Le code écrit par l'agent : le développeur réagencé

Étude 21 — Réagencer le travail à l'âge des agents

Mathieu Guglielmino

2026-07-13

Le fil

𝓒 Journal des mises à jour

- **2026-07-13** — Création du dossier et du journal (étude 21, pilier Change/Métiers) : thèse « la valeur remonte la chaîne », 5 schémas (grand écart perception/réalité METR, adoption vs confiance, dette GitClear, friction de vérification, pyramide junior FR), ancrage METR/Copilot/GitClear/Stack Overflow/APEC, appel du module 21 baromètre conseil-esn.

La thèse

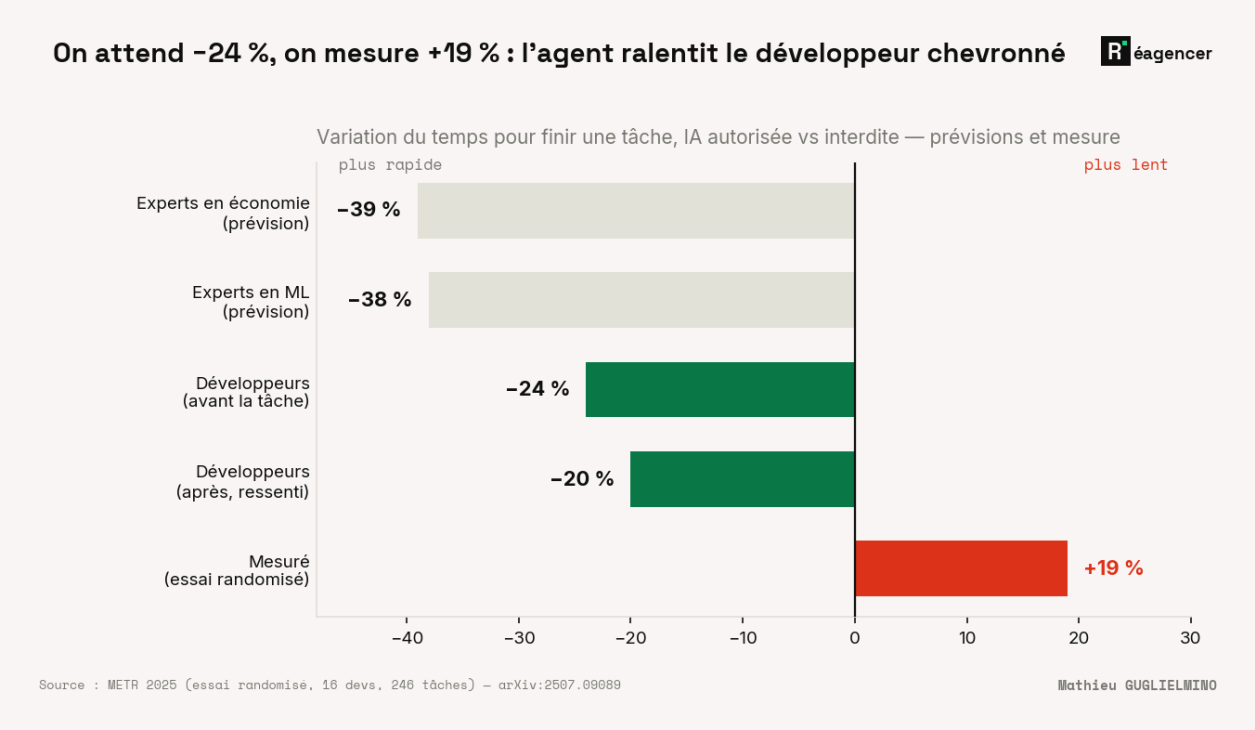
L'agent ne supprime pas le développeur : il **déplace sa valeur de l'écriture du code vers la spécification, la revue et la responsabilité du système**. La promesse « tout le monde codera » se retourne — ce qui devient rare et cher, ce n'est plus taper du code, c'est **savoir quoi demander, vérifier ce qui revient, et porter la responsabilité de ce qui part en production**. Corollaire : l'agent rejoue, en accéléré, **l'effondrement de la pyramide junior** déjà documenté ailleurs (étude 01) — et il le rejoue là où on l'observe le mieux, puisque le développement logiciel est le premier métier qualifié massivement assisté par agents. Le développeur est le **canari** du travail du savoir. Ce dossier n'est pas un réquisitoire anti-IA : il ne dit pas « l'IA ralentit » ; il dit **où va la valeur**, et à quelles conditions.

---

1. On attend −24 %, on mesure +19 % : le grand écart

Commençons par l'essai le plus dérangeant de ce dossier, parce qu'il casse l'intuition la plus partagée sur le sujet — celle qui voudrait qu'un assistant de code accélère mécaniquement le travail.

```
analyse.fig_ecart(d);
```



METR a fait travailler **16 développeurs open-source expérimentés** (environ cinq ans sur leur dépôt) sur **246 tâches réelles**, avec un accès contrôlé à l'IA de frontière début 2025 (Cursor Pro, Claude 3.5/3.7)[1]. Avant de commencer, les développeurs eux-mêmes prévoyaient un gain de temps de **24 %** ; des experts en économie tablaient sur **39 %**, des experts en apprentissage automatique sur **38 %**. Le résultat mesuré, dans un essai contrôlé et randomisé, sur du code de production réel : **+19 %** de temps en plus. L'IA a ralenti ces développeurs chevronnés. Et le plus troublant n'est pas là : **après** avoir fini leurs tâches, les développeurs continuaient de croire qu'ils avaient gagné environ **20 %** de temps. La perception d'un gain survit à sa réfutation par les faits — c'est précisément pour cela qu'il faut mesurer, et non se fier au ressenti collectif, aussi unanime soit-il.

Le contrepoint est tout aussi vrai — et c'est l'écart entre les deux qui est l'histoire. Sur une tâche neuve et cadrée (écrire un serveur HTTP en JavaScript, essai Copilot 2023, n = 95 programmeurs Upwork), le même type d'outil rend **55,8 % plus rapide** (71,17 minutes contre 160,89 minutes)[2]. Les deux résultats ne se contredisent pas : ils mesurent des mondes différents — une tâche jetable et bien définie d'un côté, un dépôt mûr qu'il faut d'abord comprendre avant d'y toucher de l'autre. **On ne les additionne jamais.** Le gain d'écriture sur tâche simple n'est pas l'effet net sur du code de production ; l'écart entre les deux mesure, en creux, tout ce que l'IA ne

[1] METR, « Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity », 2025 — essai contrôlé randomisé, 16 développeurs open-source expérimentés, 246 tâches réelles, IA de frontière (Cursor Pro + Claude 3.5/3.7). Prévision développeurs −24 % ; experts en économie −39 % ; experts en ML −38 % ; auto-estimation a posteriori −20 % ; réel mesuré **+19 %** (ralentissement). https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/ — papier : https://arxiv.org/abs/2507.09089

[2] Peng, Kalliamvakou, Cihon & Demirer, « The Impact of AI on Developer Productivity: Evidence from GitHub Copilot », 2023 — essai contrôlé, 95 programmeurs (Upwork), tâche « serveur HTTP en JavaScript » : groupe assisté 55,8 % plus rapide (71,17 min contre 160,89 min), p = 0,0017, IC95 [21 %, 89 %]. https://arxiv.org/abs/2302.06590

fait pas encore à la place du développeur — comprendre un système existant, ses contraintes implicites, son histoire.

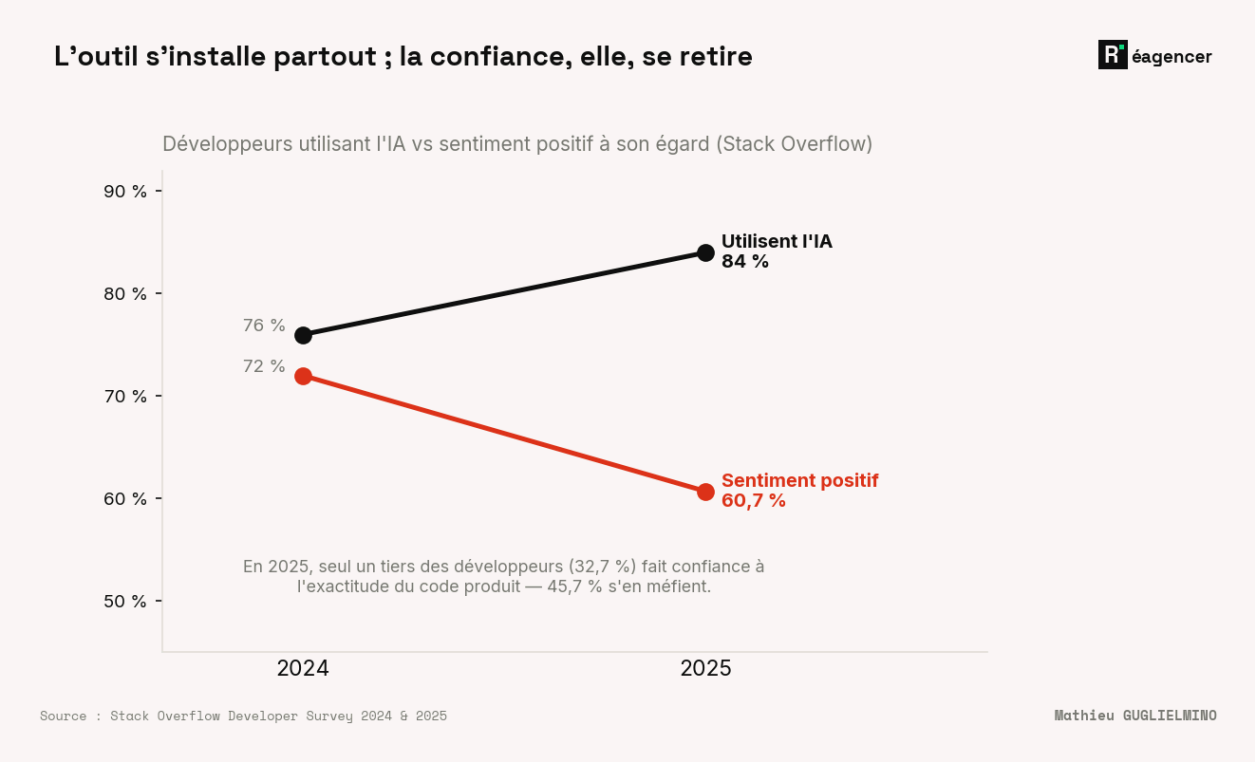
Ce grand écart fixe le cadre du reste du dossier. Si l'agent ralentit le travail sur du code mûr tout en accélérant l'écriture de code neuf, c'est que le geste qui se réagence n'est pas « écrire », c'est « comprendre, cadrer et vérifier dans un système qui existe déjà ». Et c'est précisément le geste que les sections suivantes retrouvent, sous des formes différentes : l'adoption qui grimpe sans emporter la confiance, la dette qui s'accumule faute de consolidation, la frustration qui se concentre sur la vérification, et l'entrée du métier qui se resserre sur les tâches justement absorbées par l'agent.

---

2. L'outil s'installe partout ; la confiance, elle, se retire

Si l'agent ralentit parfois et rassure peu, pourquoi son usage continue-t-il de progresser ? Parce que l'adoption d'un outil et la confiance qu'on lui porte sont deux courbes indépendantes — et elles divergent sous nos yeux.

```
analyse.fig_adoption_confiance(d);
```



Selon le Developer Survey de Stack Overflow, la part de développeurs qui utilisent l'IA ou prévoient de le faire est passée de **76 %** en 2024 à **84 %** en 2025[3]. Dans le même temps, le sentiment positif à l'égard de ces outils a reculé, de l'ordre de **72 %** en 2024 à **60,7 %** en 2025. L'outil devient une infrastructure — quelque chose qu'on utilise par défaut, sans même y penser — sans devenir un objet de confiance. En 2025, seul **un tiers** des développeurs (**32,7 %**) déclare faire confiance à l'exactitude du code produit ; **45,7 %** s'en méfient plus qu'ils ne s'y fient.

[3] Stack Overflow, Developer Survey 2025 — section IA. Usage IA 76 % (2024) → 84 % (2025). Sentiment positif ≈ 72 % (2024) → 60,7 % (2025). Confiance dans l'exactitude 32,7 % vs méfiance 45,7 % (2025). Frustrations : « presque juste, pas tout à fait » 66 % ; débugger le code IA prend plus de temps 45 % ; l'IA gère mal les tâches complexes ≈ 40 %. https://survey.stackoverflow.co/2025/ai/

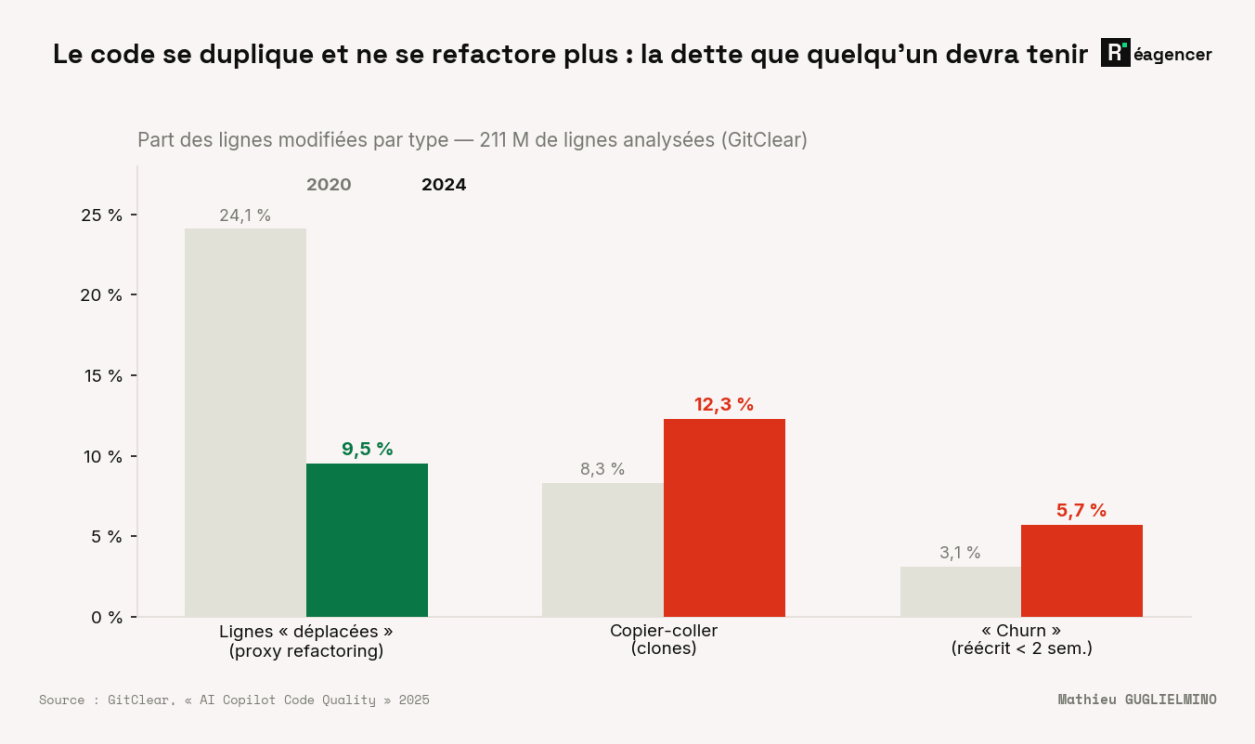
Ce n'est pas une incohérence, c'est une lucidité. On peut très bien utiliser un outil tous les jours et ne pas croire un mot de ce qu'il affirme sans vérification — c'est même, historiquement, l'attitude professionnelle correcte face à un stagiaire brillant mais parfois halluciné. La défiance mesurée ici n'est pas de la technophobie : les deux sections suivantes montrent qu'elle est **fondée sur des faits observables** — une dette de code qui s'accumule (section 3) et une charge de vérification concrète, nommée et chiffrée (section 4). L'adoption dit ce qu'on fait ; la confiance dit ce qu'on en pense après coup. Le développeur d'aujourd'hui a fait la part des deux — et c'est exactement la posture que ce dossier défend : praticien, pas enthousiaste ni procès.

---

3. Le code se duplique et ne se refactore plus

L'adoption et la défiance racontent un ressenti collectif. GitClear apporte une mesure — à l'échelle de l'écosystème, pas d'un laboratoire — sur ce que devient réellement le code.

```
analyse.fig_dette(d);
```



Sur **211 millions de lignes** de code analysées entre 2020 et 2024, la part des lignes « déplacées » — le proxy retenu par GitClear pour désigner le geste de refactoring, c'est-à-dire consolider du code existant plutôt qu'en ajouter — s'effondre de **24,1 %** à **9,5 %**[4]. Dans le même temps, la part de code copié-collé (les clones) grimpe de **8,3 %** à **12,3 %**, et le « churn » — du code réécrit moins de deux semaines après avoir été livré, signe qu'il n'était pas prêt — passe de **3,1 %** à **5,7 %**. Le mouvement d'ensemble est net : on écrit davantage de code, mais on le consolide de moins en moins. Chaque ligne « déplacée » en moins, chaque duplication en plus, c'est une dette technique que **quelqu'un devra tenir** — un développeur, une équipe, un jour, quand il faudra comprendre pourquoi trois versions à peine différentes de la même fonction cohabitent dans la base.

[4] GitClear, « AI Copilot Code Quality », 2025 — 211 millions de lignes de code analysées, 2020-2024. Lignes « déplacées » (proxy de refactoring) 24,1 % → 9,5 % ; copier-coller 8,3 % → 12,3 % ; « churn » (réécrit sous deux semaines) 3,1 % → 5,7 %. https://www.gitclear.com/ai_assistant_code_quality_2025_research

C'est une mesure d'écosystème, pas un essai contrôlé. GitClear observe une corrélation temporelle — la période de la démocratisation des assistants de code coïncide avec ce recul du refactoring et cette hausse du copier-coller — et l'attribue à l'IA. Le rapport ne dispose pas d'un groupe de contrôle qui isolerait cette cause d'autres évolutions du secteur sur la même période. Ce dossier le présente comme un **signal fort**, cohérent avec le reste du tableau (l'adoption qui explose, la défiance qui grandit, la frustration centrée sur la vérification) — pas comme une preuve causale isolée.

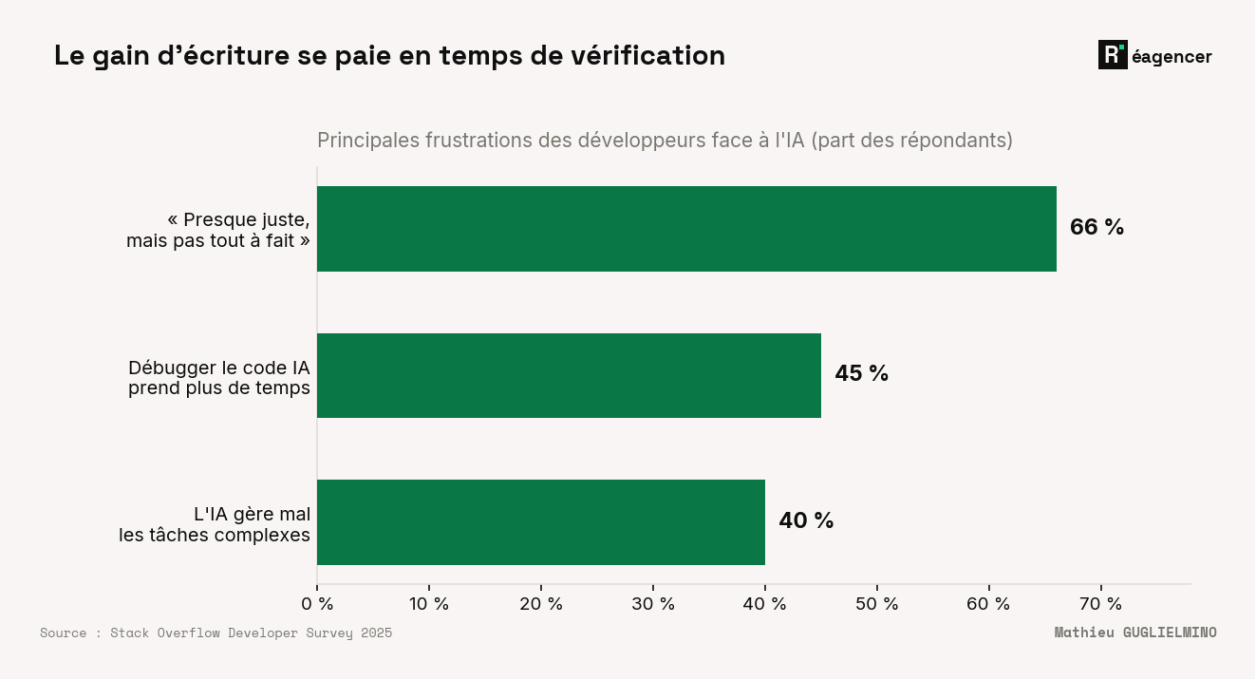
Ce que ce chiffre déplace, très concrètement, c'est la nature du travail restant. Écrire du code neuf devient plus facile et plus rapide ; le maintenir propre, cohérent, non dupliqué — l'activité de fond qui rend un système soutenable dans la durée — recule. Ce n'est pas un hasard si la section suivante montre que la frustration n°1 des développeurs porte exactement sur ce point : la difficulté à faire confiance à ce qui a été produit sans le repasser soi-même à la loupe.

---

4. Le gain d'écriture se paie en temps de vérification

Voici le mécanisme central de la thèse, au plus près du quotidien des développeurs : ce que l'agent fait gagner en écriture, il le reprend ailleurs.

```
analyse.fig_friction(d);
```



La première frustration citée par les développeurs dans le Developer Survey de Stack Overflow 2025, à **66 %**, tient en une phrase : le code produit est « presque juste, mais pas tout à fait »[5]. Vient ensuite un constat plus direct : **45 %** des développeurs disent que **débugger** du code écrit par l'IA leur prend plus de temps que débugger leur propre code. Et environ **40 %** jugent que l'IA gère mal les tâches complexes. Le motif commun de ces trois chiffres, c'est que l'erreur de l'agent n'est presque jamais une erreur grossière et immédiatement visible — c'est une erreur

[5]Stack Overflow, Developer Survey 2025 — section IA. Usage IA 76 % (2024) → 84 % (2025). Sentiment positif ≈ 72 % (2024) → 60,7 % (2025). Confiance dans l'exactitude 32,7 % vs méfiance 45,7 % (2025). Frustrations : « presque juste, pas tout à fait » 66 % ; débugger le code IA prend plus de temps 45 % ; l'IA gère mal les tâches complexes ≈ 40 %. https://survey.stackoverflow.co/2025/ai/

plausible, qui ressemble à du bon code, et qu'il faut donc lire attentivement pour la détecter. Ce type d'erreur coûte plus cher à traquer qu'une erreur franche.

C'est ici que la thèse du dossier devient tangible : le temps ne sort pas du travail, il **change de nature à l'intérieur du travail**. Il sort de l'écriture — plus rapide, plus fluide, déléguée à la machine — et rentre dans la **revue, la vérification, et la responsabilité de ce qui part en production**. C'est exactement le déplacement que documente Mathieu Acher, chercheur Inria au sein de l'équipe DiverSE, à travers le défi LLM4Code : l'agent produit couramment un code qui couvre la majorité d'une tâche, mais c'est le **dernier morceau** — la correction fine, l'intégration propre, la conformité aux contraintes implicites du système — qui résiste, et qui reste à la charge du développeur[6]. Or comprendre ce dernier morceau suppose d'abord de comprendre le morceau produit par l'agent — un code qu'on n'a pas écrit soi-même, ce qui est structurellement plus coûteux à vérifier qu'à écrire. Une enquête de presse professionnelle chiffre le symptôme à l'échelle organisationnelle : les gains de productivité mesurés à l'échelle individuelle s'annulent souvent une fois absorbés par les frictions de coordination et de vérification en équipe[7]. Le gain existe ; il se déplace, et une partie se dissipe en chemin.

---

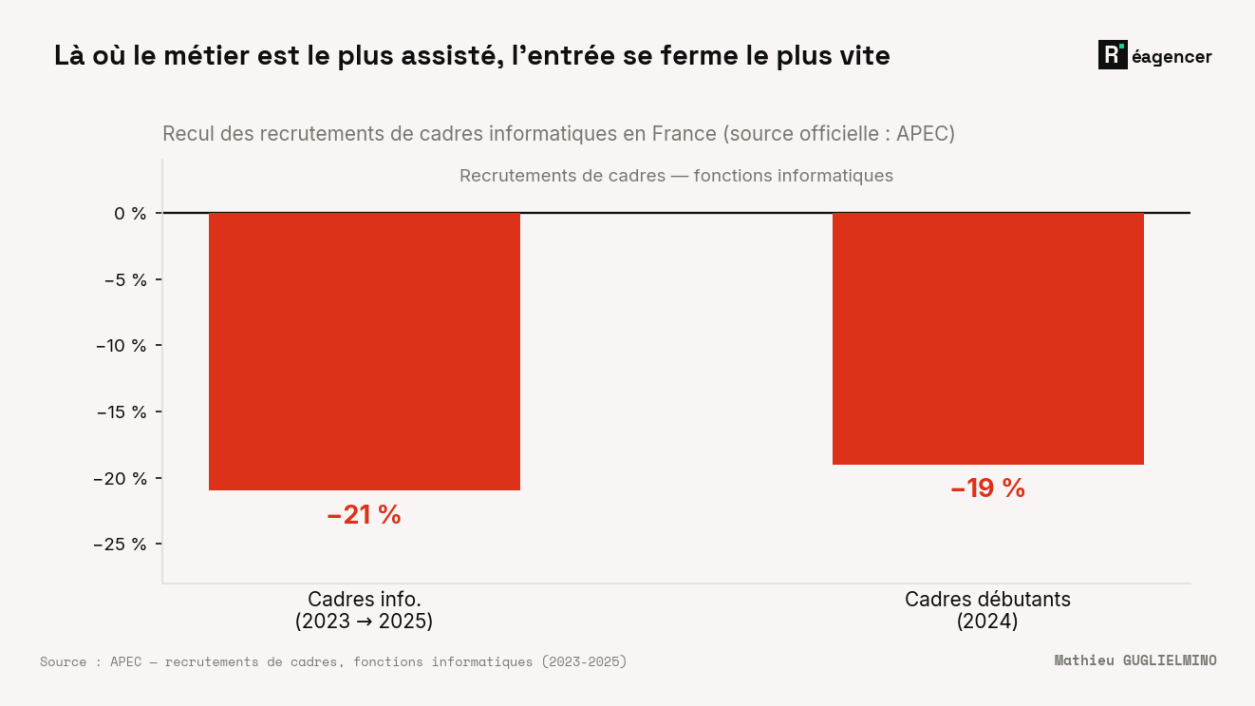
5. Là où le métier est le plus assisté, l'entrée se ferme le plus vite

Si la valeur remonte de l'écriture vers la spécification et la vérification, qui perd le plus à ce déplacement ? Les tâches d'entrée de métier — celles qu'on confie en premier à un junior — sont justement celles que l'agent absorbe le mieux : boilerplate, tests, correction de bugs simples, documentation, refactoring élémentaire. La France en donne une mesure en temps réel.

```
analyse.fig_pyramide_junior(d);
```

[6] Inria, défi LLM4Code (Mathieu Acher, équipe DiverSE) — cadre le « dernier 10 % » que l'agent ne fait pas et la charge de compréhension du code généré qui revient au développeur. https://www.inria.fr/en/best-ia-developer-code-diverse

[7] LeMagIT, « Développeurs et IA : des gains de productivité annulés par les frictions organisationnelles, étude ». https://www.lemagit.fr/actualites/366627990/Developpeurs-et-IA-des-gains-de-productivite-annules-par-les-frictions-organisationnelles-etude-A



Selon l'APEC, les recrutements de cadres des fonctions informatiques ont reculé de **21 %** entre 2023 et 2025 ; les recrutements de cadres **débutants**, spécifiquement, ont reculé de **19 %** sur la seule année 2024. Le développement logiciel est le premier métier qualifié massivement exposé aux agents ; c'est aussi, en France, l'un de ceux où l'entrée se referme le plus vite. C'est le mécanisme documenté ailleurs dans ce programme d'études sur l'effondrement de la pyramide junior — moins de tâches d'apprentissage confiées, moins de postes d'entrée ouverts — observé ici en temps réel, sur le métier qui sert de canari au reste du travail du savoir.

Garde-fou d'honnêteté n° 1 de ce dossier : ce recul est aussi cyclique, probablement pour moitié. Le secteur du numérique français traverse un ralentissement de croissance indépendant de l'IA — Numeum chiffre la croissance sectorielle à **3,5 %**, contre **5,8 %** un an plus tôt. Attribuer la totalité du recul des recrutements débutants à l'agent serait malhonnête ; c'est **probablement les deux** qui se conjuguent, le cycle de financement du secteur et l'absorption des tâches d'entrée par l'IA. Une donnée déclarative éclaire l'intention, sans en être la mesure : une étude Deel/IDC fin 2025 rapporte que **67 %** d'un panel d'entreprises françaises **prévoit** de réduire le recrutement de profils juniors sous trois ans[8]. C'est une intention affichée par des décideurs, pas une mesure de l'emploi réalisé — à ne jamais citer comme telle.

Reste un paradoxe que ce dossier ne referme pas : si l'entrée du métier se ferme, qui apprendra, dans dix ans, à faire ce que les seniors d'aujourd'hui savent faire — cadrer un système, repérer une erreur plausible, porter la responsabilité d'une mise en production ? Un métier qui n'a plus de juniors n'a, mécaniquement, plus de futurs seniors. C'est la question ouverte la plus lourde de ce dossier, et elle dépasse le seul développement logiciel : c'est celle de tout métier du savoir qui laisserait l'agent absorber ses tâches d'apprentissage sans organiser, ailleurs, la formation de ceux qui devront un jour le superviser.

---

[8]Deel / IDC (fin 2025), panel d'entreprises françaises, cité par ChannelNews — 67 % du panel prévoit de réduire le recrutement de juniors sous trois ans (intention déclarée, pas une mesure d'emploi). https://www.channelnews.fr/ladoption-de-lia-freine-le-recrutement-des-profils-junior-dans-lit-155777

6. Ce que la donnée ouverte ne dit pas encore

Angle mort majeur — la France manque de mesure sur son propre terrain. Les données ouvertes mobilisées ici permettent de poser l'écart perception/réalité (METR, échelle internationale), la dégradation mesurée de la qualité du code (GitClear, échelle mondiale), la défiance grandissante des développeurs (Stack Overflow, échelle mondiale) et le recul officiel de l'entrée dans le métier (APEC, France). Elles ne permettent **pas** de dire : la part de code réellement produite avec assistance dans les équipes françaises (éditeurs, ESN, DSI) ; la productivité revendiquée par les dirigeants face à la dette ressentie par leurs équipes techniques — le « on va plus vite mais on répare plus » ; la recomposition réelle **junior → superviseur d'agent** (qui pilote l'agent, qui valide l'architecture, et si la triade se déploie vraiment sur le terrain français) ; ni l'effet sur le recrutement junior tel que vécu par les décideurs, au-delà de l'intention déclarée. C'est exactement ce que le **module 21 du baromètre** `conseil-esn` (dirigeants d'éditeurs et d'ESN, leads/CTO tech) est conçu pour produire, avec une question **miroir** possible côté baromètre `decideurs-ia` (la DSI utilisatrice, en face). Le delta entre la productivité revendiquée et la dette constatée est, très probablement, l'histoire à venir de cette étude.

Réagencer le métier de développeur

Ce dossier ne dit pas que l'agent ralentit le développeur — l'écart entre METR et Copilot montre que c'est selon le terrain. Il dit que la valeur du métier **se déplace** : de l'écriture, de plus en plus déléguée, vers la spécification (savoir quoi demander), la revue (vérifier ce qui revient) et la responsabilité (porter ce qui part en production). Ce déplacement a un coût observable — une dette de code qui s'accumule faute de consolidation, une friction de vérification qui absorbe le gain d'écriture — et un effet social déjà mesurable en France : l'entrée du métier se referme là où l'assistance est la plus installée. La question qui reste ouverte n'est pas technique, elle est organisationnelle : comment former les seniors de demain — ceux qui sauront cadrer et vérifier un système — si les tâches par lesquelles on apprenait ce métier sont précisément celles que l'agent absorbe en premier ?

Sources et références

Calculs et schémas : Réagencer. Journal de recherche en cours — Réagencer, juillet 2026. Données ouvertes au 2026-07-13.