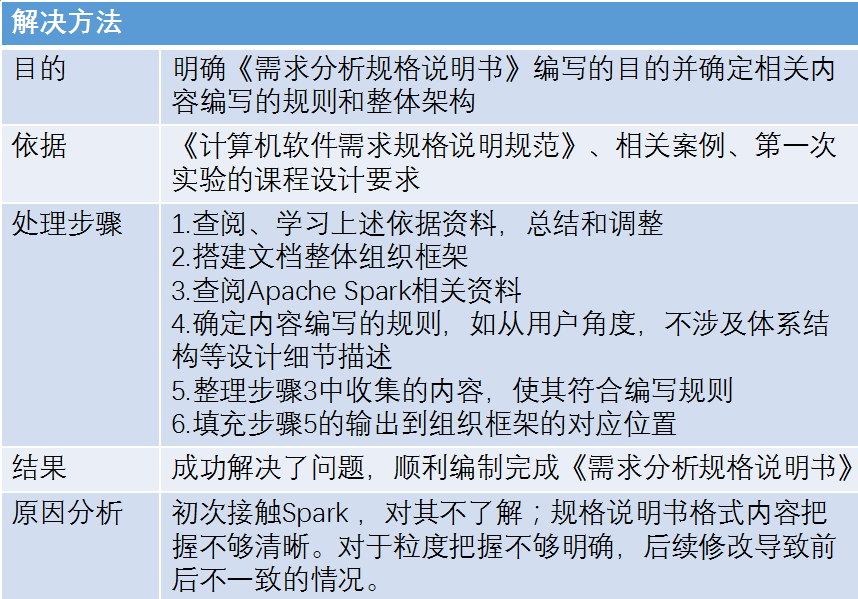
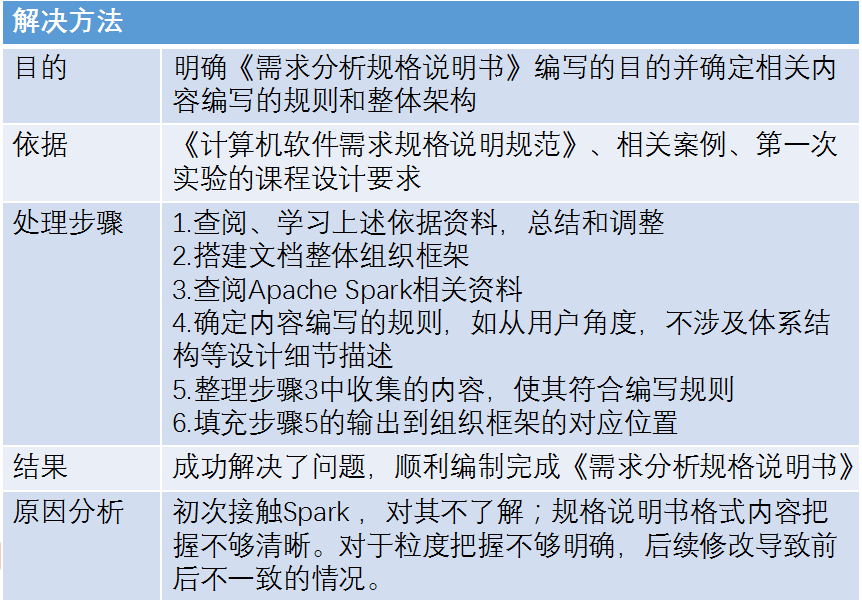
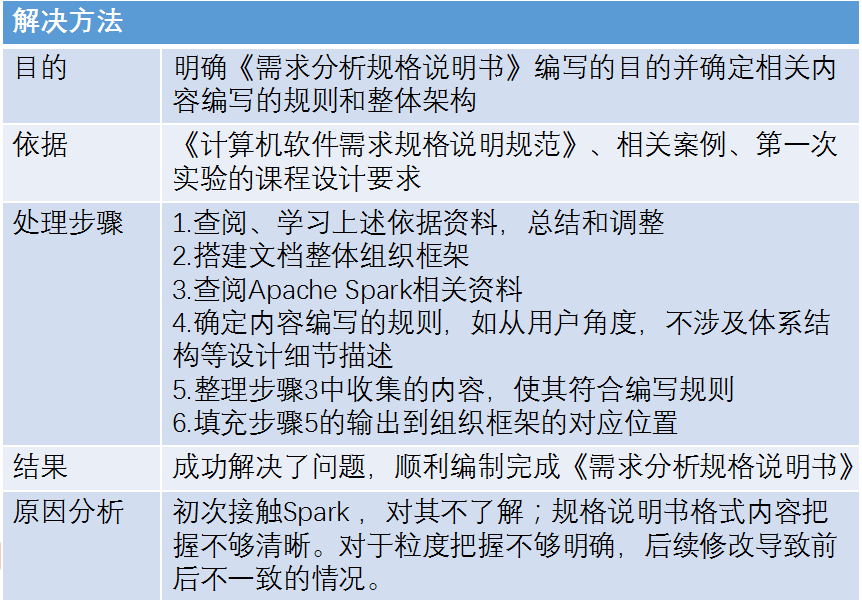
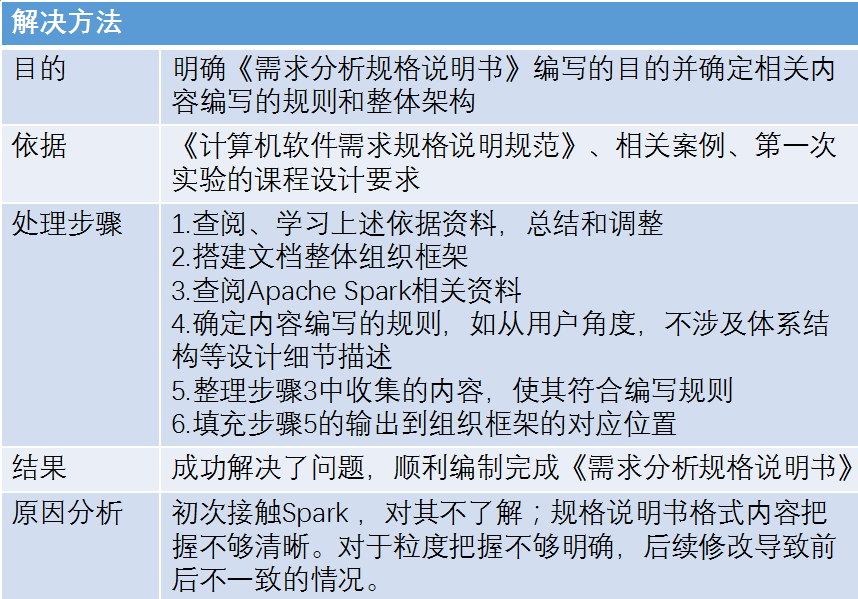
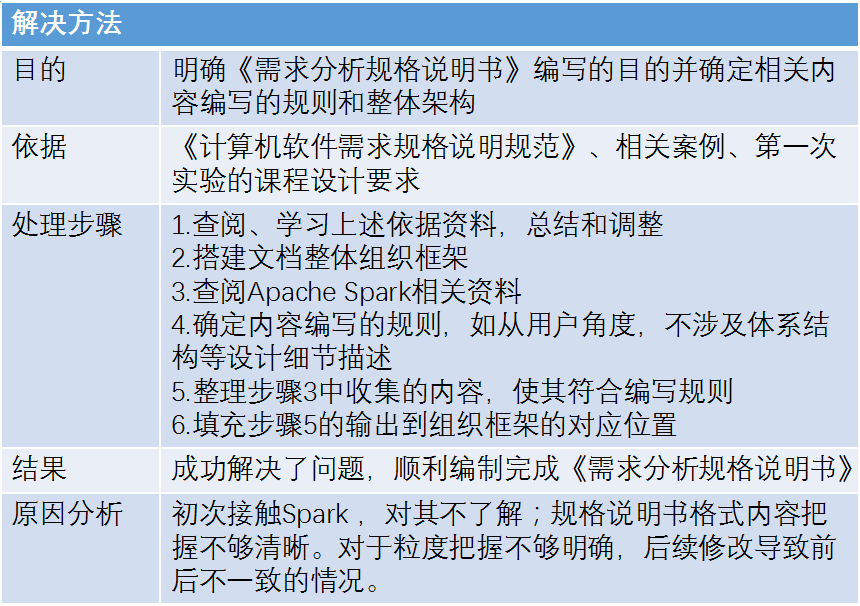
# 实验1 软件需求分析

**1.1 主要问题**

《需求分析规格说明书》中描述对象、描述深度拿捏不准、与后续文档存在前后不一致的情况。

**1.2 解决方法**



**实验2 软件需求评审**

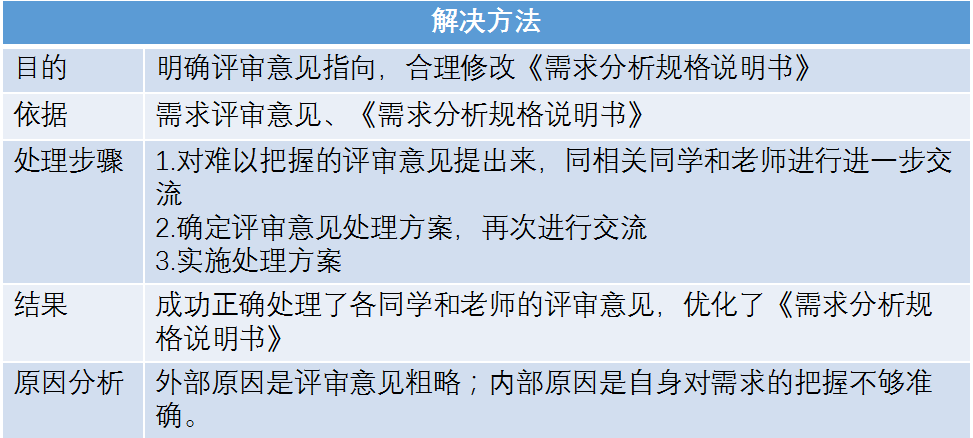
**2.1 主要问题**

1）在需求评审阶段，存在难以把握评审意见实际指向，如x较少、y模块的描述较简单。

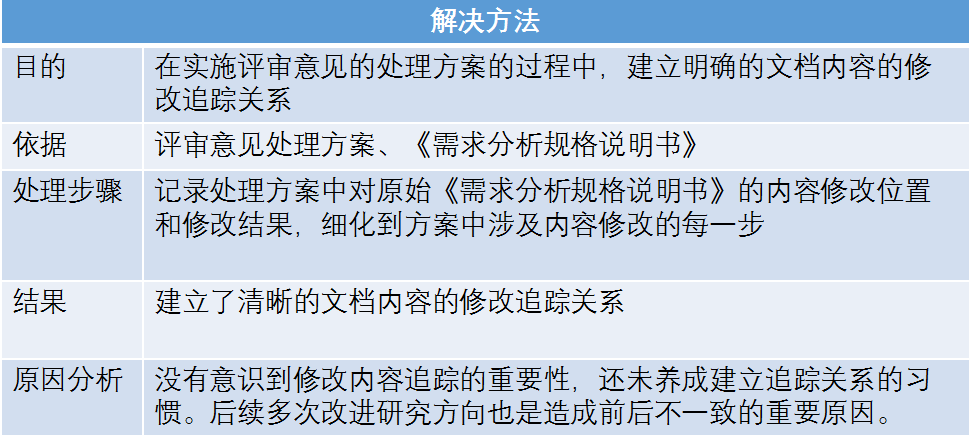
2）在需求评审阶段，未实现良好的文档内容的修改追踪关系。

**2.2 解决方法**

针对问题1）：



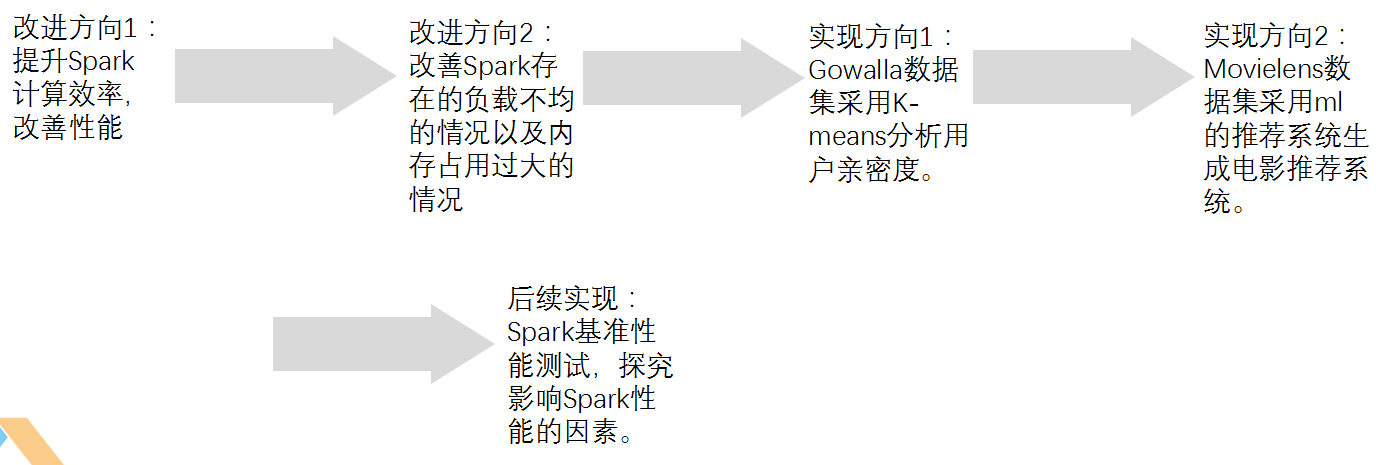
针对问题2）：



# 实验3 软件产品改进与展示

**3.1 主要问题**

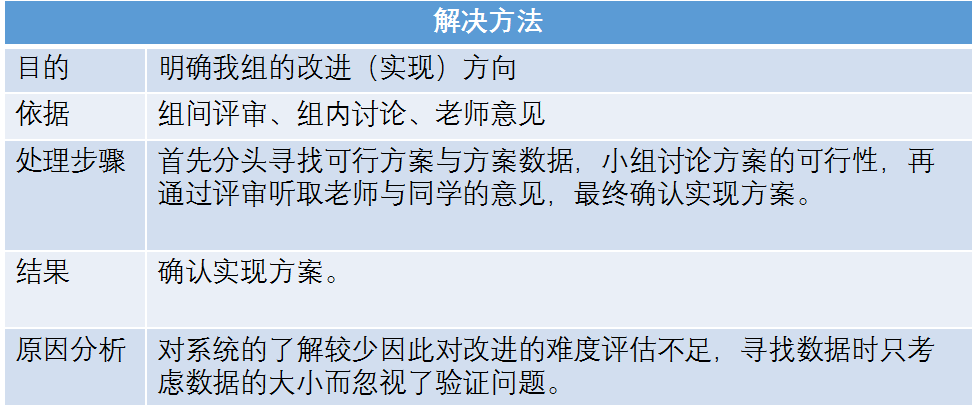
1）在明确改进方向上，我组进行了多次调整，对进度产生了一定影响。



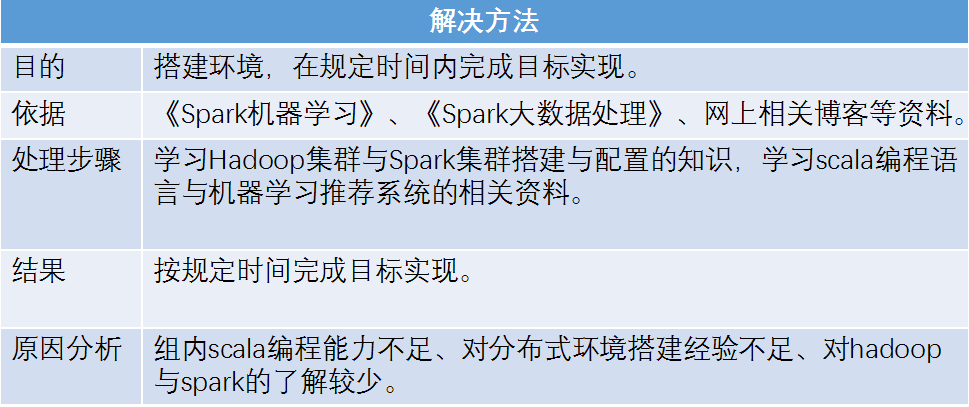
2）实现过程中，我组由于对scala语言以及spark的不熟悉，导致环境搭建与编程实现在初期效率较低。

**3.2 解决方法**

针对问题1）：



针对问题2）：



# 实验4 软件测试

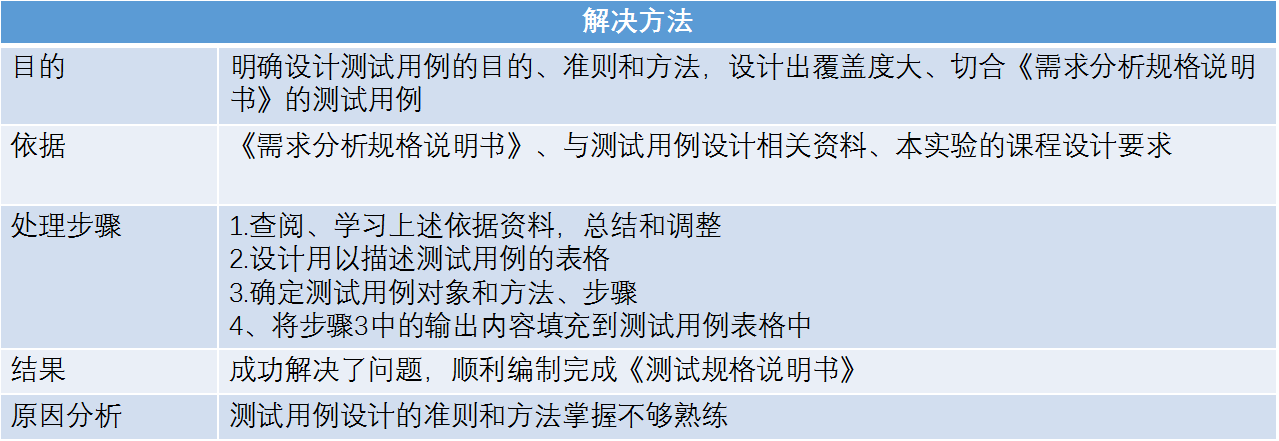
**4.1 主要问题**

1. 测试用例设计的不够符合要求。

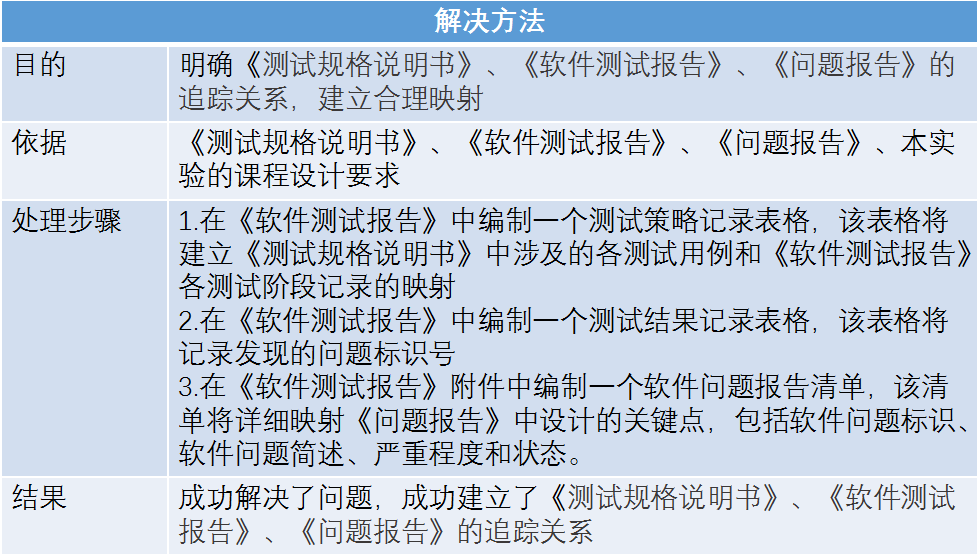
2）《测试规格说明书》、《软件测试报告》、《问题报告》三者的追踪关系有待进一步优化明确。

**4.2 解决方法**

针对问题1）：



针对问题2）：

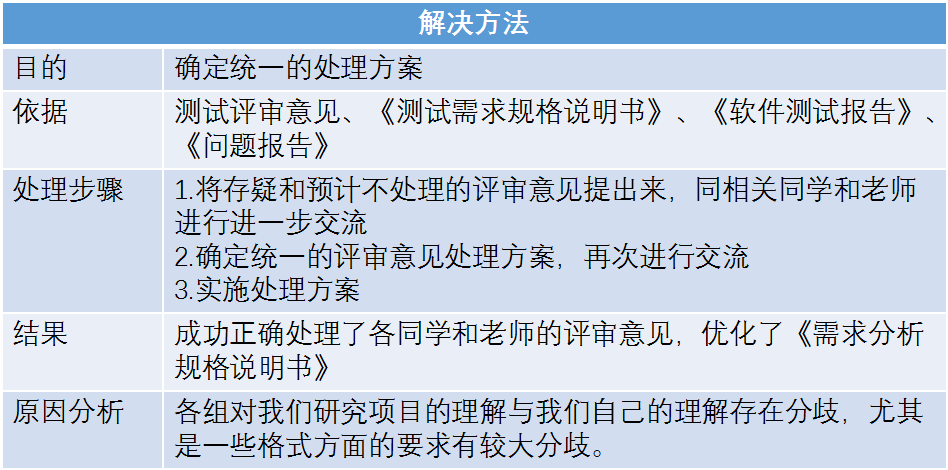


**实验5 软件测试评审**

**5.1 主要问题**

在测试评审阶段，会存在与评审意见认识不一致的现象。

**5.2 解决方法**



# 实验6 软件进度计划与控制

## 6.1 主要问题

本实验的最大问题就是：工时总数超标，即：调整前总工时为3926h。核算到每日每人工作量为8h之多，这明显是不符合实际情况的。因此，需要进行改进调整。

## 6.2 改进之道

根据项目的实际情况，以及老师提出的建议，我们对project的mpp文件做了如下几点调整：

1. 设置日程：每日工时为2h，每周工时为14h。即：每人每天2h，每周7日工作制；

2. 清除了父级人物中的人员分配。此举将工时大概降低了4倍，符合老师的推理。

3. 根据实际情况，对个别任务的资源分配进行了unit操作。

## 6.3 改进对比

如图6.1和6.2所示，改进前总工时为：3926h，改进后总工时为：647h。



图6.1 改进前总工时



图6.2 改进后总工时

## 6.4 工时分布

如图6.3所示，孟翰：162h；苏若：165hh；李璇：166h；孙敏芳：165h

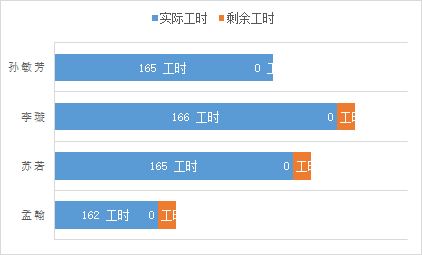


图6.3 资源/工时统计

单就工时来看，小组内分工基本持平。这是源于每周小组会议上，会根据之前的累计工作量，按照实际情况，进行下一周任务的均衡分配。

但是，工时并不能够完全反应个人工作量的问题，比如，身为组长的孟翰同学，承担了大部分代码工作，相对来说，同一工时的代码工作要比文档工作的技术含量更高。因此，工时仅作为组员贡献率的一个主要参考因素。

## 6.5 其他统计指标分析

### 6.5.1 燃尽图

本项目组的燃尽图如图6.4所示，我们发现燃尽图的实际曲线与基线几乎完美符合，这在实际软件工程实践中，几乎是不可能实现的。究其原因，是因为我们开始基础能力有限，并不能很好的使用project做出合理的计划，并如实记录实际项目进度。而是，将其作为了一个项目进度的实际记录工具，因为每周小组会议后，都会对之前的mpp文件进行实时更新更正，以符合项目的实际进度，这就失去了project软件的原有作用。这个教训值得我们引以为戒。

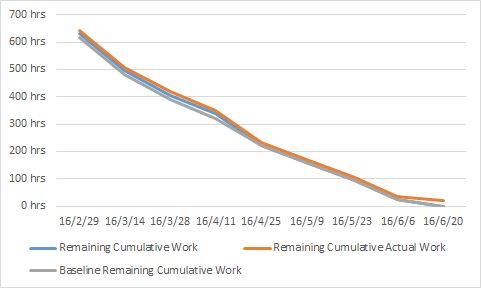


图6.4 燃尽图

### 6.5.2 工时统计

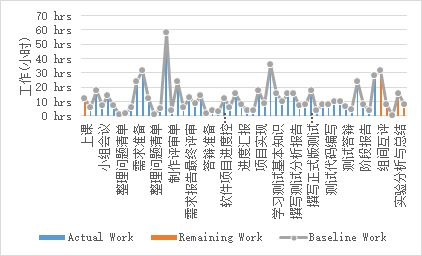


图6.5 工时统计

我们项目组的工时统计如图6.5所示，由图6.5可知，我们项目组的大部分时间用在了软件再评审、项目实现和组间互评上。项目实现和组间互评需要大量时间，这点是毋庸置疑的。所以，针对于个性情况，即：我们组的软件再评审阶段耗费了大量工时/资源，做出分析。由于我们组的实验目的前后总共进行了3次更正改进，这个时间恰好处于再评审阶段，而实验目的的改进，恰恰需要不断的同步修正改进项目需求设计规格说明书，所以我们组的工时资源有大块时间耗费在了这一阶段之上。这也告诉我们，从一开始就要尽量根据实际情况和自身能力，确定好项目的实验目的，即我们要做什么。

### 6.5.3 资源剩余可用性

如图6.6所示，我们可以看到我们项目组的资源（人员工时）可用性的变化情况，每个月的月底，可用性都会降至极点，这一现象恰恰符合本次软件工程实验的阶段性，即：每个月月底恰恰是需求、测试、统计三大模块的结尾阶段，这一阶段的人员/资源利用率较高，剩余利用率处于该附近时期的最低点。即，大家都在忙于该阶段的总结评审。

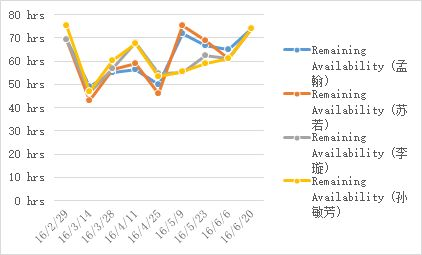


图6.6 资源剩余可用性

# 实验7. 工作量估计与统计分析

本实验的工作量统计，由每阶段的文件统计体现，实际情况如表7.1所示：

图7.1 软工实验文件统计

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **阶段** | **名称** | **所需文档** | **时间（周）** | **版本数量** |
| 实验1 | 软件需求分析 | 软件需求规格说明书 | 4-9 | 7 |
| 实验2 | 软件需求评审 | 软件评审报告 | 6-7 | 2 |
| 软件问题报告 | 4-8 | 4 |
| 实验3 | 软件产品改进与展示 | 软件设计模型及文档 | 11-13 | 3 |
| 改进与展示的软件产品 | 15 | 1 |
| 实验4 | 软件测试 | 被测软件执行结果与分析报告 | 13 | 1 |
| 软件问题报告 | 12 | 1 |
| 实验5 | 软件测试评审 | 软件测试的评审报告与问题报告 | 11-14 | 5 |
| 实验6 | 软件进度计划与控制 | 工作日志 | 3-14 | 14 |
| 进度控制分析 | 3-17 | 17 |
| 实验7 | 工作量估计与统计分析 | 工作量分析报告 | 6-14 | 3 |
| 实验8 | 配置管理 | 变更与管理分析报告 | 10-16 | 3 |

其余每个阶段具体的字数统计，请见本实验的统计分析说明书。

# 实验8. 配置管理

## 8.1 Git管理

Git目录的前后变更如图8.1、8.2所示。图8.1为参照师兄师姐的作业做的目录，后续实践证明这样设计目录容易出现杂乱的问题，主要原因是实验初期对整个实验的阶段和制品不是很熟悉。

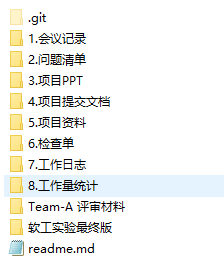


图8.1 初始版Git目录

如图8.2，是最后提交的实验作业Git目录，这是根据老师最终需要提交的作业，按每个阶段和相应的小实验做的划分，较为清晰明确。这告诉我们，以后要先提纲挈领，才能更好地建造出性价比高的房子/工程。

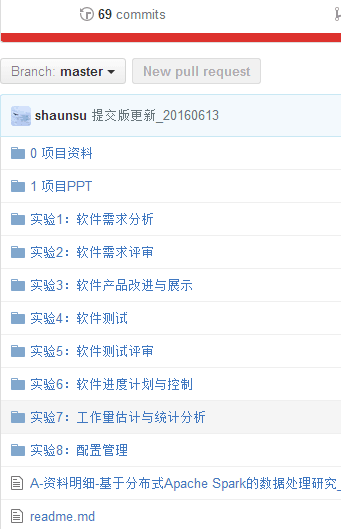


图8.2 最终版Git目录

## 8.2 上传统计

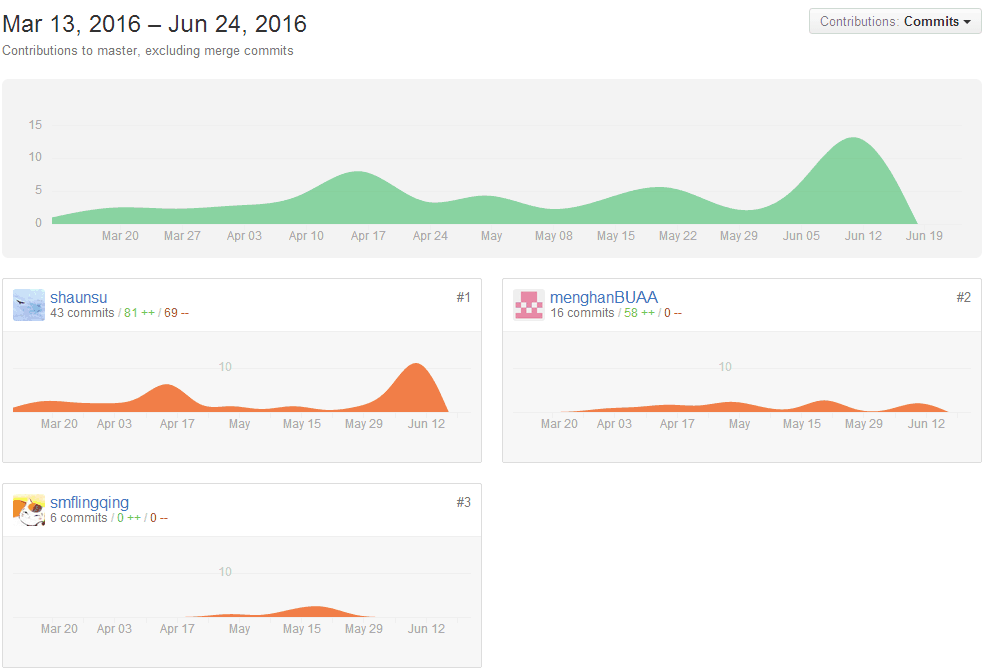


图8.3 上传统计

如图8.3所示，这是我们组Git上传记录的分布图，从图中可以看出，与上个实验中的分布相同，每个子阶段/月底，会有频繁的提交更改记录。同时，因为我们组为了便于管理，只通过个别人上传了记录，所以无法从本记录中得知每个组员的大概贡献。

## 8.3 文件命名

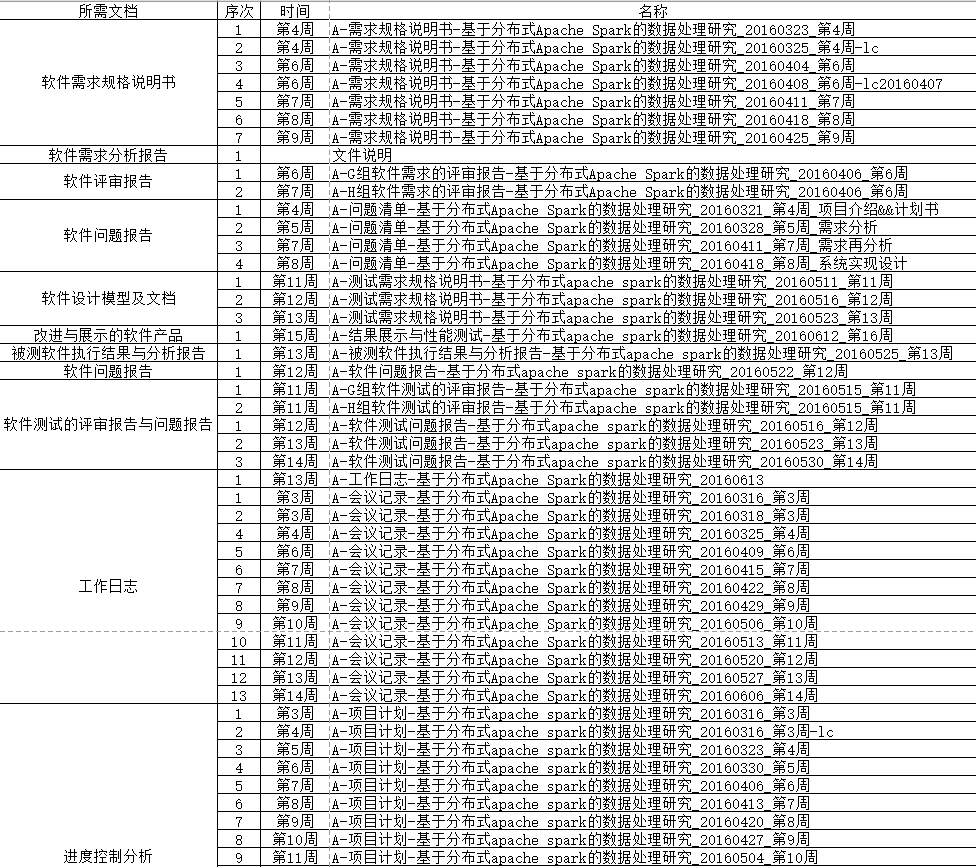


图8.4 文件命名

尽管，很多组同学给我们提出了文件命名冗余的问题，但是我们觉得这样命名也有它的可取之处。

1） 格式：A-\*\*\*\*\*-基于分布式Apache Spark的数据处理研究\_2016\*\*\*\*\_第\*周；

2） 命名规则：

* “A”：表明组别，便于老师&同学区分；
* “\*\*\*\*\*”：说明文档类型，例如：需求规格设计说明书；
* “基于$$$研究”：提醒老师&同学研究内容，因为老师需要查看的文档很多，不同组的研究内容又不相同，所以我们将本组研究内容添加在文件名信息中，这样老师就可以时时看到本组的研究内容，进而确定研究环境，这也是沉浸交互的一种表现方式；
* “2016\*\*\*\*”：便于我们组同学配置管理和后期统计需要；
* “第\*周”：同上，便于我们组同学配置管理和后期统计需要。

客观来说，我们组这样的命名方式是具有其独特性和优点的。