# Automated Unit Test Generation

Fuzzing for Java and beyond

**Dmitry Ivanov**, *Huawei*, *korifey@gmail.com*, *@korifey_ad*
Saint Petersburg Research Center, R&D Toolchain Labs, Director

**Daniil Stepanov**, *ITMO*, *stepanov0995@gmail.com*

# RnD Toolchain

LABS

We are a team of experts that create next-gen tools for software development, bringing the latest research achievements to production. Our mission is to enhance developers productivity and efficiency, product quality and security with cutting-edge solutions. We leverage the latest technologies and methodologies to drive industry evolution.

## AI for code

PRODUCTIVITY

Boosting Large Language Models by structured code models and code analysis

## Cooddy

QUALITY

Source code analysis tool using Data-Flow Analysis and Static Symbolic Execution

## IdeaLS

PRODUCTIVITY

Intellij IDEA plugin that turns IntelliJ IDEA into an LSP server and delivers the full power of IDEA's language

https://toolchain-labs.com/

UNITTEST BOT

Java/Kotlin C/C++ Python JavaScript Go .NET Demo Research About us

Overview User guide Download

# Imagine you don't have to write tests.

But you still have a perfect bug detector for your code.
And no false positives among bugs.
Tests are generated auto*magically* — with the highest code coverage, fine-tuned mocking, and human-readable test descriptions.
Sounds fantastic? No more.

Try UnitTestBot online demo

Watch how it works

https://www.utbot.org/

# Plan for today

- **[Part I, product]** – Why we need to generate tests?
- **[Part II, environment]** – Theory, products and competitions
- **[Part III, challenges]** – From academia to real-world

# PART I : TEST GENERATION

# Pain point

Programmers **don't like** to write tests

Unit tests:

- Best defense against **regression** (quality increases)
- Kind of living **specification** (understanding increases)
- Errors found by unit tests **easier to correct** (costs reduces)

# Solution

Generate unit test automatically

## Unit test generation

**Goal:** to fixate code behavior

*Regression suite*

Criteria: generate **minimum** number of unit tests that will cover **maximum** lines of code

## Safety verification

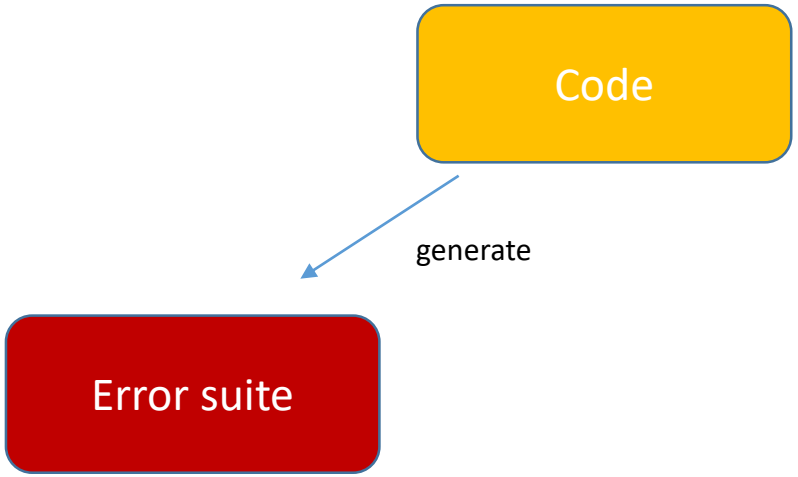
**Goal:** to find bugs and vulnerabilities

*Error suite*

Criteria: Find **maximum** number bugs and express them in form of tests

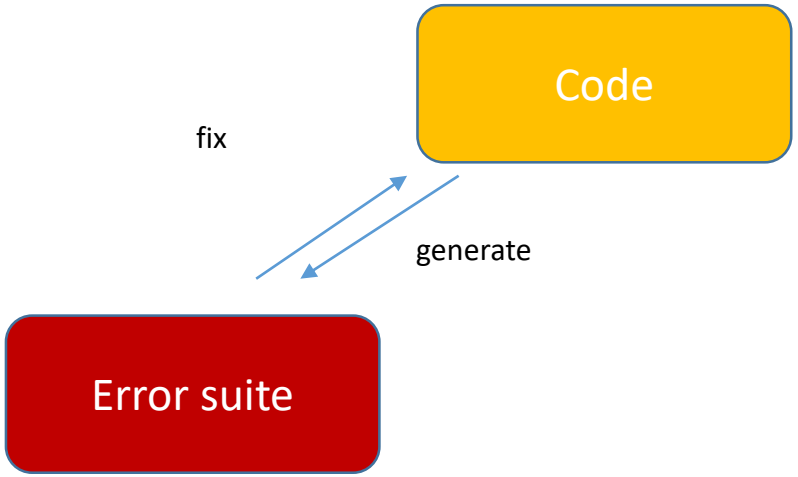
# No tests

# Error suite



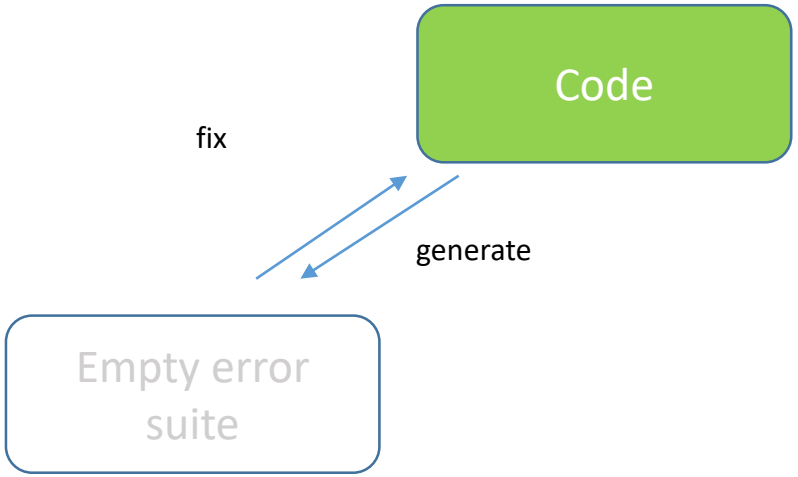

Code contains NPE, StackOverflows and so on

# Error suite



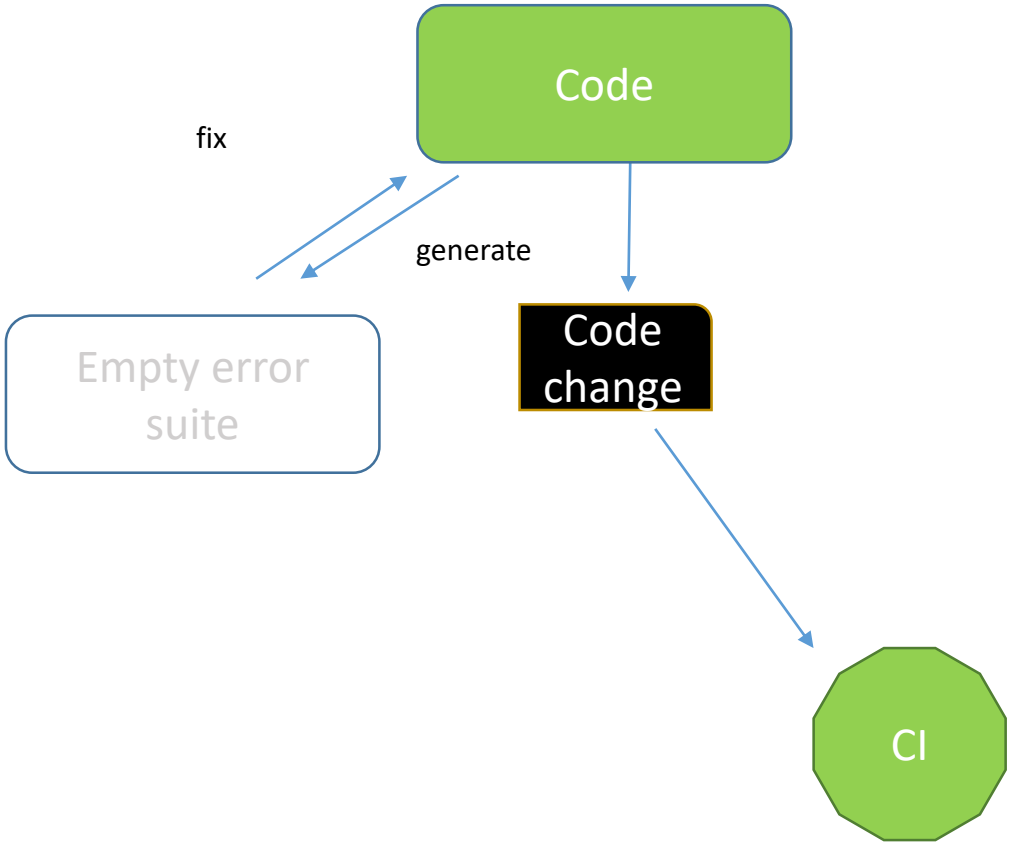

Code contains NPE, StackOverflows and so on

# Error suite




Code hasn't NPE, StackOverflows and so on

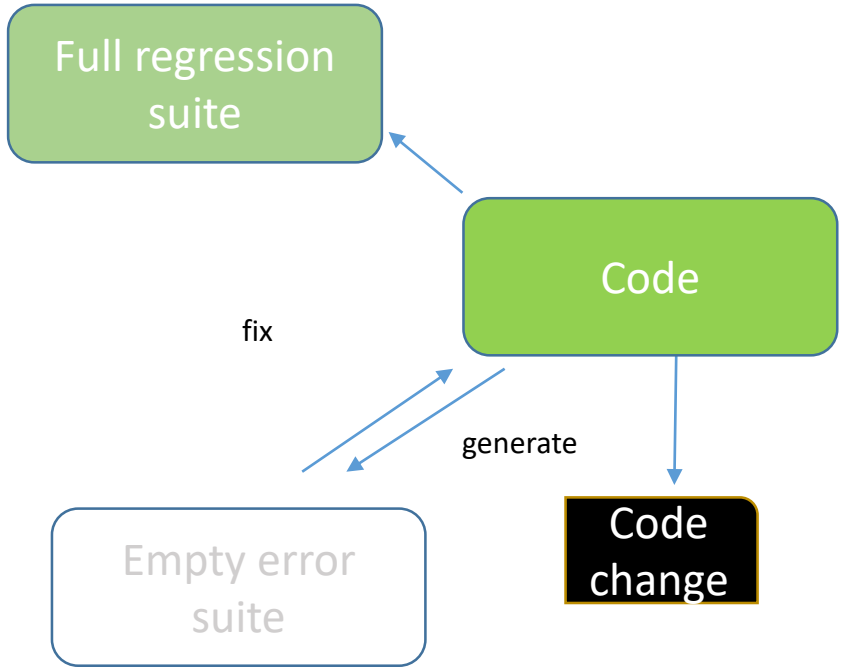
# Regression




Code hasn't NPE, StackOverflows and so on

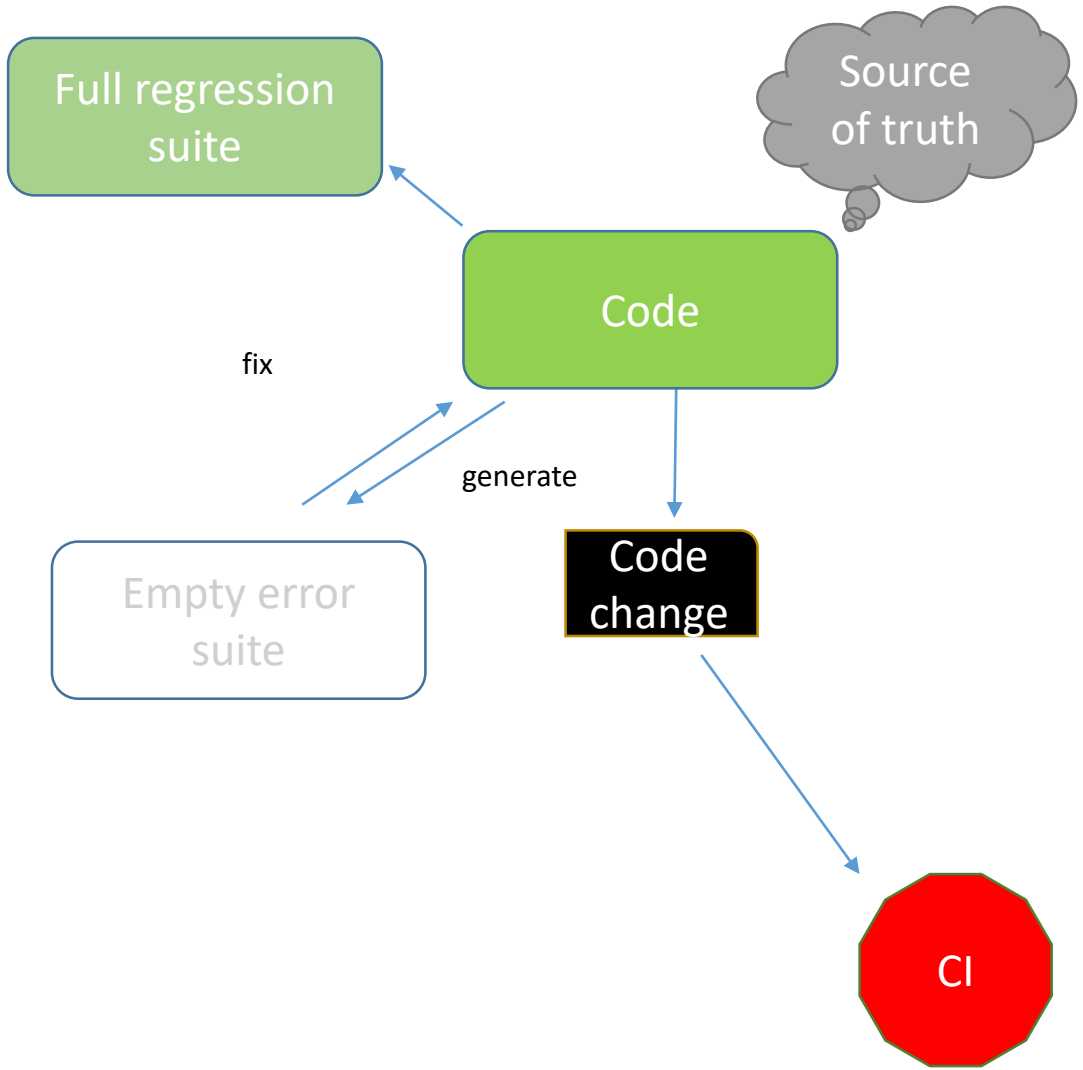
If developer commit change nobody will notice bug until it happens on production

# Regression suite




Code hasn't NPE, StackOverflows and so on

# Regression suite



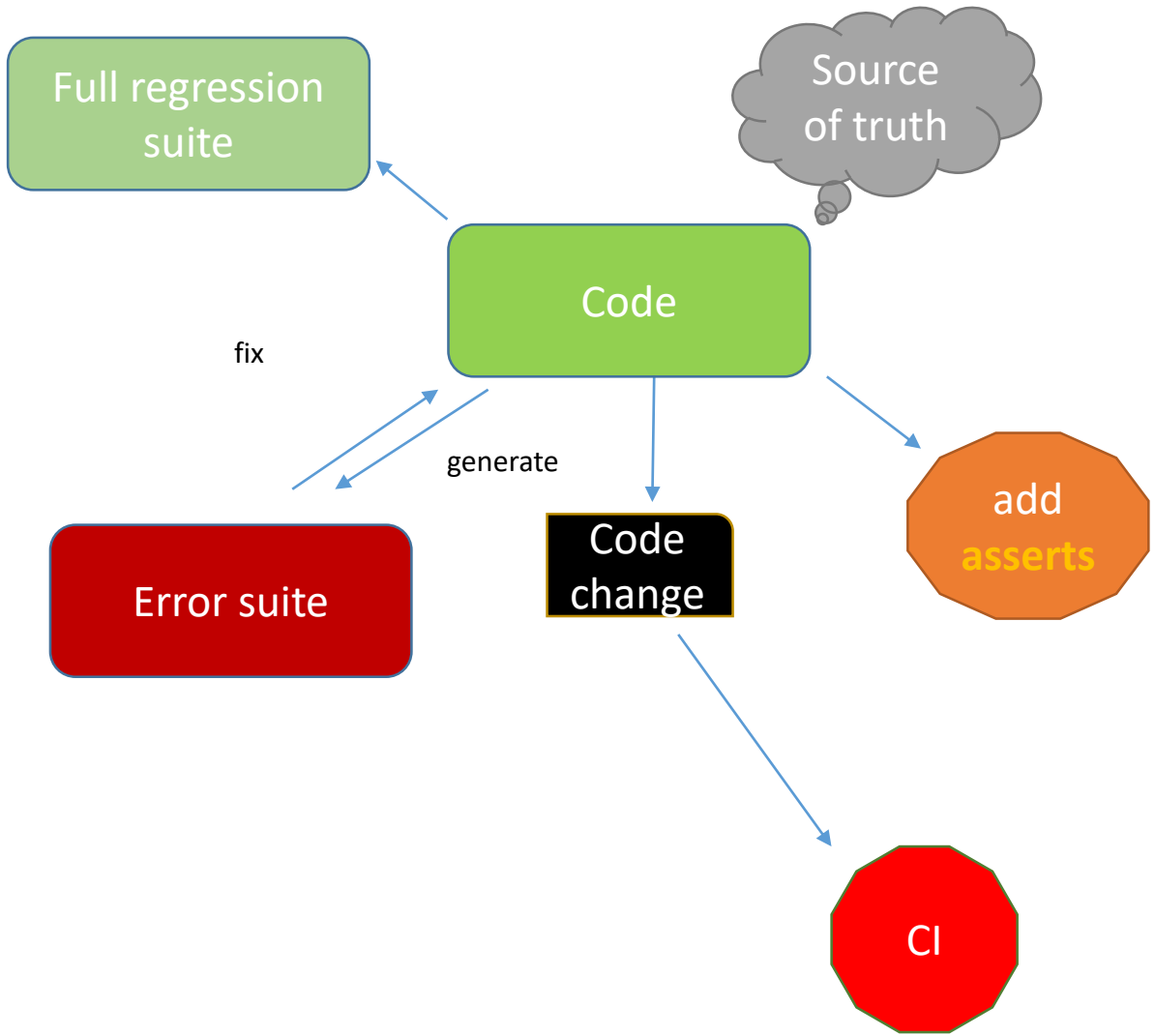

Code hasn't NPE, StackOverflows and so on

Now behavior is fixated.
Red CI status means one of two things:

- Code change breaks correct behavior
- Initial code behavior wasn't correct

Anyway it's *easy to localize* problem

# Specification




Code hasn't NPE, StackOverflows and so on

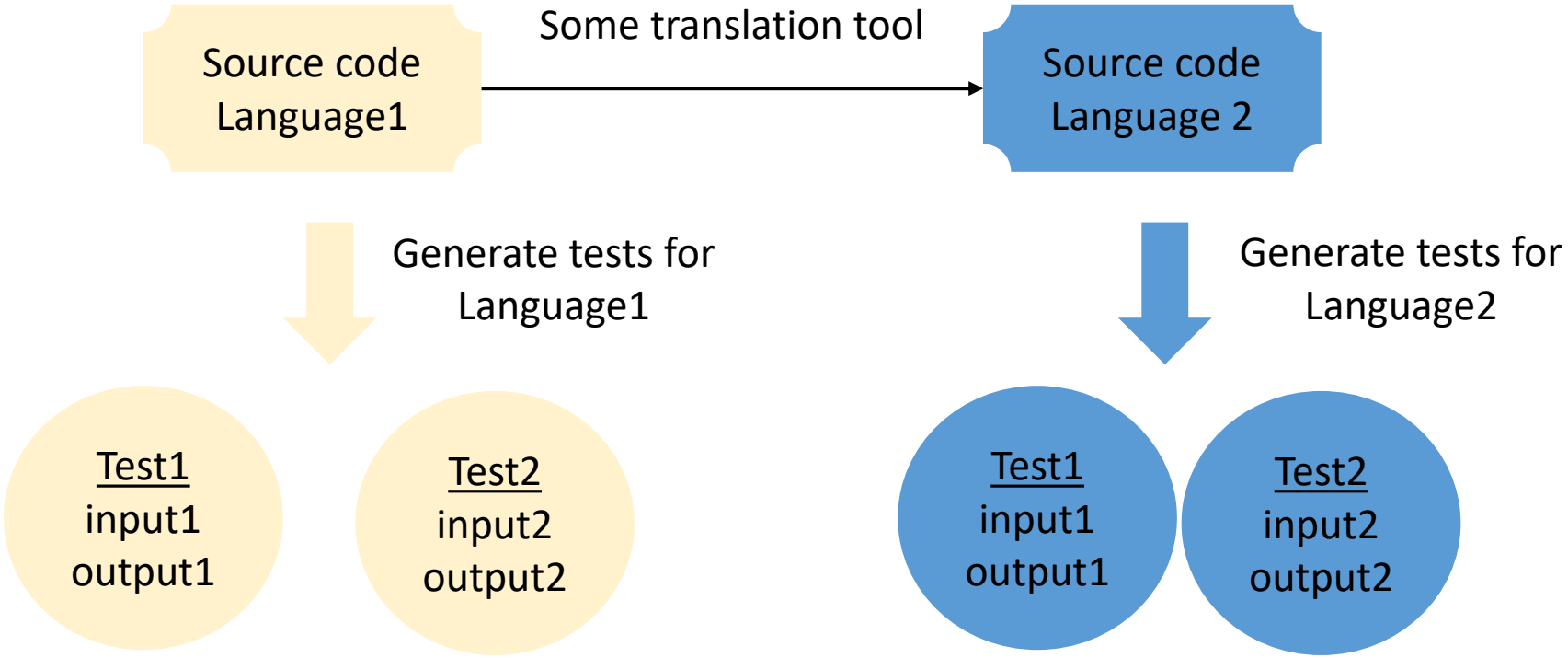
Now behavior is fixated.
Red CI status means one of two things:

- Code change breaks correct behavior
- Initial code behavior wasn't correct

Anyway it's easy to localize problem

Code is tested against **specification** formalized by **asserts**

# Code translation




Leveraging Automated Unit Tests for Unsupervised Code Translation, Baptiste Rozière et all, 2022

# PART II : ENVIRONMENT

# Academia competitions


https://sv-comp.sosy-lab.org/2024/


https://sbst22.github.io/


https://test-comp.sosy-lab.org/2024/


https://sbft24.github.io/

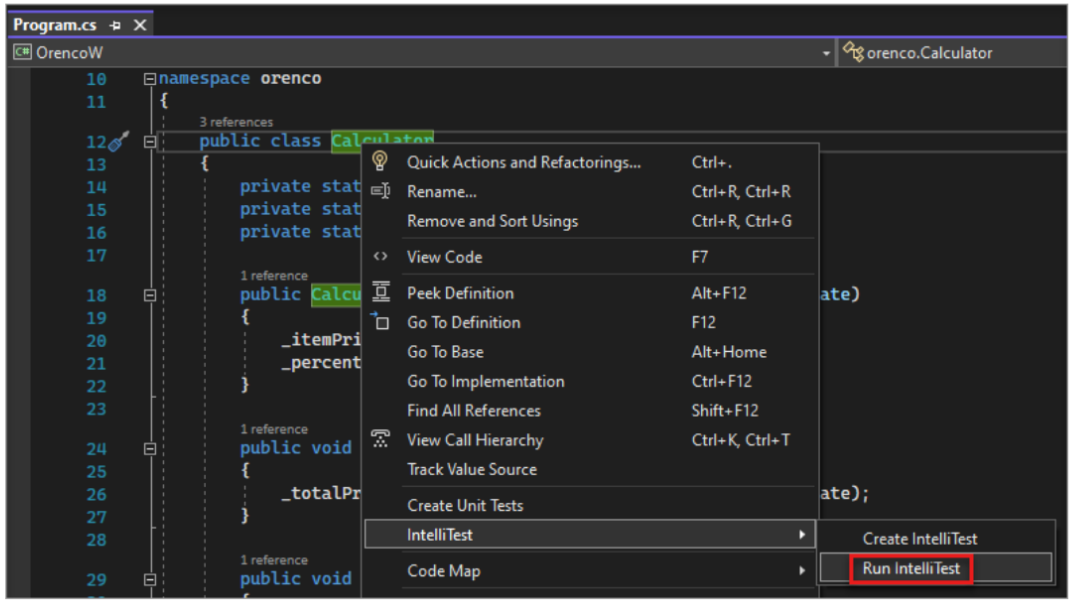
# Industry adoption

## Explore: Use IntelliTest to explore your code and generate unit tests

To generate unit tests, your types must be public.

1. Open your solution in Visual Studio and then open the class file that has methods you want to test.

2. Right-click on a method and choose **Run IntelliTest** to generate unit tests for the code in your method.




VSharp

https://github.com/VSharp-team/VSharp

integrated into UnitTestBot .NET

is competitive to IntelliTest

| Project (C#) | Methods count | C# lines count | Count of methods failed to analyse | | Line coverage (%) | |
|---|---|---|---|---|---|---|
| | | | IntelliTest | V# | IntelliTest | V# |
| JetBrains.Lifetimes | 46 | 412 | 4 | 0 | 64,1 | 85,8 |
| PowerShell | 28 | 1433 | 2 | 0 | 17,7 | 27,5 |
| CosmosOS | 42 | 795 | 2 | 0 | 49,8 | 58,5 |
| Unity | 34 | 3748 | 10 | 0 | 25,6 | 32,9 |
| Custom tests | 47 | 492 | 8 | 0 | 86,2 | 94,7 |

# Fuzzers

*Black-box*

Random inputs generation until crash

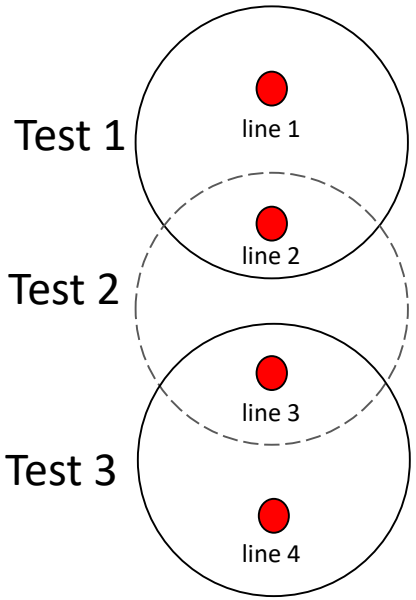
Grey-box

```
while (coverage is not enough) {
  generate new test()
  add to suite if coverage increases
}
```

*Evolutionary algorithm*

- Generations (test suites)
- Cross-over existing tests
- Mutate existing tests

## Unit Test Minimization




**Hitting set** is NP-complete problem

Table 3.3: Apache Commons-Lang testing results

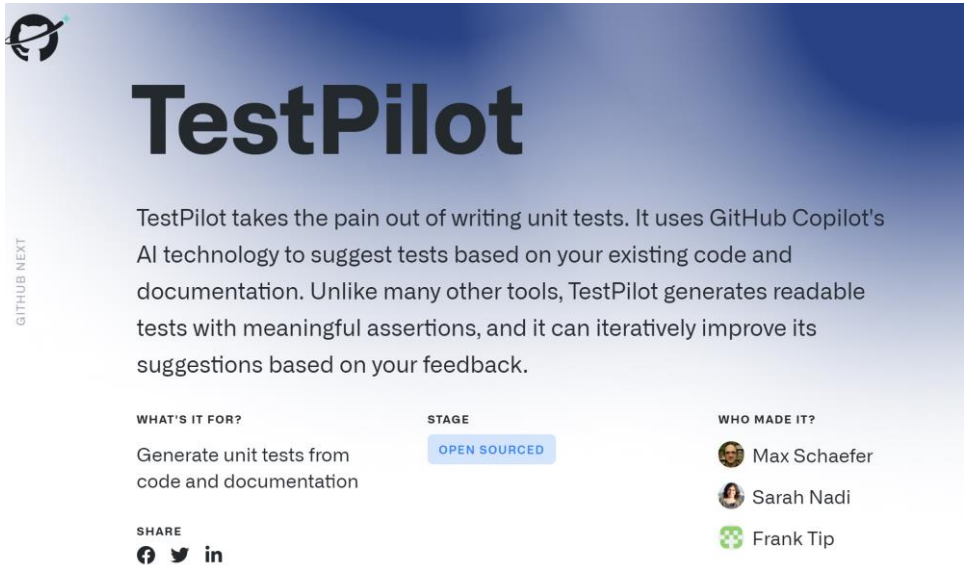
| Algorithm | Full time | Algorithm time (ms) | Accepted | Discarded |
|---|---|---|---|---|
| Naive | 1m 18s | 51 | 1904 | 5740 |
| Naive Essential | 1m 18s | 49 | 1904 | 5740 |
| Greedy | 1m 18s | 905 | 1891 | 5753 |
| GASeeker #1 | 1m 17s | 267 | 1902 | 5742 |
| GASeeker #2 | 1m 19s | 1016 | 1906 | 5738 |
| LPSeeker | 1m 18s | 2040 | 1890 | 5754 |
| PBE | 1m 19s | 1041 | 1890 | 5754 |

EVOSUITE

Automatic Test Suite Generation for Java

# AI Generation


**DiffBlue** raised 22M$ on round A from Goldman Sachs in 2017




https://githubnext.com/

Ponicode, Machinet and others

# Symbolic execution

Program code with execution path

```
int SatisfiesCriteria(int x, int y)
{
    var z = 0;

    if (x > 0)
        z = x * x;

    if (y > x)
        if ((z + y) % 2 == 0)
            return 1;
        else
            return -1;

    return 0;
}
```

Encoded by symbolic execution

First-order logic formula

```
x > 0
  &&
z == x * x
  &&
y > x
  &&
(z + y) % 2 == 0
```

SMT-solver

Params of function that allow execution to run through specific path

x := 1  y := 3

KLEEF https://github.com/UnitTestBot/klee is the best on TESTCOMP-24 error suite prerun

UTBOT https://github.com/UnitTestBot/UTBotJava is the best on SBST-2021, SBST-2022, SBFT-2023 among symbolic execution engines

# PART III : Challenges

# Symbolic execution problems for test generation

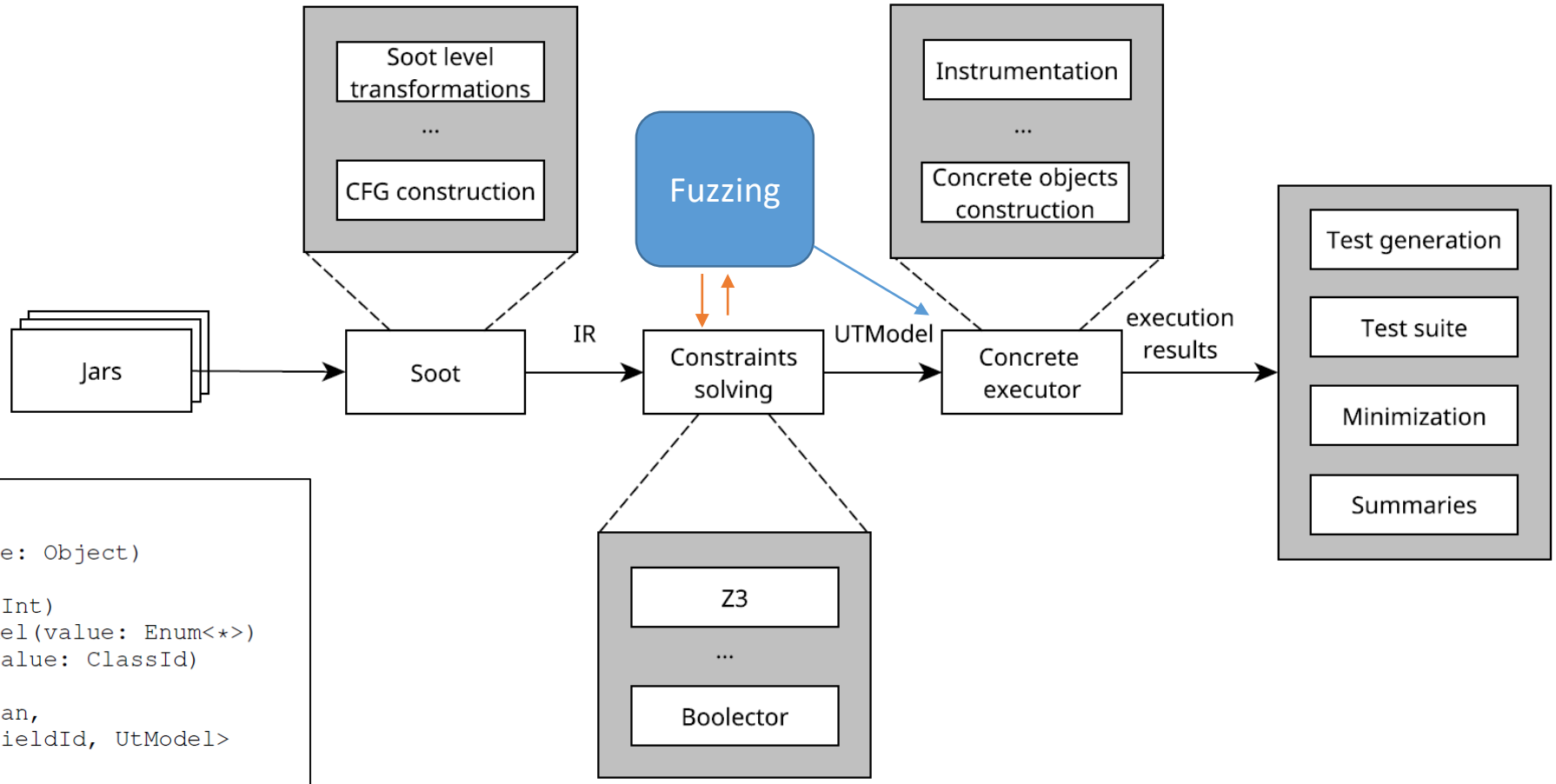
- Libraries complexity => Approximations / Mocks
- Frameworks support (aka Spring)
- Public API instead of reflection

```
bool HaveDiscount(AgeInfo info) {
    if (info.getAge() >= 12
          || info.getAge() < 60) {
        return false;
    } else {
        return true;
    }
}

class AgeInfo {
  private int age;
  int getAge() { return age; }

  AgeInfo() {}
  AgeInfo(int age) { this.age = age; }
}
```

```
@Test
bool TestHaveDiscount() {
    var info = createByReflection("AgeInfo");
    setField(info, "age", 12);
    bool res = HaveDiscount(info);
    assertFalse(res);
}
```

```
@Test
bool TestHaveDiscount() {
    var info = new AgeInfo(12);
    bool res = HaveDiscount(info);
    assertFalse(res);
}
```

# Fuzzing in Unit Test Bot




```
UtModel(classId: ClassId)
....UtNullModel()
....UtPrimitiveModel(value: Object)
....UtVoidModel()
....UtReferenceModel(id: Int)
........UtEnumConstantModel(value: Enum<*>)
........UtClassRefModel(value: ClassId)
........UtCompositeModel(
            isMock: Boolean,
            fields: Map<FieldId, UtModel>
        )
........UtArrayModel(
            stores: Map<Int, UtModel>
        )
........UtAssembleModel(
            instantiation: UtStatement,
            modifications: List<UtStatement>
        )
........UtLambdaModel(captured: List<UtModel>)
```

# Engineering and results

1. **Models** instead of real objects
2. Separate **process** allow to be tolerant to crashes
3. **Threads** inside separate process for Thread.stop()
4. Restart **<clinit>** after each run
5. Exploration/exploitation in fuzzing
6. **Exploration** with *junit-quickcheck*
7. Extract **constants** from source code *(if (a == 42) …)*
8. **Exploitation** – mutate best seed by invocation of random method, constants mutation, built-in mutation of collections and arrays
9. Generate **mocks** using Mockito (for interfaces without implementation)

TABLE I
DESCRIPTION OF THE SBFT BENCHMARK

| Project | #CUTs | #Sampled CUTs |
|---|---|---|
| Collections | 473 | 26 |
| JSoup | 246 | 14 |
| Ta4j | 256 | 30 |
| Spatial4j | 92 | 13 |
| Threeten-extra | 77 | 17 |

TABLE IV
FINAL RANKINGS.

| Tool | CoverageR | UnderstandabilityR | OverallR |
|---|---|---|---|
| EVOSUITE | 1.79 | 2.23 | 1.83 |
| UTBOT-CONCOLIC | 2.61 | 2.13 | 2.56 |
| UTBOT-FUZZER | 3.76 | 3.00 | 3.68 |
| KEX-SYMBOLIC | 4.995 | 3.95 | 4.89 |
| KEX-CONCOLIC | 3.95 | 3.69 | 3.92 |

TABLE II
FUZZING MODULE SBFT-23 COMPETITION RESULTS

| Project | Budget 30 sec | | Budget 120 sec | |
|---|---|---|---|---|
| | Lines coverage, % | Conditions coverage, % | Lines coverage, % | Conditions coverage, % |
| Threeten-extra | 80.07 | 68.20 | 83.22 | 72.67 |
| Collections | 71.90 | 60.16 | 76.70 | 67.09 |
| JSoup | 35.32 | 19.70 | 39.22 | 22.95 |
| Spatial4j | 52.56 | 41.34 | 52.02 | 42.60 |
| Ta4j | 23.40 | 6.57 | 26.15 | 8.22 |

# Thanks for your attention

- Microsoft IntelliTest
- githubnext.com
- diffblue.com **use VPN!**
- toolchain-labs.com
- utbot.org
- github.com/vsharp-team/vsharp
- github.com/UnitTestBot/klee
- github.com/UnitTestBot/UTBotJava
- test-comp.sosy-lab.org/2024/
- https://sbft24.github.io/

**Dmitry Ivanov**, *Huawei SRC*, *korifey@gmail.com* *@korifey_ad*
**Daniil Stepanov**, *ITMO*, *stepanov0995@gmail.com*