Introduction

Question: Have you ever been in trouble to apply or grasp DDD concepts in your code?

Did you ever feed in pain to make the link between DDD concepts and your code?

Well... you are not alone, right Bruno?

Absolutely, I hjave to confess: I was in the same situation. I've seen a lot of very good talk about DDD. But as a developer, I felt some difficulties to apply them in my code bases. More Particulary in Legacy situations.

Question : Yes, who here have been already involved into a Legacy Situation?

(pause - smile)

Reason why you would liked to propose a live-coding dedicated session with legacy as the starting point.

Yes Thomas, when I called you last winter to propose you such live-coding session in Paris, That was my intent indeed.

Ok, but first thing first: let's introduce ourselves. Bruno?

I'm Bruno BOUCARD, I'm working in Paris at 42skillz, the company we've founded together. My current job is technical coach for developers. I teach TDD, clean code, Refactoring techniques, DDD among other things.

Well, I'm Thomas PIERRAIN, co-organizer of the DDD meetup in Paris. I'm obsessed with use cases (vs. solution first approach) and I'm basically a CRUD application killer 😉

Ok, before we start, we have some gifts for you.

Yes, 2 reference books. 2 bibles 😉

- Eric's one about DDD and
- Martin Fowler one about refactoring

To win one of themn, you will have to tweet as hell with the hastag #EDDD during our session. We will need a referee to give us the name of the 2 winners at the end of our session. Ok? Well, let's go!

Question: who has already took a train in his life?

Good. Our domain for today is: "Train seats reservation". In particular: optimization of seats reservations within trains.

We will start from a legacy application that we have just discovered this morning.

Question: What would be your first action to discover a legacy app coded by someone else?

Us: we don't live dangerously. WE NEVER REFACTOR A LEGACY CODE BASE WITHOUT A SAFETY NET!

Reason why we have just introduced a test harness before our session.

Ok, let's get back to our case.

We have just been hired by the TrainTRain company (a startup). They asked us to investigate *why they are currently paying too much fees to the official national train operator*.

Indeed?, this start-up compete with others to offer the best possible user experience for train reservation.

Let's have a look at their contexts

Bounded context diagram (with HTTP links)

The offered service is pretty much simple: you provide a train id and the number of seats requested, and you get back a booking reference and the list of the seats reserved.

Bruno, can we do a demo?

Actually TrainTrain is providing a web api. Let's call it to see how it works.

- Demo
- Demo with break point

Let's dive into the code!

Question: What would be your first action? Where would you start for such code review?

In our case, we will start with the acceptance tests we have made this morning.

As you will see, all our acceptance tests are based on the same pattern:

- we don't call any HTTP requests, but are using fakes for external services instead.
- The system under test is the WebTicketManager considered as a black box: we send input, and check the output
- First test: it's the happy path: We reserve 3 seats in an empty train of 10 seats
- Second test: is to highlight a reservation failure. Here we have a train of 10 seatzs with 6 already reserved and we ask for 3 seats. According to our domain expert, this should not work due to a business constraint: WE ARE NOT ALLOWED TO RESERVE MORE THAN 70% OF A TRAIN CAPACITY. So here with 10 seats, the max allowed is 7 seats. 6+3 => 9 which is not acceptable.
- Third test: was the one where discovered a bug. Indeed, the domain expert told us that a reservation should always be in the same coach (to avoid to split families and friends). Here we have spoted a case where our system reserved us 2 seats in 2 different coaches (here 10A and 1B). Instead, we should have expected to have 1B and 2B.

VERY INTERSTING: because this problem should be the root cause of the extra fees. Indeed, imagine you have requested 2 seats for you and your friend. Once you realize you will be separated, you will most likely cancel the reservation. And since all the external services calls are charged to TrainTrain, they will be done for nothing in the first reservation attempt.

It's pure waste for TrainTrain startup perspective. **So we are here to fix that...**

Question: How would you start?

The code we have just seen this morning is really crappy. Thus we need to clean the mess before being able to fix the bug.

We need to refactor, and this is where our test harness will help.

To do that, we first have to ignore the falling test. Otherwise we would never seen if we break the existing code or not.

Ok, let's go!

Let's see how the reserve method works.

1st pass

2nd pass

-