Rethinking Unit Testing:

Automating the Generation of Java Unit Tests

Markus Zimmermann
@zimmskal

geecon

symflower

PRAGUE
24-25 Oct 2022

Join sli.do for asking questions

https://sli.do

Code: #geecon2022

Room: 9

A major problem in software development

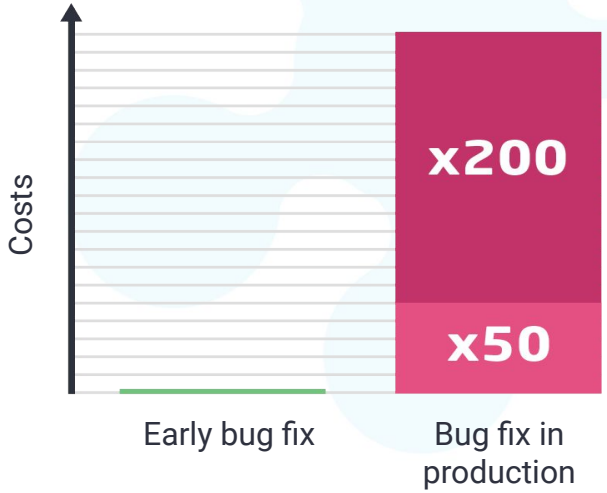
- Software testing is **hard**
  - Code is often not designed for automated testing
  - Infrastructure and tools are not always optimal
  - Time pressure to finish features and fixes
- Not because we cannot test or lack education
- Usually too little time to test thoroughly
- **Even with enough time, there could be another bug**

Should we give up on software testing?

(Hint: no)

Finding and fixing bugs early on saves time

- With shift-left testing:
  - No black-box reports
  - No release hoops
  - Less rework
- Other time consumers of bugs:
  - **Identify if a bug exists at all**
  - **Find a reproducer**
  - Debug and fix the problem

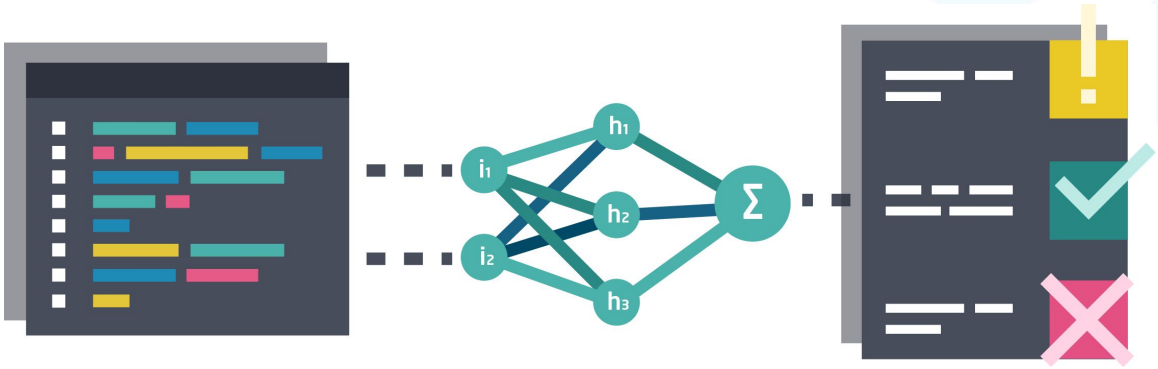


How could we be more productive?

- Automating the answers to the following questions:
  - Are there bugs in the existing implementation?
  - Is the implementation doing what it should do?
  - If not, how can we reproduce the problem?
- **Even just a partial answer saves a lot of time**

Generate unit tests from the implementation

The ideal tool fully automatically finds tests that reflect the current implementation. Meaning it generates for each relevant path and error possibility one test case.



Computing test candidates

Don’t guess, compute tests!

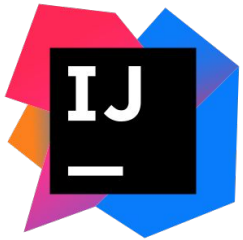
Use Symbolic Execution (SE)

- Established research topic
- Checks **every** functionality
- Computes **targeted** test cases
- Reaches **highest** test coverage
- **Finds bugs automatically**

A hands-on example

Follow along yourself:

get.symflower.com



A hands-on example: specification

Specification: Create a copy function that copies an array of strings from one parameter to another, and return the result. (We are interested in the workflow, not a perfect solution)

Follow along yourself:

get.symflower.com

A hands-on example: final form

```
class Copy {
   static String[] copy(String[] from, String[] to) {
       if (from == null || to == null) {
           throw new IllegalArgumentException();
       }

       for (int i = 0; i < from.length; i++) {
           to[i] = from[i];
       }

       return to;
   }
}
```

Follow along yourself:

get.symflower.com

Three ways of software testing

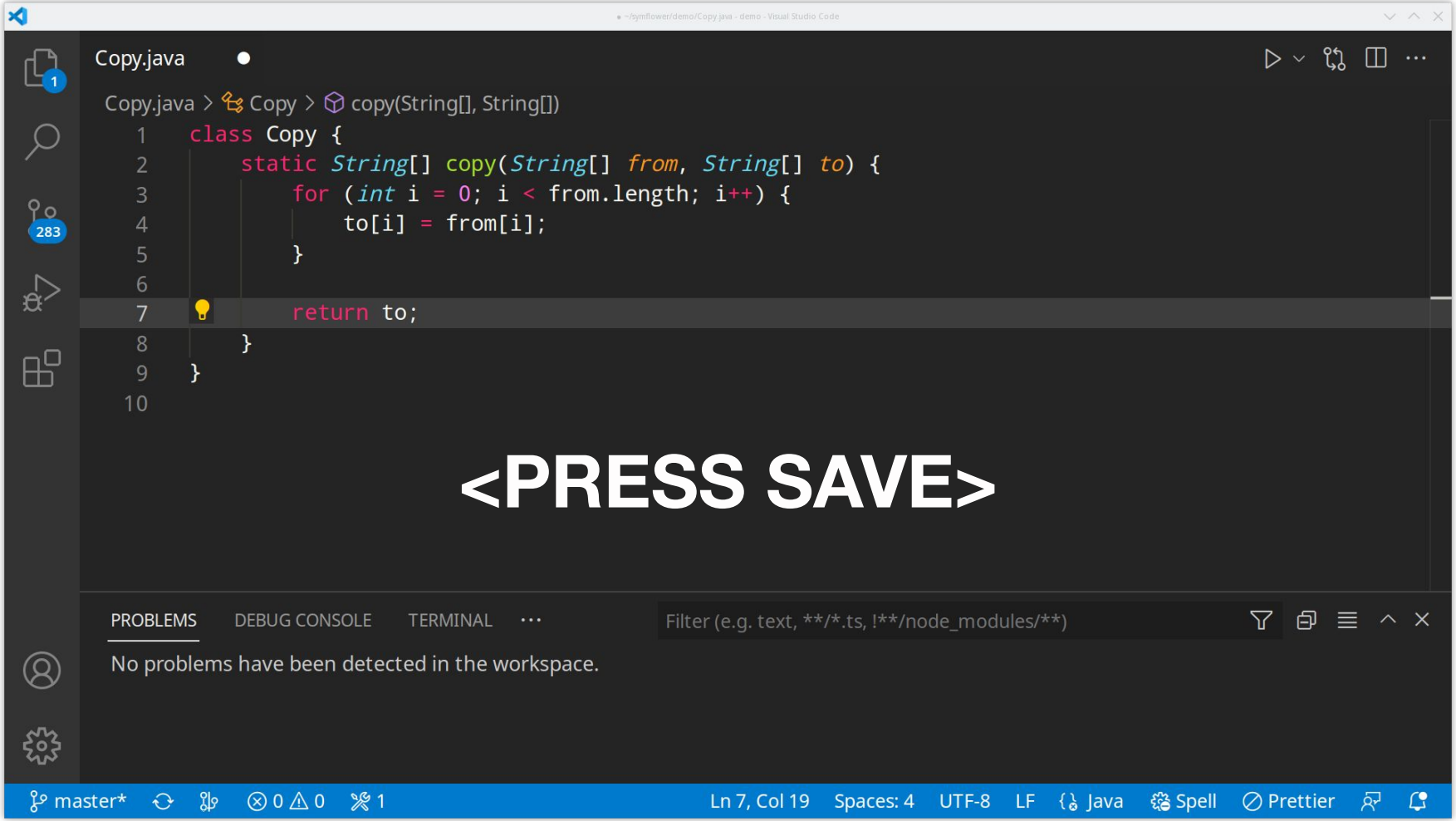
1. Do not write tests at all
2. First implement then write tests
3. First write a test then implement (TDD)

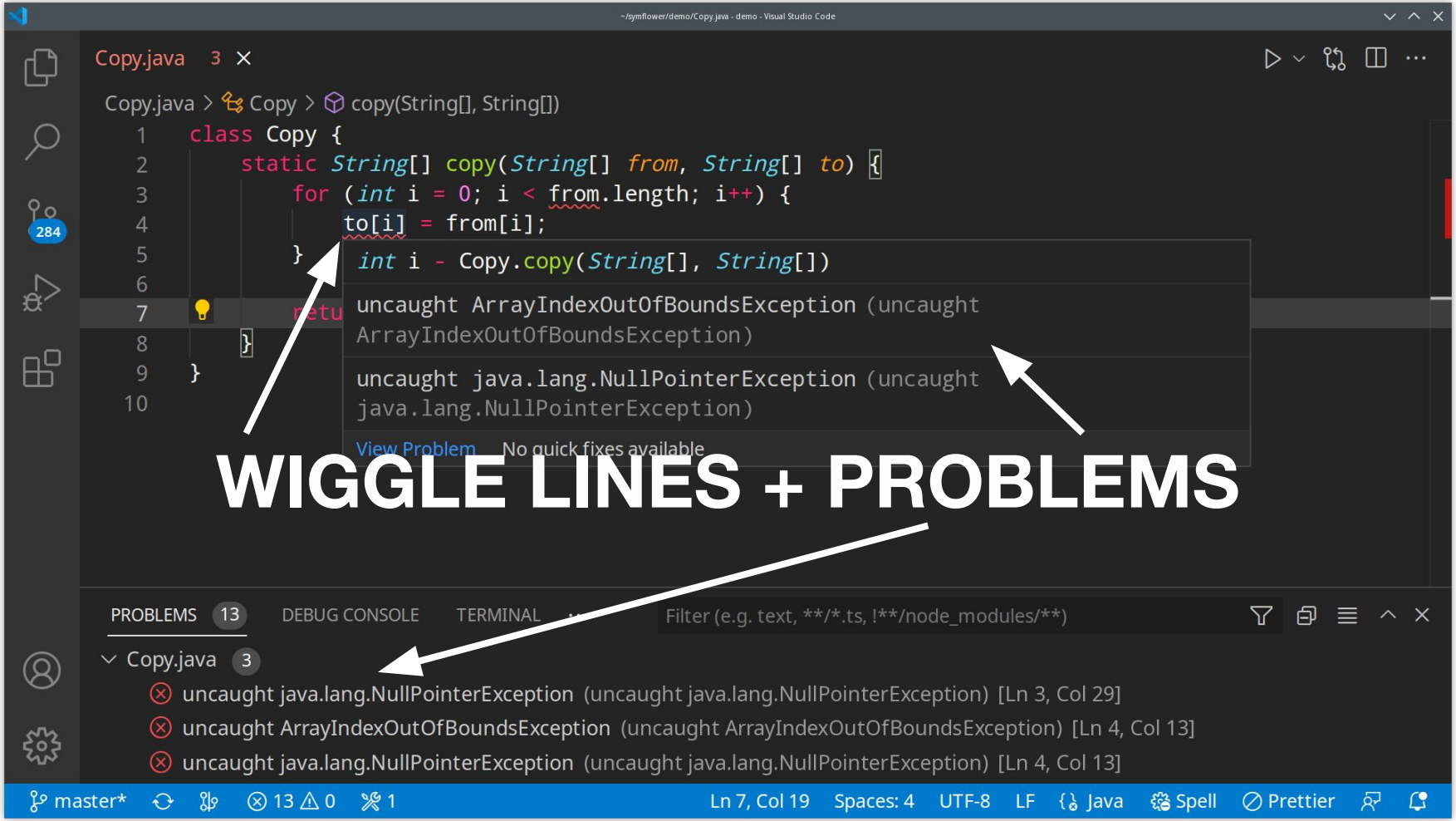
Follow along yourself:

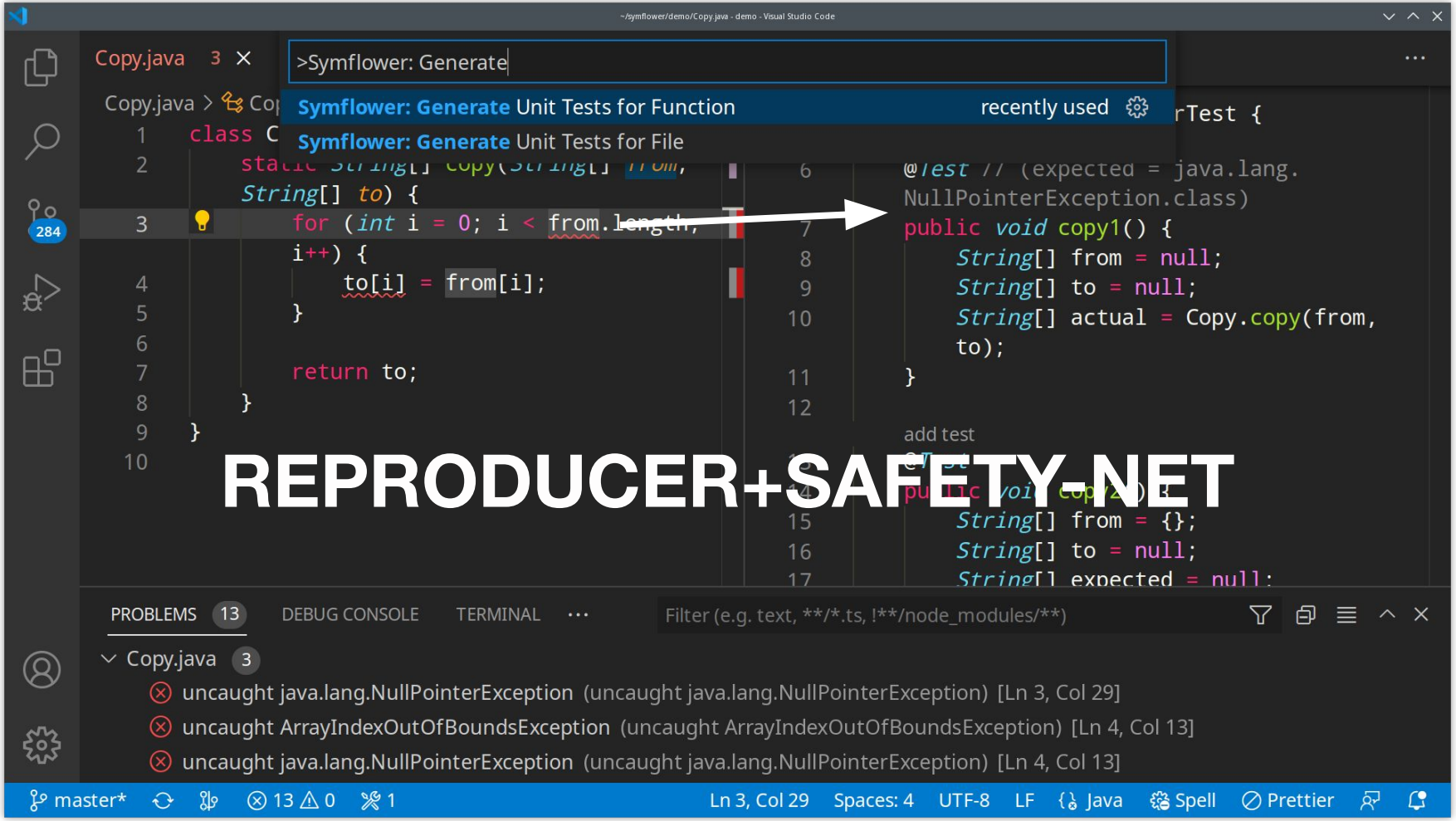
get.symflower.com

How generated tests change software testing

1. **Do not write tests at all**
   - Without generated tests
     - Development at first “feels faster”
     - Cannot know if changes break existing behavior
     - Requires manual test cycles
   - With generated tests
     - Problems can be found automatically
       Similar to a static analysis, but with reproducer!
     - Safety-net to see behavior changes

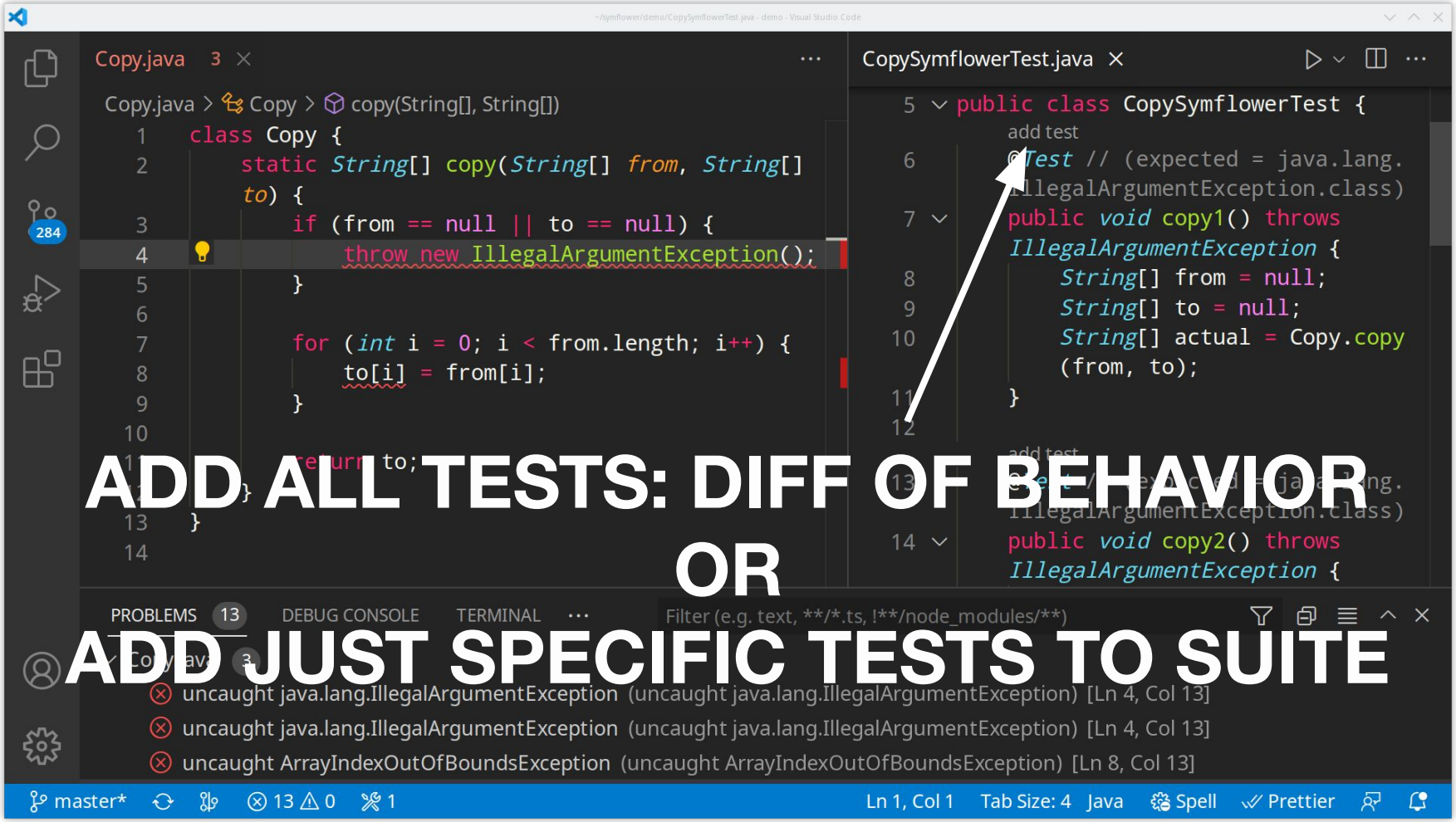

Copy.java
Copy.java > Copy > copy(String[], String[])
class Copy {
static String[] copy(String[] from, String[] to) {
for (int i = 0; i < from.length; i++) {
to[i] = from[i];
}
return to;
}
}
<PRESS SAVE>
PROBLEMS
DEBUG CONSOLE
TERMINAL
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.
master*
0 0
1
Ln 7, Col 19
Spaces: 4
UTF-8
LF
Java
Spell
Prettier


~/symflower/demo/Copy.java - demo - Visual Studio Code
Copy.java 3
Copy.java > Copy > copy(String[], String[])
class Copy {
static String[] copy(String[] from, String[] to) {
for (int i = 0; i < from.length; i++) {
to[i] = from[i];
}
retu
}
}
int i - Copy.copy(String[], String[])
uncaught ArrayIndexOutOfBoundsException (uncaught ArrayIndexOutOfBoundsException)
uncaught java.lang.NullPointerException (uncaught java.lang.NullPointerException)
View Problem
No quick fixes available
WIGGLE LINES + PROBLEMS
PROBLEMS 13
DEBUG CONSOLE
TERMINAL
Filter (e.g. text, **/*.ts, !**/node_modules/**)
Copy.java 3
uncaught java.lang.NullPointerException (uncaught java.lang.NullPointerException) [Ln 3, Col 29]
uncaught ArrayIndexOutOfBoundsException (uncaught ArrayIndexOutOfBoundsException) [Ln 4, Col 13]
uncaught java.lang.NullPointerException (uncaught java.lang.NullPointerException) [Ln 4, Col 13]
master*
13 0 1
Ln 7, Col 19
Spaces: 4
UTF-8
LF
Java
Spell
Prettier

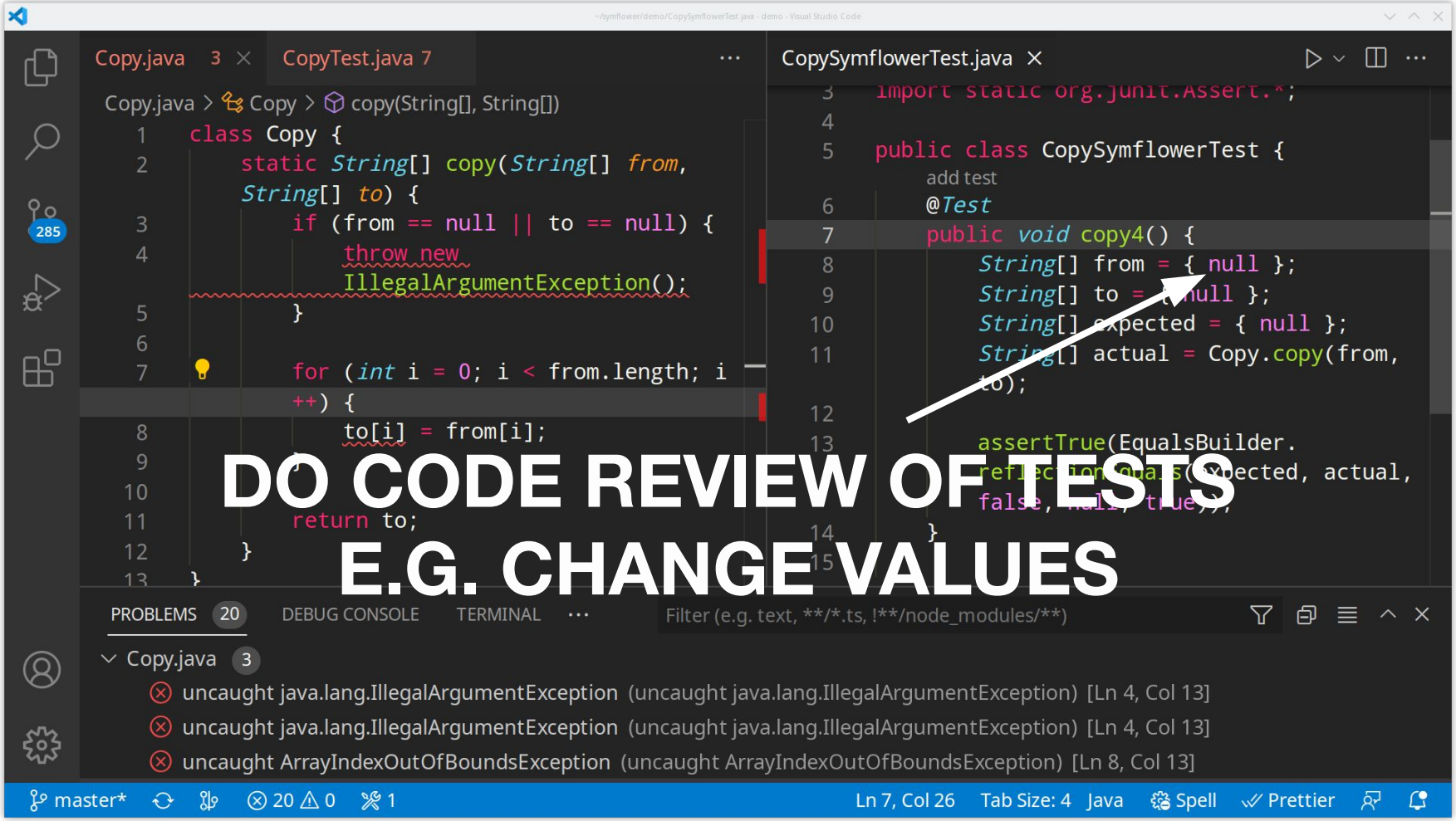

~/symflower/demo/Copy.java - demo - Visual Studio Code
Copy.java 3
>Symflower: Generate
Symflower: Generate Unit Tests for Function
recently used
Symflower: Generate Unit Tests for File
Copy.java > Co
class C
String[] to) {
for (int i = 0; i < from.length;
i++) {
to[i] = from[i];
}
return to;
}
}
rTest {
@Test // (expected = java.lang.
NullPointerException.class)
public void copy1() {
String[] from = null;
String[] to = null;
String[] actual = Copy.copy(from,
to);
}
add test
String[] from = {};
String[] to = null;
String[] expected = null;
REPRODUCER+SAFETY-NET
PROBLEMS 13
DEBUG CONSOLE
TERMINAL
Filter (e.g. text, **/*.ts, !**/node_modules/**)
Copy.java 3
uncaught java.lang.NullPointerException (uncaught java.lang.NullPointerException) [Ln 3, Col 29]
uncaught ArrayIndexOutOfBoundsException (uncaught ArrayIndexOutOfBoundsException) [Ln 4, Col 13]
uncaught java.lang.NullPointerException (uncaught java.lang.NullPointerException) [Ln 4, Col 13]
master*
13 0
1
Ln 3, Col 29
Spaces: 4
UTF-8
LF
Java
Spell
Prettier

How generated tests change software testing

2. **First implement then write tests**
   - Without generated tests
     - Manual labor + corner cases can be missed
     - Tests can be varying in quality
     - Manual maintenance
   - With generated tests
     - Previous advantages apply
     - Tests no longer need to be written manually
     - Do a code review for generated tests

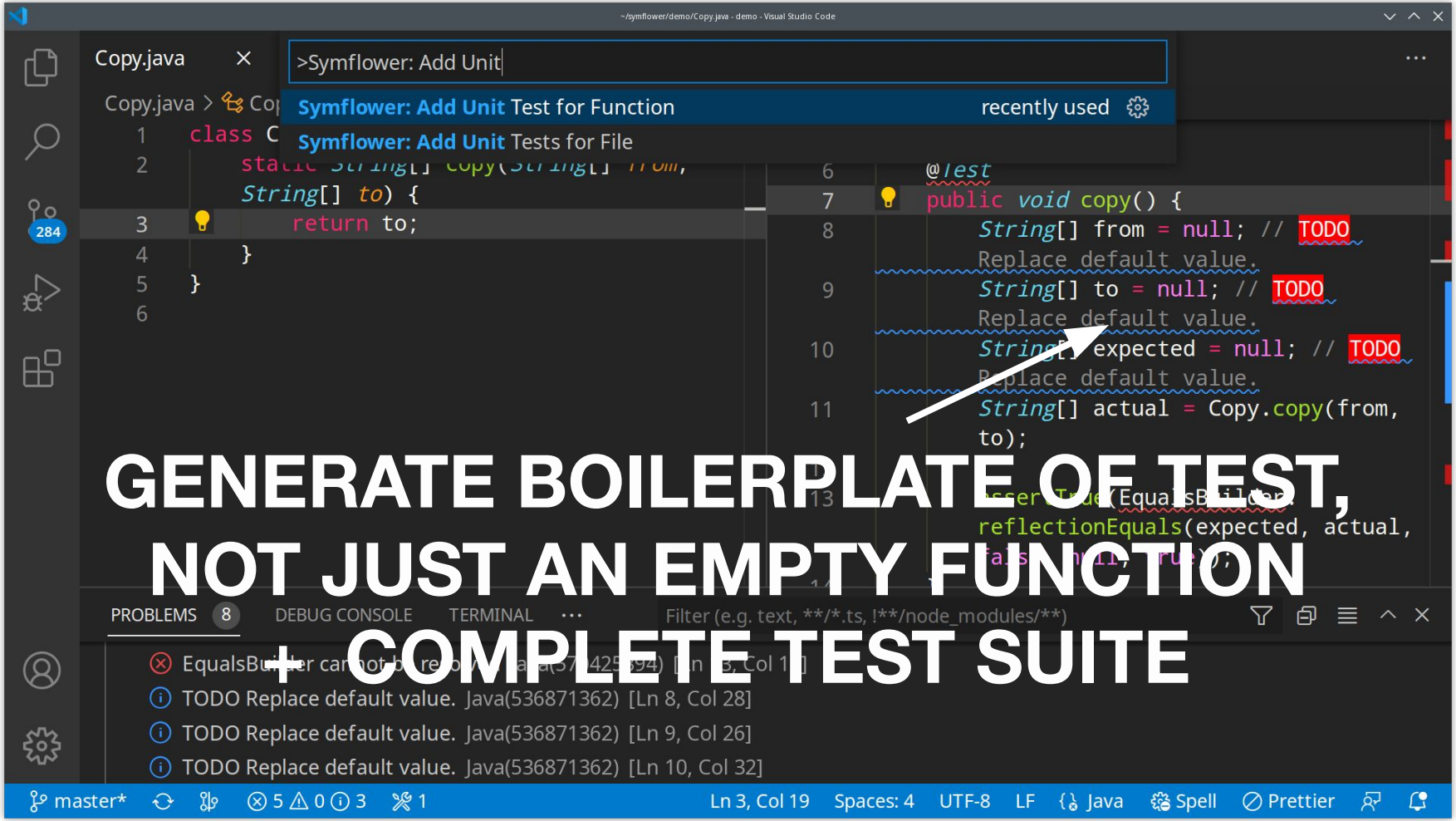


ADD ALL TESTS: DIFF OF BEHAVIOR OR ADD JUST SPECIFIC TESTS TO SUITE


Copy.java 3
CopyTest.java 7
Copy.java > Copy > copy(String[], String[])
class Copy {
static String[] copy(String[] from,
String[] to) {
if (from == null || to == null) {
throw new
IllegalArgumentException();
}
for (int i = 0; i < from.length; i
++) {
to[i] = from[i];
}
return to;
}
}
CopySymflowerTest.java
import static org.junit.Assert.*;
public class CopySymflowerTest {
add test
@Test
public void copy4() {
String[] from = { null };
String[] to = { null };
String[] expected = { null };
String[] actual = Copy.copy(from,
to);
assertTrue(EqualsBuilder.
reflectionEquals(expected, actual,
false, null, true));
}
DO CODE REVIEW OF TESTS
E.G. CHANGE VALUES
PROBLEMS 20
DEBUG CONSOLE
TERMINAL
Filter (e.g. text, **/*.ts, !**/node_modules/**)
Copy.java 3
uncaught java.lang.IllegalArgumentException (uncaught java.lang.IllegalArgumentException) [Ln 4, Col 13]
uncaught java.lang.IllegalArgumentException (uncaught java.lang.IllegalArgumentException) [Ln 4, Col 13]
uncaught ArrayIndexOutOfBoundsException (uncaught ArrayIndexOutOfBoundsException) [Ln 8, Col 13]
master*
20 0
1
Ln 7, Col 26
Tab Size: 4
Java
Spell
Prettier

How generated tests change software testing

3. **First write a test then implement (TDD)**
    - Without generated tests
        - Advantages of TDD! (e.g. nice testable APIs)
        - But, same regular disadvantages as 2.
        - Usually very happy-path heavy
    - With generated tests
        - Previous advantages apply
        - Boilerplate of tests can still be generated
        - Overlooked cases are reviewable



GENERATE BOILERPLATE OF TEST, NOT JUST AN EMPTY FUNCTION + COMPLETE TEST SUITE

symflower

get.symflower.com

hello@symflower.com

Let’s go to -> sli.do <- for questions