

软件测试人员的思考问题方式

测试员有很多不同的背景，测试团队是多元化的集体，但是大多数人都同意：测试员的思考方式是不同的。怎么不同？有人说测试员是“消极”思维者。测试员会抱怨这种说法，认为自己喜欢征服，他们在报告坏消息时有一种特别的兴奋感。这是一种普遍观点。我们提出另一种观点。测试员并不抱怨，他们提供的是证据。测试员并不喜欢征服，他们喜欢打破产品没有问题的幻觉。测试员并不喜欢发布坏消息，他们喜欢把客户从虚假信念中解放出来。我们的观点是，按测试员的方式思考意味着实践认识论。测试运用的是认识论，不是靠傲慢或谦卑。

本章旨在把测试员的大脑开发成，经过仔细调谐的推理机器。请记住：要用精神力量做好事，而不做坏事。

经验 1，测试运用的是认识论

读者看到这个题目会说：嘿，回来！我们在这里不是要讨论对电影明星的新崇拜。请相信我们。认识论是能够帮助测试员更好测试的一个哲学分支。

认识论研究如何认识所了解的东西：研究证据和推理。这是科学实践的基础。研究认识论的人有科学家、教育家和哲学家，当然还有精英级的软件测试员。学习认识论的学生研究科学、哲学和心理学，目标是了解怎样才能改进我们的思维。我们使用的术语比经典定义要宽，以便能够更多地利用批评性思维的最新成果。将认识论运用于软件测试，要问与以下类似的问题：

- 怎么知道软件足够好？
- 如果软件并不是足够好，怎样才能知道？
- 怎么知道已经完成了足够的测试？

苏格拉底早在 2400 年前就提倡并描述了对信念的批判性观察，因此我们把他看作是最早的认识论者。直到今天，哲学家、科学家和心理学家都还在继续研究认识论。作为测试员，这就是我们的遗产。

经验 2，研究认识论有助于更好测试

直接与软件测试有关的认识论问题包括：

- 如何收集和评估证据。
- 如何进行有效的推论。
- 如何使用不同逻辑形式。
- 拥有合理的信念意味着什么。
- 形式和非形式推理之间的差别。
- 非形式推理的常见谬误。
- 自然语言的含义与模糊性。
- 如何做出好的决策。

从来也没有研究过这些问题的很多人也能测试得很好，但是如果要做得比很好还好，就要研究这些问题。研究认识论可帮助测试员设计有效的测试策略，更好地意识到自己工作中的错误，理解自己的测试能够证明什么、不能证明什么，并编写出无懈可击的测试报告。

以下是三本具有很高可读性的入门书：

- 《批判性思维的工具：心理学的元思想》(Tools of Critical Thinking: Metathoughts for Psychology)(Levy 1997)。这本书是针对精神病医生写的，但是对测试员也很有用。书中每一章都是有关更好思维的不同思想。不一定把它全读完，可以挑选任何一章阅读。

- 《思考与决策》(Thinking and Deciding)(Baron 1994)。这是讨论思维世界的一本可读性很高的普通教科书,是很好的入门书。
- 《研究的技巧》(The Craft of Research)(Booth、Colomb 和 Williams 1995)。这是一本有关批判性阅读和写作的很好的书籍,包括如何组织有说服力的论据。主要针对大学生读者。

经验 3, 认知心理学是测试的基础

如果说认识论告诉我们的是应该怎样思考,那么认知心理学告诉我们的是我们是怎样思考的。与测试有关的一些问题包括:

- 人的感觉和记忆可靠性。
- 信念从哪里来。
- 信念如何影响人的行为。
- 做出决策所使用的偏见和捷径。
- 如何了解并分享所知道的信息。
- 如何考虑复杂事情。
- 在压力下如何思考。
- 如何识别模式。
- 如何把想法和事物分类。
- 如何注意事物之间的差别。
- 记忆事件中的失真。
- 如何重新构建部分记忆的事件(例如不可再现的程序错误)。

从来也没有研究过这些问题的很多人也能测试得很好,但是如果要做得比很好还好,就要研究这些问题。研究认知心理学有助于理解影响测试员工作成绩的因素,以及影响人们解释自己工作方式的因素。

开始研究认知心理学,不能不看《旷野中的认知》(Cognition in the wild)(Hutchins 1995)。Hutchins 研究海军航海团队,以及他们怎样协同工作。这本书的很多内容也都与软件项目和测试团队有关。

有关思考心理学的一本有用的书是《理论与证据:科学推理的能力的开发》

(Theory and Evidence: The Development of Scientific Reasoning)(Koslowski 1996)。在这本书中, Koslowski 研究了人们如何使用因果关系理论进行系统推理。这可以解释为什么测试不只是查看外部行为,并对照简单的预期描述进行检查。

经验 4, 测试在测试员的头脑中

优秀测试和平庸测试之间的差别在于测试员如何思考:测试员的测试设计选择,解释所观察到的现象的能力,以及非常令人信服地分析描述这些现象的能力。测试的其他工作大部分是一般的办公室工作。如果看到两个测试员并排工作,不一定能看出谁的测试更好。他们工作中能够看得到的部分外表相同,这说明:

- 很多人认为测试很容易,因为可以很容易地模仿优秀测试员的外表看得到的行为,并且他们没有好的测试的其他标准。
- 如果要成为优秀测试员,就要学会像优秀测试员那样思考,而不是模仿他们的行为。
-

经验 5, 测试需要推断,并不只是做输出与预期结果的比较

流行的观点认为,测试员只是执行测试用例,并对照预期结果比较执行结果。这种观点把测试看作是简单的比较活动,没有看到一些聪明人必须设计测试,并确定预期输出。想想

看,测试设计人员几乎从来没有得到过应该测试什么的权威指导,更不要说应该期望什么了。可以得到的指导是要解释的主体。在现实生活中,大多数测试设计都是基于推断,或基于与测试员的推断有关的经验。不仅如此,这些推断还要随时间发生变化。像测试员那样思考,就是要掌握探索式推断的艺术。

探索式推断听起来可能像是奇怪的想法,这意味着要以一种不能事先预测的方式,通过一种思想引出另一种思想,然后再引出下一种思想。有关探索式推断的一本很好的书是《证明与反驳:数学发现的逻辑》(Proofs and Refutations: The Logic of Mathematical Discovery)(Lakatos, 1976)。关于这本书需要注意的是,Lakatos 如何说明数学和科学推理过程是探索式的,而不是脚本化的。甚至数学家也是积极探索地推理,而不是通过运用枯燥的公式。他们像测试员那样思考!

经验 6, 优秀测试员会进行技术性、创造性、批判性和实用性地思考

各种类型的思考都要考虑测试的实施。但是我们认为需要提出四种主要思考:

- 技术性思考。对技术建模并理解因果关系的能力。这包括诸如相关技术事实的知识和使用工具并预测系统行为的能力。
- 创造性思考。产生思想并看到可能性的能力。测试员只能以能够想象得到的方式进行测试,只能寻找猜想会存在的问题。
- 批判性思考。评估思想并进行推断的能力。这包括在自己的思考中发现并消除错误的的能力,将产品观察与质量准则关联起来的能力,以及针对特定信念或所建议的行动过程构建有说服力的测试用例的能力。
- 实用性思考。把想法付诸实施的能力。这种能力包括诸如运用测试工具,并使测试手段和力量与项目范围适应的技能。

总之,像测试员那样思考,会最终导致相信事物可能不像外表看起来那样。不管事物是怎样的,都可能有差别。我们发现,当测试过程以最具破坏性的方式失败时,根本原因最有可能是视野狭窄。换句话说,这不是运行了一万个测试,而本来应该运行一万零一个问题;问题是没有想象出测试的总体大纲,没有做即使有两倍时间和资源也不会做的测试。

经验 7, 黑盒测试并不是基于无知的测试

黑盒测试意味着产品内部知识在测试中不起重要作用。大多数测试员都是黑盒测试员。为了做好黑盒测试,就要了解用户,了解他们的期望和需要,了解技术,了解软件运行环境的配置,了解这个软件要与之交互的其他软件,了解软件必须管理的数据,了解开发过程等等。黑盒测试的优势在于测试员可能与程序员的思考不同,因此有可能预测出程序员所遗漏的风险。

黑盒测试强调有关软件的用户和环境知识,这一点并不是所有人都喜欢的。我们甚至把黑盒测试描述为基于无知的测试,因为测试员自始至终都不了解软件内部代码。我们认为这反映出对测试团队角色的根本误解。我们不反对测试员了解产品的工作原理。测试员对产品了解得越多,了解产品的方式越多,越能够更好地测试它。但是,如果测试员主要关注的是源代码,以及能够从源代码导出的测试,则测试员所做的工作也许就是程序员已经做过的,并且测试员关于这些代码的知识要少于程序员。

经验 8, 测试员不只是游客

测试员对产品做的大量不是测试的事,有助于测试员对产品的了解。测试员可以浏览产品,看看产品由什么组成,怎么工作。这样做有很高价值,但这不能算是测试。测试员和游客之间的差别在于,测试员把精力放在评估产品上,而不只是见证产品。虽然不必事先预测

产品应该表现出的行为，但是试验产品能力的活动还没有成为测试，除非而且直到测试员运用某种如果问题存在就能标识的原理或过程时，这种活动才能成为测试。

经验 9，所有测试都试图回答某些问题

所执行的所有测试，都是要回答有关现实的产品和应该得到的产品之间关系的某个问题。有时测试员完全没有意识到自己在回答问题。如果测试员只是在寻找明显的问题可能还好，但是在很多情况下，问题并不会闪烁着“请报告我”的提示自己跳出来。产品的有些错误行为用户可能一眼就会看出，尽管测试员可能没有注意到。在任何测试活动中，都要问自己什么样的问题应该推动自己评估测试策略，否则就会更像是游客，而不是测试员。

经验 10，所有测试都基于模型

测试员在设计测试时，头脑中可能会有一个想象的图景，也可能有功能清单或某种图表。测试员会有谁的用户、用户关心什么的一些概念。所有这些都是模型。不管模型是什么，测试都主要基于产品模型进行，而不是实际产品。有缺点的模型会产生有缺点的测试。学会一种对产品建模的新方法，就像是学会了观察产品的一种新方法。

要研究建模问题。测试员对建模艺术越精通，越能够更好地测试。有关需求分析和软件体系结构的教科书和课程会有所帮助。获得各种建模技能的一种很好方法就是研究系统的思考。请参阅《通用系统思考引论：25 周年版》(An Introduction to General Systems Thinking : Silver Anniversary Edition)(Weinberg 2001)。

经验 11，直觉是不错的开始，但又是糟糕的结束

测试员很想根据自己的直觉使用具体的测试数据，或判断具体的输出，即测试员自己知道的“本能感觉”，即使说不出来使用这些知识的合理性的理由。我们认为这是有用的感觉，但是只是在开始时更有用，而不是在其他时候。

除了直觉有很强的偏见这个事实之外，真正的问题还在于测试员试图让其他人(例如程序员和经理)认真地对待自己的错误报告和质量评估。除非这种发现是基于大家都有的直觉，否则测试员的工作建议很可能不被采用。

因此，我们建议把直觉用作指南，但不能用作合理性证明。当有“这是问题，因为它显然是问题”的想法时，可考虑换一种方式：“这是问题，因为我观察到产品行为与需求 x 、 Y 和 z 矛盾，而我的客户很看重这些需求。”

经验 12，为了测试，必须探索

为了很好地测试产品，测试员就必须研究它，必须深入它。这是一种探索过程，即使已经有了产品的完美描述。直到通过想象或接触产品本身而探索规格说明之前，所得到的测试都会是肤浅的。即使充分研究了产品，对产品有了很深的了解，仍然要探索问题。因为所有测试都是采样，而且样本永远也不可能完备，探索式思考要在整个测试项目过程中，在寻求最大化测试价值时起作用。

这里所谓的探索，是指有目的的漫游：带着一般使命在某个空间中漫游，但没有预先确定的路线。探索包括不断学习和实践。常常需要返回、重复或在没有经过培训的人看来是浪费的其他过程。也许正因为如此，探索对于测试和对软件工程的重要性，常常没有得到重视，总之受到这个领域的文献作者和顾问的嘲笑。

证明我们关于探索核心重要性的论断超出了本书的主题范围。生动地体验探索的一种方法，就是观察自己在不看印在盒子上的完成图的情况下，如何拼接拼图板，或玩“20 点问题(Twenty Questions)”或“策划(Mastermind)”游戏。请注意，通过预先严格确定的步骤进行，

在这些活动中会怎样遇到更多的困难，得到更少的回报。

经验 13, 探索要求大量思索

探索就是侦查，是没有边界的搜索。可把探索看作是在太空中遨游，需要前向、后向和侧向思索。

- 前向思索。根据已知探索未知，从所看到的探索还没有看到的，注意支流和副作用。例如，看到一个打印菜单项，点击看看会发生什么。
- 后向思索。从怀疑或想象的东西返回到已知，尝试证实或否定自己的推测。例如：怀疑是否有打印这个文档的方法，于是打开菜单并检查是否有打印菜单项(Solow 1990)。
- 侧向思索。让自己的工作由于新冒出的想法而转移，然后再将探索主题返回到主线索上(de Bono 1970)。例如：这个图非常有意思。嘿!我想该打印一些更复杂的图，看看会怎么样。

即使没有要测试的产品，也可以探索。可以使用同样的思索过程探索一组文档，或与程序员面谈。通过构建更丰富、更具想象力的产品模型，探索也会不断取得进展。这些模型以后会使测试员设计出有效的测试。

经验 14, 使用诱导推断逻辑发现推测

诱导推断(abductive inference)又叫做假设归纳(hypothetical induction),是一种测试员每天都要使用的关键推理形式的有些怪的术语：最佳解释的推理。

其主要内容是：

1. 收集一些数据并要找出其中的意义。
2. 构造可能说明这些数据的各种解释。
3. 收集更多的数据，以确定或否定每种解释。
4. 从候选解释中，选择能够最一致地说明所有重要数据的解释，如果没有足够证据证实任何结论，则继续搜索。

诱导推断是科学和测试的基本方法。医生在为病人诊断时就要使用这种方法，测试员在判断产品是什么和不是什么、产品应该怎样运行或不应该怎样运行时，也要使用这种方法。如果要更好地进行诱导推断,则：

- 收集更多的数据。
- 收集更重要的数据。
- 收集更可靠的数据。
- 理解应用于数据的原因和效果。
- 找出可以说明数据的更多、更好的解释。
- 收集更多否定每个解释的数据。
- 收集更多区分解释的数据。
- 除非某个解释能说明所有重要数据，并且明显得到比其他解释更好的解释,否则不要确定解释。

诱导是寻找好的解释的一种系统化方法。尽管诱导推断过程并不提供绝对确定性,但是在很多情况下,这都是最佳手段。

经验 15,使用猜想与反驳逻辑评估产品

20 世纪初，哲学家 Karl Popper 引入了猜想与反驳(conjecture and refutation)方法(Popper 1989)，用于如何区分宗教和科学问题。这种方法基于科学家永远也不能绝对肯定任何具体事实,或关于自然的理论这个前提。任何东西都是猜想。有些猜想很强。例如重力的存在。

它是猜想而不是绝对肯定的事实，是因为能够想象出新的信息，如果这种信息存在，就会使人们拒绝该猜想。Popper 注意到，虽然我们不能证明猜想是真，但是却可以证明猜想是假的。因此，他提出给定猜想的置信度，只能来自反驳它。但又反驳不了的努力。

这种给出猜想并尝试反驳的方法，以三种重要方式应用于测试：

- 测试的目的是显示产品失效，要比显示产品正常更有力。如果想知道产品是否能够正常运行，寻找方法反驳其正常运行，这样的测试可能更好。
- 有关软件(软件有怎样的行为、如何好等)已经牢固形成的信念应该受到质疑。这意味着应该能够想象出新的与已有信念矛盾的信息。否则，我们的信念只不过就是信仰。信仰在私人生活中是有益的，但是对测试是有害的。
- 警惕声称以超过测试员所运行的具体测试的方式，检验或确认了产品的测试。任何量的测试都不能提供产品质量的确认性。

经验 16,需求是重要人物所关心的质量或条件

可以从很多种“需求”定义中选择适合测试员的定义。作为测试员，必须认识到谁的关于质量的观点最重要(并不是每个人的观点都同等重要)。然后了解对于产品他们要什么，不要什么。这种需求与软件工程的“需求”(在“需求文档”中发布的，并由有批准权限的人批准的一组陈述)和所有种类的规格说明没有差别。至于测试，产品应该具备或满足的任何质量或条件都是需求。

不同客户通过产品要得到不同的东西，他们不一定知道要什么，而且所要的东西会随时发生变化。这位测试员的工作更有意义。欢迎测试。

经验 17,通过会议、推导和参照发现需求

如果期望得到一迭印刷精美、文件袋封条盖有全球有效印章的需求，那还是另找工作吧。我们所经历的最好情况，需求文档(包括所有种类的产品规格说明、用例、多媒体文档等)是不完整、不准确的，尽管需求文档提供了信息并且是有帮助的。我们所看到的最差情况，文档是不完整、不准确、没有提供信息并且是没有帮助的。

测试员把项目文档(产品的显式规格说明)看作是惟一需求来源会影响其测试过程。在我们所管理的所有测试团队中，坚持这样要求会招致反驳。

需求信息到达测试员主要有三种途径：

- 会议。找出其有关质量的意见具有影响力的人，与他们交流，了解他们最关心什么。
- 推导。通过外推已知的项目和产品其他信息，确定什么需求最重要。
- 参照。既发现显式，也发现隐式规格说明，并以此作为测试的基础。

在很多项目中，优秀测试员所使用的大多数需求要么来自推导，要么来自隐式规格说明的参照。搜寻测试所需的信息，是测试员的工作。

有一本关于这个问题的书：《探索需求：设计之前的质量》(Exploring Requirements: Quality Before Design)(Gause 和 Weinberg 1989)。

经验 18,既要使用显式规格说明，也要使用隐式规格说明

并不是包含测试所依赖重要信息的所有参照都是显式地提供给测试员的：

- 显式规格说明是一个有用的需求信息源，经过客户的权威确认。(“是的，这就是规格说明，是产品描述。”)
- 隐式规格说明是没有经过客户权威确认的一个有用的需求信息源。(“这不是规格说明，但是有意义。”)

隐式规格说明的威信来自其内容的说服力和可信性，而不是客户的批准。在大多数情况

下，只有部分隐式规格说明与当前产品有关。隐式规格说明有很多种形式：

- 竞争对手的产品。
- 相关产品。
- 同一产品的老版本。
- 项目团队之间的电子邮件讨论。
- 顾客意见。
- 杂志文章(例如，有关产品老版本的综述)。
- 相关主题的教科书(会计书籍适用于会计应用程序)。
- 图形用户界面(GUI)风格指南。
- 操作系统(o / s)兼容性需求。
- 测试员自己的丰富经验。

当产品与显式规格说明冲突时，测试员的报告任务相对简单一些：“产品违反了规格说明，因此产品也许错了。”当违反的是隐式规格说明时，测试员的报告必须详细一些：“在 Microsoft Office 中，功能键 F4 固定用于重复命令。除非我们也这样定义，否则在日常工作中也使用 Office 的用户会感到困惑。”虽然没有人说 Microsoft Office 是这个产品的规格说明，但是客户可能会同意采用与 Office 一致的用户界面会提高可使用性。如果是这样，则 Office 就是这个产品的一个隐式规格说明。

有些测试员可能会问，为什么设计人员不把所有有用的信息都放入显式规格说明，使得他们不必再从隐式资源中分辨规格说明。回答很简单：虽然这样做方便了测试员，但是很昂贵，而且没有必要。客户相信测试员能够使用所需的各种参考资料快速找出重要的问题。

经验 19,“它没有问题”真正的含义是，它看起来在一定程度上满足部分需求

任何时候听到有人说“我试过了，它没有问题”、“我保证它没有问题”或“它现在更好了”，我们建议把“它没有问题”解释为“它看起来在一定程度上满足部分需求”。测试员应该立即想到的一些问题包括：

- “它”是什么?我们正在谈论的是产品的哪个部分?
- 外观是什么情况?到底观察到了什么?
- 检查了哪些需求?正确性如何?性能如何?
- 为了通过测试要在多大的程度上满足该需求?只是刚刚通过,还是超过指标很多?
- 它什么时候没有问题?测试覆盖了多大范围的条件?通过这些条件可以安全地推广到多远?

如果不愿意，可以默默地问自己。关键是如果对“它没有问题”没有进一步地限定，它就会是模糊的。测试员所认为的“它没有问题”的意义，可能与别的人定义不同。

经验 20,测试员所能得到的只是对产品的印象

不管测试员对产品的质量有什么看法，都是猜想。不管猜想有多么好的支持,也不能肯定自己是对的。因此，任何时候报告产品质量状态时,都应该用有关测试方法和测试过程的已知局限性的信息，对报告进行限定。

经验 21,不要将试验与测试混淆起来

试验的含义是什么，可能表示测试员执行一段探索式测试，产生一些没有文档或试验产品的临时性试验；也可能表示测试员编写一套可执行测试程序，或一套显式的测试过程；也可能表示某种高水平的测试矩阵、测试大纲或一套测试数据。

试验的概念是自包含的、实在的，与其他方便(我们通篇使用方便(convenient)概念，因

为它是测试界的标准行话)试验不同,但也是受限的。关键还是测试,而不是如何将测试打成被称为试验的包。测试是任何至少包含以下四种活动的活动:

- 配置。准备要测试的产品,将其置于正确的起始状态,否则测试结果会受到不良变量的影响。
- 运行。向产品输入数据,向产品发命令,以某种方式与产品交互。否则,产品只是放在那里,测试员能够做的只是评审,而不是测试。
- 观察。收集有关产品行为信息、输出数据、系统整体状态、与其他产品的交互等方面的信息。测试员不能观察所有事物,但是没有观察的任何事物都可能使测试员看不到问题所在。
- 评估。运用规则、推理或可检测所观察到数据中存在问题的机制。否则要么不能报告问题,要么只是把数据提交给客户,由客户自己进行评估。

试验产生的可能有很多形式,不要过于关注形式,要保证有这四种活动发生。要关注执行这些活动的思考者,关注试验是否很好地完成了预想的策略和测试使命。

经验 22,当测试复杂产品时:陷入与退出

有时复杂性可能是无法抗拒的。测试员的意志可能会被击垮。因此,当要测试复杂和使人畏惧的功能集合时,可间歇进行。人的头脑具有处理复杂问题的惊人能力,但是不要指望马上就能理解复杂产品。可试着先研究复杂产品 30 分钟或一个小时,然后停下来干点别的。这就是陷入与退出(P1unge in and quit)法。不要担心在这段不长的时间内效率不高,如果觉得问题太多,要尽快退出。

这种方法的优点是,除了选择产品的一部分并研究外,绝对不需要计划。经过几个轮次的陷入与退出,就会开始明白产品的模式和轮廓,很快就会在头脑中更系统、更具体地测试和研究策略。这种方法很神奇。最终,会掌握足够的知识以设计全面的测试计划,如果认为这些计划能够完成自己的任务。

经验 23 运用试探法快速产生测试思路

试探法(hcuristic)是一种经验规则,是一种基于经验做出猜测的方法。这个词源自希腊语,表示“开始发现”。试探法并不能保证得到正确的答案或最佳答案,但是很有用。最早运用试探法的著作是《如何解决它》(How to Solve it)(Polya 1957)。

出于可能的测试用例数量是无限的,因此肯定要选出在所面临的时间和预算约束条件下有效的少量测试用例。有经验的测试员会收集并共享能够改进其猜测质量的测试试探方法。一组好的试探方法有助于很快地生成测试。以下是采用试探法测试的一些例子:

- 测试边界。边界更有可能暴露规格说明的模糊问题。
- 测试所有错误消息。错误处理代码与主流功能代码相比,一般比较弱。
- 测试与程序员的配置不同的配置。程序员已经偏信自己的配置没有问题。
- 运行比较难设置的测试。在其他条件相同的情况下,易于设置的测试更有可能已经被执行过。
- 避免冗余测试。如果某个测试实际上是重复其他测试,就不会产生新价值。

为了明智地运用试探法,请注意:试探法中并没有智慧,智慧来自测试员。试探法所能做的,只不过就是为测试员的思考提出建议。盲目使用自己并不了解的试探法并不是好的测试实践。在收集测试方法时,要了解每个方法背后的原理,以及更适用和不太适用的条件。

经验 24,测试员不能避免偏向,但是可以管理偏向

测试员是有偏向的,这使得测试员选择一部分测试的可能性要比其他测试大。如果有一

个很长的编辑字段，测试员也许更可能输入诸如 1111111111，而不是 3287504619，因为输入字符重复的字符串，要比从 0 到 9 随机选择数字更容易。也许这是一种很小的偏向，但仍是一种偏向。更糟的偏向是，大多数测试员倾向于测试最可视的功能，不管是不是最重要的功能。此外，大多数测试员还倾向于考虑认为与自己类似的用户，倾向于使用非常简单、非常荒谬的输入，而不是具有中等复杂度的现实输入。

以下是一些常见偏向：

- 同化偏向。更有可能把未来的测试结果解释为总体上证实自己对产品的看法。
- 证实偏向。更有可能关注确实会证实自己对产品看法的测试结果。
- 可用性偏向。如果头脑中已经想到一种用户以某种方式操作的场景，则更有可能认为这种操作更常出现。
- 最初印象偏见。更信任所做的第一次观察。
- 最新印象偏见。更信任所做的最近一次观察。
- 框架效应。对错误报告的反应与措辞有很大关系，不管其真正含义如何。
- 知名偏向。把碰巧认识的用户意见放在更重要的地位。
- 表达偏向。期望较小的问题也许有较小的原因，而严重问题会有大原因。

测试员不能避免这些偏向，因为这些偏向在很大程度上已经固化在头脑中。测试员能够做的是管理偏向。例如，只需通过研究偏向并在实践中注意，这样在思考时就可以更好地进行补偿。多样化也可以抵御过强的偏向。如果测试员集体谈论测试问题，可以将一个测试员的偏向降低到最低限度。

根据定义，试探法也是一种偏向。使用试探法，是因为其偏向可以以比较好的方式引导测试员。

经验 25,如果自己知道自己不聪明，就更难被愚弄

骗子说，最容易上当的人，是绝对自信不会被愚弄的人。作为测试员也可以把这条定律用于自己的工作中。证明自己容易被愚弄。做到这一点并不难，只需仔细看看自己在测试中犯的错误。任何时候都要注意其他测试员所发现的自己本来也可以发现，但是没有发现的问题。

如果真心认为自己容易被愚弄，也会比较谨慎一点，在考虑自己的测试策略细节时就会更认真一点。这是一种新测试员能够提高的最快的方法之一，因为知道自己可以被愚弄是一种态度，并不是特殊技能或知识。新测试员的问题是，对于他们来说，这个定律只是一条信仰（“人家告诉我，我应该认为我可以被愚弄……”），而有经验的测试员的感觉和反应，会通过实际教训唤起和加强（“我还记得 1994 年的那次大教训。我们怎么也没有想到病毒会感染我们自认为性能非常可靠的磁盘。我的声誉在这一天全给毁了”）。

经验 26,如果遗漏一个问题，检查这种遗漏是意外还是策略的必然结果

如果掷币猜边时，猜的是国徽面，出现的却是字面。这是否意味着做出了差的决策？以任何理性的观点看都不是这样。除非在硬币上做了手脚，否则出现任何一回的机会都是 50%。出现字面没有什么可奇怪的，只是不够幸运罢了。决策策略没有问题。

在测试过程中没有发现程序错误时也存在同样问题，同样也会困扰客户。在研究测试策略出现了什么问题之前，先不要自责。出现遗漏，是否因为忠实地执行了好的测试策略，并只是碰巧没有发现那个特定的问题？如果是这样，可保持原有方针不变。确实有这种情况。但是，如果遗漏程序错误是因为测试策略关注了错的问题类型，可利用这个机会改进测试策略。

经验 27,困惑是一种测试工具

当测试员感到困惑时,这可能是某种重要的预示。

- 规格说明不清楚吗?规格说明中的模糊点,常常是为了掩盖有影响力的项目相关人员之间的重要分歧。
- 产品不清楚吗?产品可能有严重问题。
- 用户文档不清楚吗?产品的这个部分可能太复杂,有太多的特例和不一致性要描述。
- 内部问题只是难以理解吗?我们试图自动化的有些系统具有内在的复杂性,或包含复杂的技术问题。程序员也认为它们复杂、困难,并导致自己犯遗漏、误解和过于简化的错误。

测试员对产品、技术和一般测试问题了解得越多,自己的困惑就会成为更有力的指南针,指出重要问题所在。

在测试过程中,如果对产品一无所知,那么至少知道自己在困惑。在这种情况下,困惑可以成为最佳交付内容,即提出也许其他人没有勇气提出的问题。

经验 28,清新的眼光会发现失效

理解事物,是把新信息吸收到已知信息中,同时修改已知的信息以适应新信息的高智力过程。测试员在理解了产品或功能部件之后,会在头脑中形成映射,并且头脑不再那么努力工作。对于测试员来说这可能是个问题。当非常了解产品后,会对产品做出更多的假设,并更少检查这些假设。

这种情况对于测试至少有三点提示:

- 第一次接触产品或功能时,要特别注意使自己困惑和烦恼的地方。这可能说明用户也会有类似反应。
- 当与团队的新成员一起工作时,与他们一起测试。观察他们在了解产品时的反应。
- 警惕陷入测试惯例。即使没有遵循严格的测试脚本,也可能对特定功能太熟悉,以至于以越来越窄的方式进行测试。在任何可能的地方引入多样性,或改由其他测试员负责。

经验 29,测试员要避免遵循过程,除非过程先跟随自己

警惕其他人的过程。测试用例和过程的描述,常常不提测试的内部设计目标。这非常容易使测试员在执行测试时并不太理解如何建立测试,或寻找什么。换句话说,测试员并没有真正跟上过程。一般来说,测试过程的编写和设计都比较差,因为没有多少优秀测试员像擅长计算机那样擅长程序设计人员的工作。如果要遵循测试过程,最好采用自己设计、自己拥有或已经完全了解的过程。

为了得到最好结果,测试员必须掌握自己的测试,而不是自己的文档。要使过程跟随自己。

如果确信那些过程很好,也至少要研究一下过程的工作原理。请参阅《使我们聪明的事物:机器时代的人性保护》(Things that Make Us Smart : Defending Human Attributes in the Age of The Machine)(Norman 1993)和《信息的社会寿命》(The Social Life of information)(Brown 和 Duguid 2000)。

经验 30,在创建测试过程时,避免“1287”

我们中的 Bach 曾经见过一位测试员编写的测试过程包含“在字段中输入 1287 个字符。”这 1287 是从哪里来的?测试员解释说. 他的测试想法只不过是在一个小输入字段中,输入非常多的字符。因为她听说测试过程必须具体,因此小心地数了自己已经输入的字符数,1287,这就是过程中的这个数的由来----一个任意数,现在却被永远供奉起来,就像是印在水泥人行

道上的猫的脚步一样。

过于详细没有什么好处。当编写测试过程时,要避免与测试概念无关的细化。包含所有必要的信息和设计与解释测试所需的细节,但是要让未来的测试员有创造性和判断力地执行,让未来测试员引入变化以使现在所编写的测试过程新鲜、高效。

经验 31,测试过程的一个重要成果,是更好、更聪明的测试员

我们经常听到反对产生很少或不产生文档的测试的理由,好像测试的惟一价值就是通过测试产生的文档。这种观点忽略了测试的一种意义深远的重要产品:测试员本身。

好的测试员要永远都在学习。随着项目的进展,他们不断加深对产品的了解,逐渐从各个方面提高对产品的反应能力和敏感性。了解产品并已经经历过一两次产品发布的有经验的测试员,即使没有任何指示,但与有一套如何测试产品的书面指示的没有经验的测试员相比,他们测试的有效性也要高得多。

软件测试领域内的一些顾问和论文作者看起来相信,只要提供测试过程,测试效果差的测试员就可以变成测试效果好的测试员。在我们看来,这是一种差的实践,这反映出他们对测试和进行有效测试的人的根本误解。

在评估测试过程时,首先要看项目测试员的素质,要看他们怎么思考,以及这种思考怎样对其行为产生影响。只有掌握了这些信息才能评估他们的工作产品。

经验 32,除非重新发明测试,否则不能精通测试

不要彻底改造轮子。请等一下,难道轮子不是历史上被重新发明次数最多的吗?这不是好事情吗?毕竟我们的汽车是装在充气轮胎上,而不是花岗岩圆盘上。轮子如果说不是数以百万计的话,也有数以千计的变种。也许我们可以从中得到启发。在我们看来,重新发明东西至少有两个原因:通过改造使其适应新条件,了解其工作原理。而要掌握它,这两个方面都需要。

我们有的同事忠告测试专业的学生不要重新发明测试,也不要重新发明测试思想。我们不同意这种观点。这就像是学习科学又不想做实验一样。通过其他思想家进行学习是没有错的,我们认为这样学习是很重要的。但是如果这是学习的惟一方式,那永远也不会成为测试技艺的行家。这样的人将是技术员,但不会再进一步了。按照指示做不会掌握任何东西,就像要沿高速路上火星一样。我们提倡要像伟大的机械师和伟大的程序员那样地学习测试:把东西分解,琢磨其工作原理,再以新的方式组装到一起。不要把自己限制为只是接受智慧的服务者,而应该使自己成为智慧的创造者。

在学习过程初期,要重新发明不是非常好的测试、想法、手段和文档。这是正常的。要永远使头脑运转,观察其他测试员,研究和不断评估如何筛选自己的思想。如果想要善于做到这一点,就必须实践。

我们这样做已经有很多年了,我们仍然在重新发明,仍然在反思老的想法。我们所尊敬的每个同事都是这样走向精通之路的。

QQ 群: 3105889

全力打造以苏州本地测试人员为主的群,欢迎大家加入