软件过程和项目计划

> 小组：Autograd
> 选题：课程信息系统

拟采用模型

软件开发模型采用增量模型，将项目整体划分为若干构件，并在对每一构件的目标功能进行细化后，评估并设置每个功能点的优先级（本项目拟采用3级优先级）。开发按优先级从高到低进行开发，在每个优先级内逐个完成并交付构件，交付和评价后将对后续开发目标进行完善和更新，直至最终交付产品。

分工

管理工具

选择GitHub作为管理工具，使用其以下功能：

- 远程仓库：代码托管
- branch：分支管理
- issue：对开发过程的所有待讨论事项进行讨论和记录，任务分配
- milestone：记录开发过程，进行版本标记

合作方式

小组成员角色及任务如下：

- 产品经理（1）：需求分析、功能设计、产品评估、版本管理等
- 项目架构师（1）：设计项目整体框架、代码评审、测试编写等
- 后端开发（2）：完成项目后端开发任务
- 前端开发（2）：完成项目前端开发任务

开发过程采用SVN管理，即所有开发人员并行开发，完成一个阶段开发任务后向主库提出提交申请，由相应Reviewer进行代码评审后将代码提交到主库，Reviewer对主库代码直接负责。

具体分工

- 产品经理：江雪颖
- 项目架构师：冯玉彤
- 后端开发：刘本林、饶永铭
- 前端开发：黄佳辉、张欣怡

进度

整体开发进度约为10周：

1. 需求分析及功能设计（1周）：通过多渠道获取需求，编写用户故事，并由此进行功能设计
2. 项目架构（1周）：确定项目设计模型，搭建框架，设计数据库
3. 第一阶段开发（3周）：完成项目基本功能，进行第一阶段评估与迭代
4. 第二阶段开发（2周）：完成项目进阶功能，进行第二阶段评估与迭代
5. 第三阶段开发（2周）：完成项目可扩展功能，进行第三阶段评估，给出未来可扩展功能
6. 软件测试（并行）：对每个功能模块进行功能测试，对重点开发部分进行单元测试
7. 软件部署及发布（1周）：对项目进行线上部署

版本管理计划

版本管理模型

采用4条分支进行版本管理：

1.Master

项目主分支，处于项目最新的可发布状态。

2.Develop

开发分支，处于项目最新开发状态。由于本项目规模较小，不再使用多feature分支，统一在Develop分支进行开发。

3.Release

发布分支，项目由Develop分支移至该分支进行发布测试及相应修改，完成后合并到Master及Develop分支。

4.Hotfix

紧急修复分支，在项目发布后出现问题时，将Master分支移至该分支进行bug修复，完成后合并到Master及Develop分支。

版本管理策略

采用SVN管理，在不同阶段的管理策略如下：

开发阶段

所有提交代码经过code review后提交到Develop分支，所有开发任务完成并通过测试后进入预发布阶段。

预发布阶段

将Develop分支移至Release分支进行部署，通过部署测试后并入Master及Develop分支，并进入发布阶段。

发布阶段

项目在Master分支发布，在出现紧急问题时，移至Hotfix分支进行紧急修复，修复完成后并入Master及Develop分支。

版本设置

- 本学期开发版本主版本号均为0，正式发布版本主版本号为1
- 按照开发优先级顺序，每个开发阶段设置为0.1, 0.2, 0.3
- 对于每个开发阶段问题修复后的版本，设置为0.1.*

风险管理计划

需求风险

- 初期需求获取不准确 / 不全面
  应对策略：需求获取时尽量采用多种渠道，覆盖面向人群。
- 需求在开发过程中发生变化
  应对策略：在当前选课系统不发生重大变化前，本项目需求较为稳定，风险程度低。

技术风险

- 所需数据接口关闭
  应对策略：提前获取所需数据，或采取人工导入方式。
- 前后端开发发生冲突
  应对策略：采用前后端耦合低的Restful设计，提前设计接口。

成本风险

本项目除线上发布可能使用的服务器外，不存在其他成本因素，故风险程度较低。

进度风险

- 某一阶段开发超出预期时间
  应对策略：leader及时监督开发进程，如延期无法避免，可考虑适当减少扩展功能的开发。
- 某一部分开发由于人员原因影响整体进程 应对策略：每一部分配置两名开发人员，尽可能保证完成开发任务。