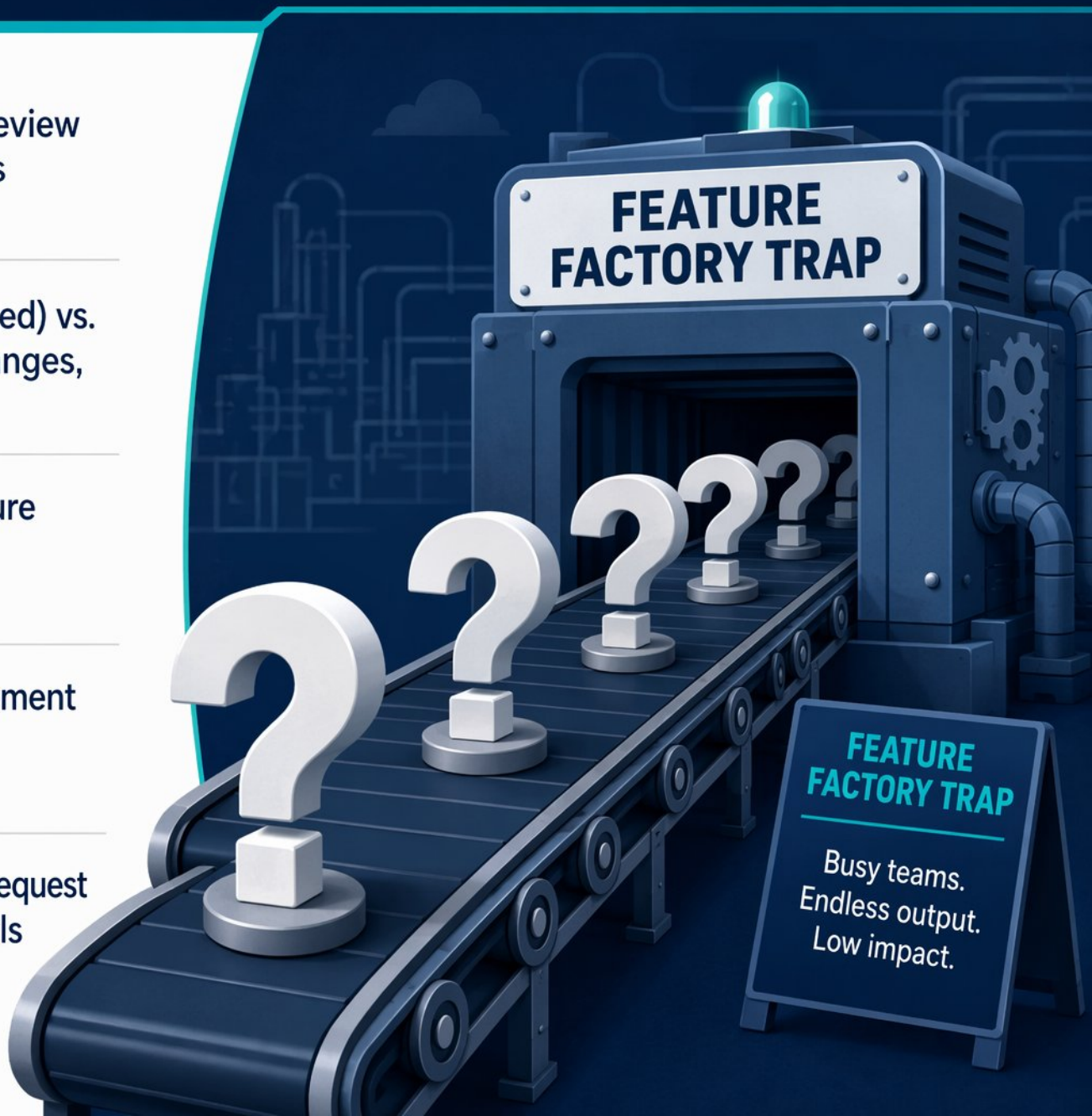
Writing User Stories and Acceptance Criteria

- **User Stories:** lightweight placeholders for collaborative conversations, not mini-specs
- **Acceptance Criteria:** specific, testable conditions that define when a story is truly done
- **Addresses real-world pain:** ambiguous requirements, rework, missed expectations, feature factories
- **Session outcome:** write value-driven, testable stories and criteria for predictable delivery
- **Hands-on session** blending theory, practice, and team-based exercises

The Real Cost of Unclear Requirements

- Endless rework, sprint review arguments, and features nobody uses
- Outputs (features shipped) vs. outcomes (behavior changes, value delivered)
- Poor stories create 'feature factories'—busy teams delivering low impact
- Shift from writing requirement documents to placing conversational bets
- Example: a vague 'login' request leads to missed user goals and rework



The 3 Cs: Card, Conversation, Confirmation

THE 3 Cs LENS

CARD

A reminder of the need—
not a spec

CONVERSATION

The primary locus of value;
details emerge through
collaborative dialogue

CONFIRMATION

Acceptance criteria that
transform intent into a shared,
verifiable contract

- Skipping conversation leads to 'telephone game' misunderstandings and delivery failures
- Stories are placeholders for conversation, not mini-requirements documents

User Story Anatomy and the INVEST Criteria

User Story Template

As a <role>, , I want <goal> so that <benefit>.

Role
The user persona doing the action.

Goal
The action or need the user wants.

Benefit
The outcome or value the user receives.

- **Template:** "As a <role>, I want <goal> so that <benefit>." Each part has a purpose.
- **Role** must be a real user persona, not "system" or "developer."
- **Goal** is a user-centric action, not a technical task.
- **Benefit** is the measurable outcome—why this matters.
- **Quality filter:** INVEST (Independent, Negotiable, Valuable, Estimable, Small, Testable).

Outcome over Output: The 'So That' Clause

WEAK

“I can see my orders”

(restates the action, no value)

VS.

STRONG

“I can track my spending and reconcile monthly”

(measurable impact)

Business outcomes: lagging financials (revenue, retention)

Product outcomes: leading behavior changes (conversion, activation)

A strong “so that” prevents scope creep and guides trade-offs

Vertical Slicing: Delivering Value Every Iteration

- Vertical slice: a thin, end-to-end feature cutting through all architectural layers.
- Horizontal slice: a UI-only or DB-only story that delivers zero standalone value.
- Cake analogy: a vertical slice is a complete, edible bite; a horizontal layer is just raw sponge.
- Enables fast feedback, incremental delivery, and agile transparency.
- Start with a "walking skeleton" or "steel thread"—the thinnest possible vertical slice first.

Practical Story Splitting Strategies

- **Split when story exceeds ~1-3 days or is unestimable.**
- **Heuristics: Workflow Steps, Business Rules, Data Variations, Operations (CRUD).**
- **Meta-pattern: Find core complexity, reduce variations, then split by those variations.**
- **Example: "Purchase product" → Add to cart, Enter shipping, Enter payment, Confirm.**
- **Anti-patterns:** Avoid splitting by technical layer (frontend/backend) or happy path vs. error handling only.

Acceptance Criteria: Defining the Conditions of Satisfaction

- **AC** are specific, testable conditions a story must meet to be accepted
- **AC** differ from DoD: AC are story-specific; DoD is a universal quality gate
- **Two formats:** Rule-based (checklist) and Scenario-based (Given-When-Then)
- **Good AC:** clear, concise, testable, outcome-focused, and measurable
- **Example:** "User can add items to wishlist" → bulleted AC checklist

DEFINITION OF DONE

- Code is complete
- Peer reviewed
- Unit tests written
- Build succeeds
- Deployed to environment
- Docs updated

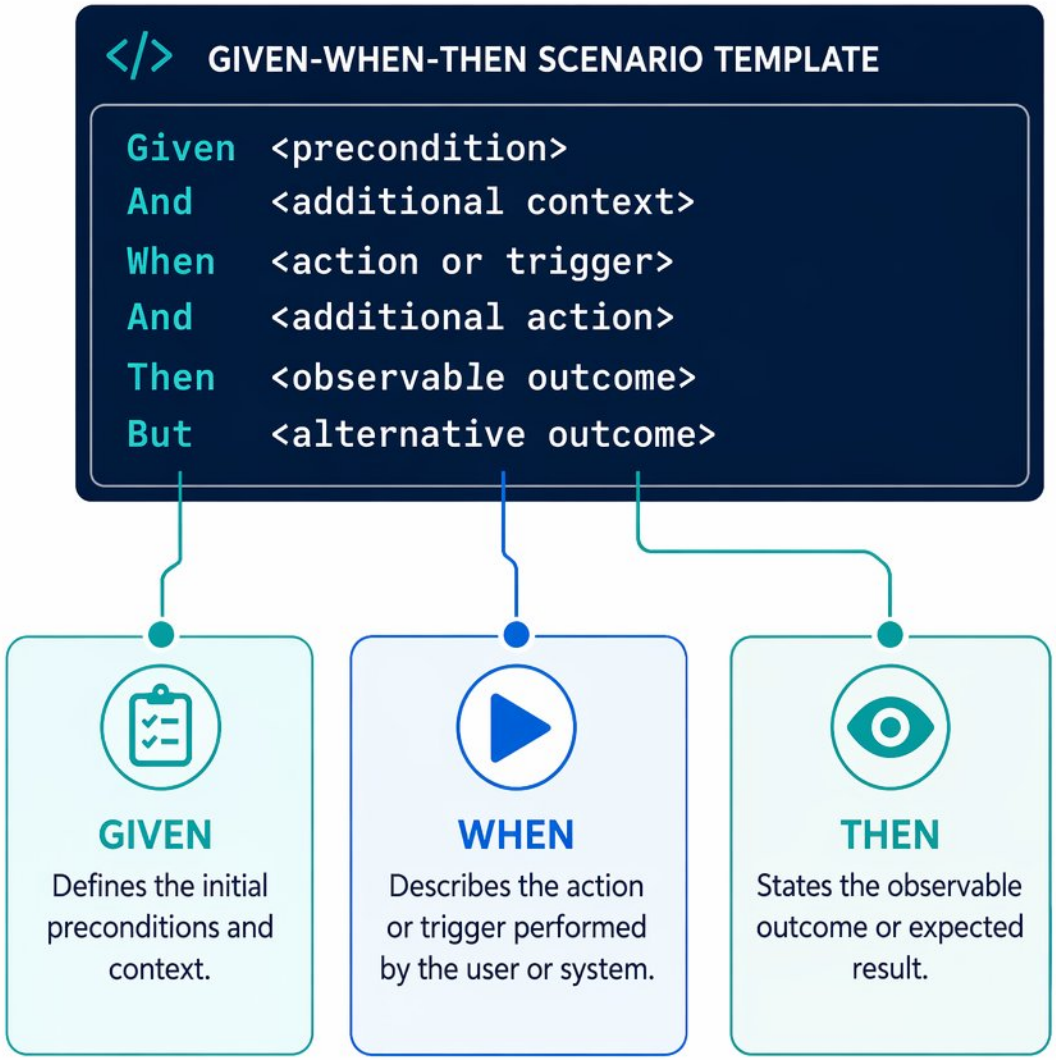
VS.

STORY-SPECIFIC ACCEPTANCE CRITERIA

- User can add items to wishlist
- Item is saved to user's wishlist
- Wishlist is accessible from profile
- User can remove items from wishlist
- An empty wishlist message is shown when no items exist

Writing Gherkin-Style Acceptance Criteria

- Given-When-Then: a structured, testable scenario format from BDD
- Given = preconditions; When = action/trigger; Then = observable outcome
- Use And/But to extend steps without repeating Given, When, or Then
- Separate scenarios for happy path, error handling, and edge cases
- GWT scenarios translate directly into automated tests (Cucumber, SpecFlow)



Collaborative Story-Writing Workshop

- Groups receive a raw, ambiguous need (e.g., 'improve checkout')
- **Task 1:** Write a story with a real role and measurable benefit
- **Task 2:** Self-assess and refine the story using INVEST criteria
- Facilitator prompts for clarity on user roles and value statements
- **Debrief:** common successes and challenges in translating need to story

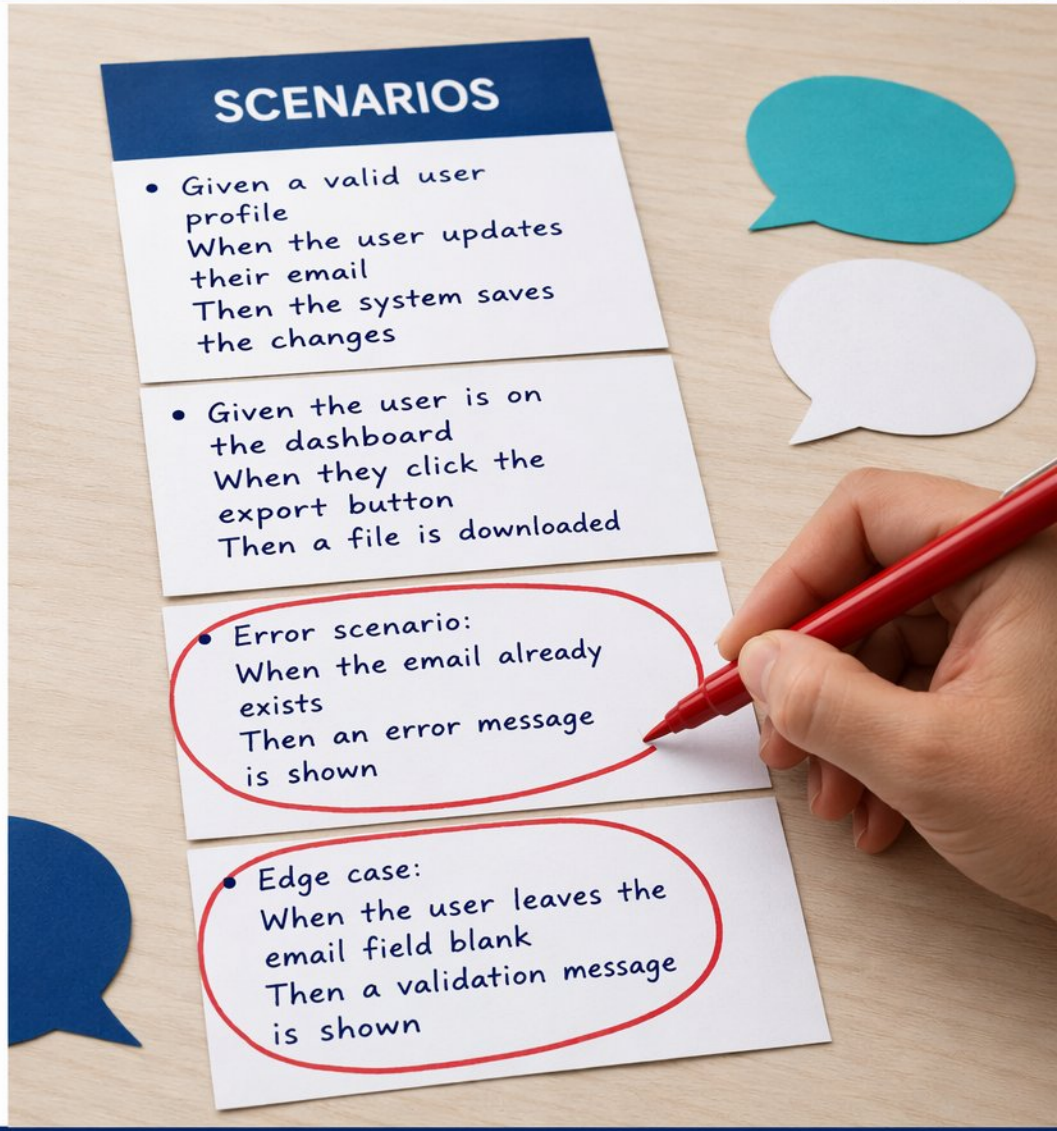
Refining Acceptance Criteria Together

Task 3: Write 3-5 criteria mixing checklist rules and Given-When-Then scenarios

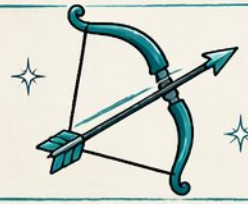
Task 4: Document at least one error scenario and one edge case

Peer review: spot ambiguity, missing conditions, or solutioning in another group's AC

Common pitfalls: vague language, implementation details, missing preconditions



Common Anti-Patterns and How to Fix Them

ANTI-PATTERN	HOW TO FIX IT
 • Too vague / missing 'why': write with a specific persona and measurable outcome.	 Write with a specific persona and measurable outcome.
 • Technical tasks as stories: reframe as user value; use 'Chore'/'Enabler' for tech work.	 Reframe as user value; use 'Chore'/'Enabler' for tech work.
 • Over-specification / solutioning: keep story a conversation starter, detail in AC.	 Keep story a conversation starter, detail in AC.
 • Bloated, unfinishable stories: apply vertical slicing; enforce a sprint-fit size limit.	 Apply vertical slicing; enforce a sprint-fit size limit.
 • Writing in isolation: use collaborative refinement, walkthroughs, and 3 Amigos (BA, Dev, QA).	 Use collaborative refinement, walkthroughs, and 3 Amigos (BA, Dev, QA).

Avoiding the Feature Factory: Outcome-Driven Backlogs

OUTPUTS

What you ship.
The features and deliverables.

VS.

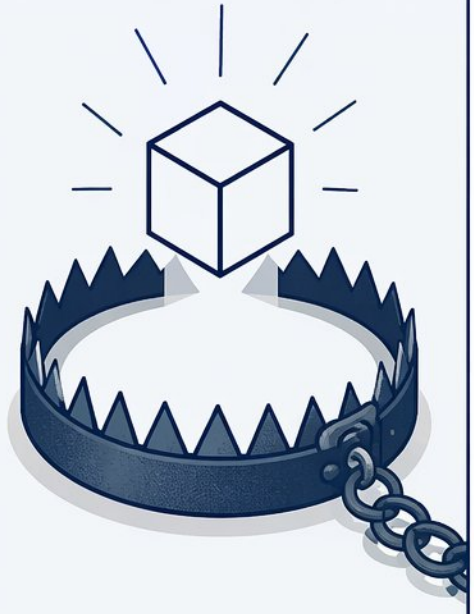
OUTCOMES

The user behavior change your work creates.

HYPOTHESIS STORY: **"We believe that [doing X] for [user] will achieve [Y]"**

- Output is what you ship; outcome is the user behavior change it creates
- Ship 10 features (output) vs. increase activation from 10% to 15% (outcome)
- Use Hypothesis Stories: "We believe that [doing X] for [user] will achieve [Y]"
- Validate with data: adoption, retention, conversion, and support ticket trends
- Shift stakeholder questions from "When will X ship?" to "How will we know X works?"

FEATURE FACTORY TRAP

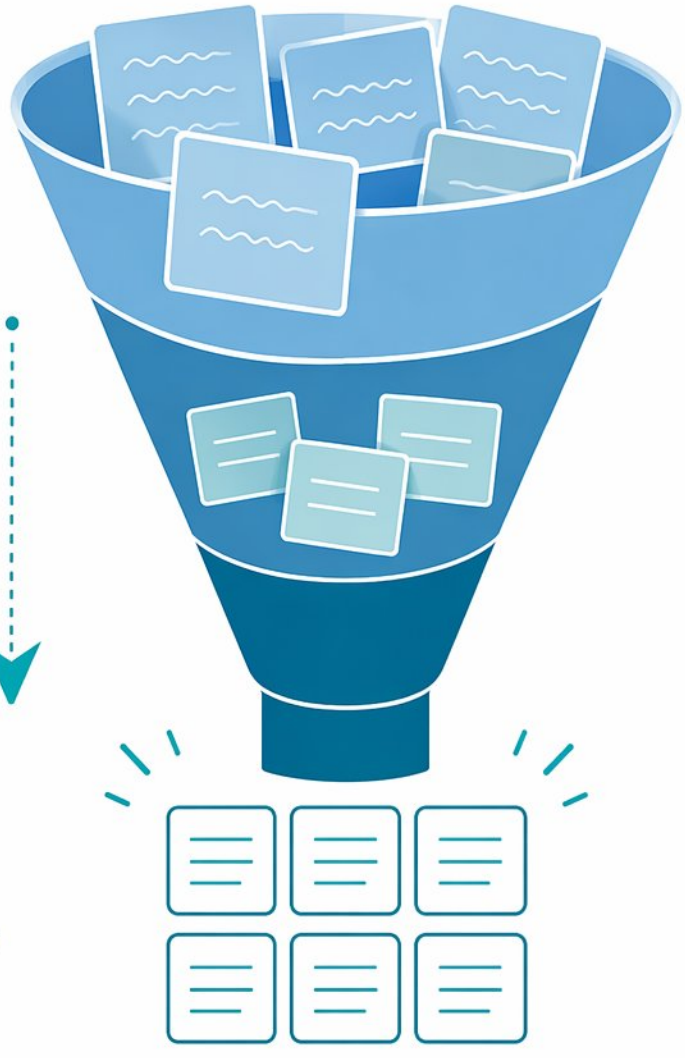


Busy shipping,
but not moving outcomes.

Refining the Backlog: From Good to Great

Large & Fuzzy

High-level, unclear



Small & Detailed

Clear, ready for sprint

- **Backlog Refinement:** ongoing, collaborative prep of stories for upcoming sprints
- **Use a Definition of Ready (DoR):** story format, INVEST, AC, dependencies, size
- **Just-in-time elaboration:** top stories small & detailed; lower stories larger & fuzzier
- **The 3 Amigos** (Product Owner, Developer, Tester) refine stories and AC together
- **Healthy backlog:** regular grooming, split large items, remove zombie stories

Integrating Stories into Your Agile Workflow

- Map the story lifecycle: idea → refinement → sprint planning → implementation → review → done

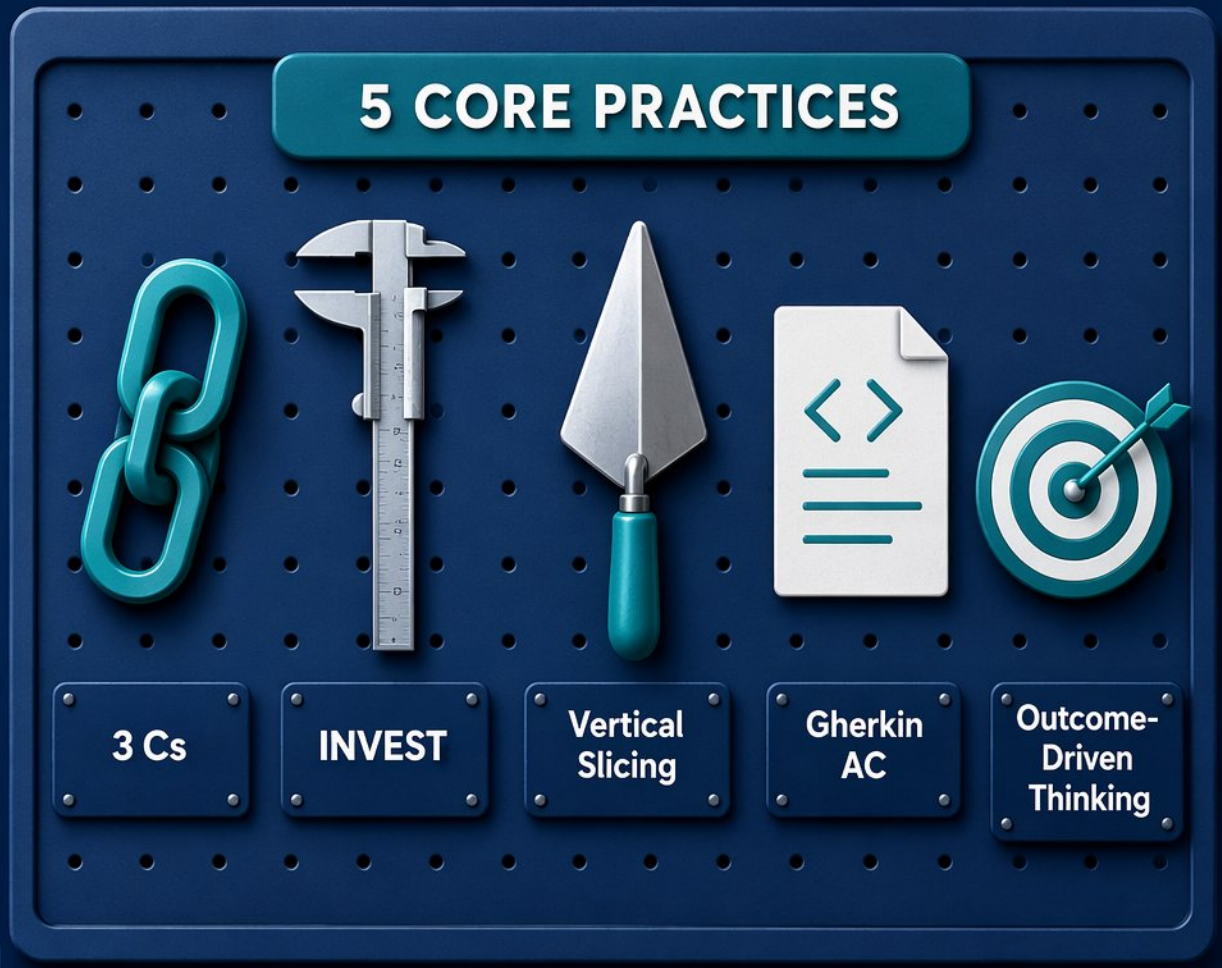
- Use stories and AC in Sprint Planning to commit and build a sprint backlog

- AC drive development and testing: TDD, BDD, and automated checks within the sprint

- Sprint Review demo: showcase a working, vertically sliced story that meets its AC

- Update stories and AC based on stakeholder feedback and sprint learnings

Action Plan and Next Steps



NEXT STEPS

- Recap 5 core practices: 3 Cs, INVEST, Vertical Slicing, Gherkin AC, Outcome-Driven Thinking
- Audit top 5 backlog items against INVEST; run a story-splitting workshop
- Pilot GWT format for one story; hold retrospectives on story quality
- Great stories create shared understanding, not perfect documentation

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/95de2a25/public