### 中文版（参考对照用，英文版在下面）

##### 绪论（陈实）

软件设计开发是一个反复迭代的过程，随时面临着需求的变更，功能的增删，应用的调整。在软件设计开发的初始阶段很难建立可适应各种需求的软件架构。在面向新的需求及软件后期维护时，主要依靠软件重新架构实现，软件重构的一个重要原则是仅改变软件内部结构，而不改变其外部行为，通过软件重新架构可显著减少软件的维护成本，延长软件的使用寿命。

##### 4. 正文

###### 4.1 软件重构的定义（陈实）

软件重构是指在不改变软件的功能和外部可见性的情况下，为了改善软件的结构，提高清晰性、可扩展性和可重用性而对软件进行的改造。简而言之，重构就是改进已经写好的软件的设计。也就是说，改进后的软件其原有的功能和行为都不改变，改变的仅仅是程序内部的编码。重构是在源代码级别上进行的，目的是使原有的代码在重构后具有良好的程序结构。软件重构是改善软件可理解性、可维护性和可扩展性的关键软件技术，它是软件工程领域的研 究热点及重要实践。软件重构不仅改变了软件过分依赖前期设计的局面，力求得到恰如其分 的软件结构，同时还保持了设计的简单性以及灵活性。

###### 4.2 软件重构的作用与好处（陈实）

软件重构最大的作用和好处就是降低了系统或者软件整体的复杂度。软件开发中的复杂度当然是越低越好。在实际软件开发过程中，除了逻辑上的复杂度、设计上的复杂度之外，复杂度涉及到方方面面。

**4.2.1 程序逻辑的复杂度**

线性顺序执行的复杂度为1, 出现分支以后要乘以分支的个数。分支可以是条件判断也可以是循环。所以尽可能的避免分支的出现是降低程序逻辑复杂度的重要手段。如果程序分支不可避免，要尽可能的把程序分支放到最高的逻辑层。这样做的目的是为了避免在下层处理的时候出现发散式的分支。发散式的分支会急剧的增加程序的复杂度。复杂度越高，程序越难维护。

**4.2.2 架构设计的复杂度**

架构设计涉及到模块设计和系统设计。要尽可能的把一些公用的模块或者子系统抽取出来，这些公用的功能可能会被所有其他的业务模块或系统所调用。在调用这些公用功能的时候，越简单越好，并且调用者不需要关心具体的内部实现，只需要知道如何使用。这样做的目的是专注于业务代码的设计。

**4.2.3 系统部署的复杂度**

系统部署包含几个不同的阶段如开发阶段、测试阶段和生产阶段。不管是哪个阶段，部署的步骤越少越不容易出错。有些系统天然的需要很多指令的配置，如果是这样的情况，需要编写一个批处理的文件来简化外部使用者的部署步骤，把多个步骤变成一步。

**4.2.4 测试的复杂度**

测试分白盒测试和黑盒测试。白盒测试的复杂度直接关联着代码层级的复杂度，代码层级的复杂度越高，白盒测试的复杂度也就越高。白盒测试需要注意的一个重要问题是不要使白盒测试这部分的代码脱离实际业务代码的设计。黑盒测试的复杂度来自于业务需求分析。要有非常清晰的文档说明，需要对测试步骤和预期结果写的非常清楚。

**4.2.5 技术的复杂度**

技术的发展趋势一般是越发展越简单，功能越强大。在设计和开发的过程中，要避免使用老旧的技术。简化我们的学习过程，提高开发效率，增强整个项目的可维护性。

###### 4.3 软件重构的研究现状（刘浩宇）

（那几篇里到处搜搜）

###### 4.4 软件重构的方法与技术（陈戒）

（主要参考 01）

### 英文版（论文用）

##### Introduction

Software design and development is an iterative process, facing changes in requirements, additions and deletions of functions, and adjustments to applications at any time. It is difficult to establish a software architecture that can adapt to various needs in the initial stage of software design and development. When facing new requirements and software maintenance, it mainly relies on software rearchitecting. An important principle of software rearchitecting is to only change the internal structure of the software without changing its external behavior. Software rearchitecting can significantly reduce software maintenance costs and extend the service life of the software.

##### 4. Body

###### 4.1 Definition of Software Rearchitecting

Software rearchitecting refers to the transformation of software in order to improve the structure of the software, improve clarity, scalability and reusability without changing the function and external visibility of the software. In short, rearchitecting is improving the design of an already written software. Which means that the original functions and behaviors of the improved software will not be changed, only the code inside the program will be changed. rearchitecting is carried out at the source code level, and the purpose is to make the original code have a good program structure after rearchitecting. Software rearchitecting is a key software technology to improve software comprehensibility, maintainability and scalability. It is a research hotspot and an important practice in the field of software engineering. Software rearchitecting not only changes the situation that the software is overly dependent on the previous design, but also strives to obtain an appropriate software structure, while maintaining the simplicity and flexibility of the design.

###### 4.2 The role and benefits of software rearchitecting

The biggest function and benefit of software rearchitecting is to reduce the overall complexity of the system or software. The complexity in software development is of course as low as possible. In the actual software development process, in addition to the complexity of logic and design, complexity involves all aspects.

**4.2.1 The complexity of program logic**

The complexity of linear sequential execution is 1, and after the branch appears, it must be multiplied by the number of branches. Branches can be conditional judgments or loops. Therefore, avoiding the occurrence of branches as much as possible is an important means to reduce the complexity of program logic. If program branching is unavoidable, try to place program branching at the highest logical level. The purpose of this is to avoid divergent branches in the lower layer processing. Divergent branching will dramatically increase the complexity of the program. The higher the complexity, the harder the program is to maintain.

**4.2.2 Complexity of Architecture Design**

Architecture design involves module design and system design. It is necessary to extract some common modules or subsystems as much as possible, and these common functions may be called by other business modules or systems. When calling these public functions, the simpler the better, and the caller does not need to care about the specific internal implementation, only needs to know how to use it. The purpose of this is to focus on the design of the business or service code.

**4.2.3 Complexity of System Deployment**

System deployment consists of several different phases such as development phase, testing phase and production phase. No matter which stage it is, the fewer steps to deploy, the less error-prone it is. Some systems naturally require the configuration of many commands. If this is the case, it is necessary to write a batch file to simplify the deployment steps of external users and turn multiple steps into one step.

**4.2.4 Complexity of Testing**

Testing is divided into White Box Testing and Black Box Testing. The complexity of White Box Testing is directly related to the complexity of the code level. The higher the complexity of the code level, the higher the complexity of White Box Testing. An important issue that needs to be paid attention to in White Box Testing is not to make the code of this part of White Box Testing deviate from the design of the actual business code. The complexity of Black Box Testing comes from business requirement analysis. To have very clear documentation, it needs to write very clearly about the test steps and expected results.

**4.2.5 Complexity of Technology**

The development trend of technology is generally that the more it develops, the simpler it is, and the more powerful it is. During the design and development process, avoid using old technology. Simplify our learning process, improve development efficiency, and enhance the maintainability of the entire project.

###### 4.3 Research Status of Software Rearchitecting

###### 4.4 Methods and Techniques of Software Rearchitecting