软件需求工程结题报告

庄毅非 何迪 李予谦 应凌凯 刘奕骁

2022 年 12 月 22 日

1 引言

1.1 编写目的

本文档的编写旨在总结此次开源项目分析平台的开发工作, 对整个项目的开发过程、目的、产品、意义进行评价, 对人员的分配、工作情况进行分析总结, 以提升团队的项目开发能力, 促使团队成长。根据老师的要求, final report 应该是对 final presentation 的文字形式的补充, 而我们的 final presentation 是以时间轴的顺序, 按照各个 milestone 的推进进行展开的, 所以以下将以文字的形式, **按照各个 milestone 的顺序**, 详细叙述项目的变迁。

1.2 软件系统名称

开源软件分析平台

1.3 相关人员和项目支持

1.3.1 任务提出者

浙江大学软件需求工程任课老师万志远, 以及助教王懿丰学长。

1.3.2 开发者

2022 浙江大学软件需求工程小组 fiSSure

1.3.3 用户

网站游客、管理员

1.3.4 读者对象

- 项目经理
- 未来需要对项目进行迭代开发的人员
- 用户

2 Milestone 1: Team Workflow & Management

我们在组队和分组之后，就讨论并统一了项目的开发和管理方式。这也在 milestone 1: team workflow and management 中体现了出来。具体来说，主要是以下三个方面。

1. 在组会方面，我们小组会在每周二的晚上 9:30 进行会议，会议一般是在在钉钉上进行的线上会议，但是有时候也会在 32 舍楼下开会。我们的每一个会议都会有一个同学进行会议的主持和纪要的撰写，这个工作一般是由我来负责。会议纪要使用 markdown 撰写，并会转换为 pdf 格式的文件存储在 github 仓库的 process 文件夹中，以下是我们的开会记录和会议纪要的截图。

图 1: 组会记录截图

图 2: 会议纪要截图

2. 我们使用 git 进行代码版本管理，将仓库托管在 github 上，仓库地址是 https://github.com/Yifei-Zhuang/zjusre_2022_fiSSure，不过现在还是 private 的，在课程结束之后我们会将仓库转为 public，在代码版本控制方面，我们建立了两个分支，master 和 develop，大家日常的代码都提交到 dev 分支上，经过代码审核员（主要是我和李予谦）审核之后，确保没问题，再 merge 到 master 分支上作为项目的迭代版本。到目前为止，我们的 dev 分支已经接近有 160 次 commit。

3. 最后，在文档撰写方面，我们使用 latex 编写，在 overleaf 上创建在线文档，从而使得团队的成员能够及时参加项目文档的撰写，跟进项目需求的变更，实现高效的团队管理（事实上，这份文档也是在 overleaf 上编写完成的）。

3 Milestone 2: Vision & scope

在第一个 milestone 之后，很快就是第二个 milestone vision and scope，我们也开始着手进行项目的需求分析工作，由于这时候我们所拥有的参考资料仅仅只有老师 ppt 上的对项目愿景的介绍，老师还

没有对项目具体要实现什么功能进行明确的表述, 所以我们小组再这时候的工作主要就是参考 pingcap 旗下的知名开源项目分析平台 ossinsight，对项目要包含的功能需求和非功能需求进行了粗略的概括, 但是这时候的需求都还有模糊和不明确，也有很多遗漏的部分，这个时期我们对需求的理解还不够深入。以下是当时我们通过观察 ossinsight 所归纳出的一些需求.

3.1 功能需求

1. 分析仓库总体信息，包括 fork、stars 等等
2. 可视化仓库 issue、pull、commit 活动（主要是频率）随时间变化的情况，支持多种图表类型（折线图、饼图等），也应该可以支持按天、月、年进行显示
3. 可视化项目的 top10 贡献者随时间变化的情况
4. 分析项目贡献者来自的国家，地区等，并且支持词云、气泡图等显示方式
5. 能够实现简单的用户管理功能，包括注册，登录等
6. 用户应该能够实现收藏，点赞，评论等
7. 用户应该能够进行仓库之间的对比
8. 可视化项目代码行数随时间变化的情况

3.2 非功能需求

1. 性能需求
   (a) 能够对数据进行高效快速的爬取和存储
   (b) 响应时间在 3s 以内
   (c) 初次导入一个仓库在 10min 以内
   (d) 初次计算一个仓库的信息在 2min 以内
2. 质量需求
   (a) 鲁棒性高
   (b) 具有良好的扩展性
   (c) 可维护性强
3. 接口需求
   (a) 软件：我们选择 linux 发行版（支持 centos、ubuntu）作为操作系统
   (b) 硬件：2Core 4G 轻量应用服务器

4 Milestone 3: SRS

需求的真正明确是在研讨会之后，我们在研讨会之前对要询问的问题进行了分类，将其主要归类为三类，分别是：

1. 确认需求内容，比如显示项目 commit 信息变化，具体来说要显示那些信息；
2. 明确一些要求比较模糊的需求的具体内容，比如分析设计相关的话题，我们要显示哪些分析结果；
3. 确定需求的优先级，比如用户系统的实现优先级高低等。我们当时也列出了用力挂图，方便和老师沟通。
4. 询问是否有遗漏的需求。

我们在 workshop 之后也立即召开了一次小组会议，在会议上，我们对老师要求的需求和其他小组提到的需求一一列出，并按照紧急程度和重要性对需求进行了列表分析，分析结果如下。

编号	名称	紧急程度	重要性	优先级
SE-UC-1	分析不同时期 issue 活动的频率的变化	低	高	高
SE-UC-2	分析不同时期 pull 活动的频率的变化	低	高	高
SE-UC-3	分析不同时期 commit 活动的频率的变化	低	高	高
SE-UC-4	分析不同时期 issuer 活动的频率的变化	低	高	高
SE-UC-5	分析不同时期 puller 活动的频率的变化	低	高	高
SE-UC-6	分析不同时期 commiter 活动的频率的变化	低	高	高
SE-UC-7	分析 issue 从提出到第一次得到 response 的平均时间变化	低	低	低
SE-UC-8	分析 issue 从提出到 closed 所需要的平均时间变化	低	高	高
SE-UC-9	界定出核心贡献者（28 定律），统计数量，按照时间顺序显示在图表上	高	高	高
SE-UC-10	显示核心贡献者背后的公司比例	低	低	低
SE-UC-11	将和设计有关的 issue 和 pull 分离出来	高	高	高
SE-UC-12	使用表格分析上述分理处来的 issue 和 pull，绘制热门话题随时间变化的图表	高	高	高

这样，我们就确定了要开发的需求的优先级，于是我们也按照工作的难易程度，制定了项目开发的各个 milestone，我们使用增量开发模式，在每一个 milestone 截止前都会留下一到两天作为 buffer，同时进行进度的监督，并且在 milestone 结束之后，我们也会按照大家的意见对前一个 milestone 的结果进行迭代，从而保障项目保质保量完成。以下是我们当天晚上召开的组会所讨论得到的各个 milestone 包含的功能需求开发条目以及对应的 ddl 日期。

图 3: milestone timeline

这个阶段我们小组的产出是 milestone 3 软件需求说明书 v1.0，当时我们的 SRS 还是存在一些缺陷的，比如用例标号意义不明，数据流图一些符号使用存在问题等，这些问题也在中期展示中被老师

指了出来，后续我们按照学到的知识，以及老师和同学们的建议，对需求说明书进行了及时的迭代，最终产生了我们上交的 v2.0 版本。

我们首先按照需求绘制了用例图和用例表来描述我们的用例，在用例图中，每一个用例都和我们进行用例分析的条目是一一对应的，之后我们按照老师上课给出的用例挂图的格式对每一个用例进行了详细的说明，包括其用例名，参与者，前置条件，后置条件，触发器，主干过程等等，我们也在上述用例的基础上建立了系统的上下文图，数据流图等。在这之后，我们也对系统要使用的数据进行了规定，定义了数据词典。关于 SRS 的详细内容，包括用例图，数据流图，上下文图等，请查看我们上交的 SRS v2.0 文档，这里因为要控制篇幅，就不再列出。

5 Milestone 4: Design & Coding

在 milestone3 之后，我们就进入了 design and coding 阶段，由于我们是在学长学姐已经有的项目 doubleC 上做二次的迭代开发，并且 doublec 原来的项目结构也比较清晰，所以我们在分析了项目架构以及函数功能等实现相关的内容之后，就着手进行了开发工作。由于我们在项目一开始就规定了项目代码的规范，规定了前后端沟通的方式等，所以开发工作相对比较顺利.

但是也不是一帆风顺的。我们在开发过程中也是对项目进行不断地迭代，才让项目和需求的理想要求越来越接近，拿设计话题分析这部分来说，我们**理想的完整的需求**是程序能够对不同时期的 issue 和 pull request 中将和设计相关的那部分分离出来，然后对每一个时间段内的这些 issue 和 pull 进行分析，提取出讨论的热门话题的具体内容。

在后端的开发中，在 **Version 1** 中，我们是计划在后端使用 NLP 对 issue 的 pull request 进行语义分析，进行特征文本的提取，虽然这样做能够比较精确的实现对不同时期设计需求的提取，但是由于这样做需要一份已经打好标记的数据集来训练模型，并且我们小组的炼丹能力有限，所以我们在讨论之后很快将这个版本否决掉了。后来在 **Version 2** 中，我们小组的刘奕骁同学提出可以使用 elastic search 搜索框架进行模糊搜索，这样虽然结果会比较粗略，但是可以实现对错别字、同词根变形等内容的检索，并且能够考虑数据的权重，并且由于 es 在文本的模糊搜索方面具有比较好的性能，对设计话题的计算也不会导致过长的响应时间，从而保障了用户体验。但是这样做又带来了一个问题，就是我们使用的 mongodb 的数据需要及时同步到 es 中，否则会导致数据的不一致，这里我们使用了一个监听工具监听 mongo 的日志，从而将 mongo 的更新及时同步到了 es 中去，从而完成了后端对设计话题的计算工作。

在后端完成了计算相关模块之后，负责前端的同学很快就开始根据其提供的 api 进行开发，在**前端的 Version 1** 中，一开始我们计划通过折线图或词云图的形式展示将与设计相关的热门话通过排序来分析最热门的话题，但是这样不能够直观反应主要设计话题的占比变化，所以在最后的 **Version 2** 中，我们决定用一种时间轴 + 饼图的模式来直观地展示社区设计相关话题的变化情况。

事实上，我们在开发中遇到了许许多多上述的这种情况，这也让我们明白了，需求模型不是一开始就能够确定了，而是在开发中不断迭代，最后才能确定的。具体的迭代过程，以及我们对需求变更的管理，这里因为篇幅原因，将会在附录中讲述。

以下对我们最终项目的一些亮点进行概括。

1. 对项目设计话题的分析。

   (a) 后端实现：我们使用开源工具 mongo-connector，将 MongoDB 的数据实时同步到 Elastic-Search 中。然后我们为代码、可维护性、鲁棒性、性能、配置、文档、测试等七个热门话题分别指定一批 ElasticSearch 的模糊查询关键词。每当 issue/pr 的 json 内容中出现相同词根或关键词本身，就会被 ElasticSearch 搜索到并计分，我们可以为搜索结果设定一个合适的分数阈值，当得分超过阈值时，认为一个 issue/pr 和一个话题相关性较强，将其统计进该话

题的总数。另外，ElasticSearch 也支持对 String 类型的时间进行筛选，方便了我们对不同时期的各个话题热门程度进行分析。

(b) 前端实现：我们前端中利用 axios 的方式从 http 网页上获得信息，但是由于需要获得的数据较为庞大，得到数据时图表早已渲染好。因此我们利用 useState 对图表做出更新。在图表方面我们使用了时间轴 + 饼状图的方式来展示与设计相关的 pull 与 issue。

2. 对项目核心贡献者的分析

(a) 后端实现：这里我们的实现相对简略，只根据二八原则对 commit 数量进行统计。我们规定，在一段时间内，将贡献者按 commit 数降序排序，从前面开始取，直到 commit 数量和到达这段时间的仓库总 commit 的 80%，定义这些人为这段时间内的核心贡献者。我们的后端在导入一个仓库时，就已经爬取了其所有 commit 信息。这里先将一定时间内的 commit 过滤出来，再根据 commiter 进行统计即可。

(b) 前端实现：

后端将各个年份的核心贡献者打包发送给前端，前端在接受到这些数据之后首先将这些数据重组并统计，给出描述各个年份的核心贡献者数量的曲线图。随后我们在右侧提供了一个可以展示具体年份中核心贡献者名字和对应的 commit 数量的列表，而在这个列表之上，我们提供了一套与之相连的根据年份数量动态生成的按钮组，该按钮组可以供用户选择他们要查看的具体年份。

3. 对项目的 issue、commit 和 pull 等活动，以及每月的 issuer、commiter 和 puller 变动进行可视化

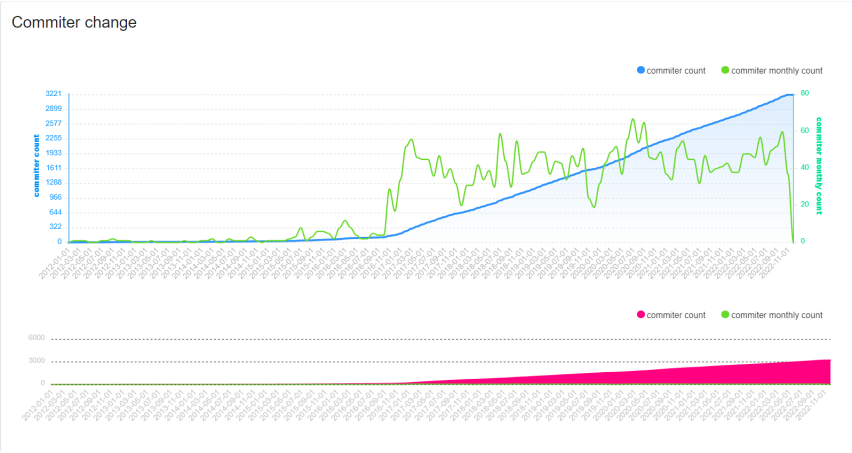

图 4: 项目信息可视化

4. 统计核心贡献者，分析公司分布在前端实现中，我们经过一系列的迭代，认为比起使用词云图来说，使用气泡图更加能够反应公司的占比，通过查阅相关资料，我们最终选择放弃项目中一开始使用的 react-apexchart，而是使用 highchart 进行气泡图的绘制，这样我们的气泡图就能够在拥有和用户鼠标之间的强交互性以及类似动画一样的丝滑展示效果的同时，兼具比较好的渲染性能。以下是我们气泡图的一个截图。

5. 实现仓库对比最终的效果如下。

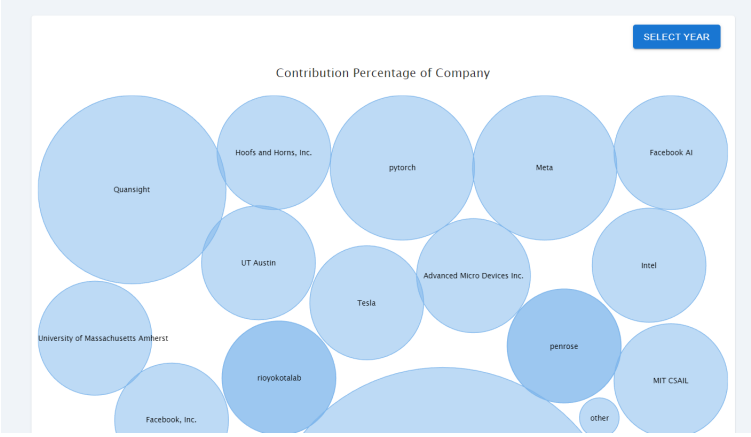
SELECT YEAR
Contribution Percentage of Company
Quansight
Hoofs and Horns, Inc.
pytorch
Meta
Facebook AI
UT Austin
Tesla
Advanced Micro Devices Inc.
Intel
University of Massachusetts Amherst
penrose
riyokotalab
MIT CSAIL
Facebook, Inc.
other

图 5: 公司气泡图

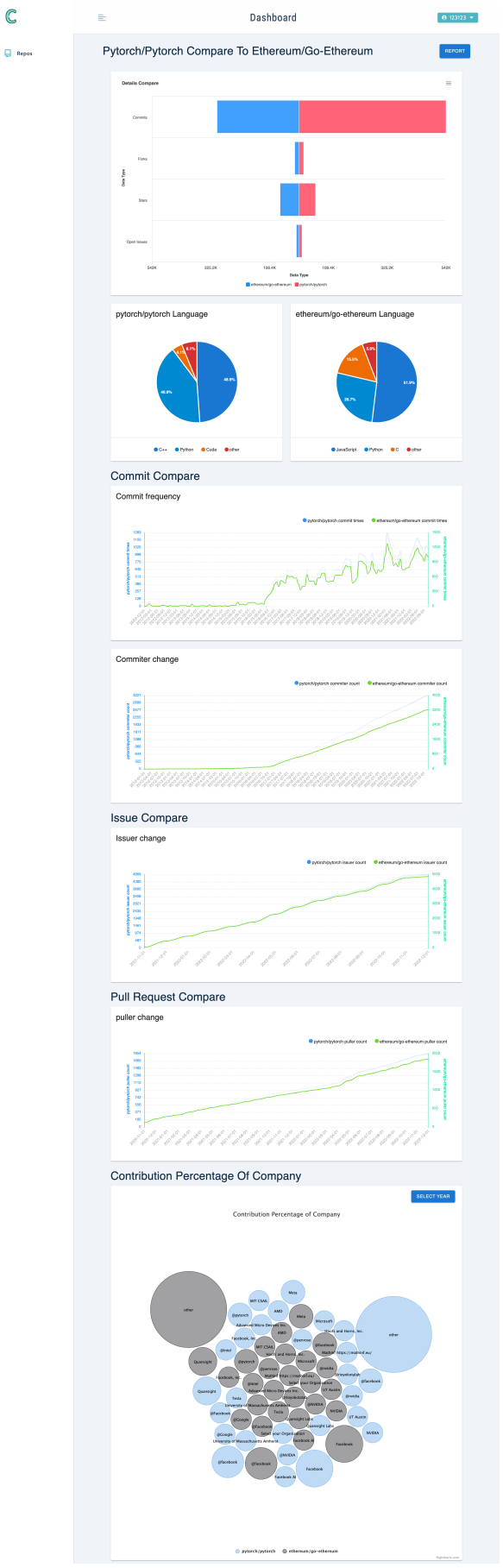
Dashboard
123123
Repos
Pytorch/Pytorch Compare To Ethereum/Go-Ethereum
REPORT
Details Compare
Commits
Forks
Stars
Open Issues
Data Type
pytorch/pytorch Language
ethereum/go-ethereum Language
Commit Compare
Commit frequency
Commiter change
Issue Compare
Issuer change
Pull Request Compare
puller change
Contribution Percentage Of Company
SELECT YEAR
Contribution Percentage of Company

图 6: 仓库对比页鸟瞰图

6 开发心得

在整个项目开发过程中，我们团队也有几点比较深刻的体会。

1. 在团队管理方面，我们要按时召开组会。沟通项目进度，扫清负责不同模块的同学之间的理解偏差和沟通障碍，从而保障项目质量；然后就是要充分了解各个组员的特点，将工作分配给擅长的人去做，可以达到事半功倍的效果；另外就是制定计划要合理，在 milestone 截止之前留下几天作为 buffer，可以保证项目按期完成
2. 在需求管理方面，我们的体会就是需求是需要不断迭代的，我们在开发过程中并不是一下子就能够实现预期需求的效果。然后就是一下建模语言（比如 UML）是很重要的，充分利用他们可以降低项目的沟通成本。

A 附录

A.1 需求迭代过程

我们将根据具体的需求进行分类，来详细讲述我们对于具体的需求是如何进行迭代的。由于一些需求十分的相近，因此我们会将他们放在一起进行阐述。

注：SE-UC 是 Software Engineering - Use Case 的缩写

A.1.1 分析频率变化

原始需求 SE-UC-1: 分析不同时期 issue 活动的频率的变化
SE-UC-2: 分析不同时期 pull 活动的频率的变化
SE-UC-3: 分析不同时期 commit 活动的频率的变化
SE-UC-4: 分析不同时期 issuer 活动的频率的变化
SE-UC-5: 分析不同时期 puller 活动的频率的变化
SE-UC-6: 分析不同时期 commiter 活动的频率的变化

开发前的预想 在实际的开发正式开始前我们已经发现，SE-UC-1 3 已经在原先的代码中完成，因此我们选择基于原先的版本继续开发。我们准备使用曲线图，并提供可以更改展示的时间区间的方法。在设想中是准备使用顶部按钮组/Label 来切换显示的时间格式（年月日）。

最终的效果 最终我们实现的效果依然选择了曲线图。由于使用顶部按钮组/Label 有着无法选择具体时间段的局限性，我们改用了可以直接滑动窗口选择展示区域的底部时间选择栏（图中红色部分）。

A.1.2 统计 issue 时间变化

原始需求 SE-UC-7: 分析 issue 从提出到第一次得到 response 的平均时间变化
SE-UC-8: 分析 issue 从提出到 closed 所需要的平均时间变化

开发前的预想 最初的开发中我们并没有一个明确的图表去合理的展示上述需求，在参考了 OSS insight 之后我们决定采用 BOX graph 来合理的表示每一个 issue 从提出到第一次得到 response 的时间。

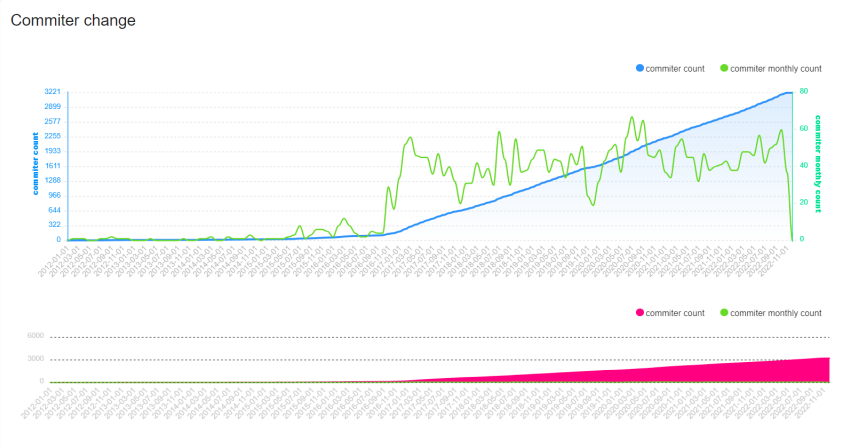

图 7: commiter 变化统计

最终的效果 最终我们依旧选择了 BOX graph 的表示方法，但对于数据处理上上作出了改变，由于计算每一个 issues 所需的时间资源消耗复杂，因此我们同样参考了 OSS insight，展示对一段时间窗内的 issue 被 response 的平均时间。

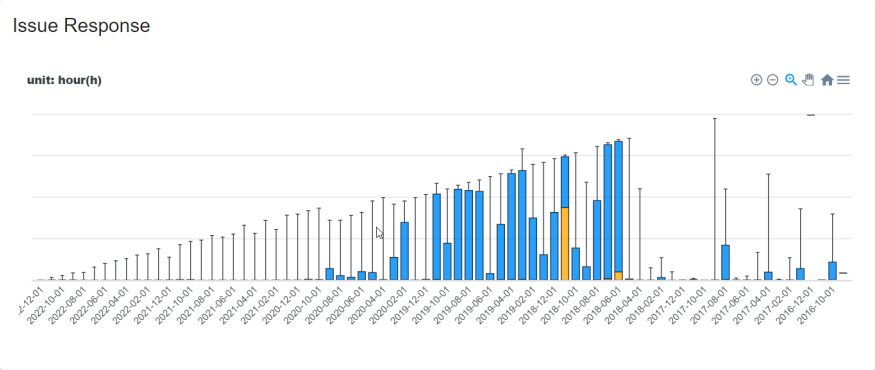

图 8: issue 从提出到第一次得到 response 的平均时间变化

A.1.3 界定并显示核心贡献者

原始需求 SE-UC-9: 界定出核心贡献者（28 定律），统计数量，按照时间顺序显示在图表上

开发前的预想 我们对核心贡献者的定义为：在一个时间段内，提交了该时间段内 80% 的 commits 的贡献者们（按贡献次数降序排列，贡献次数多的优先计入，直到满 80% 为止），将其计入核心贡献者。我们预期采用折线图的方式对各个时期的核心贡献者进行统计。并给出一个列表进行对各个核心贡献者进行展示。

最终的效果 我们依照我们开发前的设想为核心贡献者绘制了折线图。考虑到我们的折线图是以年为单位展示的，因此其 X 轴会比较短，所以我们决定将折线图与核心贡献者列表放在同一行。而核心贡献者列表由上方的浮动按钮组以及下方的列表组成，下方列表的内容会根据上方按钮组的选择而动态切换。

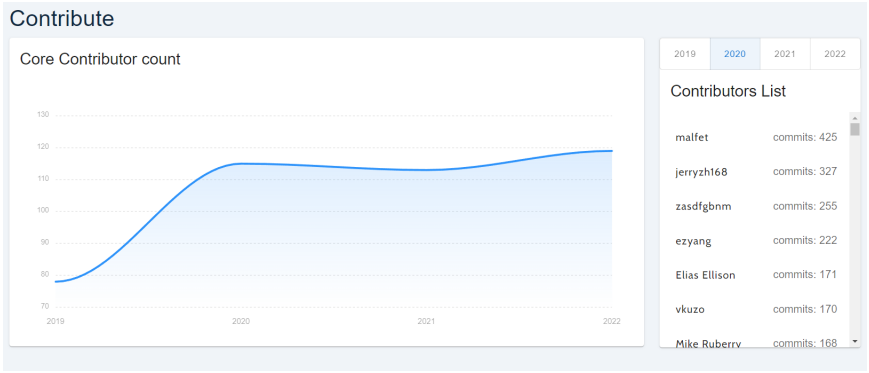

图 9: 核心贡献者数据

A.1.4 显示核心贡献者公司

原始需求 SE-UC-10: 显示核心贡献者背后的公司比例

开发前的预想 在我们完成对核心贡献者的界定与显示后，我们希望能够显示核心贡献者背后的公司比例，即从贡献者的比例显示转变到对于公司贡献占比的显示。我们预期采用气泡图的方式对社区核心贡献者背后的公司比例进行统计，并通过对应气泡图的大小数量来对各个公司贡献占比进行展示。

最终的效果 我们依照我们开发前的设想为核心贡献者背后的公司比例绘制了气泡图。并且在这基础上我们将核心贡献者背后的公司依据贡献者本身拆分成若干数量且大小不一的气泡，优化了公司贡献占比的展示，方便了我们后续对公司与社区关系之间更好地分析。

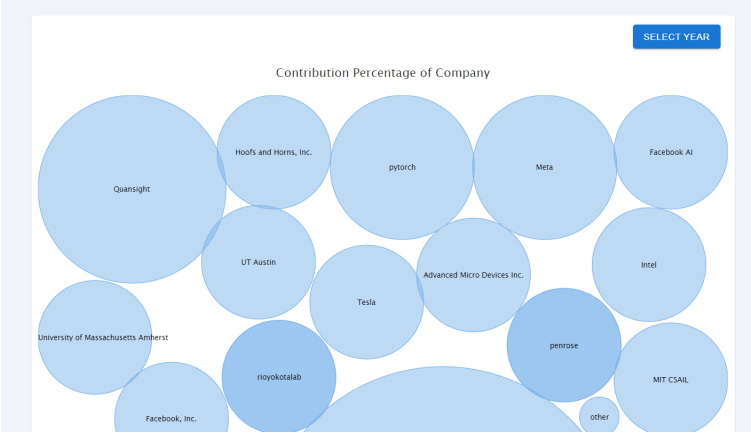

图 10: 公司占比气泡图

A.1.5 分离设计相关的话题

原始需求 SE-UC-11: 将和设计有关的 issue 和 pull 分离出来（标准自洽即可）

开发前的预想 通过 NLP 或模糊搜索的方式，将 issue 和 pull request 根据不同的主题进行分类。由于 NLP 缺乏大量已经标记的数据集，并且我们在机器学习方面能力有限，所以打算采用模糊搜索的方式分离话题。

最终的效果 因为数据存储在 MongoDB 这样的 NoSQL 数据库中，并且我们期望模糊查询的能力不局限于 like "*key*" 的形式，所以采用了 ElasticSearch 作为数据的搜索引擎，实现了存储引擎和搜索引擎的分离。为了让搜索引擎能够及时获取存储引擎的更新，采用 mongo-connector，并开启 MongoDB 的副本模式，监听 MongoDB 的更新日志来同步到 ElasticSearch。

得益于 ElasticSearch 的倒排索引和强悍的全文计分模糊搜索机制，我们对不同话题预先设定好关键词集后，便可以从 ElasticSearch 中获取到任意时间段内的 issue 和 pull request 话题数量分布。

当然，这个结果也并不是精确的，通过对搜索出来的内容的观察，发现如“测试”话题设置了“test”关键词，但是许多主要目的并非测试，而是增加新功能/报告 bug 等的内容，因为也做了少量测试，涉及到"test" 这个词而被包括进去。然而，判断一个 issue / pull request 的核心要义到底是什么而非断章取义，这脱离了模糊查询的能力范围，需要 NLP 和大量优秀数据的训练。但瑕不掩瑜的是，我们分离出来的话题内容总得来说已经具备较好的相关性。

A.1.6 分析展示热门话题

原始需求 SE-UC-12: 使用表格分析上述分理处来的 issue 和 pull，绘制热门话题随时间变化的图表

开发前的预想 并没有明确的展示方法，仅仅只是想要通过简单的表格或者词云图的形式将与设计相关的热门话题展示出来，并加以排序，但是这并没有很好的满足我们预想中能够展示社区活跃度这一预想要求。

最终的效果 最终我们为了突出社区的活跃度变化所以增加了时间轴，并且在这基础上通过饼图来更好的表示热门话题随时间的变化过程。通过不同时间热门话题不同的占比，我们就可以更好的观察社区中热门话题的变化情况。

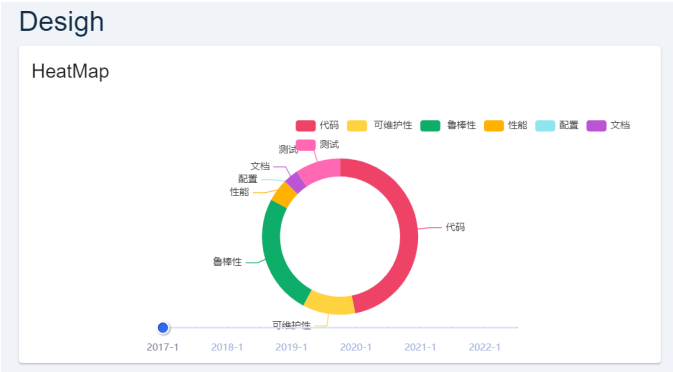

图 11: 分析展示热门话题

A.1.7 仓库对比

原始需求 SE-UC-12: 仓库对比

开发前的预想 我们预期将在仓库的详情页面提供一个仓库对比的按钮，点下按钮之后就会弹出一个页面，可以选择对比的仓库，选择完成之后可以进入到一个仓库对比的数据展示。而对比数据的展示方面，在我们预想中有两种思路，一种是直接使用文字对比的数据进行展示；另一种是选择和我们的主页面一样，将所有图表改成拥有两个系列的图表，在一张图上进行对比。

最终的效果　我们设计了如图中的仓库对比选择按钮，是采用了一个下拉框结合确认按钮的形式，并没有采用之前设想的弹出新窗口的设计。一是这种方案比较简单，二也是这样也相对比较整洁，很好地利用了之前的空余空间进行设计。

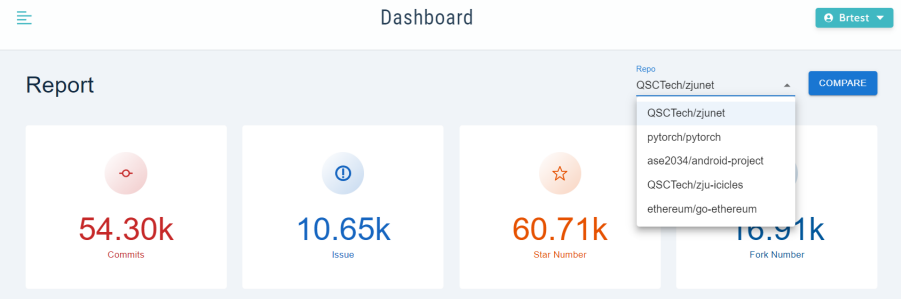

图 12: 仓库对比按钮

我们在对比数据的展示方面并没有采取之前的两种思路，而是采用了第二种的改良版本。第一种思路过于简单，并不能达到较好的对比效果；而第二种思路看似比较美好，但是由于原本的图表已经比较复杂，有些图表中已经存在了不止一条曲线，如果再加上一个对比的仓库的数据，将会使整个图标的信息变得非常复杂，因此我们舍去了一些不重要的数据对比来简化整个功能。我们也在之前 Compare 按钮的同个地方放置了回到原先页面的按钮，使整个页面的风格统一。我们仓库对比展示数据方面，参考了 GitHub 仓库对比工具——github-compare，通过对两个不同仓库之间 commits,forks,stars,open issues 等各项数据的对比，以方便每个仓库依次从以上各指标去其仓库首页看一下相关数据，进而更合理地衡量其稳定与否。

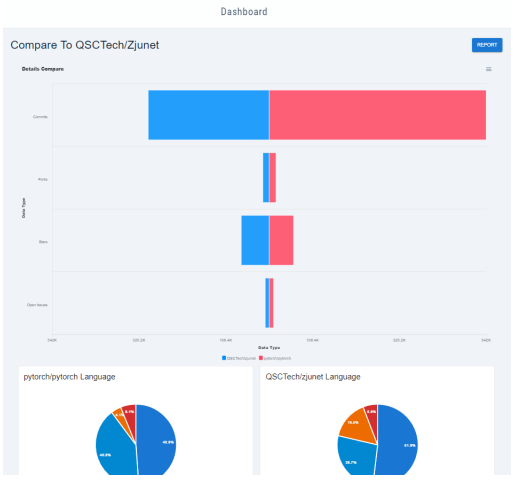

图 13: 仓库对比效果示意图