應用個人軟體程序支援程式設計的教學
Applying PSP to Support Teaching of Programming Course

劉建宏　　吳佳融
台北科技大學資訊工程系
{cliu, t5598022}@ntut.edu.tw

陳淑玲
南台科技大學管理與資訊系
slchen@mail.stut.edu.tw

摘要

在基礎的程式設計教學過程中，教師可以藉由了解學生其撰寫程式作業所花費的時間、程式的缺陷(defect)、程式大小(size)與缺陷密度(defect density)等資料，了解學生目前的學習狀況，以及可能遭遇的問題，作為改進程式設計教學的參考，而達到更好的教學成效。然而收集這些資料是件繁複的工作，為了減少學生收集資料的負擔以及提高資料的正確性，本論文亦提出了一套輔助工具，可以透過 Eclipse IDE 自動收集和彙整學生撰寫程式作業的流程相關資料，並提供這些資料的分析，讓教師了解個別學生以及全班的學習成效，以作為教學的參考。同時學生也可透過這些分析結果，掌握自己的學習情況與程式撰寫能力改善的趨勢，進而提高其學習的動機。

關鍵詞：個人軟體程序、程式設計教學、Eclipse

一、前言

由於軟體在許多領域的重要性日益增加，使得程式設計教學被許多科系視為必修的基礎課程。目前一般的程式設計課程，教師皆會安排若干課後作業，做為學生課後練習，其目的不外乎是希望能藉此提高學生程式撰寫的能力，並讓教師可以了解學生學習的情況。一般而言，透過分析繳交的作業程式功能是否達到需求規定，教師可以了解學生是否具有相當程度的程式撰寫能力。然而若能了解學生在程式練習過程中的一些資訊，例如：練習過程所出現的程式錯誤，以及程式撰寫所花費的時間等，教師將更能了解學生的努力程度、練習過程所遭遇的問題、以及學生程式撰寫的能力指標，並可作為教學改善的參考。

要瞭解與提升個人程式開發的能力，Watts Humphery 提出個人軟體程序(Personal Software Process, PSP)[1]的概念，亦即透過收集個人其程式開發流程的資料，包含程式大小、開發時間、和缺陷相關資料等，了解個人程式開發的能力，並藉由分析這些資料，找出流程可能存在的問題。同時，針對這些問題改變其流程，以改善其程式的品質與生產力(productivity)，達到提升個人軟體開發的能力。

透過PSP的觀念，我們可藉由收集學生程式作業練習過程中的一些資訊，例如程式的大小、程式開發時間、缺陷數量與缺陷描述等資訊，並提供這些資料的分析，以了解學生程式設計的能力，透過量化的數據指標，如生產力、缺陷出現的次數與密度等，讓教師了解個別或是全班學生其程式開發能力指標的變化與趨勢，提供教師作為教學改善的參考。同時，亦可讓學生了解其自身程式開發的能力與可改善的地方，作為其提升自我開發能力的參考。

收集學生個人程式練習過程的資訊有助於了解其開發能力，然而收集這些資料卻是件繁複的工作。學生必須自行計算每個程式的大小、不同開發階段(phase)的時間、每個缺陷植入與排除的階段、缺陷修正時間(fix time)與缺陷敘述等，並手動紀錄這些資訊。這對一個正在學習基礎程式設計的學生而言，是項額外的工作負擔。

為讓學生能專注於程式的撰寫，並減輕作業的負擔，以及提高所收集之資料的正確性，一個能結合至程式整合開發環境(Integrated Development Environment, IDE)，自動記錄程式開發過程相關資料的輔助工具有其需要。

有鑑於此，本論文提出一個程式設計教學的輔助工具PDAT(Programming Data gathering and Analysis Tool)，此工具採用Client-Server架構，在Client端透過Eclipse整合開發環境[2]，自動收集學生程式練習過程的相關資料。Eclipse是一個廣受歡迎的開放源碼(open source)整合開發環境，適用於Java與C/C++程式的開發。此外，Eclipse平台可以透過外掛(plugin)的方式，擴充其原有的功能。因此，我們可利用外掛軟體，在學生撰寫程式作業的同時，自動收集其作業撰寫的流程資料。這些資料可透過Internet傳送至Server端，以進行資料的分析與呈現。藉由流程資料的分析，學生可以清楚地了解自己每次作業所花費的時間、所開發之程式的大小、以及編譯和測試時所產生之缺陷的相關資訊，明瞭自己學習的情況。同時，教師除了可以知道個別學生的學習趨勢外，亦可知道全班學生的整體學習狀況，以及學生程式撰寫過程中可能遭遇的障礙，做為調整教學內容的參考。

本論文的架構如下：第二節敘述缺陷資料收集與測試案例的考量。第三節將描述如何在Eclipse IDE中收集所需要的各項程式練習過程資料和收集的方法。第四節內說明本系統如何應用所收集到的各項相關資料，提供量化的分析，以及分析結果可能呈現的意涵。第五節簡單介紹PDAT系統的架構以及客戶端資料收集功能的展示。第六節則為本論文的結語與未來研究方向。

二、缺陷收集與測試案例的考量

由於學生於程式練習中所產生的缺陷(defect)資訊有助於教師了解學生常犯的錯誤，以及可能遭遇到的程式設計問題，因此收集缺陷的相關資料有其必要。以程式作業而言，學生可能在編譯和測試其程式作業時偵測並收集這些缺陷資料。其中編譯階段所偵測到的缺陷資訊大都是跟程式語言的語法(syntax)有關，可從IDE的編譯器獲得。測試階段所偵測到的缺陷，則是與程式邏輯有關，可透過單元測試工具，例如：JUnit[3]或CppUnit[4]，來自動擷取。而使用單元測試工具表示必需要有測試案例(test case)，透過收集測試案例執行的結果，以獲得與程式邏輯有關的缺陷資訊。

測試案例的來源可由教師事先提供或由學生自行開發，各有其優缺點。若是由教師提供測試案例，可能的優點如下：

(a) 學生不需要自行撰寫測試案例，減少額外的學習負擔，讓學生專注在程式設計的學習上。
(b) 測試案例內容可以統一。因為測試案例事先由教師統一提供，因此所有的學生可以依據同樣的標準收集缺陷資訊。
(c) 測試案例可盡量詳盡完整。測試案例由教師提供，能夠提供更為完整的測試案例，而缺陷資料的收集也會更完整。

但由教師提供測試案例則可能會有以下的缺點：

(a) 程式練習內的物件名稱與方法名稱必須統一。因為測試案例是由教師統一提供，故必須對程式作業之物件與方法的命名上做規定。但此問題可由教師提供程式作業的框架(skeleton code)解決。
(b) 無法反應學生對作業功能需求的認知是否正確。若是藉由學生自行設計測試案例，其測試案例的設計是否完整地涵蓋到所有的需求，通常可以反映出學生對程式作業功能的需求是否了解。

另外，由學生自行撰寫測試案例可以帶來一些優點：

(a) 不需限制物件與方法的名稱。學生自行撰寫自己的程式碼的測試案例，其程式碼之架構與命名可以有較高的彈性，不需做任何限制。
(b) 可以反映出學生對作業功能需求的認

知程度，以及訓練學生測試案例設計的能力。學生若對作業程式功能需求了解，以及具備一定的測試設計能力，其測試案例一般而言會較為完整。然而，教師仍需有完整的測試案例，以作為評分的標準。

反之，若由學生自行撰寫測試案例，可能的缺點為：

(a) 學生所提供的測試案例可能不夠完整，無法偵測出程式邏輯方面所有的缺陷，這可能會造成對功能性缺陷(functional defect)資料的收集不夠完整。

(b) 增加學生額外負擔。測試案例的設計對多數程式設計的初學者仍是陌生的領域，需要額外的知識與訓練，對學生而言是一項額外的負擔。

考慮在基礎程式設計教學時，若教師能提供測試案例，可以收集較一致與完整的缺陷資料，同時可減輕學生的負擔，因此本論文建議採取教師提供測試案例的選項。然而，教師可視情況決定採用的方式，甚至混合兩種選項皆可。

三、程式作業練習過程資料的收集

由於人工收集程式作業練習資料是件繁複的工作，除了會增加學生額外的負擔外，所收集的資料容易錯誤，因此PDAT採用自動收集。透過Eclipse內的Workspace Plug-in的擴展點[5]，監聽Eclipse程式編輯器、編譯器和單元測試工具的事件，擷取得所需要的資料。

為擷取程式練習的過程的資料，我們將程式練習的過程分成兩個階段：開發階段與測試階段。其中開發階段是指包括了coding與compile的時間，而測試階段則是指在執行測試案例時的階段。另外，按照PSP的概念，PDAT所收集的資料可分為三大類，包括：程式大小(size)，時間(time)和缺陷(defect)。按照開發或測試階段分類，詳細資訊列於表一。

表一 收集的資料類型

Phase	**Development**	**Testing**
Size	Numbers of class Numbers of method Lines of code (LOC)	none
Time	Development time	Test time
Defect	Number of defects (syntax), Defect fix time, Defect description	Number of defects (functional), Defect fix time, Defect description

在表一中，開發階段的LOC數與方法數(Numbers of method)是針對該次練習內每個類別(class)進行收集，而若該次作業練習內有一個以上的類別時，類別的數量(Number of class)也會一併紀錄。而開發階段的時間(Development time)是指利用PDAT編輯器開始撰寫程式至學生點選單元測試工具開始執行測試的這段時間。同時，在這段時間內編譯器自動編譯時(auto build)所產生的錯誤(error)皆會視為語法錯誤(syntax defect)，其錯誤訊息會被自動紀錄以供後續分析之用。若在程式撰寫期間有中斷，例如：關閉Eclipse，該中斷時間將會從Development time中自動扣除*。

此外，開始執行單元測試後，程式練習的流程便進入測試階段，直至測試案例全部通過為止，這段時間，包含修改程式，均視為測試階段的時間(Test time)。相對於開發階段，測試階段的缺陷資料主要是由監聽和擷取Eclipse和單元測試外掛工具而得。另外，為了解不同缺陷所造成的影響，例如：邏輯錯誤的影響通常較語法錯誤嚴重，資料的收集除了缺陷的數目(Number of defects)和缺陷的描述(Defect description)外，PDAT亦收集每個缺陷的修正時間(fix time)。

為自動收集缺陷的修正時間，我們將每次編譯(或測試)的時間記錄下來，假設第i次和第$i+1$次編譯(或測試)的時間分別T_i和T_{i+1}，若缺陷的錯誤訊息出現在第i次

*我們假設學生在撰寫作業時，不會同時處理其他工作。且欲中斷程式撰寫時，會將程式存檔，並關閉 Eclipse。

編譯(或測試)結果的錯誤訊息視窗，亦即Eclipse的Problems view或JUnit的Failure視窗，但在第*i+1*次編譯或測試結束時，該缺陷未再出現於錯誤訊息視窗，此表示該缺陷已經被修正，這段時間(T_{i+1}–T_i)即為該缺陷的修正時間。若第*i+1*次編譯或測試時，有N個缺陷未再出現於錯誤訊息視窗，則個別缺陷的修正時間為(T_{i+1}–T_i)/N，亦即取連續兩次編譯(或測試)間隔時間的平均值做為這些缺陷其修正時間的近似值[6]。

上述缺陷修正時間的計算是假設在(T_{i+1}–T_i)這段時間內，至少有一個缺陷被排除。倘若這段時間沒有任何缺陷被排除(即 N=0)，則學生將持續修改程式，並進行編譯(或測試)，直到第 *j* 次編譯(或測試)時有至少一個以上的缺陷被排除為止，此時這些缺陷的修正時間將為(T_j–T_i)/N[†]。

四、程式練習過程資料的分析

在學生完成作業練習，並繳交程式練習過程的資料後，教師可對所收集來的資料進行分析的動作。我們將分析分為兩部份，一是針對學生個別的分析，一是針對全班學生的分析。

4.1 個人資料分析

針對學生個人的分析方面，我們著重於幾個可以幫助了解學生程式撰寫能力與調整教學的參考指標：(1)作業花費的時間；(2)語法熟悉度；(3)較難排除的缺陷；(4)常見的缺陷；(5)程式生產力；和(6)缺陷密度。以下將就各參考指標解說其產生方式與作用。

- **作業花費的時間**

每次程式練習時所花費的時間，包含開發階段時間、測試階段時間、以及兩項時間的總和。透過這些時間的資料，教師可以了解學生對此次作業努力的程度，評估該作業練習對個別學生的負擔是否適當。而透過生產力和缺陷密度的交叉分析，亦可了解學生對此程式練習是否有困難。一般而言，花費時間高、生產力低、缺陷密度高者，顯示其在練習該程式作業可能遭遇困難，故而需要較多的時間。

此外，開發階段時間和測試階段時間的比例也可以讓教師了解學生是否有進行足夠的程式規劃與設計。一般而言，測試階段時間的比例過高，且缺陷數目很多者，通常顯示其在撰寫程式時可能缺乏足夠的規劃與設計，以致於需要較多的測試時間來排除錯誤。

- **語法熟悉度**

在開發階段所產生的編譯錯誤，這類的缺陷大都與程式語言的語法有關，透過所紀錄的錯誤訊息，可以幫助教師了解個別學生對語法的熟悉度。例如：當JAVA程式使用未定義的變數或物件名稱時，編譯器會產生類似“xxx cannot be resolved”的錯誤訊息。若是錯誤訊息為“This method must return a result of type String”，則可知學生進行程式練習時，方法回傳值的type錯誤或忘記提供。

此外，若缺陷資料相當多，教師可對語法錯誤出現的次數進行排序，了解學生常犯的某種語法錯誤，這些語法錯誤通常都代表該學生對某個觀念一直不清楚。例如：若常出現“Null pointer”的錯誤，可能顯示學生對指標的觀念不了解，而常出現“Type mismatch” 的錯誤，則可能表示該學生對資料型態的意涵不甚了解。教師可由該次程式練習所要求的範疇，並藉由此次作業的缺陷敘述，了解個別學生其語法熟悉度，並做為加強學生哪方面語法觀念的參考。

- **較難排除的缺陷**

透過所收集的缺陷資料，我們可以知道學生對排除每個缺陷所花費的修正時間。缺陷修正時間的長短一般與產生該缺陷之背後問題的複雜度有關。通常編譯錯

[†]因為程式修改的關係，使得在(T_j–T_i)這段時間內可能會產生新的缺陷，導致無法得到較精確的缺陷修正時間。

誤產生的原因大都是語法不熟悉，或是一時打錯字(typo)，其缺陷修正的時間一般而言較短。因此，並非真正的問題所在。而真正困擾學生的缺陷通常是邏輯方面的錯誤，此類的錯誤會需要較多的時間分析程式，故需要花費更多時間才能排除。

這些較難排除的缺陷可以透過對缺陷修正時間做排序而得到，教師可以藉此了解學生此次作業可能遭遇的邏輯問題，並參考其他的量化指標，分析學生是否有些觀念可能學習不良。若多數學生有類似的邏輯錯誤，亦可能顯示此處教師講解較少或並未講解，使得學生需要花費較長的時間去自行解答，教師可以將此一資訊作為改進教學內容的參考。

- **常見的缺陷**

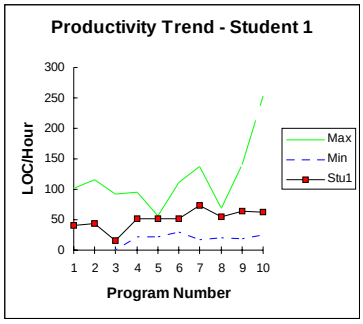
另外，透過型態類似的缺陷其出現次數的多寡，可以得知學生最常發生的缺陷為何種類型。此類缺陷的發生原因也是值得教師特別關注的地方，特別是常出現的邏輯錯誤，例如：指標未初始化、陣列索引超出邊界範圍、記憶體分配錯誤等，通常顯示學生某些觀念、語法或例外處理等一直有問題。這些問題是該學生迫切需要改善的地方，教師可以指導學生改善的方法，以避免這些問題一直重複的出現。

- **生產力**

在學生完成作業練習後，我們可以得知學生對此程式練習所花費的時間總數(Total_Hours)，以及整個程式作業的大小(Total_LOC)，如此便可得到該名學生的生產力(productivity)。計算公式如下：

$$\text{Productivity} = \text{LOC} / \text{Hours} \qquad (1)$$

透過計算生產力，可以知道該名學生撰寫程式的速度，可與缺陷密度做交叉分析，以評估學生程式設計的能力[7]。一般而言，生產力高，且缺陷密度低者顯示其開發能力較強。同時，可以透過生產力的趨勢走向圖，了解學生的程式撰寫能力是否有所進步。此外，學生亦可以透過其個人的生產力趨勢與全班平均的生產力趨勢的比較，了解自己與同學在程式開發能力上的差異。例如：圖一顯示某學生個人生產力的趨勢圖，從圖中可以看出該學生的生產力呈現上下起伏的情況，然而整體而言，其生產力有改善的趨勢。



圖一、學生個人的生產力紀錄

一般而言，生產力高比較好。但須注意的是，生產力的高低並不等同學習成果，因為程式寫的快，不代表程式的品質好。因此，生產力僅能作為一項參考的指標，通常需與缺陷密度做交叉分析。另外，教師可根據學生的生產力，估計下次程式作業學生可能所需要花費的時間，對決定下次作業的負擔亦具有參考價值。

- **缺陷密度**

缺陷密度為每千行程式中所產生的缺陷數目，通常可用來表示程式品質的好壞，其計算的公式如下：

$$\text{Density} = \text{NumberOfDefect}/\text{KLOC} \qquad (2)$$

學生程式作業的缺陷密度高低亦可以顯示其程式撰寫能力的好壞，通常程式撰寫能力佳者，其缺陷產生的數目應較少，故缺陷密度較低。反之，程式撰寫能力差者，其缺陷產生的數目應較多，所以缺陷密度較高。

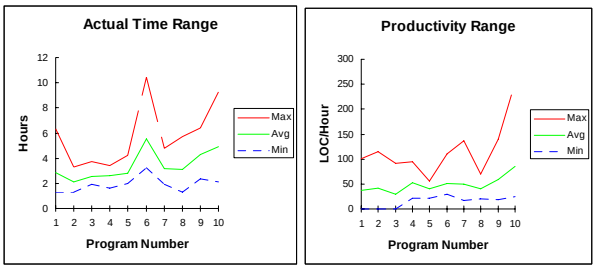
此外，如前面所言，缺陷密度可與生產力一同分析比對，以確切了解學生的學習成效。例如：對程式設計的初學者而言，其生產力和缺陷密度一般會呈現起伏的狀況，若起伏在一定的範圍內，表示學習情況穩定。若起伏甚大，則表示其學習過程可能忽好忽壞。另外，若學生生產力高，

且缺陷密度亦相對較高者，該學生可能未仔細思考程式的設計，便立即開始撰寫程式。如果學生生產力低，缺陷密度也高，則反映出該學生在學習上可能有很大的障礙，教師可能需要特別加予輔導。

4.2 全班資料整合分析

除了分析學生個人每次作業練習，以及其累次資料的趨勢走向外，全班學生的統計資訊對教師與學生個人亦可提供極佳的參考價值。藉由統計程式練習花費時間、生產力與缺陷密度的全班最大值、最小值以及平均值的趨勢走向，可讓教師了解全班學習的情況，學生也可以透過自己個人與全班整體表現的差異，得知自己與其他同學的差距，增強自己的學習動機。

圖二為作業花費時間與生產力的全班統計圖。從圖中可以了解作業所需的平均時數，教師可以判斷作業對學生的負擔是否適當。此外，最大值與最小值的差距，亦可顯示學生的學習狀況是否差異過大。而透過最大值或最小值與平均值的差異，也可能發現值得教師注意的地方，例如：最小生產力遠低於平均生產力，則可能代表某位學生的學習狀況有問題，值得教師關切。



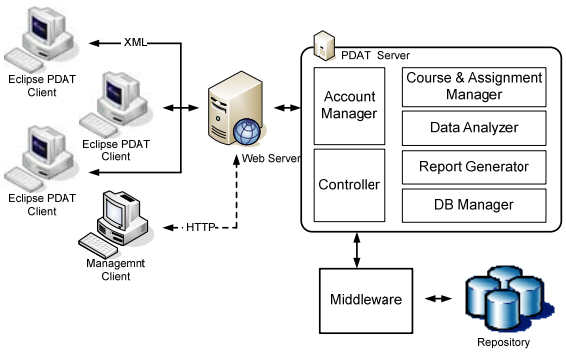
圖二、作業花費時間與生產力的全班統計圖

另外，將全班的缺陷資料整理，依照缺陷出現的次數或缺陷修正時間長短的排序，教師可以得知全班學生最常發生的錯誤，以及何種缺陷類型最令全班學生困擾，提供教師指導學生避免這些缺陷與改善其教學的參考。

必需說明的是，參考指標所隱含的訊息，透過教師仔細的分析，可提供相當多的資訊作為教學改善的參考，上述第四節所提供的分析說明，僅是這些參考指標的一些分析建議，教師可綜合這些指標和自己的教學經驗，提出其他更多有助於教學改善的分析。

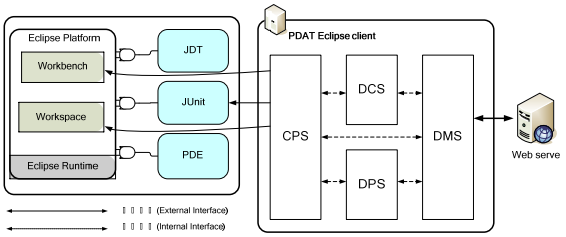
五、輔助工具架構與部分功能展示

為減輕學生收集作業練習過程資料的負擔，我們計畫開發一套輔助工具PDAT，可以透過Eclipse IDE自動收集和彙整學生撰寫程式作業的流程相關資料，以及產生參考指標的分析圖表，讓教師了解學生的學習成效，以作為教學的參考。PDAT主要採用client-server之架構，如圖三所示。其中伺服器端採Web平台，主要是提供資料彙整、分析和產生報表的功能。個別學生每次作業練習過程的資料將送到伺服器端彙整，以便得到全班的作業練習資料，這些資料透過統計分析可以產生如第四節所提的各種參考指標，並產生不同的分析圖表提供教師參考。此外，伺服器端亦提供課程、作業與帳號管理的功能。



圖三、PDAT系統架構圖

相對於伺服器端的Web平台，客戶端則是一個Eclipse開發平台的外掛程式，負責收集學生作業練習過程的相關資料。透過與Eclipse整合，當學生利用Eclipse平台撰寫程式作業時，第三節表一所列的流程資料會被自動收集，並以XML的格式儲存成檔案，當作業完成後，該檔案再傳送至伺服器端做後續資料的分析與統計。圖四顯示PDAT客戶端(PDAT-Client)的架構，以及其與Eclipse平台之間的關係。



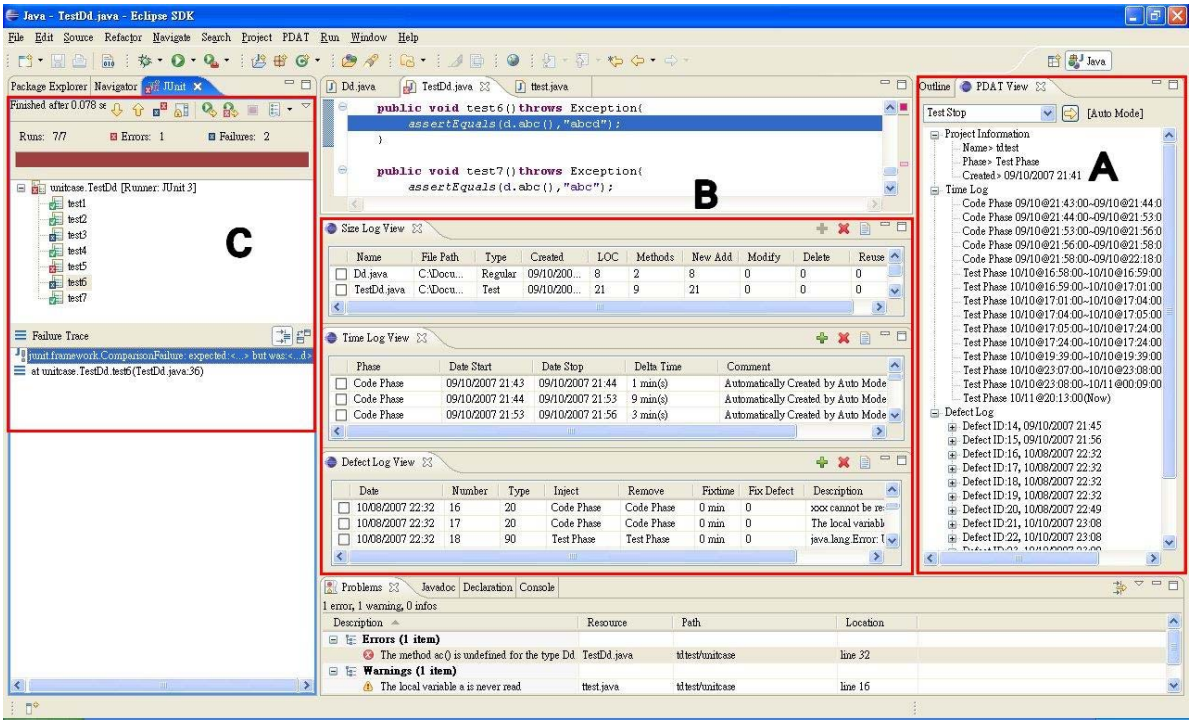
圖四、PDAT-Client架構

其中PDAT-Client主要子系統的功能簡略描述如下：

- Control Plug-in Subsystem(CPS)：為外掛程式控制子系統，透過Eclipse的擴充點，負責監聽Eclipse的事件，接受使用者命令，以及管理資料記錄視窗(log view)等。
- Data Collection Subsystem(DCS)：為資料收集子系統，負責記錄和計算學生程式練習過程的相關資訊，例如：程式大小、練習時間與缺陷相關資料等。
- Data Process Subsystem(DPS)：為資料處理子系統，可讓學生檢視所收集和記錄的資訊，並可對個人資料進行初步的分析，如花費的時間、生產力等。
- Data Management Subsystem(DMS)：為資料管理子系統，負責儲存和管理所收集和紀錄的程式練習過程資料，並可將這些資料傳送至伺服器端。

圖五是PDAT-Client端在Eclipse中執行的畫面。當學生開啟新專案，開始撰寫程式作業時，必須先指定此次流程資料收集的預設存放目錄，以便存放所收集的資料。然後學生便可進行程式的開發，系統會自動記錄相關的流程資料，包含開發與測試階段的時間、程式大小與缺陷相關資料等，並分別顯示在對應的紀錄視窗中(Log View)，如圖五B所示。

其中，程式大小視窗(Size Log View)提供程式物件的名稱、位置、建立的時間以及程式碼行數等相關資訊；時間記錄視窗(Time Log View)包含流程階段名稱、階段的起迄時間、階段花費時間及註解等資訊；缺陷視窗(Defect Log View)則包含缺陷的產生時間、型態、缺陷修正時間及缺陷描述等。這些缺陷包含了程式撰寫過程中所產生的編譯錯誤，以及單元測試未通過的測試案例(圖五C所示)。此外，系統尚提供一個主視窗(PDAT View)，讓學生可以檢視整個作業練習所記錄的資料摘要(project summary)，如圖五A所示，以掌握自己的現況。



圖五、PDAT-Client 的執行畫面

六、結語與未來方向

在這篇論文中，我們敘述如何透過程式作業練習過程的資料，協助教師了解學生學習的狀況，並作為程式設計教學改善的參考。我們建議一些作業練習過程中必須收集的資料，例如：程式大小、開發時間和缺陷等。我們並且提出了一些量化的參考指標，包含：(1)作業花費的時間；(2)語法熟悉度；(3)較難排除的缺陷；(4)常見的缺陷；(5)程式生產力；和(6)缺陷密度，以及說明教師如何透過這些參考指標來了解個別學生程式開發的能力、學生所遭遇到的問題、以及全班整體的學習成效。教師可藉由這些資訊作為指導學生改善其程式設計能力的依據，和調整課程內容的參考。

為減輕學生資料收集的負擔，我們提出一個程式開發流程資料收集的輔助工具，透過與Eclipse開發平台的結合，讓學生在撰寫程式的同時，工具可以自動記錄其作業練習過程的相關資訊，這些資料將作為後續參考指標推導和分析的基礎。此外，為彙整和分析這些資料，我們亦描述一個以Web為平台的系統架構，以便彙整全班的作業練習資料，進行統計分析，並產生趨勢分析圖表，提供教師或學生參考。

未來我們計畫繼續開發PDAT的Web平台，以自動整合學生作業練習的資料，提供教師關於學生程式設計能力的分析統計資訊，作為教學的參考。此外，在整套輔助工具完成後，我們亦計劃將把這套系統用於實際的基礎程式教學上，進行個案學習(case study)，以了解所提的參考指標對協助教學改善的成效，並希望能藉由更多的實務經驗，協助改進資料收集與分析的方法，以期能對程式教學改善有所助益。

誌謝

本論文部分由國科會研究計畫 NSC 96-2221-E-027-033 所補助，特此感謝。

七、參考文獻

[1] Watts S. Humphrey, A Discipline for Software Engineering, Addison Wesley, 1995.

[2] Eclipse. http://www.eclipse.org

[3] JUnit. http://www.junit.org/

[4] CppUnit. http://sourceforge.net/projects/cppunit

[5] Berthold Daum, *Professional Eclipse 3 for Java Developers*, Wrox, 2005.

[6] Chien-Hung Liu and Yu-Chun Huang, “A PSP Eclipse Plug-in Tool for Collecting Time and Defect Data”, In *Proceeding of 2006 Conference on Open Source*, Taiwan, Nov. 2006.

[7] Watt S. Humphrey, “Using A Defined and Measured Personal Software Process”, *IEEE Software*, May 1996, pp.77-88.