ugtest

Tobias Trautmann

GCSC

May 18, 2020

Outline

Introduction to testing

Goals of testing

- increase trust in its results

Goals of testing

- increase trust in its results
- make code maintainable

Goals of testing

- increase trust in its results
- make code maintainable
- make code refactorable

Goals of testing

- increase trust in its results
- make code maintainable
- make code refactorable
- make it sufficiently robust

Goals of testing

- increase trust in its results
- make code maintainable
- make code refactorable
- make it sufficiently robust
- check if it performs its functions within an acceptable time

Goals of testing

- increase trust in its results
- make code maintainable
- make code refactorable
- make it sufficiently robust
- check if it performs its functions within an acceptable time
- check wether in runs its intended environments

$\Rightarrow$ Testing software is a **necessity**

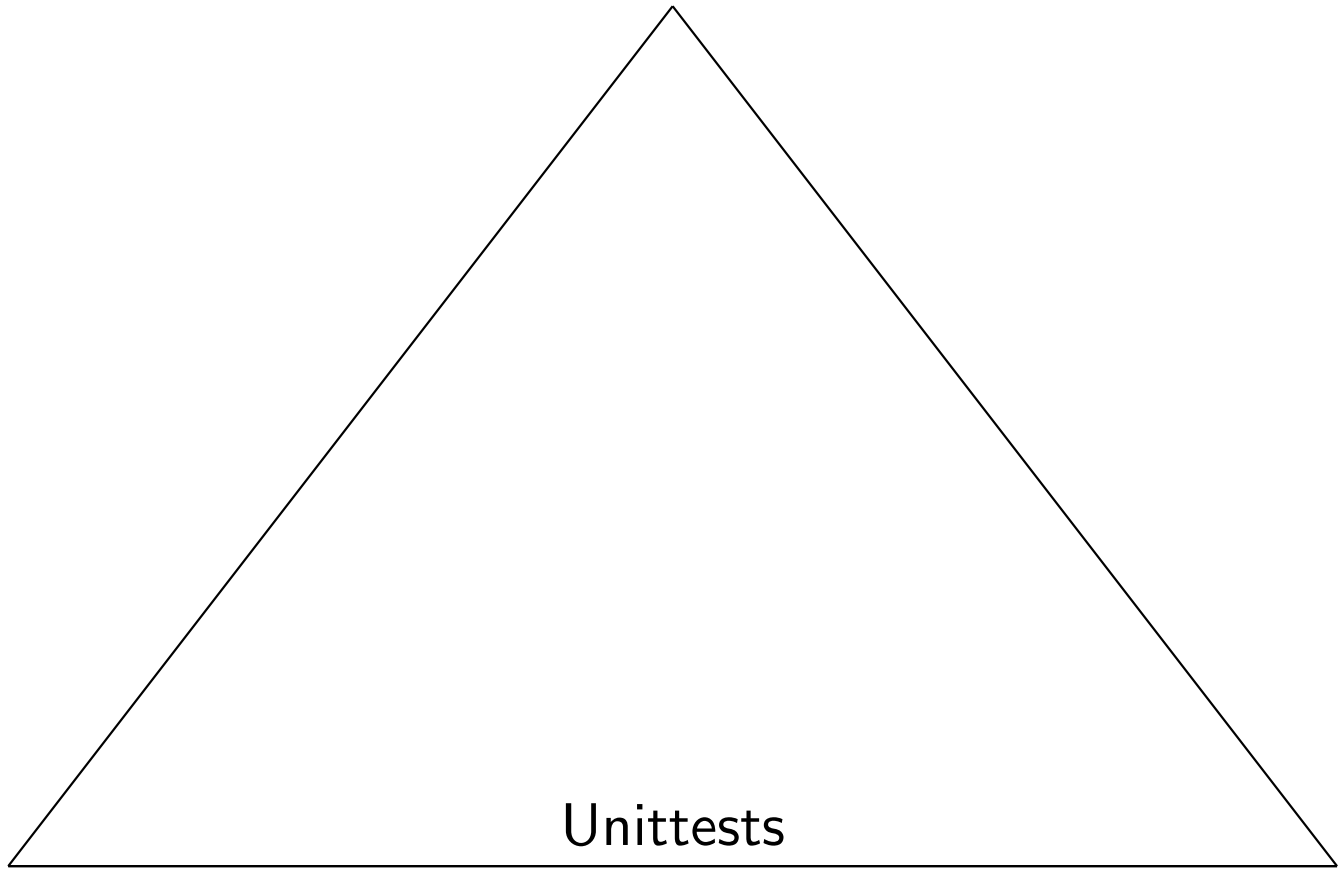
Defects

Where do defects come from?

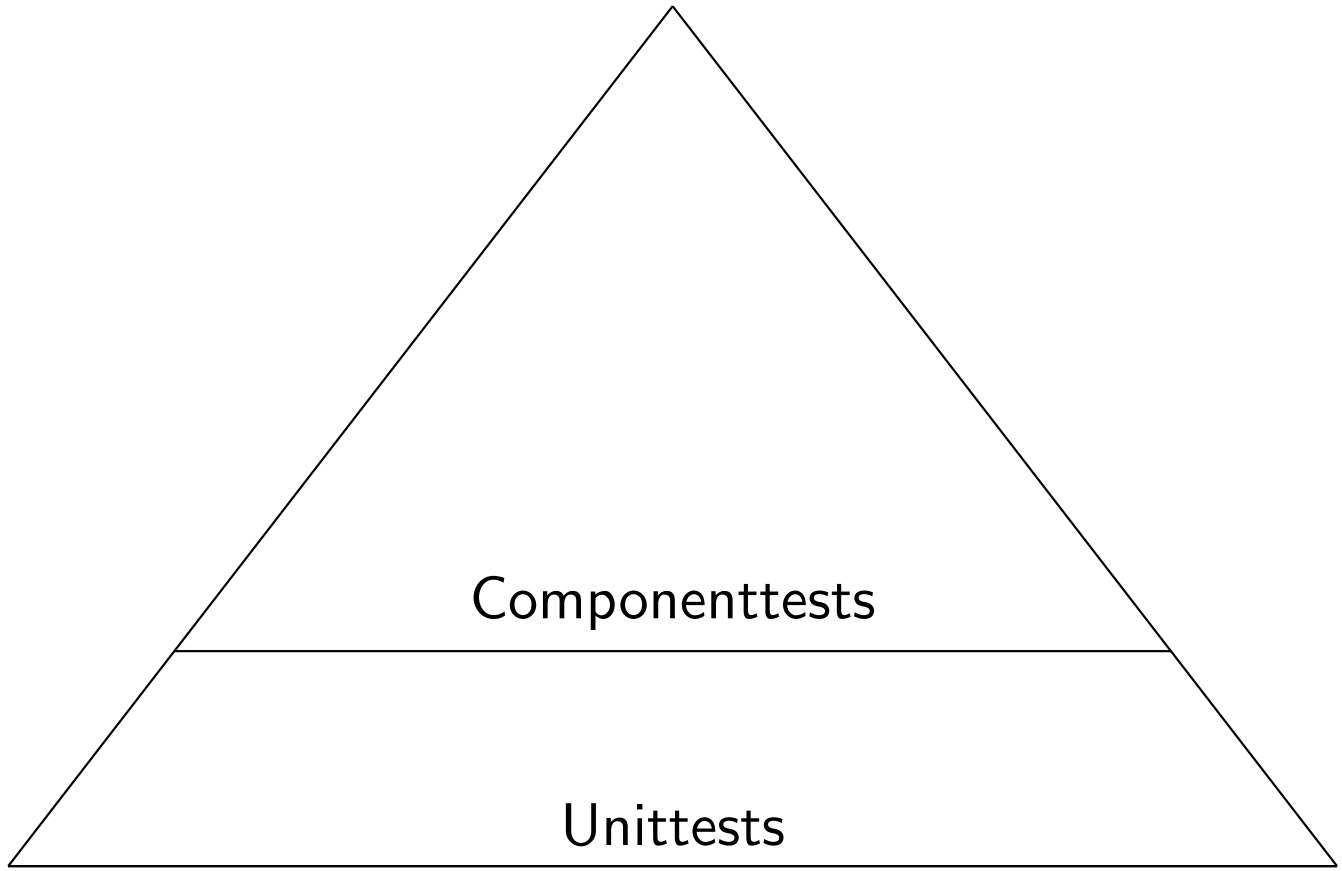
- script error (lua)
- bug in code
- integration
- error in design

Are you responsible for it?
mitigation runtime error logging
$\Rightarrow$ Makes clear what to test with which priority

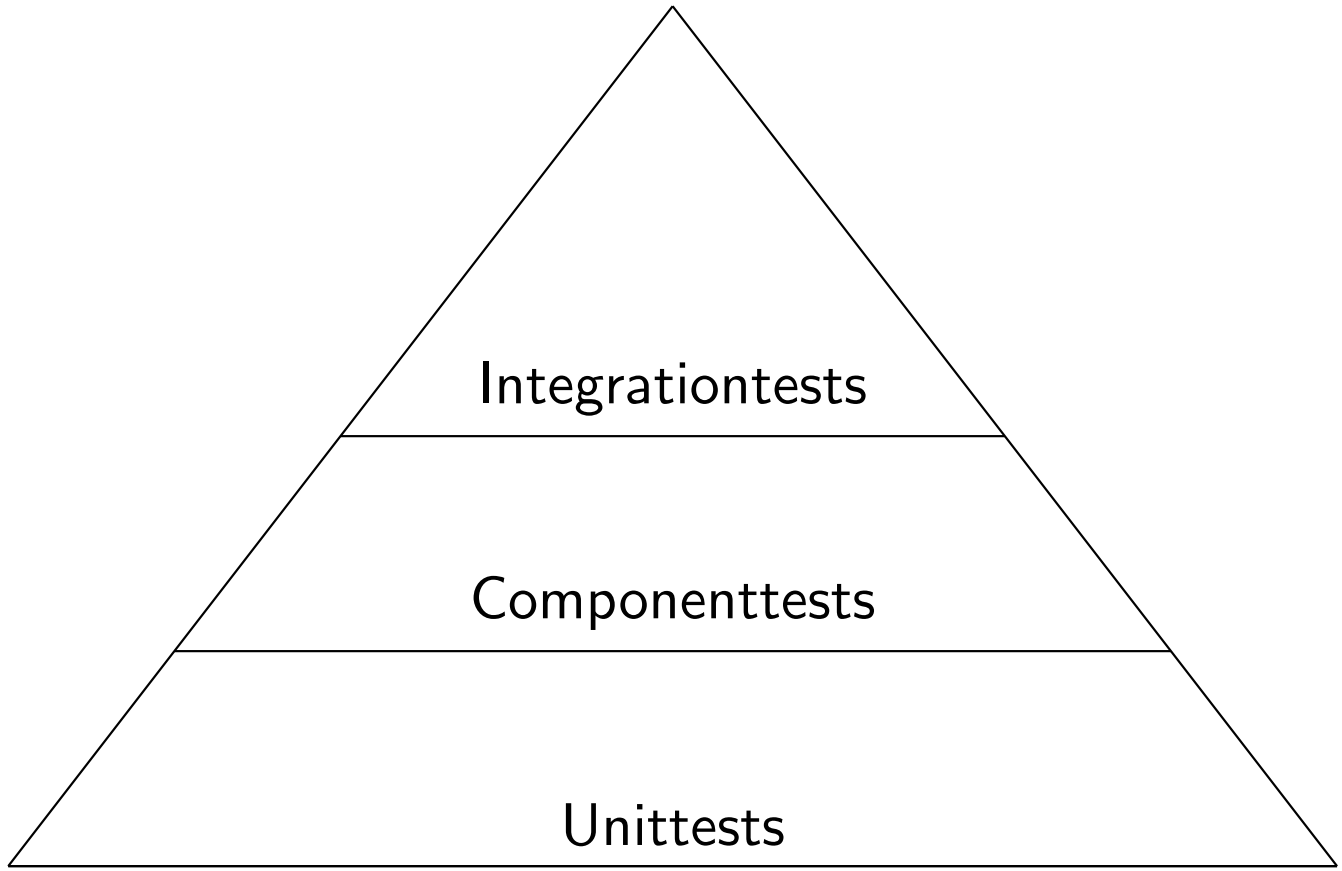
Efficency



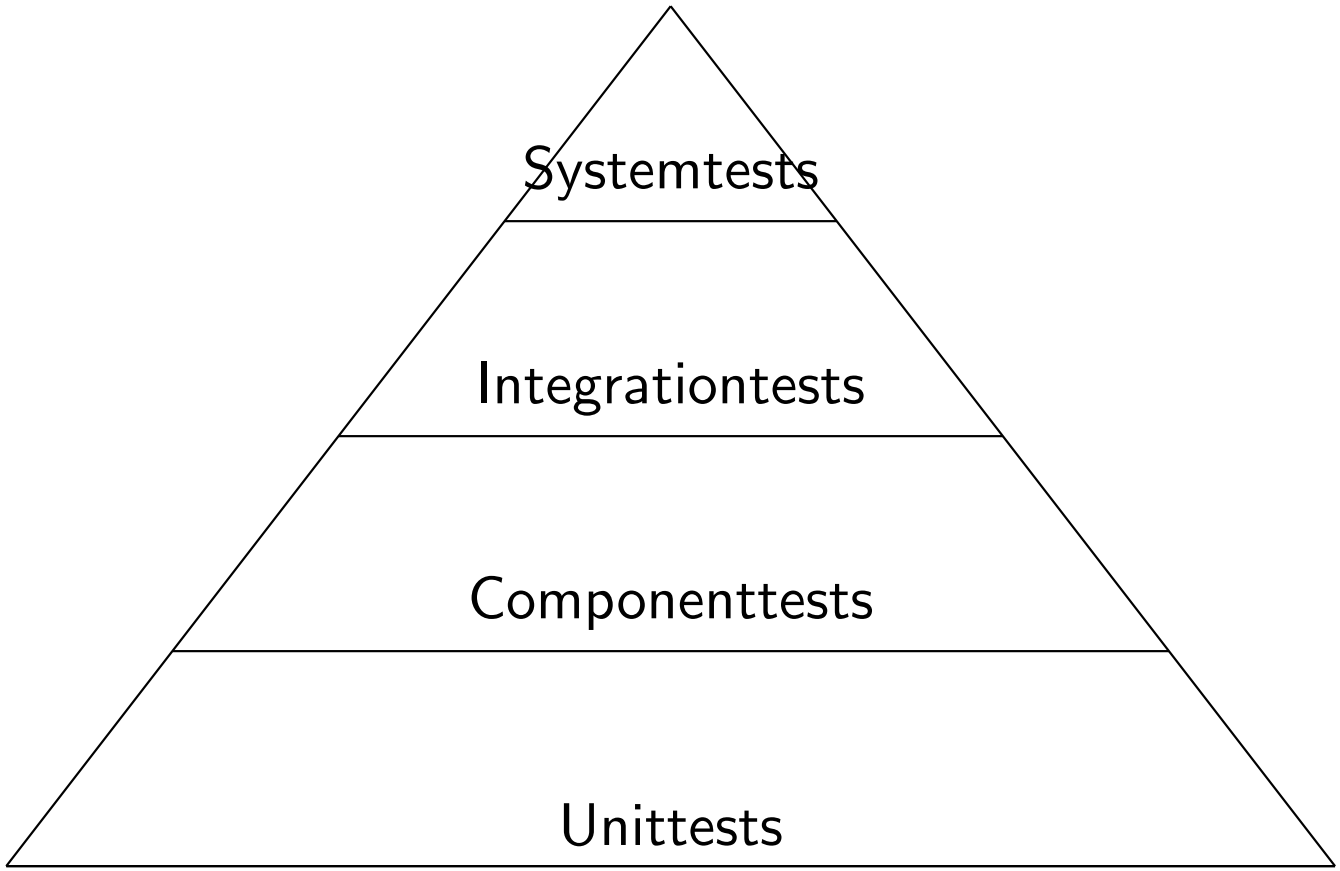
Efficency



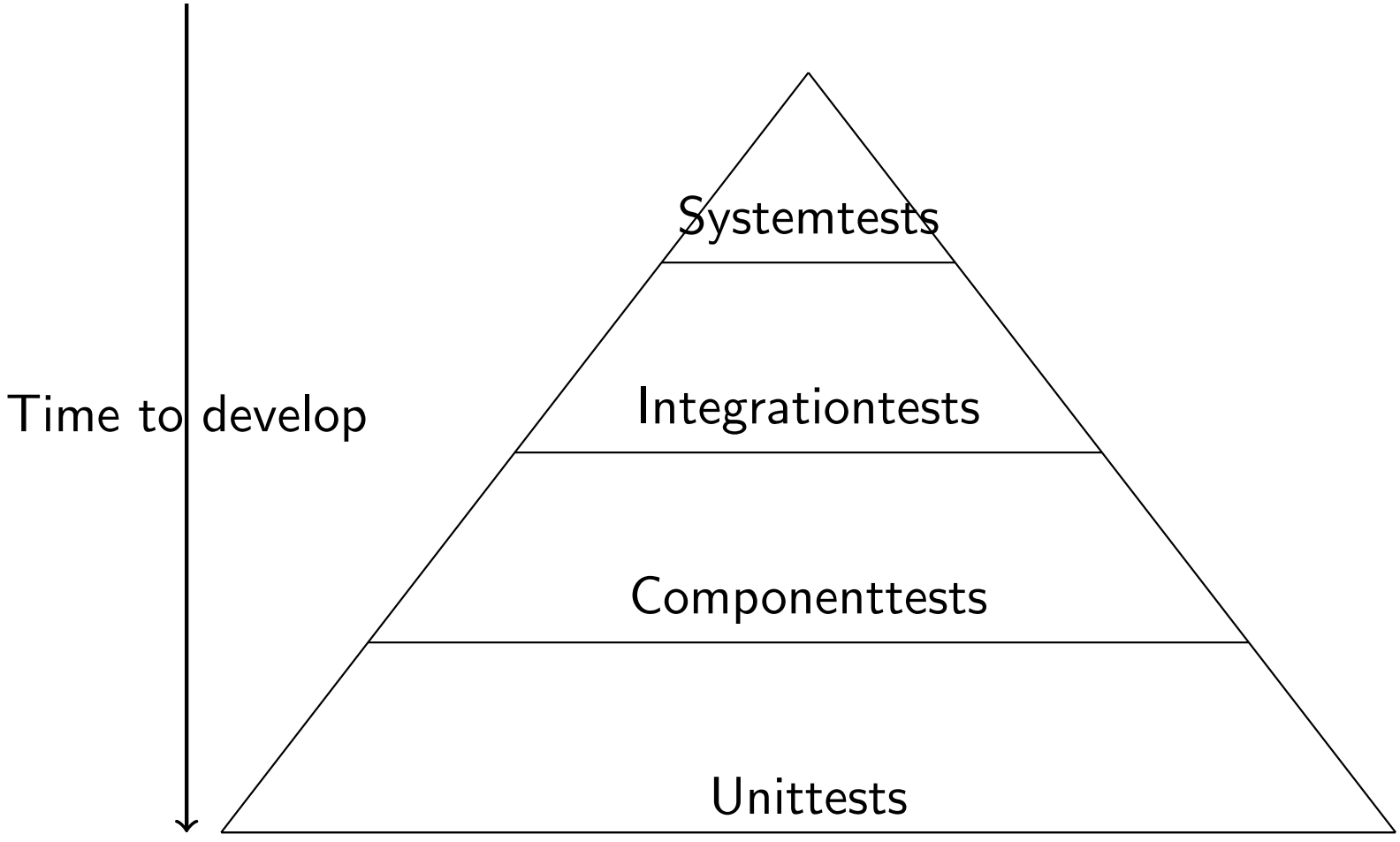
Efficency



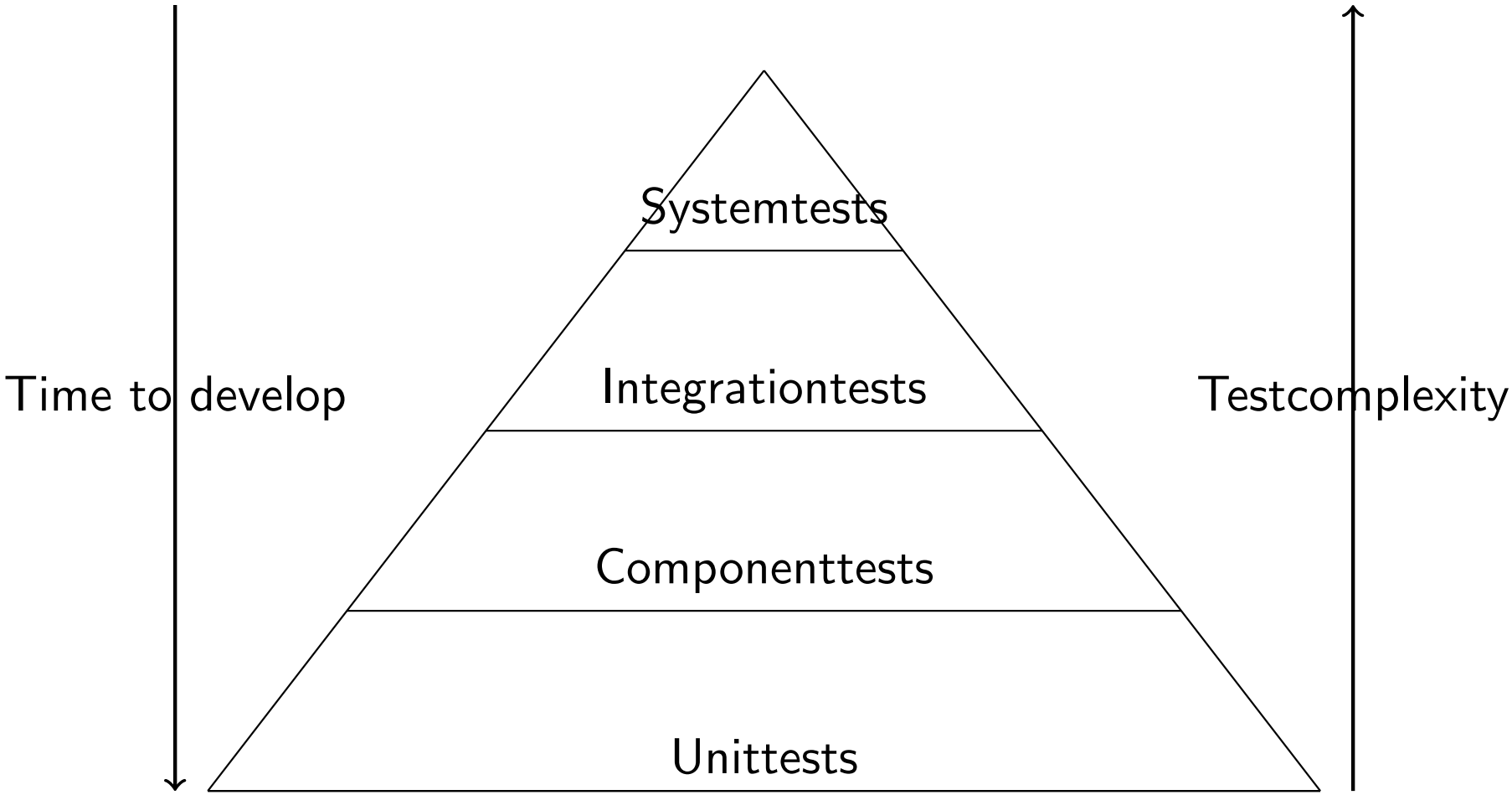
Efficency


Systemtests
Integrationtests
Componenttests
Unittests

Efficency

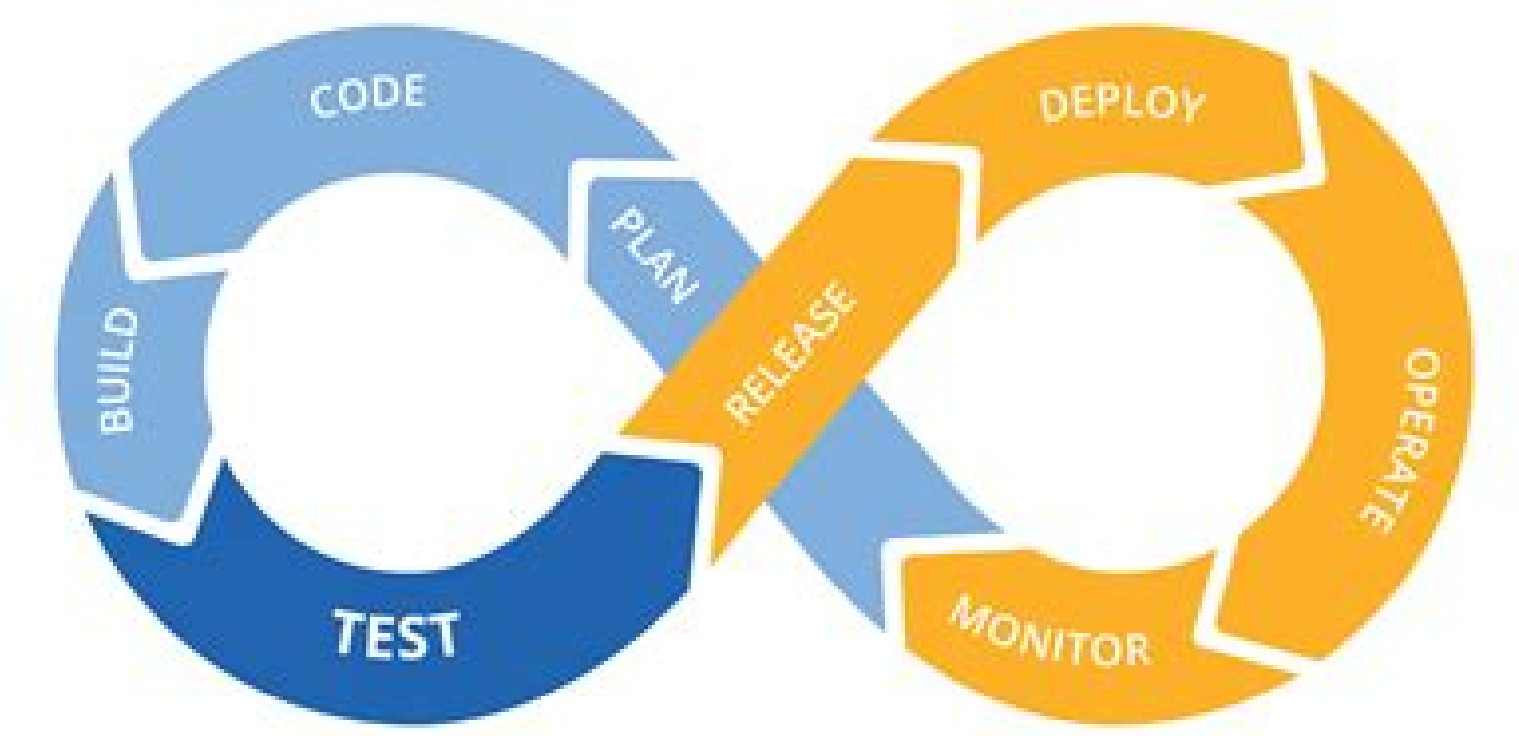

Time to develop
Systemtests
Integrationtests
Componenttests
Unittests

Efficency

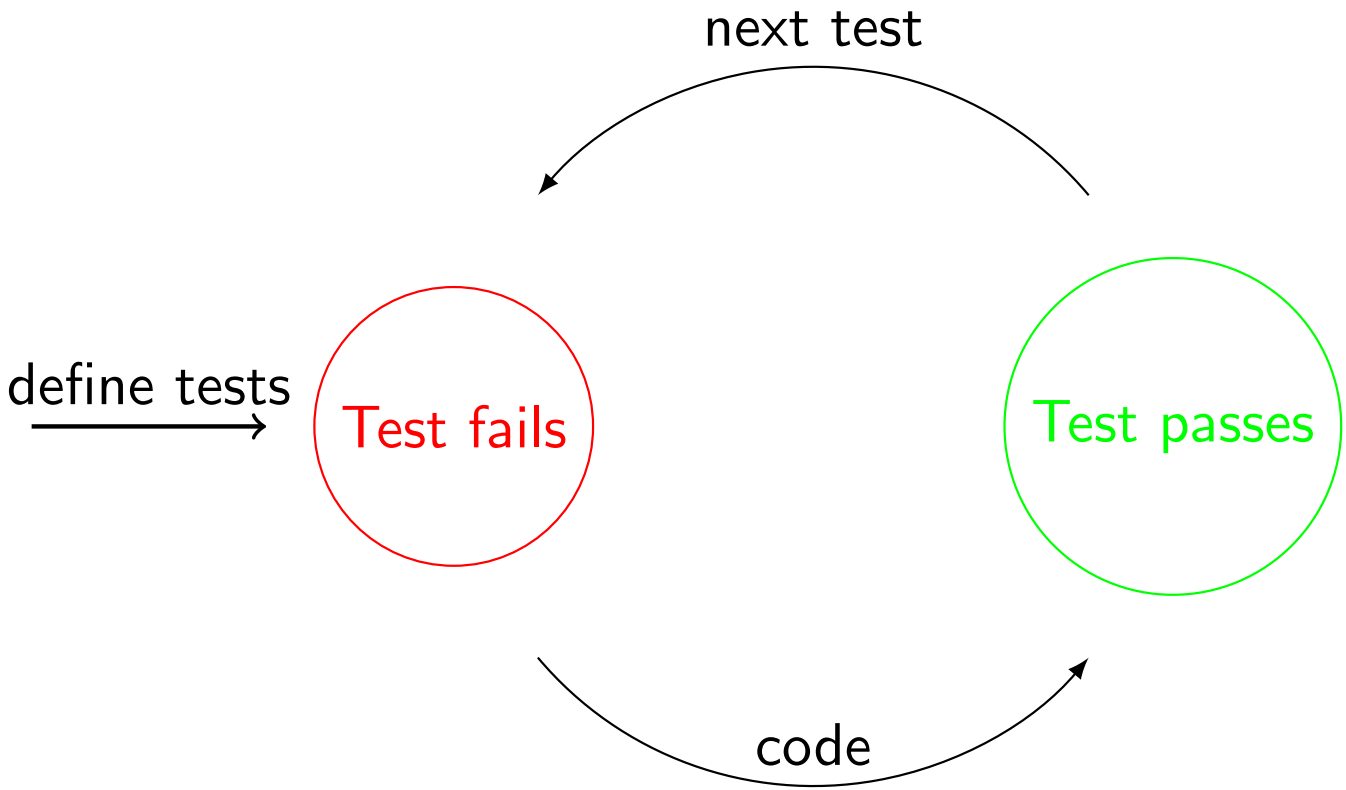

Time to develop
Systemtests
Integrationtests
Componenttests
Unittests
Testcomplexity

Approaches

Continous Integration / Continous Delivery



Test driven development



Boost.Test

Structure

```
#include <boost/test/included/unit_test.hpp>

//uncomment if using templates
//#include <boost/test/test_case_template.hpp>

//uncomment if testing in parallel
//#include <boost/mpl/list.hpp>

//stuff

BOOST_AUTO_TEST_SUITE(<testsuite_name>)

    BOOST_AUTO_TEST_CASE(<testcase_name>)
    {
        //Testcase here
    }
    BOOST_AUTO_TEST_CASE(<testcase_name>)
    {
        //Testcase here too
    }

BOOST_AUTO_TEST_SUITE_END()
```

Assertion Levels

assertion level	error counter	test continuation
warn		yes
check	++	yes
require	++	no

Float point comparison

Exception handling

Fixtures

```
struct UGbase
{
    //Call UGInit before testcase starts
    UGbase()
    {
        ug::UGInit(&framework::master_test_suite().argc,
            &framework::master_test_suite().argv);
    }
    //call UGFinalize after test case ends
    ~UGbase()  {
        ug::UGFinalize();
    }
};
BOOST_AUTO_TEST_SUITE(fixtureshowsuite)
BOOST_AUTO_TEST_CASE(fixtureshowcase, UGbase){
    //your test needing a clean ug
}
BOOST_AUTO_TEST_SUITE_END()
```

Templates

```
typedef boost::mpl::list<int, long, unsigned char>
    test_types;

BOOST_AUTO_TEST_SUITE(templateshowsuite)
    BOOST_AUTO_TEST_CASE_TEMPLATE(templateshowcase, T,
        test_types)
    {
        //your super fancy template test
    }
BOOST_AUTO_TEST_SUITE_END()
```

Testing

Test execution

- add buildflags "–fprofile-arcs –ftest-coverage –fPIC" as well as no optimization for code coverage analysis
- build ug with UGTest and your plugin activated
- your plugin contains tests in a top level folder named "tests"
- executable named "ugtest_unit" and "ugtest_system" lands in ug4/bin
- list of params
- example:

  ug4/bin $ ./ ugtest_unit −−log−level=ALL −−log−format=HRF
- Show result

Automatization with Jenkins

- Cobertura
- two builds one serial, one parallel -¿ two test runs
- Code coverage: gcovr can produce xml for cobertura
- needs log_format=XML

Automatization with Docker

- Container stuff
- Dockerfile

Additional

Open issues (?)

- git branching & releases
- naming conventions for tests
- definiton of done
- tooling
- error handling
- design for testability
- test data
- mocking

Additional resources

- Boost.Test 1.58 documentation
- ugtests github
- Antipatterns
- Docker Documentation
- newest Boost.Test
- dreckstools

References

- wiki
- Basiswissen Softwaretest