# לבנות תוכנה אחרת: Kiro ופיתוח מונחה־ספק

AWS Summit Tel Aviv 2026 ,סיכום סשן — TLV-DVT202

Hila Fish, Solutions Architect, AWS | Adir Gozlan, Sr. Specialist TAM AI/ML, AWS | Shon Paz, VP Platform, בנק הפועלים

10 בספטמבר 2026 | משך: כ־39 דקות | הופק מהקלטת הסשן

מסלול: AWS for Developers in the AI Era

# על המסמך הזה

המסמך מסכם את הסשן TLV-DVT202 מתוך AWS Summit Tel Aviv 2026. הוא נבנה מתוך התמלול המלא של ההקלטה ומהסליידים שחולצו מהווידאו עצמו, כך שאפשר לחזור לתוכן בלי לצפות שוב בכל ההרצאה. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

ההרצאה ניתנה בעברית בשלושה קולות: שני דוברים מ־AWS שהציגו את הכלי ואת המתודולוגיה, ולקוח מבנק הפועלים שסיפר מה מזה באמת רץ אצלו בפרודקשן. שמות המוצרים והפיצ'רים בגוף המסמך נורמלו לאנגלית.

### איך לקרוא את זה

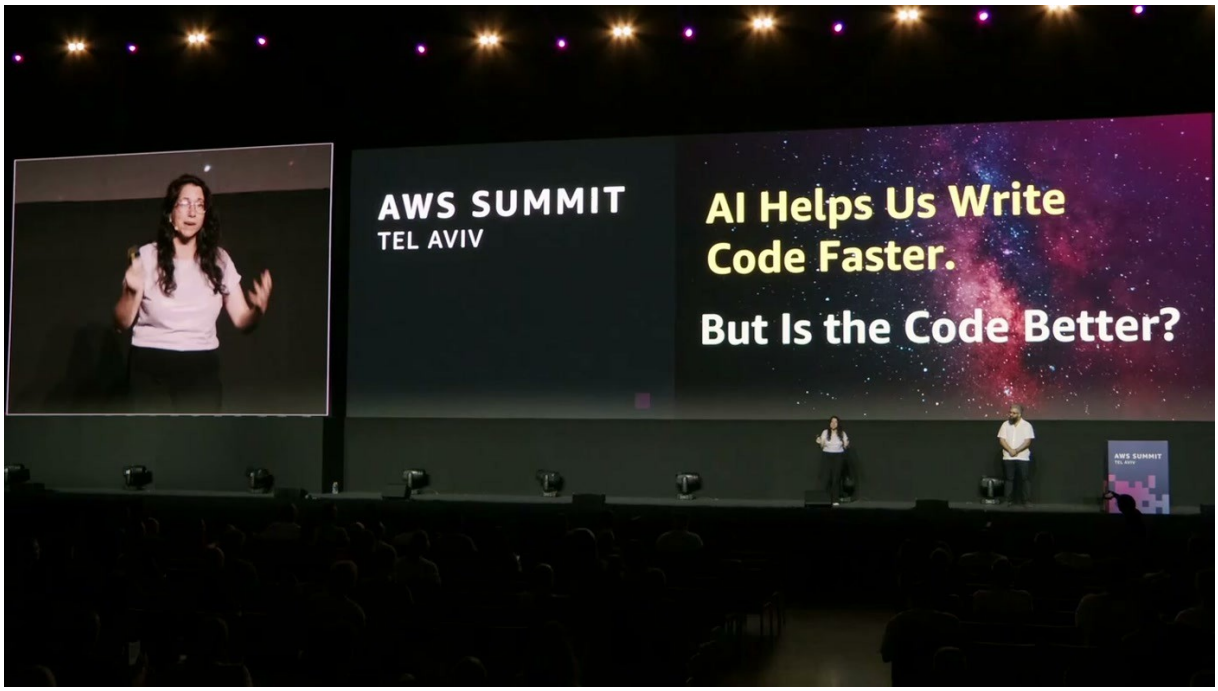
- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. הטענה המרכזית של ההרצאה והמספרים שהוצגו כתמיכה.
- **פרקים 2–3 — הבעיה:** הפרדוקס של מהירות מול איכות, והמעבר מפרודוקטיביות אישית לארגונית.
- **פרקים 4–6 — המתודולוגיה והכלים:** Spec-Driven Development, Powers, ו־Steering Files.
- **פרק 7 — מהשטח:** בנק הפועלים — מודרניזציה של מיינפריים ו־agent לתמיכה ב־CI/CD.
- **פרקים 8–9 — סביב ה־IDE:** Kiro CLI, iOS, Web, Kiro Crew, ושליטה ארגונית.
- **פרק 10 — מה לוקחים מכאן:** המסקנות כפי שהדוברים עצמם ניסחו אותן.

> **הערת מקור** המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך מונחים לועזיים ושמות מוצרים הופיעו בו בשיבוש פונטי ותוקנו ידנית מול הסליידים. כל ציטוט אומת מול ההקלטה בחותמת הזמן המצוינת. מספרים ונתוני מחקר מובאים כפי שהוצגו על הבמה ולא אומתו מול המקורות המקוריים.

# 1. תקציר מנהלים

- **מהירות אינה איכות — וזה הפער שההרצאה מנסה לסגור.** הדוברים פתחו בשאלה אם הקוד שנכתב מהר יותר הוא גם קוד טוב יותר, וענו בפירוש שלא. הנתונים שהוצגו: כמות הקוד שנכתב עלתה ב־741% לפי מחקר של MIT ו־Wharton, אבל השילוח לפרודקשן עלה ב־20% בלבד [3:07].
- **ה־10x Developer עדיין מיתוס.** לפי מחקר של METR שהוצג על הבמה, השיפור בפועל הוא בסביבות פי 1.4 [3:07]. בסקר GitLab שצוטט, 79% מהמפתחים דיווחו שהם מהירים יותר — אבל 82% דיווחו שהתוצר מביא איתו בעיות [7:43].
- **הפתרון המוצע הוא לא עוד יכולת, אלא חוזה.** Kiro — Spec-Driven Development שואל שאלות לפני שהוא כותב שורת קוד, ומייצר שלושה קבצים: requirements.md, design.md, tasks.md. רק בשלב השלישי נכתב קוד [10:50–9:19].
- **ההבחנה המעניינת היא בין פרודוקטיביות אישית לארגונית.** Steering Files ו־Powers, Custom Powers קיימים כדי שאותה קונפיגורציה בדיוק — ספריות, קונבנציות, lint rules, כלים פנימיים — תגיע לכל מפתח בארגון בכמה קליקים [16:57–13:52].
- **בנק הפועלים הוא ה־proof point, והוא לא תיאורטי.** הבנק משתמש ב־AWS Transform ו־Kiro כדי להוציא קוד COBOL מהמיינפריים לספקים עסקיים ומשם לקוד מודרני. שירות אחד כבר עלה לאוויר על EKS ו־RDS ב־landing zone שלהם [21:33].
- **ההמלצה החוזרת של הלקוח היא לא על הכלי אלא על ההקשר.** ההשקעה שמשתלמת היא ב־steering, ב־context management וב־harness engineering — ככל שההקשר מדויק יותר לארגון, ה־agent מיושר יותר אליו [33:47].

## 2. הפרדוקס: יותר קוד, לא יותר מוצר

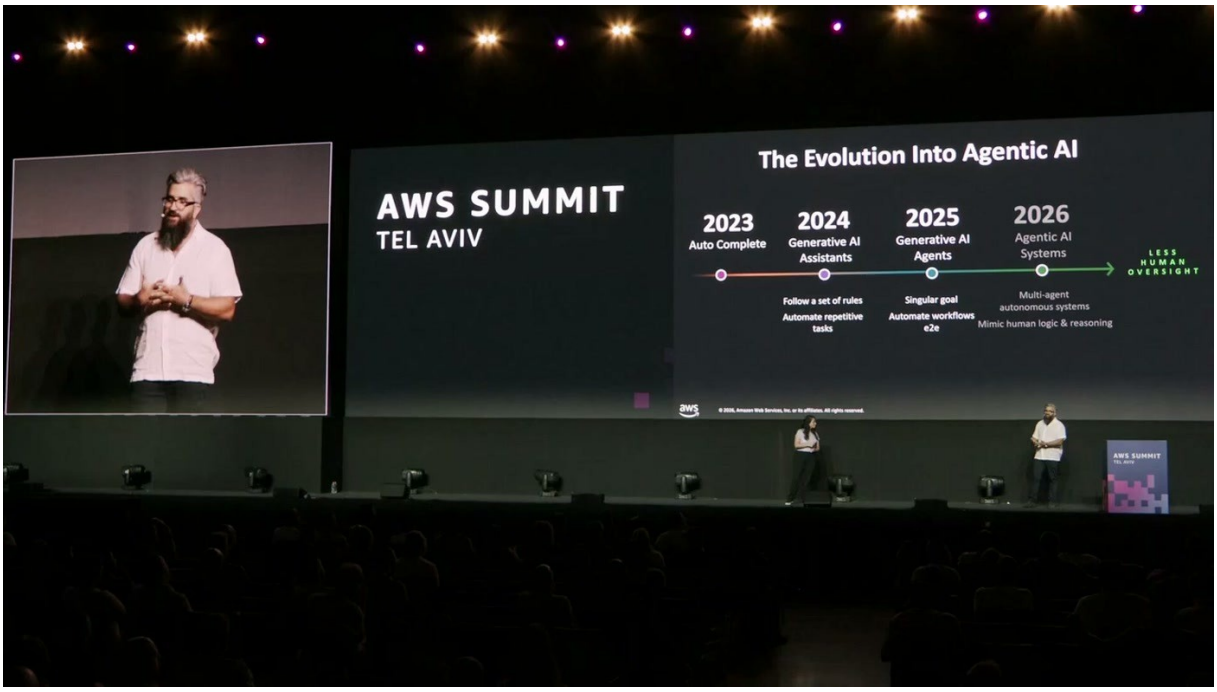



*שקופית הפתיחה מציבה את השאלה שההרצאה כולה תלויה בה — לא האם ה־AI מהיר, אלא האם התוצר טוב יותר.*

ההרצאה נפתחה בהשוואה: לפני שלוש שנים מפתחת שרצתה להביא מוצר לפרודקשן ישבה, תכננה, כתבה קוד, הריצה טסטים ותיקנה בלולאה. היום, נאמר, מפתח יושב מול CLI או IDE ותוך שניות ספורות יש לו קוד שכנראה עובד. השאלה היא מה קורה אחר כך.

> **— [0:00]** אז האם AI עוזר לנו לכתוב קוד מהר יותר? אין ספק. אבל האם הקוד, התוכנה עצמה, האפליקציה, השירות שאנחנו מעלים לפרודקשן, האם הוא טוב יותר ואיכותי יותר? זו כבר שאלה אחרת.

הטענה נתמכה בקו זמן קצר של ההיכרות שלנו ככלל התעשייה עם כלי AI: 2023 שנת ה־Auto Complete, 2024 שנת ה־Assistants (שני חלונות פתוחים, IDE וצ'אטבוט, והרבה מאוד copy-paste), 2025 שנת ה־Agents — שבה היכולות עברו ממתן תשובות לפעולה סביב מטרה, ואיתן הגיע האקוסיסטם: MCP, skills, harnesses, memory. 2026 הוצגה כשנת ה־AI Systems: מערכות של agents שעובדות יחד, לוקחות משימות end-to-end, ודורשות פחות ופחות מעורבות אנושית [1:32–3:07].




*קו הזמן שהוצג: מ־Auto Complete ב־2023 עד Agentic AI Systems ב־2026, עם החץ שמסמן פחות פיקוח אנושי.*

ואז הגיעו המספרים. מחקר של MIT ו־Wharton שעקב, לדברי הדוברת, אחרי יותר מ־100 אלף מפתחים, מצא שכלי AI הגדילו את כמות הקוד שנכתב ב־741% — אבל השילוח לפרודקשן עלה רק ב־20%.

> **— [3:07]** הם מצאו שכלי AI הגדילו את כמות הקוד שנכתב ב-741% — זה מספר בלתי נתפס — אבל השיפינג לפרודקשן עלה רק ב-20%.

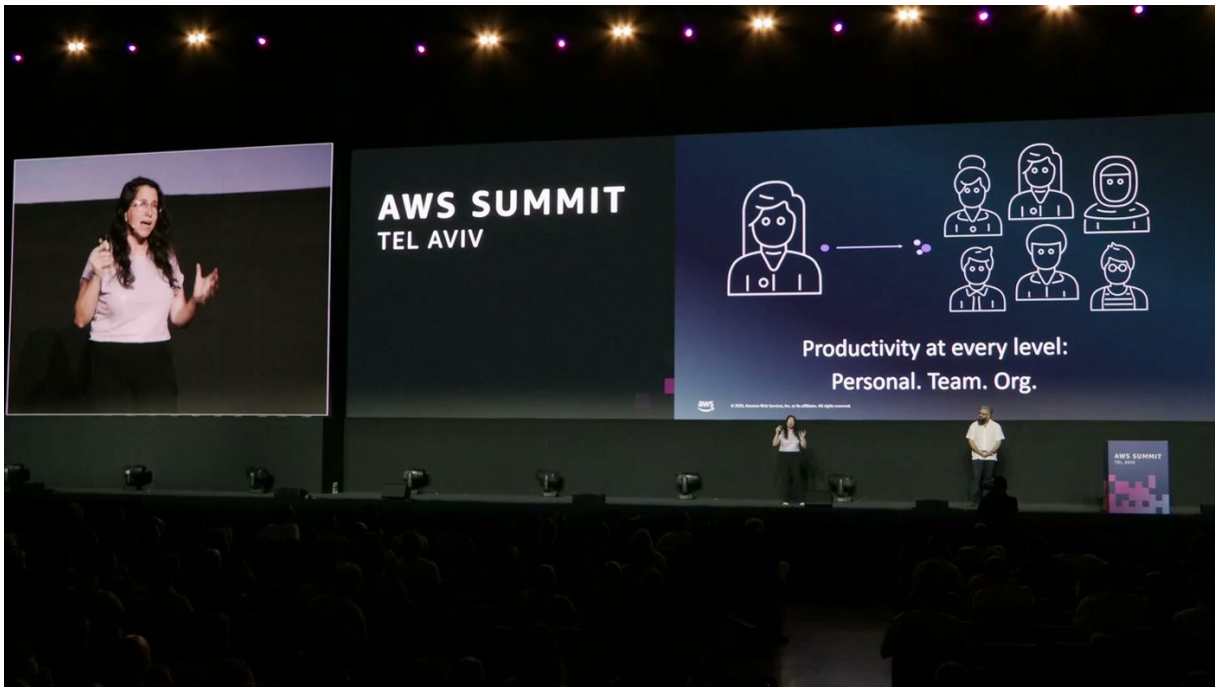
לצד זה הוצג מחקר של METR על מיתוס ה־10x Developer: העלייה בפועל היא בסביבות פי 1.4 [3:07]. הדוברת הוסיפה שהיא מאמינה שנגיע לשם, ושהיא רואה לקוחות מתקדמים לשם — אבל "אנחנו עדיין לא שם".

המחקר השלישי שהוצג היה דוח GitLab משנת 2026: 79% מהמפתחים דיווחו שהם מהירים יותר אך שהתוצר לא נכנס למוצר, ו־82% דיווחו שהתוצר מביא איתו בעיות. הניסוח על הבמה היה חד:

> **— [7:43]** אמנם אנחנו הוספנו כאן המון המון יכולות, אבל זה עדיין וייב קודינג. וייב קודינג על סטרואידים.

האבחנה הטכנית שנלוותה לכך: ה־coding agents מאופטמים לעשות דברים, לא לפשט אותם. התוצאה היא הרבה דופליקציה, מעט מאוד עבודת refactoring, והצטברות מהירה של חוב טכני בקצב שהארגון לא ערוך אליו [7:43].

# 3. מפרודוקטיביות אישית לפרודוקטיביות ארגונית




*שלוש הרמות שהדוברים טענו שכל כלי AI צריך לתת עליהן מענה בו־זמנית.*

אחת הסיבות שהוצגו לפער היא ה־mindset. הדגש המקובל הוא על הכלים ועל הסקילים האישיים, אבל מפתח לא עובד בוואקום: יש צוות, יש תהליכי change management, ויש סטנדרטים ארגוניים שאם הקוד לא עומד בהם — הוא פשוט לא יגיע לפרודקשן.

> **— [4:40]** אנחנו צריכים לזכור שאנחנו לא נמצאים לבד. יש לנו אקוסיסטם שלם, צוותים לעבוד איתם, תהליכים שלמים לעבוד איתם.

מכאן נגזרה דרישת המסגרת שלפיה נבחן Kiro לאורך ההרצאה: כלי צריך לתת מענה בפרודוקטיביות ובאפקטיביות בשלושת הרבדים — האישי, הצוותי והארגוני [4:40].

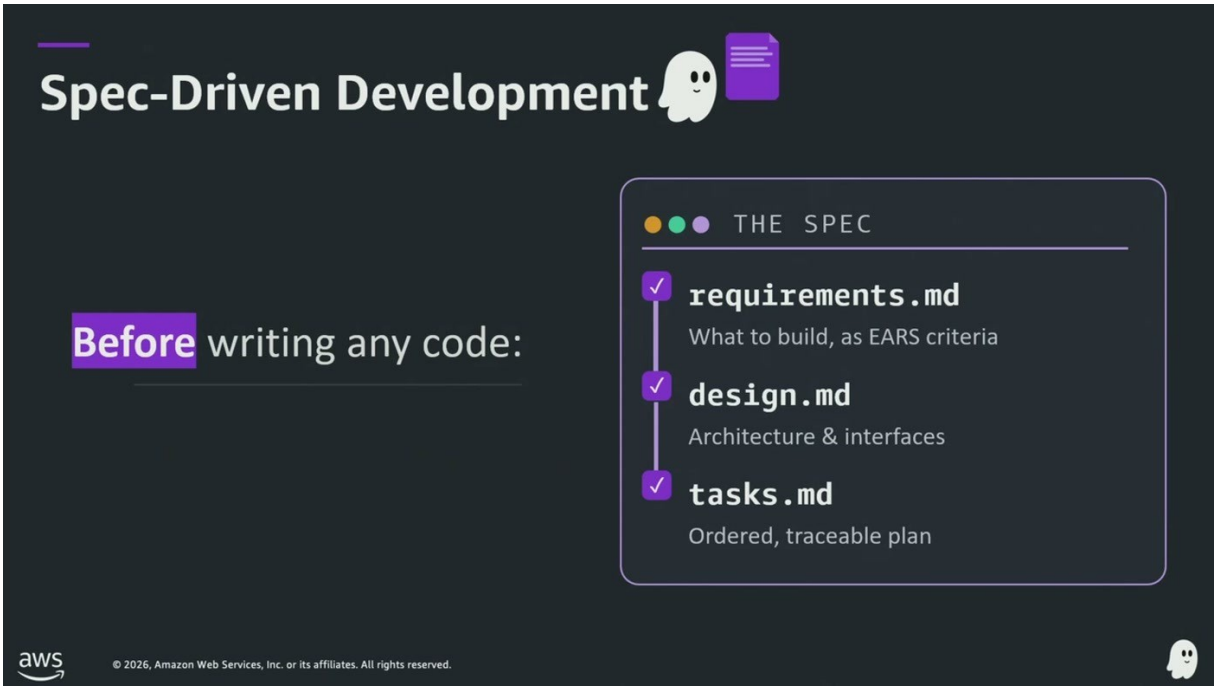
## 3.1 מה המפתחת מקבלת ביום הראשון

החלק הראשון של ההדגמה עבר על ההגדרות שמפתחת בודדת עושה לעצמה ב־Kiro IDE, בסדר שבו היא באמת תעשה אותן [6:11]:

- **MCP Servers —** חיבור לכלים חיצוניים. בדוגמה שהוצגה: GitHub כדי למשוך את הריפוזיטורי, ו־Jira כדי למשוך טיקטים וטאסקים. ההגדרה היא בקובץ, ולאחר ה־config כל הטולים זמינים.
- **Skills —** הנחיות שמגדירות פרסונות, טאסקים ואופן עבודה, ונטענות באופן יזום לפי ההקשר.
- **Steering —** הנחיות שמגדירות איך להשתמש ב־MCP server ביעילות, גם מבחינת עלות וגם מבחינת ביצועים.
- **Hooks —** אוטומציות. הדוגמה שניתנה: לעדכן דוקומנטציה או קובצי לוקליזציה תוך כדי כתיבת הקוד. אפשר להגדיר אותן בשפה חופשית, ו־Kiro כותב את האוטומציה עצמה.

הנקודה של הדוברים הייתה שבשלב הזה נדמה שהמפתחת סיימה — יש לה setup, יש לה כלים, היא יכולה לכתוב פרומפט ולקבל קוד. וזה בדיוק המקום שבו, בלשונה, "all hell breaks loose" כשמגיעים לפרודקשן [7:43].

# 4. Spec-Driven Development: לבנות חוזה לפני שכותבים קוד




*שלושת הקבצים שנכתבים לפני שורת הקוד הראשונה, ומה כל אחד מהם אמור להכיל.*

התשובה של Kiro לפער הזה היא מתודולוגיה מובנית בכלי, ולא פיצ'ר נפרד. השינוי שהוצג הוא במיינסט: מלהגיד לכלי מה לעשות, למצב שיתופי שבו המפתחת והכלי עובדים יחד לעבר תוצאה משותפת [9:19].

> **— [9:19]** אני אומרת: אני רוצה לפתח איזשהו פיצ'ר. במקום שקירו ירוץ לכתוב קוד, הוא שואל אותי שאלות, הוא מנסה להבין מה אני מנסה להשיג כאן.

הדוגמה שניתנה ממחישה למה זה לא טקס ריק: בבקשה "תוסיף תמיכה לאתר בספרדית", Kiro שואל באיזה ניב של ספרדית מדובר — שאלה שמשפיעה ישירות על התוצר, ושהמפתחת עצמה אולי לא הייתה שואלת. אם אין לה תשובה, זו נקודה טובה להביא product owner לשולחן [9:19].

רק אחרי שלב השאלות נכתב הספק, בשלושה חלקים:

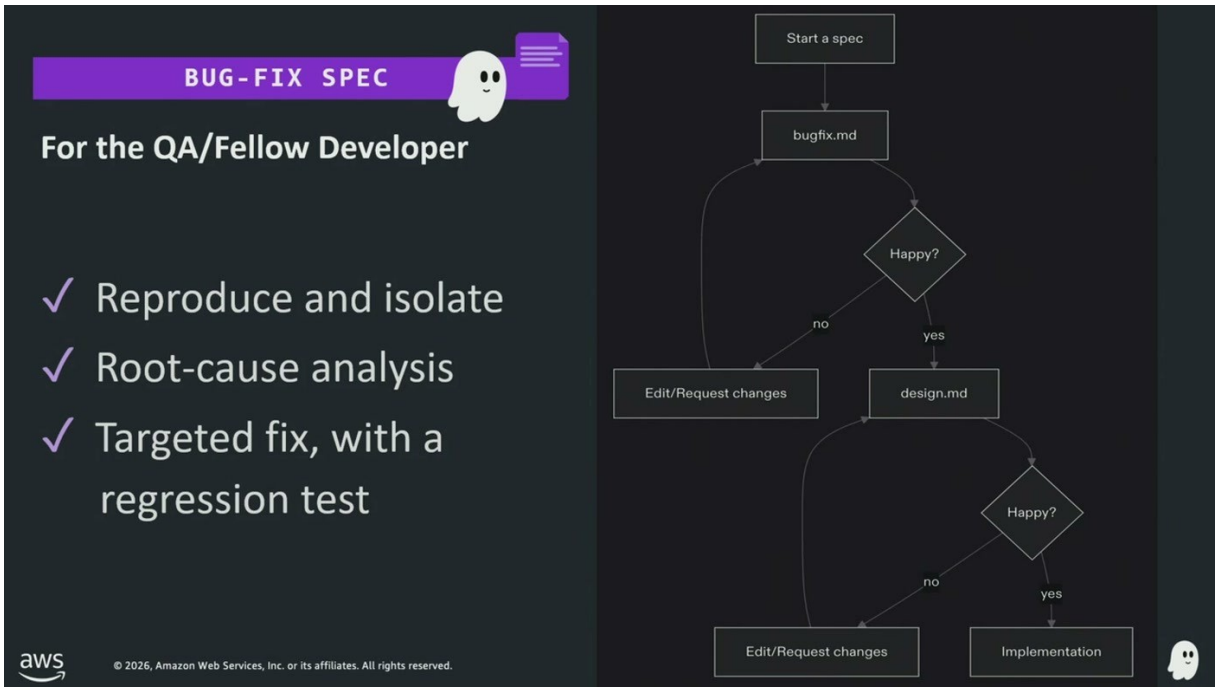
- **requirements.md —** החלקים העסקיים: על מה הפיצ'ר בא לענות, user stories ו־acceptance criteria.
- **design.md —** הארכיטקטורה, האינטרפייסים, API routes, טיפול בשגיאות וטסטים — שנכללים בצורה נייטיבית ולא כמחשבה שנייה.
- **tasks.md —** תוכנית מסודרת וניתנת למעקב של איך הפיצ'ר ייבנה. רק כאן נכתבת שורת הקוד הראשונה.

השליטה נשארת אצל המפתחת לאורך כל התהליך: אפשר לעצור בכל שלב ולומר שזה לא מה שהתכוונו אליו, ואפשר להריץ את הטאסקים בבת אחת או אחד־אחד, לפי כמה פיקוח רוצים [10:50].

> **— [10:50]** סוג של בניית חוזה עם קירו, לפני שאנחנו מתחילים לעבוד.

## 4.1 פלייבורים שונים לאותו חוזה

ההערה החשובה כאן היא שלא כל חוזה מתאים לכל סוג עבודה. Kiro מגיע עם מספר וריאציות של תהליך הספק out of the box [12:20]:




*הוריאציה ל־QA ולתיקון באגים — שלושה שלבים שמחליפים את ה־user stories, לצד תרשים הזרימה של התהליך.*

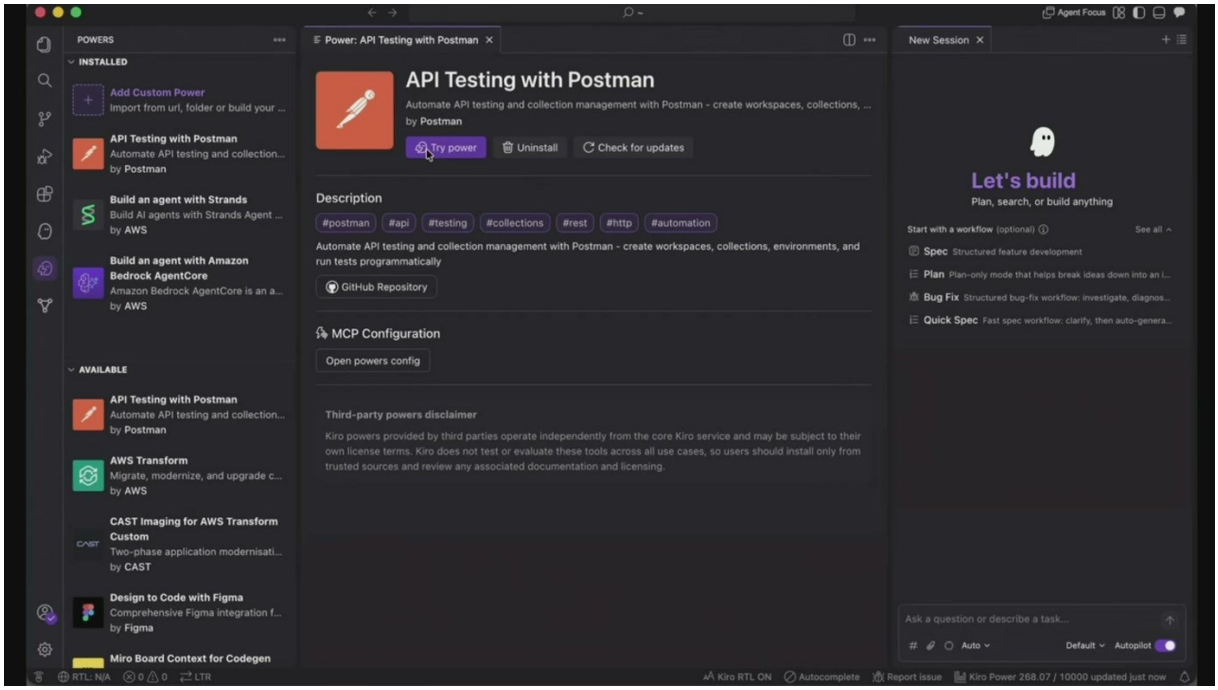
- **Bug-Fix Spec —** למפתח או לאיש QA שבא לתקן באג. user stories ו־business requirements לא רלוונטיים כאן; מה שרלוונטי הוא reproduction, root-cause analysis ו־fix ממוקד עם regression test.
- **Design-First Spec —** לארכיטקט שמביא טכנולוגיה ולא פיצ'ר. מתחילים מדיאגרמת ארכיטקטורה, ו־Kiro עושה לה reverse engineering כדי להסיק ממנה את ה־requirements וה־user stories.

המכנה המשותף, כפי שנוסח על הבמה: כל האפשרויות סביב Spec-Driven Development קיימות כדי לאפשר קולבורציה עם ה־AI — לנסח בצורה מפורטת יותר, וממספר פרסונות, לאן רוצים להגיע לפני שמתחילים [13:52].

## 5. Powers: אריזה של best practices לרמה ארגונית


*שקופית המעבר שמסמנת את חצי ההרצאה השני — מהפרט לארגון.*

הבעיה שהפיצ'ר בא לפתור הוגדרה במפורש: כשמפתחת רוצה להשתמש ב־MCP server, היא צריכה לעשות את האינטגרציה בעצמה, להכיר את הטולים שבתוכו, לדעת מתי להפעיל כל אחד, ולכתוב steering כדי שההקשר לא יתנפח. Power הוא bundle שחוסך את כל זה [13:52].




*קטלוג ה־Powers בתוך ה־IDE — כל שורה היא חבילה מפרטנר, כאן נפתחה זו של Postman.*

החבילה מגיעה מפרטנרים (בדוגמאות שהוזכרו: Figma, Postman) וכוללת שלושה דברים יחד — את ההתקנה והאינטגרציה של ה־MCP server, את קובץ ה־steering שמסביר איך להשתמש בו בחוכמה מבחינת עלות וביצועים, ואת ה־hooks המשלימים.

**[13:52]** — בקליק אחד אנחנו מקבלים בנדל שמכיל גם את האינטגרציה ואת ההתקנה של ה־MCP סרבר, אבל גם את ה־steering file.

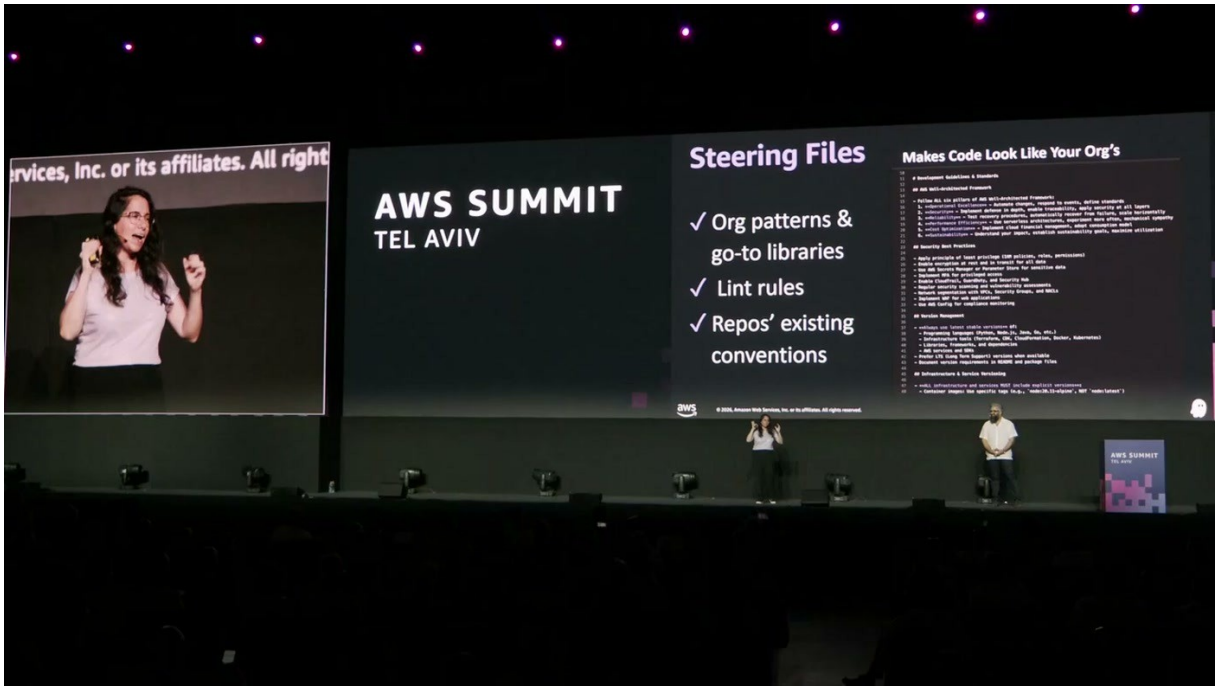
### 5.1 Custom Powers — שני דפוסי שימוש שנצפו אצל לקוחות

- **התאמה של כלי פרטנר לארגון:** לקחת Power קיים, להוריד ממנו יכולות, להשאיר רק את הסט הרלוונטי, ולארוז מחדש — כך שכל מפתח בארגון מקבל את אותה גרסה עם אותם best practices מוטמעים [15:26].
- **עטיפה של כלים פנימיים:** Jira on-prem, או כלי שהארגון בנה בעצמו. עוטפים אותו במעטפת של Kiro Power, מוסיפים את ה־best practices הפנימיים, ונותנים ל־agentic coders גישה אליו בצורה מבוקרת [15:26].

הנקודה שהודגשה בשני המקרים היא לא היכולת אלא ההפצה:

> **— [15:26]** כל מפתח ומפתחת שיש לו וורקסטיישן עם קירו — כמה קליקים ויהיה להם את אותה גרסה של אותו כלי בדיוק.

## 6. Steering Files מול Skills




*מה נכנס לקובצי ה־steering, ולצדם דוגמה לקובץ הנחיות ארגוני אמיתי.*

החלק הזה הבהיר הבחנה שחזרה לאורך כל ההרצאה. שני המנגנונים נותנים הנחיות ל־Kiro, וההבדל הוא באופן הטעינה:

| | Skills | Steering Files |
|---|---|---|
| מתי נטענים | באופן יזום, לפי טאסק או פרסונה | אוטומטית — פר פרויקט או ברמה רחבה |
| למה משמשים | להלביש על ה־agent פרסונה או יכולת למשימה מסוימת | כללי התנהלות שתמיד יחולו, לא משנה על איזה ריפו הוא עובד |
| איפה הערך הארגוני | התמחות נקודתית | אחידות: patterns ארגוניים, go-to libraries, lint rules, קונבנציות קיימות |

> **— [16:57]** סקילס — שהוא טוען באופן יזום. Steering Files הוא טוען בצורה אוטומטית, או פר פרויקט או בצורה רחבה.

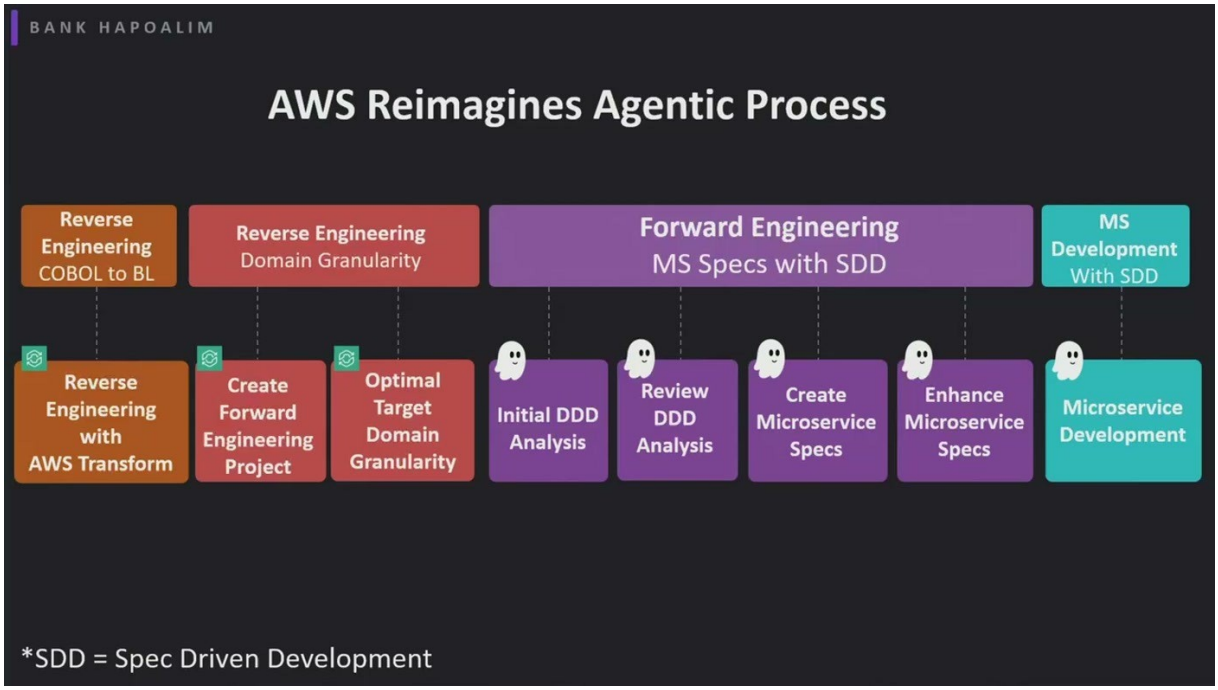
הניסוח המעשי שניתן: Steering Files הם המקום להגדיר את ה"עשה ואל תעשה" הארגוני — איך אתם רוצים ש־Kiro יתנהג כמפתח מן המניין, יפעיל reasoning בהתאם, ויכתוב קוד כמו שהארגון שלכם רוצה [16:57].

# 7. מהשטח: בנק הפועלים

בשלב הזה עלה לבמה Shon Paz, VP Platform בבנק הפועלים, ששיתף שני use cases נפרדים. הוא פתח בשאלה לקהל מי מכיר את המילה "מיינפריים", ומיד אחריה — מי מכיר COBOL.

## 7.1 Transform & Reimagine — הוצאת COBOL מהמיינפריים

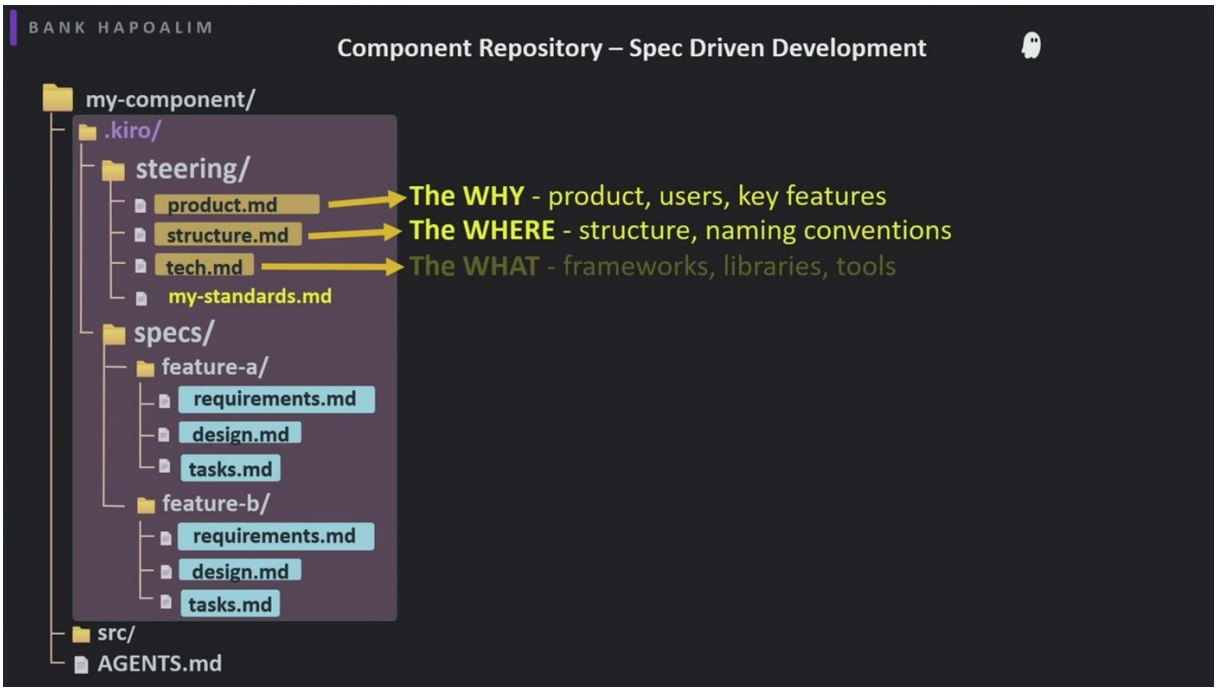
ההחלטה התקבלה בבנק לפני כשנה־שנתיים, בעקבות re:Invent. הבעיה שהוצגה היא בראש ובראשונה בעיית ידע: רוב הקוד במיינפריים נכתב ב־COBOL, אין הרבה ידע סביבו, והאנשים שיודעים לכתוב אותו כבר לא נמצאים בתעשייה. התוצאה, לדבריו, היא שהרבה מהדומיינים העסקיים בבנקים — בישראל ובעולם — מתקשים להתקדם קדימה מעצם השימוש במיינפריים [18:30–20:02].




*התהליך המלא של הבנק: reverse engineering מ־COBOL לשפה עסקית משמאל, ומשם forward engineering לספקים ולמיקרו־שירותים.*

התהליך עצמו מורכב משני חצאים. בחצי השמאלי, AWS Transform קורא קוד COBOL שנכתב לפני 50 שנה ומפיק ממנו מבנה של business functions ודומיינים עסקיים — משכנתאות, שוק ההון וכדומה. בחצי הימני, אותם דומיינים ממופים לספקים, ומהספקים — באמצעות Spec-Driven Development — נוצר קוד מודרני [20:02–21:33].

**התוצאה בפועל** הרגע שזכה למחיאות כפיים מהקהל: שירות אחד כבר עלה לאוויר בקונסטלציה הזאת — "זה ממש שירות שיושב על EKS עם RDSים בתשתיות AWS שלנו בלנדינג זון" [21:33]. הדובר הדגיש שזו לא עבודה שלו לבדו אלא של אנשים רבים בבנק.



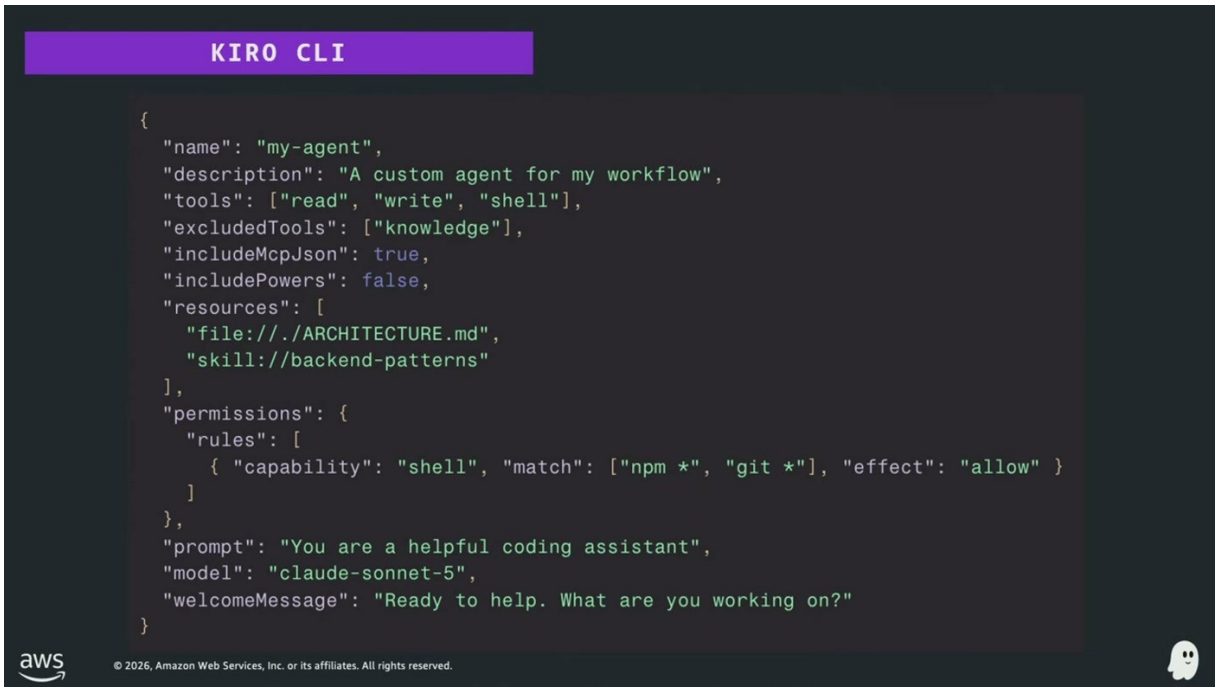

*מבנה הריפוזיטורי שהבנק אימץ: ספריית kiro. עם steering ו־specs לצד קוד המקור, והחלוקה ל־WHY, WHERE ו־WHAT.*

הדרך שבה זה נשזר בעבודה היומיומית היא דרך הריפוזיטורי: הבנק לקח את ה־SDLC הקלאסי ושזר לתוכו את כל מה שהוצג בחלק הראשון — steering עם הסטנדרטים, הקונבנציות והסטאק, ולצדו ספקים שכל אחד מהם מכיל requirements, design ו־tasks. היתרון שהודגש הוא בהתחלה של מפתח חדש:

> **— [21:33]** החל מהרגע שעשיתי git clone, אני מקבל את כל הקונטקסט וההארנס מסביב, ואני פשוט יכול להתחיל לפתח.

# 8. Kiro CLI, Custom Agents, וה־agent שפותח טיקטים

החזרה לבמה של דוברי AWS עברה מה־IDE ל־CLI. הטענה לא הייתה שיש שם יכולות אחרות — מדובר באותו סט יכולות — אלא שה־form factor פותח דברים אחרים: כתיבת pipelines, סקריפטים, ו־automated workflows שרצים לבד [23:05].

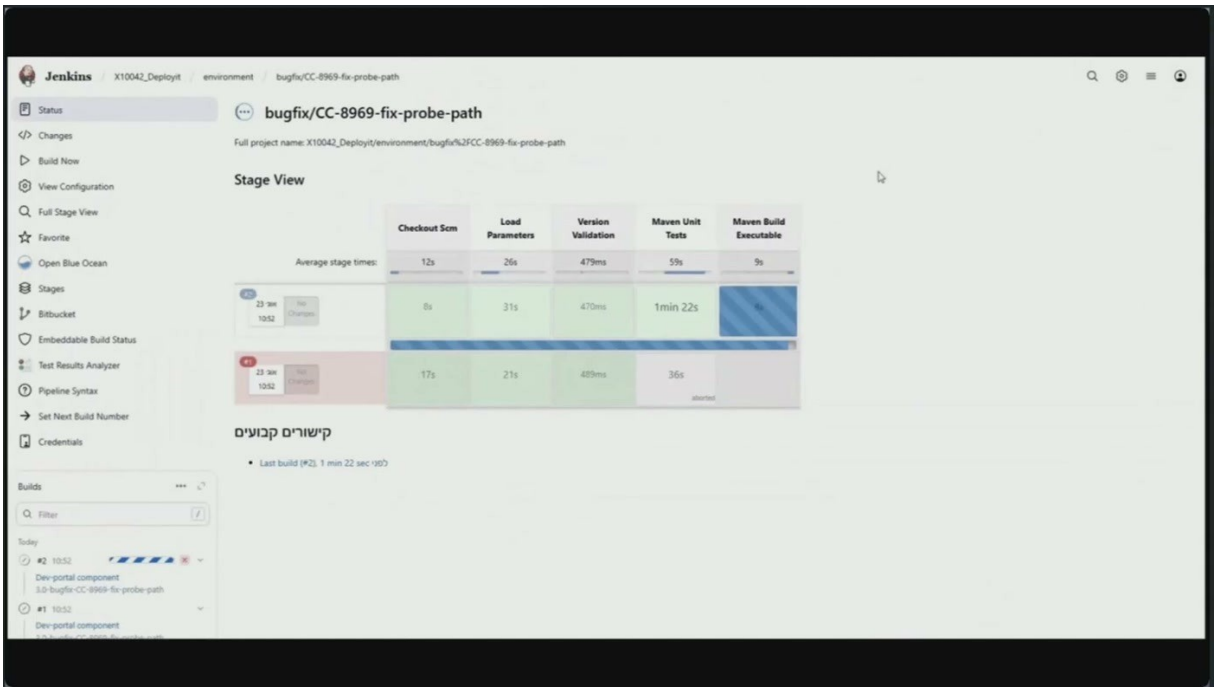



*קובץ ההגדרה של Custom Agent — טולים מותרים ואסורים, הרשאות shell, משאבים, ומודל.*

המנגנון הוא Custom Agents — אפשר לחשוב עליהם כפרסונות. הם קיימים גם ב־IDE, אבל דרך ה־CLI אפשר לטריגר אותם מתוך תהליכים וכלים אחרים. הדוגמה שניתנה: agent בשם Code Reviewer עם system prompt משלו, שמופעל מ־GitHub Action כך שבכל פתיחת PR הוא עושה review, בודק, ומציע חלופות ופתרונות [23:05–24:37].

## 8.1 הדמו של בנק הפועלים: Customer Success Agent ל־CI/CD

ה־use case השני שהוצג מהשטח נולד ממה שהדובר כינה בעיית כוח אדם. צוות ה־CI/CD בבנק מקבל זרם קבוע של טיקטים, ורבים מהם חוזרים על עצמם — בעיות שכבר יודעים איך פותרים, שנפתחות שוב כי יש מפתח חדש. המטרה הייתה לעשות offload לצוות כדי שיתעמק בדברים חשובים יותר [24:37].




*מסך מתוך הדמו החי — ה־agent חוקר את ה־pipeline שנכשל ומייצר ניתוח שורש.*

מה שה־agent עושה, לפי ההסבר שליווה את הדמו: הוא מקבל את הטיקט, מתחבר לכלים לאורך ה־SDLC — Jenkins, Argo CD, ו־OpenShift שבו רצים הקונטיינרים — ועובר מערכת־מערכת וקומפוננטה־קומפוננטה כדי להפיק root cause analysis. את ההקשר הוא שולף באמצעות MCP servers ו־Kiro Powers, והניתוח מוחזר לטיקט עצמו [26:08].

> **— [26:08]** האג'נט הזה גם דוחף PR... נותן כמובן את ה-Root Cause Analysis, מה הבעיה בפועל, וברגע שהמפתח ממרג'ג' בעצם את הקוד, מתחיל תהליך ה-CI המתוקן.

השאיפה שהוצהרה היא שהמפתח יצטרך רק למרג'. הדובר הדגיש שהדמו הוקלט מסביבת הבנק ושהתקלות אצלם מתועדות בעברית. בסיום הדמו צוין שה־CI אצלם לוקח כמה דקות והוקרן ב־fast forward [26:08].

## 9. מעבר לשולחן העבודה: iOS, Web ו־Kiro Crew

החלק הזה סקר שני surfaces נוספים ומוצר שלישי, במטרה להראות כיצד התהליך נעשה אוטונומי יותר [27:39]:

- **Kiro ל־iOS (בפריוויו):** לפתוח את האפליקציה בפגישה או בדרך למשרד, לחבר ריפוזיטורי, להתחיל תהליך agentic מהטלפון — ולהגיע למשרד כשהוא ממשיך לרוץ.
- **Kiro Web (זמין):** ה־environment המרוחק שבו הדבר הזה בעצם רץ. סנדבוקס מרוחק למשימות שרוצים בהן agentic coding בלי אינטראקציה — עבודות backlog, דברים שכבר יודעים שעובדים לבד.
- **Kiro CLI headless:** אפשר לטריגר את התהליך בלי אדם בלולאה כלל.

החוסר שזוהה על הבמה היה agents: long-lived sessions שרצים לאורך זמן, עושים משימות, וצריכים זיכרון של מה שקרה קודם. התשובה לכך היא Kiro Crew.

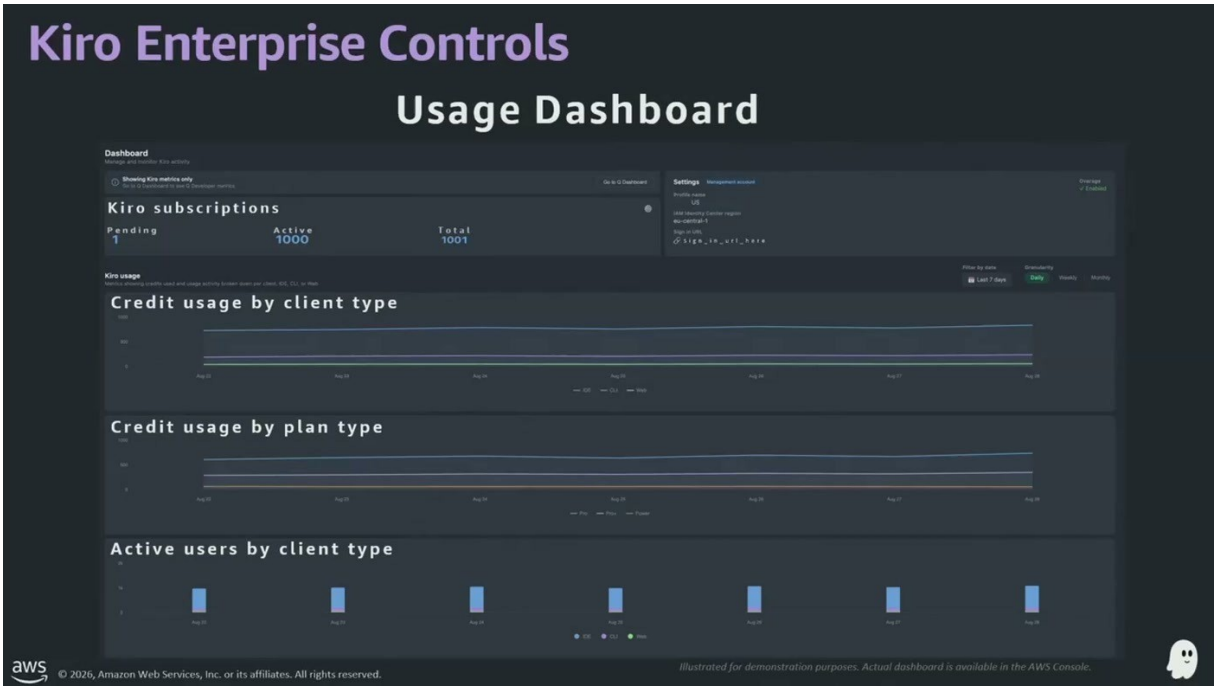

*ממשק הניהול של Crew — לוח משימות מתוזמן ל־agents שרצים לאורך זמן.*

Crew, שלפי ההרצאה שוחרר כ־open source כחודש לפני הסאמיט, מותאם לסשנים ארוכי חיים: מרימים מכונה שהדבר רץ עליה 24/7, ומנהלים באמצעותה crews — סט של agents שעובדים יחד על פרויקט או צוות מסוים, עם skills ויכולות שהוגדרו להם. הידע שלהם מנוהל לאורך זמן ב־knowledge graph שנבנה מאחורי הקלעים, וה־skills הספציפיים ל־crew נקראים lessons [29:11–30:42].

**— [30:42]** קירו קרו התחיל כמוצר פנימי. תוך כמה חודשים — עשרות אלפי משתמשים, יותר מ-500 קונטריביוטרס, מה שדחף את AWS לשחרר את זה כאופן סורס.

# 10. Enterprise Controls: ניהול, עלות וקישוריות פרטית

החלק האחרון עבר מההיבט הפונקציונלי לאדמיניסטרטיבי, בשלושה רבדים.




*הדשבורד כפי שהוצג — צריכת קרדיטים לפי client type ולפי plan type, ומשתמשים פעילים.*

- **ניהול סאבסקריפשנים:** הקצאת משתמשים או קבוצות, והחלפת tiers — הכל מתוך ה־AWS Management Console. הטווח שצוין: ממשתמש בודד ועד אלפי משתמשים, "כמו שיש לבנק הפועלים" [30:42].
- **Usage Dashboard:** ויזיביליות ומוניטורינג כדי לדעת שהערך תואם לציפיות — צריכת קרדיטים לפי client type ולפי plan type, משתמשים פעילים, ו־acceptance rate של הקוד. הודגש שהמסך שהוצג הוא אילוסטרציה ולא נתוני לקוח אמיתיים [30:42].
- **Private connectivity:** עבור דרישות אבטחה ורגולציה — הגדרת VPC Endpoints ב־VPC של הלקוח ובייפאס לפרוקסי, כך שכל התעבורה בין המפתחים ל־Kiro עוברת ב־PrivateLink ולא בתעבורה הציבורית [32:13].

## 10.1 מודל התמחור: קרדיטים במקום טוקנים

נקודה שהוצגה כהבדל תפיסתי: בעוד כלים רבים בשוק עובדים לפי token usage, Kiro עובד לפי credit usage. הטענה שצוטטה על הבמה מיוחסת לאנדי ג'סי, לפיה Kiro יכול להיות "up to 50% יותר cost effective" מול אלטרנטיבות [32:13].

> **כלי לבדיקה עצמאית** הדוברים הפנו ל־Agent Cost Bench — כלי open source שמאפשר benchmarking בין Kiro לכלים אחרים לפי סוגי טאסקים שונים, והמליצו לבדוק איתו מה נותן את הערך הטוב ביותר במקום להסתמך על ההצהרה [32:13].

# 11. מה לוקחים מכאן

ההרצאה נחתמה בשני סטים של takeaways — אחד מהלקוח ואחד מ־AWS. השילוב ביניהם הוא כנראה החלק השימושי ביותר של הסשן.

## 11.1 מהצד של הלקוח

> **— [33:47]** מישהו פעם חכם אמר לי: הטכנולוגיה היא לא כאן לשם הטכנולוגיה.

האזהרה הראשונה הייתה נגד הנטייה להכריח agent לכל use case שנראה מעניין. יש דברים שאפשר לפתור בצורה פשוטה מאוד, ולהחלטה יש משמעויות רחבות יותר — עלויות, היבטים תפעוליים, היבטי אבטחה. השורה התחתונה: צריך לדעת להעריך איפה זה נותן ערך לביזנס בסוף [33:47].

האזהרה השנייה הייתה החשובה יותר לדבריו, והיא חזרה על מה שנאמר בחלק הראשון של ההרצאה מכיוון אחר — שהעבודה האמיתית היא בהקשר, לא בכלי:

> **— [33:47]** הנושא של איך אנחנו מייצרים alignment, איך אנחנו עושים context management, harness engineering... ככל שתתנו יותר context ותהיו יותר מדויקים, הקודינג אייג'נט שאיתו אתם עובדים יהיה אליין יותר לארגון.

ההסתייגות שנלוותה: "מדויק" נתון לפרשנות. זה צריך להיות מדויק לארגון שלכם, כי כל ארגון שונה ובכל ארגון יש סטאק אחר [33:47].

## 11.2 מהצד של AWS

- **הרעיון חשוב יותר מהמימוש:** גם אם לא Spec-Driven Development ספציפית — הרעיון של לייצר חוזה ולהתמקד ב־outcome לפני שכותבים שורת קוד או פרומפט ראשון, חשוב בעבודה עם כל כלי AI. ב־Kiro זה נתמך נייטיבית [35:19].
- **הכלים זזים מהר, הארגון פחות:** האקוסיסטם משתנה כל שני וחמישי. לא פחות חשוב מהיכולת עצמה הוא לוודא שהארגון יודע להכיל אותה בצורה אחידה — "קירו... scaling with us": אותה קונפיגורציה, IDE, CLI, skills ו־steering עובדים אותו דבר ליחיד, לצוות ולארגון של אלפים [35:19].
- **בדקו את התוצר, לא רק את המהירות:** "הראינו לכם שלקירו יש Powers — תזכרו, גם לכם יש". ההמלצה הייתה לעשות ולידציות, בין אם דרך spec ובין אם דרך steering, כדי לוודא שהמוצר באמת יכול להגיע לפרודקשן [35:19].

> **משאבים שהוזכרו** הדוברים הפנו ל־kiro.dev להורדה ולשימוש חופשי, וציינו שהוכרז חינם לשנה לסטודנטים. בדף הרפרנסים שליקטו הוזכרו Agent Cost Bench, Kiro Analyzer ו־Kiro Workshop, לצד המחקרים שצוטטו בהרצאה. כן הוזכר ערוץ היוטיוב של AWS ישראל שבו מתפרסמים תכנים בעברית, כולל סשנים מהיום הזה [36:51].

# נספח: מילון מונחים

| מונח | מה זה, כפי שהוגדר בהרצאה |
|---|---|
| Spec-Driven Development | מתודולוגיה מובנית ב־Kiro: שאלות הבהרה, ואז requirements.md, design.md ו־tasks.md — לפני שורת הקוד הראשונה. |
| Skills | הנחיות ל־agent שנטענות באופן יזום לפי טאסק או פרסונה. |
| Steering Files | הנחיות שנטענות אוטומטית, פר פרויקט או ברמה רחבה — המקום לכללים הארגוניים: patterns, ספריות, lint rules וקונבנציות. |
| Hooks | אוטומציות שרצות תוך כדי העבודה (עדכון דוקומנטציה, לוקליזציה). ניתן להגדיר אותן בשפה חופשית. |
| Powers | חבילה מוכנה מפרטנר שכוללת התקנת MCP server, קובץ steering והוקים משלימים — הטמעה בקליק אחד. |
| Custom Powers | אותה מעטפת, כשהארגון אורז בה כלי פרטנר מותאם או כלי פנימי משלו, ומפיץ לכל המפתחים. |
| Custom Agents | פרסונות עם system prompt והרשאות משלהן, שניתן לטריגר מתוך pipelines ותהליכים (למשל code reviewer ב־GitHub Action). |
| Kiro Crew | מוצר open source לסשנים ארוכי חיים: ניהול crews של agents, זיכרון ב־knowledge graph, ו־lessons כ־skills ייעודיים. |
| Kiro Web | סנדבוקס מרוחק בענן להרצת משימות agentic ללא אינטראקציה. |
| AWS Transform | השירות שבו בנק הפועלים משתמש כדי לקרוא קוד COBOL מהמיינפריים ולהפיק ממנו מבנה של business functions ודומיינים עסקיים. |
| Credit usage | מודל החיוב של Kiro, להבדיל מ־token usage שנהוג בכלים אחרים. |
| Agent Cost Bench | כלי open source ל־benchmarking עלות בין Kiro לכלים אחרים לפי סוגי טאסקים. |