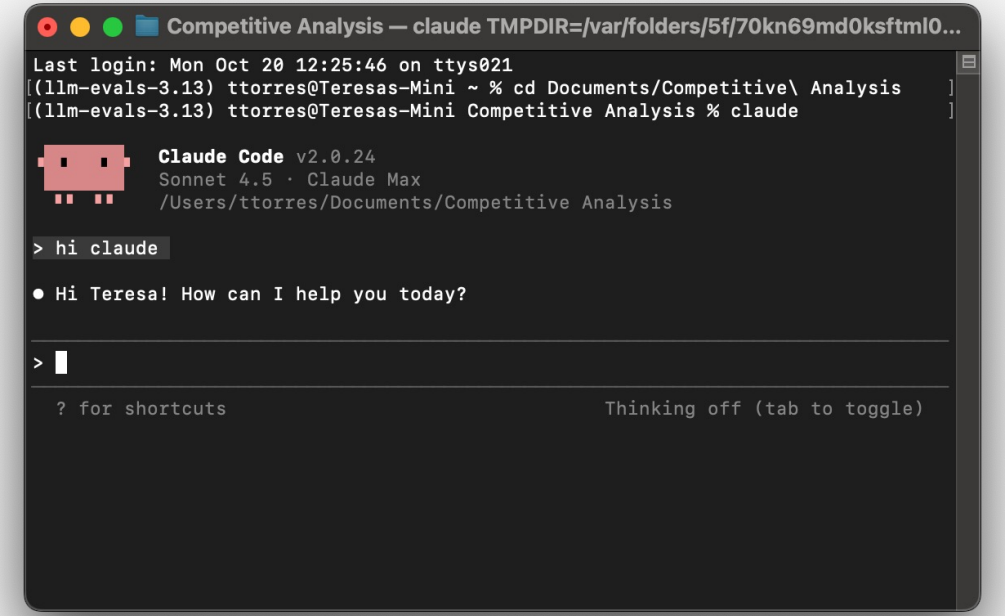
How Claude Code is Built

The Power of **Simplicity** in AI-Assisted Development

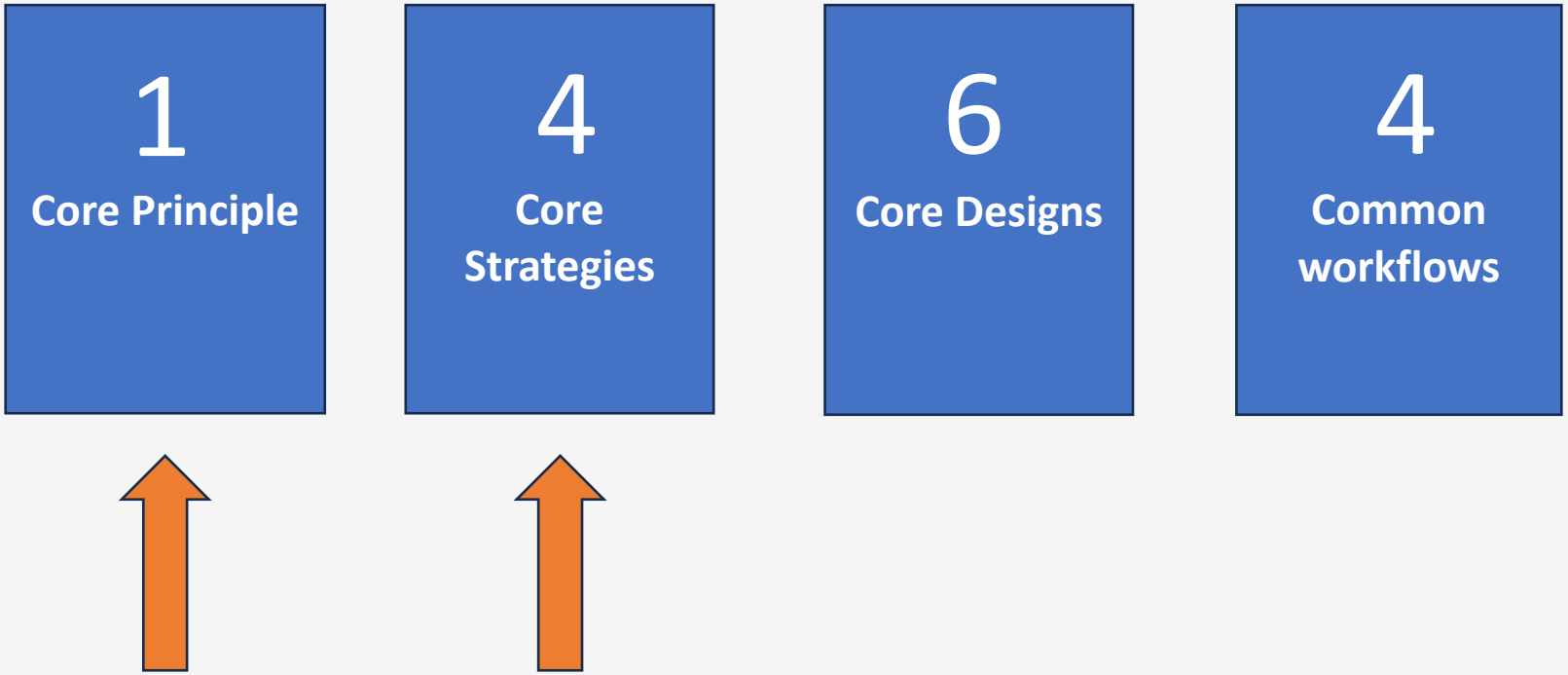


Key highlights of Claude Code

Claude Code: An AI coding assistant that **writes, debugs, and refactors code** through natural language conversation in CLI

- **Rapid adoption** by developers worldwide for faster development since Nov 2024
- Built by a **small team** of 2 engineers at Anthropic, now expanding to 10
- Claude Code **wrote 80%** of its own codes

How Claude Code is Built



Core Principle: Simplicity Over Complexity

UNIX philosophy applied to AI

Simple Approach

- **Thin lightweight** shell wrapper around powerful model
- **Direct access** to full capabilities
- **Transparent** behavior and costs

Complex Approach

- **Multiple layers** of abstraction
- **Hidden optimization** systems
- **Black box** with unclear behavior

Transparent Behavior

Predictable Performance

Full Model Access

Core Strategy I: Explore-Plan-Code Workflow

1

Explore

Read files and understand structure

Identify relevant components

Map dependencies

2

Plan

Design solution strategy

Consider edge cases

Outline implementation steps

3

Execute

Write and test code

Run verification checks

Commit changes

Systematic approach ensures thorough understanding before implementation

Core Strategy II: Test-Driven Development Integration

1. Red: Write Failing Test

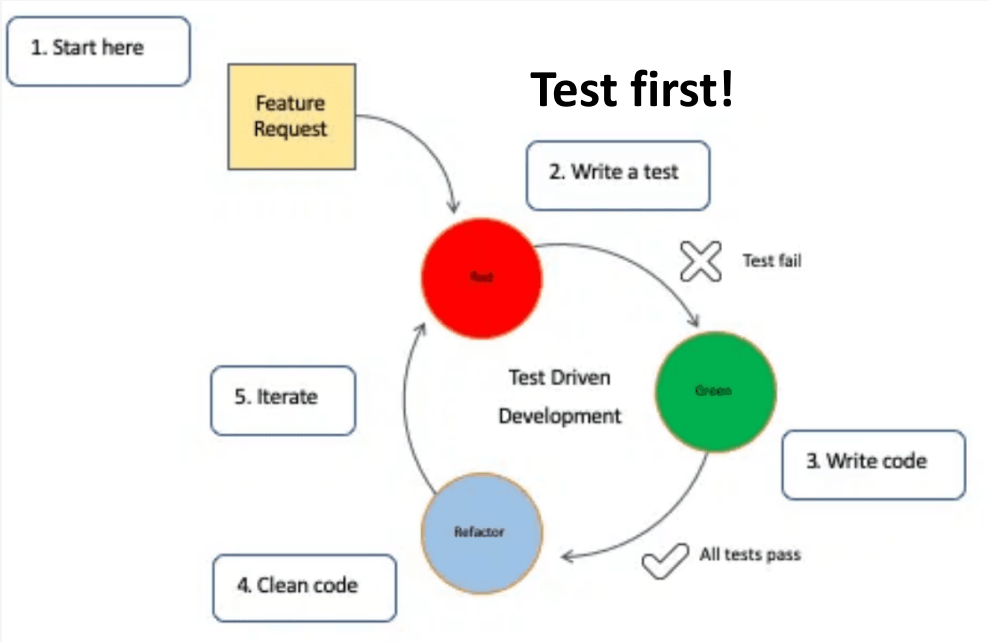
You define the expected behavior with a test that fails

2. Green: Claude Implements

Claude writes code to make the test pass

3. Refactor: Improve Quality

Claude optimizes code while keeping tests green



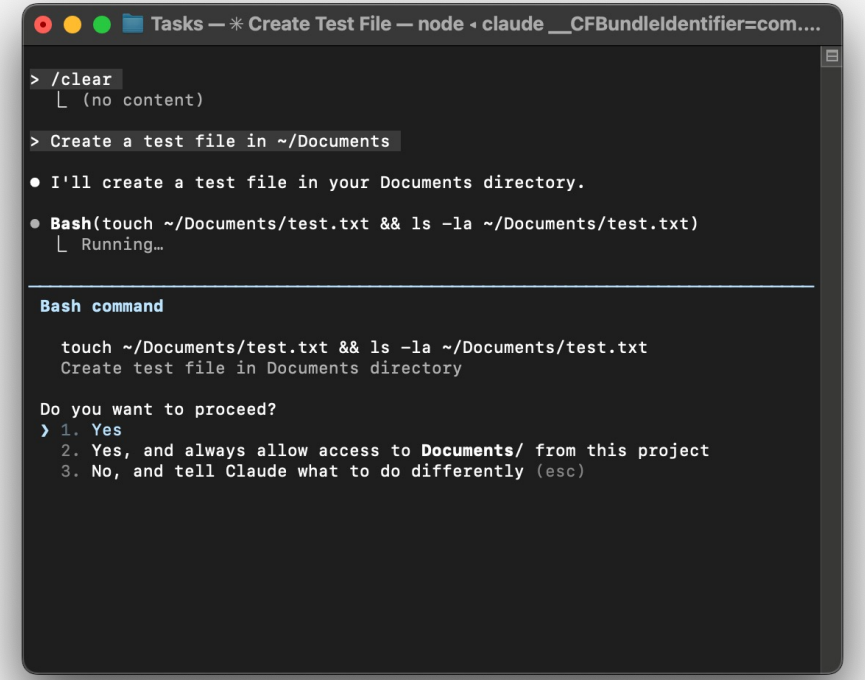
Core Strategy III: Conservative Permissions Model

AI Requests Permission
Claude asks before executing actions

User Reviews Action
You see exactly what will happen

User Approves or Denies
You make the final decision

Action Executes
Only if you approve



You Stay in Control

Core Strategy IV: Extended Thinking for Con

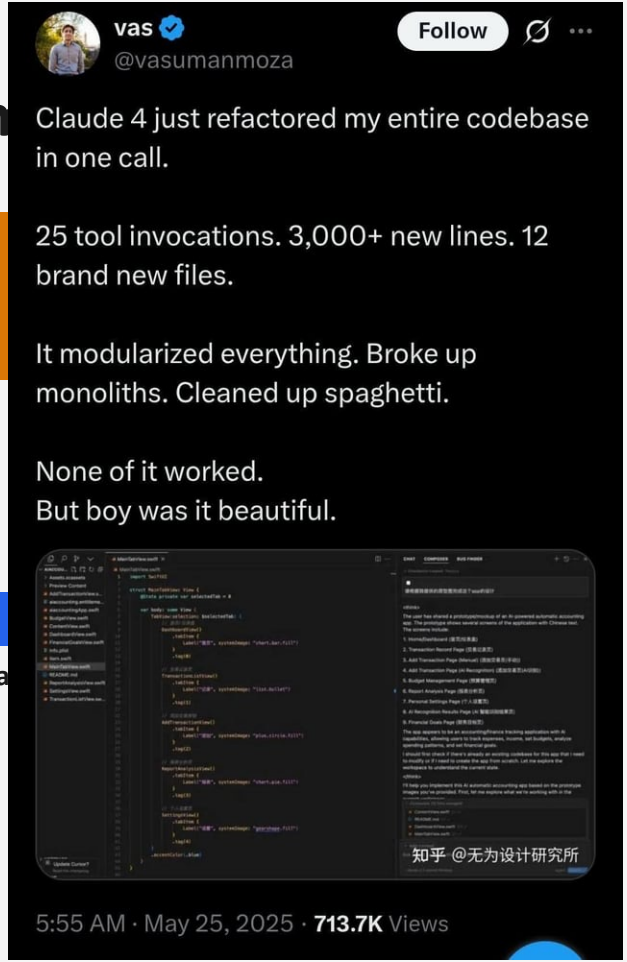
Simple
Single-step tasks

Medium
Multi-step workflows

Example: Architectural Refactoring

1. **Analyze current architecture**
2. **Identify bottlenecks**
3. **Design new structure**
4. **Plan migra**

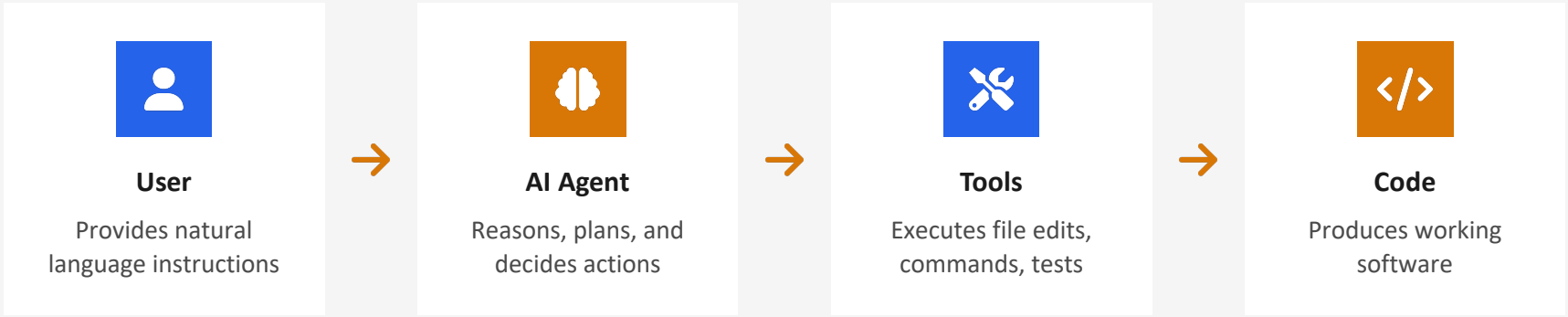
We recommend using the word "think" to trigger extended thinking mode: "think" < "think hard" < "think harder" < "ultrathink."



Outline

1	4	6	4
Core Principle	Core Strategies	Core Designs	Common workflows

Core Design I: The Agentic Architecture



The Agentic Loop in Action

1. **Observe**
   Read codebase and context
2. **Think**
   Reason about the problem
3. **Plan**
   Decide on approach
4. **Act**
   Execute changes with tools
5. **Verify**
   Run tests and check results
6. **Repeat**
   Continue until complete

Core Design II: Agent Memory with CLAUDE.md

An ideal place for documenting:

- Common bash commands
- Core files and utility functions
- Code style guidelines
- Testing instructions
- Repository etiquette (e.g., branch naming, merge vs. rebase, etc.)
- Developer environment setup (e.g., pyenv use, which compilers work)
- Any unexpected behaviors or warnings particular to the project
- Other information you want Claude to remember

AI remembers your conventions and patterns across the entire project

Core Design II: Agent Memory with CLAUDE.md

Human-readable markdown format:

```
# Bash commands
- npm run build: Build the project
- npm run typecheck: Run the typechecker

# Code style
- Use ES modules (import/export) syntax, not CommonJS (require)
- Destructure imports when possible (eg. import { foo } from 'bar')

# Workflow
- Be sure to typecheck when you're done making a series of code changes
- Prefer running single tests, and not the whole test suite, for performance
```

Where to put CLAUDE.md:

- The **root** of your repo, or wherever you run Claude from (the most common usage).
- Any **parent** of the directory where you run Claude.
- Any **child** of the directory where you run Claude.
- Your **home folder** (~/.claude/CLAUDE.md), which applies it to all your claude sessions

Core Design III: Tool Access

1. Use Claude with bash tools: Document frequently used tools in CLAUDE.md
2. Use Claude with MCP:

3. Use custom slash commands: **store prompt templates** in Markdown files within the .claude/commands folder.

.claude/commands/fix-github-issue.md

```
Please analyze and fix the GitHub issue: $ARGUMENTS.

Follow these steps:

1. Use `gh issue view` to get the issue details
2. Understand the problem described in the issue
3. Search the codebase for relevant files
4. Implement the necessary changes to fix the issue
5. Write and run tests to verify the fix
6. Ensure code passes linting and type checking
7. Create a descriptive commit message
8. Push and create a PR

Remember to use the GitHub CLI (`gh`) for all GitHub-related tasks.
```

File Editor

Read and write code files across the entire codebase

Terminal

Execute commands, run tests, and manage dependencies

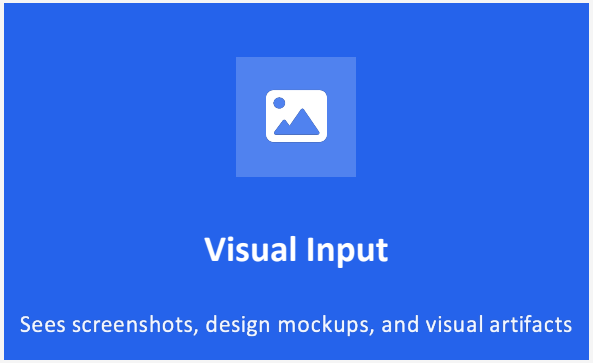
Browser

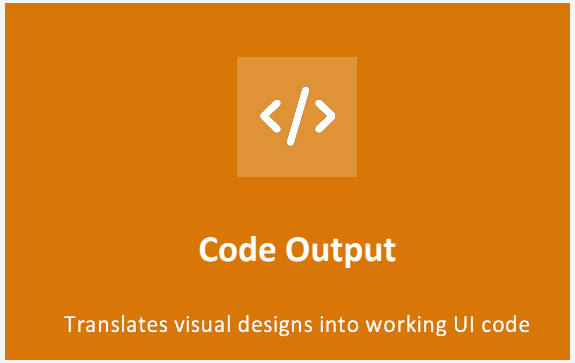
Test and debug web applications in real-time

Version Control

Commit changes and manage Git operations

Core Design IV: Multimodal Perception

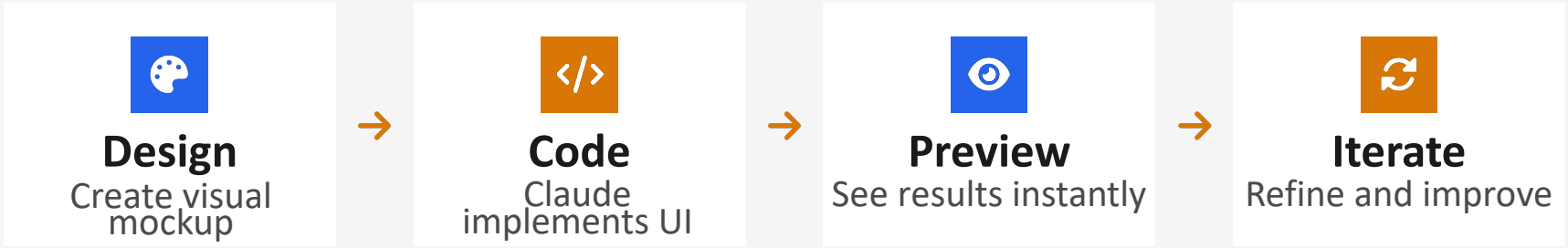




Use Case

Rapid prototyping from visual designs to working code

Core Design IV: Multimodal Perception



Core Design V: Multi-Claude Strategy

Frontend Claude

- Build UI components
- Implement design mockups
- Handle user interactions
- Optimize frontend performance

Backend Claude

- Create API endpoints
- Design database schema
- Implement business logic
- Handle authentication

Core Design VI: Headless Mode for Automation

Run Claude Code programmatically without interactive UI

```
claude -p "Stage my changes and write a set of commits for them" \
  --allowedTools "Bash,Read" \
  --permission-mode acceptEdits
```

Code Review
Automated code review

Log Analysis
Analyze and summarize failures

Test Verification
Quality verification

Documentation
Keep Documentation up-to-date

Effective Human-AI Collaboration Tips

Do's

- Provide **specific instructions** and clear requirements
- Share **rich context** about your project and goals
- Give **active guidance** and iterative feedback
- Define **clear success criteria** and objectives

Don'ts

- Make **vague requests** without details
- Omit **important context** or background
- Just watch **passively** without guiding
- Leave **objectives unclear** or undefined

Outline

1	4	6	4
Core Principle	Core Strategies	Core Designs	Common workflows

Get started in 30 seconds

Prerequisites:

- A **Claude.ai** (recommended) or **Claude Console** account

Install Claude Code:

macOS/Linux | Homebrew | Windows | NPM

```
brew install --cask claude-code
```

Start using Claude Code:

```
cd your-project
claude
```

Common workflows I: Explore, plan, code, commit

Focus on one task at a a time:

1. Ask Claude to read relevant files, images, or URLs, but explicitly tell it not to write any code just yet.
2. Ask Claude to make a plan for how to approach a specific problem, using the word "think" to trigger extended thinking mode: "think" < "think hard" < "think harder" < "ultrathink.”
3. Ask Claude to implement its solution in code. Ask it to explicitly verify the reasonableness of its solution.
4. Ask Claude to commit the result and create a pull request. Have Claude update any READMEs or changelogs with an explanation of what it just did.

Common workflows II: Write tests, commit; code, iterate, commit

Test-first workflow:

1. Ask Claude to write tests based on expected input/output pairs. Be explicit about the fact that you're doing test-driven development
2. Tell Claude to run the tests and confirm they fail. Explicitly telling it not to write any implementation code at this stage is often helpful.
3. Ask Claude to commit the tests when you're satisfied with them.
4. Ask Claude to write code that passes the tests, instructing it not to modify the tests. Tell Claude to keep going until all tests pass.

Common workflows III: Write code, screenshot result, iterate

Visual driven development:

1. Give Claude a way to take browser screenshots (e.g., with the Puppeteer MCP server, an iOS simulator MCP server, or manually copy / paste screenshots into Claude).
2. Give Claude a visual mock by copying / pasting or drag-dropping an image, or giving Claude the image file path.
3. Ask Claude to implement the design in code, take screenshots of the result, and iterate until its result matches the mock.
4. Ask Claude to commit when you're satisfied.

Common workflows IV: Use Claude to interact with git

Git-management:

1. Searching *git* history to answer questions like "What changes made it into v1.2.3?", "Who owns this particular feature?", or "Why was this API designed this way?”
2. Writing commit messages. Claude will look at your changes and recent history automatically to compose a message taking all the relevant context into account
3. Handling complex git operations like reverting files, resolving rebase conflicts, and comparing and grafting patches

Key Takeaways

Simplicity Beats Complexity

Unix philosophy applied to AI: thin wrapper around powerful model delivers transparency and predictability

Transparency Builds Trust

Clear costs, visible behavior, and user control create confidence in AI-assisted development

1	4	6	4
Core Principle	Core Strategies	Core Designs	Common workflows

The Future of Agentic Development

1 2024-2025

Current State

AI coding assistants help developers write code faster and debug more efficiently. Developers remain in full control.

2 2026-2027

Near Future

AI becomes a true collaborative partner, handling complex architectural decisions and multi-file refactoring autonomously.

3 2028+

Long-term Vision

Anyone with an idea can create sophisticated software. Technical barriers dissolve, creativity and design taste becomes the only limit.

Democratizing Software Creation for Everyone