



ICM: Interpretable Context Methodology — Folder Structure as Agent Architecture

Relatório em português (PT-BR)
Estrutura de pastas como arquitetura de agentes

Um método para workflows sequenciais, transparentes e human-in-the-loop

Pesquisa: Jake Van Clief & David McDermott
Eduba · University of Edinburgh

Agosto de 2026

01 Resumo Executivo

ICM (Interpretable Context Methodology) é uma metodologia de arquitetura de workflows de IA que usa pastas numeradas, arquivos Markdown e scripts locais como mecanismo de coordenação. Em vez de esconder o estado em uma camada de orquestração, o sistema coloca o estado onde pessoas e agentes podem inspecioná-lo: no filesystem.

A proposta vem do paper *Interpretable Context Methodology: Folder Structure as Agent Architecture*, de Jake Van Clief e David McDermott (Eduba, University of Edinburgh), arXiv:2603.16021. O trabalho conecta tradições de Unix, decomposição modular, compilação em múltiplas passagens e interação humano-IA.

O problema resolvido é específico: frameworks de agentes são excelentes para sistemas concorrentes e autônomos, mas podem ser infraestrutura excessiva para processos sequenciais nos quais uma pessoa revisa o resultado a cada etapa. Nesse cenário, trocar um prompt, inserir uma etapa ou recuperar um estágio deveria ser uma edição visível — não uma mudança de código, redeploy ou investigação de logs.

Em uma frase: ICM trata a pasta como máquina de estados e o Markdown como interface de controle.

O modelo operacional é simples: um agente lê o arquivo de entrada da etapa, carrega apenas o contexto necessário, executa um trabalho delimitado e grava um artefato editável. A pessoa inspeciona, ajusta ou aprova; então a próxima etapa é acionada. Scripts locais cuidam do trabalho mecânico — buscar dados, mover arquivos, formatar ou renderizar — sem pedir que um LLM faça o que código determinístico faz melhor.

O ganho central é **interpretabilidade operacional**: qualquer colaborador com um editor de texto consegue localizar a etapa atual, entender seus contratos e alterar o comportamento. O custo é aceitar execução majoritariamente sequencial, recuperação manual e menos automação de ramificações.

Origem e escopo

- Autores: Jake Van Clief e David McDermott; afiliação: Eduba / University of Edinburgh.
- Paper publicado como arXiv:2603.16021v2 (18 mar. 2026), sob licença CC BY 4.0 no HTML do arXiv.
- A comunidade Clief Notes divulga a metodologia e reúne praticantes em um espaço de discussão.
- ICM é um protocolo de organização, não um produto, SDK ou substituto universal para frameworks.

02 Os 5 Princípios de Design

Os princípios são critérios de projeto. Eles reduzem ambiguidade, diminuem contexto desperdiçado e preservam pontos de intervenção humana.

1. One stage, one job — uma etapa, um trabalho

Cada pasta de estágio deve produzir uma transformação reconhecível. Uma etapa pesquisa; outra roteiriza; outra produz. O princípio herda a modularidade Unix e a decomposição de Parnas: componentes pequenos são mais fáceis de substituir, revisar e repetir. Se uma etapa precisa de “e também”, provavelmente há duas responsabilidades.

2. Plain text as the interface — texto simples como interface

Markdown, texto e arquivos comuns são a API do workspace. São versionáveis, pesquisáveis, copiáveis e legíveis por pessoas e modelos. A interface não depende de um banco proprietário ou de uma tela específica.

3. Layered context loading — carregamento em camadas

O agente não recebe o workspace inteiro. Ele começa com roteamento mínimo e abre contratos, referências e artefatos conforme a etapa exige. O alvo prático é contexto focado — aproximadamente 2 mil a 8 mil tokens por etapa — em vez de um prompt monolítico.

4. Every output is an edit surface — toda saída é editável

O resultado de uma etapa não é um “fim”; é uma superfície de revisão. O humano pode corrigir evidências, tom, estrutura ou decisões antes da compilação seguinte. Isso materializa controle humano, transparência e depuração por fonte.

5. Configure the factory, not the product — configure a fábrica, não o produto

Referências, contratos, voz, schemas e templates formam a fábrica: devem ser estáveis e reutilizáveis. Outputs e runs são produtos: mudam por execução. Separar os dois evita que uma correção em um deliverable seja confundida com uma alteração permanente no processo.

Regra prática

Se o comportamento deve mudar em cada execução, edite o produto. Se deve mudar em todas as execuções futuras, edite a fábrica. Se só o estágio atual precisa saber algo, coloque a informação no contrato desse estágio.

03 A Hierarquia de 5 Camadas

A hierarquia funciona como um sistema de orientação progressiva. Cada nível responde a uma pergunta e aponta para o próximo sem carregar detalhes desnecessários.

Camada	Arquivo / Área	Pergunta	Papel	Orçamento típico
L0	CLAUDE.md / AGENTS.md	Onde estou?	Roteamento global, identidade e limites	300–800 tokens
L1	CONTEXT.md da raiz	Para onde vou?	Mapa de tarefas e caminhos	200–500 tokens
L2	CONTEXT.md do estágio	O que faço?	Contrato de controle: entradas, ação, saída, critério de pronto	200–500 tokens
L3	references/, _shared/	Quais regras valem?	Fábrica: voz, convenções, glossário, políticas e schemas estáveis	500–2 mil tokens
L4	output/, artefatos de run	Com o que trabalho?	Produto por execução: fontes, rascunhos, decisões, entregáveis	Variável

Fluxo de leitura

```
L0 CLAUDE.md / AGENTS.md → orientação
■■ L1 CONTEXT.md → seleção do caminho
■■ L2 stage/CONTEXT.md → contrato da etapa
■■ L3 references/ → regras e material estável
■■ L4 output/ → artefatos da execução
```

A camada 0 deve ser pequena e estável. Ela aponta, não explica tudo. A camada 2 é o ponto de controle: quando um estágio tem entradas, ação, formato de saída e critérios explícitos, a execução fica auditável. L3 é “fábrica”; L4 é “produto”. Misturar os dois causa deriva: referências antigas parecem instruções atuais e rascunhos contaminam o processo.

04 As 5 Formas (Forms)

As formas são esqueletos recorrentes. Escolher a forma certa é mais importante do que adicionar automação: a estrutura deve refletir como o trabalho realmente muda.

Pipeline — linha de produção

Use quando a mesma sequência é repetida e cada execução gera um deliverable. A numeração carrega a ordem.

```
project/
  ■■■ CLAUDE.md
  ■■■ CONTEXT.md
  ■■■ references/
  ■■■ stages/
    ■ ■■■ 01_research/CONTEXT.md + output/
    ■ ■■■ 02_script/CONTEXT.md + output/
    ■ ■■■ 03_production/CONTEXT.md + output/
  ■■■ runs/
```

Umbrella — guarda-chuva

Use quando pipelines distintos compartilham marca, voz e referências.

```
brand/
  ■■■ CLAUDE.md
  ■■■ CONTEXT.md
  ■■■ 01_pillars/CONTEXT.md
  ■■■ 02_brand_voice/CONTEXT.md
  ■■■ 03_video_production/
  ■■■ 04_course_production/
  ■■■ _shared/ (voice.md, glossary.md, refs/)
```

Record Library — biblioteca de registros

Use quando a unidade principal é um registro acumulativo, como cliente, pessoa ou sessão.

```
workspace/
  ■■■ CLAUDE.md
  ■■■ _templates/record-template/
  ■■■ records/acme-corp/CONTEXT.md + history/
  ■■■ records/client-b/...
  ■■■ _index/log.md
```

Knowledge Bundle — pacote de conhecimento

Use quando o produto é o próprio conhecimento navegável: corpus bruto, pipeline de extração e bundle compilado.

```
knowledge/
  ■■■ CLAUDE.md
  ■■■ corpus/ (raw sources)
  ■■■ extraction/ (factory pipeline)
    ■ ■■■ 01_ingest/ ■■■ 02_normalize/
  ■■■ bundle/ (product: index, topics, links, Q&A)
```

Context Map — mapa de contexto

Use quando a organização é um grafo de equipes, processos, dados e padrões.

```
org/
  ■■■ CLAUDE.md
  ■■■ teams/marketing/Marketing.md
  ■■■ teams/engineering/Engineering.md
  ■■■ processes/
  ■■■ data/
  ■■■ patterns/
```

05 Tabela Comparativa: ICM vs. Frameworks

A comparação não diz que um lado “vence”. Ela mostra diferentes superfícies de controle: código e runtime, de um lado; filesystem e revisão humana, do outro.

Dimensão	Framework (LangChain / CrewAI / AutoGen)	ICM
Mudar ordem	Editar orquestração, testar e redeploy	Renomear ou reordenar pastas
Modificar prompt	Editar configuração ou código do agente	Editar arquivo Markdown
Adicionar/remover estágio	Criar classe e atualizar orquestrador	Adicionar/deletar pasta com contrato
Inspecionar estado	Logs, tracing ou dashboard	Abrir a pasta e ler arquivos
Passar para outra pessoa	Documentar ambiente e dependências	Copiar o workspace
Quem altera	Tipicamente desenvolvedor	Qualquer pessoa com editor
Recuperação	Retry, fallback e exceções embutidos	Reexecutar manualmente o estágio falho
Ramificação condicional	Roteamento programático	Decisão humana entre pastas/etapas
Concorrência	Coordenação paralela nativa	Sequencial por desenho
Integração externa	APIs, auth e ferramentas no runtime	Scripts locais ou conexões MCP

A tabela expõe o trade-off. ICM ganha em legibilidade, editabilidade e handoff; frameworks ganham em concorrência, recuperação automática, branching e integrações complexas. Uma arquitetura híbrida é possível: ICM como superfície de controle e scripts/MCP como executores mecânicos.

06 Exemplos Reais

Os exemplos abaixo ilustram como a mesma lógica pode servir produção de mídia, educação e pesquisa aplicada.

Script-to-Animation Pipeline

Três estágios: **pesquisa** → **script** → **produção** via Remotion. A pesquisa reúne fontes e claims; o script transforma evidência em narrativa editável; a produção converte o roteiro em cenas, assets e vídeo renderizado. Cada pasta registra o handoff e permite que o criador corrija a história antes de pagar o custo visual.

```
animation/  
■■■ 01_research/output/sources.md  
■■■ 02_script/output/script.md  
■■■ 03_production/output/ (Remotion project, render.mp4)
```

Course Deck Production

Cinco estágios: **extração de conteúdo** → **planejamento estrutural** → **rascunho de slides** → **design visual** → **montagem final**. A separação torna visível o momento em que fatos viram estrutura e o momento em que estrutura vira composição visual.

```
course-deck/  
■■■ 01_extract/CONTEXT.md + output/notes.md  
■■■ 02_structure/output/outline.md  
■■■ 03_draft/output/slides-draft.md  
■■■ 04_visual/output/design-spec.md  
■■■ 05_assembly/output/deck.pptx
```

Mini-Course Workspace — Jes Fink-Jensen

Adaptação importante: execução sem Python, com LM Studio Bionic e LLM local (Qwen). O pipeline tem quatro estágios: **01_scope** → **02_research** → **03_outline** → **04_draft**. A pasta **shared/** guarda input.md, voice.md e audience.md; **pipeline/** guarda os contratos e saídas. A estrutura prova que ICM é agnóstico ao provedor e pode operar local-first.

07 Onde ICM Funciona — e onde NÃO funciona

ICM é deliberadamente uma solução com domínio de validade. A pergunta correta não é “posso colocar agentes aqui?”, mas “o filesystem é uma boa interface para este fluxo?”

Funciona bem quando:

- O processo é sequencial, repetível e tem revisão humana entre etapas.
- Entregáveis são conteúdo, análise, treinamento, pesquisa ou políticas — objetos naturalmente editáveis.
- O usuário é uma pessoa ou uma equipe pequena e valoriza local-first, transparência e handoff simples.
- O custo de escolher a próxima etapa manualmente é menor que o custo de construir e manter um orquestrador.
- A rastreabilidade importa: decisões, fontes e versões precisam continuar visíveis.

Não funciona bem quando:

- Há colaboração multiagente em tempo real: isso pede message-passing e coordenação de presença.
- A carga exige alta concorrência, filas, isolamento de estado, locks e escalabilidade horizontal.
- Branching deve acontecer automaticamente no meio do pipeline, baseado em sinais e políticas complexas.
- Retries, timeouts, circuit breakers e garantias transacionais são requisitos centrais.
- Integrações externas, autenticação e compliance precisam de um runtime centralizado e observável.

O padrão de intervenção

Experiências da comunidade relatam um padrão em U: edições pesadas no primeiro estágio, refinamentos leves no meio e intervenção forte na etapa final. Isso faz sentido: no início se corrige escopo e evidência; no fim se corrige a decisão pública/entregável. O meio pode ser mais automatizado sem perder controle.

Sinal de alerta

08 A skill icm-architect

A skill **icm-architect** transforma uma conversa sobre um processo em um workspace ICM — ou reorganiza um diretório existente. Ela empacota os invariants, as formas, templates copiáveis e um teste de validação.

Os 10 invariants

- Uma pasta, um trabalho.
- Arquivo de entrada pequeno e estável (CLAUDE.md/AGENTS.md, cerca de 60 linhas).
- Numeração codifica ordem (01_, 02_, ...).
- Todo contrato de pasta é explícito (CONTEXT.md).
- Fábrica e produto separados.
- Toda saída é uma superfície de edição.
- Carregar apenas o necessário para a etapa (2 mil–8 mil tokens).
- Texto simples, linkável e consultável.
- Filesystem é a máquina de estados.
- Instanciar copiando templates.

Build mode

Extraí da conversa os estágios, gates humanos, entradas, saídas e material estável; escolhe uma das cinco formas; cria o esqueleto mínimo; escreve os contratos; e valida com o walk test. O resultado deve ser pequeno o suficiente para ser entendido sem documentação paralela.

Restructure mode

Faz inventário do diretório, classifica cada arquivo como catálogo, contrato, fábrica, produto ou morto, propõe um mapa de migração para aprovação, migra e valida. A regra é não destruir informação silenciosamente: preservar ou marcar o que não couber.

Walk test

*Com memória vazia, o agente deve conseguir responder: **onde estou?** e **para onde vou?** lendo o arquivo de entrada e apenas duas leituras adicionais. Depois deve saber qual ação executar e onde gravar o status. Se precisa varrer tudo, o roteamento falhou.*

09 Recomendações para OpenCode

OpenCode pode adotar ICM sem alterar o método. A adaptação principal é de convenção de descoberta e de linguagem operacional.

OpenCode pode adotar ICM sem alterar o método. A adaptação principal é de convenção de descoberta e de linguagem operacional.

Elemento	Adaptação recomendada
Arquivo L0	Use AGENTS.md no lugar de CLAUDE.md; mantenha-o pequeno, estável e orientado a roteamento.
Referências a Claude	Remova menções específicas a Claude Code, Claude Opus ou Agent Teams. Descreva papéis e contratos de forma agnóstica.
Backend	Registre no contexto apenas capacidades necessárias. OpenCode pode usar qualquer backend LLM; não acople a arquitetura ao provedor.
Invariants	Mantenha os 10 invariants intactos: são propriedades de filesystem e interface, não de modelo.
Comandos/scripts	Use scripts locais, shell e MCP para ações mecânicas. Documente entradas, saídas e falhas no CONTEXT.md.
Handoff	Inclua STATUS.md ou um artefato equivalente por run quando o workspace precisar sobreviver a sessões.

Implementação mínima

```
workspace/
■■■■ AGENTS.md           # L0: orientação para OpenCode
■■■■ CONTEXT.md          # L1: mapa e próximo passo
■■■■ shared/             # L3: voz, audiência, regras
■■■■ 01_stage/CONTEXT.md # L2: contrato
■   ■■■■ output/         # L4: produto editável
■■■■ 02_stage/CONTEXT.md
```

Não replique a semântica de um framework dentro de AGENTS.md. O arquivo deve apontar para contratos; os contratos devem apontar para referências; outputs devem registrar o trabalho. O backend pode mudar sem reestruturar o workspace.

10 Referências e Próximos Passos

Para aprofundar, comece pelo paper, depois inspecione um template e finalmente faça um pequeno walk test em um processo real. A melhor validação não é uma demo: é conseguir retomar o trabalho após uma pausa sem depender da memória da conversa.

Referências primárias

- Van Clief, Jake; McDermott, David. *Interpretable Context Methodology: Folder Structure as Agent Architecture*. arXiv:2603.16021v2. <https://arxiv.org/abs/2603.16021>
- Texto HTML do paper: <https://arxiv.org/html/2603.16021v2>
- Repositório da skill icm-architect: <https://github.com/RinDig/icm-architect>
- Template model-agnostic: <https://github.com/ktnCodes/icm-template>
- Comunidade Clief Notes: <https://www.skool.com/cliefnotes>
- Discussão do paper: <https://www.skool.com/cliefnotes/icm-research-paper>

Linhas intelectuais

- McIlroy / tradição Unix: programas pequenos, composição e texto como interface.
- Parnas: decomposição modular e ocultação de decisões.
- Kernighan & Pike: poder nas relações entre programas.
- Make e compiladores multi-pass: arquivos como artefatos e coordenação.
- Horvitz, Shneiderman e interação humano-IA: controle, visibilidade e edição direta.

Checklist de adoção

- Comece com um pipeline pequeno e três estágios.
- Escreva AGENTS.md e CONTEXT.md antes de prompts longos.
- Defina uma saída editável e um critério de pronto por estágio.
- Separe shared/references (fábrica) de output (produto).
- Execute o walk test com um agente sem contexto da conversa.
- Só introduza um framework quando concorrência, branching automático ou confiabilidade operacional justificarem seu custo.

Conclusão

ICM não elimina engenharia; ele escolhe uma superfície mais simples para um conjunto específico de problemas. Ao transformar pastas, contratos e artefatos em arquitetura visível, ele permite que a organização do trabalho — e não uma camada de infraestrutura — seja o centro do sistema.