Ways of Working

Build the product right. Ship value fast. Learn continuously

Overview

- Product development happens in uncertain and complex environments, where requirements are never absolute truths.
- They are **hypothesis** about how users behave and how businesses create value.
- **Continuous Delivery** is how we test those hypothesis in the real world.
- Through small, frequent, high-quality releases, we move from assumption to evidence turning delivery into a scientific learning loop.

From Projects to Experiments

- Traditional delivery treats output as success. We treat **learning as the unit of progress.**
- Each release is an experiment that answers a question:

  *“We believe that **[doing this]** for **[these users]** will achieve **[this outcome]**. We’ll know we’re right when we see **[this measurable signal]**.”*

- This approach lets us validate or falsify assumptions quickly.
- Continuous Delivery enables us to **learn fast, adapt safely, and ship continuously.**

Principles

- **Faster Time to Market:** shorten lead time from idea to impact.
- **Higher Quality:** Automation, Testing, and observability built into every stage.
- **Better Products:** Guided by user behaviour and evidence, not assumptions.
- **Lower Costs:** Reduced rework through early validation.
- **Low-risk Releases:** Small, Reversible, Measurable changes.
- **Happier Teams:** Empowered to own outcomes, not just output.
- **Eliminate Waste:** Focus on what delivers learning and value.
- **Early Feedback:** Continuous validation from code to customer.

Principles continued

Each release reduces risk across four dimensions:

- **Value:** Does it matter?
- **Usability:** Can users do it?
- **Feasibility:** Can we build it?
- **Viability:** Should we build it?

Process and Practices

Our delivery process connects discovery outcomes to production learning forming a closed feedback loop of build, measure, and learn.

1. **User Story Mapping:** Begin with a User Story Map to build shared understanding across product, design, and engineering. It aligns everyone on the user journey, product vision, and flow of value, turning ambiguity into clarity before implementation starts.

Shared understanding is the foundation of shared ownership.

Process and Practices continued

2. Define the Hypothesis: Start with a clear, testable statement of intent.

*"We believe that **[this capability]** Will result in **[this outcome]**. We will know we have succeeded when **[we see a measurable signal].**"*

This transforms delivery from output planning to evidence generation.

Process and Practices continued

3. Define Outcome and Persona: Each MVP starts with a clear outcome written from the perspective of a real user persona. Rather than listing features, we describe what success looks like *for the person involved*. This outcome narrative anchors scope, context, and measurable value.

For example:

Talia connects a legacy project and sees its initial TDD health score.
Her developer, Devin, gets immediate feedback on a new PR, preventing a TDD violation.
Their manager, Marcus, gains confidence that Code Clinic can stop tech debt from growing.

Process and Practices continued

This single narrative defines:

- **Who** the key personas are (Tech Lead, Developer, Engineering Manager/CTO)
- **What** outcome each expects
- **Why** it matters (reduced tech debt risk, visible confidence in process)

From these outcomes, we derive the **Minimum Valuable Scope -** the smallest slice of functionality needed to make that outcome true.

Process and Practices continued

We use this pattern across all MVPs:

1. Identify personas directly tied to the experiment.
2. Write outcomes in first-persona form (“As [persona], I can...”).
3. Map features only if they are essential to achieving that outcome.

This practice keeps **MVPs** anchored in value, not volume.

4. Event Storming and Bounded Contexts: Map domain events and define clear boundaries to align technical architecture with product understanding.

Process and Practices continued

5. Modular Architecture: Adopt a mono-repo structure for modular, incremental delivery and high cohesion.

6. Setup CI/CD Pipeline: Automate integration, testing, and deployment early. Pipelines are the nervous system of learning velocity.

7. Pick the First Story: Choose the first vertical slice that delivers value and tests the end-to-end flow.

8. Define Acceptance Scenarios: Collaborate with Business and QA to establish the **Definition of Done** through shared acceptance tests.

Process and Practices continued

9. Example Mapping - Making the Implicit Explicit: Before diving into technical tasks, we use Example Mapping to clarify assumptions and align understanding across Product, QA, and Engineering.

Example Mapping is a structured conversation technique that turns vague acceptance criteria into shared, concrete examples.

It helps us uncover ambiguity, challenge assumptions, and create a common mental model before writing a single line of code.

Process and Practices continued

Each session uses four types of cards:

Card Type	Represents
🟨 Story (Yellow)	The user story or business rule
🟩 Example (Green)	Concrete examples or scenarios
🟥 Rule (Red)	Business rules or constraints
🟦 Question (Blue)	Unknowns or ambiguities

Process and Practices continued

Through this quick collaborative mapping, teams:

- **Expose ambiguity early** (before it becomes rework)
- **Align on expected behaviors** across roles
- **Feed scenarios directly into BDD acceptance tests**
- **Document decisions visually** for future reference

Once examples are clarified, we move into technical sequencing and design, ensuring implementation matches intent without hidden gaps.

Process and Practices continued

10. Design Scalable Architecture: Create high-level designs and sequence diagrams for clarity and reusability.

11. Testing Discipline: TDD, BDD, and Continuous Feedback
Testing is not a phase; it’s how we design. Our practice blends Test-Driven Development (TDD), Behavior-Driven Development (BDD), and Modern Test Pyramid approach.

Process and Practices continued

Test	Purpose	Run Frequency	Confidence Type
Unit Tests (TDD)	Verify application-side business behavior	Every Commit	Fast feedback on regressions
Narrow Integration Tests	Verify presentation & infrastructure behavior within the component	Every Commit	Fast feedback on regressions
Component Tests	Verify behavior of components (front end behavior, microservices behavior)	Every Commit	Fast feedback on regressions
Contract Tests	Verify communication between a component and its dependencies	Every Commit	System integrity across boundaries

Process and Practices continued

Test	Purpose	Run Frequency	Confidence Type
Smoke Tests	Verify that the system is up & running	On every release to any environment	Fast feedback on regressions
E2E Tests	Verify system behavior, however the scenarios are limited	On every release to acceptance environment	Fast feedback on regressions
Acceptance Tests (BDD)	Verify behavior of the system.	On every release to acceptance environment	System is working as expected by business
External System Contract Tests	Verify external system contracts matches our expectations	On every release to acceptance environment	System integrity across external systems

Process and Practices continued

Test	Purpose	Run Frequency	Confidence Type
Mutation Tests	Assess test suite strength	Periodic	Evidence of coverage quality
Exploratory & Usability Tests	Human-centered validation	Pre- and Post-release	Experience insight
Telemetry & Analytics Checks	Observe real usage and impact	Continuous	Learning from production

Process and Practices continued

12. Continuous Integration: Pipelines run every layer of tests automatically:

- **Commit Level:** Unit Tests + Component Tests + Contract Tests + Narrow Integration Tests
- **Release Level:** Smoke Tests + Exploratory Tests + Usability Tests
- **Acceptance Level:** Smoke Tests + Acceptance Tests + External System Contract Tests + E2E Tests

Each layer provides fast feedback ensuring **code is always in a releasable state.**

Process and Practices continued

13. QA performs Exploratory and Usability Testing: QA’s role goes beyond verification; they explore. They validate unknowns through **exploratory testing,** ensuring real-world usability and uncovering scenarios automation may miss.

14. Business-Driven Releases: Every time code is in a releasable state, the business decide when to release. Releases are no longer technical events; they are business events driven by learning goals.

Process and Practices continued

15. Post-Release Monitoring: Once deployed, we observe user behavior and system telemetry to validate our assumptions. Monitoring isn’t reactive; it’s part of learning cycle. Each metric becomes feedback to improve the next hypothesis.

16. Pivot or Persevere: After analyzing outcomes, if our hypothesis was false, we pivot. If it holds, we persevere. Continuous Delivery is about shortening the loop from learning to action.

Learning Through Delivery

The outcome of a delivery cycle is not just working software, it’s measurable evidence and learning.

To learn effectively, we follow a scientific method, the PDCA loop:

Plan -> Do -> Check -> Act

Continuous Delivery becomes the practical expression of that scientific loop giving teams the ability to learn quickly, safely, and continuously.

The Culture Behind Continuous Delivery

Continuous Delivery requires a mindset shift from “shipping code” to “learning safely”. Without the right values, no amount of tooling or process can sustain flow, quality, or trust.

Avesta’s CD culture is grounded in the core Extreme Programmming (XP) values that make high-frequency delivery sustainable.

The Culture Behind Continuous Delivery continued

Value	Behavior it Drives
Communication	Open, real-time dialogue across product, design, and engineering. Feedback loops replace handoffs.
Simplicity	Build only what is necessary now. Optimize for change, not prediction.
Feedback	Every test, pipeline, and metric exists to shorten the learning loop.
Courage	Ship often, refactor boldly, and face production reality early.
Respect	Assume good intent, embrace pair programming, review code kindly, and treat quality as collective responsibility.
Humility, Curiosity, Empathy	Acts of giving credit to others, admitting mistakes, asking for feedback, or showing openness to learning. Asking thoughtful questions, exploring unfamiliar ideas, experimenting, or reading widely. Listening actively, validating others' feelings, showing compassion, or offering support.

Why It Matters

At Avesta, Continuous Delivery isn’t a DevOps capability; it’s a product learning system. It ties discovery, design, and delivery into one continuous cycle of hypothesis, experiments, evidence, and decisions. It gives structure to creativity ensuring that innovation is repeatable, measurable, and directed toward business growth.

It gives teams the power to:

- Learn faster than competition
- Reduce risk without reducing speed
- Align engineering excellence with business outcomes