单位代码_____

学　　号 00000000

分 类 号 00000000

密　　级 不涉密

北京航空航天大學

BEIHANG UNIVERSITY

毕业设计(论文)

GitHub开发人员组成和软件缺陷关系的量化分析（文献综述）

院（系）名称 XXX学院

专业名称 XXXXXXXXXXXXX

学生姓名 XXXX

指导教师 XXXX

2017年12月

本人声明

我声明，本论文及其研究工作是由本人在导师指导下独立完成的，在完成论文时所利用的一切资料均已在参考文献中列出。

作者： XXXX

签字：

时间： 2017年12月

GitHub开发人员组成和软件缺陷关系的量化分析（文献综述）

学　　生：　XXXX
指导教师：　XXXX

摘　　要

保证代码质量是软件开发的一项重要内容。软件工程研究人员已经从诸多角度开展了相关工作，探寻了各种可能的代码质量影响因素。本文调研了可能影响软件质量的三大类要素：代码归属，软件开发指标以及评审行为。同时，本文还调研了当今世界上最大的开源软件开发平台：GitHub的一些基本情况，包括其影响力、软件开发的核心机制等。这些调研有助于我们设计针对GitHub上开源项目开发过程中的各因素对软件项目质量所造成的影响的量化探索实验。日后的研究中，我们会关注那些涉及到根据代码改动及其评审的行为特征来划分的用户组成的影响因素，同时还要控制其他可能对项目质量产生影响的因素。

关键词：软件质量，代码归属，软件开发指标，评审行为，现代代码评审方式，GitHub，开源项目，开发者组成

A Quantitative Analysis of the Relationship Between the Composition of Developers on GitHub and Software Quality (Literature Review)

Author： XXXXXXX

Tutor： XXXXXXXX

Abstract

Ensuring the quality of the code is a very important component of software development. Software engineering researchers have done some researches on various perspectives, exploring diverse kinds of influencial factors of software quality. In this paper, we reviewed three general factors which would influence software quality: code ownership, software development metrics and review activities. At the same time, we surveyed some basic information about the biggest open-source software development platform: GitHub, including its influence and core mechanisms of open-source software development. Those reviews and surveys help us design the quantitative experiment about exploring the impact of quality-influential factors on the quality of GitHub's open-source repositories. In the research stages later, we will put more emphasis on those quality-influential factors which involve the users' composition divided according to the characteristics of code modification and reviewing conduct, at the same time other quality-influential factors will be controlled.

Key words: Software Quality, Code Ownership, Software Development Metrics, Review Activities, Modern Code Review, GitHub, Open-source Repository, Composition of Developers

目录

1 文献综述

1.1 背景及目的

1.1.1 综述模型

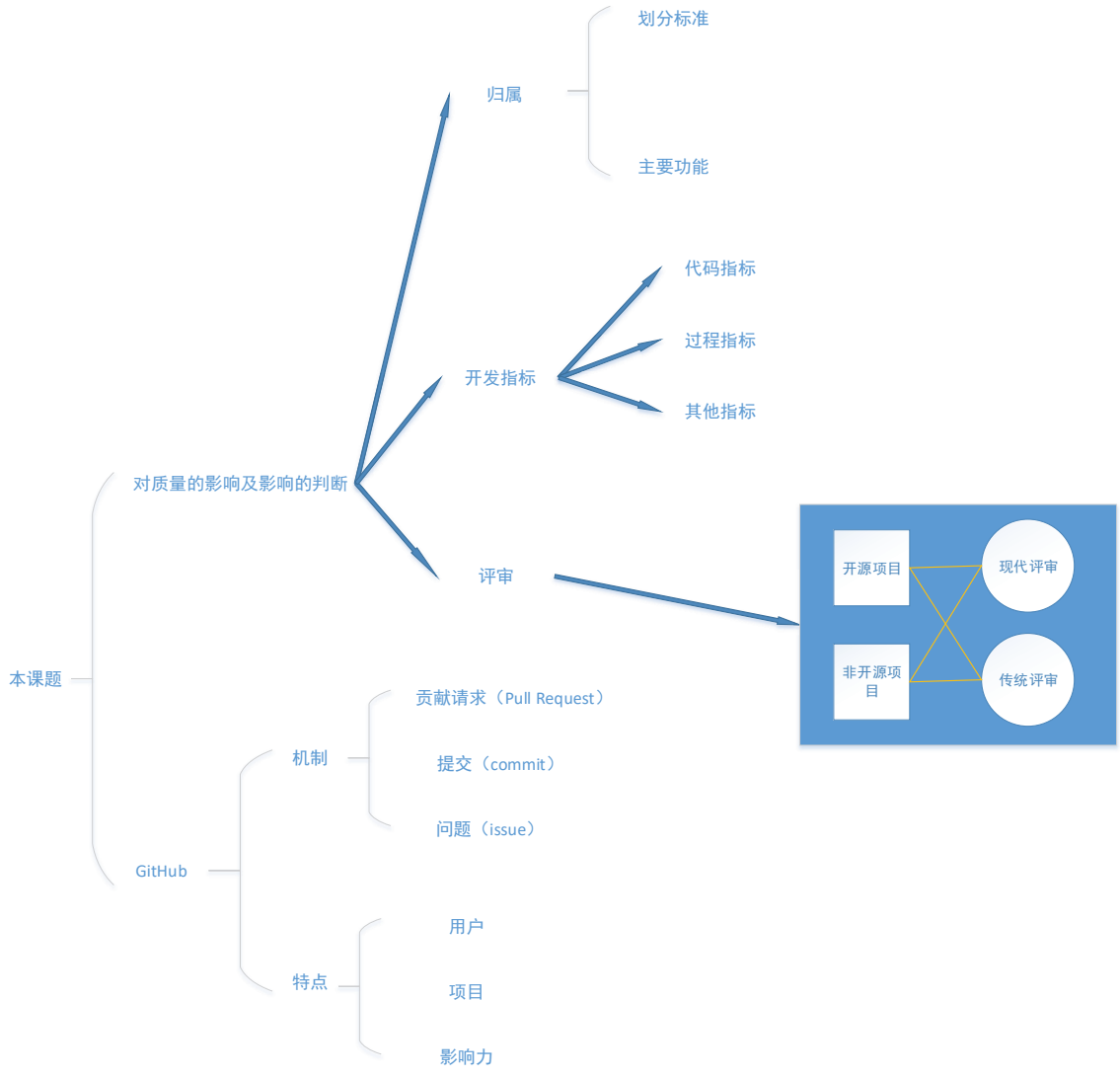

图 1.1: 本次研究的综述模型

1.1.2 代码质量及其影响因素

现有研究分析人员组成对软件质量的影响，主要从三个角度开展，分别是代码归属、代码评审以及其他的项目指标分析。

在一个大型的软件开发团队里，代码归属是非常重要的。某一个开发者在这个软件的某一模块当中的代码归属越强，则这个开发者就对这个模块负有越大的责任[1]。根据Mockus等人的研究，弄清代码归属问题，可以最大程度上找到具有解决某项问题的相关知识和经验的人，以提高合作开发软件的效率[2]。现在的大型软件项目由数量庞大的模块组成。传统的代码归属指标，主要是根据某一用户在某一个软件模块中改动的代码量或是其比例来计算的[3]。在一些过去的研究中，研究人员通过改动某个软件部分来决定哪些开发人员在这个软件部分上具有“专业”的经验[2]，有些研究则通过这样的“经验”来推荐解决某些问题的最“专业”的人员[4]。

涉及到软件项目开发特性的一些指标值也可以用来预测项目的质量。Menzies等人认为代码指标（code metrics）在缺陷预测方面能够起到比较好的作用，这些代码指标可能包括代码行数指标、操作符以及操作数的个数等[5, 6]，以及代码分支、节点的个数等[7]。Matsumoto等人在他们的研究当中发现了一些开发者特征与软件可靠性之间的关系。他们使用了一些开发指标，比如每一个开发者做出的代码改动数（churn）、每一个开发者做出的提交（commitment）数量和每一个项目的开发者的数量等[8]。Moser等人则发现了一些软件开发的过程指标（process metrics），例如一个文件被进行了几次bug修改、所有的改动当中加入代码的平均数，代码改动总量（增加的代码数减去删除掉的代码数）等指标与软件质量的关系[9]。

代码评审是成熟的软件开发项目当中重要的一环，它的目的就是评估（evaluating）开发者提交上去的代码贡献（code contribution）的质量[10]。经典的，正式的代码评审机制，例如被使用了超过三十年的同业审查（peer review）方法[11]，这些方法在实行的过程中往往有着严格的规定，例如线下交流，评审清单等，以确保评审的品质[12]。这种审查方式严谨并且很大程度上能够保证评审的质量。相比之下，现代代码评审（Modern code Review）[11]机制就采取非常灵活的组织形式，可以满足远程合作的需求，使得评审行为不再收到地域的限制。相当一部分软件业巨头，包括Google, Cisco, Microsoft在开发他们的产品的时候采用这种评审方式；于此同时也有相当多的公共项目审查工具涌现而出，比如CodeCollaborator, ReviewBoard, Gerrit, Crucible等[11]。研究人员也逐渐在将目光聚集到在软件开发过程中广泛采用的现代代码评审机制上。无论是传统的正式评审，亦或是现代代码评审方式，其主要的动机都是为了改善代码的品质。同时，现代代码评审还能够很方便地起到知识传递，增加团队的了解，思考针对某些问题的替代解决方法[13]。已经有一些研究涉及到代码评审与软件项目的质量之间的关系。其主要研究的评审行为是评论（comment），包括对代码改动进行评价、提问、解答，或是提出改

进意见甚至是不相同的解决方案[14–17]。一些研究表明，评审行为和软件质量之间也存在着关联。虽然代码审核的初衷是截获缺陷，不过在某些情况下缺陷依然可能不会被查出来[10, 12, 14]。所以，参与到评审行为和代码改动的人员是可以影响软件质量的。

1.1.3 GitHub介绍

GitHub[18–21] 是基于Git[22]的开源代码托管、修改控制以及社会编码平台[23]。很好地应用了Git的分支特性。开发者可以复制一份主分支（主要的代码库）中的代码到本地，作为子分支。开发者在子分支做出了代码改之动后，便可以向父分支发起贡献请求(Pull Request[24])，以请求将自己所做的代码改动合并到主分支当中。同时GitHub提供了非常方便的原始数据获取机制[25]，所以学术界有非常多针对GitHub 的数据挖掘研究[26]，比如针对用户之间的合作行为的研究，也有关注于获取项目于项目、用户与用户之间的关系的特性的研究[18, 23]。

GitHub社区中的任何用户都可以创建自己的子分支并发起贡献请求。由于开发者素质良莠不齐，对项目本身的了解也不尽相同，所以，他们做出的贡献请求若直接合并到主分支中，有可能会给项目带来质量问题，比如引入bug[10]，或者是导致发布后缺陷（post-release defects）[1]。为了保证缺陷尽可能不被引入，GitHub采用代码评审机制。

项目的核心人员以及其他普通用户可以查看这个贡献请求的代码更改并开展评论，这些评论有可能包含对代码改动的评价，提问和解答[27]等。项目的决策者[28]可以结合讨论的结果和自己的判断，对此次贡献请求进行决策，即决定是准许还是拒绝这个贡献请求所做出的代码改动被合并到主分支中。结合用户对于项目的贡献请求，以及项目决策者对贡献请求的管理，可以说，在GitHub项目的开发中，开发者、软件贡献请求的发起者和评审参与者们处在现代代码评审机制的大背景之下，这些人组成了一个非集中制的、为某一个开源软件项目贡献力量的虚拟开发团队[29]。这样一来，GitHub上的开发评审行为就可以和传统的软件开发评审行为进行类比了。

Github是一个典型的基于知识的（knowledge-based）工作环境。这个网站整合了很多关于用户的社会特征（social features），在软件开发项目的任何时间任何地点都有体现[18]。从相关的研究当中可以发现，在GitHub开放的环境之下，软件项目开发的参与者为数众多，参与者的组成以及开发者之间的关系较为复杂[23]，用户对项目所做出的改动和评审这样的贡献行为也颇具多样性。Github具有贡献请求（pull requests），代码提交（commits）以及问题（issue）这些机制[26]，在这么一种软件开发情境下具有不同行为特征的用户对项目质量的影响与其他软件开发环境下的必定有所不同。

1.2 研究现状

对于代码归属对于质量的影响，有一些学者已经有了自己的看法。在一些商用软件的开发背景之下，有一些研究人员发现，如果开发人员对于软件的某个部分有较强的代码归属，那么这样的开发人员对于代码的质量会有积极的贡献，而较弱代码归属的开发者倾向于在这个部分当中引入缺陷[3]。Greiler 等人也在研究中发现，在文件和目录这两个等级当中被查出来的bug数量，与代表着低代码归属的指标有着类似的相关关系[30]。也有一些研究者在开源项目上得到了不完全相同的结论，比如较之软件的一些内在属性，例如模块的大小，代码归属指标未必会对代码缺陷数量有更大的影响效果[31]。

根据McIntosh的说法，软件缺陷已经成为衡量软件质量的一个非常流行的标准了[32]。现代软件开发当中的质量的评估，已经有学者在这方面做出了研究。这些研究都试图使用一些与代码改动者或者软件项目有关的一些指标来预测软件缺陷。Matsumoto根据开发者指标计算得到的研究结果是，软件的某个模块被越多的人动过，那么这个模块就越有可能出现错误[8]。Moser等人则在eclipse项目上做了一些数据统计的工作，发现了一些软件开发的过程指标，相较之与代码指标（code metrics），更能够准确地预测软件缺陷的产生[9]，尽管代码指标的使用的历史更久，使用范围更加广泛[33]。同时，还有一些其他的指标值也被证实于项目质量有所关联，比如，Kononenko等人更关心一些关乎到个人的指标（personal metrics），比如说审查者的工作负担、经验，以及代码审查参与度指标（participation metrics），比如参加了的开发者的数量等，这些指标与代码质量之间的联系[10]；Hassan则考虑到了代码改动在项目当中的分布的复杂性对错误的产生可能造成的影响，这种代码改动复杂性被称为“熵”[34]。

也有一些学者在关于评审对项目质量可能会产生的影响这一课题上做出了研究。Kononenko等人在他们基于mozilla的研究中发现，对代码改动的审查结果表明，54%的代码改动（changes）是会引入bug的[10]。Ghosh等人认为在GitHub这样的开源项目管理网站上，只要评审行为足够充分，那么潜在的问题很大程度上都会被发现[35]，这在定性的角度肯定了评审对软件质量的积极作用。McIntosh等人在研究当中总结出了一些评审过程与代码缺陷之间的一些关系。他们发现了，在现代代码评审这种轻量化的评审机制之下，如果评审的过程不能满足某些要求，那么评审就很容易将缺陷遗漏掉[32]。McIntosh等人还发现，现代代码评审机制的一些特性已经被定性地分析了，但是其对软件造成的影响的定量分析这方面的研究仍相对较少。他们通过对一些热门项目的挖掘，发现代码改动的评审覆盖率低，则容易将更多的缺陷引入到项目中，并且这从经验上确证了一个直觉：评审质量不佳的代码会给大型软件项目的质量带来负面影响[12]。同时，

他们还发现，开发者参与评审与否，对于项目缺陷的数量的影响是具有显著的统计学意义的[32]。Patanamon 等人通过对Qt等项目的研究发现，有缺陷、或是有出现缺陷的隐患的文件，往往没有经过严格的评审；评审不但能够修正一时的缺陷，还能从长远的角度改善代码质量[14]。

将代码改动和评审同时考虑，重点进行评审和质量的关系的研究也比较少。Mcintosh等人发现，在Gerrit平台上有着许多代码改动的软件部分（component）如果没有一项专家指标（代码改动或者是评审），那么这样的代码改动是有缺陷倾向的[32]。Patanamon等人也在Gerrit平台上对Qt，Openstack等项目做了这方面的研究，在他们的研究中，一个用户在一个模块当中被划分进了不同的角色分类，而划分的逻辑基于其评审和改动行为的特征。他们认为做出了主要的评审行为且次要的改动行为的用户如果占据了更大的比例，则这样的模块的质量往往会更好；而做出主要改动和次要评审行为的人员会给项目带来更多发布后缺陷[1]。

以上对于代码改动以及评审的研究，绝大部分不是基于GitHub上开展，有的并非基于开源软件开发环境，有的基于其他开源软件开发平台，故而也就没有涉及到GitHub的最重要的核心机制：贡献请求[26, 36]；而且这里所涉及到的评审行为也仅仅是评论行为，并没有包含决策行为。

然而涉及到GitHub上贡献请求中的代码修改和项目质量这两个方面的研究，一般不对两者之间的关系进行直接的定量分析，比如Vasilescu等人发现，使用持续集成（continuous integration）方法，可以使得更多贡献请求被准许合并，同时还能保证不引入用户反馈的bug[37]。针对贡献请求评审行为，较少的研究着眼于评审行为与代码质量的关系。对于评论任务，更多的研究关注于如何更好地将贡献请求推荐给评论者，从而提升评论的效率、加快贡献请求的评审进程[36, 38, 39]。有些则是直接避免人工评审而直接通过机器学习的手段去寻找潜藏bug的代码改动[40]。有的研究者对评审中的决策行为做出研究。Yu等人认为Github 极大地便利了潜在项目贡献者对项目做出贡献，但是这也造成了决策者疲于应付数量极为庞大的贡献请求审核工作，这样以来就可能造成评审进度的延迟[41]。Jiang等人探索了一条如何将贡献请求推荐给最合适的决策者去决策的道路[28]。Gousios 等人调查了749 个决策者的决策行为之后，定性地研究了以确保代码的质量为目的决策者的行为会展现出什么样的特点，并且得到了一个结论：决策者很难保证他们的项目的质量，并且很难决定哪个贡献请求应当优先被采纳，这说明决策的过程对于决策者而言是一个重大的挑战[19]。从上述针对评论和决策的研究中可以得出，GitHub上的决策行为对质量可能存在影响，但是这些研究没有在定量的层面上探索包括

评论和决策在内的评审行为会对与项目质量带来什么样的影响。

1.3 问题的提出

我们希望能够量化分析GitHub项目开发者的组成会对开源项目的质量造成什么样的影响。由于之前的工作没有考虑到GitHub的贡献请求这一独特开源软件开发机制，或者没有考虑决策这一重要的评审行为。因此接下来的研究，我们需要将这些前人暂时没有考虑到的这些因素一并纳入研究范围。我们认为，所有对项目开发过程做出过代码改动、评论和决策行为的参与者，是项目的开发者（developer）。同时，之前的研究当中发现的，可能会影响开源项目质量的指标，包括过程指标，代码指标，个人指标等，应该被作为控制变量加以考虑。

我们的研究不局限于分析人员组成是否会造成影响，更是要分析这样的影响有什么特点，例如造成了好的还是不好的影响，哪种人员组成的影响程度更大，这些结论需要我们建立关系模型来得出。代码质量的评判指标为一个模块是否有发布后缺陷。这些结论可以帮助我们发现有利于减少软件缺陷的人员组成方式，对于GitHub上开源项目的人员组成合理化起到一定的指导作用，以求获得最好的代码质量[42]。github是世界上最大的代码持有网站，并且正在成为互联网上最重要的软件代码来源之一[26]。根据官方数据显示，目前GitHub已经拥有超过340万的注册用户。据统计，截至2014年一月，已经有超过一千零六十万个开源项目（repository）托管在GitHub上[23]，其中不乏一些被广泛使用、意义重大的项目。随着GitHub的影响力与日俱增，这些结论所产生的指导作用可以带来一定的效益。

参考文献

[1] Patanamon Thongtanunam, Shane Mcintosh, Ahmed E Hassan, and Hajimu Iida. Revisiting code ownership and its relationship with software quality in the scope of modern code review[J]. 2016, pages 1039–1050.

[2] Audris Mockus and James D. Herbsleb. Expertise browser: A quantitative approach to identifying expertise[J]. 2008, pages 503–512.

[3] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. Don't touch my code!: examining the effects of ownership on software quality[A]. In *ACM Sigsoft Symposium and the European Conference on Foundations of Software Engineering*[C]. 2011:4-14.

[4] David W. Mcdonald and Mark S. Ackerman. Expertise recommender: a flexible recommendation system and architecture[A]. In *ACM Conference on Computer Supported Cooperative Work*[C]. 2000:231-240.

[5] M. H Halstead. Elements of software science[J]. *Elsevierence*, 1977.

[6] A complexity measure[J]. *Software Engineering IEEE Transactions on*, 1976, se-2(4):308 – 320.

[7] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors[J]. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING SE*, 2007, 33(1):2–13.

[8] Shinsuke Matsumoto, Yasutaka Kamei, Akito Monden, Ken Ichi Matsumoto, and Masahide Nakamura. An analysis of developer metrics for fault prediction[A]. In *International Conference on Predictive MODELS in Software Engineering, Promise 2010, Timisoara, Romania, September*[C]. 2010:1-9.

[9] Raimund Moser. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction[A]. In *ACM/IEEE International Conference on Software Engineering*[C]. 2009:181-190.

[10] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W. Godfrey. Investigating code review quality: Do people and participation matter?[A]. In *IEEE International Conference on Software Maintenance and Evolution*[C]. 2015:111-120.

[11] Peter C. Rigby and Christian Bird. Convergent contemporary software peer review practices[A]. In *Joint Meeting on Foundations of Software Engineering*[C]. 2013:202-212.

[12] Shane Mcintosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. *The impact of code review coverage and code review participation on software quality: a case study of the qt, VTK, and ITK projects*[M]. 2014:192-201.

[13] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review[A]. In *International Conference on Software Engineering*[C]. 2013:712-721.

[14] Patanamon Thongtanunam, Shane Mcintosh, Ahmed E. Hassan, and Hajimu Iida. Investigating code review practices in defective files: An empirical study of the qt system[A]. In *Mining Software Repositories*[C]. 2015:168-179.

[15] Moritz Beller, Alberto Bacchelli, Elmar Juergens, and Elmar Juergens. Modern code reviews in open-source projects: which problems do they fix?[A]. In *Working Conference on Mining Software Repositories*[C]. 2014:202-211.

[16] Yida Tao, Donggyun Han, and Sunghun Kim. Writing acceptable patches: An empirical study of open source project patches[A]. In *IEEE International Conference on Software Maintenance and Evolution*[C]. 2014:271-280.

[17] Jason Tsay, Laura Dabbish, and James Herbsleb. Let's talk about it: evaluating contributions through discussion in github[A]. In *ACM Sigsoft International Symposium on Foundations of Software Engineering*[C]. 2014:144-154.

[18] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github:transparency and collaboration in an open software repository[A]. In *Cscw 12 Computer Supported Cooperative Work, Seattle, Wa, Usa, February*[C]. 2012:1277-1286.

[19] Georgios Gousios, Andy Zaidman, Margaret Anne Storey, and Arie Van Deursen. Work practices and challenges in pull-based development:the integrator's perspective[A]. In *International Conference on Software Engineering*[C]. 2015:358-368.

[20] Casey Casalnuovo, Bogdan Vasilescu, Premkumar Devanbu, and Vladimir Filkov. Developer onboarding in github: the role of prior social links and language experience[A]. In *Joint Meeting on Foundations of Software Engineering*[C]. 2015:817-828.

[21] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in github[A]. In *ICSE*[C]. 2014:356-366.

[22] Jon Loeliger. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*[M]. 2009.

[23] Ferdian Thung, Tegawende F. Bissyande, David Lo, and Lingxiao Jiang. Network structure of social coding in github[A]. In *European Conference on Software Maintenance and Reengineering*[C]. 2013:323-326.

[24] Georgios Gousios, Martin Pinzger, and Arie Van Deursen. An exploratory study of the pull-based software development model[A]. In *International Conference on Software Engineering*[C]. 2014:345-355.

[25] Fragkiskos Chatziasimidis and Ioannis Stamelos. Data collection and analysis of github repositories and users[A]. In *International Conference on Information, Intelligence, Systems and Applications*[C]. 2016:1-6.

[26] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining github[A]. In *Working Conference on Mining Software Repositories*[C]. 2014:92-101.

[27] Mohammad Masudur Rahman and Chanchal K Roy. An insight into the pull requests of github[J]. 2014, pages 364–367.

[28] Jing Jiang, Jia Huan He, and Xue Yuan Chen. Coredevrec: Automatic core member recommendation for contribution evaluation[J]. *Journal of Computer Science and Technology*, 2015, 30(5):998–1016.

[29] Oskar Jarczyk, Blazej Gruszka, Szymon Jaroszewicz, Leszek Bukowski, and Adam Wierzbicki. *GitHub Projects. Quality Analysis of Open-Source Software*[M]. 2014:80-94.

[30] Michaela Greiler, Kim Herzig, and Jacek Czerwonka. Code ownership and software quality: A replication study[A]. In *Mining Software Repositories*[C]. 2015:2-12.

[31] Matthieu Foucault, Jean Remy Falleri, and Xavier Blanc. *Code Ownership in Open-Source Software*[M]. ACM, 2014:1-9.

[32] Shane Mcintosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. An empirical study of the impact of modern code review practices on software quality[J]. *Empirical Software Engineering*, 2016, 21(5):2146–2189.

[33] Foyzur Rahman and Premkumar Devanbu. How, and why, process metrics are better[A]. In *International Conference on Software Engineering*[C]. 2013:432-441.

[34] Ahmed E. Hassan. Predicting faults using the complexity of code changes[A]. In *IEEE International Conference on Software Engineering*[C]. 2009:78-88.

[35] Satrajit S. Ghosh, Arno Klein, Brian Avants, and K. Jarrod Millman. Learning from open source software projects to improve scientific review[J]. *Frontiers in Computational Neuroscience*, 2012, 6(4):18.

[36] Yue Yu, Huaimin Wang, Gang Yin, and Charles X. Ling. Reviewer recommender of pull-requests in github[A]. In *IEEE International Conference on Software Maintenance and Evolution*[C]. 2014:609-612.

[37] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github[A]. In *Joint Meeting on Foundations of Software Engineering*[C]. 2015:805-816.

[38] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. Reviewer recommendation for pull-requests in github[J]. *Information & Software Technology*, 2016, 74(C):204–218.

[39] Jing Jiang, Yun Yang, Jiahuan He, Xavier Blanc, and Li Zhang. Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development[J]. *Information & Software Technology*, 2017, 84(C):48–62.

[40] Mikolaj Fejzer, Michal Wojtyna, Marta Burzanska, Piotr Wisniewski, and Krzysztof Stencel. Supporting code review by automatic detection of potentially buggy changes[J]. *Communications in Computer & Information Science*, 2015, 521:473–482.

[41] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. Wait for it: Determinants of pull request evaluation latency on github[A]. In *Mining Software Repositories*[C]. 2015:367-371.

[42] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. Review participation in modern code review: An empirical study of the android, qt, and openstack projects[J]. *Empirical Software Engineering*, 2016, pages 1–50.