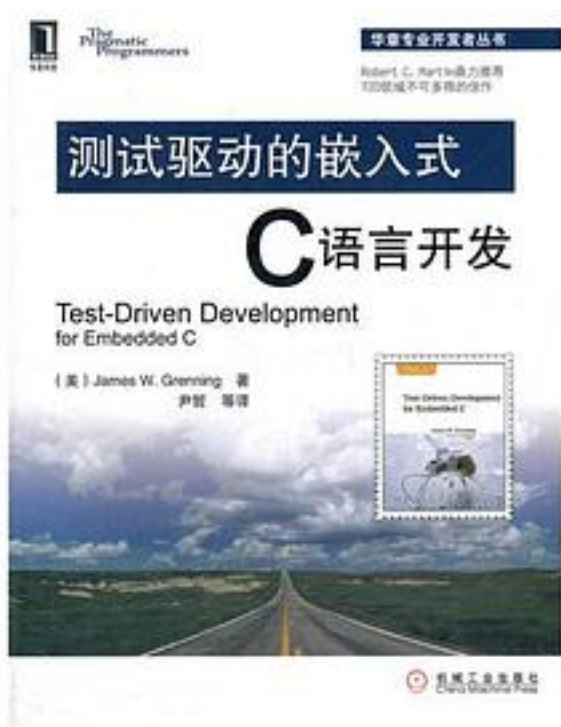


# 测试驱动的嵌入式C语言开发



[测试驱动的嵌入式C语言开发 下载链接1](#)

著者:James W. Grenning

出版者:机械工业出版社

出版时间:2012-1

装帧:平装

isbn:9787111366232

《测试驱动的嵌入式C语言开发》深入介绍如何把测试驱动的开发方法应用于嵌入式C语言开发，第一部分介绍了两个开源的测试框架，通过测试驱动开发方法开发第一个模块；第二部分深入介绍了与系统中其他模块进行交互的代码的测试技术，如测试替身、仿制对象等；第三部分介绍了设计与持续改进代码，如写出更好代码的一些重要原则，建立可测并灵活设计的高级技术，改进已有代码的实践方法——重构技术，改进遗留代码，以及编写和维护测试的指导原则。《测试驱动的嵌入式C语言开发》的代码几乎全部用C写成，并且可以用于嵌入式的、受约束的开发和执行环境。

《测试驱动的嵌入式C语言开发》是作者多年实践经验的总结，实用性强，适合嵌入式

C/C++语言程序员、工程师阅读。

作者介绍:

目录: 对本书的赞誉

译者序

推荐序一

推荐序二

前言

致谢

第1章 测试驱动开发 1

1.1 为什么我们需要TDD 2

1.2 什么是测试驱动开发 3

1.3 TDD的机理 4

1.4 TDD的微循环 5

1.5 TDD的好处 7

1.6 对于嵌入式开发的益处 7

第一部分 开始

第2章 测试驱动开发的工具和约定 9

2.1 什么是自动化单元测试框架 9

2.2 Unity: 一个全部用C实现的自动化测试框架 10

2.3 CppUTest: 一个用C++实现的自动化单元测试框架 16

2.4 单元测试也会崩溃 19

2.5 “四阶段”模式 19

2.6 我们到哪里了 19

第3章 开始一个C语言模块 21

3.1 具有可测性的C模块的那些元素 21

3.2 LED驱动都做些什么 22

3.3 写一个测试列表 23

3.4 写第一个测试 23

3.5 先测试驱动接口再测试驱动内部实现 29

3.6 增量式前进 34

3.7 测试驱动开发者的状态机 36

3.8 测试要做到FIRST 37

3.9 我们到哪里了 37

第4章 一路测试直到完成 39

4.1 从简单入手“生长”出解决方案 39

4.2 保持代码整洁——边做边重构 53

4.3 重复直到完成 55

4.4 声明完成之前先向回走一步 61

4.5 我们到哪里了 61

第5章 嵌入式系统TDD策略 63

5.1 目标硬件的瓶颈 63

5.2 双目标开发的好处 64

5.3 双目标测试的风险 65

5.4 嵌入式的TDD循环 65

5.5 双目标的不兼容性 67

5.6 和硬件一起测试 71

5.7 欲速则不达 74

5.8 我们到哪里了 74

第6章 是的，但是…… 75

6.1 我们没那个时间 75

- 6.2 为什么不在写了代码之后再写测试 77
- 6.3 测试也需要维护 78
- 6.4 单元测试不能发现所有的bug 78
- 6.5 我们的构建时间太长 79
- 6.6 我们有现存的代码 79
- 6.7 我们的内存有约束 79
- 6.8 我们不得不和硬件交互 80
- 6.9 为什么要用C++的测试框架来测试C 81
- 6.10 我们到哪里了 81
- 第二部分 测试有合作者的模块
- 第7章 测试替身 83
  - 7.1 合作者 83
  - 7.2 脱离依赖关系 84
  - 7.3 何时使用测试替身 86
  - 7.4 用C来仿冒，下一步 88
  - 7.5 我们到哪里了 89
- 第8章 监视产品代码 90
  - 8.1 灯光调度测试列表 90
  - 8.2 对于硬件和操作系统的依赖 91
  - 8.3 链接时代换 92
  - 8.4 监视被测试代码 93
  - 8.5 控制时钟 97
  - 8.6 先0后 198
  - 8.7 处理多个的情况 110
  - 8.8 我们到哪里了 115
- 第9章 运行时绑定的测试替身 116
  - 9.1 测试随机性 116
  - 9.2 冒仿函数指针 118
  - 9.3 外科手术般地插入间谍 120
  - 9.4 用间谍来校验输出 124
  - 9.5 我们到哪里了 127
- 第10章 仿制对象 129
  - 10.1 闪存驱动程序 129
  - 10.2 MockIO 136
  - 10.3 测试驱动开发驱动程序 138
  - 10.4 模拟设备超时 142
  - 10.5 这值得吗 144
  - 10.6 用CppUMock来仿制 144
  - 10.7 生成仿制对象 147
  - 10.8 我们到哪里了 148
- 第三部分 设计与持续改进
- 第11章 SOLID、灵活并可测试的设计 149
  - 11.1 SOLID设计原则 150
  - 11.2 C语言中的SOLID模型 152
  - 11.3 演进的需求和有问题的设计 154
  - 11.4 用动态接口来改进设计 160
  - 11.5 更灵活的基于类型的动态接口 168
  - 11.6 做多少设计才是足够的 171
  - 11.7 我们到哪里了 173
- 第12章 重构 174
  - 12.1 软件的两个价值 174
  - 12.2 三项关键技能 175
  - 12.3 代码中的坏味道以及如何改进它们 176
  - 12.4 转化代码 184

12.5 那性能和大小怎么办 199  
12.6 我们到哪里了 201  
第13章 为遗留代码加测试 203  
13.1 遗留代码改动准则 203  
13.2 童子军原则 204  
13.3 遗留代码改动步骤 205  
13.4 测试点 206  
13.5 两步结构体初始化 209  
13.6 崩溃直到通过 211  
13.7 鉴别测试 216  
13.8 为第三方代码做学习测试 219  
13.9 测试驱动缺陷修正 220  
13.10 增加策略测试 221  
13.11 我们到哪里了 221  
第14章 测试的模式与反模式 223  
14.1 “喋喋不休”测试反模式 223  
14.2 “拷贝一粘贴一调整一重复”反模式 224  
14.3 “格格不入的测试用例”反模式 225  
14.4 “测试组之间的重复”反模式 227  
14.5 “不尊重测试”反模式 228  
14.6 “行为驱动开发”测试模式 228  
14.7 我们到哪里了 229  
第15章 结束语 230  
第四部分 附录  
附录A 开发系统的测试环境 233  
附录B Unity快速索引 237  
附录C CppUTest快速索引 241  
附录D 开始之后的LedDriver 245  
附录E 操作系统隔离层的例子 248  
附录F 参考书目 255  
• • • • • ([收起](#))

[测试驱动的嵌入式C语言开发\\_下载链接1](#)

标签

测试驱动

嵌入式

TDD

C

测试

C/C++

软件工程

软件开发

## 评论

里面这个项目和cmockery挺像的.这本书比较简单

-----  
评：这本书对于自己的帮助不大，不过也是有帮助的。起码告诉我：  
1.cppUnit这个测试框架； 2.对于依赖目标系统的嵌入式软件，仿造系统调用和硬件行为是应该的； 3.做到完美的测试是不可能的，但是做总比不做要强，毕竟经过测试的代码才能让人感觉踏实。

-----  
用了2个小时读完了，里面讲的概念太熟悉了，代码都不需要看，就当复习了。

-----  
补一下软件开发的“常识”，并立刻付诸实践。  
TDD（注重前期单元测试）与DLP（依赖后期调试），C单元测试框架介绍，重构与持续改进，双目标开发的好处，Mock，SOLID原则，让这些支持起你的代码质量。

-----  
后面几章不是那么贴合主题，类似代码大全里写的软件架构思维

-----  
TDD介绍的深入浅出，而且关于如何把S.O.L.I.D的软件设计理念灵活运用到嵌入式系统开发中，第十一章做了一个很好的展示，中英文看了2遍，不错，推荐。

-----

本来想给5星，但不少印刷错误不能忍……对于C程序猿，该书是难得一见的手把手教你TDD的好书。其实重要的还是思想，不要浮躁，不要心急，一步一步前进。敏捷没有传说中那么可怕。

可测试性强的代码必定拥有者良好的架构和层次，保持你的代码clean，随时注意代码中的坏味道。想办法构建你的模拟测试环境，你会发现一切都很和谐。

-----  
目前见到的唯一一本关于测试驱动的嵌入式C语言开发的书籍。有空还要再好好读读。

-----  
准备用单元测试驱动开发来填补从high level design到code之间的环节。很实用的step by step教程，可惜绝版了

-----  
期望永远是好的，太形式主义。没懂花这么大功夫讲它爪子。

-----  
对翻译起码有个要求~译者至少读过以后用自己的语言重新转述一遍而不是简单的对字翻译~很多英文的长句子读起来很难受~另外像里氏替换翻成列丝科芙这样的问题机械工业出版社的编辑没有能发现的吗？

-----  
这本书的原始源代码，在新版gcc/clang编译器上无法正常编译，会报很多错误。我把修改过的源代码上传到了github：<https://github.com/dleecn/tddcode>

-----  
[测试驱动的嵌入式C语言开发\\_下载链接1](#)

## 书评

### 部分1 第一章

基本思想,增量式开发，先根据接口写测试代码（结对测试?一个人写测试代码，或者都是自己写），自动化执行(使得可复用),并遵循“微循环”的模式 第二章 工具介绍

run\_test\_case = test\_setup+test+test\_teardown <运行make -i -f MakefileUnity.mk...

