

RUCM 项目报告

项目概述.....	2
项目设计.....	3
类图说明.....	3
规则解析部分.....	3
规则实体部分.....	4
报告部分.....	5
RUCM 部分.....	6
代码实现.....	9
相关技术.....	9
NLP.....	9
BFS.....	10
开发协作流程.....	11
Git.....	11
分模块开发.....	12
问题与解决方法.....	13
nlp server.....	13
部分规则的实现过于复杂.....	13
测试报告.....	15
概述.....	15
测试目的.....	15
需求实现程度.....	16
测试功能点.....	16
测试样例说明.....	17
测试结果统计.....	18
测试用例执行情况.....	19
Bug 统计.....	19
附录.....	22
一致性验证.....	23
代码与图之间的一致性.....	23
时序图、活动图与类图之间的一致性.....	23
时序图、活动图与代码的一致性.....	23
项目总结.....	24

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。

项目设计

类图说明

我们的设计主要分为 4 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 可以是一个自然的自然语言句子，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）

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_sentence(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_sentence_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, 表示该单词的状态:

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

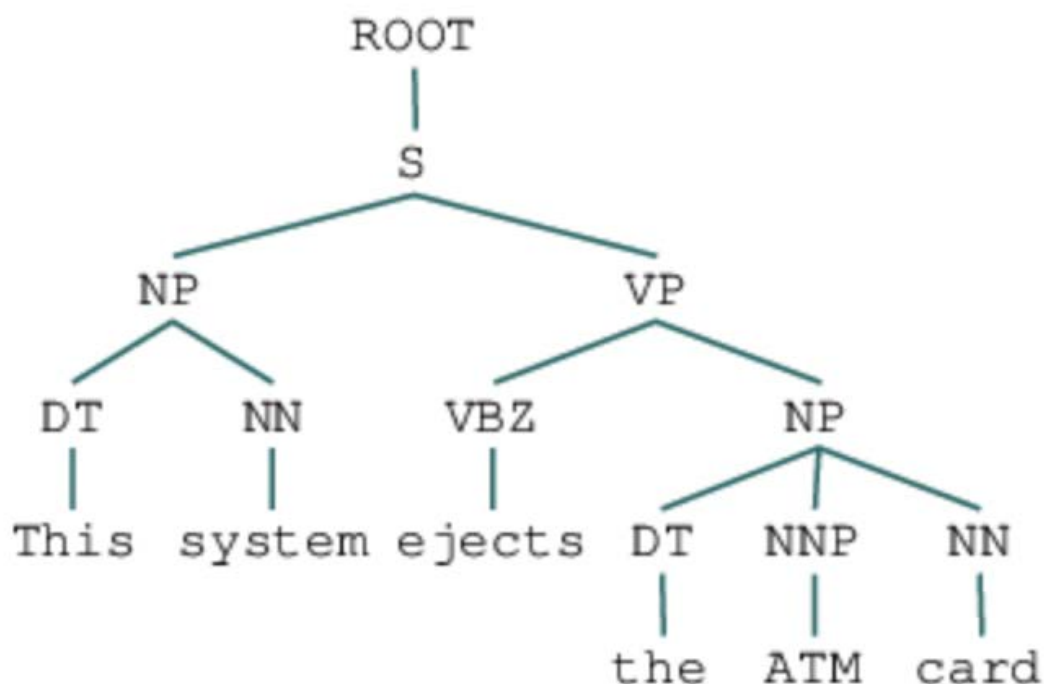
noun 名词

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。

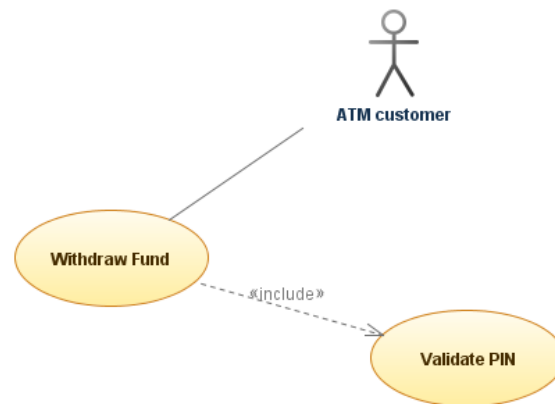
需求实现程度

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

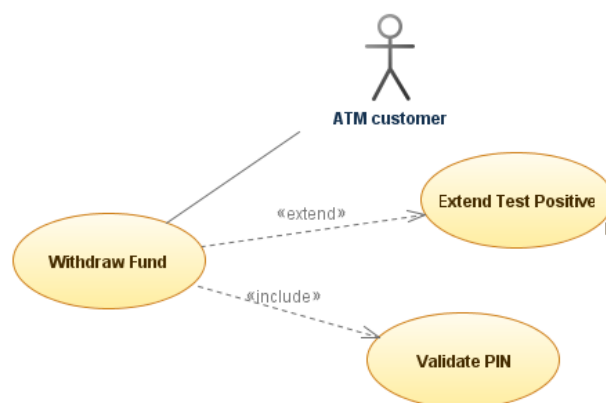
测试类型	测试模块	测试功能点
功能测试	Rule 规则检查	1、默认规则 1 测试
		3、默认规则 3 测试
		4、默认规则 4 测试
		5、默认规则 5 测试
		8、默认规则 8 测试
		10、默认规则 10 测试
		11、默认规则 11 测试
		12、默认规则 12 测试
		13、默认规则 13 测试
		15、默认规则 15 测试
		17、默认规则 17 测试
		18、默认规则 18 测试
		19、默认规则 19 测试
		20、默认规则 20 测试
		21、默认规则 21 测试
		22、默认规则 22 测试
		23、默认规则 23 测试
		24、默认规则 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。

 TestError_default1.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default3.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default4.rucm	2018/12/11 22:13	RUCM 文件	44 KB
 TestError_default4_2.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default5.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default8.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default10.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default11.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default12.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default14.rucm	2018/12/11 22:13	RUCM 文件	44 KB
 TestError_default15.rucm	2018/12/11 22:13	RUCM 文件	44 KB
 TestError_default17.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default18.rucm	2018/12/11 22:13	RUCM 文件	51 KB
 TestError_default19.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default20.rucm	2018/12/11 22:13	RUCM 文件	58 KB
 TestError_default21.rucm	2018/12/11 22:13	RUCM 文件	44 KB
 TestError_default22.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default23.rucm	2018/12/11 22:13	RUCM 文件	49 KB
 TestError_default24.rucm	2018/12/11 22:13	RUCM 文件	45 KB
 TestError_default25.rucm	2018/12/11 22:13	RUCM 文件	46 KB
 TestError_default26.rucm	2018/12/11 22:13	RUCM 文件	42 KB
 testNormal.rucm	2018/12/11 22:13	RUCM 文件	45 KB

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	第一版本测试				
是否解决完毕	第二版本：部分解决完毕				
是否满足产品线	D 级				
自身上线标准	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。

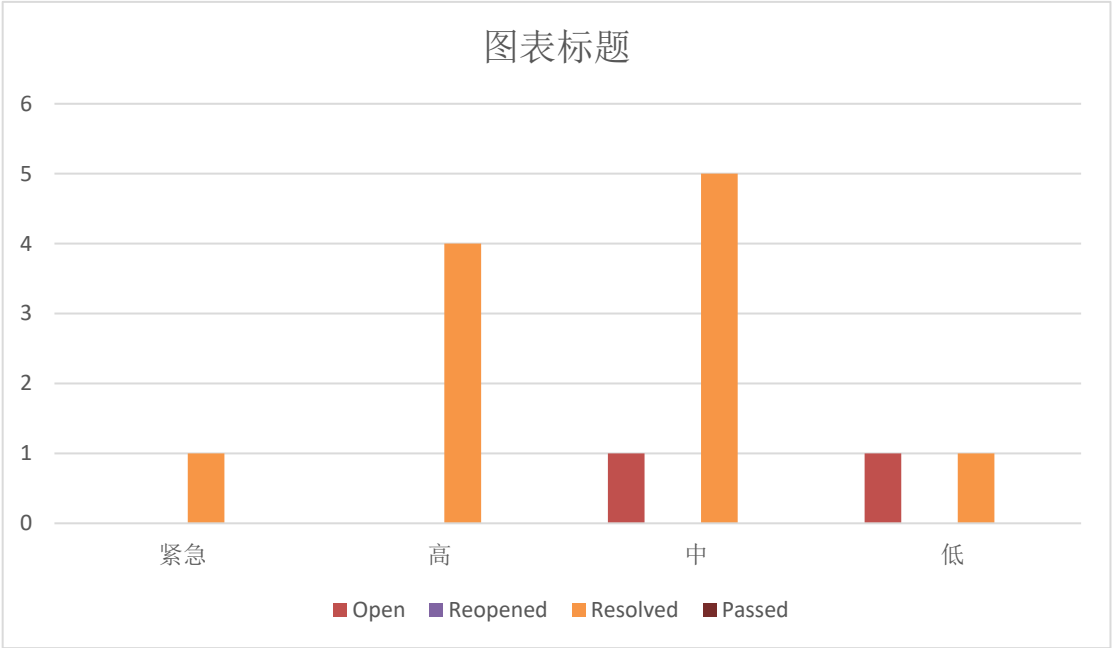
B 级：所有功能都已实现，还有遗留 bug，但是有规避措施，不影响用户使用。

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

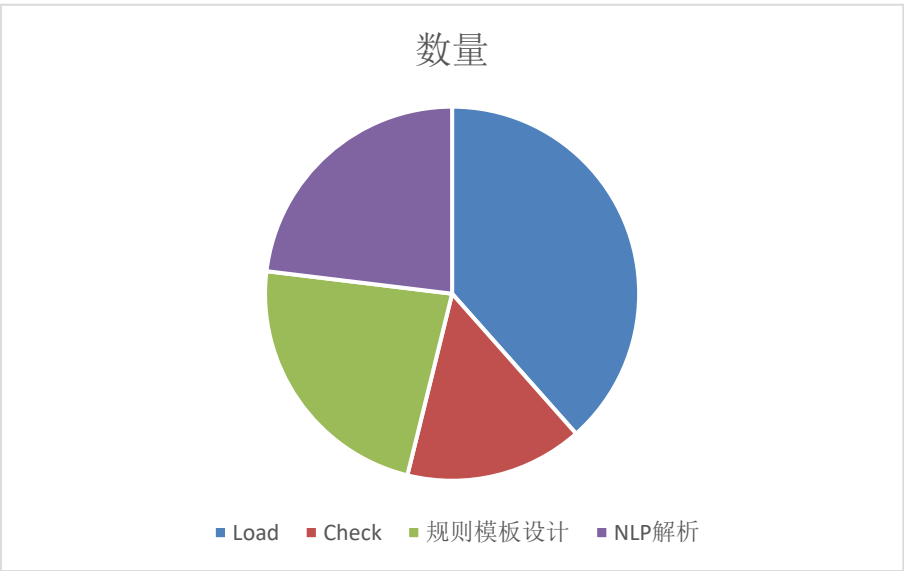


2. 不同 status 下 Bug 严重等级分布表

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

Use Case Specification	
Use Case Name	Withdraw Fund
Brief Description	ATM customer withdraws a specific amount of funds from a valid bank account.
Precondition	The system is idle. The system is displaying a Welcome message.
Primary Actor	None
Secondary Actors	None
Dependency	INCLUDE USE CASE Validate PIN
Generalization	None
Basic Flow	<div>Steps</div> <div>1 INCLUDE USE CASE Validate PIN</div> <div>2 ATM customer selects Withdrawal through the system</div> <div>3 ATM customer enters the withdrawal amount through the system.</div> <div>4 ATM customer selects the account number through the system.</div> <div>5 The system VALIDATES THAT the account number is valid.</div> <div>6 The system VALIDATES THAT ATM customer has enough funds in the account.</div> <div>7 The system VALIDATES THAT the withdrawal amount does not exceed the daily limit of the account.</div> <div>8 The system VALIDATES THAT the ATM has enough funds.</div> <div>9 The system dispenses the cash amount.</div> <div>10 The system prints a receipt showing transaction number, transaction type, amount withdrawn, and account balance.</div> <div>11 The system ejects the ATM card.</div> <div>12 The system displays Welcome message.</div> <div>Postcondition ATM customer funds have been withdrawn.</div>
Bounded Alternative Flow	<div>DFS Basic Flow 5-7</div> <div>1 The system displays an apology message MEANWHILE the system ejects the ATM card.</div> <div>2 The system shuts down.</div> <div>3 ABORT</div> <div>Postcondition ATM customer funds have not been withdrawn. The system is shut down.</div>
Global Alternative Flow	<div>IF ATM customer enters Cancel THEN</div> <div>1 The system cancels the transaction MEANWHILE the system ejects the ATM card.</div> <div>2 ABORT</div> <div>3 END-IF</div> <div>Postcondition ATM customer funds have not been withdrawn. The system is idle. The system is displaying a Welcome message.</div>
Specific	DFS Basic Flow 6

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
NatureType	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。