

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

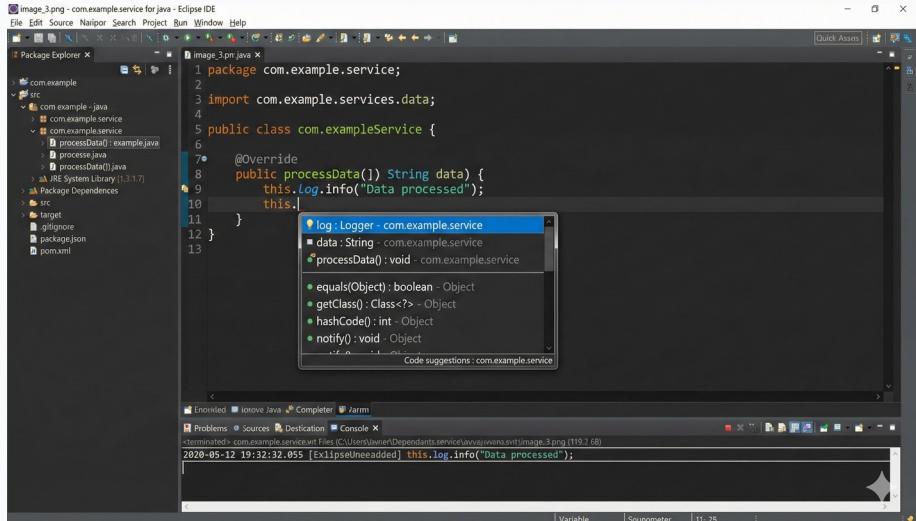


Early Computing

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 of any kind



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

A screenshot of a web browser window showing a Stack Overflow question. The URL in the address bar is stackoverflow.com/questions/12345678/how-to-filter-an-array-of-objects-in-javascript. The page displays a question with 152 answers. One answer has been accepted, indicated by a green checkmark icon. The accepted answer is from a user named DevNewbie and was posted on January 13, 2023. The code provided in the answer is:

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

The question asks how to filter an array of objects in JavaScript based on a property value. Below the question, there are sections for "Related Questions" and "Hot Network Questions", each listing several similar questions with their own details.

Stack Overflow Era

The collective human knowledge base. Upvotes, green checkmarks, copy-paste workflows.

Characteristics:

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

Extinction Event: This workflow declines as conversational AI emerges

A screenshot of Microsoft Visual Studio Code interface. The title bar says "process_data.py - Microsoft Visual Studio Code". The left sidebar shows a file tree with "process_data.py" selected. The main editor area displays the following Python code:

```
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 line "def load_and_process(filepath):" has a light gray rectangular box highlighting it, indicating it is the current target for AI prediction. Below the editor are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is active, showing the command "PS C:\Users\admin.App del> []". At the bottom, status bar items include "Ln 10, Col 1", "Spaces: 4", "UTF-8", "LF", "Python", and a Copilot icon.

The Cambrian Explosion

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

Key Innovations:

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

Copilot Tabnine Codeium

The screenshot shows a Visual Studio Code interface with a dark theme. At the top, the menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. A tab bar shows "Claude AI - Visual Studio Code". The left sidebar has sections for Explorer, Chat, and AI. The main area is titled "AI Claude AI" and contains a conversation:

How do I write a Python function to calculate the Fibonacci sequence recursively?

Here is a Python function to calculate the Fibonacci sequence recursively:

```
```python
def fibonacci(n):
 if n <= 1:
 return n
 else:
 return(fibonacci(n-1) + fibonacci(n-2))

Change this value for a different result
terms = 10

check if the number of terms is valid
if terms <= 0:
 print("Please enter a positive integer")
else:
 print("Fibonacci sequence:")
 for i in range(terms):
 print(fibonacci(i))
```


This code defines a function `fibonacci(n)` that calls itself to calculate the terms. The examples shows the first 10 terms.


```

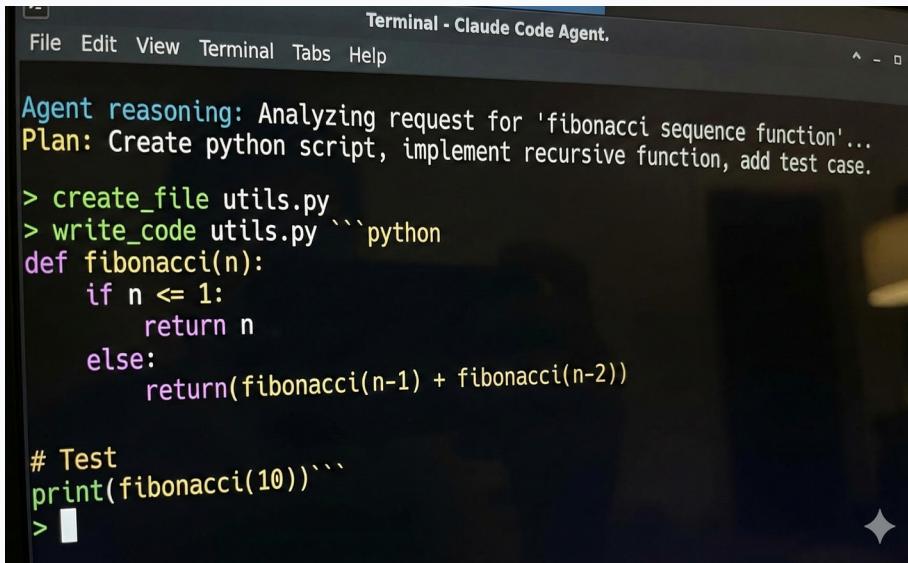
The Dialogue Begins

Natural language in, explained code out. AI understands intent and explains its reasoning.

Paradigm Shift:

- Ask questions in natural language
- Receive explained, documented code
- Iterate through conversation

ChatGPT Claude Gemini



Terminal - Claude Code Agent.

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))``  
> █
```

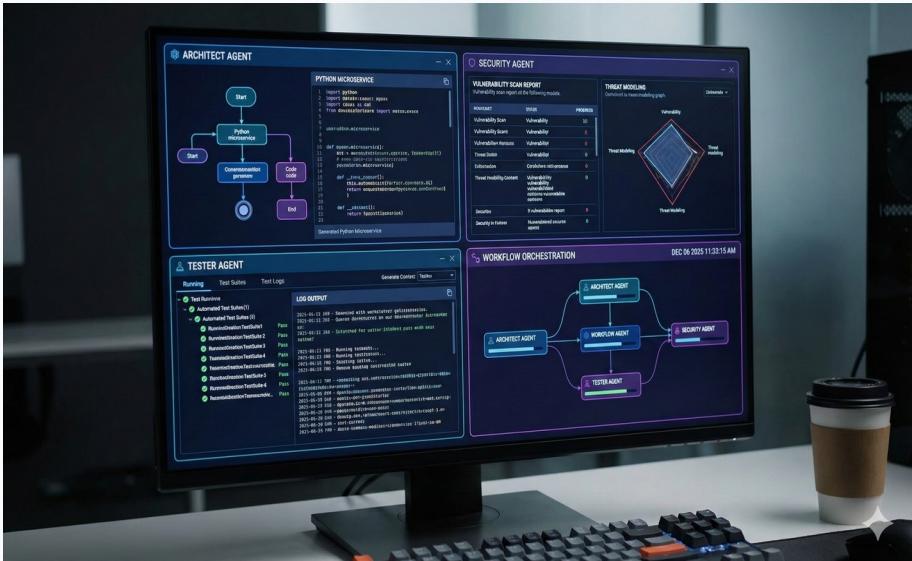
From Dialogue to Autonomy

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

Agentic Capabilities:

- Autonomous planning and reasoning
- File system operations
- Multi-step task execution

Claude Code Cursor Windsurf



Emergence of Civilization

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

Multi-Agent Architecture:

- Specialized agent roles
- Workflow orchestration
- Division of labor at scale

Architect Security Tester Workflow

You Are Here

Industry Standard — Modern Technology Stacks



YOU ARE HERE

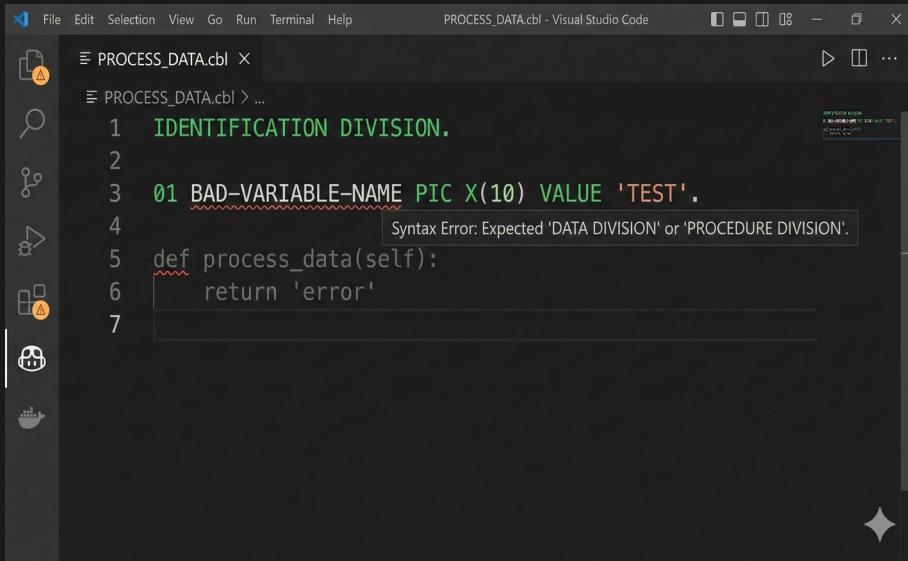
LEADING EDGE

Python, JavaScript, TypeScript, Go, Rust

TOOLING

Claude Code, Cursor, Windsurf, Copilot

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



A screenshot of Visual Studio Code showing a COBOL program named `PROCESS_DATA.cbl`. The code includes an `IDENTIFICATION DIVISION`, a `01` definition for a variable named `BAD-VARIABLE-NAME` with a `PIC X(10)` type and a `VALUE 'TEST'` initial value. A syntax error message is displayed: "Syntax Error: Expected 'DATA DIVISION' or 'PROCEDURE DIVISION'." Below this, there is Python code: `def process_data(self): return 'error'`. The code editor has a dark theme with light-colored syntax highlighting.

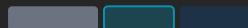
But for Mainframe...

Modern AI tools trained on Python fall apart with COBOL. Mixing languages, wrong structure, syntax errors.

The Reality:

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

MAINFRAME POSITION:



The Licensing Fallacy

Why buying seats does not close the gap



THE ASSUMPTION

Give developers Copilot or Claude licenses and the problem is solved

What licenses give you:

- Access to generic AI models
- Training data dominated by modern stacks
- Tools designed for cloud-native workflows



THE REALITY

The gap is in the tools, not the seats

What mainframe needs:

- Specialized COBOL/CICS/DB2 training
- Mainframe toolchain integration
- zOS-aware context and workflows

Where Investment Needs to Go

Closing the mainframe AI gap

1

Tooling Integration

VS Code extensions bridging modern IDEs with mainframe workflows

2

Specialized Training

Fine-tuning or RAG with COBOL/CICS/DB2 codebases

3

Context Bridges

Systems exposing zOS context to AI — copybooks, JCL, CICS regions

4

Specialized Agents

Purpose-built agents understanding mainframe ecosystems

Key insight: These investments make AI licenses valuable for mainframe — not the other way around

THE BOTTOM LINE

The AI revolution in development is real — but mainframe is being left behind

Closing the gap requires strategic investment in tooling and integration, not just seat licenses.

1

Industry is at Gen 3a —
autonomous agents

2

Mainframe is stuck in Pre-
AI / early Gen 1

3

The gap is in tooling, not
licensing