成　绩	
评阅人	

复 旦 大 学

研 究 生 课 程 论 文

论文题目：　基于自习座位预约系统的敏捷实践探析

修读课程：　软件过程管理（SOFT620011）

选课学期：　2024-2025 学年第二学期

选课学生：　宋安洋（24210240291）

完成日期：　2024.6.12

目　　录

摘　　要

随着技术的快速发展，传统的软件开发方法逐渐暴露出响应迟缓和效率低下的问题，尤其是在需求变化迅速和复杂度较高的项目中。为了解决这些问题，敏捷开发作为一种迭代式、增量式的开发方法应运而生，强调快速响应、灵活调整和频繁交付。敏捷开发通过 Scrum 等框架促进了团队间的协作与自组织，极大提高了开发效率和质量。此外，DevOps 理念的兴起推动了开发与运维的深度融合，通过自动化工具，持续集成和部署流水线等技术手段，优化了软件交付流程，缩短了产品上市时间，并提升了软件的稳定性和可靠性。

本文主要围绕需求管理和质量管理的完整过程进行展开，基于对理论的深入剖析，详尽探讨了在敏捷开发环境下，如何通过用户故事和迭代等需求管理方法，以及如何通过自动化测试和部署流水线等质量管理方法确保软件质量的高效保障，并结合在华为 DevCloud 上的具体实践，详尽阐述了我们是如何将这些理念与技术应用于自习座位预约系统这个项目，探讨了这些理念和技术在我们具体实践中的目的、作用和意义，并基于理论分析了实践的效果、问题、优缺点以及可能的改进，通过这个过程中讲述了我对这些方法的理论和实践的深入理解。最后，本文简要提及了对系统开发结果与其中技术难点的探讨。

关键词：敏捷开发方法；Scrum；DevOps；用户故事；冲刺/迭代；自动化测试；持续集成/持续部署；部署流水线

An Exploration of Agile Practices Based on Study Room Booking System

Abstract

With the rapid development of technology, traditional software development methods have gradually exposed issues such as slow response and inefficiency, especially in projects with rapidly changing requirements and high complexity. To address these problems, Agile development has emerged as an iterative and incremental approach, emphasizing quick response, flexibility, and frequent delivery. Agile development, through frameworks such as Scrum, has promoted collaboration and self-organization among teams, significantly improving development efficiency and quality. In addition, the rise of DevOps has driven deep integration between development and operations. Through automation tools, continuous integration, and deployment pipelines, it has optimized the software delivery process, shortened time-to-market, and improved software stability and reliability.

In this paper, I mainly focus on the complete process of requirements management and quality management. Based on an in-depth analysis of the theory, it discusses in detail how, in an Agile development environment, methods such as user stories and iterations are used for requirements management, and how automated testing and deployment pipelines are applied to ensure the efficient assurance of software quality. Additionally, through practical examples on Huawei DevCloud, this paper elaborates on how these concepts and technologies were applied to the study room booking system project. It explores the purpose, role, and significance of these methods and technologies in our practice, and based on theoretical analysis, discusses the effects, problems, advantages, disadvantages, and potential improvements of the practice. Through this process, I provide a deep understanding of these methods in theory and practice. Finally, it briefly mentions a discussion of the system development results and the technical challenges encountered.

Keywords: Agile Development Method; Scrum; DevOps; User Stories; Sprint/Iteration; Automated Testing; Continuous Integration/Continuous Deployment; Deployment Pipeline

1 引言

随着信息技术的迅猛发展，软件开发环境变得愈发复杂，客户需求不断变化。传统的软件开发方法在面对快速变化和高复杂度的需求时，暴露出了诸如响应迟缓、效率低下等问题。为了解决这些问题，敏捷开发方法应运而生，并逐步取代了传统的瀑布式开发模型。敏捷开发强调快速响应、灵活调整以及频繁交付，并通过 Scrum 等框架实现了团队间的协作和自组织，极大提高了软件开发的效率与质量。此外，随着 DevOps 的兴起，开发与运维的协同工作成为软件开发新趋势，通过自动化工具和持续集成，DevOps 优化了软件交付流程，缩短了产品上市时间，提升了软件的稳定性和可靠性。

在本文中，我深入探讨了敏捷开发、Scrum 和 DevOps 的实际应用，并以华为 DevCloud 为例，展示如何在我们的自习座位预约系统这个实际项目中落实这些管理理念和技术。首先在第二章节中，我从宏观角度概述了软件过程管理的发展史并引出了对敏捷、Scrum、DevOps 相关理论及其意义的阐述，同时在这个过程中深入理解并详尽分析了这些理论概念是怎样对应到 DevCloud 中以及应当如何使用，具体使用的叙述在第三四章节中。在第三、四章节中，基于展开细叙了需求管理和质量管理过程中的相关理论，我们详细描述了对应到我们的项目具体是怎么做的。具体来说，在需求管理中，我首先讲述了我对用户故事的深入理解，深入理解用户故事中的 what、why、how，在此基础上叙述了我们是怎样遵循理论编写用户故事的，有着什么样的优缺点，然后简要介绍了我们的迭代过程与任务分解。在质量管理中，我首先分析了自动化测试的重要意义、简要介绍了我们的自动化测试是怎样做的，然后对应于第二章中已经探讨过的 DevCloud 理念详细讲述了我们在本项目中的整个部署流水线的过程。最后，在第五六章中，我简要补充了我们的开发过程、开发结果的展示以及对其中技术难点的理解。

2 软件过程管理概述

从最初的硬件驱动到独立的、逐步规范化的软件开发过程，软件项目管理经历了从顺序化、形式化到灵活、快速响应的演变。传统的瀑布式管理逐渐被敏捷和 DevOps 等并行、灵活的方式所替代，这标志着项目管理理念和技术方法的深

刻变革。这些发展使得软件开发更好地适应了快速变化的业务需求和技术环境。

20 世纪 50-60 年代，是软件工程初步探索的时代。软件开发最初处于硬件工程的阴影下，软件工程师需要同时处理硬件和软件问题。早期的开发方法如“Code-and-fix”和“英雄主义开发”被广泛应用，缺乏规范的过程管理，导致了大量的缺陷和低效的开发，这种非结构化和难以维护的源代码被称作意大利面条式的代码。

20 世纪 70 年代，是形式化方法与瀑布模型的时代。随着软件的复杂性增加，开始尝试引入形式化方法以规范化软件开发。瀑布模型作为早期的线性开发过程，强调按顺序进行需求分析、设计、编码、测试等阶段。这一阶段的挑战包括缺乏反馈、过度依赖文档、以及形式化方法学习成本高等。

20 世界 80 年代，生产率和过程不断改进，是软件开发从量化管理和过程改进走向标准化、工具化的关键时期。通过引入软件能力成熟度模型（SW-CMM）等过程改进模型，结合软件复用和面向对象的方法，软件开发逐步从不规范的“英雄主义”走向更加高效、可预测的流程管理。技术的进步和工具的普及，使得软件开发的效率和质量得到了显著提升，为后来的敏捷方法和 DevOps 等新兴管理模式奠定了基础。

20 世纪 90 年代，螺旋模型与并行过程出现，敏捷和轻载的开发过程逐步兴起。为了应对瀑布模型带来的长周期和不灵活性，螺旋模型出现，它支持并行工程和风险管理，强调在多个阶段进行迭代和反馈。同时，CMM 软件过程体系被正式提出，组织级的过程改进得到关注。此外，开源软件的兴起代表了另一种并行开发的实践方式。

21 世纪初的 10 年，随着软件开发需求变得更加灵活和快速，敏捷方法应运而生。2001 年，《敏捷宣言》明确提出：个体和交互胜过过程和工具、可以运行的软件胜过详尽的文档等核心价值观，强调快速交付、客户合作和应对变化。经典的敏捷方法包括 Scrum、XP（极限编程）等，推动了开发过程的灵活性和团队的自组织。

2010 年以来，是 DevOps 兴起与爆发的时代。DevOps 的出现标志着开发与运维的一体化，进一步推进了敏捷思想的应用。精益开发与 Kanban 方法为高效的

开发和运维流程提供了理论基础。技术的进步，如虚拟化、云计算和微服务架构，结合自动化工具，使得软件开发周期更短，质量更高，运维更灵活。

2.1 敏捷开发方法

敏捷开发方法（Agile Software Development）是一种强调灵活性、快速迭代和持续改进的软件开发方法论。它应对了传统瀑布式开发模型的局限，尤其是在应对需求变更、复杂性和不确定性方面。敏捷并非单一的开发技术，而是一个包含不同框架和实践的总称 Umbrella Term，如 Scrum、XP、Kanban 等。这些方法在实践中都遵循敏捷宣言的核心价值观和原则。通过提供更灵活的开发框架、更频繁的交付周期和更强的客户参与感，敏捷开发方法不仅仅提高了开发效率和质量，更通过不断的创新和灵活应变，帮助组织在复杂和动态的环境中维持了竞争力。

为什么要敏捷？软件开发本质上是一项探索性活动，充满不确定性和复杂性。传统的“瀑布式”开发模型往往假设需求和解决方案是清晰的、可预测的，但现实中的项目往往面临不断变化的需求、技术的创新以及复杂的业务环境。第一，复杂性与不可预测性。软件开发需要应对高复杂度和不确定性，单纯依靠详细的前期规划无法解决这些问题。第二，需求变化。随着项目进展，客户的需求可能发生变化，敏捷强调适应这种变化而不是固守最初的计划。第三，早期价值交付。传统开发往往要等到项目后期才交付成果，而敏捷则通过短周期的迭代交付，尽早交付可用的软件版本。

怎么做敏捷？敏捷方法强调通过迭代和增量的方式推进开发。每个迭代（通常是 2-4 周）都会交付一个可用的软件版本，并在迭代后进行回顾与调整，从而不断改进开发过程。敏捷的实践：如每日站立会议、测试驱动开发、结对编程、回顾、持续集成等。敏捷的原则：敏捷宣言包括 12 个原则，简单概括来说，敏捷强调快速交付、高度协作、灵活应变、持续改进和自组织团队，以实现高效、优质的软件开发。敏捷的价值观：以敏捷宣言作为敏捷社区共同认可的价值观，不同的方法集可能有各自的描述。

敏捷的意义。正如在本小节第一段所概述的那样。敏捷更加灵活多变，也能

更好提高效率和质量，更具体来说。第一，适应变化。敏捷的核心优势在于应对快速变化的需求环境。通过迭代和反馈，开发过程始终保持灵活，能够及时调整方向，避免项目偏离客户期望。第二，交付高质量的可用软件。通过频繁的交付和测试，敏捷确保了软件质量的持续提升，并且能及时发现和修复问题。第三，提高团队协作与士气。敏捷强调团队的自组织和自主性，成员之间更紧密的合作和沟通有助于增强团队的创造力和凝聚力，最终提高整体生产力。第四，提升客户满意度。敏捷关注客户需求，保证开发始终围绕客户价值展开，增加了客户参与和反馈的机会，能够及时做出调整，最终提高客户的满意度。第五，减少浪费。通过敏捷的价值观“简化工作、避免不必要的工作”来减少浪费。例如，减少冗余文档和繁琐的计划工作，将精力集中在最核心的目标上。

2.2 Scrum

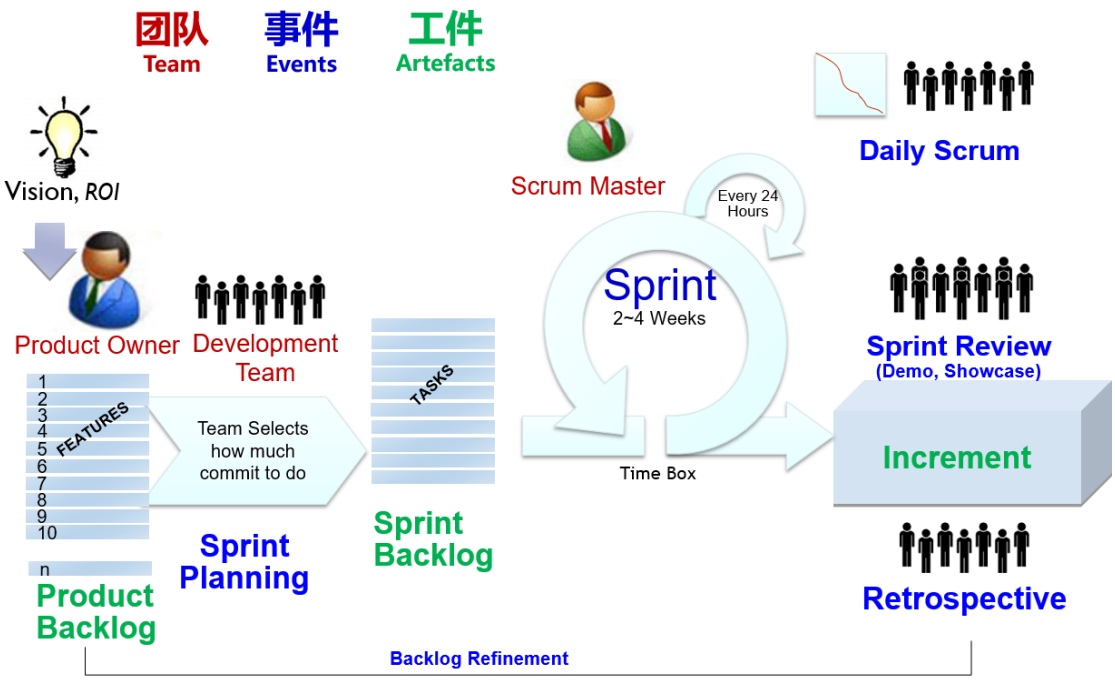

图2.1 Scrum过程模型

Scrum是一个周而复始、持续改进的循环：计划（Plan）->执行（Do）->评审（Review）。这个循环又称迭代周期、冲刺（Sprint）。Sprint是一个时间盒（Time box），通常是2-4周。每个Sprint的Review的反馈（Feedback）影响

下一次 Sprint 的 Plan。Do 的结果得到一个产品或者说潜在可发布增量（SW，Software/Work）。随着每个完成的 Sprint，SW 是在不断增大的，即产品功能和价值都在累积和增长。即 Scrum 以增量方式交付实际价值，而不是等到项目结束。Scrum 过程模型如图 2.1 所示。

Scrum 定义了三个核心角色。他们共同协作以确保产品的成功交付。

1.产品负责人（Product Owner，PO）。PO 是产品的愿景（Vision）和投资回报率(ROI) 的最终负责人。他对产品的价值最大化负全责。PO 的核心活动是管理和维护产品待办列表(Product Backlog)。这意味着他需要与利益相关者沟通，收集需求，并根据业务价值、风险、依赖性等因素对产品待办列表中的各项特性（Features)进行排序和优先级划分。他必须清晰地传达产品目标和产品待办列表项。

2.Scrum 主管（Scrum Master，SM）。SM 的职责是促进 Scrum 框架的正确应用和理解、消除障碍，帮助团队和产品负责人更好地工作，并促进 Scrum 实践的改进。SM 的核心活动是确保 Scrum 事件按时举行、保持积极和富有成效；帮助团队移除障碍；教导产品负责人如何有效地管理产品待办列表；教导开发团队自组织和跨职能；保护团队免受外部干扰。

3.开发团队（Development Team）。开发团队负责将产品待办列表中的高优先级项转化为可用的增量（Increment）。开发团队的核心活动是在 Sprint 规划(Sprint Planning)中，选择他们在当前 Sprint 中能够承诺完成的待办事项并决定如何实现这些待办事项。

Scrum 定义了三个核心工件。它们代表了工作或价值，旨在最大化透明性。

1. 产品待办列表（Product Backlog）。一个动态的、按优先级排序的产品特性、功能、需求、增强和缺陷修复的列表。它是产品所有已知工作的唯一来源。产品代办列表由产品负责人持续进行待办列表梳理（Backlog Refinement），确保产品待办列表项清晰、估算合理，并为未来的 Sprint 做好准备。

2. Sprint 待办列表（Sprint Backlog）。由开发团队在 Sprint 规划会议上创建，包含当前 Sprint 中将要完成的产品待办列表项，以及将这些项转化为“完成”状态所需要执行的详细任务 (TASKS)。它是开发团队对 Sprint 期间交付增

量的承诺。

3.增量（Increment）。增量是一个已完成的、可用的、并且符合产品定义“完成”标准的软件部分。它是所有 Sprint 完成的产品待办列表项的总和，并且必须是可发布的。每个 Sprint 结束时都必须产生一个潜在可发布的增量。这确保了持续的价值交付和尽早获得反馈。

Scrum 定义了五个正式事件。每个事件都有特定的目的和时间盒 (Time Box)，确保了规律性和透明性：

1.冲刺（Sprint）：Scrum 的核心，迭代周期，一个固定长度的时间盒 (通常为 2-4 周)。在一个 Sprint 中，Scrum 团队完成所有开发工作：规划、开发、测试、集成、评审和回顾。

2. Sprint 规划（Sprint Planning)。在 Sprint 开始时，通常持续 8 小时或更短。Scrum 团队共同参与，确定下一个 Sprint 的目标，以及在 Sprint 中将要完成哪些产品待办列表项，以及如何实现这些项（即创建 Sprint 待办列表）。

3. 每日站会（Daily Scrum）。每天同一时间、同一地点举行，通常为 15 分钟的时间盒。主要参与者是开发团队，SM 和 PO 可以旁听。旨在检查前一天的工作，并计划接下来 24 小时的工作。它帮助团队成员之间同步，识别障碍，并调整 Sprint 待办列表以达到 Sprint 目标。

4.Sprint 评审（Sprint Review）。Sprint 结束时，通常持续 4 小时或更短，Scrum 团队和利益相关者共同参与，检查已完成的增量，并根据需要调整产品待办列表。团队向利益相关者展示完成的工作，并收集反馈。这是一个非正式的会议。

5. 回顾会议（Retrospective）。Sprint 评审之后，Sprint 结束之前，通常持续 3 小时或更短。Scrum 团队共同参与，检查上一个 Sprint 中团队在人员、关系、过程和工具方面的表现。识别并计划在下一个 Sprint 中可以改进的领域。目标是持续改进。

在华为 DevCloud 中。我们创建了一个 Scrum 项目，并随着本课程项目的进行，完整地体验了 Scrum 框架的整个过程。下面是我的一些体验与理解。

首先，成员管理中可以设置角色，可以看到 devcloud 中的角色有：项目管

理员、项目经理、产品经理、测试经理、运维经理、系统工程师、Committer、开发人员、测试人员、参与者、浏览者。其中，我是项目管理员，这是不可设置的，谁创建的项目就是谁，项目管理员就是 Scrum 主管 SM；产品经理对应 PO，但是 PM 和 PO 应当是不同的，在本段落后续会详细探讨；Committer、开发人员、测试人员显然是属于 Development Team 的。而对于参与者、浏览者，显然不是 Scrum 核心角色。而测试经理在 Scrum 中也是不存在的，Scrum 中没有管理型测试角色，测试工作由 Development Team 负责。运维经理在 Scrum 中同样是不存在的，运维任务是纳入开发团队职责，DevOps 化的。然后说两个关键的问题，缩写都是 PM 的项目经理和产品经理。传统非敏捷团队的产品经理和敏捷的产品负责人 PO 有一些共通点，我认为他们的共通点是都负责 PRD（Product Requirements Document）编写、与开发沟通协作、优先级排序，但是这个过程中 PM 面向战略，而 PO 则是面向执行性。不同点是，PM 负责需求调研和产品规划，而 PO 只是部分参与；PO 主导迭代管理，而 PM 只是参与为主。下一个问题是项目经理 PM 和 Scrum 主管 SM 的区别。在 devcloud 中，我还有一个默认的角色项目经理 PM，但 PM 和 SM 显然是不同的。我在本科曾有过一个名为软件项目管理的课程，和本课程名字很像，但内容截然不同。传统非敏捷团队的项目经理 PM 本质就是字面义管项目的，是项目交付的总负责人，注重成果，管理的内容包括九大知识域：项目整体管理、范围管理、时间管理、成本管理、质量管理、人力资源管理、沟通管理、风险管理、采购管理。而 SM 只涉及到其中的质量、沟通、风险管理，SM 注重的是团队和流程优化，实际工作内容旨在帮助团队理解并践行 Scrum,对 Scrum 流程的实施和团队敏捷实践负责。从权力和决策角度来看，项目经理 PM 拥有项目最高的决策权（根据组织结构不同，也可能是项目管理办公室经理 PMO），并负有管理职责。而 SM 无传统意义上的权威，仅是“仆人式领导”，不指定任务，只是促进团队自己决定怎么做。

然后，devcloud 的需求管理中，规划栏目下，可以创建思维导图规划，从 epic 直到 user story 这一级都是应当由 PO 完成的。这个 user story 就是 Product Backlog,并且在每个 sprint 循环结束后由 PO 根据团队 Retrospective 的结果不断 Refinement。然后在迭代栏目下，就是一次 Sprint 的 planning 与

backlog 了，PO 和 SM 参与，开发团队 Sprint Planning 选择哪些要做、预计时间，就是 devcloud 这里的创建迭代这个功能，然后这个过程中一般应当由开发团队将 user story 进一步拆分为 task，得到的这个 sprint 要完成的这些 task 就是 Sprint Backlog。然后在工作项栏目下的 Backlog 栏目下，我们可以看到完整的 Backlog。

有关需求管理我们在 devcloud 更具体的使用与实践内容将在第三章需求管理中进一步叙述。

2.3 DevOps

DevOps 是一种将开发（**Dev**elopment）与运维（**Op**erations）深度融合的理念和实践体系，旨在通过端到端的协作、自动化和持续反馈，提升软件交付的速度、质量和可靠性。

简单来说，敏捷本身只是开发方法，DevOps 是敏捷方法向部署的延伸。敏捷开发日趋成熟的背景下，部署成为瓶颈，引发了对开发与运维之间协同效率的关注，2009 年 Flickr“每十分钟部署一次”的实践成为 Dev 和 Ops 有效结合在一起的关键里程碑。

敏捷宣言：个体与交互 高于 过程与工具；工作的软件 高于 详尽的文档；客户合作 高于 合同谈判；响应变化 高于 遵循计划；而 DevOps 宣言延续了敏捷宣言，只是将第三条进一步扩展为了客户及程序员合作 高于 合同谈判。

而在此基础上 Gene Kim 提出了三大 DevOps 实践原则：①思考系统的端到端流程 ②增加反馈回路 ③培养一种不断实验以及通过反复实践达到精通的文化。

DevOps 不拘泥于某一特定方法，但常融合以下敏捷方法论：Scrum：着重于项目管理；XP：着重于软件开发最佳实践（如 TDD、持续集成）；Kanban：通过可视化和限 WIP 优化流程控制。具体实践包括：持续集成（CI）与持续部署（CD）、自动化测试、配置管理与版本控制、容器化部署与动态编排（如 Docker, Kubernetes）等。

在华为 DevCloud 中。其中代码托管功能（和 github 一样），实现了持续集

成、分支管理、版本控制等。代码检查功能控制了编写上的质量问题。持续交付功能下，可以看到发布管理功能新建云服务器环境，编译构建功能进行打包，然后部署功能中配置环境并完成对应用包的部署。整个过程可以集成为流水线功能。

有关 CICD、自动化测试等内容及我们在 devcloud 更具体的使用与实践内容将在第四章质量管理中进一步叙述。

3 需求管理

3.1 用户故事

为什么使用用户故事？传统非敏捷方法中，总是绘制大量的 UML 图，其中需求管理主要使用用例图。用例的优势是细节的同时且具备层次性、整体性，并且可以轻易贯穿开发过程；但劣势是有时候过于复杂、细节过多，任务也需要进一步拆分，并且变更相对困难，影响也可能更大。敏捷方法中，用例图的绘制不再是必要的。可以仅使用用户故事的方式维护需求。

什么是用户故事？故事由三方面组成：卡片（Card）：一个书面的、简短的故事描述，用来做计划并在开发过程中起到提醒的作用；对话（Conversation）：针对故事描述进行的交流，用来澄清故事的细节；确认（Confirmation）：测试，用于表达和归档故事细节且可用于确定故事何时完成。用户故事具体描述是什么样的呢？模板形如作为 X（角色），我要 Y（能做什么），以便 Z（达到什么目的或实现什么价值）。具体的我们在 devcloud 中的应用会在本小节后面介绍。

用户故事的优势与不足。用户故事具有许多优势，例如：用户故事强调对话交流而不是书面沟通；用户故事没有技术术语，能被客户和开发人员同时理解；用户故事的大小适合于做计划；用户故事适合于迭代开发；用户故事鼓励推迟考虑细节；用户故事支持随机应变的开发；用户故事鼓励参与性设计；用户故事传播隐性知识。不过，它也存在着一些不足，如：面对大型项目，用户故事数量庞大，相互关系错综复杂、难以捉摸；需求的可追溯性欠缺；大型团队中隐性知识如果不记录下来，难以得到必要的共享——书面文档与面对面交流的平衡。

如何编写用户故事？基本原则：INVEST。独立的 Independent：避免在故事之间引入依赖关系；可讨论的 Negotiable：用户故事是关于功能特性的简短描

述，仅提醒将来需要交流的内容，不必遵守书面合同或需求文档，不需要包括所有相关的细节；有价值的 Valuable to Purchasers or Users：与客户团队密切合作，由客户来写故事；可估计的 Estimatable：每个故事需要花费多少时间完成；小的 Small：合适的小，根据数据边界、逻辑顺序、不确定性或技术障碍进行拆分或合并；可测试的 Testable：故事完成的标准应当是通过了测试。

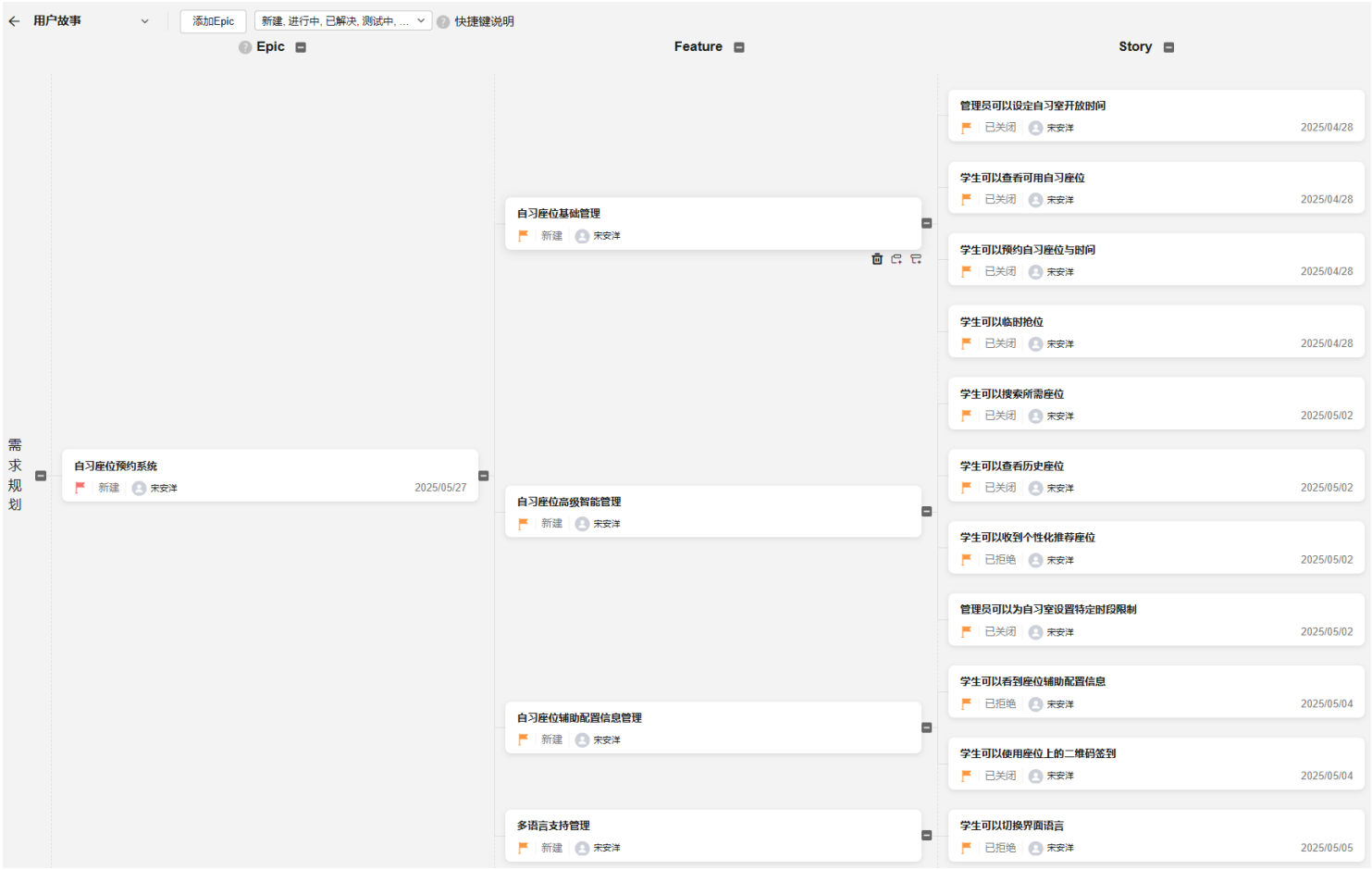

图3.1 用户故事

在华为 DevCloud 中。我们编写的用户故事的整体需求规划如图 3.1 所示。整体来看，第一，我们编写的用户故事是用户中心化的，故事都以“学生可以...”或“管理员可以...”开头；第二，我们编写的用户故事注重功能性，故事描述了具体的功能点，例如“设定自习室开放时间”、“预约自习座位”、“二维码签到”等，这有助于开发团队理解需要实现什么；第三，我们的用户故事具备初步的颗粒度，相对于大型功能模块，我们的故事颗粒度已经有所拆分，单个功能点相对

明确，适合于迭代开发，可以在一个迭代中完成，并提供可交付的价值。从 INVEST 原则的角度来看，我们编写的用户故事尽可能地尝试去满足 INVEST 原则，但缺陷也是依然存在的。第一，Independent。我们尽可能尝试让用户故事间不产生依赖，但依赖是无法避免的，“学生可以查看可用自习座位”和“学生可以预约自习座位与时间”之间存在明显依赖。没有“查看”就无法“预约”。不过，我们这样也使得拆分得尽可能更小更独立，在这里，即便没有“预约”功能，也至少能“查看”展示座位信息。而“预约”则可以作为一个独立的、后续的故事。“学生可以临时抢位”实质上也依赖于“查看可用座位”和“预约”功能，是预约功能的一个补充，但是这个用户故事粒度的设计可能并没有很合理，预约可能 3 个小时完成，但作为扩展补充，抢位可能 1 个小时完成。但是我们也想不出抢位该和谁合并。相比之下，“学生可以切换界面语言”这个故事就是一个很好的设计，系统最初功能都是中文实现，然后多语言的这个故事，我们并没有设计成用户可以切换英语界面、用户可以切换日语界面...而是一个合并的，都归结为其他语言，这是一个不错的设计；第二，Negotiable。我认为这一点我们实现的还可以，故事都以“学生可以...”或“管理员可以...”开头，是简短的功能描述，具备可讨论性。并且严格遵守作为 X 我想要 Y 以便于 Z 的形式，具备简短但明确的可讨论的价值与细节，使得团队能够清楚的最终目标和意义，如图 3.2 所示。并且确定的细节我们将其变成响应的验收测试，具体内容我们将在第四章质量管理中进一步叙述。第三，Valuable to Purchasers or Users。这一条某种意义上是最容易的，同样地，我们严格遵守作为 X 我想要 Y 以便于 Z 的形式，如图 3.2 所示，以便于 Z，可以很好地明确价值，并有助于团队在开发过程中做出更好的决策，并在优先级排序时提供依据。不过知易行难，真的落实到实际开发场景中，能否与客户很好地密切合作，让用户完成价值明确的用户故事，或许并不容易。我们需要牢牢把握一个原则：开发与客户是非对抗的；第四，Estimatable。这点或许我们做的并不好。在第一条中，我们也提到了，我们编写的用户故事的颗粒度实质上有些参差不齐。并且有些故事可能仍然包括多个子任务，导致难以精确估计开发时间，例如“学生可以看到座位辅助配置信息”，这涉及多种配置信息的展示，实际上可能是比较复杂的。此外，还有技术穿刺的问题，个性化推

荐涉及协同过滤等复杂的算法和数据分析，需要一定的时间探索。第五，Small，合适的小。我们从 epic 到 feature 到 user story 逐步拆分，如图 3.1 所示。我们有一些做的比较好的地方，在第一条中也提到过，例如先“查看”后“预约”，这是根据逻辑顺序的功能拆分。但是第四条中，我们也提到有些故事偏大并且有些有技术穿刺问题，说实话，我感觉不太好拆出大小、功能上合适的故事，就没有进一步拆分，就没有尝试再拆了，这也是我们做的不好的地方；第六，Testable。每个用户故事应当有具体的验收标准。可测试这一条我认为我们做的还可以，具体内容将在第四章质量管理中进一步详细叙述。

图3.2 用户故事描述信息示例

3.2 迭代

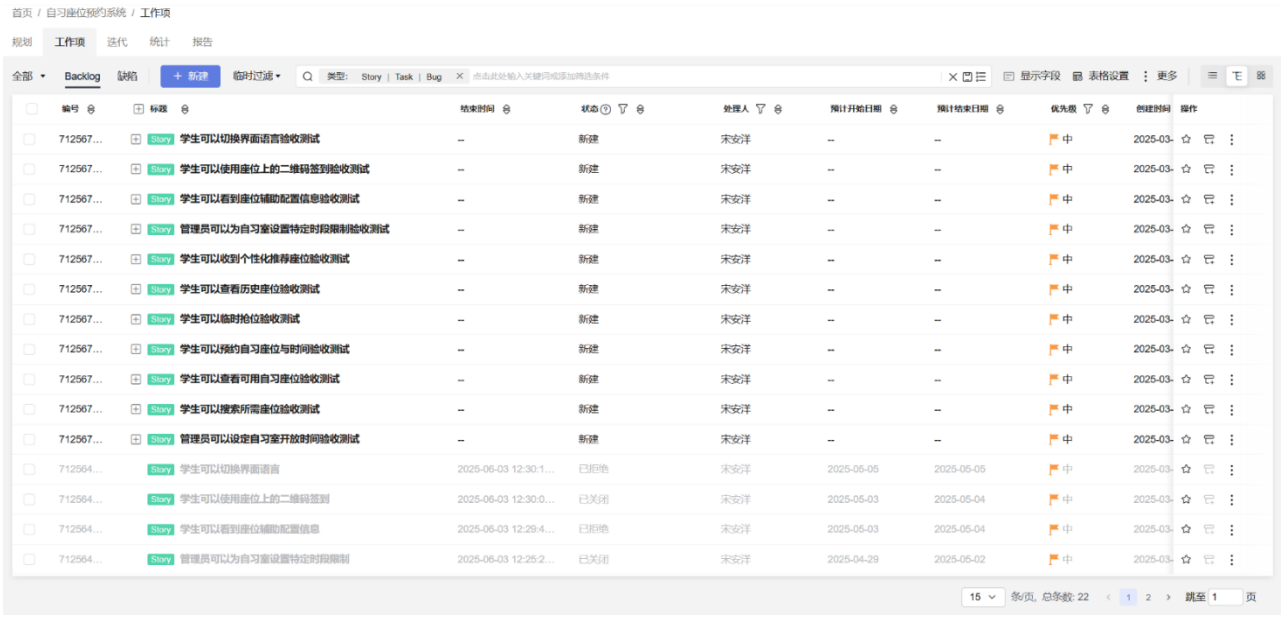

图3.3 Product Backlog

我们的 Product Backlog 如图 3.3 所示，其中的内容即全部的用户故事。其中验收测试用例也是在 devcloud 用户故事的思维导图规划中编写的。因此你在上图的 Product Backlog 中也会看到验收测试相关的工作项。对于验收测试，具体的测试用例，这里不做详细叙述，测试相关内容将在第四章质量管理中进一步叙述。

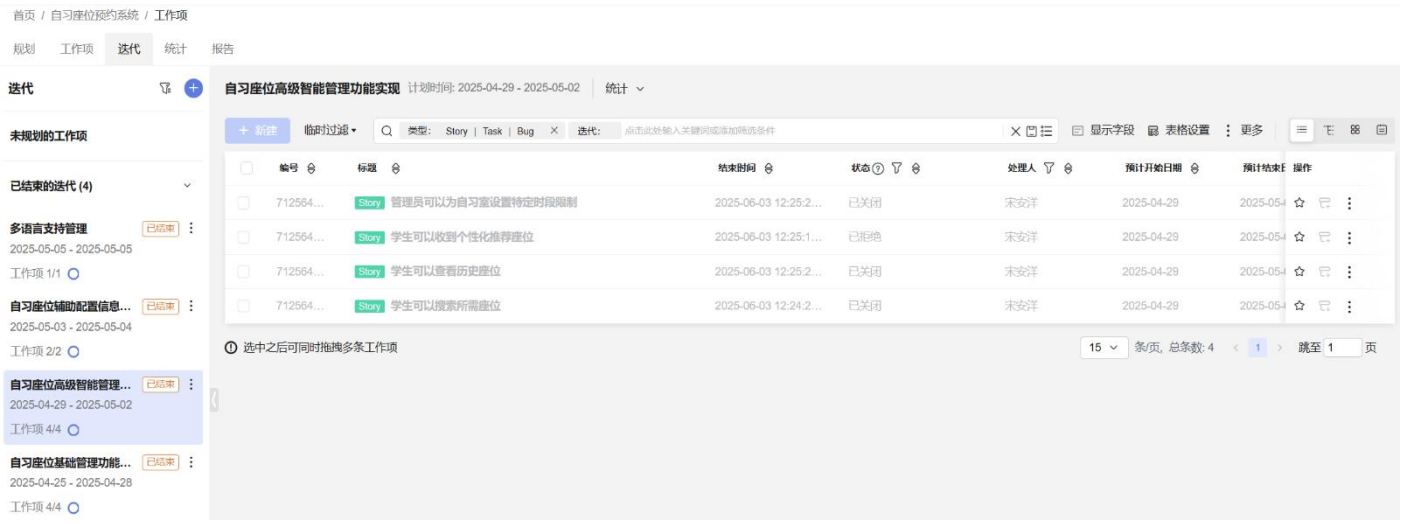

图3.4 迭代 Sprint Backlog 内容示例

我们的 Sprint Backlog 如图 3.4 所示。我们一共有四次迭代，每次迭代选取的用户故事都是集中于解决一个 feature。第一次迭代完成了自习座位基础管理的用户故事对应的系统功能开发，耗时 4 天。第二次迭代完成了自习座位高级智能管理的用户故事对应的系统功能开发，耗时 4 天，图 3.4 中展示的详细便是本次迭代，其中个性化座位推荐这个用户故事因为技术穿刺的问题没有时间做了被拒绝取消。第三次迭代完成了自习座位辅助配置信息管理的用户故事对应的系统功能开发，拒绝取消了配置信息这个用户故事，因为需求大且不明确且复杂且并非核心功能，仅保留了签到，耗时 2 天。第四次迭代完成了多语言支持管理的用户故事对应的系统功能开发，拒绝取消了该用户需求，用途不大只是锦上添花或者说实话其实是懒得做，耗时 1 天。

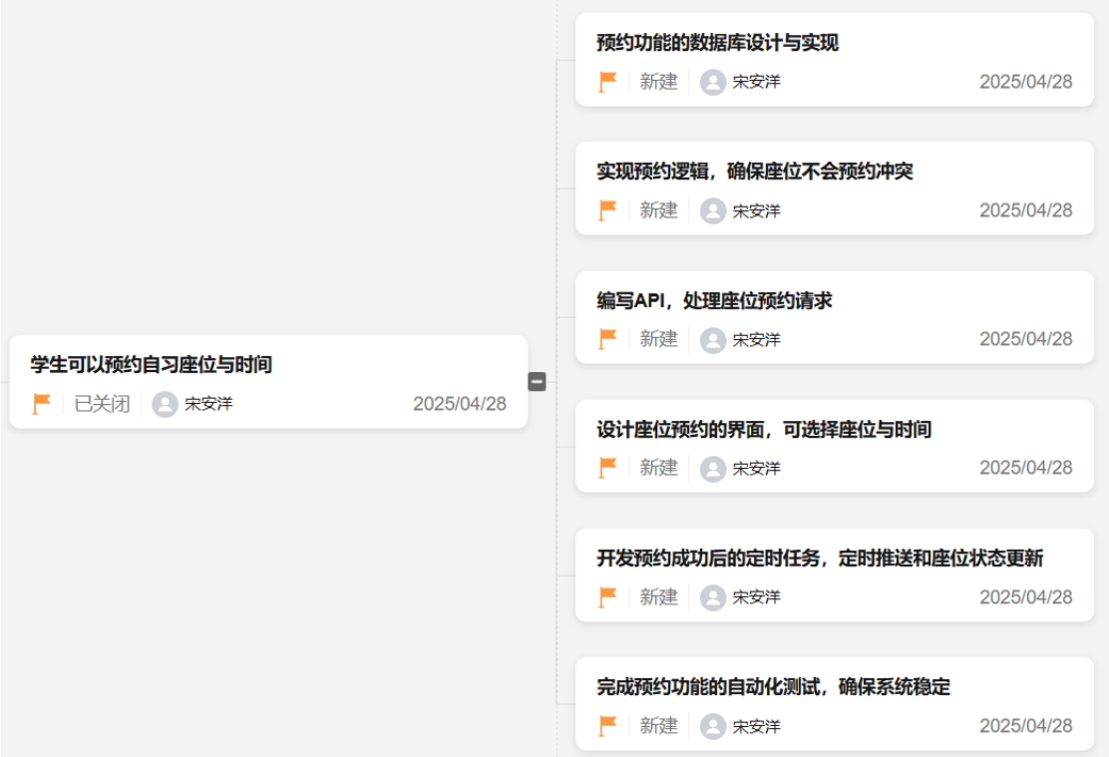

图3.5 任务分解

在第 2.2 节中我们介绍过 Scrum 过程模型，在每次迭代开始的时候，我们对选取的用户故事进一步完成任务分解。用户故事是以用户为中心，主语是用户，是业务性的。而任务是开发人员分解出来的，以开发者为中心，是技术性的。如

图 3.5 所示，以学生可以预约自习座位这个关键的用户故事为例，我们分解出六个任务，这六个任务是分职能、分阶段、面向开发者的，从数据库到后端服务到 controller 到前端，然后到定时和推送这个技术难点进一步完善预约功能的逻辑，最后到测试。

4 质量管理

4.1 软件评审与静态扫描

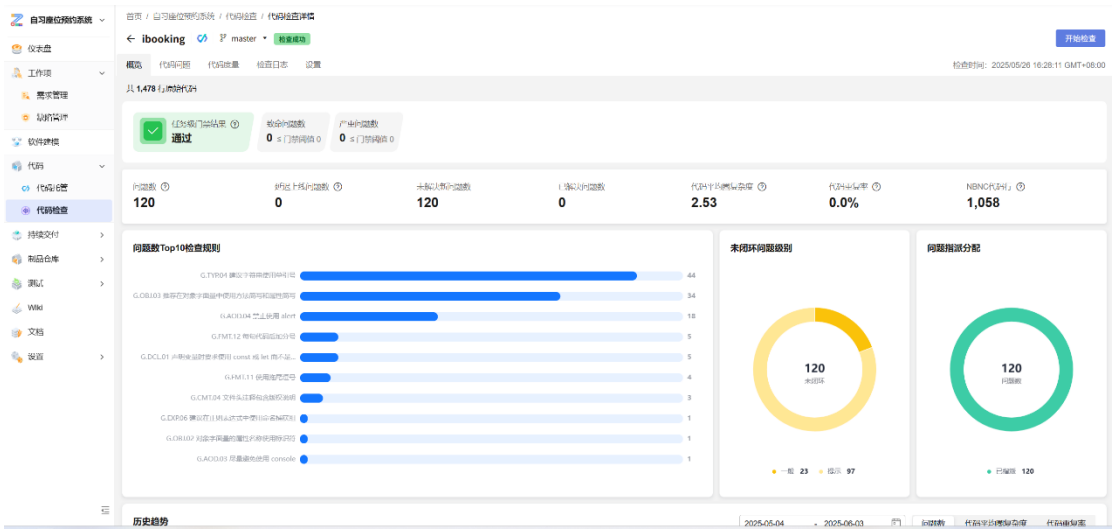

图4.1 代码检查

Karl E. Wiegers 曾指出：“不管你有没有发现他们，缺陷总是存在，问题只是你最终发现他们时，需要多少纠正成本。”“评审的投入把质量成本从昂贵的后期返工转变为早期的缺陷发现。”为了尽早从软件工作产品中识别并消除缺陷，从而提高软件开发质量和效率，应运而生的便是软件评审了，主要方式是通过文档阅读、讨论等进行主观判断。软件评审也会带来许多其他好处，包括促进质量文化、促进优秀开发实践的交流和传播、激发团队合作精神。软件评审的类别包括：管理评审、同级(同行)评审、项目总结评审、状态评审。同级评审方法从最正式到最不正式包括：审查、小组评审、走查、结对编程、同级桌查/轮查、临时评审。在本项目中，原计划我们打算结对编程，由李天佑同学作为 Navigator，协助我的开发，但考虑到他需要完成自动化测试任务，因此本项目中我们没有采

用人工评审，只是简单的静态扫描，使用的是 DevCloud 的代码检查功能，代码检查结果如图 4.1 所示。

DevCloud 代码检查中，问题级别一共有四级：致命、严重、一般、提示。默认的质量门禁是致命问题数<=0，严重问题数<=0。所以这里我们就没有对代码检查中出现的问题进行处理，总体来说是没问题的。我们的代码中提示级问题有 97 个，一般级问题有 23 个。一般级问题中，检查出的问题主要都是同一个问题，都是 vue 前端代码“G.AOD.04 禁止使用 alert”，描述中说明“JavaScript 的 alert、confirm 和 prompt 被广泛认为是突兀的 UI 元素，应该被一个更合适的自定义的 UI 界面代替。此外，alert 经常被用于调试代码，部署到生产环境之前应该全部删除。”正确示例中指出应当用自定义方法如 customAlert 等。对于我们当前的项目我们认为其实保持 alert 也问题不大，所以就忽略掉了。然后如图 4.1 所示，我们的代码平均圈复杂度为 2.53，平均圈复杂度在 1 到 5 是极低风险，在代码编写的时候我们有注重这件事情，一个函数只实现单一功能，形成良好规范，提高可维护性和可测试性。

4.2 自动化测试

自动化测试是很重要的。但是在本项目中我们做的实质上并不好。自动化测试常用于测试驱动开发（Test-Driven Development，TDD）。TDD 是一种软件开发方法论，它强调先写测试再写实现代码。TDD 有着许多好处，包括：第一，可以带来更清晰的需求理解。写测试用例相当于先定义系统行为，有助于开发者更好理解需求；第二，有助于减少缺陷。TDD 驱动下，开发过程自动覆盖大量测试路径，从源头减少 Bug；第三，可以使代码更容易维护。小步重构的文化，促使代码更具可读性和扩展性；第四，有助于使得项目整体设计上更优。TDD 促使开发者写出低耦合、可测试的代码，自然推动面向接口编程；第五，在开发人员编写代码的过程中可以得到快速的测试反馈，往往可在数秒内发现问题，而不是等集成时才发现。除了应用于 TDD，自动化测试本身也是非常有意义的一种技术手段。一方面，可以降低人力成本，一次编写，反复执行，长期节约人工测试投入，且因比手工快很多，还能频繁验证系统行为是否正确。另一方面，有助于持续集

成，随着新功能的不断集成、或者原有功能的更改，随之而来的是引入新问题的风险，自动化测试可以在每次修改代码后自动运行一整套测试，从而减少风险，这是非常非常有意义的。在本项目中，我们没能在开发前就完成自动化测试代码的编写，我们没有很好地贯彻实施 TDD 方法论，但是在此之后对自动化测试部分进行了完成也依然是非常有意义的，如果未来需要继续去改动、去集成，自动化测试在其中的重要作用是毫无疑问的。

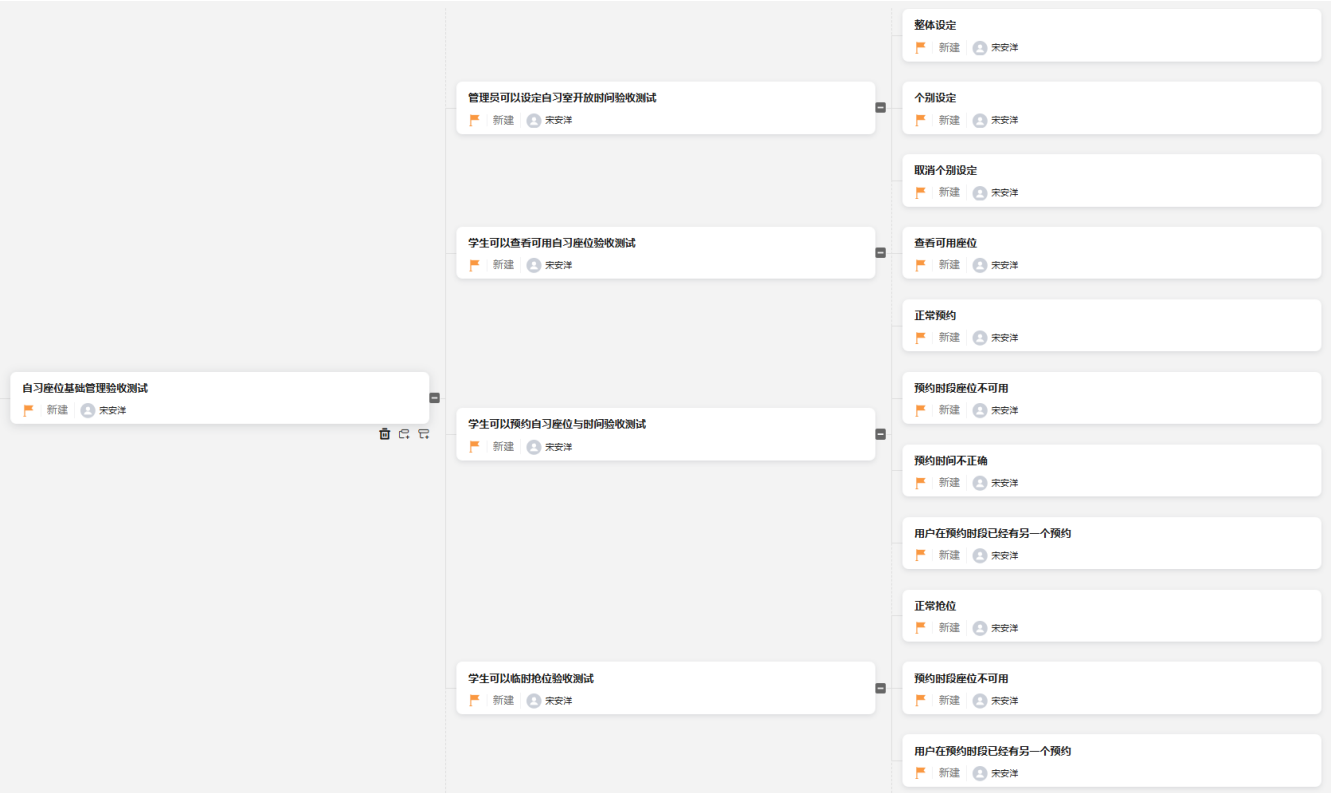

图4.2 自动化验收测试用例示例

图4.3 自动化验收测试用例描述信息示例

如图 4.2 和 4.3 所示是我们的自动化验收测试用例和详细描述信息示例。图 4.2 中给出的是自习座位基础管理这个 feature 下的 4 个用户故事对应的验收测试用例。我们的验收测试用例尽可能全面考虑各种需要特判的场景，尽管我们的测试代码没能尽早完成，我们的测试用例在开发前就写好了，用来指导我们的开发过程，这是非常有意义的。对各种特判情况的尽快能全面的考虑，例如如图 4.2 所示，学生可以预约自习座位这个用户故事，我们的验收测试用例包括 4 个场景，正常预约、预约时段座位不可用、预约时间不正确、用户在预约时段已经有另一个预约。场景描述的写法我们遵循给定 X 当 Y 那么 Z 的写法，细化到特定场景对应的确定的、可执行的输入输出样例。如图 4.3 所示，例如用户在预约时段已经有另一个预约这个场景，我们的详细描述信息是：

给定 用户 A，座位 S，且用户 A 在 2025 年 5 月 1 日 10:00-11:00 已有另一个预约，且当前时间为 2025 年 5 月 1 日 8:00。

当 A 预定 S，预定时间段为 2025 年 5 月 1 日 9:00-11:00。

那么 预约失败。

单元测试时只需要完全参照我们的测试用例描述信息进行实现即可。

如图 4.4 所示是我们的自动化测试程序的一个展示。一个正确的自动化测试的做法是，针对你特定的场景，拟定的输入输出样例，然后你预期这个输出结果会是什么样的，然后用 assert 语句进行断言。与预期输出不符，就会报错 AssertionError，这样就可以快速知道问题是什么了，如果没有报错，全部通过了，那就是自动化测试通过了。如图 4.4 所示，我们针对各个 controller 的各个 api 的各个场景，完成了一些测试函数，然后在 main.py 中像写故事那样 CRUD 调用一遍，来实现我们完整的自动化测试。

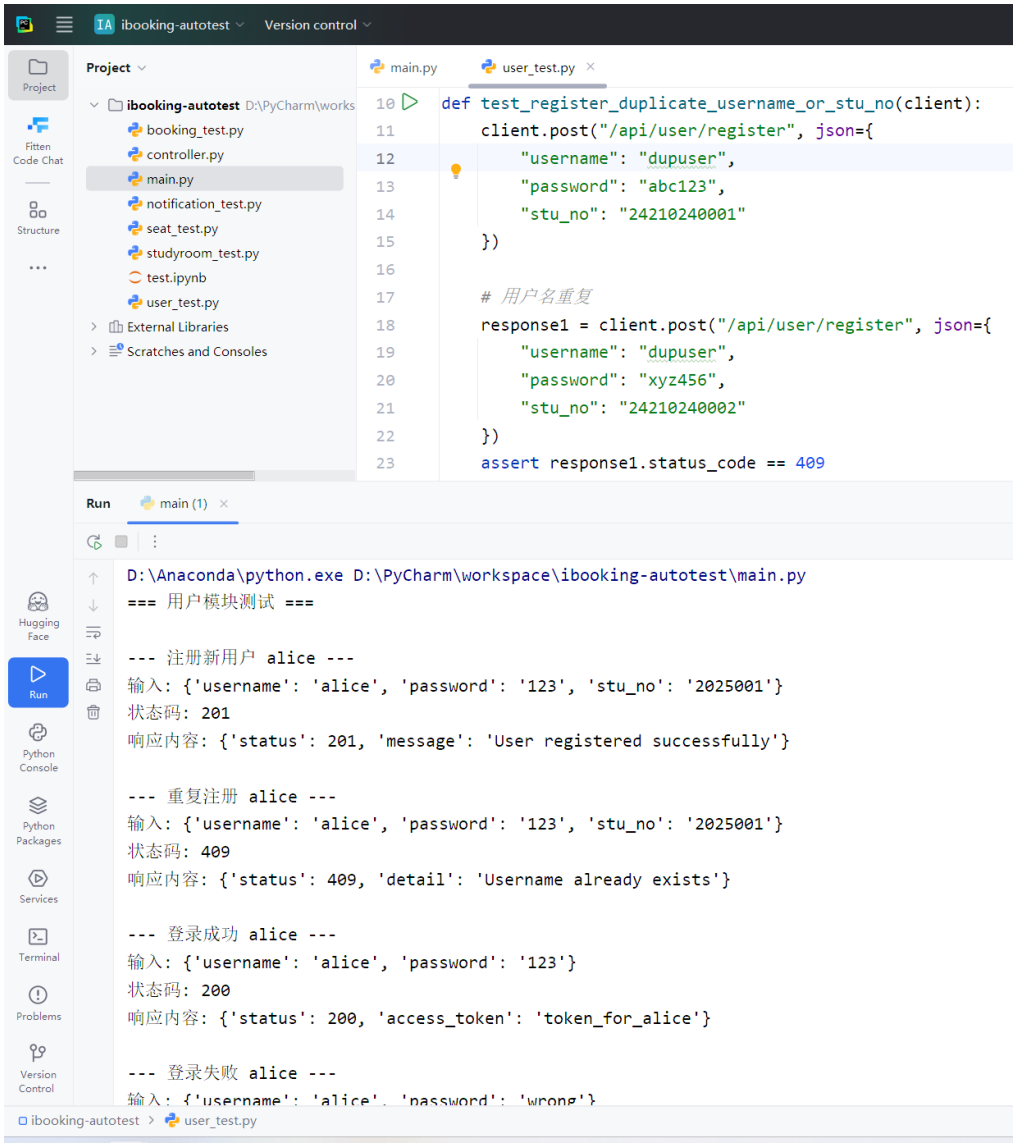

图4.4 自动化测试程序展示

用户在预约时段已经有另一个预约

```
booking_data_other = {"user_id": "2",
                "seat_id": "939753dc5ef14733a16345ceceac95f9",
                "begin_time": "10:00:00",
                "end_time": "11:00:00"}
booking_data = {"user_id": "2",
                "seat_id": "939753dc5ef14733a16345ceceac95f9",
                "begin_time": "9:00:00",
                "end_time": "11:00:00"}
test_booking_user_booking_exist(booking_data, booking_data_other, show_detail=True)
```

✓ [6] < 10 ms

状态码：409
响应内容：{'status': 409, 'message': '预约失败！该预约时段您已有其他预约！'}

图4.5 定制化的自动化测试程序展示

除开 main.py 完整测试整个系统，实际上你应该实现分开的部分的单元测试。特别是，考虑到有时候你可能想更自由地去自动化测一部分或者某些部分，然后也是方便调整和观察，所以这时候就可以充分发挥 python 脚本语言的特性了，我们可以在 notebook 里更加自由地测试和观察。如图所示是“用户在预约时段已经有另一个预约”这个验收测试用例，可以看到 assert 是通过了的，没有报错，并且也展示了响应结果方便我们观察。

4.3 持续集成

持续集成简单来说就是用好 github，当然，在这个项目中我们使用的是 DevCloud 的代码托管功能，如图 4.6 所示。

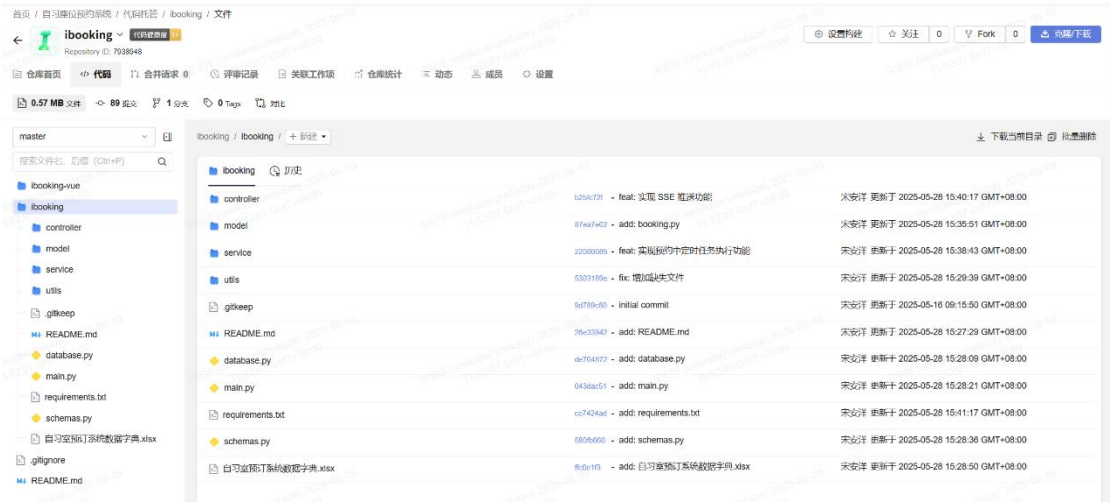

图4.6 代码托管示例

github 使用的过程中，在我们看来最重要的最核心的功能有两个，一个就是 commit 不断持续集成，另一个是分支与合并，这二者构成了分布式版本控制与管理的关键，是项目尤其开源项目中大家能够分工合作共同贡献独立而不排斥并能高效解决冲突的关键。对于分支与合并，在 github 中，我们可以通过 fork 得到分支，也就是创建仓库的一个副本。然后你可以 clone 到你本地，进行修改，实现你的功能，然后 commit 到你的分支里。然后你可以在你的分支中 Pull requests 来请求合并，原仓库的管理者来处理你的 request 决定 merge。由于本

项目由我单独开发，没有和小组成员的协作，因此在 DevCloud 中我们没有分支与合并的这个过程。不过持续集成过程中，我们有注重于 commit 的 comment 的规范性。一个规范的 comment 可以使得提交修改的内容便于理解，有助于生成变更日志和版本发布，有助于简化版本回滚和修复，便于团队协作与沟通、代码审查和质量控制，提高持续集成过程的质量与效率。如图 4.6 所示，我们的 comment 遵循了较为良好的规范，保证每次提交易于理解，每次提交限于完成一次逻辑功能，针对于一个分解出来的 task。

4.4 部署流水线

在第 2.3 节中我们提到过 DevOps 的重要性。开发团队主要关注于持续集成，然而这时候问题就出现了。例如，构建和运维团队的人员一直等待说明文档或缺陷修复；测试人员等待“好的”版本构建出来；新功能开发完成几周后，开发团队才收到线上缺陷报告；开发快完成时才发现当前软件架构无法满足系统的一些非功能需求。因此，我们需要更完整的端到端的方法来交付软件，我们需要持续部署，我们需要一个端到端的拉式的部署流水线系统。

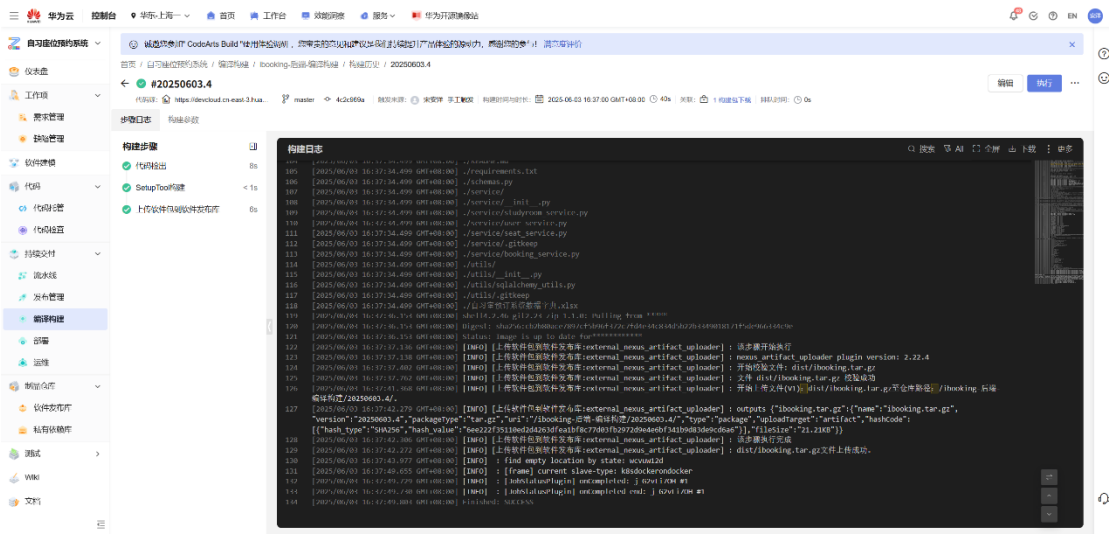

图4.7 编译构建示例

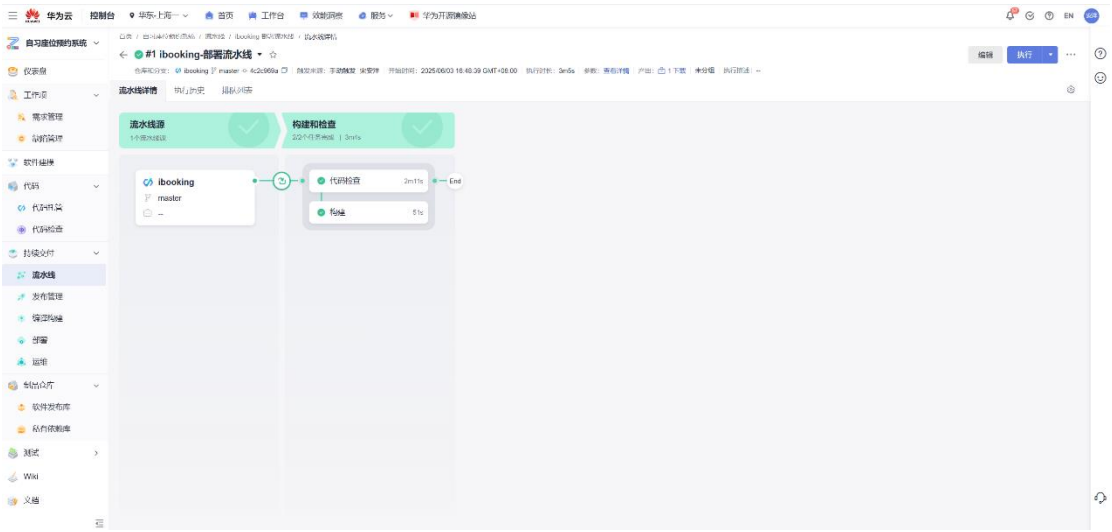

图4.8 流水线示例

如图 4.7 所示是编译构建，将项目文件打包为软件包以用于发布，在 DevCloud 中左侧制品仓库->软件发布库中可以看到打包好的软件包。然后在部署的时候我们可以配置环境，并使用软件包进行部署。由于没有服务器，在这里我就没有通过 DevCloud 进行部署，在本地简单尝试了一遍部署，并形成了一份文档，报告中就不细叙了。然后整个部署过程应当形成一个部署流水线，如图 4.8 所示。当然，由于我们没有在 DevCloud 中进行部署，所以流水线阶段中没有包含部署这个过程。

5 系统分析与设计

Scrum 的理念是“Just enough design, just in time”。在 Scrum 项目中，UML 图、ER 图等的绘制不是必要的。这些图的绘制应当是为了增强沟通和对需求的理解，而不是“流程产物”。不过，我认为 UML 图尽管看起来有点多余，但是数据库设计还是非常有必要的。于是我们这里就不对系统整体进行学术化的分析与设计，但是仅对数据库进行了详细的设计。

5.1 数据库设计

在 Scrum 项目中，或许一个数据字典就足够了，只要足够清晰表达实体和关系即可，不用追求学术化的 ER 图。为了直观清晰，这里我们还是简单绘制了一下 ER 图。由于 Chen Notation 椭圆太多，看着很乱，这里我们采用的是 Crow’s

Foot Notation。当然实际上，我基本不需要画或者说额外做什么，Navicat 右上角 DDL（Database Definition Language）按钮可以直接查看建表语句，然后 draw.io 可以直接导入建表语句快速绘制，然后我只需要自己连线连出关系或者用 Mermaid 代码来实现这个关系的绘制，ER 图如图 5.1 所示。

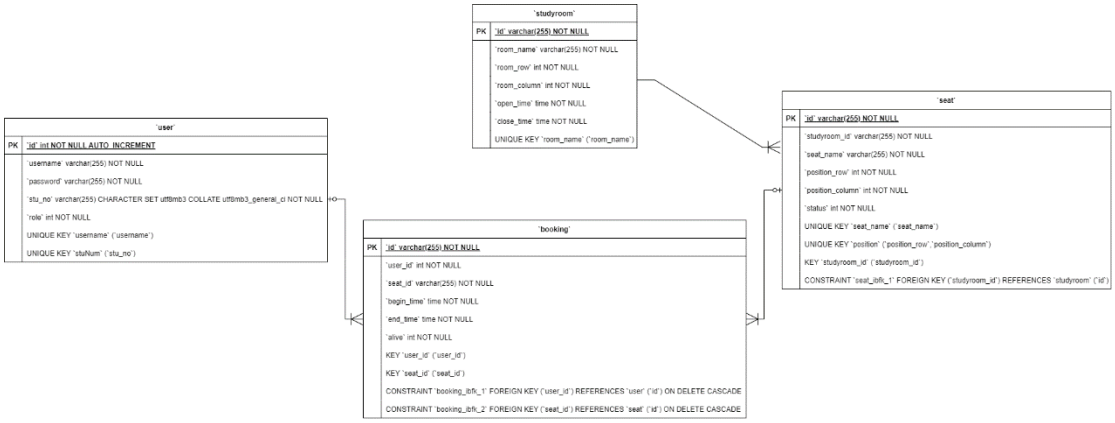

图5.1 ER 图

Table Name	Column	Type	Comment
user	id	int	自增主键 id
	username	String	用户名。非空。唯一
	password	String	密码。非空
	stu_no	String	学号。非空。唯一
	role	int	角色（用户 0，管理 1）。非空
studyroom	id	String	主键 uuid
	room_name	String	自习室房间名。非空。唯一。如 JB301
	room_row	int	自习室座位行数。非空
	room_column	int	自习室座位列数。非空
	open_time	Time	自习室每天开放时间（时分秒）。非空
	close_time	Time	自习室每天关闭时间（时分秒）。非空
seat	id	String	主键 uuid
	studyroom_id	String	外键 studyroom 的 uuid。非空。ON

			DELETE CASCADE
	seat_name	String	座位名称。非空。唯一
	position_row	int	座位行号。非空。
	position_column	int	座位列号。非空。(行,列)唯一
	status	int	当前座位状态（1 未占用，2 已占用，3 已签到）。非空
booking_yyyymmdd	id	String	主键 uuid
	user_id	int	外键 user 的 id。非空。ON DELETE CASCADE
	seat_id	String	外键 seat 的 uuid。非空。ON DELETE CASCADE
	begin_time	Time	预约开始时间（时分秒）。非空
	end_time	Time	预约结束时间（时分秒）。非空
	alive	int	0 表示已过期，1 表示未过期。非空

表 5.1 数据字典

ER 图对应的数据字典如表 5.1 所示。其中，seat 表要设置定时器，每日零时要把 status 全部更新为 1；booking_yyyymmdd 是日分表 yyyymmdd，需要设置定时器，每日生成一个可预订表 booking_yyyymmdd。关系上，如图表 5.1 所示，实体集 studyroom 和 seat 是一对多关系，不需要联系集，seat 中外键是 studyroom.id。这是结论，或者从数据库基础原理来证明一下，多方做主键，联系集主键是 seat.id，外键是 studyroom.id，联系集和实体集 seat 主键完全相同，可以合并到 seat，于是不需要单独的联系集。实体集 user 和 seat 是多对多关系，因此需要联系集 booking。user 和 booking 是一对多关系，seat 和 booking 也是一对多关系。booking 主键是 booking.id，外键分别是 user.id 和 seat.id。

6 系统开发

我们的系统开发，数据库使用的是 mysql。后端使用的是 python，使用的框架包括数据库持久层框架 sqlalchemy、Web 框架 FastAPI、定时任务调度框架

APScheduler、推送框架 SSE。前端框架 vue。本章节首先简单展示一下我们的系统，然后简单探讨一下开发过程中的两个技术难点。

6.1 系统展示

如图 6.1 所示是我们系统其中一个前端界面的展示。图中展示了未在规定时间内完成签到，然后出现了模态框提醒，同时会记录违约并自动取消预约。有关系统的更多展示在我们录制的演示视频中。

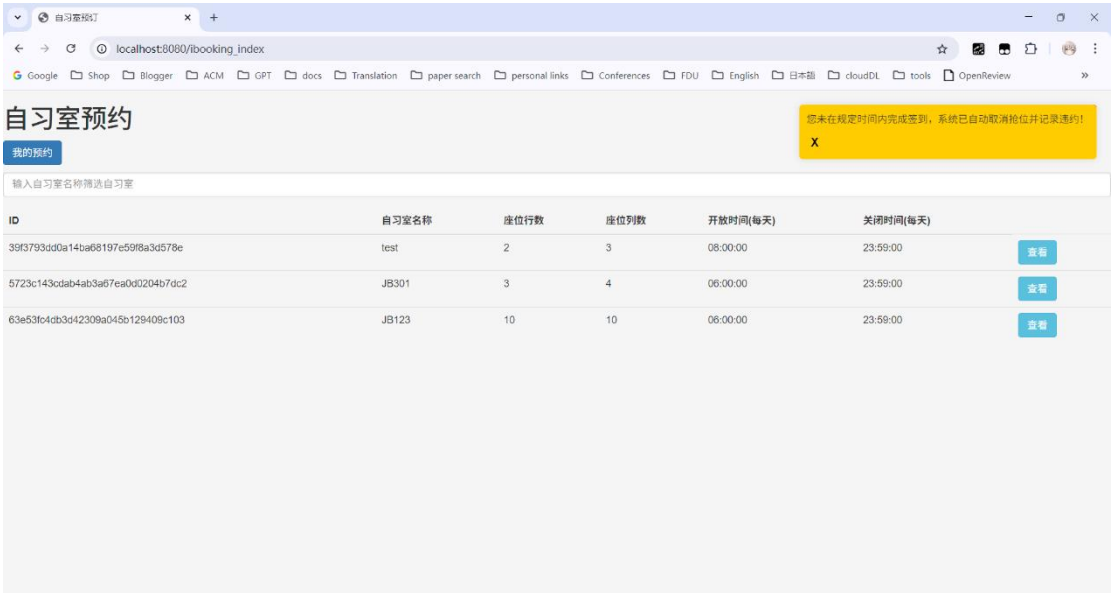

图6.1 系统展示图

我们实现的系统功能对应于第三章需求管理中叙述的用户故事。参考第 3.2 小节迭代关闭掉的功能，有些功能被我们放弃了，例如个性化座位推荐、座位辅助配置信息管理等。在实现基本 CRUD 的基础上，我们对预约功能进行了详细的实现与完善，其中有两个技术难点，分别是定时任务调度和推送，在下一小节中我们将具体探讨一下对这两个技术难点的理解。

6.2 技术难点

6.2.1 定时任务调度

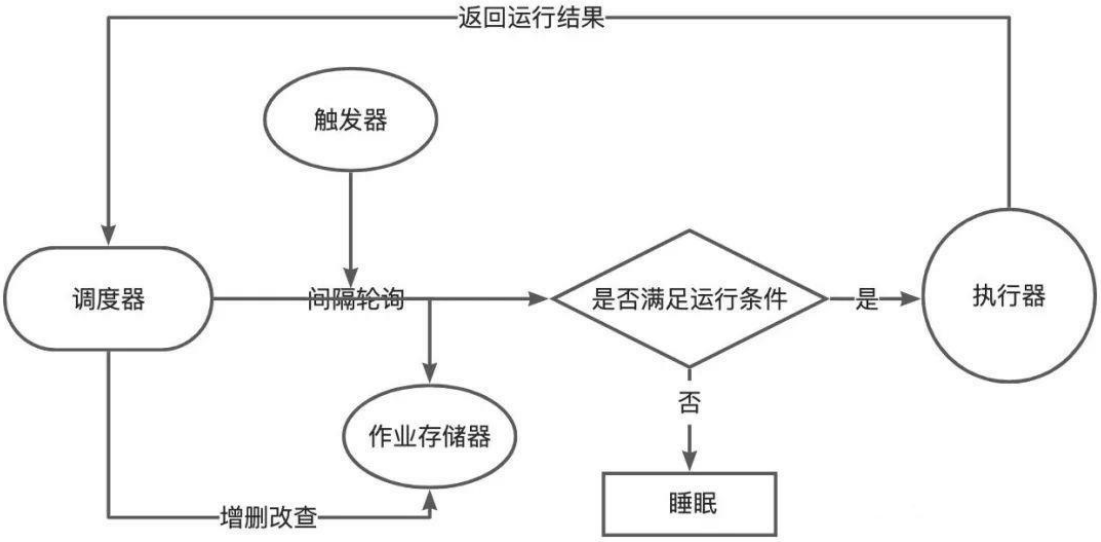

图6.2 APScheduler 定时任务调度运行原理流程图

尽管简单地使用 sleep()、threading.Timer 等方法也是可以实现定时任务功能的，但怎样都是不完美，效果差，太耗费资源。如果我们自己尝试实现一个高质量的定时任务调度，这是很麻烦的。于是我们在 python 中使用的是现成的 APScheduler 框架，类似于 Java Springboot 中的 @Scheduled 注解，调度在框架内部被进行管理，调度机制本质是定时任务的注册和触发机制。而这个过程对定时任务的处理与操作系统进程的三态是很相像的。具体来说，APScheduler 启动时会首先启动一个调度线程池，作为调度主线程。在这个线程池中，任务的执行会被安排到线程池中的线程上运行。这个线程池并不是单一的主线程，而是包含了多个工作线程的池子。首先，我们可以用调度器（Scheduler）对作业 CRUD，作业（Job）将被存储在作业存储器（Job Store）中。然后，主线程的这个调度器会间隔一定时间轮询，对作业短暂唤醒，对比各个作业对应的触发器（Trigger）定下的规则，看是否满足运行条件。例如，预订了一个要在 10 点运行作业的 cron 表达式，轮询时判断系统时间到没到 10 点。如果没满足，作业就会恢复睡眠状态，如果满足了，取出作业，在执行器（Executor）中执行并反馈结果。

6.2.2 推送

客户端和服务器间实时通信主要包括四种方式：短轮询、长轮询、SSE（Server-Sent Events）、WebSocket。

短轮询是最简单最基础的一种方式，客户端每隔一个固定间隔时间向服务器发送请求，询问是否有新的数据或者事件。轮询间隔较短，则对服务器压力极大，轮询间隔较长，则消息延迟。因此短轮询更适合于对实时性要求极低的场景。

长轮询是对短轮询的一种改进，在间隔一定时间发送请求的基础上，客户端请求后，如果服务器没有数据，服务器不会立即返回，而是保持连接直到有新数据再返回。长轮询可以基本实现较低的延迟，但是服务器需要保持多个长时间打开的连接，依然有较高的资源消耗。长轮询一般适用于客户端不支持 SSE 或 WebSocket 的场景，比如早期的许多浏览器、聊天应用等。

短轮询和长轮询技术中，服务端是无法主动给客户端推送消息的，都是客户端主动去请求服务端来获取最新的数据的。而 SSE 是一种可以主动从服务端推送消息的技术。相比于短轮询和长轮询的不断轮询，SSE 则是客户端建立好了一个单向的 HTTP 长连接，然后通过接受 EventSource 事件流的方式一直等待着服务器发过来的新数据。SSE 的实时性是不错的，并且服务器负载是很低的。

WebSocket 是一个双向网络通信协议，可在单个 TCP 连接上进行全双工、持久化的通信，位于 OSI 模型的应用层。WebSocket 使得客户端和服务器之间的数据交换变得更加简单，只需要完成一次握手，二者之间就可以创建持久性的连接，并进行双向数据传输，直到主动断开。尽管 WebSocket 对服务器的负载也是较高的，但是支持双向通信并且实时性极好。WebSocket 适用于需要高频、双向通信的应用，如实时游戏、在线聊天、协作工具、金融交易等。

在本项目中，只需要服务器向前端单向推送提醒通知、超时违约与自动取消通知等消息，也不需要实时游戏那种最高的实时性。因此我们选择使用的是 SSE 作为推送过程中的实时通信方式。

7 总结

软件开发的管理方法和技术经历了从瀑布式到敏捷开发的演变，而 DevOps 的出现进一步推动了开发与运维的一体化。敏捷开发通过迭代式、增量式的开发模式，提高了开发的灵活性和响应速度，使得团队能够迅速适应变化的需求，交付更高质量的软件。而 Scrum 作为敏捷开发的核心框架，通过清晰的角色定义、工件管理和迭代反馈机制，确保了项目的透明度、团队的协作与持续改进。DevOps

则通过自动化和持续集成实现了开发与运维的无缝衔接，大大提高了软件交付的效率和质量。在本文中我们深入分析了从需求管理到质量管理的方方面面，深入探讨了我们是如何将这些理论实际应用到我们自习座位预约系统开发与测试的软件过程管理的完整实践过程中。结合 Scrum 和 DevOps 的实践，开发团队能够有效管理项目需求、进行持续集成与部署，从而更好地满足客户的需求和市场的变化。通过包括在华为 DevCloud 平台上的实际应用，我们看到了这些方法在真实项目中的巨大潜力。未来，希望随着技术的不断发展和方法论的不断完善，软件开发的流程能够向更高效、灵活、可靠的方向继续不断迈进。