

---

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

## TIMELINE

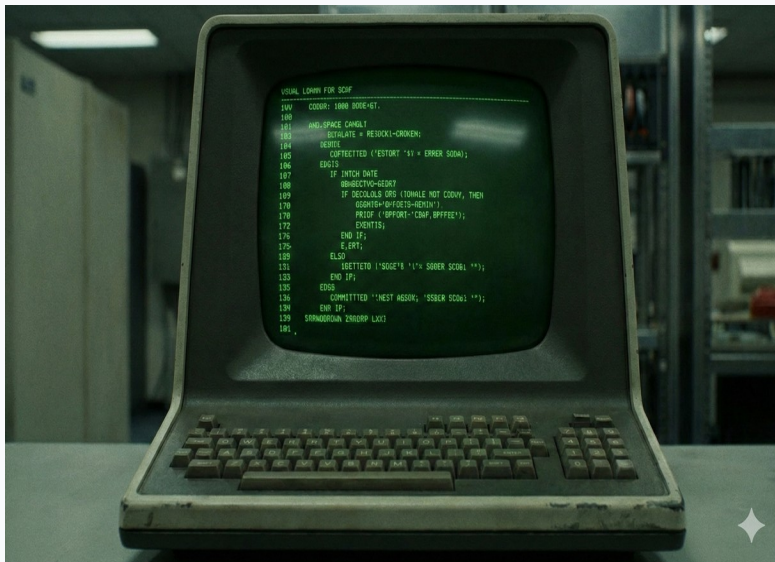
PRE-AI

GEN 1

GEN 2

GEN 3a

GEN 3b



## 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

## PRE-AI ERA

Sophisticated Tooling • 1990s-2010s

### TIMELINE

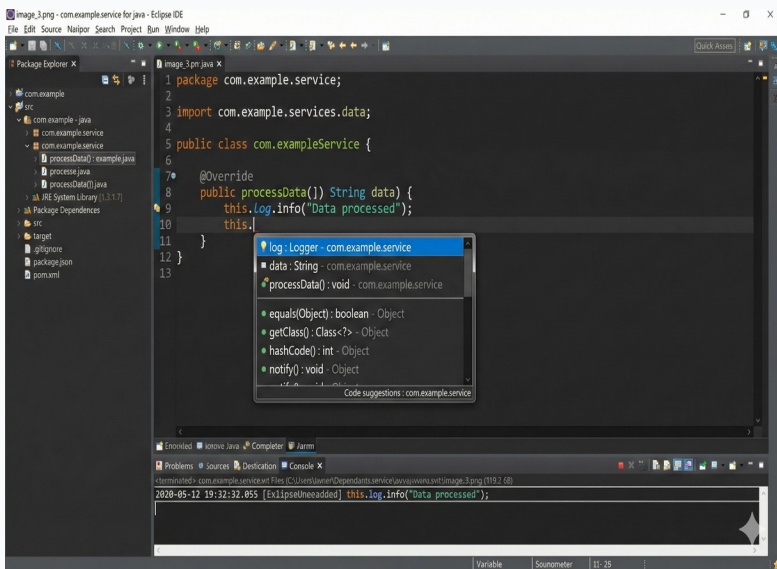
PRE-AI

GEN 1

GEN 2

GEN 3a

GEN 3b



## 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

The screenshot shows a Stack Overflow page for the question "How to filter an array of objects in JavaScript based on a property value?". The question was asked 2 weeks ago by a user named "siler" and has 152 votes. The accepted answer, provided by "DevNewbie" (158 votes), shows the following code: 

```
const filteredArray = data.filter(item => item.active === true);
```

 The answer explains that the `filter` method is used to create a new array with elements that pass a test. The test is a function that returns `true` for objects where `item.active` is `true`. The page also includes sections for "Related Questions" and "Hot Network Questions".

## 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

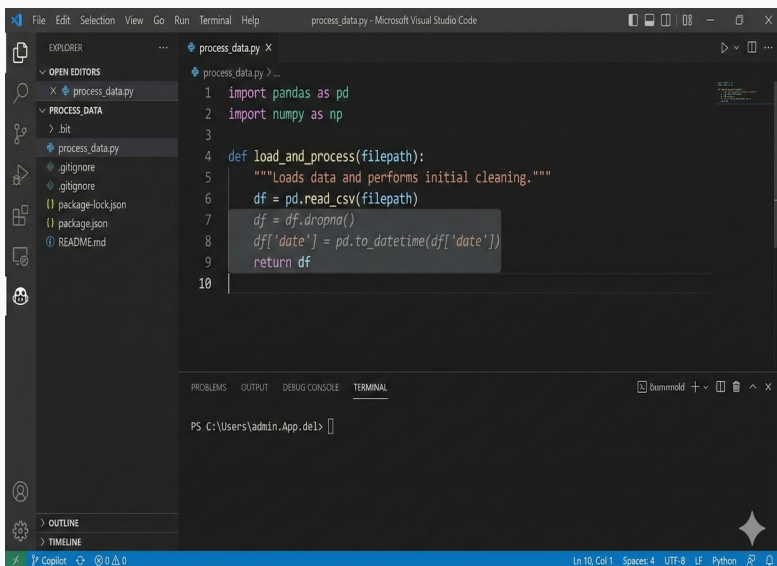
PRE-AI

GEN 1

GEN 2

GEN 3a

GEN 3b



```
1 import pandas as pd
2 import numpy as np
3
4 def load_and_process(filepath):
5     """Loads data and performs initial cleaning."""
6     df = pd.read_csv(filepath)
7     df = df.dropna()
8     df['date'] = pd.to_datetime(df['date'])
9     return df
10
```

## 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

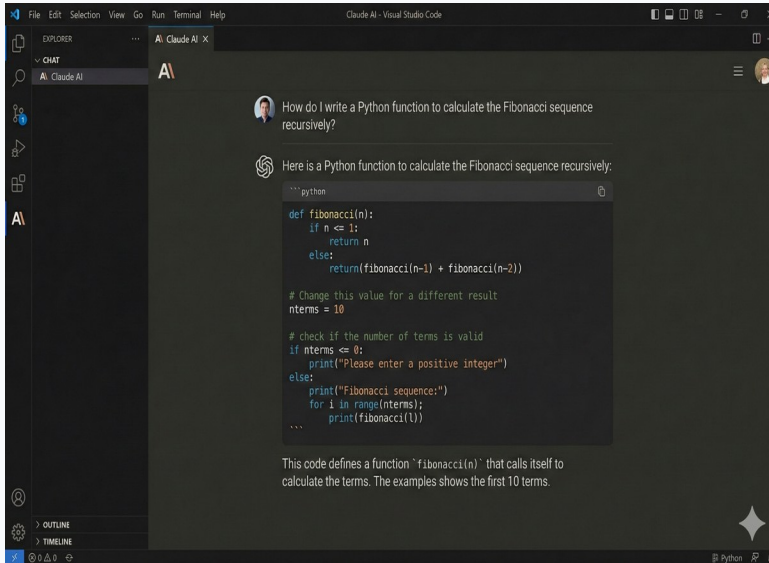
PRE-AI

GEN 1

GEN 2

GEN 3a

GEN 3b



## 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

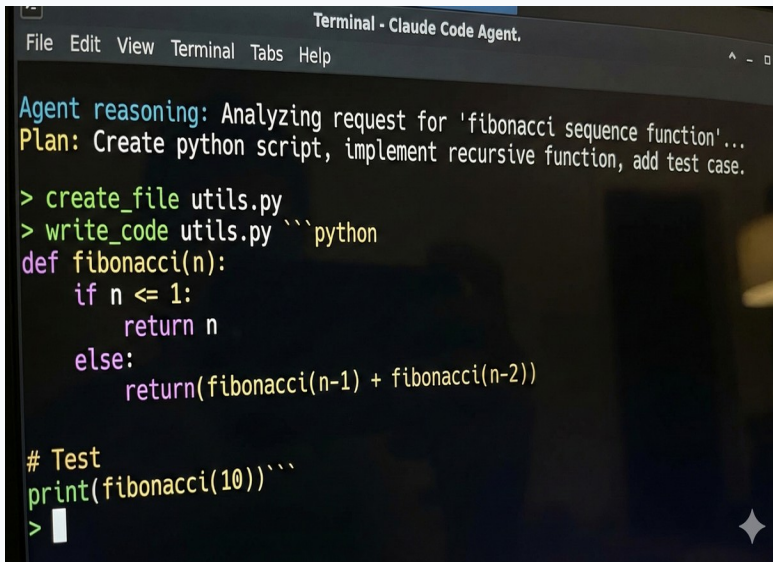
PRE-AI

GEN 1

GEN 2

GEN 3a

GEN 3b



```
File Edit View Terminal Tabs Help

Agent reasoning: Analyzing request for 'fibonacci sequence function'...
Plan: Create python script, implement recursive function, add test case.

> create_file utils.py
> write_code utils.py ```python
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))

# Test
print(fibonacci(10))```
> 
```

## 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

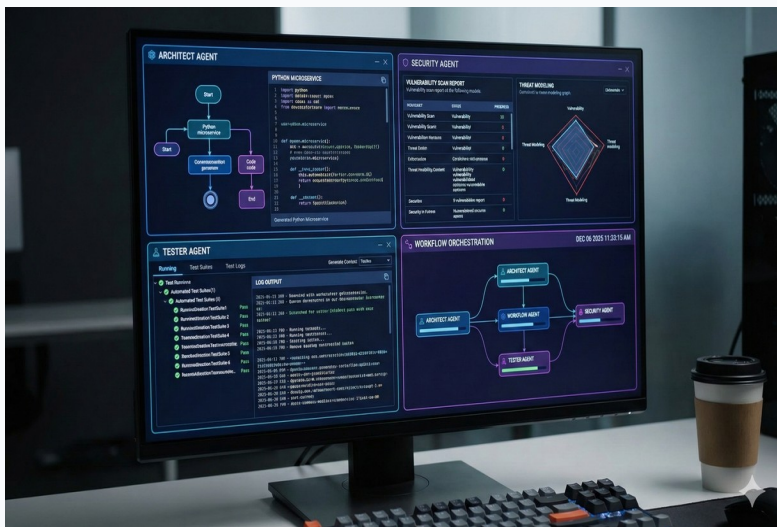
PRE-AI

GEN 1

GEN 2

GEN 3a

GEN 3b



## 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

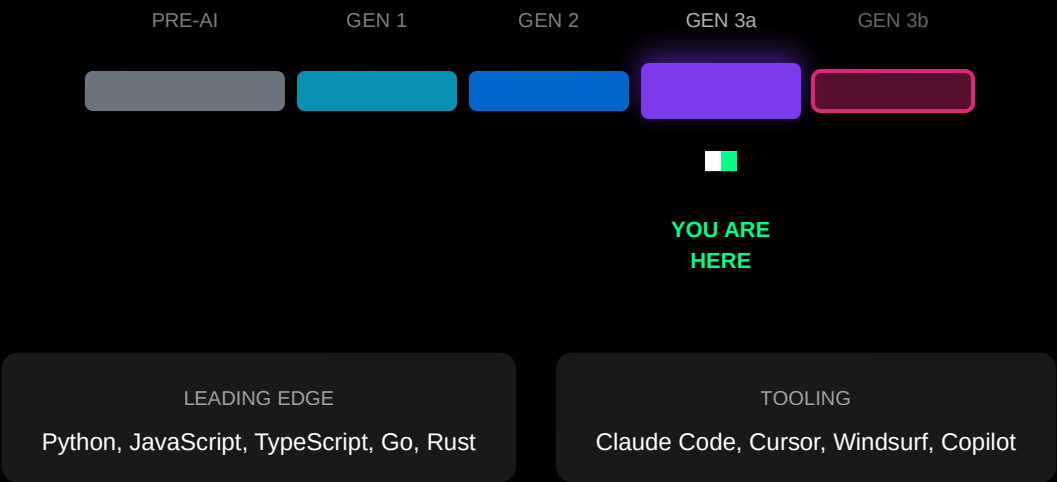
Tester

Workflow



# You Are Here

Industry Standard — Modern Technology Stacks



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

## ⚠ THE GAP

Mainframe Reality

### THE GAP

PRE-AI

GEN 1

GEN 2

GEN 3a

GEN 3b

```
PROCESS_DATA.cbl X
PROCESS_DATA.cbl > ...
1  IDENTIFICATION DIVISION.
2
3  01 BAD-VARIABLE-NAME PIC X(10) VALUE 'TEST'.
4
5  def process_data(self):
6      return 'error'
7
```

Syntax Error: Expected 'DATA DIVISION' or 'PROCEDURE DIVISION'.

## 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