当代代码审查的过程方面和社会动态：

来自开源开发和微软工业实践的见解

摘要：许多开源和商业开发人员实践当代代码审查，这是一个轻量级的、非正式的、基于工具的代码审查过程。为了更好地理解这个过程及其好处，我们通过对开源软件开发人员和微软开发人员的调查收集了关于代码审查实践的信息。我们的分析结果表明，开发人员大约花费10-15%的时间在代码审查上，随着经验的增加，工作量也会增加。开发人员认为代码审查很重要，指出除了发现缺陷，代码审查还提供其他好处，包括知识共享、社区建设和维护代码质量。提交评审的代码质量有助于评审者形成对其队友的印象，这可能会影响未来的合作。我们发现微软和开放源码软件受访者之间有大量的相似之处。一个有趣的区别是，虽然开放源码软件的受访者认为代码审查是印象形成的重要方法，但微软的受访者认为知识传播更重要。最后，我们发现分布式和同地办公的微软团队之间没有什么区别。根据我们的发现，值得重点研究的三个关键领域是： 1）探索代码评审的非技术益处，2）帮助开发人员表达评审意见，3）在代码评审期间帮助评审人员理解程序。

**1引言**

近年来，许多开源软件（OSS）和商业项目都采用了同行代码审查[2]，这是一种开发人员将其代码置于同行审查之下的做法。虽然当代代码评审[45]的基本概念与传统的Fagan检查[22]相似，但也存在显著差异。Fagan检查是一个重量级的过程，要求参与者在多个阶段举行同步会议。相反，当代代码评审被定义为轻量级的、更非正式的、异步的，并由专用工具支持[2]。尽管研究表明Fagan检查可以提高软件质量[21]，但其典型的高成本和形式阻碍了广泛采用[29]，[54]。相反，当代代码审查解决了Fagan检查的许多缺点，并显示在行业和OSS环境中越来越多地采用[3]、[38]、[45]。

由于许多OSS项目，例如Apache[47]、Chromium1、Mozilla2、Qt3和Android4，现在需要在将新代码合并到主项目代码库之前进行同行评审，因此有大量开发人员定期参与代码评审。从行业角度来看，Google[56]和Facebook[33]已经采用了强制性代码审查，大约50，000名微软开发人员积极实践代码审查[14]。我们最近对OSS开发人员的调查发现，他们每周大约花6个小时在代码审查上[11]。考虑到有大量的开发人员进行代码审查，用于代码审查的总时间是相当可观的。因此，提高当代代码审查的有效性可以大大提高软件开发的生产力。

为了改进流程或实践，实证研究人员采用三步法：（1）了解当前流程以确定改进机会；（2）评估当前流程和新想法；（3）通过采纳建议[58]来改进这一进程。这项研究的高层次目标是更好地理解当代代码审查过程及其益处，这是该实证框架的第一步。具体来说，该研究的目标是通过收集有关1）代码审查过程，2）开发人员对代码审查的期望，以及3）代码审查如何影响开发人员对其同行的印象的信息来提供理解。为了收集这些信息，我们严格设计并验证了一个调查工具。我们向来自36个流行的OSS项目的代码审查参与者发送了调查问卷，并收到了287份回复。我们已经公布了调查工具的验证以及开放源码软件调查的部分结果[11]。

先前的研究已经确定了OSS和商业环境中软件工程实践之间的差异[34]，[42]。由于之前的工作没有专门针对代码审查，我们在商业环境（即微软）中重复了调查，以分析OSS和商业组织之间是否存在与我们的研究目标相关的任何差异。为了更深入地了解OSS和微软开发人员之间的相似或不同之处，我们特别从微软招募了两种类型的调查参与者：从事并置项目的参与者和从事分布式项目的参与者。因为代码审查是一个互动的过程，我们假设从事分布式项目的微软参与者对代码的看法与OSS参与者（其项目也是分布式的）相似。我们的微软调查收到了416份回复。

本研究的主要贡献是：

•更好地理解开发人员对当代代码审查的看法；

•更好地理解开发人员在代码审查期间协作的原因和方式；

•关于代码审查的非技术益处的经验证据；

•开放源码软件和微软项目之间的代码审查实践比较；

•系统设计和分析软件工程（SE）调查的过程说明。

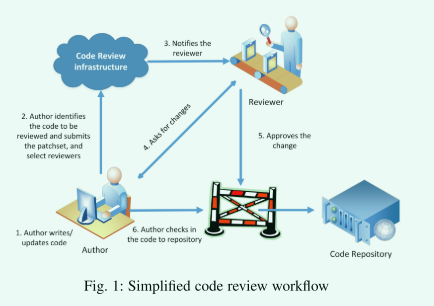
本文的其余部分组织如下。第2节简要介绍了当前的代码审查流程和以前的代码审查文献。第3节定义了研究问题。第4节介绍了研究方法。第5节描述了研究参与者的特征。第6节提供了结果。第7节讨论了结果的含义。第8节描述了对有效性的威胁。最后，第9节提供了未来工作的方向，并对本文进行了总结。

**2背景**

本节简要介绍当代代码评审的背景和先前研究。

2.1当代代码审查工作流程

当代代码评审的一个关键方面是它是基于工具的。一些流行的代码审查工具包括：Gerrit5、Phabricator6和ReviewBoard7。图1提供了当代代码评审工作流程的简化概述。首先，作者创建一个补丁集（即在一个版本中添加或修改的所有文件），以及更改的描述，并将该信息提交给代码审查工具。然后作者（或其他人）为补丁集选择审阅者。然后，代码审查工具向审查者通报即将到来的审查。在查看过程中，这些工具以并排显示的方式突出显示修订之间的更改。审阅者和作者可以在代码中插入注释。审查后，作者可以处理评论并上传新的补丁集，以启动新的审查迭代。此审查周期重复进行，直到审查者批准更改或作者放弃更改。如果审阅者批准更改，则作者提交补丁集或要求项目提交者将补丁集提交到项目存储库。



2.2当代代码评审研究概述

近年来，关于理解当代代码审查实践的研究已经有了几项。Rigby发表了一系列研究，研究了OSS项目中的非正式同行代码审查实践[46]、[47]、[48]，并比较了商业和开源项目之间的审查实践[45]。为了描述代码审查实践，Rigby和German提出了一套代码审查指标（即接受率、审查者特征、顶级审查者与顶级提交者、审查频率、每个补丁的审查者数量和补丁大小）[46]。其他研究人员计算了五个OSS项目的类似指标，并得出结论，基于项目的年龄和文化，代码审查实践在OSS项目中有所不同[1]。然而，这些发现在后来的一项研究中是矛盾的，该研究发现，尽管五个OSS项目和几个商业项目之间存在很大差异，但它们的代码审查指标在很大程度上是相似的[45]。

在看到OSS项目成功采用代码审查实践后，许多商业组织最近采用了同行代码审查实践[2]，[3]，[44]，[53]。与OSS项目相反，微软的代码审查参与者使用同步和异步通信媒介。他们还认为代码审查期间的沟通对于理解代码更改和设计原理至关重要。微软开发人员表示需要保留代码审查通信，以满足以后的信息需求[53]。微软的另一项研究发现，尽管发现缺陷是代码审查的主要动机，但其他好处（例如知识传播、团队意识和确定更好的解决方案）可能更为重要。主要的挑战是理解代码更改[2]。

虽然这些研究描述了商业项目中代码审查的特点，但只有一项侧重于代码审查的定量方面的研究[45]对OSS和商业项目的代码审查实践进行了比较和对比。由于开发人员的动机和项目治理在OSS和商业组织之间不同[32]，[42]，代码审查协作在OSS和商业项目之间也可能不同。缺乏研究是从OSS和商业开发人员那里收集关于当代代码审查的信息的动机之一。

虽然大多数早期的探索性研究侧重于理解代码审查实践，但最近的一些研究侧重于理解不同因素对代码审查的影响。代码审查特征，如审查规模、组件、优先级、组织、审查者特征和作者经验，对审查完成时间和结果都有显著影响[5]。此外，审阅者先前在更改或审查工件方面的经验以及审阅者的项目经验增加了他/她提供有用反馈的可能性[14]。虽然这些研究侧重于技术人为因素和代码更改的特征，但没有研究侧重于非技术人为因素（即作者的声誉以及作者和审稿人之间的关系）。因为代码审查促进了人与人之间的直接协作，所以更好地理解各种人为因素的影响对于改进代码审查实践至关重要。

最近的一些研究调查了代码审查的各种技术好处。尽管代码评审的主要目标是缺陷检测，因为四分之三的评审意见与可维护性问题有关，但对于需要高可维护性代码的项目，代码评审可能更为有利[6]。代码评审对软件质量有重大影响。最近的一项研究发现，低代码评审覆盖率（即已评审的变更比例）和低评审参与率（即评审员人数）通常都会增加发布后缺陷的可能性[38]。虽然这些研究侧重于代码评审的技术效益，但只有一项研究[2]探讨了代码评审的非技术效益。关于非技术利益（即印象形成、知识共享和指导）的证据大多是轶事。关于代码评审的各种好处的经验证据可以鼓励项目经理对其项目采用代码评审。

虽然这项先前的工作为当代代码审查提供了一些重要的见解，但仍有许多关键方面尚未探索。首先，经常使用代码评审的开发人员应该能够描述代码评审有用或无用的场景。其次，经验丰富的评审员还应该能够描述代码评审的最佳策略，并帮助其他开发人员编写可接受的代码。最后，由于代码评审涉及参与者之间的直接协作，各种类型的社会互动对于成功的代码评审至关重要。然而，代码评审的这三个方面还没有得到研究者足够的重视。我们工作的目标之一是更好地理解这些方面，以指导项目经理关于代码评审有用性的决策，并帮助提高代码评审的有效性。

2.3我们之前的调查

本文中的工作建立在我们之前对OSS项目中的当代代码审查实践的调查结果之上[11]。在该论文中，我们开发并验证了第4节中所述的调查工具。我们利用该调查收集了287名OSS开发人员的数据，这些开发人员一直活跃在当代代码审查领域。该论文报告了调查的主要定量结果之一，特别是参与当代代码审查一段时间的OSS同行之间存在高度信任、可靠性、专业知识感知和友谊。在本文中，我们扩展了这些结果，以包括第一次调查的定性数据，并将这些结果与商业开发商进行的第二次调查的结果进行比较。

**3 研究问题**

为了实现研究目标，我们探索了八个研究问题。本节的其余部分定义了每个问题。

3.1规范审查的重要性

代码评审需要大量的努力。他们将代码合并到主分支延迟1-2天[45]。然而，最近的研究表明，只有四分之一的代码评审注释与功能缺陷有关[6]，[19]，这就提出了开发人员是否认为代码评审所花费的努力是有益的问题。为了更好地理解开发人员如何看待代码评审的重要性，第一个研究问题是：

RQ1：为什么开发人员认为代码审查对他们的项目很重要（或不重要）？

3.2代码审查流程

因为项目经常要求使用代码评审，所以开发人员花费大量时间执行代码评审。为了量化这一努力，第二个研究问题是：

RQ2：开发人员平均花多少小时进行代码审查？

在我们之前对OSS项目中基于代码审查的社交网络的研究中，我们观察到子社区的存在以及一些开发人员对之间的大量互动[12]。随后，我们发现，在OSS项目中，经验丰富的开发人员比新手收到更及时的审查请求反馈[13]。这些结果表明，互动的历史可能会影响评论者对特定评论的接受程度和优先顺序。下一个研究问题调查这一现象。

RQ3：开发人员如何决定是否接受传入的代码审查请求？

评审者可以使用不同的标准来确定代码更改是否具有高质量。例如，评论者可能对编码风格对质量的影响有不同的意见[14]。下一个研究问题旨在更好地理解这些因素：

RQ4：哪些代码特征表示低质量代码？

代码审查的目标不仅是识别代码更改中的问题，而且是帮助作者解决这些问题。经验丰富的评审人员可以指导代码作者了解编码技术、项目设计或API使用。下一个问题旨在更好地理解这个指导过程：

RQ5：审阅者如何帮助低质量代码的作者将其改进到包含在项目中所需的水平？

3.3代码评审对同行印象的影响

在代码审查过程中，激烈的互动让参与者有机会获得对同行能力的独特见解。例如，如果审阅者反复发现特定代码作者的贡献是高质量的，则审阅者可能认为该作者是高度胜任或聪明的。因此，代码审查协作可以帮助参与者形成对彼此的准确认知。此外，审稿人可能更信任由已知有能力的作者做出的与项目相关的决定。因为当代代码审查的一些主要好处是非技术性的，即超出缺陷检测[2]，所以了解这些非技术性的好处是很重要的。为了帮助确定这些益处，我们提出了三个研究问题。第一个问题，从积极的方面来说：

RQ6：使用高质量或杰出的解决问题的方法如何影响审阅者对代码作者的看法？

相反，低质量的代码可能会导致负面印象。因此，下一个问题是：

RQ7：写得不好的代码如何影响审阅者对代码作者的看法？

正如前面的研究问题中所提到的，代码审查可能会对团队成员对彼此的印象产生积极或消极的影响。为了帮助判断代码审查的整体价值，我们问：

RQ8：代码评审对同行印象的影响是什么？

**4研究方法**

我们进行了两次调查来回答研究问题，并将OSS项目与商业项目进行比较。第一次调查针对的是OSS开发人员。第二项调查针对的是微软开发人员。我们已经公布了第一次调查的部分结果[11]。本节的其余部分介绍了调查设计流程、参与者选择标准、试点测试、数据收集和数据分析方法。

4.1勘测设计（Survey Design）

因为我们的目标是测量同龄人的印象结构，所以我们遵循公认的社会和行为研究方法来构建量表[20]，[23]。在这种方法中，研究人员没有直接向参与者询问每个感兴趣的结构，而是定义了一些关注同一基本结构不同方面的量表项目。然后，在分析过程中，研究人员能够基于不同的量表项目集对该结构有更全面的理解。

为了了解同伴印象，我们确定了四个关键结构。对于每个结构，我们定义了一组报表（比例项目）。我们从心理学、信息科学或组织行为学的成熟量表中得出这些结论。为了确保它们对于软件工程来说是完整的，我们添加了一些额外的语句。四种结构以及语句的源是：

1）可信赖性[30]，[37]，[43]，[51]；

2）可靠性[30]、[43]、[49]；

3）专业知识感知[49]；

4）友谊[16]、[49]。

表1列出了每个构造的语句。对于每一项陈述，回答者使用7分制来表明它是否更好地描述了代码审查合作伙伴或非代码审查合作伙伴。我们将等级定义如下：1=描述CodeReview合作伙伴而不是非代码审查合作伙伴，4=同等描述两者，7=描述非代码审查合作伙伴而不是代码审查伙伴。为了避免任何偏见，调查工具以随机顺序呈现陈述，而没有相应结构的名称。

该调查还包括四个多项选择题、十四个开放式问题和一个评级量表问题，以解决研究问题和收集人口统计数据。表2列出了这些附加问题（为简单起见，重新编号）。在本文的其余部分，我们将通过这些数字来引用这些问题。

请注意，对于这两组问题，开放源码软件调查和微软调查之间存在一些微小的差异。第4.4节解释了这些差异。

4.2参与者选择

开发人员必须参与足够数量的当代代码审查（作为作者或审查者），才能准确理解代码审查过程、代码审查的非技术益处以及对同行印象形成的影响。为了确保有效的结果，我们只调查了具有丰富经验的开发人员。对于调查1，我们挖掘了34个OSS项目的代码审查库，这些项目使用Gerrit、ReviewBoard或Rietveld，以确定参与了至少30个代码审查请求的开发人员（作为作者或审查者），并确定了2，207名开发人员。类似地，对于调查2，我们查询了微软的CodeFlow分析平台[8]，以选择2000名参与了至少30次当代代码审查的开发人员。

研究目标之一是分析来自商业组织的开发人员是否根据其项目是并置还是分布式而表现出不同的行为。我们假设从事分布式项目的商业开发人员更有可能表现得像OSS开发人员（其项目也是分布式的）。为了在所有研究问题中检验这一假设，我们特别从两种类型的项目中招募了微软开发人员：1）大多数开发人员集中的项目，2）大多数开发人员分散的项目。

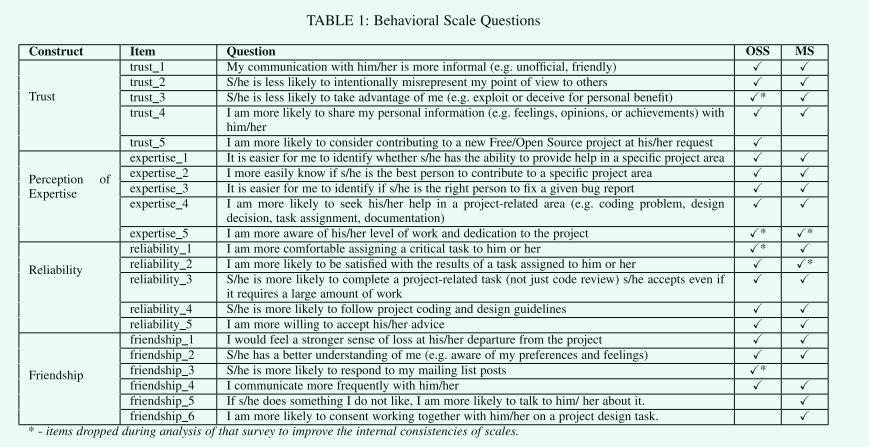
4.3试点测试

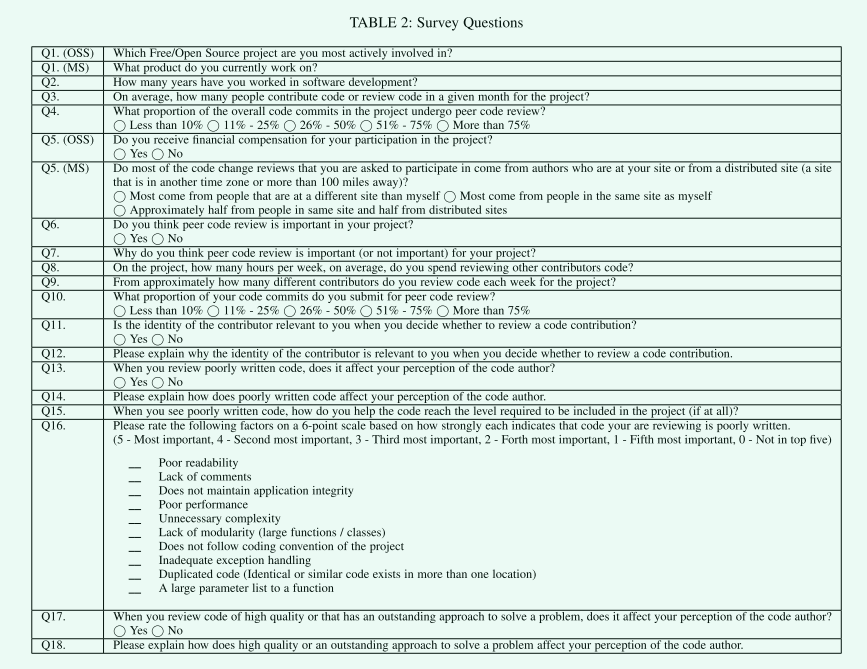
为了确保尺度项（语句）相对于构造的可理解性和有效性，我们进行了五次试点测试。首先，心理学、计算机科学和管理信息系统的研究人员回顾了这些问题。这一审查导致了几个变化，包括：（1）将评分标准从5分增加到7分，（2）重新措辞一些问题以消除偏见，以及（3）增加问题以更广泛地看待代码审查过程。

其次，软件工程研究生进行了试点调查，以确定在理解问题方面的任何困难，并估计完成调查所需的时间。

第三，我们向数据库中两个项目的20名OSS代码审查参与者发送了调查问卷。该版本调查的完成率较低。为了解决这个问题，我们重新表述了一些问题，重新编排了行为量表问题（表1），使它们看起来不那么令人生畏，并重新排序了一些问题。

第四，我们向数据库中另一个项目的24名OSS代码审查参与者发送了改进后的调查。我们收到了足够多的回复来分析这四个同伴印象结构的内部一致性。这一分析表明，可靠性量表具有可疑的内部一致性。因此，我们在这个结构中增加了两个问题。





最后，我们向来自其他11个项目的117名OSS代码审查参与者发送了调查问卷。完成率接近20%，每个结构的内部一致性充分（Cronbach sα>0.710）。因此，我们认为该调查已准备好进行广泛分发。我们将飞行员的反馈纳入了最终调查。

4.4数据收集

两次调查的数据采集过程有一些不同。

4.4.1 OSS调查

2013年2月，我们向OSS数据库中的2，046名活跃的当代代码审查参与者发出了调查邀请。其中，231封电子邮件无法送达，剩下1815份有效邀请。两周后，我们发送了一封提醒邮件。大约两周后，在每日回复率降至几乎为零后，我们结束了调查。我们收到了287份回复（回复率约为16%）。

4.4.2微软调查

关于开放源码软件调查的问题Q1和Q5是专门针对开放源码软件开发的。在微软调查中，我们修改了这些问题，以适应商业开发环境。两个量表项目（信任5和友谊3）是针对开放源码软件开发环境的，因此被排除在微软调查之外。由于这些项目被排除在外，友谊量表只剩下三个项目。为了确保所有量表都有足够数量的项目，我们在友谊量表中增加了两个项目。表1的最后两栏准确显示了每项调查中包括的项目。2013年9月，我们向2，000名曾参与当代代码审查的微软开发人员发出了修订调查的邀请。一周后，我们结束了调查。我们收到了416份回复（回复率约为21%）。

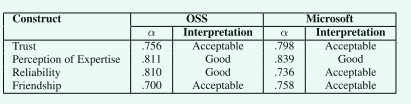
4.5数据处理与分析

以下小节描述了行为量表问题和五个开放式问题的数据处理和分析步骤。

4.5.1行为量表问题

我们分析了调查的三种有效性形式。首先，我们使用来自心理学、管理学和计算机科学的专家研究人员来审查调查问题的面部效度[36]。其次，我们从先前验证的量表中仔细选择合适的项目，以确保内容的有效性[36]。最后，我们使用VARIMAX旋转进行主成分分析，以测量量表项目的结构效度[36]。表3报告了基于两次调查的量表α系数测量及其解释。

这种方法保证了本研究中使用的行为量表的高信度和效度。在前一篇文章中，我们已经详细描述了调查工具的信度和效度度量[11]。我们还使用类似的过程验证了Microsoft调查中使用的四个修改量表的可靠性和有效性。



4.5.2开放式问题

对于开放式问题，我们遵循了系统的定性数据分析过程。首先，两名分析师（本科生研究经验（REU）学生）从对开放源码软件调查的每一份答复中提取了一般主题。接下来，前两位作者与这些分析师合作，为每个问题开发一个商定的编码方案。使用这些编码方案，两位分析师独立地编码了反应。编码后，他们检查结果以识别任何差异。他们讨论并解决了这些差异。对于微软的调查，最后两位作者使用了相同的过程来分析定性数据。对于两次调查中的五个开放式问题，我们总共编码了2626个回答。

**5 人数统计**

为了给结果提供适当的背景，本节描述了受访者所代表的项目的人口统计数据以及受访者本身的人口统计数据。

5.1代表的项目

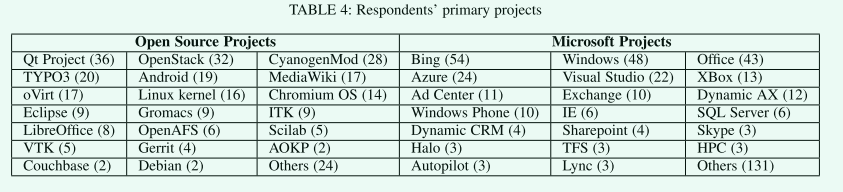
表4提供了问题Q1（表2）关于受访者主要项目的结果。括号中的数字代表列出该项目的答复者人数。作为这些项目中代码审查频率的一个指标，大约83%的开放源码软件受访者和90%的微软受访者表示，他们项目中75%以上的代码更改都经过代码审查。此外，在两次调查中，几乎三分之二的受访者表示，他们将每一项代码变更提交给代码审查。因此，调查受访者正在积极使用当代代码审查。

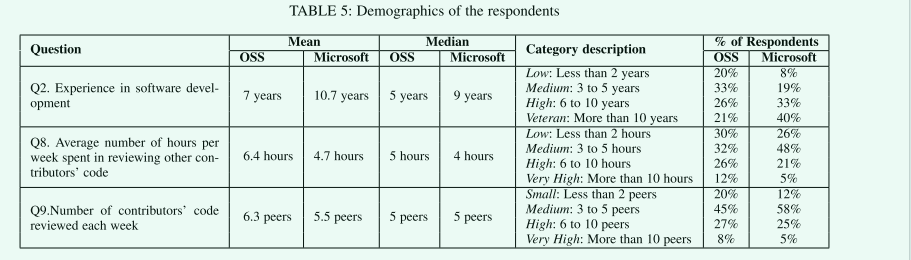
5.2受访者的人口统计数据

表5显示了调查问题2、8和9的结果。在每种情况下，我们通过分析响应的频率分布将响应分为四类。然后，我们检查了这些类别，以确保它们也具有逻辑意义。

在OSS调查中，60%的受访者获得了项目工作报酬，40%是志愿者（Q5）。付费参与者的百分比并不令人惊讶，因为我们从中抽取的项目列表包括一些大型赞助项目（例如Android、Chromium OS、QT Project和OpenStack），其中大多数参与者都是赞助公司的员工。这一分布略高，但与以前的开放源码软件调查相同，以前的开放源码软件调查有40%-50%的付费参与者[10]，[31]。

在微软的调查中，大约69%的受访者表示，大多数代码审查请求来自同一站点的开发人员，13%来自不同的站点，其余的则各占一半。





**6 结果**

以下小节描述了两次调查的结果。对于第3节中介绍的八个研究问题中的每一个，我们将OSS调查的结果与Microsoft调查的结果进行比较。为了帮助澄清结果，我们还摘录了对开放式问题的定性回答。在本节中，我们使用唯一的标识符来标识每个受访者，OSS-XXX和MS-XXX分别表示来自OSS调查和微软调查的受访者。除非明确说明，开放源码软件和微软答复者的意见是相似的。因此，所选择的引文最好地代表了两个样本（OSS和Microsoft）的一组答复。

作为编码过程（第4.5.2节）的结果，每个开放式问题都有大量的详细类别。为了演示结果，我们将详细类别抽象为数量较少的高级类别。使用更详细的类别对数据的进一步分析可以在补充网站12上找到。在定性分析中，每个开放式响应可以匹配多个代码。因此，百分比的总和可以大于100%。

对于每个问题，我们使用Shapiro-Wilk检验[52]来检验答案分布的正态性。在非正态分布的情况下，我们使用非参数统计。

6.1 RQ1：为什么代码评审很重要？

在回答问题6时，98.6%的OSS受访者和100%的Microsoft受访者认为代码审查对他们的项目很重要。图2显示了受访者认为代码评审对代码质量很重要的原因（Q7）。尽管两项调查中的回答相对顺序相同，但OSS受访者和微软受访者之间的回答分布存在显著差异（χ2=21.38，df=5，p<0.001）。

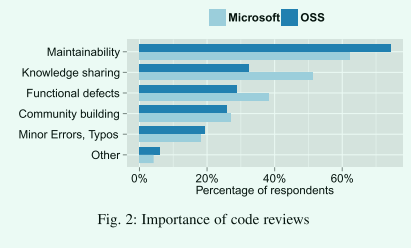
开放源码软件的受访者比微软的受访者更强调可维护性。由于开放源码软件的参与者来自不同的地点、背景和专业知识水平，提交的代码质量可能差别很大。因此，OSS评审员必须更加关注保持一致的代码质量。相反，微软开发人员的代码质量差异较小。因此，微软受访者能够在代码审查期间更专注于发现缺陷和提高项目意识。

微软开发人员告诉我们，知识共享是代码审查的主要目的之一。团队中的新人通常会参与评审，这样他们就可以更多地了解代码库以及如何进行代码评审。在某些情况下，专家和经验较少的开发人员之间存在明确的导师-学员关系，这在代码审查中得到了体现。我们不知道在OSS项目中代码审查有类似的用途。

有趣的是，在两次调查中，消除功能缺陷只是代码审查的第三个最重要的原因。这一结果与早期的结论一致，即当代代码审查的其他好处，即知识转移和确定更好的解决方案，可能比缺陷检测更重要[2]。先前关于软件检查的研究还报告了软件检查带来的以下好处：缺陷识别[27]，[50]，知识共享[15]，[50]，[57]，提高项目意识[15]，【50】，以及降低开发成本[26]，【40】。此外，我们之前的工作发现，大约一半的代码审查意见与可维护性问题有关，只有不到四分之一与功能缺陷有关[14]。以下小节详细介绍了开发人员认为代码审查对其项目非常重要的原因。

6.1.1提高可维修性

每个调查中的大多数受访者（OSS：71%，Microsoft：61%）表示，代码审查在效率以及其他可维护性属性方面提高了项目的可维护性，即易读性、可测试性、遵守风格指南、遵守应用程序完整性以及符合项目要求。一般来说，当开发人员知道更改需要经过同行评审时，他们会更加谨慎。



如果你知道有人会看你，你就穿得更好。当你知道有人会质疑你的某些决定时，要么你不做决定，要么你准备为它辩护。因此，一般来说，它通过迫使您以其他人的方式看待您的代码来提高质量并使您成为更好的开发人员。[MS-50]

除了代码质量之外，审阅者还评估代码是否符合项目要求。这种一致性在OSS项目中尤其重要，因为贡献者可能有不同的个人目标。

通过要求核心维护者的批准，它还有助于将不需要的代码排除在外。[OSS-48]

代码审查和随后的讨论也有助于维护项目设计约束，并产生更好的设计

它允许（大量）代码库特定领域的专家及早发现变更问题，帮助产生改进的设计思想。[开放源码软件-118]

同样的

.重新考虑了关键的设计决策，最后想到“哇，我做这件事的方式很愚蠢，原因很多。”[MS-80]

代码审查的另一个好处是生成可读性更强的代码。为了帮助审阅者更容易地理解代码，开发人员使用文档、注释和适当的缩进来使代码更具可读性。

代码审查有助于查看“我的代码”是否以应有的方式被阅读和解释。[MS-25]

可读的代码还有助于长期的可维护性，特别是对于大型和长期的项目。

代码是我们最宝贵的资产，也是我们最大的负债。但是我们很少有时间对已经完成的功能进行再投资，所以无论我们检查什么，都必须具有正确的质量。您可以测试很多东西，但是您真的不能测试可运维性和代码库调试的容易程度。所以为了避免虫子农场，你真的必须回顾一下。[MS-312]

代码审查也有助于实施通用的编码风格，这是可维护代码的关键特征之一。

代码样式甚至很重要，因为代码只写了一次，但读了很多次。事实上，代码可以由其他开发人员维护，因此遵循指导原则很重要。我们发布了严格的编码指南。[MS-290]

6.1.2促进知识共享

代码审查促进了多种类型的知识共享。代码评审交互帮助作者和评审者学习如何使用新方法解决问题。审稿人通常不仅要确定问题，还要解释为什么作者的方法可能会导致潜在的问题。审查还有助于社会化项目细节，例如架构、通用API和现有的库。

...将信息传播给更多的人，这样如果有人生病、度假或离开团队，系统的所有知识就不会丢失，帮助分享有用的实用程序的知识，这样我们就不会有重复的系统做同样的事情。[MS-196]

代码审查通过确保至少有一个或两个审查者也知道代码更改来提高项目成员的项目意识。

...它有助于确保一个群体中的不止一个成员熟悉任何变化，确保所有变化都是（至少在某种程度上）理智的，并有助于培养在变化之前受到变化影响的人的反馈实际上发生了。[OSS-194]

代码审查还允许高级项目成员指导新人。

...代码审查通常是开发人员之间关于软件的知识转移和头脑风暴的主要方法之一。它们是在项目和技术上培养新开发人员的关键路径的一部分，并且它们通常是有经验的开发人员在上下文中分享交易技巧和知识的地方。[MS-190]

除了新人，更有经验的项目成员也可以通过代码审查来学习。

...允许我们利用代码库中每个人学到的经验教训，而不是每个人都会遇到。[MS-355]

6.1.3消除功能缺陷

审稿人经常会发现逻辑错误、特殊情况、安全问题或作者可能忽略的一般不兼容问题。

根据我的经验，代码审查极大地减少了bug数量。在没有审查人员提出一些改进建议或缺陷注释的情况下，很少有更改被接受。[OSS-145]

经验丰富的安全审查人员通常能够在代码审查期间识别关键的安全问题。

越多人看越多，你不能让任何人从社区中合并任何东西到你的代码中（安全风险）。[OSS-105]

最后，代码审查有助于向更广泛的受众告知商定的更改，从而有助于避免不兼容问题（即，构建中断）。

... 确保功能/缺陷可以集成到项目的其他部分，由其他开发人员完成。[MS-241]

6.1.4.鼓励社区建设/合作

通过促进开发人员和审阅者之间的直接协作，代码审阅鼓励社区建设和协作。虽然微软和OSS的受访者都提到了社区建设，但他们认为社区建设在OSS项目中非常重要。

.. 帮助开发人员感觉自己是F/OSS社区的一部分。它还提供了一个参与的框架，允许人们对提交的变更发表意见并感到参与。[OSS39]

此外，将各种利益相关者纳入审查过程促进了团队成员之间的协作。

它支持对系统实施的协作贡献。团队中的每个人对系统的每个部分都有发言权。[MS-100]

最后，代码审查还可以帮助开发人员更好地了解彼此的专业知识并建立关系。

让那些通常不在一起工作的工程师（例如竞争公司的员工）之间产生协作和同志情谊。[开放源码软件-37]

6.1.5识别小错误、打字错误

开发人员通常不会注意到自己的小错误和错别字。在没有代码审查的情况下，识别这些小问题可能非常耗时。此外，开发人员可能会忘记更新注释，这对于项目的长期可维护性至关重要。在大多数情况下，大多数小错误或拼写错误都是在代码审查期间发现的。

它有助于捕捉人为错误/拼写错误。两双眼睛总比一双好。[MS-182]

6.2 RQ2：代码评审花费了多少时间？

根据Q8，OSS开发者每周花在代码审查上的时间中位数为5小时，微软开发者为4小时。考虑到每周40个工作小时，这个结果表明开发人员将10%-15%的时间花在代码审查上。此外，OSS开发者花在代码审查上的时间（Mann-Whitney U，p=0.05）明显比微软开发者多

因为经验较少的开发人员更有可能邀请经验丰富的团队成员执行代码审查，所以我们假设经验丰富的开发人员会花更多的时间执行代码审查。图3显示了开发经验与花在代码审查上的平均小时数。由于每周检查时数的分布与正态分布显著不同，我们使用了非参数方差分析（即Kruskal Wallis H），这表明这些差异具有统计学意义（OSS:χ2=8.16，p=0.043，Microsoft:χ2=8.43，p=0.038）。

对于OSS受访者，付费贡献者花在代码审查上的时间明显多于志愿者参与者，中位数为5小时而非3小时（Mann Whitney U，p<0.001）。这一结果是有意义的，因为付费贡献者经常充当守门员，通过防止错误、不需要的或恶意代码来维护软件的完整性。作为看门人，付费参与者将因此审查来自许多不同同行的代码，并花费更多时间进行代码审查。为了支持这一观察结果，Q9的结果表明，与志愿者相比，付费贡献者每周审查来自更多同龄人的代码，中位数为5名同龄人vs.4名同龄人（Mann-Whitney U，p=.009）。

6.3 RQ3：开发者如何决定是否接受审核请求？

超过一半的开放源码软件受访者和三分之二的微软受访者表示，在接受代码审查请求时，作者的身份很重要（问题11）。图4显示了受访者认为代码作者的身份很重要的原因（Q7）。尽管确定的因素在两次调查中是共同的，但回答的分布存在显著差异（χ2=24.09，DF=4，P<.001）。例如，开放源码软件的答复者强调非技术因素（即声誉和关系），而微软公司的答复者则强调技术因素（即时间/努力和专门知识）。这一结果加强了其他研究[32]，[41]中发现的OSS开发人员对声誉和关系的重视。

相反，贡献者的专业知识领域和审查所需的时间/精力是微软调查对象的优先考虑因素。与开发人员的讨论揭示了这一结果的原因。就专业知识而言，经验丰富的Microsoft开发人员经常会收到大量的审查请求。因此，为了最大限度地减少审查时间，他/她更有可能接受他/她具有专业知识的审查。此外，由于Microsoft开发人员必须管理他们的时间竞争需求，并且产品的时间表很紧，因此开发人员必须经常根据完成任务所需的时间做出决策。因此，参与代码审查的选择很大程度上取决于所需的估计时间。以下小节提供了每个因素的详细信息。

6.3.1与作者的关系



审阅者与代码作者的关系和交互历史通常会影响是否接受审查请求的决定。与代码作者的关系非常重要，特别是对于OSS开发人员，他们在接受代码审查时考虑与代码作者关系的可能性几乎是MS开发人员的两倍。

我们彼此认识。我们了解彼此的优势和劣势，我们可以改变我们的审查方式，以满足特定开发人员的需求。这是人类自然执行的优化。是净阳性吗？我想是的。[OSS-60]

此外，为了优化审查代码所花费的时间，审查者通常会选择审查已经审查了他/她的代码的作者的代码。

这感觉像是一种“交换”——如果投稿人在过去以彻底/及时的方式审查了我的代码，我愿意报答。[OSS-20]每周40%60%的时间花在代码审查上（中位数）

其次，由于来自值得信赖的作者的代码更改更有可能需要较少的审查工作，因此审查者和作者之间的信任级别非常重要。

若您审查来自您已经非常熟悉和信任的人的代码，那个么您只能更多地关注于检测粗心的错误，而较少关注代码更改的总体设计。[OSS276]

最后，审阅者优先考虑来自队友或同事的请求。

... 我公司还有其他的程序员也在做这个项目，如果他们提交了一些对时间敏感的东西，我可能会优先考虑这些审查，以保持进展。第一次投稿的人往往有点轻重缓急。[OSS-227]

6.3.2作者声誉

根据他们的目标或项目角色，一些审稿人可能会寻找声誉良好的作者，而另一些审稿人可能会关注声誉较差的作者。为了充分利用花费在代码审查上的时间，一些审查者倾向于来自他们认为能够生成高质量代码的作者的审查请求。

通常做评论的时间是有限的。我更喜欢贡献者的更改，因为我知道他们提出了好的更改（高代码质量、好的提交消息、小范围、专注于一件事），因为我知道我可以很快完成审查。我也更喜欢贡献者的更改，他们自己通过对其他人的更改进行代码审查来提供反馈。[OSS106]

相反，一些审阅者是看门人，他们关注来自新的或麻烦的作者的代码更改。

我更倾向于审查新加入团队的开发人员所做的更改，以及对编码标准的遵从性较差或对其审查有大量重要评论的开发人员所做的更改。[MS-90]

有时，代码作者的经验也会影响接受或拒绝审查请求的决定。开发人员通常会接受有经验的贡献者的审查请求，希望了解优秀的技术或设计。相反，专家和/或代码所有者可能更倾向于审查来自经验不足的开发人员的代码，以保持高代码质量。

有些人工作出色，我想向他们学习，所以我回顾了他们的CR，看看他们做了什么，尽管我几乎从未发现问题。[MS-319]

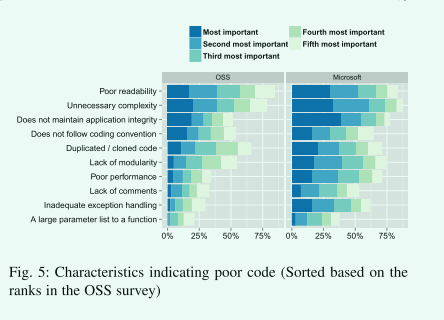
6.3.3专业领域

许多审阅者更喜欢审阅与他们的工作领域或专业知识密切相关的代码更改。代码作者的身份通常可以帮助他们确定更改是否相关。专业领域是一个非常重要的因素，特别是对于微软开发人员来说，要优先考虑即将到来的审查。

我收到了很多来自多个团队的代码审查请求。我只回顾我所在领域的事情，而《改变》的作者经常帮助我确定改变是否与我相关。[MS-325]

一些审阅者甚至拒绝与他们的专业领域无关的代码审阅。

区域所有权和专业知识很重要。如果我不熟悉某个领域，根据其复杂性，我会拒绝或部分审查它。[MS-12]



6.3.4预期审查时间/工作量

根据作者的身份，审稿人通常可以预测所需的工作量。为了最大限度地利用花费在代码审查上的时间，一些审查者将注意力集中在最需要关注的领域。

我的时间本来就有限，所以我选择优先为经验较少的开发人员进行代码审查。对于那些我知道有高质量贡献历史的人的代码，我不太可能花时间审查。[OSS-176]

相反，为了减少工作量，一些审阅者倾向于避免对已知的较差编码者进行更改。有时，您希望快速审查您非常信任的贡献者的代码。

其他时候，您可能会选择忽略来自已知贡献者的工作，这些贡献者通常会产生糟糕的工作。[OSS-114]

6.4 RQ4：哪些特征表示低质量代码？

根据我们之前的工作[10]和常见代码气味（源代码中通常对应于系统中更深层次问题的任何症状）[25]，我们确定了低质量代码的十个特征。我们要求受访者在代码评审期间根据这些特征的重要性对其进行排序。由于调查工具的局限性，在Microsoft调查中，受访者以6分制对每个特征进行评分，而不是按顺序排列（即，他们可以将多个特征评为最重要的特征，而不是仅评为一个）。因此，对于Microsoft调查，最重要的数据的总数大于100%。然而，我们认为这可能不是一个问题，因为我们只对比较两次调查的特征等级感兴趣。对于这两项调查，我们分别使用前两个评级（即最重要和第二重要）的数量计算特征的等级。图5总结了受访者如何对每个特征的相对重要性进行评级（Q17），这些特征是根据他们在OSS调查中的排名进行排序的。

事实上，不必要的复杂性和可读性差是每个调查中的前三个特征，这表明简单（而不是复杂）和可读的代码更容易审查。有趣的是，在两次调查中，缺乏评论的排名都很低（OSS：第8位，Microsoft：第9位）。这些结果的组合表明，审阅者希望代码是直接的和自我文档化的，而不是需要大量的注释来解释它。因为审阅者必须理解代码才能正确地审阅它，所以复杂的方法，即使注释得很好，也可能需要更长的时间来审阅。

在这两项调查中，不维护应用程序完整性也是排名前三的特征之一。对于长期项目，保持一致的设计非常重要。违反项目设计的功能不仅会增加未来维护的负担，还会打开错误甚至漏洞。这样的代码通常表明作者缺乏关于项目设计的知识，或者作者缺乏对项目的关心/奉献。因此，作者应该注意提交的代码更改要保持应用程序设计约束。

在两次调查中，九个特征中有八个特征的排名相似（相差不超过两个排名）。唯一的例外是特征：不遵循该项目的编码惯例（在开放源码软件中排名第四，在微软中排名第八）。这一结果有两种可能的解释。首先，虽然微软的受访者可能认为编码惯例问题很重要，但他们可能不会根据这些问题来判断代码质量，因为这些问题更容易解决。其次，由于Microsoft开发人员经常使用自动化工具来识别和解决编码约定问题，因此他们在代码审查期间可能较少关注这些问题。

6.5 RQ5：开发人员如何帮助改进低质量代码？

图6列出了被调查者使用的方法，以帮助编写较差的代码达到包含在项目中所需的质量水平（Q15）。OSS受访者和微软受访者的回答分布存在显著差异（χ2=93.29，DF=6，P<.001）。这种差异主要是由两个因素造成的。

首先，微软受访者更有可能使用其他渠道（即面对面、Skype、即时消息或电子邮件）与作者沟通。他们发现这些沟通有助于迅速解决任何误解。相反，在开放源码软件项目中，面对面的交流可能不是一种选择。有趣的是，开放源码软件开发人员可以使用一些工具（例如Skype或其他互联网语音/视频技术），但他们没有这样做。

其次，当其他方法不成功时，OSS受访者更有可能自己重写代码。由于开放源码软件的参与者可能没有义务采取后续行动，他们可能不会作出必要的修改，使代码可以接受。如果代码更改很重要，那么审阅者可能会选择只解决问题，而不是等待原作者。相反，微软的受访者很少自己重写糟糕的代码，主要原因有两个：1）微软开发人员需要跟进，2）审阅者知道他们必须指导低质量代码的作者，以帮助他们学习如何编写更好的代码。

6.5.1提供意见

每次调查中，超过80%的受访者通过代码审查工具提供评论，以帮助作者改进写得不好的代码。评审人员通常指出代码的特定缺陷，并要求作者修复这些问题。

对于有问题的补丁的特定问题，或小的编码风格/约定问题，我将通过逐点反馈和建议来回复补丁。对于主要的系统问题，例如普遍使用不正确的编码风格/约定，或基本架构问题，我将对此类问题的第一个实例进行回复，并对问题进行总结，并指出存在更多我没有引用或评论的问题。[OSS-46]

许多审阅者提供了重构代码以使其更具可读性的提示或建议。

我尝试给出语法提示或改进建议（比如重写函数以降低复杂性，指出哪里有重复代码以及如何共享代码，并建议将大型函数拆分为小型函数）。[MS-403]

一些受访者还提到了建设性批评的重要性，以避免伤害代码作者的感情。

批评代码，而不是作者。描述更好的方法，不要只是诋毁选择的方法。询问选择方法的原因，不要攻击选择。指出可能忽略的边缘情况。如果适用，请直接参考编码标准。[MS-73]

6.5.2重写/修复代码

在不要求作者对审查作出回应的项目中，即一些开放源码软件项目中，一些审查者发现只处理低质量代码比提供评论更快。

在某些情况下，我自己处理代码更改，并向提交者解释我进行后续更改的原因。[OSS-276]

审查人员承认，这种做法可能不是最好的方法。

通常情况下，这可以归结为拒绝它或通过自己编写它。我知道这不是一个好的做法，但我们都是志愿者。[OSS-154]

6.5.3提供指导

在低质量代码的作者缺乏项目或编程知识的情况下，指导可能是提高代码质量的最佳方法。

当我认为对项目有益时，我会尝试为贡献者提供设计指导。我可能还会提供一些特定语言的指导，或者至少让投稿人参考我认为有帮助的相关文档。[OSS-71]

另一种类型的指导是提出问题，以帮助作者理解潜在的代码问题。

问一些问题，比如在场景中发生了什么，以引导他理解。如果这是由于缺乏对基础技术的理解，那么给出提示来提出知识。[MS-126]

6.5.4提供示例

一些审阅者更喜欢提供示例代码或指导作者在项目中使用其他编写良好的代码。

我通常会向原作者指出项目中现有的代码示例，以供参考。[OSS58]

6.5.5拒收直至合格

一些审阅者倾向于拒绝代码更改，直到它们满足项目质量标准。

在我确信代码更改符合团队的质量标准之前，我不会签字。如果作者不理解或不同意我的反馈，我通常会与他们坐下来详细讨论。[MS-132]

6.5.6与作者沟通

超过四分之一的微软受访者更喜欢通过电子邮件或即时消息讨论代码更改，而不是在代码审查工具中讨论。他们发现这种类型的交流有助于避免代码审查工具中的长时间讨论和作者的尴尬。这一比例远远高于开放源码软件答复者。

有时，对于更困难的问题，我会在旁边创建一个电子邮件线程，以便更好地来回讨论整个变化，而不是讨论审查中指出的一小部分。

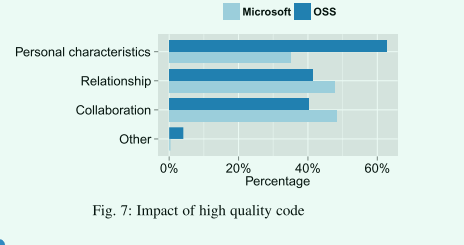
如果我认为改动写得很差，我倾向于保持私下谈话，以避免让作者尴尬。[MS-145]如果可行，一些Microsoft审阅者也倾向于与作者面对面讨论问题，以消除潜在的误解。

通常，最好的选择是去他们的办公室，看看他们是从哪里来的，以及他们是否疏忽或遗漏了信息。[MS-34]

6.6 RQ6：高质量代码的影响是什么？

每项调查中都有超过85%的受访者表示，高质量的代码或使用出色的方法解决问题会影响他们对代码作者的看法（Q17）。如图7所示，受高质量代码（Q18）影响的同行认知方面在OSS受访者和微软受访者之间存在显著差异（χ2=25.81，DF=3，P<.001）。

对于开放源码软件的受访者来说，高质量代码的最大影响是对代码作者个人特征的积极印象的增加。由于OSS参与者之间缺乏物理互动，社会技术互动（例如，通过代码审查）在形成对队友个人特征的印象方面变得更加重要[10]。这一因素对微软受访者的重要性较低可能是因为工业组织中的开发人员有其他观察和评估同行特征的方法，例如参加面对面的会议、近距离工作、经常交流以及参加非工作社交活动，如午餐。

相反，微软的受访者表示，高质量代码的最大影响是与代码作者建立更牢固的关系和未来的合作。因为微软大约75%的代码审查是由代码作者的队友执行的[14]，审查者很可能已经知道了作者的个人特征。相反，代码评审帮助评审者判断代码作者的智力和编码技能。高质量的代码可以带来更多的尊重、赞赏和信任。

6.6.1创造了对个人特征的积极认知

大约60%的OSS受访者和36%的Microsoft受访者表示，高质量的代码或出色的问题解决方法是代码作者个人特征的证据。首先，高质量的代码可以表明作者有能力，是该领域的专家，表现出色，并且拥有专业技能。

我会假设作者在某一特定领域有经验或技能，如果我在该领域看到好的作品。如果作者倾向于生成符合编码风格指南的干净、结构良好的代码，我将假设他/她以良好的组织和彻底的方式工作。[MS-68]

第二，高质量的代码可能是贡献者能力的标志。

我的观点是，你可以通过他们的变化的质量来推断开发人员的质量。没有经过深思熟虑的改变意味着草率的思考，但整洁而紧凑的改变表明了我所欣赏的思想的准确性。[MS-216]第三，代码更改的质量可以表明代码作者的敬业程度。

高质量的代码表明作者关心项目，并考虑了其变更的后果。高质量的代码也可以提升项目，因为新的想法被注入到社区中。[OSS-31]

最后，高质量的代码可能是作者对项目有很好理解的标志。

这意味着作者花了大量的时间来努力理解手头的问题，并想出了一个很好的解决方案。[OSS-220]

6.6.2帮助建立关系

大约37%的开放源码软件答复者和47%的微软答复者表示他们希望与杰出的代码作者建立关系。他们认为好代码的作者是值得信赖的，应该拥有额外的任务和特权。信任在OSS项目中非常重要，因为获得核心成员的信任是贡献者获得提交权限的唯一途径。

..“杰出的方法”并不多见，但写得很好、文档化良好的代码表明作者是可以信任的。他/她可以获得批准人权限或成为模块的维护者。[OSS-167]

此外，受访者表示对高质量代码的作者的尊重和钦佩有所增加。

高质量的代码说明了它的作者。我认为软件开发更像是一门艺术，而不是一门真正的工程学科。从这个角度来看，编写高质量代码的人是值得尊重和认可的人，因为他将自己的知识/经验与智慧相结合，创造出独特的解决方案。[OSS-180]

最后，编写高质量的代码还可以使作者在社区中赢得更好的声誉。

解决问题的能力给人留下了深刻的印象，这真的是精英统治在起作用。投入时间并以出色的方法解决问题的人会建立良好的声誉。[OSS144]

6.6.3鼓励未来合作

大约36%的开放源码软件答复者和48%的微软答复者表示，他们希望今后与高质量代码的作者合作，因为他们认为这些作者是他们可以向其学习的专家贡献者。

如果有人用“杰出的方法”写了一些代码，给我留下了深刻的印象，我可能会在那之后阅读他们所有的代码审查，以便了解更多。[MS-103]

一些开发人员经常自愿审查这些作者提交的代码更改，主要是为了学习。我将更有可能在未来咨询这个人，因为他们在代码问题上是一个久经考验的执行者。

我可能会通过注册代码审查来更多地关注他们未来的工作，但更多的是了解他们的工作，而不是逐行挑出代码。[MS-333]

除了学习之外，开发人员在解决问题遇到困难时，还会向优秀的代码作者寻求帮助。

我知道这是一个在面对困难问题时可以求助的人，可以指望他对我的代码进行适当的审查并提出改进建议。[OSS-109]

持续提交高质量的代码可以提高作者的可信度，因为受访者表示，如果很忙，他们会花更少的时间来审查这些作者的代码。

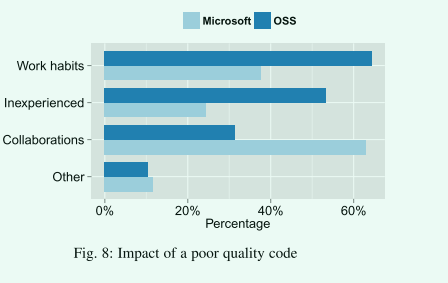
如果这个人在我非常忙的一天发送另一个代码审查，我不会担心查看它，因为我相信他们在这个新的代码审查中也“做了正确的事情”。[MS128]

最后，编写高质量的代码更改会提高可靠性，并分配更复杂或关键的任务。

看到写得很好的代码增加了我对作者的信任，我知道我将能够在未来的高复杂性或高重要性的任务中依赖该作者。[MS-341]

6.7 RQ7：低质量代码的影响是什么？

每项调查中都有超过四分之三的受访者表示，写得不好的代码会对他们对代码作者的看法产生负面影响（问题13）。如图8所示，在OSS调查对象和微软调查对象之间，受编写较差的代码（Q14）影响的同行认知方面存在显著差异（χ2=52.79，DF=3，p<.001）。开放源码软件的受访者更有可能对写得很差的代码的作者的经验和个人工作习惯形成负面印象。形成负面印象的原因与形成正面印象的原因大致相同（RQ6）。此外，微软的受访者认为低质量代码的作者是不称职的。他们不太可能在未来与这些作者合作，因为这种合作所需的时间和精力预计会增加。



6.7.1对工作习惯产生负面印象

大约62%的开放源码软件答复者和38%的微软答复者表示，低质量代码对他们对作者工作习惯的看法产生了负面影响。和以前一样，微软的受访者可能认为这一因素不如OSS受访者重要，因为开发人员有许多其他互动方式，能够评估彼此的特点和工作习惯。更具体地说，审稿人认为提交低质量代码是作者粗心大意或对同行缺乏尊重的表现。

代码写得很差，不遵循指导方针等。给人留下不好的印象，它给人一种草率的印象，就像编写代码的人没有花时间提供具有适当质量的代码，却期望它被审查。在我看来，提供草率的代码是缺乏尊重的表现。[OSS260]

其次，审阅者认为低质量代码的作者懒惰或缺乏对项目的奉献精神。

作者要么没有投入足够的时间来调查他正在解决的任务，要么对他正在构建的系统不了解。不管是哪种情况，他都应该花更多的时间去了解他在做什么。[MS-41]

虽然低质量的代码确实会影响人们的认知，但评论员意识到人们确实会犯错误，因此，当作者无法或不愿意从以前的错误中吸取教训时，他们可能会原谅最初的几个错误，并开始形成负面印象。

对于偶尔糟糕的代码，影响不大-人们有休息日，或者可能只是误解了他们正在修改的特定代码区域。但是，如果作者继续犯类似的错误，那么这将大大降低我对其能力的看法。[开放源码软件-176]

此外，并非所有的错误都有相同的影响。评论者更容易接受由于缺乏知识或理解而导致的错误，而不是容易避免的错误。

如果代码包含错误就可以了，我不会认为作者粗心大意。然而，如果代码包含编码风格、可读性、重复代码和其他容易避免的问题，我会认为作者粗心大意，没有致力于项目。[OSS-197]

6.7.2表明缺乏经验

大约51%的开放源码软件答复者和24%的微软答复者表示，低质量代码表明作者在项目或编程方面缺乏经验。

这可能意味着他们还没有理解他们正在修改的代码是如何工作的，因此他们的任何贡献都可能需要非常仔细地审查。[OSS-196]

审稿人还认为，低质量代码的作者可能不称职或缺乏智慧。

无论程序员的经验水平如何，他们都可以写出清晰、可读、健壮的代码。令人惊讶的是，人们可以在没有学会这样做的情况下走很长的路。这让我质疑他们天生的智慧和对项目的奉献精神。[OSS-25]

6.7.3影响未来合作大约

62%的Microsoft受访者和27%的OSS受访者表示，对编写糟糕代码的作者形成的印象影响了他们未来的合作。许多受访者对拙劣代码作者的能力表示怀疑。

写得不好的代码通常向我表明作者缺乏编码经验或技术技能，这（负面）影响了我如何看待作者在工作中的总体表现。[MS-45]

失去尊重是阻碍未来合作的另一个因素。

根据糟糕代码的“严重性”，我可以感觉到，在极端情况下，作为代码作者，我会对这个人和他们的智慧失去尊重。例如，思考“他们是否试图构建/测试它”，或者“他们为什么认为这是正确的方法？”。这应该简单得多，但他们不知道。[OSS-78]

在极端情况下，评审员可能会失去对低质量代码作者的信任，并仔细检查该作者未来的更改。

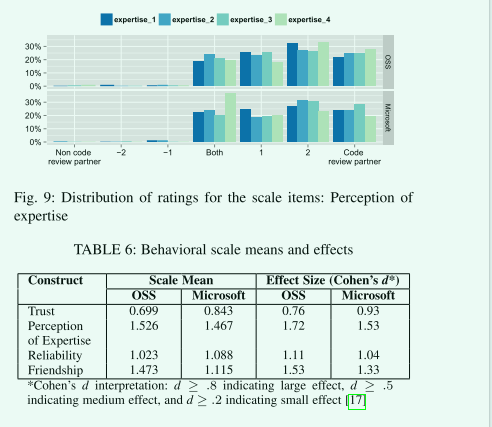
我不太信任开发人员，并且知道我必须对未来的更改进行更详细的代码审查。[MS-349

最后，由于编写糟糕的代码需要更长的时间进行评审，一些评审员不太可能接受这些作者的评审请求。

写得不好的变更也需要更多的审查时间，所以我觉得一个人如果总是写得不好，从长远来看会浪费很多人的时间。我也期望那个人的变更写得很糟糕，并且不期待审查变更的前景。[MS-166]

6.8 RQ8：代码审查对同行印象的影响是什么？

使用行为量表，我们专注于了解代码审查对同伴印象形成的四个方面的影响：信任、可靠性、专业知识感知和友谊。为了便于分析和呈现结果，我们对量表进行了重新编码，以使量表项目对印象形成的影响更加明显。重新编码的等级是：-3：描述非代码评审合作伙伴，而不是代码评审合作伙伴，0：同等地描述两者，3：描述代码评审伙伴，而不是非代码评审伙伴。为了避免负比例值使结果产生偏差，我们在数据收集过程中没有使用此比例值。



例如，图9显示，对于四个专业知识感知量表项目，大多数受访者（约70%至80%）认为他们对代码评审合作伙伴的专业知识的感知优于非代码评审合作伙伴。所有四个量表都显示出类似的趋势。

表6显示了两次调查中四个行为量表项目的项目均值。在所有情况下，量表平均值均为正值，且显著高于量表的中点（0-两者相等）。对于行为量表问题，我们没有观察到两次调查结果之间的任何显著差异。我们还使用Cohen s D（表6最右边的两列）估计了效应大小。八个病例中有七个有很大的影响。只有开放源码软件调查中的信任量表显示出中等效应大小。结果表明，代码评审对OSS和MS项目中代码评审参与者之间建立四种类型的同行印象（即信任、专业知识感知、友谊和可靠性）总体上有很大的积极影响。

这些结果为RQ6（高质量代码的影响）和RQ7（低质量代码的影响）的结果提供了一些见解，这表明代码评审对印象形成既有积极的影响，也有消极的影响。这项分析表明1）代码评审对印象形成有很大的积极影响，2）大多数受访者对代码评审合作伙伴的看法优于非代码评审合作伙伴。

7讨论

本节对第6节所述调查结果的详细分析进行了进一步讨论。特别是，本节强调了从结果中产生的七个主题。

7.1开放源码软件和微软之间的差异在强调最重要的代码审查方面，开放源码软件调查对象与微软调查对象存在显著差异。开放源码软件审查员的重点是与核心团队成员建立关系。在通过代码审查形成对队友的印象时，OSS受访者表示，代码作者的个人特征和工作习惯是最重要的。这种强调是有意义的，因为开放源码软件团队的成员可能没有机会通过更传统的互动类型（即面对面的工作和社交互动）来形成对其队友的印象，而像微软这样的商业组织的成员则会有这种印象。因此，代码审查对于形成队友的印象变得更加重要。相反，微软的受访者认为知识传播方面更重要。代码审查作为一种媒介，指导新团队成员了解项目设计、编码约定和可用的API或库。

同样，在决定是否接受代码审查请求时，开放源码软件受访者最重要的因素是他们与代码作者的关系以及代码作者的声誉。这种关注是由维持当前关系和改善与知名开发商关系的愿望所驱动的。相反，对于微软的受访者来说，在决定是否接受审查时，最重要的因素是代码作者的专业知识（即，开发人员是否编写了他/她可以学习的好代码）和审查变更所需的努力。在决定邀请谁审查其代码时，最重要的因素是审查者的专业知识（即，他/她是否具有审查该代码的专业知识并能够提供有用的反馈）。

除了这里提到的差异之外，OSS和Microsoft开发人员的结果是相似的。第7.2-7.5节描述了开放源码软件调查对象和微软调查对象的类似结果。

7.2代码审查的好处

虽然有经验证据表明代码审查可以提高软件质量[18]，[57]，但代码审查的好处要广泛得多。关于这些其他好处的证据大多是道听途说。这些调查的结果开始为这些益处提供更多的证据。在两次调查中，几乎所有的调查对象都发现代码审查对他们的项目很重要，原因包括：知识传播、关系建立、更好的设计和确保可维护的代码。对于大型和长期项目，这些收益可能非常重要，并且很难通过其他方式实现。因此，即使是很少提交低质量更改的高技能开发人员的项目，仍然可以从代码审查实践中受益。

7.3同行印象形成

对于OSS和MS参与者来说，代码审查的关键非技术好处之一是它在印象形成中的作用，即开发人员如何形成队友的意见。调查结果显示，代码审查最重要的社会因素是获得对团队成员专业知识的准确认知。提交审查的代码的质量是形成队友认知的一个重要方面。例如，简单、易于理解、自我文档化并且需要最少审查时间的代码更改会得到审查者的高度赞赏，并且可能会提高社会地位。开发人员对队友的态度是积极还是消极可能会影响未来的代码审查。例如，受访者表示，他们对不值得信任的队友提交的代码进行了更彻底的审查，而不是对他们视为专家的队友提交的代码进行审查。此外，在代码审查期间形成的印象也可能影响未来的协作、关系和工作实践。因此，代码审查不仅是确保代码更改质量的关键实践，也是形成成功项目的社会基础的关键实践。

7.4对未来合作的影响

在两次调查中，超过四分之三的代码审查参与者对他们的同行有强烈的正面印象，这表明代码审查可能会影响未来的合作。当审阅者从作者那里找到高质量的代码时，它可以通过两种方式增加协作。首先，她/他更有可能报名审查作者未来的变化，以便从中学习。第二，他/她认为作者是专家，能够提供改进他/她的代码的建议，并将作者添加为其更改的审阅者。相反，写得很差的代码通常需要花费更多的精力来审查，并且审查者可能不会接受来自糟糕代码的作者的未来审查请求。此外，对他们的专业知识水平的不确定性可能导致开发人员避免要求糟糕代码的作者审查他/她的代码。结合这两种情况，开发人员的代码审查合作伙伴更可能是那些他/她认为是优秀作者和专家的同行。相反地，他/她被判定为差的代码作者的同行将由于不频繁的交互而成为非代码审查伙伴。

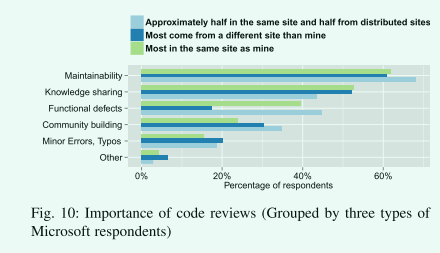
7.5感知专业知识的影响

评审员对代码作者专业知识的感知不仅影响代码评审请求的接受或拒绝，还影响代码的审查水平。首先，在接受输入代码评审方面（见第6.3节），感知到的专业知识影响不一。一些受访者倾向于审查专家提供的代码，以学习并尽量减少审查时间。其他受访者优先考虑新来者的评论，或关注最需要关注的领域。这两种方法可能都是正确的，这取决于评审员的专业知识。

第二，就规范的审查水平而言，如果审查人员对特定设计选择不确定，则他/她更有可能信任专家的设计选择，并质疑非专家的理由。此外，如果评审员没有足够的时间对专家作者的代码进行彻底评审，他/她将根据专家作者正常正确执行的假设，在粗略评审后批准该代码。但是，作者只有在持续提交高质量代码更改后才能获得此专家身份。

7.6分布式团队与并置团队的影响

对微软开发人员重复调查的目标之一是调查OSS项目的分布式性质在多大程度上影响了代码审查的理解和实践。为了研究这种影响，我们调查了微软内部分布式团队的成员和并置团队的成员。我们预计分布式团队的成员会强调代码审查的人际关系方面，因为他们亲自形成这些关系的能力有限（就像并置团队的成员一样）。有趣的是，在分析来自微软调查的数据时，我们发现来自并置团队的开发人员的反应与来自分布式团队的开发人员的反应几乎没有差别。例如，图10显示了三组Microsoft受访者如何认为代码审查对他们的项目很重要。事实上，正如第7.1节所讨论的那样，总体而言，开放源码软件调查的受访者与微软调查的受访者之间的差异要大得多（无论受访者是在并置团队还是分布式团队）。因此，我们最初的假设，即来自微软分布式团队的受访者的回答与来自OSS团队的受访者的回答相似，是不被支持的。



这一令人惊讶的结果表明了两种可能的解释：1）受代码审查影响的某些类型的印象（例如，编码能力）可能不依赖于面对面的互动；2）代码审查过程（例如，审查接受和未来的合作）可能更多地取决于项目文化（即，OSS项目与商业项目具有不同的文化[28]），而不是开发人员的物理位置。

7.7有偿与志愿开放源码

软件开发人员开放源码软件参与者的动机可能会受到他们是否因其贡献而获得财政补偿的影响。此外，有偿开放源码软件参与者的目标可能与自愿开放源码软件参与者的目标不同。先前的研究已经确定了付费参与者和志愿者参与者在印象形成（即，付费参与者更有可能在面对面会面的基础上形成印象）和感知体验（即，志愿者更有可能感知与同伴一起工作的负面体验）方面的几个差异[10]。

作为对RQ2的回应，我们发现，与自愿的OSS参与者相比，付费的OSS参与者与更多的同行合作，并在代码审查上花费更多的时间。这一观察结果可能表明，付费的开放源码软件参与者充当了开放源码软件项目的看门人。

对于其余的研究问题，涉及代码审查的重要性、代码审查过程和代码审查的影响，结果没有显示付费开放源码软件参与者和自愿开放源码软件参与者之间的任何差异。因此，无论OSS参与者是有偿的还是志愿者，似乎都不会影响代码审查过程的关键方面，也不会影响代码审查如何影响同行印象的形成。

8对有效性的威胁

本节讨论已解决和未解决的有效性威胁。它围绕四种常见的有效性威胁类型进行组织。60%

8.1内部效度

参与者选择是内部效度的主要威胁。受试者人群由至少参与了30次代码评审的评审员组成（作为作者或评审员）。使用不同的阈值可能会产生不同的结果，但我们没有证据表明这种情况。由于八个研究问题（RQ2-8）中有七个与代码审查过程相关（即接受审查请求、判断不良代码、改进不良代码、好/坏代码的影响），我们坚信，如果没有足够的代码审查经验，开发人员无法为这些问题提供适当的答案。

此外，还有一种威胁，即只有那些在代码审查方面有积极经验的受试者才会花时间回应调查。没有证据表明这种自我选择发生了。但是，即使是这样，因为调查的目的是收集代码审查各个方面的信息，那些有积极经验的人可能会提供最好的反馈。

8.2构想效度

调查设计过程特别关注减少结构有效性威胁。这一过程耗时约8个月，包括专家审查和多次试点测试。设计过程包括以下减少偏差的做法：

•将关于感兴趣的主题的问题放在其他调查问题之后，以防止假设猜测，

•以随机顺序呈现量表问题，

•在所有相关页面上提供代码审查合作伙伴和非代码审查合作伙伴的明确定义，以及

•以公正的方式仔细措辞问题。

第三，我们进行了多重信度和效度测试，具有广泛使用和高度推荐的措施，以确保建构效度。

8.3 8.3外部效度

由于开放源码软件界的广泛多样性，其结果可能无法代表所有开放源码软件项目。事实上，由于大多数受访者来自著名的、成功的开放源码软件项目，他们可能是技能更高、积极性更高的开放源码软件开发人员。在其他类型的开放源码软件项目中，代码审查对软件质量和团队社会结构的影响可能有所不同。

就微软开发人员而言，他们可能并不代表所有的商业组织。为了减少这一潜在威胁，受访者来自开发流程（例如瀑布式与敏捷）、硬件平台（例如移动、桌面、服务器和数据中心软件）、部署方法（盒装产品与Web服务）、操作系统（IOS、Windows、Windows Phone和Linux）、位置（美国、欧洲和亚洲）和工作流程（例如一些团队要求对所有签核进行两次审查，其他团队则较为宽松。一些人想在入住和考试前复习，另一些人则在考试后复习。有些包括测试人员和开发主管的评审，有些则不包括）。在Microsoft中，不同项目的软件开发流程、项目管理和发布周期各不相同。有趣的是，在先前的代码审查研究中，Rigby和Bird调查了多个商业组织和开源项目中的代码审查实践和度量[45]。令人惊讶的是，他们发现在代码审查方面研究的不同系统之间几乎没有差异，这支持了这样一种观点，即一家公司的代码审查结果可能与其他组织相关。

此外，微软的代码审查工作流程与其他大型商业组织（如Facebook[33]、Google[56]、VMware[3]、Cisco[18]和Oracle[55]）所使用的类似，这些组织已采用强制性代码审查实践。在这些组织中，代码审查已成为一项重要的软件质量保证实践，类似于测试、跟踪/发现错误和自动化构建系统，这些都是成熟软件工程项目的方面。

关于微软等大公司的工业研究，一个常见的误解是，微软的软件项目不能代表其他软件项目。虽然项目的规模可能更大，但Microsoft的大多数开发实践都是从通用软件工程社区改编而来的，并且也在Microsoft外部使用。另一个常见的误解是，一家公司或一个项目的实证研究不够好，为学术界提供的价值很小，对科学发展没有贡献。历史证据表明并非如此。Flyvbjerg提供了几个有助于物理学、经济学和社会科学发现的个案案例[24]。贝弗里奇（W.I.Beveridge）在《社会科学》（Social Sciences）一书中再次指出：“更多的发现来自密集的观察，而不是应用于大型群体的统计数据”[7]。即使在SE领域，大型商业公司（如微软[9]、[39]、谷歌[35]和思科[18]）先前的案例研究也提供了有用的见解。请注意，这一论点不应被解释为对集中于大样本或整个人群的研究的批评。对于Basili[4]所倡导的经验知识体系的发展，这两种类型的研究都是必不可少的。

8.4结论有效性

每项调查的回复数量都很大，足以减轻小样本量带来的任何威胁。此外，卡方检验（在本分析中最常用）并不假设数据呈正态分布。对于非正态分布的变量（根据Shapiro-Wilk检验），我们使用非参数检验。

9结论

本文描述了两个调查的结果，以更好地理解执行代码审查的实践和动机。这些结果对研究人员和从业人员有几点启示。首先，尽管只有四分之一的代码审查意见是关于功能缺陷的，但不应阻止从业者进行代码审查。代码审查提供了其他几个好处（即知识传播、关系建立、更好的设计和确保可维护的代码），这些好处对于大型或长期项目至关重要。有趣的是，大多数代码审查研究都集中在缺陷检测上。开发人员认为更重要的代码审查的其他方面没有得到太多的关注。因此，代码审查的这些其他方面（即建立关系、知识共享、实现更好的设计）需要额外的重点研究。

其次，这些调查的结果表明，代码审查对建立关系和未来的合作有很大的影响。审查意见的措辞粗心大意可能会导致代码作者的负面情绪，并阻碍未来的合作。例如，如果审阅者提供建设性批评，则作者更有可能进行所需的更改，而如果审阅者的评论被视为攻击，则作者更有可能与审阅者争论。因此，审阅者应该仔细考虑代码作者将如何听取他们的审阅意见。这一结论保证了在两个方向上的进一步研究：1）实证验证代码审查意见中的情绪表达（即正面或负面）如何影响代码审查结果和长期合作，以及2）如何帮助审查者在代码审查期间阐明适当的意见。

最后，有效的代码审查需要审查者付出大量的努力来彻底理解代码。这项研究的结果表明，审查人员更喜欢审查简单、自我记录和易于理解的代码更改。在提交代码更改以供审查时，作者应牢记这些代码特征。这一结果也可能引起程序理解研究人员的兴趣。用于理解代码更改的大量时间可以通过适当的程序理解技术来改进。