测试自动化后 我们需要什么样的 QA？

我们先讨论一下在传统的瀑布模型下 QA 是如何工作的，其中最主要的问题是什么;然后作为对比，我们再来看看敏捷团队里的 QA 是如何工作的，工作重点又是什么;最后，我们详细看一看在新的职责下，QA 应该如何做。

瀑布开发模型

即使在今天，在很多企业中瀑布模型仍然是主流。每一个需求都需要经过分析、设计、开发、测试、上线部署、运维等阶段。虽然一些企业已经开始实施敏捷开发，比如项目/产品以迭代的方式运作，也有诸如每日站会、代码检视等敏捷实践，但是如果仔细审视，你会发现其实开发模式从骨子里来说还是瀑布：按照软件组件划分的部门结构(详见康威定律)、按照职能划分的团队(开发和测试分属不同部门)、过长的反馈周期、永远无法摆脱的集成难题等等。

随着软件变得越来越复杂，团队里没有任何一个人可以说出系统是如何运作的，也不知道最终用户是谁，以及最终用户会以何种方式来使用最终的软件。

更糟糕的是，按照职能划分的团队在物理上都是隔离的，比如独立的测试部门，独立的运维部门，整日忙碌而难以预约到档期的业务人员，当然还有经常疲于交付，无处吐槽的苦逼开发。由于这些隔离，信息的反馈周期会非常长，一个本来很容易修复的缺陷可能在 4 周之后才会被另一个部门的测试发现，然后通过

复杂的工作流(比如某种形式的缺陷追踪系统)流到开发那里，而开发可能还在拼命的完成早就应该交付的功能，从而形成恶性循环。

瀑布模式中的 QA

在这样的环境中，QA 们能做的事情非常有限。在需求开始时他们会参加需求澄清的会议，制定一些测试计划，然后进行测试用例的设计。有的企业会用诸如 Excel 之类的工具来记录这些用例。这些写在 Excel 里的，“死”的用例作用非常有限。而最大的问题在于：它们无法自动化执行。另外，在实际软件开发中，需求总是会经常发生变化，需求的优先级也会有调整，然后这些记录在 Excel 中的“死”的用例会很快过期，变得无人问津。

除此之外，QA 中的有些成员会使用工具来录制一些 UI 测试的场景，然后在每个新版本出来之后进行回放。然而，当 UI 发生一点变化之后，这些自动化的用例就会失效：比如 HTML 片段中元素位置的调整，JavaScript 的异步调用超时等等。

显然，这种单纯以黑盒形式来检查功能点的测试方式是不工作的，要真正有效的提升软件质量，仅仅通过事后检查远远不够，软件的质量也应该内建于软件之中。QA 的工作也应该是一个贯穿软件生命周期的活动，从商业想法到真实上线，这其中的所有环节都应该有 QA 的参与。

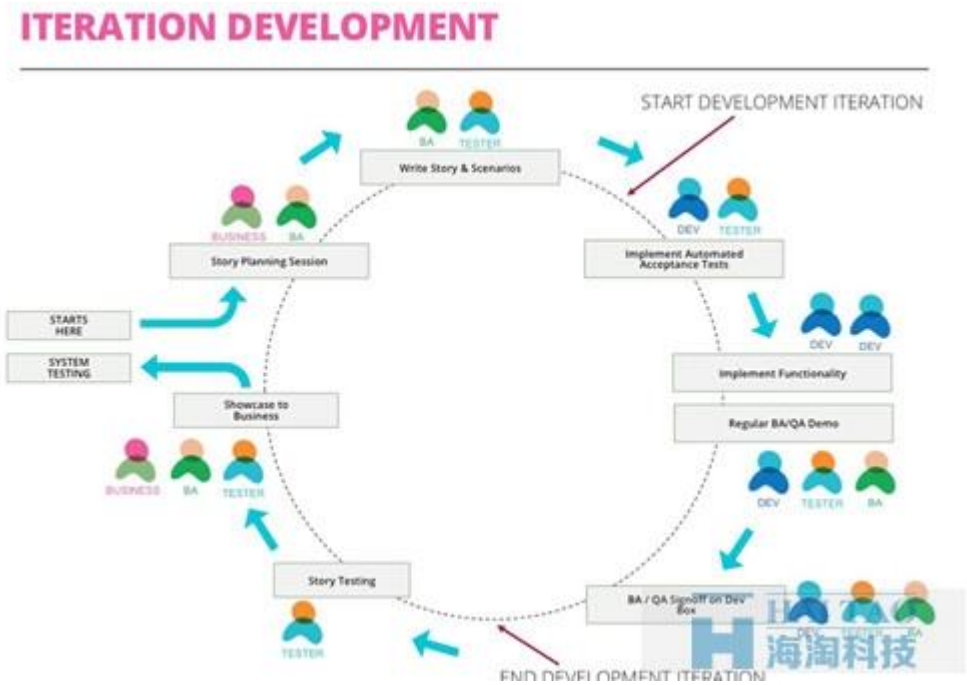
系统思考

如果不从一个系统的角度来思考软件质量，就无法真正构建出健壮的、让业务和团队都有信心的软件系统。质量从来都不只是 QA 的职责，而是整个团队的职责。

关于软件质量，一个根深蒂固的误解是：缺陷在开发过程中被引入，然后在测试阶段被发现，最后在 QA 和开发的来回撕扯中被解决(或者数量被大规模降低)，最后在生产环境中，就只会有很少的，优先级很低的缺陷。

然而事实上，很多需求从开始就没有被仔细分析，业务价值不很确定，验收条件模糊，流入开发后又会引入一些代码级别的错误，以及业务规则上的缺陷，测试阶段会漏掉一些功能点，上线之后更是问题百出(网络故障、缓存失效、黑客攻击、操作系统补丁、甚至内存溢出、log 文件将磁盘写满等等)。

在一个敏捷团队中，每个人都应该对质量负责，而 QA 则以自己的丰富经验和独特视角来发掘系统中可能的质量隐患，并帮助团队将这些隐患消除。



我在 ThoughtWorks 的同事 Anand Bagmar 在他的演讲 What is Agile testing- How does automation help? 中详细讨论过这部分内容。

QA 到底应该干什么?

本质上来说，任何软件项目的目标都应该是：更快地将高质量的软件从想法变成产品。

将这个大目标细分一下，会得到这样几个子项，即企业需要：

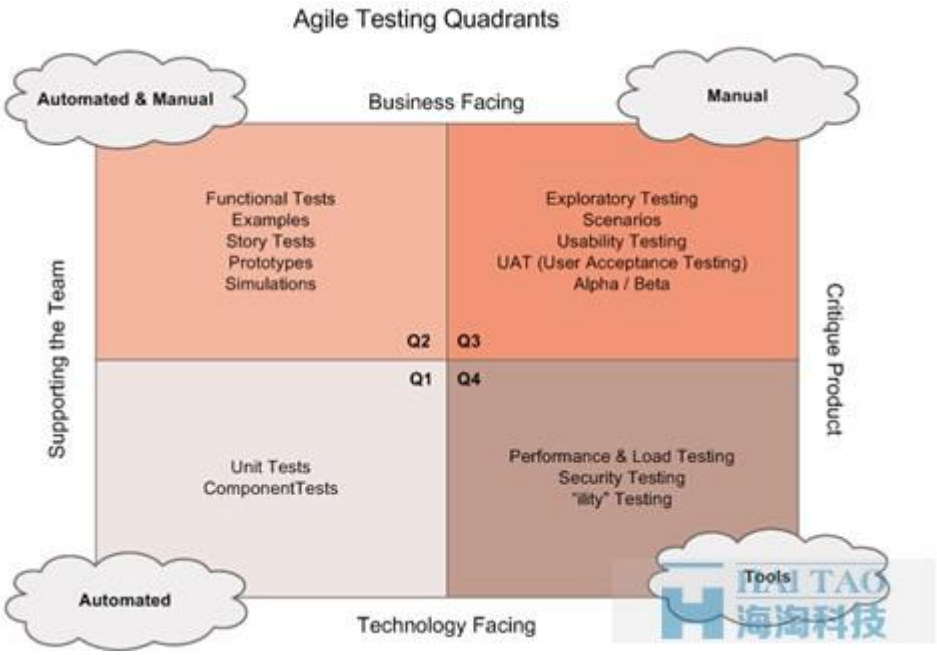
- 更大的商业回报(发掘业务价值)
- 更短的上线时间(做最简单，直接的版本)
- 更好的软件质量(质量内嵌)
- 更少的资源投入(减少浪费)

其实就是传说中的多、快、好、省。如果说这是每一个软件项目的目标的话，那么团队里的每一个人都应该向着这个目标而努力，任何其他形式的工作都可以归类为"浪费"。用 Excel 记录那些经常会失效，而且无法自动执行的测试用例是浪费，会因为页面布局变化而大面积失效的 UI 测试也是浪费，一个容易修复的缺陷要等到数周之后才被发现也是浪费。

在这个大前提下，我们再来思考 QA 在团队里应该做什么以及怎么做。

QA 的职责

Lisa Crispin 在《敏捷软件测试》中提到过一个很著名的模型：敏捷测试四象限。这个模型是 QA 制定测试策略时的一个重要参考：

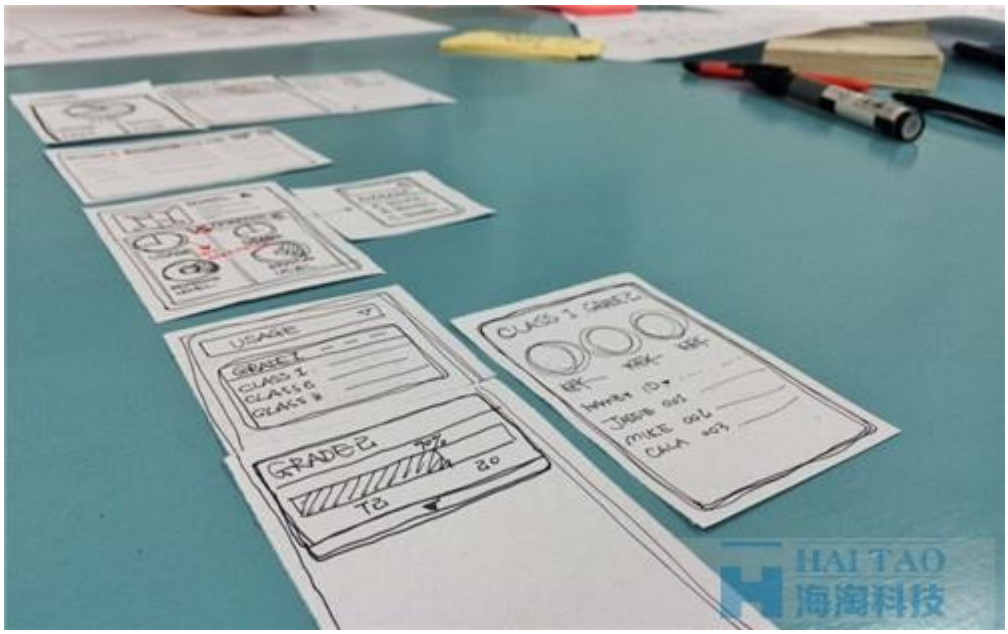


如果按照纵向划分的话，图中的活动，越向上越面向业务;越向下越靠近技术。横向划分的话，往左是支撑团队，往右是评价产品。

其实简化一下，QA 在团队里的工作，可以分为两大类：

1. 确保我们在正确的交付产品
2. 确保我们交付了正确的产品

根据这个四象限的划分，大部分团队可能都会从 Q2 起步：QA 会和 BA，甚至 UX 一起，从需求分析入手，继而进行业务场景梳理，这时候没有具体的可以被测试的软件代码。不过这并不妨碍测试活动，比如一些纸上原型的设计：



这一阶段之后，我们已经有了用户故事，这时候 QA 需要和开发一起编写用户故事的自动化验收测试。当开发交付一部分功能之后，QA 就可以做常规的用户故事测试了，几个迭代之后，QA 开始进行跨功能需求

测试和探索性测试等。根据探索性测试的结果，QA 可能会调整测试策略，调整测试优先级，完善测试用例等等。

根据项目的不同，团队可以从不同的象限开始测试策略的制定。事实上，Q1-Q4 仅仅是一个编号，与时间、阶段并无关系，Lisa Crispin 还专门撰文解释过。

关于 QA 如何在软件分析的上游介入，并通过 BDD 的方式与业务分析师一起产出软件的各种规格描述，继而通过实例来帮助整个团队对需求的理解，ThoughtWorks 的林冰玉有一篇文章很好的介绍了 BDD 的正确做法。如果将 QA 的外延扩展到在线的生产环境，制定合理的测量指标，调整测试策略，强烈推荐林冰玉写的另一篇文章产品环境中的 QA。

其他职责

事实上，软件生命周期中有很多的活动处于灰色地段。既可以说是应该开发做，又可以说应该 QA 做，甚至可以推给其他角色(比如 OPs)。不过我们知道，一旦涉及角色，人们就再也不会按照全局优化的思路来应对问题了。这种灰色的活动包括：

- 持续集成的搭建
- 测试环境的创建与维护
- UAT 上的数据准备
- 代码中的测试代码的维护
- 测试代码的重构

在团队实践中，这些活动我们通常会让 QA 和开发或者 OPs 同事一起结对来完成。一方面避免知识孤岛的形成，另一方面在跨角色的工作中，也可以激发出更多不同的思路。

万能的 QA?

虽然在这些活动中，QA 都会参与，但并不是说团队里只要有一个 QA 就可以了。QA 在参与这些活动时，侧重点还是有很大不同的。

比如需求分析阶段，如果有 QA 的加入，一些从 QA 角度可以发现的有明显缺陷的场景，则可以在分析阶段就得到很好的处理。另一方面，尽早介入可以设计出更合理的测试计划(比如哪些功能的优先级比较高，用户会更频繁使用，那么对应的测试比重也会更高)。在 Story 分析与书写阶段，QA 可以帮助写出更加合理的验收条件，既满足业务需求，又可以很好的指导开发。

在和开发一起编写澄清需求时，主要是编写自动化验收测试，而不是实际编写业务逻辑的实现(虽然 QA 应该参与 Code Reivew 环节，学习并分享自己的观点);甚至在上线运维阶段，QA 还需要和 OPs 一起来设计用户数据的采集指标(比如用户访问的关键路径，浏览器版本，地区的区分等)，从而制定出新的测试策略。