搞笑談軟工

敏捷開發，設計模式，精實開發，Scrum，軟體設計，軟體架構

2011 年 12 月 28 日 星期三

亂談軟體設計（3）：Single-Responsibility Principle

今天與後續的幾集會介紹 Robert C. Martin 在 Agile Software Development 這本書 Section 2 Agile Design 所提到的幾個 Teddy 覺個很有用的 design principles。今天先講第 95-98 頁的：

SRP: Single-Responsibility Principle
A class should have only one reason to change.

『A class should have only one reason to change』，這是什麼東東？？一開始看到這些 principles 的解釋真的是會被氣死，覺的好像看到武俠小說裡面的武功口訣一樣，什麼『天之道，損有餘而補不足，是故虛勝實，不足勝有餘...』，讀完這些抽象的口訣之後還是學不會。不過別擔心，Martin 在書中其實解釋得很清楚，想學好設計的鄉民們 Teddy 是很推薦去看一下這本書，尤其是 Section 2 的內容。

Martin 的這本書是 2003 年出版的，但是在這之前 Martin 有把 Section 2 中所提到的五個設計原則的 pdf 檔放到網路上（不知道現在是否還找的到），所以 Teddy 是因為先看過這些設計原則覺的寫得很好，後來這本書出版後 Teddy 才買回家『收藏』。言歸正傳，不想看書的鄉民們就看 Teddy 的懶人版本說明吧。

還記得 Teddy 在『亂談軟體設計（1）』所提到的 cohesion（內聚）與 coupling（耦合）嗎，SRP 主要的目的就要是增強一個類別（class）或是介面（interface）的內聚力。如何增強介面內聚力？~~用二秒膠~~ 藉由將介面中不相關的責任（responsibility）移到另外的介面中，也就是說不要讓一個介面『耦合（參雜）』一個以上的責任。

如果鄉民們有學過物件導向分析與設計（OOAD），可能也許應該還記得，OOAD（或是說軟體設計）從頭到尾都在講一件事：Responsibility。設計軟體系統的時候，要有那些類別或介面，而這些類別或介面要包含那些方法（methods）與資料（data members or attributes）。類別與類別之間要有什麼關係，是 associations，aggregation 或是 composition 的關係...等等等。在思考這些問題的時候，一個很基本的概念，就是 responsibility assignment（責任分配）。

Responsibility assignment 看似容易其實是很難的（因為這只是一個抽象的概念），如果分派了不合適的責任給不相關的類別，或是類別之間的互動關係搞錯了，那麼系統就會變得不易了解當然也很難擴充與維護。所以 Teddy 可以大膽的假設一下，這些所謂的『設計原則（design principles）』其實都是在教鄉民們如何做好責任分配。

Martin 在第 96 頁提到：

If a class has more than one responsibility, then the responsibilities become coupled. Changes to one responsibility may impair or inhibit the ability of the class to meet the others. This kind of coupling leads to fragile designs that break in unexpected ways when changed.

A better design is to separate the two responsibilities into two completely different classes.

如果鄉民們有看懂上面這一段，SRP 講來講去還是又回到 Teddy 在『亂談軟體設計（2）』中提到的一個觀念：

開發軟體最怕的不是『寫程式』或是『不會寫程式（寫不出程式）』，最怕的是『改程式（改那些原本已經寫好測試過可以動的程式）』。

假設一個類別或是介面同時存在兩種責任（稱之為 A，B），那麼很有可能當鄉民們想修改 A 的時候，而卻影響了 B。或是說，因為 B 的關係，卡住 A 讓 A 變得不好修改。Martin 在書中舉了兩個例子，Teddy 講另外一個親身體驗的例子。

當年 Teddy 還在唸書的時候，有一年 Teddy 當研究所 OOAD 課程的助教，幫忙 review 學生的設計。有一次 Teddy 看到某位學生設計了一個用來處理程式設定檔的類別，在此稱之為 ProfileManager，這些設定資料最後以 XML 格式存在檔案中。ProfileManager 就包含了至少三種以上的責任：

- 有一堆 getter/setter 用來讓 client（caller）把讀取與設定資料（這些資料最後會以圖形的方式顯示在畫面上）。
- 處理 XML 檔案。將 ProfileManager 的資料存成 XML 格式並且將 XML 檔案轉成 ProfileManager 物件。
- 列印。將 ProfileManager 所代表的圖形列印出來。

這樣的設計是非常常見的設計，因為在物件導向程式設計會教大家把『動作（operations）』與『資料（data）』集中起來，然後『封裝』在一個類別中。所以，對這個學生來說，他就很自然的把上面三種 responsibilities『封裝』在 ProfileManager 類別之中。因為從該學生的角度來看，他只看到『一種 responsibility』，而不是三種 。

Responsibility assignment 和 separation of concerns 其實是很接近的觀念。如果把『處理 XML』與『列印』抽離出來（separation of concerns），那麼 ProfileManager 的確是同時涵蓋了三種不同的責任。但是，如果真的把不同的責任都逐一分開放到不同的類別中，可能會造成系統中有太多很小的別，而這些被切割出來的類別（例如處理列印或是 persistence 類別）很可能並不會被重複使用（因為系統規模還很小），那麼這樣的設計就有 ~~過度包裝~~ over design 的嫌疑。

講到這邊鄉民們應該會有一個問題，也是最後，最重要的問題，那就是：

到底什麼情況應該把兩個以上的責任徹底分開？什麼情況放在一起是 OK 的？

課本第 97 頁有答案：

If the application changes in ways that affect the signature of the connection functions, then the design will smell of Rigidity... In that case the two responsibilities should be separated. If, on the other hand, the application is not changing in ways that cause the two responsibilities to change at different times, then there is no need to separate them.

友藏內心獨白：這樣也叫做懶人版本？太長了啦，看不懂。
張貼者： Teddy Chen 於 上午 10:24