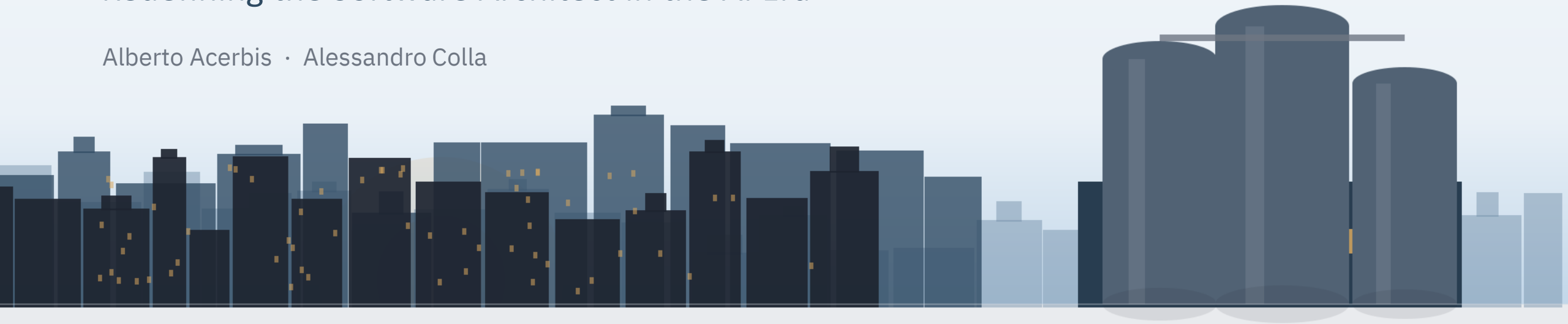
# Spec-Driven Development

## Redefining the Software Architect in the AI Era

Alberto Acerbis · Alessandro Colla



https://github.com/BrewUp/DWX26

# The Promise

AI promises to make software development easier.
The hard part is making intent survive generation.

We believe the first sentence.

This talk is about the second.

# A new BrewUp feature

feature request

```
Implement order confirmation for BrewUp.

An order can be confirmed when:
  - the customer payment has been authorized;
  - all requested beers are available in the warehouse.

When the order is confirmed, reserve the stock.
```

**Sales**
commercial commitment

**Warehouse**
physical stock

**Payment**
authorization · new for this story

# The generated feature

Polished, compiles, tests green.

SalesOrder

```
SalesOrder
├─ PaymentAuthorizationId?
├─ StockReservationId?
└─ ConfirmOrder(...)

also generated:
+ SalesOrderConfirmed event
+ ReserveItemStock command
+ handlers, read models, tests
```

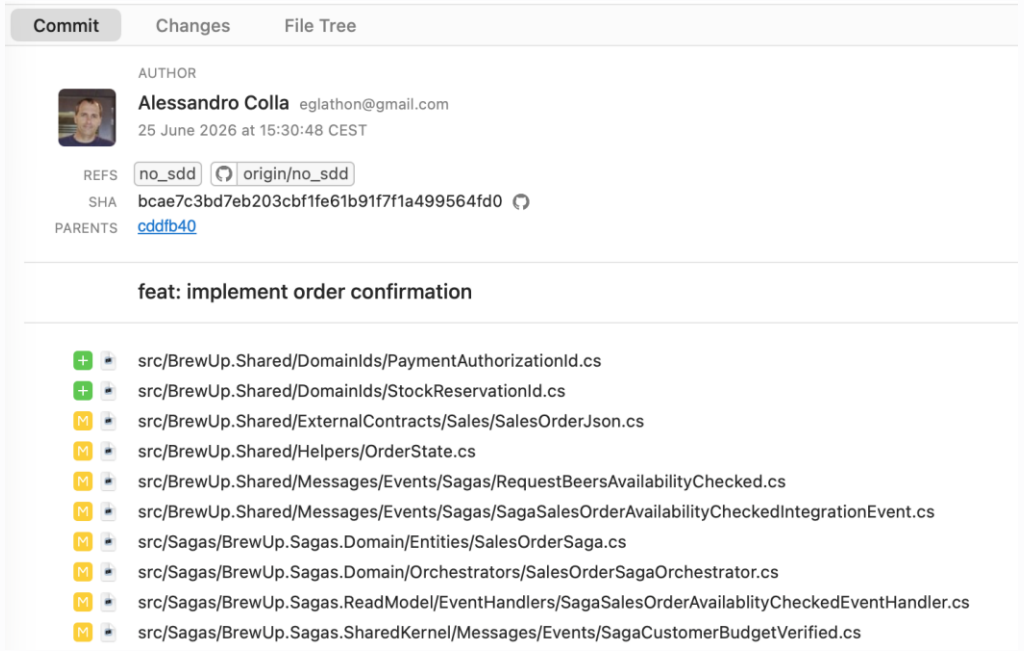
SalesOrder.cs

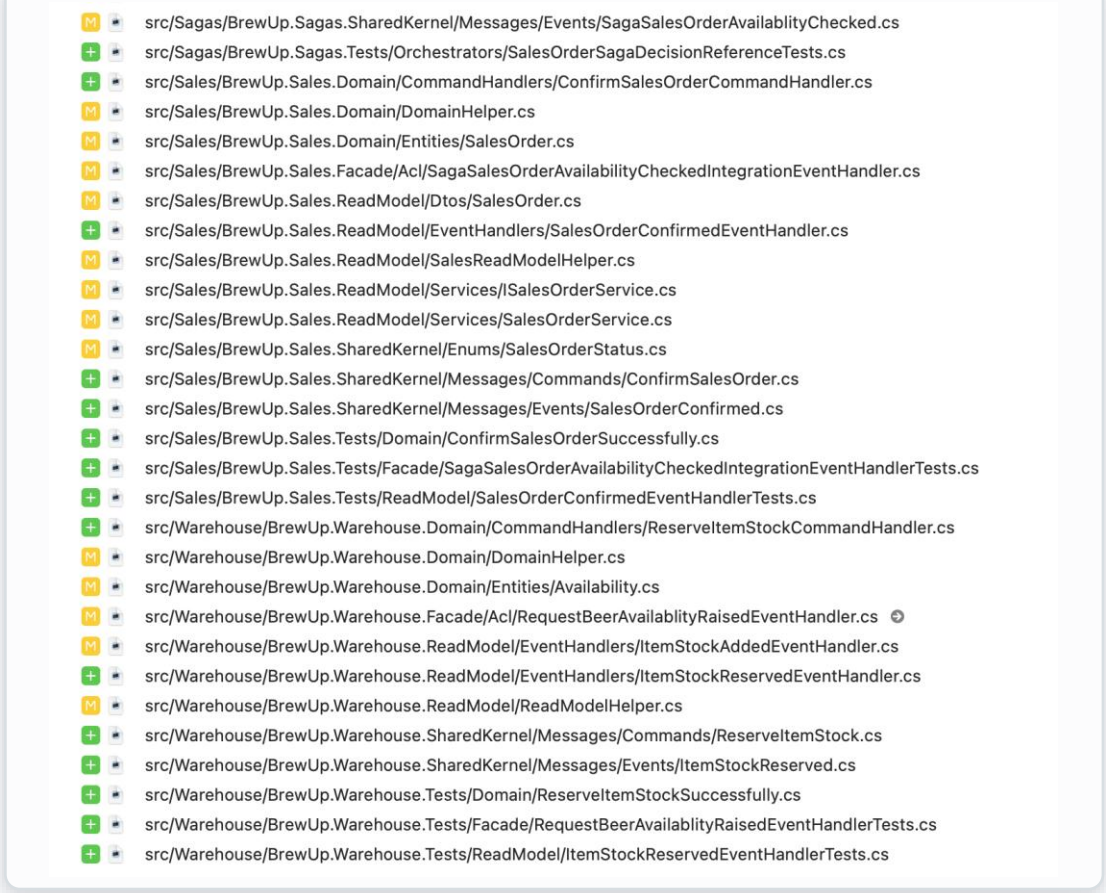
```
internal void ConfirmOrder(
    PaymentAuthorizationId paymentAuthorizationId,
    StockReservationId stockReservationId,
    Guid correlationId)
{
    RaiseEvent(new SalesOrderConfirmed(
        new SalesOrderId(Id.Value),
        paymentAuthorizationId,
        stockReservationId,
        correlationId));
}
```

# The generated feature

Compiles and tests are green. --> 39 files touched with 14 new files

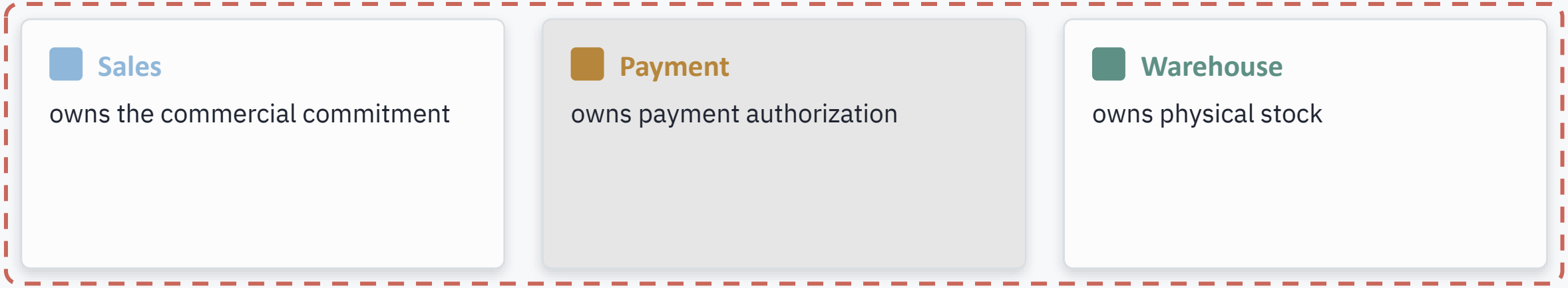
Just one commit



# Would you approve this PR?

# One aggregate, three authorities




## DDD speaking, Payment is implicit and not explicit.

# The right nouns, invented evidence

The references look right. The evidence is manufactured by the workflow.

where the ids come from?

```
// saga
new PaymentAuthorizationId(correlationId.ToString());

// warehouse ACL
new StockReservationId(Guid.CreateVersion7().ToString());
```

- **An id — but no Payment authorization decision**
- **An id — but no Warehouse reservation lifecycle**

**An identifier is not evidence unless the owning authority produced the decision it refers to. it refers to.**

# One transaction?

```
Authorize payment
        +
Reserve stock
        +
Confirm Sales Order
        =
```

**One transaction?**

**Payment authorized**

**Stock reservation failed**

*Several policies are plausible. None is correct until someone with domain authority decides.*

# Availability is not a durable fact

```
Order A              Order B

check: available     check: available
          \           /
       reserve
```

```
Reserve first → payment fails → stock remains held
```

**The simple Confirm() method is a distributed business process.**

# A longer prompt?

Conversation A

**prompt v1**

Conversation B

**prompt v2**

Conversation C

**prompt v3**

**Synchronized copy and paste is not governance.**

**Every new session does not consider previous ones.**

# From prompting to engineering discipline.

Used in this story

Constitution → specify → clarify → plan → task → implement

# Prompt versus specification

| Prompt | Specification |
| --- | --- |
| Requests an output | Records a decision |
| Lives in a conversation | Lives with the system |
| May be copied | Is referenced and checked |
| Invites assumptions | Makes constraints explicit |

**A prompt asks for an output.**
**A specification records decisions that subsequent outputs must respect.**

# spec.md

Language and ownership become durable.

/speckit.specify · Ubiquitous Language

**Sales Order**
A commercial commitment made by a customer.

**Payment Authorization**
An outcome produced by Payment.

**Stock Reservation**
An outcome produced by Warehouse.

memory · Bounded Context Rules

**BC-004**
Payment Authorization is produced by Payment. Sales stores the id as evidence.

**BC-006**
Stock Reservation is produced by Warehouse. Sales stores the id as evidence.

**BC-011**
The agent MUST NOT invent timeout, retry, expiry, void or release policy.

# Where humans must decide

```
/speckit.clarify
```

1. Confirmation sequencing (OQ-1 / FR-010)

2. Payment authorized but stock not reservable (OQ-2 / FR-011)

3. Partial reservation and the confirmation invariant (OQ-3)

The agent may identify the ambiguity. ambiguity.

**The domain expert must resolve it. resolve it.**

**The specification must preserve the resolution.**

# Let’s plan and watchout for tokens cost!

```
/speckit.plan
```

This is the moment where we choose the tech stack and we create the implementation plan

As per /specify, we suggest to use a powerful frontier model to generate the plan

# Create the tasks list

```
/speckit.task
```

```
Generate actionable task lists for implementation

Also, here we prefer to use a powerful frontier model
```

# Start to code

```
/speckit.implement
```

Execute all tasks to build the feature according to the plan

We can use a very cheap model because it just has to write code following strict rules

**What is the result?**

**Show me the code!**

# Behaviour before mechanism

| Specification | Plan |
| --- | --- |
| What must happen? | How is it coordinated? |
| Who owns the decision? | Which component coordinates? |
| Which outcomes are valid? | Events, retries, process manager? |
| What happens on failure? | Which technical mechanism? |

**SDD prevents technical design from silently inventing domain behaviour.**

**AI is not your senior architect. It’s a very fast junior with no memory of your business, no taste, and no shame.**

# The implementation still violated the specification

It has the two references we asked for — but the workflow manufactured them.

```
// saga
var paymentAuthorizationId =
    new PaymentAuthorizationId(
        correlationId.ToString());

// warehouse ACL
StockReservationId stockReservationId =
    new(Guid.CreateVersion7().ToString());

reserveCommands.Add(new ReserveItemStock(...));
```

Invented evidence

**Payment did not produce the authorization. Warehouse did not define a reservation lifecycle. The id was minted, not earned.**

# Make architectural drift visible

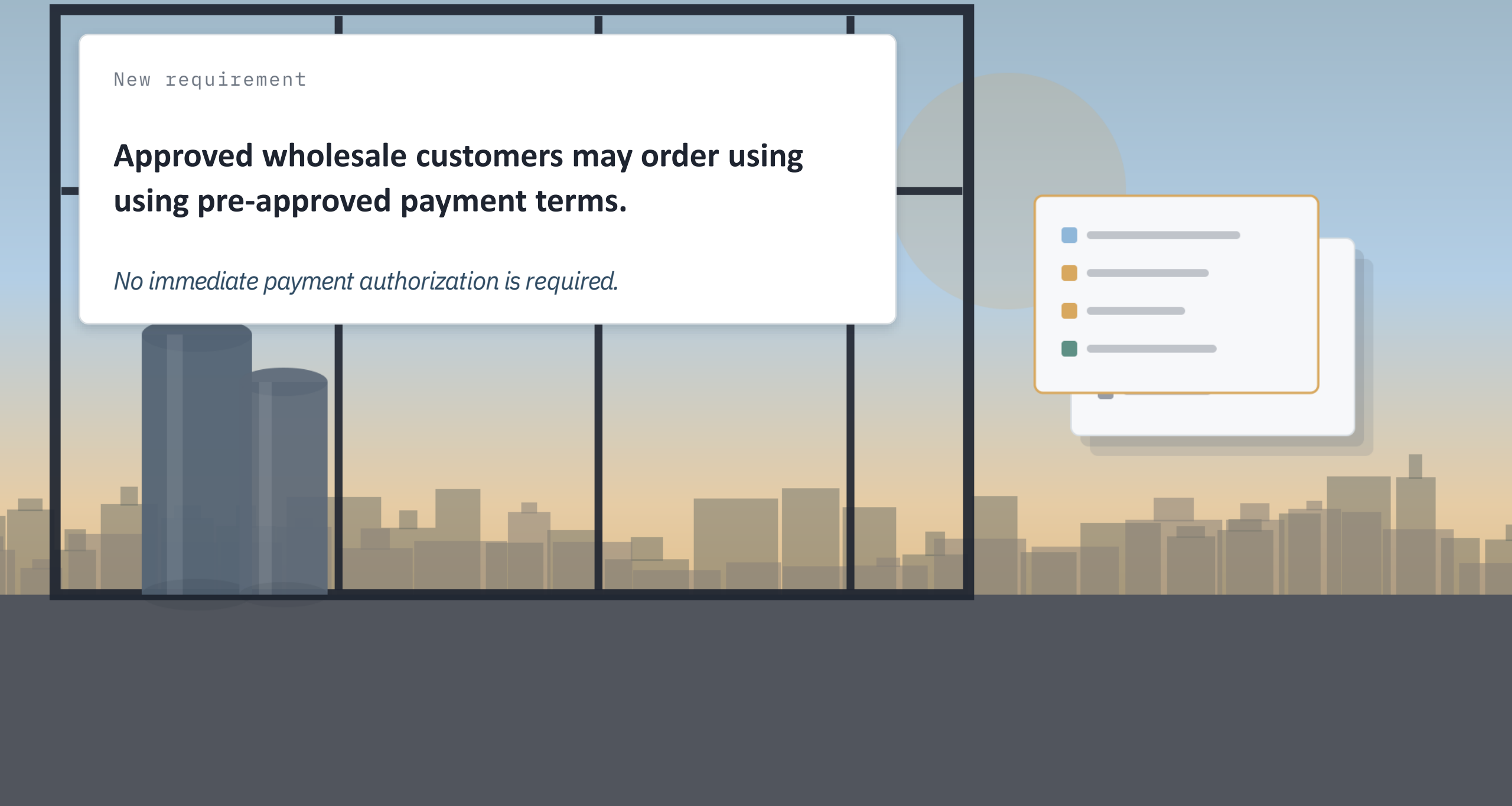
```
/speckit.analyze

CRITICAL — Invented external decision evidence

 PaymentAuthorizationId and StockReservationId are recorded,
 but their domain meaning is not produced by the owning
 authorities recorded in the specification.

 BC-003  Payment owns authorization outcomes.
 BC-004  Payment Authorization is produced by Payment.
 BC-006  Stock Reservation is produced by Warehouse.
 BC-007  Warehouse owns physical stock and reservations.
 BC-011  The agent must not invent reservation lifecycle,
         timeout, retry, void or release policy.
```

**SDD does not guarantee obedience.** It makes architectural drift inspectable.


New requirement
Approved wholesale customers may order using using pre-approved payment terms.
No immediate payment authorization is required.

# What changes first?

The specification changes first — and shows the blast radius.

A **Specification**

B Plan

C Code

spec.md · Confirmation Policy

```
Confirmed when:
  - stock has been reserved
  - either payment is authorized
    or valid payment terms are recorded.
```

/speckit.analyze · after the change

**HIGH — Requirement no longer covered**

```
Task T-18 requires PaymentAuthorizationId for every Sales Order confirmation.
The evolved spec allows PaymentTermsApprovalId for eligible wholesale customers.
```

# The architect as context governor

| Still necessary | More explicit with agents |
| --- | --- |
| Designing components and boundaries | The same boundaries, encoded as decision authority |
| Communicating decisions | Decisions made persistent and checkable |
| Reviewing the design and the code | Reviewing the whole artifact chain |
| Resolving ambiguity during implementation | Exposing ambiguity before execution |

## Not deterministic output — verifiable decisions.

**The architect designs the context within which the AI is allowed to operate.**

A prompt describes a task.

A specification records a decision.

Architecture makes those decisions survive implementation.

**The prompt described the workflow.**
**The specification defined who had the authority to perform it.**

# Thanks!

aacerbis

alberto.acerbis@intre.it

alessandrocolla

alessandro.colla@evoluzione.agency