Courage!

TDD and embedded software

Matthew Eshleman
covemountainsoftware.com

Background - Matthew Eshleman

- 15+ years of embedded software development, architecture, management, and project planning
  - Delivered 30+ products with many more derivative projects. Millions of people have used my code. No one has threatened me yet (*except one guy in marketing*.)
- Recently: leading a safety and quality focused firmware project using TDD methodologies.
- Learn more: http://covemountainsoftware.com/consulting

Background - inspiration

- First conceptual exposure to TDD was about 10 years ago at a conference. The speaker stated: “Test driven development gives you COURAGE.” That comment went straight to my core. I got it.

- Last year: read the book “Test Driven Development for Embedded C” by James Grenning, providing further motivation.

Recently…

- Leading firmware development for splitsecnd's crash detection and emergency response device.
- Test Driven Development - a must for a device intended to help people in emergency situations.
- **Today**: overview of TDD and lessons learned.

What is TDD? (Test Driven Development)

- Write tests before writing production/target code!
- Elecia and Grenning in http://embedded.fm/episodes/109:
    - “Transform your debug time into TDD time”
    - Grenning: TDD vs “Debug Later Programming” (DLP)

Benefits of TDD

- Functionality is tested without hardware
- Test code without backends/servers
- Error cases that are difficult or nearly impossible to test in the real world can be tested in the test project/framework.
    - Example: DNS failure. Happens, but…
- The tests act as a supplement or even replace specifications
- Tests supplement or replace code documentation
- **Courage** to refactor, rework, and rapidly iterate.

Test Driven Development work flow

- Write a test - It will not compile, this is clearly a “fail”.
- Fix headers. Add some shell code. Build, compile, link.
- Run Tests. New tests **should fail**.
- Develop code until it passes the test.
- Refactor.
- One Behavior At a time.
- Keep Tests fast. Encourages their use.

TDD needs a good Test Framework

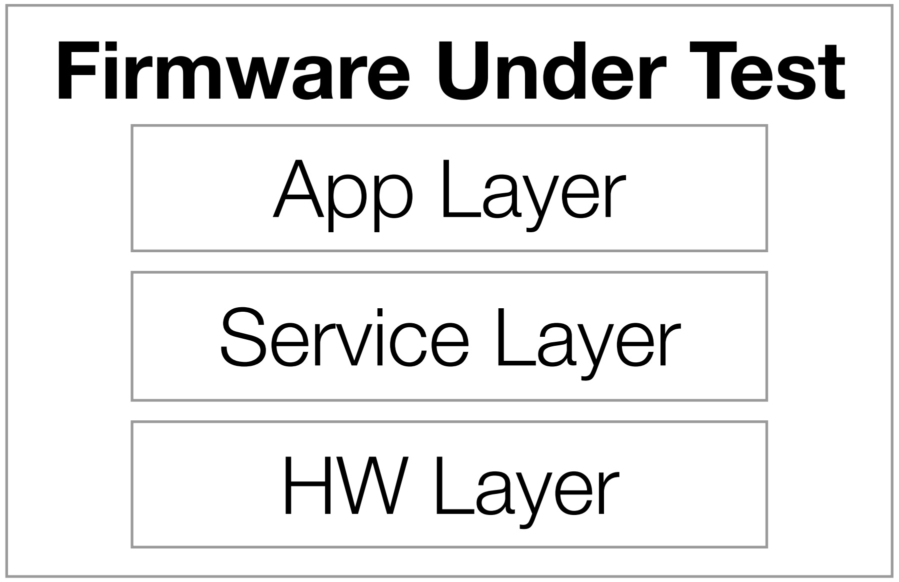
- Many different Frameworks
  - **CppUTest** (C++)
    - Today’s focus, very C and C++ friendly, useful for embedded
  - Unity (C code only)
  - Google test (C++)
  - See more: http://accu.org/index.php/journals/1326

Platform, cross compile, etc

- Test projects
    - Generally run on a PC
        - Drawback: Assembly optimized code.
    - Consider your test environment.
        - Example, 32bit vs 64bit host OS and compiler, etc.
    - Very likely need multiple projects due to C/C++ linker limitations
    - Speed: keep the tests as fast as possible.
    - Behavioral Driven versus “unit test” driven

Example highlevel project/build setup

Example Target Firmware Architecture



Firmware Under Test

App Layer

Service Layer

HW Layer

Example Build or Project Setup

Projects (compiled outputs)

- **App Test Project**
  - Links to target App Layer code
  - Service Layer mock() code
  - App Layer Test code
  - Target: PC test app
- **Service Layer Test Project**
  - Links to Service Layer code
  - HW Layer mock() code
  - Service Layer Test code
  - Target: PC test app
- **Firmware project**
  - Target: cross-compiled for actual target

Mocks!

- A “mock” module provides the same API or interface as real code, enabling the test environment to setup, control, and inspect the mock while testing code that requires the module being mocked.
- In CppUTest a mock’ed function might look like this:

```
bool BatteryIsOk() {
  mock().actualCall(“BatteryIsOk”);
  return (bool) mock().returnIntValueOrDefault((int)true);
}
```

What does a test look like?

```
TEST(CrashDetectionTests, SampleCrashDataInducesCrashEvent) {

  //  <<SNIP>> crash callback setup

  OpenTestFile(“data/crash_accel_data.txt");

  // Inject Data into mock 1 sample at a time
  AccelData_t *sample;
  while ((sample = GetNextSample()) != nullptr) {
    CrashDetectionInjectSample(sample);
    // Speed up test by exiting on Crash Callback
    if (bCallBackCalled) {
      break;
    }
  }

  CloseTestFile();

  //Check to make sure the crash detected Callback was called
  CHECK(bCallBackCalled);
}
```

Another example test showing mock() usage

```
TEST(AppButtonTests, EightSecondButtonPressAndHoldResetsMicrocontroller) {

  // <<SNIP>> remove minor setup code

  //we are about to send the 8 second button hold event.
  //we expect a reset to be requested

  mock().expectOneCall("NVIC_SystemReset");
  mock().ignoreOtherCalls();

  //send the 8 second hold event
  mock_btn_mgr::GetEightSecHoldHandler()(BTNMGR_HOLD_8_SECS);
  PROCESS_APP_QUEUE();

  mock().checkExpectations();
}
```

Lessons Learned

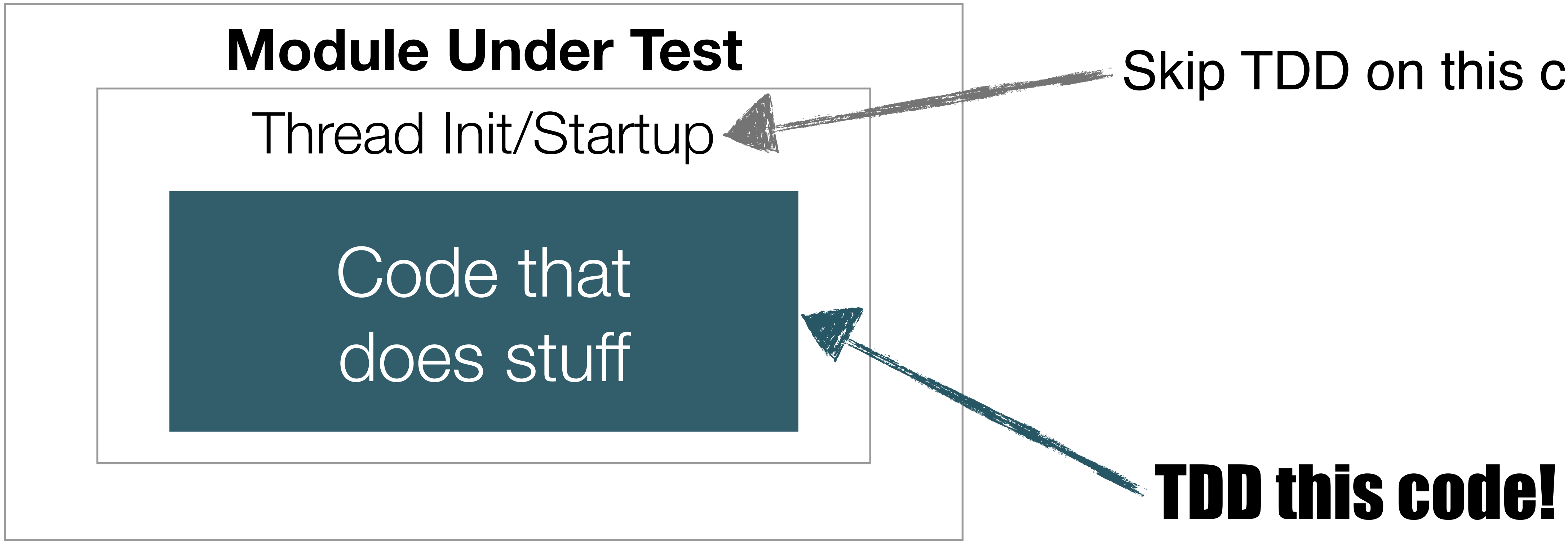
Lessons: Don’t be so strict!

- Don’t be afraid to punch through and reveal “internals” when necessary.
  - Example: Test an internal statemachine that would typically be static/private code.

Lessons Learned: Threads

- Don’t test threads, test the code that runs in a thread.
  - Why?
    - Threading behavior is difficult or impossible to test cross platform
    - Threads are typically all about blocking: blocking on semaphores, queues, timers, etc. Blocking slows down the test projects

Lesson: Threads continued



Module Under Test

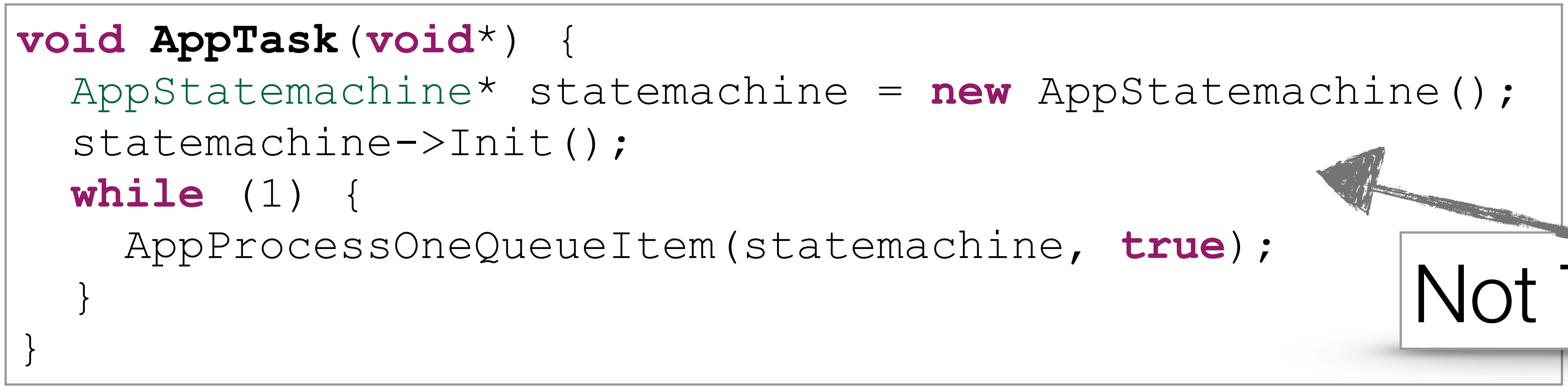
Thread Init/Startup

Code that does stuff

Skip TDD on this code

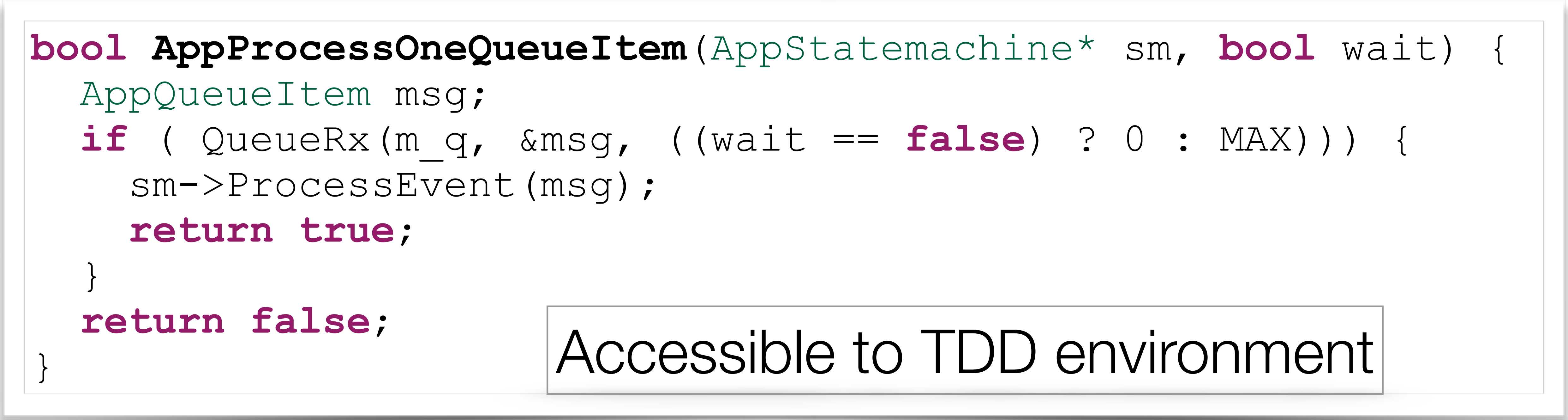
TDD this code!

Example thread + queue + statemachine



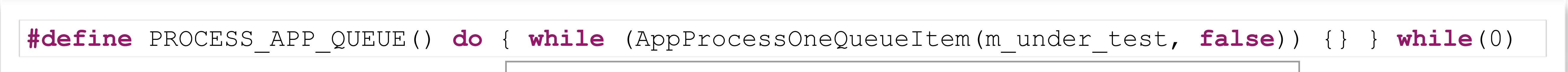
```
void AppTask(void*) {
  AppStatemachine* statemachine = new AppStatemachine();
  statemachine->Init();
  while (1) {
    AppProcessOneQueueItem(statemachine, true);
  }
}
```

Not Tested



```
bool AppProcessOneQueueItem(AppStatemachine* sm, bool wait) {
  AppQueueItem msg;
  if ( QueueRx(m_q, &msg, ((wait == false) ? 0 : MAX))) {
    sm->ProcessEvent(msg);
    return true;
  }
  return false;
}
```

Accessible to TDD environment



```
#define PROCESS_APP_QUEUE() do { while (AppProcessOneQueueItem(m_under_test, false)) {} } while(0)
```

Example Macro in TDD projects

Lessons: Time

- “Time” and embedded often go hand and hand.
    - For Today:
        - Timers: periodic, one-shot, etc
    - Other:
        - Current Time (timestamps, calendars, etc)
        - “Real time” responsiveness

Time: RTOS Timers

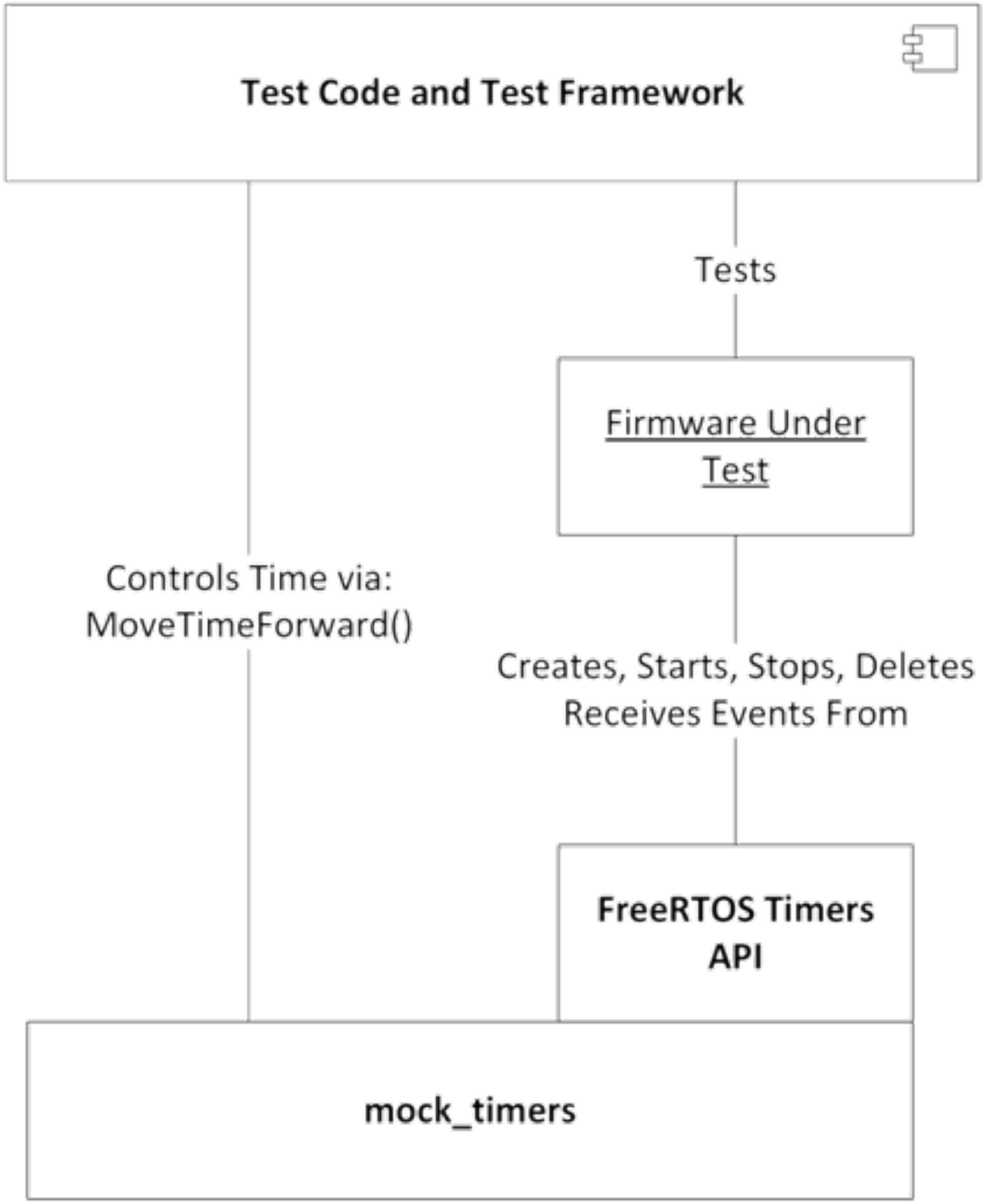
- FreeRTOS timer API:
  - TimerCreate
  - TimerStart
  - TimerStop
  - etc

Mock()’ing Timers

//mock timers external API.
//Used by tests to control time in the timer subsystem

```
#include <chrono>
namespace mock_timers {

  //Init() - prepare internal data structures
  void Init();

  //Destroy() - destroy everything (all mock timers, etc)
  void Destroy();

  //MoveTimeForward() - move time, firing any timers
  //  that would have expired during this timeframe
  void MoveTimeForward(std::chrono::milliseconds ms);

}
```

Mock()'ing Timers continued

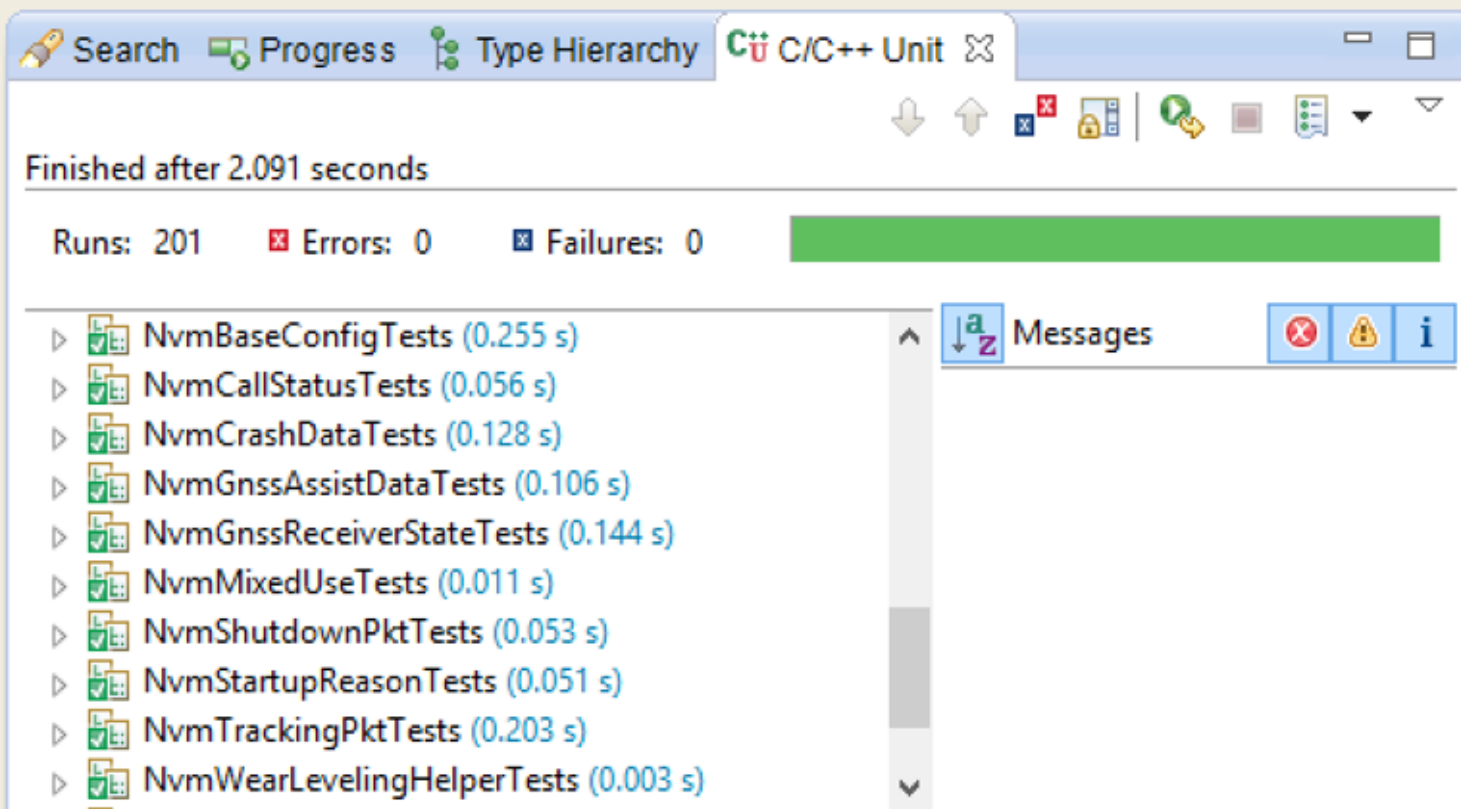

Test Code and Test Framework
Tests
Firmware Under Test
Controls Time via: MoveTimeForward()
Creates, Starts, Stops, Deletes
Receives Events From
FreeRTOS Timers API
mock_timers

Example test showing mock timer usage

```
TEST(AppStartupPktTests,
     StartupPktSentWithoutAckWillRetryAfter10Seconds) {

  // <<SNIP other setup code where Startup Pkt was already sent once>>

  mock().expectOneCall("CellMgrSendStartupPkt");
  mock().ignoreOtherCalls();

  //after 10 seconds, confirm Startup Pkt is sent again
  mock_timers::MoveTimeForward(std::chrono::milliseconds(1000 * 10));
  PROCESS_APP_QUEUE();

  mock().checkExpectations();
}
```

And where does this take us?

Green Lights! Oh the Courage!



Finished after 2.091 seconds

Runs: 201 Errors: 0 Failures: 0

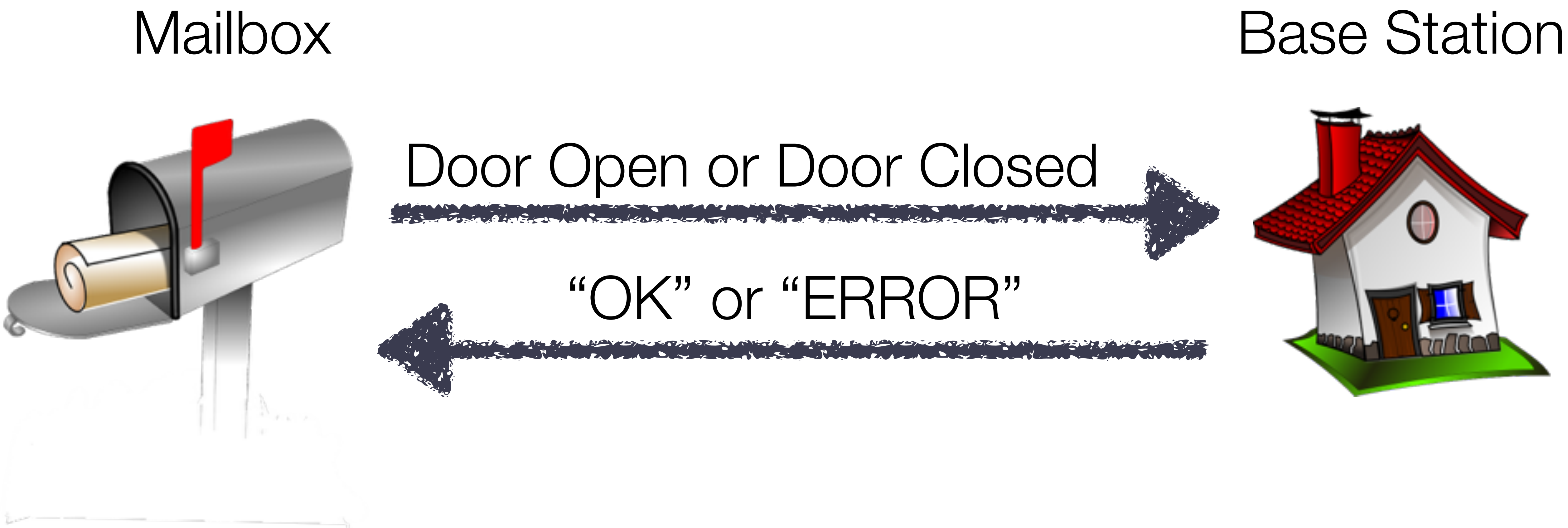
NvmBaseConfigTests (0.255 s)
NvmCallStatusTests (0.056 s)
NvmCrashDataTests (0.128 s)
NvmGnssAssistDataTests (0.106 s)
NvmGnssReceiverStateTests (0.144 s)
NvmMixedUseTests (0.011 s)
NvmShutdownPktTests (0.053 s)
NvmStartupReasonTests (0.051 s)
NvmTrackingPktTests (0.203 s)
NvmWearLevelingHelperTests (0.003 s)

- **332 Tests with 40718 Checks**

How can you start?

- Grenning: “How to start - **Write a test!**”
- Any questions before the online session??

Dojo Exercise



Mailbox

Base Station

Door Open or Door Closed

“OK” or “ERROR”

- http://cyber-dojo.org
- C2482D

Thank you!

Any questions?
matthew@covemountainsoftware.com