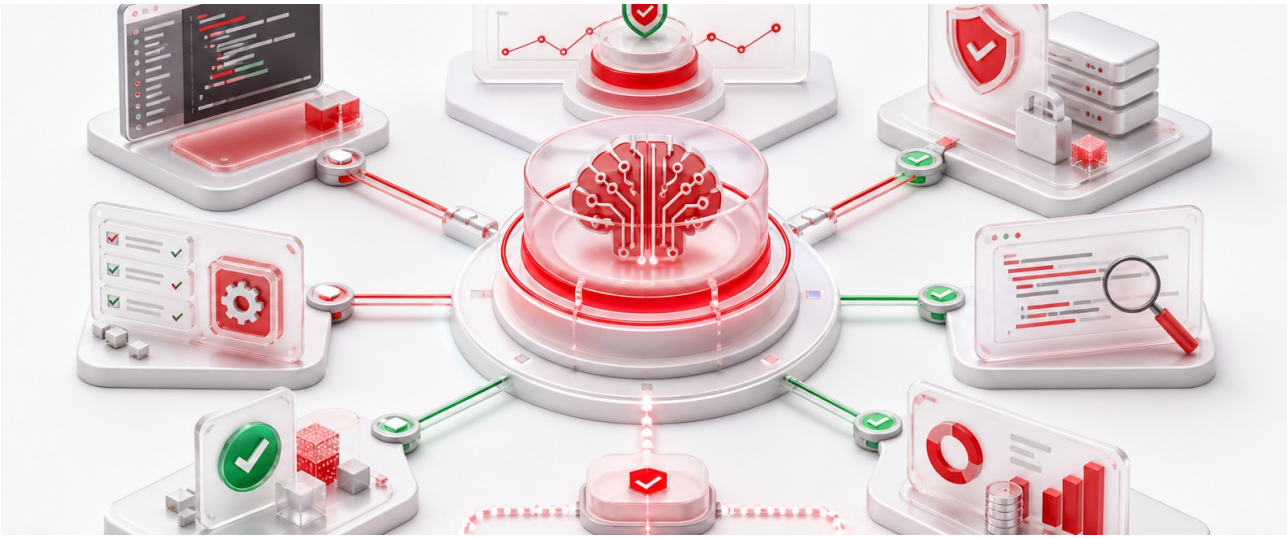
Notilis Tech

IA et codage : ce que les décideurs achètent vraiment quand ils achètent un assistant

Pourquoi le retour sur investissement d'un agent de codage ne se joue pas dans l'IDE, mais dans la chaîne de production logicielle que l'organisation accepte ou refuse de transformer.

2026-05-20 · IA de codage · Delivery · Gouvernance · ROI



Résumé

L’IA de codage ne doit pas être évaluée comme un simple outil de productivité individuelle. Quand une organisation achète un assistant ou un agent de codage, elle introduit un nouvel acteur dans sa chaîne de production logicielle.

Son retour sur investissement dépend surtout de la capacité de l’organisation à spécifier, tester, relire, sécuriser, mesurer et gouverner ce que l’IA produit.

La bonne question pour une direction n’est donc pas : « est-ce que mes développeurs codent plus vite ? ». C’est : « est-ce que le coût complet d’un changement utile, validé, sécurisé et maintenable baisse réellement ? ».

Sommaire

Contexte de lecture

Audience

Cet article s'adresse aux CTO, DSI, RSSI, responsables produit, directions achats et comités de direction qui doivent arbitrer l'adoption de l'IA dans le cycle de développement.

Ce que cet article couvre

Il propose un cadre de décision pour évaluer les coûts complets, les risques, les métriques, les politiques de données et les conditions de passage à l'échelle des assistants et agents de codage.

Ce que cet article ne couvre pas

Il ne compare pas les outils du marché et ne promet pas un ROI automatique ; il traite des décisions de gouvernance à prendre indépendamment du fournisseur retenu.

Introduction

En 1986, Fred Brooks publiait *No Silver Bullet* (lisez le, vraiment, lisez le), un essai dont l'idée centrale était qu'il n'existe pas de technologie, de méthode ou d'outil unique capable de produire à lui seul un gain d'un ordre de grandeur dans la productivité, la fiabilité ou la simplicité du développement logiciel.. Quarante ans plus tard, on a essayé pas mal de choses mais on toujours du mal à le démentir : les méthodes formelles, le génie logiciel et l'UML, le RAD, l'offshoring massif, les frameworks miracles, les méthodes agiles façon culte, le low-code, le no-code. À chaque fois, le même cycle. Promesse révolutionnaire, déploiement euphorique, déception graduelle, intégration dans la boîte à outils. Et à chaque fois, l'industrie a redécouvert que le coût d'un changement logiciel utile ne se réduit pas au coût d'écriture du code.

Mais ça y est, on y est, cette fois c'est la bonne. Cette fois c'est l'IA. Cette fois c'est « vraiment » différent.

Ou pas...

Acheter un assistant de codage n'est pas une décision d'outillage. C'est une décision de transformation du delivery. Cela engage des budgets variables, des politiques de données, des règles de sécurité, des contrats fournisseurs, une gouvernance des accès, des métriques nouvelles, une révision des processus de review et une responsabilité claire sur ce qui peut être généré, par qui, dans quel contexte et avec quel niveau de validation.

Une organisation peut très bien acheter beaucoup d'IA et ne livrer ni plus, ni mieux. Une autre peut en acheter modestement et transformer profondément sa capacité à produire du logiciel utile, testé, sécurisé et maintenable. La différence ne se joue presque jamais dans le modèle. Elle se joue dans le système qui l'entoure.

L'IA de codage est un amplificateur. Elle amplifie les forces d'une organisation structurée. Elle amplifie les faiblesses d'une organisation qui ne l'est pas. Ce n'est pas une nuance. C'est la décision centrale que doit prendre une direction technique avant d'autoriser un déploiement à grande échelle.

Le piège du débat sur la productivité

Le débat public tourne autour d'une question mal posée : est-ce que l'IA rend les développeurs plus rapides ?

Cette question est confortable. Elle donne l'impression qu'il suffit de mesurer un avant, un après, de comparer les durées et de trancher. Un développeur mettait quatre heures, il en met deux, donc l'IA est rentable, emballez c'est pesé.

Sauf que ce raisonnement ne tient pas très longtemps.

D'abord parce qu'il confond le temps d'écriture du code avec le temps de production d'un changement logiciel fiable. Or écrire le code n'est qu'une partie du travail. Il faut comprendre le besoin, cadrer la solution, respecter l'architecture, écrire ou adapter les tests, vérifier les effets de bord, passer la CI, faire relire, corriger, documenter, déployer, observer, maintenir, et parfois supprimer ce qui n'aurait jamais dû être écrit.

Ensuite parce qu'il confond la vitesse individuelle avec la performance collective. Un développeur peut produire plus vite une PR. Mais si cette PR devient plus longue, plus difficile à relire, plus risquée et plus pénible à maintenir, le système global n'a pas forcément gagné quoi que ce soit. Il a juste déplacé le bouchon.

Enfin parce qu'il confond la production de code avec la production de valeur. Plus de code ne veut pas dire plus de produit. Plus de code ne veut pas dire plus de qualité. Plus de code ne veut pas dire plus de sécurité. Plus de code veut juste dire plus de code à comprendre, tester, maintenir, faire évoluer et, parfois, supprimer.

Les études disponibles racontent une histoire beaucoup plus nuancée que les démos LinkedIn.

Une expérience contrôlée sur GitHub Copilot a montré, sur l'implémentation d'un serveur HTTP en JavaScript, un gain de vitesse de 55,8 % pour les développeurs équipés. [1] Le chiffre est réel. Il est aussi extrêmement spécifique : une tâche bornée, un objectif clair, un cadre expérimental. Ce n'est pas une preuve que l'IA améliore mécaniquement le delivery complet d'une équipe produit sur une codebase réelle pendant un trimestre. C'est un peu comme conclure qu'on est un excellent skieur après avoir descendu une piste verte de cinquante mètres : techniquement vrai, opérationnellement discutable.

À l'inverse, une étude METR publiée en 2025, menée sur des développeurs open source expérimentés travaillant sur leurs propres dépôts matures, a mesuré dans ce contexte un ralentissement moyen de 19 % avec l'IA. Plus dérangeant encore, les participants restaient convaincus d'avoir été plus rapides, avant comme après l'expérience. Les mesures disaient l'inverse. [2]

Ce résultat doit être lu avec prudence. L'échantillon est limité. Les projets sont particuliers. Les outils correspondent à un moment donné de l'état de l'art. METR a d'ailleurs publié depuis une mise à jour expérimentale plus nuancée, avec des signaux plus favorables mais difficiles à interpréter proprement. Ce n'est donc pas une preuve que l'IA ralentit nécessairement les développeurs.

Le point qui résiste, lui, est plus simple et plus important : la productivité ressentie n'est pas la productivité réelle. Et un comité de direction qui pilote l'adoption d'un outil sur la base de la productivité ressentie pilote en réalité un bruit psychologique.

Les travaux DORA racontent la même tension au niveau organisationnel. Ils ne disent pas que l'IA serait bonne ou mauvaise par nature. Ils montrent plutôt qu'elle agit comme un multiplicateur : elle peut augmenter l'activité individuelle tout en dégradant le système de delivery si elle grossit les batchs, allonge les reviews ou affaiblit les boucles de feedback. Dans l'une de ses analyses, DORA associe ainsi une hausse de l'adoption de l'IA à une baisse du throughput et de la stabilité du delivery. [3]

Ce n'est pas une condamnation de l'IA. C'est un rappel : un accélérateur mal intégré peut dégrader le système qu'il prétend accélérer.

La bonne question, pour un décideur, n'est donc pas « est-ce que mes équipes codent plus vite ? ». C'est « est-ce que le coût complet d'un changement utile, validé, sécurisé et maintenable est plus bas qu'avant ? ». Cette question-là est beaucoup plus difficile à mesurer. Elle est aussi la seule qui ait une vraie valeur économique.

Ce que l'on achète vraiment

Le coût visible d'un assistant de codage, c'est le siège. C'est aussi, en général, le moins intéressant des coûts.

Quelques dizaines d'euros par développeur par mois sont vite rentabilisés si l'outil supprime ne serait-ce qu'une heure de friction utile. Le débat budgétaire ne se joue pas là. Et honnêtement, n'importe quel commercial peut vous démontrer ce ROI-là sur un coin de table en cinq minutes.

Le débat se joue dans tout ce qui vient autour.

L'intégration dans l'IDE et dans Git. Les accès aux tickets et au repository. Les runners CI dédiés. Les environnements sandboxés. Les logs et l'observabilité des appels modèle. Les politiques de données. Les audits de conformité. La formation des équipes. Le support interne. Les exceptions à gérer. Les revues juridiques des contrats fournisseurs. Les contrôles de dépendances. Les règles sur les secrets. Les conventions de prompts. Les templates de specs. Les outils complémentaires de sécurité.

Bref, tout ce qui n'apparaît pas sur la grille tarifaire du fournisseur, mais qui détermine si l'outil va produire des résultats ou des incidents.

Et puis il y a la nouveauté du moment : le coût variable.

L'époque où un développeur pouvait lancer dix CLI tournant en parallèle sur des taches distinctes est en train de vivre ses derniers jours. Cette époque où nous vivions dans un monde d'opulence et où le token était si abondant que l'on n'arrivait pas à dépasser nos quotas est déjà derrière nous.

Un agent et a fortiori, une équipe orchestrée d'agents, qui utilisent un modèle premium, lisent une codebase, explorent plusieurs fichiers, lancent des tests, corrigent leur patch, recommencent, produisent des PR, ne coûtent pas la même chose qu'une suggestion de ligne. Ce n'est pas le même profil d'usage, ni le même modèle économique. Et ça les fournisseurs IA commencent à sérieusement le sentir passer. La généralisation du codage assisté par des agents IA, sur des runs longs, en multi-agents orchestré, sur plusieurs projets en même temps, impacte sérieusement leur capacité technique et leur porte-monnaie. Leurs infrastructures, n'ont tout simplement pas été conçues pour ça et même eux se rendent compte qu'ils ont sous-estimé l'évolution à laquelle nous avons assisté dans les deux dernières années.

Et ce qui devait arriver, arriva : ils commencent tous à sonner la fin de la récréation.

GitHub a par exemple annoncé une transition de Copilot vers une facturation par crédits IA calculée à partir de la consommation de tokens, incluant les tokens d'entrée, de sortie et de contexte mis en cache selon les modèles utilisés. [7] Ce glissement n'est pas un détail comptable. C'est un changement structurel. Et les autres fournisseurs, OpenAI et Anthropic en tête, s'engagent sur la même voie.

Pour une direction financière, cela signifie une chose très concrète : le coût de l'IA de codage n'est plus une ligne fixe par siège. C'est une ligne hybride composée d'un coût utilisateur, d'un coût d'usage, d'un coût par modèle, d'un coût par contexte et d'un coût par agent. La structure ressemble plus à celle d'un cloud qu'à celle d'une licence logicielle classique. Elle doit être budgétée, suivie et plafonnée comme telle. Mais quoi qu'il en soit elle devient moins lisible et moins prévisible d'un mois à l'autre.

Une organisation qui ne met pas en place un suivi de consommation par équipe ou par projet découvrira la facture trop tard. Et la découverte tardive, dans le contexte agentique, peut piquer. J'ai déjà vu une équipe consommer en une nuit l'équivalent d'une semaine habituelle de tokens. Il suffit d'un agent mal borné, d'une boucle de correction instable ou d'une suite de tests qui échoue en continu pour transformer un usage expérimental en incident budgétaire. Le scénario est banal : un développeur lance un prompt en fin de journée espérant gagner du temps pour le lendemain et rentre chez lui, l'agent commence à tourner en rond, personne ne surveille, et la consommation dérive jusqu'au lendemain matin. Ce n'est pas un risque théorique. C'est un risque opérationnel encore trop rarement abordé dans les démonstrations commerciales.

Les coûts invisibles que le contrôle de gestion ne voit pas

Le coût qu'un comité d'investissement voit le moins est aussi le plus important : celui de la review, de la dette et du rework.

Si l'IA produit plus vite, la review devient mécaniquement plus importante. Et relire du code généré par IA n'est pas toujours plus simple que relire du code écrit par un humain. Parfois c'est même plus difficile.

Un humain laisse des traces de ses intentions. Il connaît le contexte de l'équipe. On peut lui demander pourquoi il a fait tel choix, et il peut expliquer qu'il a repris un pattern existant, qu'il a contourné une

contrainte, qu'il a arbitré entre deux options.

L'IA, elle, produit un résultat plausible. Elle peut fournir une explication après coup, mais cette explication n'est pas nécessairement la cause réelle du code produit. Elle rationalise une série de décisions déjà prises, en tentant de trouver une raison valable à pourquoi elle a fait ce choix. Et parfois elle rationalise trop bien, en inventant une histoire qui a l'air cohérente mais qui n'a rien à voir avec ce qui s'est réellement passé.

Le reviewer doit donc vérifier davantage. Est-ce que le code respecte le besoin. Est-ce qu'il respecte l'architecture. Est-ce qu'il ne duplique pas quelque chose qui existe déjà trois dossiers plus loin. Est-ce que les tests sont utiles ou juste verts. Est-ce qu'une dépendance a été ajoutée sans raison. Est-ce qu'un comportement implicite a changé. Est-ce qu'un risque de sécurité a été introduit. Est-ce que le patch est plus gros que nécessaire.

C'est un vrai travail. Et ce travail n'apparaît dans aucun tableau de bord d'adoption. Et pourtant c'est précisément là que se joue une partie importante du ROI.

Vient ensuite la dette technique.

L'IA rend facile la production d'abstractions inutiles, de wrappers, de helpers, de scripts temporaires, de variantes de fonctions, de tests faibles et de documentation verbeuse. Chacun de ces éléments semble marginal au moment de l'écriture parce qu'ils ont l'air de simplement répondre à un besoin ponctuel. Accumulés sur des mois, ils deviennent une masse de code à maintenir.

GitClear a observé dans ses analyses de dépôts une hausse du code dupliqué, du churn court terme et une baisse relative du refactoring ou du code déplacé. [8] La causalité exacte est discutable, la source est commerciale (et je préfère le dire), mais le signal est cohérent avec ce que beaucoup d'équipes constatent empiriquement. L'IA facilite le copier-coller sophistiqué. Elle ne le crée pas, le copier-coller existait déjà avant, mais elle en abaisse le coût, et c'est ce qui le rend dangereux.

Le dernier coût invisible est le plus subtil. C'est l'érosion de la compréhension.

Si les équipes délèguent trop vite l'écriture sans reconstruire mentalement la solution, elles peuvent perdre progressivement la maîtrise du système. Pas leurs compétences générales car un bon développeur restera un bon développeur. Ils perdent plutôt leur « intimité » avec la codebase. Et cette intimité est précisément ce qui permet de sentir qu'un changement est suspect, qu'une abstraction ne colle pas, qu'un test est faible. C'est une compétence tacite, aussi difficile à mettre dans une métrique que l'instinct de l'inspecteur Colombo, mais essentielle au delivery long terme.

Une organisation qui perd cette compétence tacite le découvre généralement trop tard. Au pire moment. Quand un incident en production demande à quelqu'un de comprendre vite un module que plus personne ne comprend vraiment.

Un décideur sérieux inscrit donc ces coûts dans son business case dès le départ. Pas comme une menace abstraite. Comme une ligne explicite : surcoût de review, dette générée, perte progressive de maîtrise. Une adoption qui ne provisionne pas ces postes ne sait pas ce qu'elle paye.

La sécurité n'est pas un slogan

Un argument récurrent dans les présentations commerciales est que l'IA améliore la sécurité parce qu'elle peut détecter des vulnérabilités.

C'est vrai. C'est aussi incomplet au point d'en devenir trompeur.

Oui, l'IA peut aider à repérer certaines failles. Elle peut expliquer une alerte SAST, proposer un correctif, repérer une injection SQL évidente, signaler une validation d'entrée absente, alerter sur un secret oublié dans un commit, identifier une mauvaise configuration, aider à écrire un test de non-régression après correction.

Mais elle peut aussi introduire des vulnérabilités.

L'étude *Asleep at the Keyboard?* a produit 1 689 programmes sur 89 scénarios de sécurité et trouvé qu'environ 40 % étaient vulnérables. [4] Ce chiffre date, et il faut le replacer dans son contexte. Les modèles ont évolué. Les garde-fous ont évolué. Mais le mécanisme qu'il révèle est plus durable que la mesure : un assistant entraîné sur du code réel a appris aussi des patterns faibles, incomplets, obsolètes ou vulnérables. Et quand on lui demande de produire du code, il ne garantit pas magiquement la sécurité de ce qu'il produit.

J'entends déjà des voix se lever pour me dire que les modèles récents sont nettement meilleurs sur la sécurité que ceux de 2021. C'est vrai. Mais vous, êtes-vous capable de dire, à un instant donné, à quel niveau exact se situe le modèle utilisé par votre équipe sur le scénario précis qu'elle est en train de traiter ? Donc tant qu'on ne sait pas, il faut continuer à valider.

Pour un RSSI, cela signifie que l'IA de codage n'est ni une menace à interdire ni un outil de sécurité à promouvoir. C'est un acteur supplémentaire dans le SDLC, avec ses propres capacités et ses propres risques. La sécurité applicative reste un problème de système. Threat modeling. Secure coding guidelines. SAST. SCA. DAST quand c'est pertinent. Review humaine. Contrôle des dépendances. Règles sur les secrets. Sandboxing des agents. Policy-as-code. Permissions limitées.

Bref, tout ce qu'un RSSI sérieux faisait déjà avant l'IA. Mais qu'il doit désormais étendre à un nouveau type d'acteur et un acteur encore moins prévisible qu'un développeur borné et à qui il est tentant de laisser plus de liberté parce qu'il est « un outil » et pas « une personne ».

Le NIST Secure Software Development Framework reste pertinent précisément parce qu'il rappelle que la sécurité doit être intégrée au cycle de développement, et pas ajoutée comme une prière à la fin du pipeline. [5] OWASP rappelle de son côté, dans son Top 10 dédié aux applications LLM, que les systèmes IA introduisent leurs propres risques : prompt injection, disclosure d'informations sensibles, supply chain LLM, mauvaise gestion des outputs, excessive agency, consommation non bornée et dépendance à des outils ou composants mal contrôlés. [6]

Autrement dit, un déploiement d'IA de codage qui ne s'accompagne pas d'une mise à jour explicite des pratiques de sécurité ne renforce pas la sécurité. Il déplace simplement le périmètre du risque, souvent vers des zones moins observées. Et ce n'est pas le même métier de défendre ce qu'on observe et défendre ce qu'on ignore.

Spécifier mieux, ou ne rien gagner

Beaucoup d'organisations imaginaient que l'IA permettrait de moins spécifier. Décrire grossièrement le besoin, laisser le modèle remplir les trous. Le marketing y a beaucoup contribué, et honnêtement, c'était assez logique de l'espérer.

En pratique, c'est l'inverse.

Un agent avec une mauvaise spec ne produit pas une bonne solution. Il produit plus vite une mauvaise solution. Une équipe qui ne sait pas formuler clairement ce qu'elle veut ne devient pas agile parce qu'elle ajoute une IA. Elle devient simplement capable de matérialiser plus rapidement ses ambiguïtés. Et matérialiser plus rapidement une ambiguïté, ce n'est pas un progrès. C'est juste la mise en lumière d'un autre problème prioritaire à résoudre d'urgence.

L'IA augmente donc la valeur des bonnes specs. Et elle punit les mauvaises.

C'est probablement l'effet organisationnel le plus structurant, et le plus ironique. Beaucoup voyaient l'IA comme une dispense de spécification. C'est exactement l'inverse. Plus on veut déléguer à un agent, plus il faut être précis sur le résultat attendu, les contraintes, les interdits, les conventions, les tests, les limites de scope et les critères d'acceptation.

Le prompt n'a pas remplacé la spec. Il a rendu visible l'absence de spec.

Pour une direction produit, cela veut dire que la qualité du travail de cadrage redevient un goulet d'étranglement. Mais un goulet d'étranglement assumé et profitable. Les organisations qui investissent dans des specs propres, des critères d'acceptation testables et des conventions documentées récolteront les gains. Celles qui se contentent de tickets vagues récolteront du code plausible et inutile. Voire pire, du code plausible et nuisible, ce qui est plus dur à détecter qu'un bug franc.

C'est aussi une bonne nouvelle, au passage. Les bonnes pratiques de cadrage que beaucoup d'équipes connaissent depuis dix ans, sans toujours les appliquer parce que personne n'avait le temps, deviennent enfin rentables à court terme. L'IA paye le travail de spec parce qu'elle l'amplifie.

L'IA n'a pas rendu obsolètes les bons product managers et les bons architectes. Elle les a rendus indispensables.

La grille de risque par cas d'usage

Tous les usages de l'IA dans le code ne portent pas le même risque. Parler de « l'IA dans le développement » comme d'un bloc unique est probablement l'erreur la plus fréquente dans les comités de direction. On en sort avec une politique uniforme qui interdit trop là où ce n'était pas nécessaire, et qui autorise trop là où il aurait fallu encadrer.

Voici une grille opérationnelle, à utiliser comme point de départ et à adapter au contexte.

Cas d'usage	Valeur probable	Risque	Contrôle nécessaire
Explication de code existant	Forte	Faible	Vérification humaine
Documentation technique	Forte	Faible à moyen	Relecture
Boilerplate	Forte	Moyen	Conventions et tests
Génération de tests	Forte	Moyen	Review des assertions
Refactoring local	Moyenne à forte	Moyen	Tests solides
Correction de bug local	Moyenne à forte	Moyen à élevé	Reproduction et test de non-régression
Migration de dépendance	Forte	Élevé	CI, SCA, plan de rollback
Refactoring transverse	Forte	Élevé	Découpage et review architecture
Aide à la review de sécurité	Forte	Élevé	SAST, SCA, DAST et humain
Scripts CI/CD et exploitation	Moyenne à forte	Très élevé	Sandbox et moindre privilège

Cas d'usage	Valeur probable	Risque	Contrôle nécessaire
Code critique	Variable	Très élevé	IA assistant uniquement
Données sensibles	Variable	Très élevé	Politique stricte ou interdiction

La règle de décision tient en une phrase. Plus le changement est local, réversible, testable et peu sensible, plus l'IA peut être utilisée librement. Plus le changement est transverse, irréversible, critique ou sensible, plus l'IA doit être encadrée.

Une politique d'entreprise sérieuse définit donc trois zones. Une zone verte où l'usage est libre dans le cadre des outils approuvés. Une zone orange où l'usage est autorisé sous conditions de review, tests et conventions. Une zone rouge où l'IA est soit interdite, soit limitée à un rôle assistant, sans génération directe acceptée.

Ce n'est pas glamour. Ça ne fera pas une super keynote. Mais c'est probablement ce qui distingue une adoption utile d'un futur postmortem.

Les niveaux de maturité.

On peut représenter l'adoption de l'IA de codage selon plusieurs niveaux. Aucun n'est intrinsèquement supérieur à un autre, et c'est important de le dire parce que la tentation de « monter de niveau » pour le plaisir de monter est réelle dans les organisations.

Le niveau 0, c'est l'absence d'IA structurée. Pas forcément absurde dans des environnements très contraints, mais cela produit souvent du shadow AI : les équipes utilisent quand même des outils, mais sans cadre. Et là, on n'a pas un risque zéro. On a un risque qu'on ne voit pas et dont on ne connait pas l'ampleur.

Le niveau 1, c'est l'assistant individuel non gouverné. Chacun avec son outil, ses prompts, ses habitudes. Les gains locaux peuvent être réels. Les risques sont diffus : fuite de données, incohérence, dépendances douteuses, dette invisible, absence de mesure. C'est l'état dans lequel se trouvent beaucoup d'organisations aujourd'hui, qu'elles le sachent ou non.

Le niveau 2, c'est l'assistant encadré. L'équipe définit des conventions, des règles minimales, des usages autorisés, une review obligatoire, une politique de tests. C'est généralement le meilleur point d'entrée pour une organisation qui démarre sérieusement.

Le niveau 3, c'est l'intégration dans le workflow. IDE, Git, tickets, tests, CI, documentation. L'IA n'est plus à côté du système. Elle passe par le système.

Le niveau 4, c'est l'usage d'agents spécialisés avec permissions limitées. Un agent pour les tests. Un agent pour la documentation. Un agent pour la préparation de migration. Un agent pour le triage de PR. Chacun avec un scope, des permissions et des validations propres.

Le niveau 5, c'est la plateforme agentique gouvernée, observée, auditée et mesurée. Logs, sandbox, policy-as-code, budget, métriques, audit, conformité, intégration complète au delivery. C'est le niveau industriel.

Le niveau 5 n'est pas un objectif universel. Une petite équipe produit peut avoir un excellent ROI au niveau 2 ou 3, à condition de bien faire ce qu'elle fait. Une DSI bancaire, un éditeur logiciel exposé ou une organisation soumise à des contraintes réglementaires fortes devra probablement viser le niveau 4 ou 5 pour éviter que l'adoption ne devienne incontrôlable.

La bonne question, en comité, n'est jamais « à quel niveau sommes-nous ? ». C'est « à quel niveau devrions-nous être, compte tenu de notre risque, de notre volume et de nos moyens ? ». La maturité n'est pas une course. C'est un alignement.

Les politiques fournisseurs et de données

Une décision d'adoption à l'échelle d'une organisation est aussi une décision contractuelle, et cette dimension est souvent sous-traitée trop tard dans le processus.

Les politiques de données varient selon les offres, les plans et les contrats. OpenAI indique par exemple que les données envoyées à l'API ne sont pas utilisées pour entraîner les modèles par défaut, sauf opt-in explicite. Les offres business peuvent avoir des règles différentes des usages grand public, et c'est précisément pour cette raison qu'il faut lire les contrats, pas seulement les pages marketing. [9] GitHub documente de son côté que Copilot construit ses suggestions à partir du code autour du curseur, de certains fichiers ouverts et d'informations de workspace ou de repository selon les cas. [10] Ces détails ne sont pas des détails. Ils définissent ce qui sort de l'organisation, ce qui est conservé et ce qui est utilisé.

Pour une direction juridique et un RSSI, les questions à trancher avant tout déploiement sont précises. Quels fichiers peuvent être envoyés à un fournisseur. Quels dépôts sont exclus. Que fait-on des logs. Quelles options de rétention. Quels fournisseurs sont autorisés. Quelles données sont strictement interdites. Quelles options de zonage géographique sont activées. Quel niveau de chiffrement en transit et au repos. Quelles obligations de notification en cas d'incident.

Ces questions ne sont pas nouvelles. Elles ont déjà été posées pour le cloud public, pour la collaboration en ligne, pour les outils SaaS. Elles se posent à nouveau, dans un contexte où le code source de l'entreprise et parfois les données qu'il manipule transitent par des modèles tiers. Et où des agents peuvent désormais prendre des décisions et lancer des actions opérationnelles à partir de ces données.

L'absence de réponse claire à ces questions n'empêche pas l'usage. Elle l'empêche simplement d'être traçable. Et un usage non traçable, en cas d'incident ou d'audit, c'est généralement le pire des deux mondes.

Les métriques que les comités de direction devraient regarder

Les métriques les plus dangereuses sont les plus faciles à obtenir.

Nombre de prompts. Nombre de lignes générées. Nombre de suggestions acceptées. Nombre de PR ouvertes. Volume de documentation produite. Toutes peuvent monter pendant que la qualité baisse. Elles mesurent l'activité, pas la valeur. Et le problème, c'est qu'elles font de très beaux dashboards. Le commercial les adore. Le comité de direction les adore aussi, parce qu'elles vont dans le bon sens. Personne n'a envie de gâcher ce beau moment partagé en demandant bêtement si le delivery réel s'est amélioré.

Pourtant, c'est là que se joue le sujet.

Les métriques intéressantes sont moins spectaculaires :

- lead time
- cycle time
- change failure rate

- MTTR
- taux de rollback
- taux de défauts post-livraison
- temps de review
- taux de rejet en code review
- volume de code généré, accepté, réécrit puis supprimé
- coût API par ticket
- taux de tests générés modifiés par humain
- complexité cyclomatique
- duplication
- churn
- vulnérabilités introduites par PR
- âge moyen des vulnérabilités ouvertes
- coût de correction post-livraison
- satisfaction développeur
- temps d'onboarding

Même ces métriques doivent être manipulées avec prudence. Une couverture de tests peut monter avec de mauvais tests. Un cycle time peut baisser parce qu'on découpe artificiellement les tickets. Un coût par ticket peut baisser à court terme et exploser à long terme si la maintenance se dégrade. Bref, aucune métrique n'est innocente, et toutes peuvent être détournées par l'optimisation locale.

Au moins, ces métriques regardent le bon objet. Le changement livré et maintenu, pas la génération initiale.

Pour un comité de direction, la métrique centrale tient en une question. Combien coûte un changement utile, validé, sécurisé et maintenable, avant et après l'IA. Toute l'instrumentation devrait servir à approcher cette question, même imparfaitement. Le reste, c'est du tableau de bord décoratif. Et un tableau de bord décoratif coûte en réalité plus cher qu'il ne semble, parce qu'il rassure à tort.

Où le ROI est probablement positif, où il ne l'est pas

Le ROI est probablement positif lorsque les tâches sont répétitives, bien spécifiées, localement testables et peu sensibles. Générer du boilerplate. Écrire une première version de tests. Expliquer du code existant. Produire une documentation technique. Préparer une migration simple et répétitive. Générer des scripts internes non critiques. Aider à l'onboarding. Résumer une PR. Transformer des critères d'acceptation en scénarios de test.

Dans ces cas-là, l'IA retire de la friction sans prendre trop de décisions dangereuses. Et c'est précisément ce qu'on veut.

Le ROI devient incertain lorsque l'IA touche à des fonctionnalités complexes, du legacy peu testé, des migrations larges, du refactoring transverse, de l'analyse d'impact ou de la correction de bugs difficiles. Elle

peut aider. Mais le coût de validation peut manger le gain, et parfois bien au-delà.

Le ROI peut devenir négatif lorsque l'IA est utilisée sans cadre sur du code critique, des données sensibles, des scripts d'exploitation, de la sécurité, des permissions cloud, du CI/CD ou de l'architecture structurante.

Là, le problème n'est pas que l'IA est incapable. Le problème est que le coût d'une erreur est élevé. Et plus le coût d'une erreur est élevé, moins on peut se permettre de confondre assistance et délégation.

Une organisation sérieuse arbitre donc son investissement non en fonction de ce qui est techniquement possible, mais en fonction de ce qui est économiquement et opérationnellement défendable. Ce n'est pas la même chose, et c'est même rarement la même chose.

Décisions concrètes pour un comité de direction

Voici les décisions qu'un comité de direction devrait avoir prises explicitement avant tout déploiement à l'échelle. Pas les mentionner vaguement. Les avoir tranchées, écrites, communiquées et révisées.

Quelles familles d'outils sont autorisées, et lesquelles sont interdites. Une politique implicite est une politique inexistante. Le shadow AI naît du vide laissé par les directions qui n'ont pas voulu choisir.

Quels périmètres sont en zone verte, orange et rouge. Cette grille doit être nommée, écrite, communiquée et révisée tous les six mois. Au minimum.

Quelles politiques de données s'appliquent. Avec un opt-out explicite sur l'entraînement, des règles claires sur les secrets, une exclusion documentée des dépôts sensibles, un zonage géographique défini. Si la direction juridique ne sait pas répondre à ces questions, le déploiement est prématuré.

Quelles permissions sont accordées aux agents. Aucun accès production par défaut. Aucune élévation de privilège sans validation. Un journal d'actions. Une révocation possible. Un agent de codage avec accès libre au réseau et au shell n'est plus un assistant, c'est un acteur opérationnel — et il doit être traité comme tel.

Quel suivi de consommation est mis en place. Au niveau équipe, au niveau projet, avec des plafonds, des alertes et un budget glissant. Le coût variable est désormais une réalité, pas une exception. Et il ne se gouverne pas après coup.

Quelles métriques sont remontées en comité. Pas l'adoption brute. Le coût complet, la qualité, la stabilité du delivery, les défauts post-livraison, la satisfaction des équipes. Bref, des indicateurs qui ne flattent personne mais qui informent vraiment.

Quelles formations sont déployées. Pas une démo de trente minutes le jour du lancement. Une vraie acculturation aux limites, aux risques et aux usages utiles. Avec un volet sécurité explicite, parce que c'est précisément ce qui manque le plus dans les formations actuelles.

Quelle responsabilité existe en cas d'incident. Qui répond si du code généré introduit une vulnérabilité en production. Qui répond si un agent prend une action non autorisée. Cette responsabilité doit être assignée avant l'incident, pas après. Le « avant » évite un débat de cabinet juridique sous pression. Le « après » garantit qu'il aura lieu.

Aucune de ces décisions n'est technique. Toutes sont politiques au sens organisationnel. Et toutes conditionnent le retour sur investissement bien plus que le choix de l'outil lui-même.

Et finalement j'y vais ou pas ?

La réponse tient à trois critères, et l'ordre dans lequel on les regarde compte.

Le premier est le rythme d'évolution, parce qu'il déplace toute la question. « Est-ce trop tôt ? » suppose qu'il existe un bon moment, plus tard, où l'outil serait stabilisé et où l'on saurait enfin ce qu'on adopte. Cette hypothèse n'a aucun fondement. La façon dont on travaille avec l'IA aujourd'hui n'a plus rien à voir avec celle d'il y a trois ans, ni même avec celle d'il y a six mois. Les pratiques qui fonctionnent aujourd'hui seront remplacées par d'autres dans douze mois. Ce n'est pas une phase de transition vers un état stable, c'est l'état stable. L'adaptation permanente est le régime de fonctionnement, pas une période exceptionnelle à traverser. Attendre la fin de la transition, c'est attendre quelque chose qui n'arrivera pas. La question utile n'est donc plus « quand y aller ? », mais « à quel moment commence-t-on à apprendre, et sur quels périmètres ? ».

Le deuxième critère est la position des concurrents. Elle est difficile à juger, parce que l'adoption a longtemps été silencieuse. Pendant deux ans, beaucoup d'éditeurs ont utilisé l'IA discrètement. Je connais personnellement des développeurs qui prenaient soin de ne pas pousser leurs fichiers de spécification dans les dépôts, par crainte que leurs collègues ne les prennent pour de simples opérateurs de prompt. Cette gêne un peu honteuse, n'a plus court. L'adoption est aujourd'hui massive, et il faut partir du principe qu'elle l'est aussi chez vos concurrents. Ce qui reste incertain, en revanche, c'est leur niveau de maturité (usage individuel gouverné, workflow encadré, agents supervisés) et ce niveau-là est beaucoup moins visible que l'adoption brute. La bonne hypothèse est qu'ils ont commencé, mais qu'ils ne sont pas forcément en avance sur les sujets qui comptent. C'est un argument pour démarrer sérieusement, pas pour paniquer.

Le troisième critère est votre culture de la spécification et de la documentation. C'est lui qui détermine ce que l'adoption vous coûtera, pas si elle est possible. Une organisation qui spécifie mal n'est pas empêchée d'adopter ; elle adoptera plus douloureusement, et l'IA agira d'abord comme un révélateur de ses lacunes avant de devenir un levier. En revanche une fois les lacunes révélées il sera obligatoire de les combler. De mon expérience, je retiens surtout que l'adoption pousse spontanément les équipes à structurer ce qu'elles n'avaient pas pris le temps de structurer. Ce n'est pas confortable, mais ce n'est pas non plus un blocage. C'est l'accélération forcée d'un travail de cadrage qu'il aurait fallu faire de toute façon.

Reste la décision elle-même. « Y aller » ne veut pas dire « y aller partout en même temps ». La grille de risque proposée plus haut sert précisément à ça : distinguer les périmètres où l'on peut apprendre vite, sur des changements locaux, réversibles et testables, de ceux où il faut temporiser parce que le coût d'une erreur reste prohibitif. La vraie question n'est donc pas « adopter ou non », c'est « par où commencer ». Et la réponse à cette question-là est rarement « nulle part », parce que la non-décision n'est pas la neutralité. Pendant qu'on hésite, l'usage s'installe quand même, soyez certains que dans vos équipes on utilise l'IA, simplement l'adoption se fait dans votre dos, sans que vous n'en pilotiez ni le périmètre, ni les risques, ni les coûts. Et c'est sûrement un problème que vous aurez à gérer tôt ou tard même si vous repoussez l'adoption de l'IA.

Conclusion

L'erreur la plus fréquente dans les comités de direction est de traiter l'IA de codage comme une commodité de productivité. Un outil de plus dans la trousse. Une ligne d'abonnement dans un budget. Une initiative de transformation dans une présentation.

C'est autre chose.

C'est un nouvel acteur dans la chaîne de production logicielle. Un acteur qui ne porte pas la responsabilité du code. Un acteur qui amplifie les forces et les faiblesses du système qui l'entoure. Un acteur dont le coût n'est plus seulement un siège, mais aussi un usage, un modèle, un contexte, un agent, une intégration, un risque et une exposition.

Acheté sans politique, sans sandboxing, sans révision des pratiques de sécurité, sans révision des critères de review, sans révision des specs et sans révision des métriques, l'outil produit ce qu'il sait produire de mieux. Du code plausible. Plus vite. Pas nécessairement mieux.

Acheté avec discernement, intégré dans un système qui sait spécifier, tester, relire, mesurer et corriger, ce même outil peut transformer durablement le coût de production de logiciel utile.

La position raisonnable, pour une direction, tient en quelques phrases.

L'IA peut produire du code plus vite.

Mais le code n'est pas le produit.

Le produit, c'est du logiciel utile, testé, sécurisé, cohérent, livré et maintenu.

Tant qu'on ne mesure pas cela, on ne mesure pas l'impact de l'IA sur le développement logiciel. On mesure simplement la capacité d'un assistant à remplir plus vite un dépôt Git.

Le jour où l'IA fera vraiment gagner du temps à grande échelle, ce ne sera pas grâce à un meilleur modèle. Ce sera parce que des organisations auront accepté de transformer leur système de production logiciel pour le rendre plus rigoureux qu'avant.

Pas moins rigoureux.

Plus rigoureux.

Et ça, ce n'est pas une décision d'achat. C'est une décision de direction.

Références

1. Sida Peng, Eirini Kalliamvakou, Peter Cihon, Mert Demirer, « The Impact of AI on Developer Productivity: Evidence from GitHub Copilot », Microsoft Research, 2023.
https://www.microsoft.com/en-us/research/publication/the-impact-of-ai-on-developer-productivity-evidence-from-github-copilot/

2. Joel Becker, Nate Rush, Elizabeth Barnes, David Rein, « Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity », METR, 2025.
https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/

3. DORA / Google Cloud, « Impact of Generative AI in Software Development » et « State of AI-assisted Software Development 2025 ».
https://dora.dev/ai/gen-ai-report/

4. Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, « Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions ».
https://arxiv.org/abs/2108.09293

5. NIST, « Secure Software Development Framework, SP 800-218 ».
https://csrc.nist.gov/pubs/sp/800/218/final

6. OWASP, « Top 10 for Large Language Model Applications 2025 ».
https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/

7. GitHub, « GitHub Copilot is moving to usage-based billing », 2026.
https://github.blog/news-insights/company-news/github-copilot-is-moving-to-usage-based-billing/

8. GitClear, « AI Copilot Code Quality: 2025 Data Suggests 4x Growth in Code Clones ».
https://www.gitclear.com/ai_assistant_code_quality_2025_research

9. OpenAI, « Data controls in the OpenAI platform ».
https://developers.openai.com/api/docs/guides/your-data

10. GitHub Docs, « Concepts for providing context to GitHub Copilot ».
https://docs.github.com/en/copilot/concepts/context

11. GitHub, « Does GitHub Copilot improve code quality? Here's what the data says ».
https://github.blog/news-insights/research/does-github-copilot-improve-code-quality-heres-what-the-data-says/

12. JetBrains, « Understanding AI's Impact on Developer Workflows », 2026.
https://blog.jetbrains.com/research/2026/04/ai-impact-developer-workflows/

13. Stack Overflow, « Developer Survey 2025 — AI ».
https://survey.stackoverflow.co/2025/ai