21.08.2019
MUNICH / SCALA MEETUP

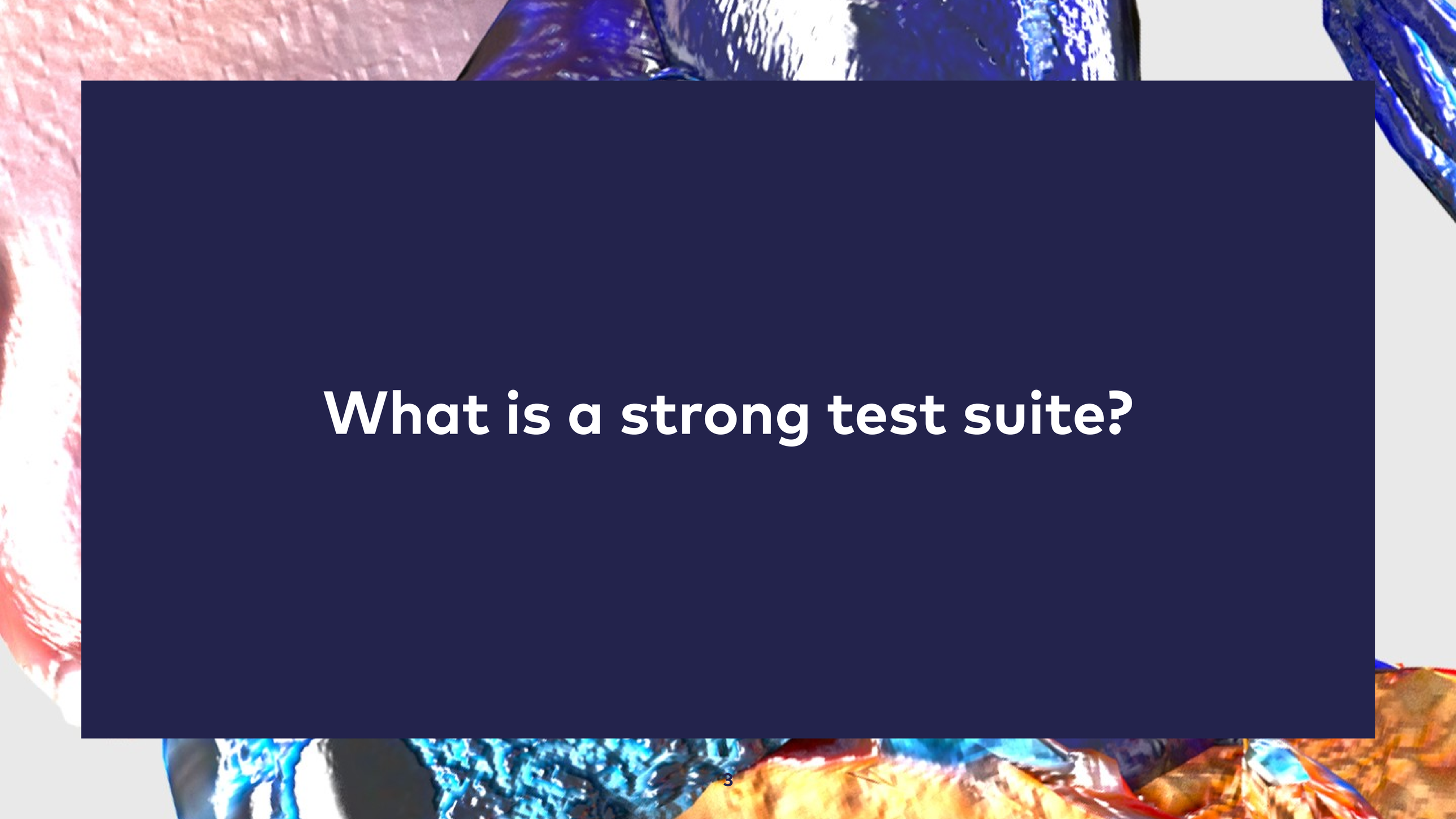
Testing in the postapocalyptic future

Daniel Westheide

Twitter: @kaffeecoder

INNOQ


Postapocalyptic present



What is a strong test suite?

Coverage?

Complex logic in need of a test

```
object Math {
  def nonNegative(x: Int): Boolean = x >= 0
}
```

100% branch coverage ☺

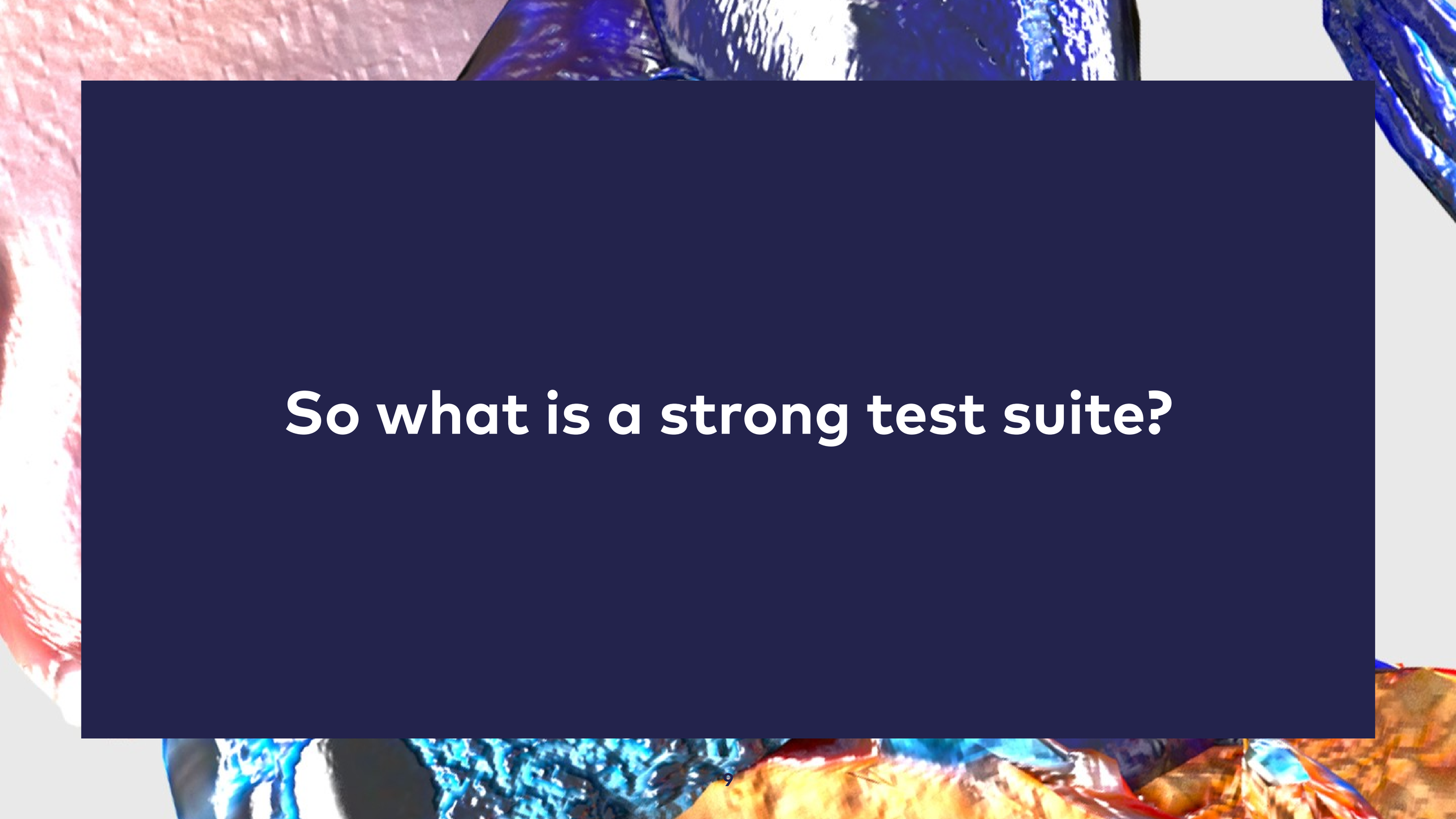
```
import minitest._

object MathTest extends SimpleTestSuite {

  import Math.nonNegative

  test("1 is non-negative") {
    assert(nonNegative(1))
  }

  test("0 is non-negative") {
    assert(nonNegative(0))
  }

  test("-1 is negative") {
    assert(!nonNegative(-1))
  }
}
```

Still 100% branch coverage 

```
import minitest._

object MathTest extends SimpleTestSuite {

  import Math.nonNegative

  test("1 is non-negative") {
    assert(nonNegative(1))
  }
}
```

100% coverage? Holy moly! 😱

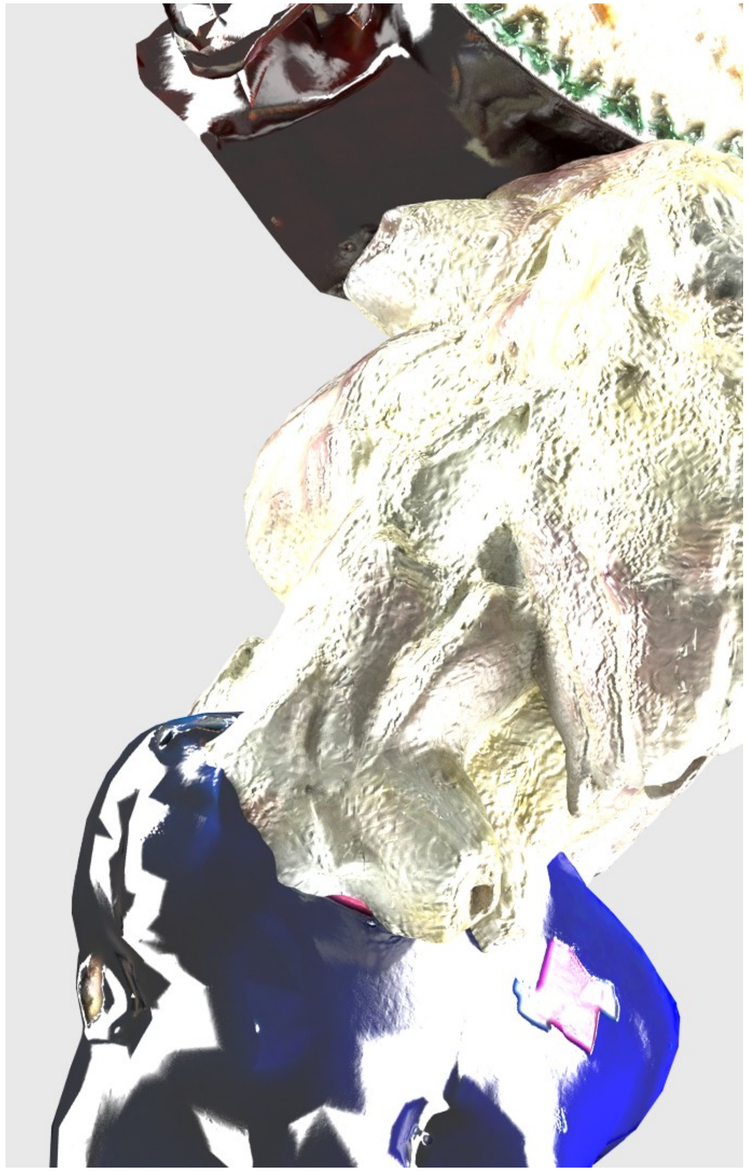
```
import minitest._

object MathTest extends SimpleTestSuite {

  import Math.nonNegative

  test("1 is non-negative") {
    println(nonNegative(1))
  }
}
```



So what is a strong test suite?

Characteristics of a strong test suite

- tests actually have assertions 😉
- not just the happy path
- covers corner cases



Property-based testing

More maths

```
object Math {

  def nonNegative(x: Int): Boolean = x >= 0

  def nonNegativeRatio(xs: List[Int]): BigDecimal = {
    val count = xs.count(nonNegative)
    BigDecimal(count) * 100 / xs.size
  }

}
```

Example-based testing

```
import minitest._

object MathTest extends SimpleTestSuite {

  import Math._

  test("100% non-negative ratio") {
    assertEquals(nonNegativeRatio(List(1, 2, 3)).toInt, 100)
  }

  test("0% non-negative ratio") {
    assertEquals(nonNegativeRatio(List(-5, -3, -2)).toInt, 0)
  }

  test("50% non-negative ratio") {
    assertEquals(nonNegativeRatio(List(-5, 1, 0, -1)).toInt, 50)
  }

}
```

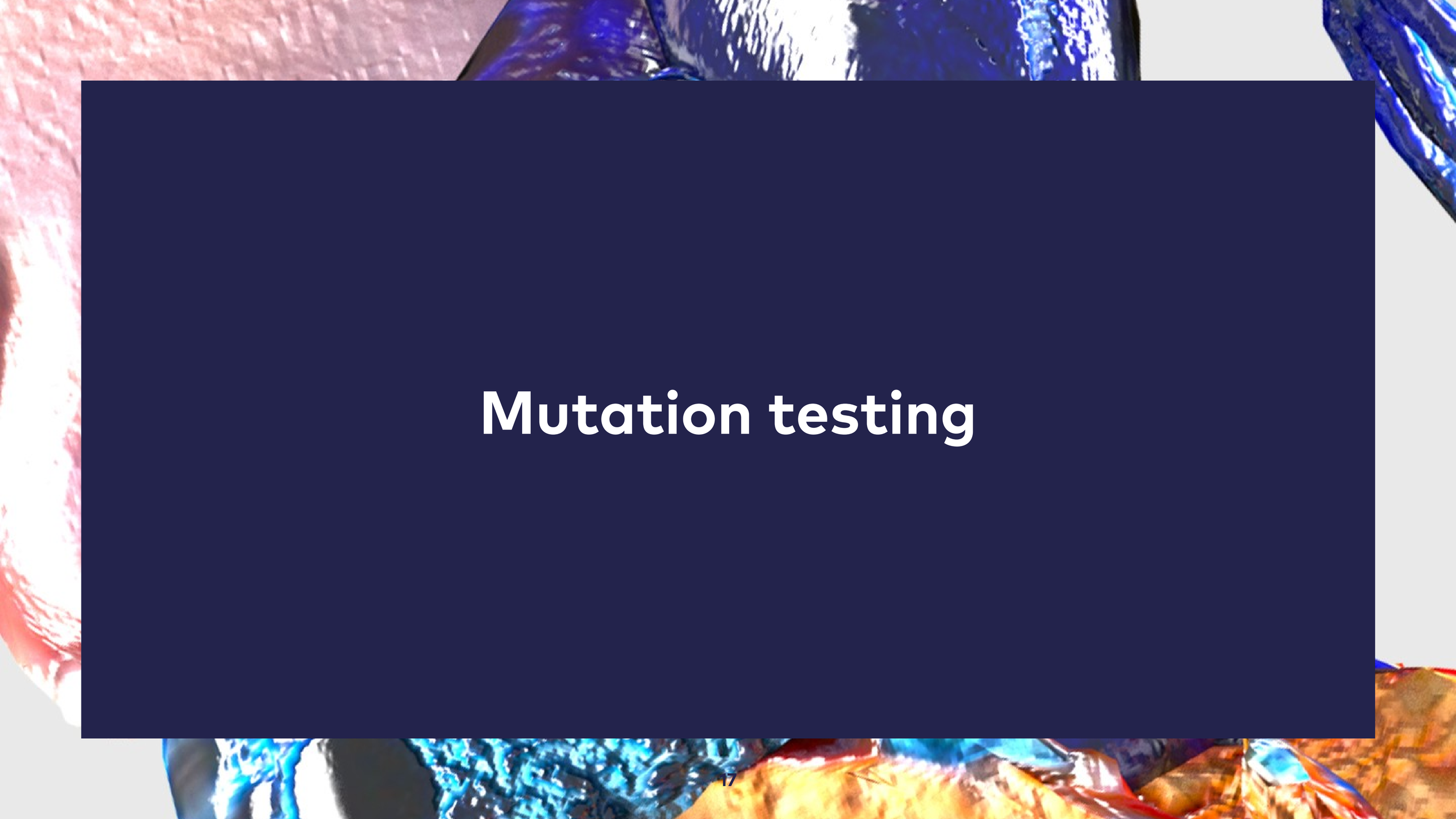
Property-based testing

```
import minitest.SimpleTestSuite
import minitest.laws.Checkers
import org.scalacheck.Prop.AnyOperators

object MathProperties extends SimpleTestSuite with Checkers {

  import Math._

  test("ratios of given numbers and inverted numbers adds up to 100") {
    check1((xs: List[Int]) => {
      val ys = xs.map(invert)
      (nonNegativeRatio(xs) + nonNegativeRatio(ys)).toInt ?= 100
    })
  }

  private def invert(x: Int): Int = x match {
    case 0 => -1
    case Int.MinValue => Int.MaxValue
    case _ => x * -1
  }

}
```

Boom! 💥

```
- ratios of given numbers and inverted numbers adds up to 100 *** FAILED ***
  Exception raised on property evaluation. (Checkers.scala:39)
  > ARG_0: List()
  > Exception: java.lang.ArithmeticException: Division undefined
    minitest.api.Asserts.fail(Asserts.scala:103)
```


Who watches the watchers?



Mutation testing

Mutation testing

How does it work?

- It changes the code under test
- Then runs your test suite against the modified code
- Do your tests care?

Mutations

Examples:

- Replace > with >=
- Replace > with <
- Replace >= with >
- Replace >= with <=

From mutation to mutant

```
// function under test:
def posRange(start: Int, end: Int): Boolean = start > 0 && end > start

// mutant 1:
def posRange(start: Int, end: Int): Boolean = start >= 0 && end > start

// mutant 2:
def posRange(start: Int, end: Int): Boolean = start < 0 && end > start

// mutant 3:
def posRange(start: Int, end: Int): Boolean = start > 0 && end >= start

// mutant 4:
def posRange(start: Int, end: Int): Boolean = start > 0 && end < start

// mutant 5:
def posRange(start: Int, end: Int): Boolean = start > 0 || end > start
```

Detecting a mutant

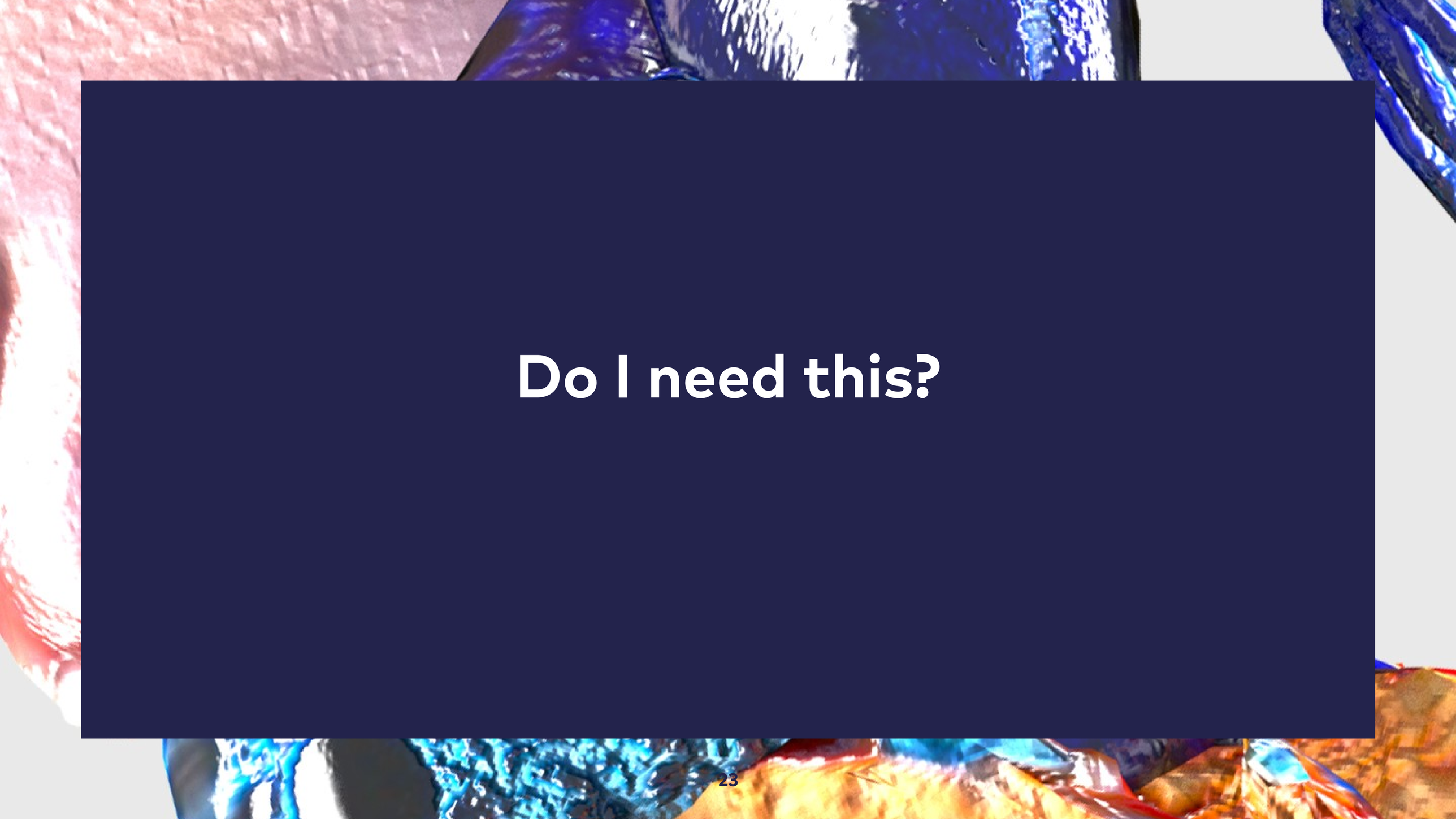
```
def nonNegative(x: Int): Boolean =
  x >= 0

// MUTANT 1:
def nonNegative(x: Int): Boolean =
  x > 0

// MUTANT 2:
def nonNegative(x: Int): Boolean =
  x < 0

// MUTANT 3:
def nonNegative(x: Int): Boolean =
  x == 0
```

```
import minitest._

object MathTest extends SimpleTestSuite {
  import Math._

  // FAILS FOR MUTANT 1 and 2!
  test("0 is non-negative") {
    assert(nonNegative(0))
  }
  // FAILS FOR MUTANT 2 and 3 !
  test("1 is non-negative") {
    assert(nonNegative(1))
  }
  // FAILS FOR MUTANT 2 and 3!
  test("-1 is negative") {
    assert(!nonNegative(1))
  }
}
```

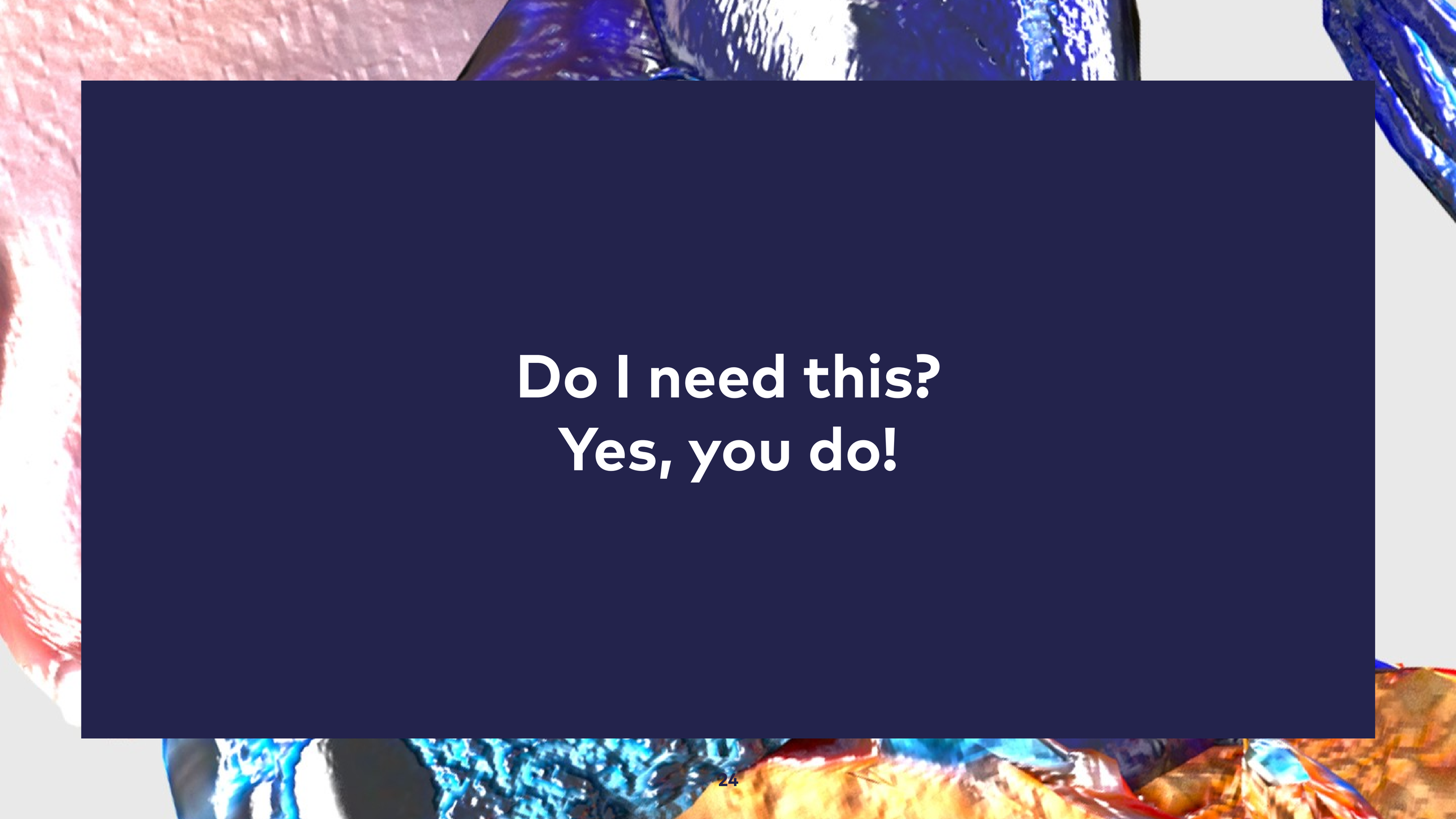
Missing a mutant

```
def nonNegative(x: Int): Boolean =
  x >= 0

// MUTANT 1:
def nonNegative(x: Int): Boolean =
  x > 0

// MUTANT 2:
def nonNegative(x: Int): Boolean =
  x < 0

// MUTANT 3:
def nonNegative(x: Int): Boolean =
  x == 0
```

```
import minitest._


object MathTest extends
SimpleTestSuite {

  import Math._

  // SUCCEEDS FOR MUTANT 1!
  test("1 is non-negative") {
    assert(nonNegative(1))
  }
}
```



Do I need this?



Do I need this?
Yes, you do!

“Okay Daniel, you have convinced me! I need this mutation testing thing for my project! Where can I buy it?“

HAYDEN HIPSTER

Purchasing Director 🛒
at Brontosaurus Enterprise Solutions

Daniel Westheide
@kaffeecoder

(2/2) Can't help thinking that ScalaMeta would be a great foundation for a Scala mutation testing library.

2:50 PM - 2 Jan 2017

5 Likes

3 5


Die Schwarze Auge
Einblick in das Ge
nes Gegenübers.
Kosten: 5 Astralpunkte.
Reichweite: 20 Meter.
Dauer: 10 Sekunden.
XIII SALANDER-MUTANDERER –
Sei ein anderer!
Beschreibung und Wirkungsweise: Diese Form
klassische Zauber, mit dem der Magier

Salander Mutanderer

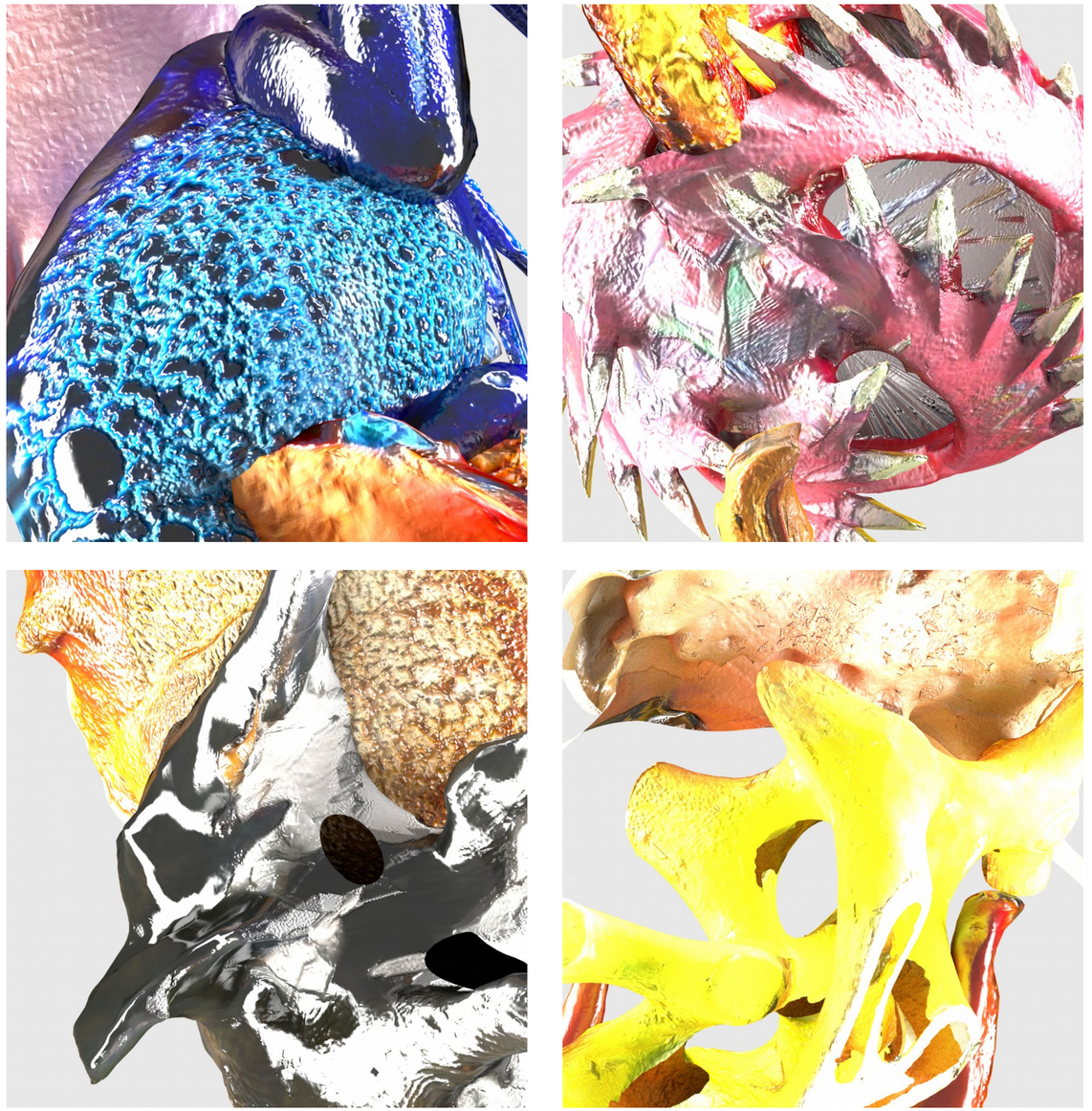
Ingredients: SBT and Scalameta

i. Find all Scala source files under test

ii. For each source file, yield mutants for applicable mutations

iii. Run test suite for each mutant

iv. Analyse and log results



Salander Mutanderer

```
def salanderMutanderer(incantation: Incantation)(
    implicit logger: ManagedLogger): Unit = {

  val results = for {
    sourceFile <- scalaSourceFiles(incantation)
    mutant <- summonMutants(sourceFile)
  } yield runExperiment(incantation, mutant)

  val stats = analyseResults(results)
  logResults(stats)
}
```

Scalameta

- library for reading, analysing, transforming, and generating Scala code
- Tree
  - syntax tree representation of Scala code
  - pattern matching
  - comparison
  - traversal

Traversing a tree

```
def collect[T](fn: PartialFunction[Tree, T]): List[T]
```

Defining a mutation

```
type Mutation = SourceFile => PartialFunction[Tree, List[Mutant]]

val `Change >= to > or ==` : Mutation = sourceFile => {
  case original @ Term.ApplyInfix(_, Term.Name(">="), Nil, List(_)) =>
    List(
      Mutant(UUID.randomUUID(),
        sourceFile,
        original,
        original.copy(op = Term.Name(">")),
        "changed >= to >"),
      Mutant(UUID.randomUUID(),
        sourceFile,
        original,
        original.copy(op = Term.Name("==")),
        "changed >= to ==")
    )
}
```

The spellbook

```
val spellbook: List[Mutation] = List(
  `Change >= to > or ==`,
  `Replace Int expression with 0`
)
```

Summoning mutants

```
private def summonMutants(sourceFile: SourceFile): List[Mutant] =
  sourceFile.source.collect(Mutations.in(sourceFile)).flatten

object Mutations {
  def in(sourceFile: SourceFile): PartialFunction[Tree, List[Mutant]] =
    spellbook
      .map(_.apply(sourceFile))
      .foldLeft(PartialFunction.empty[Tree, List[Mutant]])(_.orElse(_))
}
```

Result of an experiment

```
sealed trait Result extends Product with Serializable
final case class Detected(mutant: Mutant) extends Result
final case class Undetected(mutant: Mutant) extends Result
final case class Error(mutant: Mutant) extends Result
```

Running an experiment

```
private def runExperiment(incantation: Incantation, mutant: Mutant)(
    implicit logger: Logger): Result = {

  val mutantSourceDir = createSourceDir(incantation, mutant)

  val settings = mutationSettings(
    mutantSourceDir,
    incantation.salanderTargetDir / mutant.id.toString)

  val newState = Project
    .extract(incantation.state)
    .appendWithSession(settings, incantation.state)

  Project.runTask(test in Test, newState) match {
    case None                    => Error(mutant)
    case Some((_, Value(_)))     => Undetected(mutant)
    case Some((_, Inc(_)))       => Detected(mutant)
  }
}
```

Example

$ sbt salanderMutanderer

```scala
object Math {
  def nonNegative(x: Int): Boolean = x >= 0
}
```

```scala
import minitest._

object MathTest extends SimpleTestSuite
{
  import Math._

  test("-1 is non-negative") {
    assert(!nonNegative(-1))
  }
}
```

```
[info] Passed: Total 1, Failed 0, Errors 0, Passed 1
[info] Total mutants: 2, detected mutants: 0 (0%)
[info] Undetected mutants:
[info] /home/daniel/projects/private/salander/src/main/scala/lib/Math.scala:5:38:    changed >= to ==
[info] /home/daniel/projects/private/salander/src/main/scala/lib/Math.scala:5:38:    changed >= to >
[success] Total time: 8 s, completed 6 Jun 2019, 11:24:13
```

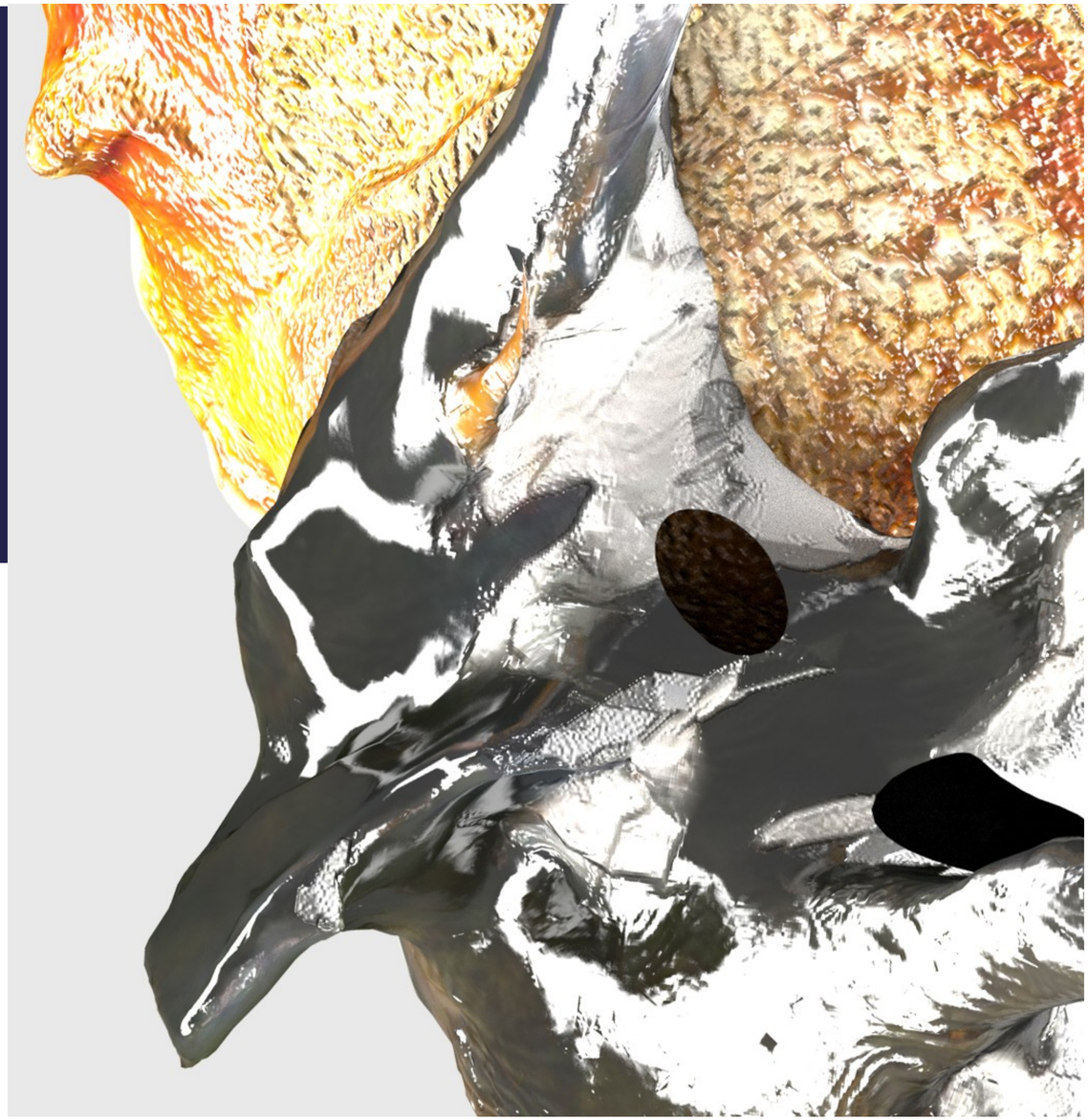
Weaknesses

- very limited set of mutations
- only basic console reporting
- performance

A slight case of thread necromancy

“The future is already here — it’s just not very evenly distributed.“

WILLIAM GIBSON



Supported mutations

- **boolean literal:** e.g. replace `true` with `false`
- **conditional expression:** e.g. `if (x < 3) ...` => `if (false) ...`
- **equality operator:** e.g. replace `>` with `>=`
- **logical operator:** e.g. replace `&&` with `||`
- **method expression:** e.g. `a.take(b)` => `a.drop(b)`
- **string literal:** e.g. replace “`Magic`” with “”

HTML report

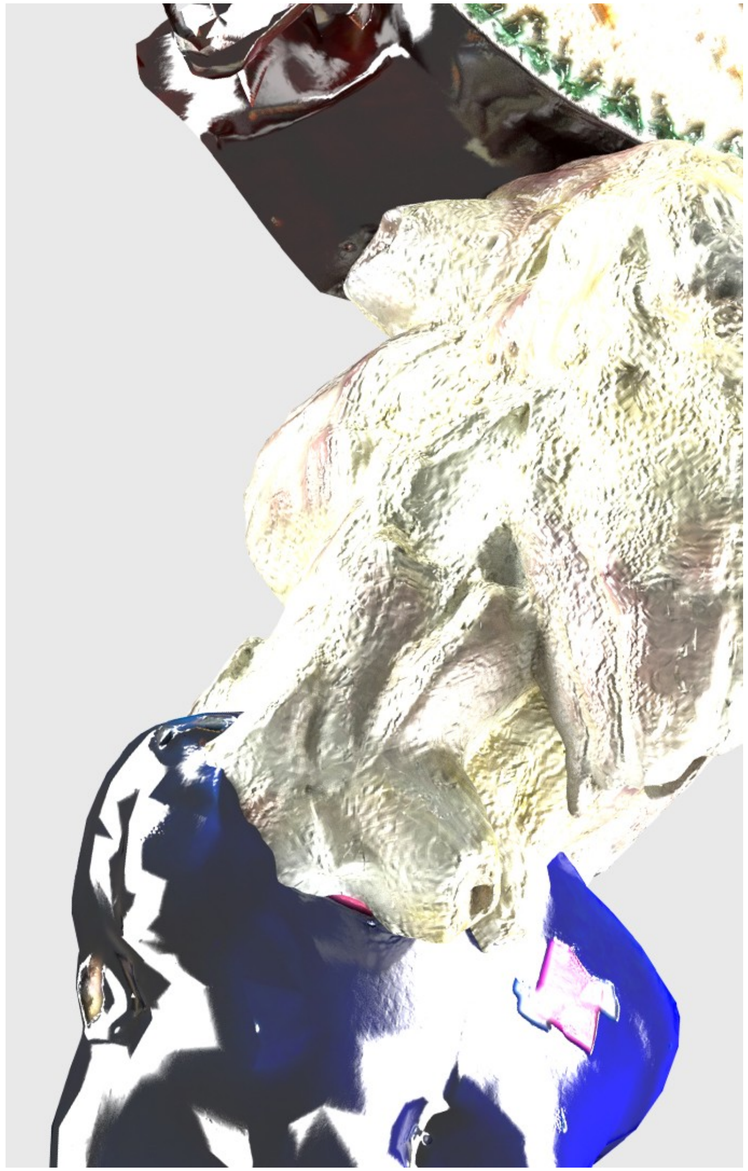
☑ Killed (1) ☑ Survived (2) Expand all

```
package com.danielwestheide.ktxfr.wordcount

import com.danielwestheide.kontextfrei.DCollectionOps
import com.danielwestheide.kontextfrei.syntax.SyntaxSupport

trait WordCount extends SyntaxSupport {

  def counts[F[_]: DCollectionOps](text: F[String]): F[(String, Long)] =
    text
      .flatMap(line => line.split(2" "))
      .map(word => (word, 1L))
      .reduceByKey(_ + _)
      .sortBy(_._2, ascending = 3 true ~~false~~)

  def formatted[F[_]: DCollectionOps](counts: F[(String, Long)]): F[String] =
    counts.map {
      case (word, count) => 4 s"" ~~s"$word,$count"~~
    }
}
```

StringLiteral

Performance

- there are faster compilers than scalac
- potentially great number of mutants
  - mutate source file
  - compile
  - run tests



Mutation switching

```
def nonNegative(x: Int): Boolean = sys.env.get("ACTIVE_MUTATION") match {
  case Some("0") => x > 0
  case Some("1") => x <= 0
  case Some("2") => x == 0
  case _ => x >= 0
}
```

Usage

- available as a plugin for **SBT** or **Maven**
- `addSbtPlugin("io.stryker-mutator" % "sbt-stryker4s" % "0.6.1")`
- `sbt stryker`
- configuration using *stryker4s.conf* file
  - disable certain mutations
  - narrow down source files to be considered for mutation
  - change base directory
  - set threshold for detection rate

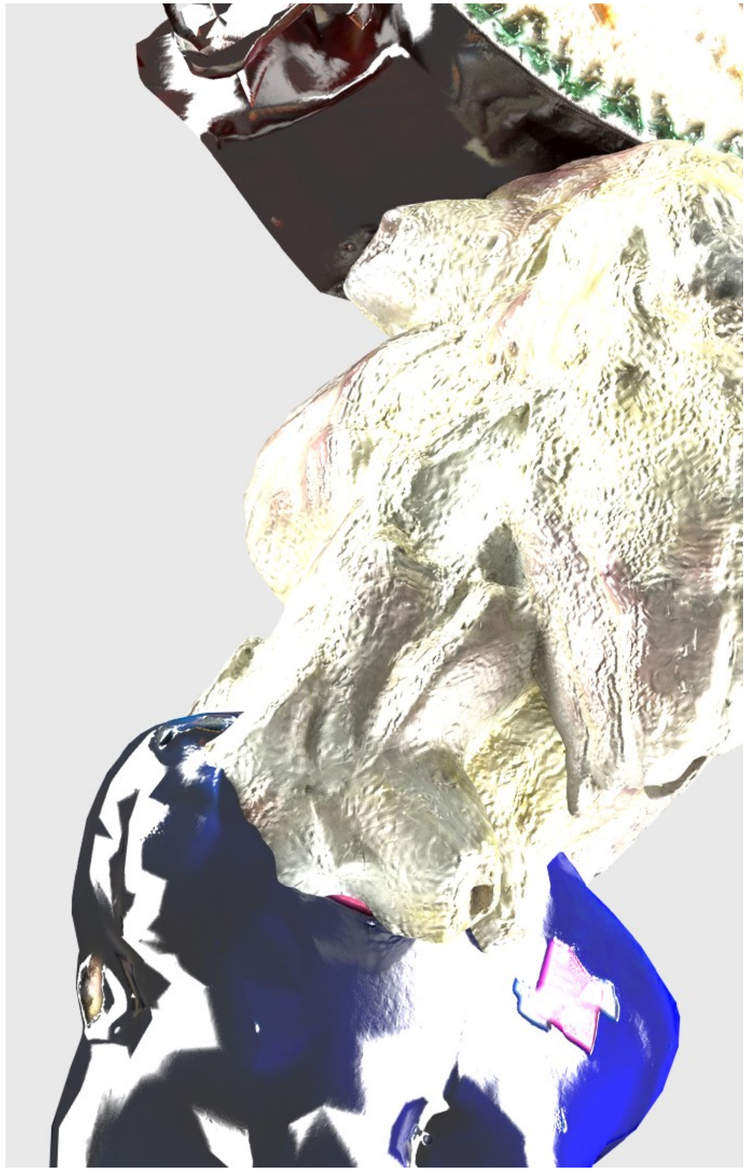
Usage patterns

- do not put this into your delivery pipeline
- do not enforce a minimum detection rate
- run either
  - locally from time to time
  - in a scheduled job
- plan time to go through the report and improve your test suite

Challenges and limitations

Non-compiling mutants

- having a `filter` method doesn't guarantee that there is a `filterNot` method
- even operators are just methods, e.g. > and >=
- problem: no type information available
- compile errors cause the whole mutation test run to fail

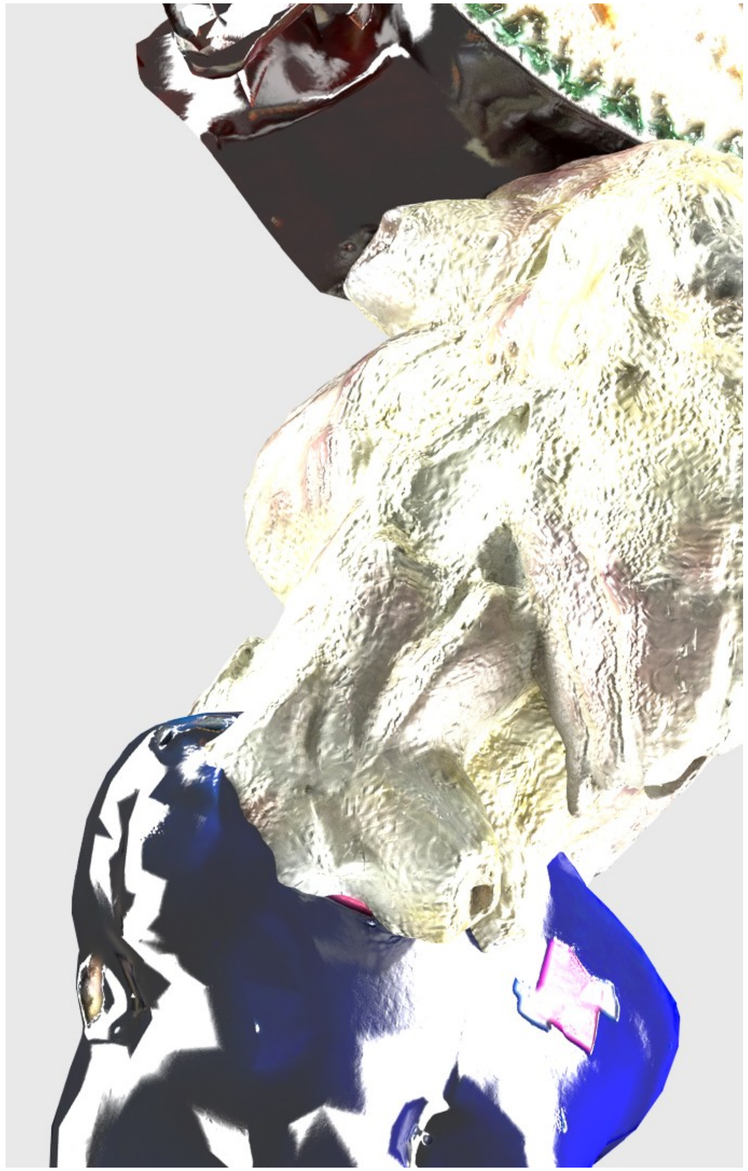


Other issues

- Infinite loops caused by mutations cannot be stopped
- no native support for multi module projects
- performance: running the complete test suite for each mutant

Ideas and plans

- Performance:
  - keeping the test process alive
  - ignore mutants without test coverage
  - for each mutant, only run tests hitting the respective code
- Robustness
  - rolling back mutations causing compile errors
  - exploring feasability of type analysis using SemanticDB



Towards a postapocalyptic future!

Links

- https://github.com/dwestheide/salander
- https://github.com/stryker-mutator/stryker4s
- https://stryker-mutator.io/blog/2018-10-6/mutation-switching
- https://scalameta.org/

Thank you! Questions?

Daniel Westheide
daniel.westheide@innoq.com

Twitter: @kaffeecoder
Website: https://danielwestheide.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin
Germany
+49 2173 3366-0

Ludwigstr. 180E
63067 Offenbach
Germany
+49 2173 3366-0

Kreuzstr. 16
80331 München
Germany
+49 2173 3366-0

Hermannstrasse 13
20095 Hamburg
Germany
+49 2173 3366-0

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
+41 41 743 0116