《完美软件》

图书基本信息

书名：《完美软件》

13位ISBN编号：9787121099311

10位ISBN编号：7121099314

出版时间：2009-12

出版社：电子工业出版社

作者：Gerald M.Weinberg

页数：345

译者：宋锐

《完美软件》

前言

软件测试的目的是什么？ 我想大多数人对于这个问题的回答会是“保证交付高质量的软件”。也许从最终目的上来说是这样的，但是单纯只是软件测试本身是无法为软件的质量提供保证的。软件的质量是从需求分析甚至是刚刚产生软件相关概念的时候就开始产生，受到整个开发过程影响的一个性质，而测试只是用来将这一性质数值化、表面化的过程。是不是拥有良好的测试过程就能够保证交付高质量的软件呢？显然是不够的。要保证软件具有较高的品质，需要管理、开发、测试等多个部门的共同努力，尤其是管理部门如何看待测试，如何利用测试获得的信息是至关重要的。 当我们按照一个预定的过程对软件进行测试，却未能获得预期的结果时，往往会说：“这次测试失败了。”或者是“这个测试没有通过。”而实际上正确的说法应该是：“软件在这个测试中失败了。”或者是“软件没能通过这个测试。” 在许多人看来，这种说法上的差异也许是微不足道的，或者只是文字游戏。但是事实上，它反映了大家对于软件测试的心理误区。人们往往并不能很清楚地区分测试与除错、开发过程甚至是软件本身的关系。而这一误区正是导致对测试产生不切实际的期望，夸大或者忽视测试的作用的根源。 如果你从事测试已经有很长时间了，就很可能听到过这样一些问题：“你为什么没有发现那个问题？” “我们为什么要在可以自动化测试的情况下雇佣人来进行测试？” “一定要让它在下周可用。” 而产生这些问题的原因，往往就在于人们对软件测试的误解。本书的主要目的，就是要破除这样的一些误解，还软件测试以本来面目。这本书的预期读者非常广泛，涉及与软件开发有关的所有人，包括但不限于软件开发人员、测试人员、客户、项目经理以及高层管理人员，等等。 本书的作者Gerald M. Weinberg在软件行业浸淫了五十余年，几乎开发过所有类型的软件，出版了四十余本技术书籍，发表过数百篇技术文章。他对软件项目的管理、设计、开发和测试具有极其丰富的经验，对于与软件开发有关人员的心理尤其有深入的研究。他的《The Psychology of Computer Programming》（中译本名为《程序开发心理学》）开创了“以人为本”的研究方法，它以其对程序员们的智力、技巧，团队和问题求解能力等方面独特的视角和敏锐的观察经受住了时间的考验，具有深远影响。 而在《完美软件——对软件测试的各种幻想》这本书中，Weinberg采用的是纯散文和故事化的风格，而没有使用花哨的指标和图表。他深入浅出地讲述了与软件测试有关的各种误解与错觉，可以为行政人员、经理或者开发人员提供有关测试的足够信息，以便他们理解测试所要面对的挑战，进而对测试设置恰当的期望并就这些期望进行清晰的交流。 本书回答的问题包括： 为什么在看起来测试只会耽搁时间的时候还要进行测试？ 为什么无法一开始就构建正确的软件，从而不需要测试？ 需要对所有可能性都进行测试吗？ 为什么不对所有可能性都进行测试？ 是什么原因导致测试如此困难？ 为什么测试需要这么长的时间？ 是否有可能构建完美的软件？ 为什么我们不能接受一些缺陷？ 本书的入手点更多的是出于心理学而不是计算机技术。通过对心理学的研究，我们会发现人类本身就是不完美的，自然也不应该期望软件是完美的。由于除了某些最简单的情况以外，软件的可能路径实际上都是无限的，因此对软件进行穷举测试通常是不可能的。由于人类容易出错，而且程序中产生缺陷的可能途径也非常多，所以没有任何缺陷的（完美的）程序是一个无法实现的目标。虽然这不是什么好消息，但是事实就是如此。 受乐观主义和必要性的推动，人们每年仍然会创造数百万行代码，而这些程序中的相当一部分将会被发布到到客户群中。因此，谨慎的开发人员别无选择，只能通过建立测试过程来发现软件中最重要的和最有可能出现的那些缺陷。 正如Weinberg 指出的那样，有些人确实选择只进行很少的测试，有时是依赖于诸如“如果我们运气好的话，就没有人会遇到那个缺陷”的想法。其他一些人只是将该缺陷重新定义成一个特性就继续别的工作。这些做法既不合乎道德，也可能导致法律问题。

Weinberg以非常诙谐而含蓄的风格介绍了他遇到过的众多例子，揭示了与软件测试有关的众多常见误解与谬论，还揭露了在测试中常见的各种欺诈方式。通过阅读本书，应该认识到软件测试是一个与人的心理活动密切相关的过程；测试工作的核心是收集信息，这些信息是关于软件产品、开发过程以及测试过程本身的；要达到“保证软件质量的目的”，不仅仅要进行良好的测试，更重要的是要对测试获得的信息进行合理的利用。 本书是给涉及软件开发的所有人的一本指南，告诉他们：对于软件而言，完美是一个不现实的目标。因此，本书对软件开发团队中的所有人都是必备的。

《完美软件》

内容概要

《完美软件:对软件测试的各种幻想(中英文对照)》是从事软件行业五十余年的Gernald M. Weinberg针对软件测试所写的新作。他在软件项目的管理、设计、开发和测试方面都具有极其丰富的经验，对于与软件开发有关人员的心理尤其有深入的研究。在《完美软件:对软件测试的各种幻想(中英文对照)》中，他重点讨论了与软件测试有关的各种心理问题及其表现与应对方法。作者首先阐述软件测试之所以如此困难的原因——人的思维不是完美的，而软件测试的最终目的就是发现对改善软件产品和软件开发过程有益的信息，故软件测试是一个信息获取的过程。接着，作者利用丰富的经历和大量的实例，展现了在软件测试中可能会出现的各种与人的心理有关的现象、误区、欺诈，以及容易犯下的常见错误等等。《完美软件:对软件测试的各种幻想(中英文对照)》的重点不是告诉大家要做什么或者说如何做，而更多的是让读者明白在与软件测试相关的活动中会出现某些特定现象的原因。理解这些与人的心理有关的现象有助于与软件开发有关的所有人之间更好地就软件测试的目的和实现过程进行沟通，从而实现具有更高品质的软件。

作者简介

温伯格，软件领域最著名的专家之一，美国计算机名人堂代表人物，Weinberg&Weinberg）顾问公司的负责人。Weinberg精力旺盛、思想活跃，从20世纪70年代开始，他总共撰写了40多本书籍和数以百计的论文。在西方国家乃至全球，Weinberg拥有大量忠实的读者群，他们甚至建有专门的组织和网站，讨论和交流大师的重要思想。可以说，Weinberg近年来的每本新书都是在万众瞩目中推出的。

《完美软件》

书籍目录

《完美软件》

章节摘录

产品何时准备好接受测试了？这个问题并不总是很容易回答，但是可以通过问一些更容易回答的问题来了解一些信息，帮助确定何时尚未准备好接受测试。　　对于产品是否存在一个以上的问题.需要通过测试来帮助回答？如果没有任何问题，就没有理由进行测试。　　是否希望知道该问题的答案？如果不想知道答案，就不要提出问题，也就是说，不要进行测试。　　是否只是无聊地对测试结果感到好奇？如果并不准备对测试结果进行估算或据此采取行动，就不要考虑测试。如果只是为了满足无聊的好奇心而进行测试，代价就太高了。不过从另一方面来说，积极的好奇心是有效测试的一个重要方面，因为你永远无法准确地知道自己应该寻找什么。因此，如果准备花一些时间来进行探索，保持好奇心就有助于让测试物有所值，不过一定要对发现的情况追根究底。　　是否能够和测试人员预先就如何算通过测试达成一致？如果不能就怎样才能算通过测试、怎样不能算通过测试达成一致，那你想通过测试得到什么呢？　　是否已经预先就怎样的测试是成功的、怎样又是不成功的达成了一致？是否认可只要是能够提供新信息的测试，就至少可以被看作是部分成功的？　　是否以为测试的结果会替你做出决定？不可能从纯技术的角度做出商业决策。当然应该使用测试得到的信息来支持商业决策，但是不要用测试代替商业决策过程。例如，交付一个未能通过某些测试的系统可能是一个不错的商业决定。与之相反，交付一个通过了所有测试的系统也可能是一个不良的商业决定。

《完美软件》

精彩短评

1、想充实一下自己
2、不错的软件测试书
3、世上没有完美的软件，测试是为了让用户能更好地使用它。说到测试，读了这本书后你会明白：测试不只是测试，测试也不是你想像中的测试。
4、Coffee 昨日推荐的
5、读的第三本温伯格的书了，所以能学到的就少了点。但是依然是本好书，真想完全理解还得实操，可惜我现在没这个机会。
6、电子工业出版的，感觉不太好。中英文结合，排版不好看。
但是书我还是买了。花了一天的时间看完了。
感觉很有温柏格的风格。但是感觉还可以写得更好。

7、很久之前的书，但是其中的观点在现在还是对的
8、软件测试...
9、读过的软件测试类著作中很贴近工程实际的一部，工作若干年后一看，几乎所有的问题都发生过，心有戚戚焉。推荐从事软件测试和QA的同学读一读，对职业发展也有帮助！
10、一通百通的道理
11、翻译的有点差劲
12、读起来非常舒服，道理却很深刻！
13、翻译太绕口，多亏是双语版
14、这是一本客观评价软件开发中与测试人员交互现状的书。可以被用来攻击，也可以用来防御。如温伯格过去的研究，这些都可以用心理学来解释。或许是一部测试员心理学，或者又一部阴阳博弈式的著作，提醒每一个人重视测试方法的应用，也提醒每一个人要尽量避免无知的测试人员所带来的干扰。学会和测试人员共事，需要一种技巧。
15、第一本“力荐”的书，尤其是将软件测试作为职业的测试人一定要读一读本书！
16、感觉像写段子的~由于不是做测试，走马观花翻了一遍
17、好吧……这周HoF准备说这个……
18、度过他的好几本书了，感觉这一本是最不值的。或许是里面的东西已经在他的其他书里都应经讲到过了吧。还是觉得那套《软件-质量-管理》最好。
19、重读。
20、哈~ 若是一个开发人员没有在这本书里找到自己的影子会有两种可能：伪开发人员，代码圣人
21、有点玄学的感觉，收获不大
22、等待做笔记整理了。。。
23、作者把书名翻译错了。。把illusion译成幻想。。幻想你妹啊
24、是一本QA的书。
25、温伯格对测试的深刻见解，软件和测试一样，都不可能完美
26、“人生命中很大一部分是在查找自己的错误”
27、有点贵额
28、这本书从心理学的角度去分析工程质量的把控，这是我从来没有注重过的角度。作者分析了心理学从工程的初始到结束如何影响工程质量，遗憾的是这些影响来自人类的弱点，想要避免恐怕没有通用方法
29、你不可能仅仅通过测试来保证软件的质量，就像人们不能仅仅通过体检来提高身体素质一样。软件测试只是一个采样过程。
30、花了两周时间看完了，内容不算多，但是还是比较全面的。本书关注软件测试的各个方面，偏向于方法的研究。感觉每章最后的“常见错误”很有指导意义，可以做一个摘要出来经常提醒一下自己。

31、用了很多例子, 说明了大众对测试的误解

精彩书评

1、测试不是提高产品质量的过程，而是收集产品会在运行过程中做什么之类信息的过程，但其可以作为提高产品质量的参考。弄清楚测试的目的，该采用什么样的策略，不要做无用功。如果不准备给开发人员必要的时间或资源来fix bug，那么测试也是无用功。在很多情况下，项目成员可能希望在交付产品后继续进行测试，以便收集对客服和支持人员有用的信息。
2、作为芸芸众程序员的一员，我对软件开发中的一切都充满问题。今天是关于测试，作为一名唯物主义者，我相信众物都有其神，于是我找到了测试之神。我问：神仙，为什么我们需要测试？大神用怜悯的眼神看着我，说到：我可怜的孩子，之所以需要测试，都是上帝的错啊，上帝创造了你们，但是因为没有测试，所以你们都是不完美的、不理智的，你们会被情绪、环境左右，会犯错。大神说这话的时候充满了怜悯，当然，他对任何事情都充满怜悯。我想，我该叫他怜悯之神。我说：哦，这么说上帝也是不完美的，因为他也需要测试？怜悯之神果然是神仙：咳咳，说到哪儿了，你吃饭了吗？我问：我们需要测试，可是，为什么我看测试人员只是天天敲键盘而已啊？怜悯之神说：你敲键盘是在写代码，测试人员敲键盘是在获取当前系统的信息，这两者真正的工作都是那些脑力活动，是在敲击键盘这个物理行为之前以及伴随这些物理行为的脑力活动。如果你敲击键盘的时候没有进行思考，那么你就不是在进行开发和测试；而且，如果你在思考但是没有敲击键盘，你还是可能在进行开发和测试。我说：关键是思考。怜悯之神说：是的。他显然对后半句犹豫了一下，但是还是说了，人类一思考，我们就发笑。我想，一点都不好笑，连朝鲜都一比零小胜巴西了。我说，其实对软件开发人员来说，我们也应该通过使用自己的产品来对它进行测试。怜悯之神点点头，说，这叫做“吃自己的狗食”，如果你都对自己的产品感到担心和鄙夷，别人为什么要买它？我说，可是，除非我们开发的是开发工具，否则我们根本不可能用它。地球人都知道，做减肥药广告的从来不喝减肥药，买火腿肠的从来不吃火腿肠....人人都需要是化学家。怜悯之神说：所以完全由开发人员构成的测试样本不太可能代表整个用户群。我自言自语道，所以我们需要测试人员，需要另一个角度的思考。我问：尽管测试人员测试了，但是为什么系统还是存在缺陷哩，难道他们不对所有可能性都进行测试的吗？怜悯之神翻了我一个白眼，叹了口气。作为报复，在下面的叙述中，我将称他为白眼之神。白眼之神说，对任何程序而言，可能进行的测试数据都是无限的。测试也许可以令人信服的表明存在缺陷，但是永远无法表明不存在缺陷。我想了想，说，我们现在的系统需要导入客户的遗留数据，光报表就多达20万，如果一张张的校验报表内容和样式，那么一定会死人的。白眼之神说，那你们是怎样测试的哩？我说，抽取样本测试哈。我说，我明白了，由于我们无法测试所有的可能性，那么任何测试实际上都是某种程度的样本测试，这些样本以某种方式代表整个可能测试集合的一个部分或者片段。白眼之神点点头，说，测试只是采样！我说，正像我们去医院体检，所有化验单上都写着，本结果只对该样本负责。白眼之神说，实际上，采样也是一个心理过程，而且是一个感性过程，令某人满意的样本也许会让另外一个觉得一点都不满意。我说，由于不可能进行穷举测试，所以我们往往在两个目标间徘徊：希望测试能够覆盖所有令人感兴趣的条件，希望测试集减少到可以管理和承受的程度。就像吃自助餐，希望吃到所有东西，但是又要不把肚皮撑破。白眼之神说，所以我们需要尽可能选择那些具有最强代表性的样本进行测试，而这是测试人员的优势。另外，多样化样本发现的问题可能超过大样本发现的问题，同样，测试团队多样化也可能发现更多的问题。我说，是的，同样是抽血，一些人会要求早上不吃饭抽取静脉血，一些人则需要间隔取几次血。我问：开发中，我们采用了TDD的方式，我们单元功能测试的覆盖率也很不错，这样，我想我们可以减少测试人员？减少系统测试？神说，你会因为一架飞机保证它所有的部件在组装前都进行了测试而乘坐它的处女航吗？我说，哦，罗老号发射失败了。神说，但是，良好的单元功能测试能够为系统测试去除噪音。我说，缺陷都不是自己钻进去的，而是开发人员放前去的。单元功能测试能够在一定程度上阻挡开发过程中缺陷的进入，同时阻挡一些简单的缺陷，系统测试不能捕获所有缺陷，相对单元功能测试，系统测试会花费更长的时间和成本，这些时间和成本能够通过单元功能测试得到部分节约。神说，开发人员的测试保证了他对需求的理解和实现的一致，但是开发人员对需求的理解和真正的需求一定一致吗？我说，所以需要测试人员即时验收。神说，另外，关于TDD，从心理学的角度说，一个人是很难发现自己的错误的，所以TDD和结对编程一起实践会有比较好的效果。我问：为什么很多项目最后的集成测试阶段会那么长哩？这总是导致我们的项目延期，而几乎所有的项目都会延期。神说，说说你们的集成测试阶段都在做些什么？我说，噢，我们拉出一个发布分支，冻结代码提交，开始进行测试，找出缺陷，查明缺陷原因，根据重

要性修改缺陷，然后重新测试，以此循环。恩，问题在于此时总会发现大量缺陷，并且我们每修复一些缺陷总会引入新的缺陷，所以这个时间拉得很长。神说，修复缺陷的同时引入新缺陷，这叫做故障反馈率（FFR），也就是一个修复让系统中产生了另一个缺陷的情况所占的百分比。这个数值如果在50%以下就很幸福了，另外，压力和试图提高缺陷修复速度都会提高FFR。我说，也就是修复缺陷的同时引入新缺陷是个正常情况。恩，可是我的问题是为什么最后的测试阶段需要花费这么长时间？神说，难道你不觉得是你们经理错误的估计了最后集成测试阶段所应该花费的时间？我说，噢！神说，此外，测试不等于除错。我想了想，说，是的，说的是最后集成测试阶段，但实际上包含了两个行为：测试和除错，相比测试，除错占据了更多时间。恩，我们现在的一个项目特性，由于开发时没有考虑性能问题，导致上线前花费了大量时间进行调优。神说，项目延期的主要问题在于测试推迟。我说，是的，这个给我的印象很深，如果在开发阶段就进行大数据的测试，那么会节省很多时间。一个缺陷，发现的越晚，修复的成本就越高。神说，要测试先行，并且测试往往在项目开始开发前都已经开始了，测试需要贯穿于整个开发过程，频繁进行，而是不推迟到最后的集成测试阶段。TDD和持续集成是很不错的实践，但是它们仅仅是测试中的一部分。我说，那么，测试保证了质量。神说，三鹿没有测试吗？我说，....神说，测试只是提供信息。至于这些信息的定义、重要性以及所要采取的反应都取决于人，而人做出的决定都是感性的，利益驱动的。神说，如果开发的产品本身就质量低劣，进行测试你不觉得是浪费大家的时间吗？对低劣的代码测试得再好又有什么用呢？一段时间发现的缺陷越多，并不意味着剩下的缺陷越少，而总是意味着会出现更多的缺陷。我说，我明白了，测试只是收集信息，除此之外，并不能做其他任何事情。正如我们去体检，体检报告只能反映出当时我们的身体状态，至于健不健康，则取决于我们平时的生活习惯，这是两个分开的事情，体检并不保证我的健康。神说，很多大公司非常看重测试部门，原因其实是他们无法看清楚他们自己的开发过程，于是只能从测试里获取信息，而这些信息对整个软件开发来说只是一部分而已。我看到过有项目经理向测试人员询问是否可以交付，这根本上就是在推卸责任，信息和作出决定根本是两回事，更何况测试收集的信息只是部分信息呢，如果是这样，不如让测试人员来当经理好了。我说，怪不得每次去检查身体问化验医师有没有问题正不正常，化验医师总是不耐烦的说去问医生呢。我问：什么测试才是良好的测试呢？神说，你觉得呢？我说，测试的目的是发现缺陷，所以发现的缺陷越多，那么测试越好。神说，然后呢？我说，我会告诉测试人员，啊哈，你们发现的缺陷越多，我就发给你们越多的奖金。神说，又是一个糟糕管理的绝佳范例。我说，确实会有问题，这样对测试人员来说，一个低劣的系统对他们的价值最高，他们会爱上那些糟糕的开发人员，引发办公室恋情。这和整个团队的目标-交付高质量的系统矛盾，并会在开发部门和测试部门之间引起矛盾和争执。还有，发现的缺陷会增加，但是获得信息的质量却降低了。神说，你混淆了测试的目的，测试的目的不是发现缺陷，而是获取我们所需要的信息。作为对比，如果你所有体检的项目都正常，你会觉得体检没有价值吗？我说，恩，我会很高兴。神说，测试只是采样，所以良好的测试需要能够根据上下文覆盖系统中最重要的部分，而且又不能膨胀到不可管理的地步。我说，想起来了，我们在体检的时候往往会有不同的检查套餐，例如针对白领的，会着重检查胃和颈椎；针对40岁以上人群的，会增加癌症因子筛查。也就是测试并不是一个孤立的行为，它需要根据不同的情况采取不同的策略。而且，我们总希望体检时不用花太多的钱。神说，“良好”并不是属于某个测试的属性，它只能是测试、实现、成本和应用场景四者之间关系的属性。我说，无法衡量测试？神说，作为比较，系统上线后用户的使用也可以看做是测试的继续。我说，你的意思是我们可以对系统在使用中出现的缺陷加以追踪，然后进行统计和分析，例如测试比较典型的遗漏了哪种缺陷，而哪种测试实际上是没有太多必要的，因为用户根本都没有使用该功能。这样，以后我们就可以对测试加以改进。我问：那么，我们需要对所有缺陷都跟踪记录？神说，你们没有对所有缺陷都跟踪记录？我说，有些缺陷太小了，例如拼写错误，有些缺陷很容易就修复，所以测试人员就直接和我们说了，没有填写缺陷记录。神说，你怎么看？我说，我觉得还不错，没有一个开发人员愿意自己的程序里出现错误，我们往往把程序看成了自己的扩展，指出程序有错误就是在指出我有错误，所以，对于一些很小、由于疏忽引起的错误，测试人员直接和我说而不记录让我感觉好一些。神说，缺陷只有在系统交付后才成为错误。我说，好吧，可是报告我的程序出现缺陷会让我受到批评。神说，那是另外一个关于团队管理的问题了。如果一个人花在指责其他人上面的时间越多，那么解决该问题的可能性就越低。我说，可是，我们应该反对文档，文档阻碍了交流，实际上没有太大的价值。神说，文档在不使用的情况下才没有价值。一些人反对的不是文档，而是自己写文档。信息在文档化之后才能被追踪和统计，特别是作为项目信息一部分的缺陷信息，它反映了项目的状态，一定不

能够被忽略，如果为了“ 节省时间 ”和“ 面子 ”而不记录缺陷，那么导致的结果就是项目状态的丢失，会给项目后续的判断带来影响，以后会浪费更多的时间和金钱。我说，我想起来了，每次去取体检结果时，我都会看到本次体检指标与历次的对比，这次的结果提醒我胆固醇又提高了，颈椎则好了很多，所以要少吃肥肉多锻炼，同时晓庆教我的颈椎病锻炼要继续坚持。我问：收集信息似乎只是第一步，信息的处理过程又是怎样的呢？神说，你们是怎么做的呢？我说，首先，测试人员会选择样本对系统进行测试。比如发现系统不能上传图片了，测试人员会报告一个缺陷，记录下详细的信息。神说，这是第一步，摄取信息，此时，信息在被人确定含义之前是没有意义的。我说，接着，我们会看到这个缺陷报告，并和测试人员进行交流。测试人员表示了担忧，因为上传图片对客户来说是一个很重要的功能，现在连它都不能正常工作是否意味着系统的其他部分也存在很多问题。神说，测试人员在给信息赋予含义。我说，我们和测试人员一起重现了这个缺陷，系统报出的问题是权限认证失败，我们认为这是一个权限方面的问题，上传图片的其余功能应该不受影响，也许是有人改动过当前测试用户的权限，也有可能是上一次提交破坏了什么东西，总之影响不会很大，因为这块有很多的单元功能测试，如果是提交破坏，也应该是页面上的代码，这段代码不多，错误能够很快定位和修复。神说，你们在给信息赋予含义，不同的人会给信息确定不同的含义。我说，接下来，我们会讨论这个缺陷的优先级。测试人员认为这个缺陷很重要，应该立刻修复；我们认为这个缺陷虽然重要，但是非常容易修复，并且此时有另外一个非常重要的缺陷需要修复；项目经理询问了我们各自对这个缺陷的理解，说，我们现在有一个特别重要的关于图片搜索的缺陷需要修复，我认为这个缺陷最重要，因为后天需要给客户的老板进行演示，上传图片的功能也很重要，但是不在演示的范围之内，但是既然很容易定位，我们也可以迅速的查明引起问题的原因，但是不修复。神说，现在在确定信息的重要性，并做出反应。不同的人会给信息确定不同的重要性。我说，那么完整的过程应该是：摄取信息->确定含义->确定重要性->做出反应。神说，是的。我说，等等，我看到问题了。因为不同的人会确定不同的含义，不同的人会确定不同的重要性，所以对做出决定的人来说，他一定要有自己的理解和权衡，并根据自己的重要性和其他信息最终做出决定。但是现实情况却并非如此，很多信息在提供给管理者之前已经被信息提供者赋予了他所定义的含义和重要性，而管理者由于不能获取其他信息或干脆就没有思考，所以就直接根据信息提供者所定义的含义和重要性做出决定了。神说，你想到了什么？我说，忠臣、奸臣和昏君。其实没有忠臣和奸臣，只有昏君。看似管理者高高在上，手握生杀大权，但其实权利早就被信息提供者瓜分殆尽了。神说，你扯远了。我问：好吧，下一个问题是如何避免测试困难呢？神说，你觉得是大的系统容易测试还是小的系统容易测试？我说，当然是小的系统容易测试。神说，那么，让系统尽可能的小。我说，现在一个普通游戏都要好几个G。神说，第一，让需求受控。这是一项很重要的管理工作，如果没有很好的完成这项工作，那么就是管理上的失败。我说，这似乎很困难，因为客户总是在要求我们加功能，而有些功能确实是很重要的。神说，如果在项目后期要求增加功能，而这项功能又是必需的，那么实际上是在告诉我们需求分析出现了问题，出现了需求遗漏。而一项功能如果最终因为各种原因其实不能使用，那么也是需求分析出现了问题，出现了需求冒进，脱离了实际情况。神说，要控制需求的增长，就需要决策者、需求分析人员和客户来区分某件事对客户来说真的是必需的，需要价值驱动。我说，第二呢？神说，第二，不要试图在软件中处理所有的可能情况。我说，这个我有印象，我们需要向新系统中导入遗留报表，由于遗留数据跨越十年，所以存在很小一部分不规范的网页，如果在程序里包含对所有报表的处理，那么是相当困难的事情，最后我们选择了手动处理这部分数据。神说，第三，代码质量。最好的实现永远是最简洁的代码，要尽量减少代码的复杂性。两个具有相同物理规模的程序在内部复杂性上可能有极大的不同，而这最终会是决定测试工作难度的主要因素。我说，我们可以通过增量构建，不一次做所有事来控制代码的复杂性。神说，此外要注意组件之间的分离，特别是多团队协作的项目。我说，是这样的，对于大型项目，往往会划分以模块为单位的小组开发，小组之间、模块之间要尽量减少依赖和不必要的沟通，因此代码可以有冗余，每个开发小组都要各自独立，有自己的分析人员、开发人员和测试人员。提到增量构建，我又有了新的问题。我问：我们现在采用迭代式开发，那么每次迭代完成后对客户的演示能不能算是用户验收测试呢？神说，演示不是测试。我说，客户触摸到了真实的系统，直观了解到了系统的功能和使用方式，获取了系统信息，然后进行反馈，这可以看作是部分用户验收测试。神说，对客户来说，如果没有真正的使用该系统，那么就没有进行测试。此外，客户获取的信息难道不是你们准备好的吗？我说，是这样，演示之前我们会进行很多准备工作。神说，实际上是在准备客户能够获取的信息。我说，那么最理想的迭代开发应该是持续部署？神说，持续部署和持续增量使用。

3、英文名：Perfect Software : and Other Illusions about Testing 作者：【美】Gerald M Weinberg前言帮助那些想了解测试的人第1章 进行测试的原因“ 人生命中很大一部分是在查找自己的错误 ”真正的人知道他们无论如何尽力地想完成好一项工作，都有可能会出错。真正的人还知道并不是他一个人存在这样的不可靠，其他人同样如此。我们关心的是真正的人如何才能让自己做出的决定比不屑于测试的完美主义者做出的决定更好。实际上，真正的人做出的决定的确比那些认为自己总是完美的人做出的决定更好。不过只是更好的决定，而不是完美的决定。通过测试我们需要保证软件避免的一些问题：1）无法满足关键的用户约束2）无法完成大家想要它完成的工作3）向用户强制提出提出无法接受的成本和/或要求4）软件本身的失败让用户丧失信心5）用户感觉到浪费时间6）太贵对以上6个方面的检验也称之为“ 验收测试 ”信息有助于降低风险。但是要注意，信息的获取需要成本，这些成本很可能超过了信息本身的价值。虽然解决测试提出的问题应该有可能降低风险，但是对风险的评估是主观的（因为它是对未来的评估），同时不同的人对同一种风险的感受是不一样的。进行测试的7步曲：1）软件是否能够做我们希望它做的工作？2）如果软件不能完成我们希望它做的工作，需要做多少工作才能解决这个问题？3）软件是否没有做我们不希望它做的事？4）软件是否能够达到我们的意图，做我们认为值得做的事情？5）软件是否满足了其他的商业需求？6）失败的可能性和后果严重性如何？（测试越多，失败的可能性越低，但严重性越高）小结测试的根基在于心理学，在于对人脑行为的研究。如果人类是完美的思考者，我们就不需要对工作进行测试。如果我们是没有感情的机器人，就总是可以通过合理的方式来使用测试降低我们做出的决定中蕴含的风险。如果我们是从同一个模子克隆出来的，那么我们就会用相同的方法来评估风险。但是，上帝是不会让我们的生命枯燥无聊的，我们是不完美的、不理智的、价值驱动的、相互不同的人。因此，我们要进行测试，并对测试（过程和方法）进行测试。常见错误1）力争完美：这是不可能实现的，试图做不可能的事会让你完蛋2）不做决定：做决定是由经理负责，测试人员的指责是为决定提供信息（但不是所有的信息）3）为使认识作出决定所需的所有信息：在作出决定是未能考虑所有的相关信息，就是在拿管理、人员和用户满意度冒险4）错误地考虑了风险的优先级：如果你只关心自己是否升职，那么只要你不再对产品负责，那么就不用关心快速交付产生的成本何时才会终结5）相信测试可以改进产品：测试是收集有关一个产品的信息，但测试本身并不会修复它发现的那些错误。测试并不会改进产品，改进是由那些修复测试发现缺陷的人实现的。6）相信存在一个“ 测试阶段 ”来完成他的所有测试而且只进行测试：现实里面，“ 测试阶段 ”实际上是“ 修复阶段 ”，测试阶段常常在测试完成之前就结束了（在不考虑由于不看车窗外而发生交通事故的情况下，安全驾驶常常会花更长的时间）。第2章 测试无法做的事“ 不用害怕完美——那是永远无法达到的 ”经理们常常要做出带有风险的绝对。一般来说，如果根据下面这6个问题的答案来做决定，可以减少其中的危险：1）现在就要交付吗？或者晚一些交付？或者根本不交付？2）项目是否要取消？3）是否要继续这个项目并增加更多的资源？4）是否要减小项目覆盖的范围？5）是否准备将这个产品推向更广阔的市场？6）是否要以较低的价格来销售这个版本的产品？然而，很多事情是测试无法做到的：1）信息未必有助于降低风险：由于测试是一个获取信息的过程，所以总会存在是否值得为更多的测试进行投入的问题，而这又是一个需要作出的决定。一方面，有些时候进行更多的测试会增加风险（时间成本），测试不可能不花时间或者免费进行。另一方面，测试提供的信息也会增加风险。有时候，无知确实是一种幸福（尤其是从法律角度而言）。因此，要时刻记住：当心可能会被证明是过量的信息。不过需要认真考虑的是，首要的目标是什么——成功的产品、避免诉讼还是职业生涯发展？2）不会使用测试得到的信息：我们不能把信息的质量和数据的数量等同起来，我们投入测试不是为了得到数据，而是为了得到信息。在这种情况下，也许缺乏信息本身就体现了最重要的信息，要让测试的钱花的值，需要继续进行质询的过程，并解释在测试报告下隐含的那些信息。如果不探寻隐藏在表面下的这些问题，就无法让在测试上的投资物有所值。所以如果不准备使用测试产生的信息，那就不要对测试进行投入。3）决定是感性（而非理性）：人们都存在一种感情倾向，不希望发现自己犯了错误。所以当你试图影响这些与情绪有关的因素时，很可能事与愿违。4）不良的测试（考虑不周或者执行过程不佳）：如果你怀疑自己正在进行不良的测试，不妨考虑根本就不进行测试。5）产品未准备好接受测试：通过以下问题来了解一些信息，帮助确定产品是否准备好接受测试：1）产品是否存在一个以上的问题，需要通过测试来帮助回答？2）是否希望知道该问题的答案？3）是否只是无聊地对测试结果感到好奇，而并不打算对发现的情况追根究底？4）是否能够和测试人员预先就如何算通过测试达成一致？5）是否已经预先就怎样的测试是成功的、怎样又是不成功的达成了一致？6）是否认为测试的结果会替你做出决定？7）测试结果是否游客恩导致你

改变决定？（如果没有可能的话，为什么想要知道结果，更别说还要为之而投入了？）小结如果存在任何原因导致不需要使用测试结果提供的信息，就没有必要进行测试。而且，如果测试将要得到的这些信息是不相关的或者不可靠的，最好就不要使用它，那么一开始也就没有必要花钱进行测试。常见错误1）不尊重测试人员：测试从来都不是只要雇了人就可以坐等结果的事。要么换人，要么帮助他们提高可信度2）过度尊重测试人员：如果让他们代替你作出决定，那你应该辞职去给他们打工3）让测试人员当替罪羊：这是道德问题了.....4）不使用通过测试或者其他渠道获得的信息：实际上如果不使用这些信息的话，它们实际上就不能看作是信息5）做出感性而不是理性的决定：并不是说要绝对的理性，但在作出决定的时候至少应该冷静并控制好自己的情绪6）不对测试数据进行质量评估：数字本身就是数字，验证和批判是不可或缺的7）在未做好充分准备的情况下进行测试：准备好在问题出现的时候发现它们的存在。准备好学习、设计、有效行动、目的和本性的基本了解8）未能协调好测试和项目其他工作的关系：如果没有必要的时间和资源去修复测试中发现的问题，那还是浪费时间测试了9）催促测试人员：测试是非常精细的工作，催促只会导致危险的有误导性的结果10）不坚持要求经理们做出应有的勤奋：如果允许经理们以不了解测试过程为借口允许缺陷通过，可能还是不测试更好11）只是由于别人的决定和你的不同就认为他们是非理性的：很多看起来非理性的决定在不同的价值取向下是理性的，避免在经理、测试和开发之间的大部分冲突。12）未能认识到测试产生的信息有多种用途：在很多情况下，项目成员可能会希望在交付产品后继续进行测试，以便收集对客服和支持人员有用的信息第3章 不对所有可能性进行测试的原因“ 测试也许可以令人信服地表明存在缺陷，但是永远无法表明不存在缺陷 ”我们无法实现“ 测试所有可能 ”这样的要求，这是因为1）首我们的大脑不仅会犯错，而且它的容量也是有限的2）其次，没有人的生命可以无限延长（长到给我们足够的时间去做所有可能的测试）3）测试需要成本（而对任何程序来说可能进行的测试数目是无限的）这里有一些不可能对所有可能性进行测试的因素：1）可进行测试的数目是无限的：测试可以让人筋疲力尽，但它是不可能穷尽的2）测试最多只是采样：由于我们无法测试所有的可能性，所以任何实际的测试集都是某种程度的样本——以某种方式代表整个可能测试集合的一部分或者偏短（我们通常理所当然的希望它是一个良好的代表）。本质上，采样是一个心理过程，而且也是一个感性过程。令某人满意的样本也许会让另一个人觉得一点儿也不满意。3）信息的成本可能超过无知的成本4）我们可以用较少的测试获取更多的信息：任何测试集都是一种采样方法；无论你有多少资源，都要尽可能选择那些具有最强代表性的测试集。我们通常在2个目标之间进行取舍，来达到我们认为比较满意的测试结果：测试覆盖所有令人感兴趣的条件，以及将测试集减小到可以管理、可以承受的程度。其实，做测试很像吃自助餐：在面对无法完成的测试任务集，测试人员必须要注意到有限的测试、资源和时间带来的限制。为了进行良好的测试，测试人员还必须注意自己的个性，也就是他们选择自助餐的方式。小结本质上，任何特定的候选产品上可以进行的测试数目都是无限的。经理们和测试人员必须尽力了解采样给测试过程带来的风险，而不是要求执行“ 所有的 ”测试。常见错误1）要求“ 对所有的可能性 ”进行测试：当要求不可能实现时，就不知道得到的会是什么，除了知道它不是不可能的（也就是得到的和要求的是不一样的）2）不理解采样过程：很少有经理（实际上很少有人）能很好地理解采样过程。那样的话，要么教育自己理解采样，要么请专家对采样过程进行审查。不管采用哪种方法，都要准备好出现采样错误的可能性。3）获取信息的成本超过了信息带来的好处：（你的地下室或者车库是否堆满了从来没有真正需要过的小玩意？）当心你所要求的信息。4）为了“ 做样子 ”进行测试：不要欺骗自己认为得到的信息有很高的质量5）没有使用所有的信息源：从测试结果中收集的信息从其本性来说就是有限的，但是如果你足够警醒，就会发现周围还有很多其他类型的信息未被采用。6）认为虽然人不能进行穷举测试，但是机器可以：并不只是人脑是有限的，测试工具同样是有限的。即使有产品能够“ 进行所有测试 ”，你也不可能查看所有的结果。7）限制资源导致增加风险：在样本规模减少以后，很可能出现采样错误。多样化的样本发现的问题可能会超过大样本发现的问题（同样地，与扩大测试团队相比，让测试团队成员多样化会发现更多的问题）。第4章 测试和除错的区别“ 正确的定义可以防止或结束争论 ”测试常常有个坏名声，原因在于大家常常混淆了测试到底是什么和大家以为测试能做什么。如果混淆了测试的不同的区别会导致冲突、抱怨和项目失败。以下是测试所包括的活动：1）通过测试来发现2）查明问题3）定位缺陷4）确定重要性5）修改6）解决问题7）通过测试来学习8）以上7个方面的活动互相切换测试会随着机构的成长而发生变化，作为测试人员也会在不同的任务之间切换，甚至会被安排去做杂工。因此在这样角色混淆的机构下，建立合理的测试以便修复的过程是非常困难的事情。因为有一点很重要：测试的每个活动是不一样的，它们

是不同的过程。当机构中的人没有就测试人员应该将问题查明到何种程度达成一致，冲突就不可避免了。这是一件很可怕的事情，将这些工作混为一团带来的唯一效果就是让测试过程看起来没完没了。所以如果不进行明确分工，就无法对整个项目加以改善，提高项目成功的可能性。有一条试探法简明扼要地说明了、确定谁应花多长时间做哪项工作的原则：项目中没人可以随意浪费任何人的时间。这里要注意两个方面，首先时间限制原则是试探性的，而不是严格的原则，可以通过很多因素来调整试探结果；其次是这个原则的一个主要应用就是是不是要浪费时间来做出非常细微的区分，不要浪费时间争论特定的活动的精确定义，而是让人们都投入进来共同完成工作。小结在引人注目的“ 测试 ”标题下往往囊括了很多需要不同技能的工作。这样笼统的做法影响了计划、估算和工作分配，甚至会对整个项目造成损害。常见错误1）认为可以为定位错误做出时间计划：我们要知道需要多少时间来定位缺陷的唯一方法是先知道这些缺陷在哪里。可是如果我们已经知道它们在哪里了，就根本不需要定位它们。2）未考虑任务切换导致的时间损失：任务切换可能是有益的，但是也是有代价的。一般情况下如果要切换的任务数目达到了5项，可能就无法完成任何工作。大多数人面对这种情况的时候会简单地完全放弃某些工作，而这是危险的做法。3）将测试当做可以被任何原因打断的低优先级工作：如果要进行可靠的测试，就需要集中精力。4）要求测试人员查明每个故障：如果一件事情不是测试人员的职责，那就最好不要安排，这是在长期工作中最有效的做法。5）要求测试人员定位每个问题：这样的要求是开发人员所需要的相应技能。一般而言测试人员不具备这些技能，虽然有些时候他们可能会提供一些有益的线索。6）修改而不重测：匆忙完成的修改很可能让情况更糟。7）忽视交叉连接关系：由于程序员的活动会驱动对测试的需要，所以测试和编程是绑在一起的。8）对可测性不重视：为了可以测试而设计和构建的代码可以显著降低测试的各个方面所需要的时间和精力。9）坚持要求所有缺陷都是“ 可重现 ”的：对间歇出现的缺陷要投入大量精力来追踪，而不是把它作为推迟测试和修改的借口；使用已经获得大量的信息，而不是用无理的要求来浪费测试人员的时间。10）混淆了测试和“ 建立和执行测试用例 ”：测试人员的大部分工作是不能仅通过确定的测试用例来包含的。应该考虑转换成按照测试活动来进行思考。11）要求对公司的开发过程进行大调整：如果因为受到不公正的对待，想用鲁莽的、反叛的态度来对抗你的上级，这是愚蠢的想法，即使你想让你更受尊敬，也无法用无礼的要求来改变环境，为此你应该去尝试一些别的方法。第5章 元测试“ 观察可以有很多收获 ”总会存在一些有关产品质量的其他信息就在那样摆在周围，只需要加以收集就可以了。不过只有那些注意观察，而且认识到这些信息的相关性的经理才能做到。当局者迷，旁观者清——由于来自开发机构之外的顾问们看待其客户的问题时的心理是不一样的，他们常常可以看到开发方未能察觉的一些信息。嘲笑别人无法看到两个“ 问题 ”之间的联系是很容易的，不过这种“ 失明 ”在那些对某种情况涉入太深，或者投入了太多感情的人身上是很典型的现象。所以学会识别那些免费的信息是对测试进行成功管理的秘诀之一。以下是我们才展开正式测试之前可以发现的一些问题：1）我们有说明书，但是找不到了2）我们的错误太多了，导致缺陷数据库无法高校运转3）我们没找到任何缺陷，实际上我们并没有真正地找4）我们修改记录让缺陷看起来没那么严重5）这不是我的组件中得问题，所以我不记录6）我不知道在测试错误的应用程序7）我们不测量最差的组件，因为花地时间太长8）我们发现了这么多缺陷，不会还有更多的9）我们的测试证明程序是正确的10）我们运行了很多测试用例，根本就看不过来11）如果我们的软件在有3名用户时良好，显然它在有一百名用户时也不会有问题12）我们不希望测试人员知道我们将忽略他们提供的信息13）我没有报告缺陷，所以开发人员不会对我发脾气14）我们不需要测试它，因为开发人员非常有水平我们在展开真正的测试之前，可以对以上的“ 元信息 ”进行归纳整理，它们能够告诉你有关准确度、相关性、全面性等信息。当然上面的那些问题只是让你来理解的一些线索，而不是你一下子就想到的怀疑的问题。也许在这些表现下面都有某个并非显而易见的解释需要去探究。使用上面提供的线索来验证你的直觉，或者找出另外的解释。而且，即使你的直觉是正确的，要让别人感到信服也不是件容易的事。你可能需要收集其他的证据，或者找到其他的支持者，来影响可以对情况加以改变的人。而如果在你看来解决方案显而易见而问题却始终存在，除非你能找出他们如何让那些在你看来非常明显的原因合理化的，否则往往会很难说服他们解决这个问题。小结如果学会了如何使用元信息，也就是有关信息质量的信息，就可以显著提高测试的功效并降低成本。常见错误1）认为测试报告中包括了所有的相关信息：即使对所有的测试都进行了报告，里面最多也只包含了一半你所需要的信息。2）认为测试可以端坐在办公室中就知道测试进行得如何：应该对测试的执行过程及其所隐含情况进行直接观察，从而验证测试报告。3）认为测试可以“ 证实 ”某些事情是正确的：测试只能显示出某些事情失败了，或者是在特定的条件下没有失败

（我们的行业不是关于证实，而是关于证据和推论的）。4）认为只是文档的存在就具有一定的价值：文档不使用情况下没有任何价值，相反如果使用了导向错误的文档，就会比没有价值更糟5）允许待评估/修复/分配的缺陷清单增长到超过人的理解能力：如果处理速度未能跟上缺陷的增长速度，士气就会受到影响，导致进度进一步的落后。最好停止制造新的缺陷转而清除已有的缺陷，所有人都会感觉更好而且工作更好。如果必要的话，可以对缺陷进行一次“特赦”6）责备他人导致他们感到的隐藏缺陷的动机：缺陷在出了办公室之后才成为错误。每当有人伪造记录的时候，都会导致损失有助于对项目进行管理的信息7）奖励有这种动机的人：要进行有效的测试，既要集中精力又要有目的。没有目的地集中精力看起来也许不错，但不取得多少成果8）未记录所有发现的故障情况：被捕获的故障具有很高的价值，不应该被忽略。如果未记录这些故障，就很可能会忽略他们。如果忽略了他们，就是在浪费金钱和时间，为了“节省时间”而不记录故障情况会导致事与愿违的效果9）过度记录发现每个缺陷的情况：一方面，记录的代价可能很高；另一方面，提高报告的成本会显著提升测试人员进行自我审查的可能性（即导致不敢提交过多的缺陷报告），所以要想办法降低开销10）让情绪来决定测试和报告的内容：如果不小心的话，就会鼓励出现以强凌弱和相互依赖的情况。软件构件工作本身就已经够困难的了...11）使用虚假的模型对进展进行评估：发现的缺陷越多，将要发现的缺陷就会也越多，而不是反过来。完美的开发人员是不存在的，不管他们纸上谈兵有多厉害12）假定最会可靠而且正确地遵循正式的过程说明：过程说明只是一个理想过程的说明，而不是对实际做法的说明。在没有任何系统可以保证遵循了过程说明的情况下如此假设，是一种愚蠢的做法13）相信客观性：所有的信息都是理解得出的，所以都存在一定程度的主观性。要留意影响到你所使用的信息质量的那些因素14）未能对使用模板产生的所有文档进行认真地审查：模板保证了文档的形式是标准的，这样的文档通常看起来很漂亮，但是却常常会掩饰一些不可靠的信息第6章 信息免疫“错误不会因为反复传播而变成事实，事实不会因为没有人看见而变成错误”虽然测试的目的是提供信息，但大家常常会将这些信息看成某种威胁。由于信息会带来这么多的威胁，所以我们都产生了一种“免疫系统”，在我们面对那些不想听到的信息时跳出来保护自己。信息免疫会破坏你为测试作出的最好的努力，因为有关缺陷的消息会变成对牛弹琴。我们在生存规则受到威胁的时候会感到害怕：在我们自尊程度比较低而某些交互触发了生存规则的时候，我们就会采取防卫措施。因为如果生存规则被打破，会导致我们队自身安全产生强烈的恐惧感。测试极为容易触及这样的生存规则。经常能碰到很多生存规则：“我一定要保持完美”、“我一点也不笨”、“我必须按照进度工作”、“我必须实现承诺”、“如果不能说好听的，就什么都不要说”、“如果我不能找到问题，就不是在完成工作”、“不要让任何人生气”。以下是心理学家所作的防卫措施的6项分类：1）我们压抑无法接受的事物：压抑就是不承认或者忽略我们认为无法接受的想法、感觉和记忆。结果就好比鸵鸟吧自己的脑袋埋到沙子里面：“如果我看不见，那就不存在。”2）我们让不可接受的事物合理化：合理化就是让没有意义的、愚蠢的或者是无理的举动看上去是合理的。当然，如果你对于此无力解决，也不必感到惊讶，通常，逻辑没能实现的事，合乎逻辑的辩论也无法改变现状。3）我们将自己的负面品质投射给其他人：负面的投射，就是批评其他人具有我们自己身上也有的某种不希望有的品质。4）我们转移指责从而免除自己的责任：转移就是指责并非我们为题的真正来源的人或事，从而免除我们自己的责任。但实际上，在任何情况下，任何人无法控制“聪明”人能学会什么。5）我们对自己的不足进行过度补偿：过度补偿就是为了弥补某些真实的或者是想象的个人不足而做的过了头。6）我们在觉得失去控制时开始出现强迫：强迫就是无法摆脱某种负面的行为模式。小结信息是中性的，但是人们对信息做出的反应很少会是中性的。要对测试信息进行评估，就必须考虑人们的情绪防卫措施——压力、合理化、投射、转移、过度补偿和强迫。保持警醒、深思熟虑和注重实效有助于消除情绪上的混乱，避免不合逻辑的过程对测试造成的破坏。常见错误1）在人们感到害怕时未能发现：如果注意观察，很容易看出别人采取的防卫机制。2）制造令人害怕的环境：如果因为报信的人带来了你不想听到的消息而指责他，结果就是再也无法听到你应该听到的信息了。3）做出决定时受恐惧的影响超过了事实的影响：要取得成功，就要注意自己的生存规则，以及它们如何影响到你对信息的摄入。4）做出决定时受希望的影响超过了事实的影响：对某些事寄予希望没什么问题，但是希望某事发生并不等于指望某事会发生。5）沉迷于强迫行为：一个人的强迫行为往往会导致其他人产生恐惧和防卫的行为，最优先的做法应该是努力让自己是合乎道理的。6）认为任何对你的观点的争论都是某种过度敏感的做法：与合理化行为可能像是一个好的论据一样，一个好的论据也可能听起来像是在合理化，大部分人不喜欢进行争论。要学着带着赞赏的态度听取异议和辩解，试着在反对观点中寻早和尊重那些有价值的内容，实

际上总会有某些内容是有价值的。7）完全拒绝：一个人在拒绝承认事实的时候能够达到极为夸张的程度...8）认为在这里不可能发生：防卫行为是人的反应，来自任何地方都可能发生。第7章 如何应对防卫反应“ 没有哪种感情会像恐惧那样剥夺大脑进行行动和推理的能力 ”人们有时会变得具有防卫性，尤其是当他们觉得被不公正地指责为具有防卫性的时候尤其如此。事实上，指责具有防卫性是在讨论中让某人的观点边缘化的常用方法。不要玩这样的花样。这样做会破坏信息。不要将对方的行为定义为防卫性的，但是按照它是防卫性的反应来对待，看看它是否会在一些温和的沟通中下表现出来。以下是应对防卫性反应的一般步骤：1）确定恐惧：防卫性反应是受恐惧驱动的，虽然隐含其下的恐惧通常无法被看见，但是确实存在。藏式一下能否确定对方害怕的是什么，然后看看在找到方法减轻那种恐惧之后会怎么样。要注意的是防卫性反应会让人们对令人不快的信息免疫。2）使用危机思维：某些防卫反应是无效的论点，在不同的场合要根据现状来解决问题，比如通过逻辑能够知道论点是否是基于逻辑的，通过观察对方一段时间的行为，你就可以知道他除了别的问题，对他而言某种可怕的问题。留意他的行为模式，最终可以发现他恐惧的根源。如果没有发现，最后也可以确定对方的价值是否能够和他经常性的、强迫性的指责别人带来的问题相比。然后可以采取相应的措施。3）实践实践再实践：通过足够的实践，将可以更好地辨识别人的防卫反应并加以解决。某些防卫性反应有效的原因在于它们都反映了一定的事实，在某些情况下它们是合理的，而不是合理化反应。如果它们没有任何合理性，就很容易被摒弃。4）对自己进行测试：看看自己处于对方对着你叫嚷的场合，你会如何反应？是否能够保持冷静并且做很好的处理？小结如果你有正确的起步，并且保持警醒、深思熟虑和主动，就可以避免情绪的混乱支配你的生活工作。常见错误1）没有考虑差异：每个人都有自己的规则，每个人在自己的规则受到威胁的时候都会感到恐惧。但是规则是不同的，做出的反应也是不同的。要平等地对待每一个人，但并不一定要采用完全相同的方法。2）对别人说他们不关心质量：每个人都是相当关心质量的，虽然他们也许不了解如何获得高质量。或者，也许他们知道如何获得与他们对质量的看法一致、但与你看法并不一致的质量。不要侮辱他们而是要教育他们。尤其是要让他们认识到不同的人对质量会有不同的、但是同样有根据的理解，并教育他们在这一特定的场合下起作用的是哪种理解。3）思维不投入：如果你认为自己所处的情况非常紧急，根本不能接受要对错误进行考虑，那么最好马上就辞职。4）对自己过于苛求：需要通过实践来学习如何辨识和有效地应对防卫反应。如果自己除了一些错误，不妨让自己稍微放松一下。5）对自己不够严格：如果你是一名经理，处理其他人可能影响他们工作完成情况的反应就是你的职责。如果你不改善自己这一技能，你就没有做好工作。是什么在阻碍你这样做？第8章 良好测试的要素“ 没什么事物是好的或者坏的，而是思维让事实有了好坏之分 ”与其从技术上讨论良好的测试要素，不如从管理的角度来看待这个问题，事实上这样更有效：如何才能知道测试是否进行得很好？你对测试结果又能够有多少信任？1）永远无法确切地知道：“ 良好 ”并不是属于某个测试的属性，而只会是用来描述某个测试与某个实现之间的特定关系的属性。我们永远无法通过孤立地看某个测试来指导一个测试是否是良好的——胆识确实有很多方法可以得知某个测试是否很可能是糟糕的（比如元测试）2）只能根据事实来评估良好性：缺陷多少你永远无法确定，因为缺陷可能在很长时间后才从产品中暴露出来，甚至永远不会暴露出来。不过，除了通过等待体验结果的积累，你可以预测结果，也可以随机测试，还可以进行并行比较。3）可能希望故意插入一些缺陷：要注意的是，即使所有已知缺陷都被找到了，也并不能提供更多可靠的信息4）对良好性的估算总是统计性的：对良好性的估算只是一个估算。我们只能从统计意义上估算良好性，因为我们永远无法确定地知道特定系统中有或者曾经有多少缺陷。除了依靠独立专家的第二方或第三方一件。大部分经理还可以通过回来下列类型的问题来自行对非差性（non-badness）进行很多评估：1）测试是否名义上能够提供需要的信息？2）是否进行了文档记录？如果没有的话，你是否亲子观察了测试过程，还是由某个你信任的人进行得观察、报告或是执行？3）它是否是诚实的？4）是否理解它？如果不行的话，你怎么可能知道它到底是好还是坏？5）它是否至少覆盖了那些最重要的部分？6）是否确实完成了？是否有办法知道到底完成了那些工作？7）是否可以看出测试和演示之间的区别？（演示只是让系统看起来很不错，而测试应该能表现出系统的真实样貌）8）对倾向和状态的报告是否过于简单化和定期化了？（会导致表面化，或者可能会遗漏某些重要的信息）9）不同类型的测试活动之间是否有不一致的地方？10）你可以明察秋毫么？小结永远无法确切地知道测试是否完成得很好；但是如果测试完成得不好，有很多方法可以知道或估算出来。常见错误1）未考虑到底在寻找哪些信息：测试在确实考虑了寻找哪些信息的情况下已经够困难的了，如果根本不考虑要寻找哪些信息，基本上就不可能进行了。所以，需要考虑最终的测试目标，并思考如何得知自己

将会需要哪些其他的信息。2）根据发现了多少缺陷来衡量测试人员的好坏：测试人员会对这样的衡量方法做出反应，不过恐怕不是按照你所期望的方式。发现的缺陷数量会增加，但是获得信息的质量会降低。3）认为可以确定一个测试有多好：在对测试的准确度和实用性进行评估时应保持警惕和怀疑的态度。否则的话，就会被自然的本性所惩罚，完美主义是不会有好结果的。4）未考虑上下文环境：即使有这样的可能，测试也很少会在所有的情况下都是同等重要的。如果没有考虑可能的使用模式，测试就不会有多少效果。5）在不了解产品内部结构的情况下进行测试：了解正在测试的软件结构有助于确定要尝试的特殊案例、细节特性和重要范围，所有这些都有助于缩小存在于软件能够做什么和它在实际使用中会做什么之间的推理缺口。6）测试时过于了解产品的内部结构：一般用户通常对系统没什么了解，因而不会容忍某些缺陷。所以某些测试最好能够模拟那些初级用户可能采用的行动方式。7）给出对缺陷的统计估算时将它们当做固定的、确定的数字：在陈述对缺陷的估算数据是总是应该给出一个范围，更好的方法是给出统计分布图或者图形。8）未对测试的“不良性”进行度量：使用一个检查清单，提出如上的10个问题，或者其他问题9）不能保证开发过程良好地完成：对低劣地代码测试得再好又有什么用呢？10）未考虑由于发现大量缺陷导致的测试效率的损失：发现大量缺陷可能会让测试人员看起来工作得不错，但是会减缓测试的进度，降低测试的覆盖范围，或者同时导致这两种后果。第9章 有关测试的主要误区“专家就是避免了小错误而落入重大误区的人”也许有无数种方法来把测试进行得很糟糕，其中有一些非常严重的错误的想法会毁掉任何一个项目。以下是一些可能出现的误区：1）指责误区：花在寻找用来指责其他人上面的时间和精力越多，解决该问题的可能性就越低。2）穷举测试误区：请记住，不可能对所有可能性都进行测试3）测试产生质量误区：如果不能做测试以外的事情，而且只允许有限制的测试，这实际上是在阻止提高质量4）分解误区：整体并不是其组成部分的简单叠加，测试了部件不代表就是测试了整个系统5）合成误区：反之亦然，测试了系统也不代表对系统的各个部件进行了测试，系统测试对模块的运行方式与我们队模块进行真正测试时的运行方式是完全不一样的6）所有测试都相同误区：测试的本质实际上是利用不完整的信息进行决策（全息栅格理论），所以必须要在需求、意图、实现三者之间两两做测试，不同类型的测试也就是对不同角度去观察项目，这样才能得到更加确切的信息。7）随便哪个笨蛋都可以测试误区：良好的测试总是探索性的，先前测试的结果会对将来的测试产生很大的影响。这一过程是调查性的，所以不仅要明白测试背后的动机，还要根据有关产品和测试的最新信息来充实实际上进行的测试。小结学会识别一些有关测试的主要误区可以消除项目经理们大约一半的常犯的错误。常见错误1）认为指责可以起长期作用：也许可以看到指责带来的短期效果，但是它带来的后果可能不是有益的，而会像是拿根棍子去捅一条狗一样，反倒被咬一口2）认为对问题的第一印象总是正确的：第一印象是重要的，但是测试问题通常要求进行更多的分析，尤其是如果你发现自己正在指责别人。3）认为可以对任何事物进行“穷举”测试：如果要求“穷举测试”，通常得到的就是测试人员以不同方式进行欺骗，对他们的经理进行欺瞒，制止最终的反叛4）认为可以采用“投机取巧”的方法开发软件，然后通过测试提高质量：投机取巧就是投机取巧，它是低质的，而且会很难测试5）以为系统测试可以捕获所有缺陷而将单元测试当做冗余的加以忽略：系统测试不仅不能捕获所有缺陷，反而会花更长的时间和更多的成本，远远超过通过忽略有效的单元测试所能够节约的6）以为单元测试可以捕获所有缺陷，而认为系统式冗余的所以忽略：你会因为一家飞机保证它所有的部件在组装前都进行了测试而乘坐它的处女航吗？7）期望测试可以产生质量：质量是整个开发过程的产物。不良的测试会导致不良的质量，但是良好的测试并不一定能导致良好的质量，除非整个开发过程的其他部分都是恰当的并且得到了正确的执行第10章 测试不仅仅是敲击键盘测试的真正工作是那些脑力活动。如果你敲击键盘的时候没有思考，你就不是在进行测试。而且，如果你在思考但是没有敲击键盘，你还是有可能在进行测试，具体来说有以下几点：1）毫无目的地敲击键盘不是测试2）白手套测试/覆盖测试（查看项目原始文档，与目前进展进行比较）：拿结果和定义对比一下3）狗食测试（在你喂狗以前，先自己尝一下狗食）：软件开发人员应该通过使用自己的产品来对它进行测试4）对测试人员也要进行测试5）可以在没有意识到的情况下进行测试（比如吃饭的时候先吃一小口来测试是不是好吃）6）演示不是测试（演示只是测试用户是否容易上当）小结一种行为要成为测试，那么不管它是否包括了敲击键盘，都需要寻找会对行为产生影响的信息。常见错误1）认为计算机会读心术：计算机只会做你要它做的事，而不会管这是不是你真正想做的事2）未能验证软件销售宣传：仅仅敲击键盘并不能完成这项工作，虽然如果在敲击键盘时经常遇到崩溃的话也许会有帮助。最好是针对特定的宣传内容建立测试计划3）未能在测试中使用覆盖范围测试工具（例如白手套测试）：可以构建或者安装某些工具

来得知系统的哪些部分从来没有被任何测试接触过。这些不会也许不会积累灰尘，但是如果没有任何测试接触过它们，显然不能认为它们已经测试过了4）认为覆盖测试可以证明某些内容已经测试过了：只是表明代码的所有部分都被某些测试接触过并不意味着这些部分都被彻底地测试过了。代码覆盖范围同样不能表明所有功能都被彻底测试过了。需要分析测试的相关性和全面性，才能确定是否已经进行了彻底的测试，也就是说，要进行思考5）混淆过程文档和测试过程：过程是你实际上做的事，而过程文档则是某人希望你做的事情，是一种理想状况。过程和过程文档很少会完全一致，甚至有时没有一点相同的地方。我们应该将宝贵的时间用于对过程，也就是对人们实际上做的事情进行观察。可以利用节约下来的时间决定最好是将少数的哪几个过程用准确地文档备份下来。6）混淆文档和事实：上面是一般情况，而特殊情况就是混淆测试脚本和测试，混淆测试报告和测试，混淆需求文档和需求等7）未能“吃自己的狗食”：如果你对自己的产品都感到担心或者鄙视，别人又为什么要买下它？8）在狗食测试中只使用不具代表性的“狗”：除非是在开发软件或开发工具，否则完全由软件开发人员构成的样本不太可能代表整个用户群9）未对测试人员进行测试，或者对他们测试过多：对每个人的工作都要进行评估，但并不是不停地评估。某些时候，需要相信人们能够在没有上级站在边上监督的情况下完成工作10）假装演示就是测试：你既可能是做出这种行为的人，也可能是接受这种行为的人。没法说哪种做法更糟——欺骗别人或者欺骗自己。第11章 信息摄取“即使是耳语，大声也无法和清晰的声音相比”只要还记得测试是一项信息收集活动，你就会知道测试就是一项纯粹的沟通活动，而毫无疑问的是，沟通问题往往是最难最有挑战性的问题。在Virginia Satir交互模型里，沟通可被解析为以下4步骤：摄取、确定含义、确定重要性、做出反应。以下是一些有关信息摄取要注意的地方：1）人们听取信息时是有选择性的：多采取书面形式给出确切定义2）数据来源会影响到摄取：搜先要检讨你自己，然后选择合适的人去传达你想说的话3）时机也会导致差异：人们在将注意力投向其他地方的时候也会遗漏某些信息。所以这时候如果没有得到别人注意，就没有什么必要提供信息，因为对方不太可能接受这些信息。4）人们会出现信息过载：划分优先级，定时清理收件箱5）减少测试数量也许可以传递更多的信息6）寻找测试之外的信息摄取7）不要混淆理解和摄取8）使用数据质疑来过滤理解小结摄取是一个主动的过程，要尽可能地了解那些限制摄取的因素、信息的来源，以及数据如何获得了带有偏差的含义。被动地等待别人将数据给你并不会成为受害者，但至少会让他们可以潜在的控制你将会得到哪些错误。常见错误1）未考虑到底希望得到哪些信息：不要只是来了什么信息就接受什么，你可以做出选择2）没有积极地探求希望得到的信息：测试工作得到的信息中大部分是没有用的，但是只要有一双慧眼，仍然可以从中获得大量的有用信息3）混淆了摄取和含义：在有人确定数据的含义之前，数据是没有意义的。对相同的数据，不同的人会给出不同的意义。先收集数据，然后坐下来考虑这些数据可能具有至少三种含义4）禁止测试人员在某些地方寻找缺陷：这些禁地中可能有大量值得提取的数据5）没有为测试准备充足的设备和工具：工具不一定是商业软件包，而应该是提供一个像医生看病那样所需要的诊断环境6）出现金象综合症：如果大象是大而无用的，那么不管是白象还是金象，都是一样的。当然因为它很昂贵，购买它的人肯定不希望浪费，被看成傻瓜——这样做的后果就是更大的浪费和成为更大的傻瓜第12章 确定含义“犹豫不决是成功的首要因素。因为鉴于只有一种办法：什么都不做，但是有很多方法来做这些事。而可以确定的是只有一种方法是正确的，所以与不断前进的人相比，由于犹豫不决而原地不动的人误入歧途的可能性要小得多”没有错误的程序，只有不同的程序：如果对测试数据结果不能想出至少三种可能的解释，就说明思考得还不够。对数据进行解释以确定含义需要勤奋和开放的态度，不要错误地认为更华丽的报告会好一些。如果你认为你有一个报告系统不需要通过阅读、分析和解释就能给数据赋予含义，那就意味着你是在受到这个报告系统的操纵。不管测试结果是以什么样的方式交给你的，都要当心被它们表面上的客观性所误导。在进行解释之前先弄清楚期望的是什么，反之如果你对方不清楚期望是什么，就永远不能确定地说他是错误的。而在你给报告赋予含义之前，先要弄清楚你期望的是什么，否则所有的说法看起来或者听上去都会使对的（或者全都是在胡扯）。而如果在解释结果的时候不清楚产品被期望要做什么，可以通过提出下列问题入手：1）它是否完成了重要的人希望它完成的工作？2）它是否没有做重要的人不希望它做的事？3）这些测试结果告诉了我哪些与这两个问题相关的情况？如果不清楚重要的人希望它完成什么工作该怎么办？在那种情况下，可以根据你认为他们希望完成什么以及你发现的情况来写报告，并列出两者之间的差异。如果这样做有益的话，还可以给出一些他们可以希望得到什么的建议，并说明你是如何得出这些想法的。不过要注意，不要用你的想法过度地影响到他们，否则的话，他们也许表面上会同意，但是在你完成测试的时候却会说：“不，那不是我想要的。”

我们可以通过不给出任何含义来处理模糊性。一般而言，确定报告的含义时，在盲目地要求更多的信息前，至少应该从已经获得的信息、间接信息、未获得的信息（缺失信息也是一种信息）入手。这里也要注意，同样的话可能具有不同的含义，所以不要让我们的话变成别人指责的理由。至少，我们希望他们做的是单纯地找出改进的可能，所以一定要注意检查避免导致其他人进入防卫状态。而且，不光是话语，还有文字、数字、图片以及其他水边什么东西，我们要对表面相同但实际来自不同来源的事物进行理解时都要保持怀疑的态度。充满矛盾的是，在试图对含义进行沟通的时候，有时候使用模糊而不是非常精确的语言会更有效。其原因在于，人们常常会直接略过确定含义的机会去赋予含义。这种对无意识状态下赋予的含义的瞬时感性反应会转而组织他们听取你意图表达的含义（千万不要说对方是错的）。如果希望在信息可能有不同含义的情况下进行有效的沟通，就要考虑信息接收者的心理。尽量使用他们的术语，并注意任何口头反应的语气，即使你认为那个人不可理喻也要如此。小结数据本身并不会说话，它们也不是没有任何模糊含义的。需要由人来给他们接收的数据加上含义，而每个人的做法都不一样。最好记住对同样的数据可以有很多种可能的解释。　常见错误1）直接得出有关数据含义的结论：在采取行动之前至少考虑3中可能的含义。2）运行测试时没有提前记录预期的结果：从心理角度来看，在产品“看起来没问题”时宣称测试成功而产品没有缺陷实在是太诱人了。在测试之前，先记录好对测试结果的预期，不过也要准备好接受“惊喜”。3）提前过度记录预期的结果：对前一点很容易做得过了头，这样会分散对其他更有价值的预期结果的注意。可以列出一份这样的预期结果供所有的测试使用。4）试图完全只靠自己确定含义：不同的思维会确定不同的含义。让整个团队都参与进来，从而降低忽略某个重要结论的可能性。5）认为意义完全是由含义决定的：不同的思维会确定不同的重要性。第13章 确定重要性“艺术的目标不仅是展示事物的外观，还要展示它们的内在意义”重要性就是由负责决定要如何对缺陷进行处理的人赋予该缺陷的价值。而在具体现实里面有以下的一些情况：1）不同的人会给同样的信息赋予不同的重要性；2）公共的重要性也许对每个人也不一样；（如果想对重要性做出明智的估算，就要想办法消除所有个人打算，包括你自己）3）重要性依赖于上下文环境；4）不能总是根据金钱来确定重要性；（人们实际上一直在用价格衡量人的生命）5）不要采用过细的尺度；（人的生命价值总是主观的，实际上，所有对重要性的度量都是主观的）6）首先解决重要问题；7）听从自己的情绪反应；（重要性的实质是你赋予的结果的含义带来的情绪影响）小结我们的情绪承载了关于事情很多重要的信息。如果我们注意情绪，认真听取，先解决重要的事再解决不重要的事，就可以对获得的数据做出最好的处理。常见错误1）混淆重要性和修改的难度：不要将缺陷的类型和缺陷的重要性混为一谈。有可能一个系统崩溃的问题是不重要的，而某个拼写错误确是灾难性的。2）错误判断做出反应速度的重要性：如果测试结果或者某个问题的解决方案来得比我们预期的快，我们会“怀疑”它们是否是正确的。相应地，如果为了得到答案花掉的时间和我们预期的差不多，我们就“知道”这个问题被仔细地、正确地解决了。3）没有认识到重要性是政治性的：进监狱是不能单纯用金钱来衡量的。4）认为有“理性的”或者客观的方法来评估重要性：也许可以使用数字和算法来为评估重要性提供帮助，但是最终的评估步骤总是要度量人对那些数字的情绪反应。5）在获得新信息后没有重新评估重要性：重要性是整个系统的一种性质，包括用于构建该系统的系统或过程。要经常回顾对重要性做出的评估，并做好改变它的准备。6）让废话影响到对重要性的评估：不管我们是有意的还是无意使用的，某些词语会掩盖有意义的信息。7）忽略了采取行动对项目团队本身的重要性：没有人会希望自己的工作被忽视。要尽可能区分开忽视某些缺陷和有意识地选择不修复某些缺陷的行为。第14章 做出反应“有丰富的学识和技能，受过良好的训练，说优雅的语言；这就是好运气”对缺陷的正确反应应该很简单：发现它们；评估它们；修复它们。在项目早期我们可以这样做，但是随着发现的缺陷越来越多，我们会没有足够的时间来做出正确的反应，以下是可能的原因：1）管理不善：长期来看，不存在运气不好的时候，有的只是居安思危而已。2）项目最后会赶进度：软件出现的时间很短，没有人能了解它的本质。3）接近项目结束时还有很多事没有做。4）对测试所需时间的估算与现实差距很大。好天气估算（认为一切都会按照计划进行）、不切实际的过程模型、低质的过程数据、没有过程数据。5）错过了可以有所改变的时刻。小结如果一个项目到测试之前都没有获得良好的管理，大部分好的反应都将无法使用。在很多情况下，除了重新开始，从一开始就采取正确的做法以外，没有哪种反应可以挽救这样项目。常见错误1）依赖运气：运气总是宠爱管理良好的项目。2）为了赶上进度而减少分配给测试的和i建和资源：如果你不关心质量，就可以赶上任何进度，但是那样的话为什么还要测试？3）未能在测试提供了有关产品实际状态的信息后调整进度和估算：没有人能很好地预测未来，足以做出绝对的承诺。任何可以

对未来做出承诺的人一定是某种魔鬼。4）未能手机过程数据：错误是犯下的，而不是天生的。对犯错的位置和时间越了解，就越容易发现、查明、定位和修复它们，或者在下一次出现类似情况时避免错误。5）不了解测试何时开始：测试在产生项目概念甚至更早的时候就已经开始了。如果不了解这一点，就根本无法了解测试。6）不要死马当做活马医：如果你发现特定的构建中满是问题，需要大量的修复工作，就不要盲目地继续测试和报告问题，因为广泛的修复工作会让你的测试报告变成一堆废纸。第15章 避免软件测试变得越发困难“ 执着于傻年头的蠢人会成为智者 ”如果想让自己的下一个项目做得比前一个项目好，理解下一个项目为什么实际上会变得比前一个更困难是个不错的开始。如果前一个项目耗费巨大而无所成就，那就更值得坚持——不是坚持完成或者批准哪个项目，而是坚持从中进行学习。从根本上来看，软件测试会变得更困难的原因在于我们变得更有野心。因为随着软件的增大，会造成以下4个方面的问题愈发严重：1）可能出现故障的地方增多（故障数目）；2）难以查明故障的原因（查明花的时间）；3）在不产生副作用的情况下修复一个故障很困难（修复花的时间）；4）每次为了维修测试而关闭在生产使用上得损失严重（损失的机会成本）；对此，我们能在以下4方面做一些努力：1）让系统尽可能小（需求受控）；在软件中能够做任何事并不意味着在同一个系统中应该做所有事。2）让“ 系统 ”模型是可扩展的；更多的问题可能来自将和程序进行交互的人，因为他们是所有系统中最危险的。应该警醒地检查你开发的简单系统是如何与更大的、极其复杂的系统纠缠在一起的。3）增量构建有清晰接口的分立组件（测试先行）；不要一次做所有事，而是在可以支配的时间内完成那些你有可能完成的最重要的工作。这样，即使遇到预期之外的问题（几乎一定会遇到），你也可以按照进度计划交付某些可能会有用的东西，即使还没有最终完成它。4）减少进去产品的缺陷数目（尽早去掉缺陷）；缺陷不是爬进去的，而是放进去的，是由人在任意时间放进去的。既然想办法尽早取出缺陷，那么了解开发过程中何时会阐述缺陷就很重要。小结虽然未获得良好组织和执行的测试肯定会延长这项工作的时间，但是随着产品变得更大更复杂，内在的系统动态特性会让测试和修复花更长的时间。如果能够了解这些动态特性，就有办法在一定程度上抵御它的影响。常见错误1）低估旧有的、修补过的代码的复杂度：这些代码就和排水管道一样错综复杂；2）不允许讨论这些问题，更不允许对其进行度量：任何时候如果听到有人说“ 我们不能讨论那个问题 ”，那么不管正在做什么，都需要停下来讨论这个问题。对于“ 我们不能对它进行度量 ”的说法同样如此。3）没有根据当前经验表明的需要调整过程数据：没有任何一个软件开发或维护项目没有遇到过意外之事。这也是无法预先将测试计划到尽善尽美，而必须采用探索性方法的原因之一。4）将早期结果当做后期结果的指示器：对初始案例至少应知道一件事——它们从定义来看就是和后期案例不一样的。5）将测试人员当做“ 阻止交付的坏人 ”：和其他人一样，有部分测试人员是坏人。但是无论好坏，它们都只是信息的提供者。它们并不做出有关交付的决定——如果他们做决定的话，那他们的经理就是坏人。6）测试人员认为他们自己是“ 质量警察 ”：即使他们是警察，他们也不是法官、陪审团或者立法者。第16章 不使用机器进行测试“ 99%的失败都是来自习惯找借口的人 ”最重要的测试工具不是计算机而是人的头脑——与眼睛、耳朵以及其他感官联系在一起的头脑。无论多少计算能力都无法补偿没有头脑的测试，但是大量的脑力可以补偿缺少或者无法获得的计算机。投入大量脑力对一个系统进行测试的最简单方法是技术评审，大家坐到一起来分析和记录某些技术产品的优劣之处，技术评审实际是“ 收集有关某个产品的信息以便能够将该信息用语某种目的 ”的过程。技术评审最大的优点在于，可以用于那些不能马上通过机器来测试的项目内容。以下是技术评审的步骤：1）即时评审：在某人给出某个工作产品不能或不应被评审的原因时发生的这类评审（注意要常常进行元评审——对评审系统进行评审）；2）对最差部分进行评审可以让人了解缺陷的严重性；3）事实并不总是能令人信服的；4）测试人员是颇有价值的评审者；即使测试人员真的不能在评审过程中找到缺陷（事实并非如此），他们也可以从评审的经历中学到一些东西。评审所能够提供的最大益处是提供知识：1）通过观察开发人员可能产生的存在瑕疵的思维模式，测试人员可以了解如何设定更好的测试；2）通过较早地对规格说明进行评审，测试人员可以更早了解到他们测试计划的范围；3）通过熟悉设计，测试人员可以加快检测缺陷的过程，然后帮助查明它们；4）通过参与评审，测试人员可以学习如何能对他们自己的测试用例、测试计划、测试驱动程序和工具进行更好的评审。而且，测试人员参加评审还有一个额外的好处，就是与那些单纯坐等开发人员交给他们一些东西进行测试的人相比，他们可以更快地进入项目。小结有很多不使用计算机就可以进行测试的方法，但是没有哪种测试可以不使用头脑。尽可能早和尽可能频繁地进行测试，而且要使用能够凝聚起的所有才智。常见错误1）未认识到以技术评审作为测试的补充形式的价值：如果一个项目没有包含了经常性技术评审的过程，那

么无论它的机器测试过程有多好，它都不可能超过平常水平；2）屈服于众多争论之一而略过技术评审（或者为此忽略开发过程的某些部分）：各个系统中那些令人头痛的、容易出错的组件大约有十分之九都绕过或者省略了开发过程中的一个或多个步骤；3）将技术评审当做惩罚：要培养一种气氛，让每次评审成为对所有参与者都有益的经历；4）为了节省时间而省掉评审：错误是导致损失项目时间的首要原因。省掉评审总会让项目花掉更多的时间；5）未评审设计和代码的可测试性：只要在设计系统的时候考虑了它的可测试性，那么在开始运行任何测试之前，就可以节约至少一半的测试成本；6）评审者中未包含测试人员：测试人员可以提供他们独特的视角，提高评审的有效性。测试人员也需要每次评审提供的教育机会。7）没有认识到学习的价值：最终，学习是评审中最有价值的部分。也许你会由于难以将学习所得定量化而没有考虑到它的价值，但是如果你确实想将它定量化，也是可以做到的。第17章 测试欺诈“ 在虚弱的时候很难抵制幼稚的建议..... ”掉进相信测试误区的陷阱是一回事，成为测试欺诈的牺牲品又是另一回事。在两种情况下，我们都因为迫切希望某件事是真实的而相信它是真的。但是在测试欺诈中，我们让自己受到某些人的欺骗，他们由此收益。以下是一些警告标志，这些标志的出现都意味着你在成为测试欺诈的牺牲品：1）我们会卖给你一个神奇的工具：并不是任何工具都一无是处，但是没有任何一种能够完全实现供应商们在销售它们宣传的那些神奇的承诺；2）我们的演示是欺诈：演示并不是测试。演示也许可以用来培养客户，但是它们不是测试，因为演示是专门设计用于证明某个观点的，而且最重要得是为了避免出现意外；3）这么多的证明信表明它一定很好：推销材料中常常会包含一些伪测试结果（例如产品比较、证书或者保证），这些材料表面上含有信息，实际上却只能用于确定某些只要收取费用或某种形式就让别人使用他们的名义的假内行。4）通过定价来欺诈（定价欺诈）：高价欺诈与低价欺诈，注意，销售人员从不说真话；5）我们的工具会读心术：工具只是工具，他们仍然需要人，会思考的人，来进行有意义的操作；6）我们保证你不用做任何事：谨防替换欺诈（用高级人员吸引客户，但是收了钱以后就用廉价的实习生替换），所以，如果某件事好像不是真的，那它很可能就不是真的。“ 你不用做任何事 ”的承诺永远都是一种欺诈。7）我们一起密谋（共谋欺诈）：经理千万不要幻想着能够远距离地控制软件开发，只使用一些数字而不是至少通过亲自的观察来验证这些数字，否则数字的制造者们会合伙投机取巧来欺诈这个愚蠢的经理。工具的秘密在于：良好的工具可以增强有效性。如果测试的有效性是消极的，增加工具只会放大消极作用。而如果需要对工具或者软件的可靠证明材料，就要亲自获得它们。如果供应商不希望你这样做，就要对他当心了。可惜，人们会相信任何事，只要它听起来足够好，购买者请当心自己的愿望。识别欺诈的方法之一是它们总是承诺不劳而获：要拒绝某件看起来不错的事可能很难，而要清除由于接受它而造成的后果则会更难，唯一能够不劳而获只有后悔。小结当你不顾一切地想要清除缺陷，交付产品的时候，就容易落入各种测试欺诈的陷阱，它们都承诺快速无痛的解决方案。常见错误1）单纯依靠数字来管理项目：数字可能会有用，但是只有在通过亲身观察加以验证，并置于有关上下文环境中时才是有用的；2）通过第三方接受证明信：不要依靠那些无法通过直接和递交它的人进行交谈来加以验证的证明信；第18章 忘却型欺诈“ 乐观主义者看见玫瑰花而不是它的刺；悲观主义者盯着它的刺，忘却了玫瑰花 ”有很多欺诈者和贪婪的人准备趁机利用你遇到测试难题时的绝望情绪就够坏的了，但是最常见的欺诈却是无意中由于忘却而自行犯下的。它们通常来自我们无意中忽略信息中那些令人不舒服的部分时的乐观主义，而这些部分带来的痛苦会迫使我们认识到真实情况到底有多糟。1）推迟文档化；2）不明确的测试报告：同一句话对很多不同的人可以带去很多不同的含义，而对测试报告的不同理解可能会带来很高的成本；3）伪造的测试报告（阻止了改进）：人们确实会成长，而建立一个利于成长的环境总是一种好的管理方法。只是要注意，你需要了解让你偏离一个成功过程的所有真实原因，以及这样做可能会发出哪些意外的信息；4）在别的地方进行报复；5）早期的答案可能产生误导；6）“ 量 ”不是“ 质 ”的同义词；7）不要将非测试活动当做测试；8）太整洁了，不可能是真的：真实的数据永远不会那么漂亮；9）电子表格中的垃圾还是垃圾；小结当我们迫切希望所有事都进展顺利的时候，就很容易在我们数据中带上我们的幻想。常见错误1）使用有关已交付产品中错误的早期报告来估算交付的全部错误：在早期，用户还没有探索产品的全部功能，不常使用的特性也许更可能会有缺陷；2）让缺陷报告机制冗长或难以使用：任何妨碍对发现的缺陷立即进行记录的事物都会对测试报告造成扭曲；3）建立一个鼓励伪造测试报告的指责环境：不管收到的消息让你感到多不愉快，永远不应该因为别人带来这条消息而惩罚他们；4）重视形式超过内容：入股报告很整洁，那不错。但是首先也是最重要的是，报告应该是及时和准确的；5）重视量超过质：如果为忙忙碌碌提供奖励，就会有大堆的测试，但不会完成多少真正的测试；尾声“ 美国大

众希望在剧院中看到的是一场有快乐结局的悲剧”如果你属于对任何缺乏完美的事物都无法感到高兴的人，那就不可能给你一个快乐的结局。在这个世界上不会有完美存在，所以对你而言就不会有快乐。毫无疑问，我破门要控制那些在接近项目进度计划结尾时可能会使我们的知识和逻辑不为所措的那些情绪。人生没有结局，人生一场努力超越那些过于人性化弱点的过程，这个过程不会是完美的，但是会快乐。

4、在书店粗略的翻过，就让我放弃了买这本书的想法，双语版的，貌似理论强于实践，而且价钱有点狠。读起来倒是挺轻松的，但是没什么特别惊喜的地方，其实很多时候测试依靠的是领导的决断和公司的发布制度。哎，还是期待那本快上市的《测试之美》吧。

5、断断续续一个多月，今天终于把它读完了，觉得很有收获，有很多很有益的思考，建议所以把软件测试作为职业的朋友读一读！

6、这本书很早就有人推荐我看，如今看到一半觉得写的很好，无论是产品经理、开发人员还是测试人员，都建议读一下。作为一本谈软件测试的书，作者列举的很多现实的例子，有些我们按常理推想是根本不可能发生的例子，但是它们却实实在在的发生了。通过这些例子，以及每一章结尾的常见错误总结，我们定会对软件开发和测试有一些新的认识。1.对待自己开发的产品，应保持专心和好奇。尤其是测试时，更不能讳疾忌医，多一些怀疑的态度总是好的。近期的一些经历也说明，产品经理必须有这种态度，不要过于“爱惜”自己的产品。2.阿楠1年前曾就敏捷开发问我：如何衡量开发的软件是高质量的。我当时的回答是依靠测试体系的完善。现在产品的测试体系比以前好了很多，但只是减少了产品bug的数量。高质量的软件，在满足需求的情况下，最重要的还是在于开发人员的编码；而敏捷开发，成败的关键亦在于开发人员的整体素质。此外，作者从一些心理学的角度看待测试及软件工作者的行为，使读者更能深刻认识事情的本质。再次推荐本书！

《完美软件》

章节试读

1、《完美软件》的笔记-第169页

狗食测试他们编写轿车上的控制刹车和转向系统用的嵌入式软件，在发布软件之前，每个和开发该软件有关的人都要在一条测试道路上以每小时100英里的速度驾驶一辆使用该软件的轿车。

2、《完美软件》的笔记-全文

第1章 进行测试的原因
不可能有完美的软件，而测试也不可能做到覆盖率100%，但测试可以提供降低风险的信息。人不是完美的，因此编出的程序也必定是不完美的。我们只能做到力争完美，测试实际上更像一种心理行为。
经理的职责是做出管理决定，而不要问测试人员诸如“软件可以交付了吗”这类本该由经理自己决定的事。不要相信测试可以改进产品，测试只是发现问题，产品质量和完善归根结底是开发人员的工作。

第2章 测试无法做的事
人们都存在一种感情倾向，不希望发现自己犯了错误。那么，建立一种奖励出现缺陷数量较低的开发人员的制度，并不一定能刺激他们写出更好的代码，肯定会有一些开发人员故意隐瞒错误和增加同测试人员的冲突。
几种常见的错误：
1.不尊重测试人员；
2.过度尊重测试人员；
3.让测试人员当替罪羊
4.忽略通过测试得到的信息，或对测试数据背后隐藏的问题不关心、不清楚原因。
作为经理，首先应对测试有足够的了解，而不允许经理们以不了解测试过程为借口允许缺陷通过。否则应任命专职的测试经理。

第3章 不对所有可能性进行测试的原因
任何测试用例集都是一种采样方法，无论你有多少资源，都要尽可能选择那些具有最强代表性的测试集。如果不得不减少测试资源，那么应该同时减少测试集的规模，否则代价就是测试质量的降低。获取性价比高的采样集是一种经验和直觉，需要经历过较多的实际产品测试才能获得。

第4章 测试和除错的区别
为定位错误做出时间计划是比较难的，但是可以根据经验为一个功能的测试预估一下可能花费的时间，同时测试人员也应该有一个上限时间——如果一个功能花费了太多时间做测试，应该考虑一下性价比的问题了。
任务切换是最浪费时间的。考虑以下场景：4名开发人员，4个产品，均为2周一次迭代（每个产品的起止时间不固定，一般在迭代开始的计划会议和结束前的回归测试是比较耗时的两大块。应该如何分配测试人员？
测试人员的主要工作是发现问题，而定位问题主要是开发人员的工作。对于不易重现的错误，测试人员如果花大量精力去重现是需要投入大量精力的。经理应权衡，比如有些无法复现的bug，可以在以后的迭代中持续跟踪，但不要刻意去测试重现，而是在日常测试中留心是否重现。

第5章 元测试
以下情形有些虽然听起来像是笑话，但它们确实发生在某些组织中：
我们有详尽的测试说明书，但是找不到了，或者找到时发现落满了灰尘；
我们无法进行测试，因为测试系统被40000多条bug记录搞瘫了；

（十天后）我还没有完成这个产品的测试，而是对XP系统进行了全面的测试，因为我发现一个小bug可能是XP系统的原因；
我们运行了60万个测试用例，系统中没有出现崩溃，但其详细的测试结果我们根本就没有时间看；
我们解决了测试人员提交的bug，但我们没有反馈给他们结果；
我没有报告缺陷，所以开发人员不会对我发脾气，我们将继续和平相处下去；

第6章 信息免疫
信息是中性的，但是人们对信息做出的反应很少会是中性的。要对测试信息进行评估，就必须考虑人们的情绪防卫措施：
我们在生存规则收到威胁的时候会感到害怕；
我们压抑无法接受的事物；
我们让不可接受的事物合理化；
我们将自己的负面品质投射给其他人；
我们转移指责从而免除自己的责任；
我们对自己的不足进行过度补偿；
我们在觉得失去控制时开始出现强迫。

第7章 如何应对防卫反应
指责某人具有防卫性是在讨论中让某人的观点边缘化的常用方法。不要玩这样的花样，这样做会破坏信息。
不要将对方的行为定义为防卫性的，但是按照它是防卫性反应来对待，看看它是否会在一些温和的测试下表现出来。
对待不同人的防卫反应，需要因人而异。
不要用对别人说他们不关心质量：要引导和教育他们而不是侮辱他们。

第8章 良好测试的要素
测试是否覆盖了产品最重要的部分。
测试时不了解产品内部结构或过于了解内部结构，不如介于二者之间的效果好。
测试人员的时间应主要用于测试设计和执行，而不要把过多的时间浪费在安装配置、缺陷调查和报告上。
很多谬论和弄虚作假的行为都可能扭曲测试，经理们对待测试应专心和好奇，做到明察秋毫才能改善此情况。
对待自己开发的产品，应保持专心和好奇。尤其是测试时，更不能讳疾忌医，多一些怀疑的态度总是好的。近期的一些经历也说明，产品经理必须有这种态度，不能太“爱惜”自己的产品。

第9章 有关测试的主要误区
指责 误区：某个人花在寻找用来指责的其他人上面的时间和精力越多，解决该问题的可能性就越低。同时，指责很具有传递性。此外，指责也许能带来短期效果，但是它带来的长期后果可能不是有益的。
认为可以采用“投机取巧”的开发软件，然后通过测试提高质量。质量是整个开发过程的产物，不良的测试会导致不良的质量，但是良好的测试并不一定能导致良好的质量，除非整个开发过程的其他部分都是恰当的并且得到了正确的执行。
系统测试不仅不能捕获所有缺陷，反而会花更长的时间和更多的成本，其性价比远远低于单元测试。当然，也不能完全依靠单元测试，必须辅以系统测试。

第10章 测试不仅仅是敲击键盘
未能“吃自己的狗粮”：如果你对自己的产品都感到担心或者鄙视，别人又为什么要买它？同时，也要避免狗食测试中只使用不具代表性的“狗”：除非是在开发软件开发工具，否则完全由软件开发人

员构成的样本不太可能代表整个用户群。
假装演示就是测试：你既可能是做出这种行为的人，也可能是接受这种行为的人。没法说哪种做法更糟——欺骗别人或者欺骗自己。

第11章 信息摄取
萨提亚交互模型将任何沟通都分解成四个主要的阶段：摄取->确定含义->确定重要性->做出反应

以软件交付日期为例，如何避免沟通时双方对信息含义理解的偏差（或者说各自的倾向性）：
1.只采用书面形式给出和接受某个日期；
2.不要给出或接受“点日期”比如“9月1日”，而是只给出和接受以范围形式写出的估算的日期。
如果你发现别人更愿意接受另一个人转述你要传达的信息，不妨想一下自己以前做过什么让大家认为我不是一个可靠的来源？我将来要做些什么来提高他们对我作为信息来源的信任？

如果要对缺陷进行分类，可以遵循以下原则：
（1）只使用四个缺陷报告类别：0～3级
0级的问题会阻碍其他的测试
1级表示如果该问题没有解决，那么产品就不能交付
2级表示如果该问题没有解决，产品的价值就会显著地降低
3级表示如果产品交付时有大量类似的问题，该缺陷才是重要的。
（2）为任何一个缺陷报告分类的时间不能超过一分钟，否则就给他一个临时的“0级”类别并放到一边。（可以使用番茄定时器）
（3）如果时间允许，对临时的“0级”缺陷报告进行重新检查，时间一般每个不能超过5分钟，否则就将其转给更了解到人以获得更多信息。

金象综合征：一头白象也许是一件大而无用的东西，但是如果它是金子做得而且花了很多钱，就会诱惑一些人无论如何都要想办法用它。如果昂贵的测试工具设计有问题、不可靠，或者迫使测试人员只是为了适合该工具而选择本来没有其他原因会选择的方法进行测试，那么这种工具会导致大量的问题。

第12章 确定含义
不同的思维会确定不同的含义，因此最好让整个团队都参与进来；同样，不同的思维会确定不同的重要性。

第15章 避免软件测试变得越发困难
应对测试成本上升的方法：
1.通过合理的控制需求，让系统尽可能的小。开发人员经常犯的一个错误就是试图在软件中处理所有的可能情况。
2.增量构建，测试先行。
3.尽早测试、尽早解决缺陷。

第16章 不使用机器进行测试
投入大量脑力对一个系统进行测试的最简单的方法是技术评审。但不要把技术评审当作惩罚或刻意挑错，而应让每次评审成为对所有参与者都有益的学习经历。
通常，最迅速的评审方法是对“最差的”部分先评审。
测试人员是颇有价值的评审者（和敏捷开发计划会议时需要测试人员参加类似）：
1.通过观察开发人员可能产生的存在瑕疵的思维模式，测试人员可以了解如何设定更好的测试。
2.通过较早的对规格说明进行评审，测试人员可以更早了解到他们测试计划的范围。
3.通过熟悉设计，测试人员可以加快检测缺陷的过程，然后帮助查明他们。

4.通过参与评审，策划死人员可以学习如何能对他们自己的测试用例、测试计划、测试驱动程序和工具进行更好的评审。

第17章 测试欺诈
某些经理幻想着能够远距离地控制软件开发，只使用一些数字而不是至少通过亲自到观察来验证这些数字。当经理采取这样的态度时，他们就让自己很容易成为共谋欺诈的对象。在这种欺诈中，两组人会共同投机取巧。数字可能会有用，但是只有在通过亲身观察加以验证，并置于有关的上下文环境中时才是有用的。
识别欺诈的方法之一就是它们总是承诺不劳而获。

第18章 忘却型欺诈
回归测试不等于新测试：一个回归测试执行的次数越多，对测试产品和测试过程带来的坏处就越多。
计数不等于思考：对测试进行计数可以产生很多灾难性的效果。如，测试人员可能会避免建立任何冗长的或者复杂的测试。他们可能会停止相互帮助进行测试，停止进行其他不会被算作测试用例的有益的测试活动。
当我们迫切希望所有事都进展顺利的时候，就很容易在我们的数据中带上我们的幻想。

版权说明