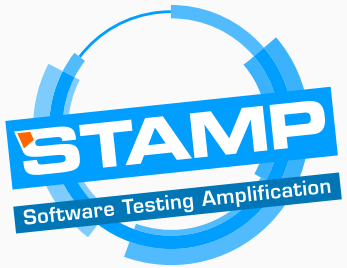
Spoon: Source Code Analysis And Transformation For Java

Benjamin DANGLOT

27th, June 2017

OW2Con'17



History

A library to write your own analysis and transformation in Java.

- 2005: project creation par R. Pawlak and N. Petitprez (INRIA Lille).
- 2008: Phd Thesis C. Noguera.
- 2014: Spoon on Github.
- 2016: Spoon at OW2.


351 stars

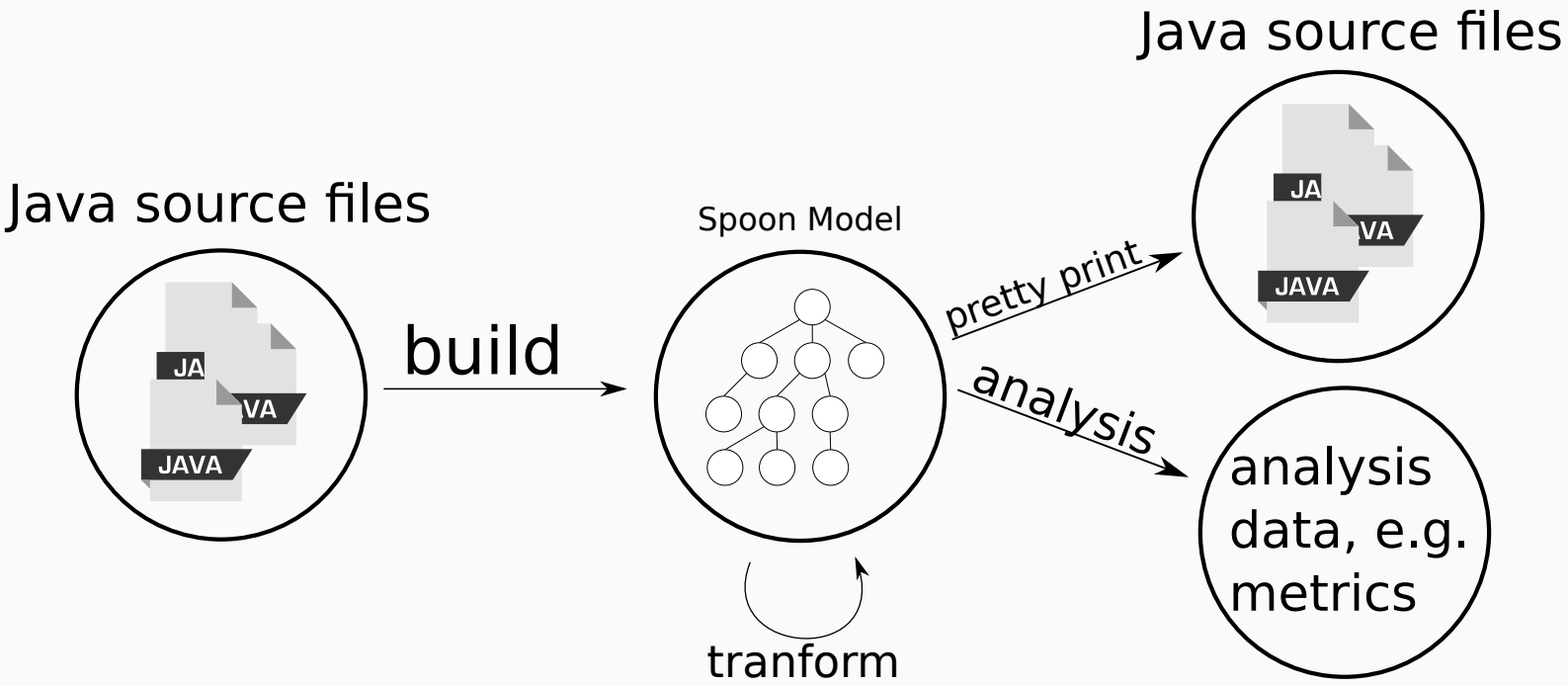

1895 commits since 2014-04-01


30 contributors


21 releases

Spoon:

A library to write your own analysis and transformation in java.



Usage examples:

- test improvement
- transpilation, e.g. Java to Javascript
- detection of bad smells
- automatic refactoring

Spoon:

A library to write your own analysis and transformation in java.

Spoon: Abstract example

A library to write your own analysis and transformation in java.

```
CtClass<?> tacos = factory.Class().get("fr.inria.Tacos")
```

Abstract example

A library to write your own analysis and transformation in java.

```
CtClass<?> tacos = factory.Class().get("fr.inria.Tacos")

tacos.getMethods().forEach(System.out::println);
```

Abstract example

A library to write your own analysis and transformation in java.

```
CtClass<?> tacos = factory.Class().get("fr.inria.Tacos");

tacos.getMethods().forEach(System.out::println);

CtMethod methodToBeRemoved =
        tacos.getMethodsByName( s: "removeOnions").get(0);
tacos.removeMethod(methodToBeRemoved);
```

Abstract example

A library to write your own analysis and transformation in java.

```
CtClass<?> tacos = factory.Class().get("fr.inria.Tacos")

tacos.getMethods().forEach(System.out::println);

CtMethod methodToBeRemoved =
        tacos.getMethodsByName( s: "removeOnions").get(0);
tacos.removeMethod(methodToBeRemoved);

CtField beef = factory.createField();
tacos.addField(beef);
```

Abstract example

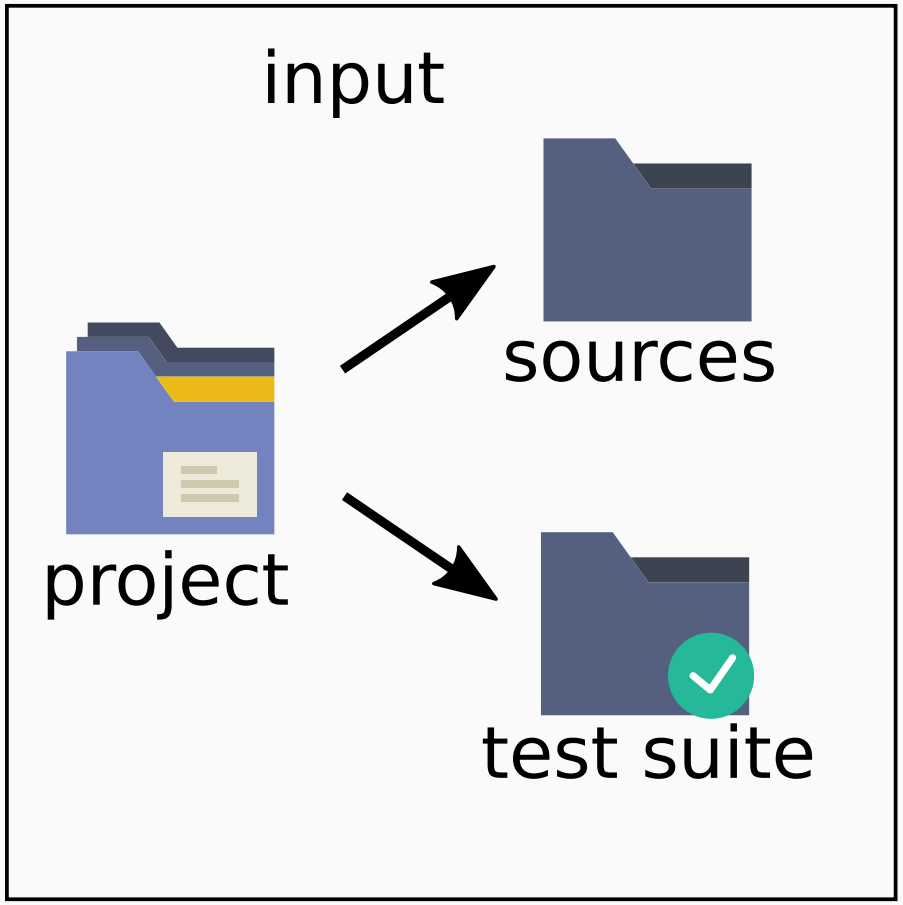
A library to write your own analysis and transformation in Java.

```
CtClass<?> tacos = factory.Class().get("fr.inria.Tacos")

tacos.getMethods().forEach(System.out::println);

CtMethod methodToBeRemoved =
        tacos.getMethodsByName( s: "removeOnions").get(0);
tacos.removeMethod(methodToBeRemoved);

CtField beef = factory.createField();
tacos.addField(beef);
```

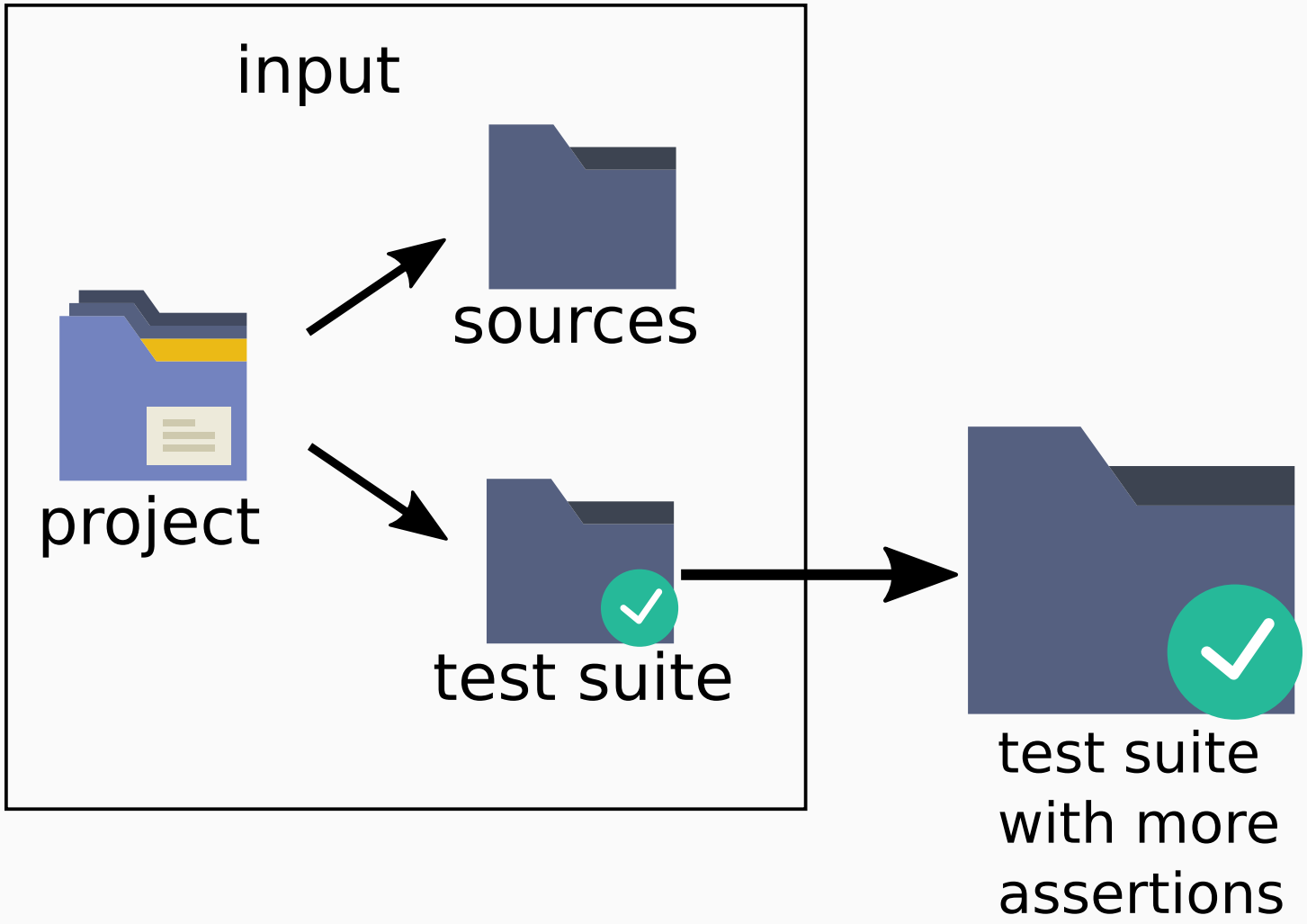
Related to: Javapoet, JavaParser and ASM.

Concrete Example

Concrete Example



Concrete Example



Concrete Example

Goal: generate all assertions for a test method.

Concrete Example

Goal: generate all assertions for a test method.

```
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
}
```

Concrete Example

Goal: generate all assertions for a test method.

```
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
}
```

Concrete Example

Goal: generate all assertions for a test method.

```
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
}
```

Concrete Example

```java
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
}
```

Improve the test with 2 new assertions

```java
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
    assertEquals( expected: true, benjamin.isHungry());
    assertEquals( expected: "All code is guilty until proven innocent.",
            benjamin.getQuote());
}
```

Workflow

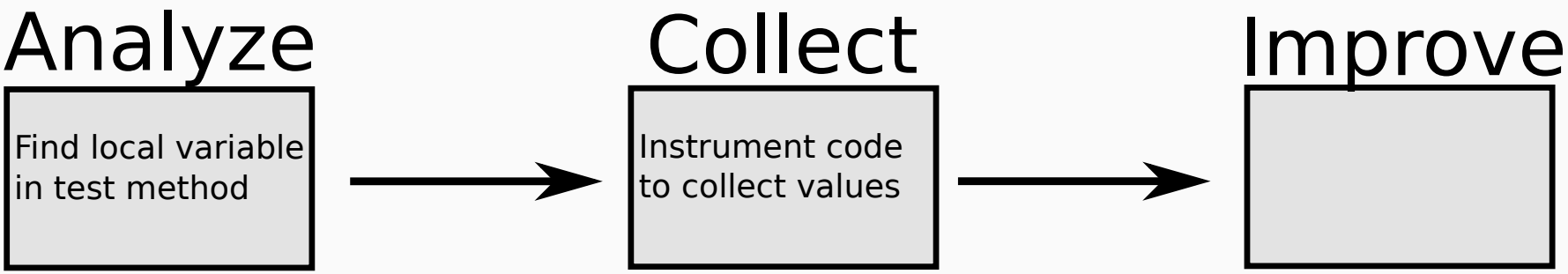
Goal: generate all assertions for a test method.

Workflow

Goal: generate all assertions for a test method.

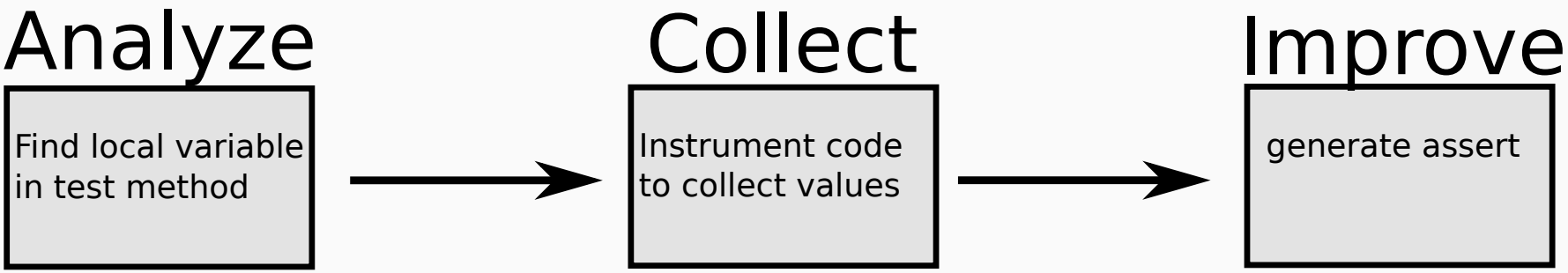
Workflow

Goal: generate all assertions for a test method.



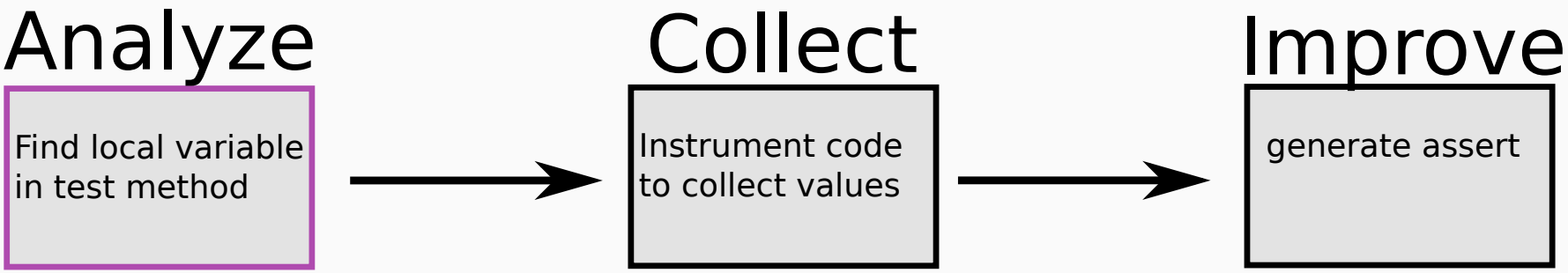
Workflow

Goal: generate all assertions for a test method.



Step 1: Analyze

Find the local variable in the test.



Analyze: goal

Find the local variable in the test.

```
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
     assertEquals( expected: 24, benjamin.age());
}
```

Analyze: code

Find the local variable in the test.

```
List<CtLocalVariable> analyze(CtMethod testMethod) {
    TypeFilter filterLocalVar =
            new TypeFilter(CtLocalVariable.class) {
                public boolean matches(CtLocalVariable localVariable) {
                        return !localVariable.getType().isPrimitive();
                    }
            };
    return testMethod.getElements(filterLocalVar);
}
```

Analyze: code

Find the local variable in the test.

```
List<CtLocalVariable> analyze(CtMethod testMethod) {
    TypeFilter filterLocalVar =
            new TypeFilter(CtLocalVariable.class) {
                public boolean matches(CtLocalVariable localVariable) {
                        return !localVariable.getType().isPrimitive();
                    }
            };
    return testMethod.getElements(filterLocalVar);
}
```

Analyze: code

Find the local variable in the test.

```
List<CtLocalVariable> analyze(CtMethod testMethod) {
    TypeFilter filterLocalVar =
            new TypeFilter(CtLocalVariable.class) {
                public boolean matches(CtLocalVariable localVariable) {
                        return !localVariable.getType().isPrimitive();
                    }
            };
    return testMethod.getElements(filterLocalVar);
}
```

Analyze: code

Find the local variable in the test.

```
List<CtLocalVariable> analyze(CtMethod testMethod) {
    TypeFilter filterLocalVar =
            new TypeFilter(CtLocalVariable.class) {
                public boolean matches(CtLocalVariable localVariable) {
                        return !localVariable.getType().isPrimitive();
                    }
            };
    return testMethod.getElements(filterLocalVar);
}
```

Analyze: code

Find the local variable in the test.

```
List<CtLocalVariable> analyze(CtMethod testMethod) {
    TypeFilter filterLocalVar =
            new TypeFilter(CtLocalVariable.class) {
                public boolean matches(CtLocalVariable localVariable) {
                    return !localVariable.getType().isPrimitive();
                }
            };
    return testMethod.getElements(filterLocalVar);
}
```

Analyze: code

Find the local variable in the test.

```
List<CtLocalVariable> analyze(CtMethod testMethod) {
    TypeFilter filterLocalVar =
            new TypeFilter(CtLocalVariable.class) {
                public boolean matches(CtLocalVariable localVariable) {
                        return !localVariable.getType().isPrimitive();
                    }
            };
    return testMethod.getElements(filterLocalVar);
}
```

Analyze: code

Find the local variable in the test.

```
List<CtLocalVariable> analyze(CtMethod testMethod) {
    TypeFilter filterLocalVar =
            new TypeFilter(CtLocalVariable.class) {
                public boolean matches(CtLocalVariable localVariable) {
                        return !localVariable.getType().isPrimitive();
                    }
            };
    return testMethod.getElements(filterLocalVar);
}
```

Analyze: output

Local variable in the test found.

```
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
     assertEquals( expected: 24, benjamin.age());
}
```

Step 2: Collect

Collect values of accessible fields of the local variable.

Collect: goal

Collect values of accessible fields of the local variable.

```java
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
}
```

Instrumentation

```java
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
    Logger.observe( name: "Benjamin#isHungry", benjamin.isHungry());
    Logger.observe( name: "Benjamin#getQuote", benjamin.getQuote());
}
```

Collect: code

Collect values of accessible fields of the local variable.

```
void instrument(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToObserve =
                createObserve(getter, invocationToGetter);
        testMethod.getBody().insertEnd(invocationToObserve);
    });
}
```

Collect: code

Collect values of accessible fields of the local variable.

```
void instrument(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToObserve =
                createObserve(getter, invocationToGetter);
        testMethod.getBody().insertEnd(invocationToObserve);
    });
}
```

Collect: code

Collect values of accessible fields of the local variable.

```
void instrument(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToObserve =
                createObserve(getter, invocationToGetter);
        testMethod.getBody().insertEnd(invocationToObserve);
    });
}
```

Collect: code

Collect values of accessible fields of the local variable.

```
void instrument(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToObserve =
                createObserve(getter, invocationToGetter);
        testMethod.getBody().insertEnd(invocationToObserve);
    });
}
```

Collect: code

Collect values of accessible fields of the local variable.

```
void instrument(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToObserve =
                createObserve(getter, invocationToGetter);
        testMethod.getBody().insertEnd(invocationToObserve);
    });
}
```

Collect: code

Collect values of accessible fields of the local variable.

```
void instrument(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToObserve =
                createObserve(getter, invocationToGetter);
        testMethod.getBody().insertEnd(invocationToObserve);
    });
}
```

Collect: code

Collect values of accessible fields of the local variable.

```
void instrument(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToObserve =
                createObserve(getter, invocationToGetter);
        testMethod.getBody().insertEnd(invocationToObserve);
    });
}
```

Collect: output

Instrumentaiton to collect values of the local variable.

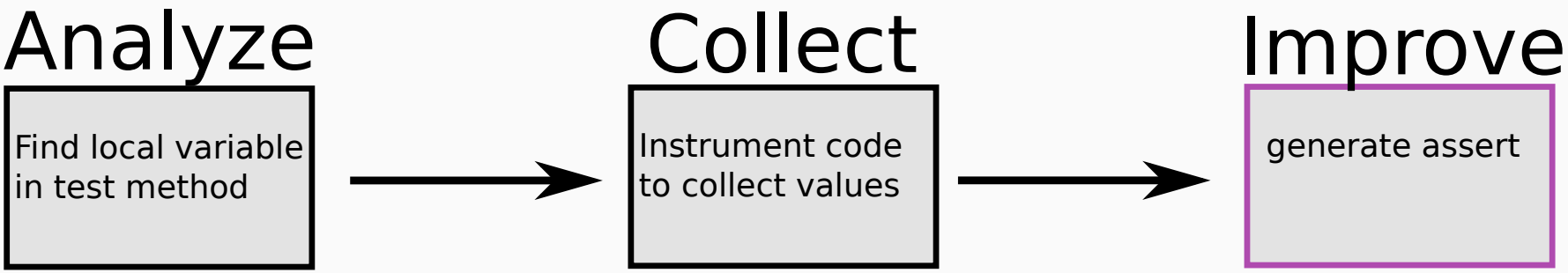
```java
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
}
```

Instrumentation

```java
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
    Logger.observe( name: "Benjamin#isHungry", benjamin.isHungry());
    Logger.observe( name: "Benjamin#getQuote", benjamin.getQuote());
}
```

Step 3: Improve

Add assertion on accessible fields of the local variable.



Improve: goal

Add assertion on accessible fields of the local variable.

```java
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
    Logger.observe( name: "Benjamin#isHungry", benjamin.isHungry());
    Logger.observe( name: "Benjamin#getQuote", benjamin.getQuote());
}
```

Generation

Generation

```java
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
    assertEquals( expected: true, benjamin.isHungry());
    assertEquals( expected: "All code is guilty until proven innocent.",
            benjamin.getQuote());
}
```

Improve: code

Add assertion on accessible fields of the local variable.

```
void addAssertion(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        String key = getKey(getter);
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToAssert = createAssert( name: "assertEquals",
                factory.createLiteral(Logger.observations.get(key)), // expected
                invocationToGetter); // actual
        testMethod.getBody().insertEnd(invocationToAssert);
    });
}
```

Improve: code

Add assertion on accessible fields of the local variable.

```
void addAssertion(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        String key = getKey(getter);
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToAssert = createAssert( name: "assertEquals",
                factory.createLiteral(Logger.observations.get(key)), // expected
                invocationToGetter); // actual
        testMethod.getBody().insertEnd(invocationToAssert);
    });
}
```

Improve: code

Add assertion on accessible fields of the local variable.

```
void addAssertion(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        String key = getKey(getter);
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToAssert = createAssert( name: "assertEquals",
                factory.createLiteral(Logger.observations.get(key)), // expected
                invocationToGetter); // actual
        testMethod.getBody().insertEnd(invocationToAssert);
    });
}
```

Improve: code

Add assertion on accessible fields of the local variable.

```
void addAssertion(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        String key = getKey(getter);
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToAssert = createAssert( name: "assertEquals",
                factory.createLiteral(Logger.observations.get(key)), // expected
                invocationToGetter); // actual
        testMethod.getBody().insertEnd(invocationToAssert);
    });
}
```

Improve: code

Add assertion on accessible fields of the local variable.

```
void addAssertion(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        String key = getKey(getter);
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToAssert = createAssert( name: "assertEquals",
                factory.createLiteral(Logger.observations.get(key)), // expected
                invocationToGetter); // actual
        testMethod.getBody().insertEnd(invocationToAssert);
    });
}
```

Improve: code

Add assertion on accessible fields of the local variable.

```
void addAssertion(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        String key = getKey(getter);
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToAssert = createAssert( name: "assertEquals",
                factory.createLiteral(Logger.observations.get(key)), // expected
                invocationToGetter); // actual
        testMethod.getBody().insertEnd(invocationToAssert);
    });
}
```

Improve: code

Add assertion on accessible fields of the local variable.

```
void addAssertion(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        String key = getKey(getter);
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToAssert = createAssert( name: "assertEquals",
                factory.createLiteral(Logger.observations.get(key)), // expected
                invocationToGetter); // actual
        testMethod.getBody().insertEnd(invocationToAssert);
    });
}
```

Improve: code

Add assertion on accessible fields of the local variable.

```
void addAssertion(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        String key = getKey(getter);
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToAssert = createAssert( name: "assertEquals",
                factory.createLiteral(Logger.observations.get(key)), // expected
                invocationToGetter); // actual
        testMethod.getBody().insertEnd(invocationToAssert);
    });
}
```

Improve: code

Add assertion on accessible fields of the local variable.

```
void addAssertion(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        String key = getKey(getter);
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToAssert = createAssert( name: "assertEquals",
                factory.createLiteral(Logger.observations.get(key)), // expected
                invocationToGetter); // actual
        testMethod.getBody().insertEnd(invocationToAssert);
    });
}
```

Improve: code

Add assertion on accessible fields of the local variable.

```
void addAssertion(CtMethod testMethod, CtLocalVariable localVariable) {
    List<CtMethod> getters = getGetters(localVariable);
    getters.forEach(getter -> {
        String key = getKey(getter);
        CtInvocation invocationToGetter =
                invok(getter, localVariable);
        CtInvocation invocationToAssert = createAssert( name: "assertEquals",
                factory.createLiteral(Logger.observations.get(key)), // expected
                invocationToGetter); // actual
        testMethod.getBody().insertEnd(invocationToAssert);
    });
}
```

Improve: output

Test method improved with two new assertions

```java
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
    Logger.observe( name: "Benjamin#isHungry", benjamin.isHungry());
    Logger.observe( name: "Benjamin#getQuote", benjamin.getQuote());
}
```

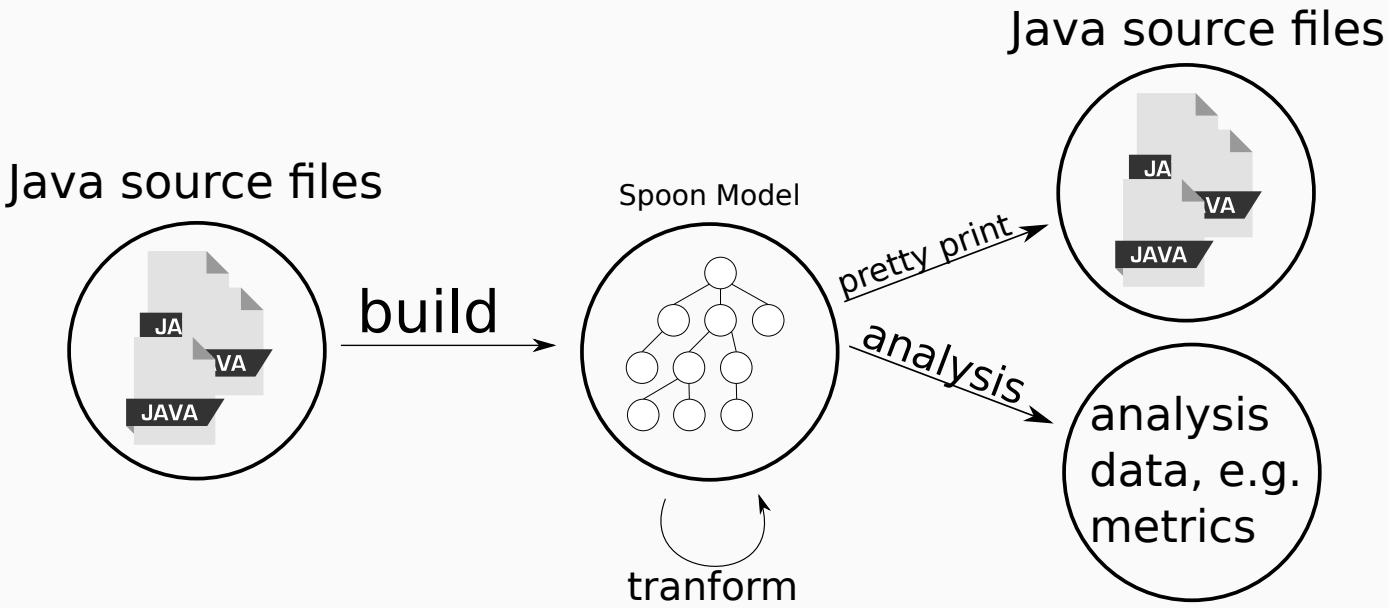
Generation

Generation

```java
@Test
public void doYouKnowBenjamin() throws Exception {
    Benjamin benjamin = new Benjamin();
    assertEquals( expected: 24, benjamin.age());
    assertEquals( expected: true, benjamin.isHungry());
    assertEquals( expected: "All code is guilty until proven innocent.",
            benjamin.getQuote());
}
```

Spoon:

A library to write your own analysis and transformation in java.



Usage examples:

- test improvement
- transpilation, e.g. Java to Javascript
- detection of bad smells
- automatic refactoring

demo avalaible:

`http://github.com/SpoonLabs/spoon-examples.git`