L'Intelligence Artificielle dans le Développement Logiciel **Guide stratégique**

Comment transformer une technologie prometteuse en avantage organisationnel durable

Un guide à l'usage des décideurs

Jean-Claude LAROCHE

MAI 2026

DINUMIA

L'Intelligence Artificielle dans le **Développement Logiciel Guide stratégique**

Jean-Claude LAROCHE

DINUMIA

PRÉFACE

J'ai rencontré Jean-Claude Laroche il y a maintenant plusieurs années, à une époque où les enjeux liés à l'intelligence artificielle suscitaient encore davantage de questionnements que de convictions, mais où les développements d'Oxyl dans ce domaine étaient déjà largement avancés. Je n'avais à l'époque qu'une idée : faire partager les découvertes extraordinaires auxquelles notre exploration concrète de l'IA nous conduisait : avec Jean-Claude, nos premiers échanges ont rapidement révélé chez lui une qualité rare : la capacité à discerner, au-delà de l'effervescence technologique, les transformations profondes qui se préparaient pour les organisations.

Là où beaucoup voyaient avant tout une innovation technique, Jean-Claude a immédiatement perçu une évolution stratégique majeure. Cette intuition s'explique sans doute par son parcours : il n'aborde pas la technologie comme une fin en soi, mais comme un levier de transformation des entreprises, des métiers et des modes de gouvernance.

Depuis lors, il n'a cessé d'alerter, de sensibiliser et d'encourager les directions des systèmes d'information à anticiper les conséquences de cette révolution. À une époque où ces sujets demeuraient encore marginaux dans de nombreux cercles de décision, et où nous avions le sentiment de « prêcher dans le désert », il défendait déjà la nécessité pour les organisations de se préparer à l'émergence de nouvelles formes d'automatisation, d'assistance et d'intelligence collective.

Les évolutions récentes lui donnent largement raison. Les thématiques liées aux systèmes agentiques, à l'automatisation intelligente et à la transformation des modèles opérationnels occupent désormais une place centrale dans les réflexions stratégiques des grandes entreprises et des principales organisations professionnelles du numérique.

L'ambition de ce guide dépasse toutefois la seule question de la productivité ou de la performance opérationnelle. L'intelligence artificielle ne se contente pas d'automatiser des tâches : elle apprend, mémorise et structure progressivement les connaissances, les processus et les décisions de l'entreprise. Cette mémoire numérique devient un actif stratégique de premier ordre.

Face à cette évolution, une question essentielle se pose : celle de la maîtrise de cet actif. Les grandes plateformes technologiques proposent des solutions toujours plus performantes, mais elles créent également de nouvelles dépendances. Jean-Claude défend ici une approche fondée sur la souveraineté, l'autonomie et la capacité des organisations à conserver la maîtrise de leurs données, de leurs savoir-faire et de leurs choix stratégiques, en l'appliquant au secteur du développement logiciel. Non par défiance à l'égard de l'innovation, mais parce qu'il considère que ces enjeux dépassent largement le cadre de l'entreprise et touchent à notre capacité collective à préserver notre indépendance économique et technologique.

Cette conviction, portée avec constance depuis plusieurs années, irrigue l'ensemble de cet ouvrage. Elle témoigne d'une vision qui privilégie l'intérêt général, la pérennité des organisations et la responsabilité des décideurs face aux transformations en cours.

C'est cette vision que je vous invite à découvrir au fil des pages qui suivent.

Fabrice Reby, fondateur d'Oxyl

AVANT-PROPOS

> « La vérité, comme souvent, est plus nuancée — mais aussi plus actionnable. »

Ce guide s'adresse aux DSI qui s'interrogent sur la manière de saisir l'intelligence artificielle pour leur activité de développement logiciel — sans succomber aux promesses exagérées ni passer à côté d'une transformation réelle.

Le marché est saturé de discours contradictoires. D'un côté, des vendeurs qui promettent un facteur dix de productivité, le remplacement des développeurs, une révolution à portée de clic. De l'autre, des directions informatiques qui ont déjà vécu des transformations technologiques annoncées comme décisives et qui se montrent légitimement sceptiques.

La vérité, comme souvent, est plus nuancée — mais aussi plus actionnable. L'intelligence artificielle appliquée au développement logiciel produit des effets réels, mesurables et documentés. Mais ceux-ci ne peuvent se produire qu'à un certain nombre de conditions qui nécessitent de dépasser les idées reçues.

Ce guide s'appuie d'une part sur une expérience concrète que nous avons pu acquérir auprès de la société Oxyl, spécialisée dans l'industrialisation du développement logiciel grâce à l'IA, d'autre part sur les travaux de recherche les plus récents dont les enseignements ont été partagés avec Oxyl, et dont les références sont mentionnées à la fin de ce document : ce guide a pour objectif de montrer ce qui fonctionne, ce qui ne fonctionne pas, pourquoi, et comment le DSI peut orienter sa stratégie en conséquence.

Il s'organise en quatre parties. La première décrit le paradoxe productif auquel font face la quasi-totalité des organisations ayant adopté l'IA sans cadre structuré. La deuxième expose les principes d'une industrialisation réussie. La troisième traite de l'enjeu humain. La quatrième offre un cadre de décision stratégique et un plan d'action. Ces parties sont structurées en chapitres dont certains comportent des recommandations aux DSI mises en exergue dans des encadrés.

Aucune solution ou produit n'est recommandé dans ces pages. Ce guide a pour seul objectif d'armer le décideur des questions à poser, des critères à exiger, et des pièges à éviter.

Table des matières

PARTIE I	**Le paradoxe productif**	06
01	L'accélération qui n'accélère pas l'organisation	06
02	La dette technique invisible : le vrai coût de l'IA non industrialisée	07
PARTIE II	**Les fondements d'une industrialisation réussie**	08
03	De l'assistant individuel à l'usine logicielle	08
04	La gouvernance par l'architecture, pas par la charte	10
05	« Souveraineté » et données : une contrainte d'architecture	10
PARTIE III	**L'enjeu humain**	11
06	Le bus factor, risque opérationnel difficilement pris en compte par les DSI	11
07	Apprendre en faisant : la revue de code comme levier de compétence	12
08	Amplification des équipes, pas remplacement	14
PARTIE IV	**La décision stratégique**	15
09	Faire, faire faire ou combiner : le cadre de décision ROI	15
10	Plan d'action pour le DSI	17
	Conclusion	18
	Références et sources	19

PARTIE I - Le paradoxe productif

En 2024, 75 % des entreprises utilisaient des assistants IA pour le développement logiciel. Et pourtant, les métriques organisationnelles de livraison n'ont pas progressé en conséquence. Pour nombre d'organisations, elles ont même reculé. Comprendre pourquoi est le préalable à toute décision stratégique sensée.

/Chapitre 1

L'accélération qui n'accélère pas l'organisation

Des chiffres qui ne s'additionnent pas

Le tableau est à première vue déconcertant. Vos développeurs codent plus vite qu'il y a deux ans. Les demandes d'intégration des évolutions dans le SI de l'entreprise s'accumulent. Les métriques individuelles progressent, mais les délais de livraison n'ont pas changé, la liste des fonctionnalités à développer reste saturée, et les retards sur les projets stratégiques persistent.

Le rapport DORA 2024 (Google Cloud, Accelerate State of DevOps Report) livre un verdict contre-intuitif : malgré l'adoption massive des assistants IA dans les équipes de développement, le volume de leur production a reculé de 1,5 % et la stabilité des déploiements a baissé de 7,2 % par rapport à 2023. Les développeurs codent plus vite individuellement, mais la mesure de la capacité de livraison de l'organisation montre une régression.

DORA nomme ce phénomène l'« AI Productivity Paradox » : l'IA augmente le volume sans améliorer la capacité collective. Davantage de code, davantage de demandes d'intégration des évolutions dans le SI de l'entreprise, mais une organisation qui ne progresse pas — voire qui régresse sur les indicateurs qui comptent.

> ***Les développeurs produisent davantage individuellement, mais l'équipe dans son ensemble ne gagne ni en résilience, ni en compréhension partagée du système.***
> — DORA State of DevOps Report 2024, Google Cloud

L'étude METR 2025 confirme ce diagnostic sur un panel de développeurs open-source expérimentés : l'assistance IA a ralenti leur productivité de 19 % en conditions réelles, contre une attente de +24 %. Le paradoxe est d'autant plus frappant que les développeurs eux-mêmes croyaient aller plus vite.

L'analogie de l'autoroute

Pour comprendre ce paradoxe, une analogie exposée dans les publications d'Oxyl aide. Imaginez une autoroute. Chaque voiture roule 20 % plus vite qu'il y a un an. Pourtant, le temps de trajet moyen n'a pas changé. Pourquoi ? Parce qu'il y a deux fois plus de voitures sur la route, que les échangeurs n'ont pas été redessinés, et que personne ne coordonne les flux.

L'IA sans cadre industriel, c'est exactement cela : plus de code, plus de demandes d'intégration des évolutions dans le SI de l'entreprise, plus de vélocité apparente — et le même embouteillage au moment de la revue, de l'intégration, de la mise en production. Donner un assistant IA à chaque développeur sans cadre organisationnel, c'est comme distribuer des voitures de course sur des routes départementales sans code de la route. La vitesse individuelle grimpe. La qualité de livraison collective ne suit pas.

Les vrais chiffres à retenir

McKinsey a publié en 2024 son étude la plus complète sur l'impact de l'IA générative en développement logiciel. La conclusion est sobre : les développeurs assistés par IA sont à 25 à 30 % plus susceptibles de réaliser leurs tâches dans le délai imparti.

Pas ×10. Pas ×5. Une probabilité de réalisation améliorée d'un quart. C'est un gain réel, significatif, mesurable. Mais c'est un gain individuel, sur des tâches isolées, dans des conditions contrôlées.

> **POINT CLÉ POUR LE DSI**
>
> Toute promesse supérieure à 30 % de gain de productivité individuelle doit être questionnée. Si un fournisseur annonce un facteur ×10, posez trois questions : sur quel périmètre ? mesuré comment ? pendant combien de temps ? Les benchmarks publics illustrent bien le décalage entre marketing et production. SWE-bench Lite (tests sur corrections de 1 à 5 lignes) affiche 70 % de réussite pour les meilleurs agents. SWE-bench Pro (tâches réalistes multi-fichiers) tombe à 23 %. La différence entre les deux est la différence entre une démo et la production.

/Chapitre 2

La dette technique invisible : le vrai coût de l'IA non industrialisée

Une accumulation silencieuse

Le volume de code est une moitié du problème. L'autre moitié, c'est ce que ce volume contient.

L'étude GitClear 2025 a analysé des millions de lignes de code produites avec et sans assistance IA. Résultat : le code dupliqué a quadruplé entre 2020 et 2024. Le ratio de réusinage de code — la part du travail qui consiste à simplifier et restructurer — a chuté de 24 % à 9,5 %.

En effet, l'IA générative optimise pour la réalisation de la tâche en cours, pas pour la maintenabilité du système. Elle propose le patron de conception, ou « pattern », le plus probable, pas le plus adapté à votre architecture. Les développeurs, soumis à la pression de la vélocité, acceptent la suggestion et passent à la tâche suivante. Le code fonctionne. Il passe les tests. Il sera un cauchemar à maintenir dans six mois.

Cette dette est invisible parce qu'aucun outil d'analyse du code source ne la détecte, aucun test ne la dépiste, aucun tableau de bord de productivité ne la mesure. Elle s'accumule silencieusement jusqu'au jour où un réusinage de code qui aurait pris deux jours en prend vingt.

Les signaux d'alerte à surveiller

Pour le DSI, certains signaux d'alerte sont à surveiller, dès lors qu'ils révèlent un ralentissement progressif de l'organisation : une accumulation de tickets[1] de travail à reprendre, ou une difficulté croissante à faire évoluer les systèmes. Les budgets de maintenance augmentent sans que les équipes comprennent clairement pourquoi.

1. Dans toute la suite de cet ouvrage, nous désignerons par « ticket » un document qui décrit une tâche à faire, une fonctionnalité à ajouter, ou un problème à corriger dans un logiciel.

SIGNAUX D'ALERTE : LES INDICATEURS À SURVEILLER

- Augmentation du nombre de tickets de travail à reprendre (corrections de corrections)
- Allongement des cycles de revue de code
- Réticence croissante de l'équipe à toucher certaines parties du code
- Augmentation du ratio maintenance / nouvelles fonctionnalités
- Absence de réduction de la liste des fonctionnalités à développer malgré une vélocité individuelle en hausse

Si vous observez ces signaux six à douze mois après le déploiement d'assistants IA, c'est le signe que vous avez accéléré la production des individus sans industrialiser le processus.

La dette technique invisible générée par l'IA non encadrée est probablement le coût caché le plus sous-estimé de la transformation numérique actuelle. Elle se manifeste des mois plus tard dans les budgets de maintenance, les délais de livraison et la frustration des équipes.

Le diagnostic

Face à ce constat, la bonne réponse n'est pas de renoncer à l'IA. C'est de changer l'approche. C'est un problème d'ingénierie système : optimiser un composant sans optimiser le système dégrade le système. La solution n'est pas un meilleur outil individuel — c'est un meilleur système.

PARTIE II - Les fondements d'une industrialisation réussie

Si le problème est systémique, la solution est systémique. Cette partie expose les principes qui différencient une approche industrielle d'une simple distribution d'outils individuels. Ces principes constituent la base d'un cahier des charges que le DSI devrait selon nous exiger de toute solution ou démarche.

/Chapitre3

De l'assistant individuel à l'usine logicielle

Les leçons du fordisme appliquées au code

Le principe de l'industrialisation du développement logiciel avec l'IA reprend les enseignements du fordisme : introduire des boucles de contrôle dans le processus de fabrication : décomposer une opération complexe en tâches atomiques, vérifier la qualité à chaque étape, installer des capteurs qui détectent les défauts avant qu'ils ne se propagent.

W.E. Deming l'a formalisé quarante ans plus tard : la qualité ne s'inspecte pas à la fin, elle se construit à chaque étape. C'est exactement ce principe qui manque à l'IA générative quand on la laisse coder sans cadre.

La différence entre un développeur assisté par IA et une organisation industrialisée est la suivante : dans le premier cas, c'est la production de l'individu qui est optimisée. Dans le second c'est celle de l'ensemble du système. L'un produit du code. L'autre produit du code garanti

L'unité atomique : le ticket normé

Une chaîne de montage ne fonctionne que si les pièces qui y entrent sont standardisées. L'équivalent logiciel

est le ticket normé. Un ticket qui répond à des critères stricts :

- **Atomicité :** une seule fonctionnalité, un seul périmètre, aucune ambiguïté sur le résultat attendu.
- **Tests d'acceptation définissables a priori :** avant la première ligne de code, on sait comment vérifier que le travail est fait.
- **Périmètre calibré :** moins d'un jour-homme estimé. Ce qui dépasse est découpé.
- **Absence de dépendance instable :** aucune API tierce imprévisible.

Ce qui signifie que des tickets qui ne répondent pas à ces critères, comme les réusinages de code massifs, les migrations d'architecture, les tickets exploratoires doivent être exclus.

> **QUESTION POUR LE DSI**
>
> Quel pourcentage de votre liste des fonctionnalités à développer est composé de tickets qui respectent ces critères ? Dans la plupart des backlogs audités[2], ce ratio oscille entre 30 et 60 %.
> C'est votre périmètre d'industrialisation potentielle immédiate. La question n'est pas « peut-on tout automatiser ? » — la réponse est non. La question est « qu'est-ce qui peut l'être de manière fiable, pour produire quelle valeur ? »

La chaîne industrielle de développement logiciel[3] : des points de vérification à chaque étape
Une chaîne industrielle de développement logiciel assisté par IA ne doit pas se contenter de générer du code. Elle doit soumettre ce code à des contrôles séquentiels — des points de vérification — qui bloquent la progression si la qualité n'est pas au rendez-vous.

Qualification → Spécification → Génération → Tests automatiques → Revue humaine → Intégration continue → Validation en production.

À chaque étape, une vérification. Le code ne passe à l'étape suivante que si les critères de l'étape précédente sont remplis.

Ce que cette chaîne de production apporte par rapport à l'assistant individuel :

- Les défauts sont détectés au plus tôt, avant qu'ils ne se propagent.
- La gouvernance est structurelle, pas déclarative. Elle ne dépend pas de la vigilance de l'équipe.
- La traçabilité est complète : chaque décision, chaque validation, chaque résultat est auditable.
- Le taux de succès est mesurable sur un périmètre défini, ce n'est pas une estimation.

Le système auto-améliorant : l'avantage cumulatif
Une chaîne de production statique est optimale au moment de sa livraison et dégradée dix-huit mois plus tard — parce que les projets évoluent, les conventions changent, les équipes apprennent. Il est donc légitime, à l'heure de l'IA, d'attendre de la chaîne de production qu'elle s'améliore en permanence.

Le principe est le suivant : chaque ticket traité génère des signaux. Chaque correction apportée lors d'une revue est une donnée. Ces signaux alimentent une boucle de rétroaction qui améliore les paramètres de la chaîne de production — les points de vérification, les conventions, les patterns — et les nouveaux paramètres doivent être testés sur un corpus de tickets historiques avant d'être appliqués.

> Conseil aux DSI : interrogez vos fournisseurs sur ces points précis : Comment la chaîne de production évolue-t-elle ? Quel est le mécanisme de validation des évolutions de cette chaîne ? Qui contrôle les modifications ?

2. En développement logiciel, le terme backlog désigne la liste priorisée des tâches à réaliser.

3. Dans la suite du document, nous utiliserons aussi le terme d'« usine ».

/Chapitre 4

La gouvernance par l'architecture, pas par la charte

Le piège des chartes IA

La plupart des approches de gouvernance de l'IA en entreprise reposent sur des chartes, des bonnes pratiques que l'on espère voir suivies. C'est nécessaire comme fondement éthique. C'est insuffisant comme mécanisme de contrôle.

Une gouvernance efficace de l'IA en développement logiciel doit être inscrite dans l'architecture du système de développement logiciel, pas seulement dans les comportements humains.

Les éléments de gouvernance structurelle comprennent :

- **Les analyseurs statiques :** un code non conforme ne doit pas progresser jusqu'à sa mise en production.
- **La chaîne CI/CD :** si le développement a échoué, le code n'avance pas parce que la chaîne CI/CD le bloque.
- **La revue humaine** doit être obligatoire sur les décisions architecturales et les demandes d'intégration de nouvelles lignes de code dans le SI.
- **La validation post-déploiement :** il doit y avoir une vérification systématique de l'absence de régression quelques jours après la mise en production du code.

Cette gouvernance doit être mécanique, reproductible, auditable.

Un principe d'architecture souhaitable dans les déploiements de l'IA est le tiering des modèles : l'utilisation de différents niveaux de sophistication selon la nature de la tâche :

NIVEAU	DESCRIPTION	COÛT
Niveau 0 Scripts déterministes	Outils d'analyse du code source, analyseurs statiques, valideurs de format. L'IA n'est pas nécessaire à ce niveau, mais des règles précises sont indispensables.	Faible — traite une large part des vérifications courantes
Niveau 1 Modèles d'IA légers hébergés localement	Triage, classification, pré-analyse.	Marginal — les données ne quittent pas le périmètre
Niveau 2 Grands modèles experts	Génération de code complexe, revue approfondie. Mobilisés uniquement quand les niveaux inférieurs ne suffisent pas.	Élevé — usage ciblé

Cette architecture doit pouvoir réduire la facture de 60 à 70 % par rapport à une approche tout-niveau-2, et elle doit embarquer des exigences de « souveraineté » native selon le principe suivant : les données sensibles sont traitées localement sans jamais quitter l'infrastructure.

/Chapitre5

« Souveraineté » et données : une contrainte d'architecture

Le vrai problème de la « souveraineté »

Héberger nos données en France ne sert à rien si notre outil IA envoie notre code source à un serveur extra européen, typiquement américain, à chaque utilisation. C'est pourtant ce que font la plupart des assistants de codage du marché : à chaque suggestion, notre code source — y compris les noms de variables métier, les structures de données internes — transite vers un modèle hébergé chez un hyperscaler américain.

La question n'est donc pas « où sont stockées nos données ? » mais « à quel moment nos données quittent-elles notre périmètre, et pour aller où ? »

L'AI Act et le durcissement du cadre réglementaire

Le cadre réglementaire se précise rapidement. L'AI Act européen (Règlement UE 2024/1689) est entré en vigueur progressivement depuis février 2025. Il est en cours de révision à l'heure où nous écrivons ces lignes, mais ses principes fondateurs ne changeront pas : il introduit des exigences spécifiques pour les systèmes d'IA à haut risque — catégorie qui inclut les systèmes utilisés dans des processus décisionnels critiques.

Une chaîne de production de code qui génère et valide du logiciel déployé en production entre dans ce périmètre pour un nombre croissant d'applications. Gartner anticipe que la conformité IA deviendra une contrainte structurante pour la majorité des applications critiques en entreprise d'ici 2027.

> **RISQUE RÉGLEMENTAIRE À ANTICIPER PAR LE DSI**
>
> Les organisations qui n'auront pas d'architecture « souveraine » à horizon 2027 ne seront pas en avance — elles seront en retard réglementaire. Questions à poser à tout fournisseur :
>
> **1-** Où est traité le code source à chaque étape de la chaîne de production ?
>
> **2-** Quelles données transitent vers des modèles hébergés dans le cloud, et sous quelle forme ?
>
> **3-** Ces traitements sont-ils auditables ? Et si oui comment ?
>
> **4-** Existe-t-il un engagement contractuel chez le fournisseur sur la localisation des traitements sensibles ?

La bonne pratique est une architecture en tiers clairement documentée, avec des engagements contractuels sur la localisation de chaque type de traitement.

PARTIE III - L'enjeu humain

Les deux premières parties se concentrent sur le code et les systèmes. Cette troisième partie traite de ce qui est souvent le facteur déterminant de la réussite ou de l'échec de l'intégration de l'IA dans le développement logiciel : les personnes. L'intérêt de toute DSI n'est pas de considérer les personnes comme des facteurs de résistance à convaincre, mais comme un capital à développer et des risques à gérer.

/Chapitre 6

Le bus factor, risque opérationnel difficilement pris en compte par les DSI

Une métrique, pas une intuition

Si votre meilleur développeur démissionne vendredi, que se passe-t-il lundi ? La réponse, dans la plupart des organisations, est brutale : personne ne sait exactement ce qu'il faisait, comment il le faisait, ni pourquoi il avait fait certains choix d'architecture trois ans plus tôt. Le code est là. La connaissance est partie.

Ce risque porte un nom formalisé par la recherche en génie logiciel : le bus factor. C'est le nombre de personnes dont la perte simultanée mettrait en péril un module ou un projet. Un bus factor de 1 signifie qu'une seule personne détient un savoir critique non documenté.

Avelino et ses co-auteurs ont publié en 2016 une étude analysant 1 434 projets open source GitHub. Résultat : une proportion significative de projets — développés par des communautés internationales parfois nombreuses — repose sur une seule personne critique. C'est pourquoi ce problème constitue un facteur de risque important dans le développement logiciel.

Les organisations peuvent accumuler une dette de connaissance silencieuse qui se révèle au pire moment.

Ce que l'organisation perd avec le départ du développeur critique : les raisons derrière les choix d'architecture, les contournements spécifiques au contexte métier, les dépendances non documentées avec d'autres systèmes, les cas limites que seule l'expérience permet de connaître.

Dans le secteur privé, ce scénario ralentit un produit ou dégrade un service. Dans le secteur public, il menace la continuité de service : un système de gestion des marchés publics dont personne n'ose toucher le moteur de calcul. Un portail citoyen figé parce que le seul développeur qui comprenait l'authentification est parti en retraite.

> **TEST DU VENDREDI SOIR POUR LE DSI ET SES ÉQUIPES**
>
> Inventoriez vos cinq modules les plus critiques. Pour chacun, comptez combien de développeurs peuvent intervenir seuls en production. Si ce chiffre est « 1 » pour au moins un module, vous avez un risque documenté, mesurable et corrigeable. Cet indicateur se mesure en 30 minutes à partir de l'historique des contributions et des revues de code. Il devrait figurer dans votre cartographie des risques opérationnels.

Le paradoxe de l'IA individuelle face au bus factor

L'IA individuelle amplifie les silos au lieu de les briser. Chaque développeur apprend à utiliser l'IA de son côté, développe ses propres habitudes, ses propres prompts. Aucun de ces usages n'est partagé ni visible par les autres membres de l'équipe.

Le résultat est documenté par GitClear 2024 : dans les projets utilisant massivement l'assistance IA individuelle, le code dupliqué a été multiplié par huit et le réusinage de code s'est effondré. Les développeurs copient des patterns générés par l'IA au lieu de les comprendre et les intégrer dans une architecture cohérente.

Gartner le souligne dans son rapport Predicts 2025 : 80 % des effectifs techniques devront être formés aux nouvelles technologies d'ici 2027. Le défi n'est pas la vitesse de production. C'est la distribution de la connaissance avant qu'elle ne disparaisse.

/Chapitre 7

Apprendre en faisant : la revue de code comme levier de compétence

Pourquoi les approches classiques échouent

Face au bus factor et au besoin de montée en compétence, les organisations disposent d'un répertoire classique de méthodes et d'outils :

La formation en salle : la courbe d'oubli est connue de tous. Ce que le développeur apprend en salle, il commence à l'oublier dès la semaine suivante — la formation est déconnectée de son contexte de travail quotidien.

Le pair programming : efficace à deux, difficile à organiser au-delà. Il transfère la connaissance d'une personne à une autre, à raison d'un seul binôme à la fois. Pour réduire le bus factor sur l'ensemble d'un périmètre technique, il faudrait des années !

La documentation : la majorité de la documentation technique devient obsolète en moins de six mois. De plus, elle transmet de l'information, pas le savoir-faire.

L'externalisation : quand le prestataire part la connaissance part avec lui. L'organisation a acheté du temps, pas de la compétence.

Ce que dit la recherche : la revue de code fonctionne à l'échelle

Bacchelli et Bird ont publié en 2013 à l'ICSE une étude qui montre que la revue de code par les pairs est le mécanisme le plus efficace pour le transfert de compétences dans une équipe de développement. Ce résultat a été confirmé depuis par de multiples études empiriques.

La revue de code possède trois propriétés que les autres méthodes n'ont pas simultanément :

- **Contextuelle :** le développeur apprend sur du code réel, dans son projet, avec ses contraintes. Pas sur un exercice générique.
- **Continue :** elle se fait à chaque livraison, et elle est intégrée au flux de travail quotidien.
- **Scalable :** chaque revue implique au minimum deux personnes. Sur un flux régulier, la connaissance se distribue naturellement à l'ensemble de l'équipe.

La limite historique de la revue de code était la suivante : pour être formatrice, elle exigeait un référentiel de qualité — un développeur senior dont le code sert de modèle. Ce rôle revenait à celui-là même sur lequel la concentration de connaissance posait un problème.

Une chaîne industrielle de développement logiciel comme référentiel objectif

L'apport d'une chaîne industrielle de développement logiciel IA est ici décisif. Celle-ci produit du code normé, structuré, testé, conforme aux conventions du projet — constant d'un ticket à l'autre. C'est un référentiel objectif.

Quand un développeur revoit ce code, il ne révise pas le style d'un collègue. Il confronte sa compréhension du système à un standard documenté et reproductible. Il apprend le standard du projet, pas l'opinion d'une personne.

Les quatre types de revues formatrices

Toutes les revues ne forment pas de la même manière. Quatre types de revues répondent à des objectifs pédagogiques distincts :

La revue de conformité (niveau junior) : c'est la vérification que le code respecte les normes du projet — nommage, structure des fichiers, gestion des erreurs. Dix revues de conformité transmettent les conventions mieux qu'une journée de lecture de documentation.

La revue d'architecture (niveau intermédiaire) : il s'agit de l'évaluation des décisions techniques. Cette revue développe une vision systémique que ni la formation en salle ni la documentation ne transmettent.

La revue cross-module (levier anti-bus factor) : un développeur spécialisé revoit un module qui n'est pas le sien. Après six mois de revues cross-module régulières, un développeur qui ne connaissait qu'un seul module en comprend par exemple trois. Le bus factor passe de 1 à 2, puis à 3.

La revue pédagogique (exposition intentionnelle) : revue assignée sur une technologie que le développeur ne maîtrise pas. Le développeur découvre la technologie non pas dans un cours, mais dans le code de son propre système — avec des patterns réutilisables dès le ticket suivant.

CE QUE VOIT LE MANAGER : LA REVUE COMME FORMATION EN SITUATION DE TRAVAIL

En termes RH, la revue croisée est une formation en situation de travail (FEST). Elle ne nécessite ni salle, ni formateur externe, ni interruption de la production. Pour la Fonction Publique : des organismes comme le CNFPT et l'ANFH peuvent soutenir ou financer des démarches de type AFEST/FEST dans le cadre des plans de formation ou de dispositifs d'accompagnement des établissements. Dans ce cadre, les métriques de progression de revue peuvent constituer des indicateurs de résultats attendus. Le temps moyen à prévoir par revue : 30 à 45 minutes. Résultat mesurable attendu est une qualité du code améliorée et la compétence du relecteur augmentée. Coût comparé entre les formations classiques et un dispositif reposant sur des revues : formation externe certifiante 3 000 à 5 000 € / développeur / an, résultats non mesurables à 6 mois. Revue croisée structurée : incluse dans le coût de la chaîne de production, et résultats doivent être mesurables dès le 3e mois.

/Chapitre 8

Amplification des équipes, pas remplacement

Le repositionnement : qui fait quoi

La chaîne industrielle de développement logiciel ne remplace pas les développeurs.

Dans une équipe classique, les développeurs seniors passent une part significative de leur temps sur des tâches à faible valeur ajoutée — correction d'un bug de formatage, ajout d'un champ, mise à jour d'une dépendance. Non par choix, mais par nécessité.

Une chaîne de production prend en charge ce périmètre sur les tickets qualifiés. Les développeurs doivent pouvoir se repositionner sur les tâches que seuls des humains peuvent accomplir : qualification des tickets complexes, architecture et conception, revue croisée formatrice, mentorat.

Le cercle vertueux de l'organisation apprenante

Un phénomène d'amplification de la montée en compétence doit pouvoir s'observer. Plus l'équipe revoit de code produit par la chaîne de production, plus elle comprend le système. Plus elle comprend le système, mieux elle qualifie les tickets. Mieux les tickets sont qualifiés, plus l'usine est efficace. Plus l'usine est efficace, plus elle libère de temps pour les revues formatives.

Peter Senge décrit ce mécanisme dans The Fifth Discipline comme une organisation apprenante : une organisation dont la capacité à apprendre s'améliore avec le temps, parce que les résultats de l'apprentissage alimentent les conditions de l'apprentissage suivant.

Ce cercle doit pouvoir se manifester concrètement de la manière suivante : les premiers mois, les développeurs apprennent les conventions en revoyant le code normé ; les mois suivants, ils commencent à qualifier eux-mêmes les tickets ; à partir du sixième mois, la connaissance est distribuée, le bus factor a baissé, l'équipe fonctionne de manière plus résiliente.

Le champion interne : signal d'appropriation

Chaque équipe qui adopte une chaîne industrielle de développement logiciel voit émerger un profil particulier : le champion interne. Ce n'est pas toujours le plus senior. C'est celui qui comprend le mieux la jonction entre la machine et le métier.

On le reconnaît à ses comportements : il lit les demandes d'intégration des évolutions des autres sans qu'on le lui demande ; il documente spontanément des choix d'architecture ; il teste les limites de l'usine sur des tickets volontairement ambigus pour comprendre son périmètre réel.

CNFPT Centre national de la fonction publique territoriale - ANFH Association Nationale pour la Formation permanente du personnel Hospitalier - AFEST Action de formation en situation de travail - FEST Formation en Situation de Travail

L'émergence d'un tel profil est le signal le plus fiable que la transformation a pris racine. Tant que la montée en compétence dépend d'un accompagnement externe, elle est fragile. Quand elle est portée par un champion interne, elle est durable.

CONSEIL AU DSI : MESURER LA PROGRESSION POUR LA PILOTER

Les métriques utiles pour piloter la montée en compétence :

- Scope de modules : nombre de modules sur lesquels chaque développeur peut intervenir (lecture / correction / évolution)
- Bus factor par module critique : combien de développeurs peuvent intervenir seuls en production
- Qualité des retours de revue : profondeur des commentaires, taux d'acceptation des corrections proposées
- Nombre de revues cross-module par développeur et par mois

Ces métriques constituent un tableau de bord de santé organisationnelle — justifiable auprès des tutelles et des organismes de formation.

PARTIE IV - La décision stratégique

Les parties précédentes ont posé un diagnostic et décrit des principes. Cette nouvelle partie traduit tout cela en décisions concrètes : comment évaluer le retour sur investissement, comment arbitrer entre faire et faire faire, et quelles sont les premières actions à engager.

/Chapitre 9

Faire, faire faire ou combiner : le cadre de décision ROI

Construire en interne : honnêteté sur ce qui est faisable

Toute direction informatique confrontée à cette problématique se pose la question : pourquoi ne pas construire notre propre solution ?

La question est légitime : en effet, vous pouvez construire une usine. Les composants sont disponibles. Les modèles sont accessibles. Une équipe de trois ingénieurs seniors peut construire une usine fonctionnelle en six à douze mois.

Ce qu'elle ne peut pas construire facilement, c'est la boucle d'auto-amélioration — le mécanisme qui permet à l'usine de progresser au fil des tickets traités. Ce n'est pas un problème de compétence. C'est un problème de données et de temps. Construire une usine est faisable. Construire le système qui l'améliore est ce que l'expérience acquise sur des centaines de tickets en production permet seule d'obtenir.

Cette exigence peut donc conduire à décider de s'adjoindre la compétence et les services d'un prestataire spécialisé. L'intérêt du recours à un fournisseur est examiné ci-dessous.

Le coût réel de l'offshore : les six coûts cachés

L'offshore est souvent présenté comme une alternative. Son coût apparent est compétitif — autour de 25 $/h pour les principales destinations. Mais le coût apparent est une fiction comptable. Les six coûts cachés documentés de l'offshore sont les suivants :

- **Décalage horaire :** chaque question prend 24 heures. Trois allers-retours = trois jours ouvrés supplémentaires.
- **Turnover :** 20 à 30 % par an (NASSCOM, Everest Group). Chaque départ emporte du contexte métier. Chaque arrivée coûte 2 à 4 semaines de montée en compétence.
- **Documentation absente :** premier poste sacrifié sous la pression des exigences de livraison.
- **Dette technique accumulée :** le développeur offshore est mesuré sur sa vélocité, pas sur la maintenabilité du code.
- **Surcoût de communication :** les spécifications doivent être trois fois plus détaillées.
- **Management overhead :** 15 à 25 % du coût total du contrat.

Ces six facteurs multiplient le TJM réel de l'offshore par 1,5 à 2,5 selon la complexité du projet. Le 25 $/h affiché devient 40 à 60 $/h en coût complet. L'inflation salariale dans les centres offshore est de 5 à 8 % par an (Everest Group 2024). L'offshore est un coût linéaire qui monte. Une usine IA bien conçue est un investissement dont le rendement s'améliore avec le temps.

Un cadre ROI en cinq variables

Cinq variables suffisent à calculer le seuil de rentabilité d'une industrialisation IA :

- **C1 — Coût actuel par ticket :** TJM × heures moyennes, management et overhead inclus.
- **C2 — Coût industriel par ticket :** coût de production via une usine (médiane observée : ~400 €/ticket sur un périmètre d'un jour-homme).
- **C3 — Coûts cachés évités :** travail à reprendre, dette, coordination. Historique de réouverture × coût moyen de correction.
- **V — Volume mensuel qualifiable :** nombre de tickets par mois éligibles. Se mesure par audit du backlog.
- **I — Investissement initial :** coût de mise en place de l'usine.

$$\text{Seuil de rentabilité} = I \div [(C1 + C3 - C2) \times V]$$

Si ce seuil dépasse douze mois, l'industrialisation n'est probablement pas rentable. Si le volume qualifiable est inférieur à huit tickets par mois, le ROI sera marginal. Ce seuil s'applique notamment aux équipes de cinq développeurs ou moins.

POUR LE DSI : QUESTIONS À POSER À TOUT FOURNISSEUR AVANT DE SIGNER

1- Sur quel périmètre de tickets vos résultats sont-ils mesurés ? Quelle est la définition exacte du succès ?
2- Quels sont vos critères d'inclusion et d'exclusion ? Sont-ils publiés et auditables ?
3- Quel est votre taux d'échec et comment les tickets échoués sont-ils traités et facturés ?
4- Votre usine est-elle statique ou dispose-t-elle d'un mécanisme d'amélioration continu ?
5- Quel est votre modèle économique — régie au temps passé ou forfait au résultat ?
6- Quel SLA proposez-vous sur la souveraineté des données ?
7- Montrez-moi un scénario où votre offre n'est pas rentable pour mon organisation.

/Chapitre 10

Plan d'action pour le DSI

Phase 1 : Diagnostic (4 semaines)

Avant tout engagement, trois diagnostics à conduire en parallèle :

Le diagnostic du backlog : sur un échantillon de 50 à 100 tickets récents, calculer le ratio de tickets qualifiables selon les critères d'atomicité, de testabilité et de périmètre. Ce ratio détermine le volume mensuel potentiel et l'équation ROI.

Le diagnostic du bus factor : pour chaque module critique du SI, calculer combien de développeurs peuvent intervenir seuls en production. Tout module avec un bus factor de 1 est un risque opérationnel à inscrire dans la cartographie des risques.

Le diagnostic de la base de code : identifier les zones bien testées, bien structurées — périmètre naturel d'une usine — et les zones qui nécessitent un travail préalable.

Phase 2 : Expérimentation (mois 2-3)

Ne pas démarrer par un grand projet de transformation. Démarrer par cinq tickets.

Ces tickets doivent être choisis avec soin : visibles par toute l'équipe, périmètre clair, résultat immédiatement observable et vérifiable. La qualité du premier résultat détermine la crédibilité de tout le dispositif.

Inclure dans les premières revues le développeur le plus sceptique de l'équipe. Sa validation a dix fois plus de poids que celle d'un enthousiaste. S'il dit « le code est propre », l'équipe suit.

Critère de passage : au moins 4 tickets sur 5 livrés avec succès (c'est-à-dire avec ce type de critères : « merge request » acceptée, intégration continue au vert, zéro défaut à J+15).

Phase 3 : Montée en charge (mois 4-6)

Augmenter progressivement le volume. Comparer chaque mois les métriques réelles aux projections ROI.

C'est à cette phase que le ROI devient mesurable. C'est aussi à cette phase que la montée en compétence devient visible dans les métriques : étendue des modules par développeur, bus factor des modules critiques.

Partager les métriques avec l'équipe — pas seulement avec la direction. Les métriques comme instrument de contrôle managérial tuent l'adhésion. Comme miroir de la progression collective, elles la renforcent.

Phase 4 : Autonomie (mois 7+)

À partir de ce stade, un champion interne doit pouvoir émerger et prendre le relais. L'équipe qualifie elle-même ses tickets et anime les revues cross-module.

Un bon dispositif rend l'organisation autonome. La valeur d'un accompagnement externe doit augmenter à mesure que la dépendance à cet accompagnement diminue.

CAS OÙ NE PAS DÉPLOYER UNE CHAÎNE INDUSTRIELLE DE DÉVELOPPEMENT LOGICIEL

- La base de code n'a aucun test et aucune documentation (préalable : audit de « reverse engineering »).
- L'équipe comporte moins de 5 développeurs (le volume qualifiable ne justifie pas l'investissement).
- Les tickets ne sont pas décomposables en unités atomiques (le problème est en amont, dans la spécification).
- Le domaine change plus vite que le temps de qualification (prototypage pur, spécifications qui changent chaque semaine).

Conclusion

Ce que l'IA doit vraiment changer pour votre organisation

L'intelligence artificielle bouleverse actuellement le développement logiciel. Mais elle ne le change pas de la manière dont la plupart des discours le présentent.

Elle ne remplace pas les développeurs — elle repositionne leur temps vers des tâches à plus haute valeur ajoutée, à condition que le système soit conçu pour cela.

Elle n'accélère pas automatiquement l'organisation — elle peut la ralentir si elle est déployée sans cadre industriel, comme le montrent les données DORA 2024.

Elle n'améliore pas automatiquement la qualité — elle peut aggraver la dette technique si les points de vérification de la qualité ne sont pas intégrés à chaque étape.

En revanche, correctement industrialisée, elle produit des effets réels et cumulatifs :

- Un taux de réussite élevé et mesurable sur un périmètre défini de tickets qualifiés.
- Une réduction du bus factor par la distribution de la connaissance via la revue croisée.
- Une montée en compétence continue, mesurable, intégrée au flux de travail quotidien.
- Une architecture de souveraineté native, conforme aux exigences réglementaires à venir.
- Un ROI positif dès le troisième ou quatrième mois sur un volume qualifiable suffisant.

La vraie question pour le DSI n'est donc pas « faut-il adopter l'IA ? » — la réponse est oui, mais avec un cadre. Et la question est « quel cadre, et comment le construire ou l'acquérir ? »

Les organisations qui répondent correctement à cette question aujourd'hui bénéficieront d'un avantage organisationnel cumulatif — en matière de compétences, de résilience et de capacité de livraison — qui sera très difficile à combler pour celles qui attendront.

> **Le plus grand risque, en 2026, n'est pas d'adopter l'IA trop tôt. C'est de l'adopter de la mauvaise manière — et d'intensifier le travail des individus tout en dégradant l'efficacité de l'organisation.**

Ce guide vous a donné quelques outils pour éviter ce piège.

Références et sources

Les données et résultats cités dans ce guide s'appuient sur les publications suivantes :

- Google Cloud — Accelerate State of DevOps Report 2024 (DORA). Throughput organisationnel −1,5 %, stabilité des déploiements −7,2 % malgré adoption massive de l'IA. « AI Productivity Paradox ».
- McKinsey Digital — « Unleashing Developer Productivity with Generative AI », 2024. Gains mesurés : 25-30 % de probabilité accrue de réalisation dans le délai.
- McKinsey Global Institute — « The Economic Potential of Generative AI », 2023 (mise à jour 2024).
- GitClear — « AI-Generated Code Quality Report 2025 ». Code dupliqué ×4 entre 2020 et 2024 ; ratio de réusinage de code de 24 % à 9,5 %.
- GitClear — « Coding on Copilot: 2024 Data Suggests Downward Pressure on Code Quality ». Code dupliqué ×8 ; effondrement du réusinage de code.
- METR — « Measuring AI's Impact on Experienced Open-Source Developers », 2025. L'assistance IA a ralenti la productivité de 19 % (vs attente +24 %).
- Avelino, G. et al. — « A Novel Approach for Estimating Truck Factors », ICPC 2016. Analyse du bus factor sur 1 434 projets open source GitHub.
- Bacchelli, A. & Bird, C. — « Expectations, Outcomes, and Challenges of Modern Code Review », ICSE 2013. La revue de code par les pairs est le mécanisme le plus efficace pour le transfert de compétences.
- Spadini, D. et al. — « When Testing Meets Code Review », ICSE 2018. Confirmation empirique du rôle de la review.
- Gartner — « Predicts 2025: AI and the Future of Work ». 80 % des effectifs techniques à former aux nouvelles technologies d'ici 2027.
- Parlement européen — Règlement (UE) 2024/1689 (AI Act). Obligations de transparence depuis février 2025 ; systèmes à haut risque : obligations complètes d'ici 2027.
- Jimenez, C.E. et al. — « SWE-bench », Princeton University, 2023-2024. SWE-bench Lite : >70 % ; SWE-bench Pro : ~23 %.
- NASSCOM / Everest Group — Données sectorielles sur le turnover IT offshore, 2023-2024. Fourchette 20-30 %.
- Everest Group — « Global Delivery Cost Trends and Salary Inflation in IT Services », 2024. Inflation salariale 5-8 %/an dans les centres offshore.
- Deming, W.E. — Out of the Crisis, MIT Press, 1986.
- Senge, P. — The Fifth Discipline: The Art & Practice of the Learning Organization, Doubleday, 1990.