[Unit] Testing like a Lazy Pro

or How To Write a Rock-Solid Test Harness

by Marcel Roth and Dino Pollano

Who is Marcel?

Technical University Berlin

Thesis: Musical instrument recognition using Hidden Markov Model

zplane.development

Project: Parameterization of a convolution reverb

DSP-related companies (GER, NL, UK)

18 years of ones and zeros (i.e. 10010 years)

Spitfire Audio

Projects: LABS, CI/CD pipeline, unit testing

Who is Dino?

Bournemouth University

Rebel Technology/OWL Guitar pedal

QMUL

L-ISA / L-Acoustics

Spitfire Audio

- Hans Zimmer Strings
- LABS
- BBCSO
- AIR
- Internal tools/DSP work

What’s to come?

- Testing audio in JUCE
- Unit testing in General
- Demo
- Other testing
- Further reading

TESTING AUDIO IN JUCE

What are we playing with?

VSCode

Juce (CMake project)

Catch2

Github (MCRJuce)

Let’s create ‘Linear Panning’

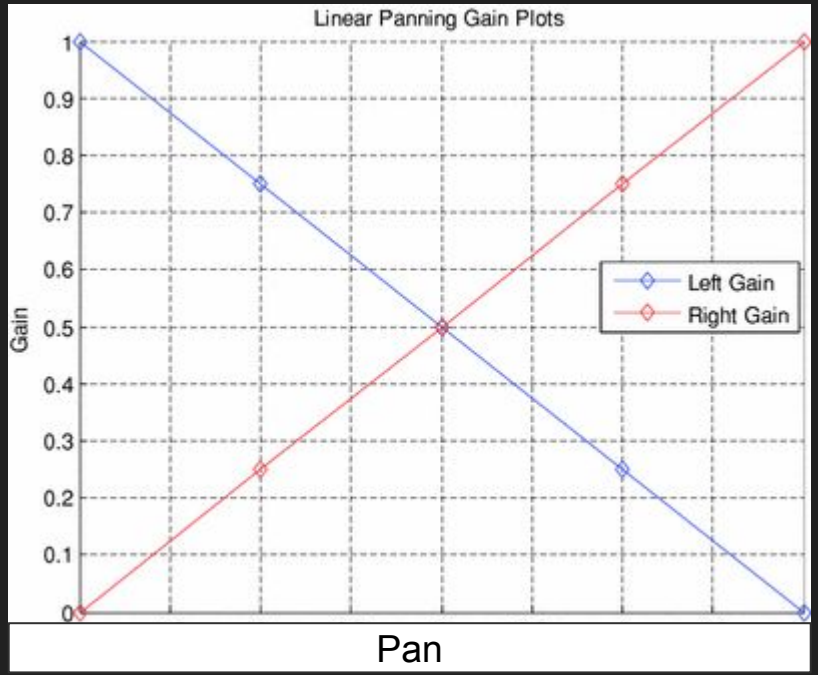
X = (Mono) input signal
AL = Left output channel amplitude
AR = Right output channel amplitude
PAN = Panning value (0 to 1)

Equation:

AL = (1-PAN) * X
AR = PAN * X

Examples:

- PAN = 0, AL = X, AR = 0
- PAN = 1, AL = 0, AR = X
- PAN = 0.5, AL = X/2, AR = X/2



Floating point comparison

REQUIRE (ValueA == ValueB);

=>

REQUIRE (std::abs (ValueA - ValueB) < epsilon);

UNIT TESTING IN GENERAL

What is a unit?

The smallest, possibly testable bit of code.

- A function (or 2)
  - read - write
  - save - load
- A class
  - Reader - Writer
  - Saver - Loader
- A module (is this integrated testing?)
  - ...

TDD and BDD

What is test-driven development (TDD)?

Write the tests first.

What is behaviour-driven development (BDD)?

“Test adding positive numbers”,
“Test adding negative numbers”,
“Test adding fractal numbers”,…

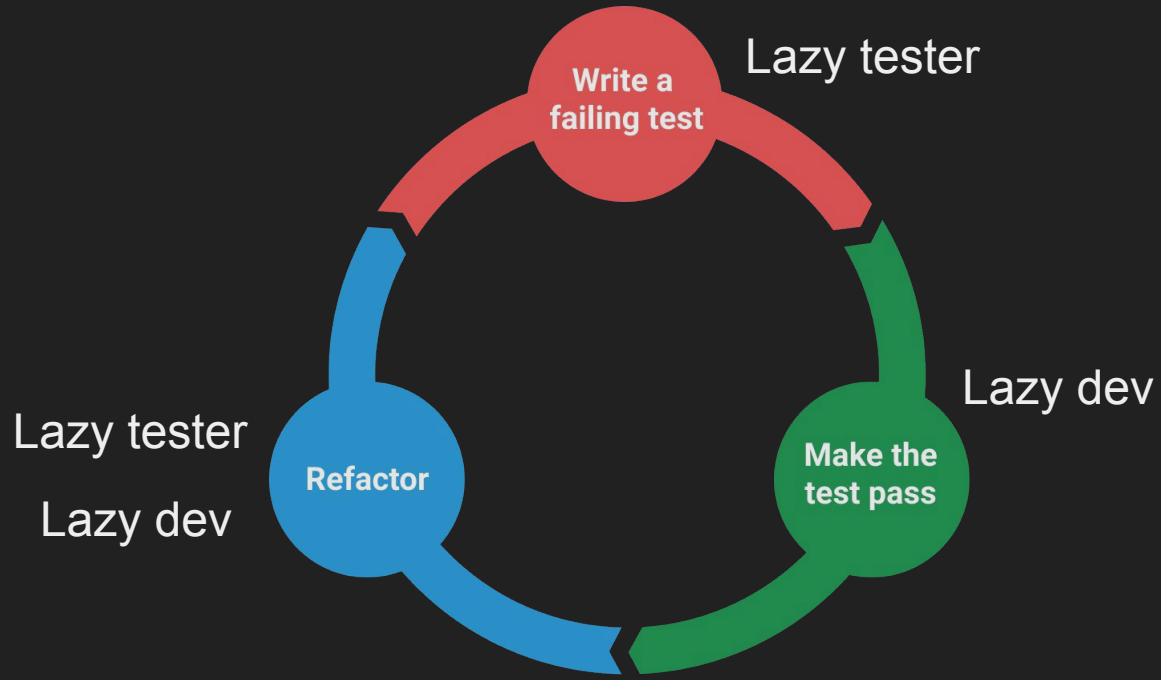
“When a is 1 and b is 1, then result is 2.”
“When a is -1 and b is -1, then result is -2.”
“When a is 0.5 and b is 0.1, then result is 0.6.”

SCENARIO: We want to add numbers

GIVEN: a is 1 and b is 1

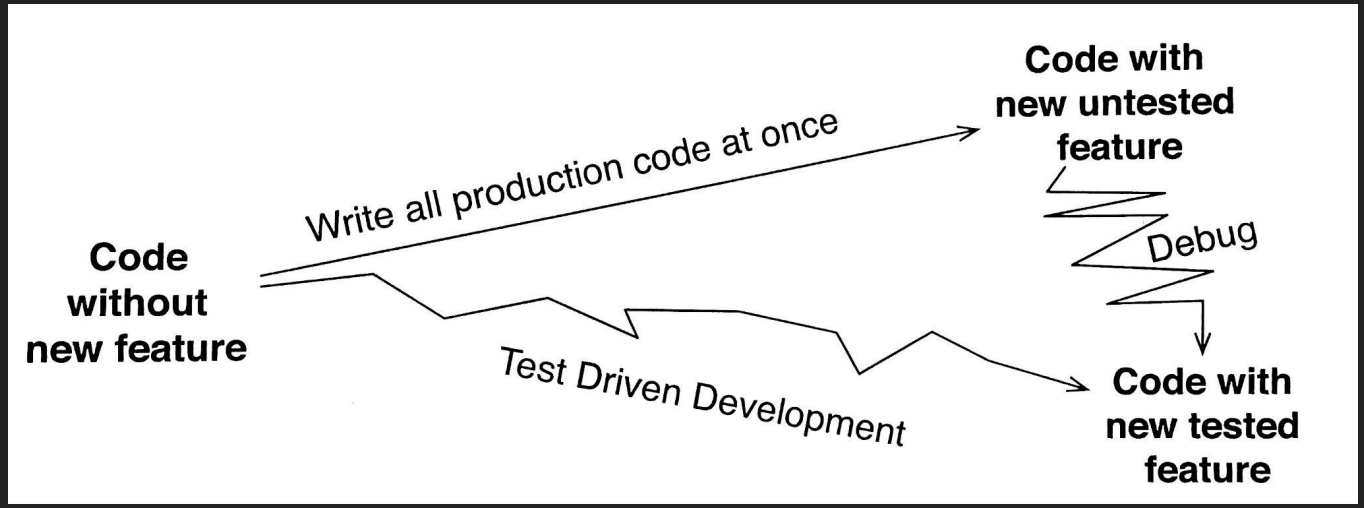
THEN: the result is 2

The lazy circle of a life in unit testing



“It takes too much time”

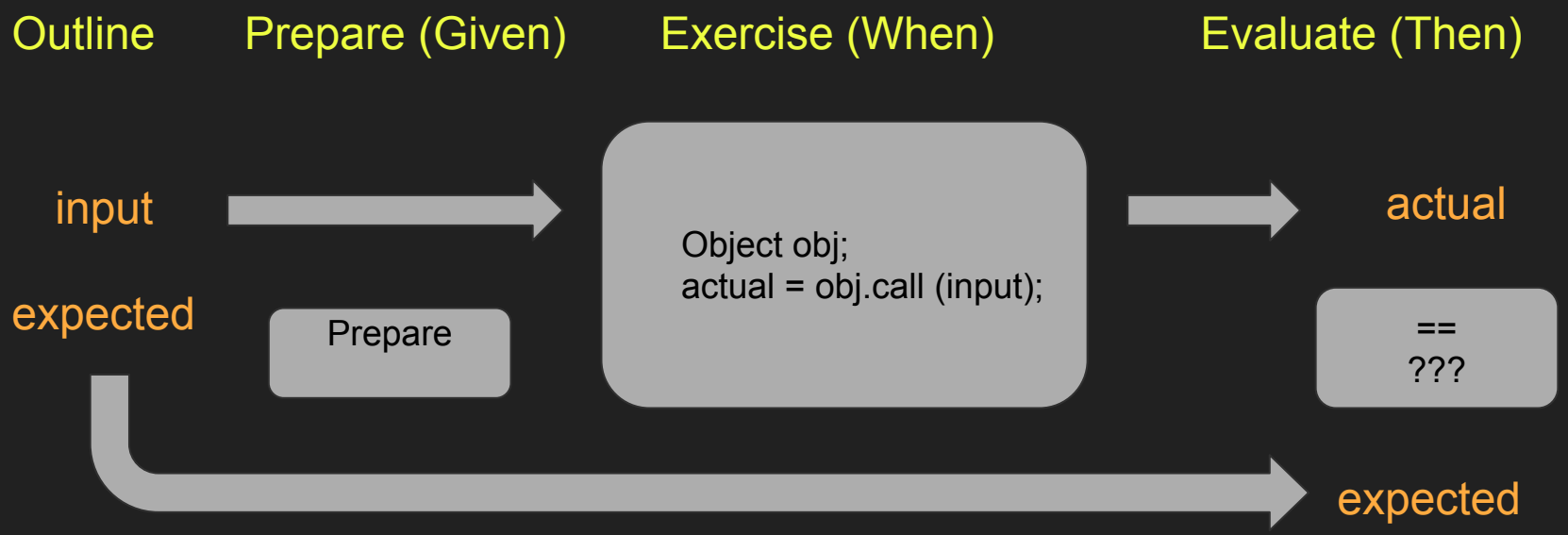
“Debug-Later” Programming vs Test-Driven Development



Rule #1

Thou shall not rewrite the code your testing.

Basic concept



a = 1; b =1;
expected = 2;

mathOp =
add;

actual =
mathOp(a, b);

REQUIRE
(actual ==
expected);

DEMO

OTHER TESTING

Property testing

MONO_INPUT & (PAN < 0.5)

=>

LEFT > RIGHT

LEFT > MONO_INPUT/2

Random input: samples, pan

=>

Conditions are always met.

Test types

Unit tests

There is nothing smaller we can test.

Integrated tests

There are a bunch of units working together in the test.

End-to-End tests

User-like testing. Interacting with the interface.
(Focusrite juce-end-to-end)

Test sizes (Google?)

small, medium, large

Performance tests

Testing with a stopwatch, warm-up, different hardware, different systems

Resource tests

Eyes on the memory.

Manual tests

Mocking

Test => UUT => Collaborators

Car => Engine

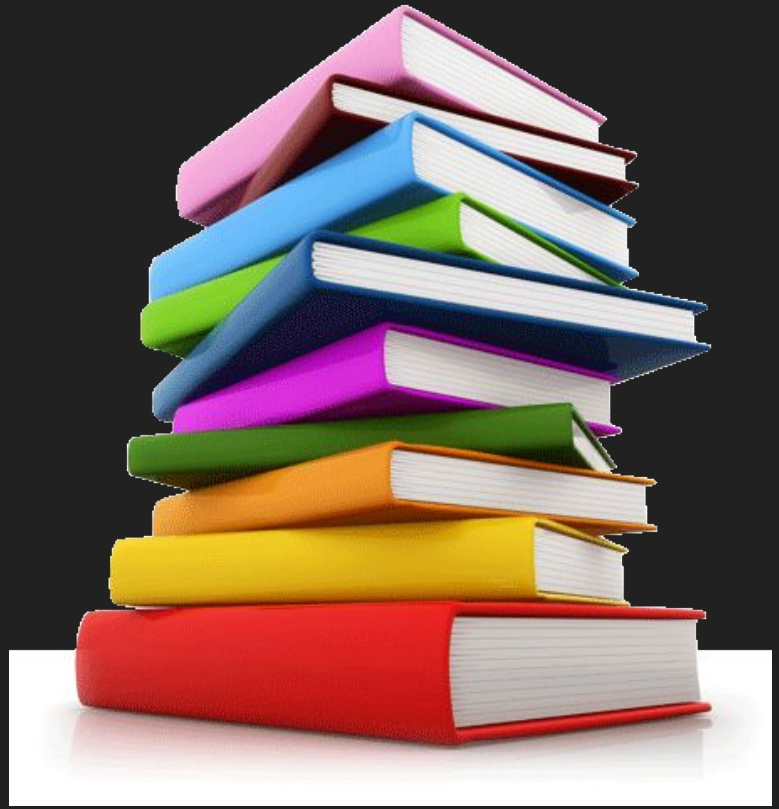
Test: Car => Car => MockEngine

- Mocking framework (GTest, Trompeloeil,...)
- Dependency injection
- Collaborators
- Wrappers (Filesystem)

Kent Beck

- Godfather of unit testing
- Inventor of
  ‘Extreme Programming’
- Part of the
  ‘Agile Development’ Group

FURTHER READING



Links

LinkedIn: Marcel Roth, Dino Pollano

https://github.com/Audiodroid/Smooosie

Property-based testing video

Unit-Testing framework: Catch2

juce-end-to-end (Focusrite)

Test-Driven Development for Embedded C

Unit testing guru: Kent Beck

Any Questions or Further Thoughts?