Test-Driven Development For Embedded C++ Programmers

#421

By James Grenning and Robert Martin

Object Mentor, Inc.
What is Test Driven Development?

• An iterative technique to develop software
• As much (or more) about design as testing
  – Encourages design from user’s point of view
  – Encourages testing classes in isolation
  – Produces loosely-coupled, highly-cohesive systems
• As much (or more) about documentation as testing
• Must be learned and practiced
  – If it feels natural at first, you’re probably doing it wrong
Typical Development Cycle

Requirements → Design → Code → Test and Fix → Test and Fix → Test and Fix → Test and Fix
Iterative/Evolutionary Development Cycle
Typical development cycle
– Working Features
Iterative/Evolutionary Development Cycle
– Working Features
Automated Tests Provide a Safety Net

- Once a test passes, it is re-run with every change
- Broken tests are not tolerated
- Side affect defects are detected immediately
- Assumptions are continually checked
The Test Driven Development Cycle

Start

Write a test for new capability

Compile

Fix compile errors

Refactor as needed

Run the test
And see it pass

Write the code

Run the test
And see it fail
Lots of Small Steps

• Shortest distance between two points

![Diagram](https://example.com/diagram.png)

• Use test-driven to get from A to B in very small, verifiable steps

• You often end up in a better place
Do the Simplest Thing

• Assume simplicity
  – Consider the simplest thing that could possibly work
  – Iterate to the needed solution

• When coding:
  – Build the simplest possible code that will pass the tests
  – Refactor the code to have the simplest design possible
  – Eliminate duplication
The Rules of Simple Design

IN PRIORITY ORDER!

1. The code passes all tests
2. There is no duplication
3. The code expresses the programmer’s intention
4. Using the smallest number of classes and methods

Higher priority rules must be satisfied first
Automated Tests

• Unit tests
  – Tests that show the programmer that the code does what is expected
  – Specifies what the code must do
  – Provide examples of how to use the code (documentation)
  – All tests are run every few minutes, with every change

• Acceptance tests
  – Tests that show the stake holders that the code delivers the feature
  – All tests are run at least daily

• All tests are Automated, you run them with every change
What is Tested?

- Every class (module) has one or more unit tests
- Test everything that can possibly break

```
int EventLog::GetCapacity()
{
    return capacity;
}
```

- If it can’t break, don’t test it
  - Always a judgment call
Testing Frameworks

• Tests must be automated
  – Otherwise they won’t be run

• Most OO languages have a testing framework, xUnit
  – JUnit, CppUnit(Lite), PyUnit, NUnit, VBUnit
  – A simple tool
  – Collects, organizes and automatically calls your test code

• Graphical test runner
  – Green bar makes you feel good

• Could be added to build environment
Building Test Classes

- All of the testing frameworks work similarly
- Your class inherits from a test framework class, allowing your test to be plugged into the framework
CppUnitLite

- Free C++ unit test harness
- Uses macros to make test definition easy
- Can be used to test C code
- Tests are written that check binary conditions
- Tests are repeatable

- Download it from www.objectmentor.com
EventLog Example - Demo

- Create a class that logs events
- Each event has a character string and an integer
- The log throws out the oldest entry once it exceeds its capacity
- The log can print itself

- See paper for complete example

```
EventLog

  + LogIt(const char*, int)
  + GetCount( ) : int
  + Print( )
```
Focus on Interface

• The test treats the object being tested like black box
• Encourages design to be done from a client point of view
  – The test is a user
• You confront interface design issues
  – What are the parameters?
  – What is the return type?
  – What is the behavior?
  – Who controls object lifetime?
Embedded Test-Driven Developers

• Get code working in a friendlier environment prior to running on the target
  – Feedback
  – Efficient

• Decouple the application logic from the specific hardware dependencies

• Feed events into the system, verify the response
Design Impacts

• Test-first design promotes testing a class in isolation
  – It must be decoupled from other classes
• Produces loosely coupled, highly cohesive systems
  – The hallmark of a good design
  – Object Oriented Design Principles and Programming Languages help
Testing a System of Objects

• Sure, unit tests work fine for a simple class like EventLog. But what about a class that collaborates with other classes?

• Home alarm system example
  – Front panel with LEDs, push buttons, times square display
  – Phone line
  – The hardware won’t be ready for 3 months (one week before delivery)
Collaborators

• Most classes being tested need collaborators
  – e.g., Panel and phone line.
• Sometimes you can test with the real collaborators
• Sometimes you can’t or shouldn’t
  – The hardware is not ready, or it is slow, or hard to control
  – It is difficult to get the response needed from the collaborator
• Impersonate collaborators with a Mock Object
Mock Objects

- Unwanted dependencies can be broken with an interface
Demo

- Email me at grenning@objectmentor.com for the example home guard code, or leave me a card with your request.
Learning Test-First Design

• A skill which must be practiced
  – Initially awkward

• Requires discipline
  – Peer pressure
  – “I know how to write the class, but I don’t know how to test it”

• *It's an addiction rather than discipline*
  – Kent Beck – Author of
    – *Extreme Programming Explained*
    – *Test Driven Development*
Productivity and Predictability

• Defects kill predictability:
  – Cost of fixing is not predictable
  – When they materialize is not predictable

• Test-driven is predictable:
  – Working at a steady pace
  – Results in fewer bugs
  – More productive than “debug-later programming”

• Test-driven programmers rarely need the debugger
Objections Heard

• “I know how to just write the code, but I don’t know how to test it.”
• “We have to write twice as much code”
• “I have to debug twice as much code.”
• “We have a testing department.”
• “I can test my code after I write it.”
• “That might work on easy software but our problem is really tough”
• “You need the target hardware”
How do Acceptance Tests Fit In

Acceptance tests use the application the same way the hardware does, only they bypass the hardware.
Core System Logic Depends on Hardware Specifics
Separate Core System Logic from Hardware Specifics
Test Core System Logic Independent of Hardware Specifics

Tests and Simulations
What’s in it for Embedded Developers

• Decouples embedded application from target hardware
• Progress can be made without target hardware
• Side effect safety net
• Can avoid costly simulators in favor of simpler Mock Objects
• Can avoid many long debus session