



June 2nd - 4th 2014  
Copenhagen, Denmark



GRAILS



gradle

# IDIOMATIC SPOCK



Rob Fletcher



@rfletcherEW





**SPOCK WITH STYLE**

# REFACTOR WITH HELPER METHODS



```
def "user can purchase uniform"() {  
  given:  
    to LoginPage  
    loginForm.with {  
      email = "aaron.pyle@teleworm.us"  
      password = "ahroh6tie4oCh"  
      submit()  
    }  
  
  and:  
    to ProductPage, "Starfleet uniform (TOS)"  
    addToBasketButton.click()  
  
  and:  
    at BasketPage  
    checkoutButton.click()  
  
  expect:  
    at CheckoutPage
```

```
    when:  
      checkoutForm.with {  
        street = "2415 Woodland Avenue"  
        city = "Garyville"  
        state = "LA"  
        zip = "70051"  
        cardType = "Mastercard"  
        cardNo = "5555700540812370"  
        expiry = "11/2017"  
        cvc = "003"  
        submit()  
      }  
  
    then:  
      message == "Purchase successful"  
  
    and:  
      1 * paymentMock.charge("USD", 65)  
  }
```







```
@Shared user = [  
  email: "aaron.pyle@teleworm.us"  
  password: "ahroh6tie4oCh"  
  street: "2415 Woodland Avenue"  
  city: "Garyville"  
  state: "LA"  
  zip: "70051"  
  cardType: "Mastercard"  
  cardNo: "5555700540812370"  
  expiry: "11/2017"  
  cvc: "003"  
]
```

```
@Shared product = [  
  name: "Starfleet uniform (TOS)"  
  price: ["USD", 65]  
]
```

```
def "user can purchase uniform"() {  
  given:  
    loggedInAs user  
    addToBasket product  
  
  when:  
    completeCheckout user  
  
  then:  
    message == "Purchase successful"  
  
  and:  
    1 * paymentMock.charge(product.price)  
}
```

# SHARED HELPER METHODS

```
@Category(GebSpec)
class StoreInteractions {
    void loggedInAs(user) {}

    void addToBasket(String productName) {}

    void completeCheckout(user) {}
}
```

```
@Mixin(StoreInteractions)
class CheckoutSpec extends GebSpec {}
```

# ...IN GROOVY 2.3

```
trait StoreInteractions {  
    void loggedInAs(user) {}  
  
    void addToBasket(String productName) {}  
  
    void completeCheckout(user) {}  
}  
  
class CheckoutSpec extends GebSpec  
    implements StoreInteractions {}
```



# FUNCTIONAL GROOVY FOR ASSERTIONS

when:

```
def results = ships.findByAllegiance("Federation")
```

then:

```
results.size() == 3
```

```
results[0].allegiance == "Federation"
```

```
results[1].allegiance == "Federation"
```

```
results[2].allegiance == "Federation"
```



when:

```
def results = ships.findByAllegiance("Federation")
```

then:

```
results.every {  
  it.allegiance == "Federation"  
}
```

Condition not satisfied:

```
results.every { it.allegiance == "Federation" }
```

```
|  
|  
| false
```

```
[Gr'oth, Enterprise, Constitution, Constellation, M'Char, Haakona]
```



when:

```
def results = ships.findByAllegiance("Federation")
```

then:

```
results.allegiance.every {  
  it == "Federation"  
}
```

Condition not satisfied:

```
results.allegiance.every { it == "Federation" }
```

```
|           |  
|           |  
|           | false
```

```
| [Klingon, Federation, Federation, Federation, Klingon, Romulan]  
[Gr'oth, Enterprise, Constitution, Constellation, M'Char, Haakona]
```



# ONE [LOGICAL] ASSERTION

*The Single Responsibility Principle for tests*

```
def "can list ships by allegiance"() {  
  when:  
    def results = ships.findByAllegiance("Federation")  
  
  then:  
    results.allegiance.every {  
      it == "Federation"  
    }  
  
  and:  
    results instanceof ImmutableList  
}
```



# BLOCK GRAMMAR

*when / then or given / expect*

when:

crew << "Kirk" << "Bones" << "Scotty"

then:

"Science officer" in crew.openPositions



given:

```
crew << "Kirk" << "Bones" << "Scotty"
```

expect:

```
"Science officer" in crew.openPositions
```

given:

crew << "Kirk"

when:

crew << "Picard"

then:

thrown TooManyCaptainsException



# MOCKS & STUBS

The right tool for the job

```
def paymentMock = Mock(PaymentService)

def "payment is taken at the end of checkout"() {
  given:
    loggedInAs user
    addToBasket product

  when:
    completeCheckout user

  then:
    1 * paymentMock.charge(product.price)
}
```



```
def "presents ships as a bullet list"() {  
  given:  
    def shipList = ["Enterprise", "Constellation"]  
  
  and:  
    def shipStore = Mock(ShipStore)  
  
  when:  
    def output = ships.render()  
  
  then:  
    1 * shipStore.list() >> shipList  
  
  and:  
    $(output).find("ul > li").text() == shipList  
}
```

```
def "presents ships as a bullet list"() {  
  given:  
    def shipList = ["Enterprise", "Constellation"]  
  
  and:  
    def shipStore = Stub(ShipStore) {  
      list() >> shipList  
    }  
  
  when:  
    def output = ships.render()  
  
  then:  
    $(output).find("ul > li")*.text() == shipList  
}
```



# CLEANING UP RESOURCES

```
def handle = DBI.open("jdbc:h2:mem:test")  
@Subject ships = handle.attach(ShipStore)
```

```
def setup() {  
    ships.createTable()  
}
```

```
def cleanup() {  
    ships.dropTable()  
    handle.close()  
}
```



```
@AutoCleanup handle = DBI.open("jdbc:h2:mem:test")
```

```
@AutoCleanup("dropTable")
```

```
@Subject ships = handle.attach(ShipStore)
```

```
def setup() {  
    ships.createTable()  
}
```

```
@Shared
@AutoCleanup handle = DBI.open("jdbc:h2:mem:test")
```

```
@Shared
@AutoCleanup("dropTable")
@Subject ships = handle.attach(ShipStore)
```

```
def setupSpec() {
    ships.createTable()
}
```



# ENFORCE PRECONDITIONS

Another use for the *expect* block

```
@Shared handle = DBI.open("jdbc:h2:mem:test")

def "writes to the database when data is added"() {
    given:
        def user = new User(name: "Spock")

    when:
        userStore.persist(user)

    then:
        selectInt("select count(*) from user") == 1
}
```



```
@Shared handle = DBI.open("jdbc:h2:mem:test")

def "writes to the database when data is added"() {
    expect:
    selectInt("select count(*) from user") == 0

    given:
    def user = new User(name: "Spock")

    when:
    userStore.persist(user)

    then:
    selectInt("select count(*) from user") == 1
}
```

```
def "a completed job cannot be restarted"() {  
  given: "all tasks succeed"  
  task1.execute(_) >> new DefaultTaskResult(SUCCEEDED)  
  task2.execute(_) >> new DefaultTaskResult(SUCCEEDED)  
  task3.execute(_) >> new DefaultTaskResult(SUCCEEDED)  
  
  and: "the job has been run"  
  def jobExecution = launchJob()  
  
  expect: "the job to have completed successfully"  
  jobExecution.exitStatus == ExitStatus.COMPLETED  
  
  when: "the job is restarted"  
  resumeJob jobExecution  
  
  then: "an exception is thrown"  
  thrown JobInstanceAlreadyCompleteException  
}
```



**ACCESS PREVIOUS VALUES**

```
@Subject stack = new Stack()
```

```
def "size increases when we add items to a stack"() {  
    when:  
        stack.push "foo"  
  
    then:  
        stack.size() == 1  
}
```



```
@Subject stack = new Stack()

def "size increases when we add items to a stack"() {
  given:
    def oldSize = stack.size()

  when:
    stack.push "foo"

  then:
    stack.size() == oldSize + 1
}
```

```
@Subject stack = new Stack()
```

```
def "size increases when we add items to a stack"() {  
    when:  
        stack.push "foo"  
  
    then:  
        stack.size() == old(stack.size()) + 1  
}
```





# TESTS AS DOCUMENTATION



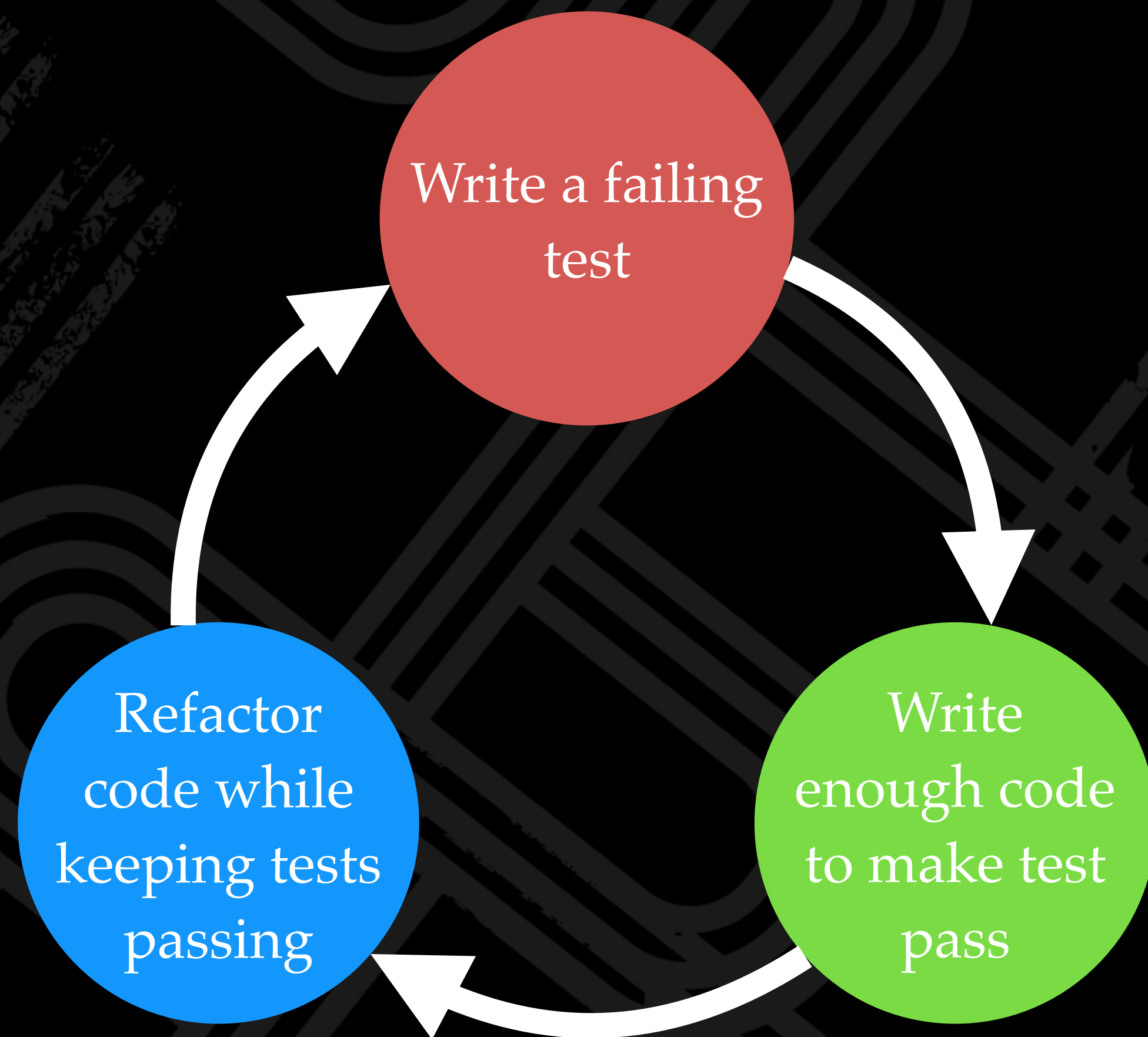
*“Programs must be written for people to read, and only incidentally for machines to execute.”*

—Abelson and Sussman

*“Always code as if the guy who ends up  
maintaining your code will be a violent  
psychopath who knows where you live.”*

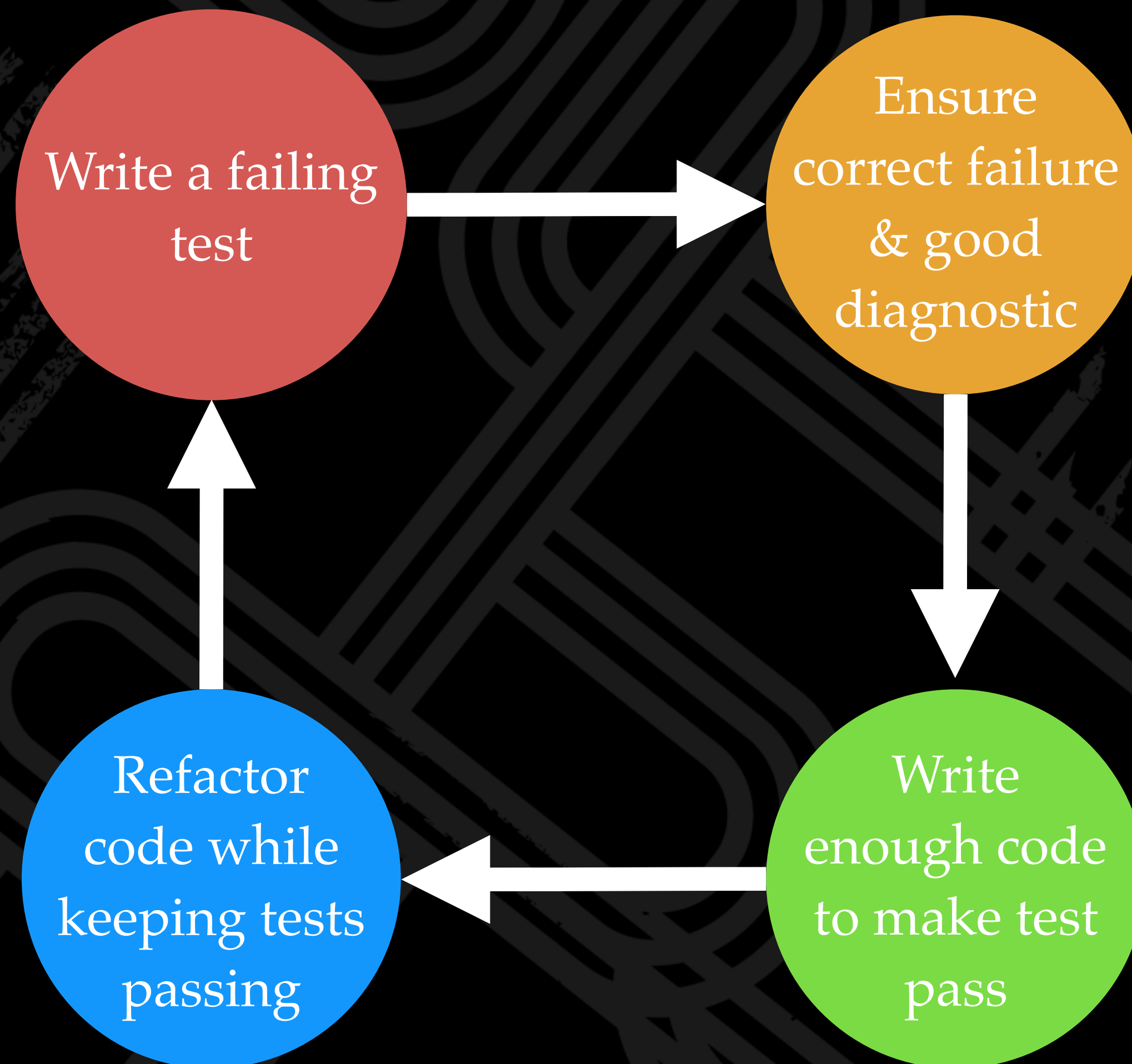
—M. Golding

# RED... GREEN... REFACTOR





# RED... AMBER... GREEN... REFACTOR



# THE PRIME DIRECTIVE

# NEVER

TRUST A TEST YOU  
HAVEN'T SEEN FAIL





# USE DOCUMENTATION ANNOTATIONS



```
@Subject ship = new Starship("Enterprise")

@Subject(Starship)

@Issue("https://github.com/robletcher/betamax/issues/37")

@See("http://atompunkera.tumblr.com/")

@Title("a readable title")

@Narrative("""as a developer
            I want to die
            so I can end the pain""")
```

# EFFECTIVE UNROLLING

```
@Unroll  
def "rejects an input value of '#input'"()
```



```
@Unroll("#input converted to #format is '#output'")  
def "translates strings to another format"()
```

```
@Unroll  
class ParameterizedSpec extends Specification
```



**USE UNROLL DESCRIPTIONS**

```
def "can tell if the string '#string' is an integer or not"() {  
  expect:  
    string.isInteger() == shouldBeInteger  
  
  where:  
    string | shouldBeInteger  
    "ABC"  | false  
    "123"  | true  
    "1.2"  | false  
    "1 2"  | false  
    "12a"  | false  
}
```



- OK can tell if the string 'ABC' is an integer or not
- OK can tell if the string '123' is an integer or not
- OK can tell if the string '1.2' is an integer or not
- OK can tell if the string '1 2' is an integer or not
- OK can tell if the string '1a2' is an integer or not

```
@Unroll
def "the string '#string' is #description"() {
    expect:
    string.isInteger() == expected

    where:
    string | expected
    "ABC"  | false
    "123"  | true
    "1.2"  | false
    "1 2"  | false
    "12a"  | false

    description = expected ? "an integer" : "not an integer"
}
```



```
@Unroll("the string '#string' is #description")
def "identifies strings that are integers"() {
    expect:
    string.isInteger() == expected

    where:
    string | expected
    "ABC"  | false
    "123"  | true
    "1.2"  | false
    "1 2"  | false
    "12a"  | false

    description = expected ? "an integer" : "not an integer"
}
```

- OK the string 'ABC' is not an integer
- OK the string '123' is an integer
- OK the string '1.2' is not an integer
- OK the string '1 2' is not an integer
- OK the string '12a' is not an integer



# SEPARATE TEST DATA FROM TEST LOGIC

where blocks without *@Unroll*



```
def "passes data from stream to callback"() {  
  given:  
    def callback = Mock(Function)  
  
  and:  
    service.source = Stub(StreamSource) {  
      connect() >> Observable.from("foo", "bar", "baz")  
    }  
  
  when:  
    service.readStream(callback)  
  
  then:  
    1 * callback.apply("foo")  
    1 * callback.apply("bar")  
    1 * callback.apply("baz")  
}
```

```
def "passes data from stream to callback"() {  
  given:  
    def callback = Mock(Function)  
  
  and:  
    service.source = Stub(StreamSource) {  
      connect() >> Observable.from(*data)  
    }  
  
  when:  
    service.readStream(callback)  
  
  then:  
    data.each {  
      1 * callback.apply(it)  
    }  
  
  where:  
    data = ["foo", "bar", "baz"]  
}
```





# SPOCK ANTI-PATTERNS



# TEST ORGANIZATION

Think *behavior* not units of code

```
@AutoCleanup
@Shared handle = DBI.open("jdbc:h2:mem:test")
@Subject ships = handle.attach(ShipStore)

def setup() {
  ships.createTable()

  insert("ship", "Enterprise", "Federation")
  insert("ship", "Constellation", "Federation")
  insert("ship", "M'Char", "Klingon")
}

def "can find by allegiance"() {
  when:
    def results = ships.findByAllegiance("Federation")

  then:
    results.allegiance.every {
      it == "Federation"
    }
}
```

```
def "can find by name"() {
  when:
    def result = ships.findByName(name)

  then:
    result.name == name

  where:
    name = "M'Char"
}

def "inserting writes to the database"() {
  when:
    ships << new Ship(name, allegiance)

  then:
    count("ship", name: name) == 1

  where:
    name = "Haakona"
    allegiance = "Romulan"
}
```

# TAUTOLOGICAL TESTS



# FALSE MONIKER TESTING

```
@Subject def ships = new ShipStore()

def "can find ships by allegiance ordered by age"() {
  given:
  ships <<
    new Ship("Enterprise", "Federation", Year.of(2245)) <<
    new Ship("Adventure", "Federation", Year.of(2376)) <<
    new Ship("Haakona", "Romulan", Year.of(2357))

  expect:
  def matches = ships.findByAllegianceNewestFirst("Federation")
  matches.name == ["Enterprise", "Haakona", "Adventure"]
}
```

```
def "can find ships by allegiance ordered by age"() {  
  given:  
    ships <<  
      new Ship("Enterprise", "Federation", Year.of(2245)) <<  
      new Ship("Adventure", "Federation", Year.of(2376)) <<  
      new Ship("Haakona", "Romulan", Year.of(2357))  
  
  expect:  
    def matches = ships.findByAllegianceNewestFirst("Federation")  
    matches.allegiance.every { it == "Federation" }  
    matches.enteredService == matches.enteredService.sort().reverse()  
}
```



# INTERROGATING INTERNAL STATE

# WHEN BLOCK BLOAT

```
def "items can be viewed in the basket after selection"() {  
  when:  
    products.each {  
      to ProductPage, it  
      addToBasket()  
    }  
    to BasketPage  
  
  then:  
    basket.size() == 3  
    basket.items.title == products  
  
  where:  
    products = ["Starfleet uniform", "Tricorder", "Phaser"]  
}
```



```
def "items can be viewed in the basket after selection"() {  
  given:  
    products.each {  
      to ProductPage, it  
      addToBasket()  
    }  
  
  when:  
    to BasketPage  
  
  then:  
    basket.size() == 3  
    basket.items.title == products  
  
  where:  
    products = ["Starfleet uniform", "Tricorder", "Phaser"]  
}
```

# FAIL-FAST ASSERTIONS

# STEPWISE SPECS



@Stepwise

```
class StackSpec extends Specification {
```

```
  @Shared @Subject stack = new Stack()
```

```
  @Shared value = "foo"
```

```
  def "can push to the stack"() {
```

```
    expect:
```

```
    stack.push(value) == value
```

```
  }
```

```
  def "stack should have content"() {
```

```
    expect:
```

```
    stack.peek() == value
```

```
  }
```

```
  def "can pop from the stack"() {
```

```
    expect:
```

```
    stack.pop() == value
```

```
  }
```

```
  def "the stack should be empty"() {
```

```
    expect:
```

```
    stack.empty()
```

```
  }
```

```
}
```





# THINKING OUTSIDE THE BOX

Strange new worlds of testing



# TCK SPECIFICATIONS

One specification, multiple implementations









```
abstract class ShipStoreSpec<T extends ShipStore> extends Specification {  
  
  @Subject T ships  
  
  def "can insert a new ship"() {  
    when:  
      ships.insert(new Ship("Enterprise", "Federation"))  
  
    then:  
      ships.list().size() == old(ships.list().size()) + 1  
  }  
  
  def "can find ships by allegiance"() {  
    // ...  
  }  
}
```

```
class MemoryShipStoreSpec extends ShipStoreSpec<MemoryShipStore> {  
  def setup() {  
    ships = new MemoryShipStore()  
  }  
}
```

```
class PersistentShipStoreSpec extends ShipStoreSpec<PersistentShipStore> {  
  
  @AutoCleanup Handle handle  
  
  def setup() {  
    handle = DBI.open("jdbc:h2:mem:test")  
    ships = handle.attach(PersistentShipStore)  
    ships.createTable()  
  }  
  
  def cleanup() {  
    ships.dropTable()  
  }  
}
```



- ▼  MemoryShipStoreSpec (idiomaticspock.tck)
  -  can insert a new ship
  -  can find ships by allegiance
- ▼  PersistentShipStoreSpec (idiomaticspock.tck)
  -  can insert a new ship
  -  can find ships by allegiance

# DATA-DRIVEN PARAMETERIZATION

Beyond data tables

```
@Unroll
```

```
void "the #table table has a primary key"() {
```

```
    expect:
```

```
    with(handle.connection) { Connection connection ->
```

```
        connection.metaData.getPrimaryKeys(null, null, table).next()
```

```
    }
```

```
where:
```

```
    table << tableNames
```

```
}
```



```
@Shared @AutoCleanup Handle handle
@Shared Collection<String> tableNames = []

def setupSpec() {
  handle.connection.with { connection ->
    def rs = connection.metaData.getTables(null, null, "%", ["TABLE"] as String[])
    while (rs.next()) {
      tableNames << rs.getString(3)
    }
  }
}
```

# CONDITIONAL TESTS

```
@IgnoreIf({ javaVersion <= 1.7 })
```

```
@IgnoreIf({ env.SKIP_INTEGRATION_TESTS == "yes" })
```

```
@IgnoreIf({ properties."os.name" =~ /Windows.*/ })
```



```
static boolean isReachable(String url) {  
    try {  
        def connection = url.toURL().openConnection()  
        connection.connectTimeout = 1000  
        connection.connect()  
        true  
    } catch (IOException ex) {  
        false  
    }  
}
```

```
@IgnoreIf({ !isReachable("http://www.netflix.bv/") })  
class NeedsRealNetworkConnectionSpec extends Specification {
```

**@Memoized**

```
static boolean isReachable(String url) {  
    try {  
        def connection = url.toURL().openConnection()  
        connection.connectTimeout = 1000  
        connection.connect()  
        true  
    } catch (IOException ex) {  
        false  
    }  
}
```

```
@IgnoreIf({ !isReachable("http://www.netflix.bv/") })  
class NeedsRealNetworkConnectionSpec extends Specification {
```

# TEST JAVASCRIPT WITH NASHORN



```
@Shared engine = new ScriptEngineManager().getEngineByName("nashorn")
@Shared @Subject moment

def setupSpec() {
    getClass().getResourceAsStream("/moment.js").withReader { reader ->
        engine.eval reader
    }

    moment = engine.invokeFunction("moment")
}
```

```
@Unroll
def "The date #date in friendly format is #expectedResult"() {
    expect:
    engine.invokeMethod(moment, "from", date.toString()) == expectedResult

    where:
    date | expectedResult
    now().plusDays(2) | "2 days ago"
    now().plusMinutes(2) | "2 minutes ago"
    now() | "a few seconds ago"
    now().minusDays(1) | "in a day"
}
```





# COOL FEATURES

*Fascinating*



# UNROLL AUTOCOMPLETION

```
@Unroll("#string converted to #style is '#ex'")
def "can convert case"() {
    expect:
    string.convert(style) == expected

    where:
    string      | style || expected
    "foo bar baz" | CAMEL || "FooBarBaz"
    "foo bar baz" | KEBAB || "foo-bar-baz"
    "foo bar baz" | SNAKE || "foo_bar_baz"
}
```

# INPUT & OUTPUT PARAMETERS

```
def "can convert case"() {  
  expect:  
  string.convert(style) == expected  
  
  where:  
  string | style || expected  
  "foo bar baz" | CAMEL || "FooBarBaz"  
  "foo bar baz" | KEBAB || "foo-bar-baz"  
  "foo bar baz" | SNAKE || "foo_bar_baz"  
}
```

# TYPED PARAMETERS

```
def "can convert case"(String string, CaseFormat style, String expected) {  
  expect:  
  string.convert(style) == expected  
  
  where:  
  string      | style || expected  
  "foo bar baz" | CAMEL || "FooBarBaz"  
  "foo bar baz" | KEBAB || "foo-bar-baz"  
  "foo bar baz" | SNAKE || "foo_bar_baz"  
}
```



# JUNIT RULES

```
@Rule TemporaryFolder temporaryFolder = new TemporaryFolder()

def "can copy a resource from classpath to a file"() {
    given:
        def resource = getClass().getResource("/test.txt")
        def file = temporaryFolder.newFile()

    when:
        resource.withReader { file << it }

    then:
        file.text == resource.text
}
```

# HAMCREST MATCHERS





# STUPID SPOCK TRICKS

# IMPORT ALIASING

```
import java.lang.Void as Should
```

```
class MyBddSpec {
```

```
    Should "exhibit some behavior"() {
```

```
        // ...
```

```
    }
```

```
}
```



# MULTIPLE WHEN/THEN BLOCKS

```
given:  
def stack = new Stack()  
  
when:  
stack.push "foo"  
  
then:  
stack.pop() == "foo"  
  
when:  
stack.pop()  
  
then:  
thrown EmptyStackException
```