process-ai

Framework de mapeamento de processos para PMEs — com agentes de IA.

Voltado para **Pequenas e Médias Empresas**, o process-ai coloca uma equipe de 7 agentes de IA — Déa, Bento, Miguel, Júlia, Zanoni, Laura e Tiago — para conduzir qualquer pessoa, por perguntas e respostas, até a documentação completa de um processo: cadeia de valor, SIPOC, hierarquia, diagramas BPMN, POPs e relatório final. **O usuário não precisa ser especialista em processos nem em metodologia; os agentes sabem — e conduzem.**

Para quem é

O process-ai é feito para **Pequenas e Médias Empresas** que precisam documentar seus processos mas não têm orçamento para consultoria especializada nem equipe dedicada de BPM.

O dono da empresa, o gerente, o analista — qualquer pessoa pode usar. Basta responder às perguntas que os agentes fazem. Ninguém precisa saber o que é BPMN, SIPOC ou metodologia de mapeamento.

Por que existe

Nas PMEs, os processos críticos vivem na cabeça das pessoas. Quando alguém sai, o conhecimento vai junto. A operação fica frágil, não escala e reprova em auditorias. Sair disso exige:

- **Consultoria** — cara, R$ 15-50 mil por processo
- **Ferramenta BPM** — exige especialista, curva de meses
- **DIY com planilha** — informal, não rastreável, não auditável

O **process-ai** é a quarta via: herda o **rigor do Reversa** (confiança 🟢🟡🔴, rastreabilidade SHA-256, não-destrutivo) e o **modo de conduzir do BMad** — aplicados ao domínio de processos. O resultado: **mapeamento profissional de processos ao alcance de qualquer PME**, sem treinamento, sem consultor, sem licença cara.

A equipe

Agente	Papel	Artefatos
Déa 🧭	Condutora — orquestra a pipeline, abre gates de qualidade, entrega o resumo final	`summary-report`
Bento 🔍	Descoberta — entrevista guiada, SIPOC, cadeia de valor	`discovery-interview`, `sipoc`, `value-chain`
Miguel 🏗️	Mapeamento — hierarquia Macro→Tarefa (5 níveis)	`hierarchy`

Agente	Papel	Artefatos
Júlia 📐	Modelagem — fluxo BPMN 2.0 XML + gargalos	`flow`
Zanoni 📋	Padronização — POPs + diagnóstico consolidado	`pop`
Tiago ✍️	Escritor — consolida todos os artefatos no relatório final de documentação (10 seções)	`process-report`
Laura 🗄️	Arquivista — ingestão de documentos (PDF/DOCX/PPTX/XLSX/CSV/XML) como material de referência	`reference-material`

Instalação

```
# Pré-requisito: Node.js ≥ 24 LTS
cd meu-projeto
npx process-ai
```

`npx process-ai` instala o framework no diretório atual. Em **TTY** (terminal interativo) abre um prompt curto (diretório, method-pack, IDE); em **CI** (não-TTY) ou com flags, instala headless com defaults.

O que o install faz:

- copia as skills (condutora Déa + 6 especialistas: Bento, Miguel, Júlia, Zanoni, Tiago e Laura) para `.claude/skills/`;
- cria `.process-ai/config` (installer-managed, regenerado a cada install) e `.process-ai/config.user` (seus overrides — nunca sobrescritos pelo installer);
- escreve `.process-ai/install-manifest.toml` — o **manifest de instalação** (versão, IDE, pack ativo, e cada arquivo com seu SHA-256), que habilita `update`/`status` e a detecção de arquivos modificados.

A instalação é **idempotente** — pode ser re-rodeada sem efeito colateral. Para instalar como dependência de projeto, `npm install process-ai` executa o mesmo install no `postinstall`.

```
npx process-ai                                    # install (interativo em TTY; headless em CI)
npx process-ai install --target <dir>             # install explícito (headless)
npx process-ai --version                          # versão do framework instalado (-V)
npx process-ai@latest install --status            # estado da instalação (força versão mais recente)
npx process-ai@latest update [--target <dir>]     # atualiza/repara instalação existente
npx process-ai uninstall [--target <dir>] [--purge]  # remove skills + manifest
npx process-ai ingest   --path <arquivo|diretório>   # ingere PDF/DOCX/PPTX/XLSX/CSV/XML como reference-material
```

Flags de install: `--target <dir>` (default: cwd), `--ide <id>` (v1: `claude-code`), `--pack <id>` (default: `bpmn-sipoc`), `--full` (instala tudo, não-interativo), `--status` (apenas relata estado).

Update detecta a instalação prévia via manifest: re-instala se a versão mudou (`stale`) ou se algum arquivo foi editado (`modified`), fazendo **backup** `.bak` dos editados antes de sobrescrever. **Uninstall** remove skills + manifest mas **preserva** `.process-ai/config` e o estado de sessão; `--purge` remove todo o `.process-ai/`.

> 📦 **Atualizar um projeto em andamento:** primeiro verifique se há atualização disponível:
>
> ```
> npx process-ai@latest install --status
> ```
>
> Se mostrar `⚠ Instalado (vX.Y.Z) mas o framework está em vX.Y.Z+1`, rode:
>
> ```
> npx process-ai@latest update
> ```
>
> O sufixo `@latest` é **importante**: sem ele, o npx pode usar uma versão em cache e não detectar que o framework foi atualizado. O update é **não-destrutivo** — preserva `.process-ai/config`, checkpoints, estado de sessão e artefatos já gerados.

Por que instalar? O Claude Code (engine v1) descobre slash-commands pelos arquivos em `.claude/skills/` do projeto. O `npx process-ai` coloca as skills **fisicamente** em `.claude/skills/process-ai/SKILL.md` e faz o scaffolding do config. Em engines futuros (Codex, Cursor, Gemini CLI), cada adapter fará o equivalente — a porta `IdeSetup` (ver `docs/toolkit.md`) isola esse conhecimento.

Após a instalação, o slash-command `/process-ai` está disponível no Claude Code dentro do projeto (aceite o diálogo de workspace trust).

Uso

```
/process-ai
```

A Déa pergunta: "*Qual processo vamos mapear?*"

A partir daí, ela conduz o usuário pela pipeline completa:

```
[Ingestão documental (Laura)] → Gate 0 (escopo) → Bento (descoberta)
→ Gate 1 → Miguel (hierarquia) → Gate 2 → Júlia (BPMN)
→ Gate 3 → Zanoni (POPs) → Gate 4 → Tiago (relatório)
→ Gate 5 → Resumo final + Relatório de Confiança
```

A ingestão documental é opcional e pode ser executada antes ou durante a sessão: `/process-ai-laura` ou `process-ai ingest --path <arquivo|diretório>` converte PDF, DOCX, PPTX, XLSX, CSV e XML em

artefatos `reference-material` que os especialistas usam como evidência.

Cada gate **bloqueia** a próxima etapa até aprovação humana, exibindo a contagem de itens 🟢 (confirmados), 🟡 (inferidos) e 🔴 (gaps).

Confiança 🟢🟡🔴

Toda afirmação de cada agente carrega um **marcador de confiança mecânico**:

- 🟢 **Confirmado** — fonte citada e **verificável** (SHA-256 do artefato + trecho conferido)
- 🟡 **Inferido** — concluído pelo agente a partir de indícios, **sem fonte direta**
- 🔴 **Gap** — desconhecido, não documentado, **precisa de decisão**

O toolkit valida cada 🟢: a fonte citada **resolve** a um artefato commitado no repositório local. Trechos (`excerpt`) são verificados como **substring** do conteúdo canônico da fonte. Fontes fantasmas ou forward-refs são **degradadas a** 🟡 automaticamente.

Nenhum artefato sai sem marcador. Nenhuma inferência é apresentada como fato.

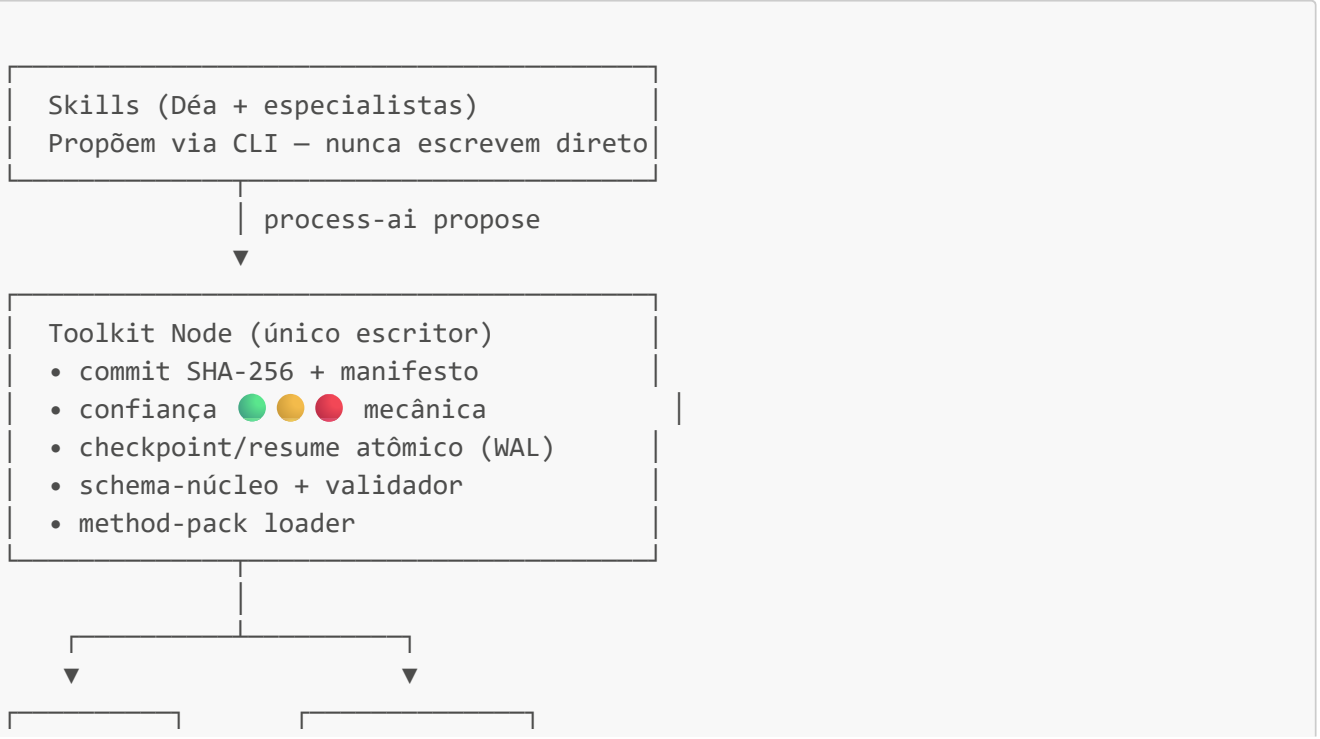
Rastreabilidade bidirecional

Cada afirmação é **navegável à sua fonte nos dois sentidos**:

- **Forward:** afirmação → fonte (artefato + SHA-256 + trecho)
- **Reverse:** artefato-fonte → todas as afirmações que o citam

O relatório de confiança consolidado lista **todos os itens por nível** (claimId, statement, fonte, degradationReason, excerpt-status), com breakdown por tipo de artefato e índice reverso.

Arquitetura



```
  | _process |       | .process-ai/ |
  | _ai_out  |       | (checkpoint, |
  | put/     |       |  manifestos, |
  |(artefatos)|      |  ledger)     |
  └──────────┘       └──────────────┘
```

Invariantes de arquitetura:

- **AD-1:** O toolkit é o **único escritor** — skills só propõem via CLI
- **AD-2:** Method-packs **estendem aditivamente** o schema-núcleo (nunca redefinem)
- **AD-3:** Core **engine-agnostic** — importa só `node:*` + relativos
- **AD-4:** Checkpoint é a **fonte autoritativa** — commit+checkpoint atômicos
- **AD-5:** 🟢 exige **fonte verificável** — ghost/forward-ref → 🟡
- **AD-6:** BPMN on-disk é **XML 2.0 canônico** toolkit-owned
- **AD-7:** Distribuição via **npm** — bootstrap registra no engine

---

Method-packs

O framework é **method-agnostic**: a metodologia (BPMN+SIPOC, Lean, Six Sigma, etc.) é empacotada como **method-pack plugável**.

```toml
# method-packs/bpmn-sipoc/pack.toml
[pack]
name = "bpmn-sipoc"
version = "1.0.0"
artifact_types = ["sipoc", "value-chain", "flow"]
```

Cada pack declara schemas **aditivos** (estendem o schema-núcleo sem redefinir), prompts por especialista e glossário method-specific. O validador **rejeita** packs que tentam alterar pipeline, papéis ou invariantes do core.

Criar um pack: veja `docs/method-packs.md`.

Desenvolvimento

```
git clone https://github.com/sandovalmedeiros/process-ai.git
cd process-ai
npm install
npm test          # 313 testes, 0 falhas
npm run typecheck # tsc --noEmit
```

Estrutura:

```
process-ai/
├── bin/                    # CLI + bootstrap
├── toolkit/
│   ├── src/                # Core engine-agnostic
│   └── adapters/           # Engine adapters (v1: Claude Code)
├── scripts/                # Scripts Python de ingestão
(PDF/DOCX/PPTX/XLSX/CSV/XML)
├── skills/                 # Skills dos agentes
├── method-packs/           # Method-packs plugáveis
│   └── bpmn-sipoc/         # Pack padrão v1
├── tests/                  # Testes determinísticos (313)
└── docs/                   # Documentação
```

Guias:

- Contribuindo
- Criar method-packs
- Criar adapters
- Arquitetura do toolkit

Licença

Feito com 🧭 pelo método **Reversa** e a condução do **BMad**