

Automated Software Testing: Fuzzing Techniques (Cont.)

Zhoulai Fu

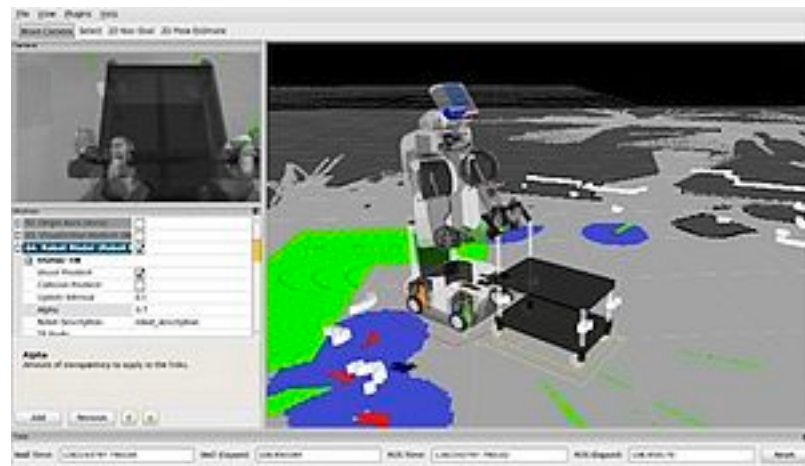
IT University of Copenhagen

September 30, 2021

Today



Sanitizing techniques



Fuzzing for real-world



Two mini-projects



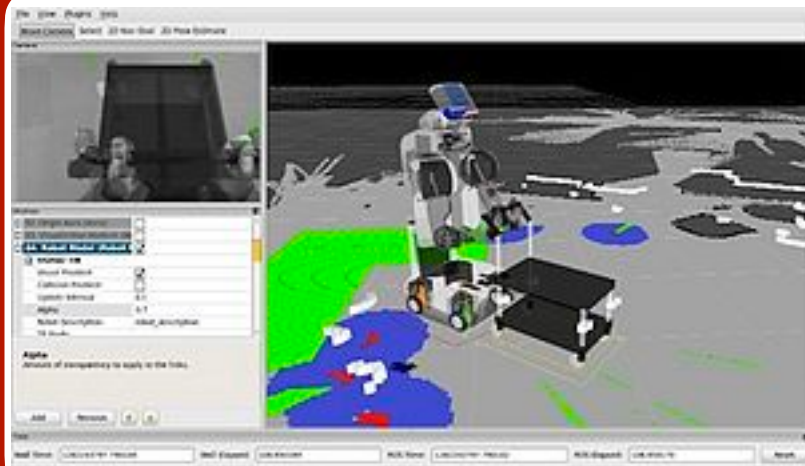
Sanitizing techniques

- In automate testing, an utmost challenge is _____
- Free spec includes _____
- Sanitizing is free spec integrated in today's compilers

DEMO



Sanitizing techniques



Fuzzing for real-world



Two mini-projects

How fuzzing works?

```
69  int main(int argc, char *argv[])
70  {
71      char *usage = "Usage: %s\n"
72                  "Text utility - accepts comma
73                  "\tInput          | Output
74                  "\t-----+-----
75                  "\tu <N> <string>  | Upperc
76                  "\thead <N> <string> | The fi
77      char input[INPUTSIZE] = {0};
78
79      // Slurp input
80      if (read(STDIN_FILENO, input, INPUTSIZE) < 0)
81      {
82          fprintf(stderr, "Couldn't read stdin.\n");
83      }
84
85      int ret = process(input);
```

- Need “main”, the fuzz driver
- Instrument code to monitor coverage.
- Generate input data.

How fuzzing works?

From vulnerable.c

```
69  int main(int argc, char *argv[])
70  {
71      char *usage = "Usage: %s\n"
72                  "Text utility - accepts comma
73                  "\tInput          | Output
74                  "\t-----+-----
75                  "\tu <N> <string>    | Upper
76                  "\thead <N> <string> | The fi
77      char input[INPUTSIZE] = {0};
78
79      // Slurp input
80      if (read(STDIN_FILENO, input, INPUTSIZE) < 0)
81      {
82          fprintf(stderr, "Couldn't read stdin.\n");
83      }
84
85      int ret = process(input);
```

- “main” function
- interacts with software under test
- takes an input from stdin or file

Aft-fuzz

Input corpus

?

Afl-gcc

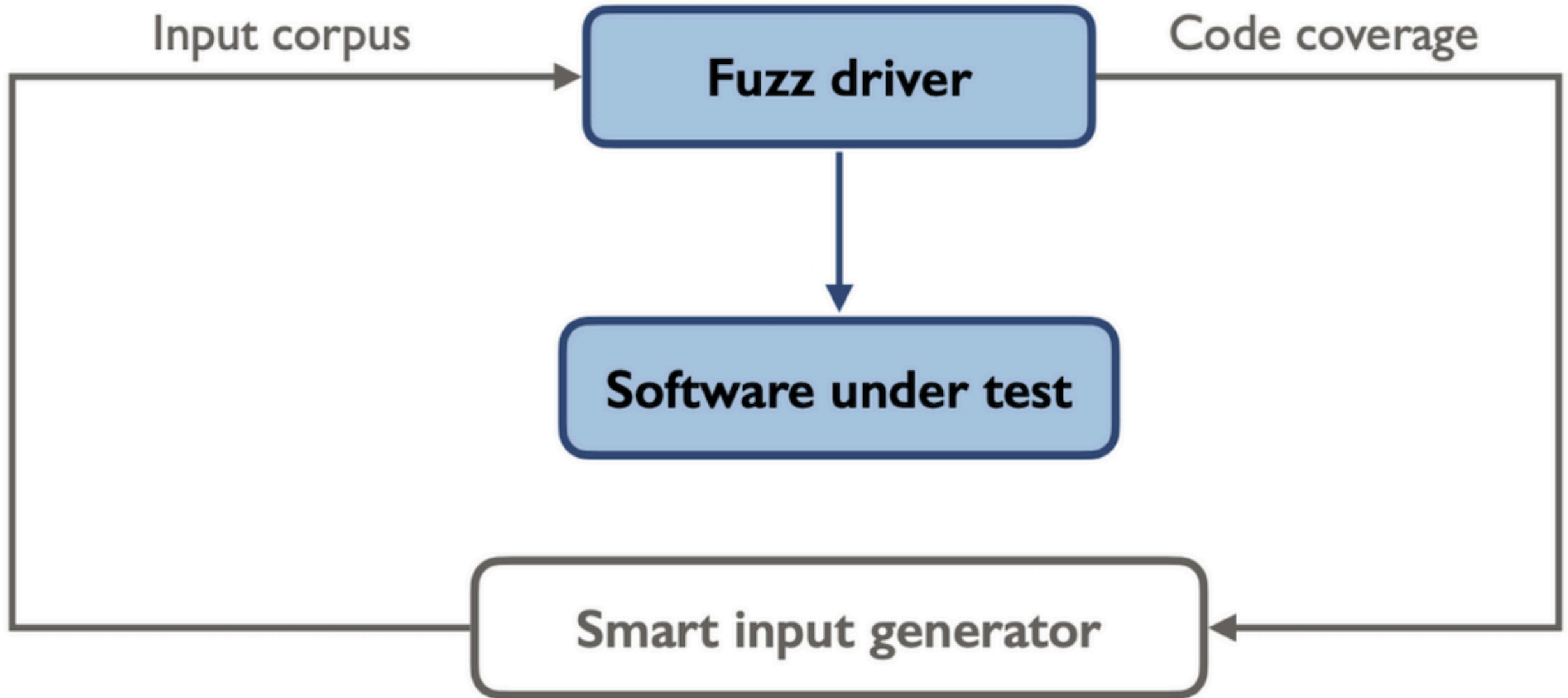
Code coverage

Fuzz driver

Software under test

Smart input generator

How to get Fuzz Driver?



How to get Fuzz Driver?

- [FuzzGen: Automatic Fuzzer Generation](#)
- [IntelliGen: Automatic Driver Synthesis for Fuzz Testing](#)
- [FUDGE: Fuzz Driver Generation at Scale](#)
- [WINNIE : Fuzzing Windows Applications with Harness Synthesis and Fast Cloning](#)
- Written manually in general
- But can be automated for software involving floating-point calculation!

The image consists of two vertically stacked screenshots of a ROS2 TurtleSim environment. The top screenshot shows a terminal window on the left with a list of INFO messages, each followed by a timestamp in brackets. The messages are: INFO [1580633793.744], INFO [1580633797.681], INFO [1580633797.681], INFO [1580633799.615], INFO [1580633799.615], INFO [1580633803.551], INFO [1580633803.551], INFO [1580633805.487], INFO [1580633805.487], INFO [1580633809.424], INFO [1580633809.424], INFO [1580633811.359], INFO [1580633811.359], INFO [1580633815.295], INFO [1580633815.295], INFO [1580633817.232], and INFO [1580633817.232]. The terminal prompt is 'zhfu@ubuntu:~\$ rosr'. To the right of the terminal is a TurtleSim window with a blue background. A white line extends from the left edge of the window to a small turtle icon on the right. A mouse cursor is visible near the top center. The bottom screenshot shows the same TurtleSim window, but the terminal window now displays a series of 'no! I hit the wall! (Clampi' messages. The TurtleSim window shows a small turtle icon in the center of the blue field. A mouse cursor is visible near the top center.

```
Czhfu@ubuntu:~$ rosrun
```

[illegible]

Before

```
1 void forward(ros::Publisher twist_pub) {
2     if (hasReachedGoal()) { ... }
3     else commandTurtle(twist_pub, 1.0, 0.0);
4 }
5 ...
6 int main(int argc, char** argv){
7     ...
8 }
```

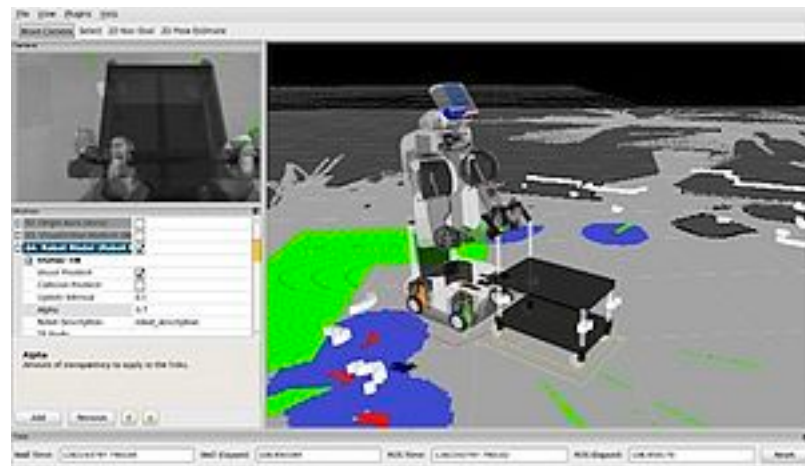
Fuzzing Robot Operating System

After

```
1 double g1, g2;
2 void forward(ros::Publisher twist_pub) {
3     if (hasReachedGoal()) { ... }
4     else commandTurtle(twist_pub, g1, g2);
5 }
6 ...
7 int main(int argc, char** argv) {
8     if (scanf("%lf,%lf", &g1,&g2) != 2) return -1;
9     ...
10 }
```



Sanitizing techniques



Fuzzing for real-world



Two mini-projects