prAMD:一种 GitHub pull-request 协作机制自动设计方法

郑 阳 郑丽伟 牟永敏
(北京信息科技大学计算机学院 北京 100101)
(北京信息科技大学软件工程研究中心 北京 100101)

摘 要 社会协作编码的开放性使得软件在开发过程中可以汇聚集体智慧。pull-request 作为 GitHub 中开发人员贡献代码的主要方式,若开发人员之间不能有效地沟通协作,整个开发过程就会变得迟缓低效。因此为了提高协作效率,将 pull 开发模型下开发人员之间的协作过程建模为一种 Agent 协作模型。结合实际开发过程中开发人员扮演的不同角色,以及各个角色具有的相关特征,基于自动机制设计方法计算得到满足优化目标的协作机制,一定程度上可以更好地促进开发人员协作,提高开发效率。

关键词 GitHub Pull-request 自动机制设计

中图分类号 TP311 **文献标志码** A DOI:10.3969/j.issn.1000-386x.2024.09.002

PRAMD: A GITHUB PULL-REQUEST COLLABORATION MECHANISM AUTOMATIC DESIGN METHOD

Zheng Yang Zheng Liwei Mu Yongmin
(*School of Computer Science, Beijing Information Science and Technology University, Beijing 100101, China*)
(*Software Engineering Research Center, Beijing Information Science and Technology University, Beijing 100101, China*)

Abstract The openness of social collaborative coding makes it possible to gather collective wisdom in the process of software development. Pull-request is the main way for developers to contribute code in GitHub. If developers cannot communicate and cooperate effectively, the whole development process will become slow and inefficient. In order to improve the efficiency of collaboration, the collaboration process between developers under the pull development model was modeled as an Agent collaboration model. Combined with the different roles played by developers in the actual development process and the relevant characteristics of each role, the collaboration mechanism that met the optimization goal was calculated based on the automatic mechanism design method, which could better promote the collaboration of developers and improve the development efficiency to a certain extent.

Keywords GitHub Pull-request Automatic mechanism design

0 引 言

社会编码是在协作和分布式环境中进行的,外部开发人员可以通过编码、测试等其他的一些软件工程活动参与软件项目的开发,这就使得核心的开发团队能够汇聚外部开发人员的智慧。由于社会编码文化的发展,GitHub 作为最受欢迎的平台之一已经聚集了大量优秀的开发人员,而且平台所吸引的用户数量仍然在不断上升。平台中蕴含的各种特性,一直都受到研究人员的关注。

为了提高协作效率,推荐合适的开发人员参与协作或者推荐适宜的项目给开发人员。Jiang 等[1]提出一种利用时间衰减关系和文件相似度来预测集成者的方法改善了集成过程。杨程等[2]对流行度、社交关联度等多因素量化分析,向开发人员推荐他们可能感兴趣的项目。pull-request 审查过程是一个典型的众包工作,为了缩短一个新的请求提交给问题追踪器到评论

收稿日期:2021-03-29。国家自然科学基金项目(61402043)。**郑阳**,硕士生,主研领域:多 Agent 理论与技术,软件工程。**郑丽伟**,副教授。**牟永敏**,教授。

者开始讨论的时间,对该请求推荐评论者从而降低了审阅延迟[3-6]。

探索了影响协作背后的因素,为推荐工作提供了更好的依据。Soares 等[7]通过挖掘关联规则,发现评论者比较喜欢分析一些缺乏经验的申请人提交的请求等现象。Jiang 等[8]通过比较不同属性,发现评论者的主观能动性是对评论者预测的重要因素,可以用于改善评论者的推荐。pull-request 管理是最重要的项目活动,发现了影响评估延迟以及有助于更快合并的因素,为协作中的开发人员提供了有价值的信息[9-10];Li 等[11]提出一种结合结构洞理论、生存分析、自我为中心的社交网络新方法对最终修订结果进行预测。Hu 等[12]发现越少的多评审行为意味着评论者对其越高的关注,处理请求的延迟就会变得较少。杨波等[13]证明了软件开发过程中一些影响因素之间存在一定的相关性,可以对开发人员提出一些建议,更好地促进软件开发过程。

除此之外,着眼于整个软件开发的协作过程,做了如下工作。Rodriguez 等[14]引入了情绪分析技术,识别和监控开发人员所编写文本中潜在的情绪,从而可以与开发人员及时沟通。Hu 等[15]就增值服务而言,构建社交网络有效地识别影响力高的个体。Vasilescu 等[16]通过用户调查从而了解 GitHub 中的开发人员如何感知个人属性以及团队的协作等,对开发人员的行为分析做了新的补充。Ehls[17]探索了项目中开发人员流失的原因,发现了项目失败的模式,在软件开发过程中可以尽量避免这些模式。

从上述对 GitHub 平台中蕴含的特性进行的相关研究,可以发现研究者通过从不同的角度分析得出的相关结论,对于促进开发人员之间的协作、提高软件开发过程中的效率,具有一定的指导意义。为了进一步地促进协作提高开发效率,本文基于自动机制设计方法对不同角色的开发人员协作过程进行建模,最终得到满足目标函数的机制。图 1 给出 pull 开发模型中工作流程的总体概述。

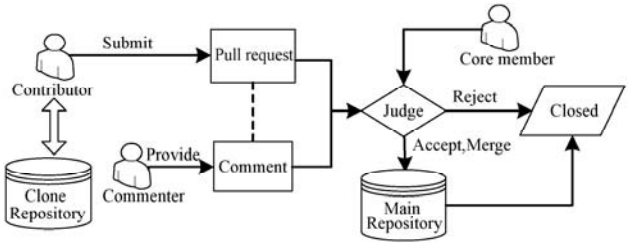

图 1 GitHub 开发人员协作流程

在 GitHub 平台上,绝大多数的项目都会使用 pull 开发模型,开发人员能够在不经允许的情况下对存储库创建分支并对其进行更改,当开发人员想要将修改集成到主开发线时可以提交请求,这种灵活的贡献方式聚集了大量的开发人员。从图 1 可以看出 pull 开发模型下开发人员主要分为三种角色:贡献者,评论者,核心成员。贡献者即请求者,通过提交 pull-request 把自己对项目的更改提交到主存储库中。评论者对贡献者提交的请求进行审核,通过评论的方式提出自己的建议,核心成员即集成者,考虑评论者的意见,对该请求做出最终的决策。

不同角色的开发人员在 pull 开发模型下相互协作,其中一个典型的工作流[13]包括以下几个步骤:

1)讨论问题:就需要添加一个新的功能或者需要改进的功能而言,许多开发人员一起进行讨论,最终决定要添加或者改进的功能。

2)指定事务:经过讨论之后,需要为确定添加或者改进的功能专门指定一个事务即创建一个问题。

3)执行事务:在该事务上创建一个功能分支来实现或者改进功能。

4)审查事务:功能实现或者改进完成后,会通过 pull-request 进行代码审查。

5)问题解决后迭代:pull-request 代码审查过程可以一直持续,直到请求合并到主开发线,问题被解决。

GitHub 平台中受开发人员喜爱的项目会收到越来越多的请求,这会使得 pull 开发模型下开发人员有效的协作变得更为重要。鉴于该模型下集成者在对请求者提交的请求作出决定之前会考虑评论者意见的协作方式,结合他们本身各自具有的特征,本文提出一种 GitHub pull-request 协作机制自动生成方法简称为 prAMD,该方法通过把协作中的开发人员扩展为 Agent 群体,对协作下不同角色的开发人员具有的特征进行类型划分,不同角色的开发人员对应不同的产出,类型和产出不同对应的效用函数也不一样。通过在给定的约束条件下,得出满足约束的机制,选出目标函数效用最大的机制,从而提高协作效率。

1 基本概念

在基于 pull 开发模型中,扮演着不同角色的开发人员扩展为具有不同特征的 Agent 群体,每一种角色的 Agent 群体都有自己的目标。在开发模型中的贡献者都希望自己的请求能够被快速地处理和接受。而每一个评论者也希望自己的评论能够被引用,从而成为关键的评论者。对于项目的集成者来说希望能够快速地处理这些请求,尤其是那些能够促进项目发展的请求。所以不同角色的自利群体之间的有效协作对于整个软件开发过程具有重要的作用。

自动机制设计是通过设置约束条件,所有 Agent 行为都满足给定的约束,如不能通过某种方式虚假地报告自己的类型产生不期望的结果,从而得出机制的过程。在自动机制设计中,Agent 类型集以及产出集给定的条件下,机制定义就是 Agent 类型集合到产出的一种映射关系。具体的自动机制设计过程即通过把自动机制设计建模为计算优化问题,按照所有参与协作的 Agent 具有的特性对 Agent 类型进行划分,Agent 间的协作所产生的可能的结果作为产出,使用实数来刻画 Agent 类型到产出之间的效用关系,通过目标函数得出的 Agent 类型到具体产出之间的映射就是我们所期望的合适的机制[18-19]。

因此,结合自动机制设计把协作中不同角色的开发人员建模为 Agent 群体寻找有助于协作的合适机制。为了保证 Agent 群体形成有效的协作,参考自动机制设计理论,一般我们假设所有 Agent 应满足下列约束:个体理性约束保证了每一个参与协作的 Agent 都能获得一定的收益;激励合作约束确保了参与协作的 Agent 不会提供虚假的类型报告。下面给出具体的约束定义。

1.1　个体理性约束

个体理性约束是指对于参加协作的每一个 Agent 而言,它所获得的效用不低于不参与协作的效用,否则 Agent 就不会参与协作,这种情况下对于参与协作的 Agent 来说就是个体理性的。基于 Agent 对自己类型以及其他参与者类型的了解,根据 pull 开发模型的实际参与者的协作过程出发,我们定义两类个体理性约束 ex interim 和 ex post 约束。ex interim 约束是指当 Agent 不考虑其他参与者的类型仅仅知道自己的类型,Agent 总是会参与协作的。ex post 约束是指 Agent 知道包括自己在内的所有参与者的类型,Agent 总是会参与协作。具体的定义如下:

定义 1　确定性机制。一个 Agent 可以获得本身所创造的所有收益的确定性机制由一个结果选择函数组成:$o:\Theta_1\times\Theta_2\times\cdots\times\Theta_i\times\cdots\times\Theta_N\rightarrow O$。

定义 2　ex interim。确定机制若个体理性约束类型是 ex interim,即对于任意一个 Agent i,以及 Agent i 类型集 $\theta_i\in\Theta_i$ 中任意一个类型满足下面的定义,其中 u_i 表示 Agent 的效用值:

$$\mathrm{E}_{(\theta_1,\theta_2,\cdots,\theta_{i-1},\theta_{i+1},\cdots,\theta_n)\mid\theta_i}[u_i(\theta_i,o(\theta_1,\theta_2,\cdots,,\theta_n))]\geqslant 0$$

定义 3　ex post。确定机制的个体理性约束类型若是 ex post,意味着对于任意一个 Agent i,以及类型向量 $(\theta_1,\theta_2,\cdots,\theta_n)\in\Theta_1\times\Theta_2\times\cdots\times\Theta_N$ 满足下面的定义:

$$u_i(\theta_i,o(\theta_1,\theta_2,\cdots,\theta_n))\geqslant 0$$

1.2　激励合作约束

为了使 Agent 群体永远不应该有错误报告其类型的动机避免影响最终的决策判断,引入激励合作约束。根据 Agent 对其他协作中的参与者类型的了解,有两种类型的约束。优势策略是指如果一个 Agent 不考虑其他 Agents 报告的类型,尽管其他 Agents 的类型已经知道,真实地报告自己的类型总是最优的。贝叶斯-纳什均衡是指 Agent 对其他参与者类型不了解的情况下,任何一个 Agent 都不能通过转换到其他的类型变得更好。具体的定义如下:

定义 4　优势策略。对于任意一个 Agent i 来说,存在的任意类型组合的向量 $(\theta_1,\theta_2,\cdots,\theta_i,\cdots,\theta_n)\in\Theta_1\times\Theta_2\times\cdots\times\Theta_i\times\cdots\times\Theta_N$,以及 Agent i 可以转换的其他类型 $\hat{\theta}_i\in\Theta_i$ 应满足定义:$u_i(\theta_i,o(\theta_1,\theta_2,\cdots,\theta_i,\cdots,\theta_n))\geqslant u_i(\theta_i,o(\theta_1,\theta_2,\cdots,\hat{\theta}_i,\cdots,\theta_n))$

定义 5　贝叶斯-纳什均衡。如果贝叶斯-纳什均衡对于任意一个 Agent i,任意 Agent i 所属类型 $\theta_i\in\Theta_i$,以及 Agent i 可以转换的其他类型 $\hat{\theta}_i\in\Theta_i$,一个确定机制下满足如下的定义:

$$\mathrm{E}_{(\theta_1,\theta_2,\cdots,\theta_{i-1},\theta_{i+1},\cdots,\theta_n)\mid\theta_i}[u_i(\theta_i,o(\theta_1,\theta_2,\cdots,\theta_i,\cdots,\theta_n))]\geqslant \mathrm{E}_{(\theta_1,\theta_2,\cdots,\theta_{i-1},\theta_{i+1},\cdots,\theta_n)\mid\theta_i}[u_i(\theta_i,o(\theta_1,\theta_2,\cdots,\hat{\theta}_i,\cdots,\theta_n))]$$

2　pull-request 协作中的自动机制设计

GitHub 中基于 pull 开发模型下起着不同作用的开发人员包括请求者、评论者和集成者三种角色,三者相互协作实现开发过程的不断迭代。请求者把对代码的更改通过创建一个新的 pull-request 提交到代码评审平台,评论者会留下对该请求的意见,集成者会考虑评论者的建议对请求做出最终的决策,整个协作过程都得益于群体的智慧。下面根据协作中开发人员角色的不同以及不同角色具有的某种特性,结合自动机制设计,从三者的角度出发给出具体的类型划分和产出定义以及相应的效用函数。

2.1　请求者

首先从请求者的角度出发,考虑到积极性高的请求者,希望自己提交的请求能够被接受的愿望很大,即如果请求者表现出强烈的期望,当评论者对他所提交的请求提出修改意见时,请求者能够快速地对修改意见做出回应,继续申请提交。而对于积极性低的请求者来说自己请求能否被接受不太关心,也就意味着即便评论者给出了修改意见,该请求者立即做出回应的可能性也很低,对于整个协作而言可能需要花费很长时间用于等待回应。请求者积极程度 p_{active} 定义如下:

$$p_{\text{active}}=\frac{N_{\text{p\_accept}}}{N_{\text{p}}+1} \tag{1}$$

式中:N_{p}表示请求者在项目中所提交请求的总个数;$N_{\text{p_accept}}$表示请求者在总的提交请求中被接受的请求所占的个数。一个请求者被接受请求的个数与提出请求的总个数加 1 之比作为积极性的度量。在分母上加 1 可以防止那些提交很少请求的人获得很高的积极性,如果没有额外的加 1 操作,那些只提交过一次请求并且被接受的请求者将会获得与提交过 50 次请求且均被接受的请求者一样的概率。实际上后面的请求者应该具有更高的积极性。请求者类型集合记为 Θ_1^{req},想要成为该项目的真正贡献者的期望按照积极程度分为两种类型,概括为积极性高 θ_1^{req}以及积极性低 θ_2^{req},其中 $\theta_1^{\text{req}},\theta_2^{\text{req}}\in\Theta_1^{\text{req}}$。设定阈值为 λ,当请求者所求 p_{active}的值大于或等于 λ 时我们认为请求者的积极性高,否则认为请求者积极性低。可能的产出有四种即 $o_1^{\text{req}},o_2^{\text{req}},o_3^{\text{req}},o_4^{\text{req}}\in O_1^{\text{req}}$。

o_1^{req}:自己成为项目的真正贡献者,即自己的请求被接受,别人的请求被拒绝。

o_2^{req}:自己和别人都成为项目的真正贡献者,即请求都被接受。

o_3^{req}:别人成为项目的真正贡献者,即自己的请求被拒绝,别人的请求被接受。

o_4^{req}:自己和别人都没有成为项目的真正贡献者,即请求都被拒绝。

即便是对积极性低的请求者也仍然希望自己的请求被接受而不是拒绝,而对于整个项目的协作过程而言,积极性高的请求者对项目的开发过程更有帮助,集成者更愿意及时处理这类请求。为了更好地促进协作,在确定偏好这方面变得尤为重要。请求者积极性的类型直接反映了请求者本身对所提出请求的渴望程度,影响着不同产出之间的效用函数,而积极性类型高的请求者对应好的产出产生的效用更大,我们更希望这样的协作方式;为了量化这种偏好,设定请求者效用函数的定义如下:

$$U_1(\theta^{\text{req}},o^{\text{req}})=\bar{p}_{\theta\_\text{active}}+\frac{1\times\Delta N_{p_{\text{accept}}}+0.5\times others(\Delta N_{\text{p\_accept}})}{N+1} \tag{2}$$

式中:$\bar{p}_{\theta_\text{active}}$代表参与问题下的不同类型请求者积极程度的均值;$\Delta N_{\text{p_accept}}$表示在一个问题被解决时自己提交的请求被接受次数的改变;$others(\Delta N_{\text{p_accept}})$表示别人请求被接受次数的改变,$N$ 表示是否存在自己与其他请求者的请求共同被接受的情况。如果存在自己与别人共同解决了一个问题,对请求者本身而言个人产生的效用会变得相应较少。也会出现所有参与该问题解决的请求者都没能提出较好的方案,我们鼓励这些开发人员积极地参与到项目的开发,所以使用他们各自类型下积极性的均值作为效用取值。

2.2 评论者

为了保证软件开发过程中的质量,对一个新请求的评审是分布式软件开发的重要组成部分,所以评论者在协作中保证软件项目的质量起着必不可少的作用。而对于评论者而言在开发协作的过程中,经常会出现一个评论者对多个请求进行评审并在多个请求之间多次切换他们所要审核的请求。多评审的情况使评论者不只关注一个请求,可以提高开发工作效率。但是多评审分散了评论者的关注焦点和时间,如果评论者不能很好地协调对多个请求的关注焦点,多评审的行为不仅没有提高效率,还会增加处理请求的时间开销。因此本文将评论者评论的聚焦程度量化为评论集中度,定义如下:

$$p_{\text{foc}}=\frac{1}{-\sum_{i=1}^{n}p_i\log_2 p_i+1} \tag{3}$$

式中:n 代表某个评论者一周内所评论的总数;p_i表示本周评论的所有请求中不同请求所占的比例。所以通过对评论者集中度的考虑,评论者对一个请求的集中度越高就意味着对请求更多的关注,可以更及时地参与协作,减少协作开发中的时间。评论者类型集合记为 Θ_2^{rev},其中包含两个类型,集中度高 θ_1^{rev}和集中度低 θ_2^{rev}。和请求者类似同样设定一个阈值为 μ,当评论者的 p_{foc}取值大于等于阈值 μ 时,我们认为评论者的集中度高,小于阈值 μ 时认为请求者的集中度低。这里我们考虑对同一个请求下评论者的协作状态,对应的有四种产出即 $o_1^{\text{rev}},o_2^{\text{rev}},o_3^{\text{rev}},o_4^{\text{rev}}\in O_2^{\text{rev}}$。

o_1^{rev}:自己成为关键评论者,即自己的意见被采纳,其他评论者的意见未采纳。

o_2^{rev}:自己和别人共同成为关键评论者,即自己的意见和某些其他评论者的意见同时被采纳。

o_3^{rev}:自己的意见没有被采纳,而其他评论者成为关键评论者。

o_4^{rev}:自己和别人都没能成为关键评论者,即提出的意见均未被采纳。

类似请求者,评论者具体的效用函数定义如下:

$$U_2(\theta^{\text{rev}},o^{\text{rev}})=\bar{p}_{\theta\_foc}+\frac{1\times\Delta N_{\text{rev\_accept}}+0.5\times others(\Delta N_{\text{rev\_accept}})}{N+1} \tag{4}$$

式中:$\bar{p}_{\theta_\text{foc}}$代表参与的评论者不同类型集中度的均值;$\Delta N_{\text{rev_accept}}$表示在当前所要解决的问题下自己的意见被

采纳成为关键评论者次数的改变;$others(\Delta N_{\text{rev_accept}})$表示别人意见被采纳次数的变化;$N$ 表示问题解决之前是否存在其他评论者的意见被采纳的情形,同样地,评论者本身若与其他评论者提出的意见同时被采纳所获得效用比独自成为关键评论者的效用较少。存在评论者对提交的请求都没有提出较好的意见,仍然用他们本身具有的不同类型下集中程度的均值作为效用函数取值。

2.3　集成者

从集成者的角度来说,他们很难对一个新的请求立即做出决定,因为当他们在阅读代码考虑项目的总体状况时,需要考虑很多方面如提交的代码质量以及风格是否符合要求,还需要各种资源如时间、同事的建议、知识等。所以他们在社会网络中的位置对处理请求的影响很重要,因为社会网络中的联系是人们获得信息和资源的重要途径。软件开发过程中相互联系的开发人员通常具有相似的资源和信息,那些与不同开发群体建立联系的集成者往往拥有更丰富的资源和信息。当某个集成者节点连接到与他有所有关系的节点,而这些节点本身没有连接时即与该集成者有联系的开发人员之间没有联系,该集成者就具有了结构上的优势。而这种结构上的优势就意味着他们与不同编码风格等开发人员合作方面更有经验,从而有结构优势的集成者有更多的资源做出决定。这种结构上的优势我们通过效率 E_{f_i}进行量化,越高的效率值就代表着非冗余关系在网络中所占的比例越高,首先给出有限大小 E_{s_i}的定义:

$$E_{s_i} = \sum_j \left[1 - \sum_q p_{iq} m_{jq}\right] \quad q \neq i,j \tag{5}$$

式中:E_{s_i}代表在网格中每一个节点 i 的非冗余关系连接即与节点 i 连接的其他节点没有相互连接;p_{iq}代表连接 i 与节点 q 的分数。

$$p_{iq} = \frac{w_{iq} + w_{qi}}{\sum_j (w_{ij} + w_{ji})} \quad i \neq j \tag{6}$$

式中:w_{iq}表示节点 i 与节点 q 的关注关系,若 w_{iq}为 1,则节点 i 关注节点 q。

E_{s_i}中 m_{jq}是代表节点 j 和 q 之间相对于最大连接强度 j 的相对连接长度:

$$m_{jq} = \frac{w_{jq} + w_{qj}}{\max(w_{jk} + w_{kj})} \quad j \neq k \tag{7}$$

最终通过有限大小与连接总数 N_i即节点度的比值得出效率 E_{f_i}:

$$E_{f_i} = \frac{E_{s_i}}{N_i} \tag{8}$$

我们通过效率进行类型划分概括为效率高 θ_1^{integ}和效率低 θ_2^{integ}两种类型,且 $\theta_1^{\text{integ}}, \theta_2^{\text{integ}} \in \Theta_3^{\text{integ}}$。设置阈值 ω,当 Ef_i的取值大于等于阈值 ω 时,集成者的效率高,而 Ef_i的取值小于阈值 ω 时,集成者的效率低。从一个所发布的问题出发,同一个问题下集成者所处理请求的结果导致的问题的状态给出以下两种产出即 θ_1^{integ}, $\theta_2^{\text{integ}} \in \Theta_3^{\text{integ}}$。

o_1^{integ}:问题解决,即集成者从所提交的请求中找到了合适的请求对其进行了合并,从而问题得到解决。

o_2^{integ}:问题未解决,即所有请求者都没能提出好的解决方案导致没有请求合并。

鉴于效率高的集成者所花费的时间开销较小,具体集成者的效用函数定义如下:

$$U_3(\theta^{\text{integ}}, o^{\text{integ}}) = \overline{E}_{f_i} + \Delta N_{\text{integ\_accept}} \tag{9}$$

式中:$\overline{E}_{f_i}$代表可能对提交请求做出决策的不同集成者类型效率的均值;$\Delta N_{\text{integ_accept}}$表示为了解决当前的问题,集成者所处理的接收请求的数量变化。类似请求者及评论者,即使集成者处理了该问题下的请求,但是并没有找到合适的解决方案,仍然会获得效用值即本身效率类型下所具有的均值,促进集成者积极地处理这些待集成的方案。

2.4　目标函数

根据 pull 开发模型下的请求者、评论者和集成者的协作过程与自动机制设计相结合,通过协作中的开发人员不同角色具有的某些特征,给出了具体的类型定义。通过考虑某一个问题下的贡献者可能具有的结果给出相应的产出。评论者和集成者也是类似,评论者对请求者提交的解决方案都可能会提出相关的建议,集成者是对最终是否找到了合适的请求进行合并从而问题得到解决的可能产出。根据我们在 pull 开发模型下对协作过程的建模,所得到的机制即类型到产出的映射一共有 256 种组合方式。为了使贡献者、评论者和集成者具有的特征综合量化,寻求协作中开发人员最大化效用总和,从而得出我们最希望的一组或多组开发人员的组合方式。考虑到不同类型的参与者在协作中的重要程度不同,因此基于参与者类型设置了相关的权重,权重越大代表协作中所起作用越重要,把目标函数进行线性综合,设置目标函数如下:

$$g = \text{maximize}(\alpha \times U_1 + \beta \times U_2 + \gamma \times U_3) \tag{10}$$

式中:α、β、γ 分别代表协作中开发人员所占权重;U_1、U_2、U_3 代表协作中开发人员各自效用。

2.5　自动机制设计求解

自动机制设计在本问题背景下具象化为,根据类型和产出的具体定义,找到能够满足目标函数的机制

即角色具有的不同类型到结果的对应关系。在 pull-request 协作方式设计中应用这些机制将有利于提高开发人员协作效率,促进软件的发展。

假设不同角色的开发人员在协作中所起到的作用同等重要,令请求者、评论者、集成者在目标函数中开发人员所占的权重为 1,任意能够使目标函数最大化的机制即为本问题的一个解。下面只列出可能的 16 种候选机制如图 2 所示。

$$(\theta_1^{req},\theta_1^{rev},\theta_1^{integ})\rightarrow(o_1^{req},o_1^{rev},o_1^{integ})$$
$$(\theta_2^{req},\theta_1^{rev},\theta_1^{integ})\rightarrow(o_1^{req},o_1^{rev},o_1^{integ})$$
$$(\theta_1^{req},\theta_1^{rev},\theta_1^{integ})\rightarrow(o_1^{req},o_2^{rev},o_1^{integ})$$
$$(\theta_1^{req},\theta_1^{rev},\theta_2^{integ})\rightarrow(o_1^{req},o_1^{rev},o_1^{integ})$$
$$(\theta_1^{req},\theta_1^{rev},\theta_1^{integ})\rightarrow(o_2^{req},o_1^{rev},o_1^{integ})$$
$$(\theta_2^{req},\theta_1^{rev},\theta_1^{integ})\rightarrow(o_1^{req},o_2^{rev},o_1^{integ})$$
$$(\theta_1^{req},\theta_2^{rev},\theta_1^{integ})\rightarrow(o_1^{req},o_1^{rev},o_1^{integ})$$
$$(\theta_2^{req},\theta_1^{rev},\theta_2^{integ})\rightarrow(o_1^{req},o_1^{rev},o_1^{integ})$$
$$(\theta_1^{req},\theta_1^{rev},\theta_2^{integ})\rightarrow(o_1^{req},o_2^{rev},o_1^{integ})$$
$$(\theta_2^{req},\theta_1^{rev},\theta_1^{integ})\rightarrow(o_2^{req},o_1^{rev},o_1^{integ})$$
$$(\theta_1^{req},\theta_1^{rev},\theta_1^{integ})\rightarrow(o_2^{req},o_2^{rev},o_1^{integ})$$
$$(\theta_2^{req},\theta_2^{rev},\theta_1^{integ})\rightarrow(o_1^{req},o_1^{rev},o_1^{integ})$$
$$(\theta_1^{req},\theta_1^{rev},\theta_1^{integ})\rightarrow(o_2^{req},o_3^{rev},o_1^{integ})$$
$$(\theta_1^{req},\theta_1^{rev},\theta_2^{integ})\rightarrow(o_2^{req},o_1^{rev},o_1^{integ})$$
$$(\theta_2^{req},\theta_1^{rev},\theta_2^{integ})\rightarrow(o_1^{req},o_2^{rev},o_1^{integ})$$
$$(\theta_1^{req},\theta_2^{rev},\theta_2^{integ})\rightarrow(o_1^{req},o_1^{rev},o_1^{integ})$$

图 2 候选机制

基于 pull 开发模型下协作中的开发人员包含了三种 Agent 群体,对不同群体具有的某一种特性进行类型划分,并没有考虑一种群体所具有的多种特性。考虑群体多种特性时,类型的划分逐渐增多,对应的产出也会变得复杂,得到的组合的结果会变得更多。在这种情况下遍历无法得到最优解,我们可以采用规划方法进行组合优化求解,根据自动机制设计的相关工作,为了研究机制设计问题的计算复杂度,把自动机制设计描述为一个决策问题,证明了机制设计对于此类问题来说都是 NP-complete[20-21],案例中我们采用优化工具 cplex 对该决策问题进行求解。

3 案例分析

为了清晰地刻画 pull 开发模型下自动机制设计求解过程,通过一个具体的案例分析来展现。对于 GitHub 中的任何一个问题都可以按照类似的过程设计求解,这里不做过多展现。

选取 GitHub 上某 Web 开发项目中存在一个问题即枚举类型转换不能与嵌套条件一起工作,本文以此问题为例对机制的生成过程进行展示和分析,在该问题下存在多个请求者提出了若干个解决方案,而每一个解决方案下又存在多个评论者对此解决方案进行审核并提出相关的意见,最终集成者通常是项目的核心人员考虑评论者的意见对该解决方案做出最终的决定即合并或者拒绝。为了有效地请求集成以更好地解决问题,使用我们提出的 GitHub 协作机制自动生成方法。

具体地,该问题存在 3 个请求者分别为 vivek、zzak、senny,他们都对该问题感兴趣,并为问题的解决提供了各自的方案即都提交了各自的 pull-request。根据构建 GitHub 协作机制自动生成方法选取的请求者具体特征,我们认为积极性高的请求者更有利于提高开发人员的协作效率。为了对请求者的积极性进行量化,求出每一个请求者具体的 p_{active} 值,通过与具体的阈值作比较,大于或等于阈值的请求者积极性高,小于阈值的请求者积极性低,得出请求者具体的类型。本文选取请求者的均值作为阈值,对 3 个请求者 vivek、zzak、senny 求出他们对应的 p_{active} 值分别为 0.7、0.8 和 0.3。因此得出三个请求者的具体类型,vivek 和 zzak 的积极度大于阈值为高积极性即 θ_1^{req},senny 为低积极性即 θ_2^{req}。

请求者对请求被接受的渴望程度与其可能的产出具有不同的效用,请求者所产生的效用越大,越有利于找到合适的解决方案。为了使数据更加统一,以下各个参与者的效用函数进行归一化处理,使用优化工具 cplex 求解过程中请求者 vivek、zzak 和 senny 对应可能产生的效用函数如表 1 所示。

表 1 请求者效用值

请求者	r_outcome			
	1	2	3	4
vivek	0.85	0.82	0.78	0.68
zzak	0.85	0.82	0.78	0.68
senny	0.79	0.74	0.69	0.57

即 vivek、zzak 的效用函数为:

$U_1(\theta_1^{req},o_1^{req})=0.85$ $U_1(\theta_1^{req},o_2^{req})=0.82$

$U_1(\theta_1^{req},o_3^{req})=0.78$ $U_1(\theta_1^{req},o_4^{req})=0.68$

senny 的效用函数为:

$U_1(\theta_2^{req},o_1^{req})=0.79$ $U_1(\theta_2^{req},o_2^{req})=0.74$

$U_1(\theta_2^{req},o_3^{req})=0.69$ $U_1(\theta_2^{req},o_4^{req})=0.57$

每一个请求者所提交的 pull-request 即所提出的解决方案中,都会有相关的评论者对该请求提出自己的意见,同样在构建协作机制自动生成方法的过程中,选取了评论者的具体特征即集中度,集中度高的评论者当请求者与之交流时能够快速地作出回应,一定程

度上可以缩短处理请求的时间,提高合并速率,通过 p_{foc}来量化评论者的集中度。当多个请求者都提交了 pull-request,而每一个 pull-request 下存在多个评论者,需要计算每一个请求者所提交的 pull-request 下所有评论者的集中度,得到每一个评论者 p_{foc}的取值,通过与评论者的阈值即均值进行比较,确定评论者的集中度类型。

评论者多评审行为带来的关注焦点和时间的差异与其同一个问题下评论者协作状态存在的可能产出之间的效用函数,评论者所产生的效用越大,对协作中的评审过程越有利。vivek、zzak 和 senny 所提交 pull-request 下,vivek 和 zzak 都有 4 个评论者参与审核讨论,而 senny 有 3 个评论者。通过计算请求者 vivek 所提交的 pull-request 下参与审核的评论者 dzunk、santib、fylooi、murdho 的 p_{foc}值分别为 0.76、0.5、0.43、0.41。与评论者的阈值比较确定 dzunk 为高集中度即 θ_1^{rev},santib、fylooi 和 murdho 为低集中度即 θ_2^{rev}。可能具有的效用函数如表 2 所示。

表 2　评论者效用值

评论者	r_outcome			
	1	2	3	4
dzunk	0.85	0.82	0.78	0.68
santib	0.81	0.77	0.72	0.61
fylooi	0.81	0.77	0.72	0.61
murdho	0.81	0.77	0.72	0.61

请求者 vivek 下的评论者 dzunk 对应的效用函数:

$U_2(\theta_1^{rev},o_1^{rev})=0.85$　$U_2(\theta_1^{rev},o_2^{rev})=0.82$

$U_2(\theta_1^{rev},o_3^{rev})=0.78$　$U_2(\theta_1^{rev},o_4^{rev})=0.68$

请求者 vivek 下的评论者 santib、fylooi 和 murdho 对应的效用函数:

$U_2(\theta_2^{rev},o_1^{rev})=0.81$　$U_2(\theta_2^{rev},o_2^{rev})=0.77$

$U_2(\theta_2^{rev},o_3^{rev})=0.72$　$U_2(\theta_2^{rev},o_4^{rev})=0.61$

请求者 zzak 和 senny 所对应的评论者类型计算也是类似,请求者 zzak 提交的 pull-request 下通过计算评论者 sunny 和 tvanderpol 的集中度为高类型,评论者 maryammouse 和 mcgregordan 的集中度都为低类型。请求者 senny 所提交的 pull-request 下的评论者只有 drogus 为高集中度类型,rafaelfranca 和 eugeneius 都为低集中度类型。

请求者 zzak 下的评论者 sunny 和 tvanderpol 对应的效用函数:

$U_2(\theta_1^{rev},o_1^{rev})=0.86$　$U_2(\theta_1^{rev},o_2^{rev})=0.82$

$U_2(\theta_1^{rev},o_3^{rev})=0.79$　$U_2(\theta_1^{rev},o_4^{rev})=0.69$

请求者 zzak 下的评论者 maryammouse 和 mcgregordan 对应的效用函数:

$U_2(\theta_2^{rev},o_1^{rev})=0.81$　$U_2(\theta_2^{rev},o_2^{rev})=0.77$

$U_2(\theta_2^{rev},o_3^{rev})=0.73$　$U_2(\theta_2^{rev},o_4^{rev})=0.62$

请求者 senny 下的评论者 drogus 对应的效用函数:

$U_2(\theta_1^{rev},o_1^{rev})=0.86$　$U_2(\theta_1^{rev},o_2^{rev})=0.82$

$U_2(\theta_1^{rev},o_3^{rev})=0.78$　$U_2(\theta_1^{rev},o_4^{rev})=0.69$

请求者 senny 下的评论者 rafaelfranca 和 eugeneius 对应的效用函数:

$U_2(\theta_2^{rev},o_1^{rev})=0.81$　$U_2(\theta_2^{rev},o_2^{rev})=0.76$

$U_2(\theta_2^{rev},o_3^{rev})=0.72$　$U_2(\theta_2^{rev},o_4^{rev})=0.61$

项目的发展和集成者之间有着重要的关系,集成者最终对请求者提交的 pull-request 做出决定,影响着问题的解决和项目的发展。具有更多资源的集成者在处理 pull-request 的过程中有着更多的信息和经验,能够更加快速地处理请求者提出的 pull-request。为了更好地集成,在协作机制自动生成方法中,通过效率 E_{f_i}来量化集成者具有的结构上的优势。一个大项目中往往有很多个集成者进行维护,为了更快地处理 pull-request,我们更希望那些拥有更多资源的集成者来处理这些 pull-request。计算项目中每一个集成者的效率 E_{f_i},与集成者的阈值比较,判断出集成者效率的类型。集成者仍然选取均值作为阈值。

不同集成者在结构上的差异与其所处理问题对应的产出,具有各自的效用取值,集成者所产生的效用越大,越有利于协作的决策过程。通常情况下一个项目往往有很多个集成者,为了计算方便这里仅仅选择该项目中的 5 个集成者分别为集成者 dhh、fxn、jamis、javan 和 kamipo,与之对应的每一个集成者的效率 E_{f_i}分别为 0.54、0.78、0.84、0.67、0.59;与集成者对应的阈值作比较,集成者 fxn 和 jamis 的效率为高类型即 θ_1^{integ},集成者 dhh、javan 和 kamipo 的效率为低类型即 θ_2^{integ},求解中各个集成者可能产生的效用函数值如表 3 所示。

表 3　集成者效用值

集成者	i_outcome	
	1	2
dhh	0.79	0.74
fxn	0.85	0.82
jamis	0.85	0.82
javan	0.79	0.74
kamipo	0.79	0.74

即集成者 fxn 和 jamis 对应的效用函数:

$U_3(\theta_1^{integ},o_1^{integ})=0.85$ $U_3(\theta_1^{integ},o_2^{integ})=0.82$

集成者 dhh,javan 和 kamipo 对应的效用函数:

$U_3(\theta_2^{integ},o_1^{integ})=0.79$ $U_3(\theta_2^{integ},o_2^{integ})=0.74$

通过第三节中请求者、评论者、集成者不同类型到结果的映射关系给出相应的效用函数,在计算目标函数的过程中仍然认为不同角色的开发人员所起的作用是一样的,此模型设置下可能得到的机制为 256 种。本案例下 cplex 中得出能够使目标函数最大化的机制是$(\theta_1^{req},\theta_1^{rev},\theta_1^{integ})\rightarrow(o_1^{req},o_1^{rev},o_1^{integ})$,即为本问题的一个合适解。此问题下我们所期望的开发人员的协作方式如下:请求者 zzak 与其请求下的评论者 sunny 或者 tvanderpol 和集成者 fxn 或者 jamis 协作。我们可以看出得到的合适机制条件下存在多种不同开发人员的协作方式,而这些协作方式对我们的开发过程更加有利。

对一个问题下所有提交 pull-request 的请求者,每一个 pull-request 下对应的所有评论者以及项目中所有集成者都可以量化得出相应的类型。通过 GitHub 协作机制自动生成方法,结合不同角色的开发人员具有的类型到可能结果的映射,使得目标函数最大化选出我们所期望的机制。而该机制下对应的开发人员是我们最期望看到的协作方式。

而在实际的项目中,协作下的开发人员往往是松散的,协作的实际情况也是复杂多样的,这里我们仅仅选取了请求者,评论者以及集成者的某一种特征来展现协作机制自动生成方法的过程。不同角色的开发人员所具有的影响协作效率的特征往往有很多种,对应与在构建协作机制自动生成方法中每一种角色都应该考虑多种类型相应的会有不同的效用函数,对于项目中每一个问题的解决都是如此,由于计算的复杂性在求解的过程中可以采用优化工具来帮助我们得出合适的机制。合适的机制下可能会对应多种开发人员的组合结果,实际情况下由于一个问题中参与协作的人员有限,出现组合爆炸的可能性不大,这里不做过多讨论。

4 结 语

基于 pull 的开发模型被 GitHub 中大部分项目所采用,开发人员可以通过 pull 开发模型在软件开发过程中相互沟通协作。由于相互协作的开发人员往往是无组织的,因此如何使开发人员更好地协作促进开发效率具有重要的意义。影响开发人员协作的因素有很多,不同角色的开发人员具有的某些特征都会对开发效率产生影响,如请求者活跃度较小不能对评论者提出的建议及时作出回应,增大了等待请求者的时间开销;评论者经常通过兴趣爱好寻找请求给出评审意见,寻找请求所花费的时间会带来请求的决策延迟;集成者所拥有的信息资源等对请求的决策都具有重要的影响。因此我们通过引入自动机制设计,把 pull 开发模型下相互协作的开发人员建模成为 Agent 协作模型,通过结合贡献者、评论者、集成者具有的特性对 Agent 类型进行划分,类型到可能产出的映射设定具体的效用函数,在某些约束条件下得出使目标函数最大的机制,在机制的规范下能够得到我们期望的开发人员在 pull 开发模型下协作的方式。

本文的主要贡献点如下:

1)基于 GitHub 平台中在 pull 开发模型下开发人员实际的协作过程建模成为 Agent 群体协作。

2)基于 Agent 群体自动机制设计理论,给出一种 GitHub pull-request 协作机制自动生成方法。

GitHub 中的软件开发协作是一个复杂的过程,协作中的开发人员具有多种特征。将不同角色开发人员的多种特征考虑进来,自动机制中的类型划分和产出情况也将变得更加复杂,在未来的工作中我们将逐步深入研究这种复杂场景下的机制生成方法,以期对实际开发过程改进有所帮助。

参 考 文 献

[1] Jiang J, Lo D, Zheng J, et al. Who should make decision on this pull request? Analyzing time-decaying relationships and file similarities for integrator prediction[J]. Journal of Systems and Software,2019,154:196 - 210.

[2] 杨程,范强,王涛,等. 基于多维特征的开源项目个性化推荐方法[J]. 软件学报,2017,28(6):1357 - 1372.

[3] Yu Y, Wang H, Yin G, et al. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? [J]. Information and Software Technology,2016,74:204 - 218.

[4] Yang C, Zhang X, Zeng L, et al. RevRec: A two-layer reviewer recommendation algorithm in pull-based development model[J]. Journal of Central South University,2018,25:1129 - 1143.

[5] Yu Y, Wang H, Yin G, et al. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration[C]//2014 21st Asia-Pacific Software Engineering Conference,2014:335 - 342.

[6] Yu Y, Wang H, Yin G, et al. Reviewer recommender of pull-requests in GitHub[C]//2014 IEEE International Conference on Software Maintenance and Evolution,2014:609 - 612.

(下转第 28 页)

（2）系统提供的各项指标计算、异常日志等能力，进一步减少运维工作人员的故障排查时间，将故障定位耗时由平均 3 小时降低至 1 小时，大幅度提高故障处置水平。

4 结 语

本文介绍了以探针技术为基础，创新设计并实现了一套服务质量监控系统，并在 OneNET 物联网平台进行了应用，系统能够完成不同协议类型探针的管理，如 MQTT、CoAP、TCP 接入等，并且通过可视化页面实现各项业务监控指标的展示和分析，一定程度上降低运维人员的工作量。本课题在物联网平台服务质量监控方面实现了一定的成果，但本系统暂未对探针采集到的数据进行足够深入的挖掘和分析，接下来将重点实现对探针数据分析，利用卷积神经网络等深度学习方法，实现服务质量的预测分析，实现智能化运维，进一步提升运维业务的效率和质量。

参 考 文 献

[1] 钱志鸿，王义君. 物联网技术与应用研究[J]. 电子学报，2012，40(5)：1023 - 1029.

[2] Li H, Tong J, Weng S, et al. Detecting a business anomaly based on QoS benchmarks of resource-service chains for collaborative tasks in the IoT[J]. IEEE Access, 2019, 7(99): 165509 - 165519.

[3] Mineraud J, Mazhelis O, Su X, et al. A gap analysis of Internet-of-Things platforms[J]. Computer communications, 2016, 89/90(1): 5 - 16.

[4] Eleonora B. The internet of things vision: Key features, applications and open issues[J]. Computer Communications, 2014, 54: 1 - 31.

[5] 邢宁哲，纪雨彤. 基于分布式探针的电力数据通信网综合监测方法[J]. 电力信息与通信技术，2016，14(1)：38 - 43.

[6] 尹隆波. 基于网络探针采集数据，采用粒子群 SVM 实现网络安全态势感知[J]. 电脑知识与技术，2020，16(33)：64 - 65.

[7] 殷泰辉. 网络流量探针的关键技术研究[D]. 长沙：国防科学技术大学，2007.

[8] 马思峻，肖荣，成江伟. Android 应用性能数据采集探针研究[J]. 计算机应用与软件，2017，34(7)：192 - 197.

[9] 中移物联网乔辉：OneNET 已累计设备连接数超过 1.7 亿 将发布 5.0 版本[EB/OL]. [2020 - 07 - 01]. http://finance. sina. com. cn/stock/relnews/hk/2020-07-01/doc-iircuyvk1461030. shtml.

[10] OneNET. 中国移动物联网开放平台[EB/OL]. [2021 - 04 - 29]. https://open. iot. 10086. cn/.

（上接第 16 页）

[7] Soares D, Júnior M, Plastino A, et al. What factors influence the reviewer assignment to pull requests? [J]. Information and Software Technology, 2018, 98: 32 - 43.

[8] Jiang J, Yang Y, He J, et al. Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development[J]. Information and Software Technology, 2017, 84: 48 - 62.

[9] Soares D, Júnior M, Murta L, et al. Acceptance factors of pull requests in open-source projects[C]//30th Annual ACM Symposium on Applied Computing, 2015: 1541 - 1546.

[10] Yu Y, Wang H, Filkov V, et al. Wait for it: Determinants of pull request evaluation latency on GitHub[C]//2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015: 367 - 371.

[11] Li L, Goethals F, Baesens B, et al. Predicting software revision outcomes on GitHub using structural holes theory[J]. Computer Networks, 2016, 114: 114 - 124.

[12] Hu D, Zhang Y, Chang J, et al. Multi-reviewing pull-requests: An exploratory study on GitHub OSS projects[J]. Information and Software Technology, 2019, 115: 1 - 4.

[13] 杨波，于茜，张伟，等. GitHub 开源软件开发过程中影响因素的相关性分析[J]. 软件学报，2017，28(6)：1330 - 1342.

[14] Rodriguez P, Jurado F. Sentiment analysis in monitoring software development processes: an exploratory case study on GitHub's project issues[J]. Journal of Systems and Software, 2015, 104: 82 - 89.

[15] Hu Y, Wang S, Ren Y, et al. User influence analysis for Github developer social networks[J]. Expert Systems with Applications, 2018, 108: 108 - 118.

[16] Vasilescu B, Filkov V, Serebrenik A. Perceptions of diversity on GitHub: A user survey[C]//2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering, 2015: 50 - 56.

[17] Ehls D. Open source project collapse-sources and patterns of failure[C]//Hawaii International Conference on System Sciences, 2017: 5327 - 5336.

[18] 郑丽伟，金芝. 需求驱动的主动网构实体聚合[J]. 软件学报，2008(5)：1083 - 1098.

[19] Braggion E, Gatti N, Lucchetti R, et al. Strong Nash equilibria and mixed strategies[J]. International Journal of Game Theory, 2020, 49: 699 - 710.

[20] Conitzer V, Sandholm T. Complexity of mechanism design[C]//18th Conference on Uncertainty in Artificial Intelligence, 2002: 103 - 110.

[21] Balcan M, Sandholm T, Vitercik E. Sample complexity of automated mechanism design[C]//30th International Conference on Neural Information Processing Systems, 2016: 2091 - 2099.