JCrasher: Automated Unit Test Generation with Random Testing

Vince Protani

Overview

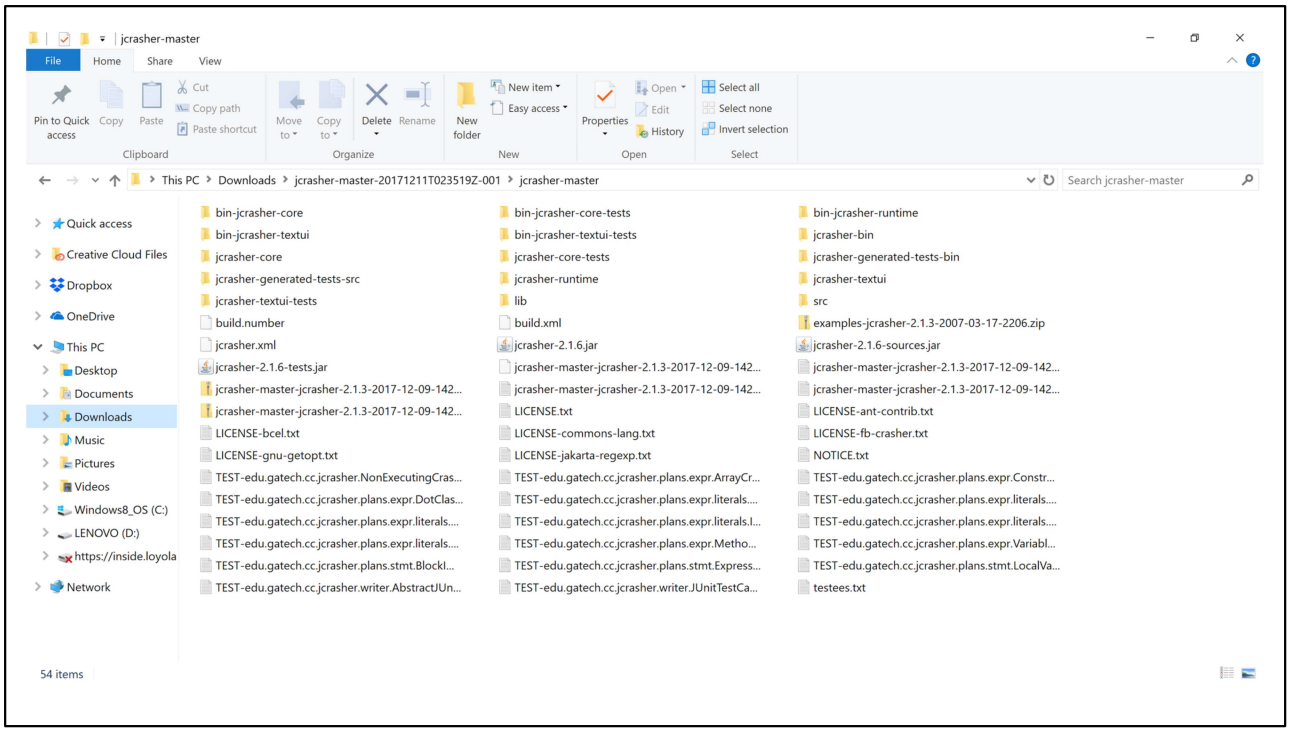
- Developed in early 2000s
- Idea: cover large input domains quickly without exhaustive testing
- Used by researchers and a couple of university courses
- Last updated in 2007

Developers: Christoph Csallner and Yannis Smaragdakis (university professors)
Idea: though there's no method to the madness, random testing can cover a lot of ground with little effort
Used: researchers in universities, at MIT, and at Microsoft Research; undergrad course on software verification and validation; graduate course on program verification
Hasn't been touched since 2007: incompatible with Java programs beyond Java 4; Eclipse plugin no longer works

How it works

- Run as Ant program
- Inspect byte code for input space
- Generate and JUnit unit tests
- Distinguish between expected and unexpected exceptions
- Report unit test results

```
[vprotani@linus]:~/Downloads/examples> ant -f jcrasher.xml
Buildfile: /home/vprotani/Downloads/examples/jcrasher.xml

jcrasher:
    [unjar] Expanding: /home/vprotani/Downloads/jcrasher-2.1.3.jar into /home/vprotani/Downloads/examples

init:

testee.compile:
    [javac] /home/vprotani/Downloads/examples/use-jcrasher.xml:67: warning: 'includeantruntime' was not set, defaulting to buil
d.sysclasspath=last; set to false for repeatable builds

test.generate:

test.compile:
    [javac] /home/vprotani/Downloads/examples/use-jcrasher.xml:97: warning: 'includeantruntime' was n t set, defaulting to buil
d.sysclasspath=last; set to false for repeatable builds
    [javac] Compiling 7 source files to /home/vprotani/Downloads/examples/jcrasher-generated-tests-bin
    [javac] warning: [options] bootstrap class path not set in conjunction with -source 1.4
    [javac] warning: [options] source value 1.4 is obsolete and will be removed in a future release
    [javac] warning: [options] target value 1.4 is obsolete and will be removed in a future release
    [javac] warning: [options] To suppress warnings about obsolete options, use -Xlint:-options.
    [javac] 4 warnings

test.run.filtering:
     [java] Java Result: 1

test.archive:
      [zip] Building zip: /home/vprotani/Downloads/examples/examples-jcrasher-2.1.3-2017-12-10-1915.zip
   [delete] Deleting: /home/vprotani/Downloads/examples/use-jcrasher.xml

BUILD SUCCESSFUL
Total time: 2 seconds
13.716u 0.576s 0:03.21 444.8%   0+0k 3312+3176io 0pf+0w
[vprotani@linus]:~/Downloads/examples>
```



After building view: executable, dependencies, build files, SUT

Welcome to JCrasher

An automatic robustness tester for Java

Abstract

JCrasher is an automatic robustness testing tool for Java code. JCrasher examines the type information of a set of Java classes and constructs code fragments that will create instances of different types to test the behavior of public methods under random data. JCrasher attempts to detect bugs by causing the program under test to "crash"--to throw an undeclared runtime exception. Although in general the random testing approach has many limitations, it also has the advantage of being completely automatic: no supervision is required except for online inspection of the test cases that have caused a crash. Compared to other similar commercial and research tools, JCrasher offers several novelties:

- JCrasher transitively analyzes methods, determines the size of each tested method's parameter-space and selects parameter combinations and therefore test cases at random, taking into account the time allocated for testing
- JCrasher defines heuristics for determining whether a Java exception should be considered a program bug or the JCrasher supplied inputs have violated the code's preconditions
- JCrasher includes support for efficiently undoing all the state changes introduced by previous tests
- JCrasher produces test files for **JUnit**--a popular Java testing tool
- JCrasher can be integrated in the Eclipse IDE

Download

Download JCrasher--it's free. You can still get the old JCrasher version we used for the experiments that appeared in Software--Practice & Experience 2004.

Check 'n' Crash and DSD-Crasher are JCrasher's successors. Check 'n' Crash uses a more directed search to find bugs--by building on ESC/Java's theorem proving techniques. DSD-Crasher adds another dynamic analysis step at the beginning of the static-dynamic Check 'n' Crash analysis pipeline.

Using JCrasher as an Eclipse 2 plug-in

To provide a common case example, the following assumes that you are running Windows and have already installed Eclipse in `c:\Program Files\Eclipse`

1. Download and unpack `Jcrasher_0.1.9_eclipse.zip`.
2. Move the content (=the directory `cc.jcrasher_X.Y.Z`) to
   `c:\Program Files\Eclipse\plugins`
   Double-check that you now have the following files
   - `c:\Program Files\Eclipse\plugins\cc.jcrasher_X.Y.Z\jcrasher.jar`
   - `c:\Program Files\Eclipse\plugins\cc.jcrasher_X.Y.Z\plugin.xml`
3. Start Eclipse and include both the JUnit and the JCrasher libraries in the projects, in which you want to use JCrasher
   - Select the project, right click, `Properties`, `Java Build Path`, `Libraries`, `Add External Jars...`
   - `c:\Program Files\Eclipse\plugins\org.junit_X.Y.Z\junit.jar`
   - `c:\Program Files\Eclipse\plugins\cc.jcrasher_X.Y.Z\jcrasher.jar`
4. Select any set of top-level Java types of one of the projects you have prepared above, right click, `JCrasher: Generate JUnit TestCase`.

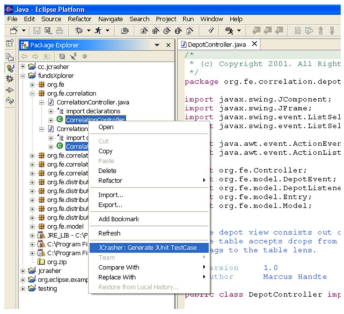


Figure 1. Invoking JCrasher conveniently as an Eclipse plug-in.

Simple homepage, not much to offer information-wise
Pretty ancient looking page/implementation

Project
Source
Issues
Wikis
Downloads

jcrasher

An automatic robustness tester for Java

Usage

The most convenient way to use JCrasher is by running an Ant script like jcrasher.xml:

```
ant -f jcrasher.xml
```

It performs all steps completely automatically.

Example

The Ant script performs the following steps. You can follow along with the example files included in the examples directory. Download the examples directory, open a command line in your copy of the examples directory, and run our Ant script:

```
ant -f jcrasher.xml
```

1. jcrasher.xml compiles the Java source files under test. It looks for a file like testees.txt and interprets every line as the location of a Java source file under test, relative to the src property in jcrasher.xml. Our testees are trivia/DivByZero.java, trivia/ManyParameters.java, and trivia/NullDeref.java located in the src directory.
2. jcrasher.xml runs JCrasher on the compiled Java classes under test. JCrasher will generate JUnit test case (as Java source files) for the classes under test. Now we should have the JUnit test cases DivByZeroTest1, DivByZeroTest2, DivByZeroTest3, ManyParametersTest1, etc. Additionally, JCrasher has generated a test suite that aggregates all generated test cases in JUnitAll. (All these files are included in the result zip file generated in step 5.)
3. jcrasher.xml compiles the generated Java source files (the JUnit test cases).
4. jcrasher.xml runs JCrasher's extended version of JUnit on the generated test cases. This will enhance the JUnit text output by suppressing redundant warnings.
5. jcrasher.xml packages the results into a zip file like examples-jcrasher-2.1.3-2007-03-17-2206.zip. This zip file includes the original testee sources, the generated JUnit test case sources, the log output of JCrasher and JUnit, and the configuration files used: jcrasher.xml and testees.txt.

Project
Source
Issues
Wikis
Downloads

jcrasher

ID	Status	Summary
5	New	I'm trying to run JCrasher on a javaclass but it is giving compliation error Type-Defect Priority-Medium
4	New	Compilation failure of the Generated Junit test cases Type-Defect Priority-Medium
3	New	How to give multiple classpath entry in Jcrasher.xml Type-Defect Priority-Medium
2	New	Even after generating the test code , the JCrasher Main process is not coming out Type-Defect Priority-Medium
1	New	Some JCrasher-generated test cases do not compile with Java 6 Type-Defect Priority-Medium

Pretty helpful, gives a rundown of how to execute
Most issues addressed but unanswered

Programs under Test

```java
/*
 * DivByZero.java
 *
 * Copyright 2006 Christoph Csallner and Yannis Smaragdakis.
 */
package trivia;

/**
 * Throws java.lang.ArithmeticException: / by zero
 *
 * @author csallner@gatech.edu (Christoph Csallner)
 */
public class DivByZero {

    /**
     * Crashes for i==0.
     */
  public static double inverse(int i) {
        return 1/i;
    }

  public static double always(int i) {
        return i/0;
    }
}
```

```java
/*
 * ManyParameters.java
 *
 * Copyright 2007 Christoph Csallner and Yannis Smaragdakis.
 */
package trivia;

/**
 * @author csallner@gatech.edu (Christoph Csallner)
 */
public class ManyParameters {

    /**
     * @return sum of parameters
     */
    public int sum(int a, int b, int c, int d, int e, int f, int g) {
        return a + b + c + d + e + f + g;
    }
}
```

```java
/*
 * NullDeref.java
 *
 * Copyright 2006 Christoph Csallner and Yannis Smaragdakis.
 */
package trivia;

/**
 * Throws java.lang.NullPointerException
 *
 * @author csallner@gatech.edu (Christoph Csallner)
 */
public class NullDeref {

    /**
     * Crashes for o==null.
     */
    public static int foo(Object o) {
        return o.hashCode();
    }
```

Incompatible with desired programs (Hangman, Calculus)
Features unable to be used in Java 4, like generics

Plus side: expected errors, large domains. Down side: lack of complexity to observe behavior later in program
Expected errors: divide by zero, large inputs exceeding int, null dereference

```
.............................
Time: 0.12
There were 4 errors:
1) test1(trivia.DivByZeroTest2)java.lang.ArithmeticException: / by zero
        at trivia.DivByZero.inverse(DivByZero.java:19)
        at trivia.DivByZeroTest2.test1(DivByZeroTest2.java:42)
2) test0(trivia.DivByZeroTest3)java.lang.ArithmeticException: / by zero
        at trivia.DivByZero.always(DivByZero.java:23)
        at trivia.DivByZeroTest3.test0(DivByZeroTest3.java:31)

FAILURES!!!
Tests run: 2201,  Failures: 0,  Errors: 4

Suite name: JUnitAll
Exceptions and Errors after filtering (E): 2
Exceptions and Errors total (e):  4
Run time: 285ms
```

Total significant errors caught

Total test cases between tests of all three programs

Output for entire test suite
Missing expected errors in ManyParameters and NullDeref

Random(?) Testing

```java
/**
 * JCrasher-generated test case.
 */
public void test22() throws Throwable {
  try{
    int i1 = 0;
    int i2 = 0;
    int i3 = -1;
    int i4 = 0;
    int i5 = 1;
    int i6 = 0;
    int i7 = 0;
    ManyParameters m8 = new ManyParameters();
    m8.sum(i1, i2, i3, i4, i5, i6, i7);
  }
  catch (Throwable throwable) {throwIf(throwable);}
}
```

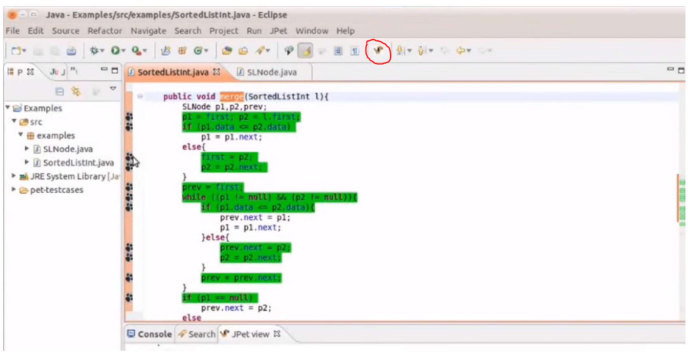
Generated inputs for ManyParameters all look like this ⇒ not hitting important bugs

Testing Methodology

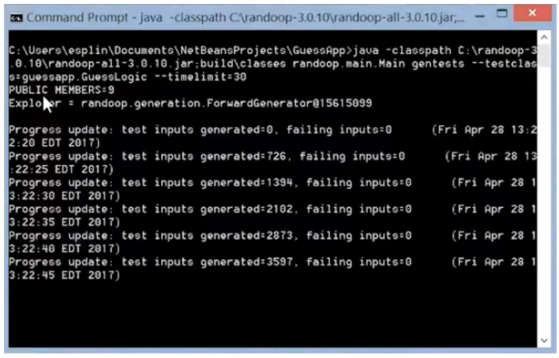
- Black box testing
- No knowledge of procedures other than parameters
- Good for testing early on
- Start to programs like Hangman or VendingMachine

Not aware of how the program operates, just what data is required
Good for testing at the beginning of a program since there's no need to manually create test cases -- easy bugs can be discovered with little effort
Programs that rely on possibly fault human interaction are good to test

Feasible Alternatives



jPET: Eclipse plugin (~2011)



Randoop: command-line Java program (2017)

jPET
- Easy to use interface
- Specification selection: input domain, coverage criteria
- Inspection of unit tests

Randoop
- Command-line program
- Only dependent on Java (no Ant)
- Slower, but specifications available

Both have specifications and both are more up-to-date than JCrasher

Conclusions

- Random testing good for initial tests
- JCrasher
  - Outdated
  - Non-friendly interface
  - Incompatible with current programs
  - Requires knowledge of build files and many other pieces of the program
  - Everything considered… big no-no
- Viable alternatives
  - jPET
  - Randoop