大型金融产品维护及开发的综合审查

赵书祥　王文展　(博科公司 100071)

摘要:银行核心系统具备运行时间长、运行环境相对稳定、业务知识比较专业、系统比较复杂、用户量大等特点,对于银行软件系统的开发和维护,从需求分析、设计、编码和测试的审查是提高软件开发质量和效率的有效方法,审查应当以有效为目标,子项目负责人作为审查员的重要角色,对所负责的子系统架构和规范负责。一个好的软件系统需要健全有效的文档系统为基础。

关键词:银行核心系统 软件维护 审查 复审 子系统 子系统负责人

1 引言

审查(Inspection)、走查(Walkthrough)和复审(Review)是软件过程中为了提高软件开发效率和质量的重要手段,国内外很多公司已经采用,并为审查的列表做了很详尽的说明。

为了发现缺陷,人们采用了包括同级评审(Peer Review)在内的多种项目评审。为了确保被审查文档的质量,应该邀请不同评审者从各自的角度进行检查。

评审的目标不仅是捕捉错误,收集改进的建议,而且帮助作者在未来创建更好的工作产品。另外,由于评审的进行使大量人员对软件系统中原本不熟悉的部分更为了解,因此,评审还起到了提高项目连续性和培训后备人员的作用。

2 审查和测试

审查不同与传统的白盒测试,并不是实际地去运行程序,也不是进行案例覆盖,大部分审查工作在系统尚不能运行之前,就已经开展。

在 G. J. Myers 的经典著作《软件测试技巧》中,给出了测试的定义:"程序测试是为了发现错误而执行程序的过程"。而审查不需要执行程序,所以从狭义的角度来说,审查不能算作测试。但从宏观来说,审查和测试一样,目的都是发现程序的错误,提高软件的质量。

测试是传统软件开发过程的核心,如果将测试定义为用不同方法去发现程序错误,提高软件质量的过程,那么审查倒是可以算作测试的一种方法。

测试可视为分析、设计和编码 3 个阶段的"最终复审",在软件质量保证中具有重要地位。为了确保软件的质量,较理想的做法应该是对软件的开发过程,按软件工程各阶段形成的结果,分别进行严格的审查。

3 审查的必要性

3.1 审查的作用

(1) 互相学习,相互搭接。银行系统的维护组的一个成员可能对若干子系统比较熟悉。如果在每次新业务开发时,都派不同的人去参加这个审查过程,专家提出的建议和设计方案都是相当有价值的,可以比较精辟地概括一个新子系统的经脉。这对于整天接客户电话的维护人员来说是非常重要的。

实践证明,一个定期参加本人和他人工作的正式和非正式复审的程序员,他的经验的积累速度是那些单独工作的程序员的三倍。

(2) 提高软件生产率。维护工作的审查对维护工作的生产率提高是很明显的,这是因为维护审查有以下作用:

- 减少了上机运行的次数
- 减少了对失败运行的分析次数
- 减少了改动上的改动
- 未来改动上的复杂程度会降低
- 减少了生产上的改动,减少了为修改文件而反复运行的次数。

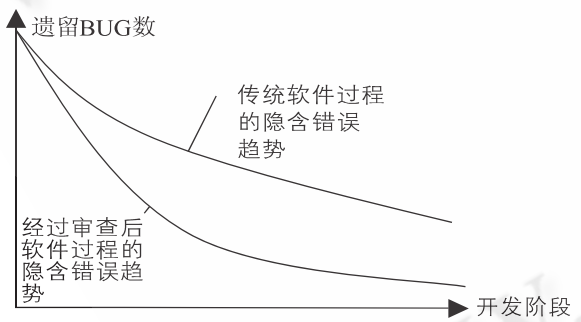
据资料介绍,实例测算结果为,审查活动之后,在项目开发工作的头 6 个月里,产品崩溃的现象减少了 77%。一个软件公司比较了产品在审查前后的两种现场问题报告,发现问题减少了 5/6。

(3) 提高员工参与意识和主动性。在每一个开发

和维护项目中,前期的审查工作都包含收集优化思想的趋势。系统的优化思想不是只有专家才能提出来的,一些有头脑的员工一样能提出非常不错的建议。作为子系统负责人和构架工程师可以担负收集这些思想的角色,提高员工的参与意识和工作积极主动性。如果一个员工的努力是向着他所想象的方向进展,可以相见,他会全身心地投入并对这个方向负起责任来。

(4) 改变软件隐含错误发展趋势。传统软件工程,强调测试的地位,经过单元测试,集成测试,系统测试,验收测试,系统实际运行,系统的遗留 BUG 在逐渐减少,经过用户的长期使用,可以假设隐含 BUG 趋近于 0。而经过严格审查的软件则会在开发的早期,隐含错误就会达到很低的水平,这两条曲线的差别在于审查的效率和测试的效率。如果审查的过程只是走过场,而测试是比较认真的,那么两条曲线可能靠得非常贴近,那么审查仅仅是浪费时间的,领导可以拿来当反面教材的依据。

如果审查做得非常有效,大量的错误会在软件早期,比如单元测试之前就已经排除了,根据这样的数据做出的两条曲线,将会相差很大,而且传统测试曲线在很长时间内都无法接近下曲线。



3.2 后续开发和维护问题

作为银行软件,特别是核心系统,后续开发和维护是繁多而漫长的过程,少则几年,多则十几年、几十年。原有设计经过很多人的修改和维护,已经面目全非。如果没有合理的审查机制,后续开发和维护会使系统变得越来越难以维护。仅仅代码的理解和阅读就将是一件让人望而生畏的事情。

系统维护过程中,对代码作少量的修改是很频繁的事情。调查显示,修改的代码的错误率远高于新写代码,修改错误的同时引入了更多的错误,补丁程序可能带来新的补丁,仅回归测试一项的工作量就很难估算。

4 审查的方法

4.1 角色分析

一般审查委员会建议 3 –7 人,如果增加相关人员或新员工,则不在此列。角色包括审查领导、记录员、读者、作者等。但对于银行系统的后续开发、系统优化、系统维护这样的软件操作来说,审查人员可以包含以下角色:

(1) 系统架构师。对于一个银行软件系统,涉及到的业务总是交织在一起,而且有些业务有历史原因,有些还有地域的差别。同时,如果核心业务系统和其他业务系统相关联,情况就更加复杂。这些情况很难由一个普通的人员所掌握,对于总体上能够把握修改范围以及影响面规模的尺度的人员,是不可或缺的。

(2) 规范师(代码、文档等)。代码和文档的作者将注意力放在程序的正确性和结构的清晰上,但无法面面俱到,但一些细节可能会对后来的读者造成影响。作为代表文档和代码的所有读者的人员,需要能够客观、规范地约束文档和代码的格式,规范的代码和文档将形成一种文化传承下去。

(3) 子系统构架师。一个大型的软件系统往往分为很多子系统,几个大的模块。子系统的负责人要为自己的子系统的架构、逻辑、业务功能等负责,并同时出具修改对子系统的影响和修改和测试规模。

(4) 业务代表。业务代表一般会参与某一需求的所有会议和讨论,他将拥有较为准确、全面的需求及更改说明。作为客户的代言人,业务代表应当对需求的准确性负责。审查是对软件系统本身的很专业的会议,不适于客户参加。

(5) 实际生产运行代表。一个系统往往在很大范围内使用,各机构的使用情况对于软件部门来说未必了解得很全面。一个或多个、现场或远程的审查人员是需要的。实际生产科技管理部门,往往拥有最贴近客户的需求和最实效的方案,他们往往提出最具价值的建议,他们能避免在生产上引起大麻烦的设计。他们的参与主要集中在需求分析和设计的审查。

(6) 项目负责人。项目负责人的职责是督促会议的有效按时进行,并为审查安排时间。如果项目无法为审查安排时间,就应当向上级部门报告当前状况。

(7) 接口涉及的系统代表。如果项目跨越了软件系统,或者修改影响了其他系统,影响到的系统应当有

代表参与所有阶段的审查,在审查之前要阅读审查的文档对象,并和本组内的其他人讨论自己不确定的因素。代表要为所代表的系统负责,并将可能对自己所代表的系统造成的影响限制在一定范围内,并提请项目负责人的注意。

当然,还有可能需要其他角色的参与。比如,如果系统的改造涉及到的测试环境问题还需要由测试代表进行审查。

4.2 审查及技术复审流程

(1) 需求和功能分析审查。需求的提出往往由客户方业务人员提出,提出时未必经过缜密的思考和实际生产较为全面透彻地调研,而且需求提出后,需求文档的语言对于软件人员来说,理解上可能会有较大的差距。软件人员根据需求描绘出开发完成后的目标模样,让需求方确认,方可实施软件开发。

软件人员作出功能和需求分析后,系统架构师,子系统负责人,业务代表,实际生产运行代表应当作详尽的审查,这是至关重要的一步,对于工期和成本影响相当大,对于成本准确估算也是关键的一步。如果范围控制不好,会造成一连串的修改和测试,会大大增加成本,而对系统功能的实现却不会有很大提高。

(2) 总体设计审查。如果需求和功能分析开展得很成功,设计审查可以拿上述分析的审查结果作为依据,因为这个审查结果中凝聚了专家们的智慧。

在设计审查过程中,子系统负责人将扮演非常重要的角色。对于一个大型银行系统的开发及维护优化的设计,一般规模都与子系统相当。或者跨越几个子系统。与其他系统相联系的项目就可能修改接口子系统的代码。子系统负责人就是使一个模块的设计尽量满足各子系统的设计规范和最优配置。

(3) 详细设计的审查。一般说来,详细设计的一句说明,在生成代码后,大约对应 3 – 10 行代码。按照这个习惯,审查人员一般可以估算出详细设计是否到位。对于过于粗略的设计,编码的过程中会离原有的设计和需求有较大的偏差,详细设计的审查对软件的控制将不会特别有效。

但是,一个过于详尽的设计将会是一种浪费。而且也不便于以后的阅读。不便于阅读的东西从实际上来说是没有用的。对于这样的设计,读者会说,不如直接看代码算了,这样的设计在审查完后就变成了实际的废品。

当然并非一定有这样的对应关系,特别是对关键程序的修改,一句代码在程序中往往起关键作用,影响数条路径。对于这样的修改的说明,应当分几个方面说明。比如:

修改原因

影响的业务规则

影响的程序名称列表

影响的数据结构列表

修改后的目标

这些对测试案例的编写能起到很重要的作用。

(4) 代码的组内评审。软件一旦形成代码,就意味着前半部分的工作尘埃落定,需求上、设计上、架构上、接口上都不能再有改动。如果仍然需要这些改动就意味着需求变更、设计审查的不足。要在这个时候弥补,就意味着浪费。特别是需求的改动常常牵连设计的改动、设计的审查工作再次安排。

代码评审只对编码过程中的错误进行检查,当然,也有可能从代码审查的时候看到架构上的缺陷、需求上的不足、设计上的不足,不过仍然比在测试阶段找到这些问题值得庆幸。如果有些项目的过程没有包含前期的审查的工作,代码审查就一定要实施。让代码审查作为“最终审判”者。

对于银行大系统来说,商业语言往往简化了编程的风格。可以将这些东西放到较为次要的位置,重点放在程序的流程和功能的完备上。特别是无效的业务判断,很难测试的点,环境难以搭建、难以多次测试的点上。往往会出现这样的现象,一个程序一眼看出的错误,交付用户的时候还未发现。或者有些错误在生产上造成重大影响的错误。代码的审查应当对这些负责。

(5) 测试方案和测试案例的审查。测试的目的就是提高软件的质量,测试方案决定了测试的效率。业务代表和系统架构师应当出席测试方案的审查。

测试人员一般会参考设计文档和需求文档。作为测试负责人和主测试员,应当参与需求分析审查、设计审查,主要是学习业务的知识。

审查人员中,可以允许客户代表参加。熟悉业务的客户对目标系统最为关注,测试的重点一般能够较好地把握。

(6) 投产前的文档检查。银行系统的用户和开发者一般关系比较密切,在一套软件开发完成后,产品必然会包含大量的文档。下发是一个很正式的过程,下

发之前应当以各种渠道,将写好的文档交相关的客户部门代表进行审查,避免发生误解和疏忽的内容。如果产品下发后相关人员无法理解或进行误操作,结果往往需要软件人员来处理。

上述相关人员包括产品安装,产品使用培训,产品维护等人员。

4.3 审查根据错误的级别进行重视程度划分

一般资料将错误按严重性可分为 4 类:

(1) 死机。系统崩溃或挂起;

(2) 致命。使系统不稳定、或破坏数据、或产生错误结果,而且是常规操作中经常发生或非常规操作中不可避免的;

(3) 严重。系统性能或响应时间变慢、产生错误的中间结果但不影响最终结果,如:显示不正确但输出正确;

(4) 一般。界面拼写错误或用户使用不方便。银行软件对于不能运行或产生错误数据是不能容忍的,因为是数省乃至全国使用的系统,面对的是可能上亿的银行客户,带来的直接是经济损失和名誉损失。审查应当把一部分注意力集中于对环境依赖的死代码,或许在测试环境中能通过,但可能造成无法运行的地方着重研究。

在每级审查中都不能忽视严重性很高的错误,作为质量的最后门槛,测试案例必须设置改变环境并考虑版本因素的案例,以最大限度保证严重问题不在生产上出现。

就是因为上述问题的严重性,导致审查人员往往忽视一个很重要的问题——柜员操作的方便性。银行柜员数量巨大,培训难度很大。一个需要很繁杂步骤,或者按键很别扭的操作,往往在投产后在生产上存在数年之久,而这样的痛苦又无法有效地反映到银行的需求部门,提起需求部门的重视。而对于软件人员来说,这往往不会是一个问题。

4.4 文档整理

一个软件产品,特别是大型软件产品,文档非常重要,如果一个完好的文档可以比较容易转换为代码,而从代码看出原有设计却相当困难。

文档往往会有多个拷贝,和程序一样,版本问题就凸现出来,控制版本的方式可以通过所有相关人员都访问一个资源库,经过权限控制来修改这个资源库。读者不要再去访问本机所存储的文档。直接访问资源库的文档肯定能获得最新的规格表。邮件只用于发送链接,最好使用网站本身的提醒功能和认证机制,将任何次修改都记历史记录,历史记录一般不允许修改和删除。

实时更新就意味着实时发布,每个相关人员获得文档更新的通知后,自然会去查看。而作者知道肯定会有人去查看,自然会对自己编写的文档负起责任来,并大大提高文档的质量和信息含量。

同时,文档系统不应作为考核依据,因为考核必然会引起最有价值的文档的内容贬值,作者会更注重形式,而不是内容。

4.5 子系统负责人

本文提出子系统负责人,主要是降低公司和项目风险而提出的一个概念。专家,一般是 爱学习或能力很强的人,他们往往知道得比较多。但这样的人不能给予太多的希望,会让你很失望的,他们没有太多时间培训新人,同时又要被公司委以重任,常常在关键时候无法在场,甚至一个比较急的问题也没有时间解决。

要从机制上减少对专家的依赖,需要增加文档的管理,同时增加子系统的负责人角色。在一个大型银行系统里,内容往往无法被一个人掌握。但大型银行系统可以根据业务范围和软件结构,分为几个相对独立的子系统。子系统之间使用接口通讯。各子系统有各自系统的文档系统。子系统负责人负责整理、规范、汇编、归档、发布这些文档,并负责子系统本身结构的合理性。

参考文献

1 《Design and code inspections to reduce errors in program development》 IBM SYSTEMS JOURNAL, VOL15, NO 3, 1976; ? 1976, 1999 M. E. Fagan.

2 《Using multiple adaptive regression splines to support decision making in code inspections》 Systems and Software 73 (2004) 205 – 217 Lionel C. Briand, Bernd Freimut, Ferdinand Vollei.

3 《Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products, 3rd ed》. Daniel P. Freedman, Gerald M. Weinberg.

4 《Lessons Learned in Software Testing》 Cem Kaner James Bach Bret Pettichord.