### 基于客观和主观数据的在分布式软件开发中同行代码审查的有效性研究

### **作者：Eduardo Witter dos Santos and Ingrid Nunes**

**摘要**：代码审查是提高软件质量的潜在方法，为使之生效，它取决于不同的因素，并且这些因素中许多都已经在文献中进行了调查，以确定能提升最终代码质量的场景。然而，与分布式软件开发相关的因素变得越来越普遍，其却很少被探索，地理距离也可能会给代码审查过程带来更多的挑战。因此，在本文中将介绍一个混合方法研究的结果，该方法研究了代码审查在分布式软件开发中的有效性。我们调查了一些可能会影响同等代码审查结果的因素，这项研究主要是对收集到的一些客观数据的分析，这些客观数据来源于一项包括201名成员的软件项目以及一份对50名具有代码审查经验的从业者的调查结果。我们对客观数据的分析得出了这样的结论：大量更改的代码行数会导致延长代码审查的期限以及会减少消息的数目，虽然涉及到的团队、地点和参与审查者的数量通常会提升审查者的贡献，但对持续的时间有严重的影响。这些结论与在持续时间和参与情况对代码审查的影响的调查中所得出的结论是一致的。然而，在关于对代码审查的贡献的影响中，这一意见不同于从历史数据中得出的结果，其主要体现在分配方面。

**关键词**：代码审查、分布式软件开发、实证研究、调查

#### 1 背景

在软件开发中，代码审查是一种提升软件质量的普遍做法，它是基于同行静态代码分析的基础之上的。有研究表明，代码审查能够减少代码发布后检测到的缺陷的数量，主要是它有足够的代码覆盖率、相关的约定以及审查者的参与（McIntosh 等人 2014）；其次，代码审查是促进知识共享的公认方法，它也同样有利于开发者以及审查者（Hundhausen 等人 2013）；最后，它还利于团队协作，因为它创建了源码的集体所有权，这是集体协作工作的结果，而非个人（Bacchelli 和Bird 2013，Thongtanunam 等人 2016b）。现在，代码审查不像前几十年的软件开发那样正式，在过去，它是一种典型的代码复查形式（Fagan 1986），它需要正式的会议和审核表（Kollanus 和 Koskinen 2009）。现在，这种做法变得更加随意，它被称之为*现代的代码审查*（MCR）（Bacchelli 和Bird 2013），现代代码审查通常是由工具辅助执行的，比如Gerrit（Google 2017a）。

代码审查的有效性取决于不同的因素，当它不能达到预期效益时，它就变成了一个既费时又费力的任务（Czerwonka 等人 2015；Thongtanunam等人 2016a）。比如，如果在完成变更和同行代码审查间存在时间间断，这就可能会导致工作部分受阻，还可能会影响整个软件的发布（Thongtanunam 等人 2015b）。同样，如果在代码审查上缺乏活力，这样团队的工作进程就会被拉长，因为在待审的同时可能又会接到新任务。此外，编码任务和审查任务之间的切换可能会给开发者的工作带来负面影响。

为了了解哪些因素对代码审查的有效性产生了积极和负面的影响，之前就有人进行了相关研究（Thongtanunam等人 2015a；Baysal等人 2016；Yang 2014；Bosu等人 2015），其中包括一些技术性和非技术性的因素，例如补丁大小、变更的性质或者是开发者的公司。此外，为了评估代码审查的有效性，他们会采用不同的标准，比如审查的持续时间和代码审查之后被检测出的缺陷的数量。因此，我们也得出一些和代码审查相关的结论，例如，相比来自同一团队的开发者，来自其他团队的开发者在代码审查中提出反馈虽然更少，但更有用（Bosu等人 2015）。尽管到目前为止其他研究已经得到了所有有意义的结论，但在分布式软件开发中，代码审查在不同地理位置背景下的研究却十分有限（Sengupta等人 2006），尽管其在过去几十年内变得越来越流行。在90年代后期，研究人员专注于在分布式场景下使用正式代码检测（包括会议）（Perpich 等人 1997; Stein 等人 1997）。相反，在现代的代码审查中，工具的支持和异步通信有助于处理地理分布不同的情况，然而以往的研究中却没有探索地域分布对代码审查结果（例如审查周期与审查者的参与）的影响，在最近的对分布式软件开发下的代码审查的研究中，也只限制于代码检测的经验报告（Meyer 2008）。

因此，本文将专注探讨技术性和非技术性的因素是如何影响在分布式软件开发环境下代码审查的有效性指标的。我们展示了混合法研究的结果，其中我们调查了四个影响因素之间的关系和代码审查的有效性，四个影响因素包括代码的更改行数、涉及的团队、设计的地点以及活跃的审查者。由于没有单一的指标来衡量审查是否有效，所以我们衡量和分析了不同的审查结果，这些结果可以被看作是审查有效性的一个指标，例如审查者的参与与评论的数量。研究包括（1）从软件项目收集的客观数据的分析；（2）对具有审查经验的50名从业者的调查。这项研究是基于先前提出的工作的延申（Witter dos Santos 和 Nunes 2017），这项研究补充了之前的这项调查，这也使我们能够将所得结果与这两种研究方法进行比较。

我们研究的第一部分，称之为仓库挖掘，其来源于从代码审查数据库中提取的庞大的数据（8329次提交和39237个评论），而这个代码审查数据库是在72周内从一个包括201名成员的项目中获得的。从这些调查结果来看，我们得出了一下结论：大量的更改代码行会导致审查过程的持续时间变长并且会减少消息的数量，尽管涉及到的团队、地点和审查者的参与通常会改善审查者的贡献，但对周期会有严重的影响。这些结论与在持续时间和参与情况对代码审查的影响的调查中所得出的结论是一致的。然而，在关于对代码审查的贡献的影响中，这一意见不同于从历史数据中得出的结果，其主要体现在分配方面。

本文的剩下部分将以以下方式来排列。首先，我们会在第二部分讨论下相关的工作；然后，我们会在第三部分提供一些关于目标项目的细节以及描述下目标项目的代码审查过程；接下来，我们会在第四部分描述我们的研究环境；在第五部分，我们会分析在第一部分和第二部分中呈现出来的结果；在第六部分将呈现我们的讨论结果；最后，我们得出的结论将在第七部分中呈现出来。

#### 2 相关工作

自从Fagan（1976）在正式代码审查方面的开创性工作以来，许多研究人员（Parnas 和 Weiss 1985; Bisant 和 Lyle 1989; Martin 和 Tsai 1990）提出改善这种结构良好的、阶段性的代码审查形式的方法。随着分布式软件开发的流行，其他研究人员（Perpich 等人 1997; Stein 等人 1997）也探索了当相关人员不能在约定地点见面时如何使代码审查变得可行。尽管分布式开发受到许多研究人员和从业者的欢迎，但自从2000年早期以来，正式代码审查及其变体却还是很少受到关注（Kollanus 和 Koskinen 2009）。

最近，许多集中在现代代码审查上的工作已经完成，这些工作包括从调查代码审查成功的因素到引荐合适的审查员的方法。比如在Balachandran （2013）的方法中，他推荐的是那些对正在审查的部分代码做最近更新的那些人，他的这个方法被Thongtanunam（2014）等人改善，因为对于那些有特定特征的项目，如果使用文件路径相似性这一方法（FPS），它就要考虑到以前具有相似路径或文件名的更改这一因素，这些方法也因考虑到过去提交信息的相似性（Xia等人 2015）以及可能审查者的最近活动（Zanjani等人 2016）等原因而得到扩展。Viviani 和 Murphy（2016）就采取了另一个方向，他们为每一位审查员确定待审查的优先次序，而不是为每个变化都找到一位最佳的候选审查员，这是因为几个项目的审查请求都高度集中在一小群贡献者中（Yang 2014）。

尽管在代码审查领域做出了这些重要的贡献，但去了解影响代码审查有效性的因素也是很重要的，例如在推荐审查员的同时提供改善的基础。然而很多研究还致力于探讨对代码审查更深层次的理解以及它的影响因素（比如代码更改行数和个人经验）和结果（审查周期及审查员之间的探讨）。尽管我们的研究也和这些相似，但是他们没有专注于分布式软件开发。我们接下来就来讨论基于现有调查的技术性和非技术的影响因素。

**技术性因素的相关调查：**研究的是技术性因素的不相关性。Thongtanunam等人（2015a）的研究中表明，如果审查者没那么严格，他们在缺陷发生率高的文件中检测出的缺陷就会越少，他们就只关注了表层，比如说编程规范，而没有关注到需求的功能方面。在最近的一项研究（Thongtanunam 等人 2016a）中，同一开发者指出，修复bug通常比实现一个新特性能更快的收到第一个反馈，此外，他们报道说，相比那些那些描述不当的提交信息，较详细的解释性提交信息的更改更能受到审查者的关注。

针对代码审查持续时间（工作日内），我们调查了一些可能的影响因素。Bosu等人（2015）得出一个结论，即在大多数的分析案例中，补丁大小会影响审查的持续时间，而发布计划中的任务优先级和被影响的软件组件只会偶然影响到由Baysal等人（2016）分析的一些项目。

**非技术性因素的相关调查：**一些非技术性的因素最近也受到关注，就如Czerwonka（2015）等人陈述的，在公司内部或项目中自然而然出现的社交网络应被当作是审查者的特殊技能或他们的审查意愿。对三个开源项目（Yang 2014）的社交网络的分析表明：那些活跃在这些社交网络中的审查员发挥着中心作用，通常他们也是最重要的贡献者。Bosu等人（2015）观察到，在一个特定的组织中，75%的代码审查反馈来自于开发者团队成员，但相比来自其他团队的反馈来说作用稍微逊色一点。相反的，Baysal等人（2016）指出：基于对几个研究案例的分析，当多个团队对同一项目做出贡献时，代码审查反而会花更多的时间完成，并且这会导致更高的错误率，这取决于哪一个团队该编写或审查修补程序。

在代码审查中，开发者的经验也是一个非技术性的因素。公司的高层人员和公认的专家通常会接收到更优先、更快、更详细的反馈（Baysal 等人 2016; Rahman 等人 2016）。审查员的经验也是一个非技术性的因素，基于对一个大公司的调查表明（Bosu 等人 2015）－一年内，公司提供反馈的质量提升后，会基本稳定在这个水平上。

在涉及到三个大型开源项目的研究中，Thongtanunam 等人（2016a）也对非技术性因素进行了调查，他们重点讨论了代码审查如何受到审查文件中先前发生事件的影响。他们得出的结论是（1）审查文件在一开始收到很慢的反馈时，在将来也可能收到很慢的反馈；（2）审查文件在过去如果有更多的开发者和审查者的话它也会受到更多的关注；（3）更改的文件的数量、目录以及提交消息的长度同样也很重要。

**总结：**考虑到影响代码审查的许多因素都已被调查过，我们把先前每一个研究的分析都总结到了表1中。这张表的行表示已被检测的影响因素，列表示与代码审查相关联的审查结果，在表格中，我们列出了影响因素和审查结果之间的联系。

表1：影响代码审查因素及以往研究成果综述

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 影响因素 | 未讨论 | 评论有用性 | 反馈时延 | 参与审查 | 发布后的缺陷 | 审查时间 | 审查迭代 | 审查质量 |
| 先前缺陷的数量 | ♣ |  | ♣ | ★ |  |  |  | ★ |
| 受影响的模块 |  |  |  |  |  | ♠ |  |  |
| 开发者经验 |  |  |  | ♣ |  | ♠ |  |  |
| 开发者的公司 |  |  |  |  |  | ♠ |  |  |
| Bug修复或新特性 |  |  | ♣ |  |  |  | △ |  |
| 提交信息的大小 | ♣ |  |  |  |  |  |  |  |
| 其他审查者 |  | ◇ |  |  |  |  |  |  |
| 先前讨论的长度 | ♣ |  |  |  | ♥ |  |  |  |
| 开发者数量 | ♣ |  | ♣ |  |  |  |  |  |
| 审查者数量 | ♣ |  |  |  |  |  |  |  |
| 补丁大小（文件） | ♣ | ◇ |  |  |  |  |  |  |
| 补丁大小（代码行数） |  |  |  |  |  |  |  |  |
| 先前反馈时延 |  |  | ♣ |  |  | ♠ | △ |  |
| 优先级 |  |  |  |  |  |  |  |  |
| 审查范围 |  |  |  |  | ♥ † | ♠ |  |  |
| 审查速度 |  |  |  |  | ♥ |  |  |  |
| 审查者经验 |  | ◇ |  |  |  | ♠ |  |  |
| 源码类型 |  | ◇ |  |  |  |  |  |  |

♣ Thongtanunam 等人 （2016a）

◇ Bosu 等人 （2015）

♥ McIntosh 等人 （2014）

♠ Baysal 等人 （2016）

△ Beller等人（2014）

★ Thongtanunam等人（2015a）

† Shimagaki 等人 （2016）

其中一些研究分析了针对FLOSS（免费、开源软件）项目的消息完整性，比如OpenStack、Qt和 LibreOffice项目，这些项目都具有分布式软件开发的特点。而我们强调的是，这些研究大多数都没有调查地域分布这一影响因素：即与地域分布相关的因素是一种随机变量，而不是自变量。就拿Baysal等人（2016）报道的来说，一些被分析的公司中，有的公司具有位于同一地点的小组，而其他公司则使用分布式软件开发，然而Baysal他们并没有把这个因素当作因变量。相同的，Bosu等人（2015）发现来自其他团队的评论更加有用一点，但他们却不考虑共同位置以及参与的团队的数量这些因素。

可以看到，影响因子和结果的不同组合已经存在分析。和以往的研究不同的是，这次我们的研究侧重于分布式软件开发，因此，我们会关注到其他影响因素，比如涉及的城市和团队的数量。我们调查的一些影响因素里像补丁大小（代码行数）早已经被研究了，但不是在分布式软件开发的场景下。此外，我们分析了代码审查的四种不同结果，这些结果将在下一节中与我们的研究设置的其他详细信息一起描述。

#### 3 研究科目

我们的研究是基于对一些数据的分析，这些数据是从一个商业的软件项目和一家软件开发公司的开发者那里收集到的，由于项目规模很大，我们才能够收集有关其代码审查的大量信息。接下来我们会描述项目的代码审查过程，提供有关收集的数据的详细信息，并描述我们调查的参与者的特征。由于保密协议，我们就不能再提供进一步的信息了。

##### 3.1 代码审查过程

图1概述了项目的代码审查过程。首先，开发者发送一段代码进行审查，审查者可以是自己，可以是其他人，任何人也都可以在任何时间添加自己或他人作为审查者，这就允许任何感兴趣的开发者来做出贡献。此外，在我们的目标项目中，Gerrit配置了逐个审阅者插件（Google 2017c），该插件根据Balachandran（2013）提出的对要审查的文件的最新更改自动添加审查者，然后由审查者自动分析代码，该审查员需要检查若干质量标准，例如编译，循环复杂性，缺少的文档，失败的单元测试以及其他静态分析和运行时验证。如果任何关键测试失败，这种自动验证通常需要不到15分钟的时间来执行并拒绝更改，以便开发者可以修复报告的问题。人工审查员和开发者可以为每行代码进行讨论、询问和提供建议。 此外，每位审查员都可以投票使用以下值之一来汇总其反馈。

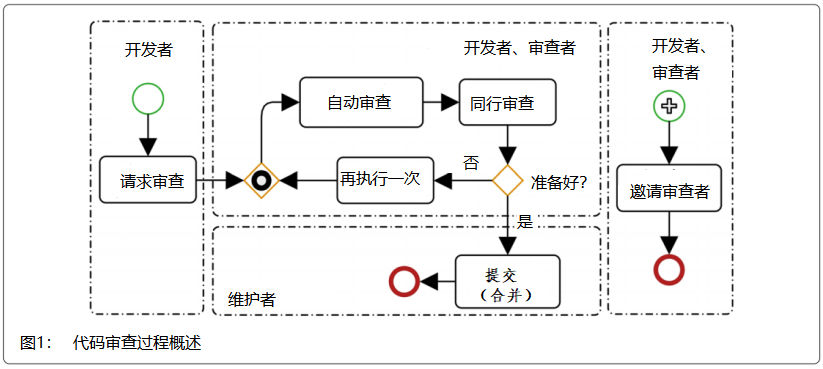
**否决** 审查员认为，如果不解决报道中的问题或者回答中所提出的问题，这会导致无法整合变更，还会阻止提交的合并。

**拒绝** 审查者建议在变更合并前进行修复。

**中立** 审查者通常会问一些比较容易回答的问题。

**接收** 审查者认为，尽管更改是足够的，但它需要来自其他开发人员的更多审查。

**批准** 只有与提交相关联的模块的维护者才有这种否决权，因为他们负责模块质量。维护人员可以执行技术审查，但还必须验证相关开发人员在受邀审查者列表中是否遗漏，以及代码审查的整体状态是否充分。



需要注意的是，除了维护人员之外，所有受邀的评审人员都没有义务提供反馈。在批准更改之前，每个模块的维护者应该考虑最重要的审阅者是否已经审查了代码。最后，如果满足以下所有条件，则可以提交经审查的代码：（i）自动审稿人不会拒绝；（ii）没有否决权；（iii）维护人员已批准更改。 如果所有这些条件成立，维护者就能够将更改合并到目标分支中。

##### 3.2 分析的数据

这次研究的目标项目涉及到了使用C、C++以及Yang（国际工程任务组（IETF）2017）语言开发的用于路由器和交换机的嵌入式系统的操作系统。这个项目一共有269个仓库，其中有63个使用Python、Vagrant和Ansible来专门测试自动化。我们考虑到操作系统代码及其测试都是同一项目的一部分，因为开发人员为每个任务实现了固件和测试，所以我们把仓库都配置为拒绝合并而不进行代码审查。

数据是从2014年7月开始的72周这个时期所挖掘的。在收集到的数据中，我们一共有11109条代码审查，在过滤这些数据后（见下一节），我们获得了与39237条评论相关的8329条代码审查（平均评论为4.7条评论）。此类代码审查与以下相关：（i）201名经验丰富的开发人员；（ii）同一国家和时区的4个不同城市的4个开发地点；（iii）21个不同的团队。给定团队的成员在同一地点工作－即没有分布式团队，所有的团队都被组织成功能团队，他们使用带有三周冲刺的Scrum方法，以达到每三个月就发布一个新版本，其中连续集成管道就用来在几个测试环境中运行功能测试，而这些测试环境就包含了带有真实产品和模拟器的网络拓扑结构。

##### 3.3 调查的参与者

我们对同一公司的软件开发人员进行调查的目的就在于能够将开发人员的认知和具体的客观数据进行比较。该调查于2018年1月进行，我们随机邀请了80位开发者进行参与，其中50人自愿在一周内给出答复（答复率为62.5%），另外有5名参与者表示他们在代码审查（作为开发者和审查者）方面（非常）缺少经验，我们就将这些答复丢弃了，因为他们可能会提供不可靠的答复。由于我们的调查涉及其他项目以及研究的第一部分处于不同的时期，因此在我们收集数据期间，许多参与者不是我们目标项目的开发人员，但是这些项目的参与者使用了Gerrit来实施和加强现代代码审查的实践。尽管在整个软件开发过程中可能存在分歧，但这些项目都是采用图1中的代码审查流程来执行的。

表2提供了45名参与者的详细信息，包括年龄、受教育程度以及各自的经验，我们可以看到，能够表明参与者经验的回答是基于参与者的主观观点自我报道的，其中97.8%的参与者具有多个团队项目的中等到更高等的经验，而93.3%的参与者具有多个不同地点项目的中等到更高等的经验，这表明他们在软件开发系统中是有一定经验的。

#### 4 结论

代码审查是一种重要的静态验证技术，它可以提高软件质量以及促进软件项目中的知识共享。为了确定代码审查实际上成功了的情景，许多研究都调查了不同影响因素和审查结果之间的关系，但是，当开发人员和审查员位置距离较远时，现代代码审查在分布式软件开发环境中的有效性这类的调查却是有限的。

在本文中，我们呈现了混合方法研究的结果，这由两部分组成。在第一部分仓库挖掘中，我们从一个软件项目中提取了大量的代码审查信息，这个软件项目致力于为嵌入式系统开发操作系统，这个系统包括了201名开发人员，这些人分布在4个城市，涉及到21个团队。我们调查了补丁大小（代码行数）、团队数量、位置数量和活跃审查者数量如何影响审查的持续时间、审查者参与度和评论密度（一般和评论者）。我们发现：代码审查的持续时间深受所有调查因素的影响，补丁越大、团队数量、位置数量以及活跃的审查者越多，审查的持续时间就会越长；同样，审查人员的参与在所有案例中都受到负面影响，但这取决于要审查的代码的行数；如果一个相对较小的补丁被其他团队或地点的审查者而非开发者审查的话，审查者的评论密度相对会更高；每位审查者的审查评论密度受到所涉及到的地点数量的正面影响，也受到其他因素的负面影响。

在研究的第二部分，我们进行了一项调查，收集了四个影响因子对代码审查结果（审查持续时间、参与者以及评论总数）的感知影响的数据。我们从软件开发人员那里得到了50份答复，这些开发者都是在分布式软件开发项目中具有现代代码审查经验的人，我们发现：如果影响因子具有更高的值，那么对分析的代码审查的结果具有相似的影响；持续时间和参与者受到负面影响；评论总数受到补丁大小、团队和地点的负面影响，但受到活跃审查者数量的正面影响。

由于在我们的研究中调查了大量的数据，我们无法识别代码审查的特定情况，这有助于我们进行其他分析并进一步解释我们的数据。即使这是可能的，考虑到我们使用过去的数据进行完整的审查，开发人员可能不会记住特定的案例，然而，我们的研究给我们带来了对于未来研究的见解。首先，我们的目标是进行一项涉及到开发人员和管理人员的观察性研究，以验证我们基于当前研究所得出的结论是否成立。其次，我们可以使用代码审查的数据进行进一步的分析，例如，投票的比率（否决、拒绝、批准和中立反馈）、作为开发者和审阅者的贡献数量的影响（总体上和在同一模块或文件中的）以及其他审查者的特征是否是值得调查的问题。

由于补丁大小被证明是一个重要的影响因素，我们还计划在代码审查期间去分析其他形式的复杂性和工作量，假设将要审查的十行代码添加到一个复杂模块比添加到一个简单的模块要花费更多的努力，这就可以分析其他复杂性指标的影响因素，例如类的总量、文件、行数以及总的循环复杂性。

对于模块化系统，一些影响因子来自于模块之间的关系和每个模块的角色。 例如，依赖模块的数量可能影响代码审查的参与或持续时间，并且与实现用户界面的模块相比，关键的基础架构模块可能具有不同的代码审查动态。 因此，我们还计划分析代码审查中模块在架构方面的影响。

最后，我们考虑了许多衡量指标以指出审核的有效性，并旨在调查是否有可能通过组合不同的审核结果来推导出能够反映审核有效性的单一指标。