LA RÉVOLUTION DES PRATIQUES AGILES À L'ÈRE DE L'IA

Ce que le SDLC agentique change vraiment au Manifeste, aux méthodes et aux rôles agiles



Luc Legardeur
CEO WeScale - Co fondateur, Solario

Sommaire

Avant-propos

Chapitre 1 — Le Manifeste Agile est-il mort ?

Chapitre 2 — Retour du cycle en V ?

Chapitre 3 — Scrum, Kanban, XP, SAFe : qui survit ?

Chapitre 4 — Que deviennent les rôles ?

Chapitre 5 — Les artefacts : ce qui meurt, mute, survit

Chapitre 6 — La nouvelle économie des rôles

Chapitre 7 — SWOT : l'agilité face au SDLC agentique

Conclusion

Avant-propos

Ce livret part d'une provocation volontaire. Ce n'est pas la question « l'agilité va-t-elle survivre ? » qui compte, mais une autre, plus inconfortable : **qui, dans l'agilité, a construit sa valeur sur autre chose qu'un rituel de coordination entre humains**.



Je tente de mettre en exergue un même principe : ce qui, dans l'agilité, servait à combler un déficit de visibilité entre humains disparaît avec la traçabilité continue qu'exige l'agentique. Ce qui portait une décision, une intention ou une responsabilité, en revanche, survit — et se durcit.

> *L'agilité ne meurt pas. Elle se désincarne.*

CHAPITRE 01

Le Manifeste Agile est-il mort ?



Snowbird, 2001

Tout commence en février 2001, à Snowbird, dans l'Utah. Le décor : des montagnes, de la neige, du soleil, des pistes de ski et dix-sept informaticiens. Dix-sept spécialistes du développement logiciel, adeptes de méthodes dites « légères », qui se retrouvent pour discuter d'un problème très concret. Parmi eux, Kent Beck, créateur de l'eXtreme Programming et du TDD, ainsi que Jeff Sutherland et Ken Schwaber, co-créateurs de

Scrum. Ils se posent une question : pourquoi est-ce si compliqué de fabriquer des logiciels ?

Le développement logiciel, à l'époque, fonctionne ainsi : on rédige des documents. Puis on rédige des documents qui décrivent les documents. Puis on organise des réunions pour discuter des documents. Puis on découvre que le client voulait autre chose. Alors on rédige un nouveau document. Et quelque part, au milieu de tout cela, un développeur finit par se demander : « Mais... on ne pourrait pas simplement lui montrer le logiciel ? » Le problème n'est peut-être pas qu'on ne travaille pas assez ; c'est qu'on travaille, parfois, de la mauvaise manière.

De ce constat naît une phrase fondatrice : « Nous découvrons de meilleures façons de développer des logiciels en le faisant et en aidant les autres à le faire. » Puis les quatre valeurs, que l'on connaît aujourd'hui presque par cœur sont gravées dans le marbre :

— Les individus et leurs interactions plutôt que les processus et les outils.

— Un logiciel qui fonctionne plutôt qu'une documentation exhaustive.

— La collaboration avec les clients plutôt que la négociation contractuelle.

— L'adaptation au changement plutôt que le suivi d'un plan.

Simple, évident, pour nous, maintenant.

Quatre valeurs, quatre bascules

Vingt-cinq ans plus tard, le SDLC agentique rebat ces quatre valeurs mais pas de la même façon pour chacune.

« Un logiciel qui fonctionne plutôt qu'une documentation exhaustive » est la ligne qui bascule le plus nettement. En 2001, la documentation était vue comme un fardeau bureaucratique qui ralentissait la livraison: un mal qu'on tolérait mais qu'on minimisait. Dans le SDLC agentique, la documentation change de nature : elle n'accompagne plus le logiciel, elle le produit directement. Un PRD ou un ADR n'est plus un artefact de conformité rédigé après coup pour l'audit, c'est littéralement l'input que l'agent exécute. Poussée jusqu'au bout, cette logique donne une formule : **la spec devient l'artefact source.** Le Manifeste avait raison sur le fond — le logiciel qui marche reste ce qui compte — mais le moyen d'y parvenir s'est inversé : **c'est par une documentation rigoureuse, pas malgré elle, qu'on obtient du logiciel qui marche vite et bien**. Le risque à surveiller : si cette documentation redevient statique plutôt que vivante, on retombe exactement dans le travers que le Manifeste dénonçait de la paperasse qui ment sur l'état réel du système.

« Les individus et leurs interactions plutôt que les processus et les outils » est la valeur la plus mise sous tension existentielle. L'interaction primordiale du développement n'est plus humain-humain mais humain-agent, et cette interaction ne peut fonctionner sans processus explicites — Constitution, patterns, niveaux d'autonomie qui sont précisément les « processus et outils » que le Manifeste plaçait en second rang. **Ce n'est pas que les individus comptent moins ; c'est que sans gouvernance formelle, l'agent n'a rien à quoi se raccrocher**.

La lecture la plus honnête : cette valeur ne s'inverse pas tant qu'elle se déplace **l'interaction reste centrale, mais elle se joue désormais autant dans la qualité du dialogue humain-agent (le prompt, le contexte, le CLAUDE.md) que dans la réunion d'équipe classique.**

« La collaboration avec les clients plutôt que la négociation contractuelle » tient bon, et se trouve même institutionnalisée plus formellement qu'avant : le Sign-Off à six dimensions du PRD (Valeur, Design, Tech, Légal, QA, Ops) est une manière de scripter la collaboration client plutôt que de la laisser informelle. On perd un peu de la spontanéité « café avec le client » que prônait le Manifeste ; on gagne en traçabilité de ce qui a été réellement négocié et validé.

« L'adaptation au changement plutôt que le suivi d'un plan » est la valeur la plus intéressante, parce qu'elle se retrouve défendue par les deux camps du débat sur le retour du cycle en V, pour des raisons opposées — un débat que le chapitre suivant développe. Techniquement, elle est renforcée : les specs sont « pulled up », pas « pulled up-front », et régénérer un système à partir d'une spec révisée coûte une fraction de ce qu'il coûtait à une équipe humaine. Mais organisationnellement, elle est menacée par un mécanisme nouveau que le Manifeste n'avait pas anticipé : le drift problem. Une fois qu'on a investi lourdement dans une spec précise, la tentation est grande de patcher le code en douce plutôt que de rouvrir le document source — et le lien entre plan et réalité se délite silencieusement.

Douze principes, trois inversions

Sur les douze principes qui accompagnent les quatre valeurs, trois méritent d'être signalés parce qu'ils s'inversent presque littéralement.

« La simplicité — l'art de maximiser le travail non fait » devient contre-intuitive : le SDLC agentique parie sur plus de travail de spécification en amont, pas moins, en misant sur le fait que cet investissement évite un travail de correction en aval bien plus coûteux.

« Les meilleures architectures émergent d'équipes auto-organisées » tient toujours, mais l'équipe auto-organisée inclut désormais l'agent comme partie prenante du système sociotechnique, sous des garde-fous de gouvernance : Constitution, Patterns que le principe original ne prévoyait pas.

« Livrer du logiciel qui fonctionne fréquemment » est massivement accéléré : les cycles se compriment de la semaine à potentiellement l'heure, au point que ce principe, autrefois un objectif ambitieux, devient presque un minimum trivial — déplaçant la vraie difficulté vers la question de savoir si ce qui est livré si vite est bien ce qu'il fallait livrer.

Le fil qui relie tout cela : le Manifeste opposait des paires de valeurs en présupposant un monde où chaque unité de rigueur: documentation, process, plan se payait en vitesse perdue. Le SDLC agentique casse ce compromis : la rigueur documentaire ne coûte quasiment plus de vitesse, parce que c'est justement elle que l'agent exécute. Le Manifeste n'a donc pas été réfuté ; son arbitrage de fond a été rendu partiellement obsolète par un changement de coûts qu'il ne pouvait pas anticiper en 2001.

> *Le Manifeste n'a pas été réfuté ; son arbitrage de fond a été rendu partiellement obsolète par un changement de coûts qu'il ne pouvait pas anticiper en 2001.*

CHAPITRE 02

Retour du cycle en V ?



Est-ce le retour du cycle en V ? Non, pas au sens strict — mais la nuance est plus intéressante que le simple oui ou non, et les sources se contredisent ouvertement là-dessus, ce qui vaut la peine d'être exposé plutôt que tranché artificiellement.

Il faut dire ce qui se joue sur le terrain : les directeurs des systèmes d'information ont, ces derniers mois, le sourire aux lèvres. Une forme de revanche, en quelque sorte. Il redevient possible de s'engager sur le triptyque coût, fonctionnalités, délais et sans rechigner. C'est précisément ce regain de confiance qui ravive le débat sur le cycle en V.

Non — Marc Brooker (AWS)

Le cycle en V suppose que les specs soient pulled up-front : complètes, figées, validées avant que la construction ne commence, avec un flux d'information à sens unique, de la spec vers le code. Ce que produit le SDLC agentique est différent : des specs pulled up, c'est-à-dire rendues explicites et versionnées, mais pas achevées avant coup. Elles restent des artefacts vivants qu'on retouche à mesure que l'exécution révèle des arbitrages non anticipés.

La différence n'est pas cosmétique. Dans un cycle en V, réviser la spec en phase de build est un échec de process ; dans le modèle agentique, c'est le mode de fonctionnement normal. Brooker insiste aussi sur le fait que ce sont des humains qui continuent à trancher les compromis, de façon réactive — l'agent ne fait qu'accélérer l'exécution, pas la nature du pilotage. Et contrairement à une équipe humaine sous cycle en V, l'agent n'a « pas d'ego » et ne résiste pas psychologiquement à voir sa spec révisée dix fois.

Oui — Uberto Barbini (expert DDD)

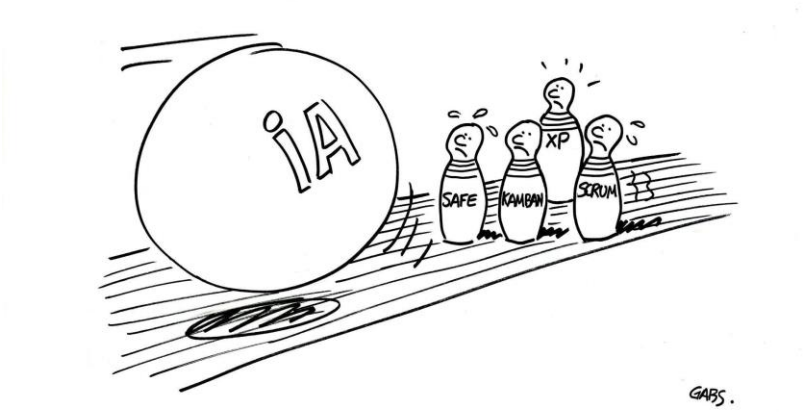
Son argument central : une spécification suffisamment précise pour générer de façon fiable le système voulu est, par construction, aussi complexe que le système lui-même — on n'a fait que déplacer la complexité du code vers le texte, pas la supprimer. Il pointe aussi le « drift problem » : les correctifs appliqués en production ne remontent jamais mécaniquement dans la spec, qui se met donc à mentir sur l'état réel du système au fil du temps. Et la non-déterminisme de la régénération aggrave le problème plutôt que le neutraliser.

Une position intermédiaire — databoozt

Cette troisième voix défend l'idée d'une « synergie naturelle » entre IA agentique et cycle en V, tout en reconnaissant franchement que le problème de fond n'est pas résolu : des spécifications complètes en amont restent « largement inatteignables pour des systèmes complexes ». Le vrai terrain d'accord entre les trois lectures n'est donc pas la forme, cycle en V ou non mais l'exigence qu'elles pointent toutes : celle d'expliciter l'intention avant l'exécution, sans pour autant réimporter la rigidité d'un plan figé une fois pour toutes.

CHAPITRE 03

Scrum, Kanban, XP, SAFe : qui survit ?



Un choc frontal

Quels sont les impacts sur les méthodes agiles elles-mêmes ? Les quatre grandes méthodes ne sont pas menacées de la même façon, parce qu'elles ne reposent pas sur le même mécanisme central. Celles qui synchronisent des humains par des réunions périodiques s'effondrent ; celles qui organisent un flux ou une discipline d'ingénierie s'en sortent nettement mieux certaines en sortent même renforcées.

Scrum — le plus disloqué

Le sprint comme unité de temps fixe n'a plus de sens quand le grain de travail devient la journée-agent. Le daily standup perd sa fonction de synchronisation dès que l'état du système est tracé en continu. La vélocité en points s'efface devant des métriques de flux, de temps de revue de PR, de taux de succès CI au premier passage. Ce qui survit c'est la user story, la Definition of Ready. La rétrospective le fait sous une forme durcie et formalisée, pas sous sa forme rituelle d'origine.

Kanban — le mieux préparé

À l'inverse, Kanban est la méthode la mieux préparée à cette transition, parce qu'elle n'a jamais reposé sur un timebox à casser. Sa mécanique fondatrice: visualiser le flux, limiter le travail en cours, tirer plutôt que pousser, mesurer lead time et cycle time se superpose presque exactement à un pipeline CI/CD : les colonnes du board deviennent les états du graphe de spec, et les cartes deviennent des PR ou des runs d'agent plutôt que des tickets humains.

Le déplacement le plus important porte sur ce que limite le WIP : ce n'est plus la capacité de codage de l'équipe qui est la ressource rare, c'est la capacité de revue humaine. Les limites de travail en cours doivent désormais borner le nombre de PR générées par des agents ouvertes simultanément, sous peine de noyer les relecteurs. Kanban ne meurt pas : il devient, de fait, le langage visuel naturel du tableau de bord d'une usine agentique.

XP — renforcé

XP est le cas le plus intéressant, parce que ses pratiques sont d'ingénierie et non des rituels de coordination Le TDD ne disparaît pas, il devient quasi obligatoire : l'exigence de critères Given/When/Then exécutables avant tout build est du TDD élevé au rang de contrat. L'intégration continue ne s'affaiblit pas, elle devient le verrou central. Les hooks et la CI sont littéralement le mécanisme déterministe qui contrebalance la nature probabiliste des agents.

Les standards de code survivent sous une forme méconnaissable mais fidèle à l'esprit : ce sont AGENTS.md et CLAUDE.md, des standards enfin faits pour être appliqués mécaniquement plutôt que rappelés de mémoire en revue de code. Le pair programming, en revanche, mute profondément : le binôme devient humain-agent plutôt qu'humain-humain. La propriété collective du code recule partiellement — les impératifs d'audit NIS2 et AI Act réintroduisent une propriété nommée par zone. Et le client sur site s'efface au profit d'un travail amont beaucoup plus rigoureux de captation de sa voix dans le PRD.

SAFe — le plus exposé

SAFe est le plus exposé des quatre à un vrai risque d'obsolescence, précisément parce que son rituel signature — le PI Planning, deux jours en salle tous les trimestres pour découvrir les dépendances entre équipes — répond à un problème que la traçabilité continue du graphe de spec résout structurellement mieux. Le rôle de Release Train Engineer se contracte pour la même raison que le Scrum Master, mais à l'échelle du portefeuille — vers quelque chose comme un architecte de gouvernance multi-repos.

Ce qui survit de SAFe, c'est sa couche vraiment stratégique : le Lean Portfolio Management, l'arbitrage budgétaire parce que ce sont des décisions de jugement humain sur l'allocation de ressources rares.

Le fil conducteur

Tout ce qui existait pour compenser un déficit de visibilité entre humains disparaît ou se réduit drastiquement. Tout ce qui portait une discipline d'ingénierie déterministe ou une décision de jugement humain survit et se trouve même renforcé par l'exigence de rigueur qu'imposent des agents qui, eux, ne comblent jamais une ambiguïté par du bon sens.

CHAPITRE 04

Que deviennent les rôles ?



Cinq portraits, un même fil : distinguer la part administrative de chaque rôle, celle qui recule et sa part de jugement et de gouvernance, celle qui grandit.

Le Scrum Master : la moitié administrative meurt

La question mérite qu'on distingue les trois fonctions que recouvre aujourd'hui le rôle, parce que la révolution agentique ne les frappe pas de la même façon.

La fonction la plus exposée, c'est la facilitation de cérémonies au sens strict: sprint planning, daily standup, mêlée de synchronisation. Elle repose sur un postulat qui s'effondre dès qu'on gouverne un dépôt de code à coups de PRD à sections atomiques et de critères Given/When/Then. Un Scrum Master

dont toute la valeur ajoutée est de faire tourner Jira et d'animer ces rituels se trouve, structurellement, dans la même position qu'un développeur dont toute la valeur ajoutée est d'écrire du code répétitif.

La deuxième fonction, le déblocage d'obstacles, ne disparaît pas, elle change de nature. L'obstacle typique n'est plus « l'équipe X ne priorise pas notre dépendance », mais plutôt « le PRD n'a pas de NFR chiffrées », « cet ADR n'a pas d'alternative écartée », « AGENTS.md ne couvre pas ce cas limite et l'agent improvise ». C'est exactement le travail d'un mainteneur de Constitution ou d'un propriétaire d'AGENTS.md — un rôle de gardien du graphe de spec. Un bon Scrum Master a déjà les réflexes pour occuper ce poste ; c'est un déplacement latéral, pas une reconversion complète.

La troisième fonction, le coaching de la maturité d'équipe, est probablement celle qui grandit le plus. C’est la conduite du changement que doit mener le Scrum Master.

Le résumé tient en une phrase : le rôle ne meurt pas, mais sa moitié administrative, oui — et seule sa moitié gouvernance et changement survit.

Le Product Owner : plus central, moins approximatif

Là où le Scrum Master voit sa fonction historique s'éroder, le Product Owner est plutôt appelé à gagner en centralité, mais au prix d'une exigence de rigueur qui rend une partie des PO actuels obsolètes s'ils ne s'y adaptent pas. Le renversement tient en une phrase : la spec devient l'artefact source, le code n'en est qu'un artefact dérivé.

Historiquement, cinq à dix pour cent du cycle allait dans la qualité du PRD ; l'essentiel se négociait ensuite en réunion, et le PO pouvait rester approximatif sur le fond parce que des humains combleraient les ambiguïtés en cours de route par inférence contextuelle. Cette marge de manœuvre disparaît dès qu'un agent exécute la spec littéralement : il ne s'arrête pas pour demander une clarification, il choisit une interprétation plausible — et quand plusieurs agents travaillent en parallèle, chaque ambiguïté se multiplie en divergences coûteuses à réconcilier.

Concrètement, le métier se déplace sur trois axes. La forme, d'abord : un problem statement, des objectifs SMART chiffrés et des non-objectifs explicites, des critères d'acceptation en Given/When/Then systématiques sont désormais obligatoires.. L'autorité, ensuite : le PO ne signe plus seul. Le Sign-Off à six dimensions produit, technique, sécurité, légal, UX, opérationnel, dilue son pouvoir de décision unilatéral dans une gouvernance partagée. La frontière de compétence, enfin : le PO répond au « quoi et pour qui », jamais au « comment technique ».

Les responsabilités qui restent sont plus étroites qu'avant, mais non délégables : le PO porte spécifiquement la dimension Valeur du Sign-Off: garantir que le problème est réel, que la métrique de succès est mesurable, que l'impact business est justifié. Un PO qui reste au niveau du backlog construit à l'instinct, sans NFR quantifiées, sans kill-switch documenté, devient le goulot d'étranglement de toute la chaîne, pire, une source de risque.

Le développeur : de l'auteur au vérificateur

Le développeur est le rôle sur lequel la rupture est la plus directe, parce que c'est littéralement sa matière première — écrire du code qui cesse d'être rare. Mais l'endroit où se loge la valeur ne disparaît pas : il migre vers l'amont et vers le contrôle.

La partie la plus exposée est celle que la doctrine agentique dit que l'humain pilote, l'agent suggère ou exécute un plan validé étape par étape.

Ce qui ne se délègue pas, en revanche, grandit en importance. D'abord l'autorité décisionnelle : un ADR engage une responsabilité humaine, un choix irréversible ou coûteux à inverser, justifié par des alternatives explicitement écartées et aucun agent ne peut légitimement l'endosser à la place d'un architecte ou d'un tech lead. Ensuite l'écriture des conventions elles-mêmes : AGENTS.md et CLAUDE.md ne sont pas de la documentation annexe, ce sont les fichiers qui déterminent ce que produiront tous les agents du repo. Les maintenir vivants, est devenu une compétence d'ingénierie à part entière.

Le troisième déplacement est le plus structurant : du rôle d'auteur au rôle de vérificateur et d'orchestrateur. Sans automatisation des règles, la revue humaine devient le goulot d'étranglement. Le développeur qui survit à l'échelle est celui qui sait choisir le bon pattern d'orchestration, qui plafonne les itérations et pose un kill-switch accessible en moins de trente secondes, qui instrumente l'observabilité qui trace le coût par run. Un mélange de compétences d'architecte logiciel et de SRE appliqué à des flottes d'agents.

Reste un point de vigilance honnête : cette bascule pose un vrai problème de filière. Le développeur senior d'aujourd'hui s'est formé en écrivant justement le code que les agents écrivent maintenant à sa place et la question de savoir comment former les juniors de demain sans ce passage obligé n'a pas de réponse consensuelle dans l'industrie.

L'architecte : le rôle qui résiste le mieux

L'architecte est, des rôles passés en revue, celui dont la fonction centrale résiste le mieux parce que la doctrine entière repose sur un principe qui le concerne directement : la décision est l'artefact source, le code n'en est que l'expression. Tant que ce principe tient, quelqu'un doit continuer à prendre et à justifier les décisions.

Le premier déplacement est celui du format. Un architecte qui produit des documents de solution en diapositives, déconnectés du repo de code, voit son influence s'évaporer sans qu'aucun agent ne l'ait « remplacé » : une décision qui ne vit pas dans un ADR chargeable au runtime n'a plus d'effet sur ce que produisent les agents. Le deuxième déplacement élargit son périmètre : l'architecte devient le garant de la Constitution, et celui qui fixe, zone par zone les niveaux d'autonomie. Le troisième déplacement est proprement nouveau : l'architecture de la fabrique agentique elle-même: choisir entre un workflow à chemins prédéfinis et un agent à autonomie dynamique. Choisir le pattern avant le framework, jamais l'inverse.

Les responsabilités qui l'accompagnent sont celles d'un gardien de mémoire plus que d'un rédacteur. Sa responsabilité centrale est l'honnêteté du document. Le corollaire est une responsabilité accrue plutôt qu'un rôle en déclin. Le risque n'est donc pas l'obsolescence, mais l'écart entre deux profils d'architectes : celui qui continue à produire des artefacts narratifs sans levier opérationnel sur les agents, et celui qui écrit des décisions exécutables et gouverne l'autonomie effective de la fabrique.

UX Designer et coach agile : le jugement protégé

Pour le designer, un fait structurant se dégage : le Design occupe l'une des six signatures obligatoires du Sign-Off, la dimension Utilisabilité, au même rang que le Tech Lead sur la Faisabilité ou le QA sur la Vérifiabilité. Cette autorité ne protège que la partie jugement du métier, pas la partie production. Produire des maquettes ou des variantes de composants à partir d'un design system existant est un travail répétitif et fortement pattern-matchable, exactement le type de tâche que les outils génératifs commencent à absorber. Ce qui résiste, c'est le jugement sur l'expérience humaine réelle — la recherche qualitative, l'interprétation de signaux ambigus, l'empathie pour un cas limite. Le déplacement le plus concret : écrire pour être lu par un agent, pas seulement par un développeur humain, avec des états intermédiaires documentés comme des exigences testables et une accessibilité chiffrée comme n'importe quelle autre exigence non fonctionnelle.

Le coach agile, lui, est de tous les rôles passés en revue celui qui a le moins à craindre de la bascule — non parce qu'il serait protégé par une signature de gouvernance, mais parce que sa fonction n'a jamais été de combler un déficit de visibilité opérationnelle. Le coach agile a toujours vendu du changement de comportement humain, pas de la synchronisation de statut et le changement de comportement humain est justement ce que cette transition demande en quantité industrielle.

Ce qui change, c'est le contenu du coaching. Le nouveau terrain à trois dimensions : accompagner le deuil professionnel réel que vit un développeur qui passe d'auteur de code à réviseur d'agents ; coacher l'écriture, en faisant monter en compétence

PO, designers et développeurs sur la rigueur contractuelle du PRD et de l'ADR ; et calibrer la confiance humaine dans un système probabiliste, en évitant à la fois le rubber-stamping des PR d'agents par lassitude de la revue, et la sur-supervision paralysante d'un agent pourtant fiable.

CHAPITRE 05

Les artefacts et les cérémonies ce qui meurt, mute, survit



La grille de lecture la plus utile consiste à se demander pourquoi chaque artefact agile existait. Certains servaient à pallier un manque de visibilité entre humains ; d'autres à porter une responsabilité ou une intention. Les premiers s'effacent dès que la traçabilité devient continue et automatique. Les seconds survivent, parce que la vitesse d'exécution ne supprime ni la responsabilité ni l'intention , elle les rend simplement plus urgentes à formaliser correctement.

Ce qui disparaît

Le sprint comme unité de planification, avec son sprint planning et son sprint backlog, s'atrophie au point de devenir résiduel : un découpage en tranches de deux semaines n'a plus de sens quand la granularité de travail devient la journée-agent, voire le run d'agent unique. Le daily standup comme rituel de synchronisation de statut disparaît pour la même raison : l'état du système est lisible en permanence via la traçabilité constitutionnelle. Les story points et la vélocité s'effondrent aussi, parce qu'ils mesuraient une incertitude d'effort humain ; le nouveau coût à piloter, c'est le token, le run, la latence. Le sprint review en tant que démonstration formelle bihebdomadaire cède la place à des paliers de rollout continus, avec des critères de décision chiffrés.

Ce qui évolue

Le nom ou l'intuition reste ; la substance change complètement de nature et de support. La user story ne meurt pas, mais elle mute d'un post-it « en tant que... je veux... afin de... » vers une exigence fonctionnelle atomique, à identifiant stable, verbe testable, liée à un ADR et à des critères Given/When/Then. La Definition of Done évolue d'une checklist discutée en réunion vers un ensemble de vérifieurs déterministes exécutés en intégration continue et en hooks git. Le product backlog évolue d'une liste de tickets priorisés vers un pipeline d'intentions et de PRD à différents stades de maturité, le graphe de spec en est la forme moderne. La rétrospective survit dans sa fonction d'amélioration continue, mais son objet change : elle porte moins sur « comment on a travaillé ensemble » que sur l'audit

des artefacts de gouvernance beaucoup plus proche d'un post-mortem SRE que d'un rituel d'équipe.

Ce qui survit

Quasiment inchangés dans leur fonction : le RFC ou Design Doc, voisin du PRD et de l'ADR pour le « comment » d'une solution ponctuelle. La Definition of Ready survit aussi, simplement durcie en check-list binaire opposable avant le Sign-Off. La roadmap reste nécessaire, mais recentrée sur une trajectoire de métrique North Star et de contre-métriques plutôt que sur une liste de fonctionnalités promises par trimestre.

Le fil directeur : tout ce qui servait à faire circuler de l'information entre humains disparaît, remplacé par la traçabilité native des artefacts. Tout ce qui portait une décision, une intention ou une responsabilité survit et se durcit — parce que rendre cette responsabilité explicite et opposable est précisément la condition pour qu'une chaîne d'agents puisse s'exécuter sans dérive.

CHAPITRE 06

La nouvelle économie des rôles

Quels sont les nouveaux rôles qui émergent ? Une recherche externe fait apparaître des familles de métiers organisées par fonction plutôt que par ordre alphabétique: les titres se recoupent beaucoup d'une source à l'autre, ce qui est en soi un signal que le marché n'a pas encore stabilisé sa nomenclature.

Construire les agents

L'Agentic AI Engineer, ou AI Agent Engineer, construit les boucles d'agent proprement dites : tool calling, orchestration de sous-agents, mémoire, harnais d'évaluation. C'est le rôle le plus proche du développeur traditionnel, repositionné. L'AI Agent Architect conçoit l'architecture système multi-agents — un rôle senior, souvent issu d'architectes backend ou de plateformes ML. Le Context Engineer succède au Prompt Engineer en élargissant le périmètre : pipelines RAG, architecture mémoire, intégration d'outils, garde-fous de gouvernance. Le LLM Engineer construit applications et infrastructure au-dessus des grands modèles de langage, souvent le point d'entrée le plus généraliste vers les rôles agentiques.

Faire tourner les agents en production

L'AgentOps Engineer, ou AI Operations Manager, applique la discipline SRE et MLOps à des systèmes non déterministes : observabilité, tracing, évaluation continue, coût et latence, gestion de version des modèles et des prompts, réponse aux incidents. L'Evals Engineer conçoit les systèmes de vérification

de la qualité des sorties d'IA. Le Verification Engineer combine QA traditionnelle et validation métier; un rôle jugé particulièrement critique en fintech, santé, légal et paiements.

Gouvernance, sécurité, conformité

L'AI Trust & Safety Lead pose les politiques d'usage acceptable, conçoit les systèmes de modération. L'AI Security Engineer, ou Red-team Engineer, mène les tests L'AI Compliance/Legal Advisor traduit la réglementation — AI Act européen, règles sectorielles — en décisions produit concrètes. Le Chief AI Officer porte la stratégie IA, le risque et la gouvernance au niveau exécutif.

Produit et design

L'AI Agent Product Manager porte la roadmap, la stratégie d'évaluation, les arbitrages coût, qualité et latence; proche du Product Owner tel que décrit plus haut, mais formalisé côté marché anglo-saxon. L'Agentic Workflow Designer traduit des processus métier en workflows agentiques et détermine où placer l'humain dans la boucle. L'AI UX Researcher étudie le comportement des utilisateurs face à des produits IA non déterministes.

Développeur repositionné plutôt que nouveau titre

Plusieurs sources notent que ce n'est pas un nouveau titre qui émerge, mais un empilement de responsabilités nouvelles sur le titre existant de développeur ou de tech lead : validateur d'architecture, relecteur sécurité et conformité, orchestrateur de workflows IA, gardien de la qualité de code, propriétaire de la gouvernance et de l'accountability.

Un point de vigilance : les titres de façade

Une partie de ces intitulés est explicitement qualifiée de « titres de façade » plutôt que de rôles réellement distincts : Builder, Vibe Coder, AI-Native Developer, ou des titres gonflés comme « Principal Agentic Context Architect », qui seraient surtout des habillages marketing ou des tentatives d'échapper aux grilles de salaire existantes. Le Context Engineer lui-même est cité, par une source, comme partiellement dans cette zone grise. Cela vaut la peine d'être gardé en tête avant de recopier ces titres tels quels dans une fiche de poste.

CHAPITRE 07

SWOT : l'agilité face au SDLC agentique

Les valeurs fondatrices survivent mieux que les rituels qui les incarnaient. Le bilan qui suit le montre en quatre temps.

Forces

Le flux visuel façon Kanban sort renforcé : piloter des agents en parallèle, c'est du WIP-limit à l'échelle. XP gagne un second souffle; le TDD et l'intégration continue deviennent le filet de sécurité indispensable quand le code se génère plus vite qu'un humain ne peut le relire. Et l'inspect-adapt reste, sous une autre forme, le seul antidote connu au drift entre spec et réalité.

Faiblesses

Le daily stand-up et le planning poker perdent leur objet quand la vélocité de code n'est plus la contrainte. SAFe s'effondre sous son propre poids bureaucratique, trop lent pour un système qui livre en continu. Les story points mesurent un effort de frappe qui a cessé d'exister. Et le Scrum Master, sans rituel à faciliter, doit se réinventer ou disparaître.

Opportunités

Une nouvelle économie de rôles s'ouvre — Agentic AI Engineer, Context Engineer, AgentOps pour qui sait s'y positionner tôt. Le PRD et l'ADR cessent d'être de la paperasse de gouvernance pour devenir l'artefact source qui pilote littéralement

l'exécution : Product Owner et architecte gagnent en levier stratégique ce qu'ils perdent en micro-management du sprint. Le coach agile peut monter en gamme vers l'accompagnement du changement organisationnel plutôt que l'animation de rituels. Et c'est l'occasion de vendre ce que peu d'acteurs structurent aussi bien : la gouvernance, de la Constitution aux Patterns, comme chaînon manquant entre la promesse agentique et sa mise en production sérieuse.

Menaces

Le vrai danger n'est pas le retour du cycle en V — c'est sa version non assumée : des specs qu'on croit vivantes mais qu'on ne remet jamais à jour, pendant que le code dérive en silence. La confusion des titres « Vibe Coder », « Principal Agentic Context Architect » dilue la crédibilité du marché aussi vite qu'elle l'a créée. Les chiffres marketing invérifiables exposent à une désillusion brutale au premier projet raté. Et sans traçabilité rigoureuse, la vitesse agentique devient un passif réglementaire plutôt qu'un actif.

Conclusion

> *L'agilité ne meurt pas, elle se désincarne ou se réinvente. Les rituels s'effacent, les valeurs se réinjectent dans la gouvernance documentaire.*

Ni les nostalgiques du daily, ni les évangélistes du tout-agent : ceux qui savent spécifier vivant.

L'auteur

Pour poursuivre la conversation.

Luc Legardeur
luc.legardeur@wescale.fr · linkedin.com/in/luclegardeur