# Le développement logiciel à l'ère de l'IA générative : vers des applications « jetables » ??

AXES Conseil

Pôle Systèmes d'information de Ségic

axes.fr

axesconseil@axes.fr

L'histoire du développement informatique est jalonnée de révolutions techniques dont les finalités sont de produire des logiciels plus rapidement et à moindre coût. Aujourd'hui, une nouvelle rupture est en cours : l'avènement de l'IA générative (IAG), capable de produire du code à partir d'instructions rédigées en langage naturel. Cela transforme profondément le métier du développement logiciel.

Derrière cette accélération spectaculaire dans la production du code, se cache une question essentielle : si le coût de création d'une application devient marginal, ne faut-il pas désormais voir le code comme une production jetable ?

## Forces et faiblesses de l'IAG dans la génération du code

L'IAG excelle dans la production de lignes de code à partir d'une demande exprimée en langage naturel. La rédaction de centaines de lignes de code opérationnel en quelques minutes est aujourd'hui une réalité.

Cependant, si l'IAG a montré de réelles capacités, elle présente également des limitations. Ell ne garantit ni « l'élégance technique » ni la cohérence globale d'un système complexe par la faiblesse de ne pouvoir avoir une vision globale d'un système.

## Le modèle historique : penser et construire pour durer

Depuis plusieurs décennies, les grandes organisations s'appuient sur des méthodes structurées de développement. L'historique cycle en « V » demeure une référence reposant sur un principe simple : définir précisément les besoins avant de développer, puis vérifier méthodiquement que chaque exigence est satisfaite. Les approches dites « AGILE » permettent d'introduire de la souplesse, mais les principes de « spécification

/ développement / test » restent valables. Ces approches, impliquant des temps importants, ont un point commun : le développement s'inscrivait dans la durée. Le code devait pouvoir être maintenu et corrigé.

Dans un contexte où l'IAG peut produire un prototype fonctionnel en quelques heures, certaines organisations pourraient être/sont/seront tentées de remettre en question la pertinence économique d'une maintenance prolongée du code et la nécessité de constamment réduire la dette technique.

## L'émergence possible d'un développement « jetable »

L'idée d'un logiciel jetable peut sembler provocatrice. Pourtant, de nombreux systèmes modernes reposent déjà sur des composants éphémères : conteneurs, fonctions serverless, environnements de test temporaires ou infrastructures reconstruites automatiquement. Dans le monde du Cloud, la reconstruction régulière est souvent préférée à la réparation.

L'idée du code jetable repose sur le principe de redévelopper l'intégralité du code à chaque évolution fonctionnelle ou technique importante. L'application cesse alors d'être un patrimoine à préserver qu'il convient de faire évoluer. Elle devient un produit consommable dont la valeur réside essentiellement dans sa capacité à répondre rapidement à un besoin temporaire ou d'une pérennité indéterminée.

Cette approche renforcerait encore le paradigme que les données sont l'actif principal. Ce constat est déjà partagé, dans le monde de la géomatique, par exemple.

## Des bénéfices potentiels…

Une telle approche présente plusieurs avantages théoriques :

- réduction de la dette technique,
- adaptation plus rapide aux besoins et
- coûts de maintenance réduits.

Il est également observé une possibilité de simplification technologique, car les DSI ne seraient plus contraintes de conserver dans des temps longs des environnements techniques compatibles avec les applications.

## …et des interrogations…

Cette vision soulève toutefois des interrogations :

### *La perte de connaissance métier ?*

Un logiciel mature contient souvent des milliers de règles implicites accumulées au fil des années. Beaucoup ne figurent dans aucune documentation.

Reconstruire une application suppose d'être capable d'expliciter ces connaissances. Or, cette tâche demeure difficile, même avec l'aide de l'IA.

### *La qualité du code généré ?*

Les développeurs constatent déjà plusieurs limites récurrentes :

- Une tendance à produire du code redondant ou peu optimisé
- Des difficultés à maintenir une cohérence sur des projets de grande taille
- Une compréhension imparfaite des règles métier spécifiques
- Des régressions fonctionnelles lors des évolutions successives
- L'introduction de vulnérabilités de sécurité difficiles à détecter

Produire du code n'est pas produire un système robuste. Les performances, la sécurité, la résilience ou la maintenabilité restent des sujets exigeant une expertise humaine.

### *Traitement des systèmes critiques ?*

Certaines applications critiques (eg. ERP, applications publiques nationales, logiciels pilotant des infrastructures énergétiques, des dispositifs médicaux et/ou administratifs, etc.) exigent une stabilité et pérennité incompatibles avec une reconstruction fréquente.

### *Devenir du métier de développeur ?*

L'essor de l'IA ne signifie probablement pas la disparition des développeurs. Leur rôle évoluera plutôt vers de nouvelles responsabilités : formalisation des besoins métier, validation des solutions générées, supervision de la qualité et de la sécurité, etc.

Autrement dit, la valeur se déplacera progressivement de l'écriture manuelle du code vers l'ingénierie des systèmes et la maîtrise des connaissances métier. Le développeur de demain sera moins un artisan du code qu'un architecte capable de dialoguer avec des intelligences artificielles spécialisées. Un peu comme le tourneur fraiseur qui est devenu opérateur de machine-outil programmable dans les années 1990.

## … et une confirmation : l'importance des données

L'accélération des phases de développement et l'approche « jetable » du code confirme que les données sont la richesse du système d'information. Les gains gagnés sur le développement auraient vocation à être alloués sur les données (structuration, qualité, documentation) pour augmenter leur valeur.

## Conclusion : du logiciel durable au logiciel régénérable

L'émergence de l'intelligence artificielle générative remet en question un principe fondamental du génie logiciel : l'idée qu'un programme doit être maintenu et enrichi dans le temps.

Dans certains contextes, notamment pour les applications métier peu critiques, il est envisageable que la reconstruction périodique devienne plus économique que la maintenance continue. Le concept d'application « jetable » pourrait alors s'imposer comme une nouvelle stratégie industrielle.

Toutefois, cette approche ne saurait devenir universelle. Les systèmes complexes, critiques ou fortement réglementés continueront d'exiger des méthodes rigoureuses inspirées du cycle en V ou de ses héritiers modernes (développement agile). De plus, la qualité des données, la maîtrise des règles métier et la gouvernance des architectures demeureront des enjeux que l'IA ne résoudra pas seule.

L'avenir ne semble donc pas opposer développement classique et développement jetable. Il pourrait plutôt voir émerger un modèle « hybride » : des applications rapidement générées et régulièrement renouvelées autour d'un patrimoine beaucoup plus stable et précieux, constitué **des données,** des **connaissances métier** et des **règles de gouvernance.**