# Software Testing

Sibylle Schupp[1]

[1]Institute for Software Systems/Institut für Softwaresysteme
Hamburg University of Technology (TUHH)

Spring 2022

## Lecture 1

## Welcome!

http://www.sts.tuhh.de

Software Testing is brought to you by the

Institute for Software Systems

Lecturer

- Prof. Sibylle Schupp

Teaching assistant

- Sascha Lehmann

## When, where, & what

- Lectures: weekly
- Projects: throughout the term (6 deadlines)
- Course homepage
  - Stud.IP is a “learning platform”
  - Go to https://e-learning.tu-harburg.de
  - Login with your TUHH-username/password
  - Search for course “Software Testing”
- Check the Stud.IP page for
  - Project descriptions
  - News, announcements, . . .
- The course language is English.

## Project-based course

This course is entirely project-based. No exam.

- Your task is to test a real-world project:
    - Divided in a sequence of project phases
    - Defined along different testing methods
- **All work takes place during the lecture period**.
- Clear focus on writing software

## Outline

1 Terminology and test automation

## In-class exercise

Read carefully through the following code.

## In-class exercise: FindLast

## In-class exercise: LastZero

## In-class exercise: CountPositive

## In-class exercise: OddOrPos

## Outline

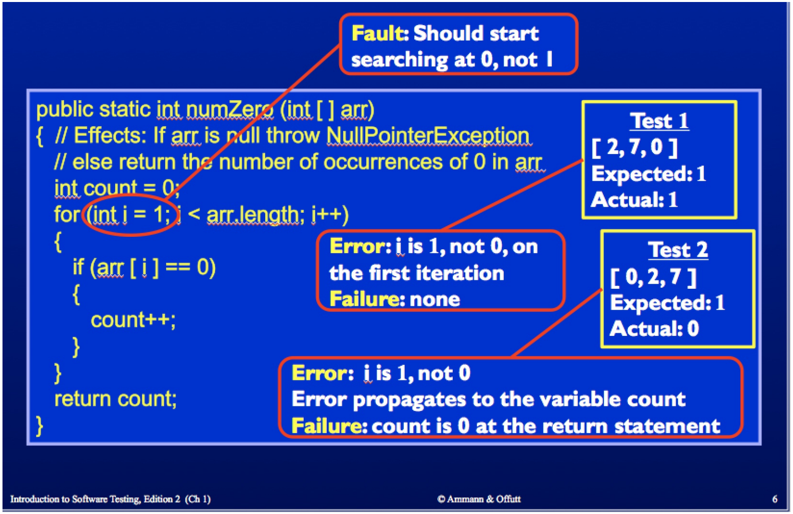
1 Terminology and test automation
- Terminology
- Test automation
- Organization

## Fault, error, failure

**Definition**

- A fault is a static defect in the code.
- A failure is incorrect, externally visible behavior of the software with respect to a (formal or informal) specification or description of the expected behavior.
- An error is an incorrect internal state, where a fault manifests itself.

## Example (fault, failure, error)



**Fault:** Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{  // Effects: If arr is null throw NullPointerException
   // else return the number of occurrences of 0 in arr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
   {
      if (arr [ i ] == 0)
      {
         count++;
      }
   }
   return count;
}
```

**Test 1**
[ 2, 7, 0 ]
Expected: 1
Actual: 1

**Error:** i is 1, not 0, on the first iteration
**Failure:** none

**Test 2**
[ 0, 2, 7 ]
Expected: 1
Actual: 0

**Error:** i is 1, not 0
Error propagates to the variable count
**Failure:** count is 0 at the return statement

Introduction to Software Testing, Edition 2 (Ch 1) © Ammann & Offutt 6

## Analogies

- A patient reports to the physician a set of symptoms ($\rightsquigarrow$ failure).
- The physician identifies the ailment ($\rightsquigarrow$ fault).
- The physician examines anomalous parameters such as blood values ($\rightsquigarrow$ error).

Where the analogy breaks down:

- The software fault results from a human mistakes.

## What's a bug, then?

*This thing gives out and then that. 'Bug'—as such little faults and difficulties are called—show themselves, and months of anxious watching, study, and labor are requisite before commercial success—or failure—is certainly reached. (Edison, 1878)*

- Term not well-defined.
- What is the meaning of "bug" in the following famous quote:

  *Testing shows the presence, not the absence of bugs (Dijkstra, 1970)*

## Correctness

- Failure: incorrect behavior (with respect to . . . )
- Correct behavior?
  - No natural laws, no mathematics
  - Requirements/specification only!
- Finding incorrect behavior
  - Software (behavior) complex
  - Abstractions needed
  - Model-driven test design

## From failures to faults

- Testing: software quality assessment by running the software and observing its execution.
- A test failure denotes the execution of a test that results in a failure.
    - Given a failure. Next step: find the underlying fault (debugging).
    - Yet, not every test effects a fault to manifest itself (in a failure).

## RIP model

The RIP model captures three necessary conditions so that a failure can be observed:

1. Reachability: the location(s) of the fault must be reached.
2. Infection: the state of the program must be corrupt.
3. Propagation: the infected state must effect an incorrect output or incorrect final state.

## Coverage criteria

- Complete testing is infeasible
  - Ex.: int foo(int A, int B, int C)
  - 32-bit: $4 \cdot 10^9$ values per parameter, $4^3 \cdot 10^{27}$ combinations
- Goal: find “good” input values
- Coverage:
  - Defines “good”
  - Guides the search through the input space
  - Helps to avoid overlap of tests

## Outline

1 Terminology and test automation
- Terminology
- Test automation
- Organization

## JUnit

https://junit.org/junit5

- JUnit is a popular framework for test automation in Java. It supports
  - the organizion of test prerequisites
  - the execution of tests
  - the comparison between expected and actual output
- *Unit tests* are tests of the functional requirements of methods and objects.
  - In contrast, *system tests* concern the behavior of the entire system.
  - System-level tests (of *performance, security, usability, . . .* ) concern non-functional properties.

## A simple example

https://junit.org/junit5/docs/current/user-guide/#writing-tests

```java
public class Calculator {
    // ...
    public int add(int a, int b) {
        return a + b;
    }
}
```

```java
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

class CalculatorTest {
    private final Calculator calculator = new Calculator();

    @Test
    void addition() {
        assertEquals(2, calculator.add(1, 1));
    }
}
```

## Example (writing JUnit tests)

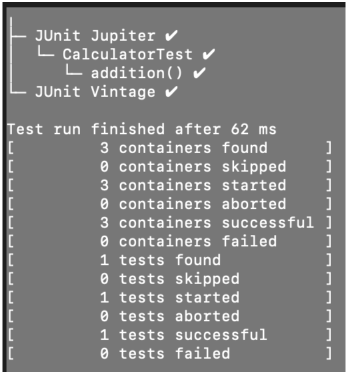
- Tests are Java classes that use JUnit 5 annotations.
  - In the example: `@Test`, `assertEquals`
  - For the full API, see: https://junit.org/junit5/docs/current/user-guide/#writing-tests-annotations
- For the file names of tests naming conventions exists so that tests can be found automatically
  - Ex. (Maven): https://junit.org/junit5/docs/current/user-guide/#running-tests-build-maven-filter-test-class-names

## Example (running JUnit tests)

```
javac Calculator.java
javac -cp junit-platform-console-standalone-1.8.0-SNAPSHOT.jar:.
    CalculatorTest.java
java -jar junit-platform-console-standalone-1.8.0-SNAPSHOT.jar --
    disable-ansi-colors --class-path . -c CalculatorTest
```

- Compile the class under test and the test class as usual with `javac`.
  - Make sure the `CLASSPATH` variable is set properly (or set `-cp`).
- JUnit 5 tests can be run from the command line (as above), using the console launcher.
  - Include in the `CLASSPATH` the jar file `junit-platform-console-standalone-<version>.jar`.
- Alternatively, JUnit tests can be run within various IDEs (IntelliJ, Eclipse, . . . ) and build systems (Maven, Cradle, Ant, . . . ).

## Example: JUnit 5 output



```
│
├─ JUnit Jupiter ✔
│  └─ CalculatorTest ✔
│     └─ addition() ✔
└─ JUnit Vintage ✔

Test run finished after 62 ms
[         3 containers found      ]
[         0 containers skipped    ]
[         3 containers started    ]
[         0 containers aborted    ]
[         3 containers successful ]
[         0 containers failed     ]
[         1 tests found           ]
[         0 tests skipped         ]
[         1 tests started         ]
[         0 tests aborted         ]
[         1 tests successful      ]
[         0 tests failed          ]
```

- Note the tree structure. The tests are in the leaves.
- Make sure that the tests are found by setting appropriate options.
  - Ex.: `--classpath . -c CalculatorTest`

## Example: assertions

https://github.com/junit-team/junit5/tree/master/documentation/src/test/java/

```
import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import org.junit.jupiter.api.Test;

class AssertionsTest {
    private final Person person = new Person("Jane", "Doe");
    @Test
    void dependentAssertions() {
        assertAll("properties",
            () -> {
                String firstName = person.getFirstName();
                assertNotNull(firstName);
                assertAll("first name", // dependent assertion
                    () -> assertTrue(firstName.startsWith("J")),
                    () -> assertTrue(firstName.endsWith("e"))
                );
            },
            () -> {
                String lastName = person.getLastName();
                assertNotNull(lastName);
})}; }
```

## Assertions in JUnit 5

- Test oracles are expressed using assertions.
  - Assertions can be named, grouped, nested.
  - Lambda expressions delay execution.
- Assertions need to be imported before they can be used. The assertion language is rich:
  - We have seen: `assertEqual`, `assertNotNull`, `assertAll`
  - Other assertions: `assertArrayEquals`, `assertThrow`, `assertDoesNotThrow`, `assertSame`, `assertTimeout`
  - For the complete API, see the documentation: `https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html`
  - JUnit 5 also allows for the use of assertions from JUnit 4.

## Exceptions

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;
import org.junit.jupiter.api.Test;

public class ExceptionTests {
    private final Calculator calculator = new Calculator();

    @Test
    void exceptionTesting() {
        Exception exception = assertThrows(
            ArithmeticException.class,
            () -> calculator.divide(1, 0));
        assertEquals("/ by zero", exception.getMessage());
    }
}
```

- Exceptions are tested using the `assertThrows` assertion.
  - Its arguments are the expected exception type and an executable.
  - Its return value is an exception. This exception can be used in other assertions.

## Outline

1 Terminology and test automation
- Terminology
- Test automation
- Organization

## Teaching method

In this class, we take a project-based approach.

- All deadlines during the term. Must work regularily during the term
- Must be able to attend the mandatory meetings.
- No exam. No peak performance required

## Project organization

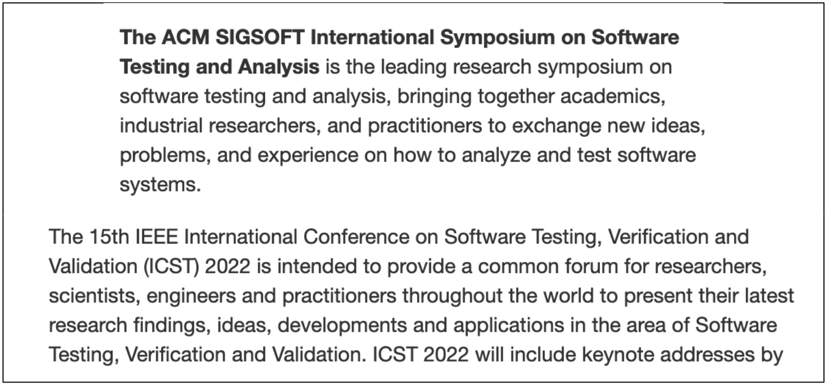
- The project phases should be done in a team.
  - Exception: each student must pass Project Phase 0 individually.
  - Register your group in StudIP.
  - Use the forum if you are looking for additional team members.
- You organize the tasks mostly independently. You are also responsible for setting up meetings in your group.
  - In the exercise session, you can ask for help and feedback.
- You upload your artifacts electronically.
  - Each project phase description specifies the form and **format** of the deliverables.
  - Submissions **must** meet that specification (will not be graded otherwise)
- You meet with the course assistant roughly bi-weekly.
  - The meeting is mandatory.
  - Those meeting are **the only mandatory part** of the course.
  - They take place during the exercise session.

## Prerequisites

**Coding-intensive** course at the **Master** level.

- Prerequisites in programming
    - Knowledge of object-oriented programming.
    - The subject system is in Java, the implementation language is Java.
- Prerequisites in software engineering/computer science
    - Project experience beyond homework assignments, e.g., through programming labs or small software projects
    - Knowledge of the fundamental data structures of computer science (stack, list, tables, tree, graphs)
    - Rudimentary knowledge of JUnit testing.
- All students with a IIW/CS Bachelor meet the prerequisites
    - Students with other degrees might have to catch up quickly
- Further, you must have the **time** to take the course

## Research component


**The ACM SIGSOFT International Symposium on Software Testing and Analysis** is the leading research symposium on software testing and analysis, bringing together academics, industrial researchers, and practitioners to exchange new ideas, problems, and experience on how to analyze and test software systems.

The 15th IEEE International Conference on Software Testing, Verification and Validation (ICST) 2022 is intended to provide a common forum for researchers, scientists, engineers and practitioners throughout the world to present their latest research findings, ideas, developments and applications in the area of Software Testing, Verification and Validation. ICST 2022 will include keynote addresses by

- One project phase is a research project. You are exposed to papers accepted by ISSTA or ICST, two leading conferences in the field and . . .
  - learn about scientific writing;
  - practice academic presentations and discussion.

## Grades

| Points | Grade |
|---|---|
| 51-55 | 4.0 |
| 56-60 | 3.7 |
| 61-65 | 3.3 |
| 66-70 | 3.0 |
| 71-75 | 2.7 |
| 76-80 | 2.3 |
| 81-85 | 2.0 |
| 86-90 | 1.7 |
| 91-95 | 1.3 |
| 96-100 | 1.0 |

## Grading

In the course of the term, each student accumulates points.

- You are graded individually, even if you work in a team.
  - Details vary per project phase, see the project phase sheets
- Mandatory meetings:
  - The entire group meets with the teaching assistant
  - Be prepared for practical and theoretical questions on the parts you "signed" with your name.
  - If you miss an arranged meeting, you will not get points for the project phase discussed.
    - **Exceptions**: you are excused for the reasons the examination office accepts, provided you can give proper proof (e.g., a note from the medical doctor).
    - If you miss a meeting excused, you can make up for it.
- Cheating: if you submit solutions that are not your own:
  1. $\boxed{\text{You lose \textbf{all} points earned up to that time}}$, including the ones for the submission in question.
  2. The originator loses **all points for that submission** if known to us.

## Deliverables

| | | out | due | points |
|---|---|---|---|---|
| 0. | Project Phase 0 | - | (W1) | 0 |
| 1. | Project Phase 1 | W1 | W3 | 20 |
| 2. | Project Phase 2 | W3 | W5 | 20 |
| 3. | Project Phase 3 | W5 | W7 | 20 |
| *Research* | *(Special Sessions)* | *W7* | *W10* | 20 |
| 4. | Project Phase 4 | W8 | W10 | 20 |
| 5. | Project Phase 5 | W12 | W14 | 20 |

- The table shows the maximal points for each deliverable.
  - The points add up to 120 points. For the final grade, the best 5 results count.
  - W1 refers to the first week of classes, W8 is the week after the Pentecost break. For the deadlines: see project phase descriptions.
- Deliverable 0 checks your prerequisites.

## Passing requirements

The final grade is based on the individual students' work.

A student passes the course if they meet the following requirements:

- Pass Deliverable 0.
- The sum of scores of the best 5 project phases is a passing score.

## What if . . . ?

- You missed the deadline for
  - . . . deliverable $\geq 1$ or want to achieve more points?
  - We put a late submission policy in place:
    - You may submit late (or resubmit) by a later deadline, but will miss points.
    - The late submission deadline is exactly one week after the official deadline. The maximal number of points is set down to 10 points.
    - For Deliverables 0,5 and the Research Component, there is no late submission.
  - For the final grade of a project phase, the last submission counts (even in case your performance worsened).
- . . . deliverable 0? This deliverable is mandatory. If you do not pass it, you cannot pass the course.

## What if . . . ? (part 2)

- You missed
  - . . . the group meeting with the teaching assistant with a proper excuse: a new meeting will be arranged
  - . . . without an excuse: the points of the project phase do not get counted
- If you fail the entire course, please be aware of the fact that the course is not offered in the winter.
  - Next possibility: next summer
  - **No exception possible**

## Textbook

PAUL AMMANN AND JEFF OFFUTT
INTRODUCTION TO
SOFTWARE
TESTING
Second Edition

## Lecture

- Main focus on chapters 5-9
  - Project parts aligned with lecture topics
- Lecture notes follow the slides from the textbook [AO]
  - Slides also available from the authors' webpage: https://cs.gmu.edu/˜offutt/softwaretest
  - Figures and slides from [AO] unless said otherwise

## Rules: no cell phones

## Material

Most material is available for download. For everything else, we keep the rights. Distribution of the material is not permitted. Remember that, by downloading, **you confirmed personally** not to pass on any material.

Author: https://de.freepik.com/fotos-vektoren-kostenlos/etikett

## Email!

- TUHH professors, assistants, and administration communicate by email with you.
- Please use your TUHH address (not the one of a private provider).

## References

- AO, Ch03
- AO, Ch01, Ch02, Ch04 for further reading
- http://junit.org/