Collaboration entre développeurs humains et agents IA pour la R&D en informatique

Bryan Fruchart[1], Benoit Le Blanc[2]

[1] DIMARC, Euratechnologies, Lille

[2] Ecole nationale supérieure de cognitique, IMS UMR-5218 CNRS–UB–Bordeaux-INP

mél : <bryan@dimarc.fr>

Résumé

La collaboration entre développeurs et outils d'IA peut se faire dans l'action, bien au-delà du simple échange de messages. Inspirés par la communication stigmergique, nous proposons une démarche de développement logiciel dans un environnement dédié à la coordination humains-IA. Des catégories de fichiers Markdown servent de traces persistantes entre agents lancés en sessions parallèles et sept étapes de développement permettent un flux de développement contrôlé. Cet environnement est utilisé au quotidien pour la R&D d'une entreprise qui commercialise des agents IA personnalisés, destinés à automatiser la recherche, l'extraction et la synthèse de documents.

Mots-clés

Agents IA, Collaboration humains-IA, Environnement de développement, Stigmergie.

Abstract

Collaboration between developers and AI tools can happen in action, far beyond the simple exchange of messages. Inspired by stigmergic communication, we propose a software development approach within an environment dedicated to human-AI coordination. Some categories of Markdown files serve as persistent traces between agents running in parallel sessions, and seven development steps enable a controlled development flow. This environment is used daily for the R&D of a company that markets customized AI agents designed to automate document search, extraction, and synthesis.

Keywords

AI Agents, Human-AI Collaboration, Development Environment, Stigmergy.

1 Introduction

En informatique, la coordination du travail entre les développeurs a constamment fait l'objet d'ajustements, d'innovations et de transformations. Langages de modélisation, programmation par objets, méthodes agiles, plateformes de gestion de code, revues de code source, chaines d'outils, etc. sont autant de concepts qui ont marqué les périodes du développement logiciel, avec leur cohorte d'anglicismes associés [4, 10, 1]. Aujourd'hui, l'entrée en jeu des « agents IA » marque un changement de paradigme. Un agent IA est un logiciel capable d'exécuter des tâches de manière autonome, pour le compte d'un utilisateur, ici un développeur logiciel. Mais cette collaboration devrait se faire dans l'action, et dépasser ainsi le simple échange de messages entre un prompt du développeur exposant son besoin et les morceaux de code que lui retourne un agent conversationnel. Pourtant, les outils actuels d'assistance du développeur tels que l'auto-complétion [5], ou même la construction d'agents de codage autonomes [24], continuent de considérer l'agent informatique comme un opérateur ponctuel, sans aucune projection de son action à l'échelle du projet de développement dans son entièreté.

Notre constat de départ est que l'agent conversationnel actuel est sans mémoire. Il peut disposer d'un historique des interactions si le développeur s'est identifié, mais chaque session repart de zéro quant au code qui est à générer. De ce fait, le développeur ne travaille pas avec un agent IA, mais avec des dizaines d'instances successives de cet agent prototype. Aussi, le principe de l'interaction par échange de messages ne peut pas passer à l'échelle du développement d'un projet informatique complet.

La question qui en ressort est celle de savoir comment coordonner les efforts du développeur avec les apports que lui amène l'agent IA. Pour explorer cette question, nous avons suivi la piste cherchant à structurer l'environnement, plutôt que de cadrer, ordonner ou articuler la « conversation » entre le développeur et son agent IA. Cette structuration de l'environnement entre agents opérant chacun à sa façon, rappelle la communication stigmergique.

2 État de l'art

Le concept de stigmergie a été avancé par le zoologiste français Pierre-Paul Grassé pour expliquer la façon dont les termites constructeurs se coordonnent sans communication directe entre les individus [11]. Chacun travaille de son côté et pose des traces à destination de ses congénères. Ces traces laissées dans l'environnement à chaque action, vont venir stimuler l'exécution d'actions complémentaires de la part des autres individus de la colonie. Il y a bien un « projet global » qui en émane, sans qu'aucun individu n'en ait le plan, l'intention ou le dessein.

Sur cette base, une distinction a été posée entre la stigmergie quantitative de la stigmergie qualitative [22]. La première se fonde sur la modulation de la réponse de l'agent en fonction de la concentration d'un signal. C'est ce qui se passe avec les

phéromones des fourmis, plus ou moins concentrées. La seconde repose sur des stimuli différents pour déclencher des réponses différentes. C'est ce que l'on retrouve chez les termites.

La stigmergie, en tant qu'intelligence en essaim, a été transposée vers le développement logiciel il y a déjà une vingtaine d'années, avec le modèle ACO (Ant Colony Optimization) [8] : des agents artificiels déposent des traces numériques pour guider l'exploration collective d'un espace de solutions.

L'idée d'employer la stigmergie comme un mécanisme de coordination universel apparait un peu plus tard [13, 14] : chaque action d'un agent laisse une trace disponible à tous et sert d'origine à une autre action d'un autre agent qui viendra modifier cette trace. Il s'agit là d'une coordination sans planification centralisée, sans communication directe, et ne nécessitant pas la co-présence des agents sur un même lieu au même moment. Le travail collectif fonctionne par lecture et modification des mêmes traces. Ce mécanisme permet d'échafauder des constructions très complexes sans que les actions ne soient planifiées.

La stigmergie a aussi été mobilisée comme l'outil intellectuel pour comprendre comment la collaboration humaine à très grande échelle se développe [9]. Ainsi, l'encyclopédie collaborative Wikipedia, ou encore le mouvement de développement logiciel Open Source fonctionnent selon une forme de stigmergie. Le résultat global émane des actions des uns et des autres, par petites touches sollicitées par des messages laissés à la collectivité : « section vide », « référence incomplète », « article incompréhensible » pour Wikipedia, ou tout simplement des relevés de Bugs pour le mouvement du logiciel libre. Chaque message stimule de nouvelles contributions.

En épistémologie, considérant que la connaissance sociale est fondamentalement stigmergique, des chercheurs ont mis en évidence l'intérêt de déterminer quel est le vecteur de support du savoir partagé [17]. C'est cette piste qui a été explorée pour proposer un cadre au développement du logiciel libre [3]. Les équipes FLOSS (Free/Libre Open Source Software) coordonnent leur travail par le code en lui-même. L'objet du travail sert de médium stigmergique, combinant à la fois des mécanismes implicites, comme l'état du code, et des mécanismes explicites, comme les conventions de partage par exemple.

Pour qu'une coordination stigmergique puisse fonctionner, il semble nécessaire de formaliser trois affordances socio-techniques [6] : la visibilité du travail, sa combinabilité, et des genres définis de contributions.

Par ailleurs, l'émergence des modèles de langage basés sur des textes de codes informatiques a permis de générer automatiquement des bouts de logiciel [5], ouvrant également la voie à des recherches sur la fabrication d'agents de développement logiciel autonomes. Park et al.[18] ont montré que des agents à base de LLM et évoluant dans un environnement partagé peuvent produire des comportements sociaux émergents, via un « flux de mémoire » (memory stream) dans un langage qui nous est naturel. Il s'agit là aussi d'un mécanisme de stigmergie, même si les auteurs n'emploient pas ce terme.

ReAct est une autre approche appliquée à un ensemble diversifié de tâches de langage et de prise de décision qui démontre une certaine efficacité par rapport à des méthodes de référence, en plus d'une meilleure interprétabilité et fiabilité humaines [25]. Dans ReAct, l'agent alterne entre traces de raisonnement et actions, ce qui traduit bien là aussi une démarche stigmergique.

Dans le domaine du génie logiciel, des propositions exploitant les LLM commencent à arriver. ChatDev [20] ou MetaGPT [15] orchestrent plusieurs agents spécialisés, exploitant des LLM, et se coordonnant avec des artefacts partagés, tels que des spécifications, du code ou des tests. Un panorama des systèmes multi-agents les plus récents offrant des solutions à base de LLM pour les différentes étapes du cycle de vie du développement logiciel a même été dressé [12]. Les auteurs concluent que le domaine est déjà très compétitif mais que des recherches doivent encore être menées sur l'amélioration des capacités individuelles des agents et l'optimisation de leur synergie.

Une équipe de l'Université de Princeton propose un benchmark (SWE-bench [16]) et une architecture (SWE-agent [24]) avec lesquels ces chercheurs montrent que la capacité d'un agent à résoudre des problèmes réels va dépendre de la qualité de son interface avec l'environnement.

Des réalisations récentes revendiquent des performances, comme [23] qui décrit une infrastructure basée sur trois composants (une mémoire active, des agents spécialisés, une base de connaissance via des documents) et employée pour un développement logiciel comprenant plus de cent mille lignes de C#. Toutefois, certaines études de l'impact sur la productivité des développeurs rendent des avis contrastés. Sans surprise, l'autocomplétion (telle que GitHub Copilot) accélère les tâches isolées avec toutefois un score que l'on peut considérer comme modeste, allant de 26% à 56% [19, 7]. Mais [2] observe dans certains cas un handicap de cette autocomplétion chez des développeurs expérimentés lorsqu'ils travaillent sur leur propre code. Enfin [21] présente une étude qui montre que les agents réorientent l'effort des travailleurs, de la mise en œuvre vers la supervision, ce qui profite particulièrement au travail auditable et aux travailleurs experts. D'une certaine façon, le travail humain passe du syntaxique (rédaction du code informatique) au sémantique (spécifier les besoins et évaluer les résultats).

3 Notre approche

Afin de procéder à cette réorientation du travail du codeur vers la seule supervision, laissant les tâches syntaxiques à la machine, nous avons développé une approche par primitives stigmergiques. Nous reprenons les trois propriétés de [6] pour caractériser ces primitives : visibles (par des fichiers textes dans le répertoire de travail), combinables (par un héritage en cascade) et typées (par un rôle défini attribué à chaque primitive).

Nous proposons ainsi un environnement de développement où humain et agents se coordonnent par un ensemble de traces que chacun laisse dans des fichiers Markdown. Ces fichiers sont des traces persistantes, partagées, et qui guident le comportement de chaque agent, sans nécessité de

synchronisation directe. L'environnement modifié par l'un stimule l'action de l'autre. On retrouve là le point fondamental appelé « mécanisme universel de coordination » par [13, 14].

3.1 Architecture de l'environnement

Un espace de travail multi-domaines est constitué avec trois zones :

work/
- **codebase/** # Monorepo Python — 5 applications IA
- **gestion/** # Pilote deux entreprises commerciales et un espace personnel
- **doc/** # Recherche scientifique, rapports et publications, dossiers de candidatures

Un même agent opère dans les trois domaines : ce n'est pas l'agent qui change ou se spécialise, c'est l'environnement dans lequel il évolue qui va venir typer l'action.
Lorsque l'agent pénètre dans un dossier, il charge automatiquement le fichier CLAUDE.md du dossier courant ainsi que ceux de tous les dossiers parents. Ce mécanisme crée un héritage de contexte sans qu'aucune instruction ne soit répétée. Le CLAUDE.md racine porte sur des principes de haut niveau, tels que « comprendre avant d'agir », « chaque action a une intention », « isolation stricte ». Chaque niveau hiérarchique vient compléter avec ses propres fichiers CLAUDE.md :
- Niveau racine (work/) : principes de travail, profil, langue.
- Niveau domaine (codebase/, gestion/, doc/) : stack, discipline, nature du travail.
- Niveau application (assistant/, extracteur/, ...) : config LLM, contraintes spécifiques.

Ce mécanisme d'héritage rappelle la factorisation par classes en Programmation orientée objets et satisfait les trois affordances de Crowston et al. [6] : la visibilité est assurée par le fait que tout se passe dans des fichiers texte que l'on peut inspecter ; la combinabilité provient de l'héritage en cascade ; les genres sont bien définis puisque chaque niveau a un rôle.

3.2 Primitives comme traces stigmergiques

Pour chaque application, nous suivons un format standardisé, avec neuf fichiers qui viennent décrire les primitives qui constituent l'interface de coordination :
- CLAUDE.md : contient les instructions pour l'agent, et la table de routage vers les autres primitives.
- VISION.md : correspond à la boussole du projet (quoi, pourquoi, comment, expérience cible).
- SPECS.md : correspond au contrat fonctionnel, ainsi qu'au contrat d'interface pour l'intégration.
- ARCHITECTURE.md : contient la carte technique (diagrammes Mermaid, modules, dépendances).
- TODO.md : est la mémoire de travail partagée (versions, tâches, notes de session).
- README.md : est l'onboarding, où l'on retrouve les commandes de lancement.
- TESTING.md : reprend la stratégie de test (6 niveaux, conventions de mock, matrice).
- BENCHMARKING.md : contient le protocole de benchmark (métriques, tiers, résultats historiques).
- LESSON_LEARNED.md : reprend les leçons actionnables, extraites des sessions passées.

Chaque primitive a une audience définie (agent, développeur, partenaire), un rôle précis et une fréquence de mise à jour. L'agent lit les primitives pour comprendre le contexte ; l'humain les met à jour pour guider l'agent.
En termes stigmergiques, chaque primitive est une trace qualitative [22] : un stimulus typé qui déclenche un comportement spécifique de l'agent (SPECS.md → implémenter selon le contrat ; TODO.md → prioriser les tâches ; lesson_learned.md → éviter les erreurs passées).
Le flux associé à la session se déroule en sept étapes :
1. LIRE → TODO.md + SPECS.md
2. CADRER → VISION.md si le besoin n'est pas clair
3. PLANIFIER → Mode plan (sauf patch < 10 lignes)
4. VALIDER → Attendre confirmation humaine
5. DÉVELOPPER
6. TESTER → Non-régression + tests pour chaque nouvelle fonction
7. CLORE → Mettre à jour TODO.md + ARCHITECTURE.md

Ce flux est inscrit dans chaque CLAUDE.md d'application. L'agent le suit systématiquement. La validation humaine (étape 4) est obligatoire pour tout changement non trivial.

3.3 Approches Test-Driven et Benchmark-Driven

Les tests doivent être écrits avant de commencer la création du code. Chaque version commence par des tests RED (c-à-d la définition du contrat), puis l'implémentation va venir les passer au GREEN.
Il y a six niveaux de tests : unitaire modèles, unitaire modules, fonctionnel workflows, fonctionnel scénarios, E2E, non-régression.
Tout ceci se fait avec une règle absolue : ne jamais faire appel au réseau durant les tests. En conséquence, tous les LLM, API et I/O sont mockés.
Tout changement structurel doit être benchmarké avant/après. Ceci repose sur une répartition en trois tiers : environ 6 questions pour le smocke, une trentaine de questions pour le standard et une soixantaine de questions pour l'approfondissement.
Un double score est effectué, avec un score quantitatif (latence, tokens, steps, taux d'erreur) et un score qualitatif (LLM-as-a-judge, 4 dimensions notées entre 0 et 10 pour l'exactitude, la complétude, la pertinence, et le format).

3.4 Savoir-faire codifié

Les skills, ou savoir-faire, sont des prompts spécialisés invocables par commande slash, organisés en cascade comme les CLAUDE.md (globales, par domaine, par app).
Il y a trois catégories de skills.

Les premiers sont les skills globaux. On y trouve :
/introspect — Distiller la session en savoir durable. L'agent relit la conversation, identifie ce qui est durable et structurant, propose des ajouts aux CLAUDE.md au bon niveau de la cascade. Filtre : pas d'état temporaire, pas de duplicata, pas d'hypothèse non validée.
/skill-craft — Forger et affûter les skills. L'agent scanne les skills existants, diagnostique leur pertinence, propose des

améliorations ou de nouveaux skills. 5 critères : nécessaire, précis, actionnable, court, testable.

/finish — Clôture de session. Enchaîne /skill-craft puis /introspect. Le savoir-faire d'abord (skills), le savoir ensuite (CLAUDE.md). Chaque passe a son propre cycle « proposition/validation ».

/clean — Confronte la structure réelle du répertoire au CLAUDE.md de ce niveau. Identifie les écarts et propose de mettre à jour la documentation ou de traiter les fichiers.

Les deuxièmes sont les skills de développement (codebase/) :

/test <app> — Dashboard de santé avec détection automatique des stubs.

/state — Vue d'ensemble du monorepo (versions, progression, résumé).

/migrate-app — Migration d'une app externe vers le framework en quatre phases.

Les troisièmes sont les skills métier :

/dossier (doc/) — Révision de dossiers de candidature contre un cahier des charges : conformité, cohérence interne, corrections systématiques.

/debrief (gestion/) — Structuration de retours de RDV : faits, décisions, actions, chiffres, alertes. Cascade automatique dans les fichiers de suivi.

3.5 Boucle de rétroaction

Le système évolue à travers trois boucles de rétroactions.

Il y a tout d'abord la boucle intra-session : Le flux en 7 étapes met à jour TODO.md et ARCHITECTURE.md. Les notes de session capturent le POURQUOI des décisions, pas seulement le QUOI.

Il y a ensuite les boucles inter-sessions (/finish) : En fin de session, l'agent distille l'expérience, d'abord en savoir-faire (nouveaux skills ou améliorés), puis en savoir (CLAUDE.md enrichis), avec une validation humaine avant chaque modification.

Il y a enfin les boucles structurelles (/clean) dont l'objet est de produire une confrontation périodique entre la réalité du répertoire et la documentation. L'environnement se nettoie et se documente progressivement.

En conséquence de quoi, chaque interaction laisse des traces qui améliorent les interactions suivantes. C'est le mécanisme stigmergique fondamental [13, 14] : l'environnement s'enrichit session après session, et la qualité de la coordination croît, sans que l'agent lui-même ne soit modifié.

4 Résultat

L'environnement présenté ci-dessus est utilisé au quotidien pour la R&D d'une entreprise qui développe des agents IA spécialisés (extraction, RAG, rédaction, assistant dual-process, architecture cognitive). Il fonctionne sur plusieurs applications Python en parallèle pour le développement logiciel, la gestion d'entreprises et la rédaction de dossiers :

saas/ (produits SaaS)
- assistant/ — assistant dual-process (proche du S1/S2)
- documentalist/ — Q-R sur doc. privés (RAG dual-index)
- sergic_documentalist/ — fork vertical client (04/ 2026)
- extracteur/ — extraction structurée
- parser/ — extraction intelligente de fichiers
- docuv2/ — chantier en cours

commandes/ (livraison client)
- rse_writer/ — co-rédacteur de rapports RSE

research/ (R&D scientifique)
- agi/ — démonstrateur architecture cognitive IPSEL
- IPSEL_ARC/ — agent IPSEL testé sur ARC-AGI-3
- deep_flow/ — banc d'essai architect. cognitive profonde

Deux entreprises commerciales sont gérées via le même paradigme, et des rédactions de documents administratifs ou de réponses à appel à projets utilisent ces outils d'IA.

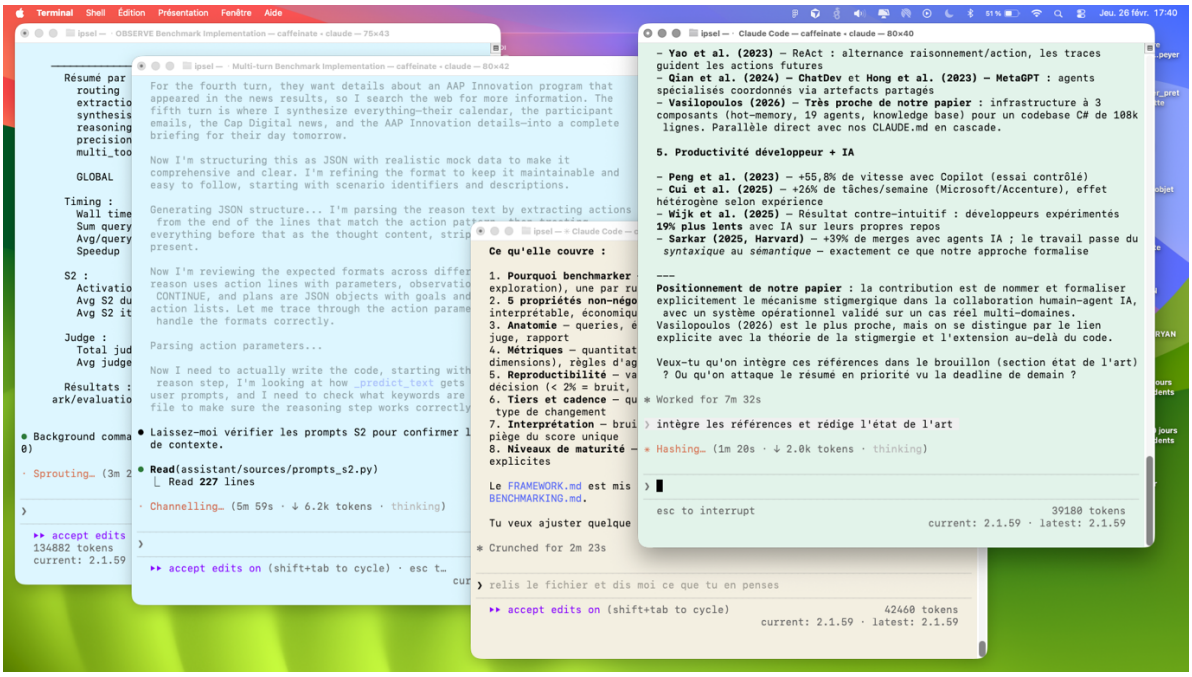

Figure 1 : Exemple de session de travail

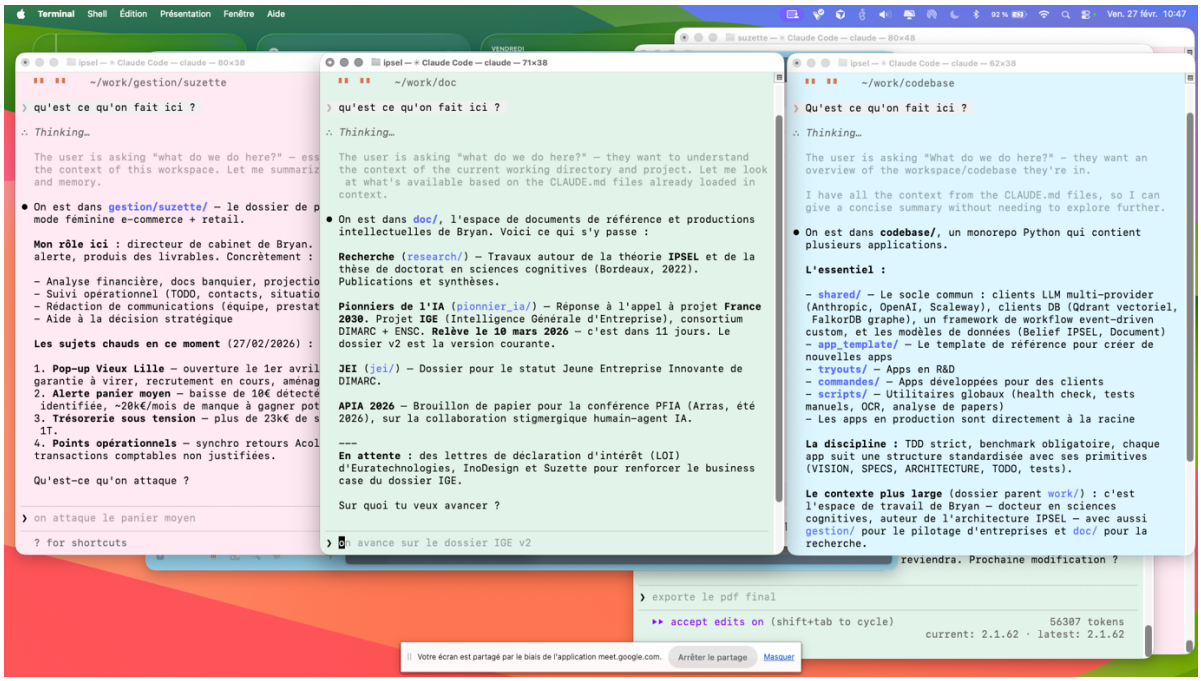

Figure 2 : Exemple de trois agents répondant différemment à la même question

La figure 1 montre un exemple d'écran, avec une session de travail au cours de laquelle plusieurs agents (un agent par fenêtre) travaillent sur les mêmes données. Chacun de ces agents complète le travail des autres.
La figure 2 montre comment une même question (« Qu'est-ce qu'on fait ici ? ») produit des réponses différentes, car les agents évoluent dans des environnements différents puisqu'ils appartiennent à des projets différents. On notera que pour des raisons de clarification, chaque environnement (rédaction d'un rapport administratif, gestion d'une société commerciale, etc.) est coloré différemment. Ici l'agent en rose travaille pour la gestion d'une société commerciale (société réelle dans le domaine du e-commerce), l'agent en vert est une sorte de majordome (tenant à jour la To Do List des activités en cours qui suivent chacune leur calendrier), l'agent en bleu sert à développer le modèle IPSEL (architecture logicielle destinée à produire une logique émergente des données de l'environnement).

5 Discussion

Nous venons de présenter une démarche de développement logiciel dans un environnement dédié à la coordination humains-IA. Cette démarche va même au-delà de la production de code logiciel, car ce sont les mêmes principes qui servent à gérer des besoins commerciaux pour maintenir un site web, répondre aux demandes de clients, traiter des aspects administratifs, tenir un calepin multi-thèmes, etc.
Cette démarche est fortement inspirée par la stigmergie, en ce sens que les agents IA peuvent certes communiquer avec l'humain par des fenêtres de commande mais font avant tout leur travail en se basant sur des ordres inscrits dans des fichiers et produisent eux-mêmes des mises à jour de ces fichiers.
Partant du principe qu'un programme informatique peut activer toute une chaine d'outils logiciels, et que la fenêtre de commande est l'un de ces outils, il devient alors possible, au niveau de chacun des agents, de lire et d'écrire dans des fichiers textes pour prendre connaissance du travail fait, de ce qui est à faire et de modifier en conséquence les fichiers. Ces fichiers constituent bel et bien une trace environnementale pour tous ces agents IA qui les mobilisent. Au sein de chaque répertoire dédié à une application, ce sont des catégories de fichiers Markdown qui servent de traces persistantes entre agents lancés en sessions parallèles. Chacun de ces agents suit un processus en étapes de développement, avec un flux de développement bel et bien contrôlé par l'humain. La méthodologie prévoit un mécanisme explicite de fork vertical, ainsi l'environnement gère la divergence sans pollution de l'application parente. Il s'agit là d'une variante stigmergique : un nouveau territoire avec ses propres traces, mais dans une infrastructure partagée
Ce qui distingue cette approche tient dans l'agent IA. Ce n'est pas un outil que l'on interroge mais il s'agit d'un acteur qui évolue dans un environnement structuré. Son comportement émerge de l'environnement, pas d'un prompt monolithique. La piste suivie cherche à enrichir l'environnement plutôt qu'à structurer la conversation. Cette vision rejoint celle de Yang [24] sur l'importance de l'interface agent-environnement, et se généralise à l'échelle d'un projet entier.
Un autre point de distinction tient dans l'héritage CLAUDE.md en cascade qui permet de factoriser les instructions sans répétition, comme un système de classes en POO. Vasilopoulos [23] propose une approche similaire avec sa « hot-memory constitution », mais sans héritage hiérarchique.
Les skills méta-cognitifs (/introspect, /skill-craft, /finish) apporte également une particularité à l'architecture présentée car ils créent une boucle d'amélioration continue où l'agent affine lui-même ses propres instructions — sous validation humaine. Ce mécanisme n'a pas d'équivalent dans

les systèmes multi-agents existants tels que [20] ou [15] car ceux-ci restent statiques une fois déployés.
Contrairement aux outils spécialisés tels que [16], notre environnement traite le développement logiciel comme un sous-cas d'une collaboration stigmergique plus générale entre agents IA et humain.
Remarquons aussi que l'évaluation des coûts est intégrée au workflow, puisque des benchmarks systématiques sont réalisés, avec mesure de tokens et de latence. Ce point répond directement à la préoccupation soulevée par les études de productivité telles que [2] sur la nécessité de mesurer l'impact réel plutôt que l'impact perçu.
Notre approche actuelle n'est pas issue d'un design ex-nihilo. Elle est le fruit de plusieurs cycles d'observation et de correction. Elle continue encore d'évoluer. Cela rejoint ainsi la dynamique stigmergique : le système s'améliore par traces successives.
Il est clair que les recherches actuelles sur l'usage des LLM pour produire du code informatique, ouvrent tout un champ d'application stimulant et stupéfiant. Il s'agit bien d'un changement de paradigme dans l'histoire du développement logiciel !

6 Références

[1] K. Beck, et al. *Manifesto for Agile Software Development*. 2001. En ligne: http://agilemanifesto.org/

[2] J. Becker, N. Rush, E. Barnes, and D. Rein. *Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity*. arXiv preprint arXiv:2507.09089

[3] F. Bolici, J. Howison, and K. Crowston. Stigmergic coordination in FLOSS development teams: Integrating explicit and implicit mechanisms. Cognitive Systems Research, 38, 14-22, 2016.

[4] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Object Technology Series, 1999.

[5] M. Chen, et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv:2107.03374.

[6] K. Crowston, J. S. Saltz, A. Rezgui, Y. Hegde, and S. You. Socio-technical Affordances for Stigmergic Coordination Implemented in MIDST, a Tool for Data-Science Teams. *Proceedings of the ACM Human-Computer Interaction*, 3(CSCW), 1-25, 2019.

[7] K. Z. Cui, M. Demirer, S. Jaffe, L. Musolff, S. Peng, and T. Salz. The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers. *Management Science*, 2025.

[8] M. Dorigo, M. Birattari, and T. Stützle. *Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique*. IRIDIA Technical Report N° TR/IRIDIA/2006-023, 2006.

[9] M. Elliott. Stigmergic Collaboration: The Evolution of Group Work. *M/C Journal*, 9(2), 2006.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Compagny, 1995.

[11] P.-P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie : Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1), 41-80. 1959.

[12] J. He, C. Treude, and D. Lo. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision and the Road Ahead. *ACM Transactions on Software Engineering and Methodology, 34(5), 1-30*, 2025.

[13] F. Heylighen. Stigmergy as a universal coordination mechanism I: Definition and components. *Cognitive Systems Research*, 38, 4-13, 2016.

[14] F. Heylighen. Stigmergy as a universal coordination mechanism II: Varieties and evolution. *Cognitive Systems Research*, 38, 50-59, 2016.

[15] S. Hong, et al., MetaGPT: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2023. arXiv:2308.00352.

[16] C.E. Jimenez, et al. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* Publié comme acte de conférence ICLR, 2024. arXiv preprint arXiv:2310.06770.

[17] L. Marsh, and C. Onof. Stigmergic epistemology, stigmergic cognition. *Cognitive Systems Research*, 9(1-2), 136-149, 2008.

[18] J. S. Park, et al.. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th annual ACM symposium on user interface software and technology*, 1-22, 2023.

[19] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer. *The Impact of AI on Developer Productivity: Evidence from GitHub Copilot*. arXiv preprint arXiv:2302.06590.

[20] C. Qian, et al. ChatDev Communicative Agents for Software Development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 15174-15186), 2024.

[21] S. K. Sarkar. *AI agents, productivity, and higher-order thinking: Early evidence from software development*. 2026. Available at SSRN 5713646.

[22] G. Theraulaz, and E. Bonabeau. A Brief History of Stigmergy. *Artificial Life*, 5(2), 97-116, 1999.

[23] A. Vasilopoulos. *Codified Context: Infrastructure for AI Agents in a Complex Codebase*. 2026. arXiv preprint arXiv:2602.20478.

[24] J. Yang, et al. SWE-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37, 50528-50652, 2024.

[25] S. Yao, et al. ReAct: Synergizing Reasoning and Acting in Language Models. In *The eleventh international conference on learning representations,* 2023.