软件测试外文翻译版
电子杂志
创刊号

出版单位： 51testing 软件测试论坛软件测试外文翻译版
2005 年 9 月

创刊语

作者：skinapi

一晃软件测试外文翻译版已经创版一年了，在这一年的时间内，依靠大家的共同努力，翻译了不少好的外文资料，为广大测友提供了近距离了解和学习国外测试先进思想的机会。为了便于大家的收藏和学习，希望通过电子杂志的形式不定期的将版内好的资源收集起来，另一方面也是希望能促进大家积极参与版内的翻译和学习活动。

杂志中暂时考虑包含三方面的资源：

- 原创翻译：版友自己翻译的作品，将以专栏的形式出现，鼓励大家树立个人品牌。
- 原创读后感：版友自己写的读后感，也将以专栏的形式出现，这一部分是着重鼓励的，因为翻译的目的是为了让更多的测友学习。
- 网络资源：包括国内和国外好的与软件测试外文资料相关的网站等资源。

为了让大家能更好的参与软件测试外文翻译版的活动，考虑采取以下一些措施：

- 好的外文资料将通过邮件列表的形式通知给大家，如果哪位版友有兴趣翻译请发站内短信给我，将根据短信的先后顺序选择由谁翻译。
- 翻译完成后就可以发到版里了，可以考虑帖子标题前加上***专栏字样^_^。
- 相关资源积累到一定数量就将以电子杂志的形式进行整理，然后和大家分享。

大家如果有什么意见或者好的想法都可以站内短信通知我。总之是希望我们大家能一起把外文翻译版办的更红火，并且都能不断学到新东西、得到提高。

原创翻译

[connie专栏]

如何写测试计划

作者：Stevan Zivanovic
翻译：connie

当我访问一些站点的时候，发现一个很常见的问题，人们总是无法说出为什么他们要测试他们正在测试的内容。幸运的话，我将会得到这样的答案：这些都是在测试计划中计划好的。但是当我看测试计划的时候，却发现辛苦工作换来的 20-30 页计划文档的内容，却仍然没有告诉我他们到底在测试什么以及为什么要这样测试。

现在我并不是打算重新分析 IEEE 829 模板的测试计划。这是一份非常棒的文档，能给我们提供很多帮助。我想看一下整体的目的以及计划的必要性。撇开整体的开发方法不谈，如 SCRUM 或 Waterfall，我们需要花费一些时间来给测试的方法下定义。

无论这是一个活泼轻快的故事性记叙或是一个非常正式、规范的详细的项目，好的开发都要从计划开始，这一点也适用于测试。为了做好这项工作，需要对相关的及适当的测试进行仔细的思考。以下是我们在下定义时能用到的一些好问题：

我要测试的是什么？

这个问题的内容包括与之相关的软件、基础结构以及相关的系统和应用。你可以把你将要测试的系统用图解的形式表示出来，用来表现哪个软件将会依赖哪部分硬件以及如何与其他应用建立连接。

怎样进行测试工作？

我要用什么方法来开展测试工作？我是否会用到探测性测试、测试驱动设计、确认测试或者其他任何一种方法，又或者是这些方法的综合。这样做可以帮助我们突出重点，并合理利用资源来支持我们的测试活动，包括人力和技术资源。

测试的目的是什么？

我是否需要保证关键系统的安全性满足运行条件？应用程序是否已经足够稳定以供在某地发行试用版？或者更加基本的，我的客户以及商业需求到底是什么？考虑清楚这些以后将决定测试需要进行的深度以及精确程度。

需要测试什么内容？

为了回答上面这个问题，需要对系统的各个方面都有认识，你可以在需要测试的内容里面突出关键部分。可以列出关键及风险内容、属性、场景或者测试技术（如，我知道我需要进行一些性能测试，但是我还不是十分清楚我到底需要测试什么）

通过回答上述所有问题并将他们与其他各方面联系起来，那么在向所有利益相关者阐明并清楚地定义测试什么内容和为什么这么测试时，就很快可以得到重要的改善了。

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=15857&fpage=1

如何在快速变更的环境中开展测试工作

作者：Randall W. Rice, CSTE, CSQA
翻译：connie

如同一句古老的谚语说的，世界上唯一永恒的东西就是改变。在软件开发过程当中，传统的开发模式的一个缺点就是他能够呈现较少的改变甚至没有改变。真实世界每天都在改变。正因为此，其他的开发模式，如快速应用开发（RAD）已经提升到可以包含改变并在计划过程中使用改变来精炼开发。

在 RAD 模式帮助软件开发者更早的得到更新的版本并更快的运行程序的同时，他也给测试人员带来很多令人头疼的烦恼。因为每个改变都可能产生新的缺陷。找到新缺陷的唯一的方法就是使用回归测试，回归测试可以完整的重复先前的测试过程并与得到的结论进行比较，以便找到不同的地方。

接下来就产生了 2 个问题：1）能否在快速变更的需求过程中进行彻底的测试？2）在测试快速变更的软件时需要采用什么策略？

能否在快速变更的需求过程中进行彻底的测试？

答案是，不能。这是一个恶作剧式的问题，因为在大多数情况下，即便是在稳定的环境当中，进行彻底的测试都是不可能的。我们可以将这个问题改成这样，“能否在快速多变的环境中进行有效的测试？”我们能否期望在测试软件时能够充分有效的利用人力及其他资源？我们能否期望在测试中找到预期数量的缺陷？

通过观察使用 RAD 模式进行开发的项目，我发现要找到缺陷和得到任何程度的成效的本质的部分就是测试过程。由于在软件开发中，大多数我们使用的测试标准是规范是使用不重复的过程，所以很多人在测试 RAD 环境的软件时，采用的方法是平时我们在其他环境中使用的方法——在各个不同的地方使用较少的用例来查找问题和缺陷。

我们可以采用什么样的策略呢？

研究每个单独的环境当中需要采用的策略将会耗费过多的时间，这里有一些通用的策略，可以在快速改变的测试环境中使用。

首先，你必须接受一个事实，那就是不可能花去 6 个星期的时间来测试一个天天都在改变的软件。这就意味着你需要限定测试过程使得他可以快速并有效的进行。

做一个风险程度的估定。知道风险的标准是关键所在，因为你需要在一个较短的时间内来确定你要测试内容的优先级。风险越高，册是的优先级越高。

自动化测试过程。回归测试工具帮助你在短时间内进行重复的测试工作。好的工具需要在软件及培训方面进行重要的投资，但他不能一天不间断的工作。有一些事情需要在自动测试之前考虑并提前做好。

你必须拥有软件的工作标准文档来跟将来的测试进行比较。

你必须定义需求，测试用例和测试环境。工具只能记录并回放基于操作者的操作行为的活动。

数据维护是关键。你将怎样维护这些测试数据？例如，如果你使用

回归测试工具来添加一个记录，脚本文件将得到“重复的记录”这样的错误。

这将耗费相当多的时间以及金钱来使得这个工具融入到你们的公司。在此之前还必须对相关人员进行工具使用的培训。因此，必须使人们相信，跟目前工作当中需要建立测试脚本和测试用例这些琐碎的事情相比，这样做对于长期利益的发展是非常有价值的。

在快速多变的环境中进行测试并不是不可能的，但是他需要能够做出快速的响应，灵敏的进行操作，并且与变更的轨迹保持一致。那些不愿意考虑新技术，如自动化测试的公司、组织将不能在快速多变的开发环境中开展有效的测试工作。这就好比是手工工具来盖一座房子，很显然他最终是无法成功的。

在快速多变开发环境中同样也需要一个对于机构和过程的新观念。单独使用工具并不是最终的答案。必须有一个能快速执行并且充分有效的利用人力资源和时间的操作程序。要达到工具使用、操作程序以及人力的完美结合，必将是一个很大的挑战。如果你想得到用户验收测试相关的更多的培训及方法，可以通过以下的链接联系e-mail Randy Rice.

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=15674&fpage=1

[skinapi 专栏]

我要告诉测试新手的

作者：Randall W。Rice, CSQA, CST, CSTM
翻译：skinapi

前言

因为已经带领和训练测试团队多年，所以按惯例我总有些东西确定需要传达给测试新手。不管你是一个测试新手还是一个经验丰富的测试专家，都有不少有益的东西需要牢记在心。

1、你是一个检查者，你不需要为质量负责

很多测试人员误入歧途，不明白他们是评测产品的而不是控制产品的。这两者之间有着天壤之别。例如，一个测试团队花费好几周时间测试并发现很多缺陷，只是为了看着管理层决定发布一个有已知严重缺陷的产品。测试团队经常会感到士气受挫，置疑他们测试的目的。

我询问团队中的成员他们是否被支付薪水了，通常得到的回答都是“是”。我又询问他们是否尽力去做工作了，再一次，通常得到的回答都是“是”。我于是告诉他们，“你们做了你们的工作。你们尽力测试，发现了缺陷并进行了上报。那么现在可以回家休息了。实际上，作为一名测试人员唯一失败的地方是不上报一个已知的缺陷。”

这不会提高士气，但却有助于事情向正确的方向发展，特别是能让人不用每天晚上都在家接着办公。

很多测试人员，包括我，当我们刚开始测试工作时，似乎会觉得自己对我们所测试的系统应用的质量负责。尽管这个工作的出发点是让人钦佩的，可实际上我们测试人员对于产品的质量基本没有控制能力。也是由于这个原因，测试人员不为质量负责。现在问题是管理层并不总是能看到这种区别。所以经常看见管理层提出类似于“我们付钱给这些人不是为了获得高质量的软件吗？”的问题。

2、缺陷都是有价值的

每一个缺陷都是深入了解和提高的机会。我们可能只有一次机会观察到一个缺陷，所以我总是告诉测试人员始终保持高度注意力，不要为测试的乏味所折磨。

缺陷信息可能是可获取的项目数据中最有效的资源之一。但是这都取决于我们能多好的捕捉和传达我们所发现的缺陷的相关信息。

每个缺陷都会花费整个组织的金钱。如果我们不能从中更进一步了解产品，我们会浪费大量时间和金钱。当我们把一个错误转换成一次深入了解的机会时杠杆作用就出现了。让我们面对它——有些教训只能通过经历来学习的。

由于一个缺陷而责备谁不会有任何好的作用。责备只会让士气低

落、沟通中断。这就像不断鞭打一匹死马希望它能活过来一样。

3、你报告第一个问题之前一切都是美好的

这就是一个测试人员所面对的现实。你可以计划测试，获取所需要的资源，看起来所有人都站在你这边。可当你报告第一个问题之后，事情就开始变得紧张了。

出现这种态度上的突然变化的原因是现在你在批评某些人的工作了。自尊心使得自我收到伤害，关系变得紧张。有些情况下自尊心是值得期盼的，只要知道当你开始发现问题的时候态度有可能变化就可以了。

我经常建议测试人员做的一件事是读一读一些你过去写的缺陷报告，假设自己是接收缺陷报告的人。你会发现自己需要更老练一些。写一个没有任何挖苦语句的缺陷报告可能没什么乐趣，但它的确有助于和开发人员之间保持一个好的关系。

4、只能测试你能观察的

你可能总想测试一些真正有创造性的用例，但如果你没有办法观察到结果，那有什么意义？尽管有些应用让你能观察到很多，但仍然有你没办法接近的，例如结构、隐藏的对象、后台进程等。

5、别忘记你是怎样到一个地方的

我不是在谈论知道为什么你走进一个房间，而是在测试时执行的步骤。对于测试新手常见的是发现了一个重大的缺陷，但却无法复现它以便定位解决。这样你只会觉得不舒服，不知道自己到底是真发现了一个缺陷，还是说仅仅是错误的使用了应用。

你能用来跟踪你的测试步骤的方法有测试脚本、测试记录、敲键记录器如 Spector 和屏幕视频捕捉工具如 Hypercam。

6、标准和流程是你的朋友

尽管标准和流程让一些人觉得受限，但它们为你的工作提供了有价值的指导。不要拒绝标准因为它们是详细的、具体的。因此用它们指导自己更快、更一致的完成自己的工作。

7、没有足够的时间用于测试

几乎每一个测试人员都抱怨没有足够的时间用于测试，但实际情况是测试任何东西到完整的程度都是不可能有充足时间的。当你充分考虑软件的特性如可用性、安全性、兼容性、互操作性等时这一点尤其正确。

不要再抱怨缺少时间，学会根据风险来进行优先级排序，把注意力都放在对管理层很重要的应用目标上。有时候我们测试的内容超出了我们需要测试的，因为我们的目标偏离了产品的价值。

8、你不可能发现所有的缺陷

如果你测试的东西后来有缺陷被发现，不要变得气馁。你可能已经做了非常全面的工作，获得了高水平的缺陷移除，但 100%都是不可能的目标。

9、保持幽默感和对前景充满信心

经常微笑、保持健康可能是你最好的生存方式。如果你正处在困难条件下，请相信，这一切都将过去。

10、争取做到最好而不是完美

测试新手经常会陷入追求完美的过程中，认为 100%的正确才是标准。我曾经也是受害者之一，但要为自己辩护的是，我以前深受 80 年代后期类似于“ 99。9%还不够好 ”的 TQM 帖子和文章的影响。

追求完美的问题在于它会让测试进程变慢，将担心引入你所做的一切，使得你对别人更挑剔，而且通常会让你的朋友和家人感到失望。

当然，没人愿意犯错误，但他们稍不注意就出现了。想不犯错误就是否认现实。争取做到最好是一种好的习惯，表明你对工作的态度和投入程度。如果你想努力做到最好，你就会往前再多走一点。

根据我的观察，大多数人看到错误或者经历失误时都是很宽容的。人们最关心的是你对待问题的反应。

11、开发人员不是敌人

需要整个项目团队的努力才能递交高质量的产品。有时候似乎开发人员不太关心质量，这个时候事情背后可能存在隐情。这时候你需要更

好的和开发人员合作而不是反对他们。要始终牢记良好的交流是一个项目成功的关键因素。当你和开发人员站到对立面时，交流就停止了，你测试所需的很多信息也无法获取了。

12、建立和维护一个私人的交际网

你的私人和工作关系是一个很重要的资产。无论时当你有工作时还是当你没工作时他们都是一个很好的支持系统。找一个好的指导者，而当你学到足够的东西时成为别人的指导者。

13、持续锻炼自己的技能

你的技能把你和别人区分开。始终通过参加专业会议、获取认证、阅读专业资料等来不断学习。我给自己制定的目标是每周至少读一本和个人发展以及职业发展相关的书（测试、领导艺术、商业、IT 等）。

一个个人发展方面的专家说过如果你每天在任何特定的主题上花费 30 分钟进行阅读，五年之内你肯定能成为这个主题方面的专家。这一点对我是起作用的——你也可以试试。

另一种让自己始终内行并建立网络的好的方式是活跃在一些 QA 或者测试论坛上。

14、当前进变得困难，懒惰就需要创造力了

当我第一次成为一个测试团队负责人时，我用这句话做了一个字条挂在我的桌上。它不断提醒我把创造力作为我解决问题的一个杠杆。

学着从一个新的有创造性的方式来看待问题。你可能有一个好的测试计划，但你如何应付各种变化呢？弹性是一个优秀的问题解决负责人的关键特性。

15、简单并不总是很容易

我们测试中做的很多工作看起来都很简单。但是，挑战在于保持努力的连贯性。

有些解决问题的方式刚开始看起来很简单，但不要由于它简单和明显就丢弃任何一种想法。同样，不要低估实现一个简单想法所需要付出的努力。

一些看过我和 William E。Perry 合著的书“ Surviving the Top Ten Challenges of Software Testing ”评论说这些挑战都很简单且很容易解决。这就让我奇怪为什么人们还在年复一年的提出“ 人的问题 ”。我认为在大脑中产生想法比实际实现出来要简单的多。

结论

智慧比知识更重要。你可能已经学习了大量测试技术，但如果你没有足够的智慧判断什么时候采用它们，没有从整体上理解它们，你应用它们的能力将受到很大限制。对任何都有涉猎的你存在的一个问题是“ 你不知道什么你不知道 ”。智慧帮助你明白你需要知道哪些东西才能成功。

上面罗列的这些都是我希望我刚开始测试时都已经完全认识到的。我希望它们对你有帮助。

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=11538&fpage=1

[fyxu333 专栏]

测试未来的预测

作者：Harry Robinson
翻译：fyxu333

摘要：一年将尽，心理学家或者一些博学者们，又将对 2004 年或者更久的将来作出预测。在这次的周末专栏中，Harry Robinson 将向我们讲述他对测试未来的预测。

“ 预测是件很难的事情，尤其是预测未来 ” —Yogi Berra

每年十二月，小报的“ 未来预测者 ”们会向大家切揭示即将到来的一年将要发生的事情：“ 麦丹娜将要乘坐航天飞机”，“ 美国将迁都 Wichita”，等等。我将加入这个潮流，对软件测试何去何从做一个我自己的预测。并且我希望，我的预测费用能够比我的那些值得尊敬的小报

同事更高些。

我的主要预测就是，将来的软件测试与现在的软件测试看起来很不一样。原因很直接：今天的软件测试很大程度上是臭名昭著的：软件测试参与到项目中的时间太晚、贡献太少、花费太高。如果我们关心我们产品的质量以及我们的账本底线的话，我们就需要重新思考测试和质量的方法。

即使遭到一致反对，我也要说：更好的方法，对测试人员更好的培训、更好的欣赏将改革软件产业。具体地说，诸如可执行的说明书、基于模型的测试产生、BUG 预防、系统模拟这些技术，将在这场演变过程中扮演重要的角色。

下面就是我们在将来的几年里可能看到的情形。事实上，某些趋势已经开始了。

测试人员，需求撰写人员和开发人员，都将看到自己是其中的一份子。

测试人员帮助需求撰写人员

测试人员与需求撰写人员共同工作，在需求完成以后，审查以及理解需求。早期的审查以及建模可以暴露很多关于一致性、完整性和模糊性的 BUG，这个时候修补这些 BUG 付出的代价还十分小。

需求撰写人员帮助测试人员

测试小组建造模型，用于产生对其产品行为的测试。需求撰写人员审查模型，以确保他们充分覆盖了产品特征集。这样产生的测试模块将成为一个“ 可执行需求 ”。

测试人员帮助开发人员

因为需求清楚，毫不含糊，开发人员更好的理解了他们的代码将要完成什么。

在正式的将代码提交做测试之前，测试人员提供给开发人员一些模型，以便开发人员可以在自己的代码中实现它们。

开发人员帮助测试人员

基于”特征对特征”这样的方式(防止以往的“ 后期才介入开发,一股脑找出产品问题”的方式),开发人员和测试人员共同保证代码易于实施自动测试。开发人员的代码中处处都是易测试性的开关,使得错误检测更加容易。

测试人员帮助测试人员

测试用一种高级语言来模拟,因此别的特征的测试小组(甚至别的产品的测试小组)可以复查和改进测试模型。这就形成了一个测试专家的共同体。

方法日趋完善

BUG 预防和早期检测

因为现在把重点放在产品交付的质量上来了(而不是在于找到了多少 BUG), 预防实践和静态分析仪这样的检测工具将成为主流。

仿真测试

仿真工具变得很普遍,使得仿造计算机环境变得容易起来。在开发过程的早期就可以进行意外和错误流程的测试。代码稳定后,再用真实环境验证仿真是否准确无误。

及时的测试用例

庞大的测试用例管理系统将成为昔日的东西,大量的测试用例生成了却没有被使用。测试用例将不再像腐烂的存货一样被收藏起来,因此,让测试用例保持最新变得容易起来。

积极的方法

误导人的方法,比如计算 BUG 的数量、计算测试用例的数量,将不复存在。有用的方法,比如需求覆盖、模型覆盖、代码覆盖将驱动项目开发。

更少更精的测试人员

机器将代替测试人员做大部分他们以往创建测试所做的繁琐工作,测试小组需要比以往更少的测试人员,留下来的测试人员将是经过更多高度培训过的。他们所做的工作将更加有趣,因为在测试中他们将致力于更大的问题,而不是在抱怨中艰难地开展工作。

更多更好的测试

测试人员将可以在一天中进行成千上万的测试,所以,如何首先运行最有用的测试将成为一大挑战。相关的工具将允许测试人员为他们的测试区分优先级,以及将测试目标放在那些最易出现重大 BUG 的地方。

测试人员的角色更换

测试中界限模糊

在测试领域工作使得专职测试的人员和专职创建测试工具的人员界限模糊,一个既是“ 通过程序破坏事物的测试员”又是”创建程序用于破坏事物的程序员”的专业出现了,——关于如何称呼这个新的专业，新闻圈内的人们还在进行着无休止的争论。

测试与开发界限模糊

测试人员与开发人员一前一后，共同创造可测试的、高质量的代码。测试人员帮助开发人员消除需求中的问题，使得开发人员的工作更易完成，同时，开发人员写出更清晰、可测性更高的代码，使得测试人员的工作更易完成。

顾客反馈与测试合为一体

交付的产品质量更高。测试人员进行根本原因的分析，我们会问比如“ 我们怎么会遗漏了这个 BUG 呢？”或者“ 我们将来如何防止这类 BUG？”这些问题，我们的工作就是使顾客满意。

新的挑战出现

复杂和相互关联的计算机世界使得了测试安全这一类的新问题让测试人员不断努力工作，但这没关系——因为这些挑战使测试人员精力充沛。

测试人员获得尊重

测试人员将不再是在最后时刻才被叫来“ 对产品狂轰烂炸”，他们将在整个软件开发过程中提供一个可见的、重要的、增值的服务。人们意识到，测试是有益的、有趣的甚至富有乐趣。

测试变得流行

软件测试人员开始扬眉吐气，而且，由于破坏事物至少可以带来创建事物一样的乐趣，人们开始在开发和测试角色之间转换，所有的人将学到更多关于如何得到良好代码的知识。

激情“吸毒者”继续存在

新的过程运行得如此良好，使得需求撰写者，开发人员以及测试人员不再具有生命力，这就使得那些在激情掌控的世界被提升的人惶惶不可终日，那样的世界意味着工作到深夜、最后一刻测试才参与，以及如同交战开火般的会议。而这些人对于那些还没有受新的运行过程控制的公司来说还具有吸引力。

Elvis Presley 是一个软件测试员

他的会议发放材料的标题就是：“软件质量：就是现在，否则永远不可能”

今天就为将来准备

不管我的预测是否成为现实，未来也会按照它自己的方式到来，下面就是如何准备面临未来的五个意见：

1．积极地不满于现状

不要接受测试的现状，四处看看，并且思考“我们在做些什么毫无意义的事情？”

2．抛开人与人之间的封闭

领悟如何更好的测试，并且分享这些知识。只有每一个人都试图使他所写的代码达到最佳状态时，整体质量才会改进。

3．学习更多关于测试的东西

如今，行业受软件测试的创新思维激发。用参加会议，加入邮件列表，网上冲浪，这些方式来解在测试前沿发生的一切。

4．学习更多关于开发的东西

参加一个编程学习班，即使你不打算编写大量的代码。将学习班当作是在 BUG 领土上的一次侦察飞行。

5．挑战世界

正如 PC 先驱 Alan Kay 所言：“预测未来的最好方式就是开创未来”

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=3729&fpage=1

战胜软件测试十大挑战：面向人的策略

作者：William E. Perry 和 Randall W. Rice
翻译：fyxu333

软件工程在范围和规模上都有了很大的增长，因此，实现单独的某个项目可以使用的方法很多。各种可能路径的组合很容易就达到成千上万甚至更多，这就导致了对它们的完全测试变得毫无可能。还有软件模块之间的动态交互，以及经过修改后的模块需要进行回归测试的问题。再有就是，软件开发人员经常不愿执行测试，而且，所有的研究都得出了可靠的结论，就是：开发人员不应该测试自己的成果。由此，需要发展独立的测试小组。

创建，激励，管理这些小组是个艰巨的任务。测试小组是介于开发者和客户之间的，在寻找错误的时候他们必须是残忍的，但又是不彻底的，因为产品不可能完美无缺。经理们应该感到满意，因为测试所花费的工作量是并不是白搭的的并且值得产品为之推迟。

十大挑战描述如下：

10）训练有素的测试

随便将某个人置于测试的位子，让他测试，这是种极大的浪费。测试是一种要求策略，能力和自豪感的职业。这些品质需要慢慢灌输，而且需要培训才能具备。

9）建立和开发人员的关系

测试人员和开发人员的关系素来就是紧张的，虽然大家的目标是一样的，但是却很容易陷入相互厌恶之中。

8）不用工具测试

如果可能，应该尽量避免不用工具测试，因为手工测试易于出错。大多数测试工具的花费可以从甚至仅仅一个严重 BUG 的发现中得到抵消。

7）向经理解释测试

测试不仅仅是软件周期的最后一步，还常被认为是为仆人保留的。从为测试所做的预算就常常可见一斑。由于在开发者手中被发现的错误只会花费在用户手中发现的错误的一小部分，所以这种看法很短见。

6）与顾客和用户交流

测试不仅仅是找出代码中的 bug，测试的一个重要的部分就是：确定已经创作出来的是不是应该创作的。这就意味着必须具备双向的交流通道。

5）为测试留下时间

这也许是所有挑战中最为困难的，在多数情况下，开始测试时开发过程已经进行了一年或者更多。毫无疑问，软件产品已经刊登过广告了，如果它具有吸引力的话，就会有用户盼着它完成，等着购买，在这种情况下，要为测试留下足够的时间变得更加困难。

4）测试“越墙”的产品

当开发人员完成开发以后，他们经常将产品“越墙”扔到测试者手中，测试人员对其进行测试，如果失败了，测试人员再将其“越墙”扔回到开发者手中（似乎开发人员和测试人员之间有一道不可逾越的墙）。如果他们改用面对面，职业的谈话来进行简单交接的话，情况会变得更好。

3）目标游移不定

这似乎是第二大困难的挑战，修改一个模块将影响到其他模块，试图将持续测试和修改 bug 交织进行的话，就是要将两个复杂的任务合并在一起。

2）在损失对损失的情况下抉择

穷举测试是不可能实现的，而且还有如此多的外力试图让软件测试者停止测试，放手的时间必将到来，所以确定测试到什么质量水平（可以结束测试）是一门真正的艺术。

1）必须说不

有的时候，测试人员不得不说，软件还没有达到发布的水平，这就需要有勇气，信念，奉献和意愿来坚持你的立场。

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=11379&fpage=1

常见的测试障碍及其解决办法

作者：未知
翻译：fyxu333

摘要

软件测试是软件生命周期的一部分。没有经过适当测试的应用软件会失去客户的信赖——不管是你的老客户或者新客户。因此，在软件发布之前，实施有效的软件测试，以排除常见的、可能导致浩劫性错误的问题，变得尤为重要。这篇文章总结了大部分测试工程所面临的常见问题，以及克服这些问题的一些方法。

不充分的测试计划及测试评估

有效的测试计划是整个测试工程中最为关键和富有挑战性的一步。测试计划及评估指出了测试任务实施的顺序以及方式，同样也指出了执行测试所需的资源。只有进行了合理和可靠的测试评估，才可能作出合适的测试计划。

- 工作量：在特定的时间内缺乏实施测试活动的资源，或者由于分配了过多资源而不能有效使用资源，都将会导致计划的推迟。
- 进度计划：进度评估将在在工作量评估以后，开发人员一般会低估测试所需要的工作量和资源，由此引发的结果就是：到最后期限不能将软件提交到到软件的最终用户，或者只能发布经过部分测试的软件产品。
- 成本：如果预算不正确，软件成本将变得更加昂贵；它还可能使得某些测试活动被取消，导致项目的更多质量上的不可靠。

如何解决

测试在总的工作量中的百分比，应该参照在以前的类似测试中的标准比例，并且考虑到一般管理费用，并由此估计个体活动的小时数量，然后再推断结果。一些由于缺乏知识丰富的人力导致的不合适的测试，比如使用测试人员时缺乏或没有经验，同样也会导致低劣的测试质量。别忘记考虑以下几点：

- 用以提高人力资源对于本领域或技术的知识水平所需的培训时间
- 用以解决提前预见的风险所需的缓冲时间

不明确的需求

没有足够的文档，在软件版本发布出去之前，测试人员通常要花费更长的时间以及留下更多的错误。需求的不明确使测试设计阶段变得冗长乏味。在这个阶段发现和修改需求的不足所付出的代价将大大低于在接受阶段将会导致代价。如果测试人员匆匆的阅读需求的话，可能会有许多暗示的或者潜在的需求被忽略掉。因此，在测试的开始阶段彻底的了解需求尤为重要。

如何解决

在书写测试用例之前，测试人员可以复查需求，准备一个对需求的疑问单，敦促其将需求明确化，以保证发布出高质量的产品 。另外，还可以准备一份关于需求的缺陷单。

测试覆盖的不足

一个好的测试包将达到较高的覆盖率。没有足够的测试用例，就不能完全的测试整个功能。测试覆盖只是测试质量的一个度量。如果没有达到高的测试覆盖率，那么测试过程必须要加强。另一个因素就是，没有完全考虑到可能范围的测试数据。

如何解决

与需求相关的测试用例号可以在 excel 表格中靠着需求的地方标注出来，这样，可以保证为每个需求设计了测试用例。低测试覆盖率表明了测试过程的问题，这些问题可能需要改进测试生成技术或者培训测试人员。市面上有不少度量测试覆盖率的工具出售。

要测试应用系统中的所有条件不太可能，但设计合法及非法的数据以达到完全覆盖正常的处理操作是可以准备的。在准备测试数据的时候，可以应用边界值分析，等价类划分这些技术。

测试环境失控

测试环境越接近最终产品环境，测试的可靠性就越高。缺乏这样的环境将导致最终产品中不可预料的结果

如何解决

测试应该在一个可控制的环境下进行,由此测试可以独立与开发或者产品环境。测试环境应该规测试小组所有，没有他们的允许，环境不能有任何改动。

为保证及时搭建环境以及很好的管理环境，可以采用一些衡量标准。测试环境

应该充分的代表将要执行的测试，它应该接近或者和产品环境相同。测试经理或者某个协调者管理与开发组的信息流通非常必要，并且负责环境搭建，版本控制，认可等等。如果建立了一个独立的测试小组，那么同样建立一个配置小组就会更加理想。

结果测试

对测试人力和资源需求的低估，将导致在项目开发周期的结束阶段时才开始辛苦的测试，这个时候要修补测试人员发现的错误变得很困难，而且由于时间的约束，在测试文档中将不会太注重细节。

如何解决

需求定义好以后就可以尽快的开始测试计划。可以让测试过程和软件开发过程同步进行。

测试文档的不充分

不充分/不合适的测试文档（包括测试计划，测试规格说明书，缺陷报告，等等）导致在分析应该测试什么/再测试什么以及该测试的相关领域中浪费很多时间，这些都会反过来影响产品的发布或质量。

如何解决

在文档上也应该花费足够的工作量，因为测试文档在整个测试阶段是一个非常重要的任务

从开始 SDLC 起，就应该留心准备所有与测试相关的文档，并且对文档持续更新。

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=4540&fpage=2

[carol2000 专栏]

针锋相对：一个实际的质量管理规划

作者：Robert J. Altizer 和 Donald E. Downin

翻译：carol2000

预备知识：

1. 质量的定义：质量是反映实体（产品、过程或活动等）满足明确和隐含需要的能力的特性总和。
2. 质量特性：为了便于生产企业内部从事质量管理工作，评价产品质量状况，以便最大程度满足用户的质量要求，而把其适用性要求定量的具体的表示。在质量形成全过程的各个环节，都应从保证使用质量的要求触发，提出定量的要求，以便明确质量责任，确保使用质量。
3. 数据/质量特性值：质量特性值通常表现为各种数值指标，即质量指标。一个具体产品需用多个指标来反映它的质量。测量或测定质量指标所得的数值，即质量特性值，一般称为数据，常分为计数值和计量值两类。
4. 软件质量：保证软件质量就是要求满足明确声明的功能和性能需求、明确文档化的（documented）开发过程以及专业人员开发的软件所具有的所有隐含特征。具体可理解为：
   - 软件需求是质量度量的基础，与需求不符就是质量不高
   - 指定的标准定义一组指导软件开发的标准，如果不能按照这些准则，就可能导致质量不高
   - 通常隐含需求是不被提及的（如软机易维护性）

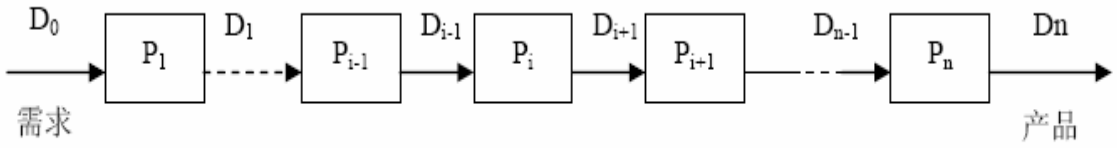
5．ANSI/IEE 规定的 6 个软件品质的特性要素：

- 正确性(Correctness)：功能达到设计规范并满足使用者需求
- 可靠性(Reliability)：于规定期间和条件下，仍能维持其性能水准
- 易实用性(Usability)：使用者学习、操作、准备输入、理解输出所做出的努力的程度
- 效率(Efficiency)：软件执行某项功能所需电脑资源（含时间）的有效程度

- 可维护性(Maintainability)：当环境改变或软件发生错误时，执行修改所做努力的程度
- 可移植性(Portability)：从一个电脑系统或环境移到另一电脑系统或环境的容易程度

6．质量管理是各级管理者的职责，但必须由最高管理者(OWNER)领导，但管理的实施则施涉及到组织中的所有成员

7．软件测试的对象不限于可运行的程序代码，而是包括质量管理中所涉及的在整个开发各阶段的输入、输出、过程和中间工作产品，测试活动也不限于对可运行代码的测试活动，而包括复查、走查、评审、验证等。事实上，软件整个开发过程，是一系列 “变换”或“处理和活动”过程 P1、P2、…、Pn 所组成的半序序列(参见图—1)。



图－1

图－1 中 D0 是源头（用户需求），D1、…、Dn-1 是诸中间“变换”过程的工作产品，Dn 为最终工作产品。显见，每个“变换”过程都有它的输入和输出，都可能导入（inject）这样或哪样的缺陷（Defects/Bugs），而且这种缺陷按缺陷扩大模型还会逐级扩大，尤其是在上游工作产品频繁出现更改时。不过这些中间工作产品大多只是以静态文档形态给出的。对它们的测试通常也只能采用 “静态测试”方法，其中包括工程测试（非正式的内部评审）、正式测试（正式评审）、审核测试（验证、审计）和检查性测试（检测、确认）等。

8．软件测试的经济效益主要来自以下两个方面。一是满足用户需求，提高产品的竞争力，最终提高产品的销售量。二是尽早发现缺陷，降低售后服务成本。而软件测试的最终目的就是使它带来的经济效益最大化。

9. 文档化（documented）：是指为了方便管理和维护，而将相关流程以及工具的使用方法等等信息以文档的形式保留下来。这样使得后续人员能够很快的适应相关岗位。

针锋相对：一个实际的质量管理规划

Robert Altizer and Don Downin

Motorola Semiconductor Products Sector Tempe, AZ

注意：由于排版原因，若有图片无法显示或出现大量空白页面，请选择视图模式中的 Web 版式浏览该文档

1．翻译原由：公司的资料多以个人的软件测试指导为主，这篇文章则多从团队角度考虑，因此一定的参考价值

2．初稿为原文直译。但由于中西方文化的差异，读起来很晦涩，因此做了修改

3．部分地方为避免歧义，保留了原文

4．本文的翻译约定：

Activities－动作行为或行为

Audits——核查，比 review 的程度要深

Conduct——实施

Mearure——测量，是对产品过程的某个属性的范围、数量、维度、容量或大小提供一个定量的指示

Metrics——度量，是对软件产品进行范围广泛的测度，它给出一个系统、构件或过程的某个给定属性的度的定量测量

Reviews——检查，或者复查

Performance——特性或性能

Program——规划

Process——流程

5．第一次翻译专业性的文章，不足之处请多多指教

摘要

组织常常在度量方面浪费相当多的时间和资源，但是他们作为管理工具经常是无效的，这是由于他们设计时通常没有用心考虑过商业目标的需求。为此，我们描述了一个完整的、以商业为导向的质量管

理计划。并且从产品指定的功能范围、责任人、功能需求目标和相应的改良计划、以及对每个测量质量属性的管理检查等方面把该计划扩充为一个“目标—问题－度量（GQM）”的模型。

关键字

数据驱动管理、商业成功因素、质量属性、产品功能目标、定制度量计划样板、质量检查流程、GQM

Robert J. Altizer 简介：

Altizer 为目前 Motorola 半导体部门的软件总工程师。他坚持管理并且在商业、政府以及软件工程的投产环境、分布系统的发展、局域网软件和安全以及质量确保等方面累积了较长的技术经验。Altizer 曾经担任过好几个工程过程小组（engineering process groups (EPGs) 的组长，负责引导产品生产、流程的发展、使用以及改良，为 Motorola 的商业宗旨（包括各方面的纪律和领域）提供支持。他在生产流程中增加了一个有效的环节，并且至今仍在使用。他是一名合格的 SEI SW-CMM 评估人，还一直在为 Motorola 社团的软件技术和质量审计而努力的服务。

注：CMM-Capability Maturity Model 能力成熟度模型，是评价一个软件企业项目管理及研发实力的国际标准，用于衡量和规范软件企业技术管理水平，是软件企业参与国际竞争的通行证和提高竞争力的最佳途径

Donald E. Downin 简介：

Donald E. Downin 是当前 Motorola 半导体组关于测试工程软件流程和质量的主管。他坚持管理并且制造系统的软件工程、系统信息、控制系统调查以及质量确保等方面累积了较长的技术经验。Downin 当前负责好几个工程流程组，包括运用一个全球化的测试构架来管理重点团队，为 SCG 工具和度量委员会服务，并且一直为 Motorola 社团的软件质量审计做服务。Downin 是一个合格的 SEI SW-CMM 评估人，并且在 Motorola 中每年执行好几个评估计划。

引言

组织常常沦陷于为了测量而测量的怪圈中而无法自拔，虽然它们一直老老实实地收集和报告度量标准，但是这些常常与组织的项目和

行为有很少的联系，甚至与商业和技术目标也没有任何关系。这种现象我们有时称之为“喂养社团的大猩猩”(盲目的行为)，这种度量计划对于任何人都没有现实意义。数据信息收集常常是很偶然而且不完整的，并且也伴随着项目生产负担的加重。度量计算和表格也许需要用一个很庞大的详细说明文档来定义，但是却只能捕获到关于项目的很少信息。而且为了完成这份庞大的文档，真正具有重要价值的报告常常被忽视了，就像没有人在管理中对度量计划责任人认同，或者对他的结论有兴趣一样。修改改度量与商业运作无关。为此，我们定义了一个质量管理规划，它描述了一个切实可行的方案，不仅继承和补充了传统的正统观点，还改进了其不足的地方。

我们意识到：一个有效的数据驱动管理规划应该合并了比度量定义以及一个社团委任更多的东西。作为我们持续的流程改良活动的一部分，我们也想把它发展成一个具有统计流程管理的能力的流程。我们想管理能从质量管理规划中发掘出需求信息，因为它对判断满足商业成功与用户满意目标之间的平衡关系有着关键性作用。我们希望项目团队成员能够理解他们工作中所产生的度量数据被用于干什么。并且我们也希望关于测量功能和对我们产品和流程的驱动改良有一个牢固的基础。我们的质量管理规划手册告诉了我们如何运用定量的和统计学的管理方法来确保我们的产品，服务和流程不但能满足我们客户的需求也能满足我们的商业目标。下面描述了我们的计划：

1. 确立以商业为导向的质量管理计划；质量目标和度量；预防和调整行动计划；bug 预防计划以及战略发展计划。
2. 质量属性的制定是以有效的商业价值以及对指定客户的承诺为基础的
3. 运用度量来为每一个被采用的质量属性去管理目标，部门，组织活动来使过程影响决定（make process change decisions）
4. 包括了基于样板的质量管理计划在项目管理文档中
5. 我们的质量管理规划覆盖了产品生命周期的每个阶段（in all phases of a product life cycle)，从开发，生产，推广到技术支持

宗旨和成功的标准

这个质量管理规划包含下列宗旨和成功的标准：

1. 提供研发队伍实现产品并且使其尽可能完美的能力，这一切基于商业需求。当处理能力的基准被所有队伍所建立，并且都在运用该质量管理规划以达到目标导向的时候才会满意
2. 在整个产品周期里（throughout the life cycle）改善队伍管理可交付的产品质量的能力。当所有的队伍经过适当的关于质量管理规划的训练，同时存在一个支持的结构框架用以捕获，报告，以及分析数据时才会满意
3. 一旦生产计划被研发组（Engineering Process Group--EPG）核准,就要使质量管理过程以及标准制度化.当评定质量管理规划被指示为正确的时候才会满意
4. 在研发过程中（into the development environment）系统地消除引入的缺陷（defect）。当缺陷削减计划恰到好处并且显示一直处于工作状态，而且在处理过程中错误（fault）和已发布版本中的缺陷（defect）趋于零

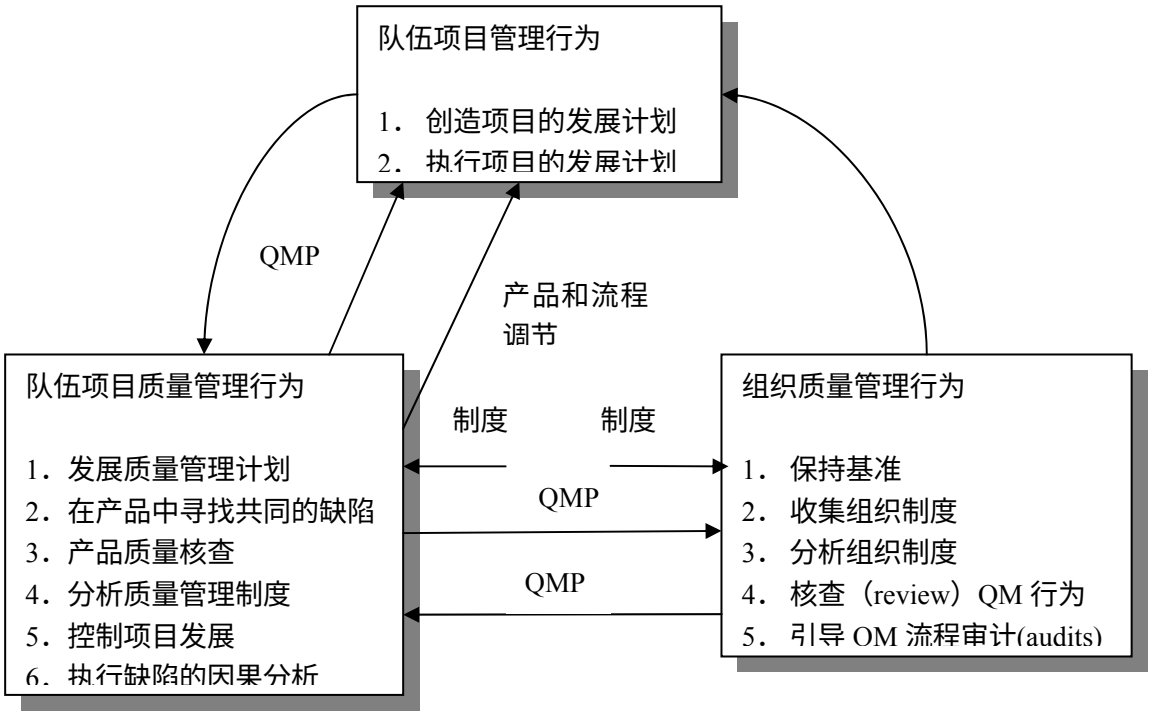
质量管理策略

作为质量管理规划发展中的一部分，我们需要适时更新我们组织的质量管理策略

1. 一般的管理者主要的责任是为了整个组织的质量工作以及为了整个组织质量管理规划的成功
2. 每个研发，支持以及服务队伍应该建立并且运用一个针对自己的质量管理计划
3. 周期性的复查应该坚持使用在每一个项目，队伍，部门以及组织层次上，包括质量管理计划内容，质量目标，以及缺陷预防计划和战略改良计划，为了完成工作目标，预防或者修正的行为也需要坚持
4. 组织的工程过程小组（EPG）拥有这本手册以及其内容，也有责任（responsibility）去坚持贯彻他。这个软件的质量管理者应该跟踪并且随时报告质量检查过程并且使其该过程更加有效

质量管理流程

一个完美的项目级质量管理流程由三个互动行为层组成，包括项目的管理层，项目的质量管理层以及组织的质量管理层。质量管理流程坚持的动作行为由三个功能实体组成：项目队伍管理，项目队伍质量管理以及组织质量管理。这些动作行为间的相互关系由下图来说明描述：



例一：简化地质量管理过程流

团队项目管理行为

这些应该由项目经理以及任务主管们来完成，并且有规律的项目管理行为的组成的重要补充是质量管理计划行为。

1. 创造项目发展计划，基于下面几点：一个切实可行的项目发展模板，他包括了质量管理计划（QMP）.这个活动提供了一个包括 QMP 的项目发展计划文档

2. 执行这个项目发展计划，基于下面几点 :项目发展计划文档（包括了 QMP）；该产品生产流程中的校正结果来源与：产品生产过程检查，在项目组织控制中分析相关数据，以及 QM 行为检查。这些行为提供了关于改进目标和动作行为功能的度量数

据。

团队质量管理行为

这些行为应该有团队成员或者其他被团队所认同的成员在质量管理计划中来完成。每一个计划中的质量属性都有一个特定的设计人来对此负责，并且此人应当有收集所有相关数据信息、报告、核查、计划以及计划执行权的责任和义务。

- 发展质量管理计划，基于：商业成功因素；商业目标；项目特定功能目标；项目功能的基准和控制限度；缺陷预防计划，基于已知的共同缺陷或者缺陷的引入原因，这些将在随后的课程中教大家如何鉴别。该行为提供了：记录为了达到特定功能的处理流程；记录目标和行为的改良计划。
- 寻找共同缺陷，基于：在数据库处理过程中的缺陷的历史记录（今后课程中学习）；在数据库处理过程中的缺陷引入原因（今后课程中学习）；相关的缺陷分类系统领域。这些行为提供了：为 QMPs 准备的缺陷预防计划。
- 产品生产质量核查，基于：生产数据和度量；质量管理计划；趋势以及峰值（指报告图表中的）。这些行为提供了：数据核查；生产校正；QMP 校正。
- 产品流程质量核查，基于：流程数据和度量；质量管理计划；趋势以及峰值（指报告图表中的）；流程记录。这些行为提供了：数据核查；流程校正；QMP 校正。
- 收集产品和流程度量，基于：数据收集需求由 QMP(已计划的度量)所定义；度量数据基于改良目标和行为的功能；核查数据从产品和流程检验和核查中收集。这些行为提供了：分析基础；报告质量状态的基础。
- 分析数据和控制项目，基于：收集度量和潜在的（underlying）数据;功能基准和控制限度。这些行为提供了：趋势和峰值特性的识别；预防或者纠正行为的基础；产品/流程校正基础；组织影响
- 执行缺陷原因分析,基于：系统数据库的缺陷跟踪；数据库检查；系统缺陷分类；问题解决技术（problem-solving

techniques)，比如：Ford 8D (处理流程 8 原则 Discipline)，TRIZ（发明问题解决理论 Theory of Inventive Problem Solving ）等等。这些行为提供了：导致问题的根本原因；对缺陷注入环节及原因的洞察力；缺陷预防计划的基础。

- 报告质量状态，基于：在 QMP(项目度量)中被定义的度量以及潜在的数据信息；趋势和峰值；相关的缺陷分类系统域；审计报告。这些行为提供：产品质量特性；流程质量特性；区域管理范围内的质量管理状态

质量组织行为

这些行为由一个指派的人来执行，他通常负责好几个关系到组织利益的项目或者部门；通常由 SQA 经理负主要的责任。

- 维持产品和流程能力基准和控制限度，基于：产品/流程基准和控制限度基准以及控制限度信息的改变。这些行为提供了：产品./流程的性能基准和控制限度的更新。
- 收集和分析组织度量,基于：度量和在 QMP（项目度量）中潜在的数据信息；审核报告；项目度量和数据分析对组织影响。这些行为提供了：对基准和控制幅度改变的建议；组织质量核查标准。
- 检查质量管理行为，基于：项目质量状态报告；审核报告；组织质量度量。这些行为提供了：预防或者校正项目管理行为的基准；战略行动的制度基准；缺陷预防计划的基准；总经理的决定指引着 QMP 的校正；总经理的决定指引着产品/流程的校正。
- 实施行为审计，基于：定义 QMPs;观察出定义 QMP 和组织 QM 流程的一致性；定义 QM 规划评定流程；各种质量流程步测。这些行为提供了：基于 QMP 的修正；该组织关于该质量管理规划（QMP－审计报告）质量的反馈。

质量管理计划

质量管理计划（Quality management plans –QMPs)被定义在项目/产品、部门或者组织级。其中的每一个计划的定位被确保在仅仅在所定义的范围内，并且将一系列的质量属性管理计划与商业成功的相关

因素包含在内。一个质量属性管理计划文档需要将定量的功能以及改良目标明确化，这样能为一个商业与成功息息相关的质量属性和每个目标的所有者一致，度量、数据、分析方法以及被用于支持达到改目标的行为也是如此。较高层次的质量管理流程被定义为由项目层来修改，改流程包括跨越多种项目的（部门级的）或者是跨越好几个部门的（组织级的）。部门和组织级的质量管理计划必须包括战略质量改良计划，但在更多已被限定范围的项目计划中，该部分常常会被忽略。

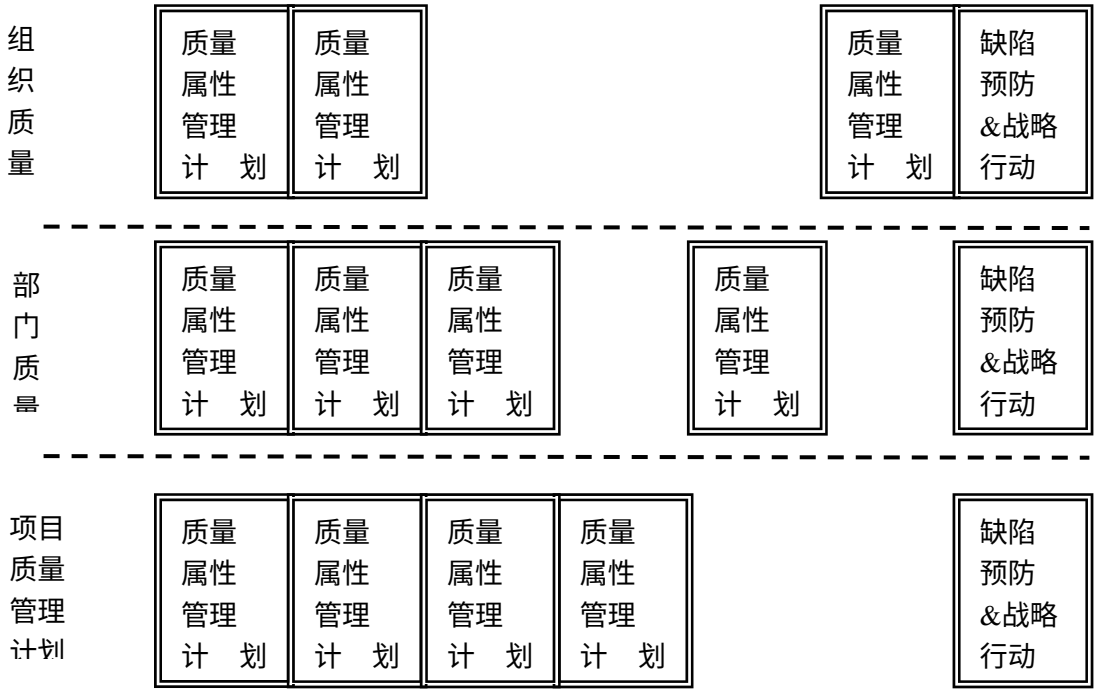


图 2：质量管理计划组织

我们的组织管理、EPG、部门管理以及项目团队也许定义了组织范围、部门范围和项目级别质量属性。图 2 显示了项目、部门和组织质量管理计划的结构。每一个计划只包括改级别内的与商业成功相关的质量属性；计划数量取决于该级别的管理者与组织质量管理者的协商而定。不是所有的质量属性都将被该组织的所有成员所测量或管理到，虽然一些（如缺陷率、次品开销等）能在三个级别里都能被报告到。

这里关于质量属性只有两条最根本的要求：

- 每一条属性都必须支持被标识的产品或者是商业目标流程
- 每一条都必须有一个被标识的主人，他必须该属性负责并且有权为了达到功能目标而将资源分配和引导行为的发展

如果任何建议性的质量属性与被标识好的商业目标间缺少一个清晰的链接环节或者不能将其交给一个指定人，我们将抛弃他。有一些质量属性，如产品标准和软件质量测量，也许会被你的组织之外所定义或者是他人收集、报告的，在外部管理中运用也许会被认为是强制性的。如果外部组织的“强制性”质量属性并没有满足包含在该规划内的判断标准，我们的 EPG 有责任通知他们。

所有的质量管理计划都要被核查，制度者、每一个质量属性的负责人、团队工程师、定义该计划范围的适当的管理层以及 SQA 经理所认可。

建立质量属性计划

该部分将描述一个质量属性管理计划的每一个元素。请参考附录中的每一个示例计划

商业案例（Business Case）

建立质量管理计划的第一步,我们需要建立大量的质量属性计划并且使基础的商业案例能够管理它们。所有的待检的和以前所遗留的质量属性（包括一些强制管理度量，如 CMM4 等）都将接受相同的判断标准：如果我们不能指定一个成功的商业 yinx 因素、商业目标、范围或者为找到对应的负责人，该条质量属性计划将不会被包含在内。然后我们将指定该质量属性并且指出它将集中与内在（如流程）还是外在（如产品或者产量）导向。这些属性也许是不确定的，但是一定于定义在该阶段的商业成功因素相容。例如质量属性包含缺陷级别、可用性、可靠性。

商业成功因素往往与质量属性相关联。它们聚焦在内或者外表，但是从客户的角度来看他们一定指向一个合乎需求的结果。最初所定义的商业成功因素也许是高度抽象的（highly qualitative）；它们随后将在质量属性管理计划中被精炼成具体的定性的测量标准。例如商业成功因素包括易用性、可靠性、可用性和小体积（small size）。

商业目标在支持每一个被定义的商业成功因素里定义了一个性能级别。他们也许聚焦在内部或者表面，但是一定指引到达所被认定的成功因素。例如商业成功因素，它们也许高度抽象但随后会被精炼成定性的标准。例如商业目标包括满足通过发展新产品或者增强原有产品功能来满足客户需求，维持或者改良以发布产品的质量（减少缺

陷）等。

该条质量属性的负责人是主管，例如项目经理、部门经理或者组织的总经理等。该负责人的责任是确保该属性对应的质量管理计划被充分定义和执行，同时它也对产品显著的功能特性有着个人兴趣。我们希望一个负责人能在管理商业时运用质量属性，并且在需要交付的度量报告中显示当前的产品功能。为了达到该目标，负责人通常需要确保产品的功能特性数据被收集，并且度量结果被计算和报告出来。产品功能将在指定的计划中被核查，包括功能目标是否达到；预防和校正行为是否被跟踪进行；该行为计划是否完成等。

我们评定是否达到指定目标的方法将被执行基于一些质量模型或者时被收集的观点，该方法主要为操纵问题（operational questions）所表现的特征。例如：操作问题为了支持达到“减少客户在已发布的产品中可能发现产品缺陷”目的也许会：

- 过去发布的产品中缺陷衡量的类型是什么
- 对每个产品类型来说每个已发布产品的缺陷标准是什么？
- 在过去发布的产品中中有多少个缺陷？已经发布了多少同类的产品？

度量定义

根据 GQM，我们基于上面所定义的操纵问题的方法，定义了指定的度量标准来跟踪产品的质量属性。所有的操作问题必须定位到至少一个度量上。这些度量定义需包含：名字、一个简要说明、一个关于所用到算法的细节的参考资料、对发生管理度量核查处的鉴定标准，还包括编译人、计算人和出版该度量者的名字。我们常用一个样表或其他的可视化的手段来显示管理核查。一个看似简单的图标也许显示了多个度量数据。

为了更完整的说明，我们常常描述所有的要素元素需求来计算被选择的度量。我们列出每一个需求要素的数据信息，与资源或者对应的收集机制、收集频率以及提供数据的责任人并列在一起。一个数据元素也许会被重复利用在多个度量中。

注：收集机制：即收集数据的工具或者数据库，或者是收集和报告数据的某种机制。

功能基准、目标和预防/校正行为触发器

我们为每一个指定的度量都定义了一个名义上的预期基准，并且该功能所达到的目标包含在质量属性计划范围之内。我们也指定了一个控制限度（如果知道的化），预防行为触发器就是提醒主人该行为的趋势将不在控制范围之内，校正行为触发器是用来启动立即行为的。

功能目标是对达到每一个指定度量功能的层次定量描述。我们的状态目标包括指定目标价值、趋势、控制紧缩度等。目标趋势包括指定测量和达到目标的时间、核查功能是否按预期执行等。项目级的度量目标应该坚持到项目结束；没有达到目标也许就会是退出发布产品的根本原因。部门和组织级的度量，所有的目标都要包含一个目标日期和产品必须达到的功能级别。

预防行为触发器是属于功能级，趋势，甚至会导致一个预防行为被计划和执行。完美的预防行为能够预防校正行为的出现。预防行为将被调用以保持产品功能在指定的范围和控制幅度之内，以使功能在变得不良以前进行调整。成功的完成一个预防行为计划需要预先阻止校正行为需求。预防行为计划也许会被包括在最初的质量属性管理计划或者之内，或者是被追加在处理操作环境需求里面。

校正行为触发器也属功能级，趋势，甚至会导致一个校正行为被计划和执行。校正行为对预防行为的失败或者是一些意料之外的特殊事件来说是必须的。校正行为将被调用来使以变得不良的产品功能回到一个指定的范围或是控制幅度之内。校正行为需求也许会跟随一些意料之外的或是特殊事件的发生而上升。校正行为计划也许会被包括在最初的质量属性管理计划或者之内，或者是被追加在处理操作环境需求里面。

管理核查计划

该阶段列举了正式管理核查将检查质量属性管理计划是否符合标准。质量管理计划在该环节所包含的文档化内容包括：会议类型、核查频率、功能目标鉴定标准和它们所对应的被核查到的度量、对于核查该流程的相关责任人清单、以前质量管理核查计划的成功案例等等。人物清单中最少需要包含每一个质量管理计划的责任人以及项目经理或者是其他的关键人物。

管理核查问题

在质量属性计划中我们建议当核查度量功能时以提问方式进行。管理核查问题应当能探测到度量报告下面的东西，能引起我们更深层次的思考，但这也许需要更多的细节度量或者数据才能表现出来。它们也更深入的向着目标探查，并且需要执行预防或者是校正行为计划。一些管理核查问题为了支持“将顾客在已发布的产品中遇见缺陷的概率降到最低”的例子也许如下：

- 在报告时期我们所发布的产品的价格是多少？同类的新产品价格是多少？
- 在已发布的和当前所支持的产品总价是多少？
- 在已发布产品中被报告的缺陷类型和严重程度是多少？我们有计划来控制它们吗？

度量参考

该阶段提供了在 4.3.3 环节所定义的度量的相关信息细节，包括所有数据定义、公式、算法等等，需要从所提供的数据中计算出度量价值。通常定义在其他地方的度量也许还要包括一些其他的参考资料；但是本地度量一定要根据上述方法定义。

缺陷预防计划

缺陷预防计划包括一系列的能够使缺陷在质量属性中引入率降低的行为；这些特殊行为将直接在缺陷发生前预防它们，将缺陷的发现和消除分离开来。因此，不是所有的质量属性都需要缺陷减缩计划。

战略改良计划

战略改良计划包括一系列行为，将被用来影响改良在质量属性管理计划的质量属性。这就是“如何”到达质量管理计划的功能目标。当指定的项目的功能目标无法被跟踪时，战略改良计划通常被核查来判定“执行困难（do differntlies）”，并且根据此来更新改良计划。对质量管理计划的责任人来说，核查并且赞同这些改变是非常重要的。

计划批准

所有的质量管理计划都将被制度者、每个质量属性的责任人、团队工程师、该计划所定义范围内的对应的管理层、以及 ATO SQA 经

理所检验。

质量核查

质量核查是我们质量管理规划中不可分割的一个环节。在质量核查中，组织功能、部门功能或者是项目功能都将根据它的项目度量来测量。质量核查流程也包括核查下列内容的进步程度：质量目标、预防和校正行为计划的内容、缺陷预防计划和战略改良计划等等。

质量核查流程

质量核查时我们组织每月核查的一部分，主要包括核查每个负责人责任是否到位。一个质量核查的最小团体应该包括一个经理、一个履行 OWNE 职责的人、团队成员和组织质量代表。

处于核查状态下的项目或者部门的动态度量表需要适当的备份数据和详细描述，将内容和动态的预防计划、校正计划、缺陷预防计划以及战略改良计划整合在一起。分配给核查 EPG 和 SQA 质量管理计划的时间并不是项目或者部门级职责的全部。总而言之，质量核查也许应该坚持在部门或者是项目级上。

为了效率，只有每个计划中最需要解决的问题（the top issues）才会被核查。只有新的预防或者是校正行为需求被触发时，该建议行为计划才会被核查。行为计划也许会在质量核查中被证明或者是被更详细的补充。

经理们需要对质量属性负责，其核查问题大纲需要被包含在相关的质量管理计划内。他们需要在质量属性管理计划中特别注意如何到达所定义好的质量目标所付出的每一个进步的过程。任何一个需要管理层抉择的问题在核查期间都必须被答复，或者是受到指定人及时的答复。组织的负责人对核查流程有最终裁决权。

软件质量确保经理在整个核查流程中扮演了一个记录员的角色，记录了从项目的早期质量核查到确保流程进行的全过程。

质量管理规划评定

我们的组织 SQA 经理应该从质量管理规划中意见的采纳和制度化等 4 方面来评定，包括：策略和过程、支持的环境、训练和计划执行方式；质量管理规划的实现和运用方式；季度性的度量报告以及随

后一定时期内的第一手用户反馈报告。

参考资料：略

相关表格：略

注：有一部分表格可参见 PPT

PPT

一．带着公制化的观点去管理经常会导致失败

组织收集并且报告了”标准”公制常与项目或者商业目标没有关系。数据常常是零星的或者说是不完善的，这种报告常常被管理层所忽视。因此 bug 数量也不足以“ 喂饱 社团的大猩猩 ”

注：粗暴一些方式俗称“大猩猩”测试法。除了不能拳打脚踢嘴咬外，什么招术都可以使出来。例如在测试客户机－服务器模式的软件时，把网络线拔掉，造成通信异常中断。

二．我们能正确的做吗？

1．我们拥有什么：

健全的测量文化

成熟的 3 层质量管理能力

2．我们想要什么：

以统计学来管理生产过程的能力

为了质量度量报告管理需求的能力

团队的每一个成员都能理解质量度量所起的作用

巩固测量工作的基础并且持续地推动产品生产流程中的工艺改进

三．摩托罗拉 SPS 质量管理计划

1．以商业需求为导向

商业导向质量管理计划以合适的范围以及合理的目标为基础

2．质量属性

质量属性以有意义的商业价值以及对所指定客户的承诺为基础

3．复查功能

复查工作反对质量目标（注：不知道这里是指回归测试是为了做出质量更好的产品）

4．运用度量来管理行为并且改变产品以及生产流程的决定

5.建立一个稳定而有效的发展支持环境(可持续性发展战略？!)

四．所有的产品以及分阶段的产品的生产周期

1．新的系统和生产发展

2．综合的购买零件

3．补充（update）或者加强（enhancement）现有的系统或者产品

4．制造产品

5．对已发布产品持续的技术支持工程

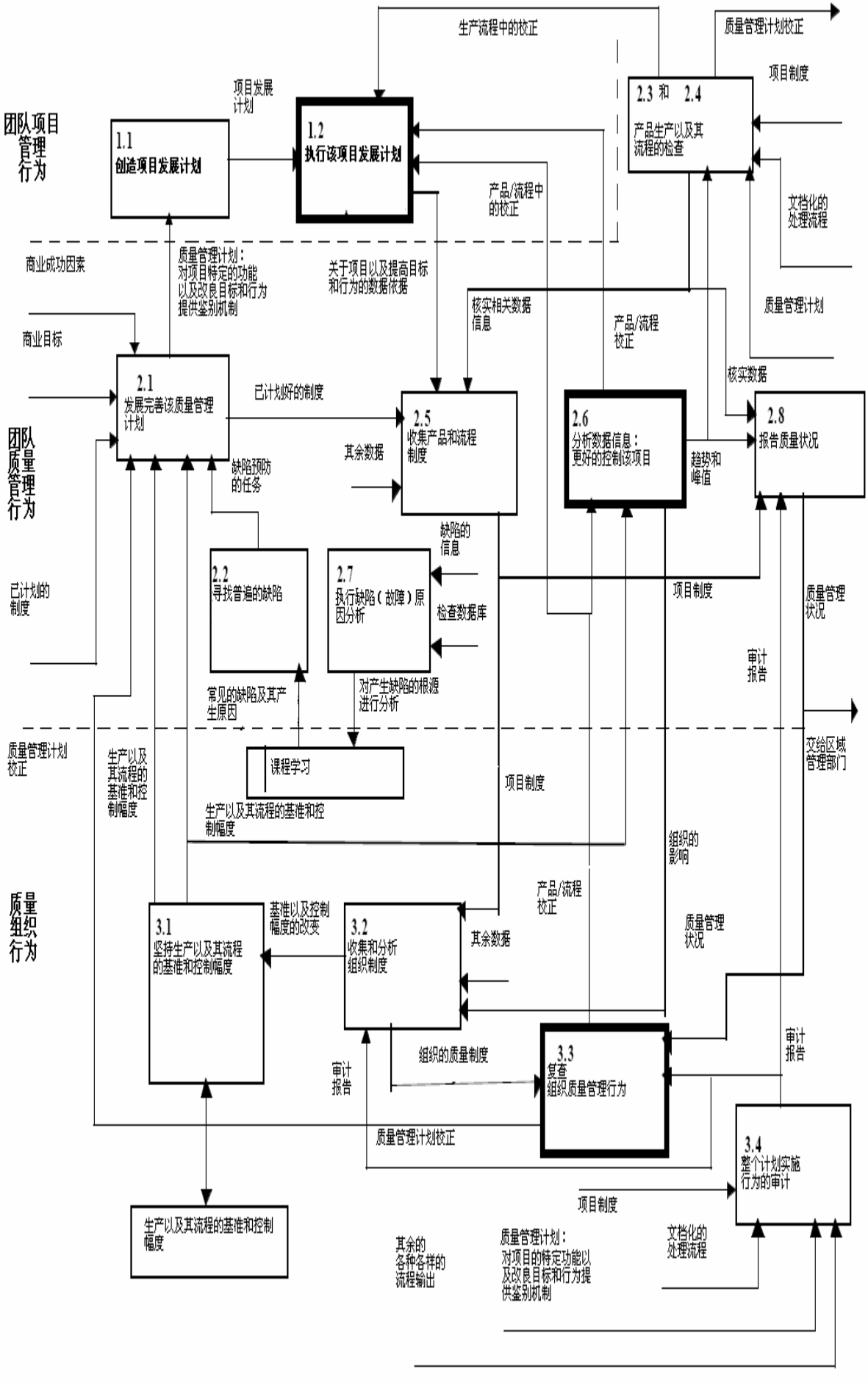
6．对基础设施（infrastructure）的支持

7．生产流程的改善行为

五．质量管理过程

见上图

六．质量管理的整个流程


团队项目管理行为
1.1
创造项目发展计划
项目发展计划
1.2
执行该项目发展计划
生产流程中的校正
质量管理计划校正
项目制度
2.3 和 2.4
产品生产以及其流程的检查
产品/流程中的校正
文档化的处理流程
商业成功因素
质量管理计划：对项目特定的功能以及改良目标和行为提供鉴别机制
关于项目以及提高目标和行为的数据依据
核实相关数据信息
产品/流程校正
质量管理计划
商业目标
2.1
发展完善该质量管理计划
已计划好的制度
2.5
收集产品和流程制度
2.6
分析数据信息：更好的控制该项目
核实数据
2.8
报告质量状况
团队质量管理行为
其余数据
缺陷预防的任务
趋势和峰值
缺陷的信息
2.2
寻找普遍的缺陷
2.7
执行缺陷（故障）原因分析
检查数据库
项目制度
质量管理状况
已计划的制度
审计报告
常见的缺陷及其产生原因
对产生缺陷的根源进行分析
质量管理计划校正
生产以及其流程的基准和控制幅度
课程学习
交给区域管理部门
项目制度
生产以及其流程的基准和控制幅度
组织的影响
产品/流程校正
质量组织行为
3.1
坚持生产以及其流程的基准和控制幅度
基准以及控制幅度的改变
3.2
收集和分析组织制度
其余数据
质量管理状况
组织的质量制度
3.3
复查组织质量管理行为
审计报告
审计报告
质量管理计划校正
3.4
整个计划实施行为的审计
项目制度
生产以及其流程的基准和控制幅度
其余的各种各样的流程输出
质量管理计划：对项目的特定功能以及改良目标和行为提供鉴别机制
文档化的处理流程

七．目标和成功标准

目标	何时满意
提供给研发队伍制定现实生产和其中过程质量目标的能力，这一切基于商品需求	1.过程能力标准有所有队伍共同完成 2.所有的队伍都要使用质量管理计划来度量目标
在整个生产流程中提高队伍产品质量可交付的能力	1.所有队伍都需要适当的关于质量管理流程的训练 2．一个机构框架存在于捕获，报告和分析度量数据信息
制度化质量管理过程，标准，并且该过程由EPG核准	评定QMP流程并且正确的执行该指示
在开发环境中系统的消去产生的缺陷	1．缺陷减少计划是适当的并且一直在起作用 2．在过程中的错误和即将法度的产品的错误增长趋势是趋于零的

八．质量管理策略

1．总经理对组织的质量负有最终责任（ultimate responsibility）

2．每一个发展、支持以及服务团队都要运用一个针对自己的质量管理计划

3．对每个计划、目标、功能、行为、缺陷预防都要保持周期性的质量复查，并且持续的优化他们（continuous improvement）

4．对质量核查进程进行有效的跟踪和报告

九．质量管理计划

1．定义在项目/生产、部门以及组织级别

2．包括一系列的对应与适当范围属性的质量管理计划

3．对质量管理属性的基本需求：

a.必须在支持同一产品（identified product）或者商业目标（达成共识）

b.必须有一个共同的、有着责任心以及权威的(responsibility and authority)主人来分配资源和直接引导整个团队来达到该目标（direct actions toward achieving the goall）

十．质量管理计划的组织框架

质量管理计划的组织框架

Mgmt——management

Attribute——属性

Strategic Action Plan——战略行动计划

十一．如何建立一个质量属性管理计划

建立若干个质量属性，这样的商业案例就能够管理他们了

成功的商业因素是与质量属性的好坏息息相关的

商业目标为每个已经定义好的商业成功因素（business success factors）定义了一个功能级别

该质量属性的所有者就是冠军、典型的项目领导者、部门经理或是组织的总经理

十二．质量属性鉴定表

质量属性栏	发布软件的缺陷：内在的查看已交付的缺陷

产品成功因素	1．采用六重（或者更多）质量检查机制在发布的产品中来增加用户验收成功率 2．减少花在重复工序上的时间将更多的时间用于发展产品更多的价值
产品目标	在已交付的软件使用上对用户的专业要求尽可能的小
范围	组织级
所有人	一般管理者

十三．运用可操作的问题来勾勒目标(GQM)

下面的问题就是关于如何支持目标的

“如何使客户在我们已发布的产品中发现缺陷的可能性降至最小？”

——对已发布产品的缺陷测量的目的是什么？

——每个产品的缺陷错误率的基准是多少？

——当前每一个产品的错误出现率是多少？

——缺陷类型的分布情况如何？

——这些缺陷是在哪个环节被引进的？

——花了多久来解决他们？

——还有多少没有解决？

十四．定义度量

- 从可操作地问题上来导出度量
- 定义度量的细节要素：

——规则系统和数据

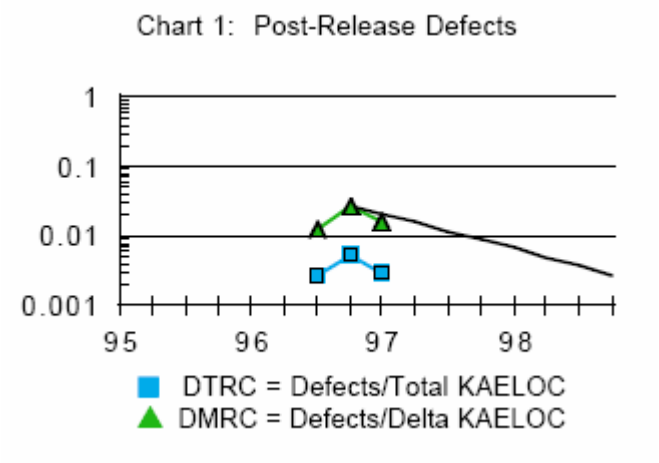
——在哪个环节进行检查

——数据信息的来源

——对编译、计算以及发布等每一个环节负责

——管理核查问题

——例表



十五．度量数据信息的定义

M # 	数据信息 要素 	数据信息 资源 	收集频率 	责任人
1 2 3	在总共和以修改的发布代码中的基准错误密度	组织生产流程的数据库	一次； 根据需要而更新	部门 EPG 报告; 主管 SW 经理； SQA 经理
1 2 3	基础大小、新的以及总共发布的软件为了每一个产品或者项目的测量	项目所构建的管理系统	每个季度	部门 EPG 报告； SQA 经理
1 2 3	每个小组成员关于发布软件的新的或者致命错误的报告	项目缺陷 跟踪系统	每个季度	部门 EPG 报告； SQA 经理

注：delta 源自古希腊符号△，软件度量中常用 △表示致命问题。

十六．每一个度量都包括。。

- 链接成为一个成功的商业指引
- 一个主人必须运用他来管理这个业务经营在所指定的范围内

——主人需要他来运作业务

——度量的制定者知道该度量将会被使用

十七．基准特性、目标以及行为触发器

为了每个度量，我们定义了：

——该基准（预期目的）的功能是为了每一个特定的度量

——在计划范围内该特定目标一定能达到（to be achieved）

——可控度（如果知道的化）

——预防（preventative）行为触发器

——校正（corrective）行为触发器

十八．基准、可控度、目标以及触发器

M#	基准&可控度	目标特性	预防行为触发器	控制行为触发器
1	DTRC=4 dpm (0.004Defects/KAELOC)	DTRC=1dpm by 1Q1998	DTRC比先前的报告更伟大	DTRC>8 dpm 为了2个连续的报告
2	DMRC=20 dpm (0.020Defects/KAELOC)	DMRC=8dpm by 1Q1998	DMRC比先前的报告更伟大	DMRC >30 dpm为了2个连续的报告
3	8X (60% 每年)	10X 每两年（68%每年）	DMRC 增长因子比目标比率低	DMRC 增长因子低于目标比率为了比2个连续的报告更多

十九．管理检查计划

1．质量管理计划包括一份关于产品功能将如何被检查的说明：

——检查会议的类型和频率

——被检查的度量和目标

——必需的出席人

2．管理检查问题包含探索每一个质量计划所报告的度量之下的东西，其目的是为了更好的理解

二十．Management Review Questions

管理检查问题

Q#	M#	对已发布的产品缺陷提问
1	1，2	有多少软件我们发布在申报阶段？他们有多少是新的？
2	1，2	总共有多少软件在发布阶段并且被当前支持？
3	1，2	有多少类并且有多严重的缺陷被报告在已发布的软件中？我们有什么计划去控制他们？
4	1，2	哪个软件存在最多的问题？为什么？
5	3	归根节底这是由什么引起的？我们如何解决他？
6	1，2	有多少缺陷被我们的用户所找到？
7	3	我们在上一季度解决了多少问题？每一个要花多久去解决？
8	3	有多少仍然没有被解决？为什么？

二十一．其他的质量计划组件

度量参考文献

——详细的文献，包括所有的数据定义、算法、规则等等

缺陷预防计划

——行为会导致低缺陷的引入率（这些缺陷不是与所有的质量属

性都有关联）

战略预防计划

——长期行为为了影响和改善某些质量属性（也许与“流程级”的范围没有关联）

二十二．质量检查

- 一个项目、部门或者组织的功能特性可从其质量度量来表现的，并且由其管理部门来检查
- 引导（Conducted）是我们每月组织操作检查的一个有规律的部分
- 角色定义：

——经理　　——属性的主人

——团队成员　——SQA 管理者

二十三．质量检查流程

- 一个必要团体（quorum）是必须的（包括管理者、主人、团队成员、质量代表）
- 现行项目：

——当前质量度量表格包括适当的备份数据信息以及上下文描述

——预防和校正行为计划的内容和状态（触发器）

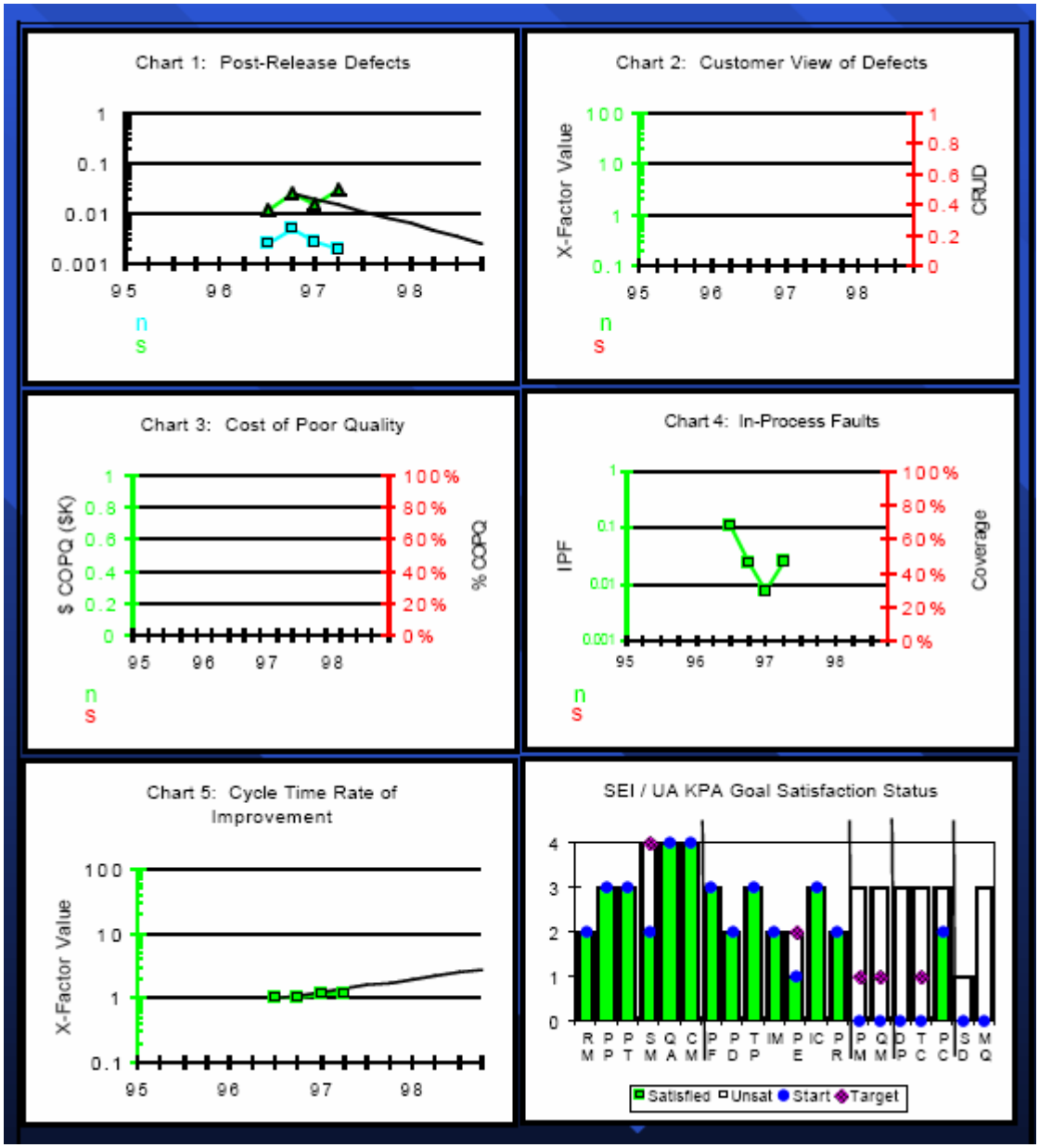
——缺陷预防，战略改良计划

- 管理者关于检查问题的责任

二十四．一个组织级度量的报告

注：由于缺乏进一步详细的资料，故这些报告的具体撰写过程未知。

估计是 95’-98’的产品战略报告。



注：一般问题常用■表示，致命问题用▲表示

二十五．质量管理评估

1．成功的要素：承诺，资源，训练

2．对项目的自我评估需要：

明确并且赞同该政策和过程

制定并且检查软件所支持的环境

训练该项目的参予者

执行：检查并且坚持按照计划发展和工作

二十六．总结

我们的质量管理规划：

——以商业成功为标准来定义质量管理部门

——保持行为始终处于一个可控范围内（a tractable scope）

——在我们的所有权以及经常的管理方面确保适当的质量检查

——运用一个简单的、可裁减的、可统计的基础质量计划模型

——在每个工程技术学科内为测量功能和持续的驱动产品、生产流程的改善提供一个牢固的基础

——为达到 SEI 四级标准提供一套切实可行的方法

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=6307&fpage=1

[fennek 专栏]

使用 Rational Robot 进行快速的自动化测试：案例研究

作者：Penny Witt （Odyssey 软件咨询公司测试顾问）
翻译：姚伟 测试部

名词解释

对本文中的一些关键的脚本概念做以下的翻译约定和解释。

- 快速脚本—利用Robot等脚本录制和生成工具，针对每一个单一的测试步骤形成对应的测试脚本，在本中文指步骤描述性脚本，特指最小粒度的测试脚本。
- 功能区域—被测试的应用程序中，按照功能特性划分的模块区域，本文中指由相同类型的活动组成的各功能块。
- 活动—被测试的应用程序中，用户为完成某项操作所引发的系统活动，本文中指系统中一般业务活动。
- SHELL脚本—由相同的活动类型下所对应得各个快速脚本组成的集合，本文中指将某项业务活动分解为若干小粒

度步骤之后，由这些步骤对应的快速脚本组成的二级脚本。

- 主脚本—由若干SHELL（二级）脚本组成的按功能区域划分的一级脚本。
- 通用脚本—在大多数的功能区域中都存在的一些操作，把它作为一般的通用脚本，由其他的一些脚本调用，本文中指一些经常在应用程序会重复出现的标题栏和对话框。
- 脚本命名—对录制的快速脚本、SHELL（二级）脚本和主脚本的名称进行统一的规则约定和管理。
- 清屏—在本文中指某个SHELL（二级）脚本中第一个操作的步骤，主要将此步骤作为此SHELL脚本的初始操作和结束操作。
- AUT—application under test，指处于测试状态的应用程序。
- trial and error—试错法，即通过试探和排除错误的方法而加入竞争的行列，以便去适应选择。也指从问题开始，经过尝试性解决，消除错误，然后又提出新问题，人们通过尝试和消错，通过猜想和反驳来取得知识的进步。
- hot fix—热修复。特指应用程序发布后，开发商为用户提供的应急补丁程序。

前言

使用Rational®Robot快速地生成自动化测试脚本，是一种非常有效的脚本开发方法，并且，在测试大型的AUT时，可以根据脚本的修改要求，对脚本进行全面的改进。

通常，实现自动化的功能测试需要几小时或几天的时间。并且，当两个或者三个项目的ID发生变更时，我们要修改每一个具有这些ID的自动化脚本。然而，利用这种快速的脚本生成方法，我们的团队可以在15或者20分钟之内开发出自动化测试。应用程序中的ID发生变化时，我们仅仅需要修改一个脚本，并把它集中到500到1000个测试中；不仅如此，它可以自己生成文档，且不需要专业的编程人员，也不需要额外的培训。

有一家总部设在太平洋西北海岸的公司，需要开发并实施此方法，于是他们从Odyssey软件咨询公司找到并雇佣了我。在两个月的时间里，我帮助他们的测试人员将100多个手工测试转为自动化测试，并装进一个回归序列（suite）中。到了第8个月，我们已经有了300个自动化测试，同时，这个公司又购买了5个以上的Rational Robot的使用许可（licenses），这样，全部的8台测试机都可以执行8到12个小时的功能测试了。那一年的年底，公司的管理层对自动化测试的优势有了深刻的印象，他们为总账系统的测试创建了一个环境，并指派了一个手工测试人员将总账测试转变为自动化测试（以我们现有的SHELL脚本为开始），寻找另一个全时的自动化测试人员。

现在，这个项目已接近尾声，完成了美国和加拿大的全部8个站点，并开始运行。同时，这意味着软件发布时一旦出现问题，就会造成巨大的影响。而我们的AUT非常的复杂且集成度很高，以至于测试组在每次的hot fix发布之前都要进行回归测试，不过，由于使用了快速的自动化测试方法，我们可以在计划的时间外来执行这些测试。

背景

我们要测试的AUT由一个企业范围的系统组成，它包含了产品目录、销售点信息到总账处理过程的各个环节。设计的使用容量为1500人，在美国和加拿大有超过8个以上的站点，使用的开发语言为C++，后台的数据存储使用SQL Server 2000。当我到达该公司时，开发阶段已进入4年计划的第三年；基本的应用程序已趋于稳定，但系统中还有大量的遗留特征功能等待开发。

测试由手工测试组成，包含了5个测试人员。虽然管理层希望这些测试员能够实现自动化的测试过程，但是他们都没有提出一个可行的方法。

测试员们使用了一种“纪录并回放”的自动化方法，但是他们发现维持这种自动化测试太耗时间了，并且对于他们而言都有着不小的难度。

另外，这个公司非常熟悉（从数据文件中提取信息的）自动化脚本的执行过程。在每一个新的功能被开发完成时，他们都会使用此方法对系统进行全时地数据装载。但这看上去并不能转移给功能测试，它和这些数据的关系不大，并且需要执行更多的贯穿整个AUT的计算、按钮和各种分支功能的测试。

公司也尝试过指定一个专门编程人员，让他来开发自动化测试，并全时地为测试提供帮助，但收效甚微；而这些开发人员也不了解被测试系统（AUT）的当前状况，实际上现在的系统已经可以为自动化测试提供足够的便利条件了。

因此，我们的问题是如何使测试人员获得相关的知识，对自动化测试进行维护。很明显，工具的学习曲线太长，以至于自动化测试人员不能独立地编制自动化测试脚本。同时，每个手工测试人员都不愿花上2个或3个小时去进行自动化的测试工作。而开发组则需要一套方法，希望可以快速自动地编码，同时在出现问题时，测试员和开发人员可以容易地理解。

测试分析

在复查了一些被测试系统（AUT）的标准处理过程之后，我们发现系统的使用步骤（点击一个按钮、输入一个字段、选择一个处理过程等等）是并存的，使用顺序是灵活多变的。这意味着每个单独的活动都可以形成一个单独的脚本。利用这种方法，我们可以把这些脚本集合在一起，来测试全部的系统功能。

这些快速生成的脚本是非常独立的，他们不依赖于前面或后面所产生的脚本。换句话说，在活动或活动正确地完成之前，他们不能对某一现有状态进行合法性确认。而这种合法性确认的变化，依赖于脚本的使用顺序。

对一般的自动化程序来说，消除合法性确认是一次彻底的改变。通常情况下，在验证测试中的关键功能区域时，产生的失败影响了全部的测试，那么我们认为测试本身一定出现了错误。而测试执行后的日志为我们展现了每一个验证的结果，以及整个测试的最终结果。在没有中间验证的情况下，要精确地确定出错误是很困难的，因为日志中只是简单地标注出了“ Fail ”的结果字样。

测试步骤不会显示在日志中，它们会被当作代码的一部分。这使脚本调试变得异常困难；虽然在日志中会显示通过或者失败的结果，但测试人员还是要阅读测试代码，以找出那些有问题的步骤。一个程序员执行了50个功能测试，但是要记住每个过程中的所有步骤几乎是不可能的。相比之下，这些快速生成的测试脚本，可以由一个主脚本来调用。而测试日志则会显示每个步骤地执行结果（通过/失败），这些脚本被称为步骤描述性脚本。

我们发现标题栏不仅可以通过功能区域本身，也可以在功能区域内部进行变动。这对Windows自动化来说是必不可少的，从而可以把它的关注点建立在窗口标题栏的基础上。一般说来，每个功能区域中的字符始终会留在标题栏上。由于全局变量和库函数是难于进行文档化的，同时也很难对其进行跟踪，所以我们希望尽可能少的使用他们。我们的目标是使用快速生成的脚本，它们描述了单个的步骤，可以使任何一个人都能清楚我们的测试内容。使用库函数和变量会影响我们易读性的目标。相反，这些快速生成的测试脚本很难使用一个一致的标题栏名称。这就要求我们应该根据功能区域来划分所有的测试脚本。

接下来，我们需要一个跟踪方法来确定每一个功能测试。而我们的销售点系统会在每笔交易中保留一个字段，以此来跟踪顾客或供应商的号码，以及AUT自带的计数系统。使用这种顾客/供应商的号码字段来达到跟踪的目的是非常合适的：它不会影响AUT中的处理过程，同时通过系统可以方便地为用户提供帮助。

最后，我们与手工测试人员一起共享测试环境，我们需要选择一些特定的顾客、供应商和产品号码，作为自动化测试所独有的属性，然后，根据功能测试来设置他们的属性和约束条件。

测试设计

我们在AUT中跟踪测试时所使用的名称，要能够标示出所做的功能测试，同时需要确定可能存在的测试周期。我们决定为功能测试的ID（测试以此为开始）创建一个跟踪名称。然后，分别以小时、分钟和秒来做为测试执行的时间单位。通过观察AUT的交易队列，我们可以根据功能测试来确认正确的测试，同时根据时间单位确认周期。下面是确认的测试和周期。

确认测试和周期

采购活动中的Supplier Reference字段，第一个功能测试的跟踪名称为：“ P001-12:22:04 ”。

我们采用首字母命名规则，对功能测试脚本进行命名，以区分其他的脚本。另外，我们决定根据功能区域来确定所有的脚本，脚本名称中的第一个字母需要能够标示出此功能区域。对于功能测试脚本来说，我们决定使用一个字母来表示此功能区域，并为功能描述尽可能

地留出一定的空间。在字母后面是功能测试的顺序编号。接下来是测试的简要描述，把它放在功能测试编号的后面。最后，我们在描述的结尾加入SHELL字样，以此对功能脚本和快速生成的脚本加以区分，并标示出他们所拥有的那些快速脚本。快速脚本可以用功能区域的开始名称命名，接下来对该脚本的活动进行简短的描述。

请看下面的示例：

Functional Test Names and Rapid Script Names

The first functional test in sales was

"S001 - Simple Sales Order SHELL"

The menu bar rapid script for File->Open in Sales was

"Sales - Open Transaction".

从上面的示例中，我们看到每个功能测试都是一个“SHELL 脚本”，并且由许多的单独的快速脚本组成。由于包含的快速脚本需要知道SHELL脚本的跟踪名称，因此，我们在SHELL脚本的开始处把它放入到一个全局变量中。为了确保系统能够判断出每个功能测试的开始位置，我们要在SHELL脚本的开头和结尾处进行清屏操作。

请看下面的脚本示例：

SHELL Scripts Holding Rapid Scripts

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''S001 - Simple Sales Order SHELL''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'''''''''GLOBAL VARIABLE''''''''''''''''''''''''
ShellScript = "S001 - " + Str(Time$)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''
CallScript "Sales - File New"
CallScript "Sales - Customer = Walker Lumber"
CallScript "Sales - Transaction Type = Sales Order"
CallScript "Sales - Customer PO"
CallScript "Sales - Tools-Goto Lines"
CallScript "Sales - 12031321020 Lumber Item"
CallScript "Sales - Release Shipment"
```

```
CallScript "Sales - Retrieve Order"
CallScript "Sales - Verify Order Released"
CallScript "Sales - File New"
```

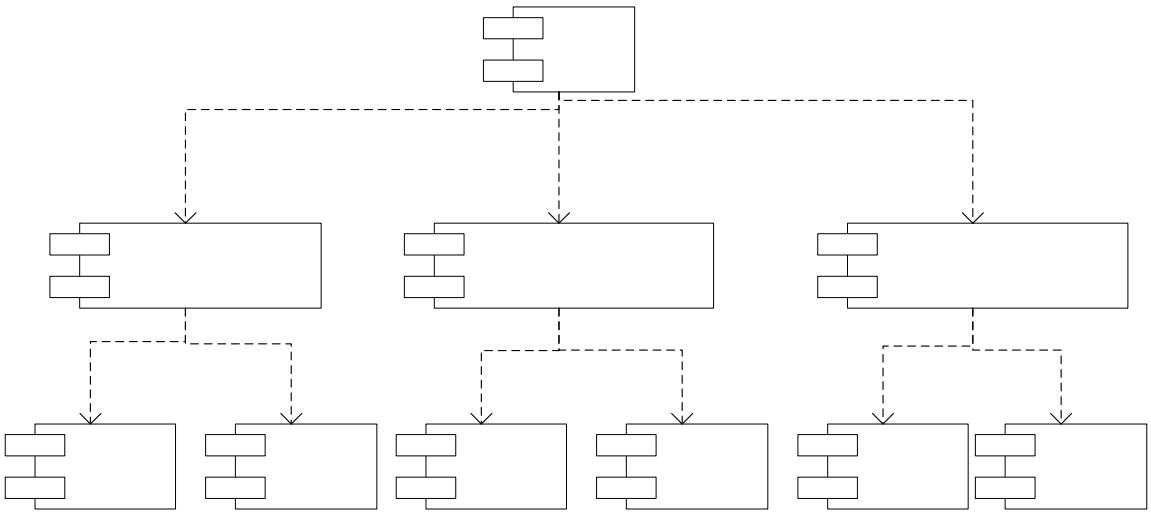
我们估计在每个功能区域中可能具有上百个测试，并且了解到我们需要同时运行多台测试机。并且，在某个功能区域越界进入到其他的功能区域中时，我们发现了一些潜在的不真实的纪录锁定。为了避免此问题，我们为每个功能区域分配了属于它自己的顾客和供应商集合。每台测试机使用一个主脚本运行一个单个的功能区域，如下例所示：

Master SHELL Script

```
''''''''''''''''''''''''''''''''''''''''''''
'''''MASTER SALES SHELL'''''
''''''''''''''''''''''''''''''''''''''''''''
CallScript "S001 - Simple Sales Order SHELL"
CallScript "S002 - Simple Sales Quote SHELL"
CallScript "S003 - Simple Sales Adjustment SHELL"
CallScript "S004 - Simple Credit Memo SHELL"
CallScript "S005 - Simple Material Movement SHELL"
```

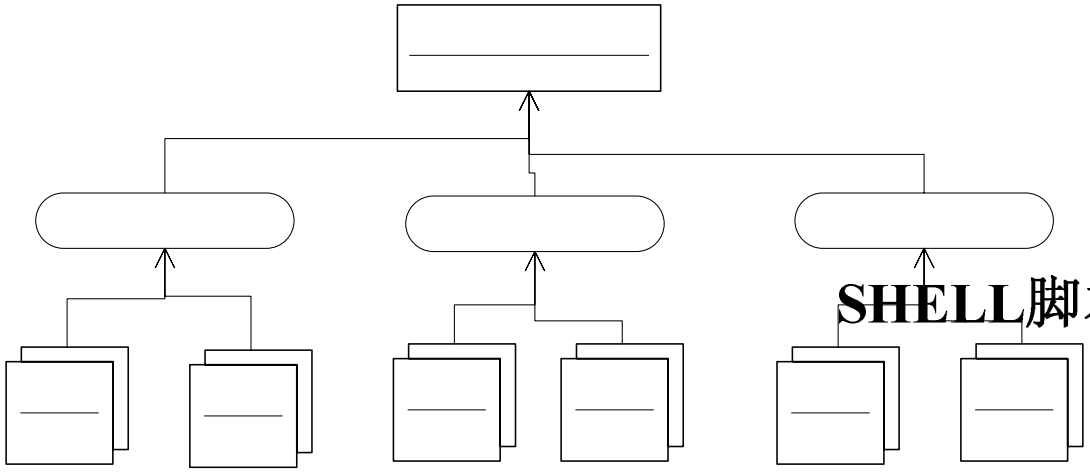
译者注：下图所示各级测试脚本的组成结构：

- 主脚本的级别最高，由若干SHELL（二级）脚本组成。
- SHELL脚本作为二级脚本，由若干低级别的快速脚本—小粒度按步骤描述的脚本组成。
- 最低级别的脚本为快速脚本，以各个活动中的基本步骤为粒度单位所对应的脚本。



下图所示各级脚本分类标准：

- 功能区域--相同类型的活动集合。
- 活动--同类型活动步骤的集合。
- 步骤--实现每种活动的各操作步骤。



SHELL脚本（二级）

下图所示设计的测试分布：

- 测试机1--功能区域1--主脚本1
- 测试机2--功能区域2--主脚本2

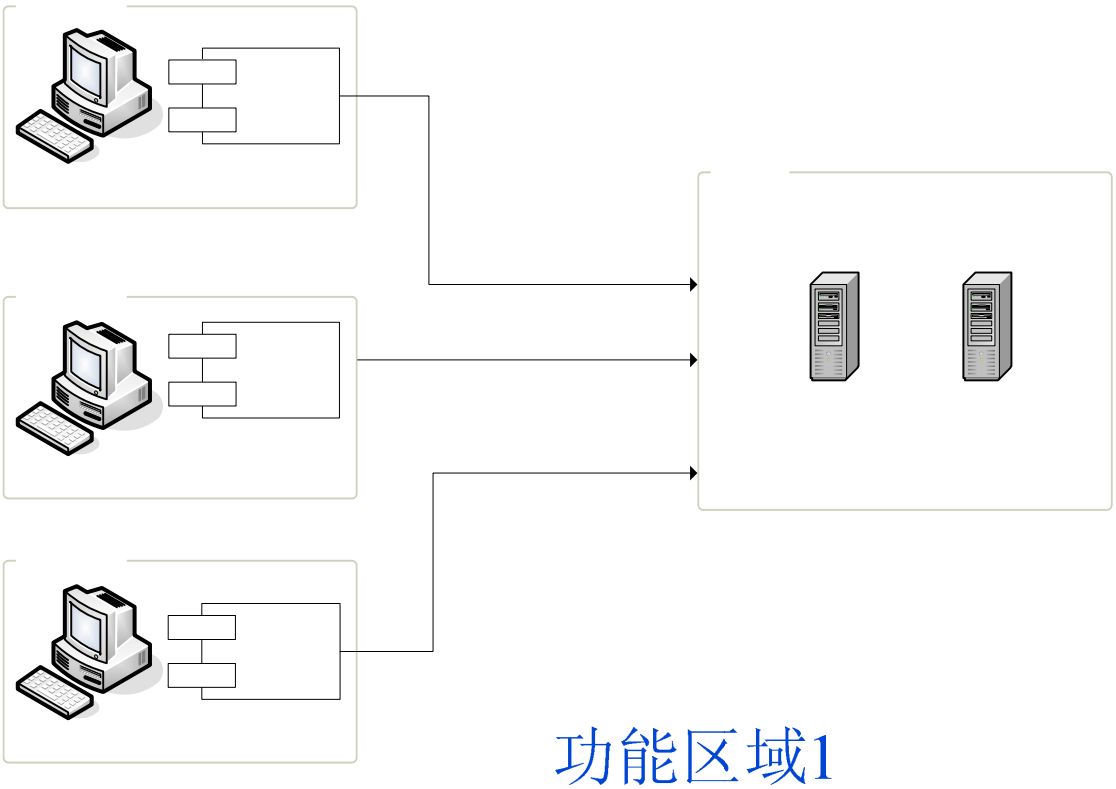
...

..

快速脚本1　　**快**

.

- 测试机n--功能区域n--主脚本n



测试开发

第一个月里，我们的测试产生了30个功能测试，其中每个功能测试包含了10到30个快速脚本。在第二月里，我们又添加了超过50个测试。在我们创建更多的功能测试时，我们创建的快速脚本会越来越少，因为已经有了很多这样的脚本。在第8个月里，我们已经有了300个功能测试，可以运行的快速脚本也有450个之多。

通常，我们一个小时可以创建3个功能测试。这项技术就是复制和新功能相似的SHELL脚本，然后根据新功能的变化，修改或添加其中的快速脚本。

构造周期（修改AUT中的小bug）对于由SHELL脚本组成的回归测试序列来说，效果似乎非常有限。但是新的特征会影响脚本的数量，而这些脚本往往会被破坏。同时，每个功能区域都具有新的对象ID，这就需要我们修改那些快速脚本。另外，这些快速脚本的好处也是显

主脚

测试机1

功能区域2

主脚

而易见的：只要我们修改一个快速脚本的ID，使用此脚本的每个SHELL脚本都会自动地进行相应的修改。因此，尽管我们的SHELL脚本会成倍的增加，但对我们来说，这些对象ID不会变多。

测试人员能够利用脚本取得成功。同时，程序员也可以参与进来，每周利用2个小时和测试人员一起来开发新的SHELL脚本。另外，这些脚本也覆盖了测试人员在每次新的构造之后所要进行的冗余测试。当手工的冗余测试越来越少时，测试员们会逐渐地给与自动化测试越来越多的支持。这些脚本是自解释的脚本—每个脚本对应一个测试步骤。这样，在执行回归测试的时候，程序员可以先分析出现的错误，来确定是否是脚本的问题（如，ID错误、新字段等等）。如果自动化脚本的问题没有引起失败的结果，那么测试人员有责任对AUT本身的问题进行分析。

我们的命名规则

我们需要控制好我们的命名规则，它可以在短时间内使我们的脚本变得清晰。在开始阶段，测试人员以活动的名称对脚本进行命名。但在一段时间之后，我们发现这些名称不是测试人员所使用的标准术语，他们会以不同的名称来约定相同的活动。另外，快速脚本的名称也没有和屏幕上的功能操作键关联在一起，这样使得其他程序员和测试员很难确定这些脚本的行为是什么。当测试员在脚本中标明“ Book the Transaction ”（定购交易）时，它就是此功能操作的最终结果，此功能的操作步骤—选择Tools->Record Receiving Results，然后选择Tools-> Done Receiving。当然，有一些活动的名称是正确的，但是这些名称没有说明你如何获得这个活动。例如，把一个活动指定为“ Releasing Transaction ”，它表示的功能操作—选择菜单栏中Shipment->Release，也可以点击工具栏中Release图标。有这样一个快速脚本，它表示的操作是打开搜索面板，此时，该脚本的名称可以用你想要查找的产品类型来命名，但是这个名称并没有说明是否应该点击按钮，进入菜单栏，或者是右键点击鼠标。因此，如果功能测试失败了，我们很难精确地定位产生失败结果的步骤。

在400多个脚本的列表中寻找一个正确的快速脚本可能会使人非常的困惑。有时这些脚本的名称非常口语化，且多为屏幕上的功能名称。所以，我们需要利用严格的规则来命名这些快速脚本，首先以功能区域为准，然后是区域中的活动，以及活动中所使用的数据名称。

利用这种方法，功能区域可以按照字母顺序排列，这样，把所有的push-button活动排列在一起，把全部的tab活动放在一起。我们输入字段时，把字段旁边的标签放在此功能区域的后面，标签的后面跟着一个“ = ”号，以及输入的字段值。最后，按功能区域列出菜单栏的选择集，使用菜单栏上的名称，加一个连字符“ - ”，跟着是下拉菜单中的名称。下面是一个命名规则的例子：

Naming Convention

Sales - Customer = Alen Afery

Sales - Customer = Wayne World

Sales - BUTTON Continue

Sales - BUTTON Go

Sales - File-New

Sales - File-Open

Sales - TAB Additional

Sales - TAB Delivery

译者注，对应上面所描述的例子：

快速脚本的命名规则

菜单栏名称 – 下拉菜单栏名称 – 操作名称

菜单栏名称 – 操作的字段名称 = 字段值

菜单栏名称 – 操作类型 操作名称

建立一致性规则

随着测试的进行，我们会发现一个明显的问题，在整个项目组人员的范围内，我们实施快速方法的过程缺少一致性。首先，我们把活动放进一个脚本中一起执行。例如，脚本“ Purchasing - Tools-Record Received ”是一个菜单选项，同时也包含了下一个菜单选项“ Shipment->Done Received ”，因为它们往往需要结合在一起执行。但是作为功能测试，这样做会增加复杂性，我们发现在“ Tools->Record Received ”和“ Shipment->Done Received ”选项之间可以进行很多其他的操作。于是，我们不得不回过头来把原来的快速脚本分割成两个独立的脚本。另外，功能键（热键）的使用也没有预期中的多。在我

们重写功能区域的代码时，对于热键的使用来说，要避免修改对象ID，所以我们开始大量的使用热键以代替对象的点击操作。这种转变，并不会增加或减少我们用自动化测试发现的AUT失败结果的数量。

此外，在新的发布之后，我们使用的是已被修改的对象ID，同时，我们发现这些对象可以与那些未做修改的对象隔离开，放入到一个单独的脚本中。例如，双击一个对象，在屏幕上展现此对象的细节，来完成各种各样的功能。其间，由于可双击的对象，其ID会经常发生改变，而对应的功能细节很少需要进行修改，所以，双击操作可以放入到一个单独的快速脚本中。

已被破坏的快速脚本，它们往往由更小的脚本组成，这对我们来说是一个很重的负担。而且，任何人都能猜到，那些SHELL脚本包含了这样的快速脚本。同时，我们是通过每个SHELL来开发测试程序的，所以应该检查原有的快速脚本，并把它放入到一个新的脚本中去。某些情况下，我们需要做多次修改，这样很难用编写代码的方式来实现，因此应该复查所有的SHELL脚本。

功能验证是另外一个被修改的区域。通常在快速脚本列表和一个验证了此功能测试结果的快速脚本的地方，结束SHELL脚本的执行。修改主要的对象ID，并进行了一次发布之后，我们注意到大多数验证功能的脚本只被一个SHELL脚本所使用。由于我们可以在其他的脚本中反复地重用这些快速脚本，所以，验证功能的脚本并不符合实际的标准。实际上，它们很难获得正确的对象ID：使用者需要另外一个脚本来验证所有的SHELL脚本。因此，我们把SHELL脚本中的功能验证放在SHELL脚本的结尾处，并删除这样的一个快速脚本。对于一个外行来说，这意味着阅读此SHELL脚本是件既困难又吃力的事情。

当程序员越来越熟悉应用程序的时候，他会发现AUT的各种功能区域里总是要突然出现很多一模一样的对话框。这些对话框有他们自己的标题栏，无论是从哪个功能区域访问这些对话框，它们都不会变化。由于我们是根据功能区域来区分和命名所有的快速脚本的，所以，这些对话框会被反复的编写到脚本中去--每个功能区域至少会有这样的一个脚本存在。因此，我们在开发新的功能区域时，可以把它当作一个通用脚本，并命名为“U”。这样的话，此功能区域中的脚本名称可以用U打头，后面是标题栏和对话框的名称。如此，我们可以减少冗余脚本的数量。

多种环境和发布

我们每周所进行的回归测试证明了它的价值，于是，管理层希望加大自动化的力度，测试AUT发布间隙所产生的那些hot fix。这对于前面所描述的—命名规则和建立一致性来说是一个很大的且需要克服的障碍。

首先，我们需要为hot fix的测试环境建立一个单独的数据库，以此来映射我们所使用的那些字段。但在每周测试后的发布期间，数据库会有一些变化，而变化后所使用的字段并没有包含在我们的测试数据库中。

其次，新的构造被放入每日的发布当中。而这些新代码会使对象ID发生变化，而这些变化并没有包含在hot fix代码中。因此，我们需要修改快速脚本中的一些对象ID，同时，这依赖于它们是否会运行在未来的或那些重要的功能代码中。

在此之前，我们只需要定义一个全局变量：在SHELL脚本中使用一个名称来跟踪每个测试。现在，我们则需要更多的全局变量，在发布期间指定脚本所要测试的内容。

对自动化进行扩展

我们的自动化测试只需要一个程序员就可以了，他和测试人员一起利用一周的前三个工作日来开发新的测试，在随后的两个工作日里进行回归测试。这种思想首先是通过自动化的回归测试检查新的构造，查看新的代码是否破坏了AUT中原有的代码。然后，测试人员开始他们的工作，确认基础代码并未发生变更；或者，最糟的情况是他们知道哪些地方不需要测试，因为有些测试不得不重新进行。

我们的回归测试由300多个SHELL脚本组成，并运行在8台测试机上。在两天的回归测试中，我们使用了一种非常一致的格式。首次执行期间，我们让全部的8台测试机持续运行8到12个小时。在此过程中，会出现失败的测试，尤其是那些包含了最多的功能的测试。程序员很快就能找出失败的脚本，并在下一个脚本中重新开始执行Master SHELL。当全部的脚本执行完成时，把失败的脚本当作一个问题放入失败的脚本列表中，并重新执行这些脚本以确定原因：是自动化脚本的错误还是AUT本身的错误？我们开发了一种数据表格，把所有的自动化SHELL脚本都罗列出来，并将此次回归测试的结果标注为

Pass/Fail。如果SHELL脚本的结果为“ Fail ”，把它指定为一个AUT错误，并交给测试人员查找此问题。某些情况下，查找的过程意味着要观察SHELL脚本的运行情况，或者是手工地访问此SHELL中的快速脚本列表。

同时，我们引入了一个次要环境，让另一个人来维护交替（alternating）的代码，但是要通过大量的“ 试错法 ”来确定此人应具备的技能。这种形式的程序通常是很极端的，有编程经验的人大多都不适应这种程序。不是因为它有多么的复杂，事实上，刚好相反。

大多数的程序员都会使用编码器，如代码库、全局变量、数组、循环、go to以及函数语句来创建应用程序。但是，我们的自动化代码与此类型的程序代码不同，它要明确地描述一个完整的活动，以完成所需的测试，同时，非程序员也不需要立即理解这些代码，而且，我们要用大量的文档来维护它们，这些描述都有一个简明扼要的脚本名称。显然，程序员并不适合这项工作。

另一种方法是由一个测试人员利用他或她的测试经验和我们一起工作。我们最初的要求是执行回归测试并分析它们。分析中的重要部分是理解我们执行的功能测试的内容。每次有失败的脚本时，我们必须确定问题的出处：究竟是AUT问题还是脚本本身并不适合此次测试？未来的发布很可能会有hot fix所没有的新特征。当自动化代码寻找一个不同的对象ID或者数据时，我们需要一个IF语句来确定脚本代码所执行的路径，是否应该执行未来的代码/数据或hot fix的代码/数据。

还需要更多的工作

我们在过去一年多的时间里，持续地使用了这种快速的自动化测试方法；目前，我们有超过500个SHELL脚本，以及1200个快速脚本。我们的测试覆盖了应用程序中绝大多数的区域，但是我们必须执行更多的测试，以完成我们所有的测试任务。

随着测试数量的增长，回归测试的时间在相当大的程度上并没有增加。越来越多的测试块开始趋于稳定，新的变更也越来越少。同时，经过对AUT不断的改进，使顾客订购单（Customer PO）或供应商参考号（Supplier Reference Number）的访问在整个系统中变得越来越普遍。这与我们在SHELL脚本中用以确定测试的字段是相同的，所以

我们的脚本可以更为有效地确认访问的顺序。对AUT的改进，消除了很多以前的测试设置，以及测试名称的搜索。

如果全部的团队成员都能严格地遵从系统的各项标准，那么编制脚本的工作会变得更为顺利。和大多数临近结束的项目一样，团队成员们会认为这些标准是多余的，往往会把它抛之脑后。一旦管理层开始意识到自动化测试的益处，我们便要更加严格地实施这些标准，因为它们决定了脚本的用途和质量。

一段时间过后，测试人员越来越信赖他们初期所进行的自动化测试。相反，他们看上去好像没有多少时间去开发新的自动化脚本。作为一个折中的办法，在测试人员完成测试后，我们需要手工的测试文档，记录每个新的bug修复。然后，我们可以使用这个新的测试文档来创建新的SHELL脚本，或者，把它添加到现有的SHELL脚本中去。

此外，在创建了一个封闭的环境，对总帐帐目的余额进行了严格地自动化测试之后，管理层开始大力地推行自动化测试。这样，我们需要确保数据的完整性，并允许我们在每个新的测试周期开始时清除原有的结果数据。进行总帐测试的手工测试员构造帐目模块的SHELL脚本，以此创建帐目的回归测试。

在这个AUT临近完成时，我们的焦点逐渐集中到了系统的性能和自动化脚本的使用上。设计人员现在可以定位系统的性能问题，自动化测试人员可以采用类似的SHELL脚本，进行性能测试，从而使系统设计员可以收集并分析性能的统计数据。如上文所提到的，我们在全部的8个站点上完成了AUT的安装，同时，进行最后阶段的修复和展示。对于每一个包含在此项目中的人来说，现在的项目风险比以往任何时候都高：对于客户管理而言，具有缺陷的发布会引起大范围的问题；就公司的雇员而言，这些问题使客户对公司的最终印象造成负面的影响；而测试组非常关心系统的变动，这是因为我们的AUT的复杂度和集成度都很高，以至于每一个小小改动都会影响整个系统的运行状况。

要减少这些风险，测试组需要在常规发布和hot fix发布之前，优先进行回归测试。他们知道手工的回归测试不可能覆盖所有的区域，在时间计划内依然会有未测试的遗留。这样的话，及时地发布系统的hot fix就变得更加的关键，因为完全的回归测试是不可能做到的。当

构建好一个hot fix以待发布时，由测试组确定此修复所造成的影响，然后选择执行相应的自动化SHELL脚本。

自从我们雇佣了另外一个全时的自动化测试人员以来，我们能够管理正常的回归测试并编制新的 SHELL 脚本。经过了连续的三周时间，我们已经能够进行回归测试和热修复，同时，又编制了 80 个新的 SHELL 脚本。

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=13646&fpage=1

制定项目的测试策略

作者：Michael Kelly
翻译：fennek

名词

烟雾测试：一组用以确定系统处于稳定状态、所有的主要功能都具备并且能够在“正常”条件下运行的测试用例。

烟雾测试的目标是显示稳定性、而不是发现系统的每个 bug。

你需要一个测试策略。为什么？

最近，我需要为自己工作的项目制定一份完整的测试策略。在我刚来到这个项目组时，我发现开发人员试图使用一个不完整的瀑布生命周期模型。这就意味着开发过程要具备专门的团队基础。刚好，这个项目组有大约 12 个开发人员，正处在利用并行开发工作来研究更多迭代方法的过程中。有一个测试新手，工作非常努力，为项目提供仅有的测试。而我为了帮助他们测试，也加入了这个项目。与此同时，项目组雇用了新的项目经理和架构师，非常主动地承诺会在剩余的一年时间内结束项目。这种沉闷的情景是不是听起来很熟悉？我猜它会是这样，因为这不是我第一次碰到这种情况。

这里，首要的问题是制定一份测试策略。什么是测试策略？这要看是谁在问你。这篇文章里，我们会把测试策略作为所有测试阶段、测试技术和项目所使用的测试工具的目标。最重要的一点，测试策略应该使测试过程中的交流变得更为容易，而它会影响到整个项目组。

通过制定测试策略来指导我们的工作，下面是项目组所碰到过的一些具体问题，我们需要寻找一些解决他们的方法：

- 缺乏可重复性测试—-项目缺少回归测试。
- 缺乏可见性测试—-没有收集衡量结果的指标，唯一的标准就是发布代码的期限。
- 反作用的构建过程—-他们只对项目的紧急程度构建响应，没有预测其他构建人的需要。
- 没有对测试环境或测试数据进行控制。
- 代码发布后，没有进行单元或集成测试。
- 没有简单的自动化过程，没有测试过程。

下面的故事会告诉我们如何定义并实施一个测试策略。

让我们开始吧

在制定测试策略时，你需要和项目中的关键人物一起，将关注点放在你们所面对的问题上，制定一个长期的解决方案，可以在整个项目周期内实施。除了上面列出的那些问题之外，我们的项目解决方案还要满足测试策略的基本需求：*在开发周期内，帮助项目组尽可能早的找到最严重的 bugs*。想尽早地发现最严重的缺陷，需要把项目的测试部分和开发部分联合在一起，包括不同的测试阶段、测试类型、项目环境，以及如何在环境、角色、职责之间升级代码，还有普遍使用的工具。

这个看起来是不是有点复杂？实际上，它比你想象的简单。

保持简单：写字板上的计划

测试策略应该尽量的简单，这样我们可以将其展现在白色的写字板上，同时，一个简短的会议，你应该可以把它的含义解释给项目组内的任何一个人。在你花时间把测试策略的细节写入文档之前，清晰地定义简化的概念是简单性的保证。把想法和内容写到写字板上的过程是一件非常有参与性的事情，它可以帮助人们共享意见和观点，这时候，写字板往往是最好的媒介。人们使用写字板时，会画一些漂亮的图和流程图，而这些图形符号每个人都能理解。

制定测试策略时，你需要把项目组的其他人包含进来。一般有，项目经理、开发主管、架构师、DBA（数据库管理员），以及其他一些关键人物，他们具有

一些可利用的技术资源，所以他们可能具有更好的想法。此外，你的测试策略应该覆盖整个项目的生命周期，让每个人都能按照它的方式工作。这意味着你需要这些技术人员投入其中，以保证它成功地实施。至少，他们可以给你更多有关测试类型的现实想法（单元测试、代码复查、执行期分析等等）。我通常试着寻找那些最大程度地包含在项目中的人，和他们一起开会讨论。因为，他们的洞察力和建议往往是非常宝贵的。

第 1 步：基本策略轮廓

拿出一个空白的写字板，和大家一起开始。首先，把每个你想要捕获、制定的测试方面写到写字板上，分成列的形式。对于我们的项目，我把测试阶段（包含了项目所执行的测试类型）、不同的代码环境和衡量指标各分成了一列，其中的衡量指标决定我们何时在不同的环境之间移动代码。图一展现了它的基本面貌。

图一 写字板—-轮廓

我使用红色标注出这些定义。列出项目组当前并存执行的每个测试阶段；在测试阶段的下面，列出要执行的测试类型。而寻找测试类型的过程，会帮助你清晰地定义测试阶段，同时，你可以把每个阶段所代表的含义和产生的内容分成一组。这里没有所谓的“ 正确 ”定义；唯一重要的一点，就是大家都认可你所使用的定义。你也可能需要定义测试的类型，但更重要的是确保每个人都能够理解如何区分不同的测试类型。记住，你要使大家能够自由地讨论测试策略，一个清晰的框架需要清晰的定义。

第 2 步：列出当前构建条件

接下来，列出不同的项目环境，以及当前所使用的衡量指标，后者决定了何时在环境之间移动构建。在我们的项目中，我询问了每一个参加会议的人，发现项目组会执行系统测试和一些回归测试，以及为少量用户提供的专门的接受测试。而这里缺少单元测试和集成测试的一致性，以及通常在代码提交给用户之后所要进行的代码复查工作。系统测试中，我们利用一些功能测试和生命周期测试，将大部分的需求做了验证。而在前面的迭代中，项目组执行了一个或两个临时的潜在测试，所以我们已经包含了那些测试。所有的回归测试，在时间允许的情况下，从前面的一次发布开始，是手工地基于测试用例的测试。

我们具有 5 个项目环境。每个开发人员有着他们自己的本地环境，接着，他们要将其全部集成到一个普遍的开发环境中。一旦被集成，项目要构建一个测试环境，以进行系统测试的工作。然后，基于需求验证和发布日期（关键的衡量指标），把代码移动到质量保证（QA）的环境中。用户根据发布的版本对大多数的期望功能进行复查，然后在一系列的结束标志（另一个关键的衡量指标）出现后，代码即可移动到产品中。图 2 展现了我们的测试策略，它包含了上面所说的全部内容。在衡量指标的那一列中，我们对每个环境中的需求验证级别进行了讨论，确定了那些能准确地反映当前过程的数字。

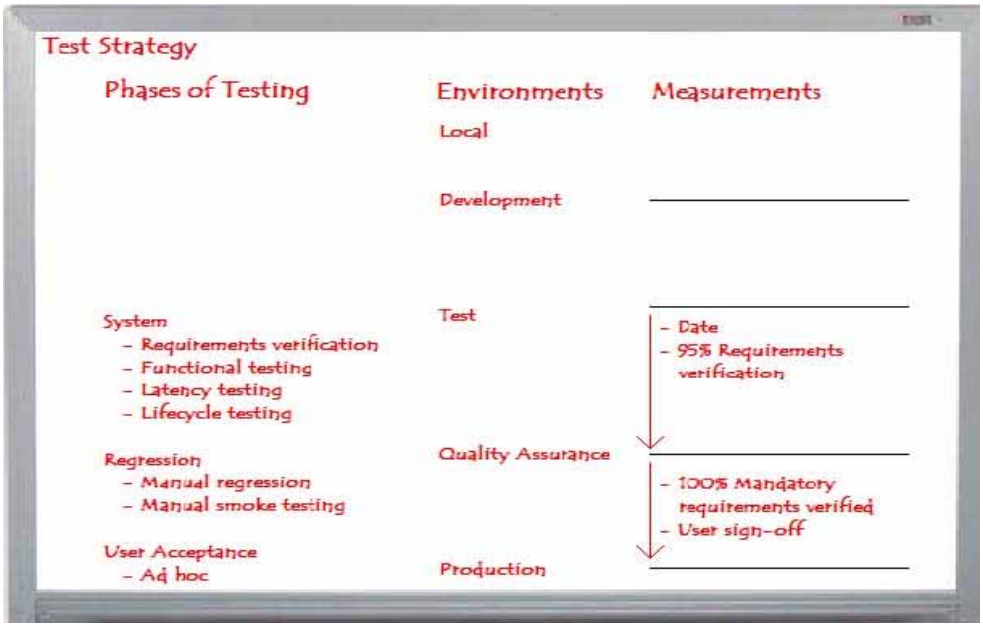


图 2 写字板—-当前的测试状态

第 3 步：突如其来的改进

一旦所有的人都同意了写字板上有关衡量指标的内容，我们就可以开始制定项目测试希望达到的目标了。我们谈论了一些有关实践和工具的内容，后者可以

帮助我们提高效率，并与我们当前的资源（人力和财力）水平相符合。同时，我们意识到，很可能不能再扩大测试组的规模了，这样我们会设法让开发人员为测试提供帮助，而谈论的关注点也随之开始围绕在此问题上。当然，我们也可以把关注点放在我们所经历的那些关键问题上。（还记得前面我所罗列的那些问题吗？）

在第六届 IEEE 关于 Web Site 发展的国际研讨会上，Hung Nguyen 为我们描述了一种制定测试策略的技术—-获得一个“ bug centric ”。他的方法是查看产品中已发现的问题，然后将这些具体问题视为目标，反向地创建策略。他的关注点是设法添加可见性，以此确定成本，来提高产品的发布速度。无论你的上下文背景是什么，都要确保自由讨论的小组能够了解到他们所要解决的问题。如果你的测试策略没有一个明确的完成目标，那么就后退一步，先建立一个清晰的目标。最糟的莫过于你的测试策略具有一个错误的目标—-这个策略注定会以失败告终。它会产生新的问题，也会把现有的问题变得更糟。最好的情况，它偶尔可能会解决一些问题，但同时，会让我们更加的难于解决剩余的问题（实际没有如此困难）。

在我们自由讨论的时候，商定了许多问题，并得到了一些结论：

- 利用单元测试和集成测试，我们可以尽早地发现更多的问题，并准备好自动化测试的初始级别，同时，它们为我们提供了一些衡量指标，这些指标让我们可以更好的跟踪开发过程，这样，我们可以做出决定—-何时移动我们的代码。（多数情况下，我们使用 J2EE 和 Oracle 来构建应用程序，同时，也使用一些其他的技术补充。但不论 J2EE 或 Oracle，它们都具有非常健壮和自由的单元和集成测试工具。）
- 系统测试中，我们以每次发布用户基线为结束，用户基线会增长，同时他们也会逐渐地要求一些更为精确的性能测试—-尽管我们对此还只是略知皮毛。
- 我们不能再依赖于需求验证，不能再继续将其作为我们主要的测试类型了。尽管那是非常重要的事情，因为我们不能忽略安全性测试、可用性测试、配置测试和数据完整性测试，以及上百种其他类型的测试。
- 我们决定进行一些基于 session（session-based）的探索性测试，而最初是以成对的方式执行该测试的，直到我们更为适应这种类型测试的过程，同时，也发展了我们快速学习和解决问题的能力。一旦我们适应了探索性测试的工作，那么我们可以开始执行更多的 sessions。

- 我们发觉，需要建立一个正规的且自动化的烟雾测试，它适用于所有的环境，它和自动化回归测试的脚本集一起被用来测试那些高风险的功能，以及高容量的事务处理。
- 我们知道，用户的接受测试（UAT）远远达不到它应有的效果。因此，我们提出要制定更为详细的 UAT 测试计划，将其与测试脚本和培训材料一起提供给用户，以帮助他们快速地提高。然而，这并不意味着我们希望能够全权负责 UAT 的工作，由我们提供更多的指南、资源和培训来帮助用户进行接受测试，我们的目的只是希望 UAT 执行的更为顺利。
- 我们商定了代码何时可以在环境之间移动的衡量指标。无论是单元测试，还是集成测试，90%的测试通过率对代码而言已经足够了，甚至可以从中了解到一些还会出现的 bug—-只要不存在长期影响系统正常运行的 bug 就行。
- 我们决定要执行严格的代码复查，以保证在早期（更可取的是在写完或接近完成代码时）就发现问题，而不是在代码发布之后。我们创建了烟雾测试之后，代码必须 100%的通过这些测试，这样才能前进到下一个级别。
- 系统测试中，我们无论如何都不能让任何严重或高级别的缺陷遗留到接下来的过程中，但是也存在这样的一些缺陷，是我们所能容忍的，我们可以和用户进行交流，以此来确定他们是否希望问题现在就被修复，还是放在后面解决。
- 我们使用了代码覆盖的测试工具，根据它添加了一些相关的衡量指标，同时根据工具的缺陷趋势分析，来帮助我们衡量系统测试工作的效果。

我在写字板上记录了会议内容，如图 3 所示，分别用不同的颜色进行了标注。

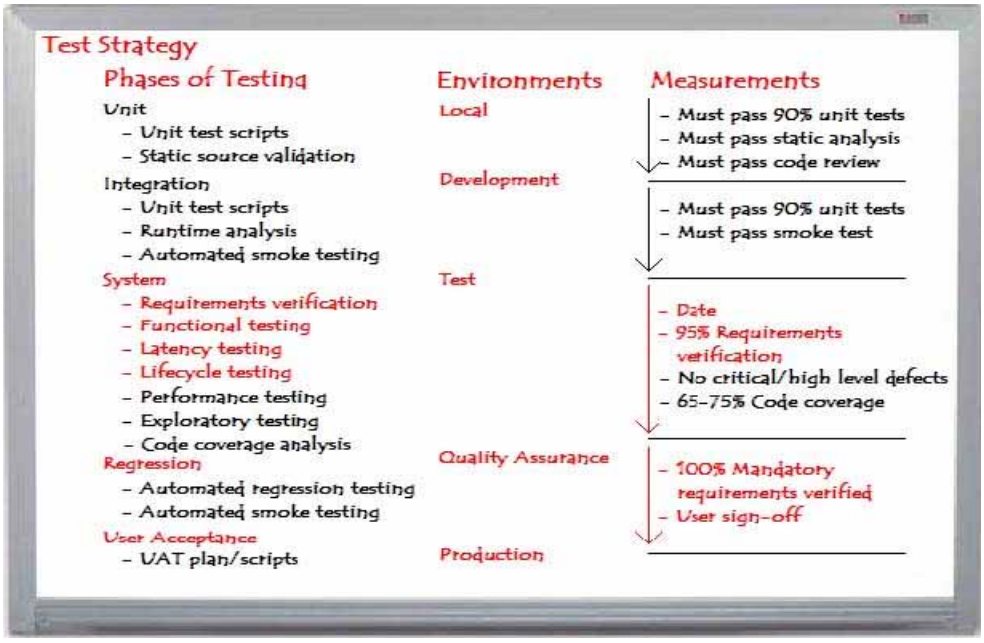


图 3 写字板—-添加的测试类型和衡量指标

第 4 步：组织计划

这个时候，我询问了会议室中的每一个人，一起来检查我们刚才所达成的（写在写字板上）共识，这种感觉就像是我们已经成功地执行了这个计划。下一步，划分职责和活动的实际区域范围。我们花了几分钟时间在写字板上做了相应的标注，如图 4 所示的蓝色方括号和箭头。

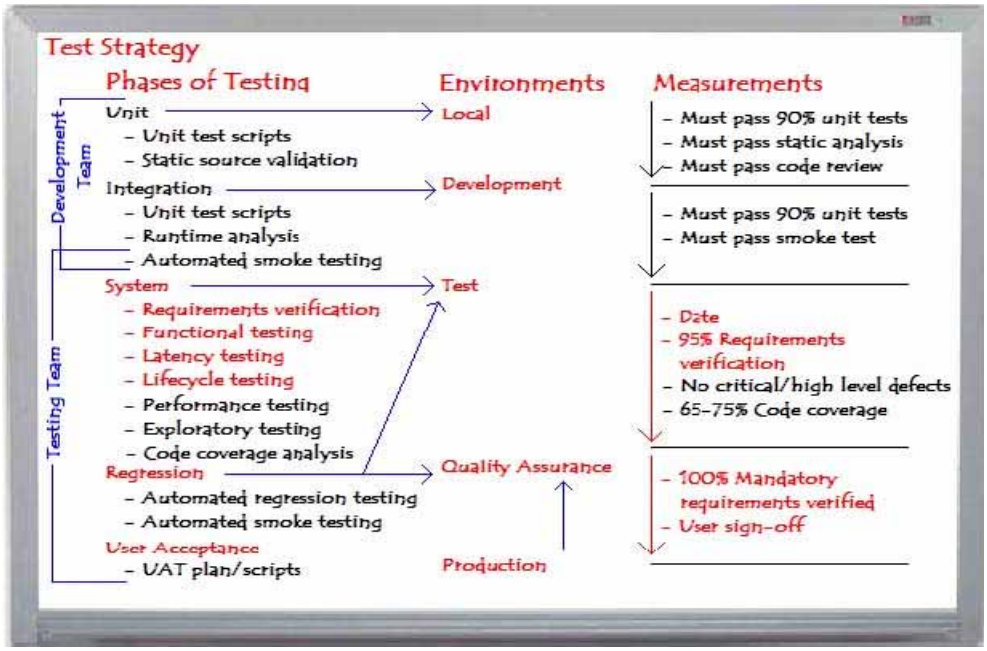


图 4 书写板—-职责、环境

这些分组反映出项目中所包含的工作小组。当然，你的项目可能包含了更多

的工作小组—多个开发或测试组，甚至有独立的行政或 QA 组。写字板上的蓝色箭头表示我们要执行的测试类型与环境的关联关系。虽然还不算完美，但这些内容为我们提供了一个测试提纲，使我们知道了大多数测试工作的分布情况。

第 5 步：组织工具

最后一步，我们需要计划测试中实际所使用的工具，把这些工具添加到我们的测试策略里。在这一方面，该公司以 IBM Rational 的相关产品为主，于是我们确定了主要的测试工具，当然，我们也需要其他一些有帮助意义的工具作为它们的补充。比如单元测试，我们选用 JUnit，因为我们的开发人员知道该如何使用它—-另外，免费和容易上手的特点也是选择它的原因。静态分析，我们选用 Jlint。其他的工具，我们全部选用 Rational 的产品：使用 ClearCase 进行资源和测试资产的控制；使用 ClearQuest 跟踪问题；Purify、Quantify 和 PureCoverage 被用来进行运行期分析；需求管理（rm）工具使用 Requisite Pro；自动化测试使用 Robot 和 TestManager。本来，我们也讨论过使用其他一些运行期分析和资源控制工具，但是考虑到统一的平台更便于我们的管理。图 5 展现的写字板上，包含了这些信息。

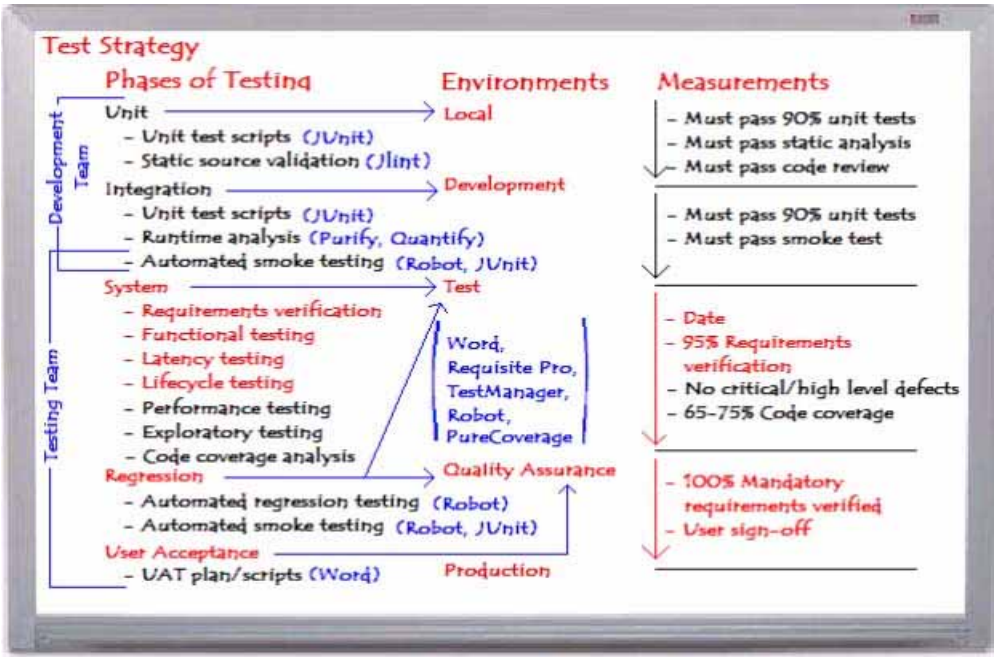


图 5 书写板—-最终所形成的测试策略

完成这些之后，接下来我们可以实施它了。

实施

现在，你已经有了一个策略，将它共享给项目中的每一个人。通过写字板来收集每个人的看法—-或者，更好的方法，使用 Visio 把写字板上的内容转变成幻

灯片。让制定这份策略的人来帮助你解释策略和你要实施的计划。每一个参与策略制定的人都可以帮助你，这样不会让人有这不过是你一个人空想的感觉，并且你会获得来自于整个项目组的支持。回答人们的问题，得到他们的反馈，准备好策略变更。因为有一些人可能知道更好的工具，更合适的技术，或者更有意义的衡量指标。

一旦大家都同意，把该测试策略作为一个可接受的解决方案，那么就可以制定一个实施计划了。在此计划中，回答下面的问题：

- 包含了各新测试类型的迭代过程是什么？（划分测试类型对应的每个迭代过程。）
- 我们如何对之前没有做过测试的小组进行测试培训？（事关测试资源的利用和分配。）
- 我们何时开始安装、配置新的测试工具，并进行相关的培训？（测试工具的使用问题，会影响测试的实际进度。）
- 由谁来负责每个测试阶段的管理工作？（指定一个测试负责人。）
- 我们如何计划这份测试策略的修订和更新工作？（需要控制测试策略的版本变更。）
- 我们如何衡量这份测试策略的有效性？（对该测试策略的效果进行评估，评估的标准是什么？）
- 由谁来负责该测试策略的维护工作？（我们应该有自己的配置管理员来维护这些测试资产。）

更进一步的思考，你会遇到其他一些实施方面的问题，这些和你的项目背景有关。但是，你只要能确保下面的一些情况就可以了：你拥有所需的资源（人、硬件和软件）；你有时间和能力给项目组内的人做相关的培训；你是个越干越起劲的人。

这篇文章所讨论项目的测试策略还没有具体地实施。我们发现了一些变更，比之前面的更具效力。我们已经完成了测试策略，但每次的迭代过程，我们依然关注具体的新工具和新技术，或者关注与人员的培训，以使其具有更高的效率。我们测试策略很简单，它具有的格式也使我们可以容易地修改和更新，在我们开发其他的软件时，发现它不仅灵活，而且很有帮助性。

参考

更多有关制定测试策略的信息，你可以在以下的帮助资源中寻找：

- Lessons Learned in Software Testing: A Context-Driven Approach (Wiley, 2001), by Cem Kaner et al., has an entire chapter dedicated to planning a test strategy.
- Software Testing: A Guide to the TMap Approach (Addison-Wesley, 2001, ISBN 0201745712), by Martin Pol et al., also has a chapter on the topic.
- "Test Strategy and Test Plan," by Jeff Nyman, is an excellent resource for more information on creating a test strategy.
- Try the Heuristic Test Strategy Model (PDF), by James Bach.

更多有关我们在文章中所使用的工具信息，可在以下链接中寻找：

- Jlint (http://artho.com/jlint/)
- JUnit (http://www.junit.org/index.htm)
- IBM Rational (http://www-306.ibm.com/software/)

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=7137&fpage=2

[jody 专栏]

测试计划评估模型

作者：James Bach
翻译：jody

"这个测试计划有多好?"，问题的答案只能用参考测试计划应该是什么样的思想来回答。尽管有大量说明测试计划文档格式的公共标准，但是这些标准基本上没有提供区别测试计划好坏的依据。此文档的模型说明了最基本的概念，测试计划的作用，测试计划应该符合的标准，以及一些对确定标准是否符合特定功能有所帮助的启发：

术语和概念

测试计划

测试计划是一系列指导或者描述预期测试过程的思想。这些思想常常只有部分文被文档化，跨越了多个文档，也经常随着项目的进展而被改变。

测试计划文档

测试计划文档就是那些任何可以传达测试计划信息的文档。然而，测试计划文档不是测试计划的唯一信息来源。测试计划信息还包括项目的口头传授和公司的文化。

测试策略

测试策略是设计和执行测试的方法，用以支持有效的质量评估。测试策略是测试可以覆盖产品的那一部分以及使用哪种测试技术的计划。测试策略与执行策略的细节部署不同。测试策略是整个测试过程的灵魂。。

测试方案

测试方案是执行测试策略所使用的方法、要提交的结果。测试方案是测试过程的主干。

测试计划的作用

测试计划的作用就是测试计划预期要完成的事，下面就是一个理想的测试计划功能列表。然而，测试计划文档可以仅仅表述预期功能中的一部分；其余部分功能或是在其它文档中表述，或是由测试管理者和测试人员个人直接管理，并且这些功能没有任何文档支持。这样，测试计划的好坏只能用测试计划要具有的功能来判断，或者是用其他方法无法完成而策划计划可以完成的功能来判断。

■有利于质量评估，以便给出关于产品的明智而及时的决定

■描述并证明涉及技术需求和技术风险的测试策略（包括所提到的测试覆盖）。促进对测试策略优点和局限性的认识。

■为了保证测试项目的进展，描述并证明所有必须符合的特殊要求或入口准则，，同样，需要描述所有确定何时停止测试的出口或步骤；

■支持测试方案的启动和组织工作：包括事先准备，工作人员的配备，责任的分工，设备需求，任务计划和日程安排。

■支持测试方案和测试策略的每日管理和每日评估。

■支持测试团队的有效协调、合作，以及其它关系，还有测试团队与项目其他人员的有效协调、合作。

■确定和管理那些会冲击项目的风险或问题。

■说明测试方案的交付和交付过程。

■记录历史信息，对过程审核、过程改进以及以后的测试项目都有所帮助。

测试计划质量标准

这些标准与测试计划如何更好的执行其功能有关。在某种程度上，测试计划符合这些标准，那么这就是好的测试计划。准确的讲，测试计划如何好到——“足够好”，有赖于测试环境因素和判断。

■有效性：测试计划是否有效的执行了预期的功能？

■准确性：对相对应的事实描述是否准确？

■效率性：是否有效的利用了可用资源？

■适应性：是否可以容许测试项目合理的变化和发生的不可预期事情？

■清晰性：测试计划自身描述的内容是否一致，是否足够明确？

■可用性：测试计划文档是否简明、可维护、并且很好的组织起来了。

■兼容性：测试计划是否可以接受外部强加的要求？

■基础性：测试计划是否是有效测试计划过程的产物？

■可行性。是否在执行测试计划组织的能力范围内？

测试计划的启发

为了确定测试计划多大程度上符合以上测试标准，我们利用了启发的方法。也就是说，测试计划评估是建立在被广泛接受的经验法则

基础上，而这些经验法则是我们通过实践和学习获得的。

下表中的每一个启发都涉及到上面说明的一个或多个的标准和功能。唯一没有相应启发的是兼容性。这是因为兼容外部强加的需求，需要那些需求相关的专业知识。

我们是用通用的规则和规则的简要基本原则部分来描述每一个启发。这些基本原则的初衷是帮助我们确定何时何处来应用启发。

启　发	启发根据的基本原则
1. 测试应该优先快速发现重要问题，而不是相同的紧急程度下，企图一下找出的所有问题。	在项目中越晚发现的问题，不能及时安全解决的风险就越大。问题存在以后，越早发现问题，无法解决问题的风险就越小。
2. 测试策略应该将大部分注意力集中在有潜在技术风险的部分，然而给小风险部分一些注意力仅仅是怕万一风险分析错误。	完全意义上的测试是不可能的，但是我们永远不可能知道我们对技术风险的理解是否完全准确。
3. 测试策略应该说明测试平台配置、产品如何操作、如何观测产品以及如何利用这些观测评估产品。	草率对待或忽略这四个基本测试活动会增加检测不到重要问题的可能性。
4.测试策略应该使用多种测试技术和观点。评估测试覆盖的方法应该考虑多维覆盖，包括结构、功能、数据、平台、操作和需求。	使用线性方式的单一测试技术不可能暴露所有重要问题。我们永远不可能确保我们发现了所有问题。多样化可以减小测试策略只考虑了某一种问题的风险。
5. 测试策略应该说明如何设计及如何得出测试数据。	通常测试策略是围绕功能和编码而制定的，从而将测试数据留给测试人员自由的编制。这样常常就意味着测试策略太着重于确认功能，而没有充分的考虑可靠性。
6. 不是所有的测试都应该详细的预先说明。测试策略应该包含合理的变化，用测试员根据测试状况推理的能力，将测试员注意力集中在重要的，但不可预期的问题上。	严格的测试策略很可能会使问题的特别子问题没有被覆盖，但是，在一个复杂的系统中，严格的测试策略会减少重要问题没有被覆盖的可能性。测试中合理的变化，如交互式、探测性测试的结果可扩大偶然测试覆盖面，但没有足够扩大基本的测试覆盖面。
7. 根据隐含的需求（需求的充分扩展，不仅仅是需求描述的那样）测试是很重要的。	由于需求一般都定义的不完全，以及描述需求的自然语言固有的不明确性，所以仅根据清晰描述的需求来测试，是不会暴露所有

	重要问题的。
8.测试方案应该推动方案中描述的所有的功能的协同工作，特别是开发人员、技术支持和技术文档。为了更好的了解客户和用户需求，测试人员也要尽可能的与实际的客户和用户沟通 。	其他团队和1常常掌握与产品问题或潜在问题有关的信息，而这些信息对测试团队是有用的。他们的观点有助测试人员对风险更好的分析。测试员也掌握了对自己有用的信息。
9. 测试方案应该与开发者协商，有助于开发者构建一个更可测的产品。	测试策略能否达到它的目的在很大程度上受产品的可测性影响。
10.测试计划应该着重测试策略和测试方案中的那些非常规、方案说明方面。	事实上，任何值得做的软件项目都涉及到特别的技术挑战—优秀测试必须要考虑的技术挑战。一个十分普通的测试计划通常意味着一个差劲的测试计划过程。同样，十分普通的测试计划也只是意味着是一个不变的样本。
11.测试方案应该根据需要，合适的采用手工测试方法或者自动化测试方法。手工测试允许测试过程中对测试的改进和对关键问题的思考，而自动化测试应该被用于那些要求重复次数多，高速度以及不用判断的测试。	许多测试方案受到错误认识的困扰，例如，人工测试使用特定的测试脚本，或者是在测试执行期间，自动化测试复制测试过程中人的思维方式，这对于测试是很有效的。人工和自动化测试不是同一事物的两种形式。它们是完全不同的两类测试技术。
12.应该给出测试日程安排，并且证明测试日程安排的合理性。证明的过程中，需要考虑测试日程安排对软件开发过程的依赖、产品的可测试性、上报问题的时间、项目团队的风险评估等问题。	测试计划中独立的测试日程安排常常意味着测试是独立的活动，这种看法是错误的。测试日程安排可以是完全独立的，不过前提条件产品具有很好的、软件开发可以及时完成，以及测试过程不会被频繁的报告问题打断。
13.测试过程应该尽可能不受关键路径影响。 测试与开发并行进行，发现值得解决的问题比开发者解决这些问题快，可以做到不受影响。	为了转移压力，裁减测试过程是重要的。
14. 测试员和开发者之间的反馈要尽可能通畅。在定义测试周期的时候，应该考虑在回归测试之前，能够对开发人员产品的增加的功能和功能修改作出迅速的反馈。测试人员和开发者任何时候	为了最大程度的提高质量改善的有效性和速度，互相靠近工作是非常重要的。也有助于测试不受关键途径的影响。

都应该尽可能互相靠近工作。	
15.为了有助于测试方案的评估和调整，测试方案应该使用有关质量的信息渠道，而不是正式的测试。 这些渠道包括检视、域范围测试，或者是测试团队以外的人员非正式的测试。	检查产品的质量信息，这些信息是用测试团队之外的不同的方法收集的，这样可以暴露正式测试策略的盲点。
16. 所有与测试策略有关的文件，包括测试用例和过程，应该由一些没有写文档的人员评审。根据文档的重要程度，采用与之匹配的评审过程	视野狭隘对于测试职业是非常大的冒险。评审不仅仅有助于暴露测试设计盲点，而且有助于促进测试人员之间关于测试实践的交流和同行之间的学习。

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=1412&fpage=1

[smartbaby 专栏]

软件缺陷分析：实际测试的建议与测试过程缺陷分析的简单模型

作者：Jon T.Huber
翻译：smartbaby 和 skinapi

摘要

本文主要介绍两个模型。第一个模型扩展当前流行的关于软件缺陷源分类的 Hewlett Packard 模型。这个扩展详述了以软件开发缺陷趋势为基础的软件测试度量。第二个模型考虑软件测试专家将软件测试缺陷归类。期望通过这两个模型，软件测试专家能够更好地分析软件产品和测试的缺陷，使软件测试专家能够贯彻活动的细节以改进软件测试。

介绍

人类行为的任何一个观察家都承认事情发生后的认识总比事情发生前的认识更符合实际。软件测试所面临的挑战就是将前一个项目中所获得的知识运用到下一个项目中去。一个项目结束后的软件开发

缺陷分类和分析的结果常常用于指导下一个项目的软件开发过程的改进。常常被忽略掉的是与软件测试相关的有助于软件开发缺陷分析。既然软件测试的目的是发现缺陷，分析软件开发缺陷特点和趋势，然后测试这些范围，将有助于关注软件测试工程师的工作。本文介绍的第一个模型使得绘制软件测试明确范围的软件开发缺陷趋势图成为可能。

软件开发的缺陷趋势分析在软件工业中是一个公共的实践活动。分析软件测试缺陷趋势是相当重要的，但是时常不被注意到。与可用来进行软件开发缺陷分类的多模型形成对比的是，这里仅仅只有少数模型可以利用，并且被证明对于软件测试缺陷的分类是实用的。本文第二部分将介绍一个模型和存在于软件测试组织中的基础理论，对软件测试缺陷进行分类。

软件开发缺陷分析数据在实际测试中的意义

本章将介绍：

1.介绍 Hewlett Packard 缺陷分类模型

2.从七个 Hewlett Packard 项目中介绍缺陷分类数据

3.说明第一个模型“关注基于缺陷来源的软件测试”的资格

4.应用模型到一个软件开发项目以及接下来的应用

下面的模型（图 1-1），在 1986 年 Hewlett Packard 软件度量协会开发的，被用于下面的描述缺陷分类饼图中（图 1-2 和 1-3）。

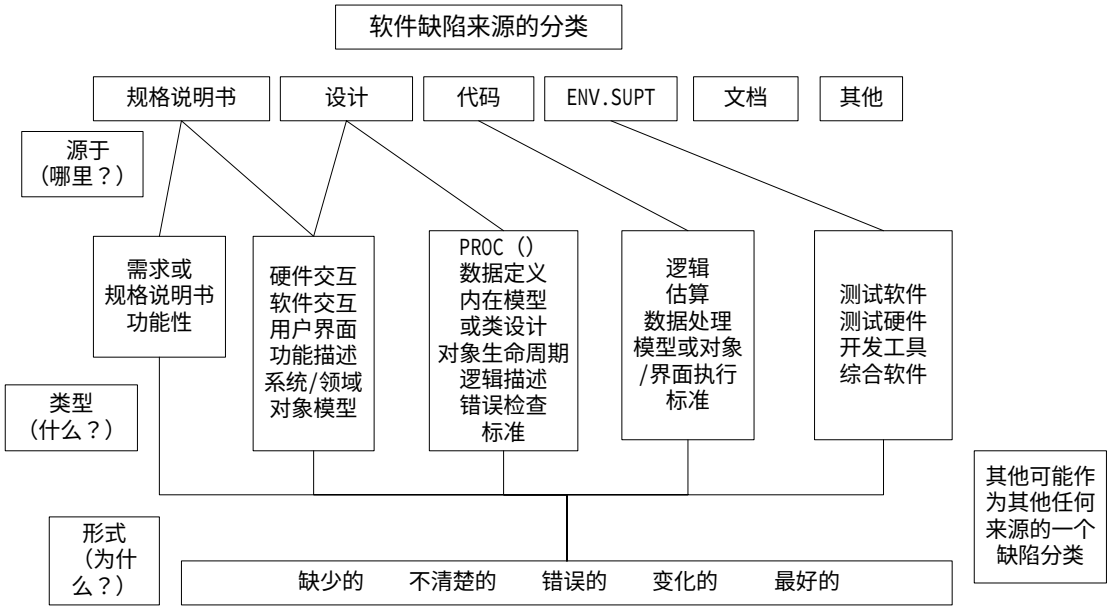


图 1-1

图 1-2 和 1-3 表现了从七个 Hewlett Packard 项目得到的缺陷平均、实际和加权的趋势图。

前八个缺陷来源（实际）

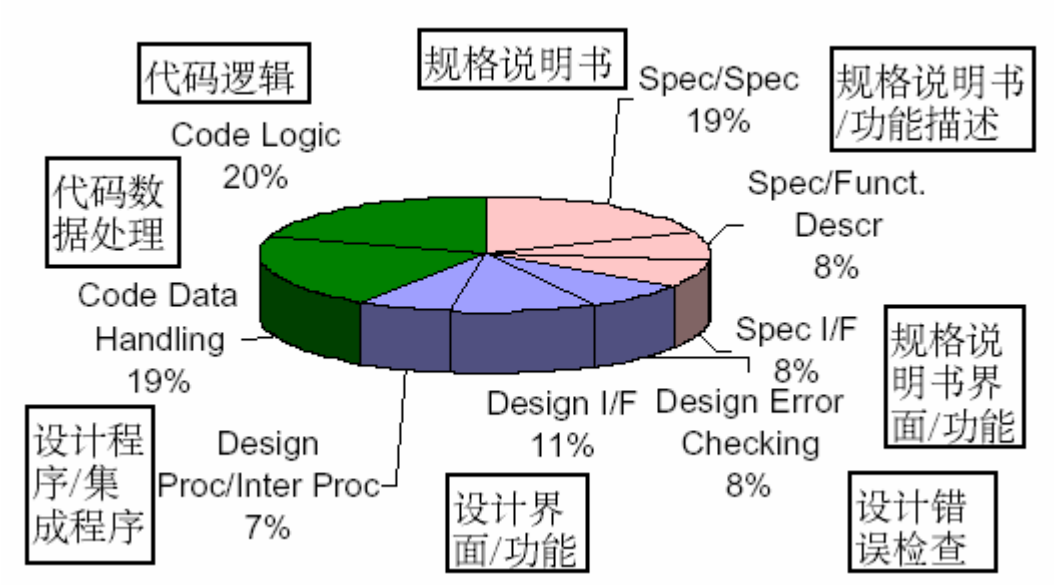


图 1-2

前八个缺陷来源（加权）

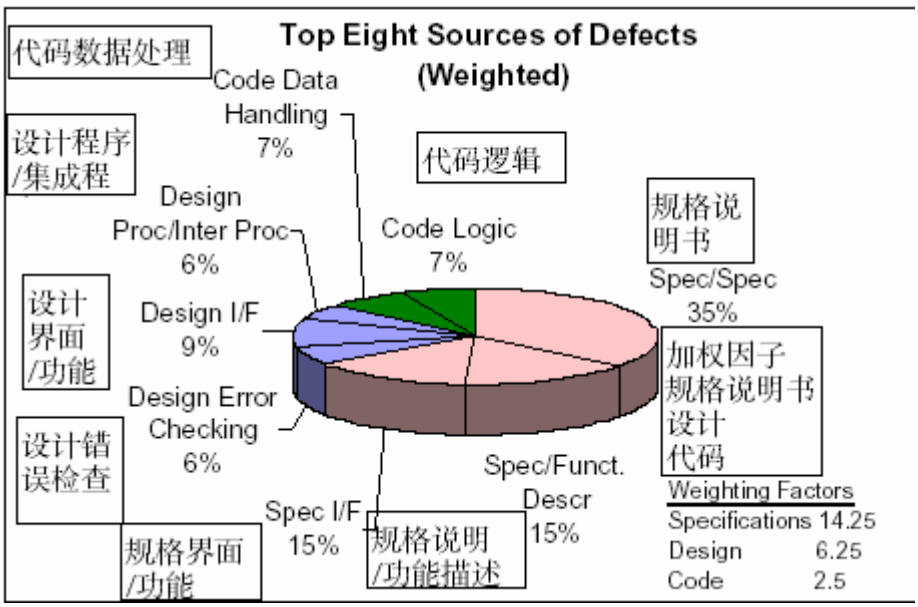


图 1-3

图 1.2 和 1.3 的简要说明

图 1-1 中的开发模型已经成功应用于许多 Hewlett Packard 项目多个领域。第一个问题被问到是为什么用到加权估算。加权估算的要领在于所有的缺陷并不是均等产生的。换句话，修复一个源于规格说明书的缺陷要比修复一个源于代码的缺陷使开发团队消耗更多的精力和时间。提到这个工作产品的数量的主要原因是由于我们需要改变修复一个规格说明中发现的缺陷与修复一个代码级的缺陷的比例。举例说，一个开发人员要为了改一个规格说明发现的缺陷，需要改规格说明书，需要修改设计，更改代码。而另一方面，一个代码错误仅仅需要更改一个工作产品，即源代码。图 1-2 中给出的是公认的加权因子，这些加权因子可能在一个软件开发项目与在其他软件开发项目是不同的。记住在任何一个软件开发项目中都存在这样一个重要的因素，加权因子。

测试建议（模型）

下面提到的模型（图 1-4）将给出缺陷分类，这些分类是以他们的定义为基础，最可能出现的适当的软件测试阶段。以使用在图 1-1 中模型介绍的经验为基础，大多数软件开发缺陷丛生在规格说明书、设计或者代码中。这个模型主要集中在绘出源于三个特别的软件测试阶段中的缺陷类型。软件测试工程师应该非常关注那些缺陷丛生的测试范围。这个模型将起到帮助的责任。使用这个模型，可能会推翻工

程师在特别范围的缺陷分类以及猜测出哪些测试能发现哪些缺陷。

基于源代码的软件测试重点

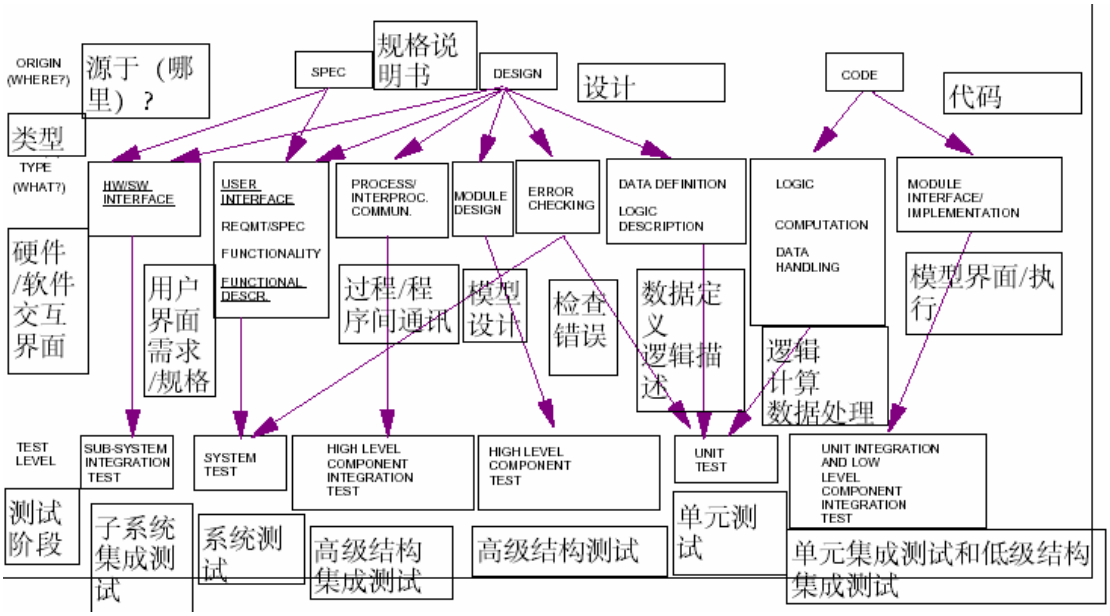


图 1-4

适用于模型的工业测试阶段

接下来的描述仅仅适用于图 1-4 中涉及到软件测试阶段。

单元测试－测试可以单独测试的小块的软件（比如，装配、编译、载入、测试）。通常一个程序员的工作包括几百行的源程序。对于一个打印机，一个单元的例子可能就是设置纸张方向的功能。

单元/低级结构集成测试－测试针对用到的界面和与其他已经测试正确的单元结构有交互的缺陷。对于一个打印机，低级结构可能就是所有的“页面设置”的设置（比如，页面尺寸，纸张来源，拷贝数以及方向）。这个阶段的一个集成测试将测试跟每一个“页面设置”的设置有关的交互。

高级结构测试－测试结构接近需求树（一个表示需求结构组成的图形）的顶端。就打印机来说，软件打印驱动就是一个高级结构的例子。

高级结构集成测试－测试针对用到的界面和与其他已经测试正确的高级结构有交互的缺陷。就打印机来说，测试包括软件驱动与其他高级结构比如安装和反安装的交互。

子系统集成测试－测试针对用到的界面和与其他已经测试正确

的子系统有交互的缺陷。就打印机来说，可能包括测试关于打印机韧件，软件和硬件的交互。

系统测试－测试针对用户如何使用产品。就打印机来说，可能包括测试加 3rd 部分的应用和操作系统，但是并不包括打包、学习产品等等（解决测试）。

关心范围的缺陷类型建议测试

规格说明书的缺陷类型

如果缺陷分类显示了大量的规格说明/需求中的缺陷，可能要考虑的是渐增的系统测试的数量和类型。

规格说明书的缺陷类型似乎是最适合系统测试的阶段的，因为绝大多数的缺陷与用户或者产品使用有关。在 Hewlett Packard 缺陷类型模型中，规定了规格说明书的缺陷是“在一个系统或者系统执行误解了用户或者目标需求定义的错误”。许多规格说明书中的缺陷被发现有缺少的，不正确的或者添加功能的错误。然而，很可能这些缺陷最好通过系统测试来显示，考虑比如适时性，规格说明书的测试（测试功能性），处理流程（软件程序如何运行），特写交互，配置，边界配置，硬件配置，性能、恢复性、安全性等等。测试这些范围可能更接近于模仿用户如何使用产品。

在“硬件和软件交互”的范围中关心规格说明书的缺陷，并不意味着增加系统测试的重点。对于一个打印机来说，至少硬件和软件是子系统，因此相当大的缺陷类型范围证明需要增加子系统集成测试的重点。

设计缺陷类型

显示设计缺陷类型的测试阶段相当分散，“用户交互”设计类型最好在系统测试中进行。像在定义中规定的缺陷类型“错误设计的问题将如何影响它的环境和/或用户”。由于这里的重点在用户或者使用者上，大量的缺陷类型显示应该增加系统测试的重点。

“程序/程序间通讯”设计缺陷是“不正确的交互和产品中的程序间通信”。关于交互关系的建议是“高级结构集成”测试，在这个阶段主要产品处理过程的交互已经被测试过了。一个需要加强的地方是不同交互测试的技术应该是适当的。比如呼叫对和域的测试技术，

呼叫对测试集中在使用产品的主要处理程序的交互。域测试检查极端值的一个或者多个输入变量来检查高级结构间的交互和通信。

关注“硬件和软件交互”的设计缺陷指出应该加强子系统集成测试。

“模型设计”的定义缺陷类型规定“过程的控制（逻辑）流和执行的问题”。这个“过程”可能最接近于上面提到的测试阶段的高级结构。高级结构测试主要针对产品的重要过程。同样地，“模型设计”缺陷的关注点应该引发是否测试人员应该更认真地进行高级结构测试的检查。高级结构测试应该尽可能出现在最孤立的上下模块间。可能的技术是私有分支结构测试、测试套件的自动回归以及对非法值、无限制值的比率增加的测试。

“数据定义”，“逻辑描述”和“错误检查”设计缺陷类型主要建议一个不同单元测试技术的图。每一个这些缺陷类型的主要原因是与内部结构和程序的数据相关的。软件工程师和开发人员对于大多数的单元测试有责任，因为他们是除了测试外最熟悉内部工作和软件产品设计的人。

“数据定义”的设计缺陷，就像“在模型/产品中使用的不正确的数据结构设计”定义的一样，以及“逻辑描述”就像“在传递预期的运算或者逻辑流时使用到不正确的数据”。基于这些描述，在使用数据流技术时这些缺陷类型将更有效。这里一个结构化测试的例子，发生在决定那些路径需要测试时要考虑数据对象（事件的顺序可能会改变数据对象的状态）

“错误检查”设计缺陷的百分比意义重大说明应该更关注单元测试。明确地，无论已经写好的功能/单元是否像预期的那样挡道，程序员都必须对这些功能/单元进行完全测试。这个测试包括单元是否适当地处理错误。如果规格说明书或需求中的“错误检查”缺陷更明确的话，在系统测试阶段可能需要投入更多的状态和过渡测试。一个精确的图解状态应该清晰的给出指定软件操作的输入和输出，比如像这种高水平的错误条件。

代码缺陷类型

基于“逻辑”定义的代码缺陷，是“被忽略的案例和步骤、重复的逻辑、被忽视的极端条件、不必要的功能或者误解等错误”，通过

逻辑基础的单元测试技术可以发现绝大多数的缺陷。

类似于“数据定义”和“逻辑描述”的设计缺陷，代码“数据处理”和“计算问题”缺陷可能在单元测试阶段使用数据流测试更有效。这两种缺陷都必须与代码是如何进行数据编译和计算打交道。

“模型交互界面/执行”缺陷类型的百分比意义重大说明单元和低级结构的交互界面的弱点。这些缺陷类型暗示在呼叫对和域测试中应该更关注交互界面的测试技术。

项目的模型应用

模型描述包括了软件测试人员分析软件缺陷数据类型和趋势的方法。模型和不同类型的测试说明可能适合基于多种软件缺陷类型的提炼，是对软件测试工程师有帮助的一个工具。如果使用恰当，软件测试工程师可以更好的确定在刚刚结束的项目中缺陷是从哪里丛生的，在不久的将来决定关注哪些方面的测试。通过缺陷的数据分类例子，本文将展示模型是如何应用到项目中的。

缺陷变化百分比

一个项目的基线比较

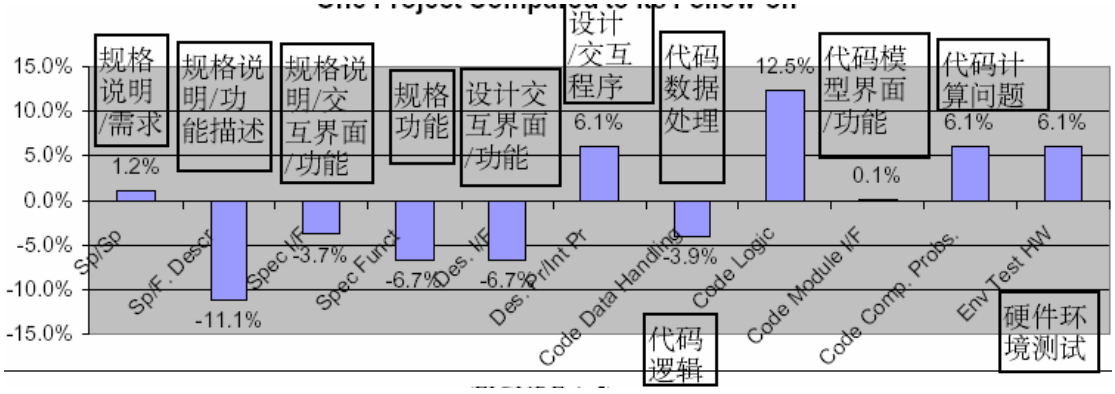


图 1-5

图 1-5 中显示的数据描述了一个项目与其基线相比缺陷类型百分比的增大或者减少情况：

-规格说明书缺陷——减少 20.3%

-设计缺陷——减少 0.6%

-代码缺陷——增加 14.8%

通过这些数据，看起来针对减少规格说明书缺陷百分比的过程改

进，比如早期测试的一个规格说明书检查活动，事实上可以从缺陷的来源来改善缺陷百分比。源于规格说明书中的缺陷类型在“需求或规格说明书”中会越来越多。最近参于需求工程活动的项目针对阐明和改进需求，期望这些新的需求能够增强这个产品线已经启动的系统测试度量。

并不是所有的缺陷分类百分比都像设计的缺陷分类百分比一样是意义重大的。例子中的项目努力创造和检查界面文档，这个可能是为什么“设计交互界面”缺陷不出现在基线产品百分比图表上面的原因。除了“设计交互界面”缺陷，基线项目的缺陷可归于“设计程序/程序间”缺陷。这些缺陷类型指出应增加高水平的结构呼叫对和域测试。

代码缺陷从一个项目到下一个项目增加 14.8%。对于两个项目来说，在“数据处理”、“逻辑”和“模型交互界面/执行”内容上出现意义重大的缺陷类型。在基线项目中，也存在线代码缺陷类型范畴的“计算问题”。根据模型，“模型交互界面/执行”缺陷指出需要更关注单元/低级别的结构集成测试。界面测试技术比如附件对和域测试技术应该在这个阶段考虑。“数据处理”和“计算问题”缺陷指出在单元测试阶段处理数据的困难。增强数据流测试应该有助于揭示这些领域的其他弱点。代码“逻辑缺陷”指出增强对哪些可能发生在程序逻辑流中的关注。逻辑基础的测试技术，比如使用 Boolean Algebra 进行测试代码，增加发现改进“代码逻辑”缺陷的机会。

每一个软件开发项目都有不同的缺陷趋势，比如一个项目与另外一个项目的测试关注的重点就是不一样的。上面的说明阐明了任何一个软件开发项目如何从他们自己的项目中收集缺陷分类数据来改进测试度量。以这些测试组织的结果为基础来关注测试度量可以：

1. 在实际项目缺陷数据中更好的计划测试工作量
2. 关注测试在哪些最可能隐藏缺陷的地方
3. 假定当前产品是基于前一个项目可重用和/或优势代码，当前的产品能够很好的利用它来增加改进。基于前面刚刚结束的项目的缺陷趋势来调整当前项目的测试重点，可以实现这个目标。

为了软件开发项目的分类和定义缺陷类型，软件工业中产生了许多工作。这些工作的重要性被认为是略低于软件测试的。本文推荐一

个帮助软件测试进行专业分析软件测试缺陷趋势的模型。

软件测试缺陷分析的一个简单模型

本文的这一部分将：

- 介绍并详细说明第二种名为“软件测试缺陷分类”的模型。
- 讨论成功使用该模型所必须的环境。
- 提供一个通过应用该模型到项目中来产生分析数据的例子。

尽管业界对这个话题有不同的观点，但还是应该将软件测试缺陷的记录和分析看成一个普通的习惯。用于分类软件过程中缺陷的 Hewlett Packard 模型成功的一个原因在于该模型中的类型数足够少，使得工程师对他们的缺陷进行分类时没有什么困难。这些类型的划分基于导致缺陷的原因，而不是基于各种失败的可能原因。从而使得该模型将测试缺陷的类型数限制在可管理的范围内。

现实中至少有三方面的软件测试缺陷，分别是测试过程缺陷、测试管理缺陷和被测产品缺陷。测试过程缺陷是指进行成功的测试所必须的策略和过程中所包含的错误。当进行成功的测试所必须的资源和政策没有到位时，就可以认为存在一个测试管理缺陷。而那些导致测试本身改变的错误被称为被测产品缺陷。图 1-6 中所描述的分类模型包含了可归为以上三方面缺陷之一的缺陷类型。

下面所描述的模型部分基于 Boris Beizer 博士所定义的“测试定义 Bug 或者测试执行 Bug”。

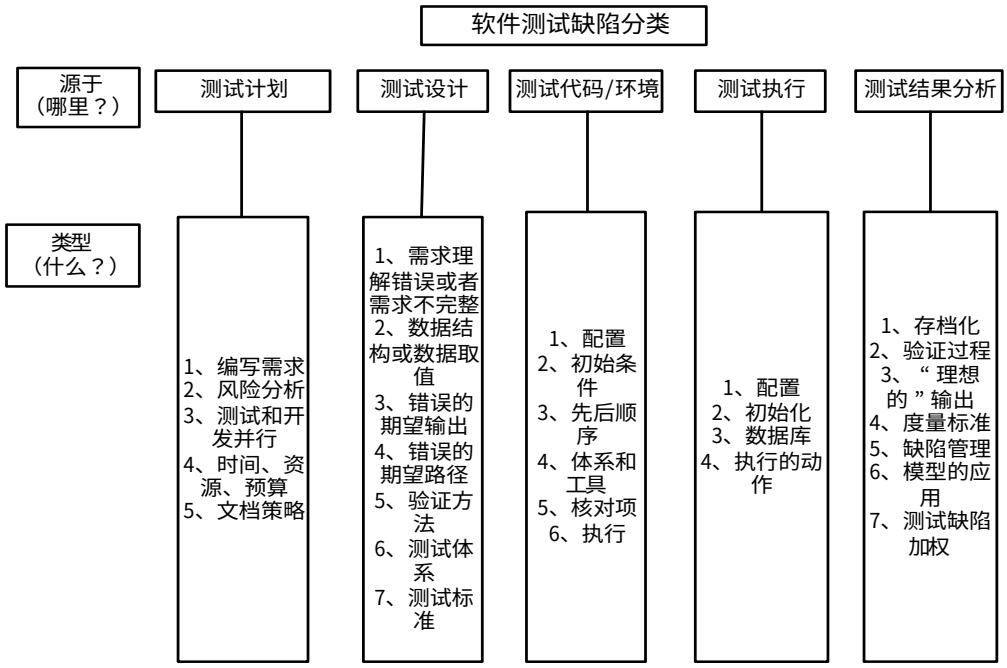


图 1-6

测试计划

编写需求

测试需求总是存在的，问题在于它们被记录在什么地方。这种类型的缺陷是指文档化的测试需求根本不存在或者是低质量的。

风险分析

任何测试过程都存在风险。这种类型的缺陷是指没有在测试计划中包含正常的风险分析。这些风险包括测试新的技术、采用新的测试体系和工具、测试外包、测试生命周期中某些阶段缺乏资源、基于多协议或多语言的测试等。要想成功的进行测试，就必须对这些风险区分优先级、制定应对意外的计划、确定测试策略中的其它各种漏洞。

测试和开发并行

在产品生命周期的某些阶段，软件开发人员和软件测试人员的协作是必须的。当最初的需求产生后，测试人员和开发人员需要通过沟通确认需求是可测的和能被实现的（可编码的）。这时候，开发人员开始创造这个产品而测试人员开始进行测试。一旦产品的测试过程中发现缺陷，测试人员和开发人员之间就必须再次进行沟通。通过分析

来判断这个缺陷到底是一个测试缺陷还是一个开发缺陷。在制定最初的需求时或者出现缺陷时，如果测试人员和开发人员之间的沟通不顺，就产生了这种类型的缺陷。

时间、资源、预算

测试部管理上的支持是基本的要求。当只有少量或者没有管理上的支持并且没有制定一个实际的包含充足的时间、资源和预算的测试进度表时，这种类型的缺陷就出现了。

文档策略

判定哪些文档对于测试过程的成功是最基本的很重要。最少应该有书写的测试需求和一个全面的测试计划。对于任何项目，计划和评估总是随着实际情况而不断变化的。因此必须有一个用于评审测试需求和测试计划变更的变更管理策略存在。这个变更管理策略必须得到开发人员和测试人员的一致同意。如果未经双方的协商测试需求和测试计划就发生变更，这种类型的缺陷就发生了。

测试设计

需求理解错误或者需求不完整

由于测试设计者没有正确理解需求，导致测试和被测的单元不匹配。一旦该测试被应用到被测产品上，该问题就显现出来了。这种类型缺陷产生的另外一个可能原因是提供给测试的软件需求没有完全的写入文档。设计的测试用例最少应该记录配置信息、开始条件、测试执行的步骤、期望输出等。

数据结构或数据取值

测试中用到的数据对象或者它们的值错误。这种类型的缺陷由错误设计的数据结构或者测试中给数据赋错误的值所导致。

错误的期望输出

测试的期望输出和实际正确的输出不一致。

错误的期望路径

输出是正确的但是通过错误的期望路径获得的。这种情况下测试只是碰巧正确通过。

验证方法

用于验证输出的方法是错误的或者不可行的。

测试体系

用于构建测试的方案或者方法不存在或者错误。同时，被测对象没有很好的写入文档或者被理解。

测试标准

测试设计并没有满足大家都接受的设计标准。

测试代码和测试环境

配置

硬件配置、软件配置或者测试环境是错误的。测试设计中的配置部分写成了文档但是错误的。测试运行的环境不正确。配置包含多种因素如操作系统、应用、外围设备、网络等。

初始条件

为测试指定的初始条件是错误的。设计的测试用例的开始条件已写入文档但是错误的。这样的例子包括测试用例开始时数据变量和函数参数的取值错误。

先后顺序

测试执行的先后顺序错误。

体系和工具

测试集的管理和执行工具以及正确使用它们所必须的体系没有正确建立或者根本不存在。

核对项

这种类型的缺陷包括缺少退出和进入测试的标准、发布记录、获得代码的流程、成功测试所需的资源、在测试执行之前搭建环境所必须的其它核对项。评估测试环境时需要考虑以下因素：从开发团队接收到了哪些东西，是否可以开始测试以及怎样判断这一点等。

执行

用于测试所必须的测试代码有错误。也可能是测试工具或者其它用于测试的工具有错误。

测试执行

配置

执行过程中测试所指定的配置和环境没有用到。指定了一个正确的测试环境，但由于测试执行的一个错误，该环境并不是实际用到的环境。

初始化

被测组件没有被初始化成正确的状态或值。测试的开始条件正确写入文档但由于测试执行的一个错误导致初始化错误。这样的例子包括测试开始执行时数据变量和函数参数赋值错误。

数据库

用于支持测试的数据库错误或者数据没有正确输入。

执行的动作

简单的比如按键或者按按钮错误。但是这种类型的缺陷有可能比这种简单的输入错误更复杂。例如，由于某些原因当测试执行时它没有按照设计来执行，这些因素比如网络流量、机器停工、测试所依赖的工具失效等。

测试结果分析

存档化

测试结果未存档。没有记录已执行的测试情况。最后一次运行测试用例时，保存测试结果以便将来查看是很必要的。而且随后的项目也经常需要参考前面项目的测试结果。

验证过程

验证输出的过程没有正确执行。为验证输出所定义的方法是正确的但操作错误。

“ 理想的 ”输出

这种类型缺陷的产生在于没有任何更新测试“ 理想 ”输出的过程。

除非测试执行的结果和测试的理想输出进行了比较，很多测试都不能认为是通过的。如果测试执行的结果和“理想”输出保持一致，那么就说这个测试通过。而我们经常发现的是，当同一个测试在产品某一版本和其下一版本执行时，该测试“理想”的输出需要进行修改。

度量标准

没有合适的东西来度量测试过程和产品的状况。有些度量程序关注被测产品的状况而不是测试过程和产品的状况。通过有效性和效率来度量测试过程很重要。用于度量的一些重要因素是：测试中发现的缺陷、不同类型的覆盖、自动化的程度、估计的开销和实际开销的比较、满足进度的能力、测试团队生产能力。

缺陷管理

缺陷必须由可靠的测试组织来管理。这种类型的缺陷产生于缺陷管理系统只是关注开发的产品而没有尝试将缺陷和发现它们的测试联系起来。

模型的应用

为了成功应用“软件测试缺陷分类”模型，有以下几点需要考虑：

- 在缺陷管理系统中产品缺陷和测试缺陷存在差别。这就意味着这两种缺陷类型都要记录在缺陷管理系统中。
- 测试工程师必须在第一时间记录测试缺陷。
- 在每个测试项目之后，测试团队都要负责对缺陷进行分类。

对于任何失效分析和过程改进而言，只分析一次趋势收效甚微。为了获得充分的好处，团队必须负责在每个项目后对缺陷进行分类，将过程改进应用到下一个项目，然后重新评估缺陷的趋势。这将有助于估量过程改进是否使软件测试得到改善。假设在这个虚构的例子中测试缺陷的分布趋势如下所示：

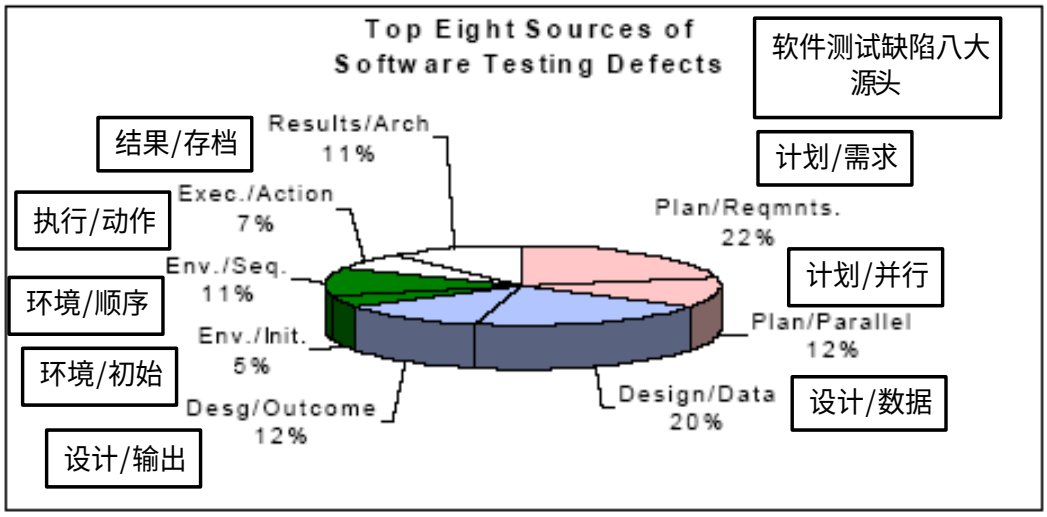


图 1-7

如果这些数据来自于一个真实的项目，测试团队就需要考虑重点在哪些方面进行测试改进。从图中可以看出缺陷更明显的集中在测试计划和测试设计。如果团队决定将过程改进重点放在测试设计上，下一步将是分析相当百分比的缺陷集中在测试设计的根本原因。这样从该项目得到的改进策略就变的明显了。该改进策略会被应用到下一个项目中去。当下一个项目结束时，需要再次分析缺陷趋势以判定测试改进到底有没有作用。通过这种不断的改进，软件测试缺陷分类和根本原因分析所带来的好处就会变得明显。

测试缺陷加权

如前所述，软件开发中的缺陷的影响不是完全相同的。这一点同样适用于软件测试中的缺陷。归类为测试计划缺陷或测试设计缺陷的缺陷相对于测试代码/环境缺陷、测试执行缺陷和测试结果分析缺陷而言更难理解和修复。例如，初始化一个很重要的数据变量为一个错误的值就没有检测和修复测试设计中没有考虑测试这个变量的缺陷困难。关于加权因子应该随着软件测试项目的变化而变化已经形成了共识。需要牢记的非常重要的一点就是对于任何软件测试项目，加权因子都是存在的。

结论

软件开发工程师开始分析缺陷趋势并基于统计数据进行改进已经很多年了。现在是时候软件测试工程师进行同样的工作。这篇文章

介绍了两种模型。第一种模型有助于弄清软件开发中缺陷的趋势，并在这些趋势的基础上来判定测试在将来应该关注哪些方面。第二种模型介绍介绍了一种分类软件测试缺陷的方法，使得能够更好的理解项目测试生命周期中哪些改进是可行的。两个模型都可以让软件测试组织获益于事后经验的积累。这种积累来源于软件开发和软件测试缺陷趋势相关数据的分析。并且这种积累可以转换成定位和改进软件测试实践的推动力。

致谢

感谢我的经理 Len Schroath 和技术经理 Anne Vermilion。感谢他们对手下员工的充分信任。

还要感谢我的同事 Susan Davis,、Mike Dunlap 和 Felix Silva，他们对这篇文章提供了有价值的建议。

同时也要感谢 Bob Grady，他写了三本书和大量的文章并介绍我进入软件失效和根本原因分析的领域。没有从他的经验和著作中所学到的东西，我是不可能写出这篇文章的。

参考文献

1. Grady, Robert B., “Practical Software Metrics for Project Management and Process

Improvement”. Prentice Hall, Inc., (1992), pp. 127-128, 223-227.

2. Boehm, B., “Software Engineering Economics”. Englewood Cliffs, NJ: Prentice-Hall, Inc.,

(1981), p. 40.

3. Beizer, B., “Software Testing Techniques”. Boris Beizer, (1990). Printed by Van Nostrand

Reinhold, NY. Software Testing Levels, as contained in (Figure 1-4) and the explanation

following the diagram, are based in part on definitions in various sections of “Software Testing

Techniques”.

4. Beizer, B., “Software Testing Techniques”. Boris Beizer, (1990). Printed by Van Nostrand

Reinhold, NY, pp. 475-476. The defect types as contained in (Figure 1-6) and the explanation

following the diagram are based in part on Bug Statistics and Taxonomy as defined in

“Software Testing Techniques”.

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=1741&fpage=1

[binary 专栏]

每日构造与冒烟测试

作者：Steve McConnell
翻译：binary

如果你想创建一个只包含一个源程序文件的简单程序，那么你只需要编译、连接那一个文件就可以了。如果是一个团队项目组，有着许多甚至上千个源程序文件，那么要创建一个可执行程序的过程就变得更复杂、更耗时。你必须用各种各样的组件将程序逐步建立起来。

在微软或其它一些软件公司中惯例是：每日构造并做“冒烟测试”。每天都对已完成的源程序进行编译，然后连接组合成可执行的程序，并做“冒烟测试”，以简单的检查该执行程序在运行时是否会“冒烟”。

带来的好处

虽然这是一个非常简单的过程，但却有非常重要的意义：

1、能最小化集成风险

项目组可能遇到的一个很大的风险是，项目组成员根据不同的系统功能各自开发不同的代码，但是当这些代码集成为一个系统的时候，也许系统完成不了预期的功能。这种风险的发生取决于项目中的这种不兼容性多久才被发现，由于程序界面已经发生了变化，或者系统的主要部分已经被重新设计和重新实现了，相应的排错工作将非常困难和耗时。极端情况下，集成的错误可能回导致项目被取消掉。每日构造和冒烟测试可以使这种集成错误变得非常小，而且便于解决，防止了很多集成问题的产生。

2、能减小产品低质量的风险

这种风险是和集成不成功、集成出错相关联的。每天对集成的代

码做一些少量的冒烟测试，即可杜绝项目中那些基本的质量问题。通过这种方式，使系统达到一种周知的良好状态，维护这样的系统可以防止系统逐步恶化到耗费大量时间排查质量问题的地步。

3、能简单化错误诊断

当系统每天都进行 build 和测试时，系统任何一天发生的错误都能够变得十分精细，便于排查。比如在 17 日系统还运行正常，18 日就出错了，那么只需要检查这两次 build 之间的代码变化就可以了。

4、能极大鼓舞项目组的士气

看到产品的不断成长，能够极大的鼓舞项目组的士气，有时甚至不管这个产品到底用来做什么。开发人员可能会为系统显示了一个矩形而感到激动。通过每日构造，产品每天进步一点点，保证项目士气的持续高涨。

进行每日构造和冒烟测试

虽然说这是一个简单枯燥的活，每天进行 build，每天进行测试，但也有着一些值得注意的细节：

1、每天坚持

每日构造，最重要的就是“每日”。如 Jim McCarthy 所说，把每日构造看作是项目的“心跳”，没有“心跳”的话，项目也就死了(Dynamics of Software Development, Microsoft Press, 1995)。Michael Cusumano and Richard W. Selby 描述了另外一种隐含的比喻，把每日构造比作项目的“同步脉冲”(Microsoft Secrets, The Free Press, 1995)。不同开发人员写的代码在他们的“脉冲”之间肯定都会存在“同步”的差异，但是必须有这样一个“同步脉冲”，使得这些代码能够组合为一个整体。当项目组坚持每天把这些不同的“脉冲”组合到一起的时候，开发人员脱离整体的情况就会得到极大程度的杜绝。

有些项目组把这一过程简化为“每周 build 一次”。这样带来的问题是，某一次 build 失败后，可能要回溯好几周才能找到原因。如果这种情况发生的话，已经得不到经常 build 带来的好处了。

2、严格检查每一次 build

要保证每一次 build 的成功，就必须保证 build 后的结果（也可称为 build）是可以正常运行的，如果 build 不可运行，那么本次 build 被认为是不成功的，同时应该将修复此次 build 的工作提高到项目组

最高级别来处理。

对于如何衡量一个 build，每一个项目组都会定义一些自己的标准，这些标准需要设定一个严格的质量级别来处理那些特别严重的缺陷，同时也需要具有一定的伸缩性来忽略掉那些微不足道的缺陷，一些不适当的关心也许会使整个过程举步为艰。

一个好的 build 起码应该具备以下条件：

●能够成功编译所有的文件、库，以及其它相关组件；

●能够成功链接所有的文件、库，以及其它相关组件；

●不能存在任何使得系统无法运行或者运行出错的高级别故障；

●当然，必须通过冒烟测试

3、每天进行冒烟测试

冒烟测试应该是对整个系统流程从输入到输出的完整测试。测试不必是面面俱到的，但是应该能够发现系统中较大的问题。冒烟测试应该是足够充分的，通过了冒烟测试的 build 就可以认为是经过充分测试、足够稳定的。

不进行冒烟测试的 build 是没有太大价值的。冒烟测试就像一个哨兵，在阻止着产品质量恶化和集成问题的产生，不进行冒烟测试，每日构造可能会变成浪费时间的练习。

冒烟测试必须随着系统的扩充而扩充。最初，冒烟测试可能是非常简单的，比如验证系统是否会打印“ Hello World ”，随着系统功能的扩充，冒烟测试需要越来越充分。最初的冒烟测试也许只需要几秒钟来执行，逐渐地，测试可能会花费 30 分钟，1 小时，甚至更长。

4、建立一个专门的 build 小组

在很多项目组，维护每日构造，并更新冒烟测试用例，会耗费一个人工作的大部分时间。因此在一些大的项目中，这项工作独立成不止一个人来完成的全职工作。比如在 Windows NT 3.0 的研发中，就有一个由四个全职人员组成的专门的 build 小组(Pascal Zachary, Showstopper!, The Free Press, 1994)。

5、为 build 增加修订，如果这样做有意义的话

一般开发人员不会每天都经常向系统中快速的增加实际的代码，通常是每隔几天，他们在开发好完成某个功能的一套代码以后，然后集成到整个系统中。

6、规定一些导致 build 失败的惩罚措施

很多执行每日构造的项目组都会规定一些惩罚措施，来惩罚那些导致 build 失败的行为。从最开始，项目组成员就清楚的知道，build 的正常执行是项目组的头等大事。一个失败的 build 是项目组的意外，无法成为项目组工作的准则。必须坚持：导致 build 失败的同事，必须停下手中的工作，首先来解决 build 失败的问题。如果一个项目组的 build 经常失败的话，久而久之的，再来谈 build 的正确性就没有意义了。

有种轻松的惩罚措施，能够突出解决问题的优先性。Some groups give out lollipops to each "sucker" who breaks the build. This developer then has to tape the sucker to his office door until he fixes the problem. 有些项目组会惩罚犯错的同事戴上山羊角，或者向一个项目基金捐献 5 块钱。

有些项目组对此的惩罚就有点残酷了。微软的开发人员，在一些知名度很高、很重要的产品如 Windows NT，Windows 95，Excel 等产品后期研发中，被要求随时带着寻呼机，如果你的代码导致 build 失败的话，即使是凌晨 3 点钟，也会要求你立即来处理这个问题。

7、即使在压力下也需坚持每日构造和冒烟测试

当项目进度的压力越来越大时，维护每日构造的工作看起来有些浪费时间，但是恰恰相反。在压力之下，开发人员丢掉一些平时的规定，会采用一些设计和实现的捷径，这在平时压力较小的环境下一般时不会用的。代码的 review 和单元测试也可能会比平时粗心一些，这些代码的状态变化也会比平时快很多。

为防止这种情况的出现，每日构造会坚持相关的规定，让压力下的项目保持在正轨上。代码仍然每天在不断变化，但是构造过程使得这种变化每天都可控。

谁能够从每日构造这种过程中得到好处呢？一些开发人员会抗议说，由于他们的项目太大，每天进行 build 是没有实际意义的。但是为什么现在最复杂的软件项目组却能够成功的执行每日构造的制度呢？本文首发时，Windows NT 包括了 560 万行代码、分布在 4 万个源程序文件中，项目组仍然可以坚持每日构造。

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=3609&fpage=1

[clear_jiang 专栏]

无规范的系统测试

作者：Sanford M.Sorkin
翻译：clear_jiang

无规范的测试对质量保证部门来说通常是一个主要的挑战，QA 部门无法意识到一个项目已经接近尾声，并且没有时间来处理未预料的事情。从涉及全部范围来看,这直接的反应是带来接下来提出的问题：

不能再来一次。

没有规范我怎样预设测试？

交付期是什么日子？

让我们开始。

不管这些回答如何，应用程序必须被测试。接下来我们可以处理以下这些问题：缺陷防止、初期质量评价中包含的位势值、以及这些是怎么产生的，除此之外，至今为止我们拥有一个需求去开展测试，对于这点来说可能是我们唯一拥有的需求。

没有QA 部门的预先计划，普通的测试过程必须以一些文档编制任务而先进行，而这些文档任务通常将是把工作转交给QA 组的必要条件。接下来的步骤是从收到将被测试的工作开始，通过编制文档这些步骤，然后到不同类型的测试上。

第一步：创建功能说明
第二步：排列工作量（写规范）
第三步：为管理供时间评估和测试覆盖率
第四步：回顾需求并开始管理项目
第五步：自动化测试决定
第六步：其他准备

方法和程序的变化对于每一个测试项目来说是会发生的，但是对于一个特殊的工作来说是极少的。因此，寻找共性来确定早先的测试材料和计划是否能被从

重新使用以便更容易的处理意外的工作。

第一步：创建功能说明

对于刚刚开始的测试要抵住诱惑。这一步骤充满潜在的危险。团队将是成功完成这个项目的关键，最好的第一步是排列这个项目，并且确定哪些东西是我们期望从测试过程中应得到的。这将理所当然，给应用程序创建说明书是必要的。很明显这个工作应该由某个人早先完成，并且现在需要非常努力的去做它，但是这才是第一步。

如果没有编写规范将导致怎样的结果？如果你依然在考虑这个问题的话，考虑如下内容：

对目标没有一个明确的说明。

不能确定项目的测试将持续多久。(尽管你已经得到交付产品的交付日期。)

不清楚谁应该被分配到这个项目中。

除了无完整的信息而进行测试这一所知的风险外，提供其他任何准确的风险评估变得十分的困难。

一部分或全部进行自动测试变得不可能。

通过一个没有文档说明的系统来解决长时间的维护和系统升级这些问题，将变得很复杂。

我们可能不知道测试什么时候完成。

如果被测试的应用软件是一个简单独立的系统，涉及到的列表则比较冗长 。但如果我们准备在客户端/服务器这样的环境中进行测试，这将变得更加复杂。要将测试这类应用软件的不充足性相关的风险做一个简短的说明呈交给管理者。同时也有必要提醒网络小组并且让他们参与测试系统的通讯方面。

测试客户端/服务器应用软件的其他复杂性将使测试计划的风险和意外事故部分是冗长的。（确认这一信息非常接近计划的实质，以便让它在检阅中不被忽略掉。）

“ 没有时间做计划，我晚了，我晚了。 ”

如果毫无疑问的要马上开始的话，至少你要努力回答以下问题：

项目的经营目标是什么？
客户是谁以及他们的期望是什么？
客户的经验水平是什么？
客户支持将愿意准备回答这些问题吗？
应用软件是在新的开发或维护环境中被测试？
谁着手与这个项目，以及全体的开发方法将遵从谁的。
这个系统必须什么时候投入使用？
什么是系统故障所含有的风险？（费用，信心，商机，等等）
什么是系统注重的主要功能？

这一系列潜在的问题是无止境的，因此要保持调查以便将问题最小化。一旦了解了主要功能，那么辅助功能将会开始显示，你将应该制作文档并且进行下一步。

简要说明一下，如果是一个客户端/服务器应用软件，必须采取特别的考虑以便让项目组中的所有其他潜在参与者的参与进来－尤其是网络小组，他们将负责系统的通讯方面。

第二步：排列工作量（写规范）

规范可以简单列出每一条系统需求的大纲形式来写。如果团队有一个标准的格式来写规范的话，应该使用这一格式。获得需求的方法如下：

与应用软件的交付测试人员进行交谈，尽管迫使这个人履行诺言可能是困难的。
运行程序并尽可能的检查编码。
会见客户。（尽量小心，告诉客户你不知道他们试图完成什么可能不一个很好的策略）
回顾编程的注释和相关的东西，尽可能找到更多的文档。

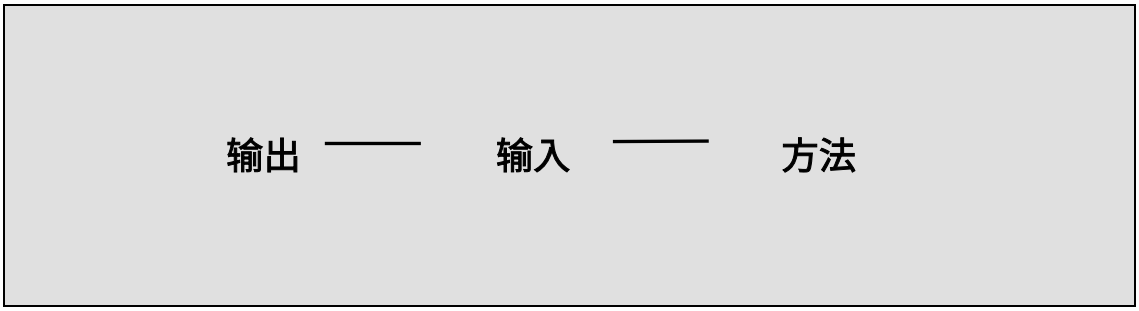
从内容中得到提示，细读帮助屏幕，寻找叙述信息。如果是一个维护项目，

设法确定系统中什么东西已经被更改或增加了什么，这样使你最初的大部分测试停留在某些方面。你仍然可能得不到所有的需求，但这可以明确可以开始了。

当你会见客户或与编程人员交谈时，他们趋向于叙述通过列举所有的输入，通过必需的可知处理步骤产生预期的输出，而得到这个系统意图是什么。一种更有效的会见技巧是要他们通过列举系统的所有期望输出而开始的。确定输出提供了一个系统实现目标的理性认识，并且开始了鉴定系统测试计划和验收测试计划内容的过程 。这些输出包括：

报告
屏幕/显示
消息/界面
文件
声音(音频信息)

下一步讨论系统的输入接着便是产生输出的必需处理步骤。记住当提供了理想的系统需求，没有说明书我们要逆向确定需求是什么？如果当测试计划得到评估，每个需求将得到确认。



图一：会见方法

一旦创建了整个系统的图示，则到了逐步展开评估的时候了。管理必须见多识广以保证能做出最好的决定。

排列工作量的产物将是一个文档，文档充分详细的列出了每条系统需求，这些细节能让其他小组回顾工作以及项目评提供了帮助。需求的定义将要完成是不大可能的，但是然而，现在它们是建立了功能说明和设计文档的基础，这让将来的任何升级变得便利。

接下来的样本文档包含了风险评估以及测试应用软件的相关费用。虽然在某些方面可能没有完成，它提供了一个机会给其他人在复阅时填写缺少的要素。

这个文档可能包括的部分：

功能需求

风险和费用

报告

显示

信息

这些是需求文档的最基本的要素，然而与十分有计划的方案相比则显得比较稀疏。

需求文档的例子：

功能需求：

这个文档里的功能需求是通过以下的途径产生的：会见客户，回顾系统产生的报告，读有关屏幕帮助的上下文信息，以及使用每个屏幕输入。

风险和费用：

并不是所以的输出都能被鉴别，没有资料被用来鉴别那些列出的所有可能的系统错误信息。合理的猜测将用于测试个别的领域。将要进行现场测试因为没有单元测试计划可以用来评估的。

报表：

按产品说明排列的清单
按 SKU 排列的清单
按日期排列的清单

屏显：

图标屏幕，标志
主菜单
输入清单
报告清单

信息：

输入清单	无效的 SKU
	没有找到的条目
报告清单	输出到屏幕，还是打印机？
	无效的日期输入

第三步：为管理提供时间评估和测试覆盖率

在排列工作量的基础上，现在就可以提供测试工作量的评估。利用团队的评估方法，现在应该可能形成测试时间和覆盖率的合理评估。如果团队没有一个标准的评估方法，考虑下面的指导方针：

每屏 10－15 系统测试案例，中到低风险情况

每屏 20－15 系统测试案例，高风险

每份报告1-2个测试用例过滤标准

每条消息1－2　测试案例。

样品测试评估

评估测试时间和覆盖率

	风险	测试评估
屏幕		
图标	低	2
主菜单	低	5
输入列表	高	25
报告清单	中等	10
消息		15
报告		20
综合		13
总共		90

测试计划－ @15/天， 90/15 =6天
测试执行－ @25/天， 90/25 =4天
调试&修复－再测试 @25 =1天
总共 ＝11天

第四步：评估需求并开始管理项目

项目的大约大小已经确定并且测试工作量的估计有了一个基本的评估。然而，在进行之前我们应该通过预排的评估结果，接着告知管理结果。如果可能，在这点上介绍一个项目管理工具，用来记录有时间表的计划好的测试。

项目的管理软件也将产生报告，这些报告可以用来显示在可利用的时间里有多少测试可以被完成。项目管理工具的另外一个好处是：在评估中，变化被不可避免的提出，这些有冲突的变化通过软件可以容易的被评定。

第五步：自动化测试决定

关于让测试自动化的决定将反映出部门的早先自动化经验和当今的专门技能。如果团队没有自动测试的经验，这时可能不是最好的时间去开始学习。这点的一个例外可能是不平常的情行，我们很惊奇一个非常大的项目，我们得出的测

试评估在1000小时的范围之内。

当处理一个新的项目时虽然并不建议人们学习自动测试工具，但如果你有这方面的经验， 应该考虑自动化测试。最有效的方法是让测试小组编写测试案例，然后将案例转交给自动化测试的专家。这些专家可能是2个或三个，他们的工作职责是为其他小组提供自动化测试。

如果决定使用自动测试，记得增加时间用来评估。另一个考虑的因素是，这个项目将会返工，可能会使被省略掉的很少使用的回归测试来作为一种补救措施。

第六步：其他准备

时间表提供了组织结构，但是他们不会在将来使用额外的时间来完成工作。回顾你的测试列表，如果可能的话重新安排，尽量不要遗漏任何事情。

文档记录每件事，即使你的工作只是一小会儿，每件事情必须有文档记录。当问题来临的时候，你将有足够的时间去调查问题。

从项目的管理软件中频繁产生的项目状态报告。

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=1371&fpage=2

[bambooman 专栏]

测试数据的选择

作者：未知
翻译：bambooman

在系统测试中，测试数据应该根据需求覆盖每一个参数的可能的取值。既然要测遍每一个值是不可能的,那么应该在每个定价类中选取几个值。一个定价类就是一个被认为是类似值的集合。

理想情况下，检测错误条件的测试用例应该与功能测试用粒分开来写，并且应该有校验错误信息和日志的步骤。实际上，如果错误测试用例还没有写出来，测试人员可以用执行功能测试用例来检测错误条件。如果可能,应该清楚那些测试数据能构触发错误。

等价类的例子:

字符串

- 空字符串
- 只包含一个空格的字符串
- 一字符串开头或／和结尾的字符串
- 语法上合法：短和长值
- 语法上合法：合法的语义和不合法的值
- 语法上不合法的值：非法字符或者组合
- 确定测试特殊字符，例如#, ", ', &, 和 <
- 确定测试可以从国际化键盘上输入的"外文"字符

数字

如果可能的话，是以下空的字符串

- 数字 0
- 在范围内的正数，小数（靠近零的数字）和大数
- 在范围内的负数，小数（靠近零的数字）和大数（绝对值）
- 超出范围的正数
- 超出范围的负数
- 零开头的数字
- 语法上不合法的输入，例如含有字母

标识符

- 空字符串
- 语法上合法的值
- 语法上合法的值: 重复索引，非法索引
- 语法上非法的值

Radio 选项

- 选中一项
- 如果可能，一项也不选

Checkbox 选项

- 选中项
- 未选项

下拉菜单

- 按顺序选择每一项

滚动菜单列表

- 如果可能，一项也不选
- 按顺序选择每一项
- 如果可能，选择多项
- 如果可能，选择所有项目

文件上传

- 空白
- 0 字节文件
- 大文件
- 短文件名
- 长文件名
- 如果可能，语法上非法的文件名，例如："File With Spaces.tar.gz" （带空格）

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=9848&fpage=2

原创读后感

[skinapi 专栏]

如何复现不可复现的 Bug

作者：skinapi

文章来自以下链接：

http://skinapi.cnblogs.com/archive/2005/08/07/209426.html

从标题来看大家可能会觉得晕，这里说到的不可复现是指这些 Bug 有时出现，有时候不出现。相信大家在测试过程中肯定遇到过这种 Bug，不少这种不可复现的 Bug 定位起来非常困难，可能很长时间都不能得到解决。能否复现这些不可复现的 Bug 成为大家关注的一个话题，最近国外的测试专家 James Bach、Jonathan Kohl 等对这个话题进行了一些探讨，这里把他们的一些思路理出来和大家分享。

要想复现不可复现的 Bug，需要先提到一个概念就是 ET（Exploring Test），也就是探索式测试，这种测试方法是由 James Bach 首先提出来的，在所掌握的被测对象的信息不是很充分的情况下，这是一种很有效的测试方法，如果大家感兴趣，我再整理一篇 ET 的文章出来。

在给大家阐述如何复现不可复现的 Bug 的思路之前，先说说为什么要复现这些不可复现的 Bug（废话两句^_^）。对于整个项目或者产品而言，如果这些不可复现的 Bug 是很严重的 Bug，比如导致系统崩溃等，如果不能及时、准确的定位和解决，最终发布出来的软件到达用户手中后，一旦出现势必会影响软件已经公司在用户心中的形象，严重的会“迫使”用户选择竞争对手的产品，这些显然都是公司所不愿看到的。而对于测试人员而言，出现了这些不可复现的 Bug，实际上是一次很好的锻炼和提高机会，如果只是提交缺陷报告将这个大皮球踢给开发人员，不仅丧失了一次提高测试水平的机会，还有可能破坏和开发人员之间的关系。

废话完了，进入正题。当出现不可复现的 Bug 时，大家可以从以下五个方面来进行考虑：

1、被测对象的版本信息

我测试的到底是哪个版本，这主要是有两个作用：一是确认我测试的是正式的软件版本，如果不是就先记录下该问题，然后选择正式的版本进行测试（开发人员基于尝试的一次非正规的修改可能会导致不可复现的 Bug）；二是可以和其它版本进行对比，如果其它的版本没有类似的问题，就可以去对比这两个版本之间的区别。

2、环境

这里的环境是指出现不可复现的 Bug 时所对应的测试环境等，比如测试所用的计算机，如果出现不可复现的 Bug，那我换一台机器是不是还会出现类似的问题，也就是说通过环境的改变来进一步搜集不可复现 Bug 的相关信息。

3、模式

这里的模式是指我对这个 Bug 如何出现的一个理解，先给这个 Bug 设定一个模式，比如是不是数据库通信中断，然后再进行测试，收集更多的信息去修改和完善这个模式，这样不断进行，最终直到 Bug 能完全复现为止，这个时候只要使用这个模式就可以复现出 Bug 了。

4、人

这里提到的人有两个含义：一是测试是由人来进行的，人的操作、人的思维方式会有不同，通过分析这些信息也有可能找到这些不可复现的 Bug 的蛛丝马迹；二是想复现不可复现的 Bug，往往需要多个人之间的相互协作，比如测试人员、开发人员等，通过大家的沟通和协作就能更容易去复现了。

5、测试工具

通过一些 debug 工具或者 log 工具等搜集内存等信息，根据这些信息来进行分析，找出不同信息之间的共同点，比如某一块内存始终都会被改写等，通过这种方式来去复现 Bug。

上面的五个方面都是和 ET 的思想紧密相关的，通过不断的测试和不断的信息收集和分析，逐步的把模糊的、不确定的测试变成清晰的、确定的测试，这样就能复现那些不能复现的 Bug 了。考虑信息时

可以从以上五个方面来进行考虑。

相应的文章链接：

http://www.kohl.ca/blog/archives/000115.html

http://blackbox.cs.fit.edu/blog/james/archives/000197.html

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=16523&fpage=1

LFM 和 POM，软件测试相关的两个模型

作者：skinapi

文章来自以下链接：

http://skinapi.cnblogs.com/archive/2005/08/09/211065.html

在micahel的blog（http://blogs.msdn.com/micahel/）上看到有两个和模型相关的概念：LFM和POM，在baidu和google上都查了一下，相关的链接很少，也不知道是不是micahel自个整出来的东西，由于这两个概念和测试还是有关系的，所以这里我按照自己的理解整理一下，供大家参考。^_^

先看 LFM，它是 Logical Functional Model 的缩写，翻译过来就是逻辑功能模型。具体来说就是根据用户的行为得到一个模型，然后根据这个模型进行测试用例的设计，这样设计出来的用例和用户的实际操作是非常吻合的，从而保证了测试的高效率。这种方式我在进行评审时使用过，感觉还是蛮有效果的。

再看 POM，它是 Physical Object Model 的缩写，翻译过来就是物理对象模型，这是和基于 UI 测试相关的一个概念。对于 UI 而言，它可以看成多个界面元素的组合，具体功能可以会保持不变，但功能对应的界面元素会发生变化，那考虑到这种情况该如何测试呢？这样就引入了 POM，将功能从具体的界面元素中抽象出来，这样设计出来的用例即使界面元素发生变化，用例也不需要变化。

以上是我对这两个概念的一个理解，不一定完全正确，还需要进一步在实践中进行尝试和总结，也欢迎大家提出自己的看法。抛开这

两个概念的名称不说，里面包含的思想还是蛮有意思的。^_^

对应版内链接为：

http://www.51testing.com/cgi-bin/viewthread.php?tid=16524&fpage=1

网络资源

http://www.bugbash.net/

介绍：主要特色是用一些漫画来阐述一些测试的问题，大家闲的时候可以看看。^_^

http://www.51testing.com/Jdwz.htm

介绍：收集大量国外软件测试网站，偏学术。

http://www.qaforums.com/

介绍：国外知名的软件测试论坛。

http://www.informit.com/articles/index.asp?st=41271&rl=1

介绍：有不少英文的软件测试文章。

http://www.softwareqatest.com/qatfaq1.html

介绍：对不少软件测试基本问题进行了解答。

http://www.workroom-productions.com/papers.html

介绍：上面有文章，有录音。

http://blog.csdn.net/imlogic

介绍：版友“ 我叫 kiki ”的软件测试翻译 blog，有不少原创翻译作品。

http://www.testingreflections.com/

介绍：国外一个专门收集测试文章的网站，可以看成软件测试的一个网摘网站，更新很及时，重点推荐。