Design Patterns in MCP: Thoughtboxes

a new paradigm for agentic reasoning tools

10 min read · 5 days ago

glassBead Follow

. . .

"Every thought is creative."

- Kendrick Lamar, "Mr. Morale"

. . .

note: this post is part of the rollout for Thoughtbox, my newest reasoning MCP under the Kastalien Research umbrella, now available in beta via Smithery.

introduction

back in May, i wrote a blog post about a category of MCP servers called "model enhancement servers." i defined this category as "technology, natively designed for use by an LLM, intended to extend the user's (that is, the model's) abilities in a variety of circumstances."

today, i'm focusing on a specific subcategory of these that i call "***thoughtboxes***".

you have likely already used MCP thoughtboxes in your own work. Sequential Thinking and Clear Thought are two popular MCP servers that adhere to the thoughtbox pattern. in this installment of the Design Patterns in MCP series, we'll take a look at

- what thoughtboxes are
- what thoughtboxes are ***not***
- how to design your own thoughtbox server + examples

finally, we'll review some open opportunities for you to take the Thoughtbox Pattern and extend it or improve on it.

let's dive in.

. . .

artistic visualization of thoughtboxes, Nano Banana

. . .

defining thoughtboxes

thoughtboxes are members of a category of MCP server that provides structured workspaces for LLM cognition. they share four core characteristics:

1. **Architectural Structure**: thoughtboxes provide organized patterns for thought (numbered sequences, cells, hierarchies) rather than blank space
2. **Cognitive Affordances**: they feature design choices that make specific reasoning modes more natural (branching for hypothesis exploration, execution for verification, revision for iterative refinement)
3. **Process Orientation**: thoughtboxes scaffold the *process* of cognition, not just outputs. The value is in how it shapes thinking, not just what it produces
4. **Addressability**: thoughtboxes provide stable references enabling non-linear navigation, revision, and composition (thought numbers, cell IDs, etc.)

an MCP server created with these design paradigms turns an LLM's linear context (token stream) into navigable cognitive space while the LLM interacts with the server's tools.

a quick visualization

to help us visualize what this looks like from an agent's perspective, let's imagine two approaches to outlining an essay: thinking out loud to a friend, and using a storyboarding app.

in the first approach, we talk it out off the top of our heads to a friend, who transcribes what we're saying as we go and provides some order to our words. if we say "my first point is X," describe X for a few minutes, then say "wait, no, X should be my second point. my first point should actually be Y, let me reorganize…", our friend will probably be frustrated with us, because the structure of what we're saying exists only as verbal corrections layered on top of each other. in the second approach, we use an imaginary storyboarding app. at startup, this app lets us guess how many sections of the essay we'll have and creates "slots" for each section. like before, we expound on X for our first point for a few minutes, then realize that X

should be our second point and that our first point should actually be Y. using the app, we simply swap the two slots' order in the GUI: no frustration necessary.

> *thoughtboxes make it* **cheap** *for an agent to iterate on its own reasoning*

it's *costly* for our friend to parse through our verbal corrections and try to connect what we're saying to what we said before, because our modality isn't *navigable:* they can't travel back through time to a checkpoint in our verbal spitballing session to "correct the record," because the record is being created after the fact.

in contrast, it's *cheap* for the app to implement our corrections, because our modality *is* navigable: our points have ***addresses*** as soon as they're expressed, so we can just swap the addresses of two points, or go in and change the information held at a certain address.

in this way, we can say that thoughtboxes make it *cheap* for an agent to iterate on its own reasoning.

what thoughtboxes aren't

before we continue to concrete examples, it's important to note a few things that thoughtboxes are **not.**

- Service Connector (Gmail MCP): Connects to external services, doesn't scaffold cognition
- Memory/RAG (Memory MCP): Stores/retrieves information, doesn't scaffold active reasoning
- Simple Template (TODO list): Provides structure, but lacks rich cognitive affordances

. . .

designing thoughtboxes

sequential thinking goes both ways

the actual coding work to build thoughtboxes can be simple. here's an example to illustrate: did you know agents can use Sequential Thinking backwards? i'll prove it.

first, configure your favorite MCP client application (Claude Desktop/Code, Roo Code, Goose, etc.) with the Sequential Thinking MCP server, then give the agent this

prompt:

```
Use the Sequential Thinking MCP server to think about whatever you like,
but instead of starting at Thought 1 of N total thoughts,
start at Thought N and work backwards.
```

your agent will reason backwards as instructed.

millions of people have likely used Sequential Thinking at this point, but a quick Perplexity search didn't turn up evidence that anyone's documented using it that way. to reiterate: ***our agents have been able to use vanilla Sequential Thinking to reason by inversion this whole time!***

why didn't we know this? my guess is that it's because unless otherwise instructed, if an agent is told to specify a number N of total thoughts and reason through a problem in that many thoughts, the most salient way to do that is to progress 1 -> N. Once that's been done, the agent has completed its task and will either pause or move on.

the point is: providing process-oriented tools to agents can be as simple as prompting the agent to follow a certain process in a tool description or a slash command.

a thoughtbox's "cognitive affordances" can be quite simple from a coding perspective.

design patterns in thoughtboxes

for a couple of examples of how to design thoughtboxes, let's look at the clear_thought tool declaration on Thoughtbox. this server is a successor to Clear Thought 1.5, and the pilot MCP server from Kastalien Research* (now in public beta):

```
const CLEAR_THOUGHT_TOOL: Tool = {
  name: "clear_thought",
  description: `A detailed tool for dynamic and reflective problem-solving thro
This tool helps analyze problems through a flexible thinking process that can a
Each thought can build on, question, or revise previous insights as understandi
Supports forward thinking (1→N), backward thinking (N→1), or mixed approaches.
```

```
When to use this tool:
- Breaking down complex problems into steps
- Planning and design with room for revision
- Analysis that might need course correction
- Problems where the full scope might not be clear initially
- Problems that require a multi-step solution
- Tasks that need to maintain context over multiple steps
- Situations where irrelevant information needs to be filtered out

Thinking Approaches:

**Forward Thinking (Traditional Chain of Thought)**: Start at thought 1, work s
- Use when: Exploring unknowns, brainstorming, open-ended analysis, discovery
- Pattern: thoughtNumber 1 → 2 → 3 → ... → N
- Example: "How can we improve user engagement?" Start with current state, expl

**Backward Thinking (Goal-Driven Reasoning)**: Start at thought N (desired end
- Use when: Designing systems, planning projects, solving well-defined problems
- Pattern: thoughtNumber N → N-1 → N-2 → ... → 1
- Example: "Design a caching strategy for 10k req/s" Start with success criteri
- Tip: Begin with the desired outcome, then repeatedly ask "what must be true i

**Mixed/Branched Thinking**: Combine approaches or explore alternative paths us
- Use when: Complex problems requiring multiple perspectives or hypothesis test
- Pattern: Use isRevision, branchFromThought, and branchId to create alternativ

Patterns Cookbook:
The patterns cookbook guide is automatically provided as an embedded resource a
You can also request it at any time using the includeGuide parameter.
The cookbook contains core reasoning patterns with examples and usage guidance.

Key features:
- You can adjust total_thoughts up or down as you progress
- You can question or revise previous thoughts
- You can add more thoughts even after reaching what seemed like the end
- You can express uncertainty and explore alternative approaches
- Not every thought needs to build linearly - you can branch or backtrack
- Generates a solution hypothesis
- Verifies the hypothesis based on the Chain of Thought steps
- Repeats the process until satisfied
- Provides a correct answer

Parameters explained:
- thought: Your current thinking step, which can include:
* Regular analytical steps
* Revisions of previous thoughts
* Questions about previous decisions
* Realizations about needing more analysis
* Changes in approach
* Hypothesis generation
* Hypothesis verification
- next_thought_needed: True if you need more thinking, even if at what seemed l
```

```
- thought_number: Current number in sequence (can go beyond initial total if ne
- total_thoughts: Current estimate of thoughts needed (can be adjusted up/down)
- is_revision: A boolean indicating if this thought revises previous thinking
- revises_thought: If is_revision is true, which thought number is being recons
- branch_from_thought: If branching, which thought number is the branching poin
- branch_id: Identifier for the current branch (if any)
- needs_more_thoughts: If reaching end but realizing more thoughts needed

You should:
1. Start with an initial estimate of needed thoughts, but be ready to adjust
2. Feel free to question or revise previous thoughts
3. Don't hesitate to add more thoughts if needed, even at the "end"
4. Express uncertainty when present
5. Mark thoughts that revise previous thinking or branch into new paths
6. Ignore information that is irrelevant to the current step
7. Generate a solution hypothesis when appropriate
8. Verify the hypothesis based on the Chain of Thought steps
9. Repeat the process until satisfied with the solution
10. Provide a single, ideally correct answer as the final output
11. Only set next_thought_needed to false when truly done and a satisfactory an
  inputSchema: {
    type: "object",
    properties: {
      thought: {
        type: "string",
        description: "Your current thinking step",
      },
      nextThoughtNeeded: {
        type: "boolean",
        description: "Whether another thought step is needed",
      },
      thoughtNumber: {
        type: "integer",
        description:
          "Current thought number (can be 1→N for forward thinking, or N→1 for
        minimum: 1,
      },
      totalThoughts: {
        type: "integer",
        description:
          "Estimated total thoughts needed (for backward thinking, start with t
        minimum: 1,
      },
      isRevision: {
        type: "boolean",
        description: "Whether this revises previous thinking",
      },
      revisesThought: {
        type: "integer",
        description: "Which thought is being reconsidered",
        minimum: 1,
      },
      branchFromThought: {
```

```
          type: "integer",
          description: "Branching point thought number",
          minimum: 1,
        },
        branchId: {
          type: "string",
          description: "Branch identifier",
        },
        needsMoreThoughts: {
          type: "boolean",
          description: "If more thoughts are needed",
        },
        includeGuide: {
          type: "boolean",
          description:
            "Request the patterns cookbook guide as embedded resource (also provi
        },
      },
      required: [
        "thought",
        "nextThoughtNeeded",
        "thoughtNumber",
        "totalThoughts",
      ],
    },
    annotations: {
      audience: ["assistant"],
      priority: 0.85,
      readOnlyHint: false,
      destructiveHint: false,
      idempotentHint: false,
    },
  };
```

examining the code above, we notice two significant features. first, Thoughtbox has provided instructions in the clear_thought tool declaration on how the agent can use the tool (forward CoT, backward chaining, and branching, which enables Tree of Thought-style and parallel reasoning). note that nothing here *enforces* a particular usage: we simply inform the agent of what it *could* do.

second, we have a reference to a "patterns cookbook" guide that is provided both as a standalone resource, and as an embedded resource in the first tool call and tool calls where thoughtNumber = totalThoughts. this is a way to provide the model with a "menu" of options for building out long reasoning processes with the available capabilities.

this pattern is a form of progressive disclosure, just as the toolhost pattern is. the agent can see the tool description (*cheaper*) before it calls the tool. it is only after the agent actually calls clear_thought that the cookbook (*costlier*) is put into its context window.

advice on building thoughtboxes

a few words of advice here.

first, an important derivative principle of thoughtbox MCP's is that despite their obvious utility as agentic reasoners, **the servers themselves do not perform any reasoning!** thoughtboxes are infrastructure, not intelligence: when an agent uses a thoughtbox, all of the intelligence in that process lives on the client side. the model does not collaborate with the server to reason: it simply uses the server as a tool to enhance its own reasoning.

this is important to keep in mind throughout the building process, because our coding agents do not have any explicit examples of this pattern in their training data. if we fork the Thoughtbox repo and then ask Roo Code or Gemini CLI or Claude Code to make improvements or alterations to it, they will almost certainly write code that evaluates model input on the server, and this may not be obvious to us at the time.

a good rule of thumb: ***don't accept any diffs with regex in them.*** by default, the only evaluation thoughtboxes do on an MCP client's input is schema validation (note: thoughtboxes may offer notebook functionality for literate reasoning).

finally, though it's certainly possible to implement thoughtbox functionality on a wrapper MCP, by default, thoughtboxes do not connect to any external services. as a subset of model enhancement servers, they are still standalone general-use technologies designed for agentic use.

. . .

conclusion/opportunities for extension

this has been a high-level overview of the "thoughtbox" category of MCP servers. there is a lot more to be said about these servers, but in the interest of keeping this post readable, we've stayed pretty close to the fundamentals here.

there are many avenues you can take to extend this pattern. a few examples are:

- structures that enable open-ended exploration of a search space, as found in Jeff Clune's work (MAP-Elites, SOAP, etc.)
- patterns for statefully tracking reasoning chains across sessions, and enabling the MCP client application to review past reasoning and improve on it session to session
- timeline-based thought structures for temporal/causal reasoning
- implement thoughtbox functionality on a search-based MCP server like those from Firecrawl or Perplexity

thanks for reading. feedback is welcome in the comments, or via email at glassBead@kastalienresearch.ai.

Model Context Protocol | Mcp Server | Llm Reasoning | Agentic Ai | Ai Tools

More from glassBead

glassBead

MCP 101: Episode 1, Model Enhancement Servers

hello, AI development community. i've been teasing a large set of resources on Model Context Protocol for months now, and what i've come...

May 1 •••

In The Generator by Jim the AI Whisperer

We've been wrong about how AI thinks this whole time—and my "Chain of Babble" theory proves it

How I dramatically improved AI accuracy on a complex task by replacing Chain of Thought reasoning with "Blah Blah Blah"

Oct 13 •••

In The Generator by Thomas Smith

The Right Way to Use AI At Work

An experiment from Stanford shows why we're all doing it wrong

Oct 28 •••

glassBead

Design Patterns in MCP: Toolhost Pattern

...

Aug 9 •••

See all from glassBead

Recommended from Medium

Daniel Avila

Claude Code Learning Path: a practical guide to getting started

After spending months diving deep into Claude Code, I wanted to share the learning path that worked for me. This isn't from official...

Oct 30 •••

Guangya Liu

MCP Sampling: Architecture, Workflow, and Practical Guide

Not a Medium member? Visit https://gyliu513.github.io/

Aug 21

Amos Isaila

Building Your Own MCP Server: Angular Best Practices Refactoring Tool —Part 2

In Part 1, we explored the theory behind Model Context Protocol, now it’s time to get our hands dirty.

4d ago

In UX Planet by Nick Babich

Claude For Code: How to use Claude to Streamline Product Design Process

Anthropic Claude is a primary competitor of OpenAI's ChatGPT. Just like ChatGPT this is versatile tool that can be use used in many...

Oct 16 •••

In Storybook by Michael Shilman

Storybook 10

ESM-only, 29% lighter, module automocking, and more

4d ago •••

OutSight AI

Beyond the Demo: How Far Can Coding Agents Really Take Figma-to-Code?

With the emergence of coding agents like Claude Code, Codex, and MCP servers, there's a lot of noise about design-to-code automation. With...

Aug 5 •••

See more recommendations