

Comprehensive Testing Strategies For Modern Microservice Architectures

Adelina Simion

Technology Evangelist @ Form3



Nout me

- Paragelist @ Form3
- 💻 Java & Go engineer
- LinkedIn Learning instructor
- Author of "Test-Driven Development In Go"
- classic_addetz
- in adelina-simion
- addetz





About us

- Real-time payments processing platform
- Multi-cloud platform (AWS, GCP, Azure)
- Go, IaC (Terraform)
- SecDevOps culture
- Fully remote





Bug fixing

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

- A) <25%
- 25-50%
- >50%



Creator: jenifoto | Credit: Getty Images/iStockphoto Copyright: jenifoto



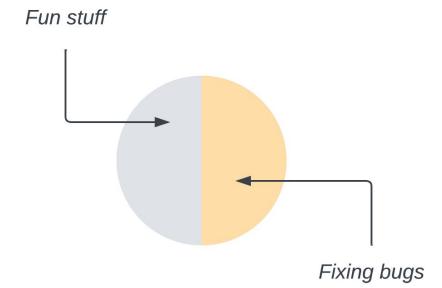
Bug fixing

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

- A) <25%
- B) 25-50%
- C) >50%

Developers spend 25-50% of their time fixing bugs, with some spending up to 75% of their time bug fixing.

(Source: Rollbar survey)

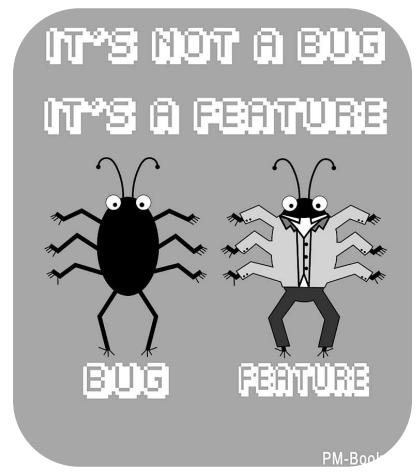






Common bug causes

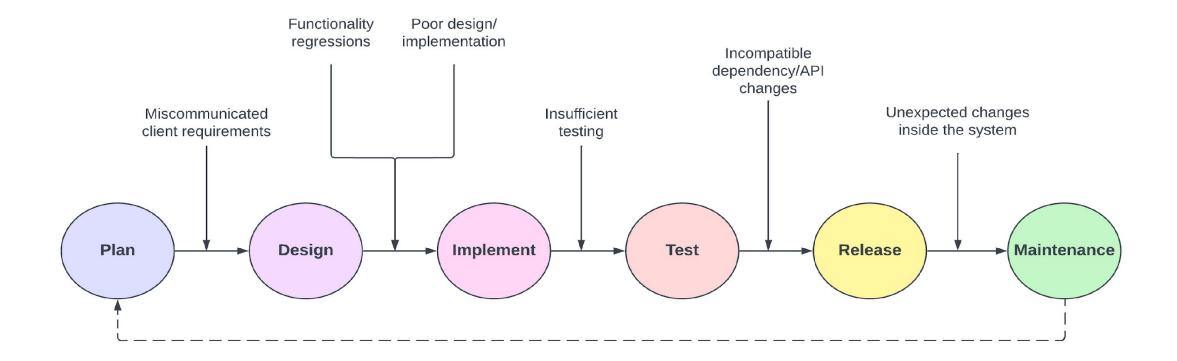
- Miscommunicated client requirements
- Poor design/implementation
- Insufficient testing
- Functionality regressions
- Incompatible dependency/API changes
- Unexpected changes inside the system



Creator: PM Books



Bugs in the SDLC

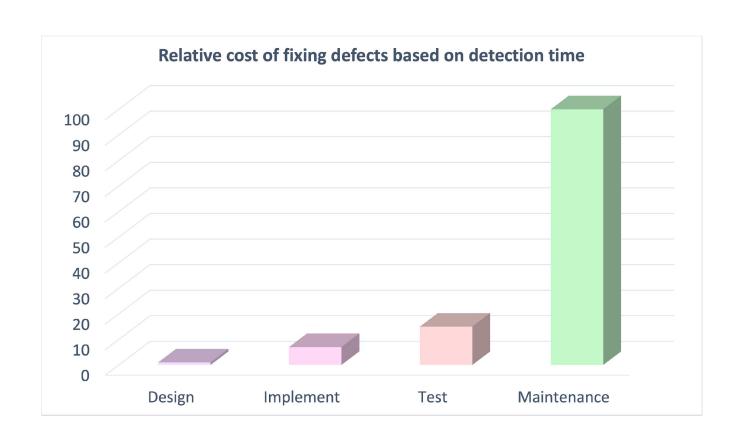




Exponential costs

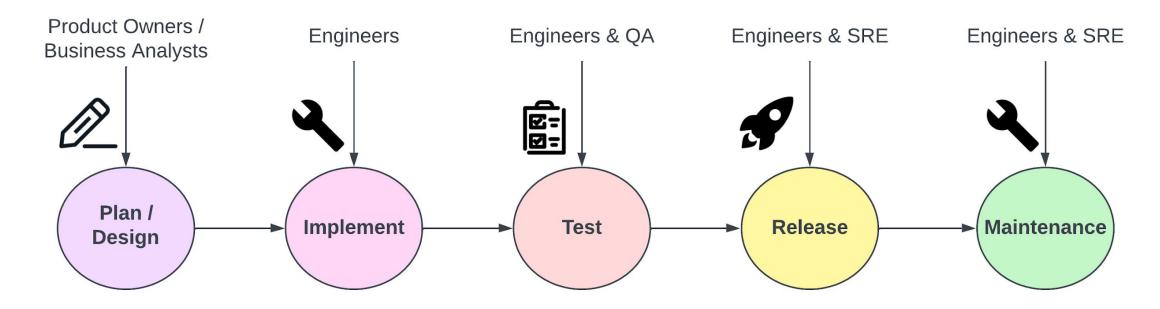
- Detect early, detect often to avoid delays and increased costs (the "shift left" motto)
- The cost of fixing bugs increases up to 100 times when identified after release.

(Source: *IBM study*)





An engineer's life ...

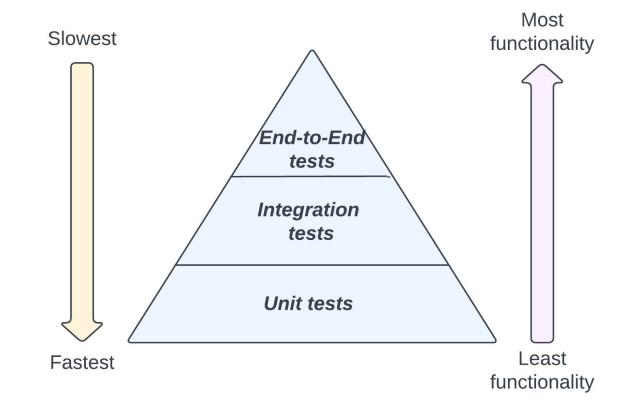


Main players in the SDLC



The testing pyramid

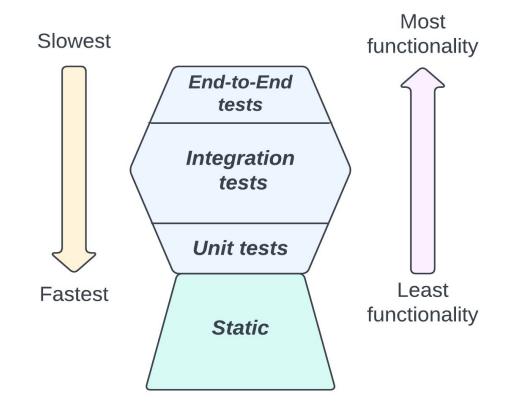
- **Unit tests** verify internal code mechanisms.
- **Integration tests** verify multiple units work well together.
- **End-to-end tests** verify user interactions.





The testing trophy

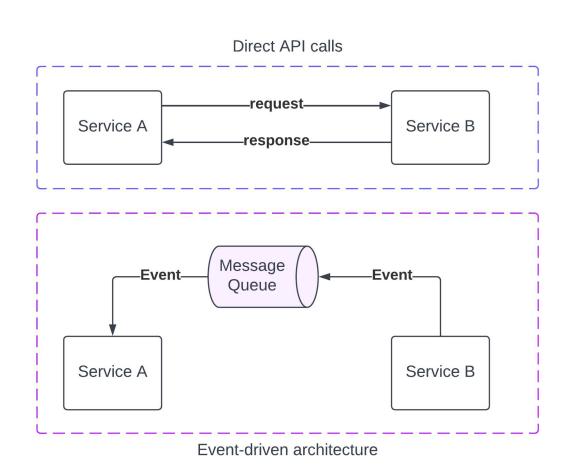
- Tests have diminishing returns and can also add a maintenance burden.
- Common static checks are:
 - Reviews
 - Static analysis
 - Checkstyle
 - Linters
- Code coverage should be in 80-90% range.





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.





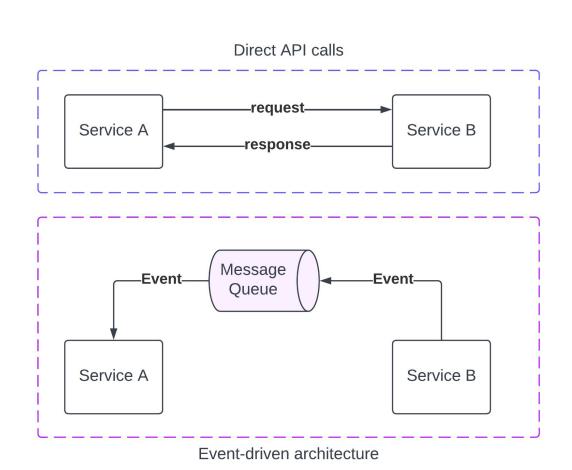
Everything can go wrong!

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

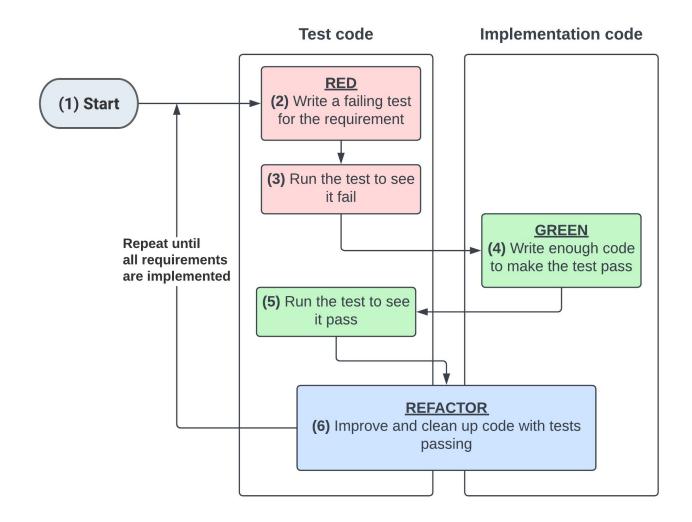




Let's start pulling the testing lever!

Unit tests

- 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 packages.
- 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

Edge cases coverage through unit tests

- 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



Introducing the Conference Talks service!

```
Server
                                 GET /events
  Client
                                 GET /events/{id}
                                 GET /events/{id}&day=X
GET /events
                                                 GET /events/uuid-1
  "events":
                                                    "talks": [
        "id": "uuid-1",
                                                         "title": "Comprehensive Testing Strategies For
       "name": "European Women in Tech",
       "date_start": "28/06/2023",
                                                            Modern Microservice Architectures",
                                                         "speakers": ["Adelina Simion"],
       "date_end": "29/06/2023",
                                                         "date": "29/06/2023",
       "location": "Amsterdam"
                                                         "time": "12:10"
       "id": "uuid-2",
                                                         "title": "Embracing Platform Engineering",
       "name": "DevBcn",
                                                         "speakers": ["Bukola Johnson"],
       "date_start": "03/07/2023",
                                                         "date": "29/06/2023",
       "date_end": "05/07/2023",
                                                         "time": "14:15"
        "location": "Barcelona"
```





Show me the code!

github.com/addetz/testing-strategies-demo







Moving up the testing pyramid!

	<u>Integration tests</u>	End-to-end tests
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 httptest 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 too.

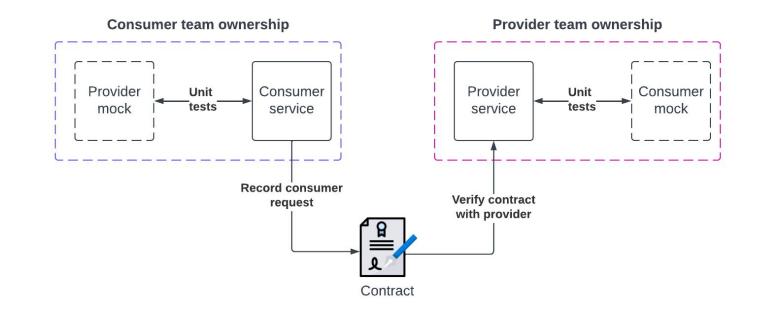


Ocumentation Overview Package httptest provides utilities for HTTP testing. Index ¶ Constants func NewRequest(method, target string, body io.Reader) *http.Request type ResponseRecorder func NewRecorder() *ResponseRecorder func (rw *ResponseRecorder) Flush() func (rw *ResponseRecorder) Header() http.Header 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()



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.





#EWIT23



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



Ocumentation

Overview

Consumer Tests
Matching
Provider Tests
Provider Verification

Provider States
Publishing Pacts to a Broker and Tagging Pacts
Using the Pact Broker with Basic authentication
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:

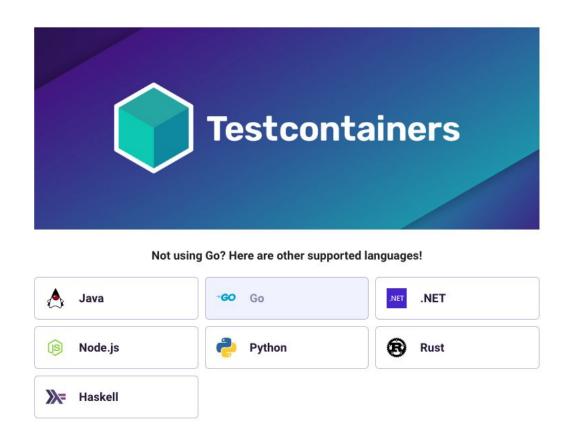


The testcontainers project

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



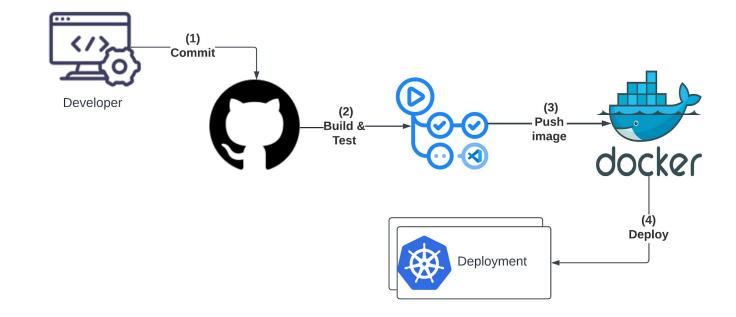
github.com/testcontainers/testcontainers-go





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 longer running 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





- Unit tests verify the behaviour of the isolated parts of our code.
- Integration tests extend test scope without losing performance.
- Contract testing verifies the API integration between microservices.



- Unit tests verify the behaviour of the isolated parts of our code.
- Integration tests extend test scope without losing performance.
- Contract testing verifies the API integration between microservices.
- Tests should be your quality gates to prevent outages.

Summary

- Unit tests verify the behaviour of the isolated parts of our code.
- Integration tests extend test scope without losing performance.
- Contract testing verifies the API integration between microservices.
- Tests should be your quality gates to prevent outages.

Embrace a testing and quality culture in your teams!



Thank you for your time!

Hope you have fun testing your applications \(\text{\text{\text{\text{\text{\text{\text{\text{test}}}}}}}



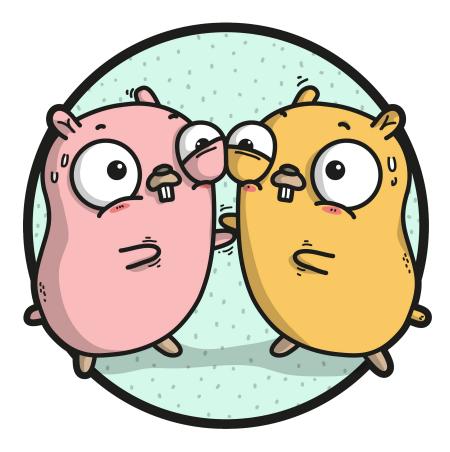


github.com/addetz/testing-strategies-demo



Let's be pals!

- classic_addetz
- in adelina-simion
- addetz



Creator: https://github.com/ashleymcnamara/gophers