Deep Space Orbiter

odyssey in code

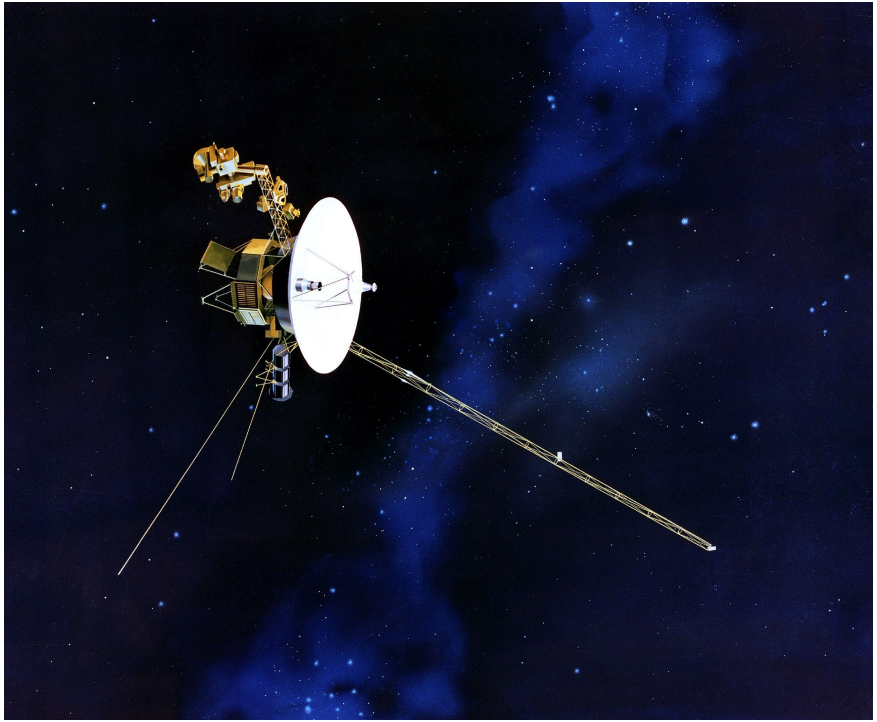
Fabian

Idea

- Implement something with interwoven / complicated “business logic”
- Try Solo development
- Experience evolving requirements
- Experience “the obvious”

And

- Share my experience



So it had to be complicated

```
16  /**
17   * 2D Space for simplicity
18   * <p>
19   * Swiss Space Force launches an Orbiter towards Mars
20   * <p>
21   * The Orbiter starts with its camera point to BACKWARD and its antenna FORWARD
22   * <p>
23   * The antenna and camera can not point in the same direction
24   * The camera can only record if the antenna faces a planet
25   * Te camera should record the closest planet as long as possible
26   * (thus the camera should record Earth and after half way record Mars)
27   * The camera can  record 2x1 Mkm worth of flight and when it's full it has to transmit
28   * (the camera recording can not be interrupted, it will always record 1M km worth of footage)
29   * The antenna should try to transmit if it's facing earth if the camera has recorded something (and the camera is not recording)
30   * The camera should no longer record if it is full and should instead turn away for the antenna to transmit.
31   * with 100mW
32   * and after 20 km with 200mW
33   * <p>
34   * <p>
35   * The navigationsystem will tell the orbiter current time and distance from the planets.
36   * <p>
37   *
38   *                    [         ]
39   * o          BACKWARD [ Orbiter ]  FORWARD         O
40   * Earth               [         ]                  Mars
41   * <p>
42   * <---------------395.04 million km ------------->
43   * <p>
44   * For simplificaiton distance is equal to time: Every Mkm = 1h recording
45   * The orbiter can always receive/communicate, the antenna is only required for transmitting "HD Video Recordings".
46   */
```

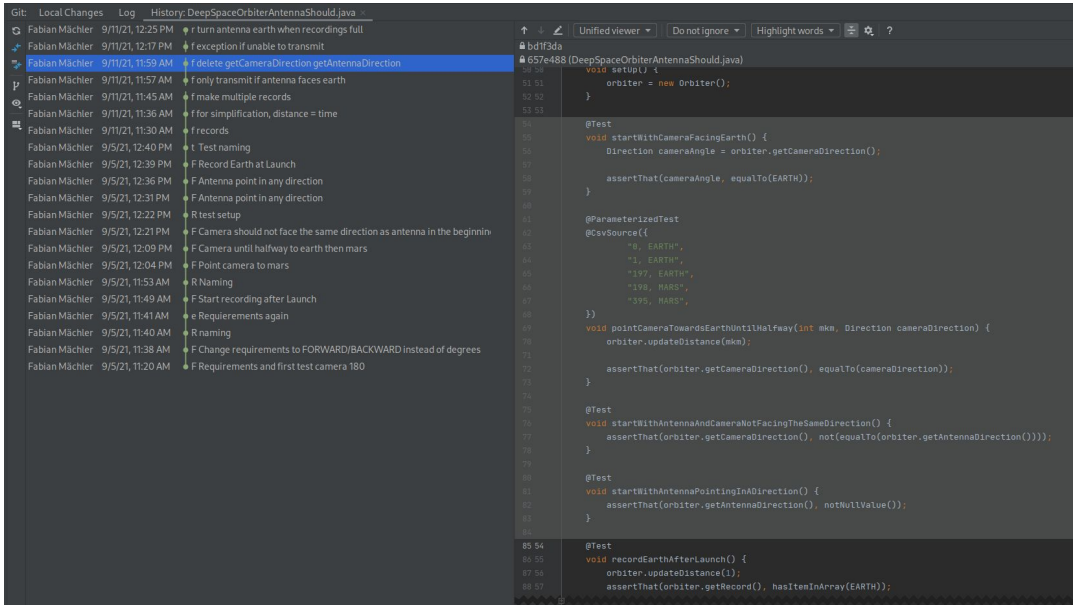
Call Getters in Tests? -> Smell?

- getCameraDirection() and getAntennaDirection() in Testcode
- “Forward-” and “Backward Facing” are internal concepts
- So is Direction Earth and Mars
- Find another way to test it
  - We can observe the received recordings instead
- -> Write new tests

```
@Test
void recordEarthAfterLaunch() {
    orbiter.updateDistance( newDistance: 1);
    assertThat(orbiter.getRecord(), hasItemInArray(EARTH));
}
```

And the nice part

- Delete a lot of old code



Drastic Implementation change without changes in the Tests

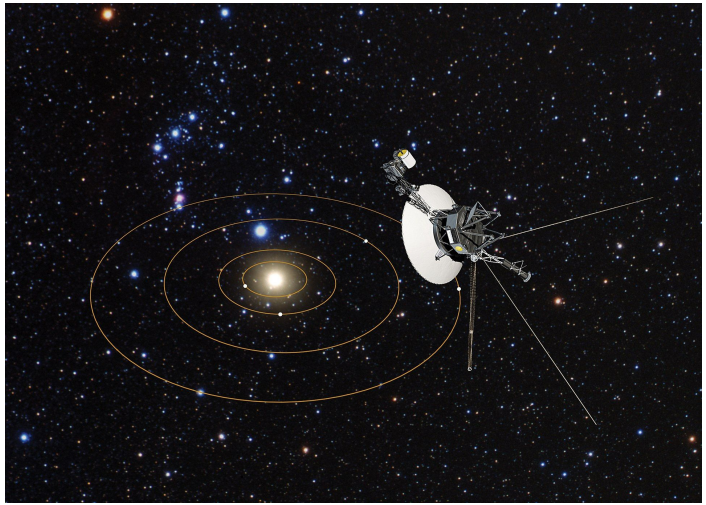
- I could change from “CameraDirection and AntennaDirection” to “Slots which a Devices occupies” without changes in the tests

```java
public class Orbiter {

    public static final int DISTANCE_TO_MARS = 395;
    private int distance;

    private Direction cameraDirection;
    private Direction antennaDirection;


    private final LinkedList<Direction> records = n

    public Orbiter() {
        cameraDirection = EARTH;
    }
```

```java
public class Orbiter {

    public static final int DISTANCE_TO_MARS = 395;
    private Device directionEarthSlot;
    private int distance;

    private Direction cameraDirection;

    private final LinkedList<Direction> records = ne

    public Orbiter() {
        directionEarthSlot = CAMERA;
    }
```

State of the Project

- About half of the requirements are implemented
- There are refactorings in the production code open
- But the Tests/Requirements seem to be in good shape to me



Other observations

- First degree of freedom: Focus on camera direction, not the recording
  - But it’s connected (See Getter-Smells)
- Shaping the requirements involved a lot erroring and thus learning
  - Did I make myself to be the only expert in this field?
- Lot of naming is wrong (I didn’t notice)
  - Maybe the Mob would help here too
- Change of requirements during implementation
  - Implementing challenges the requirements
- DirectionEartSlot -> I have nothing telling me how to implement this
  - Try and error?
  - Train the production code like a neural network?

Bonus Slide: Questions along the way

- How do I test the camera and antenna not facing the same way?
  - Expect an Exception if they collide?
  - This could happen in any case
    - Add Assertion in each Test? (they should only test one thing)
- Exceptions to the rescue?

```
@Test
void shouldThrowException_IfTryingToTransmitAndUnableTo() {
    orbiter.updateDistance( newDistance: 1);
    assertThatThrownBy(() -> orbiter.getRecord())
            .isInstanceOf(Exception.class)
            .hasMessageContaining("Antenna faces the wrong way");
}
```

Unfixable Tests?

After implementing the Exception (Antenna does not face Earth now) the following test fails:

```
@Test
void recordEarthAfterLaunch() {
    orbiter.updateDistance( newDistance: 1);
    assertThat(orbiter.getRecord(), hasItemInArray(EARTH));
}
```

But we have new Tests covering this too, just much more complicated

```java
@Test
void shouldThrowException_IfTryingToTransmitAndUnableTo() {
    orbiter.updateDistance( newDistance: 1);
    assertThatThrownBy(() -> orbiter.getRecord())
            .isInstanceOf(Exception.class)
            .hasMessageContaining("Antenna faces the wrong way");
}
```

```java
@Test
void transmitTwoRecordsAfter2mkm() {
    orbiter.updateDistance( newDistance: 1);
    orbiter.updateDistance( newDistance: 2);
    assertThat(orbiter.getRecord(), is(new Direction[]{EARTH, EARTH}));
}
```

-> Interwoven and complicated requirements lead to complicated tests...

Code

You can find the code here: https://github.com/unSinn/alcor-presentation