Extracted from:

Test-Driven Development for Embedded C

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

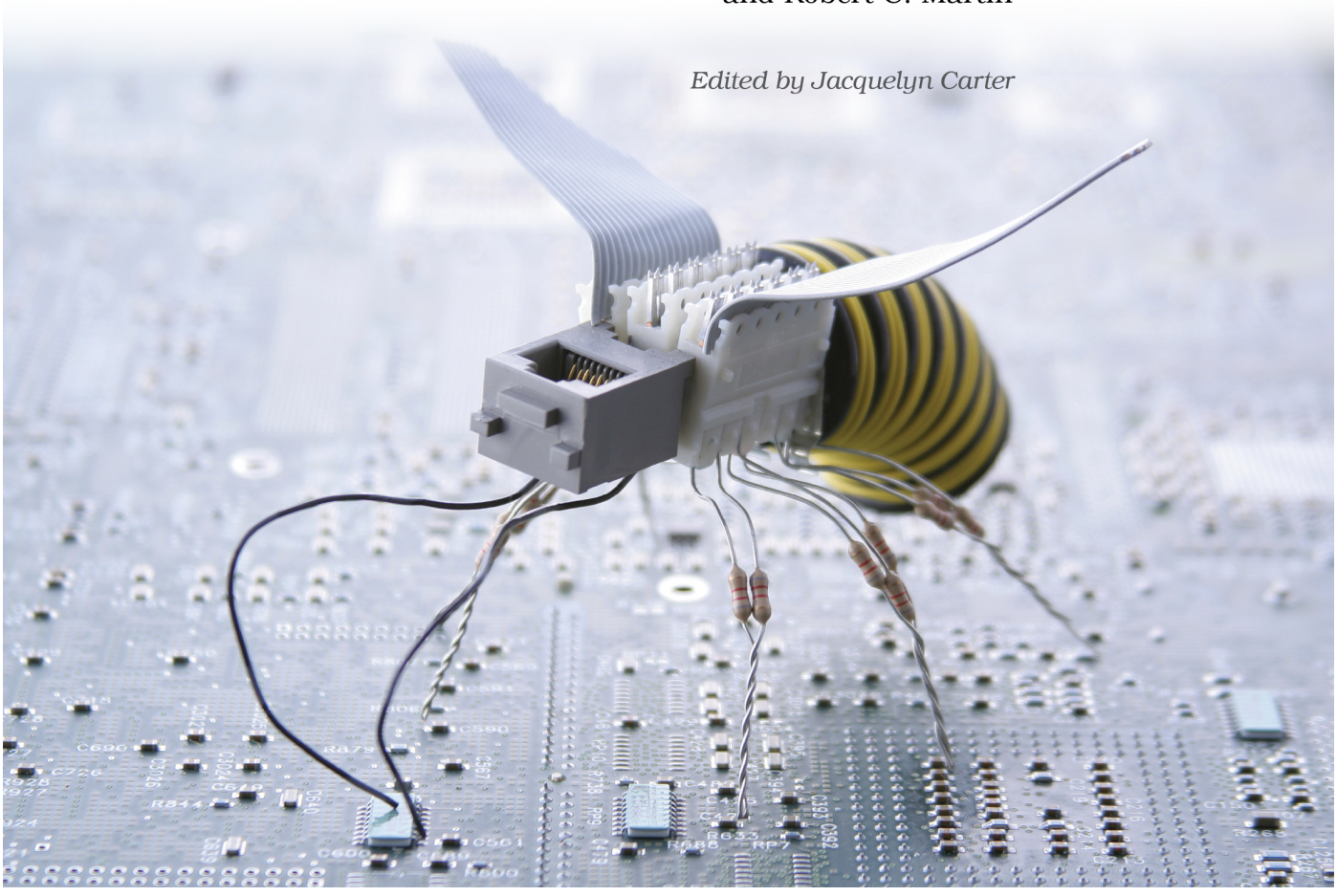
The Pragmatic Programmers

Test-Driven Development for Embedded C

James W. Grenning

Forewords by Jack Ganssle
and Robert C. Martin

Edited by Jacquelyn Carter



Test-Driven Development for Embedded C

James W. Grenning

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina

Contents

Foreword by Jack Ganssle . . . . . ?

Foreword by Robert C. Martin . . . . . ?

Acknowledgments . . . . . ?

Preface . . . . . ?

1. Test-Driven Development . . . . . ?
   - 1.1 Why Do We Need TDD? ?
   - 1.2 What Is Test-Driven Development? ?
   - 1.3 Physics of TDD ?
   - 1.4 The TDD Microcycle ?
   - 1.5 TDD Benefits ?
   - 1.6 Benefits for Embedded ?

Part I — Getting Started

2. Test-Driving Tools and Conventions . . . . . ?
   - 2.1 What Is a Unit Test Harness? ?
   - 2.2 Unity: A C-Only Test Harness ?
   - 2.3 CppUTest: A C++ Unit Test Harness ?
   - 2.4 Unit Tests Can Crash ?
   - 2.5 The Four-Phase Test Pattern ?
   - 2.6 Where Are We? ?

3. Starting a C Module . . . . . ?
   - 3.1 Elements of a Testable C Module ?
   - 3.2 What Does an LED Driver Do? ?
   - 3.3 Write a Test List ?
   - 3.4 Writing the First Test ?
   - 3.5 Test-Drive the Interface Before the Internals ?

3.6 Incremental Progress ?
3.7 Test-Driven Developer State Machine ?
3.8 Tests Are FIRST ?
3.9 Where Are We? ?

4. Testing Your Way to Done . . . . . ?
4.1 Grow the Solution from Simple Beginnings ?
4.2 Keep the Code Clean—Refactor as You Go ?
4.3 Repeat Until Done ?
4.4 Take a Step Back Before Claiming Done ?
4.5 Where Are We? ?

5. Embedded TDD Strategy . . . . . ?
5.1 The Target Hardware Bottleneck ?
5.2 Benefits of Dual-Targeting ?
5.3 Risks of Dual-Target Testing ?
5.4 The Embedded TDD Cycle ?
5.5 Dual-Target Incompatibilities ?
5.6 Testing with Hardware ?
5.7 Slow Down to Go Fast ?
5.8 Where Are We? ?

6. Yeah, but... . . . . ?
6.1 We Don't Have Time ?
6.2 Why Not Write Tests After the Code? ?
6.3 We'll Have to Maintain the Tests ?
6.4 Unit Tests Don't Find All the Bugs ?
6.5 We Have a Long Build Time ?
6.6 We Have Existing Code ?
6.7 We Have Constrained Memory ?
6.8 We Have to Interact with Hardware ?
6.9 Why a C++ Test Harness for Testing C? ?
6.10 Where Are We? ?

Part II — Testing Modules with Collaborators

7. Introducing Test Doubles . . . . . ?
7.1 Collaborators ?
7.2 Breaking Dependencies ?
7.3 When to Use a Test Double ?

7.4 Faking It in C, What's Next ?
7.5 Where Are We? ?

8. **Spying on the Production Code** . . . . . ?
8.1 Light Scheduler Test List ?
8.2 Dependencies on Hardware and OS ?
8.3 Link-Time Substitution ?
8.4 Spying on the Code Under Test ?
8.5 Controlling the Clock ?
8.6 Make It Work for None, Then One ?
8.7 Make It Work for Many ?
8.8 Where Are We? ?

9. **Runtime-Bound Test Doubles** . . . . . ?
9.1 Testing Randomness ?
9.2 Faking with a Function Pointer ?
9.3 Surgically Inserted Spy ?
9.4 Verifying Output with a Spy ?
9.5 Where Are We? ?

10. **The Mock Object** . . . . . ?
10.1 Flash Driver ?
10.2 MockIO ?
10.3 Test-Driving the Driver ?
10.4 Simulating a Device Timeout ?
10.5 Is It Worth It? ?
10.6 Mocking with CppUMock ?
10.7 Generating Mocks ?
10.8 Where Are We? ?

Part III — Design and Continuous Improvement

11. **SOLID, Flexible, and Testable Designs** . . . . . ?
11.1 SOLID Design Principles ?
11.2 SOLID C Design Models ?
11.3 Evolving Requirements and a Problem Design ?
11.4 Improving the Design with Dynamic Interface ?
11.5 More Flexibility with Per-Type Dynamic Interface ?
11.6 How Much Design Is Enough? ?
11.7 Where Are We? ?

12. Refactoring . . . . . ?
12.1 Two Values of Software ?
12.2 Three Critical Skills ?
12.3 Code Smells and How to Improve Them ?
12.4 Transforming the Code ?
12.5 But What About Performance and Size? ?
12.6 Where Are We? ?

13. Adding Tests to Legacy Code . . . . . ?
13.1 Legacy Code Change Policy ?
13.2 Boy Scout Principle ?
13.3 Legacy Change Algorithm ?
13.4 Test Points ?
13.5 Two-Stage struct Initialization ?
13.6 Crash to Pass ?
13.7 Characterization Tests ?
13.8 Learning Tests for Third-Party Code ?
13.9 Test-Driven Bug Fixes ?
13.10 Add Strategic Tests ?
13.11 Where Are We? ?

14. Test Patterns and Antipatterns . . . . . ?
14.1 Ramble-on Test Antipattern ?
14.2 Copy-Paste-Tweak-Repeat Antipattern ?
14.3 Sore Thumb Test Cases Antipattern ?
14.4 Duplication Between Test Groups Antipattern ?
14.5 Test Disrespect Antipattern ?
14.6 Behavior-Driven Development Test Pattern ?
14.7 Where Are We? ?

15. Closing Thoughts . . . . . ?

Part IV — Appendixes

A1. Development System Test Environment . . . . . ?
A1.1 Development System Tool Chain ?
A1.2 Full Test Build makefile ?
A1.3 Smaller Test Builds ?

A2. Unity Quick Reference . . . . . ?
A2.1 Unity Test File ?
A2.2 Unity Test main ?
A2.3 Unity TEST Condition Checks ?
A2.4 Command-Line Options ?
A2.5 Unity in Your Target ?

A3. CppUTest Quick Reference . . . . . ?
A3.1 The CppUTest Test File ?
A3.2 Test Main ?
A3.3 TEST Condition Checks ?
A3.4 Test Execution Order ?
A3.5 Scripts to Create Starter Files ?
A3.6 CppUTest in Your Target ?
A3.7 Convert CppUTest Tests to Unity ?

A4. LedDriver After Getting Started . . . . . ?
A4.1 LedDriver First Few Tests in Unity ?
A4.2 LedDriver First Few Tests in CppUTest ?
A4.3 LedDriver Early Interface ?
A4.4 LedDriver Skeletal Implementation ?

A5. Example OS Isolation Layer . . . . . ?
A5.1 Test Cases to Assure Substitutable Behavior ?
A5.2 POSIX Implementation ?
A5.3 Micrium RTOS Implementation ?
A5.4 Win32 Implementation ?
A5.5 Burden the Layer, Not the Application ?

A6. Bibliography . . . . . ?

Index . . . . . ?