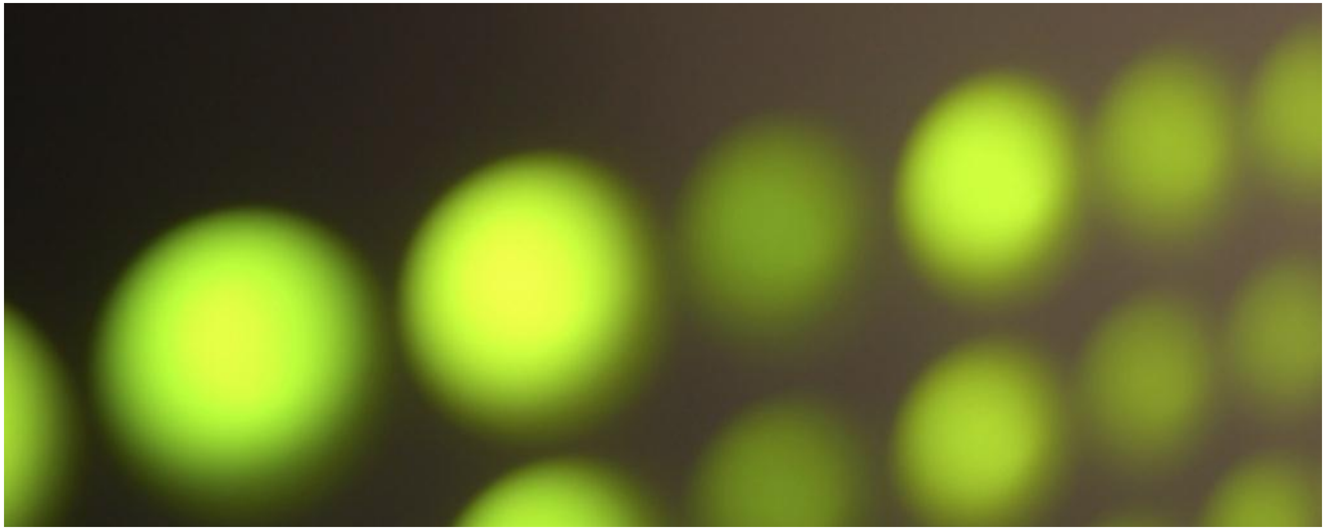
Full-On TDD of Embedded Software

1:30pm, Monday 2 July 2012, BCS SPA conference



Immo Hüneke

© Zühlke 2012

Test Driven Development – a quick refresher

Immo Hüneke

© Zühlke 2012

Origins of TDD

Kent Beck, who is credited with having developed or "rediscovered" the technique, stated in 2003 that TDD encourages simple designs and inspires confidence.

A 2005 study found that using TDD meant writing more tests and, in turn, programmers who wrote more tests tended to be more productive.

© Zühlke 2012

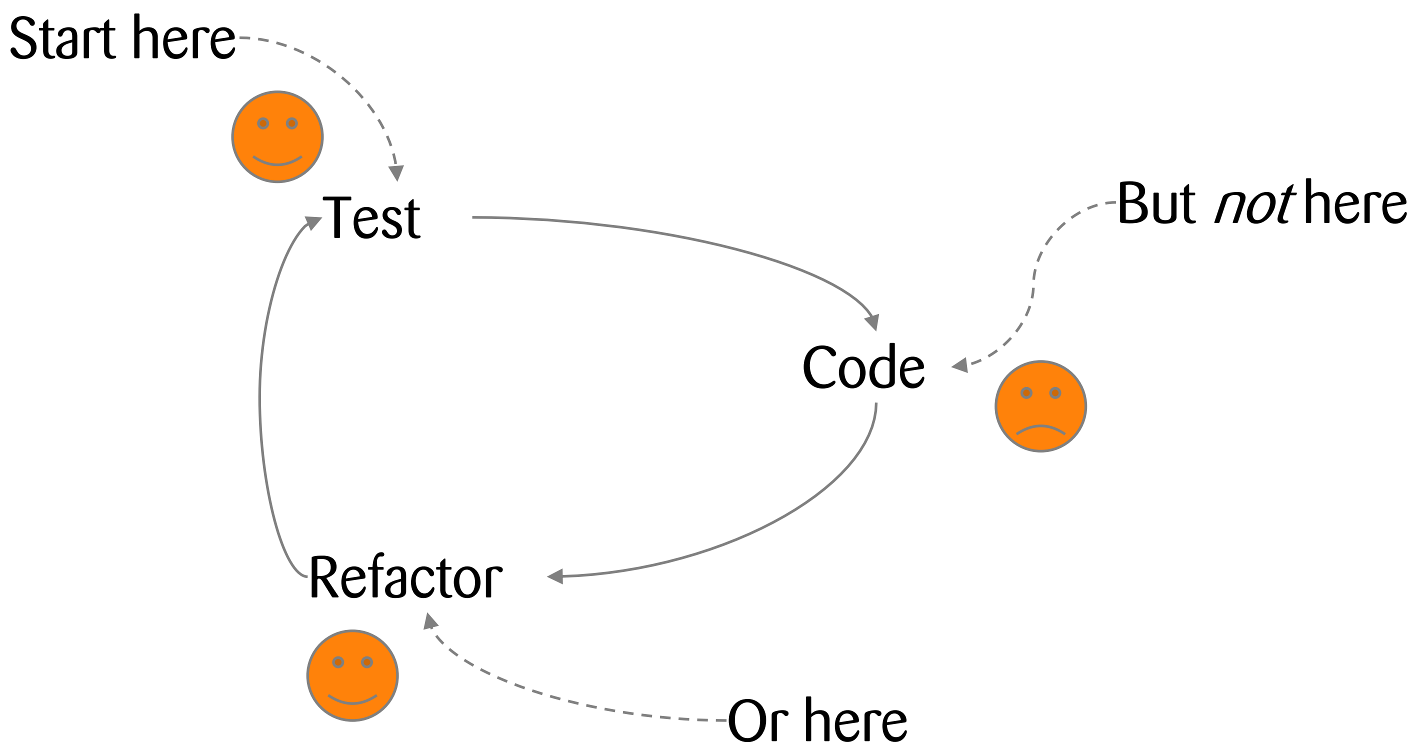
TDD is a Good Fit for Embedded Software

Leads to a sound software architecture

- Good hardware abstraction
- Fully tested software components deployed
- No defects, few surprises

 © Zühlke 2012

The TDD cycle



Rule: Only change the code base as a response to a failing test

© Zühlke 2012

TDD Rule

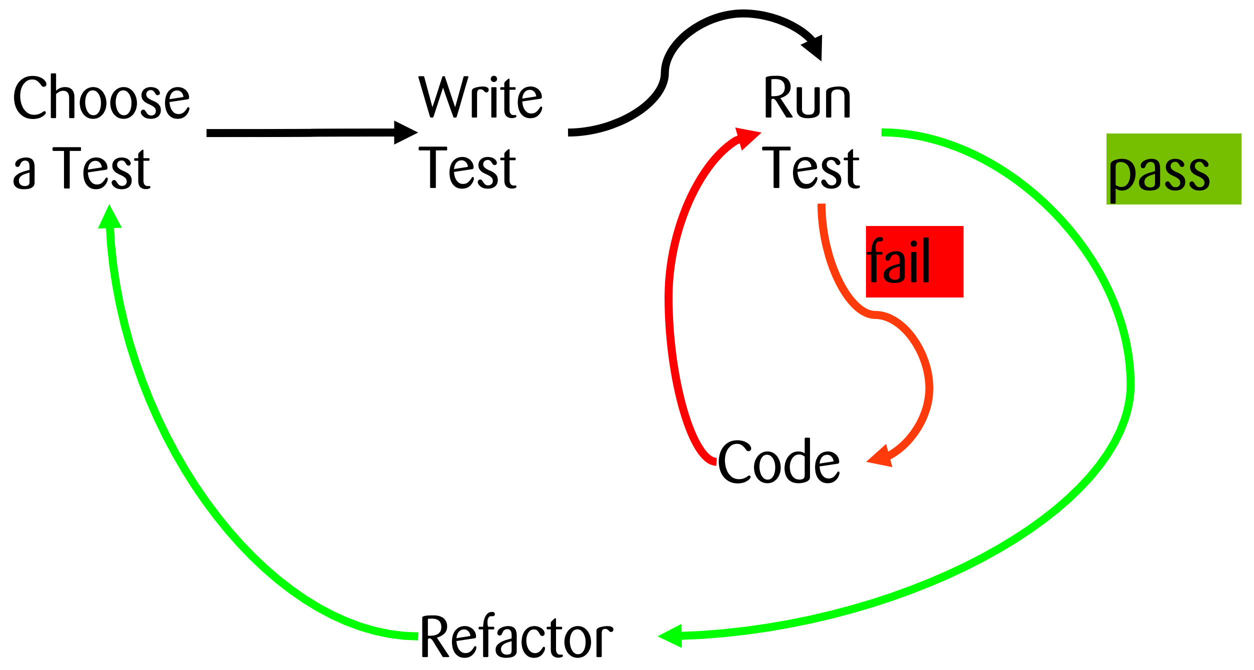
Only

- add new code
- or enhance existing code

in response to a failing test

 © Zühlke 2012

The TDD cycle in detail



Failure could be compilation, link or programmatic failure

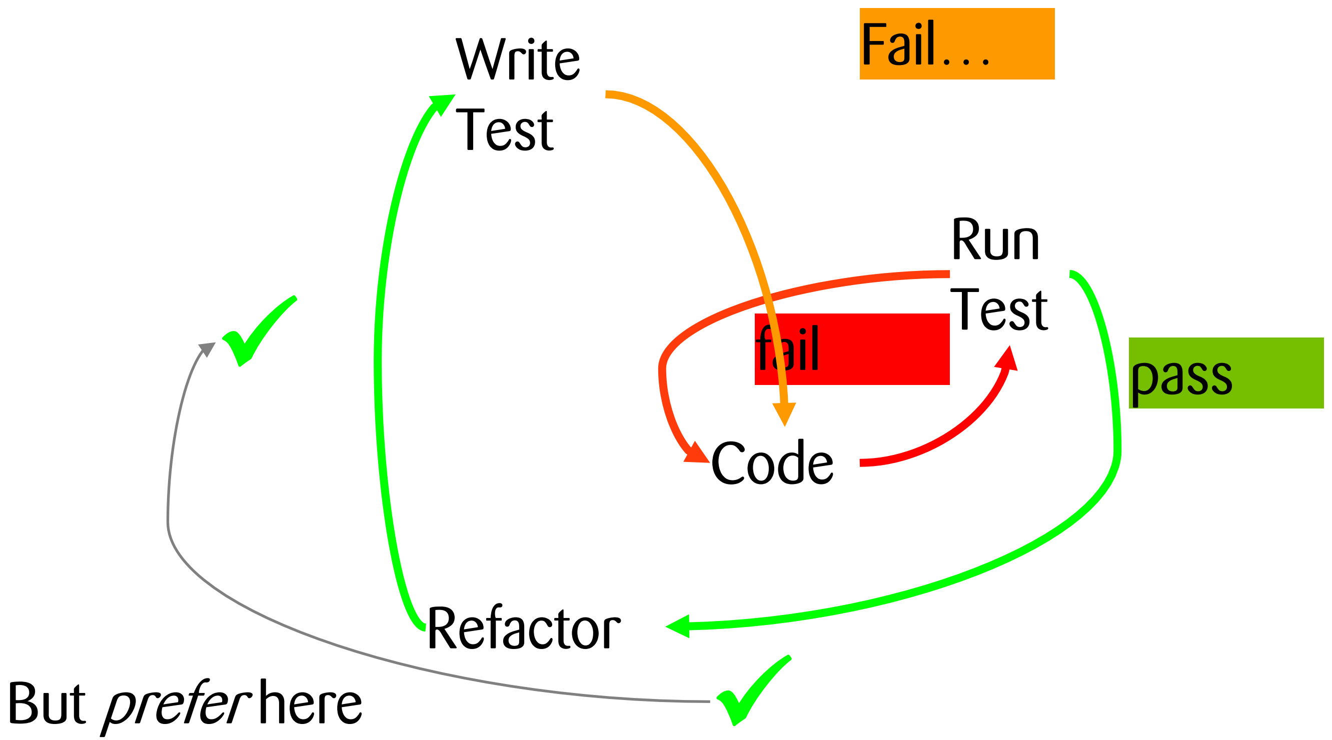
© Zühlke 2012

TDD Rule

1. Code a failing test
2. Announce boldly *why* it will fail
3. *See* it fail
4. Do the *simplest thing* to make it pass
5. (and keep *all* the others passing)
6. Go to 1

© Zühlke 2012

The TDD cycle: when to check-in



 © Zühlke 2012

TDD Rule

Check in only on a green bar

- and only when you have code you're proud of
- "When the bar is **green**, the code is clean" – K. Beck

New functionality is added during "orange bar" time

- The time during which tests fail as expected
- Most xUnit's don't have an **orange** bar, so…
- "When the bar is **red**, we're forging ahead" – K. Braithwaite

Unfortunately the unit-test tooling for C / C++ tends not to display any coloured bars at all ☹

© Zühlke 2012

Available Tools

Commercial Tools

- Cantata++ (C and C++)
- Various others, which I have not tried and can't recommend

Free Open Source Tools

- Unity (C)
- CppUnit (C++)
- CppUTest (C and C++)
- Google C++ Mocking and Testing frameworks (C++)
- Boost Test Libraries (C and C++ - but mainly C++)

© Zühlke 2012

Available Tools

Commercial Tools

- Cantata++ (C and C++)
- Various others, which I have not tried and can't recommend

Free Open Source Tools

- Unity (C)
- CppUnit (C++)
- CppUTest (C and C++)
- Google C++ Mocking and Testing frameworks (C++)
- Boost Test Libraries (C and C++ - but mainly C++)

© Zühlke 2012

References

Wikipedia

- Test-driven development
- List of unit testing frameworks
  - Follow download links from here

Literature

- Grenning, James: *Test Driven Development for Embedded C* (Pragmatic Programmers 2011)
- James's web site: http://www.renaissancesoftware.net/

© Zühlke 2012

First Example Exercise

Driving an alarm sounder

Immo Hüneke

© Zühlke 2012

C Function Example

Activate an alarm sounder

Simple buzzer

Two bits of a memory-mapped register

- One controls sound on/off
- The other bit controls pitch high/low
- By driving both bits, can play very simple tunes

Want an API that program components can call to generate different ack/alarm sounds

© Zühlke 2012

Suggested API

Data types:

```
typedef enum{AlarmDriver_LOW,
AlarmDriver_HIGH} AlarmDriver_Pitch;
typedef unsigned int
AlarmDriver_Duration;
```

Operations:

```
void AlarmDriver_play(AlarmDriver_Pitch
pitch, AlarmDriver_Duration duration);
void AlarmDriver_stop();
```


© Zühlke 2012

Getting Started

Create basic project structure from the command line (shock, horror) using ceedling

Run a shell on the root of your memory stick

```
> _SETPATH.BAT
> cd \work\SPA2012
> ceedling new Buzzer
> cd Buzzer
> rake module:create[AlarmDriver]
```

Import project into Eclipse

- New → [Other →] C project → Name: Buzzer
- Project type: empty executable; toolchain: MinGW

© Zühlke 2012

Getting Started (cont.)

We could install Aptana's RadRails workbench

Simpler to just configure rake as external tool

- Run → External Tools → External Tools Config's
- Right-click "Program" and select "new"
- Name: TestBuzzer
- Location: \Ruby193\bin\rake.bat
- Working directory: select variable "project_loc"
- Arguments: **test:all**
- Run (test compiles and runs module AlarmDriver)

© Zühlke 2012

Expected output

```
Test 'test_AlarmDriver.c'
-------------------------
Generating runner for test_AlarmDriver.c...
Compiling test_AlarmDriver_runner.c...
Compiling test_AlarmDriver.c...
Compiling AlarmDriver.c...
Compiling unity.c...
Compiling cmock.c...
Linking test_AlarmDriver.out...
Running test_AlarmDriver.out...

-------------------------
IGNORED UNIT TEST SUMMARY
-------------------------
[test_AlarmDriver.c]
  Test: test_module_generator_needs_to_be_implemented
  At line (14): "Implement me!"

-------------------------
OVERALL UNIT TEST SUMMARY
-------------------------
TESTED:   1
PASSED:   0
FAILED:   0
IGNORED:  1
```


© Zühlke 2012

First Test Suite

Add API definitions to AlarmDriver.h

Extend API with initialisation method

```
#include <stdint.h>

…
void AlarmDriver_init(uint16_t
                          * registerAddress);
```

Create “mock hardware” in test_AlarmDriver.c

```
static uint16_t hwRegister;
```

Initialise the fake hardware register

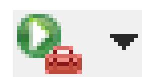
```
void setUp(void) {
  hwRegister = 0x5555;
}
```


© Zühlke 2012

First Test Suite (cont.)

Replace generated test in test_AlarmDriver.c

```
void test_initedAlarmShouldBeOff (void) {
  TEST_ASSERT(hwRegister != 0);
  AlarmDriver_init(&hwRegister);
  TEST_ASSERT_EQUAL(hwRegister, 0);
}
```

Run the test by clicking “External Tool” icon 

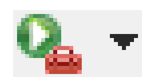
What happens and why?

© Zühlke 2012

First Test Suite (cont.)

Replace generated test in test_AlarmDriver.c

```
void test_initedAlarmShouldBeOff (void) {
  TEST_ASSERT(hwRegister != 0);
  AlarmDriver_init(&hwRegister);
  TEST_ASSERT_EQUAL(hwRegister, 0);
}
```

Run the test by clicking “External Tool” icon 

What happens and why?

Now add an implementation to pass this test

Refactor (how?) and **check in on green**

© Zühlke 2012

Suggested Test List

Check correct register value written for on/low pitch

Check correct register value written for stop

Check correct register value written for on/high pitch

Check buzzer stops after the requested time

- You will need a mock clock interrupt for this

Check safe behaviour if not initialised

© Zühlke 2012

Second Example Exercise

Reading a Temperature Sensor

Immo Hüneke

© Zühlke 2012

C++ Function Example

Read a temperature sensor

Temperature is proportional to duty cycle

See http://www.smartec.nl/pdf/DSSMT16030.PDF

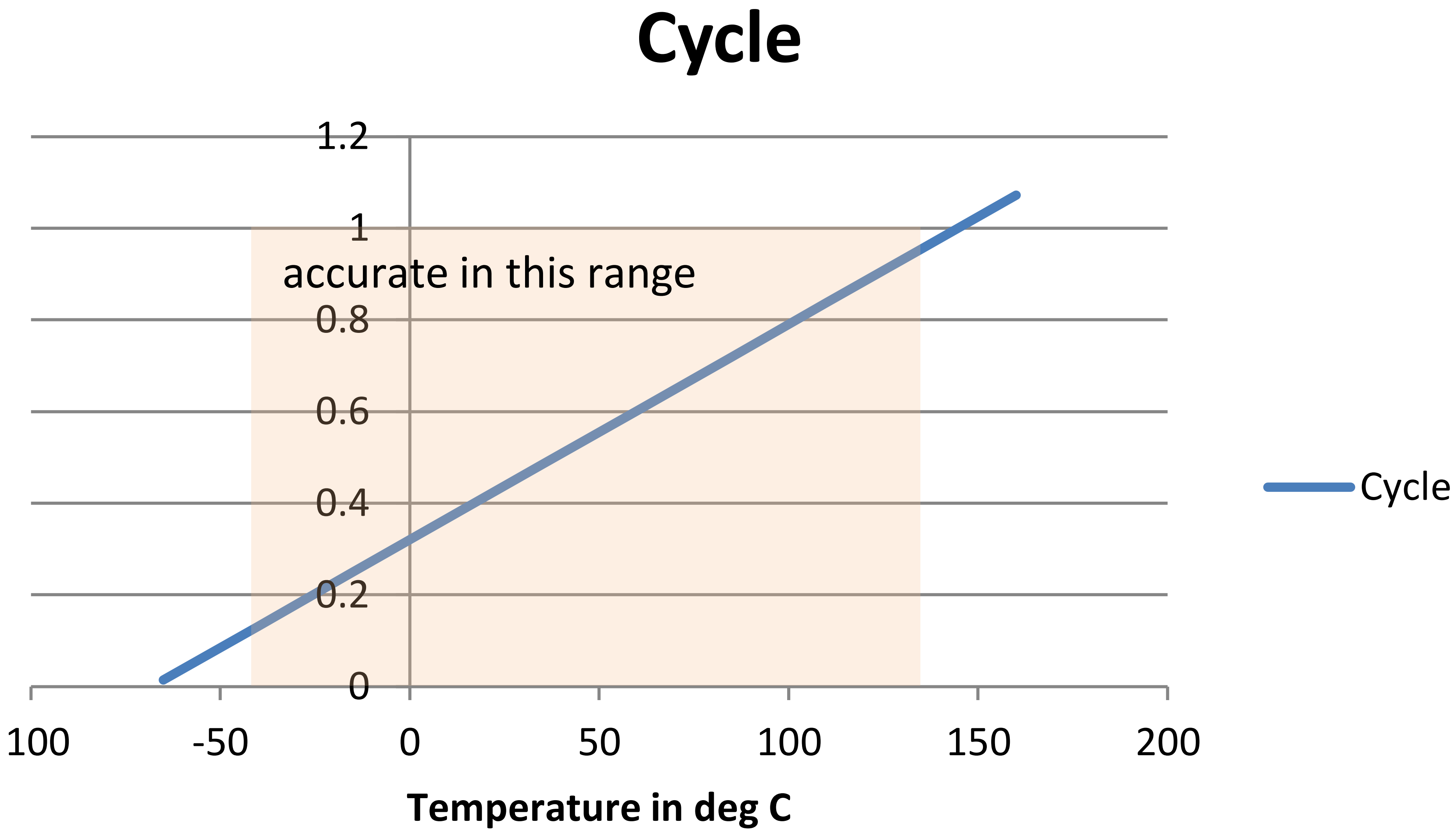
Frequency 1-4 KHz

Duty cycle = 0.320+0.00470*t (t=degrees C)

Range -65 to +160C (accurate -45 to +130C)

© Zühlke 2012

Temperature Sensor Response Curve



© Zühlke 2012

Reading the sensor

Need to detect ratio of time at +5V to total waveform period

Could feed into big capacitor & read analog

- Need A-to-D conversion

Could sample several times per ms and average

- Long time to converge
- Beat effects

Could sample hundreds of times per ms

- Blocks CPU for significant amount of time

Could generate an interrupt on both rising and falling edge

- Simple circuitry: one latch to generate interrupt, one to read value
- Use system clock to determine time elapsed

© Zühlke 2012

Suggested API (low level)

Data Types:

```
class TempSensorHwIface;
class TempSensorHwIface::Handler; // abstr
class MicroSecClock;
```

Operations (not including constructors etc):

```
void TempSensorHwIface::registerHandler(
    TempSensorHwIface::Handler * handler);
void TempSensorHwIface::interrupt();
bool TempSensorHwIface::isDutyCycleHigh();
void MicroSecClock::reset();
long MicroSecClock::elapsedTime();
```


© Zühlke 2012

Suggested API (high level)

Data Types:

```
class TemperatureSensor;
```

Operations (not including constructors etc):

```
void TemperatureSensor::reset();
void TemperatureSensor::update();//callbck
bool TemperatureSensor::valid();
float TemperatureSensor::degCelsius();
```


© Zühlke 2012

Getting Started

New C++ project

- Name: TemperatureSensorTest
- Type: Executable / Hello World C++ Project
- Toolchain: MinGW GCC

Advanced Settings button [All Configurations]

- CppUTest
  - Add C++ include path E:\CppUTest\include
  - Add library CppUTest & search path E:\CppUTest\lib
- Add error parsers for GNU Make and Linker

© Zühlke 2012

Build and run just for fun

Click in source of “main”

Select “debug” from the build configurations (little hammer icon)

Verify that build completes OK

Once built, select project at the left and run it

Verify that “Hello World” message appears in console

© Zühlke 2012

Convert to a test project

Full source code of TemperatureSensorTest.cpp:

```
#include "CppUTest/CommandLineTestRunner.h"

int main(int argc, char** argv) {
  return CommandLineTestRunner::RunAllTests(argc, argv);
}
```


© Zühlke 2012

Check that test harness runs

Build and run it again

Output should look like this:

```
OK (0 tests, 0 ran, 0 checks, 0
ignored, 0 filtered out, 0 ms)
```

You are now ready to start adding test cases

Test cases go in this project, production code in another project called TemperatureSensor

© Zühlke 2012

Create the production project

Create an empty static library C++ project

- Name: TemperatureSensor
- Toolchain: MinGW GCC
- Create folder src, add it as a source folder and tell Eclipse to ignore the default source folder

Create an interface in src folder

- New C++ class: TemperatureSensor
- Namespace: TemperatureSensor
- Unit test: TemperatureSensorTest.cpp

Move the test source file to the test project

© Zühlke 2012

Create production project (cont.)

Build the new project (all configurations)

Add the new project to test project build path

- Properties → C/C++ General → Paths and Symbols
  - All Configurations → Includes → Add → Workspace → /TemperatureSensor/src
  - All Configurations → Libraries → Add → TemperatureSensor
  - Debug → Library Paths → Add → Workspace → /TemperatureSensor/Debug
  - Release → Library Paths → Add → Workspace → /TemperatureSensor/Release

© Zühlke 2012

First test case

Write a test that calls the interface (see next)

- Will it compile?
- Will it link?
- Will it run?
- Will it pass?

© Zühlke 2012

TemperatureSensorTest.cpp

```
#include "CppUTest/TestHarness.h"
#include "TemperatureSensor.h"
namespace TemperatureSensor {
  TEST_GROUP(TemperatureSensorTestGroup) {
    void setup() {}
    void teardown() {}
  };
  TEST(TemperatureSensorTestGroup, \
      TemperatureIsNotValidUntilReset) {
    CHECK_EQUAL(false, sensor->valid());
    sensor->reset();
    CHECK_EQUAL(true, sensor->valid());
  }
} /* namespace TemperatureSensor */
```


© Zühlke 2012

Getting the test to compile

Create a pointer to an instance of sensor:

```
...
namespace TemperatureSensor {
TemperatureSensor * sensor;
...
```

Add missing methods to class declaration:

```
...
   virtual ~TemperatureSensor();
   void reset();
   bool valid();
//float degCelsius();
};
```


© Zühlke 2012

Getting the test to link

Implement the missing methods

- Place insert cursor where the methods should go
- Right-click method in header file
- Choose Source → Implement Method
- Fill in method bodies (minimal implementation)

```
static bool isValid = false;
void TemperatureSensor::reset() {
  isValid = true;
}
bool TemperatureSensor::valid() {
  return isValid;
}
```


© Zühlke 2012

Getting the test to run

Does the test run?

Why would you not expect it to?

Why is your expectation confounded?

© Zühlke 2012

Exercises

Add tests to measure the temperature

- Sketch out a test list to give yourself a backlog
- You will need a mock hardware clock and sensor reading (see next slide)
- These have been developed for you to save time: copy from ModelAnswers folder, study and use

Modify requirements: allow temperature reading to settle before reporting "valid"

- How long for?

© Zühlke 2012

What is a Mock?

zühlke
empowering ideas

A stub

- Implements an protocol
- Returns sensible values
  - Hard coded
- Simulates expensive or hard to obtain objects
  - Data sources
  - Proxies for remote systems
  - Non-deterministic objects, especially time-based ones

A Mock is a stub, and more

- Mocks have *expectations*
- Will *fail the test* if not used as specified
- Play roles in collaborations

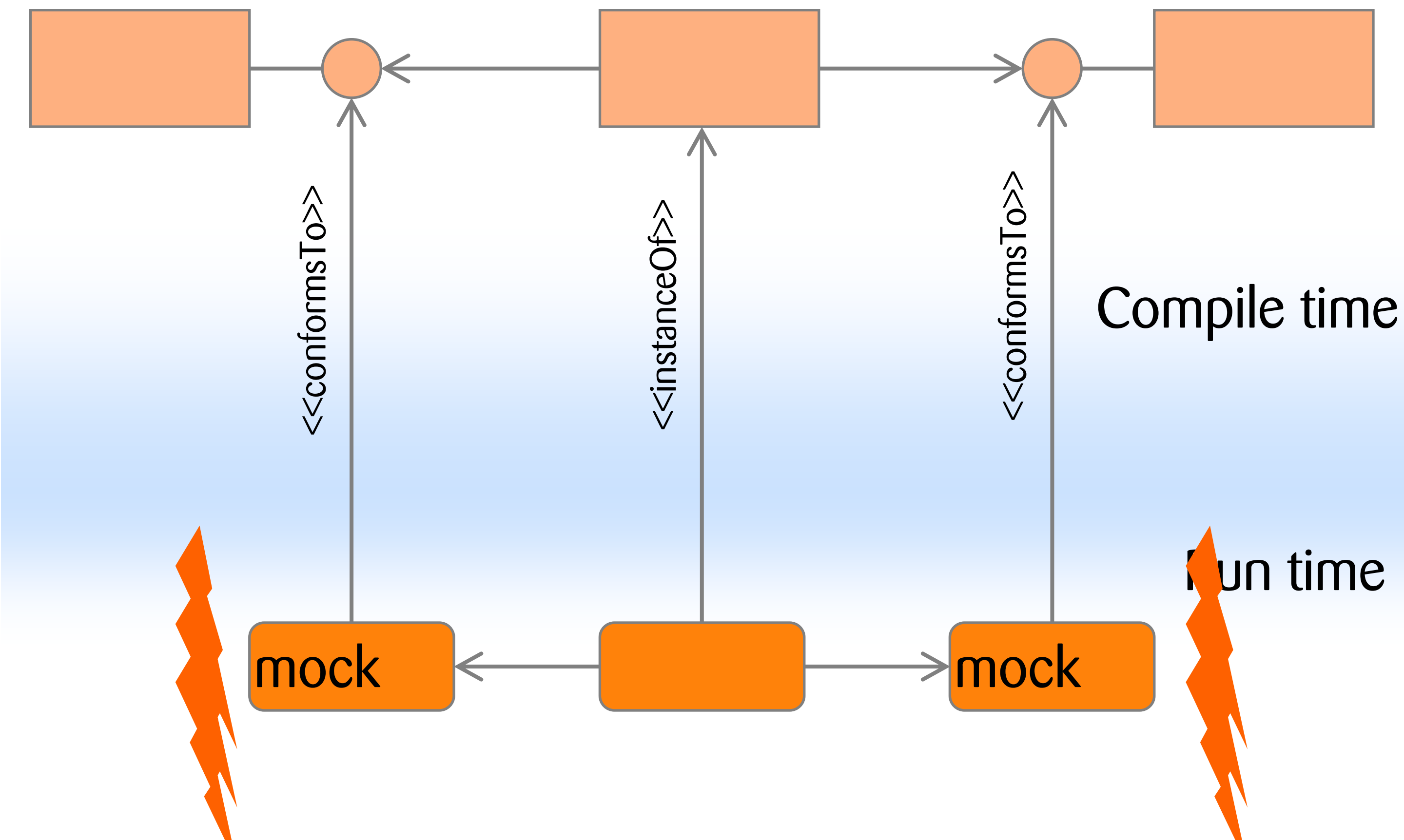
© Zühlke 2012

Mock Test Scenarios

- **Mocked tests are executable CRC scripts**
- The "unit" collaborates with other roles to fulfil its responsibilities
- Collaborators will complain if they are misused

© Zühlke 2012

Mock Roles



© Zühlke 2012

Technique Notes

Use Mocks to capture design ideas

- Speculative implementation of interfaces
- Explore collaborations without creating a class first

Use Mocks to enhance design

- Capture CRC thinking
- Pass in a mock
  - Data source
  - Strategy object
  - State object
  - Other collaborators: dependency injection

But *don't* Mock away the whole world

- ~~"A unit test has only one object that isn't a mock"~~

© Zühlke 2012

Mock implementation of hardware

Create an interface and test implementation in the production project

- New C++ class: TempSensorHwIface
- Namespace: TemperatureSensor

Fill in low-level API details (see earlier slide)

Create a derived mock temperature sensor hardware interface in the test project

- Unit test: MockTempSensorHwIfaceTest.cpp
  - It can be useful to unit-test mocks!

© Zühlke 2012

Retrospective

What did we learn?

Immo Hüneke

© Zühlke 2012