רוב ה-factories היום נמצאים בדרגה השנייה. בדרגה השלישית יש שני מוקדי מתח: כולם רוצים למזג ל-main, וכמה אנשים צריכים לעבוד יחד על אותו לוח.

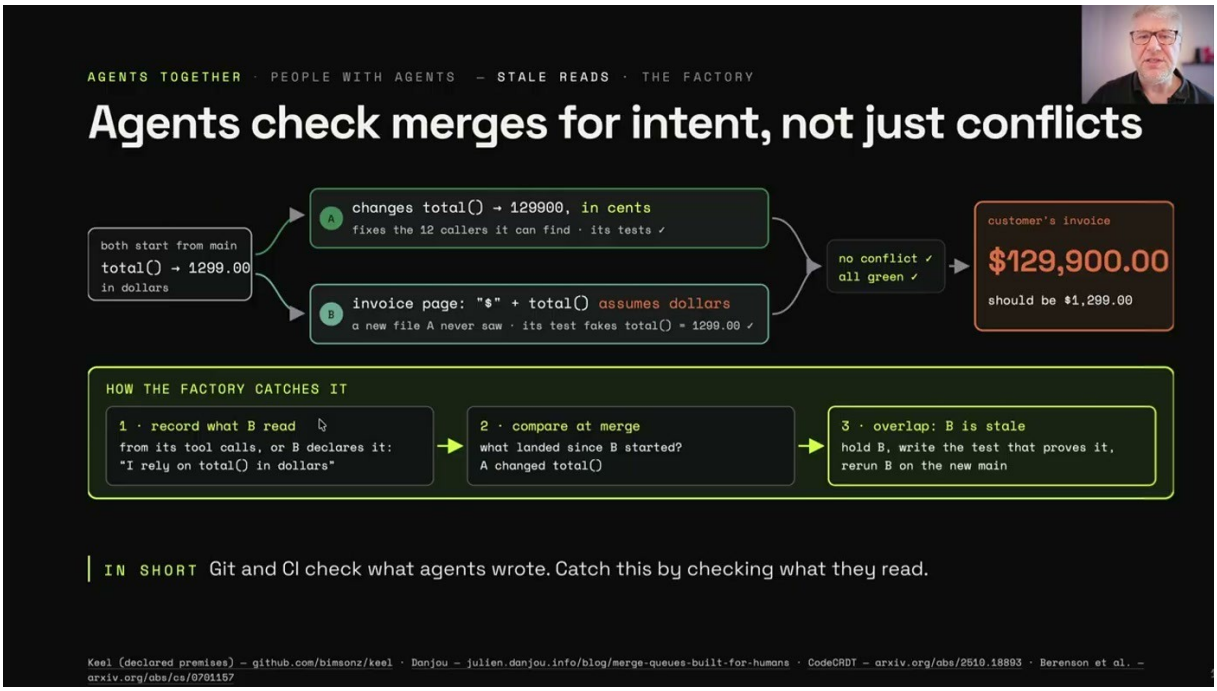
הסימפטום הראשון הוא נפח: "it's almost like a factor 10 of more and more pull request landing". מכאן יותר merges, agents שמבטלים זה לזה שינויים ("one agent changing something then the other agent changing it back"), וכל merge גורם לשאר ה-PRs לעשות rebase ולהריץ טסטים מחדש. התוצאה היא CI/CD איטי ויקר יותר. בנוסף, התור מתמלא בתיקוני תקלות של agents ובתחזוקה של ה-factory עצמו (שיפור skills ו-loops), וכל אלה דורשים אישור אנושי.

> **Patrick Debois** So why can't we flip this from being this concurrency and it being an exception to becoming the routine. —

3. ענף main אחד: חלוקה, תפיסת משימות ומיזוג

הצעד הראשון הוא לחלק את העבודה לפני שכותבים קוד, כמו שמחלקים בין אנשים: אחד על authentication ואחר על payment. אפשר להיעזר בגרף ה-AST (בתמלול: "ASD graph") כדי לזהות חלקים שקל להפריד, ולרדת לרמת קובץ, פונקציה ואפילו הקלדות בתוך קבצים. בזמן הביצוע agent תופס כרטיס ("it claims the ticket") ומודיע מראש אילו קבצים ושינויים הוא מתכוון לגעת בהם, כדי ש-agents אחרים יראו ולא יחפפו.

הבידוד הולך הרבה מעבר ל-worktrees: "You basically are freezing the world of an agent". לכל agent יש מכונה משלו, הקפאה של ה-context, גרסת API נעולה, snapshot של ה-DB ו-test data משלו, ו-digital twins או mocks לכל מה שבחוץ. כש-agents בכל זאת מתנגשים יש כמה אסטרטגיות: לחכות, לנסות שוב אחרי המתנה, לחזור לתור, או לנהל משא ומתן מחדש על התוכנית תוך כדי ביצוע. גם המיזוג עצמו יכול להיות חכם יותר: לא לפי שורות אלא לפי פונקציות, agent שמתקן את הקונפליקט בעצמו, ואפילו agent שצופה שינוי שעוד לא הגיע ומתאים את הקוד שלו מראש.

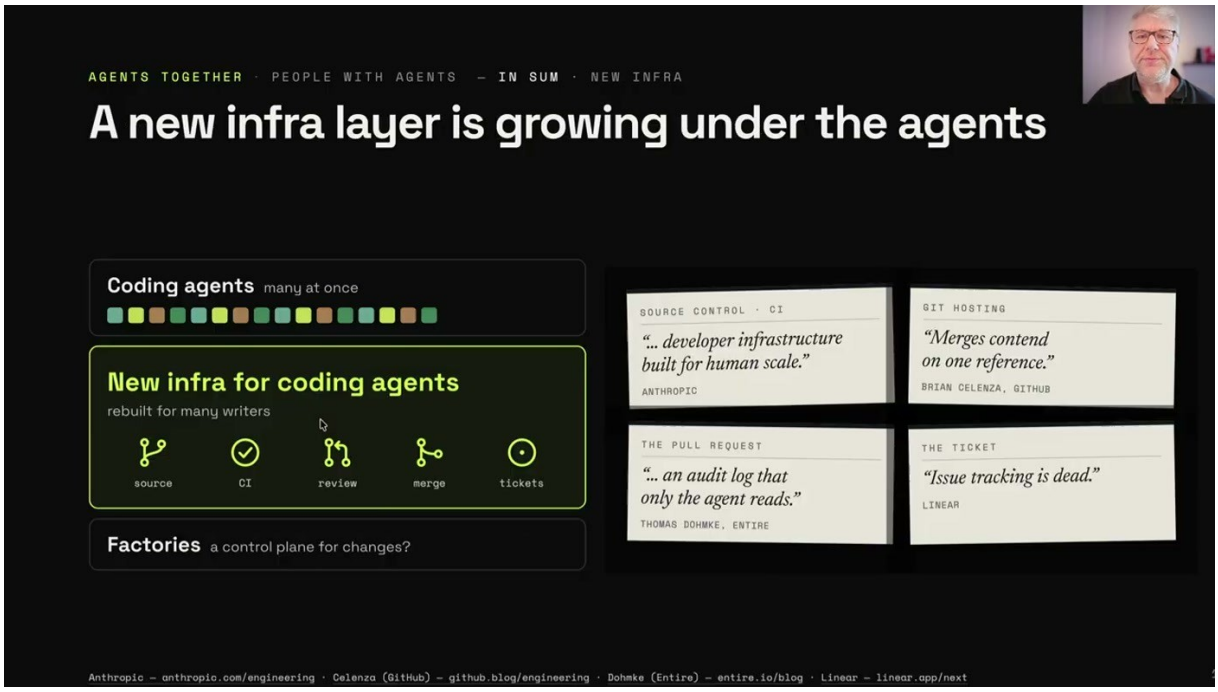


"Agents check merges for intent, not just conflicts": שינוי אחד הופך את total() לסנטים, השני מציג total כדולרים, והחשבונית יוצאת $129,900.00 במקום $1,299.00. בתחתית: "Git and CI check what agents wrote. Catch this by checking what they read".

הדוגמה המרכזית בפרק: PR אחד משנה את חישוב הסכום של חשבונית לסנטים, ו-PR מקביל משנה את הדוח שמציג אותו. שניהם עוברים merge ושני ה-test suites עוברים, ובכל זאת התוצאה שגויה. השקף מוסיף ניסוח חד שלא נאמר כך בדיבור: Git ו-CI בודקים מה ה-agents כתבו, והפתרון הוא לבדוק מה הם קראו. ההצעה היא ש-agents יבדקו עקביות של כוונה בין merges, יזהירו או יתקנו את ה-PR.

הדפוס השני הוא speculative merge trains (בתמלול: "merge strains"). במקום rebase של כל התור כשמיזוג אחד נכשל, המערכת בודקת את ה-PRs הנכנסים בזוגות ומזהה אילו תואמים זה לזה. PR שנכשל יוצא מהתור והשאר ממשיכים, כך שהתור מתקדם מהר יותר.

— **Patrick Debois** So they're not conflicting in code but they're conflicting on intent.



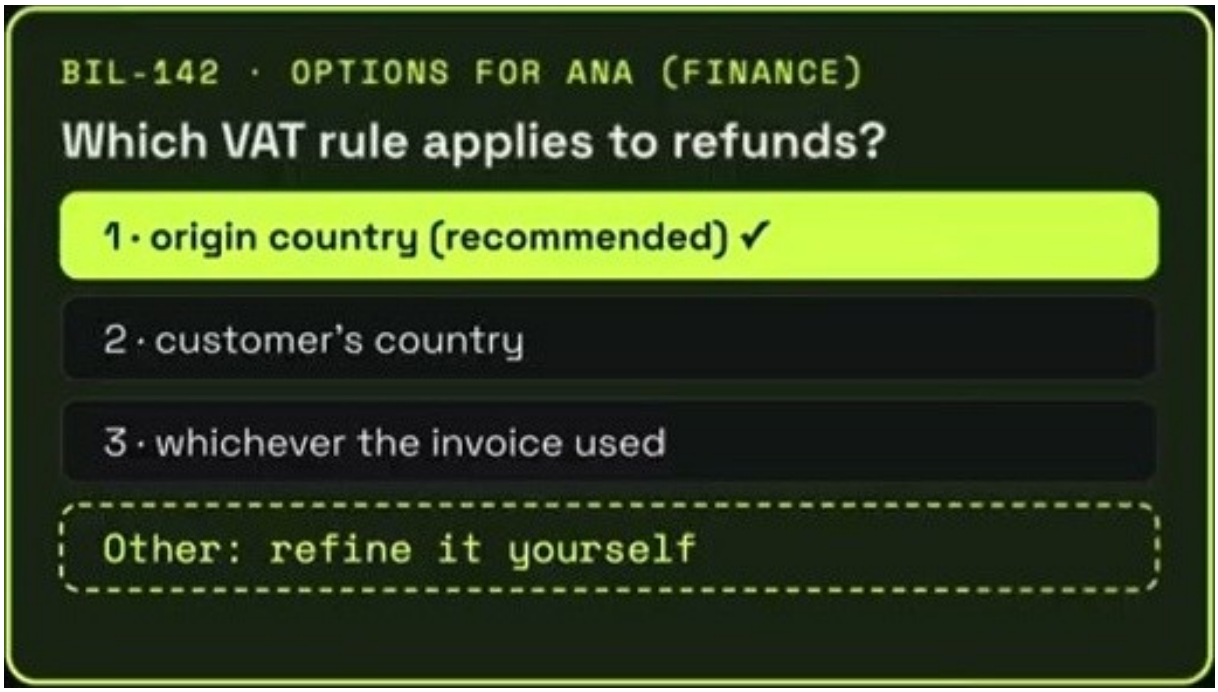
"A new infra layer is growing under the agents": מעל, coding agents שרצים במקביל. באמצע, תשתית חדשה ל-coding agents (source, CI, review, merge, tickets). למטה, Factories כ-control plane לשינויים. בצד, ציטוטים מהתעשייה כמו "Merges contend on one reference" ו-"Issue tracking is dead".

Debois מסכם את החלק הזה בתחזית: נבנית שכבת developer infrastructure חדשה, וגם הספקים מכוונים אליה. היא כוללת merge אחר, analytics ותכנון טוב יותר. היום, לדבריו, "a lot of the factories are just glorified loops, as we would do with the CireCD, but the concurrency will only increase". לפי ההקשר, "CireCD" הוא כנראה CI/CD בשיבוש של ה-ASR.

4. לוח אחד לאנשים

הצד השני הוא האנשים. בעבודה מרובת משתתפים כולם עובדים על task board משותף שמראה אילו שינויים בדרך, משהו שלא קיים ב-solo factory. בגלל שכל ה-agents צריכים תשומת לב, רוצים גם לצמצם את מספר הפעמים שהם פונים לאדם. השינוי שהדובר מדגיש הוא agent שמצטרף לצ'אט: "It is not you prompting, but it's you kind of pulling an agent into a conversation". ה-agent אוסף רמזים מכל חברי הצוות, מחדד את הכרטיס, שואל מה הטסט ומה הדרישות, והכרטיס הופך למעין plan mode משותף בלוח.

לפי Debois, ב-factory של Tessl גם אנשי מכירות, שיווק ותמיכה משתתפים בשיחות האלה. בעבר הם פתחו כרטיס וחיכו שבועות. ב-vibe coding הבעיה הייתה שהתוצאה לא עמדה בפרקטיקות ההנדסיות. עכשיו הם עובדים על אותו pipeline, עם אותם skills ו-loops, ויכולים להתניע פיצ'ר שמהנדס ימשיך. לאיטרציות מהירות הוא מביא רעיון של HelixML: כשה-agent נתקע, נפתח IDE עם בדיוק ה-context שלו, אדם קופץ פנימה לעזור, ואחר כך מחזיר את העבודה ל-factory.



"Only real decisions reach a person, as options to pick from": מה שכבר הוחלט נשלף מהיסטוריה, ושגרה זולה לטעות נאספת ל-daily batch. החלטה אמיתית (איזה כלל VAT חל על החזרים) מגיעה לבעלת התחום כאפשרויות: Ana בוחרת origin country ו-BIL-142 ממשיך.

העיקרון: לצמצם את מספר ההחלטות, ולהציג את מה שנשאר כאפשרויות לבחירה ולא כשאלה פתוחה, כמו שעושים coding agents היום. כל החלטה נרשמת ב-ledger משותף, מעין decision log. agents ואנשים יכולים לחפש בו, לבדוק מי החליט מה ומתי, ולשלוף ממנו context בסשן הבא. הוא מוסיף ששינויים צריכים להיות קטנים, כי גם יכולת ה-review של אדם היא צוואר בקבוק. אפשר לפצל ל-PRs נפרדים ולכמה merge trains כדי לאפשר review מהיר.

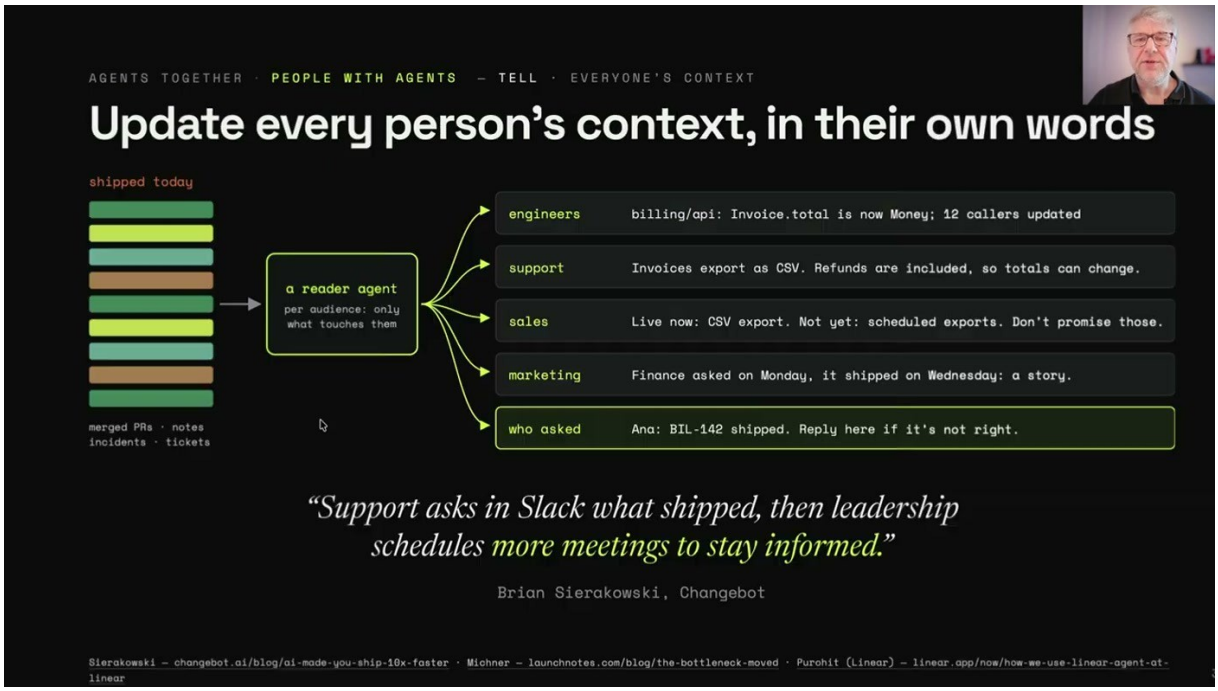
5. Review לפי סיכון

Debois מתאר שלוש שכבות. הראשונה מקטינה את הסיכוי לכשל: context וכלים טובים יותר, ו-agents אדברסריאליים שעוברים על הקוד לפני שהוא מגיע לאדם. השנייה מקלה על ה-reviewer: צילומי מסך, וידאו, הוכחות ותוצאות טסטים. השלישית היא הכרעה לפי השפעה. יש אנשים שאומרים לו שהם כבר לא מסתכלים על הקוד, והוא שואל מה ההשפעה בפרודקשן. שינוי בתיעוד יכול לעבור, שינוי ב-authentication דורש review אנושי, ובאמצע יש gradual release עם מעקב אחרי שגיאות.

מהצד של הפרודקשן מצמצמים את ההשפעה ולא רק את הסבירות. אם השינוי עלול לאבד נתונים, יש fast restore? מה קורה אם node נופל? אפשר לעשות rollback מהיר? לדבריו אלה החלטות ארכיטקטורה.

> **Patrick Debois** So what you want to do is when things go wrong, you want to have it being cheap — to restore or to fix.

6. תקשורת וארגון: ה-factory team



"Update every person's context, in their own words": מה שנשלח היום עובר דרך reader agent שמתרגם אותו לכל תפקיד (engineers, support, sales, marketing, who asked). בתחתית ציטוט: "Support asks in Slack what shipped, then leadership schedules more meetings to stay informed." שם המצוטט בשקף לא קריא.

Debois עובד מרחוק, ורוב הצוות יושב בלונדון. לדבריו כבר לפני ה-factory היה לו קשה לעקוב אחרי מה שהשתנה. ב-multiplayer factory נוצר כל כך הרבה קוד שגם שיווק ותמיכה לא מצליחים לעקוב, ולכן הם צריכים תרגום של השינויים לשפה של התחום שלהם. מבחינתו הפצת שינויים ("radiating change") היא פיצ'ר מרכזי ב-factory ולא תוספת.



"Skills for the routine. People for the exceptions": בצד השגרה, רישום skills עם גרסה ובעלים (ui-patterns, auth-rules, api-conventions, known-issues, refund-vat). בצד החריגים, אנשים מצטרפים רק כשצריך. בתחתית: "Record every answer, and promote the ones that recur into a tested skill, with an owner".

השורה התחתונה שלו: "automate what you can, but make sure you are able to route to a human to make the right decision". כדי לנהל את כל זה נוצר צוות חדש, factory team, בדומה ל-platform team שניהל Kubernetes או את הענן. הצוות הזה אחראי על הבעלות על ה-skills, על עדכניות ה-context, על החיבורים, ועל

התחזוקה והמוניטורינג של ה-factory. בסוף ההרצאה הוא מסתכל קדימה: personal agents שיעבדו בלוח בשמך, יקבלו החלטות ויענו ל-agents אחרים. הוא גם מפנה ל-concurrent.cd, אתר שבו הוא אוסף אנשים, כלים וטכניקות בתחום.

> — So my advice is brace yourself. We're not there yet **Patrick Debois**.

7. מה עושים מחר

- **למדוד את התור:** כמה PRs נכנסים, כמה rebase reruns יש ומה עלות ה-CI. זה מאפשר לראות אם המקביליות כבר מכבידה.
- **לחלק עבודה לפני ההרצה:** להקצות לכל agent תחום או קבצים, ולדרוש ממנו להודיע על הקבצים שבכוונתו לשנות בעת תפיסת הכרטיס.
- **לבודד יותר מ-worktree:** snapshot של ה-DB, גרסת API נעולה ו-mocks לעולם החיצוני לכל agent.
- **להוסיף בדיקת intent ל-merge queue:** ולבדוק במיוחד שינויים ביחידות, בפורמטים ובחוזים בין רכיבים שעוברים טסטים כל אחד בנפרד.
- **לנהל review לפי סיכון:** להגדיר אילו תחומים (למשל תיעוד) עוברים אוטומטית ואילו (למשל authentication) מחייבים אדם, ולהשקיע ב-rollback ו-restore זולים.
- **לפתוח decision log משותף:** ולהפוך תשובות שחוזרות על עצמן ל-skill עם בעלים.
- **למנות בעלים ל-factory:** גם אם עוד אין צוות ייעודי, מישהו צריך להיות אחראי על ה-skills וה-context.

שאלה פתוחה	למה זה חשוב
איך מזהים קונפליקט כוונה בצורה אמינה?	הדובר לא הראה כלי או שיעור הצלחה. false positives עלולים לחסום את התור.
כמה עולה ה-CI בפועל ב-multiplayer factory?	נאמר שזה "more costly", אבל בלי מספרים.
מי מכריע כש-agents מנהלים משא ומתן מחדש על תוכנית?	לא הוסבר מי מאשר שינוי בחלוקת העבודה תוך כדי ביצוע.
איך שומרים על ה-decision ledger קטן ושמיש?	log שגדל בלי סוף פוגע ב-context במקום לעזור לו.
מה מוכן היום ב-Tessl ומה עדיין מחקר?	הדובר עצמו אומר "We're not there yet".

8. מילון מונחים

מונח	פירוש
Software factory	pipeline שבו coding agents מבצעים חלק גדול מהעבודה, מתכנון ועד merge
Solo / multiplayer factory	factory של מפתח בודד על הלפטופ, לעומת factory שכמה אנשים ו-agents חולקים
Concurrent delivery	המונח של הדובר ל-delivery שבנוי מראש לשינויים מקבילים כשגרה
Worktree	עותק עבודה נפרד של ה-repo, משמש לבידוד agents
AST graph	גרף תחביר של הקוד, משמש לזיהוי חלקים שקל להפריד
Digital twin	העתק מדומה של מערכת חיצונית לצורך בדיקות
Intent conflict	שינויים שמתמזגים בלי קונפליקט קוד אבל סותרים זה את זה בכוונה
Speculative merge train	merge queue שבודק PRs בצירופים מראש, כך שכישלון של אחד לא חוסם את השאר
Decision ledger	יומן החלטות משותף ל-agents ולאנשים
Gradual release	הפצה הדרגתית של שינוי עם מעקב אחרי שגיאות
Factory team	צוות דמוי platform team שאחראי על ה-skills, ה-context והתחזוקה של ה-factory