73 | 软件质量管理：单元测试、持续构建与发布

2020-01-14 许式伟

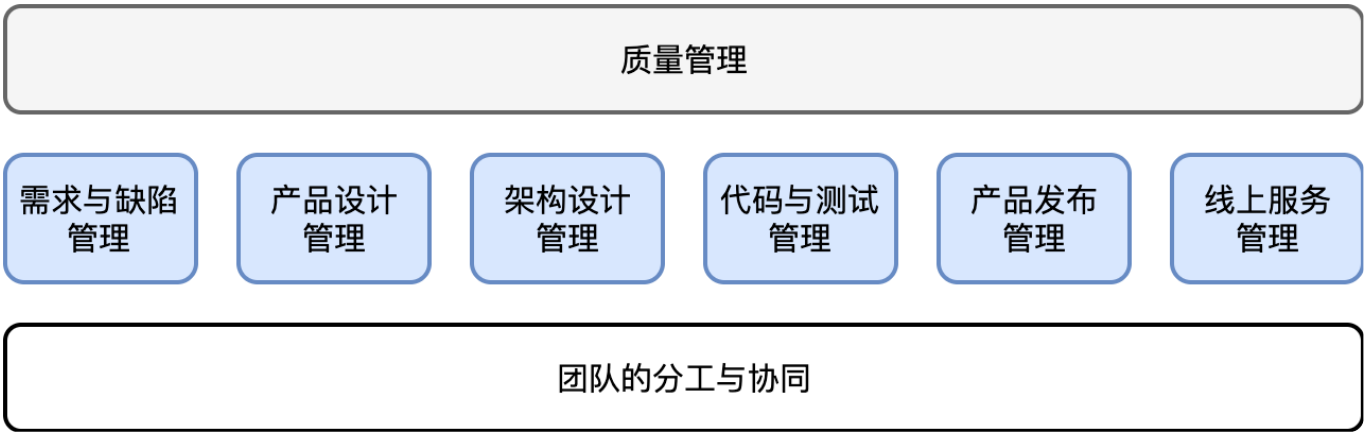
讲述：姚迪迈
时长 07:57 大小 7.29M

你好，我是七牛云许式伟。

上一讲 “72 | 发布单元与版本管理” 我们聊了版本管理中，只读思想给软件工程带来的确定性价值，它在软件工程质量管理中也是很核心的一点。

软件质量管理

今天我们聊聊软件工程中，我们在质量管理上其他方面的一些思考。事实上，软件质量管理横跨了整个软件工程完整的生命周期。

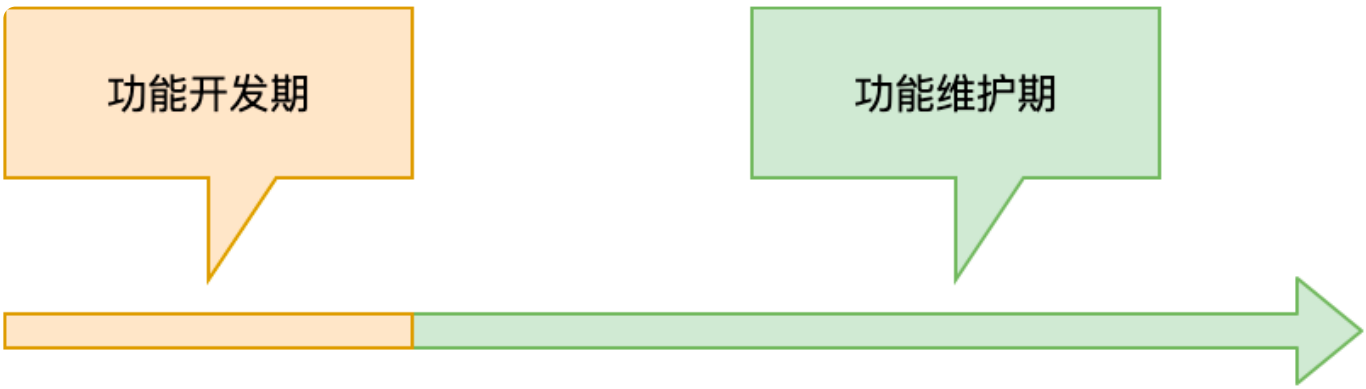


软件工程与传统工程非常不同。它快速变化，充满不确定性。不仅如此，一个软件工程往往是生命周期以数年甚至数十年计的工程。对于传统工程，我们往往把一个工程同时也称之为项目，项目工程。但软件工程不同，虽然我们平常也有项目的概念，但软件工程并不是一个项目，而是无数个项目。每个项目只是软件工程中的一个里程碑（Milestone）。

这些都决定了软件工程质量管理的思想与传统工程截然不同。在传统工程中，设计的工作往往占比极少，重复性的工作占据其生命周期的绝大部分时间。所以传统工程有极大的确定性。检查清单（Check List）很可能就已经可以很好地实现其工程质量的管理。

但对于软件工程来说，设计工作在整个工程中持续发生。哪怕是非设计工作，比如编码实现，也仍然依赖个体的创造力，同样存在较强的不确定性。显然，检查清单（Check List）完全无法满足软件工程的质量管理需要。

那么，到底应该怎么管理软件工程的质量？每次谈软件工程质量保障的时候，我总会先画下面这张图：



它谈的是软件的生命周期，或者也可以理解为软件中某项功能的生命周期。我们把软件或软件的某项功能生命周期分成两个大的阶段，一个阶段是开发期，一个阶段是维护期。开发期

与维护期是相对而言的，只是在表征上，开发期有更强的设计属性。维护期虽然也持续会有设计工作，但是工作量会小一个数量级以上。

为什么划分出开发期与维护期是重要的？

因为开发期的时间跨度虽然可能不长，但是它的影响太大了，基本决定了后期维护期的成本有多高。

这也意味着软件工程是需要有极强预见性的工程。我们在开发期恰如其分地多投入一分精力，后面在维护期就有十倍甚至百倍以上的回报。

设计工作的质量至关重要。但是它执行上又不太有复制性，可复制的只是设计范式和设计思维。

我们只能在这种执行的不确定性中找工程上的确定性。

如何做到？

单元测试

首先，做好自动化测试。自动化测试对软件工程的重要性是不言而喻的。如果是一项一次性的工程，我们可以基于常规的手工测试。但常规测试的缺点在于：

其一，一般常规测试是基于手工的，不具备可回归性。因此，常规测试的效率不高，一次完整的测试集跑下来可能需要几天甚至一周之久。

其二，易于缺乏效率，所以往往为了赶工会导致测试仅仅针对典型数据，测试的覆盖率往往也很低。

软件工程的生命周期往往几年甚至几十年之久，我们必然关注单次测试的效率。所以自动化测试的核心价值就在于可回归性与提高测试的覆盖率。

自动化测试与常规测试相比，风格上有很明显的不一样，它有如下重要特征。

自动化、可回归性。

静默（Quiet）。没有发生错误的时候，就不说话。

案例执行的安全受控。某个案例执行的失败，不会影响其他案例的正常运行。

从分类来说，一般自动化测试我们分两个层次：一个是模块级的单元测试，一个是系统级的集成测试。

无论从什么角度来看，模块的单元测试都是重中之重的大事。原因是，单元测试的成本是最低的。

关于测试成本，我们可以从两个维度看。

其一，单元测试的实施成本低，最容易去做。不少高级语言比如 Go 语言甚至在语言内建的工具链上就直接支持。而集成测试虽然也有自动化的方法和支持工具，但是往往需要更高额的代价。

其二，减少问题发现的周期，进而降低问题的修复成本。单元测试将问题发现周期缩短，基本上在问题现场就发现问题，这降低了 Bug 的修复成本。如果问题在系统的集成测试阶段发现，那么从问题定位，到回忆当初实现这段代码时候的思路，到最终去解决掉它，必然需要多花费几倍甚至几十倍的时间。

因此，我们鼓励更严格的单元测试要求，更高的单元测试覆盖率，以尽可能把发现问题做到前头。

但仍然有不少公司在推广单元测试上遇到了不小的麻烦，推不起来。

对于这一点，我们认为首先要改变的是对推广单元测试这件事情的认知。我们不把推广单元测试看作是让大家去多做一件额外的事情，而是规范大家做单元测试的方法。

为什么这么说？因为实际上单元测试大家都会去做，很少有人会不经验证就直接交付。但是验证方式上可能有各种 “土” 方法，比如用 print，用可视化的界面做输入测试，用调试工具做单步跟踪等等。

但是这些方法代价其实一样不低，但是却不可回归，正确与否还需要人脑临时去判断。

更重要的是，这些方法最大的问题是没有办法去固化已知的 Bug，最大程度保留下来我们的测试案例。

这其实才是最核心的一个认知问题：我们应当重视我们的测试代码，它同样也是我们的开发成果，理应获得和模块的功能代码同等重要的地位，理应被保留下来。

解决了这个认知上的共识问题，自动化测试就能够被很好地推动起来。当前这方面的工具链已经非常完善，不至于会在工具上遇到太大的障碍。

持续构建，持续发布

其次，我们降低软件工程不确定性的方法是：持续构建，持续发布。

我们鼓励更小的发布。我们鼓励更短的发布周期，更高的发布频率。这能够让发布的负担降低到最低。

这种极度高频交付的机制与传统工程的质量管理机制迥异。但是它被证明是应对软件工程不确定性的最佳方式。为什么会这样？

其一，交付的功能越少，因为错误而发生回滚的代价越低，影响面越小。如果我们同时发布了数十个功能，却因为某一个功能不达标而影响整体交付，这其实是降低了软件的功能交付效率。更好的方式显然是把这个出问题的功能回滚，把其他所有功能都放行。

其二，交付频率越高，我们对交付过程的训练越频繁，过程的熟练度越高，执行效率也越高。当交付成为一个自然习惯后，我们会把交付看作功能开发的一部分，而不是以前大家对研发的理解，认为做完功能就完事，后续上不上线与我无关。我们会鼓励更多把研发的绩效与功能线上的表现关联起来，面向客户价值，而非仅仅面向功能开发。

当然这种极度高频交付的机制，意味着它对软件工程的系统化建设有更高的要求。

当然，除了日构建与发布平台外，我们也需要在其中加入各种质量管理的抓手。比如：

自动化运行单元测试案例（unit test）；

单元测试覆盖率检查（code coverage）；

静态代码质量检查（lint）；

人工的代码互审（code review）；

灰度发布（gray release)；

A/B 测试（A/B testing）；

……

结语

今天我们更加完整地探讨了软件工程的质量管理。整体来说，软件工程与传统工程在质量管理上的理念是迥异的，甚至往往是反其道而行之的。究其原因，还是因为软件工程的核心在于如何在高度的不确定性中找到确定性。

如果你对今天的内容有什么思考与解读，欢迎给我留言，我们一起讨论。下一讲我们谈谈“开源、云服务与外包管理”。

如果你觉得有所收获，也欢迎把文章分享给你的朋友。感谢你的收听，我们下期再见。

上一篇　72 | 发布单元与版本管理

下一篇　74 | 开源、云服务与外包管理

精选留言 (5)

写留言

Aaron Cheung
2020-01-14

我们不把推广单元测试看作是让大家去多做一件额外的事情，而是规范大家做单元测试的方法。 感觉知道的有单元测试的项目都没有四成……

展开

3

Geek007
2020-01-16

相比单元测试，似乎code review在我司更难推动。code review需要人的投入，而且往往需要有经验的工程师，有经验的工程师有他们自己的开发任务，他们会把code review当成是额外的工作，动力不够，责任心也不够，code review的质量因而也不能控制。有时候迫于发布的压力，code review做的很粗糙。code review 必要么？许老师对这个问题有何看法？谢谢。

展开

作者回复: code review 非常重要，我们不 code review 就没法合并 pull request。

2

leslie
2020-01-14

最近就碰到一个现状在“持续构建，持续发布“方面的问题：公司业务最近刚好进入了高速增长期，每次交付的功能很多且目前上传频率基本上已经是每天的业务的低谷-凌晨那一段时间；如何权衡这种现状？按照老师前面课程提及的方式只能是每次只上传核心/最重要的更新功能，不知道老师是否有更好的建议。谢谢老师的分享，期待老师后面章节的更新。，

展开

作者回复: 如果发布很频繁，需要在发布系统和架构上优化，保证发布出问题可以一键回滚，另外升级要做到不影响用户，这样发布随时都可以，而不是只能在业务低谷。

1

Bachue Zhou

2020-01-21

大部分服务器项目或库项目一般都有单元测试，但涉及到图形化界面的项目可能就非常不乐观了，纯计算的代码太少也够简单不需要做测试，涉及到 UI 交互的代码量很大却不知道如何测试。最后还是要靠手测。

作者回复: ui测试的确复杂很多，但也不是不可能

我在你的视线里

2020-01-14

持续构建和持续发布还有一种小步快跑的感觉。但是不是会影响客户体验。

作者回复: 是的

2