

# FUNDAMENTALS OF AI-AUGMENTED ENGINEERING: A CONVERSATIONAL APPROACH

Taner Onsay, Ph.D. & Sebnem Onsay  
Authors  
August 2026 · Revision 1.0

© 2026 AI First Engineering

Problem



Model



Compute



Analyze



Validate



Improve

*The engineering workflow you will use throughout this book  
by Taner & Sebnem Onsay,*

CHAPTER - 2

PART 5 · ≈ 50 MIN

# Test · Debug · Validate

AI will help you write the code. Only you can decide whether to trust it — and one disagreement worth having.

5

Problem

›

Model

›

Compute

›

Analyze

›

**Validate**

›

Improve



# The search is built. The functions are named. It runs.

## PART 3 · DONE

### You invented the search

```
while stopping_distance(v+step, ...) <= d_avail
```

Discovered by hand, made visible in Excel, written as pseudocode, automated in Python.

## PART 4 · DONE

### You gave its tasks names

```
def stopping_distance(v, t_r, a):
```

Functions — testable pieces, built one at a time, each held to a hand-checked case.

**The code runs. It prints 20.0 m/s.** *A vehicle, a hazard, and 50 m of road are waiting on that number.*

**Today's question is different in kind: should you trust it?** Testing, debugging, and validation are how engineers earn the right to say yes — and today ends with a disagreement worth having.

Problem



Model



Compute



Analyze



**Validate**



Improve



# RUNS $\neq$ CORRECT

LEVEL 1

**The code runs**

Python raised no errors. This is the weakest possible claim.

LEVEL 2

**The code implements the intended algorithm**

It faithfully does what your pseudocode says — verified by tracing and test cases.

LEVEL 3

**The algorithm represents the engineering problem**

The procedure itself answers the right question, with the right model and conditions.

LEVEL 4

**The engineering result is validated**

Checked against independent evidence: hand calculation, analytical result, physical reasoning.

Each level can hold while the next one fails. **Engineering lives at Level 4.**

Problem



Model



Compute



Analyze

**Validate**

Improve



YOUR FIRST BUG — SMALL ENOUGH TO SEE WHOLE

# Three lines, one slip, no error message

THREE LINES FROM PART 1 — WITH ONE SLIP

```
# fall time — Chapter 1 model  
  
GET height h          # 20 m  
  
COMPUTE t = sqrt(2*h) / g  
  
REPORT t
```

It runs. It prints a number. No red text anywhere.

BUT YOU KNOW THIS ANSWER

It prints: **0.64 s**

From Chapter 1 you KNOW a 20 m drop takes  $\approx 2.0$  s — you measured it.

**The program did not announce a bug. Your known case did.**

*Can you see the slip? (Look at what is inside the square root.)*

**This is what a real bug looks like: quiet.** Syntax errors shout; the dangerous mistakes run politely and lie. Hold this three-line example — we will hunt its cousins in the 30-line search next, and then come back and catch this one, step by step.

Problem



Model



Compute



Analyze



**Validate**



Improve



# Four ways to be wrong

## SYNTAX ERROR

Fails Level 1

```
while stopping_distance(v + step, t_r, a) <= d_avail  
    v = v + step
```

Missing colon — Python refuses to run. Loud, honest, easiest to fix. AI explains these well.

## LOGICAL ERROR

Runs · fails Level 2

```
while stopping_distance(v, t_r, a) <= d_avail:  
    v = v + step
```

Tests  $v$  instead of  $v + \text{step}$  — reports the first UNSAFE speed as safe. Runs perfectly. Lies quietly.

## WRONG PROBLEM

Runs · fails Level 3

```
d = stopping_distance(v, t_r, a)  
print("Distance:", d)
```

Beautiful code for the forward problem — but the question was maximum safe speed. Correct answer to the wrong question.

## PHYSICAL NONSENSE

Runs · fails Level 4

```
v_kmh = 72          # km/h!  
d = stopping_distance(v_kmh, t_r, a)
```

72 fed as m/s ( $\approx 260$  km/h). Stopping distance  $\approx 440$  m. Every line correct — the result is physically absurd.

*For each card ask: what evidence would catch this? Interpreter · boundary test case · re-reading the question · physical reasonableness.*

Problem



Model



Compute



Analyze



Validate



Improve



THE HUNT, WORKED ON THOSE THREE LINES

# Catch it with evidence — four moves, no guessing

## 1 · RUN & OBSERVE

Observed:  $t = 0.64 \text{ s}$

No error message. Just a number.

## 2 · KNOWN CASE

By hand,  $h = 20 \text{ m}$ :  $t = \sqrt{2 \cdot 20 / 9.81} \approx 2.02 \text{ s}$

Expected vs actual disagree. Now there IS a bug — provably.

## 3 · LOCALIZE & DIAGNOSE

Only one line computes.  $\text{sqrt}(2 * h) / g$

Check its units:  $\sqrt{\text{m}}$  — not seconds! The  $\div g$  slipped outside the root. The SIZE of the error ( $\div \sim 3.1 \approx \sqrt{g}$ ) points at the cause.

## 4 · CORRECT & RETEST

$\text{sqrt}(2 * h / g) \quad t = 2.02 \text{ s} \quad \checkmark$

Fixed — and proven fixed, against the same known case.

**Evidence first, explanation second — on three lines.** No staring, no guessing, no 'why is my code wrong?'. A known case detected the bug, units located it, a retest closed it. The next slide writes this chain down as the method — and shows how to hand the evidence to AI.

Problem



Model



Compute



Analyze



Validate



Improve



# Evidence first → explanation second



## WEAK QUESTION

***“Why is my code wrong?”***

No evidence, no expected value, no test case. AI will guess — sometimes confidently, sometimes wrongly — and you cannot judge the guess.

*Debugging by vibes.*

## ENGINEERING QUESTION

***“For  $v = 20$ ,  $t_r = 1.0$ ,  $a = 7$ , I expect  $d_{stop} \approx 48.6$  m by hand, but my program returns 68.6 m. Here is my pseudocode and code. Help me identify possible causes.”***

A test case, an expected value, an observed value, and the artifacts. Now AI can actually help — and you can judge its answer.

*AI's diagnosis is a hypothesis to test — not an authoritative answer.*

***Same rule as the Chapter 1 experiments — gather the evidence before proposing the explanation.***





# Test cases are constructed, not hoped for



## Known case

Hand-calculate one full case ( $v = 20 \rightarrow 48.57$  m) and demand agreement.

*catches implementation slips*



## Boundary case

Sit exactly on the limit:  $d_{\text{stop}} = d_{\text{avail}}$ . Safe or unsafe? Did the code decide the way YOU decided?

*catches  $\leq$  vs  $<$  confusion*



## Edge case

Feed extremes:  $v = 0$ ,  $d_{\text{avail}} = 0$ , a very large step. Does behavior stay sensible?

*catches hidden assumptions*



## Nonsense probe

Can any input produce a negative distance, or a stopping distance shorter at higher speed?

*catches broken physics*

**Visual plausibility is not a test.** “The output looked reasonable” has approved a great many wrong answers.

Problem



Model



Compute



Analyze



Validate



Improve



*The Chapter 2 hunt*

# Did AI actually solve the engineering problem — or merely produce code that runs?

In Chapter 1 you hunted a wrong formula. Now you hunt the algorithm itself: its logic, its stopping criterion, its units, its boundary handling, its step size — and the assumptions it changed without telling you.

Problem



Model



Compute



Analyze



**Validate**



Improve



# Live demonstration — the class attacks the code

## THE DEMO — RIGHT NOW

We ask an AI, casually:

*“Write a Python program to find the max safe speed for a car that has 50 m to stop.”*

We run whatever it produces — it will almost certainly run.

Then the class attacks it with the four test types.

## WHAT TO HUNT FOR



What reaction time did it assume — and did it tell us?



km/h or m/s? mph? Did it say?



Is exactly-at-the-limit safe or unsafe in its code?



What step size — and does the answer change if we shrink it?



Did it silently change the model (add brake lag? drop reaction distance?)

Verdicts allowed: **ACCEPT** · **MODIFY** · **REJECT** — with reasons. If the AI got it right, saying so is also engineering.



# Validation: independent evidence, layered

1

## Hand-calculated known case

One full case worked on paper. Catches implementation slips.

SHARES THE MODEL

2

## Excel vs Python

Two implementations of the same procedure must agree.

SHARES MODEL AND ASSUMPTIONS

3

## Analytical solution

Solve  $d_{\text{avail}} = v \cdot t_r + v^2/2a$  for  $v$  algebraically — a genuinely different method.

INDEPENDENT METHOD

4

## Physical reasonableness

$d_{\text{stop}}$  grows with  $v$ ; a wet road lowers  $v_{\text{max}}$ ; magnitudes plausible vs. driver-handbook values.

INDEPENDENT EVIDENCE

Always ask: what assumptions, data, or methods do these checks SHARE? **AI #2 agreeing with AI #1 is not validation.**

Problem



Model



Compute



Analyze



Validate



Improve



# The search says 20.0 — algebra says 20.37

## THE ANALYTICAL ANSWER

$$v_{\max} \approx 20.37 \text{ m/s}$$

$d_{\text{avail}} = v \cdot t_r + v^2/2a$  is a quadratic in  $v$ . Solving it ( $t_r = 1.0 \text{ s}$ ,  $a = 7 \text{ m/s}^2$ ,  $d_{\text{avail}} = 50 \text{ m}$ ) — quadratic formula, provided, not derived here.

## THE SEARCH'S ANSWER

$$v_{\max} \approx 20.0 \text{ m/s}$$

With  $\text{step} = 0.5 \text{ m/s}$ , the loop last confirms  $v = 20.0$  — because 20.5 would already overshoot ( $50.5 \text{ m} > 50 \text{ m}$ ). The true answer sits between two tested speeds.

### Is the algorithm wrong?



No. It answered exactly what it was asked, to the resolution it was given. The disagreement is a property of the step size, not a bug.

### Shrink the step — watch the answers converge.



$\text{step } 0.5 \rightarrow 20.0$  ·  $\text{step } 0.1 \rightarrow 20.3$  ·  $\text{step } 0.01 \rightarrow 20.36 \text{ m/s}$ .  
Finer steps buy accuracy and cost computation.

**Something interesting happened here. We will return to it.** You have just met step size, resolution, and convergence — informally. Chapter 3 formalizes them; Chapter 4 returns to this exact problem with a better search.

Problem



Model



Compute



Analyze



Validate



Improve



# Your reaction time enters the model

1

Measure your own reaction time — ruler drop, several valid trials

2

Convert catch distance to  $t_r$  with the Chapter 1 model:  $t_r = \sqrt{2d/g}$

3

Excel table → recognize the repetition → write the pseudocode

4

AI critiques your pseudocode → Python with functions

5

Test · BREAK THE AI · debug with evidence

6

Validate · wet-road scenario · defend a maximum safe speed

## MODEL → INSTRUMENT

Chapter 1: free fall was the phenomenon under study.

Chapter 2: the same free-fall model measures your reaction time.

*State the assumption: a ruler-drop catch is a simple reaction ( $\approx 0.2$  s); a driver's full response is longer. Feeding one into the other is a modeling choice you must defend.*

**Your number. Your algorithm. Your code. Your decision — defended with evidence.** Everything documented in your engineering record, failures included.

Problem



Model



Compute



Analyze



Validate



Improve



# Explore / Practice · Prepare for Lab 2

## EXPLORE / PRACTICE — SHORT, PREDICTION-FIRST

- 1 • **Pseudocode a real procedure** (making coffee, checking tire pressure): label inputs, decisions, repetition, outputs.
- 2 • **Predict-then-run:** a six-line loop is provided. Write its output BEFORE executing. Then run it and reconcile.
- 3 • **Ask AI to explain a short WHILE loop** in plain engineering language. Grade the explanation: accurate or not?
- 4 • **Spot the flaw:** a short AI-written snippet contains one logical (not syntax) error. Find it by reading, confirm by running.

## PREPARE — ARRIVE READY

- Excel opens and computes on your laptop
- Your Python notebook environment runs one test cell
- Re-read this chapter's pseudocode page
- IF / ELSE · FOR · WHILE — concepts, not syntax
- Read the stopping-distance problem statement
- **Write your prediction of  $v_{\max}$  for the stated scenario — before any computing**

The laboratory opens with a short conceptual readiness check — ideas from this chapter, zero syntax trivia.

## What must be true before the laboratory

THINK › PSEUDOCODE › AI › CODE

RUNS ≠ CORRECT

The engineer designs and understands the logic.

AI can help express the logic in code.

The engineer must still test, validate, and approve the result.

**Ahead:** engineers constantly compute by repeating a calculation until a condition is met. You built that machine by hand in this chapter. The rest of the book makes it smarter — better searches, controlled error, and methods for equations algebra cannot solve at all. It begins with one question: why did the search say 20.0 while algebra said 20.37?

Problem

›

Model

›

Compute

›

Analyze

›

Validate

›

Improve

# AI = Programming Coach:

“Help me understand why my algorithm/code behaves this way.”



## What we established in this Chapter Computational Thinking & AI as a Coach:

- Engineering procedures can be expressed as algorithms.
- Decisions and repetition turn procedures into computational workflows.
- **THINK → PSEUDOCODE → AI → CODE**
- **Running program ≠ Correct engineering result.**

Now use AI to extend your understanding.



## AI COMPANION PROMPT: Algorithm Coach Prompt:

“Here is my engineering problem and my pseudocode. Do not write the code for me yet. Walk through my algorithm with me step-by-step. Ask me questions about my inputs, decisions, repetition, and stopping condition. Help me find weaknesses in my logic before I code it.”



## ENGINEER'S CHECK BOX:

- ☐ Don't start with code. **Own the algorithm first.**
- ☐ Code runs? **That does not mean it is correct.**
- ☐ Something looks wrong? **Give AI expected vs. actual evidence.**
- ☐ AI suggests a change? **ACCEPT • MODIFY • REJECT — and explain why.**



## AI COMPANION PROMPT: Go One Step Deeper Prompt:

“Give me a simple engineering problem that requires a decision, repetition, and a stopping condition. Do not solve it for me. Coach me as I develop the algorithm, write pseudocode, implement it, and test whether my result is correct.”