Notilis Tech

Industrialiser les workflows : Markdown avant frameworks, skills avant automations

Industrialiser les workflows d’agents de codage progressivement : Markdown, skills, puis automations seulement quand le processus est stable.

2026-05-31 · Skills · Automations · MCP

Résumé

Quand une équipe obtient de bons résultats avec un agent de codage, la tentation suivante est d’industrialiser : réutiliser les bons prompts, créer des skills, automatiser les tâches répétitives et brancher des outils externes.

Le bon ordre reste simple : workflow Markdown, skill, puis automation seulement quand les inputs, outputs, validations et risques sont maîtrisés.

Cet article explique comment passer d’un bon prompt isolé à une procédure réutilisable, sans construire une plateforme trop tôt. Je prends Codex comme cas d’étude, parce que c’est mon outil quotidien, mais la progression vaut largement pour les autres agents de codage.

Sommaire

Contexte de lecture

Audience

Cet article s'adresse aux équipes qui commencent à réutiliser leurs bons workflows avec un agent de codage et se demandent quand passer aux skills ou aux automations.

Ce que cet article couvre

Il propose une trajectoire d'industrialisation : workflow Markdown, skill, automation, MCP et permissions, dans cet ordre de maturité.

Ce que cet article ne couvre pas

Il ne recommande pas de construire une plateforme interne complète avant d'avoir stabilisé les workflows manuels.

Pourquoi commencer par Markdown

Un workflow Markdown est souvent le meilleur premier outil d'industrialisation. Il est versionné, lisible, modifiable par toute l'équipe, facile à relire, compatible avec un agent de codage et suffisamment simple pour évoluer rapidement.

Un workflow Markdown décrit une procédure : quand l'utiliser, quels inputs fournir, quelles étapes suivre, quelles validations exécuter, quelle trace produire et quels risques surveiller.

Exemple :

```
# Workflow: Bugfix avec test de régression

## À utiliser quand

Un bug est reproductible et le comportement attendu est clair.

## Entrées

- Description du bug
- Étapes de reproduction
- Comportement attendu
- Module concerné
- Commandes de validation

## Étapes

1. Inspecter le code et les tests concernés.
2. Reproduire le bug ou expliquer pourquoi la reproduction n'est pas possible.
3. Ajouter ou mettre à jour un test de régression.
4. Implémenter le correctif minimal.
5. Exécuter les tests ciblés.
6. Exécuter une validation plus large si nécessaire.
7. Produire une trace de validation.

## Do not

- Refactorer du code non lié.
- Modifier les contrats d'API publics.
- Ajouter des dépendances sans approbation.
```

Ce format semble simple, mais il transforme une pratique individuelle en procédure d'équipe. Il permet aussi de détecter ce qui n'est pas encore clair. Si l'équipe n'arrive pas à écrire les inputs ou la validation, il est trop tôt pour automatiser.

La maturité d'un workflow

Tous les workflows ne méritent pas d'être industrialisés. Un workflow mérite d'être réutilisé lorsqu'il revient souvent, produit un résultat vérifiable, possède des inputs stables, et peut être relu avec des critères clairs.

Une grille de maturité peut aider :

Niveau	Description	Action recommandée
0	Usage ponctuel, non stabilisé	Garder en prompt ad hoc

Niveau	Description	Action recommandée
1	Bon résultat une fois	Documenter l'exemple
2	Répété plusieurs fois avec succès	Créer un workflow Markdown
3	Inputs, outputs et validation stables	Créer une skill
4	Exécution récurrente et faible ambiguïté	Envisager automation
5	Critique ou sensible	Ajouter gouvernance, approvals et review renforcée

Cette grille évite une erreur fréquente : transformer un prompt qui a marché une fois en automation durable. Un succès ponctuel ne prouve pas que le workflow est stable. Il prouve seulement qu'il a fonctionné dans un contexte.

Skill ou automation : la différence simple

Une skill et une automation ne résolvent pas le même problème.

Une skill est une capacité réutilisable. Elle dit à l'agent comment traiter un type de tâche : quelles règles suivre, quelles étapes appliquer, quels fichiers consulter, quels scripts utiliser éventuellement, et quel format de sortie produire.

Une automation est un déclenchement planifié ou récurrent. Elle dit à l'agent quand relancer une tâche : maintenant, plus tard, toutes les heures, tous les jours, ou tant qu'une condition doit être surveillée.

La différence peut se résumer ainsi :

Objet	Question principale	Exemple
Workflow Markdown	Comment l'équipe veut-elle travailler ?	Procédure de correction de bug avec test de régression
Skill	Comment l'agent doit-il exécuter ce type de tâche ?	Une skill regression-bugfix que l'agent peut appliquer à la demande
Automation	Quand l'agent doit-il relancer cette tâche ?	Une automation qui vérifie chaque matin les PR en attente de review

Une skill ne tourne pas toute seule. Elle spécialise l'agent pour une tâche.

Une automation n'est pas une procédure métier complète. Elle planifie ou relance une instruction dans le temps.

En pratique, une automation peut utiliser une skill, mais elle ne la remplace pas.

La règle pratique est simple :

```
Workflow Markdown = la procédure de l’équipe.
Skill = la procédure donnée à l’agent.
Automation = la procédure relancée dans le temps.
```

Si la procédure n’est pas claire en Markdown, elle ne doit pas devenir une skill. Si la skill n’est pas stable à la demande, elle ne doit pas devenir une automation.

Skills : transformer une procédure stable en capacité réutilisable

Une skill permet de regrouper des instructions, procédures et ressources réutilisables pour un type de tâche. Selon la documentation OpenAI, Codex peut lire des skills depuis plusieurs emplacements : repository, user, admin et system locations. Les repository skills peuvent notamment vivre dans **.agents/skills** selon le niveau de dossier concerné.[1] Les détails varient selon les outils, mais le principe reste le même : sortir les instructions stables du prompt courant.

L’intérêt d’une skill n’est pas de rendre un prompt plus « stylé ». L’intérêt est de sortir les instructions stables du prompt courant. Si une procédure revient souvent, si elle contient des étapes précises, si elle impose une validation, si elle demande un format de sortie, alors elle peut devenir une skill.

Une skill est pertinente pour des workflows comme : bugfix avec test de régression, review sécurité ciblée, migration locale de composant, refactoring sous contraintes, génération de documentation technique, préparation de PR, ou analyse post-incident.

Un skeleton de skill peut être :

```
# Skill: regression-bugfix

## Objectif

Aider à corriger un bug reproductible en ajoutant ou en mettant à jour un test de régression
avant d'implémenter le correctif minimal.

## Quand l'utiliser

Utiliser cette skill quand :
- le bug est reproductible
- le comportement attendu est clair
- le module concerné est connu
- les commandes de validation sont disponibles

## Entrées requises

- Description du bug
- Étapes de reproduction
- Comportement attendu
- Fichiers ou module concernés
- Commandes de validation

## Procédure

1. Inspecter le code concerné et les tests existants.
2. Reproduire le bug ou expliquer pourquoi la reproduction n'est pas possible.
3. Ajouter ou mettre à jour un test de régression.
4. Implémenter le plus petit correctif acceptable.
5. Exécuter la validation ciblée.
6. Exécuter une validation plus large si nécessaire.
7. Produire une trace de validation.

## Format de sortie

- Résumé
- Fichiers modifiés
- Tests ajoutés ou mis à jour
- Validations exécutées
- Limites connues
- Points d'attention pour la review

## Do not

- Refactorer du code non lié
- Modifier les contrats publics sauf demande explicite
- Ajouter des dépendances sans approbation
```

Cette skill n'est pas une automation. Elle reste une procédure réutilisable que l'agent peut appliquer sur demande. C'est précisément ce qui la rend utile avant de passer à des workflows plus autonomes.

Créer une skill concrètement

Une skill commence généralement par un dossier contenant un fichier SKILL.md. Généralement une skill peut contenir des instructions, des ressources et éventuellement des scripts ou des assets.[1] Un format courant ressemble à ceci :

```
.agents/
  skills/
    regression-bugfix/
      SKILL.md
      references/
      scripts/
      assets/
```

Le fichier obligatoire est SKILL.md. Les dossiers references, scripts et assets sont optionnels. Ils deviennent utiles lorsque la skill doit embarquer une documentation plus longue, des commandes répétables ou des modèles de fichiers.

Pour créer une première skill, il n'est pas nécessaire de construire un framework. Il suffit souvent de transformer un workflow Markdown stable en fichier SKILL.md.

Exemple de prompt à donner à l'agent :

```
Crée une repository skill nommée regression-bugfix dans .agents/skills/regression-bugfix/.

Objectif :
Transformer notre workflow Markdown de correction de bug en skill réutilisable.

Contraintes :
- Crée un fichier SKILL.md.
- Garde la procédure courte et actionnable.
- Inclue les entrées requises.
- Inclue les étapes d'exécution.
- Inclue le format de sortie attendu.
- Inclue les interdictions explicites.
- N'ajoute pas de scripts pour l'instant.
- Ne modifie aucun autre fichier.
```

Une fois créée, la skill doit être testée sur un cas réel mais limité. Le bon test n'est pas de vérifier que l'agent « comprend » la skill. Le bon test est de vérifier que deux runs comparables produisent des résultats comparables : même type d'analyse, même discipline de modification, même format de sortie, même niveau de preuve.

Skills locales, repo et distribution

Pour Codex, OpenAI distingue plusieurs emplacements de skills : limitées au repository, limitées à l'utilisateur, définies au niveau admin et skills système.[1] Cette distinction est utile pour décider où placer quoi.

Une skill au niveau du repository sert à standardiser un workflow propre au projet. Une skill locale à un dossier peut servir à un microservice ou un module spécialisé. Une skill au niveau utilisateur sert à des habitudes personnelles. Une skill au niveau admin peut servir à fournir des réglages par défaut dans un environnement partagé.

Le choix doit suivre la portée réelle du workflow. Une procédure propre à un service de paiement ne doit pas être installée globalement. Une procédure générale de bugfix peut être partagée plus largement. Une procédure expérimentale peut rester dans un dossier de travail avant d'être promue.

La règle pratique :

```
Plus une skill est partagée, plus elle doit être stable, testée et maintenue.
```

Automations : planifier une tâche, pas inventer une procédure

L'automation est puissante, mais elle fige les défauts d'un workflow. Si une procédure est ambiguë manuellement, elle sera encore plus dangereuse automatisée. Si la validation est floue, l'automation produira des résultats difficiles à interpréter. Si le scope est mal défini, l'automation peut créer du bruit de manière répétée.

Il est toujours obligatoire de tester une automation manuellement dans un thread régulier avant de la planifier et si possible sur plusieurs jours dans des conditions différentes, afin de vérifier que le prompt est clair, que les outils se comportent comme prévu et que le diff produit est facile à relire.[2] Cette règle devrait être traitée comme un principe d'ingénierie : une automation doit venir après la stabilisation, pas avant.

Les automations peuvent être pertinentes pour des tâches comme : surveiller une PR, vérifier régulièrement un état, relancer une review, produire un triage, ou exécuter une procédure récurrente. Mais elles doivent avoir un prompt durable, une condition d'arrêt, un scope clair, une stratégie de validation et un propriétaire humain.

Créer une automation concrètement

Une automation se crée depuis un thread Codex en décrivant trois choses :

```
1. La tâche à exécuter.
2. Le moment ou la fréquence d'exécution.
3. Le contexte d'exécution attendu.
```

Exemple simple :

```
Crée une automation qui vérifie chaque matin les pull requests ouvertes sur ce repository.

Objectif :
Identifier les PR qui ont reçu de nouveaux commentaires de review ou qui échouent en CI.

Cadence :
Tous les jours ouvrés à 8h30.

Comportement attendu :
- Lister les PR concernées.
- Résumer les nouveaux commentaires.
- Identifier les échecs CI.
- Ne modifier aucun fichier automatiquement.
- Produire une synthèse courte dans le triage.
- Signaler les cas nécessitant une intervention humaine.

Permissions :
Lecture seule si possible.
Aucune correction automatique sans demande explicite.
```

Ce prompt ne décrit pas seulement une tâche. Il décrit une tâche répétée dans le temps. C'est ce qui change le niveau de risque.

Une automation doit donc être testée comme un petit processus d'exploitation, pas comme un simple prompt. Avant de la planifier, il faut l'exécuter manuellement dans un thread classique et vérifier que le résultat est lisible, borné, utile et peu ambigu.

Si le prompt produit trop de bruit en exécution manuelle, il produira le même bruit automatiquement. Simplement plus souvent.

Un bon candidat à l'automation a plusieurs propriétés :

Critère	Question
Fréquence	La tâche revient-elle régulièrement ?
Stabilité	Les inputs et outputs sont-ils prévisibles ?
Validation	Le résultat peut-il être vérifié automatiquement ou semi-automatiquement ?
Risque	Le blast radius est-il limité ?
Ownership	Quel humain relit, ajuste et arrête l'automation ?
Traçabilité	Les runs produisent-ils une trace exploitable ?

Si ces réponses ne sont pas claires, le workflow doit rester manuel ou piloté par une skill.

Permissions et automations : prudence accrue

Une automation n'a pas le même profil de risque qu'une interaction manuelle. Elle peut s'exécuter en arrière-plan, à intervalles réguliers, et potentiellement modifier des fichiers ou consommer des ressources sans supervision immédiate.

La documentation OpenAI indique notamment que les automations utilisent les réglages de sandbox par défaut et rappelle que le full access en background porte un risque plus élevé, car Codex peut changer des fichiers, exécuter des commandes et accéder au réseau sans demander.[3] Ce point doit être intégré à la stratégie d'équipe.

Le niveau de permission doit donc être proportionné :

Workflow	Permission recommandée
Rappel ou suivi sans modification	Read-only si possible
Review périodique	Read-only ou workspace-write limité
Correction automatique de feedback	Workspace-write + review obligatoire
Changement critique	Pas d'automation sans gouvernance explicite
Accès réseau ou outils externes	Autorisation explicite et logging

La règle n'est pas d'interdire les automations. La règle est de ne pas les traiter comme de simples prompts planifiés. Une automation est un processus.

MCP : connecter des outils sans casser le contrôle

MCP peut servir à connecter Codex à des outils externes : documentation, Figma, navigateur, logs, Sentry ou autres serveurs selon les besoins et la configuration. Pour Codex, OpenAI documente la configuration de serveurs MCP via **config.toml** ou la CLI, avec une logique partagée par la CLI et l'extension IDE.[4] D'autres agents disposent de mécanismes équivalents ou voisins, mais l'enjeu reste identique : contrôler ce que l'agent peut voir et faire.

MCP n'est pas un outil magique, c'est simplement un outil externe qui augmente la capacité de l'agent, mais aussi son périmètre opérationnel. Plus l'agent peut voir et faire de choses, plus le harness doit être explicite.

Avant d'ajouter un serveur MCP, il faut se demander :

```
Quel problème de workflow cet outil résout-il ?
Quelles données expose-t-il ?
Quelles actions permet-il ?
Quels logs ou traces produit-il ?
Quel niveau de permission est acceptable ?
Comment le résultat sera-t-il validé ?
```

MCP est donc à traiter comme une extension du harness, pas comme un gadget.

Workflow, skill, automation : comment décider

La décision peut être résumée ainsi :

Situation	Bon format
Tâche unique ou exploratoire	Prompt ad hoc
Tâche répétable mais encore en apprentissage	Workflow Markdown
Procédure stable et réutilisable	Skill
Tâche récurrente, stable, faible ambiguïté	Automation
Travail parallèle complexe	Subagents ou orchestration progressive
Domaine critique	Gouvernance, approvals, review renforcée

Cette table évite la complexité prématurée. Elle donne aussi une trajectoire : documenter, stabiliser, extraire, automatiser.

Les erreurs fréquentes

- La première erreur consiste à créer un framework interne avant d'avoir stabilisé les workflows. L'équipe se retrouve à maintenir une infrastructure qui automatise des pratiques encore floues.
- La deuxième erreur consiste à recopier les mêmes instructions dans tous les prompts. Si une instruction revient souvent, elle doit devenir durable : **AGENTS.md**, workflow, skill ou config selon sa nature.

- La troisième erreur consiste à transformer un prompt réussi en automation. Il faut plusieurs exécutions réussies, avec inputs et validations stables.
- La quatrième erreur consiste à oublier le propriétaire humain. Une automation sans owner devient un bruit récurrent.
- La cinquième erreur consiste à étendre les permissions pour résoudre un problème de cadrage. Si l'agent n'arrive pas à produire un bon résultat, le premier réflexe doit être d'améliorer le workflow, pas de lui donner plus d'accès.

Checklist opérationnelle

Avant de créer une skill, il faut vérifier que :

- [ ] le workflow revient souvent ;
- [ ] les inputs sont connus ;
- [ ] les étapes sont stables ;
- [ ] la validation est explicite ;
- [ ] le format de sortie est utile ;
- [ ] la procédure peut être maintenue par l'équipe.

Avant de créer une automation, il faut vérifier que :

- [ ] le workflow a été testé manuellement ;
- [ ] le diff produit est facile à relire ;
- [ ] un propriétaire humain est identifié ;
- [ ] les permissions sont limitées ;
- [ ] les runs produisent une trace exploitable ;
- [ ] les conditions d'arrêt ou d'escalade sont écrites.

Cette checklist protège l'équipe contre l'industrialisation prématurée.

Conclusion

Industrialiser le travail avec un agent de codage ne signifie pas construire tout de suite une plateforme complexe. La meilleure progression est souvent plus simple : workflow Markdown, puis skill, puis automation lorsque le processus est stable.

Le Markdown rend la procédure visible. La skill rend la procédure réutilisable. L'automation rend la procédure récurrente. Mais chaque niveau augmente les exigences de clarté, de validation et de gouvernance.

Le bon objectif n'est pas d'automatiser plus vite. Le bon objectif est de rendre les bons gestes répétables sans amplifier les mauvais.

Articles liés

- Construire un harness de projet : **AGENTS.md**, commandes de validation et limites de tâche
- Fermer la boucle qualité : tests, review humaine et evidence notes
- Orchestrer à l’échelle d’une équipe : subagents, Symphony et gouvernance technique

Références

1. OpenAI Developers, « Agent Skills — Codex », sections sur le format et les emplacements de skills, consulté le 2026-06-07. https://developers.openai.com/codex/skills

2. OpenAI Developers, « Automations — Codex app », sections sur la création et la gestion des automations, consulté le 2026-06-07. https://developers.openai.com/codex/app/automations

3. OpenAI Developers, « Automations — Codex app », section sur sandbox et full access, consulté le 2026-06-07. https://developers.openai.com/codex/app/automations

4. OpenAI Developers, « Model Context Protocol — Codex », consulté le 2026-06-07. https://developers.openai.com/codex/mcp