Rapid Engineering with AI

Building at the Speed of Thought

AI Fund Engineering Practices

The Paradigm Shift

From "Vibe Coding" to Rapid Engineering

Like architects who design buildings, not construct them:

- The effort and cost of coding → 0
- Engineering decisions remain critical

We're not writing less code, we're making more decisions

What is Rapid Engineering?

- **Effort of implementation** approaches zero
- **Engineering judgment** becomes everything
- **System thinking** over syntax knowledge
- **Architecture decisions** over implementation details

Software is becoming more about engineering, less about coding.

The 0-to-1 Journey

AI Fund Venture Studio Reality

```
Idea → Prototype → First Customer → Enterprise
  ↓         ↓              ↓              ↓
Hours     Days          Weeks          Months
```

Optimization objective constantly shifts:

- Speed first - bias to action
- Then plan the path to scaled up deployment
- Without painting yourself into corners

Multi-Axis Optimization

Early Stage: Optimize for speed
Growth Stage: Optimize for stability
Enterprise: Optimize for security & compliance

The New Reality:

- Put objectives in the spec
- AI generates code to match
- Code is disposable artifact
- **Specs become the IP**

Architecture Exploration Advantage

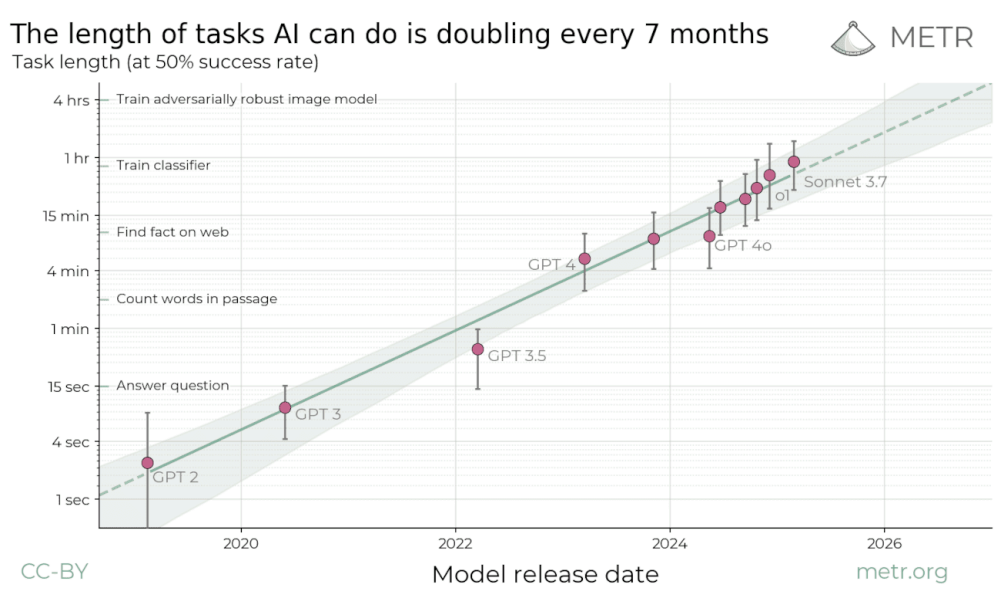
Old World: Stuck with your first architecture forever

Rapid Engineering: Try 3 architectures in a week

Real Examples:

- **APIs**: Stateful API had high latency variance - switched to stateless in hours
- **Databases**: Tested Firebase/NoSQL vs Supabase/SQL side-by-side
- **Providers**: Swapped OpenAI ↔️ Gemini implementations

Tasks complexity exploding



- Task complexity increases exponentially
- What was impossible yesterday is routine today

Why Agentic Search > RAG

RAG (Cursor): Retrieves similar code chunks
Agentic (Claude Code): Explores with purpose

Claude Code with Opus 4:

- Searches intelligently across files
- Builds understanding iteratively
- Maintains context between searches
- Adapts strategy based on findings

How to Use Claude Code Effectively

The Narrative Doc Approach:

- **Write a comprehensive brief** before starting
- **Front-load ALL context** in one document
- **Include**: Background, objectives, constraints, architecture
- **Avoid context-exhausting conversations**

Claude Code: The Memento Principle

Treat Claude like the protagonist in Memento:

- **No memory between sessions** - leave notes!
- **Use CLAUDE.md files** as persistent project memory
- **Place in root or subfolders** for context-specific notes
- **Document decisions & architecture** for "future Claude"

Example: Narrative Doc Structure

```
## Project: User Analytics Dashboard

### Background
We're migrating from Google Analytics to a custom solution...

### Objectives
- Real-time user behavior tracking
- GDPR compliant data handling
- Sub-100ms query performance

### Architecture Decisions
- PostgreSQL with TimescaleDB for time-series
- React Query for caching layer
- Event-driven updates via WebSockets
```

Key Do's and Don'ts

✅ **Do:** Write narrative docs | Use CLAUDE.md files | Describe outcomes | Test everything

❌ **Don't:** Drip-feed requirements | Rely on memory | Skip docs | Assume persistence

Why This Approach Works

- **Narrative docs** = Complete context in one shot
- **CLAUDE.md files** = Persistent team knowledge
- **Outcome focus** = AI figures out implementation
- **Front-loading** = Preserves context window

Result: More complex tasks, fewer iterations

Engineers as Multipliers

New Engineering Superpowers:

1. **Tool Curation**: Which AI for which task
2. **Pattern Teaching**: Effective prompts & workflows
3. **Quality Gates**: Good vs problematic output
4. **Architecture Guidance**: Avoiding dead ends

One engineer can enable 10x more experiments

For Non-Engineers

Rely on Engineers for:

- **Tool Selection**: Claude vs Copilot vs Cursor
- **Best Practices**: How to prompt effectively
- **Quality Assessment**: Is this production-ready?
- **Architecture Review**: Will this scale?

Engineers aren't gatekeepers, they're enablers

Rapid Engineering Workflow

1. **Define the experiment** (Business)
2. **Architect the approach** (Engineering)
3. **Rapid implementation** (AI + Human)
4. **Validate with users** (Product)
5. **Iterate or pivot** (Team)

Days, not months

Technical Debt: Strategic Choice

Real Progression Example:

1. **Week 1-4**: Manual testing only (faster with few features)
2. **Month 2**: Add deployment automation (manual was slowing down)
3. **Month 3**: Automated test suite (complexity demands it)

Key: Document WHEN to transition, not IF

Rapid Engineering is Full Stack

AI handles all layers:

- Frontend (React, Vue, etc.) ✓
- Backend (APIs, databases) ✓
- Infrastructure (Docker, K8s) ✓
- ML/Data pipelines ✓

Engineers guide: Deployment strategy & architecture

The Future is Already Here

Today's Reality:

- Autonomous debugging ✓
- Architecture generation ✓
- Cross-codebase refactoring ✓
- Continuous optimization ✓

These aren't future promises - use them now!

What Remains Human

AI Can't Replace:

- Business judgment
- User empathy
- System thinking
- Human-in-the-loop design

We guide the tools, not the other way around

Key Takeaways

1. **Rapid Engineering > Vibe Coding**
2. **Speed first, enterprise later** (but plan for it)
3. **Agentic AI changes everything**
4. **Engineers are multipliers**, not bottlenecks
5. **Strategic technical debt** is a superpower

Resources

Tools:

- **Claude Code** - Agentic coding
- **Gemini 2.5 Pro** - AI Studio (1M token context window)

Advanced Learning:

- **Mastering Claude Code**: https://www.youtube.com/live/6eBSHbLKuN0

Thank You!

Remember:

We're not automating engineering.
We're engineering at the speed of thought.

Questions?

Contact: eli@aifund.ai