2021.09 – "Planes" solution code review - does it utilize anything from DDD?

When expanding from single responsibility principle we need to keep to a responsibility that an entity **ought to retain.**

- Plane should know **how** to update itself – ok (Start, Update public methods)
- Plane should not know **when** or **why** to update itself - ok (PlaneLifetimeManager knows that)

Domain objects - *Plane, Airport*. Entities should:

- Have Id – ok (Name property, inside contract)
- created with domain life time In mind, domain objects with rich business logic over state - yes
- public methods, private setters, properties settable only through methods - yes
- entities are not immutable like value objects - yes
- constraint how it can be created by adding certain constructors – has non default constructor
- static factory method instead of putting heavy logic in constructor, helps to avoid inconsistent state when doing properties validation tere (decided not to use it for now)

Services – place for behavior that does not belong elsewhere.

- SelectNewDestinationAirport **ought not to** be In Plane domain object – not ok, extract away to some service that will hold logic for determining how to switch destination airport
- Bad Weather randomizer method in Airport – not ok, extract away to some service

Bounded context

- **ubiquitous language** - project complexity too low for boundaries between bounded contexts - somewhat exists in object naming and namespaces
- possible to distinct 3 basic Bounded Contexts for the future, with separate architecture, responsibility and deployability: Plane and Airport (hosting, domain object and object lifetime manager in each) and API with layers for http controllers and service for storing traffic info.

Shared Kernel

- *Utils project* - ok - logic used across projects but not belonging in any of them, too small for a separate project *(http client, geo navigation, serice random name generator*)
- *Contract project* - some common contracts between bounded context is needed - partially ok - each bounded context / layer inside project should have only info that it needs for its work. Can be improved - eg.: color info not needed everywhere. Plane should have its Name private property, not a contract. Contract like structure with plane info should be created and returned by API in methods for UI. Similar method with airports but intended for Backend Plane should have list of airports with names and coordinates, not every property in airport. Same model everywhere simplifies code a lot and may be good for small project complexity, but the big code smell that it causes is that the projects and services seem to work a bit like a distributed state machine.

Problem Domain, Core Domain, Sub-Domains - not thought about
Value object – immutable - not used
Aggregates - not used, maybe it d come in handy in model refactoring as we have associated objects
Anti corruption layers - not needed, just do model refactoring right

Design allowed to relatively quickly remove whole mqtt logic and replace i with using http api.
Didn't get stuck in perfection zone like Eric Evans advised.