打车软件系统

体系结构设计文档

<3.0>

2019.12.6

平雅霓

学号：2017211949

班号：2017211504

北京邮电大学软件学院

2019 年秋季学期

1. 修订记录

日期	描述	作者	内容
<2019-12-6>	<版本 1.0>	<平雅霓>	<第一次修订>
<2019-12-11>	<版本 2.0>	<平雅霓>	<第二次修订>
<2019-12-13>	<版本 3.0>	<平雅霓>	<第三次修订>

2. 文档审批

以下的软件需求规格说明书已被接受并认可：

签名	打印版姓名	职位	日期
	<平雅霓>		

目录

1. 介绍

在这一部分，主要介绍了本体系结构文档编写的目的、范围、概念定义、以及用到的参考文献。

1.1. 目的

本文档为体系结构设计文档，旨在阐述现代打车软件系统的总体结构，并设计好软件系统的总体设计策略以及所需要用到的技术,提高软件开发过程中的能见度。同时该文档还能使开发过程中涉及到的不同的人员方便地沟通和写作。使管理者便捷地对软件开发过程控制与管理，使得本系统的开发能够更加高效。

本体系结构设计文档作文产品立项和产品开发的参考文档，给出各用户详细的功能要求，系统功能块组成及联系，进程部署和硬件要求等，有益于提高软件开发过程中的能见度，便于软件开发过程中的控制与管理。此体系结构设计文档是进行软件项目设计开发的基础，也是编写测试用例和进行系统测试的主要依据，它对开发的后续阶段性工作起着指导作用。同时此文档也可作为软件用户、软件客户、开发人员等各方进行软件项目沟通的基础。

用户能够从本文档了解到软件系统的功能和整体结构；开发人员能够从本文档中了解到软件系统预期的功能以及相应的架构，并依此对系统进行设计和开发；测试人员能够从文档中了解到软件系统需要测试哪些部分；维护人员能够根据本文档知道如何对软件系统进行维护；而项目管理人员则能从本文档中预估软件系统开发的大致时间流程以及制定相应的开发计划

1.2. 范围

软件系统名称：打车软件系统。

文档内容：

本体系结构设计文档详述了通用打车软件系统的项目相关信息、体系结构需求、解决方案、系统的质量分析和评价，并绘制了体系结构图、用例图、逻辑图、过程图、部署图。

本软件体系结构设计文档能够满足系统的质量和可靠性要求，并且分析了未来的升级维护中可能遇到的问题及相应的解决方案。

应用范围：本软件体系结构设计文档适合于滴滴打车、美团打车、高德打车等软件的总体应用结构，目的是满足系统的质量和可信赖性要求，以及打车软件系统未来的维护、运行和升级改造等要求。

1.3. 定义，首字母缩略字及缩写

术语	定义
打车软件系统	一个拥有乘客实时打车、预约车辆，司机接单抢单等主要功能的软件。
用户角色	用户角色是指按照一定参考体系划分的用户类型，是能够代表某种用户特征、便于统一描述的众多用户个体的集合。
功能性需求	功能性需求规定开发人员必须在产品中实现的软件功能，用户利用这些功能来完成任务，满足业务需求。功能需求更关注系统的输入、加工（业务流程）直到输出的情况。
非功能性需求	非功能性需求是指依一些条件判断系统运作情形或其特性，而不是针对系统特定行为的需求。 主要是系统的一些关键的特性，例如可靠性、响应时间、存储空间、I/O 设备的吞吐量、接口的数据格式等
功能结构图	功能结构图是将系统的功能进行分解，按功能从属关系表示的图表。
用例图	用例图是指由参与者（Actor）、用例（Use Case）以及它们之间的关系构成的用于描述系统功能的静态视图，被称为参与者的外部用户所能观察到的系统功能的模型图，主要用于对系统、子系统或类的功能行为进行建模。

表 1.定义与术语

术语	定义
B/S 结构	B/S 架构的全称为 Browser/Server，即浏览器/服务器结构。Browser 指的是 Web 浏览器，极少数事务逻辑在前端实现，但主要事务逻辑在服务器端实现，Browser 客户端，WebApp 服务器端和 DB 端构成所谓的三层架构。B/S 架构的系统无须特别安装，只有 Web 浏览器即可。
C/S 结构	C/S 架构是一种典型的两层架构，其全称是 Client/Server，即客户端服务器端架构，其客户端包含一个或多个在用户的电脑上运行的程序，而服务器端有两种，一种是数据库服务器端，客户端通过数据库连接访问服务器端的数据；另一种是 Socket 服务器端，服务器端的程序通过 Socket 与客户端的程序通信。
ATAM	Architecture Tradeoff Analysis Method(构架权衡分析方法），它是评价软件构架的一种综合全面的方法。这种方法不仅可以揭示出构架满足特定质量目标的情况，而且（因为它认识到了构架决策会影响多个质量属性）可以使我们更清楚地认识到质量目标之间的联系——即如何权衡诸多质量目标。
DFD	Data Flow Diagrams，数据流图从功能的角度对系统建模，追踪数据的处理有助于全面地理解系统，数据流图也可用于描述系统和外部系统之间的数据交换。
Broker	代理器

表 2. 缩写语

1.4. 参考

[1] 王安生，《软件工程化》[M].北京:清华大学出版社，2014

1.5. 概述

本文档将依次介绍：介绍、结构展示、结构目标和约束、用例图、逻辑图、过程图、部署图、大小与性能、质量等方面展开叙述。

2. 结构化展示

本文档将体系结构呈现为一系列视图：用例视图、逻辑视图、进程视图和部署视图。

用例视图是从外部用户的角度捕获系统，子系统或类的行为。 它将系统功能划分为对活动者具有意义的事务。这些功能片被称为用例。用例通过系统与一个或多个活动者之间的一系列消息描述了与活动者的交互。术语活动者包括人员、其他的计算机系统和进程。

逻辑视图：包含了类、接口、协作，静态方面用 类图和对象图表现，动态方面用活动图、状态图、交互图表现。

进程视图：侧重系统的运行特性　关注非功能性的需求　性能　可用性　。服务于系统集成人员　方便后续性能测试。强调并发性、分布性、集成性、鲁棒性　容错　、可扩充性、吞吐量等。和逻辑实体类似，可用类图（扩展）、活动图、交互图、状态图表现。

部署图是结构图的一种，它展示了系统的架构。部署图可以用于描述规范级别的架构，也可以描述实例级别的架构。

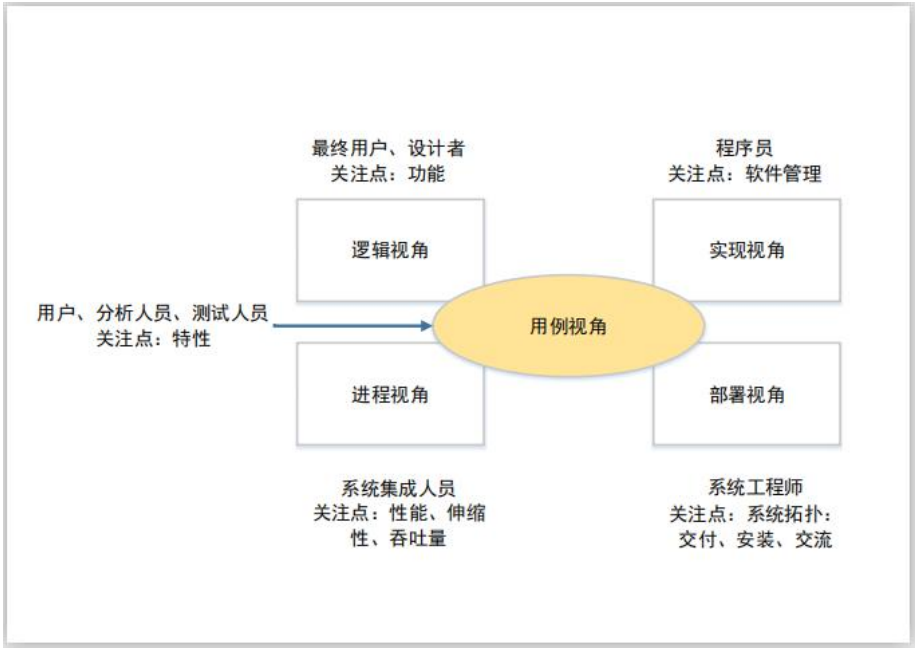

图 1 结构化展示图

3. 结构性目标和约束

3.1. 目标

打车软件系统结构主要分为三层：表示层、业务逻辑层、数据层。

三层 C/S 体系结构模式

表示层：客户端运行用户界面,向用户提供数据输入输出,响应用户动作并控制用户界面,把业务逻辑与用户界面分开,简化客户端的工作。

业务逻辑层：是系统实现业务逻辑与数据操作的核心部门，它的任务是接受用户的请求，首先执行扩展的应用程序并与数据库进行链接，通过 SQL 方式向数据库服务器提出数据处理申请，然后等到数据库服务器将数据处理的结果提交给 Web 服务器，再由 Web 服务器将结果传回给客户端。它提供所有的业务逻辑处理功能，整个系统中对数据库的操作都在这一层中完成。

数据层：数据库服务器运行数据引擎,负责数据存储。其任务是接受中间层对数据库操作的请求,实现对数据库的查询、修改、更新等功能,把运行结果返回给中间层。

下图结构化视图，也是最终要实现的结构性目标。

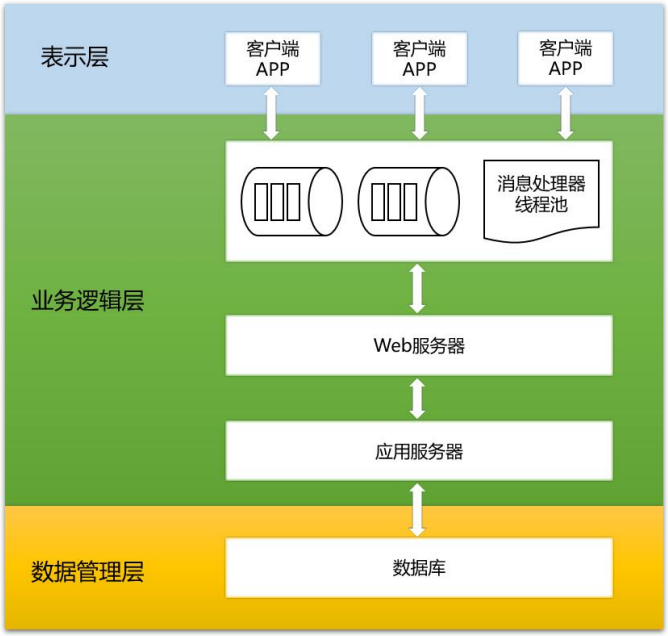

图 2 系统结构图

3.2. 体系结构用例

本说明书的用例是从使用者的角度出发，描述用户所期望的与整个运行系统的交互。根据《软件需求分析说明书》对本系统用户需求的分析以及对用例角色的陈述，本 体系结构说明书将用户的需求与系统进行交互。

角色	功能
软件开发人员	1. 清楚知道软件的整体结构。 2. 清楚知道各模块间的接口，方便编写。 3. 清楚知道自己所要完成的功能。
产品经理	1. 知道软件系统的规模。 2. 根据体系结构安排工作给各开发小组。 3. 决定开发策略。 4. 规划工期进度安排
测试人员	1. 明确系统的性能指标。 2. 明确系统的组织结构。 3. 编写测试用例的依据。
甲方人员	1. 对该体系结构文档给出评价。 2. 验证其是否能满足自己的需求。 3. 项目验收的标准。
老师	对该体系结构文档提出修改建议。

表 3.体系结构用例

3.3. 约束条件

（1） 甲方要求在 3 个月内完工。

（2） 可以满足随时随地查询使用的原则，因此要求采用 B/S 架构。

（3） 系统必须能支持 50000 个并发用户对系统发出服务请求，且其访问速度不得大于 3s；

（4） 法律和政策方面的限制：开发此软件时，将严格按照有关法律和政策执行。

4. 用例图

用户角色

HiTaxi 打车软件系统共有四种用户角色：打车者、司机、业务监测人员、系统管理人员。下面本文将针对四种用户画出它们的用例图。

4.1. 打车者

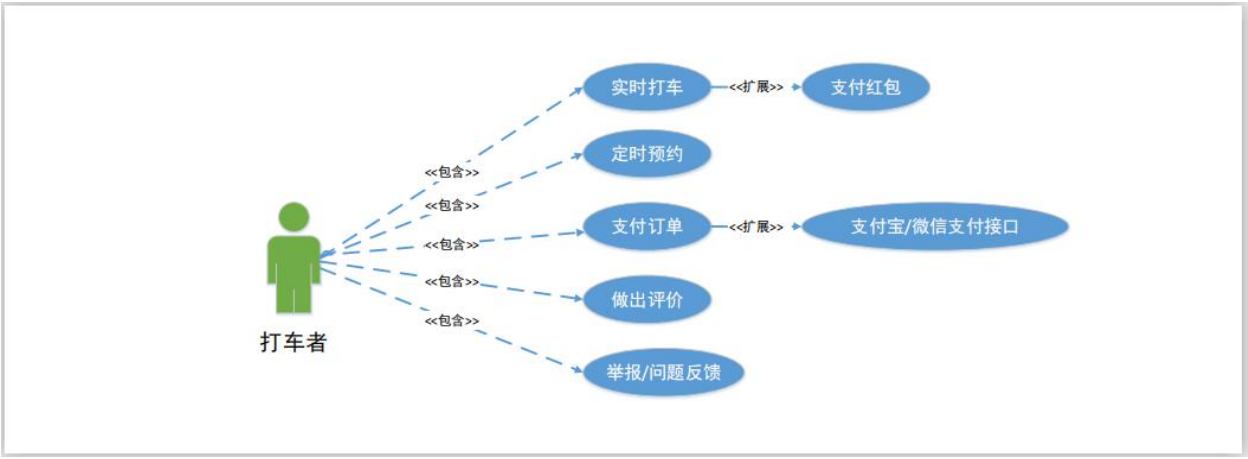

图 3. 打车者用例图

设计论述：

打车者的**主要功能**为：

（1） 实时打车：随时打车。打车者输入上车点、目的地，选择车型。

（2） 定时预约：预约车辆。打车者选择预约时间、上车地点、目的地，选择车型。

（3） 支付订单：使用第三方接口进行支付。从打车软件通过接口跳转到第三方支付软件（支付宝或微信等），打车者进行信息确认，完成支付，同时经过系统的计算后，将纯利润直接支付给司机，完成此订单。

（4） 做出评价：对本次订单的服务做出评价。订单完成后，乘客端弹出评价窗口，乘客可选择提供的可供参考的评价，也可以自行撰写评价，评价后点击提交。

（5） 举报或者反馈问题：举报司机不合法行为、反馈订单中遇到的问题。 打车者点击举报或者问题反馈按钮，撰写问题所在，然后点击提交。

其中实时打车有**扩展功能**：添加红包。

4.2. 司机

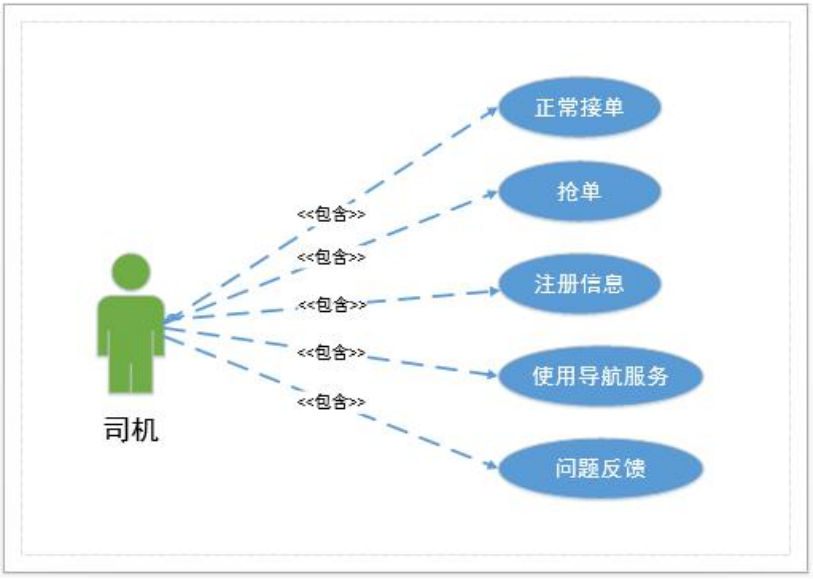

图 4. 司机用例图

设计论述：

司机的主要功能为：

（1） 正常接单：系统派单。司机服从调度。

（2） 抢单：与其他司机抢单。司机看见想接的单可以进行抢单。

（3） 注册信息：加入约车服务。

（4） 使用导航：根据导航行驶。

（5） 反馈问题：反馈订单中遇到的问题。如果订单中存在问题，司机可以及时反馈获得反馈结果。

4.3. 业务检测人员

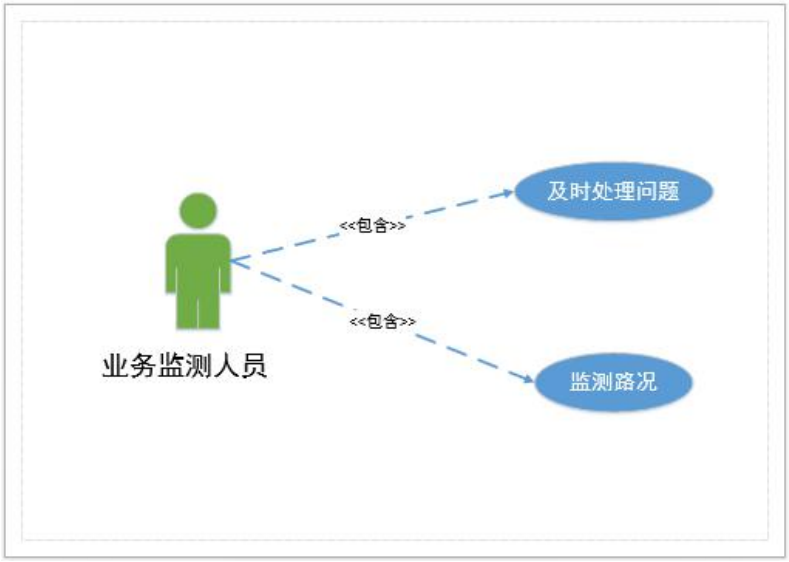

图 5. 业务监测人员用例图

设计论述：

业务监测人员的主要功能为：

（1） 及时处理问题：及时处理订单中出现的问题。乘客或者司机提交问题或者联系客服，客服针对现实情况进行相应的处理。

（2） 监测路况：及时向司机反馈路况以便走更方便的路线。

4.4. 系统管理员

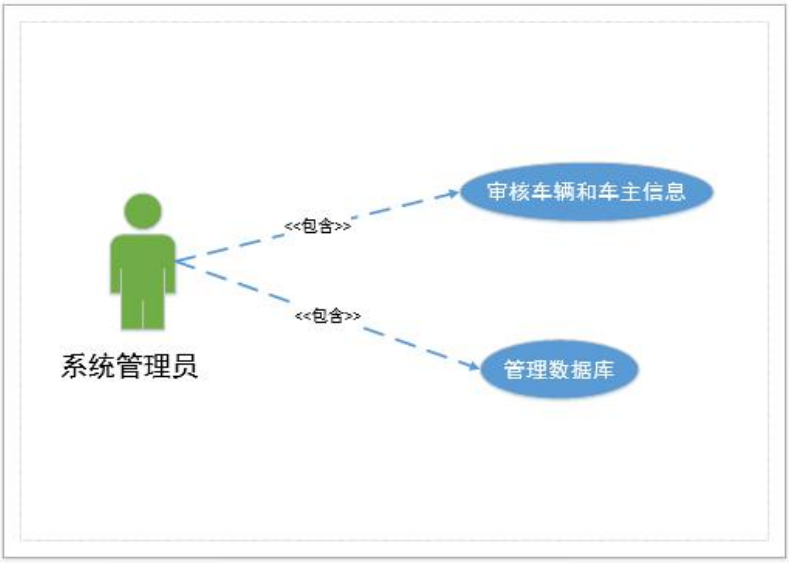

图 6. 系统管理人员用例图

设计论述：

系统管理人员的主要功能为：

（1） 审核车辆和车主信息：查看该车和车主是否符合可以参加约车服务的要求。车主填写自己和车辆的信息（要求实名制），然后点击提交，管理员工作时进行审核，若车主和车辆都符合要求，管理员通过其申请，否则不通过申请。

（2） 管理数据库：对数据库进行增删改查操作。

5. 逻辑图

逻辑图是对体系结构的逻辑视图的描述。描述最重要的类，它们在服务包和子系统中中的组织，以及这些子系统在层中的组织。还描述了最重要的用例实现，例如，架构的动态方面。类图可以用来说明架构上重要的类、子系统、包和层之间的关系。

打车软件系统的逻辑视图由三个主要包组成:用户界面、业务服务和业务对象。

用户界面 package 包含角色用于与系统通信的每个表单的类。边界类用于支持登录、查看实时附近车辆，查看打车预计费用，打车、预约车、支付、查看历史订单、评价、领优惠券、抢单等功能。

业务服务 package 包含一些控制类，用于与计费系统交互、派车、控制乘客和司机的注册、管理乘客评价。

业务对象 package 包括工件的实体类和与打车软件系统接口的边界类。

5.1. 结构综述——包、子系统层

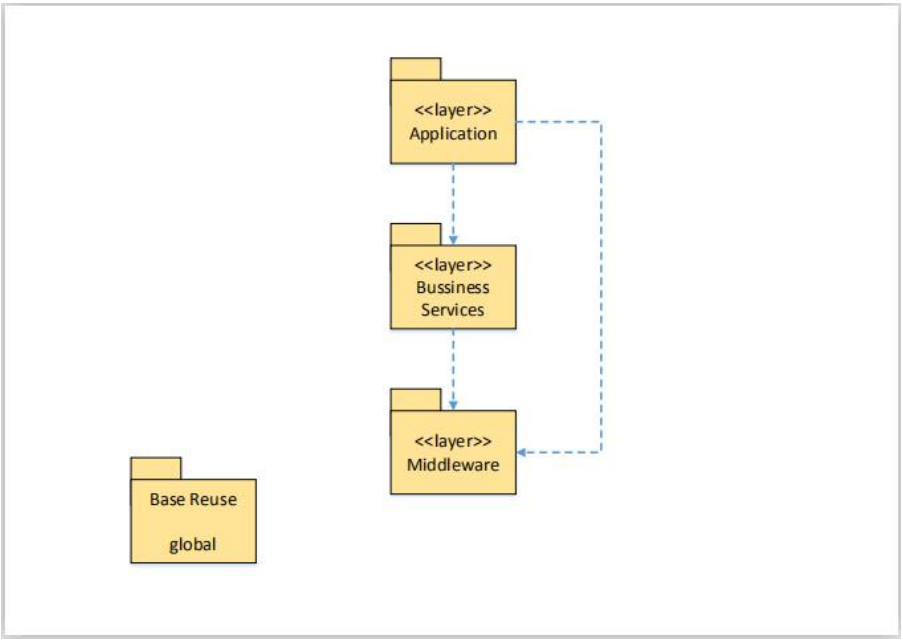

图 7. 包、子系统图

5.1.1. 应用

这个应用层拥有所有表示用户看到的应用程序屏幕的边界类。这一层依赖于流程对象层;这跨越了客户端与中间层的分离。

5.1.2. 业务服务

业务服务流程层具有表示驱动应用程序行为的用例管理器的所有控制器类。这一层表示客户端到中间层的边界。业务服务层依赖于流程对象层;这跨越了客户端与中间层的分离。

5.1.3. 中间件

中间件层支持对关系 DBMS 和 OODBMS 的访问。

5.1.4. 基础重用

基础重用包包括支持列表功能和模式的类。

6. 过程视图

体系结构的过程视图的描述。描述系统执行中涉及的任务(进程和线程)、它们的交互和配置。还描述将对象和类分配给任务。

流程模型说明了作为可执行流程组织的课程注册类。存在支持学生注册、教授功能、注册关闭和访问外部计费系统和课程目录系统的流程。

6.1. 过程

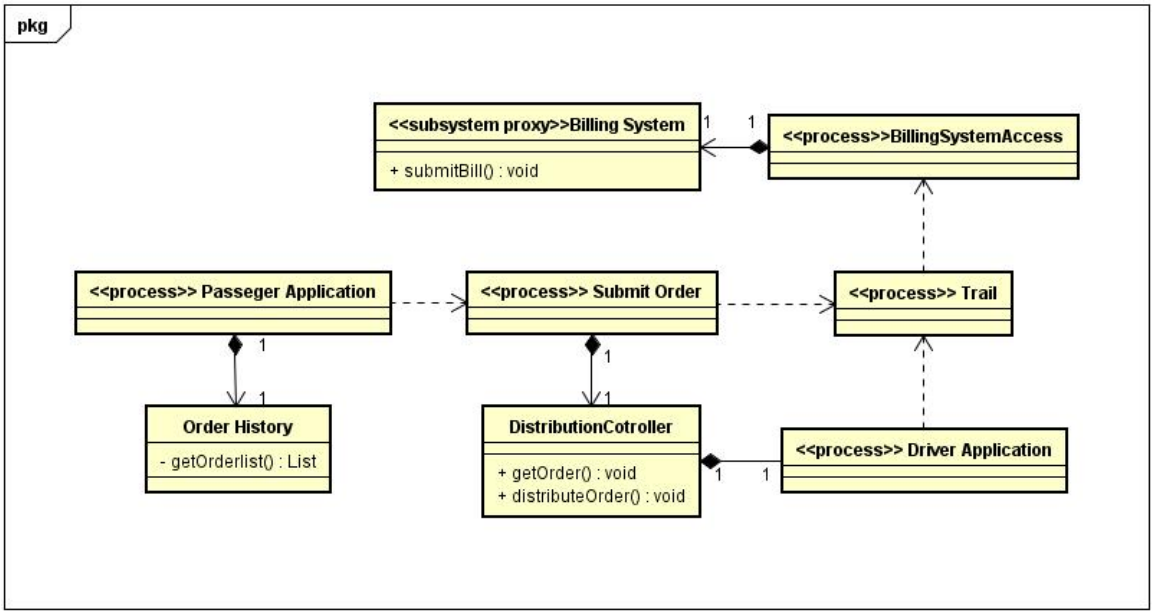

图 8. 过程图

6.1.1. BillingSytem

这是一个子系统，用来生成每次订单结束后的账单。

6.1.2. BillingSystemAccess

这个过程用来建立与子系统 BillingSystem 的访问

6.1.3. Order History

这是用来存放用户历史订单的表单。

6.1.4. Passenger Application

管理乘客功能，包括用户界面处理和与业务流程的协调。
对于当前正在使用软件的每个乘客，都有一个此过程的实例。

6.1.5. SubmitOrder

这个过程用来向服务器发送乘客的订单。

6.1.6. Trail

这个过程用来达成交易，即系统派车后连接乘客端和司机端。

6.1.7. DistributionController

这支持派车的用例。系统可以根据情况进行派车，司机也可以使用此进行抢单。

6.1.8. Driver Application

管理司机功能，包括用户界面处理和与业务流程的协调。
对于当前正在使用软件的每个司机，都有一个此过程的实例。

6.2. 元素设计过程

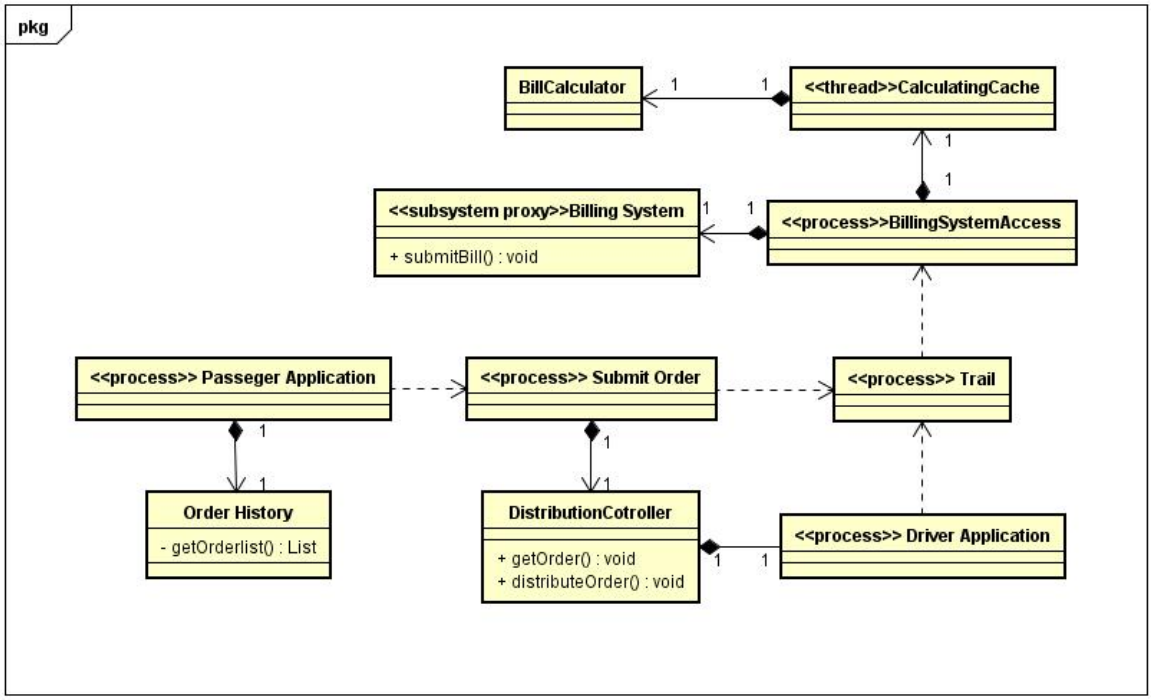

图 9.元素设计过程图

6.2.1. CalculatingCache

计算缓存线程用于从异步计算账单。

6.2.2. BillCalculator

账单计算器，用于计算账单金额。

6.3. 设计模型依赖性的过程模型

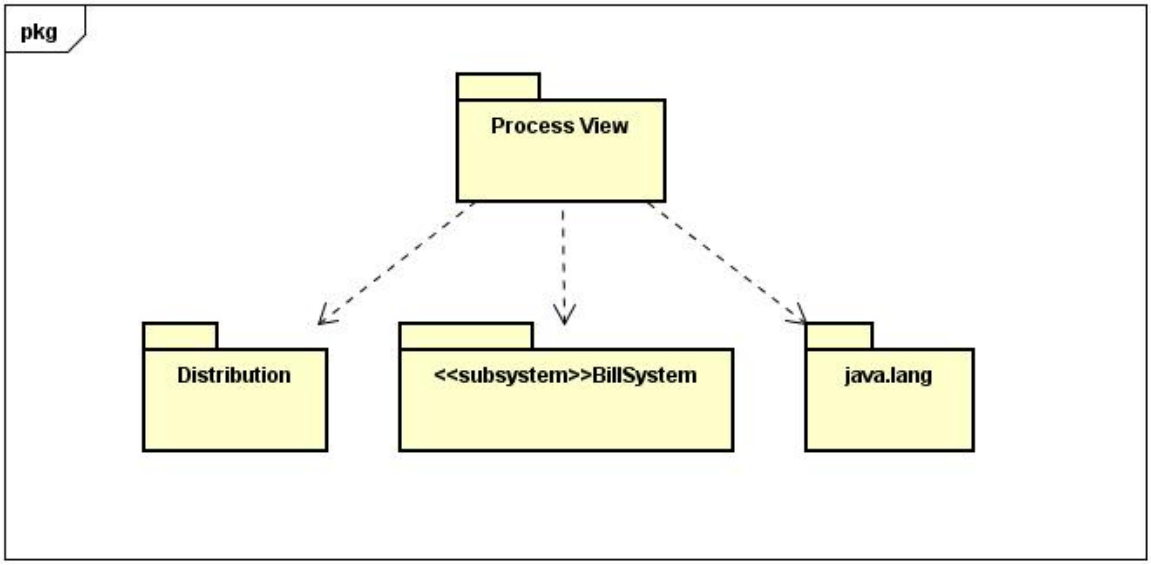

图 10. 过程模型图

6.4. 实现过程

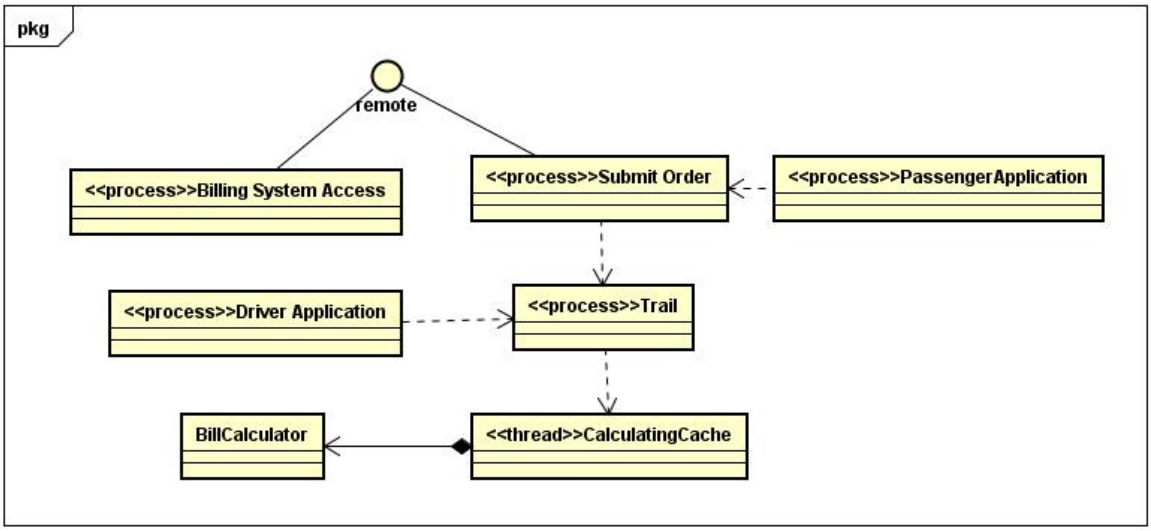

图 11.实现过程图

6.4.1. Remote

远程接口用于标识所有远程对象。任何远程对象都必须直接或间接实现此接口。只有在远程接口中指定的那些方法可以远程使用。

6.4.2. Runnable

Runnable 接口应该由其实例打算由线程执行的任何类来实现。该类必须定义一个名为 run 的无参数方法。

此接口旨在为希望在活动状态下执行代码的对象提供通用协议。例如，Runnable 是由 class Thread 实现的。

激活仅仅意味着一个线程已经启动，还没有被停止。

6.4.3. Thread

线程是程序中的执行线程。Java 虚拟机允许应用程序拥有多个并发运行的执行线程。

每个线程都有一个优先级。优先级高的线程优先于优先级低的线程执行。每个线程可能被标记为守护进程，也可能不被标记为守护进程。当在某个线程中运行的代码创建一个新线程对象时，新线程的优先级最初设置为创建线程的优先级，并且只有在创建线程是一个守护进程时，新线程才是守护进程。

7. 部署视图

体系结构的部署视图描述了最典型的平台配置的各种物理节点。还描述了任务(从流程视图中)到物理节点的分配。

本节按物理网络配置组织;每个这样的配置都由部署关系图来说明，然后是流程到每个处理器的映射。

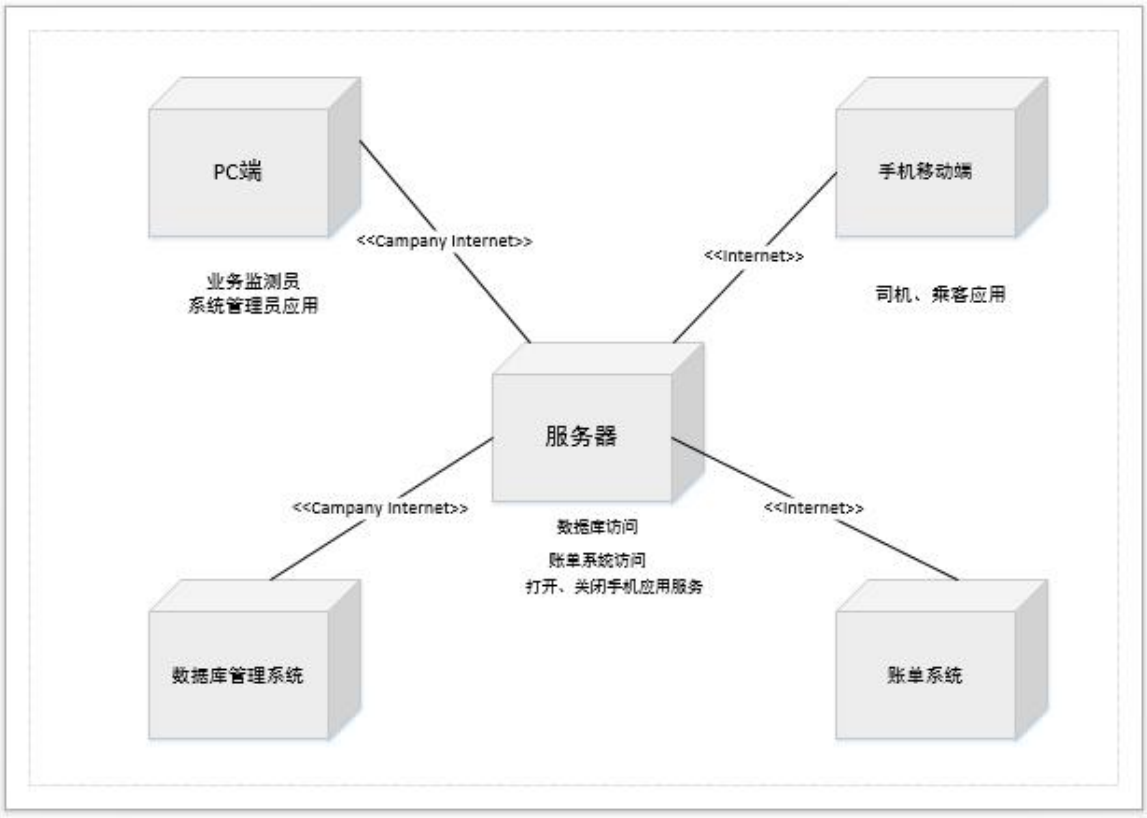

图 12.部署图

7.1. PC 端

业务监测员和系统管理员通过公司内网连接服务器登录系统进行工作。

7.2. 手机移动端

乘客和司机通过因特网连接服务器获取相应服务。

7.3. 服务器

所有用户可以用过内网或者因特网连接到服务器。服务器提供了 PC 端、移动端、数据库系统、账单系统的访问。

7.4. 数据库管理系统

数据库管理者可以通过公司内网将数据库系统连接到服务器，用户数据也要通过服务器的连接写入数据库。

7.5. 账单系统

账单系统通过和服务器连接，计算每次订单的金额。

8. 大小和性能

所选的软件架构支持关键的规模和时间需求，如补充规范所规定:

本系统在满足上诉要求情况下应当能够：

（1） 打车者订单发送响应时间：

在同时有 30000 打车者发送订单的情况下，系统的响应时间不得大于 2s；

在同时有 10000 打车者发送订单的情况下，系统的响应时间不得大于 1s；

在同时有 5000 打车者发送订单的情况下，系统的响应时间不得大于 0.2s；

（2） 司机抢单响应时间：

在同时有 100 司机抢单的情况下，系统的响应时间不得大于 1s；

在同时有 50 司机抢单的情况下，系统的响应时间不得大于 0.5s；

在同时有 10 司机抢单的情况下，系统的响应时间不得大于 0.2s；

（3） 其它所有交互功能反应速度：

在系统的用户数目为 10000 的情况下，响应时间不超过 2s；

（4） 对处理数据量的要求：

对数据的处理量在 G 的级别时系统的响应应当能够在 5s 之内；

本系统在可靠性上应当能够：

（1） 出错率

在累计 10 万次服务请求的情况下，其出错次数不得高于 3 次。

在累计 100 万次服务请求的情况下，其出错次数不得高于 60 次。

（2） 稳定性

在 10 万用户的并发下，连续 5 个小时的工作情况下，系统不会出现停止服务的情况。

（3） 易恢复性

在停机后进行恢复之后，恢复损失的数据不得大于 1 小时

本系统在可维护性上应当能够：

要求系统 1 年的停机维护时间不得超过 12 个小时。

8.1. 性能与效率

效率是指在规定条件下软件产品所占用的资原数量以及所提供的功能和性能的能力。包括时间特性、资源利用性。

(1) 时间特性：系统执行速度需要满足以下条件：系统响应快。

用户发送打车请求的响应时间不超过 0.5 秒；

司机能够接到订单消息的时间不超过 0.5 秒。

(2) 系统响应时间考虑到同时进行操作的用户人数,打车软件系统应当能最低支持 10000 位用户，期望正常支持 20000 位用户，在支持 30000 名用户时系统仍能正常使用，支持高并发操作。

8.2. 可靠性

可靠性是在指定条件下使用时，软件产品维持规定的性能级别的能力。

软件可靠性可以进一步分解为 3 个子特性：成熟性、容错性、易恢复性。

软件需要能较长时间下稳定运行，同时该软件需要具备一定的故障恢复能力，即有一定的容错能力。当用户的操作不当引起某些故障或者是由于操作系统或者网络发生故障时，系统需要具备一定的故障恢复能力。 选择数据库产品时，要考虑一定的数据负载能力。本软件要求对乘客的订单详情和司机的订单详情保持一致。同时，可以对数据使用异地容灾备份的方式，从而进一步提高打车软件系统中数据抵抗外界破坏的能力。

8.3. 可用性

易用性是从用户使用观点来看的，指在指定条件下使用时，软件产品被理解、学习、使用和吸引用户的能力。易用性可以分解为 4 个子特性：易理解性、易学性、易操作性、吸引性。

打车软件面向的用户为普通群众，且操作习惯、受教育程度、年龄阶段、接受事物能力等都各不相同，这就要求系统具备良好的人机交互能力。因此移动端和 PC

端界面应设计得人性化且简洁，符合 GUI 的基本标准，符合一些程序的设计习惯，操作应当简易，符合普通的手机用户或电脑用户使用手机和电脑的习惯。

还应该在软件中设置操作引导，指导初用者快速掌握使用此软件。

8.4. 安全性

打车软件涉及到多种用户角色的重要记录，因此数据库中的信息必须要得到良好的保护。为了保护数据，系统配置文件和数据库存储文件应当进行加密处理，并使用例如防火墙这样的安全的连接保证传输过程的数据信息安全性。另外，司机与乘客的详细信息应该是互相不可见的，因此还需要做好可见性设置。

用户在操作时也要使用可靠的操作系统来保证系统的操作安全，确保系统在一个安全可靠的环境下运行。

另外，系统还应当是能被重建的，应当能被有效控制，抗干扰能力强，系统应保证不会因恶意攻击而崩溃，系统开发过程不存在明显漏洞。

8.5. 可维护性

维护性是指软件产品可被修改和维护的能力。包括易分析性、易修改性、稳定性、易测试性。该软件的结构、接口、功能以及内部过程在开发以及跟踪阶段，容易被维护人员理解。同时，该系统有良好的测试和诊断系统错误的功能，还需要在应用于不同环境时，不需要通过大幅度的接口与内部过程修改，就能使用户进行使用。

8.6. 可移植性

软件产品从一种环境迁移到另外一种环境的能力。包括易安装性、共存性、易普换性。

易安装性：该软件能够跨平台移动运行，PC 端包括 Windows 服务器平台以及 Linux 平台，移动端包括 ios 系统和安卓操作系统。

共存性：软件应当能够和其他软件共存于一个平台上且不存在冲突。

易替换性：系统可被容易地卸载，也易被高版本的系统替换，做到及时更新。

9. 质量

软件架构支持质量要求，如补充规范所规定:

1. PC 端桌面用户界面应与 Windows 95/98 兼容。
2. 移动端应用应有 ios 版本和安卓版本。
3. 移动端的用户界面应设计得易于使用，并应适合于会使用手机的所有用户。
4. 软件系统的每个功能都应该为用户提供内置的在线帮助。在线帮助应包括使用系统的步骤说明。在线帮助应包括术语和缩写词的定义。
5. 打车的 PC 客户端部分的升级应通过 internet 下载。移动端可在软件商城进行下载。

ATAM：Architecture Tradeoff Analysis Method(构架权衡分析方法），它是评价软件构架的一种综合全面的方法。这种方法不仅可以揭示出构架满足特定质量目标的情况，而且（因为它认识到了构架决策会影响多个质量属性）可以使我们更清楚地认识到质量目标之间的联系——即如何权衡诸多质量目标。

9.1. 场景分析

通过场景可以很好地评估体系结构。SEI 开发了通过手工评估和确认体系结构的方法，称为体系结构折中分析方法(ATAM --Architecture Tradeoff Analysis Method)。它是评价软件构架的一种综合全面的方法。这种方法不仅可以揭示出构架满足特定质量目标的情况，而且可以使我们更清楚地认识到质量目标之间的联系。本文档根据 ATAM 方法来对城市共享停车管理系统的体系架构进行分析和评估。

9.1.1. 用例场景

用例场景是从使用者的角度出发，描述用户期望的与整个运行系统的交互。

场景-1	• 乘客希望，登录进入主页面后，系统能自动定位手机位置信

	息，并显示附近车辆的位置以及多久之后可以打到车。 • 司机用户希望，登录进入主页面后，系统能够自动定位车辆位置，并自动获取当前订单信息。 • 司机用户希望，接单后接入导航系统速度要快，而且导航系统必须是成熟的软件厂商提供的应用接口，导航的精准度、可靠性和实时性要高。 • 该场景代表了用户期望系统易于使用，即易用性。
场景-2	• 当系统发生故障时，缓存系统要能在 1 秒钟内从一个处理器切换到另一个处理器。 • 系统测试时间至少为 100 小时，测试参与用户至少为 1000 人。本系统应该避免因为故障而导致的失效的能力，在进行测试的过程中，本系统应该尽可能遍历所有的可能故障情况，保证系统的成熟。 • 系统具有一定的容错能力和故障恢复能力，在发生硬件或者软件异常时，应该具有依然具有服务能力。服务能力虽然下降，但不会导致系统崩溃无法使用。 • 本系统在发生停机故障之后，应该保证十分钟之内可以修复并且正常运行。 • 该场景代表了系统可靠性要求。
场景-3	• 系统中的数据得到更新后，系统的其他使用者页面的数据要在 2 秒内得到更新。 • 50,000 个用户能够并发登陆系统，且此时系统产生的延迟不超过 10 秒。 • 对用户的查询操作，系统的响应时间不超过 5 秒。 • 该场景代表了系统的性能要求。

表 5- 1 用例场景

9.1.2. 增长性场景

增长性场景是指预期未来系统修改时可能发生的场景。

场景-1	• 可以增加新的数据服务器，降低远程用户的访问时间。
场景-2	• 通过向数据库提供商购买更多的 license，来提升系统数据库的并发度。

表 5- 2 增长性场景

9.1.3. 探索性场景

探索性场景是推动系统封装和降低工作压力的场景。场景的目标是解释当前设计的边界条件的限制，揭露出可能隐含着的假设条件。

场景-1	• 可能增加共享单车的服务。
场景-2	• 可能增加与出租车公司的服务，避免市场的恶性竞争。
场景-3	• 改进系统的可使用性，从 98%提升到 99.999%。
场景-4	• 正常情况下，当一半服务器宕机时，不影响整个系统的可使用性。

表 5- 3 探索性场景

9.2. 原型分析

9.2.1. 效用树

效用树（utility trees）提供自顶向下的机制，直接和有效地将系统的业务具体分解到质量的属性场景，效用树的输出是场景的优先权列表，可供评估人员在短时间内发现体系结构中存在的问题。

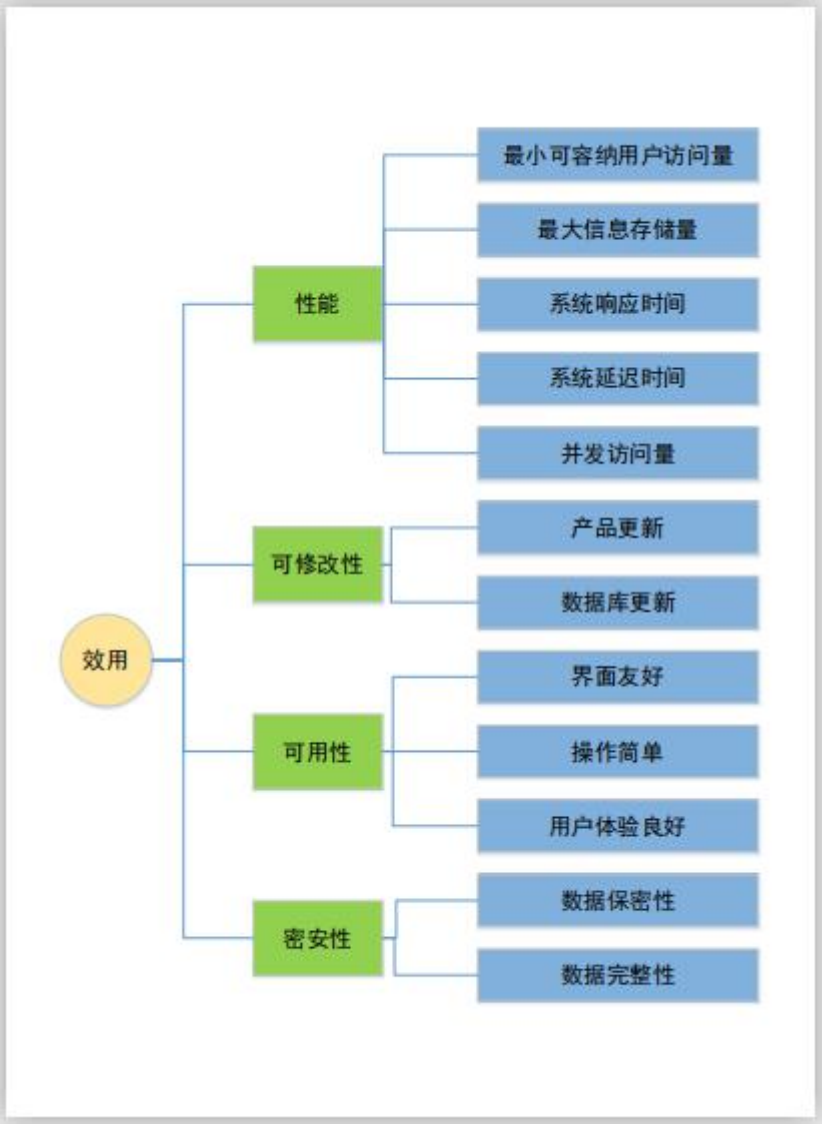

图 5- 1　城市共享停车管理系统质量效用树

9.3. 风险

在开发新的软件系统的过程中，由于存在许多不确定因素，软件开发失败的风险是客观存在的。在系统运行过程中，由于外部环境变化或者自身设计的不足，也可能产生各种风险。因此，风险分析对于软件项目管理是决定性的。

（1） 对需求变化的适应性：在软件的开发过程中，用户很可能会中途更换需求，或者要求增加新的需求。

（2） 软件易用性：软件的易用性是影响软件是否被用户接受的关键因素。在软件产品中，设计复杂，功能强大而完备，但因为操作繁复而被搁置者屡见不鲜。造成的主要原因在于缺乏软件开发中软件体系结构的宏观把握能力。

（3） 软件的可伸缩性：是指软件在不进行修改的情况下适应不同的工作环境的能力。由于硬件的飞速发展和软件开发周期较长的矛盾，软件升级的需要显得非常迫切。如果软件的升级和移植非常困难，软件的生命期必定很短，使得化费巨大人力物力开发出的软件系统只能在低性能的硬件或网络上运行，甚至被废弃不用，造成巨大的浪费。

（4） 软件的可维护性：为了保证软件的较长使用寿命，软件就必须适应不断的业务需求变化，修改的成本和周期都直接和软件的体系结构相关。一个好的软件体系结构可以尽可能地将系统的变化放在系统的配置上，即软件代码无需修改，仅仅是在系统提供的配置文件中进行适当的修改，然后软件重新加载进入运行状态，就完成了系统部分功能和性能要求的变化。对于重大改动，需要打开源代码进行修改的，也仅仅是先继承原先的代码，然后用新的功能接替原先的调用接口，这样将把软件改动量减小到最低。

9.3.1. 技术风险

在软件项目开发和建设的过程中，战略管理技术因素是一个非常重要的因素。项目组一定要本着项目的实际要求，选用合适、成熟的技术，千万不要无视项目的实际情况而选用一些虽然先进但并非项目所必须且自己又不熟悉的技术。如果项目所要求的技术项目成员不具备或掌握不够，则需要重点关注该风险因素。重大的技术风险包括：软件结构体系存在问题，使完成的软件产品未能实现项目预定目标;项目实施过程中才用全新技术,由于技术本身存在缺陷或对技术的在掌握不够深入，造成开发出的产品性能以及质量低劣。

预防这种风险的办法一般是经常和用户交流工作成果、品牌管理采用符合要求的开发流程、认真组织对产出物的检查和评审、计划和组织严格的独立测试等。软件质量的保证体系是软件开发成为可控制过程的基础，也是开发商和用户进行交流的基础和依据。所以制定卓有成效的软件质量监督体系，是任何软件开发组织必不可少的。

9.3.2. 进度风险

工期的综合要求：用户通常会对工期提出要求，比如第一个版本在 3 个月内完工，第二个版本在 6 个月内完工，第三个版本要在 9 个月内完工等等。体系结构的设计需要很好的支持上述要求。

软件的工期常常是制约软件项目的主要因素。软件项目工期估算是软件项目初期最困难的工作之一。很多情况下，软件用户对软件的需求是出于实际情况的压力，希望项目承担方尽快开发出软件来。在软件招标时，开发方为了尽可能争取到项目，对项目的进度承诺出已远远超出实际能做到的项目进度，使项目在开始时就存在严重的时间问题。软件开发组织在工期的压力下，往往放弃文档的编写与更新，结果在软件项目的晚期大量需要通过文档进行协调时，却拖累软件进度越来越慢。此外，由于用户配合问题、资源调配等问题也可能使软件项目不能在预定的时间内完成任务。软件项目过程中有自身的客观规律性，用户对软件项目的进度要求不能与软件开发过程的时间需要相矛盾。

对于这种风险解决方案一般是分阶段交付产品、增加项目监控的频度和力度、多运用可行的办法保证工作质量避免返工。在项目实施的时间进度管理上，需要充分考虑各种潜在因素，适当留有余地;任务分解要详细，便于考核;在执行过程中，应该强调项目按照进度执行的重要项，再考虑任何问题时，都要经保持进度作为先决条件;同时，合理利用赶工期及快速跟进等方法，充分利用资源。应该避免：某方面的人员没有到位，或者在多个项目的情况下某方面的人员中途被抽到其他项目，或身兼多个项目，或在别的项目中无法抽身投入本项目。为系统测试安排足够的时间，能使项目进度在改变之初就被发现，这对及时调整项目进度至关重要。在计划制定时就要确定项目总进度目标与分进度目标;在项目进展的全过程中，进行计划进度与实际进度的比较，及时发现偏离，及时采取措施纠正或者预防，协调项目参与人员之间的进度关系。

9.3.3. 质量风险

任何软件项目实施过程中缺乏质量标准，或者忽略软件质量监督环节都将对软件的开发构成巨大的风险。有些项目，用户对软件质量有很高的要求，如果项目组

成员同类型项目的开发经验不足，则需要密切关注项目的质量风险。矫正质量低下的不可接受的产品,需要比预期更多的测试、设计和实现工作。

预防这种风险的办法一般是经常和用户交流工作成果、品牌管理采用符合要求的开发流程、认真组织对产出物的检查和评审、计划和组织严格的独立测试等。软件质量的保证体系是软件开发成为可控制过程的基础，也是开发商和用户进行交流的基础和依据。所以制定卓有成效的软件质量监督体系，是任何软件开发组织必不可少的。