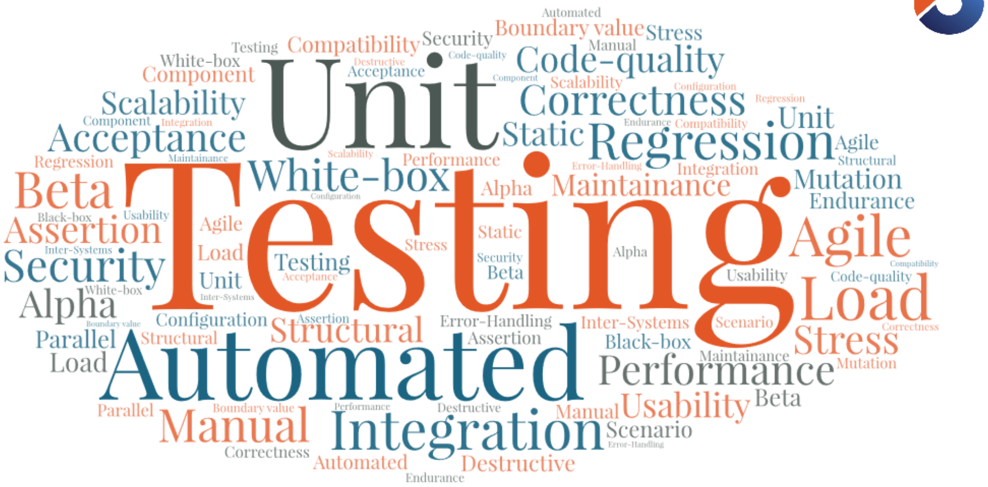
Technische Universität Braunschweig

Suresoft
SUSTAINABLE RESEARCH SOFTWARE



HeFDI Code School

Introduction to Software Testing

Harikrishnan Sreekumar and Yannik Hüpel, 24th November 2023

Workshop objectives

- Familiarize with testing concepts from a research software perspective
- How to incorporate testing in our code development routines
- Capable of coding basic unit tests, acceptance tests and code quality checks
- Know and aim towards test automation

Agenda

Motivation for testing research software

Role of software testing for sustainable development

Fundamentals of software testing

Major types of software testing

- Unit tests and code coverage + Hands-on with PyTest + Break
- Acceptance tests + Hands-on with PyTest + Short Break
- Code quality tests + Hands-on with PyLint

Test process automation with GitLab CI

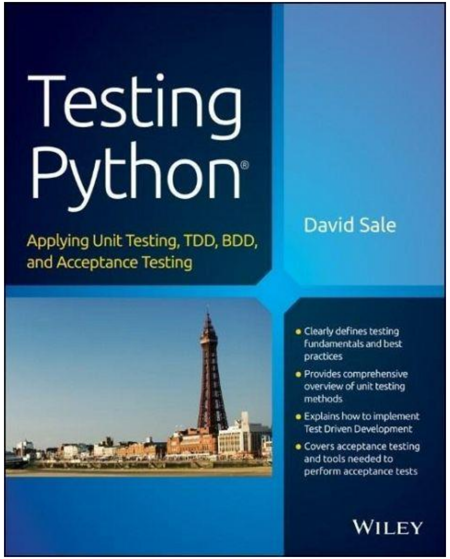
Demonstration of the in-house code elPaSo’s test environment

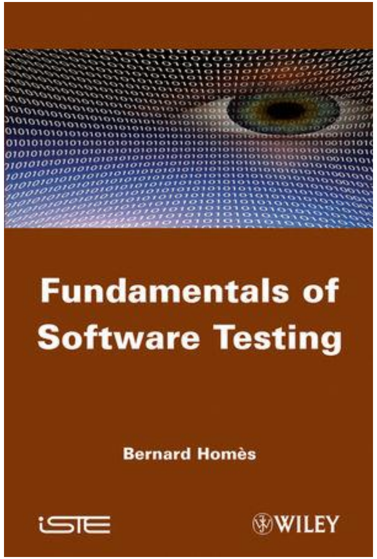
Information

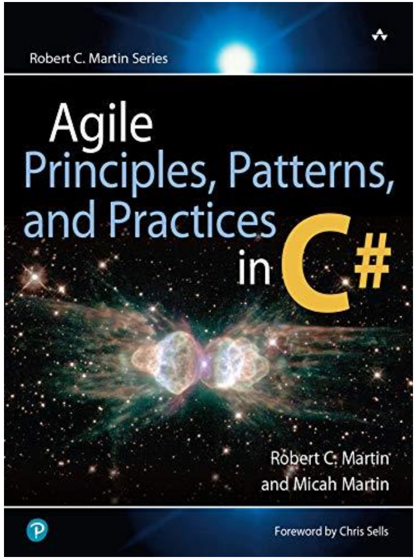
- Workshop slides and codes in Zenodo
- Workshop preparation
  - Live sharing via VS Code
  - Running locally? – python with numpy, pytest and pylint
- We look forward to your questions and experiences – please unmute and interrupt anytime during the workshop or post in chat
- We use python as our standard language
- We use the main room for our hands-on session – no break-out rooms
- Planned breaks after every hour
- More documentation in

https://suresoft.dev/knowledge-hub/software-testing/

Literature recommendation


Testing Python®
David Sale
Applying Unit Testing, TDD, BDD, and Acceptance Testing
• Clearly defines testing fundamentals and best practices
• Provides comprehensive overview of unit testing methods
• Explains how to implement Test Driven Development
• Covers acceptance testing and tools needed to perform acceptance tests
WILEY




Robert C. Martin Series
Agile Principles, Patterns, and Practices in C#
Robert C. Martin and Micah Martin
Foreword by Chris Sells

Motivation for testing in science

Motivation | Software testing? Why?

[https://www.codementor.io]

Motivation | Causes of hidden software bug - An example

Ariane 5 – The Worst Software Bugs in History

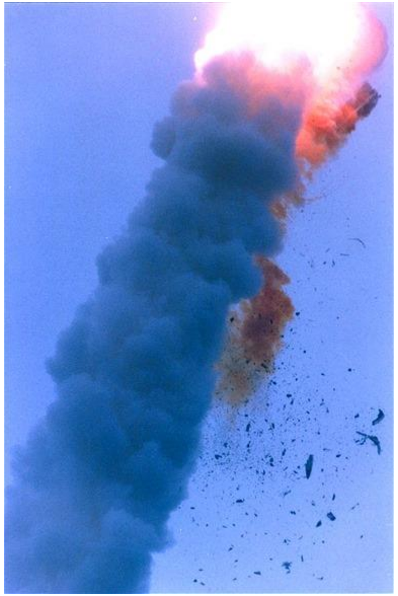


Photo source: https:// www.esa.int
Article: https://www.bugsnag.com/blog/bug-day-ariane-5-disaster

Motivation | Necessity for software testing

- First step towards code sustainability
- Ensures and documents the correct behaviour of a software
- Contributes to the overall software quality
- Quickly identifies defects/bugs in a developing code → save time for debugging

“Scientists spend 57% of the time finding and fixing bugs”

[P. Prabhu et al., A Survey of the Practice of Computational Science, 2011]

Motivation | Survey results

- **24.14%** do not consider testing because of lack of time
- **27.59%** add tests to old codes
- **41.38%** miss sufficient knowledge for testing

Role of software testing for sustainable development

In Academia

PhD Researcher 1 → Develops feature A ✓

PhD Researcher 2 → Develops feature B ✓

Does feature A still work? ✗

PhD Researcher 3 → Develops feature C ✓

Do features A and B still work? ✗

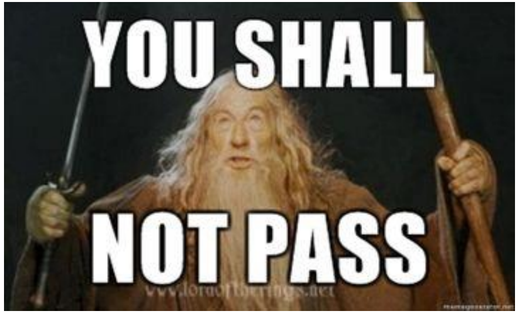
[FreeDigitalPhotos.net]

Software testing – powerful tool to …

- Localize bugs in your large code base
- Get immediate feedback on your new code integration
- Document stable behaviour of your software
- Enhance code credibility

- All the above, in large groups – define your standards centrally
- Ensure a stable release at all times

- Towards sustainable development!
- Main component in a continuous integration framework is testing



[https://browsee.io/blog/]

Reduce unnecessary bugs!

Fundamentals of software testing

“Software testing shows the presence of bugs, not their absence”

[https://www.hexacta.com/testing-in-software-more-than-finding-bugs/]

Definition of software testing

“Software testing is a set of activities with the objective of identifying failures in a software or system and to evaluate its level of quality.”

[B. Homès: Fundamentals of Software Testing. 2012]

“Software testing is the process of executing a program with the intend of finding errors.”

[J. M. Myers et al.: The Art of Software Testing. 2011]

“Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do.”

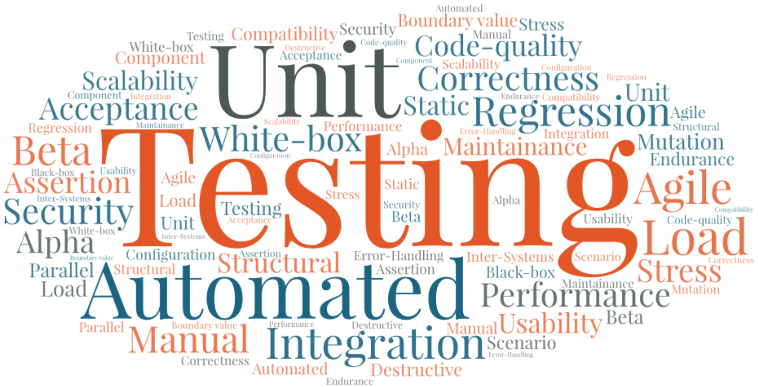
[IBM: What is software testing? https://www.ibm.com/topics/software-testing]

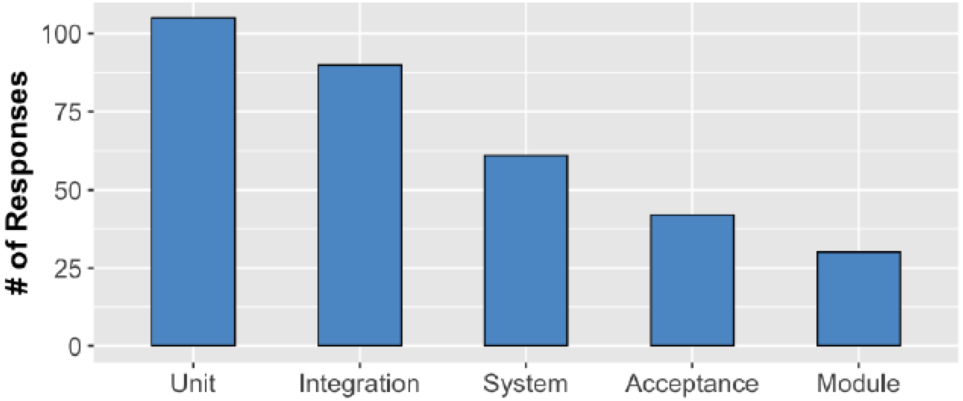
Types of software testing

150+ types of tests and still increasing…

→ But, what are relevant for research software?

Software testing for research softwares



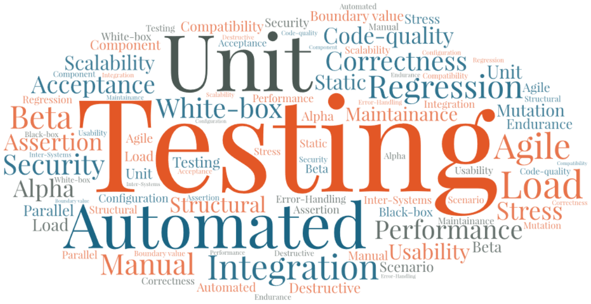


Commonly used testing in research softwares

[N. U. Eisty, et al.: Testing Research Software: A Survey. 2022]

Workshop focus → Unit testing, acceptance testing, code-quality testing

Functional and non-functional tests



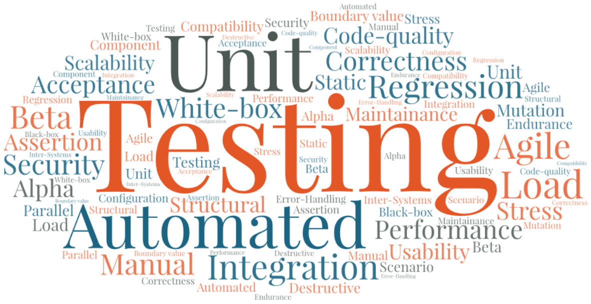
Functional tests

- Focus on the proper functioning of the software and it‘s components
- Example: Correctness/accuracy (unit-, acceptance testing)

Non-functional tests

- Focus on the non-functional aspects like performance, software‘s usability, code quality, stability, testability, adaptability, portability, etc.

Manual and automated testing



Manual testing

- Oldest methods
- Typically done by a QA tester (black-box)
- Tests different features of the software

Automated testing

- Most efficient – faster and more aspects are tested
- Main component of continuous integration and deployment
- Typically done by the developer with the help of testing tools (white-box)

→ Focus of this workshop

Technische Universität Braunschweig

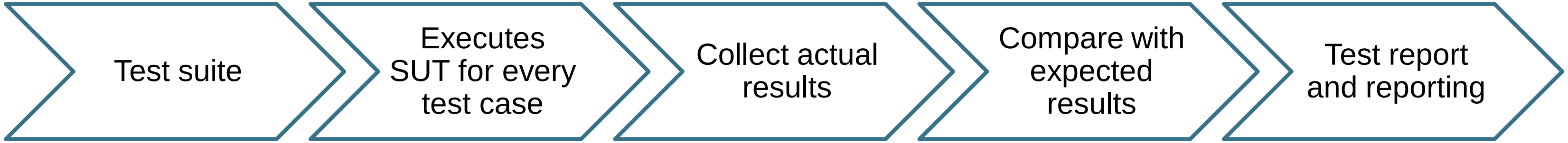
Suresoft
Sustainable Research Software

Typical automated software testing framework

System under test (SUT)
Software itself or software components

Test suite: Test cases + Expected results
Benchmarked software‘s expected behaviour and test specification

Test reports
Test status (Success/Failed) with test measures

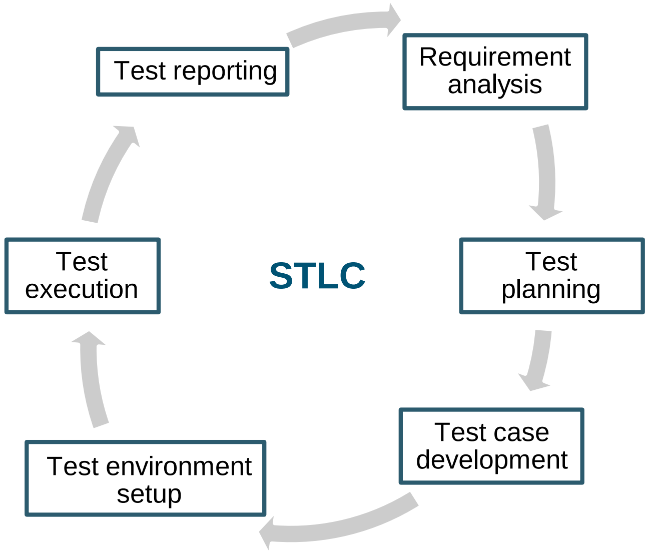

Test suite
Executes SUT for every test case
Collect actual results
Compare with expected results
Test report and reporting

Test harness

How to incorporate testing in practice?

Software testing life-cycle (STLC)

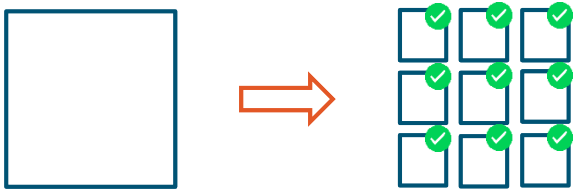
- Testing is always present in a software development cycle
- Sequential/Iterative/Incremental methodology to achieve a level of quality
- Agile model example: Test driven development (TDD) → Workshop on TDD



Unit testing and Hands-on session

Unit testing

- Tests a code at its basic level
- Codes are isolated - according to their specific functionalities - into smaller units and tested for proper operation

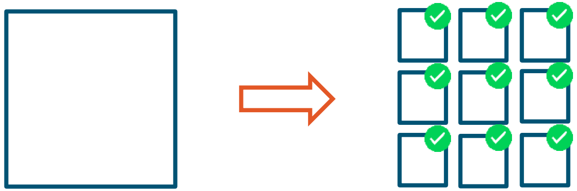


“Unit testing is more of an act of design than of verification. It is more of an act of documentation than of verification.”

[R. C. Martin and M. Martin: Agile Principles, Patterns and Practices in C#. 2006]

How to incorporate unit-testing?

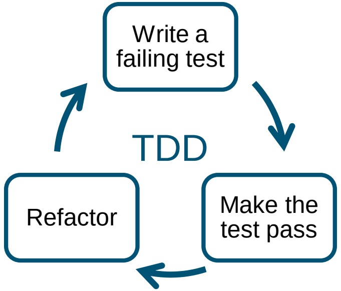
- Existing code? Breakdown into very small functions



- Writing new code? Easy! Follow a unit-testing methodology from the very start.

→ Test Driven Development (TDD)

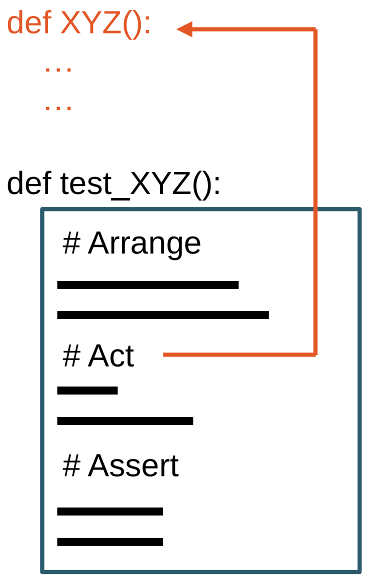
[K. Beck: Test-Driven Development. 2002] [TDD Workshop]



How to write tests? The AAA Pattern

- Three A’s: Arrange, Act and Assert
- Added advantage is that the tests are easily readable

- Arrange : Requirements to test the functions are prepared
- Act : Function under test is called and output is collected
- Assert : The expected operation of the function is checked



```
def XYZ():
    …
    …

def test_XYZ():
    # Arrange

    # Act

    # Assert
```

Tools for unit-testing

Python

- pytest [https://docs.pytest.org/en/stable/]
- unittest [https://docs.python.org/3/library/unittest.html]

C++

- GoogleTest [https://github.com/google/googletest]

Assertions

- Assertions checks whether the outcome meet certain expectations
- Boolean expression: true means assertion success and false means assertion fail
- Does sanity check – checks if certain assumptions are valid
- Great for documentation, debugging and testing

PyTest uses python’s standard `assert`:

- `assert 1 == 1 # success`
- `assert “Hello” == “Hallo” # fails`
- `assert 3.14159265359 == pytest.approx(3.14, 1e-3) # success`

Test metrics: Code coverage

- Analysis method which determines the amount of code executed by a test suite and which are not.
- We aim for the best code coverage with unit testing
- Code coverage types:
  - Functional coverage : how many functions are tested
  - Branch coverage : how many execution paths are tested
  - Line/statement coverage : how many lines of code/statements are tested
- Coverage tools

Python

- `coverage` [https://coverage.readthedocs.io/en/6.4.2/]

C++

- `GNU gcov + lcov` [https://gcc.gnu.org/onlinedocs/gcc/Gcov.html, https://github.com/linux-test-project/lcov]
- `Intel codecov + profmerge`

Hands-on | Writing your first unit test

Example project – Matrix Calculator

- Perform basic matrix operations: **Add, Multiply, Inverse**
- Can handle different matrix format: **Dense**
- Can handle user-written linear solvers: **Jacobi iterative solver**

Hands-on | Writing your first unit test

Example project – Matrix Calculator

```
|- src
    |-- MatrixAlgebra
          |--- dense_matrix.py
```

Matrix Addition add(A, B)	$C = A + B$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$
Matrix-Vector Multiply matrix_vector_multiply(A, b)	$c = A \times b$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}$
Matrix Inverse matrix_inverse(A)	$C = A^{-1}$	$\begin{bmatrix} 10 & 1 \\ 3 & 8 \end{bmatrix}^{-1} = \begin{bmatrix} 0.1039 & -0.0130 \\ -0.0390 & 0.1299 \end{bmatrix}$

Hands-on | Writing your first unit test

Example project – Matrix Calculator

```
|- src
    |-- MatrixAlgebra
          |--- dense_matrix.py
    |-- MatrixSolver
          |--- jacobi_solver.py
```

How to start with unit-testing?
→ Demonstration

Solve

solve(A, b)

$$c = A^{-1}b$$

$$\begin{bmatrix} 10 & 1 \\ 3 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 0.6233 \\ 0.7662 \end{bmatrix}$$

jacobi_solver uses the dense_matrix functionalities

Hands-on | Writing your first unit test

Start testing and increase code coverage to 100% | 20 minutes

- Write 3 unit tests to test the functions `add, matrix_vector_multiply, matrix_inverse` implemented in `dense_matrix.py | XX% CC`
- (Optional) Write additional tests where matrix entries are float values and use `pytest.approx | XX% CC`

Matrix Addition add(A, B)	$C = A + B$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$
Matrix-Vector Multiply matrix_vector_multiply(A, b)	$c = A \times b$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}$
Matrix Inverse matrix_inverse(A)	$C = A^{-1}$	$\begin{bmatrix} 10 & 1 \\ 3 & 8 \end{bmatrix}^{-1} = \begin{bmatrix} 0.1039 & -0.0130 \\ -0.0390 & 0.1299 \end{bmatrix}$

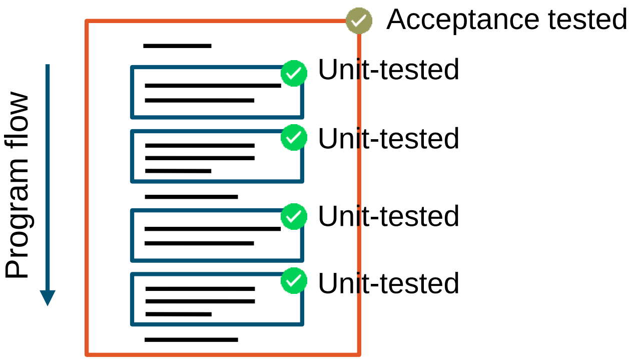
Acceptance testing and Hands-on session

Why unit testing is not enough…

[https://www.aligneddev.net/]

Acceptance testing

- Tests the application as a whole and ensure proper operation
- Acceptance testing perform verification
- Documentation of stable application state and execution
- Black box testing



"If unit testing verifies that the code does exactly what the programmer expects it to do, then acceptance testing verifies that the code does what the user expects it to do."

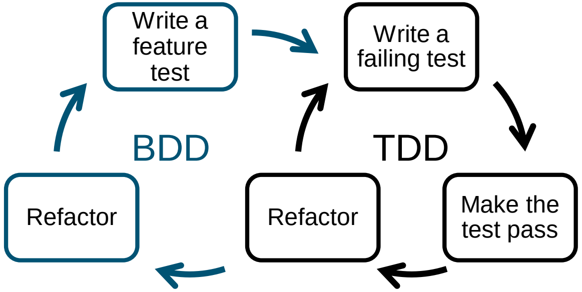
[D. Sale: Testing Python: Applying Unit Testing, TDD, BDD and Acceptance Testing. 2014]

How to incorporate acceptance-testing?

- Design specific test cases which executes certain features of the application
- Aim for maximum code coverage with the various test cases
- Follow an acceptance-testing methodology in your development life-cycle

→ Behavior Driven Development (BDD)

[D. Sale: Testing Python: Applying Unit Testing, TDD, BDD and Acceptance Testing. 2014]



We will still use the AAA pattern!

Tools for acceptance-testing

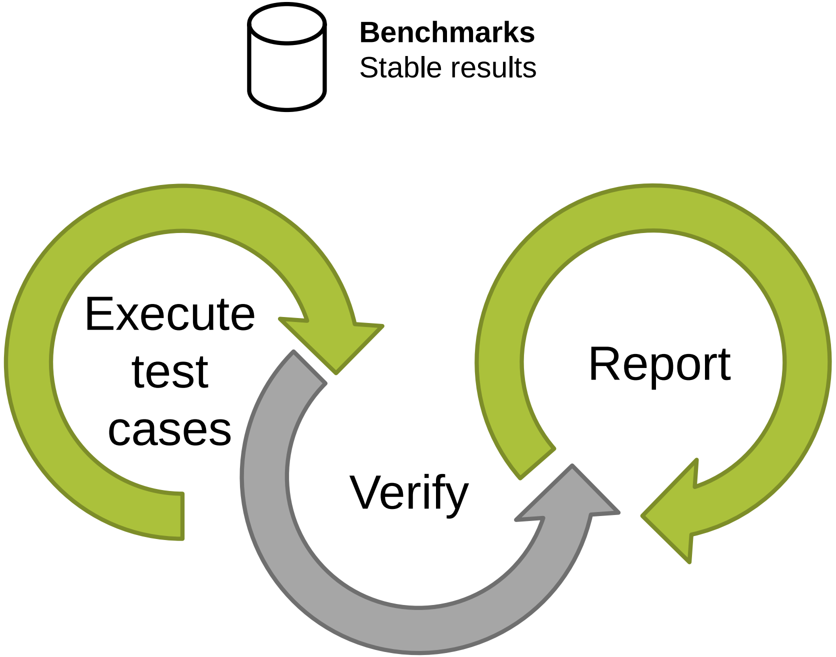
Python

`pytest`	[https://docs.pytest.org/en/stable/]
`robot`	[https://robotframework.org/]

…

`fieldcompare`	[https://gitlab.com/dglaeser/fieldcompare/]
automate	[https://git.rz.tu-bs.de/akustik/elPaSo-AUTOMATE]
Custom made ???	

Typical workflow for acceptance testing



Hands-on | Writing your first acceptance test

Example project – Matrix Calculator

```
|- main.py
```

How to start with acceptance-testing?
→ Demonstration

Case “add” py main.py --add	Performs addition of supplied matrix data
Case “solve” py main.py --solve	Performs solving of supplied matrix data
Case “default” py main.py xyz	Exits with a failure code: exit(-1)

Hands-on | Writing your first acceptance test

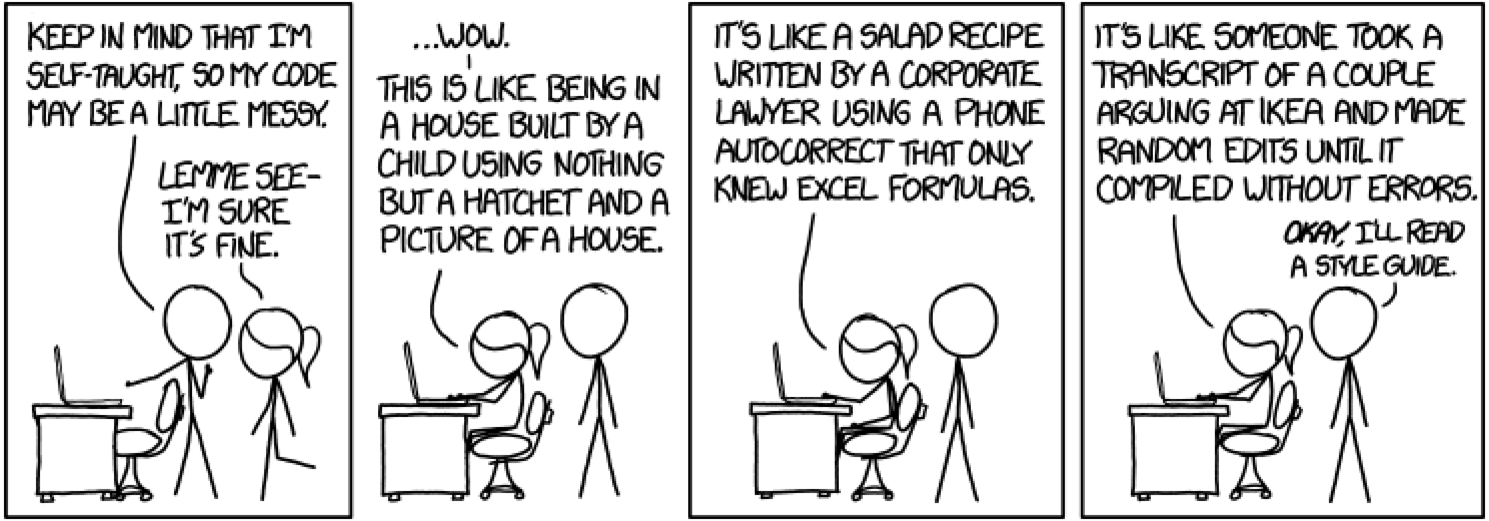
Start testing and increase code coverage to 100% | 10 minutes

- Write an acceptance test to check the “solve” case | XX% CC
- (Optional) Write an `application-death-test` to check “default” case | XX% CC

Case “solve”	**Performs solving of supplied matrix data**
main(‘--solve’)	Assert with reference solution → loadmat('./data/ref_result_system100x100_solve.mat')[‘result']
Case “default”	Exits with a failure code: exit(-1)
main(‘xyz’)	

Code-quality testing and demonstration

Code quality testing


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

[What is code quality, how to measure and improve code quality? (codegrip.tech)]

Code quality testing

- Quality code consists of those features that cater to the need of customers and subsequently provide product satisfaction
- Quality code is free from deficiencies
- Quality code measures how well code can communicate between developers

Motivation for code quality testing

- Poor quality code tends to die early because it might entail substantial technical debt
- Quality code makes your software:
  - More sustainable (minimum changes over time)
  - Robust (can cope with error usage)
  - Promotes easy transferability
  - Increases readability
  - Decreases technical debt

How do we conduct code quality checks?

- Occurrence of software defects and software quality are related
- Code quality gets overlooked in favor of programming speed → Can accumulate to a huge workload

→ Linter is a tool that automatically checks the quality of the code fitting to your conventions

Tools for code-quality checks

Python	`PyLint` `Flake8`	[https://pylint.pycqa.org/en/latest/] [https://flake8.pycqa.org/en/latest/index.html]
C++	`Clang-Tidy`	[https://clang.llvm.org/extra/clang-tidy/]
…	`SonarQube`	[https://www.sonarqube.org/]

Test process automation

Test process automation

Testing procedures are repetitive and time consuming

The testing process can easily be conducted by a script running automatically

→ Test process automation

What is test process automation?

- Automating the testing procedure
- Automating the management and application of test data and results

What Is Automation Testing? (codecademy.com)

Test process automation

Testing procedures are repetitive and time consuming

The testing process can easily be conducted by a script running automatically

→ Test process automation

80% of organizations use automation testing and it is projected to increase in the next years

What is test process automa

- Automating the testing pr
- Automating the management and application of test data and results

Motivation for test automation

- Cost

  Automated testing will lead to testing without manpower

- Speed

  More tests can be concluded in the same amount of time

- Effectiveness

  Usually automated tests find bugs sooner

What are easily automated tests?

Repetitive tests

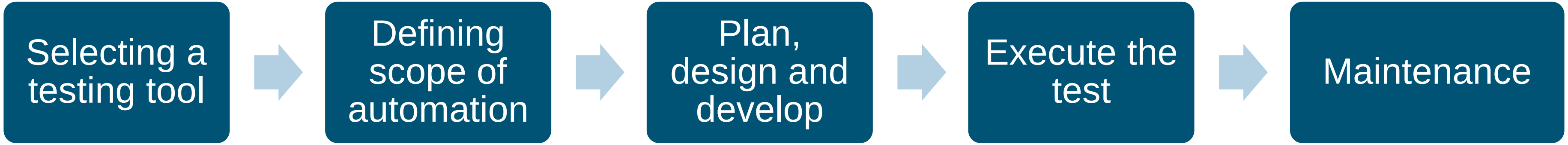
Time-consuming tests

Tests for multiple builds

Tests vulnerable to human error

Frequently used tests

How do we automate testing?



Selecting a testing tool

Defining scope of automation

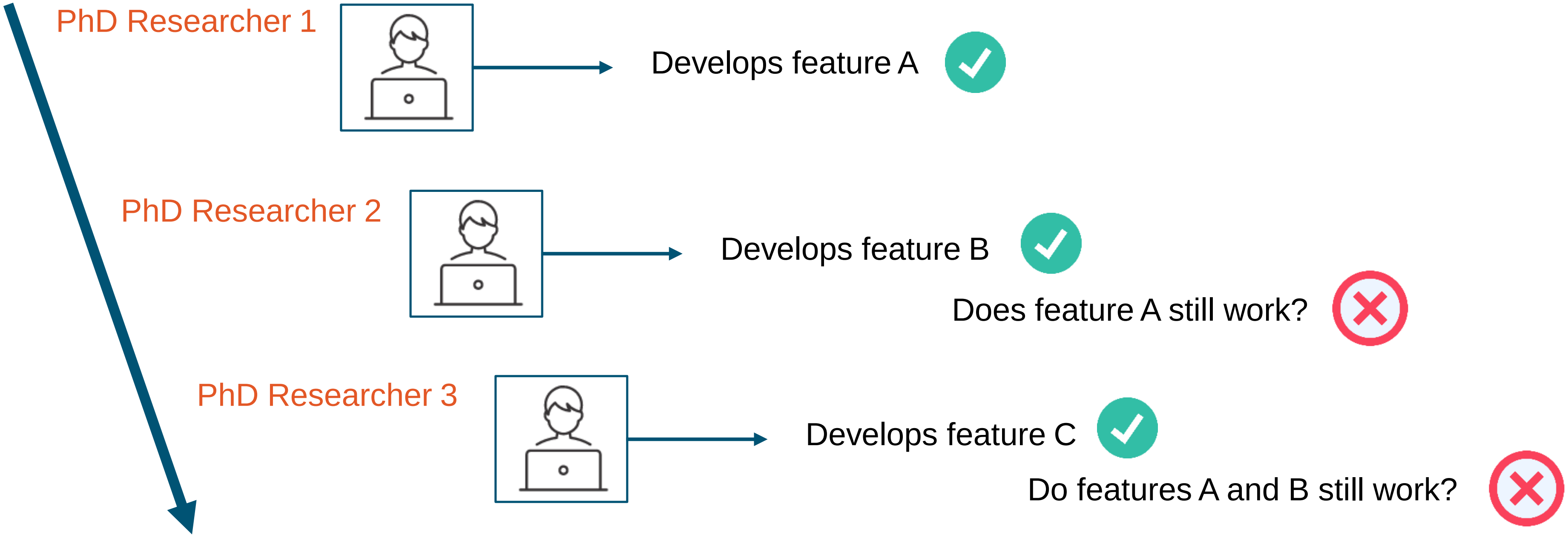
Plan, design and develop

Execute the test

Maintenance

Test process automation with Gitlab CI

Motivation | In Academia



PhD Researcher 1 → Develops feature A ✓

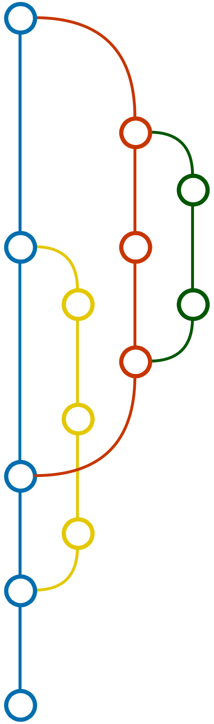
PhD Researcher 2 → Develops feature B ✓

Does feature A still work? ✗

PhD Researcher 3 → Develops feature C ✓

Do features A and B still work? ✗

Motivation | Developing in groups



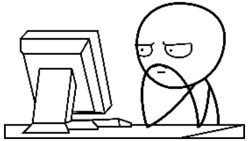
To prevent complex integration:

“Commit code frequently”
[Duval et al. practices]

“Everyone commits to the mainline everyday”
[Fowler practices]

⇩

Merge conflicts, bugs, defects, broken routines



⇩

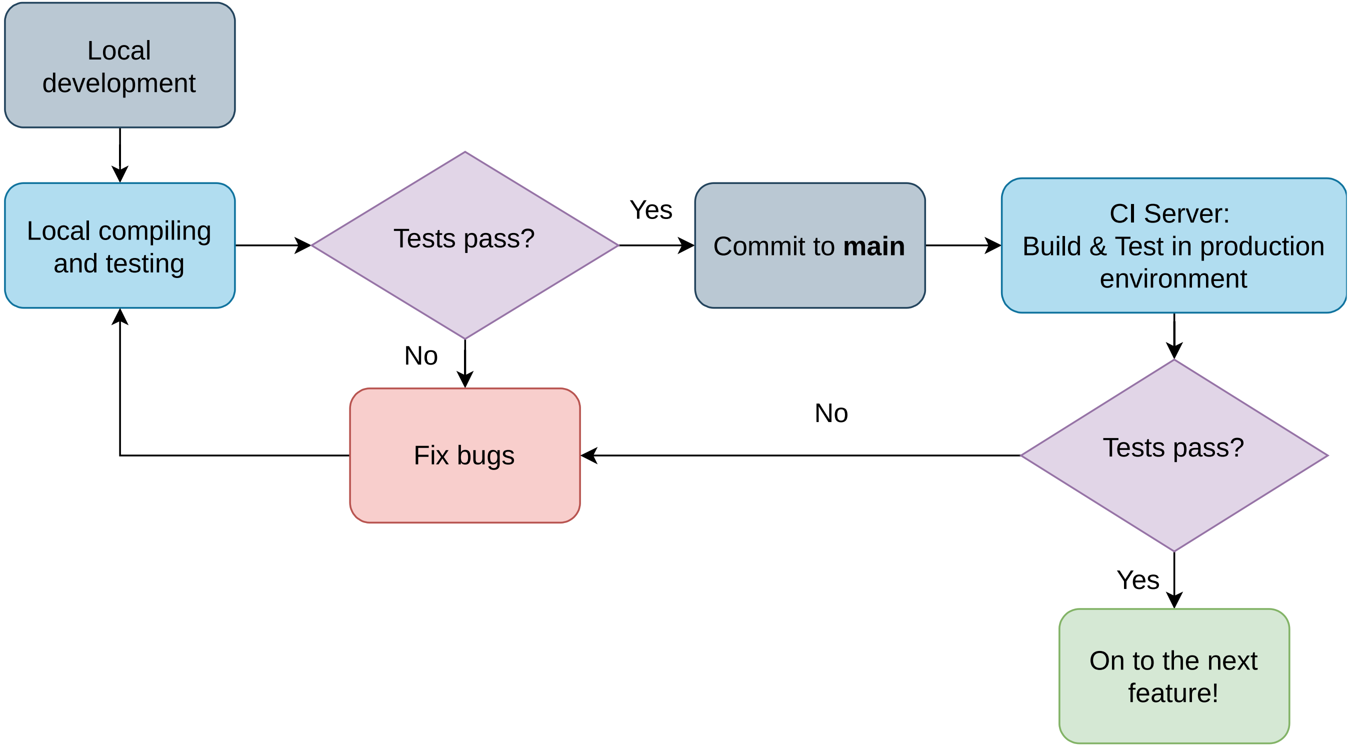
With CI → Better quality control over new features and their effect on existing implementation – through automated build and test routines ✓

What is continuous integration?

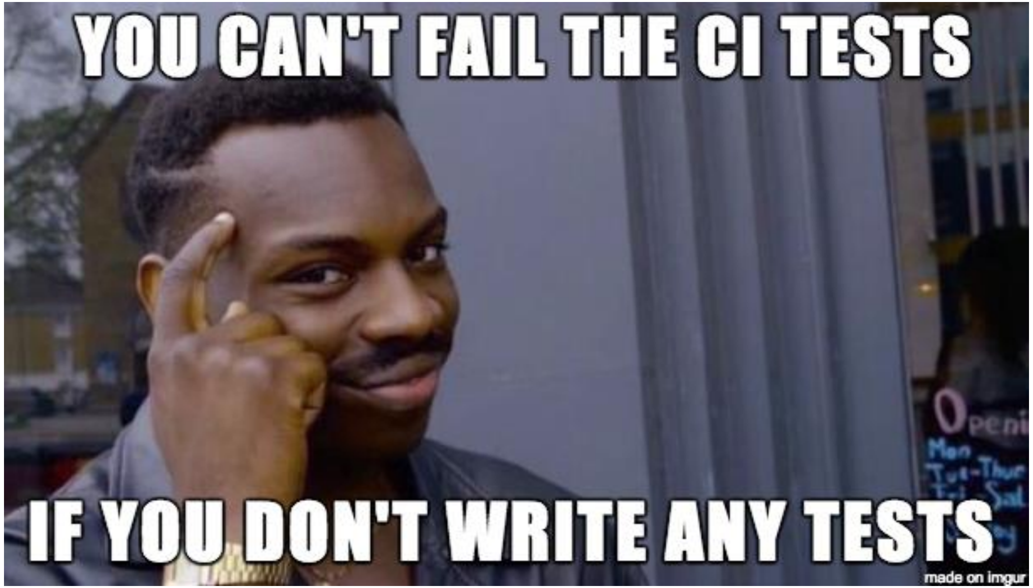
“Practice of automating the integration of code changes from multiple contributors into a single software project.”

[altassian.com]

Workflow | Continuous integration

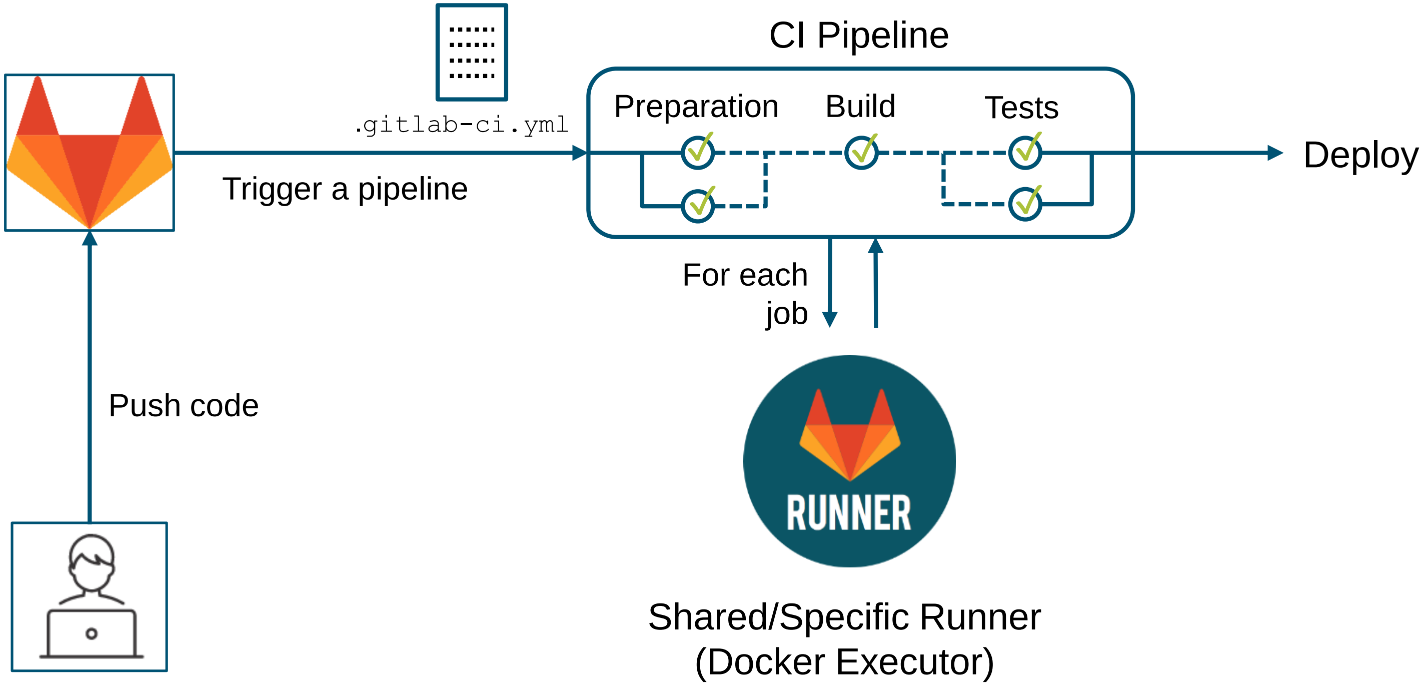


Write tests!



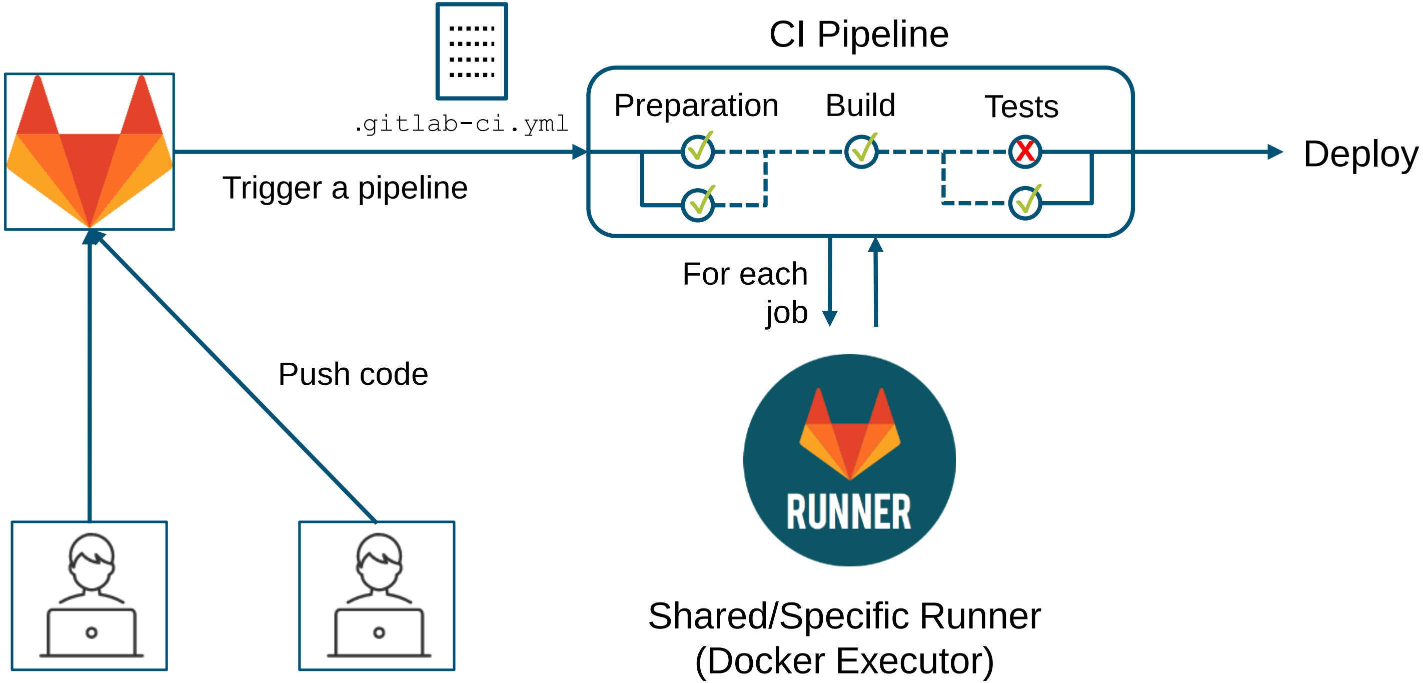
[https://itnext.io/]

Continuous integration with GitLab



Shared/Specific Runner
(Docker Executor)

Continuous integration with GitLab



Demonstration of elPaSo testing framework

elPaSo | About

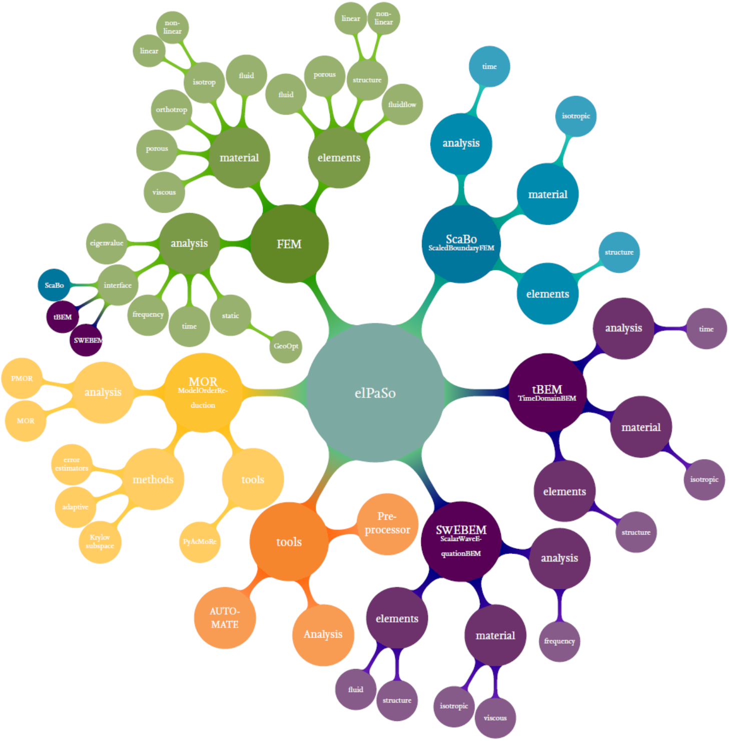
Elementary **Pa**rallel **So**lver (elPaSo)

- Performs vibroacoustic analysis in the modal, static, time and frequency domain
- Based on FEM, BEM, SBFEM
- Efficient computing strategies - parallel computing, model order reduction

INSTITUT
FÜR AKUSTIK
TECH elPaSo

https://akustik.gitlab-pages.rz.tu-bs.de/elPaSo-Core/

https://git.rz.tu-bs.de/akustik/elPaSo-Core/

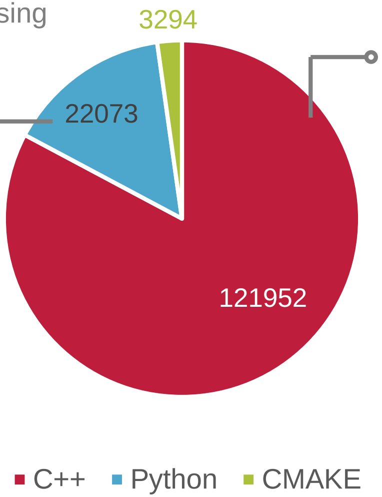


Source: InA/TU Braunschweig

elPaSo | Source code

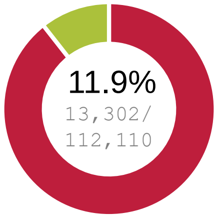
Programming language and SLOC:

- GUI
- Pre- and post-processing routines
- Visualizer

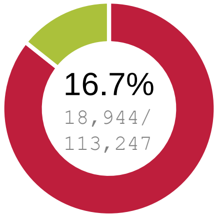


- Numerical methods
- Material definitions
- Coupling interfaces
- Solver routines

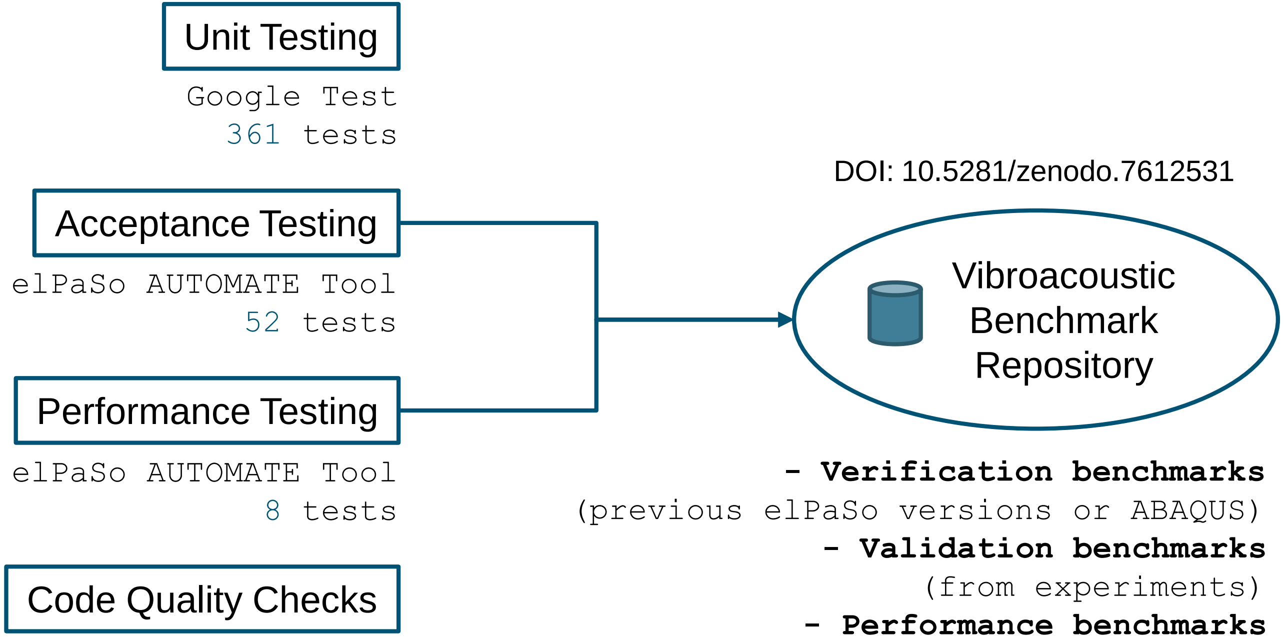
elPaSo | Testing Framework



Unit test coverage



Acceptance test coverage



Unit Testing

Google Test
361 tests

Acceptance Testing

elPaSo AUTOMATE Tool
52 tests

Performance Testing

elPaSo AUTOMATE Tool
8 tests

Code Quality Checks

Clang-Tidy

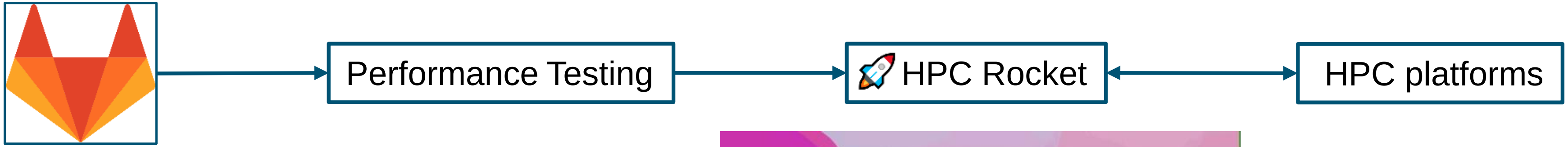
DOI: 10.5281/zenodo.7612531

Vibroacoustic Benchmark Repository

- **Verification benchmarks**
  (previous elPaSo versions or ABAQUS)
- **Validation benchmarks**
  (from experiments)
- **Performance benchmarks**
  (Scalability with MPI and OMP threads)

elPaSo | Performance Testing on HPC platforms

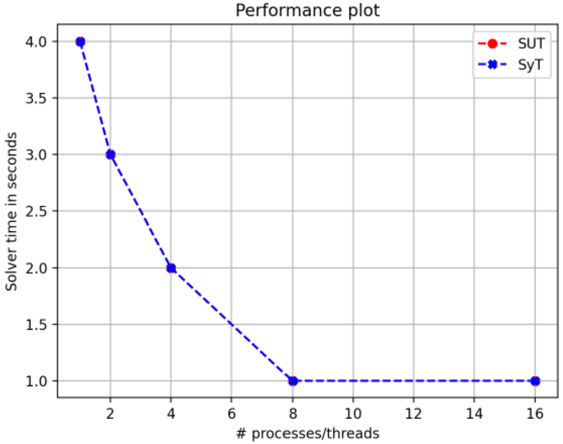
Performance testing with hybrid MPI+OMP parallelization



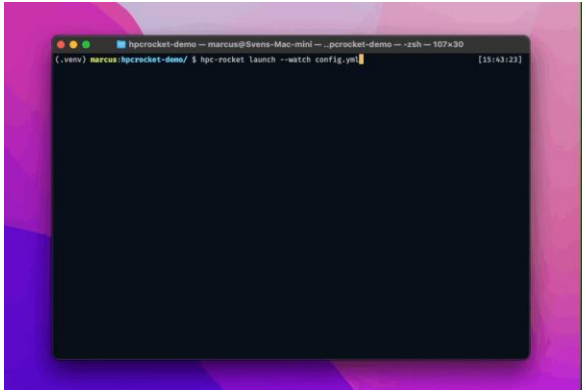
Performance Testing

🚀 HPC Rocket

HPC platforms



CPARDISO solver timings



hpcrocket-demo — marcus@Svens-Mac-mini — ..pcrocket-demo — -zsh — 107×30

```
(.venv) marcus:hpcrocket-demo/ $ hpc-rocket launch --watch config.yml                    [15:43:23]
```

[https://github.com/SvenMarcus/hpc-rocket]

Work in progress for large-scale problems

elPaSo | How tests are incorporated?

Unit testing

- New codes → Test driven development
- Legacy codes → Refactoring and make it testable

→ **Demonstration**

Acceptance testing

- New feature/ new research publication → New benchmark

Features of the elPaSo AUTOMATE Tool

- Python tool running elPaSo benchmarks and compare with set reference
- Execute tests in a HPC cluster with `HPC-Rocket` for computationally expensive tests
- Issue reporting – `python-gitlab` for automated issue creation in GITLAB issue board
- Detailed technical report (currently generated as PDF, in future also as Gitlab pages)

Tips for software testing

Test recommendation

Application Class*	Recommendations
0	Automated tests are recommmended but not required
>=1	The software should have unit tests that verify the most important features
>=2	The software should have an extensive test suite including unit, integration and acceptance tests
3	The previous recommendations are mandatory for applications of this class

*

0	Small scripts only intended for personal use
1	Software intended to be used and extended by others
2	Software with long-term development and maintanability requirements
3	Mission-critical software

[https://suresoft.dev/knowledge-hub/research-software-guidelines/guidelines/]

Tips for software testing

- Choose the best suitable type of testing for your code → Start with unit-testing
- Always write tests first before writing production code → Forces the system to be testable → TDD Workshop
- Designing test codes for legacy codes → Break dependencies and refactor codes to make them testable
- Design clear and simple test cases
- Test name should be self-explaining and sufficiently elaborate
- Defining a set of domain specific benchmarks
- Benchmarks are often computationally expensive → Connect your tests to run on a high-performance computing cluster (HPC-Rocket, Jacamar CI)
- Tests are done in specific environments → Containerization

https://suresoft.dev/knowledge-hub/continuous-integration/containers/

Thank you for your attention