TESTING

Unit Tests and Beyond

24.10.2023 I JAKOB FRITZ I TIME-X HACKATHON DARMSTADT

JÜLICH
Forschungszentrum

OVERVIEW

Continuous Integration

- What is CI?
- Why is CI important?
- Other parts of CI apart from testing

Scopes of tests

- Unit Tests
- Integration Tests
- End-to-End Tests

Strategies for tests

- Golden Master tests
- Property based testing
- Fuzzy testing
- Mutation testing

JÜLICH Forschungszentrum

CONTINUOUS INTEGRATION

What is it?

- Automized jobs that run regularly (“continuously”) at your code (e.g. at every push; every day; …)
- Continuous Testing as part of Continuous Integration
- Other parts of CI/CD (Continuous Integration / Continuous Deployment) not focus of this talk

CONTINUOUS INTEGRATION

Why is it important?

Automizing has multiple advantages:

- Identical for everyone (regardless, who pushes to the server)
  - More reproducible
  - Less error-prone
- Cannot be forgotten to run

CONTINUOUS INTEGRATION

Other parts of CI apart from testing

- **Compiling**: Creating an executable version of your code (if required)
- **Linting**: Static analysis of your code. Often fast, as no compilation / execution is needed. Can find pitfalls.
- **Auto-Formatting / Style checking**: Check whether the code satisfies a certain style. This increases readability and maintainability across developers (and maybe your future self)

OVERVIEW

Continuous Integration

- What is CI?
- Why is CI important?
- Other parts of CI apart from testing

Scopes of tests

- Unit Tests
- Integration Tests
- End-to-End Tests

Strategies for tests

- Golden Master tests
- Property based testing
- Fuzzy testing
- Mutation testing

SCOPES OF TESTS

Why testing?

Reason for testing:

➔ Finding bugs

Reason for finding bugs:

➔ Making the user happy (generally) / making the results reproducible (in science)

So what makes a user happy / the results reproducible?

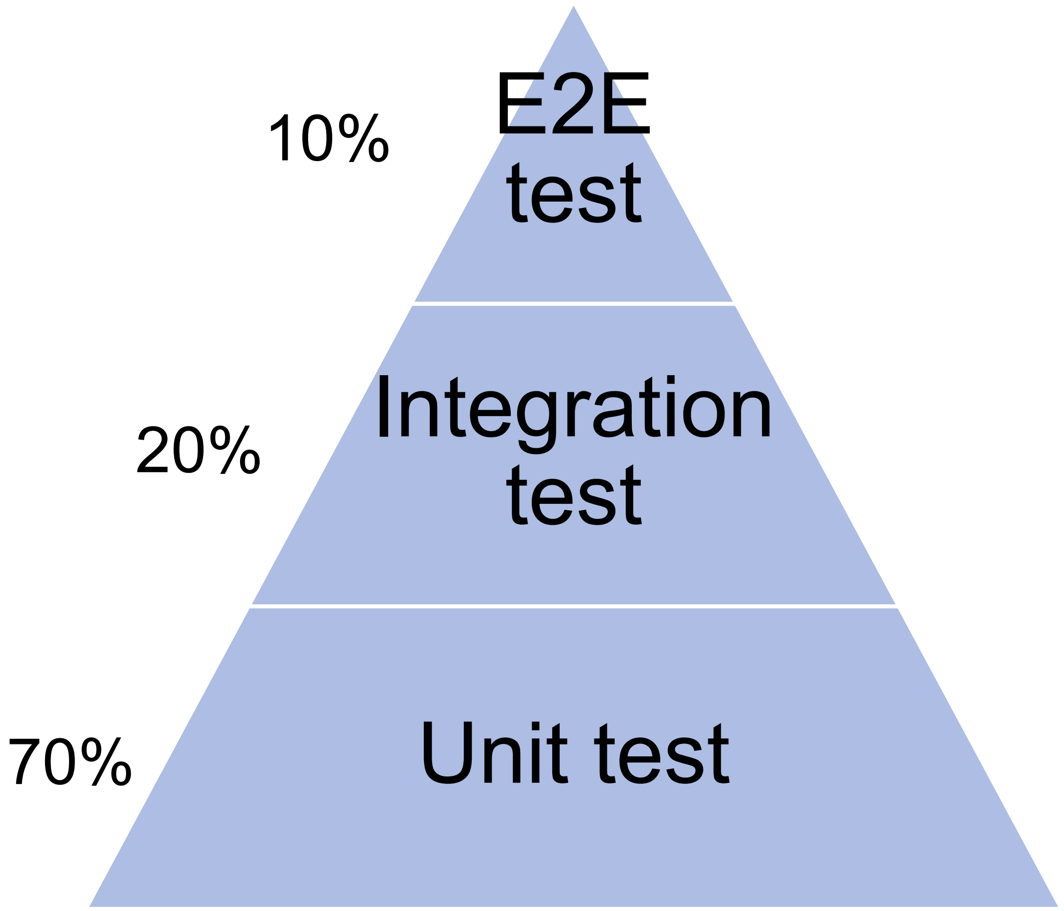
Test added ➔ Test fails ➔ Bug reported ➔ Bug fixed

😑 😑 😑 🙂

Based on: https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html

SCOPES OF TESTS

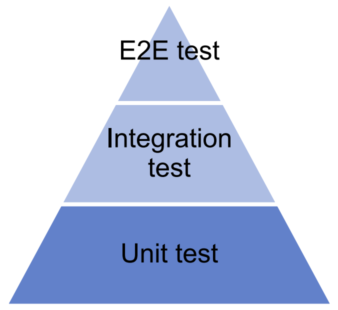
	Unit test	End-to-End test
Fast	🙂	🙁
Reliable	🙂	🙁
Isolates failures	🙂	🙁
Simulates a real user	🙁	🙂



Based on: https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html

SCOPES OF TESTS

Unit tests

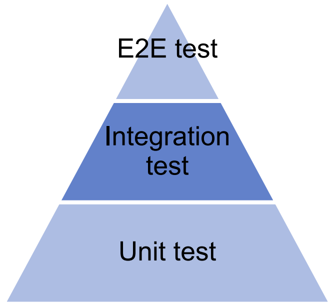


- Idea: Test a single function
- Fast execution & easy to locate bugs
- Ideally hermetic tests
- Most of the tests should be Unit tests (~70%)

JÜLICH Forschungszentrum

SCOPES OF TESTS

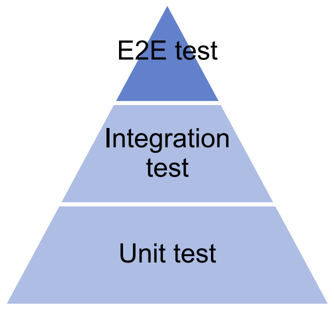
Integration tests



- Idea: Test combination / interaction of functions (usually only a few; often only 2)
- Slower execution compared to Unit tests and harder to use to localize bugs
- Either using Mock-ups or real other components
- Can induce flakiness (as relying on other components; network; …)
- Should be fewer tests than unit-tests (~20%)

SCOPES OF TESTS

End-to-End tests



- Idea: Test whole Software/system
- Even slower execution compared to Unit and Integration tests
- Harder to localize bugs
- Not hermetic (by definition)
- Should be the fewest tests (~10%)

OVERVIEW

Continuous Integration

- What is CI?
- Why is CI important?
- Other parts of CI apart from testing

Scopes of tests

- Unit Tests
- Integration Tests
- End-to-End Tests

Strategies for tests

- Golden Master tests
- Property based testing
- Fuzzy testing
- Mutation testing

JÜLICH Forschungszentrum

STRATEGIES FOR TESTS

Golden Master testing

- What it is:
  - Classic approach
  - Providing input and expected output & comparing real to expected output
- When to use it:
  - To test specific cases (e.g. examples)
  - To test complex cases when it is hard to specify all details (e.g. complex input files)
- Downsides:
  - Limited test scope
  - When using files: watch out for timestamps

- How to use it:
  - Prepare input and output (variables or files)
  - Start function with given input
  - Check if created output equals expected output
- Examples:
  - `assert sum(2,3)==5`

  -
    ```
    create_db()
    assert new.db == prepared_example.db
    ```

STRATEGIES FOR TESTS

Property based testing

- What it is:
  - Check not for specific output, but for properties of the output
- When to use it:
  - To generalize test cases
  - To find edge-cases
- Downsides:
  - Difficult when creating complex data-structures
  - An addition rather than replacement for golden master tests (so more effort, but not more line coverage)

- How to use it:
  - Define properties of input
  - Start function with (automatically) created input
  - Check if output satisfies checks
- Examples:

```
@given(list(characters()))
def TestAmazingSort(input):
    output = AmazingSort(input)
    assert set(input) == set(ouput)
    assert isSorted(output)
```

Further reading: https://hypothesis.works/articles/what-is-property-based-testing/
https://en.wikipedia.org/wiki/QuickCheck

JÜLICH Forschungszentrum

STRATEGIES FOR TESTS

Fuzzy testing

- What it is:
    - Fuzzy testing throws arbitrary input at your function to see if the function returns unexpected errors
    - Similar to property based testing, but normally wider input and less precise output check
- When to use it:
    - To test functions for robustness against user- or interaction errors
    - To find edge cases / strange bugs nobody anticipated and tested for
- Downsides:
    - Rather a smoke test
    - Not testing for correctness, but only for failures

Further reading: https://hypothesis.works/articles/what-is-property-based-testing/
https://en.wikipedia.org/wiki/American_fuzzy_lop_(fuzzer)

JÜLICH Forschungszentrum

STRATEGIES FOR TESTS

Mutation testing

- What it is:
  - “Mutation testing is a technique for systematically mutating source code in order to validate test suites. It makes small changes to a program's source code and then runs a test suite; if the test suite ever succeeds on mutated code then a flag is raised” (https://www.oreilly.com/pub/e/3560)
  - “Essentially, mutation testing is a test of the alarm system created by the unit tests.” (mutatest.readthedocs.io/en/latest/install.html#mutation-trial-process)
- What it does it:
  - Alter your code and check if tests now fail
- When to use it:
  - When added many (unit) tests to have high coverage
  - When unsure how well the tests actually test the code
  - To see if tests are sensitive enough to detect (unintended) changes in the code
- Packages to use (not tested by me):
  - Mutatest: https://mutatest.readthedocs.io/en/latest/ (python)
  - Mutmut: https://github.com/boxed/mutmut (python)

SUMMARY

Continuous Integration

- Easier than manual
- More reproducible

Scopes of tests

- Focus on Unit Tests
- A few Integration Tests
- Very few End-to-End Tests

Strategies for tests

- Compare precise results
- Check properties
- Test for raised errors
- How precise are your tests

SUMMARY

Thank you for your attention!

I’m happy to answer questions!

Feel free to reach me: j.fritz@fz-juelich.de

JÜLICH Forschungszentrum