PRE-AI GEN 1 GEN 2 GEN 3a GEN 3b

The Geological Ages of AI-Assisted Development

An Evolution Timeline — And Where We Stand Today

[Logo Placeholder]

TIMELINE

PRE-AI

GEN 1

GEN 2

GEN 3a

GEN 3b

PRE-AI ERA — The Foundation • 1950s-1980s

Early Computing

The origin of life — green phosphor displays, worn keyboards, industrial computing.

Characteristics:

- Manual code entry on terminals
- Punch cards and batch processing
- No syntax assistance
- Knowledge in manuals and minds

TIMELINE

PRE-AI

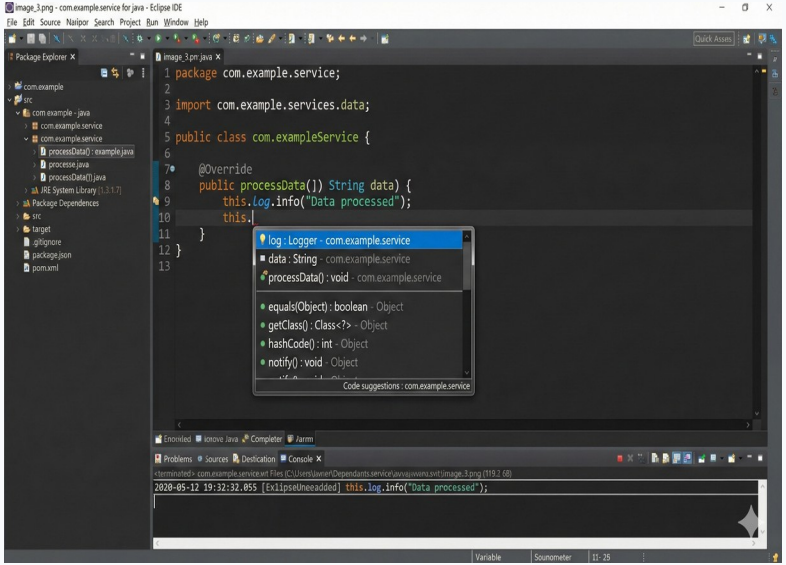
GEN 1

GEN 2

GEN 3a

GEN 3b

PRE-AI ERA — Sophisticated Tooling • 1990s-2010s



Classic IDE Era

Sophisticated but deterministic. Intelligent suggestions powered by parsing — no learning.

Characteristics:

- Syntax highlighting and navigation
- Rule-based autocomplete
- Static analysis and linting
- Integrated debugging

TIMELINE

PRE-AI

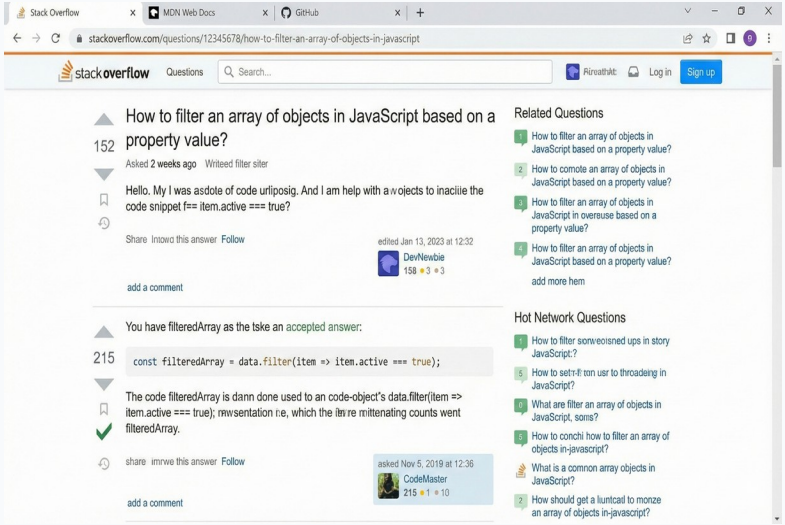
GEN 1

GEN 2

GEN 3a

GEN 3b

PRE-AI ERA — Collective Knowledge • 2008-2020s



Stack Overflow Era

Collective human knowledge. Upvotes, green checkmarks, copy-paste workflows.

Characteristics:

- Search → Find → Copy → Adapt
- Community-curated answers
- Human experts as gatekeepers

⚠️ **Extinction Event:** Declines as conversational AI emerges

TIMELINE

PRE-AI

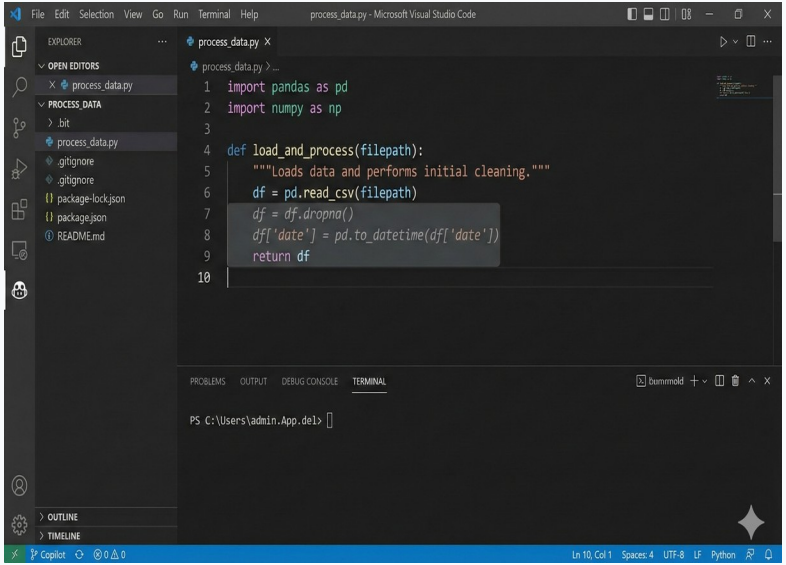
GEN 1

GEN 2

GEN 3a

GEN 3b

GEN 1 — AI ERA BEGINS ML Code Completion • 2021-2022



The Cambrian Explosion

For the first time, AI predicts multiple lines of contextual code — not just method names.

Key Innovations:

- ML-powered predictions
- Context-aware multi-line suggestions
- Trained on billions of code lines

GitHub Copilot | Tabnine | Codeium

TIMELINE

PRE-AI

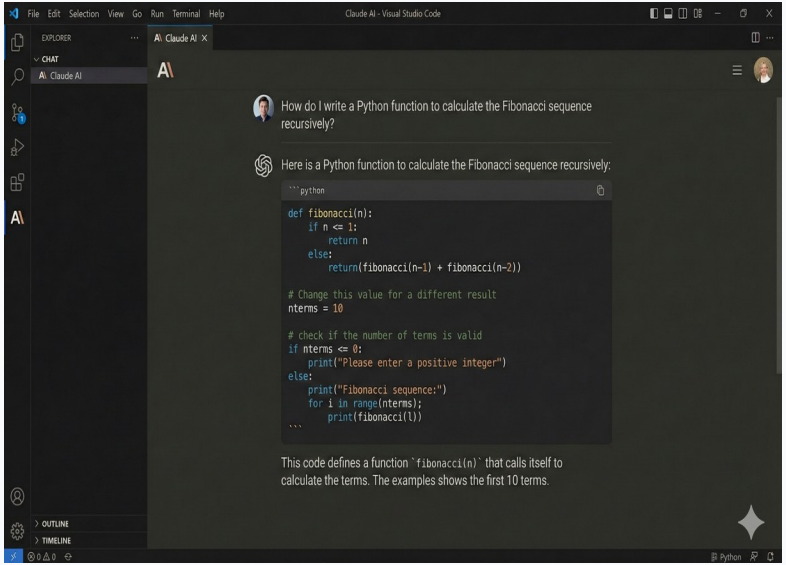
GEN 1

GEN 2

GEN 3a

GEN 3b

GEN 2 — Conversational AI • 2022-2024



The Dialogue Begins

Natural language in, explained code out. AI understands intent, provides context, explains reasoning.

Paradigm Shift:

- Ask in natural language
- Receive explained code
- Iterate through conversation

ChatGPT | Claude | Gemini

TIMELINE

PRE-AI

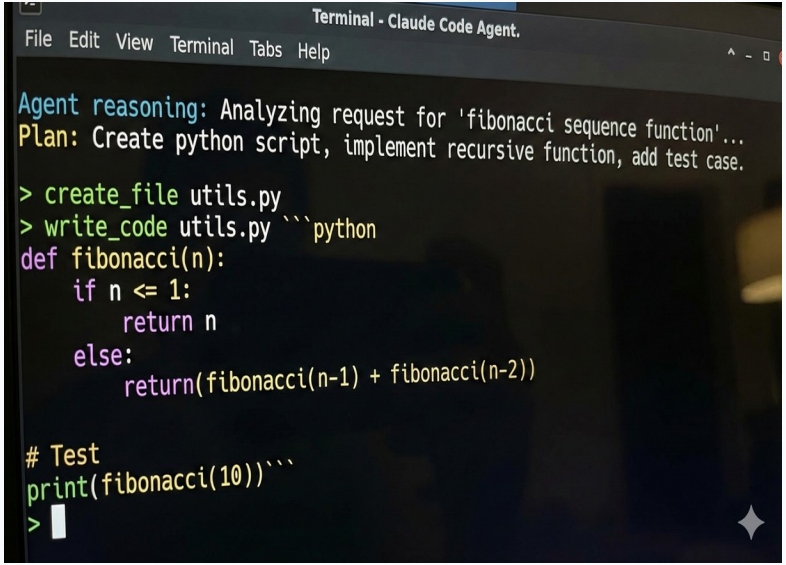
GEN 1

GEN 2

GEN 3a

GEN 3b

GEN 3a — Autonomous Agents • 2024-Present



From Dialogue to Autonomy

AI reasons, plans, and executes. You provide intent, it delivers outcomes.

Agentic Capabilities:

- Autonomous planning
- File system operations
- Multi-step execution
- Self-correction

Claude Code | Cursor Agent | Windsurf

TIMELINE

PRE-AI

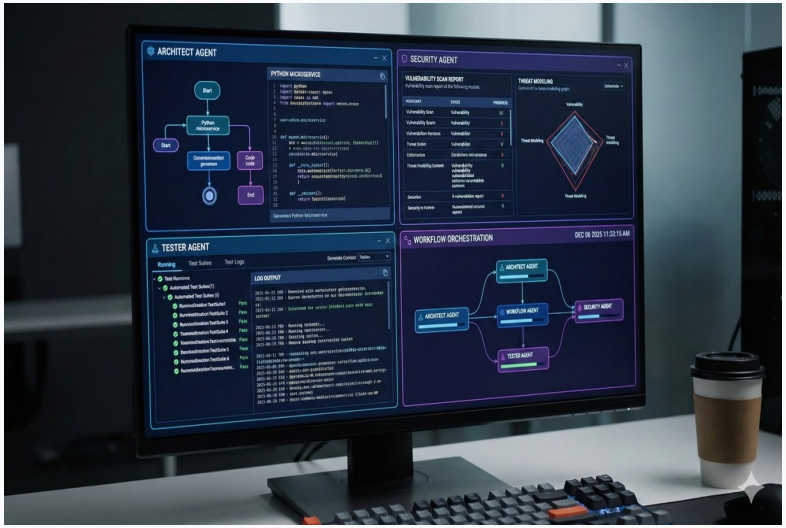
GEN 1

GEN 2

GEN 3a

GEN 3b

GEN 3b — Agentic Factory • Emerging Now



The Emergence of Civilization

Specialized agents — Architect, Security, Tester — orchestrated into a collaborative factory.

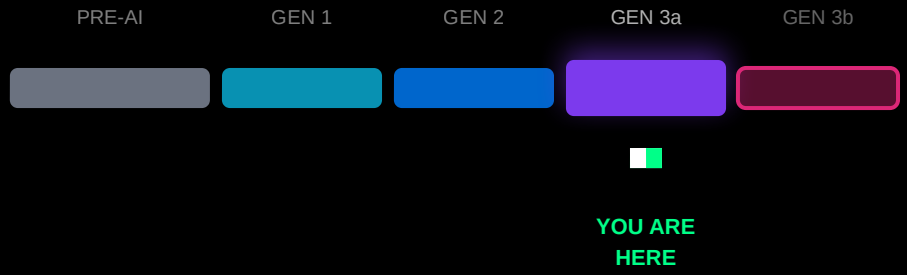
Multi-Agent Architecture:

- Specialized agent roles
- Workflow orchestration
- Collaborative problem-solving

Architect | Security | Tester | Workflow

You Are Here

Industry Standard — Modern Technology Stacks



LEADING EDGE

Python, JavaScript, TypeScript, Go, Rust

TOOLING

Claude Code, Cursor, Windsurf, Copilot

For modern cloud-native technology stacks, the industry is at the Gen 3a frontier

THE GAP

PRE-AI

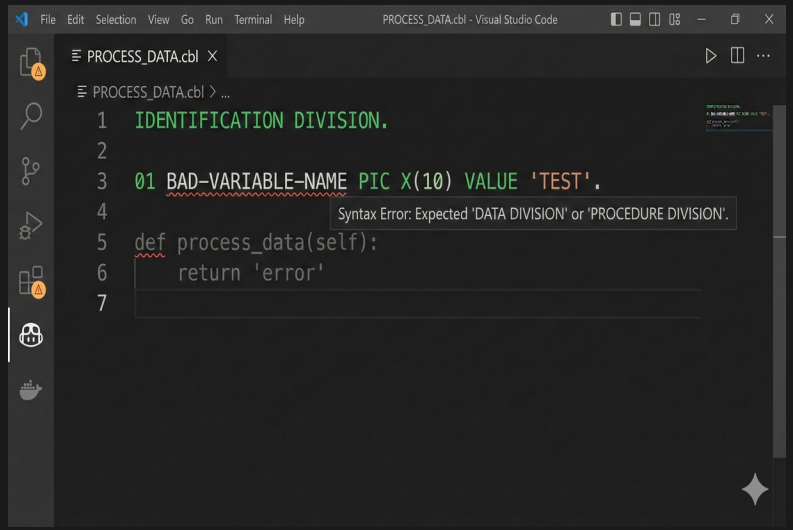
GEN 1

GEN 2

GEN 3a

GEN 3b

⚠️ THE GAP — Mainframe Reality



But for Mainframe...

Modern AI falls apart with COBOL. The AI hallucinates Python inside a COBOL file.

The Reality:

- Limited COBOL/CICS/DB2 training data
- No mainframe toolchain integration
- AI can't "see" zOS environments

zOS / COBOL / CICS / DB2 — Still stuck in Pre-AI era

The Licensing Fallacy

✗ The Assumption

"Give developers AI licenses = problem solved"

What licenses give you:

- Access to AI tools
- Chat interfaces
- Code completion for modern languages

✓ The Reality

"The gap is in the tools, not the seats"

What mainframe actually needs:

- COBOL-aware AI models
- zOS environment integration
- SCLM/Endevor toolchain bridges
- Context about our specific systems

Key Insight: Buying more seats on a train that doesn't go to your destination doesn't help you get there.

Where Investment Needs to Go

1. Tooling Integration

- VS Code SCLM extensions
- zOS Explorer enhancements
- MCP bridges to mainframe

2. Specialized Training

- COBOL fine-tuning
- CICS/DB2 patterns
- Enterprise SDLC context

3. Context Bridges

- zOS environment exposure
- Dataset access patterns
- Job scheduling awareness

4. Specialized Agents

- COBOL migration agent
- JCL optimization agent
- Mainframe security agent

Investment Priority: Build the bridge BEFORE buying tickets to cross it.

Key Takeaways

1. **Industry is at Gen 3a/3b frontier** — autonomous agents and agentic factories are here for modern stacks
2. **Mainframe is stuck in Pre-AI** — AI tools trained on Python/JS fail catastrophically with COBOL
3. **Gap is tooling, not licensing** — investment needs to go into integration, training, and specialization