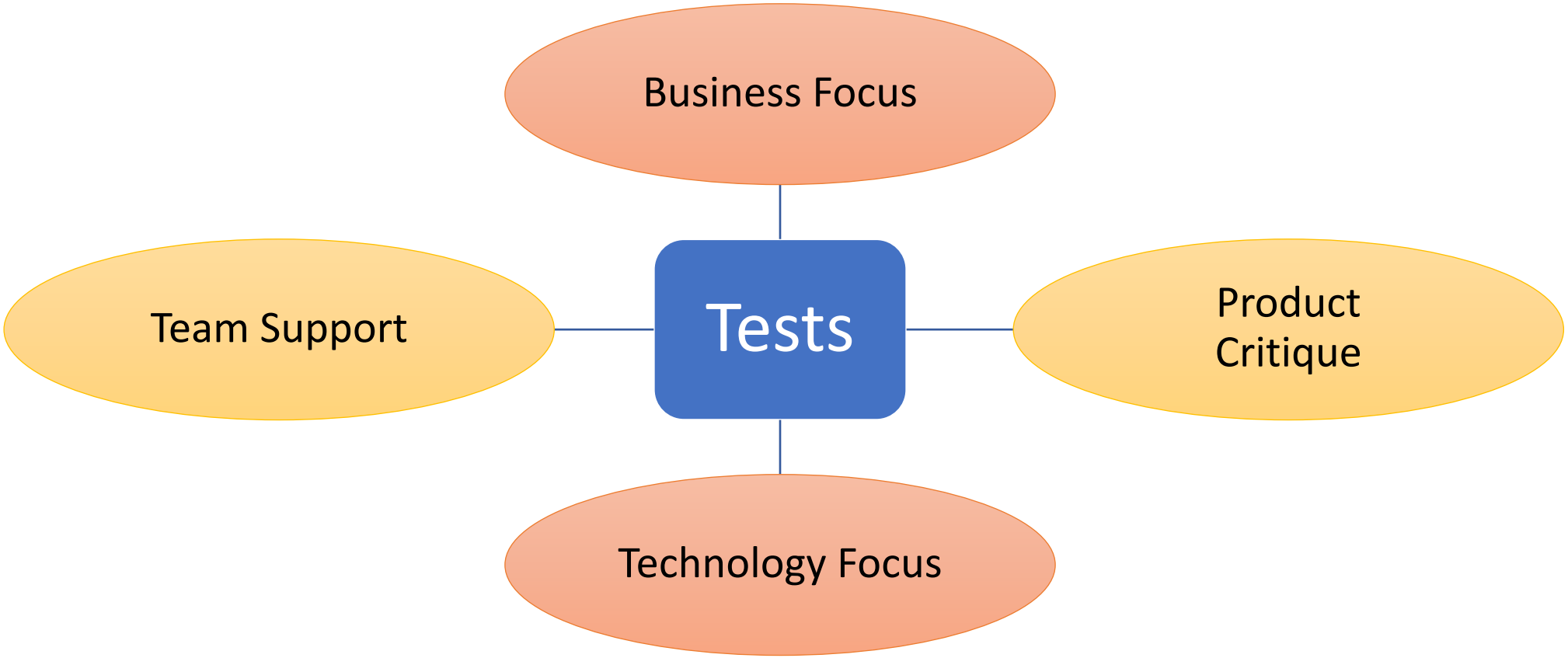
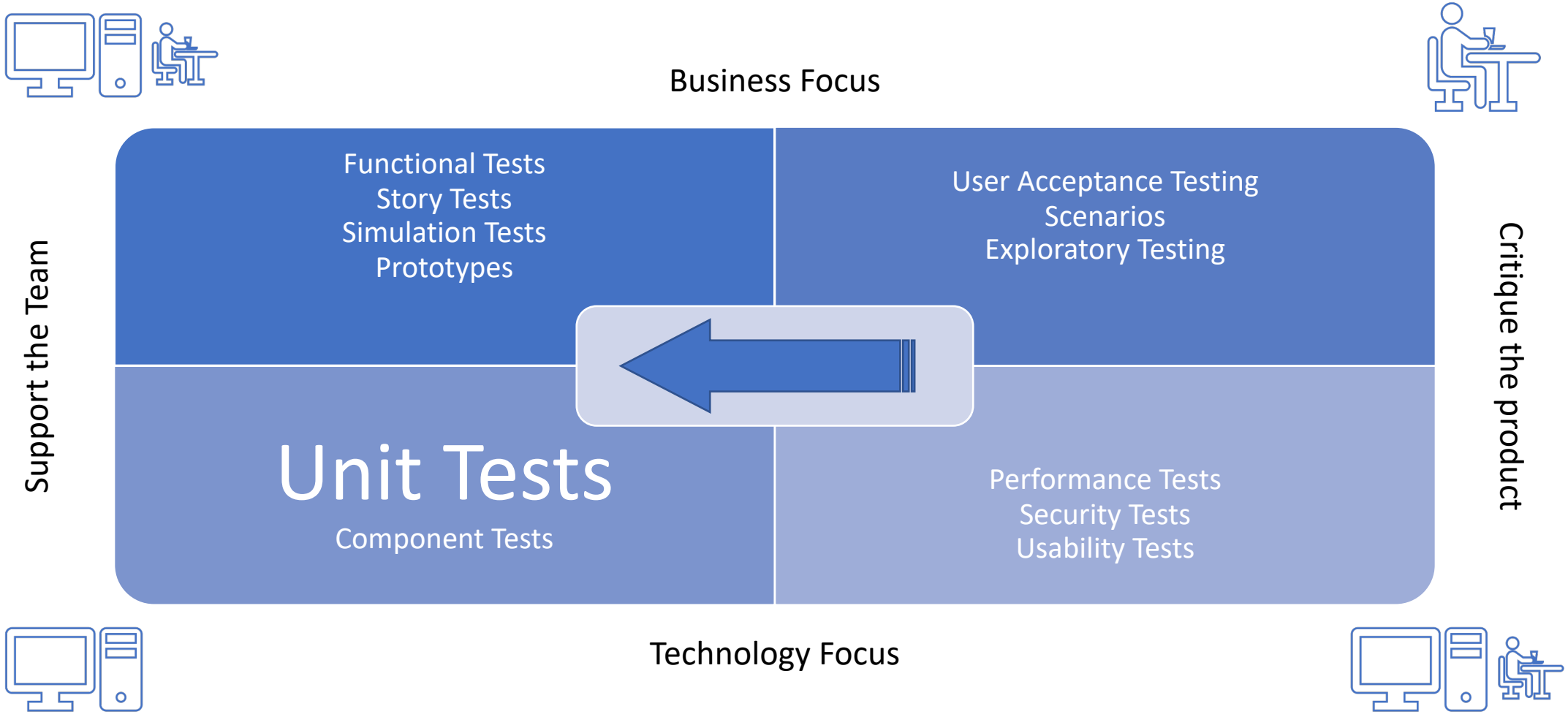
Writing Good Unit Tests

Dr. Matthias Hölzl

Two Dimensions for Tests



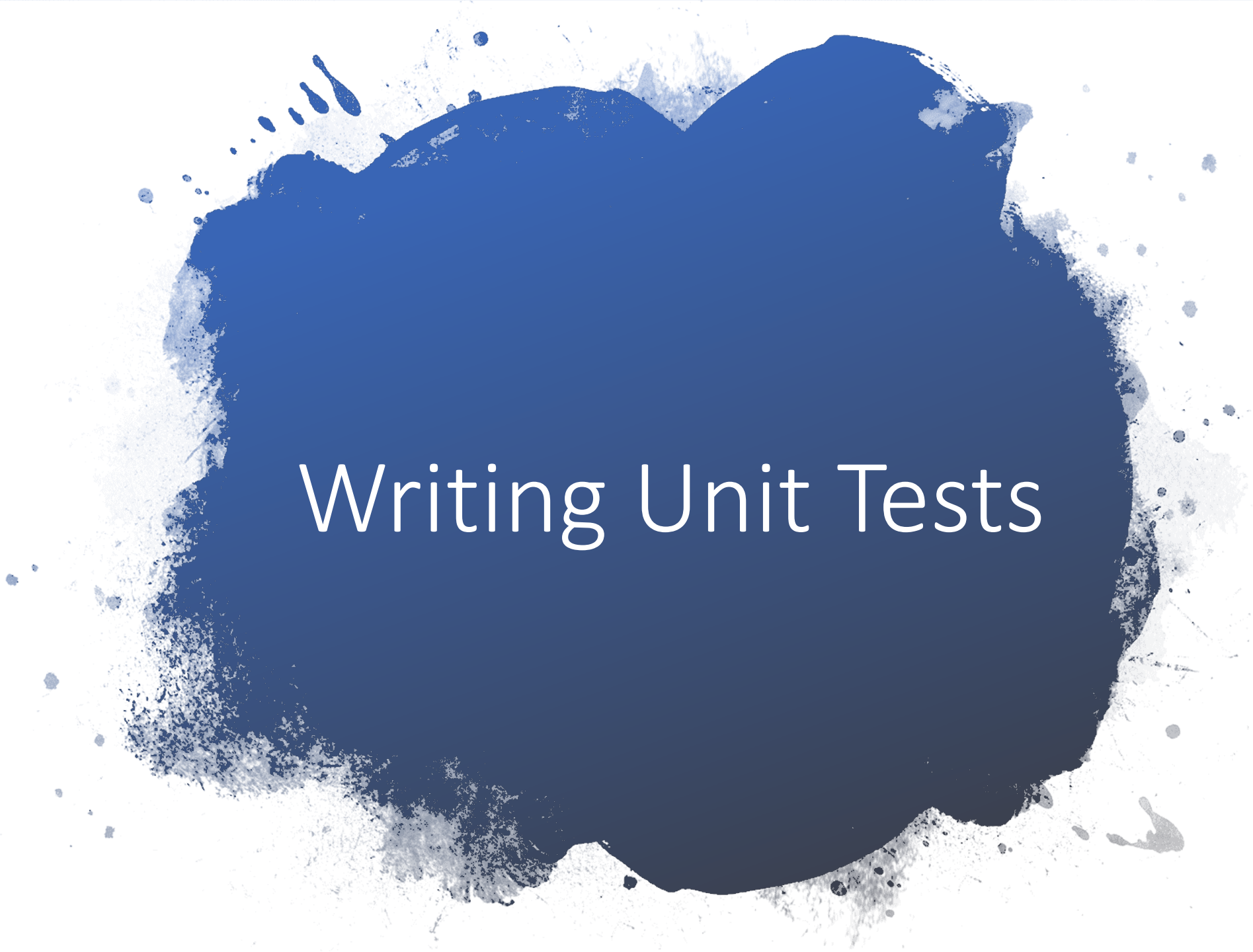
Testing Quadrants



Unit Testing: Why?

Tests should improve the quality of our product and enable us to develop with constant velocity

Properties of Unit Tests


Writing Unit Tests

Format of a Unit Test

- Arrange
- Act
- Assert

- Given
- When
- Then

```
TEST(MySubsystem, Output_WhenScreenIsDefaultSize)
{
    Screen unit{80, 25};
    std::string text{“Example Output”};

    unit.writeText(text);

    ASSERT_THAT(unit.getText(), Eq(text));
}
```

Ideal vs. Reality

- What your test should look like:

- What tests often look like:

Writing Good Unit Tests

Why not?

- Example: Password entry
  - Passwords with max 20 characters
  - 80 possible characters (A-Z, a-z, 0-9, !@#$%...)
  - 80^{20} = 115.292.150.460.684.697.600.000.000.000.000.000.000 Inputs
  - If each test takes 10ns it would take 10^{24} years to test all inputs
  - The universe is ~ 1.4 x 10^{10} Years old

Results, State, Behavior

```
int Add1(int i);


class Adder {
public:
  void SetInput(int i);
  void Compute();
  int GetResult();
}
```

```
class BehavioralAdder {
public:
  void CallAdder(int i) {
    a.SetInput(i);
    a.Compute();
  }
private:
  Adder a;
}
```

Your code has to allow this style of testing