DEVELOPER DAYS 2014 EUROPE

Test driven development for Invisible Hardware

Samuel Gaist

Edeltech Ltd
Switzerland

Samuel Gaist

Software Development Engineer

- Partner at Edeltech Ltd
- Developer at IDIAP Research Institute

Qt Experience

- User since Qt 3.2
- Contributor for ~ 2 years (QtCore, QtGui, QtMacExtras)
- 10’000+ posts in Qt Forum

Outline

1. AgiScan, a film scanning application
2. Invisible Devices
3. Test Driven Development with Invisible Devices
4. Demo
5. Wrap Up
6. Q/A

1. AgiScan, a film scanning application

USB

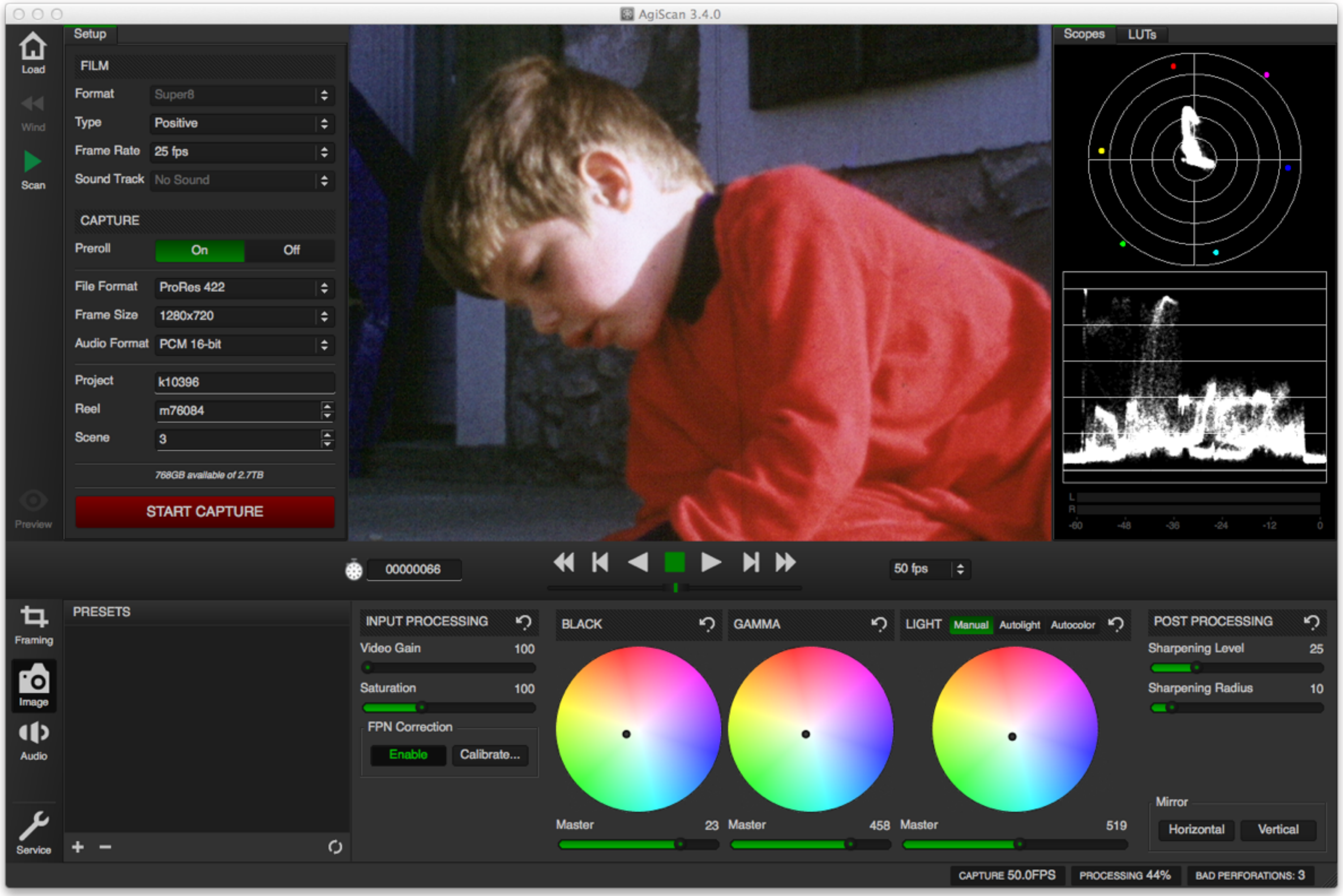
Embedded App
Proprietary

Audio/Video

Qt Desktop App
Mac OS / Windows

DEVELOPER DAYS 2014 EUROPE

1. AgiScan, a film scanning application



1. AgiScan, a film scanning application

The Application (AgiScan) requires the device (Film scanner).

- to setup the GUI since the App supports different scanner models with different hardware options.
- to setup the audio and video capture subsystems since each scanner model produces different resolutions, image formats, etc...

2. Invisible Devices

- It can’t be moved next to your desk.
  because it’s too big or heavy
- It doesn’t exist yet.
  next week, promised
- It exists but you can’t have it.
  because someone needs it or it’s too expensive
- The only one we have has been sold yesterday.
  we’ll build you a new one if we get an order...

2. Invisible Devices

I can’t have one = INVISIBLE

So how do I develop my Application ?

3. Test Driven Development with Invisible Devices

Write a Mock Object

to mimic the behaviour of the Invisible Device

in controlled ways.

3. Test Driven Development with Invisible Devices

Use an Adapter Object

to communicate with the device when it is available,

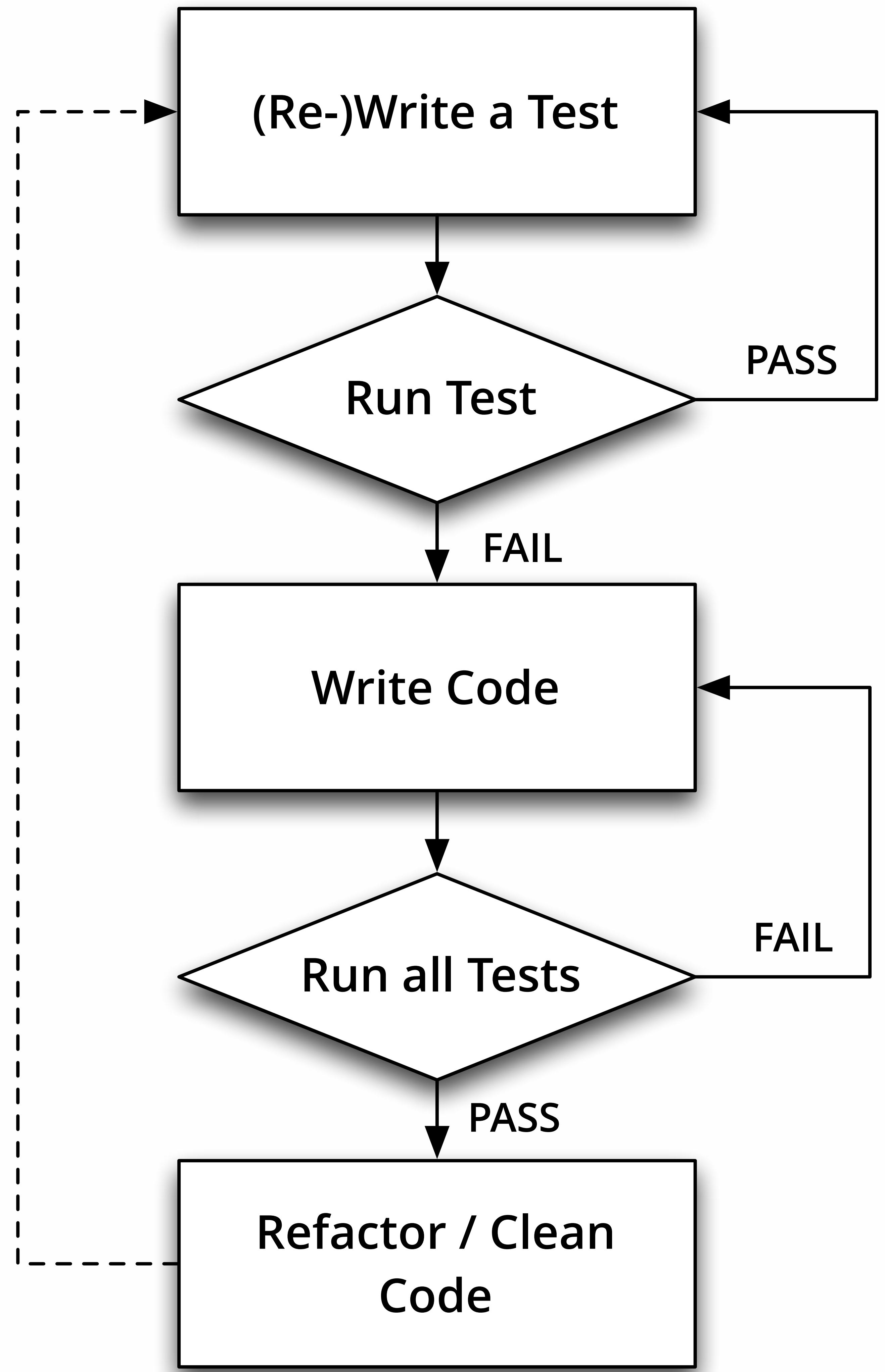
or with the Mock Object.

Apply TDD principles

because we want to sleep well.

3. Test Driven Development with Invisible Devices

TDD Friendly Reminder



http://en.wikipedia.org/wiki/Test-driven_development

TDD Benefits

1. Know how your code works.
2. Know that your code works.
3. Help other developers understand your code.
4. Know when you break something.

Levels of testing

UNIT TEST

Each class, e.g. the Mock Object

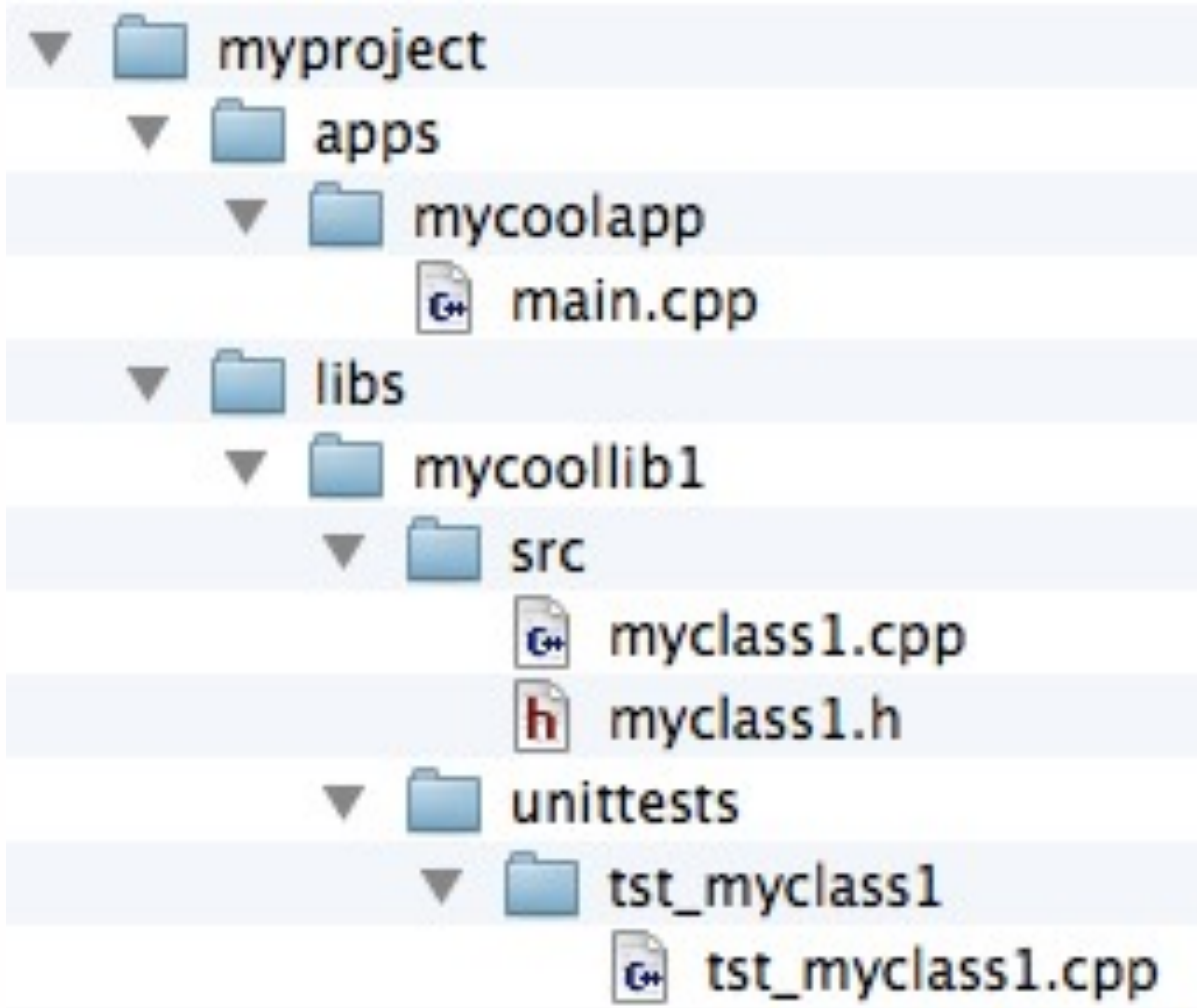
FUNCTIONAL TEST

Several classes, e.g. Mock Object + Adapter

SYSTEM TEST

Application: e.g. Scenario based user interaction, Squish

Project Architecture



Using QTest

```
#include <QtTest>
#include <QCoreApplication>

#include "mycoolclass.h"

class UnitTestMyCoolClass : public QObject
{
    Q_OBJECT

private Q_SLOTS:
    void testCase1();
};

QTEST_MAIN(UnitTestMyCoolClass);

#include “tst_UnitTestMyCoolClass.moc”
```

Let’s write a unit test.

Know your Device

1. **What interface does it provide ?**

   Serial Port, Network, ...

2. **What is the communication protocol ?**

   Commands, Responses

To write the Mock Object.

The Mock Object

```
#include <QObject>

class MockDevice : public QIODevice
{
    Q_OBJECT

public:
    MockDevice();

protected:
    qint64 readData(char *data, qint64 maxlen) Q_DECL_OVERRIDE;
    qint64 writeData(const char *data, qint64 len) Q_DECL_OVERRIDE;
};
```

Define the Controller API

1. **How will the application interact with the controller ?**

   Signal/Slots, Events, other ?

2. **Level of abstraction**

   Map the “device lingo” to sensible methods / properties.

To write the Adapter Object.

The Adapter Object

```
#include <QObject>

class Controller : public QObject
{
    Q_OBJECT

public:
    Controller(QObject *parent = 0);

    void setDevice(QIODevice *device);

public Q_SLOTS:
    void doSomething();
    void doSomethingElse();

private:
    QIODevice *_device;
};

void Controller::doSomething()
{
    _device->write(DoSomethingCommand);
}
```

QIODevice

QBuffer, QSocket, QSerialPort…

3. Test Driven Development with Invisible Devices

Let’s write a controller and test it.

Detect the actual device

Using an enumerator (Bonjour, QSerialPortInfo)

Write a Factory

to return the Actual Device or the Mock Device.

and pass it to your Controller

using ::setDevice(QIODevice*)

4. Demo

Demo

5. Wrap up

- Using a Mock Device we can develop an application controlling a device without the actual hardware.
- The Tests ensure that the Mock Device and the Application work correctly, and allow developers to reproduce bugs occurring with the actual hardware.
- The Mock Device can be used to create a Demo Mode.

Q/A

Questions ?