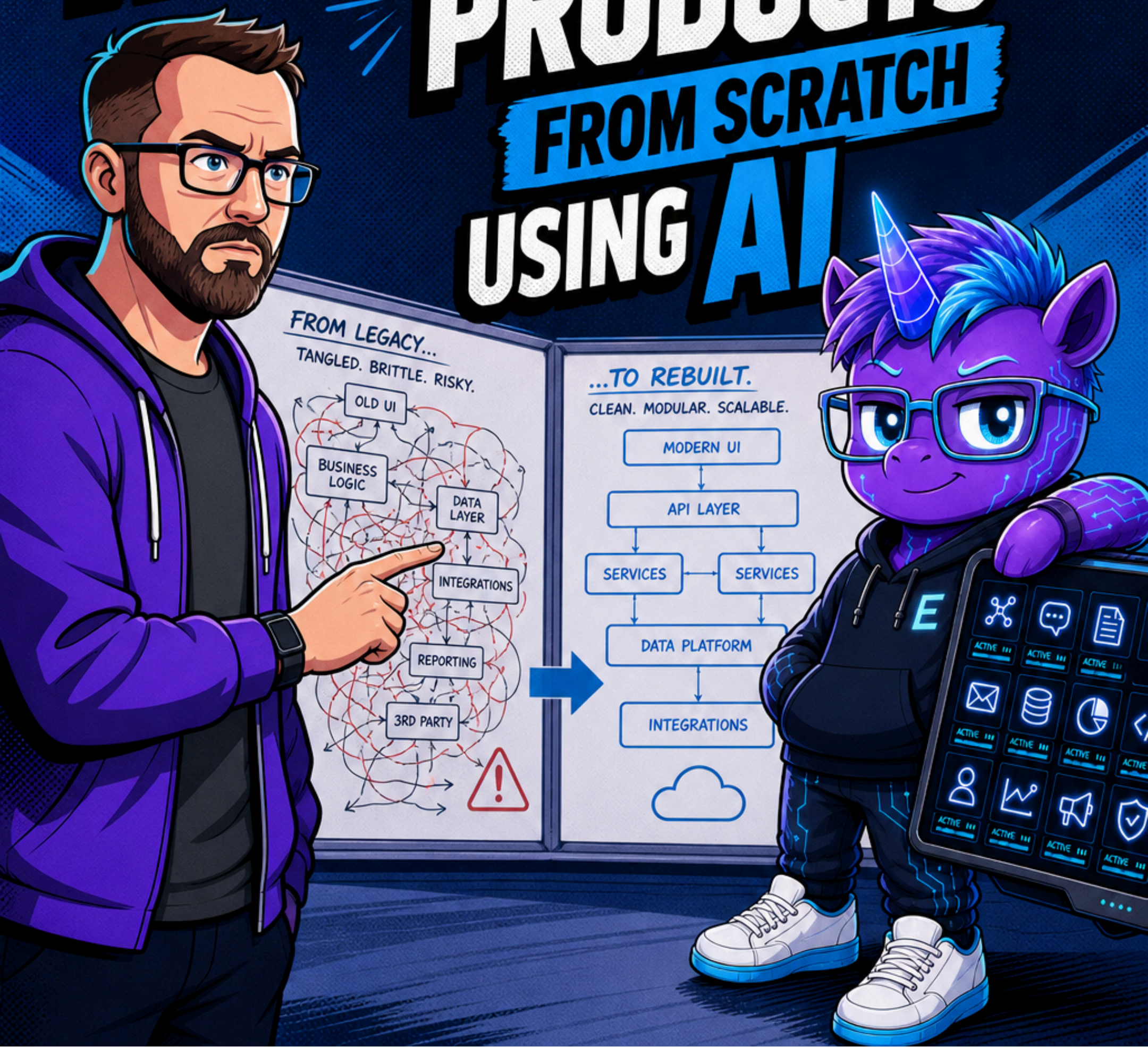
HOW TO REBUILD LEGACY PRODUCTS FROM SCRATCH USING AI



THE REAL PROBLEM WITH LEGACY SYSTEMS ISN'T THE CODE

IT'S THE KNOWLEDGE THAT'S TRAPPED INSIDE IT.

Decades of business rules, regulatory accommodations, and edge-case fixes live in codebases that have never been documented. The original developers are gone. The specs — if they ever existed — don't match what's actually running. And the system itself has become the only reliable source of truth about what the business actually does.

***That's the problem AI can* genuinely help you solve.**
Not by rewriting your COBOL, but by reading it.

LEGACY SYSTEMS CONSUME A DISPROPORTIONATE SHARE OF IT BUDGETS
and are frequently cited as barriers to AI adoption.

AI-AUGMENTED MODERNISATION CAN CUT TIMELINES BY
40–50%
by automating code translation, dependency mapping, documentation, and QA.

These are projected averages from specific engagements, not universal guarantees, but the direction of the evidence is consistent. The tools exist. The question is whether your team is using them **strategically** or just pointing them at the problem and hoping for output.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  LEGACY-PROC.
AUTHOR.      UNKNOWN.

PROCEDURE DIVISION.
0010-MAIN
  PERFORM 1000-INIT.
  PERFORM 2000-PROCESS.
  PERFORM 9000-EXIT.
  STOP RUN.

1000-INIT.
  IF WS-DATE = SPACE.
    MOVE 'Y' TO WS-FLAG
  ELSE
    PERFORM 1500-LOAD.

1500-LOAD.
  PERFORM VARYING I
    FROM 1 BY 1
    UNTIL I > WS-LIMIT
    IF WS-TABLE(I)
      PERFORM 3000-UPDATE.
  ...
3000-UPDATE.
  COMPUTE WS-TOTAL
    WS-TOTAL + WS-AMOUNT
  IF WS-TOTAL > 999999
    MOVE 'Y' TO WS-OVER
    PERFORM 4000-WRITE.

4000-WRITE.
    WRITE WS-RECORD
      INVALID KEY
        PERFORM 8000-ERROR.
    END-WRITE.
...
9000-EXIT.
  EXIT.
```

BUSINESS RULES (UNDECLARED)

REGULATORY ACCOMMODATIONS (HISTORICAL)

EDGE-CASE FIXES (UNKNOWN)

ONLY RELIABLE SOURCE OF TRUTH

STEP ONE:
USE AI TO READ THE CODEBASE BEFORE YOU TOUCH IT

BEFORE ANY REBUILD DISCUSSION, YOU NEED A COMPLETE PICTURE OF WHAT THE EXISTING SYSTEM ACTUALLY DOES.
NOT WHAT DOCUMENTATION SAYS. WHAT IT *DOES*.

MODERN LLMS ARE GENUINELY CAPABLE HERE.

TOOLS LIKE GITHUB COPILOT CHAT, CURSOR, AND AMAZON Q LET YOU ASK SPECIFIC QUESTIONS ABOUT FUNCTIONS, MODULES, AND BUSINESS LOGIC UNDOCUMENTED FOR DECADES.

LEGACY CODE

```
412  function calcPremium(data) {
413      var base = 0;
414      if (data.type == "A") {
415          base = data.amount * 0.0125;
416      } else if (data.type == "B") {
417          base = data.amount * 0.018;
418      } else {
419          base = data.amount * 0.021;
420      }
421
422      if (data.region == "EU") {
423          base = base * 1.17;
424      } else if (data.region == "APAC") {
425          base = base * 1.08;
426      }
426
426      if (data.vip == true) {
427          base = base * 0.92;
426      }
425
425      return Math.round(base * 100) / 100;
428  }
423
428  function getEligibleDiscounts(id) {
429      var results = [];
427      var q = "SELECT * FROM discounts
488          WHERE customer_id = '" + id;
426      var rs = db.execute(q);
426      while (rs.next()) {
429          // business rules hidden in here
427          if (rs.get("active") == 'Y' &&
              rs.get("start_date") <= today() &&
              rs.get("end_date") >= today()) {
              results.push(rs.get("code"));
          }
      }
      return results;
  }
  }
5+    // 3,842 more lines...
```

AI ANALYSIS (COPILOT CHAT)

SYSTEM SUMMARY

This codebase is a legacy pricing and discount engine with tightly coupled business logic, direct DB access, and implicit rules scattered across modules. High risk of side effects.

KEY FUNCTIONS

- calcPremium(data)
  - Calculates premium based on type, region, and VIP status
  - Contains hard-coded multipliers
  - Rounds to 2 decimal places
  - Used in 27 places
- getEligibleDiscounts(id)
  - Retrieves active discounts for a customer
  - Business rules embedded in SQL + code
  - Date logic uses server time
  - Used in 15 places

DEPENDENCIES

- Direct DB access (db.execute)
- Global config object (config.*)
- Utility libs: dateUtils, mathExt

RISK AREAS

- ⚠ Hard-coded rates and rules
- ⚠ Implicit date/time assumptions
- ⚠ No input validation
- ⚠ Shared mutable state

SUGGESTED NEXT STEPS

- ✓ Map call graph for pricing flow
- ✓ Identify test coverage gaps
- ✓ Extract business rules to config
- ✓ Add regression tests before change

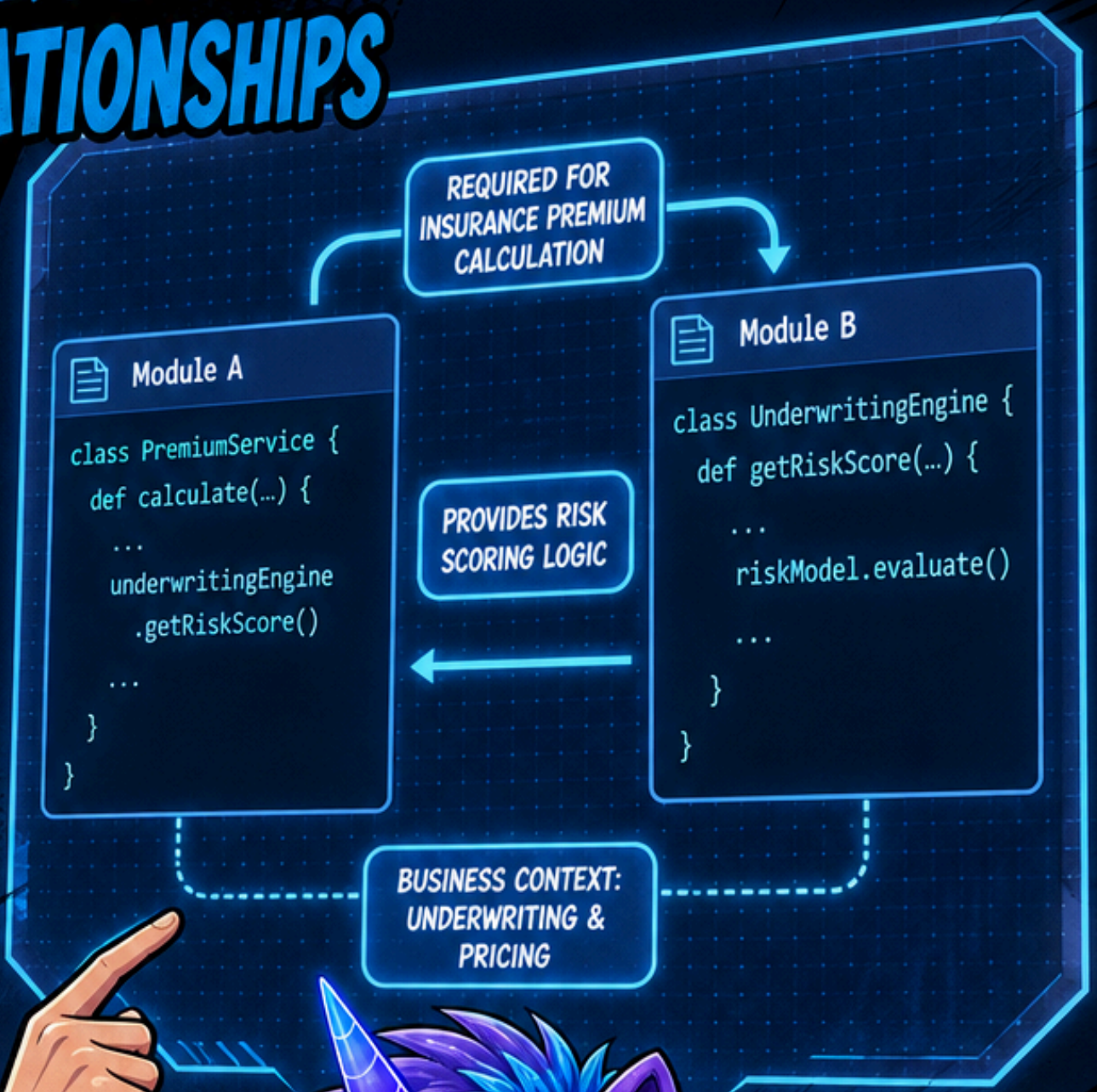
Would you like a call graph or date diagram for this module?

BEYOND CALL MAPPING: SEMANTIC UNDERSTANDING OF MODULE RELATIONSHIPS

AI-POWERED ANALYSIS MOVES PAST SIMPLE DEPENDENCY TRACING. INSTEAD OF "MODULE A CALLS MODULE B," YOU GET:

"MODULE A NEEDS MODULE B FOR INSURANCE PREMIUM CALCULATION."

THAT CONTEXTUAL INTELLIGENCE — UNDERSTANDING **WHY** MODULES CONNECT, NOT JUST THAT THEY CONNECT — IS WHAT REVEALS THE REAL BUSINESS LOGIC BURIED IN LEGACY SYSTEMS.



FOR COBOL-HEAVY ENVIRONMENTS, USE PURPOSE-BUILT TOOLING

General-purpose models underperform on mainframe code. Vendors including **IBM** (watsonx Code Assistant for Z), **Broadcom**, and **OpenText** offer tools fine-tuned for COBOL — delivering code summaries, business-intent abstraction, and Q&A for legacy comprehension, module refactoring, and variable renaming.

```cobol
01   IDENTIFICATION DIVISION.
02   PROGRAM-ID. PAYROLL.
03   AUTHOR.  FINANCE-DEPT.
04   DATA DIVISION.
05   WORKING-STORAGE SECTION.
06   01  WS-EMPLOYEE-RECORD.
07     05 WS-EMP-ID       PIC 9(05).
08     05 WS-NAME         PIC X(30).
09     05 WS-SALARY       PIC 9(07)V99.
10   PROCEDURE DIVISION.
11     PERFORM CALCULATE-SALARY.
12     PERFORM PRINT-REPORT.
13     STOP RUN.
```

IBM watsonx Code Assistant for Z
- Code Summaries
- Business-Intent Abstraction
- Q&A for Legacy Comprehension

BROADCOM
- Code Summaries
- Business-Intent Abstraction
- Q&A for Legacy Comprehension

opentext™
- Code Summaries
- Business-Intent Abstraction
- Q&A for Legacy Comprehension

THE RESULT:

AN INVENTORY, NOT A REWRITE

AI-assisted analysis can process tens of thousands of lines in hours — not the weeks manual archaeology requires.

The output is a map. Every module annotated with its business purpose, its dependencies, and the rules it enforces. That inventory is what makes the next step possible.

THE TRAP TEAMS FALL INTO

*Refactoring incrementally or translating line by line preserves the legacy architecture while changing the technology. You inherit the same structural problems, same coupling, same implicit assumptions — **just in modern code.***

EXTRACT VALUE, NOT CODE

USE AI TO EXTRACT **BUSINESS LOGIC** FROM THE CODEBASE AND STORE IT AS HUMAN-READABLE, EXECUTABLE **SPECIFICATIONS.**

THEN REIMPLEMENT FROM THOSE SPECS ON A **MODERN STACK** — CLEANLY, WITH NO HISTORICAL BAGGAGE CARRIED FORWARD.

BUSINESS LOGIC SPEC (v1)

ENTITIES
- Customer
- Order
- Subscription
- Invoice

RULES
- Active subscription required
- Discount applies to first order
- Invoice due 30 days from issue

WORKFLOWS
- Order → Validate → Invoice → Collect

CONSTRAINTS
- One active subscription per customer
- No invoicing without valid order

INVARIANTS
- Order total ≥ 0
- Invoice total = Order total − Discount

EVENTS
- OrderCreated
- InvoiceIssued
- PaymentReceived

REIMPLEMENT ON MODERN STACK

Web App → API Gateway → Services → Data Store

Auth · Events · Observability · Testing

THE EXTRACTION PROCESS

1

ANNOTATED LOGIC DOCS

- AI summaries of each module
- Inputs, outputs, rules
- Exceptions
- Contradictions flagged

2

STRUCTURED SPECIFICATIONS

- Plain language moves into formal spec format
- Verifying required functionality

3

```
Feature: Checkout
  Scenario: Successful
    Given user has items
    When user checks out
    Then order is confirmed
```

BDD SCENARIOS

- Gherkin scenarios become executable specs
- Bridging business requirements and implementation

BUILD CLEAN
FROM A CLEAN FOUNDATION

- Choose your stack based on current engineering merit — not what legacy used.
- Build feature by feature.
- Nothing ships until its BDD scenario passes.
- Failing scenarios in your pipeline immediately signal when behaviour drifts.

Feature: User Login

Scenario: Successful login

Given a registered user
When the user enters valid credentials
Then the user is authenticated
And redirected to dashboard

PIPELINE: RUN #2487

SCENARIO: SUCCESSFUL LOGIN
PASS

SCENARIO: INVALID PASSWORD
PASS

SCENARIO: ACCOUNT LOCKED
PASS

SCENARIO: PASSWORD RESET
PASS

CLEAN FOUNDATION.
PREDICTABLE DELIVERY.
NO SURPRISES.

1. GENERATE ANNOTATED LOGIC DOCUMENTS.

RUN YOUR AI ANALYSIS TOOLING ACROSS THE FULL CODEBASE AND PRODUCE MODULE-BY-MODULE SUMMARIES OF WHAT EACH COMPONENT DOES IN PLAIN LANGUAGE.

MODULE: PAYMENT_PROCESSOR

PURPOSE:
Handles customer payments, validates transactions, and updates account balance.

INPUTS:
- payment_request
- user_id
- amount
- currency

OUTPUTS:
- transaction_id
- status
- updated_balance

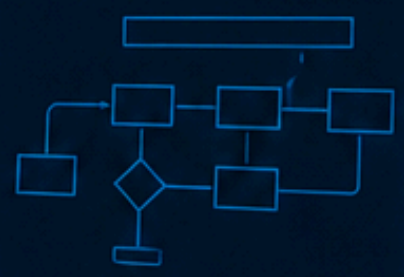
VALIDATION RULES:
- amount > 0
- currency is supported.
- user account is active

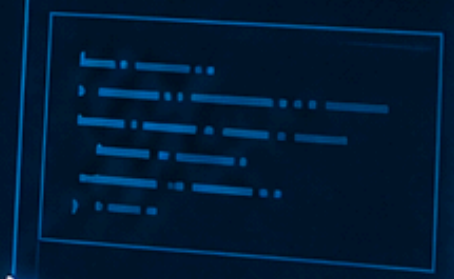
EXCEPTIONS:
- InsufficientFundsError
- InvalidCurrencyError
- PaymentGatewayError

DOMAIN CALCULATIONS:
- Fees = amount x 2.9% + $0.30
- Net Amount = amount - Fees

FLAGGED: CONTRADICTORY LOGIC
Inconsistent fee calculation in PRICING_SERVICE module.
- Risk: Incorrect charges

/CODEBASE
- /AUTH_SERVICE
- /USER_SERVICE
- /PAYMENT_PROCESSOR
- /PRICING_SERVICE
- /NOTIFICATION_SERVICE
- /REPORTING_SERVICE
- /INVENTORY_SERVICE
- ...





FLAG CONTRADICTORY OR DUPLICATED LOGIC— THOSE ARE YOUR HIGHEST-RISK AREAS.

2. Translate annotated logic into structured specifications.

Move the plain-language summaries into a formal spec format.

By extracting specs from legacy code, teams can verify that modernisation efforts preserve required functionality while eliminating undocumented behaviours.

The distinction is that traditional specs are read by humans, while modern executable specs act as validation gates.

This is where BDD (behaviour-driven development) earns its place in the process.

3. WRITE BDD SCENARIOS FROM EACH SPECIFICATION.

```
Feature: Fixed-rate loan interest calculation

  Scenario: Monthly interest on a standard
    residential mortgage
    Given a loan principal of $500,000
    And an annual interest rate of 6.5%
    And a term of 30 years
    When the monthly repayment is calculated
    Then the monthly repayment amount should
      be $3,160.34
    And the total interest paid over the
      term should be $637,722.40

  Scenario: Early repayment penalty applies
    within fixed-rate period
    Given a loan with 3 years remaining in
      a 5-year fixed period
    And an early repayment fee of 1.5% of
      outstanding principal
    When the borrower requests full
      early repayment
    Then an early repayment fee should be
      calculated and disclosed
    And the payoff amount should include
      principal, accrued interest, and the fee
```

4. CHOOSE A MODERN STACK BASED ON WHAT THE SPECS DEMAND, NOT WHAT THE LEGACY SYSTEM USED.

YOU'RE NO LONGER CONSTRAINED BY THE ORIGINAL ARCHITECTURE. THE SPECS DEFINE THE BEHAVIOUR. THE STACK SHOULD BE CHOSEN ON CURRENT ENGINEERING MERIT: MAINTAINABILITY, PERFORMANCE, ECOSYSTEM MATURITY, AND YOUR TEAM'S CAPABILITY.

STACK COMPARISON

CRITERIA	React + Node.js	Next.js + Node.js	Spring Boot	Go + Fiber
MAINTAINABILITY	✓	✓	✓	✓
PERFORMANCE	✓	✓	–	✓
ECOSYSTEM MATURITY	✓	✓	✓	✓
TEAM CAPABILITY	–	✓	✓	✓
OVERALL FIT	★★★★☆	★★★★★	★★★☆☆	★★★☆☆

5. REIMPLEMENT AGAINST THE SPECS.

BUILD THE NEW SYSTEM FEATURE BY FEATURE

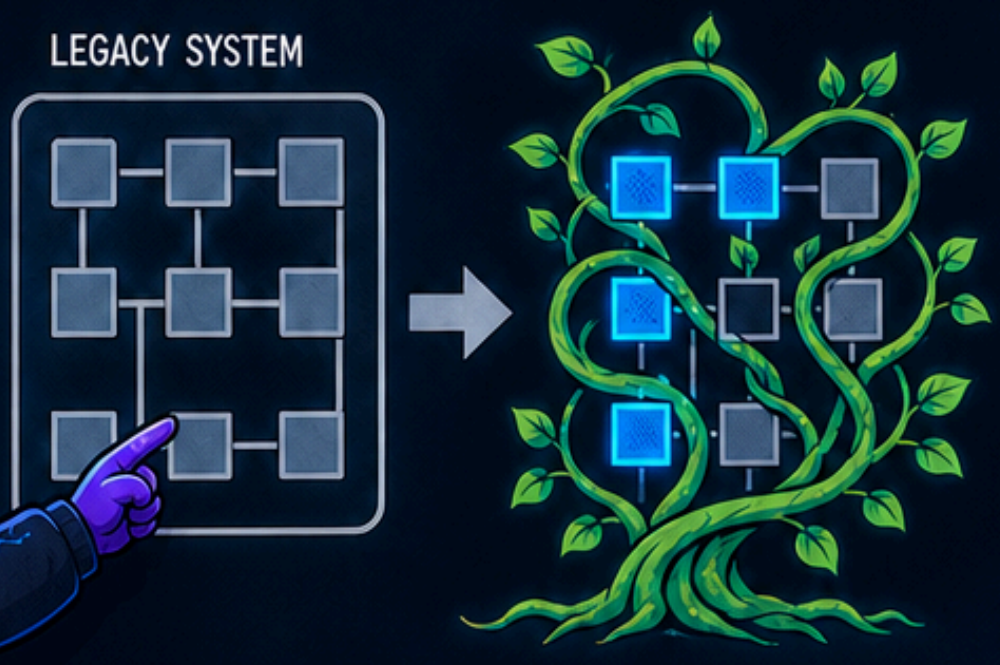
Reimplement against the specs. Build the new system feature by feature, with your BDD scenarios as the acceptance gate for each one. Nothing ships until the scenario passes.

These aren't just technical tests — they are living proof that the system is meeting the specific needs of the business. When automated scenarios run in your development pipeline, they generate living documentation that must stay current. If the system's behaviour changes, the automated scenario breaks, immediately signalling that attention is needed.

This approach means you carry forward the intent of the legacy system while leaving behind its structural debt.

THAT'S A FUNDAMENTALLY DIFFERENT OUTCOME FROM A TRANSLATION OR A REFACTOR.

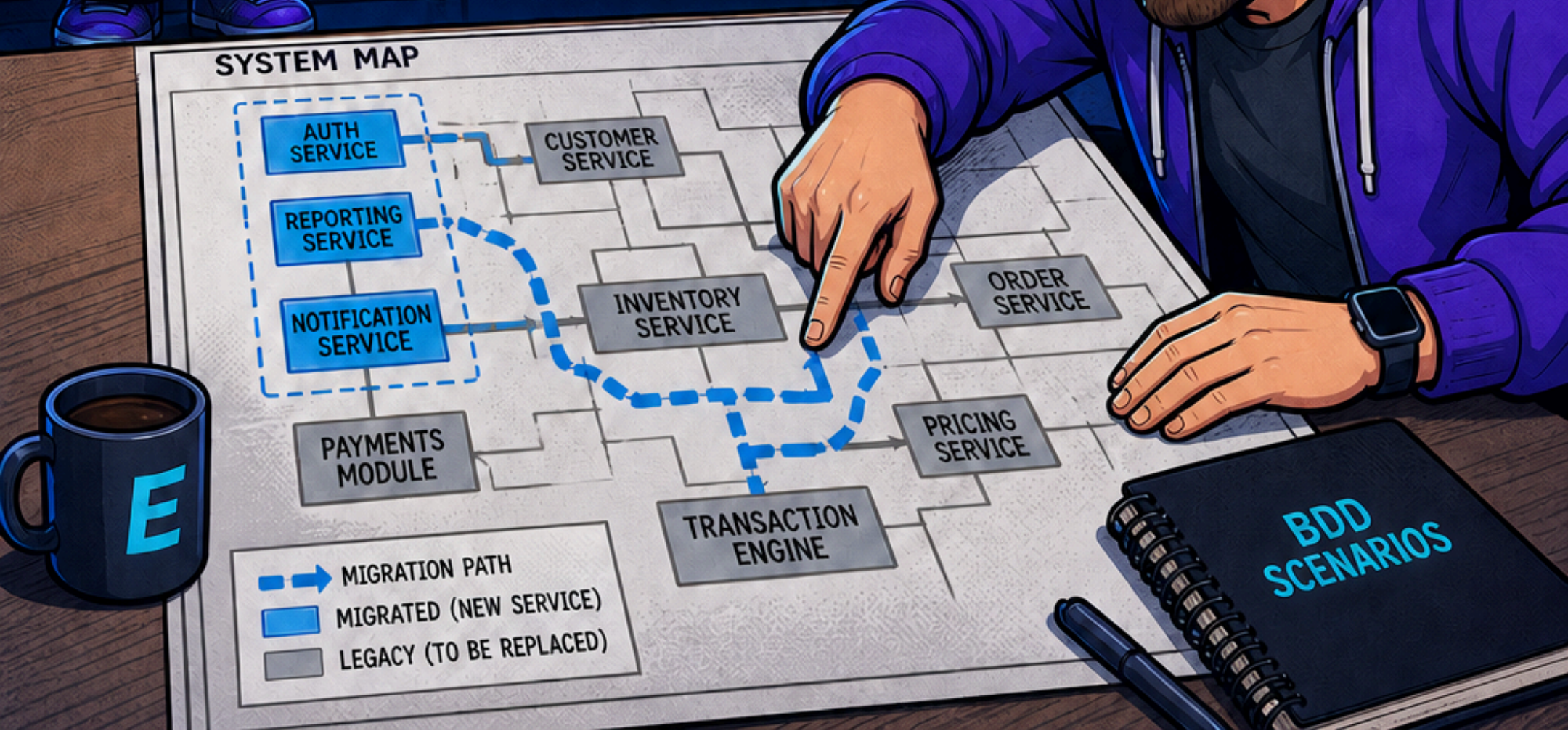
STEP THREE:
REPLACE INCREMENTALLY WITH THE STRANGLER-FIG APPROACH



NEW SERVICE

LEGACY MODULE

- ✓ BUILD IN PARALLEL
- ✓ ROUTE TRAFFIC INCREMENTALLY
- ✓ VALIDATE MODULES
- ✓ DECOMMISSION LEGACY
- ✓ CONTINUOUS FEEDBACK LOOPS



STEP FOUR:
BUILD A REGRESSION SAFETY NET FROM OBSERVED BEHAVIOUR

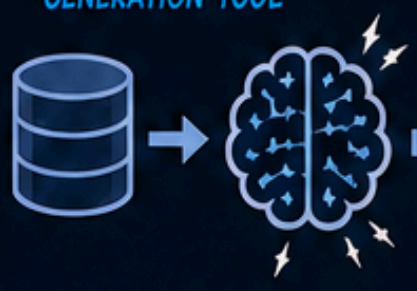
THIS IS REAL BEHAVIOUR. LET'S LOCK IT IN.

1. CAPTURE OBSERVED BEHAVIOUR

TRANSACTION LOGS (LEGACY SYSTEM)

TIMESTAMP	INPUT	OUTPUT	STATUS
10:15:23	{"action":"calc","a":7,"b":3}	{"result":10}	OK
10:15:24	{"action":"update","id":842}	{"status":"ok"}	OK
10:15:25	{"action":"search","q":"abc"}	{"results":[...]}	OK
10:15:26	{"action":"price","sku":"A123"}	{"price":19.95}	OK
10:15:27	{"action":"submit","id":842}	{"status":"accepted"}	OK
...	...	...	...

2. FEED TO AI TEST GENERATION TOOL



3. GENERATED REGRESSION TESTS

- UNIT TESTS
- INTEGRATION TESTS
- REGRESSION TESTS

GENERATED REGRESSION TEST RESULTS

TOTAL TESTS	PASSED	FAILED	SKIPPED
1,248	1,248	0	0

test_calc_addition()	PASSED
test_update_record()	PASSED
test_search_partial_match()	PASSED
test_price_lookup_valid_sku()	PASSED
test_submit_duplicate_id()	PASSED
test_edge_case_empty_input()	PASSED
test_null_handling()	PASSED
...	

STEP FIVE:
KEEP DOMAIN EXPERTS IN THE ROOM AT EVERY MILESTONE

BDD SCENARIO: LOAN REPAYMENT CALCULATION

GIVEN a loan account with a principal balance of $250,000
and an annual interest rate of 6.5%
and a repayment term of 30 years

WHEN the borrower makes a monthly payment

THEN the system calculates the interest component using the daily balance method

AND rounds the payment to 2 decimal places using HALF_UP rounding

AND updates the principal balance accordingly

EXAMPLES:

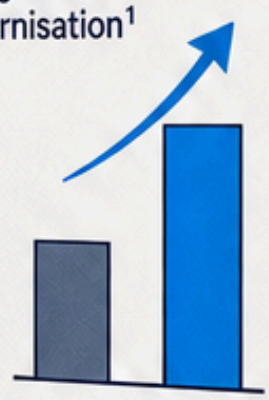
PRINCIPAL	RATE	TERM	PAYMENT
$250,000	6.5%	30 YEARS	$1,580.17
...			

WHAT THIS LOOKS LIKE ON A BOARD SLIDE

FASTER OUTCOMES

40–50%

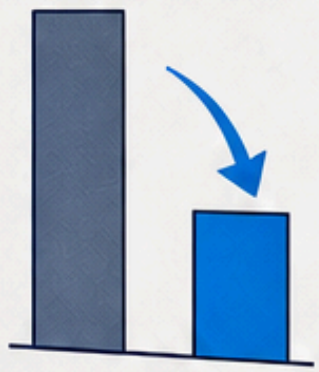
faster timelines with AI-augmented modernisation[1]



LOWER RISK

70%+

of full rewrites over budget or timeline[2]



LASTING VALUE

- Executable BDD scenarios
- Annotated business logic documents
- AI-generated regression suite

[1] McKinsey research on AI-augmented modernisation.

[2] Industry analyst & practitioner reports.

RESEARCH FIGURES

AI-AUGMENTED MODERNISATION	40–50%
FULL REWRITE OVERRUN RATE	70%+
SUSTAINABLE DELIVERY VALUE	✓ ✓ ✓

SPEC LAYER VALUE

- ✓ BDD scenarios
- ✓ Business logic docs
- ✓ Regression suite
- ✓ Onboarding & enablement
- ✓ Long-term safety net


EVOTRON
STUDIO
AGENTS
STATUS
LOAD
ACTIVE
https://evotronstudio.co.nz