编号_____

南京大学

课 程 作 业

题　目　**Software Architecture Assignment 1**

学生姓名	王一辉
学　　号	191250142
学　　院	软件学院
专　　业	软件工程
班　　级	二班

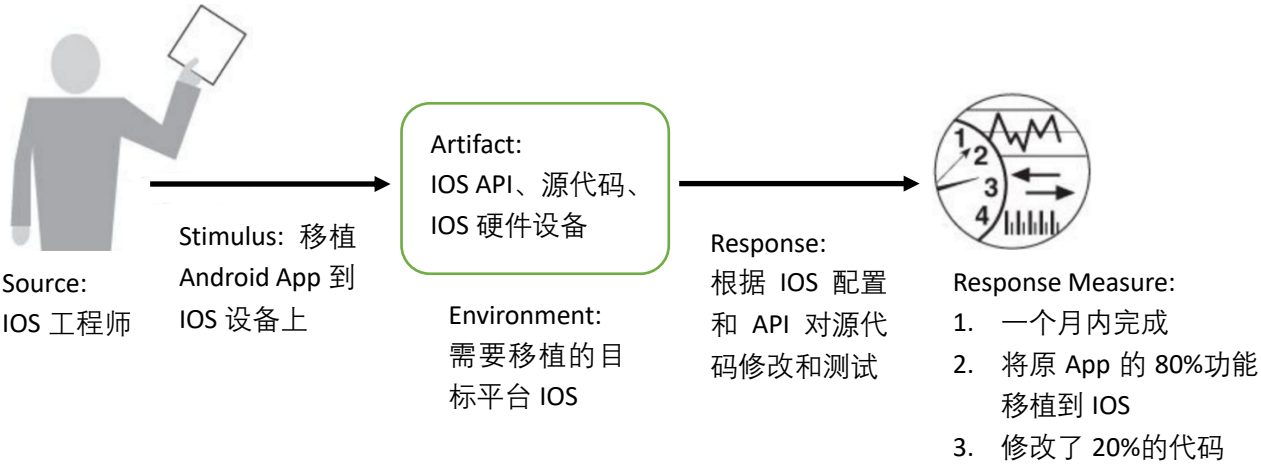
2022 年 5 月

I. Portability (可移植性) vs. Flexibility (灵活性)

❖ Portability General Scenario 可移植性通用方案

Portion of Scenario	Possible Values
Source	用户、开发者、管理员
Stimulus	将开发中或已发布的软件系统从当前环境和平台移植到其它环境和平台
Artifact	代码、数据、接口、硬件、配置……
Environment	系统需要在多平台上运行；不同的平台会有不同的 API 与规范，需要系统在编写、构建、运行时适应；
Response	以下之一或多个: ➢ 移植到其他平台； ➢ 测试多平台运行一致性； ➢ 多平台同步或异步构建、部署；
Response Measure	以下之一或多个: ❖ 百分之多少的代码需要为了适应不同平台修改； ❖ 系统需要移植到多少平台； ❖ 百分之多少功能受影响，需要多久进行移植；

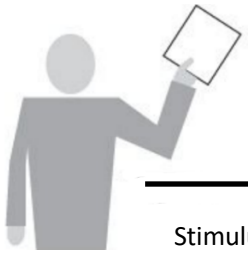
❖ Portability Concrete Scenario 可移植性具体方案



❖ Flexibility General Scenario 灵活性通用方案

Portion of Scenario	Possible Values
Source	用户、开发者、管理员
Stimulus	源代码相关依赖已更新；源代码需要适应新环境；原本实现的功能需要修改/延拓/添加内容；
Artifact	代码、接口
Environment	系统对相关依赖已更新，且原本依赖已过时；系统需要运行在新的环境中；系统需要适应新的需求；
Response	以下之一或多个： ➢ 修改/添加代码； ➢ 测试被修改的代码； ➢ 部署新的代码和包；
Response Measure	以下之一或多个： ❖ 修改需要耗费的时间以及人员成本； ❖ 修改的代码会影响到原代码的百分之多少； ❖ 修改后的代码有多少与原代码冲突，会产生怎样严重的 bug（无/轻微/严重/致命）；

❖ Flexibility Concrete Scenario 灵活性具体方案



Source:
开发者

Stimulus：为原本的 Web 项目添加系统能

Artifact:
源代码、接口

Environment:
开发框架更新，不支持就版本。有新需求。

Response:
在源代码基础上尽可能添加代码；测试代码；

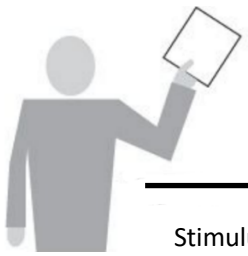
Response Measure:
1. 三天内完成修改
2. 10%的新代码与源代码冲突较少
3. 修改后无严重 bug

II. Reusability (可重用性) vs. Maintainability (可维护性)

❖ Reusability General Scenario 可重用性通用方案

Portion of Scenario	Possible Values
Source	开发者
Stimulus	源代码有小的更新或者没有更新;
Artifact	代码、应用和软件成品
Environment	应用与软件原本的运行环境; 或已经更新的环境;
Response	以下之一或多个: ➢ 重用原本的代码; ➢ 在更新的环境下测试原本代码; ➢ 部署和使用原本的代码和程序;
Response Measure	以下之一或多个: ❖ 原本的代码有多少是可以重复使用的; ❖ 产生小的修改后百分之多少功能可以完好运行; ❖ 重用代码可以减少多少时间、成本、人力;

❖ Reusability Concrete Scenario 可重用性具体方案



Source:
开发者

Stimulus: 新开发的应用要使用过去的框架

Artifact:
源代码、过去的框架

Environment:
运行环境有轻度的更新

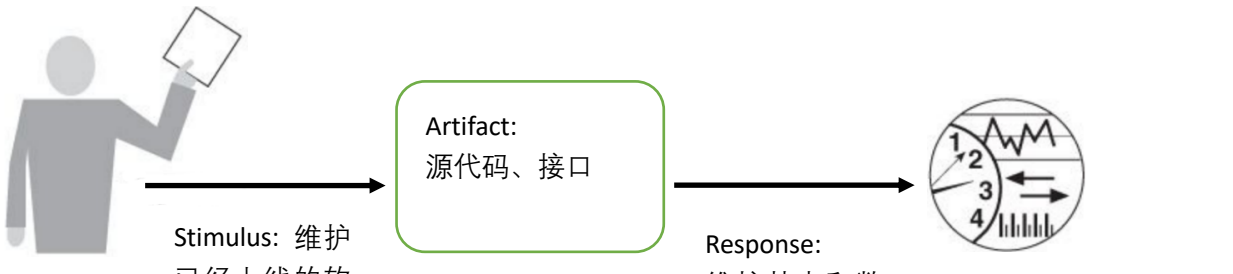
Response:
直接重用过去的框架，仍满足现状的需求

Response Measure:
1. 功能运行良好，重用 70%
2. 2 天内结束新开发流程
3. 修改后的 90%代码可以完好运行原功能

❖ Maintainability General Scenario 可维护性通用方案

Portion of Scenario	Possible Values
Source	开发者
Stimulus	源代码需要修改；
Artifact	代码、接口、应用
Environment	应用与软件原本的运行环境；或已经更新的环境；新的需求与修改要求；
Response	以下之一或多个： ➢ 修改原本的代码； ➢ 测试代码是否达到需求； ➢ 部署；
Response Measure	以下之一或多个： ❖ 需要多少新技术和架构才能维护软件； ❖ 有百分之多少的功能被维护影响； ❖ 维护代码会消耗多少时间、成本、人力；

❖ Maintainability Concrete Scenario 可维护性具体方案



Source:
开发者

Stimulus: 维护已经上线的软件

Environment:
业务量上涨，需要维护新的数据接口

Response:
维护其中和数据相关的代码

Response Measure:
1. 较少地对源代码修改
2. 在 2 天之内上线新的产品
3. 80%的功能被维护

Strategy	Tactics	Impacts			
		Portability	Flexibility	Reusability	Maintainability
减少软件对于外部系统与环境的依赖	尽可能地使用内部功能和内部的方法	✓更易移植	×耦合提高不灵活	✓重用内部模块，可重用性增强 ×对外部系统的可重用性减弱	×内部耦合增多，可维护性减弱
	减少外部系统的API，尽量使用跨平台 API	✓更易移植	✓跨平台 API 使得多平台可维护 ✓外部 API 使用减少，编码更灵活	✓外部依赖减少，更独立更可能重用	✓跨平台 API 更易于跨平台维护 ✓外部依赖减少，易于维护
尽可能根据系统特征设计适合的需求与功能	进行代码设计时使用设计模式，提高可修改性	-无显著影响	✓设计模式使用，更易修改和添加	-无显著影响	✓合适的设计模式，使得代码更易于维护
	分功能分模块进行软件设计，保证模块间独立性	✓分模块移植，更高效且方便	✓分模块更灵活，使得代码易于组装，修改，添加	✓可分模块重用	✓独立模块易维护 ✓模块更易分工

Strategy	Tactics	Impacts（简写）			
		Portability	Flexibility	Reusability	Maintainability
定期维护、更新代码	在依赖更新时，审查源代码	-	✓	-	✓
	定期进行 code review	-	✓	-	✓
采用持续集成、持续部署方案	每一次通过测试就上线	-	-	-	✓
	集成测试、部署后完善功能测试	✓	-	✓	✓
确定需求时考虑用户范围，做用户调研	根据用户设计系统功能	✓	-	-	-
	设想目标用户，在代码预留设计	✓	✓	-	✓
开源	开源到社区，提供社区版本	✓	✓	✓	✓

Strategy	Tactics	Impacts（简写）			
		Portability	Flexibility	Reusability	Maintainability
开源	可由社区成员提出意见、bug	-	✓	-	✓
持续跟进用户体验	发放使用体验调查问卷	✓	-	-	✓
	上线错误报告功能与错误审查	-	-	-	✓
文档化	编写代码前完善需求、设计文档	-	×	✓	✓
	开发人员编写开发日志	-	✓	-	✓
促进开发人员合作	结对编程	-	✓	-	✓
	定期召开代码审查会议	-	✓	-	✓

Quality Attribute Debate

Definition of Quality Attribute:

- ***Quality Attributes*** are over and above of system's functionality, which is the basic statement of the system's capabilities, services, and behaviors.

在我看来，“复杂性”和“资源利用”是质量属性。在我研究之后，我认为，这两个特征与软件系统的行为密切相关。

我的研究主要从卡耐基梅隆的一篇相关技术报告展开。这篇技术报告展示了他们所归纳的软件质量属性，并从相关影响因素和影响表现展开叙述。为了更加明确我对于质量属性的定义，我查阅了互联网上的有关资源与教材。在研究过程中，我不断问自己一个问题：这两个因素符不符合质量属性的定义。围绕着其有关的表现，我进行了资料的收集。除此以外，我还分析了此前学习过还有出现过的质量属性。并且，我还考虑了复杂性与资源利用对于其他指令属性的影响和关联。最终，结合定义以及和其他质量属性相关联的比对，我得出结论：“复杂性”和“资源利用”是质量属性。

首先，“复杂性”是一个“很大”的词。在软件世界中有许多种复杂性，比如“计算复杂性”、“分支复杂性”还有“交互复杂性”。我们都知道，计算复杂性主要指的是我们在代码设计中的算法复杂度，复杂度会直接影响到我们的程序的效率能力。高效快速的代码可以为我们节省非常多的成本，也可以改善用户的体验；此外，分支复杂度等有关与代码的复杂性和程序开发紧密相关，比如分支过多且冗余的代码非常难以调试，这样的代码往往需要很高的成本用于后期维

护；交互复杂度主要是从用户角度考虑的，过度.　　复杂的 UI 设计与交互逻辑会让整个软件摸不着头脑，为用户制造很大的麻烦。

综上所述，复杂度在通常都影响着软件系统的能力、行为和服务质量。很明显，我们该把复杂度归到质量属性这一类。但是我们应该将复杂度尽可能地细分，比如我以上分清的三类。在卡耐基梅隆的一篇技术报告中，我查找到了相关交互复杂度的概念，他们认为这是一项有关系统表现的质量属性。

“资源利用”也是一项非常重要的软件系统的质量属性。在卡耐基梅隆的技术报告中，他们提到，资源是影响软件表现的重要因素。对于软件系统来说，资源主要分为硬件和软件。硬件资源主要有计算设备比如 CPU 和 GPU，还有存储设备比如内存、外部存储器，甚至寄存器；硬件主要由管理员管理，但是软件开发人员应该尽可能编写硬件友好的程序，并且在不同硬件上的适配性和利用率也影响着软件的表现。软件资源主要有现成的包资源、网络资源等，这些与可维护性和安全性密切相关，如何在软件资源利用和其他质量属性之间找到平衡点也是非常重要的课题之一。系统应该在响应用户和利用资源自建找到一个合理的平衡，这样才能使得整体的表现最优。另外，分配资源也影响着软件的表现和行为，这都是软件开发人员在开发和维护过程中需要考虑的。

综上，我认为这两者都是重要的质量属性。