第 13 章　让测试为我们服务

不知各位在开发过程中有没有遇到过下述情况。

- 没有明确的目标，只能摸索着进行开发，最后成品与原定的需求不符，导致整个推翻重做
- 程序严格按照设计书实现，但在集成测试阶段发现其与周边功能结合不到一起，导致故障频出

本章将把测试的观点引入开发的各个过程之中，为解决或规避开发中出现的上述问题提供解决方案。我们希望本章内容能让各位从开发初级阶段开始就知道需要兼顾测试，明白自己需要考虑什么，需要亲眼确认什么，为各位的整个开发过程带来帮助。

NOTE

本书的中心内容是编程技法，因此不对各个测试所用的手法进行具体讲解。测试手法的相关内容有许多优秀的专业书籍可供参考，有兴趣的读者不妨一试。

·《软件测试的艺术》（机械工业出版社，2012 年）

·《软件测试 PRESS 合集》[①]（技术评论社，2011 年）

13.1　认识现状：测试的客观环境

在各位所处的环境中，测试处于怎样一个地位呢？

在从确认运行状况到进行系统测试的过程中，各个阶段都花大把时间去测试吗？还是在编码结束后只简单确认一下运行状况就发布了呢？又或者是仔细编写测试代码，但却尽量控制手动测试呢？

细化到各个测试阶段来说，由于项目的性质、交付对象 / 搭档 / 经理的方针等因素的影响，很多时候我们不能选择最合适的工作流程，所以时间安排往往不尽如人意。另外，就算日程表已经制定好，某些时间安排也会因各种情况而缩短甚至删除。

所以很多时候我们心里明白测试的重要性，却实在挤不出时间给测试。对于需要赶工的项目而言更是如此。面对火烧眉毛的日程表，我们往往会失去方向，不知道该从哪里下手。

① 原书名为『ソフトウェア・テスト PRESS 総集編』，暂无中文版。——编者注

13.2　将测试导入开发各个阶段

项目开始之后，为了能尽可能高效地完成开发的各阶段，即便时间再紧，也要尽早将测试的观点引入项目之中。因为随着开发流程的推进，修改、变更都会越来越难。

这就像身处一个陌生的环境，只要在行动前先确认一下地图，了解当前地点和目的地的位置，之后就基本不会走偏。即便方向略有偏差，只要能在转第一个弯时及时发现，我们依然能很快返回出发点。也就是说，如果途中能每到一个路口都确认一下，就算其间走错过一条路，我们也能很快折返或者在下一个路口修正方向。但是，如果行动前不作任何确认，等瞎蒙乱撞地跑到了其他城市才意识到走错了路，此时再回过头来看地图就为时已晚了。到了这个地步，不但向前走找不到目的地，就连折返也很难返回出发点，进退两难。这在系统开发上也是同样道理，我们开始项目之前必须正确理解项目的前提条件和目的，尽量在初期阶段及早发现错误，修正方向。

总之，所谓测试的观点，就是给项目定义一个正确的状态，然后在开发过程中及时发现偏离该状态的元素并对其加以改善。将测试的观点从程序测试阶段扩展至需求定义和设计阶段，这能有效地帮助我们在开发初期修正错误。

接下来我们按照开发流程一步步进行了解。

13.2.1　文档的测试（审查）

文档类的测试一般称为审查（Review），它也是测试的一种。认真进行审查能防止因理解偏差而导致的返工或者不小心实现了多余内容等问题。因此即便没有充足时间去单独审查每个文档，也要在开始实现或测试设计之前简单地将文档重新审视一遍。

文档可分为很多种，比如下面这些。

【主要文档类型】

- 需求说明
- 需求定义
- 基本设计
- 详细设计
- ER 图
- 数据表定义
- 测试说明或测试设计等

文档大致可分为三大类，一是确定开发方向的文档（需求说明、需求定义等），二是指导具体

实现的文档（基本设计、详细设计等），三是规定数据使用方式的文档（ER 图、数据表定义等）。

下面是审查文档时经常发生的问题。

【问题】

- 文档没有更新，内容陈旧
- 未指明未确定的事项
- 描述含糊不清
- 存在多个相同标题的文档文件，搞不清哪个才是最新的

对于内容陈旧的文档，只能随发现随更新，然后定期重审，没有其他办法。而其他问题都是有改善余地的，下面我们来一个个解决。

◉ 未确定的事项标明“未确定”

文档是担保程序以及测试正确性的重要指标，因此其内容务必准确详实，不能有遗漏，也不能有多余内容。

拿到文档时，不要直接动手开工，先从头到尾通读一遍文档，把其中有疑问的地方和不清楚的地方记录下来。特别是在项目刚刚开始的阶段，某些需求和规定还很模糊。此时要把不明确的部分找出来，在对象文档里注明“未确定”。明确未确定的项目是对残留的待确定项目的一种可视化，让我们能直观地看到哪里还有待确定的项目。想消除文档的遗漏、缺失等问题，这种可视化必不可少。“不清楚还有哪些不清楚的事”的情况必须尽早解决。记录未确定项目时可以同时将确定该项目所需的条件写下来，便于重审文档时确认是否需要处理该项目。

另外，查找文档漏记了哪些内容其实是一个难度很高的工作，这里介绍两种行之有效的方法。

一种是对比上一阶段的文档，查看内容是否有遗漏。由于当前阶段文档全部源于前一阶段文档，所以二者内容出现不吻合时，可以认为文档某处存在遗漏。

另一种是将功能、页面、数据等方面有关联的文档放在一起，看它们之间是否存在矛盾。如果对比文档时出现内容不契合的情况，那么文档某处很可能存在问题。

【消除不明确的点】

- 存在未确定的事项时，在对应文档内标明“未确定”
- 标注“未确定”的同时记录确定所需的条件
- 查看前一阶段的文档，确认项目是否存在遗漏
- 检查功能、页面、数据等方面的关联，确认各部分关联正常

◉ 描述时尽量避免歧义

如果阅读文档时发现内容存在模糊不清的地方，证明我们对正确的内容的定义还不够明确。

- 在表达同一个事物时用了不同说法（用语不一致）
- “○○等”“○○左右”等需要清单化的部分表述含糊不清
- 数值增减的幅度、上下限不明（比如写 1 ~ 10，看的人不清楚是按 1、2、3…还是按 1.0、1.1、1.2…的规律变化）
- 能数值化、符号化的内容却用文字描述（大于 ~ 、~ 以上、不足 ~ 等）

到了实现阶段，这类描述难免被开发者加入个人的理解，产生歧义，进而导致最后成品与我们想要的东西不一致。

若想改善上述情况，首先要将表达同一个事物的用语统一，坚决不用其他说法来表述该事物。用语的统一能带来认知的统一。需要清单化的东西一定要清单化，明确描述出所有项目。清单最好集中写在一处，其中在需要参考某些资料的部分写明相应资料的位置，以便将来出现变更时能轻松修改。

数值增减的幅度、上下限也要尽量写明。因为它会给是否限制输入内容、项目的显示宽度等设计方面带来影响。

能数值化、符号化的东西尽量用数值、符号来表达，不要用文字描述。因为文字描述更容易产生误解和歧义。

为避免开发者加入个人的理解，写文档时应避免加入多余内容，同时要将必备内容表述得尽量清楚。在发现模棱两可的表述时，如果只是单一的局部问题，我们只需即刻通知其他相关的开发人员，对该部位进行修正即可。但是当发现该问题散布在文档各处时，就要提起注意了。首先查看手头的其他文档，如果这些文档也有同样问题，说明编写文档的前期讨论做得不够充分。此时需要警示项目中的所有人员，因为该问题搞不好会影响整个项目的进程。

【消除模糊不清的描述】

- 表述同一事物时统一用词，使表述有整体感
- “○○等”“○○左右”等描述要列出所有可能性 → 例如 A. ○○、B. □□、C. △△
- 能数值化、符号化的就不用文字描述 → 用“>”“<”“≥”“≤”等符号以及具体数值替换文字描述
- 注明数值增减的幅度和上下限

◉ 明确区分最新版本和非最新版本

存在多个同名（近似名称）文档会带来混乱。相信很多人见过下面这种文件名。

- ○○○○修正版
- ○○○○最终版
- 新○○○○

这些文档的描述往往大同小异，却都各自存在一些需要修正或需要更新的地方。要解决这个现象，最好是对所有同类文件的描述进行一个汇总，做成一个最新的文档，其他的则都视为旧文档全部处理掉。

首先把可以确定的旧版本文档删除。接下来把重复的文档汇总，选出其中更新日期最近、名字最相似的两个进行比对，详细记录二者的区别，在其中一个文件中进行勘误及汇总。汇总完成后找出下一个最相近的文档重复上述工作。所有重复文档的对比、汇总完成之后，将用作比较对象的文档全部归档到一起，按照陈旧文档处理。一定要明确区分最新版本和旧版本的保存位置，防止再发生相同情况。像管理源码一样用版本控制系统来管理文档也是个不错的选择。

从防止同样情况再次发生的角度讲，如果陈旧文档已经明确失去了使用价值，可以考虑直接删除。

【保持文档处于最新状态】

- 删除确实陈旧的文档
- 根据更新日期和标题选择相近的文档逐一汇总
- 做出最新版本的文档之后，其余文档全部视为陈旧文档处理
- 确认没有问题的情况下删除陈旧文档

如果各个文档的审查工作正常进行，则能给后续阶段奠定可靠的基石，保证在遇到问题时有据可依。希望各位在开始编写代码之前先腾出一部分时间处理文档，它能让后续的时间安排更加灵活高效。

13.2.2 测试设计的编写方法（输入与输出）

测试设计的编写完成时间最晚不能晚于功能的实现。

进行测试设计时，需要根据测试内容和目的不同编写不同的输入文档。这是为了对程序进行多角度测试，从而尽量减少程序存在的缺陷。

文档	测试	测试内容 / 目的
功能需求	系统测试（又称综合测试）	最终成品严格按照需求运行
非功能需求	性能测试、压力测试（又称负荷测试）	使用者使用时感觉不到压力
基本设计	集成测试（内部集成 / 外部集成）	各功能之间是否正常协作
详细设计	UT 或单元测试	当前功能是否严格符合要求
（实现）	（试运行）	功能是否如开发者预期正常实现

这个时候，要注意测试设计的编写是否顺利。因为测试的实施项目要以各文档内记录的内容以及正常状态为依据来编写，所以这一过程中能明显看出文档的好坏。测试设计的编写轻松

且顺利，证明文档质量高，反过来如果编写测试设计时频频遇到难题，则说明文档不合格。要是文档中多处出现 13.2.1 节提及的问题，最好重新审查一遍文档。

编写测试设计时常出现的问题有以下几个。

【问题】

- 不知该写什么，写多少
- 测试设计的时间不够用
- 文档记录的内容有问题
- 没有可用作依据的输入文档

如果问题出在文档自身，那么只需要像上面所说的那样，在开始编写测试设计之前重新查看并审查文档即可解决问题。就算没有充足时间，也尽量不要在没更新文档的情况下直接根据最新状态编写测试设计。因为这样一来我们就失去了客观说明测试设计的依据。更新文档可以让所有相关人员及时共享最新信息，因此千万怠慢不得。

接下来我们逐个解决剩下的问题。

◉ 明确测试的目的

如果测试的项目只求详细，那想写多少项目都能写出来（当然也因文档而异）。同样，项目想删怎么都能删掉。所以在设计测试时要注意“必须保证的点（= 测试的目的）”，区分重要的项目和不重要的项目。

测试设计既要完成其在所属流程内应当完成的所有任务，又不能重复记录属于其前后流程的项目。一定要清楚自己编写的测试设计所负责的阶段以及需要完成的任务。

另外，对于用文字难以表述的测试设计，用另附表格的方式往往能轻松地表述清楚。需要记录多种情况时，可以在测试设计文本中写操作流程，然后另附表格来说明各个情况，这样既可以让文档看起来干净利索，又能有效防止遗漏。总而言之，就是不要拘泥于定式，选择最符合该项目测试的形式来编写测试设计。

还有，一个项目内要确认的内容必须限制在一个。比如在“点击按钮后信息写入数据库”中，“生成报告”和“各项目的保存内容正确无误”就要分开来写。这样，在实施时能更轻松地判断是 OK 还是 NG。

【写什么，写多少】

- 注意该测试内必须保证的点（= 测试的目的），区分主干和枝叶
- 不写属于其他测试中要保证的项目（防止重复）
- 难以用文字表述的另附表格
- 一个项目内需确认的内容限制在一个

◉ 从重要的部分开始写

测试设计是基于文档的，那么一旦文档编写的进度落后，势必会压缩测试设计的时间。很多时候，留给测试设计的日程安排都是非常紧的。

这里重点需要注意的地方和上面有些类似，也就是“写什么”“写多少”“能不能不写”的问题。越是时间紧的时候，越需要仔细阅读文档，然后从重要的部分开始写测试设计。如果动手之前不先读一遍，写到一半肯定会遇到瓶颈。因为我们先写了重要的部分，所以最后就算时间不够用了，我们也能保证重要的部分不出问题。如果真的完全没有时间可用，至少也要把需要检查的观点和方针逐条记录下来，这必然会对实施阶段有所帮助。

【 时间不够用的时候 】

- 仔细阅读文档
- 从重要的部分开始写
- 就算没时间做测试设计，也要将观点和方针逐条记录下来

◉ 利用所有可以利用的资源

在规模较小或人数较少的开发现场，不编写文档、文档写出来不更新导致形同虚设的情况时有发生。

在这种情况下，我们需要一边向相关负责人咨询所需信息一边推进测试设计。如果此时已经有了能够勉强运行的程序，最好一并拿来作参考。咨询来的信息一定要记录下来，哪怕只是草草作个笔记也好，这些可以共享的资料在后续阶段必然会用得到。每次咨询之后都要做个总结，这能避免我们为了同一件事反复去询问负责人。

在没有文档可用的情况下，如果能认真地编写测试设计，测试设计将成为最后进行规格核对时的重要资料。

【 没有文档的时候 】

- 观察已能运行的程序，掌握大致情况
- 必要信息直接咨询相关负责人
- 把咨询到的信息记录下来，汇总保存
- 要比有文档时更用心地编写测试设计

总而言之，编写测试设计时要注意我们上面提到的所有问题，同时综合考量程序的健壮性、发布前的日程安排等因素，在此基础上选择与测试各阶段方针相符的粒度，根据所选粒度最终完成测试设计的编写。

还有，由于需求变更时必须同时修改测试设计，所以测试设计要尽量选用方便添加、删除操作的格式，免得修改时遇到麻烦。

13.2.3　测试的实施与测试阶段的轮换（做什么，做多少）

本部分所讲的问题通常出现在正式开始测试的时候。实施测试时需要同时准备相关文档以及测试设计。

开始实际测试程序时常出现的问题有下面几个。

【问题】

- 没时间实施所有测试
- Bug 太多，测试难以进行
- 不清楚在发现 Bug 时如何报告
- 没有测试设计

下面我们来对这些问题进行逐条改善。

◉ 实施测试时也要有优先顺序

有些时候，我们不缺文档和测试设计，但交付期前紧迫的时间让我们无法完成数量庞大的全部测试项目。从整个项目的优先顺序来讲，人们往往会先照顾交付期，把执行全部测试项目放到其次。遇到这种情况我们应该怎么办呢?

首先，要确认交付期是否真的无法延后。缩短测试期意味着可以保证的项目相应减少，这很有可能导致发布后出现 Bug 和安全问题，直接影响信誉。

我们需要从整个项目的角度出发，重新认真衡量一下得失，确认交付期是否真的优先于上述几点。

如果考虑到项目整体平衡而需要删除某些测试项目，那么在选择项目前最好先按照以下条件对测试项目进行梳理及分类。

- 已经完成的 / 尚未完成的
- 不完成就无法发布的 / 就算没完成也可以删除而不影响发布的
- 必须详细进行测试的 / 大概确认一下就可以的

以上述分类标准为参考，给各项目设置优先顺序如下。

- 等级 A：已经实现，且不完成就无法发布的功能
- 等级 B：尚未实现，但不完成就无法发布的功能
- 等级 C：大概确认一下就可以的 / 实在不行删掉也可以的功能

A 和 B 是发布所需的最低限度的功能，因此如果它们的测试不够充分，我们连最低限度的功能都无法保证。这两类项目的测试时间一旦无法保证，必须同交付方进行交涉。

如果无论如何都无法延期，则需要将测试能细分的细分，不能细分的调整粒度（= 缩小保证范围，添加限制。此时需要保证整个团队达成共识）。

C 部分就是与时间赛跑，先把这些测试按优先级排成一条线，然后逐个实施。这里要做好心理准备，因为排在后半部分的项目多数时候没时间实施。

确定优先顺序时要避免模糊不清的等级制，一定要严格排序。这是为了保证那些必不可少的测试能够先执行完。这里一旦用了模糊不清的等级制，等到火烧眉毛的时候往往会出现“全都是 A”的情况，或者冒出来“AA 级”“S 级”之类原先根本不存在的等级。这就使当初的分级失去了意义。确定优先顺序是为了给测试项目划分界限（这里界限的标准不是“哪些需要保证”，而是“哪些可以不用保证”），以备万不得已之时可以将没必要的东西砍掉，因此决不能有半点模糊。

制定方针、共享方针之后，剩下的就是跟时间赛跑，在剩余时间内做到最好了。

关于优先顺序再补充一点，性能、压力测试等非功能测试要在环境具备一定条件时尽早实施。因为如果拖到一部分功能完成后再做，出现问题时需要排查的范围将会很大，修正 / 重新测试的周期也会很长。

另外，这些测试要在测试实施的各阶段定期进行。这听起来虽然与前面的言论自相矛盾，但要知道，非功能需求和功能需求一样，有些问题只有在集成之后才会显现出来。

【时间不够用的时候】

- 确认是否真的无法延期
- 确定优先顺序，先测试必不可少的项目
- 优先级较低的项目排成一条直线逐个执行（万不得已时直接放弃）
- 非功能需求的测试要尽早地、持续地实施

◉ 问题过多时先着眼大局

有时我们会遇到这种情况：开始测试之后，程序到处都是毛病根本无法运行，或者按日程表来说理应完成的功能却不能用，结果导致安排好的测试项目无法推进。这种时候就要暂时放下测试，先把已经完成的功能和未完成的功能分个类，做成清单共享给团队。

实现阶段的进度一旦与认知出现偏差，很可能发展成影响整个项目进度的问题，因此需要尽早做到信息共享。如果功能已经实现，但是品质方面存在问题，则需要确定问题的优先顺序并逐个检查。出现问题时眼光不能局限在一个功能内，要尽量站在全局角度。

① 安全方面的问题
② 功能是否正常运行
③ 数据的显示、读写是否正常

④ 设计方面的问题

测试时，先像上面这样大致列出几个重点，把当前功能中优先级高的项目测试完，然后暂时跳过优先级低的项目，直接开始下一个功能的测试。等到所有优先级高的项目测试完后，再回过头来逐个解决优先级低的项目。

【缺陷过多的时候】

- 区分有缺陷的功能和未实现的功能，将未实现的部分做成清单共享出来
- 给要实施的测试项目制定优先顺序
- 目光不局限在一个功能内，测试尽量照顾到整体

◉ 缺陷报告要简洁、详细

如果在测试过程中发现缺陷，应该如何报告呢？如果只说“无法运行”“运行不正常”，负责修改的人也无从改起。所以要尽量加入一些详细信息。

- 实施时间
- 实施环境
- 实施者（负责实施的人，或者当时登录系统的用户等）
- 实施意图
- 出现的问题（与预想结果的偏差）

这些都是在重现问题或者检查问题现状时必不可少的信息。将实施时间加入报告是为了让负责人能通过程序日志进行调查。因为有些问题只有在特定时间才会发生。

报告里还要尽量留下证据。另外，执行测试时的输入值、执行前后数据库的状态最好也保存下来。这些能在发生问题时大幅减轻调查的压力。能否还原发生问题时的情况，直接影响着解决问题的速度。

遇到那些觉得不对劲，但是又无法根据文档判断是否为缺陷的问题时，也要作出报告。借助整个团队的力量解决问题要远比一个人伤脑筋来得快。另外需要注意，出现这种难以判断的情况时，意味着设计阶段很可能潜藏了大问题。

【发现缺陷的时候】

- 尽量详细记录情况并报告
- 尽可能地留下证据
- 无法判断是否为缺陷时，不要急着独自下判断，先报告

◉ 即便没有测试设计，也要保证方针共享

根据项目规模和发布前的日程安排，我们有时候会大胆放弃测试设计，或者由于时间太紧没能完成测试设计。

学习测试设计的时候我们学习过，当剩余时间真的不够用的时候，至少要保证观点或方针明确。在此基础上，再根据实施时的优先顺序，从一旦发生缺陷可能引发严重问题的地方开始测试。如果开始测试后才发现问题比想象中复杂，难以解决，至少要把需确认的项目逐条记录下来。

【 没有测试设计的时候 】

- 确认、共享观点和方针
- 优先处理最可能引发问题的地方
- 对于太过复杂难以测试的地方，逐条列出需确认的项目

由于实施测试的阶段位于整个项目的后半部分，所以往往需要与时间赛跑。但即便时间不够用，我们也要有计划地推进测试，力求提高效率，用最少的劳力换来最大的效果。

13.3 小结：测试并不可怕

本章我们讲了在项目早期阶段导入测试观点能带来的几点改善。

说得笼统一点，就是确定一个正确的状态，检查当前状态与正确状态之间有多少偏差，然后考虑怎样做能修正这些偏差。至于思路和需确认的事项等，很多都可以直接拿到编程的过程中重复利用。测试能够担保的东西其实并不多，充其量不过是在有限的范围内做个保证，比如“能防止这类错误”“这种情况在预料之中”等，不可能保证完全的正确性。测试能够证明“这个缺陷已被修复”，却不能证明“这里已经没有缺陷了”。

另外，设计者、实现者都是活生生的人，难免犯错和疏忽。所以，无论我们在开发过程中多么谨慎，也不可能将出现问题的可能性降为零。

测试要有计划性。无计划的测试不但浪费时间，获得的成效也低。相反地，有计划的测试能帮助我们提高程序的品质。由于往往到了项目后半期，测试才受到重视，所以人们的注意力常会被实现代码的过程吸引，忘记去关注测试。但是我们希望各位能在项目的初期就去有意识地兼顾测试，放开胆子与测试携手同行，从而提高开发的品质。