Using Developer Testing to Boost the Quality of Your Code

Jfokus 2018

Alexander Tarlinder

The Goal of This Talk

Map out what areas to explore by throwing in some theory

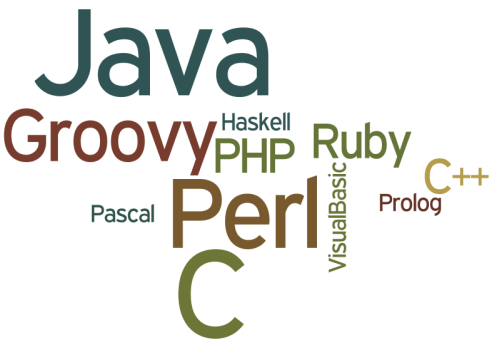
Explain the scope of a developer's responsibility

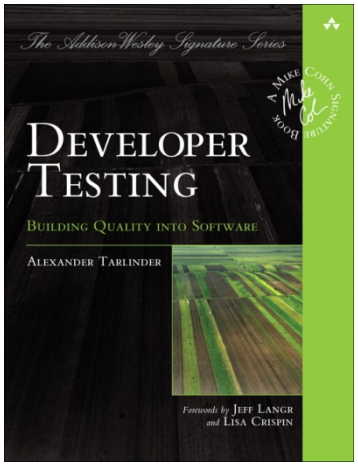
Give s precise quality vocabulary

Describe & sell developer testing

Make you DO developer testing back at home

About Me





http://developertesting.rocks

http://www.techbookreader.com
http://dictionaryofagile.org

Potentially Shippable Product Increment

“Automate everything”

“Testing to support development”

“Build quality in”

“QA should find nothing”

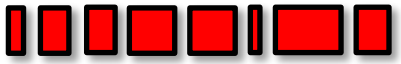
Test-first development and courage

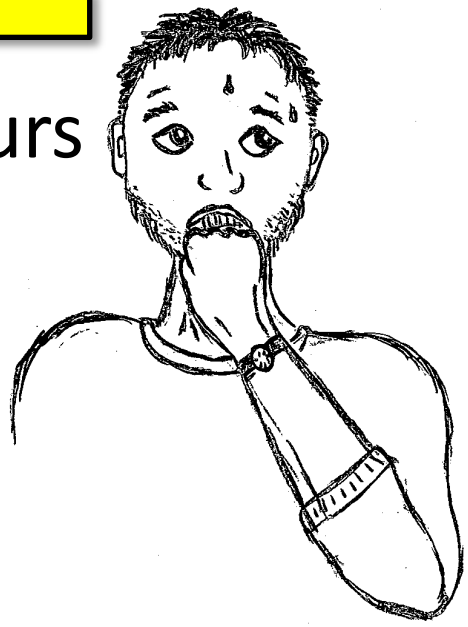
The Clock's Ticking

Scrum: weeks

Kanban: days

Continuous delivery: hours





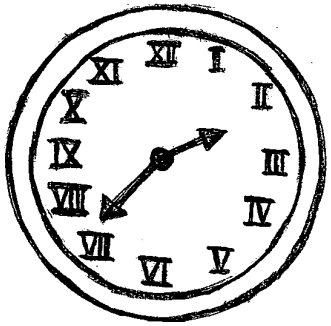
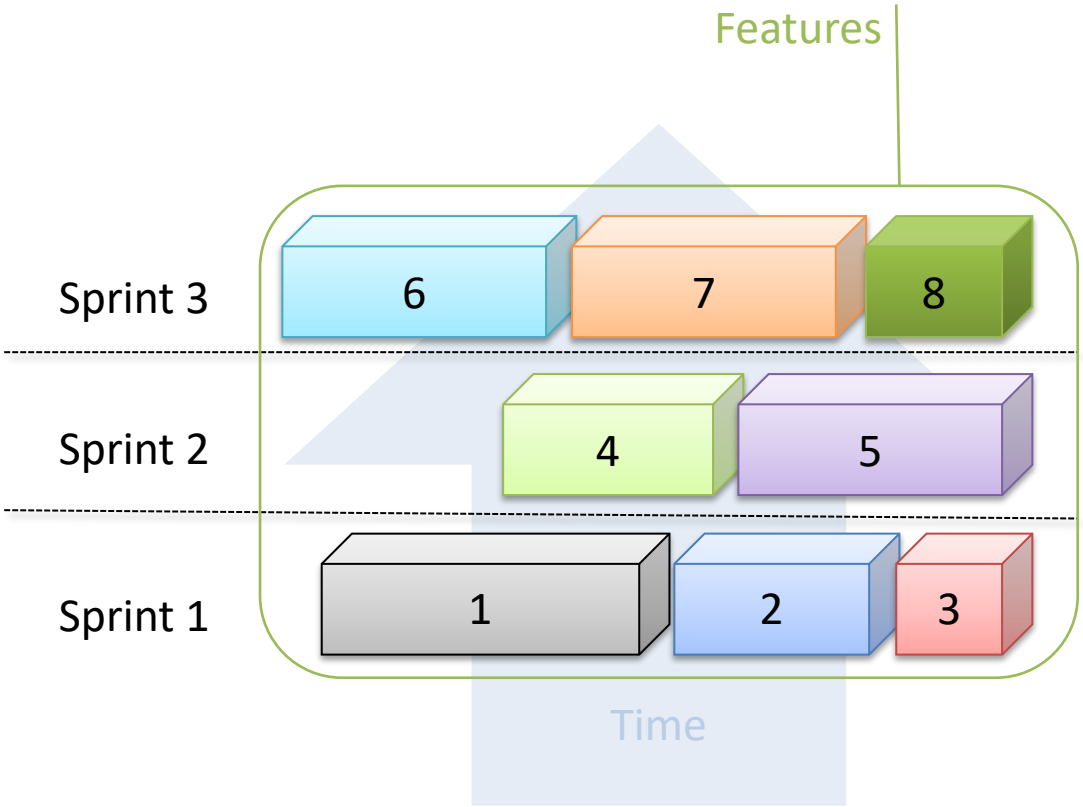
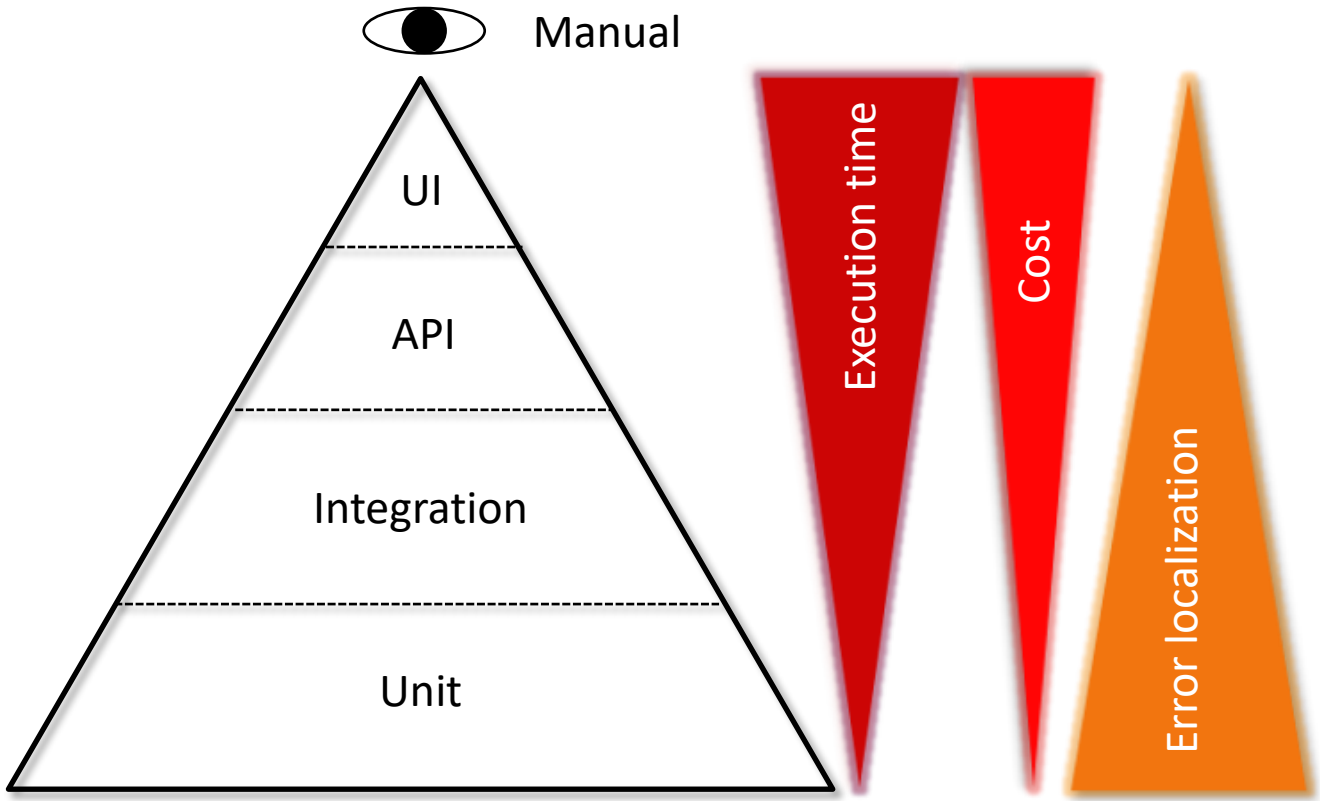


Illustration: Teresia Tarlinder

The Regression Testing Challenge

Sprint	Build*	Test
1	1+2+3	1+2+3
2	4+5	1+2+3+4+5+6
3	6+7+8	1+2+3+4+5+6+7+8
**) Building new features requires refactoring old ones.*		




Manual
UI
API
Integration
Unit
Execution time
Cost
Error localization

Unit Tests

- Fully automated
- Self-verifying
- Idempotent
- Run in isolation
- Fast

Hopefully a recap

Unit Tests: naming

UnitOfWork_StateUnderTest_ExpectedBehavior

`Checkout_LoyalCustomers_Get5PercentDiscount()`

BDD-style: something should

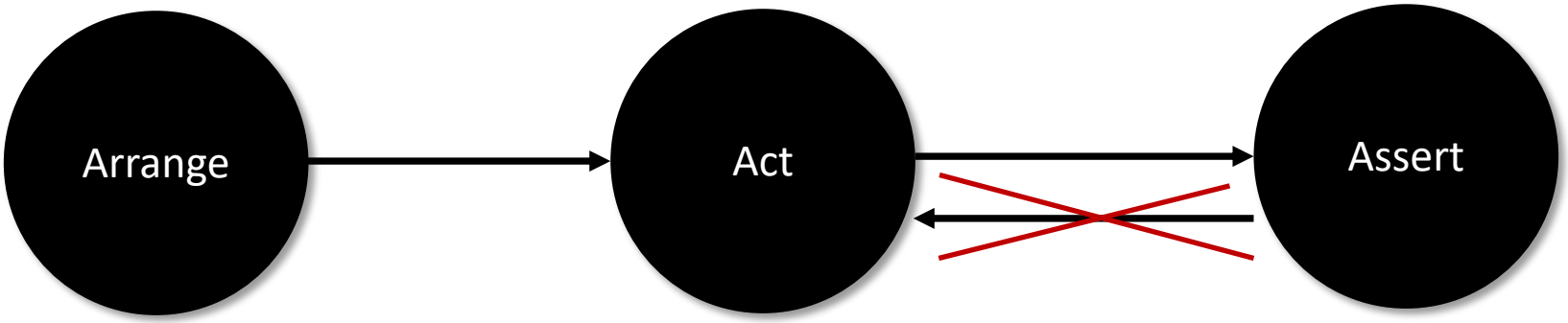
`shouldGive5PercentDiscountToLoyalCustomers()`

A definitive statement

`loyalCustomersGet5PercentDiscount()`

Unit Test Structure

- Arrange → Act → Assert
- Given → When → Then
- Setup → Exercise → Verify → Teardown



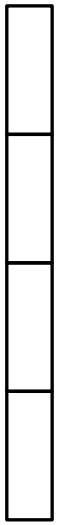
Syntactic vs Semantic Assertions

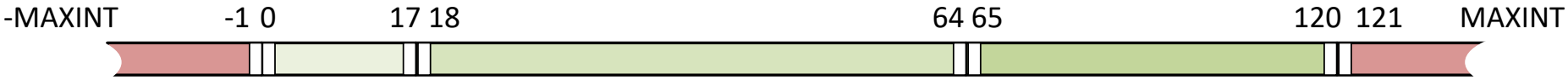
```
assertArrayEquals(new String[] {"Hello", "Jfokus", "2018"},
        "Hello Jfokus 2018".split(" "));
```

```
String[] words = "Hello Jfokus 2018".split(" ");
assertEquals("Hello", words[0]);
assertEquals("Jfokus", words[1]);
assertEquals("2018", words[2]);
```

But how many???

```
If this happens …
Or this …
Or that …
Otherwise just …
```



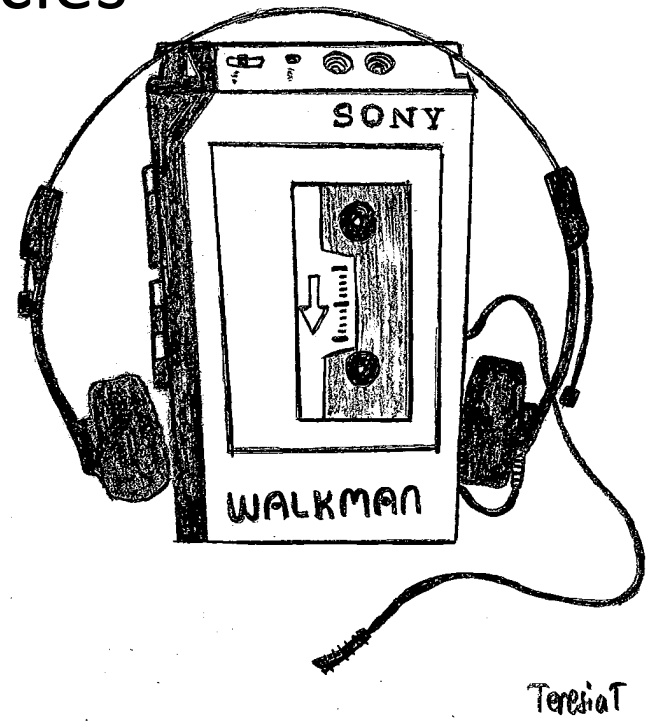


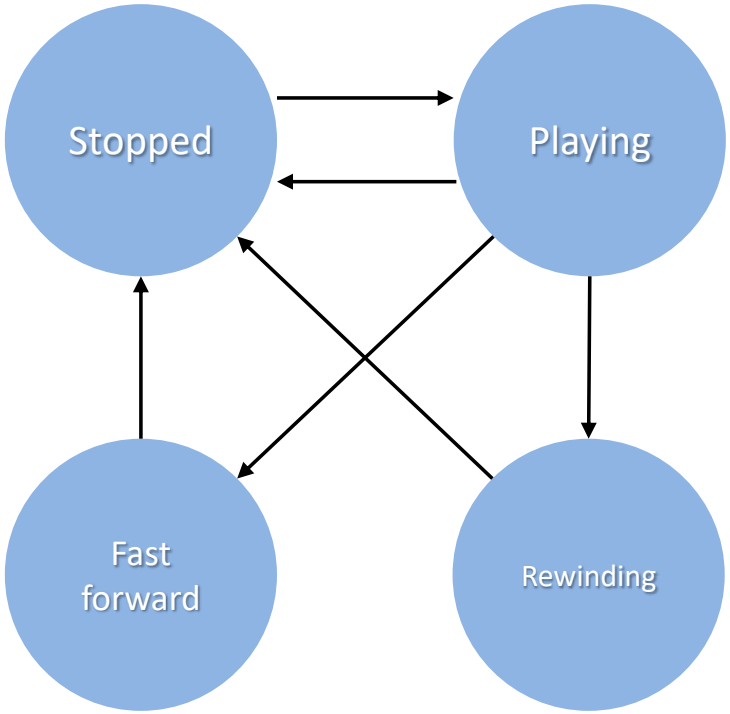
Decision Tables

Age	Investigation	Grant loan
5	—	No
17	—	No
18	Yes	Yes
18	No	No
30	—	Yes
64	—	Yes
65	Yes	Yes
65	No	No
69	Yes	Yes
70	—	No

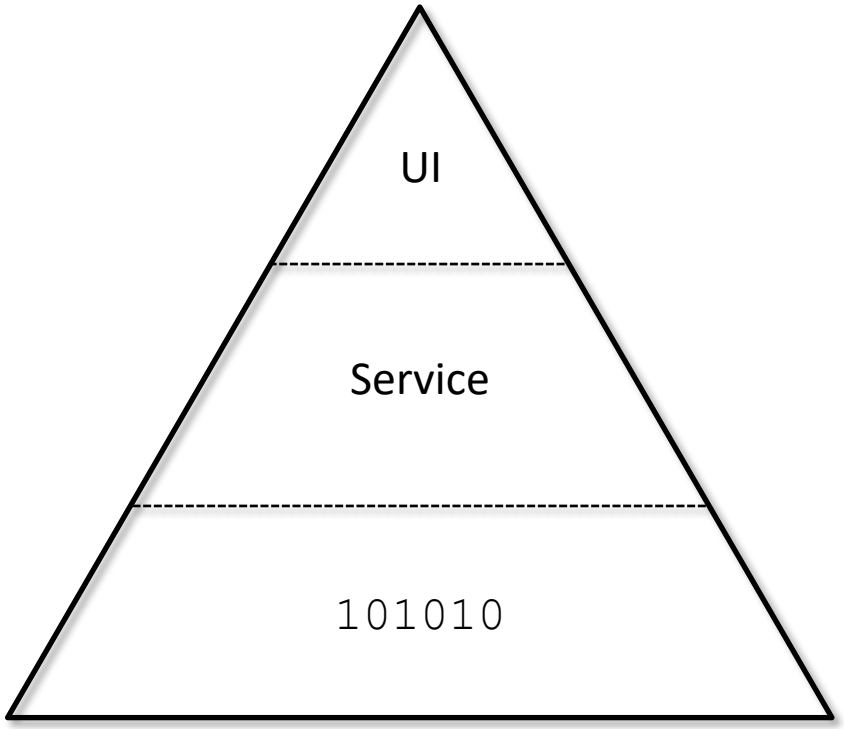
State Transition Testing

- Parsers
- Flows/Wizards
- Lyfecycles

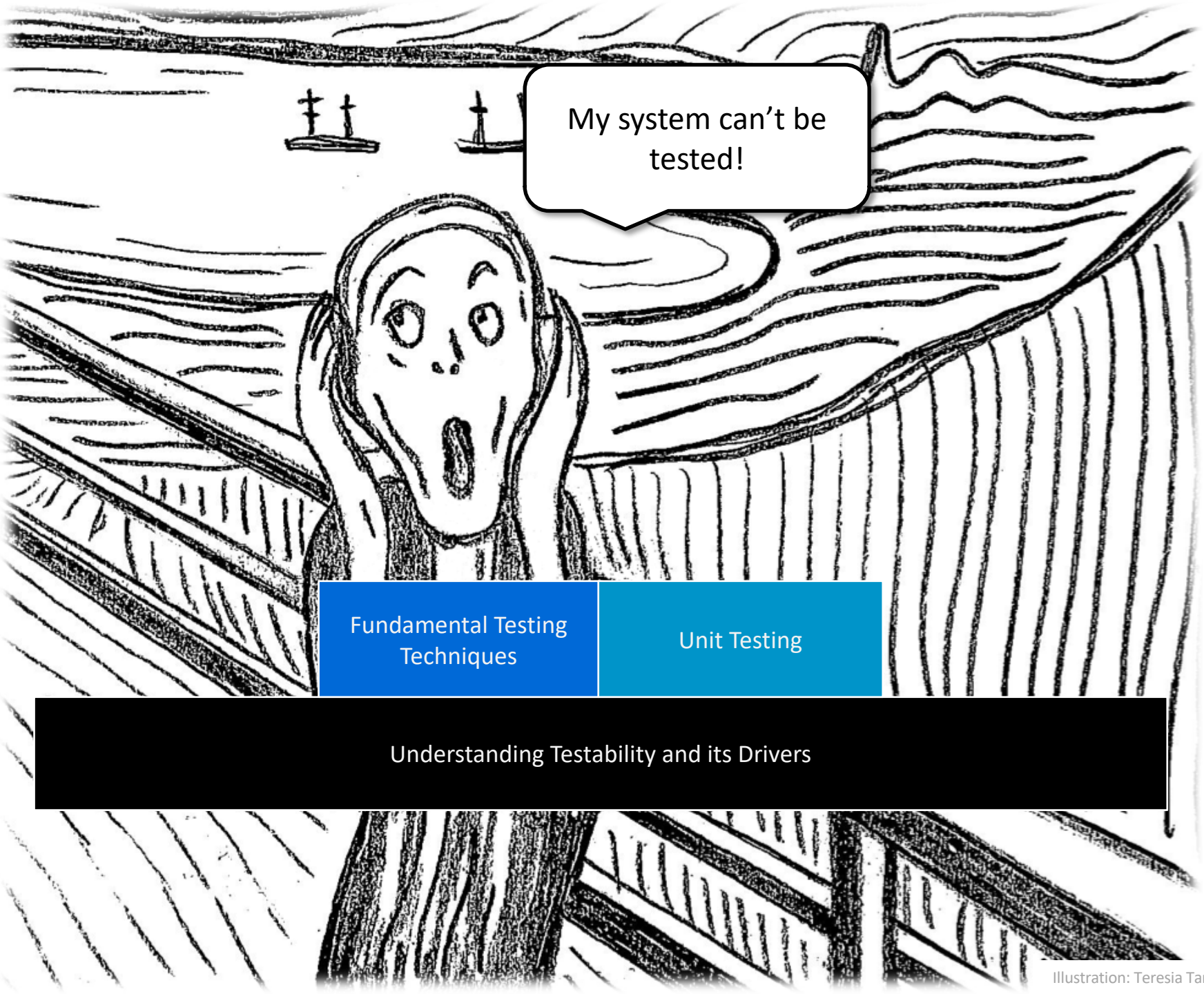




So there you have it!



Fundamental Testing Techniques	Unit Testing



Fundamental Testing Techniques

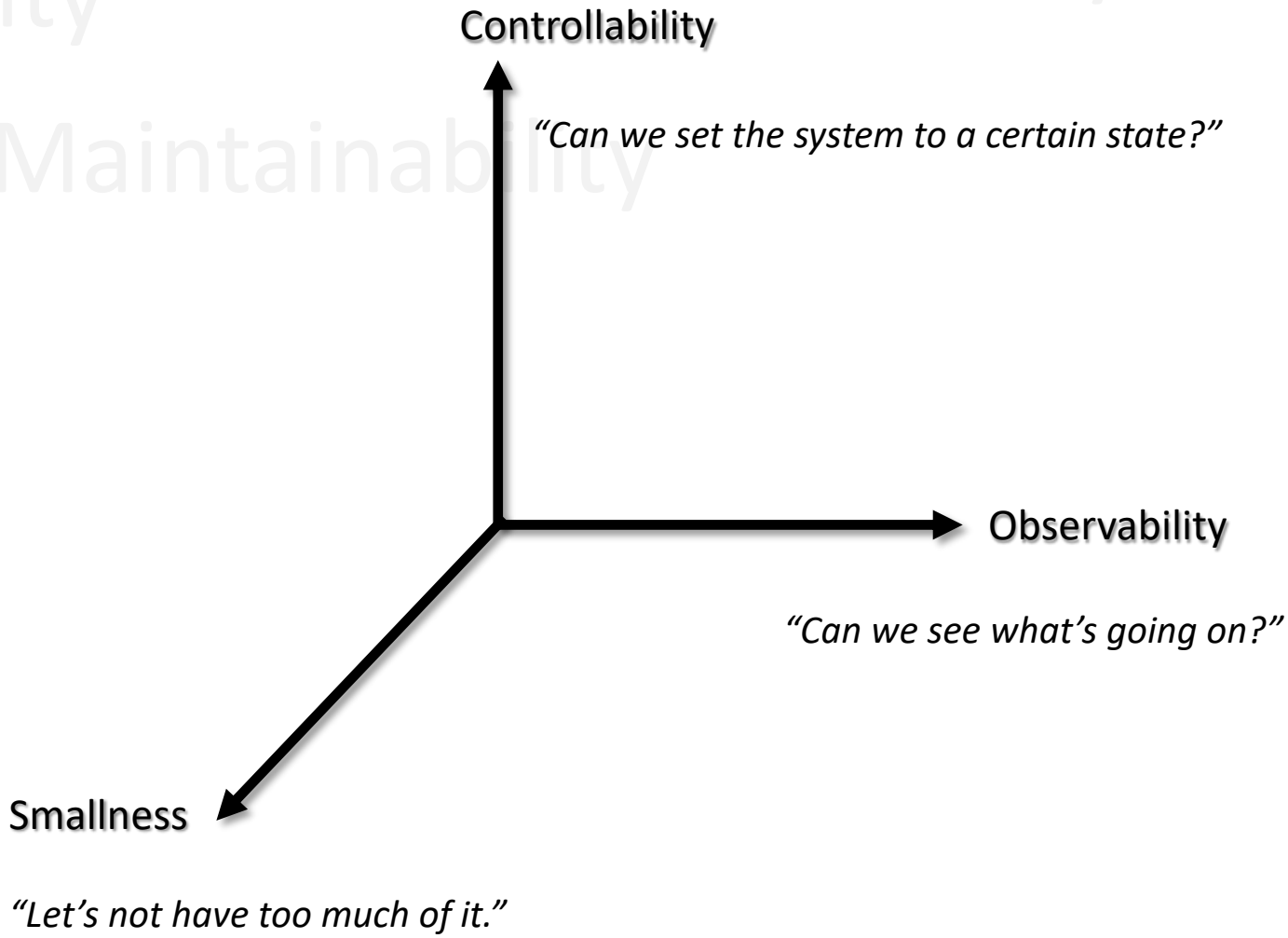
Unit Testing

Understanding Testability and its Drivers

Illustration: Teresia Tarlinder

Testability

Portability Performance Usability Security Maintainability



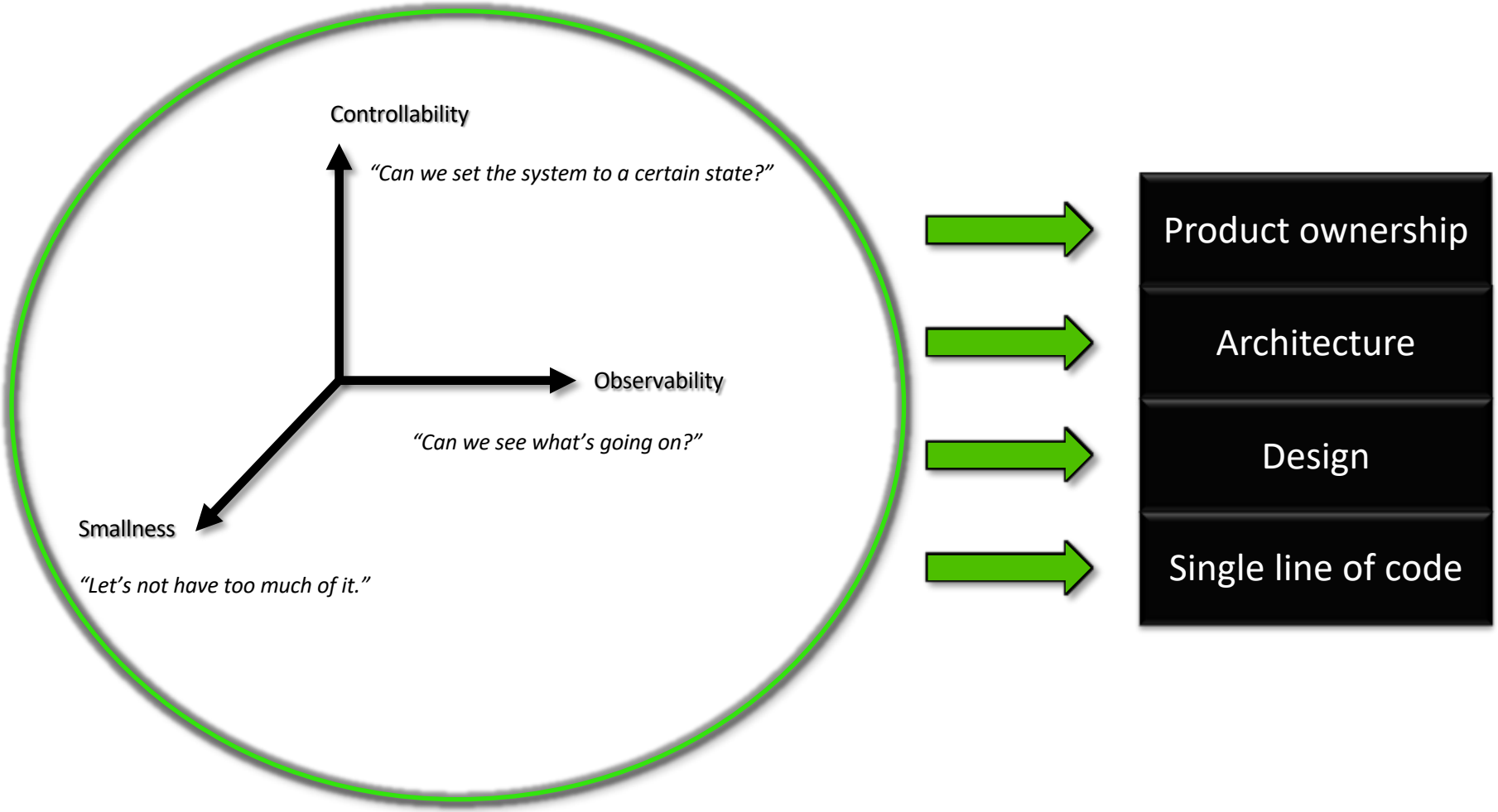
Smallness

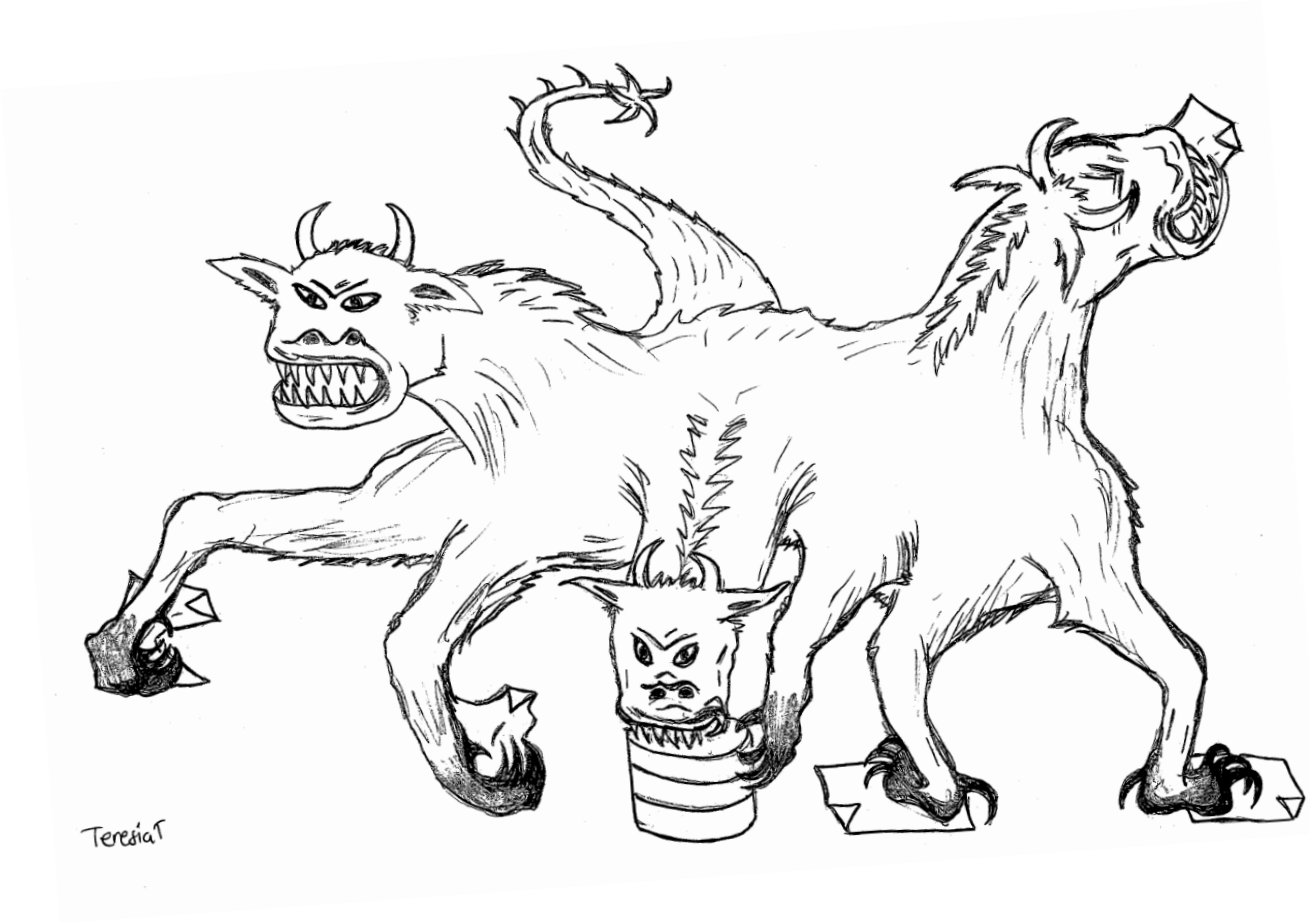
Few features

- Active product ownership
- Pruning

Small codebase

- Singularity
- Level of abstraction
- Efficiency
- Reuse


Controllability
"Can we set the system to a certain state?"
Observability
"Can we see what's going on?"
Smallness
"Let's not have too much of it."
Product ownership
Architecture
Design
Single line of code



Legacy code is the #1 reason to avoid developer testing

(#2 reason being not knowing what to test)

Dependency Breaking

```
public class LegacySystem {
    public LegacySystem() {
        …
        new UntestableStuff().performLongOperation();
        …
    }
}
```

```
public class LegacySystem {
    static UntestableStuff collaborator;
    public LegacySystem() {
        …
        collaborator.performLongOperation();
        …
    }
}
```

```
public class LegacySystem {
    public LegacySystem(UntestableStuff collaborator) {
        …
        collaborator.performLongOperation();
        …
    }
}
```

Dependency Breaking

```
public class LegacySystem {
    private UntestableStuff collaborator;
    public void doIt() {
        …
        collaborator.performLongOperation();
        …
    }
    public void setUntestableStuff(UntestableStuff collaborator) {
        this.collaborator = collaborator;
    }
}
```

```
public class LegacySystem {
    public void doIt() {
        …
        getUntestableStuff().performLongOperation();
        …
    }
    UntestableStuff getUntestableStuff() {
        return new UntestableStuff()
    }
}
```

NOT

EVERYTHING

IS A *#!

MOCK!

Test Doubles

Mocks

- Can make tests fail
- Verify indirect output
- Test interactions

Stubs

- Can't make tests fail
- Provide indirect input

Dummies

- Good names for irrelevant arguments

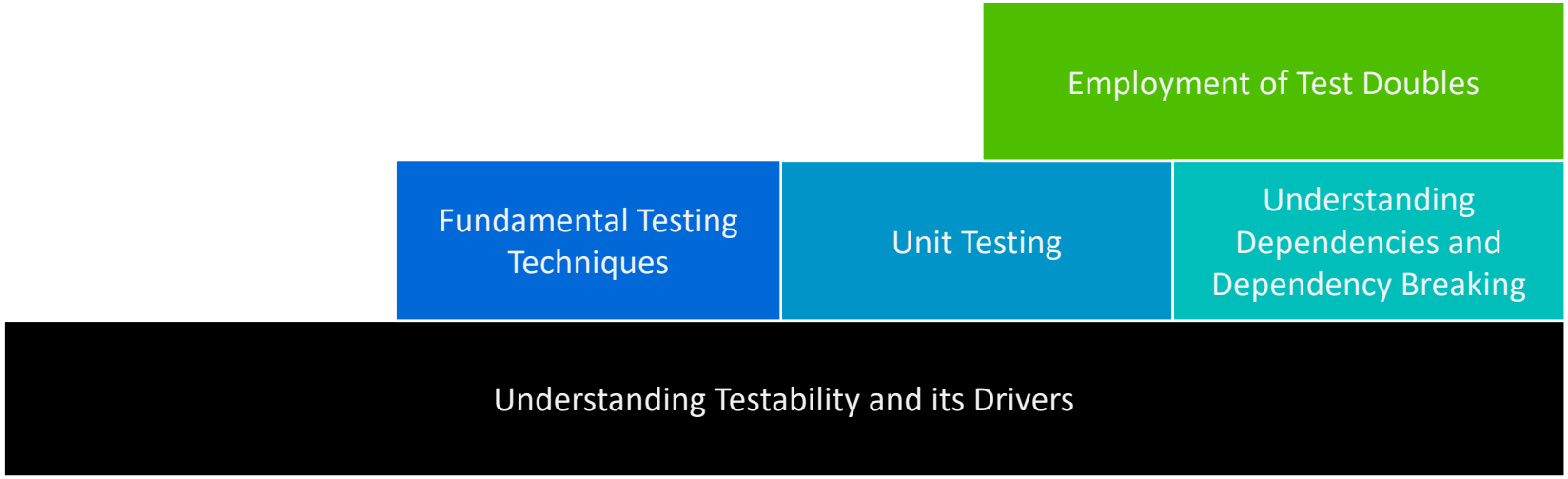
Fakes

- The in-memory database that you never use anyway

Spies

- Created by mocking frameworks and pretend to be mocks

Getting there…



Data-driven Testing

```
childrenAged5NotAllowedToTakeALoan()

childrenAged17NotAllowedToTakeALoan()

adultsAged18RequireAnInvestigation()

adultsAged18AllowedToTakeALoanAfterInvestigation()

adultsAged30AllowedToTakeALoan()

adultsAged64AllowedToTakeALoan()

adultsAged65AllowedToTakeALoanAfterInvestigation()

seniorsAged69AllowedToTakeALoanAfterInvestigation()

seniorsAged70NotAllowedToTakeALoan()
```

Parameterized tests

```
def "Loan policy takes age and an investigation into account"() {

    given: "Our loan policy"
    def loanPolicy = new LoanPolicy()

    when: "It's applied to a customer's age and the presence of an investigation"
    def result = loanPolicy.applyTo(age, investigationDone)

    then: "It tells whether a loan is approved"
    result == approved

    where:
    age | investigationDone || approved
    5   | false              || false
    17  | false              || false
    18  | true               || true
    18  | false              || false
    30  | false              || true
    64  | false              || true
    65  | true               || true
    69  | true               || true
    70  | true               || false
}
```

Theory Tests

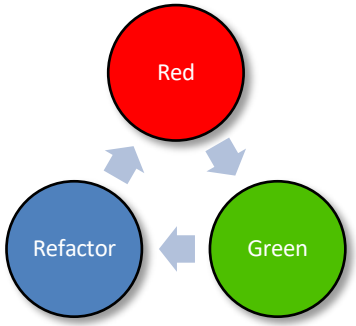
$$\{5, 17, 18, 30, 64, 65, 69, 70\} \times \{\text{true}, \text{false}\} =$$

Data points

Cartesian product

(5, true)
(5, false)
(17, true)
(17, false)
(18, true)
(18, false)
(30, true)
(30, false)
(64, true)
(64, false)
(65, true)
(65, false)
(69, true)
(69, false)
(70, true)
(70, false)

Test-driven Development



- Patterns -

- Fake it
- The obvious
- Triangulate

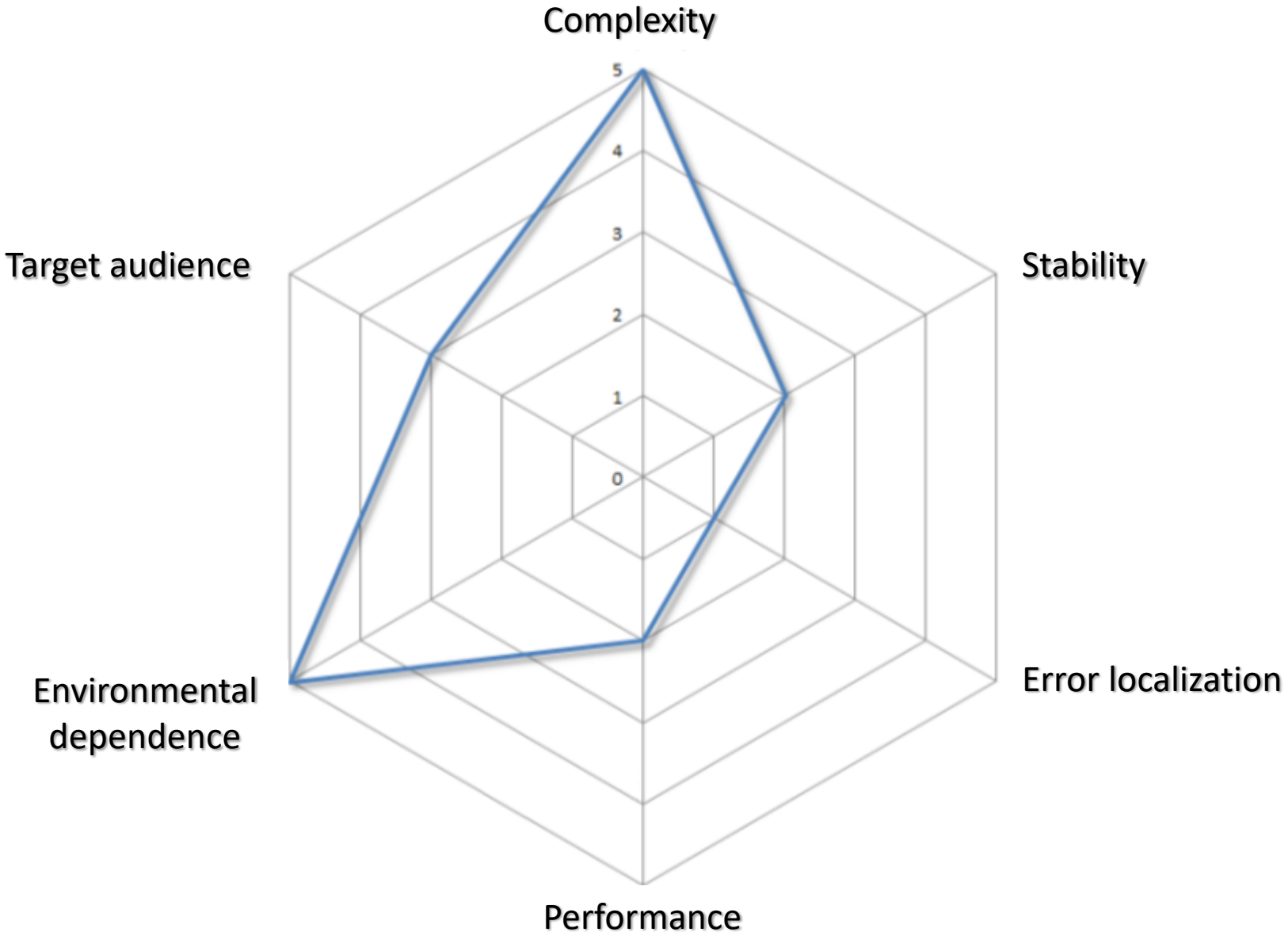
– Test Order –

1. Degenerate case
2. Happy path
3. Explore & learn
4. Error handing

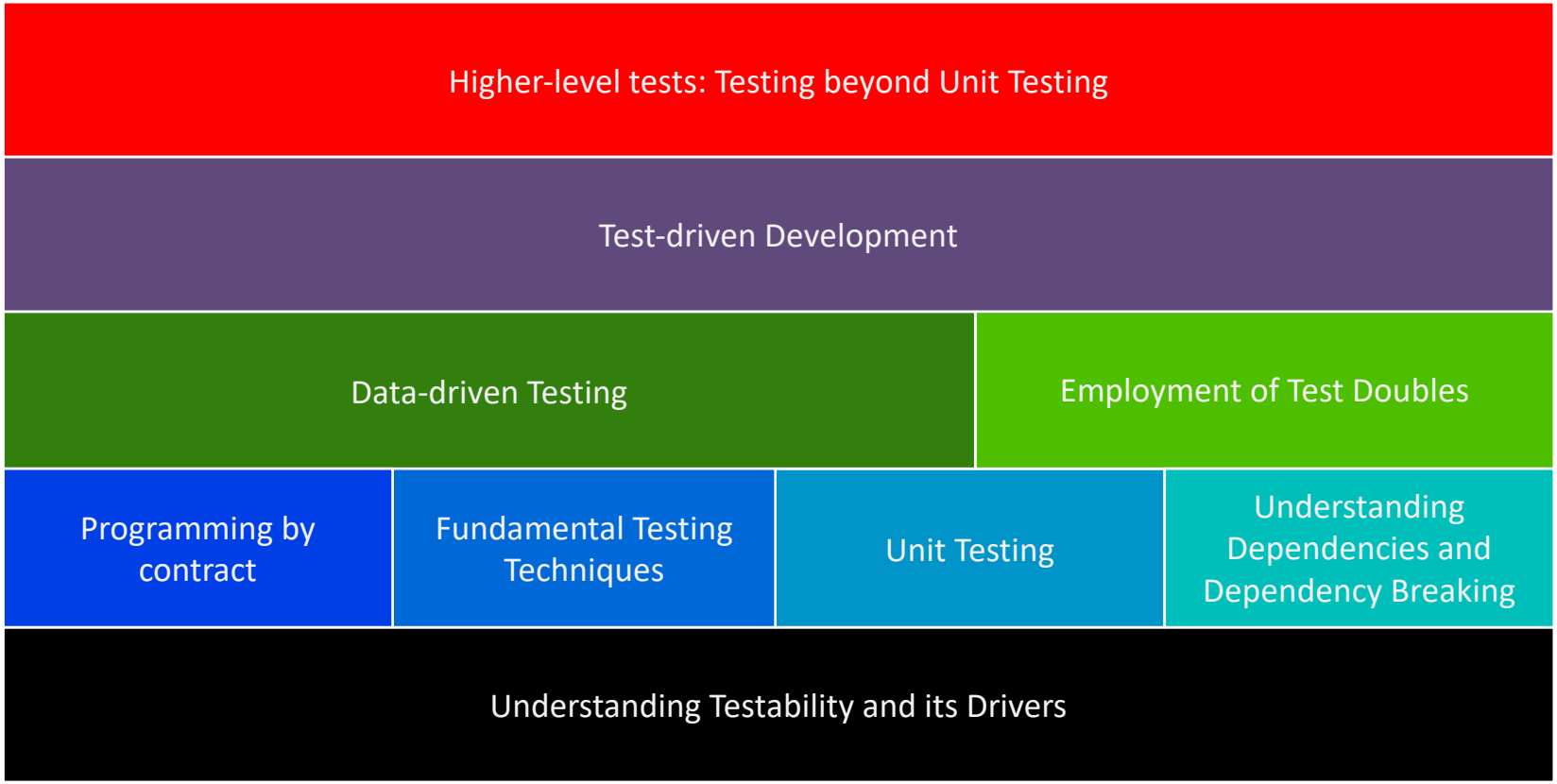
Higher-level tests

- Enclosed in transactions
- Services or components
- Interact with other systems
- Run through the UI
- Invoke another system

Properties of Higher-level Tests



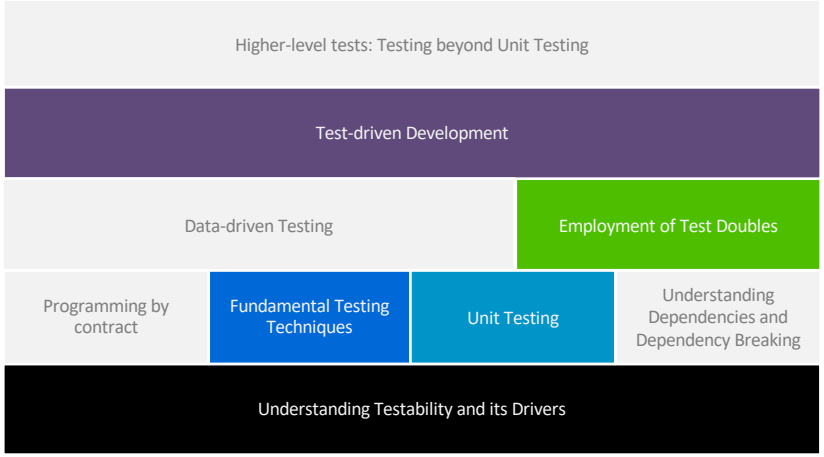
The Core Competencies of Developer Testing



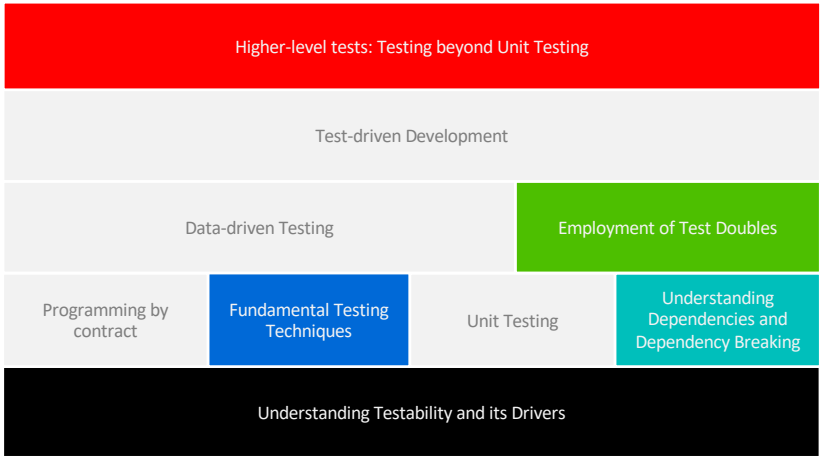
The Developer Testing Formula

- ✓ Understand the Core Competencies
- Determine a profile of your system based on
  - Architecture
  - Mission criticality
  - Life expectancy
- Apply the Core Competencies accordingly

Profile 1: Early stage green field



Profile 2: Classic legacy



Profile 3: Business rule-heavy system

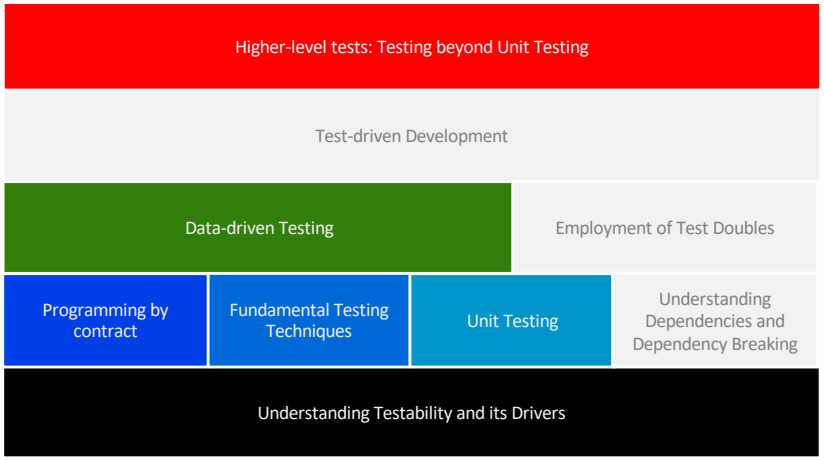
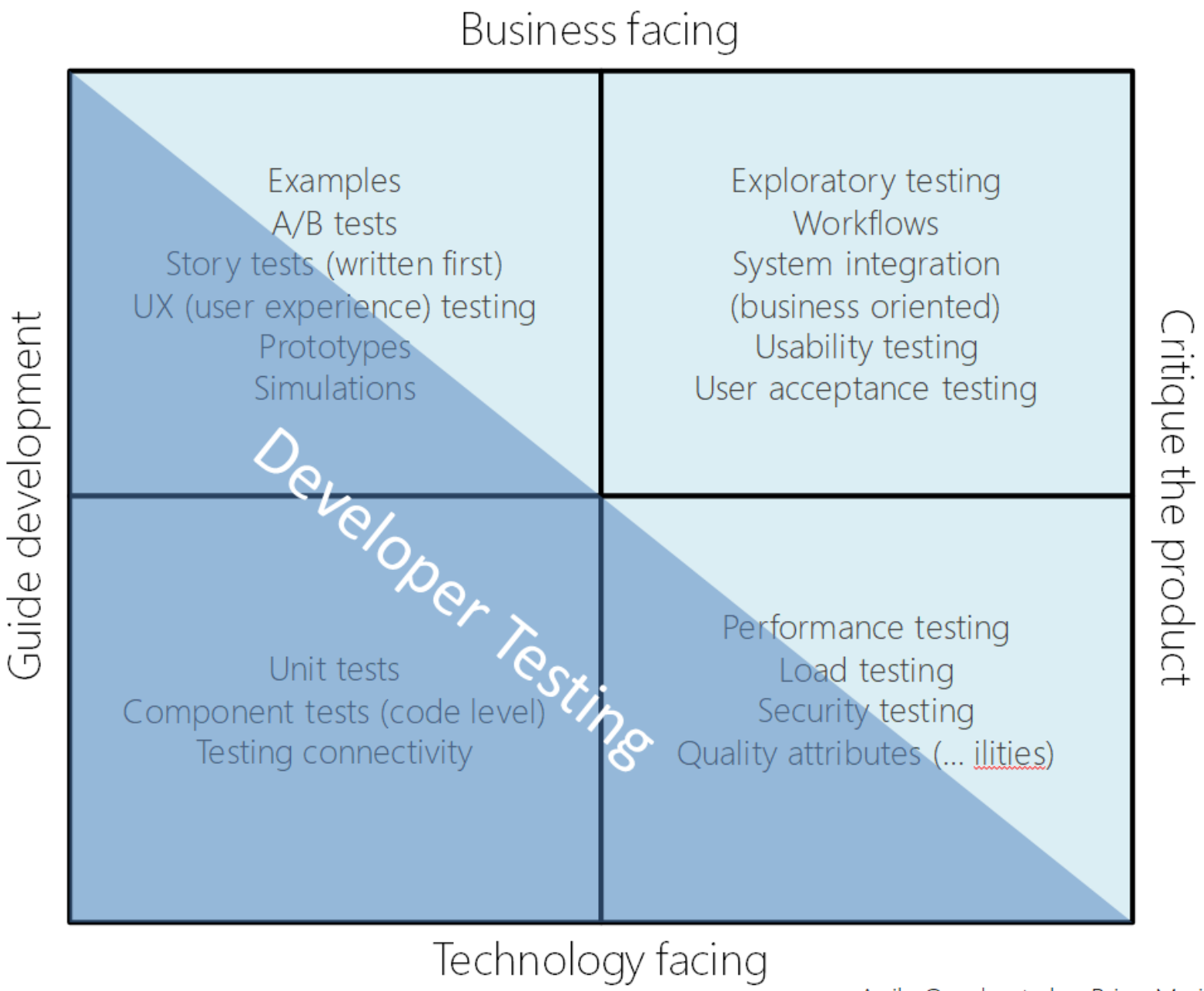


Illustration: Teresia Tarlinder

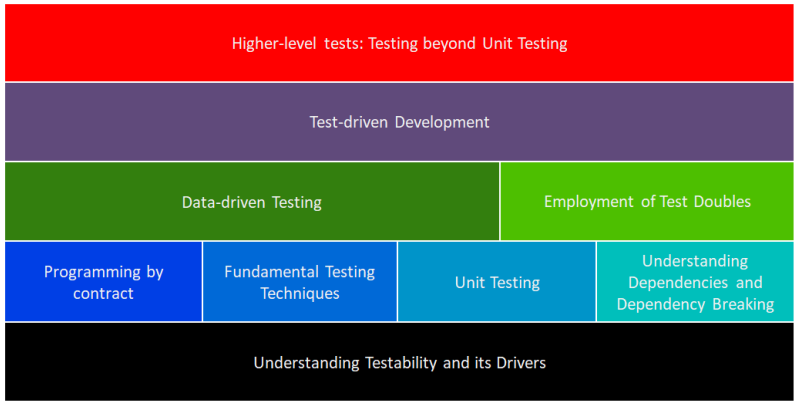


Agile Quadrants by: Brian Marick,
Lisa Crispin & Janet Gregory

Summary

Developer testing is the **systematic** and **intentional** use of testing tools and techniques while coding.

There are 9 Core Competencies



Further details: http://developertesting.rocks