AXONIQ

// A WORKFLOW GUIDE 04 STEPS

Working with Dev Agents.

Turn chaotic requirements into validated, event-sourced code. **Specify first. Generate second. Review formally.**

USE THIS WHEN

You are turning a fuzzy feature request into specifications an agent can build without guessing.

YOU WILL LEAVE ABLE TO

Write a sealable Given / When / Then spec, isolate one handler per command, and review generated code against your consistency boundaries.

ONE WORKED EXAMPLE, CARRIED THROUGH EVERY STEP Place Order FOLLOW IT ↓

01

DISCOVERY

▸ Place Order

Event Modeling.

Event Modeling lays out a visual, jargon-free timeline of what users do, what the system records, and what they see. It reads left to right, like a storybook.

USER

COMMAND
Place Order
an intention · can fail

RECORDS →

EVENT
Order Placed
an immutable fact

PROJECTS →

VIEW
My Orders
what users see
EVENTUALLY CONSISTENT

The three mechanics. Commands are intentions that succeed or fail. Events are immutable facts you cannot un-happen. Views are projections built from events, so they are eventually consistent, never the source of truth.

02

SPECIFICATION

▸ Place Order

Pressure-Test One Command.

Slice one command vertically and write **Given / When / Then** precise enough to hand to an agent. The same spec doubles as an executable test.

Place Order ↓ VERTICAL SLICE · COMMAND UNDER TEST

GIVEN PRECONDITIONS	Customer is logged in · Cart has items · Stock checked inside the decision's **Dynamic Consistency Boundary (DCB)**
WHEN TRIGGER	**PlaceOrder** command is issued
THEN EMITS THIS HANDLER	**OrderPlaced** · **StockReserved** // atomic within the DCB spanning Order + Inventory
THEN REACTS DOWNSTREAM	Confirmation email // async reactor, not a synchronous postcondition
THEN REJECTS FAILURE	Out of stock · Payment declined → recorded as **OrderRejected** // failure is a fact too

The critical question. What must be true to succeed, and every way it can fail? Given / When / Then pins behavior. It does not pin concurrency, idempotency, or ordering, which is why a human still reviews in step 4.

03

ARCHITECTURE

▸ Place Order

One Engineer, One Handler.

Flip Conway's Law into a tool: one independent command handler per command, so team structure maps onto architecture. Slices are built in parallel.

LANE 01	A	**PlaceOrderHandler**	// owns one command
LANE 02	B	**CancelOrderHandler**	// independent logic
LANE 03	C	**UpdateAddressHandler**	// no shared handler code
SHARED KERNEL	event types, value objects, and the My Orders projection still need deliberate coordination		

The benefit. Conway's Law works for you. Handler logic stays isolated, so the conflict surface shrinks to the shared event and projection layer, where you coordinate on purpose rather than by accident.

04

GENERATION & REVIEW

▸ Place Order

Hand It to the Agent.

Pass the sealed spec to the **Axoniq Dev Agent**. It generates the code. You shift from author to reviewer.

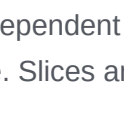
SPEC.GWT — SEALED

GIVEN logged in, cart filled

WHEN PlaceOrder issued

THEN OrderPlaced emitted

HANDOFF →



AXONIQ DEV AGENT

GENERATES →

```
// ILLUSTRATIVE, LANGUAGE-AGNOSTIC
@CommandHandler
fun handle(cmd) {
  reserve(stock)
  emit(OrderPlaced)
}
```

AUTOMATED GATE The same spec runs as an Axon test fixture: **given** past events, **when** the command, **then** expect events or rejection. It verifies the agent's code before you read it.

YOU REVIEW WHAT THE SPEC CANNOT SAY

- expected-version / concurrency
- command idempotency
- event ordering
- Sliced Architecture holds
- DCB holds

Your final check. Each decision is strongly consistent within its boundary; views stay eventually consistent across them. Validate the boundaries hold, and the system is correct by construction.

// THE RESULT

Model deliberately. Specify precisely. Slice cleanly. *Review formally.*