敏捷测试全攻略

目录

敏捷开发中的软件测试.....3
什么是敏捷测试？.....3
敏捷测试中测试人员扮演的角色.....3
单元测试和可接受性测试.....3
测试人员是否应该拥抱敏捷开发？.....3
敏捷测试实践.....4
敏捷测试的挑战.....4
什么是敏捷开发？.....4
为什么以前的开发模式不再适用？.....5
测试的作用.....5
敏捷测试的挑战之一：测试员是否不再需要了？.....5
敏捷测试的挑战之二：测试不完整的软件.....5
敏捷测试的挑战之三：可接受性测试是否过于简单了？.....5
敏捷测试的挑战之四：把测试员作为项目组的一部分.....6
敏捷测试的挑战之五：测试什么时候完成？.....6
敏捷测试的挑战之六：我们还需要 bug 跟踪系统吗？.....6
敏捷测试的挑战之七：用什么质量标准来度量敏捷项目.....6
敏捷测试的挑战之七：回归测试.....7
敏捷测试的挑战之八：回归测试工具.....7
敏捷测试用例设计.....8
测试用例的粒度.....8
基于需求的测试用例设计.....9
测试用例的评价.....9
测试用例数据生成的自动化.....10
敏捷自动化测试.....10
公式化的典型的自动化测试过程.....10
敏捷自动化测试的原则.....11
工具支持测试.....11
到处是工具.....12
“工具铁匠”的任务.....12
测试员可能会问“工具铁匠”的问题.....12
管理敏捷自动化测试.....12
敏捷测试指引（1）-简介.....13
敏捷测试指引（2） - 测试与例子.....14
敏捷测试指引（3）- 用面向技术的例子支援程序员.....16
敏捷测试指引（4）- 用面向业务的例子支援项目组.....18
激发出正确的代码.....18
促进项目交流.....18
让可能发生的更加明显.....19

敏捷测试指引（5）- 用面向业务的例子批判产品.....20
敏捷测试指引（6）- 用面向技术的例子批判产品.....21
敏捷测试指引（7）- 敏捷项目中的测试员.....22
敏捷开发中的 7 种测试类型.....25

敏捷开发中的软件测试

陈能技
2007-9-5
参考：Bret Pettichord 的《Agile Testing - What is it? Can it work?》和《Agile Testing Challenges》

敏捷宣言：
个体和交互比过程和工具更有价值；
能工作的软件比全面的文档更有价值；
顾客的协作比合同谈判更有价值；
及时响应变更比遵循计划更有价值。- www.agilemanifesto.org

什么是敏捷测试？

测试遵循敏捷宣言进行，把开发作为顾客看待。项目的测试采用敏捷方法论。
敏捷测试的原则与上下文驱动测试（Context-Driven Testing：www.context-driven-testing.com）的原则有交集，例如，上下文驱动测试的七大原则中的第三条：工作在一起的项目组成员是项目的上下文的最重要的组成部分。就与敏捷宣言中的“个体和交互比过程和工具更有价值”一样强调人的作用。

敏捷测试中测试人员扮演的角色

1、 测试是项目的“车头灯”，它告诉大家现在到哪了，正在往哪个方向走。
2、 测试为项目组提供信息，使得项目组基于可靠的信息作出正确的决定。
3、 “BUG”是让用户感觉烦恼的东西，测试人员不作出发布的决定。
4、 测试员不保证质量，整个项目组对质量负责。
5、 测试不是抓虫子的游戏，它的目的不是纠缠在错误中，而是帮助找到目标。

单元测试和可接受性测试

测试驱动开发，开发人员在写代码之前要先写单元测试，用于激发代码的编写、改进设计（降低耦合度，增加内聚）、支持重构。很多开源的测试工具支持单元测试（xUnit）。
用户故事是需要编码实现的功能特性的简短的描述。可接受性测试验证用户故事的完整性。理想的情况下，用户故事是在代码编写前就写完。

测试人员是否应该拥抱敏捷开发？

有些人说 XP 会导致差的质量并且是偷懒的借口。我认为 XP 是令人激动的，并将在行业中改进测试的实践。
参见《Testers Should Embrace Agile Programming 》

（http://www.io.com/~wazmo/papers/embrace_agile_programming.html ）

敏捷测试实践

通过对话产生测试，谁来测试？客户往往太忙了，不可能参与测试。定义测试是很关键的活动，应该把开发人员和顾客代表包括进来，不要只是测试员自己做。
把用户故事转换成测试。测试提供的是目标和指南、及时的反馈、进度度量。测试应该用指定的格式出现，以便让用户或顾客能清楚明白，还要足够的明确，以便能被执行。
开发人员负责提供支持自动化测试的特性。大部分情况下，为产品添加测试接口，而尽量不用外部测试工具。
计划在每个迭代中探索产品，寻找 bug、遗漏的特性和改进的机会。

进一步学习
Lessons Learned in Software Testing - www.testinglessons.com
Ward Cunningham's acceptance testing framework - fit.c2.com
Agile Testing Papers - www.testing.com/agile www.pettichord.com

敏捷测试的挑战

陈能技
2007-9-5
参考：Bret Pettichord 的《Agile Testing - What is it? Can it work?》和《Agile Testing Challenges》

我们从上下文驱动测试的七大原则(www.context-driven-testing.com)可以看出，上下文驱动测试倾向于快速的反馈和适应变化的环境。所以上下文驱动测试的很多原则和做法可以应用到敏捷开发的软件测试中来。

什么是敏捷开发？

敏捷开发是递增式的、迭代的、不断调整的开发模式。我们逐渐地建立起软件系统，能看到系统在成长，能展示进度。通过多次发布或项目的阶段检查点，每一次都比上一次靠近目标。迭代包括需求的开发和测试，典型的迭代周期是 2 周。目标随着从上一次的迭代中学到的东西、反馈以及商业机会而调整。
在敏捷开发中，工作被分解成“故事”，也叫特性或用例，组合成任务分派给不同的程序员。定义好接受标准，开发直到单元测试和接受测试通过才算完成。
敏捷开发讲求合作，结对进行编程，避免个人拥有专门的知识，代码属于项目组共有。
在敏捷开发中不存在回退，讲究持续地集成，单元测试（通常使用测试驱动的开发方式），持续地进行回归测试。

为什么以前的开发模式不再适用？

以前的开发模式要求有详细的测试计划，但是缺乏足够的时间来写，而且测试计划受很多因素的影响经常改变。
以前的开发过程会专门留出一个阶段来测试，但是你不能定义进入和退出的标准，测试阶段会随之而过。
以前的开发模式强调变更控制，但是现在的软件需求变更非常频繁，变更成了家常便饭。
以前的开发模式要求测试要对软件做出权威的判断，但是测试很难做出权威的关于软件正确性的判断。

测试的作用

有价值的东西有么提供产品，要么提供服务。那么测试提供什么产品或服务呢？有人认为测试提供调试通过的、经过测试的软件。这是错误的回答。测试不提供产品，测试提供信息，关于开发过程中的软件的状态的信息，以便基于这些信息做出决定。

敏捷测试的挑战之一：测试员是否不再需要了？

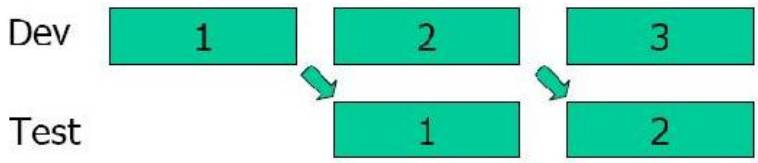
既然有开发人员做单元测试了，我们还需要测试员吗？有些项目团队采用了敏捷开发方式后把测试员都给解雇了，然后过了不久他们就后悔了。
测试可以是除 QA 或测试员外的人来做，例如业务分析员，有些项目团队让开发人员来做接受性测试。
但是有专门的测试员带来两个好处：
1、 专注于用户使用而不是软件的技术实现
2、 专注于发现软件的错误而不是证明完整性

敏捷测试的挑战之二：测试不完整的软件

频繁的迭代产生的测试版本很多时候是不完整的，测试员如何测试这些不完整的代码呢？
“故事”应该从业务价值方面来定义。一个“故事”应该在一个迭代周期内完成。好的“故事”是不容易定义出来的，但是差的故事对测试人员的影响比对开发人员的影响还要大。有时候测试人员需要帮助定义“故事”。

敏捷测试的挑战之三：可接受性测试是否过于简单了？

测试人员如果只是做可接受性测试，只是验证“故事”是否完整，岂不是太简单了？这样怎么能做好测试呢？
其实，每一个迭代都需要额外的测试，而不仅仅是局限于验证“故事”的完整性。



在迭代测试中还要按需进行下面类型的测试 ：
探索性测试：同时学习系统、计划和执行测试，寻找 bug、遗漏的特性和改进的机会。
组合交互测试：专注于特性之间的交互。
场景测试：模拟真实世界的场景进行测试。
疲劳测试：长时间地执行软件
业务循环测试：基于月末、季度末等业务循环的边界来执行场景
压力测试：对系统施加强大的压力进行测试

敏捷测试的挑战之四：把测试员作为项目组的一部分

把测试员作为项目组中的一员不是牺牲了他们作为一个组织的完整性吗？
测试员一直被认为是受压迫的对象，经常坐在一起互相诉苦、互相支持。现在是时候结束这种情况了。测试员应该跟开发人员和分析师坐在一起，当项目组中有更多的正式或非正式的沟通时才有可能达到敏捷。

敏捷测试的挑战之五：测试什么时候完成？

没有专门分配的时间来完成测试，我们怎么知道什么时候测试应该结束？
敏捷测试员需要根据项目和产品的风险来调整测试。基本上测试的优先级应该跟“故事”的优先级一致。BUG 列表也提供了测试完整性的提示。
一个好的测试员是永远都能找到需要完成的测试来做的。
为什么需要跟开发人员结对进行测试呢？因为开发人员对潜在的错误有一定的洞察力，测试员对约束和错误的时机有一定的洞察力。而他们在一起能使自动化测试更加成功。
测试员应该自动化可接受性测试，使用与开发环境一样的编程语言来编写可接受性测试的代码，重用单元测试的框架，使软件更加可测。
利用“灰盒”测试。设法弄清楚系统各模块之间的关系，分析变更的影响，看什么是需要测试的，什么是可以不测试的。弄清楚 bug，bug 的表面现象是什么？产生 bug 的根本原因是什么？弄清楚风险，使用解决风险的测试策略，调整测试目标。

敏捷测试的挑战之六：我们还需要 bug 跟踪系统吗？

有些人说敏捷团队不需要跟踪 bug，只需要把发现的 bug 尽快修正就行了。
这种做法只适用于开发过程的测试，如果是一个完整迭代的测试，你就需要 bug 跟踪系统，因为有些 bug 不是在这个迭代马上修改的。

敏捷测试的挑战之七：用什么质量标准来度量敏捷项目

其中一个最好的质量标准是发布后逃逸的 bug 数量。不幸的是，这是个事后的衡量标准。

采用每个迭代后计算逃逸 bug 数量的方法能标识代码的质量。

我们还可以从 bug 学习到很多东西：
1、 是否有些类型的 bug 在可接受性测试中发现的，其实是可以在单元测试就发现呢？如果是，把它加入到单元测试。
2、 我们是否能让 bug 的发现过程或 bug 的诊断更简单？
3、 我们是否能让程序员不那么容易犯这种普通的错误？

敏捷测试的挑战之七：回归测试

伴随着频繁的迭代，我们需要频繁地重新测试，单元测试是不足够的。我们怎样有效地进行用户层面的回归测试呢？
你不一定需要在每次的迭代都做完整的回归测试。可以每个迭代运行一部分的测试。需要某种程度上的用户层次的自动化回归测试。

敏捷测试的挑战之八：回归测试工具

大部分的商业测试工具在敏捷环境下都不是很好用。大部分有这些缺点：
1、 指定的语言
大部分商业测试工具会指定某种语言，例如：WinRunner（TSL）、SilkTest（4test）、Robot（Test Basic），但是一些新的工具也开始使用标准语言，例如：Astra QuickTest（VB Script），XDE Tester（Java）
参考 http://www.stickyminds.com/se/S2326.asp

2、 与源代码控制的结合不好
很多工具没有与源代码控制工具集成，使用临时文件和目录（WinRunner），参考 http://paulhammant.com/blog/000245.html
关键信息存储在 Repositories 中，例如 Rational

3、 很难与持续集成配合使用
缺乏外部调用的 API，不允许作为一个库被使用，因此很难与持续集成整合在一起。一些新的工具则有所改进，例如 TestComplete

4、 不能在所有机器上部署（受 License 限制）
受限制的、昂贵的 License，使得很多开发人员不能例如工具运行测试

这些问题使得他们对于整个团队来讲不够实用。敏捷团队倾向于构建自己的测试工具和利用开源工具。

开源测试工具
现在已经出现很多开源的测试工具，支持 windows、Java、Web 等平台，现在大部分都集中在 web 平台，例如:HttpUnit、WTR 等。

关于 Agile Testing，可以参考以下资料：
•Agile Testing Papers
http://www.testing.com/agile
•“Where are the Testers in XP?”
http://www.stickyminds.com/s.asp?F=S6217_COL_
•Mailing List
http://groups.yahoo.com/group/agile-testing/

敏捷测试用例设计

陈能技
2007-9-20

敏捷宣言：
个体和交互比过程和工具更有价值；
能工作的软件比全面的文档更有价值；
顾客的协作比合同谈判更有价值；
及时响应变更比遵循计划更有价值。
- www.agilemanifesto.org

并非每个企业都能严格按敏捷的相关开发方法进行项目管理，例如测试驱动、XP、SCRUM 等。也并非都需要按这些方式管理才能实现敏捷。只要我们理解了敏捷的原则和精髓，我认为很多方法、很多地方都可以应用敏捷的思想，实现敏捷的管理。
测试用例的设计是其中一项。

测试用例的粒度

测试用例可以写得很简单，也可以写得很复杂。最简单的测试用例是测试的纲要，仅仅指出要测试的内容，如探索性测试（Exploratory Testing）中的测试设计，仅会指出需要测试产品的哪些要素、需要达到的质量目标、需要使用的测试方法等。而最复杂的测试用例就像飞机维修人员使用的工作指令卡一样，会指定输入的每项数据，期待的结果及检验的方法，具体到界面元素的操作步骤，指定测试的方法和工具等等。

测试用例写得过于复杂或过于详细，会带来两个问题：一个是效率问题，一个是维护成本问题。另外，测试用例设计得过于详细，留给测试执行人员的思考空间就比较少，容易限制测试人员的思维。

测试用例写得过于简单，则可能失去了测试用例的意义。过于简单的测试用例设计其实并没有进行“设计”，只是把需要测试的功能模块记录下来而已，它的作用仅仅是在测试过程中作为一个简单的测试计划，提醒测试人员测试的主要功能包括哪些而已。测试用例的设计的

本质应该是在设计的过程中理解需求，检验需求，并把对软件系统的测试方法的思路记录下来，以便指导将来的测试。

大多数测试团队编写的测试用例的粒度介于两者之间。而如何把握好粒度是测试用例设计的关键，也将影响测试用例设计的效率和效果。我们应该根据项目的实际情况、测试资源情况来决定设计出怎样粒度的测试用例。

软件是开发人员需要去努力实现敏捷化的对象，而测试用例则是测试人员需要去努力实现敏捷化的对象。要想在测试用例的设计方面应用“能工作的软件比全面的文档更有价值”这一敏捷原则，则关键是考虑怎样使设计出来的测试用例是能有效工作的。

基于需求的测试用例设计

基于需求的用例场景来设计测试用例是最直接有效的方法，因为它直接覆盖了需求，而需求是软件的根本，验证对需求的覆盖是软件测试的根本目的。

要把测试用例当成“活”的文档（Effective Software Testing : 50 Specific Ways to Improve Your Testing – Elfriede Dustin），因为需求是“活”的、善变的。因此在设计测试用例方面应该把敏捷的“及时响应变更比遵循计划更有价值”这一原则。

不要认为测试用例的设计是一个阶段，测试用例的设计也需要迭代，在软件开发的不同的阶段都要回来重新审视和完善测试用例。

测试用例的评价

测试用例设计出来了，质量如何，如何提高测试用例设计的质量？就像软件产品需要通过各种手段来保证质量一样，测试用例的质量保证也需要综合使用各种手段和方法。

测试用例的检查可以有多种方式，但是最敏捷的应当属临时的同行评审。我认为同行评审，尤其是临时的同行评审，应该演变成类似结对编程一样的方式。从而体现敏捷的“个体和交互比过程和工具更有价值”，要强调测试用例设计者之间的思想碰撞，通过讨论、协作来完成测试用例的设计，原因很简单，测试用例的目的是尽可能全面地覆盖需求，而测试人员总会存在某方面的思维缺陷，一个人的思维总是存在局限性。因此需要一起设计测试用例。

除了同行评审，还应该尽量引入用户参与到测试用例的设计中来，让他们参与评审，从而体现敏捷的“顾客的协作比合同谈判更有价值”这一原则。这里顾客的含义比较广泛，关键在于你怎样定义测试，如果测试是对产品的批判，则顾客应该指最终用户或顾客代表（在内部可以是市场人员或领域专家）；如果测试是指对开发提供帮助和支持，那么顾客显然就是程序员了。

因此，参与到测试用例设计和评审中来的人除了测试人员自己和管理层外，还应该包括最终用户或顾客代表，还有开发人员。

测试用例数据生成的自动化

在测试用例设计方面最有希望实现自动化的，要当属测试用例数据生成的自动化了。因为设计方面的自动化在可想象的将来估计都很难实现，但是数据则不同，数据的组合、数据的过滤筛选、大批量数据的生成等都是计算机擅长的工作。

很多时候，测试用例的输入参数有不同的类型、有不同的取值范围，我们需要得到测试用例的输入参数的不同组合，以便全面地覆盖各种可能的取值情况。但是全覆盖的值域可能会不可思议地广泛，我们又需要科学地筛选出一些有代表性的数据，以便减轻测试的工作量。在这方面可利用正交表设计数据或成对组合法设计数据。

可利用一些工具，例如 TConfig、PICT 等来产生这些数据。

在性能测试、容量测试方面，除了设计好测试用例考虑如何测试外，还要准备好大量的数据。大量数据的准备可以使用多种方式：编程生成、SQL 语句生成（基于数据库的数据）、利用工具生成。

工具未必能生成所有满足要求的数据，但是却是最快速的，编程能生成所有需要的数据，但是可能是最复杂、最慢的方式。所以应该尽量考虑使用一些简单实用的工具，例如 DataFactory 等。

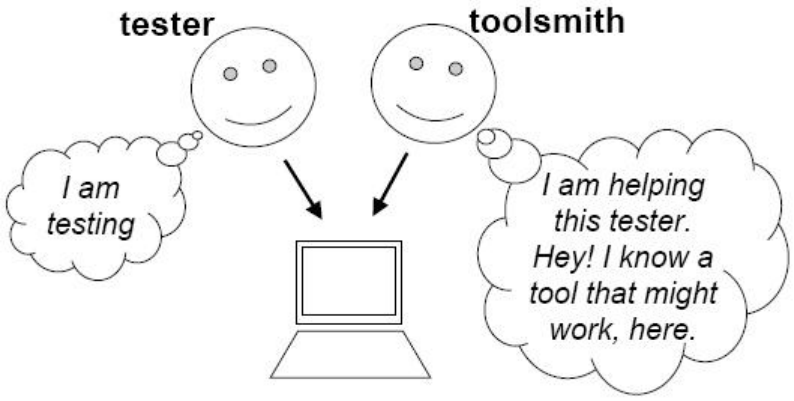
敏捷自动化测试

陈能技
2007-9-2
原文：Agile Test Automation – James bach

公式化的典型的自动化测试过程

1、 购买一个昂贵的 GUI 测试执行工具（例如 Rational、Mercury、Compuware 等）
2、 定义很多测试用例
3、 招聘一个自动化测试组实现每个测试用例的自动化执行
4、 构建一个完整的测试库和框架
5、 不断地完善和修正

如果你的产品很容易测试并且变更不大的话，以上方式很适合。但是关于自动化测试，我们为什么想得那么狭窄？
尝试把自动化测试想成是“任何使用工具来支持测试”。敏捷自动化测试就是把敏捷开发的原则应用在测试自动化上。



敏捷自动化测试的原则

1、测试自动化意味着使用工具支持测试项目的各个方面，不仅仅是测试执行方面。
2、当测试自动化得到指定的程序员（toolsmiths-“工具铁匠”）支持时，会不断地顺利进行。
3、“工具铁匠”由测试员领导。
4、“工具铁匠”收集并应用各种各样的工具来支持测试。
5、“工具铁匠”帮助实现可测特性并“打造”工具以便利用这些可测特性。
6、 组织实现测试自动化是为了完成某个短期的目标。
7、 避免盲目进行长期的自动化测试任务，而不是基于业务场景的分析。

工具支持测试

1、 测试创建（数据和脚本的产生）
工具可以产生特定的数据，例如：随机的 Email 信息，或产生数据库，或产生组合参数来覆盖我们的测试。
2、 系统配置
工具可以保持或重现系统参数，把系统设置到某个特定的状态，或创建或回滚到一个“ghost”的磁盘
3、 模拟
工具可以为测试模拟一些不具备的环境条件，这些环境可能会很难出现或提供起来很昂贵。
4、 测试执行
工具可以操作软件系统本身，模拟用户的 GUI 操作或绕过 GUI 层直接使用某些测试接口。
5、 问题分析
工具可以使某些不可见的东西可见。稳定地分析产品或分析 log 文件，或监视系统参数。
6、“预言”
“预言”是通过某些机制来判断错误或成功。工具可以自动地判断产品的某些类型的错误条件。
7、记录和覆盖分析
工具可以帮助记录测试过程覆盖的地方和未覆盖的地方。
8、 试管理
工具可以记录测试结果，组织测试用例。

到处是工具

1、 MSDN 库
2、 微软的很多开发工具都包括很多有用的小工具
3、 微软兼容性工具包和其他免费工具（www.microsoft.com）
4、 基于网页的测试资源（HTML checkers、accessibility analyzers 等）
5、 widows 资源包
6、 脚本语言（例如：Perl、Ruby、TCL）和相关库
7、 共享资源库（www.download.com）
8、 操作系统监视工具（www.sysinternals.com）
9、 开源测试软件（www.opensourcetesting.org）
10、探索性测试的监视软件（www.spectorsoft.com）
11、项目组其他人正在使用的工具

“工具铁匠”的任务

1、 快速响应测试员的请求并提供协助
2、 查找影响测试效率的问题
3、 调查测试员关心的问题的可能的解决方案
4、 应用技术改进测试过程
5、 提供产品的可测性功能特性
6、 研究工具并学习怎样使用
7、 收集开发人员或测试员创建的工具
8、 对产品进行评审以便计划自动化的可能性

测试员可能会问“工具铁匠”的问题

1、 我怎样测试这个新的功能？
2、 我如何才能看到产品内部做了什么？
3、 我如何判断测试是否通过？
4、 有没有办法让我能自动地执行这些操作？
5、 有没有办法让 bug 重现更加容易些？
6、 帮助我调查这些 bug
7、 这里有一个测试要执行，你能否帮助我产生 1000 个变量？
8、 我的测试覆盖了产品的多少地方？
9、 我想对产品进行压力测试，是否有什么工具可以使用？

管理敏捷自动化测试

1、 请求清单
请求清单是测试员发出的自动化测试要求

2、 任务清单
任务清单是每位“工具铁匠”收到的分派任务
3、 移交清单
移交清单是目前被测试组使用的解决方案。每个都包括解决方案对测试效率的影响的简要描述
4、 维护清单
维护清单是需要改进的解决方案的清单。可以考虑把它分成两部分：关键维护和增强维护。
5、 障碍清单
障碍清单是所有尚未解决的影响测试效率的问题清单。这些问题需要新的昂贵的工具、实际的可测试性改进、或者需要更多工作而难以在短期实现

对于一个大型的测试组来说，至少需要一名“工具铁匠”，但是不要把所有测试员都作为“工具铁匠”，因为这样做的成本太高，这样所有测试员都要像“工具铁匠”一样思考问题。

敏捷测试指引（1）-简介

陈能技
2007-9-24
原文：Agile Testing Directions – Introduction （Brian Marick）

在 XP Agile Universe 上，两个人-或许更多-告诉我说，我在敏捷测试的发展方面贡献不够。我在过去 5 年里花了太多的时间说我不知道敏捷测试会怎样，没有足够的指示和指导。“但是让我们看看，也许我们可以找到。”他们可能是对的。因此我让本文作为这方面的一个起点。

我先重申一些普遍的概念区别，以作为起点。

如果你听到别人在谈论敏捷项目中的测试，问一下那些测试是面向业务的还是面向技术的，会对你有很大的帮助。面向业务的测试是你可以用一个业务专家感兴趣的术语来向他描述测试。如果你通过电话描述测试回答了什么问题，你可以使用业务领域的术语：“如果你支取超过你的账户余额的现金，系统是否会自动给予你一笔与超出部分等额的贷款？”

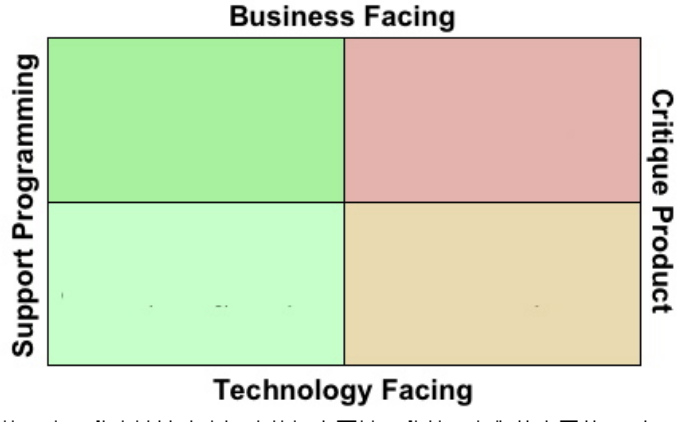
面向技术的测试是你使用程序员的领域的术语来描述测试：“不同的浏览器会通过不同的方式实现 Javascript，所以我们测试产品是否能在最主要的浏览器上工作。”或者：“如果用户记录不存在，PersistentUser#delete 不应该执行。”

（这些分类有着很多模糊的界线，例如，选择测试哪个浏览器配置，部分是业务决定。）

问一下正在讨论测试的人，他们希望测试支援编程还是批判产品。对于“支援编程”，我的意思是程序员把测试作为编程的主要组成部分。例如，一些程序员编写测试用例来告诉他们下一步应该写什么代码。通过编写那些代码，他们改变一些程序的行为。这些更改之后通过运行这部分的测试来保证他们修改的是他们需要的。运行其它的测试来确保更改的行为不会

影响其它不需要更改的部分。

批判产品的测试则不是专注于编程方面。而是在已完成的产品上查找发现产品的不足之处。如果把这两类区别放到一起，就得到下面的矩阵图：



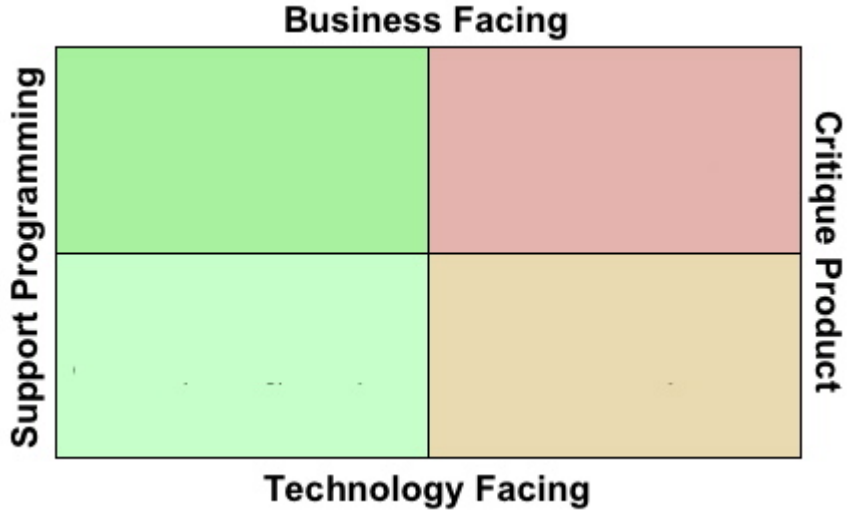
接下来，我会谈谈这个矩阵的每个区域，我关于它们的发展的预测。

敏捷测试指引（2） - 测试与例子

陈能技
2007-9-24
原文：Agile Testing Directions – Tests and Examples （Brian Marick）
'It all depends on what you mean by home.'
[...]
'Home is the place where, when you have to go there,
They have to take you in.'
'I should have called it Something you somehow haven't to deserve.'
-- Robert Frost, "The Death of the Hired Man"

“这在于你如何理解家的概念。”
…
“家是你随时随地可以去的地方，是他们会让你进去的地方”
“我应该把它称为你不知何故，不值得拥有的东西”
-- Robert Frost，“The Death of the Hired Man”

上次，我画了这样一张矩阵图：



左边是偏向“支援编程”的测试，右边是偏向“批判产品”的测试。但是两种测试的意义和内涵存在很大的不同。

对于支援编程，测试主要作为准备和保证。你通过写测试代码来阐明关于问题的思考。你把它作为说明性的例子来描述代码应该怎样做。幸运地是，它同时是活跃地检查代码的说明性例子，即重新保证。这些测试也会找 bug，但是那是第二目的。

在另一方面，测试是关于暴露主要错误和遗漏。这里，测试的原义就是关于 bug。有其它的意义，但是首要的意义是最主要的。（很多测试员，尤其是最好的测试员，在他们的身上已经融入了那些词语的内涵。）

我想做个尝试。如果我们在矩阵的左边不使用“testing”和“test”这些词语会怎样？如果我们把它叫做“checked examples”（检查例子）怎样？

设想两个 XP 程序员坐在代码前面。他们开始构建一个例子来说明下一步要做什么。他们会检查，在代码还没写之前。（如果写了，那是很特殊个别的情况。）他们编写代码。检查例子是否运行正确，其他例子也保持正确。然后继续下一个例子用于展示下一步应该做的事情。

替换词语有意义吗？是否只是文字上的替换而已？你做一些尝试，然后回答这些问题吧。尝试经常使用“example(例子)”，经常使用让它听起来不会感觉很奇怪。现在，当你坐在代码前面时，是否根本改变了你的观点？是否有了一些不同：在你向客户要求一个例子，而不是一个测试时。加上一些形容词：example(例子)是否看起来更具激发性、更生动、更有深刻内涵？与强大的测试存在怎样的区别？（“强大”作为附加给测试的典型的形容词。）测试人员在 XP 项目中，每个人都在制作例子，没有人做测试，这样是否看起来更轻松些？

敏捷测试指引（3）- 用面向技术的例子支援程序员

陈能技
2007-9-24
原文：Agile Testing Directions – Technology-Facing Programmer Support（Brian Marick）

为了帮助讨论和理解，我把“敏捷项目中的测试”这一主题分解成 4 个区分的主题。今天，我讲一下我们怎样使用面向技术的例子来帮助和支援程序开发。

这里适用的一个是测试驱动开发，在 Kent Beck 的书中，David Astel 最近的书，Phlip,J.B.Rainsberger 接下来的书中都讨论了这种开发方式。我认为测试驱动开发（我现在会叫它例子驱动开发，example-driven development）是可靠的。它不是主流的开发方式，但是看起来要成为主流了。套用 Geoffrey Moore 的话，我认为它正在跨越鸿沟的路途中。

也可以这样说，例子驱动开发已经从 Thomas Kuhn 所说的“革命性的科学”发展到“正规科学”。在正规科学中，人们扩展了特定方法的适用范围。因此我们现在很多人在 GUI 应用 EDD（sic），设法让它与遗留代码（legacy code）一起工作，讨论 mock objects 的好的使用方法，讨论处理私有方法的技术，等等。但是这些都不是重大的发展。

我希望不久的将来会看到更多同时拥有测试和编程的技巧的人被吸引到敏捷项目中来。这些人既不会成为像纯测试人员那样的好测试员，也不会成为像纯编程人员那样好的编程人员。但是那还是不错的，如果你跟我一样相信敏捷项目应该重视通才多于重视专才的话。

我不是这样的混合人物。我没有与纯编程人员一起做很多的结对编程。但是我注意到，在维持编程的进度和目标的希望与确保很多好的测试的主意被考虑到的希望之间存在着紧张的关系。我发现我自己在进入程序员模式和退出并考虑全局的之间振荡。从经验看来，我们需要更好的思路来管理这个过程，以及关于什么类型的“测试思考”在编码过程中是合适的。

也可能有一些测试员在敏捷项目中不担当程序员的工作。不过，他们中的一些人还是跟程序员结对讨论单元测试（关于程序员如何检查代码）。程序员学习如何避免哪些类型的 bug，测试员学习他们正在测试的是什么。不知何故，加拿大的卡尔加里成为了这些活动的温床，我指望 Jonathan Kohl，Janet Gregory 和其他人告诉我怎样才能做好。

我需要强调这些都是关于人的。传统上，测试员与编程人员会有很紧密（或者很广阔）的关系。对于矩阵的支援程序员这部分，我相信传统的关系是不合适的。

我使用术语“checked examples（检查例子）”作为支援程序员的测试。我们可以把这个概念一分为二。一部分作为指引下一步编码的决定。另外一部分是自动化的例子，作为“改变探测器”（change detectors），看刚才的修改是否是你期待的。

通常的习惯是，改变探测器仅仅是保留的代码的保护性例子。（你让你的单元测试套件作为救助，一个对应一个地，在编码的同时测试。）那不是逻辑上的要求。我喜欢先建立一些关于什么时候做其他事情的知识。

例如，考虑这样的维护场景：你首先开发了一些代码例子。一个月后，别人加了一个新的例子并改变了代码以便匹配。很多之前为那部分代码开发的例子都成了“bad examples”坏例子了。（测试失败，但是因为他们现在是错误的，不是因为代码错误。）修正那些例子以便他们一致。我的意思是在下表的左边的事件序列本来是期望与右边的测试一样的。（先看左边栏，然后看右边。）

Example foo written	Example bar written
Code written to match foo	Code written to match bar
Example bar written (foo invalidated)	Example better-foo written (bar is still a good example)
Code changed to match bar - oops, now foo doesn't check out	Code changed to match better-foo (and bar continues to check out)
Update foo to be better-foo	

最近坏掉的例子被重写以便匹配一个理想的开发顺序，在这个过程中不需要重写任何例子。但是为什么？在上表的左列，新的例子 foo 没有用于驱动开发，只是作为检查。对于驱动开发来说是最佳的可能对于检查来说不是最佳的。

假设软件系统开发切面层（shearing layers），切面层的接口通常是不会改变很大的。为了可维护性，在修改时移植坏掉的例子到界切面层是有意义的。取代一个例子对应一个类的一个方法的做法，我们现在使用一个例子对应整个子系统。那可能会很糟糕 - 想想调试 – 但是它减少了维护的负担，甚至能提供关于子系统的行为的完整文档的好处。

我希望人们分清楚两个角色 – 指引不远的将来和重新检查过去 – 会发现有建设性的知识。例如，什么时候编写面向技术的“改变探测器”（而这些与指引编程没有任何关系）可能会有用？

我在上面说的测试驱动开发是“期中一个适合”今天的主题的东西。那么其他适合的是什么？我不知道。EDD 是否是最合适的？（是否最近可能发生革命性的改变？）我也不知道 – 我要依赖那些善于打破旧习的人来帮我找出来。对此我将非常感兴趣。

敏捷测试指引（4）- 用面向业务的例子支援项目组

陈能技
2007-9-24
原文：Agile Testing Directions － Technology-Facing Programmer Support（Brian Marick）

为了帮助讨论和理解，我把“敏捷项目中的测试”这一主题分解成 4 个区分的主题。今天，我讲一下我们怎样使用面向业务的例子来帮助和支援整个项目组的工作（不仅仅是程序员）。

我指望项目例子做三件事：激发程序员写出正确的代码，促进技术专家与业务专家之间的对话，帮助业务专家更快地在产品中实现可能的特性。让我们一步步来分析他们。

激发出正确的代码

这是测试标准的驱动（或例子驱动）设计从内部接口到整个产品接口的直接推导。为了添加一个新的功能特性，先用 1 个或多个例子来说明它应该怎样工作。然后程序员编写代码来匹配那些例子。编写和维护例子直到需要的代码都得到了。然后功能特性也就完成了（到目前为止）。

虽然推断是直接的，但是我们在得到细节之前还有一段路要走，在充分理解实践之前。我在下面会讲更多关于这方面的内容。

促进项目交流

把例子仍过墙给程序员并期待他写出正确的代码与仍给他需求文档一样是行不通的。程序员需要上下文、背景和关于默认惯例的一些知识。他们通过与业务专家交流得到那些东西。例子可以改善交流，通过提供一些东西给大家讨论。它们把讨论具体化。

例子可以提供显著帮助的地方，我想，是得出一个公共的词汇表。我喜欢这个想法：领域方面的术语应该通过转化到代码中的对象来使其“具体化”。不是幼稚地按“写出业务领域并在名词下面划横线”的面向对象的设计方式，而是用 Eric Evans 的领域驱动设计（Domain-Driven Design）的更完善的方式

因此我们必须有一个对领域知识的理解从模糊到具体的过程，转化到 0 和 1 的过程。看起来例子是一个中间步骤，逐渐让知识不模糊的过程。但是，使用例子来指引程序员，还有很多

需要学习和研究。

让可能发生的更加明显

我们希望业务专家在他们意识到产品通过 A 的方式做了 B，通过 X 的方式做了 Y，因此有必要做一些新的 Z，这些他们以前没有想到的事情的时候，发出“啊哈!”的声音。我们也同样希望项目组的其他人有类似的表现，这样他们可以向业务专家提议一些东西。简而言之，我们需要创造力。

可能最佳的释放创造力的方式是让你亲手编写软件并测试。但是另外一种方式是解释例子给其他人听。是否曾经在找 bug 方面存在困难，然后在开始解释你的代码给别人听的时候蹦出很多错误？对于我来说，在写用户文档的时候也会有类似的效果：我使用例子来解释软件背后的基本概念，它们是如何组合在一起工作的。我会经常发现它们不工作。与碰到 bug 的感觉是一样的，虽然我要解释的可能不是一个真正坐在我前面的人，而是一个假想的读者。

因此，我们创建例子和讨论例子的方式可能会加速产品的进展。

我下一年专注的几个研究方向中的其中一个就是面向业务的例子。我已经准备好了$15000 用于调查能很好地使用它们的机构。如果你知道有这样的机构，请联系我。在调查了他们之后，我希望能讲关于下面的一些故事：

1、 关于例子的进度的故事。什么时候创建它们？在程序员开始编码之前创建了多少例子？什么例子最先创建？

2、 关于人们围绕例子的交流的故事。谁参与了？交流的形式是怎样的？谁把例子写下来？业务专家来写的时候是怎样的？程序员来写呢？测试人员呢？（当不同的人之间切换时人们注意到什么了？）在把例子转换到代码的过程中例子变化的情况是怎样的？

3、 关于面向业务的例子和面向技术的例子（单元测试）之间的交互的故事。程序员在什么时候、怎样把注意力从一个转到另外一个的？面向顾客的例子是否经常检查？例子是否会从一个分类转到另外一个分类？

4、 关于面向业务的例子影响设计和代码结构的故事

5、 关于 FIT 的故事。对于什么样的系统最合适？FIT 最好的一个特性之一是它鼓励把说明性的文字环绕着例子 – 大家怎么利用它呢？当人们从其他方法（用脚本语言编写例子）转移到 FIT 的时候，学到了什么东西？而转向其他方向的人们学到了什么？当开发一个项目的词汇表的时候 FIT 和脚本语言比较起来会怎样？

6、 关于推动代码的例子（“…这里是功能 X 的另外一个重要的方面”）和排除 bug 的例子（“…不要忘记产品要在这种情况下工作”）之间的平衡的故事。怎样的 bug 需要预防，怎样的 bug 应该留给产品批判的阶段（矩阵表的另外一半）？（参见 Bill Wake 的"generative" and "elaborative" tests。）

7、 关于检查性例子与变化检测者之间的区别的故事。在面向业务和面向技术方面是否存在不同。

只有当我们把这些故事都收集起来之后，关于面向业务的例子的实践才能被很好地理解，才能成为惯例，就像面向技术的例子的实践一样。

敏捷测试指引（5）- 用面向业务的例子批判产品

陈能技
2007-9-24
原文：Agile Testing Directions – business-facing product critiques（Brian Marick）

为了帮助讨论和理解，我把“敏捷项目中的测试”这一主题分解成 4 个区分的主题。今天，我开始讲矩阵的右边：产品批判。

使用面向业务的例子来设计产品是好的，但是假设例子是错误的怎么办？谁都会犯错误。业务专家会忘记一些用户真正需要的东西。或者业务专家错误地表达了需求，而程序员却非常忠诚地实现了错误的东西。

那些错误，如果记起来了或注意到了，则可能会当作 bug，也可能被认为是需要的功能特性。但两者的界限很模糊。我会简单地把它们叫做“问题”。

问题是如何引起项目组的注意的呢？
- 很多敏捷项目组会在一个迭代结束时向业务专家和感兴趣的外部人员演示软件系统。这时候会是激怒某个人的时候，“噢…我是那样说过，但是我不是这个意思”。
- 敏捷项目喜欢频繁地发布软件给它的用户（可能比用户希望更新的频率还要频繁）。当用户试用产品的时候，他们会指出存在的问题。

这些反馈循环很紧凑，比传统的项目要紧密，因为敏捷项目喜欢短的迭代。但是它们不是最理想的。业务专家可能由于过于靠近项目而不能以新的、没有偏见的眼光去发现问题。用户通常不报告他们在软件使用中发现的问题。当他们反馈问题时，报告得又不够专业以致没办法执行。而反馈的周期不能像敏捷项目希望的那样的频繁。可能仅仅是针对一行代码的改变的反馈，但是需要等上 3 个月才能从用户那收集到。

因此，我们需要额外的产品批判形式 – 能注意到用户会怎样，而且能及时注意到。

产品批判的方式拥有前期创建的例子所不具备的资源：一个新的迭代版本的真正工作的软件。当你描述某些目前不存在的东西的时候，你是在脑海里操作一个抽象的东西，一个你想象中的物品。当你亲手操作一个产品的时候会激发不同的理解和判断。你可能注意到，当试驾一辆汽车的时候，你不会专注于它的规格说明。操纵是与思考不一样的。

因此，面向业务的产品批判应该专注于操作方面，尝试逼近真正的不同用户的体验。就像 James Bach，Cem Kaner，Elisabeth Hendrickson 他们所说的探索性测试（Exploratory）的形式。

进一步的，我发现我们在尝试至少 5 种类型的探索性测试：
1、 一个探索性测试员
2、 结对探索性测试员。James Bach 和 Cem Kaner 可能在这方面有最多的经验。
3、 探索性测试员与一个程序员结对。Jonathan Kohl 会在 2004 年 1 月的 STQE 杂志（http://www.stqemagazine.com/）有一篇这样的文章。我在这方面没有什么经验，程序员也喜欢这样的方式。值得注意的是，当我在 RoleModel Software 进行这种方式的时候，它导致了一个有趣的并且有用的关于基础程序的讨论。那样的话，它成了一种类似回顾的方式，这进一步让我相信它是迭代结束时很好的一种活动。
4、 让探索性测试员与项目中的业务专家结对
5、 让探索性测试员与感兴趣的非参与者（用 SCRUM 的术语来说就是“chickens，小鸡”），例如经理主管、新用户等等。

对于每一种，我们应该探索关于什么时候测试员应该是项目组外的人的问题。这些外部的测试人员不会存在偏见和先入为主，但是存在缺点就是：他们需要花更多的时间来学习一些基本的东西。那也会使发现的问题存在一些偏离。

当我一年前开始在敏捷项目谈论探索性测试的时候，我想它会在发现 bug 的同时对提出一些大胆的关于产品的新构思有启迪作用。一个过程能发现两类问题。有一段时间，我把它叫做“探索性学习”来强调它的扩展的角色。

后来我推断这两个目标不能很好地走在一起。因为找 bug 实在是太诱人了 – 对于功能特性的思考会在探索性测试的过程中迷失。有些时候能同时出现，但是不足够。所以我想可能需要一个单独的功能特性的脑力风暴活动。关于这一点，现在我还没有什么特别好的主意。“需要进一步的研究”。

敏捷测试指引（6）- 用面向技术的例子批判产品

陈能技
2007-9-26
原文：Agile Testing Directions – technology-facing product critiques（Brian Marick）

为了帮助讨论和理解，我把“敏捷项目中的测试”这一主题分解成 4 个区分的主题。今天，我将完成矩阵的右边部分：面向技术的产品批判，而不是面向业务的。

我选择探索性测试作为面向业务的产品批判的工具。但是虽然它也可能找到安全性问题、性

能问题、通常在压力下才出现的 bug、可用性问题（例如对色盲人士的适用性）等，但是我不会依赖它来完成这些方面的测试。而且，这些非功能性的问题或非功能性的需求很难用例子来详细说明。所以看起来预防或找出这些 bug 目前为止还未纳入到我们的故事中来。幸运的是，还有矩阵的最后一个四分区之一。

我想关键是，找出这样的非功能需求的 bug 更多的是技术性问题。你不能随意地就能知道一些安全性的知识。性能测试可以说是“妖法”。可用性不是个“需要你知道很多计算机知识”的技术性的话题，但是它要求你知道很多关于人的知识（Mark Pilgrims 的 Dive Into Accessibility ，见 http://diveintoaccessibility.org/，是个针对这方面的丰富知识的入门介绍）。

虽然我老是说敏捷项目需要“通才”，但是这里的区域则需要的是“专才”。如果安全性是对于你的项目来说很重要的话，找个安全专家，在很多安全领域拥有丰富经验的人。（也就是说，安全知识要比领域知识重要。）这些人能教会项目组怎样构建安全的产品、并测试安全性是否被构建到产品中。

（有趣的是：这些区域给我的印象是在设计和批判的角色之间的分离没有产品功能开发那么明显。Jakob Nielsen 既写关于可用性设计的东西，也写可用性测试方面的东西。安全性方面的人物也是类似的，像 Gray McGraw 和 Bruce Schneier，除了 James Whittaker 好像专注于安全测试方面。我不知道我的印象是否正确？对于性能测试人员好像没那么正确，虽然我知道很多优秀的性能测试员也能出色地设计出高性能的系统。）

因此，敏捷好像没有给这些人带来什么东西。这些专家继续存在，他们发展成不同的等级，他们值得进一步的发展，他们掌握了很多好的东西。可能会不如想象中的正确，但是我想他们应该就这样继续保持着。

看起来我好像完成了我的关于敏捷测试的未来指引的系列。但是还有一个问题：究竟，在敏捷项目中是否应该有测试员？对于这是个热点问题，我应该覆盖到。

敏捷测试指引（7）- 敏捷项目中的测试员

陈能技
2007-9-27
原文：Agile Testing Directions －Testers on agile projects（Brian Marick）

敏捷项目中是否应该有测试员呢？

首先：替换测试员的是谁？是让非测试专业人员（程序员、业务专家、技术文档编写人员等）来执行这样的活动：帮助创建指导性的例子和对产品进行批判？还是，反过来，让测试员来做编程、业务分析、技术写作等工作呢？把“测试”仅仅作为技能的集合而存在，在项目中拥有充足的数量，用于服务所有需要这些技能的任务。

为什么非专业测试员会是个坏主意？这里是一些可能的原因：

l 测试技能很难学到。如果你尝试作为测试员同时是程序员，或者作为测试员同时是技术文档编写人员，你不会拥有所需要的最少的技能来成为足够好的测试员。

l 假设你是世界上最好的篮球运动员，同时是最棒的洗车工。你可能还是愿意让别人帮你洗你的车，因为比起你自己洗车省下的钱，你可以赚取更多的时间来打篮球。这就是相对优势的一个例子。因此，为什么不让一个懂得测试的诀窍的人只是做测试，而让一个在编程方面相对强的人专注于编程呢？

l 测试虽然说不上是是一种天生的技能，但是某些人就是喜欢吹毛求疵，有些人则不擅于批判。

l 很多人在找自己工作的错误时会有困难。因此把测试和其它任务混在一起的话会测试得很糟糕。中间存在太多情绪上的利益冲突了。

l 测试员能从“有用的无知”得到很多益处。不知道实现的细节使得他们更容易从用户的角度看什么类型的错误是用户容易犯的。

论据
让我首先解释一下“最小所需技能”和“相对优势”。这些论据在面向技术的产品批判中是最强的，像安全性测试或可用性测试。在一个实际的项目，我一定能看到专门的安全性测试员。在小一点的项目，我能看到临时出现的安全性测试员。

对于我在面向业务的产品批判中所依赖的探索性测试员，我不那么确信。探索性测试员发现的这么多的 bug 中，很多是程序员可以预防的，如果他们能经常看看那些 bug 并使它们内在化。把 bug 内在化的最好的途径是把程序员纳入到寻找 bug 的行列，而不仅仅是修改 bug。如果测试人员来写其中的一些代码，则 bug 会少些。因此这个论据是反对拥有专业的测试员的。

换言之，我认为人们都有最小所需的探索性测试技能。

我想相同的原因适用于矩阵的左边-面向技术的检查性例子（单元测试）和面向业务的检查性例子（客户测试）。我把这些方面的东西教给测试员。程序员也可以做。业务专家也可以做，虽然可能很少有人有机会达到最小技能的水平。那是为什么面向业务的例子是被团队创建的，而不是仍过墙的。实际上，团队沟通是如此的重要，以致应该把所有相对优势的影响淹没。

现在，让我们看看所谓的“天生的能力”。当 Jeff Patton 给我们展示以使用为中心的设计的例子时，其中一个练习是为一个假想的会议文件评审系统创建角色。我当时创建了类似“不情愿的文件评审者”、“过度疲劳的会议主席”、“拖延的作者”等角色。有些人评价说，“你能看得出 Brian 是个测试员”。我们都轻笑为什么我会倾向于悲观的例子。

但是最重要的是 – 那是学术行为。我这样做是因为我有意识地去找出那些会跟开发人员希望不一致的操作系统的人。我的预感是我天生比别人更倾向于批判，但是我学着成为一名合适的测试员。我想普通的程序员也可以。我还没有碰到过一名程序员是过分乐观主义者，以

致认为其他人的软件是世界上最好的。

是你自己引起的所谓“情绪上的利益冲突”。我曾经让 Elisabeth Hendrickson 对我的一个程序进行探索性测试。我颇为得意-我认为我的面向技术和面向业务的例子都通过了。当然，她很快地发现了一个严重的 bug。我不仅仅是震惊，而且做出了很多测试员都很熟悉的自我保护性的反应。

后来我在最后期限之前做了一些探索性测试，意识到我在用户界面的“不重要的”部分编码工作做得不够好，但是我感觉不情愿去攻击这些 GUI，因为我实在是不希望修改 bug。

因此这是个真正的问题。我希望我们可以通过一些实践来减少它。例如，就像结对编程的程序员倾向于保持诚实地重构，那么在探索性测试时保持诚实地攻击代码。在进度压力下不愿意重构 – 导致的是设计的“债”的积累 – 不是一个可以永远摆脱的问题，但是项目组要学会处理它。也许对于情绪上的利益冲突也一样。

跟情绪上的利益冲突相关的问题是“有用的无知”。假设是在第五次迭代。由测试员/程序员/其他人员组合在一起，从开始就工作在一起。当对产品进行探索时，他会有开发习惯。如果有两种方法做某件事，他会选择同一个。当他使用产品时，他不会犯很多概念上的错误，因为他知道产品应该是怎样工作的。他的团队已经写了很多指引性的例子 – 在他们做这个的时候，已经建立了一个清晰的“理想用户”应该是怎样的一个模型，他们会在想象其他类型的用户会怎样方面碰到麻烦。

这是个很难解决的问题。角色扮演能帮助解决。Elisabeth Hendrickson 教测试员在测试过程中要时不时假定极端的人物。如果 Bug Bunny 来使用这个产品会发生什么事情呢？他是个麻烦制造者，总是探究别人的弱点，总是愚弄权威。想象一下卓别林在摩登时代的角色：天真、毫无准备、被迫工作得更快。另外一个有用的技术是 Hans Buwalda 的肥皂剧测试方法（Soap Opera Testing）。

我希望这些技术能帮助解决问题，尤其是结对在一起时（互相之间会感染）。但是我不禁要想伪造的无知不能替代真实的东西。

组队

因此，是否应该包含测试员在敏捷项目中呢？要看情况而定。但是如果我负责一个重要的产品的敏捷项目组的组队，我会考虑以下方式作为默认的做法：

l 我会找一到两个拥有丰富测试经验的人。他们应该懂一些编程的知识。他们应该擅长于跟业务专家讨论并很快地获取一些领域知识。首先，我会依靠他们来确保面向业务的例子可以很好地工作。然后我会期待他们学习更多的编程，编写基础代码，教程序员，成为程序员一样的人。个性非常重要。他们必须喜欢新奇的事物，他们不应该把他们的个性情绪化地包含到工作中，他们应该习惯于服务其他人。

l 如果这些人在探索性测试中做得很出色应该受到赞赏。但是，不管怎样，整个项目组都会得到关于探索性测试的培训。我会让外部的探索性测试教练定期地来造访。他们既会扩展培训，还会做一些探索性测试。作为一个持续的监督，用于防止项目组过于习惯于产品而不能找到足够的 bug。

l 对于非功能缺陷，像可用性、安全性、性能比较看重的产品，我会购买专门的技术（定点的顾问、造访型的顾问、或者招聘这方面的专门技术人员）。他们会对创建产品提出建议、培训项目组、测试产品。

l 我会极力让真正的用户参与（不仅仅是代表用户的业务专家）。可能会让项目组成员到用户那边，而不是反过来。我会让项目组成员把自己想象成人类学家来研究产品所在的领域，不仅仅是倾听 bug 和功能特性的请求。

是否有测试员在项目组中？谁关心这个？- 总之会有好的测试，即使指出某个活动中有测试会日益地困难，像说“那里，就在那里。那就是测试，没别的”一样。

敏捷开发中的 7 种测试类型

在 08 年的 STP 第 6 期杂志，Glenn Jones 在《Fly into agile development with agile testing》一文中把敏捷开发中的测试分为 7 种类型：
（1）自动化回归测试（Automated regression test）
运行自动化测试代码来验证当前的修改没有破坏已有的功能。

（2）单元测试（Unit test）
验证单元级别的代码工作是否正常。

（3）公共 API 测试（Public API test）
验证被第三方开发人员调用的 API 可正常工作，并且得以文档化。

（4）私有 API 测试（Private API test）
验证内部使用的 API 工作是否正常。

（5）命令行测试（Command-line test）
验证在命令行输入的命令工作正常。

（6）用户界面测试（User interface test）
验证界面层的功能是否正常。

（7）“狗粮”测试（Dog-food test）
这里用了一个有趣的名字“Dog-food test”，自己的“狗粮”自己先尝尝！在企业内部使用自己开发的产品，通过这种实际地使用来确保功能正确，满足使用要求。