Softwaretests and ug4

Tobias Trautmann

May 25, 2020

Contents

0.1 Introduction

What is testing?
What do we test for?
Why do we need it?

0.1.1 Debugging vs testing

Debuggin	Testing
unstructured	methodical
only if a bug occurs	continous
only during programming	
	can be highly automated
	covers different granularity

0.1.2 Goals of testing

- increase trust in its results
  - assure that specific functions work
- make code maintainable / refactorable
  - check wether it still functions
- make it sufficiently robust
  - correct exception, edge case handling, corecctness of results, stability
- check if it performs its functions within an acceptable time
  - efficient enough
- check wether in runs its intended environments
  - can be build for its environments
  - environment specifics are considerd

0.1.3 Cans and Can'ts

Testing Software if done correct can do all of those things, but it can't guarantee universal functionailty and isn't a direct improvement to the software.

0.1.4 Defects

Defects are what we're looking for when testing. They can be seperated into different categories by where they come from. All of those defects can be detected by testing. Defects in software exist from the beginning. There is no such thing as deterioration.

bug	fault	issue
error in code	design is lackluster	requirements changed
⇒instability	⇒instability	system still stable
e.g. missing semicolon	e.g. wrong approach	e.g. different button arangement

⇒ Shows up what can actually go wrong, which of those problems are how important and therefore enables testing efficiently.

0.2 Test pyramid

Not actually accurate, but enough for showcasing

0.2.1 Unit tests

whitebox tests, very narrow scope, very granular, no complexity, test unit of behaviour

0.2.2 Component tests

whitebox tests, slightly wider scope than unit tests, component works internally

0.2.3 Integration tests

whitebox tests, even wider scope, check interaction between components (e.g. interfaces)

0.2.4 System tests

whiebox/blackbox test, systemwide scope, test complex user interaction, comples, not granular

0.2.5 Acceptance tests

blackbox tests, user tests wether system is sufficently usable

0.2.6 What & How much to test

Start writing tests as soon as you start programming functional code. Around one fourth to half of your dvelopment time should go into testing. It is nearly impossible to reach 100% test coverage. You will only find new bugs when writing new tests. The more you try to cover the more subtle bugs become.

0.3 Boost.Test

Boost.Test provides automated test discovery, test filters and labeling, exception handling, mocking, test data generation as well as logging.
You should write at least one test per function and constructor. Your tests will want to

cover correctness of output, correct error handling, as well as edge cases (magic numbers).

0.3.1 Structure of a test

testsuite per file looks like this

0.3.2 Basic checks

will show this in a bit assertion levels will be logged differntly

0.3.3 Exceptions

can catch exceptions and check wether no exception or a specific exception was thrown.

0.3.4 Floating point comparison

Is supported, can be relative or absolute

0.3.5 Fixtures

will be called for each testcase. can be defined for single test cases or whole suites

0.3.6 Templates

repetetive tests with different data types

0.4 Testing

0.4.1 UGTest

Buikd flags for gcovr. automated test file discovery. params for boost. test executable. own target?

0.4.2 Jenkins

automation server. can do a lot of things. show UGTest on jenkins. can do code coverage, code sanity, interactions with git, build, release processes, Pipelines

0.4.3 Pipelines

describes workflows, but automated. what steps do i need to do before my code gets a release? What do i want to automate? IS it efficent to automate this Process?

0.5 Not quite testing but important

0.5.1 Software development strategies

CICD: automate as much as you can-¿continous changing software easily doable tdd: extreme frontload for static projects, but testing included from the beginning
Test should be developed along with the productive code to minimize expenses afterwards

0.5.2 Branching

One Branch per feature. features get developed and tested, only when stable test, merge into maste/release branch, ocasional version branches for versioning and minor bugfixes in those versions. pull request?

0.5.3 Toolchain

Categorie	Standad needed?	Tool in use	Tool to use
OS dev		MacOS, Win10, CentOS, Ubuntu	?
OS user		MacOS, Win10, CentOS, Ubuntu	?
OS server		?	?
Languages	y	C++17, Lua	C++17, Lua
IDE		Ecllipse, VSCode	?
Versioning	y	git	git
Publication platform	y	github	?
compiler	y	gcc, clang	?
Libraries	y	Boost 1.58, LAPLAS, BLAS,	?
Documentation user	y	-	?
Documentation dev	y	Doxygen	?
Bugtracking	y	-	?
Work asignment	?	-	?
Automation server	y	Jenkins	?
Testautomation	y	Boost	?
Build automation	y	cmake	?
Code coverage	y	gcovr	?

0.5.4 Documentation

Different Documentations for different people. priorities: 1. devloper to maintain code, 2. user to use product Code is Documentation but not enough.

0.5.5 Design for testability

0.5.6 Definiton of done