软件工程

Software Engineering

极夜酱

目录

Chapter 1 需求工程

1.1 软件开发

1.1.1 软件开发

什么是好的软件？一个好的软件需要满足三点要求：

1. 一般要求（general requirements）
2. 法律要求（legal requirements）
3. 具体要求（specific requirements）

一般要求包括软件系统需要有详细的文档、可读性高的代码、易于修改和维护的代码、能够开展测试、容易移植、使用方便等方面。

法律要求表示系统必须要遵守相关的法律法规。例如加拿大安大略省制定了残疾人无障碍法案，要求软件必须考虑照顾到残障人士的使用；加拿大反垃圾邮件立法规制了垃圾邮件、间谍软件、恶意软件和僵尸网络等问题。

具体要求即来自客户和用户具体的需求，软件系统应该要能够实现客户和用户提出的功能。

想要开发出一款好的软件，必须要拥有良好的编程能力，这可以通过大量练习提高。同时也需要具备良好的沟通和规划能力，例如在处理问题时，应该与客户或者熟悉问题的人沟通，确保理解了客户/用户的需求。编程前也可以先查找别人已经做过的相关内容，不要重复造轮子。

1.2 需求（Requirements）

1.2.1 用户故事（User Story）

用户故事是用几句话描述用户会做的一件事，以及他们为什么要这样做。用户故事是关于用户将会做什么，而不是怎么做。

用户故事一般采用这样的格式："As a [user type], I want [some action] so that [some reason]."。例如，"作为一个活跃用户，我想不用每次登录都输入账号密码，以便更方便快速地登录。"

1.2.2 需求

需求描述了软件做的某一件事，它可以是功能性（functional）或非功能性（non-functional）的。同样一个需求只需要描述做什么，而不是如何去做。

需求的作用是为了让开发者和客户都清楚最终产品想要的预期是什么。

一个好的需求应该具备以下条件：

- 分类（categorized）：使用 MuSCOW 法则对需求进行分类。
  - MUSTS：系统能够满足客户的基本需求，类似于最小可行产品（minimum viable product）
  - SHOULDS：满足客户基本或更高级的需求
  - COULDS：能够使系统变得更好的额外需求
  - WON'TS：系统不该做的事情
- 有优先级（prioritized）：对不同的需求指定一个优先级，需要遵守 MUSTS > SHOULDS > COULDS > WONT'S。
- 逻辑合理：需求之间的依赖关系需要合理，例如一个 MUST 需求不能依赖于一个 SHOULD 需求。

- 时间估计（time estimate）：一个需求应该需要在 15 天内完成，如果不能完成，应该把它分解成多个小需求。

1.3 原型设计

1.3.1 原型设计（Prototyping）

原型是指没有功能的模型，它不一定要包含一个程序的所有方面。

原型设计是与客户/用户沟通的一种工具，让客户/用户感觉他们自己也是设计者。原型设计的目的是为了验证 UI 是否能够可以被正常使用，让开发者能够以用户的角度思考问题。

1.3.2 纸上原型（Paper Prototyping）

纸上原型也称低保真原型（low fidelity prototyping），通过纸笔等工具设计，通常是 UI 设计的第一步。它具有制作过程快速、成本低、更改快速、效率高的特点。

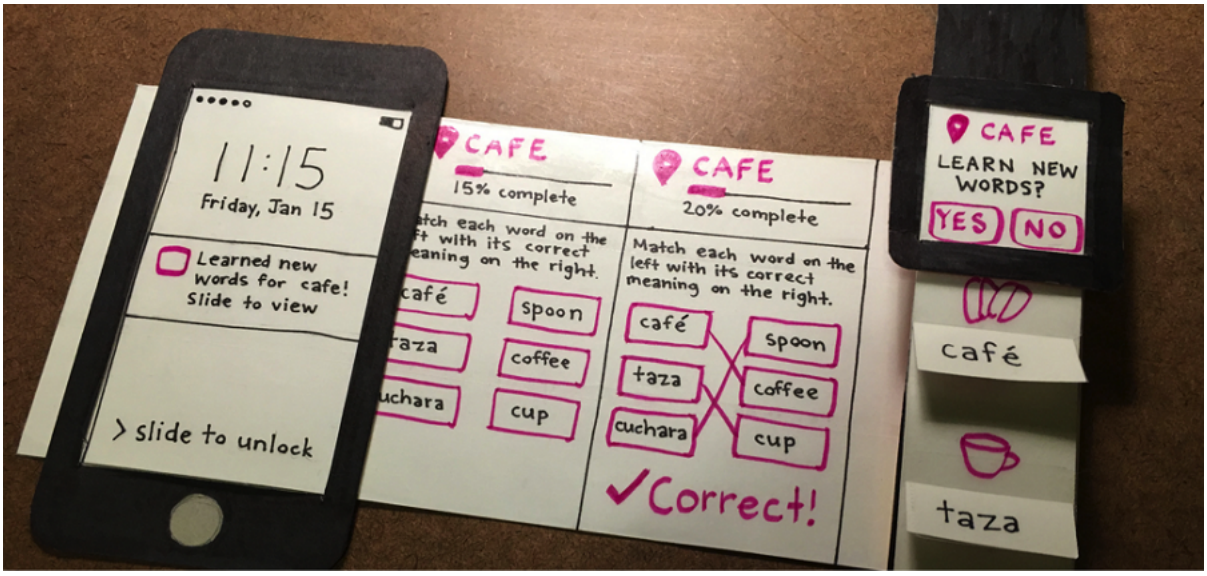

图 1.1: 纸上原型

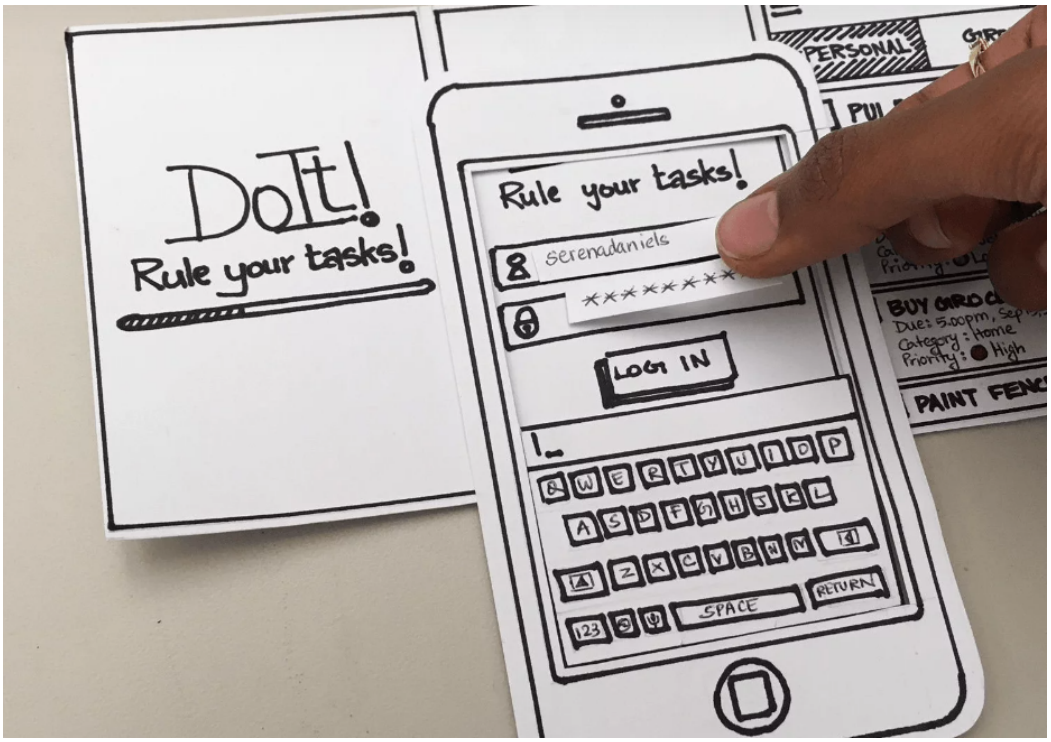

图 1.2: 纸上原型

纸上原型的目的并不是为了让 UI 更加美观，而是为了确保系统的流程是否存在问题。

在完成纸上原型后，需要让参与测试的人根据用例进行测试。在测试过程中，不要提供任何帮助或者指导，以此来验证用户交互是否流畅。

最后，记录下在纸上原型测试中哪些地方是正常的、哪些地方存在问题或遗漏的。

1.3.3 线框图（Wireframing）

线框图也称高保真原型（high fidelity prototyping），它看上去非常接近最终产品，几乎完全按照实物来制作，原型中甚至包含产品的细节、真实的交互、UI 等。

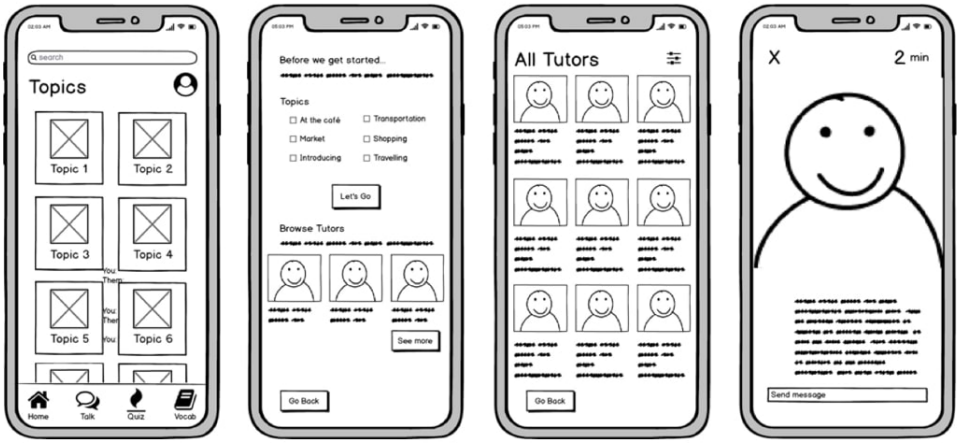

图 1.3: 线框图

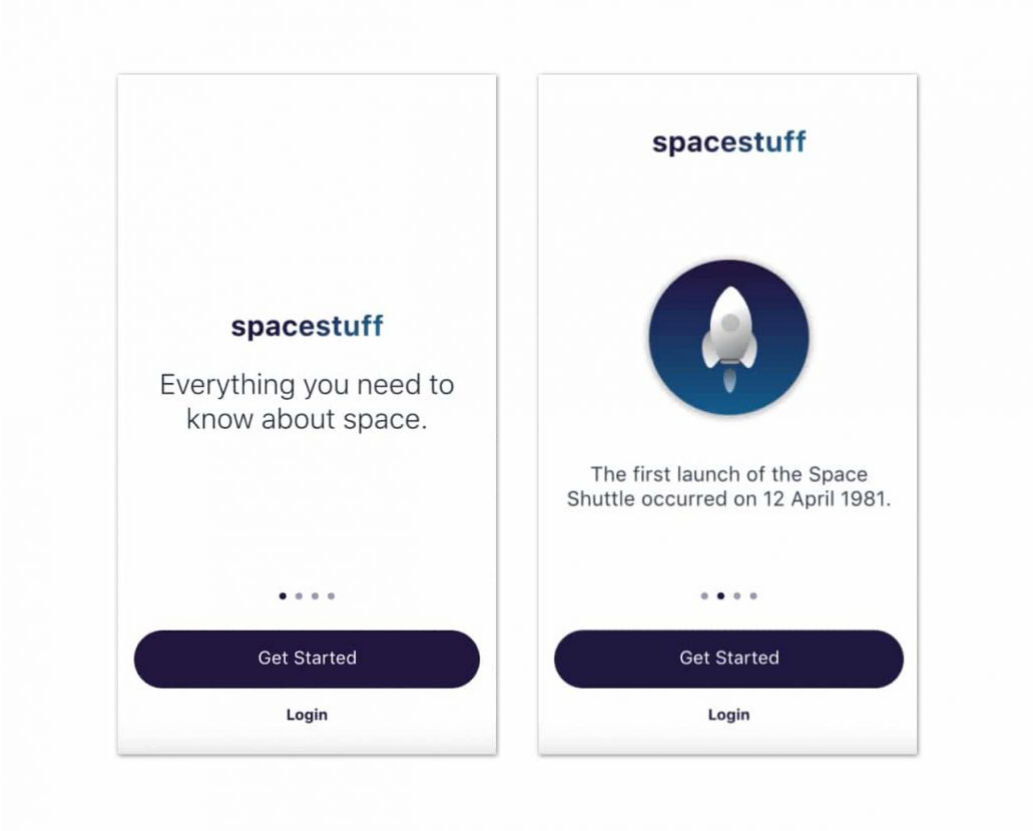

图 1.4: 线框图

高保真原型的目的在于让用户提供反馈，例如 UI 中控件的位置和 UI 的美观程度。

Chapter 2 软件过程

2.1 软件过程模型

2.1.1 瀑布模型（Waterfall）

软件工程是使用系统化的、严格约束的、可量化的方法开发、运行和维护软件。

瀑布模型是一个软件生命周期模型，开发过程是通过一系列阶段顺序开展的，从系统需求分析直到产品发布，项目开发从一个阶段流动到下一阶段，这也是瀑布模型名称的由来。直到 80 年代早期，它一直是唯一被广泛采用的软件开发模型。

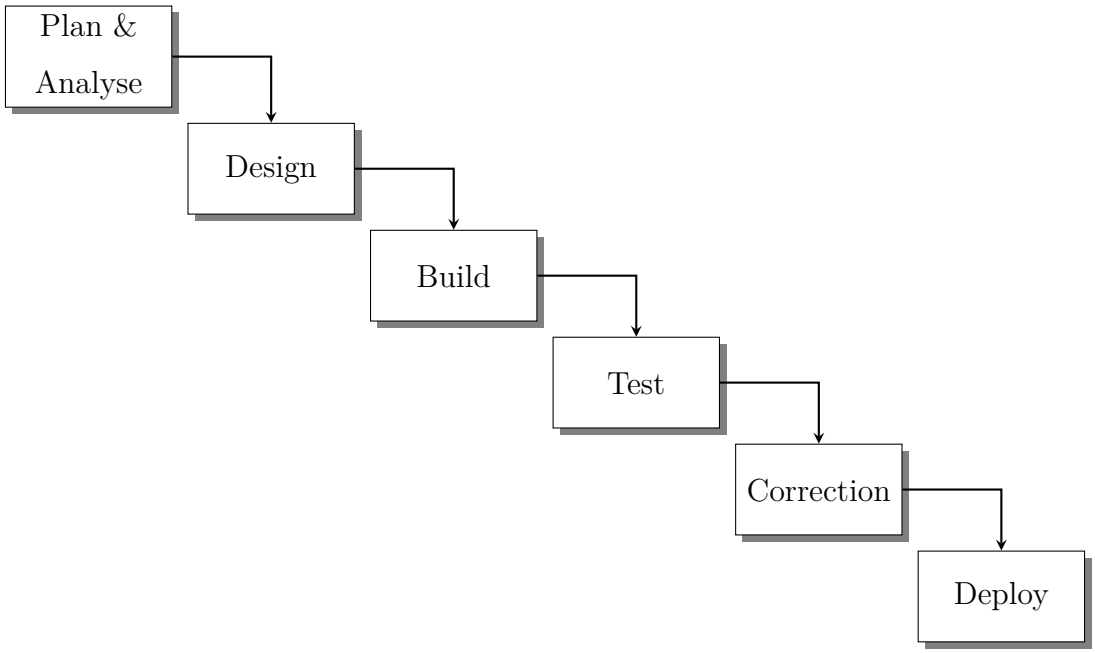

图 2.1: 瀑布模型

瀑布模型的特点是各阶段间具有顺序性和依赖性，必须等前一阶段的工作完成之后，才能开始后一阶段的工作。前一阶段的输出文档就是后一阶段的输入文档。

因此，瀑布模型是由文档驱动的，在可运行的软件产品交付给用户之前，用户只能通过文档来了解产品。瀑布模型几乎完全依赖于书面的规格说明，很可能导致

最终开发出的软件产品不能真正满足用户的需要。也不适合需求模糊的系统。

传统的瀑布模型过于理想化，人在工作过程中不可能不犯错误。因而产生了加入迭代过程的瀑布模型。

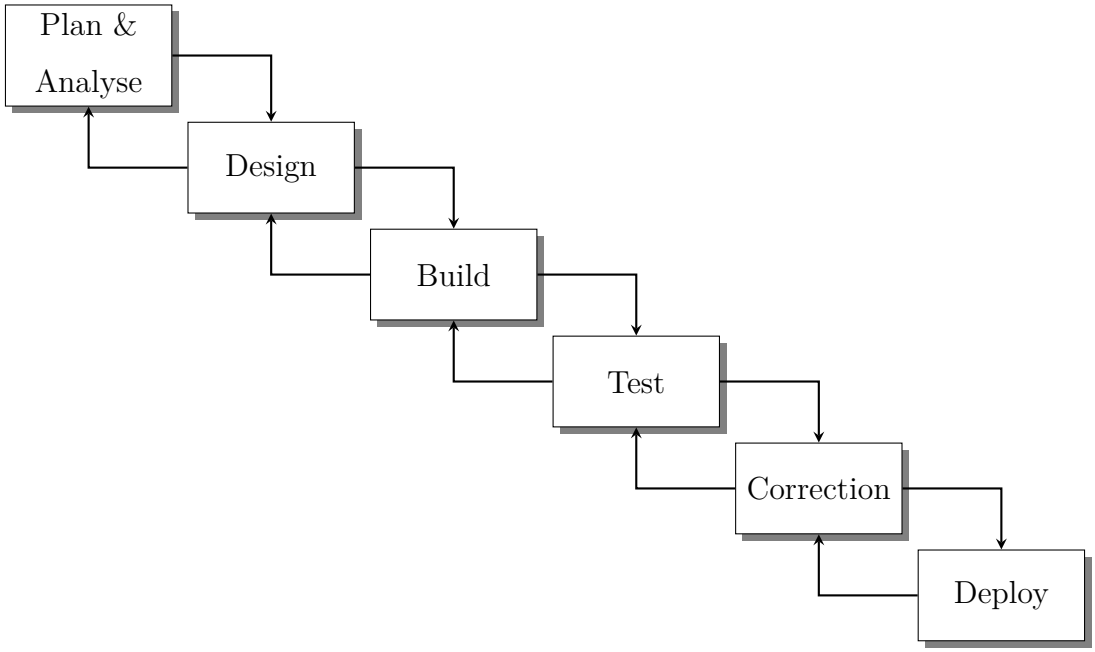

图 2.2: 加入迭代过程的瀑布模型

2.1.2 增量式开发（Incremental Development）

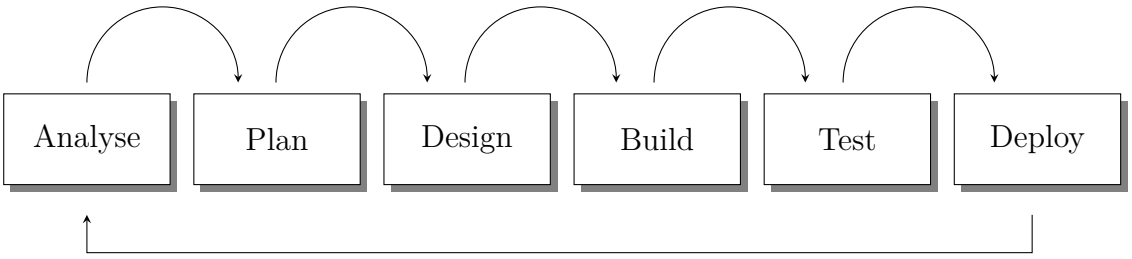

图 2.3: 增量式开发

增量式开发允许开发人员加以利用系统的早期交付版本，重新参考和修改以达到预期的结果。增量式开发较为灵活快速，在开发的每个周期都会产出一个能够交付的版本。这样较为频繁的交付，能够使客户持续地提供反馈，从而获得更高质量的产品。

2.2 敏捷开发

2.2.1 敏捷开发（Agile）

推行新的软件过程模型的原因在于，传统的开发方式有着更高的失败率，大部分项目的支出都远超出预算水平。

很多项目的开发阶段都超出了预期的时间，同时由于软件开发的特性，反而会导致"Adding people to a late project just makes it later"。

敏捷开发方法是一种将精力集中在软件本身，而不是设计和文档上。它依赖迭代的方法来完成软件的描述、开发和交付。

敏捷方法的核心思想体现在：

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

这些思想主要表现为注重客户的参与，客户会在开发过程中紧密参与，为软件系统提供新的需求和评估。同时敏捷开发要求能够接受变更，设计的系统需要能够适应新的需求变化。

2.2.2 敏捷项目管理

Scrum 方法是一个通用的敏捷方法，它更加注重迭代开发的管理。

在 Scrum 中，每个开发周期都包含开发任务的分配、实现、当前版本演示和会议讨论。每个周期都有一个固定的长度，通常为 2~4 周，每个周期开发者会根据开发任务的清单，依据需求的优先级各自实现需求。在该周期无法完成的任务，会

被继续放入任务清单，在下个周期继续完成。

在每个周期结束后，项目团队的所有成员需要参加会议讨论，根据当前版本的软件，回顾开发过程、交流问题等。

Scrum 使得产品被分解为多个可管理的部分，整个团队所做的任务都是可见的，有助于改善团队见的沟通。同时也能阶段性地交付产品给用户，在下个周期能够根据用户反馈进行修改。

Chapter 3 UML

3.1 UML

3.1.1 UML（Unified Modeling Language）

统一建模语言 UML 是一种为面向对象系统的产品进行说明、可视化和编制文档的一种标准语言。

UML 中包含了一系列不同类型的图：

- 类图
- 组件图
- 部署图
- 对象图
- 封装图
- 复合结构图
- 剖面图
- 用例图
- 活动图
- 状态机图
- 序列图
- 通讯图
- 交互概览图
- 时序图

这些图主要可分为三大类：

1. 功能模型（functional model）：从用户的角度展示系统的功能，如用例图。
2. 对象模型（object model）：采用对象、属性、操作、关联等展示系统的结构，如类图、对象图。
3. 动态模型（dynamic model）：展现系统的内部行为，如时序图、活动图、状态图。

3.2 用例图

3.2.1 用例图（Use Case Diagram）

用例图用于描述从用户角度所看到的系统功能。通过用例图，人们可以获知系统不同种类的用户和用例。

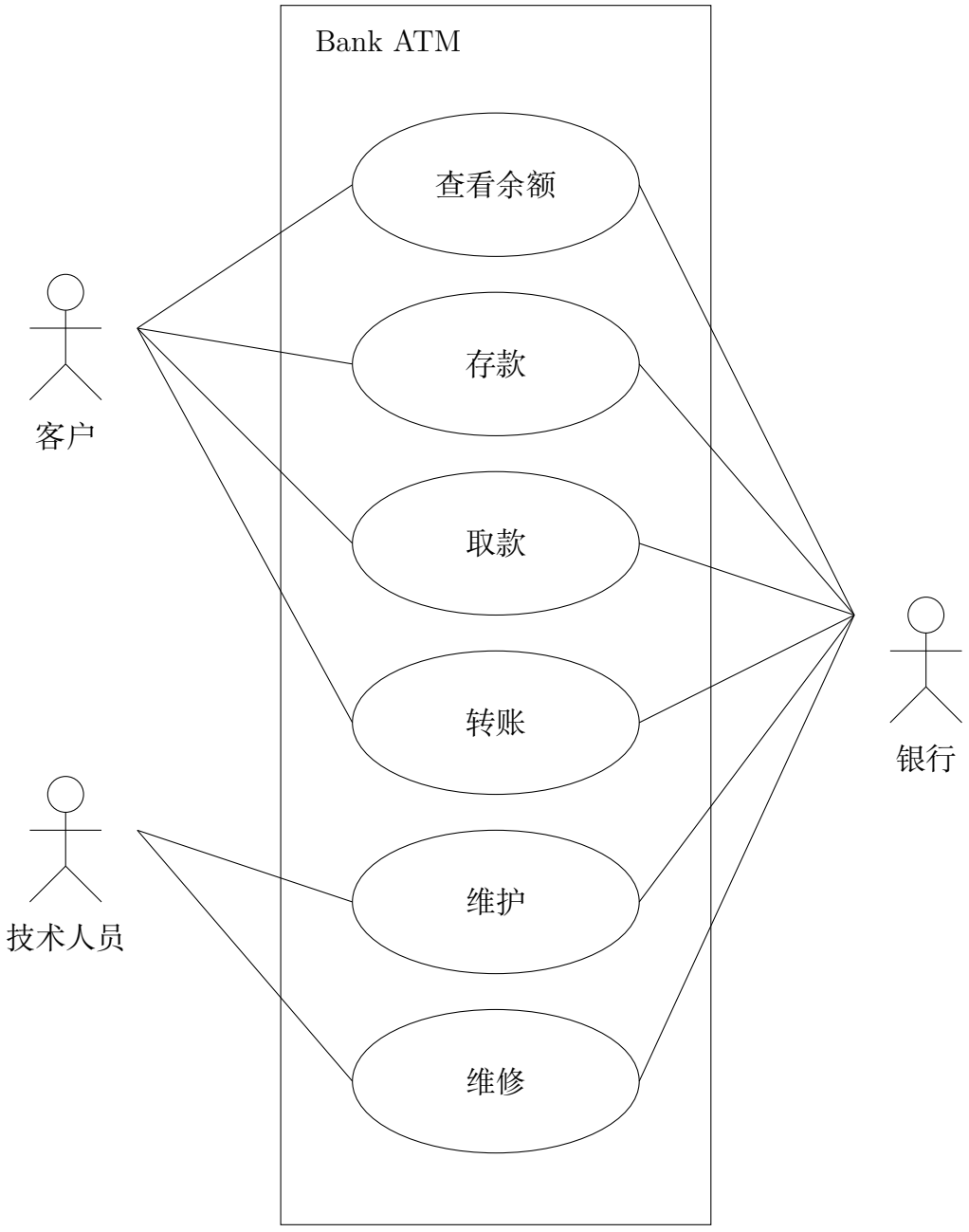

图 3.1: 银行 ATM 系统

3.3 类图

3.3.1 类图（Class Diagram）

类图用于显示模型的静态结构，例如类的内部结构以及类与类之间的关系。

一个类包含属性和方法两部分，在类图中 public 属性和方法使用“+”表示，private 属性和方法使用“-”表示，protected 属性和方法使用“#”表示。

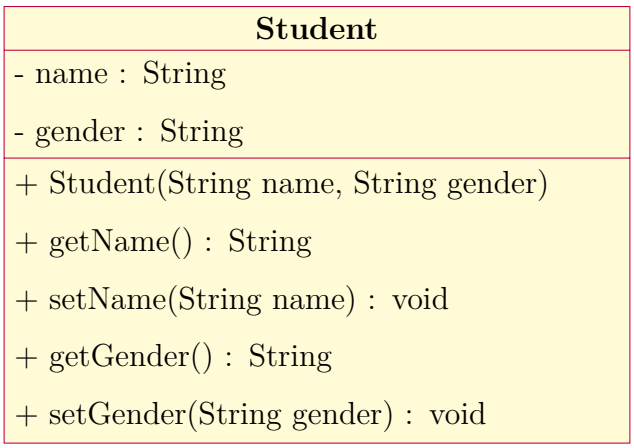

图 3.2: Student 类

3.3.2 继承（Inheritance）

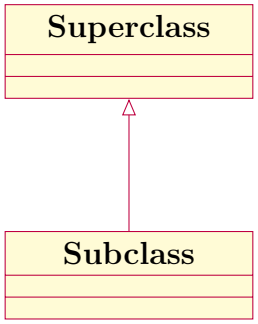

图 3.3: 继承

子类和父类之间存在“is a”的关系，例如 Dog 和 Cat 都属于 Animal 的子类。

3.3.3 关联（Association）

图 3.4: 关联

关联关系使一个类能够知道另一个类的属性和方法。关联可以是双向的，也可以是单向的。

关联关系还可以指定多重性（multiplicity）。例如一个 Student 可以和多个 Professor 存在关联。

一个 Professor 可以和多个 Student 存在关联。

多个 Professor 同样也可以和多个 Student 存在关联。

3.3.4 聚合（Aggregation）

聚合关系是关联联系的特例，用于表示整体和部分的关系，即 “has a”。但是整体和部分有各自独立的生命周期。

例如 Monther 有一个 Child，但是如果 Mother 死了，Child 仍然存活。

图 3.5: 聚合

3.3.5 组合（Composition）

组合用于表示“part of”的的关系，因此具有组合关系的两个类具有相同的声明周期。

例如血液细胞是身体的一部分，人死了，血液细胞也会死。

图 3.6: 组合

3.4 时序图

3.4.1 时序图（Sequence Diagram）

时序图通过描述对象之间发送消息的时间顺序显示多个对象之间的动态协作。

时序图中包括以下元素：

1. 角色（actor）/对象（object）：系统角色，可以是人或者其它子系统。
2. 生命线（lifeline）：表示对象在一段时期内的存在。
3. 控制焦点（activation）：在时序图中每条生命线上的窄矩形代表活动期。
4. 消息（message）：类角色通过发送和接受信息进行通信。

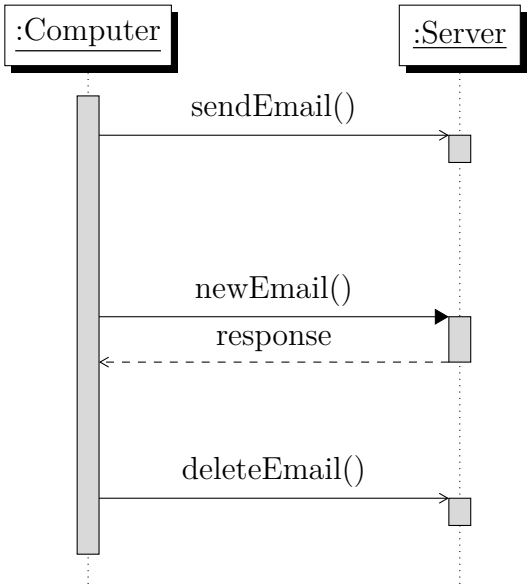

图 3.7: 时序图

Chapter 4

4.1

4.1.1