测试计划

最佳实践

文档版本	02
发布日期	2023-04-23

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 DevOps 敏捷测试之道

本文介绍企业在敏捷和DevOps的逐步转型过程中，测试如何应对挑战，如何有的放矢进行测试，建立适合产品自身发展阶段、产品特点的敏捷测试能力。

敏捷和 DevOps

敏捷和DevOps转型始终是被业务目标和客户需求驱动的。市场竞争环境越来越激烈，新商业模式的创新和变现时间窗口越来越短，催生更多的企业采取精益创业的方式，捕捉市场需求后，尽量缩短TTM产品面世时间，快速推出MVP产品并快速响应客户需求迭代产品。

以华为为例，在2008年左右的时候，华为的项目还是采用传统的交付方式。例如，在年初开始一个项目，在项目立项之初就会把客户的需求全部收集好，包括一些用户的反馈，并把需求做了全年的排序。年中的时候发布产品给用户，两个月之后再出一个补丁，最终年底出一个正式的版本。当时版本交付的节奏还是比较慢的，但是对质量要求比较强。因为产品发布给客户以后，下一个补丁需要两个月，如果用户在这个期间发现产品问题，他们只能再等两个月，而在这期间如果用户不接受我们的产品，会导致项目前功尽弃，所以对产品的质量有严格的要求。

产品逐渐向敏捷方向发展，这时有一部分研发工具平台已经陆续转到上云，一些测试类的工具也需要转型。之前产品的交付是半年、两个月发一次，转型之后变成一个月，甚至两周发一次，但这时的转变并不彻底，与客户的交付过程仍然存在一些问题。后来越来越多的工具向平台化、服务化方向转型，这个时候一些商业模式发生了根本性的变化，也就是说当需求上云了以后，用户更加快速的介入进来。基于云平台，把一些功能快速的开发出来，然后频繁的和用户去商量，听取客户意见，牵引产品做快速迭代，这种交付方法使得交付周期一下变快了，之前是半年交付一次，现在是一周、两周，更有甚者，可能一两天就把功能发布出去了。从需求的角度来说发生了巨大变化，基本做到了小步快跑，快速试错。

测试债务

从瀑布到敏捷再到DevOps，在开发和测试生产率和需求交付效率提升的过程中，不同的组织或多或少面临一些积压问题没有解决，影响测试能力和测试价值的持续提升。

- 从对测试的重视程度上看，有的公司存在重开发、轻测试的情况，测试人员职业发展受限；手工测试人员不熟悉编程，开发人员对测试重视不足；测试工作量高，但人员配比低。
- 从部门组织和流程和文化上看，测试人员对需求理解不足，测试和开发之间的部门墙导致信息不透明、沟通协作滞后和不足，质量向速度过分妥协，以及忽视敏捷文化和价值观的培养塑造。

- 从测试和产品技术和方法上看，产品耦合度高、可测试性差，测试过于依赖黑盒功能测试，测试策略、方法不恰当，测试环境部署时间长，频繁升级等。

测试的焦点：业务价值的质量

测试首先是一个质量活动，做测试就是要保证质量；其次是一个工程性的活动，即在有限的时间、人力、资源投入内获得尽可能大的产出价值。质量有多个维度，需要有一个焦点：业务价值的质量，也就是产品“对客户呈现的价值”的质量。测试围绕业务价值去做，确定质量在功能、安全性、性能、易用性、兼容性等多个维度上的权重和优先级，而不是说一个测试上来之后，就把测试相关的关系点、关联点全部做测试。

让我们来看几个例子：例如现在正在做一个线上支付的功能，对这个功能最关心的方面肯定是安全，所以相关的测试用例关键点就应该围绕安全大做文章，一定把安全保证好；再比如，现在要做一个线上商城，面向用户是老百姓，不仅要让年轻人会用，也要让老人都会用，那么就要关注易用性；除此之外，电商举办大规模抢购促销活动，那就还需要关注性能。因此，测试要求瞄准产品本身的业务价值，确定产品的目标，相应的制定质量关键点，制订相关的测试策略，然后实践落地。落地之后还要基于一些不良的效果不断的进行反馈、循环，校验整体的测试过程是否达到预期结果，这就是我们的测试焦点。

常规安全与弹性安全

在我们常规的设想中，通常是哪个地方不安全，就一定要把所有不安全的因素找出来、清除掉。这是常规的做法，但却偏向于理想，在实际工作中是不可能把整个系统中不安全的因子全部识别到的，这其中涉及能力、架构等各方面的原因。

因此，在此基础上演变出了弹性安全，即通过场景模拟的方式将不安全因素尽量展现出来，从而基于这种不安全场景，给出快速的修复方案弥补这个不安全因素，从用户角度来讲是感知不到的。从产品来讲，它的商业目的和质量目的都可以达到，这就是所谓的弹性安全，即便发生了错误，能够及时快速的修复漏洞或者自我修复，达到正常工作的目的。

测试左移和测试右移

左移就是前移，尽量把活动向前移。例如BDD（Behavior Driven Development，行为驱动开发），基于场景直接设计出符合这个场景的用例，来匹配这个设计；契约测试，服务和服务本身之间有耦合，我们可以通过契约测试解耦，以防导致问题。

测试右移是指要把测试活动的覆盖范围尽量向后蔓延。通常的测试只进行到了版本发布之前，测好之后发布一个软件包，而测试右移要把软件包发布到生产环境，以及到线上运营环节，都要去做测试。

在这两个方面也有一些相应的实践，例如线上拨测，主动线上监控用户的一些行为，并从行为轨迹里面快速捕捉相应的问题，主动推送给相关的责任人，让他去关注并且解决。线上的过程可以通过一些测试手段，不断的反馈给真正的开发人员，让他知道当前产品的整体表现，开发人员就会快速的针对产品作出应对方案。

产品发展不同时期的测试策略

是否在团队组建之初，就要把整个自动化测试的能力构建起来呢？其实这有一个过程，下面从软件的成熟周期的角度，看一下如何构建测试自动化的能力。

在软件初期探索阶段，产品是一个不确定的状态，从前端的风格和整体的布局到后端的API都时刻在变化当中，而且变化比较频繁，由于自动化用例的生命周期比较短，所

以在这个时候创建一些自动化测试用例是不太划算的。而这个时间段的产品，往往特性是可控制的，只有几个测试，因此可以以手动为主，不考虑自动化，让产品能够快速识别错误点，让用户能用起来。

到了产品扩张阶段，用户认可产品，这时候会出现两个现象：第一是用户量增长，第二是需求数量增长。这时候必须要考虑自动化，因为在这个阶段每一次迭代的全量验证成本会越来越大，而交付的速度也会越来越快。我们不可能每一轮上线的时候都全部用手工做测试，这时候旧的模块就需要自动化用例去保证。

到产品提取阶段，产品已经到了需求的饱和期，产品的利益增长也到了饱和期，这时候要严格控制产品需求，自动化用例的职责变成守护，不允许变动引入额外的风险点、大的特性变动，导致对成熟的用户造成攻击。

团队规模对测试建设的影响

当团队规模在5个人以下，团队处于探索阶段，这时质量活动可以仅仅局限于测试的自组织阶段，只是做一些基础类测试管理活动，把缺陷管理起来，做一些回归测试。在这个阶段主要是建立一个测试管理的流程和机制，并没有接触到自动化测试。

随着项目的进一步扩大，逐渐增长到5-10人的团队规模，这时测试工作量突然增加，可能会有专门的测试人员进来，这个测试人员会去和开发人员进行串联，把需求转化成自动化测试的用例，搭建持续集成，逐步演进一些测试手段。——这个阶段已经开始做一些自动化的尝试。

团队进一步增大，一个人可能搞不定工作量的时候，会招聘更多的测试人员，成立专门的测试团队，这个团队就从自动化测试转向测试自动化，把更多的管理工作做进来。在这个管理过程中，我们会做一些产品的对接，包括开发专门的工具，实现自动化的整体能力，不仅仅是自动化执行了。

经过上面几个演进周期之后，测试团队具备了很多的测试自动化经验，这个时候可以进行面向云化的转型，现在很多团队都在进行DevOps转型，最关心的方面就是组建DevOps的全功能团队。那么之前转型的这些人在做什么？原有10-15人的测试专项团队做什么？在这个阶段团队要把测试专项能力向服务化能力转型：测试专员会在团队创建初期进行赋能，包括测试工程搭建、早期的测试用例怎么写、标准化模板的编制、针对非功能性测试的专项能力的赋能；所有团队进行测试流程的评审，包括测试策略、测试计划、测试用例的评审，再看一下整个团队里面流程上还有哪些改进的；从各个方面整个专项测试团队向服务化进行转型，帮助所有团队完成自动化转型。

自动化测试和测试自动化

这里要澄清一个概念，就是测试自动化（ Test Automation ）。

测试自动化的目的是减少手工测试和手工操作。测试自动化不仅仅包括自动化测试执行（ Automated Testing ），还包括其它所有可以减人力投入的活动，例如自动化创建测试环境、自动化部署被测系统、自动化监控、自动化数据分析等。很多自动化测试只是测试的执行部分，例如把一些测试执行的人工测试手段做成自动化测试。但是测试自动化不仅仅是只是执行，还包括了从环境的获取到生成测试数据、执行自动化测试，最终生成结果。如果有问题，会自动推送给相关的人，对应的组织解决。自动生成测试报告，测试人员直接拿到测试结果。

2 有的放矢制定测试计划

本文介绍如何确定测试目的、划定测试范围、制订测试策略、组建测试团队、准备测试工具和环境、制订测试计划。

凡事预则立，不预则废。在团队开展测试活动之初，需要制定相应的测试计划，指导整个测试周期中测试人员的测试活动。测试计划描述了测试目的、测试对象、测试范围、测试策略、测试活动、测试方法、测试资源和进度等，确定测试项、被测特性、测试任务、谁执行任务、各种可能的风险。测试计划可以有效预防计划的风险，保障计划的顺利实施。

为什么要制定测试计划

- 确保测试活动围绕测试目的开展工作，服务于明确的测试目标。
- 确定测试对象的被测特性和需求清单，框定测试范围。
- 选取适合于团队技术能力和工具组合的测试策略和方法。
- 尽早识别测试活动开展中可能面临的风险因素，并及时解决。
- 合理预估测试工作量和人员、资源需求，编制测试项目计划。
- 帮助测试人员分解测试活动和任务，编排个人工作计划。
- 指导测试执行活动、及时纠正和补救执行偏差。
- 作为相关文档，与利益干系人汇报沟通。

什么时间做测试计划

测试活动包含测试计划、测试设计、测试执行等。一般而言，测试计划排在首位，在测试周期的早期阶段开展。实际上根据不同的项目使用的开发模式、团队组织形式等，测试计划可能在多个时间节点中开展，例如：

- 某企业面向客户的移动APP产品，在第一个版本发布前，测试部门在接收到产品的测试申请后，接收测试需求做上线前系统测试计划。
- 某内部软件项目，产品立项完成需求确认后，架构师已经设计出产品架构方案和设计方案，产品未进入实质开发阶段，开发组织制定开发计划，测试组长制定测试计划。
- 某对外商用产品，例行提交做专项安全测试，安全测试部门根据历史安全测试计划和测试报告，评估新增产品特性变更，制定新版本的安全专项测试计划。
- 某软件测试团队有明确的功能测试、性能测试、安全测试等分工，测试团队主管制定主测试计划后，各类型测试专家制定类型的测试计划。

整体来讲，建议尽早开始测试计划相关工作，因为测试计划制定时间越早，越容易在早期阶段指导、规范产品的质量活动，提高产品的可测试性，测试人员从攻击者的视角指导产品架构和设计中的质量要素，符合测试左移的思想。

但越在产品的早期阶段，测试计划的粒度越粗，缺少细节可执行层面的测试用例、可供使用的测试环境等，需要随着项目的推进不断细化。测试计划不是一成不变的，随着测试项目的开展，测试计划逐步详细，包含越来越多的信息。测试计划的细化和完善过程中需要注意审视初始的测试目的、测试范围和测试设计和策略等。

谁来制定测试计划

- 测试项目、测试团队负责人。
- 安全、性能、可靠性等专项测试负责人。
- 经验丰富的测试工程师、测试架构师。

测试计划需要包含哪些内容

根据ISO、IEEE等有关于测试文档的相关标准，在项目实践中可以选择在测试计划中包含哪些内容，内容多少与项目、团队规模相关，小团队测试可以精简测试计划。

- 测试目的

  概述为什么要做这个测试，需要实现什么样的测试目标。测试目的是测试计划的源头，测试需要聚焦产品的业务价值。如果是产品整体测试计划，需要结合产品的业务属性，将质量的功能、安全、性能、易用性、兼容性、扩展性等维度融入测试目的，例如金融类产品对安全性要求很高。

- 测试范围

  详述被测系统（测试对象）的名称、版本、特性、需求、环境、测试项，明确需要测试什么、不测试什么。

- 测试策略

  明确测试类型、测试场景、测试方法，策略性说明如何测试。

- 测试方案

  详述测试使用的方案，例如集成步骤和顺序、测试步骤和顺序、测试方法、测试工具、测试用例设计和执行方法等。

- 测试环境

  描述测试所需要的硬件、软件、测试工具的名称、规格、数量、版本、帐号等信息，以及测试环境的准备、预定、还原、释放等管理策略。

- 测试人员

  详述测试人员数目、分工、职责，如测试架构师、测试开发工程师、性能测试工程师、测试环境管理人员等。

- 测试进度计划

  说明测试的计划开始和结束时间，测试总体进度安排，关键的阶段性进度检查点。测试进度计划结合开发计划，需要综合考虑测试方案、环境、人员等资源和任务的约束依赖关系。

- 测试准入条件

  明确接纳启动测试的准入条件，如产品规格说明书中的功能已经实现、基本流程和准入测试用例通过等，以避免由于测试基础条件不具备影响测试计划实施。

- 测试发布标准和交付件

明确测试完成需要满足的条件、测试通过/不通过的标准、测试完成后需要产出的交付件，例如测试报告，说明测试报告需要包含的内容。

- 风险

  分析当前项目运作中可能存在的风险，以及应对风险的规避、解决等措施。风险举例：人力到位风险、人员技能和领域知识风险、开发转测试时间风险。

测试计划评审

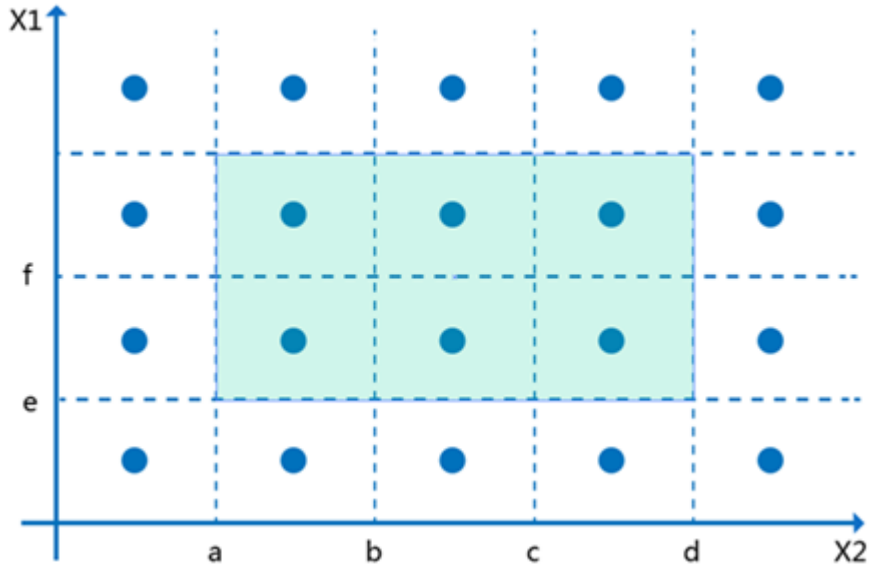
测试计划编写完成后，邀请关键干系人，如项目经理，测试经理，产品经理，架构师，运维经理等对测试计划的正确性、全面性以及可行性等进行评审。

3 典型测试设计方法介绍

测试设计是测试活动中起到承前启后作用的重要环节，根据测试计划分析测试对象、测试场景、测试类型、测试环境等，根据测试策略选取合适的测试方法和测试技术，设计测试用例。测试设计有场景分析法、等价类划分、边界值分析、因果图、判定表、正交法等方法。灵活运用测试设计方法可以帮助减少测试用例冗余，提高测试覆盖率、用例可维护性、用例复用程度，减轻无效的测试执行工作量，改进测试效果。

等价类划分

把系统输入数据划分为若干等价类，在每个等价类内部，选择所有输入数据进行测试与只选择其中一个输入数据进行测试是等价的。如果一个输入数据不能发现系统错误，则该等价类内部的其它所有输入数据都不能发现系统错误。等价类划分测试方法就是从每一个等价类中选取一个数据作为测试输入，提高测试场景的覆盖率，并减少无效测试工作量。



等价类划分包括有效等价类和无效等价类。有效等价类总结合理的有意义的输入数据构成的集合，即通常所说的常规路径测试输入。利用有效等价类可检验程序是否实现了规格说明中所规定的功能和性能。与之相反，无效等价类是总结不合理或无意义的输入数据构成的集合。

- 举例1：输入条件是一个布尔量，可获得一个有效等价类和一个无效等价类 。

输入条件：是否备份数据。

有效等价类：备份数据（TRUE）。

无效等价类：不备份数据（FALSE）。

- 举例2：输入条件规定了输入值的范围，可获得一个有效等价类和一个无效等价类。

  输入条件：大于1、小于3的数。

  有效等价类： 2。

  无效等价类： 0、4。

- 举例3：规定了输入数据必须遵守的规则，可获得一个符合所有规则的有效等价类和若干个从不同角度违反规则的无效等价类。

  输入条件 ： 大于零的正整数。

  有效等价类 ： 1。

  无效等价类： 0、-10、10.1 …

等价类划分方法重点关注的是输入值的合理划分，设计测试用例时，要同时考虑这两类等价类，不仅要能接收合理的数据，更重要的是要能处理意外场景的输入。使用等价类划分法进行测试设计时，对待分析的特性列举出可能的输入，进行等价类划分和归类，分析输出的每一个等价类作为一条测试用例，可以使用表格或者思维导图等工具辅助进行等价类划分。等价类划分法也常常结合其它方法使用，如边界值分析法。

- 举例：规定了输入数据是月份，月份是从1至12的整数一组12个值，可获得12个有效等价类和一个无效等价类，如下表。

输入条件	有效等价类	编号	无效等价类	编号
输入月份	1,2,3,4,5,6,7,8,9,10,11,12	0001	13	1001

根据等价类划分，编写如下测试用例。

序号	测试用例	覆盖等价类编号
0001	输入正确的月份	0001
0002	输入不正确的月份	1001

边界值分析

经验告诉我们，大量的错误是发生在输入或输出范围的边界上，边界值分析就是在划分的等价类区域的边界及其附近选取测试数据，设计测试用例，可以遵循以下方法和原则。

- 当输入条件规定了值的范围时

  选取刚达到这个范围边界的值，以及刚超出这个范围边界的值作为测试输入。如输入值规定是0-100整数，需要针对0和100设计用例，还要针对-1、1、99、101设计用例。

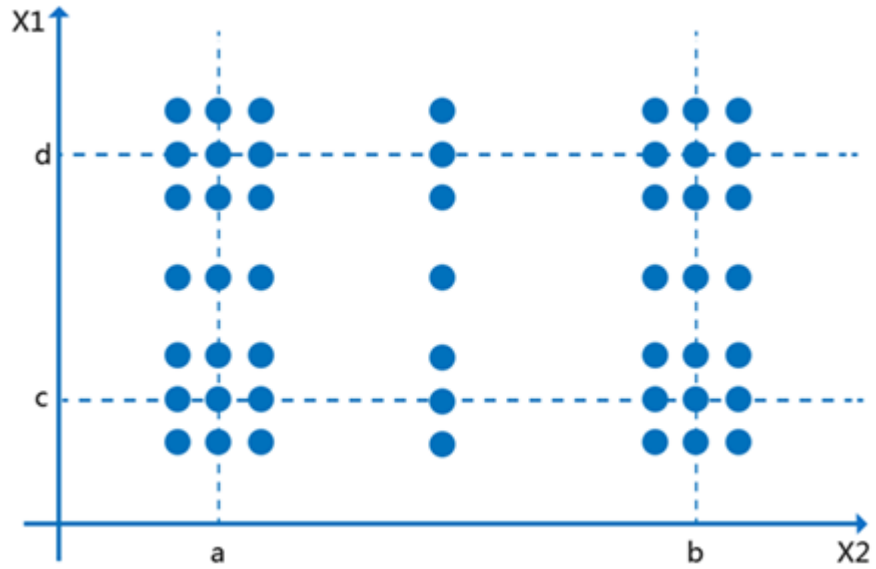
- 当输入条件规定了值的个数时

选取最大个数、最小个数、比最小个数少1、比最大个数多1的数作为测试输入。如上传附件的个数1~10个，则测试用例可取1和10，还应取0和11。

- 当输入、输出是有序集合时

  选取集合的第一个元素和最后一个元素作为测试输入。如输入为有序数组，数组值为1至7，分别代表星期一至星期日，那么选择1和7作为测试输入。

- 在规定了输入数据的一组值（假定n个），并且程序要对每一个输入值分别处理的情况下，可确立n个有效等价类和一个无效等价类。
- 分析规格说明，找出其它可能的边界条件。



经过以上步骤进行分析之后可能会产生大量的测试项，这些测试项中可能会存在重复的情况，此时可以对这些重复的测试项进行合并。边界值分析法经常和等价类法配合使用，在等价类中取值时一般选取边界值。

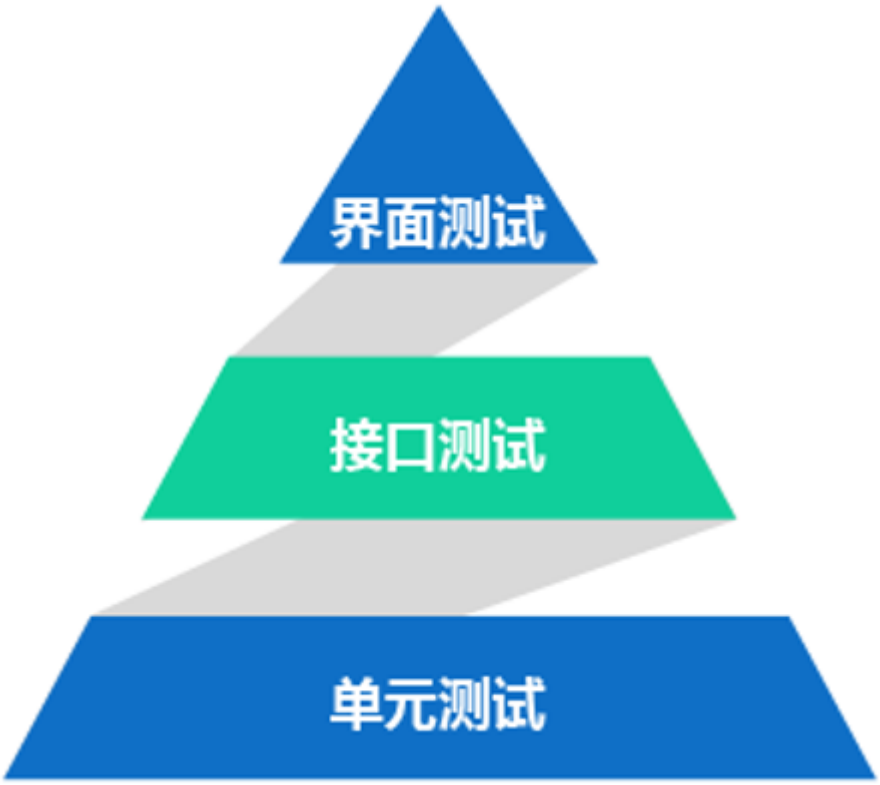
4 测试金字塔和持续自动化测试

敏捷和DevOps开发模式下，产品要具备随时可发布的能力，本文介绍如何应用测试金字塔和CI/CD持续自动化测试实现高效的测试反馈，保障随时发布产品的质量。

测试金字塔

自动化测试金字塔最早是由Mike Cohn在2009年的著作《Succeeding with Agile: Software Development using Scrum》（《Scrum敏捷软件开发》）中提出。最早提出来的时候是一个三层的金字塔，从上到下分别是UI界面/Service服务/Unit单元测试，随着敏捷测试的不断推进，测试金字塔出现一些变种。实际使用中不用太拘泥于每层的名字，在服务化软件架构中Service层也可以理解为API测试。

这种下宽上窄的三角形结构，代表在各层自动化的建议投入分配比例，越接近底层的单元测试建议的投入最多，接口测试居中，界面层建议的投入最少。



Martin Fowler关于测试金字塔有这样一段评论：“GUI测试用例还很脆弱，如对系统的一些修正可能导致很多用例的失败，这时候你需要重新录制。你可以放弃录制的方法来解决这个问题，通过写GUI测试代码，但是这样效率非常低。就算你已经很精通了GUI测试代码的编写，端到端的GUI测试用例也很容易出现不可预期结果的问题，因此，基于GUI的自动化测试是脆弱、耗时（包括用例维护和执行）的。所以测试金字塔要表达的是：底层应当有更多的单元测试和接口测试和逻辑测试，GUI测试用例能覆盖到主业务流程即可。”

测试金字塔中每层中涉及的测试技术均有自己的优势和局限性，由于上层GUI测试的脆弱（不稳定性）、耗时（执行效率）问题，以及问题表现位置（UI）和问题根因位置（代码）距离太远的问题，测试金字塔由关注测试数量转向关注测试质量，推荐增加底层的测试投入。

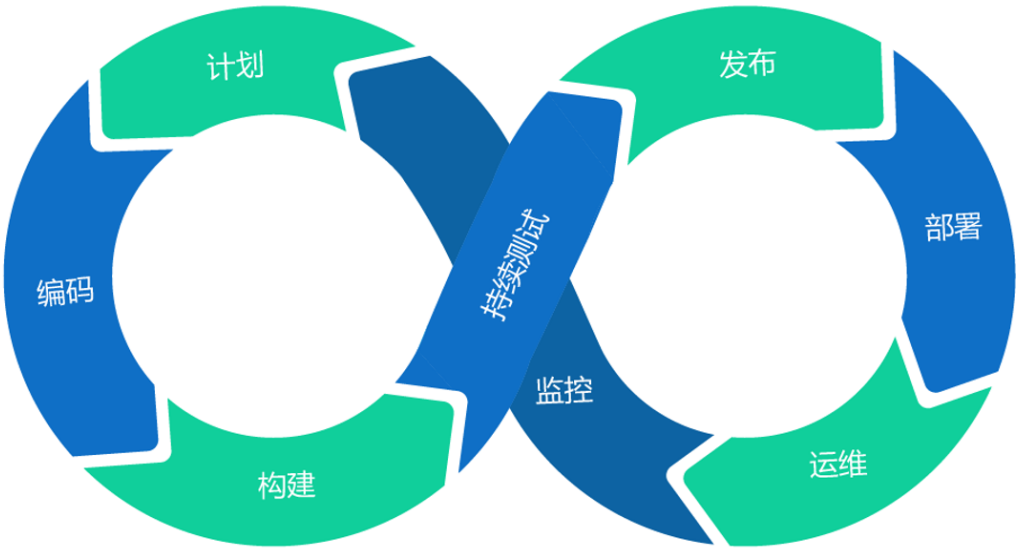
- 层次越靠上，运行效率越低，延缓持续集成的构建-反馈循环。
- 层次越靠上，开发复杂度越高，如果团队能力受限，交付进度受影响。
- 端到端测试更容易遇到测试结果的不确定性。
- 层次越靠下，单元隔离性越强，定位分析问题越容易。

原则上单元测试需要开发人员承担，很多团队中开发人手不足，优先保障功能的实现，在单元测试的投入不够，并且很多开发人员的单元测试经验不足，导致很多团队中不做单元测试或者被动执行流于形式，有人提出了金字塔结构的反模式：**蛋筒冰激凌模式**和**纸杯蛋糕模式**。

蛋筒冰激凌模式由Alister Scott在2012年提出，金字塔中界面测试和单元测试的比例倒置形成倒金字塔，界面测试之上有大量的手工测试。反模式不是建议模式，但却是实际团队中大量存在的。团队初期没有自动化测试，完全依赖手工测试，然后从功能测试自动化入手，产生了一些界面自动化测试用例。这种由外向内进行自动化测试的建设方式，导致了这样的反模式。这样的模式在测试效率、测试用例可维护性角度讲，均存在一些问题，但又往往是很多团队自发进行测试自动化能力建设过程中的必经路径，问题积累到一定阶段需要逐步向测试金字塔方向演进。

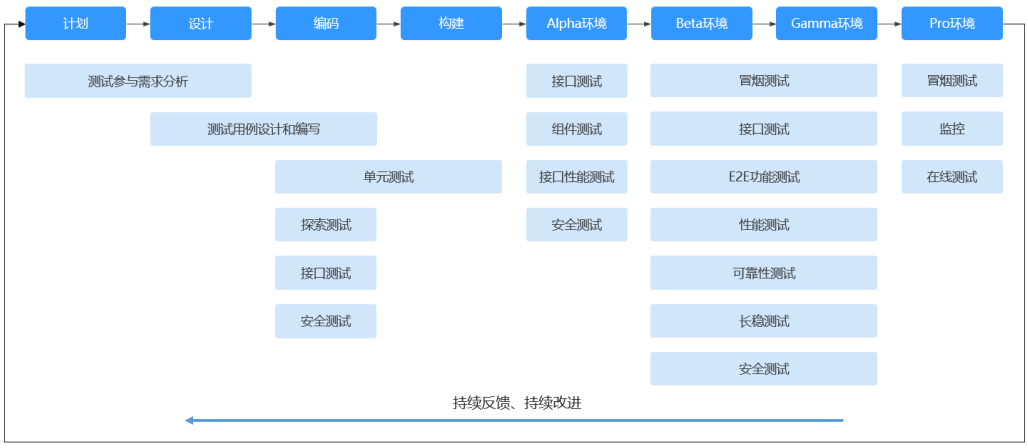
持续自动化测试

持续自动化测试是在持续集成和持续部署过程中运行自动化测试，快速反馈失败，最早源自开发人员在开发环境中通过单元测试获取快速反馈的思想。持续自动化测试是随着CI/CD（持续集成和持续部署）发展而逐步成熟的。现实需要开发人员能够越来越快的发布产品变更，修复在线问题。如果仍然依赖原来的手工测试或者开发和测试完全分离的方式，难以保障在很短的时间窗口中完成测试质量保障活动，因此需要在CI/CD过程中嵌入自动化测试“持续”保障交付物的质量。



持续测试意味着测试活动纳入到持续集成、持续反馈、持续改进循环中，持续不断的测试，贯穿了整个软件交付周期。持续测试提倡尽早测试、频繁测试和自动化测试。

“持续”体现在贯穿了敏捷、DevOps流程中交付物由小粒度逐步演变为软件成品的全过程，从代码白盒测试，到组件模块测试、接口测试、E2E（端到端）功能测试，甚至交付之后进行生产环境的在线测试。每个阶段正好映射了测试金字塔由下向上的各层，越下层的测试在越早的阶段执行，越上层的测试在越后的阶段执行。这类似于汽车制造流水线的各个环节，每个环节的组装结束后都会进行必要的检查通过才进入到下一个环节，在软件DevOps开发过程中Pipeline流水线承载了这样的组装、检查测试过程。

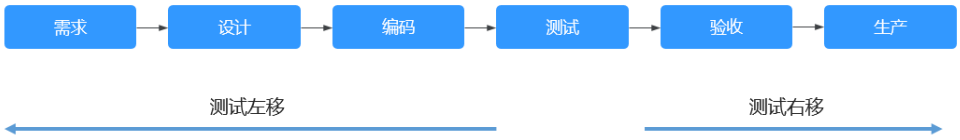


- 测试左移、测试右移

  二者强调在持续测试过程中测试活动越过了传统测试的时间、角色、部门限制，将测试活动发展为连贯的持续性的质量保障活动。

  测试左移强调测试活动尽早开展，测试人员更早地参与到软件项目前期的各项活动中，在功能开发之前定义好相关的测试用例，提前发现质量问题，开发人员参与到测试。

  测试右移强调在生产环境中测试监控，并且实时获取用户反馈，持续改进产品的用户体验满意度，提高产品质量。



- 测试自动化

  持续测试要实现快速流动和快速反馈，需要使用自动化的手段来提高效率，于是自动化单元测试、接口测试、E2E测试就应用嵌入到了DevOps流水线中。自动化测试提高了测试反馈效率，也降低了人为因素导致的错误。测试自动化不仅仅要完成测试用例脚本执行的自动化，还要实现其它所有可以减少人力投入的活动，例如自动化环境创建，自动化部署，自动化监控，自动化数据分析等。

5 缺陷处理流程和注意事项

产品缺陷处理不仅仅是测试提单，开发修复。缺陷问题单应该清晰、全面、可追溯，处理流程包含发现、重现、确认、修复、自验证、回归测试、关闭等环节。

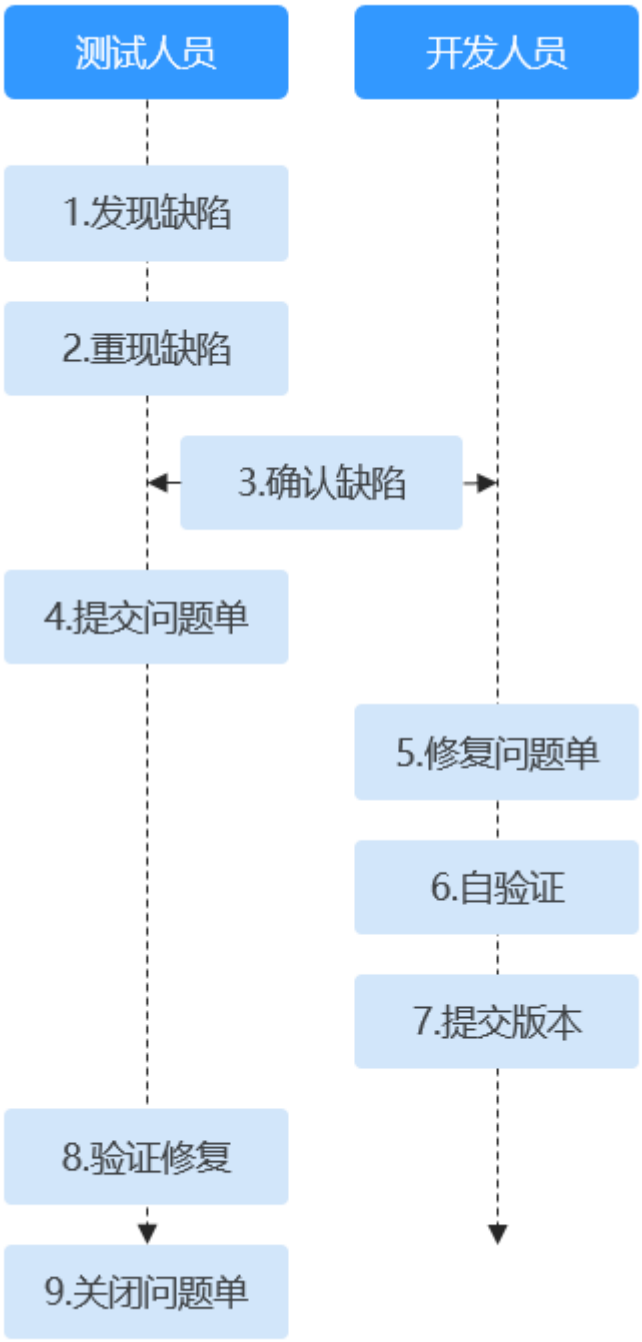
开发、测试的协作问题

产品测试过程中，测试人员会记录缺陷问题单，转给开发人员处理，跟踪问题处理和闭环关闭。缺陷单是开发和测试人员进行信息沟通的一个重要信息载体，开发、测试人员或多或少会遇到过以下问题：

- 开发抱怨测试提交的缺陷描写不细致，比如没有复现步骤、问题所在的软件版本号，导致沟通成本增加。
- 开发在本地开发环境中没有复现缺陷单中提到的问题，直接将缺陷转回给测试。
- 缺陷修复后开发工程师没有通知测试工程师，导致缺陷复查不及时。
- 测试发现问题后，没有扩展对周边相关功能的功能做测试，忽视了潜在的类似问题，开发人员也没有做类似的探索。
- 开发人员对测试人员标记的缺陷的严重级别存在异议。

缺陷处理流程

开发和测试均是软件产品质量的责任人，在产品质量保障方面有着共同的目标和意愿，区别只在于从事的工作活动内容，缺陷处理流程的制定和落地应该本着作二者之间协作的粘合剂和润滑剂的目标，帮助实现互信、高效的协作，而避免作为不作为的借口和矛盾的引火线。以下讲述了一个完整的缺陷处理流程，在实际操作中可以借鉴。



1. 发现缺陷

   在软件开发和测试中，随着代码和模块的叠加和层层调用，一个底层问题可能会表现出多个表象问题。最初发现的问题，仅仅是一个问题表象，此时不能草率给出问题内容和问题原因的结论，而需要做一番有逻辑的系统性分析。

   - 首先，需要发散分析，除了发现的第一个问题表现外，是否还有别的问题表现，这些表现是同时存在的还是有一定的依赖关系、时间先后关系，需要叠加一些更多的测试操作步骤。

     例如：在一次测试中，发现使用手机号+验证码无法成功登录某IT系统之后，需要分析使用手机号+密码等其它登录方式、使用手机号+验证码登录“其

它”系统，使用APP和浏览器或者其它设备登录、使用其它运营商的手机号是否也会出现问题。

- 其次，猜测问题的原因并验证是否是导致问题的原因。这里要避免把“测试步骤”作为原因，而应该分析测试步骤背后引起的数据变化作为原因，由此分析是否有其它场景会出现类似的问题，层层抽丝剥茧，尽量还原出问题的本质。如果是偶现问题，也需要尽量分析问题原因，请开发人员帮助定位定界问题。
- 最后，整理问题发生条件、操作步骤、问题表现。

2. 重现缺陷

   缺陷如果无法重现，开发还是难以定位问题。一般而言，保证缺陷可复现的责任在测试人员，如果问题实在是偶现，难以找到确定性的复现步骤，说明问题根源很深，需要及时寻求开发人员的帮助共同做问题分析。测试人员重现缺陷需要做到：

   - 首先，发现缺陷的测试人员，换一些输入数据或者组合、换一个测试环境，可以按照问题发生条件和操作步骤，重新还原出问题。
   - 其次，其他人员（如开发）根据缺陷文字和截图描述，可以还原出问题。

3. 确认缺陷

   测试人员在提交问题单之前，尽量和开发人员做一轮确认，包括：是否是缺陷而不是优化或者新需求，是否是重复问题，缺陷是否可以重现，是否需要补充问题日志等信息，缺陷严重程度定级是否准确，是否阻塞测试工作需要确定解决日期。

   开发、测试确认缺陷时间不做限定，可以在发现问题到开发启动问题修复之前的任何时间，鼓励尽早进行信息确认，也不限定是否直接找具体的开发人员做确认工作，也可以把缺陷单提交至模块的负责人做统一确认和反馈。

4. 提交问题单

   提交的问题单需要清晰、全面、可管理、可追溯。问题单需要有专门的缺陷管理系统，缺陷管理系统最好和需求、开发任务管理系统同源使用同一系统，以便于做统一的管理和规划。缺陷单一般包含问题单级别、类型、问题描述、根因分析、处理意见、测试建议、关联的测试用例、环境信息描述、开发定位所需日志、截图等。

5. 修复问题单

   开发人员接到问题单后，初步分析工作量安排时间计划。除了修复问题单中描述的问题外，还需要举一反三，进一步测试可能关联的场景，做深入测试，发现可能的深层次问题，并解决。修复问题单后，需要在问题单中介绍问题的根因、发生问题的条件、解决问题的方法。有的缺陷管理系统还可以把问题单和代码提交记录关联，帮助做跟踪统计和追溯。

6. 自验证

   问题修复后除了在本地开发环境验证没有问题外，还需要创建个人构建、部署至测试环境。测试环境尽量和测试人员使用的环境或者发现问题的环境保持一致，以尽量排除环境差异。在测试环境中进一步测试没有问题之后才算自验证通过。在DevOps工具中，可以通过个人级流水线实现个人构建打包、环境部署的全流程自动化，提高个人构建自验证的效率。

7. 提交版本

   代码修复完成后，经过必要的代码评审，发布至目标修复版本的代码分支。

8. 验证修复

   测试人员在测试环境部署包含缺陷修复的代码分支，验证是否完整修复问题。如果问题没有修复，或者修复引入了新的问题，需要把问题记录在缺陷单中，转回给开发人员做进一步分析和修复。

9. 关闭问题单

   在回归测试最终验证问题已经解决，并且没有产生新问题的情况下才能正常关闭。关闭问题单一般只能有三类情况：问题解决正常关闭、非问题关闭、重复问题关闭。可以在问题单中添加一些说明和图片，记录在哪个版本中已经修复解决。

测试计划服务中定制缺陷处理流程

步骤1 确定缺陷状态，例如新建、进行中、已解决、测试中、已拒绝、已关闭，TestPlan缺陷工作项模板已经预置了上述状态，也可以自己扩展添加新状态。

步骤2 设置缺陷状态流转方向，控制缺陷在某个状态下只能向指定的状态流转。

步骤3 设置缺陷预置字段和模板，指导测试和开发人员填写信息。

----结束

6 测试报告编写注意事项

测试报告对测试的过程和结果，也就是测试计划的完成情况进行总结，分析发现的问题，给出产品质量依据，为相关人提供验收和交付决策依据。测试报告一般包含测试概述、测试范围和功能清单、测试策略和方法描述、测试指标统计和分析评价、测试风险分析和披露、质量评价和发布建议等。

测试概述

对本次测试活动的简要描述，阐明目标读者，参考的测试标准，介绍测试背景和测试需求，概要性总结测试对象分析、测试需求、测试内容描述、执行过程描述等，总结主要测试结论。

主要测试结论的编写需要以报告的目标读者、即目标干系人所属角色的关注点为纲。产品经理关注风险披露和产品质量结论，测试经理关注测试成本和测试产出，开发人员关注缺陷结果和产品质量信息。

测试范围和功能清单

- 介绍测试对象的功能、应用场景、价值和作用。
- 介绍测试计划中确定的本次测试的测试范围，详述被测系统（测试对象）的名称、版本、特性、需求、环境、测试项，明确需要测试什么，不测试什么。

测试策略和方法描述

- 回顾测试策略和测试方案，如测试类型、测试场景、测试方法，策略性说明如何测试，介绍测试使用的方案，例如：集成步骤和顺序、测试步骤和顺序、测试方法、测试工具、测试用例设计和执行方法等。
- 描述测试环境，如测试所使用的硬件、软件、测试工具的名称、规格、数量、版本、帐号等信息。
- 总结测试周期和测试人员投入，即测试的计划开始和结束时间，测试总体进度，关键的阶段性进度检查点情况，测试人员数目、分工、投入工时等。

测试指标统计和分析评价

- 测试关键性指标统计：某些专项测试，例如速度、吞吐量、温度、时间、资源占用率等被测系统质量可量化指标的测试结果统计。

- 缺陷统计和缺陷分析：统计缺陷总数、按级别统计缺陷、缺陷解决率、缺陷重启率、遗留问题单数目、按模块缺陷分布、缺陷来源分布等。应用缺陷正交分析、四象限缺陷分析等。
- 测试执行情况统计：统计设计的各类测试用例数量和比例、执行测试用例数量、测试用例执行通过率、回归测试次数，测试执行人力投入和测试执行周期等。
- 测试充分性和测试能力统计：统计需求和功能特性覆盖情况，测试执行完成率、代码测试覆盖率、测试自动化率、测试用例缺陷命中率等。

测试风险分析和披露

根据测试过程和结果，分析产品是否有质量风险，列举风险、风险依据、风险级别，以及风险应对建议。风险并不意味着质量不达标，对风险的处理策略，取决于风险发生概率和发生后的损失估计，如果两者乘积很低，可以做风险接纳。

风险是主要干系人判断产品质量整体情况以及是否具备发布条件的重要依据，一定要填写，而且要逻辑严密。

质量评价和发布建议

根据公司或行业标准，以及测试的质量结果和风险分析，给出客观的质量评级和评价，供相关人参考。

根据质量评价给出发布、推迟发布、部分发布等发布建议，发布建议可以具体到特性，风险级别高的特性可以不建议发布或者受限发布。

