提高工作效率的千万条道路

Moco 组 陈伟彬

一、背景

测试千万条，质量第一条。

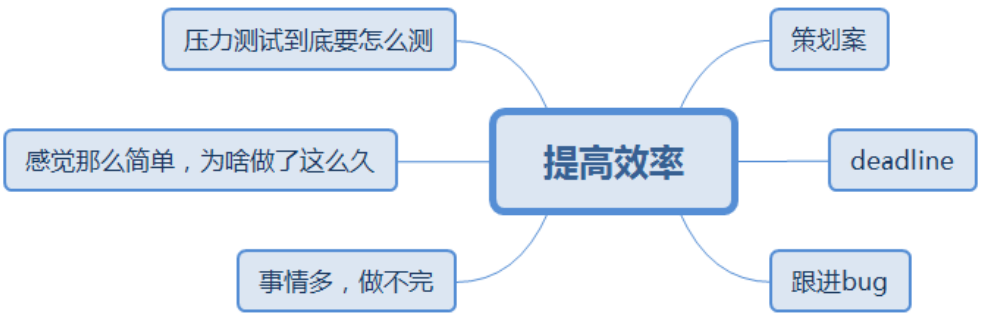
方法不规范，自己两行泪。

从客户端到服务器，从简单逻辑到压力测试，从立项到上线全流程的跟进，积累了不少测试经验，想着做做总结，一方面是对自己的沉淀，另一方面也想抽出通用的提高工作效率的方法，希望能对大家有所帮助。**篇幅较长，可见粗体和颜色标注部分快速阅读**。

本文主要对如何提高 QA 的工作效率做下总结，欢迎留言探讨。

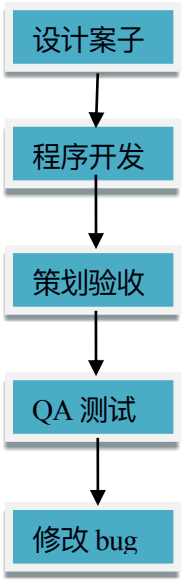
二、切入点

如何提高测试效率呢？我的思路是先用 xmind 做下思维发散，发散后进行归类总结，抽出通用的部分，再精简后进行详细分析，大家平时有什么想做但是没啥头绪的也可以尝试这样分析。



摘取了部分点列在上面，进而总结出，可按照三个大方面来分析，看如何提高工作效率，例如策划案、跟进 bug 等归类于流程角度，事情多做不完归类于个人角度，压力测试归类于是测试内容。下面从这三个方面，进行详细分析。

三、从流程角度去提高效率



平时我们一个版本的流程大致如上图所述，下面就从流程角度总结下提高测试效率的方法：

1、设计案子，QA 尽早介入

一般都是策划写完了案子，进行三方时，我们才会进行讨论，并修正里面不合理的地方。但是，**一般三方完后，程序就会立即开工了，事实上并不会留足时间进行修改**。而策划一般是不理解代码实现逻辑的，**偶尔在他们眼里简单的功能，就有可能会改动很多地方**，最终导致 QA 需要更多的进行耦合测试、回归测试，可从下面几点进行改善：

⑴ 在策划设计案子初期，可以尝试与策划沟通，**了解实现案子的目的**，尽可能减少耦合。若能基于当前已有系统进行组合实现相同体验效果，也可提出建议。

⑵ 程序写代码前，需要**评估代码实现难度**，难度较大时要跟产品进行协商，看能否换一种方式，降低实现复杂度。

2、程序开发，QA 需要了解实现原理

程序为了实现新功能，有时候难以避免会改动到旧的系统，但不一定记得告知 QA；另一方面，实现时为了代码方便，不一定会严格按照案子流程进行实现。这就会导致 QA 测试一方面留下了隐患，且旧系统和新系统不一定是同一个测试人员，容易造成测试遗漏。以下有几个办法可降低踩坑的概率：

⑴ 主动咨询程序实现原理，让程序**提供耦合系统的列表**，方便 QA 进行回归；耦合较多时，可讨论下是否有更合适的实现方案。

⑵ 若系统较大，找个时间组织程序进行**代码走读**，现场讨论潜在的隐患。代码走读

往往可以发现较多的隐患，自己看自己的代码永远都是没有问题。

⑶ 部分程序由于不知道有相关已有功能，可能会重写逻辑，可适当进行提醒。

⑷ 部分功能属于**跨版本开发，一定要做好开关**，确保新功能不会提前生效。

3、策划验收要及时

策划验收是非常有必要的，一方面程序实现上有可能与策划不一致，另一个方面即使实现了，策划在看到真实效果后，可能还要修改细节。

必要性大家都懂，但实际执行起来可能没那么容易，策划会忘了验收，或者验收不及时；也会出现 QA 测试完了，策划又改需求，QA 需要再次进行测试。

简单点，做个**泡泡推送功能**，一旦**程序完成功能拖单了，就推送给策划**，通知其及时验收即可。依据策划的执行情况，看是推送个人还是推送项目群里，群里推送多少有点压力。

4、QA 测试要合理评估，及时反馈

QA 测试阶段最主要是评估是功能完成开始，到封版本前，是否能把内容测试完毕；若不能，则需要进行调整。若测试进度的确非常紧张，则需要**提出风险，并及时跟主 QA 汇报**，最终依据情况看是否和产品进行汇报，有些功能并非一定要这个版本投放出去的，可依据实际情况做下调整。

5、修改 bug 要学会归类总结，刨根问底

bug 有就提，修改完就重新测试下，没问题就关闭，看似简单，其实才是能发现最多问题，提供提高效率的最好的流程。

⑴ **bug 产生的原因**是什么，需要理清楚，建议把原因在关闭单子时简略的写到单子里。一方面测试人员可根据原因写工具检查，或者改善相关流程来避免问题再次发生。

⑵ 对 **bug 进行分类**，计算 bug 的修复时长，看是哪类型的 bug 修复时间较长，与相关人员进行讨论，看如何缩短时间；再者，也可按照系统进行分类，看一般是什么系统 bug 较多。

通过 bug 的分类和统计，可呈现出明显的数据化结果，并**从该结果反过来去推动自身或者其余岗位的工作效率的提高**，测试效率提高的最有效的还是**从上流降低出问题的概率和数量**。

四、从个人角度去提高效率

站在不同的角度思考问题时，总是可以发现更多的惊喜，本环节则站在自身测试角度去

分析问题，相信以下问题大部分都遇到过。

1、活多？经常做不完，经常加班加点去弥补

或许不少人都有这个烦恼，工作是永远做不完的感觉。在活的总量不变的情况下，如何更好的完成测试工作，让自己不那么累呢？以下有几个方法不妨一试：

（1）每个版本初期，要了解自己所要承担的所有工作，对工作进行**优先级排序**，评估所需时间，若在 deadline 前无法完成，**及时与测试负责人反馈**。

（2）对自己测试的内容进行风险评估，存在风险的提前与组内沟通，当你所承受的工作风险总量较大时，最好是重新进行分配，将**风险均化**。

2、工作易被打断，无法专注进行测试

是否经常遇到策划让你帮忙，程序让你重现一个 bug，外网出现问题需要跟进，新人刚来经常咨询你问题，但你手头正在专注测试一个玩法，因此被三番四次打断，一来影响了测试进度，二来也影响情绪。这些时候，**要学会 Say：等会儿**

这里非常合适使用**番茄工作法**，在一个番茄时间内（25 分钟），不要被任何非紧急事情打断，如果有事情需要你做，则记录下来，排到下一个番茄时间或更后的时间里。

此外，并非所有问题都需要你支援才可以解决，例如让你帮忙录个视频截个图，要分清哪些事情是非你不可，**学会合理拒绝**，否则其他人慢慢习惯了你的帮助，会理所当然的找你。

3、效率低下，干的事情做很久才做完

一方面，可能是自己的测试方法不得当，需要改进测试方法；另一个方面，也需要认清事实，也可能是**自己不在行**，若有在行的同事，建议可以跟 TA 交换下测试内容。

或许你会觉得，不在行的迟早也是要学会的，这没有错，但前提是在你的时间允许情况下，量力而行，对于效率的提高有更多的帮助，**做完在行的事情后，反而会多出时间去学习不在行的技术和测试方法**。

4、对于分配的测试内容不在行，或跟进的流程不熟悉

由于特殊情况，如同事请假，部分你平时不是很熟悉的测试内容，可能需要你来进行测试，事实上，若等到这个节点了，也比较难一下子变得熟悉起来，但平时是可以提前做好准备的，如：

（1）**交叉测试**，可以让两个人交换下测试内容，一方面可以为后面突发情况做准备，一方面也可以提高个人适应能力。

（2）**每个版本的流程和进度跟进，可以进行轮流**，让每个人更有全局观，对开发流程更为熟悉，也有助于个人能力的提高。

5、提高风险意识，并设置相关措施

QA 除了测试以外，也要能当前测试的工作具有风险评估的能力，包含时间是否充足、是否会引起玩家不良舆论、新系统是否不够稳定等，并对存在的风险设立相关紧急处理措施，

如新系统必须有开关控制，可做到及时关闭；玩家舆论是否做好了监控，可及时查看。

事实上，若能够对自己的工作和外放的内容具有良好的风险评估能力，也说明对自身的测试工作已非常清楚，也是对自我能力的一种肯定。

6、每个月要腾出时间进行总结，改进测试方法

⑴ 每个版本结束后，无论多忙，多需要腾出时间对自己的工作进行**回顾总结**，即使是半个小时，也能对自己有很大的帮助。

⑵ 回顾自己做不好的地方，**想想改进方法**，与组内同事分享，组内同事有时候也能提供更好的思路。

⑶ 总结做的好的地方，看是否可以**推广给其余同事**，提高彼此的工作效率。

⑷ 回顾下策划、程序、运营等是否也有可以改进的地方，**适当的提出改善建议**，或者做相关的工具支持，提高其工作效率。

⑸ **将工作进行数据化分析**，例如分清楚自己在系统测试、接口跟进、bug 验收等各个部分的占比，看是哪个部分耗时最多，针对性提高自身能力。

此外，若想让策划和程序配合我们的测试改进自身的开发设计流程，最好也能提供横向或纵向的数据对比，**更具有说服力**，如你写的表格合并工具，可节省策划 1 人/天的工作。

五、从实际测试内容去提高效率

对于游戏的各个系统的测试，每个人几乎都要自己走一遭，并作为积累为后续提供更可靠的经验，但其实有不少系统的测试都比较通用，以下按照客户端、服务器、配置表三个方面分享测试方法：

1、客户端

客户端测试主要是针对特效，模型，UI 等交互进行测试，也包含数据的正常显示和更新。一直以来客户端测试都没有特别好的自动化测试方法，**若客户端功能长期不变，则可以考虑结合 Airtest 进行自动化测试**。

事实上，客户端测试可以看做是一系列操作的组合测试，使用等价类方法对用例进行精简归类，可提高测试效率。以下举两个例子：

A. **卡牌/角色测试**

很多游戏都有众多卡牌和角色需要进行测试，一般第一次测试几乎都会涵盖所有卡牌，但是卡牌数量非常多，测试起来非常耗人力和时间，二来也非常枯燥，不妨按下面步骤拆解：

⑴ 按效果**对卡牌进行分类**，类别往往一层是不够的，可分两层或三层，然后抽取不同效果的卡牌 1-2 张进行测试，即可大大缩脚测试范围。

⑵ 当然，卡牌的资源是不可以忽略的，怎么办的？可以跟程序了解资源索引的方法，一般情况下都是可以从配置表，或者固定的位置，根据卡牌 ID 找到的。从而写好**资源检查**脚本，实现自动化检查。

⑶ 基于 1，当效果发生变化，我们就可以根据相同效果或耦合效果进行**针对性回归**，节省测试成本。

B. **概率性问题跟进**

测试过程中，偶尔会出现难以重现的问题，但在外服由于玩家数量增多，概率性的问题往往就会演变成必然，若问题较为严重，则非常有必要重视，重现和定位到问题进行修复。

概率性问题虽然没有百分百通用的解决方案，但还是有科学方法辅助的：

⑴ **问题产生的条件**有哪些，概率性的问题往往都是因为符合某种条件后才可以触发，

⑵ 若能保证所有**元素都固定下来**，再对**元素进行分类**，排列组合后，就会变成有穷的测试用例，继而找到最终的答案。例子可见经典 bug 中的《诡秘莫测的数字消失之谜》

2、服务器

相比客户端，服务器也有非常多的测试方法，最重要是保证数据存储正确，一般数据存储在数据库中，但会有缓存作为中间变量，降低数据库压力。因此，**在测试时一定要确保数据最终存储到数据库中，缓存数据也需要及时更新**，否则会有数据不一致的问题。

对于服务器测试，最重要的是**了解数据库存储和配置表配置方式**，了解这两个之后，大部分测试可以辅助脚本自动化检查大大减少测试时间。

若想了解系统涉及到哪些数据库表，在不求人不看代码的情况下，有个非常简单的办法，就是**对比操作前后，数据库表中哪些字段发生了变化**。

此外，服务器主要是性能和逻辑测试相关，下面举例说下这两点：

A. **压力测试**

压力测试框架公司不少组里都有，可挑选合适的框架进行使用（如吴炜峰同事的压测框架就蛮实用的），自身则主要是实现机器人的登录及游戏内各种行为。在游戏服务器协议基

本稳定的情况下，即可进行开发，后续只需要针对新增协议进行测试即可。

⑴ **机器人一定要走所有正常玩家的协议代码，尽量避免为了支持机器人而重新写接口，这样容易漏掉部分逻辑**。

⑵ 可对重要系统进行**单点测试，例如登录、领取邮箱**奖励等。

⑶ **模拟玩家日常行为**，机器人随机采取行为，可以发现意外惊喜；例如之前提过的一个经典 bug《AB 死锁》问题，就是机器人测试过程中发现的。

⑷ **在新增公共系统**时，建议进行一轮压力测试，这个就需要前期就有扩展性较强的压测框架。

⑸ 压力测试框架除了压测以外，事实上也是一个**协议测试框架，特别是并发操作，重复操作**的测试有这莫大的帮助

B. **奖励测试**

奖励方面的测试非常重要，一方面这是出事故的重灾区，另一方面对玩家体验也有相当大的影响，有几个点需要注意测试好，基本就可以避免事故级别了：

⑴ 奖励**如何保证不会重复领取**，这个点一定要测试清楚，是否有数据库标记已领取，或者有别的方式保证，不能简单的听信程序或者游戏里看到已领取就过了。

⑵ 一定要**模拟重复领取**的情况，**重复发协议**比较靠谱。而且，根据协议中的参数，就可以知道哪些参数具体影响了奖励。

⑶ **奖励监控**，在外服新活动，或者新奖励类型时，提前让运营做好外服奖励监控脚本，一出现问题，能马上监控到。

⑷ 奖励的活动要**做好开关**，出现问题时，可第一时间暂时关闭系统，但不影响玩家其余体验，将影响降到最低。

3、配置表

相信不少组内，会写很多的表格检查脚本，对于脚本工具的编写，提几个通用的建议：

⑴ 尽可能的做到**一键使用，并推广给策划使用**，如表格检查，QA 反馈给策划，策划修改后又给 QA，QA 再次验证，如此往复事实上是占用了不少没必要的人力。

⑵ **支持对配置表进行临时修改**，一个是策划会尝试进行一些活动配置或频繁调整数值来进行体验，另一个 QA 也经常需要修改配置进行测试。

⑶ **监控配置表的提交情况**，避免策划误提文件，或提交文件后没通知 QA 而漏了测试。