论软件维护及软件可维护性

摘要：

本人2010年有幸主持并参与了国内某银行小额信贷综合系统的开发研制工作，作为技术经理主要负责需求分析、概要设计及核心模块的设计实现等工作。该系统主要面向中小企业及个人客户，为其提供融资贷款服务。由于银行项目软件通常有较长的维护周期，并且要求有效地控制维护成本和维护风险，因此需要采用多种措施来提高软件的可维护性。本文结合作者实践，讨论了软件维护的三种类型，改正性维护、预防性维护和完善性维护的特点，阐明了影响软件可维护性的三个主要因素，即分析阶段要尽可能地模块化设计，高内聚低耦合；缺陷预防需要新技术新工具的支撑；变更控制要通过UCM方式来进行闭环管理。本文结合该项目的开发过程，详细介绍了在项目中所进行的软件维护活动，为提高软件可维护性所采取的措施，并基于这些评价了实施过程中的效果。最后总结了项目在软件维护及可维护性方面取得的成绩和需要改进的地方。

正文：

众所周知，当前以中小、小微企业及个人为主的小额借款人“融资难，融资贵”的问题仍然比较突出，为解决这个问题国家出台了一系列政策措施鼓励和支持银行业开展小额信贷等创新型金融服务。 2010年5月，我所在单位受国内某银行委托开发研制该行小额信贷综合业务系统。 该系统主要为中小企业及个人客户提供无担保无抵押的信贷资金。借款人可以通过网上银行、电子邮件、手机短信、电话及邮寄等多种途径申请贷款。系统收到贷款申请后，先通过外部征信系统获取到借款人的信用信息，并基于信用信息对其进行风险评级，给出参考额度和贷款利率，再由客户经理通过系统进行复审及放款。贷款发放后，风险经理可以通过系统对借款人的还款情况进行实时监控和跟踪。与此同时，该系统还可以针对不同情况给出相应的催收建议及催收措施，从而帮助银行回收贷款，降低不良率。该系统主要包括网上银行、风险评级、授信审批和贷后监控部分，具有客户管理、账务管理、合约管理、催收管理和报表管理等功能。 我作为单位的技术骨干，担任该项目技术经理一职，主持并参与了该项目，主要工作有：需求分析，概要设计，核心模块的设计实现，协助项目经理制订项目计划。由于银行软件系统通常生命周期较长，尤其是软件的维护周期较长，往往还需要一个小型团队专职进行项目的维护工作。为了能够有效控制维护成本，提高软件的可维护性，我们团队在开发工程中采取了一系列方法措施。下面我将详细论述这些方法措施在软件维护方面的作用和意义。 通常比较常见的软件维护类型有：改正性维护、预防性维护和完善性维护。通俗来讲，改正性维护即修改软件交付后发现的缺陷，主要通过提供修复后的新版本、修复脚本或者修复补丁等方式。预防性维护是指在软件开发过程中预防缺陷的产生，是一种防患于未然的措施，正确地进行缺陷预防能够很好地降低软件的开发机维护成本。完善性维护是指由于用户在使用软件的过程中，业务需求发生了变化，因而需要对软件系统的功能进行变更的一种维护，这种维护手段不仅要求软件自身有较好的扩展性，同时也要求从管理的角度做好变更的跟踪和控制。 我认为影响软件可维护性的主要因素有三点，首先，系统在分析设计阶段是否很好地遵循了规范，有没有使用成熟可靠的通用构件，系统构架是否扩展性强，模块设计是否高内聚松耦合。其次，缺陷预防的投入是否足够，有没有使用新技术、新工具来提高效率和质量。再次，变更控制管理是否到位，有没有做到问题闭环管理，对于代码的修改是否有评审监控，对于修改后的版本是否有效管理。我们在实际开发中使用了多

种技术手段、管理手段来落实以上三点。 1. 尽可能多地使用成熟、稳定的通用构件来增强系统的可靠性、可维护性和可扩展性。 在本项目开发过程中，我们考虑到银行会根据国家政策的调整来调整风险评级的规则，会根据央行的基准利率和计算方法来调整利息的计算方法。如果每一次调整都需要修改源代码，重新编译、打包、回归测试、部署，势必会影响到维护效率，缩短系统的在线时间。我们引入了 Drools 业务规则引擎和 Groovy 脚本引擎来解决这一问题。Drools 引擎可以把一系列风险评级的推理判断规则封装在一个规则文件中，该文件是一个文本文件，可以方便地对其进行修改编辑。值得一提的是，Drools 规则文件的语法使用自然语言风格，即领域定义语言（DSL），这样业务人员也可以轻松地阅读、编写、评审规则文件。Groovy 脚本引擎可以动态地执行数学公式，将计算结果返回给主程序。公式中所用到的参数变量可以由业务人员通过 Web 页面在系统中自行设置，而无需开发人员修改程序或者修改数据。Groovy 脚本引擎在计算时，会将业务人员定义的参数以键值对的方式代入公式，从而达到灵活计算的效果，正是由于采用了 Drools 业务规则引擎和 Groovy 脚本引擎，我的团队才能有效地提高了系统的可维护性，即当业务规则和利率计算公式发生变化时，业务人员可以不依赖于开发人员和系统管理员的参与，也无需修改程序，重新部署新版本等繁琐步骤就能完成系统的修改升级，大大地提高了系统的在线时间。 2. 使用新技术、新工具进行缺陷预防。 我们认为在软件开发过程中有效的缺陷预防胜于疲于奔命的缺陷修复，因此我们引入了一系列新技术、新工具来进行缺陷预防。首先推行持续集成技术，即采用 Jenkins 持续集成引擎，每隔 15 分钟对项目代码进行一次构建，并自动执行单元测试用例，每隔 2 小时执行一次集成测试用例、代码静结构分析和代码覆盖率分析，并将结果通过邮件发送给项目成员。其次，优化开发人员的集成开发工具，1）使用 JUnit 做单元测试；2）使用 Emma 做代码覆盖率测试；3）使用 Findbugs 和 PMD 做代码静结构分析；4）使用 Checkstyle 做代码规范检查。与此同时，测试人员使用 EasyB 和 Selenium 来开发自动化回归测试用例，并将其与 Jenkins 引擎集成。质量控制人员使用 Sonar 对代码质量、规模和设计进行度量。这样就可以使团队成员从多种角度对项目质量把关，进而使整个系统的设计、开发、集成、测试过程透明可见，真正地做到了让项目在“阳光”下进行，从而有效地达到了进行缺陷预防的目的，大大地降低了缺陷逃逸率。但是另一方面，有项目成员抱怨，太多的检查预防措施耽误了开发进度，连项目经理也会有相同的顾虑。我通过和公司另一项目 A 做了对比来说服他们，A 项目基本不做预防检查，表面上看似很快的进度却在系统测试阶段搁浅了，反反复复地“回归”，以至于模块、代码千疮百孔、缝缝又补补、补丁摞补丁，甚至在上线后发现了严重缺陷，造成了公司和客户的损失。项目组成员听取了我的建议，不但没有了抱怨，更是从严、从深地积极做好缺陷防范工作，正是由于我们的项目在开发过程中就夯实了基础，才在验收测试及试运行是无任何故障，受到客户的好评。 3. 使用 UCM 变更控制 我们在项目开发过程中使用 IBM Rational ClearCase 和 ClearQuest 对变更和缺陷实施 UCM 管理。在项目维护期，闭环的变更和缺陷管理显得尤为重要。所谓 UCM，是指客户提出的变更申请和缺陷进行统一管理，包括分类，优先级评定和状态跟踪等。在我们项目中，客户通过 ClearQuest 提出变更请求，待项目经理及变更委员会批准后，新需求会分配给相应的开发人员。开发人员只可以根据 ClearQuest 的任务签出相应模块的程序代码进行修改，而对于其它不相关的代码，他们则无权限访问。开发人员完成代码修改和提交后，变更请求流转至测试人员处进行复测和回归测试，最后经用户确认后，才由发版经理发布部署。整个流程可管理、可追踪、有记录并与版本控制系统无缝集成。 4. 结束语 综上所述，我们在项目开发过程中采用成熟文档的通用构件，并引入新技术、新工具进行缺陷预防，同时结合 UCM 变更控制管理等一系列手段来提高系统的可维护性，有效地降低了软件的维护难度和维护成本。自系统上线至今未发现任何严重缺陷，系统稳定可靠，得到了客户和公司领导的肯定。同时，我也发现了一些不足之处，例如，如何平衡缺陷预防、质量控制和项目进度的关

系。我相信我和我的团队可以再不断地尝试、摸索和总结中解决好这一问题。我在日后的工作中会更努力地提高专业技术水平，为我国的软件事业贡献出自己的力量。