第七章 测试与改错

编程大师说:"任何一个程序,无论它多么小,总存在着错误。"

初学者不相信大师的话,他问:"如果一个程序小得只执行一个简单的功能,那会怎样?"

"这样的一个程序没有意义,"大师说,"但如果这样的程序存在的话,操作系统最后将失效,产生一个错误。"

但初学者不满足,他问:"如果操作系统不失效,那么会怎样?"

"没有不失效的操作系统,"大师说,"但如果这样的操作系统存在的话,硬件最后将失效,产生一个错误。"

初学者仍不满足,再问:"如果硬件不失效,那么会怎样?"

大师长叹一声道:"没有不失效的硬件。但如果这样的硬件存在的话,用户就会想让那个程序做一件不同的事,这件事也是一个错误。"

没有错误的程序世间难求。[James 1999]

错误是一种严重的程序缺陷。测试的目的是为了发现尽可能多的缺陷,并期望通过 改错来把缺陷统统消灭,以期提高软件的质量。但关于测试与改错实在没有什么高明的 方法值得大书特书,也不能表现出程序员的聪明才智。相反地,它们带来了更多的牢骚 与痛苦。因此在教学和开发实践中,测试与改错总是被当作万般无奈的工作踢到角落里。

医生可以把他的错误埋葬在地下了事,但程序员不能。我们必须要学会测试与改错, 并且把测试与改错工作做好。

7.1 对测试的理解

测试的道理并不深奥, 计算机专业人员都应该明白。但就是这么简单的事, 计算机专业的博士们也未必都已经理解。

有一天,一位比我聪明,编程比我快,学习能力比我强的计算机专业博士生恭恭敬敬地请我坐好,并且史无前例地削了苹果请我吃,为的是向我请教"软件工程"问题。你必定以为这位仁兄好学之极。非也,我和他同事三年来从未探讨过"软件工程"问题。只因为他明天要去应聘,参加面试,生怕被人问倒,就央我当晚为他恶补一把"软件工程"。他还特地问我"什么是白盒测试和黑盒测试?应该由谁来执行?"(有公司曾经这样面试应聘者)当我解释完测试的道理时,他叹了一口气说:"这些玩意儿我读大学十年来都没搞过,怎么能讲得出道理来。唉,就去碰碰运气吧。"我有"兔死狐悲"的感觉。我们这一群博士生三年来尽干些自欺欺人的事,到毕业时学问既不深也不博。个个意志消沉,老气横秋。长此以往,总有一天招聘会的大门前将贴出标语"博士与狗不得入内"。

以下是关于测试的几个重要观念。

7.1.1 测试的目的

测试的目的是为了发现尽可能多的缺陷。

这里缺陷是一种泛称,它可以指功能的错误,也可以指性能低下,易用性差等等。测试总是先假设程序中存在缺陷,再通过执行程序来发现并最终改正缺陷。理解测试的目的是个很重要的意识问题。如果说测试的目的是为了说明程序中没有缺陷,那么测试人员就会向这个目标靠拢,因而下意识地选用一些不易暴露错误的测试示例。这样的测试是虚假的。

目前高校的科技成果鉴定会普遍存在类似的虚假现象。我在读硕士时就亲身经历过这样的事。我们的项目是研究集成电路制造过程中的成品率问题。当时国内大多数工厂的集成电路成品率只有百分之几,我编写的示例程序可以将集成电路的成品率优化到98%。示例效果是如此的好,以致一位评委(某厂的总工程师)不无讽刺地说:"采用你们的成果,我们可要发大财了。"这个项目就轻易地通过了鉴定,并且不久后获得了电子工业部科技进步二等奖。这就象在考试时通过作弊取得了好成绩而被表扬。我那时尚且纯真,羞愧之余,不禁对高校科研成果的水平和真实性大失所望(现在我已不再失望,因为很少抱希望)。

一个成功的测试示例在于发现了至今尚未发现的缺陷。

测试并不仅是个技术问题, 更是个职业道德问题。

7.1.2 测试的心理要求

测试主要是由人而不是由机器执行,这就不免与心理因素相关。为了测试的真实性,对测试的心理要求是"无情"。这似乎太残酷了。开发人员不能很好地测试自己的程序是因为做不到无情。而测试人员如果做到了无情却会引起开发人员的愤怒,遭人白眼。

尽管已经明白了测试的目的是为了发现尽可能多的缺陷,但当测试人员真的发现了 一堆缺陷时,却不可乐颠颠地跑去恭喜那个倒霉的开发者,否则会打架的。

7.1.3 测试的真理

测试只能证明缺陷存在, 而不能证明缺陷不存在。

这个真理告诉我们,对于一个复杂的系统而言,无论采取什么样的测试手段都不能证明缺陷已经不复存在。"彻底地测试"只是一种理想。在实践中,测试要考虑时间、费用等限制,不允许无休止地测试。

7.1.4 测试与质量的关系

测试有助于提高软件的质量,但是提高软件的质量不能依赖于测试。测试与质量的关系很象在考试中"检查"与"成绩"的关系。

学习好的学生,在考试时通过认真检查能减少因疏忽而造成的答题错误,从而"提高"了考试成绩(取得他本来就该得的好成绩)。

而学习差的学生,他原本就不会做题目,无论检查多么细心,也不能提高成绩。 所以说,软件的高质量是设计出来的,而不是靠测试修补出来的。

7.2 测试人员的选择

测试需要开发人员参与吗? 测试需要独立的测试小组吗? 测试需要用户参与吗?

让我们先看一看 Microsoft 公司关于测试的经验教训,再回答上述问题。

7.2.1 Microsoft 公司的经验教训

在80年代初期,Microsoft 公司的许多软件产品出现了"Bug"。比如,在1981年与IBM PC 机一起推出的 BASIC 软件,用户在用".1"(或者其他数字)除以10时,就会出错。在FORTRAN 软件中也存在破坏数据的"Bug"。由此激起了许多采用 Microsoft操作系统的 PC 厂商的极大不满,而且很多个人用户也纷纷投诉。

Microsoft 公司的经理们发觉很有必要引进更好的内部测试与质量控制方法。但是遭到很多程序设计师甚至一些高级经理的坚决反对,他们固执地认为在高校学生、秘书或者外界合作人士的协助下,开发人员可以自己测试产品。在 1984 年推出 Mac 机的 Multiplan (电子表格软件)之前,Microsoft 曾特地请 Arthur Anderson 咨询公司进行测试。但是外界公司一般没有能力执行全面的软件测试。结果,一种相当厉害的破环数据的"Bug"迫使 Microsoft 公司为它的 2 万多名用户免费提供更新版本,代价是每个版本 10 美元,一共化了 20 万美元,可谓损失惨重。

痛定思痛后, Microsoft 公司的经理们得出一个结论:如果再不成立独立的测试部门,软件产品就不可能达到更高的质量标准。IBM 和其它有着成功的软件开发历史的公司便是效法的榜样。但 Microsoft 公司并不照搬 IBM 的经验,而是有选择地采用了一些看起来比较先进的方法,如独立的测试小组,自动测试以及为关键性的构件进行代码复查等。Microsoft 公司的一位开发部门主管戴夫·穆尔回忆说:"我们清楚不能再让开发部门自己测试了。我们需要有一个单独的小组来设计测试,运行测试,并把测试信息反馈给开发部门。这是一个伟大的转折点。"

但是有了独立的测试小组后,并不等于万事大吉了。自从 Microsoft 公司在 1984 年与 1986 年之间扩大了测试小组后,开发人员开始"变懒"了。他们把代码扔在一边等着测试,忘了唯有开发人员自己才能阻止错误的发生、防患于未来。此时,Microsoft 公司历史上第二次大灾难降临了。原定于 1986 年 7 月发行的 Mac 机的 Word 3.0,千呼万唤方于 1987 年 2 月问世。这套软件竟然有 700 多处错误,有的错误可以破坏数据甚至摧毁程序。一下子就使 Microsoft 名声扫地。公司不得不为用户免费提供升级版本,费用超过了 100 万美元。[Cusumano 1995]

7.2.2 测试人员的分工

从 Microsoft 公司的教训中可知,公司内部对产品的测试 (称为α测试),需要开发人员与独立的测试小组共同参与。开发人员应该执行"白盒"测试,即测试源程序的逻辑结构以及实现细节 ("白盒"是指看得见程序的内部结构)。而独立测试小组应该执行"黑盒"测试,即按照规格说明来测试程序是否符合要求 ("黑盒"是指看不见程序的内

部结构)。比如在测试一个模块时,"白盒"测试方法要对模块的所有代码进行单步跟踪测试。而"黑盒"测试方法只需测试模块的接口是否符合要求,它关心程序的外部表现而不是内部的实现细节。

小型的软件公司可能没有条件设立独立的测试小组,也有可能测试小组人员不多而 忙不过来。这时,可以让开发小组的成员相互测试对方的程序。

这里要强调的是, a 测试不能依赖于开发人员或者测试小组中的任意一方,必须是双方共同参与。"白盒测试"必须由开发者自己执行,因为别的测试人员无法了解到程序的内部实现细节。而"黑盒测试"必须由独立的测试人员执行,因为开发者难以做到客观、公正。开发者在测试自己的程序时存在一些弊病:

- (1) 开发者对自己的程序印象深刻,并总以为是正确的(自信是应该的)。倘若在设计时就存在理解错误,或因不良的编程习惯而流下隐患,那么他本人很难发现这类错误。
- (2)开发者对程序的功能、接口十分熟悉,他自己几乎不可能因为使用不当而引发错误, 这与大众用户的情况不太相似,所以自己测试程序难以具备典型性。
- (3)程序设计有如艺术设计,开发者总是喜欢欣赏程序的成功之处,而不愿看到失败之处。让开发者去做"蓄意破坏"的测试,就象杀自己的孩子一样难以接受。即便开发者非常诚实,但"珍爱程序"的心理让他在测试时不知不觉地带入了虚假成份。

软件产品正式发行前,在公司外部邀请一些用户对产品进行测试,称为β测试。β测试的涉及面最广,最能反映用户的真实愿望,但花费的时间最长,不好控制。一般地,软件公司与β测试人员之间有一种互利的协议。即β测试人员无偿地为软件公司作测试,定期递交测试报告,提出批评与建议。而软件公司将向β测试人员免费赠送或者以很大的优惠价格发行软件的正式版本。

7.3 测试的主要内容与常用方法

有一次文学考试,问高尔基是哪国人。一考生乐极而吟:"尔基啊尔基,你若不姓高,我怎知你是中国人。"这是一种瞎猜法。如果这种方法用于软件测试,人累死也测不出什么结果来。

不论是对软件的模块还是整个系统,总有共同的内容要测试,如正确性测试,容错性测试,性能与效率测试,易用性测试,文档测试等。"白盒测试"是指开发人员从程序内部对上述内容进行测试,而"黑盒测试"是指独立的测试人员从程序外部对上述内容进行测试。很多软件工程教材讲述了各种各样的测试方法并例举了不少示例[Pressman 1997] [Sommerville 1992] [杨文龙 1997]。本节简明地讲述常用的测试方法及其道理。

7.3.1 正确性测试

正确性测试又称功能测试,它检查软件的功能是否符合规格说明。由于正确性是软件最重要的质量因素,所以其测试也最重要。

基本的方法是构造一些合理输入,检查是否得到期望的输出。这是一种枚举方法。倘若枚举空间是无限的,那可惨了,还不如回家种土豆有盼头。测试人员一定要设法减少枚举的次数,否则没好日子过。关键在于寻找等价区间,因为在等价区间中,只需用

任意值测试一次即可。等价区间的概念可表述如下:

记(A, B)是命题 f(x) 的一个等价区间,在(A, B)中任意取 x1 进行测试。

如果 $f(x_1)$ 错误, 那么 f(x) 在整个 (A,B) 区间都将出错。

如果 $f(x_1)$ 正确,那么 f(x) 在整个 (A,B) 区间都将正确。

上述测试方法称为等价测试,来源于人们的直觉与经验,可令测试事半功倍。

还有一种有效的测试方法是边界值测试。即采用定义域或者等价区间的边界值进行测试。因为程序员容易疏忽边界情况,程序也"喜欢"在边界值处出错。

例如测试 $f(x) = \sqrt{x}$ 的一段程序。凭直觉等价区间应是(0, 1)和(1, $+\infty$)。可取 x=0.5 以及 x=2.0 进行等价测试。再取 x=0 以及 x=1 进行边界值测试。

有一些复杂的程序,我们难以凭直觉与经验找到等价区间和边界值,这时枚举测试 就相当有难度。

在用"白盒测试"方式进行正确性测试时,有个额外的好处:如果测试发现了错误,测试者(开发人员)马上就能修改错误。越早改正错误,付出的代价就越低。所以大多数软件公司要求程序员在写完程序时,马上执行基于单步跟踪的"白盒测试"。

7.3.2 容错性测试

容错性测试是检查软件在异常条件下的行为。容错性好的软件能确保系统不发生无法意料的事故。

比较温柔的容错性测试通常构造一些不合理的输入来引诱软件出错,例如:

- (1) 输入错误的数据类型,如"猴"年"马"月。
- (2) 输入定义域之外的数值,上海人常说的"十三点"也算一种。

粗暴一些的容错性测试俗称"大猩猩"测试,除了不能拳打脚踢嘴咬,什么招术都可以使出来。这里我举不出例子,因为我没有对程序粗暴过,并且这辈子也不打算学会粗暴。

7.3.3 性能与效率测试

性能与效率测试主要是测试软件的运行速度和对资源的利用率。有时人们关心测试的"绝对值",如数据送输速率是每秒多少比特。有时人们关心测试的"相对值",如某个软件比另一个软件快多少倍。

在获取测试的"绝对值"时,我们要充分考虑并记录运行环境对测试的影响。例如 计算机主频,总线结构和外部设备都可能影响软件的运行速度;若与多个计算机共享资 源,软件运行可能慢得像蜗牛爬行。

在获取测试的"相对值"时,我们要确保被测试的几个软件运行于完全一致的环境中。硬件环境的一致性比较容易做到(用同一台计算机即可)。但软件环境的因素较多,除了操作系统,程序设计语言和编译系统对软件的性能也会产生较大的影响。如果是比较几个算法的性能,就要求编程语言和编译器也完全一致。

性能与效率测试中很重要的一项是极限测试,因为很多软件系统会在极限测试中崩溃。例如,连续不停地向服务器发请求,测试服务器是否会陷入死锁状态不能自拔;给程序输入特别大的数据,看看它是否吃得消。

7.3.4 易用性测试

易用性测试没有一个量化的指标,主观性较强。调查表明,当用户不理解软件中的某个特性时,大多数人首先会向同事、朋友请教。要是再不起作用,就向产品支持部门打电话。只有30%的用户会查阅用户手册。[Cusumano 1995]

一般认为,如果用户不翻阅手册就能使用软件,那么表明这个软件具有较好的易用性。

7.3.5 文档测试

文档测试主要检查文档的正确性、完备性和可理解性。好多人甚至不知道文档是软件的一个组成部分。

正确性是指不要把软件的功能和操作写错,也不允许文档内容前后矛盾。

完备性是指文档不可以"虎头蛇尾",更不许漏掉关键内容。有些学生在证明数学题时,喜欢用"显然"两字蒙混过关。文档中很多内容对开发者可能是"显然"的,但对用户而言不见得都是"显然"的。

文档不可以写成散文、诗歌或者侦探、言情小说,要让大众用户看得懂,能理解。 很多程序员能编写出好程序,却写不出清晰的文档。不要说自己以前语文学得差, 现在已没救了,找借口不是办法。没有人天生就能写出好程序,都是练出来的。同理, 若第一次写不好文档,就多写几次文档,慢慢地就会写出好文档来。我上大学前不会说 普通话,不会写作文,现在我极能说会写,当个秘书或书记已绰绰有余。

7.4 改 错

在软件测试时如果发现了错误,必须请程序员改错,否则测试工作就白干了。

改错是个大悲大喜的过程,一天之内可以让人在悲伤的低谷和喜悦的颠峰之间跌荡起伏。如果改过上万个程序错误,那么少男少女们不必经历失恋的挫折也能变得成熟起来。

我从大三开始真正接受改错的磨练,已记不清楚多少次汗流浃背、湿透板凳。改不 了错误时,恨不得撞墙。改了错误时,比女孩子朝我笑笑还开心。

在做本科毕业设计时,一天夜里,一哥们流窜到我的实验室,哈不拢嘴地对我嚷嚷: "你知道什么叫茅塞顿开吗?"

我象白痴似的摇摇头。

他说:"今天我化了十几个小时没能干掉一个错误,刚才我去了厕所五分钟,一切都解决了。"

他还用那没洗过的手拉我,一定要请我吃"肉夹馍"。那得意劲儿仿佛同时谈了两个 女朋友。

在本节,我要替程序员们总结关于改错的几点思想方法:

(1)要有勇气。东北有个林场工人,工作勤奋,一人能干几个人的活。前三十年是伐树 劳模,受到周总理的接见。忽有一天醒悟过来,觉得自己太对不起森林,决心补救错误。 后三十年成了植树劳模,受到朱总理的接见。此大勇也。

程序中的错误只有开发者自己才能找出并改掉。如果因畏惧而拖延,会让你终日心情不定,食无味,睡不香。所以长痛不如短痛,要集中精力对付错误。

(2)不可蛮干。都说急中生智,我不信。我认为大多数人着急了就会蛮干,早把"智" 丢到脑后。不仅人如此,动物也如此。

我们经常看到,蜜蜂或者苍蝇想从玻璃窗中飞出,它们会顶着玻璃折腾几个小时,却不晓得从旁边轻轻松松地飞走。我原以为蜜蜂和苍蝇长得太小,视野有限,以致看不见近在咫尺的逃生之窗,所以只好蛮干。可是有一天夜里,有只麻雀飞进我的房间,它的逃生方式竟然与蜜蜂一模一样。我用灯光照着那扇打开的窗户为其引路,并向它打手势,对它说话,均无济于事。它是到天亮后才飞走的,这一宿我俩都没息好。

(3)找出错误的根源。有人问阿凡提:"我肚子痛,应该用什么药?"阿凡提说:"应该用眼药水,因为你眼睛不好,吃了脏东西才肚子痛。"

我们应该运用归纳、推理等方法尽早确定错误的根源。

(4) 在改错之后一定要马上进行重新测试,以免引入新的错误。有人在马路上捡到钱包后得意忘形,不料自己却被汽车撞倒。改了一个程序错误固然是喜事,但要防止乐极生悲。更加严格的要求是:不论原有程序是否绝对正确,只要对此程序作过改动(哪怕是微不足道的),都要进行重新测试。

7.5 小 结

优秀的程序员敢于声称自己的代码没有错误,这种自信让人羡慕不已。一个错误自身也许很微小,但是程序存在错误这件事很严重。能否做好测试与改错工作,思想认识和办事态度是最关键的。

程序员应该把测试当成份内之事,不要依赖于外界的"黑盒测试"。"黑盒测试"就 象通过提问题来判断一个人是否是个疯子,但无法知道他为什么成了疯子。让程序员对 所有的代码执行单步跟踪测试听起来很费时间,但习惯了你就感觉不到有什么不方便。 单步跟踪测试将使你以后的日子更轻松。

程序出了错误一定要改错,但是"编写优质无错"的程序才是根本的解决之道。在此,我竭力建议大家阅读 Steve Maguire 著的《Writing Clean Code: Microsoft Techniques for Developing Bug-free C Programs》(有中文译本,[Maguire 1993])。我深受此书的教诲,恭益非浅。