RESEARCH ON SOFTWARE REFACTORING

软件重构技术研究

刘 辉 李光杰 著

北京理工大学出版社
BEIJING INSTITUTE OF TECHNOLOGY PRESS

软件重构技术研究

刘　辉　李光杰　著

北京理工大学出版社
BEIJING INSTITUTE OF TECHNOLOGY PRESS

图书在版编目（CIP）数据

软件重构技术研究/刘辉，李光杰著. —北京：北京理工大学出版社，2016.4

ISBN 978-7-5682-2105-4

Ⅰ.①软…　Ⅱ.①刘…　②李…　Ⅲ.①软件开发　Ⅳ.①TP311.52

中国版本图书馆 CIP 数据核字（2016）第 062351 号

出版发行／北京理工大学出版社有限责任公司
社　　址／北京市海淀区中关村南大街 5 号
邮　　编／100081
电　　话／（010）68914775（总编室）
　　（010）82562903（教材售后服务热线）
　　（010）68948351（其他图书服务热线）
网　　址／http：//www.bitpress.com.cn
经　　销／全国各地新华书店
印　　刷／保定市中画美凯印刷有限公司
开　　本／710 毫米×1000 毫米　1/16
印　　张／16.75
字　　数／266 千字
版　　次／2016 年 4 月第 1 版　2016 年 4 月第 1 次印刷
定　　价／58.00 元

责任编辑／王玲玲
文案编辑／王玲玲
责任校对／周瑞红
责任印制／王美丽

前　言

近十年来，软件重构一直是软件工程领域的一个研究热点。伴随着极限编程等轻量级软件开发过程的兴起和流行，软件重构技术在工业界获得了广泛应用，从而刺激了学术界对软件重构的深入而持久的研究。

作者及其科研团队近年来一直从事软件重构方面的科研、教学和工程工作，在软件重构的正确性验证、重构机会推荐、重构方案推荐、重构调度等方面进行了深入的研究。并且，在 IEEE Transactions on Software Engineering（TSE）、International Conference on Software Engineering（ICSE）、joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering（ESEC/FSE）、Automated Software Engineering（ASE）等国际顶级期刊与顶级国际会议发表论文十余篇；开发了 InsRefactor 等一系列的重构工具，较大幅度地实现了重构的自动化，提高了重构的效率和效果。

本书是作者在软件重构方面研究的总结。首先介绍了软件重构的概念及其研究现状，引出作者的研究工作。之后的各章分别介绍作者在模型重构的描述、验证、重构机会推荐、调度、工具、实时重构、阈值优化等方面的工作。

本书的出版及其所介绍的研究工作得到了国家自然科学基金（No.

61272169，61472034）、教育部新世纪优秀人才支持计划（No. NCET13 - 0041）的支持，在此一并表示感谢！

刘　辉
二〇一六年三月
于北京理工大学

目　录

第 1 章 绪 论

1.1 软件重构的概念

随着需求和环境的不断变化，软件代码不断被修改和升级，其质量也不断降低。面向对象程序设计语言的出现使得人们对软件的修改变得更加容易，很多人认为只需针对新的需求增加几个类即可。但实际上，在添加新类时却需要考虑类之间结构关系的变化，有时可能涉及不同类之间变量和方法的移动，甚至需要根据不同抽象程度将现有类进行分解、合并和派生。同时，不断变化的代码也会带来命名不一致、重复等问题，使得人们对代码的理解和维护变得更加困难，导致软件维护成本在软件开发中所占比例增大，最终造成软件整体质量下降的问题。

随着软件规模的不断变大，软件演化与维护成本占据了软件开发总成本的绝大部分，不可避免地成为软件周期的一部分。为了提高软件演化效率，Opdyke 于 1992 年在博士论文[1]中提出了针对应用框架的重构技术，以方便演化过程、提高演化效率和降低维护成本。软件重构指的是在不改变软件外部特征的情况下，通过调整软件内部结构来提高软件的可理解性、可维护性和可扩展性。软件重构是提高软件质量的重要方法，其核心思想是通过优化类、变量和方法使程序具有扩展性和适应性。

2002 年出版的《Refactoring Improving the Design of Existing Code》一书[2]使软件重构技术不再限制于应用框架，而是适用于所有面向对象程序。

该书使得重构技术流行起来，并不断被越来越多的软件工程师所接受和使用。目前，几乎所有流行的集成开发工具都自带重构功能，如 Eclipse 提供了 Refactoring 模块用于重构，Visual Studio 提供了 Refactor 菜单用于重构等。

1.2 代码坏味问题

代码坏味用于确定需要重构的地方和时间，常见的代码坏味问题主要归结为如下 10 类：

（1）不合适的命名（Bad Names）

研究表明，标识符占代码量的 70% 左右。有意义的标识符有利于人们对代码的理解和维护，例如，给方法取一个有意义的名称可以让人们不必阅读方法体的代码就知道方法的功能。但由于开发时间和编程习惯等因素的影响，编程人员经常对代码中的包名、类名、方法名等标识符进行随意命名，造成代码的可读性和可理解性差。当对软件进行维护或者升级时，很难根据名称判断代码功能和实现方法。当然，即使程序的最初命名符合规范，随着系统的升级和修改，标识符名称的质量也会降低。

不合适的标识符命名主要分为如下三类：

1）词汇问题

词汇上的不合适命名主要是由于编程人员在对标识符进行命名时使用的词汇不是词典中存在或领域内通用的词汇而导致的。例如，对于变量名 qs，人们不能通过该名称判断变量所代表的含义，它可能是快速排序（quick sort）的缩写，也可能是质量标准（quality standard）的缩写。对于这类由缩写词造成的名称问题，一般采用将其扩展为标准词汇的方式进行重构，因此可以根据变量所在的上下文环境将 qs 重命名为 quickSort 或 qualityStandard。

2）语法问题

语法上的不合适命名主要是指标识符的名称结构不符合业界规范。例如，变量名称不能包含动词，方法名应该是一个动词或动词短语，类名应该是一个名词或名词短语等。下面是一个典型的由标识符结构不合理造成的不合适命名问题：

```
public class ConnectDatabase {
        public void initialization();
    }
```

在上面的代码中，类名 ConnectDatabase 是一个动词短语，方法名 initialization 是一个名词，均不符合命名规范，因此需要进行重命名重构。重构后的代码如下：

```
public class DatabaseConnecter {
        public void initialize();
    }
```

3）语义问题

语义上的不合适命名包括标识符的名称前后不一致、名称与功能不一致两种。造成名称前后不一致的原因主要是同义词和多义词的存在，例如在下面的代码中：

```
public class StudentManagement{
    public Student findStudentById(String id)
    {
        for(Student student:StudentList)
        {
            if(student.getId().equals(id))
                return student;
        }
        return NULL;
    }
}
```

可以很容易发现方法 findStudentById( ) 的名称和其功能不一致。Host 等学者通过对大量程序的研究发现，以 find 作为名称而开头的方法，其功能是查找某类对象，返回数据类型应该为布尔型，即如果查找成功，返回值为 true；否则为 false。而该方法实际返回的却是对象。当方法名称和方法体的功能不一致时，我们一般根据方法体的功能对方法名称进行重命名。对于上面的方法，将方法名中的 find 改为 get 更合适，因此重构后的代码如下：

```
public class StudentManagement{
    public Student getStudentById(String id)
    {
        for(Student student:StudentList)
        {
```

```
            if(student.getId().equals(id))
                return student;
        }
        return NULL;
    }
}
```

（2）重复代码块（Duplicated Code）

当程序的不同地方出现相同的代码时，会带来代码冗余的问题。对重复代码进行修改时，很容易因为漏掉某个地方而造成不一致。这些重复代码可能出现在同一个类中，也可能出现在两个相关子类或完全不相关的类中。

对于同一个类中的重复代码，一般进行方法提取；对于处于两个相关子类中的重复代码，首先需要将重复代码提取为一个方法，然后再将新提取的方法上移（Pull Up Method）到两个相关子类的公共父类中；对于处于完全不相关的两个类中的重复代码，一般采用提取类（Extract Class）的方法将相同代码提取到一个新类中。

下面是一个典型的重复代码实例：

```
public class TicketManagement {
    public void insert(Ticket ticket){
        try{
            Class.forName("com.mysql.jdbc.Driver");
            con =DriverManager.getConnection("jdbc:
                mysql://localhost:3306/ticket","root",
                "123");
            sql ="insert into ticket values(?,?)";
            pstm =con.prepareStatement(sql);
            pstm.setString(1,ticket.getId());
            pstm.setString(2,ticket.getDest());
            pstm.executeUpdate(sql);
            ...
        }catch(Exception e){}
}
public void delete(Ticket ticket){
```

```
    try{
            Class.forName("com.mysql.jdbc.Driver");
            con =DriverManager.getConnection("jdbc:
                 mysql://localhost:3306/ticket","root",
                 "123");
            sql = "delete ticket where id =?";
            pstm = con.prepareStatement(sql);
            pstm.setString(1,ticket.getId());
            pstm.executeUpdate(sql);
            ...
        }catch(Exception e){}
    }
}
```

上面的类是典型的数据库访问类，其中 insert 和 delete 方法分别实现数据插入和删除功能。这两个方法中含有相同的数据库连接代码段，可以采用提取方法的方式将其重构为如下代码：

```
public class TicketManagement {
    String driver = "com.mysql.jdbc.Driver";
    String url = "jdbc:mysql://localhost:3306/";
    String database = "ticket";
    String user = "root";
    String password = "123456";
    public void setConnection()
    {
        Class.forName(driver);
        con =DriverManager.getConnection(url +databa-
            se,user,password);
    }
    public void insert(Ticket ticket){
        try{
            getConnection();
            sql = "insert into ticket values(?,?)";
```

```
            pstm = con.prepareStatement(sql);
            pstm.setString(1,ticket.getId());
            pstm.setString(2,ticket.getDest());
            ...
          }catch(Exception e){}
    }
    public void delete(Ticket ticket){
       try{
            getConnection();
            sql = "delete ticket where id = ?";
            pstm = con.prepareStatement(sql);
            pstm.setString(1,ticket.getId());
            pstm.executeUpdate(sql);
            ...
         }catch(Exception e){}
    }
}
```

（3）长方法（Long Method）

当某个方法包含的功能过多或语句体过长时，人们将很难理解每个代码段的具体功能和实现方法，即使是开发人员自己，在一段时间之后也会忘记当初的设计思路。造成长方法的原因主要有方法过长、逻辑混乱和临时变量过多。对于语句过长的方法，一般采用提取方法的重构方式将实现每个功能的代码段抽取为一个新方法。对于临时变量过多的长方法，可以利用查询替换临时变量（Replace Temp With Query）、引入参数对象（Introduce Parameter Object）、保持整体对象（Preserve Whole Object）和利用方法对象替换方法（Replace Method With Method Object）、分解条件（Decompose Conditional）和长函数（Long Functions）等方式进行重构。

下面是一个典型的长方法实例：

```
public class Order
{
    ...
    public double compute(){
```