

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

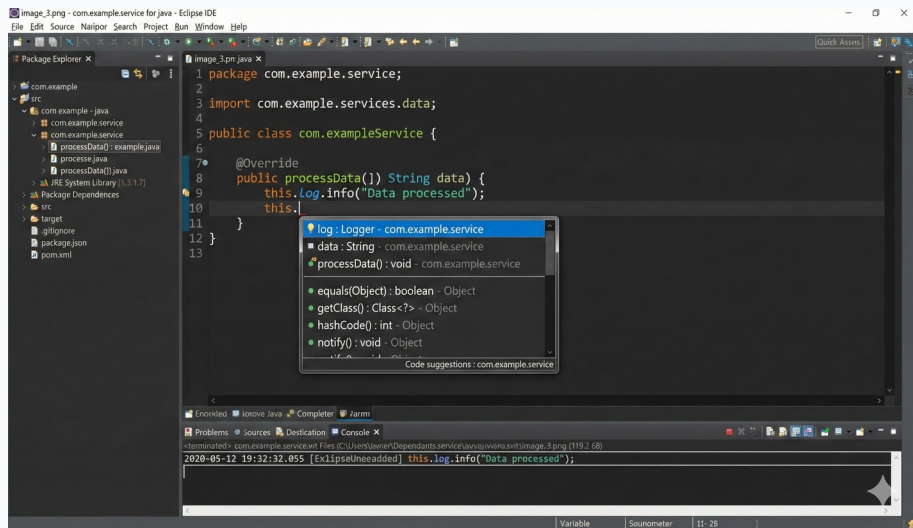


Early Computing

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

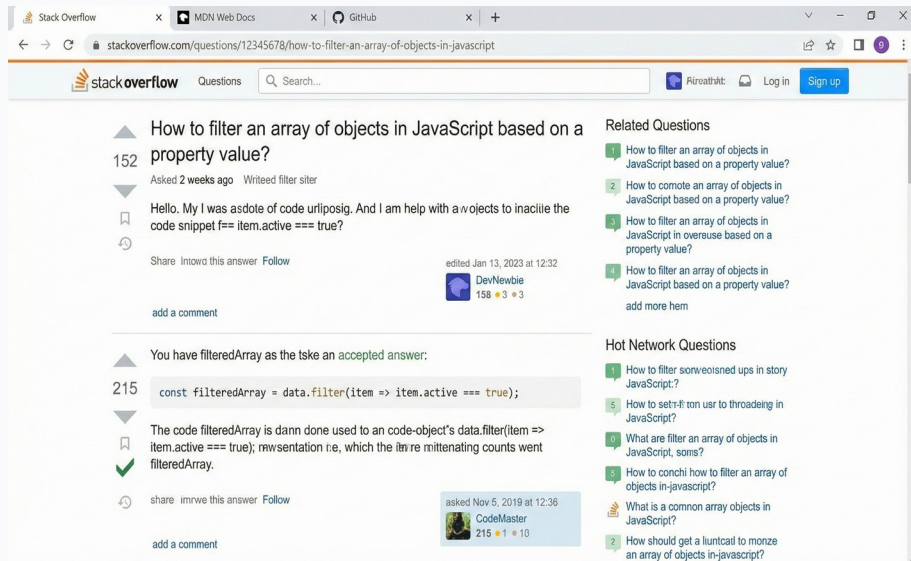


Classic IDE Era

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

Characteristics:

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



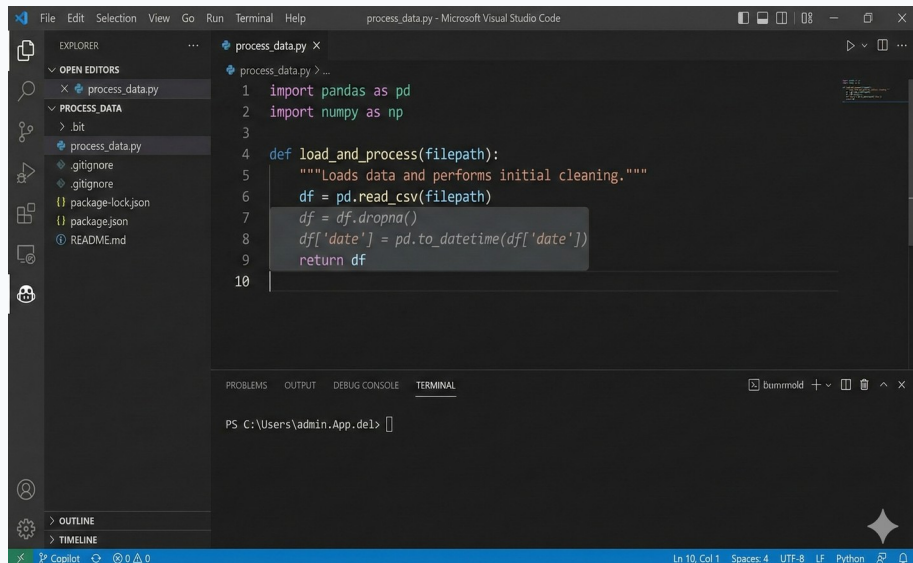
Stack Overflow Era

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



```
File Edit Selection View Go Run Terminal Help
process_data.py - Microsoft Visual Studio Code

EXPLORER
  process_data.py x
  process_data.py > ...
  process_data.py
  .gitignore
  .gitignore
  package-lock.json
  package.json
  README.md

OPEN EDITORS
  process_data.py x
  process_data.py > ...
  process_data.py
  .gitignore
  .gitignore
  package-lock.json
  package.json
  README.md

PROCESS_DATA
  process_data.py
  .gitignore
  .gitignore
  package-lock.json
  package.json
  README.md

def load_and_process(filepath):
    """Loads data and performs initial cleaning."""
    df = pd.read_csv(filepath)
    df = df.dropna()
    df['date'] = pd.to_datetime(df['date'])
    return df

10 |

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\admin\AppData\Local\Microsoft\Windows\apps\Bummmold\ >
```

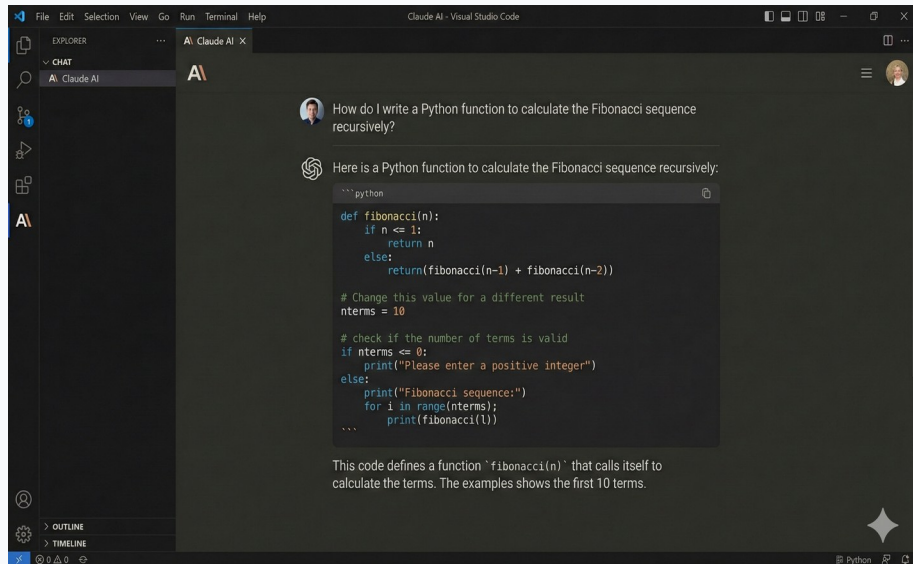
The Cambrian Explosion

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](#)



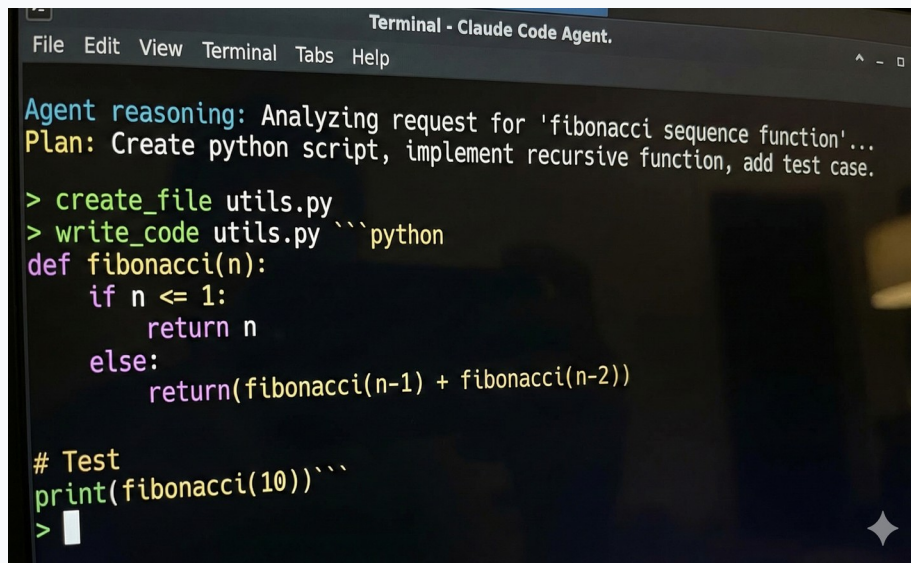
The Dialogue Begins

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))```  
> 
```

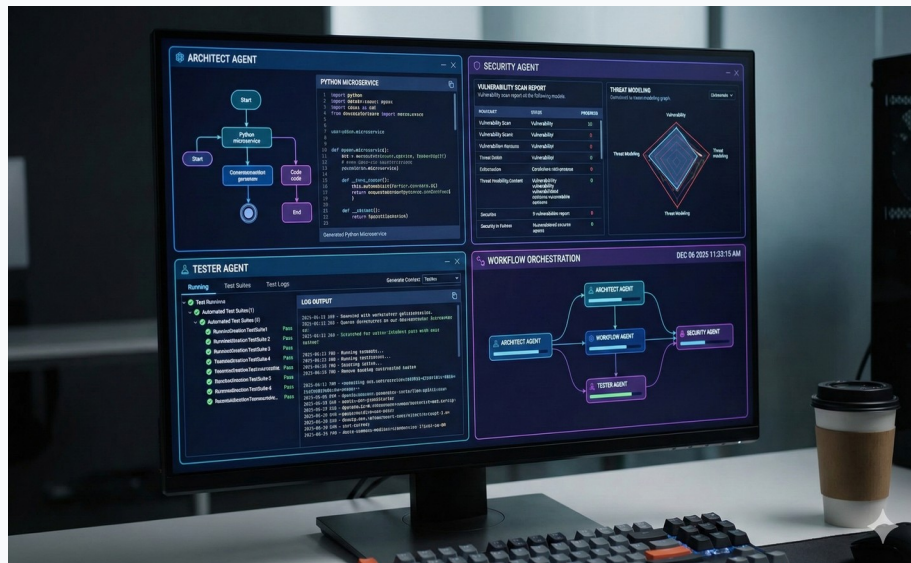
From Dialogue to Autonomy

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

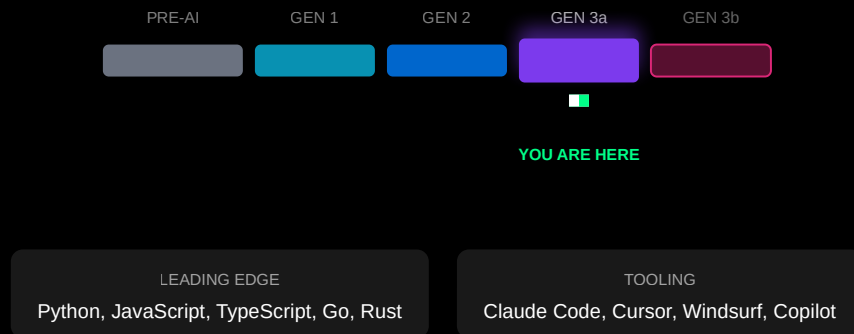
Security

Tester

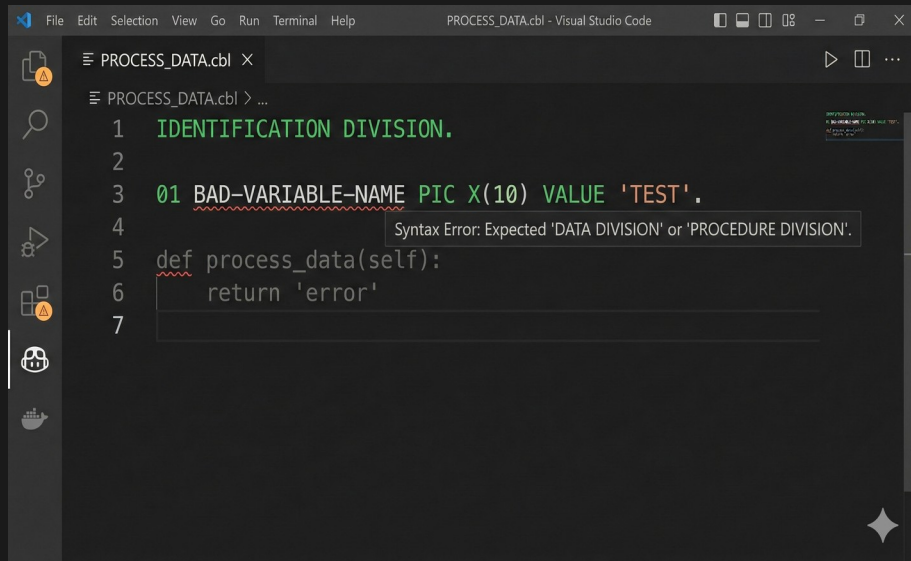
Workflow

You Are Here

Industry Standard — Modern Technology Stacks



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



```
PROCESS_DATA.cbl
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 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

Investment needs to go into tooling and integration — not just seat licenses

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