DSpot-prettifier: how to make your test more readable?

Benjamin DANGLOT

January 31st, 2019

benjamin.danglot@inria.fr

STAMP Workshop INRIA - Sophia Antipolis

Motivation: Amplified test methods are unreadable and ugly.

Motivation: Amplified test methods are unreadable and ugly.

Unclear test method name.

```
@Test(timeout = 30000)
public void testNoSolutionfound_add2386_failAssert238litNum2407_failAssert240()
```

Motivation: Amplified test methods are unreadable and ugly.

Unclear test method name.

```
@Test(timeout = 30000)
public void testNoSolutionfound_add2386_failAssert238litNum2407_failAssert240()
```

Weird variable names.

```
int[] array_1848848534 = new int[]{199,201};
int[] array_1904675524 = (int[])((org.sat4j.minisat.constraints.cnf.OriginalBinaryClause)
        o_testNoSolutionExists_remove6__12)
        .getLits();
for(int ii = 0; ii <array_1848848534.length; ii++) {
    org.junit.Assert.assertEquals(array_1848848534[ii], array_1904675524[ii]);
};
```

Motivation: Amplified test methods are unreadable and ugly.

Unclear test method name.

```
@Test(timeout = 30000)
public void testNoSolutionfound_add2386_failAssert238litNum2407_failAssert240()
```

Weird variable names.

```
int[] array_1848848534 = new int[]{199,201};
int[] array_1904675524 = (int[])((org.sat4j.minisat.constraints.cnf.OriginalBinaryClause)
        o_testNoSolutionExists_remove6__12)
        .getLits();
for(int ii = 0; ii <array_1848848534.length; ii++) {
    org.junit.Assert.assertEquals(array_1848848534[ii], array_1904675524[ii]);
};
```

Duplicated and useless statements.

```
clause.clear();
clause.clear();
clause.clear();
clause.clear();
IVecInt o_testNoSolutionExists_remove6__8 = clause.push((-99));
clause.clear();
clause.clear();
Assert.assertEquals( expected: "-99", ((VecInt) (o_testNoSolutionExists_remove6__8)).toString());
```

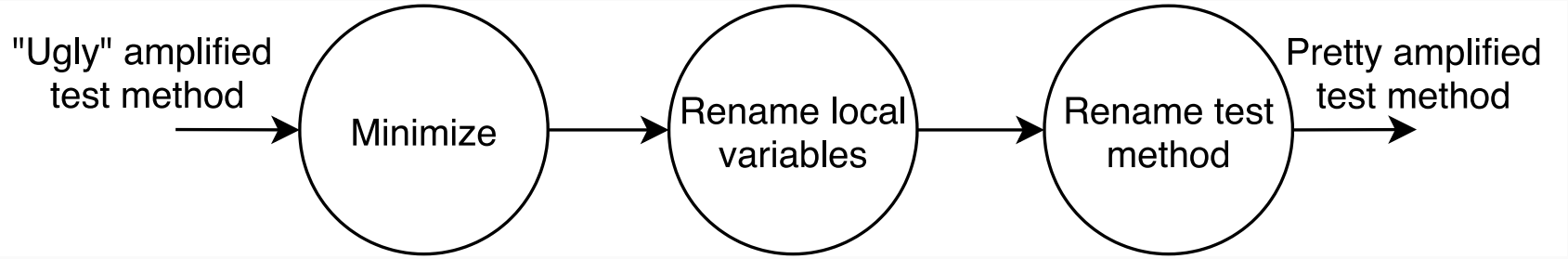
Motivation: Amplified test methods are unreadable and ugly.

Too long.

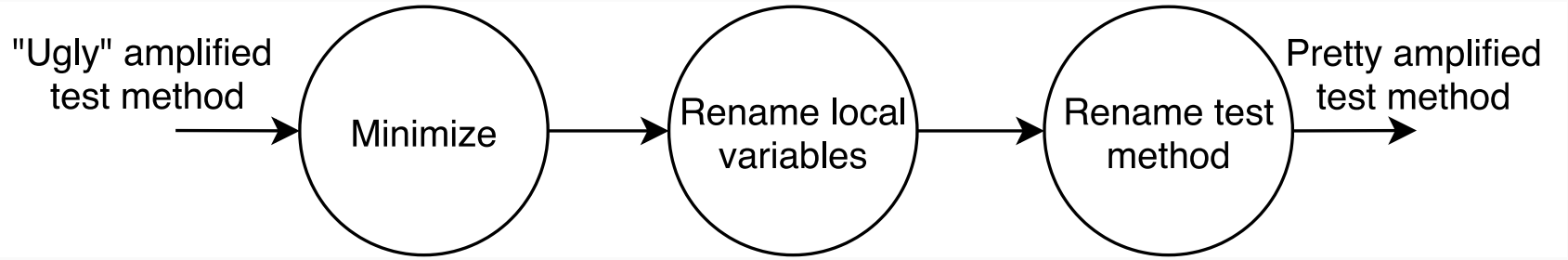
```
@Test(timeout = 30000)
public void testNoSolutionExists_remove6() throws Exception, ContradictionException, TimeoutException {
    IVecInt clause = new VecInt();
    Assert.assertEquals( expected: "", ((VecInt) (clause)).toString());
    Assert.assertTrue(((VecInt) (clause)).isEmpty());
    IVecInt o_testNoSolutionExists_remove6__3 = clause.push(100).push(99);
    Assert.assertEquals( expected: "100,99", ((VecInt) (o_testNoSolutionExists_remove6__3)).toString());
    Assert.assertEquals( expected: 99, ((int) (((VecInt) (o_testNoSolutionExists_remove6__3)).hashCode())));
    Assert.assertFalse(((VecInt) (o_testNoSolutionExists_remove6__3)).isEmpty());
    IConstr o_testNoSolutionExists_remove6__5 = solver.addClause(clause);
    Assert.assertEquals( expected: "99[?] 100[?]", ((OriginalBinaryClause) (o_testNoSolutionExists_remove6__5)).toString());
    Assert.assertEquals( expected: 199, ((int) (((OriginalBinaryClause) (o_testNoSolutionExists_remove6__5)).hashCode())));
    Assert.assertEquals( expected: 0.0, ((double) (((OriginalBinaryClause) (o_testNoSolutionExists_remove6__5)).getActivity())), delta: 0.1);
    Assert.assertFalse(((OriginalBinaryClause) (o_testNoSolutionExists_remove6__5)).isSatisfied());
    int[] array_1780117107 = new int[]{198,200};
    int[] array_1918579922 = (int[])((org.sat4j.minisat.constraints.cnf.OriginalBinaryClause)o_testNoSolutionExists_remove6__5).getLits();
    for(int ii = 0; ii <array_1780117107.length; ii++) {
        org.junit.Assert.assertEquals(array_1780117107[ii], array_1918579922[ii]);
    };
    IConstr o_testNoSolutionExists_remove6__6 = solver.addClause(clause);
    Assert.assertEquals( expected: "99[?] 100[?]", ((OriginalBinaryClause) (o_testNoSolutionExists_remove6__6)).toString());
    Assert.assertEquals( expected: 199, ((int) (((OriginalBinaryClause) (o_testNoSolutionExists_remove6__6)).hashCode())));
    Assert.assertEquals( expected: 0.0, ((double) (((OriginalBinaryClause) (o_testNoSolutionExists_remove6__6)).getActivity())), delta: 0.1);
    Assert.assertFalse(((OriginalBinaryClause) (o_testNoSolutionExists_remove6__6)).isSatisfied());
    int[] array_582027917 = new int[]{198,200};
    int[] array_2113184556 = (int[])((org.sat4j.minisat.constraints.cnf.OriginalBinaryClause)o_testNoSolutionExists_remove6__6).getLits();
    for(int ii = 0; ii <array_582027917.length; ii++) {
        org.junit.Assert.assertEquals(array_582027917[ii], array_2113184556[ii]);
    };
    clause.clear();
    clause.clear();
    IVecInt o_testNoSolutionExists_remove6__8 = clause.push((-99));
    Assert.assertEquals( expected: "-99", ((VecInt) (o_testNoSolutionExists_remove6__8)).toString());
    Assert.assertEquals( expected: -99, ((int) (((VecInt) (o_testNoSolutionExists_remove6__8)).hashCode())));
    Assert.assertFalse(((VecInt) (o_testNoSolutionExists_remove6__8)).isEmpty());
    IVecInt o_testNoSolutionExists_remove6__10 = clause.push((-100));
    Assert.assertEquals( expected: "-99,-100", ((VecInt) (o_testNoSolutionExists_remove6__10)).toString());
    Assert.assertEquals( expected: -99, ((int) (((VecInt) (o_testNoSolutionExists_remove6__10)).hashCode())));
    Assert.assertFalse(((VecInt) (o_testNoSolutionExists_remove6__10)).isEmpty());
    IConstr o_testNoSolutionExists_remove6__12 = solver.addClause(clause);
    Assert.assertEquals( expected: "-99[?] -100[?]", ((OriginalBinaryClause) (o_testNoSolutionExists_remove6__12)).toString());
    Assert.assertEquals( expected: 200, ((int) (((OriginalBinaryClause) (o_testNoSolutionExists_remove6__12)).hashCode())));
    Assert.assertEquals( expected: 0.0, ((double) (((OriginalBinaryClause) (o_testNoSolutionExists_remove6__12)).getActivity())), delta: 0.1);
    Assert.assertFalse(((OriginalBinaryClause) (o_testNoSolutionExists_remove6__12)).isSatisfied());
    int[] array_1848848534 = new int[]{199,201};
```

DSpot-prettifier

Approach: Overview

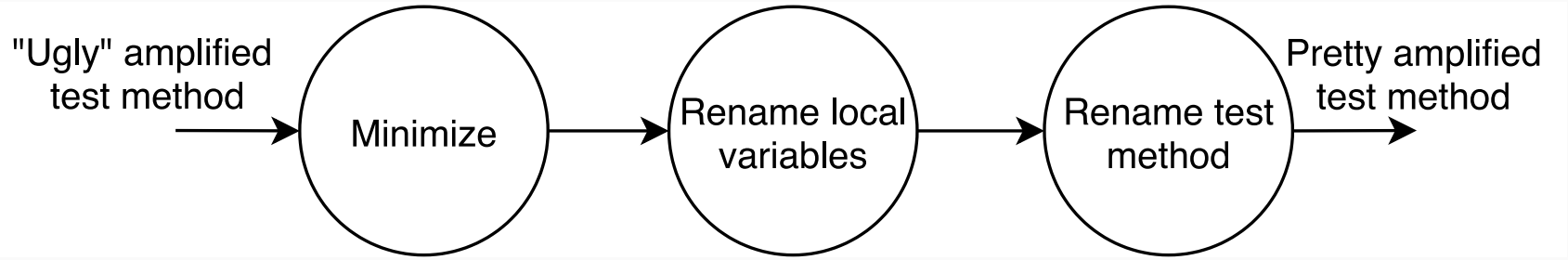


Approach: 1. Minimization



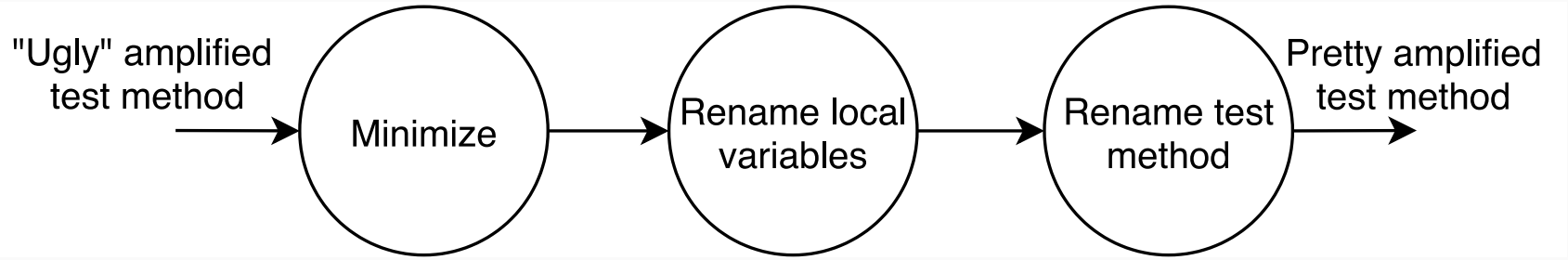
1. Minimize statements according to a specific test criterion

Approach: 2. Rename local variables



1. Minimize statements according to a specific test criterion
2. Rename local variables according to their context

Approach: 3. Rename test method



1. Minimize statements according to a specific test criterion
2. Rename local variables according to their context
3. Rename the test method according to its code

Minimize test methods

Minimize test methods

Replace multiple calls by local variables.

```
Assert.assertEquals( expected: 9, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getMonth());
Assert.assertEquals( expected: 12, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getHours());
Assert.assertEquals( expected: 0, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getMinutes());
Assert.assertEquals( expected: 0, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getSeconds());
```

Minimize test methods

Replace multiple calls by local variables.

```
Assert.assertEquals( expected: 9, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getMonth());
Assert.assertEquals( expected: 12, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getHours());
Assert.assertEquals( expected: 0, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getMinutes());
Assert.assertEquals( expected: 0, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getSeconds());
```

Minimize test methods

Replace multiple calls by local variables.

```
Assert.assertEquals( expected: 9, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getMonth());
Assert.assertEquals( expected: 12, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getHours());
Assert.assertEquals( expected: 0, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getMinutes());
Assert.assertEquals( expected: 0, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getSeconds());
```

```
final Date notBefore = ((BcX509CertifiedPublicKey) certificate).getNotBefore();
Assert.assertEquals( expected: 9, (int) notBefore.getMonth());
Assert.assertEquals( expected: 12, (int) notBefore.getHours());
Assert.assertEquals( expected: 0, (int) notBefore.getMinutes());
Assert.assertEquals( expected: 0, (int) notBefore.getSeconds());
```

Minimize test methods

Replace multiple calls by local variables.

```
Assert.assertEquals( expected: 9, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getMonth());
Assert.assertEquals( expected: 12, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getHours());
Assert.assertEquals( expected: 0, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getMinutes());
Assert.assertEquals( expected: 0, (int)
        ((BcX509CertifiedPublicKey) certificate).getNotBefore().getSeconds());
```

```
final Date notBefore = ((BcX509CertifiedPublicKey) certificate).getNotBefore();
Assert.assertEquals( expected: 9, (int) notBefore.getMonth());
Assert.assertEquals( expected: 12, (int) notBefore.getHours());
Assert.assertEquals( expected: 0, (int) notBefore.getMinutes());
Assert.assertEquals( expected: 0, (int) notBefore.getSeconds());
```

Minimize test methods

Replace local variable used once by direct method call.

```
@Test(timeout = 10000)
public void nameCollision_mg18423_mg18592() throws Exception {
    String __DSPOT_suggestion_4985 = "Vb,:r;9h)/U)b_+xS|s#";
    Object __DSPOT_tag_4967 = new Object();
    String __DSPOT_suggestion_4966 = "I `C)%1^#[3d&Ya@Us[d";
    NameAllocator nameAllocator = new NameAllocator();
    String o_nameCollision_mg18423__6 = nameAllocator.newName(__DSPOT_suggestion_4966, __DSPOT_tag_4967);
    Assert.assertEquals("I__C__1___3d_Ya_Us_d", o_nameCollision_mg18423__6);
    String o_nameCollision_mg18423_mg18592__10 = nameAllocator.newName(__DSPOT_suggestion_4985);
    Assert.assertEquals("Vb__r_9h__U_b__xS_s_", o_nameCollision_mg18423_mg18592__10);
    Assert.assertEquals("I__C__1___3d_Ya_Us_d", o_nameCollision_mg18423__6);
}
```

Minimize test methods

Replace local variable used once by direct method call.

```
@Test(timeout = 10000)
public void nameCollision_mg18423_mg18592() throws Exception {
    String __DSPOT_suggestion_4985 = "Vb,:r;9h)/U)b +xS|s#";
    Object __DSPOT_tag_4967 = new Object();
    String __DSPOT_suggestion_4966 = "I `C)%1^#[3d&Ya@Us[d";
    NameAllocator nameAllocator = new NameAllocator();
    String o_nameCollision_mg18423__6 = nameAllocator.newName(__DSPOT_suggestion_4966, __DSPOT_tag_4967);
    Assert.assertEquals("I__C__1___3d_Ya_Us_d", o_nameCollision_mg18423__6);
    String o_nameCollision_mg18423_mg18592__10 = nameAllocator.newName(__DSPOT_suggestion_4985);
    Assert.assertEquals("Vb__r_9h__U_b__xS_s_", o_nameCollision_mg18423_mg18592__10);
    Assert.assertEquals("I__C__1___3d_Ya_Us_d", o_nameCollision_mg18423__6);
}
```

Minimize test methods

Replace local variable used once by direct method call.

```
@Test(timeout = 10000)
public void nameCollision_mg18423_mg18592() throws Exception {
    String __DSPOT_suggestion_4985 = "Vb,:r;9h)/U)b +xS|s#";
    Object __DSPOT_tag_4967 = new Object();
    String __DSPOT_suggestion_4966 = "I `C)%1^#[3d&Ya@Us[d";
    NameAllocator nameAllocator = new NameAllocator();
    String o_nameCollision_mg18423__6 = nameAllocator.newName(__DSPOT_suggestion_4966, __DSPOT_tag_4967);
    Assert.assertEquals("I__C__1___3d_Ya_Us_d", o_nameCollision_mg18423__6);
    String o_nameCollision_mg18423_mg18592__10 = nameAllocator.newName(__DSPOT_suggestion_4985);
    Assert.assertEquals("Vb__r_9h__U_b__xS_s_", o_nameCollision_mg18423_mg18592__10);
    Assert.assertEquals("I__C__1___3d_Ya_Us_d", o_nameCollision_mg18423__6);
}
```

```
@Test(timeout = 10000)
public void nameCollision_mg18423_mg18592() throws Exception {
    Object __DSPOT_tag_4967 = new Object();
    String __DSPOT_suggestion_4966 = "I `C)%1^#[3d&Ya@Us[d";
    NameAllocator nameAllocator = new NameAllocator();
    String o_nameCollision_mg18423__6 = nameAllocator.newName(__DSPOT_suggestion_4966, __DSPOT_tag_4967);
    Assert.assertEquals("I__C__1___3d_Ya_Us_d", o_nameCollision_mg18423__6);
    String o_nameCollision_mg18423_mg18592__10 = nameAllocator.newName("Vb,:r;9h)/U)b +xS|s#");
    Assert.assertEquals("Vb__r_9h__U_b__xS_s_", o_nameCollision_mg18423_mg18592__10);
    Assert.assertEquals("I__C__1___3d_Ya_Us_d", o_nameCollision_mg18423__6);
}
```

Minimize: redundant invokations

Remove redundant assertions

```
Assert.assertEquals( expected: 0.0, ((double) (((UnitClause)
        (o_testNoSolutionExists__9)).getActivity())), delta: 0.1);
Assert.assertEquals( expected: "-99", ((UnitClause) (o_testNoSolutionExists__9)).toString());
// ... assertions
Assert.assertTrue(((UnitClause) (o_testNoSolutionExists__9)).isSatisfied());
Assert.assertEquals( expected: 0.0, ((double) (((UnitClause)
        (o_testNoSolutionExists__9)).getActivity())), delta: 0.1);
Assert.assertEquals( expected: "-99", ((UnitClause) (o_testNoSolutionExists__9)).toString());
```

Minimize: redundant assertions

Remove redundant assertions

```
Assert.assertEquals( expected: 0.0, ((double) (((UnitClause)
        (o_testNoSolutionExists__9)).getActivity())), delta: 0.1);
Assert.assertEquals( expected: "-99", ((UnitClause) (o_testNoSolutionExists__9)).toString());
// ... assertions
Assert.assertTrue(((UnitClause) (o_testNoSolutionExists__9)).isSatisfied());
Assert.assertEquals( expected: 0.0, ((double) (((UnitClause)
        (o_testNoSolutionExists__9)).getActivity())), delta: 0.1);
Assert.assertEquals( expected: "-99", ((UnitClause) (o_testNoSolutionExists__9)).toString());
```

Minimize: search-based algorithm

Apply a search-based algorithm to minimize test methods

Minimize: search-based algorithm

Apply a search-based algorithm to minimize test methods

```
List<Method> minimized_list = Collections.singletonList(testMethod);
for (Method test : minimized_list) {
    List<Statement> statements = test.getStatements();
    for (Statement statement : statements) {
        statements.remove(statement);
        Method method = new Method(statements);
        if (testCriterion.isOk(method)) {
            minimized_list.add(method);
        }
    }
}
return findShortest(minimized_list);
```

Rename local variable

Look to Context2Name,

- unminified javascript code
- found a name for variable from the context, i.e. the code.
- TODO: create a prototype for java, if applicable

Rename test method

Code2vec is based on a artificial neural network.

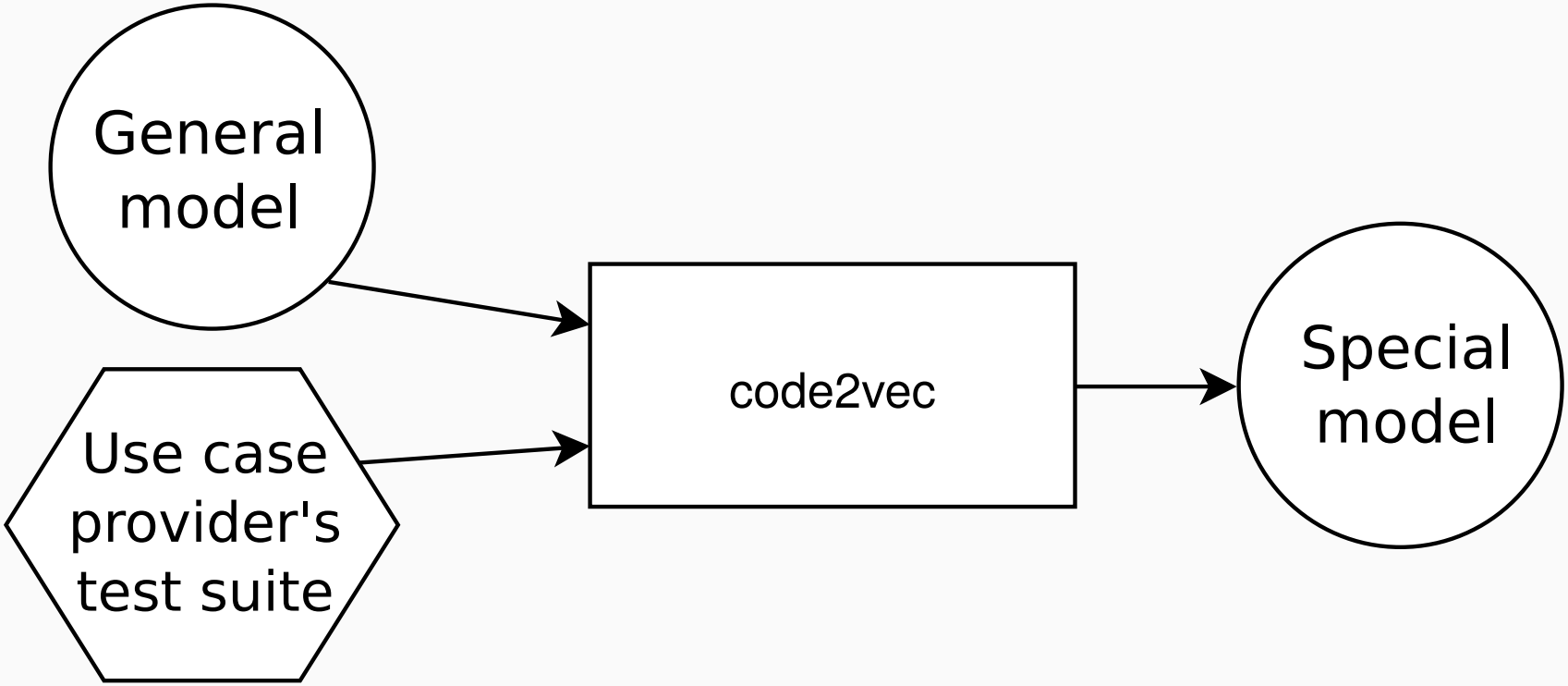
- Train a model with java code
- Predict label from a sequence of statements

Rename test method

First, training a general model using a lot of test methods from GitHub.

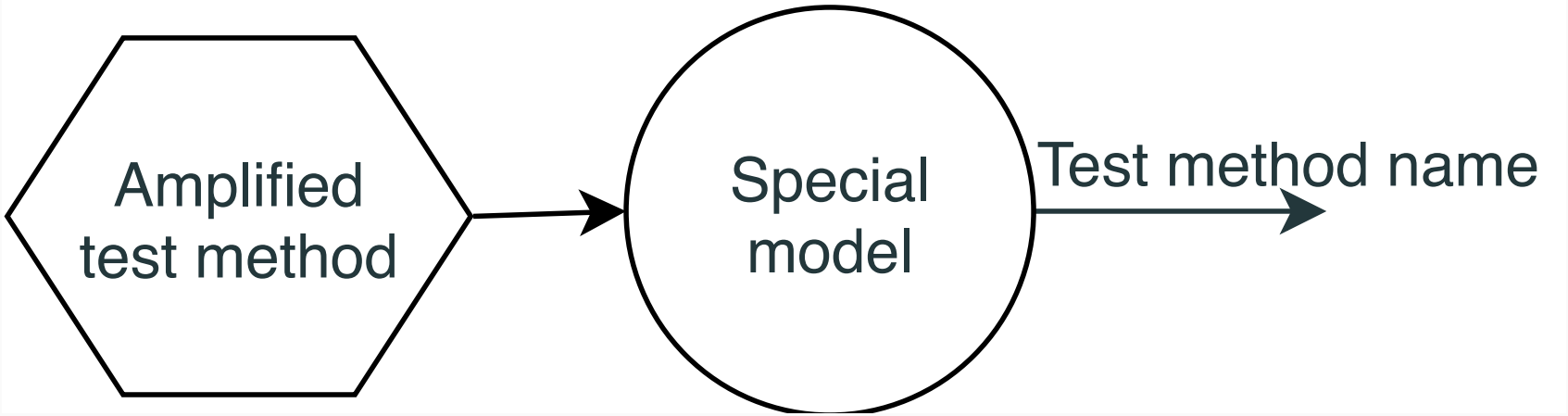
Rename test method

Then, specialize the model with the test suite of a use case provider.



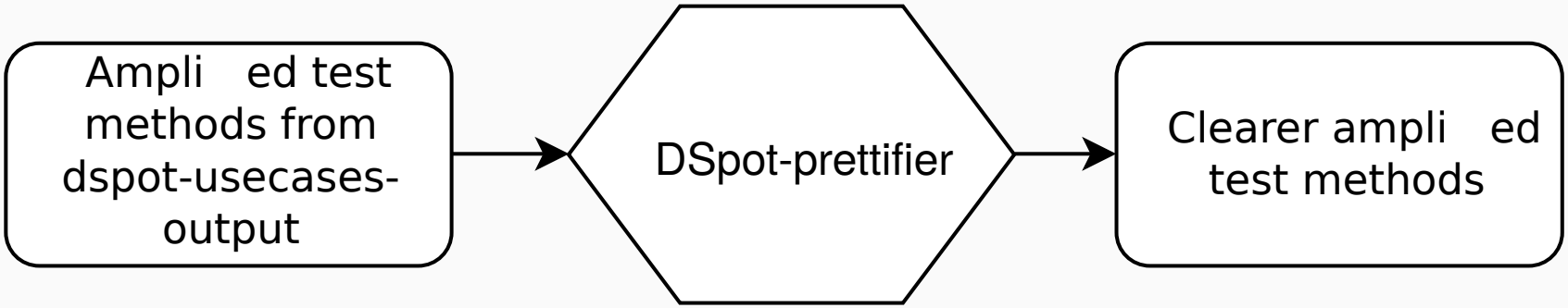
Rename test method

Predict the name of the amplified test method.

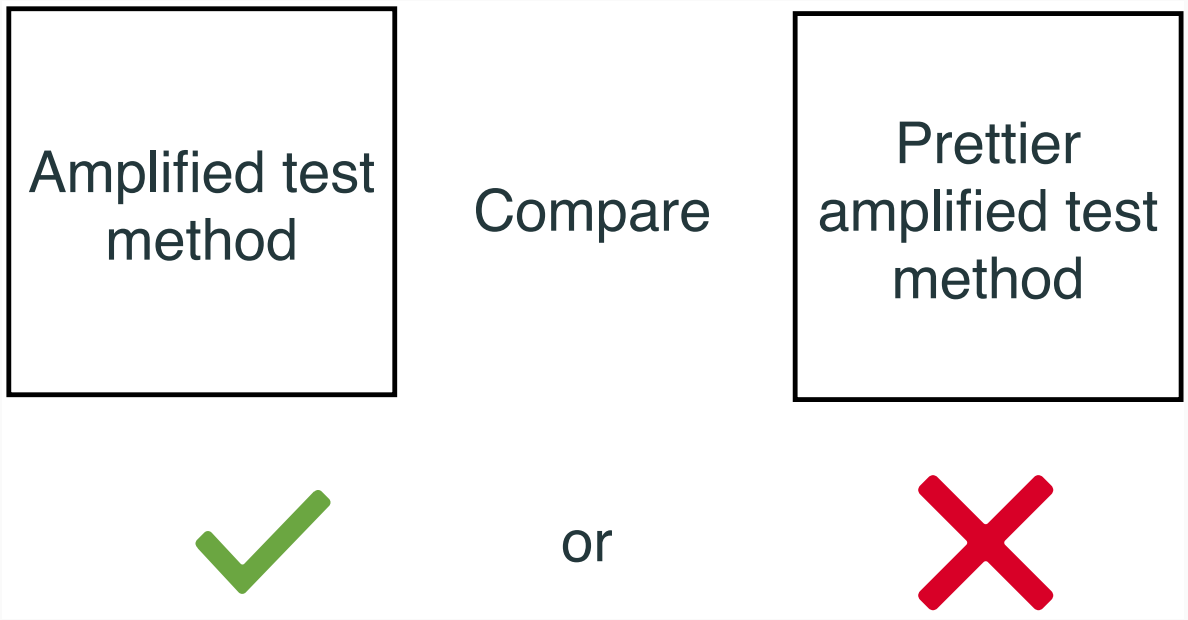


Evaluation

For each use case provider and result from *dspot-usescases-output*.



Evaluation: Comparison



To tell if the DSpot-prettifier's output is relevant or miss information.

Evaluation's protocol

- 2 different developers / UC
- 2 sessions: one open-source, one dedicated to the UC
- 15 tests per session
- Maximum 2 minutes per test method should be granted
- $2 \times 30m = 1hour$ per developper

Important: your evaluation is crucial for the paper to validate or not the approach.

We plan to submit a special issue of *The Programming Journal*, **1st June 2019**.