

对 End-to-End 测试说不

转载 weixin_34029680 2016-08-04 19:20:51 380 收藏

文章标签： 测试 javascript 开发工具

你是否还记得有那么一部电影，你和你的朋友都很想看，但是看过就后悔了。你是否还记得你的团队发现了一个“杀手级特性”，发布后看到的却是随弹”。

好的idea常常在实践中失败，而在测试的世界中，一个的好的idea常常在实践中失败，是因为测试策略是基于End-to-End的。

测试人员会花费时间写各种自动化测试，包含单元测试、集成测试、和End-to-End测试，他们花费最多的时间在end-to-end测试上，将产品或者服务当进行测试。值得一提的是，这些测试模拟了真实的用户场景。

理论上的 End-to-End 测试

虽然end-to-end并不是一个好的实践，但是不乏有人说在理论上他非常合理。

Google十大信条第一条：以用户为中心,其他一切自然水到渠成。因此，end-to-end测试聚焦于真实用户场景听起来像是毫无问题的，这种策略也有赢者：

- **开发人员**：可以将几乎所有测试工作都交给其他人来做。
- **管理人员和决策者**：模拟了真实用户场景，可以很直观的看到测试失败的影响。
- **测试人员**：避免测试用例没有验证到真实的用户行为，并且给测试人员带来了成就感。

实践中的 End-to-End 测试

理论上这种测试策略简直是完美的，在实践中出了什么问题呢？我们从测试人员那里搜集到一些反馈。

我们假定测试的基础建设非常完善，每天晚上可以做到：

- 软件的最新版本构建完成
- 构建好的版本部署到测试环境
- 所有end-to-end测试用例在测试环境上运行
- 测试报告以邮件形式发送给团队成员

整个团队在为了新特性的发布忙碌着，转眼间截止日期要到了。为了确保产品的高质量，end-to-end测试用例至少要通过90%。离截止日期还有一天：

Days Left	Pass %	Notes
1	5%	Everything is broken! Signing in to the service is broken. Almost all tests sign in a user, so almost all tests failed.
0	4%	A partner team we rely on deployed a bad build to their testing environment yesterday.
-1	54%	A dev broke the save scenario yesterday (or the day before?). Half the tests save a document at some point in time. Devs spent most of the day determining if it's a frontend bug or a backend bug.
-2	54%	It's a frontend bug, devs spent half of today figuring out where.
-3	54%	A bad fix was checked in yesterday. The mistake was pretty easy to spot, though, and a correct fix was checked in today.
-4	1%	Hardware failures occurred in the lab for our testing environment.
-5	84%	Many small bugs hiding behind the big bugs (e.g., sign-in broken, save broken). Still working on the small bugs.
-6	87%	We should be above 90%, but are not for some reason.
-7	89.54%	(Rounds up to 90%, close enough.) No fixes were checked in yesterday, so the tests must have been flaky yesterday.

分析

尽管有不少问题，但是测试最终找到了真正的bug。

好处：在用户使用之前，影响用户体验的bug被找出并被修复。

问题：

- 团队里程碑推迟一周（伴随着大量的加班）
- 定位一条失败的测试用例非常痛苦，花费了大量的时间
- 团队配合问题和环境问题多次毁掉了测试结果
- 许多小的bug被隐藏在大的bug后面
- 测试有时候包含不确定性
- bug修改后回归测试往往耗时较久，往往要第二天才知道结果

找到了end-to-end策略的问题所在，接下来我们需要改变测试策略来避免这些问题。那么，正确的策略应该是什么样的？

测试的真正价值

一般来说，一旦用例执行失败，测试人员的工作就结束了。记录bug，然后开发人员解决bug。把end-to-end策略分解一下，我们需要从“聚焦用户”跳出失败的测试用例为用户带来了什么好处。答案在这里：

测试用例失败并不会直接让用户受益。

首先，不要被这个结论吓到。一个产品被开发出来后，测试人员并不能左右它。那么如果一条失败的测试用例并不能让用户受益，那什么才能让用户受益？

修复一个bug才会直接让用户受益。

只有当软件没有bug用户才能愉快的使用产品。要修复bug，你必须知道bug存在于哪里。要知道bug存在于哪里，需要测试来找出它（如果测试找不到定会找到它）。但是，从用例执行失败，到修复bug，在整个过程中，最后一步才是真正有价值的。

Stage	Failing Test	Bug Opened	Bug Fixed
Value Added	No	No	Yes

因此，要评估任何一个测试策略，不能只是评估它如何发现bug。还必须评估这种策略下，开发人员是如何 修复（防止） bug的。

构筑有效的反馈机制

测试构筑了一个反馈机制，这种机制影响这产品的运行。好的反馈机制有几个特点：

- **快速**：没有哪个开发人员想等待几个小时来验证新的改动是不是有bug。当出现bug的时候，需要运行多次来复现。快速的反馈机制修复起bug来会果机制足够快，开发人员甚至能在每一个改变后测试一次。
- **可靠**：没有哪个开发人员想要花几个小时去debug，而debug的原因是因为一种不确定的测试。不确定的测试让开发人员对测试不信任，导致一些被忽略，甚至这些结果中有一些是真正的产品问题。
- **隔离错误**：要修复一个bug，开发人员需要找到引起bug的特定代码行。如果一个产品拥有百万行代码，找bug无异于大海捞针。

从小处着手

我们怎么构建一个理想的反馈机制？从小处着手。









单元测试

单元测试可以将产品中的一小块进行隔离。这种方式可以构建一个理想的反馈机制：

- 单元测试是快速的：测试被拆分为一个个很小的单元，每一条测试甚至耗时不到0.1秒。
- 单元测试是可靠的：简单的体系和小单元带来了可靠性。甚至可以进行封闭测试，彻底解决不确定性问题。
- 单元测试隔离错误：即使产品拥有上百万行代码，如果测试不通过，只需要在一小块代码来找bug。

单元测试 vs. End-to-End测试

使用end-to-end测试，不得不等待：编译 => 部署 => 执行测试。不确定性又贯穿始终整个测试执行过程：即使发现了一个bug，bug可能存在于代码的虽然end-to-end测试能更好的模拟用户使用场景，然而相比于其带来的问题，它带来的好处变的并不那么重要：

	Unit	End-toEnd
Fast		
Reliable		
Isolates Failures		
Simulates a Real User		

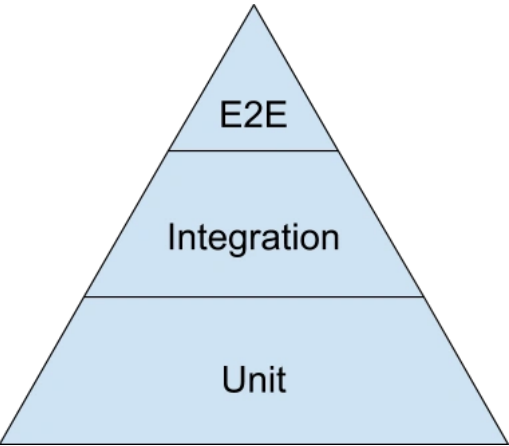
集成测试

单元测试有一个不足：即使独立的单元一个个都运行良好，也无法确保它们一起运行的时候会不会出问题。但这并不意味着要马上进行end-to-end测试是一个集成测试。一个集成测试需要一组单元（通常是两个）做为一个整体，来测试它们的行为。

如果两个单元的集成有问题，写一个更小、更聚焦的集成测试也会找到相同的bug，为什么要写end-to-end测试呢？

测试金字塔

通过了单元测试和集成测试，就需要一些end-to-end测试将整个系统作为一个整体来验证。这三种测试之间的比例，可以用一个测试金字塔来描述。以金字塔的简图：



单元测试位于金字塔底部。越往上测试规模变的越大，测试的数量越少。

Google建议的比例是"7:2:1"：单元测试占70%，集成测试占20%，end-to-end测试占10%。不同的团队比例会有些不同，但是总的来说，应该是一个尽量避免这些不同的形状：

- 倒金字塔：团队主要依赖于end-to-end测试，有很少的集成测试，基本没有单元你测试。
- 沙漏：团队有大量的单元测试，在应该用集成测试的时候使用了end-to-end测试。沙漏型有很多单元测试和end-to-end测试，但是只有很少的集成

正如金字塔是最稳定的结构一样，测试金字塔也是最靠谱的测试策略。

本文系TestBird测试工程师编译整理。想了解更多开发测试相关信息，请访问 [TestBird](#)。

原文地址：<http://googletesting.blogspot...>