软件质量体系白皮书

/thoughtworks

Strategy. Design. Engineering.

前言

软件质量体系概述

关于软件质量，不同的人根据自身的领域知识和经验，对其有着不同的认知：有人简单地将软件质量等同于软件测试，有人说到软件质量就会想到是否有缺陷，也有人对软件质量的认知停留在是否好用的层面……实际上，软件质量内涵丰富，既包括可感知的、跟用户使用相关的外部质量，也包括影响外部质量的软件技术架构和代码相关的内部质量，还有整个软件开发生命周期中各个环节相关的过程质量。

在 Thoughtworks，我们认为软件开发是复杂的社会活动，随着业务形态的增多、技术架构的演进，软件质量的复杂性不断增加，影响软件质量的因素也越来越多。软件质量不能仅靠传统意义上的测试活动来保障，测试人员要延伸测试边界，以更全面、更系统的视角来构建软件质量体系，助力组织提高软件产品的质量。

如果我们开始设想一套完善的质量体系，这个体系应该包括许多方法论和实践，并需要自上而下的管理框架和自下而上的主动意识。

- 自上而下的框架是基于顶层设计，整体包括哪些部分，它们的作用是什么，以及互相之间的关系等。
- 自下而上的主动意识则是每个员工基于不同的角色，主动关注与质量相关的工作，并积极履行应尽的质量职责。

因此，质量体系可以由以下五大部分构成：质量图景、质量策略、质量实践、质量基础设施和质量人员。

质量图景

质量图景是质量体系的概览，展示组织的整体质量愿景和战略规划。它通常包括组织对质量定义、目标、价值观、标准、流程、方法和指标等方面的要求和规定，旨在为组织提供一个整体的、统一的、可持续的质量管理框架，以确保产品或服务能够持续地满足客户的需求和期望。

质量策略

质量策略是规划和指导质量实践活动开展的指南，包括测试流程、测试左移、持续测试、测试右移、质量门禁和质量度量等策略。

质量实践

质量实践是基于质量策略开展的具体质量活动，是实施质量保障的核心。质量实践贯穿整个软件开发生命周期，包括需求分析阶段、需求实现阶段和线上运维阶段。

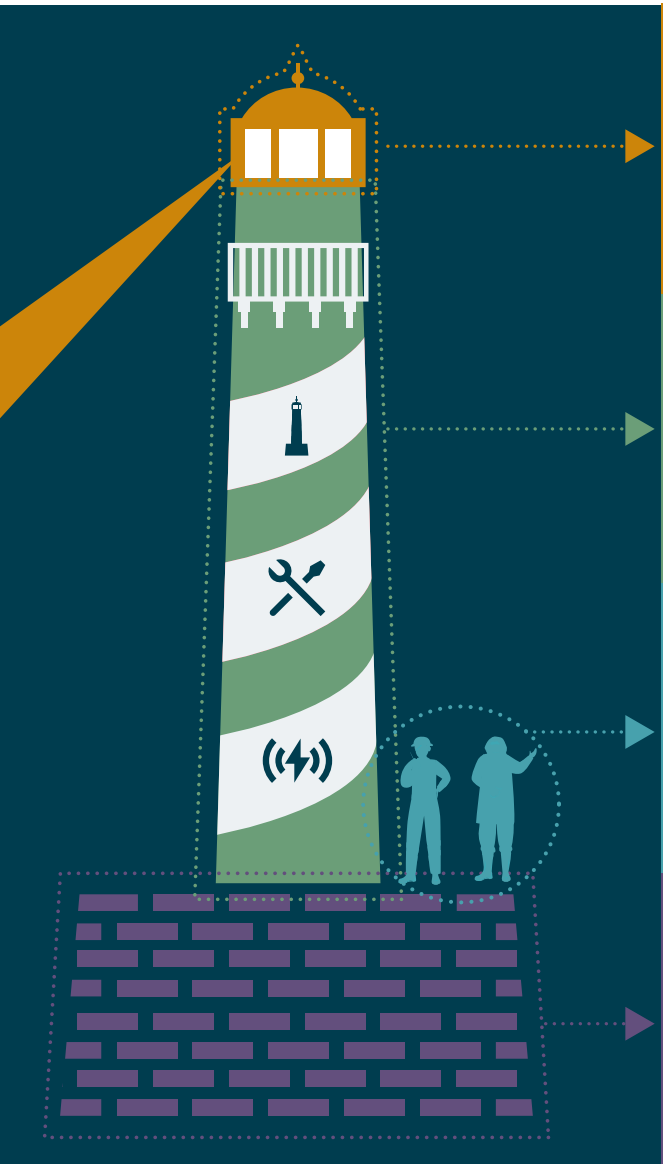
质量基础实施

全流程的标准化和大规模的自动化是高效实施质量实践的关键，而质量基础设施是标准化和自动化的有力支撑，包括支撑标准化策略规范落地的全流程管理平台、支撑持续自动化测试的工具框架和环境等。

质量人员

清晰的质量图景，明确的质量策略，规范的质量实践，成熟的基础设施，最终都需要质量人员来实施，质量人员是质量体系的基石。质量人员不仅指测试人员，包括所有参与质量工作的人员。质量人员相关的质量文化建设、质量能力建设、绩效管理和多角色协同等，都是质量体系需要考虑的内容。

软件质量体系灯塔



质量图景

质量策略、质量实践

质量人员

质量基础设施

灯塔作为一个指引方向的比喻，我们可以用它来描绘软件质量体系的整体构成。灯塔的各个部分与质量体系的各个组成部分对应如下：

灯塔顶部的灯光代表质量图景。

灯塔的灯光为船只指引方向，质量图景为整个组织软件产品质量提供了明确的导向。

灯塔的主体代表质量策略和质量实践。

正如灯塔的主体，质量策略和实践构成了整个质量体系的主体，是实现质量目标的核心。

灯塔的工作人员代表质量人员。

灯塔工作人员的努力使得灯塔能发挥其功能，质量人员的高效协同是实现质量目标的必要条件，而质量文化和团队价值观决定了质量体系各个部分如何发挥作用。

灯塔的基石代表质量基础设施。

灯塔的电源系统和通信设备是灯塔的基石，为其正常运行提供必要支持，就像质量基础设施在质量体系中为质量实践的实施提供支撑一样。

将质量体系框架与灯塔概念相结合，我们可以更直观地展示质量体系的各个组成部分以及它们之间的关系。灯塔为船只提供了指引，确保航行安全，同样，一个完善的质量体系为组织的软件产品提供了明确的质量方向，有助于确保软件系统达到高质量标准。

质量策略

测试流程

这里的测试流程并不是指执行某项测试需要哪些步骤，而是指在软件开发生命周期中需要在哪些环节实施测试相关活动。测试流程的定义就是将这些内容固化下来，形成规范。团队全体成员可以基于这些规范来执行相应的测试实践。测试流程通常与软件开发流程密切相关，需要基于开发流程来定义。

为什么需要定义测试流程?

如果没有流程规范，大家随意开展测试工作，会导致混乱。测试流程相当于测试活动开展的框架，类似编程框架。可以用来批量培训人员，让大家按照统一的方式来实施测试，规范每个人的测试行为，减少犯错的可能性，尽可能提高测试的效率和有效性。

相关主题

- 测试左移
- 持续测试
- 测试右移

示例

测试流程规范业界目前主要有以下三种形式：

1. 制定流程，编写流程规范文档，团队依据文档执行；
2. 文档定义流程，同时配套工具平台来规范执行，比如流水线、看板等；
3. 将流程全部固化到工具平台，利用工具平台实现全流程的标准化和自动化，如 Google 等。

常见误区

1. **测试是一个独立阶段，只有开发完成后才能进行测试。**
2. **软件发布后测试工作已经完成，不需要在生产环境进行测试。**事实上，测试不再是一个独立的阶段，需要在从需求到生产环境的全生命周期各个环节穿插进行，是跟需求分析、开发编码、线上运维并行的一系列活动，并且需要持续不断地进行。

测试左移

测试左移并不是全新的测试方法，而且早在 2001 年，Larry Smith 就在他的文章《Shift-Left Testing》中提出了测试左移的概念和相应的实践。测试左移的本质就是尽早地进行测试相关的工作，将这些工作移到开发过程的早期（例如业务分析和设计过程），让 QA 和 Dev 参加需求分析和需求评审，尽早进行测试分析，从而发现不合理的需求，不合理的设计等。

为什么需要将测试左移?

首先，众所周知，软件开发中最大的浪费是返工。无论是因为修补 Bug 而返工，还是因为需求理解错误而写错了代码而返工，都是巨大的浪费。测试左移是解决这些浪费的有效实践之一。

其次，软件缺陷产生的主要原因包括：业务设计本身有误、Dev 对业务需求理解有误、Dev 编写代码有误等。测试左移的主要目的是在编码阶段之前统一 BA、QA 和 Dev 的认知，以尽可能防止由于理解不正确或不完全而导致的缺陷，并将测试工作左移到 Dev 编写代码的过程中，以防止代码编写错误导致 Bug。

因此测试左移能为项目带来以下收益：

- 早发现早修复：测试左移可以尽早发现业务需求、架构设计和代码中的缺陷，从而能够及时修复它们。
- 节约返工成本：测试左移可以减少代码开发完成后的缺陷数量，从而节约修复它们所需的返工时间和人力成本。尤其是时间成本最为重要，降低时间成本可以提高软件交付的效率，缩短交付时间。
- 更好的团队协作：测试左移需要 QA 与 BA 以及 Dev 进行大量合作，从而可以促进团队不同角色人员更好地协作。
- 更好的测试有效性：测试左移也可以帮助 QA 以及 Dev 更好地分析和理解业务需求、更多的业务分支和异常情况，并且可以帮助 QA 获得更有效的、覆盖率更高的测试用例，也可以帮助 Dev 更准确和全面地进行任务拆分，从而获得更高质量的产品。
- 更早的自动化测试：由于测试左移可以在开发之前获得有效性更高的测试用例，从而可以帮助 Dev 和 QA 更早更好地实施自动化测试。在项目中实施自动化测试时，测试左移是一个重要的起点。测试左移实施得好，可以为自动化测试打下良好的基础，尤其是编写高有效性的验收条件或验收测试用例，能够帮助团队更好地实施自动化测试和 TDD 等。

相关主题

- 需求澄清
- 需求评审
- 代码评审
- Kick off
- Desk Check

常见误区

1. **测试左移只是将 QA 移动到分析和开发阶段做测试工作。**在测试左移的观念里，左移的并不是 QA 或执行测试的角色，而是为了更早发现缺陷、降低修复成本而左移的相关测试活动。
2. **QA 可以独立实施测试左移，不需要和 Dev、BA 等角色产生协作。**因为左移的不仅仅是 QA 角色所做的工作，而是所有跟更早发现缺陷、降低修复成本相关的活动，需要跨角色协作才能完成。
3. **QA 不需要提高自己的业务分析和软件开发技能，就可以高效地实施测试左移。**QA 需要业务分析的能力，才能参加需求的分析和评审。以及，掌握一定的软件开发技能，才能在设计阶段进行测试分层，做测试设计和测试用例编写。

参考

- Shift-Left Testing
- 测试左移实践
- 测试左移

测试右移

测试右移是与测试左移相对应的，它将测试活动从独立测试阶段右移到生产环境，也称为“生产环境下的 QA”。

由于生产环境的特殊性，测试右移并不是简单的测试活动右移，而是通过一些实践活动获取生产环境下用户行为、日志等质量相关信息，利用这些信息为前期的需求、开发和测试工作提供反馈，促进相应工作的优化改进，以更好地实现质量内建。

为什么需要将测试右移？

随着软件技术和架构的不断演进，软件系统的基础设施日趋复杂，业务和数据量也大量增加，生产环境中不稳定的因素在持续增多，导致系统行为变得难以预测，软件系统的不确定性日益严重。

人们无法通过预先设定的测试场景和测试脚本来有效地测试软件，开发和测试环境已经不够用，软件系统的质量保障工作受到挑战，软件系统变得愈加脆弱。为了应对这种不可预测性和脆弱性，需要将测试活动右移到生产环境中。

测试右移将质量保障的范围从需求扩大到生产环境，增加反馈来源，结合持续交付，可以帮助持续提高产品质量和优化业务价值，增强软件产品的反脆弱能力。

相关主题

- 线上回归测试
- 线上监控
- 缺陷分析

常见误区

1. **盲目关注测试右移，忽视其他持续交付实践。**测试右移仅适用于持续交付实践比较成熟的组织。如果连开发过程和测试环境的质量保证都做不好，追求测试右移只会事倍功半，舍本逐末。

参考

- QA in Production
- 测试右移——生产环境下的 QA
- 测试右移：QA 与 Ops 通力合作打造反脆弱的软件系统
- 测试右移之日志收集与监控
- 测试右移：缺陷分析如何帮助质量内建

持续测试

持续测试将测试作为整个软件开发过程中不可或缺的一部分，在各个环节进行测试验证活动。它对每个不同版本进行频繁的测试，内容包括持续功能测试、性能测试、安全测试等。形式可以是静态分析和评审，也可以是动态的手动和自动化测试。持续测试包括以下两个方面的内容：

- 在软件开发全流程的每个环节，通过频繁执行不同的测试活动，实现多环节的全面缺陷快速发现。
- 通过持续集成，将代码静态扫描和自动化测试集成到流水线，并针对每次提交到版本仓库的代码进行持续、自动化的验证。

为什么要持续测试?

只在软件“开发完成”以后才进行测试，反馈得太晚，这个阶段测试发现的问题导致返工修复成本高。而全生命周期的持续测试可以快速获取反馈，尽早发现过程中可能的缺陷和偏差，以降低成本，提高质量。例如：

- 对每次向版本仓库提交的代码都执行自动化的单元测试和验收测试，可以快速地为 Dev 提供关于本次代码变更的反馈，以便及时修复相关的缺陷。

相关主题

- 持续测试
- 测试左移
- 测试右移

常见误区

1. **只有流水线上的自动化测试是持续测试。**其实在整个软件开发生命周期中，还有很多手动验证的活动也都属于持续测试，而且是必不可少的。
2. **在发布前将所有测试案例都执行完了，就算测试完成了。**其实频繁在持续集成的新版本上持续进行探索式测试尤为重要，这样有利于发现脚本遗漏的新版本上潜在的缺陷。

参考

- 敏捷测试的核心

质量门禁

为了提高质量内建的意识和水平，我们需要在开发全过程中嵌入一些检查点，以检验当前的工件是否满足进入下一阶段的质量标准。这些检查点、检验标准和检验动作可以统称为质量门禁。

为什么需要建立质量门禁？

质量门禁可以在开发过程中发挥质量牵引的作用，确保流入下一环节的工作内容具备基本的质量水平，将风险和缺陷阻挡在当前阶段，促使团队优先处理。质量门禁可以带来以下价值：

帮助规范开发过程

通过建立明确的质量标准，质量门禁可以帮助规范开发过程，并提议对各阶段的质量认知保持一致性。

提升过程质量

质量门禁中可约束技术标准、管理标准、质量标准等多重标准，可帮助团队提升过程质量，在满足开发需求的同时，沉淀团队的优秀工作方式。

提前发现缺陷和风险，避免缺陷在测试阶段集中井喷

质量门禁的门禁作用可防止质量风险流入下一环节，促使团队及时处理当前阶段的风险和缺陷。

常见误区

1. **质量门禁导致交付延期欲速则不达。**不能因为要快速交付而降低质量标准。相反地，质量门禁可以将交付风险前置，促进更有效的交付，减少后期的返工成本。

2. **质量门禁是 QA 的职责，跟其他角色无关。**质量是团队整体的能力。质量门禁的建立和遵守需要团队中的每个人共同参与，质量门禁也会影响团队中的各个角色的工作内容，与团队成员息息相关。
3. **质量门禁只关注软件质量，而忽略其他方面。**质量门禁有助于规范过程质量，即关注软件质量，也关注团队整体的交付过程质量和交付体验。质量门禁的制定不仅局限在软件质量，凡是影响团队整体的关键节点，均可设置质量门禁。

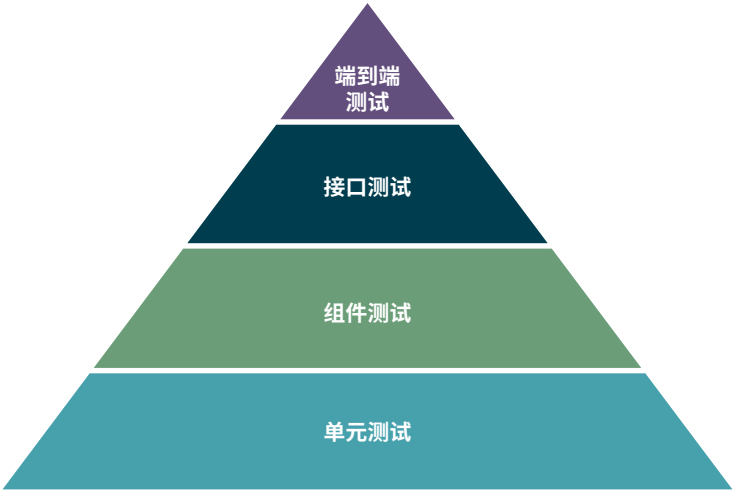
参考

- 在“去 QA 化”项目中，QA 要帮助团队形成高质量意

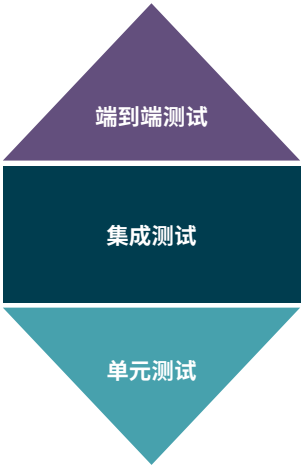
测试分层

测试分层就是根据软件系统本身的代码和架构层次，针对不同的代码层次进行不同的测试，例如单元测试、组件测试、接口测试、端到端测试等。通过利用不同层次的优缺点，系统化地互相配合，从而高效地完成软件的功能测试。

测试分层策略应根据项目的各种情况（如人力、时间、质量需求等）确定需要在哪些层次进行什么类型的测试，以及各种测试之间的关系、优先级和比例等。例如金字塔分层策略可以作为常规的分层策略，适合大部分传统项目；而蜂巢分层策略则更适合于微服务项目。



金字塔分层



蜂巢分层图

为什么需要测试分层?

通过界面级别的功能测试来全覆盖所有的测试场景，这样所需的资源投入非常巨大，而且执行效率偏低，很难满足软件开发过程中不同类型的验证和测试需求，甚至有些测试场景无法通过界面功能测试来完成。其次，随着软件规模的增加，软件架构持续演进，层次也越来越多。其中，越接近用户层的测试越能体现业务价值，集成也越多，执行效率越低；而越接近底层的测试，越能验证代码细节，执行效率越高，集成越少。

因此，在有限的人力和时间资源下，需要制定合理的测试分层策略，以尽可能小的成本完成更高的测试覆盖率，并提升测试执行效率。

如果没有有效的测试分层策略，往往很难控制不同层次之间的测试工作量和优先级等，从而导致不小的浪费，甚至无法保证基本的质量要求。

测试分层还可以帮助团队中的所有人（包括不熟悉测试和质量的人）快速清晰地了解测试工作的整体内容和不同层次测试之间的区别。

相关主题

- 端到端测试
- 单元测试
- 持续集成中的自动化测试
- 集成测试

常见误区

1. **测试分层可以做到每层测试所覆盖的场景和代码不重复，从而避免浪费。**实际上，不同测试类型之间几乎不可避免地存在重复部分，而且很难精确统计这些重复。因此，测试分层只是一个高层指导，实施时需要尽可能减少重复部分，而不能完全避免。

2. **自动化测试分层只有金字塔模型。**测试分层的具体实施受到多个因素的影响，包括系统架构、代码结构、人员能力、质量要求等。基于不同的情况，也会有不同的测试分层结构，比如遗留系统可能采用的是倒金字塔，而微服务架构的系统则更适合蜂巢模型。因此，在实践中，应根据具体情况选择适合自己的自动化测试分层模型。
3. **测试分层只针对自动化测试。**测试分层的对象可以包括手动测试和自动化测试。然而，大部分项目都倾向于使用自动化测试来实施测试分层策略，这导致了对测试分层策略只能通过自动化测试来体现的误解。

参考

- TestPyramid
- 微服务测试 - 微服务测试的思考与实践

质量度量

质量度量是指采用流程、实践、方法或工具，在软件的全生命周期内对软件质量进行定量和定性分析，从而指导质量改进和效果评估的一系列过程。

质量度量的对象是软件质量。软件质量可按照用户是否直接感知分为内部质量和外部质量。因此，一切可以评价软件内部质量或外部质量的指标、方法，都可以算作质量度量的范畴。

由于度量本身既可以是对某一数量的评估或测定，也可以是对某些抽象特质的描述或比较，因此质量度量既包括对质量相关或影响质量的指标进行定量分析，也包含对质量特征或可影响质量的其他特征进行定性分析和判定。

为什么需要度量质量?

质量度量有助于帮助团队整体感知质量状态、评价质量活动成效、预测质量风险、辅助团队质量改进的过程。

感知质量状态

开发团队整体需要能够清晰感知软件的质量状态，从而更高效地提供高质量的软件。质量相关的管理者如产品负责人、开发负责人或质量经理等，也需要通过质量度量来了解质量状态。

评价质量活动成效

为了保证软件质量，团队会开展各种质量相关的实践活动，如需求评审、开发自测、缺陷分析与预防等。我们需要知道这些质量活动的成效如何，是否能帮助提升效率改善质量，这些实践对质量的影响需要通过质量度量来体现。

暴露问题以引导质量改进

通过有效的质量度量，我们大概率能获得当前软件的内外部质量状态，以及我们采取的质量实践对质量的或直接或间接的影响。通过收集和分析不那么理想的数据和实践，可以在一定程度上暴露团队痛点问题，帮助引导质量改进的进程。

预测质量风险

团队可以基于历史质量度量的分析结果，得出影响质量的关键约束或团队关键事件。由质量度量延伸出对于团队或组织流程的认知，有助于帮助团队识别和干预后续的质量风险。

作为软件开发生命周期中其他活动的有力输入

如产品相关决策、用户洞察、体验设计等。

相关主题

- 单元测试
- 集成测试
- 端到端测试
- 非功能测试
- 验收测试
- 探索式测试
- 缺陷分析
- 持续集成中的自动化测试
- 质量度量可视化
- 质量报告

常见误区

1. **质量度量只服务于绩效考核。**质量度量不应该以服务于绩效考核为最终目的。如果将度量和绩效考核等同起来，非常容易忽视度量在管理决策中的作用。度量应该为团队整体服务，团队内的所有成员都需要关注度量的结果和其反映出的问题。
2. **质量度量给团队造成割裂，导致员工内卷。**质量度量是手段而非目的。日常观察到一些度量导向不好的结果的现象，往往是因为对度量的误读和误用，而非度量这个行为本身的问题。
3. **度量是给开发团队设计的紧箍咒。**这是对度量目标的一个误解，不管是采用什么度量方法，最终都应该导向更好的结果，而不是更差的结果。度量是为了帮助团队提高质量和效率的工具，而不是施加在开发团队身上的限制。

参考

- 敏捷项目度量怎么做
- 开发效能度量指标
- 全生命周期的质量度量
- 质量度量之定量分析

- 质量度量之定性分析
- 质量度量之全局优化
- 怎样度量需求质量

质量改进

为了提升软件质量和过程质量，组织内需要实施一系列改进措施和改进过程，这就是质量改进。作为质量管理的一部分，质量改进旨在提升组织满足质量需求的能力。

质量改进的对象是产品或服务质量以及与其相关的工作质量。最终的目标是获得比现有质量更高的产品或服务，并得到更好的质量评价。

通过质量改进的定义，我们可以得出改进的对象不仅局限于软件质量，还应包括提高交付软件过程的效率和流畅度，以及提高用户评价的方法。

为什么需要改进质量?

质量改进可以以更低的成本、更高的效率提高团队的质量。

提升外部质量

质量改进可以帮助提升外部质量，在生产环境获得更好的软件表现。

提升内部质量

质量改进可以帮助提高内部质量，优化代码质量，还清技术债务，从而获得更高的可维护性等。

提升过程质量

质量改进可以帮助提高过程质量，包括但不限于：提高交付速率，提高构建效率，缩短需求前置时间，提高流转效率等，从而获得长期的降本增效效果。

提升用户满意度

通过持续的质量改进，提高软件的外部质量、内部质量和过程质量，从而获得更好的质量评价，提高用户满意度。

构建共同的质量愿景

质量改进有助于团队逐步实现共同愿景中的长期目标。而这个长期目标的制定，必须是基于团队所有成员的共识所构建的共同愿景。

形成持续改进文化

质量改进过程更有助于团队形成持续改进文化。当遇到质量相关问题或事故时，团队会更愿意站在如何改进质量和规避风险的角度来分析问题，而非把问题的产生归因于个人主观因素一味追责。

内建质量的闭环思维

由于质量改进需要持续进行持续观测，更有利于团队构建闭环思维，进一步思考改进项的短期直接成效和长期成果。

相关主题

- 质量度量可视化
- 质量报告

常见误区

1. **没有明确改进目标也可以进行质量改进。**如果不明确质量需求，在追求质量的道路上做出努力就是徒劳无功。虽然看上去在持续改进，但很难获得预期的成效和评价。质量改进需要持续关注，才能获得更好的效果。
2. **局部改进了一定会对全局有利。**为了提升某一部分的质量，采取武断或破坏全局的改进方式，短期看局部确实得到优化，但长远看全局的质量建设可能受损。在质量改进的过程中，请务必以全局全程视角来设计过程。
3. **管理层可以根据自己的意愿要求团队进行质量改进。**如果质量改进只是少数人的要求，而没有达成团队整体的共识；或者如果质量改进只是为了服务于管理层的绩效要求，而没有真正服务于团队 ..... 这样的情况下，质量改进都会因为推进者和执行者没有达成共识而无法真正落地。

参考

- 在“去 QA 化”项目中，QA 要帮助团队形成高质量意识

质量实践：需求阶段

需求澄清

需求澄清是指在交付开发前，由不同角色触发的对需求内容进行澄清和确认。简单来说，就是需求相关的各个角色对需求本身产生的问题进行协调的过程。需求澄清是需求进入开发前必不可少的一步，也是提升需求质量的关键环节。在需求澄清中，QA 需要发挥关键作用，提出有意义的问题，促进需求质量的提升。

为什么需要澄清需求?

澄清需求价值

需要明确业务价值和用户价值的理解是否一致，这是确保需求价值正确交付的关键。开发中常见的一个误区是“做的不是用户想要的”，这通常是由于需求价值的理解和传达出现了问题。

对齐需求理解

在价值共识的基础上，对齐需求还需要各个角色对需求细节的理解达成一致，例如对业务分支、用户使用方式和常见异常情况的处理等。

减少返工

进行需求澄清有助于减少开发过程中的返工。如果在开发前未对需求进行有效澄清，有可能导致在开发过程中，需求交付价值、业务流程、业务细节等方面出现较大变更，进而造成返工的浪费情况。

明确完成定义（DoD）

通过澄清验收标准，可以帮助各个角色对需求完成的定义有清晰的理解和共识。尽早确定需求的验收标准，可以最大限度地确保开发过程不偏离需求价值。

相关主题

- 需求评审

常见误区

1. **需求质量是产品负责人（PO）或 BA 的工作，不需要其他角色过多关注。**产品需求作为开发侧的关键输入，影响着开发体系下的所有角色。因此，所有人都需要为产出高质量的需求而努力，需求的思考和澄清是每个角色的职责。
2. **需求澄清需要尽可能详细，最好覆盖所有的测试用例。**然而，需求并不需要详细到每个测试点的粒度。需求应该是可以讨论的，重点在于其价值明确，而实现方式和细节是可以协商的。

3. **需求澄清应该在开发之前进行。**有些人可能认为，只有在进入开发之前才能进行需求澄清，进入开发后就不应该再澄清需求了。这是一个误区。在整个产品开发测试过程中，任何角色对需求有疑问，都可以随时进行澄清。如果在开发过程中进行需求澄清，导致需求内容或需求范围发生变化，则应由团队共同决定如何处理这种变化。

参考

- 验收标准到底是不是测试用例?

需求评审

需求评审由需求相关的关键角色共同参与，针对即将付诸开发的各个需求逐一审视，验证需求价值，讨论实现方案和技术方案，明确需求交付成功的验收标准。有时也会在需求评审时进行需求拆分方法和拆分粒度的讨论。

为什么需要评审需求?

需求评审的价值和需求澄清是类似的，也起到确认需求价值、对齐需求理解和降低需求风险的作用。除此之外，需求评审的最主要价值是获得对需求的确认，明确这些需求可以进入研发阶段。

另外，在需求评审的过程中，检验需求的价值和完整性非常重要。这有助于减少需求传递中的失真，并且能够提前识别和干预需求交付的风险。

相关主题

- 需求澄清

常见误区

1. **需求评审应由最了解需求的人来进行，其他角色无需参与。**需求评审应该尽可能地让所有角色参与，不同角色的人员需要就需求达成一致意见，并且应基于自己的专业背景来进行进一步的评审和判断。
2. **在需求评审前不需要详细了解需求。**有些团队在需求评审之前，相关角色并未详细了解过需求，导致在评审时还需要对各个需求细节逐一讲解，造成大量时间浪费。需求评审会议通常作为产品开发的需求准入，是需要团队给予重视的。但需求评审的重点在于“评审”和“共识”，可以把对需求的阅读、初步理解和疑问在评审会议前完成，会议时重点澄清疑问、达成共识，而不必要在评审会上就需求的各种细节逐一讲解。
3. **在需求评审之前，不需要对需求进行明确的澄清。**还有一种情况也会导致评审会议冗长：在评审前，BA 没有澄清需求的价值、范围和细节等内容。当其他角色在评审时提出疑问时，很难快速做出回应，会陷入过多的细节讨论。比较推荐的做法是，在需求评审前，BA 应该已经完成了细节的澄清，这样就可以从容应对评审时大家提出的问题。

参考

- 怎样梳理需求全景？
- 怎样度量需求质量
- 测试左移：需求相关的质量保障

质量实践：实现阶段

Kick off

Kick off 全称为 Story Kick off（用户故事启动），也叫开卡，是指在 Dev 开始实现用户故事之前，与 BA 和 QA 再次澄清和确认自己对用户故事内容的理解。这是用户故事（或需求条目）生命周期中的一个环节。

为什么要做 Kick off？

1. Kick off 主要价值在于确保 BA、Dev 和 QA 对用户故事内容的理解一致，减少因理解不一致产生的缺陷和返工；
2. 虽然在需求澄清和需求评审等确认环节中进行了确认，但随着需求进一步细化，用户故事的内容可能会发生或多或少的变化。此外，时间间隔较长可能会导致对故事的理解发生偏差。因此，在正式开发之前再次进行澄清和确认是非常必要的。

相关主题

- Desk Check

常见误区

1. **简单浏览一遍 AC（Acceptance Criteria，验收标准）即可。**AC 确实是用户故事中非常关键的部分，但是 Kick off 不仅要检查 AC，还要仔细检查故事中的所有描述，包括假设、依赖和跨功能需求等，以发现遗漏的内容并予以补充。
2. **对于需求已经有很多讨论会和评审环节了，再加上 Kick off 未免会议太多了，没太大必要，感觉有些浪费。**Kick off 是保证在故事卡进入正式编码之前的最后一刻进行对齐和确认的活动。形式可以尽量轻量级，就在 Dev 的屏幕前进行。Dev 描述自己的理解，其他角色进行确认和必要的补充。时间控制不要太长，一般 15 分钟以内为宜。如果还有很多未澄清的内容，说明故事还没准备好，不建议对该故事进行 Kick off。
3. **一个迭代要开发的所有故事一起通过正式会议 Kick off，团队所有角色参与。**通常情况下，只需要负责开发某个故事的 Dev 来主导 Kick off 活动。这个活动应该在故事开发之前的最后一刻进行。不建议提前组织正式的 Kick off 会议来讨论多个故事，也不需要其他不参与该故事开发的 Dev 来参加 Kick off 活动。

参考

- Thoughtworks 的敏捷测试

单元测试

单元测试主要是针对软件系统中的一个代码片段（称为单元）进行自动化测试，以检查该代码片段是否满足设计和功能要求。在基于过程式语言的代码中，一个单元通常可以是一个模块（module），更常见的是一个独立的函数（function）或者过程（procedure）。而在面向对象语言的代码中，一个单元通常是整个接口，例如一个类（class）或一个方法（method）。通常由 Dev 负责编写单元测试。

为什么需要单元测试?

对于大型的软件系统，使用黑盒功能测试进行测试时，很难保证代码的测试覆盖率。而且，功能测试执行速度缓慢，通常需要在真实测试环境中执行，这导致 Dev 无法快速获得有关其新编写代码质量的反馈，导致问题无法及时修复，很可能致使整个项目延期。

其次，单元测试可以有效地帮助 Dev 进行代码重构。因为它在 Dev 完成代码重构之后快速执行，可以快速发现重构后的代码是否破坏了原有的功能，从而增加 Dev 重构的信心。

相关主题

- 分层策略
- 单元测试评审
- 代码覆盖率
- 持续集成中的自动化测试

常见误区

1. **单元测试中的单元只是指一个函数（function）或者一个方法（method）。**单元测试最大的误区在于对于单元的理解。虽然理论上一个单元可以是一个函数（function）或者一个方法（method），但在现实情况中，如果每个函数和每个方法都写单元测试，那么开发成本和维护成本都会非常高。因此，在真实的项目中，为了追求效率和节约成本，在保持覆盖有效性的前提下，一般都会基于业务模块来编写测试代码。只要每个业务模块都被单元测试覆盖，则整体的测试覆盖率也会很高，同时成本也相对较低。

参考

- Unit testing - Wikipedia
- List of unit testing frameworks - Wikipedia

代码评审

代码评审（Code Review）是保证代码内部质量的一种重要手段。评审人员可以是特定的资深 Dev 和技术负责人，也可以是同一个团队中的其他程序员。评审的内容包括代码的设计、功能、复杂度、命名、风格、测试等方面。评审的标准包括代码设计是否合理和精良、代码是否高效和安全、代码复杂度是否高、代码的命名和风格是否符合团队的要求、代码是否有足够和合理的测试等。

代码评审可以根据不同场景分为结对编程中的一对一代码评审、分布式团队中的在线远程专人代码评审，以及基于整个团队的团队代码评审。

为什么要代码评审?

代码编写是一项容易产生错误的工作，即使是经验丰富的程序员也不能保证第一次编写的代码完全正确且没有缺陷。随着软件规模的扩大，软件系统的开发者数量也大幅增加，他们的能力参差不齐，导致开发的软件系统问题数量也大量增加。然而，通过代码评审，可以尽早发现很多问题，从而大大减少修复的周期和成本，减少技术债的引入，提高软件的质量。

代码评审有助于尽早发现缺陷。根据卡珀斯 · 琼斯（Capers Jones）的两篇文章 Embedded Software: Facts, Figures, and Future 和 Measuring Defect Potentials and Defect Removal Efficiency，他分析了超过 12,000 个软件开发项目。在使用正式代码评审的项目中，潜在缺陷发现率约为 60%-65%。如果是非正式的代码评审，则潜在缺陷发现率不到 50%。对于大多数测试，潜在缺陷发现率会在 30% 左右。

其次，代码评审可以高效地将代码相关的各种知识共享给团队中的其他人，特别是还可以将团队代码评审的方式共享给整个团队的开发人员。这样，在原始编写代码的人员请假或者离职后，其他人可以快速接手进行开发和维护，从而降低项目风险。

常见误区

1. **项目已经成功地实施了结对编程，因此团队代码评审可以不必进行。**结对编程中的代码评审是一对一的、快速持续的代码评审。而团队代码评审更重要的作用是让整个团队的开发人员共享代码，它们的主要作用不同。因此，即使结对编程得到了很好的实施，如果想让整个团队的开发人员都有能力维护整个软件系统，也需要进行团队代码评审。

2. **代码评审只能由人工进行。**目前，常规的代码评审大多通过人工进行。然而，现在已经有大量的代码静态扫描软件可以对代码进行自动化评审。但由于相关技术的限制，自动化代码评审只能发现一些通用问题，例如内存，逻辑，安全、性能、复杂度等问题，而对于业务功能、架构设计等问题则很难进行评审。因此，好的代码评审是由人工和自动化相辅相成、共同实施的。
3. **不需要过于频繁地进行代码评审。**有些团队认为一周或更长的时间做一次代码评审可以节约时间。其实，代码评审应该像持续集成一样频繁进行，以便及时发现问题并修复。随着时间推移，问题会越来越难以发现，修复成本也会越来越高。建议每天进行团队级别的代码评审，以便及时发现问题并修复，预防缺陷进入测试阶段，节约成本。

参考

- Embedded Software: Facts, Figures, and Future
- Measuring Defect Potentials and Defect Removal Efficiency

Desk Check

Desk Check 属于用户故事（或需求条目）生命周期中的一个环节，与 Kick off 相对应。它指的是 Dev 在开发完某个故事（条目）卡之后，BA、Dev、QA 以及 UX 等故事开发相关角色对该用户故事进行快速验证确认。通常是在 Dev 的座位上进行，由 Dev 利用开发机器给所有人演示，这也是名为 Desk Check 的原因。

Desk Check 的内容包括功能、性能、安全、UI 布局等，QA 还会评审单元测试，有的团队还会对日志记录情况进行验收。

Desk Check 有多个名字，根据团队的习惯也可以叫做结卡、验卡、迷你 Showcase 等。

为什么要做 Desk Check？

Desk Check 的主要目的是尽快验证 Dev 开发的软件是否实现了用户故事中所有的 AC，且没有明显缺陷，以便尽早获得反馈，减少可能的返工和修复成本。

相关主题

- Kick off
- 单元测试评审
- 日志评审

常见误区

1. **Desk Check 需要对所有情况进行详细的检查。**最常见的误区是认为 Desk Check 检查得越详细越好，对所有正常 / 异常路径和边角 Case 都要检查一遍，以及陷入细节的讨论，导致每次 Desk Check 耗时一两个小时。其实，这是没有必要的。一般只需要对所有验收标准（AC）和典型异常路径进行验证。正常情况下，单个用户故事的 Desk Check 时间一般控制在 15 分钟左右，最多不要超过半个小时，才是比较合理的。

参考

- 高效用户故事验收

单元测试评审

单元测试评审是 QA 和 Dev 一起对 Dev 所写的单元测试进行检查，以确保其有效性。评审内容包括测试覆盖情况、分层是否合适、是否有冗余或遗漏的测试、测试命名是否有意义，以及是否包含有效的断言来验证内容等。

为什么要评审单元测试?

由于 Dev 和 QA 的思维方式和测试技能存在差异,Dev 编写的测试难免会有考虑不周的情况，因此单元测试的有效性难以保障。同时，QA 不参与单元测试相关工作，对 Dev 编写的测试缺乏感受，不利于测试分层策略的综合考虑，很难决定端到端的测试需要覆盖哪些场景，以及哪些场景在底层已经覆盖到而无需重复测试。单元测试评审环节主要是由 Dev 和 QA 合作，取长补短，利用双方的优势以获取更大的价值。

单元测试评审的价值体现在以下几个方面：

1. 利用 QA 的测试技能，可以发现 Dev 编写的单元测试可能存在的问题，让测试更有效。
2. QA 通过评审单元测试，能够更好地了解测试覆盖情况，更清楚整体的测试状态。
3. QA 评审单元测试可以督促 Dev 尽可能写好测试。
4. 增强 Dev 和 QA 的协作，有利于提高 QA 的代码理解能力和 Dev 的测试能力。

相关主题

- 单元测试
- Desk Check

常见误区

1. **Dev 认为评审是 QA 的工作。**Dev 打开 IDE，让 QA 自己看一遍测试，可能 QA 会不懂或不敢问，而 Dev 在 QA 评审过程中也没有积极地收集反馈。这样不仅没有价值，还会浪费大家的时间。建议对于初级 QA 参与的评审，Dev 可能需要相对详细地介绍每个测试；而当资深 QA 或者初级 QA 熟练以后，可以更快速地演示一遍即可。
2. **QA 认为必须评审每个测试的每个细节。**有些 QA 在评审过程中对每个测试的实现细节都要详细过一遍，每次需要一个小时以上，导致团队产生厌恶情绪，实践难以坚持。正确的做法应该是根据 Dev 和 QA 的能力对评审过程进行调整，比如相对资深的 Dev 所写的测试有效性会较高，评审过程尽量轻量级，简单查看有哪些测试即可；对于初级 Dev 所写的测试，可能需要查看断言的有效性等。
3. **单元测试评审只需要走一遍流程即可。**单元测试评审变成机械的形式，不管有用没用，机械地执行评审，久而久之会听到大家的声音，觉得这个实践没有意义。单元测试评审跟任何其他实践一样，需要定期回顾并持续优化。当发现存在问题时，团队应该一起积极分析根因，并找出对应的解决办法。

参考

- QA 评审底层测试的价值

日志评审

日志评审是 QA 和 Dev 一起对 Dev 所记录的日志进行检查，以确保日志记录的有效性和安全性。评审应包括日志规范、记录点是否有遗漏、记录信息是否齐全，以及是否暴露隐私信息等方面。

为什么要评审日志?

由于 Dev 和 QA 对日志作用和业务功能理解的差异，Dev 所记录的日志难免存在不满足实际日志需求的情况，导致日志记录的有效性难以保障。因此，QA 利用自己对系统、用户行为和测试右移中的日志需求的了解，对日志记录的有效性和安全性进行检查非常必要。

日志评审的价值体现在这几个方面：

1. 最重要的是利用 QA 的经验和技能发现 Dev 所记录的日志可能存在的问题，让日志记录更有效。
2. 通过评审日志，QA 能更清晰地了解记录日志的状况，从而增强利用日志分析缺陷的能力。
3. 通过 QA 对日志的评审，可以督促 Dev 记录关键日志并达到有效记录的效果。
4. 增强 Dev 和 QA 的协作，双方取长补短，有利于发挥各自更大的价值。

常见误区

1. **Dev 认为日志评审是 QA 的事情。**日志评审是需要 Dev 和 QA 协作完成的实践，通常需要 Dev 给 QA 演示。对于初级 QA 参与的评审，建议 Dev 需要清楚地介绍日志记录规范以及每条日志记录的内容。而对于资深 QA 或者初级 QA 已经熟练的情况下，快速地演示一遍日志功能即可。

2. **QA 认为需要评审日志记录的每个细节。**如果对于详细评审日志的每个细节都进行审核，通常会占用较长时间，最终导致团队产生厌恶情绪。建议根据 Dev 的资深程度以及以往日志评审的情况，有针对性地调整对于不同 Dev 所记录日志的评审详细程度，以尽量缩短评审时长。
3. **认为日志评审只需要按流程进行一遍即可。**日志评审和其他实践一样，不能简单地执行一遍流程，需要定期回顾并持续优化。当发现存在问题时，团队应积极地一起分析根本原因，并找出相应的解决方法。

代码覆盖率

代码覆盖率是一种计算测试执行过程中有多少源代码被执行的比率和程度的方法，属于对软件测试和质量的结果度量。通常，它根据软件系统代码的内部结构来设计和实施测试，因此属于白盒测试的一部分。

常见的覆盖率指标包括方法覆盖率、行覆盖率、分支覆盖率等。同时常见的编程语言都有相应的代码覆盖率工具。例如，基于 Java 语言的代码覆盖率工具 JaCoCo 提供了自身的代码覆盖率统计。

为什么统计代码覆盖率?

代码覆盖率是一个非常重要的测试和质量指标。

1. 它可以有效地统计各种类型的测试覆盖了多少代码，从而帮助 Dev 更有信心地进行重构和修改。代码覆盖率越高，在重构和修改之后，就越有可能找到更多被影响的代码模块，帮助 Dev 判断自己的重构和修改是否破坏了已有的功能。

2. 它还可以展示软件系统中有多少代码没有被测试到，从而有效地帮助 QA 对已有的测试用例进行查漏补缺，提高整体测试工作的有效性和测试覆盖率。
3. 它还可以帮助管理者对测试工作进行度量，如果覆盖率高，可以增加管理者的质量信心。
4. 代码覆盖率还是精准测试的基础指标之一。如果代码覆盖率不高，就无法实施精准测试。

常见误区

1. **只有单元测试才能统计代码覆盖率。**除了单元测试，常规的界面功能测试和 API 接口测试也可以统计代码覆盖率，但是一般不把 UI 测试的代码覆盖率作为度量指标。
2. **100% 的代码覆盖率就没有缺陷。**实际上，即使代码覆盖率是 100%，也只能证明每个具体的函数 / 模块本身的逻辑是正确的，但无法保证所有的函数或模块在集成时仍然是正确的。
3. **代码覆盖率体现了业务场景覆盖率。**代码覆盖率主要是指代码、函数等在相应的测试执行时是否被覆盖到，而无法体现业务场景的覆盖率。因此，仍然需要通过功能级别的测试来体现业务场景的覆盖率。

集成测试

集成测试，也叫做组装测试或联合测试。它通常是在单元测试的基础上进行的，即在保证各个单元独立正常工作的情况下，将所有或部分单元集成为子系统或系统，针对集成后的系统进行测试。

按照集成范围的不同，通常可分为系统内集成和系统间集成。系统内集成指将系统内的各单元集成起来，对系统对外提供的接口进行正确性检验和测试。系统间集成则指在不同系统之间，将有集成功能的部分进行集成，然后针对集成功能进行集成测试。

为什么进行集成测试?

防止集成缺陷逃逸

由于系统的功能具有涌现的特征，经过大量实践表明，一些模块虽然能够单独工作，但不能保证连接起来也能正常工作。一些局部反映不出来的问题，在全局上很可能暴露出来。因此集成测试可以防止集成相关的缺陷逃逸至生产环境。

关注集成相关的跨功能特性

由于集成系统的跨功能特性无法在独立单元内验证，因此在集成测试过程中，除了功能特性，我们也需要关注集成相关的跨功能特性，如性能、安全、数据一致性等。

尽早发现集成风险

通常，系统间的集成具有较高的复杂性。通过集成测试，可以尽早发现集成风险，避免集成缺陷在较晚的时间点暴露，造成难以干预的后果。

相关主题

- 单元测试
- 系统测试

常见误区

1. **如果单元测试已经通过了，那么就可以不用再进行集成测试了。**单元测试和集成测试各有不同的目标，不能因为单元测试已经通过就不进行相应的集成测试。通常情况下，这两种测试会存在一定的重叠。单元测试旨在验证单元本身的功能，而集成测试则更关注于集成视角。两种测试的结果不能互相替代。
2. **可以不在迭代内进行充分测试，等集成时机到了再进行集成测试。**为了更早地发现缺陷，我们鼓励测试往底层下沉，即在刚好能够测试的时候进行测试，而不是等待后面的集成测试。集成测试通常时间紧任务急，测试量也大，需要协调的资源也较多。在进行集成测试之前，应尽可能地发现并修复非集成性缺陷，这有助于顺利完成集成测试。

参考

- 系统集成测试的断舍离
- 集成血案启示录

端到端测试

这里提到的端到端测试（End-to-End Testing）是基于用户视角的端到端测试，不是一种严格意义的测试方法。它不限制端到端测试所采用的方法和工具，而更多强调端到端的用户视角。因此，它是一种贴近真实用户使用场景的测试思路。

端到端测试通常强调要完整测试整个软件产品，以确保软件行为按用户预期运行。

测试执行者有时需要针对特定业务场景进行端到端测试。虽然不涵盖完整的业务流程，但也可以认为这些测试属于局部业务的端到端测试。

为什么进行端到端测试？

模拟真实用户的使用场景

在较长的业务流程中，真实用户的使用场景跨越整个业务周期。在开发测试过程中，往往过于关注当前的实现，容易陷入局部而失去全局视角。端到端测试有助于帮助团队更好地建立基于用户视角的业务优先级和全局观。

关注用户体验和非功能特性

用户视角的端到端测试在测试过程中更容易发现非功能缺陷和用户体验相关的问题。有时候，质量评价的负面影响并不是由于软件缺陷造成的，而是由于用户体验或业务流程的割裂所致。

提升测试投入产出比

通常，端到端测试需要描述用户使用软件的关键路径和场景。在测试资源有限的情况下，执行高优先级的端到端测试可以在测试用例较少的情况下最大程度保证业务正常运行。

相关主题

- 单元测试
- 集成测试

常见误区

1. **端到端测试只能手动执行。**对于相对稳定的功能，可以自动化执行其端到端测试。
2. **端到端测试必须涉及完整的业务链路。**根据执行者所处的业务场景不同，如果测试验证了范围内的完整链路，则在当前上下文内，也可以认为这是一种端到端测试。

探索式测试

探索式测试是一种测试风格，主要强调个人的自由与责任，让独立的 QA 可以通过相关的学习、测试设计和测试执行等活动来持续地改善测试工作的质量。它最主要的方式就是一边做测试，一边进行分析和探索，从而发现新的测试场景，然后再对新的测试场景进行测试，如此往复直到完成足够的测试。

为什么需要探索式测试?

探索式测试的提出主要目的是解决脚本化测试的僵化和墨守成规、成本高以及难以发现更多隐秘的缺陷等问题。因此，探索式测试的优点包括：

- 节约测试成本，以最少的资源投入发现更多的缺陷和问题，从而实现快速测试并获得高的测试投资回报比；

- 具有创造性和灵活性，可以发现更多隐蔽的缺陷，例如通过一些复杂或不常用的方式才能产生的缺陷；
- 还可以轻松了解系统的易用性等。

当然，它也有一些缺点：很难重复，并且无法对系统进行系统的全面测试；有限的测试可信度；高度依赖于测试执行者的知识与经验；不适用于要执行很长时间的测试等。

虽然探索式测试有以上这些缺点，但是它的优点更为有价值。在高质量需求的项目中，一般都需要实施探索式测试作为常规测试的辅助，以尽可能发现更多的缺陷，从而进一步提升系统质量。

常见误区

1. **探索式测试没有任何规则和方法，随机乱测。**其实探索式测试是有相应的制度和方法的，比如使用 timebox，使用各种测试基础方法论等。
2. **探索式测试无需设计测试用例，也不需要生成测试用例。**实际上，探索式测试是一边分析，一边设计，一边执行测试，然后在执行完毕后需要产出测试用例。例如，如果发现了一个缺陷，相应的测试用例就需要固化下来成为回归测试用例的一部分。
3. **探索式测试一定要遵循固定的复杂流程和模型。**虽然探索式测试是一种测试风格，但有些人试图通过一种固定的、复杂的流程和模型来实施。然而，这样做往往会失去探索式测试的灵活性，导致无法达到探索式测试的目的。但是探索式测试也不能没有任何流程和模型，仍然需要一定的简单流程和模型来规范测试，并给予 QA 最大的灵活性，从而真正实施好探索式测试。

非功能测试

非功能测试主要测试软件系统中非功能性需求。与功能需求相对应，非功能需求是指为了满足客户业务需求而需要符合，但又不在功能性需求以内的特性；或者软件系统是否满足某些特定标准，如美国政府的无障碍标准 Section 508 legislation，W3C 组织的 Web 内容无障碍指南 (WCAG)，欧洲的 PII 标准通用数据保护条例 (GDPR) 和美国的 PII 标准加利福尼亚消费者隐私法案 (CCPA) 等。

非功能测试的主要测试方面有：

- 兼容性测试：验证系统能否在不同环境或者不同配置等情况下正常运行；
- 性能测试：验证系统的性能和稳定性等；
- 安全测试：验证系统是否存在相关的安全漏洞；
- 易用性测试：验证软件是否易于最终用户使用；
- 无障碍测试：验证系统是否符合相应的无障碍标准。

为什么需要非功能测试?

一般情况下，功能测试只是正常的环境中验证用户能否正常完成软件的各种业务功能。但它不能保证软件在高并发请求或长时间使用的情况下仍然具有足够好的性能和稳定性，也不能验证软件是否存在安全问题，当然也不能保证软件是否符合各国的 Accessibility 标准等特定标准。一个成功的高质量软件除了要求功能正确外，还需要保证在真实的各种情况下能正常运行。非功能测试就是尝试验证在真实的各种情况下，软件系统是否能正确运行，或软件是否符合某些特定标准的要求。

相关主题

- 持续测试
- 度量可视化
- 质量报告
- 质量度量

常见误区

1. **在开发完成后再开始非功能测试。**很多团队常常在开发完毕后才开始关注性能和安全等非功能性测试。然而，非功能性测试应当尽早展开，即在需求分析阶段就开始收集相关的非功能需求，并在设计与开发中融入这些需求的实现。
2. **只要有工具，就可以完成非功能测试。**尽管许多非功能性测试确实需要工具支持，但过分依赖这些工具可能会使我们忽视某些关键领域。例如，与产品紧密相关的非功能性需求的收集、有针对性的非功能性测试策略的制定，以及在安全测试中那些与业务紧密相关且工具无法解决的部分。
3. **测试环境的结果可以直接等比放大到生产环境。**测试环境与生产环境之间的区别对测试结果有着重大影响，而测试结果与实际情况的出入并不总是等比的。特别在性能测试方面，若仅仅以为可以按照某一比例缩减规模进行测试，然后再按相同比例将结果放大到生产环境，这样的认识是极为误导的。
4. **只要非功能测试没问题，生产环境上就不会出问题。**无论非功能测试进行得多么详尽，它仍然只能反映测试环境在当时的状况。由于真实生产环境的复杂性，其非功能状态很难预测。因此，对生产环境中的非功能进行监控和预警是极为关键的。

持续集成中的自动化测试

在敏捷开发中，持续集成是必不可少的基础实践。它的核心在于通过快速构建系统发现开发过程中的代码和配置等错误，并通过自动化测试快速检测新代码的功能是否正确。这使得开发者可以使用小步快速迭代的方式开发系统。由于持续集成需要频繁进行，因此必须使用自动化测试，以保证速度和一致性。此外，持续集成需要保证每次测试的执行都是完全相同的，这只能通过自动化测试来实现，因为人工测试难以在大规模重复执行时保证完全一致。

持续集成中有哪些自动化测试?

在持续集成流水线中，从代码到产品构建，再到部署上线的工作，基本上都可以自动化完成。其中，软件测试是代码开发、产品构建、部署上线阶段不可或缺的工作。因此，持续集成流水线应该包括必要的各类自动化测试，例如单元测试、组件测试、集成测试、接口测试、界面测试、性能测试和安全测试等。

常见误区

1. **所有的自动化测试都需要加入到持续集成流水线中。**持续集成流水线一般需要针对持续提交的代码进行构建和测试，因此每次流水线的执行时间不能太长，最好控制在 1 小时以内。并且，一些需要时间很长的自动化测试（如全量回归测试、全链路性能测试等）或者用途特定的非回归类测试（如模糊测试、变异测试等），都应该单独执行，而不是加入到持续集成流水线中。

2. **在持续集成流水线中没有加入任何自动化测试。**在许多团队中，为了追求持续集成流水线的速度，通常没有加入任何类型的自动化测试。这导致通过流水线构建出来的软件系统可能存在大量问题，严重影响后续的测试工作和交付进度。为了确保持续集成并构建出可靠的软件系统，持续集成流水线一定要加入必要的自动化测试，以快速获取反馈，尽早发现问题，减少修复成本。如果追求持续发布，还需要加入足够数量的自动化测试，以确保流水线构建出来的软件系统质量满足发布要求。

验收测试

验收测试是软件产品完成单元测试、集成测试和系统测试后，在产品发布前进行的软件测试活动。验收测试的目的是确保软件准备就绪，可以让最终用户使用并执行软件的既定功能和任务。

根据最佳实践，验收测试最好由使用软件的直接用户来进行，这样能够最真实地以用户视角进行验收。实际情况是，不同场景下的验收测试并不都是由用户来执行。有时也会由能够代表真实用户的业务方进行验收测试。验收测试的执行时机也不尽相同。有些会在产品开发的关键节点进行，例如开发完成、集成完成等节点。有些会在上线前进行统一验收，也有时会在迭代完成时进行。

关于验收测试的执行者、执行时机，不同团队可以根据自己的条件和限制酌情实施。

为什么进行验收测试?

防止用户视角的缺陷逃逸到生产环境。

验收测试也是测试的一种。在此阶段发现缺陷并将其修复，可以有效地预防这些缺陷逃逸至生产环境。

从用户的角度验证需求价值的实现程度。

验收测试主要立足于用户视角，能够尽可能真实地模拟用户的使用场景。它更多地从用户使用角度出发，而非软件实现角度。这种具备用户思维的测试方式可以更有效地验证需求价值是否被正确实现，从而更好地满足用户的需求。

常见误区

1. **验收测试都应该由用户来进行。**验收测试可以由用户来测，也可以由能够代表用户的或具备充分用户视角的其他角色来进行测试，并不一定非得由真实用户进行。
2. **已经进行大量充分的测试，没必要再进行验收测试。**在软件研发生命周期的各个阶段，都会进行各种程度不一、重点不一的测试活动，不同的测试活动目的不同。前期的大量测试更多的是在验证功能实现，哪怕已经进行了充分的测试，验收测试也是有必要进行的，因为验收测试是基于用户视角来验证需求价值的实现，与前面的各种测试有着目标、内容和重点的区别，并不能互相取代。

参考

- 验收测试
- 我们为什么需要用户验收测试

测试用例管理

在软件测试工作中，测试用例是极其重要的基础。它不仅是测试工作的内容来源，更是测试工作主要的产出物和评估标准之一，也是软件项目的交付资产之一。因此，测试用例的管理显得尤为重要。

测试用例的管理流程包括：

- 测试用例分析与设计，通过分析软件架构和业务需求等信息，采用各种测试设计方法来设计测试用例的过程。
- 编写测试用例，使用适当的测试用例管理工具和领域专用语言（DSL）等完成测试用例编写工作，并能高效地查询、修改和删除测试用例。
- 执行测试用例，使用各种测试辅助工具或自动化测试框架，以手动或自动化的方式执行测试用例，并有效地记录测试结果。
- 测试结果分析与测试用例维护，通过各种分析方法和分析工具分析测试结果，了解测试执行的稳定性和结果的有效性，并通过分析结果来维护测试用例。

此外，根据测试目的，测试用例还可以分为回归测试用例、验收测试用例、健全测试用例和冒烟测试用例等。当一个测试用例既属于回归测试，又属于冒烟测试时，需要有一个优秀的测试管理工具或者系统来对分类后的测试用例进行管理。

为什么需要测试用例管理?

在一个大中型项目中，测试用例的管理往往存在以下问题：

1. 大量的测试用例往往需要多人协作编写、阅读和维护，因此如何高效的协作是一个首先需要解决的问题。

2. 由于大型项目的维护周期一般都比较长，因此如何有效组织、使用、管理和传递测试用例给后续的维护团队和人员，也是一个困难的问题。

测试用例是项目资产的一部分，有效地分析和可视化测试用例及其结果非常重要。

测试用例管理的结果应该是我们可以高效编写、执行和维护易于阅读和理解的测试用例，并且能管理测试结果和测试集合等。这样就可以在软件开发全生命周期中更高效地实施测试工作，最终直接提升整个软件开发的效能，事半功倍。

相关主题

- 单元测试
- 端到端测试
- 验收测试
- 集成测试

常见误区

1. **测试用例只能手动执行。**经常有人误解测试用例只能手动执行。其实测试用例可以采用手动和自动化执行两种方式，各有优缺点。高效的项目通常会同时包含这两种执行方式。
2. **手动和自动化测试用例要分开管理。**现在很多项目都是把手动和自动化测试用例分开维护，其实我们可以通过活文档等方法来统一管理手动和自动化测试用例，从而大大节约管理测试用例的时间。

参考

- 测试用例编写和管理

缺陷分析

在软件的开发与测试过程中，必然会产生软件缺陷。缺陷分析就是通过技术或人工的手段，将缺陷数据收集起来，进行数据统计和深入的分析，以发现缺陷相关的洞察，从而指导团队持续改进质量。

通常，缺陷分析包含两大部分内容，一是缺陷的统计与趋势分析，二是缺陷的根因分析。缺陷的统计和趋势分析与观察周期有关，可分析当前阶段的缺陷分布，用于明确周期内的缺陷状态，也可以分析不同阶段内缺陷分布的变化趋势。主要是服务于观察缺陷的提交与修复、缺陷集中情况等数量层面的变化趋势。缺陷根因分析是就某个或某类缺陷产生的背景、时机、根本原因进行充分的分析，并得出预防该缺陷或者该类缺陷的方法或途径。

为什么进行缺陷分析?

明确当前的质量状态

缺陷数据在一定程度上揭示了当前软件版本的质量状态。通过观察缺陷数量的变化趋势、不同功能的缺陷密度等数据，可以清晰地感知软件的质量趋势和质量薄弱的功能等信息，从而帮助团队理解质量状态。

了解测试设计的有效性

那些缺陷比较集中的功能或服务，或者某几类频发发生的缺陷，即为测试设计相对缺失的部分。通过观察缺陷数据，可以指导团队更有针对性地进行测试设计。

评估缺陷修复进展

缺陷发现了是为了解决它，通过缺陷统计分析，可以得知目前的缺陷总量、不同状态的缺陷量以及缺陷在不同状态上的停留时长。这可以帮助团队分析目前缺陷的移除效率如何，便于进一步评估测试完成的时机，及时干预受阻塞的缺陷任务。

有助于缺陷预防，提升代码质量，指导团队持续改进

通过缺陷的根因分析，可得出此类缺陷的预防方法，找到根因后能从根本上预防缺陷的重复产生。Dev 能够了解到缺陷是如何产生、如何捕捉和处理的，在未来的开发过程中，有助于提升开发质量。了解到缺陷根因后，团队会一起商议缺陷预防的措施，在后续的持续交付过程中进行预防措施的实施，这有助于指导团队进行持续改进。

相关主题

- 缺陷管理

常见误区

1. **发现缺陷数量越多，越能保证软件质量。**发现缺陷数量越多，并不一定意味着软件质量越好，反而说明软件的开发质量越差。在集中测试阶段发现缺陷越多，意味着前期预防到的缺陷越少，导致团队不得不在提测后大量返工修复缺陷。
2. **缺陷统计分析费时费力，对质量提升没有用处。**缺陷统计可充分复用既有工具，尽量采取相对自动化的方式进行，但分析工作较难通过自动化实现。通常认为缺陷统计分析对质量提升无用，大部分原因是缺乏有效的分析和跟进。

3. **所有的缺陷都应该进行充分的根因分析。**缺陷的根因分析是一项消耗成本的质量活动，并不是所有的缺陷都值得被详细分析，但所有的缺陷都应该找到根因，是不是需要进一步分析，需要视缺陷提供的价值大小、影响程度和团队的资源情况综合考量。
4. **缺陷根因分析只需要获得根因并让团队知道，不需要后续的活动。**缺陷的根因分析最终应该落到实践中，是大家能够执行的具体实践，并给改进实践制定责任人和观察实践，定期的评价改进效果，这是使缺陷根因分析产生积极效果的有效途径。如果缺陷的根因分析最终产出的实践归为人的主观能动性，或是外部依赖难以改进，那么就会流于形式。
5. **缺陷根因分析是追责的行为。**缺陷的根因分析是为了找到根因，更好地预防缺陷产生，而非确定责任人。团队在进行根因分析时也应注意引导，避免追责情况产生。

参考

- 质量内建——缺陷管理实践分享
- Bug Report 该怎么做?
- 软件缺陷的有效管理
- 缺陷分析如何帮助质量内建

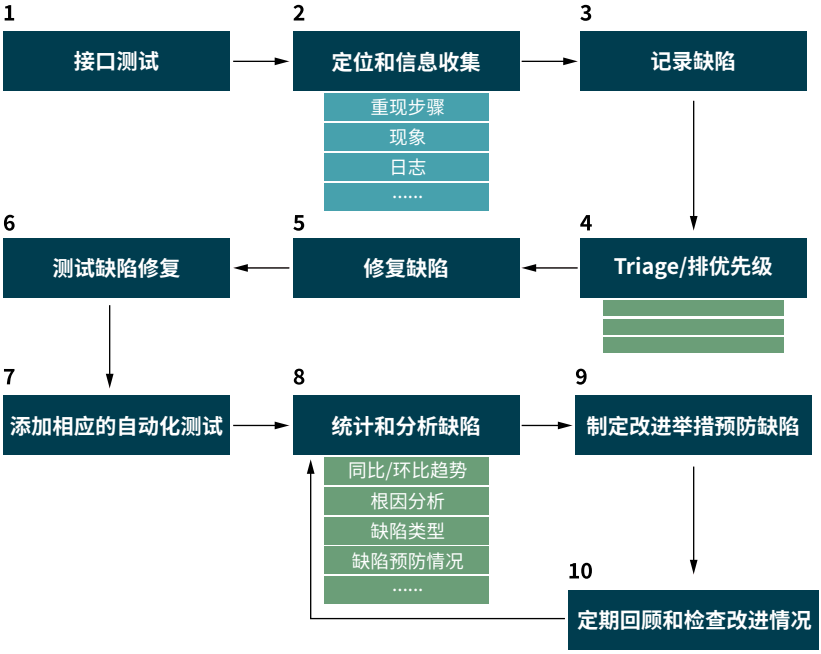
缺陷管理

缺陷管理是在软件生命周期中识别、管理、沟通、分析和总结任何缺陷的过程，将缺陷作为一项重要的软件资产进行跟踪管理，避免或减少同类的缺陷再次发生。

对缺陷的整个生命周期进行闭环管理，通常要经历以下十个环节：

1. 发现缺陷：通过自动化和手动测试、真实使用、监控等方式发现系统中的缺陷。
2. 诊断定位和信息收集：通过在开发或测试环境重试，或查看代码的方式对缺陷进行诊断定位，同时收集更多缺陷相关信息。
3. 记录缺陷：按照团队达成一致的缺陷模板，在管理系统中记录缺陷相关信息。
4. 分类和排列优先级：将缺陷按照有效性或业务模块的关联情况进行分类，排除无效缺陷，并对有效缺陷进行优先级排序。
5. 修复必要的缺陷：对缺陷进行修复，可以是通过修改代码或者修改配置等方式。
6. 测试修复情况：对修复的缺陷进行测试验证，确保已经修复。
7. 对有必要的缺陷添加相应的自动化测试：对刚修复的缺陷增加自动化测试保障，以免同样的缺陷再次出现。
8. 统计和分析：对缺陷数据进行统计，分析缺陷趋势和根本原因。
9. 根据统计和分析的结果，制定改进措施，以防止类似缺陷的重复出现。
10. 定期检查改进情况，跟进已采取的措施，确保真正地落实执行，以实现缺陷管理的闭环。

缺陷生命周期如下图所示：



缺陷生命周期

为什么要管理缺陷?

好的缺陷管理能够为项目或团队带来重要的价值：

1. 通过缺陷管理可以有效地跟踪缺陷，并将其作为一项资产供参考；
2. 缺陷可以反映出软件交付的过程质量和成效质量；
3. 缺陷统计分析有助于发现质量问题和质量风险；
4. 通过有效的缺陷管理和分析，并将结果可视化给团队，可以加强团队的质量意识，改进软件交付流程和协作方式，提高团队的质量能力。

相关主题

- 缺陷分析

常见误区

1. **缺陷被修复后，关闭它就结束了。**缺陷生命周期在关闭之后仍需要进行跟踪、统计和分析，以最终预防同类问题再次发生。
2. **只有 QA 需要关心缺陷相关数据，其他角色不需要关心。**实际上，缺陷数据需要向整个团队进行可视化展示，所有角色都应该密切关注。
3. **再小的缺陷也需要记录管理。**对正在开发的功能所暴露出来的问题也都进行逐一记录，这属于过度的缺陷管理。这种缺陷管理方式太过繁琐，往往弊大于利。对于能够及时修复的非典型缺陷，建议直接修复，无需在系统记录浪费管理成本。
4. **所有的缺陷都应该被修复。**所有缺陷都应该被处理，但不一定都需要被修复。有价值的缺陷推荐修复，没有价值、无效或修复回报较低的缺陷可以忽略，不必浪费过多的开发成本进行修复。同时，对缺陷的记录也有一定要求，鼓励记录那些具有价值且值得修复的有效缺陷。

参考

- 软件缺陷的有效管理
- 缺陷分析如何帮助质量内建
- 都是脏数据惹的祸

质量实践：上线与运维阶段

线上回归测试

线上回归测试是指当软件部署到生产环境之后，在不破坏生产真实用户数据的情况下，对系统的关键功能和严重缺陷的修复情况进行测试。

为什么要线上回归?

无论进行多么完备的测试，上线后的回归测试都是必要的。这是因为线上环境的基础设施通常比测试环境更真实和更复杂。

- 由于服务器和其他服务的物理配置不同，会出现兼容性问题。例如，一些第三方服务的版本过高或过低都会影响系统的运行。
- 软件配置信息包括文件配置、服务器代理转发配置、数据库配置、中间件配置等，可能与测试环境不同，也很容易产生问题。

相关主题

- 测试右移

常见误区

1. **无法在生产环境数据中进行测试，因为这可能会破坏数据。**可以在不影响真实用户体验的情况下，对某些功能进行只读操作。需要选择用户使用频率较低的时间段进行，以避免影响系统性能。请勿进行大数据量的只读操作。对于某些特定的特性，如果方便隔离，可以在生产环境中使用独立的测试数据进行测试。
2. **线上回归测试只能手动进行，无法实现自动化。**线上回归测试关注的是测试的内容和时机，而不是测试执行的方式。手动和自动化执行线上回归测试都是可行的。自动化执行测试可以更快速，减少测试对生产环境的影响时长，因此比手动方式更值得推荐。

参考

- QA in Production
- 测试右移——生产环境下的 QA

线上监控

线上监控是测试右移的关键实践之一。它是指对生产环境的质量状态进行信息收集和监控，包括基础设施使用情况、系统运行状态和用户行为等。通过将结果可视化呈现，需要及时响应的异常情况将以警报的方式通知相关人员，以作出及时响应。

通常情况下，可以利用基础设施工具自身提供的监控功能、日志管理工具或网站分析工具等来进行监控和报警。

建议让具有不同角色的与质量相关人员共同参与线上监控。结合业务、技术实现和测试的需求，制定监控策略，包括监控内容、预警方式和响应机制等。

为什么要进行线上监控?

由于业务、数据和基础设施的复杂性，生产环境的系统行为存在很多不确定因素。这些因素无法在测试环境或预生产环境中进行一一验证，因此利用生产环境的数据进行监控至关重要。

线上监控的价值主要有以下几个方面：

1. 预警功能异常，通过监控，可以提前发现系统出现的功能异常，并及时采取相应的响应措施，尽可能减少对最终用户的影响。
2. 展示性能趋势，通过收集请求的响应状态、时间等信息，可以分析得出系统的性能状态。如果对一段时间内的性能状态进行持续的分析监控，就可以展示性能趋势变化。
3. 暴露安全隐患，为了暴露严重的业务安全问题，比如数据被越权访问的情况，根据业务特点设置相关的监控，记录数据被访问的情况。
4. 优化业务价值，通过线上收集到的信息，可以分析出哪些业务模块或业务流程被访问较多或较少，以及用户的行为习惯。通过这些分析，可以进一步调整业务结构，优化业务价值。

相关主题

- 测试右移
- 日志评审

常见误区

1. **线上监控仅仅是 Ops 团队的职责。**线上监控需要结合 BA、Dev、QA 和 Ops 等多个角色的经验和技能，综合设定监控的内容、报警的方式，以及对于报警的响应机制等。这是需要多方协作完成的实践。

2. **只需要监控基础设施等资源的使用情况即可。**确实需要监控基础设施等生产环境资源的使用情况，这非常关键。但除此之外，对于系统行为和用户行为的监控也非常关键，不仅要监控系统异常，还要监控用户的使用流程和使用频率等，这些监控也非常有价值。
3. **只需要监控，无需进行自动报警。**监控是对整体的状态和行为进行监控。对于需要紧急处理的异常情况，务必进行自动报警。同时，不同紧急程度需要有不同的响应机制，以便及时响应并处理。

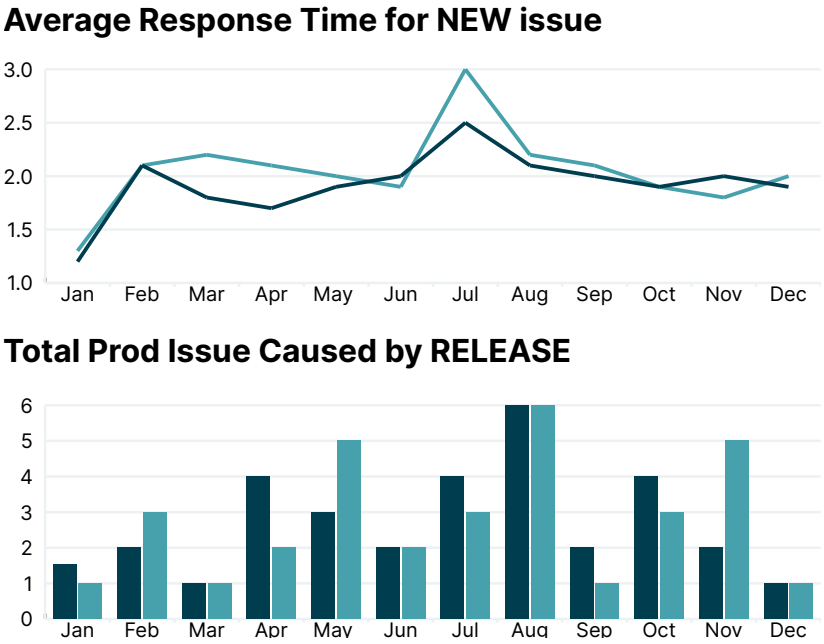
参考

- 测试右移——生产环境下的 QA
- 测试右移：QA 与 Ops 通力合作打造反脆弱的软件系统
- 测试右移之日志收集与监控

质量实践：过程质量

质量度量可视化

软件质量度量可视化是一种将软件质量度量以图形方式展示的方法，可以采用各种图表和图形，如折线图、饼图和柱状图等。这些图表可以显示各种度量指标，如代码覆盖率、缺陷密度和性能指标。



示例 - 缺陷数据可视化

为什么需要质量度量可视化?

质量度量可视化有助于帮助团队整体感知质量状态、评价质量活动成效、预测质量风险，以及在团队质量改进的过程中提供支持。

在这个过程中，通过共享和讨论可视化的数据和指标，团队成员可以更好地理解和解释软件质量的情况，促进有效的讨论和决策，增强团队之间的协作和合作能力。同时，可视化的质量报告也可以作为其他软件开发生命周期活动的有力输入，例如产品相关决策、用户洞察、体验设计等。

相关主题

- 由于质量度量可以贯穿整个软件开发生命周期，不同团队在不同阶段需要度量和可视化的内容也不尽相同。因此，所有与质量度量相关的实践，都可以是可视化的相关主题，如单元测试、集成测试、端到端测试、非功能测试、验收测试、探索式测试、缺陷分析、持续集成。

质量报告

软件质量报告揭示了软件在一定周期内的质量状态，常见的有迭代质量报告、版本质量报告。通常，质量报告包括测试时间、测试内容、测试重点、测试结果、缺陷列表、缺陷分布等测试过程记录信息，还可能包括测试团队的质量观察和改进建议。

为什么要有质量报告？

汇总和存档本周期内的软件质量情况。

质量报告包括特定周期内的测试结果、缺陷情况以及其他质量相关信息。因此，质量报告能够总结这一时期的质量状态，并作为过程记录进行存档，便于按需查阅。

与团队同步当前的质量状态，建立质量共识。

团队应该就一个开发周期内的软件质量状态达成共识。相对完善的质量报告可以帮助 QA 与团队快速对齐，建立质量共识。

便于对比分析不同阶段的质量情况，识别改进重点。

一个包含充分有效信息的质量报告是非常有价值的组织过程资产。通过跨阶段对比和分析质量报告，可以看到质量变化趋势，帮助团队识别质量风险和未来的改进重点。

相关主题

- 质量度量
- 质量度量可视化

常见误区

1. **质量报告就是测试报告。**测试报告是一个容易与质量报告一并讨论的概念。通常情况下，测试报告是质量报告的一部分。尽管我们通常不刻意区分这两个概念，但它们的本质目的是不同的。测试报告的重点在于记录测试过程和结果，而质量报告则不仅关注测试过程，还会更详细地记录与质量有关的其他信息。在其他一些场景下，质量报告单指关于软件质量状态的信息报告，不包含测试报告的内容。
2. **只有 QA 需要关心质量报告，其他角色不需要关心。**要想获得更好的质量，需要全团队共同努力。因此，所有角色都 ` 需要关注质量报告，并着重关注质量状态不佳或不稳定的部分。
3. **质量报告的目的是归档和留痕，并没有实际用途。**如果质量报告中包含充分有效的信息，可以帮助团队同步状态并指导质量改进过程。当质量报告流于形式时，不应归责于报告本身，而可能是因为报告没有包含有效信息，也可能是因为团队没有充分重视导致的。

参考

- Bug Report 该怎么做?

质量基础设施

测试工具 / 框架 / 系统

由于整个质量体系中有大量的测试和质量相关工作需要完成，为了高效实施这些工作，需要相应的工具、框架和系统等基础设施。基础设施包括几大模块：测试工具和测试框架、测试用例管理系统、缺陷管理系统、测试宿主环境、代码管理系统和持续集成系统等。

- 测试工具可以直接使用并执行测试，一般不需要二次开发；
- 测试框架则一般需要通过二次开发才能实施自动化测试；
- 测试用例管理系统可以创建和修改测试用例，跟踪测试用例，记录测试执行结果等；
- 缺陷管理系统可以创建和跟踪缺陷等；
- 测试宿主环境主要提供各种测试运行环境，包括各类虚拟化系统和容器化系统、模拟器和仿真器等；
- 代码管理系统主要管理各种测试代码和资源等；
- 持续集成系统主要控制和管理各种测试在流水线中如何准备、执行和收集测试结果等。

质量基础设施是质量体系的支撑，只有有效地使用这些工具、框架和系统，才能高效地落地质量体系中的各种实践，做到事半功倍。

基础设施包含哪些测试工具 / 框架 / 系统?

测试工具 / 框架

比如 Selenium IDE，Insomnia，Postman，JMeter、JUnit，REST Assured，Selenium Webdriver，PlayWright，Gatling 等

测试管理系统

比如 TestLink，Cucumber Studio，Zephyr Squad，Bugzilla，Mantis 等。

测试宿主环境

比如 VMWare, Docker 等。

代码管理系统

比如 Git，SVN，Perforce Helix Core 等

CI/CD

比如 Jenkins，Gitlab，GoCD 等

安全相关工具与系统

Fortify，Metasploit，Snort，Kali Linux 等

软件流程管理系统

Jira，Gitlab 等

除了以上主要的基础设施外，对于不同类型的软件系统的质量工作，还需要其他各种类型和特有的基础设施。例如各类仿真器、测试数据生成和管理系统、测试环境管理工具、风险管理工具、测试云和测试中台等。

常见误区

1. **有了高效的自动化测试工具和测试框架等基础设施，就可以实现高效的自动化测试。**高效的自动化测试包含方面太多，影响自动测试的因素也很多，比如测试用例的有效性和稳定性，被测环境的稳定性，测试工程师对于工具或者框架的熟练程度等等。因此，即使有一款高效的自动化测试工具或框架，如果没有满足上述前提条件，也无法实施高效的自动化测试。
2. **商用收费的基础设施一定比开源免费的好。**一般情况下，商用的工具、框架和系统比开源的功能更丰富，并且有商业支持。但是它们存在一个很大的缺陷，就是很难进行定制开发，对于一些特殊的系统很难进行集成。而开源的则可以通过修改和二次开发来实现。因此商业收费还是开源免费更好，还需要根据具体项目的情况来确定。

质量人员

质量文化建设

质量文化是指与质量相关的组织文化，代表着组织的质量价值观。质量文化可以概括为以下四个方面：

- 全程测试介入：我们提倡在软件生命周期的早期阶段和产品发布上线后的生产环境都介入和开展测试活动。这种全程介入的测试可以帮助我们尽早发现问题，并在后续环节中防止这些问题不断积累。
- 团队整体负责质量：我们推崇全功能团队，团队成员之间的角色分工不再那么明确，而是更加密切协作，共同为质量负责。这是我们遵循的指导性原则，能够促进跨部门协作和跨职能沟通，减少冲突，提高效率。
- 持续精准的自动化测试：自动化测试不应该一味追求覆盖率，而应该追求有目的的精准覆盖，考虑业务风险。自动化测试需要能够在持续集成流水线上持续运行，并随着功能的不断迭代进行相应的更新和增加。
- 质量内建：这是最核心的质量价值观。它将测试全程介入、团队为质量负责和持续精准的自动化测试结合起来，在敏捷软件生命周期的每个环节中做好缺陷的预防，把质量融入产品的开发构建中。质量内建的目标是通过合适的技术手段、流程和文化来最大限度地减少质量问题的出现，从而提高软件的交付质量。

为什么需要这样的质量文化建设?

通过流程、团队合作、自动化和核心价值观这四个维度来阐述质量文化，每个维度都有着各自的价值：

1. 流程维度：全程测试介入能够尽早发现问题，缩短反馈周期，提高软件开发的质量，减少问题修复成本，同时可以减少后期集中的测试工作量。
2. 团队合作维度：团队中的不同角色承担不同的质量职责，利用各自的优势，发挥每个人的最大价值。作为整体，团队对质量负责，通力合作更有利于保障质量。
3. 自动化维度：自动化测试是快速反馈的必要手段。基于业务风险精准覆盖的自动化测试更有效，也更能保障快速反馈的效率。持续运行的自动化测试可以确保系统功能不会因为新代码的提交而被破坏，同时迭代递增，确保新开发的功能被有效自动化测试覆盖。
4. 核心价值观维度：在软件开发生命周期越往后期发现的缺陷修复成本越高，质量内建则要求将质量内建到产品中，即做好缺陷的预防，以降低修复和返工成本，从而提高开发效率和软件交付的质量。

相关主题

- 测试左移
- 持续测试
- 分层策略
- 测试右移

常见误区

1. **质量只由 QA 把关。**为了保障软件质量，团队中的每个人都需要对质量负责。质量活动应该在软件开发的各个环节中展开，而不是由某个特定的角色负责。
2. **只关注阶段性质量。**不能只依靠开发完成后的测试阶段来进行测试和关注质量，而是要将测试左移到尽早的需求阶段，右移到生产环境，并且在整个软件开发生命周期的每个环节都要关注质量，持续收集质量反馈。
3. **只追求 100% 覆盖的自动化测试。**自动化测试需要平衡投入和回报。追求 100% 的自动化测试覆盖率需要大量投入，包括编写、维护和执行的资源，可能得不偿失。
4. **相比覆盖率，更需要关注测试有效性。**自动化测试需要根据产品需求和风险分析的结果制定测试策略，实现尽可能精准的测试，以获得最有效的测试覆盖率。

参考

- 敏捷测试宣言和原则
- 敏捷测试的核心
- 敏捷测试的指导性原则
- 说好的团队为质量负责呢?
- 精益测试

多角色协同

多角色协同是指质量相关人员之间的组织形式、职责分工、协作流程、规范以及具体的协作方式。传统的质量保障工作主要由 QA 承担，相应的组织架构通常是独立的测试部门。然而在新的形势下，我们提倡整个团队为质量负责，不同职责（如 BA、Dev 和 QA 等）的人员高度融合在一个交付团队内。每个角色并不局限于固定的人员，可以根据需要在不同时间段由不同的人来扮演相应角色，承担该角色的质量职责，并与团队其他角色协同完成质量保障工作。

为什么需要质量相关的多角色协同?

提倡质量相关人员高度协同，符合软件开发的特点和新形势下软件质量的要求，其价值主要体现在以下几个方面：

1. 软件开发是一项社会活动，需要多个角色的高效协同，任何单一角色或人员难以胜任。在新形势下的软件生态更加复杂，对协作要求更高，质量相关人员高度协同有利于保障交付高质量的软件。
2. 传统的按职能划分的部门会导致部门之间目标不统一，存在利益冲突，跨部门沟通困难，影响交付高质量的软件。而团队高度融合，则可以减少部门间的阻力，降低沟通成本，避免由于沟通导致的错误和延误，提高协作效率，能够更好地实现全生命周期的质量保障和团队整体为质量负责。

相关主题

- 质量文化建设
- 绩效管理

常见误区

1. **团队里有不同的角色，各自有不同的目标。**有些团队虽然将不同职责的质量相关人员融合在一起，但团队成员很难形成合力，这是因为团队没有统一的目标。要实现团队高效协同，统一的目标驱动非常关键。
2. **只有 QA 需要对软件的质量负责。**传统上，QA 承担着保障质量的责任，这种意识已经根深蒂固。如果没有认知层面的转变，即团队所有成员都意识到自己对质量的负责，实现真正意义上的融合也是很困难的。

参考

- 敏捷测试的指导性原则
- 说好的团队为质量负责呢?

绩效管理

质量人员的绩效管理主要是评估质量相关人员的工作表现和业绩情况，以帮助员工实现高绩效。绩效管理分为个人绩效和团队绩效两种形式。鉴于我们所倡导的质量文化需要团队整体对质量负责，因此推荐质量人员的绩效管理以团队绩效为主。这可以促进团队成员之间的协作和互助，激发团队的创造力和创新精神，从而提高团队的整体绩效。同时，适当的激励方式也可以帮助员工更加积极地投入工作，实现个人和团队的共同目标。

为什么要对质量人员进行团队绩效管理?

软件交付质量是每个团队成员的责任。因此，要实现高质量交付，需要所有人通力合作。对整个团队进行绩效管理和激励有助于促进整体团队合作，从而实现高质量交付。相比之下，个人绩效评估可能会导致团队内部的利益冲突，破坏整体团队合作，从而影响整体团队绩效。例如，以发现 bug 的数量分别作为 Dev 和 QA 的绩效指标，Bug 多 Dev 的绩效差，而 QA 的绩效好。这样就会形成冲突。

相关主题

- 质量文化
- 质量度量

常见误区

1. **QA 发现的 Bug 数量越多，绩效就越好。**上线前 QA 发现的 Bug 数量多，不能简单认为是 QA 工作做得好，也不代表是 Dev 工作做得不好。这需要分析产生大量 Bug 的原因，有可能跟计划、需求分析、代码质量、自动化测试等因素有关系，因此需要团队一起来制定改进举措，尽力做好 Bug 预防，做到质量内建。
2. **另一方面，生产环境中的 Bug 数量可以作为团队整体绩效的指标之一。**如果逃逸到生产环境的 Bug 数量较多，团队更需要引起重视，采取措施提高生产环境的质量。
3. **QA 不能按期完成测试任务，就是绩效不好。**这个观点忽略了很多实际情况，没有充分考虑到 QA 工作的复杂性和困难性。

4. **首先，测试任务的完成时间受到很多因素的影响。**例如软件的复杂程度、测试环境的稳定性、需求变更等，这些因素可能会导致测试任务的进度延迟，而这并不一定是 QA 的责任。
5. **其次，QA 的工作涉及很多方面，并且需要跟不同角色或多个团队密切协作，不断沟通和交流，以确保测试工作的有效性和准确性。**因此，QA 不能按期完成测试任务并不一定代表他们的绩效不好，而可能是由于复杂的测试任务、团队合作不畅等因素导致的。
6. **绩效的结果比其他任何事情包括过程都重要。**仅关注绩效的结果而忽略过程，可能导致员工为了满足短期结果指标而采取不恰当的手段，甚至引起内部不正当竞争。因此，绩效管理需要强调结果导向的同时，也需要重视达成目标的过程管理，并定期回顾与改进过程，以实现目标。
7. **个人绩效比团队整体绩效更重要。**在质量管理方面，强调个人绩效可能会不利于团队协作，削弱团队整体的绩效。相反，推崇团队绩效，并重视团队整体对质量负责的思想，可以更好地实现高质量的交付。

能力建设

质量人员的能力建设指的是对与质量相关的人员进行能力培养，以储备能够满足企业未来软件开发和交付质量要求的人员能力。为了实现这一目标，可以构建质量能力模型和质量人员发展路线等方法来指明能力建设的方向，同时根据质量人员的不同梯队，制定相应的学习机制和培训规划，有针对性地培养不同层次的质量人员能力。

为什么需要能力建设?

在质量管理领域,质量人员的能力建设对于组织在未来软件开发和交付方面实现质量要求至关重要。以下是质量能力建设的几个方面的具体价值:

1. 质量能力模型和发展路线可以指导质量人员的能力发展方向,为质量人员实现自己的工作目标提供能力改进的建议。质量人员可以根据能力模型的要求,制定自己未来的学习发展计划,通过不断提升自身的能力来提高业绩,从而进一步推动公司整体质量保障能力和交付质量的提升。此外,能力建设还可以帮助质量人员明确自己的职业发展方向,拥有更清晰的目标感。
2. 通过培训,质量人员可以获得新的知识和技能,不断提高自己的能力水平。同时,培训还可以让团队成员共同学习和成长,增强彼此之间的合作与信任,进而提高团队的绩效表现。
3. 建立质量人员的学习机制可以帮助建立学习型文化,有利于跨团队的质量人员分享知识、共同学习,从而激发创新能力和工作热情,提高工作绩效。

相关主题

- 绩效管理

常见误区

1. **培训是能力建设的唯一途径。**培训是能力建设的一种重要手段，但更重要的是通过构建能力模型和发展路线来明确组织对质量人员的能力要求和能力建设的目标，同时配合必要的培训和健全的学习机制，多方位地进行质量人员能力培养。
2. **无需提前系统地进行质量能力储备，只需在需要时安排培训来提高质量人员的能力。**一些企业或组织在质量人员能力储备方面存在这样的错误认识。其实，质量能力培养不是一朝一夕就能完成的事情。临时安排的培训缺乏系统性，可能会存在“头痛医头脚痛医脚”的问题，不利于质量人员的系统培养，也不利于组织质量能力的持续改进。甚至，临时安排的培训还可能造成浪费。因此，企业或组织应该提前规划和投入，建立系统的质量能力储备体系，并持续进行质量人员的能力建设。

参考

- 如何提升软件测试人员的能力

作者

林冰玉

Thoughtworks 总监级咨询师 / 质量赋能专家

在 2008 年加入 Thoughtworks，具有多年软件行业测试与质量管理、质量咨询经验，参与并负责过多个大中型项目的质量保障工作，先后服务于多个来自澳洲、英国、美国等海外大客户项目以及国内多家企业，行业涉及电信、医疗、银行、保险、税务、时尚领域等。擅长敏捷开发过程中的质量分析和保障工作，以及结合客户特点定制管理和体系建设改进方案，以质量专家的身份帮助团队进行质量赋能，从而实现价值交付和持续改进。目前工作重点为质量赋能、质量体系建设和管理、以及组织的质量人才培养等。

刘冉

Thoughtworks 资深软件测试和质量咨询顾问

超过 20 年软件开发和测试工作经验。作为开发工程师做过嵌入式系统和应用的开发、嵌入式 JVM 和浏览器的移植、Linux 系统内核裁剪和定制化开发等。作为测试工程师，做过多种类型系统的测试分析，测试策略，测试设计以及测试工具和自动化测试系统的开发等。其中对于 Web 应用测试，Web 服务测试，服务器性能测试，移动测试，安全测试，敏捷测试，测试驱动开发（TDD），测试分层一体化解决方案，以及代码管理（SCM）、持续集成（CI）、持续交付（CD）和 DevOps 等有深入理解。

于晓南

Thoughtworks 专家级咨询师 / 赋能讲师

拥有十余年质量交付经验，2018 年加入 Thoughtworks，曾参与和带领过各种大型复杂项目，行业包括金融、汽车和新零售等，涉及企业资源管理、项目管理、客户管理等多个业务领域，引导质量分析与流程改进。在敏捷项目管理、数字化产品管理、敏捷需求管理、质量分析与持续改进等方面有丰富经验，目前专注软件质量和数字化产品需求领域的赋能工作，为客户的产品和交付团队建立实践框架。

Thoughtworks 是一家全球性软件及技术咨询公司，集战略、设计和工程技术咨询服务于一体，致力于推动数字创新。我们在 18 个国家 / 地区的 51 个办公室拥有超过 11,500 名员工。在过去的 30 年里，我们为全球各地的众多合作伙伴倾力服务，与客户一起创造了非凡的影响力，帮助他们以技术为优势解决复杂的业务问题。

thoughtworks.com

/thoughtworks

Strategy. Design. Engineering.