源代码分析注释的质量评价框架①

高晓伟[1,2], 杜 晶[1], 王 青[1]

[1](中国科学院软件研究所 互联网软件技术实验室, 北京 100190)
[2](中国科学院大学, 北京 100190)

摘 要: 在如今的软件开发中, 开源软件的使用越来越普遍, 但是对大型开源软件的理解和维护仍然是一项复杂的工作. 开源软件通常缺乏完善的文档和注释, 想要完整的理解开源系统难度较大, 研究界产生了一种通过分析大型开源软件的源代码, 进而深入理解系统, 发现和修复系统漏洞的软件分析型任务. 源代码分析注释是软件分析型任务的一项重要产出, 它是一种以注释形式存在的细粒度代码分析报告, 数量庞大, 难以快速做出质量评价. 在传统的软件质量评价中, 对注释的评价通常局限于覆盖度和文本长度, 不能满足源代码分析注释质量评价的要求. 为了更好的评价源代码分析注释的质量, 本文结合现有的对代码注释质量评价的研究以及信息质量领域的评价方法, 提出了一种综合考虑客观质量属性和主观质量属性的质量评价框架. 结合实际的项目数据分析, 本文的方法可以更有效的检测出注释中的冗余以及无关内容, 发现相关质量问题, 从而对源代码分析注释进行更全面的质量评价.

关键词: 代码分析注释; 质量度量; 质量分析; 注释分析

Quality Evaluation Framework of Source Code Analysis Comments

GAO Xiao-Wei[1,2], DU Jing[1], WANG Qing[1]

[1](Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Science, Beijing 100190, China)

Abstract: The application of open source software is more and more popular these days, but maintenance and full understanding of huge open source system is still a hard problem. The documentations and comments that are useful for open source system understanding is limited. A new type of software analysis task emerged in research field. The source code analysis comment is an important product of software analysis task. It is a type of code analysis documentation existing like code comments. Usually the amount of source code analysis comments is huge, and it is hard to evaluate its quality in limited time. Traditional quality evaluation method of code comments, which only measure comment length or comment ratio, is too simple. It's no longer suitable for source code analysis comments' quality evaluation. Based on existing code comment quality evaluation methods and some existing works in information quality, this paper propose a more complete quality evaluation framework for source code analysis comments, taking both objective quality and subjective quality into consideration. Data gathered from real projects show that the new framework is more suitable for finding quality issues and evaluating the quality of source code analysis comments.

Key words: code analysis comments; quality measurement; quality analysis; comment analysis

1 引言

开源运动的兴起, 对推动软件产业的发展起到了举足轻重的作用. 在这个大趋势的推动下, 如今的软件开发很多都不再是从零做起, 而是充分利用现有的比较成熟的开源组件快速搭建系统, 然后逐步完善和定制. 快速理解、定制或者组合开源系统已经成为不亚于编写代码本身的一项重要技术能力.

开源软件的部署和运行往往比较简单, 但是当需

① 基金项目:国家自然科学基金(91318301,91218302,61432001);核心电子器件、高端通用芯片及基础软件产品项目(2012ZX01039-004)
收稿时间:2015-01-20;收到修改稿时间;2015-03-18

要对开源系统定制开发或者修复缺陷时, 就需要对系统本身有比较深入的理解. 开源软件由于其开发模式的特殊性, 通常缺乏完善的文档和完整的注释. 尤其是对于大型的开源软件系统, 规模庞大而且逻辑复杂, 依靠个人的力量很难对系统有比较全面的理解, 于是在研究界产生了一种通过深入、细致的分析大型开源软件的源代码, 从而发现和修复现有系统中的漏洞的软件分析型任务.

本文的研究对象是源代码分析注释. 它是这类软件分析型任务的一项基础产出, 是代码分析人员通过分析软件的源代码添加的注释, 用来帮助其他注释阅读者更好的理解该系统. 代码分析工作是其他工作的基础, 源代码分析注释的质量将影响整体分析成果的质量, 进而最终影响整个软件分析型任务成果的价值. 在软件质量评价中, 软件作为一个整体来评价质量, 评价过程中更为关注的是软件的设计和源代码本身, 对于注释通常只关注注释的覆盖度以及注释的文本长度. 而源代码分析注释是一项独立的分析成果, 用评价代码注释的方法来评价源代码分析注释的质量具有很大的局限性. 一方面, 源代码分析注释不仅仅包含文本, 还会包含一些分析图, 超链接等非文本信息, 这类信息非常有价值, 但是不会被传统的评价方法所考虑; 另一方面, 覆盖度和文本长度主要是一种工作量层面的度量, 并没有深入注释的内容, 而对分析注释而言, 最重要的是分析注释对注释的阅读者是否有帮助, 需要对注释内容进行分析; 第三, 源码注释是随代码编写的过程零散产生的, 而分析注释是伴随分析过程, 以一种相对集中的方式产生, 它不再是代码的附属品, 而是可以协作、维护的开放式分析成果. 因此对源代码分析注释的质量需要设计更全面的评价框架和评价方法.

本文结合软件质量评价、信息质量评价等领域的研究成果, 提出了一种适用于源代码分析注释的质量评价框架. 该框架从客观和主观两个维度来评价源代码分析注释的质量, 平衡了质量评价的代价和准确度. 同时, 该框架中不仅仅考虑评价传统代码注释的覆盖度、注释长度这样工作量型的指标, 而且会深入注释内容以及注释的产生过程进行综合评估. 通过结合实际案例数据分析, 本文的方法被证明可以快速发现注释中存在的问题, 并且更全面的评价注释质量.

2 相关研究

2.1 软件质量评价

软件质量评价方面的研究由来已久, 自20世纪60年代爆发软件危机以来, 软件质量问题越来越突出, 并逐渐成为软件工程领域的一个研究重点.

1968年Rubey和Hurtwick最早提出了软件度量学的概念, 从此开启了软件度量学领域的研究[1]. 随后, Boehm在1978年首次提出了层次化的质量评价模型[2], 它的划分方法也被沿用到后来的ISO/IEC 9126系列标准中. ISO/IEC 9126中将软件质量属性定义为6个质量特性和27个子特性, 6个质量特性分别为: 功能性、可靠性、易用性、效率、维护性和可移植性. 2005 年以来, ISO/IEC 又发布了 ISO/IEC 25000 系列标准, 作为 ISO/IEC 9126 的接替, 它将软件产品的质量扩充为 8 个质量特征和 31 个子特征.

在 ISO/IEC 9126 中, 软件产品的质量可以通过测量内部属性、外部属性或者使用质量的属性来评价. 内部属性主要通过测量中间产品的内在性质, 外部属性主要通过测量代码执行时的行为. 在传统质量评价中, 注释的质量主要影响软件质量的内部属性中的维护性. 在实践中, 最常用的注释相关的评价指标是注释的覆盖度和注释文本的长度.

不同于软件的代码, 注释文本不可以被执行, 也无法进行自动化的测试, 所以针对注释的质量评价方法需要结合注释的特点重新设计. 对本文要研究的源代码分析注释而言, 它不再像代码注释是作为软件的一部分来评价质量, 而是成为评价主体, 作为一种单独的软件分析成果来评价质量. 参照软件质量评价中的代码注释评价方法, 如果仅仅考虑注释的覆盖度和文本长度, 对源代码分析注释的质量评价而言, 显得非常不完整.

2.2 代码注释相关研究

良好的代码注释是规范的软件开发的要求, 代码注释通常用来记录对代码的描述, 降低开发人员之间的沟通成本, 提高软件的可维护性. 在源代码注释中, 被研究最多的是注释的演化以及注释与代码一致性的问题. [3]中以三个开源软件为例分析了代码注释随代码的演化情况. [4],[5],[6]中给出了一些检测注释和代码不一致的方法, [6]中还充分利用注释和代码的约束关系发现了 Linux 内核中的 Bug.

针对代码注释质量的研究, 目前还不是很多. 文

献[7]中设计了一种自动化评价 Javadoc 注释质量的工具 JavadocMiner, 此研究主要的分析对象是行内注释. 文中提出的方法从自然语言处理和注释、代码的一致性两个角度分析了注释的质量. 文献[8]扩展了质量元模型 QMM, 提出了基于活动的可维护性质量模型, 将 QMM 进一步分解为实体和属性, 使得元素之间的关系更加清晰. 文献[9]扩展了文献[8]中提出的质量模型, 并将其应用在了代码注释的质量评价上. 文中设计了相关性、有用性、完整性和一致性四个维度来评价注释质量, 并针对文中提出的代码和注释相关度以及注释长度的两点假设采用问卷调查的方式进行了验证.

上述方法都属于启发式的方法, 对数据的依赖性比较强. 对评价源代码分析注释而言, 这些度量手段都具有一定的局限性, 但是这些质量特征, 对设计源代码分析注释的质量评价框架有一定的借鉴意义.

2.3 信息质量相关研究

源代码分析注释不同于代码注释, 它不是代码的附属品, 而是一项单独的分析成果. 从编辑模式来看, 源代码分析注释属于一种基于协作的用户生成型学习资源, 它是一组分析人员共同工作的成果. 对于源代码分析注释而言, 相比工作量方面的评价指标, 注释使用者更关心的是注释是否有用, 也就是注释的信息质量.

研究人员对信息质量的认识也是不断深化的. 早期, 研究人员认为信息质量是信息描述客观事物或者事件的准确程度[10]. 随后, 基于用户质量认知的信息质量评价逐步成为主流. Wang 指出要提高信息质量, 首先要理解信息质量对于信息消费者而言意味着什么[11]. Naumann 和 Rolker 认为应当综合考虑用户感知、信息本身和信息的访问过程来评价信息质量[12].

丁敬达[13]整理了信息质量领域的相关研究, 并提出了维基百科词条信息质量的启发式评价框架. 维基百科也属于典型的基于协作的用户生成型学习资源, 它的质量评价方法对我们评价源代码分析注释质量有比较大的启发意义.

3 质量评价框架

3.1 框架描述

对于软件质量的评价, 需要从功能性、可靠性、易用性、效率、维护性和可移植性等方面考虑. 评价软件质量时, 通过执行测试、统计故障情况和运行时间就可以分析出软件是否满足功能性、可靠性和效率等方面的要求. 但是对于源代码分析注释而言, 它无法像软件代码那样通过执行测试来评估质量, 对它质量评价的核心在于注释的内容是否对阅读注释的人理解代码有帮助, 在于评价它的信息质量. 文献[11], [12]对信息质量的研究启发我们评价信息质量时既要考虑信息本身的客观质量, 也要考虑用户对信息的感知质量. 因此, 我们在设计源代码分析注释的质量评价框架时, 也包含了两个方面. 一方面是信息本身是否详实, 如注释的数量是否充分, 覆盖面是否足够, 注释的内容是否表述清晰等, 这方面的评价结果是可以比较客观的获取的; 另一方面是注释阅读者本身对注释有什么样的信息需求, 如阅读者本身想通过注释了解哪些信息, 解决什么问题, 这方面的评价结果相对主观, 需要与阅读者的目的和要求结合起来考虑.

另外, 因为源代码分析注释是在一种开放协作的环境下产生的知识成果, 通过分析注释的产生过程, 也对我们分析最后的制品结果是否可信, 有重要参考意义. 如[14]中就研究了编辑次数、编辑人数、合作强度对维基百科页面信息质量的重要性. 因此, 在设计源代码分析注释的质量评价框架时, 我们也考虑了过程维度的质量属性.

基于前面的考虑, 本文设计了如图 1 所示的源代码分析注释质量评价框架. 框架首先将质量属性分为客观和主观两大部分, 客观质量属性是可以通过搜集相关度量因子直接评价的质量属性, 主观质量属性需要结合具体的评价目的和评价要求来度量和评价. 对客观质量属性, 框架又分为制品维度和过程维度. 制品维度的评价对象是最终产生的源代码分析注释成果本身, 过程维度的评价对象是产生源代码分析注释的过程. 客观质量属性是客观存在的, 不会随评价人的变化而变化, 所以相对而言, 客观质量属性比较容易被自动化的提取. 主观质量属性是评价人对注释质量的主观评价, 具体的评价度量策略和方法, 会随评价对象不同以及评价目的不同而发生变化. 可以看出, 质量评价的过程是一个评价代价与评价准确度平衡的过程. 在客观质量属性的评价中, 要做到尽量的自动化, 降低评价的代价, 快速得出评价结果. 而对于主观质量属性的评价, 因为源代码分析注释的数量通常比较庞大, 主观质量属性的评价通常采用数据抽样或者结合专家经验打分、学习和建模的方式, 相对而言

代价较高.

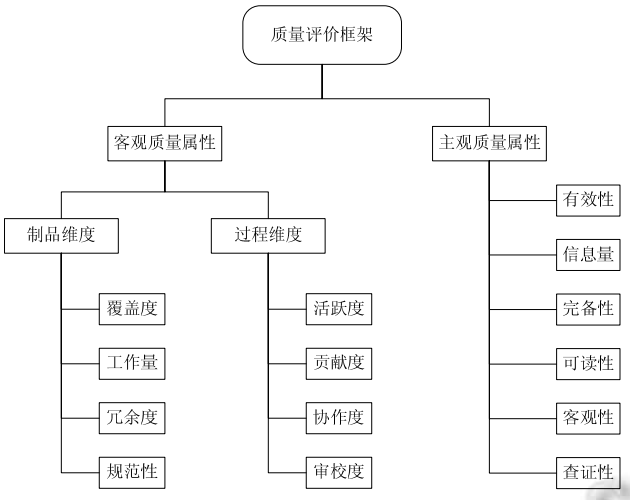

图 1 质量评价框架

框架中一共包含十四个质量属性, 下面分别解释每个质量属性的含义:

覆盖度: 覆盖度是不同类型的可注释入口被覆盖的情况. 例如, 总共有 100 个可注释的函数入口, 已注释函数 30 条, 那么对这批注释而言, 函数注释的覆盖度就是 30%. 覆盖度越高, 注释阅读人员可以使用的参考范围就越大.

工作量: 工作量是对分析人员完成这一批注释所投入的工作量多少的评价. 具体而言可以体现为注释的数量、注释的文本内容是否丰富, 有多少图片、超链接等形式更复杂的注释.

冗余度: 冗余度评价不同注释间内容的冗余程度. 冗余度是对源代码分析注释数量方面评价的进一步深化, 高质量的注释不仅仅要求数量比较充分, 还要求不要有过量的信息冗余. 在代码注释评价中不会评价冗余度, 因为代码注释是开发人员辅助开发用的, 一般不会存在冗余的注释, 但是源代码分析注释则不同, 尤其是在外包的软件分析型项目中, 接包方有时迫于时间压力会在注释中填上内容完全一样或者内容大致相仿的注释, 这样的注释往往多是没有意义或者信息量很低的, 因此需要冗余度来甄别.

规范性: 规范性用来评价注释整体的风格一致性程度, 包括信息格式的一致性, 语义前后的一致性等. 格式一致性可以通过分析注释的内容结构、格式、引用等是否一致.

活跃度: 活跃度是分析人员的编辑行为随时间的变化情况, 直观而言就是看注释是比较均匀地生成的, 还是在短期内大量添加的.

贡献度: 贡献度评价所有分析人员对注释的贡献情况, 是少数分析人员完成了绝大多数注释, 还是注释任务量在所有注释人员中得到了比较合理的分配, 即考虑注释工作量的人员多样性.

协作度: 协作度评价分析人员的协作情况, 主要关注不同分析人员对注释的编辑修改情况, 度量方式考量注释被多少分析人员编辑.

审校度: 审校度评价注释被审阅和完善的程度, 主要关注注释被修订和完善的情况, 不论是单个分析人员完成还是多个分析人员完成的, 度量方式考量注释被编辑的次数和被查看的次数.

有效性: 有效性是指根据评价人对注释指定的有效性评价标准, 对注释做出是否有效的判定, 如发包方可能规定字数少于一定数量的注释为无效注释.

信息量: 信息量是注释携带的信息多少, 一定意义上代表了注释对阅读者的帮助程度. 信息量综合考量注释的内容结构、描述方式、是否使用了图片、超链等形式更复杂的呈现方式等.

完备性: 完备性是指对于评价人觉得注释应该涵盖的各项信息要素, 注释是否完备. 例如对于函数注释, 通常应该包含对函数功能, 函数参数, 函数返回值, 函数可能出现的异常情况, 函数被调用情况的描述.

可读性: 可读性用来评价注释是否表达清晰、逻辑连贯、主题明确.

客观性: 客观性是指注释信息无偏见、中立的程度, 不要过分掺杂注释者的主观意见.

查证性: 查证性是指注释提供的信息可查考、可验证的程度, 如果是引述的其他参考资料中的信息, 应在参考资料中列出.

3.2 质量属性分析方法

3.1 节中介绍了质量框架的构成, 在执行具体的质量评价时, 还需要通过具体的分析方法来得到各个质量属性的度量结果.

本文结合源代码分析注释数据及其编辑过程数据的特点给出了 10 种分析方法, 表 1 所示为各质量属性与分析方法的对应关系.

数量分析和分布分析是对注释整体情况进行分析. 这两项分析会梳理注释路径, 得出不同目录以及不同

类别注释的数量和分布情况. 注释分类的标准可以按注释粒度分为模块注释、文件注释、函数注释和变量注释, 也可以结合具体的评价场景制定分类标准. 数量分析和分布分析主要用来评价注释的覆盖度和工作量.

表 1 质量属性对应的分析方法

	覆盖度	工作量	冗余度	规范性	活跃度	贡献度	协作度	审校度	有效性	信息量	完备性	可读性	客观性	查证性
数量分析	●	●												
分布分析	●	●												
文本长度分析		●							●					
内容结构分析			●	●					●	●	●	●		
词组分析									●	●		●	●	
引文分析														●
非文本信息分析		●							●	●				
人员分析					●	●	●	●						
编辑行为分析					●	●	●	●						
浏览行为分析								●						

文本长度分析是统计注释文本中与内容相关字符的字符数, 这里需要过滤内容无关的字符, 如一些富文本结构化标签, HTML 标签等. 文本长度是最容易获取的度量指标, 传统的代码注释质量通常使用文本长度来评价. 文本长度的评价方式虽不够精确, 但是通常比较有效.

内容结构分析是根据注释文本, 对注释内容进行标准化和结构化的过程. 通过分析文本, 得到注释常用的描述句式, 进行模板抽取, 同时也获得注释的注释元素, 进而对注释内容结构化. 内容结构分析会产出冗余注释检测结果、差异评分结果以及结构化的注释内容. 冗余注释检测结果可以用来评价冗余度, 差异评分结果可以用来评价规范性, 结构化的注释内容可以用来评价有效性、信息量、可读性、完备性.

词组分析是首先通过自然语言处理的方法对注释文本进行分词以及词性标注, 然后对注释语句的词性构成、词性分布进行分析, 以及对注释整体进行词频分析, 中心词、主题词提取等. 词组分析的结果可以用来评价注释的有效性、信息量、可读性和客观性.

引文分析主要针对比较长的、有引述外部参考资料的注释, 用来评价注释的查证性. 通过分析引文确保一些重要结论有据可依, 引文分析目前主要只能通过人工判别的方式执行.

非文本信息分析主要处理图片、超链接等非文本注释信息, 最简单的处理方式是统计注释中图片和超链接的个数, 深入分析还可以分析图片的大小, 图片元素的多少以及超链接的链接关系等.

人员分析、编辑行为分析、浏览行为分析是从注释的编辑和浏览数据中提取信息. 所有的编辑信息可以被整理成为(注释路径, 编辑人, 编辑时间)这样的三元组, 浏览信息整理成(注释路径, 浏览时间)这样二元组, 然后进一步进行分析统计, 就可以得到完成注释的分析人员构成, 编辑行为和浏览行为随时间的变化情况等信息, 进而评价活跃度、贡献度、协作度和审校度.

4 案例研究

4.1 数据背景

本文选取的案例数据来自核高基课题. 课题将整个 Linux 及 Android 源码的分析任务分成了若干子任务, 以众包的方式分包给其他相关科研院所和有关企业共同完成. 各子任务承担单位需要完成自己负责的模块及目录的源代码分析工作, 其中源代码分析注释是一项重要的验收成果. 课题会周期性的考核, 检查各子任务的完成情况.

课题为源代码分析工作搭建了专门的源代码分析平台, 分析人员可以一边查看源代码一边撰写分析注释, 通过页面上的富文本编辑器可以添加注释信息, 以及插入图片、超链接等. 分析注释通过源代码分析平台进行收集, 收集的注释数据通过经过定制的 Mediawiki 系统进行管理, 可以保存不同的历史版本及编辑信息, 数据文件保存在 Mysql 数据库中. 本文选取了两组涵盖整个注释任务周期, 而且数据总量最多的 Linux 内核模块的源代码分析注释作为案例数据进行研究, 注释总数分别为 12193 和 15262 条.

4.2 结果分析

首先我们根据 LXR(The Linux Cross Reference)的分类方法对注释数据进行了分类, 然后我们按照 3.2 中提到的分析方法对注释数据进行了分析. 通过分析数据我们发现, 两组注释中占比最大的四类注释相同, 均为函数定义、变量定义、宏定义和结构体成员注释, 占比总和均超过 90%. 可见对函数和变量的理解是理解源代码分析最基础的工作, 我们后面的质量分析也主要围绕这四种注释类型进行.

表 2　客观质量属性–制品维度

注释类型		覆盖度	工作量					冗余度	规范性
		注释覆盖度(%)	注释数	注释数占比(%)	平均文本长度	图片数	超链接数	冗余占比(%)	差异分
A	①	32.42	6678	54.77	135.42	322	194	13.93	28.3
	②	37.24	2311	18.95	62.77	7	66	45.91	26.98
	③	15.15	1328	10.89	61.7	0	156	48.12	16.14
	④	11.52	1124	9.22	48.29	0	8	17.53	18.18
B	①	43.95	9027	59.15	177.89	3131	25086	8.67	18.59
	②	40.63	2410	15.79	61.4	4	2541	40.75	13.75
	③	16.86	1212	7.94	36.27	0	7536	46.95	7.41
	④	17.43	1270	8.32	111.88	0	8075	54.41	11.14

① = 函数定义；② = 变量定义；③ = 宏定义；④ = 结构体成员

4.2.1 客观质量属性–制品维度

表 2 是客观质量属性–制品维度的度量结果. 从表中可以看出, 覆盖度上 B 组注释整体高于 A 组注释, 但两组注释中函数定义和变量定义的注释覆盖度均高于[15]中指出的开源软件源代码注释的平均覆盖度 18.67%. 工作量上, 两组注释都将最主要的注释量投入在了函数定义和变量定义上. 注释数量方面, B 组注释的注释数量比 A 组注释的多; 文本长度方面, B 组注释在函数定义和结构体成员注释上明显超越 A 组注释, 从图片数和超链接数上也可以看出, B 组注释的注释投入了相当的工作量为函数定义注释添加流程图和调用关系, 因此从工作量上看, B 组注释的注释参考意义更强. 冗余度上, B 组注释的冗余度略低于 A 组注释, 但是两组注释在变量定义、宏定义和结构体成员三类注释上的冗余度都过高. 通过工具分析, B 组注释对结构体注释时, 对每一个结构体成员都填写的是整个结构体的注释, 从而导致注释文本的平均长度较长, 但是有大量冗余注释. 规范性上, 两组注释的情况都相对较好. 表中的差异分是对注释的文本内容结构化提取以后, 注释内容结构的差异得分, 差异分越小, 规范性越好.

综合来看, 两组注释都工作量饱满, 注释也比较规范, 整体上看注释的覆盖度达到了要求, 但是变量类注释的冗余度较高, 可利用价值降低. 从制品维度看, B 组注释的注释的客观质量属性要优于 A 组注释的注释, 尤其是对于函数定义型注释, B 组注释明显投入了更多工作梳理代码的流程和调用关系, 注释内容可参考性更强.

4.2.2 客观质量属性–过程维度

过程维度的客观质量属性从注释的产生过程来反映注释的质量, 因为源代码分析注释的编辑信息是可搜集的, 因此过程维度的客观质量属性是源代码分析注释区别于传统源码注释的一个新的质量评价视角.

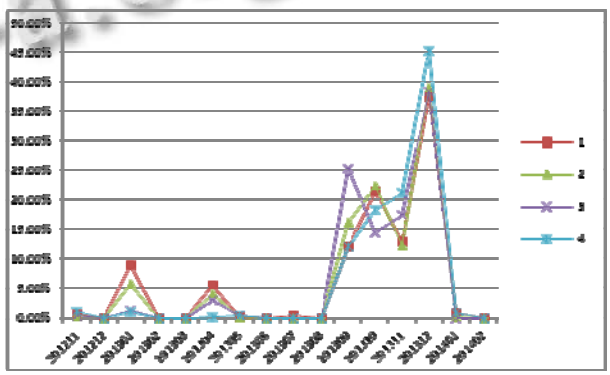

图 2　A 组注释注释编辑活跃度

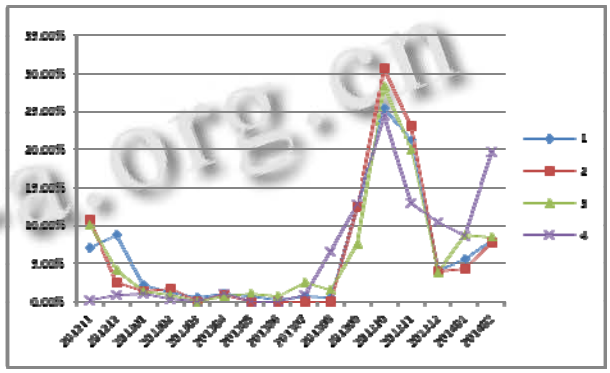

图 3　B 组注释注释编辑活跃度

图 2 和图 3 是两组注释的活跃度对比, 编号 1,2,3,4 分别代表函数定义、变量定义、宏注释和结构体成员注释, 横轴为时间, 纵轴为工作量占比. 可以看出, 两组注释添加注释最集中的时间段均为 2014 年 9 月至 2014 年 11 月. A 组注释的注释活跃度呈现出几个明显的波峰, 这些波峰的时间点基本都与项目考核的时间点吻合, 所以可以判断 A 组注释的注释一般是在考核前集中完成的.

图 4 是两组注释贡献度的对比, 左侧是 A 组注释的情况, 右侧是 B 组注释的情况. A 组注释一共有 20 名分析人员参与注释工作, B 组注释一共有 14 名分析人员参与注释工作. 图中的数字表示该分析人员贡献的注释占注释总量的百分比. 可以看出, A 组注释的注释工作量分派更加平均, 同时也有更多的分析人员参与到源代码分析的工作中, 而 B 组注释则呈现明显的差异, 最主要的 5 名分析人员承担了超过 80%的源代码分析注释工作量.

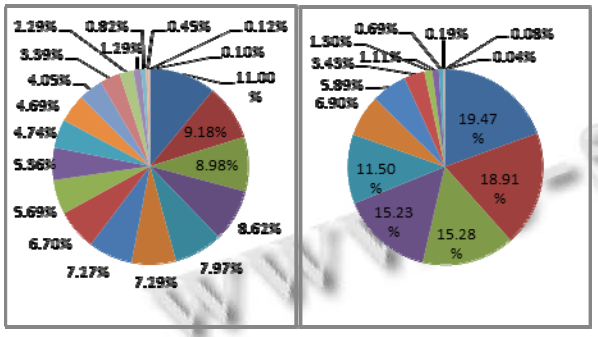

图 4 贡献度对比

表 3 协作度和审校度对比

注释类型		协作度		审校度	
		协作比例(%)	均值(%)	审校比例(%)	均值(%)
A	①	1.80	1.29	15.68	16.37
	②	1.17		10.04	
	③	1.13		20.11	
	④	1.07		19.66	
B	①	1.51	0.60	12.75	8.02
	②	0.41		8.38	
	③	0.25		8.50	
	④	0.24		2.44	

①= 函数定义; ②= 变量定义; ③= 宏定义; ④= 结构体成员

表 3 是两组注释的协作度和审校度的对比. 协作度关注参与一条注释编辑的分析人员数量, 审校度关注一条注释被编辑修改的次数, 不论是相同的分析人员编辑还是不同的分析人员编辑. 可以看出, 协作度上 A 组注释比 B 组注释略高, 但是两组注释的协作度都非常低, 绝大多数的注释都是由一位分析人员完成的. A 组注释不同类型的注释之间协作度无明显差异, B 组注释中函数定义型注释的协作度略高于其他三种注释. 审校度上, A 组注释的审校比例明显比 B 组注释的审校比例高, 也即是说, A 组注释的分析人员对已完成的注释有更多的修改和完善操作. 对比审校度和协作度很容易看出, 两组注释的审校度都明显高于协作度, 表明这两组注释的分析人员都更倾向于修改自己已经添加过的注释, 而不是去完善其他分析人员的注释.

从过程维度来看, 两组注释的客观质量属性评价结果都不是很理想, 两组注释都一定程度上存在为了应对考核而短时间内补充大量注释的情况. 两组注释的注释协作度和审校度都比较低, 大部分的注释都是添加以后就没有再完善和修改, 不同分析人员之间的协作也不充分. 结合贡献度数据也可以看出, 往往是少数分析人员承担了大量的源代码分析工作, 在面对庞大的 Linux 内核分析任务时, 分析人员的数量显得有些不足. 在个人代码分析工作量超出一定范围以后, 可能会影响最终产出的源代码分析注释的质量.

4.2.3 主观质量属性

相比于客观质量属性, 主观质量属性的判断不容易直接评价, 因为在不同的项目背景下, 评价者对于有效性、信息量、完备性等主观质量属性的判断会有所差异.

在度量主观质量属性时, 可以先从待评价的注释总体中抽取一个样本子集, 结合实际项目背景和要求, 根据专家经验对所抽取的样本的各个主观质量属性进行打分. 各主观质量属性的指标可以设计为类别型, 如有效/无效、信息量充分/信息量不足, 也可以设计为分值型, 如信息量 1 到 5 分. 在完成样本打分以后, 利用机器学习的方法或者启发式方法从各项分析结果中学习得到主观质量属性的度量模型, 达到满意效果后, 再用来评价全部注释数据.

在本案例研究中, 各主观质量属性被设计为类别型, 通过从专家打分的样本集中学习出来的规则为总体注释的评价结果如表 4 所示, 因为 90%以上的注释均为前四类注释, 故主观质量属性的评价只基于前四类注释进行.

表 4 主观质量属性度量结果

注释类型		有效性	信息量	完整性
		合格占比(%)	合格占比(%)	合格占比(%)
A	①	86.00	87.27	62.58
	②	54.09	54.78	11.64
	③	51.73	57.00	3.54
	④	82.30	82.47	0.53
B	①	91.06	92.77	47.37
	②	59.13	59.71	5.23
	③	53.05	57.10	4.13
	④	45.43	67.56	33.39

①= 函数定义; ②= 变量定义; ③= 宏定义; ④= 结构体成员

表 4 只度量了有效性、信息量和完备性三个主观质量属性, 主要是因为可读性、客观性和查证性这几个质量属性适用于比较长的注释, 而且我们实际收集的案例数据中, 对于函数注释和变量注释在依据专家经验打分时也很难对可读性进行评价, 客观性和查证性也不太适合于评价函数注释和变量注释.

从表 4 看出有效性、信息量和完备性上, 两组注释的函数定义注释的质量都比下面三类变量注释要好. 在撰写注释前, 缺乏统一的注释规范和模板, 使得注释的完备性普遍不高, 尤其是对于三类变量注释, 完备性已经非常低.

针对两组注释有效性不高的情况, 主要是因为在评价主观质量属性的时候, 对冗余注释采取了过滤处理, 冗余的注释均被判别为无效注释. 在分析注释数据的过程中, 我们也发现有的注释文本中直接包含“为了避免注释检查程序误报, 特添加这句话.”这样的内容. 因为在评审时, 针对源代码分析注释往往采用注释文本长度作为最直观的质量检测指标, 对注释内容, 只是采用抽检的方式, 所以导致有的注释出现上述没有实际价值的注释文本. 可以看出, 对注释撰写者而言, 硬性规定注释文本的长度并不一定合理. 本文开发的注释分析工具中集成的冗余检测程序可以有效的识别出这类文本, 并予以标记. 从而可以更好的评价注释的质量.

5 结语

本文结合代码注释的质量评价方法以及信息质量领域的相关研究, 提出了一种综合考虑客观质量属性、主观质量属性的质量评价方法, 并结合实际项目中的源代码分析注释数据进行了分析. 案例表明, 本文的方法可以更好的发现注释中的质量问题, 可以结合注释的内容、注释的编辑过程进行分析, 做到更精细、更全面的度量注释质量.

受本文所搜集案例数据的项目背景的限制, 本文所做的分析还是主要基于函数注释和变量注释进行, 因而对主观质量属性中的可读性、客观性和查证性这些更适合于评价长注释文本的质量属性尚没有进行深入分析, 在将来的研究中, 我们将进一步深入研究. 但是函数注释和变量注释作为注释数据中数量最庞大的注释类型, 能够对其质量进行更全面的分析仍具有重要意义.

参考文献

1 Rubey RJ, Hartwick RD. Quantitative measurement of program quality. Proc. of the 1968 23rd ACM National Conference. ACM. 1968. 671–677.

2 Boehm BW, Brown JR, Lipow M. Quantitative evaluation of software quality. Proc. of the 2nd International Conference on Software Engineering. IEEE Computer Society Press. 1976. 592–605.

3 Fluri B, Wursch M, Gall HC. Do code and comments co–evolve? On the relation between source code and comment changes. 14th Working Conference on Reverse Engineering, WCRE 2007. IEEE. 2007. 70–79.

4 Tan SH, Marinov D, Tan L, et al. @ tcomment: Testing javadoc comments to detect comment–code inconsistencies. 2012 IEEE 5th International Conference on Software Testing, Verification and Validation (ICST). IEEE. 2012. 260–269.

5 Tan L, Yuan D, Krishna G, et al. iComment: Bugs or bad comments? ACM SIGOPS Operating Systems Review. ACM, 2007, 41(6): 145–158.

6 Tan L, Zhou Y, Padioleau Y. aComment: Mining annotations from comments and code to detect interrupt related concurrency bugs. Proc. of the 33rd International Conference on Software Engineering. ACM. 2011. 11–20.

7 Khamis N, Witte R, Rilling J. Automatic quality assessment of source code comments: the JavadocMiner. Natural Language Processing and Information Systems. Springer Berlin Heidelberg, 2010: 68–79.

8 Deissenboeck F, Wagner S, Pizka M, et al. An activity–based quality model for maintainability. IEEE International Conference on Software Maintenance, 2007(ICSM 2007). IEEE. 2007. 184–193.

9 Steidl D, Hummel B, Juergens E. Quality analysis of source code comments. 2013 IEEE 21st International Conference on Program Comprehension (ICPC). IEEE. 2013. 83–92.

10 Marschak J. Economics of information systems. Journal of the American Statistical Association, 1971, 66(333): 192–219.

11 Wang RY, Strong DM. Beyond accuracy: What data quality means to data consumers. Journal of Management Information Systems, 1996: 5–33.

12 Naumann F, Rolker C. Assessment Methods for Information Quality Criteria. 2000.

13 丁敬达.维基百科词条信息质量启发式评价框架研究.图书情报知识,2014,(2):11–17.

14 Wilkinson DM, Huberman BA. Cooperation and quality in wikipedia. Proc. of the 2007 International Symposium on Wikis. ACM. 2007. 157–164.

15 Arafat O, Riehle D. The commenting practice of open source. Proc. of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications. ACM. 2009. 857–864