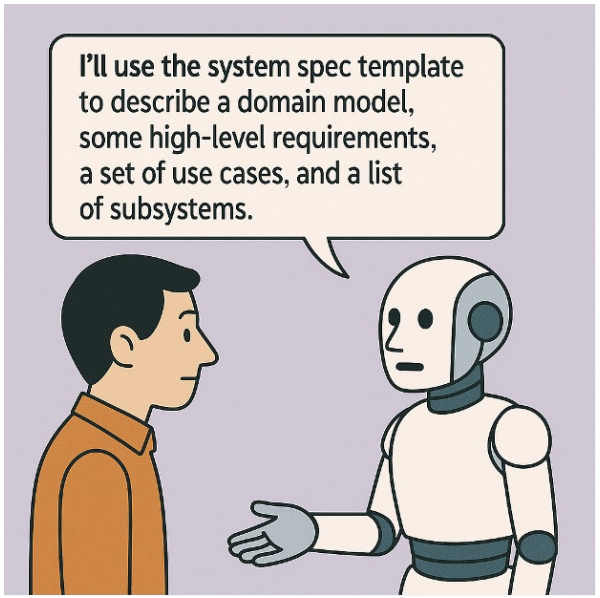
AI-Assisted MBSE for Agentic Software Synthesis

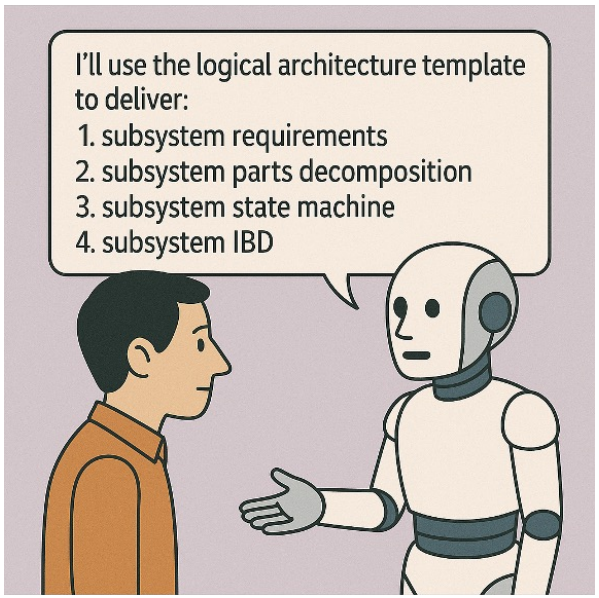
Better Specs Yield Better Outcomes

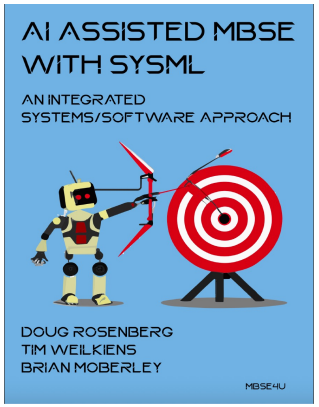
Doug Rosenberg, Parallel Agile, Inc. and Caltech CTME
Brian Moberley, Innovative Defense Technologies
Dr. Rick Hefner, Caltech CTME

We've Been Using AI to Generate SysML Specs for a couple of years

Template-driven SysML generation evolved into agentic software synthesis



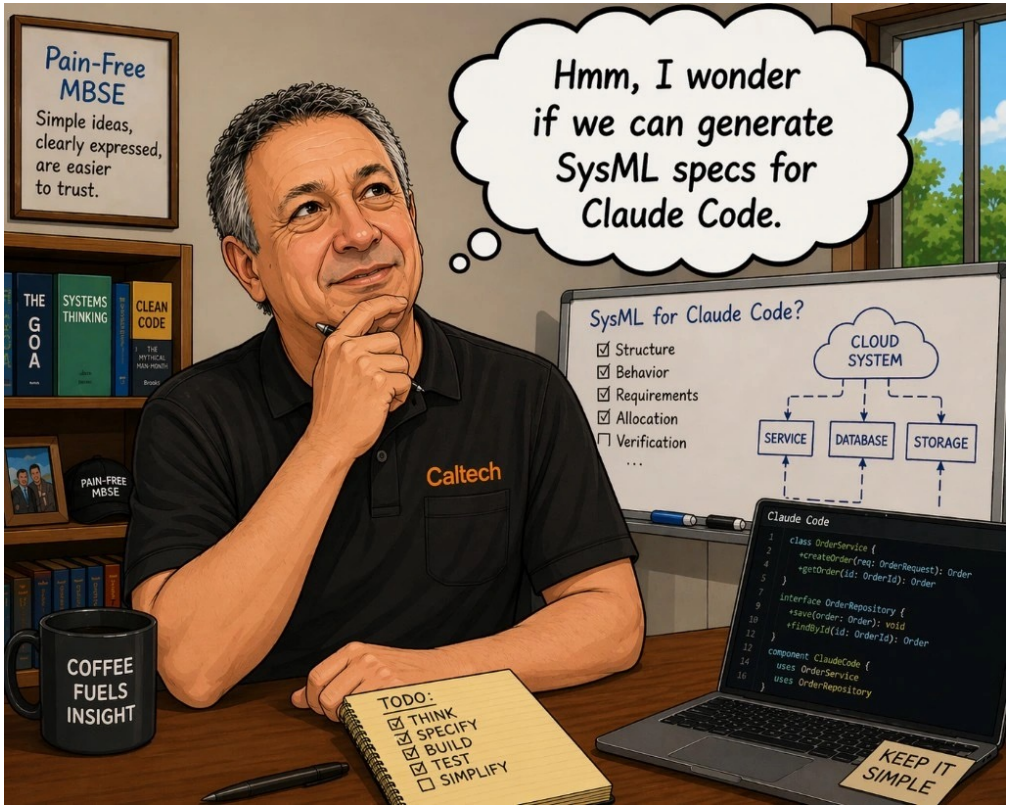




Can SysML Specs Improve AI Coding?

For months, Brian had been encouraging me to try vibe coding with Claude Code

Finally, one day…



AIM Layered Specification Structure

SysML specs commonly contain

- *Requirements*
- *Use Cases*
- *State Machines*
- *Test Cases*

These are all useful to Code Generating Agents

We designed an AIM template to produce good specs

Test coverage is improved through specifications

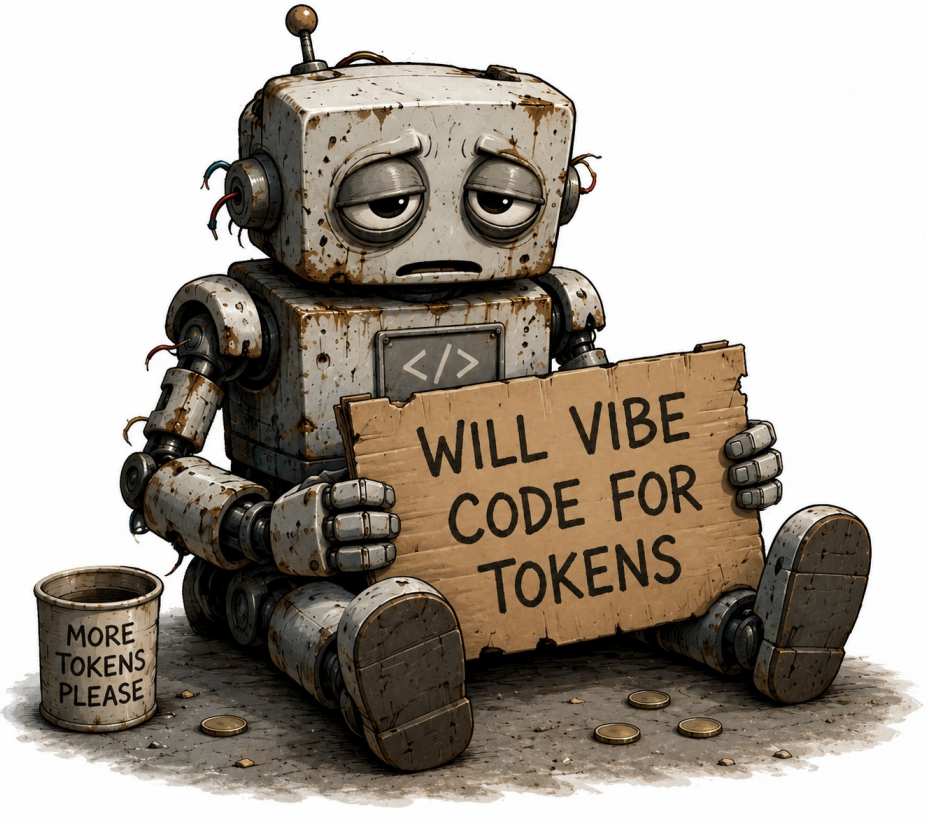
Why Vibe Coding Often Struggles

Not too different from human programming…

High token cost from repeated rebuilds
Re-coding is expensive, even with AI Agents

Vibe-coded apps are difficult to test and trust
If you have no spec, what do you test against?

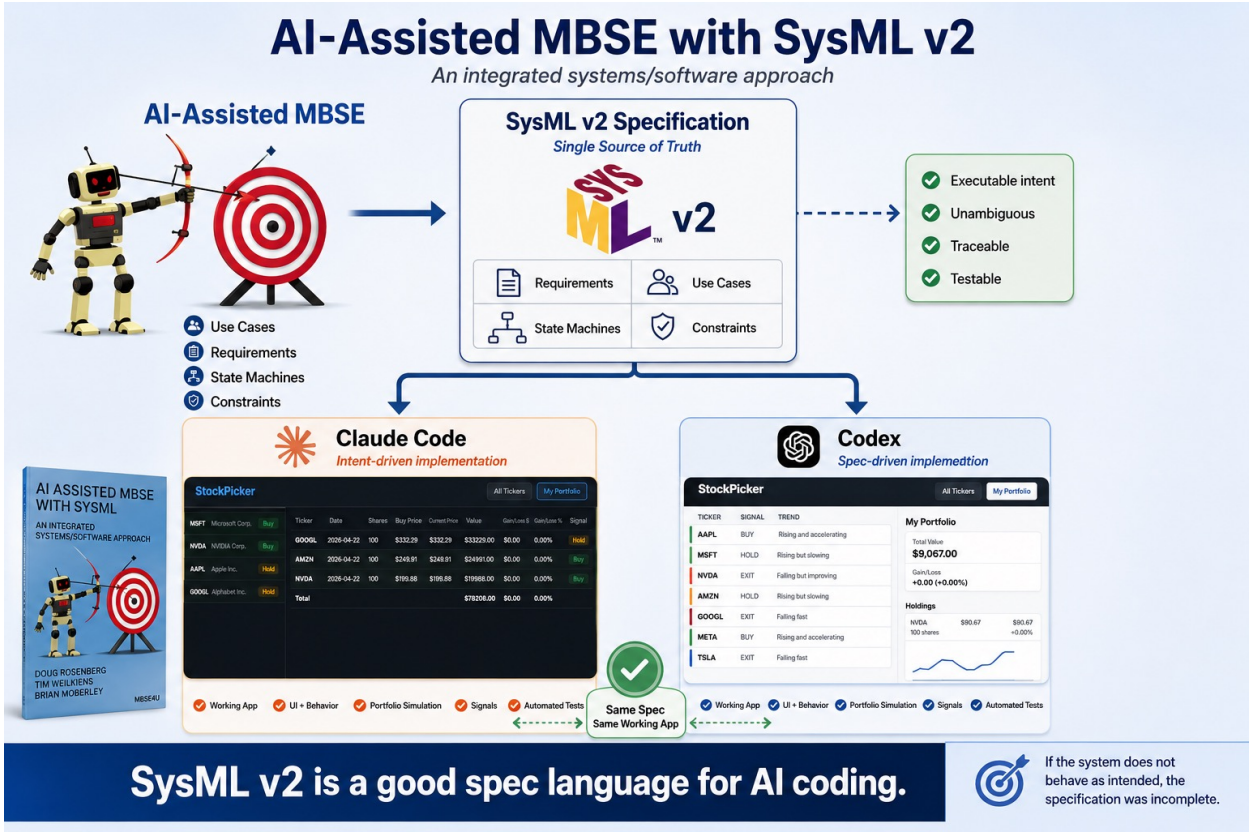
AI loses context (AI Amnesia)
It forgets what it had learned in the previous sessions and un-does things that were previously working



SysML v2 is a good language for code gen specs

We tried SysML v2 as a spec language for AI coding agents

It worked well for both Claude Code and for Codex



Lower Token Cost, Faster Progress

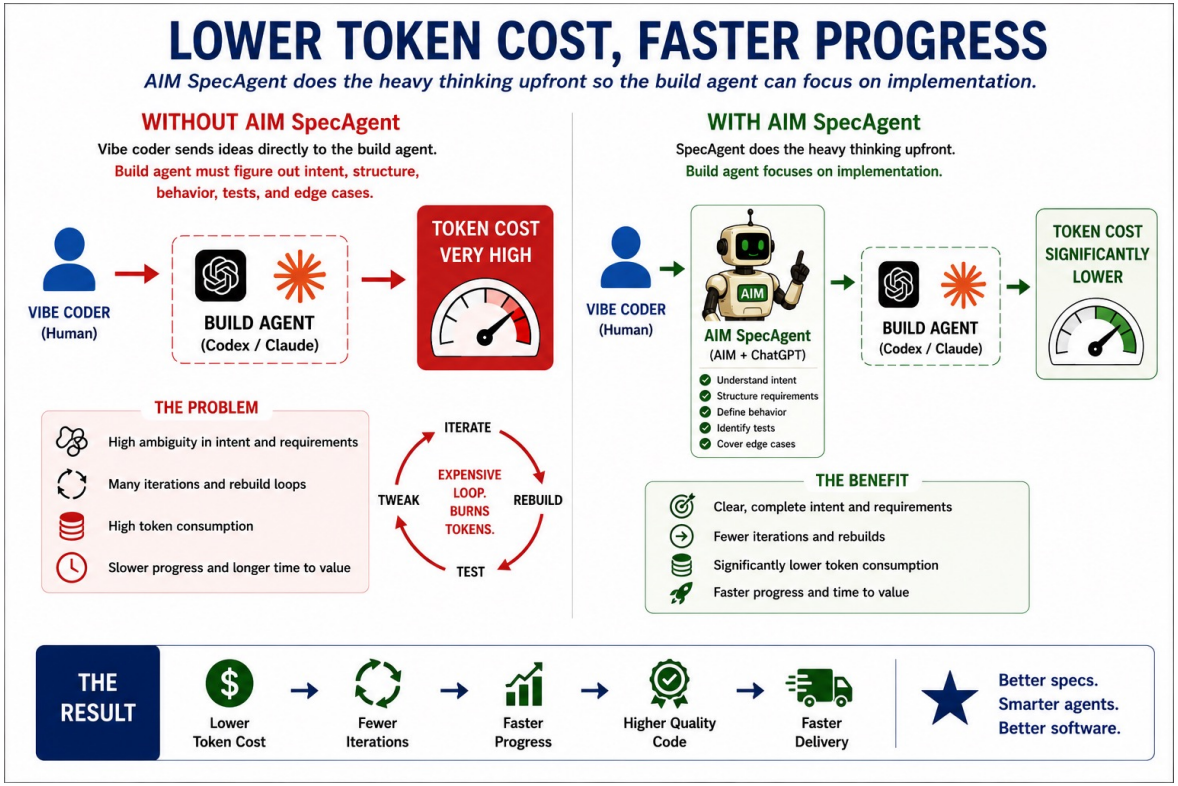
Iterating on the spec can be done without Claude Code or Codex (e.g., ChatGPT)

The spec iterations don't burn tokens

The coding agent re-writes the code a lot less than it would without the spec

Vibe coding remains very valuable, in particular, for tuning the user interface

Spec is re-generated after each coding session

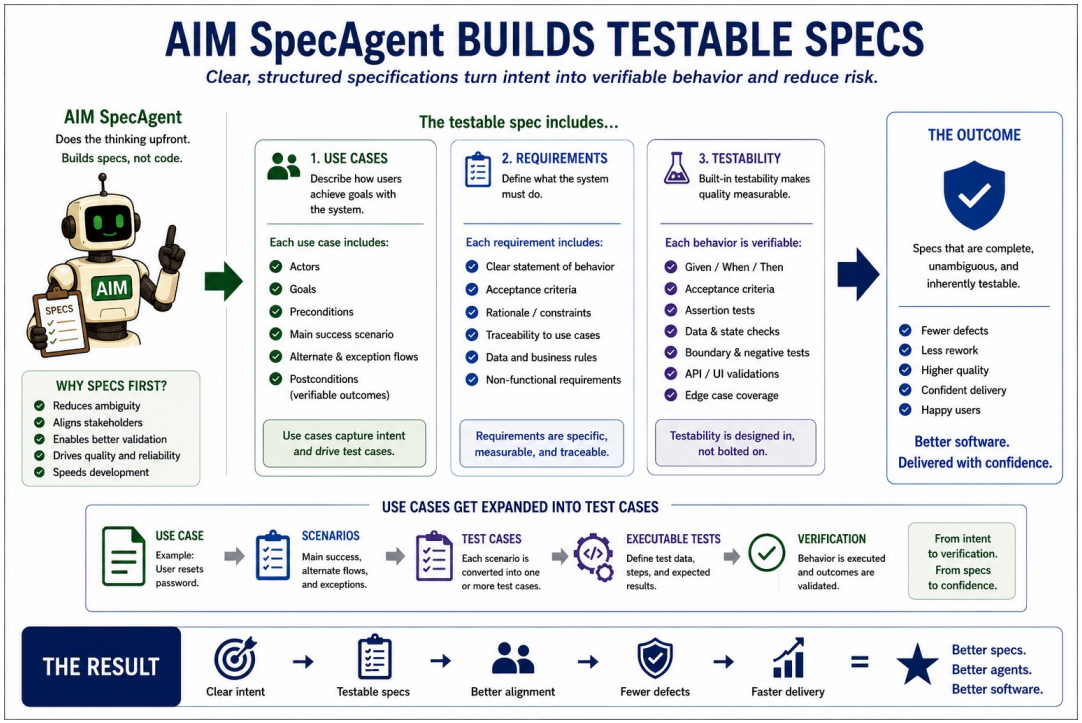


AIM Spec Agent Builds Testable Specs

We adopted test strategies from the 2003 book I wrote with Matt Stephens, notably:

- *Requirement Testing*
- *Use Case Thread Expansion*

Testing against use case threads greatly improves test coverage

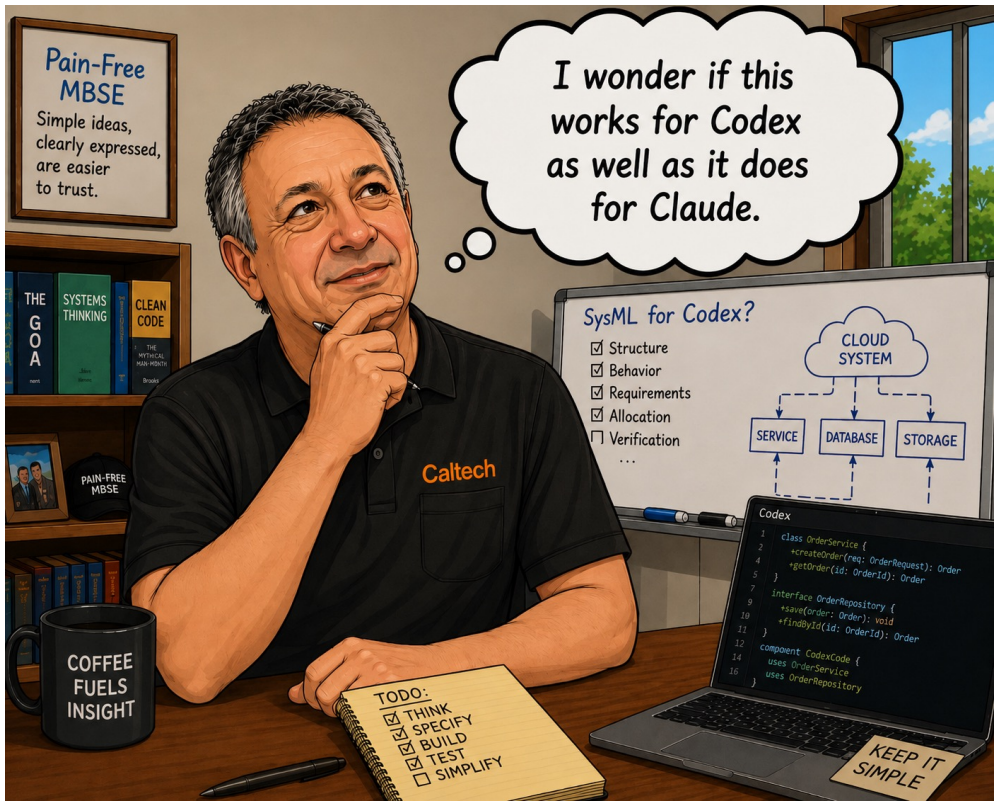


The StockPicker Experiment Begins

I had an idea back when I was in college about picking stocks that were all going up

I had never used Codex and decided to try the spec that Brian and I were using with Claude Code

It worked!!!



Claude got good results from the SysML spec

Brian and I worked together with Claude

StockPicker built in 30 minutes and ran the first time

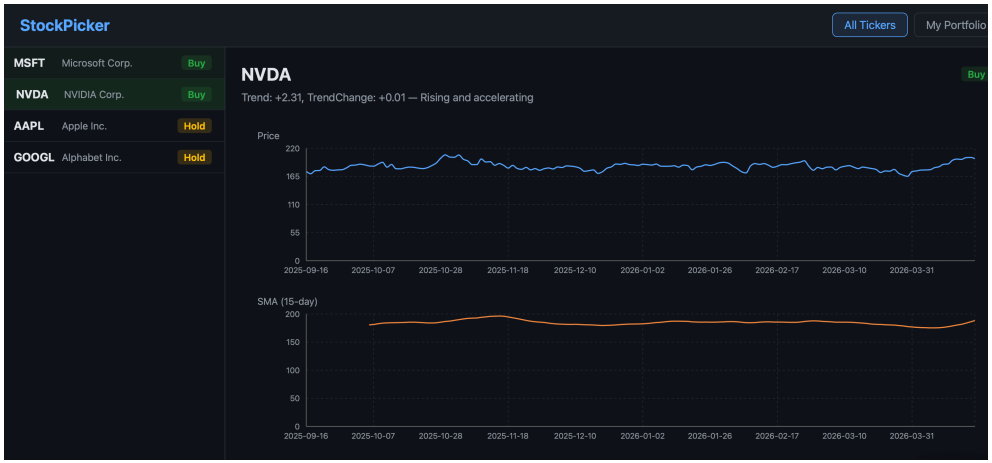
First build had a few bugs

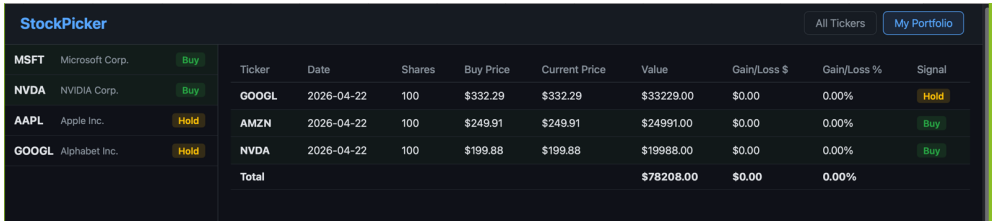
We had also forgotten to specify "web app" and online data source

We added Design Driven Testing to the spec and built again

Number of tests run increased around 4X

Next build was much more reliable

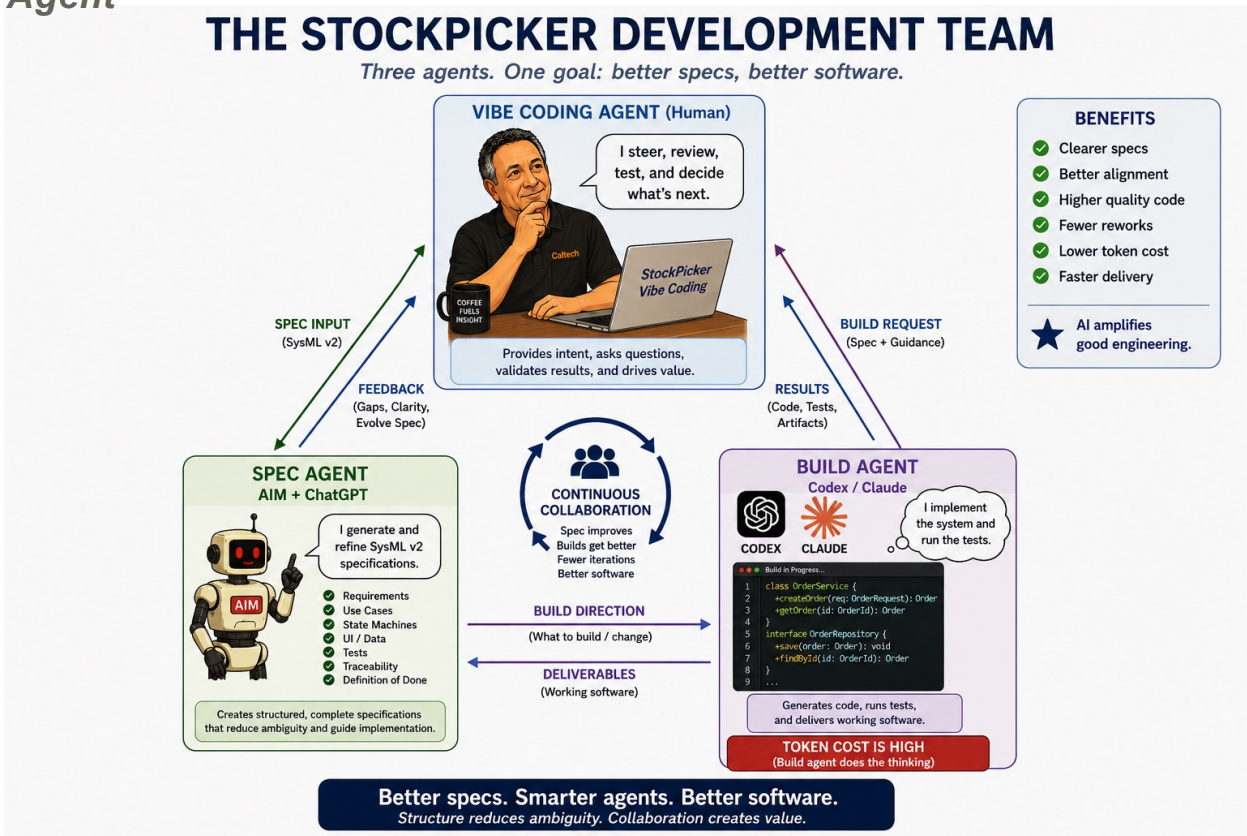




Three-Agent Collaboration

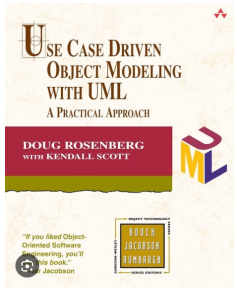
Human + AIM Spec Agent + Build Agent

I worked with both the Spec Agent (ChatGPT 5.5) and the Build Agent (Codex)



Disclaimer:

My first book featured a UML model for a portfolio management system so I was not completely inexperienced



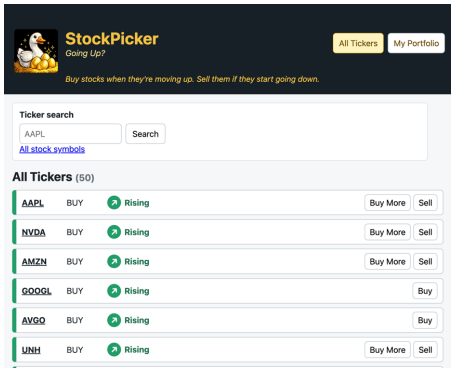
StockPicker – one human developer, and not much time spent

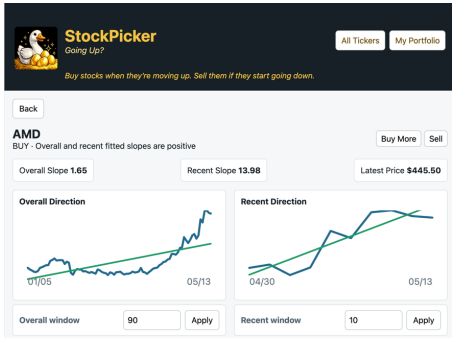
Combined Spec-Driven Development and Vibe Coding – real application; real iterations

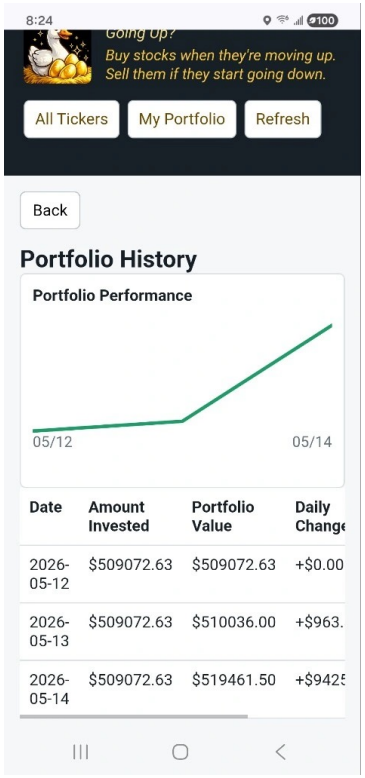
About 3 calendar weeks, part time (30 or 40 hours)

StockPicker can:

- *Access Stock Data from an online feed*
- *Build a Simulated Portfolio*
- *Examine trends for individual stocks*
- *Track Portfolio Performance over time*







Better Specs Lead to Better Outcomes

Build testable specs, get testable code

Bugs are spec errors

"If the system does not behave as intended, the specification was incomplete"

Spec gives a clear set of Requirements to test against

```
# StockPicker 3 — Consolidated Requirements Baseline

================================================
SYSTEM DESCRIPTION
================================================

StockPicker is a portfolio research and decision-support system for simulated portfolio evaluation.

The current version:
- uses daily-close market data,
- analyzes directional stock behavior,
- supports simulated portfolio construction,
- and helps users evaluate investment strategies over time.

StockPicker does not:
- connect to brokerage accounts,
- execute real financial transactions,
- or provide financial advice.

All portfolio activity in the current version is simulated.

Users remain responsible for their own investment decisions.

================================================
FUNCTIONAL REQUIREMENTS
================================================

# Platform and Architecture Requirements

ARCH_001 — Web Application
StockPicker shall operate as a browser-based web application.

ARCH_002 — Graphical User Interface
The system shall provide an interactive graphical user interface and shall not rely on command-line
interaction for normal operation.

ARCH_003 — Online Market Data Sources
The system shall obtain market data from online data services rather than from manually maintained
```

```
# Transaction History Requirements

HIST_001 — Transaction History Semantics
The visible history table shall represent simulated portfolio-event history rather than automatic
valuation history.

HIST_002 — Event Types
History shall include:
- Buy events
- Buy More events
- Sell events

HIST_003 — Buy Event History Row
A buy event shall create a history row showing:
- date
- event
- amount invested
- portfolio value after event
- snapshot delta
- cumulative profit/loss

HIST_004 — Sell Event History Row
A sell event shall create a history row showing:
- date
- event
- amount sold
- portfolio value after event
- snapshot delta
- cumulative profit/loss

HIST_005 — Automatic Valuation Rows Hidden
Automatic valuation rows created by startup refresh or market-data loading shall not appear in visible
history.

HIST_006 — No Refresh Events
Visible history shall not display refresh events.

HIST_007 — Cash-Flow Separation
Purchases and sales shall be treated as cash-flow events rather than investment gains or losses.

HIST_008 — Cumulative Profit/Loss Formula
Cumulative profit/loss shall be calculated using:
- active portfolio value
- cumulative amount sold
- cumulative amount invested

HIST_009 — Snapshot Delta Calculation
Snapshot delta shall represent the change in cumulative profit/loss relative to the previous displayed
history row.
```

Testing Use Case Threads is very effective

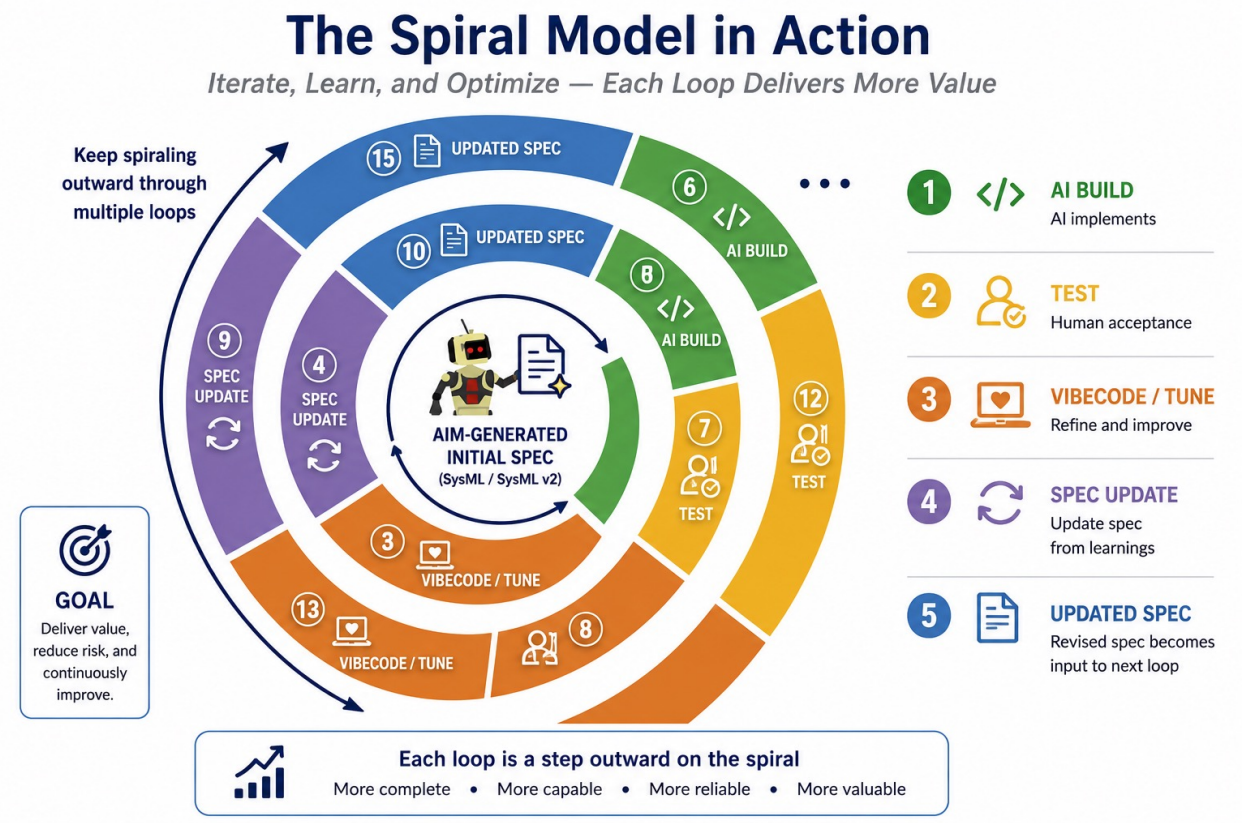
```
TEST_006 — Requirement Assertion Tests
The implementation shall include assertion tests that verify each testable functional requirement.

TEST_007 — Use Case Thread Tests
The implementation shall include use case thread tests covering:
- basic paths
- alternate paths
- exception paths

TEST_008 — Postcondition Verification
Use case thread tests shall verify stated postconditions.
```

```
================================================
USE CASE NARRATIVES
================================================

# StockPicker 3 — Use Case Narratives

## Use Case: UC_001 — View Ticker Recommendations

Actor: User

Description:
The user reviews ticker recommendations on the AllTickers page.

Basic Path:
1. System displays the AllTickers page.
2. System loads available ticker data.
3. System computes ticker recommendations using directional trend analysis.
4. System displays each ticker with its recommendation.
5. System sorts tickers by recommendation priority:
   — Buy first
   — Hold second
   — Exit last
6. System visually distinguishes recommendation categories.
7. User reviews the prioritized ticker list.
8. User selects a ticker.
9. System displays the Ticker Detail page.

Alternate Paths:

2A — No tickers available
1. System detects that no tickers are available.
2. System displays an empty-state message.
3. Use case ends.

4A — Market data unavailable for one or more tickers
1. System marks the affected ticker as unavailable.
2. System does not present unavailable data as a valid recommendation.
3. Rejoin Basic Path at Step 5.

Exception Paths:

2E — AllTickers data cannot be loaded
1. System displays a recoverable error message.
2. System does not display misleading recommendations.
3. Use case ends.

Postconditions:
- Valid ticker recommendations are visible.
- Tickers are sorted Buy, Hold, Exit.
- Unavailable ticker data is clearly marked.
```

The Spiral Model Compressed by AI

A trip around the spiral used to take weeks, months, or years

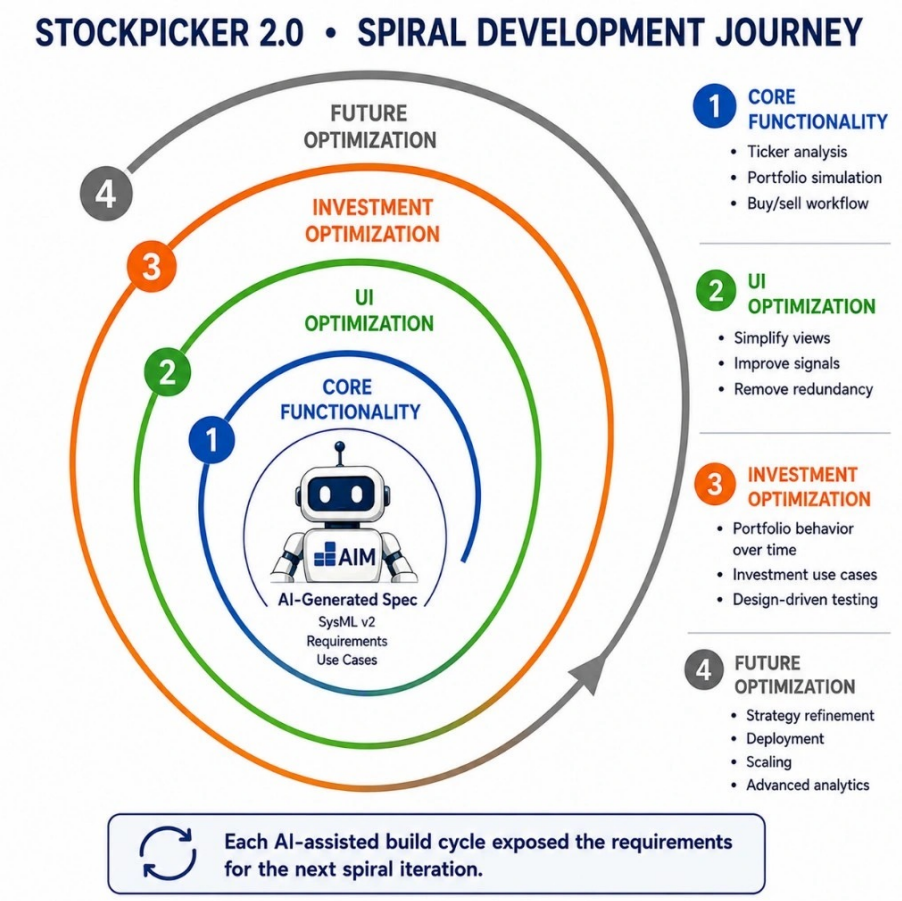
Updating the specification after each trip around the spiral enables the agentic process to scale



The Spiral Model Accelerated

Iteration time reduced *from weeks/months* ***to about an hour*** *for Spec-Build-Tweak-Update Spec*

Updating the spec at the end of each iteration is the key



One more trip around the spiral - StockPicker Test 4

Server-Side Portfolio Storage

Account Management

Multiple Portfolios per Account

```
# StockPicker Test 4 Addendum
## Accounts, Server-Side Persistence, Multiple Portfolios, and
Persistent Data Model

================================================================
========
OVERVIEW
================================================================
========

This addendum extends the StockPicker application from a single-
browser-session
prototype into a persistent multi-portfolio application with
authenticated
user accounts.

Primary goals:
- Eliminate accidental portfolio/history loss
- Move persistence responsibility from browser local storage to the
server
- Support multiple named portfolios per account
- Preserve experimental and comparative investment strategies
- Improve long-term usability and trustworthiness
- Establish persistence architecture for future subscription
deployment
```

Spec Agent generates new Requirements and Use Cases

```
--------------------------------------------------------------------------------
MULTIPLE PORTFOLIO MANAGEMENT
--------------------------------------------------------------------------------

### REQ-410 – Multiple Portfolios
An authenticated user shall be able to own one or more named
portfolios.

### REQ-411 – Create Portfolio
The system shall allow the user to create a new named portfolio.

### REQ-412 – Rename Portfolio
The system shall allow the user to rename an existing portfolio.

### REQ-413 – Delete Portfolio
The system shall allow the user to delete a portfolio.

Deletion shall require explicit user confirmation before permanent
removal
or archival.

### REQ-414 – Active Portfolio Selection
The user shall be able to select one active portfolio at a time.

### REQ-415 – Portfolio Switching
Switching portfolios shall update:
- positions
- transaction history
```

```
--------------------------------------------------------------------------------
USE CASE: UC-417 SWITCH ACTIVE PORTFOLIO
--------------------------------------------------------------------------------

Actor:
Authenticated User

Description:
The user switches from one active portfolio to another active
portfolio.

Basic Path:
1. User selects a different portfolio.
2. System validates ownership.
3. System saves pending changes to the current portfolio.
4. System retrieves selected portfolio data.
5. System restores selected portfolio state.
6. System updates active portfolio reference.
7. System refreshes:
   - charts
   - positions
   - transaction history
   - performance history
   - portfolio settings
8. System displays updated dashboard.

Exception Paths:
```

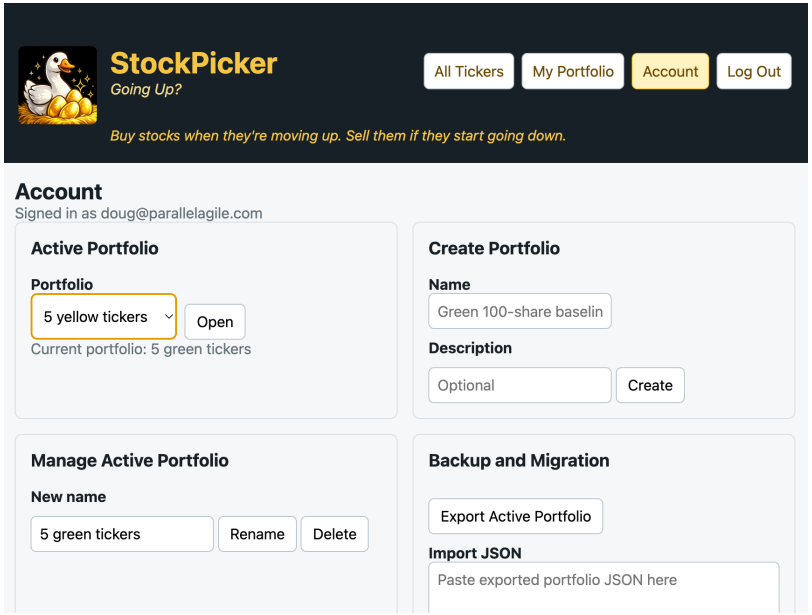
From plan to build in under an hour

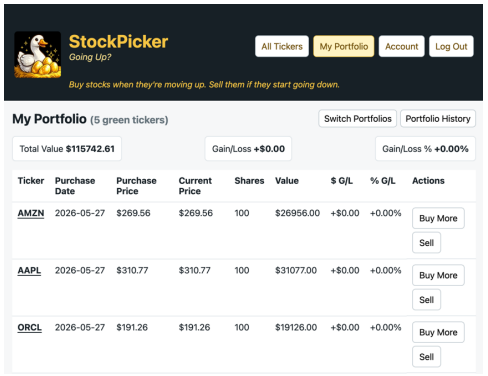
This is a fairly large jump, so I'm going to make Test 4 usable in slices. I'm starting with a server API that can own accounts, sessions, and portfolios, using server-side JSON persistence locally so we avoid adding npm dependencies. The cPanel Python deployment can map the same API shape onto Python's built-in persistence later.

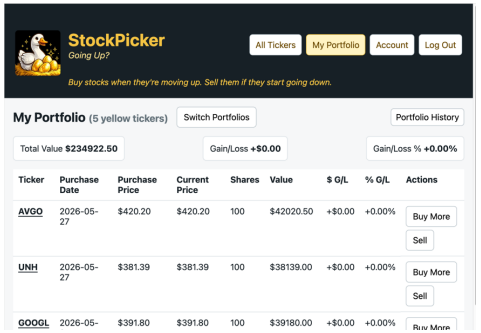
Created 1 file, edited 1 file

The first server-side account/persistence API is in place locally: register/login/session cookies, portfolio CRUD, active portfolio, import/export, password change/reset stub, and ownership checks. I'm going to add focused domain/API tests now before wiring the UI, so failures stay small and readable.

Build Agent implements the new Requirements and Use Cases

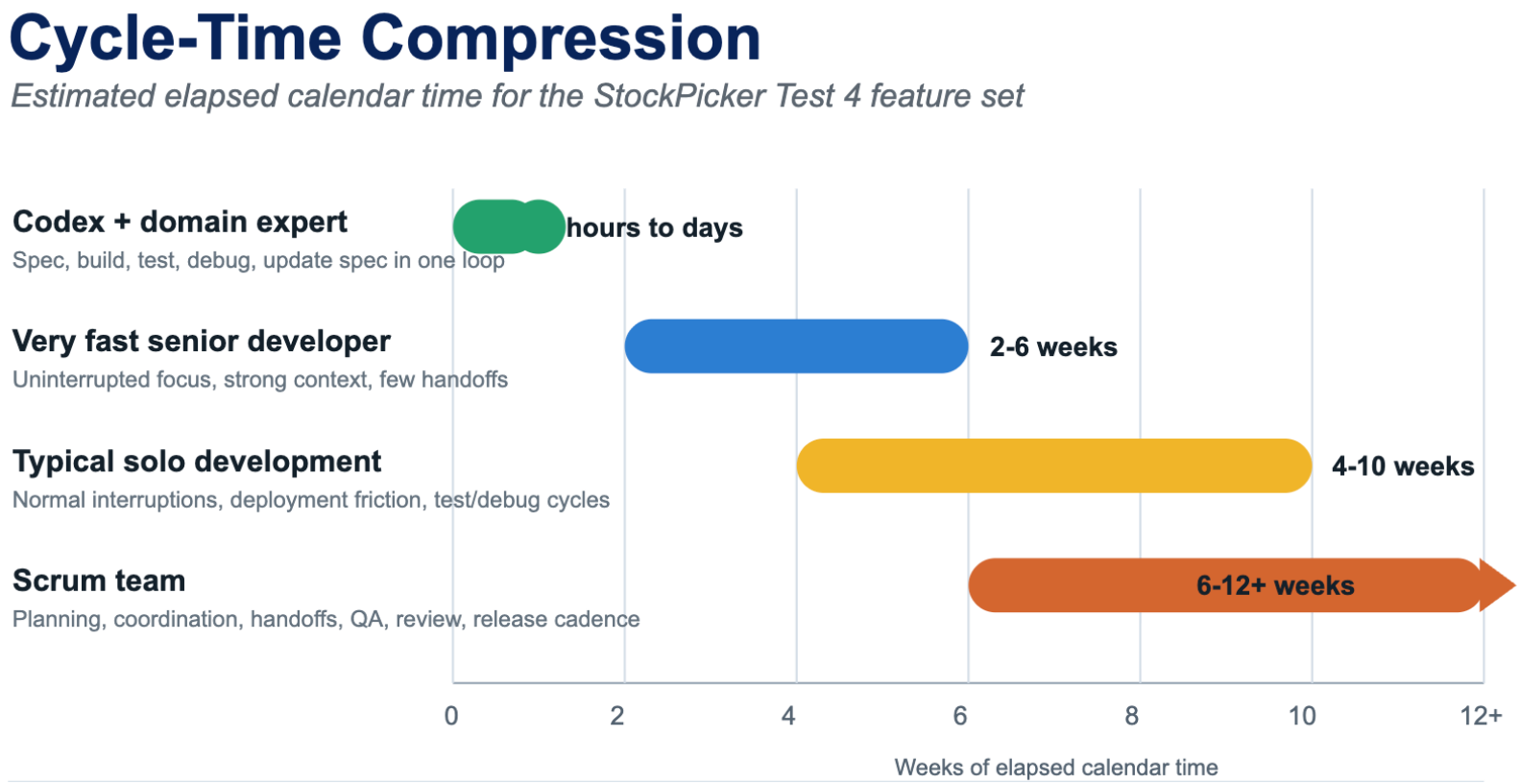






It literally builds in a few minutes when given the spec!

Time Savings is Enormous



AI Coding Agent boosts Productivity – without burning many tokens

What Did Codex Do?

From SysML-style requirements to a deployed, testable StockPicker web app

1 Interpreted The Spec

Parsed SysML-style requirements
Identified assumptions and gaps
Mapped requirements into app structure and tests

2 Built The Product

Built ticker, portfolio, and detail views
Implemented recommendation logic and charts
Added buy, buy-more, sell, and partial-sell flows
Added valuation, gain/loss, and history behavior

3 Connected Real Data

Added server-side market-data fetching
Added cache and refresh behavior
Added safeguards for stale or failed data

4 Added Persistence And Accounts

Added registration, login, logout, and sessions
Added multiple named portfolios per account
Moved portfolio state to server-side persistence
Added export/import backup support

5 Tested And Deployed

Generated regression and use-case tests
Fixed bugs from live exploratory testing
Packaged for cPanel/GoDaddy deployment
Debugged deployment and startup-file issues

6 Fed Learning Back

Captured discovered behavior as requirements
Updated spec addenda for future builds

Moving the spec activities out of the coding agent minimized token use

What Did The Spec Agent Do?

No tokens used beyond my $20/month ChatGPT subscription.

1 Structured The Idea

Turned product intuition into explicit requirements
Organized intent into features and constraints
Kept the SysML-style spec as source of truth

2 Modeled The System

Defined app structure and system responsibilities
Captured portfolio, position, and transaction concepts
Added accounts, sessions, portfolios, and persistence

3 Defined Behavior

Wrote use-case narratives and workflows
Added preconditions, postconditions, and alternatives
Clarified market dates, snapshots, and isolation

4 Guided Testing

Turned expected behavior into testable requirements
Added guidance for account and persistence tests
Identified edge cases for regression coverage

5 Updated The Baseline

Integrated lessons from live app testing
Reassembled scattered addenda into a baseline
Kept future builds aligned with what was learned

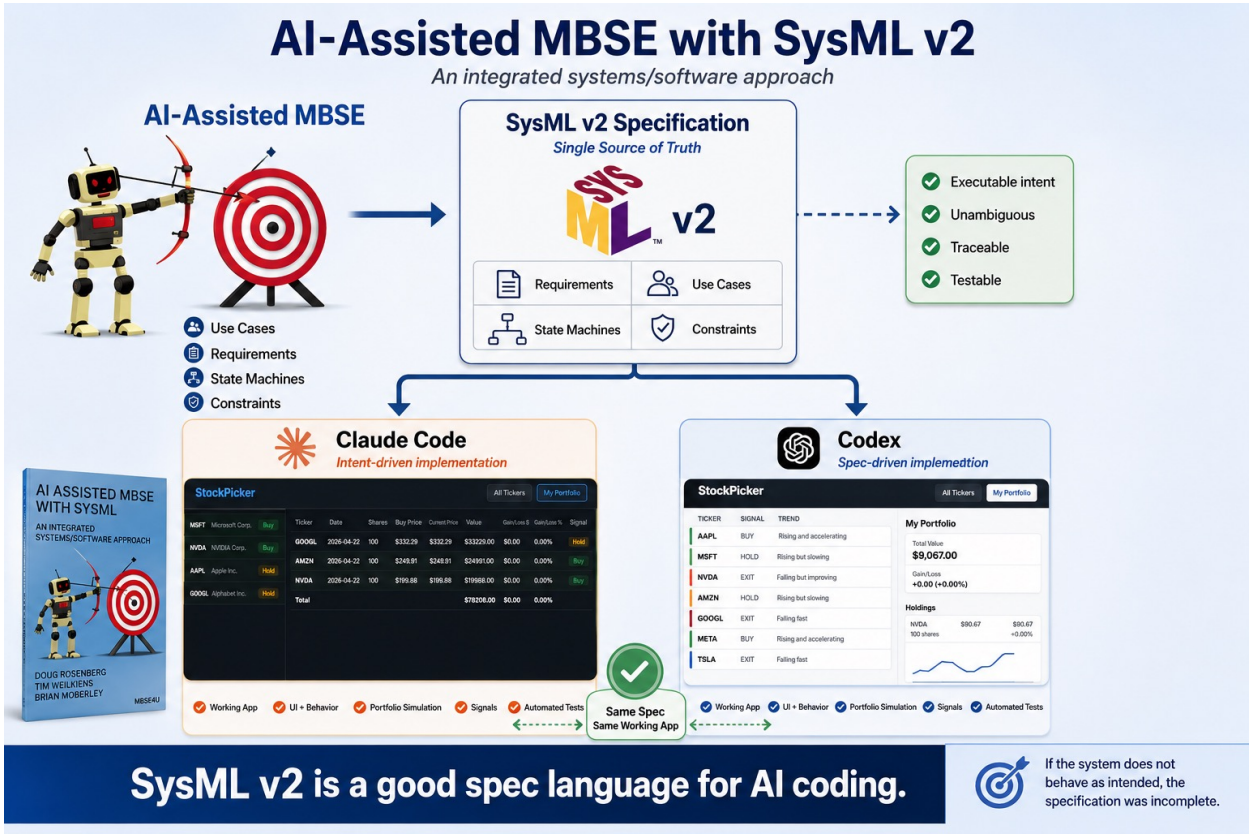
6 Reduced Build Risk

Separated brainstorming from build instructions
Made ambiguous behavior visible before coding
Gave the build agent a clearer target

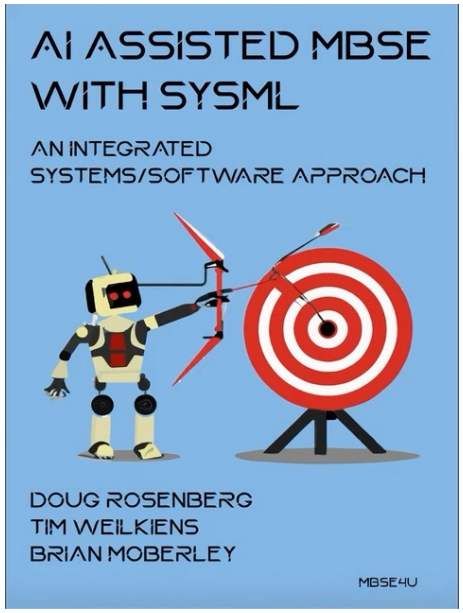
Results

Same structured specification → similar working systems

SysML v2 turns out to be a good spec language for coding agents



AIM enables rapid spec development with minimal token usage



Traditional Development:

Minimize spec changes because they are expensive

AI-Assisted MBSE:

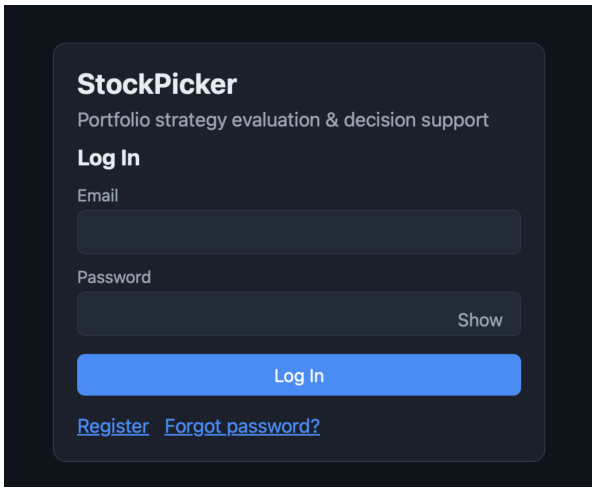
Maximize spec refinement because it is cheap

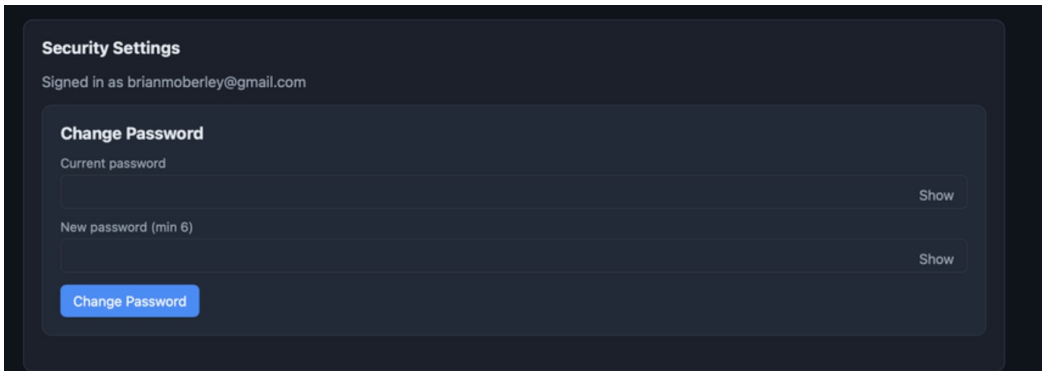
Late Last Night – We fed the Stock Picker 8 spec back to Claude

Spec is a 42-page PDF

```
STOCKPICKER SPECIFICATION BASELINE v8.0

TABLE OF CONTENTS

1. Overview
   1.1 Purpose
   1.2 Scope
   1.3 Design Philosophy
   1.4 Authoritative Data Model
   1.5 Testing Strategy

2. Glossary

3. Requirements
   3.1 Account Management
   3.2 Portfolio Management
   3.3 Portfolio History and Performance
   3.4 SQLite Persistence
   3.5 Security
   3.6 User Interface Requirements

4. Use Cases
   4.1 Use Case Inventory
   4.2 Use Case Narratives


5. State Machines
   SM-002 Market Data Cache Lifecycle
   SM-003 Portfolio Position Lifecycle
   SM-004 Overall UI Flow

6. Testing Rules
```

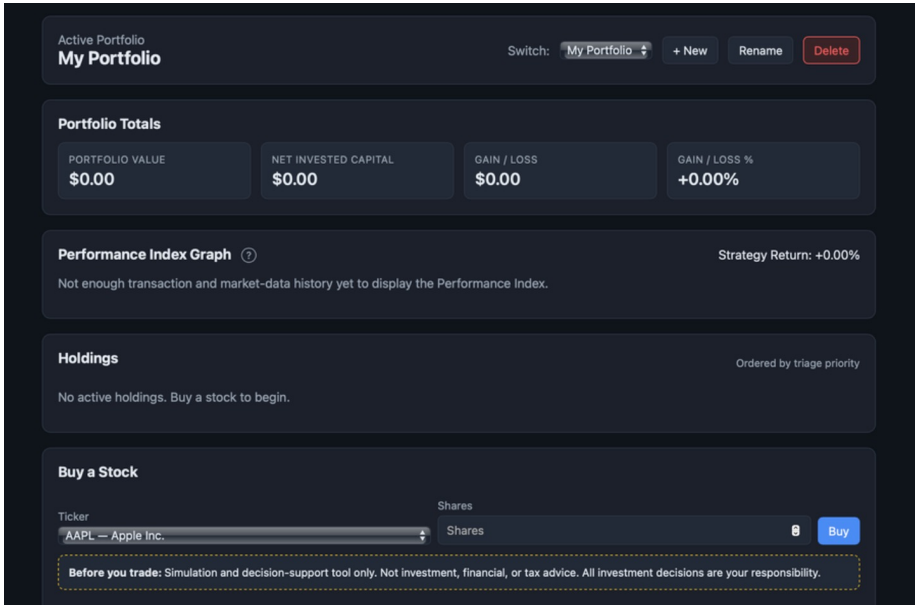
Claude took 30 minutes to build it – including Account Management

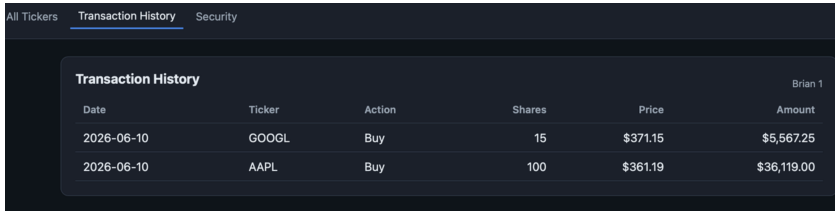




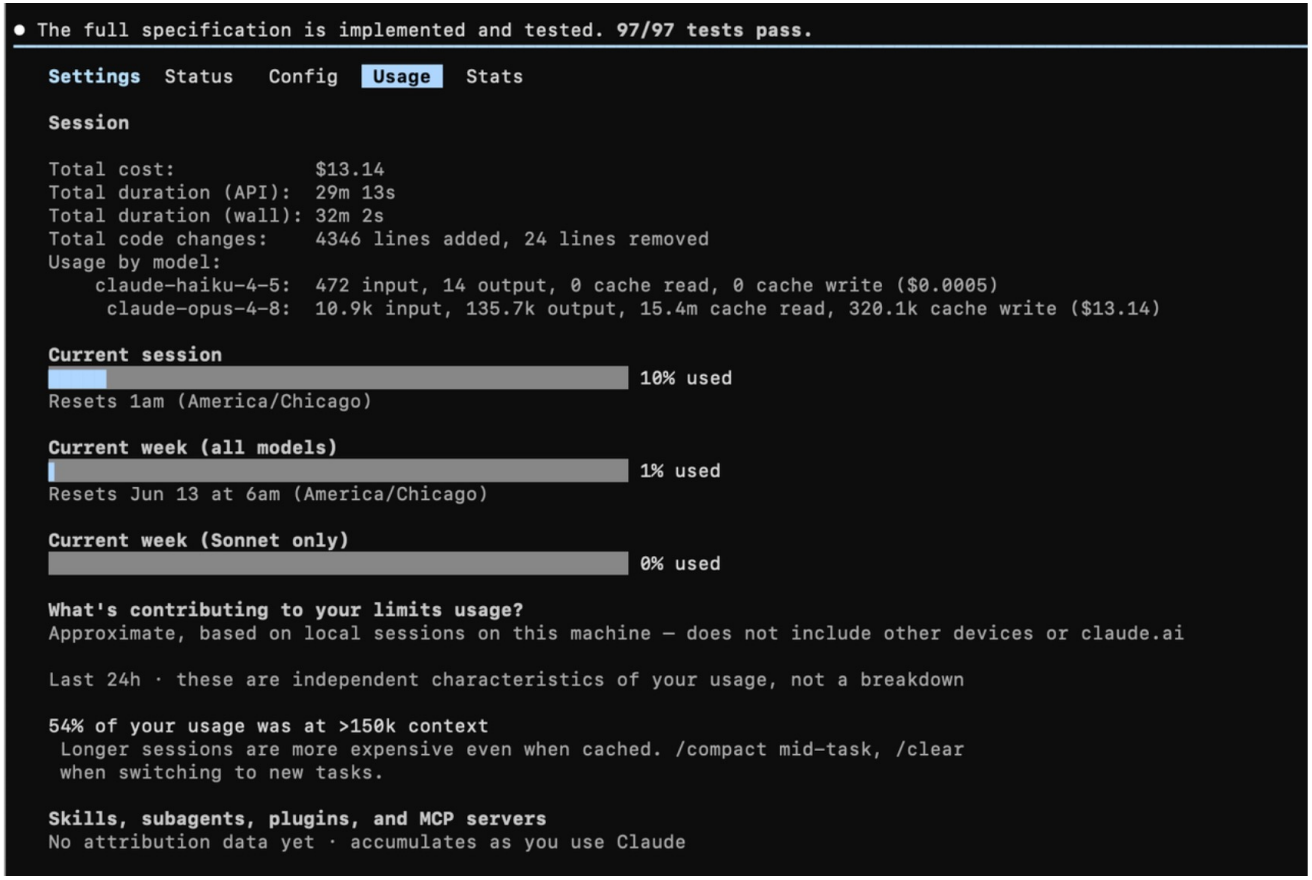
Full disclosure: Actually, it took 32 minutes and 2 seconds to build and test

Including enhancements to Portfolio Management, Transactions, etc.

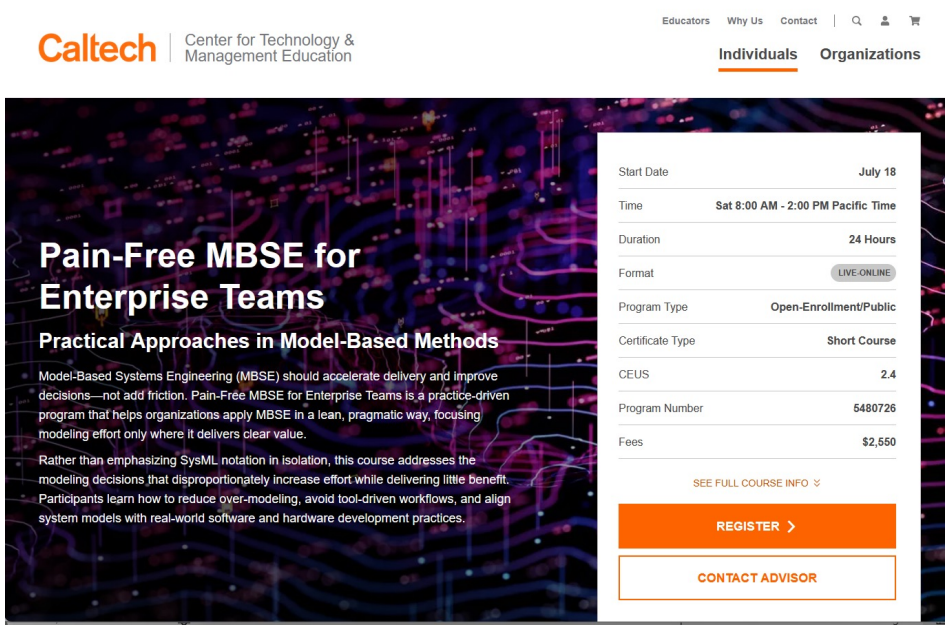


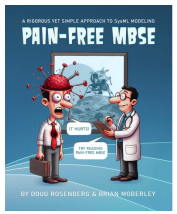


Total cost to build from spec $13. Many tokens saved by spec agent.



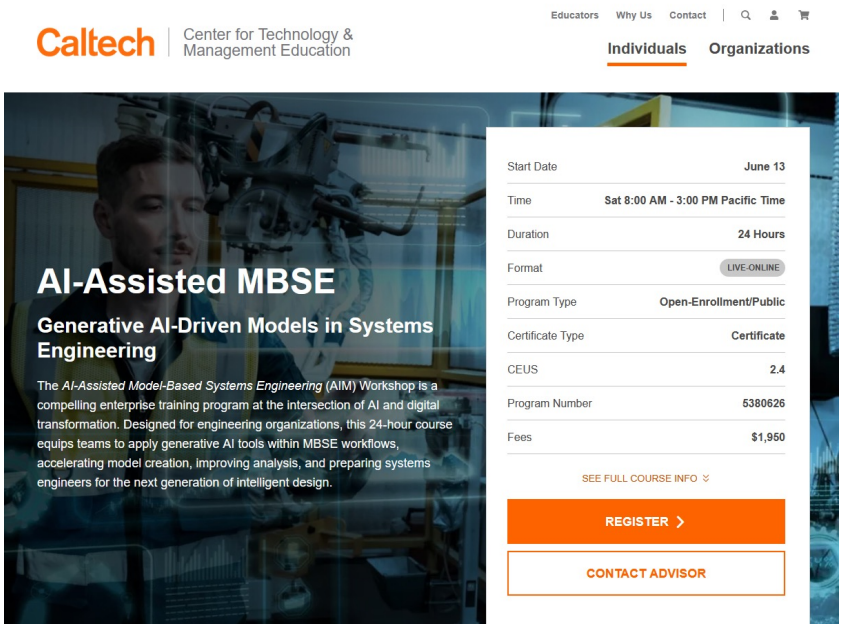
Public Courses from Caltech CMTE

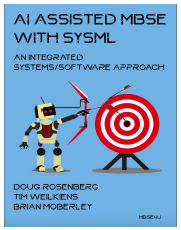




https://ctme.caltech.edu/pain-free-mbse.html

Four 6-hour sessions, starts Saturday, July 18





https://ctme.caltech.edu/ai-assisted-mbse.html

Four 6-hour sessions, starts Saturday, June 13