Model-based Testing

Cyrille Artho
KTH Royal Institute of Technology, Stockholm, Sweden
Electrical Engineering and Computer Science
Theoretical Computer Science Group
artho@kth.se

Unit Testing: Writing the tests

```
@Test void test1() {
    pos = p0;
    left();
    right();
    assert(pos == p0);
}
```

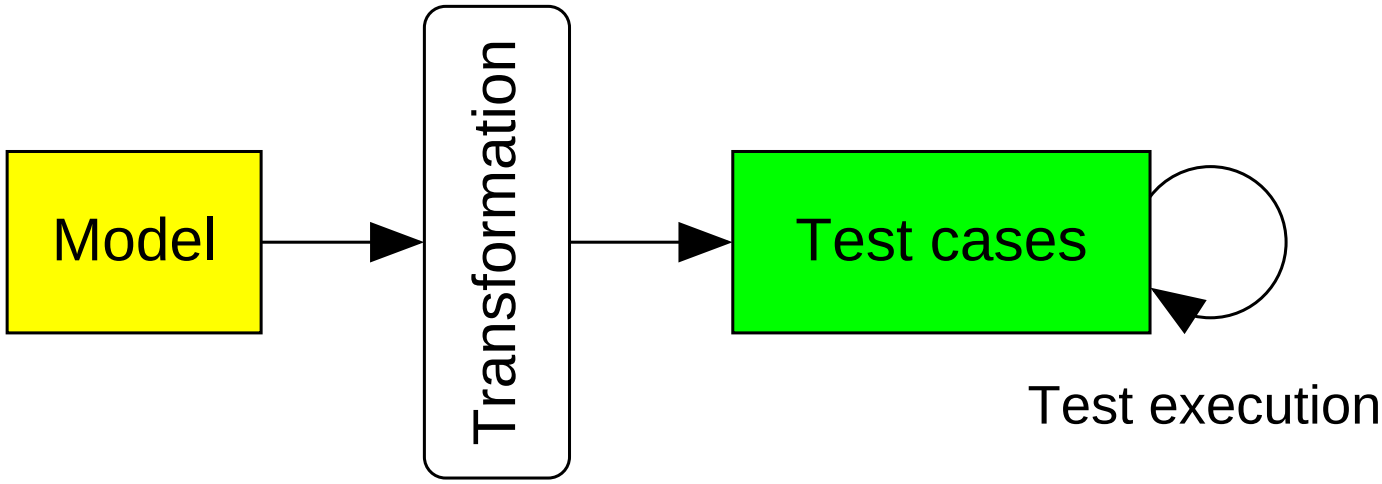
```
@Test void test2() {
    pos = p0;
    right();
    left();
    assert(pos == p0);
}
```

```
@Test void test3() {
    pos = p0;
    left();
    left();
    right();
    right();
    assert(pos == p0);
}
```

```
@Test void test4() {
    pos = p0;
    left();
    right();
    right();
    left();
    assert(pos == p0);
}
```

Can this be automated?

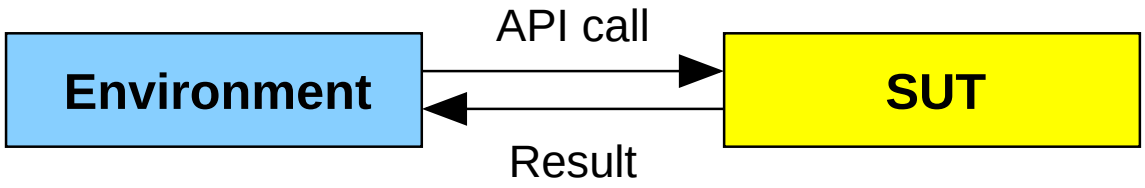
Model-based Testing



- Model contains:
  - Formalized description of the system behavior.
  - Expected output or state.
- Transformation tool generates (and/or executes) test cases.

https://en.wikipedia.org/wiki/Model-based_testing

Test Model vs. System Model



SUT = System under test; API = Application programming interface

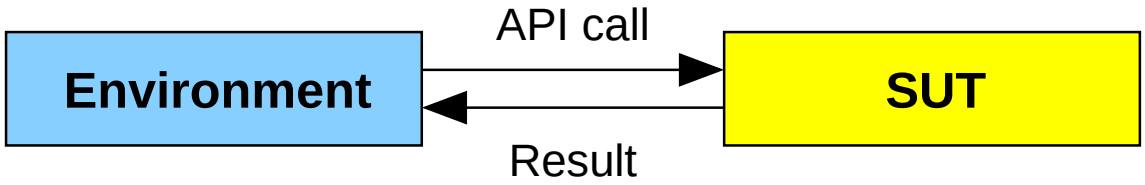
Test model

What

System model

How

Test Model vs. System Model



SUT = System under test; API = Application programming interface

Test model

- Represents **environment**.
- Models system **behavior**.
- Used to generate **test** cases.
- Model, test one module at a time; SUT itself provides counterpart.
- Model-based testing.

System model

- Represents **system** itself.
- Models system **implementation**.
- Used to **verify** system.
- Need model of most components to analyze system behavior.
- Model checking, theorem proving.

Key Challenge

Model

Real system

Model needs enough detail to create interesting test cases.

Property-based testing with „ScalaCheck”

```
import org.scalacheck.Prop.forAll
val propReverseList = forAll {
  l: List[String] => l.reverse.reverse == l
}
val propConcatString = forAll {
  (s1: String, s2: String) =>
    (s1 + s2).endsWith(s2)
}
```

Generic properties, tested with random data.

On ScalaCheck

- Ideal to test stateless functions.
- Data generators can generated specific formats/distributions of data.
- Resources:
  http://scalacheck.org/
  https://github.com/rickynils/scalacheck/blob/master/doc/UserGuide.md
- ScalaCheck—The Definitive Guide, by Richard Nilsson (artima).

Write individual tests or generate them?

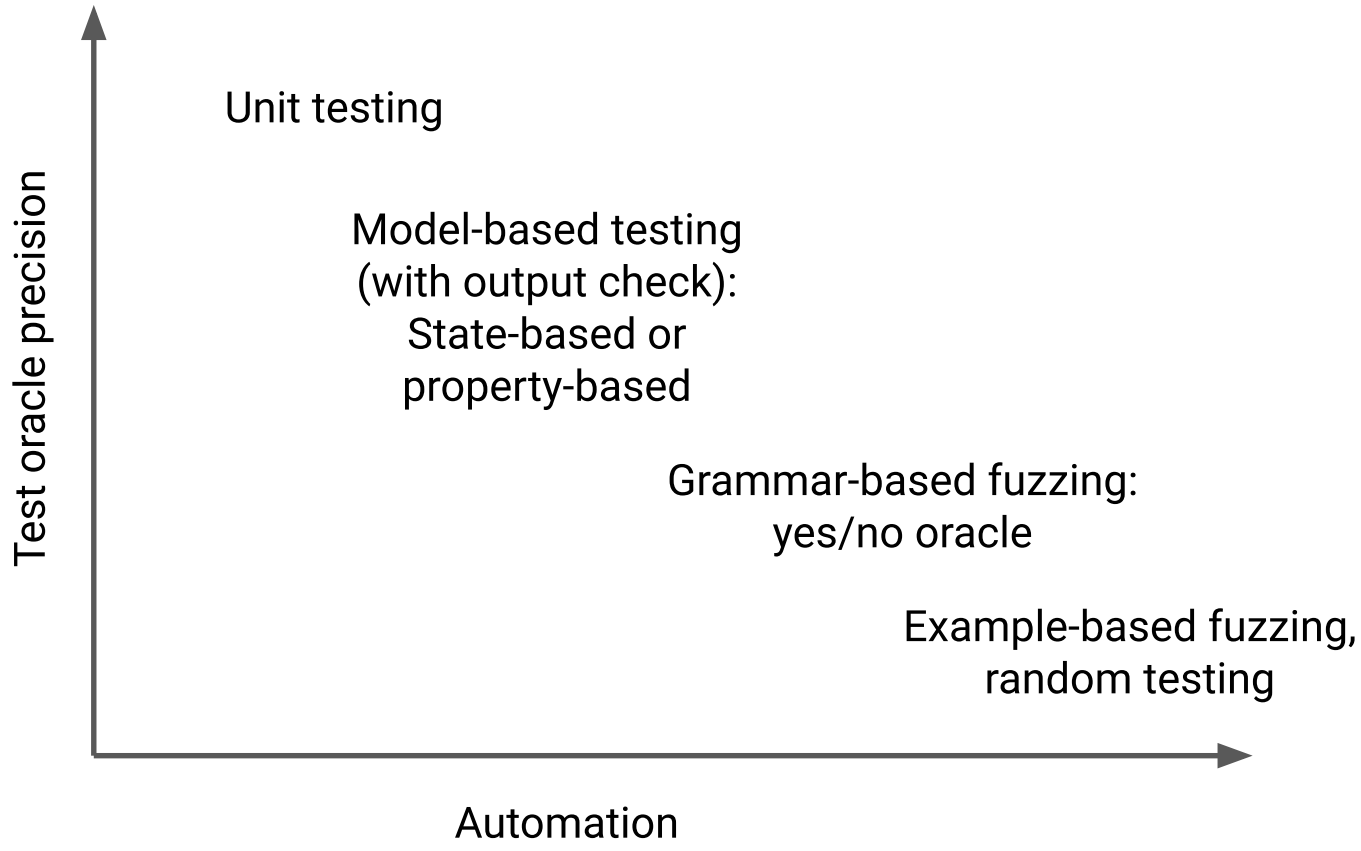
Complementary approaches!

1. **Start with individual, automated test cases.**
   This applies to unit testing as well as system testing, integration testing, etc.

2. **Use model-based testing if many similar tests are needed,**
   if a tool that supports a given problem is available.
   Properties and data generators: ScalaCheck.
   State-based systems: Modbat.

3. **Random testing can increase code coverage for free.**
   Note: less effective at finding bugs, because output is not validated!
   However, good at covering edge cases than humans forget.

Trade-offs between the approaches



Summary

- **Unit testing** automates test **execution.**
- **Model-based testing** automates test **generation.**

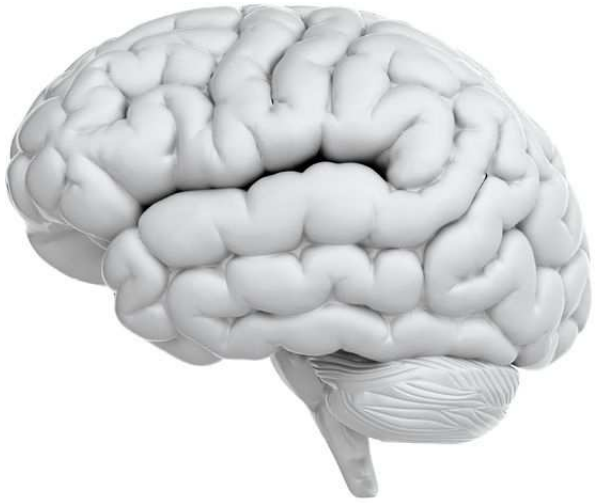
Challenges in model-based testing:

- How to abstract many test cases into a common model.
- How to specify the test output (oracle), at least partially.

Complementary approaches!

Models in Software Construction: Develop Model or Synthesize it?

Model development



✔ Easy to understand, debug.

✘ Costly, scope often limited.

Model synthesis

✘ Failed tests hard to debug.

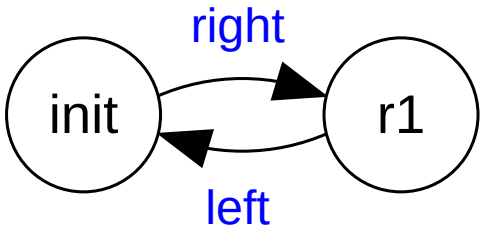
✘ Bugs in code $\rightarrow$ model.

✔ Automatic.

Modeling state-based tests with Modbat

Domain-Specific Language (DSL) based on Scala.

- Extended Finite-State Machine (EFSM) as base structure.
- Add transition functions, variables for complex state.
- Structured model but flexibility of full Scala (+ Java).




```
class Example extends Model {
  var r = 0
  "init" -> "r1" := { right; r += 1 }
  "r1" -> "init" := { left; assert (r > 0) }
}
```

Modbat and its tutorial

Modbat: `https://gitlab.com/cartho/modbat`

Tutorial: Click on „Wiki”, then „Tutorial”, or use this link:
`https://gitlab.com/cartho/modbat-tutorial/-/wikis/home`

- Java 11 and Scala 2.11.X required.
- First build Modbat with Gradle.
- Then run tutorial as shown on the web page.

Example: Java iterator

Model code

```
def next {
  require (valid)
  require (pos < dataModel.n-1)
  val res = SUT.next
  pos += 1
  assert (dataModel.data(pos) == res)
}
```

Workflow

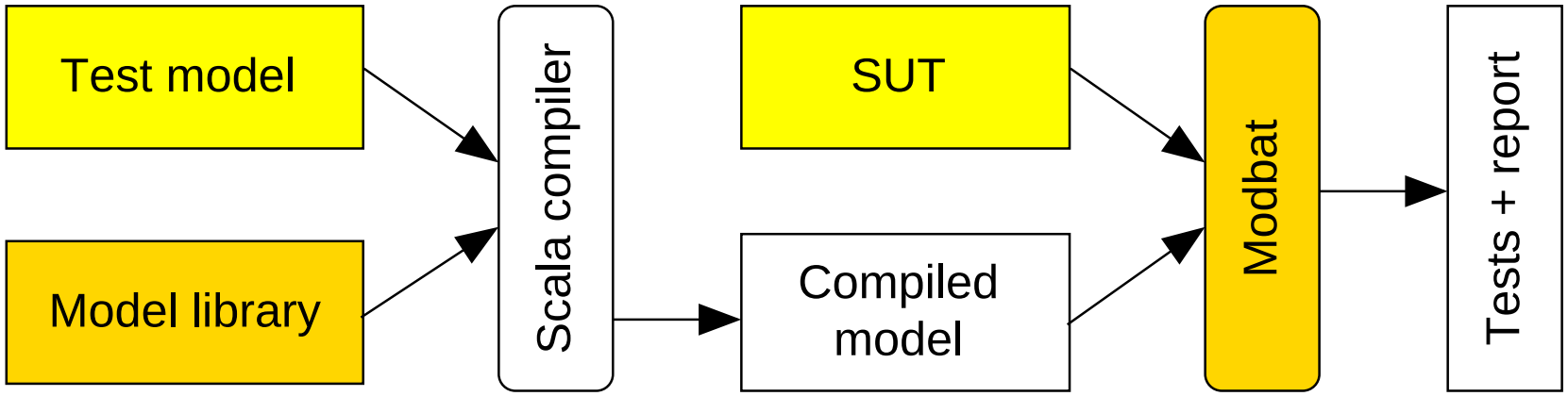
1. precondition 1
   precondition 2
2. action on SUT
   update model var.
3. verify result

Found 17-year-old bug with Modbat model!

Publication: C. Artho et al. Model-based Testing of Stateful APIs with Modbat. ASE 2015 tool demo.

Architecture and Workflow of Modbat

1. User defines test model, compiles it against DSL library.
2. Modbat runs compiled model against system under test (SUT).



- ✔ Simple yet expressive modeling language with Scala constructs.
- ✔ **One** line of code to specify each transition, non-determinism, exception.

Tutorial, Step 1: Model of Java collections (1/2)

```
class SimpleListModel extends Model {
  val N = 10 // range of integers to choose from
  val collection = new LinkedList[Integer]
    // the "system under test"
  var n = 0 // Number of elements in the collection
  def add {
    val element = new Integer(choose(0, N))
    val ret = collection.add(element)
    n += 1
    assert(ret)
  }
  def clear {
    collection.clear
    n = 0
  }
```

Model of Java collections (2/2)

```
  def remove {
    val obj = new Integer(choose(0, N))
    val res = collection.remove(obj)
    n = n - 1
  }
  def size {
    assert (collection.size == n,
            "Predicted size: " + n +
            ", actual size: " + collection.size)
  }
  "main" -> "main" := add
  "main" -> "main" := size
  "main" -> "main" := clear
  "main" -> "main" := remove
}
```

One model state for simplicity: non-deterministically try one of the four operations.

Explanation of „add”

```
def add {
  val element = new Integer(choose(0, N))
    // random number between 0 (incl.) and N (excl.)
  val ret = collection.add(element)
    // try to add new element (should always succeed)
  n += 1 // update model variable with new size
  assert(ret) // check return code
}
```

- Other functions work analogously.
- „size” only checks the size, does not update anything.

Test failure with SimpleListModel

```
./gradlew runSimpleList
[INFO] 5 tests executed, 0 ok, 5 failed.
[INFO] 2 types of test failures:
[INFO] 1) java.lang.AssertionError: assertion failed:
          Predicted size: 0, actual size: 1 at size:
[INFO]    ba471c1085a01750 2251f4042ff65867 89a677f51847fa26
[INFO] 2) java.lang.AssertionError: assertion failed:
          Predicted size: 1, actual size: 2 at size:
[INFO]    fd53cd70667aea0
[INFO] 1 states covered (100 % out of 1),
[INFO] 4 transitions covered (100 % out of 4).
[INFO] Random seed for next test would be: 4f9dd7062e2f7ae4
real 0m0.493s
user 0m0.505s
sys  0m0.078s
```

Analysis of failing test

```
[INFO] 1) java.lang.AssertionError: assertion failed:
          Predicted size: 0, actual size: 1 at size:
```

Type of uncaught exception: AssertionError, with message.

```
[INFO]    ba471c1085a01750 2251f4042ff65867 89a677f51847fa26
```

Random seeds of failed tests. A test can be replayed as follows:

```
scala -classpath build/classes/scala/test modbat.jar \
  -s=89a677f51847fa26 -n=1 model.simple.SimpleListModel
```

Each failed test also produces a trace file, e. g., 89a677f51847fa26.err.

Analysis of the trace file

A helper script provides colored output (for black background).

```
[WARNING] java.lang.AssertionError:assertion failed:
          Predicted size: 0, actual size: 1 occurred, aborting.
[ERROR] java.lang.AssertionError:assertion failed:
        Predicted size: 0, actual size: 1
[ERROR]  at scala.Predef$.assert(Predef.scala:170)
[ERROR]  at modbat.dsl.Model$class.assert(Model.scala:82)
[ERROR]  at model.simple.SimpleListModel.assert(SimpleListModel.scala:6)
[ERROR]  at model.simple.SimpleListModel.size(SimpleListModel.scala:30)
         ...
[WARNING] Error found, model trace:
[WARNING] model/simple/SimpleListModel.scala:35: add; choices = (1)
[WARNING] model/simple/SimpleListModel.scala:36: size
[WARNING] model/simple/SimpleListModel.scala:38: remove; choices = (4)
[WARNING] model/simple/SimpleListModel.scala:36: size
```

Sequence leading to failure: add(1), check size, remove(4), check size.

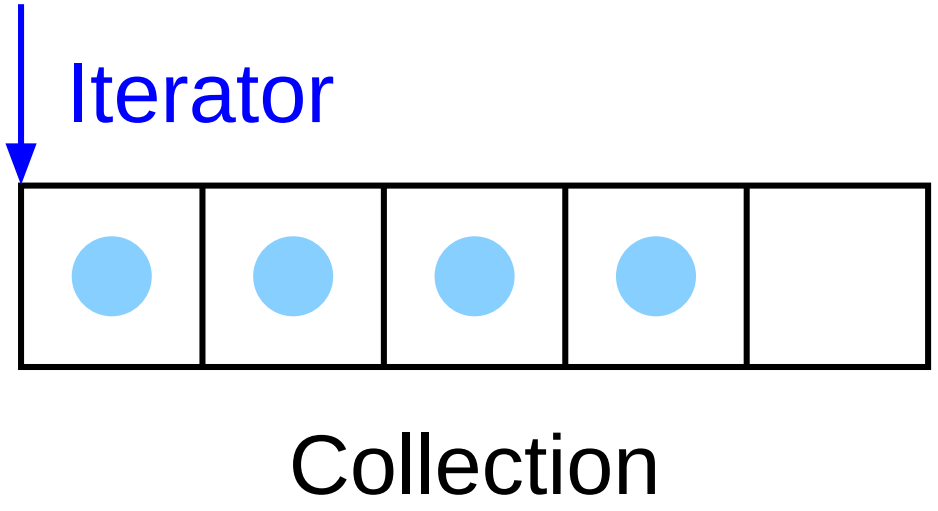
Try it yourself! How to run the exercise

```
git clone https://gitlab.com/cartho/modbat-tutorial
```

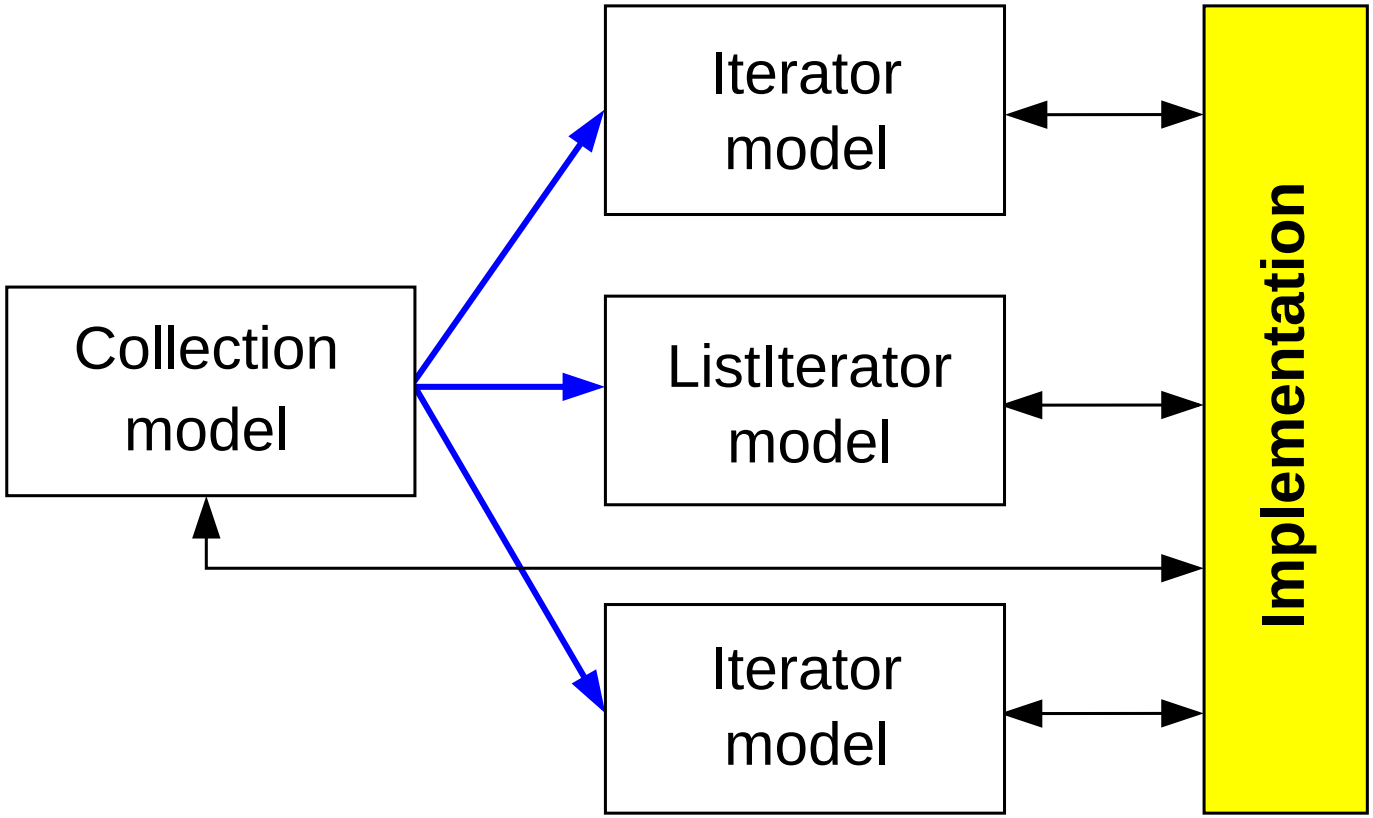
- Requires Scala 2.11.X; tested with Java 11. Everything else is provided.
- Compilation: Done automatically as part of running the examples
- Simple example: **`./gradlew runSimpleList`**
- Complex example: **`./gradlew runLinkedList`**

Java collections and iterators

	Collections	**Iterators**
Key methods	add, remove, size	hasNext, next



Iterator model



Note: This example has only plain iterators, no bidirectional ListIterators.

For an example with complex iterators, see:

https://people.kth.se/~artho/modbat/tooldemo/

Collections and iterators

- A *collection* holds a number of data items.
- An *iterator* can access these data items sequentially.
- An iterator is *valid* as long as the underlying collection has not been modified.
- *hasNext* queries if an iterator has more elements available.
- If an iterator goes beyond the last element, *NoSuchElementException* is thrown.
- If the collection has been modified, *ConcurrentModificationException* is thrown.

How to orchestrate multiple models

```
abstract class CollectionModel extends Model {
  val collection: Collection[Integer]
  // the "system under test"
  def iterator {
    val it = collection.iterator()
    val modelIt = new IteratorModel(this, it)
    launch(modelIt)
  }
```

- *launch* activates a new model instance.
- In this example, the instance is initialized with a reference to the current model and the iterator.

Tutorial, Step 2: Iterator model (1/2)

```
class IteratorModel(val dataModel: CollectionModel,
                    val it: Iterator[Integer]) extends Model {
  var pos = 0
  val version = dataModel.version

  def valid = (version == dataModel.version)

  def actualSize = dataModel.collection.size

  def hasNext {
    if (valid) {
      assert ((pos < actualSize) == it.hasNext)
    } else {
      it.hasNext
    }
  }

  def next {
    require (valid)
    require (pos < actualSize)
    it.next
    pos += 1
  }
```

Iterator model (2/2)

```
  def failingNext { // throws NoSuchElementException
    require (valid)
    require (pos >= actualSize)
    it.next
  }

  def concNext { // throws ConcurrentModificationException
    require(!valid)
    it.next
  }

  "main" -> "main" := hasNext
  "main" -> "main" := next
  "main" -> "main" := failingNext throws "NoSuchElementException"
  "main" -> "main" := concNext throws "ConcurrentModificationException"
}
```

- Preconditions determine when a given transition function is enabled.
- In this case, the preconditions distinguish normal behavior from exceptions.

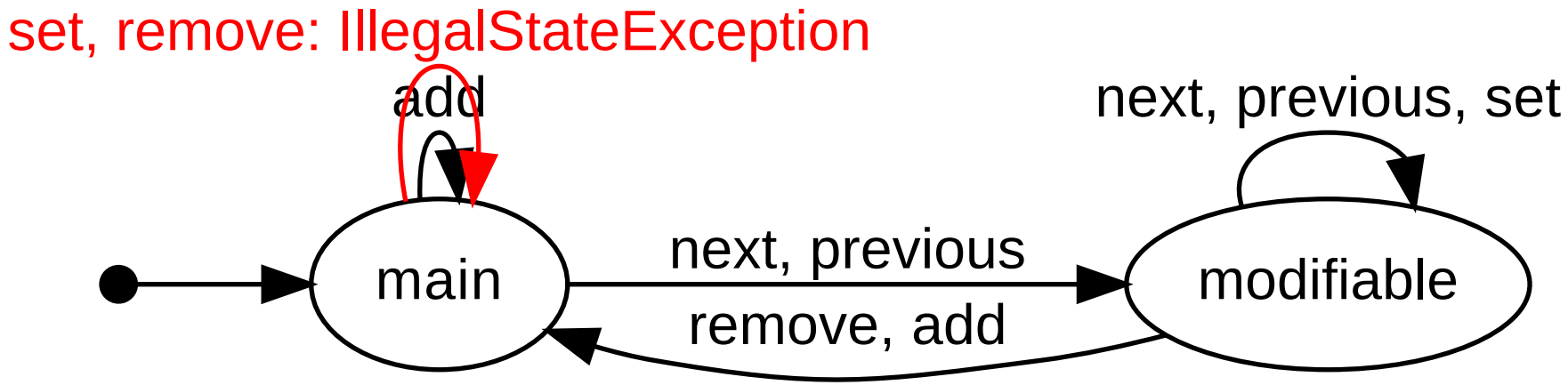
Test case generation with the example model

```
[INFO] 1000 tests executed, 997 ok, 3 failed.
[INFO] 2 types of test failures:
[INFO] 1) java.util.ConcurrentModificationException at failingNext:
[INFO]    6e8ddf360994ae26 36ae40ee3f8301d6
[INFO] 2) java.util.ConcurrentModificationException at next:
[INFO]    692927733240995
```

- Interpretation: ConcurrentModificationException is thrown by Java's iterator, but model does not expect it.
- Only 3 out of 1000 tests fail; only particular combinations of actions.
- Can you see a pattern and find the flaw in the model?
- Hint: You need to consider both the base model (CollectionModel) and the iterator model.

Tutorial, Step 3: Complex model: ListIterator

- Bidirectional Java iterator support **`next`** and **`previous`**.
- Supports **in-place modification and removal** of current element.
- Modification or removal is only possible if an element has been previously *selected* with **`next`** and **`previous`**:



ListIterator tutorial

Task 1: Fill in the missing code in „previous”! (Look at „next” for an example.)

Task 2: In `def concNext`, add another variant for the choice over functions (by writing another lambda expression).

Task 3: Add assertions in `def checkIdx` that validate the current position of the iterator.