第三单元课程作业分析

OO2018课程

计算机学院

从“有一点感觉、但仍形象化的设计思维”
向“可重复、规格化的设计思维”转变

内容提纲

- 作业训练要点分析
- 本次作业

作业9训练要点分析

- 熟悉过程规格的内涵和书写
- 初步实践基于规格的方法实现
- 实践基于异常处理的防御编程
- 初步实践基于过程规格的测试设计

作业9训练要点分析

- 熟悉过程规格的内涵和书写
  - 过程规格是一个方法与其用户交互的契约
    - 契约：权利+义务+注意事项
    - 义务：用户要保证提供有效的输入以及有效的系统状态和对象状态
    - 权利：用户能够获得满足确定条件的输出结果
    - 注意事项：方法执行过程中可能会修改用户对象状态
  - 过程规格是一个方法对实现者做出的规约要求
    - 规约：前置条件+后置条件+副作用
    - 前置条件：要求输入满足的条件（不只是数据类型）
    - 后置条件：执行结果必须满足的条件
    - 副作用：执行过程最多能够修改哪些对象变量
  - 过程规格是对方法功能的严格抽象

作业9训练要点分析

- 熟悉过程规格的内涵和书写
  - 标题：语法层面完整准确描述方法信息：可见性+返回值+方法名+参数+[抛出异常列表]
  - Requires + Effects：可判定的逻辑陈述
  - Modifies：用户可见的变量列表
  - 两种常用的Effects逻辑陈述写法
    - 基于过程的逻辑陈述(不推荐)
      - 构造A，修改B，**使得this == \old(\this) +f(A)**…
    - **基于状态的逻辑陈述(推荐)**
      - **g(x)==>f(x)**
      - x指可见的非局部变量，f(x)和g(x)是逻辑可判定的命题/谓词
  - **\result**是着眼点
    - 不管如何得到**\result**，而是规定**\result**必须满足的约束

作业9训练要点分析

- 熟悉过程规格的内涵和书写
  - public Vector<String> scan4subs(String dir)
  - /**@requires: Files.isDirectory(dir) == true
  - *@modifies: none
  - *@effects1: 构造字符串向量(vector)，循环扫描dir下所有文件，并将扫描到的文件名称插入到字符串向量中，最后返回字符串向量
  - *@effects2: 构造用于返回的字符串向量(vector)，扫描dir下所有文件并把文件名插入到字符串向量中，使得向量规模等于dir下所扫描的文件数
  - *@effects: \all String p; \result.contains(p) ==> p.substring(dir).equals(dir) && Files.isFile(p) */

作业9训练要点分析

- 熟悉过程规格的内涵和书写
  - 后置条件的概括能力是关键
    - 概括方法在各种输入情况下的执行效果
  - 例：三角形判定方法
  - public int isTriangle(Point p1, Point p2, Point p3)
    - 返回值-1：不是三角形；0:正三角形；1：直角三角形；2:其他类型三角形。
  - /*@requires: p1.valid && p2.valid && p3.valid
  - @Modifies: none
  - @Effects: ?
  - */

\result == -1==>isLine(p1,p2,p3);
\result == 0==>isRegular(p1,p2,p3);
\result == 1==>isRight(p1,p2,p3) ;
\result == 2 otherwise

作业9训练要点分析

- 后置条件基本可以分成三种情况
  - 只对\result进行约束
    - \result取值一般不难约束，难的是==>的右半部分(RHS)
    - f(\result) ==> RHS
  - 需要对比或约束\this和\old(\this)
    - 意味Modifies不为空
    - 正向逻辑：对\old(\this)的修改必须满足的约束
    - 反向逻辑：不能对\old(\this)做的修改
  - 异常情况
    - exceptional_behavior(推荐)
    - 特殊的\result(不推荐)

作业9训练要点分析

- **warranty**方法
  - 本质上是设计问题
  - Step1(**w**hy): 为什么需要这个方法？==》要它提供什么结果？
  - Step2(**a**cceptance criteria): 这个方法所提供结果正确的判定条件是什么？
  - Step3(clea**r** **r**equire): 这个方法是否需要对调用者做出一些要求，从而确保能够产生正确结果？
  - Step4(**an**ticipated changes): 这个方法执行期间是否需要修改输入数据或者所在对象数据？
  - Step5(**t**rust**y**): 使用JSF来规范整理结果

作业9训练要点分析

- 电梯系统，在写ALS调度时需要访问请求队列来确定是否有可捎带的请求。这时有两种做法：
  - 让请求队列提供一个方法，输入电梯的状态，返回可捎带的多个请求；
  - 让电梯类提供一个方法，输入一个请求，返回是否可捎带的判定结果。
- 针对这两种做法来整理过程规格，五个步骤来处理

作业9训练要点分析

- Queue: Request[] Pickup(Elevator e, Request curReq)
- Elevator: boolean Pickup(Request r)
- Step 1: Queue:Pickup根据e的状态来扫描队列中满足捎带条件的请求，如果有则返回出去，并从队列中移除。
- Elevator.Pickup根据请求r来判断是否能捎带，如果能则返回true，并存入可捎带处理的小队列；否则返回false。
- Step2：Queue.Pickup执行正确的判定条件是返回的每个request对于e的当前状态而言，都**满足捎带条件**。
- Elevator.Pickup执行正确的判定条件是如果传入的请求**满足捎带条件**，并返回true；否则返回false。
- Step3：Queue.Pickup对调用者无要求
- Elevator.Pickup对调用者无要求
- Step4：Queue.Pickup执行期间只能修改this
- Elevator.Pickup执行期间只能修改this
- Step5：使用JSF来整理规格。
- **Queue:Pickup： /*@modifies: this; @Effects: \all Request r; \result.contains(r) ==> pickupable(e, r) && \old(\this).contains(r) && \this.contains(r)==false */**
- **Elevator:Pickup: /*@modifies: this; @effects: \result == false ==> (r==null) || pickupable(\this, r)==false; \result ==true ==> pickupable(\this, r)*/**

作业9训练要点分析

- 初步实践基于规格的方法实现：反向思维
  - 理解后置条件：这个方法承诺了什么？要给出什么输出？输出必须要满足什么条件？
  - 需要的处理步骤：如何基于输入来构造相应的输出？
    - 算法（流程）？
    - 使用哪些局部变量来表示和处理中间结果？
  - 输入有哪些情况？
    - 输入的要求是什么？
    - 是否需要检查输入对要求的满足情况？
    - 是否需要对输入进行类别划分？
    - 不同类别的输入对处理步骤有什么影响？

作业9训练要点分析

- 初步实践基于规格的方法实现：反向思维
  - public List<Node> Shortestpath(Node start, Node end)
  - /*@requires: this.contains(start) && this.contains(end)
  - @modifies: this
  - @effects: (\result!=null) ==> (\result[0] == start) && (\result[\result.length-1]==end) && (\all int i; 0<=i<\result.length-1; this.connects(\result[i], \result[i+1])) && (\all List<Node> p; p[0]==start && p[p.length-1]==end && \all int j; 0<=j<p.length-1; this.connects(p[j], p[j+1]) && p.length >=\result.length)*/
- 如何实现？

作业9训练要点分析

- 实践基于异常处理的防御编程
  - 任何一个方法都需要主动去规划有哪些异常
    - 即导致自己的计算无法按照“正常流程”进行，从而不能满足后置条件的状况
  - 通过显式的方式在规格中声明相应的异常
    - 标题声明（编译器要求）
    - 后置条件归纳（把不能满足后置条件的情况也纳入到“能够满足”）
  - 捕捉异常状况的发生
    - 对输入条件和对象状态的判断
    - 通过try机制来捕捉被调用方法抛出的异常(Exception)
    - 通过try机制来捕捉虚拟机或运行平台抛出的异常(Runtime Exception)
  - 异常的处理
    - 记录异常信息
    - 屏蔽局部异常
    - 反射局部异常

作业9训练要点分析

- 实践基于异常处理的防御编程
  - 从契约设计角度，方法只需要按照后置条件抛出相应的异常
  - 从契约使用者角度，需要了解被调用方法所可能抛出异常的产生场景和条件
  - 从契约使用者角度，需要评估被调用方法所可能抛出异常对自己处理流程的影响
    - 统一处理：无实质性影响
    - 分类处理：影响需要区分
    - 反射处理：对更上层的用户也可能产生影响

作业9训练要点分析

- 初步实践基于过程规格的测试设计
  - 测试输入划分
    - 满足前置条件
      - 等价类划分
      - 边界值选取
    - 不满足前置条件
  - 返回结果判定
    - pass: 满足后置条件
    - fail: 不满足后置条件

```
public int isTriangle(Point p1, Point p2, Point p3)
/*@requires: p1, p2 and p3 are valid points according to the X-Y axis
   @Effects: \result == -1 ==> !isSame(p1,p2,p3) || isLine(p1,p2,p3);
                \result == 0 ==> isReg (p1,p2,p3);
                 \result == 1 ==> isRight(p1,p2,p3);
                 \result == 2 ==> !isSame(p1,p2,p3) && !isLine(p1,p2,p3)
&& !isReg(p1, p2, p3) && !isRight(p1, p2, p3)
```

\result==-1这个情况是否可以优化？

作业10训练要点分析

- 熟悉类规格的内涵和书写
- 实践基于规格的类实现
- 实践契约式设计方法
- 初步实践基于类规格的测试设计

作业10训练要点分析

- 熟悉类规格的内涵和书写
  - 类是一个编程单位，用户一般把类作为一个整体来使用或重用
  - 类规格定义了与用户的契约
    - 数据管理：确保数据有效
    - 提供的操作：确保在满足前置条件下，后置条件得到满足
  - 类规格定义了开发人员必须实现的规约(实现人员直接看到的规格)
    - 表示对象必须映射到抽象对象(抽象函数)：满足用户契约
    - 任意一个方法的实现都不能破坏表示对象对不变式的满足性
      - 从而确保对象整体有效
    - 任意一个方法的实现都要满足方法本身定义的规约

作业10训练要点分析

- 熟悉类规格的内涵和书写
  - Overview的撰写
    - 类整体作用的概述
    - 类所管理抽象数据的概述（不是逐个解释表示对象数据）
    - 类核心功能的概述（不是逐个解释操作）
  - 从用户角度来看，一个设计优良的类应保证所提供的操作
    - 完整性：应提供四类操作（构造、更新、查询和生成）
    - 紧凑性：不提供与所管理数据无关的操作行为
    - 便捷性：能够以简单的方式访问和更新所管理的数据
    - 安全性：阻止不受控访问，确保线程安全
    - 正确性：实现与契约一致

作业10训练要点分析

- 熟悉类规格的内涵和书写
  - 抽象函数和不变式由于涉及表示对象，用户看不到，也不关心
  - 从设计实现的角度，在定义了表示对象之后，一定要首先书写这两部分，然后实现方法
  - 抽象函数用来回答“这些表示对象为什么是必须的”？
    - 有一些表示对象无法映射到抽象对象
      - 为了访问和处理效率
      - 为了定义不变式
      - 不需要的冗余

作业10训练要点分析

- 实践基于类规格的实现
  - 搜索引擎需要管理中间的索引结果，便于快速查找数据
    - 使用倒排表，这是一个抽象对象
      - 静态列表
      - 单向链表
      - 静态列表项是单向链表的头
  - **表示对象**的识别与定义
    - 依据抽象对象：分析其中涉及的数据实体及其关系
    - 如何定义倒排表的表示对象？
      - 静态列表：Key[] keys
        - Key: String keyword; DocList docs;
      - 单向链表：DocList list
        - DocList: int docID; int pos; String left; String right; **DocList next**

作业10训练要点分析

- 实践基于类规格的实现
  - 表示对象的访问效率是难点，考验设计和算法功力
    - 如何在静态列表中快速查找一个String key?
    - 如果管理的keys规模巨大，如何处理？
    - 如果一个文档中多次出现一个String key，如何快速查找？
    - 如果单向链表长度很大，如何快速访问？

Key[] keys
Key: String keyword; DocList docs;
DocList list
DocList: int docID; int pos; String left; String right; **DocList next**

作业10训练要点分析

- 实践基于类规格的实现
  - 表示对象在一定程度上决定了更新方法和查询方法的实现技术和效率
    - 静态列表： Key[] keys
      - Key: String keyword; DocList docs;
    - 单向链表： DocList list
      - DocList: int docID; int pos; String left; String right; DocList next
    - void addOccurrence(String key, int docID, int pos, String left, String right)
    - /**@requires: key不为空(empty), docID是有效的文档ID, pos要在ID为docID的文档内容范围内, left和right是有效字符串且长度不超过两个单词。
    - *@modifies: this
    - *@effects: key被纳入到可搜索的关键词列表中， key在docID文档pos位置处的出现被纳入到list(DocList)中进行管理，且拥有left和right作为上下文字符串。*/

如何实现这个addOccurrence方法？

作业10训练要点分析

- 实践契约式设计方法
  - 契约式设计是一种基于信任机制+权利义务均衡机制的设计方法学
  - 信任机制
    - 类提供者信任使用者能够确保所有方法的前置条件都能被满足
    - 类使用者信任设计者能够有效管理相应的数据和访问安全
  - 权利义务均衡机制
    - 任何一个类都要能够存储和管理抽象对象对应的具体表示对象（义务）
    - 任何一个类都要保证对象始终保持有效（义务）
    - 任何一个类都可以拒绝为不满足前置条件的输入提供服务（权利）
      - 通过异常来提醒使用者
    - 任何一个类都可以选择自己的表示对象而不受外部约束（权利）

作业10训练要点分析

- 实践契约式设计方法
  - 契约式设计是一种致力于保证程序正确性的方法
    - 正确性：在规定的输入范畴内给出满足规定要求的输出
      - “你”如果能够保证前置条件成立，“我”就能保证后置条件成立
    - 正确性基础
      - 契约及其交互满足需求
      - 实现满足契约
  - 契约式设计的核心
    - 方法的前置条件
    - 方法的后置条件
    - 对象的不变式

作业10训练要点分析

- 实践契约式设计方法
  - 契约及其交互满足需求
    - 目前并没有可靠的办法
    - 把需求按照尽可能严谨的逻辑进行整理
      - 对用户输入的要求
      - 对用户期望结果的确认
      - 功能执行过程中系统必须保证满足的约束
    - ==》更容易与契约建立相对明确的对应关系
  - 例：提取每个文档中出现的所有单词，并使用**倒排表**管理提取到的单词、单词所出现的文档、单词在文档中的出现位置。
    - 用户输入要求：文本文档、单词符合规定
    - 用户期望结果：单词、文档、出现位置、上下文单词能够使用倒排表管理起来
    - 执行过程中要满足的约束：文档不能被修改、倒排表中数据不能被删除、倒排表结构保持有效

作业10训练要点分析

- 实践契约式设计方法
  - 实现满足契约
    - 此时可假设契约是*合适的*，即满足需求
    - 检查表示对象是否满足抽象对象的存储和表示要求
      - 应用抽象函数
    - 检查方法的所有返回或结束点
      - 前置条件满足时，后置条件是不是都满足
      - 前置条件不满足时，是否提示用户
    - 检查对象的repOK方法是否与不变式逻辑一致
  - 输入检查疑问
    - 对待可信任对方：不检查其提供的输入
    - 对待不可信任对方：检查其提供的输入

防御式编程强调识别异常和防御，致力于程序的鲁棒性。
契约式设计方法强调逻辑的严谨性，致力于程序的正确性。

作业10训练要点分析

- 初步实践基于类规格的测试设计
  - 针对表示对象，人为构造不满足不变式的状态，检查repOK是否能够发现
  - 根据方法的前置条件，设置对象的状态
    - 调用repOK
    - 调用相应方法
    - **检查方法返回结果是否满足后置条件**
    - 调用repOK
  - 是否有可能让第三步也程序化检查？
    - 为类的每个方法独立实现一个专用于测试的***effectsOK方法
      - 一般化方法
    - 专门建立一个(输入,期望输出)配对表，拿到实际输出时与期望输出进行对比
      - 枚举式方法

第11次作业训练要点

- 实践类型层次下的规格设计

第11次作业训练要点

- 实践类型层次下的规格设计
  - 抽象是OO的一个基本机制，可以形成类型层次
  - 用户在使用层次化的类时倾向于进行一般化处理
    - 高层类型必须对低层类型具有概括性
      - 抽象对象的概括性
        - 子类抽象函数与父类抽象函数结果的关系
        - 子类对象不变式与父类对象不变式的关系
      - 行为的概括性
        - 父类方法规格与子类方法规格之间的关系
    - 凡是使用高层类型引用的对象都可以替换成低层类型对象，且保证程序行为不会变化（LSP）

第11次作业训练要点

- 实践类型层次下的规格设计
  - 从规格的表示角度，子类与父类独立
  - 从规格的语义角度，层次之间具有严格的逻辑关系
    - 子类的不变式与父类的不变式
      - I_Sub(c) implies I_Super(c): c为Sub类型的对象
      - I_Super(c) does not imply I_Sub(c): c为Super或者Sub类型的对象
    - 子类重载方法的前置条件与父类方法的前置条件
      - Requires(Super::f) implies Requires(sub::f)
      - Requires(Sub::f) does not imply Requires(Super::f)
    - 子类重载方法的后置条件与父类方法的后置条件
      - Effects(Sub::f) implies Effects(Super::f)
      - Effects(Super::f) does not imply Effects(Sub::f)

第11次作业训练要点

- 实践类型层次下的规格设计
  - 子类对象正确性与父类对象正确性之间的关系
    - (Requires_super && Effects_sub) ***implies*** Effects_super
    - 子类型方法可以减弱父类型方法规定的Requires,或者加强父类型方法规定的Effects
  - 从契约的角度来理解
    - 父类的承诺，子类不能减弱
      - 有效性承诺
      - 后置条件承诺
    - 父类的要求，子类不能加强
      - 前置条件要求

典型问题分析

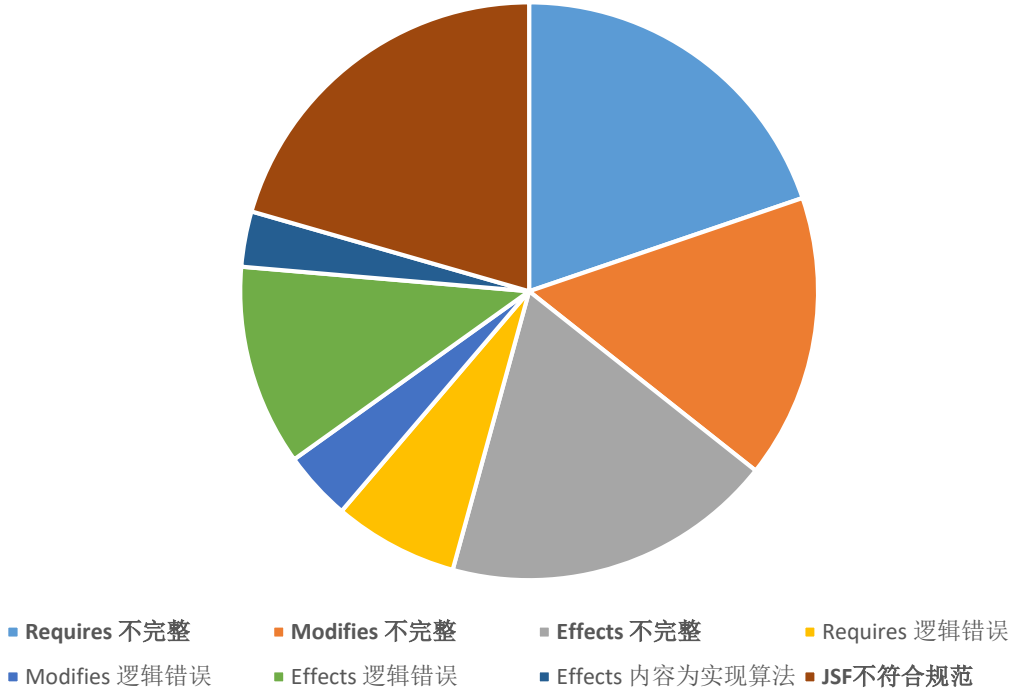
- 有些类管理的数据不具有类似于集合这种在数学上有清晰结构，如何定义其抽象函数？
  - 抽象函数用来约束实现者：属性设计(即表示对象)实现必须能够满足overview部分数据管理(即抽象对象)的概述
  - 从功能角度来定义数据的抽象表示，即看到哪些数据
- 子类的规格中，跟父类一样的部分还用不用写？
  - 如果重写了父类的方法，则需要重新撰写规格，并确保满足LSP条件
  - 抽象对象视情况补充子类额外管理的抽象数据
- 表示对象和抽象函数混淆
  - 表示对象就是一个类的属性/数据实现
  - 抽象函数是一种映射机制，用来说明表示对象如何映射到overview所阐述的数据（从而表明对象实现满足要求）

典型问题分析

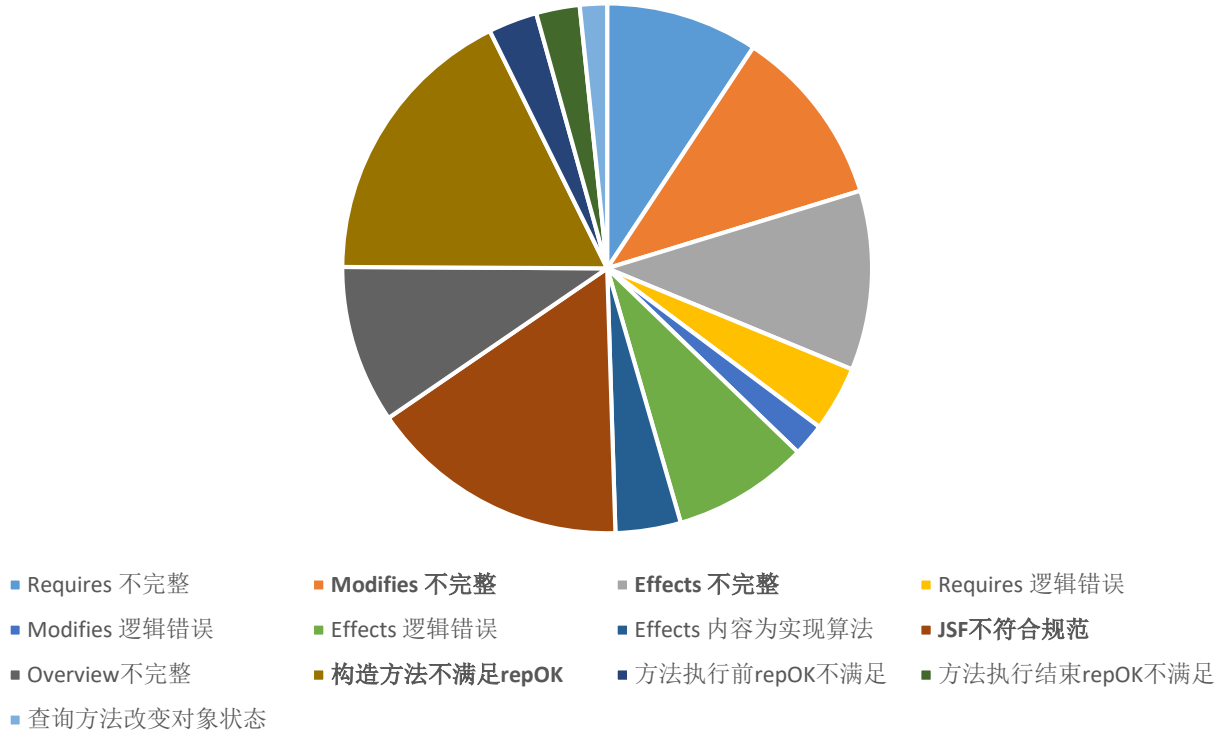
- 如果在实现过程中发现需要给一个类添加新的属性怎么办？
  - 如果分析和设计做的充分，这种问题基本不会出现
  - 修订不变式，把新增属性的约束扩充进去
  - 分析是否需要更新抽象对象，如果是，则需要修改抽象函数
- 如果在实现过程中发现需要为一个方法增加参数怎么办？
  - 如果分析和设计做的充分，这种问题基本不会出现
  - 检查前置条件并扩充（如果是全局适用过程，一般应保持）
  - 检查新增参数对方法的后置条件是否会产生影响，如果是，则修订后置条件
- 一个方法具有非常明显的算法特征(如求闭包或最短路径等)，有必要那样写后置条件吗？
  - 算法特征概括了求解思路或逻辑，但不等于讲清楚了正确解的边界条件
- 一个方法本身非常简单，有必要写规格吗？
  - 从工程角度，这样的方法确实可以不写
  - 但作为作业训练，还是要写（难的不会写，简单的又不愿写！）

两次作业的规格问题

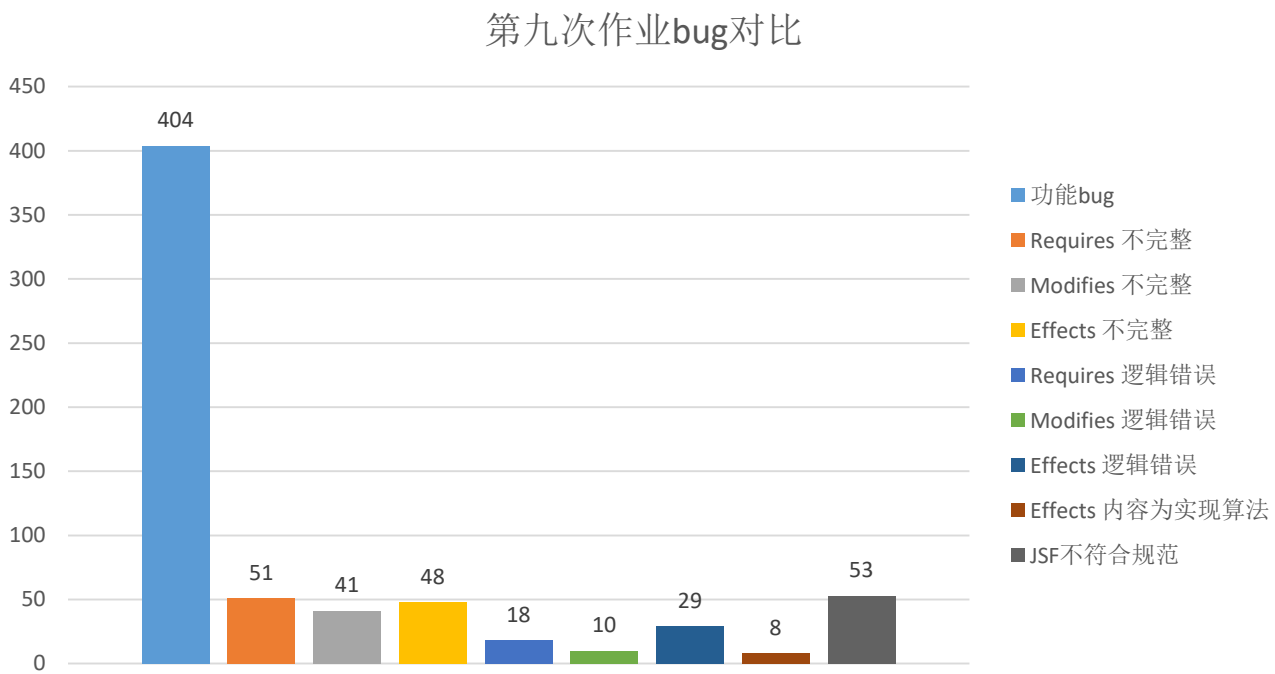
第9次作业整体规格问题分布

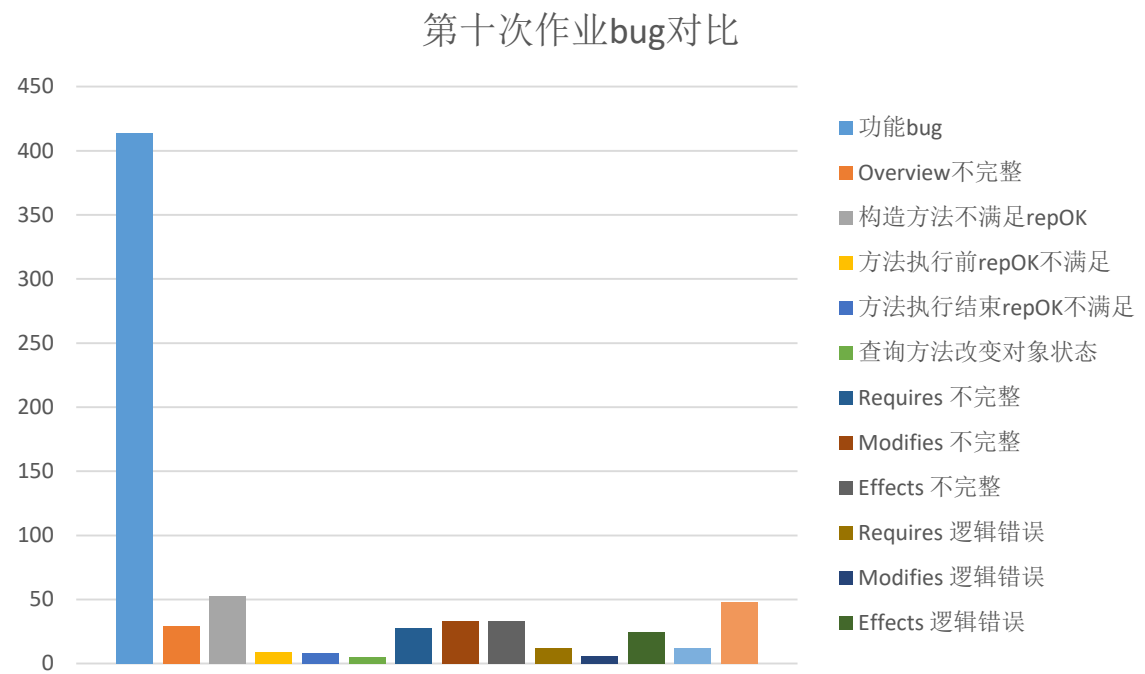


第十次作业规格问题分布



两次作业的规格bug与功能bug对比





功能bug与规格bug的相关性检验

第九次作业的功能bug与规格bug相关性检验结果

		功能bug	方法规格bug	Var(LOC)_H
功能bug	皮尔逊相关性	1	.989**	-.041
	显著性（双尾）		.000	.575
	个案数	193	193	192
方法规格bug	皮尔逊相关性	.989**	1	-.053
	显著性（双尾）	.000		.462
	个案数	193	193	192
Var(LOC)_H	皮尔逊相关性	-.041	-.053	1
	显著性（双尾）	.575	.462	
	个案数	192	192	192

**. 在 0.01 级别（双尾），相关性显著。

第十次作业的功能bug与规格bug相关性检验结果

		功能bug	类规格bug	Var(LOC)_H
功能bug	皮尔逊相关性	1	.994**	-.002
	显著性（双尾）		.000	.980
	个案数	202	202	201
类规格bug	皮尔逊相关性	.994**	1	-.008
	显著性（双尾）	.000		.915
	个案数	202	202	201
Var(LOC)_H	皮尔逊相关性	-.002	-.008	1
	显著性（双尾）	.980	.915	
	个案数	201	201	201

**. 在 0.01 级别（双尾），相关性显著。

问题改进建议

Overview要告诉使用者,一个类管理/维护什么数据，提供哪些能力（不是简单列出有哪些方法）。
Controller：出租车调度类，管理所有出租车和用户的叫车请求，并维护系统时间...。*调度类根据...<因素>来选择满足...<准则>的出租车，从而响应乘客请求。*

- 过程规格问题改进重点
  - Requires：对谁做出的要求，明确的信息是调用者可见
  - Modifies：需要明确什么
  - Effects：最终结果应满足的逻辑约束
- 子类规格问题改进重点
  - 重写方法：Sub-Req不能比Super-Req有更多要求
  - 重写方法：Sub-Effects不能比Super-Effects承诺更少
  - 重写方法：Sub-Mod不能比Super-Mod修改更多（外部可见）
  - 新增或重载方法：不能导致父类不变式被破坏
- Overview的撰写问题
  - 不能去介绍和解释类的属性，而是整体上告知用户这个类管理什么数据
  - 从整体上概述类功能，无需逐一解释每个方法
  - 通过抽象函数来确定具体属性的有效性

作业

- 针对第三单元的三次作业和课程内容，撰写技术博客
  - (1)调研，然后总结介绍规格化设计的大致发展历史和为什么得到了人们的重视
  - (2)按照作业，针对自己所被报告的规格bug以及雷同的规格bug（限于bug树的限制，对手无法上报），列一个表格分析
    - 规格bug类别（采用bug树上的名称）、每个出现所对应方法的代码行数
  - (3)分析自己规格bug的产生原因
  - (4)分别列举5个前置条件和5个后置条件的不好写法，并给出改进写法
  - (5)按照作业分析被报的功能bug与规格bug在方法上的聚集关系
    - 即给出表格：方法名、功能bug数、规格bug数
  - (6)归纳自己在设计规格和撰写规格的基本思路和体会