协作

日期：2025/06

目录

01 科学协作的演变

传统协作方式的局限

效率低下

过去科学家通过邮件进行协作，信件传递时间长，导致信息交流不及时，大大降低了协作效率。例如，在研究过程中，一个问题的反馈可能需要数周才能得到，严重影响研究进度。

存在瓶颈

传统协作方式受限于物理距离和时间，使得协作过程中容易出现沟通不畅的情况，形成协作瓶颈。比如，不同地区的科学家难以实时交流想法和数据，阻碍了研究的推进。

来源不清晰

邮件等传统方式难以清晰记录研究想法和努力的来源，导致在成果归属和知识传承方面存在困难。像在核裂变理论研究中，若采用传统协作，可能无法明确各位科学家的具体贡献。

现代协作工具的兴起

多样化的沟通方式

如今科学家可通过邮件、电话会议等多种方式进行协作，实现了即时沟通。例如，通过电话会议，全球各地的科学家能实时交流研究进展。

网络工具的便利性

网络内容和任务管理工具使协作不受地域和时区限制。如GitHub等平台，能让不同地区的科学家轻松共享和管理研究成果，提高协作效率。

促进科学软件发展

现代协作工具为科学软件的开发和审查提供了便利。例如，通过版本控制和问题跟踪系统，科学家能更好地协作开发和审查软件，提升软件质量。

02 问题跟踪系统概述

问题跟踪系统的作用

协作研究讨论

问题跟踪系统为协作研究讨论提供了集中的交流平台，成员可在系统中针对特定问题展开讨论，如在发现代码bug时，大家能在对应问题下交流解决方案，提高沟通效率。

写论文

在写论文过程中，可利用问题跟踪系统记录论文的修改意见、待办事项等。例如，标记某段落需要补充数据，或某部分需要重新组织语言，确保论文质量。

开发科学软件

对于科学软件开发，问题跟踪系统能帮助开发者管理任务，如记录新功能开发需求、修复已知bug等。它与版本控制系统紧密结合，可清晰记录每次代码修改对应的问题解决情况。

与版本控制系统的联系

问题跟踪系统与版本控制系统紧密相连，问题解决后的代码修改可通过版本控制系统记录，同时版本控制系统中的提交信息可关联到问题跟踪系统中的具体问题，方便追溯和审查。

常见问题跟踪平台

GitHub

- 类型：代码托管平台
- 核心功能：问题标签/负责人分配/代码关联
- 适用场景：开源项目、科学软件开发

Launchpad

- 类型：开源协作平台
- 核心功能：完善问题跟踪/多开发者协作
- 适用场景：纯开源项目、社区驱动开发

Bitbucket

- 类型：集成管理平台
- 核心功能：优先级设置/状态跟踪/版本绑定
- 适用场景：企业级项目、需精细管理的协作

对现代协作的重要性

- 这些平台为现代协作提供了便捷的内容管理和版本控制功能，使团队成员能跨越地域和时区限制，高效地进行协作开发、研究和论文写作等工作

03 问题跟踪系统的工作流程

发现问题的流程

1.检查是否已报告

用户或开发者发现bug后，需先检查当前开放的其他问题，确定该bug是否已被报告。例如在一个开源代码项目中，开发者发现程序运行报错，会先查看问题列表，避免重复报告。

2.创建新问题

若bug未被报告，需创建新问题描述该bug。如GitHub上的“issues”，要明确回答关键问题，如问题描述、复现所需信息、解决目标等。

3.分配任务

问题创建后，可分配给特定开发者。在开源社区，通常由对该代码领域最有经验的开发者负责，也可由感兴趣的开发者自行认领。

4.更新进度

开发者在完成任务请求中，可在问题页面添加评论和更新进度。其他协作者若有疑问或建议，可直接在页面交流。

5.提交拉取请求

任务完成后，提交拉取请求，包含新代码。新代码会经过多平台测试和质量检查，如在科学软件项目中，需确保代码准确可靠。

6.解决问题

新代码通过审核后，问题被解决或关闭，通常由开启者、项目负责人或解决者操作。

新功能需求的流程

1. 提出增强提案

在开源项目中，有新功能需求时，常需提出“增强提案”来启动流程。例如某个开源软件要增加新算法，需先提交提案说明需求。

2. 创建问题

与发现bug类似，需创建新问题描述功能需求，明确功能内容、实现目标等信息。

3. 分配任务与开发

将任务分配给合适的开发者，开发者开始进行功能开发，过程中在问题页面更新进度。

4. 提交拉取请求与审核

功能开发完成后，提交拉取请求，新代码经过审核、测试等环节，确保功能质量。

5. 功能上线

审核通过后，新功能正式集成到项目中，问题关闭。

04 创建问题

问题的核心要素

需求描述

需清晰描述错误或功能请求，例如在代码中发现的具体异常情况，像程序运行时出现的报错信息等。

必要性说明

提供足够信息以重现错误或需求，如给出特定的输入数据、运行环境等，帮助他人理解为何需要解决该问题。

解决路径提议

提出解决问题的前进方向，例如对于代码问题，可建议采用某种算法或修改某个模块来解决。

明确结束目标

设定清晰的目标，如修复特定的错误、实现某个功能等，让大家清楚问题解决的标志。

问题的元数据

标签的作用

标签可用于对问题进行分类，如按重要性分为关键、高优先级等，按类型分为错误、功能请求等，方便管理和查找

用户提醒功能

通过用户提醒（如GitHub的用户ping），能及时通知相关人员参与问题讨论和解决，提高协作效率。

与代码库的交叉链接

可将问题与代码库进行关联，方便查看问题涉及的具体代码部分，如在GitHub上能直接定位到相关代码。

提交钩子的意义

提交钩子可在代码提交时触发特定操作，如自动运行测试等，有助于保证代码质量。

问题的分配与里程碑

分配问题

分配特定开发者

为避免重复工作，问题可分配给特定开发者。在开源领域，问题创建后开发者会进行讨论，通常由对代码该领域最有专业知识的开发者处理。

未分配问题处理

未分配的问题表示无人认领，有兴趣解决的开发者可自信地自行分配。

里程碑

里程碑的概念

在GitHub中，里程碑是将问题分组，定义更广泛目标的方式，且有截止日期。

项目目标跟踪

可作为项目目标跟踪机制，如在研究组组织中，能清晰规划和跟踪研究进度。

代码发布管理

通过捆绑实现特定功能集所需的所有问题，里程碑可用于定义代码发布剩余的必要工作，适用于Grant - driven研究。

问题的讨论

GitHub讨论方式

在GitHub上讨论问题，只需在与问题关联的评论框中输入新评论。例如开发者发现代码问题后，可在此处开启讨论。

讨论内容范畴

这里适合询问和解答关于问题的澄清性疑问、分享和探讨解决问题的思路、请求和提供问题处理过程的更新

问题页面讨论优势

在问题页面直接讨论能更好地保留上下文、保证透明度和明确来源信息，比通过邮件讨论更具优势。

邮件讨论适用场景

非常开放性的讨论通常更适合邮件形式，因为问题最终是要关闭的，邮件可进行更自由的交流。

05 拉取请求

提交拉取请求

分叉仓库

以Lise Meitner为例，若要对项目进行修改，需先分叉主仓库。

克隆本地

Lise Meitner使用 `git clone git@github.com:lisemeitner/uranium_expmt` 命令，将分叉后的仓库克隆到本地。

创建分支

为了不影响主代码，Lise Meitner创建了名为“newtheory”的分支，使用 `git checkout -b newtheory` 命令，在该分支上进行新功能开发或问题修复。

编辑文件

在newtheory分支上，Lise Meitner对文本文件进行编辑，删除旧理论添加新的裂变理论内容来解决项目中的问题。

提交更改

编辑完成后，Lise Meitner使用 `git commit -am 'edits the old theory and replaces it with the new theory.'` 命令提交更改，记录此次修改的信息。

推送分支

最后，Lise Meitner使用 `git push origin newtheory` 命令将本地分支推送到GitHub上的分叉仓库，以便后续发起拉取请求。

审查与合并拉取请求

审查拉取请求

1.是否实现目标
2.是否引入新bug
3.是否包含足够测试
4.是否遵循风格指南
5.是否通过现有测试
6.是否通过新测试
7.是否通过多平台测试

界面按钮合并

在GitHub上，对于无冲突的拉取请求，可直接在拉取请求界面点击绿色按钮进行合并，操作简单便捷。

命令行操作合并

开发者也可通过命令行进行合并操作，使用 `git remote` 、 `git fetch` 、 `git merge` 和 `git push` 等命令组合。具体操作可回顾第16章内容。

THE END

谢谢！