Automated assessment of C++ exercises with unit tests

Workshop „Automatische Bewertung von Programmieraufgaben“

Tom-Michael Hesse, Axel Wagner,

Barbara Paech

Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
hesse@informatik.uni-heidelberg.de

Motivation: Direct feedback and lower effort

- C++ is subject to many programming lectures
- How to assess C++ programming exercises?
  - Currently, code is evaluated and corrected manually
  - We have CppUnit test cases for all exercises

Tutor

- What about automatic testing?
  - Available for Java
  - Students test more and benefit from instant feedback
  - Tutors can focus on programming style

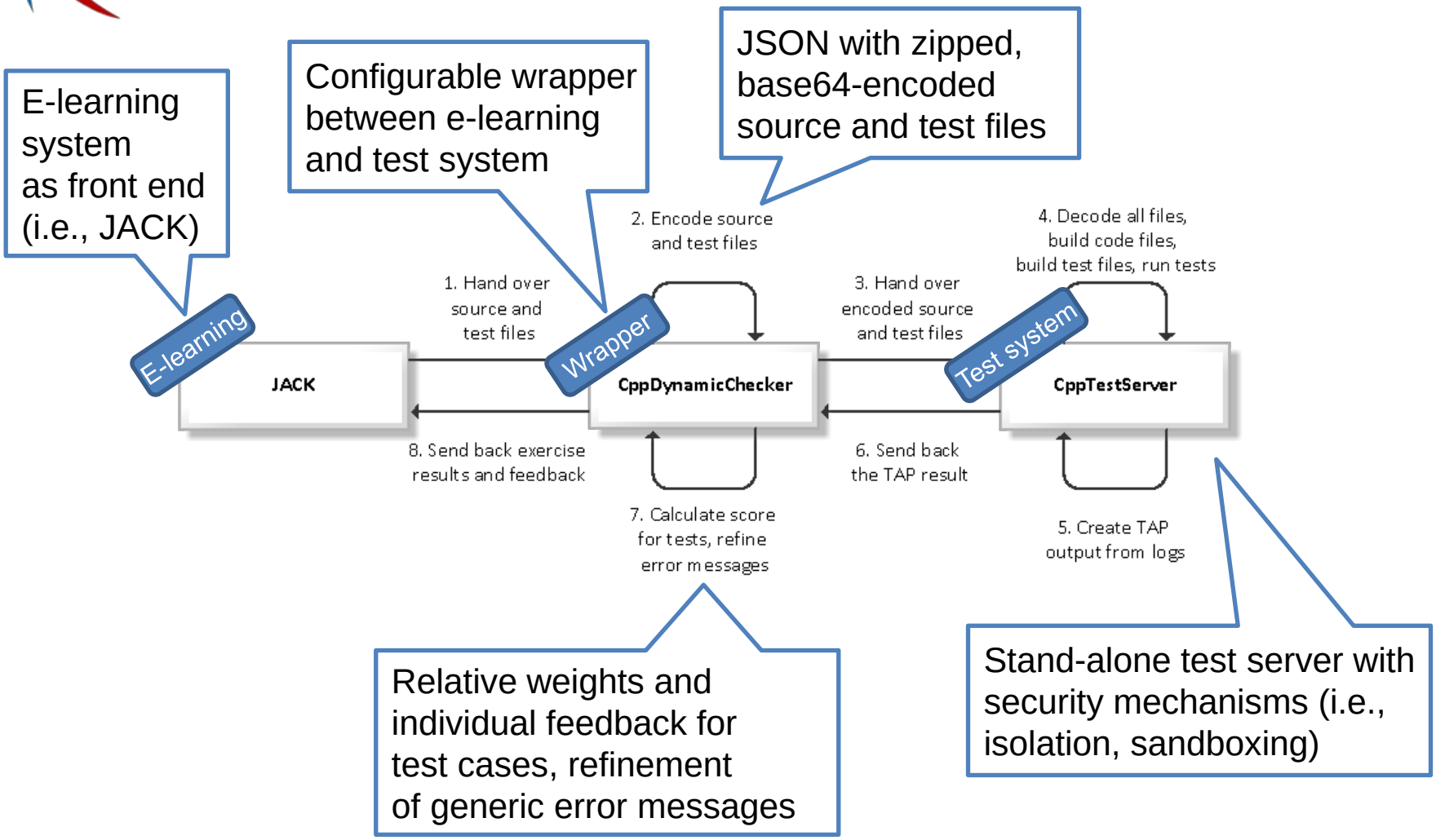
- But: Currently, there exists no system dynamically testing C++ exercises with unit tests (CI systems are not robust)

Challenge: How to test C++ automatically?

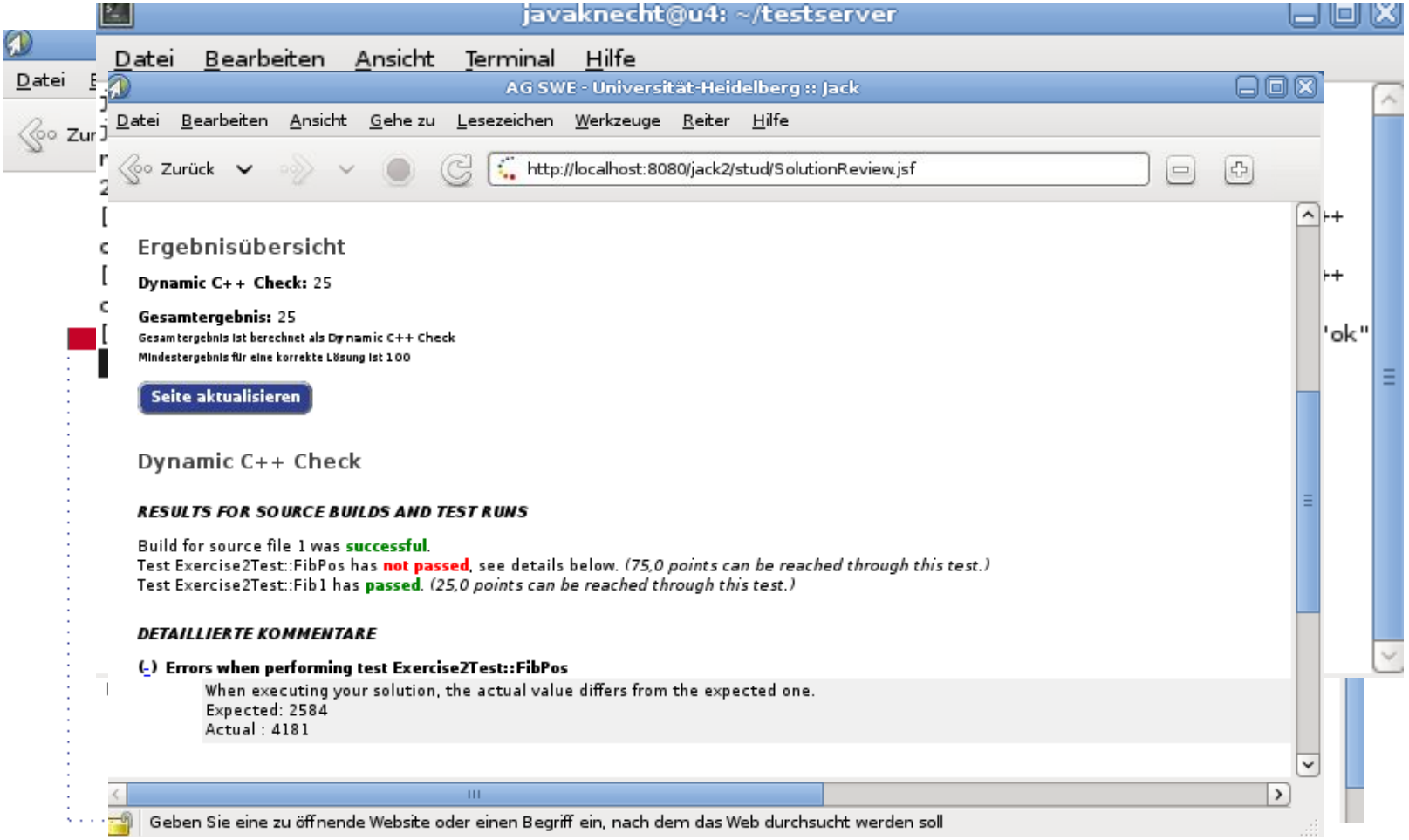
- Other universities only use scripts for particular aspects (building programs, log compiler messages)
- Testing C++ automatically is difficult, need for:
  - Secure build process (i.e., no JVM)
  - Suitable test cases and feedback
  - Automatic evaluation mechanism
  - Interoperability (exercise code, test system)

- Basic idea: Create a secure test system and integrate it with an e-learning system
  - Use e-learning system JACK [3] for management and presentation
  - Use CppUnit as test framework

Our approach: Interoperable test system



How does it look like?

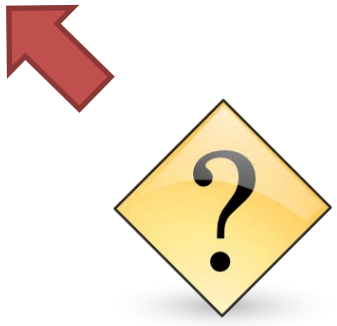


Example: Request to test system

```
{
    "suites": [
    {
         "name": "TestArray",
         "link": [ "Account", "Bank", "AccountsArray", "AccountsList",
    "TestArray" ]
    },
    {
         "name": "TestList",
         "link": [ "Account", "Bank", "AccountsArray", "AccountsList",
    "TestList" ]
    }
    ],
    "files": {
         "Account.cpp": "<gzipped, base64 kodierte Datei>",
         "AccountsArray.cpp": "<gzipped, base64 kodierte Datei>",
         ...
         "Bank.hpp": "<gzipped, base64 kodierte Datei>"
    }
}
```

Example: Response from test system

```
[{
   "name": "solution2_tests",
   "suite": {
        "ok": false,
        "tests": [
        {
        "description": "Exercise2Test::FibPos",
        "diagnostic": "equality assertion failed\nExpected: 2584\nActual:
   4181",
        "ok": false
        },
        {
        "description": "Exercise2Test::Fib1",
        "diagnostic": "",
        "ok": true
        }
        ]
   }, { ...
}]
```



Issues: Clean execution, malicious code

- Clean code execution
  - Separated temporary directory for each solution
  - Independent build process for each solution
  - Sequential test suite execution for each solution

- Build or test process manipulation: Malicious code
  - Usage of EasySandbox [4] (SECCOMP implementation as shared library)
  - Limited system calls (just read/write, exit, sigreturn)
  - Memory protection by heap limit (malloc is overwritten)
  - Loaded via LD_PRELOAD in test runner

Issues (ctd.): Runtime attacks

- Build or test process manipulation: Runtime attacks
  - Configurable limit for execution time
  - Unexpected termination will cause empty test results and no points

Open questions: Build differences, feedback

- C++ standard is not completely strict
  - Example: Size definitions of some data types are ranges
  - Differences between student systems and test system (i.e., hardware architecture, compiler) might exist
  - Program, which builds and runs on local machine, might not work within the test system

- Explanation of test cases is tough
  - Students only learn through feedback they understand
  - Test cases itself should be explained
  - Feedback messages should be closely related to the current test case

Current work

- Evaluation of our test system during lecture “Einführung in die Praktische Informatik”
  - ~500 students with weekly programming exercises
  - Focus on functional programming, typical constructs and OO

- Integration of all components for productive use
- Further implementation for test case explanation

- Current version of test system available at GitHub [5]
- Current version of integrated components will be available soon

Literature

(1) A. Hermanns, V. Jaenen, A. Heide and K. Henning: “ClaRA (C++ learning at RWTH Aachen) Change from classical teaching to e-learing”, in *Proceedings of the 7th International Conference on Information Technology Based Higher Education and Training (ITHET '06)*, 2006, pp. 185 – 190

(2) S. Naser: “Evaluating the Effectiveness of the CPP-Tutor, an Intelligent Tutoring System for Students Learning to Program in C++”, in Journal of Applied Sciences Research, 2009, vol. 5, no. 1, pp. 109 – 114

(3) JACK, http://www.s3.uni-duisburg-essen.de/research/jack.html

(4) EasySandbox, http://github.com/daveho/EasySandbox

(5) Test system, https://github.com/Merovius/bor

Tom-Michael Hesse
Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
hesse@informatik.uni-heidelberg.de