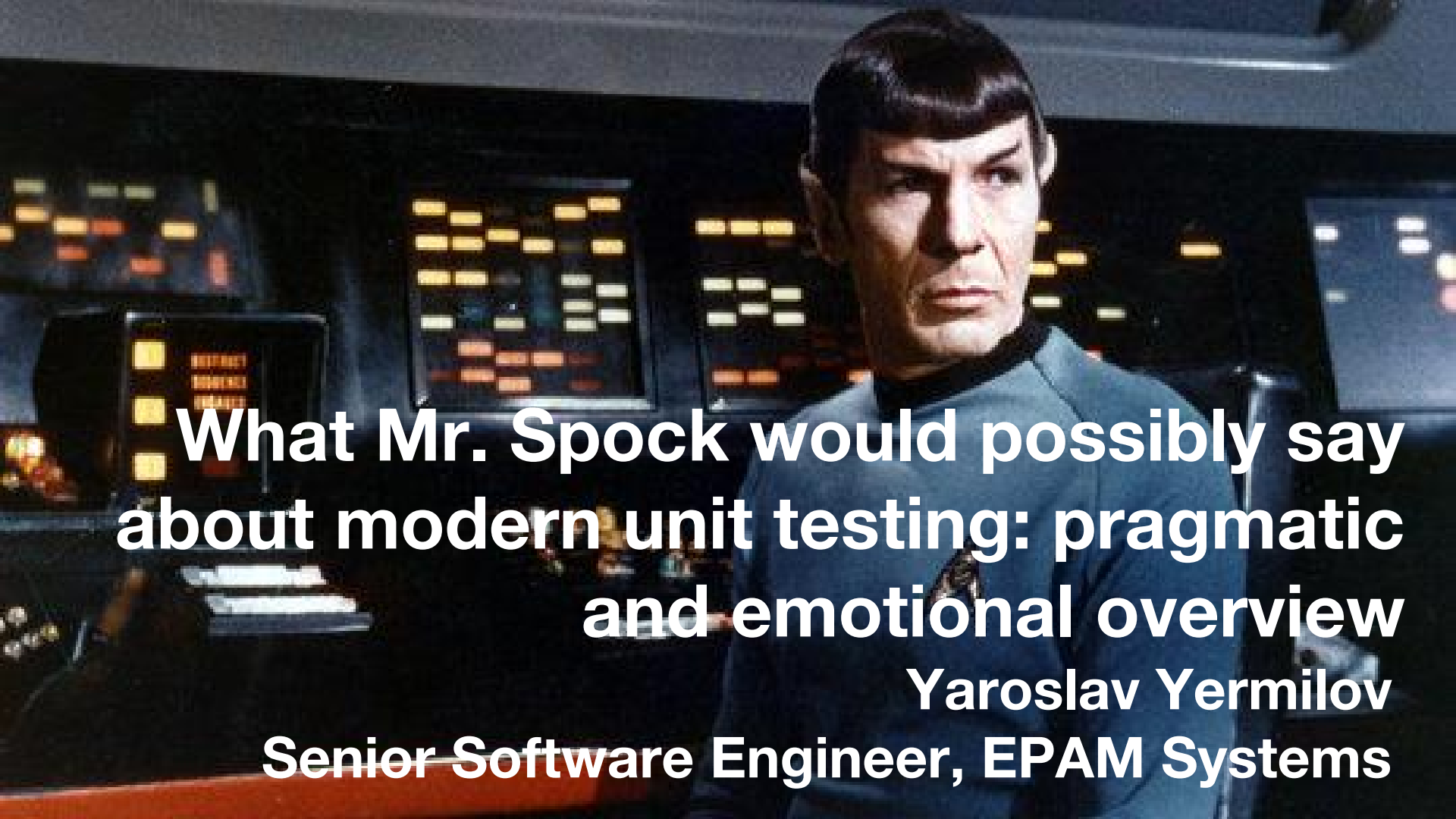


What Mr. Spock would possibly say about modern unit testing: pragmatic and emotional overview

Yaroslav Yermilov
Senior Software Engineer, EPAM Systems

About me

Yaroslav Yermilov

Senior Software Engineer,
postgraduate

https://ua.linkedin.com/pub/yaroslav-yermilov/58/682/506

https://www.facebook.com/yaroslav.yermilov

https://twitter.com/yermilov17

fond of Java, complete code, big data, data science, Groovy

About Spock

Spock

From Wikipedia, the free encyclopedia

This article is about the Star Trek character. For the pediatrician, see Benjamin Spock. For other uses, see Spock (disambiguation).

Spock, commonly **Mr. Spock** (sometimes popularly referred to as: **Spock, son of Sarek**), is a fictional character in the *Star Trek* media franchise. Spock was first portrayed by Leonard Nimoy in the original *Star Trek* series, and also appears in the animated *Star Trek* series, a two-part episode of *Star Trek: The Next Generation*, eight of the *Star Trek* feature films, and numerous Star Trek novels, comics, and video games.[1][2] In addition to this, numerous actors portrayed the various stages of Spock's rapid growth, due to the effects of the Genesis Planet, in the 1984 Star Trek film *Star Trek III: The Search for Spock*. In the 2009 film *Star Trek*, Nimoy reprised his role with Zachary Quinto, who depicted a younger version of the character, existing within an alternate timeline. Both reprised their roles in the 2013 sequel *Star Trek Into Darkness* and Quinto will reprise his role again in 2016 *Star Trek Beyond*.[2]

Spock serves aboard the starship *Enterprise*, as science officer and first officer, and later as commanding officer of two iterations of the vessel. Spock's mixed human-Vulcan heritage serves as an important plot element in many of the character's appearances. Along with Captain James T. Kirk and Dr. Leonard "Bones" McCoy, he is one of the three central characters in the original *Star Trek* series and its films. After retiring from Starfleet, Spock serves as a Federation ambassador, contributing toward the easing of the strained relationship between the Federation and the Klingon Empire. In his later years, he serves as Federation ambassador to the Romulan Star Empire and becomes involved in the ill-fated attempt to save Romulus from a supernova.[3]

Contents [hide]

Spock

Leonard Nimoy as Spock (left) beside William Shatner as James T. Kirk

Species	Half-Vulcan (paternal) Half-Human (maternal)
Affiliation	Starfleet Vulcan Government
Posting	USS *Enterprise* Second officer/Science officer Executive officer/Science officer commanding officer USS *Enterprise*-A Executive officer/Science officer Federation Ambassador-at-Large
Rank	Lieutenant commander Commander

Disclaimer

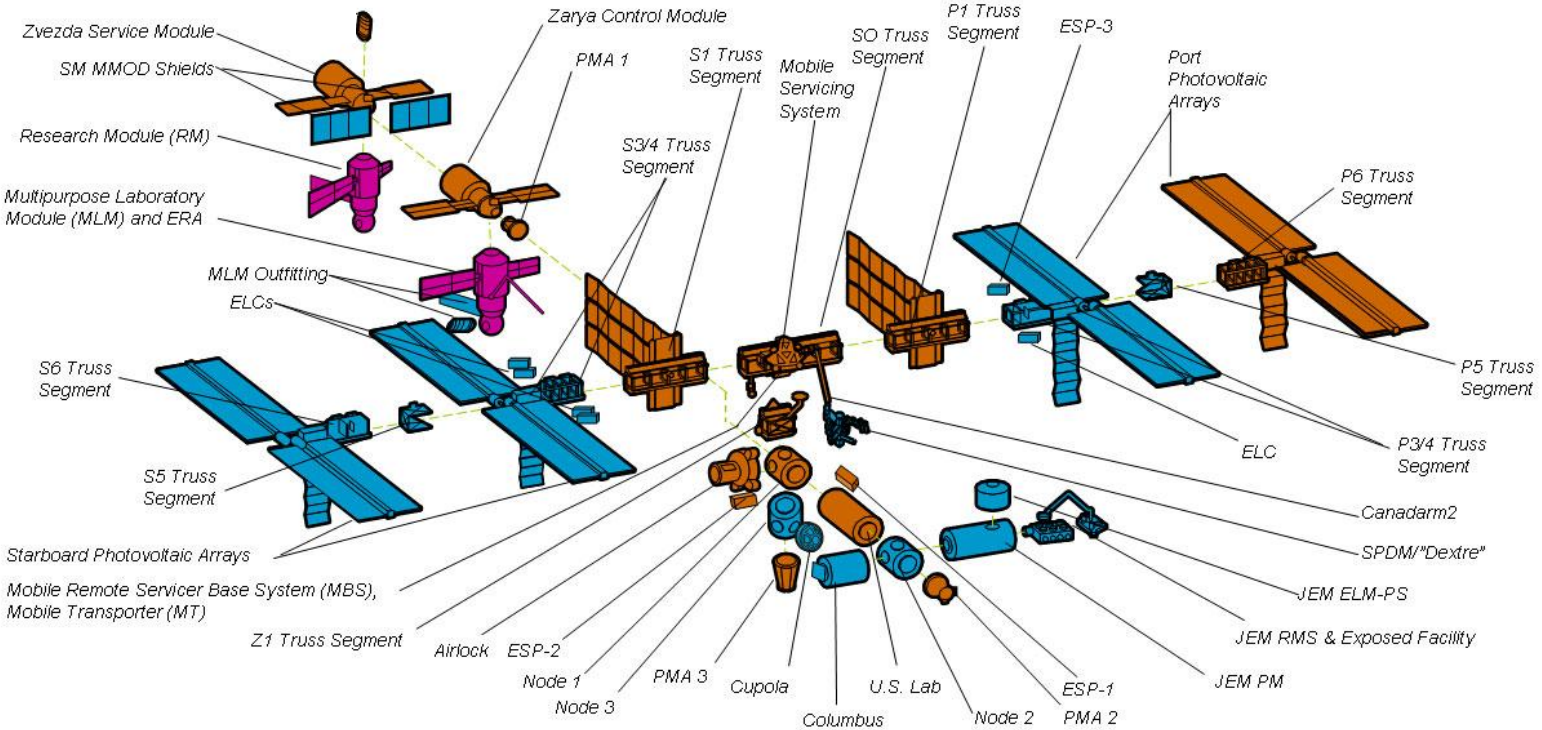
"May the Force
be With You"

-Dr Who


What’s the problem?


Better to test before going to production

Firstly, to unit-test

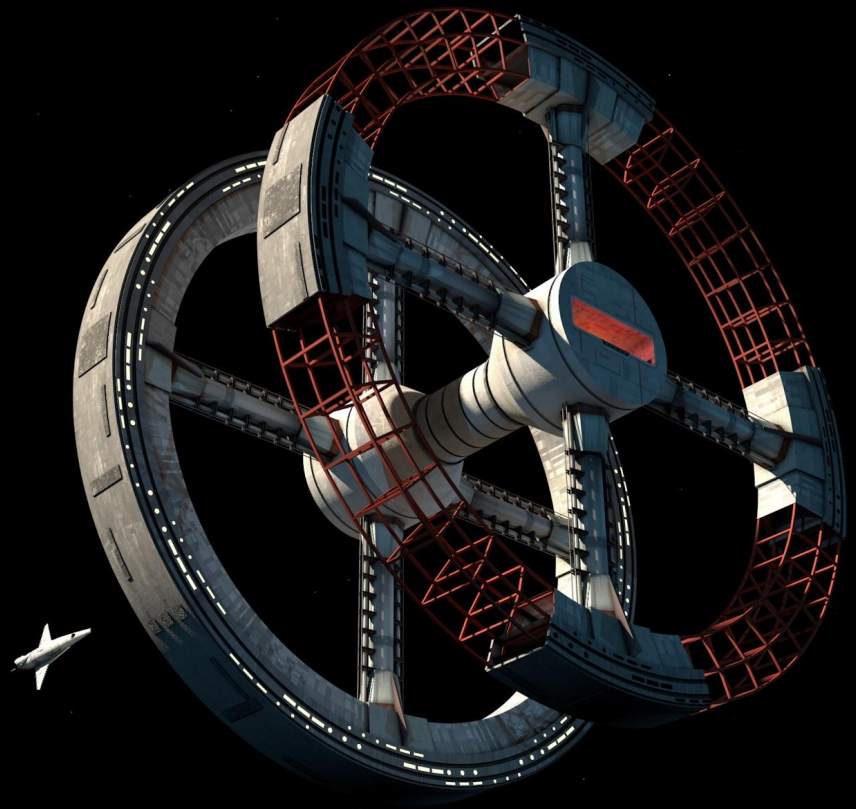


So what?

JUnit?

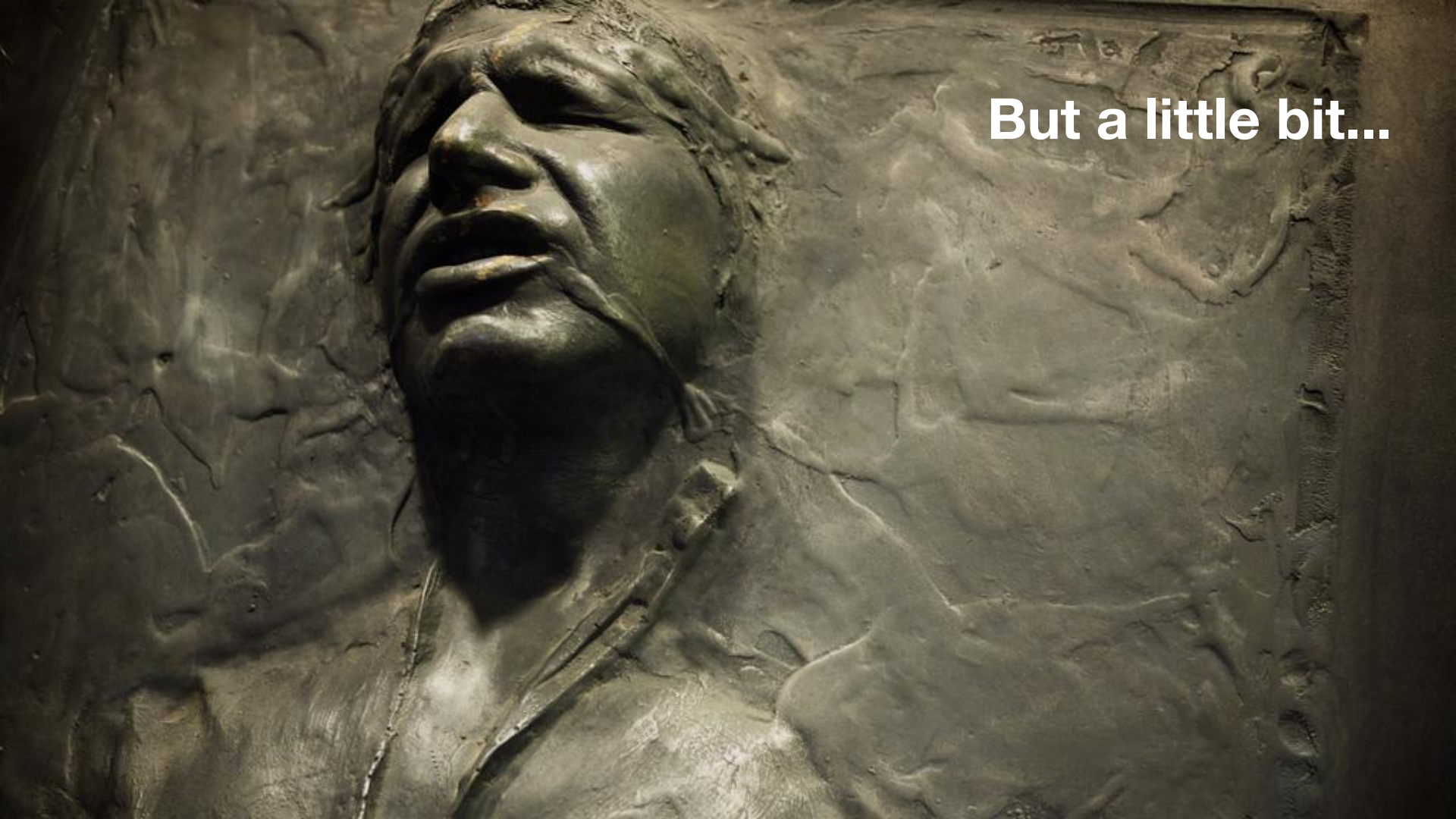
JUnit on steroids?

TestNG?

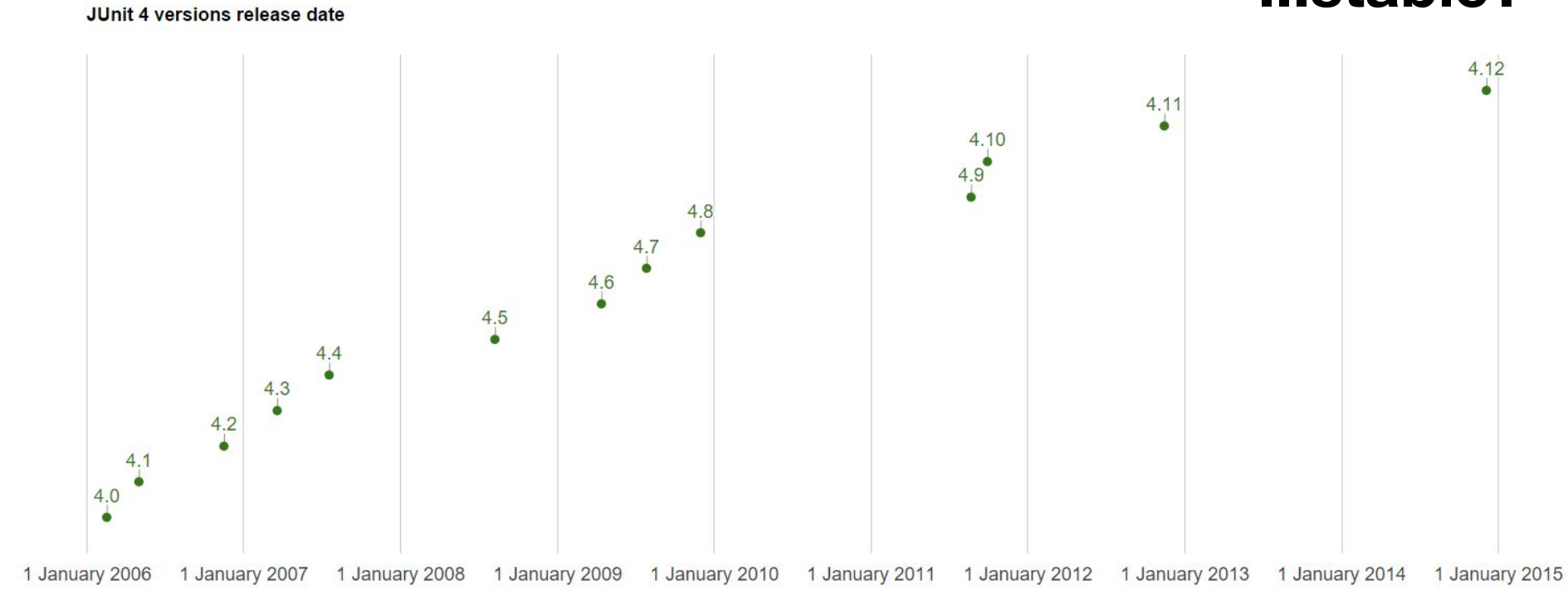


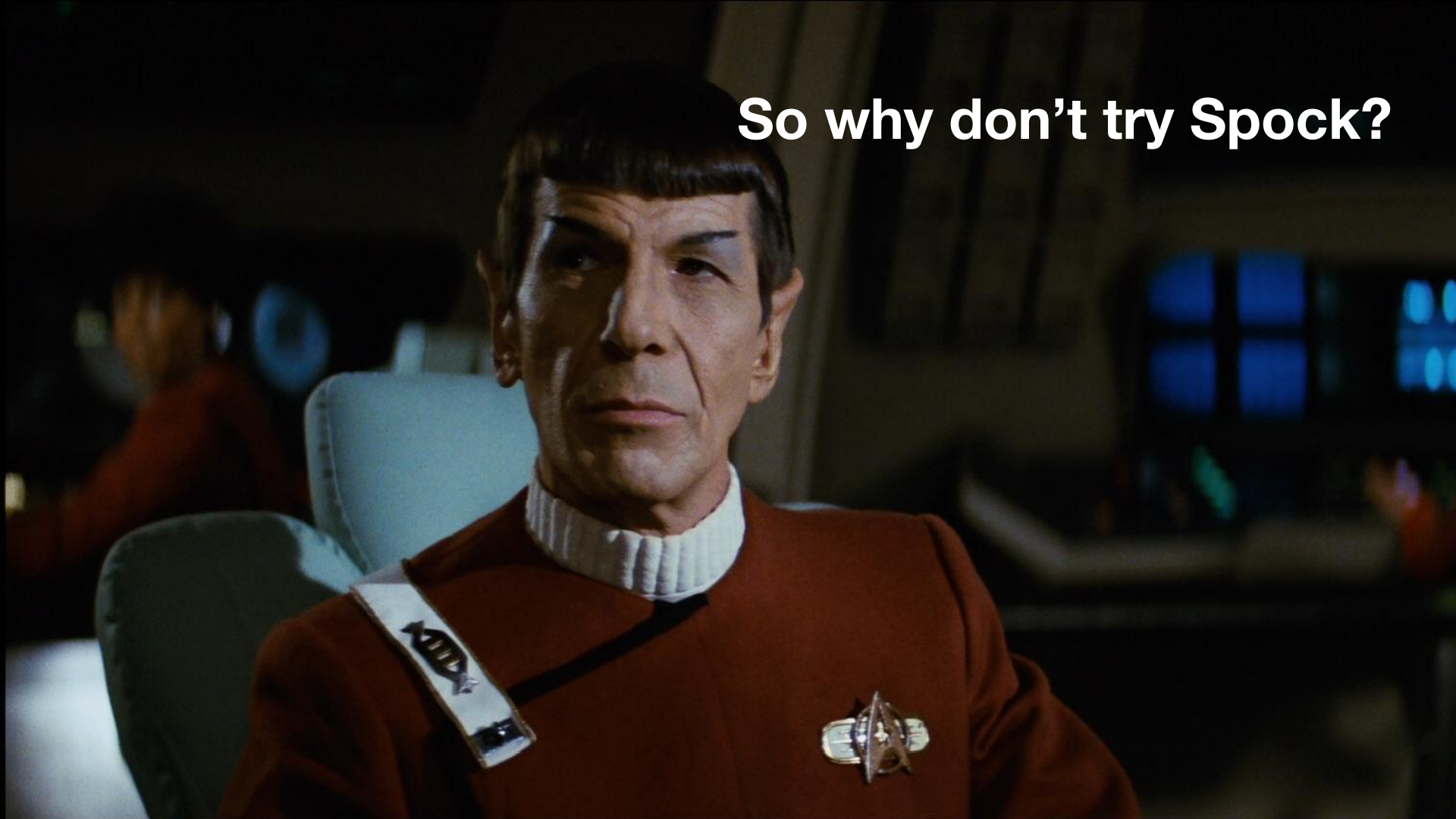
Nothing?

Well, JUnit is great


But a little bit...

...stable?

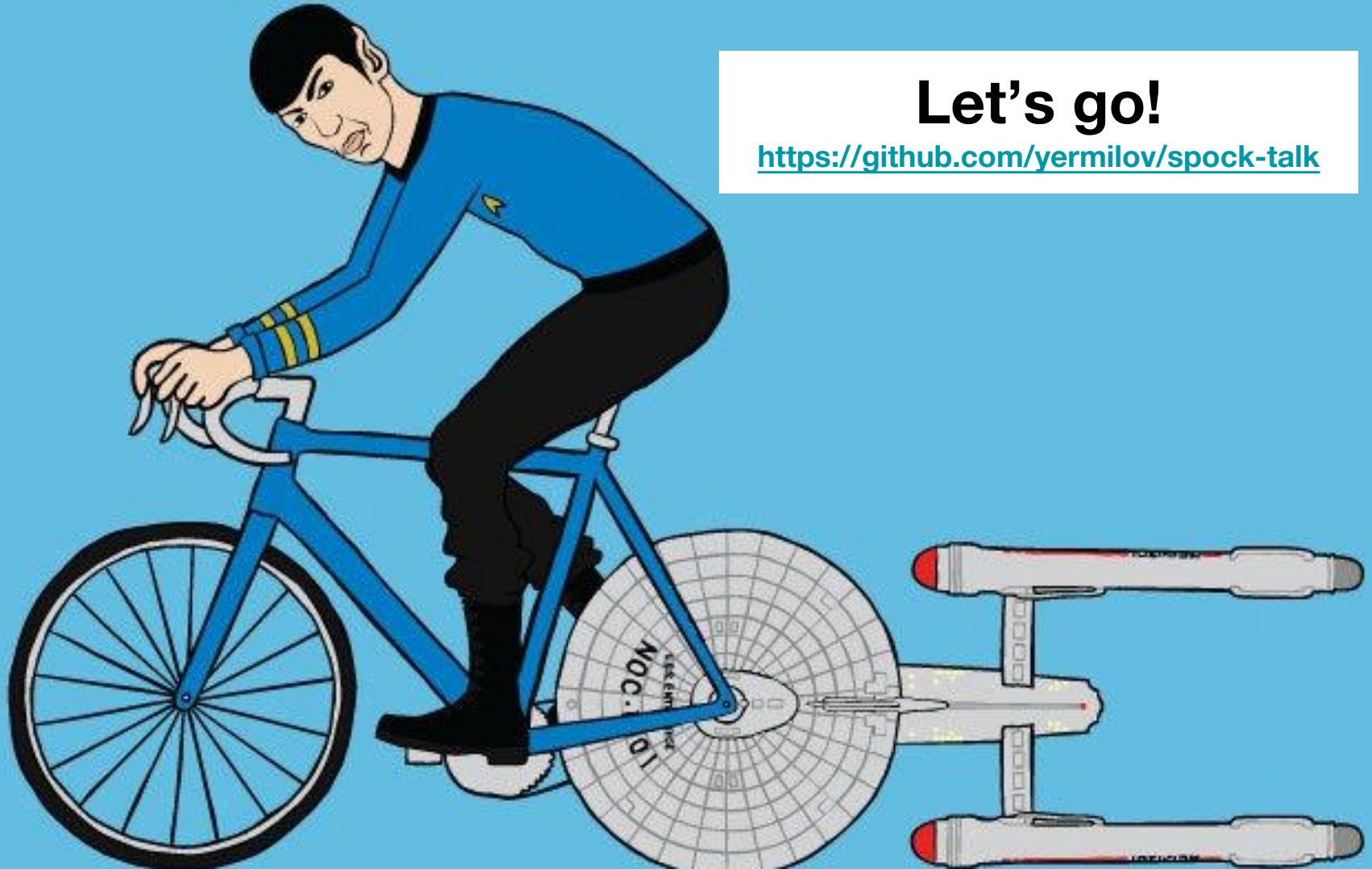

JUnit 4 versions release date
4.0
4.1
4.2
4.3
4.4
4.5
4.6
4.7
4.8
4.9
4.10
4.11
4.12
1 January 2006
1 January 2007
1 January 2008
1 January 2009
1 January 2010
1 January 2011
1 January 2012
1 January 2013
1 January 2014
1 January 2015


So why don’t try Spock?

Spock is Enterprise ready

Let’s go!

https://github.com/yermilov/spock-talk

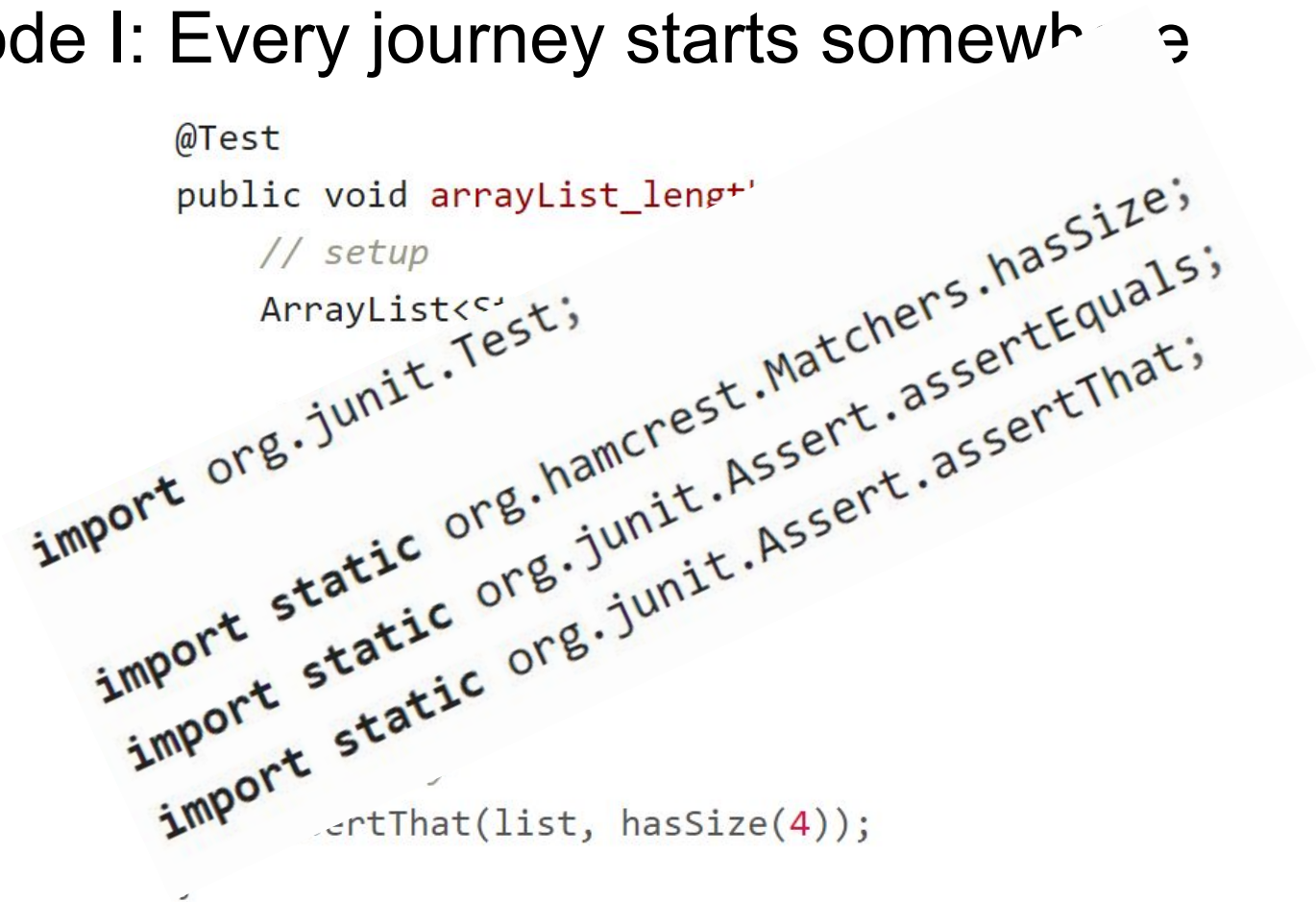



Episode I: Every journey starts somewhere

Episode I: Every journey starts somewhere

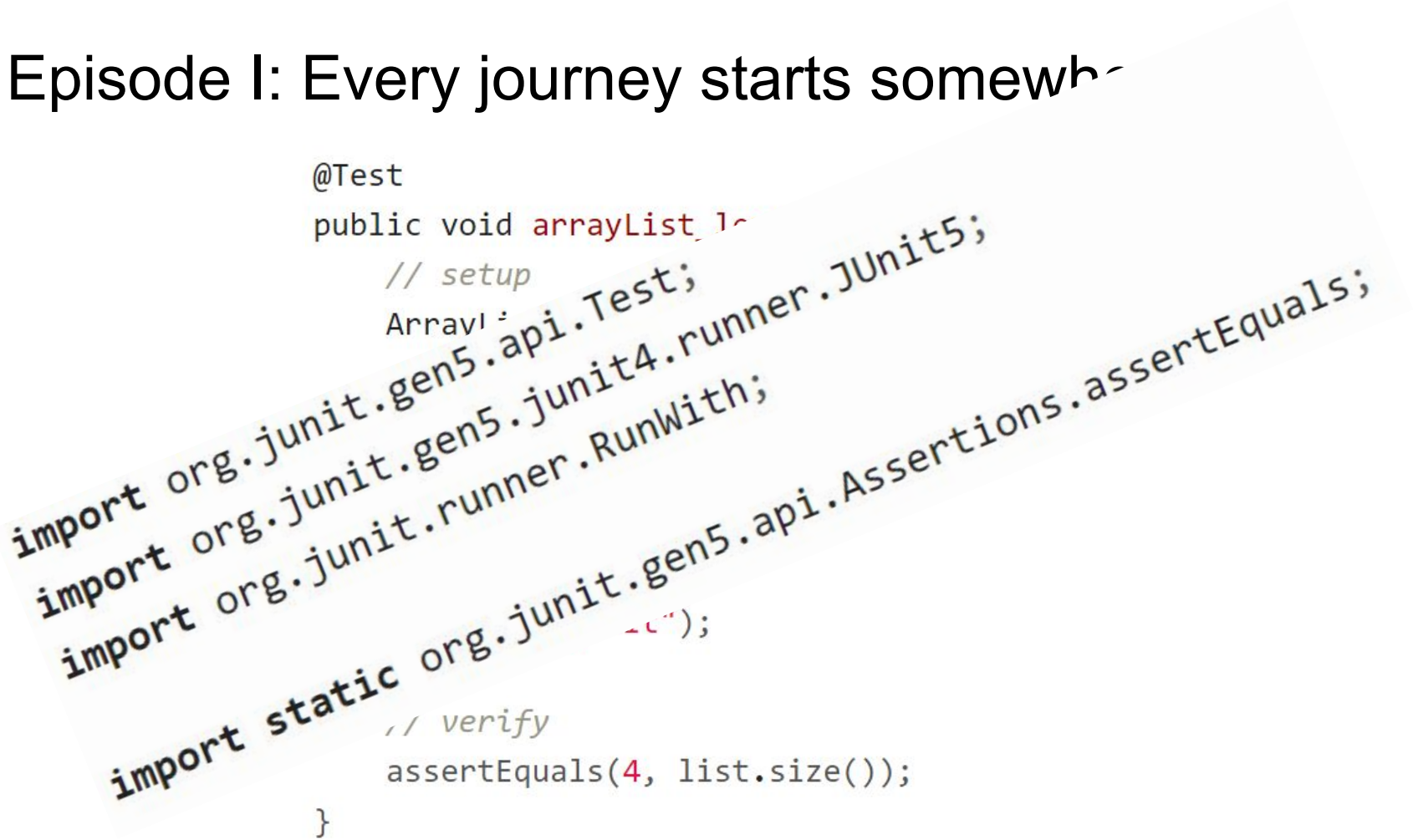
```
@Test
public void arrayList_length_ind() {
    ArrayList<String> list = new ArrayList<>();
    list.add("we");
    list.add("all");
    list.add("love");
    list.add("junit");
    assertEquals(list.size(), 4);
}
```

Episode I: Every journey starts somewhere



Episode I: Every journey starts somew

Episode I: Every journey starts somewh



Every journey starts somewhere

```
class N04S_EasyStart extends Specification {

    def 'ArrayList.size() test, but much spockier'() {
        setup:
        ArrayList<String> list = new ArrayList<>()

        when:
        list.add('we')
        list.add('will')
        list.add('love')
        list.add('spock')

        then:
        list.size() == 4
    }
```

Episode II: Assertions

Episode II: Assertions

```
then:
assertThat(list, hasSize(3));

java.lang.AssertionError:
Expected: a collection with size <3>
     but: collection size was <4>
```

```
then:
list.size() == 3

Condition not satisfied:

list.size() == 3
|      |      |
|      4      false
[we, will, love, spock]
```

Episode II: Assertions

```
then:
expect list, hasSize(3)

Condition not satisfied:

expect list, hasSize(3)
|      |
false  [we, will, love, spock]

Expected: a collection with size <3>
     but: collection size was <4>
```

Episode II: Assertions

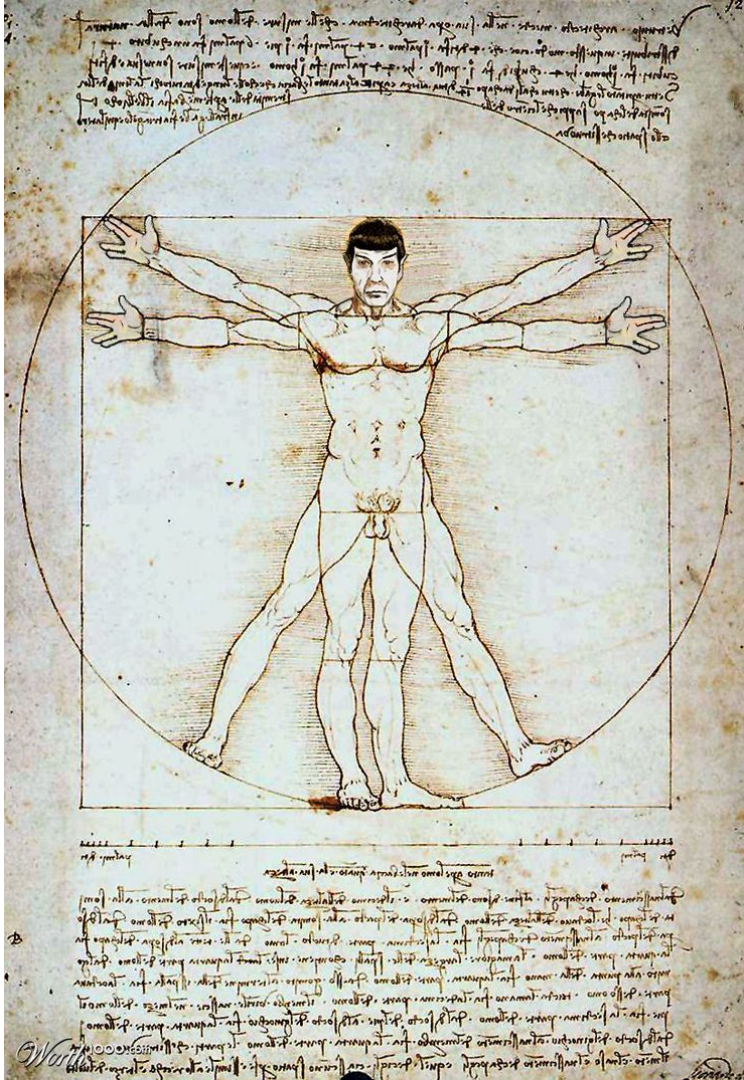
```
Condition not satisfied:

list.findAll({ it.length() < 5 }).groupBy({ it[0] }).find({ k, v -> v.size() > 1 }).value == list.findAll({ it.length() < 5 }).drop(1)
|    |                          |                  |                                    |     |  |    |                            |
|    [we, will, love]           |                  w=[we, will]                         |     |  |    [we, will, love]             [will, love]
[we, will, love, spock]          [w:[we, will], l:[love]]                              |     |  [we, will, love, spock]
                                                                                       |     false
                                                                                       [we, will]
```

Episode II: Assertions

```
@Test
void 'ArrayList.size()'() {
    // setup
    ArrayList<String> lis
    assumeThat(list, 

    // run
    lis

            ist.findAll({ it.length() < 5 }) == list.drop(2)
}
```

```
Assertion failed:

assert list.findAll({ it.length() < 5 }) == list.drop(2)
       |    |                             |  |    |
       |    [we, will, love]              |  |    [love, spock]
       [we, will, love, spock]            |  [we, will, love, spock]
                                          false
```

Episode III: Idiomatic Spock



Episode III: Idiomatic Spock

```groovy
@Title('ArrayList tests')
@Narrative('''
As Java developer
(that trust nothing)
I want to be sure ArrayList works
''')
class N07S_IdiomaticSpock extends Specification {

    @Subject
    ArrayList<String> list

    @Issue('https://github.com/yermilov/spock-talk/issues/1')
    def 'ArrayList.size()'() {
        setup: 'new ArrayList instance'
        list = new ArrayList<>()

        expect: 'that newly created ArrayList instance is empty'
        list.empty

        when: 'add value to list'
        list.add 'we'

        and: 'add one more value to list'
        list.add 'will'

        then: 'array list size should be 2'
        list.size() == 2
```

Episode III: Idiomatic Spock

ArrayList tests
As Java developer
(that trust nothing)
I want to be sure ArrayList works

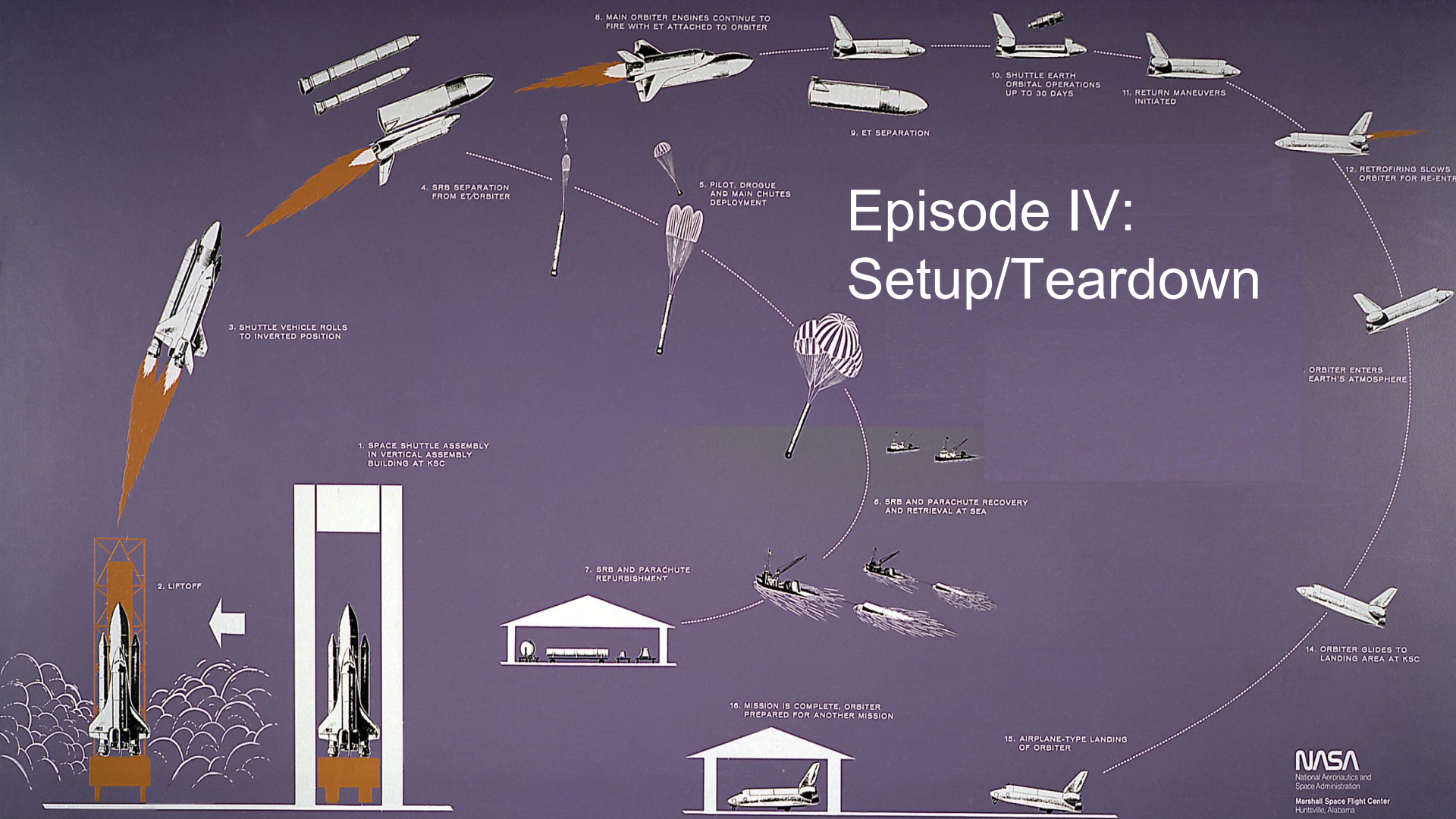
Features:

- ArrayList.size()

ArrayList.size()
Issues:

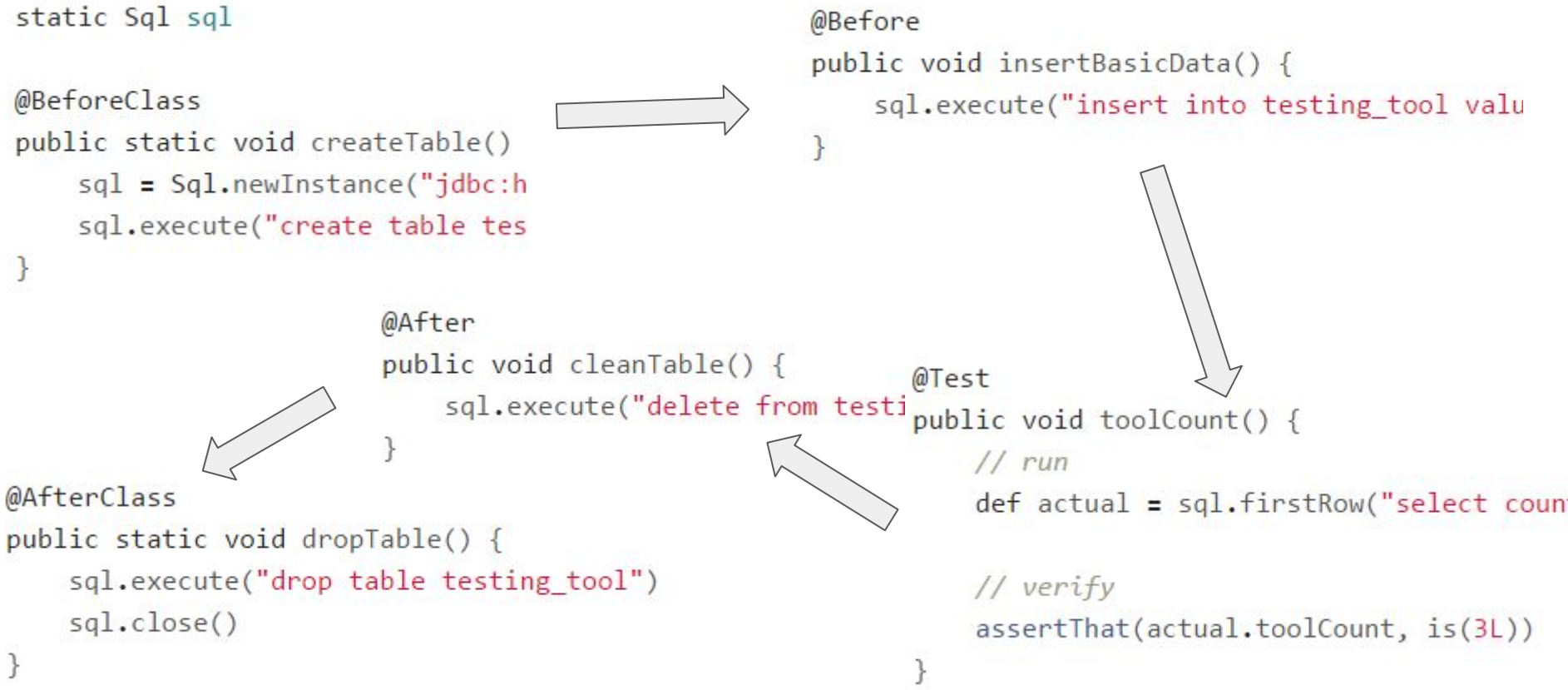
- https://github.com/yermilov/spock-talk/issues/1

Given:	new ArrayList instance
Expect:	that newly created ArrayList instance is empty
When:	add value to list
And:	add one more value to list
Then:	array list size should be 2
When:	add two more values into list

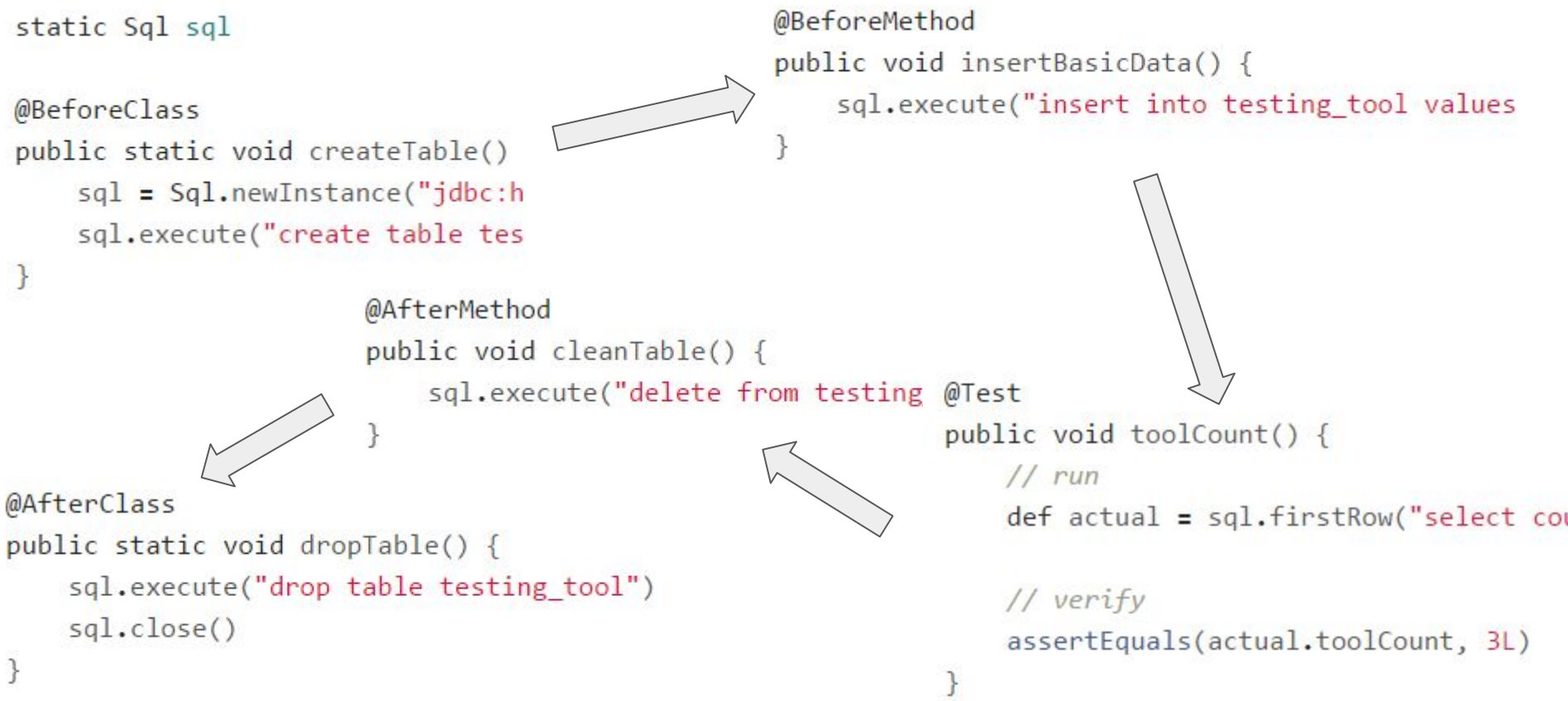


Episode IV: Setup/Teardown

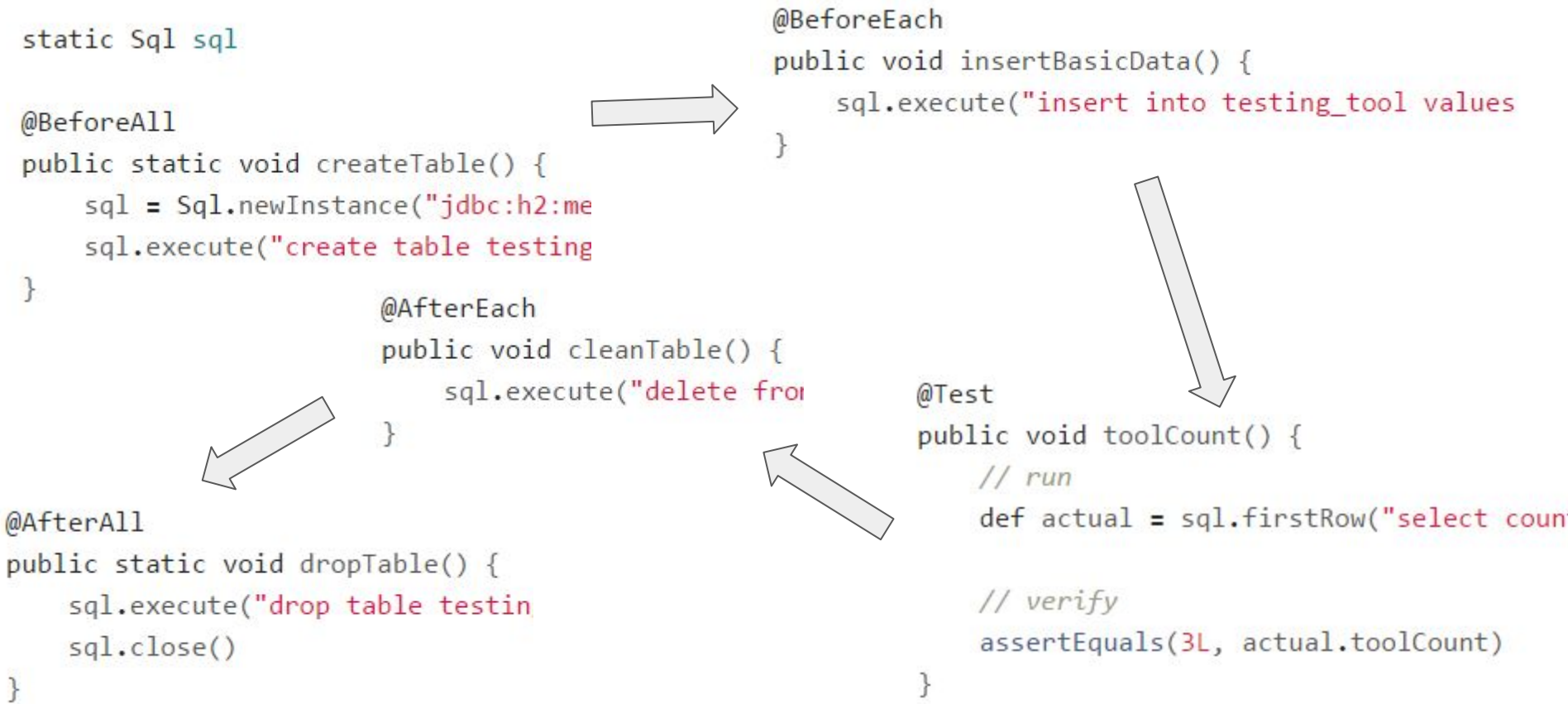
Episode IV: Setup/Teardown



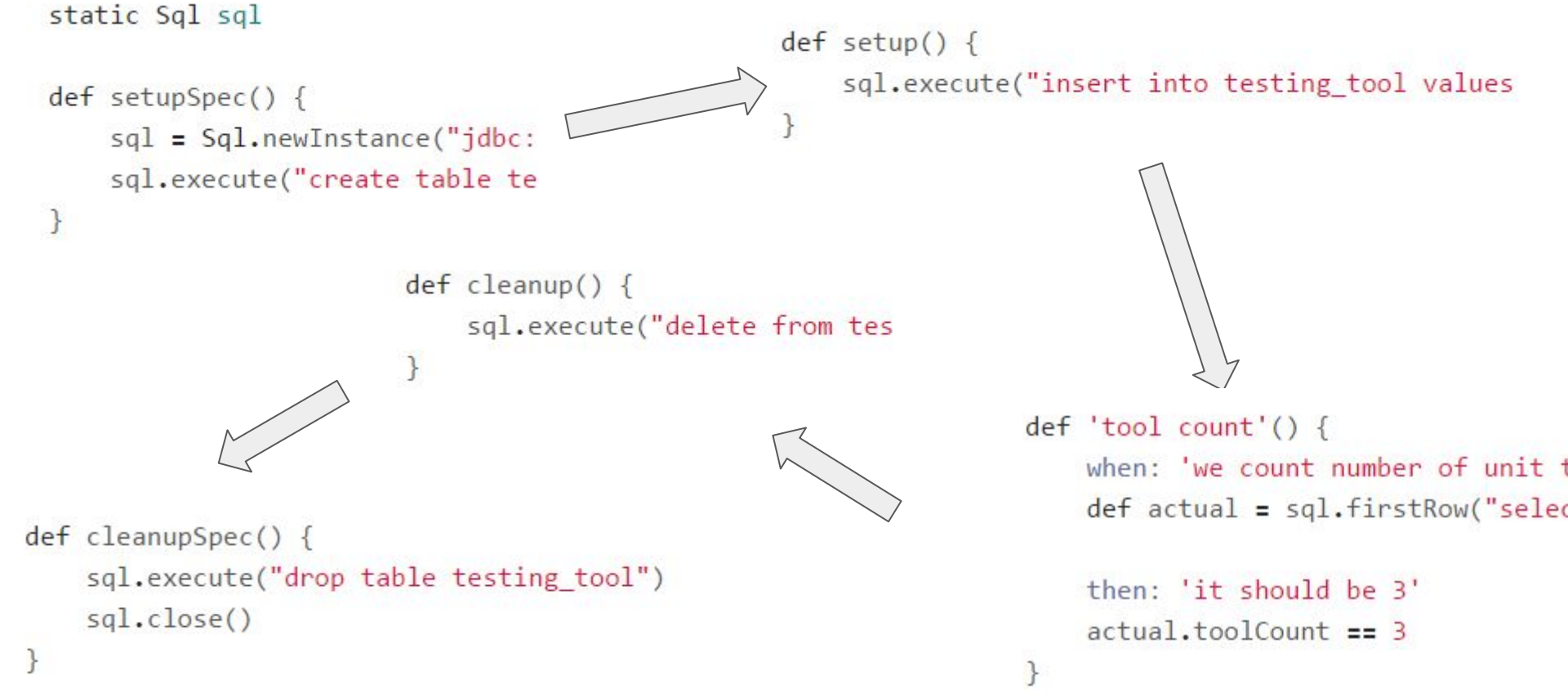
Episode IV: Setup/Teardown



Episode IV: Setup/Teardown



Episode IV: Setup/Teardown

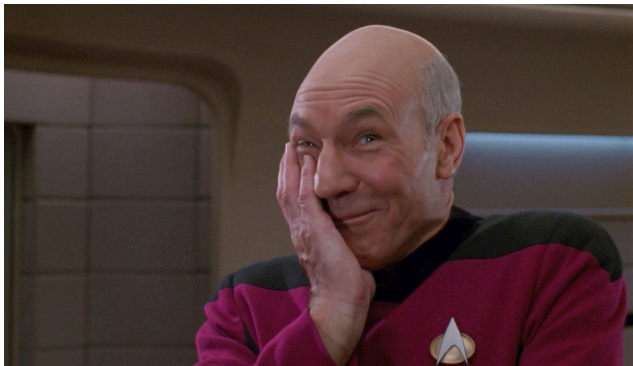


Episode IV: Setup/Teardown

```
@Shared @AutoCleanup sql

def 'JUnit5 is in game!'() {
    setup: 'add JUnit 5 to the list of unit testing tools'
    sql.execute("insert into testing_tool values (4, 'junit'

    when: 'we count number of JUnits'
    def actual = sql.firstRow("select count(*) as toolCount

    then: 'it should be 2'
    actual.toolCount == 2

    cleanup: 'remove JUnit 5 from the list of unit testing t
    sql.execute("delete from testing_tool where id = 4")
}
```

Episode IV: Setup/Teardown

```
        // run
        def actual = sql.firstRow("select count(*) as too

        // verify
        actual.toolCount == 2
    } finally {
        try {
            // cleanup
            sql.execute("delete from testing_tool where i
        } finally {
            sql.close()
        }
    }
```



Episode IV: @Shared or static?

Episode V:
Data driven test

Episode V: Data driven test

```
def 'is coin flip good enough for determine if integer is prime'() {
    when: 'we flip a coin'
    boolean coinFlip = new Random().nextBoolean()

    then: 'it will be great if coin flip predict if number is prime'
    coinFlip == Primes.isPrime(number)

    where: 'data is random'
    number << (1..28).collect({ new Random().nextInt(1000) }).findAll({ it >= 2 }).sort()
}
```

Episode V: Data driven test

is coin flip good enough for determine if integer is prime — Return

When: we flip a coin

Then: it will be great if coin flip predict if number is prime

Where: data is random

Examples:

number	
353	FAIL
361	OK
451	OK
617	OK
706	FAIL

3/5 passed

The following problems occurred:

- [353]
  - Condition not satisfied:

```
coinFlip == Primes.isPrime(number)
|        |         |       |
false    false     true    353
```

- [706]
  - Condition not satisfied:

```
coinFlip == Primes.isPrime(number)
```

Episode V: Data driven test

```
@Unroll
def 'calculate runner speed and location after some time for #description'() {
    expect: 'that runner speed is equal to expected'
    initialSpeed + acceleration * time == expectedSpeed

    and: 'runner location is equal to expected'
    initialLocation + time * (initialSpeed + acceleration / 2 * time) == expectedLocation

    where: 'there are set of precalculated data for different situations'
    initialLocation | initialSpeed | acceleration | time || expectedLocation | expectedSpeed || description
    0               | 6            | 0            | 10   || 60               | 6             || 'steady run from starting point'
    5               | 0            | 3            | 3    || 17               | 9             || 'starting from standing with acceleration'
    -50             | 10           | -1           | 10   || 0                | 0             || 'constant deceleration'
}
```

Episode V: Data driven test

calculate runner speed and location after some time for steady run from starting point Return

Expect: that runner speed is equal to expected

And: runner location is equal to expected

Where: there are set of precalculated data for different situations

calculate runner speed and location after some time for starting from standing with acceleration Return

Expect: that runner speed is equal to expected

And: runner location is equal to expected

Where: there are set of precalculated data for different situations

The following problems occurred:

- Condition not satisfied:

```
initialLocation + time * (initialSpeed + acceleration / 2 * time) == expectedLocation
|               | |    |  |            | |            |   | |     |  |
5               | 3    |  0            | 3            1.5 | 3     |  17
                18.5   13.5            4.5                 4.5     false
```

calculate runner speed and location after some time for constant deceleration Return

Expect: that runner speed is equal to expected

And: runner location is equal to expected

Where: there are set of precalculated data for different situations

Episode V: Data driven test

```
@RunWith(Parameterized.class)
public class N16J_DataTables {

    @Parameterized.Parameters(name = "calculate runner speed and location after
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {
            { 0.0,   6.0,  0.0,  10.0, 60.0, 6.0, "steady run from starting poin
            { 5.0,   0.0,  3.0,  3.0,  17.0, 9.0, "starting from standing with a
            { -50.0, 10.0, -1.0, 10.0, 0.0,  0.0, "constant deceleration"}
        });
    }

    @Parameterized.Parameter(value = 0)
    public double initialLocation;

    @Parameterized.Parameter(value = 1)
    public double initialSpeed;

    @Parameterized.Parameter(value = 2)
    public double acceleration;
```

Episode V: Data driven test

Test	Duration	Result
location[calculate runner speed and location after some time for constant deceleration]	0s	passed
location[calculate runner speed and location after some time for starting from standing with acceleration]	0s	failed
location[calculate runner speed and location after some time for steady run from starting point]	0s	passed
speed[calculate runner speed and location after some time for constant deceleration]	0s	passed
speed[calculate runner speed and location after some time for starting from standing with acceleration]	0s	passed
speed[calculate runner speed and location after some time for steady run from starting point]	0s	passed

Episode V: Data driven test

```
@DataProvider(name = "data")
public static Object[][] data() {
    return new Object[][] {
        { 0.0,   6.0,  0.0,  10.0, 60.0, 6.0 },
        { 5.0,   0.0,  3.0,  3.0,  17.0, 9.0 },
        { -50.0, 10.0, -1.0, 10.0, 0.0,  0.0 }
    };
}

@Test(dataProvider = "data")
public void speed(double initialLocation, double initialSpeed, double
    double speed = initialSpeed + acceleration * time;
    assertEquals(speed, expectedSpeed);
}
```

Episode V: Data driven test

Test	Duration	Result
location[0](0.0, 6.0, 0.0, 10.0, 60.0, 6.0)	0s	passed
location[1](5.0, 0.0, 3.0, 3.0, 17.0, 9.0)	0.001s	failed
location[2](-50.0, 10.0, -1.0, 10.0, 0.0, 0.0)	0s	passed
speed[0](0.0, 6.0, 0.0, 10.0, 60.0, 6.0)	0s	passed
speed[1](5.0, 0.0, 3.0, 3.0, 17.0, 9.0)	0s	passed
speed[2](-50.0, 10.0, -1.0, 10.0, 0.0, 0.0)	0s	passed

Episode VI: Exceptions

Episode VI: Exceptions

```
def 'empty ArrayList has no 17th element'() {
    given: 'empty array list'
    def arrayList = new ArrayList<Integer>()

    when: 'we try to retrieve element with index #17'
    arrayList.get(17)

    then: 'exception is thrown with expected message'
    IndexOutOfBoundsException exception = thrown()
    exception.message == 'Index: 17, Size: 0'
}
```

Episode VI: Exceptions

```
def 'large ArrayList has 17th element'() {
    given: 'list of 28 prime numbers'
    def arrayList = new ArrayList<Integer>()
    28.times { arrayList << Primes.nextPrime(arrayList.empty ? 1 : arrayList[-1]) }

    when: 'we try to retrieve element with index #17'
    arrayList.get(17)

    then: 'no exception is thrown'
    notThrown(IndexOutOfBoundsException)
}
```

Episode VI: Exceptions

```
@Test(expected = IndexOutOfBoundsException.class)
public void exception_oldWay() {
    // setup
    ArrayList<Integer> arrayList = new ArrayList<>();

    // run
    arrayList.get(17);
}
```

Episode VI: Exceptions

```
@Test
public void noException_oldWay() {
    // setup
    ArrayList<Integer> arrayList = new ArrayList<>();

    // run
    arrayList.size();

    // verify no exception is thrown
}
```

Episode VI: Exceptions

```java
@Test
public void noException_uglyWay() {
    // setup
    ArrayList<Integer> arrayList = new ArrayList<>();

    // run
    try {
        arrayList.size();
    } catch (Exception exception) {
        // verify no exception is thrown
        fail("Expected no exception to be thrown");
    }
}
```

Episode VI: Exceptions

```java
@Test
public void exceptionAndMessage_oldWay() {
    // setup
    ArrayList<Integer> arrayList = new ArrayList<>();

    // run
    try {
        arrayList.get(17);
        fail("Expected an IndexOutOfBoundsException to be thrown");
    } catch (IndexOutOfBoundsException exception) {
        // verify
        assertThat(exception.getMessage(), is("Index: 17, Size: 0"));
    }
}
```

Episode VI: Exceptions

```
@Rule
public ExpectedException thrown = ExpectedException.none();

@Test
public void exception_modernWay() {
    // setup
    ArrayList<Integer> arrayList = new ArrayList<>();

    // expect
    thrown.expect(IndexOutOfBoundsException.class);
    thrown.expectMessage(is("Index: 17, Size: 0"));

    // run
    arrayList.get(17);
}
```

Episode VI: Exceptions

```
@Test(expectedExceptions = IndexOutOfBoundsException.class, expectedExceptionsMessageRegExp = "Index: 17, Size: 0")
public void exception() {
    // setup
    ArrayList<Integer> arrayList = new ArrayList<>();

    // run
    arrayList.get(17);
}
```

Episode VI: Exceptions

```
@Test
public void exception() {
    // setup
    ArrayList<Integer> arrayList = new ArrayList<>();

    // run & verify
    Throwable thrown = expectThrows(IndexOutOfBoundsException.class, () -> { arrayList.get(17); });
    assertEquals("Index: 17, Size: 0", thrown.getMessage());
}
```


Episode VII:
Mocks,
Stubs,
Spies

Episode VII: Mocks, Stubs, Spies

```
Random random = Stub()
random.nextInt(26) >> 0
```

```
Random random = mock(Random.class);
doReturn(0).when(random).nextInt(26);
```

```
Random random = Stub()
random.nextInt(_) >>> [ 0, 1, 2, 3, 4 ]
```

```
Random random = mock(Random.class);
doReturn(0).doReturn(1).doReturn(2).doReturn(3).doReturn(4).when(random).nextInt(26);
```

Episode VII: Mocks, Stubs, Spies

```
Random random = Stub()
random.nextInt(_) >> { throw new RuntimeException() }
```

```
Random random = mock(Random.class);
doThrow(new RuntimeException()).when(random).nextInt(26);
```

```
Random random = Stub()
random.nextInt(_) >> { int max -> max - 1 }
```

```
Random random = mock(Random.class);
doAnswer(inv -> (int) inv.getArguments()[0] - 1).when(random).nextInt(26);
```

Episode VII: Mocks, Stubs, Spies

```
5 * random.nextInt(_)
```

```
verify(random, times(5)).nextInt(anyInt());
```

```
then: 'first random generator
1 * random.nextInt(10)
```

```
then: 'then random generator
(8.._) * random.nextInt(_)
```

```
then: 'random generator is ne
0 * random._
```

```
InOrder inOrder = inOrder(random);
inOrder.verify(random, times(1)).nextInt(10);
inOrder.verify(random, atLeast(8)).nextInt(26);
inOrder.verifyNoMoreInteractions();
```

Episode VII: Mocks, Stubs, Spies

```
1 * passwordGenerator.generate(!null)
```

```
1 * passwordGenerator.generate(_ as Integer)
```

```
1 * passwordGenerator.generate({ it >= 8 && it < 18 })
```

```
ArgumentCaptor<Integer> argument = ArgumentCaptor.forClass(Integer.class);
verify(passwordGenerator).generate(argument.capture());
assertThat(argument.getValue(), is(greaterThanOrEqualTo(8)));
assertThat(argument.getValue(), is(lessThan(18)));
```

Episode VII: Mocks, Stubs, Spies

```
def 'something terrifying'() {
    when: 'mocking goes to far'
    Mock(Random)

    then: '''that guy who ends up maintaining
             your code will be a violent psychopath
             who knows where you live'''
    (_..._) * _.(_) >> _
}
```


Episode VIII: Time travel

Episode VIII: Time travel

```
sql.firstRow("select count(*) as toolCount from testing_tool").toolCount ==
        old(sql.firstRow("select count(*) as toolCount from testing_tool").toolCount) + 1
```

```
// run
def old = sql.firstRow("select count(*) as toolCount from testing_tool").toolCount
sql.execute("insert into testing_tool values (4, 'junit', '5')")
def actual = sql.firstRow("select count(*) as toolCount from testing_tool").toolCount

// verify
assertThat(actual, is(old + 1))
```

Episode VIII: Time travel

```
@Timeout(value = 2, unit = TimeUnit.SECONDS)
def 'infinite loop'() {
    setup: 'array list'
    def arrayList = new ArrayList<String>()

    expect: 'we will add to it values forever'
    while (true) { arrayList.add('spock forever!') }
}
```

Episode VIII: Time travel

```
@Test(timeout = 2000)
public void infiniteLoop() {
    // setup
    ArrayList<String> arrayList = new ArrayList<>();

    // run
    while (true) { arrayList.add("junit forever!"); }
}
```

Episode VIII: Time travel

```
@Test(timeOut = 2000)
public void infiniteLoop() {
    // setup
    ArrayList<String> arrayList = new ArrayList<>();

    // run
    while (true) { arrayList.add("testng forever!"); }
}
```

Episode VIII: Time travel

```
def conditions = new PollingConditions(timeout: 30)

def 'find next prime after 1728 eventually'() {
    setup: 'holder for answer'
    Integer actual = null

    when: 'async start calculating next prime after 1728'
    asyncNextPrime( 1728, { answer -> actual = answer } )

    then: 'eventually answer will be found'
    conditions.eventually {
        assert actual == 1733
    }
}
```

Episode VIII: Time travel

```
def conditions = new PollingConditions(timeout: 30)

def 'find next prime after 2817 within 3 seconds'() {
    setup: 'holder for answer'
    Integer actual = null

    when: 'async start calculating next prime after 1728'
    asyncNextPrime( 2817, { answer -> actual = answer } )

    then: 'within 3 seconds answer will be found'
    conditions.within(3) {
        assert actual == 2819
    }
}
```

Episode VIII: Time travel

```
def conditions = new AsyncConditions()

def 'find next prime after 2817 within 3 seconds'() {
    when: 'async start calculating next prime after 1728'
    asyncNextPrime( 1728, { answer ->
        conditions.evaluate { assert answer == 1733 }
    } )

    then: 'within 3 seconds answer will be found'
    conditions.await(3)
}
```

Episode VIII: Time travel

```
@Test(timeout = 10000)
public void findNextPrimeEventually() throws ExecutionException, Interrupte
    // run
    Future<Integer> actual = asyncNextPrime(1728);

    // verify
    while (!actual.isDone()) {
        if (actual.isDone()) {
            assertThat(actual.get(), is(1733));
        }
    }
}
```


Episode IX:
Test control

Episode IX: Test control

```
@Ignore('will fix it before commit')
def 'this test is always ignored'() {
    // TODO FIXME test is failing
    expect: 'that 2+2=5'
    2 + 2 == 5
}
```

Episode IX: Test control

```
@IgnoreIf({ os.windows || sys['pretend.os'] == 'windows' })
def 'this test is ignored on Windows'() {
    expect: 'that we are not on Windows'
    !System.properties['os.name'].toString().toLowerCase().contains('windows')
}
```

Episode IX: Test control

```
@Requires({ jvm.java8 && env['JAVA_HOME'] != null })
def 'this test requires JAVA_HOME set and Java 8 installed'() {
    expect: 'that we are on Java 8'
    'java -version'.execute().errorStream.text.contains('java version "1.8.0_73"')
}
```

Episode IX: Test control

```
@IgnoreIf({ new Random().nextBoolean() })
def 'this test is SOMETIMES ignored'() {
    when: 'i hate my job'
    Integer.metaClass.plus = { Integer other ->
        return 5
    }

    then: 'i can make them pay'
    2 + 2 == 5
}
```

Episode IX: Test control

```
@IgnoreRest
def 'this test makes all other test ignored'() {
    expect: 'a miracle'
    2 + 2 == 5
}
```

Episode IX: Test control

```
@Requires({ N40S_ConditionalRuns_Part2.isGoogleSearchAvaiable() })
def 'this test runs only if Google Search is avaiable'() {
    setup: 'http connection service'
    def http = new HTTPBuilder('https://google.com')

    when: 'we search for the best java unit testing framework'
    def response = http.get(path : '/search', query : [q:'best java unit testing framework'])

    then: 'answer mentions spock'
    response.toString().toLowerCase().contains 'spock'
}
```

Episode IX: Test control

```
@Ignore("will fix it before commit")
@Test
public void alwaysIgnored() {
    // TODO FIXME test is failing
    assertThat(2+2, is(5));
}

@Test
public void ignoredOnWindows() {
    assumeThat(System.getProperty("os.name").toLowerCase(), is(not(containsString("windows"))));
```

Episode IX: Test control

```
@Test(enabled = false)
public void alwaysIgnored() {
    // TODO FIXME test is failing
    assertEquals(2+2, 5);
}
```

Episode IX: Test control

```
@Disabled("will fix it before commit")
@Test
public void alwaysIgnored() {
    // TODO FIXME test is failing
    assertEquals(5, 2+2);
}

@Test
public void ignoredOnWindows() {
    assumeTrue(System.getProperty("os.name").toLowerCase().contains("windows"));

    // ...
}
```

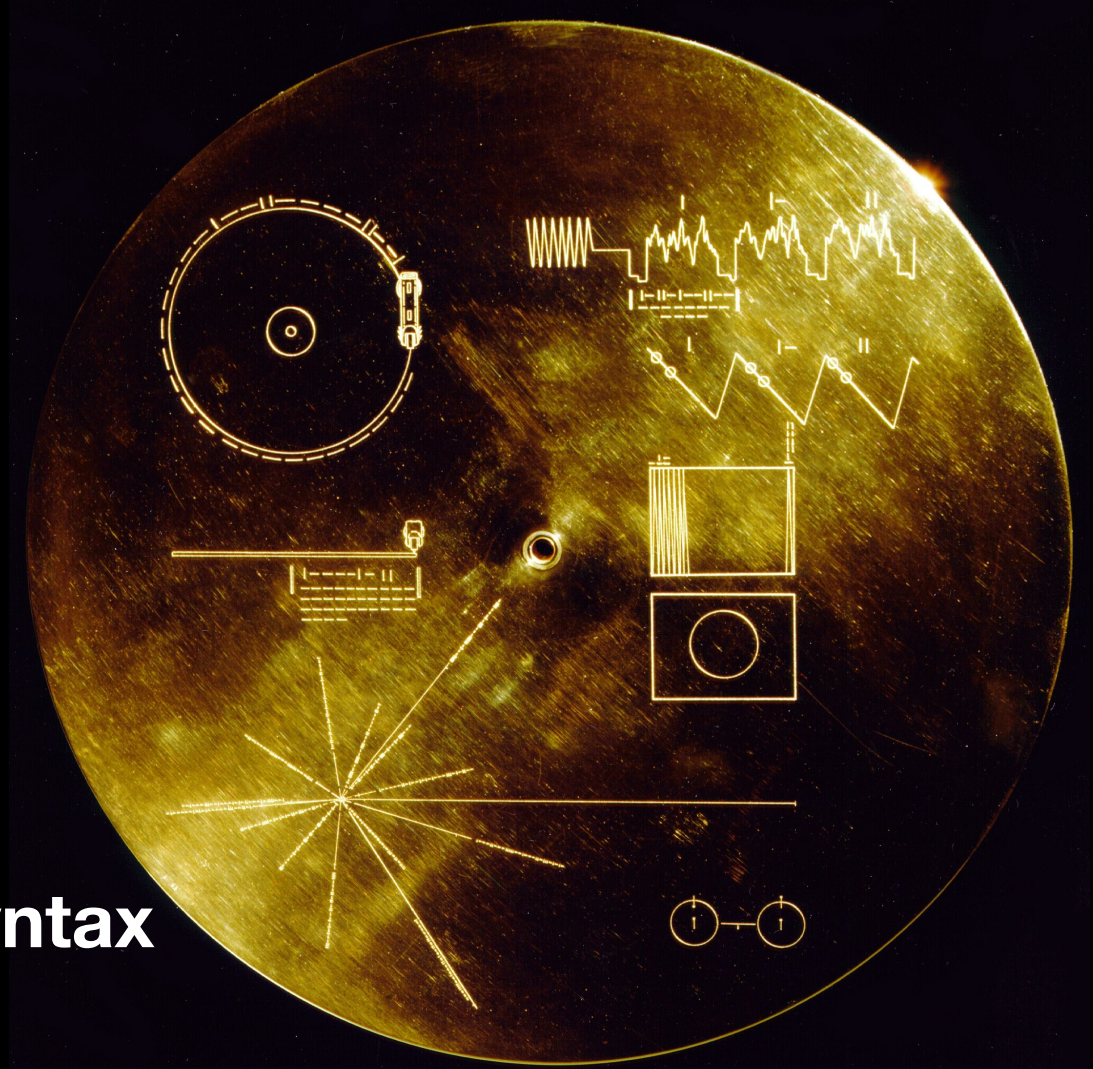
Episode X: Encore

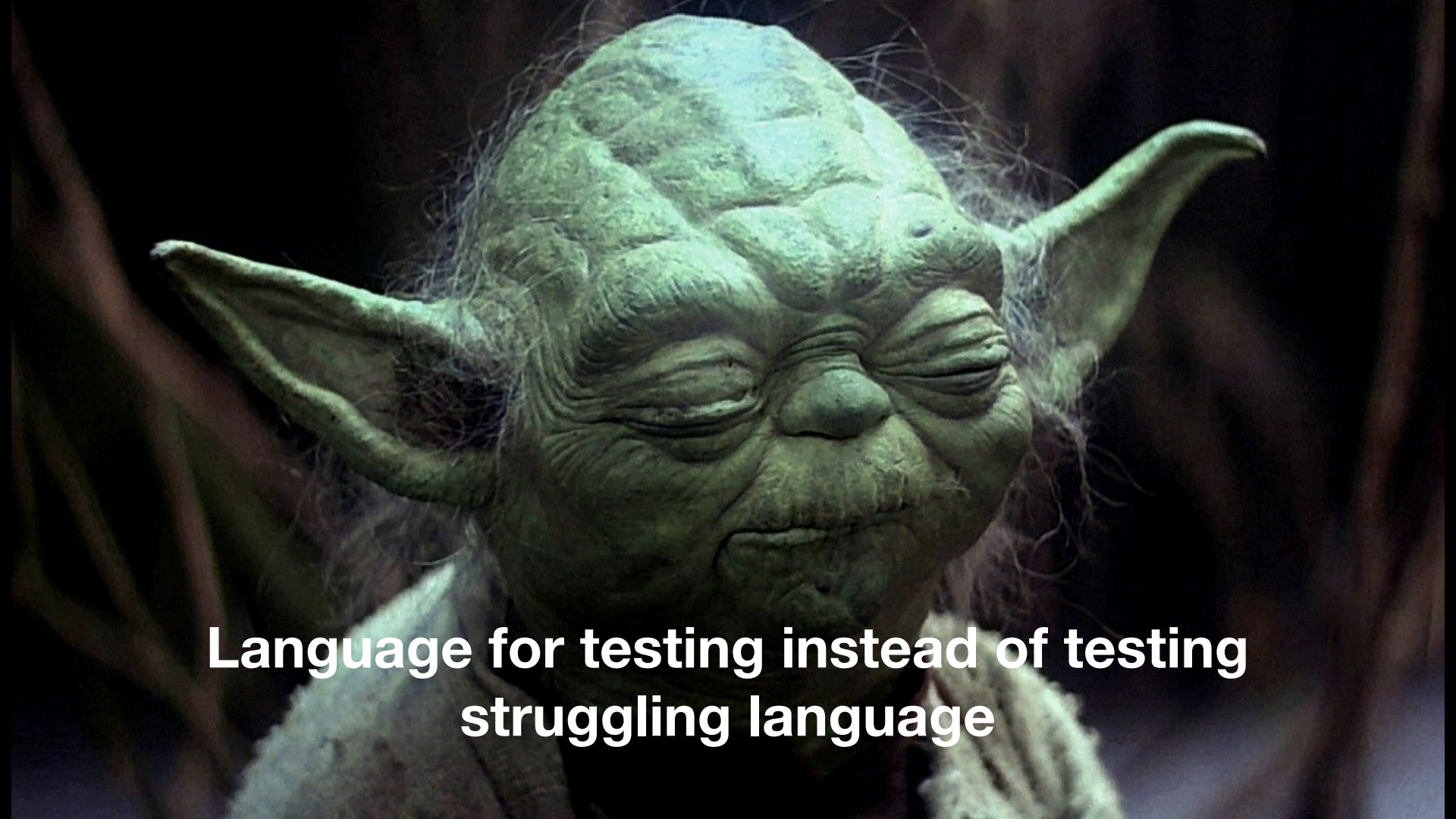
Episode X: Encore

```
@Stepwise
class N41S_SystemProperties extends Specification {

    @RestoreSystemProperties
    def 'set spock version'() {
        expect: 'that spock.version is not set'
        System.getProperty('spock.version') == null

        when: 'spock version is set'
        System.setProperty('spock.version', '1.0')

        then: 'we can retrieve its value back'
        System.getProperty('spock.version') == '1.0'
    }

    def 'check spock version is not set'() {
        expect: 'that spock.version is not set'
        System.getProperty('spock.version') == null
    }
}
```


Why one should use Spock?

Concise syntax




Language for testing instead of testing
struggling language

Clear test structure

Readable code and reports

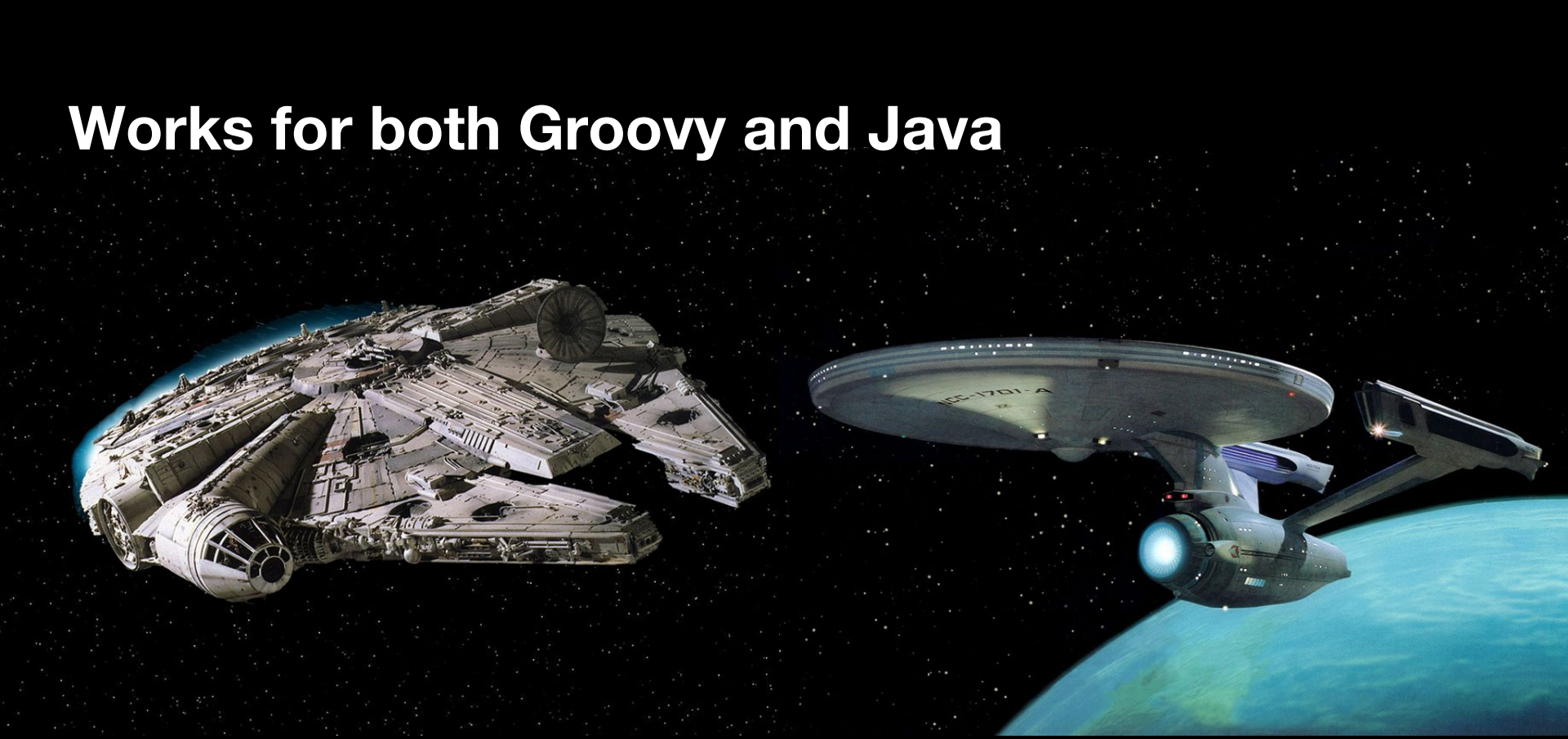
Powerful built-in features

Extensions

Useful for unit, integration and functional tests

Behaves as one more JUnit runner

Works for both Groovy and Java

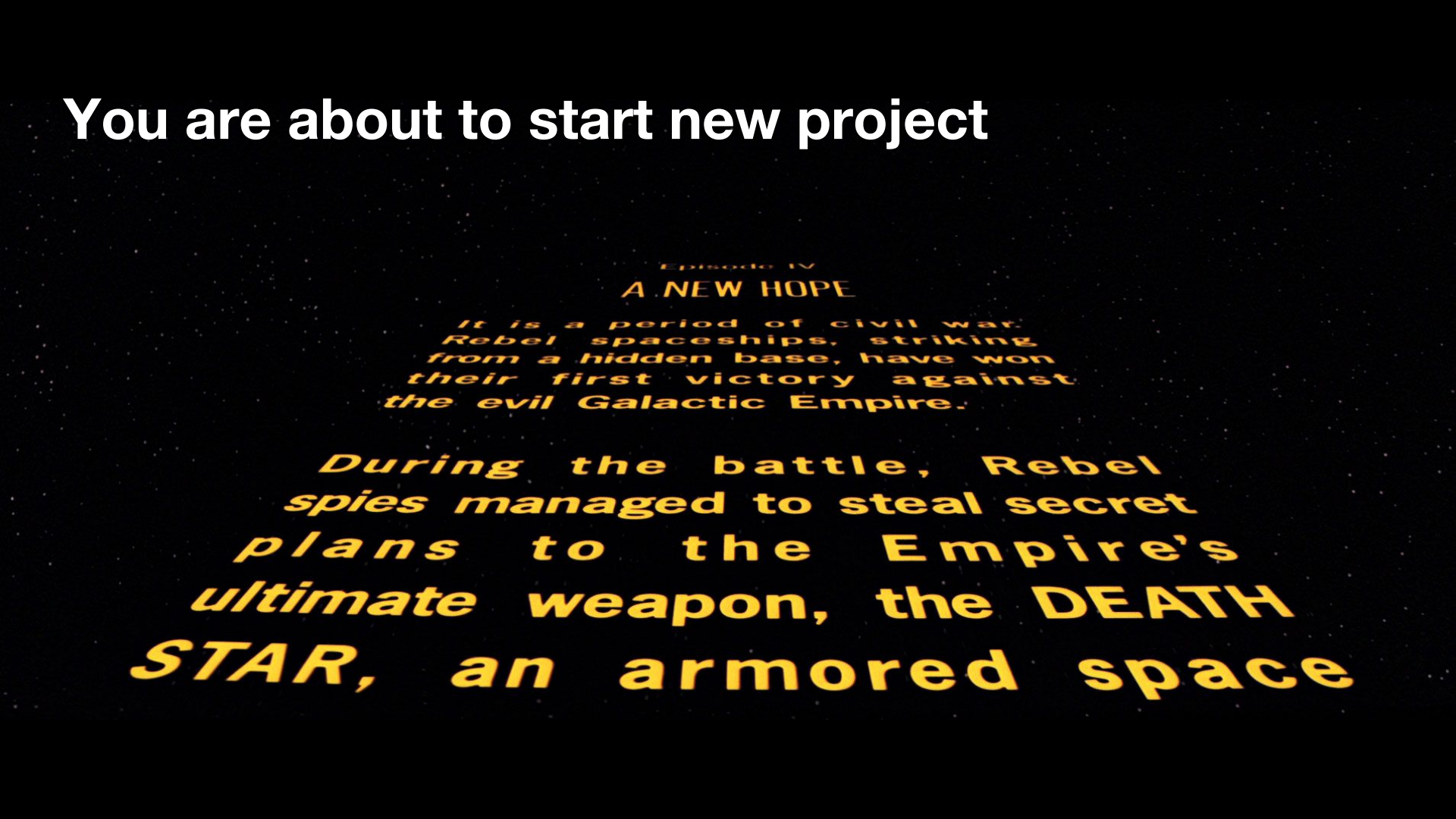


So many more reasons you’ve just seen


When one can use Spock?


Your team is ready to make one step forward

You are about to start new project



You are about to cover legacy project with unit tests

You are about to introduce functional testing in your project


Why one should not use Spock?

JUnit/TestNG really works

JUnit/TestNG has 90% similar/imitable features

Groovy and Spock DSL are tricky (not really)

When one should not use Spock?

What's next?

http://spockframework.github.io/spock/docs/1.0/index.html

http://meetspock.appspot.com/

https://github.com/spockframework

Just use it in your projects and enjoy!


Thanks!