

# 测试驱动开发及开发实践

张 扬, 黄厚宽

(北京交通大学 计算机与信息技术学院, 北京 100044)

**摘 要:** 极限编程是适应于中小型团队在需求不明确或迅速变化的情况下进行软件开发的轻量级方法学。测试驱动开发作为极限编程思想的一种主要实践, 可以有效地让程序开发人员开发出更高品质的、经过完整测试的程序。文中介绍了测试驱动开发思想, 对测试驱动开发过程给出了清晰的流程, 总结了测试驱动开发的多种模式。最后介绍了如何用JUnit进行测试驱动开发。

**关键词:** 极限编程; 测试驱动开发; JUnit

**中图分类号:** TP311.52

**文献标识码:** A

**文章编号:** 1673-629X(2006)05-0074-03

## Test Driven Development and Practice

ZHANG Yang, HUANG Hou-kuan

(School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

**Abstract:** Extreme programming (XP) is a kind of lean development method, which adapts to the medium and small-scale group who carry on software development in a situation that the demand is indeterminate or changing rapidly. Test driven development (TDD) is a kind of main practice of a programming thought of XP. TDD can let the procedure developer develop more high-quality programs which have already passed intact test. The article introduces test driven development thought, and recommends TDD developing procedure clearly. Then, summarizes many kinds of modes of TDD. Finally, the article recommends how to carry on test driven development with JUnit.

**Key words:** extreme programming; test driven development; JUnit

### 1 极限编程与测试驱动开发

极限编程 (Extreme Programming, XP) 是适应于中小型团队在需求不明确或迅速变化的情况下进行软件开发的轻量级方法学<sup>[1]</sup>。极限编程是一种轻量、高效、低风险、柔性、可预测、科学而且充满乐趣的软件开发方式。它作为一种方法论有如下特点:

- \* 周期较短, 在短周期内进行早期、具体和持续的反馈。
- \* 递增地进行计划编制。这种方法迅速提供一个总体计划, 然后在项目的整个生命周期内不断发展。
- \* 具有针对不断变化的业务需求灵活地对功能的实现进行计划的能力。
- \* 依赖于由程序员或客户编写的自动测试来监控开发进度, 使得系统得以发展并及早捕获缺陷。
- \* 依赖于口头交流、测试和源代码来沟通系统的结构和意图。
- \* 依赖于在系统存在期间一直持续的进化式设计过程。

\* 对程序员的技术水平要求不高, 但要求他们紧密协作。

\* 既可满足程序员的短期本能, 也满足项目的长期利益。

极限编程有4个原则, 分别为沟通、简单、反馈和勇气<sup>[2]</sup>。XP旨在采用许多只能通过沟通完成的实践来保持良好的沟通, 如单元测试、结对编程及任务估算。XP假设不用深谋远虑, 想的很深很远才开始动手。它要求今天能实现今天的设计就可, 不去预先考虑解决明天或后天的事情。XP强调即时的反馈。它有两种反馈模式, 分别以分钟和天的级别进行反馈; 以周和月的级别进行反馈。程序员为系统中所有可能出错的逻辑编写单元测试。他们每分钟都得到有关系统状态的具体反馈。客户隔两到三周检查一次日程, 查看开发团队的整体速度是否与计划相符, 并随之调整计划。XP要求程序员有勇气即时地修复缺陷, 即使这使原来运行通过的测试中出现了错误; 如果一天快要结束, 而代码依然失控, XP要求程序员有勇气放弃原来的代码。总之, 极限编程要求程序员快速反馈, 把每个问题都看成可以用近乎荒谬的简单设计来解决, 递增进行微小更改来解决问题。

测试驱动开发 (Test Driven Development, TDD) 作为XP编程思想的一种主要实践, 可以有效地让程序开发人员开发出更高品质的、经过完整测试的程序。测试驱动开

收稿日期: 2005-08-07

**作者简介:** 张 扬 (1983-), 男, 山西吕梁人, 硕士研究生, 研究方向为数据挖掘与数据仓库; 黄厚宽, 教授, 博士生导师, 研究方向为人工智能及模式识别。

发以测试作为开发过程的开端,它要求在编写任何产品代码之前,首先编写用于定义产品代码行为的测试,而编写的产品代码又要以使测试通过为目标。TDD不是一种开发工具,也不是一种测试方法,它是一种编码之前进行单元测试的软件开发思想<sup>[1]</sup>。

## 2 测试驱动开发过程与模式

### 2.1 测试驱动开发流程

TDD开发过程有别于传统开发流程(Waterfall),它在进行简单的概要设计后,首先进行的是测试用例的编写,然后执行测试用例进行测试。测试失败,则进行编码驱使测试通过,这就是所谓的测试驱动。最终,测试得到通过,再对代码进行重构,优化代码结构和性能。而传统流程则先进行概要设计,然后在概要设计基础上进行详细设计,在详细设计阶段尽可能设想全部问题和需求的解决方法,然后才开始编码实现详细设计。TDD开发流程图如图1所示。

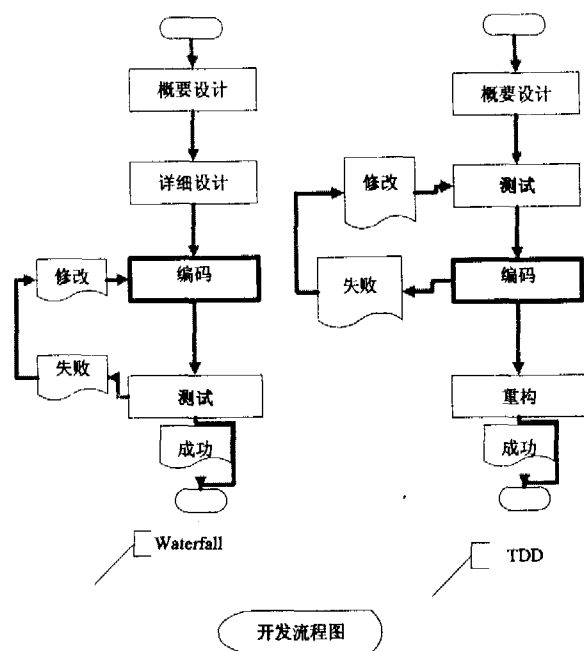


图1 开发流程图

测试驱动开发的精髓在于:将测试方案设计工作提前,在编写代码之前先做这一项工作;从测试的角度来验证设计,推导设计;同时将测试方案当作行为的准绳,有效地利用其检验代码编写的每一步,实时验证其正确性,实现软件开发过程的“小步快走”。

测试驱动开发预期有更好的代码质量。因为所有的代码都是由测试驱动而来,所以所有代码都是可以测试和必需的。在测试驱动开发过程中,调试较传统开发方式更加容易,因为在此之前所有的代码都已成功通过测试,Bug总是由那些新添的测试或代码引入的。在传统的开发方式中,有时候程序员很难准确定位Bug的真正所在,这是一件很头疼的事情。测试驱动开发较之传统流程开发有更小的开发压力,它强调今天做好今天事情既可,而

且尽可能简单地实现眼前的需求。而每一个今天的开始又建立在正确而健壮的以往代码的基础上。测试驱动开发较之传统流程开发有更快的开发速度,因为每一个当前需要实现的功能都有测试需求作为依托,从而明确并且规模很小,这有利于程序员所谓的“小步快走”。

测试驱动开发的详细开发过程如图2所示。

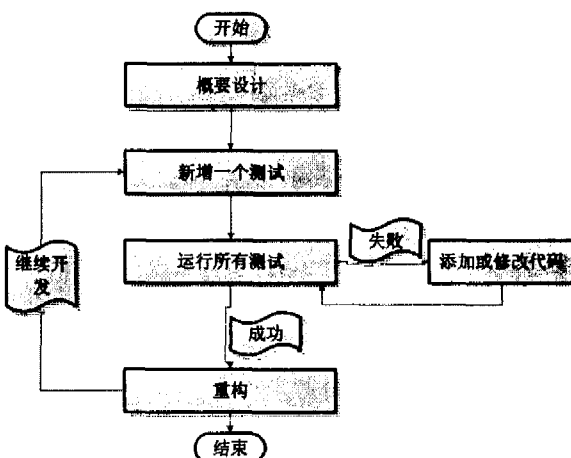


图2 TDD详细开发流程图

最后的重构步骤指在不改变代码外在行为的前提下,对代码做出修改,以改进程序的内部结构,提高其可理解性,降低其修改成本。对极限编程而言,重构并非“预先设计”的替代品,只是减轻了“预先设计”的压力。可以在得到第一个可被接受的解决方案后开始编程,而不需要保证“预先设计”正确无误。重构应该随时随地进行。

### 2.2 测试驱动开发模式

在测试驱动开发中,关键的问题如下:什么时候进行测试、如何选择要测试的逻辑和如何选择要测试的数据。测试驱动开发模式指导程序员如何解决上述问题。

#### \* 测试相互独立。

在测试驱动开发中,所运行的各种测试之间关系的期望状态是没有任何相互影响的。相互独立的测试意味着所有的测试都是不依赖于顺序的,可以随便从这些测试中挑出部分测试来运行。程序员必须将自己的问题分解为一些彼此正交的小问题,这样就使得为每个测试搭建环境简单而快捷。独立测试鼓励利用高度内聚、低度耦合的对象组合来解决问题。

#### \* 写出测试列表。

程序员在开始写测试之前,应该写一个包含所有必须要编写的测试的清单。那么,记录到列表上的就是当前程序员要去实现的测试。首先将需要实现的每种操作的范例都记录在清单上。对于目前尚不存在的操作,将其空版本记录在清单上。

#### \* 测试和断言优先。

在测试驱动开发中,程序员构建一个系统应该是从其对最终系统的描述开始的。程序员应该从希望最终代码能够通过测试开始编写一项功能。相应地,程序员应该从测试完成时能够通过的断言开始编写一个测试。在测

试优先的测试里程序员应该尽量使用容易让人理解的数据,一般不用一个常量来表达多种意思。

一般从测试列表中选择具有指导意义并且比较有把握实现的测试来进行编写。当使用一个新类里的一种新的方法时,不直接用它来编写程序,而是编写一个小测试来验证这个 API 的工作是否符合人们的愿望。当出现某种与当前讨论话题并不直接相关的想法时,那么就在列表中增加一个测试然后重新回到论题上来。当发现一个错误的时候,首先写一个尽可能小的测试并使其运行,然后再去修复这个错误。

### 3 利用 JUnit 进行测试驱动开发

单元测试(Unit Tests)指许多段的程序,写这些程序的目的是用来成批执行,以验证程序员所写的类(Class)。每一格 Unit Test 都负责送一个消息(Message)给一个特定的 Class,并且验证所传回来的值就是该 Test 所预期的答案<sup>[3]</sup>。单元测试即用来测试程序员在主要程序中所有类的公共介面(Public Interfaces)的程式。单元测试的重点在于验证程序员所写方法(Methods)所产生的结果与预期相同。常用编程语言及其对应可用来进行作其测试工具的单元测试工具,如表 1 所示。

表 1 单元测试工具表

Language	Test Tools
Java	JUnit
SmallTalk	SUnit
C++	CppUnit
Python	PyUnit
VisualBasic	VBUnit

#### 3.1 Java 单元测试框架 JUnit

JUnit 就是 Java 语言的一套程序库(toolkit),适合于普通程序员使用原本就熟悉的开发语言(Java)以及开发工具(IDE)写出这些单元测试来<sup>[4]</sup>。JUnit 源自于由 Kent Beck(极限编程创始人)所设计的单元测试框架(Unit Test Framework)<sup>[5]</sup>。

#### 3.2 利用 JUnit 进行测试驱动开发

在 Eclipse 建立 JUnit 测试,并进行驱动开发。现在开发一个“Hello World”的例子。按照 TDD 的规则,应该在代码建立以前先把测试写好。为了能够在某处开始,假设未来的类名是 HelloWorld,并且有一个方法 Say(),这个方法返回 String 的值(例如“Hello World!”)。

根据设定的程序功能,写出测试代码如下:

```
import junit.framework.TestCase;

public class TestThatWeGetHelloWorldPrompt
extends TestCase {

    public TestThatWeGetHelloWorldPrompt(
        String name) {
        super(name);
    }

    public void testSay() {
```

```
HelloWorld hi = new HelloWorld();
assertEquals("Hello World!", hi.say());
}

public static void main(String[] args) {
    junit.textui.TestRunner.run(
        TestThatWeGetHelloWorldPrompt.class);
}
```

建立测试案例的步骤如下:

- 1) 建立一个 junit.framework.TestCase 的实例。
- 2) 定义一些以“test”开头的无返回方法(例如 testWasTransactionSuccessful(), testShow(), 等等)。

TestThatWeGetHelloWorldPrompt.java 包含这些: TestCase 的子类和一个叫做 testSay() 的方法。这个方法调用了 assertEquals() 函数,它用来比较预期的值和由 say() 返回的值。

main() 方法用来运行测试和显示输出。JUnit 的 TestRunner 处理测试,提供基于图像和文本的输出表现形式。我们使用基于文本的版本,因为 Eclipse 支持它,且也适合我们。当开始运行后,基于文本的版本测试会以文本形式输出,Eclipse 会把这些输出自动变成图像界面的输出。

现在建立被测试代码:

```
public class HelloWorld {
    public String say() {
        return("Hello World!");
    }
}
```

状态条为绿,测试通过,代码实现既定目的。

### 4 测试驱动开发中容易陷入的误区

测试驱动开发中容易陷入以下几种误区:

- \* 基于各种原因的不写测试。

程序员在很多情况下有很多客观或情绪上的理由来排斥测试代码先行。他们会想:进度太紧,没有时间写测试;TDD 感觉比较古怪,无法接受;本项目有它自己的特殊性,并不适合测试驱动开发的方式;有些代码根本写不出测试……不论是在写程序的时候,还是在应用新的开发过程的时候,人都是会犯错的。测试驱动开发可以让程序员很方便地研究它们、定义它们、检查它们并可得到清晰的反馈结果。最终,程序员会发现他们不知不觉间解决了全部的问题。

- \* 测试的覆盖面太小。

有些人自诩也在应用 TDD 进行开发,但在上万行的工程代码中,只有 2,3 个 TestCase。只对某些“核心”模块或复杂算法进行测试驱动。或者仅仅是某个人试试 IDE 的新功能的结果。这种使用方式根本不是真正的测试驱动开发。测试用例并不是一条安全索,而是设计和实现的

(下转第 79 页)

查询图像,查询结果从左到右与查询图像的相似程度是递减的,(b)为分割结果,这里的查询图像来自于 Brodatz 数据库。图3展示的是另外一个来自于 MIT Vistex 数据库的查询图像纹理4的查询结果,同样,(b)也为分割结果。

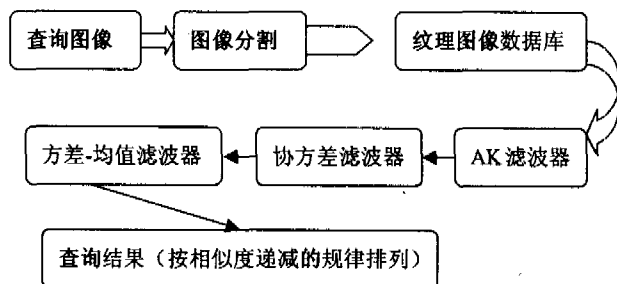


图1 多重滤波器查询结构图

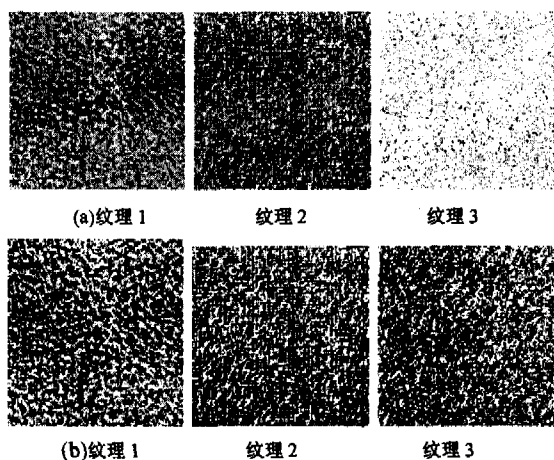


图2 纹理图像1的查询结果

通过进行非监督纹理分割,可以简便地理解图像的结构,同时可以提取一些纹理特征进行图像查询。运用假设检验,可以将纹理区域分到与它最类似的区域中。

#### 4 结论

提出了一种非监督分割框架用于纹理图像查询。通

过这种分割方法,自动提取图像各类别的一组特征参数。基于这些特征参数,可以进行有效的纹理图像查询。

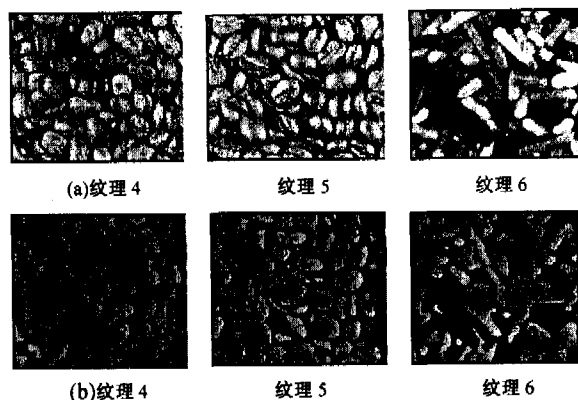


图3 纹理图像4的查询结果

#### 参考文献:

- [1] Nair D, Aggarwal J K. A focused target segmentation paradigm [A]. 4th European Conference on Computer Vision [C]. [s. l.]: [s. n.], 1996. 579-588.
- [2] Flickner M. Query by image and video content: The QBIC system [J]. IEEE Computer, 1995(9): 23-32.
- [3] Price K. Image segmentation: A comment on studies in global and local histogram-guided relaxation algorithms [J]. IEEE Trans on PAMI, 1984(3): 247-249.
- [4] Pavlidis T. Structural Pattern Recognition [M]. [s. l.]: Springer Verlag, 1991. 32-39.
- [5] Beulieu J M, Goldberg M. Hierarchy in picture segmentation: A stepwise optimization approach [J]. IEEE Trans on PAMI, 1989(2): 150-163.
- [6] Chellappa R, Jain A. Markov Random Fields: Theory and Applications [M]. [s. l.]: McGraw-Hill Book Company, 1993. 48-59.

(上接第76页)

手段,是一个组成部分。

\* 测试跨度过大。

提供了系统最外层的输入,然后就变魔术般地输出最终结果。作为验收测试是适当的,而且一般来说,写下的第一个测试就是这样的高层次测试。但是,仅仅有这种测试是不够的。需要有更多的说明产生这一结果的内部过程机制的测试代码。

判断测试跨度的一个标准就是,每个测试应该在较短时间内得以通过(比如几小时之内)。如果写下一个测试后进行一周编码后此测试才能通过,那么 TDD 对你不会有太大的帮助。应该尽快地通过虚拟对象通过高层测试,并将注意力集中到较低层次对象需要满足的测试上。

\* 测试针对代码而不是针对功能。

测试驱动开发中的测试是为了驱动开发,从而产生能

实现预期功能和意图的代码,而不是求全的测试。测试驱动开发中的测试不必求全覆盖所有的测试情况,而应该清楚地表达你希望程序达到的意图,并保证程序真的符合这些意图。

#### 参考文献:

- [1] Beck K. 测试驱动开发(中文版) [M]. 孙平平译. 北京: 中国电力出版社, 2004.
- [2] Beck K. 解析极限编程—拥抱变化 [M]. 唐东铭译. 北京: 人民邮电出版社, 2002.
- [3] Massol V. JUnit IN ACTION(中文版) [M]. 鲍志云译. 北京: 电子工业出版社, 2005.
- [4] Hunt A, Thomas D. 单元测试之道 Java 版——使用 Junit [M]. 陈伟柱译. 北京: 电子工业出版社, 2005.
- [5] 王东刚. 软件测试与 Junit 实践 [M]. 北京: 人民邮电出版社, 2005.