侃大山之单元测试

- 什么是单元测试
- 测试替身
- 单元测试的价值

“流水不腐，户枢不蠹”

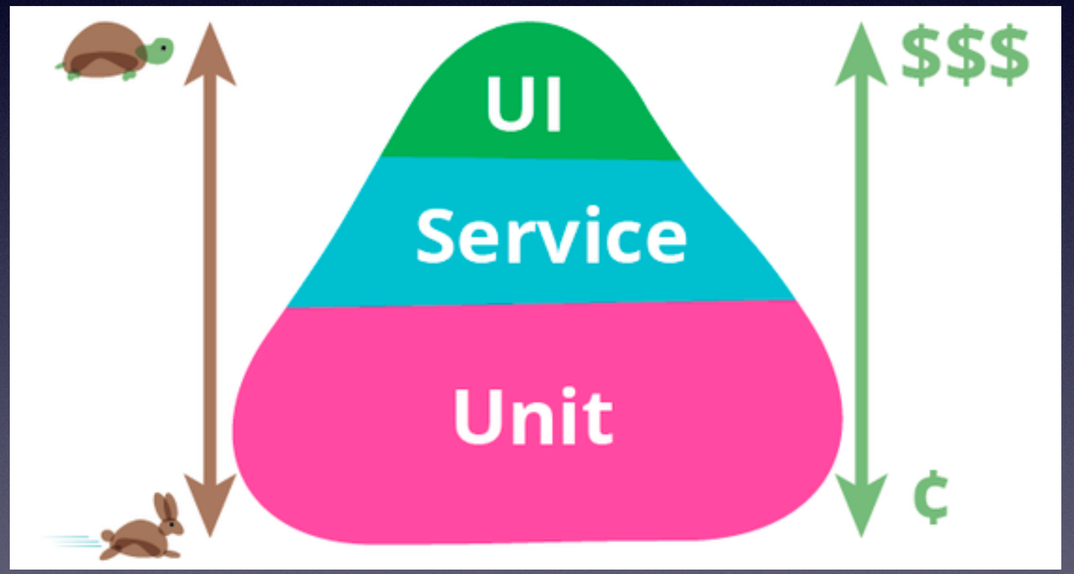
–《吕氏春秋·尽数》

起源

- 兴起于敏捷社区、极限编程
- 1997年，设计模式四巨头之一Erich Gamma和极限编程发明人Kent Beck共同开发了JUnit
- 随着敏捷大潮的流行，单元测试成了现代软件开发过程中必不可少的工具之一，越来越多的程序员推崇自动化测试的理念，作为经济合理的回归测试手段

单元测试 — what?

- 是软件测试中对软件的单个单元/组件进行测试的一个层级，目的是为了验证软件的每个单元的功能都按照设计意图运行
- 一个单元由软件中最小的可测试部分所组成，可以是一个指令、一个函数或者一段程序等。在面向对象编程中，单测的最小单位是一个方法。

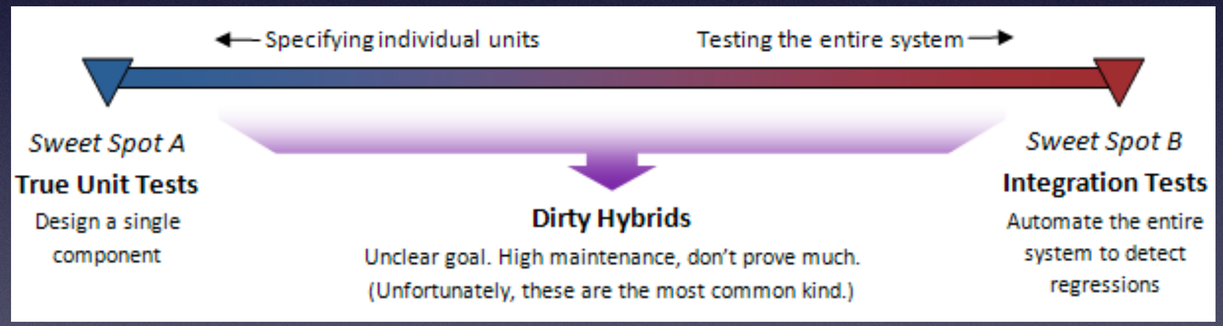


单元测试 — when?

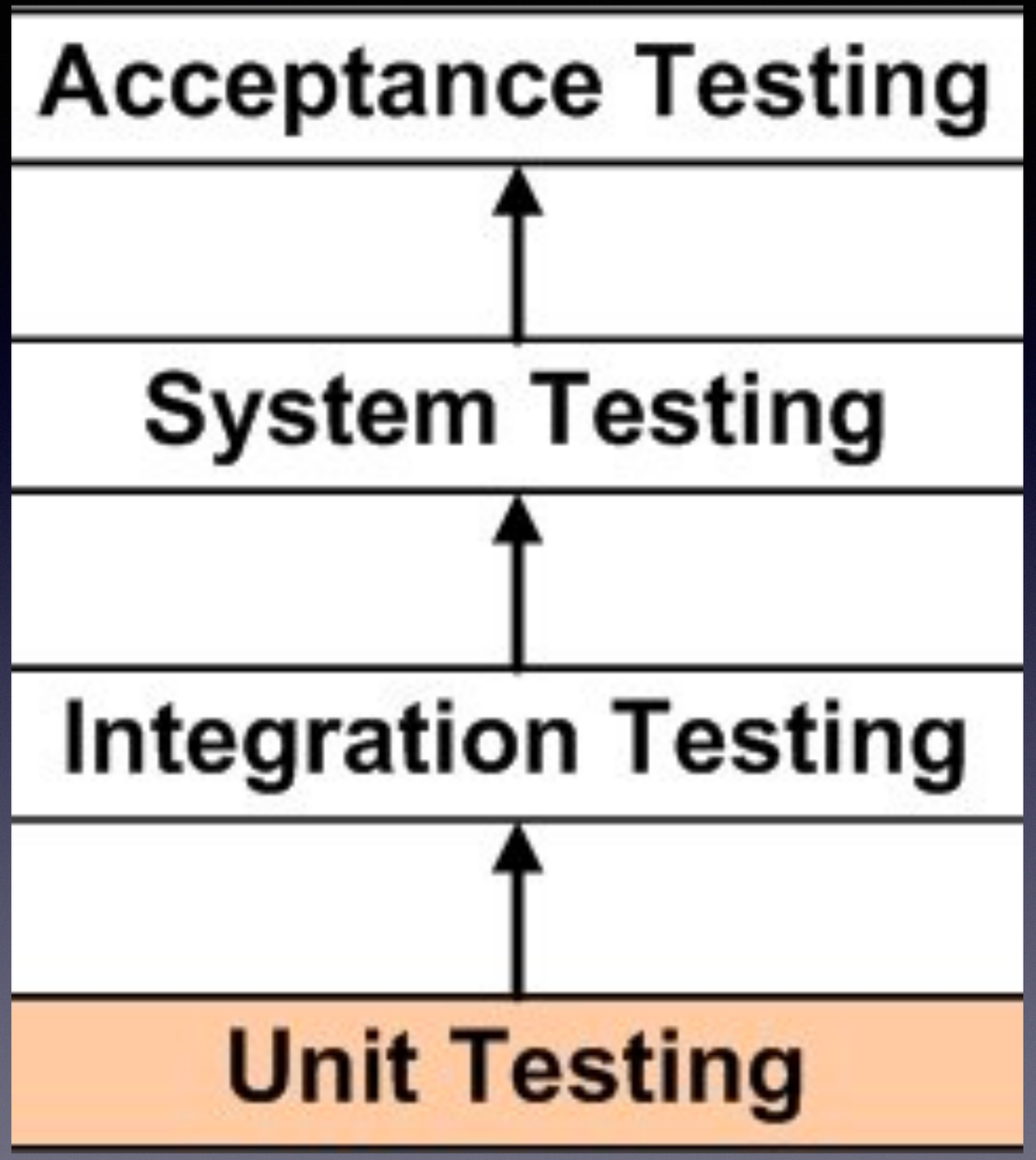
- 具体代码实现之前(TDD)
- 与功能代码同步进行
- 编写完功能代码之后

特点

- 快速
- 隔离
- 简洁
- 易于理解
- 可靠



单元测试 vs 集成测试 vs 系统测试 vs 验收测试



	who	how
单元测试	开发人员，也可以是软件测试人员	白盒测试
集成测试	开发人员或者测试人员	黑盒、白盒、灰盒都可以
系统测试	通常为测试人员	通常为黑盒测试
验收测试	内部验收通常为：产品经理、销售、客服等；外部：用户、消费者	通常为黑盒测试，通常不遵循严格的过程，也不是脚本化的，而是临时的

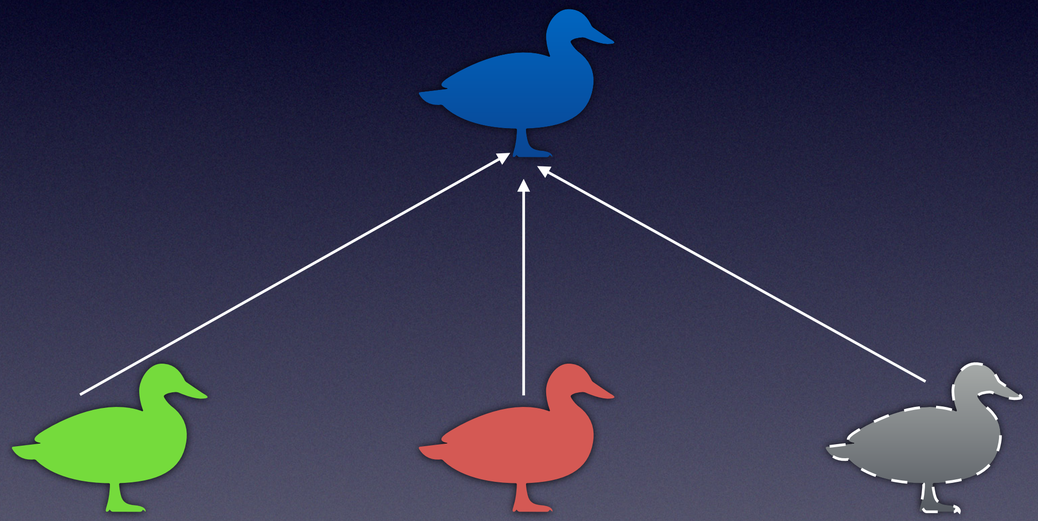
测试替身

“改变世界从自身做起”

–甘地

测试替身

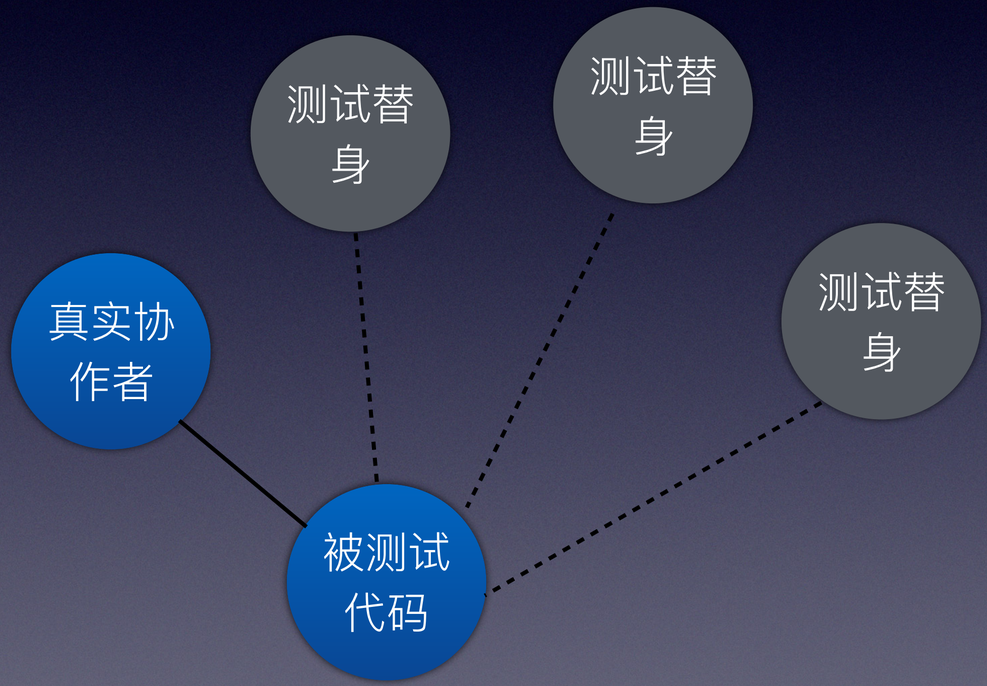
- dummy、stub、fake、mock、spy的总称
- 本质上是为了测试目的，用于替换真实协作者的对象



鸭子的测试替身，看起来像鸭子，叫起来几乎也像一只鸭子—但当然不是真鸭子

测试替身的威力

- 测试替身帮助我们隔离被测试代码，这样我们就能测试其行为的各个方面
  - 隔离被测试代码
  - 加速执行测试
  - 使执行变得确定
  - 模拟特殊情况
  - 访问隐藏信息



测试替身的类型

- 测试桩通常是短小的
- 伪造对象做事不产生副作用
- 测试间谍窃取秘密
- 模拟对象反对惊喜

使用测试替身的指南

- 为测试挑选合适的替身
- 准备、执行、断言
- 检查行为，而非实现
- 挑选你的工具

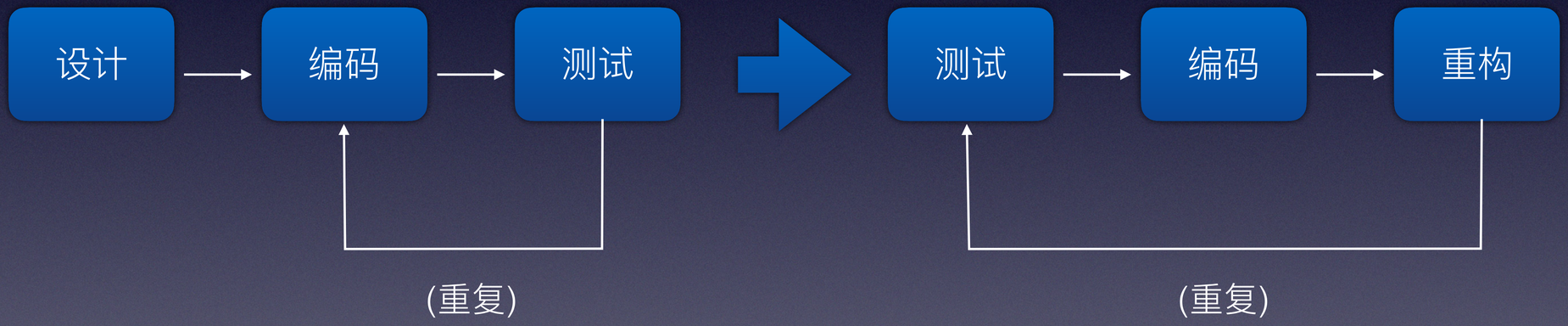
测试的价值

“工欲善其事，必先利其器”

–《论语·卫灵公》

测试作为设计工具

- TDD
- BDD

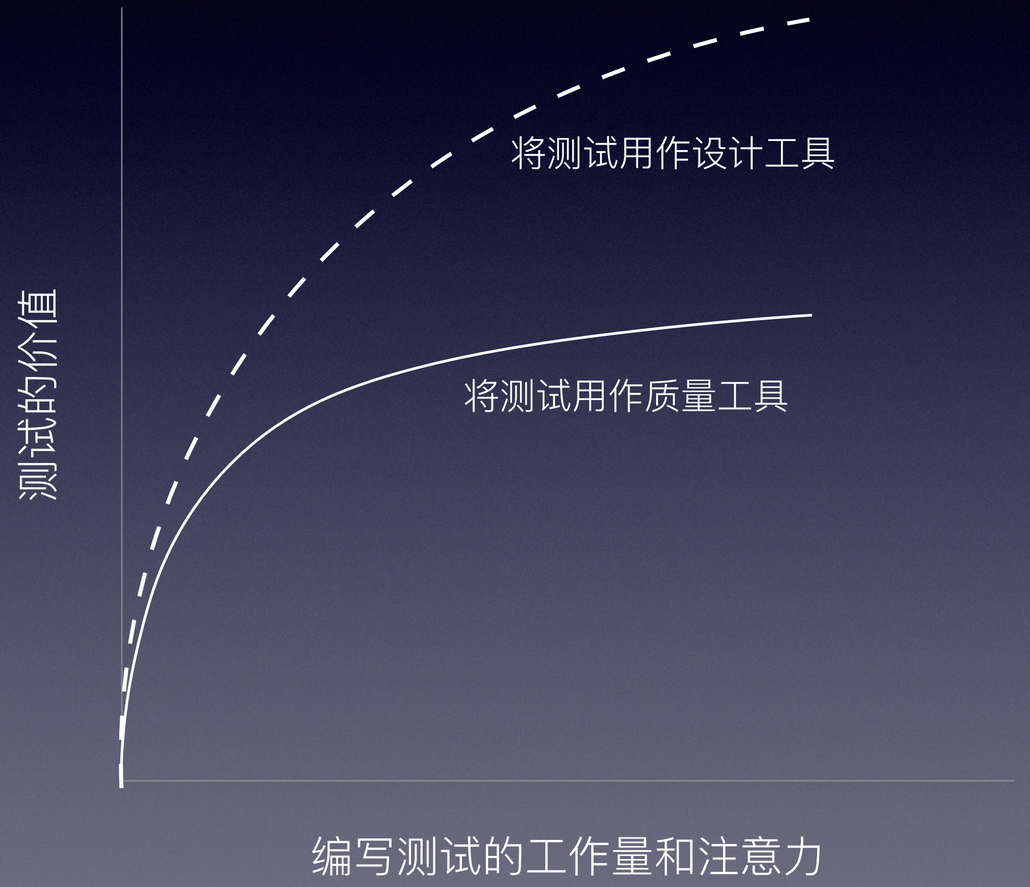


观点

- 单元测试不是用来发现bug的
- 如果不是用来发现bug，那是干嘛用的?

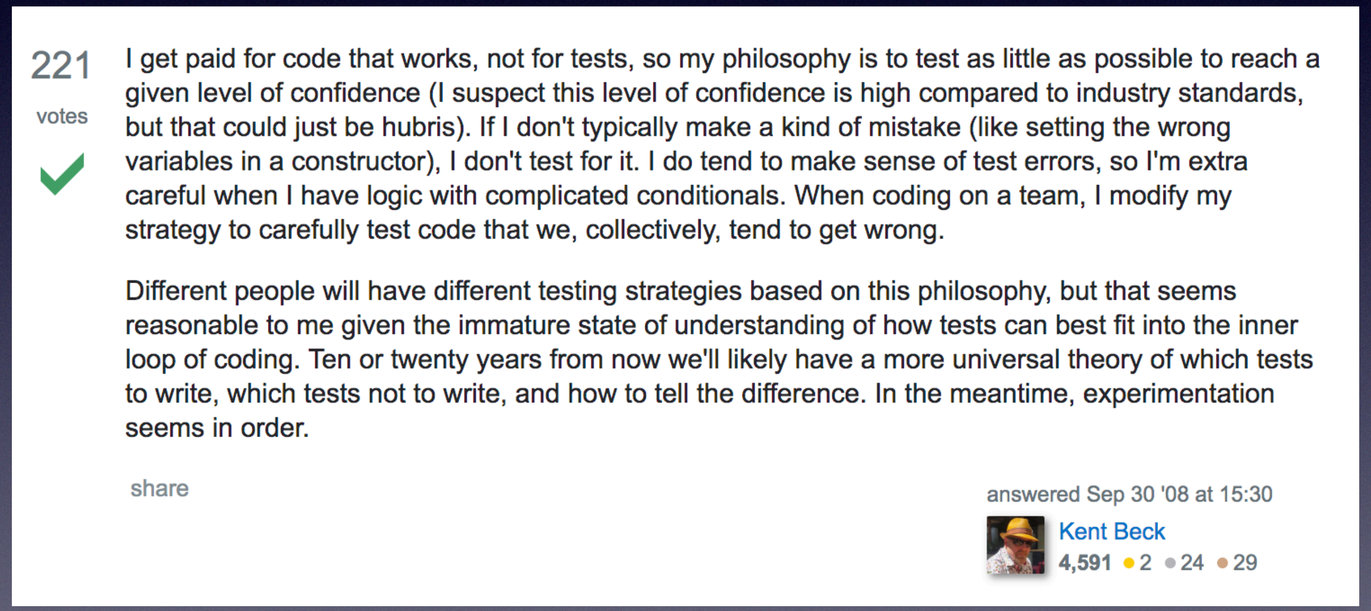
测试的价值

- 使过程更加敏捷
- 提高代码质量
- 更早地发现bug
- 促进改变&简化集成
- 提供文档
- 简化调试过程
- 创造更好的设计
- 降低成本



How deep are your unit tests?

- https://stackoverflow.com/questions/153234/how-deep-are-your-unit-tests



221
votes

I get paid for code that works, not for tests, so my philosophy is to test as little as possible to reach a given level of confidence (I suspect this level of confidence is high compared to industry standards, but that could just be hubris). If I don't typically make a kind of mistake (like setting the wrong variables in a constructor), I don't test for it. I do tend to make sense of test errors, so I'm extra careful when I have logic with complicated conditionals. When coding on a team, I modify my strategy to carefully test code that we, collectively, tend to get wrong.

Different people will have different testing strategies based on this philosophy, but that seems reasonable to me given the immature state of understanding of how tests can best fit into the inner loop of coding. Ten or twenty years from now we'll likely have a more universal theory of which tests to write, which tests not to write, and how to tell the difference. In the meantime, experimentation seems in order.

share

answered Sep 30 '08 at 15:30

Kent Beck
4,591 ● 2 ● 24 ● 29

- 我国的教育对我们最大的洗脑不是掩盖事实，而让我们习惯于标准答案，习惯于教条，从而不会思考！敏捷开发中的若干东西似乎都成了软件开发中对某种标准答案的教条
- 软件开发是一种脑力劳动，是一种知识密集型的工作，就像艺术作品一样，创作过程和成品是没有标准答案的
- 软件的质量不是测试出来的，而是设计和维护出来的。就像工匠们在一点一点地雕琢他们的作品一样
- UT的粒度是多少，这个不重要，重要的是你会不会自己思考你的软件应该怎么做，怎么测试

References

- http://blog.stevensanderson.com/2009/08/24/writing-great-unit-tests-best-and-worst-practises/
- https://blog.csdn.net/flysqrlboy/article/details/79301241
- https://coolshell.cn/articles/8209.html
- https://blog.pragmatists.com/test-doubles-fakes-mocks-and-stubs-1a7491dfa3da
- https://www.martinfowler.com/articles/mocksArentStubs.html
- https://martinfowler.com/bliki/TestPyramid.html