延伸個人軟體程序支援單元測試流程資料收集與分析

劉建宏 林猷翔 陳淑玲
國立台北科技大學 國立台北科技大學 南台科技大學
資訊工程系 資訊工程系 管理與資訊系
cliu@ntut.edu.tw t5598027@ntut.edu.tw slchen@mail.stut.edu.tw

摘要

隨著單元測試(unit testing)工具的普及和對程式品質的注重，單元測試日益受到重視，大多數組織的軟體開發流程也都包含單元測試的階段。然而，對於單元測試工作成效的評估，包含其對開發成本和程式品質的影響，相關的探討目前仍不多。本論文延伸個人軟體程序(PSP)，針對單元測試流程，探討需要收集追蹤的流程資料，並提出一些度量(metric)，分析說明這些度量可能代表的涵義。開發人員可藉由這些度量了解其單元測試設計的能力，並可作為改善其單元測試流程的參考。

關鍵詞：單元測試、個人軟體程序、測試流程。

Abstract

As unit testing tools have become widespread and quality has become a critical concern for software development, unit testing has attracted more and more attentions. Nowadays most software organizations have included unit testing phase in their software development process. However, there exist a few discussions in the literature addressing the assessment for unit testing, such as its influences to overall development cost and software quality. This paper extends Personal Software Process (PSP) and mainly focuses on the unit testing process. The process data to be collected and kept track of are discussed. Several metrics are proposed and their indications are described. These metrics not only can help developers to understand their ability in designing unit tests, but also can be useful references for developers to improve their unit testing processes.

Keywords: Unit testing, PSP, testing process

1. 前言

隨著顧客對程式品質的要求日漸提高，單元測試也逐漸受到軟體開發單位的重視，加上近幾年單元測試工具的普及，如 JUnit[1]和 CPPUnit[2]等，使得開發人員不僅需要撰寫產品碼(production code)，亦須撰寫測試碼(test code)以便能進行單元測試。有些軟體工程專家甚至提倡測試驅動開發(Test Driven Development, TDD)[3]，亦即先撰寫測試碼，再進行產品碼的開發，同時建議產品碼和測試碼的比例為 1：1，顯見測試碼開發的重要性，以及單元測試對軟體開發的影響。

隨著單元測試的重要性增加，如何改善開發人員其單元測試流程及提昇其單元測試能力，也逐漸引起注意。而個人軟體流程的改善，要以軟體工程學院(SEI)的 Watts S. Humphrey 所提出的個人軟體程序(Personal Software Process, PSP)[4~6]最廣受矚目。個人軟體程序主要是透過計畫、追蹤、測量及分析等技術與方法來改善個人軟體的開發流程，並提昇軟體的品質與生產力(productivity)，以及增加軟體開發的可預測性(predictability)。

為了解個人開發流程的現況及協助開發流程的改善，個人軟體程序要求開發人員收集其流程相關資料，包含程式大小、各流程階段的時間與缺陷等資料。然而，也許是因為單元測試工具近幾年才被廣泛使用，所以個人軟體程序僅考慮產品碼相關資料的收集，並未考慮測試碼的相關資料，如測試碼大小(size)、測試碼撰寫時間等。缺乏足夠的單元測試流程資料，使得我們無法得知開發人員其單元測試開發的能力，並正確地評估單元測試的成效。也因此在流程檢驗(postmortem)的時候可能會忽略掉這個部份，無法對單元測試流程進行有效的改善。

舉例來說，透過對測試碼資料的收集，我們可以知道產品碼和測試碼程式大小的比例是否接近 TDD[3]開發模式所建議的 1：1，藉此可以估算開發人員其所開發的測試案例是否足夠。另外，透過收集測試碼程式大小與測試碼開發時間，亦可得知開發人員其測試碼的

生產力，或許可以藉此了解開發人員其測試案例設計的能力，以及對程式需求的了解程度。

有鑑於單元測試階段對提昇軟體品質的重要性，本論文延伸個人軟體程序，針對單元測試的部分建議收集一些流程資料，包括測試碼大小、測試案例數目、測試開發時間、測試執行時間、偵測到的缺陷(defect)、缺陷修正時間(fix time)等。並根據這些測試流程資料，我們提出了一些度量分析，可作為開發人員了解其單元測試能力和改善其開發與單元測試流程的參考。

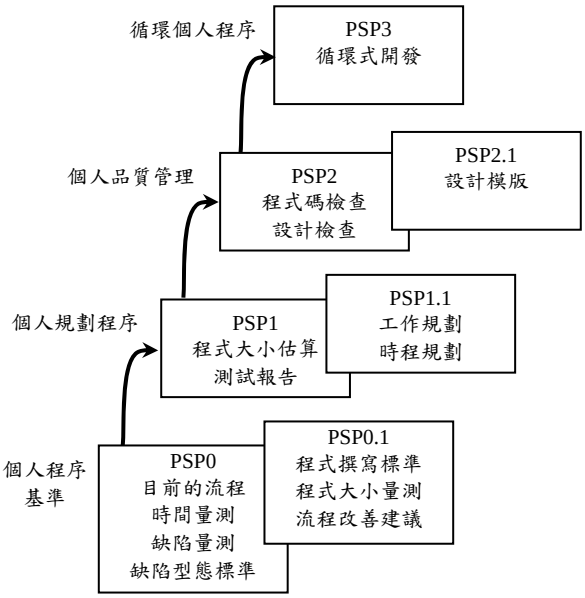
我們認為藉由單元測試相關資料的收集與分析，有助於開發人員改善其單元測試流程與測試設計能力，包含可讓開發人員了解其測試碼開發的生產力、其測試碼與產品碼的比例關係對程式品質的影響、測試碼大小與其開發時間的相關性、測試開發與測試執行花費時間的比例、以及提高單元測試流程的能見度(visibility)與預估能力等。同時，藉由這些度量的趨勢分析，可讓開發人員更容易發現其單元測試流程的問題，並加以改善，達到提高軟體品質的目的。

本論文的組織如下：第二節說明個人軟體程序相關的背景知識，第三節則是說明所需要收集的資料種類，第四節則針對度量分析進行解說和闡述其可能的涵義。最後一節為本論文的結語和未來的研究方向。

2.個人軟體程序簡介

個人軟體程序主要是透過測量和分析流程資料，幫助開發人員了解和改善個人軟體開發流程。在個人軟體程序中，Humphrey 將個人軟體開發改善程度訂定為四個階層(level)，分別為個人程序基準(The Baseline Personal Process, PSP0)、個人規劃程序(Personal Planning, PSP1)、個人品質管理(Personal Quality Management, PSP2)、以及循環個人程序(The Cyclic Personal Process, PSP3)。這四個階層共包含了七個定義嚴謹的流程(PSP0~PSP3)，如圖一所示[4]。每個流程都有明確的工作項目、應到達的標準(standard)、須填寫的表格(form)、需要收集的資料、工作腳本(script)和模板(template)。

個人軟體程序在 PSP0 和 PSP1 階層將軟體的開發分為六個流程階段(phase)：計畫(plan)、設計(design)、開發(code)、編譯(compile)、測試(test)和事後驗證(postmortem)。而 PSP2 和 PSP3 階層則分為八個流程階段，在設計階段後多了設計審查階段(design review)，以及在開發階段後多了程式碼審查階段(code review)，其劃分如表一。新增的設計審查與程式碼審查階段，有助於改善軟體開發的流程，提高軟體的品質與生產力。



圖一 個人軟體程序不同階段的演進

表一 PSP 流程階段說明表

流程階段	流程相關內容	PSP0 PSP1	PSP2 PSP3
計畫	訂定開發計劃和時程、分析需求	●	●
設計	設計架構和元件	●	●
設計審查	審核架構和元件		●
開發	開發產品程式碼	●	●
開發審查	審核產品碼		●
編譯	編譯程式碼	●	●
測試	測試和除錯	●	●
事後驗證	檢討軟體開發流程	●	●

個人軟體程序在實行上需要量測與收集許多的流程相關資料，例如程式碼的大小、開發時間、缺陷和流程階段等資訊，並藉由分析這些資料來輔助開發人員進行開發流程的改善。然而個人軟體程序目前僅注重產品碼相關資料的收集，例如產品碼大小、產品碼開發時間、產品碼類別(class)數目、產品碼缺陷等，並未收集測試碼的詳細資料，例如測試碼大小、測試碼撰寫時間、測試案例數目等，因此無法提供足夠的單元測試流程資料，作為評估與分析其單元測試工作成效的參考。

為協助改善單元測試流程，我們認為有需要延伸個人軟體程序，增加對單元測試相關資料的收集，以提高單元測試階段的能見度，作為開發人員改善單元測試能力的參考。在此需要說明的是，PSP 並未嚴格規定測試案例的設計應該是哪一個流程階段的工作，雖然一般測試案例的設計可以在計畫階段進行。此外，PSP 亦未定義測試碼的撰寫是屬於計畫或測試階段的工作，這有可能是因為早期沒有類似 JUnit 的測試工具，所以測試碼相關的資料較難獲得。但現在單元測試碼的撰寫已成為軟體開發不可忽略的重要步驟。因此，有需要將測試碼相關資料納入考慮，以獲得個人軟體流程詳細的資訊。在本論文中，我們將開發測試碼定義為測試階段的工作，這樣不僅符合開發人員的習慣，也方便測試碼和其他相關資料的收集，以及對測試流程資料的分析和比較。

3.單元測試流程資料收集

個人軟體程序需要收集相當多的流程資料，表二列出 PSPBOK(The Personal Software Process Body of Knowledge)[7]建議收集的一些流程資料。這些必須收集的資料大略可以分為三類：(1)程式碼資訊，例如新增、修改和刪除之程式碼的大小；(2)時間資訊，例如開發時間、中斷時間、流程階段花費時間等；和(3)缺陷資訊，例如缺陷型態、植入和排除階段、修正時間、缺陷描述等。

表二　PSP 資料收集表

資料名稱	描述
Time Data	所有時間相關的資料，包括開始時間、結束時間等
Interrupt time	開發過程中，工作中斷的時間
Delta time	實際的工作時間，即全部的工作時間減去 Interrupt time
Time in phase	PSP 流程階段所花的時間
Size measures	新增、修改、刪除的程式大小
Quality (defect data)	缺陷相關資料，如發現日期、缺陷型態、植入階段、排除階段、修正時間、缺陷描述等

為記錄單元測試流程的詳細資訊，除了表二的資料需要收集外，在測試階段我們額外收集一些跟測試碼有關的資訊，包含測試碼的大小、測試碼開發時間、以及測試案例的數目等，如表三所列。這些資訊將在下面的小節中詳細說明。

表三 測試階段增加資料收集表

	測試碼	時間	缺陷
收集資料名稱	測試碼大小	測試開發階段時間(TD)	該方法發現缺陷的數目
	類別的測試案例數目	測試執行階段時間(TE)	該類別發現缺陷的數目
	測試案例的執行結果	測試階段時間(TT)	缺陷修正時間總和(TF)
	專案的測試案例數目		

3.1 測試碼的數量資訊

在測試碼的部分，我們增加收集的資料如下：

- 測試碼大小：

  記錄測試碼的大小(LOC)。這項數據將直接反映開發人員的努力(effort)，並可與所收集的時間資料配合計算出測試碼的生產力，顯示開發人員其測試碼的開發能力。
- 類別的測試案例數目：

  記錄每個類別所開發的測試案例數目。這項數據有助於了解測試碼覆蓋(cover)了哪些類別，以及哪些類別的測試案例較多，可用來分析測試碼的開發是否足夠(adequate)。
- 測試案例的執行結果：

  記錄每個測試案例的執行結果，根據 JUint 的分類，測試結果可分為成功(success)、失敗(failure)和錯誤(error)三種。其中成功表示程式通過測試案例；失敗則表示程式執行結果與測試案例的預期輸出不同；錯誤則表示執行測試案列的過程中發生意外(exception)，造成測試中斷。這項數據有助於分析測試的成效。
- 專案的測試案例數目：

  統計整個專案的測試案例總數，此項數據可以了解開發人員設計測試案例的多寡。配合測試碼大小的資訊，可以分析每個測試案例的平均大小。透過測試碼生產力的資訊，可以推算開發人員其測試案例的生產力，顯示開發人員測試案例的開發能力。

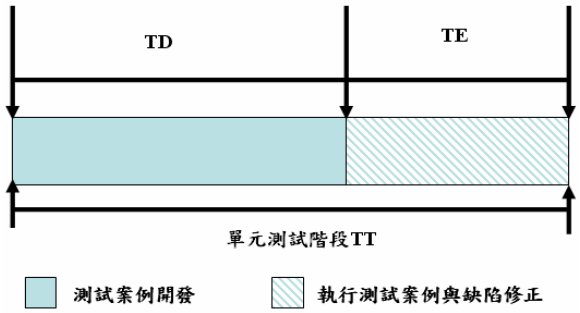
3.2 測試碼的時間資訊

在時間資訊部份，為有助於推導測試碼的生產力，我們將測試流程分成測試開發階段和測試執行階段。其中測試開發階段的工作為測試碼的開發；測試執行階段的工作則為測試案例的執行和缺陷的修正。所收集的資料說明如下：

- 測試開發階段時間(簡稱 TD)：

通常開發人員進行單元測試時可能(1)完成所有測試碼的開發之後才開始執行測試和修正缺陷，直至測試碼通過測試；或(2)開發一部分測試碼之後便開始執行測式，然後進行缺陷修正，待這部份測試碼通過測試後，再繼續開發下一部份測試碼，如此循環直到全部測試碼開發完成和通過測試。

考慮上述不同的測試工作習慣，測試開發階段時間的計算有兩種不同的方式。第一種為階段方式(stage)，亦即整個測試流程分成開發和執行前後兩個階段，如圖二所示。因此，完成測試碼開發所花費的時間即為測試開發階段時間(TD)。開始執行測試之後，直至測試通過為止，執行測試和為排除缺陷而修改產品碼或測試碼所花費的時間都歸屬於測試執行階段時間(TE)。



圖二 以階段方式進行單元測試

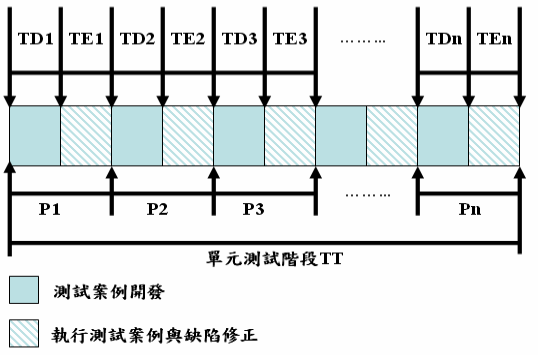
第二種為循環方式(cyclic)，亦即整個測試流程分成數個連續的子階段(即 P1 至 Pn)，每個子階段均包含測試開發和測試執行的工作，如圖三所示。每個子階段的測試開發和執行工作時間可依照第一種階段方式來計算。因此，全部測試開發階段時間 TD = TD1+TD2+…+TDn。

- 測試執行階段時間(簡稱 TE)：

測試執行階段時間的計算與測試開發階段類似。若用階段方式計算，測試執行階段時間為圖二中的 TE，包含了執行測試案例和修正缺陷的所有時間。若用循環方式計算，由圖三可知測試執行階段時間 TE = TE1+TE2+…+TEn。

- 測試階段時間(簡稱 TT)：

測試階段時間(TT)即為整個測試流程所花費的時間。若以階段方式計算，TT = TD+TE。若用循環方式計算，則 TT = TD+TE = TD1+TD2+…+TDn + TE1+TE2+…+TEn。



圖三 以循環方式進行單元測試

3.3 測試碼的缺陷資訊

在缺陷資訊部份，我們增加收集的資料如下：

- 該方法發現缺陷的數目：

記錄每個類別方法(method)所發現的缺陷數目。這項數據有助於了解每個方法的品質。若一個方法被發現有相當多的缺陷，意味著該方法需要進一步的分析以了解其缺陷較多之原因，此亦隱含該方法可能還有更多未被發現的缺陷[8]，需要撰寫更多的測試案例來進行檢驗。

- 該類別發現缺陷的數目：

記錄每個類別所發現的缺陷數目。這項資訊有助於了解類別的品質。開發人員可以針對缺陷數目異常的類別加以分析，檢驗其控制結構(structure)和狀態行為，了解造成缺陷數目較高的原因，並予以改善，以提高程式的品質。

- 缺陷修正時間總和：

統計所有缺陷的修正時間的總和，簡稱 TF。基本上 TF < TE，因為測試執行階段包含了執行測試案例的時間和修正缺陷的時間，所以修正缺陷的時間會小於測試執行階段的時間。透過 TF，我們可以了解開發人員排除缺陷所付出的成本，並可探討如何改變流程，以降低缺陷修正的時間。

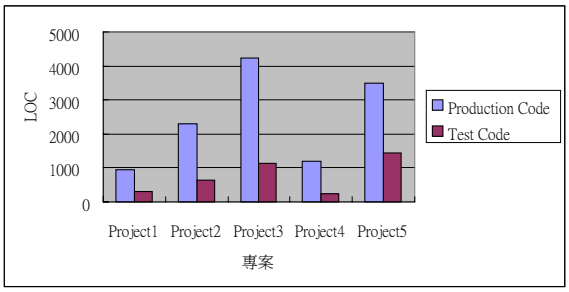
4.單元測試的度量與分析

針對測試流程收集到的資料，我們提出一些度量分析圖表，這些度量可以協助開發人員了解其單元測試的能力，幫助其找出測試工作的問題，以改善測試流程，達到提升單元測試能力和軟體品質的目標。

- **測試碼跟程式碼的大小比例圖**

圖四說明每個專案的測試碼和產品碼的

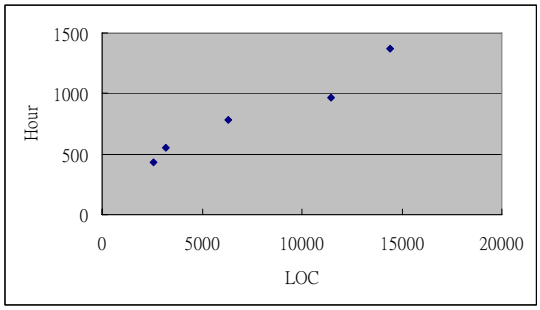
大小比例，可以讓開發人員了解測試碼和產品碼在專案上所佔的比重。圖四中 X 軸代表專案，Y 軸代表程式碼大小，深色長條顯示測試碼的大小，淺色長條則為產品碼的大小。透過圖四，開發人員可以知道其測試碼和產品碼的大小，亦可分析其測試碼和產品碼之間是否存在相依關係。另外，開發人員可以得知測試碼和產品碼比重是否符合組織的要求，亦可分析測試碼和產品碼的比例是否會影響專案的品質，例如：當測試碼和產品碼比例貼近 TDD 提出的 1:1 時，軟體品質是否有顯著的提高。



圖四 測試碼跟產品碼的大小比例圖

● **測試碼開發時間與大小關係圖**

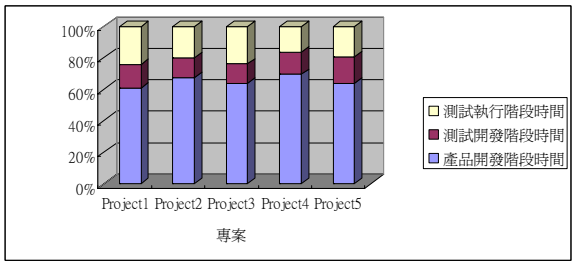
圖五說明每個專案其測試碼開發時間與測試碼大小之間的關係，其中 X 軸代表測試碼的大小，Y 軸為測試碼的開發時間，圖中的每一點則代表一個專案。透過圖五，開發人員可以了解其測試碼開發時間與測試碼大小之間是否存在有線性的關係，並可推算其線性迴歸係數。



圖五 測試碼開發時間與大小關係圖

● **開發和測試各階段時間比例圖**

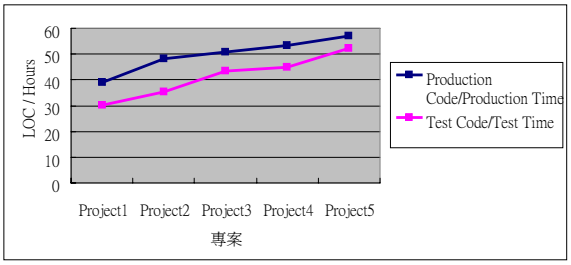
圖六說明產品開發階段、測試開發階段和測試執行階段的時間比例關係。圖六中 X 軸代表專案，Y 軸代表百分比，每一個長條顯示該專案不同階段所花費的時間比例，其中長條的下段部份為產品開發階段，中段部份則為測試開發階段，上段部分則為測試執行階段。透過圖六，開發人員可以了解開發和測試各個工作花費的時間比例是否正常，並可對時間比例過高的工作分析其異常的原因，以便進行流程改善。



圖六 開發和測試各個階段時間比例圖

● **測試碼與產品碼的生產力分析圖**

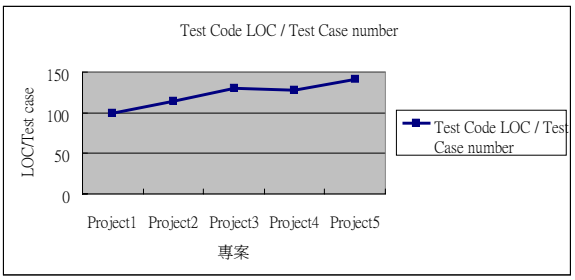
圖七說明開發人員其測試碼與產品碼的生產力(LOC/hour)。圖七中 X 軸代表專案，Y 軸代表程式碼的生產力，上方深色線條為產品碼的生產力，下方淺色線條則為測試碼的生產力。透過圖七，開發人員可以了解其開發和測試生產力的演進，以及流程改善對提高其生產力的助益。此外，圖七亦有助於分析測試碼與產品碼生產力之間是否存在有相依關係。



圖七 測試碼與產品碼的生產力分析圖

● **測試案例平均行數分析圖**

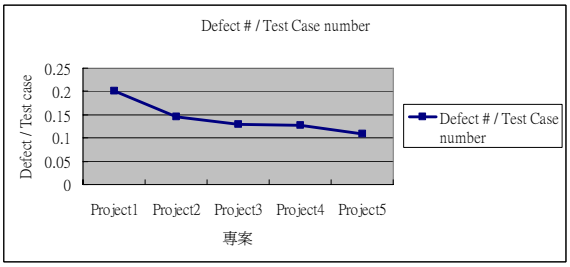
圖八為每個測試案例平均行數的分析圖，其中 X 軸代表專案，Y 軸代表測試案例的平均行數。透過圖八可以統計測試案例的平均行數大小，再配合測試碼的生產力，開發人員可以推算其撰寫一個測試案例所需的時間。若能預估單元測試所需的測試案例數目，則可預估測試開發階段所需的時間。



圖八 測試案例平均行數分析圖

● **測試案例之缺陷平均值分析圖**

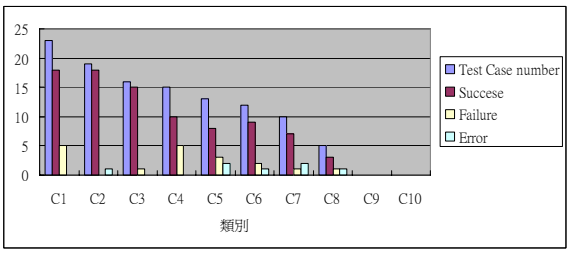
圖九描述缺陷數量與測試案例數量的關係，其中 X 軸代表專案，Y 軸則為測試案例的缺陷平均值。透過圖九，開發人員可以統計一個測試案例可以偵測到多少缺陷。假設程式的缺陷密度(defect density)變動不大(亦即開發人員其程式設計能力不變)，如果測試案例之缺陷平均值有顯著提升，則可能意味著開發人員其測試碼設計能力提高，藉此輔助我們了解開發人員的單元測試能力是否有所改善。



圖九　測試案例之缺陷平均值分析圖

● **類別測試結果統計圖**

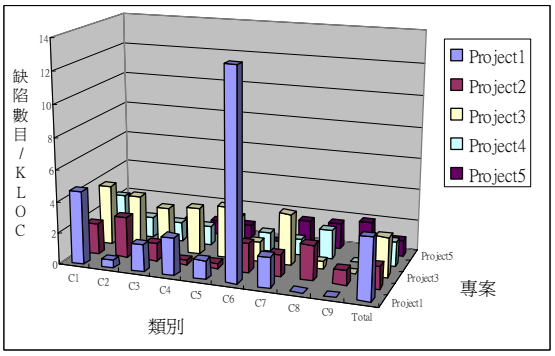
圖十說明專案中每個類別測試執行的結果，其中 X 軸代表類別，Y 軸代表測試案例的數量，每個類別的測試結果由四個長條顯示，由左至右依序為該類別的測試案例數量、測試成功的案例數量、測試失敗的案例數量、測試錯誤的案例數量。透過圖十，開發人員可以了解哪些類別有測試失敗或錯誤的情況，以及測試案例通過和未通過的比例，並分析有問題的類別，以提高測試案例通過的數量。



圖十　類別測試結果統計圖

● **缺陷密度分析圖**

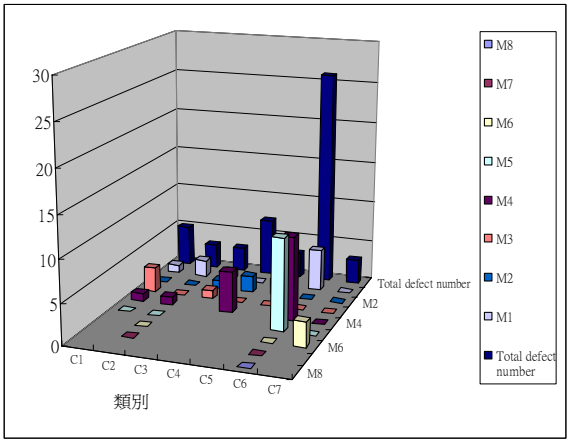
圖十一描述各個專案和其類別的缺陷密度，其中 X 軸代表專案，Y 軸代表類別，Z 軸則為缺陷密度。透過圖十一，開發人員可以了解每個專案中各個類別和整個專案的缺陷密度，得知各個類別和整個專案的品質。此外，透過圖十一，開發人員亦可以分析其專案缺陷密度的演進趨勢，了解其程式品質隨著流程改善而提升的程度。



圖十一　缺陷密度分析圖

● **專案缺陷分布圖**

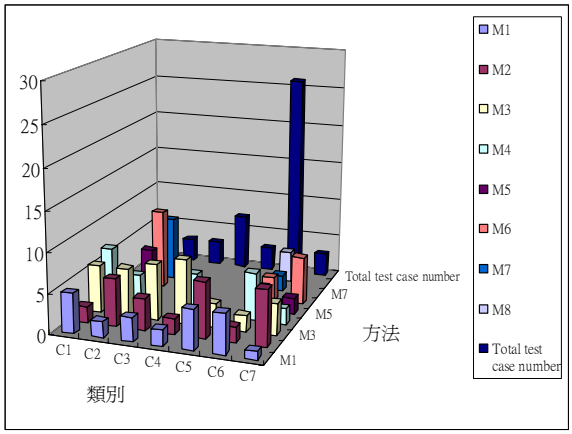
圖十二說明專案中缺陷的分佈情況，其中 X 軸代表方法，Y 軸代表類別，Z 軸則為缺陷數量。透過圖十二，開發人員可以了解每個類別內各個方法的缺陷數量和該類別缺陷數量的總和，並得知專案缺陷的分佈情形。開發人員可以檢驗缺陷數量異常的方法，如圖十二 C6 的 M5，以了解其缺陷數量較多的原因，並設計額外的測試案例來偵測該方法可能隱藏的缺陷[8]，以便排除這些缺陷，提高程式的品質。



圖十二　專案缺陷分布圖

● **測試案例覆蓋率統計圖**

圖十三說明專案中類別和方法的測試案例覆蓋率(coverage)。圖十三中 X 軸代表方法，Y 軸代表類別，Z 軸則為測試案例的數量。透過圖十三，開發人員可以得知每個類別和方法有多少對應的測試案例，了解每個類別和方法被測試案例覆蓋的程度。開發人員可以檢查是否每個方法均有對應的測試案例，並可針對測試案例覆蓋率較低的方法，增加新的測試案例，以提升單元測試的成效。



圖十三 測試案例覆蓋率統計圖

5.結語與未來研究方向

在本論文中，我們延伸個人軟體程序，增加對單元測試流程相關資料的收集，包含開發人員在單元測試階段所花費的時間、測試碼的大小和測試產生的缺陷等資訊。並且提出一些分析度量，例如：測試碼的生產力、產品碼與測試碼的比例、測試碼開發時間等，讓開發人員更清楚的了解其單元測試工作成效與趨勢，可作為開發人員改善其單元測試流程的參考，進而提升開發人員的測試設計能力，以及提高軟體的品質。

未來我們計畫開發一個流程資料自動收集與分析的工具，可以自動收集個人軟體程序所需的資料，以及單元測試相關的資訊，讓開發人員能減輕資料收集的負擔，以及提高資料的正確性。同時，藉由工具所提供的分析圖表，了解自己的開發與測試能力，作為其改善流程的參考。此外，我們亦計畫進行案例學習(case study)，收集更多的案例資料，以驗證所提出的度量對協助測試能力的改善是否有所助益，以及對軟體品質的提升的成效。同時，基於所收集的開發與測試流程資料，探討更多有助於流程改善的度量。

6.參考文獻

[1] JUnit. http://www.junit.org
[2] CPPUnit. http://www.cs.nmsu.edu/~jeffery/courses/371/cppunit/cppunit_cookbook.html
[3] Kent Beck, ***Test Driven Development: By Example***, Addison-Wesley, 2002.
[4] Watts S. Humphrey, ***A Discipline for Software Engineering***, Addison Wesley, 1995.
[5] Watts S. Humphrey, "Using A Defined and Measured Personal Software Process," ***IEEE Software*** 13(3), 1996, pp. 77-88.
[6] Personal Software Process (PSP). http://www.sei.cmu.edu/tsp/psp.html
[7] Marsha Pomeroy-Huff, Julia Mullaney, Robert Cannon, and Mark Sebern, "The Personal Software Process Body of Knowledge", ***CMU/SEI***, 2005.
[8] Boris Beizer, ***Software Testing Techniques***, Second Edition, Van Nostrand-Reinhold, 1990.