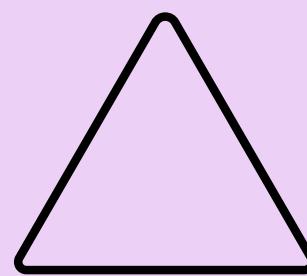




Comprehensive testing strategies for modern microservice architectures





About me



- DevRel since 2021
- Go Engineer since 2018
- LinkedIn Learning instructor
- Packt author
- Technical University of Denmark (DTU)
- Organiser of Women Who Go ❤️

 classic_addetz

 adelina-simion

 addetz

Women Who Go!



What percentage of time do you think developers spend fixing bugs?



A) <25%

Less than a quarter of our time



B) 25-50%

Between a quarter and half our time



C) >50%

The majority of our time



Developers spend 25-50% of their time fixing bugs.

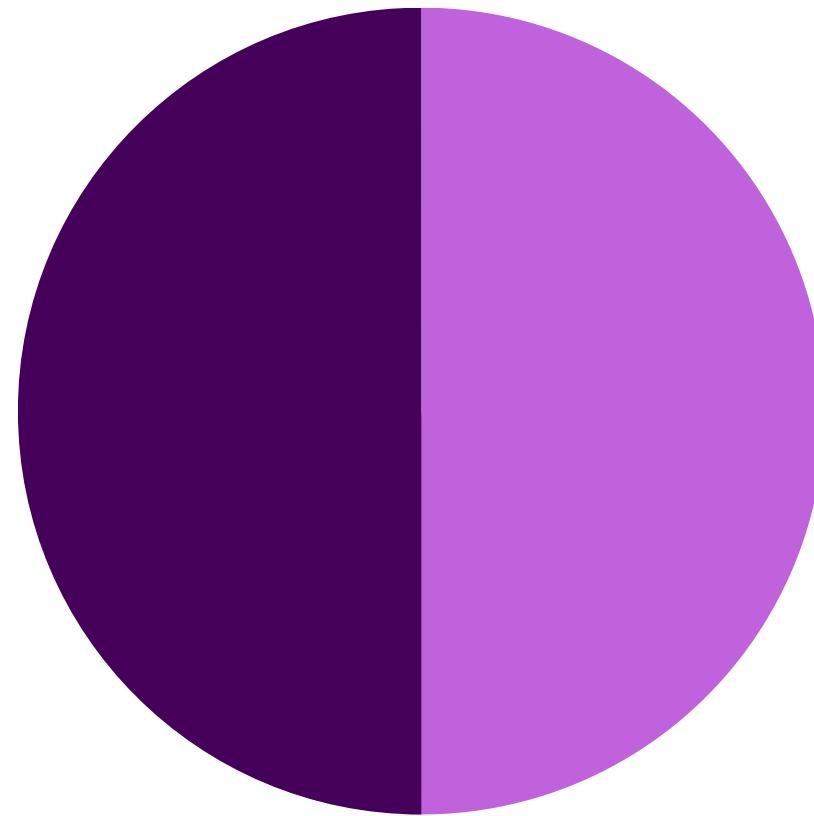


B) 25-50%

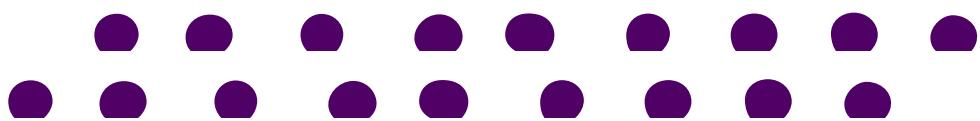
Between a quarter
and half our time

(Source: Rollbar survey)

Fixing bugs
50%



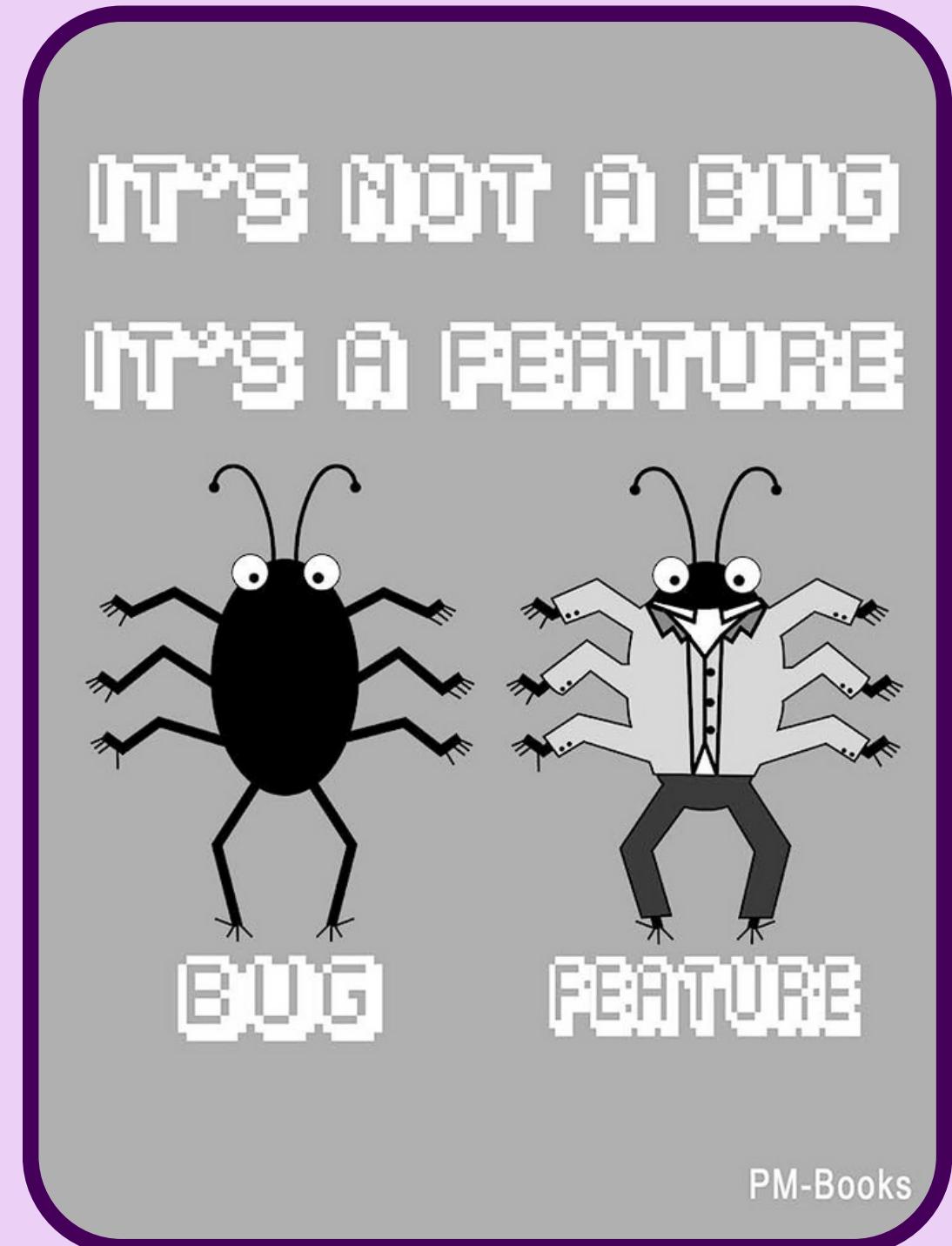
Fun stuff
50%



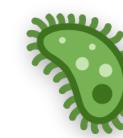


Common bug causes

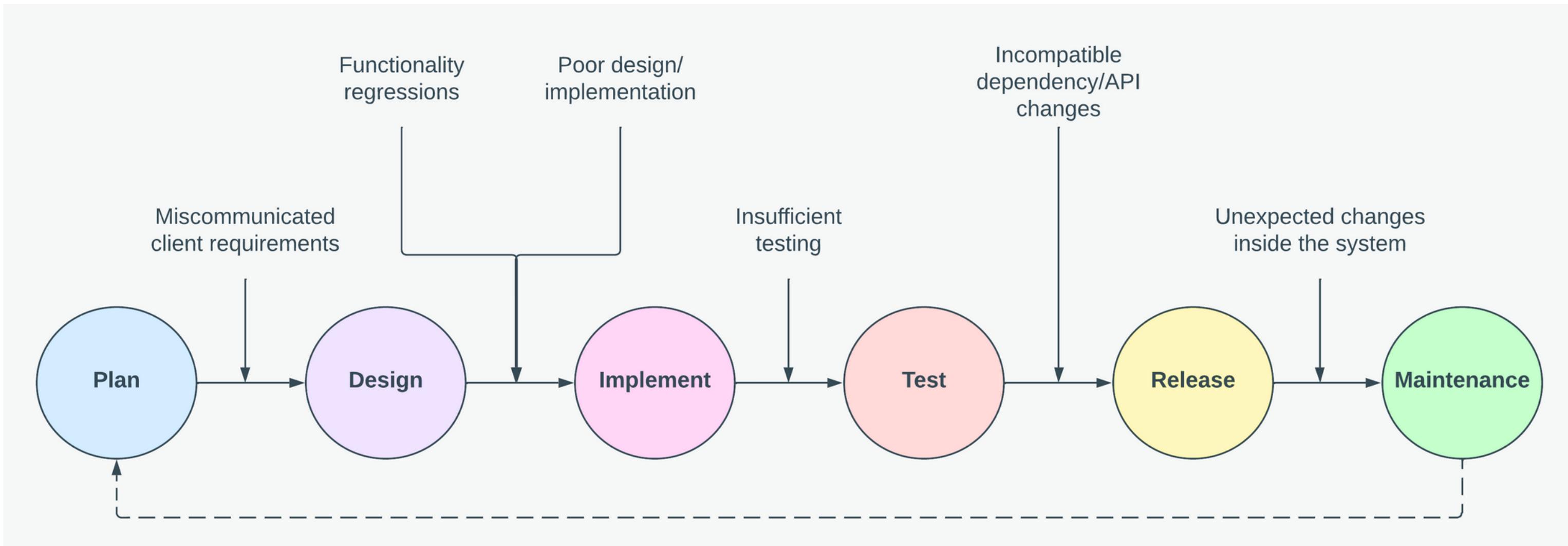
- Miscommunicated client requirements
- Poor design/implementation
- Insufficient testing
- Functionality regressions
- Incompatible dependency/API changes
- Unexpected changes inside the system

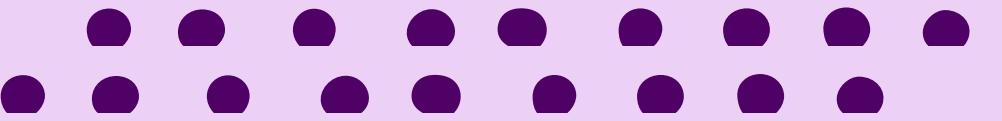


Creator: PM Books



Bugs in the SDLC

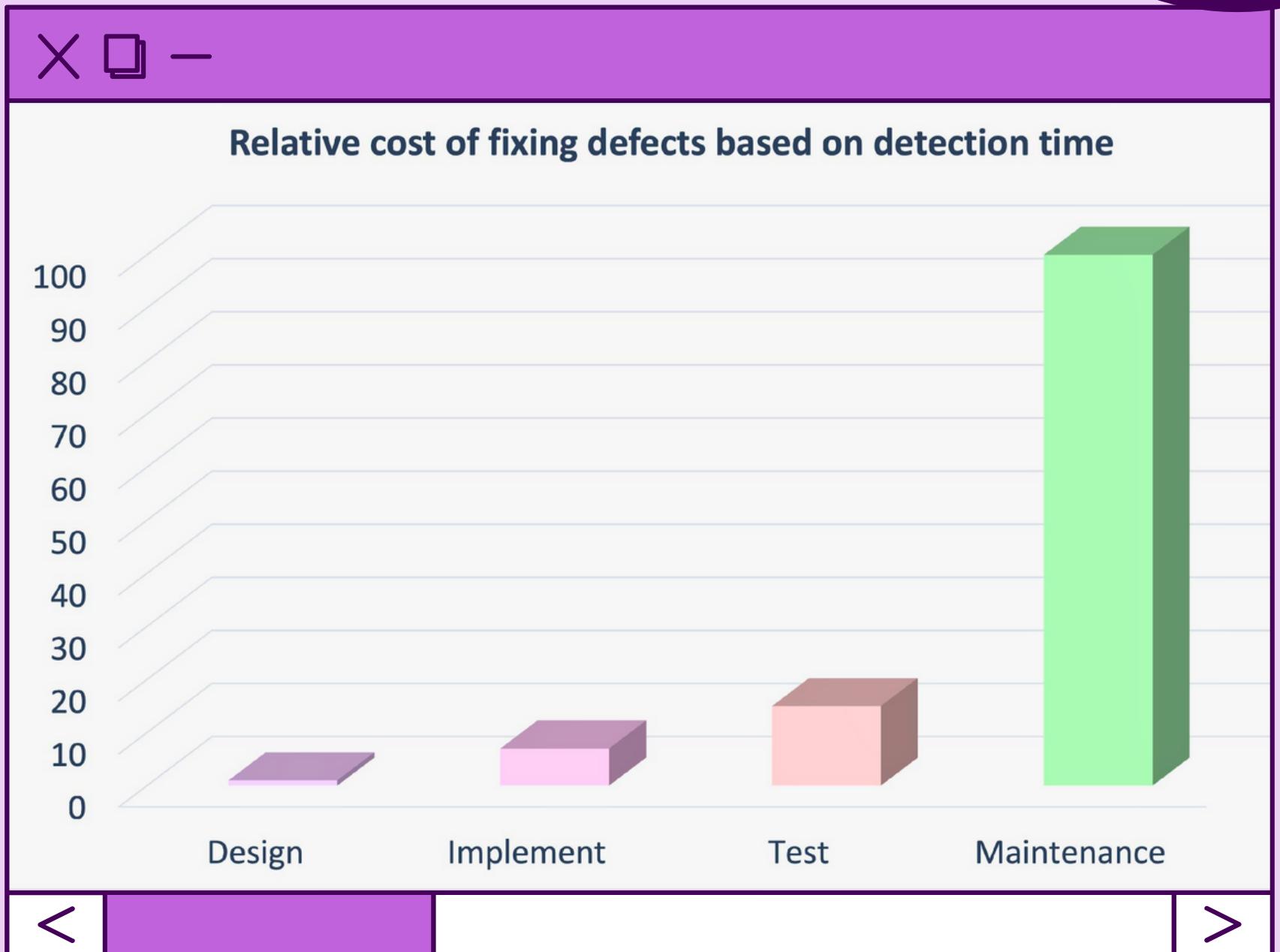




\$ Exponential costs

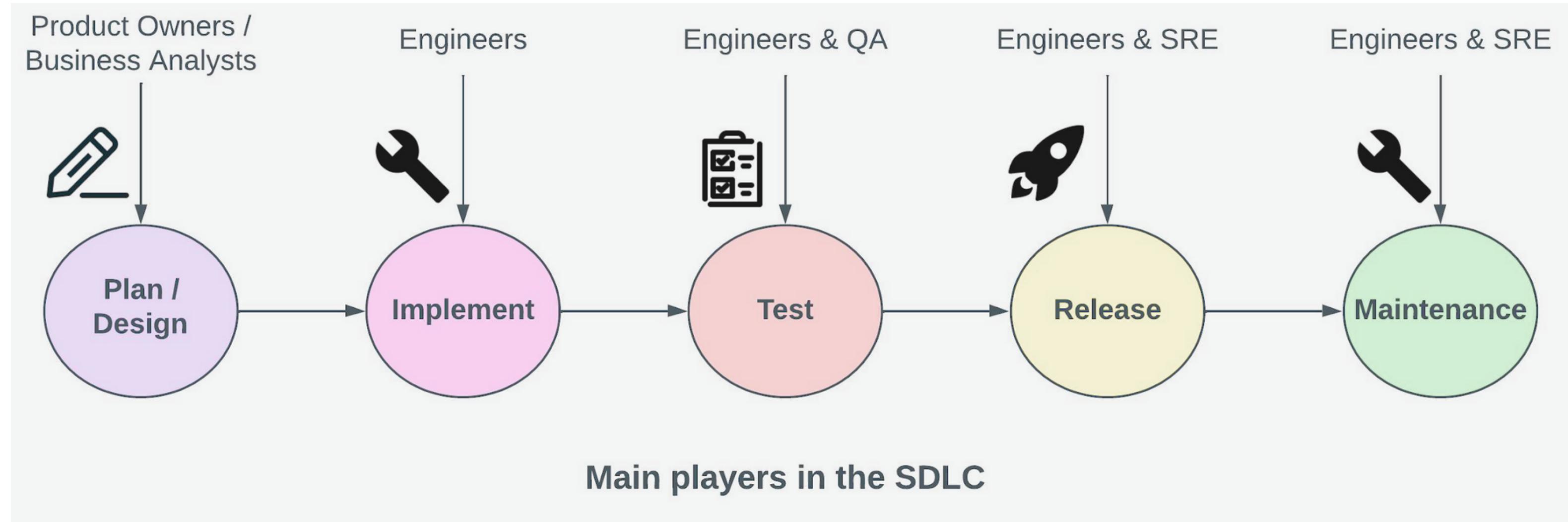
- Detect early, detect often to avoid delays and increased costs (the **“shift left”** motto)
- The cost of fixing bugs increases up to 100 times when identified after release

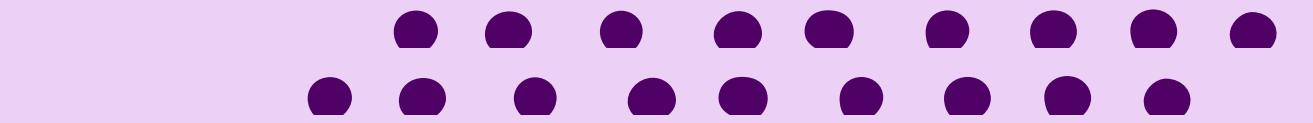
(Source: IBM study)





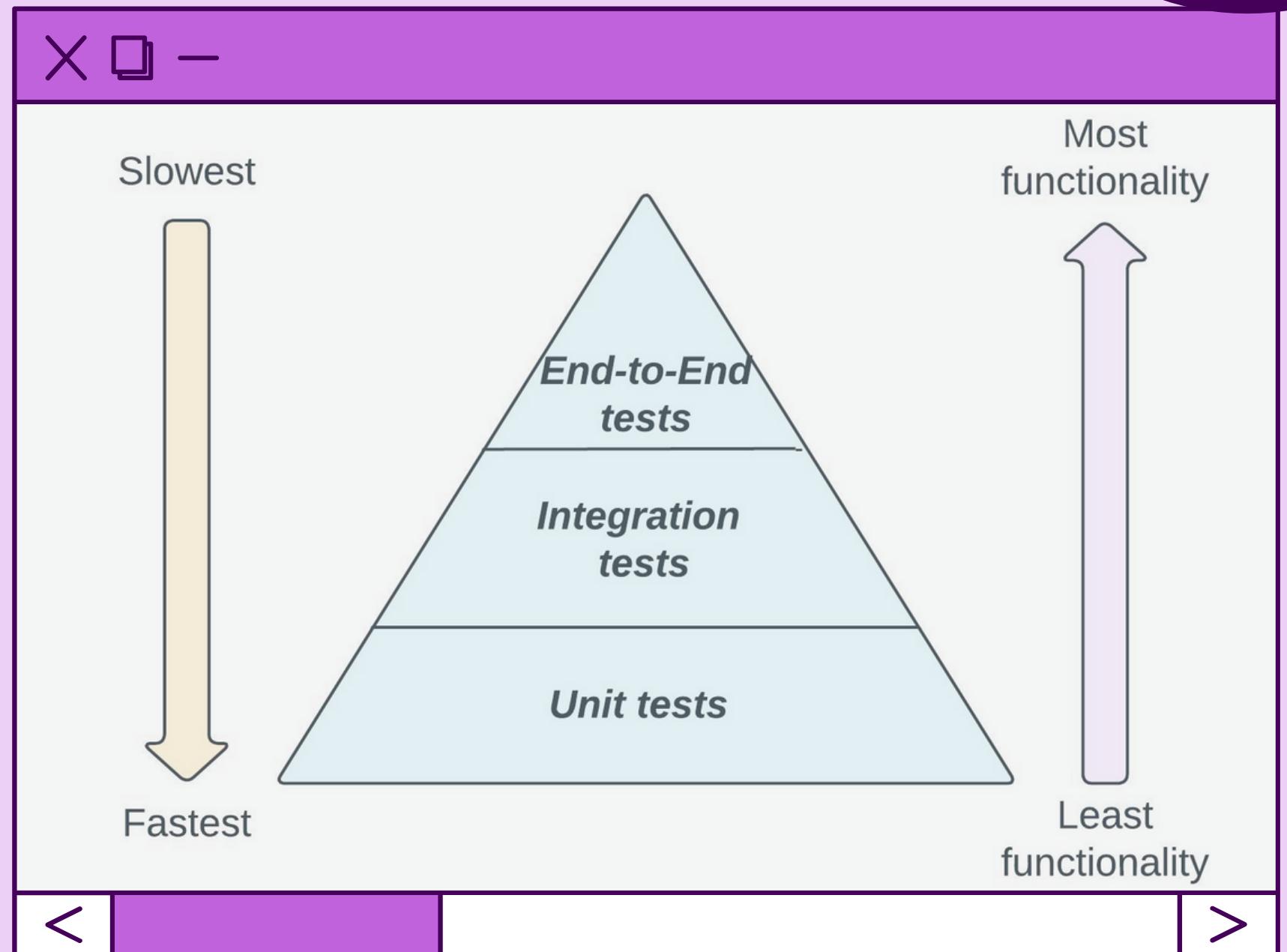
An engineer's life...





▲ The testing pyramid

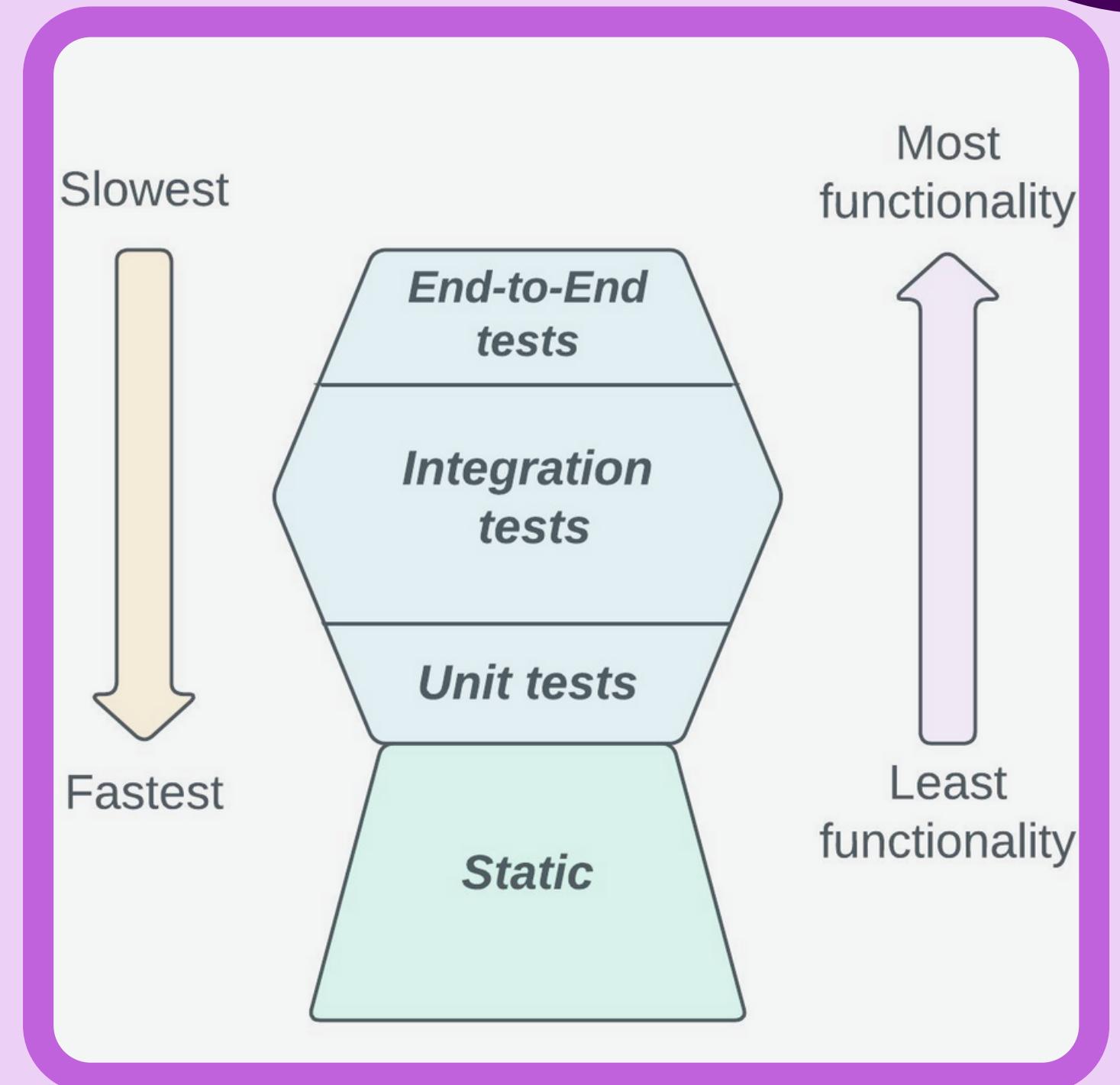
- **Unit tests** verify internal code mechanisms
- **Integration tests** verify multiple units work well together
- **End-to-end tests** verify user interactions

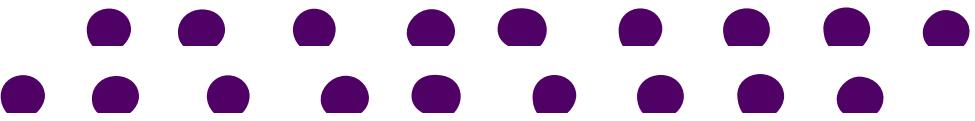




🏆 The testing trophy

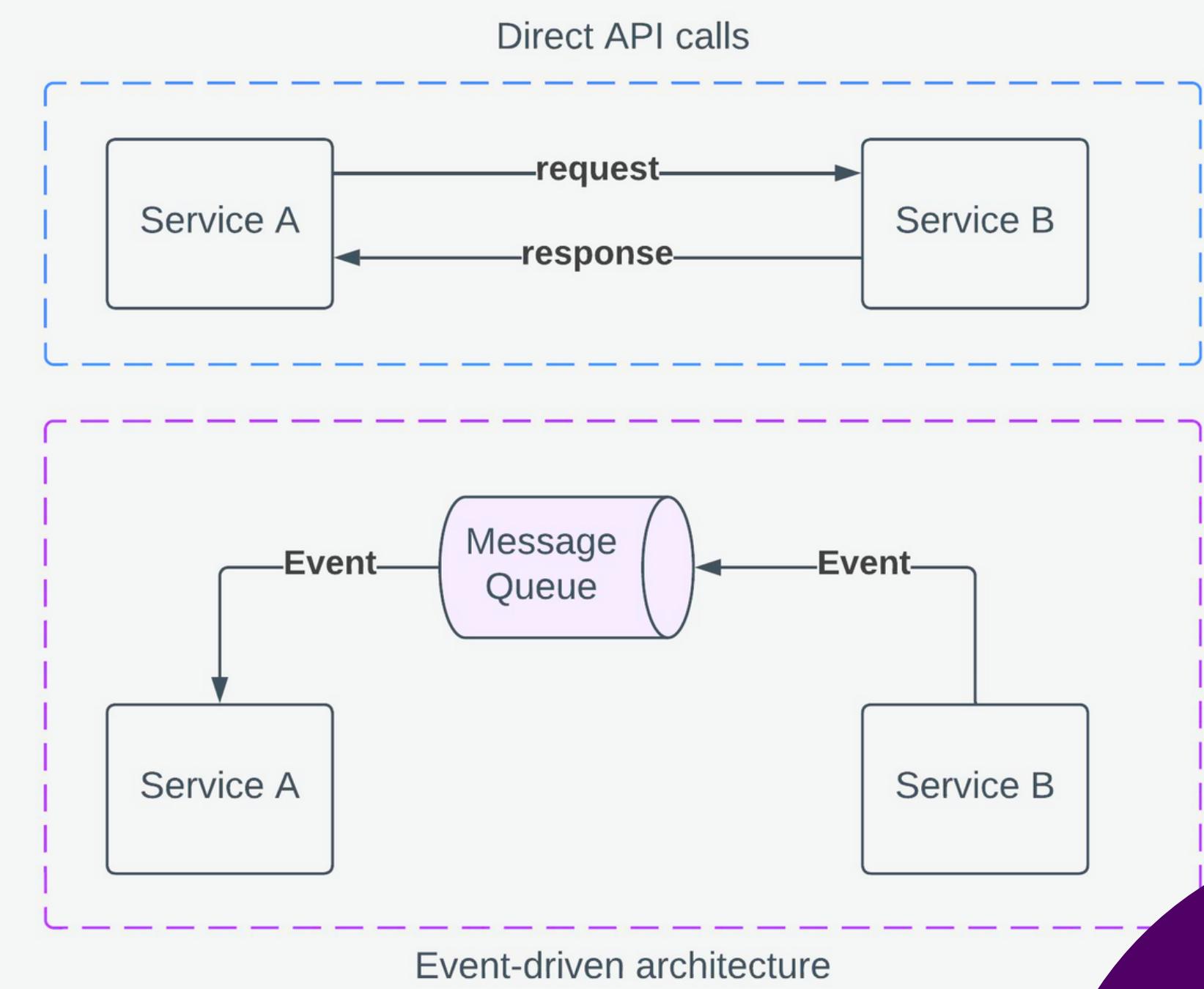
- Tests have diminishing returns and can also add a maintenance burden.
- Common static checks are:
 - Reviews
 - Static analysis
 - Checkstyle
 - Linters
- Code coverage should be in the **80-90% range**

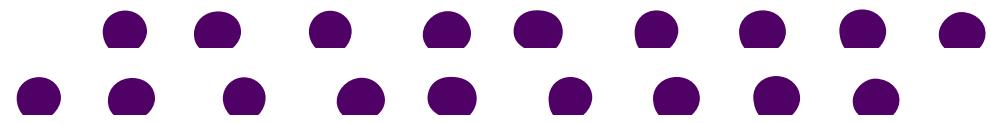




A closer look...

- Microservices change without any central oversight and are released at different times
- They might directly integrate with each other or use event driven architectures such as event buses/queues





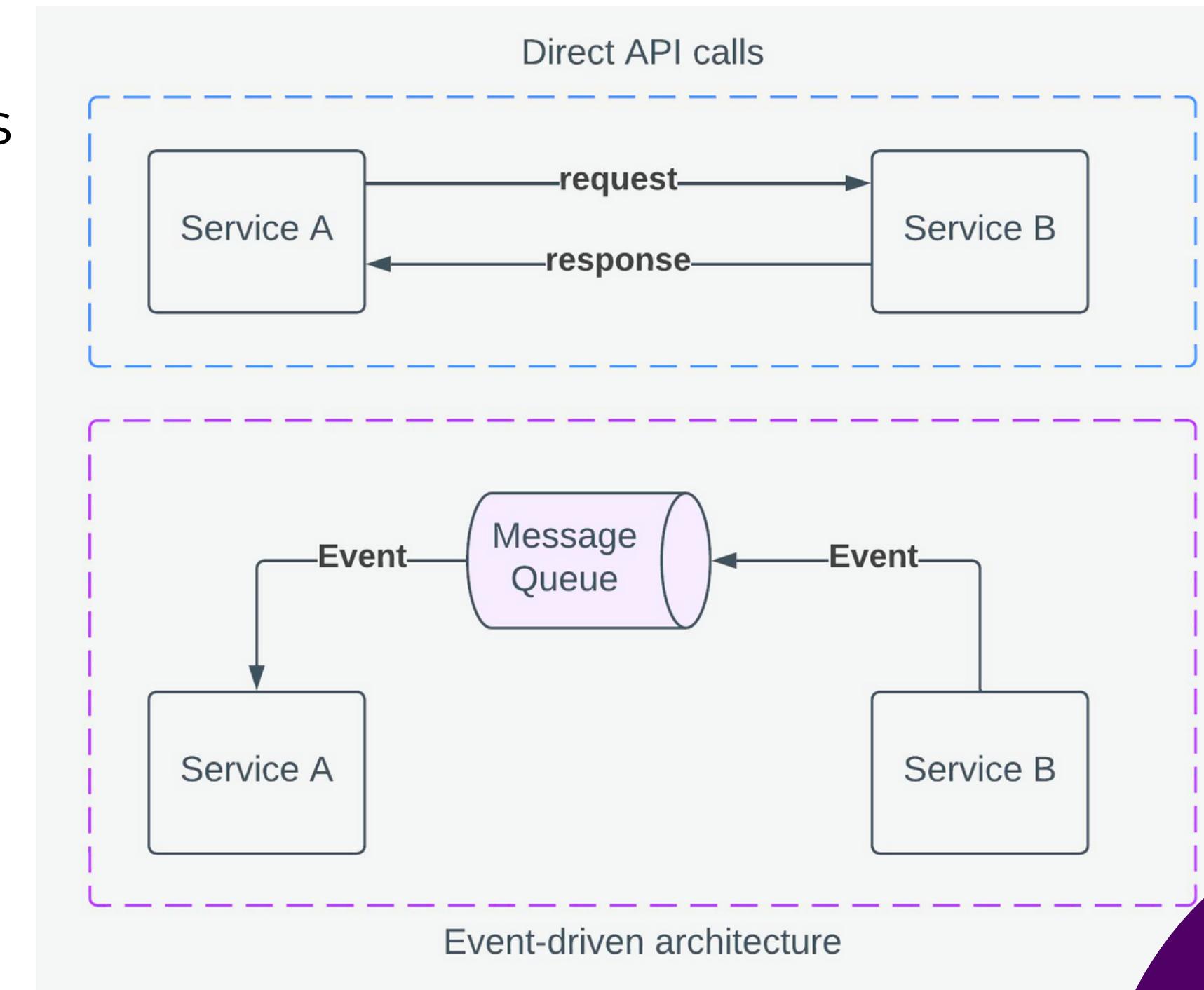
Everything can go wrong!

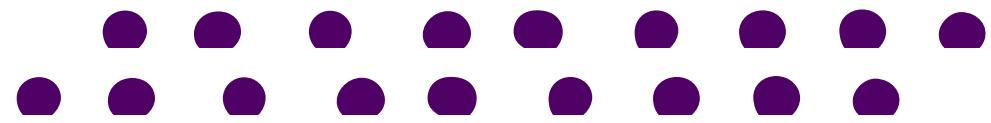
Direct API calls

- Request & response format issues
- Incorrect service behaviour
- Service outages & high latency

Event-driven architecture

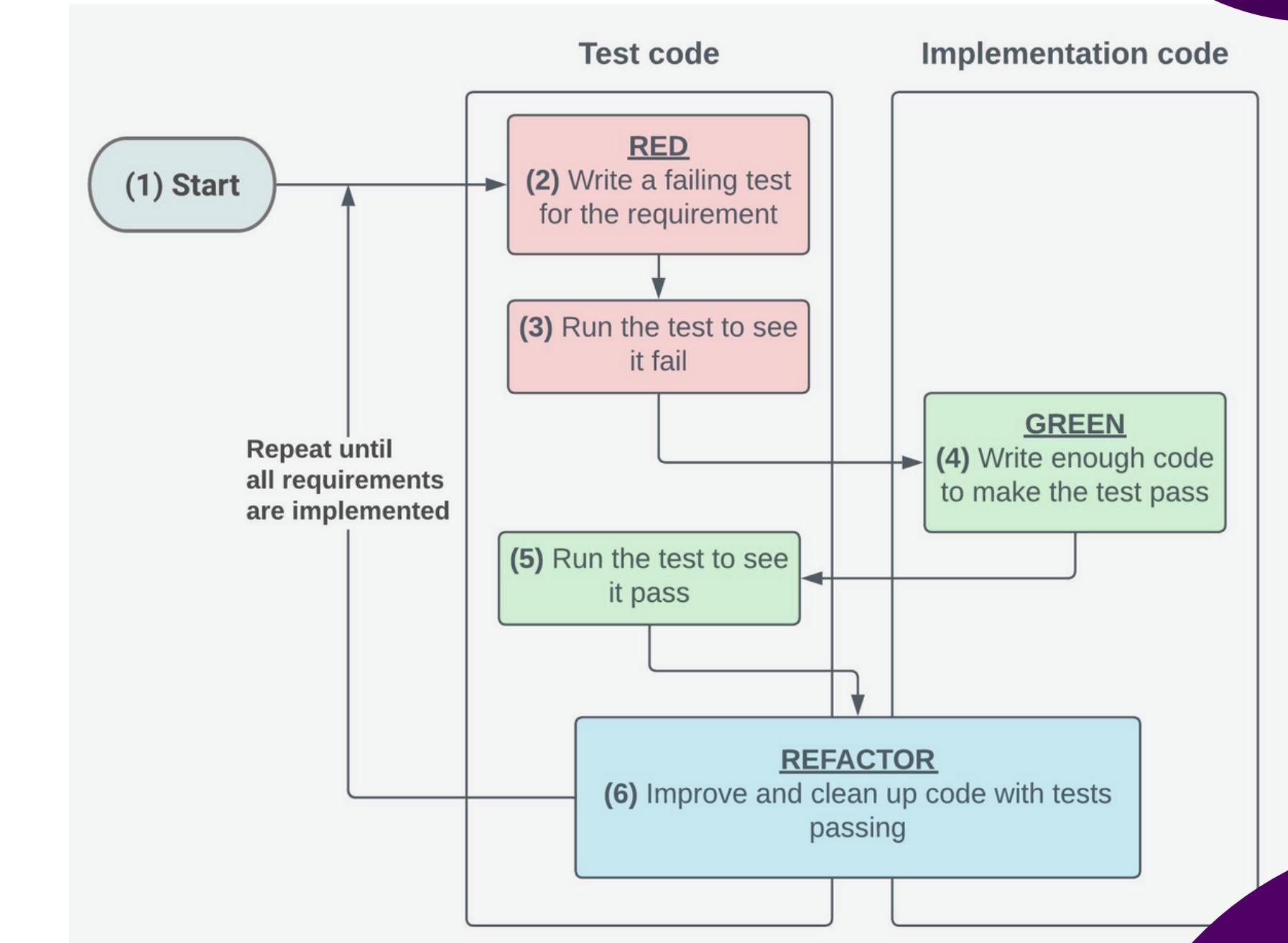
- Event payload format issues
- Message broker outages
- Events acknowledged but not processed

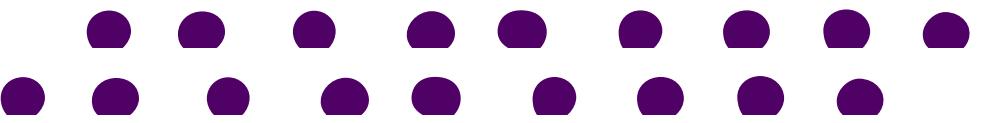




Let's start testing!

- **Unit Tests** are written alongside implementation using TDD
- Asserts the logic of the **Unit Under Test (UUT)** in isolation
- Test setup typically follows the **Arrange-Act-Assert (AAA)** structure



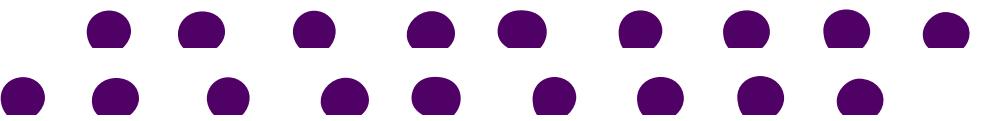


1 Unit testing in Go

- In Go, packages are the smallest unit under test as they are a **self-contained mini-API**.
- Tests are the **first clients/consumers** of our mini-APIs.
- Tests are functions that satisfy a few conventions.

```
func SayHello(name string) string {
    return fmt.Sprintf("Hello,%s!", name)
}

func TestSayHello(t *testing.T) {
    names := []string{"Anna", "Belle"}
    for _, n := range names {
        t.Run(n, func (t *testing.T) {
            want := "Hello, "+ n +"!"
            got := SayHello(n)
            if got != want {
                t.Fatalf("got:%s; want:%s", got, want)
            }
        })
    }
}
```

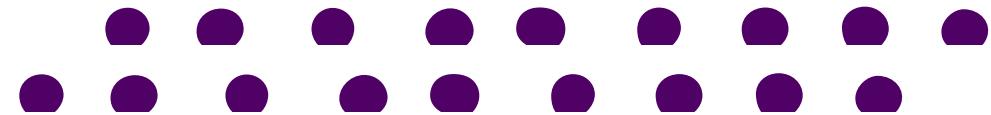
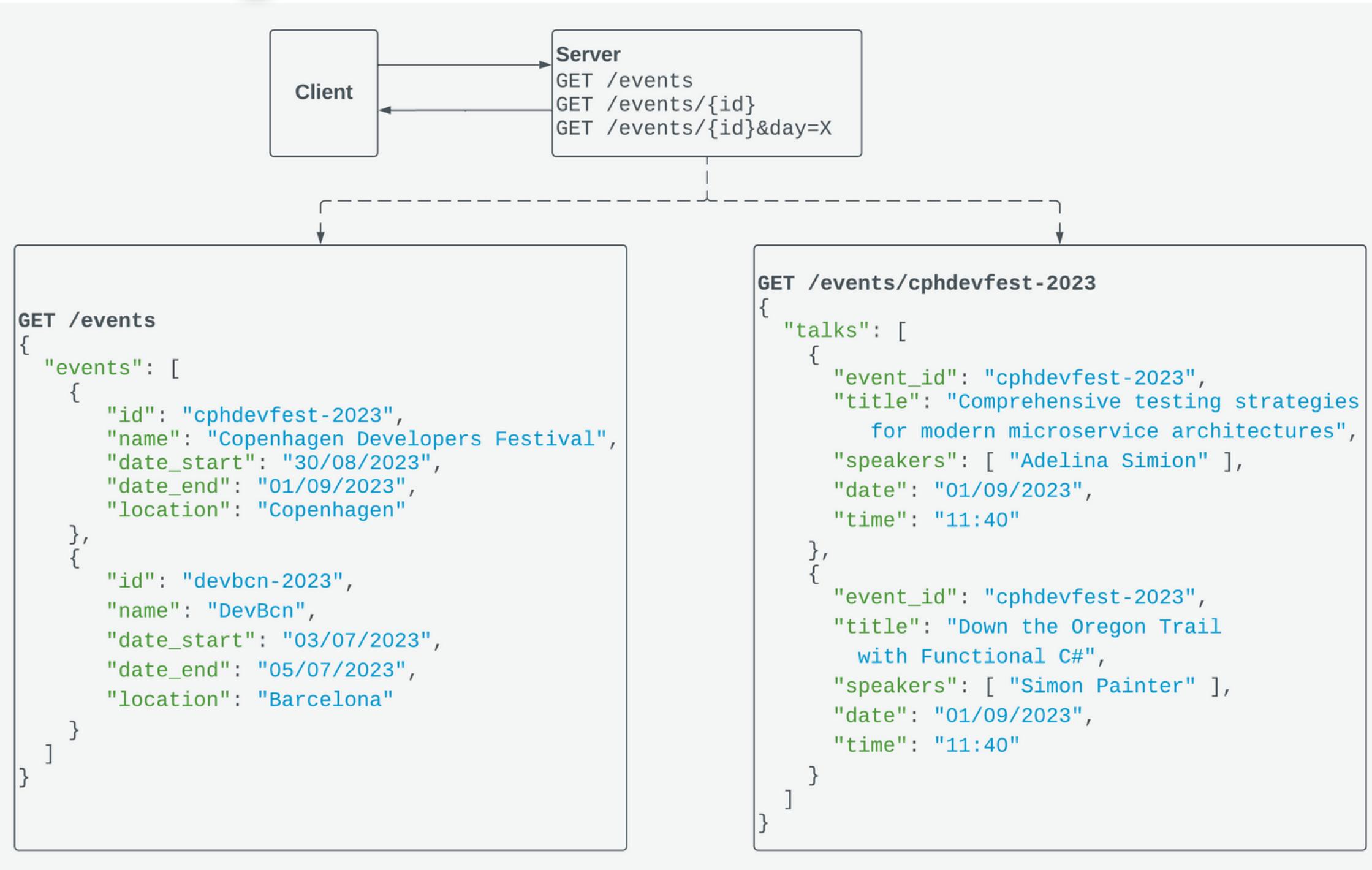


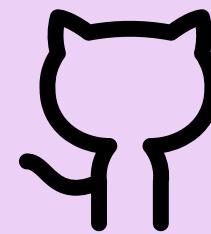
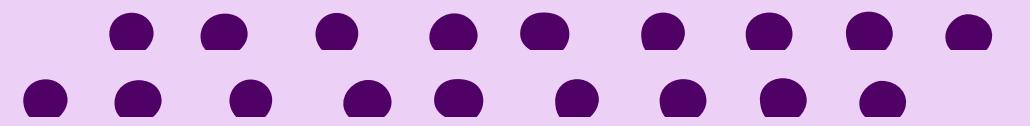
Fuzz testing

- Automated tests that are run using the same testing commands.
- **Fuzzed tests** generate values, making it easy to write many unit tests without having to manually write test cases.
- Help us to ensure **code robustness** and **cover edge cases**

```
func FuzzSayHello(f *testing.F) {
    f.Add("Anna")
    f.Add("Belle")
    f.Fuzz(func(t *testing.T, n string) {
        want := "Hello, "+ n +"!"
        got := SayHello(n)
        if got != want {
            t.Fatalf("got:%s; want:%s", got, want)
        }
    })
}
```

Introducing the Conference Talks service!





Show me the code!

github.com/addetz/testing-strategies-demo

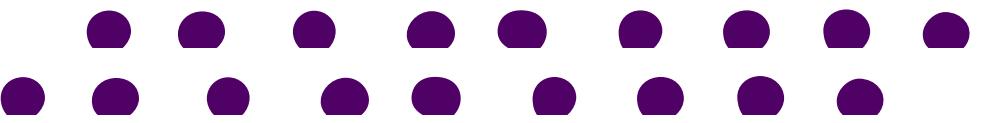




Moving up the testing pyramid!

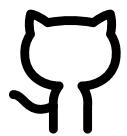
	<u>Integration tests</u>	<u>End-to-end tests</u>
Purpose	Test integration with external and internal modules	Test user flow and complete user experience
Cost	+ Fast & cheap	- Slow & expensive
Timing	+ Performed earlier in the SDLC	- Performed at the end of the development cycle, when application is running
Scope	+ Ensure that a component works with code that it does not own	- Ensure that specific user workflows perform correctly





The httpptest library

- Starts a test server under the hood and allows us to verify HTTP requests and responses
- Tests look very similar to client code, making it easier for us design client & server integrations



pkg.go.dev/net/http/httpptest

[Documentation](#)

Overview

Package httpptest provides utilities for HTTP testing.

Index ¶

Constants

func NewRequest(method, target string, body io.Reader) *http.Request

type ResponseRecorder

func NewRecorder() *ResponseRecorder

func (rw *ResponseRecorder) Flush()

func (rw *ResponseRecorder) Header() http.Header

func (rw *ResponseRecorder) Result() *http.Response

func (rw *ResponseRecorder) Write(buf []byte) (int, error)

func (rw *ResponseRecorder) WriteHeader(code int)

func (rw *ResponseRecorder) WriteString(str string) (int, error)

type Server

func NewServer(handler http.Handler) *Server

func NewTLSServer(handler http.Handler) *Server

func NewUnstartedServer(handler http.Handler) *Server

func (s *Server) Certificate() *x509.Certificate

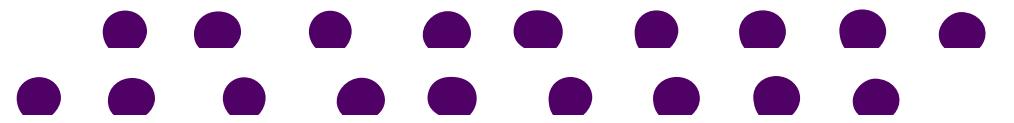
func (s *Server) Client() *http.Client

func (s *Server) Close()

func (s *Server) CloseClientConnections()

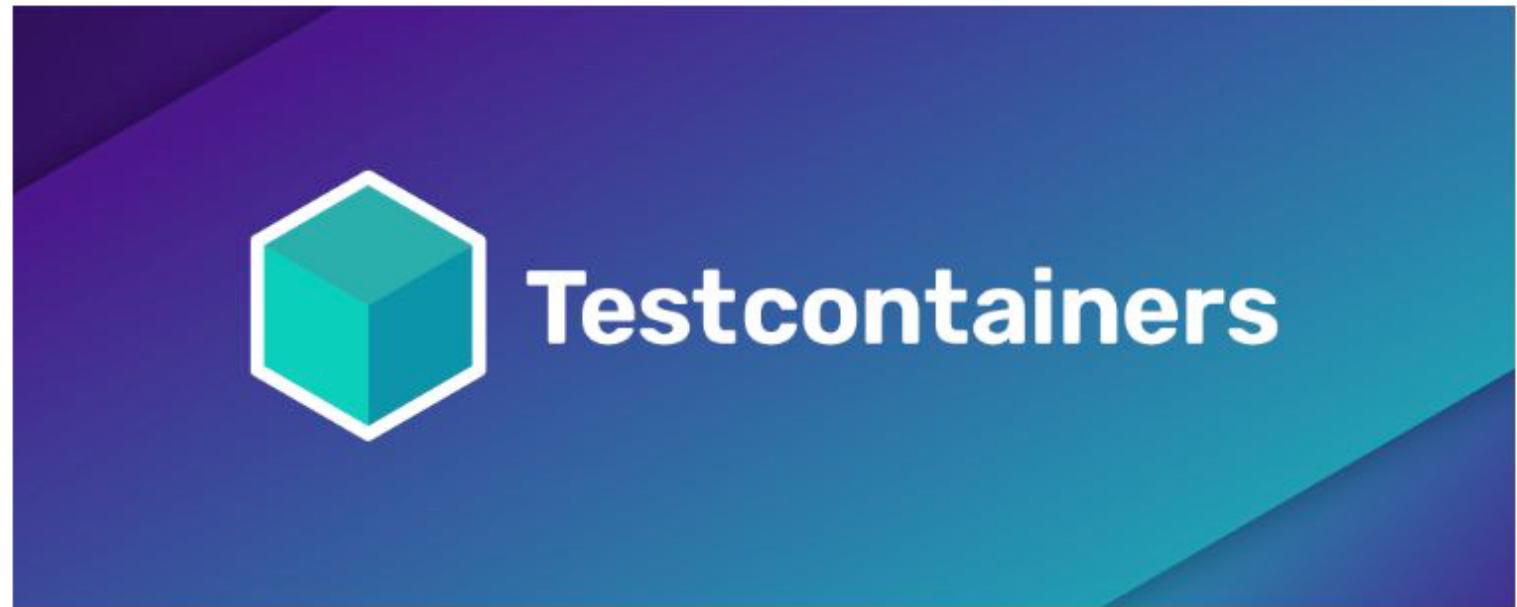
func (s *Server) Start()

func (s *Server) StartTLS()

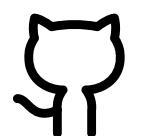


◆ The **testcontainers** project

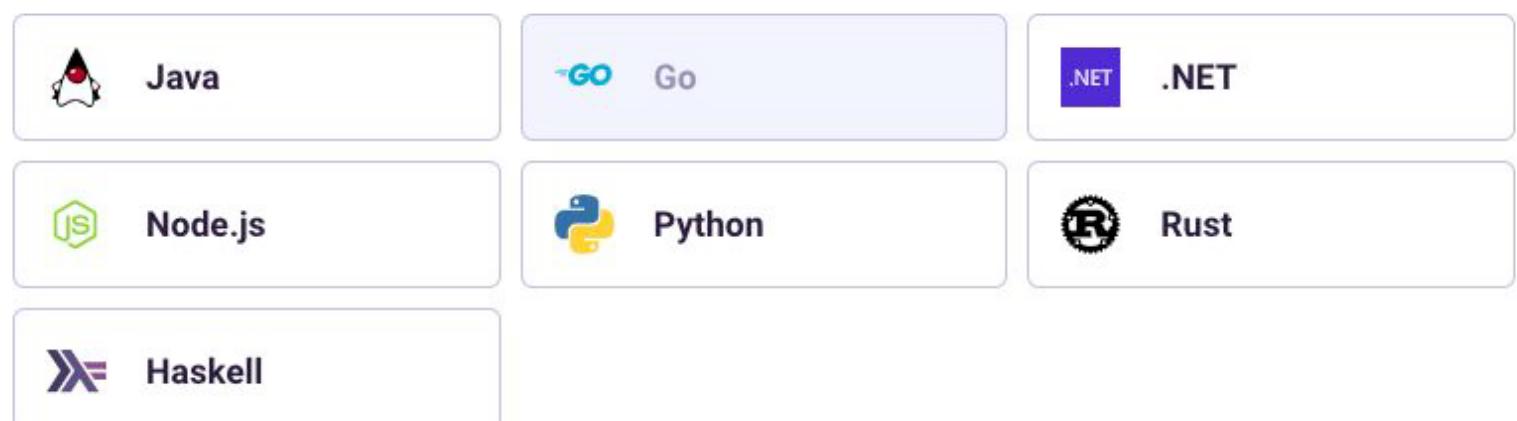
- A project which makes it easy to manage containers used for integration and end-to-end tests
- Bundles together container management together with code writing, simplifying test setup

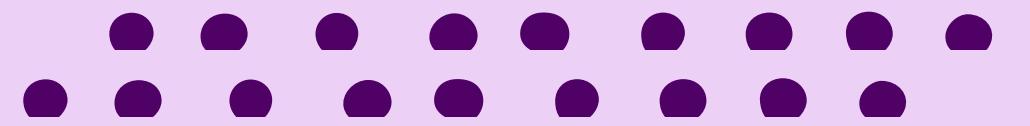


Not using Go? Here are other supported languages!

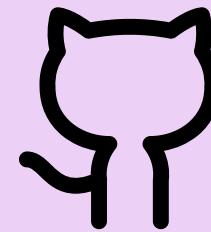


github.com/testcontainers



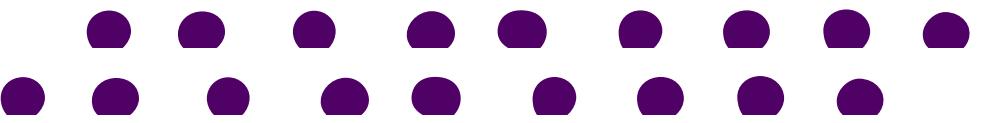


Show me the code!



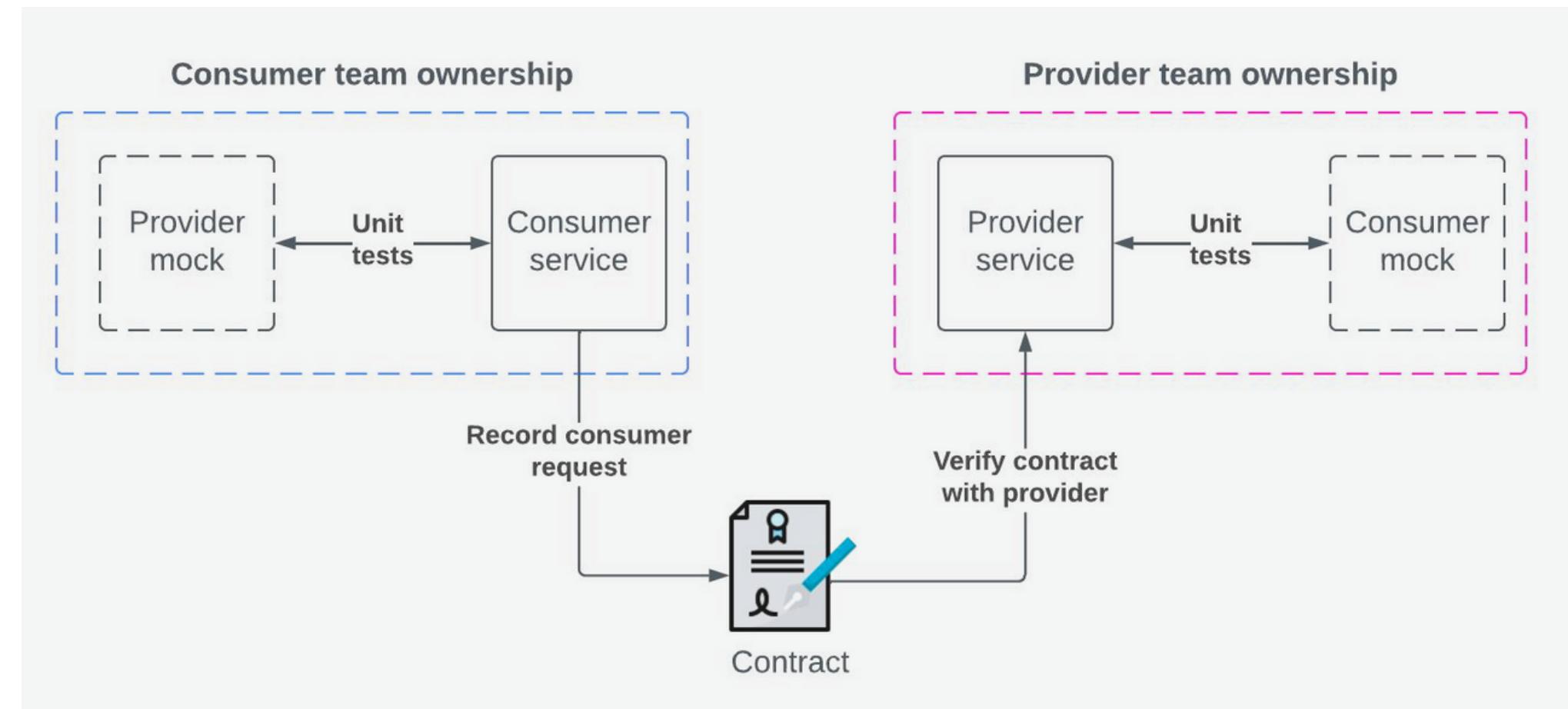
github.com/addetz/testing-strategies-demo

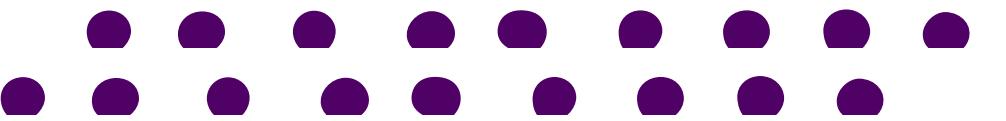




🤝 Contract testing

- The **consumer** issues the request for data from either a queue or service
- The **provider** responds to the request and sends the data
- The **contract** records the expectations of both the consumer and provider, allowing us to establish a common language





The pact tool

- Contract testing framework with libraries in over 10 languages
- Provides:
 - mocking
 - verification
 - domain-specific language (DSL)
 - playback capabilities

<> Documentation

Overview

[Consumer Tests](#) [Provider States](#)
[Matching](#) [Publishing Pacts to a Broker and Tagging Pacts](#)
[Provider Tests](#) [Using the Pact Broker with Basic authentication](#)
[Provider Verification](#) [Output Logging](#)

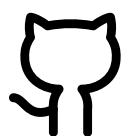
Pact Go enables consumer driven contract testing, providing a mock service and DSL for the consumer project, and interaction playback and verification for the service provider project.

Consumer Tests

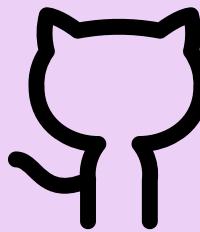
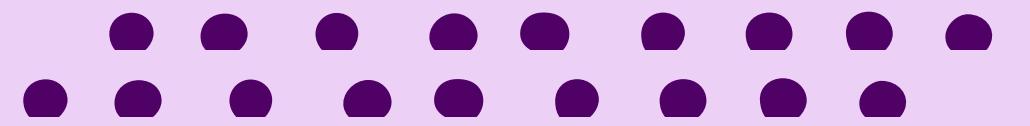
Consumer side Pact testing is an isolated test that ensures a given component is able to collaborate with another (remote) component. Pact will automatically start a Mock server in the background that will act as the collaborators' test double.

This implies that any interactions expected on the Mock server will be validated, meaning a test will fail if all interactions were not completed, or if unexpected interactions were found:

A typical consumer-side test would look something like this:



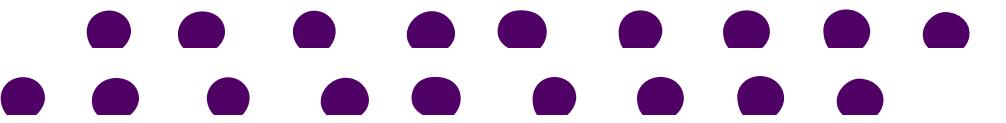
github.com/pact-foundation



Show me the code!

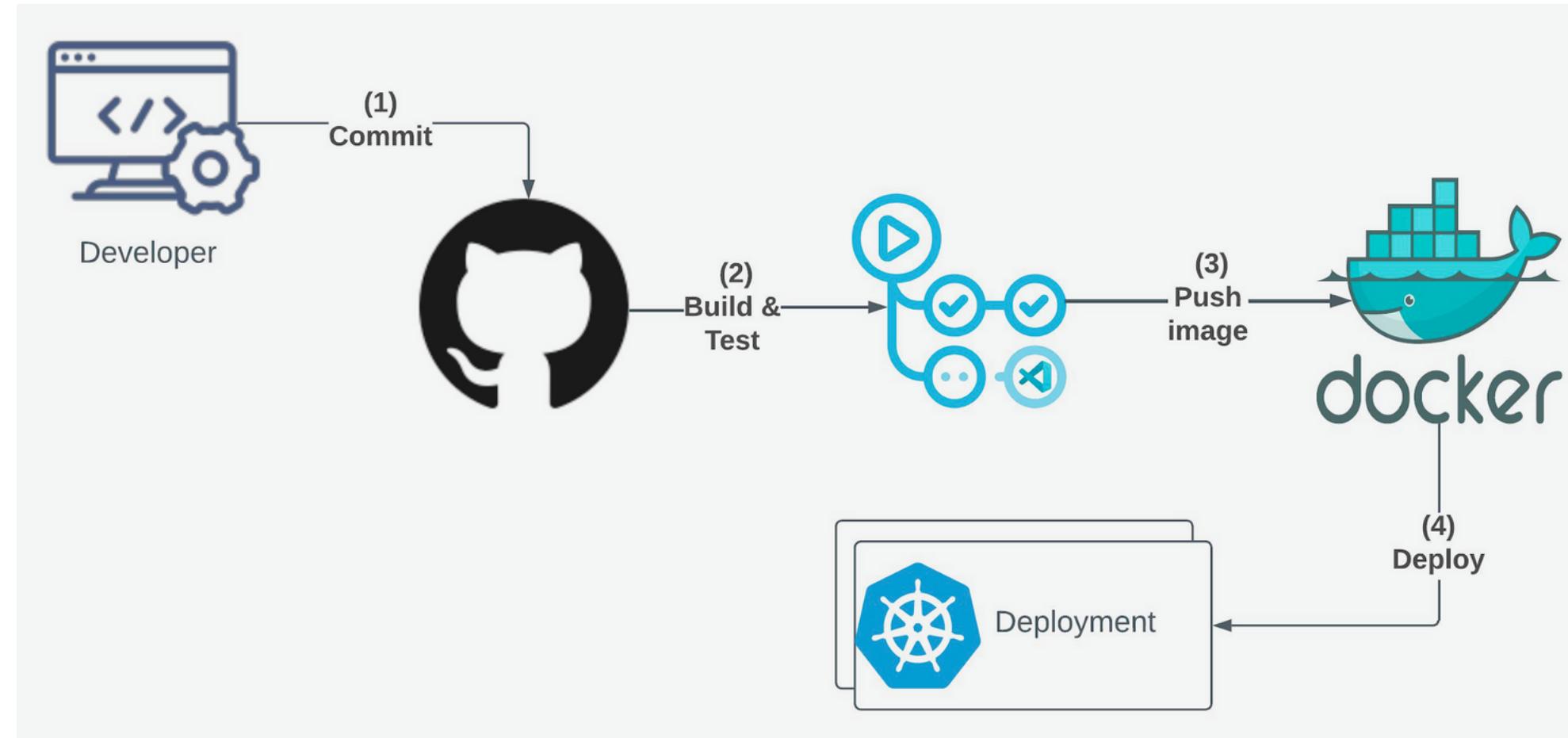
github.com/addetz/testing-strategies-demo

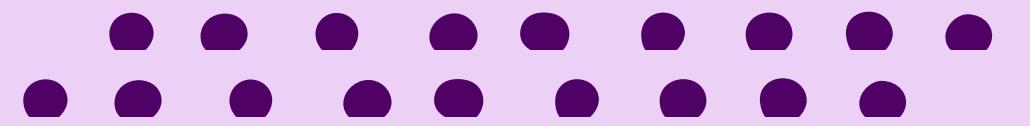




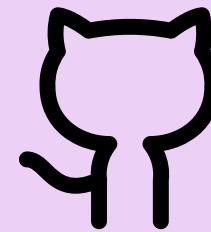
■ Making tests quality gates

- For consistency, tests should be run as part of the CI/CD pipelines to avoid outages, bugs and regressions
- Unit tests are usually run on all branches, while the costlier tests are run on the **main branch only**
- **GitHub Actions** are a popular solution for creating custom CI/CD pipelines





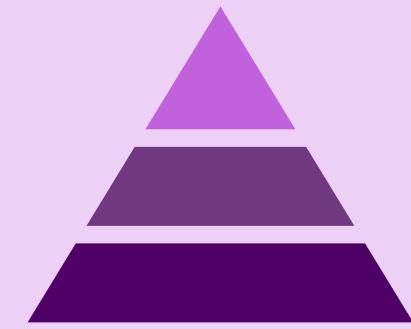
Show me the code!



github.com/addetz/testing-strategies-demo



✓ Summary



TYPES OF TESTS

Write tests at all levels: unit, integration, end-to-end and contract



QUALITY GATES

Tests should be your quality gates to prevent outages



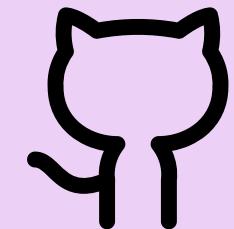
OWNERSHIP

Embrace a testing and quality culture in your teams



Thank you! ❤

Happy testing!



[github.com/addetz/testing-
strategies-demo](https://github.com/addetz/testing-strategies-demo)



classic_addetz



adelina-simion