

**基于Scrapy的模板化爬虫程序管理平台**

**实验总结报告**

v1.0

赵正阳 SY1906428

郭浩隆 SY1906430

沈一聪 SY1906510

梁远志 SY1906503

宋冰晨 SY1906429

2020年6月11日

**版本变更记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 日期 | 修改人 | 修改章节 | 备注 |
| 1.0 | 2020/6/10 | 赵正阳 | 2、3.3、5.8 | 完成与实验8相关的章节、整合初稿 |
| 郭浩隆 | 5.1~5.5 | 完成“有效方法” |
| 沈一聪 | 1、4.1、4.2、6 | 完成“概述”、“制品质量水平”和“结论与建议” |
| 梁远志 | 3.2、4.4、5.7 | 完成与实验7相关的章节 |
| 宋冰晨 | 3.1、4.3、5.6 | 完成与实验6相关的章节 |

目录

[**1.** **概述** 1](#_Toc42787885)

[1.1 项目概述 1](#_Toc42787886)

[1.2 本文内容 1](#_Toc42787887)

[**2.** **工作量统计** 1](#_Toc42787888)

[2.1 软件需求分析 2](#_Toc42787889)

[2.2 软件需求评审 2](#_Toc42787890)

[2.3 软件设计与实现 2](#_Toc42787891)

[2.4 软件测试 3](#_Toc42787892)

[2.5 软件测试评审 4](#_Toc42787893)

[2.6 软件项目计划与监控 4](#_Toc42787894)

[2.7 软件配置管理 5](#_Toc42787895)

[2.8 软件工程实验追踪与分析 5](#_Toc42787896)

[2.9 文档规模对比 5](#_Toc42787897)

[**3.** **数据分析与说明** 6](#_Toc42787898)

[3.1 实验6分析结果 6](#_Toc42787899)

[3.2 实验7分析结果 6](#_Toc42787900)

[3.3 实验8分析结果 7](#_Toc42787901)

[**4.** **制品质量水平** 8](#_Toc42787902)

[4.1 软件需求和测试需求 8](#_Toc42787903)

[4.1.1 软件需求 8](#_Toc42787904)

[4.1.2 测试需求 8](#_Toc42787905)

[4.1.3 评审质量 10](#_Toc42787906)

[4.2 软件设计与实现 11](#_Toc42787907)

[4.2.1 软件的预期设计 11](#_Toc42787908)

[4.2.2 软件最终实现评估 12](#_Toc42787909)

[4.3 进度计划与控制 12](#_Toc42787910)

[4.3.1 实验过程及特点 12](#_Toc42787911)

[4.3.2 实验进度影响因素 13](#_Toc42787912)

[4.3.3 进度控制的有效性 13](#_Toc42787913)

[4.4 配置管理 14](#_Toc42787914)

[4.4.1 配置管理方法 14](#_Toc42787915)

[4.4.2 质量水平分析 15](#_Toc42787916)

[**5.** **有效方法** 15](#_Toc42787917)

[5.1 软件需求分析 15](#_Toc42787918)

[5.1.1 如何确定软件需求 15](#_Toc42787919)

[5.1.2 如何描述业务需求 16](#_Toc42787920)

[5.2 软件需求评审 16](#_Toc42787921)

[5.2.1 如何针对他组的需求文档进行评审 16](#_Toc42787922)

[5.2.2 如何根据收到的评审意见修改项目需求 17](#_Toc42787923)

[5.3 软件设计与实现 17](#_Toc42787924)

[5.3.1 如何确定软件的架构设计与分工 17](#_Toc42787925)

[5.3.2 如何在软件设计过程中进行团队交流 17](#_Toc42787926)

[5.3.3 如何对软件设计成品进行展示 18](#_Toc42787927)

[5.4 软件测试 18](#_Toc42787928)

[5.4.1 如何设计测试用例 18](#_Toc42787929)

[5.4.2 如何制定高效的测试计划 18](#_Toc42787930)

[5.5 软件测试评审 19](#_Toc42787931)

[5.5.1 如何针对他组的测试文档和软件进行评审和测试 19](#_Toc42787932)

[5.5.2 如何根据收到的评审意见和测试结果改进测试和项目 19](#_Toc42787933)

[5.6 项目计划与监控 20](#_Toc42787934)

[5.6.1 如何有效控制项目按照计划进行 20](#_Toc42787935)

[5.7 配置管理 21](#_Toc42787936)

[5.7.1 如何合理配置目录结构 21](#_Toc42787937)

[5.7.2 如何管理历史文档 21](#_Toc42787938)

[5.7.3 如何设计文档的变更过程 22](#_Toc42787939)

[5.7.4 如何设计代码的变更过程 22](#_Toc42787940)

[5.8 工程实验追踪与分析 22](#_Toc42787941)

[5.8.1 如何进行有效的工作量统计 22](#_Toc42787942)

[5.8.2 如何根据阶段性工作量统计结果调整分工 22](#_Toc42787943)

[**6.** **结论与建议** 23](#_Toc42787944)

[6.1 实验结论 23](#_Toc42787945)

[6.2 课程建议 23](#_Toc42787946)

1. **概述**

1.1 项目概述

本次软件工程综合实验遵循了软件工程的开发过程，过程中包含了需求分析、软件测试等方面，还补充了评审相关部分，具体过程包括：软件项目计划阶段、软件需求分析阶段、软件需求评审阶段、软件产品改进与展示阶段、软件测试阶段、软件测试评审阶段。除了以上软件开发过程应用的步骤，还对软件开发的过程进行了管理与分析，具体包括：软件进度计划与控制、配置管理、工作量估计与统计分析，这些管理工作贯穿整个项目始终。

在实验过程中，既完成了软件工程开发所要求的开发过程，又在开发过程中通过软件进度计划与控制、工作量估计与统计分析、配置管理这些工作对整个项目开发进行管理和控制，在保证项目按照计划执行的同时，还保留了项目执行和开发过程中的各项证据，进一步形成了证据链，使得项目的开发与执行更具有说服力。

本次实验基本实现了预期的工作目标，通过小组合作的方式实现了Easyspider——基于Scrapy的模板化爬虫程序管理平台的完整开发过程，对开发过程进行了管理，对产生的数据进行了统计分析。

1.2 本文内容

本文主要分为6个章节对软件工程实验所做的工作进行总结：

第1章对软件工程实验课程进行了总结，并介绍了全文的主要内容；

第2章统计了共计八个阶段的实验中，每个实验的工作量；

第3章从完整性和准确性具体分析了实验6-8数据来源的可靠程度；

第4章分析了软件需求分析、测试需求分析、评审、设计与实现、进度计划与控制、配置管理等方面工作产出物的质量水平；

第5章对整体实验过程中应用的有效的项目方法进行了总结；

第6章给出了实验结论以及课程的一些建议。

1. **工作量统计**

本节将对每个实验的总工作量进行总结及分析说明。每个组员具体的工作量见实验8的《工作量统计分析报告》及相关表格。

2.1 软件需求分析

表2.1 软件需求分析工作量

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文档名称 | 字数 | 图表数 | 需求个数 | 更新版本数 |
| 软件需求规格说明书 | 14996 | 架构图：2  用例图：3  RUCM图：51  其他：3 | Scrapy框架：14  Scrapyd框架：10  Web UI: 17  调度器：10 | 15 |

在需求分析阶段，我们对所使用的Scrapy框架、Scrapyd框架以及本项目共提出了51个需求用例，并使用用例图和RUCM图对其进行规范描述。需求规格说明书的主要篇幅集中在使用RUCM图描述需求，因此文档的图表数较多。另外，文档的更新版本数较多，主要是因为组员大都初次接触比较正规的软件需求分析，因此文档初稿存在用例不完整、RUCM图不规范等问题。在实验2软件需求评审中经过其他组评审和老师指导后进行了多次修改，使最终版本成为较为规范的软件需求规格说明书。

2.2 软件需求评审

表2.2 软件需求评审分析工作量

|  |  |  |  |
| --- | --- | --- | --- |
| 评审阶段 | 提出评审意见数 | 收到评审意见数 | 修改评审意见数 |
| 初评审 | F: 24  G: 23 | A: 19  I: 24 | A: 19接受  I: 20接受、3部分接受、1拒绝 |
| 复评审 | A: 17  I: 41 | F: 21  G: 22 | F: 19接受、2部分接受  G: 18接受、4部分接受 |

在需求评审阶段，我们对4个被评审组共提出105个评审意见，共收到来自4个评审组的86个评审意见。收到的评审意见中共有1个拒绝、9个部分接受，其他全部接受。

2.3 软件设计与实现

表2.3 软件设计工作量

|  |  |  |  |
| --- | --- | --- | --- |
| 文档名称 | 字数 | 图表数 | 更新版本数 |
| 实现方案 | 6356 | 15 | 1 |

表2.4 软件实现工作量

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 仓库名称 | commit数 | PR数 | Python行数 | HTML行数 | CSS行数 | JS行数 |
| easyspider-web | 118 | 72 | 2712 | 4138 | 1118 | 642 |
| easyspider-scheduler | 11 | 8 | 508 | 0 | 0 | 0 |
| easyspider-site-douban | 53 | 0 | 1837 | 0 | 0 | 0 |
| easyspider-template | 4 | 0 | 443 | 0 | 0 | 0 |

在软件设计阶段的实现方案文档中，我们使用了组件图、流程图、顺序图、ER图等多种软件工程上常用的图来描述我们的代码设计。

从表2.4可以看出，在软件实现阶段中我们的主要工作量集中在easyspider-web仓库，即Django网站项目上。代码行数可以体现出我们的项目具有较高的难度和复杂度，commit数和PR数可以体现出我们在开发中协同工作的工作量。

2.4 软件测试

表2.5 软件测试文档工作量

|  |  |  |  |
| --- | --- | --- | --- |
| 文档名称 | 字数 | 图表数 | 更新版本数 |
| 测试需求规格说明书 | 25897 | 0 | 11 |
| 软件测试计划 | 6123 | 3 | 3 |
| 被测软件执行结果分析报告 | 4100 | 0 | 4 |
| 软件可用性报告 | 3575 | 0 | 2 |
| 软件问题报告 | 1687 | 0 | 3 |
| 性能测试报告 | 1033 | 2 | 3 |

表2.6 测试用例统计

|  |  |
| --- | --- |
| 测试用例种类 | 个数 |
| Django单元测试 | 75 |
| Scrapy单元测试 | 16 |
| 调度器单元测试 | 16 |
| 集成测试 | 3 |
| 场景测试 | 6 |
| 功能测试 | 33 |
| 非功能测试 | 11 |
| 性能测试 | 3 |

在软件测试阶段，我们共完成文档42415字、图表5个、各类测试用例163个，文档较多，工作量非常大。其中最重要的文档是测试需求规格说明书，经历两次评审和多次修改，因此更新版本数也最多。从表2.6可以看出，我们的测试用例设计比较全面，覆盖了多种测试方法，因此能够对我们的软件进行比较全面的测试。

2.5 软件测试评审

表2.7 软件测试评审分析工作量

|  |  |  |  |
| --- | --- | --- | --- |
| 评审阶段 | 提出评审意见数 | 收到评审意见数 | 修改评审意见数 |
| 初评审 | D: 37  E: 13 | B: 6  C: 29 | B: 5接受、1部分接受  C: 28接受、1部分接受 |
| 复评审 | B: 16  C: 37 | D: 17  E: 15 | D: 17接受  E: 13接受、1部分接受、1拒绝 |

在软件测试评审阶段，我们对4个被评审组共提出103个评审意见，共收到来自4个评审组的67个评审意见。收到的评审意见中共有1个拒绝、3个部分接受，其他全部接受。

2.6 软件项目计划与监控

表2.8 软件项目计划与监控工作量

|  |  |
| --- | --- |
| 进度控制分析报告字数/图表数 | 7086/10 |
| 项目进度计划(MPP)更新版本数 | 12 |
| 工作日志提交周数 | 11 (4~14) |

软件项目计划与监控主要由宋冰晨负责。各组员每周上传本周工作日志，目前共提交了11周。负责人每周会根据组员提交的工作日志来更新MPP文件，提交一个新版本。最后由负责人撰写进度控制分析报告对该实验的整体过程进行总结。

2.7 软件配置管理

表2.9 软件配置工作量

|  |  |  |  |
| --- | --- | --- | --- |
| 文档名称 | 字数 | 图表数 | 更新版本数 |
| 变更与管理分析报告 | 13755 | 0 | 1 |

软件配置管理主要由梁远志负责，主要工作包括创建GitHub仓库、制定Pull Request规则、配置CI/CD以及撰写变更与管理分析报告。

2.8 软件工程实验追踪与分析

表2.10 软件工程实验追踪与分析工作量

|  |  |  |  |
| --- | --- | --- | --- |
| 文档名称 | 字数 | 图表数 | 更新版本数 |
| 工作量统计分析报告 | 5933 | 8 | 6 |

软件工程实验追踪与分析主要由赵正阳负责。预先指定好每个实验的工作量指标及统计表格，之后每1~2周更新一次工作量统计数据。其中实验2和4的指标为评审意见个数，实验3为代码行数，其他实验主要为文档字数和图表数，此外还包括额外工作的工时（如文档整合）。

2.9 文档规模对比

图2.1展示了每个实验的文档规模对比。由于实验2和实验5只有评审意见没有文档，因此未在其中展示。图表数与字数的折算关系采用与实验8中统计工作量相同的方法。

图2.1 文档规模对比

从图中可以看出，文档规模最大的是实验4和实验1，分别对应测试需求规格说明书及其他测试文档和软件需求规格说明书，这一结果也与我们在各实验上所花费的工时比例相同。

1. **数据分析与说明**

3.1 实验6分析结果

（1）数据的完整性和准确性

实验六涉及的数据包括组员工作日志和MPP文件。

工作日志是组员自己填写的，包含的内容是组员自己在当周完成的所有任务和耗时。在数据完整性方面，目前仓库里有第4周至第14周的工作日志文件，缺少了教学第2-3周（即3月6日第一节软件工程综合实验课至3月13日第二节软件工程综合实验课）的工作日志；在数据准确性方面，工作日志里的任务耗时只精确到0.5h，且由个人上报，存在一定的主观性。

MPP文件是在每周工作日志的基础上汇总形成的，目前已经更迭了12个版本，对应教学第2周至教学第14周，因此除主观因素之外MPP文件的完整度为100%。由于工作日志缺少一周的记录，所以MPP文件1.0版本是估算得到的，因此MPP文件最终版12.0的准确性略有欠缺。

（2）影响因素

工作日志的不完整主要是由于在项目开始阶段组员对实验6-8的产出制品了解不够，在甲方（任课教师）第二节课的明确说明下才开始记录。工作日志不准确的原因有两个：一是精确度为0.5h，二是任务耗时是由个人上报的。

MPP文件最终版12.0的不准确主要是因为MPP1.0是估算得到的，而不是根据日志文件汇总形成的。

（3）误差估计

根据MPP文件生成的报表可知，项目共耗时529小时，包含217个任务。如果不考虑主观因素，误差按照0.5小时计算的话，则工时误差率为。

3.2 实验7分析结果

实验7中的统计的数据覆盖了本组的所有制品文档以及附属于这些文档的子文档，共计12个文档。代码方面，也同样覆盖了本组进行实验的所有代码仓库，且这些仓库均公开可查。因此本组实验七的分析结果是完整的。

在文档变更管理上，本组能够通过两类日志追溯更改：一类是GitHub上的提交日志，该日志能够追溯到具体是哪一位组员在何时对文档文件进行了变更。同时这些Commit记录也会附带组员填写的提交信息；另一类是文档内部的版本变更情况情况表，该表记录了文档内的详细的变更情况，同样起到追溯作用。排除因为组员个人风格导致的提交信息和变更记录的差异，本组的文档变更记录的统计数据是准确的。

在代码管理上，本组所有的代码仓库都可以通过Git的Commit日志来追溯变更情况。本组在仓库设置了分支保护，也禁止了强制推送（force push）的发生，所有的变更历史都是可信的。同时本组在部分仓库使用了Pull Request，这一机制也在GitHub的相关页面以及特定分支中有可查证的记录。因此，这些记录的统计数据都是准确的。

3.3 实验8分析结果

实验8文档《工作量统计分析报告》中涉及的统计数据包括3类：（1）每个文档中各组员完成的字数及绘制的图表数；（2）各组员编写的每种语言的代码行数；（3）其他工作的工时。

各组员完成的文档字数是由实验8负责人根据每个文档的分工，在Word中选中相应章节后左下角显示的字数累加得到。该数据的统计结果存在两方面的误差。一方面，在统计时章节标题、图表标题等文字是否计入统计未做明确规定。但与正文内容相比，这类文字的字数几乎可以忽略，因此估计对统计结果造成的误差在5%以下。另一方面，文档变更也会带来统计字数的误差。有些文档统计完字数后又进行了修改，但变更记录中没有记录每次修改增加了多少字、删除了多少字，且这个数字难以统计，因此统计结果与文档最终版本中各组员完成的真实字数也存在误差。但考虑到我们组的文档没有经过大幅修改的，大部分只是一些格式标点方面的修改，因此估计这个误差在10%左右。

各组员绘制的图表数可以明确统计得到（不包括屏幕截图），因此该数据是准确的。

各组员编写的每种语言的代码行数是在所有开发工作完成后由实验7负责人使用Git命令统计得到，因此准确度较高。但由于在Git中每行代码所属的作者是最后修改的人，因此如果两个人修改过同一行代码则这行代码会被计入最后修改的人的代码行数。但考虑到我们的开发工作有明确的分工，不同组员修改同一处代码的情况极少出现，因此估计对统计结果造成的误差在5%以下。

其他工作的工时来自组员上传的工作日志，且时长为组员自己估计，仅精确到0.5 h或1 h，因此估计该数据的误差在15%~20%。

1. **制品质量水平**

4.1 软件需求和测试需求

4.1.1 软件需求

软件需求分析阶段的主要制品为软件需求规格说明书。本项目需求规格说明书总共迭代了15个版本。其中，第一次需求评审前3个版本，第一次需求评审后9个版本，第二次需求评审后3个版本，最终版本为《H-EasySpider-软件需求规格说明书v1.3.3》。规格说明书包含中文字符数为14996，功能需求32项（采用RUCM绘制用例），非功能需求14项。就需求分析环节的文档规模而言，本小组产出的需求规格说明书在各个小组中处于极为前部的位置。

在需求规格说明书整体的编写过程中，第一次评审共收到别组评审43项问题，其中严重4，中等15，轻微24；第二次评审共收到别组评审43项问题，其中严重0，中等9，轻微34，总体而言，收到的问题数处于众多小组中的中等水平，其中大部分意见来源于文档格式问题，但也有一定数量的意见直接指出需求分析工作的疏漏和错谬。对此，我们虚心采纳了其中的76个意见，部分接收了6个意见，仅仅拒绝了1个意见，并在各个版本的文档迭代过程中，根据评审意见积极修改文档，大幅度提升了文档内容的质量，修改后的文档也获得了各个评审小组的一致认可。

在最后的互评环节中，本小组的需求分析工作总计获得13分与E组并列第一。在后续的设计实现环节中，需求规格说明书极大的帮助了软件项目的开发，其中99%以上的需求被实现。综上所述，我们认为最终版本的需求规格说明书质量达到了十分优秀的水准。

4.1.2 测试需求

件测试阶段的主要制品为测试需求规格说明书。本项目需求规格说明书总共迭代了11个版本。其中，第一次需求评审前2个版本，第一次需求评审后3个版本，第二次需求评审后6个版本，最终版本为《H-EasySpider-测试需求规格说明书v1.3.6》。测试需求规格说明书包含中文字符数为25897，其中功能需求测试用例34项，非功能需求测试用例11项。此外还包括了107项单元测试，3项集成测试以及6项场景测试以及用户友好型测试以及性能测试。除去单元测试单独编写测试代码并且统计代码覆盖率，其余所有的测试内容均编写测试用例或者测试方案对实际的测试工作进行知道。就需求分析环节的文档规模而言，本小组产出的需求规格说明书在各个小组中处于应当处于首位。

除去测试需求规格说明书以外，本小组还根据实际测试的需要，编写了一些必要的测试前需准备文档，如表4.1所示。

表4.1 测试前需准备文档

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文档名称 | 是否可用 | 是否已接收评审 | 编写者 | 备注 |
| H-EasySpider-软件需求规格说明书v1.3.3 | 是 | 是 | H组全体成员 |  |
| H-EasySpider-测试需求规格说明书 | 是 | 是 | H组全体成员 |  |
| H-EasySpider-实现方案v1.0 | 是 | 是 | H组全体成员 |  |
| 功能需求测试方案 | 是 | 是 | 赵正阳、宋冰晨、郭浩隆 |  |
| 非功能测试方案 | 是 | 是 | 梁远志、沈一聪 |  |
| 性能测试方案 | 是 | 是 | 沈一聪 |  |
| 软件测试计划 | 是 | 是 | H组全体成员 | 宋冰晨整合 |

在测试工作结束之后，为了系统性地对产出软件的测试结果进行分析总结，本小组编写了如表4.2所示的测试后需提交文档。

表4.2 测试完成后需提交文档

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文档名称 | 是否可用 | 是否已接收评审 | 编写者 | 备注 |
| 被测软件执行结果分析报告 | 是 | 是 | H组全体 | 梁远志整合 |
| 软件问题报告 | 是 | 是 | H组全体 |  |
| 性能测试报告 | 是 | 是 | 沈一聪 |  |
| 软件可用性报告 | 是 | 是 | 宋冰晨 |  |
| 单元测试结果 | 是 | 是 | 赵正阳、梁远志、沈一聪 |  |
| 集成测试结果 | 是 | 是 | 赵正阳、梁远志、沈一聪 |  |
| 场景测试结果 | 是 | 是 | 宋冰晨、赵正阳 |  |
| 功能测试结果 | 是 | 是 | H组全体 |  |

上述的所有额外文档也占据了测试工作极大比例的工作量，不亚于测试需求规格说明书本身，事实也证明这些工作是十分有必要的。

在整个项目测试过程中，第一次评审共收到别组评审35项问题，其中严重1，中等4，轻微30；第二次评审共收到别组评审32项问题，其中严重4，中等8，轻微20，总体而言，收到的问题数处于众多小组中的中等偏上水平，但考虑到测试工作的规模，比例上并不算多。测试评审意见依旧主要来源于文档格式问题，但也有少数的意见指出了测试工作的不充分性和不正确。对此，我们虚心采纳了其中的63个意见，部分接收了3个意见，仅仅拒绝了1个意见，并在各个版本的文档迭代过程以及测试工作中，根据评审意见积极修改文档，改进测试工作，大幅度提升了软件产品的质量，修改后的测试文档和软件也都获得了各个评审小组的一致认可。

在最后的互评环节中，本小组的测试工作总计获得15分，仅差一分获得满分，在所有小组中位列第一。在测试环节中，前期的测试准备工作极大的帮助了软件的改进，其中单元测试的覆盖率达到95%以上，其余各项测试的通过率达到99%以上。综上所述，我们认为最终版本的测试文档质量以及对应的测试工作达到了十分优秀的水准。

4.1.3 评审质量

在实验过程中，我们并非仅仅将目光专注于自身的项目，同时也致力于帮助改进其余的小组在各个实验环节中的工作。在评审其他小组项目的过程中，取长补短，积累的经验以及交流得到的共识对自身项目也有着长足的帮助。

在两次需求评审过程中，我们总计向其他小组提出了106个评审意见，包括轻微53，中等26，严重17，其中完全接受93个，部分接受6个，拒绝7个。

在两次测试评审过程中，我们总计向其他小组提出了103个评审意见，包括轻微48，中等38，严重17，其中完全接受88个，部分接受6个，拒绝9个。

总体而言，本小组对他组提出的问题数量在所有组中占据前列，并且接近95%的意见可以被接收或者完全接受。在评审过程中，我们本着沟通和理解的原则与他组在评审意见上求同存异，做到了互利共赢。

在最后的互评环节中，需求评审环节的总计获得11分，与D、F、G组并列第2；测试评审环节总计获得10分，与A、G组并列第4。总体而言，我们认为对其他组的评审工作在整体小组中处于较为优秀的水准。

4.2 软件设计与实现

4.2.1 软件的预期设计

Easyspider项目的最终目标概括来讲，是在允许用户无需学习爬虫技术的前提下，基于Scrapy，提供一个界面美观、使用简单、功能丰富、运行平稳和爬取高效的模板化爬虫程序管理平台。

Easyspider项目主要由Web UI、调度器和爬虫模板三部分组成。用户通过前端界面进行创建爬虫任务等操作，网站后端采用分布式架构，将爬虫任务划分为多个爬虫作业，由调度器程序将这些爬虫作业分配到Scrapyd服务器节点上，再由Scrapyd服务器执行实际的采集工作，最后在实际的爬去过程中引入反爬虫应应对手段从而保证爬虫的平稳运行。网站后端使用的数据库为MySQL，Scrapy爬虫程序保存采集结果的数据库为MongoDB，整体架构如图4.1所示。



图4.1 EasySpider整体架构

4.2.2 软件最终实现评估

在设计与实现过程中，我们严格按照需求规格说明书的要求进行产品的构建和软件的编写，在遇到需求规格说明书中未曾预想到的实际问题时，组员间积极沟通，尽可能巧妙地解决或是绕过相关问题。初步设计完成以后，经过测试、测试评审以及测试复评审的层层洗礼之后，网页项目达到了相当高的可用性，预期目标基本完成。具体的软件成果的形态可以在软件测试结果报告以及最后的项目展示环节得到，这里不再一一赘述。

整个项目研发过程在总计4个仓库中分头进行，并最后在easyspider-web中进行合并，代码规模总计大约为5500行python，4000行html，1100行css以及600行js。另外，Django项目（easyspider-web）共进行了118次commit和72次pr，easyspider-scheduler进行了11次commit以及8次pr，easyspider-site-douban和easyspider-template分别进行了53次commit和4次commit。项目开发过程中的代码规模以及commit次数大致可以作为项目质量的一个客观反映，在这一方面easyspider无疑又走在了所有小组的前列。

4.3 进度计划与控制

4.3.1 实验过程及特点

为了做到对每周计划执行进度的及时监控，原则上项目组每周召开的需要全组参与的组会次数不少于2次，其中，远程组会沟通工具采用腾讯会议。项目每周的进度控制流程如图4.2所示。

图4.2 项目周进度控制流程

项目每周的进度控制流程如下所述：

（1）通常情况下本项目组会在每周的周五召开第1次全组会来讨论确定本周的任务，并分配到个人，分配的原则主要是考虑个人意愿，最后由组长落实为会议记录上传至github。

（2）全组的第2次组会正常情况下会在每周的周一召开，或者时间由第1次全组会确定。第2次组会的主要内容是组员汇报各自工作情况，全组讨论每个汇报结果，根据讨论结果调整本周计划。

（3）每周五13:00前要求项目团队成员采用统一的日志模板填写个人工作日志，并提交至github。13:00之后由项目进度负责人汇总全组的工作日志，使用Microsoft Project项目管理工具形成本周的MPP文件。

4.3.2 实验进度影响因素

在本项目中，主要是实验三出现部分进度滞后，原因是组间的远程沟通不足。例如，由于租借的云服务器对发送邮件的端口有所限制，导致无法实现邮件验证相关功能。前端组同学在完成了邮件验证相关前端页面编写后被告知无法部署邮件功能，最终邮件相关前端工作全部作废，项目实际进度相比计划出现滞后。

4.3.3 进度控制的有效性

为了更好地控制实验三的进度，在邮件验证事件之后，实现组同学和部署负责同学就要实现的全部功能以及实现依赖的第三方组件如channels进行了深度的沟通，敲定了接下来需要实现且可以成功部署到云服务器的所有功能，避免再度出现任何一方的工作作废的情况。

4.4 配置管理

4.4.1 配置管理方法

文档的管理本组按实验要求使用了GitHub的仓库，仓库地址为：<https://github.com/bhsei/20_H>。

本组为文档定义了标准的命名规范，以本文档为例“H-EasySpider-变更与管理分析报告v1.0”，其中“H”是本组组号，“EasySpider”是本组开发项目的名称，“变更与管理分析报告”是本文档的标准名称（参照《软件工程综合实验 概述》），“v1.0”是文档的版本号。

文档的版本号视文档的变更频繁情况（需要评审的文档会更为频繁），对于变更频繁的文档使用三段的版本编号（如“v1.0.1”），其他文档使用两段的版本编号（如“v1.0”）。除了文件名外，文档的封面会注明文档的版本号以及最后修订时间。

每一个文档在封面后会放置一个“版本变更记录”表格，由对文档进行变更的组员进行填写，表格表头如下表所示，包括文档版本号、文档修订日期、修订人姓名、修改的章节范围以及备注。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 日期 | 修改人 | 修改章节 | 备注 |

Git工具能够记录每一次变更的完整信息，因此也能够回退还原某一个特定提交的版本，因此在文件树中仅保留最新的文件版本。

本组在文档方面使用集中的管理方式，在集中编写文档时，会对每一个文档指派一个组员主持编写工作，并进行分工以及整合；在需要对文档进行修订时，会对每一个文档指派一个组员主持修订，在需要多个组员需要对文档进行修改的时候，该主持者会安排一个修订顺序，每一个需要修订的组员在上一个进行修订的组员完成修订后，获取上一个组员修订的版本，进行修订工作后增加版本后上传到GitHub，以此类推。本组并没有对文档设置明确的监督过程。

本组使用Pull Request来进行代码的变更合并，通过设置GitHub的分支保护，拒绝向master分支的直接push，要求所有修改都经过Pull Request流程。并且要求Pull Request合并时至少有另一名成员进行Review。

本组在代码方面使用Pull Request（PR）过程的Review作为监督过程，组员需要对其他组员的PR进行代码审查监督。对于其他代码管理配置上的问题，也指派了负责实验七的组员进行集中的配置和管理。

4.4.2 质量水平分析

总结本组的配置管理的实践，本组的文档管理和代码管理过程具有较高的质量水平。

首先本组的文档仓库具有清晰的结构，文档具有一致的可辨识的文件名，所有的制品文档中均带有版本变更记录表格。这些措施的实践保证了本组的文档管理的质量水平。除此之外，本组文档变更修改使用串行的集中安排的方式也确保了文档管理的效率，和变更一致性。

在代码仓库管理上，本组对于多人协作的项目使用了Pull Request的方法，同时要求每一变更均有其他组员的代码审查。本组总共实施了82次的成功的Pull Request合并，也能够反映本组的代码制品管理的高水平。

1. **有效方法**

5.1 软件需求分析

5.1.1 如何确定软件需求

由于在项目开展初期组员所掌握的技术知识和各自擅长领域不同，因此我们对于软件需求的确定进行了大量的调研，主要从以下几个方面进行：

（1）对现有产品或开源项目进行分析

我们对使用不同技术的现有产品或开源项目的优点和不足之处进行分析，可以得到一些有用的信息，如：如何满足用户对某一功能的需求，有那些没有考虑到的地方，等等。同时我们对不同项目或产品的文档资料进行搜集，其中的需求分析案例也对我们确定软件需求带来了一定的启示。

（2）从用户需求角度出发进行分析

我们从用户需求角度出发，对实际工作等场景中一些功能的需求进行分析，如软件使用上的不便，或在工作学习中对于某些便捷功能的需求等。从用户的角度出发能帮助我们对用户的需求进行更好的分析，而这是对现有产品或开源项目分析所达不到的。

5.1.2 如何描述业务需求

我们使用用例图和RUCM图对业务需求进行描述。我们将系统功能分为若干模块，每个模块使用用例图进行描述，其中包括参与者、用例、边界和用例之间的关系。而RUCM对每个用例进行更细致化的描述，主要包括用例的名称、描述、前置条件、使用者、依赖、约束、流程和后置条件构成。通过以上两种作图和相关的文字描述，我们对每个业务需求都能清晰而直观的描述。

5.2 软件需求评审

5.2.1 如何针对他组的需求文档进行评审

（1）评审分工

由于每次评审需要对其他两个小组的需求文档，因此每次评审将两个小组分配给不同的成员，因此能减轻每位成员的工作量，同时使每位成员更能专注于一个小组的需求文档的评审工作。

（2）指定评审标准

由于不同成员可能会以不同的标准来对需求文档进行评审工作，因此需要制定一个统一评审标准，每位成员在评审工作时依据评审标准来对其他小组的需求文档进行评审。

评审标准主要包括以下几个方面：完整性、一致性、规范性和准确性。涉及需求文档的文字格式和图表或需求用例的完整性和规范性等其他方面。

（3）制定评审表格

为了统一评审格式和与被评审组的沟通，因此我们制定了统一的需求分析评审表，每位成员在评审工作中需要根据具体的问题进行填写。表格中主要包括以下内容：

* 被评审组的项目名称
* 版本号：用于确定评审时的文档版本，以便后续的问题追踪
* 评审日期
* 评审方式：包括线下互评和网上评审
* 问题位置：标注出问题在文档中所在的位置，用于对问题的快速定位
* 报告人：用于明确责任人，以便于被评审组的沟通
* 严重性：用于对问题的严重程度进行确定，包括轻微，一般和严重
* 处理意见：用于对被评审组提出意见，以便其进行快速修改

5.2.2 如何根据收到的评审意见修改项目需求

我们在接收到其他组的评审意见后将每个问题分配给文档中问题所在位置的相关编写人进行问题的解决。同时相关人员需要给出以下几种解决方式：接受，即接受评审意见并进行修改；部分接受，部分接受其意见并对其他部分进行不接受的解释；拒绝，拒绝评审意见并对其进行解释。其中后面两条处理方式需要相关人员与评审人进行沟通，以便消除误会，更好的完成评审工作。

5.3 软件设计与实现

5.3.1 如何确定软件的架构设计与分工

软件架构的设计由一位熟悉整体架构的成员作主导，其他组员参与讨论来进行。

具体过程是根据实验一软件需求分析阶段中划分的模块对软件的架构进行确定，如需求中的Web UI模块对应于软件中的前端等。在对不同模块间的交互进行具体设计。同时为了明确整体架构设计还需要设计若干图表进行描述，包括整体组件图、架构图、流程图、状态图、顺序图和ER图。

在确定架构设计后对每个模块使用的技术和框架进行确定，而框架的确定主要参考不同成员对不同技术和代码的熟悉程度。在确定具体使用的技术后按照每位成员对不同技术的熟悉程度进行分工。如熟悉前端技术的成员对前端进行开发，熟悉后端框架的成员负责后端的开发。

5.3.2 如何在软件设计过程中进行团队交流

今年由于特殊原因小组成员无法进行线下交流，因此团队交流方式主要为线上交流，具体包括腾讯会议和微信。而在每个模块之间的交互遇到问题时需要相关成员使用腾讯会议进行交流，如前后端的对接等。

而项目代码的同步主要使用github来进行，每位成员需要在开发前对代码进行同步操作，在修改完成后提交至自己的分支，并提交一个pull request同时通知其他成员进行审核。在发现代码的问题时需要发现问题的成员在仓库的issue处提出问题与问题的状态，并由相关成员进行处理。

5.3.3 如何对软件设计成品进行展示

软件成品为网页的形式，因此展示形式主要有截图和录屏的形式。

5.4 软件测试

5.4.1 如何设计测试用例

我们按照不用的测试类型将测试用例分为单元测试、集成测试、场景测试、性能测试、功能性需求测试和非功能性需求测试六个类别，其中功能性需求和非功能性需求依照需求文档中的需求用例对测试用例进行设计。而除了需求文档中的需求用例，还增加了不在需求用例中而在设计过程中实现的一些增强用户体验的功能所对应的测试用例，如重命名任务等。

不同测试用例设计按照软件设计与实现阶段对应功能的相关开发成员进行分工。例如：开发用户模块的成员来设计相关功能的测试用例如创建爬虫任务等；而集成测试、场景测试则由熟悉整体过程的成员来设计。

我们设计了统一的测试用例表，以便于统一每位成员的测试用例格式。测试用例具体包括以下内容：

* 测试用例名称
* 测试用例类型：如集成测试、功能性需求测试等
* 测试用例编号：用于标识测试用例
* 测试模块：测试用例对应功能模块
* 模块开发人员：测试用例对应模块的开发人员
* 测试用例编制人：用于明确责任人，便于后续的沟通
* 测试方法：测试的方法，如黑盒测试、白盒测试等
* 测试工具
* 测试前提与约束：明确测试前提
* 测试内容描述：简要描述测试内容
* 测试数据：介绍测试所需要输入的数据
* 测试过程描述：详细描述测试的步骤及每个步骤的预期结果

5.4.2 如何制定高效的测试计划

我们从测试策略、执行策略和测试管理过程几个方面来制定测试计划，每位小组成员都需要严格遵守相应的测试计划。

测试策略即测试所需要遵循的规则，包括项目的给定条件、建立有效测试的过程描述等。

执行策略描述如何执行测试、识别和报告缺陷以及实施修复的过程。

测试管理包括测试的后勤工作以及执行过程中可能发生的事件的管控处理过程。

5.5 软件测试评审

5.5.1 如何针对他组的测试文档和软件进行评审和测试

（1）评审分工

同5.2.1，将两个小组的测试文档与软件分配给不同的成员有助于减轻每位成员的工作量，同时也便于每位成员在之后与被评审组的沟通。

（2）指定评审标准

由于不同成员可能会以不同的标准来对需求文档进行评审工作，因此需要制定一个统一评审标准，每位成员在评审工作时依据评审标准来对其他小组的需求文档进行评审。

评审标准主要包括以下几个方面：完整性、一致性、规范性和准确性。涉及测试需求文档的文字格式和图表或测试需求用例的完整性和规范性等其他方面。

（3）制定评审表格

为了统一评审格式和与被评审组的沟通，因此我们制定了统一的测试需求评审表和软件问题表，每位成员在评审工作中需要根据具体的问题进行填写。

测试需求评审表与需求评审表类似，需要对问题位置、问题描述、报告人、严重性等信息进行详细的描述，以便于明确责任与以及问题的定位与后续的沟通。

软件问题表则与测试用例表类似，需要写清测试方法、问题报告人、测试步骤描述等信息。以便描述软件问题的详细信息。

5.5.2 如何根据收到的评审意见和测试结果改进测试和项目

我们在接收到其他组的评审意见后首先进行小组会议，主要商讨每个软件问题的解决策略以及相应的责任人，之后将测试需求的问题分配给文档中问题所在位置的相关编写人进行问题的解决，而软件问题则分配给相应的开发人员进行解决。

同时相关人员需要给出以下几种解决方式：接受，即接受评审意见并进行修改；部分接受，部分接受其意见并对其他部分进行不接受的解释；拒绝，拒绝评审意见并对其进行解释。其中后面两条处理方式需要相关人员与评审人进行沟通。而对于软件问题，有些暂时无法达到的性能需求或某些主观问题则需要与评审人进行沟通与解释。

5.6 项目计划与监控

5.6.1 如何有效控制项目按照计划进行

为了有效控制项目按照计划执行，所有分配的工作都必须明确负责人和提交截止日期，且明确的结果务必落实到文字如会议记录、评审问题分配文档、即时沟通工具的聊天记录或群公告等等。以测试需求分析相关文档的扩充为例，在周五小组第一次全组会上，全组讨论确定了所有待补充文档的负责人、整合负责人、提交截至日期，最后由组长落实到会议记录文档中。表5.1为项目组会议记录文档节选。

表5.1 项目组会议记录文档节选

|  |  |  |
| --- | --- | --- |
| 时间 | 记录 | 时长 |
| 2020.5.15 | 1.确定增加测试用例的分工  （1）单元测试及覆盖率统计（只列表格，每行一个测试用例）：赵正阳——Django；梁远志——调度器；沈一聪——Scrapy单元测试  （2）集成测试（设计用例表）：赵正阳、沈一聪、梁远志  （3）场景测试（设计用例表）：现有功能性测试用例中任务相关的测试用例原编写者，增加其他组件的预期输出  （4）性能测试、压力测试：沈一聪  （5）测试计划文档（冒烟测试、回归测试）：宋冰晨  （6）非功能测试-可用性测试：宋冰晨  2.评审标准、测试需求分析评审表以及软件问题表的设计：郭浩隆（周二晚之前）  3.整合工作及ddl  （1）赵正阳负责整合测试需求规格说明书，其他人周一晚之前提交  （2）梁远志负责整合测试结果，其他人周二晚之前提交  （3）郭浩隆负责评审意见和软件问题的整合，其他人周三晚之前提交  （4）暂定下周四开会讨论对评审意见的反馈以及汇报ppt | 2 h |

5.7 配置管理

5.7.1 如何合理配置目录结构

本组主要参考往届各个小组中主管感受较好的目录结构设计本组的目录结构。在实验过程中针对使用文档目录过程中发现的问题，调整目录结构。最终形成了如下的目录结构。

|  |
| --- |
| ├───1.会议记录  ├───2.评审意见  ├───3.项目PPT  ├───4.项目提交制品  │ ├───4.0实验准备  │ ├───4.1软件需求分析  │ ├───4.2软件需求评审  │ │ ├───初评审  │ │ └───复评审  │ ├───4.3软件设计与实现  │ ├───4.4软件测试  │ │ ├───1-单元测试  │ │ ├───2-集成测试  │ │ ├───3-场景测试  │ │ ├───4-功能测试  │ │ ├───5-非功能测试  │ │ └───测试用例  │ ├───4.5软件测试评审  │ │ ├───初评审  │ │ └───复评审  │ ├───4.6软件项目计划与监控  │ │ ├───工作日志  │ │ └───项目进度计划（甘特图）  │ ├───4.7软件配置管理  │ ├───4.8软件工程实验追踪与分析  │ └───4.9实验总结  └───5.资料 |

5.7.2 如何管理历史文档

Git工具能够记录每一次变更的完整信息，因此也能够回退还原某一个特定提交的版本，因此在文件树中仅保留最新的文件版本。在上传了新的版本的文档后，该文档会具有新的文件名，同时会删除旧的文件版本。当需要回溯到上一个版本时，使用git reset命令则可以根据需要回退到的版本的取出文档。这一方法使得当前的文档目录结构简洁，同时也能保留回溯版本的能力。

5.7.3 如何设计文档的变更过程

文档变更是一个较为复杂的操作，本组在接收到评审意见后，会依据撰写者对评审的问题进行分工，同时在小组会议中讨论问题以及修订文档的顺序。最终使得能够使用一种串行的方法，每一位组员完成变更后，将文档交给下一个组员。这一过程中的每一步都会提交到GitHub上，同时产生一个文档内的版本变更记录。这一方法极大程度地减少了混乱，且具有很高的效率。

5.7.4 如何设计代码的变更过程

本组在代码工作开始前就确认使用Pull Request机制进行代码变更过程的管理。本组还创建了一个测试的仓库，供组员进行流程的测试。本组在这一机制的使用上并没有遇到太大的困难。在实际使用的过程中，我们也使用了其他基于Pull Request的功能，如版本缺陷管理、自动测试和自动部署。使用这些现成的功能不仅提升了开发的效率，且能够体现本组在代码变更管理上的专业性。

5.8 工程实验追踪与分析

5.8.1 如何进行有效的工作量统计

在3.3节中提到的几类工作量中较难统计的是文档字数和其他工时。

为了统计各组员完成的文档字数，首先要明确每个组员负责的具体章节。一方面，在小组会上确定文档分工后要在写在会议记录中，以便之后参考。另一方面，在整合文档初稿时要在版本变更记录中写明每个组员负责的章节编号，以作为统计工作量的依据。另外，详细完整的版本变更记录也会使统计每个文档的更新版本数变得容易。

其他工时的统计只能参考组员在工作日志中上报的相关工时。

5.8.2 如何根据阶段性工作量统计结果调整分工

应根据阶段性工作量统计结果适当调整工作分配方式。本组在分配实验2的评审问题修改工作时是依据实验1中的文档分工，即每个组员修改自己负责部分的问题。这种分配方式看起来合理，但实际上间接导致了实验2的工作量会受实验1分工的影响。在实验4完成后的阶段性工作量统计中发现实验4的工作量非常不均衡，因此在实验5分配评审问题修改工作时调整了分配策略，尽量按照平均分配的原则，即组员不一定修改自己负责部分的问题。此外，将评审意见整合这种额外工作分配给实验4中工作量较少的组员，有助于总体的工作量保持平衡。

1. **结论与建议**

6.1 实验结论

本小组在各个组员的合作与努力下完成了实验完整流程，通过16周的学习过程大家均有收获，对软件开发的管理有了一定程度的认知。每名成员均对项目付出努力，成员总体的贡献率相对平均。各同学各尽所能，从各自擅长的方向参与软件工程的整个过程。从时间统计数据和实际工作分工可以看出，各同学任务量大致相当，结果满足基本要求。

6.2 课程建议

软件工程实验课程总体而言设计比较完整，尝试将理论和实践相互结合，实验中除了传统的代码实现以外，更多的是把注意力放在软件相关文档的编写以及团队内和团队间的合作交流上。经过一学期的学习，可以说是收益颇丰。对于课程改进的建议主要如下。

（1）增加更多的关于软件工程本身课程内容的一些教学环节，由于软件工程实验课是必修课，但难免有许多同学并没有在本科或者研究生阶段进行过软件工程的系统性学习。

（2）课程的节奏需要更多的调整，避免在开发+测试环节十分繁忙，而课程后期相对空闲的情况。