什么是软件？ 拓展人类、计算、执行、解决

何为？ 探索、实践 高效研发 面向解决实际问题

——一 理论方法和实现技术

DPC？ Description 描述清楚 Prediction 可预测

Control 可控。

三大元件：第一：1）拓展、可编程

2）能力由外界到定，内部支持升级

3）价值与问题相当。

第二：用户价值通过与系统交互完成，

每个具体交互有明确目的和操作

第三：技术进步往复不断设计出支持组装构及支持升级

自形式。

软件生命周期？ 定义： 问题定义

可行性研究

开发： 需求分析

概要设计

详细设计

编码

测试

运行： 软件维护

三大根本问题：1）构造创造有价值软件，找出形式方法

2）抽象 可动态升级自动化方法

3）设计 效率权衡。

九大重点任务. 1)软件需求和应用场景.
2)业务建模和数学模型
3)软件架构和应用框架
4)设计模式与代码重构
5)核心设计和前端编程
6)软件测试与自动测试
7)安全问题和隐私设计
8).性能评价和设计优化
9)系统部署和系统运营.

用户故事? 定义功能需求. 用户用例. 交替使用与用例.
Specific 确定性
Measurable 可评估
Achievable 可实现
Relevant 相关
Timeboxed 时间限制

敏捷开发. 需求进化为核心. 迭代.
把大项目分为多个低耦合的小项目.
Scrum. 主管角色. 负责人. 开发团队

计划-文档开发: 瀑布式.
PD 自上而下

特征: 场景 快速原型. 螺旋模型.
引入风险分析

结构化分析: 自顶向下,逐层分解.
顶层框架
中间过渡.
底层细节

设计模式：工厂、抽象工厂、单例、建造者、模板、组合、……

DRY：任何一部分知识在系统内有单一、明确、权威的描述。

重构：不影响外部表现下，对内部更改。

①清晰
②维护性差
③分析
④改进

原则：①一次做
②隐含能否
③ SRP. 单一职责
④避免重复
⑤高内聚低耦合。

代码味道 SOFA：
S short 短。
O one 一件事。
F few 少参数
A Abstraction 同层抽象。

SOLID 原则
S. simple 单一职责
O：开闭 对修改闭，扩展开。
L. 里氏替换。 意味代替接也 work.
I：依赖注入 (接口隔离)
D：迪米特。 仅和朋友交谈。

测试：单元、集成、系统、验收、功能、回归、接口集成、
冒烟、回归。

四、 设计题

■ 需求分析

1. 为下面的功能写一个可信的用户故事和两个场景，并确保你的答案足够 smart：

功能 1: 通过上课时间搜索课程

用户故事：作为一个学生，我想要通过上课时间来浏览课程，以便于我能快速找到满足我需要的上课时间的课程。

场景 1:学生通过点击网页中的上课时间如周一 3、4 节，系统自动将周一 3、4 节上课的所有课程影在网页中展示出来

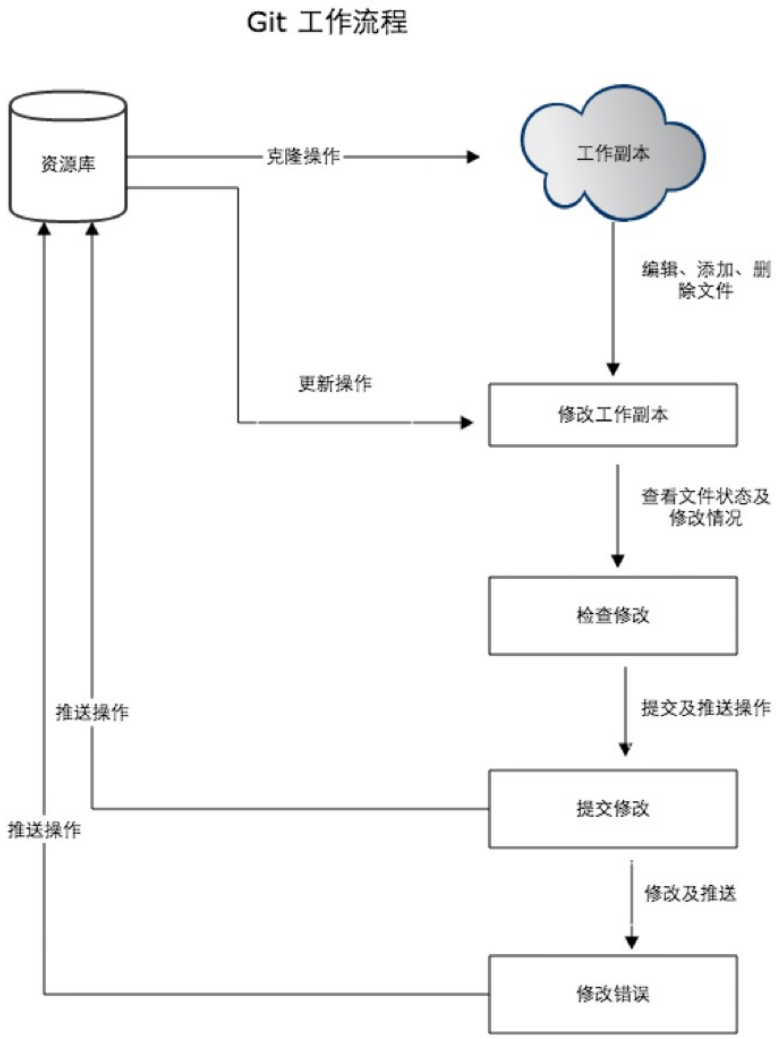
场景 2:学生通过手动输入上课时间如周一 3、4 节，系统自动将这个时间的课程在网页中显示出来

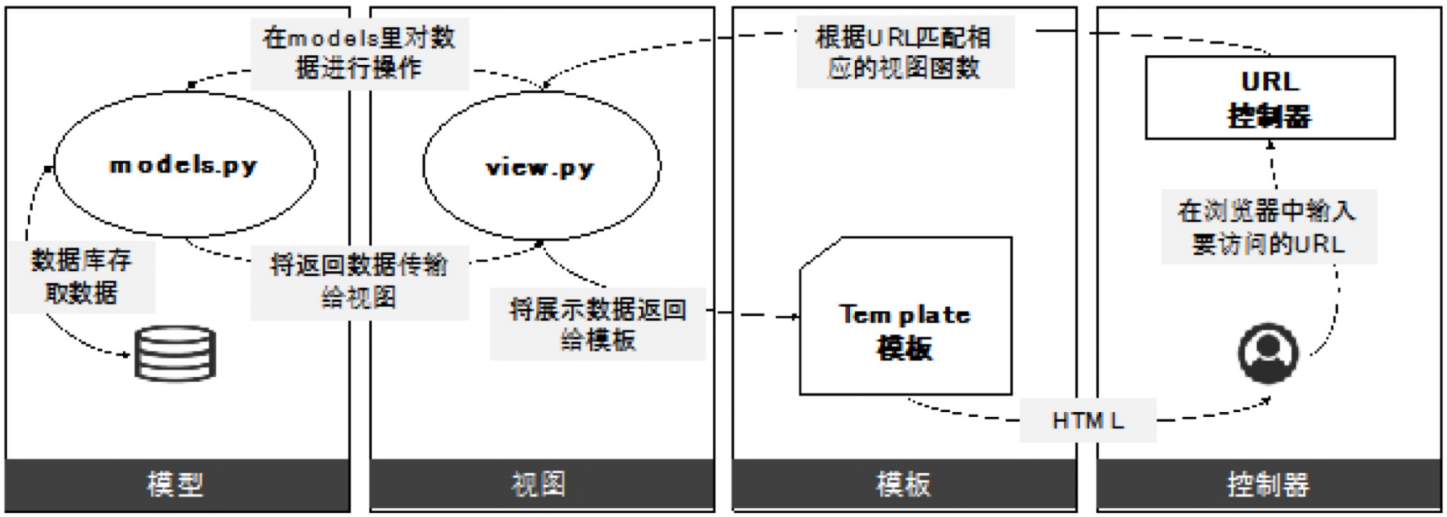
功能 2:根据学生的成绩来进行排序

```
require 'observer'          # 1 分

class TV
include Observable          # 1 分
def turn_on                 #1 分
puts "TV is on!"
changed #TV state changed   #1 分
notify_observers(self)      #1 分
end
end

class Boy
def update(tv)              #1 分
puts "I will go to watching TV!"
end
end

tv = TV.new                 #1 分
boy = Boy.new               #1 分
tv.add_observer(boy)        #1 分
tv.turn_on                  #1 分
```

8. Answer：

需求分析：该系统需要使用校园卡作为用户身份的唯一认证。创建一个 user 类，类中包含该 user 的 ID 号、用户状态（正常、异常，异常比如休学、毕业、生病等）、出校许可，公开方法预约班车 appointment()。Student 类和 Teacher 类继承 user 类，student 类里面加入私有变量比如 Teacher 变量数组 supervisor[]，私有方法比如申请出校 apply()。Teacher 类加入私有变量管理的学生 supervised[]，私有方法比如同意出校 approve()和拒绝申请 reject()。在出校门的时候对每个 user 的出校许可进行检查，如果为 true 则能够进出校园并且使用预约班车功能(里氏替换原则)。如果学校或老师有通知，可以通过 update()方法通知 supervised[]列表下的每一个学生（observer 模式）。





50. Answer:

(1). BDD 可以解决第一个问题，TDD 可以解决第二个问题。

(2). 原因：BDD 是行为驱动开发。它是一种敏捷软件开发的技术，鼓励软件项目中的开发者、QA 和非技术人员或商业参与者之间的协作。主要是从用户的需求出发，保证程序实现效果与用户需求一致。

TDD 是测试驱动开发。TDD 的原理是在开发功能代码之前，先编写单元测试用例代码，测试代码确定需要编写什么产品代码。它的基本思路就是通过测试来推动整个开发的进行，但测试驱动开发并不只是单纯的测试工作，而是把需求分析，设计，质量控制量化的过程。TDD 首先考虑使用需求（对象、功能、过程、接口等），主要是编写测试用例框架对功能的过程和接口进行设计，而测试框架可以持续进行验证。

```ruby
# app/models/user.rb
class user < ApplicationRecord
  has_many :microposts
end

# app/models/micropost.rb
class Micropost < ApplicationRecord
  belongs_to :user
  validates :content, length: { maximum: 140 }
end
```

validates :name, :email, presence: true

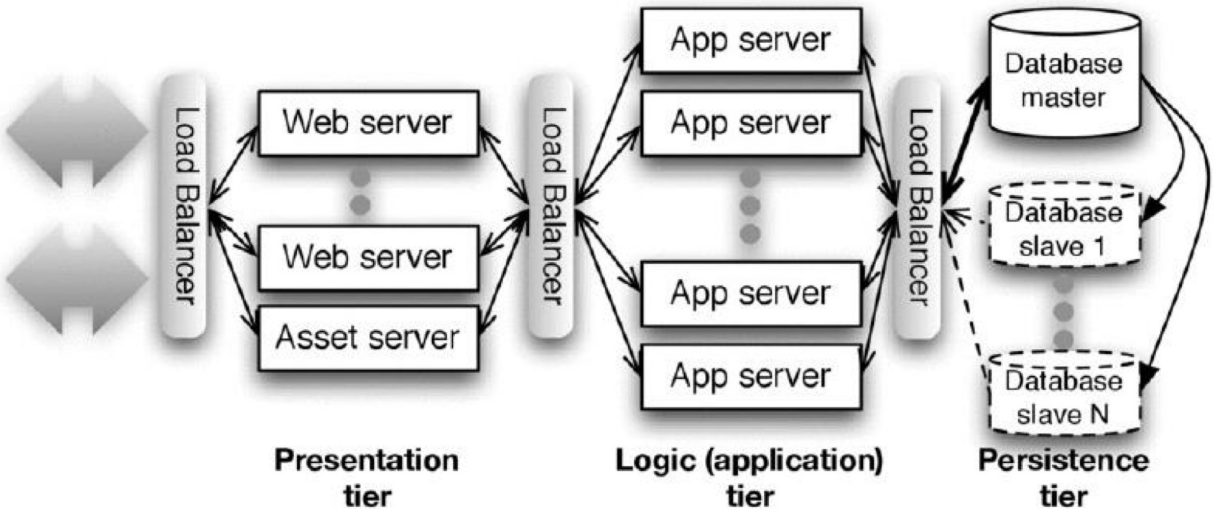
validates: content, presence: true, length: { m --- }.

请在上面的两个代码段的合适地方加入语句，（1）为 User 模型的 name 和 email 属性添加存在性验证；（2）确保微博的内容不能为空存在性验证。

```
price       #       变量
@price      #       变量
@@price     #       变量
$price      #       变量
```

```ruby
class UserTest < ActiveSupport::TestCase
  # test "the truth" do
  #   assert true
  # end

  def setup
@user = User.new(name: "Example User", email: "user@example.com",
                 password: "password", password_confirmation: "password")
  end

  test "@user should be valid" do
    assert _______ @user.valid?
  end

  test "name should be present" do
    @user.name = "     "
    ___assert_not___ @user.valid?
  end
end
```

无共享（Shared Nothing）架构的解释（多点个面的图）。



1）

三层体系结构包括表示层（渲染视图并与用户进行交互）、逻辑层（运行 SaaS 应用代码）、持久层（存储应用程序数据）。

三层无共享架构之所以称为无共享架构，是因为层内实体一般不进行内部沟通，它允许在每层中独立地增加计算机以满足需求。

2）

该架构中的负载均衡器可以通过硬件设备或专门配置的 Web 服务器均匀地分配工作量。无状态的 HTTP 可以让无共享组件成为可能，所有请求都是独立的，可以向表示层或逻辑层中的任何服务器分配任何请求。

3）

持久层是难以横向扩展的。上图显示了主从方法，是在数据库中读取比写入更频繁的时候使用的。任何从机都可以执行读取，只有主机才能执行写入，然后主机使用写入的结果尽可能更快地更新从机。这种技术只是延缓了扩展性问题的出现，而无法从根本上解决这个问题。