ugtest

Tobias Trautmann

GCSC

May 25, 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

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**

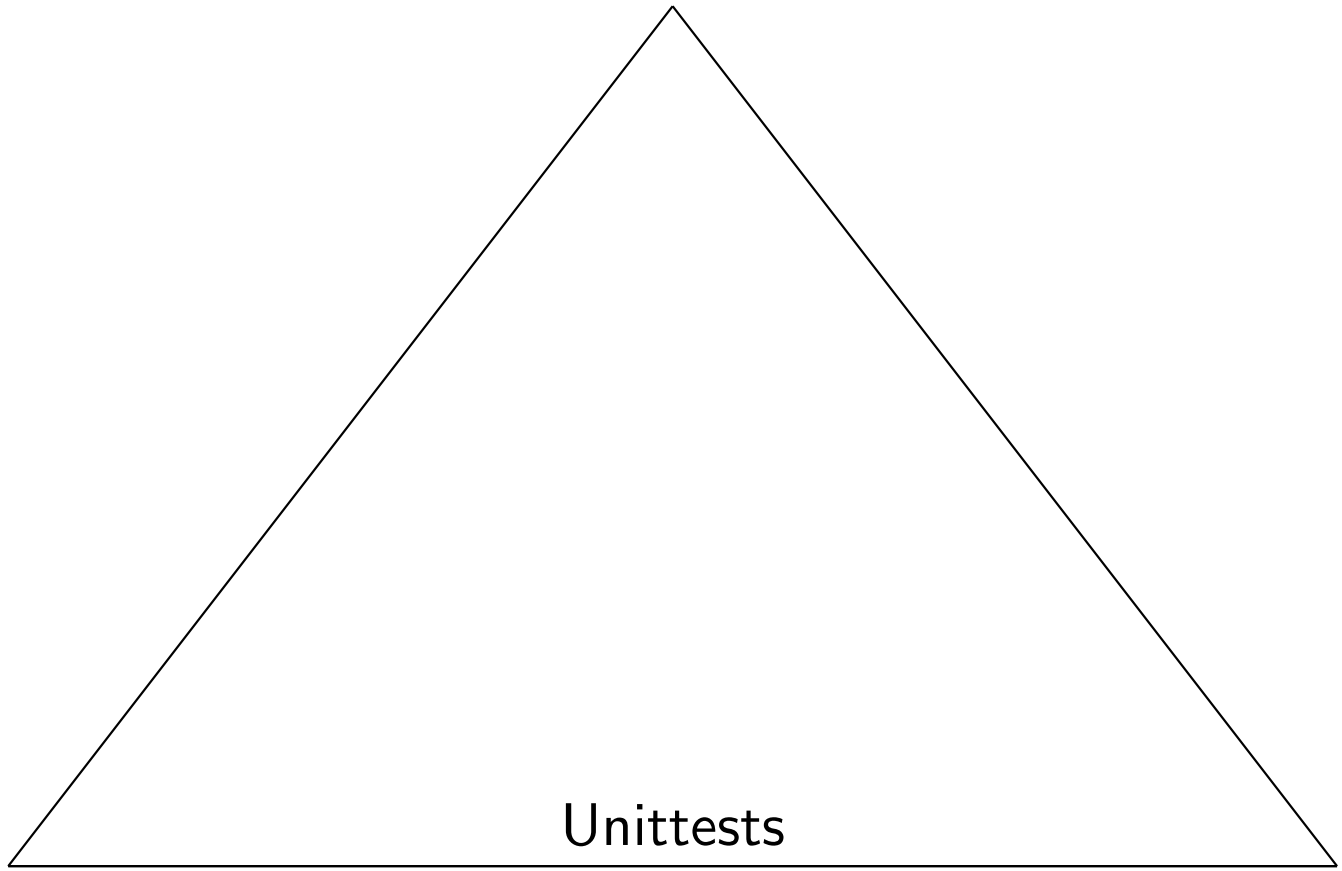
Defects

Where do defects come from?

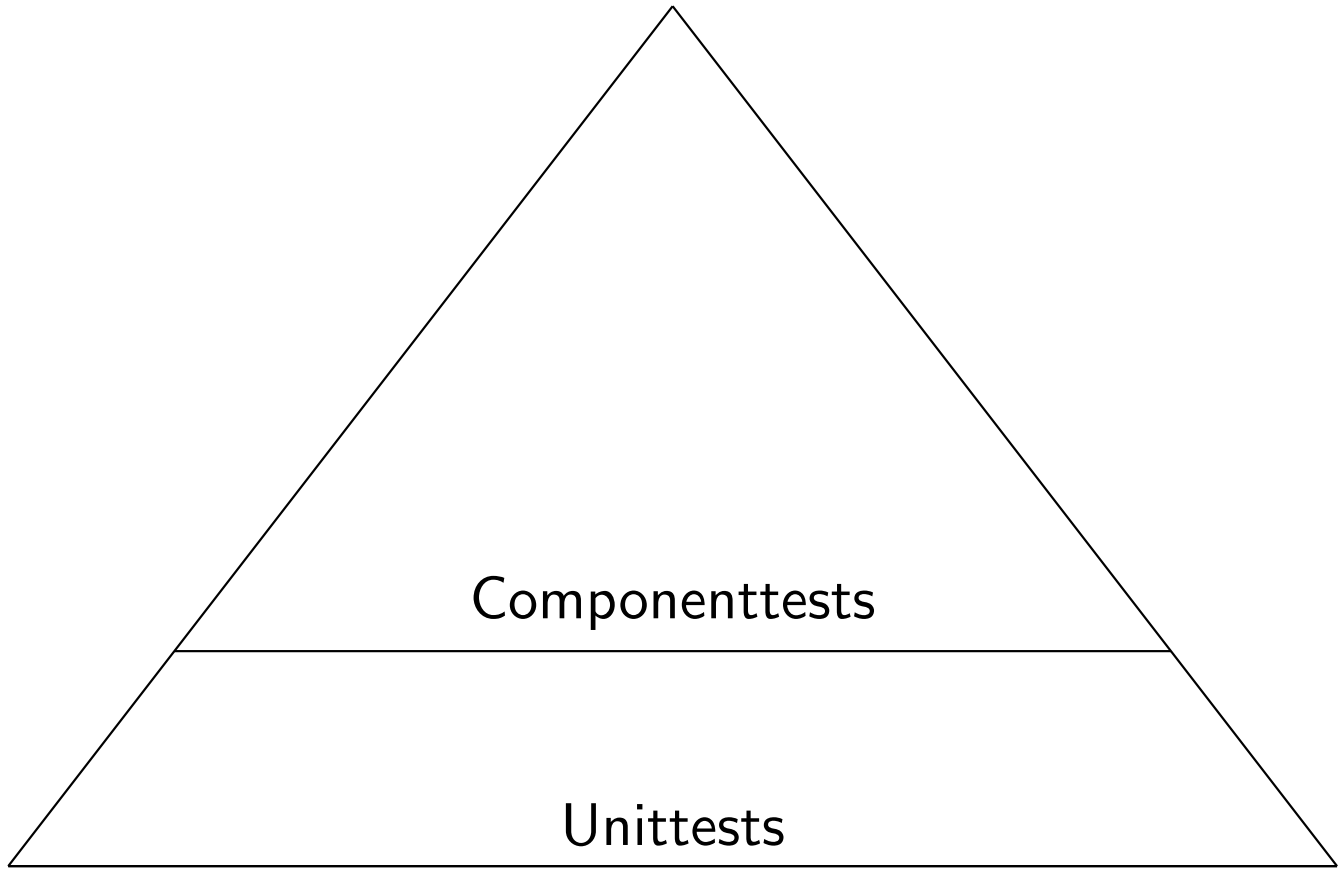
- error in design
- bug in code
- script error (lua)
- integration

$\Rightarrow$ Makes clear what to test with which priority

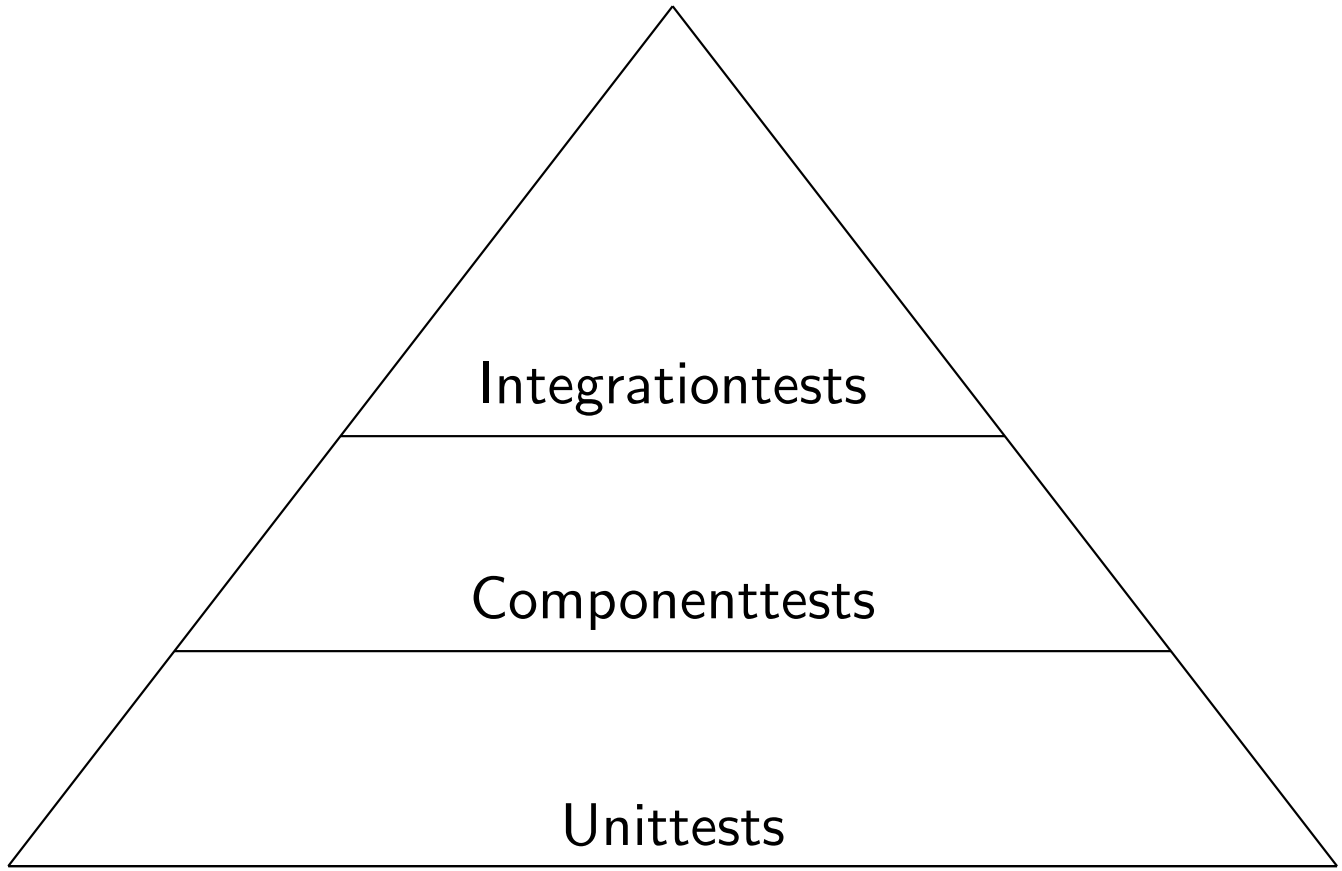
Test Pyramid



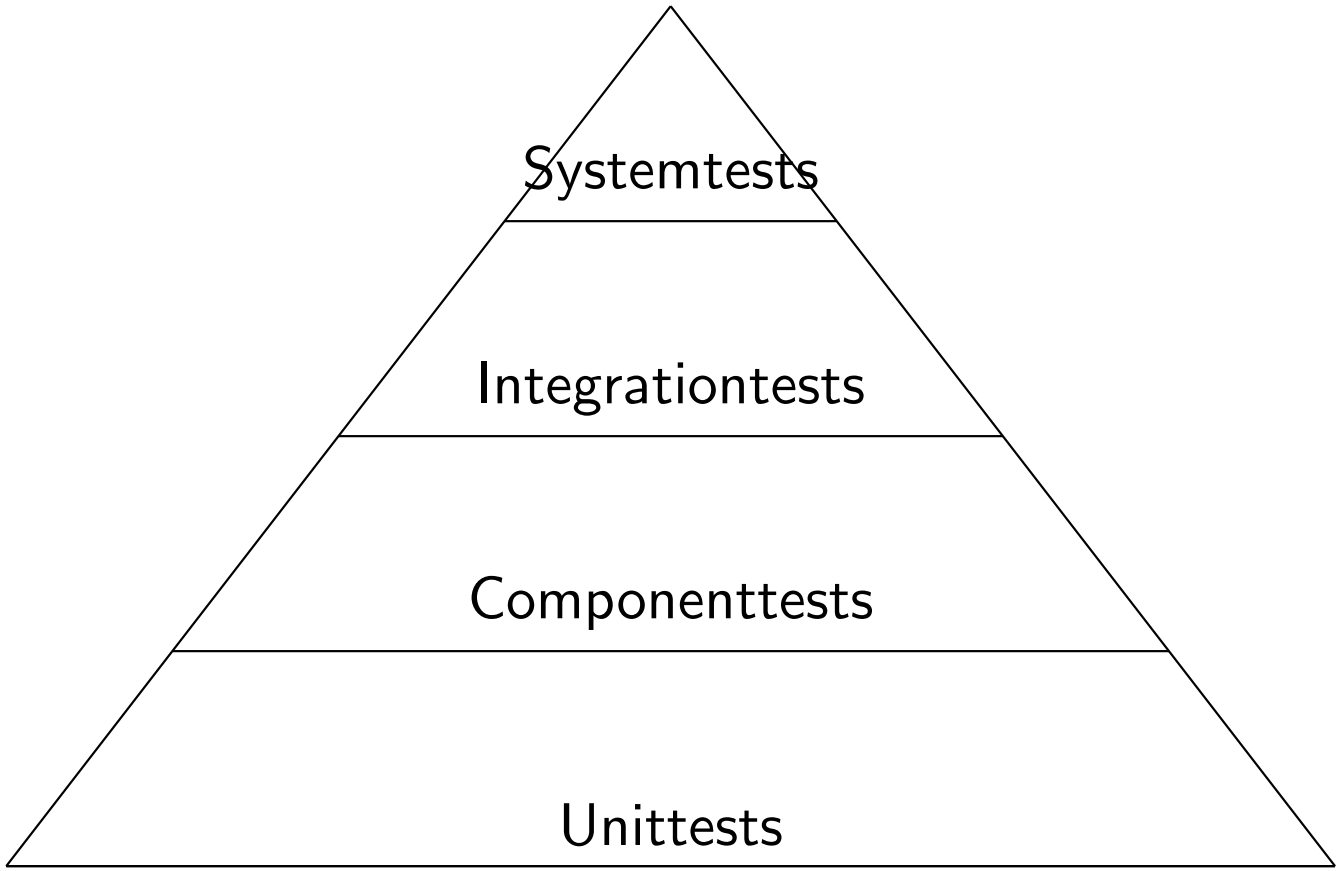
Test Pyramid



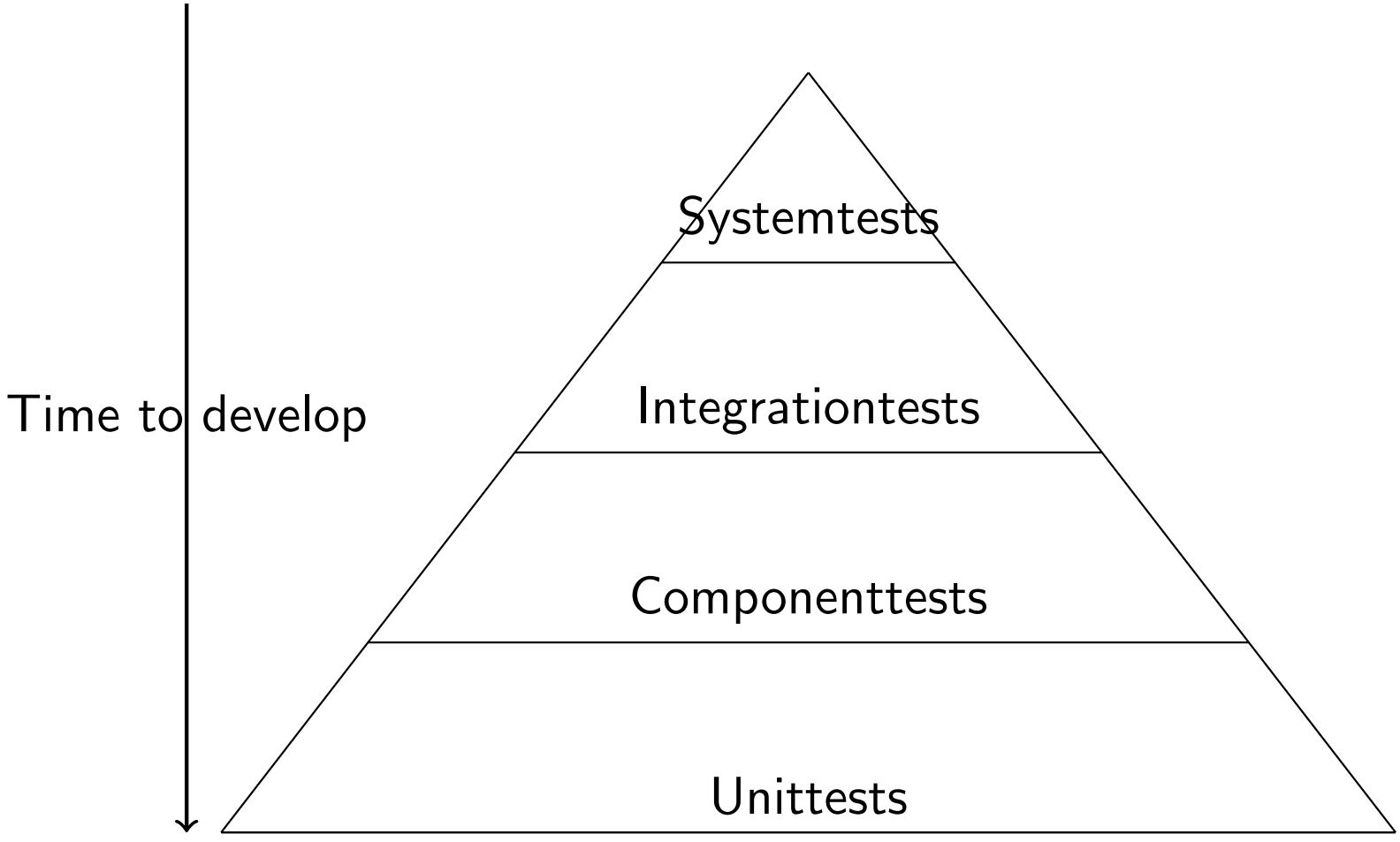
Test Pyramid



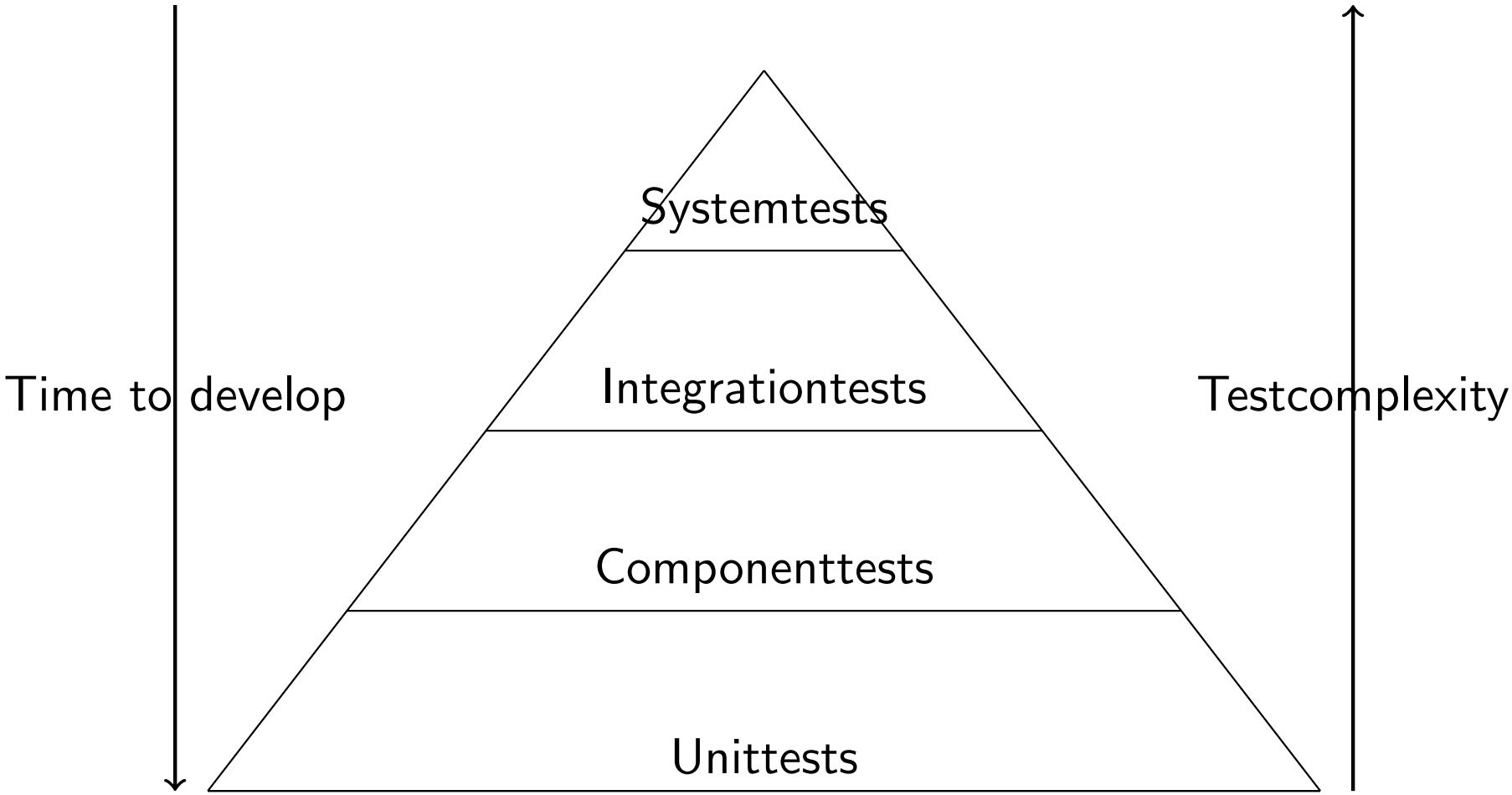
Test Pyramid



Test Pyramid



Test Pyramid



Boost.Test

Structure

```
#include "UGTest.h"

//stuff

BOOST_AUTO_TEST_SUITE(<filename>)

    BOOST_AUTO_TEST_CASE(<filename_function>)
    {
        //Testcase here
    }
    BOOST_AUTO_TEST_CASE(<filename_function_additional_info>)
    {
        //Testcase here too
    }

BOOST_AUTO_TEST_SUITE_END()
```

Assertion Levels

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

Basics

- BOOST_<level>(predicate)
- BOOST_<level>_<GE,LE,GT,LT,NE>(left, right)
- BOOST_<level>_EQUAL(left, right)
- BOOST_IS_DEFINED(SYMBOL)
- BOOST_<level>_MESSAGE(msg)

Nice to know

```
// left term of == is expanded in the logs
// check a % b == c has failed [13 % 2 != 12]
BOOST_CHECK(a % b == c);
// right term of == is not expanded in the logs
// check a == c % b has failed [13 != 0]
BOOST_CHECK(a == c % b);
```

Floating point comparison

```
float n1=1/3, n2=1/3;
float t=0.00001;
//relative
BOOST_REQUIRE_CLOSE(n1,n2,t);
//absolute
BOOST_REQUIRE_CLOSE_FRACTION(n1,n2,t);
```

Exception handling

```
void some_function(int n){
    if(n == -1){
        throw Exception;
    }
}
BOOST_AUTO_TEST_SUITE(exceptionshowsuite)
    BOOST_AUTO_TEST_CASE(tst_some_function_exceptions){
        BOOST_CHECK_NO_THROW( some_function(0) );
        BOOST_CHECK_THROW( some_function(-1),
            Exception );
    }

BOOST_AUTO_TEST_SUITE_END()
```

Fixtures

```
BOOST_AUTO_TEST_SUITE(fixtureshowsuite)
    BOOST_AUTO_TEST_CASE(fixtureshowcase, somestruct){
        //Fixture constructor called
        //Your test stuff
        //Fixture deconstructor called
    }
BOOST_AUTO_TEST_SUITE_END()

BOOST_FIXTURE_TEST_SUITE(fixtresuite, somestruct)
    BOOST_AUTO_TEST_CASE(descriptivename){
        //Fixture constructor & deconstructor called
    }

    BOOST_AUTO_TEST_CASE(descriptivename2){
        //Fixture constructor & deconstructor called
    }
BOOST_FIXTURE_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

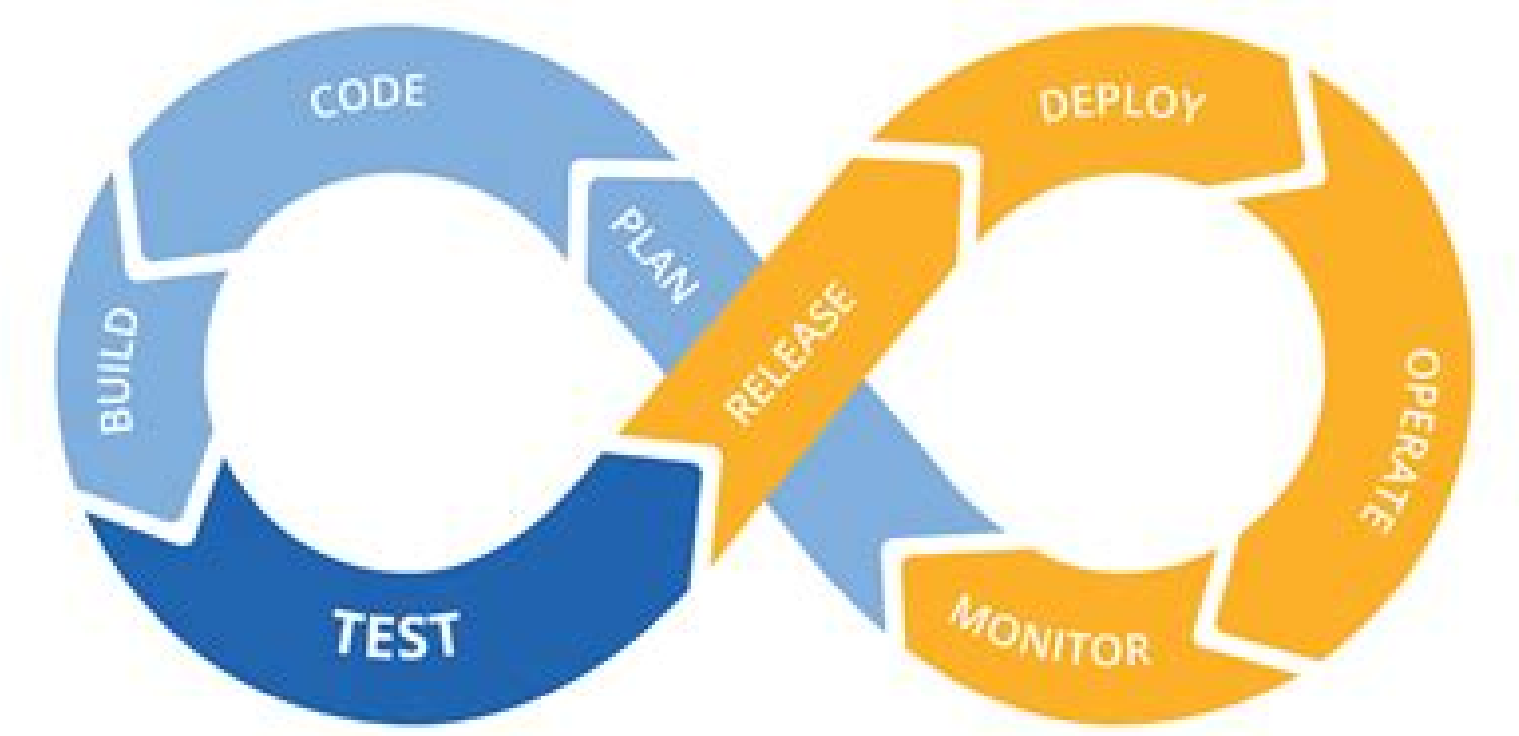
- add buildflags "-fprofile-arcs -ftest-coverage -fPIC" as well as no optimization for code coverage analysis
- build ug with UGTest and your plugin activated
- your plugin contains tests in a top level folder named "tests"
- executable named "ugtest_unit" lands in ug4/bin
- example:

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

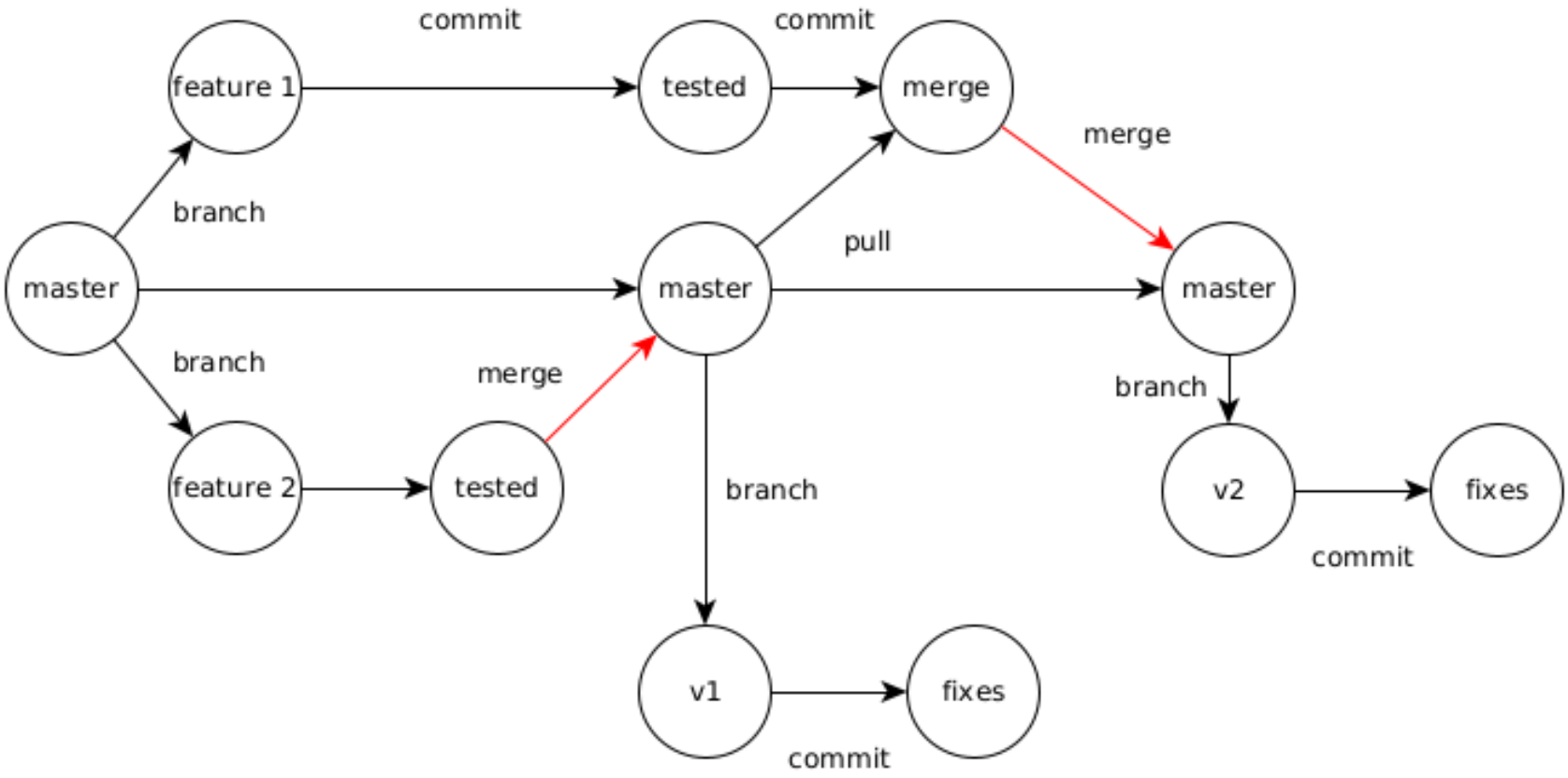
Jenkins

Additional

Continous Integration / Continous Delivery

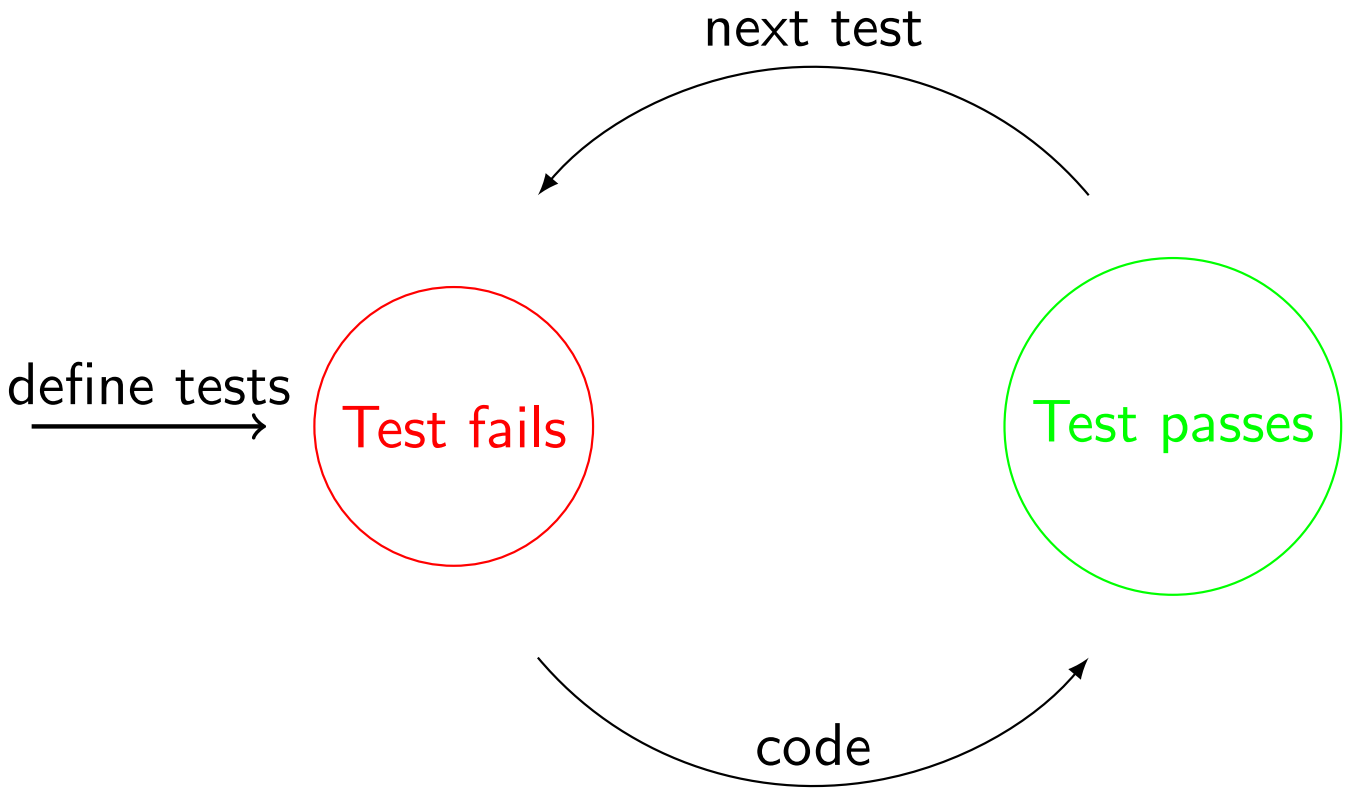


Git branching & releases



sauce

Test driven development



Toolchain

- bugtracking
- documentation user / developer (Doxygen)
- Jenkins
- gcovr (Code Coverage tool)
- git storage
- Docker
- update boost?
- gcc + clang?
- Anforderung nacherfassen!! Danach Tolls richten
- wie requiremetns testen?

Standardization

- definiton of done
- design for testability
- test structure
- naming conventions

Advanced

- test data
- mocking

Additional resources

- Boost.Test executable list of params
- newest Boost.Test
- ugtests github
- Jenkins Pipeline
- Martin Fowler
- SOLID principle
- Jenkins docs

References

- Wikipedi Softwaretests
- Boost.Test 1.58 documentation
- Basiswissen Softwaretest
- Microsoft branching