RUCM项目报告

[项目概述 2](#_Toc532473577)

[软件设计架构 2](#_Toc532473578)

[软件输入输出样例 3](#_Toc532473579)

[输入样例 3](#_Toc532473580)

[输出样例 4](#_Toc532473581)

[项目设计 5](#_Toc532473582)

[类图说明 5](#_Toc532473583)

[规则解析部分 5](#_Toc532473584)

[规则实体部分 6](#_Toc532473585)

[报告部分 7](#_Toc532473586)

[RUCM部分 8](#_Toc532473587)

[自然语言处理部分 11](#_Toc532473588)

[代码实现 11](#_Toc532473589)

[相关技术 11](#_Toc532473590)

[NLP 11](#_Toc532473591)

[BFS 13](#_Toc532473592)

[开发协作流程 13](#_Toc532473593)

[Git 13](#_Toc532473594)

[分模块开发 14](#_Toc532473595)

[问题与解决方法 15](#_Toc532473596)

[nlp server 15](#_Toc532473597)

[部分规则的实现过于复杂 16](#_Toc532473598)

[测试报告 17](#_Toc532473599)

[概述 17](#_Toc532473600)

[测试目的 17](#_Toc532473601)

[需求实现程度 18](#_Toc532473602)

[测试功能点 18](#_Toc532473603)

[测试样例说明 19](#_Toc532473604)

[测试结果统计 20](#_Toc532473605)

[测试用例执行情况 21](#_Toc532473606)

[Bug统计 21](#_Toc532473607)

[附录 24](#_Toc532473608)

[一致性验证 25](#_Toc532473609)

[代码与图之间的一致性 25](#_Toc532473610)

[时序图、活动图与类图之间的一致性 25](#_Toc532473611)

[时序图、活动图与代码的一致性 25](#_Toc532473612)

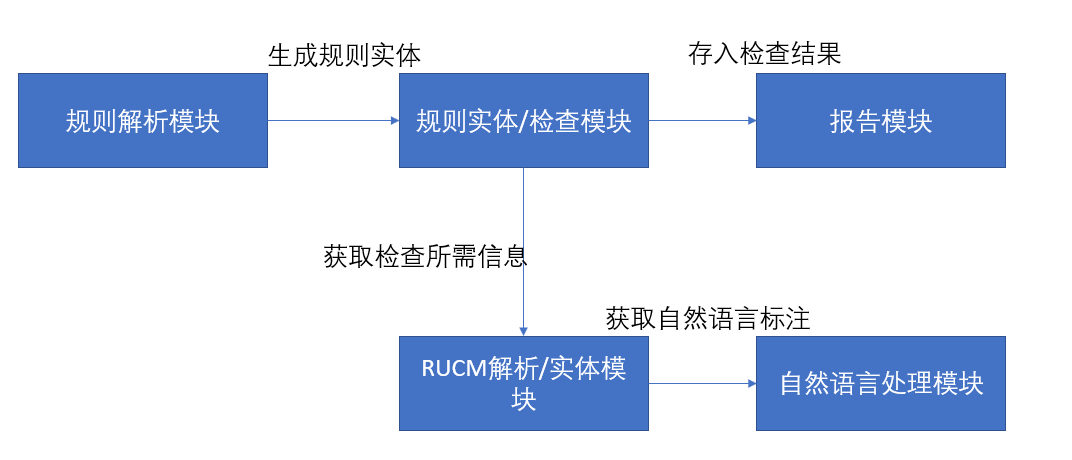
[项目总结 26](#_Toc532473613)

# 项目概述

RUCM是一种结构化和模板化的需求规格，引入了流程、结构化句型和流程控制机制。本项目以RUCM编辑器产生的rucm文件作为输入，依据课堂所讲授的RUCM规范指定相应的规则，并按照规则来自动检查一个具体的需求违反了哪些规则，同时能够支持规则的设置。

具体的介绍请参见领域分析报告。

## 软件设计架构



如图所示，本软件由规则解析模块，规则实体/检查模块，报告模块，RUCM接卸/实体模块，自然语言处理模块5部分组成。

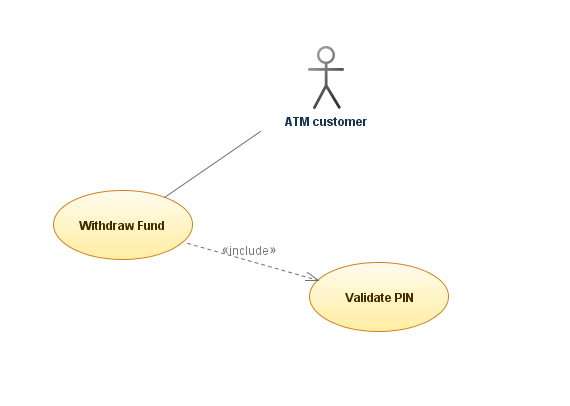
其中**规则解析模块**负责规则格式解析域检查，规则存储等功能；**规则实体/检查模块**由ComplexRule, SimpleRule, DefaultRuleXXX等部分组成，该模块的主要功能是规则的表征与检查；**规则报告模块**负责存储报告信息以及生成报；**RUCM解析/实体模块**负责检查/解析RUCM的json文件，并且将相应字段存入相应实体中并且提供对step/sentence等信息的统一查询接口；**自然语言处理模块**负责句子与词的内容分析以及相应标注。

## 软件输入输出样例

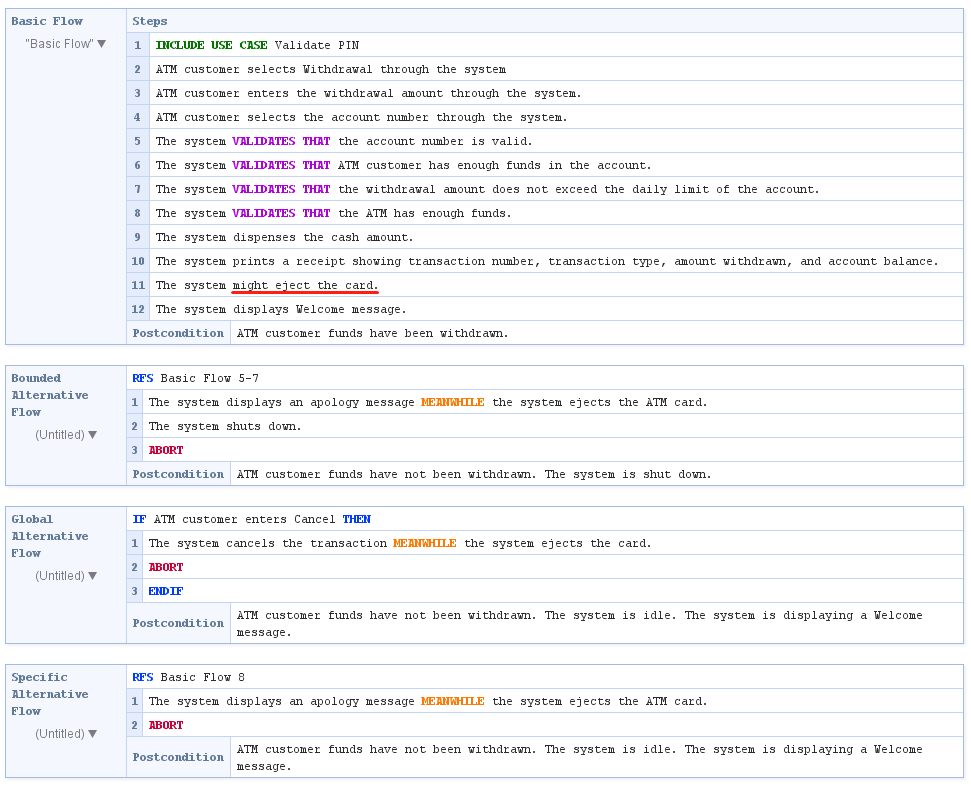
本软件以RUCM设计模型为输入，给出其是否符合RUCM规则的检查报告。

### 输入样例

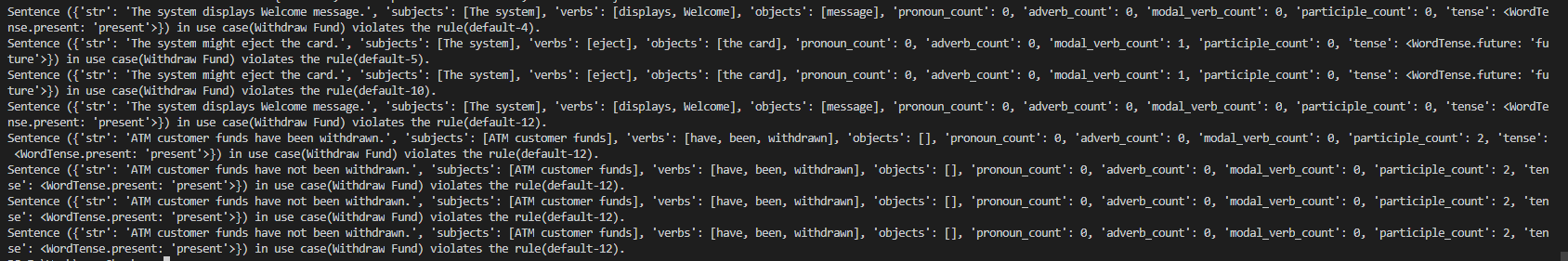
针对下如所示的用例图



我们写出了其用例描述，其中红线标注部分明显违背了RUCM的第5、第10两条规则。

****

### **输出样例**



如图所示，我们的软件检测到了相应的问题并输出了正确的报告。

关于本软件的具体使用方法请参见使用说明文档**document.pdf**。

# 项目设计

## 类图说明

我们的设计主要分为5个部分，分别是规则解析部分，规则实体/检查部分，报告部分，RUCM解析以及自然语言处理部分。接下来分别说明每个部分重要的具体类图设计。

### 规则解析部分

规则解析由类RuleLoader和RuleDB组成，RuleLoader的作用是将规则文件解析成ComplexRule/DefaultRule等类，装入ruleDB中。

* RuleLoader:

load()->bool：进行文件解析。文件解析结果将直接存放在RuleDB中。返回值代表是否解析成功

chekFileFormat(rule:dict)->bool:检查文件格式是否符合要求，包含检查相应的字段是否存在，字段的值是否合法，各个字段之间的关系是否符合约束

parseDefaultRule(rule:dict):解析默认规则

parseComplexRule(rule:dict):解析复合规则字典,具体规则格式详见rule-template.md

parseSimpleRule(rule:dict):解析简单规则

checkComplexRule(rule:dict):检查复合规则格式

checkSimpleRule(rule:dict):检查简单规则格式

* RuleDB:

ruleDB为静态数据类，它根据两部分组成，分别是用户定义规则列表(userRules:list)和默认规则列表(userRules:list)组成

### 规则实体部分

* Rule:

所有的规则都继承自rule基类，Rule的子类包括DefaultRuleXXX，ComplexRule（SimpleRule不是Rule的子类）。

id:rule规则表示

description：规则描述

status：是否启用

rtype:规则的类表，rtype必须取'user','system'之一

* ComplexRule：

一个复合规则可以由多个简单规则的检查结果综合而成

applyScope：规则的作用域，目前作用于可以为rucm中的 action字段或者所有的句子。

simpleRule：简单规则列表

op：对简单规则的综合逻辑操作

* SimpleRule：

一个simpleRule只检查一个句子中的一个字段，它的检查方式可以抽象为某个句子成分在/不在目标列表中。

target:要检查的句子成分，具体取值可以参见rule-template.md

op：最检查目标的逻辑约束可以是in/not in

val：允许/禁止列表

check(setence:Sentence):检查句子

dynamicFill:(s:str):动态填充val，比如填充rucm中的actor

DefaultRuleXXX:

无法被抽象成ComplexRule形式的规则

### 报告部分

* ErrorInfo:

包含检查错误结果的基本信息，包括检查出错误的规则，相应的用例名以及句子。

* Reporter:

静态类，用于生成报告。

### RUCM部分

* RUCMRooT：

存储所有的RUCM信息包括actor列表域usecase列表，并且提供相应get方法方便rule查询信息

usecases:用例列表

actors：actor列表

getActors():获得actor列表

getAllSteps():获得所有step列表

getAllSentences():获得所有的句子

getUseCase(usecasename:str):查找相应的usecase

* Usecase：

对应一个用例

id：用例id

name:usecase的名字

briefDescription:usecase简单描述

preConddition:前置条件

include:该用例所包含的用例

extend：该用例extend的用例

generalization:该用例泛化的用例

basicFlow：用例的基本流

specificFlows:用例分支流

findRfs(flowname:str,stepIndex:int)->bool:判断名字叫flowname的flow中是否存在序号stepIndex的步骤

* Flow:

type：区分basicflow/Specific Flow/Global Alternative Flow

name：flow名称

postcondition：flow的postcondition

steps:构成flow的step

RfsStatement:specific flow的RFS字段

id:flow的id

* step:

一个step可以由多个sentence构成

index：step的序号

val：step的字符串

sentences：step字句

parse\_step():解析句子

* sentence:

一个sentence可以是一个正常的自然语言句子，也可以是一个关键字（IF/ELSE 等）

val：sentence字符串

nature：sentence的关键字类别

* RUCMBase：

所有的RUCM元素的基类，提供向上查找父节点以及所属Usecase的相应属性

word：

val：词的字符串

type：词的类别（noun/adj/verb等）

tense：词的时态（past/present等）

### 自然语言处理部分

该部分由多个方法组成，构成nulputils模块，该部分的主要功能是解析句子/词语成分，并且给予相应的自然语义标注。

# 代码实现

## 相关技术

### NLP

由于规则检查中设计到大量的自然语言处理的部分，我们将其独立开发为一个模块nlputils.py。其对外提供了所有关于自然语言处理的接口。

包括：

parse\_sentence(sentence)，输入参数为一个类型为字符串的句子，输出(返回值)为3个字符串的列表，分别表示句子中的主语、谓语动词和宾语。

get\_verbs\_count\_of\_sentense(sentence)，输入参数为一个类型为字符串的句子，输出(返回值)为4个字符串的列表，分别表示句子中的代词、副词、情态动词、分词。

parse\_sentense\_tense(sentence), 输入参数为一个类型为字符串的句子，输出(返回值)为一个字符串，表示该句子的时态:

present 现在时

past 过去时

future 将来时

none 其他

parse\_word\_tense(sentence), 输入参数为一个类型为字符串的单词，输出(返回值)为一个字符串，表示该单词的时态:

present 现在时

past 过去时

future 将来时

none 其他

parse\_word\_type(sentence), 输入参数为一个类型为字符串的单词，输出(返回值)为一个字符串，表示该单词的类型:

verb 动词

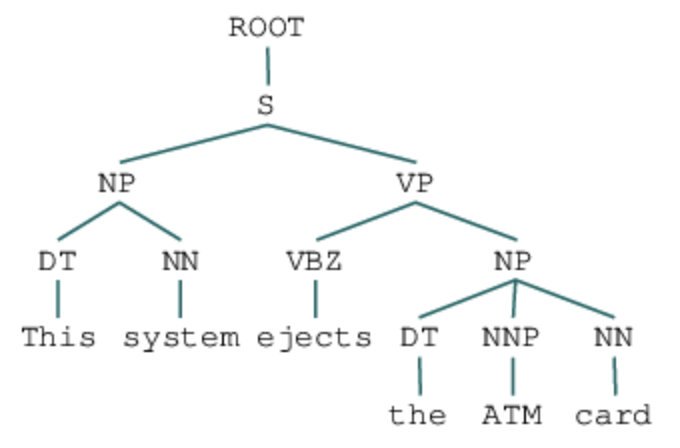
noun 名词

adj 形容词

none 其他

### BFS

在处理句子时，使用nlp工具可以获得句子的每个单词的成分，句子的构成就成了一颗树，如下图所示：



为了遍历到每个单词，我们采用广度优先搜索算法进行遍历。因为这样可以保证相同层的节点被顺序遍历到。

## 开发协作流程

### Git

本项目的开发协作使用git进行版本控制，并将repo放在GitHub上完全开源：

<https://github.com/zen1995/rucmChecker>

将组内每个成员添加为Collaborator, 这样每个就都有权限进行编辑。开发主要以2个分支的形式进行，一个master，一个lby。主要功能在master上开发，检查规则的功能在lby上开发，为了避免最后合并时冲突过多，lby分支在开发过程中会不断merge master上的更新。最后在合并分支时，我们选择集体编程这一模式。所有成员坐在一起，快速处理合并带来的冲突和bug，在2小时内完成合并任务，获得了可以跑的代码。在之后的bug修复、功能添加和测试时，所有的成员都在master上工作，不再使用新的分支。所有的commit和分支的历史信息，都可以在GitHub上找到，方便我们控制代码的所有版本，在必要时进行回退。并且知道每一行代码的修改者，在遇到bug或疑问时可以快速找到源头，进而解决。

### 分模块开发

根据设计，我们将所有的代码分为2大块

1. 规则检查
2. 其他，包括 规则和rucm的加载，错误报告，参数处理

首先，架构师同学先根据类图写好框架，把所有需要的类定义出来，其相应的方法和属性也都写好，参数含义、类型，返回值含义、类型也都通过typing包或者名字、注释规定清楚。然后，程序员再开始分为2波，各自负责一块代码。当所有模块都开发完成后，所有人聚在一起，合并模块和代码，处理冲突和bug。事实证明，这种合作开发方式是十分高效和舒适的。大家各司其职，减少了大量开会和讨论的时间。合并完成后，测试人员进行测试，发现bug后，通过git 的commit信息追踪到产生bug的程序员，将修复bug的任务交付给他。

## 问题与解决方法

### nlp server

nlp函数都依赖于nltk库和一个用于自然语言处理的Java server。开发过程中，最开始我们启动本地的server完成了代码。但在不同程序员完成各自模块，大家开始合并的时候，发现这样的设计并不友好。一方面，有些程序员并不能在本地顺利地配置好Java环境，启动本地server；另一方面，如果最后展示的时候程序还要依赖本地的server的话，还要给用户多加一步配置Java环境，启动server的步骤。经过讨论和商议，改为了在校园网的一台机器上启动这个server，24小时监听，程序默认会访问这个server，同时，给出一个参数--url，即server的链接。允许用户自己配置server。这在对于校园网外的用户和需要自己内部使用server的用户还说也是可以接受的。

### 部分规则的实现过于复杂

在实现和测试过程中，我们发现原有的设计并不能完全实现所有的规则检查。对于一些可以实现的规则，我们选择了修改原有设计，比如，在最开始的设计中，句子这个类并没有代词数量这一属性，但只有有了这个属性，一条规则才能很方便的实现。出现这种情况的原因在于，设计时虽然尽可能考虑所有的规则和情况，但毕竟抽象程度还是高于实现的，所有有些设计并不与实现完全对等。由于原来设计的低耦合性，在修改设计后，代码的修改和更新是十分迅速的。对于另外一些无法实现或没有必要实现的规则，我们选择在说明书中会陈述清楚，暂时不支持这些规则的检查。这样可以快速地生成可以交付给用户使用的版本，对于一些瑕疵选择之后的版本再解决，更加符合敏捷开发的原则。

# 测试报告

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试版本号 | 测试人 | 测试时间 | 测试范围 | 备注 |
| V1.0 | 梁保宇 | 2018/12/6 | 规则检查  文件载入 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## 概述

本次测试的功能点主要在数据读取、规则检查的正确性。检查对象为读取类Loader、规则类Rule、DefaultRule17-26中的各check()方法。

本次测试对应的开发版本为2018年12月5日完成版本。测试环境为Windows 10 + Python3.6环境，NLP模块使用远程NLP处理器测试。

更新：已测试2018年12月7日版本，测试用例同上，主要目的为检查上次测试BUG修复情况。

## 测试目的

本文档为RUCM规则检查项目的规则检查功能的测试报告，从各个方面对测试对象、测试过程进行评估，得出版本质量结论和主要风险。

更新：第二次检查主要目的为检查上次测试BUG修复情况。

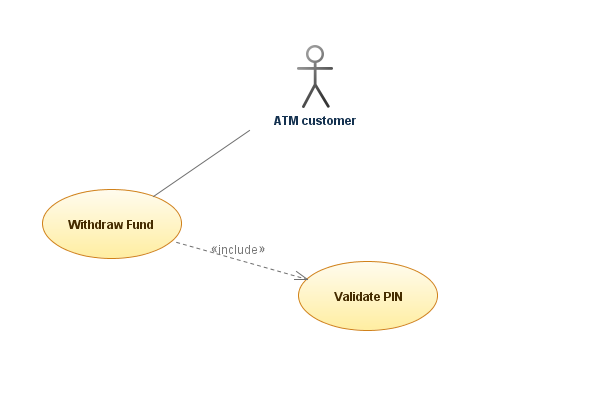
## 需求实现程度

|  |  |  |
| --- | --- | --- |
| **编号** | **需求描述** | **是否实现** |
| 1 | RuleLoader读取规则模板 | 是 |
| 2 | RUCMLoader读取rucm文件 | 是 |
| 3 | 对读取到的rucm文件进行NLP处理，填充RUCM相关各类 | 是 |
| 4 | RuleLoader生成规则，载入规则库 | 是 |
| 5 | Rule规则检查 | 是 |
| 6 | 生成错误报告 | 是 |

## 测试功能点

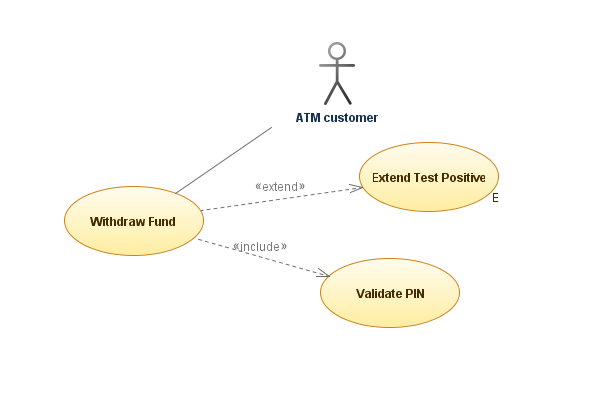
|  |  |  |
| --- | --- | --- |
| **测试类型** | **测试模块** | **测试功能点** |
| 功能测试 | Rule规则检查 | 1、默认规则1测试 |
| 3、默认规则3测试 |
| 4、默认规则4测试 |
| 5、默认规则5测试 |
| 8、默认规则8测试 |
| 10、默认规则10测试 |
| 11、默认规则11测试 |
| 12、默认规则12测试 |
| 13、默认规则13测试 |
| 15、默认规则15测试 |
| 17、默认规则17测试 |
| 18、默认规则18测试 |
| 19、默认规则19测试 |
| 20、默认规则20测试 |
| 21、默认规则21测试 |
| 22、默认规则22测试 |
| 23、默认规则23测试 |
| 24、默认规则24测试 |
| 25、默认规则25测试 |
| 26、默认规则26测试 |
| Loader测试 | 正确样例测试 |

## 测试样例说明



图一、测试采用用例图

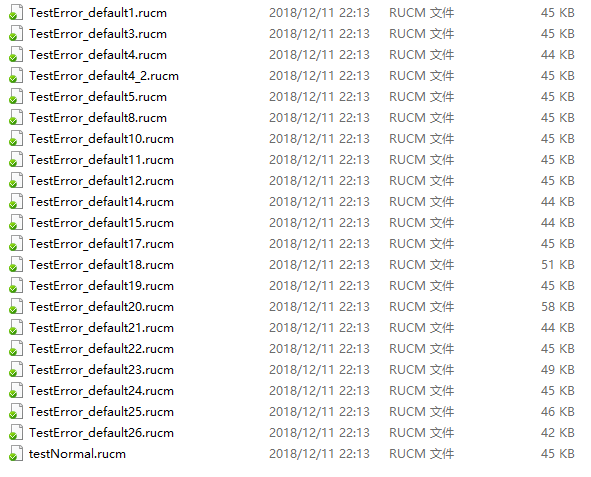
测试用例共22个文件，包括1个无违反RUCM规则的测试样例和21个违反RUCM规则的测试样例。测试样例的整体用例图如图一。Extend关键字规则检查（Default Rule 18）用例图如图二。



图二、Extend关键字规则采用用例图

正确测试用例来源为Restricted Use Case Modeling Approach User Manual中Example of Use – Withdraw Fund板块下的用例模板，含违反规则情况的测试用例来源为Quick Reference Tables板块下的各错误例子。其中，关键字规则的错误用例为自制用例，包含违反关键字规则的若干种情况。

对每个测试用例文件而言（尤其是关键字规则），为保证测试的完整性，针对每条规则可能有多条违反规则的测试用例。正确的测试用例Withdraw Fund内容见附录。所有测试用例文件放置在test文件夹中。



## 测试结果统计

测试人员：梁保宇

测试时间：2018年12月06日——2018年12月07日

测试时间：2018年12月08日——2018年12月08日

### 测试用例执行情况

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例总数 | 22 | | | 备注 | 本次迭代用例 |
| 执行的用例总数 | 22 | 执行情况 | 100% | 备注 |  |
| 通过的用例总数 | 14 | 通过率 | 64% | 备注 | 本次迭代用例 |
| 上版本遗留BUG是否解决完毕 | 第一版本测试  第二版本：部分解决完毕 | | | | |
| 是否满足产品线自身上线标准 | D级  第二版本：B级 | | | | |

版本质量等级划分：

A级：所有功能都已实现，发现的bug都解决。

B级：所有功能都已实现，还有遗留bug，但是有规避措施，不影响用户使用。

C级：主功能已实现，但存在严重bug未修复，有影响用户使用的可能。

D级：主功能未完全实现，或存在非常严重的bug未修复，无法正常使用。

### Bug统计

1. 所有Bug等级分布图

2.不同status下Bug 严重等级分布表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **severity** | **状态** | | | | |
| **Open** | **Reopened** | **Resolved** | **Passed** | **合计：** |
| 紧急 | 0 | 0 | 1 | 0 | 0 |
| 高 | 0 | 0 | 4 | 0 | 0 |
| 中 | 1 | 0 | 5 | 0 | 0 |
| 低 | 1 | 0 | 1 | 0 | 0 |
| **总计：** | **2** | **0** | **11** | **0** | **0** |

注：其中Resolved状态中包含 不可复现 和 转需求分析 状态。

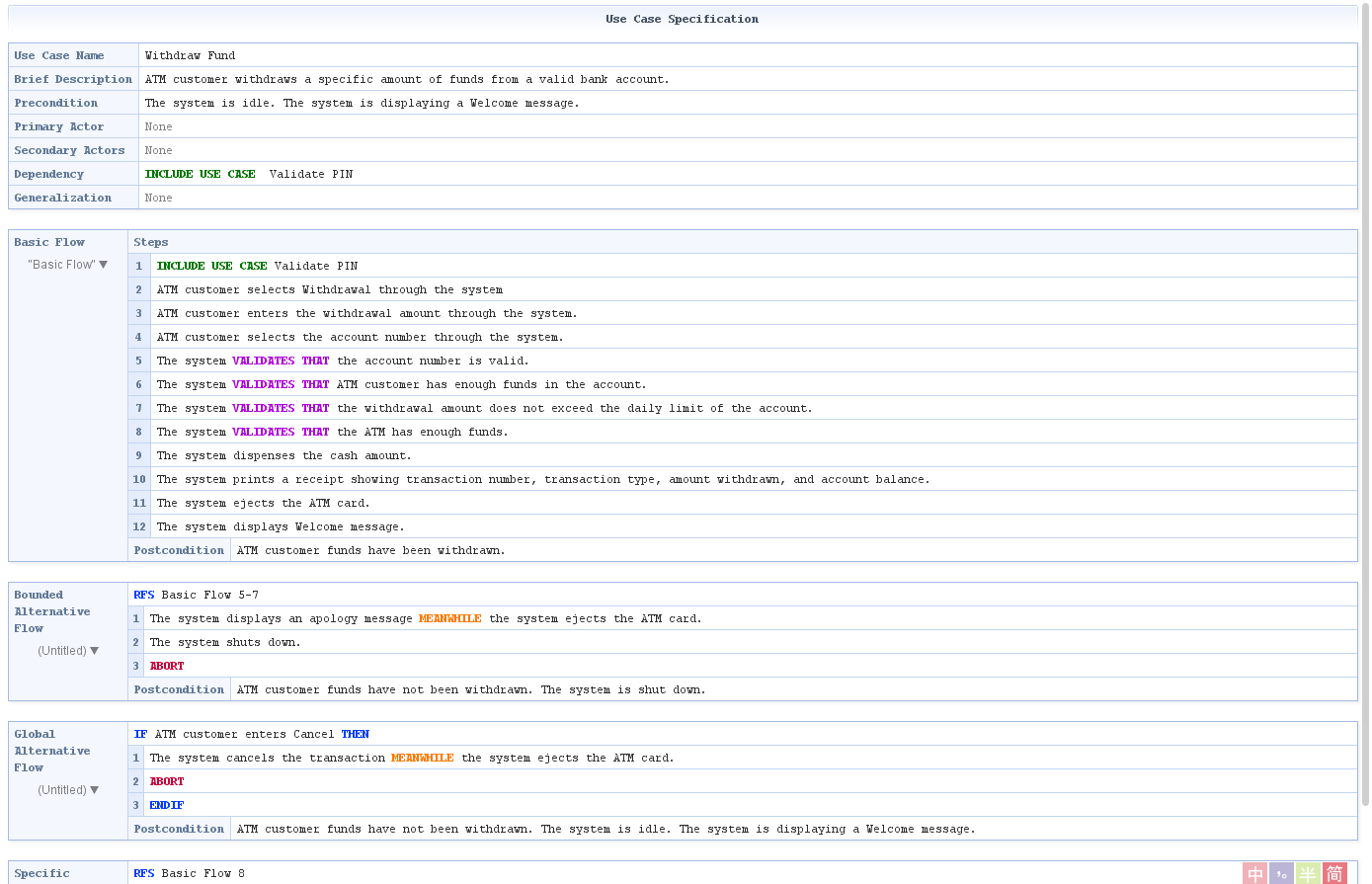
3. 所有Bug所属模块分布图

4. Bug统计

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **BUG等级** | **内容** | **状态** | **出现机率** | **备注** |
| L1 | 紧急 | 不指定规则文件时，程序直接崩溃 | SOLVED | 100% | 本次迭代 |
| L2 | 中 | Global Alternative Flow应没有Rfs字段，但是当前版本把Global Alternative Flow对应的Condition字段装入了Rfs字段，导致在获取Sentence时找不到Global Alternative Flow的对应的Condition | SOLVED | 出现Global Alternative Flow时100% | 本次迭代 |
| L3 | 低 | 解析后Sentence中前后有多余的空格 | SOLVED | 100% | 本次迭代 |
| S1 | 中 | 规则模板，默认规则中第7，12条规则出现没有宾语的情况报违反规则（例如：The system shuts down） | SOLVED | 100% | 本次迭代 |
| S2 | 高 | 规则模板中第8，10，11，13，14条规则无法检测对应规则违反情况 | SOLVED | 100% | 本次迭代 |
| J1 | 高 | 从sentence.words里面取出来的是map，而且只能取一次，取第二次就都取不出来了 | SOLVED | 100% | 本次迭代 |
| S3 | 高 | 规则模板中第三条检测不出错误用例 | SOLVED | 100% | 本次迭代 |
| C1 | 中 | 规则检查中含Nature的句子被过滤 | SOLVED | 100% | 本次迭代 |
| J2 | 中 | 现在分词无法辨析是用来做状语还是定语 | OPEN | 100% | 本次迭代 |
| L4 | 高 | 1-7条规则检查只传入了Basic Flow，没有传入Alternative Flow | SOLVED | 100% | 本次迭代 |
| J3 | 低 | The system displays Welcome message.中检测到两个动词导致违背简单句规则 | OPEN | 100% | 本次迭代 |
| C2 | 中 | defaultrule20产生的错误没有被加入错误库 | SOLVED | 100% | 本次迭代 |
| L5 | 中 | 加入嵌套IF-THEN语句后，INCLUDE USE CASE语句取出来空列表 | SOLVED | 100% | 本次迭代 |

## 附录

测试用例图



# 一致性验证

一致性验证的任务是检查各图之间的名称、关系的一致性以及图与代码之间的一致性检验。

代码与图之间的一致性：

|  |  |  |
| --- | --- | --- |
| 不一致的类 | 不一致情况频率 | 类图修改方案 |
| Flow | 6 | 删除了include、generation、extend 方法及属性，在实现中没有用到 |
| Step | 1 | 添加了parse\_step私有方法 |
| Sentence | 1 | 添加了parseSentence私有方法 |
| NatureYype | 2 | 添加了else\_，删除了重复属性 |
| RUCMLoade | 5 | 修改了返回值，添加了两个私有方法 |
| RuleSubject | 1 | 删除了两个属性，在实现中没有用到 |

## 时序图、活动图与类图之间的一致性

除部分名称从类图中的英文转变为时序图中的中文外，并未发现不一致性。

## 时序图、活动图与代码的一致性

各个图与代码之间基本符合，但是时序图相较于代码过于简略，没有很好的呈现出代码的流程。

# 项目总结

RUCM是一种结构化和模板化的需求规格，引入了流程、结构化句型和流程控制机制。本项目以RUCM编辑器产生的rucm文件作为输入，依据课堂所讲授的RUCM规范指定相应的规则，并按照规则来自动检查一个具体的需求违反了哪些规则，同时能够支持规则的设置。

本项目大致上可以分为四个模块，加载.rucm文件与规则文件模块，自然语言处理模块，检查模块，规则模板设计模块。加载模块基本完成了加载文件的功能，但是尚存在许多不确定因素没有加以测试，在目前的测试样例中，能够完成正确加载文件的功能。自然语言处理模块由于其自身具有不确定性，无法保证完成了解析句子成分的功能，但是在现有测试中，能够对绝大多数输入实现正确的划分，基本实现了功能。检查模块实现了根据规则.rucm文件的功能，在当前测试中表现良好。规则设计模板模块的功能是给用户提供详尽的规则制定模板以及相应的说明书，说明书尚不完善，模板已经提供，还需完善。总体上来说，各个模块都实现了基本的功能，项目基本实现了需求分析阶段中提出的功能性需求，实现与设计阶段的一致性基本满足。在实现过程中，由于各个模块的接口不一致、类的约束不明确等一系列原因，出现了各个模块之间变量命名不一致、功能无法实现等问题。通过返回设计阶段，完善设计，解决了这些问题。