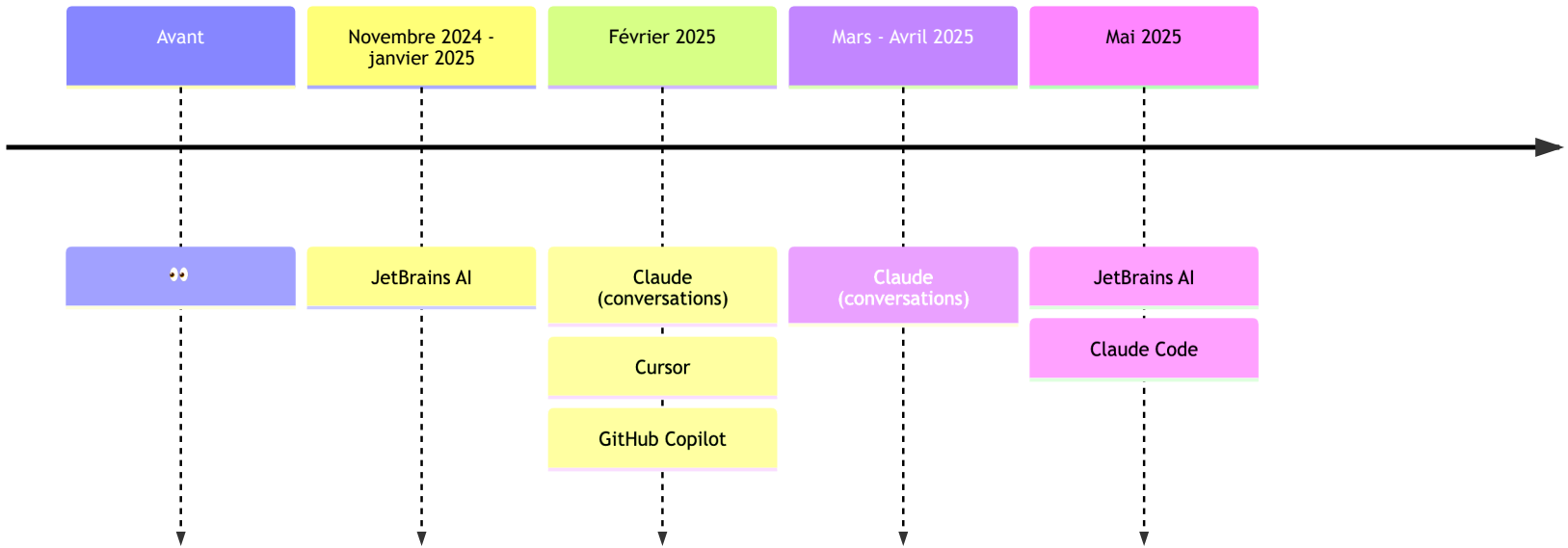
L’IA pour gérer la dette technique

À propos de moi

- Full-stack développeur chez Kumojin
- Artisan logiciel
- 14 ans d’expérience
- Utilisateur avancé de l’IA pour développer depuis 1 an

Mon parcours avec l'IA dans le développement



Le problème : la dette technique s’accumule

Comment ça arrive

- Dégradation de l’écosystème externe
- Accumulation des décisions internes
- Complexité de la croissance

L’IA peut aider… ou aggraver

✅ Potentiel

- Analyser des milliers de lignes
- Détecter des patterns obsolètes
- Proposer des migrations
- Automatiser le refactoring

⚠️ Risque

- Suggestions hors contexte
- Estimations irréalistes
- Changements non testés
- **La dette technique augmente au lieu de diminuer**

Définitions : les outils

LLMs (Large Language Models)

Systèmes d'IA générative avancés qui comprennent et génèrent du langage naturel.

Exemples : GPT-4, Claude, Gemini

Agent de codage

Outil qui lit, modifie ou corrige du code de manière autonome en utilisant le langage naturel comme entrée.

Exemples : Claude Code, Cursor, GitHub Copilot

Comment fonctionne un LLM ?

Les deux étapes clés pour créer un LLM :

Pre-training : les modèles analysent des milliards d'exemples de texte, apprenant à prédire ce qui vient ensuite

💰 Millions de $ / ⏱️ Semaines/mois / 🖥️ Milliers de GPU

Fine-tuning : les modèles sont affinés pour suivre des instructions, être utiles et éviter le contenu nuisible

📊 Moins de données / 👤 Feedback humain / ✅ Comportement souhaité

Que se passe-t-il quand vous envoyez un prompt ?

1. Votre prompt est **tokenisé** (divisé en morceaux)

```
"Analyse cette fonction" → ["Analyse", "cette", "fonction"]
```

2. Le modèle traite ces tokens à travers son réseau de neurones
3. **Prédit le token suivant LE PLUS PROBABLE** basé sur les patterns appris

```
"def calculate(" → next token: probablement "x" ou "self"
```

4. Ajoute ce token à la séquence
5. Répète jusqu'à générer une réponse complète

⚠️ **Point clé** : "Le plus probable" ≠ "Le correct"

C'est de la **prédiction statistique**, pas de la magie.

Capacités clés

✅ **Analyse massive**

- Milliers de lignes quelques secondes

✅ **Reconnaissance de patterns**

- Anti-patterns, code smells

✅ **Apprentissage par l'exemple**

- Donne-lui des exemples, il adapte

✅ **Connexion outils**

- Lire fichiers, exécuter code

Limitations

⚠️ **Knowledge cutoff** (dépend des modèles)

- Ne connaît pas versions récentes des librairies par exemple

⚠️ **Hallucinations**

- Peut inventer des APIs

⚠️ **Context window** (~200K tokens)

- Limite de mémoire

⚠️ **Raisonnement complexe**

- Algorithmes, maths avancées

La philosophie

Deux approches pour utiliser l'IA dans le développement

❌ Vibe Coding

Prompts	"Crée une page de profil utilisateur"
Validation	Pas de review
Compréhension	Copy-paste aveugle
Rapidité	⚡ Rapide au début
Long terme	💥 Dette technique

Mentalité: "Ça marche, c'est bon"

✅ Ingénierie assistée par IA

Prompts	"Refactor UserService pour utiliser DI"
Validation	Review + tests à chaque étape
Compréhension	Comprendre et adapter
Rapidité	🎯 Plus lent au début
Long terme	📈 Maintenable

Mentalité: "L'IA propose, je valide"

Objectifs de cette présentation

Ce que vous allez voir

- ✅ Faire de l'**ingénierie assistée par IA** (pas du vibe coding)
- ✅ Comprendre et **valider chaque étape**
- ✅ Appliquer sur un **vrai code legacy**

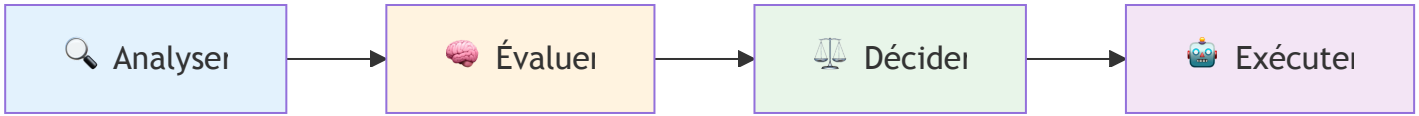
Notre cobaye : extreme-carpaccio

Un kata qui illustre une **situation professionnelle courante** :

- Hériter d'un projet non maintenu depuis plusieurs années.
- Malgré sa qualité initiale, le code présente aujourd'hui une dette technique significative.

Point positif : une suite de tests existante

Roadmap des démonstrations



Le pattern à observer

L'IA propose → L'humain valide → L'IA exécute

Pourquoi ce pattern ?

- Tire parti de la vitesse de l'IA sans sacrifier le contrôle
- Réduit la charge cognitive : l'IA explore, vous décidez
- Scalable : adaptable du code individuel aux systèmes complexes

Analyse du code

Prompt d'analyse de la dette technique

```
Analyze this codebase for technical debt and architectural issues.

Focus on:
- Code smells and anti-patterns
- Outdated dependencies and framework versions
- Architectural problems (coupling, separation of concerns, etc.)
- Security vulnerabilities from deprecated packages
- Performance bottlenecks or inefficient patterns
- Missing error handling
- Inconsistent code patterns across the codebase

For each issue you identify:
1. Specify the exact location (file and line numbers)
2. Explain why it's problematic
3. Rate the severity (critical, high, medium, low)
4. Estimate effort to fix (hours or story points)
5. Suggest a specific remediation approach

After the analysis, generate a prioritized action plan with:
- Quick wins (high value, low effort)
- Critical issues that block modernization
- Long-term refactoring goals
```

Analyse du code - Claude Code en action

Ce que Claude fait

- 🔍 **Glob** : Trouve tous les fichiers
- 📖 **Read** : Lit package.json, server.js, routes, etc.
- 🧠 **Analyse** : Identifie patterns, dépendances, code smells
- 📝 **Génère** : Rapport structuré avec priorités

Pourquoi le mode Plan ?

- ✅ Lecture seule (aucun risque)
- ✅ Idéal pour exploration et review
- ✅ Pas de modifications accidentelles
- ✅ Parfait pour apprendre un nouveau codebase

Analyse du code - Résultats

Ce que l'IA a trouvé

- **Dépendances obsolètes** : React 0.12, jasmine-node
- **Vulnérabilités CVE** : des failles de sécurité connues
- **Patterns anti-code** : callback hell, code dupliqué
- **Structure claire** : Location exacte + sévérité + effort + solution

Ce qu'il faut valider

- **Contexte métier** : "exécution de code arbitraire" → oui, c'est une feature
- **Estimations** : "Migration TypeScript : 8-10 jours" → indicateur, pas vérité
- **Knowledge cutoff** : Ne connaît pas toujours les dernières versions
- **Priorités** : L'IA ne connaît pas vos contraintes projet

Migration majeure : React 19.2

Prompt de migration

```
Plan the React migration from the frontend to the latest version (19.2).

Constraints:
- Do not migrate to TypeScript
- Ensure libraries compatibility with React 19.2
- Propose E2E testing strategy to validate before and after
```

Migration majeure : React 19.2

Approche interactive

- Claude analyse et pose des questions
- Vous guidez la stratégie
- Claude génère un plan détaillé

Comparer les alternatives avant de migrer

L'agent analyse les options et recommande la meilleure approche pour votre contexte.

Exemple de prompt

```
Search body-parser alternatives.

Compare:
- Performance and bundle size
- API compatibility
- Migration effort
- Active maintenance

Recommend the best option for our use case.
```

Remplacement par des APIs natives

L'agent identifie quand une dépendance peut être remplacée par des APIs natives de la plateforme.

Exemple de prompt

```
Analyze our use of lodash in the codebase.

Identify functions that can be replaced with native JavaScript APIs.

For each replacement:
- Show the native equivalent
- Confirm identical behavior
- Highlight any edge cases
- Estimate migration effort

Prioritize high-usage functions first.
```

Migration automatique avec codemod

L'agent écrit le codemod et vous validez le résultat

Exemple de prompt

```
Write a jscodeshift codemod to convert all var declarations to const/let based on reassignment.

Workflow:
1. Generate the codemod code
2. Explain what it does
3. Run with --dry-run on javascripts/*.js
4. Show me the diff of proposed changes
5. STOP and wait for my approval
6. After I approve, provide the final command to run without --dry-run

Do not execute the actual transformation until I explicitly approve.
```

Et dans la vraie vie ?

Utilisation de l'AI pour mettre à jour et/ou remplacer des dépendances sur des projets

- Remplacer React-Script par Vite
- Migrer de React class components à function components
- Migrer React-Router 5x à 7x
- Remplacer moment par dayjs
- Migration majeure de Material UI

Points clés

Cinq démos, une approche

1. **Analyse de code** : l'IA analyse, nous évaluons
2. **Pensée critique** : l'IA a estimé l'effort, nous contextualisons
3. **Décisions de migration** : l'IA a comparé les options, nous avons choisi
4. **Automatisation** : l'IA a écrit le code, nous avons validé

Par où commencer sur vos projets

- **Commencez petit** : analyse de code (faible risque)
- **Développez votre esprit critique** : questionnez tout
- **Élargissez progressivement** : migration → refactoring → automatisation
- **Gardez le contrôle** : l'IA propose, vous décidez

Questions ?

https://github.com/xballoy/presentation-ai-tech-debt