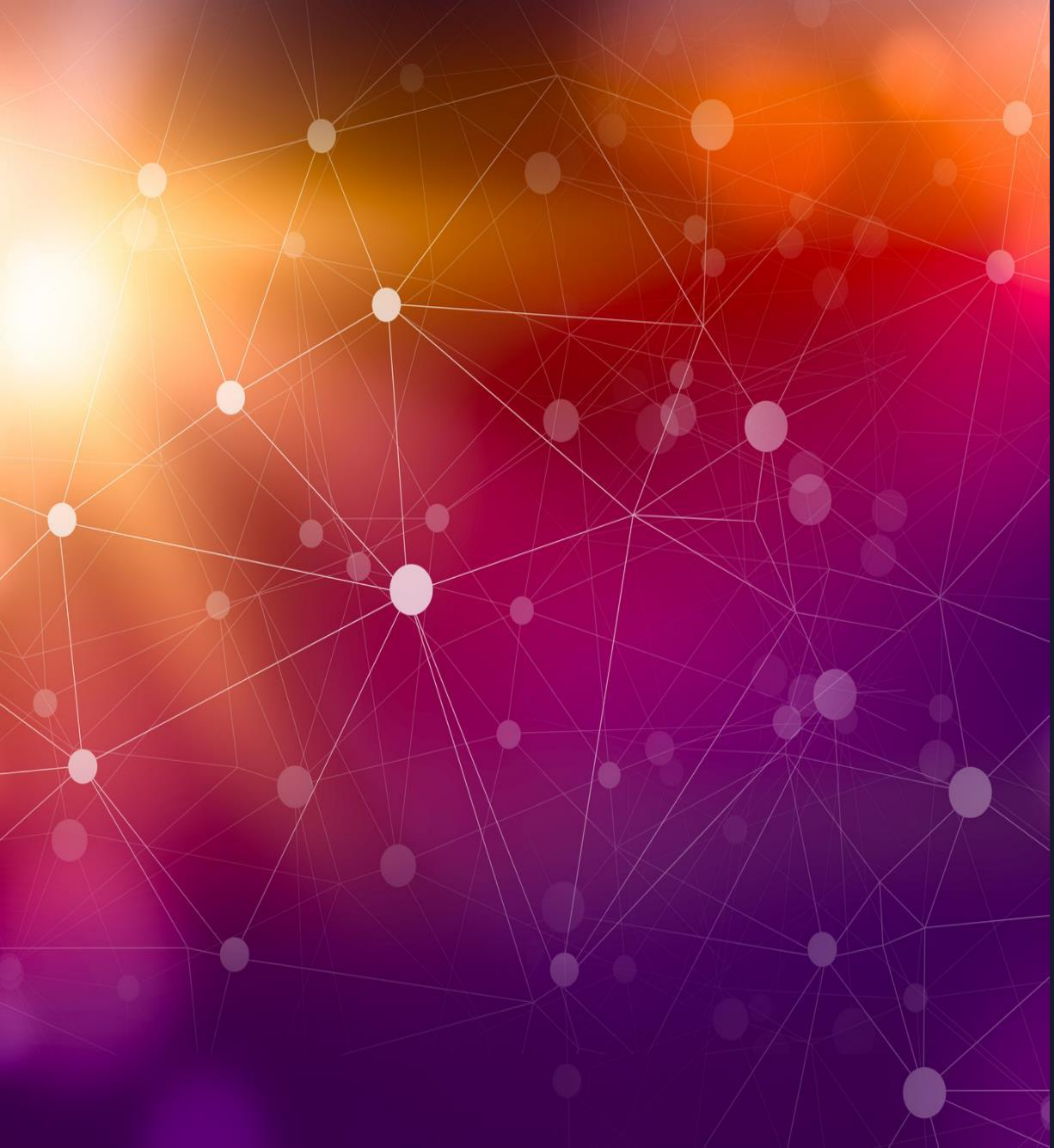


Improvements in the testing of a Golang service from my customer project

Using the TDD principles learned at Alcor Academy

Daniel Garip

Context about the microservice

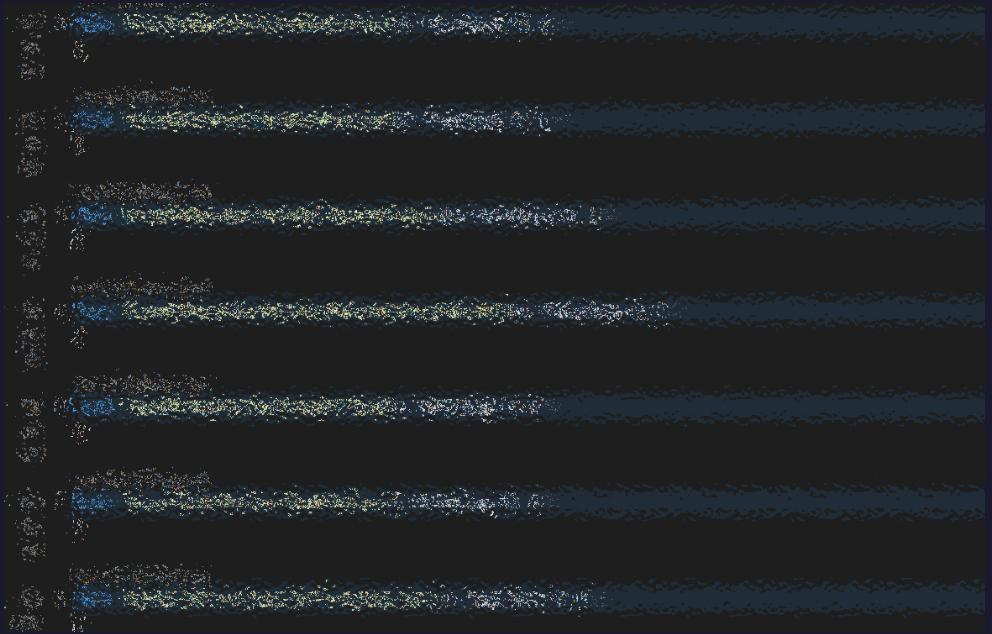
- It is a surface aggregation service
- Input is data called surface objects
- The core of the service is to do a calculation on the input (n surface objects) and return an output object (Simplified)
- The calculation can be: Mean, min, max, std and percentile
- Tried utilizing TDD with pair programming before the course

The current test setup

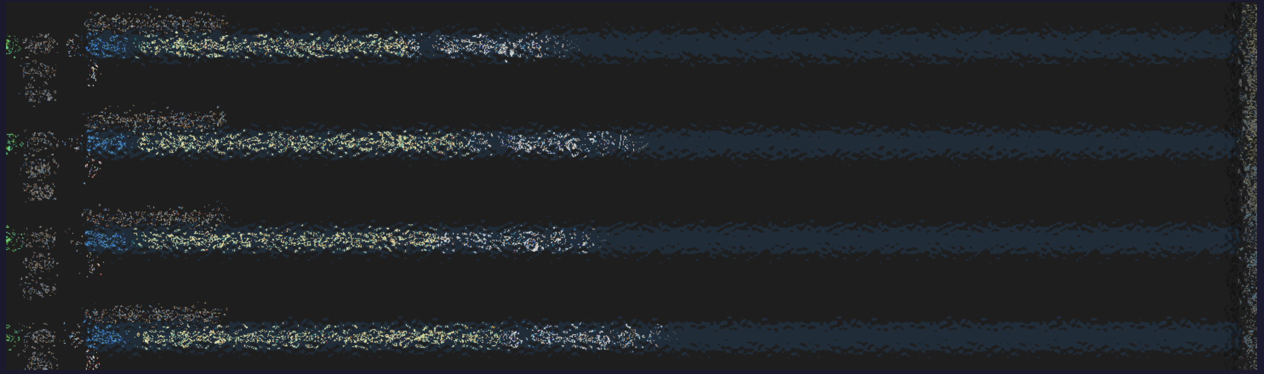
- Tests in own file
- Test coverage 61.3%
- 32 total tests

- While learning about TDD I found a lot of the test smells in our tests
- Not optimal testing and a lot of overlap
- Big room for improvement by utilizing the aspect of correct TDD

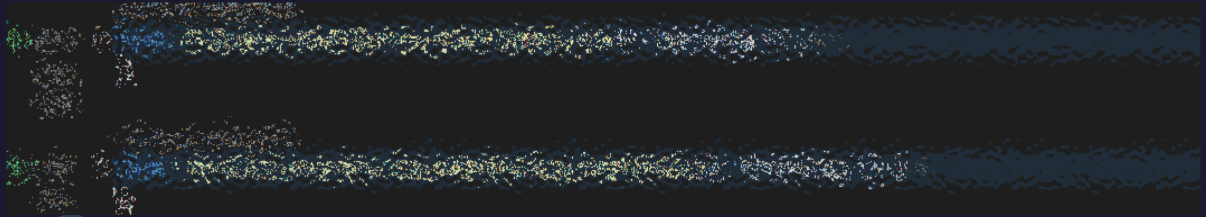
Naming of tests



[illegible]

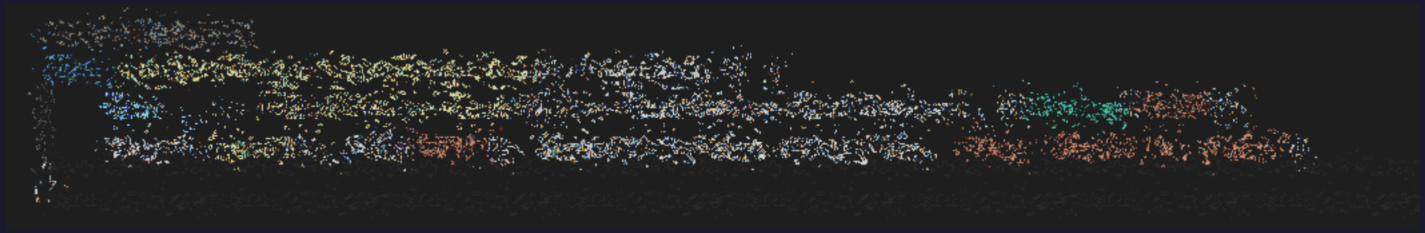


[illegible]

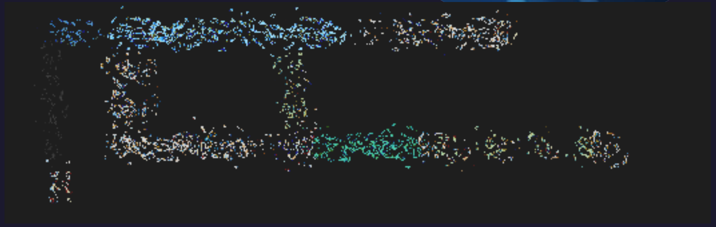


[illegible]

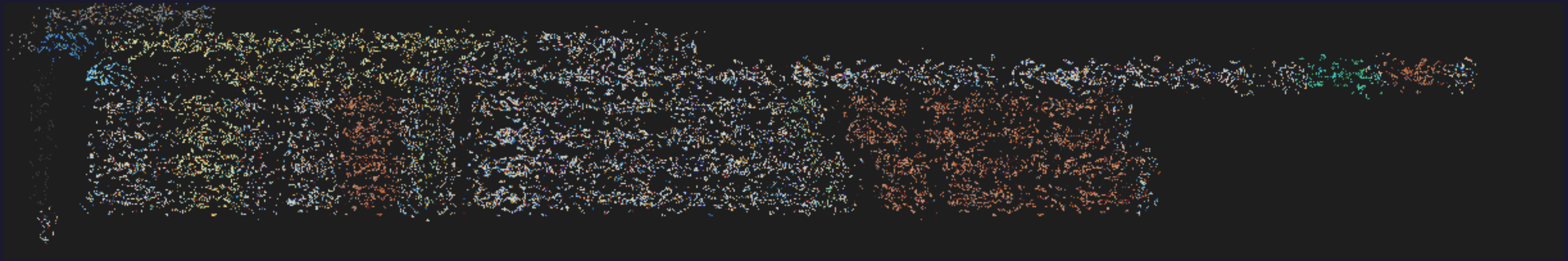
Arrange Act Assert



[illegible]



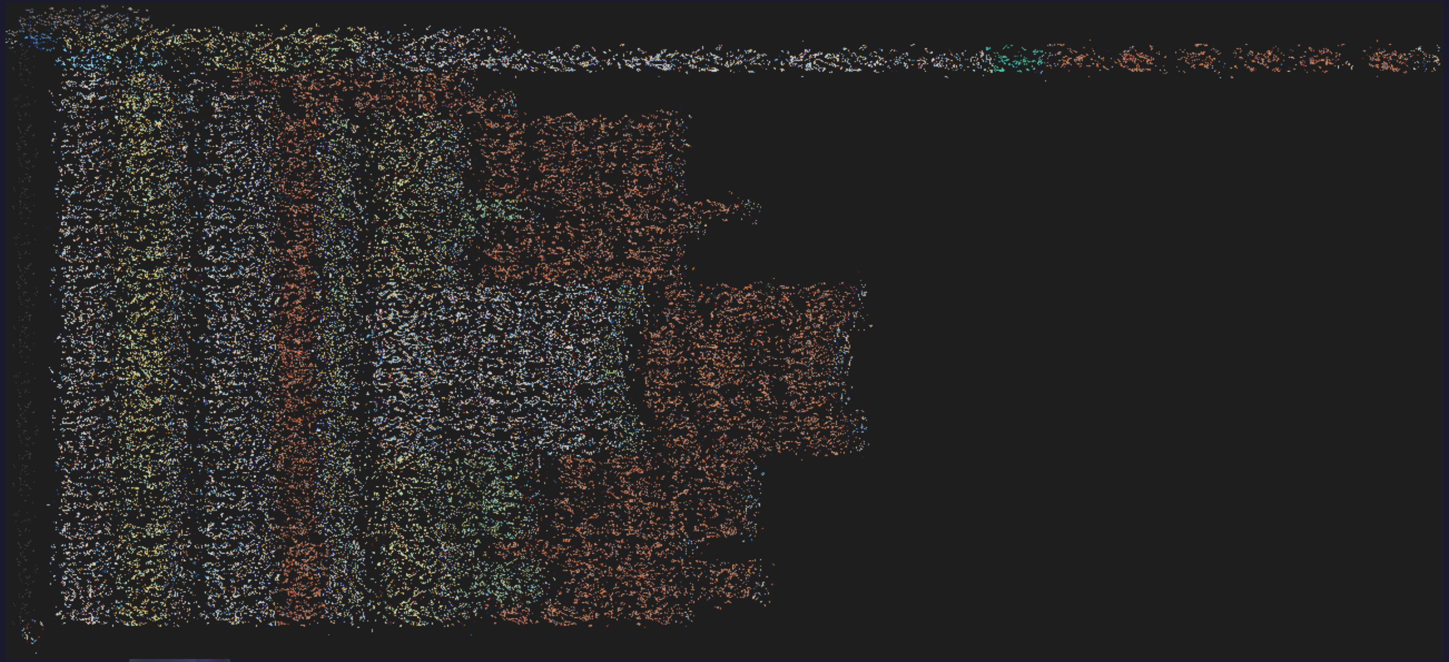
[illegible]



[illegible]

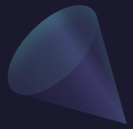
Number of assertions

- This breaks with TDD habits
- Tests should only test one single behaviour
- Only one logical assertion per test



TDD Habits first principles

- Fast. They should run very often, hence they must be fast: one second matter here
- Isolated. They should be no dependency between tests, hence they must run in any order
- Repeatable. They should always have the same result when run multiple times
- Self validating Only two states: red or green.
- Timely. Written BEFORE the code they suppose to test



To sum up

- Our tests are way to big (testing too much in each test)
- We have overlapping tests
- Naming of the tests should be even more specific.
- Have actually been using some aspects of Transformation Priority premise

Thank You

Daniel Garip

Daniel.garip@soprasteria.com

Github: danieltg96