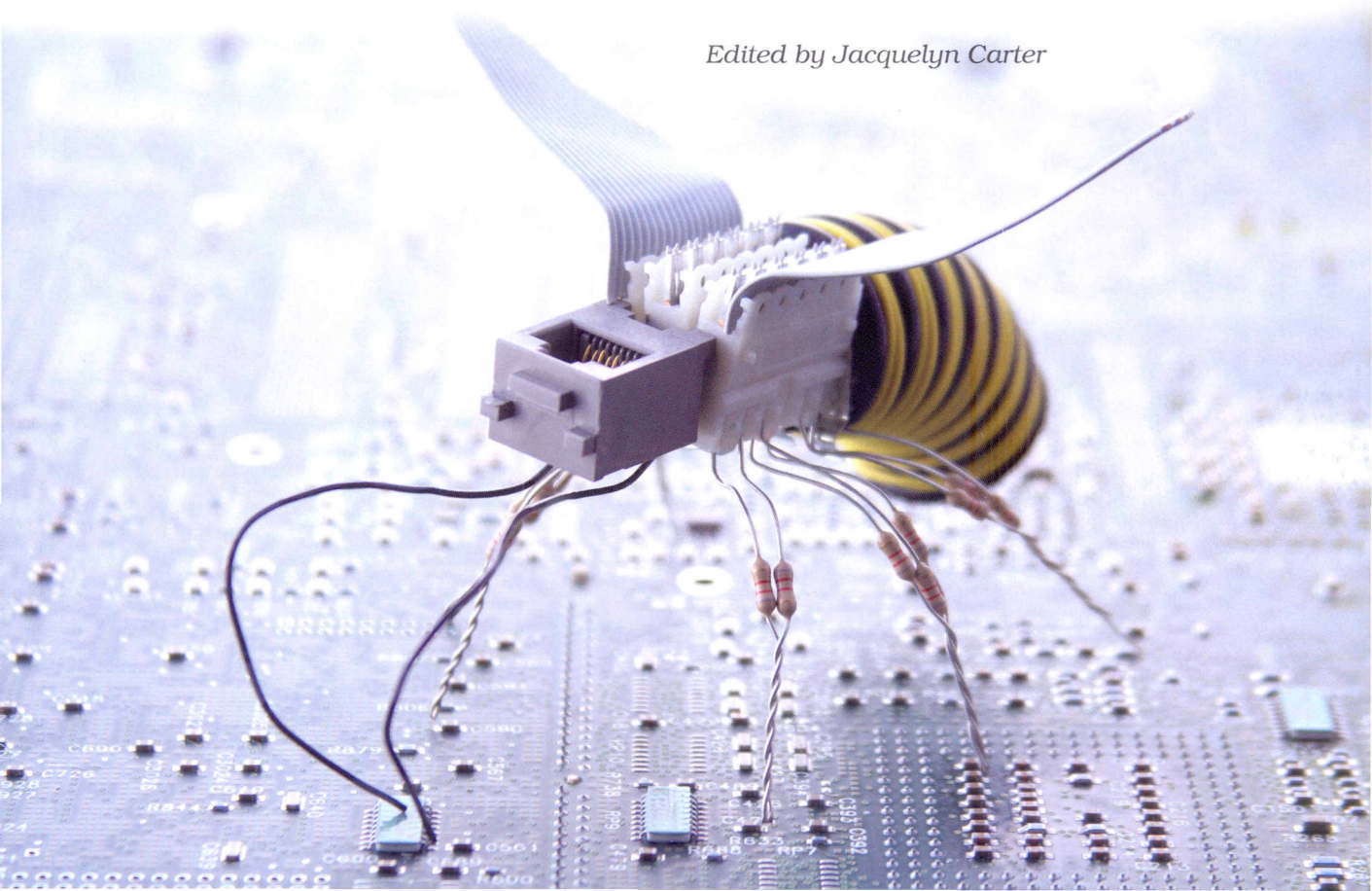


Test-Driven Development for Embedded C

James W. Grenning

Forewords by Jack Ganssle
and Robert C. Martin

Edited by Jacquelyn Carter



Contents

Foreword by Jack Ganssle	xiii
Foreword by Robert C. Martin	xv
Acknowledgments	xix
Preface	xxi
Who Is This Book For?	xxii
How to Read This Book	xxii
The Code in This Book	xxiii
Online Resources	xxiv
1 Test-Driven Development	1
1.1 Why Do We Need TDD?	2
1.2 What Is Test-Driven Development?	4
1.3 Physics of TDD	5
1.4 The TDD Microcycle	6
1.5 TDD Benefits	9
1.6 Benefits for Embedded	10
I Getting Started	13
2 Test-Driving Tools and Conventions	15
2.1 What Is a Unit Test Harness?	15
2.2 Unity: A C-Only Test Harness	17
2.3 CppUTest: A C++ Unit Test Harness	23
2.4 Unit Tests Can Crash	27
2.5 The Four-Phase Test Pattern	28
2.6 Where Are We?	28

3	Starting a C Module	31
3.1	Elements of a Testable C Module	31
3.2	What Does an LED Driver Do?	33
3.3	Write a Test List	34
3.4	Writing the First Test	35
3.5	Test-Drive the Interface Before the Internals	41
3.6	Incremental Progress	48
3.7	Test-Driven Developer State Machine	50
3.8	Tests Are FIRST	52
3.9	Where Are We?	52
4	Testing Your Way to Done	55
4.1	Grow the Solution from Simple Beginnings	55
4.2	Keep the Code Clean—Refactor as You Go	71
4.3	Repeat Until Done	74
4.4	Take a Step Back Before Claiming <i>Done</i>	81
4.5	Where Are We?	81
5	Embedded TDD Strategy	85
5.1	The Target Hardware Bottleneck	85
5.2	Benefits of Dual-Targeting	87
5.3	Risks of Dual-Target Testing	88
5.4	The Embedded TDD Cycle	89
5.5	Dual-Target Incompatibilities	92
5.6	Testing with Hardware	97
5.7	Slow Down to Go Fast	101
5.8	Where Are We?	101
6	Yeah, but...	103
6.1	We Don't Have Time	103
6.2	Why Not Write Tests After the Code?	107
6.3	We'll Have to Maintain the Tests	108
6.4	Unit Tests Don't Find All the Bugs	108
6.5	We Have a Long Build Time	109
6.6	We Have Existing Code	109
6.7	We Have Constrained Memory	110
6.8	We Have to Interact with Hardware	111
6.9	Why a C++ Test Harness for Testing C?	112
6.10	Where Are We?	113

II	Testing Modules with Collaborators	115
7	Introducing Test Doubles	117
7.1	Collaborators	117
7.2	Breaking Dependencies	118
7.3	When to Use a Test Double	122
7.4	Faking It in C, What's Next	123
7.5	Where Are We?	127
8	Spying on the Production Code	129
8.1	Light Scheduler Test List	131
8.2	Dependencies on Hardware and OS	131
8.3	Link-Time Substitution	132
8.4	Spying on the Code Under Test	133
8.5	Controlling the Clock	138
8.6	Make It Work for None, Then One	139
8.7	Make It Work for Many	154
8.8	Where Are We?	159
9	Runtime-Bound Test Doubles	161
9.1	Testing Randomness	161
9.2	Faking with a Function Pointer	163
9.3	Surgically Inserted Spy	166
9.4	Verifying Output with a Spy	170
9.5	Where Are We?	175
10	The Mock Object	177
10.1	Flash Driver	178
10.2	MockIO	186
10.3	Test-Driving the Driver	189
10.4	Simulating a Device Timeout	192
10.5	Is It Worth It?	195
10.6	Mocking with CppUMock	196
10.7	Generating Mocks	198
10.8	Where Are We?	200

III	Design and Continuous Improvement	203
11	SOLID, Flexible, and Testable Designs	205
11.1	SOLID Design Principles	206
11.2	SOLID C Design Models	209
11.3	Evolving Requirements and a Problem Design . . .	212
11.4	Improving the Design with Dynamic Interface . . .	219
11.5	More Flexibility with Per-Type Dynamic Interface .	228
11.6	How Much Design Is Enough?	232
11.7	Where Are We?	233
12	Refactoring	235
12.1	Two Values of Software	235
12.2	Three Critical Skills	236
12.3	Code Smells and How to Improve Them	238
12.4	Transforming the Code	249
12.5	But What About Performance and Size?	267
12.6	Where Are We?	270
13	Adding Tests to Legacy Code	271
13.1	Legacy Code Change Policy	272
13.2	Boy Scout Principle	272
13.3	Legacy Change Algorithm	273
13.4	Test Points	275
13.5	Two-Stage struct Initialization	278
13.6	Crash to Pass	282
13.7	Characterization Tests	287
13.8	Learning Tests for Third-Party Code	291
13.9	Test-Driven Bug Fixes	293
13.10	Add Strategic Tests	294
13.11	Where Are We?	294
14	Test Patterns and Antipatterns	297
14.1	Ramble-on Test Antipattern	297
14.2	Copy-Paste-Tweak-Repeat Antipattern	299
14.3	Sore Thumb Test Cases Antipattern	300
14.4	Duplication Between Test Groups Antipattern . . .	302
14.5	Test Disrespect Antipattern	303
14.6	Behavior-Driven Development Test Pattern	303
14.7	Where Are We?	304
15	Closing Thoughts	305

IV	Appendixes	309
A	Development System Test Environment	311
A.1	Development System Tool Chain	311
A.2	Full Test Build makefile	313
A.3	Smaller Test Builds	314
B	Unity Quick Reference	317
B.1	Unity Test File	317
B.2	Unity Test main	319
B.3	Unity TEST Condition Checks	319
B.4	Command-Line Options	320
B.5	Unity in Your Target	320
C	CppUTest Quick Reference	323
C.1	The CppUTest Test File	323
C.2	Test Main	324
C.3	TEST Condition Checks	324
C.4	Test Execution Order	325
C.5	Scripts to Create Starter Files	325
C.6	CppUTest in Your Target	327
C.7	Convert CppUTest Tests to Unity	327
D	LedDriver After Getting Started	329
D.1	LedDriver First Few Tests in Unity	329
D.2	LedDriver First Few Tests in CppUTest	330
D.3	LedDriver Early Interface	331
D.4	LedDriver Skeletal Implementation	331
E	Example OS Isolation Layer	333
E.1	Test Cases to Assure Substitutable Behavior	334
E.2	POSIX Implementation	335
E.3	Micrium RTOS Implementation	337
E.4	Win32 Implementation	339
E.5	Burden the Layer, Not the Application	341
F	Bibliography	343
	Index	347