VECTOR software

Vector Software

白皮书

如何开发高质量的软件

通过测试集中型的软件开发方法来提高软件质量

“构建技术系统所要做的工作是非常繁重的，同时还需要专业知识：语言和协议，编码与调试，测试与重构。”——James Garrett

概述

每年各个企业都在努力实现企业的主要目标。通常，这都是通过实现基于度量指标的性能目标（可能包括质量目标）以及利用最佳惯例规范商业流程来实现的。最后，要通过一些测试和报告形式来评估这些目标对企业的影响。在接受评估之前，机智的员工都会要一个评估单——但是在开发新的软件产品，确定测试目标时，他们常常会忽略这一点。

企业在设计实体产品时，会花很多时间设计产品的生成流程，以及对完成的产品进行自动化测试。大家都不希望设计出一款无法高效地进行生产和测试的产品。产品在完成设计之前，先要解决生产和测试的问题。如果所设计的产品成本太高，或者无法持续地投入生产，那么该设计就没有任何意义。

如果软件开发也使用相同的做法会怎么样？假设，我们在设计软件系统时就想了很多方法来尽量提高软件的“生产”和“测试”效率。那么，我们不仅能保证软件产品的质量，还能提供一个很好的流程来确保软件在整个产品生命周期中都有高质量的水准。

构建可测试的软件

所设计的软件产品必须具备“可测性”，这样的观念可能需要企业转变一下思维模式。最初在进行设计和原型制作时，基本注重的都是功能和性能问题。虽然这些方面也很重要，但是如果生产出来的应用程序质量很差，存在很多难以维护的 bug，那么产品功能和性能也就无从谈起了。

有一些“简单的”方法可以用来构建 “可测性”较高的软件产品，如：确保需求的完整性和正确性，在设计应用程序时要把软件测试放在心上，采用易于理解的编码方式，“真正”

执行代码评审程序。

细读软件需求

确保软件需求的完整性可以防止很多缺陷的出现。我们可以考虑一下怎么为“平方根”函数写一个说明。实际上，这是深藏在数学函数库中的一个简单的函数，但是如果条件写得不够完善，可能也会把“简单的函数”执行得很糟糕。

看看这个说明：
square_root()函数可以返回输入的所有有效值的平方根。

更好的说明是这样的：
square_root()函数可以返回输入的所有有效值的平方根，如果输入的是无效值，就会返回 0。有效输入包括正的 32 位浮点数，0，正无穷。无效输入是指：负数，负无穷，非数值。

根据这个说明，设计师可以轻松地构建一套低层级的需求和用于验证该函数能否正确执行的的测试用例。

分离控制和数据需求

在设计系统时，如果能将控制和数据处理清楚地分离开，可以简化设计，从而帮助提高测试效率。下图展示了如何它们是如何分离开的：

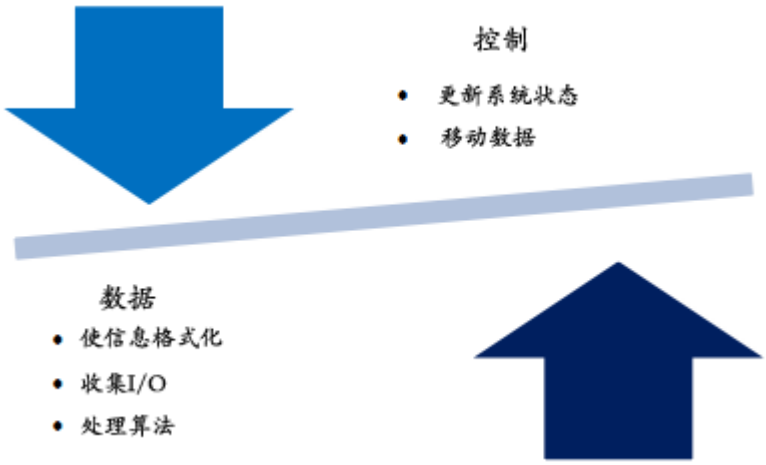

图 1：将控制和数据处理清楚地分离开可以帮助提高测试效率

清楚地将二者分开，让设计师和测试人员能够专注于输入的特定类型，运行情况和结果。此外，精简的设计能够提高软件的重复使用率，降低维护成本。

注意编码风格和架构

要提高“可测性”，首先要做的是确保代码易于理解，灵活性高。在软件开发的早期，如果开发工程师开发的软件“体积小，运行快”，那么他会非常受器重。因为软件系统的内存和

CPU 容量是很有限的，工程师一直要努力在每个应用程序中添加更多的功能。而且，以前的应用程序的代码库比现在的小很多，当语言和硬件设备发生改变时，工程师常常会将代码全部重写一遍。

如今，情况和以前大不相同了。虽然现在应用程序的组件在大小和时间方面还是有限制，但是总的来说，应用程序的大小和生命周期都大大地扩大和延长了。所以，现在构建代码时，如果代码易于理解和维护，那将是一个很大的优势。幸好，使代码易于理解可以大大提高“可测性”。

咨询公司 Virtual Solutions 的 Bob Gary 有一句经典言论“用 C 或 C++编写代码就像是在没有任何防护措施的情况下使用电锯”（引用自 Byte (1998) Vol. 23, Nr 1-4 p.40）。下面有一些在编码风格方面有助于提高“可测性”的“安全防护措施”。

图 2：利用这些窍门写出易于测试的代码

执行有重要意义的代码评审流程

如果我们问开发工程师什么样的代码算是设计良好的代码，可能每个人的回答都会不一样。执行具有重要意义的代码评审流程是一个很好的方法，可以判定代码对于其他人而言是否易懂。另外，团队会有很多自己的“团队知识”，在代码评审过程中可以运用这些知识。

提高软件质量的最佳做法

创造**高品质文化**的最佳方法就是为每个团队成员确立一个工作流程和度量标准，来解决每个部分的质量问题。比如，如果开发工程师可以在没有任何限制的情况对代码库进行修改，那么我们的代码很快就会漏洞百出。但是，如果开发工程师受到的限制太多，那么就很难发布出一个新版本。

所以我们需要一个可操作、可重复的流程。比如，开发工程师在配置管理器中对代码进行修改之前，要遵循以下规范：

图 3:　备选的软件变更流程

如果说我们的企业不了解这些流程，或者说我们的代码库已经超过 20 年了，却没有一个完整的测试架构，那么我们就需要创建一个适用于这种环境的流程了。没有人会建议创建测试用例使已部署的应用程序的覆盖率达到 100%，但是，可能会要求所有的“新”代码必须执行正确性和完整性测试。有一件事情很清楚：如果持有“我们的代码库太复杂，我们没有办法改善”这样的态度是无法提高软件质量的。

在我们向***质量文化***转型的时候，可能会受到来自开发工程师的阻力，他们认为这样做会有很多新“门槛”，耗费很多时间，而且会增加很多“额外的工作”。我们可以通过对工程师进行指导和培训来实现平稳转型。组织一群资深的团队成员来领导这场转型，以及通过供应商咨询人员帮助制定最佳做法，这两种做法成本比较低，也可以加快该转型的接受速度。当开发工程师看到他们能够更快地通过 QA 环节，并能够更快地为客户部署自己的软件时，他们才

会对这一转型真正买账。

让团队看到切实的结果是非常重要的，所以在整个开发流程中的各个阶段发现并修复 bug 之后，发布度量标准将加强内部提高质量的觉悟。经过一段时间之后，在开发阶段找到的 bug 就会增多。如下面的示例所示，经过一段时间后，Bug 的数量会越来越集中在“左”侧条形柱上。

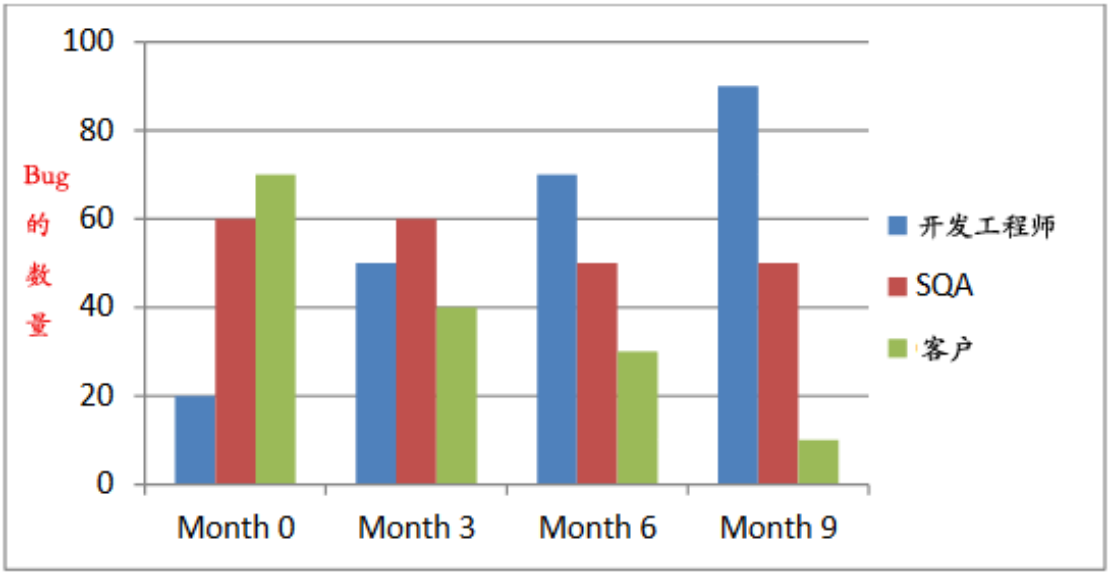

图 4：在产品交付前修复 bug 是交付高质量应用程序的最有效的方法

QA 工程师发现的 bug 越少，开发工程师就有越多的时间研究新功能，QA 工程师也会有更多的时间来进行深入测试——这样客户会遇到的 bug 就大大减少了。

执行自动化测试

要通过软件测试来提高应用程序的质量就必须保证测试的全面性。同时要求必须能够轻松、快速地完成软件测试，以保证所有的开发人员在任何时候都可以对各个版本的代码执行测试。

虽然每个企业都开发了软件构建系统，以便进行自动化的增量式应用程序构建。但是这些系统大多数都没有采用可重复使用的的增量式测试架构。通常都需要手动操作来定期执行测试，而不能够完全自动化地持续进行增量构建。

在很多团队里，所有开发工程师都采用他们自己的测试方法，而没有一个统一的组织平台来执行自动化测试。因而，从引进 bug 到发现 bug 中间要经过很长时间。中间的这段时间越长，就越难发现引发该 bug 的原因。理想情况下，每次在对软件做了修改之后，都将该修改可能影响到的测试执行一遍，然后再执行该修改。

所以，我们该怎样判断在代码发生了改变之后，需要执行哪些测试呢？基于变更的测试架构可以自动确定每次代码变更所影响到的测试子集，我们只需要执行这些测试就可以了。能够实现上述这一步的关键在于可以快速、自动化地实现以下步骤：

1. 确定代码更改所影响到的测试子集
2. 在代码发生修改之后执行这些测试
3. 报告执行失败的测试并更新代码覆盖率

采用这种方法只需要花几分钟就能完成编码和测试，而不需要花几天的时间。

图 5：采用一个可持续的测试流程能够加快产品投放市场的速度并确保所交付的应用程序的质量

开发所需要的工具

如果我们想要创造**质量文化**,必须用到以下软件工具：

- 静态分析工具：找出源代码中经常产生 bug 的代码模式
- 动态测试工具：通过黑盒测试和白盒测试，保证应用程序的正确性
- 覆盖率分析工具：评估所有测试活动的完整性
- 自动化测试工具：让团队所有的成员都能够在代码发生改变时对其执行测试

商业优势

提高软件质量和测试的完整性是企业的两大主要目标,但是在如今这个竞争激烈的经济环境下，肯定要有充分的商业理由，企业才会做出改变。下面是几大显而易见的理由：

提高软件的可测性：

- 降低开发成本
  - ✧ 可以让低成本的初级开发工程师来设计和执行测试
  - ✧ 可以减少每次代码变更所影响到的测试量
- 降低生命周期成本
  - ✧ 提高了软件的重复使用率
  - ✧ 降低了维护成本

提高软件质量：

- 集成时代码存在的 bug 减少了，因而缩短了集成所需的时间
- 集成所需的时间缩短意味着产品的发布周期也缩短了
- 客户遇到的 bug 减少了，提高了客户的满意度
- 客户满意了就会提高客户对品牌的忠诚度，从而增加公司的收入

总结

开发高质量的软件是一项艰巨的任务。如果设计时就考虑到每个开发阶段的测试问题，将有助于团队加快应用程序的交付速度，并且能够确保产品的质量。测试工作进行得越早越有利于保证产品的质量。保持谦逊的态度也十分重要——实际上，虽然编码很重要，但是并没有开发人员想象的那么重要。最后，建立测试集中型的流程，并在好用的软件测试工具上进行投资能够帮助企业解决软件的质量问题。