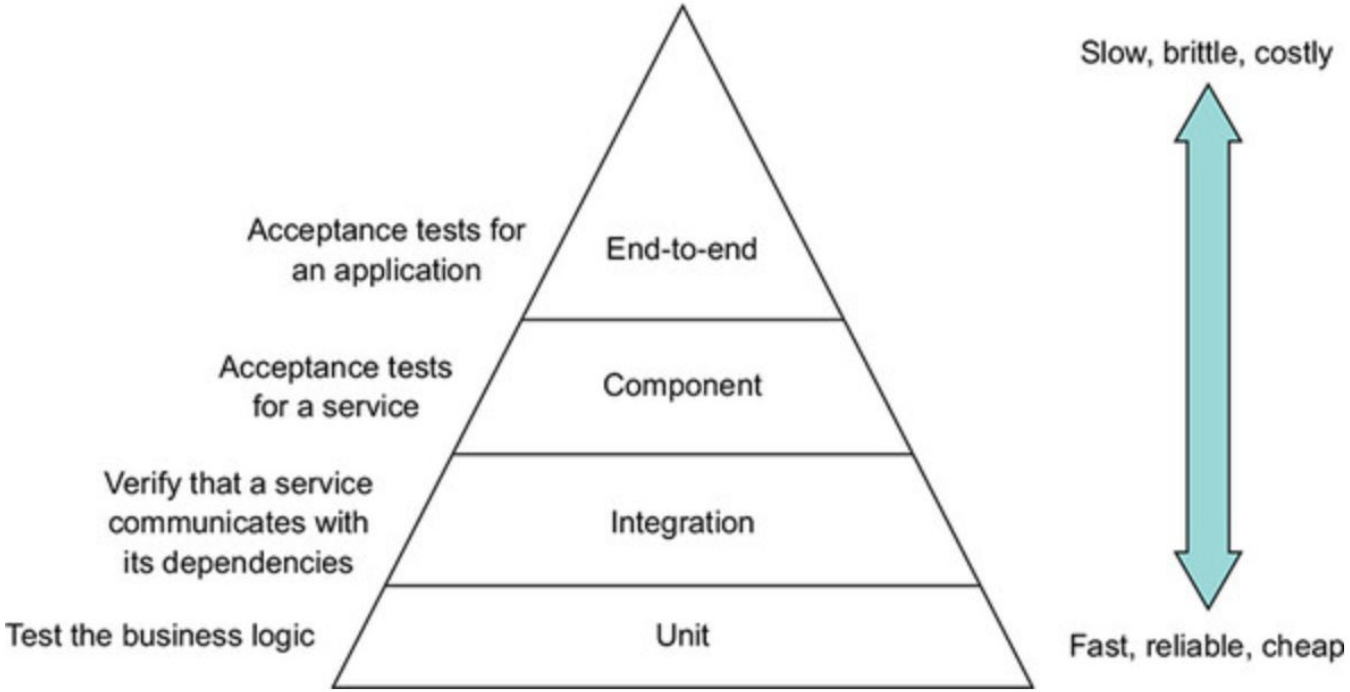
Testing

Outlines

- Introduction
- Unit Test
- Test Doubles: Stub And Mock
- Integration Test

Introduction - Test Pyramid



Unit Tests

“In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.” Wikipedia.

Go Test

- `go test` subcommand is a test driver
- `*_test.go` files are built with `go test`

```
go test
go test ./…
go test -v
go test -run regexp
go test -cover -coverprofile=output.out .
go tool cover -html output.out
go help test
```

Go Test

- Test functions
- Benchmark functions
- Example functions

Test Functions

- Begin with `Test`.

```
func TestPalindrome(t *testing.T)  {
	testCases := []struct{
		input string
		output bool
	}{
		{"aa", true},
		{"ábá", true},
		{"Aba", true},
		{"ab", false},
		{ "Abc", false},
	}
	for _, tc := range testCases {
		if IsPalindrome(tc.input) != tc.output {
			t.Errorf("IsPalindrome(%s) != %t", tc.input, tc.output)
		}
	}
}
```

Benchmark Functions

- Begin with `Benchmark`.
- Run `go test -bench regexp` to report mean execution time of operations

```
func BenchmarkPalindrome(b *testing.B)  {
      // If there is any complex initialization
      // perform it here and call b.ResetTimer to reset timer.
      // Therefore, it will not add to the measured time of each iteration.
      for i := 0; i < b.N; i++ {
            IsPalindrome("aaaa")
      }
}
```

Output

```
BenchmarkPalindrome-8           20000000                84.4 ns/op
PASS
ok      github.com/xuanit/testing/palindrome    1.785s
```

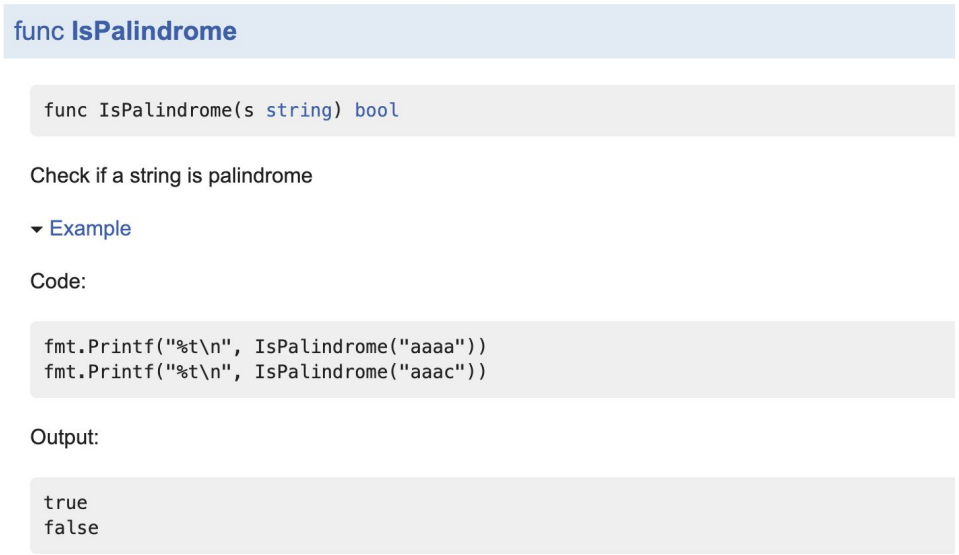
Example Functions

- Start with `Example`
- Provide machine-checked documentation

```
func ExampleIsPalindrome()  {
      fmt.Printf("%t", IsPalindrome("aba"))
      // go test will check output is printed into standard output
      // Output:
      // true
}
```

Example Functions

- Examples are shown when running `godoc`



func **IsPalindrome**

```
func IsPalindrome(s string) bool
```

Check if a string is palindrome

▾ Example

Code:

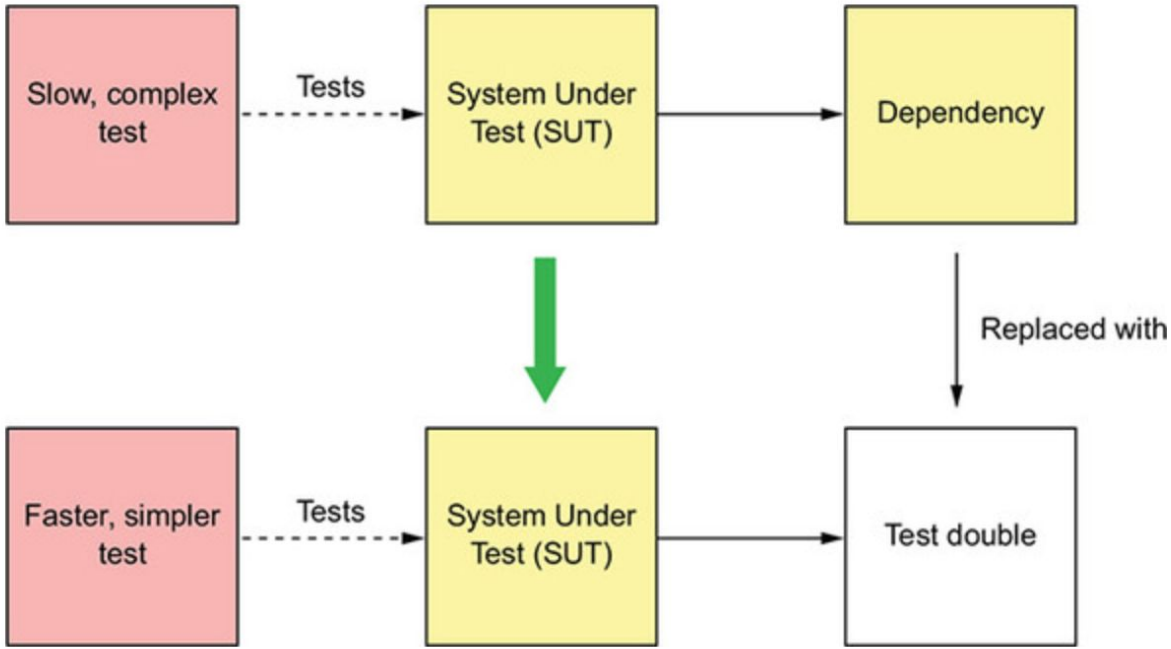
```
fmt.Printf("%t\n", IsPalindrome("aaaa"))
fmt.Printf("%t\n", IsPalindrome("aaac"))
```

Output:

```
true
false
```

Test Doubles

- A test double is an object that simulates the behavior of the dependency



Test Doubles - Stub, Mock

- Stub has a "fixed" set of "canned" responses that are specific to your test(s)
- Mock has a set of expectations about calls that are made. If these expectations are not met, the test fails
- Useful packages in Go: golang/mock, stretchr/testify/mock, vektra/mockery

Test Doubles - Stub, Mock

```
func TestDoSomeThing(t *testing.T)  {
      mockToDoRep := &mocks.ToDo{}
      mockToDoRep.On("Get", req.Id).Return(toDo, nil)

      service := Service{ToDoRepo: mockToDoRep}

      err :=  service.DoSomeThing()

      assert.Nil(t, err)
      mockToDoRep.AssertExpectations(t)
}
```

Integration Test

Integration tests verify that a service can interact with infrastructure services such as databases and other application services

- Persistence integration test
- Integration testing interactions between services. A good strategy is to use consumer-driven contract tests

Integration Test - Persistence

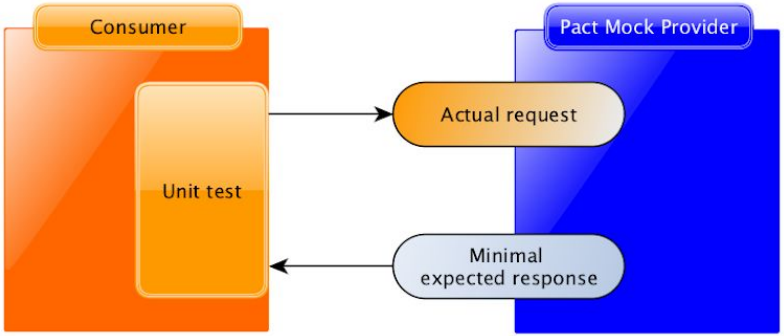
- Persistence integration tests check if the database access logic works as expected
- Docker can be used to run databases

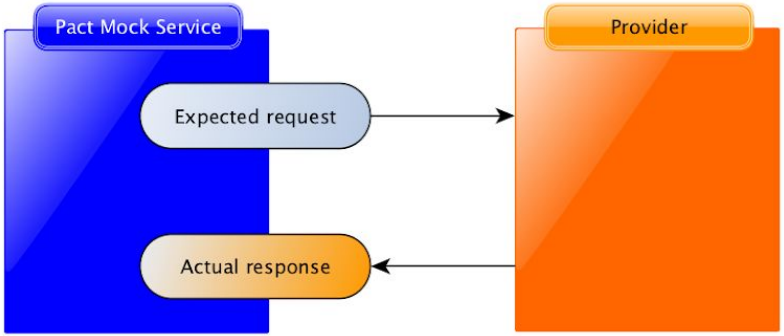
```
func (s *ToDoRepositorySuite) SetupSuite()  {
      // connect and set-up db
}

func (s *ToDoRepositorySuite) TestInsert()  {
      item := &pb.Todo{ Id: "new_item", Title: "meeting"}
      err := s.todoRep.Insert(item)

      s.Nil(err)

      newTodo, err := s.todoRep.Get(item.Id)
      s.Nil(err)
      s.Equal(item, newTodo)
}
```

Integration Test - Consumer Driven Contract Test

- Ensure that consumers and providers all agree on the API
- Consumer-side tests
- Provider-side tests
- Pact family Frameworks, Spring Cloud Contract, Mocker (an internal library at Grab)

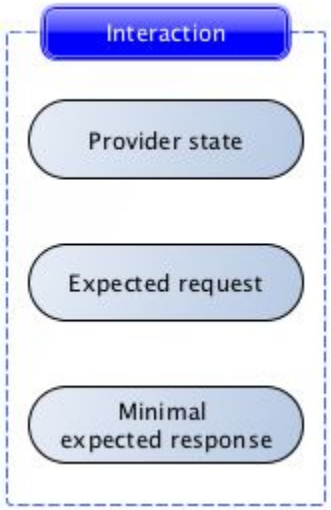
Pact Go


Consumer
Unit test
Pact Mock Provider
Actual request
Minimal
expected response


Pact Mock Service
Expected request
Actual response
Provider

Pact Go

- A Contract is a collection of agreements between a client (Consumer) and an API (Provider) that describes the interactions that can take place between them



Pact Interaction

```
	pact.
		AddInteraction().
		Given("User foo exists").
		UponReceiving("A request to get foo").
		WithRequest(dsl.Request{
			Method:  "GET",
			Path:    dsl.String("/foobar"),
			Headers: dsl.MapMatcher{"Content-Type": dsl.String("application/json"),
"Authorization": dsl.String("Bearer 1234")},
			Body: map[string]string{
				"name": "billy",
			},
		}).
		WillRespondWith(dsl.Response{
			Status:  200,
			Headers: dsl.MapMatcher{"Content-Type":
dsl.String("application/json")},
			Body:    dsl.Match(&User{}),
		})
```

References

- https://en.wikipedia.org/wiki/Unit_testing
- https://martinfowler.com/articles/mocksArentStubs.html
- https://www.jamesshore.com/Blog/Dependency-Injection-Demystified.html
- https://docs.pact.io
- The Go Programming Language
- Microservices Patterns: With examples in Java