

Understanding the **Successes** and **Pain Points** of Different Testing Strategies

Benjamin Bryant

Benjamin Bryant

Organiser of London Gophers

Go Developer Advocate @JetBrains

Prologue

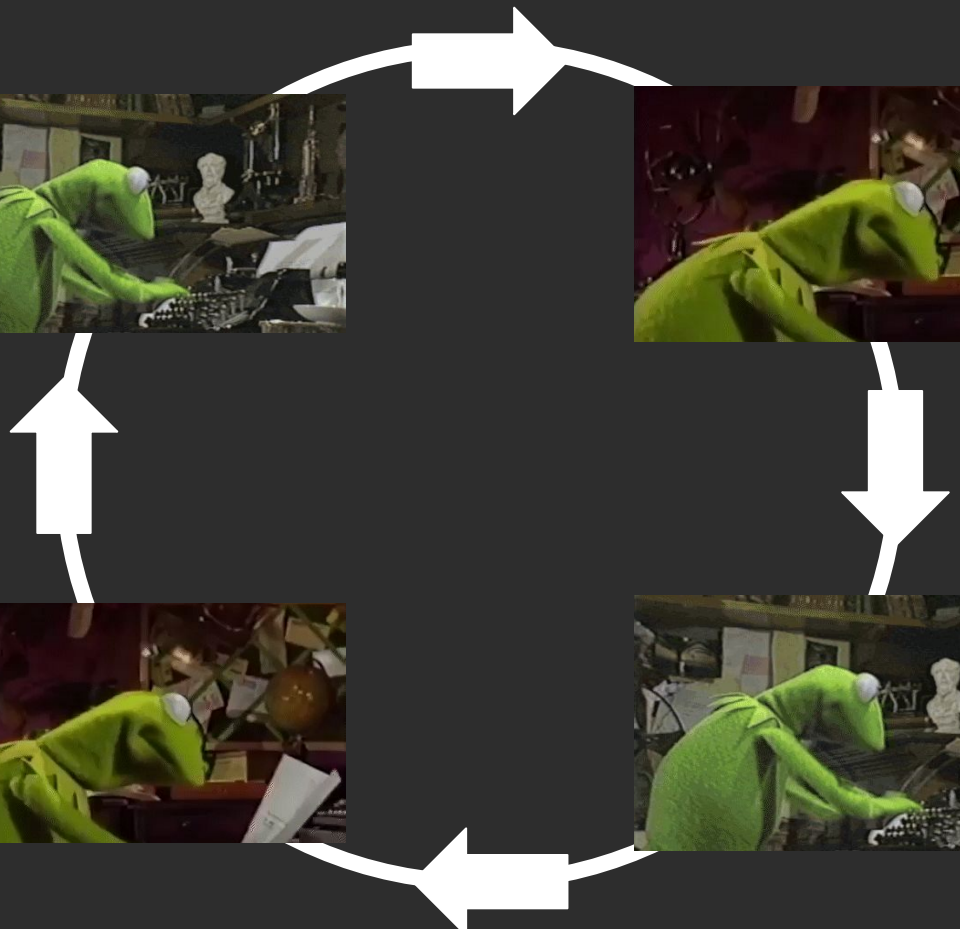
This Talk Was A
Dilly of a Donut To Write

“This is Fire!”



“This isn’t it..”





“Screw It.. The Sky is Blue!”



What Is This Talk About?

Chatted to People of Varying Skill Levels for About
30-60 Minutes.

Asked “Do You Enjoy Testing?”

What To Expect?

Part 1: Testing Overview

Part 2: Thoughts and Observations

Part 1

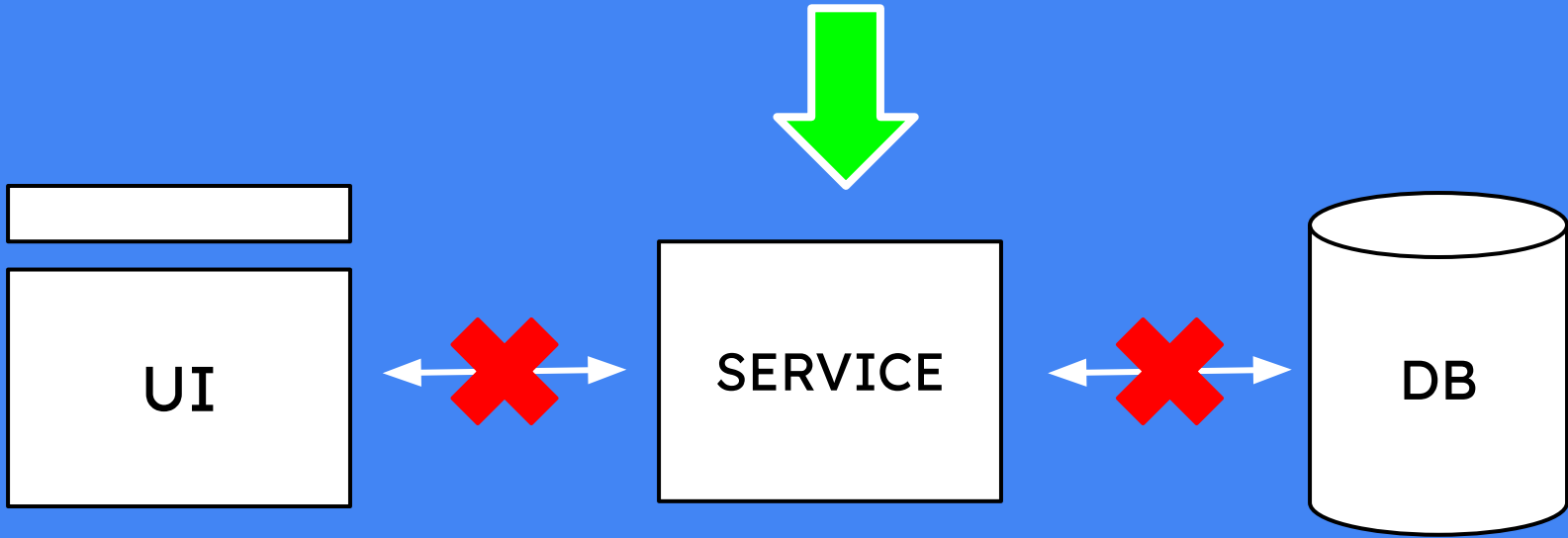
Testing Overview

Successes = What People Enjoy

Pain Points = What People Don't Enjoy

Remedies = Things to Ease The Pain

Unit Testing



```
func Add(a int, b int) int { 1 usage  
    return a + b  
}
```

```
func TestAdd(t *testing.T) {
    type args struct {
        a int
        b int
    }
    tests := []struct {
        name string
        args args
        want int
    }{
        {
            name: "3 add 5",
            args: args{
                a: 3,
                b: 5,
            },
            want: 8,
        },
    }
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            if got := Add(tt.args.a, tt.args.b); got != tt.want {
                t.Errorf("format: \"Add() = %v, want %v\", got, tt.want)
            }
        })
    }
}
```


Unit Testing

Successes

Successes - Unit Testing

Simple and Fast, In
Theory

Successes - Unit Testing

Local, Low Setup Costs

Unit Testing

Pain Points

Pain Points - Unit Testing

Can Easily Become Too
Numerous and Too Brittle

Pain Points - Unit Testing

It Can Be Difficult To Write
Them Well


Unit Testing
Remedies

Reducing Unnecessary Tests and Putting Time Into Choosing What To Test


Remedies


Try Not To Test
Everything In One,
Keep Tests Focussed

Remedies

 simple.go

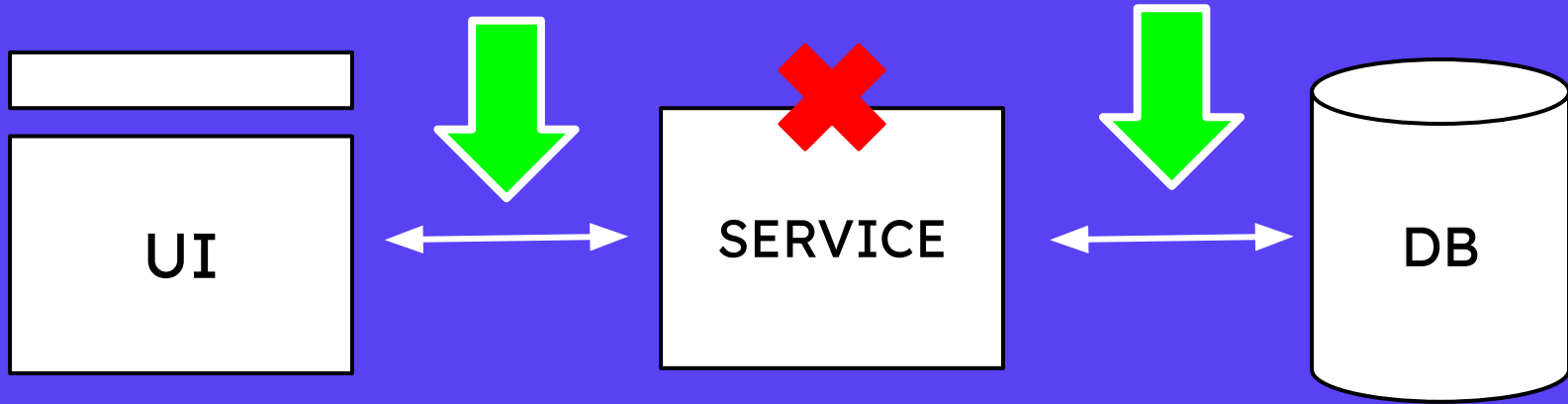
 simple_test.go

 verbose.go

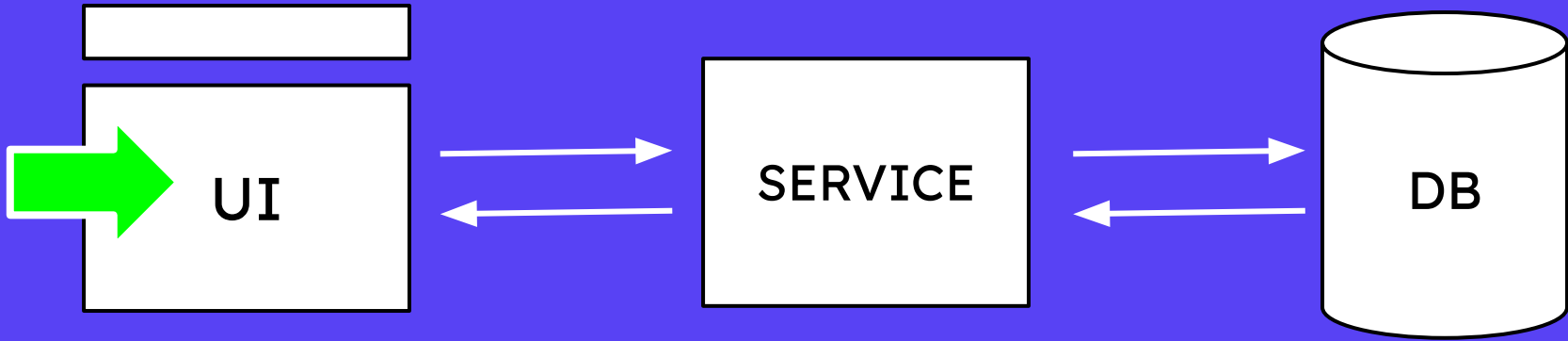
 verbose_test.go

Integration Testing/ End-To-End Testing

Integration Test



End-to-End Test



Integration/E2E Testing

Successes

Successes - Integration/E2E Testing

Highest Amounts of
Assurance.

Integration/E2E Testing

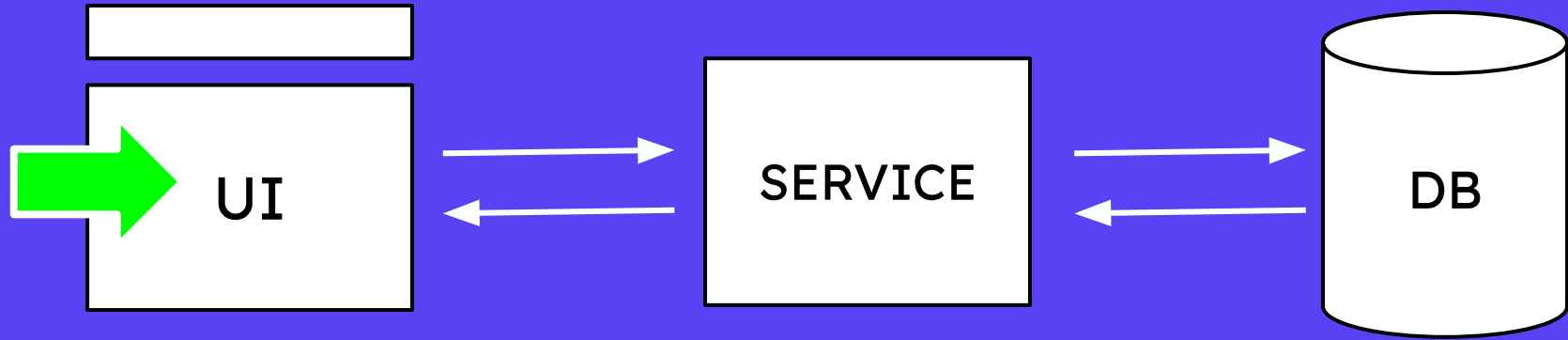
Pain Points

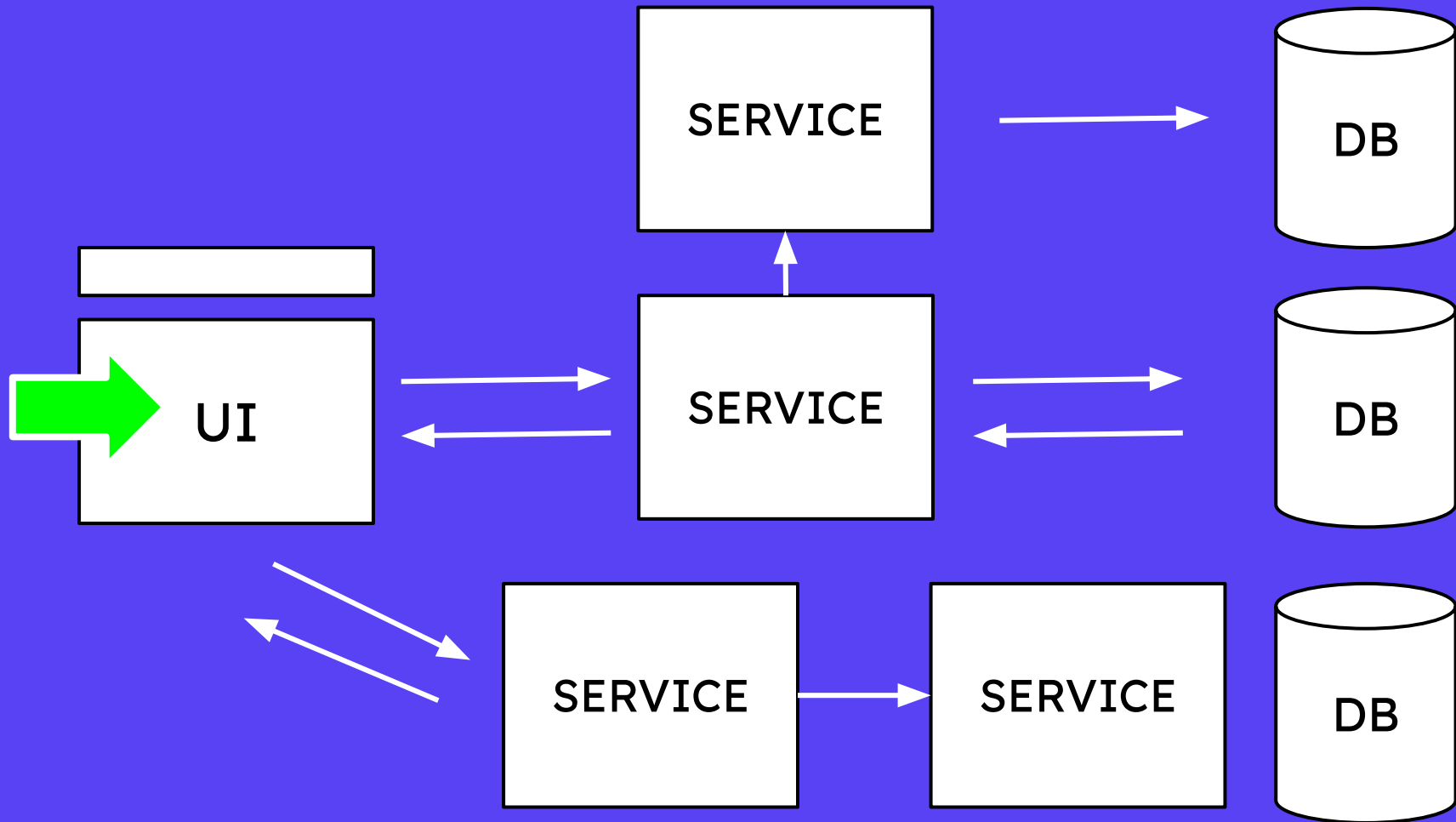
Pain Points - Integration/E2E Testing

Depending on the System,
Incredibly Difficult To Write

Pain Points - Integration/E2E Testing

Complicated To Set Up and
Maintain The Testing
Environment





Pain Points - Integration/E2E Testing

Third-Party Dependencies

Pain Points - Integration/E2E Testing

The More Complicated The
System The More Brittle
The Testing Can Get

Integration/E2E Testing

Remedies

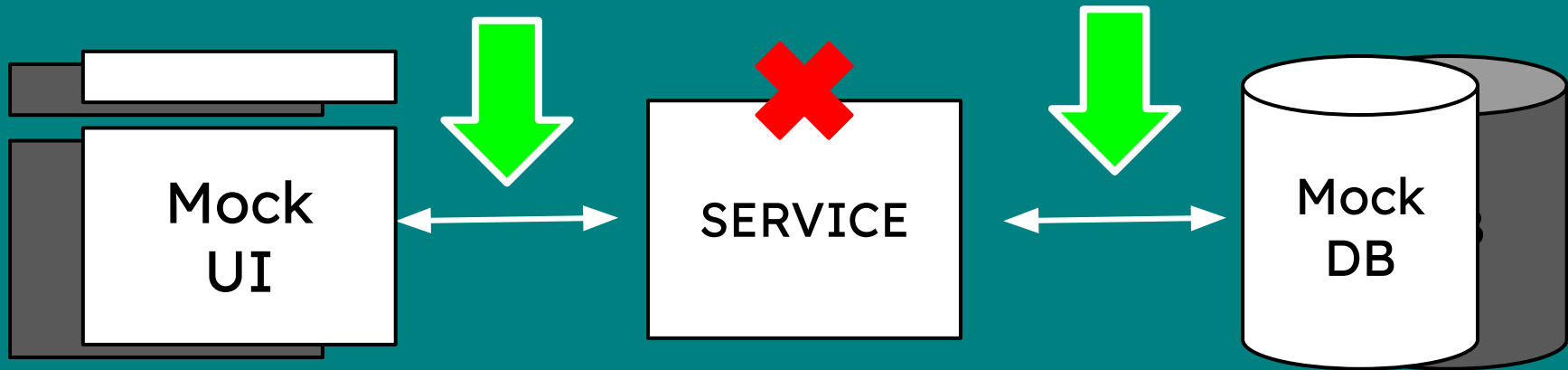
Remedies - Integration/E2E Testing

Allocate Resources To
Asses and Figure out How
To Reduce The Pain

Remedies - Integration/E2E Testing

Reducing Quantity,
Increasing Impact

Mock Testing/ Contract Testing



Unit Testing

Mock Testing
/Contract
Testing



```
func TestFoo(t *testing.T) {
    ctrl := gomock.NewController(t)
    defer ctrl.Finish()

    m := NewMockFoo(ctrl)

    // Does not make any assertions. Executes the anonymous functions and returns
    // its result when Bar is invoked with 99.
    m.
        EXPECT().
        Bar(gomock.Eq(99)).
        DoAndReturn(func(_ int) int {
            time.Sleep(1*time.Second)
            return 101
        }).
        AnyTimes()

    // Does not make any assertions. Returns 103 when Bar is invoked with 101.
    m.
        EXPECT().
        Bar(gomock.Eq(101)).
        Return(103).
        AnyTimes()

    SUT(m)
}
```

gomock [🔗](#)

Update, June 2023: *This repo and tool are no longer maintained. Please see go.uber.org/mock for a maintained fork instead.*

🔄 Run tests passing  reference

gomock is a mocking framework for the [Go programming language](#). It integrates well with Go's built-in `testing` package, but can be used in other contexts too.



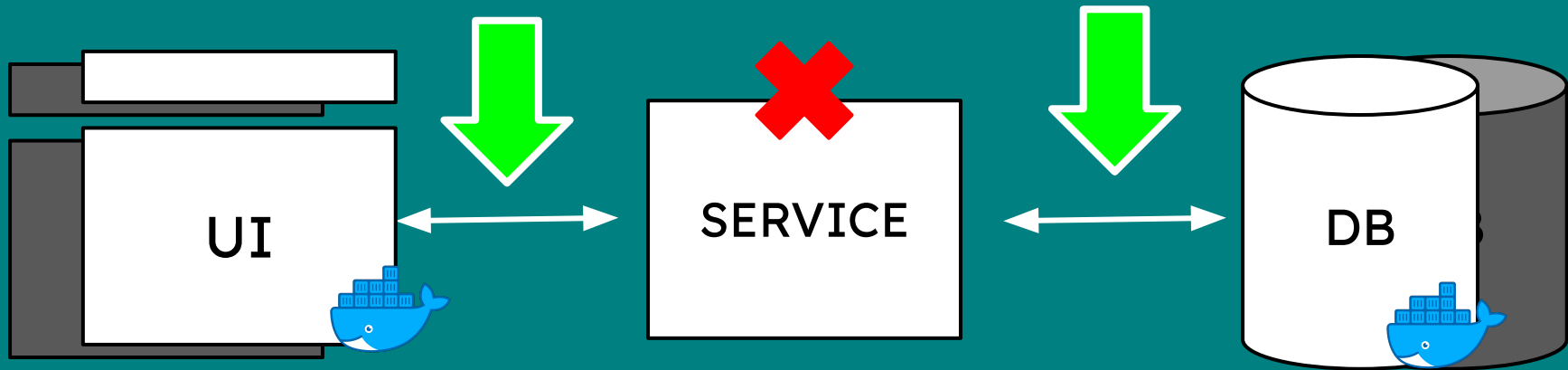
Testcontainers

HOW IT WORKS

Test dependencies as code

No more need for mocks or complicated environment configurations. Define your test dependencies as code, then simply run your tests and containers will be created and then deleted.

With support for many languages and testing frameworks, all you need is Docker.



USE CASES

How Testcontainers can help you



Data access layer integration tests

Use a containerized instance of your database to test your data access layer code for complete compatibility, without requiring a complex setup on developer machines. Trust that your tests will always start with a known state.



UI/Acceptance tests

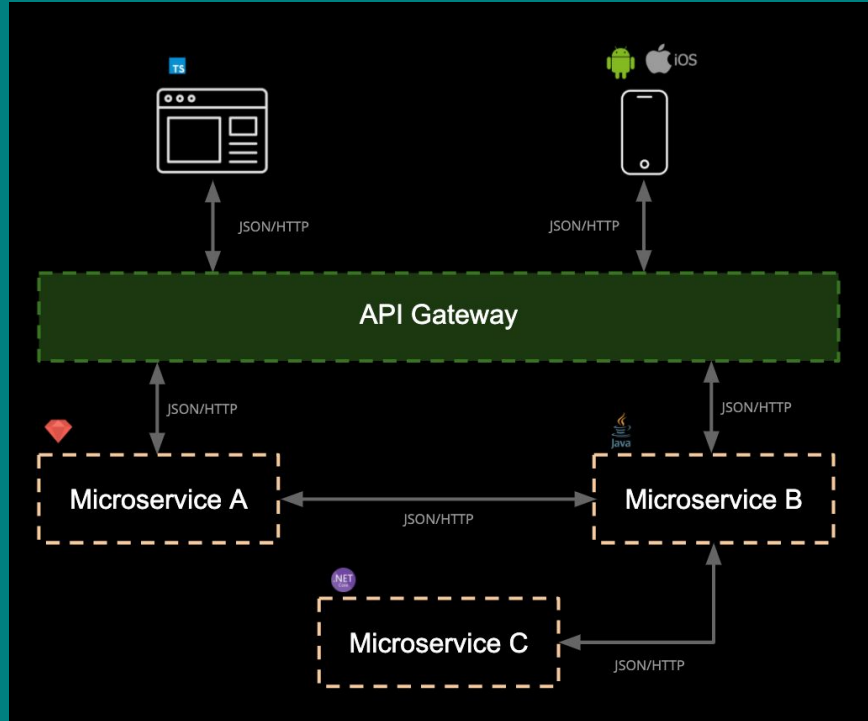
Use containerized web browsers, compatible with Selenium, to run automated UI tests. Each test gets a fresh, clean instance of the browser, without having to worry about variations in plugins or required updates.



Application integration tests

Run your application in a short-lived test mode with dependencies, such as databases, message queues or web servers, to give you a rich interactive and explorative testing environment.

Contract Testing



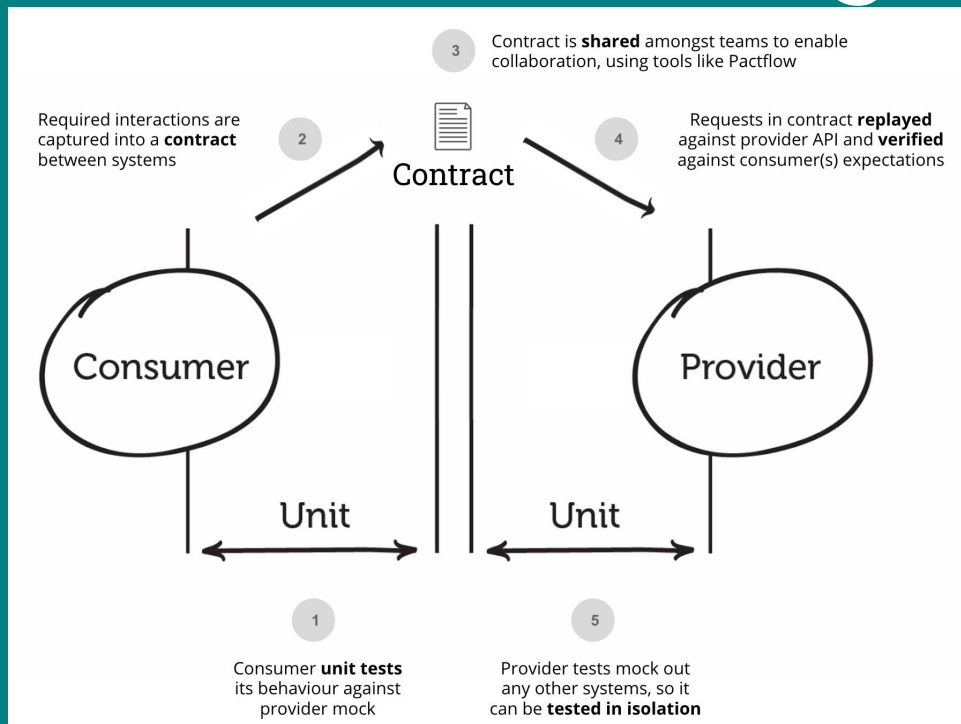
Source: <https://pact.io/>

Contract Testing

PACT 

Source: <https://docs.pact.io/>

Contract Testing



Source: <https://docs.pact.io/>

Mock/Contract Testing

Successes

Successes - Mock/Contract Testing

Provides a Way to Run
'Integration Tests' With The
Same Ease As Unit Testing

Successes - Mock/Contract Testing

Reduces Reliance on Other
Services In Order To Test
Your Service

Mock/Contract Testing
Pain Points

Pain Points - Mock/Contract Testing

Over Reliance on Mocks
Could Lead To False
Confidence

Pain Points - Mock/Contract Testing

Ultimately You Still Need A
Way To Check the Plumbing
Between System

Pain Points - Mock/Contract Testing

Setup Could Be Just As
Long As Integration
Testing


Mock/Contract Testing



Remedies

Remedies - Mock/Contract Testing

View These Tools As Ways
To *Reduce*, But Not
Completely *Replace*
Integration Testing

Code Coverage

▼  codecoverage 100% files, 50% statements

-  example.go 50% statements
-  example_test.go

30

31

32

33

34

35

36

37

38

39

40

41

```
func MultipleErrorPaths(i int) (*Thing, error) { 1 usage
    switch {
    case i <= 5:
        return NewThing(i), nil
    case i <= 10:
        return nil, NewMyCustomError(i)
    default:
        return nil, NotFoundError{}
    }
}
```

}💡

Code Coverage

Successes

Successes - Code Coverage

**It Is A Good Confidence
Builder**

Successes - Code Coverage

It Can Help Highlight Areas
That Are Missable Or
Difficult To Test

Successes - Code Coverage

On A Personal Level, It Can
Be A Feel-Good Statistic

Code Coverage

Pain Points

Pain Points - Code Coverage

**It Can Easily Be A False
Confidence**

Pain Points - Code Coverage

It Is Easy For It To Become
An Annoying Blocker

Pain Points - Code Coverage

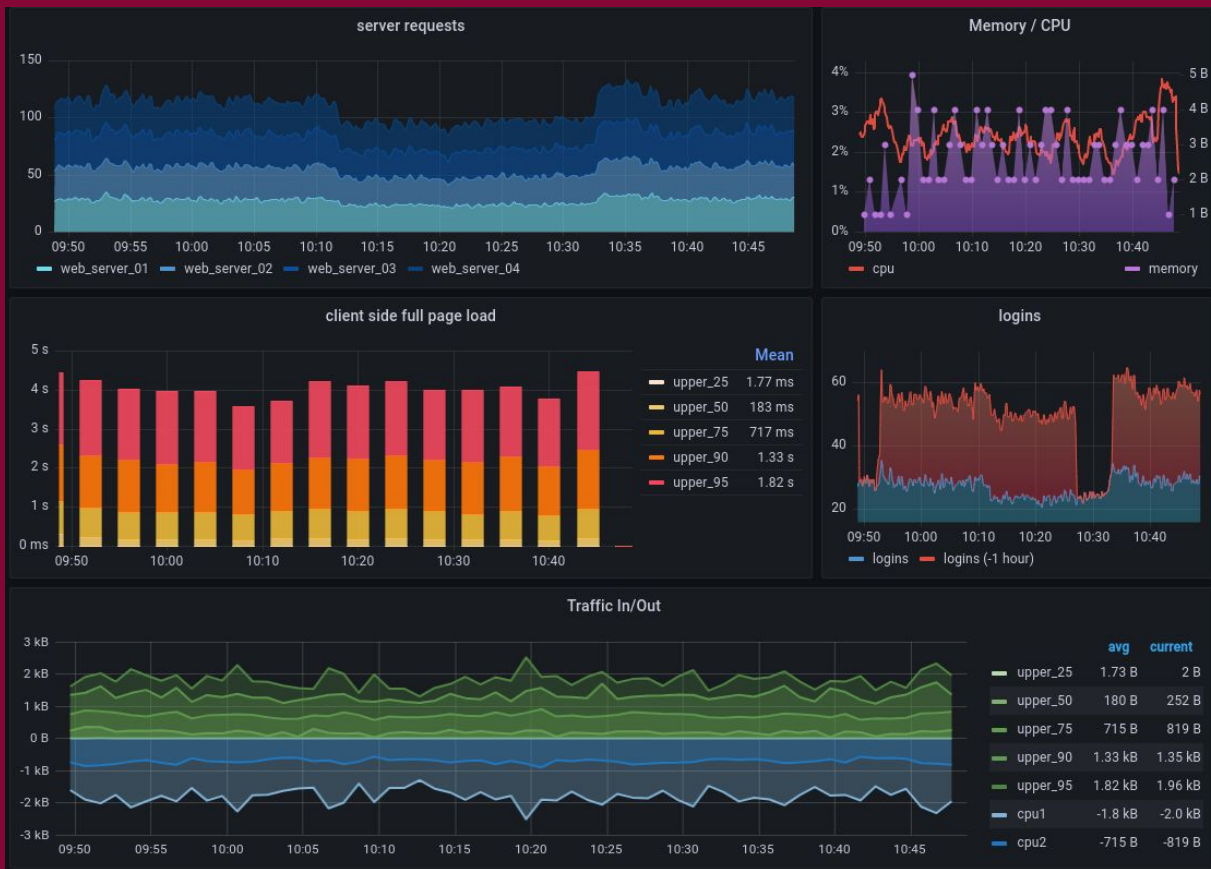
Can Lead To Bad Testing Practices If It Needs To Be Circumvented

Goodhart's Law -

***“When a measure becomes a target,
it ceases to be a good measure”***

Code Coverage
Remedies

Make It Less of a 'Hard'
Requirement



Remedies - Code Coverage

Try To Take It Back To
Being a 'Personal Win'

CI/CD
**(Continuous Integration/
Delivery)**

🏠 Summary

Jobs

✔️ build

✔️ deploy

Run details

🕒 Usage

📄 Workflow file

build

succeeded 3 days ago in 36s

🔍 Search logs



- > ✔️ Set up job 1s
- > ✔️ Install Hugo CLI 1s
- > ✔️ Checkout 6s
- > ✔️ Setup Pages 0s
- > ✔️ Build with Hugo 9s
- > ✔️ Upload artifact 16s
- > ✔️ Post Checkout 0s
- > ✔️ Complete job 0s

Continuous Integration/Delivery
Successes

Successes - Continuous Integration/Delivery

It Is Magic 🪄

Successes - Continuous Integration/Delivery

It Can Help To
Standardise A Workflow

Continuous Integration/Delivery

Pain Points

Pain Points - Continuous Integration/Delivery

If It Breaks, It Is Magic

Continuous Integration/Delivery
Remedies

Remedies - Continuous Integration/Delivery

Demystification

Part 2

Observations

Observation #1
It Takes A Village To
Test An Add Function

```
func Add(a int, b int) int { 1 usage  
    return a + b  
}
```

Unit Testing

Integration/E2E Testing

Mock/Contract Testing

Code Coverage

CI/CD

Etcetera

Observation #2

Your Journey Through Tech, Has a Big Influence When It Comes to Developing a 'Testing Is Good' Mentality



Chris James
quii



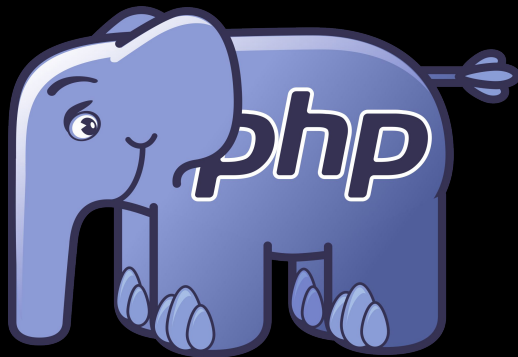
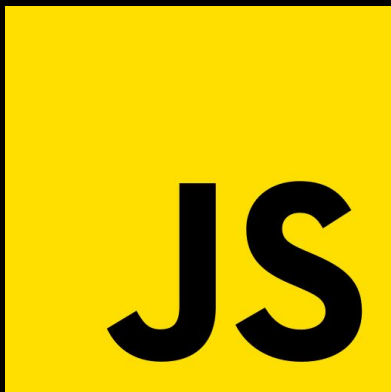
Learn Go with tests

Observation #3


**There Is Correlation Between a
Good Testing Experience and a
Good Developer Experience**


Observation #4


**In Software Engineering Testing is a
Second Class Citizen, but I Think That's
Changing**






 simple.go

 simple_test.go

 verbose.go

 verbose_test.go

Conclusions

**There Is A Lot To Talk About
On This Subject**

**Ultimately Testing
Is A Part Of The Framework
That Is Software Development**

**So If You Can,
Try Not To Forget About It
And Try To Make It Fun!**

Anyway

I've Been Me

Thank You For Your Time



Socials



<https://linktr.ee/jaminologist>