RUP®/XP 準則：先測試設計和重構

Robert C. Martin
Object Mentor, Inc.

Rational Software 白皮書

TP 159, 03/01

Rational®
the software development company

目錄

概觀

真正創新的作法在軟體競爭市場中並不常見。結構式程式設計就是其中作法之一。物件導向是其中之一。先測試設計和重構也是其中之一。

一個正確而簡單的重構定義就是以輕微變更保留程式的功能但卻變更其結構的一項行動。此定義的含意就是軟體有兩個特殊價值。第一，軟體用途的價值。第二，軟體結構的價值。根據剛才所給的定義，重構是維護及改進軟體結構價值的技術。

重構更準確的定義就是設計及實作軟體的技術，透過無數細微變更將焦點轉移到新增功能和改進結構。此定義延伸 Fowler 在他書中描述的這個字 *重構* 的意義（請看參照 [1]），並說明在 *eXtreme Programming* (XP)（請看參照 [2]）的流程中設計及撰寫軟體的方式。

先測試設計和重構是先設計再改進程式碼的作法，它在撰寫使案例通過的程式碼之前先撰寫測試案例。程式設計師選取作業，撰寫一兩個極簡易單元的測試案例（因爲程式不執行這項作業而失敗），然後修改程式，使測試通過。程式設計師持續新增更多測試案例，使它們通過，直到軟體完成它應該做的事爲止。然後程式設計師一次一小步地改進系統結構，在每一步之間執行所有測試，以確定沒有發生任何中斷。

重構範例

說明先測試設計和重構的最佳方式就是舉例。因此，我們在這裡將著手設計和實作一個小程式，示範如何完成重構。請注意，在 XP，有一對使用相同工作站的程式設計師將完成您即將看到的活動。[1]

我們將建置的應用程式是一個簡單自動行駛哩數日誌。每次使用者到加油站時，會輸入購買的油量、油價和車輛里程錶的現行讀數。系統記錄這些項目並產生某些有用的報告。我們的實作語言是 Java。

首先，我們在 Listing 1 撰寫程式碼：

TestAutoMileageLog.java Listing 1

```
import junit.framework.*;

public class TestAutoMileageLog extends TestCase
{
    public TestAutoMileageLog(String name)
    {
        super(name);
    }
}
```

我們撰寫的第一項是包含單元測試的架構。這看起來倒退了，但它是先測試概念的基礎。我們在撰寫實際應用程式碼之前先撰寫測試程式碼。我們一面進行時，您就可以看到它是如何運作的。

我們使用的測試架構叫作 JUnit，它是由 Kent Beck 和 Erich Gamma 撰寫的簡易單元測試架構。上述程式碼是要設定它所需要的一切程式碼。

現在我們需要考量第一個測試案例。此軟體的用途是什麼？它必須做的其中一件事就是記錄加油站造訪人次。這表示必須有一個保留相關資料的 FuelingStationVisit 物件。因此我們撰寫一個測試來建立此物件，然後查詢它的欄位。

我們從撰寫測試函數開始。在 JUnit，測試函數是從 TestCase 衍生的類別的任何方法，其名稱開頭爲四個字母 “test”。請參閱 Listing 2。

TestAutoMileageLog.java Listing 2

```
import junit.framework.*;
```

[1]請參閱 Rational Software 白皮書，標題：*RUP/XP 準則：雙人程式設計*。

```
public class TestAutoMileageLog extends TestCase
{
	public TestAutoMileageLog(String name)
	{
		super(name);
	}

	public void testCreateFuelingStationVisit()
	{
		FuelingStationVisit v = new FuelingStationVisit();
	}
}
```

新程式碼是粗體字。請注意，我們已經建立了一個新物件，叫作 FuelingStationVisit。我們尚未給它任何建構引數。目前我們只想要確定是否能夠建立該物件。

顯然地，這是不會編譯的（如果可以編譯，會很有意思）。若要讓它能夠編譯，我們必須撰寫 FuelingStationVisit 物件的程式碼。請參閱 Listing 3。

TestAutoMileageLog.java Listing 3.1

```
import junit.framework.*;
import FuelingStationVisit;

public class TestAutoMileageLog extends TestCase
{
	public TestAutoMileageLog(String name)
	{
		super(name);
	}

	public void testCreateFuelingStationVisit()
	{
		FuelingStationVisit v = new FuelingStationVisit();
	}
}
```

FuelingStationVisit.java Listing 3.2

```
public class FuelingStationVisit
{
}
```

此程式碼可以編譯，且測試可以執行，因此我們可以開始新增想要的功能。

TestAutoMileageLog.java Listing 4.1

```
import junit.framework.*;
import FuelingStationVisit;
import java.util.Date;

public class TestAutoMileageLog extends TestCase
{
	public TestAutoMileageLog(String name)
	{
		super(name);
	}

	public void testCreateFuelingStationVisit()
	{
```

```
        Date date = new Date();
        double fuel = 2.0;  // 2 gallons.
        double cost = 1.87*2; // Price = $1.87 per gallon
        int mileage = 1000; // odometer reading.
        double delta = 0.0001; //tolerance on floating point equality.

        FuelingStationVisit v =
              new FuelingStationVisit(date, fuel, cost, mileage);
        assertEquals(date, v.getDate());
        assertEquals(1.87*2, v.getCost(), delta);
        assertEquals(2, v.getFuel(), delta);
        assertEquals(1000, v.getMileage());
        assertEquals(1.87, v.getPrice(), delta);
    }
}
```

FuelingStationVisit.java Listing 4.2

```
import java.util.Date;

public class FuelingStationVisit
{
    private Date itsDate;
    private double itsFuel;
    private double itsCost;
    private int itsMileage;

    public FuelingStationVisit(Date date, double fuel,
                               double cost, int mileage)
    {
        itsDate = date;
        itsFuel = fuel;
        itsCost = cost;
        itsMileage = mileage;
    }

    public Date getDate() {return itsDate;}
    public double getFuel() {return itsFuel;}
    public double getCost() {return itsCost;}
    public double getPrice() {return itsCost/itsFuel;}
    public int getMileage() {return itsMileage;}
}
```

這個步驟是先把測試新增至 TestAutoMileageLog，然後再把方法新增至 FuelingStationVisit。在可以開始測試之前，有三個或四個相關的編譯。第一次執行測試。

您可能不瞭解這個極端漸進主義爲我們帶來了什麼。我們不能只撰寫 FuelingStationVisit，然後再撰寫測試程式碼嗎？真的需要測試 FuelingStationVisit 嗎？到目前爲止，先撰寫測試，甚至撰寫它們全部，沒有爲我們帶來什麼好處—除了一項。我們很清楚地知道，上述程式碼會編譯和執行。因此我們知道，如果下一次變更造成編譯器錯誤或測試失敗，則問題是出在變更，而不是先前的程式碼。這看起來只是一點點好處，但以後它會變得更加重要。

接下來，我們需要在某個地方放置 FuelingStationVisit 物件。有些物件需要保留它們。那會是什麼物件？那是想要保留及管理此資訊的*使用者*，因此，我們可以建立一個 User 物件來保留 FuelingStationVisit 物件。然而，我卻覺得 FuelingStationVisit 物件中的行駛哩數欄位很奇怪。行駛哩數是車輛的一個屬性。 FuelingStationVisit 物件在有人來加油時記錄 車輛狀態的一部分。因此，我們應該建立一個 Vehicle 物件，來保留 FuelingStationVisit 物件。

TestAutoMileageLog.java Listing 5.1

```
import junit.framework.*;
import FuelingStationVisit;
import java.util.Date;

public class TestAutoMileageLog extends TestCase
{
	public TestAutoMileageLog(String name)
	{
		super(name);
	}



	public void testCreateVehicle()
	{
		Vehicle v = new Vehicle();
		assertEquals(0, v.getNumberOfVisits());
	}
}
```

Vehicle.java Listing 5.2

```
public class Vehicle
{
	public int getNumberOfVisits()
	{
		return 0;
	}
}
```

Listing 5 顯示起始步驟。我們已建立一個新測試函數，叫作 testCreateVehicle。此函數建立 Vehicle，並確定它內含的造訪人次是零。getNumberOfVisits 的實作明顯錯誤，但它的好處是可以使測試通過。這可讓我們將它重構爲更好的解決方案。

Vehicle.java Listing 6

```
import java.util.Vector;

public class Vehicle
{
	private Vector itsVisits = new Vector();

	public int getNumberOfVisits()
	{
		return itsVisits.size();
	}
}
```

同樣地，測試通過。請注意，我們是執行所有測試，而不只是 testCreateVehicle 函數。這可確保我們所做的變更不會破壞以前可以運作的一切。

接下來，我們應該想想要如何在 Vehicle 中新增造訪人次。什麼是最簡單的測試案例？

TestAutoMileageLog.java Listing 7

```
	public void testAddVisit()
	{
		double fuel = 2.0;  // 2 gallons.
		double cost = 1.87*2; // Price = $1.87 per gallon
```

```
        int mileage = 1000; // odometer reading.
        double delta = 0.0001; //tolerance on floating point equality.

        Vehicle v = new Vehicle();
        v.addFuelingStationVisit(fuel, cost, mileage);
        assertEquals(1, v.getNumberOfVisits());
    }
```

請注意，我們沒有在此測試中建立 FuelingStationVisit 物件。看起來好像 Vehicle 的 addFuelingStationVisit 方法必須建立 FuelingStationVisit 物件，然後將它新增至清單中。

Vehicle.java Listing 8

```
    public void addFuelingStationVisit(double fuel, double cost, int mileage)
    {
        FuelingStationVisit v =
            new FuelingStationVisit(new Date(), fuel, cost, mileage);
        itsVisits.add(v);
    }
```

同樣地，所有測試通過。

我們覺得 testAddVisit 和 testCreateFuelingStationVisit 這兩個函數中有重複的程式碼不太好。這兩個函數都建立相同區域變數並起始設定它們成爲相同值。我們想擺脫此重複性。因此，我們重構測試程式，使區域變數成爲成員變數。

TestAutoMileageLog.java Listing 9

```
import junit.framework.*;
import FuelingStationVisit;
import java.util.Date;

public class TestAutoMileageLog extends TestCase
{
    private double fuel = 2.0;          // 2 gallons.
    private double cost = 1.87 * 2;  // Price = $1.87 per gallon
    private int mileage = 1000;             // odometer reading.
    private double delta = .0001;           //tolerance on floating point equality.

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

    public void testCreateFuelingStationVisit()
    {
        Date date = new Date();

        FuelingStationVisit v =
            new FuelingStationVisit(date, fuel, cost, mileage);
        assertEquals(date, v.getDate());
        assertEquals(1.87*2, v.getCost(), delta);
        assertEquals(2, v.getFuel(), delta);
        assertEquals(1000, v.getMileage());
        assertEquals(1.87, v.getPrice(), delta);
    }

    public void testCreateVehicle()
    {
        Vehicle v = new Vehicle();
        assertEquals(0, v.getNumberOfVisits());
```

```
    }

    public void testAddVisit()
    {
        Vehicle v = new Vehicle();
        v.addFuelingStationVisit(fuel, cost, mileage);
        assertEquals(1, v.getNumberOfVisits());
    }
}
```

此特定重構有一個名稱。它叫作 PROMOTE TEMP TO FIELD。您可以在參照 [1] 和在 www.refactoring.com 尋找類似的重構和套用它們的程序的清單。

請注意，單元測試的存在可讓我們快速確認此重構沒有造成破壞。當我們重構和重組應用程式時，我們將繼續利用這個優點。每當我們變更程式碼之後覺得後悔時，我們可以依靠測試來確定一切仍可運作。

在將 FuelingStationVisit 物件新增至 Vehicle 之後，現在我們可以要求 Vehicle 產生報告。我們先撰寫測試案例，從最簡單的案例開始。

TestAutoMileageLog.java Listing 10

```
    public void testSingleVisitMileageReport()
    {
        Vehicle v = new Vehicle();
        v.addFuelingVisit(fuel, cost, mileage);
        MileageReport r = v.generateMileageReport();
        assertEquals(0,r.getMilesDriven());
        assertEquals(fuel,r.getFuelConsumed());
        assertEquals(0,r.getMilesPerGallon());
        assertEquals(cost,r.getTotalFuelCost());
    }
```

爲了撰寫此測試案例，我們必須徹底地想清楚與產生報告有關的問題。首先，我們決定 Vehicle 應該有一個叫作 generateMileageReport 的方法。接下來，我們決定此函數應該傳回一個叫作 MileageReport 的物件。最後，我們決定 MileageReport 應該有數個查詢方法。

這些查詢方法傳回的值非常有趣。加一次油不足以計算行駛哩數或每加侖哩數。若要計算這些值，我們需要至少加兩次油。但另一方面，加一次油就足以計算耗油量和油價。

當然，測試案例不會編譯。因此，我們必須新增適當的方法和類別。首先，我們只新增足夠的程式碼使它編譯，但使其測試失敗。

Vehicle.java Listing 11.1

```
    public MileageReport generateMileageReport()
    {
        return new MileageReport();
    }
```

TestAutoMileageLog.java Listing 11.2

```
    public void testSingleVisitMileageReport()
    {
        Vehicle v = new Vehicle();
        v.addFuelingVisit(fuel, cost, mileage);
        MileageReport r = v.generateMileageReport();
        assertEquals(0,r.getMilesDriven());
        assertEquals(fuel,r.getFuelConsumed());
        assertEquals(0,r.getMilesPerGallon());
        assertEquals(cost,r.getTotalFuelCost());
```

```
    }
```

MileageReport.java Listing 11.3

```
public class MileageReport
{
    public int getMilesDriven() {return itsMilesDriven;}
    public double getMilesPerGallon() {return itsMilesPerGallon;}
    public double getTotalFuelCost() {return itsTotalFuelCost;}
    public double getFuelConsumed() {return itsFuelConsumed;}

    private int itsMilesDriven;
    private double itsMilesPerGallon;
    private double itsTotalFuelCost;
    private double itsFuelConsumed;
}
```

Listing 11 中的程式碼會編譯及執行，但測試失敗。現在我們需要重構程式碼，使它通過測試。首先，我們將儘可能採用最簡單的方法。

Vehicle.java Listing 12.1

```
public MileageReport generateMileageReport()
{
    MileageReport r =  new MileageReport();
    FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(0);
    r.setMilesPerGallon(0);
    r.setMilesDriven(0);
    r.setTotalFuelCost(v.getCost());
    r.setFuelConsumed(v.getFuel());
    return r;
}
```

MileageReport.java Listing 12.2

```
public void setMilesPerGallon(double mpg) {itsMilesPerGallon = mpg;}
public void setMilesDriven(int miles)     {itsMilesDriven=miles;}
public void setTotalFuelCost(double cost) {itsTotalFuelCost=cost;}
public void setFuelConsumed(double fuel)  {itsFuelConsumed=fuel;}
```

我們假設 Vehicle 只加一次油。（別擔心；稍後我們會針對其他狀況新增其他測試案例）。我們適當地設定 MileageReport 的欄位，然後傳回它。

以此方式實作 generateMileageReport 不夠聰明，因爲我們當然知道實作充其量並不完整。然而，以細微增量實作的好處，是每一次編譯和測試之間沒有太大的改變。萬一出錯，我們總是可以回到上一個版本，重新開始。我們不必除錯。

Listing 12 中的程式碼會編譯和通過測試，但它顯然不完整。爲了使它完整，我們需要想想其他測試案例。

- 沒有加油的車輛
- 有多次加油的車輛

沒有加油的案例很簡單。Listing 13.1 中的測試案例失敗，Listing 13.2 中的程式碼使它再度通過。

TestAutoMileageLog.java Listing 13.1

```
public void testNoVisitsMileageReport()
{
	Vehicle v = new Vehicle();
	MileageReport r = v.generateMileageReport();
	assertEquals(0,r.getMilesDriven());
	assertEquals(0,r.getFuelConsumed(),delta);
	assertEquals(0,r.getMilesPerGallon(),delta);
	assertEquals(0,r.getTotalFuelCost(),delta);
}
```

Vehicle.java Listing 13.2

```
public MileageReport generateMileageReport()
{
	MileageReport r =  new MileageReport();
	if (itsVisits.size() == 0)
	{
		r.setMilesPerGallon(0);
		r.setMilesDriven(0);
		r.setTotalFuelCost(0);
		r.setFuelConsumed(0);
	}
	else
	{
		FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(0);
		r.setMilesPerGallon(0);
		r.setMilesDriven(0);
		r.setTotalFuelCost(v.getCost());
		r.setFuelConsumed(v.getFuel());
	}
	return r;
}
```

接下來，我們需要考量處理多次加油的測試案例。

TestAutoMileageLog.java Listing 14

```
public void testMultipleVisitMileageReport()
{
	Vehicle v = new Vehicle();
	v.addFuelingStationVisit(5, 6.10, 17942);
	v.addFuelingStationVisit(9.8, 12.24, 18234);
	v.addFuelingStationVisit(8.3, 10.11, 18483);
	MileageReport r = v.generateMileageReport();
	assertEquals(541, r.getMilesDriven());
	assertEquals(23.1, r.getFuelConsumed(), delta);
	assertEquals(23.41991, r.getMilesPerGallon(), delta);
	assertEquals(28.45, r.getTotalFuelCost(), delta);
}
```

我們已選擇將三次加油放置到 Vehicle 中。我們以大約每加侖 1.2 美元作爲成本的基礎，以大約每加侖 30 哩 (mpg) 作爲行駛哩數的基礎。因此，我們使用 9.8 加侖旅行 292 哩，成本是 $12.24。

這裡有一個奇怪的問題。我們以大約 30 mpg 作爲每一個里程錶讀數的基礎。然而，若將行駛距離 541 除以耗油加侖數 23.1，得到的答案是 23.41991 mpg。爲什麼會有這樣的差異？爲什麼得到的答案不是接近 30 mpg？

經過仔細思考之後，才明白耗油量並不是每次加油所加的所有燃料的總和。燃料的消耗是介於兩次加油*之間*。在計算 mpg 時，不應該考慮第一次加油的燃料。

TestAutoMileageLog.java Listing 15

```
public void testMultipleVisitMileageReport()
{
	Vehicle v = new Vehicle();
	v.addFuelingStationVisit(5, 6.10, 17942);
	v.addFuelingStationVisit(9.8, 12.24, 18234);
	v.addFuelingStationVisit(8.3, 10.11, 18483);
	MileageReport r = v.generateMileageReport();
	assertEquals(541, r.getMilesDriven());
	assertEquals(18.1, r.getFuelConsumed(), delta);
	assertEquals(29.88950, r.getMilesPerGallon(), delta);
	assertEquals(28.45, r.getTotalFuelCost(), delta);
}
```

這樣看起來好多了。您絕對不會知道撰寫測試時自己會發現什麼。不過有件事情可以確定—當您在測試中和程式碼中指定同一件事情兩次時，您一定會發現這比您只撰寫程式碼有更多的錯誤。

現在我們可以試著新增程式碼，使先前的測試通過。

Vehicle.java Listing 16

```
public MileageReport generateMileageReport()
{
	MileageReport r =  new MileageReport();
	if (itsVisits.size() == 0)
	{
		r.setMilesPerGallon(0);
		r.setMilesDriven(0);
		r.setTotalFuelCost(0);
		r.setFuelConsumed(0);
	}
	else if (itsVisits.size() == 1)
	{
		FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(0);
		r.setMilesPerGallon(0);
		r.setMilesDriven(0);
		r.setTotalFuelCost(v.getCost());
		r.setFuelConsumed(v.getFuel());
	}
	else
	{
		int firstOdometerReading = 0;
		int lastOdometerReading = 0;
		double totalCost = 0;
		double fuelConsumption = 0;

		for (int i=0; i<itsVisits.size(); i++)
		{
			FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(i);
			if (i==0)
			{
				firstOdometerReading = v.getMileage();
				fuelConsumption -= v.getFuel();
			}
			if (i==itsVisits.size()-1) lastOdometerReading = v.getMileage();
			totalCost += v.getCost();
			fuelConsumption += v.getFuel();
		}

		int distance = lastOdometerReading - firstOdometerReading;
		r.setMilesPerGallon(distance/fuelConsumption);
		r.setMilesDriven(distance);
```

```
            r.setTotalFuelCost(totalCost);
            r.setFuelConsumed(fuelConsumption);
        }
        return r;
    }
```

此程式碼中的所有特殊案例使它變得不雅觀。我們需要重構特殊案例。事實上，第三個案例就足以概括一切。我們應該可以刪除其他兩個案例。

如果我們這麼做，testSingleVisitMileageReport 測試案例就會失敗。失敗是因爲加一次油的案例只包括第一次加油時所加的油。如前所述，如果只加一次油，則耗油量必須是零。因此，我們可以修正測試案例和程式碼。

Vehicle.java Listing 17

```
public MileageReport generateMileageReport()
{
    MileageReport r =  new MileageReport();

    int firstOdometerReading = 0;
    int lastOdometerReading = 0;
    double totalCost = 0;
    double fuelConsumption = 0;

    for (int i=0; i<itsVisits.size(); i++)
    {
        FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(i);
        if (i==0)
        {
            firstOdometerReading = v.getMileage();
            fuelConsumption -= v.getFuel();
        }
        if (i==itsVisits.size()-1) lastOdometerReading = v.getMileage();
        totalCost += v.getCost();
        fuelConsumption += v.getFuel();
    }

    int distance = lastOdometerReading - firstOdometerReading;
    r.setMilesPerGallon(distance/fuelConsumption);
    r.setMilesDriven(distance);
    r.setTotalFuelCost(totalCost);
    r.setFuelConsumed(fuelConsumption);

    return r;
}
```

此函數很長。我們需要將它縮短並稍微整理一下。我們先從移動一點點程式碼開始，使它們可以移到個別函數中。

Vehicle.java Listing 18

```
public MileageReport generateMileageReport()
{
    MileageReport r =  new MileageReport();

    int distance = 0;
    double totalCost = 0;
    double fuelConsumption = 0;
    double firstFuel = 0;
    double mpg = 0;

    if (itsVisits.size() > 0)
```

```
        {
            FuelingStationVisit firstVisit =
                (FuelingStationVisit)itsVisits.get(0);
            FuelingStationVisit lastVisit =
                (FuelingStationVisit)itsVisits.get(itsVisits.size()-1);
            int firstOdometerReading = firstVisit.getMileage();
            int lastOdometerReading = lastVisit.getMileage();
            distance = lastOdometerReading-firstOdometerReading;
            firstFuel = firstVisit.getFuel();

            for (int i=0; i<itsVisits.size(); i++)
            {
                FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(i);
                totalCost += v.getCost();
                fuelConsumption += v.getFuel();
            }

            fuelConsumption -= firstFuel;
            if (fuelConsumption > 0)
                mpg = distance/fuelConsumption;
        }

        r.setMilesPerGallon(mpg);
        r.setMilesDriven(distance);
        r.setTotalFuelCost(totalCost);
        r.setFuelConsumed(fuelConsumption);

        return r;
    }
```

Listing 18 是一個中間步驟。實際上它採取四個或五個更小的步驟才走到這一步。在每一個較小的步驟中，我們可以執行測試，以確定沒有什麼遭到破壞。那些重構作業的目標是要以某種方式使得程式碼更容易分割，但我們不確定該怎麼做。因此，這些初步重構作業幾乎是隨機作業。它們沒有花費太多時間，且測試已保證沒有什麼遭到破壞。

走到這一步且測試還在執行，這時我們會找到改進的方法。一開始，我們先將迴圈一分爲二[2]

Vehicle.java Listing 19

```
        if (itsVisits.size() > 0)
        {
            FuelingStationVisit firstVisit =
                (FuelingStationVisit)itsVisits.get(0);
            FuelingStationVisit lastVisit =
                (FuelingStationVisit)itsVisits.get(itsVisits.size()-1);
            int firstOdometerReading = firstVisit.getMileage();
            int lastOdometerReading = lastVisit.getMileage();
            distance = lastOdometerReading-firstOdometerReading;
            firstFuel = firstVisit.getFuel();

            for (int i=0; i<itsVisits.size(); i++)
            {
                FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(i);
                fuelConsumption += v.getFuel();
            }
            for (int i=0; i<itsVisits.size(); i++)
            {
                FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(i);
```

[2] 請參閱 www.refactoring.com 上的分割迴圈。

```
                totalCost += v.getCost();
            }


            fuelConsumption -= firstFuel;
            if (fuelConsumption > 0)
                mpg = distance/fuelConsumption;
        }
```

測試仍然在執行。接下來，我們將每一個迴圈擷取到它自己的專用方法。[3]

Vehicle.java Listing 20

```
public MileageReport generateMileageReport()
{
    MileageReport r =  new MileageReport();

    int distance = 0;
    double totalCost = 0;
    double fuelConsumption = 0;
    double firstFuel = 0;
    double mpg = 0;

    if (itsVisits.size() > 0)
    {
        FuelingStationVisit firstVisit =
            (FuelingStationVisit)itsVisits.get(0);
        FuelingStationVisit lastVisit =
            (FuelingStationVisit)itsVisits.get(itsVisits.size()-1);
        int firstOdometerReading = firstVisit.getMileage();
        int lastOdometerReading = lastVisit.getMileage();
        distance = lastOdometerReading-firstOdometerReading;
        firstFuel = firstVisit.getFuel();

        fuelConsumption = calculateFuelConsumption();
        totalCost = calculateTotalCost();

        fuelConsumption -= firstFuel;
        if (fuelConsumption > 0)
            mpg = distance/fuelConsumption;
    }

    r.setMilesPerGallon(mpg);
    r.setMilesDriven(distance);
    r.setTotalFuelCost(totalCost);
    r.setFuelConsumed(fuelConsumption);

    return r;
}

private double calculateTotalCost()
{
    double totalCost = 0;
    for (int i=0; i<itsVisits.size(); i++)
    {
        FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(i);
        totalCost += v.getCost();
    }
    return totalCost;
```

---

[3] 請參閱 www.refactoring.com 上的 EXTRACT 方法。

```
}

private double calculateFuelConsumption()
{
    double fuelConsumption = 0;;
    for (int i=0; i<itsVisits.size(); i++)
    {
        FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(i);
        fuelConsumption += v.getFuel();
    }
    return fuelConsumption;
}
```

測試仍然在執行。接下來，我們將耗油量的特殊案例移到 calculateFuelConsumption 方法中。

Vehicle.java Listing 21

```
public MileageReport generateMileageReport()
{
    ,,,

    if (itsVisits.size() > 0)
    {
        FuelingStationVisit firstVisit =
            (FuelingStationVisit)itsVisits.get(0);
        FuelingStationVisit lastVisit =
            (FuelingStationVisit)itsVisits.get(itsVisits.size()-1);
        int firstOdometerReading = firstVisit.getMileage();
        int lastOdometerReading = lastVisit.getMileage();
        distance = lastOdometerReading-firstOdometerReading;

        fuelConsumption = calculateFuelConsumption();
        totalCost = calculateTotalCost();

        if (fuelConsumption > 0)
            mpg = distance/fuelConsumption;
    }

    ...

    return r;
}

private double calculateTotalCost()
{
    double totalCost = 0;
    for (int i=0; i<itsVisits.size(); i++)
    {
        FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(i);
        totalCost += v.getCost();
    }
    return totalCost;
}

private double calculateFuelConsumption()
{
    double fuelConsumption = 0;
    if (itsVisits.size() > 0)
    {
        for (int i=1; i<itsVisits.size(); i++)
        {
            FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(i);
```

```
                fuelConsumption += v.getFuel();
            }
        }
        return fuelConsumption;
    }
```

測試仍然在執行。請注意，calculateFuelConsumption 現在可以採取權宜之計，開始總計耗油量與*第二次*加油。接下來，我們可以擷取函數來計算距離。

Vehicle.java Listing 22

```
public MileageReport generateMileageReport()
{
    MileageReport r =  new MileageReport();

    int distance = 0;
    double totalCost = 0;
    double fuelConsumption = 0;
    double firstFuel = 0;
    double mpg = 0;

    if (itsVisits.size() > 0)
    {
        distance = calculateDistance();
        fuelConsumption = calculateFuelConsumption();
        totalCost = calculateTotalCost();

        if (fuelConsumption > 0)
            mpg = distance/fuelConsumption;
    }

    r.setMilesPerGallon(mpg);
    r.setMilesDriven(distance);
    r.setTotalFuelCost(totalCost);
    r.setFuelConsumed(fuelConsumption);

    return r;
}

private int calculateDistance()
{
    int distance = 0;
    if (itsVisits.size() > 0)
    {
        FuelingStationVisit firstVisit =
            (FuelingStationVisit)itsVisits.get(0);
        FuelingStationVisit lastVisit =
            (FuelingStationVisit)itsVisits.get(itsVisits.size()-1);
        int firstOdometerReading = firstVisit.getMileage();
        int lastOdometerReading = lastVisit.getMileage();
        distance = lastOdometerReading-firstOdometerReading;
    }
    return distance;
}
```

測試仍然在執行。現在我們可以在 main 函數中移除條件式，然後清理一些零星物件。

Vehicle.java Listing 23

```
public MileageReport generateMileageReport()
{
        int distance = calculateDistance();
        double fuelConsumption = calculateFuelConsumption();
        double totalCost = calculateTotalCost();
        double mpg = 0;

        if (fuelConsumption > 0)
                mpg = distance/fuelConsumption;

        MileageReport r =  new MileageReport();
        r.setMilesPerGallon(mpg);
        r.setMilesDriven(distance);
        r.setTotalFuelCost(totalCost);
        r.setFuelConsumed(fuelConsumption);

        return r;
}

private int calculateDistance()
{
        int distance = 0;
        if (itsVisits.size() > 1)
        {
                FuelingStationVisit firstVisit =
                        (FuelingStationVisit)itsVisits.get(0);
                FuelingStationVisit lastVisit =
                        (FuelingStationVisit)itsVisits.get(itsVisits.size()-1);
                int firstOdometerReading = firstVisit.getMileage();
                int lastOdometerReading = lastVisit.getMileage();
                distance = lastOdometerReading-firstOdometerReading;
        }
        return distance;
}

private double calculateTotalCost()
{
        double totalCost = 0;
        for (int i=0; i<itsVisits.size(); i++)
        {
                FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(i);
                totalCost += v.getCost();
        }
        return totalCost;
}

private double calculateFuelConsumption()
{
        double fuelConsumption = 0;
        if (itsVisits.size() > 1)
        {
                for (int i=1; i<itsVisits.size(); i++)
                {
                        FuelingStationVisit v = (FuelingStationVisit)itsVisits.get(i);
                        fuelConsumption += v.getFuel();
                }
        }
        return fuelConsumption;
}
```

測試仍然在執行。

這是好現象。每一個函數自行包含，且與其他函數完全隔離。 main 函數小，容易瞭解。

您可能認爲這使得程式更爲複雜。雖然它的確增加了函數計數和行計數，但它也完美地分割了程式。每一個函數變得容易瞭解。

請注意，Listing 16 的案例分析已經傳回，但現在它與特定的計算函數相關聯。這比 Listing 17 好太多了，要移除 Listing 17 的案例分析必須碰碰運氣。

有人會抱怨此程式碼速度太慢。沒錯，但我們並不需要速度。當速度變成需求，而現行執行無法滿足該需求時，我們可以想點辦法。屆時，我們將對於 Listing 23 所顯示的明確性和分隔感到滿意。

結論

雖然本白皮書已在先測試設計的呈現之下示範重構的技術，它真正的目的是要傳達一種程式設計的 *態度*。程式在運作時尚未完成。固然，要讓它運作非常容易。當程式運作時儘量簡潔有力，這樣才算完成。

本白皮書主張，要達到所要的結果的理想方式是：

1. 撰寫測試案例來設計程式。在撰寫每一個測試案例之後，請撰寫讓該測試案例通過的程式碼。累計所有測試，並方便它們重複執行。
2. 程式的一部分可以運作之後，重構該部分，直到它變簡潔爲止。重構的作法是對程式碼做一連串細微的變更，並在每一次變更之後執行測試。這讓您有信心自己所做的變更不會破壞任何東西，並勇於繼續嘗試變更，直到程式碼夠簡潔爲止。

參照

[1] *Refactoring*, Martin Fowler, Addison Wesley, 1999

[2] *eXtreme Programming eXplained*, Kent Beck, Addison Wesley, 2000

Rational
the software development company

兩個總公司：

Rational Software
18880 Homestead Road
Cupertino, CA 95014
電話：(408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
電話：(781) 676-2400

免付費專線：(800) 728-1212
電子郵件：info@rational.com
網址：www.rational.com
國際辦事處：www.rational.com/worldwide